Identifying High-Potential Real Estate Investment Opportunities

Table of Contents

Refresh

Introduction

1. Abstract

1.1. Summary

Apex Assets Investment, a prominent real estate investment firm, is seeking strategic guidance to enhance their investment decisions in the USA market. Leveraging data from Zillow Research, specifically the Zillow Home Value Index (ZHVI), we have conducted a comprehensive Time Series Analysis to offer insights into

future property values, identify trends, patterns, and seasonality within the market. Our analysis aims to provide actionable recommendations to optimize Apex Assets Investment's portfolio construction.

1.2. Data Source

We utilized data from Zillow Research, which provides the Zillow Home Value Index (ZHVI), a robust measure of typical home values and market changes across various regions and housing types in the USA.

1.3. The Analysis

- Preprocessing and EDA
- Time Series Modelling
- Recommendations
- Conclusion

2.0. Overview

Apex Assets Investments is a real estate firm which provides state of art opportunities in the real estate scope such as housing. However, it is grappling with a challenge on the best avenues to do the investing. The primary goal of this project is to analyse the factors that will determine the best zipcodes to invest in and build time series model that will forecast the real estate prices of various zipcodes and provides insights and recommendations based on the built time series model in this project. The data used can be accessed from https://www.zillow.com/research/data/.

3.0. Business Problem

In today's competitive real estate market, Apex Assets Investments faces the challenge of finding the best lucrative places to invest. With countless zip codes, fluctuating prices, the pursuit of optimal investment destinations poses a task that can feel overwhelming. The objective is clear: pinpoint the top 5 zip codes with the highest potential for profit while managing risks wisely.

To tackle this challenge, Apex Assets Investments turns to data and analysis using the time serries model. By studying past trends and using advanced forecasting techniques, they aim to uncover hidden opportunities in different zipcodes. But it's not just about numbers—it's also about understanding communities, local developments, and what makes each area unique.

With this approach, Apex Assets Investments can make informed decisions that not only benefit their investors but also contribute positively to the neighborhoods they invest in. Ultimately, it's about finding the right balance between growth and stability in the ever-changing world of real estate.

4.0. Objectives

Evaluate which county exhibits the most promising highend real estate investment opportunities. Understand the trend for the 5 best Zipcodes to invest in based on Return On Investment(ROI). Forecast property values over the short and long term, aiming to identify the most favourable zip codes for investment across various counties.

5.0. Data

Our dataset comes from Zillow, and tracks the median home price for many of the country's major metro areas. These metro areas are denoted by zillows internal 'Msa' designation, which stands for 'Metropolitan Statistical Area', meaning there are more than 50,000 people living in an urbanized area. The raw dataset covers 909 American metro areas, tracking the median home value for each from April of 1996 to April of 2018

It contains the below columns

1. RegionID: Unique index
2. RegionName: Unique Zip Code
3. City: City in which the zip code is located
4. State: State in which the zip code is located
5. Metro: Metropolitan Area in which the zip code is located
6. CountyName: County in which the zip code is located
7. SizeRank: Numerical rank of size of zip code,
8. 1996-04 through 2018-04:

6.0. Data Preprocessing and EDA

Performing exploratory analysis to extract meaningful insights from the data and identify the best features to be used for modeling.

Data pre-processing

- Assumptions about data shape
- Missing values
- Data types
- Categorical variables
- Outliers or errors
- Feature Engineering/Creating features

EDA

Performing initial investigations on data to discover patterns to check assumptions with the help of summary statistics and graphical representations.

- Descriptive statistics

- Visualization
- Correlation
- Conclusion

6.1 Pre-processing

Importing the necessary libraries for the data analysis

```
#Importing libraries

import pandas as pd

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
import sklearn
import seaborn as sns
import warnings
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

warnings.filterwarnings('ignore')
plt.rcParams["figure.figsize"] = [10,5]

!pip install pmdarima
```

```
Requirement already satisfied: pmdarima in /usr/local/lib/python3.10/dist-packages (2.0.4)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from pmda
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in /usr/local/lib/python3.10/dist
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from pmda
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.10/dist-packages (from pmda
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from pmda
Requirement already satisfied: statsmodels>=0.13.2 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from pmdarima)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging>=17.1 in /usr/local/lib/python3.10/dist-packages (from pr
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from panda
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: patsy>=0.5.4 in /usr/local/lib/python3.10/dist-packages (from stats
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.4-
```

```
# Loading the data from the CSV file into a DataFrame
zillow_dataset = pd.read_csv("/content/zillow_data.csv")

#date shape
data_shape = zillow_dataset.shape
print("Shape of the dataset:", data_shape)
```

```
Shape of the dataset: (14723, 272)
```

```
#to preview the 1st 5 rows
zillow_dataset.head()
```

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	1996-0
0	84654	60657	Chicago	IL	Chicago	Cook	1	334200.0	335400.0	336500
1	90668	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	235700.0	236900.0	236700
2	91982	77494	Katy	TX	Houston	Harris	3	210400.0	212200.0	212200
3	84616	60614	Chicago	IL	Chicago	Cook	4	498100.0	500900.0	503100
4	93144	79936	El Paso	TX	El Paso	El Paso	5	77300.0	77300.0	77300

```
#to preview the last 5 rows
zillow_dataset.tail()
```

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	1996-05
14718	58333	1338	Ashfield	MA	Greenfield Town	Franklin	14719	94600.0	94300.0
14719	59107	3293	Woodstock	NH	Claremont	Grafton	14720	92700.0	92500.0
14720	75672	40404	Berea	KY	Richmond	Madison	14721	57100.0	57300.0
14721	93733	81225	Mount Crested Butte	CO	NaN	Gunnison	14722	191100.0	192400.0
14722	95851	89155	Mesquite	NV	Las Vegas	Clark	14723	176400.0	176300.0

```
def display_dataset_info(zillow_dataset):
    """
    Function to display information about a dataset including info() and the first few rows visually w

    Parameters:
    - df: DataFrame
        The dataset to be examined.
    """
    # Display the info of the dataset
    print("\033[1mInfo of the dataset:\033[0m")
    print(f"Number of rows: {zillow_dataset.shape[0]}")
    print(f"Number of columns: {zillow_dataset.shape[1]}")
    print(zillow_dataset.info())

    # Display the first few rows of the DataFrame
    print("\n\033[1mFirst few rows of the DataFrame:\033[0m")
    display(zillow_dataset.head())

display_dataset_info(zillow_dataset)
```

Info of the dataset:

```
Number of rows: 14723
Number of columns: 272
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14723 entries, 0 to 14722
Columns: 272 entries, RegionID to 2018-04
dtypes: float64(219), int64(49), object(4)
memory usage: 30.6+ MB
None
```

First few rows of the DataFrame:

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	1996-0
0	84654	60657	Chicago	IL	Chicago	Cook	1	334200.0	335400.0	336500
1	90668	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	235700.0	236900.0	236700
2	91982	77494	Katy	TX	Houston	Harris	3	210400.0	212200.0	212200
3	84616	60614	Chicago	IL	Chicago	Cook	4	498100.0	500900.0	503100
4	93144	79936	El Paso	TX	El Paso	El Paso	5	77300.0	77300.0	77300

```
# Exploring the data to check fir missing values and more about the data
def analyze_dataset(zillow_dataset):
    """
    Function to analyze the dataset including missing values, data types, and summary statistics.

    Parameters:
    - df: DataFrame
        The dataset to be analyzed.
    """
    # Check for missing values
    missing_values = zillow_dataset.isnull().sum()
    print("\033[1mMissing Values:\033[0m")
    print(missing_values)

    # Get data types
    data_types = zillow_dataset.dtypes.unique()
    print("\n\033[1mUnique Data Types:\033[0m")
    print(data_types)

    # Compute summary statistics for numerical columns
    numeric_summary = zillow_dataset.describe()
    print("\n\033[1mSummary Statistics for Numerical Columns:\033[0m")
    print(numeric_summary)

# Example usage:
analyze_dataset(zillow_dataset)
```

Missing Values:

```
RegionID         0
RegionName       0
City             0
State            0
Metro         1043
              ...
2017-12          0
2018-01          0
2018-02          0
2018-03          0
2018-04          0
Length: 272, dtype: int64
```

Unique Data Types:

```
[dtype('int64') dtype('O') dtype('float64')]
```

Summary Statistics for Numerical Columns:

```
            RegionID    RegionName      SizeRank       1996-04       1996-05  \
count   14723.000000  14723.000000  14723.000000  1.368400e+04  1.368400e+04
mean    81075.010052  48222.348706   7362.000000  1.182991e+05  1.184190e+05
std     31934.118525  29359.325439   4250.308342  8.600251e+04  8.615567e+04
min     58196.000000   1001.000000      1.000000  1.130000e+04  1.150000e+04
25%     67174.500000  22101.500000   3681.500000  6.880000e+04  6.890000e+04
50%     78007.000000  46106.000000   7362.000000  9.950000e+04  9.950000e+04
75%     90920.500000  75205.500000  11042.500000  1.432000e+05  1.433000e+05
max    753844.000000  99901.000000  14723.000000  3.676700e+06  3.704200e+06

            1996-06       1996-07       1996-08       1996-09       1996-10  \
count  1.368400e+04  1.368400e+04  1.368400e+04  1.368400e+04  1.368400e+04
mean   1.185374e+05  1.186531e+05  1.187803e+05  1.189275e+05  1.191205e+05
```

```
std    8.630923e+04  8.646795e+04  8.665094e+04  8.687208e+04  8.715185e+04
min    1.160000e+04  1.180000e+04  1.180000e+04  1.200000e+04  1.210000e+04
25%    6.910000e+04  6.920000e+04  6.937500e+04  6.950000e+04  6.960000e+04
50%    9.970000e+04  9.970000e+04  9.980000e+04  9.990000e+04  9.995000e+04
75%    1.432250e+05  1.432250e+05  1.435000e+05  1.437000e+05  1.439000e+05
max    3.729600e+06  3.754600e+06  3.781800e+06  3.813500e+06  3.849600e+06

       ...       2017-07       2017-08       2017-09       2017-10  \
count  ...  1.472300e+04  1.472300e+04  1.472300e+04  1.472300e+04
mean   ...  2.733354e+05  2.748658e+05  2.764646e+05  2.780332e+05
std    ...  3.603984e+05  3.614678e+05  3.627563e+05  3.644610e+05
min    ...  1.440000e+04  1.450000e+04  1.470000e+04  1.480000e+04
25%    ...  1.269000e+05  1.275000e+05  1.282000e+05  1.287000e+05
50%    ...  1.884000e+05  1.896000e+05  1.905000e+05  1.914000e+05
75%    ...  3.050000e+05  3.066500e+05  3.085000e+05  3.098000e+05
max    ...  1.888990e+07  1.870350e+07  1.860530e+07  1.856940e+07

            2017-11       2017-12       2018-01       2018-02       2018-03  \
count  1.472300e+04  1.472300e+04  1.472300e+04  1.472300e+04  1.472300e+04
mean   2.795209e+05  2.810953e+05  2.826571e+05  2.843687e+05  2.865114e+05
std    3.656003e+05  3.670454e+05  3.695727e+05  3.717739e+05  3.724612e+05
min    1.450000e+04  1.430000e+04  1.410000e+04  1.390000e+04  1.380000e+04
25%    1.292500e+05  1.299000e+05  1.306000e+05  1.310500e+05  1.319500e+05
50%    1.925000e+05  1.934000e+05  1.941000e+05  1.950000e+05  1.967000e+05
75%    3.117000e+05  3.134000e+05  3.151000e+05  3.168500e+05  3.188500e+05
max    1.842880e+07  1.830710e+07  1.836590e+07  1.853040e+07  1.833770e+07
```

Missing Values:

- The output displays the count of missing values for each column in the dataset.
- For instance, the column `Metro` has 1043 missing values, indicating that 1043 entries in the `Metro` column are NaN or null.

Unique Data Types:

- This section presents the unique data types found in the dataset.
- In this case, there are three unique data types:
  - `int64` : Integer data type.
  - `O` : Object data type, often representing strings or mixed data types.
  - `float64` : Floating-point numeric data type.

Summary Statistics for Numerical Columns:

- Here, summary statistics for numerical columns are provided.
- The statistics include count, mean, standard deviation, minimum, maximum, and quartiles for each numerical column.
- For example, the `RegionID` column has a mean value of approximately 81075 and a standard deviation of around 31934.

This output provides essential insights into the dataset, including missing values, data types, and distribution of numerical data, which are crucial for data understanding and preprocessing.

```
zillow_dataset.iloc[:,0:7].nunique()
```

```
RegionID      14723
RegionName    14723
City           7554
State            51
Metro           701
CountyName     1212
SizeRank      14723
dtype: int64
```

Observations:

- The dataset contains 14722 rows and 272 colums
- Some columns in the dataset contain missing values
- Data has both continuous and categorical features comprising of the following data types; objects, integers, floats
- The dataset is in wide format with the time periods appearing as columns. We will need to convert it into long format

Data Preparation.

```
def get_datetimes(zillow_dataset):
    """
    Takes a dataframe:
    returns only those column names that can be converted into datetime objects
    as datetime objects.
    NOTE number of returned columns may not match total number of columns in passed dataframe
    """

    return pd.to_datetime(zillow_dataset.columns.values[1:], format='%Y-%m')
```

This function get_datetimes(df) takes a DataFrame as input and returns only those column names that can be converted into datetime objects, treating them as datetime objects.

It assumes that the first column of the DataFrame is not a datetime column, so it starts converting from the second column (df.columns.values[1:]). The datetime conversion is performed using pd.to_datetime() function with the specified format '%Y-%m'. It returns the converted datetime columns. Note: The number of returned columns may not match the total number of columns in the passed DataFrame.

To use this function, you would call get_datetimes(df), passing the DataFrame you want to extract datetime columns from as df.

Rename RegionName to ZipCode

Per the column descriptions, RegionName actually means ZipCode. For reading purposes we renamed the column to ZipCode to avoid any confusion.

```
# Rename the "RegionName" column to "ZipCode"
zillow_dataset = zillow_dataset.rename(columns={'RegionName': 'ZipCode'})

# Check the updated DataFrame
zillow_dataset.head()
```

	RegionID	ZipCode	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	1996-06
0	84654	60657	Chicago	IL	Chicago	Cook	1	334200.0	335400.0	336500.0
1	90668	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	235700.0	236900.0	236700.0
2	91982	77494	Katy	TX	Houston	Harris	3	210400.0	212200.0	212200.0
3	84616	60614	Chicago	IL	Chicago	Cook	4	498100.0	500900.0	503100.0
4	93144	79936	El Paso	TX	El Paso	El Paso	5	77300.0	77300.0	77300.0

```
# Check the amount of missing values in the entire dataset
total_missing_values = zillow_dataset.isnull().sum().sum()
print(f"Number of missing values in the entire dataset: {total_missing_values}")
```

```
Number of missing values in the entire dataset: 157934
```

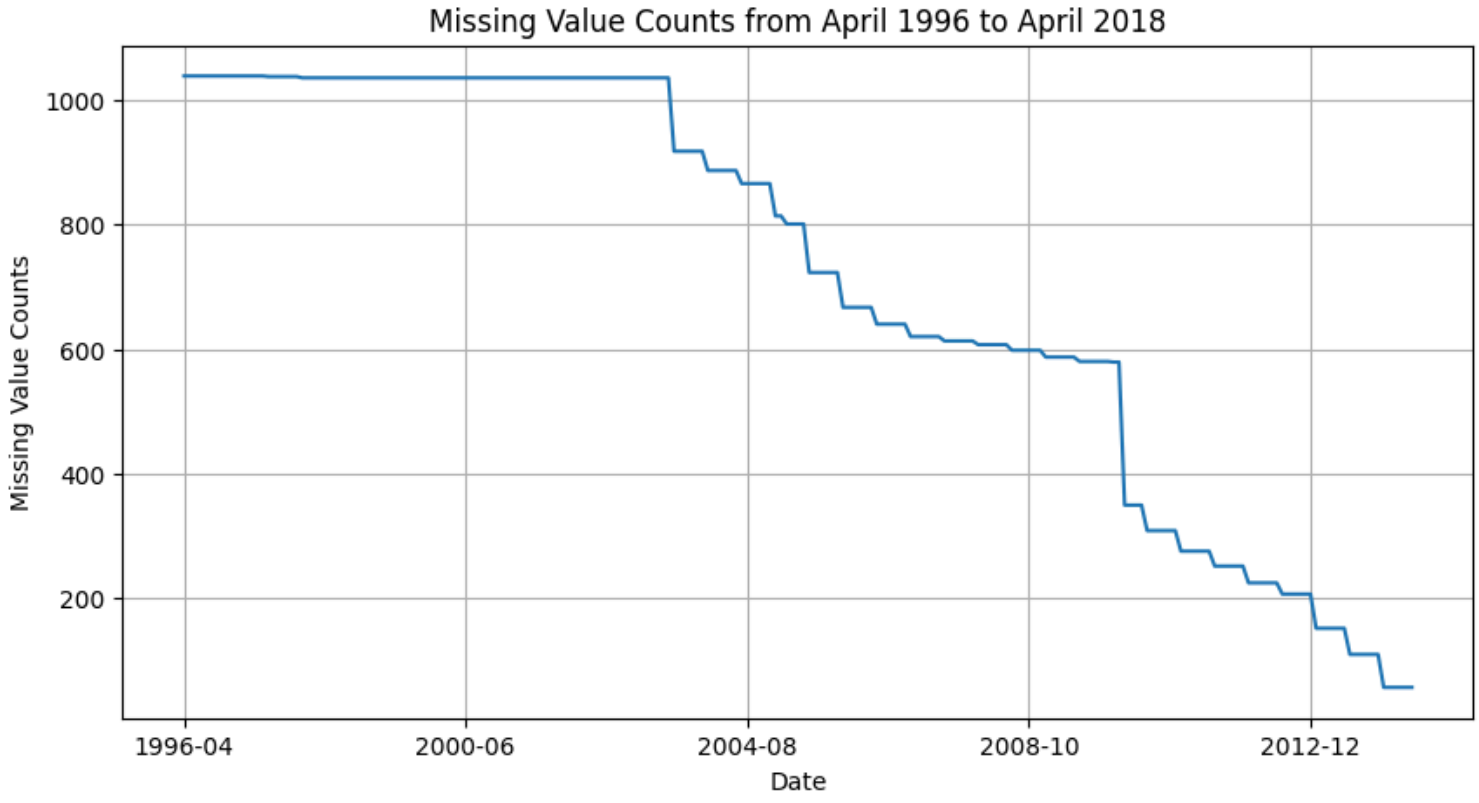
```
import pandas as pd

def explore_and_transpose_missing_data(zillow_dataset):
    """
    Function to explore missing data in a DataFrame and transpose the result.

    Parameters:
    - df: DataFrame
        The DataFrame to explore.

    Returns:
    - DataFrame
        Transposed DataFrame containing columns with missing values along with their counts.
    """
    # Set option to display all columns
    pd.set_option('display.max_columns', None)

    def explore_missing_data(zillow_dataset):
        """
        Function to explore missing data in a DataFrame.

        Parameters:
        - df: DataFrame
            The DataFrame to explore.

        Returns:
        - DataFrame
            DataFrame containing columns with missing values along with their counts.
        """
        # Calculate the count of missing values for each column
        missing_data = zillow_dataset.isna().sum()

        # Filter columns with missing values (count > 0)
        missing_data = missing_data[missing_data > 0]

        # Convert the Series to a DataFrame
        missing_data_df = missing_data.to_frame(name='Missing Values')

        return missing_data_df

    # Call explore_missing_data function
    missing_data_df = explore_missing_data(zillow_dataset)

    # Transpose the DataFrame returned by missing_data function
    transposed_missing_data = missing_data_df.T

    return transposed_missing_data

transposed_missing_data = explore_and_transpose_missing_data(zillow_dataset)
print("Columns with missing values:")
transposed_missing_data
```

```
Columns with missing values:
```

	Metro	1996-04	1996-05	1996-06	1996-07	1996-08	1996-09	1996-10	1996-11	1996-12	1997-01	1997-02	1997-03
Missing Values	1043	1039	1039	1039	1039	1039	1039	1039	1039	1039	1039	1039	1039

```
# Extract the desired time range for missing value counts
start_date = '1996-04'
end_date = '2014-06'
missing_counts = transposed_missing_data.loc[:, start_date:end_date]

# Plot the missing value counts for each column
plt.figure(figsize=(10, 6))
missing_counts.T.plot(legend=False)
plt.title('Missing Value Counts from April 1996 to April 2018')
plt.xlabel('Date')
plt.ylabel('Missing Value Counts')
plt.grid(True)
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



The graph shows a downward trajectory, it means that the number of missing values in the specified columns decreases over time within the given range (from April 1996 to June 2014). Here's the interpretation:

- Decreasing Number of Missing Values:

A downward trajectory indicates that, over the specified time range, the dataset becomes more complete as fewer missing values are observed. This can be a positive sign, suggesting that data collection or recording processes improved over time, resulting in fewer missing values in the dataset.

```
def clean_missing_data_ffill(zillow_dataset):
    """
    Function to clean missing data in a DataFrame using forward fill (ffill) method.

    Parameters:
    - df: DataFrame
        The DataFrame to clean.

    Returns:
    - DataFrame
        DataFrame with missing values filled using forward fill method.
    """
    # Apply forward fill to fill missing values
    cleaned_df = zillow_dataset.fillna(method='ffill')

    # Sanity check: Check if there are any missing values after forward fill
    if cleaned_df.isna().sum().sum() == 0:
        print("No missing values found after forward fill.")
    else:
        print("There are still missing values in the cleaned DataFrame.")

    return cleaned_df

# Example usage:
# Assuming df is your DataFrame
# Clean missing values using forward fill
cleaned_df = clean_missing_data_ffill(zillow_dataset)

# Print the cleaned DataFrame
print("Cleaned DataFrame:")
print(cleaned_df)
```

```
No missing values found after forward fill.
Cleaned DataFrame:
       RegionID  ZipCode                City State              Metro  \
0         84654    60657             Chicago    IL            Chicago
1         90668    75070            McKinney    TX  Dallas-Fort Worth
2         91982    77494                Katy    TX            Houston
3         84616    60614             Chicago    IL            Chicago
4         93144    79936             El Paso    TX            El Paso
...         ...      ...                 ...   ...                ...
14718     58333     1338            Ashfield    MA    Greenfield Town
14719     59107     3293           Woodstock    NH           Claremont
14720     75672    40404               Berea    KY            Richmond
14721     93733    81225  Mount Crested Butte    CO            Richmond
14722     95851    89155             Mesquite    NV           Las Vegas

      CountyName  SizeRank   1996-04   1996-05   1996-06   1996-07   1996-08  \
```

```
      0           Cook         1  334200.0  335400.0  336500.0  337600.0  338500.0
      1         Collin         2  235700.0  236900.0  236700.0  235400.0  233300.0
      2         Harris         3  210400.0  212200.0  212200.0  210700.0  208300.0
      3           Cook         4  498100.0  500900.0  503100.0  504600.0  505500.0
      4        El Paso         5   77300.0   77300.0   77300.0   77300.0   77400.0
      ...          ...       ...       ...       ...       ...       ...       ...
      14718   Franklin     14719   94600.0   94300.0   94000.0   93700.0   93400.0
      14719    Grafton     14720   92700.0   92500.0   92400.0   92200.0   92100.0
      14720    Madison     14721   57100.0   57300.0   57500.0   57700.0   58000.0
      14721   Gunnison     14722  191100.0  192400.0  193700.0  195000.0  196300.0
      14722      Clark     14723  176400.0  176300.0  176100.0  176000.0  175900.0

             1996-09   1996-10   1996-11   1996-12   1997-01   1997-02   1997-03  \
      0      339500.0  340400.0  341300.0  342600.0  344400.0  345700.0  346700.0
      1      230600.0  227300.0  223400.0  219600.0  215800.0  211100.0  205700.0
      2      205500.0  202500.0  199800.0  198300.0  197300.0  195400.0  193000.0
      3      505700.0  505300.0  504200.0  503600.0  503400.0  502200.0  500000.0
      4       77500.0   77600.0   77700.0   77700.0   77800.0   77900.0   77900.0
      ...         ...       ...       ...       ...       ...       ...       ...
      14718   93200.0   93000.0   92900.0   92700.0   92600.0   92600.0   92600.0
      14719   91900.0   91700.0   91300.0   90900.0   90500.0   90100.0   89800.0
      14720   58200.0   58400.0   58700.0   59100.0   59500.0   59900.0   60300.0
      14721  197700.0  199100.0  200700.0  202600.0  204900.0  207100.0  209100.0
      14722  175800.0  175800.0  176000.0  176200.0  176500.0  176700.0  176800.0

             1997-04   1997-05   1997-06   1997-07   1997-08   1997-09   1997-10  \
      0      347800.0  349000.0  350400.0  352000.0  353900.0  356200.0  358800.0
      1      200900.0  196800.0  193600.0  191400.0  190400.0  190800.0  192700.0
      2      191800.0  191800.0  193000.0  195200.0  198400.0  202800.0  208000.0
      3      497900.0  496300.0  495200.0  494700.0  494900.0  496200.0  498600.0
      4       77800.0   77800.0   77800.0   77800.0   77800.0   77900.0   78100.0
      ...         ...       ...       ...       ...       ...       ...       ...
      14718   92800.0   93100.0   93500.0   94000.0   94500.0   95100.0   95800.0
      14719   89600.0   89400.0   89300.0   89200.0   89100.0   88900.0   88700.0
      14720   60800.0   61300.0   62000.0   62600.0   63200.0   63800.0   64500.0
      14721  211200.0  213400.0  215800.0  218300.0  221000.0  223900.0  226900.0
      14722  176800.0  176900.0  177000.0  177100.0  177200.0  177400.0  177600.0

             1997-11   1997-12   1998-01   1998-02   1998-03   1998-04   1998-05  \
      0      361800.0  365700.0  370200.0  374700.0  378900.0  383500.0  388300.0
      1      196000.0  201300.0  207400.0  212200.0  214600.0  215100.0  213400.0
```

```
# checking for duplicates

print(f'The data has {zillow_dataset.duplicated().sum()} duplicates')
```

```
      The data has 0 duplicates
```

Explanation of Duplicate Checking Output

The output "The data has 0 duplicates" indicates that there are no duplicate rows present in the DataFrame. This means that each row in the DataFrame is unique, and there are no exact duplicates of any row.

In summary, checking for duplicates is an essential step in the data cleaning process to ensure data quality, accuracy in analysis, and optimal performance in modeling tasks.

Uniformity

```
# Convert Zipcode column to string type
cleaned_df['ZipCode'] = cleaned_df['ZipCode'].astype(str)

# Ensure all zip codes are 5 digits long by adding leading zeros
cleaned_df['ZipCode'] = cleaned_df['ZipCode'].apply(lambda x: x.zfill(5))

# Output the updated DataFrame and check the minimum and maximum zip codes
print("Updated DataFrame:")
print(cleaned_df)
print("Minimum Zipcode:", cleaned_df['ZipCode'].min())
print("Maximum Zipcode:", cleaned_df['ZipCode'].max())
```

```
Updated DataFrame:
       RegionID ZipCode                City State              Metro  \
0         84654   60657             Chicago    IL            Chicago
1         90668   75070             McKinney    TX  Dallas-Fort Worth
2         91982   77494                Katy    TX            Houston
3         84616   60614             Chicago    IL            Chicago
4         93144   79936             El Paso    TX            El Paso
...         ...     ...                 ...   ...                ...
14718     58333   01338             Ashfield    MA    Greenfield Town
14719     59107   03293            Woodstock    NH          Claremont
14720     75672   40404                Berea    KY           Richmond
14721     93733   81225  Mount Crested Butte    CO           Richmond
14722     95851   89155             Mesquite    NV          Las Vegas

      CountyName  SizeRank   1996-04   1996-05   1996-06   1996-07   1996-08  \
0           Cook         1  334200.0  335400.0  336500.0  337600.0  338500.0
1         Collin         2  235700.0  236900.0  236700.0  235400.0  233300.0
2         Harris         3  210400.0  212200.0  212200.0  210700.0  208300.0
3           Cook         4  498100.0  500900.0  503100.0  504600.0  505500.0
4        El Paso         5   77300.0   77300.0   77300.0   77300.0   77400.0
...          ...       ...       ...       ...       ...       ...       ...
14718   Franklin     14719   94600.0   94300.0   94000.0   93700.0   93400.0
14719    Grafton     14720   92700.0   92500.0   92400.0   92200.0   92100.0
14720    Madison     14721   57100.0   57300.0   57500.0   57700.0   58000.0
14721   Gunnison     14722  191100.0  192400.0  193700.0  195000.0  196300.0
14722      Clark     14723  176400.0  176300.0  176100.0  176000.0  175900.0

        1996-09   1996-10   1996-11   1996-12   1997-01   1997-02   1997-03  \
0      339500.0  340400.0  341300.0  342600.0  344400.0  345700.0  346700.0
1      230600.0  227300.0  223400.0  219600.0  215800.0  211100.0  205700.0
2      205500.0  202500.0  199800.0  198300.0  197300.0  195400.0  193000.0
3      505700.0  505300.0  504200.0  503600.0  503400.0  502200.0  500000.0
4       77500.0   77600.0   77700.0   77700.0   77800.0   77900.0   77900.0
...         ...       ...       ...       ...       ...       ...       ...
14718   93200.0   93000.0   92900.0   92700.0   92600.0   92600.0   92600.0
14719   91900.0   91700.0   91300.0   90900.0   90500.0   90100.0   89800.0
14720   58200.0   58400.0   58700.0   59100.0   59500.0   59900.0   60300.0
14721  197700.0  199100.0  200700.0  202600.0  204900.0  207100.0  209100.0
14722  175800.0  175800.0  176000.0  176200.0  176500.0  176700.0  176800.0

        1997-04   1997-05   1997-06   1997-07   1997-08   1997-09   1997-10  \
0      347800.0  349000.0  350400.0  352000.0  353900.0  356200.0  358800.0
```

```
1      200900.0  196800.0  193600.0  191400.0  190400.0  190800.0  192700.0
2      191800.0  191800.0  193000.0  195200.0  198400.0  202800.0  208000.0
3      497900.0  496300.0  495200.0  494700.0  494900.0  496200.0  498600.0
4       77800.0   77800.0   77800.0   77800.0   77800.0   77900.0   78100.0
...         ...       ...       ...       ...       ...       ...       ...
14718   92800.0   93100.0   93500.0   94000.0   94500.0   95100.0   95800.0
14719   89600.0   89400.0   89300.0   89200.0   89100.0   88900.0   88700.0
14720   60800.0   61300.0   62000.0   62600.0   63200.0   63800.0   64500.0
14721  211200.0  213400.0  215800.0  218300.0  221000.0  223900.0  226900.0
14722  176800.0  176900.0  177000.0  177100.0  177200.0  177400.0  177600.0

        1997-11   1997-12   1998-01   1998-02   1998-03   1998-04   1998-05  \
0      361800.0  365700.0  370200.0  374700.0  378900.0  383500.0  388300.0
1      196000.0  201300.0  207400.0  212200.0  214600.0  215100.0  213400.0
2      213800.0  220700.0  227500.0  231800.0  233400.0  233900.0  233500.0
3      502000.0  507600.0  514900.0  522200.0  529500.0  537900.0  546900.0
```

This code first converts the 'Zipcode' column to string type since Zipcodes represent locations and so should be categorical data types. Some zipcodes have four digits and others five. The column needs to be restructured to ensure all the digits are five in number. The columns with four digits seem to be missing a zero at the beginning.Then, it uses the zfill() method to pad the zip codes with leading zeros to make them all 5 digits long. Finally, it prints the updated DataFrame and checks the minimum and maximum zip codes to verify that all zip codes are now 5 digits long.

```
#Inspecting dataframe dtypes

cleaned_df.iloc[:, :10].dtypes
```

```
RegionID        int64
ZipCode        object
City           object
State          object
Metro          object
CountyName     object
SizeRank        int64
1996-04       float64
1996-05       float64
1996-06       float64
dtype: object
```

5.2. Exploratory Data Analysis and Visualization

Summary Statistics

```
#Current dataframe
cleaned_df
```

	RegionID	ZipCode	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	19
0	84654	60657	Chicago	IL	Chicago	Cook	1	334200.0	335400.0	336
1	90668	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	235700.0	236900.0	236
2	91982	77494	Katy	TX	Houston	Harris	3	210400.0	212200.0	212
3	84616	60614	Chicago	IL	Chicago	Cook	4	498100.0	500900.0	503
4	93144	79936	El Paso	TX	El Paso	El Paso	5	77300.0	77300.0	77
...	...	...	...	...	...	...	...	...	...	
14718	58333	01338	Ashfield	MA	Greenfield Town	Franklin	14719	94600.0	94300.0	94
14719	59107	03293	Woodstock	NH	Claremont	Grafton	14720	92700.0	92500.0	92
14720	75672	40404	Berea	KY	Richmond	Madison	14721	57100.0	57300.0	57
14721	93733	81225	Mount Crested Butte	CO	Richmond	Gunnison	14722	191100.0	192400.0	193
14722	95851	89155	Mesquite	NV	Las Vegas	Clark	14723	176400.0	176300.0	176

14723 rows × 272 columns

```
# Summary statistics
summary_stats = cleaned_df.describe()
print("Summary Statistics:")
summary_stats
```

```
Summary Statistics:
```

	RegionID	SizeRank	1996-04	1996-05	1996-06	1996-07	199
count	14723.000000	14723.000000	1.472300e+04	1.472300e+04	1.472300e+04	1.472300e+04	1.472300
mean	81075.010052	7362.000000	1.173483e+05	1.174700e+05	1.175896e+05	1.177060e+05	1.178331
std	31934.118525	4250.308342	8.512966e+04	8.527906e+04	8.542898e+04	8.558392e+04	8.576193
min	58196.000000	1.000000	1.130000e+04	1.150000e+04	1.160000e+04	1.180000e+04	1.180000
25%	67174.500000	3681.500000	6.790000e+04	6.790000e+04	6.810000e+04	6.840000e+04	6.850000
50%	78007.000000	7362.000000	9.830000e+04	9.840000e+04	9.850000e+04	9.860000e+04	9.860000
75%	90920.500000	11042.500000	1.420000e+05	1.421500e+05	1.424000e+05	1.425000e+05	1.425000
max	753844.000000	14723.000000	3.676700e+06	3.704200e+06	3.729600e+06	3.754600e+06	3.781800

Filtering the data to specific range suitable for our analysis

In preparation for delving into the high-end property market, Apex Assets Investment seeks to conduct a focused Exploratory Data Analysis (EDA) to gain insights into properties with an average price exceeding USD 1,000,000. This tailored analysis aims to provide Apex Assets Investment with a deeper understanding of the high-end real estate segment, facilitating informed investment decisions and strategic planning.

```
# Generate the list of date columns
date_columns = [col for col in cleaned_df.columns if col.startswith('1996') or col.startswith('1997')

# Filter the DataFrame using the dynamically generated date columns
filtered_data = cleaned_df[date_columns]

# Calculate the average property price for each row
average_prices = filtered_data.mean(axis=1)

# Filter rows where the average price is greater than or equal to $1,000,000
high_end_properties = cleaned_df[average_prices >= 1000000]

# Print or further analyze the high-end properties DataFrame
#print(high_end_properties)

high_end_properties
```

	RegionID	ZipCode	City	State	Metro	CountyName	SizeRank	1996-04	1996-05
9	97564	94109	San Francisco	CA	San Francisco	San Francisco	10	766000.0	771100.0
20	61625	10011	New York	NY	New York	New York	21	142600.0	143100.0
21	61703	10128	New York	NY	New York	New York	22	3676700.0	3704200.0
30	96027	90046	Los Angeles	CA	Los Angeles-Long Beach-Anaheim	Los Angeles	31	340600.0	341700.0
84	72442	33139	Miami Beach	FL	Miami-Fort Lauderdale	Miami-Dade	85	480200.0	480700.0
...	...	...	...	...	...	...	...	...	...
13885	96639	92091	Rancho Santa Fe	CA	San Diego	San Diego	13886	743100.0	746600.0
13900	60722	07620	Alpine	NJ	New York	Bergen	13901	1075400.0	1075900.0
14622	71578	31561	Sea Island	GA	Brunswick	Glynn	14623	13500.0	13500.0
14641	97901	94970	Stinson Beach	CA	San Francisco	Marin	14642	411500.0	414400.0
14711	95893	89413	Glenbrook	NV	Gardnerville Ranchos	Douglas	14712	562400.0	562800.0

118 rows × 272 columns

To Narrow down to the top 10 zip codes based on the average house prices

```
# Calculate the average house price
high_end_properties['AverageHousePrice'] = high_end_properties.mean(axis=1)

# Display the DataFrame with the new column
high_end_properties
```

	RegionID	ZipCode	City	State	Metro	CountyName	SizeRank	1996-04	1996-05
9	97564	94109	San Francisco	CA	San Francisco	San Francisco	10	766000.0	771100.0
20	61625	10011	New York	NY	New York	New York	21	142600.0	143100.0
21	61703	10128	New York	NY	New York	New York	22	3676700.0	3704200.0
30	96027	90046	Los Angeles	CA	Los Angeles-Long Beach-Anaheim	Los Angeles	31	340600.0	341700.0
84	72442	33139	Miami Beach	FL	Miami-Fort Lauderdale	Miami-Dade	85	480200.0	480700.0
...	...	...	...	...	...	...	...	...	...
13885	96639	92091	Rancho Santa Fe	CA	San Diego	San Diego	13886	743100.0	746600.0
13900	60722	07620	Alpine	NJ	New York	Bergen	13901	1075400.0	1075900.0
14622	71578	31561	Sea Island	GA	Brunswick	Glynn	14623	13500.0	13500.0
14641	97901	94970	Stinson Beach	CA	San Francisco	Marin	14642	411500.0	414400.0
14711	95893	89413	Glenbrook	NV	Gardnerville Ranchos	Douglas	14712	562400.0	562800.0

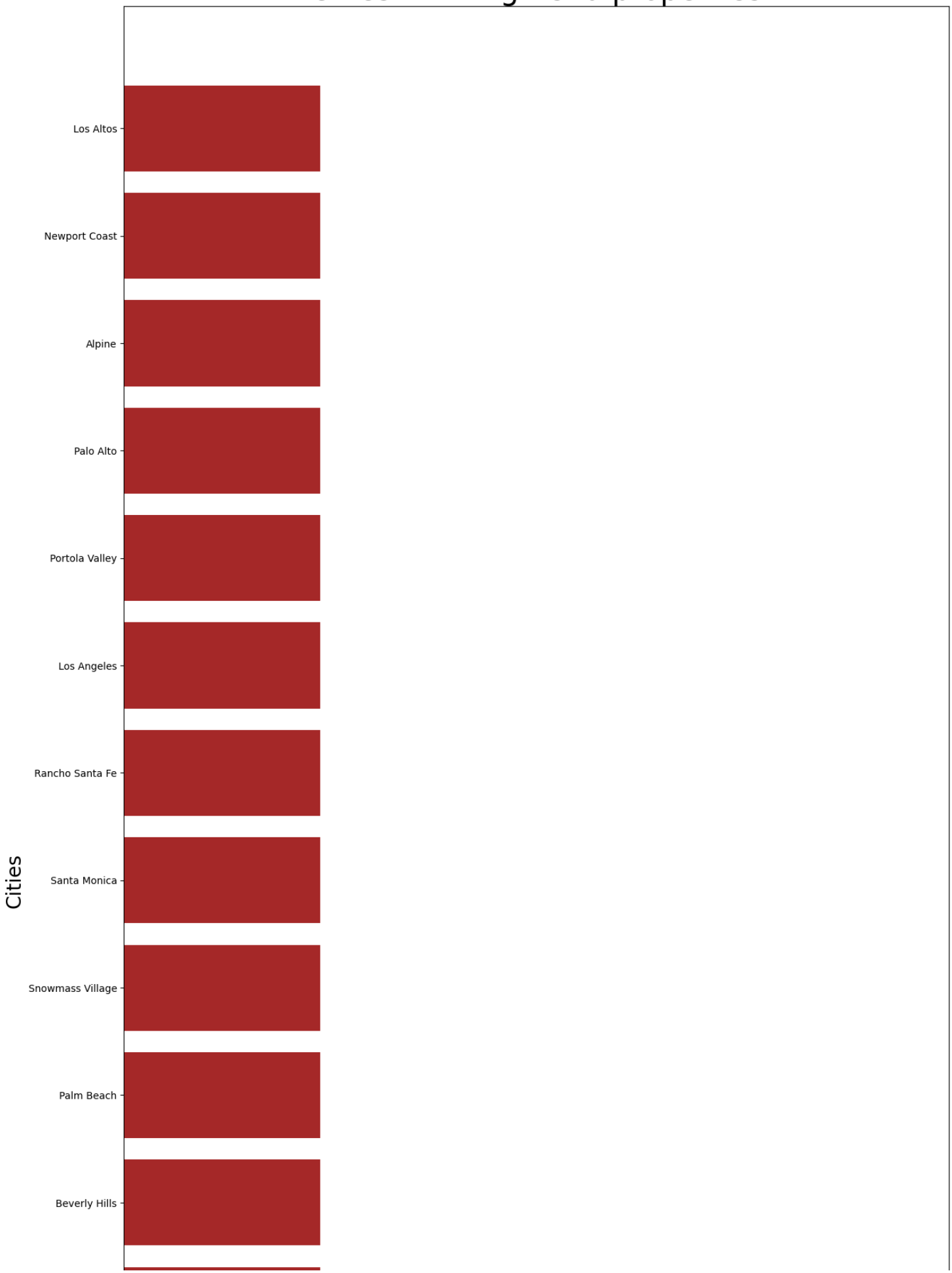
118 rows × 273 columns

```
# Sort the DataFrame by average house prices in descending order
sorted_df = high_end_properties.sort_values(by='AverageHousePrice', ascending=False)

# Select the top 20 rows
top_20 = sorted_df.head(20)

# Print or further analyze the top 20 rows
top_20
```

	RegionID	ZipCode	City	State	Metro	CountyName	SizeRank	1996-04	1996-05
272	61635	10021	New York	NY	New York	New York	273	51800.0	52000.0
21	61703	10128	New York	NY	New York	New York	22	3676700.0	3704200.0
20	61625	10011	New York	NY	New York	New York	21	142600.0	143100.0
508	61628	10014	New York	NY	New York	New York	509	266400.0	269000.0
10237	97518	94027	Atherton	CA	San Francisco	San Mateo	10238	1179200.0	1184300.0
7596	93816	81611	Aspen	CO	Glenwood Springs	Pitkin	7597	1443100.0	1453700.0
4816	96086	90210	Beverly Hills	CA	Los Angeles-Long Beach-Anaheim	Los Angeles	4817	1015400.0	1016900.0
4723	72636	33480	Palm Beach	FL	Miami-Fort Lauderdale	Palm Beach	4724	958400.0	958100.0
2026	97577	94123	San Francisco	CA	San Francisco	San Francisco	2027	849100.0	859000.0
742	97569	94115	San Francisco	CA	San Francisco	San Francisco	743	794000.0	796500.0
9	97564	94109	San Francisco	CA	San Francisco	San Francisco	10	766000.0	771100.0
12098	93818	81615	Snowmass Village	CO	Glenwood Springs	Pitkin	12099	1458200.0	1441300.0
7380	96149	90402	Santa Monica	CA	Los Angeles-Long Beach-Anaheim	Los Angeles	7381	880000.0	881400.0
8048	96621	92067	Rancho Santa Fe	CA	San Diego	San Diego	8049	991000.0	994700.0
985	96001	90020	Los Angeles	CA	Los Angeles-Long Beach-Anaheim	Los Angeles	986	711600.0	717100.0
10008	97519	94028	Portola Valley	CA	San Francisco	San Mateo	10009	845600.0	853100.0
5738	97691	94301	Palo Alto	CA	San Jose	Santa Clara	5739	709500.0	711900.0
13900	60722	07620	Alpine	NJ	New York	Bergen	13901	1075400.0	1075900.0

Cities and number of zipcodes in narrowed down data

```
#Visualizing the value counts of zipcodes in each city
fig, ax = plt.subplots(figsize=(15,32))
y = [a for a in top_20['City'].value_counts()]
x = [a for a in top_20['City'].value_counts().keys()]
ax.barh(x,y,color='brown')
ax.set_title('Cities with high end properties',fontsize=30)
ax.set_ylabel('Cities',fontsize=20)
ax.set_xlabel('Number of Zipcodes in the City',fontsize=20);
```


Cities with high end properties
Cities
Los Altos
Newport Coast
Alpine
Palo Alto
Portola Valley
Los Angeles
Rancho Santa Fe
Santa Monica
Snowmass Village
Palm Beach
Beverly Hills

Now that we can see how the cities are divided in to zipcodes we will go ahead and narrow down the data to the most promissing cities. We will do this by;

Zipcode Selection

To make informed investment decisions, we will enrich our dataset by calculating a few key metrics that provide insights into the historical performance and variability of housing prices for each zip code. Here is the breakdown of the metrics we will use:

Historical Return on Investment (ROI)

This metric provides a measure of how much the value of a property in a given zip code has appreciated (or depreciated) over the entire span of our dataset. A higher ROI indicates that properties in this zip code have historically appreciated in value at a faster rate.

Standard Deviation (Variability):

The standard deviation reflects the dispersion or variability of property prices around the mean. Higher standard deviations imply greater variability in property prices within a city or zip code. While high variability may entail greater risk, it also presents opportunities for profit, particularly for investors who can identify undervalued properties or capitalize on market fluctuations.

Mean Property Prices

The mean property prices give us an idea of the average price level in each city or zip code. Cities or zip codes with higher mean property prices generally indicate higher demand, desirability, or higher-end real estate markets. These areas are often considered prime locations for real estate investment due to potential higher returns.

Coefficient of Variation (CV)

The coefficient of variation is the ratio of the standard deviation to the mean and represents the relative variability of property prices compared to their mean. A higher CV indicates greater relative variability in property prices. Areas with higher CV values may experience more significant price fluctuations over time, presenting both opportunities and risks for investors.

Based on these insights, the best-performing cities or zip codes for real estate investment are likely to exhibit the following characteristics:

- High mean property prices, indicating strong demand and desirability.
- Moderate to high standard deviations, suggesting potential for profit through market fluctuations or identifying undervalued properties.
- Consideration of the coefficient of variation to understand the relative variability of property prices compared to their mean, balancing risk and potential returns.

```
# Calculate standard deviation (std) for each row
top_20["std"] = top_20.loc[:, "1996-04":"2018-04"].std(skipna=True, axis=1)

# Calculate mean for each row
top_20["mean"] = top_20.loc[:, "1996-04":"2018-04"].mean(skipna=True, axis=1)

# Calculate Coefficient of Variation (CV)
top_20["CV"] = top_20['std'] / top_20["mean"]

# Calculate median for each row
top_20["median"] = top_20.loc[:, "1996-04":"2018-04"].median(skipna=True, axis=1)

# Selecting columns of interest for display
selected_columns = ['City', 'ZipCode', 'mean', 'std', 'CV', 'median']

# Displaying the selected columns
top_20[selected_columns]
```

	City	ZipCode	mean	std	CV	median
272	New York	10021	8.366725e+06	6.686020e+06	0.799120	10196100.0
21	New York	10128	5.085436e+06	1.470204e+06	0.289101	5190800.0
20	New York	10011	4.771058e+06	4.226253e+06	0.885810	4898000.0
508	New York	10014	4.528408e+06	3.560877e+06	0.786342	4957700.0
10237	Atherton	94027	3.487129e+06	1.296224e+06	0.371717	3506200.0
7596	Aspen	81611	3.147124e+06	8.955363e+05	0.284557	3268700.0
4816	Beverly Hills	90210	2.789977e+06	1.222910e+06	0.438323	2598900.0
4723	Palm Beach	33480	2.634498e+06	1.007005e+06	0.382238	2682500.0
2026	San Francisco	94123	2.630977e+06	9.872977e+05	0.375259	2551500.0
742	San Francisco	94115	2.399030e+06	8.852487e+05	0.369003	2402500.0
9	San Francisco	94109	2.395636e+06	8.213880e+05	0.342868	2420400.0
12098	Snowmass Village	81615	2.300179e+06	5.694873e+05	0.247584	2444500.0
7380	Santa Monica	90402	2.292232e+06	8.815972e+05	0.384602	2204200.0
8048	Rancho Santa Fe	92067	2.170122e+06	5.435391e+05	0.250465	2199900.0
985	Los Angeles	90020	2.149644e+06	8.301214e+05	0.386167	2217900.0
10008	Portola Valley	94028	2.131495e+06	6.501195e+05	0.305006	2087700.0
5738	Palo Alto	94301	2.084380e+06	9.745928e+05	0.467570	1793800.0
13900	Alpine	07620	2.080074e+06	6.207531e+05	0.298428	2282900.0
8134	Newport Coast	92657	2.070006e+06	7.511645e+05	0.362880	2129600.0
5843	Los Altos	94022	2.066818e+06	7.992462e+05	0.386704	1890300.0

Now that we have these metrics we are going to narrow down to the top 10 perfoming zipcodes using the median prices

```
# Sort the DataFrame by median property prices in descending order
top_10_zipcodes = top_20.sort_values(by='median', ascending=False)

# Selecting only the top 20 performing zip codes
top_10_zipcodes = top_20.head(10)

# Display the top 20 performing zip codes
top_10_zipcodes
```

	RegionID	ZipCode	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	
272	61635	10021	New York	NY	New York	New York	273	51800.0	52000.0	
21	61703	10128	New York	NY	New York	New York	22	3676700.0	3704200.0	3
20	61625	10011	New York	NY	New York	New York	21	142600.0	143100.0	
508	61628	10014	New York	NY	New York	New York	509	266400.0	269000.0	
10237	97518	94027	Atherton	CA	San Francisco	San Mateo	10238	1179200.0	1184300.0	1
7596	93816	81611	Aspen	CO	Glenwood Springs	Pitkin	7597	1443100.0	1453700.0	1
4816	96086	90210	Beverly Hills	CA	Los Angeles-Long Beach-Anaheim	Los Angeles	4817	1015400.0	1016900.0	1
4723	72636	33480	Palm Beach	FL	Miami-Fort Lauderdale	Palm Beach	4724	958400.0	958100.0	
2026	97577	94123	San Francisco	CA	San Francisco	San Francisco	2027	849100.0	859000.0	
742	97569	94115	San Francisco	CA	San Francisco	San Francisco	743	794000.0	796500.0	

```
#to drop columns we do not need on the analysis
# List the columns you want to drop
columns_to_drop = ['RegionID', 'State', 'Metro','CountyName','SizeRank',]

# Use the drop() method to drop the specified columns
# Specify axis=1 to indicate that you are dropping columns
high_end1 = top_10_zipcodes.drop(columns=columns_to_drop, axis=1)

# After dropping the columns, you can verify the changes by printing the DataFrame
high_end1
```

	ZipCode	City	1996-04	1996-05	1996-06	1996-07	1996-08	1996-09	1996-10
272	10021	New York	51800.0	52000.0	52100.0	52200.0	52300.0	52500.0	52600.0
21	10128	New York	3676700.0	3704200.0	3729600.0	3754600.0	3781800.0	3813500.0	3849600.0
20	10011	New York	142600.0	143100.0	143400.0	143300.0	142900.0	142200.0	141300.0
508	10014	New York	266400.0	269000.0	272000.0	275100.0	278200.0	281100.0	283400.0
10237	94027	Atherton	1179200.0	1184300.0	1189700.0	1195400.0	1201200.0	1207300.0	1214100.0
7596	81611	Aspen	1443100.0	1453700.0	1464300.0	1475000.0	1485800.0	1496500.0	1507400.0
4816	90210	Beverly Hills	1015400.0	1016900.0	1018400.0	1019800.0	1021400.0	1023400.0	1026300.0
4723	33480	Palm Beach	958400.0	958100.0	957900.0	957800.0	958000.0	958400.0	959500.0
2026	94123	San Francisco	849100.0	859000.0	868800.0	878400.0	887800.0	897100.0	906700.0
742	94115	San Francisco	794000.0	796500.0	799000.0	801800.0	804600.0	807700.0	811400.0

Convert the dataset into time series

A copy of the dataset will be created and converted into long view while preserving df as a wide view for EDA. Both the long and wide will be relevant for EDA.

```
# Create a copy of the cleaned DataFrame with a new name
time_data = high_end1.copy()
```

```
import pandas as pd

def melt_df(time_data):
    melted = pd.melt(time_data, id_vars=['ZipCode','City',
                                  'CV' ,'std','mean','median','AverageHousePrice'], var_name='Date',va
    melted['Date'] = pd.to_datetime(melted['Date'], infer_datetime_format=True)
    melted.set_index('Date', inplace=True)
    melted.rename(columns={"median": "median"}, inplace=True)
    melted = melted.dropna(subset=['median'])
    return melted
```

```
# Apply the melt_df function to convert the DataFrame from wide to long format
melted_df = melt_df(time_data)

# Display the final cleaned data
melted_df
```

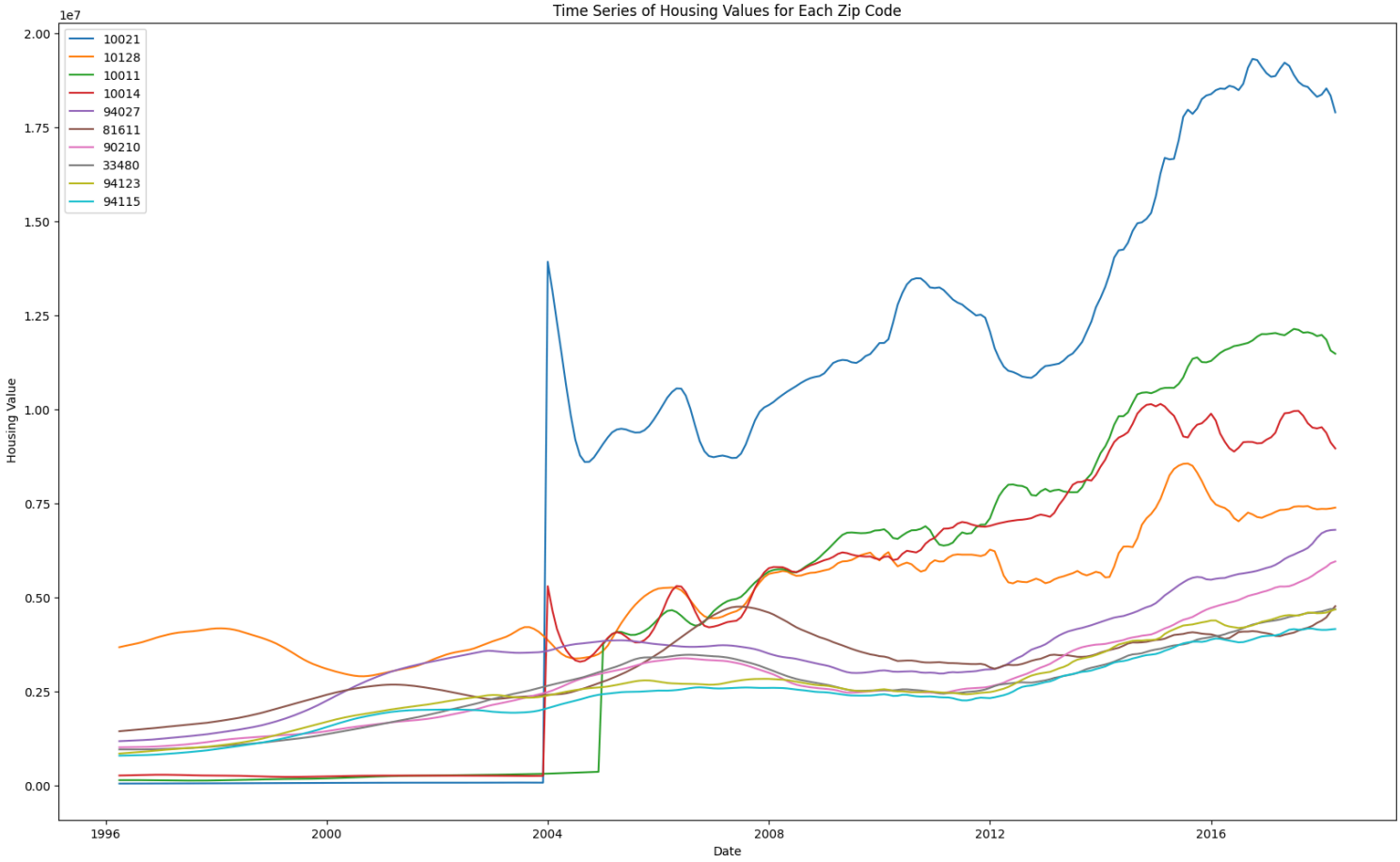
Date	ZipCode	City	CV	std	mean	median	AverageHousePrice	H pr
1996-04-01	10021	New York	0.799120	6.686020e+06	8.366725e+06	10196100.0	8.304285e+06	518
1996-04-01	10128	New York	0.289101	1.470204e+06	5.085436e+06	5190800.0	5.047574e+06	36767
1996-04-01	10011	New York	0.885810	4.226253e+06	4.771058e+06	4898000.0	4.735551e+06	1426
1996-04-01	10014	New York	0.786342	3.560877e+06	4.528408e+06	4957700.0	4.494720e+06	2664
1996-04-01	94027	Atherton	0.371717	1.296224e+06	3.487129e+06	3506200.0	3.461412e+06	11792
...	...	...	...	...	...	...	...	
2018-04-01	81611	Aspen	0.284557	8.955363e+05	3.147124e+06	3268700.0	3.123930e+06	47666
2018-	90210	Beverly	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06	59567

```
print(melted_df.columns)
```

```
Index(['ZipCode', 'City', 'CV', 'std', 'mean', 'median', 'AverageHousePrice',
       'House prices'],
      dtype='object')
```

A visualization of the top 10 zipcodes

```
import pandas as pd
import matplotlib.pyplot as plt

# creating a function that changes the dataframe structure from wide view to long view
#def melt_df(top_20_zipcodes):
   # melted = pd.melt(top_20_zipcodes, id_vars=['RegionID', 'ZipCode', 'City', 'State', 'Metro', 'Cour
   # melted['Date'] = pd.to_datetime(melted['Date'], infer_datetime_format=True)
   # melted.set_index('Date', inplace=True)
   # melted = melted.dropna(subset=['house price'])
   # return melted

# Convert the DataFrame from wide to long format
#melted_data = melt_df(top_20_zipcodes)

# Now you can iterate over each zip code and plot the data
fig, ax = plt.subplots(figsize=(20, 12))
for zipcode in melted_df['ZipCode'].unique():
    ax.plot(melted_df.loc[melted_df['ZipCode'] == zipcode].index, melted_df.loc[melted_df['ZipCode'] =

ax.set_xlabel('Date')
ax.set_ylabel(' Housing Value')
ax.set_title('Time Series of Housing Values for Each Zip Code')
ax.legend()
plt.show()
```



Observations from Time Series Plot of Median Housing Values

- From 1996 to 2008, a general uptrend is noticeable across most of the zip codes except for 3 zip codes whose prices remained constant. Subsequently, between 2008 and approximately 2012, there is a decline in prices in 6 of the chosen zipcodes. This period coincides with the aftermath of the 2008 economic recession, which resulted in decreased property values and a rise in mortgage defaults.

- Beginning around 2012, there is a stabilization in house prices, followed by a gradual and steady increase. By excluding the period impacted by the economic recession, we aim to enhance the accuracy of future predictions regarding property values.
- The plotted data illustrates the historical trends in property values for each zip code, showcasing various patterns. While some zip codes demonstrate consistent appreciation over time, indicating a stable housing market, others exhibit more volatility or distinct growth phases, suggesting fluctuations in property values.

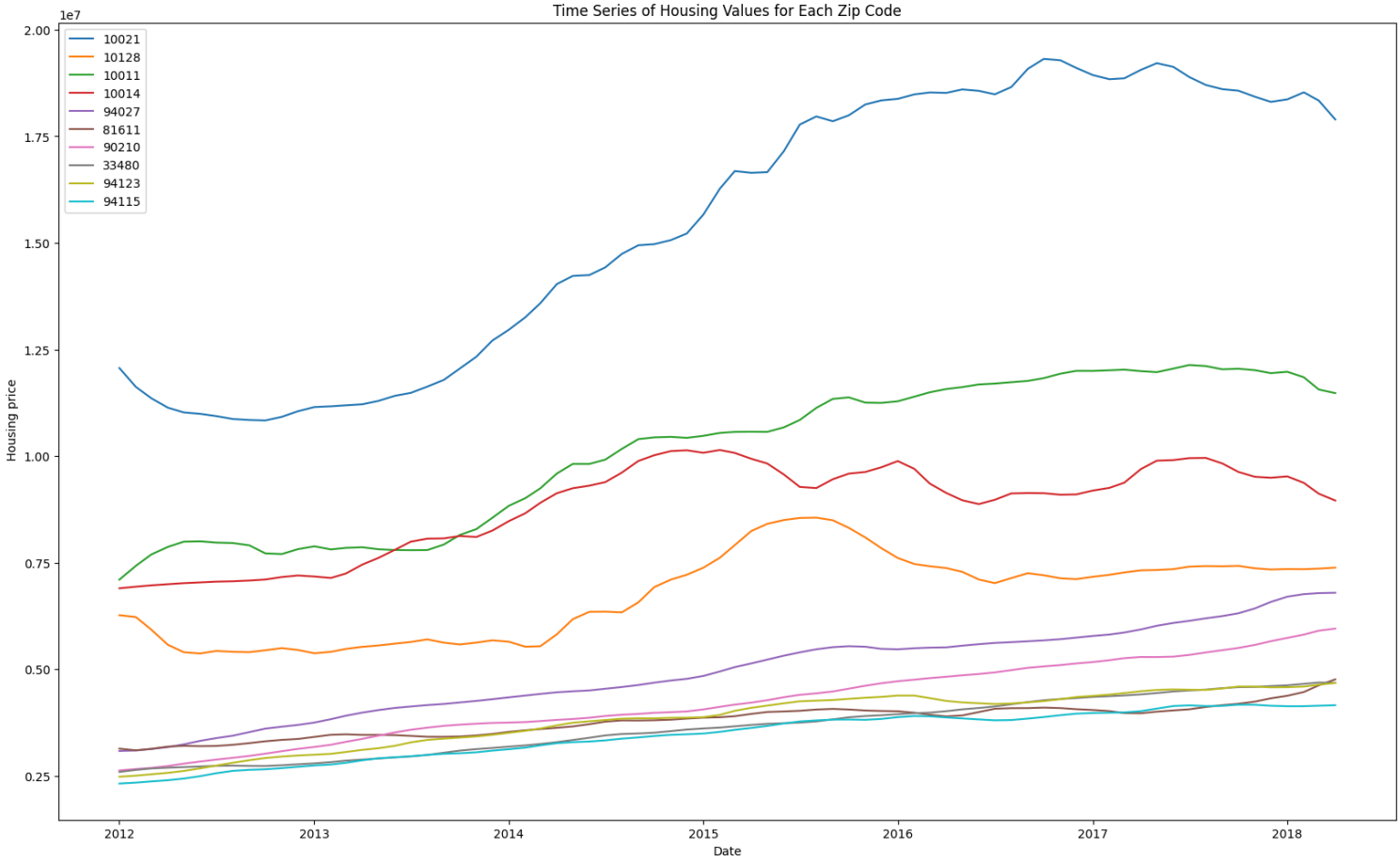
Shortening our Time Series

Since the recession is affecting most of the chosen zip codes, something that is not a frequent occurrence, we will focus on data from 2012.

```
# Define the start and end dates
start_date = '1996-04'
end_date = '2011-12'

# Find the index positions of the start and end dates
start_idx = high_end1.columns.get_loc(start_date)
end_idx = high_end1.columns.get_loc(end_date)

# Drop columns within the specified date range
filtered_df = high_end1.drop(high_end1.columns[start_idx:end_idx+1], axis=1)
filtered_df
```

	ZipCode	City	2012-01	2012-02	2012-03	2012-04	2012-05	2012-06	2
272	10021	New York	12067200.0	11624000.0	11357900.0	11136900.0	11025200.0	10991600.0	109
21	10128	New York	6271000.0	6227500.0	5931700.0	5578000.0	5401900.0	5373300.0	54
20	10011	New York	7103800.0	7428300.0	7694600.0	7872000.0	7994800.0	8003300.0	79
508	10014	New York	6902000.0	6939400.0	6970700.0	6997500.0	7022300.0	7039800.0	70
10237	94027	Atherton	3086600.0	3097900.0	3132700.0	3177000.0	3242200.0	3325100.0	33
7596	81611	Aspen	3141700.0	3102900.0	3137700.0	3189600.0	3207900.0	3200400.0	32
4816	90210	Beverly Hills	2632200.0	2663800.0	2691700.0	2734000.0	2790000.0	2840300.0	28
4723	33480	Palm Beach	2595600.0	2643200.0	2677800.0	2695200.0	2708800.0	2723800.0	27
2026	94123	San Francisco	2482900.0	2508000.0	2540500.0	2574100.0	2619500.0	2680000.0	27
742	94115	San Francisco	2323000.0	2345200.0	2373300.0	2400800.0	2440100.0	2496800.0	25

```
import pandas as pd
import matplotlib.pyplot as plt

# creating a function that changes the dataframe structure from wide view to long view
def melt_df(filtered_df):
    melted = pd.melt(filtered_df, id_vars=['ZipCode', 'City','std', 'mean', 'CV','median','AverageHou
    melted['Date'] = pd.to_datetime(melted['Date'], infer_datetime_format=True)
    melted.set_index('Date', inplace=True)
    melted = melted.dropna(subset=['house price'])
    return melted

# Convert the DataFrame from wide to long format
melted2_data = melt_df(filtered_df)

# Now you can iterate over each zip code and plot the data
fig, ax = plt.subplots(figsize=(20, 12))
for zipcode in melted2_data['ZipCode'].unique():
    ax.plot(melted2_data.loc[melted2_data['ZipCode'] == zipcode].index, melted2_data.loc[melted2_data

ax.set_xlabel('Date')
ax.set_ylabel(' Housing price')
ax.set_title('Time Series of Housing Values for Each Zip Code')
ax.legend()
plt.show()
```



Identifying Top Investment Zipcodes

To identify top investment Zipcodes, we will follow these steps:

1. **Examine Coefficient of Variance (CV)**: We will again analyze the CV for each zip code to understand the volatility or variability of housing prices. A higher CV indicates greater volatility.

2. **Set Upper Limit for CV**: Using the 60th percentile of CV values as a threshold, we will set an upper limit for acceptable risk. This helps filter out zip codes with higher-than-acceptable risk levels.
3. **Identify Promising Zip Codes**: We will then identify top 5 zip codes that offer the best historical Return on Investment (ROI) while also fitting within the defined risk profile. These zip codes represent top investment opportunities that combine favorable ROI with acceptable risk levels.

```
#Since we already calculated the other metrics we will now calculate the ROI
# Define a function to calculate ROI
def calculate_roi(row):
    initial_value = row['2012-01']
    final_value = row['2018-04']
    roi = round(((final_value - initial_value) / initial_value) * 100, 2)
    return roi

# Apply the function to each row of the DataFrame
filtered_df['ROI'] = filtered_df.apply(calculate_roi, axis=1)

# Display the DataFrame with ROI column
print(filtered_df[['ZipCode','std','mean','CV','median','ROI']])

filtered_df
```

	ZipCode	std	mean	CV	median	ROI
272	10021	6.686020e+06	8.366725e+06	0.799120	10196100.0	48.29
21	10128	1.470204e+06	5.085436e+06	0.289101	5190800.0	17.79
20	10011	4.226253e+06	4.771058e+06	0.885810	4898000.0	61.58
508	10014	3.560877e+06	4.528408e+06	0.786342	4957700.0	29.81
10237	94027	1.296224e+06	3.487129e+06	0.371717	3506200.0	120.19
7596	81611	8.955363e+05	3.147124e+06	0.284557	3268700.0	51.72
4816	90210	1.222910e+06	2.789977e+06	0.438323	2598900.0	126.30
4723	33480	1.007005e+06	2.634498e+06	0.382238	2682500.0	80.54
2026	94123	9.872977e+05	2.630977e+06	0.375259	2551500.0	88.58
742	94115	8.852487e+05	2.399030e+06	0.369003	2402500.0	79.10

	ZipCode	City	2012-01	2012-02	2012-03	2012-04	2012-05	2012-06	2
272	10021	New York	12067200.0	11624000.0	11357900.0	11136900.0	11025200.0	10991600.0	109
21	10128	New York	6271000.0	6227500.0	5931700.0	5578000.0	5401900.0	5373300.0	54
20	10011	New York	7103800.0	7428300.0	7694600.0	7872000.0	7994800.0	8003300.0	79
508	10014	New York	6902000.0	6939400.0	6970700.0	6997500.0	7022300.0	7039800.0	70
10237	94027	Atherton	3086600.0	3097900.0	3132700.0	3177000.0	3242200.0	3325100.0	33
7596	81611	Aspen	3141700.0	3102900.0	3137700.0	3189600.0	3207900.0	3200400.0	32
4816	90210	Beverly Hills	2632200.0	2663800.0	2691700.0	2734000.0	2790000.0	2840300.0	28
4723	33480	Palm Beach	2595600.0	2643200.0	2677800.0	2695200.0	2708800.0	2723800.0	27
2026	94123	San Francisco	2482900.0	2508000.0	2540500.0	2574100.0	2619500.0	2680000.0	27
742	94115	San Francisco	2323000.0	2345200.0	2373300.0	2400800.0	2440100.0	2496800.0	25

```
#Sorting dataframe to check top zipcodes
sorted_data = filtered_df.sort_values(by='ROI', ascending=False)

sorted_data
```

	ZipCode	City	2012-01	2012-02	2012-03	2012-04	2012-05	2012-06	2
4816	90210	Beverly Hills	2632200.0	2663800.0	2691700.0	2734000.0	2790000.0	2840300.0	28
10237	94027	Atherton	3086600.0	3097900.0	3132700.0	3177000.0	3242200.0	3325100.0	33
2026	94123	San Francisco	2482900.0	2508000.0	2540500.0	2574100.0	2619500.0	2680000.0	27
4723	33480	Palm Beach	2595600.0	2643200.0	2677800.0	2695200.0	2708800.0	2723800.0	27
742	94115	San Francisco	2323000.0	2345200.0	2373300.0	2400800.0	2440100.0	2496800.0	25
20	10011	New York	7103800.0	7428300.0	7694600.0	7872000.0	7994800.0	8003300.0	79
7596	81611	Aspen	3141700.0	3102900.0	3137700.0	3189600.0	3207900.0	3200400.0	32
272	10021	New York	12067200.0	11624000.0	11357900.0	11136900.0	11025200.0	10991600.0	109
508	10014	New York	6902000.0	6939400.0	6970700.0	6997500.0	7022300.0	7039800.0	70
21	10128	New York	6271000.0	6227500.0	5931700.0	5578000.0	5401900.0	5373300.0	54

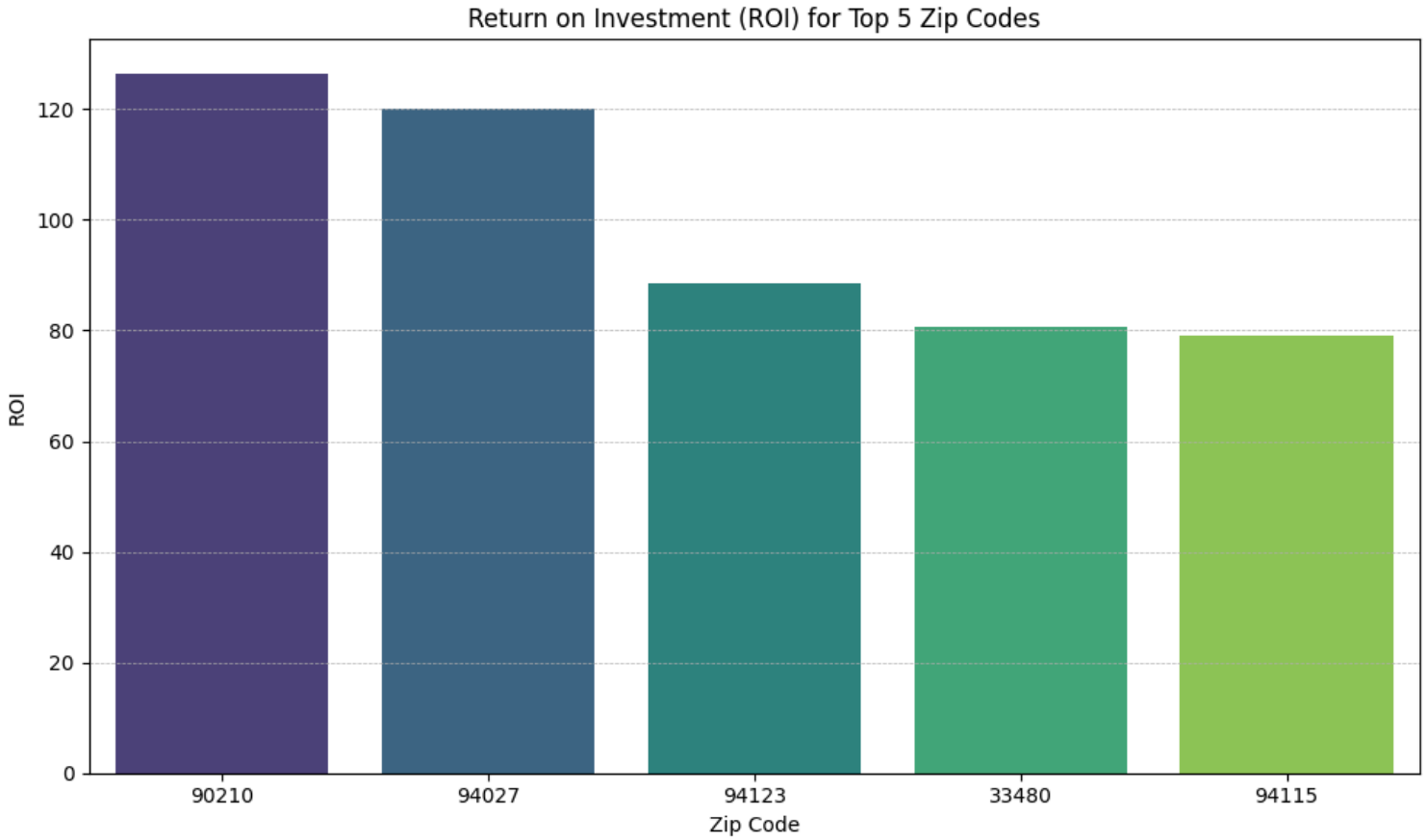
```
# Sort the DataFrame by ROI in descending order
top_5_zipcodes = sorted_data.sort_values(by='ROI', ascending=False).head(5)

# Display the top 5 zip codes with the highest ROI
top_5_zipcodes
```

	ZipCode	City	2012-01	2012-02	2012-03	2012-04	2012-05	2012-06	2012-07
4816	90210	Beverly Hills	2632200.0	2663800.0	2691700.0	2734000.0	2790000.0	2840300.0	2884600.0
10237	94027	Atherton	3086600.0	3097900.0	3132700.0	3177000.0	3242200.0	3325100.0	3389500.0
2026	94123	San Francisco	2482900.0	2508000.0	2540500.0	2574100.0	2619500.0	2680000.0	2745300.0
4723	33480	Palm Beach	2595600.0	2643200.0	2677800.0	2695200.0	2708800.0	2723800.0	2738700.0
742	94115	San Francisco	2323000.0	2345200.0	2373300.0	2400800.0	2440100.0	2496800.0	2565600.0

```
import pandas as pd

def melt_df(top_5_zipcodes):
    melted = pd.melt(top_5_zipcodes, id_vars=['ZipCode', 'City','std', 'mean', 'CV', 'median','ROI','A
    melted['Date'] = pd.to_datetime(melted['Date'], infer_datetime_format=True)

    # Set 'Date' as the index
    melted.set_index('Date', inplace=True)

    # Drop NaN values in the 'median' column
    melted = melted.dropna(subset=['ROI'])

    # Reset the index to make 'Date' visible
    melted.reset_index(inplace=True)

    return melted



# Apply the melt_df function to convert the DataFrame from wide to long format
melted3_df = melt_df(top_5_zipcodes)

# Display the final cleaned data
melted3_df
```

	Date	ZipCode	City	std	mean	CV	median	ROI	AverageHousel
0	2012-01-01	90210	Beverly Hills	1.222910e+06	2.789977e+06	0.438323	2598900.0	126.30	2.76945
1	2012-01-01	94027	Atherton	1.296224e+06	3.487129e+06	0.371717	3506200.0	120.19	3.46141
2	2012-01-01	94123	San Francisco	9.872977e+05	2.630977e+06	0.375259	2551500.0	88.58	2.61164
3	2012-01-01	33480	Palm Beach	1.007005e+06	2.634498e+06	0.382238	2682500.0	80.54	2.61505
4	2012-01-01	94115	San Francisco	8.852487e+05	2.399030e+06	0.369003	2402500.0	79.10	2.38142
...	...	...	...	...	...	...	...	...	
375	2018-04-01	90210	Beverly Hills	1.222910e+06	2.789977e+06	0.438323	2598900.0	126.30	2.76945
376	2018-	94027	Atherton	1.296224e+06	3.487129e+06	0.371717	3506200.0	120.19	3.46141

```
# Bar Plot to compare ROI for top 5 zip codes
plt.figure(figsize=(10, 6))
sns.barplot(data=melted3_df, x='ZipCode', y='ROI', palette='viridis')
plt.title('Return on Investment (ROI) for Top 5 Zip Codes')
plt.xlabel('Zip Code')
plt.ylabel('ROI')
plt.grid(True, which='both', axis='y', linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

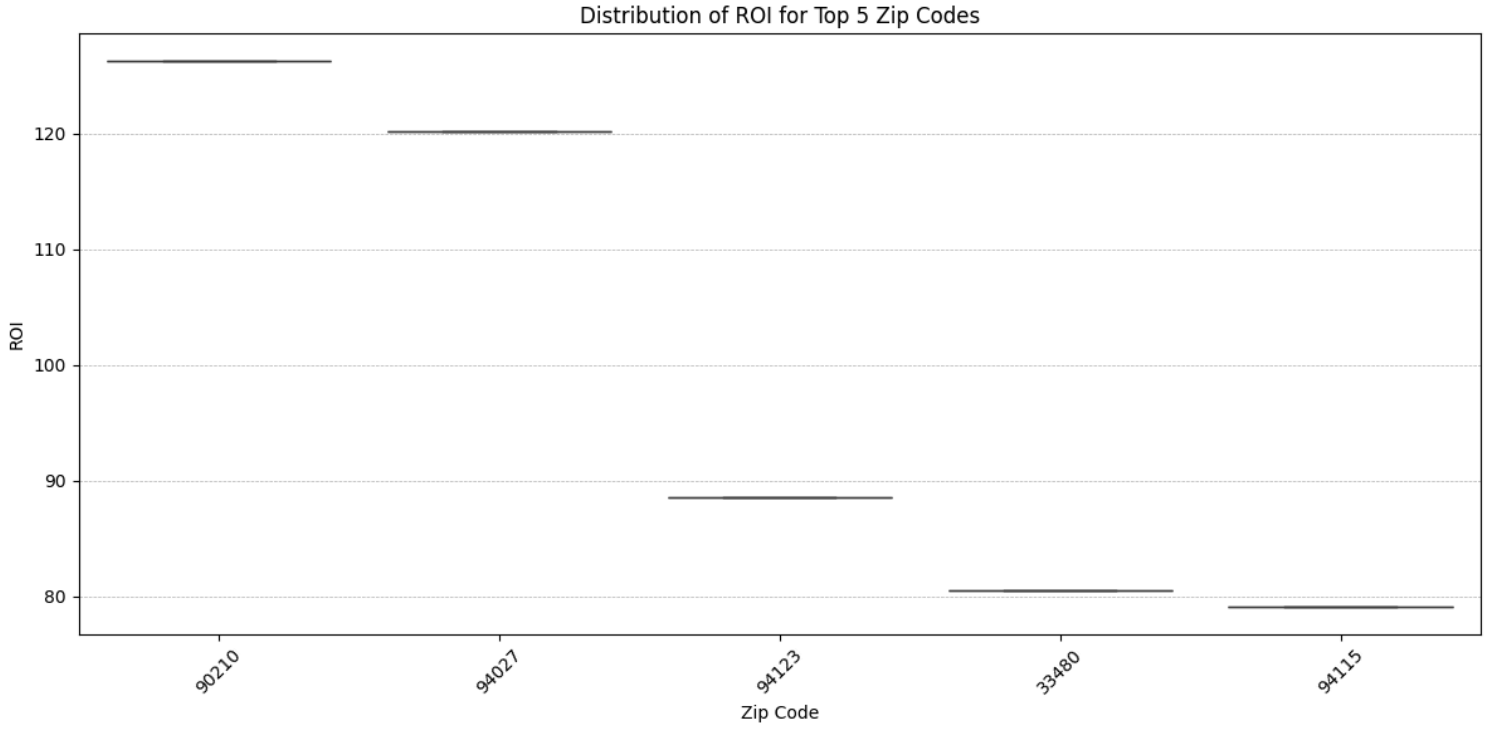



```
# Descriptive statistics of coefficients of variance.
print(top_5_zipcodes.CV.describe())

# Define upper limit of CV according to risk profile.
upper_cv = top_5_zipcodes.CV.quantile(.6)
print(f'\nCV upper limit: {upper_cv}')
```

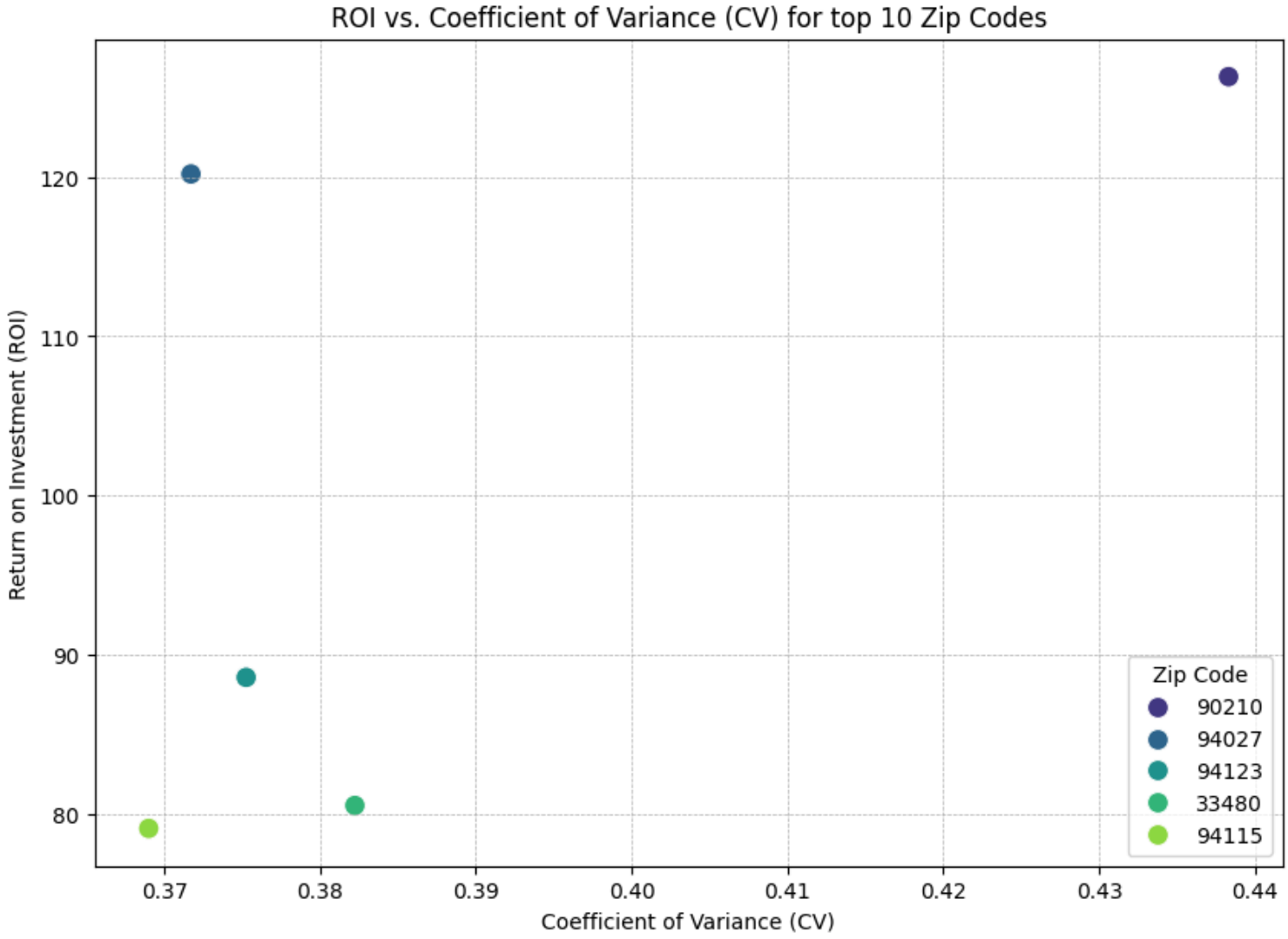
```
count    5.000000
mean     0.387308
std      0.028946
min      0.369003
25%      0.371717
50%      0.375259
```

```
75%      0.382238
max      0.438323
Name: CV, dtype: float64

CV upper limit: 0.3780504800886181
```

```
# Box Plot for distribution of ROI for top 5 zip codes
palette = sns.color_palette("tab10", 20)
plt.figure(figsize=(12, 6))
sns.boxplot(data=top_5_zipcodes, x='ZipCode', y='ROI', palette=palette)
plt.title('Distribution of ROI for Top 5 Zip Codes')
plt.xlabel('Zip Code')
plt.ylabel('ROI')
plt.grid(True, which='both', axis='y', linestyle='--', linewidth=0.5)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



```
# Scatter Plot to show relationship between ROI and CV for top 10 zip codes
plt.figure(figsize=(10, 7))
sns.scatterplot(data=top_5_zipcodes, x='CV', y='ROI', hue='ZipCode', s=100, palette='viridis', legend=
plt.title('ROI vs. Coefficient of Variance (CV) for top 10 Zip Codes')
plt.xlabel('Coefficient of Variance (CV)')
plt.ylabel('Return on Investment (ROI)')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.legend(title='Zip Code')
plt.show()
```



The scatter plot provides insights into the trade-off between risk (volatility in property values) and return (historical appreciation). Zip codes closer to the top-left corner offer higher ROI with lower risk, making them more desirable for investment.

```
# creating a function that changes the dataframe structure from wide view to long view

def melt_df(top_5_zipcodes):
    melted = pd.melt(top_5_zipcodes, id_vars=['ZipCode', 'City',
                                'ROI', 'CV' ,'std','mean','median','AverageHousePrice'], var_name='D
    melted['Date'] = pd.to_datetime(melted['Date'], infer_datetime_format=True)
    melted.set_index('Date', inplace=True)
    melted.rename(columns={"value": "houseprice"}, inplace=True)
    melted = melted.dropna(subset=['houseprice'])
    return melted


# Create a copy of the cleaned DataFrame with a new name (e.g., new_df)
new_df = top_5_zipcodes.copy()

# Apply the melt_df function to convert the DataFrame from wide to long format
melted4_df = melt_df(new_df)

# Display the final cleaned data
melted4_df
```

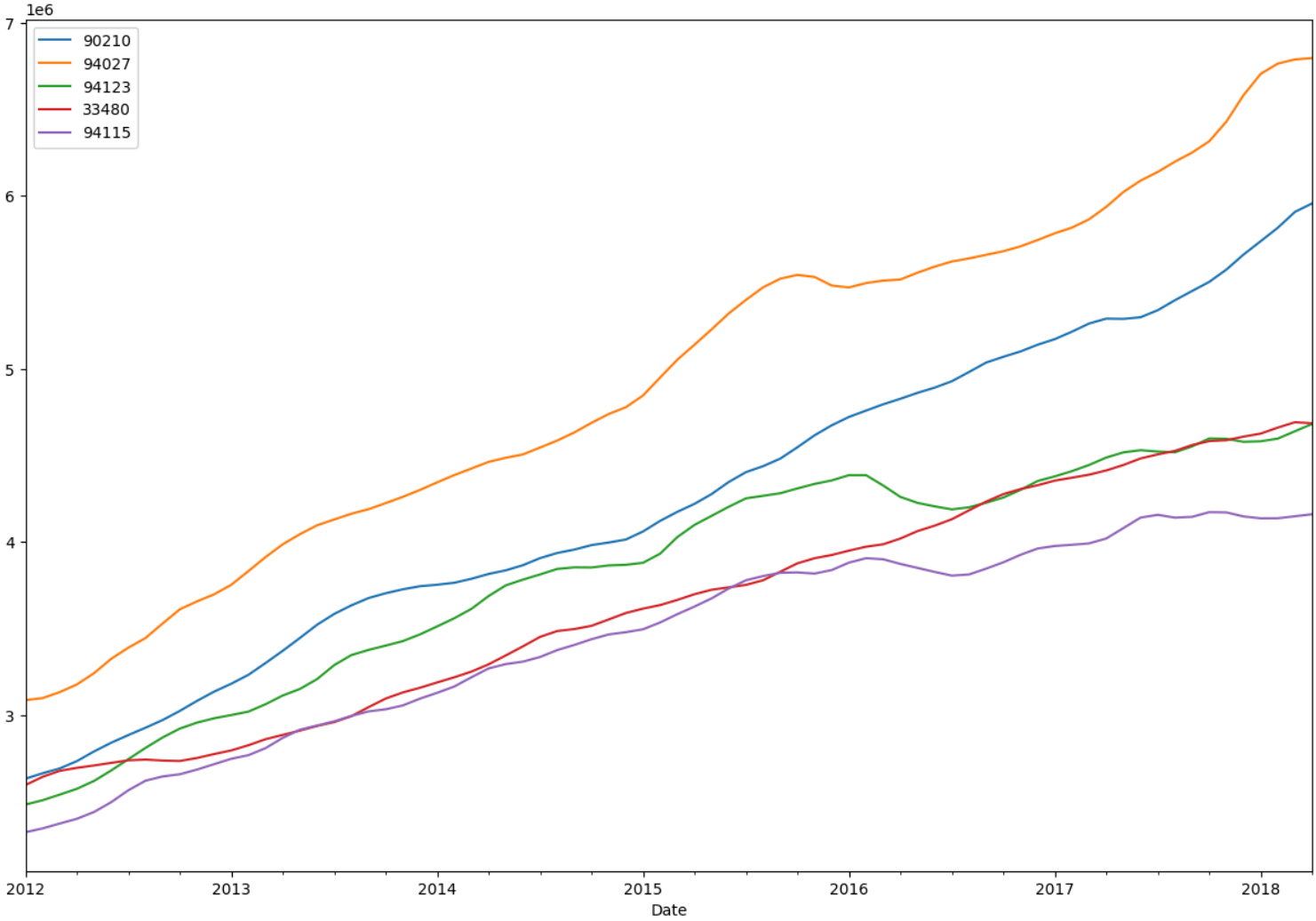
Date	ZipCode	City	ROI	CV	std	mean	median	AverageHousePrice
2012-01-01	90210	Beverly Hills	126.30	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06
2012-01-01	94027	Atherton	120.19	0.371717	1.296224e+06	3.487129e+06	3506200.0	3.461412e+06
2012-01-01	94123	San Francisco	88.58	0.375259	9.872977e+05	2.630977e+06	2551500.0	2.611643e+06
2012-01-01	33480	Palm Beach	80.54	0.382238	1.007005e+06	2.634498e+06	2682500.0	2.615054e+06
2012-01-01	94115	San Francisco	79.10	0.369003	8.852487e+05	2.399030e+06	2402500.0	2.381428e+06
...	...	...	...	...	...	...	...	...
2018-04-01	90210	Beverly Hills	126.30	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06
2018-04-01	94027	Atherton	120.19	0.371717	1.296224e+06	3.487129e+06	3506200.0	3.461412e+06

```
TS_top5 = melted4_df

print('Time series data for the 5 zip codes:\n', TS_top5.head())

# Create individualized time series for each zipcode
dfs_ts = []

for zc in TS_top5['ZipCode'].unique():
    # Create separate dataframes for each zipcode with a monthly frequency
    df = TS_top5[TS_top5['ZipCode'] == zc].asfreq('MS')
    dfs_ts.append(df)

# Print the time series data for the first zipcode in the list
print(f'\nZipcode {TS_top5["ZipCode"].unique()[0]} time series:')
dfs_ts[0].head(5)
```

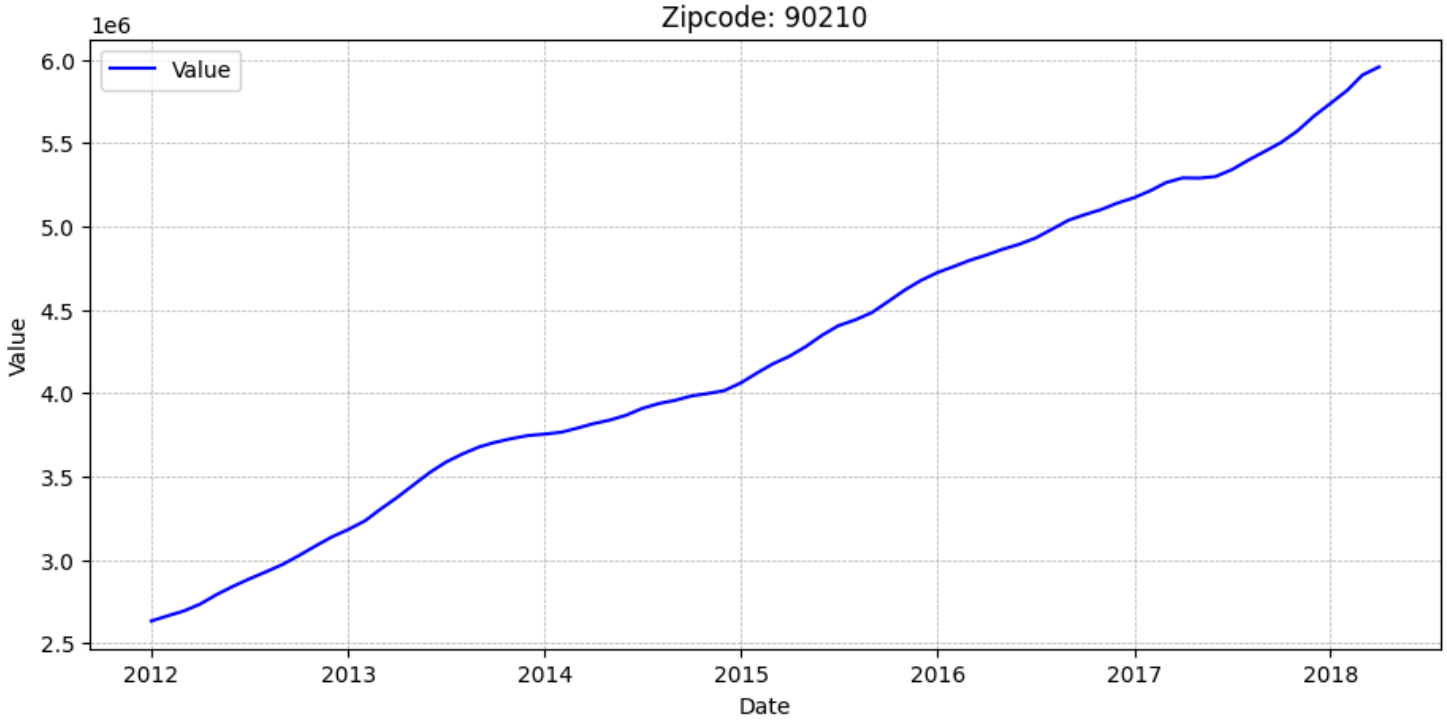
```
Time series data for the 5 zip codes:
            ZipCode           City     ROI        CV           std  \
Date
2012-01-01   90210  Beverly Hills  126.30  0.438323  1.222910e+06
2012-01-01   94027       Atherton  120.19  0.371717  1.296224e+06
2012-01-01   94123  San Francisco   88.58  0.375259  9.872977e+05
2012-01-01   33480     Palm Beach   80.54  0.382238  1.007005e+06
2012-01-01   94115  San Francisco   79.10  0.369003  8.852487e+05

                  mean     median  AverageHousePrice  houseprice
Date
2012-01-01  2.789977e+06  2598900.0       2.769457e+06   2632200.0
2012-01-01  3.487129e+06  3506200.0       3.461412e+06   3086600.0
2012-01-01  2.630977e+06  2551500.0       2.611643e+06   2482900.0
2012-01-01  2.634498e+06  2682500.0       2.615054e+06   2595600.0
2012-01-01  2.399030e+06  2402500.0       2.381428e+06   2323000.0

Zipcode 90210 time series:
```

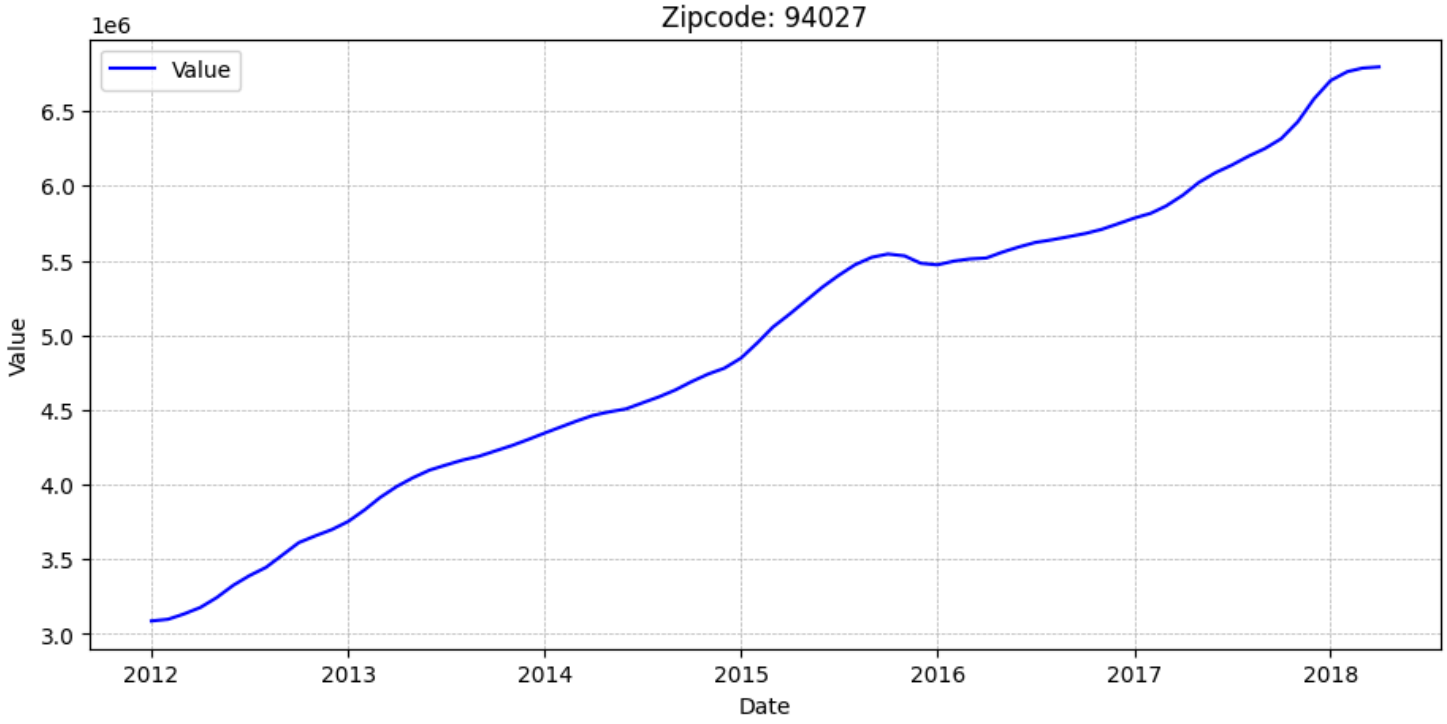
Date	ZipCode	City	ROI	CV	std	mean	median	AverageHousePrice	h
2012-01-01	90210	Beverly Hills	126.3	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06	
2012-02-01	90210	Beverly Hills	126.3	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06	
2012-03-01	90210	Beverly Hills	126.3	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06	
2012-		Beverly							

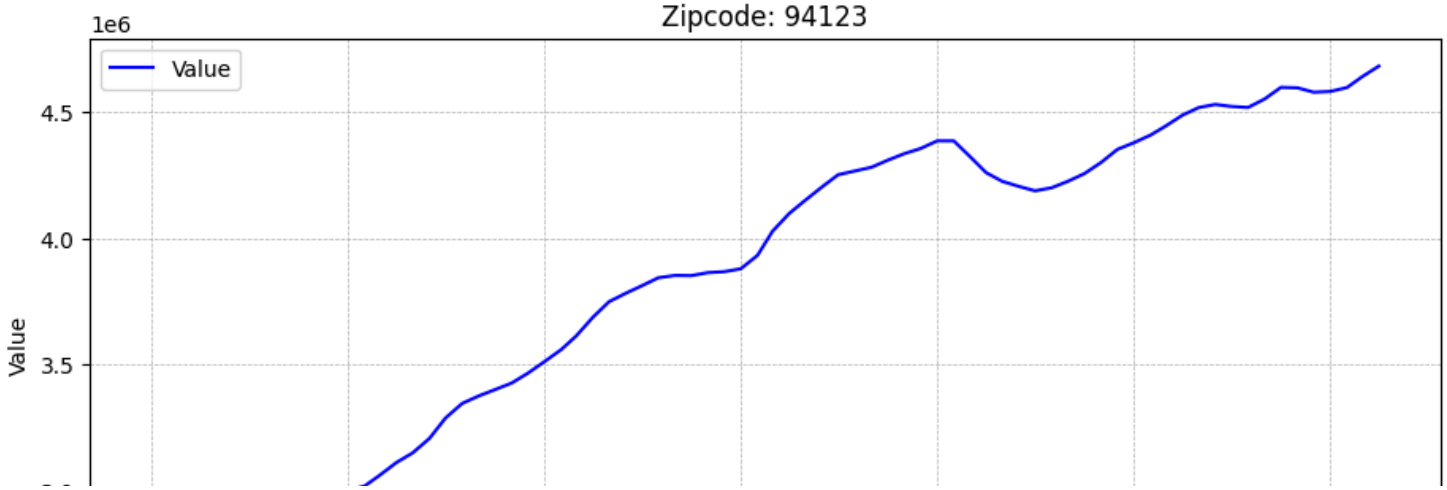
```
for i in range(5):
    dfs_ts[i]['houseprice'].plot(label=dfs_ts[i]['ZipCode'].iloc[0], figsize=(15, 10))
    plt.legend()
```


1e6
7
6
5
4
3
90210
94027
94123
33480
94115
2012
2013
2014
2015
2016
2017
2018
Date

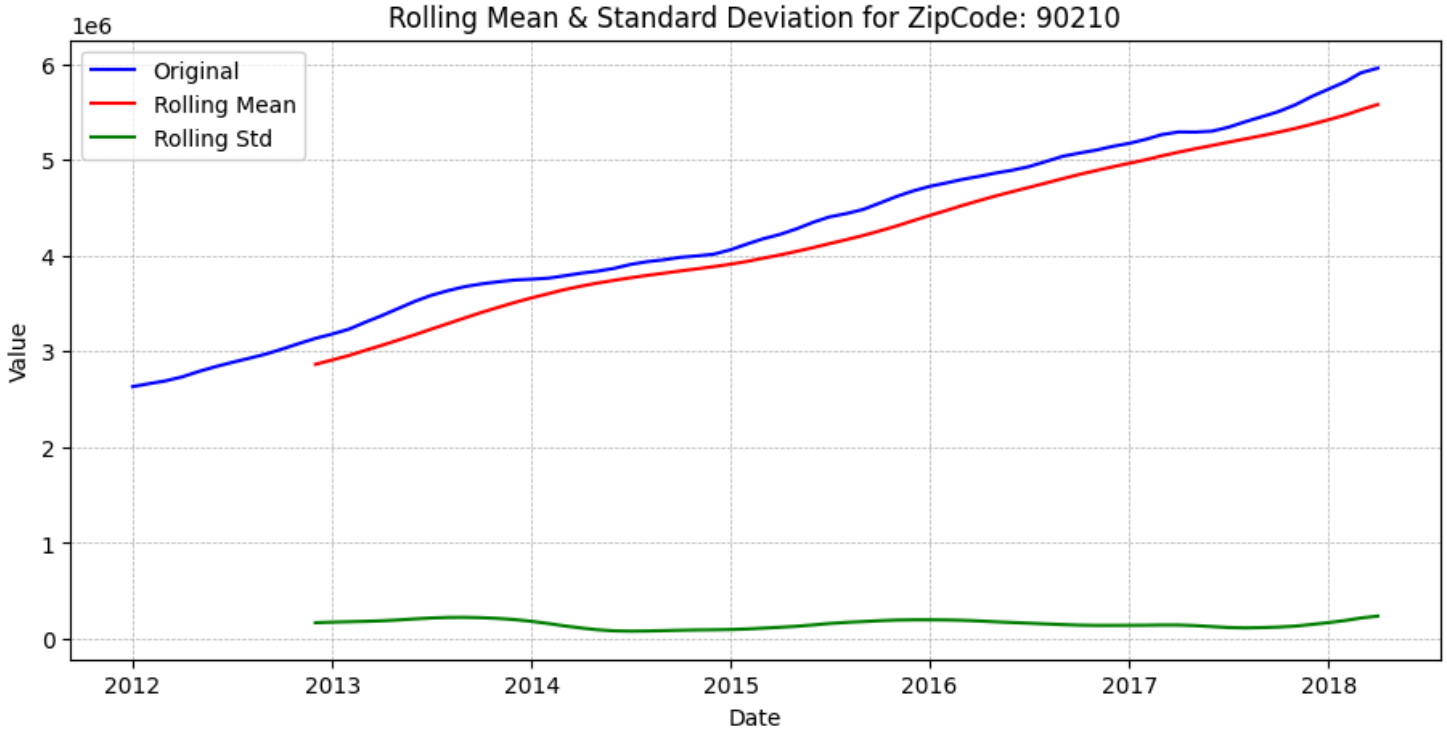
```
# Visualizing the housing prices per zipcode
for zc in range(len(dfs_ts)):
    dfs_ts[zc]['ret'] = dfs_ts[zc]['houseprice']

# Plotting the monthly returns for each of the top 10 zip codes
for i in range(len(dfs_ts)):
    plt.figure(figsize=(11, 5))
    plt.plot(dfs_ts[i].index, dfs_ts[i]['houseprice'], color='b')
    plt.title(f'Zipcode: {dfs_ts[i]["ZipCode"].iloc[0]}')
    plt.xlabel('Date')
    plt.ylabel('Value')
    plt.legend(['Value'], loc='best')
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)
    plt.show()
```

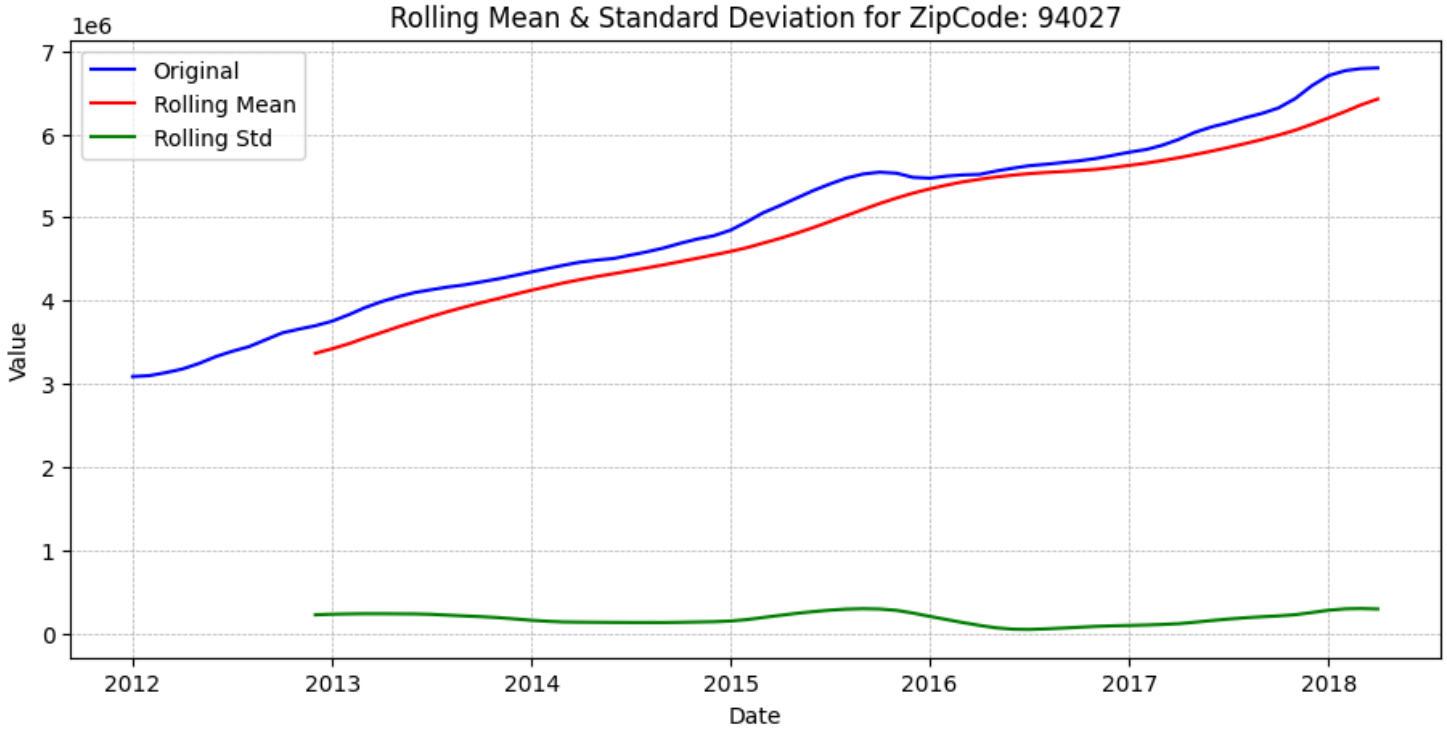

Zipcode: 90210
1e6
Value
6.0
5.5
5.0
4.5
4.0
3.5
3.0
2.5
Value
2012
2013
2014
2015
2016
2017
2018
Date

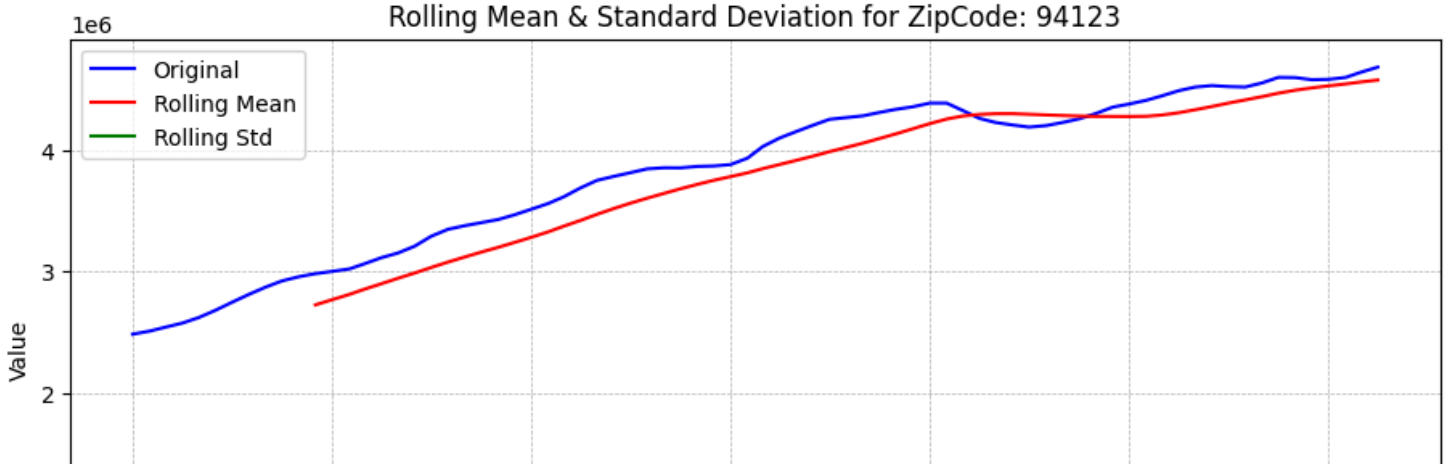

Zipcode: 94027
1e6
Value
6.5
6.0
5.5
5.0
4.5
4.0
3.5
3.0
Value
2012
2013
2014
2015
2016
2017
2018
Date


Zipcode: 94123
1e6
Value
4.5
4.0
3.5
Value

```
import matplotlib.pyplot as plt

# Plotting monthly returns with their respective rolling mean and rolling std for each of the top 10 z
for i in range(len(dfs_ts)):
    # Calculate rolling mean and rolling standard deviation
    rolmean = dfs_ts[i]['houseprice'].rolling(window=12, center=False).mean()
    rolstd = dfs_ts[i]['houseprice'].rolling(window=12, center=False).std()

    # Plotting the metrics
    fig = plt.figure(figsize=(11,5))

    # Original values in blue
    orig = plt.plot(dfs_ts[i]['houseprice'], color='blue', label='Original')

    # Rolling mean in red
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')

    # Rolling standard deviation in green
    std = plt.plot(rolstd, color='green', label='Rolling Std')

    # Add legends, title, labels, and gridlines
    plt.legend(loc='best')
    plt.title(f'Rolling Mean & Standard Deviation for ZipCode: {dfs_ts[i].ZipCode.iloc[0]}')
    plt.xlabel('Date')
    plt.ylabel('Value')
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)
    plt.show()
```


Rolling Mean & Standard Deviation for ZipCode: 90210
1e6
Original
Rolling Mean
Rolling Std
Value
Date
0
1
2
3
4
5
6
2012
2013
2014
2015
2016
2017
2018


Rolling Mean & Standard Deviation for ZipCode: 94027
1e6
Original
Rolling Mean
Rolling Std
Value
Date
0
1
2
3
4
5
6
7
2012
2013
2014
2015
2016
2017
2018


Rolling Mean & Standard Deviation for ZipCode: 94123
1e6
Original
Rolling Mean
Rolling Std
Value
2
3
4

Explanation:

Upon initial examination, the data appears to demonstrate a consistent pattern. To validate the degree of this consistency, we intend to conduct the Augmented Dickey-Fuller test. This statistical assessment will offer deeper insights into whether the data exhibits stationarity or not. Essentially, the test establishes a hypothesis: the null hypothesis assumes the presence of an order of integration within the data, suggesting it is non-stationary. Conversely, the alternative hypothesis implies the absence of such integration, indicating that the data is stationary.

By utilizing a confidence level of 95%, my criterion for rejecting the null hypothesis relies on obtaining a p-value below 0.05. This particular selection ensures a robust determination of the data's stationarity

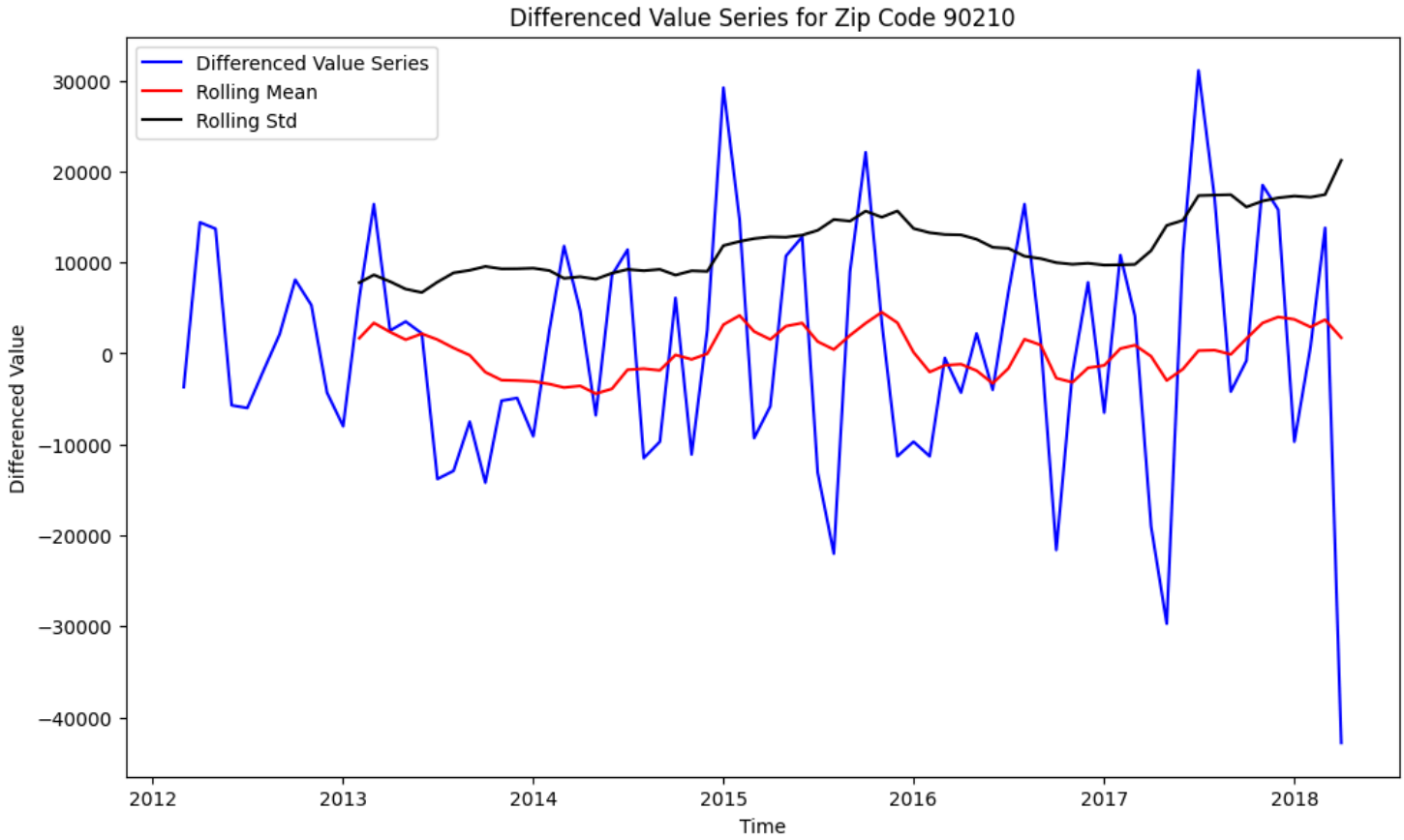
```
from statsmodels.tsa.stattools import adfuller

# Check for stationarity
for i in range(5):
    results = adfuller(dfs_ts[i].ret.dropna())
    print(f'ADFuller test p-value for zipcode: {dfs_ts[i].ZipCode[0]}')
    print('p-value:',results[1])
    if results[1]>0.05:
        print('Fail to reject the null hypothesis. Data is not stationary.\n')
    else:
        print('Reject the null hypothesis. Data is stationary.\n')
```

```
 ADFuller test p-value for zipcode: 90210
 p-value: 0.9457491257946723
 Fail to reject the null hypothesis. Data is not stationary.

 ADFuller test p-value for zipcode: 94027
 p-value: 0.893089707017959
 Fail to reject the null hypothesis. Data is not stationary.

 ADFuller test p-value for zipcode: 94123
 p-value: 0.25919917147840443
 Fail to reject the null hypothesis. Data is not stationary.

 ADFuller test p-value for zipcode: 33480
 p-value: 0.494820589996151
 Fail to reject the null hypothesis. Data is not stationary.

 ADFuller test p-value for zipcode: 94115
 p-value: 0.020768234512713285
 Reject the null hypothesis. Data is stationary.
```

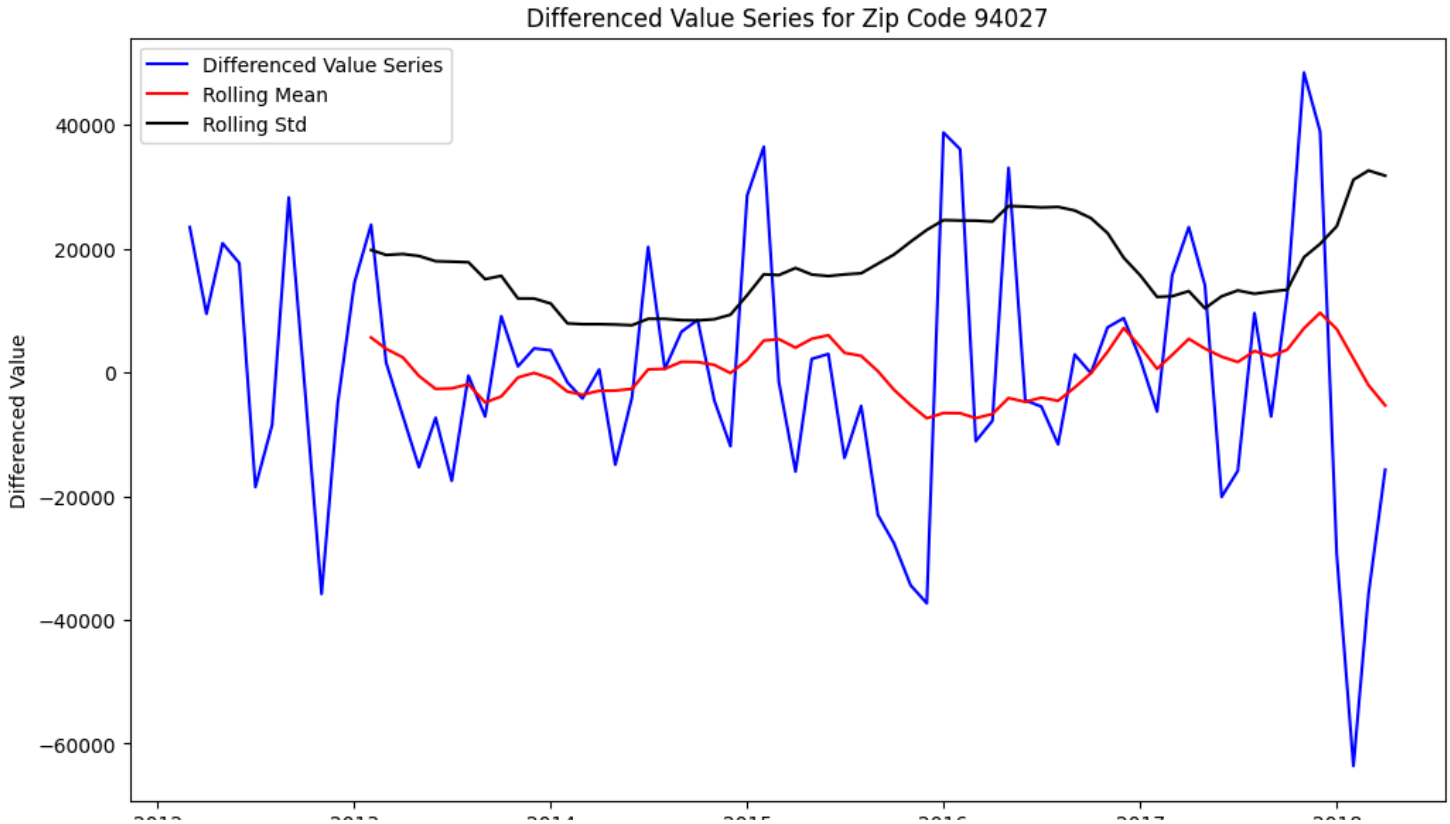
The ADF test results provide a p-value greater than or equal to the chosen significance level, we fail to reject the null hypothesis (H0). This indicates that the data is non-stationary, implying the presence of a stochastic trend or structure that evolves over time.

```
# Differencing the non-stationary time series and testing for stationarity again
differenced_adf_results = []

for i in range(5):
    # Differencing the time series
    dfs_ts[i]['differenced_ret'] = dfs_ts[i]['ret'].diff().dropna()
    dfs_ts[i]['differenced_ret'] = dfs_ts[i]['differenced_ret'].diff().dropna()

    # Get the rolling mean and std
    rolmean = dfs_ts[i]['differenced_ret'].rolling(window=12, center=False).mean()
    rolstd = dfs_ts[i]['differenced_ret'].rolling(window=12, center=False).std()

    # Conducting the ADF test on the differenced series
    results = adfuller(dfs_ts[i]['differenced_ret'].dropna())
    zipcode = dfs_ts[i]['ZipCode'].iloc[0]
    p_value = results[1]
    differenced_adf_results.append((zipcode, p_value))

    # Displaying the results
    print(f'ADFuller test p-value for differenced series of zipcode: {zipcode}')
    print('p-value:', p_value)
    if p_value > 0.05:
        print('Fail to reject the null hypothesis. Differenced data is not stationary.\n')
    else:
        print('Reject the null hypothesis. Differenced data is stationary.\n')

    # Plotting the differenced time series
    plt.figure(figsize=(12, 7))
    plt.plot(dfs_ts[i].index, dfs_ts[i]['differenced_ret'], color='blue', label='Differenced Value Se
    plt.plot(rolmean, label='Rolling Mean', color='red')
    plt.plot(rolstd, label='Rolling Std', color='black')
    plt.title(f'Differenced Value Series for Zip Code {zipcode}')
    plt.xlabel('Time')
    plt.ylabel('Differenced Value')
    plt.legend()
    plt.show()

differenced_adf_results
```

ADFuller test p-value for differenced series of zipcode: 90210
p-value: 9.801007383532519e-05
Reject the null hypothesis. Differenced data is stationary.

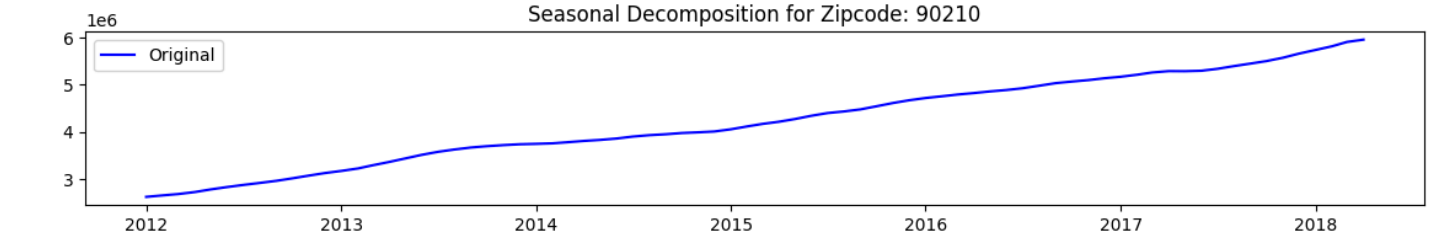


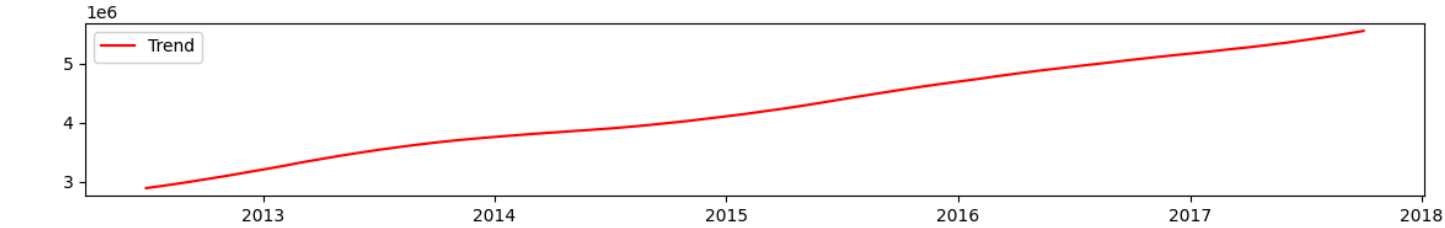
ADFuller test p-value for differenced series of zipcode: 94027
p-value: 8.636748394401712e-05
Reject the null hypothesis. Differenced data is stationary.

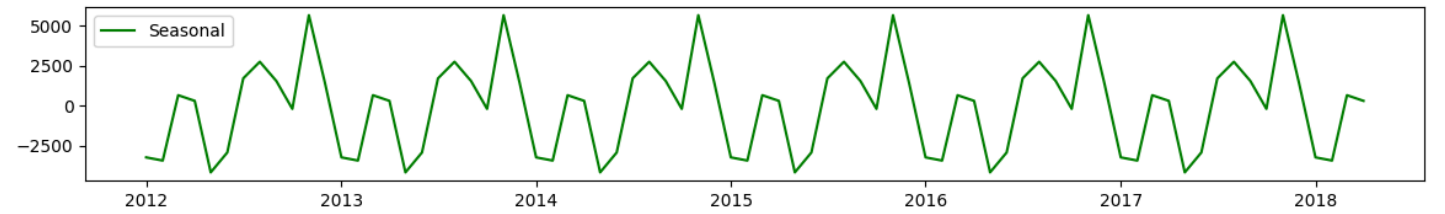


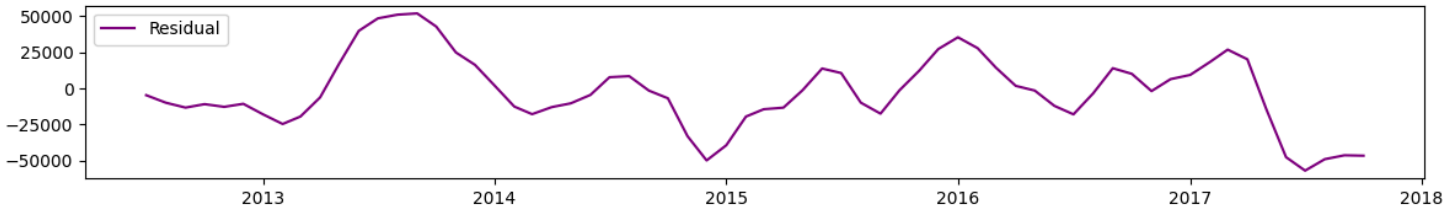
- By applying differencing, we successfully transformed the non-stationary time series data into stationary data for all of our top 5 zip codes. Lets apply seasonal decomposing which will help visualize the decomposed time series.

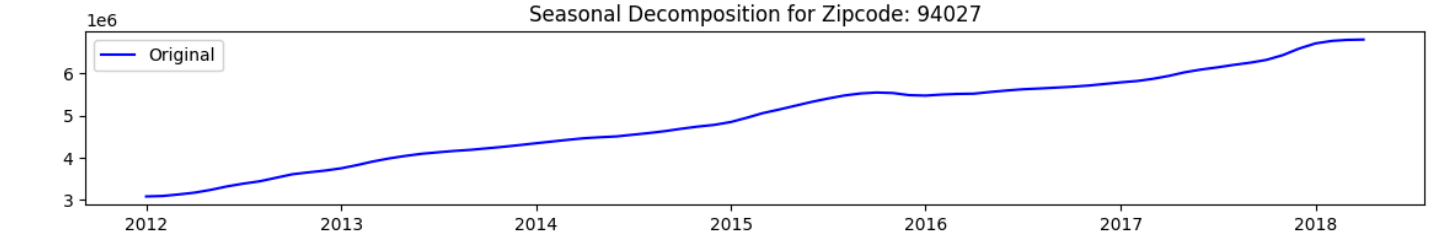
```
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

# Seasonal decomposition for each zip code
for i in range(5):
    # Perform seasonal decomposition
    decomposition = seasonal_decompose(dfs_ts[i]['ret'], period=12)

    # Plot the decomposed components with improved visibility
    plt.figure(figsize=(12, 8))
    plt.subplot(4, 1, 1)
    plt.plot(dfs_ts[i]['ret'], label='Original', color='blue')
    plt.legend(loc='upper left')
    plt.title(f'Seasonal Decomposition for Zipcode: {dfs_ts[i]["ZipCode"].iloc[0]}')

    plt.subplot(4, 1, 2)
    plt.plot(decomposition.trend, label='Trend', color='red')
    plt.legend(loc='upper left')

    plt.subplot(4, 1, 3)
    plt.plot(decomposition.seasonal, label='Seasonal', color='green')
    plt.legend(loc='upper left')

    plt.subplot(4, 1, 4)
    plt.plot(decomposition.resid, label='Residual', color='purple')
    plt.legend(loc='upper left')

    plt.tight_layout()
    plt.show()
```

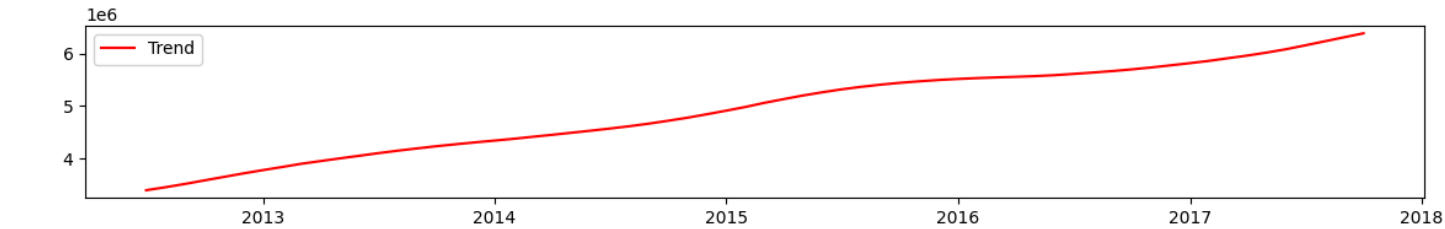

Seasonal Decomposition for Zipcode: 90210
1e6
Original

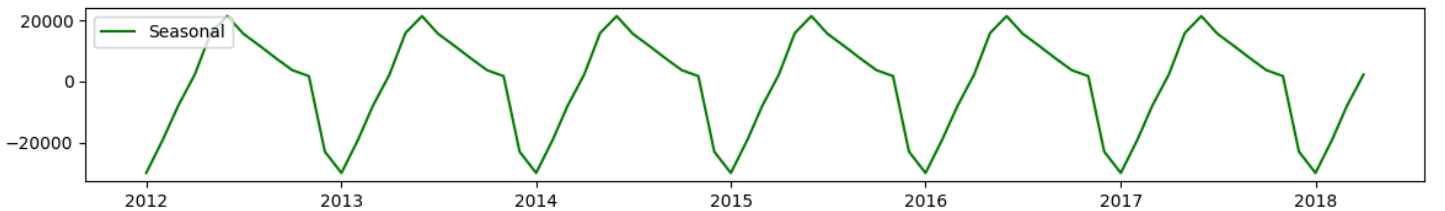

1e6
Trend

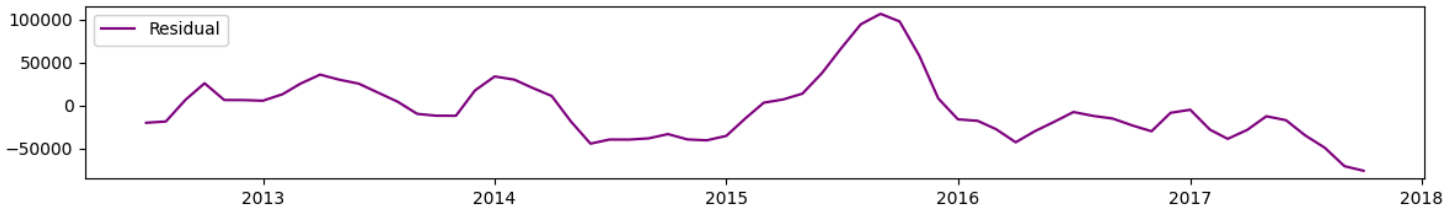

Seasonal


Residual


Seasonal Decomposition for Zipcode: 94027
1e6
Original


1e6
Trend


Seasonal


Residual


Seasonal Decomposition for Zipcode: 94123
1e6

- We will go ahead and plot their ACF and PACF plots, which will guide us on how to pass in our p,d,q parameters when fitting our time series models.

```
# The dataframes are contained in a list
dfs_ts
```

```
[           ZipCode           City    ROI        CV           std  \
Date
2012-01-01   90210  Beverly Hills  126.3  0.438323  1.222910e+06
2012-02-01   90210  Beverly Hills  126.3  0.438323  1.222910e+06
2012-03-01   90210  Beverly Hills  126.3  0.438323  1.222910e+06
2012-04-01   90210  Beverly Hills  126.3  0.438323  1.222910e+06
2012-05-01   90210  Beverly Hills  126.3  0.438323  1.222910e+06
...            ...            ...    ...       ...           ...
2017-12-01   90210  Beverly Hills  126.3  0.438323  1.222910e+06
2018-01-01   90210  Beverly Hills  126.3  0.438323  1.222910e+06
2018-02-01   90210  Beverly Hills  126.3  0.438323  1.222910e+06
2018-03-01   90210  Beverly Hills  126.3  0.438323  1.222910e+06
2018-04-01   90210  Beverly Hills  126.3  0.438323  1.222910e+06

                     mean     median  AverageHousePrice  houseprice        ret  \
Date
2012-01-01  2.789977e+06  2598900.0       2.769457e+06   2632200.0  2632200.0
2012-02-01  2.789977e+06  2598900.0       2.769457e+06   2663800.0  2663800.0
2012-03-01  2.789977e+06  2598900.0       2.769457e+06   2691700.0  2691700.0
2012-04-01  2.789977e+06  2598900.0       2.769457e+06   2734000.0  2734000.0
2012-05-01  2.789977e+06  2598900.0       2.769457e+06   2790000.0  2790000.0
...                  ...        ...                ...         ...        ...
2017-12-01  2.789977e+06  2598900.0       2.769457e+06   5661000.0  5661000.0
2018-01-01  2.789977e+06  2598900.0       2.769457e+06   5738200.0  5738200.0
2018-02-01  2.789977e+06  2598900.0       2.769457e+06   5816100.0  5816100.0
2018-03-01  2.789977e+06  2598900.0       2.769457e+06   5907800.0  5907800.0
2018-04-01  2.789977e+06  2598900.0       2.769457e+06   5956700.0  5956700.0

            differenced_ret
Date
2012-01-01              NaN
2012-02-01              NaN
2012-03-01         -3700.0
2012-04-01         14400.0
2012-05-01         13700.0
...                    ...
2017-12-01         15800.0
2018-01-01         -9700.0
2018-02-01           700.0
2018-03-01         13800.0
2018-04-01        -42800.0

[76 rows x 11 columns],
            ZipCode      City     ROI        CV           std          mean  \
Date
2012-01-01   94027  Atherton  120.19  0.371717  1.296224e+06  3.487129e+06
2012-02-01   94027  Atherton  120.19  0.371717  1.296224e+06  3.487129e+06
2012-03-01   94027  Atherton  120.19  0.371717  1.296224e+06  3.487129e+06
2012-04-01   94027  Atherton  120.19  0.371717  1.296224e+06  3.487129e+06
2012-05-01   94027  Atherton  120.19  0.371717  1.296224e+06  3.487129e+06
```

```
...           ...       ...     ...       ...           ...           ...
2017-12-01   94027  Atherton  120.19  0.371717  1.296224e+06  3.487129e+06
2018-01-01   94027  Atherton  120.19  0.371717  1.296224e+06  3.487129e+06
2018-02-01   94027  Atherton  120.19  0.371717  1.296224e+06  3.487129e+06
2018-03-01   94027  Atherton  120.19  0.371717  1.296224e+06  3.487129e+06
2018-04-01   94027  Atherton  120.19  0.371717  1.296224e+06  3.487129e+06

                median  AverageHousePrice  houseprice       ret  \
```

```
# Concatenate the list of DataFrames into a single DataFrame
df_combined = pd.concat(dfs_ts)
# Print the resulting DataFrame
df_combined
```

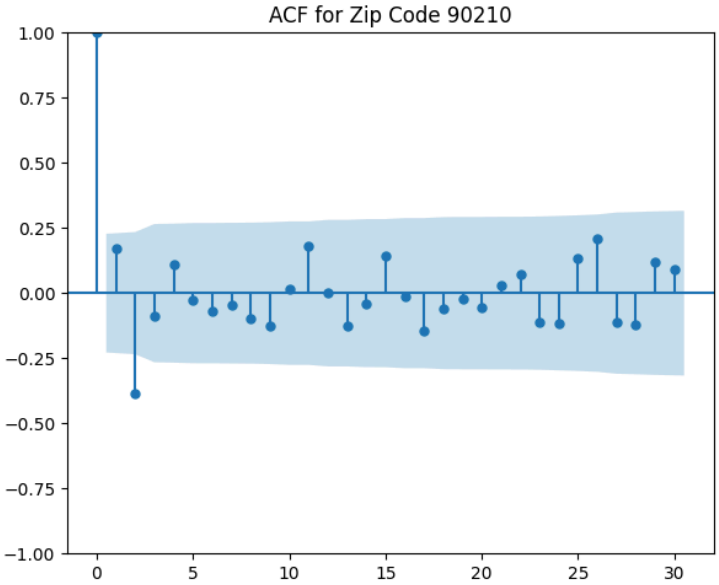
Date	ZipCode	City	ROI	CV	std	mean	median	AverageHousePrice
2012-01-01	90210	Beverly Hills	126.3	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06
2012-02-01	90210	Beverly Hills	126.3	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06
2012-03-01	90210	Beverly Hills	126.3	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06
2012-04-01	90210	Beverly Hills	126.3	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06
2012-05-01	90210	Beverly Hills	126.3	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06
...	...	...	...	...	...	...	...	...
2017-12-01	94115	San Francisco	79.1	0.369003	8.852487e+05	2.399030e+06	2402500.0	2.381428e+06
2018-01-01	94115	San Francisco	79.1	0.369003	8.852487e+05	2.399030e+06	2402500.0	2.381428e+06

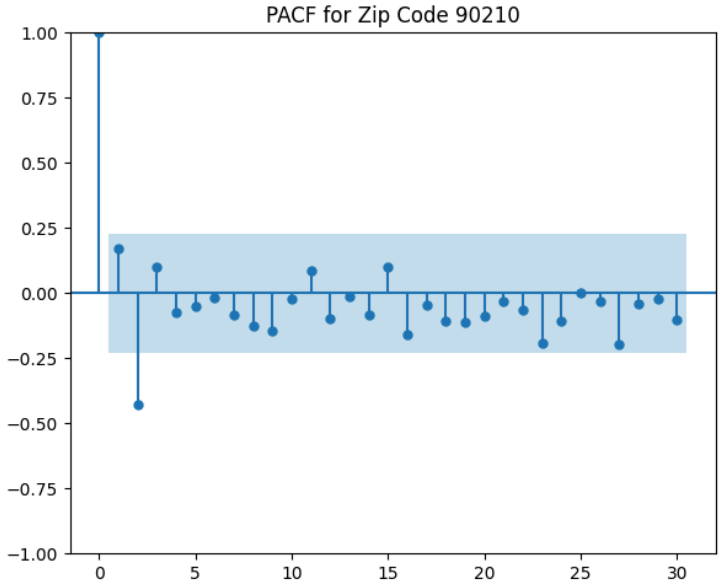
```
# Create a dictionary to store dataframes for each zipcode
zipcode_dataframes = {}

# Extract unique zipcodes from the dataframes
unique_zipcodes = df_combined['ZipCode'].unique()

# Iterate over unique zipcodes and create dataframes for each
for zipcode in unique_zipcodes:
    # Filter dataframe for the current zipcode
    zipcode_df = df_combined[df_combined['ZipCode'] == zipcode]

    # Drop rows with null values
    zipcode_df = zipcode_df.dropna()

    # Store the dataframe in the dictionary
    zipcode_dataframes[zipcode] = zipcode_df

# Access dataframe for a specific zipcode
# [94123, 33480, 90020, 94115, 94028]
zipcode_90210_df = zipcode_dataframes['90210']
zipcode_94027_df = zipcode_dataframes['94027']
zipcode_94123_df = zipcode_dataframes['94123']
zipcode_33480_df = zipcode_dataframes['33480']
zipcode_94115_df = zipcode_dataframes['94115']

# Confirm if it has worked
zipcode_90210_df
```

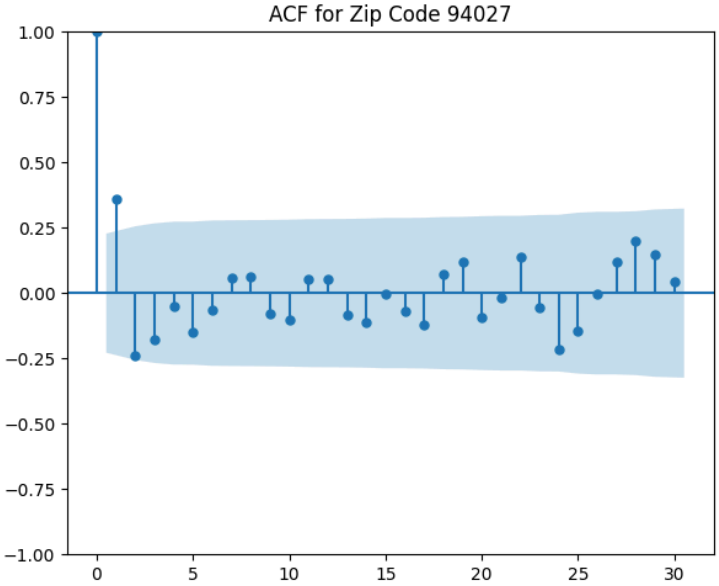
Date	ZipCode	City	ROI	CV	std	mean	median	AverageHousePrice	h
2012-03-01	90210	Beverly Hills	126.3	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06	
2012-04-01	90210	Beverly Hills	126.3	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06	
2012-05-01	90210	Beverly Hills	126.3	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06	
2012-06-01	90210	Beverly Hills	126.3	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06	
2012-07-01	90210	Beverly Hills	126.3	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06	
...	...	...	...	...	...	...	...	...	
2017-12-01	90210	Beverly Hills	126.3	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06	
2018-01-01	90210	Beverly Hills	126.3	0.438323	1.222910e+06	2.789977e+06	2598900.0	2.769457e+06	

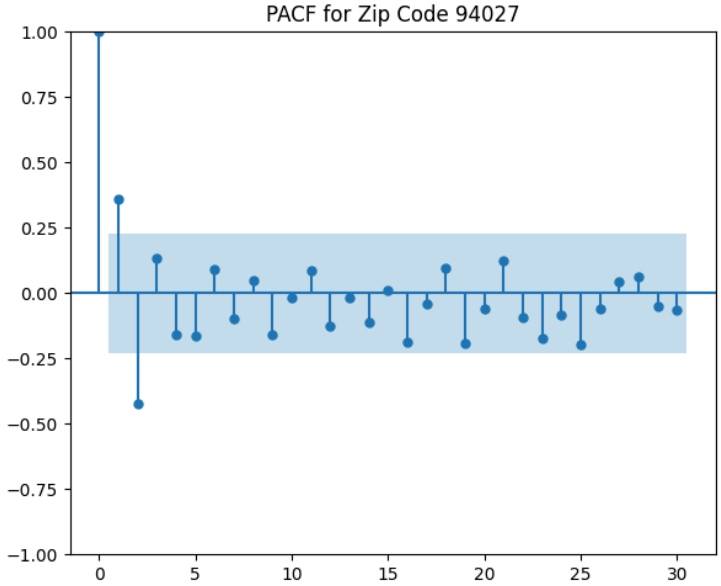
Plotting ACF and DCF.

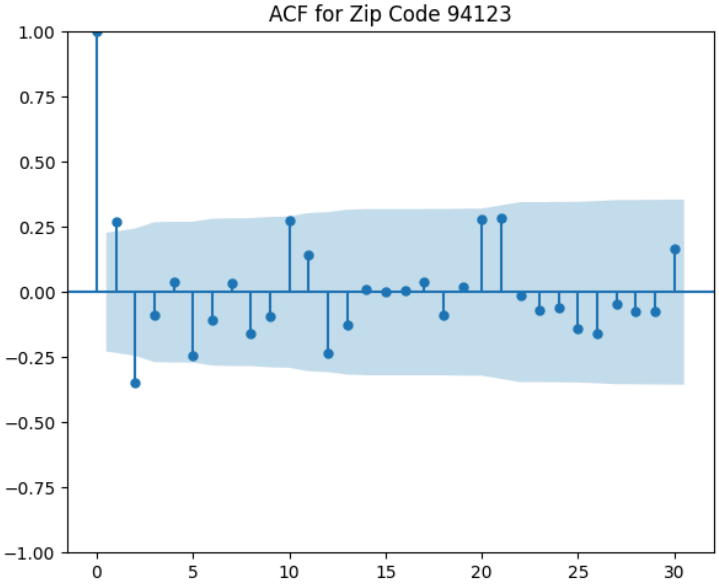
```
# Iterate over the dictionary of zipcode dataframes
for zipcode, zipcode_df in zipcode_dataframes.items():
    # Drop nulls
    zipcode_df.dropna(inplace=True)

    plt.figure(figsize=(12, 5))

    plt.subplot(121)
    plot_acf(zipcode_df['differenced_ret'], lags=30, ax=plt.gca())
    plt.title(f'ACF for Zip Code {zipcode}')

    plt.subplot(122)
    plot_pacf(zipcode_df['differenced_ret'], lags=30, ax=plt.gca())
    plt.title(f'PACF for Zip Code {zipcode}')

    plt.tight_layout()
    plt.show()
```

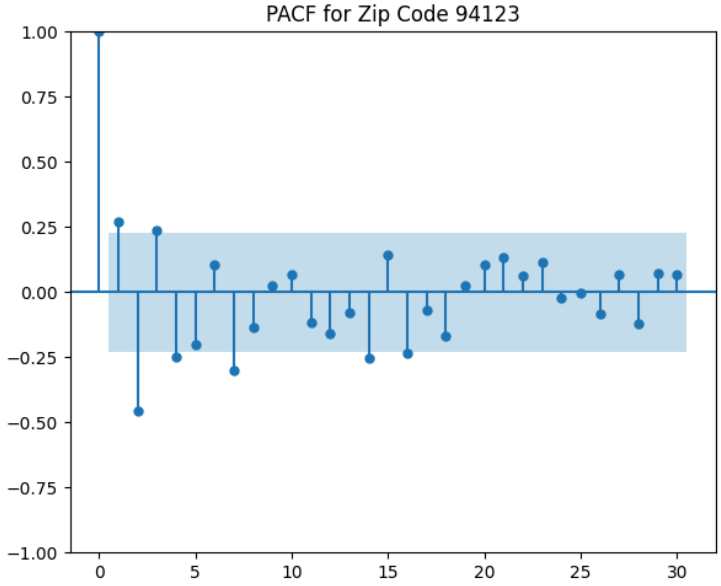

ACF for Zip Code 90210
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30

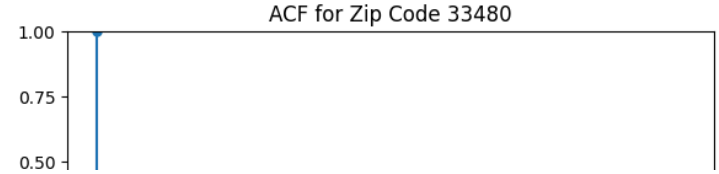

PACF for Zip Code 90210
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30

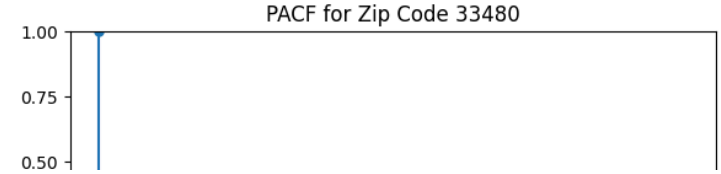

ACF for Zip Code 94027
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30


PACF for Zip Code 94027
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30


ACF for Zip Code 94123
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30


PACF for Zip Code 94123
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30


ACF for Zip Code 33480
1.00
0.75
0.50


PACF for Zip Code 33480
1.00
0.75
0.50

Train Test Split

To validate our models and assess their effectiveness, we need to split our time series data into training and test sets. In the context of time series analysis, we'll allocate the first 80% of our data to the training set and the remaining 20% to the test set. Since our data spans from January 2012, covering 6.17 years, the training set will consist of approximately 4.93 years, and the test set will include about 1.23 years of data. This ratio ensures that our models are trained on a sufficient amount of historical data while allowing for a meaningful evaluation on unseen future data. By validating our models on a test set, we can gain confidence in their predictive performance, which can be extrapolated to make more accurate forecasts for the upcoming year. This approach provides a robust basis for assessing the reliability of our predictions, especially when considering longer-term forecasting horizons.

```
from datetime import datetime

# Define the start and end dates
start_date = datetime(2012, 2, 1)  # 01/02/2012
end_date = datetime(2018, 4, 1)    # 01/04/2018

# Calculate the difference between the two dates
difference = end_date - start_date

# Convert the difference to years (assuming 365 days per year)
years = difference.days / 365

print("Number of years between", start_date.date(), "and", end_date.date(), "is:", years)
```

```
Number of years between 2012-02-01 and 2018-04-01 is: 6.167123287671233
```

```
# Filter the columns based on the date range
start_date = '1996-04'
end_date = '2011-12'
columns_to_drop = high_end_properties.columns[(high_end_properties.columns >= start_date) & (high_end_

# Drop the filtered columns
high_end_properties.drop(columns=columns_to_drop, inplace=True)

high_end_properties
```

	RegionID	ZipCode	City	State	Metro	CountyName	SizeRank	2012-01	2012-02
9	97564	94109	San Francisco	CA	San Francisco	San Francisco	10	2244700.0	2261100.0
20	61625	10011	New York	NY	New York	New York	21	7103800.0	7428300.0
21	61703	10128	New York	NY	New York	New York	22	6271000.0	6227500.0
30	96027	90046	Los Angeles	CA	Los Angeles-Long Beach-Anaheim	Los Angeles	31	938800.0	936900.0
84	72442	33139	Miami Beach	FL	Miami-Fort Lauderdale	Miami-Dade	85	1443900.0	1460300.0
...	...	...	...	...	...	...	...	...	...
13885	96639	92091	Rancho Santa Fe	CA	San Diego	San Diego	13886	1622400.0	1623900.0
13900	60722	07620	Alpine	NJ	New York	Bergen	13901	2285200.0	2280600.0
14622	71578	31561	Sea Island	GA	Brunswick	Glynn	14623	2053900.0	1992500.0
14641	97901	94970	Stinson Beach	CA	San Francisco	Marin	14642	1146400.0	1151200.0
14711	95893	89413	Glenbrook	NV	Gardnerville Ranchos	Douglas	14712	1188800.0	1187500.0

118 rows × 84 columns

```
# List the columns you want to drop
columns_to_drop = ['RegionID', 'State', 'Metro','CountyName','SizeRank']

# Use the drop() method to drop the specified columns
# Specify axis=1 to indicate that you are dropping columns
high_end1 = high_end_properties.drop(columns=columns_to_drop, axis=1)

# After dropping the columns, you can verify the changes by printing the DataFrame
high_end1
```

	ZipCode	City	2012-01	2012-02	2012-03	2012-04	2012-05	2012-06	2012-0
9	94109	San Francisco	2244700.0	2261100.0	2276900.0	2303700.0	2339400.0	2381700.0	2443600.(
20	10011	New York	7103800.0	7428300.0	7694600.0	7872000.0	7994800.0	8003300.0	7972900.(
21	10128	New York	6271000.0	6227500.0	5931700.0	5578000.0	5401900.0	5373300.0	5432800.(
30	90046	Los Angeles	938800.0	936900.0	936400.0	941100.0	954000.0	971700.0	991100.(
84	33139	Miami Beach	1443900.0	1460300.0	1478600.0	1506500.0	1540000.0	1561600.0	1577100.(
...	...	...	...	...	...	...	...	...	..
13885	92091	Rancho Santa Fe	1622400.0	1623900.0	1625400.0	1638000.0	1658400.0	1675400.0	1689800.(
13900	07620	Alpine	2285200.0	2280600.0	2256000.0	2229100.0	2209200.0	2193900.0	2207000.(
14622	31561	Sea Island	2053900.0	1992500.0	1952900.0	1932500.0	1901200.0	1863000.0	1861800.(
14641	94970	Stinson Beach	1146400.0	1151200.0	1157500.0	1189600.0	1226400.0	1229000.0	1237400.(
14711	89413	Glenbrook	1188800.0	1187500.0	1189000.0	1194400.0	1199700.0	1203000.0	1205800.(

118 rows × 79 columns

```
# creating a function that changes the dataframe structure from wide view to long view

def melt_df(high_end1):
    melted = pd.melt(high_end1, id_vars=['ZipCode','City','AverageHousePrice'
                                  ], var_name='Date',value_name='houseprice')
    melted['Date'] = pd.to_datetime(melted['Date'], infer_datetime_format=True)
    melted.set_index('Date', inplace=True)
    melted.rename(columns={"value": "houseprice"}, inplace=True)
    melted = melted.dropna(subset=['houseprice'])
    return melted


# Create a copy of the cleaned DataFrame with a new name (e.g., new_df)
model5_df = high_end1.copy()

# Apply the melt_df function to convert the DataFrame from wide to long format
melted5_df = melt_df(model5_df)

# Display the final cleaned data
melted5_df
```

Date	ZipCode	City	AverageHousePrice	houseprice
2012-01-01	94109	San Francisco	2.378056e+06	2244700.0
2012-01-01	10011	New York	4.735551e+06	7103800.0
2012-01-01	10128	New York	5.047574e+06	6271000.0
2012-01-01	90046	Los Angeles	1.034152e+06	938800.0
2012-01-01	33139	Miami Beach	1.447709e+06	1443900.0
...	...	...	...	...
2018-04-01	92091	Rancho Santa Fe	1.736507e+06	2458000.0
2018-04-01	07620	Alpine	2.064773e+06	3069100.0
2018-04-01	31561	Sea Island	1.395810e+06	2440000.0
2018-04-01	94970	Stinson Beach	1.352160e+06	2678700.0
2018-04-01	89413	Glenbrook	1.402851e+06	2161900.0

8968 rows × 4 columns

```
import pandas as pd

# Assuming your DataFrame is named df
# Reset index to ensure Date becomes a column
melted5_df.reset_index(inplace=True)

# Pivot the DataFrame
pivot_df = melted5_df.pivot_table(index='Date', columns='City', values='houseprice')

# Reset column index name
pivot_df.columns.name = None

# Display the pivoted DataFrame
pivot_df
```

Date	Alamo	Alpine	Amagansett	Aspen	Atherton	Avalon	Berkeley	Beverly Hills	
2012-01-01	1111400.0	2285200.0	1857900.0	3141700.0	3086600.0	1251800.0	1058400.0	1.859100e+06	11
2012-02-01	1108900.0	2280600.0	1852600.0	3102900.0	3097900.0	1244900.0	1057900.0	1.873533e+06	12
2012-03-01	1121700.0	2256000.0	1846300.0	3137700.0	3132700.0	1244000.0	1061700.0	1.891800e+06	12
2012-04-01	1146800.0	2229100.0	1869900.0	3189600.0	3177000.0	1247600.0	1072500.0	1.920033e+06	12
2012-05-01	1159000.0	2209200.0	1893100.0	3207900.0	3242200.0	1253400.0	1079900.0	1.954033e+06	12
...	...	...	...	...	...	...	...	...	
2017-12-01	1857100.0	3047100.0	3008500.0	4321700.0	6581800.0	1594400.0	1904700.0	3.758667e+06	19
2018-01-01	1866000.0	3054800.0	3051700.0	4381800.0	6705000.0	1626000.0	1938800.0	3.801433e+06	19
2018-02-01	1879100.0	3061200.0	3099000.0	4469700.0	6764600.0	1643500.0	1958900.0	3.840667e+06	19
2018-03-01	1891000.0	3060700.0	3132200.0	4626100.0	6788400.0	1654000.0	1970900.0	3.880300e+06	19
2018-04-01	1895600.0	3069100.0	3141100.0	4766600.0	6796500.0	1665600.0	1974100.0	3.899300e+06	19

76 rows × 82 columns

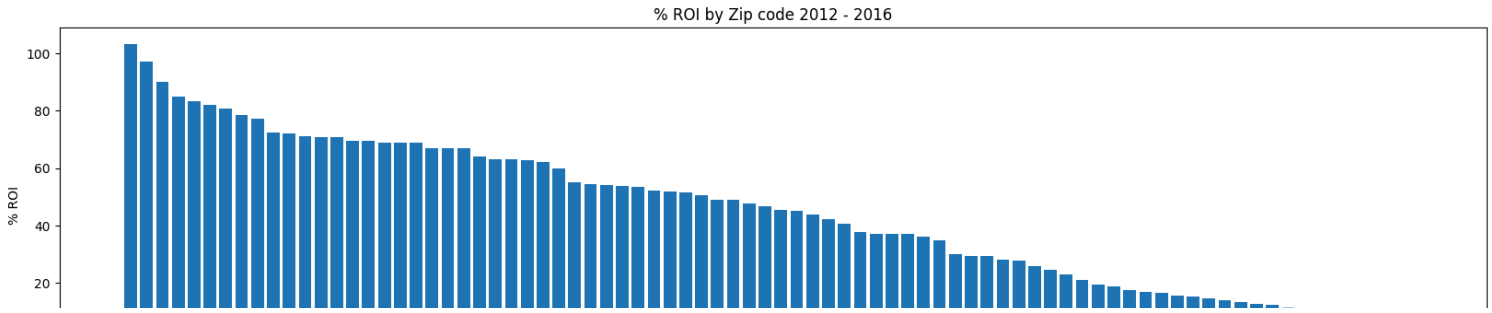
```
# Printing out the lengths of our unsplit time series
print(f'Whole series lengths: {len(pivot_df)} \n')

# Define the proportion of data to be used for training (e.g., 80%)
train_proportion = 0.8

# Calculate the index to split the data into training and testing sets
split_index = int(len(pivot_df) * train_proportion)

# Split the data into training and testing sets
train_data = pivot_df.iloc[:split_index]
test_data = pivot_df.iloc[split_index:]

# Alternatively, you can specify the split point based on a specific date
# split_date = pd.to_datetime('yyyy-mm-dd')  # Specify the date
# train_data = time_series_data[time_series_data.index <= split_date]
# test_data = time_series_data[time_series_data.index > split_date]

# Print the number of data points in the training and testing sets
print("Number of data points in the training set:", len(train_data))
print("Number of data points in the testing set:", len(test_data))

# Checking the length in years of our train and test sets
print(f'Train set length in years: {round(len(train_data) / 12, 2)}')
print(f'Test set length in years: {round(len(test_data) / 12, 2)}')
```

```
Whole series lengths: 76

Number of data points in the training set: 60
Number of data points in the testing set: 16
Train set length in years: 5.0
Test set length in years: 1.33
```

```
#import numpy as np

# Calculate the index position corresponding to 80% of the total number of rows
eighty_percent_index = int(0.8 * len(train_data))

# Retrieve the date associated with the calculated index position
last_date_80_percent = train_data.index[eighty_percent_index]

print("Last date when considering 80% of the data:", last_date_80_percent)
```

```
Last date when considering 80% of the data: 2016-01-01 00:00:00
```

```
names = []
historical_roi = []

#This for loop adds the information to the two lists
for i in range(len(train_data.columns)):

    clean_name = train_data.columns[i][:-4]

    initial_val = train_data[train_data.columns[i]]['2012-01-01']
    present_val = train_data[train_data.columns[i]]['2016-01-01']

    roi = round(((present_val - initial_val) / initial_val) * 100, 2)

    names.append(clean_name)
    historical_roi.append(roi)


# Turning the data into a pandas dataframe
roi_df = pd.DataFrame()
roi_df['City'] = names
roi_df['% ROI'] = historical_roi
roi_df.sort_values(['% ROI'], inplace=True, ascending=False)
roi_df.set_index('City', inplace=True)

# Convert column names to strings
roi_df.columns = roi_df.columns.astype(str)

fig, ax = plt.subplots(figsize=(20, 5))
plt.bar(range(len(roi_df)), roi_df['% ROI'])  # Use column index for x-axis
plt.title('% ROI by Zip code 2012 - 2016')
plt.xlabel('City')
plt.ylabel('% ROI')
plt.xticks(ticks=range(len(roi_df)), labels=roi_df.index, rotation=45)
plt.axhline(0, color='k')
plt.show()


# Displaying our top five choices based on EDA
roi_df.head()
```


% ROI by Zip code 2012 - 2016
% ROI
100
80
60
40
20