Introduction

Zillow is a real estate company that provides extensive datasets that offer insights into the housing market.The stakeholder in this project is seeking to construct residential homes in the United States that provide a return on investment.

Business Understanding

Stakeholders understanding the market environment is very crucial.This is because they rely on data-driven insights to make informed decisions regarding property investments.By using Zillow data(*zillow_data.csv*), stakeholders can gain a deeper understanding of market dynamics, identify growth opportunities, and mitigate risks associated with market fluctuations. Our stake holders in this case are;

a)Real estate investors-They seek data-driven insights to identify profitable investment opportunities.
b)Developers-Rely on market intelligence inorder to assess demand for new residential and commercial properties.
c)Policymakers-They use data to formulate housing policies and promote sustainable urban development.

Problem Statement

- Our stakeholder for this project, is a ***real estate investment firm*** seeking to construct residential homes in the United States. The main aim is to build homes and properties that have a high return on investment.
  This project entails performing a time series analysis on a Zillow dataset covering various locations across the country to identify these prime investment areas.

Objectives:

- Identify the top 5 zip codes for the real estate agency to invest in.
- Forecast future house prices in these zip codes.
- Provide insights and recommendations

Benefits:

The benefits of generating Insights and recommendations are;

1. Informed Decision-Making: This means that our stakeholders can make data-driven decisions on property acquisitions and financial investments, leading to high returns and reduced risks.
2. Competitive Advantage:Getting market Insights in a timely manner enables stakeholders to stay ahead of competitors and capitalize on market opportunities.
3. Policy Impact: Policymakers can create targeted interventions to promote housing

Data Understanding

The dataset used in this project comprises historical median house prices from various states in the USA, spanning from April 1996 to April 2018 (22 years). This data was obtained from the Zillow website (https://www.zillow.com/research/data/).

The dataset contains 14,723 rows and 272 columns, with 4 categorical columns and 268 numerical columns.

Column Names and Descriptions:

- **RegionID**: Unique identifier for each region.
- **RegionName**: Names of the regions (zip codes).
- **City**: Names of the cities for the regions.
- **State**: Names of the states.
- **Metro**: Names of the metropolitan areas.
- **CountyName**: Names of the counties.
- **SizeRank**: Rank of zip codes by urbanization.
- **Date Columns (265 Columns)**: Median house prices across the years.

Data Preparation

- Importing libraries:
  Before we embark on our journey through the housing market analysis, it's essential to equip ourselves with a comprehensive set of data science tools. Our toolkit includes numpy and pandas for advanced data handling, matplotlib and seaborn for engaging visualization,

and a collection of sklearn features ans statsmodela for data preprocessing, creation, time series analysis, and performance evaluation.

In [ ]:
```
# importing relevant libraries

# Analysis libraries
import pandas as pd
import numpy as np

# Visualization libraries
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns


# Warning libraries
import warnings
warnings.simplefilter("ignore")
warnings.filterwarnings('ignore')

# Modelling libraries
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.arima.model import ARIMA
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller

# import pmdarima as pm #a library to help with auto_arima
import itertools

# Metrics Libraries
from sklearn.metrics import mean_absolute_percentage_error
```

To begin our Time series analysis, we first import the dataset from `'zillow_data.csv'` using pandas, a powerful tool for data manipulation. We create a copy of the dataset to preserve the original data, enabling us to explore and manipulate it without altering the source. We then preview the dataset to familiarize ourselves with its structure. Using `df.head()`, we can view the top 5 rows, and `df.tail()` provides a look at the last 5 rows, giving us a comprehensive overview of the dataset's contents.

In [ ]:
```
data = pd.read_csv('zillow_data.csv')
df = data.copy()
df.head() #preview the first 5 columns of the data set
```

Out[2]:

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

5 rows × 272 columns

In [ ]:
```
df.tail() # preview the last five columns of the dataset
```

Out[3]:

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	
14718	58333	1338	Ashfield	MA	Greenfield Town	Franklin	14719	94600.0	
14719	59107	3293	Woodstock	NH	Claremont	Grafton	14720	92700.0	
14720	75672	40404	Berea	KY	Richmond	Madison	14721	57100.0	
14721	93733	81225	Mount Crested Butte	CO	NaN	Gunnison	14722	191100.0	19
14722	95851	89155	Mesquite	NV	Las Vegas	Clark	14723	176400.0	17

5 rows × 272 columns

In [ ]:
```
df.columns #view the column names
```

Out[4]:
```
Index(['RegionID', 'RegionName', 'City', 'State', 'Metro', 'CountyName',
       'SizeRank', '1996-04', '1996-05', '1996-06',
       ...
       '2017-07', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12',
       '2018-01', '2018-02', '2018-03', '2018-04'],
      dtype='object', length=272)
```

- Data Preview and Cleaning
  The RegionName contains zipcode data. It will be renamed to Zipcode.

In [ ]:
```
df = df.rename(columns={'RegionName': 'Zipcode'})
```

- Checking for duplicates
  Our dataset does not have duplicates

In [ ]:
```
df.duplicated().sum()
```

Out[6]:
```
0
```

- Checking the data information
  This is majorly the size of the data and the data types

In [ ]:
```
df.info()
```

```
<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
```

In [ ]:
```
df.select_dtypes(include='object').info()#Checking data with categorical value
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14723 entries, 0 to 14722
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   City        14723 non-null  object
 1   State       14723 non-null  object
 2   Metro       13680 non-null  object
 3   CountyName  14723 non-null  object
dtypes: object(4)
memory usage: 460.2+ KB
```

In [ ]:
```
df.select_dtypes(include='int64').info() #Checking data with numerical values(
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14723 entries, 0 to 14722
Data columns (total 49 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   RegionID  14723 non-null  int64
 1   Zipcode   14723 non-null  int64
 2   SizeRank  14723 non-null  int64
 3   2014-07   14723 non-null  int64
 4   2014-08   14723 non-null  int64
 5   2014-09   14723 non-null  int64
 6   2014-10   14723 non-null  int64
 7   2014-11   14723 non-null  int64
 8   2014-12   14723 non-null  int64
 9   2015-01   14723 non-null  int64
 10  2015-02   14723 non-null  int64
 11  2015-03   14723 non-null  int64
 12  2015-04   14723 non-null  int64
 13  2015-05   14723 non-null  int64
 14  2015-06   14723 non-null  int64
 15  2015-07   14723 non-null  int64
 16  2015-08   14723 non-null  int64
 17  2015-09   14723 non-null  int64
 18  2015-10   14723 non-null  int64
 19  2015-11   14723 non-null  int64
 20  2015-12   14723 non-null  int64
 21  2016-01   14723 non-null  int64
 22  2016-02   14723 non-null  int64
 23  2016-03   14723 non-null  int64
 24  2016-04   14723 non-null  int64
 25  2016-05   14723 non-null  int64
 26  2016-06   14723 non-null  int64
 27  2016-07   14723 non-null  int64
 28  2016-08   14723 non-null  int64
 29  2016-09   14723 non-null  int64
 30  2016-10   14723 non-null  int64
 31  2016-11   14723 non-null  int64
 32  2016-12   14723 non-null  int64
 33  2017-01   14723 non-null  int64
 34  2017-02   14723 non-null  int64
 35  2017-03   14723 non-null  int64
 36  2017-04   14723 non-null  int64
 37  2017-05   14723 non-null  int64
 38  2017-06   14723 non-null  int64
 39  2017-07   14723 non-null  int64
 40  2017-08   14723 non-null  int64
 41  2017-09   14723 non-null  int64
 42  2017-10   14723 non-null  int64
 43  2017-11   14723 non-null  int64
 44  2017-12   14723 non-null  int64
 45  2018-01   14723 non-null  int64
 46  2018-02   14723 non-null  int64
 47  2018-03   14723 non-null  int64
 48  2018-04   14723 non-null  int64
dtypes: int64(49)
memory usage: 5.5 MB
```

In [ ]:
```
df.select_dtypes(include='float64').info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14723 entries, 0 to 14722
Columns: 219 entries, 1996-04 to 2014-06
dtypes: float64(219)
memory usage: 24.6 MB
```

- The columns have three data types: Integers, Float, Object
- The dataset contains 14,723 rows and 272 columns, with 4 categorical columns and 268 numerical columns.

- **Handling missing values**

In [ ]:
```
print(f'The data has {df.isna().sum().sum()} missing values')
#checking the percentage of missing values in columns that have object data typ
df.select_dtypes(include='object').isna().sum()/len(df) *100
```

```
The data has 157934 missing values
```

Out[11]:
```
City          0.000000
State         0.000000
Metro         7.084154
CountyName    0.000000
dtype: float64
```

The missing values in the metro column will be replaced with 'missing' to avoid the risk of incorrect assumptions that might come from default imputation methods like filling with the mode, which might interfere with the data integrity.

In [ ]:
```
# imputing the missing values by replacing them with 'missing'

df.Metro.fillna('missing', inplace=True)
```

In [ ]:
```
#checking the percentage of missing values in columns that have integer data ty
df.select_dtypes(include='int64').isna().sum()/len(df) *100
```

Out[13]:
```
RegionID    0.0
Zipcode     0.0
SizeRank    0.0
2014-07     0.0
2014-08     0.0
2014-09     0.0
2014-10     0.0
2014-11     0.0
2014-12     0.0
2015-01     0.0
2015-02     0.0
2015-03     0.0
2015-04     0.0
2015-05     0.0
2015-06     0.0
2015-07     0.0
2015-08     0.0
2015-09     0.0
2015-10     0.0
2015-11     0.0
2015-12     0.0
2016-01     0.0
2016-02     0.0
2016-03     0.0
2016-04     0.0
2016-05     0.0
2016-06     0.0
2016-07     0.0
2016-08     0.0
2016-09     0.0
2016-10     0.0
2016-11     0.0
2016-12     0.0
2017-01     0.0
2017-02     0.0
2017-03     0.0
2017-04     0.0
2017-05     0.0
2017-06     0.0
2017-07     0.0
2017-08     0.0
2017-09     0.0
2017-10     0.0
2017-11     0.0
2017-12     0.0
2018-01     0.0
2018-02     0.0
2018-03     0.0
2018-04     0.0
dtype: float64
```

Integer data types have no missing values.

In [ ]:
```
#checking the percentage of missing values in columns that have float data type
df.select_dtypes(include='float64').isna().sum()/len(df) *100
```

Out[14]:
```
1996-04    7.056986
1996-05    7.056986
1996-06    7.056986
1996-07    7.056986
1996-08    7.056986
             ...
2014-02    0.380357
2014-03    0.380357
2014-04    0.380357
2014-05    0.380357
2014-06    0.380357
Length: 219, dtype: float64
```

- The missing values in the date columns will be filled through interpolation.This helps in maintaining the continuity and trends in the dataset, which is particularly important in time series data

In [ ]:
```
# interpolate missing values on date columns
df.interpolate(inplace=True)
```

- Checking different columns data types to ensure that they are correct.
- Region ID is a unique identifier so it will be dropped

In [ ]:
```
for x in df.columns[:5]:
    print(x, ":", df[x].dtype,'\n')
```

```
RegionID : int64

Zipcode : int64

City : object

State : object

Metro : object
```

In [ ]:
```
for x in df.columns[:5]:
    print(x, ":", df[x].unique(),'\n')
```

```
RegionID : [84654 90668 91982 ... 75672 93733 95851]

Zipcode : [60657 75070 77494 ... 40404 81225 89155]

City : ['Chicago' 'McKinney' 'Katy' ... 'Pine Valley' 'Esopus'
 'Mount Crested Butte']

State : ['IL' 'TX' 'NY' 'CA' 'FL' 'TN' 'NC' 'GA' 'DC' 'MO' 'OK' 'AZ' 'NJ' 'M
D'
 'VA' 'WA' 'OH' 'MI' 'MA' 'KS' 'NM' 'CT' 'NV' 'PA' 'CO' 'OR' 'IN' 'SC'
 'KY' 'AR' 'ND' 'MN' 'AL' 'DE' 'LA' 'MS' 'ID' 'MT' 'HI' 'WI' 'UT' 'ME'
 'SD' 'WV' 'IA' 'RI' 'NE' 'WY' 'AK' 'NH' 'VT']

Metro : ['Chicago' 'Dallas-Fort Worth' 'Houston' 'El Paso' 'New York'
 'San Francisco' 'The Villages' 'Nashville'
 'Los Angeles-Long Beach-Anaheim' 'Austin' 'Charlotte' 'McAllen' 'Atlanta'
 'Washington' 'San Antonio' 'Clarksville' 'St. Louis' 'Oklahoma City'
 'Phoenix' 'Baltimore' 'Miami-Fort Lauderdale' 'Brownsville'
 'Virginia Beach' 'Seattle' 'Cleveland' 'Ann Arbor' 'Boston' 'Kansas City'
 'Sacramento' 'Tucson' 'Jacksonville' 'Napa' 'San Diego' 'Albuquerque'
 'Hartford' 'Las Vegas' 'Lancaster' 'Fresno' 'Denver' 'Detroit' 'Vallejo'
 'Pittsburgh' 'Columbus' 'Portland' 'Riverside' 'Yuma' 'Ithaca'
 'Springfield' 'Fort Myers' 'missing' 'Cincinnati' 'Tampa' 'Columbia'
 'Lafayette-West Lafayette' 'Fayetteville' 'Raleigh' 'Kennewick'
 'College Station' 'Hagerstown' 'Philadelphia' 'Richmond' 'Indianapolis'
 'Tulsa' 'Orlando' 'Greenville' 'Ventura' 'San Jose' 'Findlay' 'Greeley'
 'Daytona Beach' 'Zanesville' 'Jonesboro' 'Fargo' 'Port St. Lucie'
 'Eugene' 'Rochester' 'Birmingham' 'Hot Springs' 'Durham' 'Fort Collins'
 'Lansing' 'Frankfort' 'Charleston' 'Bowling Green' 'Chillicothe'
 'Winston-Salem' 'Buffalo' 'Bakersfield' 'Visalia' 'Savannah' 'Hanford'
 'Santa Cruz' 'Hickory' 'Baton Rouge' 'Oxford' 'Warner Robins'
 'Montgomery' 'Spokane' 'New Haven' 'York' 'Gainesville' 'Longview'
 'Modesto' 'Twin Falls' 'Kalispell' 'Chattanooga' 'Salina' 'New Orleans'
 'Jackson' 'Elizabethtown' 'Louisville/Jefferson County' 'Albany'
 'Minneapolis-St Paul' 'Tallahassee' 'Flint' 'Urban Honolulu' 'Madison'
 'Pittsfield' 'Ogden' 'Billings' 'Starkville' 'La Crosse' 'Myrtle Beach'
 'Providence' 'Marion' 'Bloomington' 'Lubbock' 'Boise City' 'Little Rock'
 'Dayton' 'Augusta' 'Blacksburg' 'Memphis' 'Medford' 'Lawton' 'Bellingham'
 'Amarillo' 'Canton' 'Mobile' 'Gulfport' 'Davenport' 'Laredo' 'Stamford'
 'Mankato' 'Bangor' 'Auburn' 'Bend' 'Hilo' 'Huntsville' 'Athens'
 'Lakeland' 'Kalamazoo' 'Tyler' 'Boulder' 'Salem' 'Florence'
 'Colorado Springs' 'Michigan City' 'Sioux Falls'
 'North Port-Sarasota-Bradenton' 'Melbourne' 'Wichita' 'Reno' 'Toledo'
 'Rapid City' 'Lafayette' 'Grand Forks' 'Killeen' 'Oshkosh' 'Knoxville'
 'Binghamton' 'Kingsport' 'Burlington' 'Houma' 'Tuscaloosa' 'Williamsport'
 'Green Bay' 'Redding' 'Lexington' 'Hilton Head Island' 'Allentown'
 'Wilmington' 'Asheville' 'Santa Fe' 'Santa Maria-Santa Barbara'
 'Corvallis' 'Milwaukee' 'Stockton' 'Owensboro' 'Wooster' 'Fairmont'
 'Lynchburg' 'Dubuque' 'Trenton' 'Morgantown' 'Cedar Rapids' 'Eau Claire'
 'Sandusky' 'Fond du Lac' 'Dalton' 'Grand Rapids' 'Hattiesburg'
 'State College' 'Lincoln' 'Olympia' 'Pensacola' 'Provo' 'Peoria'
 'Manitowoc' 'Worcester' 'Roanoke' 'Kerrville' 'Scranton' 'Akron'
 'New Bern' 'Salinas' 'Waterloo' 'San Luis Obispo' 'Naples' 'Santa Rosa'
 'Moses Lake' 'Oak Harbor' 'Jefferson City' 'Cumberland' 'Milledgeville'
 'Johnson City' 'Del Rio' 'Cookeville' 'Watertown' 'Charlottesville'
 'Elkhart' 'Walla Walla' 'Appleton' 'Utica' 'Yakima' 'Wenatchee'
 'Muskegon' 'Harrisburg' "Coeur d'Alene" 'Salt Lake City' 'Midland'
```

```
'Bozeman' 'Staunton' 'Jamestown' 'Salisbury' 'Atlantic City'
'Champaign-Urbana' 'Victoria' 'Kokomo' 'St. George' 'Panama City'
'Corpus Christi' 'Mount Airy' 'Elizabeth City' 'Odessa' 'New London'
'Sierra Vista' 'Eagle Pass' 'Cheyenne' 'Wilson' 'Lebanon' 'Lewiston'
'Anchorage' 'Harrisonburg' 'Dothan' 'Key West' 'Goldsboro' 'Des Moines'
'Niles' 'Chico' 'Rome' 'Port Angeles' 'Paragould' 'Prescott' 'San Angelo'
'Lake Charles' 'Decatur' 'Kingston' 'Cape Girardeau' 'Vineland' 'Monroe'
'Youngstown' 'Sanford' 'Danville' 'Joplin' 'Kahului' 'Logan'
'Crestview-Fort Walton Beach-Destin' 'Shelton' 'Boone' 'Statesboro'
'Fort Wayne' 'Beaumont' 'El Centro' 'Punta Gorda' 'Farmington' 'Omaha'
'Macon' 'Ardmore' 'Flagstaff' 'Poplar Bluff' 'Lake Havasu City' 'Calhoun'
'Pocatello' 'Iowa City' 'Texarkana' 'Winona' 'Manchester' 'Minot'
'Bremerton' 'Plattsburgh' 'Bismarck' 'Great Falls' 'Yuba City'
'Idaho Falls' 'Grants Pass' 'Syracuse' 'Evansville' 'Alexandria'
'Marquette' 'Daphne' 'Valdosta' 'Topeka' 'Ames' 'Las Cruces' 'Searcy'
'Missoula' 'Ashtabula' 'Altoona' 'Terre Haute' 'Dover' 'Waco'
'New Castle' 'Ocala' 'Murray' 'Helena' 'Martinsville' 'Lumberton'
'Bemidji' 'Quincy' 'Galesburg' 'Spartanburg' 'Duluth' 'Racine' 'Reading'
'Lufkin' 'Pullman' 'Merced' 'Beckley' 'Sedalia' 'Williston' 'Natchez'
'Morristown' 'Muncie' 'Mountain Home' 'McMinnville' 'Ellensburg'
'Mason City' 'Fort Smith' 'Kankakee' 'Brainerd' 'Shelbyville' 'Concord'
'Ruston' 'Wausau' 'Paducah' 'Vero Beach' 'South Bend' 'Stillwater'
'Sevierville' 'Rockford' 'Durango' 'Rolla' 'Lawrence' 'Rocky Mount'
'Fremont' 'Pueblo' 'Ukiah' 'Greensboro' 'Ada' 'Hammond' 'Truckee'
'Garden City' 'Dickinson' 'North Platte' 'Coos Bay' 'Enid' 'Muskogee'
'Norfolk' 'Muscatine' 'Saginaw' 'Ottumwa' 'Seymour' 'Harrison'
'Marshalltown' 'Abilene' 'Klamath Falls' 'Parkersburg' 'Bardstown'
'Rio Grande City' 'St. Cloud' 'Fort Dodge' 'Stephenville' 'Mount Vernon'
'Erie' 'Emporia' 'Sidney' 'Madera' 'Winchester' 'Roseburg'
'Grand Junction' 'Wisconsin Rapids-Marshfield' 'Tahlequah' 'Chambersburg'
'Owatonna' 'Meadville' 'Bedford' 'Rexburg' 'Georgetown' 'Stevens Point'
'Moscow' 'Branson' 'Defiance' 'Corinth' 'Shelby' 'Sebring' 'LaGrange'
'Kingsville' 'Greeneville' 'Tiffin' 'Sumter' 'Picayune' 'Sheridan'
'Glens Falls' 'Bartlesville' 'Orangeburg' 'Marietta' 'Madisonville'
'Traverse City' 'Battle Creek' 'East Stroudsburg' 'Tupelo' 'Brunswick'
'Junction City' 'Bloomsburg' 'Natchitoches' 'Crawfordsville' 'Faribault'
'Duncan' 'Cortland' 'Cape Cod' 'Logansport' 'Bay City' 'Greenwood'
'Casper' 'Payson' 'Clinton' 'Warrensburg' 'Sioux City' 'Brenham'
'Ocean City' 'Gettysburg' 'McAlester' 'Huntington' 'Dublin' 'Dyersburg'
'Corsicana' 'Easton' 'Tullahoma' 'Rock Springs' 'Grand Island'
'Torrington' 'Dodge City' 'De Ridder' 'Keene' 'Fredericksburg' 'Ottawa'
'Hays' 'West Plains' 'Vincennes' 'Okeechobee' 'Tifton' 'Pottsville'
'Coldwater' 'Centralia' 'Juneau' 'Mansfield' 'Thomaston' 'Pittsburg'
'Batesville' 'Vernal' 'Brownwood' 'Fort Polk South' 'Hermiston-Pendleton'
'Dunn' 'Morehead City' 'Weirton' 'Beaver Dam' 'Hutchinson' 'Palatka'
'Alice' 'Willmar' 'Eureka' 'Seneca' 'Lawrenceburg' 'Peru' 'Carson City'
'California-Lexington Park' 'Thomasville' 'Newport' 'Lima' 'Connersville'
'Montrose' 'Fairbanks' 'Toccoa' 'Brookhaven' 'Mount Pleasant' 'Durant'
'Bastrop' 'Cedartown' 'Elko' 'Jefferson' 'Cadillac' 'Hannibal'
'Washington Court House' 'Blackfoot' 'Brevard' 'Gillette' 'St. Joseph'
'Sterling' 'Lewisburg' 'The Dalles' 'Wichita Falls' 'Paris' 'Baraboo'
'Henderson' 'Johnstown' 'Prineville' 'Cambridge' 'Plymouth' 'Morgan City'
'Pinehurst-Southern Pines' 'Greensburg' 'Plainview' 'Aberdeen' 'Elmira'
'Jasper' 'Sault Ste. Marie' 'Port Clinton' 'Shawnee' 'Sandpoint'
'Arcadia' 'Ludington' 'Corning' 'Whitewater' 'Warsaw' 'DuBois'
'Forest City' 'McComb' 'Deming' 'Newton' 'Pahrump' 'Moultrie'
'Russellville' 'St. Marys' 'Effingham' 'Macomb' 'Shreveport'
```

```
'Homosassa Springs' 'Coshocton' 'Laramie' 'Fernley' 'Summit Park'
'Nacogdoches' 'Altus' 'Show Low' 'Cedar City' 'Olean' 'Beeville'
'Crossville' 'Laconia' 'Yankton' 'Bogalusa' 'Oneonta' 'Bellefontaine'
'Newberry' 'Waycross' 'Red Wing' 'Gadsden' 'Riverton' 'Ponca City'
'Pampa' 'Hood River' 'Greenfield Town' 'North Vernon' 'Clarksdale'
'Astoria' 'Opelousas' 'Mineral Wells' 'Hudson' 'Brookings' 'Great Bend'
'Big Rapids' 'Grenada' 'Angola' 'Laurel' 'La Grande' 'Somerset'
'Lake City' 'Uvalde' 'Scottsbluff' 'Ogdensburg' 'Liberal' 'Port Lavaca'
'Miami' 'Holland' 'Wapakoneta' 'Heber' 'Bradford' 'Kapaa' 'Barre'
'Cullowhee' 'Wabash' 'Sunbury' 'Marshall' 'Oil City' 'Levelland'
'Clewiston' 'Union City' 'Gallup' 'Silver City' 'Moberly' 'Martin'
'El Campo' 'McPherson' 'Glenwood Springs' 'Palestine' 'Kill Devil Hills'
'Hereford' 'Susanville' 'Van Wert' 'Ontario' 'Los Alamos' 'Mexico'
'Kendallville' 'Burley' 'Wheeling' 'Woodward' 'Douglas' 'Cornelia'
'Claremont' 'Adrian' 'Huntingdon' 'Sheboygan' 'Dumas' 'Ionia'
'Arkansas City-Winfield' 'Hillsdale' 'Safford' 'Sweetwater' 'Carbondale'
'Clearlake' 'Parsons' 'Coffeyville' 'Ketchikan' 'Platteville' 'Espanola'
'Gardnerville Ranchos' 'Fort Leonard Wood' 'Pontiac' 'Atchison'
'Portsmouth' 'Price' 'Vernon' 'Evanston' 'Hailey' 'Lamesa'
'Steamboat Springs' 'Sayre' 'Grants' 'Vineyard Haven' 'Berlin'
'Bluefield' 'Hinesville' 'Wahpeton' 'Manhattan' 'Fergus Falls' 'Meridian'
'Sikeston' 'Vicksburg' 'Albemarle' 'Kennett' 'Borger' 'Jesup'
'Alamogordo' 'New Philadelphia' 'Fort Madison' 'Houghton' 'Cullman'
'Nogales' 'Indianola' 'New Ulm']
```

- The appropriate datatype for Zipcodes is typically string or object.Let's convert the data types to string. Zipcodes generally contain 5 digits but in our dataframe some have less than 5 digits.Te columns with 4 didgits will be filled with a zero at the beginning

In [ ]:
```
df.Zipcode = df.Zipcode.astype(str).str.zfill(5)
print(df.dtypes["Zipcode"])
```

```
object
```

- To solve the problems raised under business understanding, two columns will be created.
- Return on Investment (ROI)
- ROI is a measure of returns expected from investments.
- Coefficient of variation (CV)

CV is a measure of the dispersion of data points around the mean and represents the ratio of the standard deviation to the mean. It allows investors to determine how much volatility, or risk, is assumed in comparison to the amount of return expected from investments.

In [ ]:
```
# calculating and creating a new column -ROI

df['ROI'] = (df['2018-04']/ df['1996-04'])-1



#calculating std to be used to find CV
df["std"] = df.loc[:, "1996-04":"2018-04"].std(skipna=True, axis=1)

#calculating mean to be used to find CV
df["mean"] = df.loc[:, "1996-04":"2018-04"].mean(skipna=True, axis=1)

# calculating and creating a new column - CV

df["CV"] = df['std']/df["mean"]

# dropping std and mean as they are not necessary for analysis

df.drop(["std", "mean"], inplace=True, axis=1)
```

In [ ]:
```
df[["Zipcode", "ROI", "CV"]].head()
```

Out[20]:

	Zipcode	ROI	CV
0	60657	2.083782	0.256487
1	75070	0.365295	0.152680
2	77494	0.567966	0.143950
3	60614	1.623971	0.237364
4	79936	0.571798	0.178326

- **Time Series Dataset**

In this section we convert our dataset from wide format to long format .By melting the data, we prepare it for further analysis, visualization, and modeling, facilitating a more effective and efficient workflow.

In [ ]:
```
df1 = df.copy()
```

In [ ]:
```
# creating a function that changes the dataframe structure from wide view to l

def melt_df(data):
    melted = pd.melt(data, id_vars=['RegionID','Zipcode', 'City', 'State', 'Me
                                    'ROI', 'CV' ], var_name='Date')
    melted['Date'] = pd.to_datetime(melted['Date'], infer_datetime_format=True
    melted = melted.dropna(subset=['value'])
    return melted
```

In [ ]:
```
df1 = melt_df(df1)
```

In [ ]:
```
df1.head()
```

Out[24]:

	RegionID	Zipcode	City	State	Metro	CountyName	SizeRank	ROI	CV	Da
0	84654	60657	Chicago	IL	Chicago	Cook	1	2.083782	0.256487	199 04-
1	90668	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	0.365295	0.152680	199 04-
2	91982	77494	Katy	TX	Houston	Harris	3	0.567966	0.143950	199 04-
3	84616	60614	Chicago	IL	Chicago	Cook	4	1.623971	0.237364	199 04-
4	93144	79936	El Paso	TX	El Paso	El Paso	5	0.571798	0.178326	199 04-

In [ ]:
```
df1.dtypes # checking the data types of each column
```

Out[25]:
```
RegionID               int64
Zipcode               object
City                  object
State                 object
Metro                 object
CountyName            object
SizeRank               int64
ROI                  float64
CV                   float64
Date          datetime64[ns]
value                float64
dtype: object
```

In [ ]:
```
df1['Date'] = pd.to_datetime(df1['Date'], format="%y/%m") ## Converting the da
```

In [ ]:
```
df1.duplicated().sum() # checking for duplicates
```

Out[27]:
```
0
```

In [ ]:
```
df1.set_index('Date', inplace=True) # Set the 'Date' column as index
```

In [ ]:
```
df1.head() #preview the first 5 columns of the data set
```

Out[29]:

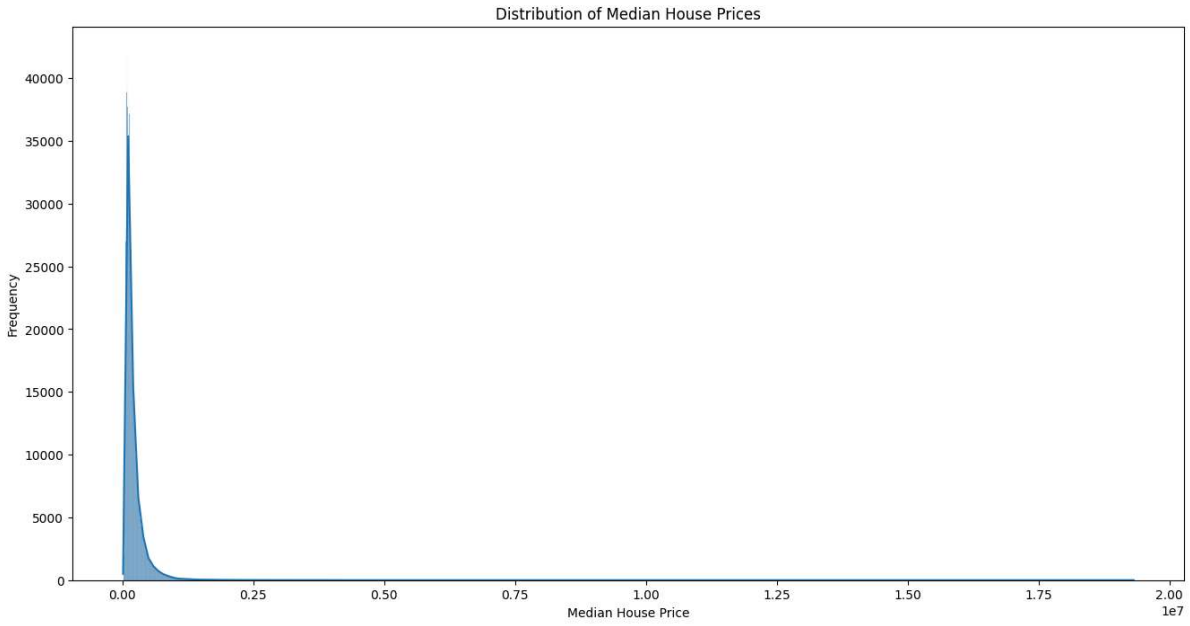
Date	RegionID	Zipcode	City	State	Metro	CountyName	SizeRank	ROI	CV
1996-04-01	84654	60657	Chicago	IL	Chicago	Cook	1	2.083782	0.256487
1996-04-01	90668	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	0.365295	0.152680
1996-04-01	91982	77494	Katy	TX	Houston	Harris	3	0.567966	0.143950
1996-04-01	84616	60614	Chicago	IL	Chicago	Cook	4	1.623971	0.237364
1996-04-01	93144	79936	El Paso	TX	El Paso	El Paso	5	0.571798	0.178326

In [ ]:
```
df1 = df1.rename(columns={'value': 'MedianHousePrice'}) # renaming the column
```

EDA

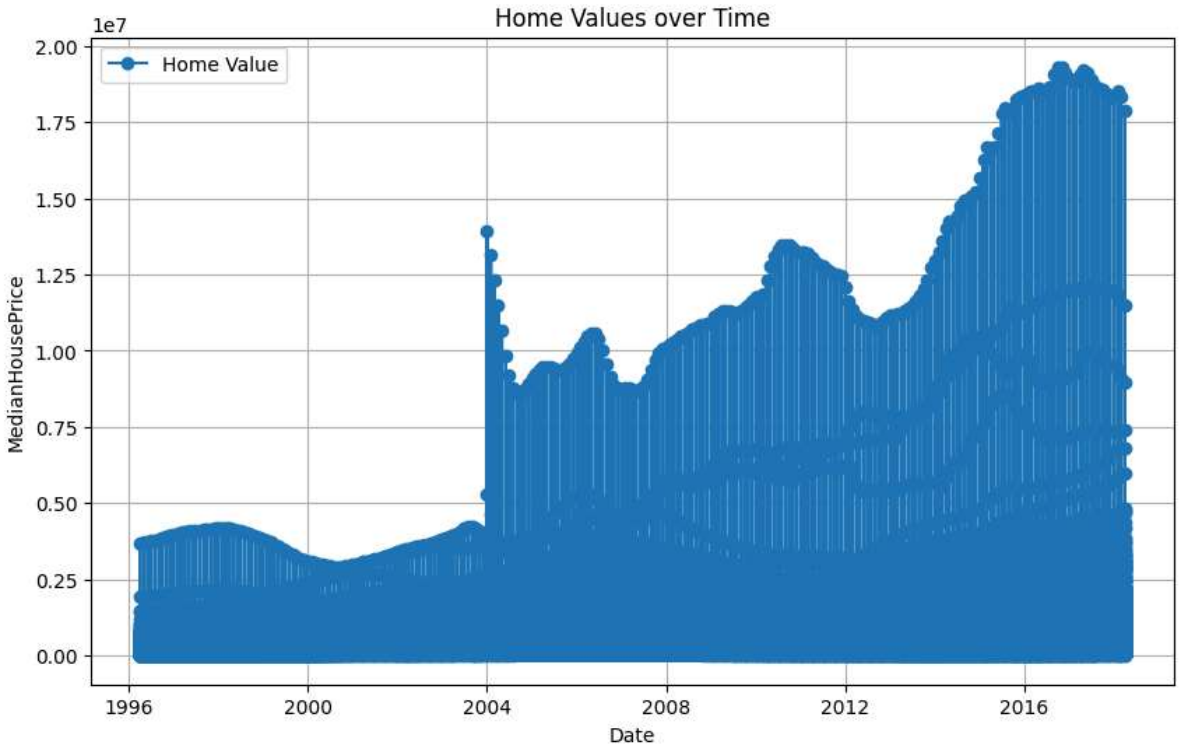
- **Distribution of house prices**

In [ ]:
```
plt.figure(figsize=(16,8))
sns.histplot(df1['MedianHousePrice'], kde=True)
plt.title('Distribution of Median House Prices')
plt.xlabel('Median House Price')
plt.ylabel('Frequency')
plt.show();
```



- Observations:
  The diagram above shows that most houses cost around 100000 dollars.

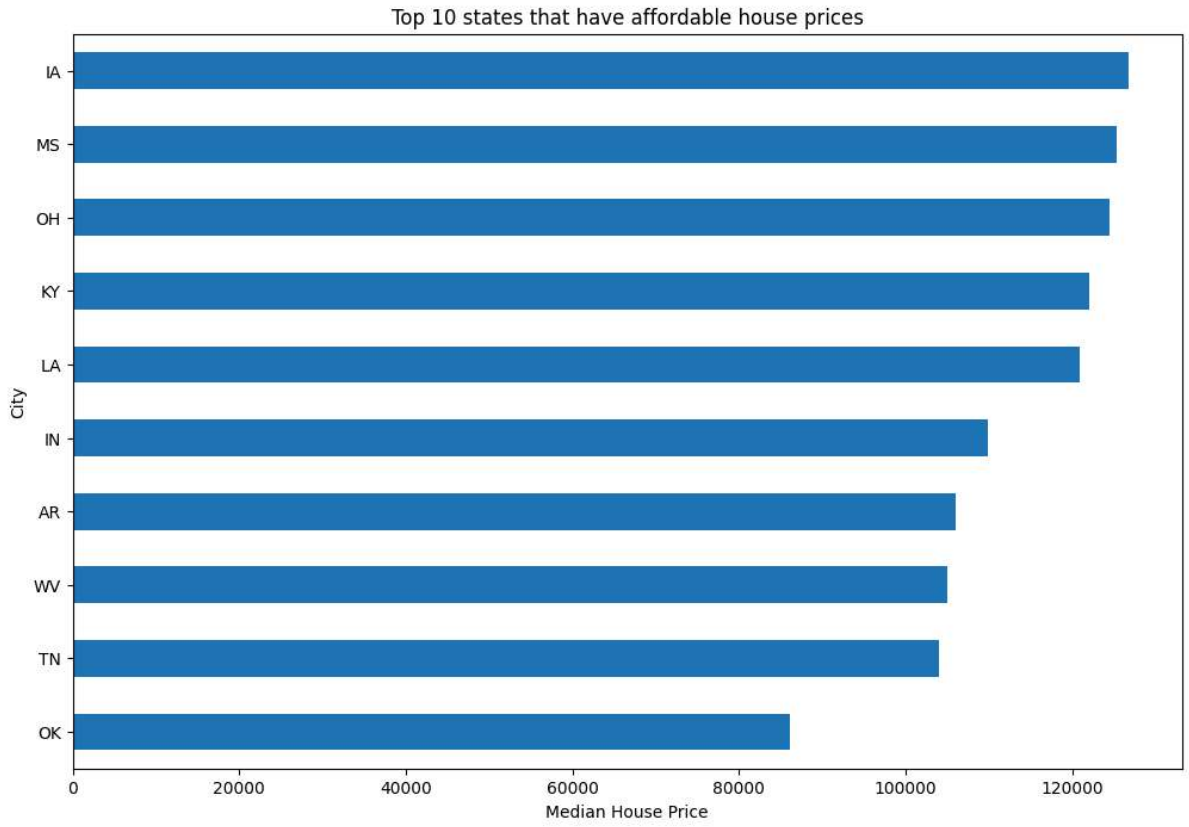
In [ ]:
```
# Plotting with Matplotlib
plt.figure(figsize=(10, 6))
plt.plot(df1.index, df1['MedianHousePrice'], marker='o', linestyle='-', label=
plt.title('Home Values over Time')
plt.xlabel('Date')
plt.ylabel('MedianHousePrice')
plt.legend()
plt.grid(True)
plt.show()
```



- ** Top 10 most affordale states**

In [ ]:

```
affordable_states = df1.groupby('State')['MedianHousePrice'].mean().sort_value

plt.figure(figsize=(12, 8))
affordable_states.plot(kind='barh')
plt.title('Top 10 states that have affordable house prices')
plt.xlabel('Median House Price')
plt.ylabel('City')
plt.show();
```

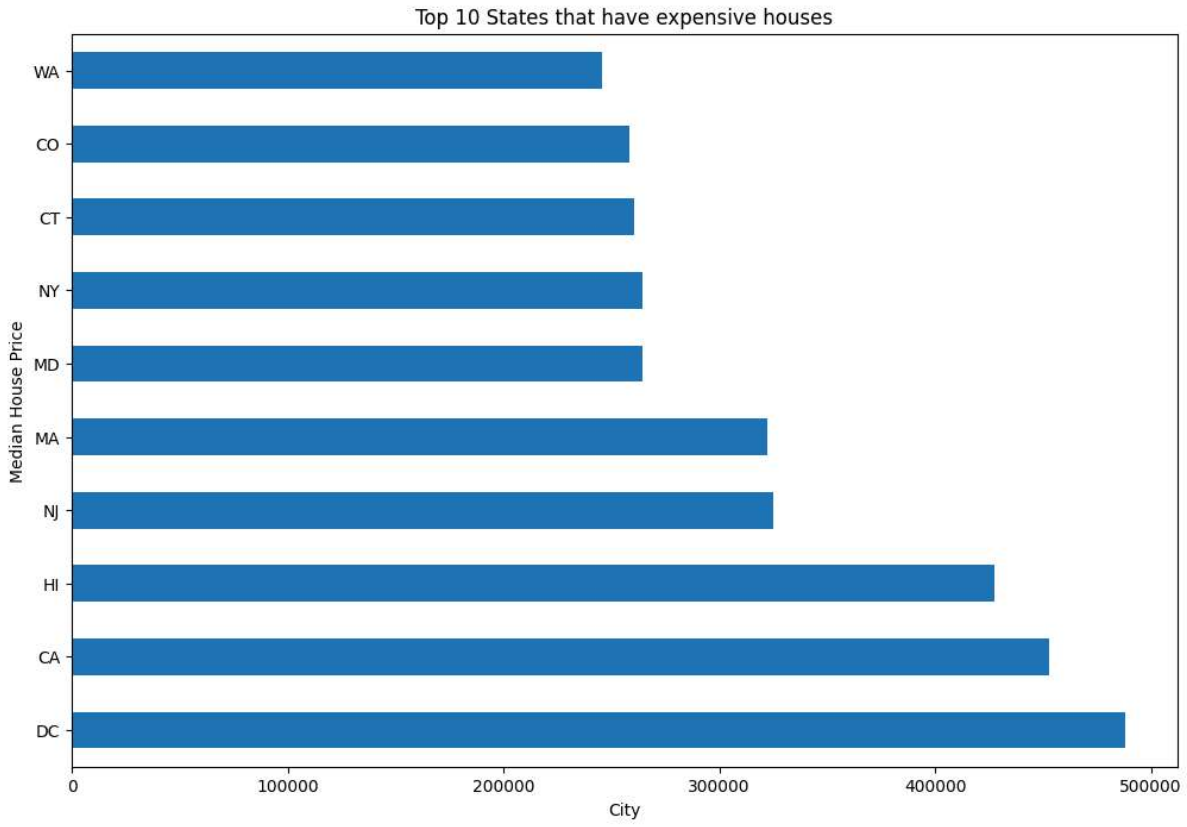


- Observations:
  The following states tend to have affordable houses , 'OK':OKlahoma being the most affordable state in the states others include; West Virginia (WV) Arkansas (AR) Alabama (AL), Indiana (IN), Kentucky (KY), Mississippi (MS), Michigan (MI), Ohio (OH) and Iowa (IA)

- **Top 10 most expsensive states**

In [ ]:

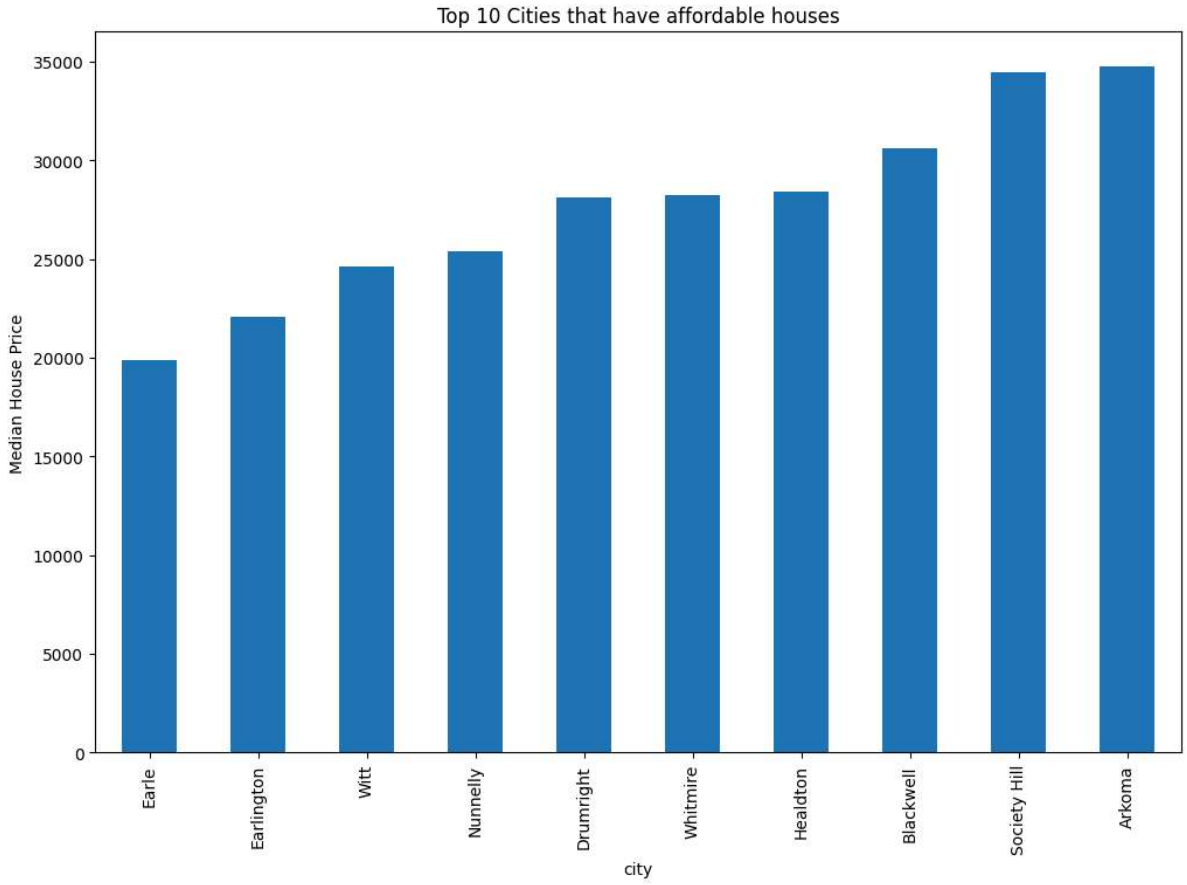
```
expensive_states = df1.groupby('State')['MedianHousePrice'].mean().sort_values

plt.figure(figsize=(12, 8))
expensive_states.plot(kind='barh')
plt.title('Top 10 States that have expensive houses')
plt.xlabel('City')
plt.ylabel('Median House Price')
plt.show()
```



-Observations:
The range of median house prices among these top 10 expensive states spans from about $250,000 to 500,000$. The District of Columbia stands out as the most expensive, while Washington and Colorado are at the lower end of the top 10 expensive states.

- **Top 10 most affordable cities**

In [ ]:
```
cheapest_cities = df1.groupby('City')['MedianHousePrice'].mean().sort_values(a

plt.figure(figsize=(12, 8))
cheapest_cities.plot(kind='bar')
plt.title('Top 10 Cities that have affordable houses')
plt.xlabel('city')
plt.ylabel('Median House Price')
plt.show();
```

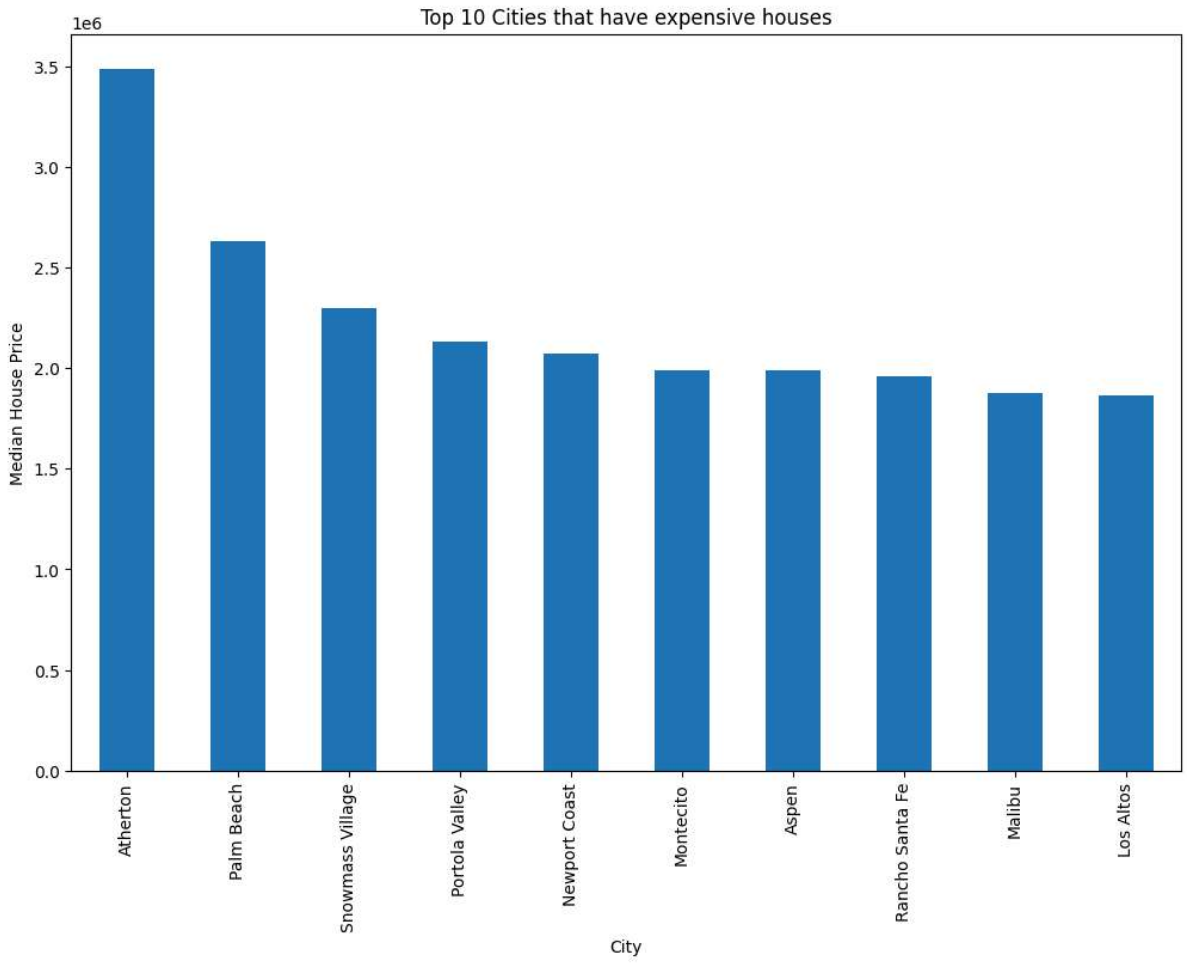


- Observations:
  The range of median house prices among these top 10 affordable cities spans from about $21,000 to 35,000$. These cities represent areas where housing is significantly more affordable compared to the national average and especially when compared to the most expensive cities.
- Arkoma has the highest median house price among the affordable cities, slightly above $35,000.

- **Top 10 cities that have the most expensive houses**

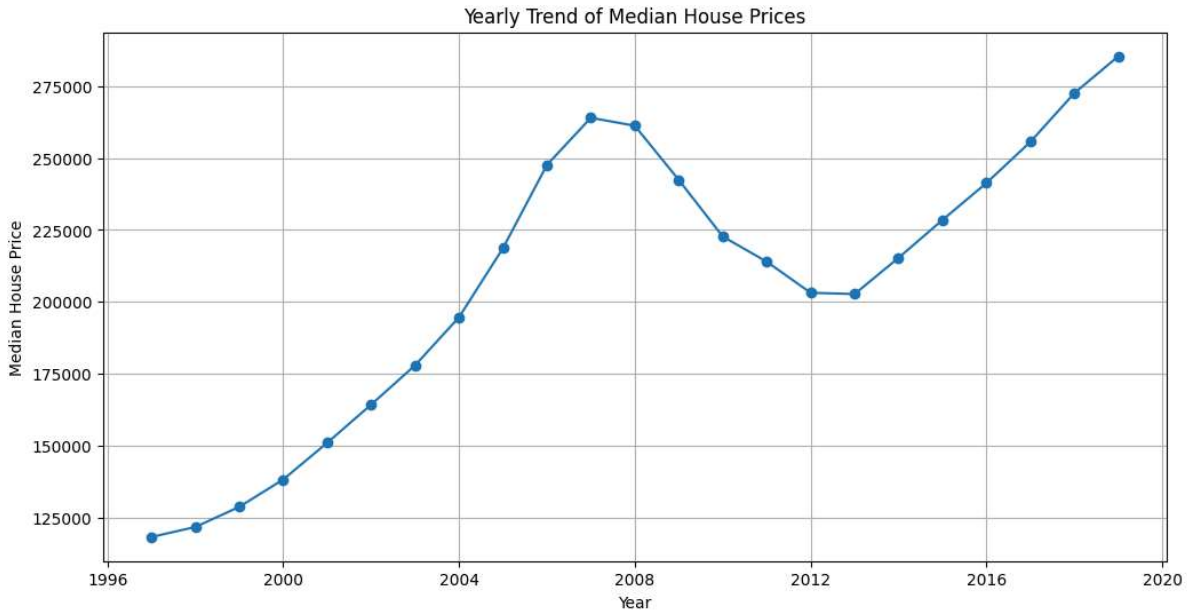
In [ ]:
```
expensive_cities = df1.groupby('City')['MedianHousePrice'].mean().sort_values(
print(expensive_cities)
plt.figure(figsize=(12, 8))
expensive_cities.plot(kind='bar')
plt.title('Top 10 Cities that have expensive houses')
plt.xlabel('City')
plt.ylabel('Median House Price')
plt.show() ;
```

```
City
Atherton            3.487129e+06
Palm Beach          2.634498e+06
Snowmass Village    2.300179e+06
Portola Valley      2.131495e+06
Newport Coast       2.070006e+06
Montecito           1.991682e+06
Aspen               1.988169e+06
Rancho Santa Fe     1.959659e+06
Malibu              1.876195e+06
Los Altos           1.866172e+06
Name: MedianHousePrice, dtype: float64
```



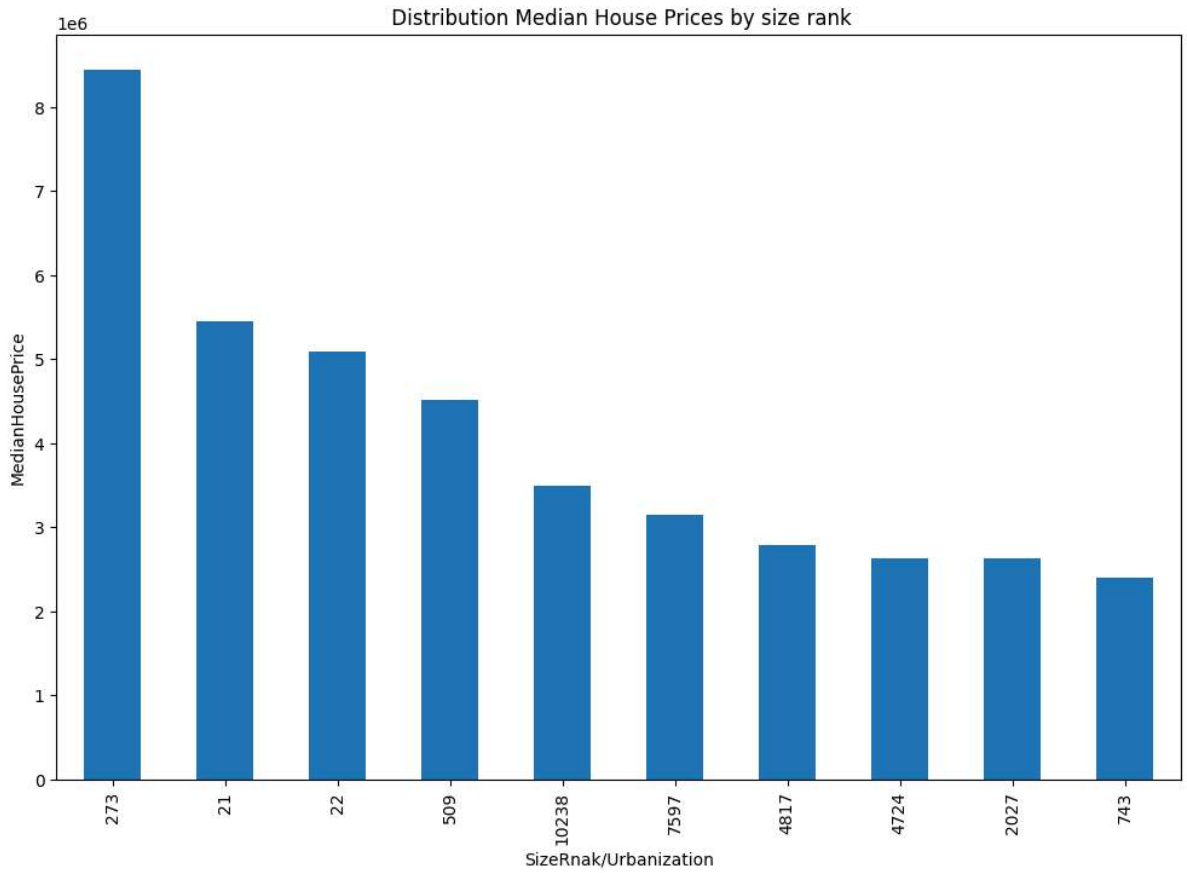
- Observations:
  There is a clear trend of high median house prices in these cities, with Atherton standing out as the most expensive. The range of median house prices among these top 10 cities spans from about $1.9 million to 3.5$ million. s

In [ ]:
```
# Resample the data by year and calculate the mean median house price
yearly_data = df1['MedianHousePrice'].resample('A').mean()

# Plot the resampled data
plt.figure(figsize=(12, 6))
plt.plot(yearly_data, marker='o')
plt.title('Yearly Trend of Median House Prices')
plt.xlabel('Year')
plt.ylabel('Median House Price')
plt.grid(True)
plt.show()
```



- Observations:Initial Growth (1996-2007):
- There is a steady increase in median house prices from 1996 to around 2007. The growth accelerates significantly from around 2000 to 2007.
- Following the peak, there is a notable decline in prices, bottoming out around 2012.
- From 2012 onwards, there is a consistent and steady increase in house prices up to 2020

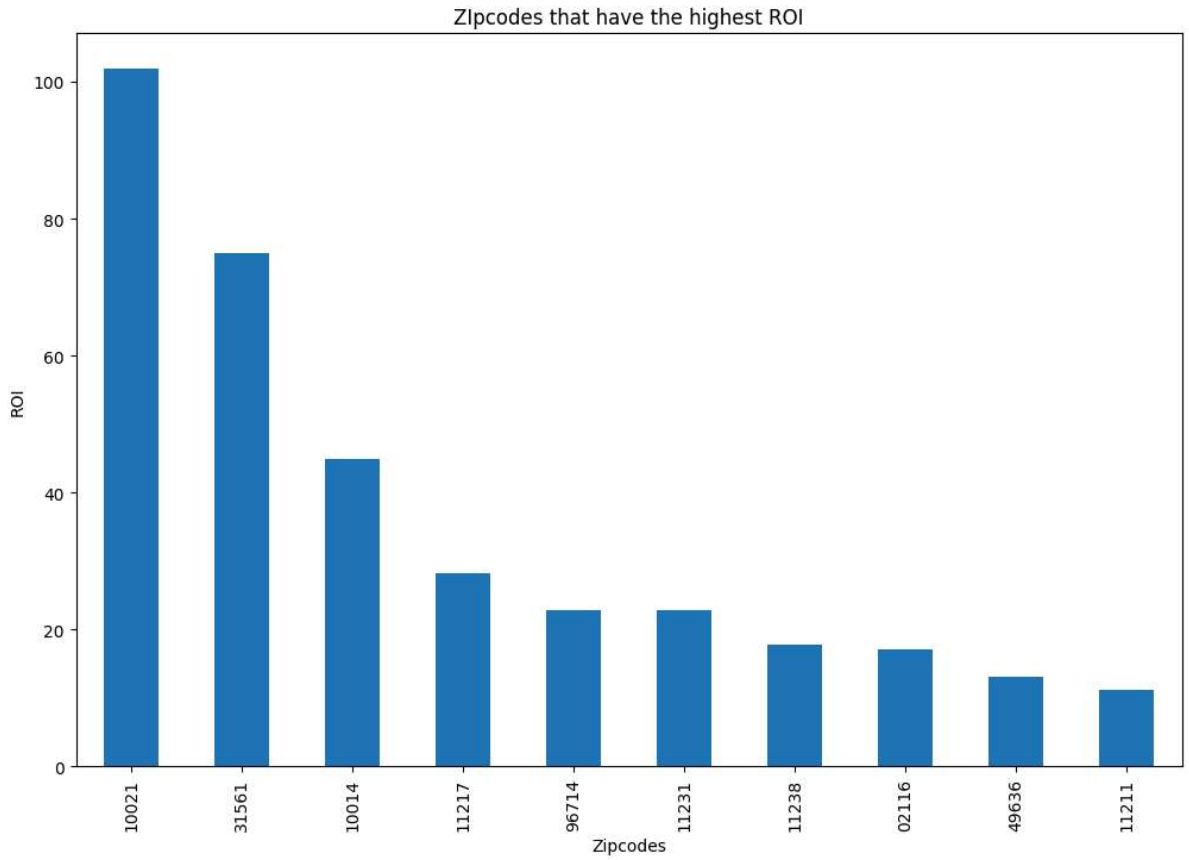
In [ ]:
```
# Does sizerank affect the House prices
urb = df1.groupby('SizeRank')['MedianHousePrice'].mean().sort_values(ascending

plt.figure(figsize=(12, 8))
urb.plot(kind='bar')
plt.title(' Distribution Median House Prices by size rank')
plt.xlabel('SizeRnak/Urbanization')
plt.ylabel('MedianHousePrice')
plt.show() ;
```



- Observations: Urbanisation has no effect on the house prices .

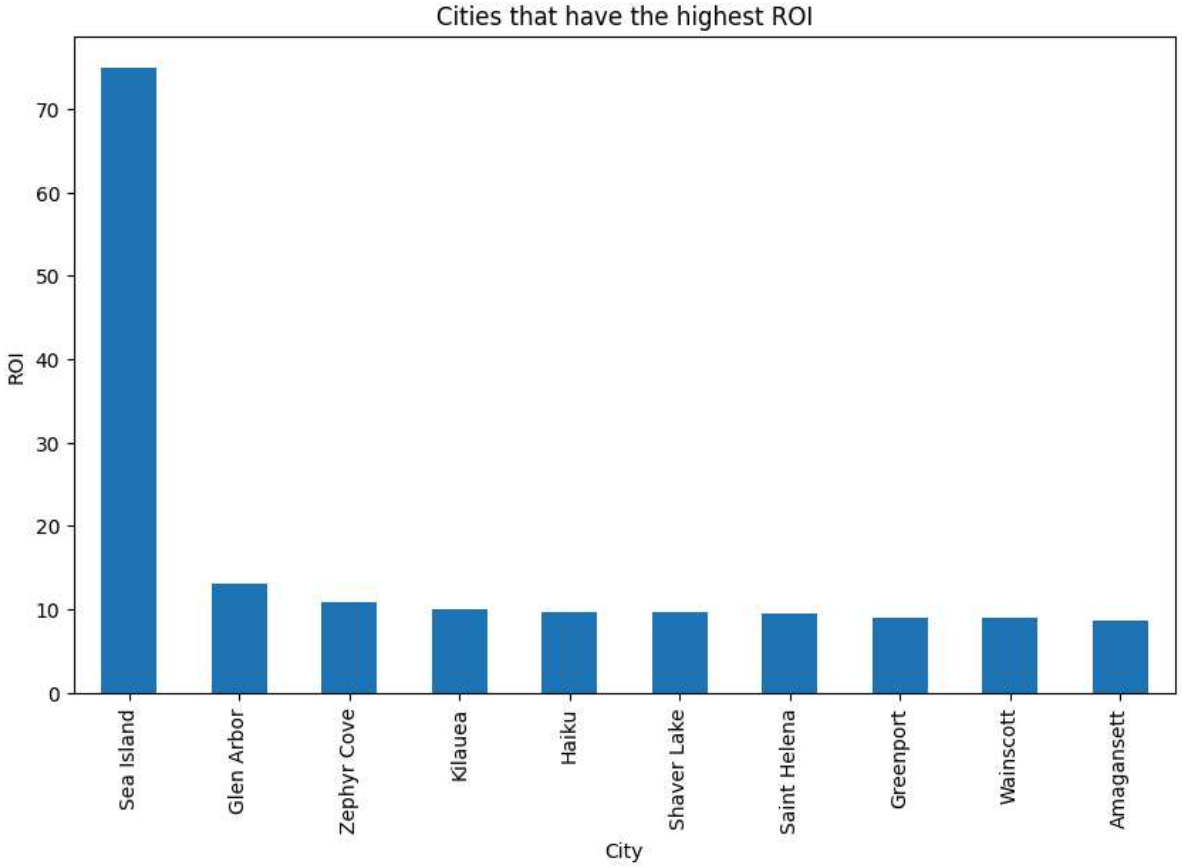
- **Top 10 Zipcodes that have the highest ROI**

In [ ]:
```
 ##What top 5 Zipcodes have the highest ROI?
zipcodes = df1.groupby('Zipcode')['ROI'].mean().sort_values(ascending=False).h

plt.figure(figsize=(12, 8))
zipcodes.plot(kind='bar')
plt.title('ZIpcodes that have the highest ROI')
plt.xlabel('Zipcodes')
plt.ylabel('ROI')
plt.show() ;
```



Top 10 cities with the highest ROI

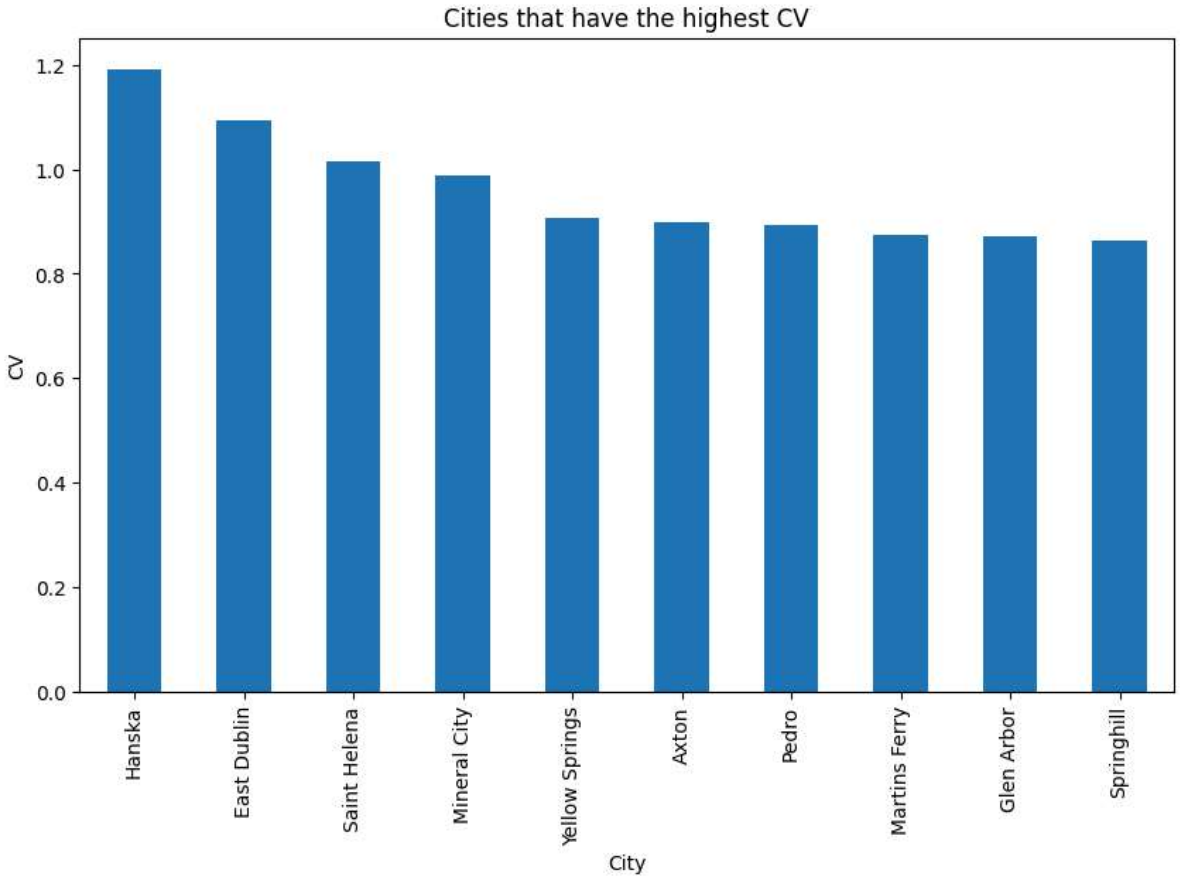
In [ ]:
```
 #Top 10 Cities  that have the highest ROI?
Cities= df1.groupby('City')['ROI'].mean().sort_values(ascending=False).head(10

plt.figure(figsize=(10, 6))
Cities.plot(kind='bar')
plt.title('Cities that have the highest ROI')
plt.xlabel('City')
plt.ylabel('ROI')
plt.show() ;
```



- The cities with the highest ROI(return on investement) are; Sea Island,Glen Arbor,Zephyr Cove,Kilauea,Haiku,Shaver Lake,Sint Helena,Green port,Wainscott and Amagansett.
- This information is valuable for real estate investors, developers, and policymakers seeking to make informed decisions and capitalize on growth opportunities in the housing market.

Top 10 Cities with the highest Risk factor(CV)

In [ ]:
```
 #Top 5 Cities  that have the highest CV?
Cities = df1.groupby('City')['CV'].mean().sort_values(ascending=False).head(10

plt.figure(figsize=(10, 6))
Cities.plot(kind='bar')
plt.title('Cities that have the highest CV')
plt.xlabel('City')
plt.ylabel('CV')
plt.show() ;
```



A high CV means that the price deviates further from the mean and that there is presence of high price volatility making it a problem to effectively predict their future median house prices.This means that real estate investors cannot invest in these areas.
The cities with the highest price volatility are:

- Hanska
- East Dublin
- Saint Helena
- Mineral City
- Yellow springs
- Axton
- Pedro
- Martins Ferry
- Glen Arbor
- Spring Hill.

In [ ]:
```
# zipcodes = df1.groupby('Zipcode')['ROI'].mean().sort_values(ascending=False)
top_zipcodes_df = zipcodes.reset_index()

# If 'Date' is already the index, reset it
if df1.index.name == 'Date':
    df1_reset = df1.reset_index()
else:
    df1_reset = df1

# Filter the original DataFrame to include only the top 5 zip codes
filtered_df = df1_reset[df1_reset['Zipcode'].isin(top_zipcodes_df['Zipcode'])]

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

# Print the resulting DataFrame
filtered_df
```

Out[42]:

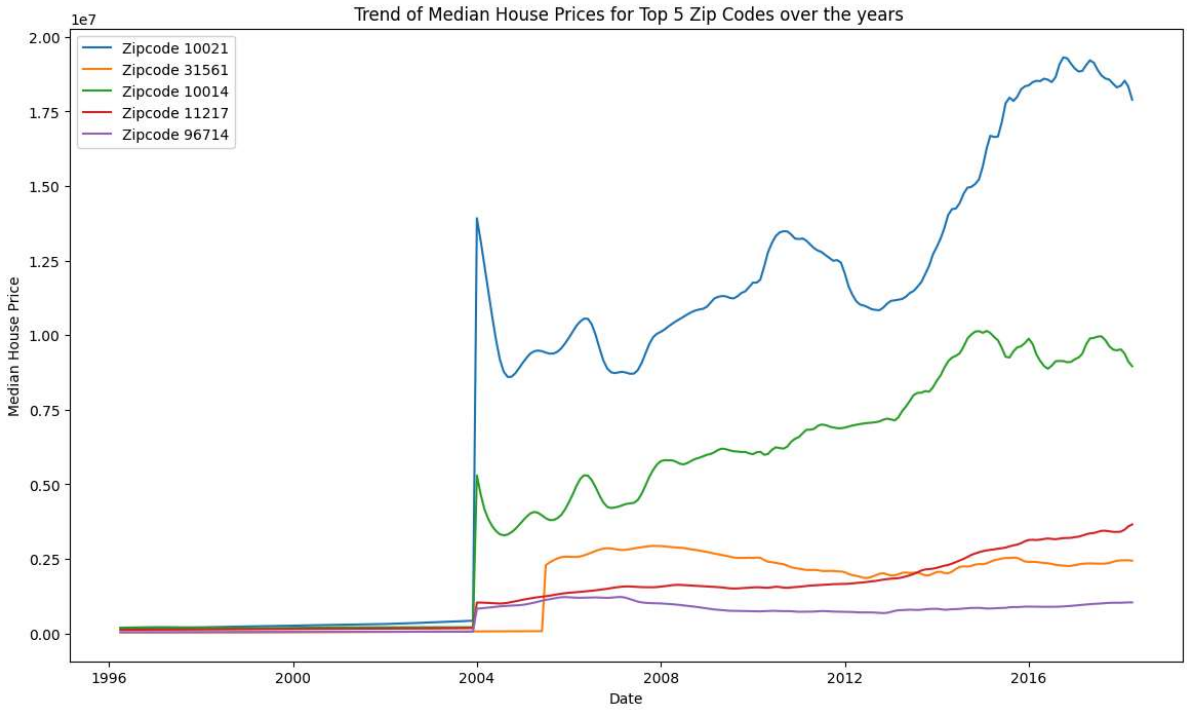
Date	RegionID	Zipcode	City	State	Metro	CountyName	SizeRank	ROI	CV
1996-04-01	62022	11211	New York	NY	New York	Kings	118	11.189940	0.66380
1996-04-01	62048	11238	New York	NY	New York	Kings	157	17.827406	0.74294
1996-04-01	61635	10021	New York	NY	New York	New York	273	101.962601	0.78180
1996-04-01	61628	10014	New York	NY	New York	New York	509	44.968702	0.79540
1996-04-01	62028	11217	New York	NY	New York	Kings	1535	28.154705	0.79313
...	...	...	...	...	...	...	...	...	.
2018-04-01	62041	11231	New York	NY	New York	New York	1996	22.794451	0.75974
2018-04-01	58630	02116	Boston	MA	Boston	Suffolk	3331	17.152082	1.30392
2018-04-01	98855	96714	Kilauea	HI	Kapaa	Kauai	12848	22.864834	0.70383
2018-04-01	79832	49636	Glen Arbor	MI	Traverse City	Leelanau	14342	13.112311	0.87297
2018-04-01	71578	31561	Sea Island	GA	Brunswick	Glynn	14623	74.953307	0.83653

2650 rows × 10 columns

In [ ]:
```
# Print the resulting DataFrame
# filtered_df.Zipcode.nunique()
```

In [ ]:

```
# Group by 'Zipcode' to analyze each zip code separately
grouped = df1.groupby('Zipcode')

# Calculate mean ROI and mean CV for each zip code
mean_roi = grouped['ROI'].mean()
mean_cv = grouped['CV'].mean()

# Analyze the top 5 zip codes based on mean ROI and low risk (mean CV)
# You may need to adjust the criteria based on your specific needs
top_zipcodes = mean_roi.nlargest(5).index
top_zipcodes

# Filter the DataFrame to include only the top zip codes
top_df = df1[df1['Zipcode'].isin(top_zipcodes)]

# Plotting the time series of median house prices for the top zip codes
plt.figure(figsize=(14, 8))
for zipcode in top_zipcodes:
    plt.plot(top_df[top_df['Zipcode'] == zipcode]['MedianHousePrice'], label=f

plt.title(' Trend of Median House Prices for Top 5 Zip Codes over the years')
plt.xlabel('Date')
plt.ylabel('Median House Price')
plt.legend()
plt.show()
```



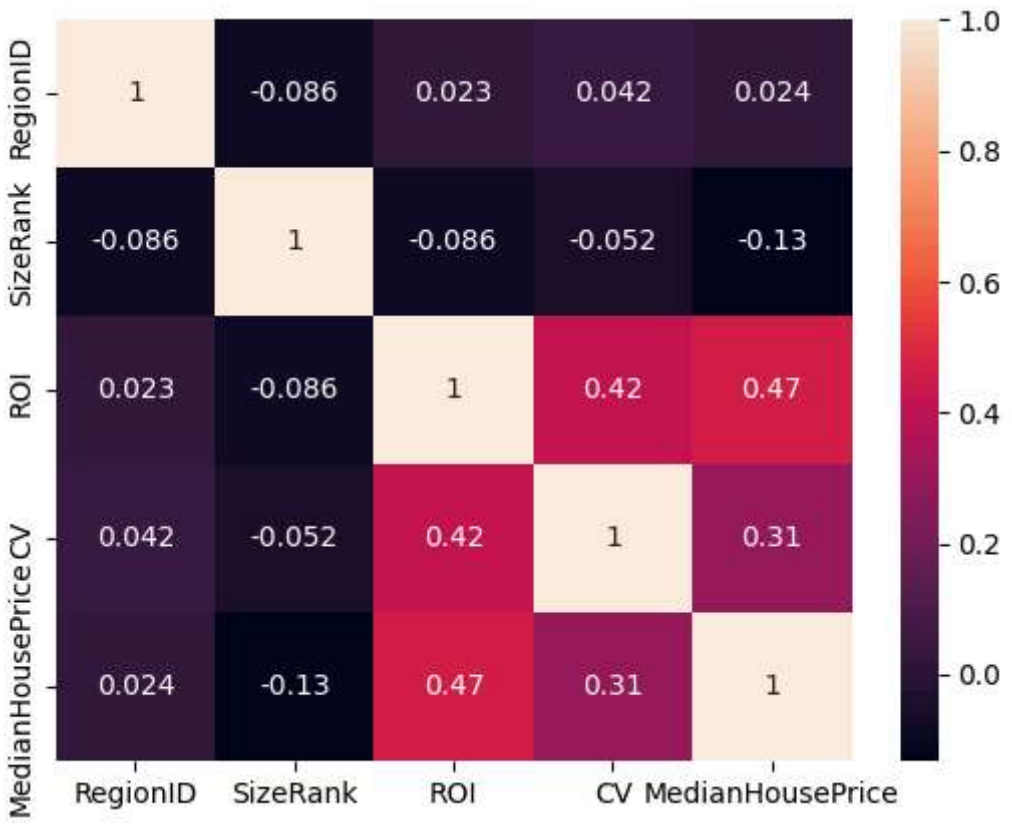
observations:

The Average house prices keep on increasing however the prices remain constant between 1996 and 2004

- **Checking the correlation of the dataset in the numeric columns**

In [ ]:
```
numeric_columns = df1.select_dtypes(include=['int64', 'float64'])
corr = numeric_columns.corr()
sns.heatmap(corr, annot=True)
# df1.ROI.dtype
```

Out[45]: <Axes: >



- The dataset does not exhibit multicollinearity

Time Series Ananlysis

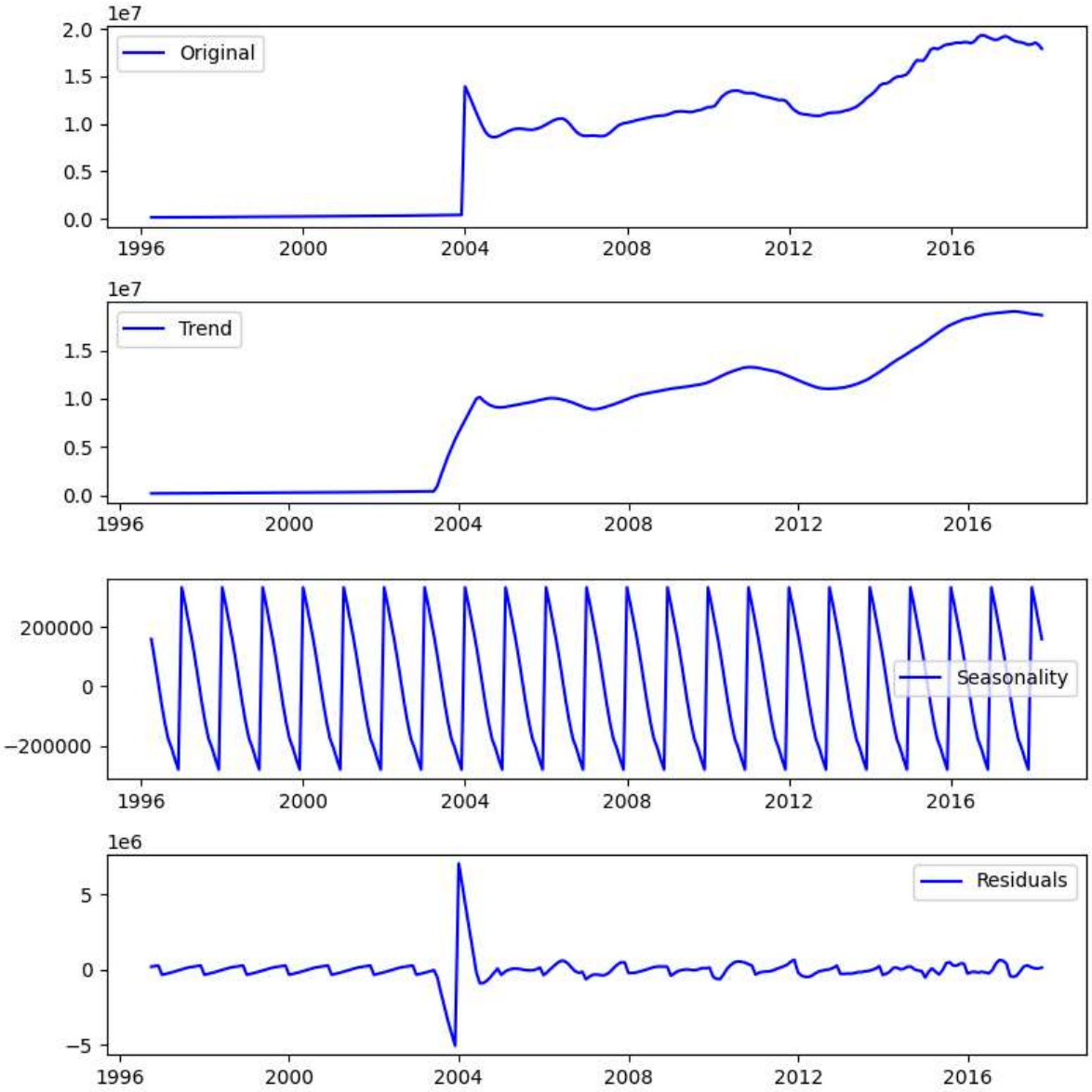
- We will focus on one zipcode among the ones that have highest ROI in our TSA

In [ ]:
```
# We will be using one of those zipcodes for modelling
z_df = filtered_df[filtered_df['Zipcode'] =='10021']
my_series =z_df['MedianHousePrice']
my_series[:10]
```

Out[46]:
```
Date
1996-04-01    173800.0
1996-05-01    173850.0
1996-06-01    173850.0
1996-07-01    173850.0
1996-08-01    173900.0
1996-09-01    174100.0
1996-10-01    174450.0
1996-11-01    175050.0
1996-12-01    176000.0
1997-01-01    177250.0
Name: MedianHousePrice, dtype: float64
```

- Time series decomposition enables us to visualize;
- Original: The original time series data.
- Trend: The long-term, overall direction of the data.
- Seasonality: The cyclical pattern that repeats over a fixed period, such as a year or a month.
- Residuals: The remaining variation in the data after the trend and seasonality have been removed. This represents the random noise in the data.

In [ ]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
def seasonal_decomposition(df):
    decomposition = seasonal_decompose(df)

    # Gather the trend, seasonality, and residuals
    trend = decomposition.trend
    seasonal = decomposition.seasonal
    residual = decomposition.resid

# Plot gathered statistics
    plt.figure(figsize=(8,8))
    plt.subplot(411)
    plt.plot(df, label='Original', color='blue')
    #plt.plot(ts, label='Original', color='blue')
    plt.legend(loc='best')
    plt.subplot(412)
    plt.plot(trend, label='Trend', color='blue')
    plt.legend(loc='best')
    plt.subplot(413)
    plt.plot(seasonal,label='Seasonality', color='blue')
    plt.legend(loc='best')
    plt.subplot(414)
    plt.plot(residual, label='Residuals', color='blue')
    plt.legend(loc='best')
    plt.tight_layout()

seasonal_decomposition(my_series)
```

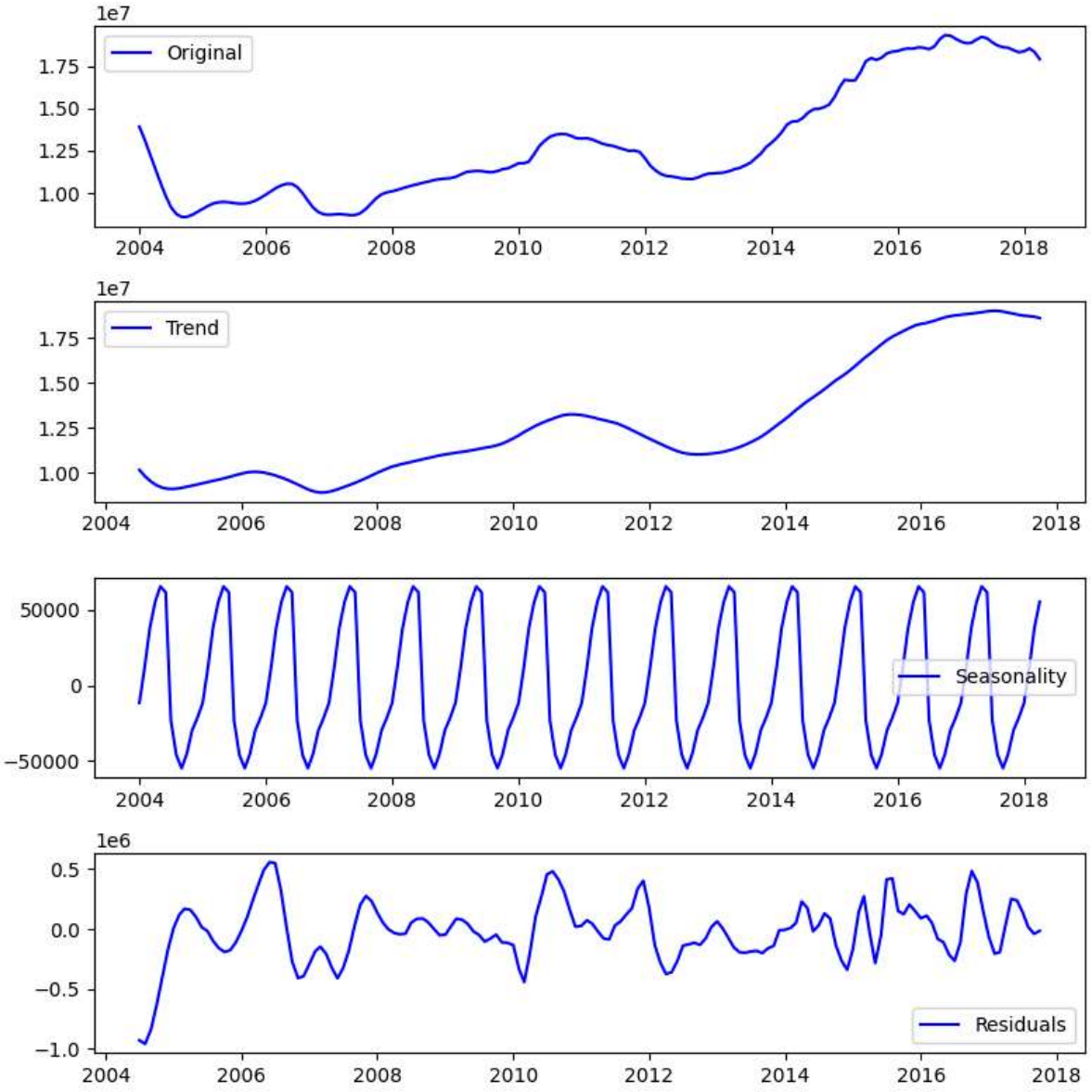


- The trend between 199 and 2004 is quite consistent whish is abnormal therefore we will focus from the year 2004 to 2018 for our analysis.

In [ ]:
```
# remove data from 1996-2004
ts = my_series['2004':]
ts.head()
```

Out[48]:
```
Date
2004-01-01    13922800.0
2004-02-01    13140500.0
2004-03-01    12333800.0
2004-04-01    11490700.0
2004-05-01    10641200.0
Name: MedianHousePrice, dtype: float64
```

In [ ]:
```
seasonal_decomposition(ts)
```



- Now this gives a better view of our series components, with seasonality, trend and residual patterns clearly visible.

Stationarity Check

- In time series analysis. The assumption made is that the data is stationary.
- Stationarity basically means that the statistical properties of the data don't change over time. This includes things like the mean, variance, and how observations are correlated with each other at different time lags.

- We use the Dickey-Fuller test to check stationarity of our series.If the p-value is less than the chosen significance level (e.g., 0.05), we reject the null hypothesis and conclude that the time series is stationary.

In [ ]:
```
def stationarity_check(df):
    dftest = adfuller(df)
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic', 'p-value',
                                             '#Lags Used', 'Number of Observat
    print(dfoutput)
stationarity_check(ts)
```

```
Test Statistic                  -1.488510
p-value                          0.539187
#Lags Used                      14.000000
Number of Observations Used    157.000000
dtype: float64
```

- The p_value is greater than 0.05, and the test_statistic is also greater than the critical values,therefore we confirm that the time series is not stationary
- Differencing is one of the techniques used to make data stationary.

In [ ]:
```
ts =  ts.diff(1).diff(12).dropna()
stationarity_check(ts)
# stationarity_check(ts)
```

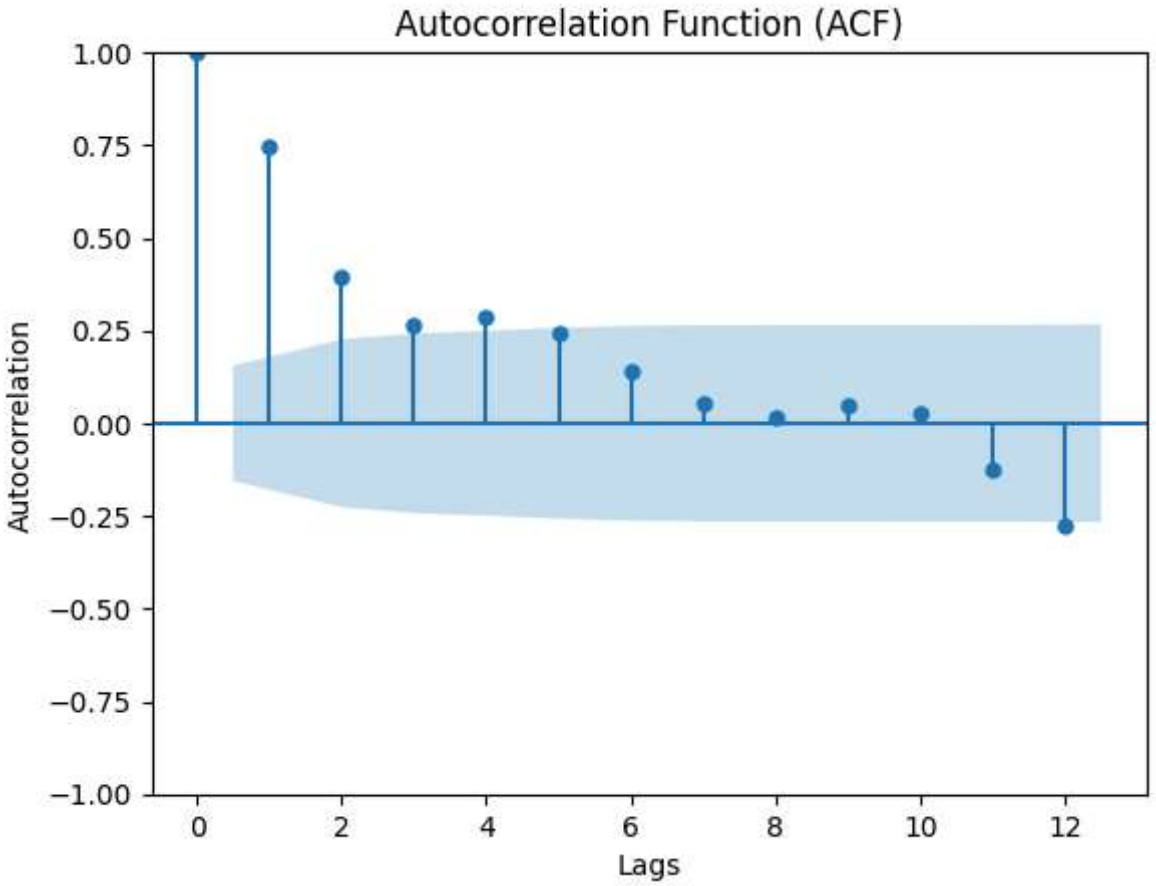
```
Test Statistic                  -3.064492
p-value                          0.029295
#Lags Used                      12.000000
Number of Observations Used    146.000000
dtype: float64
```

- The p_value is less than 0.05 and also the test statistic is less than all of the critical values. The series has been made stationary

- Next step is to plot ta autocorrelationa and partial autocorrelation plots to determine our q and p values while using the ARIMA model.

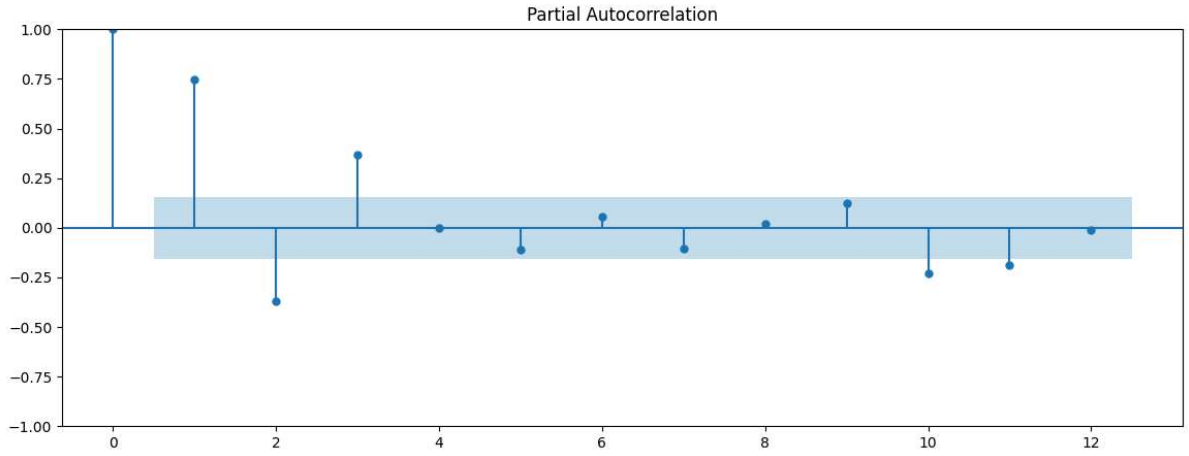
In [ ]:
```
# Plotting only one graph for ACF
plt.figure(figsize=(10, 4))
plot_acf(ts, lags=12)
plt.xlabel("Lags")
plt.ylabel("Autocorrelation")
plt.title("Autocorrelation Function (ACF)")
plt.show();
```

```
<Figure size 1000x400 with 0 Axes>
```



The value of q=4.

In [ ]:
```
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 14, 5
plot_pacf(ts, lags=12, method='ywm');
```



- p=2

- **Baseline Model**
- We will use the ARIMA model as our baseline model.

- The train-test split for a time series is a little different from what we are used to. Because chronological order matters, we cannot randomly sample points in our data. Instead, we cut off a portion of our data at the end, and reserve it as our test set.

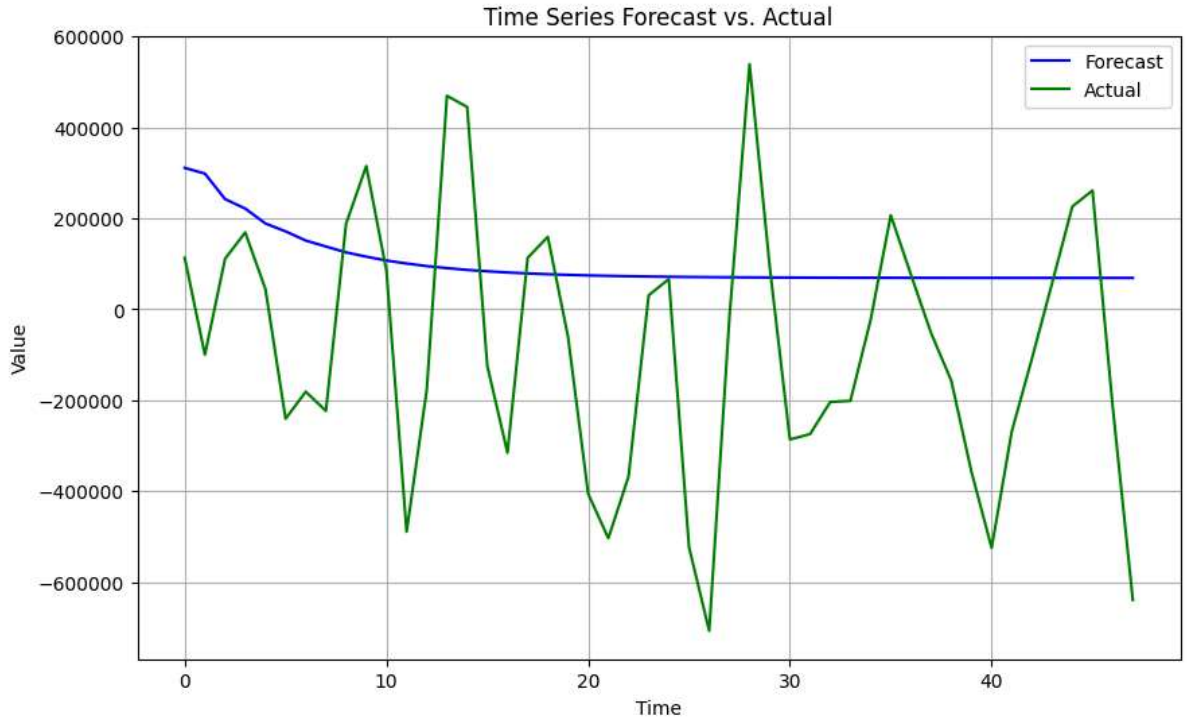
In [ ]:
```
# Define train and test sets according to the index found above
# find the index which allows us to split off 20% of the data
cutoff = round(ts.shape[0]*0.7)
cutoff
train = ts[:cutoff]

test = ts[cutoff:]
```

In [ ]:
```
# Creating ARIMA Model
from statsmodels.tsa.arima.model import ARIMA
import warnings
warnings.filterwarnings('ignore')
# Defining the ARIMA parameters
p = 2 #  Autoregressive parameter
d = 0  # Differencing parameter
q = 4# Moving average parameter

# Creating the ARIMA model
arima_model = ARIMA(train, order=(p, d, q))
arima_result = arima_model.fit()

# Printing the summary of the ARIMA model
print(arima_result.summary())
```

```
                                SARIMAX Results
==========================================================================
=
Dep. Variable:       MedianHousePrice   No. Observations:                   11
1
Model:                 ARIMA(2, 0, 4)   Log Likelihood               -1429.38
2
Date:                Mon, 01 Jul 2024   AIC                            2874.76
3
Time:                        12:28:37   BIC                            2896.44
0
Sample:                    02-01-2005   HQIC                           2883.55
7
                         - 04-01-2014
Covariance Type:                 opg
==========================================================================
=
                 coef    std err          z      P>|z|      [0.025      0.97
5]
--------------------------------------------------------------------------
-
const       6.898e+04   1.22e+05      0.567      0.571   -1.69e+05    3.07e+0
5
ar.L1          0.2062      0.265      0.777      0.437      -0.314       0.72
6
ar.L2          0.5096      0.282      1.809      0.070      -0.043       1.06
2
ma.L1          1.6219      0.298      5.448      0.000       1.038       2.20
5
ma.L2          0.8902      0.397      2.243      0.025       0.112       1.66
8
ma.L3         -0.0082      0.411     -0.020      0.984      -0.813       0.79
7
ma.L4          0.0400      0.311      0.129      0.897      -0.569       0.64
9
sigma2      9.238e+09      0.327   2.82e+10      0.000    9.24e+09    9.24e+0
9
==========================================================================
======
Ljung-Box (L1) (Q):                   0.14   Jarque-Bera (JB):
6.82
Prob(Q):                              0.71   Prob(JB):
0.03
Heteroskedasticity (H):               2.23   Skew:
0.14
Prob(H) (two-sided):                  0.02   Kurtosis:
4.18
==========================================================================
======

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (comple
x-step).
[2] Covariance matrix is singular or near-singular, with condition number 1.9
e+31. Standard errors may be unstable.
```

In [ ]:
```
# Forecasting
from sklearn.metrics import mean_squared_error
forecast = arima_result.forecast(steps=len(test))

# Evaluate forecast
mse = mean_squared_error(test, forecast)
rmse = np.sqrt(mse)
print(f'RMSE: {rmse}')

time_index = range(len(test))
# Plot the forecast and actual data
plt.figure(figsize=(10, 6))
plt.plot(time_index, forecast, label='Forecast', color='blue')
plt.plot(time_index, test, label='Actual', color='green')
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Time Series Forecast vs. Actual')
plt.legend()
plt.grid(True)
plt.show()
```

```
RMSE: 334695.9709849217
```



- Forecast (Blue Line): The blue line represents the forecasted values. It appears relatively smooth, suggesting a model's prediction over time.
- Actual (Green Line): The green line represents the actual recorded values. It is highly volatile, with sharp peaks and troughs.
- Problem in the Forecast: The forecast fails to accurately predict extreme variations seen in the actual data. At various points, the forecast underestimates or overestimates the actual values. This discrepancy indicates that the forecasting model may not handle the data's volatility effectively.

In [ ]:
```
#using auto_arima- it does a random search for the best pdq,PDQS
# !pip install pmdarima
import pmdarima as pm
sarima_model = pm.auto_arima(ts,
                             m=12,
                             seasonal=True,
                             start_p=0,
                             start_q=0,
                             start_P=0,
                             start_Q=0,
                             max_order=6,
                             test='adf',
                             error_action='warn',
                             suppress_warnings=True,
                              stepwise=True,
                              trace=False)
```

- The Akaike Information Criterion (AIC) tests the goodness of fit.It rewards models that achieve a high goodness-of-fit with little complexity
- A model that fits the data very well while using lots of features will be assigned a larger AIC score than a model that uses fewer features to achieve the same goodness-of-fit.
- Auto_arima does a random search and comes up with the best parameters(from the given ones) that reduce the AIC.

In [ ]:

```
print(sarima_model.summary())
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                   y   No. Observations:
159
Model:             SARIMAX(5, 0, 4)x(2, 0, [1], 12)   Log Likelihood
-2061.598
Date:                              Mon, 01 Jul 2024   AIC
4149.196
Time:                                     12:34:19   BIC
4189.092
Sample:                                 02-01-2005   HQIC
4165.398
                                      - 04-01-2018
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------
-
ar.L1          0.7412     22.900      0.032      0.974     -44.142      45.62
4
ar.L2          0.0914     10.021      0.009      0.993     -19.549      19.73
2
ar.L3         -0.1136      4.659     -0.024      0.981      -9.245       9.01
8
ar.L4          0.2884      1.099      0.263      0.793      -1.865       2.44
2
ar.L5         -0.0922      6.363     -0.014      0.988     -12.564      12.37
9
ma.L1          1.0601     22.849      0.046      0.963     -43.722      45.84
2
ma.L2          0.0787     31.161      0.003      0.998     -60.996      61.15
3
ma.L3         -0.4518     11.784     -0.038      0.969     -23.548      22.64
4
ma.L4          0.0574      5.963      0.010      0.992     -11.629      11.74
4
ar.S.L12      -0.2964      0.249     -1.190      0.234      -0.784       0.19
2
ar.S.L24      -0.2328      0.226     -1.028      0.304      -0.677       0.21
1
ma.S.L12      -0.6946      0.260     -2.670      0.008      -1.205      -0.18
5
sigma2      1.479e+10   5.38e-08   2.75e+17      0.000    1.48e+10    1.48e+1
0
==========================================================================================
======
Ljung-Box (L1) (Q):                   0.08   Jarque-Bera (JB):
6.92
Prob(Q):                              0.78   Prob(JB):
0.03
Heteroskedasticity (H):               2.90   Skew:
0.24
Prob(H) (two-sided):                  0.00   Kurtosis:
3.90
```

```
==============================================================================
======

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (comple
x-step).
[2] Covariance matrix is singular or near-singular, with condition number 6.7
9e+33. Standard errors may be unstable.
```

This model has a high AIC. Will look for other values of pdq and s that might lower it

- SARIMA MODEL

- We saw that there is seasonality in the data, so the best model to use is one that also caters for seasonality-SARIMAX

In [ ]:
```
my_model = sm.tsa.statespace.SARIMAX(ts,
                                     order=(1, 1, 1),
                                     seasonal_order=(1, 1, 1, 12),
                                     enforce_stationarity=False,
                                     enforce_invertibility=False)

results = my_model.fit()
print(results.summary())
```

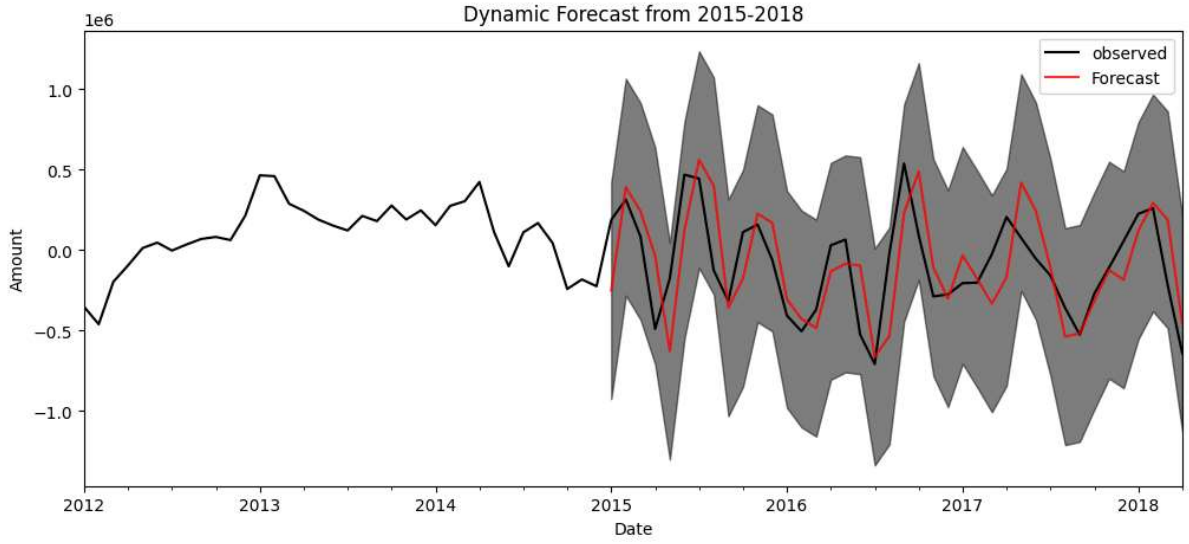
```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                    MedianHousePrice   No. Observations:
159
Model:             SARIMAX(1, 1, 1)x(1, 1, 1, 12)   Log Likelihood
-1824.719
Date:                            Mon, 01 Jul 2024   AIC
3659.439
Time:                                    12:34:47   BIC
3673.853
Sample:                                02-01-2005   HQIC
3665.296
                                     - 04-01-2018
Covariance Type:                              opg
==============================================================================
=
                 coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------
-
ar.L1         -0.0467      0.510     -0.092      0.927      -1.047       0.95
3
ma.L1          0.5443      0.410      1.328      0.184      -0.259       1.34
8
ar.S.L12      -0.5608      0.243     -2.306      0.021      -1.037      -0.08
4
ma.S.L12      -0.6857      0.156     -4.386      0.000      -0.992      -0.37
9
sigma2      1.182e+11   2.87e-13   4.12e+23      0.000    1.18e+11    1.18e+1
1
===================================================================================
======
Ljung-Box (L1) (Q):                   0.07   Jarque-Bera (JB):
2.18
Prob(Q):                              0.79   Prob(JB):
0.34
Heteroskedasticity (H):               2.71   Skew:
-0.21
Prob(H) (two-sided):                  0.00   Kurtosis:
3.47
===================================================================================
======

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (comple
x-step).
[2] Covariance matrix is singular or near-singular, with condition number 1.1
9e+40. Standard errors may be unstable.
```

- The aic has improved. So lets check the distribution of residuals for this model

In [ ]:
```
results.plot_diagnostics(figsize=(15, 12))
plt.show()
```

- **Forecasting and Model Evaluation**:
- We compare predicted values to real values of the time series, which will help us understand the accuracy of our forecasts

In [ ]:
```
import matplotlib.dates as mdates
pred = results.get_prediction(start=pd.to_datetime('2015-01-01'), dynamic=Fals

pred_ci = pred.conf_int() # this gives us the confidence interval for our fore
num_index = mdates.date2num(pred_ci.index)

plt.figure(figsize = (12,5))
ax = ts['2012':].plot(label='observed',color='black')
pred.predicted_mean.plot(ax=ax, label='Forecast', color='red',alpha=0.8)

ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=0.5)

ax.set_title('Dynamic Forecast from 2015-2018')
ax.set_xlabel('Date')
ax.set_ylabel('pRICE')
plt.legend()

plt.show()
```



-The model performed well in forcasting the actual values

In [ ]:
```
# rmse = np.sqrt(mean_squared_error(y_truth1, y_forecasted1))
# print(f"RMSE: {rmse:.2f}")
# evaluation
y_forecasted1 = pred.predicted_mean
y_truth1 = ts['2015-01-01':]
mean_absolute_percentage_error(y_truth1, y_forecasted1)
rmse = np.sqrt(mean_squared_error(y_truth1, y_forecasted1))
print(f"RMSE: {rmse:.2f}")
```

```
RMSE: 263342.52
```

The RMSE is low meaning themodel performs well.

In [ ]:
```
my_series = my_series[my_series.index >= '2004-01-01'] # Use .index to access
my_series = my_series.reset_index(name='y') # Convert the index to a column na
my_series.columns = ['ds','y'] # Rename the original index column to 'ds'
my_series.head()
```

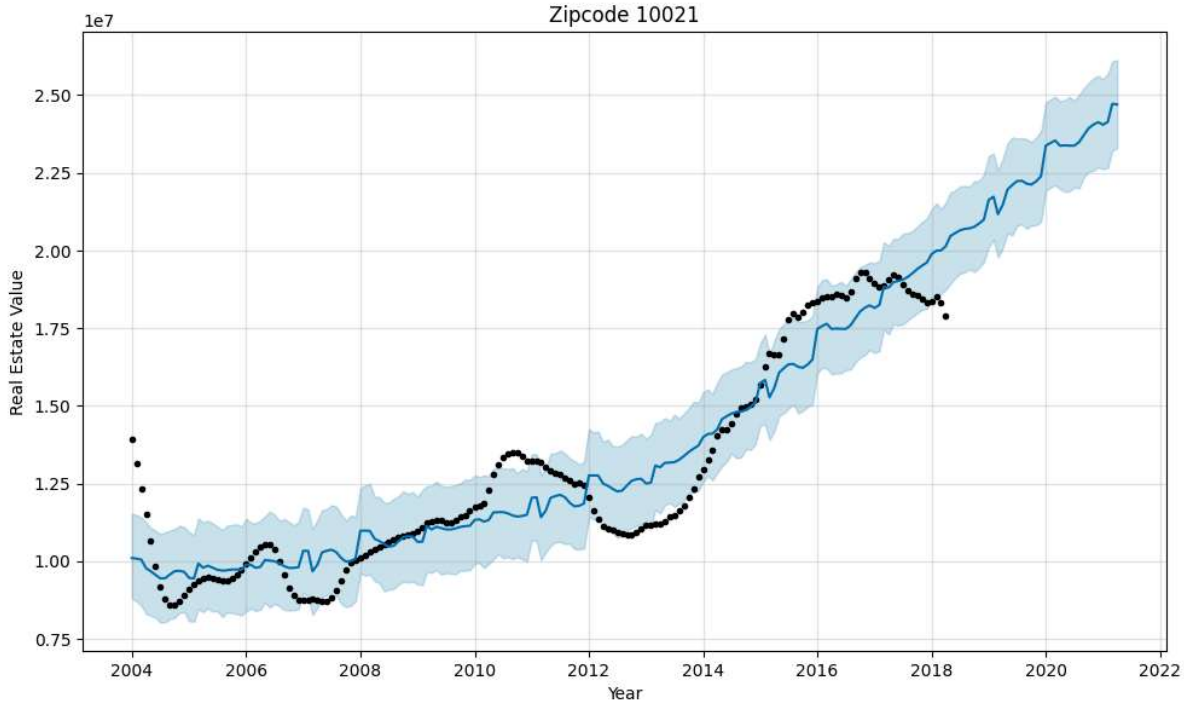
Out[64]:

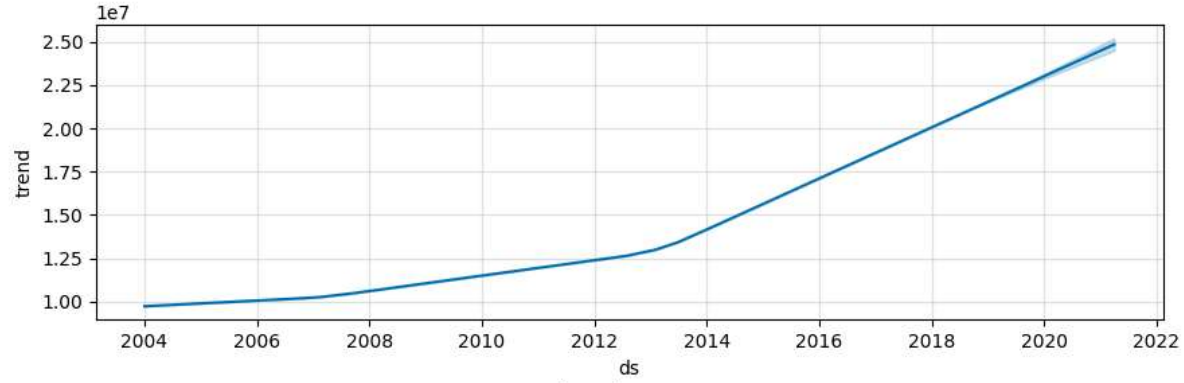
	ds	y
0	2004-01-01	13922800.0
1	2004-02-01	13140500.0
2	2004-03-01	12333800.0
3	2004-04-01	11490700.0
4	2004-05-01	10641200.0

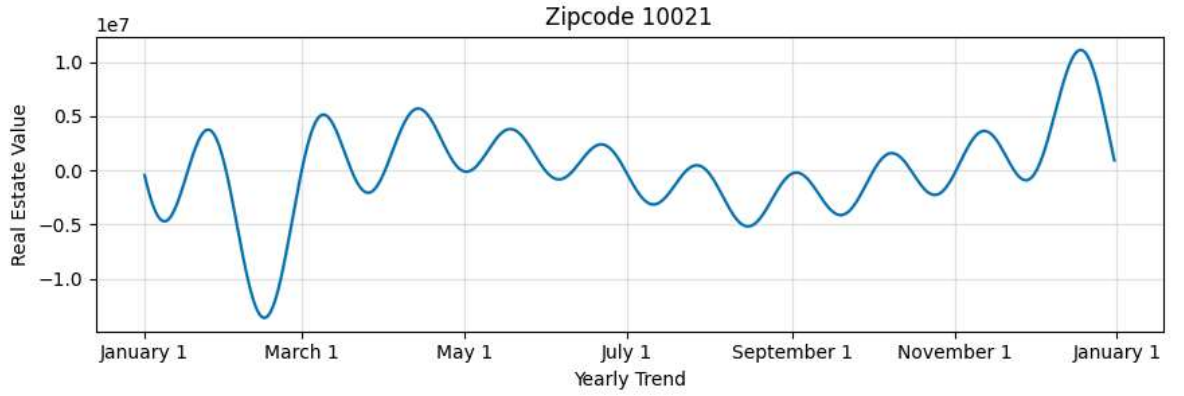
- facebook prophet

In [ ]:
```
from prophet import Prophet
model= Prophet()
model.fit(my_series)

#making predictions
future_dates = model.make_future_dataframe(periods=36,freq='MS')
forecast = model.predict(future_dates)

forecasted_data=forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']]
forecasted_data

model.plot(forecast, uncertainty=True)
plt.title('Zipcode 10021')
plt.ylabel('Real Estate Value')
plt.xlabel('Year')
model.plot_components(forecast)
plt.title('Zipcode 10021')
plt.ylabel('Real Estate Value')
plt.xlabel('Yearly Trend');
```

```
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonalit
y=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=
True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpht2uo7r8/p2d7w9mh.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpht2uo7r8/4g5_bbsp.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-packages/proph
et/stan_model/prophet_model.bin', 'random', 'seed=66685', 'data', 'file=/tmp/
tmpht2uo7r8/p2d7w9mh.json', 'init=/tmp/tmpht2uo7r8/4g5_bbsp.json', 'output',
'file=/tmp/tmpht2uo7r8/prophet_modeltqp5t_63/prophet_model-20240701123813.cs
v', 'method=optimize', 'algorithm=lbfgs', 'iter=10000']
12:38:13 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
12:38:13 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```







- The trend suggests that real estate values in this zipcode have increased significantly over time. Prices appear to have risen from around $1 million in 2004 to over 10 million by 2022.

EVALUATION

- To the stakeholder, a real estate investment firm , the best investment would be the one that promises high return on investment. When constructing residential homes in the United States, they would much be interested in the zipcodes that would promise high ROI.
- In this project the goal was to identify the top five zip codes with high ROI as well as forecasting future house prices in these zipcodes. The Return on Investment (ROI) for each row in the DataFrame was calculated by comparing the property value in April 2018 to the property value in April 1996. The calculated ROI therefore represents the percentage increase in the property value from April 1996 to April 2018. ROI of 2.083782 means the property value increased by 208.3782% over this period.

The top 5 zip codes with the highest ROI were observed as below:

1. 10021
2. 31561
3. 10014
4. 11217
5. 96714 / 11231

- The first two zip codes (10021 and 31561) were noted for having exceptionally high ROI.
- SARIMA model was the best model due to seasonality in the data with an RMSE of 263342.52 The model was able to capture the seasonality spikes when forecasting future prices in the top zipcodes.

INSIGHTS AND RECOMMENDATIONS

Focus on Top-Performing Zip Codes:

- 10021 (Upper East Side, Manhattan, NY): Known for its affluent residential area, high-end retail, and proximity to Central Park. Investing in this area is likely to yield high returns due to its sustained demand and prestigious reputation.
- 31561 (Sea Island, GA): This area is known for luxury resorts, exclusive communities, and a high quality of life. Properties here are highly sought after for both vacation homes and permanent residences, ensuring a high ROI.

Consider High ROI for the other three Zip Codes:

- 10014 (West Village, Manhattan, NY): Known for its historic charm, vibrant cultural scene, and high property demand. The West Village has shown significant property value appreciation and continues to attract a wealthy demographic.
- 11217 (Boerum Hill, Brooklyn, NY): This area has seen a surge in popularity due to its brownstone-lined streets, cultural amenities, and proximity to downtown Manhattan. The gentrification of Brooklyn makes this zip code a lucrative investment option.
- 96714 (Hanalei, HI) / 11231 (Carroll Gardens, Brooklyn, NY):
- 96714: Known for its picturesque landscape, tourism, and luxury real estate market.
- 11231: Carroll Gardens is known for its historic homes, family-friendly atmosphere, and increasing property values.

Diversification:

- Diversify investments across the top zip codes to mitigate risk. While investing heavily in the top two zip codes, it is also beneficial to allocate some resources to the other identified high ROI areas to spread the risk and maximize potential gains.

Future Forecasting:

- Stay updated with local market trends, economic indicators, and changes in urban development policies that could affect property values. Invest in zip codes with strong economic fundamentals, good infrastructure, and amenities that attract buyers and renters. Sustainable Investment Strategy:
- Consider the sustainability of high ROI. Look into factors such as local economy, employment rates and population growth to ensure that the identified zip codes will continue to provide high returns in the future.

CONCLUSION

- Investing in the identified high ROI zip codes can provide substantial returns due to their historical performance and market desirability. Prioritizing investments in zip codes 10021 and 31561 while diversifying across the other high-performing areas will help the real estate investment firm maximize its returns while mitigating risks. Continuing to forecast and monitor market conditions will ensure that investments remain profitable in the long term.