Data Discovery Analysis on Complex Time Series Data

Peter L. Severynen

Department of Computer Science and Engineering,

The University of Texas at Arlington

December 2022

Abstract

Complex time series are a ubiquitous form of data in the modern world. They have wide application across many different fields of scientific inquiry and business endeavor. Time series are used to understand and forecast weather patterns, voting patterns, computer network traffic, population health outcomes, demographic changes, the results of scientific experiments, and the performance of stocks and mutual funds. But time series can be difficult to analyze by conventional methods when the data is multivariate, incomplete, or in different formats. To address these issues, an investigation of several multivariate time series datasets was performed using the methods of automatic data discovery and derivative-based analysis. Interactive maps were constructed which displayed the results of the study. Conclusions were drawn and discussed, and an explanation was given of how this method can be applied to other multivariate time series datasets and real-world problems.

Keywords: time series, real estate, stocks, inflation, correlation, analysis, algorithm

Table of Contents

Data Discovery Analysis on Complex Time Series Data

Introduction

The Organisation for Economic Co-operation and Development (OECD) defines a time series as "a set of regular time-ordered observations of a quantitative characteristic of an individual or collective phenomenon taken at successive, in most cases equidistant, periods / points of time." (Time Series). A time series may be either univariate or multivariate. In a univariate time series, there is one variable of interest. In a multivariate time series, there are more than one variables of interest. The distinguishing characteristic about time series data is that, as Lazzeri observes, "time isn't just a metric, but a primary axis." (Lazzeri, 2020). Indeed, time is a central element of every time series dataset. In a time series, the observations have a temporal ordering and may have a temporal dependency on preceding observations. For example, if the daily high temperatures in a metropolitan area are recorded for a month, and this data is represented as a time series, then the observations have a temporal dependency on one another. If it was hot yesterday, then it is more likely that it will be hot today. When this temporal dependency aspect of time series is present in a dataset, it makes it possible to predict future values of the time series (with varying degrees of accuracy) based on previous values of the time series. This is called time series forecasting. Time series' predictive power is one of the factors that have led to their widespread use.

Part I: Applications of Time Series and Time Series Analysis

There are many situations in which it is necessary to analyze different time series data using data discovery, data mining, and machine learning (ML) techniques and methods. For example, weather patterns, climate change, voting patterns, computer network traffic, commuter traffic, healthcare data, demographics, scientific observations, and business/financial applications, etc. may all be represented as time series. In order to extract useful information from these time series and derive meaning from them, it is necessary to perform time series analysis on them.

Weather forecasting is the quintessential application of time series data. Weather stations around the world collect data on temperature, precipitation, humidity, air pressure, and other atmospheric parameters at regular intervals. This atmospheric data constitutes time series. These time series are aggregated and fed as input to supercomputers run by the National Oceanic and Atmospheric Administration (NOAA), a division of the U.S. Department of Commerce. By analyzing atmospheric time series and performing time series forecasting, accurate weather forecasts may be obtained. These forecasts are vital to the economic well-being of farmers, ranchers, builders, and others in weather-sensitive occupations and industries. In fact, according to NOAA,

> Each year, the United States averages some 10,000 thunderstorms, 5,000 floods, 1,300 tornadoes and 2 Atlantic hurricanes, as well as widespread droughts and wildfires. Weather, water and climate events, cause an average of approximately 650 deaths and $15 billion in damage per year and are responsible for some 90 percent of all presidentially-declared disasters. About one-third of the U.S. economy – some $3 trillion – is sensitive to weather and climate. National Weather Service (NWS) provides weather, hydrologic, and climate forecasts and warnings for the United States, its territories, adjacent waters and ocean areas, for the protection of life and property and the enhancement of the national economy. (NOAA, 2022)

The National Weather Service is a division of NOAA, and it provides timely and essential weather forecasts and other weather-related information to all Americans. For example, the National Weather Service accurately predicted the paths of Hurricane Ian and Hurricane Nicole, which devastated Florida in 2022. Because the National Weather Service issued a hurricane watch for the affected counties, many residents evacuated, and thousands of lives were saved as a result! This would not have been possible without modern time series analysis and forecasting algorithms.

The scientific consensus is that extreme weather events such as these are growing in frequency and severity due to climate change. Here too, time series analysis is helpful for understanding the origin and evolution of climate change. Stips et al. asserts that “the results of investigating the information flow between the major radiative forcing’s and the GMTA [global mean surface air temperature anomalies] time series clearly show that total Green House Gases (GHG), dominated in particular by CO_2 forcing, is the main driver of changing global surface air temperature” (Stips et al., 2015). Kremer et al. used a clustering technique to detect climate change in multivariate time series data from hydrology, meteorology, and oceanography (Kremer et al., 2010).

Time series analysis is also helpful in understanding voting patterns and how they change over time. For example, as set forth in the U.S. Constitution, the U.S. House of Representatives currently has 438 members who are elected from their respective congressional districts to serve two-year terms (U.S. Const. art. I, § 2.). The party affiliation of the winning candidate in successive elections forms a time series. By analyzing these time series for each congressional district, it can be determined whether a particular district leans Democratic or Republican. Political strategists use this information to target campaign advertising funds in the geographic areas where they believe that these funds will be most effective. This information also informs candidates’ voter mobilization efforts.

The role of time series analysis in understanding computer network traffic, congestion, and cyberattacks has been well-documented. Ntlangu and Baghai-Wadji trace the history of using time series to model network traffic and explain the theoretical foundations of this approach (Ntlangu & Baghai-Wadji, 2017). The number of packets traveling across a network node at predefined time intervals constitute a time series. By measuring this time series, it can be determined whether the network is congested at that node and will ultimately slow down. Moreover, as Shen observes, time series forecasting may be used to predict network traffic. (Shen et al., 2009). Time series are also an integral part of computer network security. An anomalous and sudden increase in traffic from a particular subnet or range of IP addresses may indicate a distributed denial of service (DDoS) attack on a particular server or Web site. Time series help network security professionals detect and prepare for such cyberattacks. Wu et al. used time series and fuzzy logic to improve the accuracy of "network threat frequency forecasting" (Wu et al., 2008). This work is important because it makes network connections faster and more secure.

Just as time series are used to understand and model the movement of traffic in the digital world, time series may also be used to understand and model the movement of traffic in the physical world. Time series data about traffic flow is frequently used by civil engineers when designing new transportation infrastructure and improving existing transportation infrastructure. This applies equally well in both urban and rural settings. Moreover, Bawaneh and Simon created an algorithm for detecting anomalies in city traffic, such as automobile accidents, using time series data (Bawaneh & Simon, 2019). Once civil engineers know where trouble spots frequently occur, they can recommend solutions to policymakers including widening roads, installing crash impact barrels, and adjusting traffic signage. This illustrates one of the many practical benefits of time series analysis.

Time series analysis is also an important tool in biomedical research and public health policy planning. As Ferenti observes, “Many biomedical data are available as time series, especially in the field of public health and epidemiology, where indicators are usually collected over time.” (Ferenti, 2017). For example, the number of cases of an infectious disease, the number of positive tests, and the number of deaths attributable to the disease over time are all important factors that may be represented as time series.

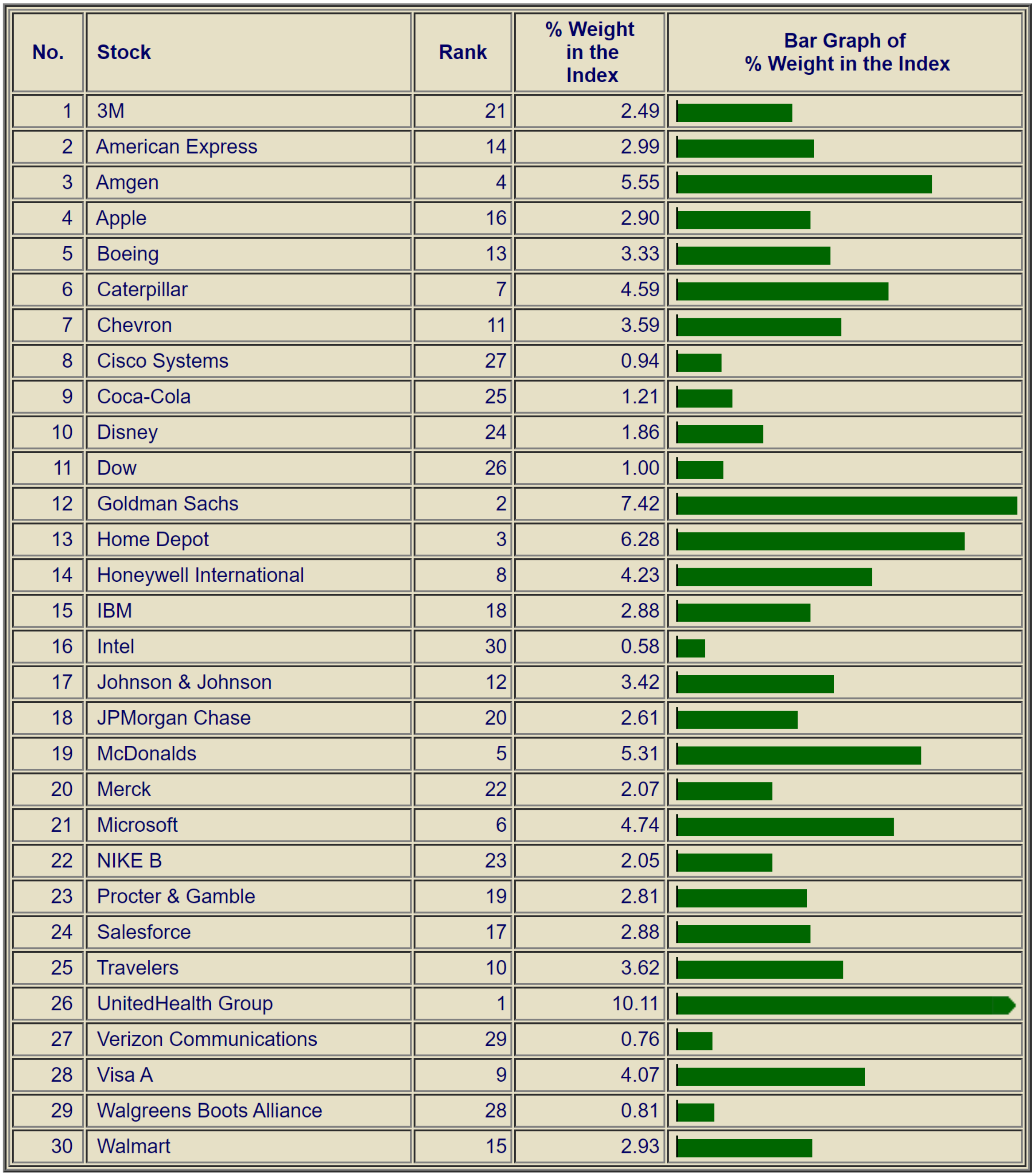
Specifically, state public health departments routinely publish datasets of COVID-19 cases, tests, and deaths. An example of such data for the state of California may be found at CHHS Open Data, a website of the California Department of Public Health (California Department of Public Health, 2022). Models and analyses based on these time series datasets inform policymakers when they are tasked with making decisions about masking, social distancing, and other public health measures that are designed to reduce transmission and prevent the healthcare system from becoming overburdened. By analyzing these time series datasets, public health experts and data scientists can track case numbers over time and assess whether COVID-19 cases are trending upward or downward as new variants evolve. Oladunni et al. modeled how COVID-19 affects different communities and demographic groups (Oladunni et al., 2021). This is important because it can potentially lead to increased health equity and targeted outreaches and educational campaigns to underserved communities (racial and ethnic minorities, rural areas, tribal lands, etc.). Ibrahim et al. describe how “data analytics, properly applied, hold great potential to target inequities and reduce disparities [in health care]” (Ibrahim, 2020). Timely, comprehensive data and accurate, high-quality analysis are prerequisites for sound and effective public health policy.

Time series analysis and forecasting is also applicable to a wide variety of economic and financial data both at an economy-wide level and at the level of individual companies and corporations. Furthermore, many kinds of economic data may be represented as time series. As Hayes observes, "[h]istorical stock prices, earnings, gross domestic product (GDP), or other sequences of financial or economic data can be analyzed as a time series." (Hayes, 2022). These are metrics that apply to the U.S. economy as a whole. On a smaller level, time series are also used to model various financial performance metrics for individual companies or groups of companies within a specific sector of the economy. For example, the quarterly sales volume of a publicly traded company, or the monthly production of Japanese steelmakers may be modeled as a time series. Unlike the macroeconomic factors stated above, these metrics apply to a particular company or economic sector. Time series analysis helps companies make projections about what their quarterly sales, revenue, and earnings are expected to be for the next year.

Part II: Data and Methods

To evaluate the merits and applicability of derivative-based analysis to a broad set of problems, three datasets were used for this study: the Zillow Single-Family Homes Time Series Metro & U.S. dataset, the Dow Jones Industrial Average (DJIA) dataset, and the Consumer Price Index (Inflation) dataset. The first dataset describes the monthly median sale price of single-family homes in the 911 largest U.S. metropolitan areas. The dataset covers the period from January 1996 to August 2021. This data was collected by the real estate company Zillow and made publicly available (Zillow, 2021). The second dataset consists of the opening and closing prices of the Dow Jones Industrial Average (DJIA). Ganti describes the DJIA as “a price-weighted index that tracks 30 large, publicly-owned [publicly-traded] companies trading on the New York Stock Exchange and the Nasdaq…[It] serve[s] as a proxy for the broader U.S. economy” (Ganti, 2022). The current composition of the index is shown in Figure 1.

Figure 1. Components of the Dow Jones Industrial Average Sorted by Stock Name



No.	Stock	Rank	% Weight in the Index	Bar Graph of % Weight in the Index
1	3M	21	2.49	
2	American Express	14	2.99	
3	Amgen	4	5.55	
4	Apple	16	2.90	
5	Boeing	13	3.33	
6	Caterpillar	7	4.59	
7	Chevron	11	3.59	
8	Cisco Systems	27	0.94	
9	Coca-Cola	25	1.21	
10	Disney	24	1.86	
11	Dow	26	1.00	
12	Goldman Sachs	2	7.42	
13	Home Depot	3	6.28	
14	Honeywell International	8	4.23	
15	IBM	18	2.88	
16	Intel	30	0.58	
17	Johnson & Johnson	12	3.42	
18	JPMorgan Chase	20	2.61	
19	McDonalds	5	5.31	
20	Merck	22	2.07	
21	Microsoft	6	4.74	
22	NIKE B	23	2.05	
23	Procter & Gamble	19	2.81	
24	Salesforce	17	2.88	
25	Travelers	10	3.62	
26	UnitedHealth Group	1	10.11	
27	Verizon Communications	29	0.76	
28	Visa A	9	4.07	
29	Walgreens Boots Alliance	28	0.81	
30	Walmart	15	2.93	

The 30 companies that comprise the Dow Jones Industrial Average and their current weights.[1]

The DJIA dataset covers the time period from December 31, 1999 to February 11, 2022. The DJIA dataset was obtained from Yahoo! Finance (Dow Jones Industrial Average (^DJI)). The third dataset lists the inflation rate as measured by the Consumer Price Index (CPI) for each month from January 1996 to December 2021; it was obtained from usinflationcalculator.com (Historical Inflation Rates: 1914-2022, 2022).

Why was Time Series Analysis Chosen as the Method of Inquiry for These Datasets?

For this study, time series analysis was chosen because of the nature of the datasets to be investigated. Lazzeri notes, "[T]ime series analysis is about identifying the intrinsic structure and extrapolating the hidden traits of your time series data in order to get helpful information from it." (Lazzeri, 2020) For the purposes of this study, each dataset was treated as separate and independent from the others, that is, there are no known dependencies of one dataset upon another. However, all the datasets may be influenced by broader economic trends, which are beyond the scope of this analysis.

Since the datasets are multivariate time series, this presents unique challenges and opportunities. Two of the challenging aspects of analyzing these datasets were that they exist in different formats and use different timescales for their observations. While all the datasets are stored as comma separated values (CSV) files, one of the datasets uses time as the horizontal axis, while the others use time as the vertical axis. Time is also measured in different increments by the disparate datasets. In the real estate and inflation datasets, the observations were recorded on a timescale of months, while in the DJIA dataset, the observations were recorded on a timescale of days. This renders the datasets incompatible for a comparative analysis at the closest level of granularity.

The multivariate nature of the datasets also afforded unique opportunities for analysis that would not have been possible otherwise. For example, as the real estate dataset included geospatial, temporal, and financial components, correlations could be drawn between the three. It was possible to track which regions appreciated faster than others because of the intersection of geography, time, and price.

Research Journey

When this research project began, the initial intent was to develop an online home buyer's assistant that could show properties and recommend offer prices. This entailed a large amount of research into home prices and the fundamentals of the real estate market. Over time this goal evolved significantly as additional data was incorporated into the data discovery and analysis process. A set of color-coded interactive maps were created, which showed the 120 largest U.S. metropolitan areas grouped by different parameters. Normalization of the data and computing the first and second derivatives made it possible to make meaningful comparisons within the dataset. Furthermore, as the researcher continued to explore the data, he found that some of his initial assumptions were validated by the analysis while others were not. For example, the median price of a single-family home doubled faster in California and Florida than it did in the Great Plains states (Map 1). However, most metropolitan areas in Texas and Louisiana did not double faster than metropolitan areas of similar size in other states, even though these states are located on the Gulf Coast of the United States and have prosperous energy industries. Eventually, as more data was incorporated, broader patterns and trends began to emerge. For example, there has been a general upward trend in asset prices over the 21 years from 2000 to 2021. However, there have also been times of sudden decreases in asset prices (e.g. stock market crashes and housing price crashes).

First, the method of derivative-based analysis will be applied to the real estate dataset, the DJIA dataset, and the inflation dataset to answer the associated research questions. The results will be discussed, and conclusions will be drawn.

Research Questions

There were ten major research questions that were addressed in this analysis:

1. "In a given metropolitan area, which 1-year, 2-year, 5-year, or 10-year period had the greatest increase in normalized price?" and "What were the highest-performing metropolitan areas in a given 1-year, 2-year, 5-year, or 10-year period."
2. "Given a certain amount of money to invest, which metropolitan area offered the best returns over the entire 24-year period?"
3. "Given two metropolitan areas, which one had a greater increase in median price over the entire 24-year period?"
4. "In a given metropolitan area, how many years will it take for the median price for a single-family home to double?"
5. "Given the price data for a metropolitan area for 2021, will prices in that metropolitan area rise or fall in the next year?"
6. "Is there a relationship between doubling time and the political party affiliation of the senators that represented the metropolitan areas and their respective states during the years in which the median price of a single-family home doubled?"

7. “Is there a statistically significant correlation between doubling time and the population size of the metropolitan area, i.e. do smaller metropolitan areas double in price faster than larger ones?”
8. “What is the regional distribution of the metropolitan areas that doubled in price, e.g. did metropolitan areas on the East and West Coasts of the United States double in price faster than metropolitan areas in the interior of the country?”
9. “Can it be determined from the available data when there is an asset price bubble in a particular metropolitan area or region?”
10. “How can an asset price bubble be defined mathematically?”

The first five research questions are important because they may be used by potential real estate investors to determine which metropolitan areas are likely to have the fastest return on investment. Leusin defines “return on investment” as “the gain or loss a real estate property generates minus its initial costs over a specific period of time” (Leusin, 2017). The last five questions are important because they deal with larger political, social, and economic trends.

The Data Pipeline

The first step in the data pipeline was preprocessing the data. Hagan explains that “Data preprocessing consists of such steps as normalization, non-linear transformations, feature extraction, coding of discrete inputs/targets, handling missing data, etc.” (Hagan, 2014). For this analysis, data preprocessing included data cleaning, type conversion, and feature extraction. First, the data was read into the program from a CSV file stored on disk. Once the data was read into the program, it was stored in a Pandas DataFrame for easy access and manipulation. Each of the fields in the DataFrame was initially interpreted as a string. Rows with missing values were dropped. Then the program performed automatic type conversion from string to float so that the powerful mathematical and linear algebra functions of Numpy could be used on the numerical features in the dataset.

Features that will be extracted from the Real Estate Dataset

Next, features were extracted from the real estate dataset. The following five features were chosen to compare metropolitan areas from the real estate dataset because they were the most informative: time, median price, normalized median price, state, and region. Normalized median price and region are derived features.

Development of Methods and Algorithms

In order to answer the research questions, several methods and algorithms were developed. Initially fixed-length time periods such as years were used to partition and analyze the real estate dataset at a higher level of granularity. Then these were generalized to multi-year spans of time. One of the challenges was the different formats of the datasets. Like all tabular datasets, the real estate dataset consists of rows and columns. Each row represents a different metropolitan area. There are 911 metropolitan areas in all. The first five columns are features, such as *RegionName*, *RegionType*, and *StateName*. The remaining columns are the months that the price data was recorded for. Therefore, the real estate dataset is unique among the datasets because in this dataset time is the horizontal axis; whereas, in the other datasets time is the vertical axis. The scale of time is different, too, among the three datasets. In the CPI dataset, time is measured in months. In the real estate dataset, time is measured in months. In the DJIA dataset, time is measured in days. All of these timescales are important, but in order to make meaningful comparisons, years were chosen as the common timescale.

It was important to normalize the prices for the real estate dataset because housing prices can vary widely from one market to another. For example, the median price of a single-family home in Selma, Alabama in August 2021 (the most recent month for which data is available in the real estate dataset) was \$60,503. The median price of a single-family home in San Jose, CA in August 2021 was \$1,564,395 – over \$1.5 million dollars! Normalization helps to even-out these differences so that appreciation and depreciation trends may be seen over time. Normalization yields an apples-to-apples comparison instead of an apples-to-oranges comparison.

There are different kinds of normalization, though. Two of the most frequently used types of normalization for vectors are L1 normalization and L2 normalization. L1 normalization takes the sum of the absolute values of the vector to be normalized. L2 normalization sums the squares of the absolute values of the elements of the vector.

$$|\mathbf{x}|_1 = \sum_{r=1}^{n} |x_r|$$

Eq. 1. The formula for L1 normalization.[2]

$$|\mathbf{x}| = \sqrt{\sum_{k=1}^{n} |x_k|^2}$$

Eq. 2. The formula for L2 normalization.[3]

For this analysis, a different method of normalization was chosen. The maximum price of a single-family home in a given metropolitan area across a fixed time period (one or more years) was found. Then each price was divided by the maximum price in order to normalize it.

The first question has two parts: "In a given metropolitan area, which 1-year, 2-year, 5-year, or 10-year period had the greatest increase in normalized price?" and "What were the highest-performing metropolitan areas in a given 1-year, 2-year, 5-year, or 10-year period." To answer the first part of this question, the following algorithm was designed:

Algorithm 1a

```
1) partition the dataset into user-specified equal length time periods
2) for each period
    a) normalize the prices by dividing each price by the maximum price for its metropolitan area over the specified timespan
    b) change in price = final price - initial price
    c) store the name of the of the metropolitan area and the change in its normalized price in a data structure
3) sort the metropolitan areas in descending order by change in price.
4) iterate through the periods comparing the increase in normalized price in each period for that metropolitan area.
5) return the highest increase in normalized price and the period in which it occurred
```

To answer the second part, the following algorithm was used:

Algorithm 1b

```
1) partition the dataset into user-specified equal length time periods
2) For a user-defined period
    a) normalize the prices by dividing each price by the maximum price for its metropolitan area over the specified timespan
    b) change in price = final price - initial price
    c) store the name of the metropolitan area and the change in its normalized price in a data structure
3) sort the metropolitan areas in descending order by change in price
4) print the user-specified number of metropolitan areas and their respective increases in normalized price
```

After these algorithms were implemented, manual testing was performed on them, and they were found to work on any number of metropolitan areas in the dataset.

The following algorithm was used to answer the second research question:

Algorithm 2

```
1) set a user-defined threshold for the maximum initial starting price
2) filter the dataset based on the threshold
3) for each metropolitan area under the threshold
    a) normalize the prices by dividing each price by the maximum price for its metropolitan area over the entire timespan of the dataset
    b) change in price = final price - initial price
    c) store the name of the metropolitan area and the change in its normalized price in a data structure
4) sort the metropolitan areas in descending order by change in price
5) return the name of the metropolitan area with the highest increase in normalized price and its change in price
```

To answer the third research question, the following algorithm was used:

Algorithm 3

```
1) for each metropolitan area in the dataset
    a) normalize the prices by dividing each price by the maximum price for its metropolitan area over the entire timespan of the dataset
    b) change in price = final price - initial price
    c) store the name of the metropolitan area and the change in its normalized price in a data structure
2) sort the metropolitan areas in descending order by change in price
3) define two variables to store the increase in normalized price for the two user-specified metropolitan areas
4) iterate through the sorted list of metropolitan areas, updating each metropolitan area's price variable
5) choose the metropolitan area that had the highest increase in normalized price over the entire timespan of the dataset
6) print the names and the respective increases in normalized price for the two metropolitan areas
```

To compute the shortest time interval for the median price of a single-family home to double, the following algorithm was used:

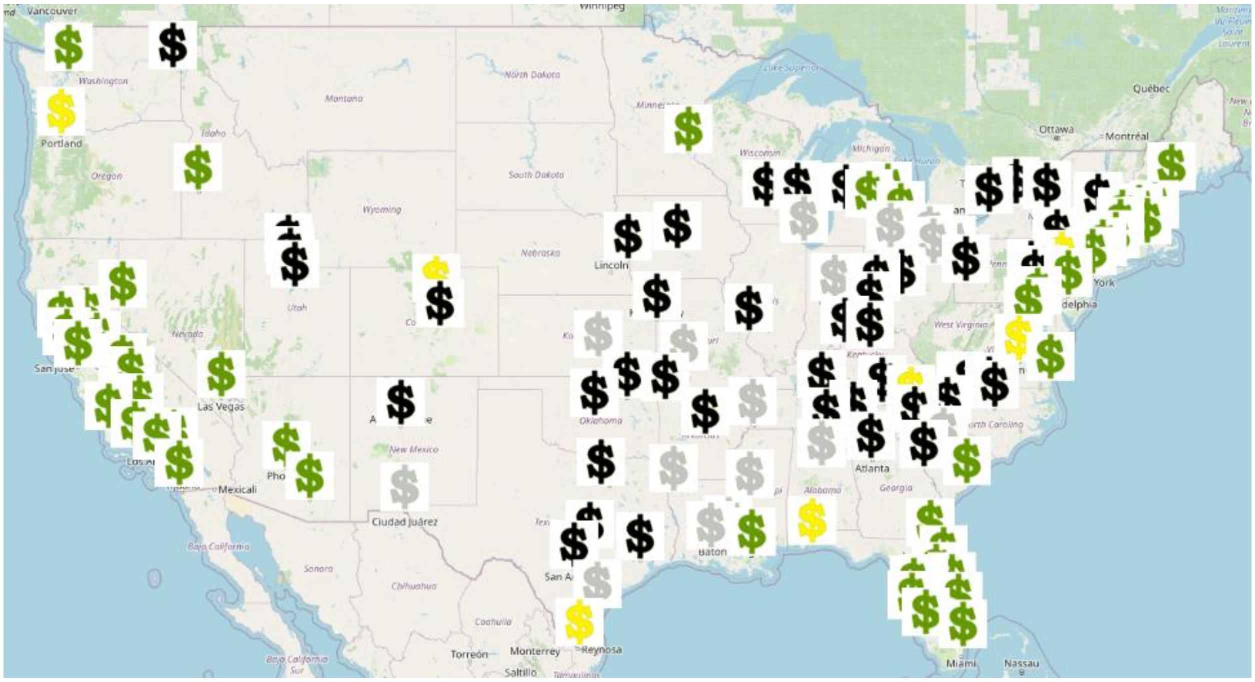
Algorithm 4

```
1) find the minimum monthly price for a user-specified metropolitan area
2) iterate through the succeeding columns until a price greater than or equal to twice the minimum price is found
3) count the number of months that occurred between these two price points
4) divide the number of months by 12 to obtain the doubling time in years
round the doubling time in years to the desired precision
```

There is also a geospatial component to this analysis because the state and region in which each metropolitan area is located were considered when assessing appreciation rates.

In this analysis, the doubling time was rounded to the nearest tenth and the results were displayed in Map 1.

Map 1. The 120 Largest U.S. Metropolitan Areas Grouped by the Time to Double



Generated with Leaflet.js[4]

Legend

Green: Metropolitan areas where the prices doubled in 0 to 10 years.

Yellow: Metropolitan areas where the prices doubled in 10 to 20 years.

Black: Metropolitan areas where the prices doubled in 20 to 30 years.

Gray: Metropolitan areas where the prices did not double during the years for which there is data available.

This map shows the largest 120 U.S. metropolitan areas by population grouped by the time for the normalized median price of a single-family home in that metropolitan area to double. The metropolitan areas denoted by the green dollar sign icon doubled in price the fastest, and the metropolitan areas denoted by the black dollar sign icon doubled in price the slowest.

Doubling times varied widely between states and within certain states. Among the 120 most populous metropolitan areas in the dataset, the shortest doubling time was San Jose, CA (4.5 years). The longest doubling time was undefined because there were some metropolitan areas (e.g. Chicago, IL and Cleveland, OH) that did not double in price during the 25 years that are recorded in the dataset, though prices in these metropolitan areas did generally increase.

To answer the fifth research question, the following algorithm was used:

Algorithm 5

1) compute the derivative of normalized price with respect to time for each year in the dataset for a user-specified metropolitan area
2) compute the median derivative for that metropolitan area over the past 24 years
3) If the derivative of price with respect to in 2021 was *more than 20% above* the median derivative for that metropolitan area over the past 24 years, then the next year was predicted to be a good (above average) year for price increases in that metropolitan area.
4) If the derivative of price with respect to time in 2021 was *more than 20% below* the median derivative for that metropolitan area over the past 24 years, then the next year was predicted to be a poor (below average) year for price increases in that metropolitan area.
5) If the derivative of price with respect to time in 2021 was *between 80% and 120%* of the median derivative for that metropolitan area over the past 24 years, then the next year was predicted to be a fair (average) year for price increases in that metropolitan area.

Algorithm 6

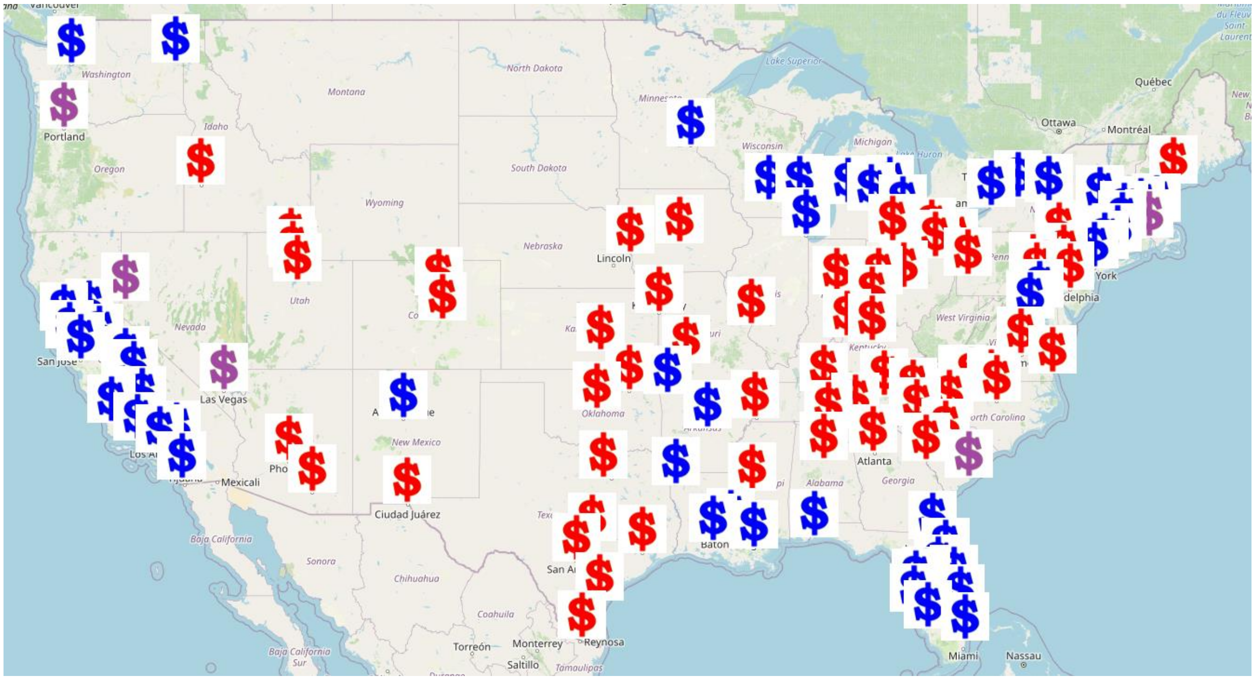
1) Assign the states to regions
2) For each state, take the metropolitan area with the highest median price and the lowest median price and compare them to the metropolitan area to the highest and lowest median price respectively in every other state in that region.

By studying these derivatives, the 2008-2009 housing crash can be clearly seen in metropolitan areas across the country.

For example, in Las Vegas, NV, the derivative of normalized price w.r.t time was negative from 2007 to 2011, and the median price of single-family homes was decreasing in Las Vegas during this time. This sharp decrease in median home prices had long-lasting effects on the residential real estate market in Las Vegas. The median price of a single-family in the metropolitan area did not reach its pre-crash high of $370,412 again until July 2021. It took approximately 15 years for real estate prices to recover in Las Vegas.

To ascertain whether there was any correlation between political representation and the rate at which residential housing prices increased in the 120 largest U.S. metropolitan areas, the party affiliation of the U.S. senators that represented these metropolitan areas' states during the periods in which the median price of a single-family home doubled were determined. The median price of single-family homes in metropolitan areas on the East and West Coasts of the U.S. tended to double faster than those in the interior of the country.

Map 2. The 120 Largest U.S. Metropolitan Areas Grouped by Their Senate Representation



Generated with Leaflet.js[4]

<u>Legend</u>

Red: Metropolitan areas that are located in a state that was represented by two Republican senators during the years in which the prices in that metropolitan area doubled.

Blue: Metropolitan areas that are located in a state that was represented by two Democratic senators during the years in which the prices in that metropolitan area doubled.

Purple: Metropolitan areas that are located in a state that was represented by one Republican and one Democratic senator during the years in which the prices in that metropolitan area doubled.

This map shows the largest 120 U.S. metropolitan areas by population grouped by the political affiliation of their state's U.S. Senate delegation. Red means two Republican senators, blue means two Democratic senators, and purple means one Republican senator and one Democratic senator.

The coastal metropolitan areas are more likely to support Democratic political candidates in Senate races as shown in Map 2, but correlation does not prove causation and other factors could be at play (e.g. education, income level, immigration, and local industries). The average doubling time of the metropolitan areas and their Senate representation during that time is shown in Table 1.

Table 1. Average Doubling Time of the Metropolitan Areas by Political Party of their Senate Representation

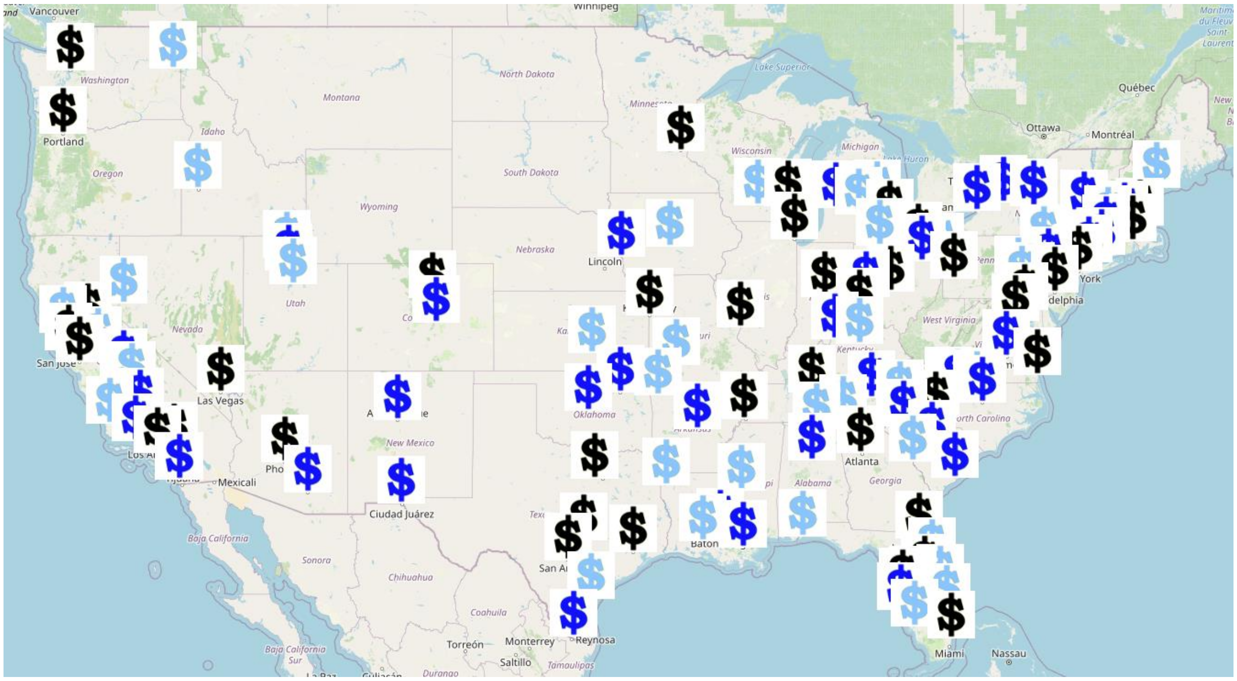
Political Representation in the U.S. Senate	Average Doubling Time in Years
Two Democratic senators	11.2
Two Republican senators	20.7
One Democratic and one Republican senator	7.7

This table shows the average time for the normalized median price of a single-family home to double in the 120 largest U.S. metropolitan areas compared to the political affiliation of their Senate representation.

Interestingly, the average doubling time of the metropolitan areas represented by Democratic and Republican senators (7.7) was lower than either the average doubling time of the metropolitan areas represented by Democratic senators (11.2 years), or the metropolitan areas represented by Republican senators (20.7 years).

To answer the question, “Is there a statistically significant correlation between doubling time and the population size of the metropolitan area, i.e. do smaller metropolitan areas double in price faster than larger ones?” the correlation between population size and doubling time was computed. The metropolitan areas are sorted by population size on Map 3.

Map 3. The 120 Largest U.S. Metropolitan Areas Grouped by Population



Generated with Leaflet.js [4]

Legend
Black: Metropolitan areas in the 1 to 40 largest U.S. metropolitan areas by population.
Dark Blue: Metropolitan areas in the 40 to 80 largest U.S. metropolitan areas by population.
Light Blue: Metropolitan areas in the 80 to 120 largest U.S. metropolitan areas by population.

This map shows the largest 120 U.S. metropolitan areas by population grouped by the population size of the metropolitan area. The metropolitan areas denoted by the black dollar sign icon were the most populous, and the metropolitan areas denoted by the light blue dollar sign icon were the least populous.

To compute the correlation between population size and doubling time, first a subset of the 120 most populous metropolitan areas was taken and plotted on a map. In the dataset, the metropolitan areas are sorted in decreasing order of population size. The most populous metropolitan area in the top 120 was New York, NY with a population of 20.1 million in its metropolitan area and *size_rank* 1 (U.S. Census Bureau, 2022). The least populous metropolitan area in the top 120 was Huntsville, AL with a population of 200,574 and *size_rank* 121 (U.S. Census Bureau, 2022). The population of each metropolitan area was not included in the dataset; therefore, the variable *size_rank* (relative metropolitan area size based on population) was used as a proxy for population. For the correlation calculation, the independent variable was *size_rank* and the dependent variable was doubling time. The Pearson correlation coefficient between *size_rank* and doubling time was 0.11. This means that the *size_rank* of the metropolitan area was not correlated with the doubling time of the metropolitan area, hence smaller metropolitan areas did not double in price faster than larger ones.

Next, the eighth research question, “What is the regional distribution of the metropolitan areas that doubled in price, e.g. did metropolitan areas on the East and West Coasts of the United States double in price faster than metropolitan areas in the interior of the country?” was investigated. Seventy-six percent of the metropolitan areas where the median price of a single-family home doubled in less than 10 years were located in a state that borders either the Atlantic Ocean or the Pacific Ocean.

The metropolitan areas that did not follow this pattern were Boise City, ID (7.2 years); Phoenix, AZ (9.3 years); Tucson, AZ (9.7 years); Minneapolis-St. Paul, MN (8.1 years); New Orleans, LA (9.6 years); Detroit, MI (7.9 years); Flint, MI (9.1 years); and Lansing, MI (8.7 years). With the exception of New Orleans, LA, these metropolitan areas are all located inland, i.e. not in a coastal state.

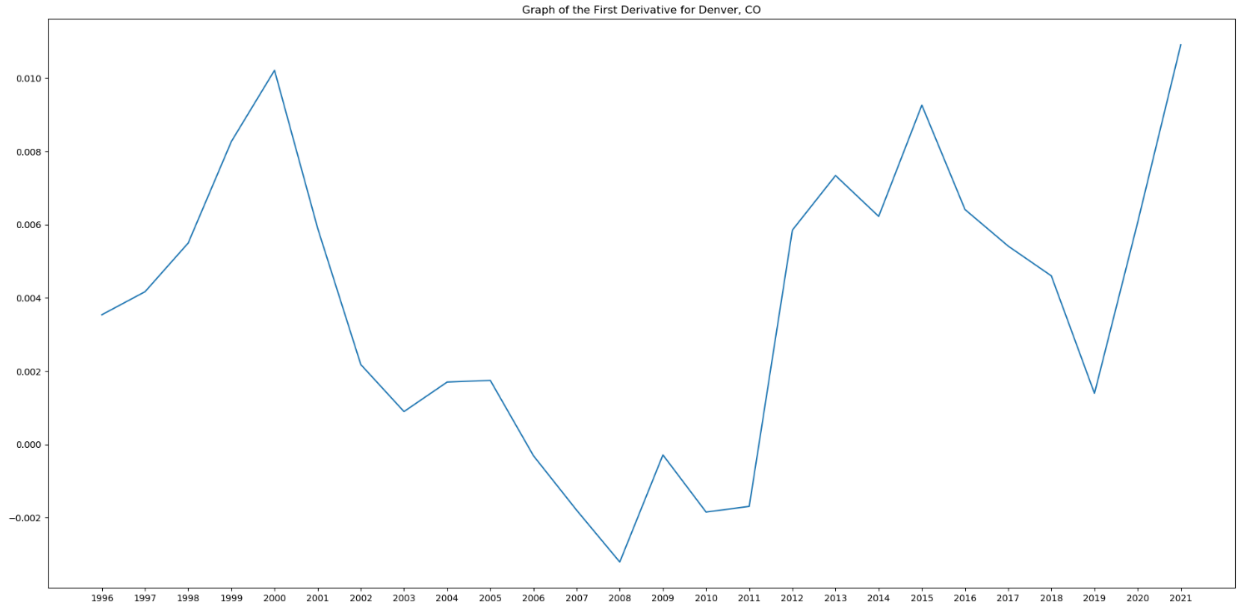
Some of the metropolitan areas with sub-10-year doubling times may be outliers. For example, New Orleans is the only metropolitan area on the Gulf Coast of the United States that had a doubling time of less than 10 years. In New Orleans the doubling time ran from January 1996 to August 2005. Detroit, MI; Flint, MI; and Lansing, MI may also be outliers because the metropolitan areas nearby them had doubling times that were twice as long as these three metropolitan areas in Michigan did.

Metropolitan areas in the interior of the country had longer times than metropolitan areas on the coasts. Only 24 percent of metropolitan areas that had a doubling time of less than 10 years were located in non-coastal states. The average doubling time of metropolitan areas in these states was 13.8 years.

Based on the analysis that was performed, several important patterns can be identified. There was a sharp and continued decrease in the median price of single-family homes in many metropolitan areas across the United States in the period 2008-2009. Metropolitan areas in states that bordered the Atlantic Ocean or the Pacific Ocean were significantly more likely to double in price than metropolitan areas in non-coastal states. Doubling times varied widely between states and within certain states. Metropolitan areas that were represented by two Democratic senators or one Republican senator and one Democratic senator doubled much faster on average than metropolitan areas that were represented by two Republican senators. The *size_rank* of a metropolitan area was not correlated with its doubling time.

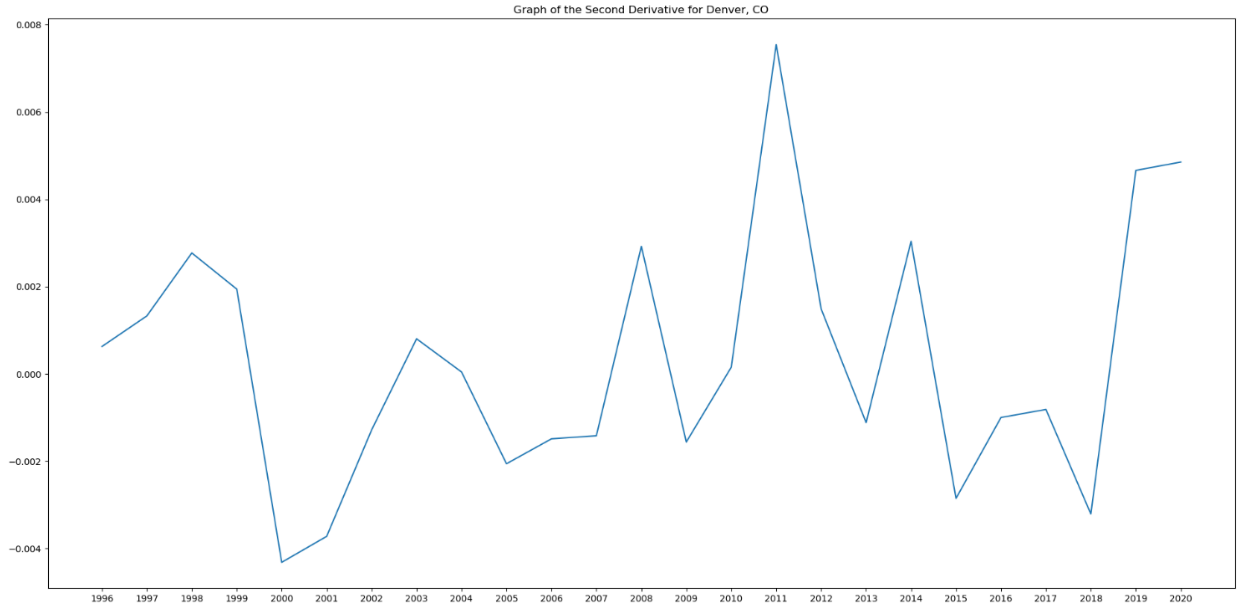
To understand how the rate of appreciation or depreciation in each real estate market changes over time, the first and second derivatives of price with respect to (w.r.t) time were calculated and compared. The first derivative shows whether the prices are increasing or decreasing. It gives the slope of the price function. The second derivative shows whether the change in price is accelerating or decelerating. The second derivative also gives the concavity of the price function. May and Bart explain, “[A] graph is concave up if the line between two points is above the graph, or alternatively if the first derivative is increasing…[A] graph is concave down if the line between two points is below the graph, or alternatively if the first derivative is decreasing. In determining [whether] a curve is concave up or concave down, we want to take the second derivative of a function, or the derivative of the derivative.” (May & Bart, 2022) The first and second derivatives for different metropolitan areas in the dataset were analyzed. The following graphs show the first and second derivatives of normalized price with respect to time for select metropolitan areas.

Figure 2. Graph of the First Derivative for Denver, CO



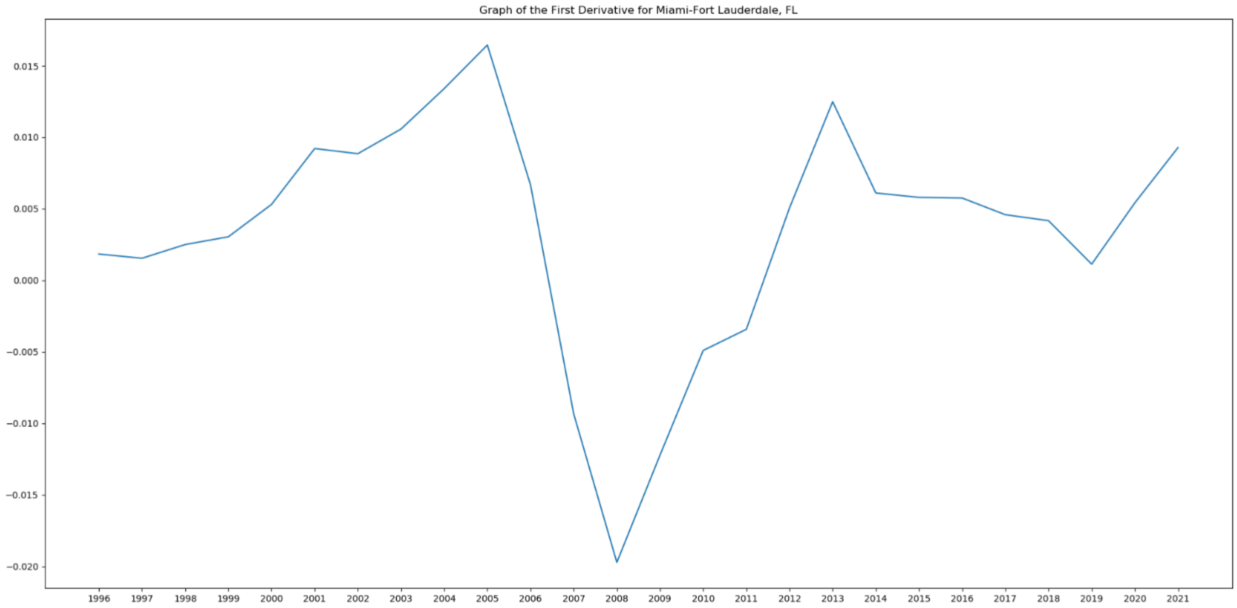
This graph shows the first derivative of the normalized median price of a single-family home in Denver, CO from January 31, 1996 to August 31, 2021. The derivative reached its lowest point in 2008, and it reached its highest point in 2021. Single-family homes in Denver were depreciating in 2007.

Figure 3. Graph of the Second Derivative for Denver, CO



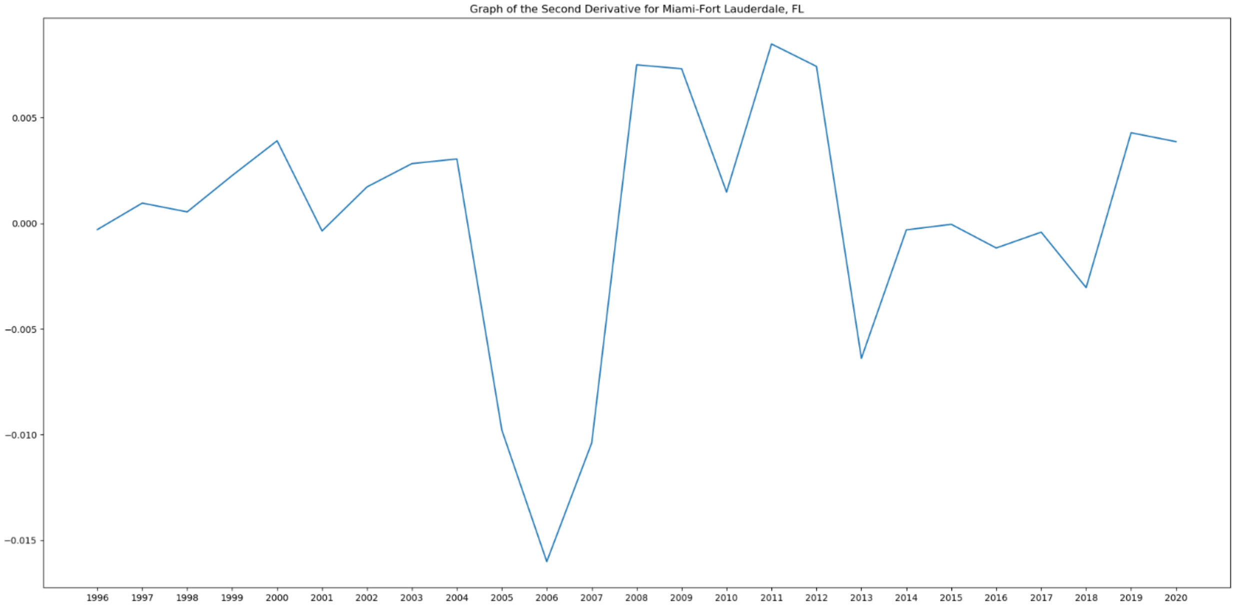
This graph shows the second derivative of the normalized median price of a single-family home in Denver, CO from January 31, 1996 to August 31, 2021. The second derivative changed more abruptly than the first derivative at its local maxima and minima. This is reflected in the fact that the normalized price is still increasing, albeit at slower rate, when the first derivative is positive and the second derivative is negative.

Figure 4. Graph of the First Derivative for Miami, FL



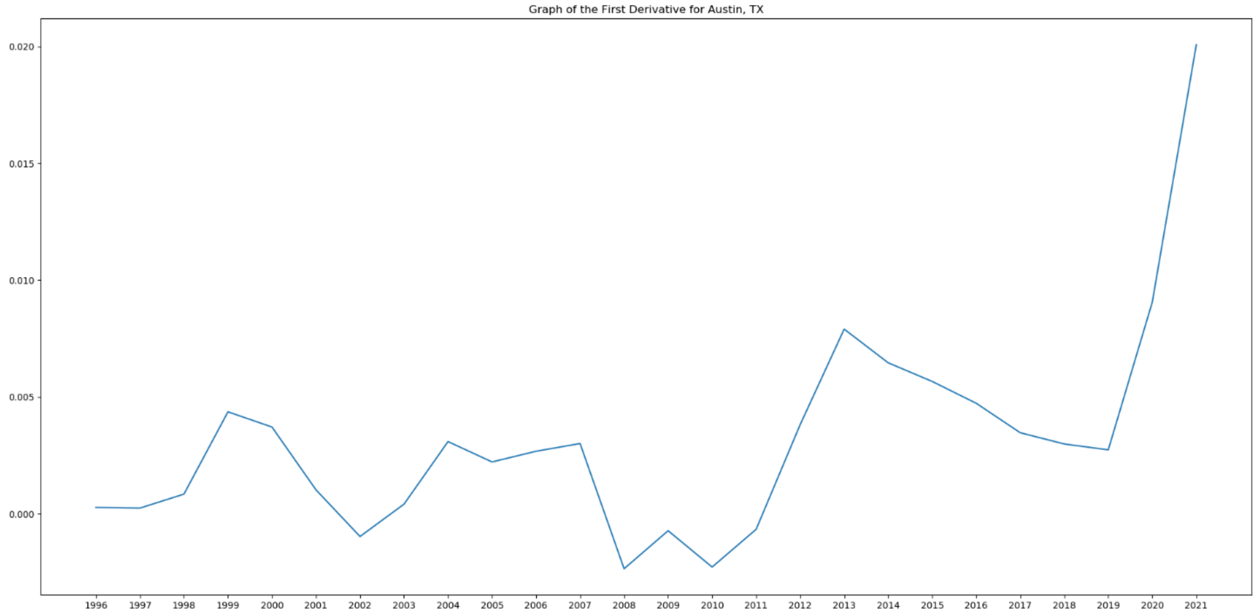
This graph shows the first derivative of the normalized median price of a single-family home in Miami, FL from January 31, 1996 to August 31, 2021. “A housing bubble, or real estate bubble,” has been defined as,” a run-up in housing prices fueled by demand, speculation, and exuberant spending to the point of collapse.” (Investopedia Team, 2020, December 25). The characteristic “V-shape” in the graph is typical of real estate markets that experienced a housing price bubble followed by a housing price crash.

Figure 5. Graph of the Second Derivative for Miami, FL



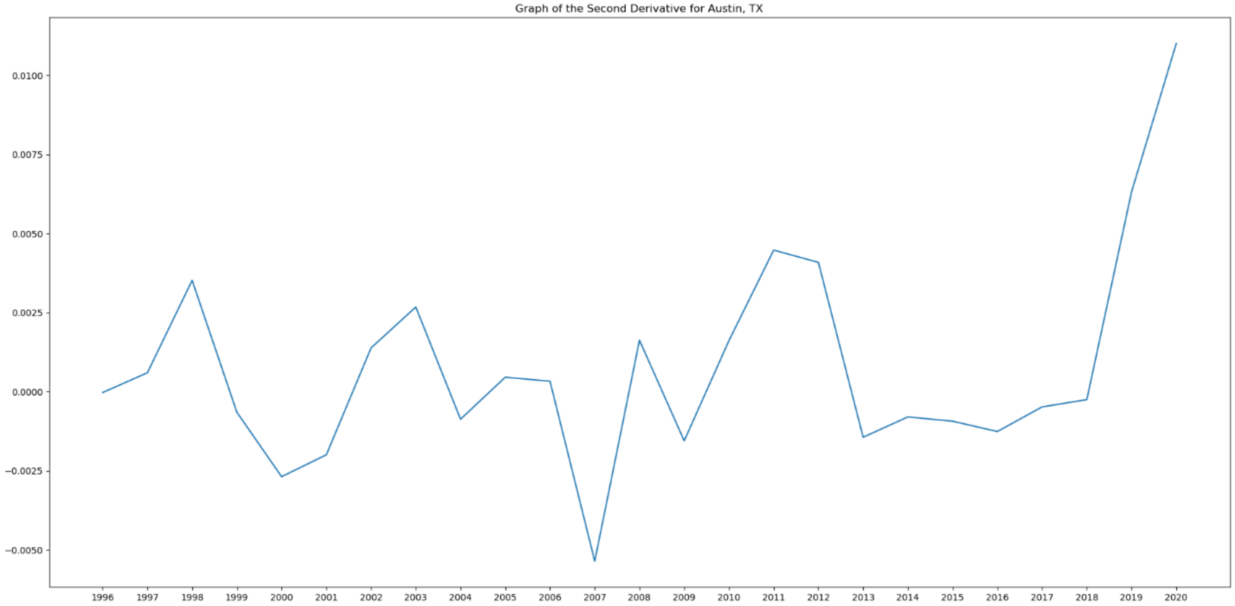
This graph shows the second derivative of the normalized median price of a single-family home in Miami, FL from January 31, 1996 to August 31, 2021. The graph of the second derivative for this metropolitan area is similar to the graph of the first derivative, but it is more staggered even in the recovery phase (the years 2010-2020).

Figure 6. Graph of the First Derivative for Austin, TX



This graph shows the first derivative of the normalized median price of a single-family home in Austin, TX from January 31, 1996 to August 31, 2021. The graph of the first derivative for Austin, TX is more stable than the graph of the first derivative for Miami, FL during the same time, but there is a rapid increase in the normalized median price from 2020-2021. This may indicate that a housing bubble was forming in this metropolitan area.

Figure 7. Graph of the Second Derivative for Austin, TX



This graph shows the second derivative of the normalized median price of a single-family home in Austin, TX from January 31, 1996 to August 31, 2021. From 2018 to 2020, the normalized median price of a single-family home was increasing at an increasing rate in Austin, TX. This provides further evidence that a housing bubble was forming in this metropolitan area at that time.

In all the metropolitan areas displayed, there was a sharp decrease in housing prices during the 2008-2009 recession, also known as the “Great Recession.” For a while, the median price of a single-family home in cyclical markets such as Miami, FL was in “free fall” that is, the price was decreasing at an increasing rate. This can be determined from those points in the dataset at which the first derivative of price with respect to time is negative and the second derivative of price with respect to time is positive.

Coastal states were defined as U.S. states that bordered either the Atlantic or the Pacific Ocean. Non-coastal states were defined as U.S. states that did not border the Atlantic nor the Pacific Ocean. It was observed that the median price of single-family homes grew faster in coastal states than in non-coastal states.

Algorithm 9.

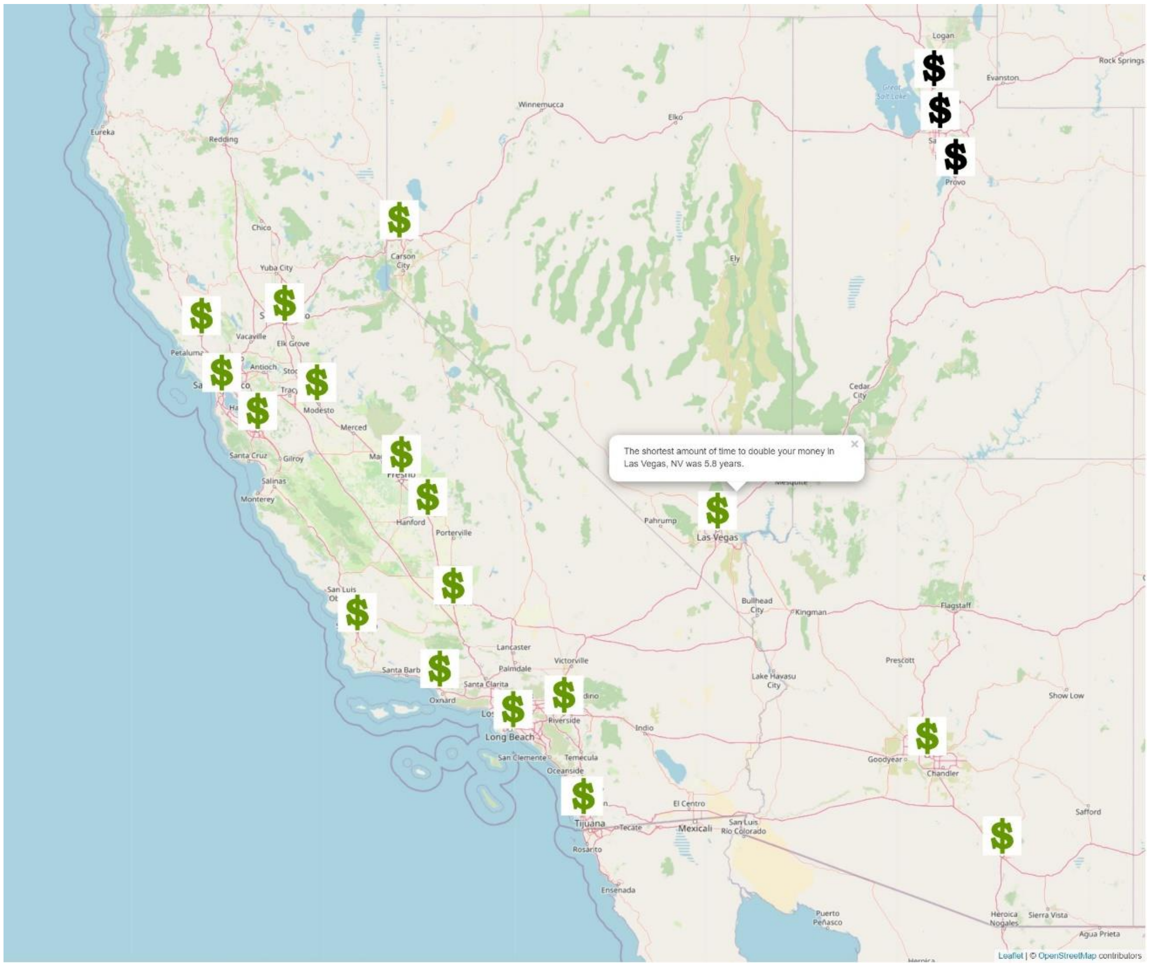
“A housing bubble, or real estate bubble,” has been defined as,” a run-up in housing prices fueled by demand, speculation, and exuberant spending to the point of collapse.” (Investopedia Team, 2020, December 25). To identify when a real estate bubble occurred in a particular region and answer the ninth and tenth research questions, the first and second derivatives were computed for the metropolitan areas in the real estate dataset in order to determine the appreciation rate, or the increase in normalized median price in each metropolitan area. The following thresholds were tested for the increase in normalized median price: 10%, 20%, and 25%, and the error ratio was computed for each threshold. If the real estate prices in a metropolitan area grew as fast as predicted based on the derivative from the previous year(s), then the prediction was considered to be correct. If the real estate prices in a metropolitan area grew much slower or much faster than predicted based on the derivative from the previous year(s), then the prediction was considered to be incorrect. Error ratio is defined as the number of incorrect predictions divided by the total number of predictions. The results of this portion of the analysis are displayed in Table 2.

Table 2. Error Ratios for Different Increase in Median Price Threshold Values

Threshold	Error Ratio
10%	47.0%
20%	50.3%
25%	52.9%

A *bubble* year was defined as a one-year period in which the following two conditions were met: the derivative of normalized price with respect to time in that year was greater than or equal to 120% of the median derivative of normalized price with respect to time in that metropolitan area over the entire 20-year span, and the first and second derivatives were positive. This means that the normalized median price was increasing at an increasing rate. Bubbles may begin or end at any time and may last for more or less than one year (12 months), but one year was a standard measure of time that could be used to compare growth rates in different metropolitan areas within a given region. These indicators were highly correlated with the appearance of real estate bubbles. For example, in the West Coast region, Los Angeles-Long Beach-Anaheim, CA; Riverside, CA; Sacramento, CA; Las Vegas, NV; Reno, NV; and Tucson, AZ all experienced a housing bubble from 2003 to 2005, as shown in Map 4.

Map 4. The Housing Bubble in California, Arizona, and Nevada in the early 2000s


The shortest amount of time to double your money in Las Vegas, NV was 5.8 years.

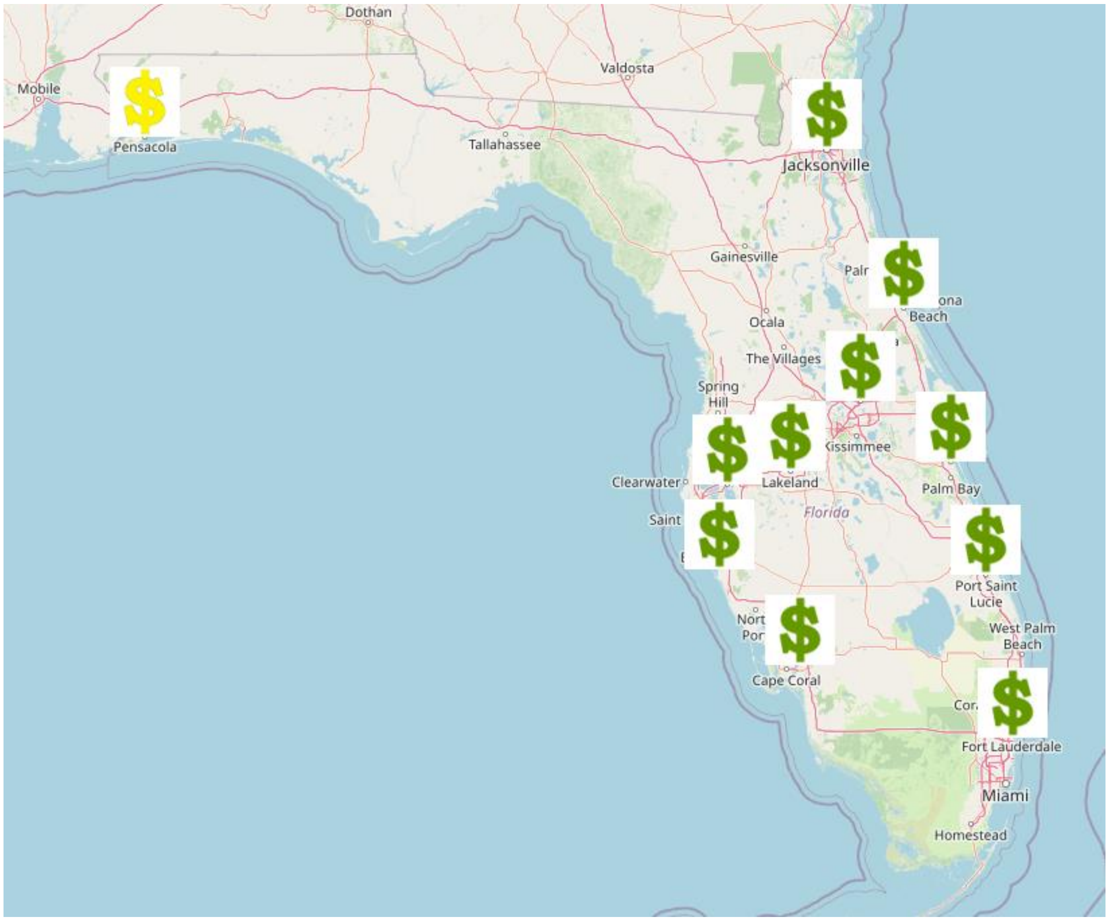
Leaflet | © OpenStreetMap contributors

The median price of a single-family home doubled in less than 10 years in the metropolitan areas depicted in green.

It should be noted that the bubble may have begun prior to 2003 or ended after 2005 in specific metropolitan areas. Because the real estate data was reported monthly, it was not possible to determine the exact start and end dates of the bubble. Nearby metropolitan areas (e.g. Phoenix, AZ, San Diego, CA) also experienced a housing bubble at approximately the same time (+/- 1 year). Therefore, it is concluded that there was a housing bubble in the states of California, Arizona, and Nevada in the early-to-mid 2000s.

A similar pattern was seen in Florida from 2001 to 2006. The conditions for a bubble year were met in most large metropolitan areas in Florida in the first half of the 2000s as shown in Map 5. The bubble there usually lasted from 3 to 5 years.

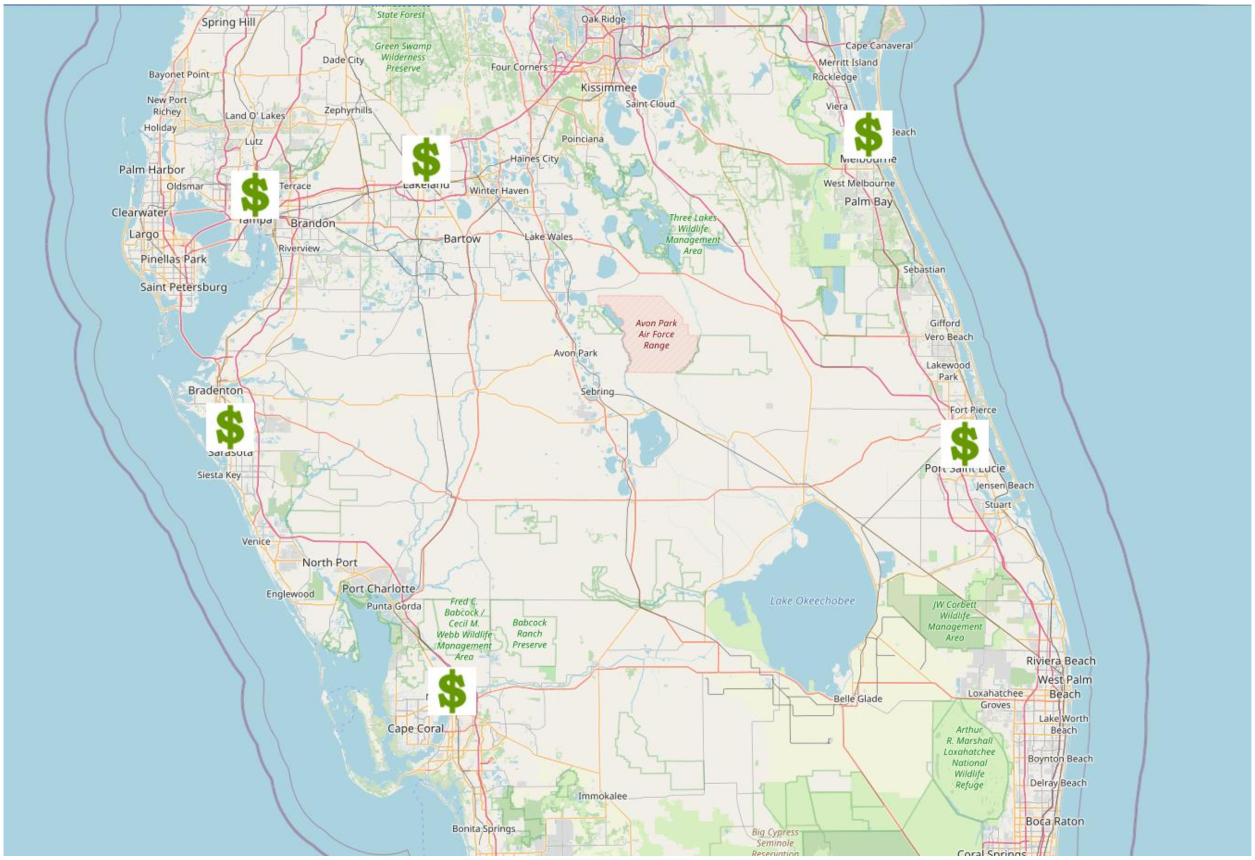
Map 5. The Housing Bubble in Florida in the early 2000s



Housing bubbles occurred in Daytona Beach, Fort Myers, Jacksonville, Melbourne, Miami-Fort Lauderdale, North Port-Sarasota-Bradenton, Orlando, Port St. Lucie, and Tampa, FL in the first half of the 2000s. Pensacola, FL was an outlier. Housing prices there doubled more slowly than they did in peninsular Florida.

Of particular interest were the metropolitan areas in central Florida. This region experienced a sustained housing bubble from 2001 to 2005 as shown in Map 6.

Map 6. The Housing Bubble in Central Florida from 2001 to 2005



There was a housing bubble in Fort Myers, Melbourne, North Port-Sarasota-Bradenton, Port St. Lucie, and Tampa, FL from 2001 to 2005. Lakeland, FL also experienced a housing bubble, but it did not last from 2001 to 2005.

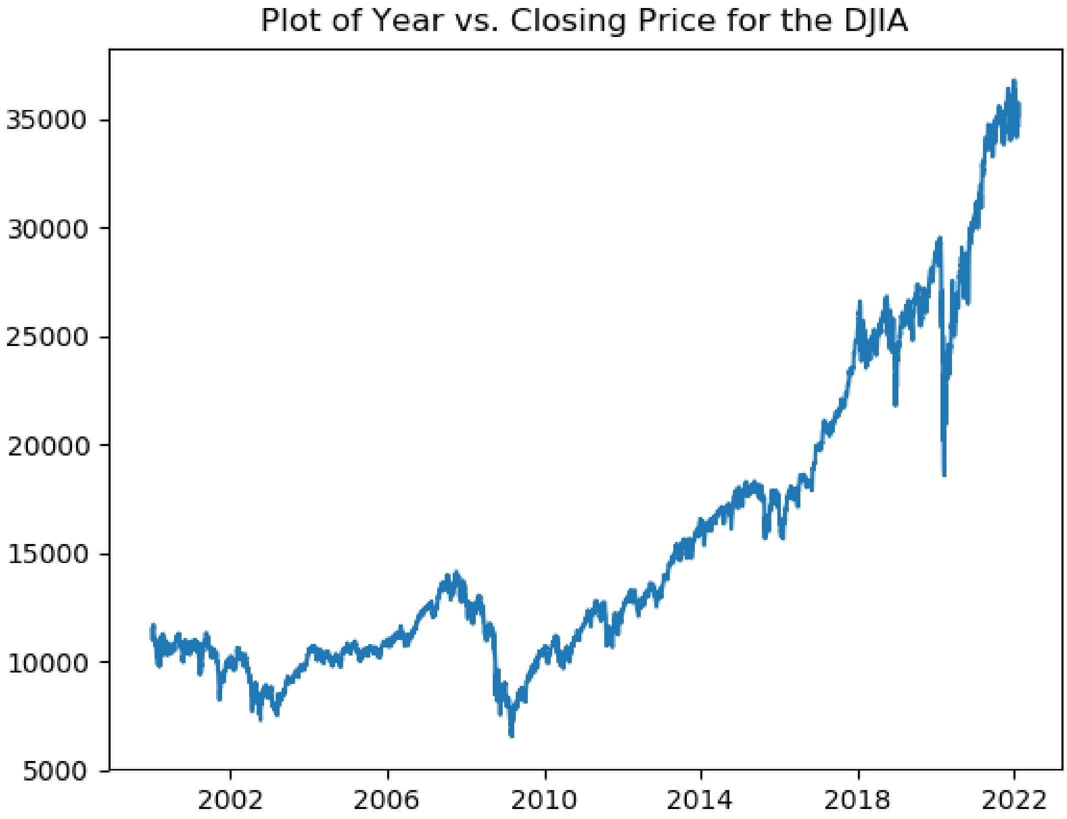
Discussion

Real estate markets are first and foremost local to a specific metropolitan area and county, but they are also affected by national and regional trends. For example, the real estate bubble in the early-to-mid 2000s affected many metropolitan areas in the West Coast and Florida. This is an example of regional trends in the real estate market.

An asset price bubble may occur in any market: residential real estate, commercial real estate, stocks and bonds, precious metals, etc. For example, an asset price bubble that occurs on a national stock market or publicly traded exchange will take the form of a stock price bubble (a.k.a. stock market bubble). The characteristics and drivers of such a bubble are often the same as the characteristics and drivers of a real estate bubble (e.g. speculation drives up the price of the asset rapidly).

In the DJIA dataset, each row is a trading day. The columns represent the features: the opening price that day, the highest price that day, the lowest price that day, the closing price that day, the adjusted closing price that day, and the trading volume that day. Ganti observes that the difference between the closing price and the adjusted closing price is, “The adjusted closing price amends a stock's closing price to reflect that stock's value after accounting for any corporate actions.” (Ganti, 2020). For this analysis, the raw closing price was used for calculating derivatives. This raw closing price over time is shown in Figure 8.

Figure 8. Graph of the Daily Closing Price of the Dow Jones Industrial Average



This figure shows the closing prices (in points) of the Dow Jones Industrial Average (DJIA) from December 31, 1999 to February 11, 2022.

Asset price bubbles can be detected by derivative-based analysis in the Dow Jones Industrial Average dataset. To identify when the Dow Jones Industrial Average experienced a stock price bubble, the first and second derivatives of closing price with respect to time and the annual percentage change in the DJIA were computed and compared (Kenton, 2022). The median derivative for the closing price with respect to time was also computed and a threshold of 120% was set, that is, if the derivative of closing price with respect to time was greater than or equal to 120% of the median derivative of closing price with respect to time across the entire 21-year timeframe, then it was considered a bubble year. The median of the first derivative of price with respect to time was 5.22. The median of the second derivative of price with respect to time was 1.93. The only year in which the first derivative was greater than or equal to 120% of the median derivative was 2022. However, this is likely due to the incomplete data available in the dataset concerning the performance of the DJIA in 2022. The dataset only contains data about the closing price for the DJIA from January 3rd through February 11th, 2022. Also, the DJIA was close to its all-time high of $36,799.65 during this period. It is likely that if data concerning the performance of the DJIA for the rest of 2022 was included, the first derivative for 2022 would not be greater than or equal to 120% of the median derivative for the 2000-2022 timeframe. Derivative-based analysis, with a threshold of 20% as used in this study, was better at detecting asset price bubbles in the real estate market than in the stock market. But this may be due to the incompleteness of the stock market data provided rather than an inherent weakness in the algorithm itself.

When a large asset price bubble bursts, it sends shockwaves throughout the U.S. economy. But the observed pattern in all three of the economic/financial datasets has been a general upward trend. That is, as time passes (*t* increases), real estate prices *usually* increase, stock prices *usually* increase, and the price of goods and services *usually* increase. However, it is not a smooth linear increase because real estate and stock prices have also seen temporary, and sometimes sharp, decreases in value, e.g. the 2008-2009 financial crisis and recession.

Part III: Other Methods for Performing Time Series Analysis

As shown in Part I, other methods have been proposed for multivariate time series data analysis and prediction.

As Ferenti explains, "The methods of time series analysis can be very broadly divided into two categories: time-domain and frequency-domain methods. Frequency-domain methods are based on converting the time series, classically using Fourier transform, to a form where the time series is represented as the weighted sum of sinusoids… The vast majority of time series analyses, however, apply time-domain methods." (Ferenti, 2017)

$$\int_{-\infty}^{\infty} F\,(k)\,e^{2\,\pi\,i\,k\,x}\;d\,k$$

Eq. 3. Fourier Transform formula [5]

Autocorrelation, cross-correlation, and regression analysis are time domain methods. Two types of regression analysis that are frequently employed on time series data are support vector machines and neural networks.

In this section, these machine learning representations will be explained at a high-level and a prospective approach for how they may be applied to the real estate dataset, the DJIA dataset, and the inflation dataset will be described. Support vector machines are frequently used for classification, but their use for regression has also been well-documented (Bishop, 2006; Ristanoski et al., 2013). A neural network of two or more layers can approximate any function, and this machine learning representation is frequently used for regression tasks also.

Support Vector Machine

In Support Vector Machine regression, a support vector machine is trained by showing it examples of the data that it is supposed to perform regression on (see Figure 9).

Figure 9. High-level Concept of a Support Vector Machine (SVM)

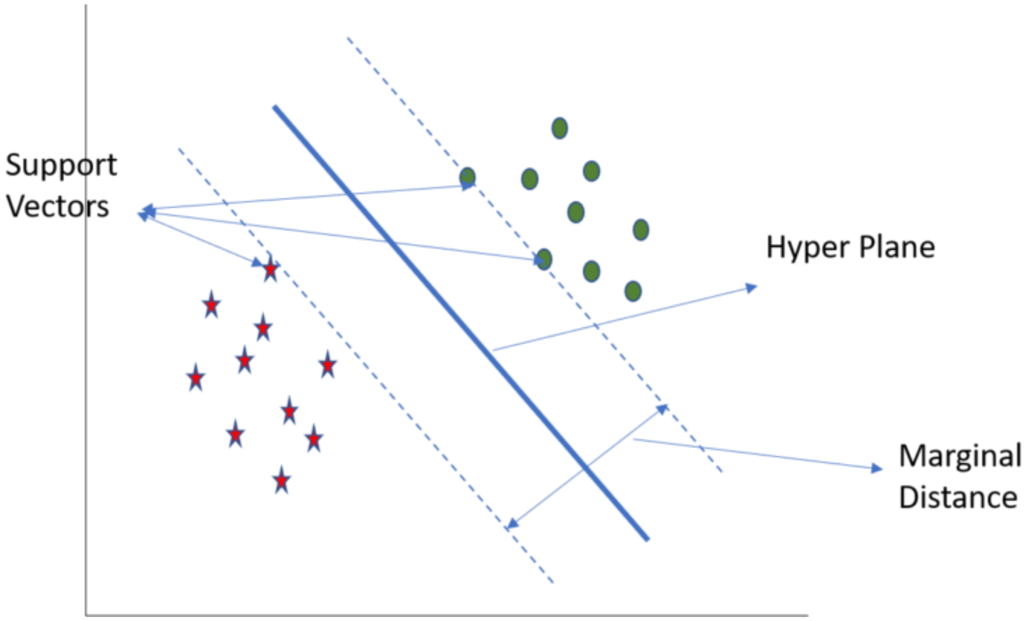


Image Credit: Gaurav Chauhan (Chauhan, 2021) https://machinelearninghd.com/sklearn-svm-starter-guide/

For the economic/financial datasets, this data would be the median price of a single-family home in the 911 largest U.S. metropolitan areas from 1996 to 2021 and the closing prices of the Dow Jones Industrial Average from 2000 to 2021. A support vector machine can be trained on these datasets and used to perform regression analysis on them. Support vector machines are resistant to outliers because “the model produced by Support Vector Regression depends only on a subset of the training data, because the cost function ignores samples whose prediction is close to their target,” according to the scikit-learn documentation (Pedregosa et al., 2011).

The data about real estate prices and stock prices may be partitioned according to an 80-10-10 split between the training set, the test set, and the validation set. Lazzeri said, “data scientists usually transform their time series data sets into a supervised learning [problem] by exploiting previous time steps and using them as input and then leveraging the next time step as output of the model.” (Lazzeri, 2020). The svm.SVR module from scikit-learn may be used to make predictions about future real estate and stock prices (Pedregosa et al., 2011).

Neural Network

Neural networks can be used to perform nonlinear regression on a dataset. The same data would be used to train the neural network that was used to train the support vector machine, although the preprocessing steps might differ slightly. The use of neural networks for time series analysis has been well-documented (Katarya & Rastogi, S., 2018; Shterev, Metchkarski, N.S., & Koparanov, K.A., 2022; Ghanbari & Borna 2021; Li, L., Huang, S., Ouyang, Z., & Li, N. 2022; Kasfi, K. T., Hellicar, A., & Rahman, A., 2016). According to Lazzeri, the first step in using a neural network on time series data is to “reframe [the] time series forecasting problem as a supervised learning problem.” (Lazzeri, 2020). Most neural network representations are designed to work on supervised learning problems.

In a neural network there is an input layer, one or more hidden layers, and an output layer as shown in Figure 10.

Figure 10. A Feed-forward Neural Network

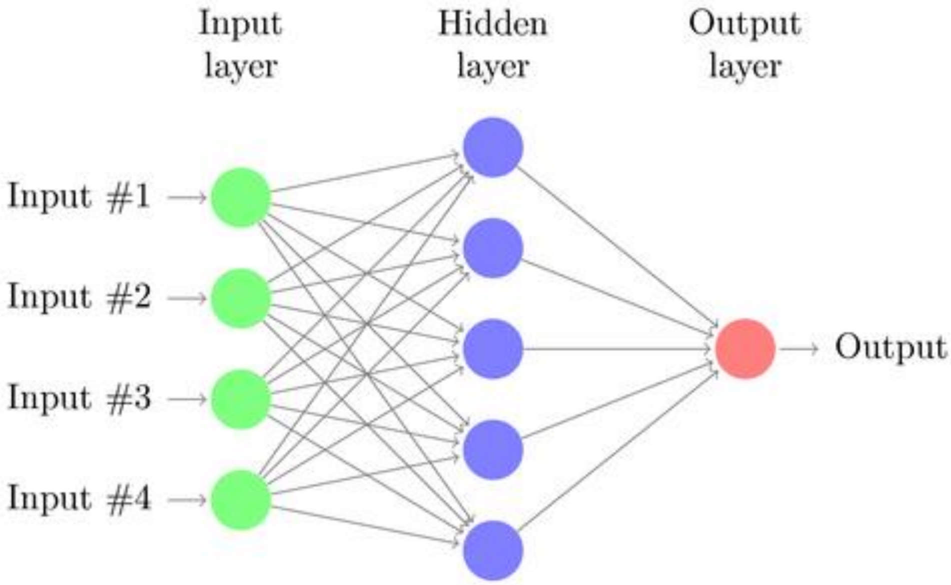


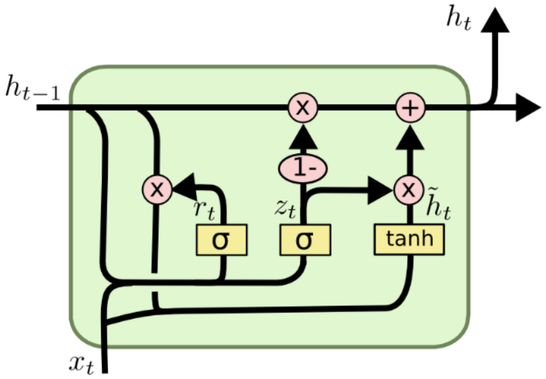
Image Credit: Gaurav Chauhan (Chauhan, 2021) https://machinelearninghd.com/sklearn-svm-starter-guide/

“Neural networks learn by adjusting the weights,” (Kamangar, F., personal communication, September 2020). TensorFlow is a framework for training neural network machine learning models. It is based on the concept of a computational graph. Goodfellow et al. said, “In the general case, an array of numbers arranged on a regular grid with a variable number of axes is known as a tensor” (Goodfellow et al., 2016). TensorFlow performs automatic numerical differentiation by backpropagating partial derivatives of tensors through a computational graph. Keras is an easy-to-use API for TensorFlow. These tools may be used to build neural networks quickly and efficiently.

Two important neural network architectures for regression problems are feed-forward neural networks and long-short term memory (LSTM) neural networks, which are a type of recurrent neural networks (RNN). Russell and Norvig contrast the two architectures as follows:

A **feed-forward network** has connections only in one direction--that is, it forms a directed acyclic graph. Every node receives input from “upstream” nodes and delivers output to “downstream” nodes; there are no loops. A feed-forward network represents a function of its current input; thus, it has no internal state other than the weights themselves. A **recurrent network,** on the other hand, feeds its outputs back into its own inputs. This means that the activation levels of the network form a dynamical system that may reach a stable state or exhibit oscillations or even chaotic behavior. Moreover, the response to the network to a given input depends on its initial state, which may depend on previous inputs. Hence, recurrent networks (unlike feed-forward networks) can support short-term memory.” (Russell & Norvig, 2009).

Figure 11. Long Term Short Term Memory (LSTM) Architecture



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Image Credit: Christopher Olah (2015, August 27). http://colah.github.io/posts/2015-08-Understanding-LSTMs/

It is proposed to train a LSTM neural network using TensorFlow and Keras to perform nonlinear regression on the real estate and stock price datasets and the performance be compared to that of the support vector machine using the R^2-score as the quantitative measure of performance.

Part IV: Time Series Forecasting

Often, time series analysis is performed with the intent of predicting future values that the time series may take, which is called time series forecasting. Baheti & Toshniwal remark, “Identifying potential trends in time series is important because it imparts knowledge about what has taken place in the past and what will take place in time to come.” (Baheti & Toshniwal, 2014). Some of the applications of time series forecasting were discussed in Part I. Now the methods used to perform such forecasting will be discussed in more detail.

As Lazzeri observes, “[T]here are four main categories of components in time series analysis: long-term movement or trend, seasonal short-term movements, cyclical short-term movements, and random or irregular fluctuations.” (Lazzeri, 2020) It has been shown that the real estate dataset and the DJIA dataset both demonstrate an upward trend. This was observed in each of the datasets individually as well as across the approximately 21 years in which they overlap. An example of seasonal short-term movements would be homebuyers buying fewer new homes in the winter and prices decreasing during the winter months (December, January, February) because of decreased demand. More specialized techniques for time series analysis might be able to detect this phenomenon in the real estate dataset, but it was not seen via the derivative-based analysis in this study because the level of granularity was annual rather than monthly. Cyclical short-term movements could be the periodic ups and downs in the closing price of the DJIA. Over long timescales (several years or more) these short-term fluctuations average out. But they are significant to day traders and financial commentators. Random fluctuations could be considered noise in the data. Preprocessing was performed on the datasets to remove missing values and incomplete rows. Potential sources of noise and specific de-noising algorithms are beyond the scope of this study.

Part V: Conclusion

Time series analysis and forecasting are important to a wide range of fields across science, industry, technology, politics, economics, and the environment. The methods of time series analysis are diverse and multifaceted. By comparing the derivatives of time series data, researchers can ascertain whether a particular feature (e.g. the price of an asset) is increasing or decreasing in value and the rate at which this change is occurring. But the most valuable insights come as data is aggregated over many experimental units and over a long period of time. Then a general trend in the data points usually emerges. For example, when the economic datasets were analyzed using derivative-based analysis it was observed that the closing price of the Dow Jones Industrial Average increased from $11.497.12 on Dec 31, 1999 to $34,738.06 on February 11, 2022. The inflation rate, as measured by the Consumer Price Index fluctuated from a low of 2.7% in December 1999 to a high of 8.5% in March 2022. Therefore, there was a general upward trend in the economic datasets.

Another insight that came out of this study was the importance of normalizing the data prior to performing computations and detailed analysis on it. Normalization is very important when the range of values of the variable of interest in a dataset is large. It permits apples-to-apples comparisons of values in a dataset.

Many representations, methods, and algorithms were researched during the course of this study. Some of them work better on time series data than others do. Sometimes modifying the structure of a representation can significantly improve its performance on time series data. For example, LSTMs have been shown to outperform feed-forward neural networks because the former take into the account the temporal component of the data (Lazzeri, 2020).

Derivative-based analysis was chosen for this study because it was simple to implement, easy to explain, and did not require time-consuming training iterations, massive amounts of training data, nor powerful GPUs. When data is aggregated over many experimental units and over a long period of time, a general trend in the data points usually emerges. Derivative-based analysis works well for identifying trends in the data and inflection points in the functions that model that data. For example, when the first and second derivatives change from being positive to negative, it may indicate that a market correction is imminent. However, derivative-based analysis is primarily a tool for understanding the past and identifying patterns in data, rather than forecasting the future, as it has limited predictive power. It was easier for the program that computed the first and second derivatives to detect when a bubble had occurred, than it was for it to predict when the *next* bubble would occur or how long it would last. Nevertheless, derivative-based analysis is an important and useful tool that data scientists and machine learning practitioners can use to provide new insights into time series data. In the future, this method could be added on to ML algorithms or incorporated as a library function in one of the popular machine learning libraries for Python. Derivative-based analysis may be used as an early-stage exploratory analysis tool to assist ML engineers in understanding their data and choosing more specific forecasting methods (e.g.) LSTMs, ARIMA, or regression algorithms.

Footnotes

[1] *Index Component Weights of Stocks in the Dow Jones Industrial Average*

indexArb https://indexarb.com/indexComponentWtsDJ.html

Accessed 2022, November 19. Copyright 2000-2022 Ergo Inc. All Rights Reserved Worldwide. Dow Jones^{SM} and Dow Jones Industrial Average^{SM} are famous, well-known, and internationally recognized trademarks of Dow Jones & Company, Inc. and have been licensed for use by Ergo Inc.

[2] Weisstein, 2022, “L^1-Norm”

[3] Weisstein, 2022, “L^1-Norm”

[4] https://leafletjs.com/examples/custom-icons/

[5] Weisstein, 2022, “Fourier Transform”

References

A

Abadi, M. (2018, May 10). *Even the US government can't agree on how to divide up the states into regions*. Business Insider. https://www.businessinsider.com/regions-of-united-states-2018-5

Agafonkin, V. (2022, September 21). Leaflet.js 1.9. https://github.com/Leaflet/Leaflet

B

Baheti, A. & Toshniwal, D. (2014). Trend analysis of time series data using data mining techniques [Abstract], *2014 IEEE International Congress on Big Data*, 430-437, doi: 10.1109/BigData.Congress.2014.69.

Bawaneh M. & Simon, V. (2019). Anomaly detection in smart city traffic based on time series analysis [Abstract], *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 1-6, doi: 10.23919/SOFTCOM.2019.8903822.

Bishop, C. (2006). *Pattern recognition and machine learning.* Springer.

Brownlee, J. (2018, April 27). *How to Calculate Correlation Between Variables in Python.* Machine Learning Mastery. https://machinelearningmastery.com/how-to-use-correlation-to-understand-the-relationship-between-variables/

Brownlee, J. (2020, November 4). *Curve fitting with Python.* Machine Learning Mastery. https://machinelearningmastery.com/curve-fitting-with-python/

C

Chauhan, G. (2021, March 16). *Sklearn svm – Starter Guide* Machine Learning HD. https://machinelearninghd.com/sklearn-svm-starter-guide/

Roesel, D. (2017, October 24). *Curve fitting in Python using SciPy and Matplotlib.* David.roesel.cz. https://david.roesel.cz/notes/posts/curve-fitting-python-scipy-matplotlib/

California Department of Public Health (2022, November 8). *COVID-19 Time-Series Metrics by County and State.* CHHS Open Data. https://data.chhs.ca.gov/dataset/covid-19-time-series-metrics-by-county-and-state

D

Dollar sign icon. Retrieved from October 26, 2021 from https://www.publicdomainpictures.net/en/view-image.php?image=82321&picture=dollar-sign

E

F

Farwaji, S. (2020, May 13). *What is the Rule of 72 in Real Estate.* Mashvisor.com.

https://www.mashvisor.com/blog/rule-of-72/

Ferenti, T. (2017). Biomedical applications of time series analysis [Abstract], *IEEE 30th Neumann Colloquium (NC),* 000083-000084, doi: 10.1109/NC.2017.8263256.

Federal Communications Commission (2016, August 25). *Degrees Minutes Seconds to/from Decimal Degrees.* Retrieved November 8, 2022 from https://www.fcc.gov/media/radio/dms-decimal

G

Ganti, A. (2020, December 28). *Adjusted Closing Price.* Investopedia. https://www.investopedia.com/terms/a/adjusted_closing_price.asp

Ganti, A. (2022, October 5). *What is the Dow Jones Industrial Average (DJIA)?* Investopedia. https://www.investopedia.com/terms/d/djia.asp

Ghanbari R. & Borna K. (2021). Multivariate time-series prediction using LSTM neural networks. *26th International Computer Conference, Computer Society of Iran (CSICC),* 1-5, doi: 10.1109/CSICC52343.2021.9420543.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* MIT Press.

Gupta, N. (2022). *Capital Appreciation.* Wall Street Mojo. https://www.wallstreetmojo.com/capital-appreciation/

H

Hagan, M. T., Demuth, H. B., Beale, M. H., & De Jesús, O. (2014). *Neural Network Design* (2nd ed). Martin T. Hagan and Howard B. Demuth.

Hages, C. (2020, September 26). *How to: Numerical derivative in Python.* YouTube. https://www.youtube.com/watch?v=utRKIlOZbtw

Hall, M. (2022, January 27). *What Does the Dow Jones Industrial Average Measure?* Investopedia. https://www.investopedia.com/ask/answers/050115/what-does-dow-jones-industrial-average-measure.asp

Hayes, A. (2022, June 12) *What Is a Time Series and How Is It Used to Analyze Data?* Investopedia. https://www.investopedia.com/terms/t/timeseries.asp

Historical Inflation Rates: 1914-2022. (2022, November 8) US Inflation Calculator. Retrieved November 8, 2022 from https://www.usinflationcalculator.com/inflation/historical-inflation-rates/

I

Ibrahim, S. A., Charlson, M. E., & Neill, D. B. (2020). Big Data Analytics and the Struggle for Equity in Health Care: The Promise and Perils. *Health equity,* 4(1), 99–101. https://doi.org/10.1089/heq.2019.0112

Investopedia Team. (2020, December 25). *What Is the Housing Bubble? Definition, Causes and Recent Example.* Investopedia.com. https://www.investopedia.com/terms/h/housing_bubble.asp

J

K

Kamangar, F. (2020, September). *CSE 5368 Neural Networks.* CSE Department. The University of Texas at Arlington.

Kasirat, K.T., Hellicar, A., & Rahman A. (2016). Convolutional neural network for time series cattle behaviour classification. *Proceedings of the Workshop on Time Series Analytics and Applications (TSAA '16).* Association for Computing Machinery, New York, NY, USA, 8–12. https://doi-org.ezproxy.uta.edu/10.1145/3014340.3014342

Katarya, R. & Rastogi, S., (2018). A study on neural networks approach to time-series analysis. *2nd International Conference on Inventive Systems and Control (ICISC)*, 116-119, doi: 10.1109/ICISC.2018.8399024.

Kenton, W. *Percentage changes and how to calculate them.* (2022, August 31). Investopedia. https://www.investopedia.com/terms/p/percentage-change.asp

Kremer, H., Gunnemann, S. & Seidl, T. (2010) Detecting Climate Change in Multivariate Time Series Data by Novel Clustering and Cluster Tracing Techniques [Abstract]. *IEEE International Conference on Data Mining Workshops*, 96-97, doi: 10.1109/ICDMW.2010.39.

L

Lazzeri, F. (2020). *Machine learning for time series forecasting with Python*. Hoboken, NJ, United States, Wiley, 2020.

Leusin, Y. (2017, December 17). *How do you calculate rate of return on investment in real estate?* Mashvisor. https://www.mashvisor.com/blog/calculate-rate-of-return-on-investment-real-estate/

Li, L., Huang, S., Ouyang Z., & Li., N. (2022). A deep learning framework for non-stationary time series prediction. *3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications* (CVIDL & ICCEA), 2022, pp. 339-342, doi: 10.1109/CVIDLICCEA56201.2022.9824863.

Lynn, Shane. (2022). *Read CSV data quickly into Pandas DataFrames with read_csv.* Retrieved November 8, 2022 from https://www.shanelynn.ie/python-pandas-read-csv-load-data-from-csv-files/

Lynn, Shane. (2022). *Pandas iloc and loc – quickly select rows and columns in dataframes.* Retrieved November 8, 2022 from https://www.shanelynn.ie/pandas-iloc-loc-select-rows-and-columns-dataframe/

M

Maina, S. (2021, October 31). *Filter a Pandas dataframe by a partial string or pattern in 8 ways.* Towards Data Science. https://towardsdatascience.com/8-ways-to-filter-a-pandas-dataframe-by-a-partial-string-or-pattern-49f43279c50f

May, M & Bart, A. (2022, October 8). *4.5 The second derivative and concavity*. Business Calculus with Excel. https://mathstat.slu.edu/~may/ExcelCalculus/sec-4-5-SecondDerivativeConcavity.html

Moreno, A. (2020, July 20). *Data normalization with Pandas and Scikit-Learn: The complete guide to clean datasets — Part 1*. Towards Data Science https://towardsdatascience.com/data-normalization-with-pandas-and-scikit-learn-7c1cc6ed6475

N

Nielsen, B. (2022, September 3). *What Causes a Real Estate Bubble?* Investopedia.com. https://www.investopedia.com/articles/07/housing_bubble.asp

NOAA. (2022, September 1). *Weather.* National Oceanic and Atmospheric Administration. https://www.noaa.gov/weather.

Ntlangu, M. B., & Baghai-Wadji, A. (2017). Modelling Network Traffic Using Time Series Analysis: A Review [Abstract]. *Proceedings of the International Conference on Big Data and Internet of Thing (BDIOT2017).* Association for Computing Machinery, New York, NY, USA, 209–215. https://doi-org.ezproxy.uta.edu/10.1145/3175684.3175725

O

Olah, C. (2015, August 27). *Understanding LSTM Networks.* Colah's blog. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Oladunni, T., Denis, M., Ososanya, E., & Adesina, J. (2021, May 18). A Time Series Analysis and Forecast of COVID-19 Healthcare Disparity. medRxiv. doi: https://doi.org/10.1101/2021.05.13.21257189

P

Pandas documentation. 2022. *Getting Started.* NumFOCUS, Inc. Retrieved November 8, 2022 from https://pandas.pydata.org/docs/getting_started/intro_tutorials/03_subset_data.html

Pandas documentation. 2022. *Merge, join, concatenate, and compare.* User Guide. NumFOCUS, Inc. Retrieved November 8, 2022 from https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html

Pandas documentation. 2022. *Indexing and selecting data.* User Guide. NumFOCUS, Inc. Retrieved November 8, 2022 from https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html

Pandas documentation. 2022. *pandas.DataFrame.to_numpy.* API reference. NumFOCUS, Inc. Retrieved November 8, 2022 from https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_numpy.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel., M, Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python *JMLR* 12, pp. 2825-

2830, 2011.

Pathak, P. (2020, May 17). *How to generate lat and long coordinates of city without using APIS in Python.* Medium.com. https://medium.com/analytics-vidhya/how-to-generate-lat-and-long-coordinates-of-city-without-using-apis-25ebabcaf1d5

Q

R

Ristanoski, G., Liu, W., & Bailey, J. (2013). A time-dependent enhanced support vector machine for time series regression [Abstract]. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '13).*

Association for Computing Machinery, New York, NY, USA, 946–954. https://doi-org.ezproxy.uta.edu/10.1145/2487575.2487655

Russell, S. J., Norvig, P. (2009). *Artificial intelligence: A modern approach (3rd ed)*. Pearson.

S

Shen, F. K., Zhang W., & Chang, P. (2009). An Engineering Approach to Prediction of Network Traffic Based on Time-Series Model [Abstract]. *2009 International Joint Conference on Artificial Intelligence,* 432-435, doi: 10.1109/JCAI.2009.104.

Shivam, K. (2021, June 29). *Python | Pandas Series.abs()* Geeks for Geeks. https://www.geeksforgeeks.org/python-pandas-series-abs/

Shterev, V. A., Metchkarski, N. S., & Koparanov, K. A. (2022). Time Series Prediction with Neural Networks: a Review. *2022 57th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST),* 1-4, doi: 10.1109/ICEST55168.2022.9828735.

Smith, A. *How to print an entire pandas dataframe in Python.* Retrieved from November 8, 2022 from https://www.adamsmith.haus/python/answers/how-to-print-an-entire-pandas-dataframe-in-python

Stewart, M. (2019, February 26). *Simple Introduction to Convolutional Neural Networks.* Towards Data Science. https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

Stips., A.K., Macias, D., Coughlan, C., and E. Garcia-Gorriz, (2015). Global climate change: Analyzing anthropogenic warming and causality [Abstract], *OCEANS 2015 - Genova, 2015,* 1-4, doi: 10.1109/OCEANS-Genova.2015.7271393.

T

Time Series. (2022, November 8) stats.oecd.org. Retrieved November 8, 2022 from https://stats.oecd.org/glossary/detail.asp?ID=2708

Trend. (2022, November 8) stats.oecd.org. Retrieved November 8, 2022 from https://stats.oecd.org/glossary/detail.asp?ID=2769

The Coding Team. (2019, May 2019). *1.5 Mapping Geolocation with Leaflet.js - Working with Data and APIs in JavaScript.* YouTube. https://www.youtube.com/watch?v=nZaZ2dB6pow

U

U.S. Census Bureau. (2022, May 16) *City and Town Population Totals: 2020-2021.* census.gov https://www.census.gov/data/tables/time-series/demo/popest/2020s-total-cities-and-towns.html

U.S. Const. art. I, § 2.

V

W

Wang, J. (2020, April 5). *How To Analyse A Single Time Series Variable: Time Series Modeling With Python Code.* Towards Data Science. https://towardsdatascience.com/how-to-analyse-a-single-time-series-variable-11dcca7bf16c

Weisstein, E. W. 2022. *Fourier Transform.* From MathWorld--A Wolfram Web Resource. https://mathworld.wolfram.com/FourierTransform.html

Weisstein, E. W. 2022. *L^1-Norm.* From MathWorld--A Wolfram Web Resource. https://mathworld.wolfram.com/L1-Norm.html

Weisstein, E. W. 2022. *L^2-Norm.* From MathWorld--A Wolfram Web Resource.

https://mathworld.wolfram.com/L2-Norm.html

Wu, F., Zhong, Y., & Wu, Q., (2008). Network Threat Frequency Forecasting Based on Fuzzy Time Series and D-S Evidence Theory [Abstract]. *2008 International Conference on Computer Science and Software Engineering*, 754-757, doi: 10.1109/CSSE.2008.284.

Wu, S. (2020, May 23). *3 Best metrics to evaluate regression model?* Towards Data Science. https://towardsdatascience.com/what-are-the-best-metrics-to-evaluate-your-regression-model-418ca481755b

X

Y

Dow Jones Industrial Average (^DJI). (2022, December 1). yahoo! finance. https://finance.yahoo.com/quote/%5EDJI/history/

Z

Zeger, S. L., Irizarry, R., & Peng, R. D. (2006). On time series analysis of public health and biomedical data. Annual review of public health, 27, 57–79. https://doi.org/10.1146/annurev.publhealth.26.021304.144517

Zillow. 2021. *Housing Data: ZHVI Single-Family Homes Time Series Metro & U.S.* [Data set]. Zillow.com. https://www.zillow.com/research/data/

Python Coding References

Retrieved [November 19, 2022] from

http://www2.gcc.edu/dept/math/faculty/BancroftED/buscalc/chapter2/section2-6.php

Retrieved [November 19, 2022] from

https://appdividend.com/2020/11/02/python-string-append/

Retrieved [November 19, 2022] from

https://blog.container-solutions.com/understanding-volumes-docker

Retrieved [November 19, 2022] from

https://bobbyhadz.com/blog/python-valueerror-could-not-convert-string-to-float

Retrieved [November 19, 2022] from

https://boscacci.medium.com/why-and-how-to-make-a-requirements-txt-f329c685181e

Retrieved [November 19, 2022] from

https://colab.research.google.com/drive/1UCJt8EYjlzCs1H1d1X0iDGYJsHKwu-NO#scrollTo=RX2SB_2O1jx7

Retrieved [November 19, 2022] from

https://datascienceparichay.com/article/get-column-names-as-list-in-pandas-dataframe/

Retrieved [November 19, 2022] from

https://datatofish.com/numpy-array-to-pandas-dataframe/

Retrieved [November 19, 2022] from

https://geeksforgeeks.org/python-check-if-a-file-or-directory-exists-2/

Retrieved [November 19, 2022] from

https://gis.stackexchange.com/questions/54405/is-there-a-way-to-populate-a-spreadsheet-of-city-names-with-their-latitudes-and

Retrieved [November 19, 2022] from

https://howtothink.readthedocs.io/en/latest/PvL_H.html

Retrieved [November 19, 2022] from

https://itsmycode.com/python-valueerror-could-not-convert-string-to-float/

Retrieved [November 19, 2022] from

https://numpy.org/doc/stable/reference/generated/numpy.flip.html

Retrieved [November 19, 2022] from

https://numpy.org/doc/stable/reference/generated/numpy.subtract.html

Retrieved [November 19, 2022] from

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.append.html

Retrieved [November 19, 2022] from

https://pandas.pydata.org/docs/reference/api/pandas.Series.divide.html

Retrieved [November 19, 2022] from

https://pandas.pydata.org/docs/reference/api/pandas.Series.idxmin.html

Retrieved [November 19, 2022] from

https://pandas.pydata.org/docs/reference/api/pandas.Series.max.html

Retrieved [November 19, 2022] from

https://pandas.pydata.org/docs/reference/api/pandas.Series.sort_values.html

Retrieved [November 19, 2022] from

https://pandas.pydata.org/docs/reference/api/pandas.Timestamp.html

Retrieved [November 19, 2022] from

https://pynative.com/python-datetime-to-seconds/

Retrieved [November 19, 2022] from

https://pythonguides.com/matplotlib-time-series-plot/

Retrieved [November 19, 2022] from

https://pythonspeed.com/articles/activate-virtualenv-dockerfile/

Retrieved [November 19, 2022] from

https://raw.githubusercontent.com/CodingTrain/Intro-to-Data-APIs-JS/source/module1/03_fetch_json/index.html

Retrieved [November 19, 2022] from

https://realpython.com/numpy-scipy-pandas-correlation-python/

Retrieved [November 19, 2022] from

https://realpython.com/python-wheels/#python-packaging-made-better-an-intro-to-python-wheels

Retrieved [November 19, 2022] from

https://runnable.com/docker/python/dockerize-your-python-application

Retrieved [November 19, 2022] from

https://sparkbyexamples.com/pandas/print-pandas-dataframe-without-index/

Retrieved [November 19, 2022] from

https://stackabuse.com/python-get-number-of-elements-in-a-list/

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/53513/how-do-i-check-if-a-list-is-empty

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/151199/how-to-calculate-number-of-days-between-two-given-dates

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/466345/converting-string-into-datetime

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/493386/how-to-print-without-a-newline-or-space

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/7152762/how-to-redirect-print-output-to-a-file

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/9159757/can-i-add-comments-to-a-pip-requirements-file

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/11285613/selecting-multiple-columns-in-a-pandas-dataframe

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/16476924/how-to-iterate-over-rows-in-a-dataframe-in-pandas

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/17071871/how-do-i-select-rows-from-a-dataframe-based-on-column-values

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/20444087/right-way-to-reverse-a-pandas-dataframe

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/21285380/find-column-whose-name-contains-a-specific-string

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/22341271/get-list-from-pandas-dataframe-column-or-row

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/24101524/finding-median-of-list-in-python

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/25852044/converting-pandas-tslib-timestamp-to-datetime-python

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/31037298/pandas-get-column-average-mean

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/31929538/how-to-subtract-datetimes-timestamps-in-python

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/33246771/convert-pandas-data-frame-to-series

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/33814887/python-pandas-printing-out-values-of-each-cells

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/33834727/how-do-you-extract-only-the-date-from-a-python-datetime

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/37093454/typeerror-must-be-string-not-datetime-datetime-when-using-strptime

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/38387529/how-to-iterate-over-pandas-series-generated-from-groupby-size

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/42286972/converting-from-pandas-dataframe-to-tensorflow-tensor-object

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/56947333/how-to-remove-commas-from-all-the-column-in-pandas-at-once

Retrieved [November 19, 2022] from

https://stackoverflow.com/questions/68357090/getting-an-error-saying-could-not-build-wheels-for-numpy-which-use-pep-517-and

Retrieved [November 19, 2022] from

https://thispointer.com/drop-first-row-of-pandas-dataframe-3-ways/

Retrieved [November 19, 2022] from

https://thispointer.com/how-to-create-and-initialize-a-list-of-lists-in-python/

Retrieved [November 19, 2022] from

https://thispointer.com/python-how-to-get-last-n-characters-in-a-string/

Retrieved [November 19, 2022] from

https://www.activestate.com/resources/quick-reads/how-to-access-an-element-in-pandas/

Retrieved [November 19, 2022] from

https://www.askpython.com/python/string/check-string-contains-substring-python

Retrieved [November 19, 2022] from

https://www.codegrepper.com/code-examples/python/python+color+text+on+windows

Retrieved [November 19, 2022] from

https://www.digitalocean.com/community/tutorials/how-to-build-and-deploy-a-flask-application-using-docker-on-ubuntu-20-04

Retrieved [November 19, 2022] from

https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-18-04

Retrieved [November 19, 2022] from

https://www.freecodecamp.org/news/how-to-substring-a-string-in-python/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/accessing-elements-of-a-pandas-series/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/check-if-element-exists-in-list-in-python/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/convert-the-column-type-from-string-to-datetime-format-in-pandas-dataframe/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/create-a-pandas-dataframe-from-lists/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/different-ways-to-iterate-over-rows-in-pandas-dataframe/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/drop-rows-from-pandas-dataframe-with-missing-values-or-nan-in-columns/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/find-maximum-values-position-in-columns-and-rows-of-a-dataframe-in-pandas/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/floor-ceil-function-python/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/get-column-index-from-column-name-of-a-given-pandas-dataframe/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/how-to-add-one-row-in-an-existing-pandas-dataframe/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/how-to-convert-datetime-to-integer-in-python/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/how-to-get-column-names-in-pandas-dataframe/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/python-broadcasting-with-numpy-arrays/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/python-dictionary/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/python-get-key-from-value-in-dictionary/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/python-numpy-nanmean-function/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/python-program-convert-string-list/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/python-program-to-convert-a-list-to-string/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/python-program-to-sort-a-list-of-tuples-by-second-item/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/python-removing-first-element-of-list/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/python-string-upper/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/sort-in-python/

Retrieved [November 19, 2022] from

https://www.geeksforgeeks.org/ways-to-filter-pandas-dataframe-by-column-values/

Retrieved [November 19, 2022] from

https://www.golinuxcloud.com/pandas-convert-column-to-float/

Retrieved [November 19, 2022] from

https://www.kite.com/python/answers/how-to-find-the-max-value-of-a-pandas-dataframe-column-in-python

Retrieved [November 19, 2022] from

https://www.kite.com/python/answers/how-to-get-the-part-of-a-string-before-a-specific-character-in-python

Retrieved [November 19, 2022] from

https://www.kite.com/python/answers/how-to-print-a-float-with-two-decimal-places-in-python

Retrieved [November 19, 2022] from

https://www.kite.com/python/answers/how-to-write-to-an-html-file-in-python

Retrieved [November 19, 2022] from

https://www.linode.com/docs/guides/how-to-use-dockerfiles/

Retrieved [November 19, 2022] from

https://www.marsja.se/pandas-convert-column-to-datetime/

Retrieved [November 19, 2022] from

https://www.pluralsight.com/guides/how-to-use-gitignore-file

Retrieved [November 19, 2022] from

https://www.programiz.com/python-programming/methods/list/index

Retrieved [November 19, 2022] from

https://www.section.io/engineering-education/how-to-share-data-between-a-docker-container-and-the-host-computer/#step-1-lets-make-a-directory-where-we-will-mount-with-the-container

Retrieved [November 19, 2022] from

https://www.statology.org/pandas-filter-rows-containing-string/

Retrieved [November 19, 2022] from

https://www.tensorflow.org/tutorials/load_data/pandas_dataframe

Retrieved [November 19, 2022] from

https://www.tutorialspoint.com/how-to-set-x-axis-values-in-matplotlib-python

Retrieved [November 19, 2022] from

https://www.w3schools.com/js/js_object_properties.asp

Retrieved [November 19, 2022] from

https://www.w3schools.com/jsref/jsref_if.asp

Retrieved [November 19, 2022] from

https://www.w3schools.com/python/gloss_python_escape_characters.asp

Retrieved [November 19, 2022] from

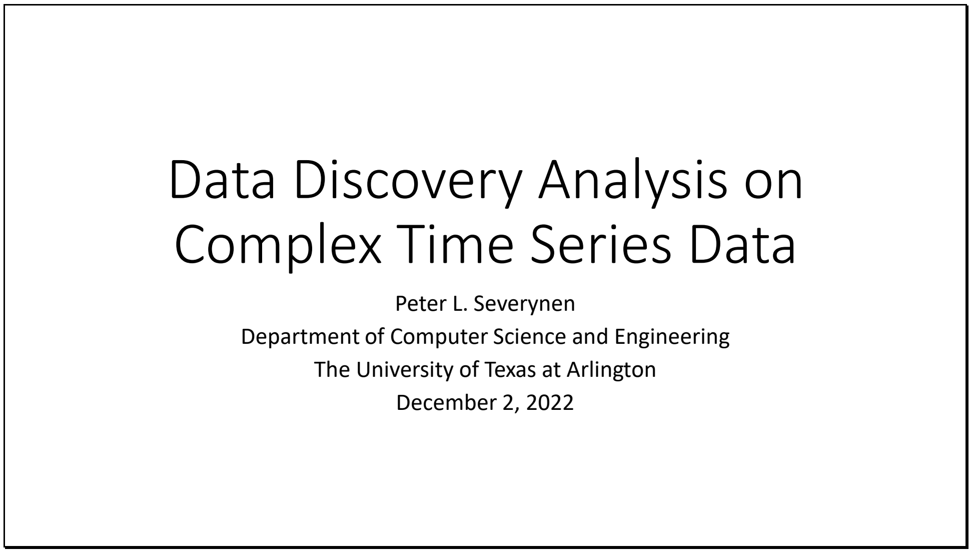
https://www.w3schools.com/python/matplotlib_labels.asp

Retrieved [November 19, 2022] from

https://www.w3schools.com/python/ref_string_split.asp

Appendix I: M.S. Thesis Defense PowerPoint Slides

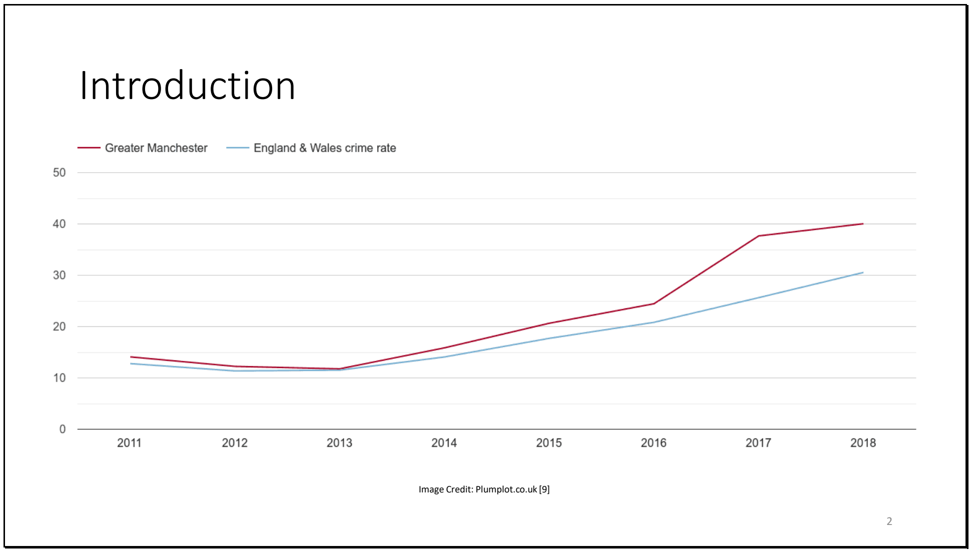
Slide 1



Data Discovery Analysis on Complex Time Series Data

Peter L. Severynen
Department of Computer Science and Engineering
The University of Texas at Arlington
December 2, 2022

Slide 2



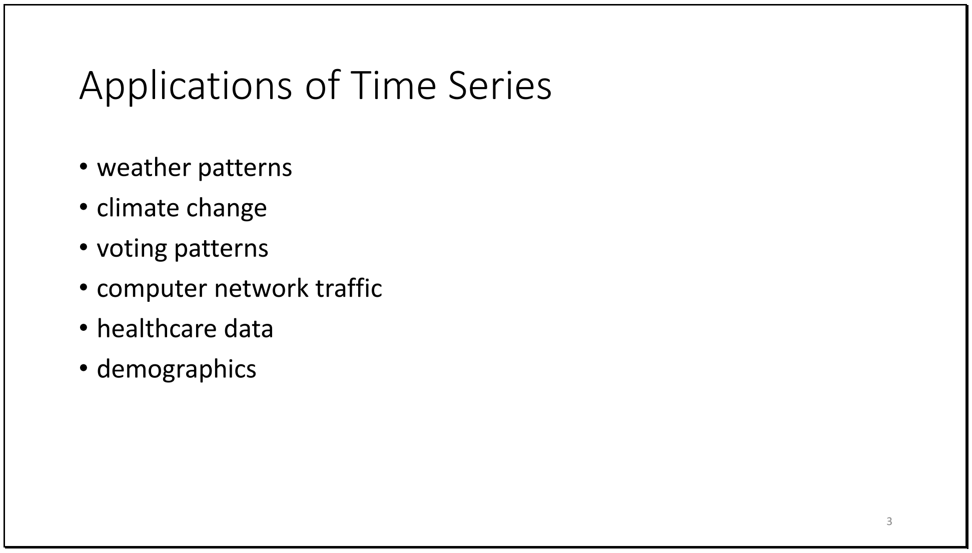
Introduction

Image Credit: Plumplot.co.uk [9]

2

-> Complex time series are a ubiquitous form of data in the modern world. They have wide application across many different fields of scientific inquiry and business endeavor. ->

Slide 3



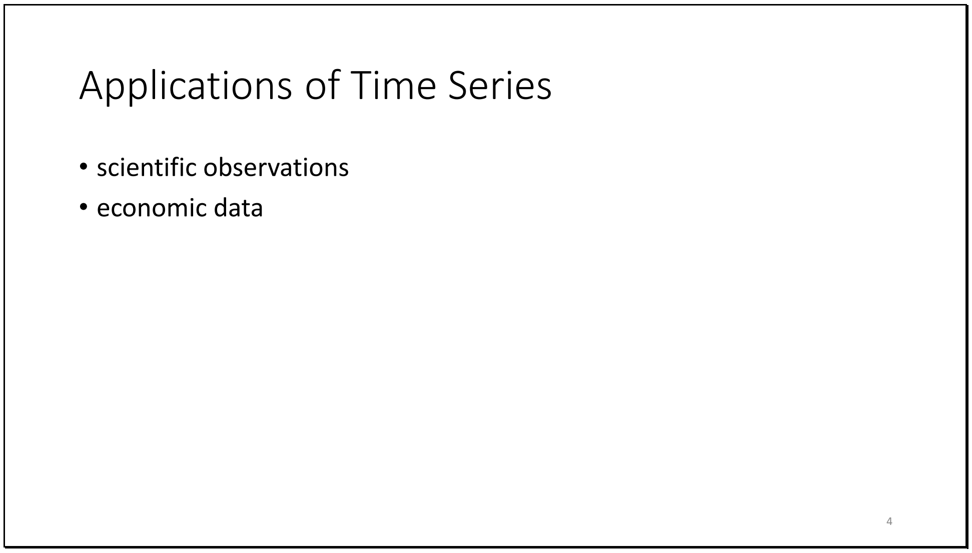
Applications of Time Series

- weather patterns
- climate change
- voting patterns
- computer network traffic
- healthcare data
- demographics

3

Time series are used to understand and forecast -> weather patterns, -> climate change, -> voting patterns, -> computer network traffic, -> population health outcomes, -> demographic changes ->

Slide 4



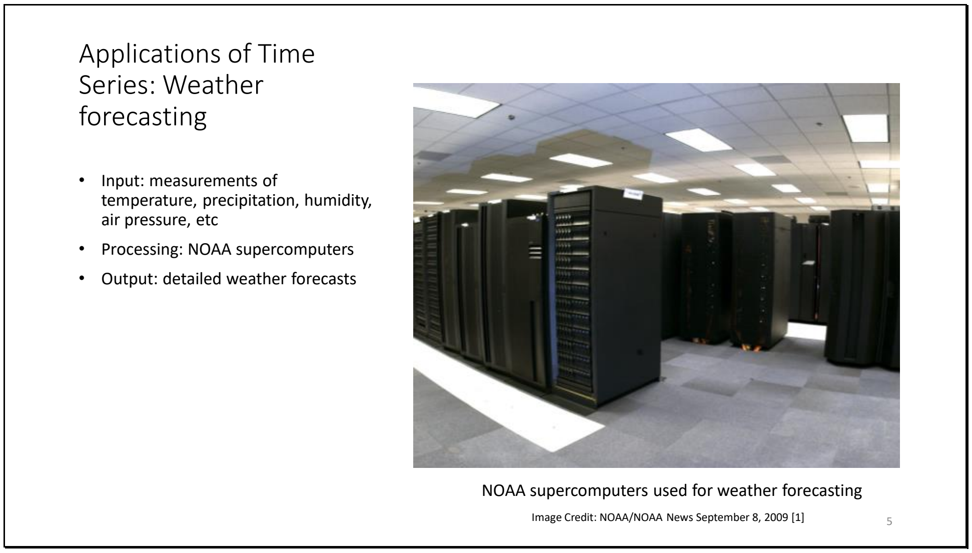
Applications of Time Series

- scientific observations
- economic data

4

-> the results of scientific experiments, and -> the performance of stocks and mutual funds. But time series can be difficult to analyze by conventional methods when the data is multivariate, incomplete, or in different formats. To address these issues, an investigation of several multivariate time series datasets was performed using the methods of automatic data discovery and derivative-based analysis. Interactive maps were constructed which displayed the results of the study. Conclusions were drawn and discussed, and an explanation was given of how this method can be applied to other multivariate time series datasets and real-world problems. ->

Slide 5



Applications of Time Series: Weather forecasting

- Input: measurements of temperature, precipitation, humidity, air pressure, etc
- Processing: NOAA supercomputers
- Output: detailed weather forecasts

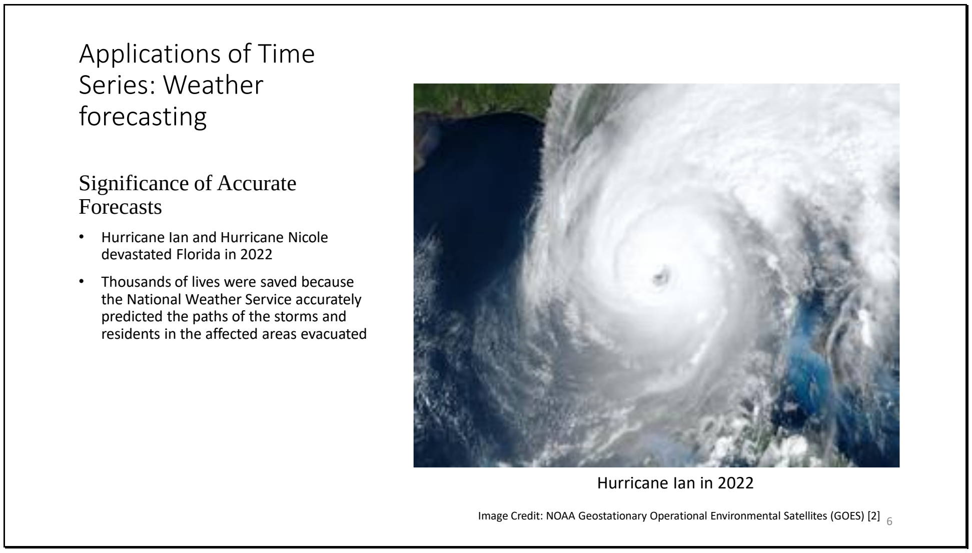
NOAA supercomputers used for weather forecasting

Image Credit: NOAA/NOAA News September 8, 2009 [1]

5

-> This atmospheric data constitutes time series. -> These time series are aggregated and fed as input to supercomputers run by the National Oceanic and Atmospheric Administration (NOAA), a division of the U.S. Department of Commerce. -> These supercomputers generate weather forecasts, but accurate weather forecasts are more than just a convenience.

Slide 6



Applications of Time Series: Weather forecasting

Significance of Accurate Forecasts

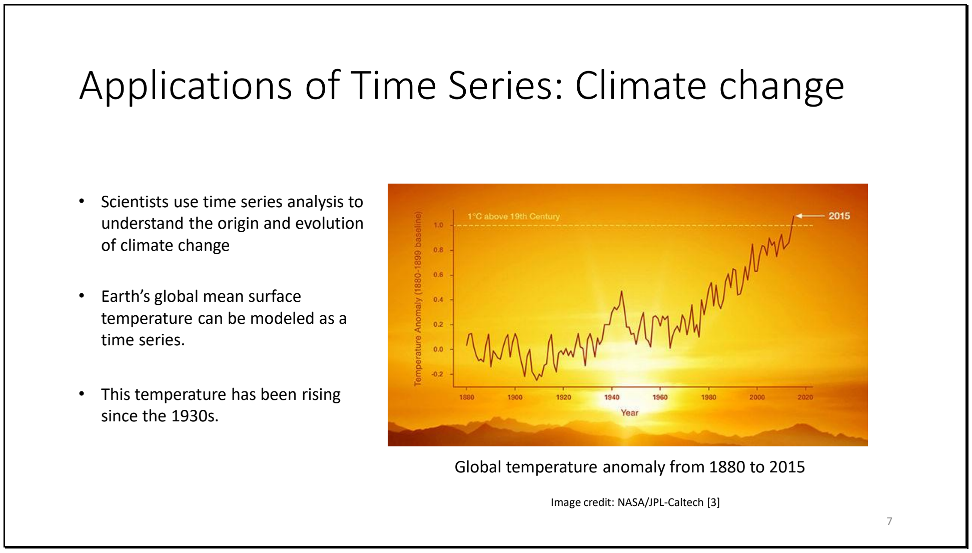
- Hurricane Ian and Hurricane Nicole devastated Florida in 2022
- Thousands of lives were saved because the National Weather Service accurately predicted the paths of the storms and residents in the affected areas evacuated

Hurricane Ian in 2022

Image Credit: NOAA Geostationary Operational Environmental Satellites (GOES) [2] 6

-> Each year, the United States averages some 10,000 thunderstorms, 5,000 floods, 1,300 tornadoes and -> 2 Atlantic hurricanes, as well as widespread droughts and wildfires. Weather, water and climate events, cause an average of approximately 650 deaths and $15 billion in damage per year, and are responsible for some -> 90 percent of all presidentially-declared disasters. About one-third of the U.S. economy – some $3 trillion – is sensitive to weather and climate. The scientific consensus is that extreme weather events such as these are growing in frequency and severity due to climate change. ->

Slide 7



Applications of Time Series: Climate change

- Scientists use time series analysis to understand the origin and evolution of climate change
- Earth's global mean surface temperature can be modeled as a time series.
- This temperature has been rising since the 1930s.

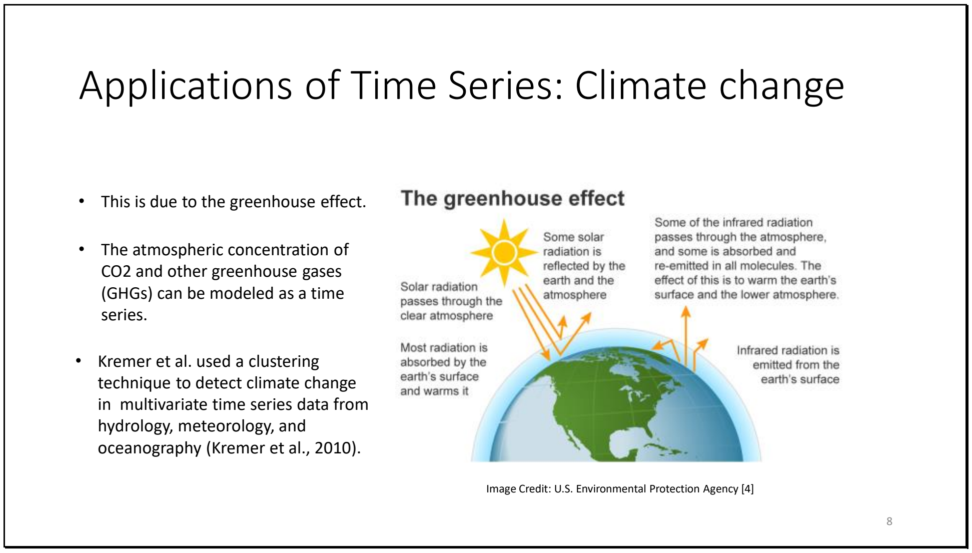
Global temperature anomaly from 1880 to 2015

Image credit: NASA/JPL-Caltech [3]

7

-> -> Climate change can be modeled using time series - >

Slide 8



Applications of Time Series: Climate change

- This is due to the greenhouse effect.
- The atmospheric concentration of CO2 and other greenhouse gases (GHGs) can be modeled as a time series.
- Kremer et al. used a clustering technique to detect climate change in multivariate time series data from hydrology, meteorology, and oceanography (Kremer et al., 2010).

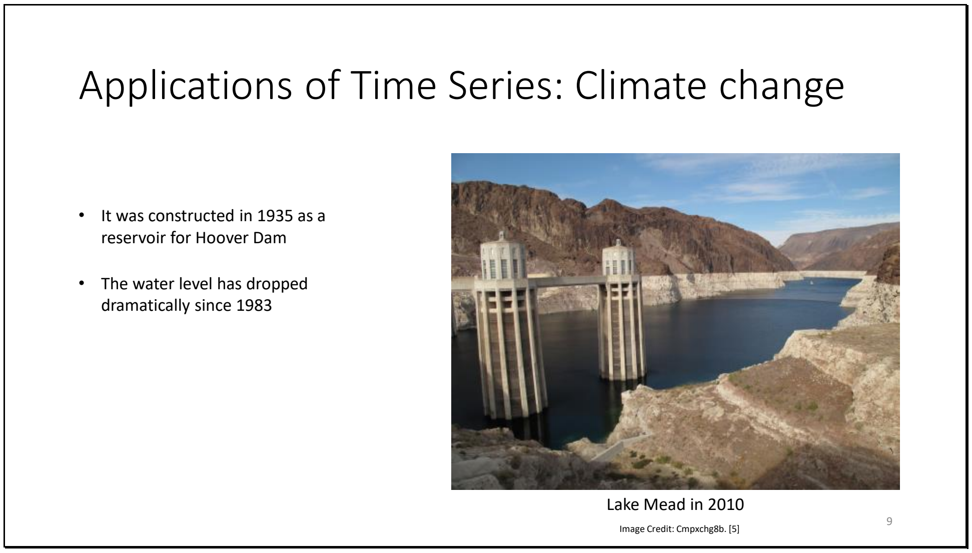
The greenhouse effect
Solar radiation passes through the clear atmosphere
Some solar radiation is reflected by the earth and the atmosphere
Some of the infrared radiation passes through the atmosphere, and some is absorbed and re-emitted in all molecules. The effect of this is to warm the earth's surface and the lower atmosphere.
Most radiation is absorbed by the earth's surface and warms it
Infrared radiation is emitted from the earth's surface

Image Credit: U.S. Environmental Protection Agency [4]

8

-> As human beings pump carbon dioxide gas into the Earth’s atmosphere by burning fossil fuels, -> this carbon dioxide absorbs infrared radiation and re-emits it, -> warming the planet’s surface and increasing the global mean surface temperature. ->

Slide 9



Applications of Time Series: Climate change

- It was constructed in 1935 as a reservoir for Hoover Dam
- The water level has dropped dramatically since 1983

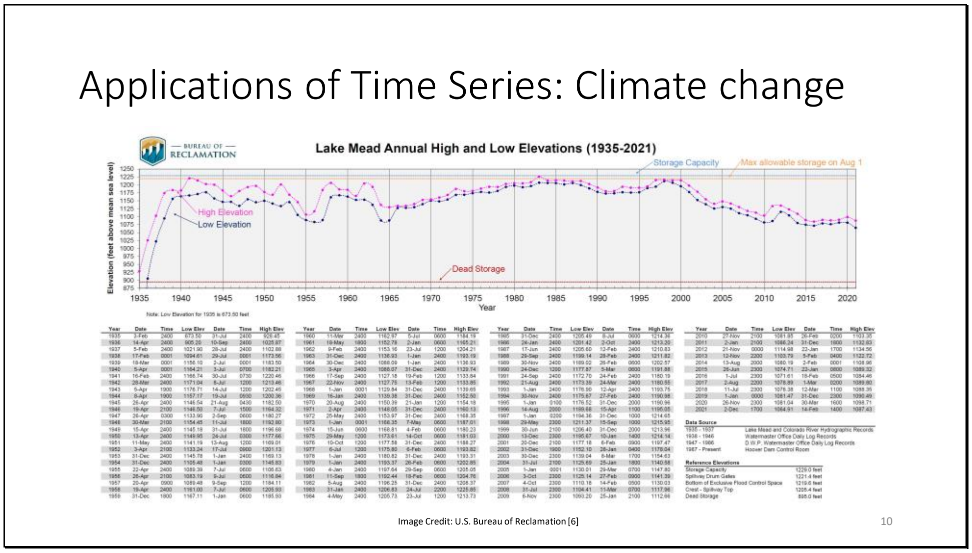
Lake Mead in 2010

Image Credit: Cmpxchg8b. [5]

9

-> This is Lake Mead. -> It shows the visible effects of climate change. -> -> I went there when I was a child, and the water level was above the so-called “bathtub ring” of white mineralized -> rock around the shore of the lake. The water level in Lake Mead may be represented as a time series

Slide 10



Applications of Time Series: Climate change

Note: Low Elevation for 1935 is 673.50 feet

Year	Date	Time	Low Elev	Date	Time	High Elev
1935	2-Feb	2400	673.50	31-Jul	2400	926.45
1936	14-Apr	2400	905.20	10-Sep	2400	1025.87
1937	5-Feb	2400	1021.90	28-Jul	2400	1102.88
1938	17-Feb	0001	1094.61	29-Jul	0001	1173.56
1939	18-Mar	0001	1156.10	2-Jul	0001	1183.50
1940	5-Apr	0001	1164.21	3-Jul	0700	1182.21
1941	16-Feb	2400	1166.74	30-Jul	0730	1220.46
1942	28-Mar	2400	1171.04	8-Jul	1200	1213.46
1943	5-Apr	1900	1176.71	14-Jul	1200	1202.45
1944	8-Apr	1900	1157.17	19-Jul	0930	1200.36
1945	26-Apr	2400	1146.54	21-Aug	0430	1182.50
1946	19-Apr	2100	1146.50	7-Jul	1500	1164.32
1947	26-Apr	0300	1133.90	2-Sep	0600	1180.27
1948	30-Mar	2100	1154.45	11-Jul	1800	1192.80
1949	15-Apr	2400	1145.18	31-Jul	1600	1196.68
1950	13-Apr	2400	1149.95	24-Jul	0300	1177.66
1951	11-May	2400	1141.19	13-Aug	1200	1169.01
1952	3-Apr	2100	1133.24	17-Jul	0900	1201.13
1953	31-Dec	2400	1145.78	1-Jan	2400	1169.13
1954	31-Dec	2400	1105.48	1-Jan	0300	1145.83
1955	22-Apr	2400	1089.39	7-Jul	0600	1106.63
1956	26-Apr	2100	1083.19	9-Jul	0600	1116.84
1957	20-Apr	0900	1089.48	9-Sep	1200	1184.11
1958	19-Apr	2400	1161.00	7-Jul	0600	1205.93
1959	31-Dec	1800	1167.11	1-Jan	0600	1185.93

Year	Date	Time	Low Elev	Date	Time	High Elev
1960	11-Mar	2400	1162.97	5-Jul	0600	1184.19
1961	19-May	1800	1152.78	2-Jan	0600	1165.21
1962	9-Feb	2400	1153.16	23-Jul	1200	1204.21
1963	31-Dec	2400	1136.93	1-Jan	2400	1193.19
1964	30-Dec	2400	1088.09	1-Jan	2400	1136.93
1965	3-Apr	2400	1088.07	31-Dec	2400	1129.74
1966	17-Sep	2400	1127.18	19-Feb	1200	1133.84
1967	22-Nov	2400	1127.75	13-Feb	1200	1133.85
1968	1-Jan	0001	1129.84	31-Dec	2400	1139.65
1969	16-Jan	2400	1139.38	31-Dec	2400	1152.50
1970	20-Aug	2400	1150.39	21-Jan	1200	1154.18
1971	2-Apr	2400	1148.05	31-Dec	2400	1160.13
1972	26-May	2400	1153.97	31-Dec	2400	1168.35
1973	1-Jan	0001	1168.35	7-May	0600	1187.01
1974	15-Jun	0600	1168.81	4-Feb	0600	1180.23
1975	29-May	1200	1173.61	14-Oct	0600	1181.03
1976	10-Oct	1200	1177.58	31-Dec	2400	1188.27
1977	6-Jul	1200	1175.80	6-Feb	0600	1193.82
1978	1-Jan	2400	1180.82	31-Dec	2400	1193.31
1979	1-Jan	2400	1193.37	26-Feb	0600	1202.85
1980	4-Jan	2400	1197.64	29-Sep	0600	1205.05
1981	11-Sep	1800	1192.44	18-Feb	0600	1204.76
1982	5-Aug	2400	1196.25	31-Dec	2400	1208.37
1983	31-Jan	2400	1206.83	24-Jul	2200	1225.85
1984	4-May	2400	1205.73	23-Jul	1200	1213.73

Year	Date	Time	Low Elev	Date	Time	High Elev
1985	31-Dec	2400	1205.49	8-Jul	0600	1214.36
1986	24-Jan	2400	1201.42	2-Oct	2400	1213.20
1987	17-Jun	2400	1205.60	13-Feb	2400	1210.83
1988	29-Sep	2400	1199.14	28-Feb	2400	1211.82
1989	30-Nov	2400	1189.02	26-Feb	0600	1202.57
1990	24-Dec	1200	1177.87	5-Mar	0600	1191.88
1991	24-Sep	2400	1172.70	24-Feb	2400	1180.19
1992	21-Aug	2400	1173.39	24-Mar	2400	1180.55
1993	1-Jan	2400	1176.90	12-Apr	2400	1193.75
1994	30-Nov	2400	1175.67	27-Feb	2400	1190.98
1995	1-Jan	0100	1176.52	31-Dec	2000	1190.96
1996	14-Aug	2000	1189.68	15-Apr	1100	1195.05
1997	1-Jan	0200	1194.36	31-Dec	1000	1214.65
1998	29-May	2300	1211.37	15-Sep	1000	1215.95
1999	30-Jun	2100	1206.40	31-Dec	2000	1213.96
2000	13-Dec	2300	1195.67	10-Jan	1400	1214.14
2001	20-Dec	2100	1177.18	6-Feb	0900	1197.47
2002	31-Dec	1900	1152.10	28-Jan	0400	1178.04
2003	30-Dec	2300	1139.04	8-Mar	1700	1154.63
2004	31-Jul	2100	1125.69	25-Jan	1800	1140.58
2005	1-Jan	0001	1130.01	29-Mar	0700	1147.80
2006	3-Oct	2300	1125.14	27-Feb	0900	1141.39
2007	4-Oct	2300	1110.18	14-Feb	0500	1130.03
2008	31-Jul	2300	1104.41	11-Mar	0700	1117.96
2009	6-Nov	2300	1093.20	25-Jan	2100	1112.66

Year	Date	Time	Low Elev	Date	Time	High Elev
2010	27-Nov	2100	1081.85	26-Feb	0200	1103.35
2011	2-Jan	2100	1086.24	31-Dec	1800	1132.63
2012	21-Nov	0000	1114.98	22-Jan	1700	1134.56
2013	12-Nov	2200	1103.79	5-Feb	0400	1122.72
2014	13-Aug	2000	1080.19	2-Feb	0001	1108.96
2015	26-Jun	2300	1074.71	22-Jan	0800	1089.32
2016	1-Jul	2300	1071.61	18-Feb	0500	1084.46
2017	2-Aug	2200	1078.89	1-Mar	0200	1089.80
2018	11-Jul	2300	1076.38	12-Mar	1100	1088.35
2019	1-Jan	0000	1081.47	31-Dec	2300	1090.49
2020	26-Nov	2300	1081.04	30-Mar	1600	1098.71
2021	2-Dec	1700	1064.91	14-Feb	1400	1087.43

Data Source	
1935 - 1937	Lake Mead and Colorado River Hydrographic Records
1938 - 1946	Watermaster Office Daily Log Records
1947 - 1986	D.W.P. Watermaster Office Daily Log Records
1987 - Present	Hoover Dam Control Room

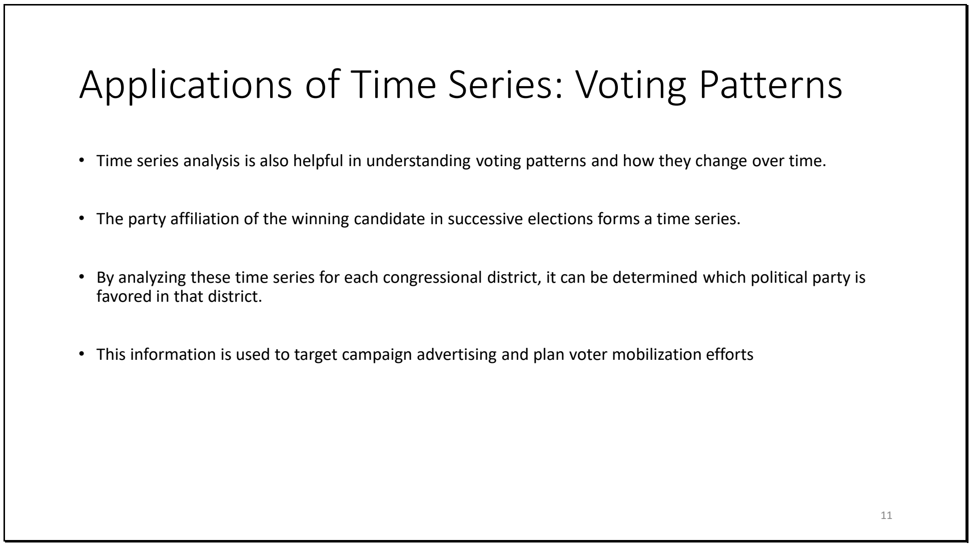
Reference Elevations	
Storage Capacity	1229.0 feet
Spillway Drum Gates	1221.4 feet
Bottom of Exclusive Flood Control Space	1219.6 feet
Crest - Spillway Top	1205.4 feet
Dead Storage	895.0 feet

Image Credit: U.S. Bureau of Reclamation [6]

10

This time series is shown here. ->

Slide 11

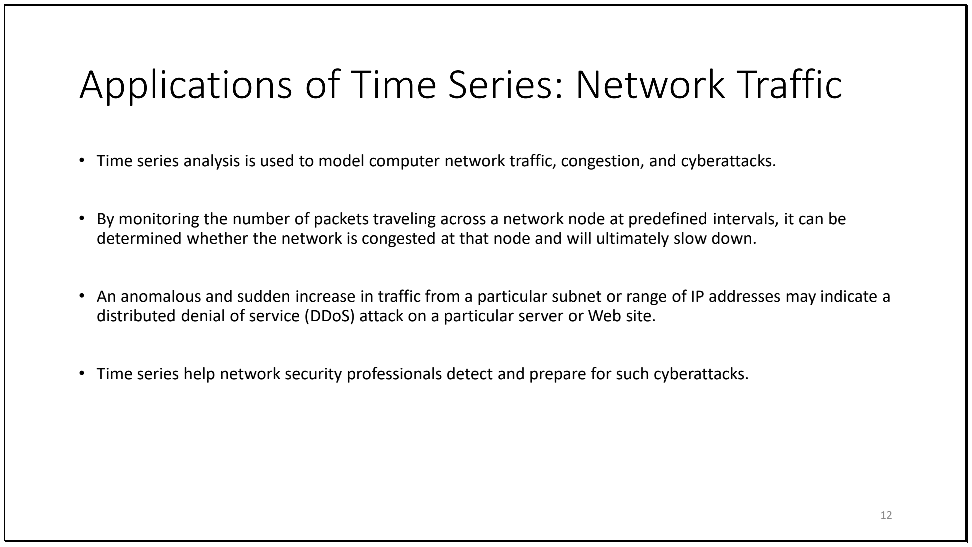


Applications of Time Series: Voting Patterns

- Time series analysis is also helpful in understanding voting patterns and how they change over time.
- The party affiliation of the winning candidate in successive elections forms a time series.
- By analyzing these time series for each congressional district, it can be determined which political party is favored in that district.
- This information is used to target campaign advertising and plan voter mobilization efforts

11

Slide 12



Applications of Time Series: Network Traffic

- Time series analysis is used to model computer network traffic, congestion, and cyberattacks.

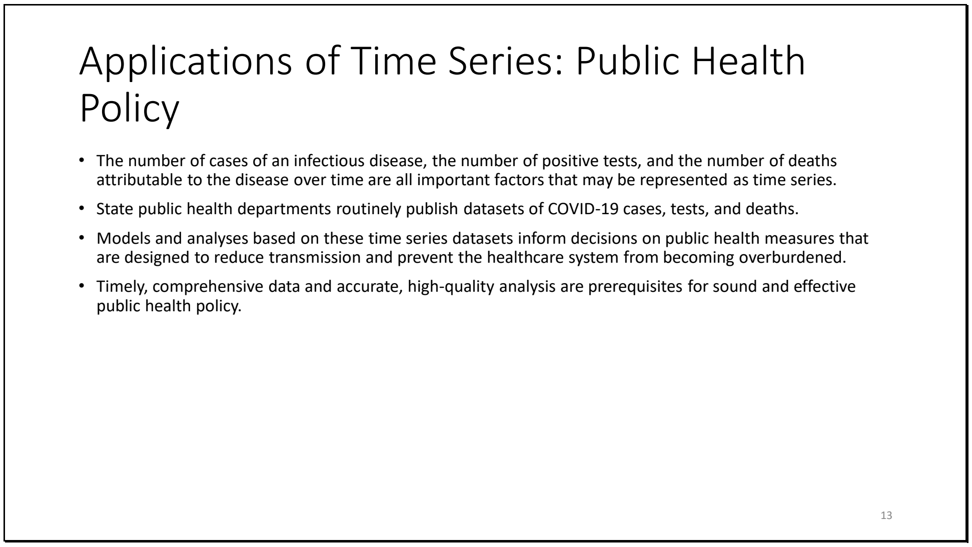
- By monitoring the number of packets traveling across a network node at predefined intervals, it can be determined whether the network is congested at that node and will ultimately slow down.

- An anomalous and sudden increase in traffic from a particular subnet or range of IP addresses may indicate a distributed denial of service (DDoS) attack on a particular server or Web site.

- Time series help network security professionals detect and prepare for such cyberattacks.

12

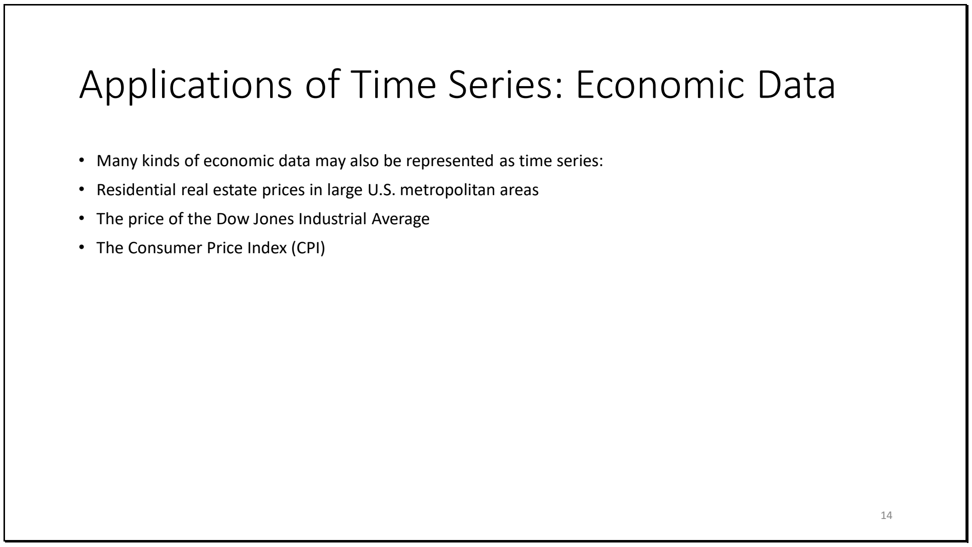
Slide 13



Applications of Time Series: Public Health Policy

- The number of cases of an infectious disease, the number of positive tests, and the number of deaths attributable to the disease over time are all important factors that may be represented as time series.
- State public health departments routinely publish datasets of COVID-19 cases, tests, and deaths.
- Models and analyses based on these time series datasets inform decisions on public health measures that are designed to reduce transmission and prevent the healthcare system from becoming overburdened.
- Timely, comprehensive data and accurate, high-quality analysis are prerequisites for sound and effective public health policy.

Slide 14



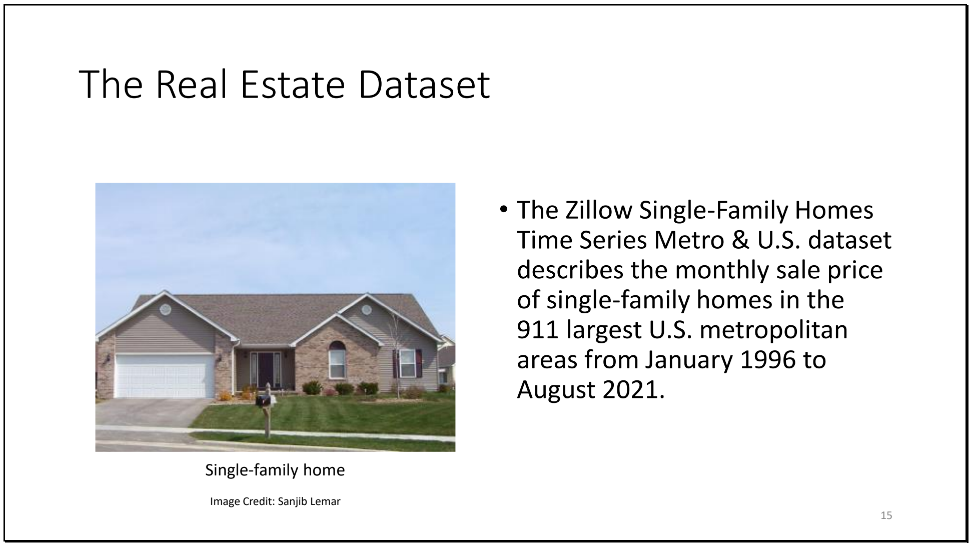
Applications of Time Series: Economic Data

- Many kinds of economic data may also be represented as time series:
- Residential real estate prices in large U.S. metropolitan areas
- The price of the Dow Jones Industrial Average
- The Consumer Price Index (CPI)

14

For this analysis, I will focus on three types of economic data

Slide 15



The Real Estate Dataset

- The Zillow Single-Family Homes Time Series Metro & U.S. dataset describes the monthly sale price of single-family homes in the 911 largest U.S. metropolitan areas from January 1996 to August 2021.

Single-family home

Image Credit: Sanjib Lemar

15

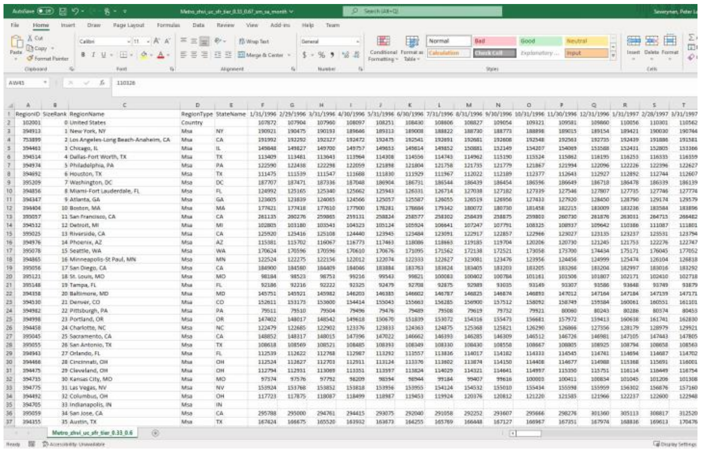
This data was collected by the real estate company Zillow and obtained from its website, zillow.com. A single-family home is a free-standing residential structure designed to be occupied by one group of biologically-related individuals. It is the preferred type of housing for most middle-class and upper-middle class families in the United States.

Slide 16

The Real Estate Dataset

There are five features in this dataset in addition to the monthly prices in the metropolitan areas:

- RegionID
- SizeRank
- RegionName
- RegionType
- StateName

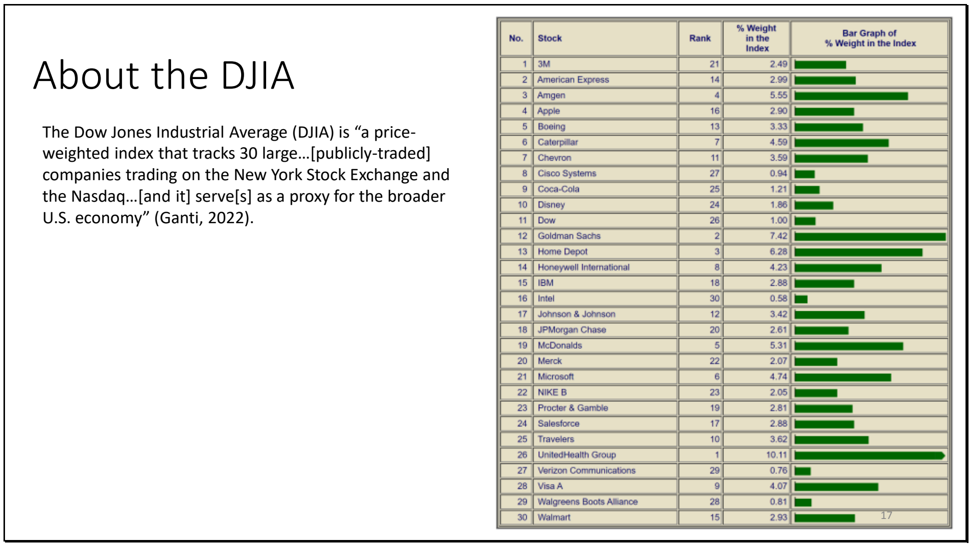


Metro_zhvi_uc_sfr_tier_0.33_0.67_sm_sa_month
Search (Alt+Q)
File Home Insert Draw Page Layout Formulas Data Review View Add-ins Help Team
AW45
110326
Metro_zhvi_uc_sfr_tier_0.33_0.6
Ready
Accessibility: Unavailable
Display Settings

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	RegionID	SizeRank	RegionName	RegionType	StateName	1/31/1996	2/29/1996	3/31/1996	4/30/1996	5/31/1996	6/30/1996	7/31/1996	8/31/1996	9/30/1996	10/31/1996	11/30/1996	12/31/1996	1/31/1997	2/28/1997	3/31/1997
2	102001	0	United States	Country		107872	107904	107960	108097	108251	108430	108806	108827	109054	109321	109581	109860	110056	110301	110562
3	394913	1	New York, NY	Msa	NY	190921	190475	190193	189646	189311	189008	188822	188730	188773	188898	189015	189154	189421	190030	190744
4	753899	2	Los Angeles-Long Beach-Anaheim, CA	Msa	CA	191992	192292	192327	192472	192475	192541	192891	192681	192608	192548	192563	192735	192439	191886	191581
5	394463	3	Chicago, IL	Msa	IL	149848	149827	149700	149757	149653	149614	149852	150881	152149	154207	154089	153588	152431	152805	153166
6	394514	4	Dallas-Fort Worth, TX	Msa	TX	113409	113481	113643	113964	114308	114556	114743	114962	115190	115524	115862	116195	116253	116335	116359
7	394974	5	Philadelphia, PA	Msa	PA	122590	122438	122298	122059	121898	121804	121758	121735	121779	121867	121994	122096	122226	122396	122627
8	394692	6	Houston, TX	Msa	TX	111475	111539	111547	111688	111830	111929	111967	112022	112189	112377	112643	112927	112892	112744	112607
9	395209	7	Washington, DC	Msa	DC	187707	187471	187336	187048	186904	186731	186544	186439	186454	186596	186649	186718	186478	186339	186139
10	394856	8	Miami-Fort Lauderdale, FL	Msa	FL	124992	125165	125340	125662	125943	126331	126714	127038	127182	127339	127546	127807	127735	127746	127774
11	394347	9	Atlanta, GA	Msa	GA	123605	123839	124065	124566	125057	125587	126055	126519	126956	127433	127920	128450	128790	129174	129579
12	394404	10	Boston, MA	Msa	MA	177421	177418	177610	177900	178281	178684	179342	180072	180730	181458	182215	183009	183236	183584	183896
13	395057	11	San Francisco, CA	Msa	CA	261135	260276	259865	259131	258824	258577	258302	258439	258875	259803	260730	261876	263031	264715	268482
14	394532	12	Detroit, MI	Msa	MI	102805	103180	103543	104323	105124	105924	106641	107247	107791	108325	108937	109642	110386	111087	111801
15	395025	13	Riverside, CA	Msa	CA	125920	125416	125108	124440	123945	123484	123091	122917	122857	122966	123027	123135	123237	123531	123794
16	394976	14	Phoenix, AZ	Msa	AZ	115381	115702	116067	116773	117463	118086	118663	119185	119704	120206	120730	121245	121753	122276	122747
17	395078	15	Seattle, WA	Msa	WA	170624	170596	170596	170610	170676	171095	171562	172138	172521	173058	173700	174434	175171	176045	177052
18	394865	16	Minneapolis-St Paul, MN	Msa	MN	122524	122275	122156	122012	122074	122333	122627	123081	123476	123956	124456	124999	125474	126104	126818
19	395056	17	San Diego, CA	Msa	CA	184900	184580	184409	184046	183884	183763	183624	183405	183203	183205	183266	183204	182997	183016	183292
20	395121	18	St. Louis, MO	Msa	MO	98184	98523	98753	99216	99543	99821	100083	100402	100784	101161	101506	101807	102171	102410	102718
21	395148	19	Tampa, FL	Msa	FL	92186	92216	92222	92325	92479	92708	92875	92989	93035	93149	93307	93586	93848	93749	93879
22	394358	20	Baltimore, MD	Msa	MD	145751	145921	145982	146203	146385	146602	146787	146825	146874	146893	147012	147164	147184	147139	147171
23	394530	21	Denver, CO	Msa	CO	152611	153173	153800	154414	155043	155663	156285	156900	157512	158092	158749	159384	160061	160551	161101
24	394982	22	Pittsburgh, PA	Msa	PA	79511	79510	79504	79496	79476	79489	79508	79619	79732	79921	80060	80243	80286	80374	80453
25	394998	23	Portland, OR	Msa	OR	147402	148017	148542	149618	150670	151839	153072	154316	155473	156681	157972	159413	160638	161741	162830
26	394458	24	Charlotte, NC	Msa	NC	122479	122685	122902	123376	123833	124363	124875	125368	125821	126290	126866	127356	128179	128979	129921
27	395045	25	Sacramento, CA	Msa	CA	148852	148317	148015	147396	147022	146662	146393	146285	146309	146512	146726	146981	147105	147443	147805
28	395055	26	San Antonio, TX	Msa	TX	108618	108569	108521	108485	108393	108349	108330	108430	108558	108667	108805	108925	108794	108658	108563
29	394943	27	Orlando, FL	Msa	FL	112539	112622	112768	112987	113292	113557	113836	114017	114182	114333	114545	114741	114694	114687	114702
30	394466	28	Cincinnati, OH	Msa	OH	112524	112627	112703	112911	113124	113376	113602	113874	114150	114408	114677	114988	115368	115691	116001
31	394475	29	Cleveland, OH	Msa	OH	112794	112931	113069	113351	113597	113824	114029	114321	114641	114997	115350	115751	116114	116449	116754
32	394735	30	Kansas City, MO	Msa	MO	97374	97576	97792	98209	98594	98944	99184	99407	99616	100003	100411	100834	101045	101206	101308
33	394775	31	Las Vegas, NV	Msa	NV	153924	153768	153852	153818	153956	153955	154124	154532	155010	155434	155598	155959	156302	156876	157160
34	394492	32	Columbus, OH	Msa	OH	117723	117875	118087	118499	118987	119453	119924	120376	120812	121220	121585	121966	122237	122600	122948
35	394705	33	Indianapolis, IN	Msa	IN															
36	395059	34	San Jose, CA	Msa	CA	295788	295000	294761	294415	293075	292040	291058	292252	293607	295666	298276	301360	305113	308817	312520
37	394355	35	Austin, TX	Msa	TX	167424	166675	165520	163932	163673	164255	165769	166448	167127	166967	167351	167974	168836	169613	170476

-> -> RegionID is a unique identifier assigned to each metropolitan area. -> SizeRank is the relative population ranking of the metropolitan area in the dataset. -> RegionName is the name of the metropolitan area. A metropolitan area may consist of one city or multiple cities in the same geographical area. -> RegionType shows whether the data in that row is for the whole country or a single metropolitan statistical area (MSA). -> StateName is the name of the state where the metropolitan area is located. The median price of a single-family home in that metropolitan area each month is listed in the columns on the right. ->

Slide 17



About the DJIA

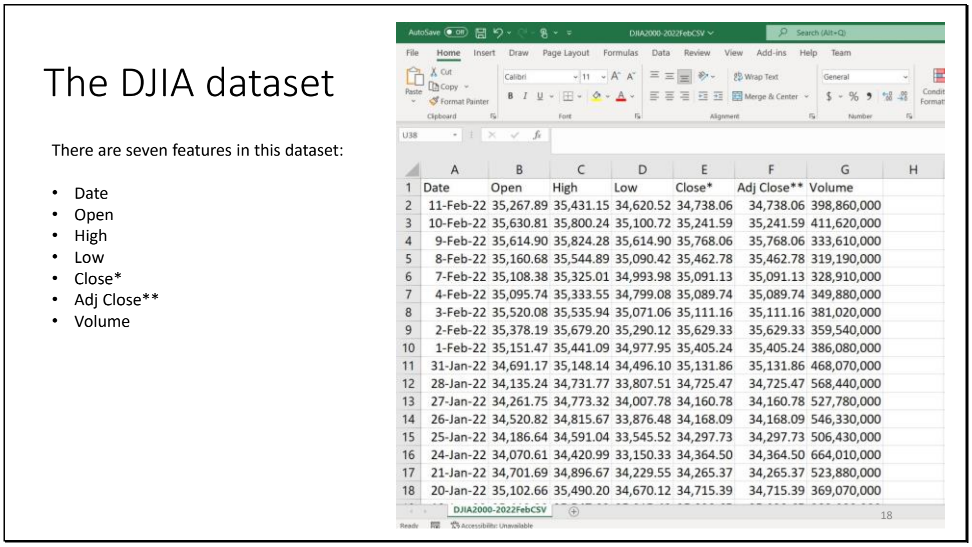
The Dow Jones Industrial Average (DJIA) is "a price-weighted index that tracks 30 large…[publicly-traded] companies trading on the New York Stock Exchange and the Nasdaq…[and it] serve[s] as a proxy for the broader U.S. economy" (Ganti, 2022).

No.	Stock	Rank	% Weight in the Index	Bar Graph of % Weight in the Index
1	3M	21	2.49	
2	American Express	14	2.99	
3	Amgen	4	5.55	
4	Apple	16	2.90	
5	Boeing	13	3.33	
6	Caterpillar	7	4.59	
7	Chevron	11	3.59	
8	Cisco Systems	27	0.94	
9	Coca-Cola	25	1.21	
10	Disney	24	1.86	
11	Dow	26	1.00	
12	Goldman Sachs	2	7.42	
13	Home Depot	3	6.28	
14	Honeywell International	8	4.23	
15	IBM	18	2.88	
16	Intel	30	0.58	
17	Johnson & Johnson	12	3.42	
18	JPMorgan Chase	20	2.61	
19	McDonalds	5	5.31	
20	Merck	22	2.07	
21	Microsoft	6	4.74	
22	NIKE B	23	2.05	
23	Procter & Gamble	19	2.81	
24	Salesforce	17	2.88	
25	Travelers	10	3.62	
26	UnitedHealth Group	1	10.11	
27	Verizon Communications	29	0.76	
28	Visa A	9	4.07	
29	Walgreens Boots Alliance	28	0.81	
30	Walmart	15	2.93	

17

Here we see the current components of the Dow Jones Industrial Average and their weights in the index. The three largest components by weight are: UnitedHealth Group (10.11%), Goldman Sachs (7.42%), and Home Depot (6.28%). Dow Chemical is 1% of the index. These weights are assigned and regularly updated by Dow Jones and Company, Inc, which publishes the Dow Jones Industrial Average.

Slide 18



The DJIA dataset

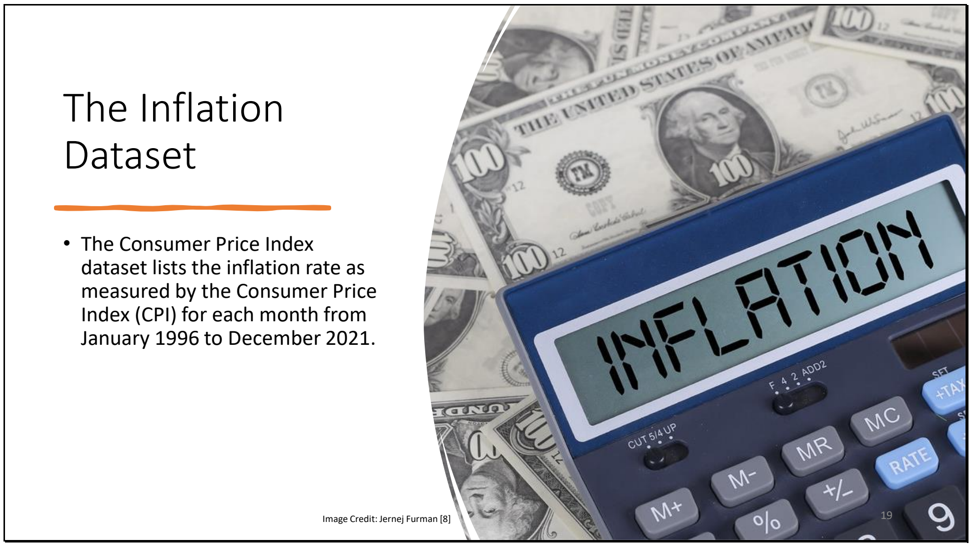
There are seven features in this dataset:

- Date
- Open
- High
- Low
- Close*
- Adj Close**
- Volume

	A	B	C	D	E	F	G	H
1	Date	Open	High	Low	Close*	Adj Close**	Volume	
2	11-Feb-22	35,267.89	35,431.15	34,620.52	34,738.06	34,738.06	398,860,000	
3	10-Feb-22	35,630.81	35,800.24	35,100.72	35,241.59	35,241.59	411,620,000	
4	9-Feb-22	35,614.90	35,824.28	35,614.90	35,768.06	35,768.06	333,610,000	
5	8-Feb-22	35,160.68	35,544.89	35,090.42	35,462.78	35,462.78	319,190,000	
6	7-Feb-22	35,108.38	35,325.01	34,993.98	35,091.13	35,091.13	328,910,000	
7	4-Feb-22	35,095.74	35,333.55	34,799.08	35,089.74	35,089.74	349,880,000	
8	3-Feb-22	35,520.08	35,535.94	35,071.06	35,111.16	35,111.16	381,020,000	
9	2-Feb-22	35,378.19	35,679.20	35,290.12	35,629.33	35,629.33	359,540,000	
10	1-Feb-22	35,151.47	35,441.09	34,977.95	35,405.24	35,405.24	386,080,000	
11	31-Jan-22	34,691.17	35,148.14	34,496.10	35,131.86	35,131.86	468,070,000	
12	28-Jan-22	34,135.24	34,731.77	33,807.51	34,725.47	34,725.47	568,440,000	
13	27-Jan-22	34,261.75	34,773.32	34,007.78	34,160.78	34,160.78	527,780,000	
14	26-Jan-22	34,520.82	34,815.67	33,876.48	34,168.09	34,168.09	546,330,000	
15	25-Jan-22	34,186.64	34,591.04	33,545.52	34,297.73	34,297.73	506,430,000	
16	24-Jan-22	34,070.61	34,420.99	33,150.33	34,364.50	34,364.50	664,010,000	
17	21-Jan-22	34,701.69	34,896.67	34,229.55	34,265.37	34,265.37	523,880,000	
18	20-Jan-22	35,102.66	35,490.20	34,670.12	34,715.39	34,715.39	369,070,000	

-> Date is the trading day on which this data was recorded. -> Open is the price of the DJIA at the opening of the stock exchange on that trading day. -> High is the highest price that it saw on that trading day. -> Low is the lowest price that it saw on that trading day. -> Close* is the price of the DJIA when the market closed, not accounting for any corporate actions, such as stock splits. -> Adjusted Close is the price after accounting for corporate actions. -> Volume is the number of shares traded that day. ->

Slide 19



The Inflation Dataset

- The Consumer Price Index dataset lists the inflation rate as measured by the Consumer Price Index (CPI) for each month from January 1996 to December 2021.

Image Credit: Jernej Furman [8]

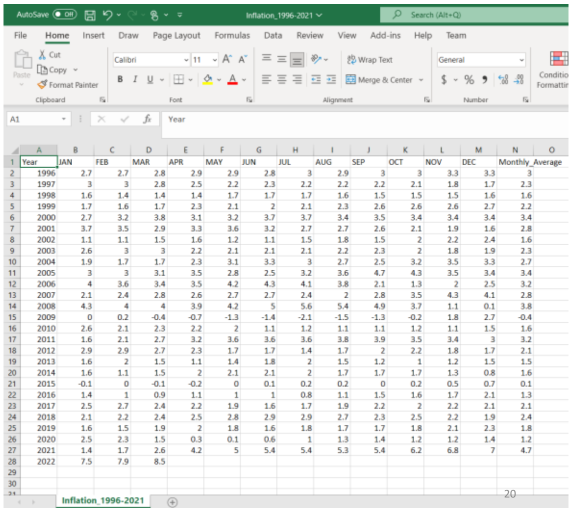
19

Inflation has hit record highs for the past two years and has become a major political issue.

Slide 20

The Consumer Price Index (CPI) Dataset

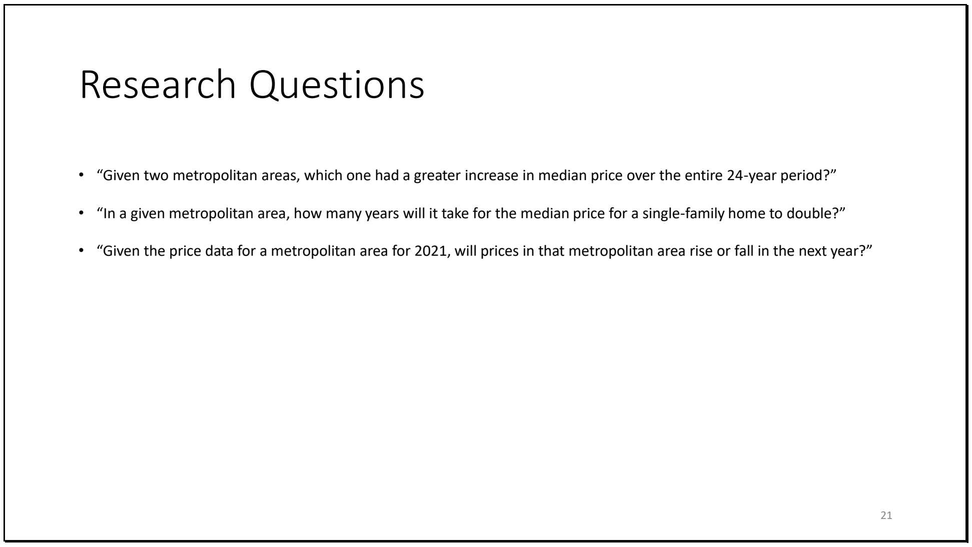
- The inflation rate is given as a percentage change in prices for each month in the dataset.
- The *Monthly_Average* column shows the average inflation rate for each year



Year	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	Monthly_Average
1996	2.7	2.7	2.8	2.9	2.9	2.8	3	2.9	3	3	3.3	3.3	3
1997	3	3	2.8	2.5	2.2	2.3	2.2	2.2	2.2	2.1	1.8	1.7	2.3
1998	1.6	1.4	1.4	1.4	1.7	1.7	1.7	1.6	1.5	1.5	1.5	1.6	1.6
1999	1.7	1.6	1.7	2.3	2.1	2	2.1	2.3	2.6	2.6	2.6	2.7	2.2
2000	2.7	3.2	3.8	3.1	3.2	3.7	3.7	3.4	3.5	3.4	3.4	3.4	3.4
2001	3.7	3.5	2.9	3.3	3.6	3.2	2.7	2.7	2.6	2.1	1.9	1.6	2.8
2002	1.1	1.1	1.5	1.6	1.2	1.1	1.5	1.8	1.5	2	2.2	2.4	1.6
2003	2.6	3	3	2.2	2.1	2.1	2.1	2.2	2.3	2	1.8	1.9	2.3
2004	1.9	1.7	1.7	2.3	3.1	3.3	3	2.7	2.5	3.2	3.5	3.3	2.7
2005	3	3	3.1	3.5	2.8	2.5	3.2	3.6	4.7	4.3	3.5	3.4	3.4
2006	4	3.6	3.4	3.5	4.2	4.3	4.1	3.8	2.1	1.3	2	2.5	3.2
2007	2.1	2.4	2.8	2.6	2.7	2.7	2.4	2	2.8	3.5	4.3	4.1	2.8
2008	4.3	4	4	3.9	4.2	5	5.6	5.4	4.9	3.7	1.1	0.1	3.8
2009	0	0.2	-0.4	-0.7	-1.3	-1.4	-2.1	-1.5	-1.3	-0.2	1.8	2.7	-0.4
2010	2.6	2.1	2.3	2.2	2	1.1	1.2	1.1	1.1	1.2	1.1	1.5	1.6
2011	1.6	2.1	2.7	3.2	3.6	3.6	3.6	3.8	3.9	3.5	3.4	3	3.2
2012	2.9	2.9	2.7	2.3	1.7	1.7	1.4	1.7	2	2.2	1.8	1.7	2.1
2013	1.6	2	1.5	1.1	1.4	1.8	2	1.5	1.2	1	1.2	1.5	1.5
2014	1.6	1.1	1.5	2	2.1	2.1	2	1.7	1.7	1.7	1.3	0.8	1.6
2015	-0.1	0	-0.1	-0.2	0	0.1	0.2	0.2	0	0.2	0.5	0.7	0.1
2016	1.4	1	0.9	1.1	1	1	0.8	1.1	1.5	1.6	1.7	2.1	1.3
2017	2.5	2.7	2.4	2.2	1.9	1.6	1.7	1.9	2.2	2	2.2	2.1	2.1
2018	2.1	2.2	2.4	2.5	2.8	2.9	2.9	2.7	2.3	2.5	2.2	1.9	2.4
2019	1.6	1.5	1.9	2	1.8	1.6	1.8	1.7	1.7	1.8	2.1	2.3	1.8
2020	2.5	2.3	1.5	0.3	0.1	0.6	1	1.3	1.4	1.2	1.2	1.4	1.2
2021	1.4	1.7	2.6	4.2	5	5.4	5.4	5.3	5.4	6.2	6.8	7	4.7
2022	7.5	7.9	8.5										

20

Slide 21



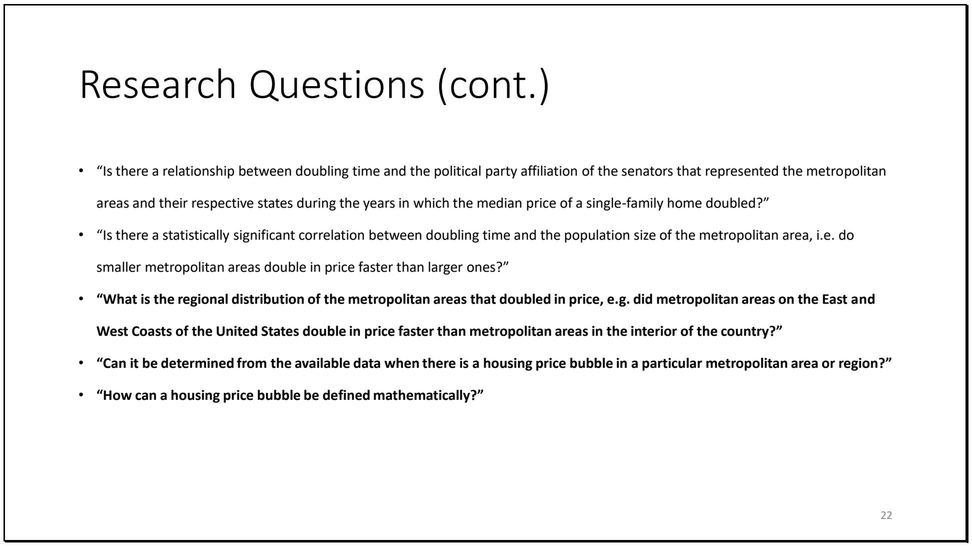
Research Questions

- "Given two metropolitan areas, which one had a greater increase in median price over the entire 24-year period?"
- "In a given metropolitan area, how many years will it take for the median price for a single-family home to double?"
- "Given the price data for a metropolitan area for 2021, will prices in that metropolitan area rise or fall in the next year?"

21

Several research questions were posed for this analysis, but I will focus on the last three.

Slide 22



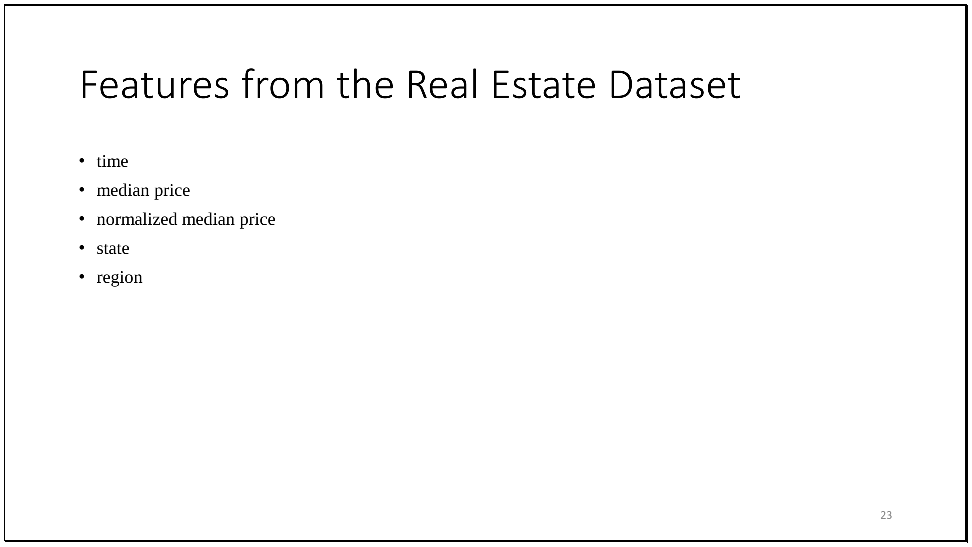
Research Questions (cont.)

- "Is there a relationship between doubling time and the political party affiliation of the senators that represented the metropolitan areas and their respective states during the years in which the median price of a single-family home doubled?"
- "Is there a statistically significant correlation between doubling time and the population size of the metropolitan area, i.e. do smaller metropolitan areas double in price faster than larger ones?"
- **"What is the regional distribution of the metropolitan areas that doubled in price, e.g. did metropolitan areas on the East and West Coasts of the United States double in price faster than metropolitan areas in the interior of the country?"**
- **"Can it be determined from the available data when there is a housing price bubble in a particular metropolitan area or region?"**
- **"How can a housing price bubble be defined mathematically?"**

22

These three questions are important because they give insight into the regional trends that were occurring in the U.S. residential housing market at that time. Each method I tested made a unique contribution to the overall analysis. First, I implemented the following algorithms to answer the specific questions, and these gave me insight into the general trends.

Slide 23



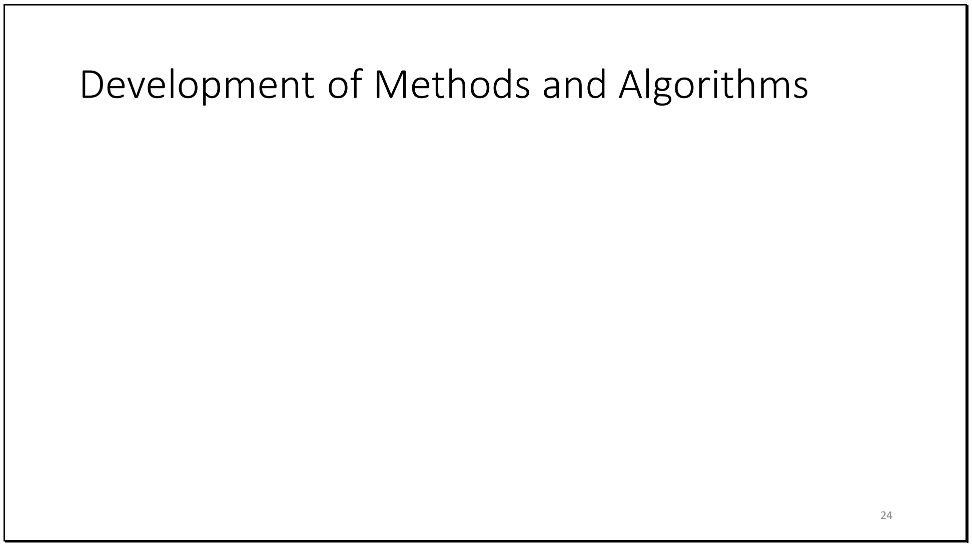
Features from the Real Estate Dataset

- time
- median price
- normalized median price
- state
- region

23

These features were chosen to compare metropolitan areas from the real estate dataset because they were the most informative. Normalized median price and region are derived features.

Slide 24

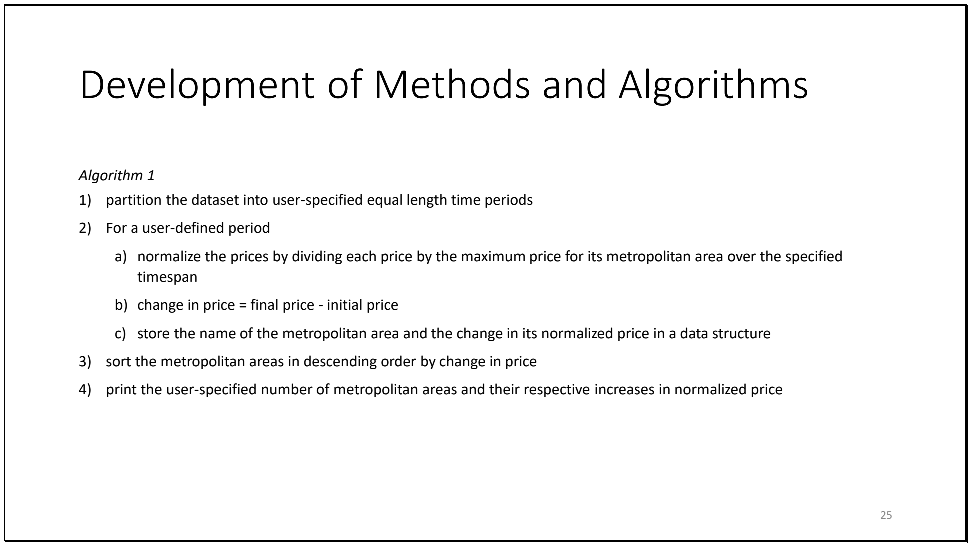


Development of Methods and Algorithms

24

In order to answer the research questions, several methods and algorithms were developed.

Slide 25



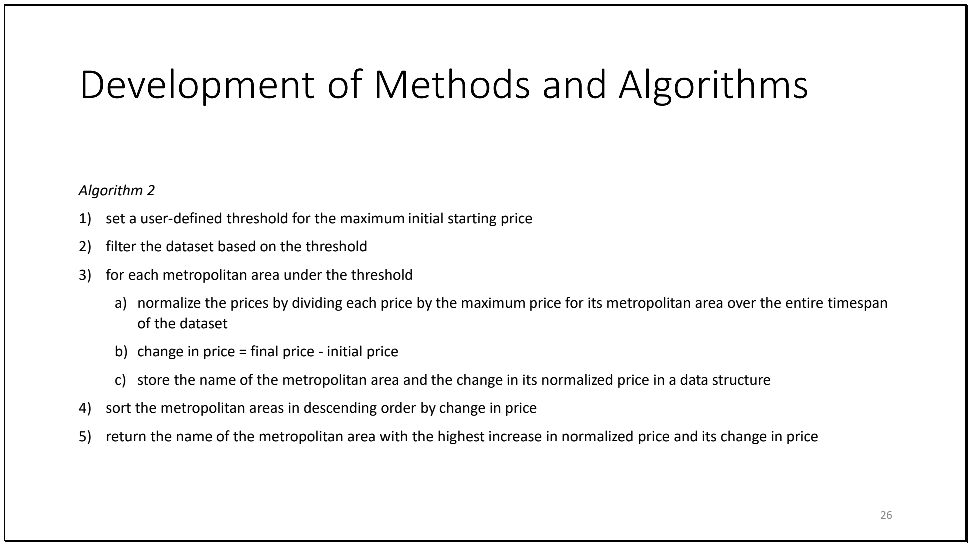
Development of Methods and Algorithms

Algorithm 1

```
1) partition the dataset into user-specified equal length time periods
2) For a user-defined period
    a) normalize the prices by dividing each price by the maximum price for its metropolitan area over the specified timespan
    b) change in price = final price - initial price
    c) store the name of the metropolitan area and the change in its normalized price in a data structure
3) sort the metropolitan areas in descending order by change in price
4) print the user-specified number of metropolitan areas and their respective increases in normalized price
```

-> In order to determine which metropolitan areas appreciated the fastest in a given time period, the following algorithm was developed. -> I partitioned the dataset into time periods. -> I normalized the prices in each time period. -> I computed the change in price -> I sorted the metropolitan areas by the change in price. -> And I printed the results for the user. ->

Slide 26



Development of Methods and Algorithms

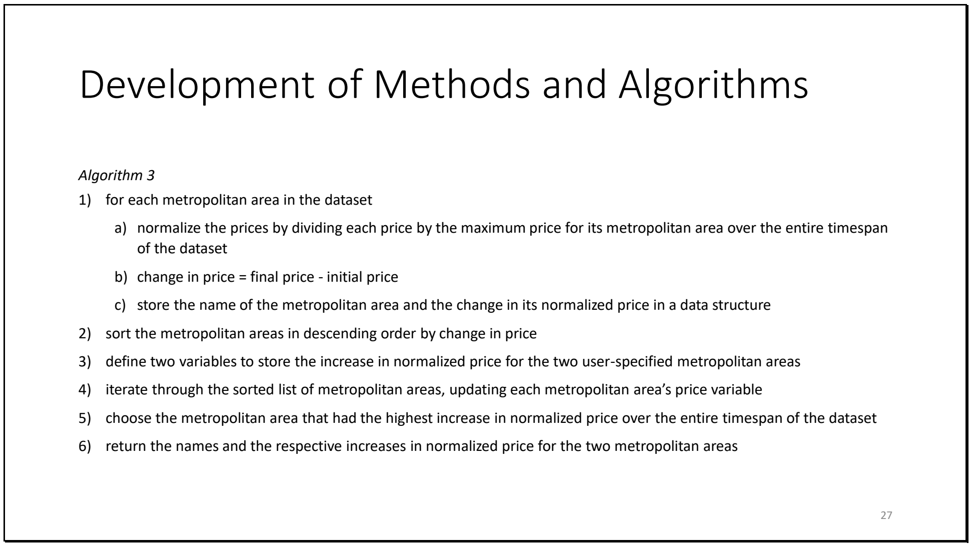
Algorithm 2

1) set a user-defined threshold for the maximum initial starting price
2) filter the dataset based on the threshold
3) for each metropolitan area under the threshold
    a) normalize the prices by dividing each price by the maximum price for its metropolitan area over the entire timespan of the dataset
    b) change in price = final price - initial price
    c) store the name of the metropolitan area and the change in its normalized price in a data structure
4) sort the metropolitan areas in descending order by change in price
5) return the name of the metropolitan area with the highest increase in normalized price and its change in price

26

-> To find which city offered the best return on investment over the timespan of the real estate dataset, -> I filtered the cities based on the desired investment amount. -> I normalized the prices. -> I computed the change in price -> I sorted the metropolitan areas, by the change in price. -> And I printed the results to the user. ->

Slide 27



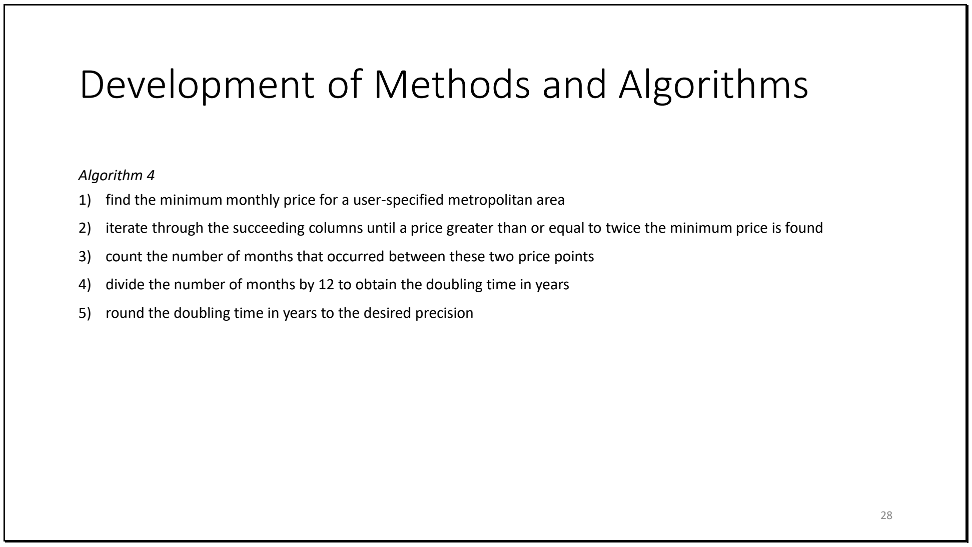
Development of Methods and Algorithms

Algorithm 3

```
1) for each metropolitan area in the dataset
    a) normalize the prices by dividing each price by the maximum price for its metropolitan area over the entire timespan of the dataset
    b) change in price = final price - initial price
    c) store the name of the metropolitan area and the change in its normalized price in a data structure
2) sort the metropolitan areas in descending order by change in price
3) define two variables to store the increase in normalized price for the two user-specified metropolitan areas
4) iterate through the sorted list of metropolitan areas, updating each metropolitan area's price variable
5) choose the metropolitan area that had the highest increase in normalized price over the entire timespan of the dataset
6) return the names and the respective increases in normalized price for the two metropolitan areas
```

In order to determine which of a PAIR of metropolitan areas had a greater increase in median price over the timespan of the dataset, a similar algorithm was used. The main difference in the two algorithms is that TWO variables were used to store the increase in normalized price the increases in normalized price for both metropolitan areas was returned.

Slide 28



Development of Methods and Algorithms

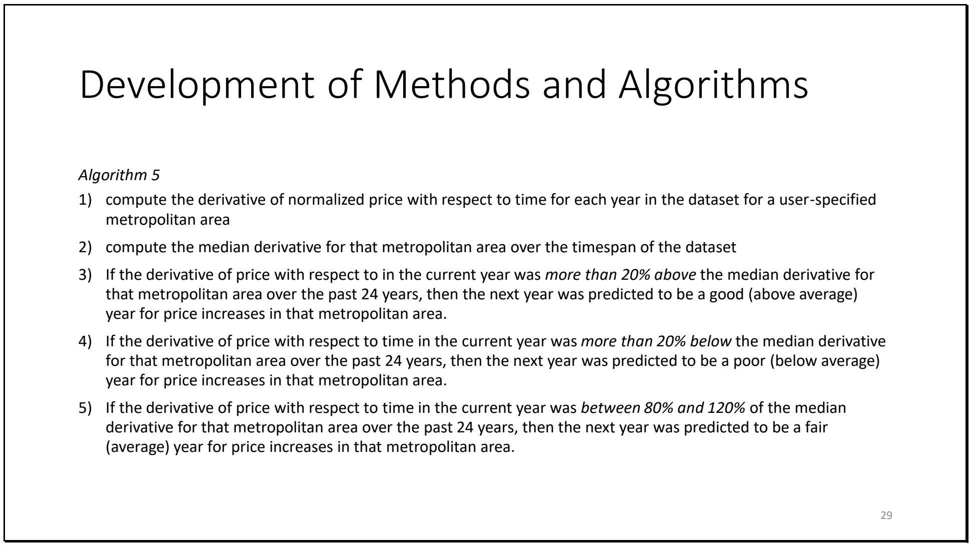
Algorithm 4

1) find the minimum monthly price for a user-specified metropolitan area
2) iterate through the succeeding columns until a price greater than or equal to twice the minimum price is found
3) count the number of months that occurred between these two price points
4) divide the number of months by 12 to obtain the doubling time in years
5) round the doubling time in years to the desired precision

28

To determine the doubling time for a metropolitan area,

Slide 29



Development of Methods and Algorithms

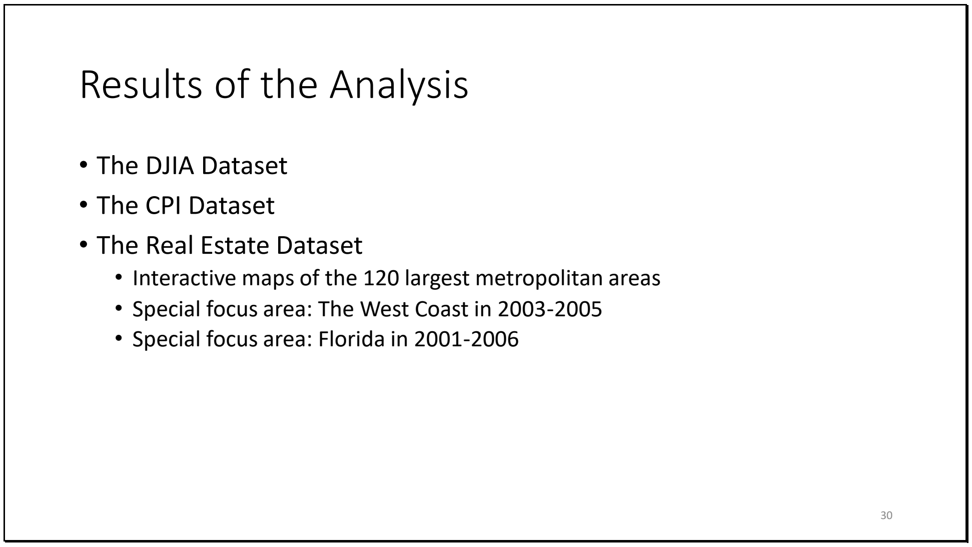
Algorithm 5

1) compute the derivative of normalized price with respect to time for each year in the dataset for a user-specified metropolitan area
2) compute the median derivative for that metropolitan area over the timespan of the dataset
3) If the derivative of price with respect to in the current year was *more than 20% above* the median derivative for that metropolitan area over the past 24 years, then the next year was predicted to be a good (above average) year for price increases in that metropolitan area.
4) If the derivative of price with respect to time in the current year was *more than 20% below* the median derivative for that metropolitan area over the past 24 years, then the next year was predicted to be a poor (below average) year for price increases in that metropolitan area.
5) If the derivative of price with respect to time in the current year was *between 80% and 120%* of the median derivative for that metropolitan area over the past 24 years, then the next year was predicted to be a fair (average) year for price increases in that metropolitan area.

29

-> To determine whether prices in a given metropolitan area would rise or fall in the next year, -> I normalized the prices -> took derivatives -> and compared the derivative for the current year -> with the median derivative for that metropolitan area. ->

Slide 30



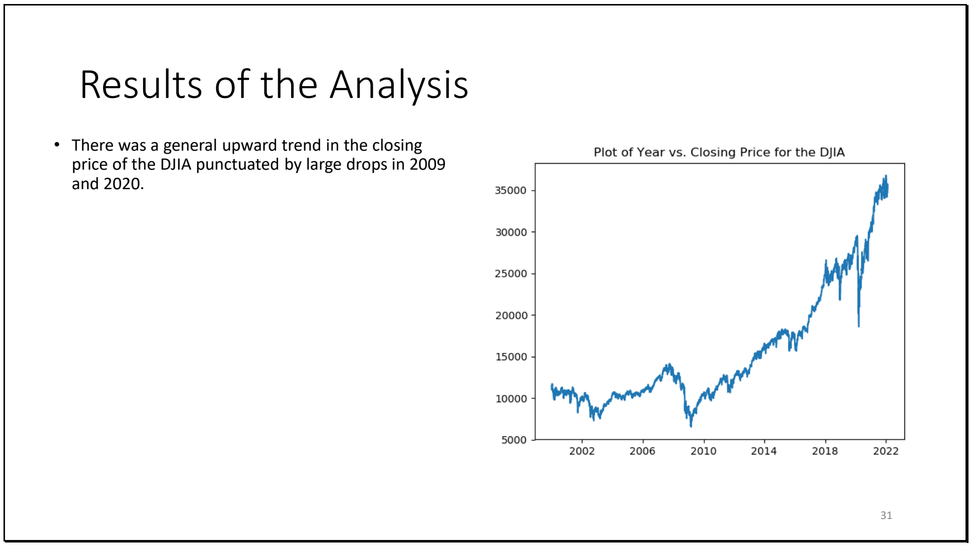
Results of the Analysis

- The DJIA Dataset
- The CPI Dataset
- The Real Estate Dataset
  - Interactive maps of the 120 largest metropolitan areas
  - Special focus area: The West Coast in 2003-2005
  - Special focus area: Florida in 2001-2006

30

The results of this analysis, will be broken into five parts. -> First, the results of the DJIA Dataset will be discussed. -> Then the results of the CPI Dataset will be discussed. -> -> Next, the results of the analysis on the real estate will be discussed with a special focus on the West Coast and Florida in the first half of the 2000s.

Slide 31

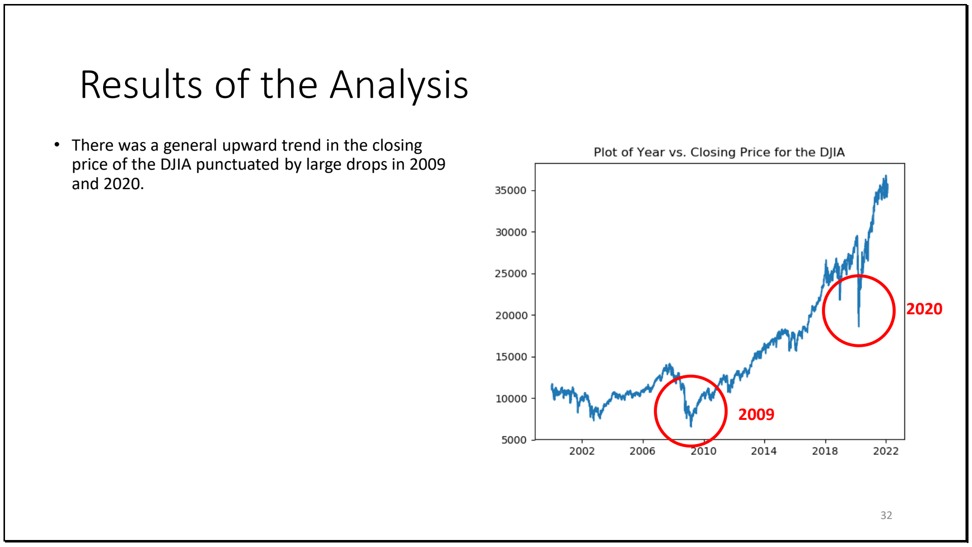


Results of the Analysis

- There was a general upward trend in the closing price of the DJIA punctuated by large drops in 2009 and 2020.

This is a graph of the closing price of the Dow Jones Industrial average from 2000 to 2022. As you can see, there is a general upward trend in the price as time increases. Also notice the two dramatic drops in the price in 2009 and 2020. The former was caused by the 2008-2009 recession, also know as the “Great Recession.” The latter was occurred during the first year of the COVID-19 pandemic.

Slide 32

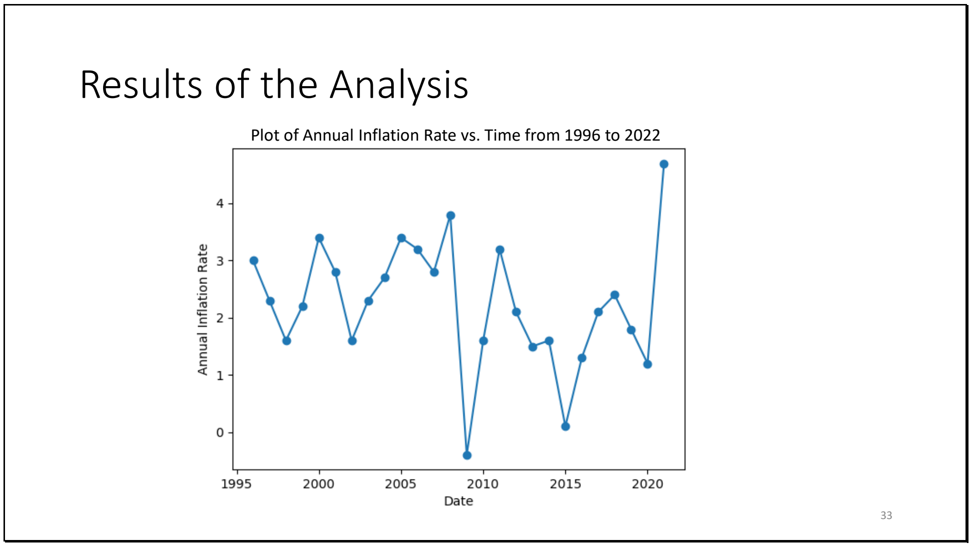


Results of the Analysis

- There was a general upward trend in the closing price of the DJIA punctuated by large drops in 2009 and 2020.

This is a graph of the closing price of the Dow Jones Industrial average from 2000 to 2022. As you can see, there is a general upward trend in the price as time increases. Also notice the two dramatic drops in the price in 2009 and 2020. The former was caused by the 2008-2009 recession, also know as the “Great Recession.” The latter occurred during the first year of the COVID-19 pandemic.

Slide 33

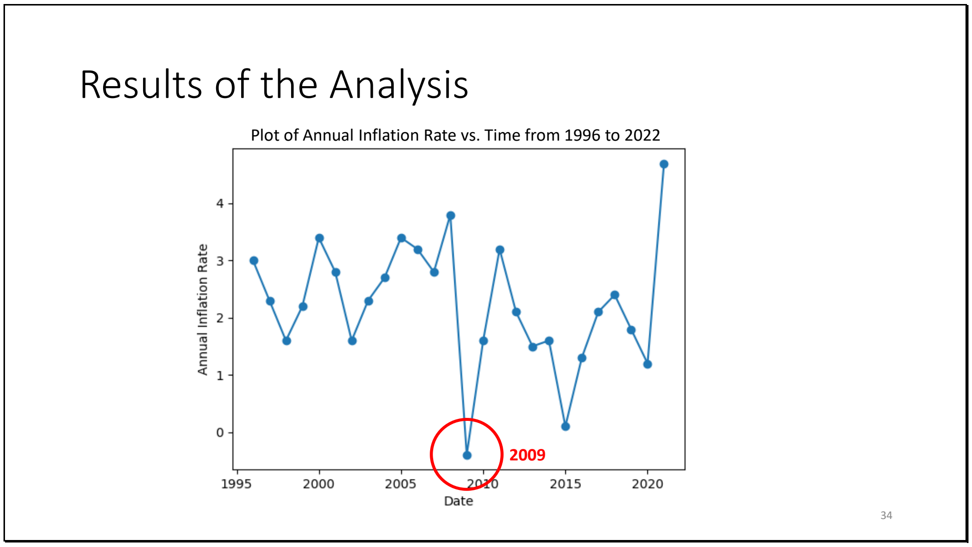


Results of the Analysis

Plot of Annual Inflation Rate vs. Time from 1996 to 2022

This graph shows the Annual Inflation Rate as measured by the Consumer Price Index

Slide 34

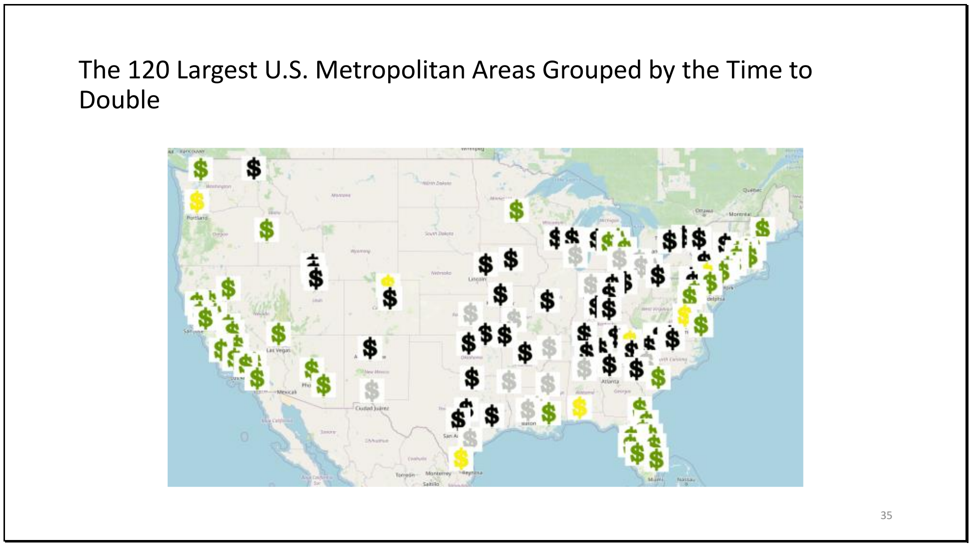


Results of the Analysis

34

As you can see from the graph, the annual inflation rate fell precipitously in 2009. Also, the annual inflation rate has risen sharply since 2020.

Slide 35

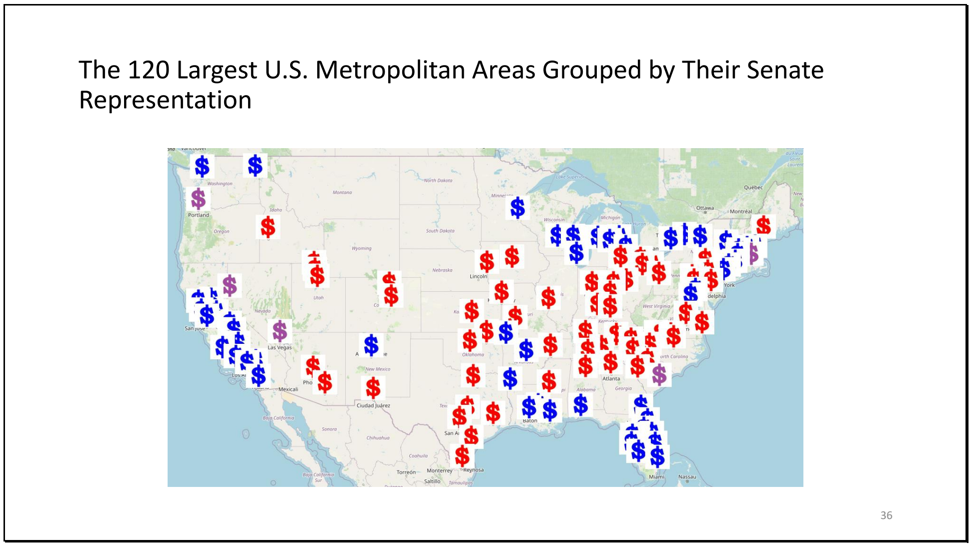


The 120 Largest U.S. Metropolitan Areas Grouped by the Time to Double

35

This map shows the largest 120 U.S. metropolitan areas by population grouped by the time for the normalized median price of a single-family home in that metropolitan area to double. The metropolitan areas denoted by the green dollar sign icon doubled in price the fastest, and the metropolitan areas denoted by the black dollar sign icon doubled in price the slowest.

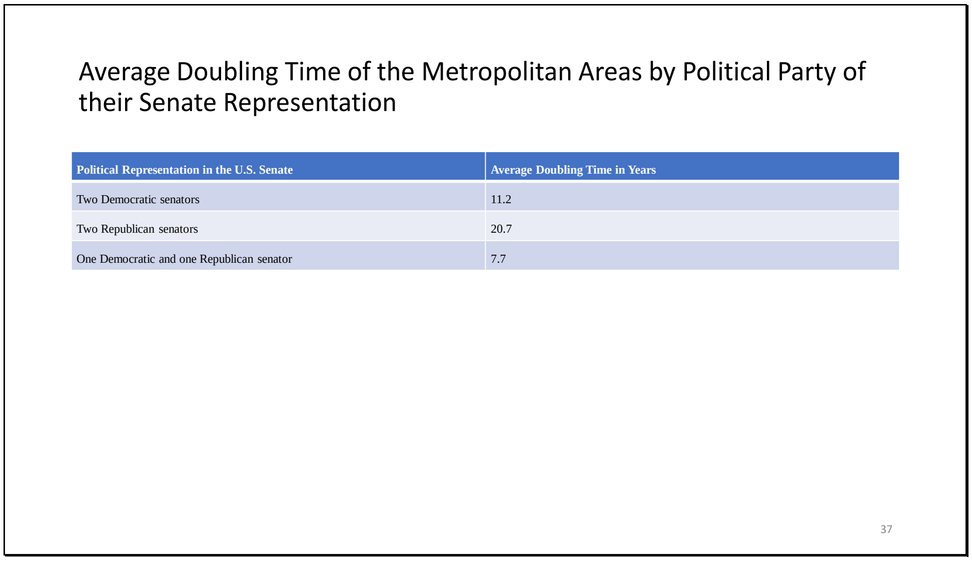
Slide 36



The 120 Largest U.S. Metropolitan Areas Grouped by Their Senate Representation

This map shows the largest 120 U.S. metropolitan areas by population grouped by the political affiliation of their state's U.S. Senate delegation. Red means two Republican senators, blue means two Democratic senators, and purple means one Republican senator and one Democratic senator.

Slide 37



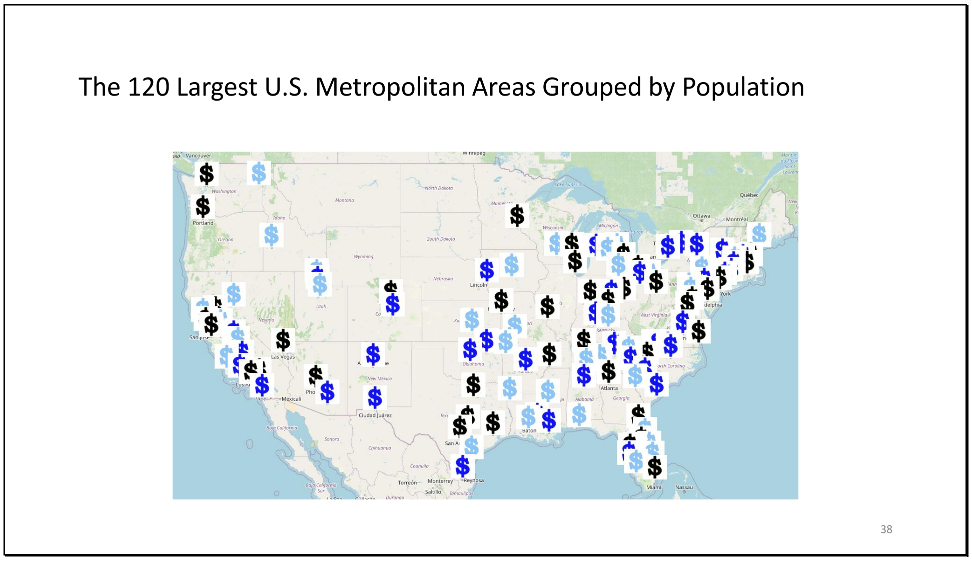
Average Doubling Time of the Metropolitan Areas by Political Party of their Senate Representation

Political Representation in the U.S. Senate	Average Doubling Time in Years
Two Democratic senators	11.2
Two Republican senators	20.7
One Democratic and one Republican senator	7.7

37

This table shows the average time for the normalized median price of a single-family home to double in the 120 largest U.S. metropolitan areas compared to the political affiliation of their Senate representation. The average doubling time of the metropolitan areas represented by Democratic and Republican senators (7.7) was lower than either the average doubling time of the metropolitan areas represented by Democratic senators (11.2 years), or the metropolitan areas represented by Republican senators (20.7 years). While the average doubling time for metropolitan areas represented by Republican senators was almost twice as long as the average doubling time of the metropolitan areas represented by Democratic senators.

Slide 38

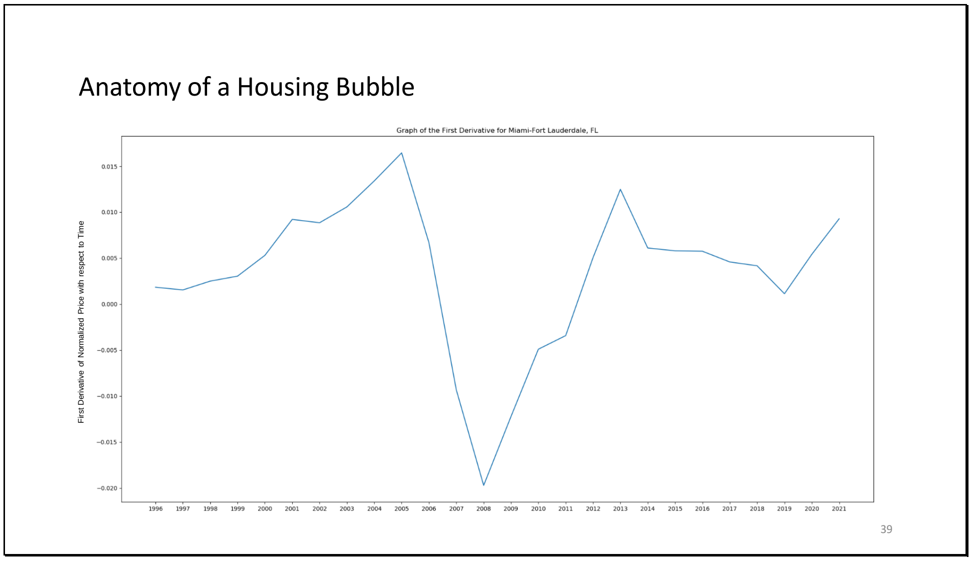


The 120 Largest U.S. Metropolitan Areas Grouped by Population

38

This map shows the largest 120 U.S. metropolitan areas by population grouped by the population size of the metropolitan area. The metropolitan areas denoted by the black dollar sign icon were the most populous, and the metropolitan areas denoted by the light blue dollar sign icon were the least populous.

Slide 39

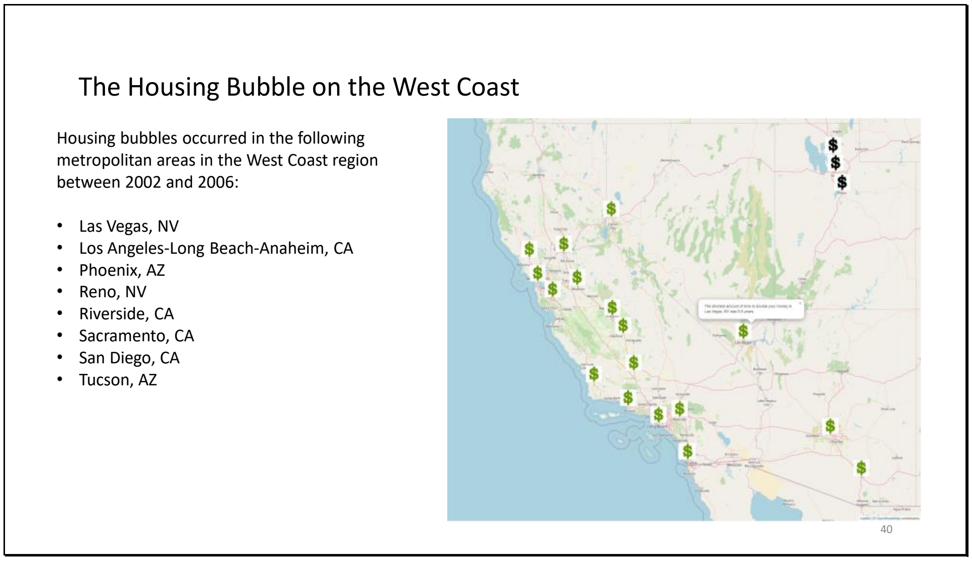


Anatomy of a Housing Bubble

39

"A housing bubble, or real estate bubble," has been defined as," a run-up in housing prices fueled by demand, speculation, and exuberant spending to the point of collapse." This graph shows the first derivative of the normalized median price of a single-family home in Miami, FL from January 31, 1996 to August 31, 2021. The characteristic "V-shape" in the graph is typical of real estate markets that experienced a housing bubble.

Slide 40



The Housing Bubble on the West Coast

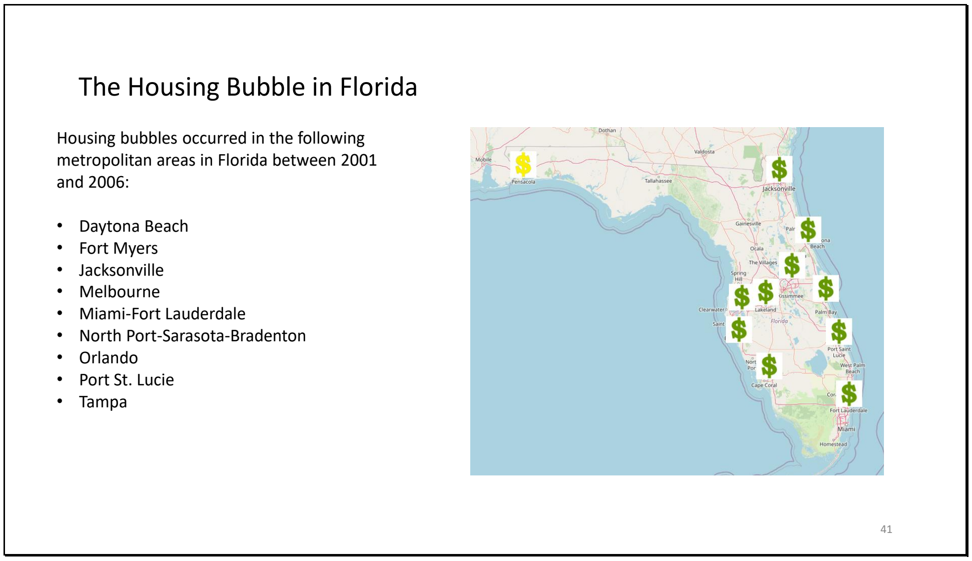
Housing bubbles occurred in the following metropolitan areas in the West Coast region between 2002 and 2006:

- Las Vegas, NV
- Los Angeles-Long Beach-Anaheim, CA
- Phoenix, AZ
- Reno, NV
- Riverside, CA
- Sacramento, CA
- San Diego, CA
- Tucson, AZ

40

For this analysis, a year was considered a bubble year if the following two conditions were met: the derivative of normalized price with respect to time in that year was greater than or equal to 120% of the median derivative of normalized price w.r.t. time in that metropolitan area over the entire 20-year span, and the first and second derivatives of normalized price w.r.t. time were positive. The metropolitan areas denoted by the green icons doubled in price in less than 10 years. These are the same areas that experienced a housing bubble between 2002 and 2006.

Slide 41



The Housing Bubble in Florida

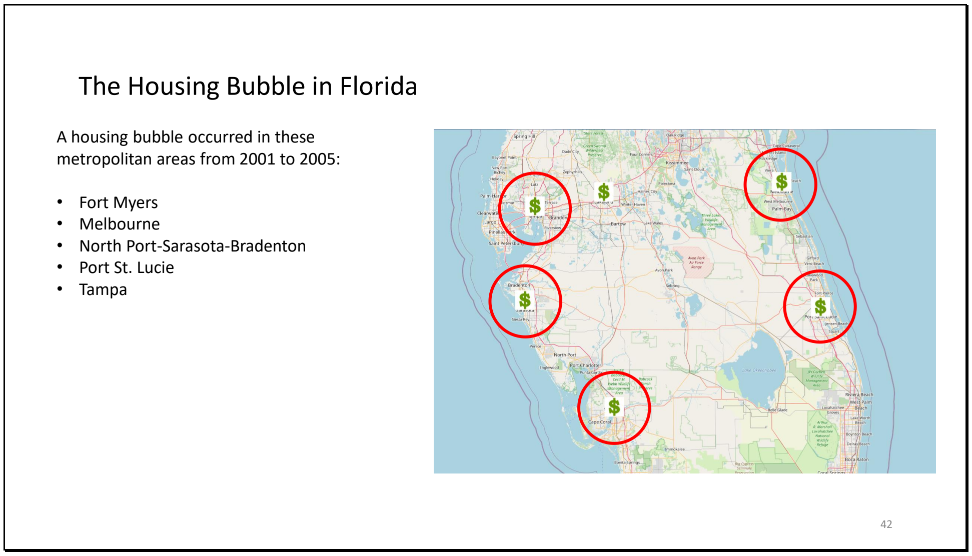
Housing bubbles occurred in the following metropolitan areas in Florida between 2001 and 2006:

- Daytona Beach
- Fort Myers
- Jacksonville
- Melbourne
- Miami-Fort Lauderdale
- North Port-Sarasota-Bradenton
- Orlando
- Port St. Lucie
- Tampa

41

A similar pattern was seen in Florida between 2001 and 2006. The conditions for a bubble year were met in most large metropolitan areas in Florida in the first half of the 2000s. The bubble usually lasted from 3 to 5 years. Pensacola was an outlier. Housing prices there doubled more slowly than they did in peninsular Florida.

Slide 42



The Housing Bubble in Florida

A housing bubble occurred in these metropolitan areas from 2001 to 2005:

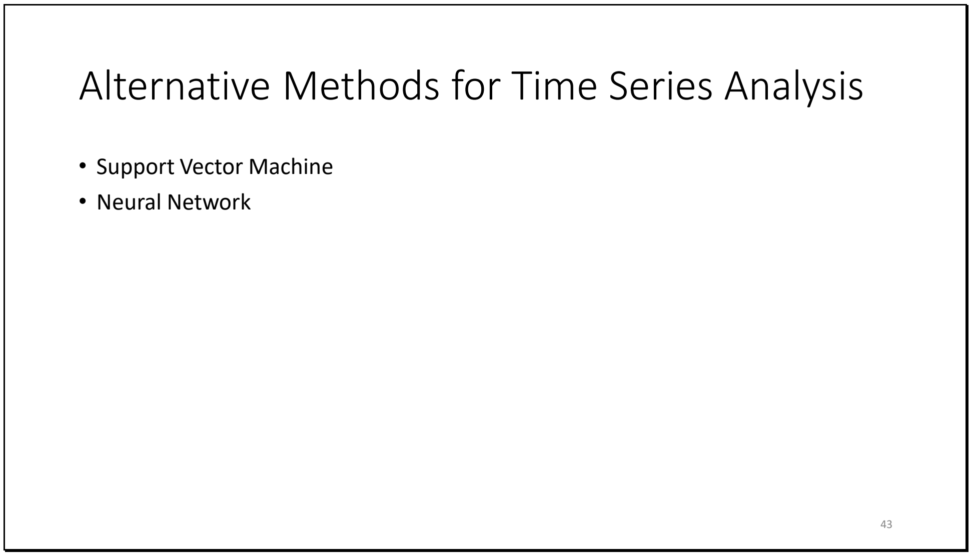
- Fort Myers
- Melbourne
- North Port-Sarasota-Bradenton
- Port St. Lucie
- Tampa

42

Of particular interest were the metropolitan areas in central Florida. -> This region experienced a sustained housing bubble from 2001 to 2005. In Fort Myers -> -> Melbourne -> -> North Port-Sarasota-Bradenton -> -> Port St. Lucie -> -> and Tampa -> ->

Lakeland also experienced a housing bubble, but it did not last from 2001 to 2005.

Slide 43



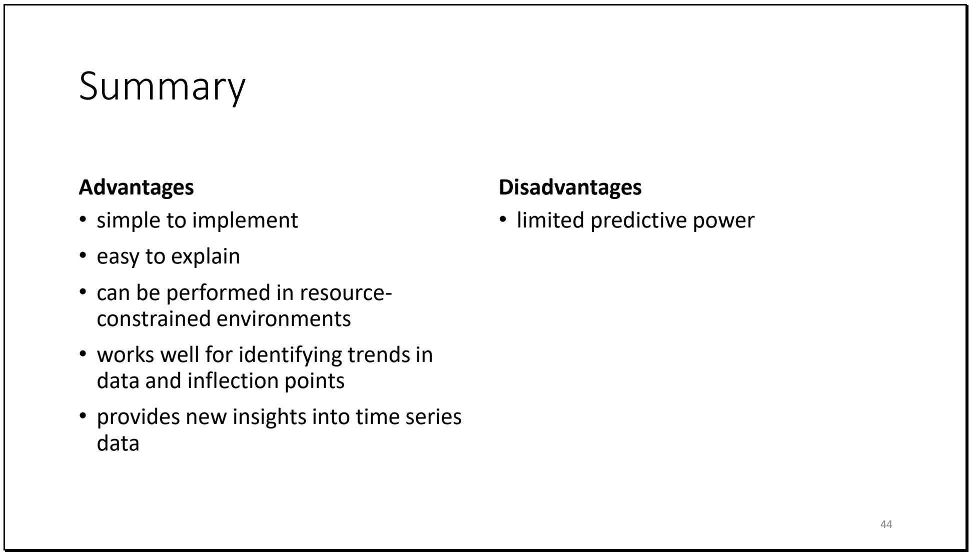
Alternative Methods for Time Series Analysis

- Support Vector Machine
- Neural Network

43

A time series dataset may be transformed into a supervised learning problem and fed as input to a support vector machine or a neural network to perform regression on it. This has the added advantage of being able to predict future values of the time series in a process called “time series forecasting.”

Slide 44



Summary

Advantages

- simple to implement
- easy to explain
- can be performed in resource-constrained environments
- works well for identifying trends in data and inflection points
- provides new insights into time series data

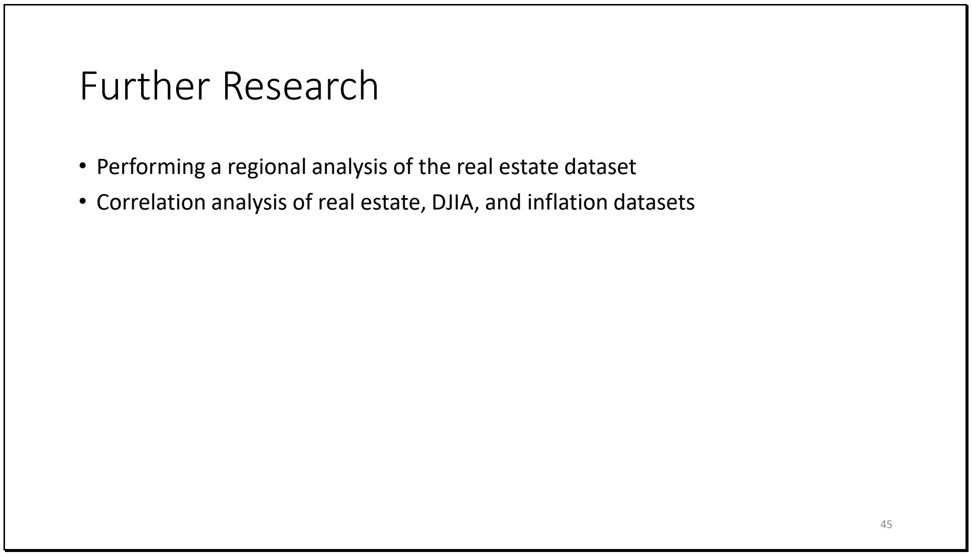
Disadvantages

- limited predictive power

44

By comparing the derivatives of time series data, researchers can ascertain whether a particular feature (e.g. the price of an asset) is increasing or decreasing in value and the rate at which this change is occurring. But the most valuable insights come as data is aggregated over many experimental units and over a long period of time. Then a general trend in the data points usually emerges.

Slide 45

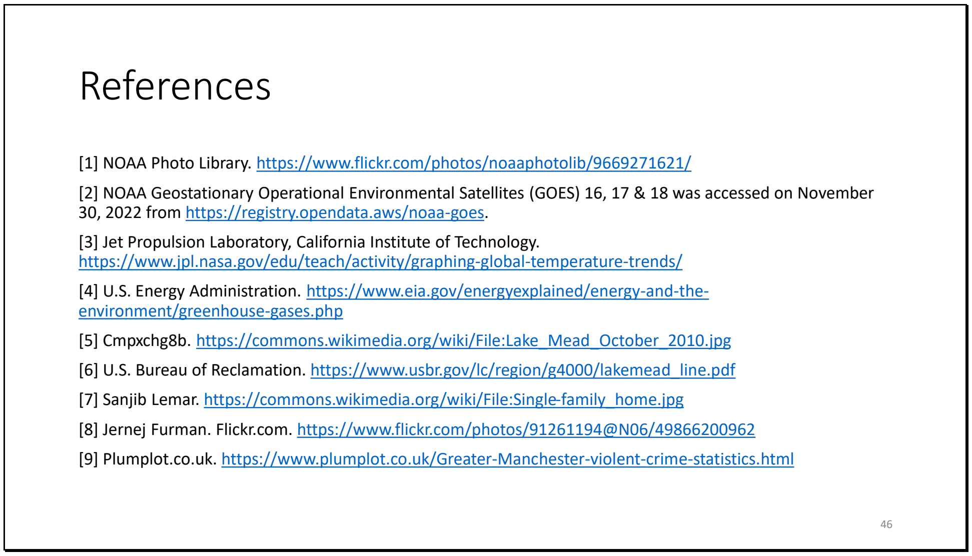


Further Research

- Performing a regional analysis of the real estate dataset
- Correlation analysis of real estate, DJIA, and inflation datasets

45

Slide 46



References

[1] NOAA Photo Library. https://www.flickr.com/photos/noaaphotolib/9669271621/

[2] NOAA Geostationary Operational Environmental Satellites (GOES) 16, 17 & 18 was accessed on November 30, 2022 from https://registry.opendata.aws/noaa-goes.

[3] Jet Propulsion Laboratory, California Institute of Technology. https://www.jpl.nasa.gov/edu/teach/activity/graphing-global-temperature-trends/

[4] U.S. Energy Administration. https://www.eia.gov/energyexplained/energy-and-the-environment/greenhouse-gases.php

[5] Cmpxchg8b. https://commons.wikimedia.org/wiki/File:Lake_Mead_October_2010.jpg

[6] U.S. Bureau of Reclamation. https://www.usbr.gov/lc/region/g4000/lakemead_line.pdf

[7] Sanjib Lemar. https://commons.wikimedia.org/wiki/File:Single-family_home.jpg

[8] Jernej Furman. Flickr.com. https://www.flickr.com/photos/91261194@N06/49866200962

[9] Plumplot.co.uk. https://www.plumplot.co.uk/Greater-Manchester-violent-crime-statistics.html

46

The image on the first slide shows "Annual crime rate per 1000 workday people"

Appendix II: Source Code

ca_reader.py

```

import pandas as pd
import matplotlib.pyplot as plt
import sys
import os
import numpy as np
from dateutil import parser

date_format = "%m/%d/%Y"
COASTAL = False

def Rank_Cities_in_Descending_Order(t):
t.sort(key = lambda x: x[1], reverse=True)
return t

if __name__=="__main__":
user_preference = input("Do you want cities in coastal or non-coastal states? ")
user_preference = user_preference.upper()
if user_preference == "COASTAL":
COASTAL = True
else:
COASTAL = False
mode = 1
period_length = 1

#period_name = input_year
str_number_of_cities_to_list = "911"

# process the dataframe
original_df = pd.read_csv("Metro_zhvi_uc_sfr_tier_0.33_0.67_sm_sa_month.csv")

# pre-processing steps
# step 1: get the column names
complete_rows_df = original_df
size_rank_column_df = original_df[['SizeRank']]
region_name_column_df = original_df[['RegionName']]
size_rank_column_series = size_rank_column_df.squeeze()
region_name_column_series = region_name_column_df.squeeze()
region_name_column_list = region_name_column_series.tolist()
number_of_rows = original_df.shape[0]
city_name_size_rank_dict = {}
for i in range(0, number_of_rows):
city_name_size_rank_dict[region_name_column_series[i]] = size_rank_column_series[i]

# step 2: drop "RegionID", "SizeRank", "RegionType", and "StateName" columns
df = complete_rows_df.drop(columns=['RegionID', 'SizeRank', 'RegionType'])
df = df.iloc[1: , :]
```

```
if COASTAL == True:
df = df[df["RegionName"].str.contains("CA|OR|WA|AK|HI|ME|NH|MA|RI|CT|NY|NJ|DE|MD|VA|NC|SC|GA|FL|TX|LA|MS|AL")]
else:
df = df[df["RegionName"].str.contains("ID|MT|ND|MN|WI|MI|VT|WY|SD|IA|PA|NV|UT|CO|NE|IL|IN|OH|KS|MO|KY|WV|AZ|NM|OK|AR|TN|DC")]
print(df.head())

#if you want cities in a single state, do

#ca_cities = df[df["RegionName"].str.contains("CA")]
```

djia_derivatives.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sys
from datetime import datetime
import statistics

if __name__ == "__main__":
djia_df = pd.read_csv("DJIA2000-2022FebCSV.csv", parse_dates=[0]) # DJIA opening/closing price dataset
djia_df['Close*'] = djia_df['Close*'].str.replace(',','') # remove commas before conversion to float
closing_price_series = djia_df['Close*']
closing_price_array = closing_price_series.to_numpy('float64')
djia_array = djia_df['Close*'].astype('float64')
first_derivative_list = []
current_date_and_first_deriv_tuple_list = []
second_derivative_list = []
current_date_and_second_deriv_tuple_list = []
time_intervals = []

for i in range(len(closing_price_array) - 1):
initial_price = closing_price_array[i+1] # because the dataset is in reverse time order
final_price = closing_price_array[i]
change_in_price = final_price - initial_price
initial_time = datetime.strptime(str(djia_df.iloc[i+1, 0]), "%Y-%m-%d %H:%M:%S")
final_time = datetime.strptime(str(djia_df.iloc[i, 0]), "%Y-%m-%d %H:%M:%S")
change_in_time = final_time - initial_time
dp_dt = change_in_price / change_in_time.days
time_intervals.append(initial_time)
if dp_dt >= 0:
is_positive = True
else:
is_positive = False
first_derivative_list.append(dp_dt)
current_date = djia_df.iloc[i, 0]
```

```
current_date_and_first_deriv_tuple = (current_date, dp_dt, is_positive)
current_date_and_first_deriv_tuple_list.append(current_date_and_first_deriv_tuple)

first_derivative_array = np.array(first_derivative_list)
for i in range(len(first_derivative_array) - 1):
initial_price = first_derivative_array[i]
final_price = first_derivative_array[i+1]
change_in_price = final_price - initial_price # this is actually the change in the derivative of price w.r.t time
initial_time = datetime.strptime(str(djia_df.iloc[i+2, 0]), "%Y-%m-%d %H:%M:%S")
final_time = datetime.strptime(str(djia_df.iloc[i, 0]), "%Y-%m-%d %H:%M:%S")
change_in_time = final_time - initial_time
dp_dt = change_in_price / change_in_time.days
if dp_dt >= 0:
is_positive = True
else:
is_positive = False
second_derivative_list.append(dp_dt)
current_date = djia_df.iloc[i, 0]
current_date_and_second_deriv_tuple = (current_date, dp_dt, is_positive)
current_date_and_second_deriv_tuple_list.append(current_date_and_second_deriv_tuple)

# lists of first derivatives
the_2000_derivatives = []
the_2001_derivatives = []
the_2002_derivatives = []
the_2003_derivatives = []
the_2004_derivatives = []
the_2005_derivatives = []
the_2006_derivatives = []
the_2007_derivatives = []
the_2008_derivatives = []
the_2009_derivatives = []
the_2010_derivatives = []
the_2011_derivatives = []
the_2012_derivatives = []
the_2013_derivatives = []
the_2014_derivatives = []
the_2015_derivatives = []
the_2016_derivatives = []
the_2017_derivatives = []
the_2018_derivatives = []
the_2019_derivatives = []
the_2020_derivatives = []
the_2021_derivatives = []
the_2022_derivatives = []

print("Table of the first derivative of closing price w.r.t. time for the DJIA dataset.\n")
print("------------------------")
```

```
for i in range(0, 30):
entry = current_date_and_first_deriv_tuple_list[i]
date_obj = entry[0]
deriv = entry[1]
deriv_2f = "{:.2f}".format(deriv)
d = date_obj.date()
if d.year == 2000:
the_2000_derivatives.append(deriv)
if d.year == 2001:
the_2001_derivatives.append(deriv)
if d.year == 2002:
the_2002_derivatives.append(deriv)
if d.year == 2003:
the_2003_derivatives.append(deriv)
if d.year == 2004:
the_2004_derivatives.append(deriv)
if d.year == 2005:
the_2005_derivatives.append(deriv)
if d.year == 2006:
the_2006_derivatives.append(deriv)
if d.year == 2007:
the_2007_derivatives.append(deriv)
if d.year == 2008:
the_2008_derivatives.append(deriv)
if d.year == 2009:
the_2009_derivatives.append(deriv)
if d.year == 2010:
the_2010_derivatives.append(deriv)
if d.year == 2011:
the_2011_derivatives.append(deriv)
if d.year == 2012:
the_2012_derivatives.append(deriv)
if d.year == 2013:
the_2013_derivatives.append(deriv)
if d.year == 2014:
the_2014_derivatives.append(deriv)
if d.year == 2015:
the_2015_derivatives.append(deriv)
if d.year == 2016:
the_2016_derivatives.append(deriv)
if d.year == 2017:
the_2017_derivatives.append(deriv)
if d.year == 2018:
the_2018_derivatives.append(deriv)
if d.year == 2019:

the_2019_derivatives.append(deriv)
if d.year == 2020:
the_2020_derivatives.append(deriv)
if d.year == 2021:
the_2021_derivatives.append(deriv)
if d.year == 2022:
the_2022_derivatives.append(deriv)
```

```

#print('|', date_obj.date(), '|', deriv_2f, '|')

print("| ... | ... |")
print("------------------------")
print(len(current_date_and_first_deriv_tuple_list), 'rows.')
print('\n')
first_entry = current_date_and_first_deriv_tuple_list[0]
part0 = first_entry[0]
#print('part0:', part0)
first_median = statistics.median(first_derivative_list)
first_median_2f = "{:.2f}".format(first_median)
print("The median of the first derivative of price w.r.t time is", first_median_2f)

years_with_derivatives_greater_than_1_20_list = []

for deriv in the_2000_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2000)
for deriv in the_2001_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2001)
for deriv in the_2002_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2002)
for deriv in the_2003_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2003)
for deriv in the_2004_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2004)
for deriv in the_2005_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2005)
for deriv in the_2006_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2006)
for deriv in the_2007_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2007)
for deriv in the_2008_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2008)
for deriv in the_2009_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2009)
for deriv in the_2010_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2010)
for deriv in the_2011_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2011)
for deriv in the_2012_derivatives:
```

```
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2012)
for deriv in the_2013_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2013)
for deriv in the_2014_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2014)
for deriv in the_2015_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2015)
for deriv in the_2016_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2016)
for deriv in the_2017_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2017)
for deriv in the_2018_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2018)
for deriv in the_2019_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2019)
for deriv in the_2020_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2020)
for deriv in the_2021_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2021)
for deriv in the_2022_derivatives:
if deriv >= 1.2 * first_median:
years_with_derivatives_greater_than_1_20_list.append(2022)

years_with_derivatives_greater_than_1_20_set = set( years_with_derivatives_greater_than_1_20_list)
print('Years in which the first derivative of closing w.r.t. time was >= 120 percent of the median derivative: ', years_with_derivatives_greater_than_1_20_set)

print("Table of the second derivative of closing price w.r.t. time for the DJIA dataset.\n")
print("-------------------------")
for i in range(0, len(current_date_and_second_deriv_tuple_list)):
entry = current_date_and_second_deriv_tuple_list[i]
date_obj = entry[0]
deriv = entry[1]
deriv_2f = "{:.2f}".format(deriv)

print("| ... | ... |")
print("-------------------------")
print(len(current_date_and_second_deriv_tuple_list), 'rows.')
print('\n')
second_median = statistics.median(second_derivative_list)
```

```
second_median_2f = "{:.2f}".format(second_median)
print("The median of the second derivative of price w.r.t time is", second_median_2f)

x = time_intervals
y = first_derivative_array
my_xticks = np.linspace(0, 56000, 1)
plt.xticks(x, my_xticks)
plt.plot(x, y)
plt.show()

```

highest.py

```
import pandas as pd
import matplotlib.pyplot as plt
import sys
import os
import numpy as np
from dateutil import parser

if __name__=="__main__":

chosen_year = input("Which year? ")
int_chosen_year = int(chosen_year)

mode = 1
period_length = 1

# process the dataframe
original_df = pd.read_csv("Metro_zhvi_uc_sfr_tier_0.33_0.67_sm_sa_month.csv")

# pre-processing steps
# step 1: get the column names
complete_rows_df = original_df
size_rank_column_df = original_df[['SizeRank']]
region_name_column_df = original_df[['RegionName']]
size_rank_column_series = size_rank_column_df.squeeze()
region_name_column_series = region_name_column_df.squeeze()
region_name_column_list = region_name_column_series.tolist()
number_of_rows = original_df.shape[0]
city_name_size_rank_dict = {}
for i in range(0, number_of_rows):
city_name_size_rank_dict[region_name_column_series[i]] = size_rank_column_series[i]

# step 2: drop "RegionID", "SizeRank", "RegionType", and "StateName" columns
df = complete_rows_df.drop(columns=['RegionID', 'SizeRank', 'RegionType', 'StateName'])
df = df.iloc[1: , :] # remove the first row because it doesn't have a state
#if COASTAL == True:
# df = df[df["RegionName"].str.contains("CA|OR|WA|AK|HI|ME|NH|MA|RI|CT|NY|NJ|DE|MD|VA|NC|SC|
GA|FL|TX|LA|MS|AL")]
#else:
# df =
df[df["RegionName"].str.contains("ID|MT|ND|MN|WI|MI|VT|WY|SD|IA|PA|NV|UT|CO|NE|IL|IN|
OH|KS|MO|KY|WV|AZ|NM|OK|AR|TN|DC")]

str_number_of_cities_to_list = df.shape[0]

# create a numeric dataframe consisting of only floats
float_df = df.drop(columns=['RegionName'])

periods = []
period_dict = {}

# partition the floating point dataframe into 25 1-year periods
for i in range(0,180,12):
period = float_df.iloc[:, i:i+12]
periods.append(period)
period_dict[0] = '1996'
period_dict[1] = '1997'
period_dict[2] = '1998'
period_dict[3] = '1999'
period_dict[4] = '2000'
period_dict[5] = '2001'
period_dict[6] = '2002'
period_dict[7] = '2003'
period_dict[8] = '2004'
period_dict[9] = '2005'
period_dict[10] = '2006'
period_dict[11] = '2007'
period_dict[12] = '2008'
period_dict[13] = '2009'
period_dict[14] = '2010'
period_dict[15] = '2011'
period_dict[16] = '2012'
period_dict[17] = '2013'
period_dict[18] = '2014'
period_dict[19] = '2015'
period_dict[20] = '2016'
period_dict[21] = '2017'
period_dict[22] = '2018'
period_dict[23] = '2019'
period_dict[24] = '2020'
print('periods', periods[0])
# the outer list is the states
# the inner list is the cities in the state
# Washington, DC will be treated as a state
list_of_state_high_tuples = [[] for i in range(51)]
list_of_state_low_tuples = [[] for i in range(51)]

# get the city names and put them in a list
city_names_list = df['RegionName'].values.tolist()
```

```
#big_city_list = city_names_list[1:40]
#all_period_list = []
for each_city in city_names_list:
abs_price_inc_list = []
highest_prices = []
lowest_prices = []
for i in range(0, len(period)):
#print('periods[j] is a', type(periods[j]) )
#print('the columns of this period are:', period.columns)
first = i
last = i+1
row = period.iloc[first:last, :]
# convert row to Panadas series
row_series = row.squeeze(axis=0)
rowmax = row_series.max()
rowmin = row_series.min()
city_names_list[i]
name_price_tuple_high = (city_names_list[i], rowmax)
name_price_tuple_low = (city_names_list[i], rowmin)
highest_prices.append(name_price_tuple_high)
lowest_prices.append(name_price_tuple_low)
if name_price_tuple_high[0] == each_city:
this_city_name = name_price_tuple_high[0]
state_abbr = this_city_name[-2:]
#print('highest price for that city was', name_price_tuple_high[1], 'in', year)
if state_abbr == "AL":
list_of_state_high_tuples[0].append(name_price_tuple_high)
if state_abbr == "AK":
list_of_state_high_tuples[1].append(name_price_tuple_high)
if state_abbr == "AZ":
list_of_state_high_tuples[2].append(name_price_tuple_high)
if state_abbr == "AR":
list_of_state_high_tuples[3].append(name_price_tuple_high)
if state_abbr == "CA":
list_of_state_high_tuples[4].append(name_price_tuple_high)
if state_abbr == "CO":
list_of_state_high_tuples[5].append(name_price_tuple_high)
if state_abbr == "CT":
list_of_state_high_tuples[6].append(name_price_tuple_high)
if state_abbr == "DC":
list_of_state_high_tuples[7].append(name_price_tuple_high)

if state_abbr == "DE":
list_of_state_high_tuples[8].append(name_price_tuple_high)
if state_abbr == "FL":
list_of_state_high_tuples[9].append(name_price_tuple_high)
if state_abbr == "GA":
list_of_state_high_tuples[10].append(name_price_tuple_high)
if state_abbr == "HI":
list_of_state_high_tuples[11].append(name_price_tuple_high)
if state_abbr == "ID":
list_of_state_high_tuples[12].append(name_price_tuple_high)
```

```
if state_abbr == "IL":
list_of_state_high_tuples[13].append(name_price_tuple_high)
if state_abbr == "IN":
list_of_state_high_tuples[14].append(name_price_tuple_high)
if state_abbr == "IA":
list_of_state_high_tuples[15].append(name_price_tuple_high)
if state_abbr == "KS":
list_of_state_high_tuples[16].append(name_price_tuple_high)
if state_abbr == "KY":
list_of_state_high_tuples[17].append(name_price_tuple_high)
if state_abbr == "LA":
list_of_state_high_tuples[18].append(name_price_tuple_high)
if state_abbr == "ME":
list_of_state_high_tuples[19].append(name_price_tuple_high)
if state_abbr == "MD":
list_of_state_high_tuples[20].append(name_price_tuple_high)
if state_abbr == "MA":
list_of_state_high_tuples[21].append(name_price_tuple_high)
if state_abbr == "MI":
list_of_state_high_tuples[22].append(name_price_tuple_high)
if state_abbr == "MN":
list_of_state_high_tuples[23].append(name_price_tuple_high)
if state_abbr == "MS":
list_of_state_high_tuples[24].append(name_price_tuple_high)
if state_abbr == "MO":
list_of_state_high_tuples[25].append(name_price_tuple_high)
if state_abbr == "MT":
list_of_state_high_tuples[26].append(name_price_tuple_high)
if state_abbr == "NE":
list_of_state_high_tuples[27].append(name_price_tuple_high)
if state_abbr == "NV":
list_of_state_high_tuples[28].append(name_price_tuple_high)
if state_abbr == "NH":
list_of_state_high_tuples[29].append(name_price_tuple_high)
if state_abbr == "NJ":
list_of_state_high_tuples[30].append(name_price_tuple_high)
if state_abbr == "NM":
list_of_state_high_tuples[31].append(name_price_tuple_high)
if state_abbr == "NY":
list_of_state_high_tuples[32].append(name_price_tuple_high)
if state_abbr == "NC":
list_of_state_high_tuples[33].append(name_price_tuple_high)
if state_abbr == "ND":
list_of_state_high_tuples[34].append(name_price_tuple_high)
if state_abbr == "OH":
list_of_state_high_tuples[35].append(name_price_tuple_high)
if state_abbr == "OK":
list_of_state_high_tuples[36].append(name_price_tuple_high)
if state_abbr == "OR":
list_of_state_high_tuples[37].append(name_price_tuple_high)
if state_abbr == "PA":
list_of_state_high_tuples[38].append(name_price_tuple_high)
if state_abbr == "RI":
```

```
list_of_state_high_tuples[39].append(name_price_tuple_high)
if state_abbr == "SC":
list_of_state_high_tuples[40].append(name_price_tuple_high)
if state_abbr == "SD":
list_of_state_high_tuples[41].append(name_price_tuple_high)
if state_abbr == "TN":
list_of_state_high_tuples[42].append(name_price_tuple_high)
if state_abbr == "TX":
list_of_state_high_tuples[43].append(name_price_tuple_high)
if state_abbr == "UT":
list_of_state_high_tuples[44].append(name_price_tuple_high)
if state_abbr == "VT":
list_of_state_high_tuples[45].append(name_price_tuple_high)
if state_abbr == "VA":
list_of_state_high_tuples[46].append(name_price_tuple_high)
if state_abbr == "WA":
list_of_state_high_tuples[47].append(name_price_tuple_high)
if state_abbr == "WV":
list_of_state_high_tuples[48].append(name_price_tuple_high)
if state_abbr == "WI":
list_of_state_high_tuples[49].append(name_price_tuple_high)
if state_abbr == "WY":
list_of_state_high_tuples[50].append(name_price_tuple_high)

# AL is index 0 in list_of_state_high_tuples
# WY is index 50 in list_of_state_high_tuples
if name_price_tuple_low[0] == each_city:
this_city_name = name_price_tuple_low[0]
state_abbr = this_city_name[-2:]
if state_abbr == "AL":
list_of_state_low_tuples[0].append(name_price_tuple_low)
if state_abbr == "AK":
list_of_state_low_tuples[1].append(name_price_tuple_low)
if state_abbr == "AZ":
list_of_state_low_tuples[2].append(name_price_tuple_low)
if state_abbr == "AR":
list_of_state_low_tuples[3].append(name_price_tuple_low)
if state_abbr == "CA":
list_of_state_low_tuples[4].append(name_price_tuple_low)
if state_abbr == "CO":
list_of_state_low_tuples[5].append(name_price_tuple_low)
if state_abbr == "CT":
list_of_state_low_tuples[6].append(name_price_tuple_low)
if state_abbr == "DC":
list_of_state_low_tuples[7].append(name_price_tuple_low)
if state_abbr == "DE":
list_of_state_low_tuples[8].append(name_price_tuple_low)
if state_abbr == "FL":
list_of_state_low_tuples[9].append(name_price_tuple_low)
if state_abbr == "GA":
list_of_state_low_tuples[10].append(name_price_tuple_low)
if state_abbr == "HI":
```

```
list_of_state_low_tuples[11].append(name_price_tuple_low)
if state_abbr == "ID":
list_of_state_low_tuples[12].append(name_price_tuple_low)
if state_abbr == "IL":
list_of_state_low_tuples[13].append(name_price_tuple_low)
if state_abbr == "IN":
list_of_state_low_tuples[14].append(name_price_tuple_low)
if state_abbr == "IA":
list_of_state_low_tuples[15].append(name_price_tuple_low)
if state_abbr == "KS":
list_of_state_low_tuples[16].append(name_price_tuple_low)
if state_abbr == "KY":
list_of_state_low_tuples[17].append(name_price_tuple_low)
if state_abbr == "LA":
list_of_state_low_tuples[18].append(name_price_tuple_low)
if state_abbr == "ME":
list_of_state_low_tuples[19].append(name_price_tuple_low)
if state_abbr == "MD":

list_of_state_low_tuples[20].append(name_price_tuple_low)
if state_abbr == "MA":
list_of_state_low_tuples[21].append(name_price_tuple_low)
if state_abbr == "MI":
list_of_state_low_tuples[22].append(name_price_tuple_low)
if state_abbr == "MN":
list_of_state_low_tuples[23].append(name_price_tuple_low)
if state_abbr == "MS":
list_of_state_low_tuples[24].append(name_price_tuple_low)
if state_abbr == "MO":
list_of_state_low_tuples[25].append(name_price_tuple_low)
if state_abbr == "MT":
list_of_state_low_tuples[26].append(name_price_tuple_low)
if state_abbr == "NE":
list_of_state_low_tuples[27].append(name_price_tuple_low)
if state_abbr == "NV":
list_of_state_low_tuples[28].append(name_price_tuple_low)
if state_abbr == "NH":
list_of_state_low_tuples[29].append(name_price_tuple_low)
if state_abbr == "NJ":
list_of_state_low_tuples[30].append(name_price_tuple_low)
if state_abbr == "NM":
list_of_state_low_tuples[31].append(name_price_tuple_low)
if state_abbr == "NY":
list_of_state_low_tuples[32].append(name_price_tuple_low)
if state_abbr == "NC":
list_of_state_low_tuples[33].append(name_price_tuple_low)
if state_abbr == "ND":
list_of_state_low_tuples[34].append(name_price_tuple_low)
if state_abbr == "OH":
list_of_state_low_tuples[35].append(name_price_tuple_low)
if state_abbr == "OK":
list_of_state_low_tuples[36].append(name_price_tuple_low)
if state_abbr == "OR":
```

```
list_of_state_low_tuples[37].append(name_price_tuple_low)
if state_abbr == "PA":
list_of_state_low_tuples[38].append(name_price_tuple_low)
if state_abbr == "RI":
list_of_state_low_tuples[39].append(name_price_tuple_low)
if state_abbr == "SC":
list_of_state_low_tuples[40].append(name_price_tuple_low)
if state_abbr == "SD":
list_of_state_low_tuples[41].append(name_price_tuple_low)
if state_abbr == "TN":
list_of_state_low_tuples[42].append(name_price_tuple_low)
if state_abbr == "TX":
list_of_state_low_tuples[43].append(name_price_tuple_low)
if state_abbr == "UT":
list_of_state_low_tuples[44].append(name_price_tuple_low)
if state_abbr == "VT":
list_of_state_low_tuples[45].append(name_price_tuple_low)
if state_abbr == "VA":
list_of_state_low_tuples[46].append(name_price_tuple_low)
if state_abbr == "WA":
list_of_state_low_tuples[47].append(name_price_tuple_low)
if state_abbr == "WV":
list_of_state_low_tuples[48].append(name_price_tuple_low)
if state_abbr == "WI":
list_of_state_low_tuples[49].append(name_price_tuple_low)
if state_abbr == "WY":
list_of_state_low_tuples[50].append(name_price_tuple_low)

# Iterate through each list to find the highest and lowest price in each state in each 1-year period
# repeat this for loop 50x, once for each state
for i in range(0, len(list_of_state_high_tuples[0])):
# for each state list, find the highest-priced city
for item in list_of_state_high_tuples[0]:
current_highest_city_for_this_state = ""
current_high_for_this_state = 0
if item[1] >= current_high_for_this_state:
current_high_for_this_state = item[1]
current_highest_city_for_this_state = item[0]
# TODO: Add a dict that maps state abbreviations to state names
if ((current_highest_city_for_this_state != "") and (current_high_for_this_state != 0)):
print("The city with the highest price in", current_highest_city_for_this_state[-2:], "is", current_highest_city_for_this_state, ".", "It had a high price of", current_high_for_this_state, "in", chosen_year)

for i in range(0, len(list_of_state_low_tuples)):
# for each state list, find the lowest-priced city
for item in list_of_state_low_tuples[i]:
current_lowest_city_for_this_state = ""
current_low_for_this_state = 0
```

```
if item[1] <= current_low_for_this_state:
current_low_for_this_state = item[1]
current_lowest_city_for_this_state = item[0]
if ((current_lowest_city_for_this_state != "") and (current_low_for_this_state != 0)):
print("The city with the lowest price in", current_lowest_city_for_this_state[-2:], "is", current_lowest_city_for_this_state, ".", "It had a low price of", current_low_for_this_state, "in", chosen_year)



```

inflation_derivatives.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sys
from datetime import datetime

if __name__ == "__main__":
acceptable_inputs = ['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP', 'OCT', 'NOV', 'DEC', 'Monthly_Average']
month_dict = {'JAN': 'January', 'FEB': 'February', 'MAR': 'March', 'APR': 'April', 'MAY': 'May', 'JUN': 'June', 'JUL': 'July', 'AUG': 'August', 'SEP': 'September', 'OCT': 'October', 'NOV':'November', 'DEC': 'December', 'Monthly_Average': 'Monthly_Average'}
month = sys.argv[1]
if month != "Monthly_Average": # input validation
month = month.upper()
if month not in acceptable_inputs:
print("Invalid input! Exiting...")
sys.exit()
inflation_df = pd.read_csv("Inflation_1996-2021.csv", parse_dates=[0]) # inflation dataset
month_series = inflation_df[month]
month_array = month_series.to_numpy('float64')
inflation_array = inflation_df[month].astype('float64')
first_derivative_list = []
current_year_and_first_deriv_tuple_list = []
second_derivative_list = []
current_year_and_second_deriv_tuple_list = []

for i in range(len(month_array) - 1):
intial_cpi = month_array[i]
final_cpi = month_array[i+1]
change_in_cpi = final_cpi - intial_cpi
initial_time = inflation_df.iloc[i, 0]
final_time = inflation_df.iloc[i+1, 0]
change_in_time = 1 # one year
dp_dt = change_in_cpi / change_in_time
```

```
first_derivative_list.append(dp_dt)
current_year = initial_time
current_year_and_first_deriv_tuple = (current_year, dp_dt)
current_year_and_first_deriv_tuple_list.append(current_year_and_first_deriv_tuple)

first_derivative_array = np.array(first_derivative_list)
for i in range(len(first_derivative_array) - 1):
intial_cpi = first_derivative_array[i]
final_cpi = first_derivative_array[i+1]
change_in_cpi = final_cpi - intial_cpi # this is actually the change in the derivative of price w.r.t time
initial_time = inflation_df.iloc[i, 0]
final_time = inflation_df.iloc[i+1, 0]
change_in_time = 1 # one year
dp_dt = change_in_cpi / change_in_time
second_derivative_list.append(dp_dt)
current_year = initial_time
current_year_and_second_deriv_tuple = (current_year, dp_dt)
current_year_and_second_deriv_tuple_list.append(current_year_and_second_deriv_tuple)

if month == 'Monthly_Average':
print("Table of the first derivative of CPI w.r.t. time for the inflation dataset for the monthly average.\n")
else:
print("Table of the first derivative of CPI w.r.t. time for the inflation dataset for the month of ", month_dict[month], '.\n', sep="")
print("------------------------")
for i in range(0, len(current_year_and_first_deriv_tuple_list)):

entry = current_year_and_first_deriv_tuple_list[i]
date_obj = entry[0]
deriv = entry[1]
deriv_2f = "{:.2f}".format(deriv)
print('|', date_obj.date(), '|', deriv_2f, '|')
#print("| ... | ... |")
print("------------------------")
print(len(current_year_and_first_deriv_tuple_list), 'rows.')
print('\n')

#plt.rcParams["figure.figsize"] = [7.00, 3.50]
#plt.rcParams["figure.autolayout"] = True
#x = np.arange(2000, 2022, 1, int)
#x = [2000, 2001, 2002, 2003, 2004, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022]
#y = first_derivative_list[0:-2]
#default_x_ticks = range(len(x))
#plt.plot(default_x_ticks, y)
#plt.xticks(default_x_ticks, x)
#plt.show()

if month == 'Monthly_Average':
```

```
print("Table of the first derivative of CPI w.r.t. time for the inflation dataset for the monthly average.\n")
else:
print("Table of the first derivative of CPI w.r.t. time for the inflation dataset for the month of ", month_dict[month], '.\n', sep="")
print("-------------------------")
for i in range(0, len(current_year_and_second_deriv_tuple_list)):
entry = current_year_and_second_deriv_tuple_list[i]
date_obj = entry[0]
deriv = entry[1]
deriv_2f = "{:.2f}".format(deriv)
print('|', date_obj.date(), '|', deriv_2f, '|')

#print("| ... | ... |")
print("-------------------------")
print(len(current_year_and_second_deriv_tuple_list), 'rows.')
print('\n')

plt.rcParams["figure.figsize"] = [7.00, 3.50]
plt.rcParams["figure.autolayout"] = True
x = np.arange(2000, 2022, 1, int)
x = [2000, 2001, 2002, 2003, 2004, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020]
y = second_derivative_list[0:-3]
default_x_ticks = range(len(x))
plt.plot(default_x_ticks, y)
plt.xticks(default_x_ticks, x)
plt.show()

```

ml1.py

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import datetime
import matplotlib.pyplot as plt
import sys

if __name__=="__main__":

# read the datasets into Pandas dataframes
djia_df = pd.read_csv("DJIA2000-2022FebCSV.csv", parse_dates=[0]) # DJIA opening/closing price dataset
cpi_df = pd.read_csv("Inflation_1996-2021.csv") # CPI inflation dataset
re_df = pd.read_csv("Metro_zhvi_uc_sfr_tier_0.33_0.67_sm_sa_month.csv") # Zillow real estate dataset

# extract the numeric features from each dataframe
numeric_feature_names_of_the_djia_df = ['Open', 'High', 'Low', 'Close*', 'Adj Close**', 'Volume']

```

```
numeric_feature_names_of_the_inflation_df = ['Year', 'JAN', 'FEB', 'MAR', 'APR',
'MAY', 'JUN', 'JUL', 'AUG', 'SEP', 'OCT', 'NOV', 'DEC', 'Monthly_Average']
numeric_feature_names_of_the_real_estate_df = ['1/31/1996', '2/29/1996', '3/31/1996'
, '4/30/1996', '5/31/1996', '6/30/1996', '7/31/1996', '8/31/1996', '9/30/1996',
'10/31/1996', '11/30/1996', '12/31/1996', '1/31/1997', '2/28/1997', '3/31/1997',
'4/30/1997', '5/31/1997', '6/30/1997', '7/31/1997', '8/31/1997', '9/30/1997',
'10/31/1997', '11/30/1997', '12/31/1997', '1/31/1998', '2/28/1998', '3/31/1998',
'4/30/1998', '5/31/1998', '6/30/1998', '7/31/1998', '8/31/1998', '9/30/1998',
'10/31/1998', '11/30/1998', '12/31/1998', '1/31/1999', '2/28/1999', '3/31/1999',
'4/30/1999', '5/31/1999', '6/30/1999', '7/31/1999', '8/31/1999', '9/30/1999',
'10/31/1999', '11/30/1999', '12/31/1999', '1/31/2000', '2/29/2000', '3/31/2000',
'4/30/2000', '5/31/2000', '6/30/2000', '7/31/2000', '8/31/2000', '9/30/2000',
'10/31/2000', '11/30/2000', '12/31/2000', '1/31/2001', '2/28/2001', '3/31/2001',
'4/30/2001', '5/31/2001', '6/30/2001', '7/31/2001', '8/31/2001', '9/30/2001',
'10/31/2001', '11/30/2001', '12/31/2001', '1/31/2002', '2/28/2002', '3/31/2002',
'4/30/2002', '5/31/2002', '6/30/2002', '7/31/2002', '8/31/2002', '9/30/2002',
'10/31/2002', '11/30/2002', '12/31/2002', '1/31/2003', '2/28/2003', '3/31/2003',
'4/30/2003', '5/31/2003', '6/30/2003', '7/31/2003', '8/31/2003', '9/30/2003',
'10/31/2003', '11/30/2003', '12/31/2003', '1/31/2004', '2/29/2004', '3/31/2004',
'4/30/2004', '5/31/2004', '6/30/2004', '7/31/2004', '8/31/2004', '9/30/2004',
'10/31/2004', '11/30/2004', '12/31/2004', '1/31/2005', '2/28/2005', '3/31/2005',
'4/30/2005', '5/31/2005', '6/30/2005', '7/31/2005', '8/31/2005', '9/30/2005',
'10/31/2005', '11/30/2005', '12/31/2005', '1/31/2006', '2/28/2006', '3/31/2006',
'4/30/2006', '5/31/2006', '6/30/2006', '7/31/2006', '8/31/2006', '9/30/2006',
'10/31/2006', '11/30/2006', '12/31/2006', '1/31/2007', '2/28/2007', '3/31/2007',
'4/30/2007', '5/31/2007', '6/30/2007', '7/31/2007', '8/31/2007', '9/30/2007',
'10/31/2007', '11/30/2007', '12/31/2007', '1/31/2008', '2/29/2008', '3/31/2008',
'4/30/2008', '5/31/2008', '6/30/2008', '7/31/2008', '8/31/2008', '9/30/2008',
'10/31/2008', '11/30/2008', '12/31/2008', '1/31/2009', '2/28/2009', '3/31/2009',
'4/30/2009', '5/31/2009', '6/30/2009', '7/31/2009', '8/31/2009', '9/30/2009',
'10/31/2009', '11/30/2009', '12/31/2009', '1/31/2010', '2/28/2010', '3/31/2010',
'4/30/2010', '5/31/2010', '6/30/2010', '7/31/2010', '8/31/2010', '9/30/2010',
'10/31/2010', '11/30/2010', '12/31/2010', '1/31/2011', '2/28/2011', '3/31/2011',
'4/30/2011', '5/31/2011', '6/30/2011', '7/31/2011', '8/31/2011', '9/30/2011',
'10/31/2011', '11/30/2011', '12/31/2011', '1/31/2012', '2/29/2012', '3/31/2012',
'4/30/2012', '5/31/2012', '6/30/2012', '7/31/2012', '8/31/2012', '9/30/2012',
'10/31/2012', '11/30/2012', '12/31/2012', '1/31/2013', '2/28/2013', '3/31/2013',
'4/30/2013', '5/31/2013', '6/30/2013', '7/31/2013', '8/31/2013', '9/30/2013',
'10/31/2013', '11/30/2013', '12/31/2013', '1/31/2014', '2/28/2014', '3/31/2014',
'4/30/2014', '5/31/2014', '6/30/2014', '7/31/2014', '8/31/2014', '9/30/2014',
'10/31/2014', '11/30/2014', '12/31/2014', '1/31/2015', '2/28/2015', '3/31/2015',
'4/30/2015', '5/31/2015', '6/30/2015', '7/31/2015', '8/31/2015', '9/30/2015',
'10/31/2015', '11/30/2015', '12/31/2015', '1/31/2016', '2/29/2016', '3/31/2016',
'4/30/2016', '5/31/2016', '6/30/2016', '7/31/2016', '8/31/2016', '9/30/2016',
'10/31/2016', '11/30/2016', '12/31/2016', '1/31/2017', '2/28/2017', '3/31/2017',
'4/30/2017', '5/31/2017', '6/30/2017', '7/31/2017', '8/31/2017', '9/30/2017',
'10/31/2017', '11/30/2017', '12/31/2017', '1/31/2018', '2/28/2018', '3/31/2018',

'4/30/2018', '5/31/2018', '6/30/2018', '7/31/2018', '8/31/2018', '9/30/2018',
'10/31/2018', '11/30/2018', '12/31/2018', '1/31/2019', '2/28/2019', '3/31/2019',
'4/30/2019', '5/31/2019', '6/30/2019', '7/31/2019', '8/31/2019', '9/30/2019',
'10/31/2019', '11/30/2019', '12/31/2019', '1/31/2020', '2/29/2020', '3/31/2020',
'4/30/2020', '5/31/2020', '6/30/2020', '7/31/2020', '8/31/2020', '9/30/2020',
'10/31/2020', '11/30/2020', '12/31/2020', '1/31/2021', '2/28/2021', '3/31/2021', '4/30/2021', '5/31/2021', '6/30/2021', '7/31/2021', '8/31/2021']

   numeric_features_of_djia_df = djia_df[numeric_feature_names_of_the_djia_df]
   numeric_features_of_inflation_df = cpi_df[numeric_feature_names_of_the_inflation_df]
   numeric_features_of_real_estate_df = re_df[numeric_feature_names_of_the_real_estate_df]

   djia_array = numeric_features_of_djia_df.to_numpy()
   cpi_array = numeric_features_of_inflation_df.to_numpy()
   re_array = numeric_features_of_real_estate_df.to_numpy()

   print('djia_array.shape:', djia_array.shape)
   print('cpi_array.shape:', cpi_array.shape)
   print('re_array.shape:', re_array.shape)

   # plot each dataset as a timeseries in matplotlib
   if sys.argv[1] == 're':
   # plot the real estate dataset
   list_of_columns = list(re_df.columns)
   print(list_of_columns)

   if sys.argv[1] == 'djia':
   # plot the DJIA dataset
   djia_df["Date"] = djia_df["Date"].astype("datetime64")
   djia_df = djia_df.set_index("Date")
   djia_df['Close*']=djia_df['Close*'].str.replace(',','')
   djia_df['Close*']=djia_df['Close*'].str.replace('.','')
   djia_df["Close*"] = djia_df["Close*"].astype("float64")
   plt.plot(djia_df["Close*"], marker='o')
   plt.xlabel("Date")
   plt.ylabel("Closing Price")
   plt.title("Plot of Daily Closing Price vs. Time for the Dow Jones Industrial Average from 2000 to 2021")
   plt.show()

   if sys.argv[1] == 'cpi':
   # plot a dataset
   cpi_df["Year"] = cpi_df["Year"].astype("datetime64[ns]")
   cpi_df = cpi_df.set_index("Year")
   plt.plot(cpi_df["Monthly_Average"], marker='o')
   plt.xlabel("Date")
   plt.ylabel("Annual Inflation Rate")
   plt.title("Plot of the Consumer Price Index (CPI) vs. Time from 2000 to 2021")
   plt.show()
```

plot_data.py

```
#!/usr/bin/env python3

import pandas as pd
import matplotlib.pyplot as plt
import sys
import os
import numpy as np
from dateutil import parser

if __name__=="__main__":
filename = sys.argv[1]
choice = sys.argv[2]
col_name = str(choice)
acc_inputs = ['Open','High','Low','Close*','Adj Close**','Volume', 'Year', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP', 'OCT', 'NOV', 'DEC', 'Monthly_Average']
if col_name not in acc_inputs:
print("Invalid input!!! Exiting...")
sys.exit()
df = pd.read_csv(str(filename))

if filename == "Inflation_1999-2021.csv":
if str(choice) == 'Year':
selected_year = input('Enter a year in range [1996, 2021]: ')
int_selected_year = int(selected_year)
# 1996 is index 0
index = int_selected_year - 1996
# print the row for that year
subset = df.iloc[index,:]
else:
selected_month = str(choice)
first_col = df['Year']
second_col = df[selected_month]
subset = pd.concat([first_col, second_col], axis=1)
print(subset)

if filename == "DJIA2000-2022FebCSV.csv":
print(df.head())
print()
print('There are', df.shape[0], 'rows in the dataset.')
print('There are', df.shape[1], 'columns in the dataset.')
dates = df["Date"]
d_list = dates.to_list()
ndl = []
for d in d_list:
numb = parser.parse(d)
ndl.append(numb)
nda = np.array(ndl)

closing_prices = df[col_name]
cpl = closing_prices.to_list()
cpf = []
for x in cpl:
num = x.replace(',', '')
cpf.append(float(num))
cpa = np.array(cpf)
title_str = 'Plot of Year vs. Closing Price for the DJIA'
```

```
plt.title(title_str)
plt.plot(nda, cpa)
plt.show()

if filename == "statewide-covid-19-cases-deaths-tests.csv":
print(df.head())
print()
print('There are', df.shape[0], 'rows in the dataset.')
print('There are', df.shape[1], 'columns in the dataset.')
dates = df["Date"]

d_list = dates.to_list()
ndl = []
for d in d_list:
numb = parser.parse(d)
ndl.append(numb)
nda = np.array(ndl)

closing_prices = df[col_name]
cpl = closing_prices.to_list()
cpf = []
for x in cpl:
num = x.replace(',', '')
cpf.append(float(num))
cpa = np.array(cpf)
title_str = 'Plot of Date vs. '+col_name+' for the DJIA'
plt.title(title_str)
plt.plot(nda, cpa)
plt.show()
```

process_dija.py

```
#!/usr/bin/env python3

import pandas as pd
import matplotlib.pyplot as plt
import sys
import os
import numpy as np
from dateutil import parser

date_format = "%m/%d/%Y"
COASTAL = False # determines whether we include only cities in coastal or non-coastal states in our analysis

def Rank_Cities_in_Descending_Order(t):
t.sort(key = lambda x: x[1], reverse=True)
return t

if __name__=="__main__":

filename = sys.argv[1]
```

```
df = pd.read_csv(str(filename))
col_name = "Close*"
#print(df.head())
print()
#print('There are', df.shape[0], 'rows in the dataset.')
#print('There are', df.shape[1], 'columns in the dataset.')
dates = df["Date"]
d_list = dates.to_list()
ndl = []
for d in d_list:
numb = parser.parse(d)
ndl.append(numb)
nda = np.array(ndl)

# calculate the stock market appreciation rate for selected year

closing_prices = df[col_name]

percentage_change_dict = {}

# 2000
last_day_price = df["Close*"][5315 - 2]
first_day_price = df["Close*"][5566 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2000] = percentage_change_2f

# 2001
last_day_price = df["Close*"][5067 - 2]
first_day_price = df["Close*"][5314 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2001] = percentage_change_2f

# 2002
last_day_price = df["Close*"][4815 - 2]
first_day_price = df["Close*"][5066 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100

percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2002] = percentage_change_2f

# 2003
last_day_price = df["Close*"][4563 - 2]
```

```
first_day_price = df["Close*"][4814 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2003] = percentage_change_2f

# 2004
last_day_price = df["Close*"][4311 - 2]
first_day_price = df["Close*"][4562 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2004] = percentage_change_2f

# 2005
last_day_price = df["Close*"][4059 - 2]
first_day_price = df["Close*"][4310 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2005] = percentage_change_2f

# 2006
last_day_price = df["Close*"][3808 - 2]
first_day_price = df["Close*"][4058 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2006] = percentage_change_2f

# 2007
last_day_price = df["Close*"][3557 - 2]
first_day_price = df["Close*"][3807 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2007] = percentage_change_2f

# 2008
last_day_price = df["Close*"][3304 - 2]
first_day_price = df["Close*"][3556 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
```

```
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2008] = percentage_change_2f

# 2009
last_day_price = df["Close*"][3052 - 2]
first_day_price = df["Close*"][3303 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2009] = percentage_change_2f

# 2010
last_day_price = df["Close*"][2800 - 2]
first_day_price = df["Close*"][3051 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2010] = percentage_change_2f

# 2011
last_day_price = df["Close*"][2548 - 2]
first_day_price = df["Close*"][2799 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2011] = percentage_change_2f

# 2012
last_day_price = df["Close*"][2298 - 2]
first_day_price = df["Close*"][2547 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2012] = percentage_change_2f

# 2013
last_day_price = df["Close*"][2046 - 2]
first_day_price = df["Close*"][2297 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
```

```
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2013] = percentage_change_2f

# 2014
last_day_price = df["Close*"][1794 - 2]
first_day_price = df["Close*"][2045 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2014] = percentage_change_2f

# 2015
last_day_price = df["Close*"][1542 - 2]
first_day_price = df["Close*"][1793 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2015] = percentage_change_2f

# 2016

last_day_price = df["Close*"][1290 - 2]
first_day_price = df["Close*"][1541 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2016] = percentage_change_2f

# 2017
last_day_price = df["Close*"][1039 - 2]
first_day_price = df["Close*"][1289 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2017] = percentage_change_2f

# 2018
last_day_price = df["Close*"][788 - 2]
first_day_price = df["Close*"][1038 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2018] = percentage_change_2f
```

```

# 2019
last_day_price = df["Close*"][536 - 2]
first_day_price = df["Close*"][787 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2019] = percentage_change_2f

# 2020
last_day_price = df["Close*"][283 - 2]
first_day_price = df["Close*"][535 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2020] = percentage_change_2f

# 2021
last_day_price = df["Close*"][31 - 2]
first_day_price = df["Close*"][282 - 2]
last_day_price_no_comma = last_day_price.replace(',', '')
first_day_price_no_comma = first_day_price.replace(',','')
diff = float(last_day_price_no_comma) - float(first_day_price_no_comma)
percentage_change = (diff / float(first_day_price_no_comma)) * 100
percentage_change_2f = "{:.2f}".format(percentage_change)
percentage_change_dict[2021] = percentage_change_2f

input_year = input("Which year do you want to search for? Enter a year in [2000, 2010]: ")
if (int(input_year) < 2000) or (int(input_year) > 2010): # check the years 2010-2020
print("Year outside acceptable range. Exiting...")
sys.exit()
coastal_preference = input("Do you want cities in coastal or non-coastal states? ")
coastal_preference = coastal_preference.upper()
if coastal_preference == "COASTAL":

COASTAL = True
else:
COASTAL = False
print("In ", int(input_year), ", the change in the Dow Jones Industrial Average was: ", percentage_change_dict[int(input_year)], " percent.", sep ="")

# calculate the real estate appreciation rate for selected year(s)

# add color to the terminal
os.system('color')

mode = 1
period_length = 1
```

```

period_name = input_year

# process the dataframe
original_df = pd.read_csv("Metro_zhvi_uc_sfr_tier_0.33_0.67_sm_sa_month.csv")

# pre-processing steps
# step 1: get the column names
complete_rows_df = original_df
size_rank_column_df = original_df[['SizeRank']]
region_name_column_df = original_df[['RegionName']]
size_rank_column_series = size_rank_column_df.squeeze()
region_name_column_series = region_name_column_df.squeeze()
region_name_column_list = region_name_column_series.tolist()
number_of_rows = original_df.shape[0]
city_name_size_rank_dict = {}
for i in range(0, number_of_rows):
city_name_size_rank_dict[region_name_column_series[i]] = size_rank_column_series[i]

# step 2: drop "RegionID", "SizeRank", "RegionType", and "StateName" columns
df = complete_rows_df.drop(columns=['RegionID', 'SizeRank', 'RegionType', 'StateName'])
df = df.iloc[1: , :] # remove the first row because it doesn't have a state
if COASTAL == True:
df = df[df["RegionName"].str.contains("CA|OR|WA|AK|HI|ME|NH|MA|RI|CT|NY|NJ|DE|MD|VA|NC|SC|GA|FL|TX|LA|MS|AL")]
else:
df = df[df["RegionName"].str.contains("ID|MT|ND|MN|WI|MI|VT|WY|SD|IA|PA|NV|UT|CO|NE|IL|IN|OH|KS|MO|KY|WV|AZ|NM|OK|AR|TN|DC")]

str_number_of_cities_to_list = df.shape[0]

# create a numeric dataframe consisting of only floats
float_df = df.drop(columns=['RegionName'])

periods = []
period_dict = {}

# partition the floating point dataframe into 25 1-year periods
for i in range(0,180,12):
period = float_df.iloc[:, i:i+12]
periods.append(period)
period_dict[0] = '1996'
period_dict[1] = '1997'
period_dict[2] = '1998'
period_dict[3] = '1999'
period_dict[4] = '2000'
period_dict[5] = '2001'
period_dict[6] = '2002'
period_dict[7] = '2003'
period_dict[8] = '2004'
```

```
period_dict[9] = '2005'

period_dict[10] = '2006'
period_dict[11] = '2007'
period_dict[12] = '2008'
period_dict[13] = '2009'
period_dict[14] = '2010'
period_dict[15] = '2011'
period_dict[16] = '2012'
period_dict[17] = '2013'
period_dict[18] = '2014'
period_dict[19] = '2015'
period_dict[20] = '2016'
period_dict[21] = '2017'
period_dict[22] = '2018'
period_dict[23] = '2019'
period_dict[24] = '2020'

# get the city names and put them in a list
city_names_list = df['RegionName'].values.tolist()
big_city_list = city_names_list[1:40]
all_period_list = []
for period in periods:
# declare a list to store the normalized price increase for each city for that period
abs_price_inc_list = []
for i in range(0, len(period)):
first = i
last = i+1
row = period.iloc[first:last, :]
# convert row to Panadas series
row_series = row.squeeze(axis=0)
rowmax = row_series.max()
# normalized row for a city
norm_series = row_series.divide(rowmax)
limit = len(norm_series) - 1
first_element_in_row = norm_series[0]
last_element_in_row = norm_series[limit]
difference_between_the_elements = last_element_in_row - first_element_in_row
city_names_list[i]
name_price_tuple = (city_names_list[i], difference_between_the_elements)
abs_price_inc_list.append(name_price_tuple)
sorted_period_list = Rank_Cities_in_Descending_Order(abs_price_inc_list)
all_period_list.append(sorted_period_list)
for k in range (0, len(all_period_list) - 1):
# get the kth period's sorted list of city name-price increase pairs
current_period = all_period_list[k]
highest_city_price_inc_pair = current_period[0]
highest_city = highest_city_price_inc_pair[0]
highest_city_increase = highest_city_price_inc_pair[1]

fp_number_of_cities_to_list = float(str_number_of_cities_to_list)
number_of_cities_to_list = int(np.ceil(fp_number_of_cities_to_list))
```

```
period_list = list(period_dict.keys())
year_list = list(period_dict.values())
idx = year_list.index(period_name)
period_number = period_list[idx]
city_names_list = df['RegionName'].values.tolist()
# input validation
if number_of_cities_to_list <= 0:
print("You are trying to list too few cities! Exiting...")
sys.exit()
if number_of_cities_to_list > len(city_names_list):
print("You are trying to list more cities than there are in the dataset! Exiting...")
sys.exit()
period = periods[period_number]
# declare a list to store the normalized price increase for each city for that period
abs_price_inc_list = []

for i in range(0, len(period)):
first = i
last = i+1
row = period.iloc[first:last, :]
# convert row to Pandas series
row_series = row.squeeze(axis=0)
rowmax = row_series.max()
# normalized row for a city
norm_series = row_series.divide(rowmax)
limit = len(norm_series) - 1
first_element_in_row = norm_series[0]
last_element_in_row = norm_series[limit]
difference_between_the_elements = last_element_in_row - first_element_in_row
city_names_list[i]
name_price_tuple = (city_names_list[i], difference_between_the_elements)
abs_price_inc_list.append(name_price_tuple)
sorted_list = Rank_Cities_in_Descending_Order(abs_price_inc_list)
avg_inc_in_normalized_price_list = []
for j in range (0, number_of_cities_to_list):
highest_city_price_inc_pair = sorted_list[j]
highest_city = highest_city_price_inc_pair[0]
highest_city_increase = highest_city_price_inc_pair[1]
avg_inc_in_normalized_price_list.append(highest_city_increase)
avg_inc_in_normalized_price_array = np.array(avg_inc_in_normalized_price_list)
avg_inc_in_normalized_price = np.nanmean(avg_inc_in_normalized_price_array)
avg_inc_in_normalized_price_2f = "{:.2f}".format(avg_inc_in_normalized_price * 100)
print("The average increase in the normalized median price of single family homes was: ", avg_inc_in_normalized_price_2f, " percent.", sep="")

# calculate the inflation rate for the selected year(s)

filename2 = sys.argv[2]
df2 = pd.read_csv(str(filename2))
int_selected_year = int(input_year)
selected_month = "Monthly_Average"
first_col = df2['Year']
```

```
second_col = df2[selected_month]
subset = pd.concat([first_col, second_col], axis=1)
# 1996 is index 0
index = int_selected_year - 1996
row = df2[index:index+1]
inflation_rate_df = row['Monthly_Average']
inflation_rate = inflation_rate_df.to_string(index=False)
print("The inflation rate was", inflation_rate, " percent.", sep="")
```

re1.py

```
#!/usr/bin/env python

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sys
import os
import termcolor as tm
import scipy.stats
from datetime import datetime
from OSMPythonTools.nominatim import Nominatim

# named constants

date_format = "%m/%d/%Y"

# sorting functions

def Rank_Cities_in_Descending_Order(t):
t.sort(key = lambda x: x[1], reverse=True)
return t

def Rank_Cities_in_Ascending_Order(t):
t.sort(key = lambda x: x[1], reverse=False)
return t
#TODO: Add a dict that maps user-friendly string command line arguments to numeric modes

if __name__ == "__main__":
# add color to the terminal
os.system('color')
# take command line arguments
cmd = str(sys.argv[1])
mode = int(cmd)
period_length = 10 # default value
# choose mode and take corresponding user input
if (mode < 0) or (mode > 20):
print('Invalid command line argument! Exiting...')
sys.exit()
if mode == 1:
period_length_arg = input('Select period length of 1, 2, 5, or 10 years: ')
period_length = int(period_length_arg)
```

```
# input validation
if (period_length < 1) or (period_length > 10):
period_length = 10 # default value
city1 = input("Enter a city and state: ")
elif mode == 2:
period_length_arg = input('Select period length of 1, 2, 5, or 10 years: ')
period_length = int(period_length_arg)
# input validation
if (period_length < 1) or (period_length > 10):
period_length = 10 # default value
if period_length == 1:
period_name = input("Enter a 1-year period between 1996 and 2020 (in YYYY format): ")
elif period_length == 2:
period_name = input("Enter a 2-year period between 1996 and 2020 (in YYYY-YYYY format): ")
elif period_length == 5:
period_name = input("Enter a 5-year period between 1996 and 2020 (in YYYY-YYYY format): ")
elif period_length == 10:
period_name = input("Enter a 10-year period between 1996 and 2020 (in YYYY-YYYY format): ")
str_number_of_cities_to_list = input("How many cities do you want to list? ")
elif mode == 3:
amount = input("Enter an amount to invest (in thousands): ")
elif mode == 4:

city1 = input("Enter a city and state: ")
city2 = input("Enter another city and state: ")
elif mode == 5:
city1 = input("Enter a city and state: ")
elif mode == 6:
city1 = input("Enter a city and state: ")
elif mode == 7:
city1 = input("Enter a city and state: ")
elif mode == 8:
city1 = input("Enter a city and state: ")
str_requested_year = input("Enter a year: ")
requested_year = int(str_requested_year)
elif mode == 9:
city1 = input("Enter a city and state: ")
str_requested_years = input("Enter a series of years (separated by spaces): ")
requested_years_string_list = list(str_requested_years.split(' '))
requested_years = [int(i) for i in requested_years_string_list]
elif mode == 10:
city1 = input("Enter a city and state: ")
elif mode == 11:
city1 = input("Enter a city and state: ")
elif mode == 13:
upper_bound_str = input("Enter the upper bound: ")
lower_bound_str = input("Enter the lower bound: ")
upper_bound = float(upper_bound_str)
lower_bound = float(lower_bound_str)
```

```
if lower_bound > upper_bound:
print("Invalid input!!!")
sys.exit(0)
elif mode == 15:
city1 = input("Enter a city and state: ")

# process the dataframe
original_df = pd.read_csv("Metro_zhvi_uc_sfr_tier_0.33_0.67_sm_sa_month.csv")
senators_df = pd.read_csv("Senators_Political_Affiliations_1996_2021.csv")

# define the regions of the United States
west_coast_states = ["WA", "OR", "CA", "NV", "AZ", "AK", "HI"]
rocky_mountain_states = ["MT", "ID", "WY", "UT", "CO"]
gulf_coast_states = ["NM", "TX", "LA", "AR", "MS", "AL"]
midwest_states = ["ND", "SD", "NE", "KS", "OK", "MN", "IA", "MO", "WI", "IL", "IN", "OH", "MI", "KY", "TN"]
east_coast_states = ["ME", "VT", "NH", "MA", "CT", "RI", "NY", "PA", "NJ", "MD", "DE", "WV", "VA", "NC", "SC", "GA", "FL"]

#index 0 is 1996
AL_series = senators_df["AL"]
AL_senators = AL_series.tolist()

AR_series = senators_df["AR"]
AR_senators = AR_series.tolist()

AZ_series = senators_df["AZ"]
AZ_senators = AZ_series.tolist()

CA_series = senators_df["CA"]
CA_senators = CA_series.tolist()

CO_series = senators_df["CO"]
CO_senators = CO_series.tolist()

CT_series = senators_df["CT"]
CT_senators = CT_series.tolist()

FL_series = senators_df["FL"]
FL_senators = FL_series.tolist()

GA_series = senators_df["GA"]
GA_senators = GA_series.tolist()

HI_series = senators_df["HI"]
HI_senators = HI_series.tolist()

IA_series = senators_df["IA"]
IA_senators = IA_series.tolist()

ID_series = senators_df["ID"]
ID_senators = ID_series.tolist()
```

```

IL_series = senators_df["IL"]
IL_senators = IL_series.tolist()

IN_series = senators_df["IN"]
IN_senators = IN_series.tolist()

KS_series = senators_df["KS"]
KS_senators = KS_series.tolist()

KY_series = senators_df["KY"]
KY_senators = KY_series.tolist()

LA_series = senators_df["LA"]
LA_senators = LA_series.tolist()

ME_series = senators_df["ME"]
ME_senators = ME_series.tolist()

MD_series = senators_df["MD"]
MD_senators = MD_series.tolist()

MA_series = senators_df["MA"]
MA_senators = MA_series.tolist()

MI_series = senators_df["MI"]
MI_senators = MI_series.tolist()

MN_series = senators_df["MN"]
MN_senators = MN_series.tolist()

MS_series = senators_df["MS"]
MS_senators = MS_series.tolist()

MO_series = senators_df["MO"]
MO_senators = MO_series.tolist()

NE_series = senators_df["NE"]
NE_senators = NE_series.tolist()

NV_series = senators_df["NV"]
NV_senators = NV_series.tolist()

NJ_series = senators_df["NJ"]
NJ_senators = NJ_series.tolist()

NM_series = senators_df["NM"]
NM_senators = NM_series.tolist()

NY_series = senators_df["NY"]
NY_senators = NY_series.tolist()

NC_series = senators_df["NC"]
```

```
NC_senators = NC_series.tolist()

OH_series = senators_df["OH"]
OH_senators = OH_series.tolist()

OK_series = senators_df["OK"]
OK_senators = OK_series.tolist()

OR_series = senators_df["OR"]
OR_senators = OR_series.tolist()

PA_series = senators_df["PA"]
PA_senators = PA_series.tolist()

RI_series = senators_df["RI"]
RI_senators = RI_series.tolist()

SC_series = senators_df["SC"]
SC_senators = SC_series.tolist()

TN_series = senators_df["TN"]
TN_senators = TN_series.tolist()

TX_series = senators_df["TX"]
TX_senators = TX_series.tolist()

UT_series = senators_df["UT"]
UT_senators = UT_series.tolist()

VA_series = senators_df["VA"]
VA_senators = VA_series.tolist()

WA_series = senators_df["WA"]
WA_senators = WA_series.tolist()

WI_series = senators_df["WI"]
WI_senators = WI_series.tolist()

# pre-processing steps
# step 1: get the column names
complete_rows_df = original_df
size_rank_column_df = original_df[['SizeRank']]
region_name_column_df = original_df[['RegionName']]
size_rank_column_series = size_rank_column_df.squeeze()
region_name_column_series = region_name_column_df.squeeze()
region_name_column_list = region_name_column_series.tolist()
number_of_rows = original_df.shape[0]
city_name_size_rank_dict = {}
for i in range(0, number_of_rows):
city_name_size_rank_dict[region_name_column_series[i]] = size_rank_column_series[i]

```

```
# step 2: drop "RegionID", "SizeRank", "RegionType", and "StateName" columns
df = complete_rows_df.drop(columns=['RegionID', 'SizeRank', 'RegionType', 'StateName'])

# create a numeric dataframe consisting of only floats
float_df = df.drop(columns=['RegionName'])

periods = []
period_dict = {}

if period_length == 1:
# partition the floating point dataframe into 25 1-year periods
for i in range(0,180,12):
period = float_df.iloc[:, i:i+12]
periods.append(period)
period_dict[0] = '1996'
period_dict[1] = '1997'
period_dict[2] = '1998'
period_dict[3] = '1999'
period_dict[4] = '2000'
period_dict[5] = '2001'
period_dict[6] = '2002'
period_dict[7] = '2003'
period_dict[8] = '2004'
period_dict[9] = '2005'
period_dict[10] = '2006'
period_dict[11] = '2007'
period_dict[12] = '2008'
period_dict[13] = '2009'
period_dict[14] = '2010'
period_dict[15] = '2011'
period_dict[16] = '2012'
period_dict[17] = '2013'
period_dict[18] = '2014'
period_dict[19] = '2015'
period_dict[20] = '2016'
period_dict[21] = '2017'
period_dict[22] = '2018'
period_dict[23] = '2019'
period_dict[24] = '2020'

if period_length == 2:
# partition the floating point dataframe into 25 2-year periods
for i in range(0,180,12):
period = float_df.iloc[:, i:i+24]
periods.append(period)
period_dict[0] = '1996-1997'
period_dict[1] = '1997-1998'
period_dict[2] = '1998-1999'
period_dict[3] = '1999-2000'
period_dict[4] = '2000-2001'
period_dict[5] = '2001-2002'
```

```
period_dict[6] = '2002-2003'
period_dict[7] = '2003-2004'
period_dict[8] = '2004-2005'
period_dict[9] = '2005-2006'
period_dict[10] = '2006-2007'
period_dict[11] = '2007-2008'
period_dict[12] = '2008-2009'
period_dict[13] = '2009-2010'
period_dict[14] = '2010-2011'
period_dict[15] = '2011-2012'
period_dict[16] = '2012-2013'
period_dict[17] = '2013-2014'
period_dict[18] = '2014-2015'
period_dict[19] = '2015-2016'
period_dict[20] = '2016-2017'
period_dict[21] = '2017-2018'
period_dict[22] = '2018-2019'
period_dict[23] = '2019-2020'
period_dict[24] = '2020-2021'

if period_length == 5:
# partition the floating point dataframe into 20 5-year periods
for i in range(0,252,12):
period = float_df.iloc[:, i:i+60]
periods.append(period)
period_dict[0] = '1996-2001'
period_dict[1] = '1997-2002'
period_dict[2] = '1998-2003'
period_dict[3] = '1999-2004'
period_dict[3] = '2000-2005'
period_dict[4] = '2001-2006'
period_dict[5] = '2002-2007'
period_dict[6] = '2003-2008'

period_dict[7] = '2004-2009'
period_dict[8] = '2005-2010'
period_dict[9] = '2006-2011'
period_dict[10] = '2007-2012'
period_dict[11] = '2008-2013'
period_dict[12] = '2009-2014'
period_dict[13] = '2010-2015'
period_dict[14] = '2011-2016'
period_dict[15] = '2012-2017'
period_dict[16] = '2013-2018'
period_dict[17] = '2014-2019'
period_dict[18] = '2015-2020'
period_dict[19] = '2016-2021'

if period_length == 10:
# partition the floating point dataframe into 15 10-year periods
for i in range(0,180,12):
period = float_df.iloc[:, i:i+132]
periods.append(period)
```

```
period_dict[0] = '1996-2006'
period_dict[1] = '1997-2007'
period_dict[2] = '1998-2008'
period_dict[3] = '1999-2009'
period_dict[4] = '2000-2010'
period_dict[5] = '2001-2011'
period_dict[6] = '2002-2012'
period_dict[7] = '2003-2013'
period_dict[8] = '2004-2014'
period_dict[9] = '2005-2015'
period_dict[10] = '2006-2016'
period_dict[11] = '2007-2017'
period_dict[12] = '2008-2018'
period_dict[13] = '2009-2019'
period_dict[14] = '2010-2020'

# get the city names and put them in a list
city_names_list = df['RegionName'].values.tolist()
big_city_list = city_names_list[1:40]
all_period_list = []
for period in periods:
# declare a list to store the normalized price increase for each city for that period
abs_price_inc_list = []
for i in range(0, len(period)):
first = i
last = i+1
row = period.iloc[first:last, :]
# convert row to Pandas series
row_series = row.squeeze(axis=0)
rowmax = row_series.max()
# normalized row for a city
norm_series = row_series.divide(rowmax)
limit = len(norm_series) - 1
first_element_in_row = norm_series[0]
last_element_in_row = norm_series[limit]
difference_between_the_elements = last_element_in_row - first_element_in_row
city_names_list[i]
name_price_tuple = (city_names_list[i], difference_between_the_elements)
abs_price_inc_list.append(name_price_tuple)
sorted_period_list = Rank_Cities_in_Descending_Order(abs_price_inc_list)
all_period_list.append(sorted_period_list)
for k in range (0, len(all_period_list) - 1):
# get the kth period's sorted list of city name-price increase pairs
current_period = all_period_list[k]
highest_city_price_inc_pair = current_period[0]
highest_city = highest_city_price_inc_pair[0]
highest_city_increase = highest_city_price_inc_pair[1]

# the modes
if mode == 1:
current_highest_price = 0
```

```
index_of_best_period = 0
for i in range (0, len(all_period_list) - 1):
current_period = all_period_list[i]
for j in range (0, len(current_period)):
this_tuple = current_period[j]
if this_tuple[0] == city1:
price_in_period_i = this_tuple[1]
if price_in_period_i > current_highest_price:
current_highest_price = price_in_period_i
index_of_best_period = i
current_highest_price_3f = "{:.3f}".format (current_highest_price)
print('The best ', period_length, '-year span to invest in ', city1, ' was ', period_dict[index_of_best_period], '. It had an increase in normalized price of ', current_highest_price_3f, ' during that period.', sep='')

if mode == 2:
fp_number_of_cities_to_list = float(str_number_of_cities_to_list)
number_of_cities_to_list = int(np.ceil(fp_number_of_cities_to_list))
period_list = list(period_dict.keys())
year_list = list(period_dict.values())
idx = year_list.index(period_name)
period_number = period_list[idx]
city_names_list = df['RegionName'].values.tolist()
# input validation
if number_of_cities_to_list <= 0:
print("You are trying to list too few cities! Exiting...")
sys.exit()
if number_of_cities_to_list > len(city_names_list):
print("You are trying to list more cities than there are in the dataset! Exiting...")
sys.exit()
period = periods[period_number]
# declare a list to store the normalized price increase for each city for that period
abs_price_inc_list = []
for i in range(0, len(period)):
first = i
last = i+1
row = period.iloc[first:last, :]
# convert row to Pandas series
row_series = row.squeeze(axis=0)
rowmax = row_series.max()
# normalized row for a city
norm_series = row_series.divide(rowmax)
limit = len(norm_series) - 1
first_element_in_row = norm_series[0]
last_element_in_row = norm_series[limit]
difference_between_the_elements = last_element_in_row - first_element_in_row
city_names_list[i]
name_price_tuple = (city_names_list[i], difference_between_the_elements)
abs_price_inc_list.append(name_price_tuple)
sorted_list = Rank_Cities_in_Descending_Order(abs_price_inc_list)
print("The", number_of_cities_to_list, "highest performing cities in",
period_name, 'were:\n')
for j in range (0, number_of_cities_to_list):
highest_city_price_inc_pair = sorted_list[j]
highest_city = highest_city_price_inc_pair[0]
highest_city_increase = highest_city_price_inc_pair[1]
highest_city_increase_3f = "{:.3f}".format(highest_city_increase)
print(j+1, highest_city, 'which had an increase in normalized price of',
highest_city_increase_3f)

if mode == 3:

# filter the rows of the dataframe based on the initial amount entered
reduced_df = float_df.loc[float_df['1/31/1996'] <= ( float(amount) * 1000) ]
# get the city names and put them in a list
city_names_list = df['RegionName'].values.tolist()
# declare a list to store the normalized price increase for each city
abs_price_inc_list = []
for i in range(0, reduced_df.shape[0]):
first = i
last = i+1
row = reduced_df.iloc[first:last, :]
# convert row to Pandas series
row_series = row.squeeze(axis=0)
rowmax = row_series.max()
# normalized row for a city
norm_series = row_series.divide(rowmax)
limit = len(norm_series) - 8
first_element_in_row = norm_series[0]
last_element_in_row = norm_series[limit]
difference_between_the_elements = last_element_in_row - first_element_in_row
city_names_list[i]
name_price_tuple = (city_names_list[i], difference_between_the_elements)
abs_price_inc_list.append(name_price_tuple)
sorted_list = Rank_Cities_in_Descending_Order(abs_price_inc_list)
highest_city_price_inc_pair = sorted_list[0]
highest_city = highest_city_price_inc_pair[0]
highest_city_increase = highest_city_price_inc_pair[1]
highest_city_increase_3f = "{:.3f}".format(highest_city_increase)
print('The best place to invest', amount, 'thousand dollars in 1996-2020 was',
highest_city, 'which had an increase in normalized price of',
highest_city_increase_3f)

if mode == 4:
# get the city names and put them in a list
city_names_list = df['RegionName'].values.tolist()
# declare a list to store the normalized price increase for each city
abs_price_inc_list = []
for i in range(0, float_df.shape[0]):
first = i
last = i+1
row = float_df.iloc[first:last, :]
# convert row to Panadas series
row_series = row.squeeze(axis=0)
```

```
rowmax = row_series.max()
# normalized row for a city
norm_series = row_series.divide(rowmax)
limit = len(norm_series) - 8
first_element_in_row = norm_series[0]
last_element_in_row = norm_series[limit]
difference_between_the_elements = last_element_in_row - first_element_in_row
name_price_tuple = (city_names_list[i], difference_between_the_elements)
abs_price_inc_list.append(name_price_tuple)
sorted_list = Rank_Cities_in_Descending_Order(abs_price_inc_list)

current_highest_price_for_city1 = 0
current_highest_price_for_city2 = 0
for j in range (0, len(sorted_list)):
this_tuple = sorted_list[j]
if this_tuple[0] == city1:
price_in_period_i = this_tuple[1]
if price_in_period_i > current_highest_price_for_city1:
current_highest_price_for_city1 = price_in_period_i
if this_tuple[0] == city2:
price_in_period_j = this_tuple[1]
if price_in_period_j > current_highest_price_for_city2:
current_highest_price_for_city2 = price_in_period_j
current_highest_price_for_city1_3f = "{:.3f}".format(
current_highest_price_for_city1)

current_highest_price_for_city2_3f = "{:.3f}".format(
current_highest_price_for_city2)
if current_highest_price_for_city1 > current_highest_price_for_city2:
print(city1, 'was a better place to invest than', city2, 'in 1996-2020.')
print(city1, 'had an increase in normalized price of',
current_highest_price_for_city1_3f)
print(city2, 'had an increase in normalized price of',
current_highest_price_for_city2_3f)
if current_highest_price_for_city2 > current_highest_price_for_city1:
print(city2, 'was a better place to invest than', city1, 'in 1996-2020.')
print(city2, 'had an increase in normalized price of',
current_highest_price_for_city2_3f)
print(city1, 'had an increase in normalized price of',
current_highest_price_for_city1_3f)

if mode == 5:
# Find how long it will take to double your initial investment in city1
idx = city_names_list.index(city1)
first = 0
last = float_df.shape[1]
row = float_df.iloc[idx, :]
# convert row to Panadas series
row_series = row.squeeze(axis=0)
rowmax = row_series.max()
rowmin = row_series.min()
string_starting_idx = row.idxmin()
numeric_starting_idx = 0
```

```
numeric_ending_idx = 0

month_dict = {}
months = float_df.columns.values.tolist()
for i in range(0, len(months)):
month_dict[months[i]] = i
numeric_starting_idx = month_dict[string_starting_idx]
double_price = rowmin * 2
current_price = 999999
for j in range(numeric_starting_idx, len(months)):
current_price = row[j]
if current_price >= double_price:
numeric_ending_idx = j
break
#convert numeric_ending_idx to string_ending_idx
string_ending_idx = months[numeric_ending_idx]
min_years = (numeric_ending_idx - numeric_starting_idx) / 12
start_year = string_starting_idx[-4:]
end_year = string_ending_idx[-4:]
int_start_year = int(start_year)
int_end_year = int(end_year)
int_start_index = int_start_year - 1996
int_end_index = int_end_year - 1996
senators = []
state_abbr = city1[-2:]
if state_abbr == 'AL':
senators = AL_senators[int_start_index:int_end_index]

if state_abbr == 'AZ':
senators = AZ_senators[int_start_index:int_end_index]

if state_abbr == 'AR':
senators = AR_senators[int_start_index:int_end_index]

if state_abbr == 'CA':
senators = CA_senators[int_start_index:int_end_index]

if state_abbr == 'CO':
senators = CO_senators[int_start_index:int_end_index]


if state_abbr == 'CT':
senators = CT_senators[int_start_index:int_end_index]

if state_abbr == 'FL':
senators = FL_senators[int_start_index:int_end_index]

if state_abbr == 'GA':
senators = GA_senators[int_start_index:int_end_index]

if state_abbr == 'HI':
senators = HI_senators[int_start_index:int_end_index]

```

```
if state_abbr == 'ID':
senators = ID_senators[int_start_index:int_end_index]

if state_abbr == 'IL':
senators = IL_senators[int_start_index:int_end_index]

if state_abbr == 'IN':
senators = IN_senators[int_start_index:int_end_index]

if state_abbr == 'IA':
senators = IA_senators[int_start_index:int_end_index]

if state_abbr == 'KS':
senators = KS_senators[int_start_index:int_end_index]

if state_abbr == 'KY':
senators = KY_senators[int_start_index:int_end_index]

if state_abbr == 'LA':
senators = LA_senators[int_start_index:int_end_index]

if state_abbr == 'ME':
senators = ME_senators[int_start_index:int_end_index]

if state_abbr == 'MD':
senators = MD_senators[int_start_index:int_end_index]

if state_abbr == 'MA':
senators = MA_senators[int_start_index:int_end_index]

if state_abbr == 'MI':
senators = MI_senators[int_start_index:int_end_index]

if state_abbr == 'MN':
senators = MN_senators[int_start_index:int_end_index]

if state_abbr == 'MS':
senators = MS_senators[int_start_index:int_end_index]

if state_abbr == 'MO':
senators = MO_senators[int_start_index:int_end_index]

if state_abbr == 'NE':
senators = NE_senators[int_start_index:int_end_index]

if state_abbr == 'NV':
senators = NV_senators[int_start_index:int_end_index]

if state_abbr == 'NJ':
senators = NJ_senators[int_start_index:int_end_index]

if state_abbr == 'NM':
senators = NM_senators[int_start_index:int_end_index]
```

```
if state_abbr == 'NY':

senators = NY_senators[int_start_index:int_end_index]

if state_abbr == 'NC':
senators = NC_senators[int_start_index:int_end_index]

if state_abbr == 'OH':
senators = OH_senators[int_start_index:int_end_index]

if state_abbr == 'OK':
senators = OK_senators[int_start_index:int_end_index]

if state_abbr == 'OR':
senators = OR_senators[int_start_index:int_end_index]

if state_abbr == 'PA':
senators = PA_senators[int_start_index:int_end_index]

if state_abbr == 'RI':
senators = RI_senators[int_start_index:int_end_index]

if state_abbr == 'SC':
senators = SC_senators[int_start_index:int_end_index]

if state_abbr == 'TN':
senators = TN_senators[int_start_index:int_end_index]

if state_abbr == 'TX':
senators = TX_senators[int_start_index:int_end_index]

if state_abbr == 'UT':
senators = UT_senators[int_start_index:int_end_index]

if state_abbr == 'VA':
senators = VA_senators[int_start_index:int_end_index]

if state_abbr == 'WA':
senators = WA_senators[int_start_index:int_end_index]

if state_abbr == 'WI':
senators = WI_senators[int_start_index:int_end_index]
if min_years <= 0:
print('The shortest amount of time to double your money in', city1, 'is unknown. Senate representation during the doubling time cannot be determined.')
else:
min_years_1f = "{:.1f}".format(min_years)
print('The minimum price was:', rowmin)
print('The minimum price occured on:', string_starting_idx)
if city1 == 'Washington, DC':
print(city1, 'This city does not vote for U.S. senators.')
```

```
else:
print(city1, 'was represented by', senators, 'in the U.S. Senate during this time.')
print('A price greater than or equal to twice the minimum price first occured on:', string_ending_idx)
print('The shortest amount of time to double your money in', city1, 'was', min_years_1f, 'years.')


if mode == 6:
# compute the derivative of normalized price w.r.t. time for a city, determine its good years and poor years in the dataset,
# and make a prediction about next year based on the derivative in 2021.
sz = city_name_size_rank_dict[city1]
derivative_list = []
current_year_and_deriv_tuple_list = []
good_years_list = []
fair_years_list = []
poor_years_list = []
current_year = 1996
print('TABLE OF THE FIRST DERIVATIVE OF THE NORMALIZED PRICE W.R.T TIME IN', city_names_list[sz])
print('----------------------------------')
for i in range(0,300,12):
year = float_df.iloc[:, i:i+12]
row = year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
derivative_list.append(dp_dt)
current_year_and_deriv_tuple = (current_year, dp_dt)
current_year_and_deriv_tuple_list.append(current_year_and_deriv_tuple)
print('|', current_year, '|', dp_dt,'|')
current_year = current_year + 1

partial_year = float_df.iloc[:, 300:308]
row = partial_year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
derivative_list.append(dp_dt)
current_year_and_deriv_tuple = (current_year, dp_dt)
current_year_and_deriv_tuple_list.append(current_year_and_deriv_tuple)
print('|', current_year, '|', dp_dt, '|')
```

```
current_year = current_year + 1
print('---------------------------------')
derivative_series = pd.Series(derivative_list)
series_median = derivative_series.median()
print('\nThe median derivative in', city_names_list[sz], 'was:', series_median)
current_year = current_year_and_deriv_tuple_list
for j in range(0, len(derivative_list)):
current_tuple = current_year_and_deriv_tuple_list[j]
current_year = current_tuple[0]
# a year is considered a "good year" if the derivative of price w.r.t time is greater than or equal to 20% above the median derivative
if current_tuple[1] >= series_median * 1.2:
good_years_list.append(current_year)
# a year is considered a "fair year" if the derivative of price w.r.t time is between 20% below the median derivative and 20% above the median derivative
if (current_tuple[1] > series_median * 0.8) and (current_tuple[1] < series_median * 1.2):
fair_years_list.append(current_year)
# a year is considered a "poor year" if the derivative of price w.r.t time is less than or equal to 20% below the median derivative
if current_tuple[1] <= series_median * 0.8:
poor_years_list.append(current_year)

current_tuple = current_year_and_deriv_tuple_list[len(current_year_and_deriv_tuple_list) - 1]
current_year = current_tuple[0]

print(tm.colored('The good years to invest in', 'green'), tm.colored(city_names_list[sz], 'green'), tm.colored('were:', 'green'), tm.colored(good_years_list, 'green'))
print(tm.colored('The fair years to invest in', 'yellow'), tm.colored(city_names_list[sz], 'yellow'), tm.colored('were:', 'yellow'), tm.colored(fair_years_list, 'yellow'))
print(tm.colored('The poor years to invest in', 'red'), tm.colored(city_names_list[sz], 'red'), tm.colored('were:', 'red'), tm.colored(poor_years_list, 'red'))

# code to plot the derivative graph
x_axis_values = np.arange(1996, 2022, 1)
plt.rcParams["figure.figsize"] = [7.00, 3.50]
plt.rcParams["figure.autolayout"] = True
x = x_axis_values
y = derivative_list
default_x_ticks = range(len(x))
plt.plot(default_x_ticks, y)
title_str = "Graph of the First Derivative for " + city_names_list[sz]
plt.title(title_str)
plt.xticks(default_x_ticks, x)
plt.show()


```

```
if mode == 7:

# compute the second derivative of normalized price w.r.t. time for a city.
sz = city_name_size_rank_dict[city1]
first_derivative_list = []
second_derivative_list = []
current_year_and_first_deriv_tuple_list = []
current_year_and_second_deriv_tuple_list = []
current_year = 1996
print('TABLE OF THE SECOND DERIVATIVE OF THE NORMALIZED PRICE W.R.T TIME IN', city_names_list[sz])
print('------------------------------------')
for i in range(0,308,12):
year = float_df.iloc[:, i:i+12]
if current_year == 2021:
year = float_df.iloc[:, i:i+8]
row = year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
first_derivative_list.append(dp_dt)
current_year_and_first_deriv_tuple = (current_year, dp_dt)
current_year_and_first_deriv_tuple_list.append(current_year_and_first_deriv_tuple)
current_year = current_year + 1
current_year = 1996
for j in range(0, 25):
change_in_derivative_of_price = first_derivative_list[j + 1] - first_derivative_list[j]
change_in_time_years = 1 # year
d2p_dt2 = change_in_derivative_of_price / change_in_time_years
second_derivative_list.append(d2p_dt2)
current_year_and_second_deriv_tuple = (current_year, d2p_dt2)
current_year_and_second_deriv_tuple_list.append(current_year_and_second_deriv_tuple)
print('|', current_year, '|', d2p_dt2,'|')
current_year = current_year + 1
# code to plot the derivative graph
x_axis_values = np.arange(1996, 2021, 1)
plt.rcParams["figure.figsize"] = [7.00, 3.50]
plt.rcParams["figure.autolayout"] = True
x = x_axis_values
y = second_derivative_list

default_x_ticks = range(len(x))
plt.plot(default_x_ticks, y)
title_str = "Graph of the Second Derivative for " + city_names_list[sz]
plt.title(title_str)
plt.xticks(default_x_ticks, x)
plt.show()

partial_year = float_df.iloc[:, 300:308]
row = partial_year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
first_derivative_list.append(dp_dt)
current_year_and_first_deriv_tuple = (current_year, dp_dt)
current_year_and_first_deriv_tuple_list.append(current_year_and_first_deriv_tuple)
print('|', current_year, '|', dp_dt, '|')
current_year = current_year + 1
print('---------------------------------')
first_derivative_series = pd.Series(first_derivative_list)
series_median = first_derivative_series.median()
print('\nThe median derivative in', city_names_list[sz], 'was:', series_median)
current_year = current_year_and_first_deriv_tuple_list


if mode == 8:
# compute the second derivative of normalized price w.r.t. time for a city for one year
sz = city_name_size_rank_dict[city1]
first_derivative_list = []
second_derivative_list = []
current_year_and_first_deriv_tuple_list = []
current_year_and_second_deriv_tuple_list = []
current_year = 1996
#print('TABLE OF THE SECOND DERIVATIVE OF THE NORMALIZED PRICE W.R.T TIME IN', city_names_list[sz])
#print('---------------------------------')
for i in range(0,308,12):
year = float_df.iloc[:, i:i+12]
if current_year == 2021:
year = float_df.iloc[:, i:i+8]
row = year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
first_derivative_list.append(dp_dt)
current_year_and_first_deriv_tuple = (current_year, dp_dt)
current_year_and_first_deriv_tuple_list.append(current_year_and_first_deriv_tuple)
current_year = current_year + 1
current_year = 1996
for j in range(0, 25):
change_in_derivative_of_price = first_derivative_list[j + 1] - first_derivative_list[j]
change_in_time_years = 1 # year
d2p_dt2 = change_in_derivative_of_price / change_in_time_years
second_derivative_list.append(d2p_dt2)
current_year_and_second_deriv_tuple = (current_year, d2p_dt2)
current_year_and_second_deriv_tuple_list.append(current_year_and_second_deriv_tuple)
if requested_year == current_year:
print('The second derivative of the normalized price w.r.t time in', city1, 'in', requested_year, 'was:', d2p_dt2)
#print('|', current_year, '|', d2p_dt2,'|')
current_year = current_year + 1

#plt.plot(second_derivative_list)
#plt.show()
partial_year = float_df.iloc[:, 300:308]
row = partial_year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
first_derivative_list.append(dp_dt)
current_year_and_first_deriv_tuple = (current_year, dp_dt)
current_year_and_first_deriv_tuple_list.append(current_year_and_first_deriv_tuple)
#print('|', current_year, '|', dp_dt, '|')
current_year = current_year + 1
#print('---------------------------------')
first_derivative_series = pd.Series(first_derivative_list)
series_median = first_derivative_series.median()
#print('\nThe median derivative in', city_names_list[sz], 'was:', series_median)
current_year = current_year_and_first_deriv_tuple_list

if mode == 9:
# compute the second derivative of normalized price w.r.t. time for a city for several years.
sz = city_name_size_rank_dict[city1]
first_derivative_list = []
second_derivative_list = []
current_year_and_first_deriv_tuple_list = []
current_year_and_second_deriv_tuple_list = []
current_year = 1996
#print('TABLE OF THE SECOND DERIVATIVE OF THE NORMALIZED PRICE W.R.T TIME IN', city_names_list[sz])
#print('---------------------------------')
for i in range(0,308,12):
```

```
year = float_df.iloc[:, i:i+12]
if current_year == 2021:
year = float_df.iloc[:, i:i+8]
row = year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
first_derivative_list.append(dp_dt)
current_year_and_first_deriv_tuple = (current_year, dp_dt)
current_year_and_first_deriv_tuple_list.append(
current_year_and_first_deriv_tuple)
current_year = current_year + 1
current_year = 1996
for j in range(0, 25):
change_in_derivative_of_price = first_derivative_list[j + 1] -
first_derivative_list[j]
change_in_time_years = 1 # year
d2p_dt2 = change_in_derivative_of_price / change_in_time_years
second_derivative_list.append(d2p_dt2)
current_year_and_second_deriv_tuple = (current_year, d2p_dt2)

current_year_and_second_deriv_tuple_list.append(
current_year_and_second_deriv_tuple)
if current_year in requested_years:
print('The second derivative of the normalized price w.r.t time in',
city1, 'in', current_year, 'was:', d2p_dt2)
#print('|', current_year, '|', d2p_dt2,'|')
current_year = current_year + 1

plt.plot(second_derivative_list)
plt.show()
partial_year = float_df.iloc[:, 300:308]
row = partial_year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
first_derivative_list.append(dp_dt)
current_year_and_first_deriv_tuple = (current_year, dp_dt)
current_year_and_first_deriv_tuple_list.append(
current_year_and_first_deriv_tuple)
#print('|', current_year, '|', dp_dt, '|')
current_year = current_year + 1
#print('--------------------------------')
first_derivative_series = pd.Series(first_derivative_list)
series_median = first_derivative_series.median()
```

```
#print('\nThe median derivative in', city_names_list[sz], 'was:', series_median)
current_year = current_year_and_first_deriv_tuple_list

if mode == 10:
# compute the second derivative of normalized price w.r.t. time for a city and display its +/- sign
sz = city_name_size_rank_dict[city1]
first_derivative_list = []
second_derivative_list = []
positive_second_derivative_list = []
negative_second_derivative_list = []
current_year_and_first_deriv_tuple_list = []
current_year_and_second_deriv_tuple_list = []
current_year = 1996
#print('TABLE OF THE SECOND DERIVATIVE OF THE NORMALIZED PRICE W.R.T TIME IN', city_names_list[sz])
#print('---------------------------------')
for i in range(0,308,12):
year = float_df.iloc[:, i:i+12]
if current_year == 2021:
year = float_df.iloc[:, i:i+8]
row = year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
first_derivative_list.append(dp_dt)
current_year_and_first_deriv_tuple = (current_year, dp_dt)
current_year_and_first_deriv_tuple_list.append(current_year_and_first_deriv_tuple)
current_year = current_year + 1
current_year = 1996
for j in range(0, 25):
change_in_derivative_of_price = first_derivative_list[j + 1] - first_derivative_list[j]
change_in_time_years = 1 # year

d2p_dt2 = change_in_derivative_of_price / change_in_time_years
if d2p_dt2 > 0:
positive_second_derivative_list.append(current_year)
else:
negative_second_derivative_list.append(current_year)
second_derivative_list.append(d2p_dt2)
current_year_and_second_deriv_tuple = (current_year, d2p_dt2)
current_year_and_second_deriv_tuple_list.append(current_year_and_second_deriv_tuple)
current_year = current_year + 1

#plt.plot(second_derivative_list)
#plt.show()
```

```
partial_year = float_df.iloc[:, 300:308]
row = partial_year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
first_derivative_list.append(dp_dt)
current_year_and_first_deriv_tuple = (current_year, dp_dt)
current_year_and_first_deriv_tuple_list.append(current_year_and_first_deriv_tuple)
#print('|', current_year, '|', dp_dt, '|')
current_year = current_year + 1
#print('---------------------------------')
first_derivative_series = pd.Series(first_derivative_list)
series_median = first_derivative_series.median()
#print('\nThe median derivative in', city_names_list[sz], 'was:', series_median)
current_year = current_year_and_first_deriv_tuple_list
print("The second derivative was positive in these years:", positive_second_derivative_list)
print("The second derivative was negative in these years:", negative_second_derivative_list)

if mode == 11:
# compute the second derivative of normalized price w.r.t. time for a city.
# print whether it's increasing-increasing, decreasing-decreasing, etc
sz = city_name_size_rank_dict[city1]
first_derivative_list = []
second_derivative_list = []
inc_inc_years = []
inc_dec_years = []
dec_inc_years = []
dec_dec_years = []
negative_second_derivative_list = []
current_year_and_first_deriv_tuple_list = []
current_year_and_second_deriv_tuple_list = []
current_year = 1996
#print('TABLE OF THE SECOND DERIVATIVE OF THE NORMALIZED PRICE W.R.T TIME IN', city_names_list[sz])
#print('---------------------------------')
for i in range(0,308,12):
year = float_df.iloc[:, i:i+12]
if current_year == 2021:
year = float_df.iloc[:, i:i+8]
row = year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
```

```
dp_dt = change_in_price / change_in_time_days

first_derivative_list.append(dp_dt)
current_year_and_first_deriv_tuple = (current_year, dp_dt)
current_year_and_first_deriv_tuple_list.append(
current_year_and_first_deriv_tuple)
current_year = current_year + 1
current_year = 1996
for j in range(0, 25):
change_in_derivative_of_price = first_derivative_list[j + 1] -
first_derivative_list[j]
change_in_time_years = 1 # year
d2p_dt2 = change_in_derivative_of_price / change_in_time_years
if ((first_derivative_list[j] > 0) and (d2p_dt2 > 0)):
inc_inc_years.append(current_year)
elif ((first_derivative_list[j] > 0) and (d2p_dt2 < 0)):
inc_dec_years.append(current_year)
elif ((first_derivative_list[j] < 0) and (d2p_dt2 > 0)):
dec_inc_years.append(current_year)
elif ((first_derivative_list[j] < 0) and (d2p_dt2 < 0)):
dec_dec_years.append(current_year)
current_year_and_second_deriv_tuple = (current_year, d2p_dt2)
current_year_and_second_deriv_tuple_list.append(
current_year_and_second_deriv_tuple)
current_year = current_year + 1

#plt.plot(second_derivative_list)
#plt.show()
partial_year = float_df.iloc[:, 300:308]
row = partial_year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
first_derivative_list.append(dp_dt)
current_year_and_first_deriv_tuple = (current_year, dp_dt)
current_year_and_first_deriv_tuple_list.append(
current_year_and_first_deriv_tuple)
#print('|', current_year, '|', dp_dt, '|')
current_year = current_year + 1
#print('---------------------------------')
first_derivative_series = pd.Series(first_derivative_list)
series_median = first_derivative_series.median()
#print('\nThe median derivative in', city_names_list[sz], 'was:', series_median)
current_year = current_year_and_first_deriv_tuple_list
print("The median price of a single-family home was increasing at an increasing
rate in: ", inc_inc_years)
print("The median price of a single-family home was increasing at an decreasing
rate in: ", inc_dec_years)
print("The median price of a single-family home was decreasing at an increasing
rate in: ", dec_inc_years)
print("The median price of a single-family home was decreasing at an decreasing rate in: ", dec_dec_years)

if mode == 12:
complete_rows_df = original_df.dropna()
state_names_df = complete_rows_df.drop(columns=['RegionID', 'RegionType', 'SizeRank'])
west_coast_region_df = state_names_df.loc[state_names_df['StateName'].isin(west_coast_states)]
rocky_mountain_region_df = state_names_df.loc[state_names_df['StateName'].isin(rocky_mountain_states)]
gulf_coast_region_df = state_names_df.loc[state_names_df['StateName'].isin(gulf_coast_states)]
midwest_region_df = state_names_df.loc[state_names_df['StateName'].isin(midwest_states)]
east_coast_region_df = state_names_df.loc[state_names_df['StateName'].isin(east_coast_states)]

current_year = 1996

west_coast_float_df = west_coast_region_df
rocky_mountain_float_df = rocky_mountain_region_df.drop(columns=['StateName'])
gulf_coast_float_df = gulf_coast_region_df.drop(columns=['StateName'])
midwest_float_df = midwest_region_df.drop(columns=['StateName'])
east_coast_float_df = east_coast_region_df.drop(columns=['StateName'])

CA_highs = []
CA_lows = []
# for loop for west coast
for i in range(0,300,12):
year_df = west_coast_float_df.iloc[:, i:i+12]
for index, row in year_df.iterrows():
prices = row[2:]
maximum_price = prices.max()
normalized_prices = prices.divide(maximum_price)
highest_price_for_this_city = normalized_prices.max()
lowest_price_for_this_city = normalized_prices.min()
city_name = row[0]
city_highest_price_tuple = (city_name, highest_price_for_this_city)
city_lowest_price_tuple = (city_name, lowest_price_for_this_city)
# append the tuple to a list
if city_name[-2:] == 'CA':
CA_highs.append(city_highest_price_tuple)
CA_lows.append(city_lowest_price_tuple)

print('CA highs', CA_highs)
print('CA_lows', CA_lows)

if mode == 13:
threshold = (upper_bound + lower_bound) / 2
# total number of cities seen so far
```

```
total_count = 1
# total number of cities mis-predicted so far
error_count = 0
# list to store sum of squared errors
see_list = []

for city1, sz in city_name_size_rank_dict.items():
# compute the derivative of normalized price w.r.t. time for a city, determine its good years and poor years in the dataset,
# and make a prediction about next year based on the derivative in 2020.
derivative_list = []
current_year_and_deriv_tuple_list = []
good_years_list = []
fair_years_list = []
poor_years_list = []
current_year = 1996
#print('TABLE OF THE DERIVATIVE OF THE NORMALIZED PRICE W.R.T TIME IN', city_names_list[sz])
#print('---------------------------------')
for i in range(0,288,12):
year = float_df.iloc[:, i:i+12]
row = year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
derivative_list.append(dp_dt)

current_year_and_deriv_tuple = (current_year, dp_dt)
current_year_and_deriv_tuple_list.append(current_year_and_deriv_tuple)
#print('|', current_year, '|', dp_dt,'|')
current_year = current_year + 1

#print('---------------------------------')
derivative_series = pd.Series(derivative_list)
series_median = derivative_series.median()
#print('\nThe median derivative in', city_names_list[sz], 'was:', series_median)
current_year = current_year_and_deriv_tuple_list
for j in range(0, len(derivative_list)):
current_tuple = current_year_and_deriv_tuple_list[j]
current_year = current_tuple[0]
# a year is considered a "good year" if the derivative of price w.r.t time is greater than or equal to 10% above the median derivative
if current_tuple[1] >= series_median * 1.1:
good_years_list.append(current_year)
# a year is considered a "fair year" if the derivative of price w.r.t time is between 10% below the median derivative and 10% above the median derivative
```

```
if (current_tuple[1] > series_median * 0.9) and (current_tuple[1] < series_median * 1.1):
fair_years_list.append(current_year)
# a year is considered a "poor year" if the derivative of price w.r.t time is less than or equal to 10% below the median derivative
if current_tuple[1] <= series_median * 0.9:
poor_years_list.append(current_year)

current_tuple = current_year_and_deriv_tuple_list[len(current_year_and_deriv_tuple_list) - 1]
current_year = current_tuple[0]

# compute the derivative of price w.r.t. for the test year 2020
test_year = float_df.iloc[:, 288:300]
row = test_year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
derivative_list.append(dp_dt)
test_tuple = (2020, dp_dt)

# predicted condition
# 0 == poor year to invest / slower than expected growth in price
# 1 == fair year to invest / expected growth in price
# 2 == good year to invest / faster than expected growth in price
predicted_condition = 0

# actual condition
# 0 == poor year to invest / slower than expected growth in price
# 1 == fair year to invest / expected growth in price
# 2 == good year to invest / faster than expected growth in price
actual_condition = 0

# a year is considered a "good year" if the derivative of price w.r.t time is greater than or equal to 20% above the median derivative
if current_tuple[1] >= series_median * upper_bound:
print(tm.colored('2020 is predicted to be a good year to invest in', 'green'), tm.colored(city_names_list[sz], 'green'), tm.colored(' because the predicted derivative is: ', 'green'), current_tuple[1])
predicted_condition = 2
# a year is considered a "fair year" if the derivative of price w.r.t time is between 20% below the median derivative and 20% above the median derivative
elif (current_tuple[1] > series_median * lower_bound) and (current_tuple[1] < series_median * upper_bound):
print(tm.colored('2020 is predicted to be a fair year to invest in', 'yellow'), tm.colored(city_names_list[sz], 'yellow'), tm.colored(' because the predicted derivative is: ', 'yellow'), current_tuple[1])
```

```
predicted_condition = 1
# a year is considered a "poor year" if the derivative of price w.r.t time is less than or equal to 20% below the median derivative
else:
print(tm.colored('2020 is predicted to be a poor year to invest in', 'red'), tm.colored(city_names_list[sz], 'red'), tm.colored(' because the predicted derivative is: ', 'red'), current_tuple[1])
predicted_condition = 0
# if the actual derivative of price w.r.t. time for the year was >= the predicted derivative of price w.r.t. for the year,
# then it was a good year for that city because real estate prices grew faster than expected
if test_tuple[1] >= series_median * upper_bound:
print(tm.colored('2020 was a good year to invest in', 'green'), tm.colored(city_names_list[sz], 'green'), tm.colored(' because the derivative was: ', 'green'), test_tuple[1])
actual_condition = 2
elif (test_tuple[1] > series_median * lower_bound) and (test_tuple[1] < series_median * upper_bound):
print(tm.colored('2020 was a fair year to invest in', 'yellow'), tm.colored(city_names_list[sz], 'yellow'), tm.colored(' because the derivative was: ', 'yellow'), test_tuple[1])
actual_condition = 1
# if the actual derivative of price w.r.t. time for the year was < the predicted derivative of price w.r.t. for the year,
# then it was a bad year for that city because real estate prices grew slower than expected or decreased
else:
print(tm.colored('2020 was a poor year to invest in', 'red'), tm.colored(city_names_list[sz], 'red'), tm.colored(' because the derivative was: ', 'red'), test_tuple[1])
actual_condition = 0
sse = np.sqrt((current_tuple[1] - test_tuple[1]) ** 2)
see_list.append(sse)
total_count = total_count + 1
if predicted_condition != actual_condition:
error_count = error_count + 1
# error ratio
error_ratio = error_count / total_count
print("The total number of cities seen is: ", total_count)
print("The total number of cities mis-predicted is: ", error_count)
print("The error ratio for threshold ==", threshold, "is: ", error_ratio)

if mode == 15:
# compute the derivative of normalized price w.r.t. time for a city, determine its good years and poor years in the dataset,
# and make a prediction about next year based on the derivative in 2021.
sz = city_name_size_rank_dict[city1]
derivative_list = []
current_year_and_deriv_tuple_list = []
good_years_list = []
fair_years_list = []
poor_years_list = []
```

```
current_year = 1996
print('TABLE OF THE FIRST DERIVATIVE OF THE NORMALIZED PRICE W.R.T TIME IN', city_names_list[sz])
print('--------------------------------')
for i in range(0,300,12):
year = float_df.iloc[:, i:i+12]
row = year.iloc[sz, :]

rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
derivative_list.append(dp_dt)
current_year_and_deriv_tuple = (current_year, dp_dt)
current_year_and_deriv_tuple_list.append(current_year_and_deriv_tuple)
print('|', current_year, '|', dp_dt,'|')
current_year = current_year + 1

partial_year = float_df.iloc[:, 300:308]
row = partial_year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
derivative_list.append(dp_dt)
current_year_and_deriv_tuple = (current_year, dp_dt)
current_year_and_deriv_tuple_list.append(current_year_and_deriv_tuple)
print('|', current_year, '|', dp_dt, '|')
current_year = current_year + 1
print('--------------------------------')
derivative_series = pd.Series(derivative_list)
series_median = derivative_series.median()
print('\nThe median derivative in', city_names_list[sz], 'was:', series_median)
current_year = current_year_and_deriv_tuple_list
for j in range(0, len(derivative_list)):
current_tuple = current_year_and_deriv_tuple_list[j]
current_year = current_tuple[0]
# a year is considered a "good year" if the derivative of price w.r.t time is greater than or equal to 20% above the median derivative
if current_tuple[1] >= series_median * 1.2:
good_years_list.append(current_year)
# a year is considered a "fair year" if the derivative of price w.r.t time is between 20% below the median derivative and 20% above the median derivative
if (current_tuple[1] > series_median * 0.8) and (current_tuple[1] < series_median * 1.2):
fair_years_list.append(current_year)
```

```
# a year is considered a "poor year" if the derivative of price w.r.t time is less than or equal to 20% below the median derivative
if current_tuple[1] <= series_median * 0.8:
poor_years_list.append(current_year)

current_tuple = current_year_and_deriv_tuple_list[len(current_year_and_deriv_tuple_list) - 1]
current_year = current_tuple[0]

#print(tm.colored('The good years to invest in', 'green'), tm.colored(city_names_list[sz], 'green'), tm.colored('were:', 'green'), tm.colored(good_years_list, 'green'))
#print(tm.colored('The fair years to invest in', 'yellow'), tm.colored(city_names_list[sz], 'yellow'), tm.colored('were:', 'yellow'), tm.colored(fair_years_list, 'yellow'))
#print(tm.colored('The poor years to invest in', 'red'), tm.colored(city_names_list[sz], 'red'), tm.colored('were:', 'red'), tm.colored(poor_years_list, 'red'))

# code to plot the derivative graph
#x_axis_values = np.arange(1996, 2022, 1)
#plt.rcParams["figure.figsize"] = [7.00, 3.50]

#plt.rcParams["figure.autolayout"] = True
#x = x_axis_values
#y = derivative_list
#default_x_ticks = range(len(x))
#plt.plot(default_x_ticks, y)
#title_str = "Graph of the First Derivative for " + city_names_list[sz]
#plt.title(title_str)
#plt.xticks(default_x_ticks, x)
#plt.show()

# compute the second derivative of normalized price w.r.t. time for a city.
# print whether it's increasing-increasing, decreasing-decreasing, etc
sz = city_name_size_rank_dict[city1]
first_derivative_list = []
second_derivative_list = []
inc_inc_years = []
inc_dec_years = []
dec_inc_years = []
dec_dec_years = []
negative_second_derivative_list = []
current_year_and_first_deriv_tuple_list = []
current_year_and_second_deriv_tuple_list = []
current_year = 1996
#print('TABLE OF THE SECOND DERIVATIVE OF THE NORMALIZED PRICE W.R.T TIME IN', city_names_list[sz])
#print('---------------------------------')
for i in range(0,308,12):
year = float_df.iloc[:, i:i+12]
if current_year == 2021:
year = float_df.iloc[:, i:i+8]
```

```
row = year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
first_derivative_list.append(dp_dt)
current_year_and_first_deriv_tuple = (current_year, dp_dt)
current_year_and_first_deriv_tuple_list.append(current_year_and_first_deriv_tuple)
current_year = current_year + 1
current_year = 1996
for j in range(0, 25):
change_in_derivative_of_price = first_derivative_list[j + 1] - first_derivative_list[j]
change_in_time_years = 1 # year
d2p_dt2 = change_in_derivative_of_price / change_in_time_years
if ((first_derivative_list[j] > 0) and (d2p_dt2 > 0)):
inc_inc_years.append(current_year)
elif ((first_derivative_list[j] > 0) and (d2p_dt2 < 0)):
inc_dec_years.append(current_year)
elif ((first_derivative_list[j] < 0) and (d2p_dt2 > 0)):
dec_inc_years.append(current_year)
elif ((first_derivative_list[j] < 0) and (d2p_dt2 < 0)):
dec_dec_years.append(current_year)
current_year_and_second_deriv_tuple = (current_year, d2p_dt2)
current_year_and_second_deriv_tuple_list.append(current_year_and_second_deriv_tuple)
current_year = current_year + 1

#plt.plot(second_derivative_list)
#plt.show()
partial_year = float_df.iloc[:, 300:308]
row = partial_year.iloc[sz, :]
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
first_derivative_list.append(dp_dt)
current_year_and_first_deriv_tuple = (current_year, dp_dt)
current_year_and_first_deriv_tuple_list.append(current_year_and_first_deriv_tuple)
#print('|', current_year, '|', dp_dt, '|')
current_year = current_year + 1
#print('--------------------------------')
first_derivative_series = pd.Series(first_derivative_list)
series_median = first_derivative_series.median()
```

```
#print('\nThe median derivative in', city_names_list[sz], 'was:', series_median)
current_year = current_year_and_first_deriv_tuple_list
#print("The median price of a single-family home was increasing at an increasing rate in: ", inc_inc_years)
#print("The median price of a single-family home was increasing at an decreasing rate in: ", inc_dec_years)
#print("The median price of a single-family home was decreasing at an increasing rate in: ", dec_inc_years)
#print("The median price of a single-family home was decreasing at an decreasing rate in: ", dec_dec_years)
for y in range(1999, 2021):
if (y in good_years_list and inc_inc_years):
print(y, 'was a bubble year for', city_names_list[sz])

if mode == 16:

# compute the second derivative of normalized price w.r.t. time for a city.
sz = city_name_size_rank_dict[city1]
first_derivative_list = []
second_derivative_list = []
current_year_and_first_deriv_tuple_list = []
current_year_and_second_deriv_tuple_list = []
current_year = 1996
#print('TABLE OF THE SECOND DERIVATIVE OF THE NORMALIZED PRICE W.R.T TIME IN', city_names_list[sz])
#print('---------------------------------')
for i in range(0,300,12):
year = float_df.iloc[:, i:i+12]
row = year.iloc[sz, :]
print(row)
rowmax = row.max()
norm_series = row.divide(rowmax)
initial_price = norm_series[0]
final_price = norm_series[len(norm_series) - 1]
change_in_price = final_price - initial_price
change_in_time_days = 12
dp_dt = change_in_price / change_in_time_days
first_derivative_list.append(dp_dt)
current_year_and_first_deriv_tuple = (current_year, dp_dt)
current_year_and_first_deriv_tuple_list.append(current_year_and_first_deriv_tuple)
current_year = current_year + 1

# write a color-coded list of cities to a file based on their Senators' political party and the doubling time
elif mode == 17:
default_stdout = sys.stdout
output_file = open('party_list.txt', 'w')
# redirect stdout to a file
sys.stdout = output_file

red_cities_count = 0
```

```
red_cities_total_years = 0
red_cities_average_doubling_time = 0

blue_cities_count = 0
blue_cities_total_years = 0
blue_cities_average_doubling_time = 0

purple_cities_count = 0
purple_cities_total_years = 0
purple_cities_average_doubling_time = 0

for k in range(1, 122):
city1 = region_name_column_list[k]
state_abbr = city1[-2:]
# special cases
if city1 == "Denver, CO":
latitude = "39.7392"
longitude = "-104.9850"
elif city1 == "Richmond, VA":
latitude = "37.53"
longitude = "-77.47"
elif city1 == "Urban Honolulu, HI":
latitude = "21.315603"
longitude = "-157.858093"
elif city1 == "Ventura, CA":
latitude = "34.275"
longitude = "-119.228"
elif city1 == "North Port-Sarasota-Bradenton, FL":
cityX = "Sarasota, FL"
# find cityX's location
nominatim = Nominatim()
city_json = nominatim.query(cityX).toJSON()[0]
latitude = city_json["lat"]
longitude = city_json["lon"]
elif city1 == "Minneapolis-St Paul, MN":
cityY = "Minneapolis, MN"
# find cityY's location
nominatim = Nominatim()
city_json = nominatim.query(cityY).toJSON()[0]
latitude = city_json["lat"]
longitude = city_json["lon"]
else:
# find city1's location
nominatim = Nominatim()
city_json = nominatim.query(city1).toJSON()[0]
latitude = city_json["lat"]
longitude = city_json["lon"]

# get city1's size rank
sz = city_name_size_rank_dict[city1]
# Find how long it will take to double your initial investment in city1
idx = city_names_list.index(city1)
first = 0
```

```
last = float_df.shape[1]
row = float_df.iloc[idx, :]
row_series = row.squeeze(axis=0)
rowmax = row_series.max()
rowmin = row_series.min()
string_starting_idx = row.idxmin()
numeric_starting_idx = 0
numeric_ending_idx = 0
month_dict = {}
months = float_df.columns.values.tolist()
for i in range(0, len(months)):
month_dict[months[i]] = i
numeric_starting_idx = month_dict[string_starting_idx]
double_price = rowmin * 2

current_price = 999999
for j in range(numeric_starting_idx, len(months)):
current_price = row[j]
if current_price >= double_price:
numeric_ending_idx = j
break
#convert numeric_ending_idx to string_ending_idx
string_ending_idx = months[numeric_ending_idx]
min_years = (numeric_ending_idx - numeric_starting_idx) / 12
start_year = string_starting_idx[-4:]
end_year = string_ending_idx[-4:]
int_start_year = int(start_year)
int_end_year = int(end_year)
int_start_index = int_start_year - 1996
int_end_index = int_end_year - 1996
# if the median home price in the given city did not double in during the 24 years in the dataset,
# then use the senate representation for the entire 24-year period as the senate representation for
# this city
if int_end_index < int_start_index:
int_start_index = 0
int_end_index = int(len(row) / 12)
if numeric_ending_idx == 0:
int_start_index = 0
int_end_index = int(len(row) / 12)
senators = []
if state_abbr == 'AL':
senators = AL_senators[int_start_index:int_end_index]

if state_abbr == 'AZ':
senators = AZ_senators[int_start_index:int_end_index]

if state_abbr == 'AR':
senators = AR_senators[int_start_index:int_end_index]

if state_abbr == 'CA':
senators = CA_senators[int_start_index:int_end_index]
```

```

if state_abbr == 'CO':
senators = CO_senators[int_start_index:int_end_index]

if state_abbr == 'CT':
senators = CT_senators[int_start_index:int_end_index]

if state_abbr == 'FL':
senators = FL_senators[int_start_index:int_end_index]

if state_abbr == 'GA':
senators = GA_senators[int_start_index:int_end_index]

if state_abbr == 'HI':
senators = HI_senators[int_start_index:int_end_index]

if state_abbr == 'ID':
senators = ID_senators[int_start_index:int_end_index]

if state_abbr == 'IL':
senators = IL_senators[int_start_index:int_end_index]

if state_abbr == 'IN':
senators = IN_senators[int_start_index:int_end_index]

if state_abbr == 'IA':
senators = IA_senators[int_start_index:int_end_index]

if state_abbr == 'KS':
senators = KS_senators[int_start_index:int_end_index]

if state_abbr == 'KY':
senators = KY_senators[int_start_index:int_end_index]

if state_abbr == 'LA':
senators = LA_senators[int_start_index:int_end_index]

if state_abbr == 'ME':
senators = ME_senators[int_start_index:int_end_index]

if state_abbr == 'MD':
senators = MD_senators[int_start_index:int_end_index]

if state_abbr == 'MA':
senators = MA_senators[int_start_index:int_end_index]

if state_abbr == 'MI':
senators = MI_senators[int_start_index:int_end_index]

if state_abbr == 'MN':
senators = MN_senators[int_start_index:int_end_index]

```

```
if state_abbr == 'MS':
senators = MS_senators[int_start_index:int_end_index]

if state_abbr == 'MO':
senators = MO_senators[int_start_index:int_end_index]

if state_abbr == 'NE':
senators = NE_senators[int_start_index:int_end_index]

if state_abbr == 'NV':
senators = NV_senators[int_start_index:int_end_index]

if state_abbr == 'NJ':
senators = NJ_senators[int_start_index:int_end_index]

if state_abbr == 'NM':
senators = NM_senators[int_start_index:int_end_index]

if state_abbr == 'NY':
senators = NY_senators[int_start_index:int_end_index]

if state_abbr == 'NC':
senators = NC_senators[int_start_index:int_end_index]

if state_abbr == 'OH':
senators = OH_senators[int_start_index:int_end_index]

if state_abbr == 'OK':
senators = OK_senators[int_start_index:int_end_index]

if state_abbr == 'OR':
senators = OR_senators[int_start_index:int_end_index]

if state_abbr == 'PA':
senators = PA_senators[int_start_index:int_end_index]

if state_abbr == 'RI':
senators = RI_senators[int_start_index:int_end_index]

if state_abbr == 'SC':
senators = SC_senators[int_start_index:int_end_index]

if state_abbr == 'TN':
senators = TN_senators[int_start_index:int_end_index]


if state_abbr == 'TX':
senators = TX_senators[int_start_index:int_end_index]

if state_abbr == 'UT':
senators = UT_senators[int_start_index:int_end_index]

if state_abbr == 'VA':
```

```
senators = VA_senators[int_start_index:int_end_index]

if state_abbr == 'WA':
senators = WA_senators[int_start_index:int_end_index]

if state_abbr == 'WI':
senators = WI_senators[int_start_index:int_end_index]
# define and initialize counter variables
RR_ctr = 0 # number of years the city was represented by 2 Republicans
DD_ctr = 0 # number of years the city was represented by 2 Democrats
DR_ctr = 0 # number of years the city was represented by 1 Democrat and 1 Republican
RD_ctr = 0 # number of years the city was represented by 1 Republican and 1 Democrat
RI_ctr = 0 # number of years the city was represented by 1 Republican and 1 Independent
DI_ctr = 0 # number of years the city was represented by 1 Democrat and 1 Independent
IR_ctr = 0 # number of years the city was represented by 1 Independent and 1 Democrat
ID_ctr = 0 # number of years the city was represented by 1 Independent and 1 Democrat
II_ctr = 0 # number of years the city was represented by 2 Independents
for m in range (0, len(senators)):
if senators[m] == 'RR':
RR_ctr = RR_ctr + 1
elif senators[m] == 'DD':
DD_ctr = DD_ctr + 1
elif senators[m] == 'DR':
DR_ctr = DR_ctr + 1
elif senators[m] == 'RD':
RD_ctr = RD_ctr + 1
elif senators[m] == 'RI':
RI_ctr = RI_ctr + 1
elif senators[m] == 'DI':
DI_ctr = DI_ctr + 1
elif senators[m] == 'IR':
IR_ctr = IR_ctr + 1
elif senators[m] == 'ID':
ID_ctr = ID_ctr + 1
elif senators[m] == 'II':
II_ctr = II_ctr + 1

# compute the total number of years that the city was represented by a Republican during the doubling time
R_total = RR_ctr + RD_ctr + DR_ctr + RI_ctr + IR_ctr
# compute the total number of years that the city was represented by a Democrat during the doubling time
D_total = DD_ctr + RD_ctr + DR_ctr + DI_ctr + ID_ctr
if city1 == "Washington, DC":
colorIcon = "blueIcon"
elif R_total > D_total:
colorIcon = "redIcon"
```

```
elif D_total > R_total:
colorIcon = "blueIcon"
elif R_total == D_total:
colorIcon = "purpleIcon"
elif (R_total == 0) and (D_total == 0):
colorIcon = "blackIcon"


string_ending_idx = months[numeric_ending_idx]
min_years = (numeric_ending_idx - numeric_starting_idx) / 12
if (min_years > 0) and (colorIcon == "blueIcon"):
blue_cities_count = blue_cities_count + 1
blue_cities_total_years = blue_cities_total_years + min_years
elif (min_years > 0) and (colorIcon == "redIcon"):
red_cities_count = red_cities_count + 1
red_cities_total_years = red_cities_total_years + min_years
elif (min_years > 0) and (colorIcon == "purpleIcon"):
purple_cities_count = purple_cities_count + 1
purple_cities_total_years = purple_cities_total_years + min_years
if min_years <= 0:
min_years_1f = "unknown"
else:
min_years_1f = "{:.1f}".format(min_years)
if min_years_1f == "unknown":
print("L.marker([", latitude, ", ", longitude, "], {icon: ", colorIcon, "}).bindPopup(\"The median home price in ", city1, " did not double between 1996 and 2021.\").addTo(map);", sep="")
else:
print("L.marker([", latitude, ", ", longitude, "], {icon: ", colorIcon, "}).bindPopup(\"The shortest amount of time to double your money in ", city1, " was ", min_years_1f, " years.\").addTo(map);", sep="")
blue_cities_average_doubling_time = blue_cities_total_years / blue_cities_count
red_cities_average_doubling_time = red_cities_total_years / red_cities_count
purple_cities_average_doubling_time = purple_cities_total_years / purple_cities_count
print()
blue_cities_average_doubling_time_1f = "{:.1f}".format(blue_cities_average_doubling_time)
red_cities_average_doubling_time_1f = "{:.1f}".format(red_cities_average_doubling_time)
purple_cities_average_doubling_time_1f = "{:.1f}".format(purple_cities_average_doubling_time)
print('The average doubling time in the cities represented by Democratic senators was:', blue_cities_average_doubling_time_1f, ".")
print('The average doubling time in the cities represented by Republican senators was:', red_cities_average_doubling_time_1f, ".")
print('The average doubling time in the cities represented by Democratic and Republican senators was:', purple_cities_average_doubling_time_1f, ".")

sys.stdout = default_stdout
output_file.close()
sys.exit()

```

```
# write a color-coded list of cities grouped by size
elif mode == 18:
default_stdout = sys.stdout
output_file = open('size_list.txt', 'w')
# redirect stdout to a file
sys.stdout = output_file
colorIcon = ""
# size rank is the independent variable
size_rank_list = []
# doubling time is the dependent variable
doubling_time_list = []
for k in range(1, 122):
city1 = region_name_column_list[k]
# special cases
if city1 == "Denver, CO":
latitude = "39.7392"
longitude = "-104.9850"
elif city1 == "Richmond, VA":
latitude = "37.53"
longitude = "-77.47"
elif city1 == "Urban Honolulu, HI":

latitude = "21.315603"
longitude = "-157.858093"
elif city1 == "Ventura, CA":
latitude = "34.275"
longitude = "-119.228"
elif city1 == "North Port-Sarasota-Bradenton, FL":
cityX = "Sarasota, FL"
# find cityX's location
nominatim = Nominatim()
city_json = nominatim.query(cityX).toJSON()[0]
latitude = city_json["lat"]
longitude = city_json["lon"]
elif city1 == "Minneapolis-St Paul, MN":
cityY = "Minneapolis, MN"
# find cityY's location
nominatim = Nominatim()
city_json = nominatim.query(cityY).toJSON()[0]
latitude = city_json["lat"]
longitude = city_json["lon"]
else:
# find city1's location
nominatim = Nominatim()
city_json = nominatim.query(city1).toJSON()[0]
latitude = city_json["lat"]
longitude = city_json["lon"]
# get city1's size rank
sz = city_name_size_rank_dict[city1]
# Find how long it will take to double your initial investment in city1
idx = city_names_list.index(city1)
first = 0
last = float_df.shape[1]
```

```
row = float_df.iloc[idx, :]
# convert row to Panadas series
row_series = row.squeeze(axis=0)
rowmax = row_series.max()
rowmin = row_series.min()
string_starting_idx = row.idxmin()
numeric_starting_idx = 0
numeric_ending_idx = 0

month_dict = {}
months = float_df.columns.values.tolist()
for i in range(0, len(months)):
month_dict[months[i]] = i
numeric_starting_idx = month_dict[string_starting_idx]
double_price = rowmin * 2
current_price = 999999
for j in range(numeric_starting_idx, len(months)):
current_price = row[j]
if current_price >= double_price:
numeric_ending_idx = j
break
# convert numeric_ending_idx to string_ending_idx
string_ending_idx = months[numeric_ending_idx]
min_years = (numeric_ending_idx - numeric_starting_idx) / 12
if min_years <= 0:
min_years_1f = "unknown"
else:
size_rank_list.append(sz)
doubling_time_list.append(min_years)
min_years_1f = "{:.1f}".format(min_years)
# if the city is in the 40 largest U.S. cities by population, color it red
if sz <= 41:
colorIcon = 'blackIcon'
# if the city is in the second 40 largest U.S. cities by population, color it blue
elif (sz >= 42) and (sz <= 81):

colorIcon = 'blueIcon'
# if the city is in the third 40 largest U.S. cities by population, color it green
elif (sz >= 82) and (sz <= 122):
colorIcon = 'skyBlueIcon'
if min_years_1f == "unknown":
print("L.marker([", latitude, ", ", longitude, "], {icon: ", colorIcon, "}).bindPopup(\"The median home price in ", city1, " did not double between 1996 and 2021.\").addTo(map);", sep="")
else:
print("L.marker([", latitude, ", ", longitude, "], {icon: ", colorIcon, "}).bindPopup(\"The shortest amount of time to double your money in ", city1, " was ", min_years_1f, " years.\").addTo(map);", sep="")
size_rank_array = np.array(size_rank_list)
doubling_time_array = np.array(doubling_time_list)
r = scipy.stats.pearsonr(size_rank_array, doubling_time_array)[0]
```

```python
print('pearson_correlation_coefficient', r)
sys.stdout = default_stdout
output_file.close()
sys.exit()

# write a color-coded list of cities grouped by doubling time
elif mode == 19:
default_stdout = sys.stdout
output_file = open('fastest_doublers.txt', 'w')
# redirect stdout to a file
sys.stdout = output_file
number_of_cities_in_east_coast_states_where_prices_doubled_in_less_than_10_years = 0
number_of_cities_in_west_coast_states_where_prices_doubled_in_less_than_10_years = 0
total_number_of_cities_where_prices_doubled_in_less_than_10_years = 0
number_of_interior_state_cities = 0
total_doubling_time_of_interior_state_cities = 0
colorIcon = ""
for k in range(1, 122):
city1 = region_name_column_list[k]
state_abbr = city1[-2:]
# special cases
if city1 == "Denver, CO":
latitude = "39.7392"
longitude = "-104.9850"
elif city1 == "Richmond, VA":
latitude = "37.53"
longitude = "-77.47"
elif city1 == "Urban Honolulu, HI":
latitude = "21.315603"
longitude = "-157.858093"
elif city1 == "Ventura, CA":
latitude = "34.275"
longitude = "-119.228"
elif city1 == "North Port-Sarasota-Bradenton, FL":
cityX = "Sarasota, FL"
# find cityX's location
nominatim = Nominatim()
city_json = nominatim.query(cityX).toJSON()[0]
latitude = city_json["lat"]
longitude = city_json["lon"]
elif city1 == "Minneapolis-St Paul, MN":
cityY = "Minneapolis, MN"
# find cityY's location
nominatim = Nominatim()
city_json = nominatim.query(cityY).toJSON()[0]
latitude = city_json["lat"]
longitude = city_json["lon"]
else:
# find city1's location

nominatim = Nominatim()
```

```
city_json = nominatim.query(city1).toJSON()[0]
latitude = city_json["lat"]
longitude = city_json["lon"]
# get city1's size rank
sz = city_name_size_rank_dict[city1]
# Find how long it will take to double your initial investment in city1
idx = city_names_list.index(city1)
first = 0
last = float_df.shape[1]
row = float_df.iloc[idx, :]
# convert row to Panadas series
row_series = row.squeeze(axis=0)
rowmax = row_series.max()
rowmin = row_series.min()
string_starting_idx = row.idxmin()
numeric_starting_idx = 0
numeric_ending_idx = 0

month_dict = {}
months = float_df.columns.values.tolist()
for i in range(0, len(months)):
month_dict[months[i]] = i
numeric_starting_idx = month_dict[string_starting_idx]
double_price = rowmin * 2
current_price = 999999
for j in range(numeric_starting_idx, len(months)):
current_price = row[j]
if current_price >= double_price:
numeric_ending_idx = j
break
#convert numeric_ending_idx to string_ending_idx
string_ending_idx = months[numeric_ending_idx]
min_years = (numeric_ending_idx - numeric_starting_idx) / 12
if min_years <= 0:
min_years_1f = "unknown"
else:
min_years_1f = "{:.1f}".format(min_years)
# check if the city is in an interior state
if (state_abbr not in ["WA", "OR", "CA"]) and (state_abbr not in ["ME", "NH", "MA", "RI", "CT", "NY", "NJ", "DE", "MD", "DC", "VA", "NC", "SC", "GA", "FL"]):
number_of_interior_state_cities = number_of_interior_state_cities + 1
total_doubling_time_of_interior_state_cities = total_doubling_time_of_interior_state_cities + min_years
# if the city has a doubling time between 0 and 9.9 years, color it green
if (min_years > 0) and (min_years < 10.0):
colorIcon = 'greenIcon'
# increment the count
total_number_of_cities_where_prices_doubled_in_less_than_10_years = total_number_of_cities_where_prices_doubled_in_less_than_10_years + 1
# check if the city is in a West Coast state
if state_abbr in ["WA", "OR", "CA"]:

```

```
number_of_cities_in_west_coast_states_where_prices_doubled_in_less_than_10_years = number_of_cities_in_west_coast_states_where_prices_doubled_in_less_than_10_years + 1
# check if city is in an East Coast state
if state_abbr in ["ME", "NH", "MA", "RI", "CT", "NY", "NJ", "DE", "MD", "DC", "VA", "NC", "SC", "GA", "FL"]:
number_of_cities_in_east_coast_states_where_prices_doubled_in_less_than_10_years = number_of_cities_in_east_coast_states_where_prices_doubled_in_less_than_10_years + 1
# if the city has a doubling time between 10.0 and 19.9 years, color it yellow
elif (min_years >= 10.0) and (min_years < 20.0):
colorIcon = 'yellowIcon'
# if the city has a doubling time between 20.0 and 29.9 years, color it red
elif (min_years >= 20) and (min_years < 29.9):
colorIcon = 'blackIcon'
# if the median price for a single-family home did not double in that city between 1996 and 2021
else:
colorIcon = 'grayIcon'
if min_years_1f == "unknown":
print("L.marker([", latitude, ", ", longitude, "], {icon: ", colorIcon, "}).bindPopup(\"The median home price in ", city1, " did not double between 1996 and 2021.\").addTo(map);", sep="")
else:
print("L.marker([", latitude, ", ", longitude, "], {icon: ", colorIcon, "}).bindPopup(\"The shortest amount of time to double your money in ", city1, " was ", min_years_1f, " years.\").addTo(map);", sep="")

number_of_cities_in_states_on_either_coast_where_prices_doubled_in_less_than_10_years = number_of_cities_in_west_coast_states_where_prices_doubled_in_less_than_10_years + number_of_cities_in_east_coast_states_where_prices_doubled_in_less_than_10_years
fraction_of_cities_in_coastal_states_where_prices_doubled_in_less_than_10_years = number_of_cities_in_states_on_either_coast_where_prices_doubled_in_less_than_10_years / total_number_of_cities_where_prices_doubled_in_less_than_10_years
fraction_of_cities_in_coastal_states_where_prices_doubled_in_less_than_10_years_2f = "{:.2f}".format(fraction_of_cities_in_coastal_states_where_prices_doubled_in_less_than_10_years)
print('\n', fraction_of_cities_in_coastal_states_where_prices_doubled_in_less_than_10_years_2f, "percent of the cities that doubled in less than 10 years were located in a state that borders the Atlantic Ocean or the Pacific Ocean.")

average_doubling_time_of_interior_state_cities =
total_doubling_time_of_interior_state_cities / number_of_interior_state_cities
print('average_doubling_time_of_interior_state_cities:',
average_doubling_time_of_interior_state_cities)
sys.stdout = default_stdout
output_file.close()
sys.exit()
```