ZILLOW DATASET TIME SERIES

1. Introduction

1.1 Business Overview

Real Estate Investment Firms provide comprehensive investment advisory services, including market research, property analysis, due diligence, financial modeling, and portfolio management. Our goal is to optimize investment decisions, mitigate risks, and ensure long-term success.

The primary focus of this project is to identify opportunities in real estate markets and capitalize them to generate significant profits. We will carefully assess and mitigate risks associated with each investment based on some factors like market volatility. We will conduct a market analysis to identify areas of high demand and growth for optimal investment and prioritize investments with the potential substantial returns based on factors like property appreciation and market demand.

With long term value our investement strategies will focus on the ability to generate consistent cashflows overtime. Real Estate Firms can achieve a long-term partnerships with clients by achieving their financial objectives through successful real estate investments.

1.2 Problem Statement.

At Matawi Real Estate Investment firm we seek to identify the top five areas for potential investment opportunities. The firm aims to maximize return on investment by strategically selecting areas that exhibit strong growth potential and promising real estate market conditions. By leveraging data from Zillow Research, our firm intends to make data-driven investment decisions and optimize investment portfolio.

The investment firm needs to determine the top five areas that present the best investment opportunities based on real estate market trends and historical data. We will conduct a comprehensive analysis of various factors, such as past price trends, growth rates, market demand, and other relevant indicators to identify the areas with the highest potential for future price appreciation.

1.3 Objectives

Main objective:

- The main objective is to develop a forecasting model that can accurately predict real estate price movements in different areas and assist in identifying the most favorable locations for investment between the period of April 1996 to April 2018.

Specific objectives:

- To assess and mitigate potential risks associated with market volatility and economic fluctuations.
- To Utilize time series analysis techniques to identify underlying patterns, trends, and seasonality in the real estate price data.
- To Build a time series predictive model that can forecast real estate prices.

- To Evaluate the forecasting model's performance by comparing its predictions against actual real estate prices
- To forecast house prices in the next subsequent years.

1.4 Success Metrics.

In this project, we will determine the best model for our analysis by considering three important metrics: AIC (Akaike Information

Criterion), BIC (Bayesian Information Criterion), and RMSE (Root Mean Square Error). These metrics will allow us to assess the

goodness of fit and predictive performance of different models. AIC and BIC provide measures of the model's complexity and how

well it balances goodness of fit and overfitting. Lower values of AIC and BIC indicate a better trade-off between complexity and

fit. Additionally, we will utilize RMSE, which quantifies the average difference between predicted and observed values. By

comparing the AIC, BIC, and RMSE values across various models, we will identify the model that demonstrates the lowest values for

all three metrics, indicating the best model for our analysis.

2. Data Understanding

The dataset used in this project consists of historic median house prices from various regions in the USA. It covers a time period of 22 years, specifically from April 1996 to April 2018. The dataset was obtained from the [Zillow website. (https://www.zillow.com/research/data/)](https://www.zillow.com/research/data/)

Here are the key details about the dataset:

- It contains 14,723 rows and 272 columns.
- Out of the 272 columns, 4 columns are categorical, while the rest are numerical.

The columns are described as follows:

- RegionID: A unique identifier for each region.
- RegionName: The names of the regions, represented by zip codes.
- City: The corresponding city names for each region.
- State: The names of the states where the regions are located.
- Metro: The names of the metropolitan areas associated with the regions.
- County Name: The names of the counties where the regions are situated.
- Size Rank: The ranking of the zip codes based on urbanization.
- Date Columns (265 Columns): These columns represent different dates and provide median house prices for each region over the years.

3. Data Preparation

In [1]:

```
# importing the Libraries
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
# Previewing the dataset.
df = pd.read_csv('zillow_data.csv')
df
```

Out[2]:

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-0
0	84654	60657	Chicago	IL	Chicago	Cook	1	334200.(
1	90668	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	235700.(
2	91982	77494	Katy	TX	Houston	Harris	3	210400.(
3	84616	60614	Chicago	IL	Chicago	Cook	4	498100.(
4	93144	79936	El Paso	TX	El Paso	El Paso	5	77300.(
...	...	...	...	...	...	...	...	..
14718	58333	1338	Ashfield	MA	Greenfield Town	Franklin	14719	94600.(
14719	59107	3293	Woodstock	NH	Claremont	Grafton	14720	92700.(
14720	75672	40404	Berea	KY	Richmond	Madison	14721	57100.(
14721	93733	81225	Mount Crested Butte	CO	NaN	Gunnison	14722	191100.(
14722	95851	89155	Mesquite	NV	Las Vegas	Clark	14723	176400.(

14723 rows × 272 columns

In [3]:

```
# A function to analyze the shape, number of columns, and information of the data
def analyze_dataset(df):
    """
    This function  outputs information about the shape,
    columns, and information of the dataset using the Pandas library.
    """
    # Output the shape of the dataset
    print("Shape of dataset:", df.shape)
    print('\n-------------------------------------------------------------')

    # Output the column names of the dataset
    print("Column names:", list(df.columns))
    print('\n-------------------------------------------------------------')

    # Output information about the dataset
    print(df.info())
    print('\n-------------------------------------------------------------')

    # output descriptive statistics about the dataset
    print(df.describe())
    print('\n-------------------------------------------------------------')
```

In [4]:

```
analyze_dataset(df)
```

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

-----------------------------------------------------------
Column names: ['RegionID', 'RegionName', 'City', 'State', 'Metro',
'CountyName', 'SizeRank', '1996-04', '1996-05', '1996-06', '1996-0
7', '1996-08', '1996-09', '1996-10', '1996-11', '1996-12', '1997-0
1', '1997-02', '1997-03', '1997-04', '1997-05', '1997-06', '1997-0
7', '1997-08', '1997-09', '1997-10', '1997-11', '1997-12', '1998-0
1', '1998-02', '1998-03', '1998-04', '1998-05', '1998-06', '1998-0
7', '1998-08', '1998-09', '1998-10', '1998-11', '1998-12', '1999-0
1', '1999-02', '1999-03', '1999-04', '1999-05', '1999-06', '1999-0
7', '1999-08', '1999-09', '1999-10', '1999-11', '1999-12', '2000-0
1', '2000-02', '2000-03', '2000-04', '2000-05', '2000-06', '2000-0
7', '2000-08', '2000-09', '2000-10', '2000-11', '2000-12', '2001-0
1', '2001-02', '2001-03', '2001-04', '2001-05', '2001-06', '2001-0
7', '2001-08', '2001-09', '2001-10', '2001-11', '2001-12', '2002-0
1', '2002-02', '2002-03', '2002-04', '2002-05', '2002-06', '2002-0
7', '2002-08', '2002-09', '2002-10', '2002-11', '2002-12', '2003-0
1', '2003-02', '2003-03', '2003-04', '2003-05', '2003-06', '2003-0
7', '2003-08', '2003-09', '2003-10', '2003-11', '2003-12', '2004-0
1', '2004-02', '2004-03', '2004-04', '2004-05', '2004-06', '2004-0
7', '2004-08', '2004-09', '2004-10', '2004-11', '2004-12', '2005-0
1', '2005-02', '2005-03', '2005-04', '2005-05', '2005-06', '2005-0
7', '2005-08', '2005-09', '2005-10', '2005-11', '2005-12', '2006-0
1', '2006-02', '2006-03', '2006-04', '2006-05', '2006-06', '2006-0
7', '2006-08', '2006-09', '2006-10', '2006-11', '2006-12', '2007-0
1', '2007-02', '2007-03', '2007-04', '2007-05', '2007-06', '2007-0
7', '2007-08', '2007-09', '2007-10', '2007-11', '2007-12', '2008-0
1', '2008-02', '2008-03', '2008-04', '2008-05', '2008-06', '2008-0
7', '2008-08', '2008-09', '2008-10', '2008-11', '2008-12', '2009-0
1', '2009-02', '2009-03', '2009-04', '2009-05', '2009-06', '2009-0
7', '2009-08', '2009-09', '2009-10', '2009-11', '2009-12', '2010-0
1', '2010-02', '2010-03', '2010-04', '2010-05', '2010-06', '2010-0
7', '2010-08', '2010-09', '2010-10', '2010-11', '2010-12', '2011-0
1', '2011-02', '2011-03', '2011-04', '2011-05', '2011-06', '2011-0
7', '2011-08', '2011-09', '2011-10', '2011-11', '2011-12', '2012-0
1', '2012-02', '2012-03', '2012-04', '2012-05', '2012-06', '2012-0
7', '2012-08', '2012-09', '2012-10', '2012-11', '2012-12', '2013-0
1', '2013-02', '2013-03', '2013-04', '2013-05', '2013-06', '2013-0
7', '2013-08', '2013-09', '2013-10', '2013-11', '2013-12', '2014-0
1', '2014-02', '2014-03', '2014-04', '2014-05', '2014-06', '2014-0
7', '2014-08', '2014-09', '2014-10', '2014-11', '2014-12', '2015-0
1', '2015-02', '2015-03', '2015-04', '2015-05', '2015-06', '2015-0
7', '2015-08', '2015-09', '2015-10', '2015-11', '2015-12', '2016-0
1', '2016-02', '2016-03', '2016-04', '2016-05', '2016-06', '2016-0
7', '2016-08', '2016-09', '2016-10', '2016-11', '2016-12', '2017-0
1', '2017-02', '2017-03', '2017-04', '2017-05', '2017-06', '2017-0
7', '2017-08', '2017-09', '2017-10', '2017-11', '2017-12', '2018-0
1', '2018-02', '2018-03', '2018-04']

-----------------------------------------------------------
<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

-----------------------------------------------------------
            RegionID   RegionName     SizeRank      1996-04
1996-05  \
```

```
count   14723.000000  14723.000000  14723.000000  1.368400e+04  1.3
68400e+04
mean    81075.010052  48222.348706   7362.000000  1.182991e+05  1.1
84190e+05
std     31934.118525  29359.325439   4250.308342  8.600251e+04  8.6
15567e+04
min     58196.000000   1001.000000      1.000000  1.130000e+04  1.1
50000e+04
25%     67174.500000  22101.500000   3681.500000  6.880000e+04  6.8
90000e+04
50%     78007.000000  46106.000000   7362.000000  9.950000e+04  9.9
50000e+04
75%     90920.500000  75205.500000  11042.500000  1.432000e+05  1.4
33000e+05
max    753844.000000  99901.000000  14723.000000  3.676700e+06  3.7
04200e+06

            1996-06       1996-07       1996-08       1996-09      
1996-10  \
count  1.368400e+04  1.368400e+04  1.368400e+04  1.368400e+04  1.36
8400e+04
mean   1.185374e+05  1.186531e+05  1.187803e+05  1.189275e+05  1.19
1205e+05
std    8.630923e+04  8.646795e+04  8.665094e+04  8.687208e+04  8.71
5185e+04
min    1.160000e+04  1.180000e+04  1.180000e+04  1.200000e+04  1.21
0000e+04
25%    6.910000e+04  6.920000e+04  6.937500e+04  6.950000e+04  6.96
0000e+04
50%    9.970000e+04  9.970000e+04  9.980000e+04  9.990000e+04  9.99
5000e+04
75%    1.432250e+05  1.432250e+05  1.435000e+05  1.437000e+05  1.43
9000e+05
max    3.729600e+06  3.754600e+06  3.781800e+06  3.813500e+06  3.84
9600e+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  1.47
2300e+04
mean   2.795209e+05  2.810953e+05  2.826571e+05  2.843687e+05  2.86
5114e+05
std    3.656003e+05  3.670454e+05  3.695727e+05  3.717739e+05  3.72
4612e+05
min    1.450000e+04  1.430000e+04  1.410000e+04  1.390000e+04  1.38
0000e+04
25%    1.292500e+05  1.299000e+05  1.306000e+05  1.310500e+05  1.31
9500e+05
50%    1.925000e+05  1.934000e+05  1.941000e+05  1.950000e+05  1.96
7000e+05
```

```
75%    3.117000e+05  3.134000e+05  3.151000e+05  3.168500e+05  3.18
8500e+05
max    1.842880e+07  1.830710e+07  1.836590e+07  1.853040e+07  1.83
3770e+07

            2018-04
count  1.472300e+04
mean   2.880399e+05
std    3.720544e+05
min    1.380000e+04
25%    1.324000e+05
50%    1.981000e+05
75%    3.211000e+05
max    1.789490e+07

[8 rows x 268 columns]

------------------------------------------------------------
```

The dataset has 14723 rows and 272 columns,4 categorical and the rest are numerical

3.1 Data Cleaning

In [5]:

```
#Checking for duplicates and missing data
def cleaning(data):
    "This is a simple function to get missing and duplicated values"
    missing = data.isna().sum().sum()
    duplicated = data.duplicated().sum()
    return (f"There are '{missing}' missing values and '{duplicated}' duplicated
```

In [6]:

```
cleaning(df)
```

Out[6]:

```
"There are '157934' missing values and '0' duplicated values in the
dataset"
```

In [7]:

```
# Creating a dataframe to display datatypes and, the unique values.
desc = []
for i in df.columns:
    desc.append([
        i,
        df[i].dtypes,
        df[i].nunique(),
    ])

pd.DataFrame(data = desc, columns=['Feature','Dtypes','Sample_Unique'])
```

Out[7]:

	Feature	Dtypes	Sample_Unique
0	RegionID	int64	14723
1	RegionName	int64	14723
2	City	object	7554
3	State	object	51
4	Metro	object	701
...	...	...	...
267	2017-12	int64	5248
268	2018-01	int64	5276
269	2018-02	int64	5303
270	2018-03	int64	5332
271	2018-04	int64	5310

272 rows × 3 columns

In [8]:

```
def missing_values_percentage(df):
    total_missing = df.isnull().sum().sum()
    total_cells = df.size
    percentage_missing = (total_missing / total_cells) * 100
    return percentage_missing

missing_values_percentage(df)
```

Out[8]:

```
3.943759463983923
```

The missing values are 3.94% of the entire dataset.Let's preview the percentage of the missing values per column.

In [9]:

```
missing_values = df.isnull().mean() * 100

# Print the list of columns in the DataFrame along with their missing percentages
for column in missing_values.index:
    print(column, missing_values[column])
```

```
RegionID 0.0
RegionName 0.0
City 0.0
State 0.0
Metro 7.084154044691979
CountyName 0.0
SizeRank 0.0
1996-04 7.056985668681655
1996-05 7.056985668681655
1996-06 7.056985668681655
1996-07 7.056985668681655
1996-08 7.056985668681655
1996-09 7.056985668681655
1996-10 7.056985668681655
1996-11 7.056985668681655
1996-12 7.056985668681655
1997-01 7.056985668681655
1997-02 7.056985668681655
1997-03 7.056985668681655
1997-04 7.056985668681655
```

The percentage of the missing values per column is still low ranging from 1%-7% thus we chose to fill the missing values for the metro column with missing then dropping the missing values in the date columns.

In [10]:

```
## Fill the `metro` column with the word "missing"
df['Metro'].fillna('missing', inplace=True)

## Handling the date columns' missing values
df.dropna(inplace=True)
missing_values_percentage(df)
```

Out[10]:

```
0.0
```

In [11]:

```
print(missing_values_percentage(df))
print(cleaning(df))
```

```
0.0
There are '0' missing values and '0' duplicated values in the datas
et
```

The dataset doesn't have any missing values or any duplicates. Since region ID is the unique identifier, let's check if there is any duplicates in that column.

In [12]:

```
df[df['RegionID'].duplicated(keep=False)]
```

Out[12]:

RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	1996-06	...

0 rows × 272 columns

The data doesn't have any duplicated ID.

In [13]:

```
def check_value_counts(data):
    for column in data.columns:
        print(f'value counts for {column}')
        print(data[column].value_counts())
        print('-------------------------------------------','\n')

check_value_counts(df)
```

```
value counts for RegionID
84654    1
74304    1
73138    1
81233    1
60685    1
        ..
90754    1
74052    1
67659    1
89247    1
95851    1
Name: RegionID, Length: 13684, dtype: int64
-------------------------------------------

value counts for RegionName
60657    1
37330    1
34602    1
```

The data doesn't have any data inconsistencies.

3.2 Feature engineering

In [14]:

```
#rename RegionName column to Zipcode
df.rename(columns={'RegionName':'ZipCode'}, inplace=True)
```

In [15]:

```
#convert Zipcode column values to string
df.ZipCode = df.ZipCode.astype('string')
```

In [16]:

```
print(df.ZipCode.min())
```

```
1001
```

In [17]:

```
# The zipcodes need to be 5 digits long, so a zero will be added to the ones that
df['ZipCode'] = df['ZipCode'].str.zfill(5)
```

In order to address the issues identified in the business understanding phase, two new columns will be generated: one for calculating the return on investment (ROI) and another for determining the coefficient of variation. The coefficient of variation measures the extent of data point dispersion around the mean and indicates the ratio of standard deviation to the mean. This enables investors to evaluate the level of risk involved relative to the ROI.

In [18]:

```
# Calculating and creating a new column - ROI
df['ROI'] = (df['2018-04'] / df['1996-04']) - 1

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

# Calculating mean to be used for 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 columns as they are not necessary for analysis
df.drop(["std", "mean"], inplace=True, axis=1)
```

In [19]:

```
df
```

Out[19]:

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

13684 rows × 274 columns

4. Exploratory Data Analysis

In [22]:

```
melted_df = df.copy()# creating a copy of the dataset
```

The original dataset has 265 datetime columns which makes it challenging to do any data analysis and visualization. We'll melt the dataframe so that the dates are in one column and have the values in one column.

In [23]:

```
def melt_data(df):

    melted = pd.melt(df, id_vars=['ZipCode', 'RegionID', 'SizeRank', 'City', 'Sta
    melted['time'] = pd.to_datetime(melted['time'], infer_datetime_format=True)
    melted = melted.dropna(subset=['value'])
    return melted#.groupby('time').aggregate({'value':'mean'})

melted_df = melt_data(melted_df)
melted_df
```

Out[23]:

	ZipCode	RegionID	SizeRank	City	State	Metro	CountyName	ROI
0	60657	84654	1	Chicago	IL	Chicago	Cook	2.083782
1	75070	90668	2	McKinney	TX	Dallas-Fort Worth	Collin	0.365295
2	77494	91982	3	Katy	TX	Houston	Harris	0.567966
3	60614	84616	4	Chicago	IL	Chicago	Cook	1.623971
4	79936	93144	5	El Paso	TX	El Paso	El Paso	0.571798
...	...	...	...	...	...	...	...	...
3626255	01338	58333	14719	Ashfield	MA	Greenfield Town	Franklin	1.212474
3626256	03293	59107	14720	Woodstock	NH	Claremont	Grafton	1.435814
3626257	40404	75672	14721	Berea	KY	Richmond	Madison	1.336252
3626258	81225	93733	14722	Mount Crested Butte	CO	missing	Gunnison	2.476714
3626259	89155	95851	14723	Mesquite	NV	Las Vegas	Clark	1.024943

3626260 rows × 11 columns

In [23]:

```
analyze_dataset(melted_df)
```

```
Shape of dataset: (3626260, 11)

------------------------------------------------------------
Column names: ['ZipCode', 'RegionID', 'SizeRank', 'City', 'State',
'Metro', 'CountyName', 'ROI', 'CV', 'time', 'value']

------------------------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3626260 entries, 0 to 3626259
Data columns (total 11 columns):
 #   Column      Dtype
---  ------      -----
 0   ZipCode     string
 1   RegionID    int64
 2   SizeRank    int64
 3   City        object
 4   State       object
 5   Metro       object
 6   CountyName  object
 7   ROI         float64
 8   CV          float64
 9   time        datetime64[ns]
 10  value       float64
dtypes: datetime64[ns](1), float64(3), int64(2), object(4), string
(1)
memory usage: 304.3+ MB
None

------------------------------------------------------------
           RegionID      SizeRank           ROI            CV
value
count  3.626260e+06  3.626260e+06  3.626260e+06  3.626260e+06  3.62
6260e+06
mean   8.098700e+04  7.153076e+03  1.325605e+00  2.282449e-01  2.08
0499e+05
std    3.242861e+04  4.226827e+03  8.659875e-01  8.027290e-02  2.11
9435e+05
min    5.819600e+04  1.000000e+00 -5.326087e-01  4.127471e-02  1.13
0000e+04
25%    6.680875e+04  3.491750e+03  7.856907e-01  1.657495e-01  9.89
0000e+04
50%    7.784300e+04  7.037500e+03  1.139484e+00  2.237441e-01  1.48
8000e+05
75%    9.112900e+04  1.075225e+04  1.619833e+00  2.818460e-01  2.39
8000e+05
max    7.538440e+05  1.472300e+04  1.118994e+01  6.975408e-01  8.55
8700e+06

------------------------------------------------------------
```

The new dataset has 3626260 rows and 11 columns.The data is from 4th April 1996 to 4th April 2018.The house with the lowest price has a price of 11300 dollars and the one with the highest price has a price of 8558700 dollars. The highest ROI on a house is 11.2% and the lowest ROI on a house is -53.3%.

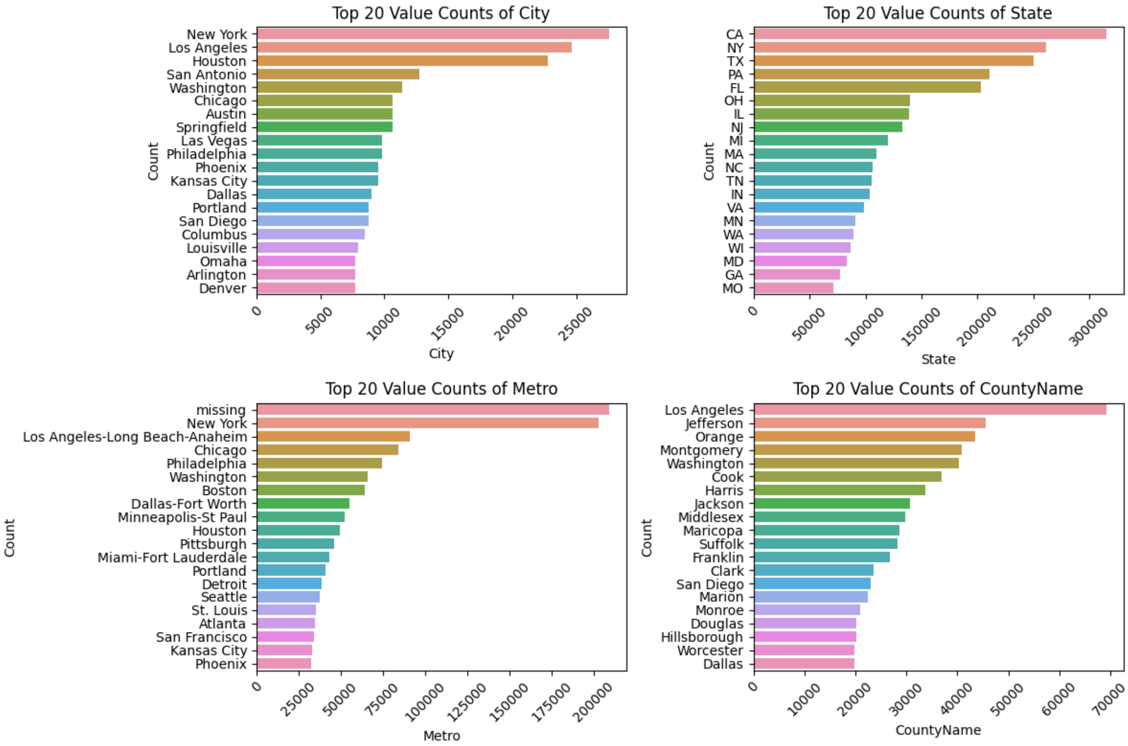
4.1 Univariate Analysis

In [24]:

```
def plot_value_counts(data, columns, top_n=20):
    """
    Plots bar plots of value counts for the specified columns in the given datase
    considering only the top_n items.
    Parameters:
    data (DataFrame): The dataset to analyze.
    columns (list): List of column names to plot value counts for.
    top_n (int): Number of top items to consider (default: 20).
    """

    num_plots = len(columns)
    num_rows = 2
    num_cols = 2
    fig, axes = plt.subplots(num_rows, num_cols, figsize=(12, 8))
    fig.tight_layout()
    for i, column in enumerate(columns):
        row = i // num_cols
        col = i % num_cols
        ax = axes[row, col]
        value_counts = data[column].value_counts().head(top_n)
        sns.barplot(y=value_counts.index, x=value_counts.values, ax=ax)
        ax.set_title(f'Top {top_n} Value Counts of {column}')
        ax.set_xlabel(column)
        ax.set_ylabel('Count')
        ax.tick_params(axis='x', rotation=45)
    # Hide empty subplots if there are any
    if num_plots < num_rows * num_cols:
        for i in range(num_plots, num_rows * num_cols):
            row = i // num_cols
            col = i % num_cols
            fig.delaxes(axes[row, col])
    plt.tight_layout()
    plt.show()
```

In [25]:

```
columns_list = ["City","State","Metro","CountyName"]
plot_value_counts(melted_df, columns_list)
```



The top 5 cities, states metro and counties with the highest number of houses are:

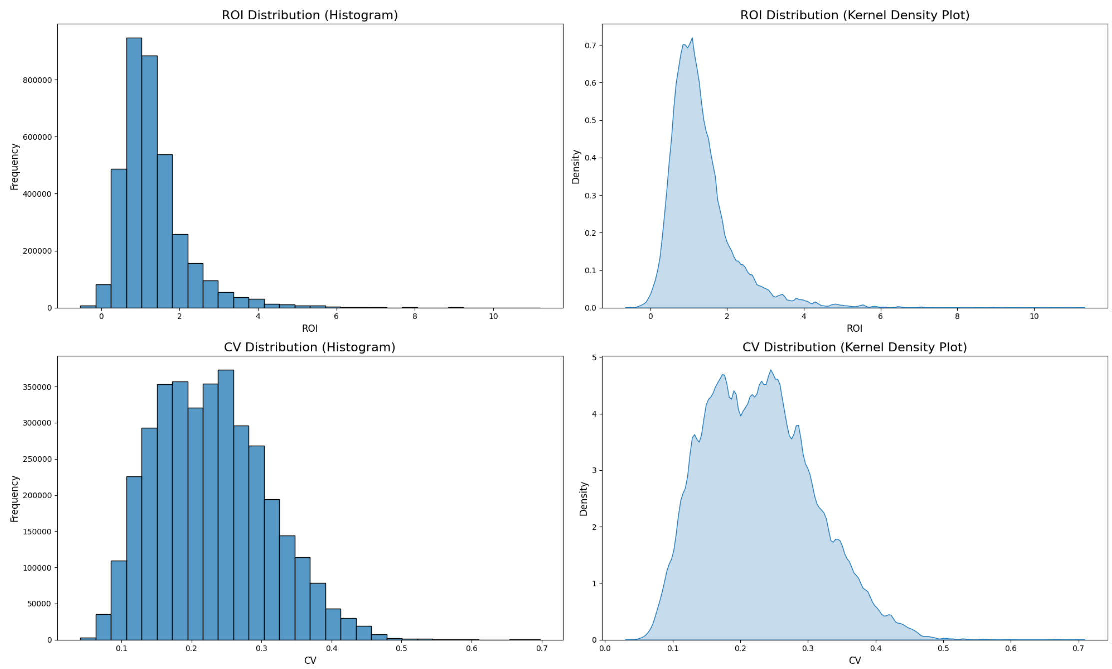
- cities: New York, Los Angeles, Houston, San Antonio and Washington
- states: CA, NY, TX,PA,FL
- metro: New York, Los Angeles, Chicago, Philadelphia, Washington
- counties: Los Angeles,Jefferson, Orange, Washington, Montgomery

In [26]:

```
import seaborn as sns
import matplotlib.pyplot as plt

def plot_column_distributions(data, columns):
    num_columns = len(columns)
    fig, axes = plt.subplots(num_columns, 2, figsize=(10*2, 6*num_columns))

    for i, column in enumerate(columns):
        ax1 = axes[i, 0]
        ax2 = axes[i, 1]

        # Plot histogram using seaborn
        sns.histplot(data[column], ax=ax1, bins=30, kde=False, edgecolor='black')
        ax1.set_title(f'{column} Distribution (Histogram)', fontsize=16)
        ax1.set_xlabel(column, fontsize=12)
        ax1.set_ylabel('Frequency', fontsize=12)

        # Plot kernel density plot using seaborn
        sns.kdeplot(data[column], ax=ax2, fill=True)
        ax2.set_title(f'{column} Distribution (Kernel Density Plot)', fontsize=16
        ax2.set_xlabel(column, fontsize=12)
        ax2.set_ylabel('Density', fontsize=12)

    # Adjust the spacing between subplots
    plt.tight_layout()
    plt.show()
```

In [27]:

```
continuous_columns = ['ROI','CV','value']
plot_column_distributions(melted_df, continuous_columns)
```



ROI: The distribution is positively skewed.Most of the houses have an ROI between 1% and 2% .It also has a long tail showing that there are outliers, houses with higher ROI impliying higher return.

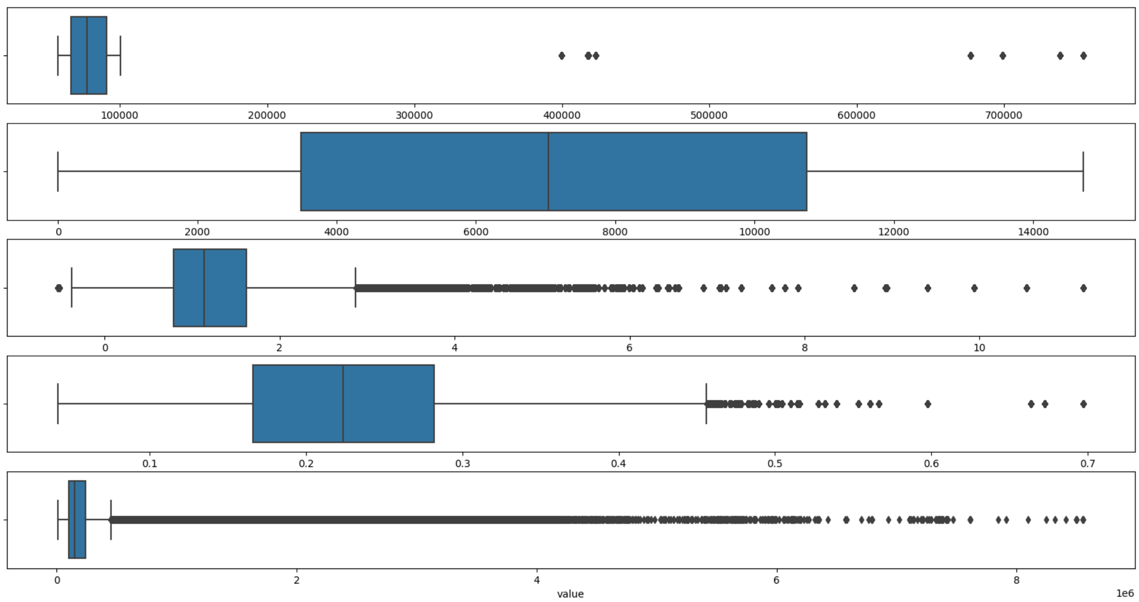
Value: The distribution of the house prices is positively skewed showing that most houses are lowly priced and it also has a long tail showing that there are outliers ie the extremely highly priced houses.

CV: The plot shows that most of the houses have a cv between 0.1 and 0.3 which shows that their prices are close to the mean thus less risk but it also has a long tail showing that there are outliers, houses with higher

In [28]:

```
def check_outliers(data, columns):
    fig, axes = plt.subplots(nrows=len(columns), ncols=1, figsize=(20,10))
    for i, column in enumerate(columns):
        # Use interquartile range (IQR) to find outliers for the specified column
        q1 = data[column].quantile(0.25)
        q3 = data[column].quantile(0.75)
        iqr = q3 - q1
        print("IQR for {} column: {}".format(column, iqr))        # Determine the
        outliers = (data[column] < q1 - 1.5 * iqr) | (data[column] > q3 + 1.5 * i
        print("Number of outliers in {} column: {}".format(column, outliers.sum()
        sns.boxplot(data=data, x=column, ax=axes[i])
plt.show()

num=melted_df.select_dtypes('number')
columns=num.columns
check_outliers(melted_df, columns)
```

```
IQR for RegionID column: 24320.25
Number of outliers in RegionID column: 26765
IQR for SizeRank column: 7260.5
Number of outliers in SizeRank column: 0
IQR for ROI column: 0.8341421408774268
Number of outliers in ROI column: 195040
IQR for CV column: 0.11609651734251925
Number of outliers in CV column: 14575
IQR for value column: 140900.0
Number of outliers in value column: 275048
```

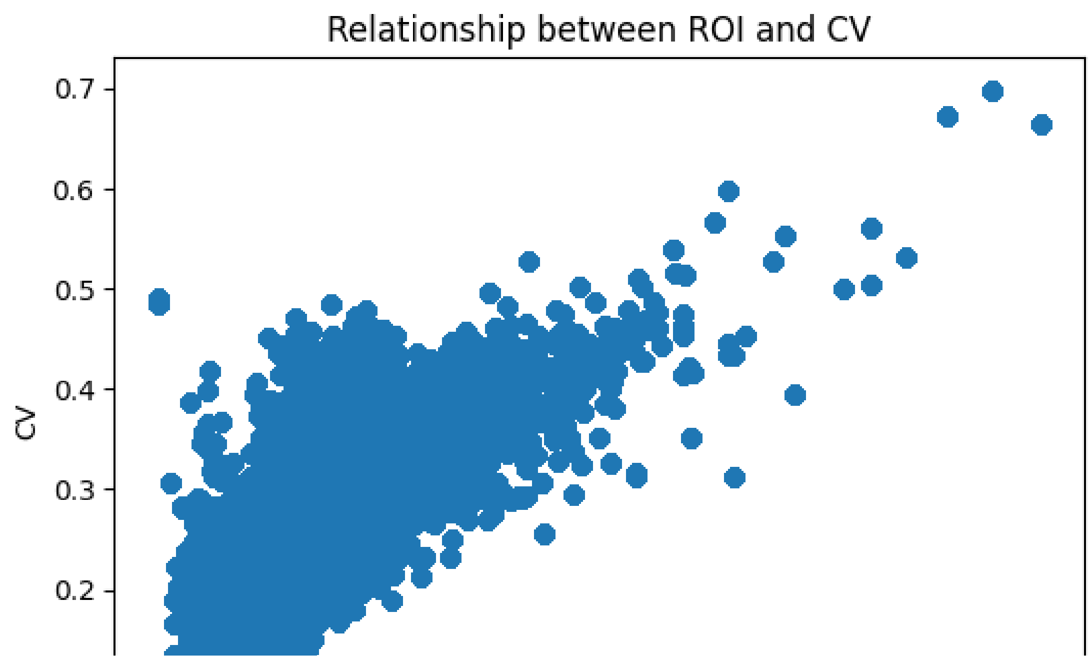


The box plots shows that there are outliers in the dataset especially in the prices(value) column which shows there are some houses that are highly priced which might provide useful information for the analysis, thus we won't remove the outliers.

4.2 Bivariate Analysis

In [29]:

```
def scatter_plot(x,y, x_label, y_label):
  plt.scatter(x,y)
  plt.xlabel(x_label)
  plt.ylabel(y_label)
  plt.title(f'Relationship between {x_label} and {y_label}')
  plt.show();

scatter_plot(melted_df['ROI'],melted_df['CV'],'ROI','CV')
```

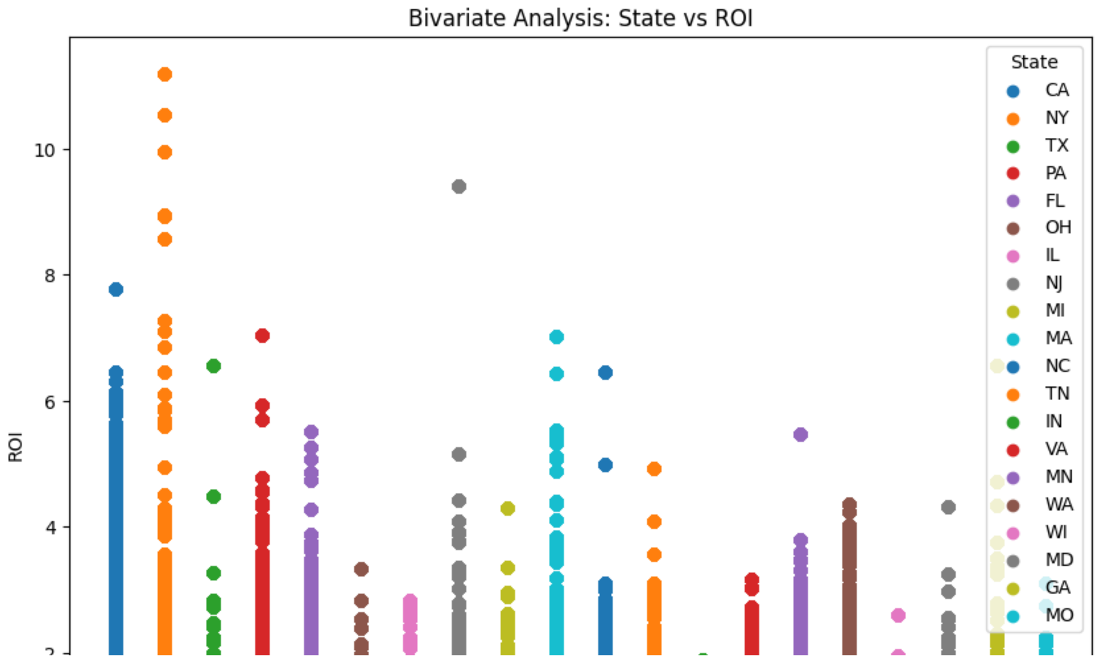


The plot shows the relationship between the return on investment and the coefficient of variation. It shows that the two have a strong positive relationship, that is, that increase in CV leads to increase in ROI and vice versa. This implies that the higher the risk, the higher the return.

Since the two have such a strong relationship, findings using ROI will be similar to those using CV . Let's examine how the other variables are related to ROI.

In [30]:

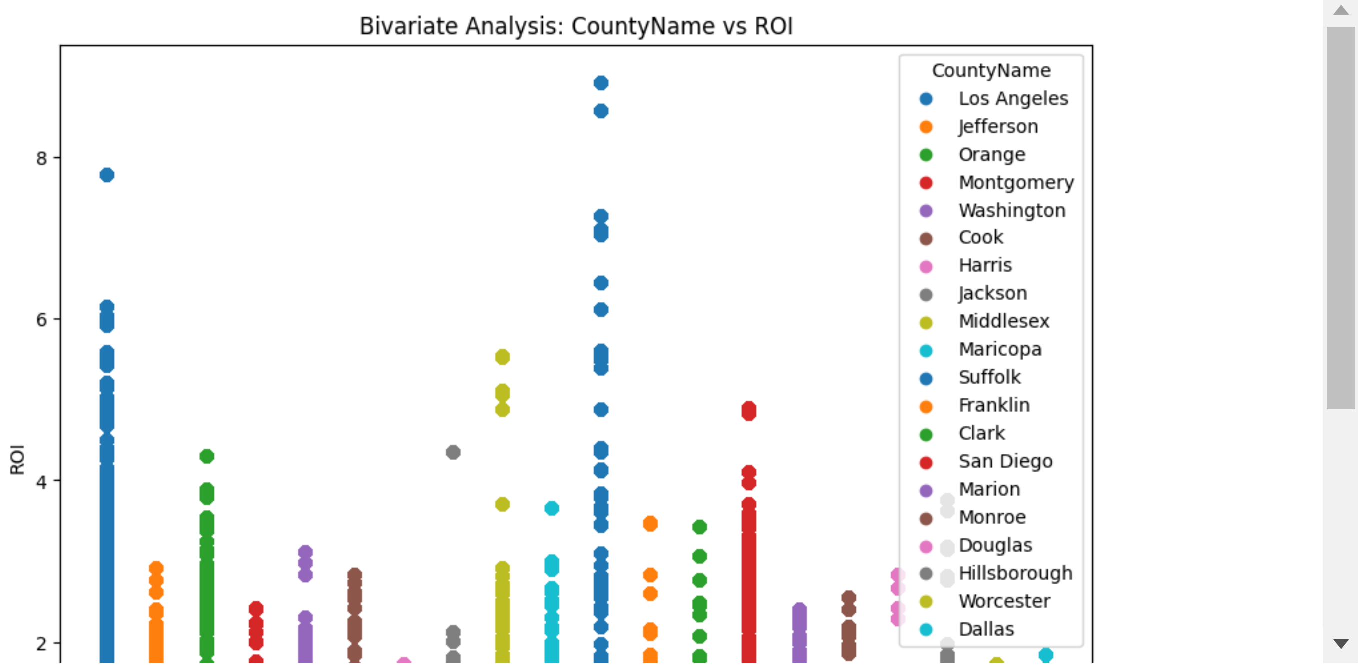
```
def plot_bivariate_analysis(data, x_column, y_column, top_n=20):

    top_categories = data[x_column].value_counts().nlargest(top_n).index
    data_top = data[data[x_column].isin(top_categories)]

    plt.figure(figsize=(10, 8))
    for category in top_categories:
        category_data = data_top[data_top[x_column] == category]
        plt.scatter(category_data[x_column], category_data[y_column], label=categ

    plt.title(f'Bivariate Analysis: {x_column} vs {y_column}')
    plt.xlabel(x_column)
    plt.ylabel(y_column)
    plt.xticks(rotation=90)
    plt.legend(title=x_column)
plt.show()
plot_bivariate_analysis(melted_df, 'State', 'ROI')
```



The above plot shows that the state with the highest return on investment is NY.

In [31]:

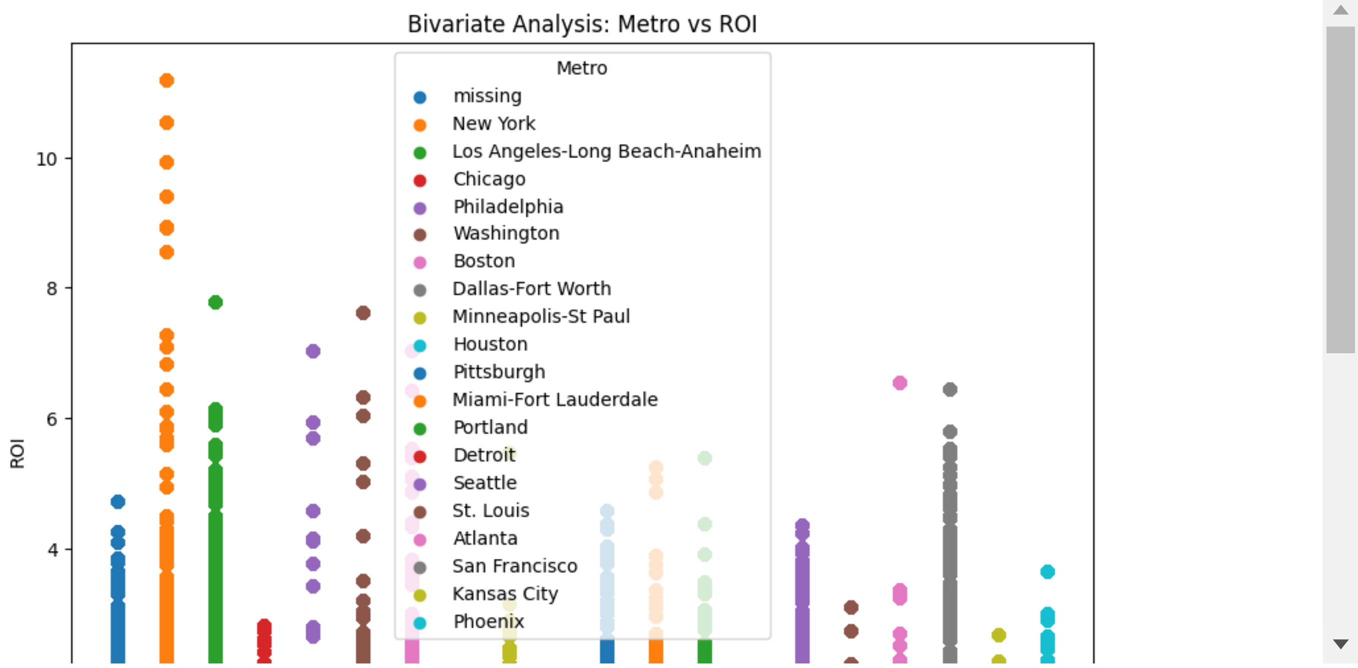
```
plot_bivariate_analysis(melted_df, 'CountyName', 'ROI')
```



The county with the highest ROI is Suffolk.

In [32]:

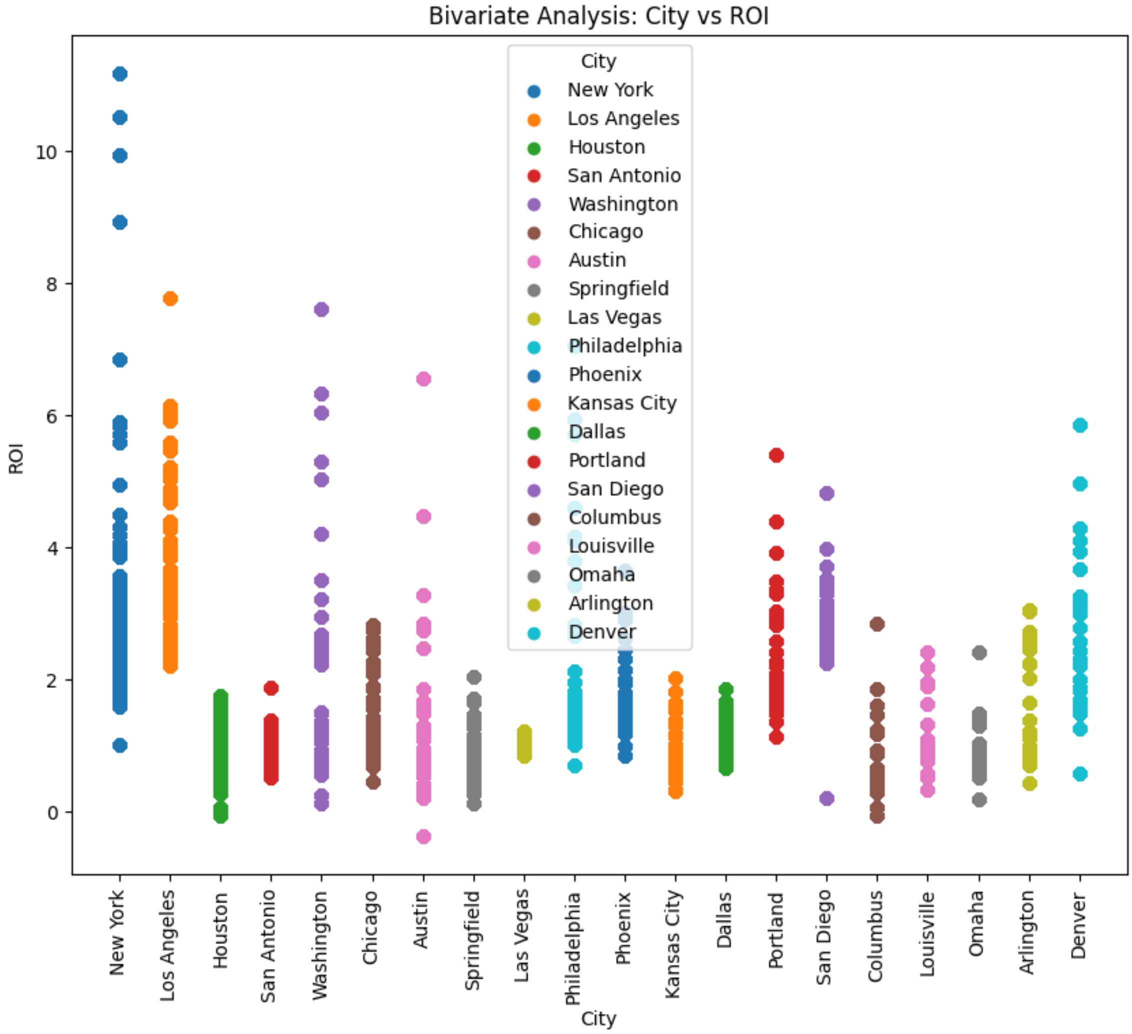
```
plot_bivariate_analysis(melted_df, 'Metro', 'ROI')
```



The metro with the highest ROI is New york.

In [33]:

```
plot_bivariate_analysis(melted_df, 'City', 'ROI')
```



The city with the highest ROI is NewYork. From the above analysis, we can conclude that properties in NewYork have the highest return on investment. Let's analyse the cities, states, metro and counties that have the highest ROI(return) but lowest CV(risk).

In [34]:

```
def get_top_rows(data, cv_column, roi_column, value_column, num_rows=10000):
    # Sort the DataFrame based on the value column in descending order,
    # coefficient of variance column in ascending order,
    # and return on investment column in descending order
    sorted_data = data.sort_values([value_column, cv_column, roi_column], ascendi
    # Get the top N rows
    top_rows = sorted_data.head(num_rows)
    return top_rows# Usage example
top_rows = get_top_rows(melted_df, 'CV', 'ROI', 'value', num_rows=10000)
```

In [35]:

```
top_rows
```

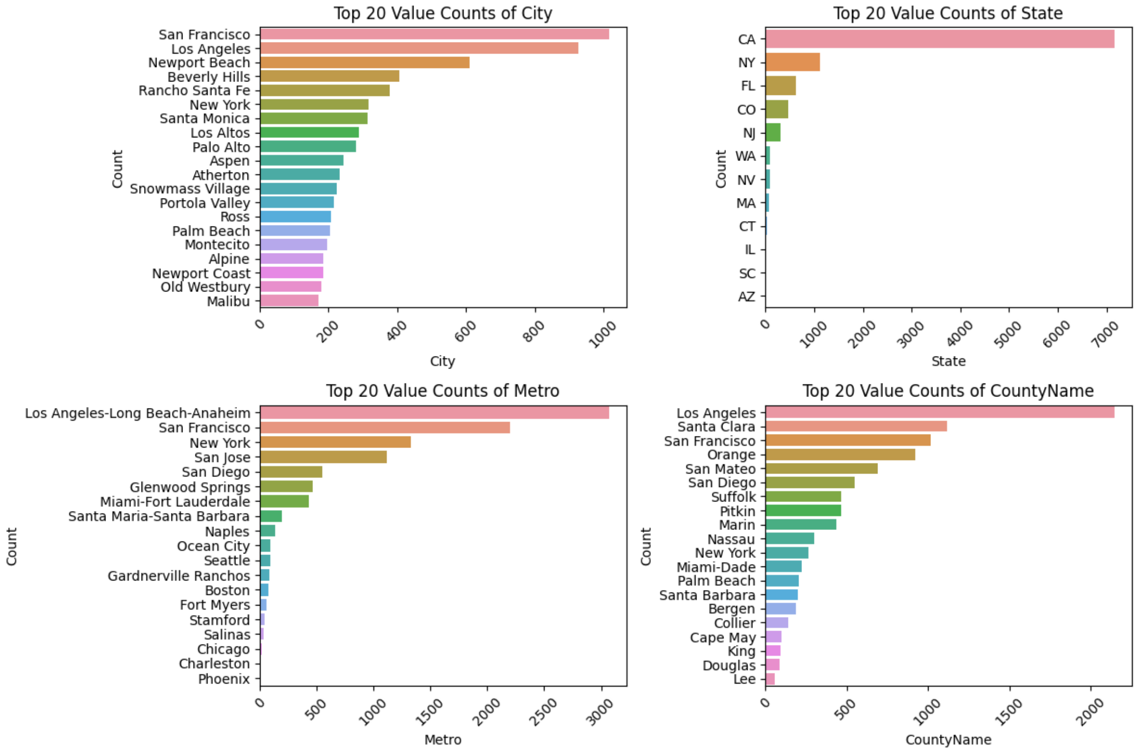
Out[35]:

	ZipCode	RegionID	SizeRank	City	State	Metro	CountyName	ROI	
3174708	10128	61703	22	New York	NY	New York	New York	1.009030	0
3161024	10128	61703	22	New York	NY	New York	New York	1.009030	0
3147340	10128	61703	22	New York	NY	New York	New York	1.009030	0
3188392	10128	61703	22	New York	NY	New York	New York	1.009030	0
3133656	10128	61703	22	New York	NY	New York	New York	1.009030	0
...	...	...	...	...	...	...	...	...	
3222545	94507	97715	7000	Alamo	CA	San Francisco	Contra Costa	3.871755	0
1560851	92037	96602	893	San Diego	CA	San Diego	San Diego	3.410964	0
3549808	94127	97581	5798	San Francisco	CA	San Francisco	San Francisco	4.228665	0
2240464	11024	61986	10436	Great Neck	NY	New York	Nassau	2.117189	0
2681342	92091	96639	13886	Rancho Santa Fe	CA	San Diego	San Diego	2.307765	0

10000 rows × 11 columns

In [36]:

```
plot_value_counts(top_rows, columns_list)
```



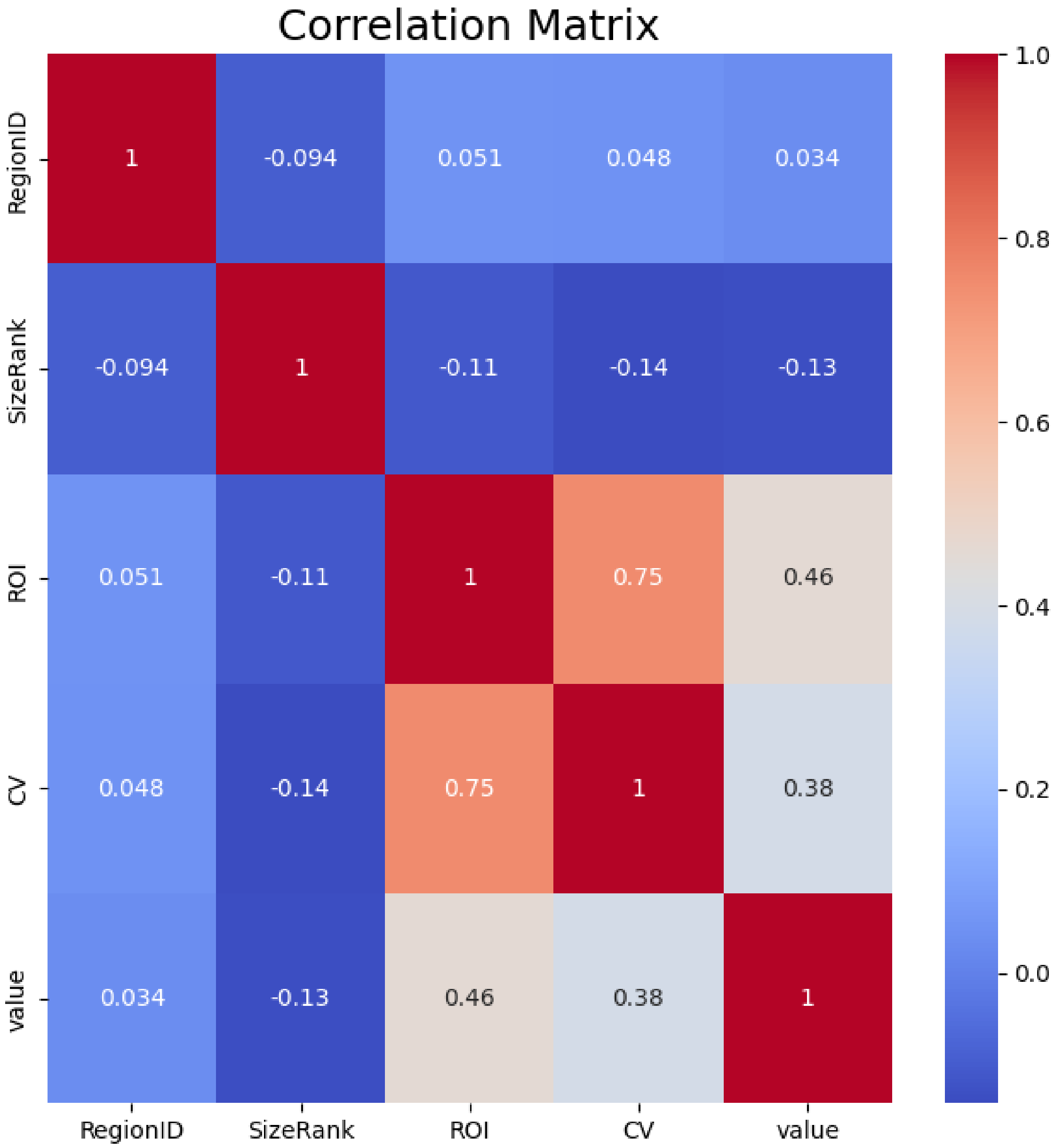
The top 5 cities, states metro and counties with the highest return and lowest risk are:

- cities: San Fransisco, Los Angeles, Newport beach, Beverly Hills and Rancho Santa Fe
- states: CA, NY,FL, CO, NJ
- metro: Los Angeles,San Fransisco,New York, San Jose, San Diego
- counties: Los Angeles,Santa Clara, San Fransisco, Orange, San Mateo

4.3 Multivariate Analysis

In [37]:

```
corr_matrix = melted_df.corr()
fig, ax = plt.subplots(figsize=(8,8))

# Set the figure size to 12 inches by 12 inches
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', ax=ax)
plt.title('Correlation Matrix', fontsize=18)
plt.show();
```



From the heat map, we can observe that most of the features exhibit weak relationships with each other, except for ROI and CV, which display a strong relationship.

4.4 Time series analysis

In [24]:

```
ts= melted_df[['value', 'time']]
ts['time']=pd.to_datetime(ts['time'])
ts.set_index('time', inplace=True)
ts
```

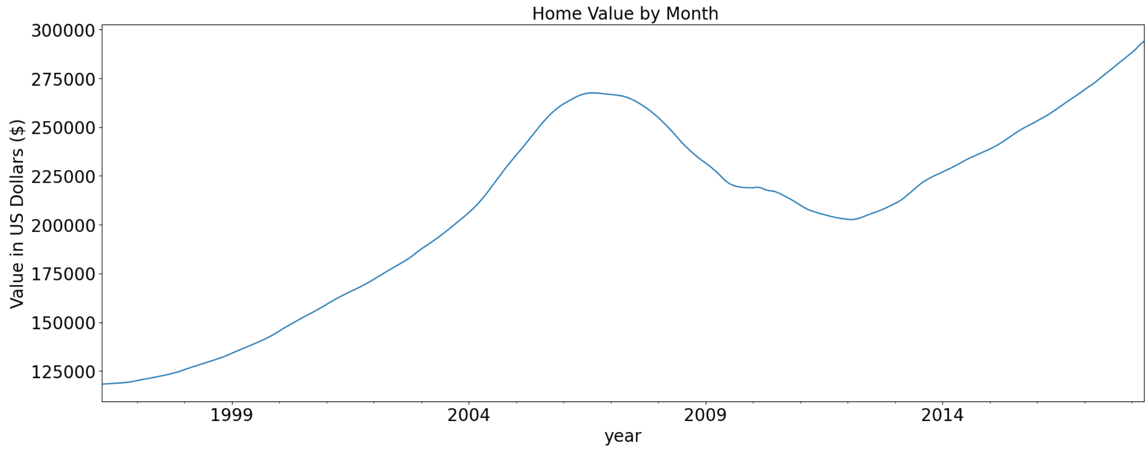
Out[24]:

	value
time	
1996-04-01	334200.0
1996-04-01	235700.0
1996-04-01	210400.0
1996-04-01	498100.0
1996-04-01	77300.0
...	...
2018-04-01	209300.0
2018-04-01	225800.0
2018-04-01	133400.0
2018-04-01	664400.0
2018-04-01	357200.0

3626260 rows × 1 columns

In [25]:

```
def plot_monthly_time_series(data,col):
    time_series_monthly_value = data.resample('MS').mean()[col]
    time_series_monthly_value.plot(figsize=(22, 8))

    plt.title(' Home Value by Month', fontsize=20)
    plt.ylabel('Value in US Dollars ($)', fontsize=20)
    plt.xlabel('year', fontsize=20)
    plt.yticks(fontsize=20)
    plt.xticks(fontsize=20)

    plt.show()

# Call the function with your data
plot_monthly_time_series(ts, 'value')
```



The plot of the housing prices indicates an overall upward trend from 1996 to around 2007, followed by a downward trend until approximately 2013, and then an upward trend again.

The year 2007 marked the beginning of the global financial crisis, which had a significant impact on the housing market. The crisis was characterized by the bursting of the housing bubble and subsequent financial turmoil, leading to a decline in housing prices in various regions. The downward trend observed until 2013 can be attributed to the aftermath of the crisis, with factors such as reduced demand, stricter lending practices, and general economic uncertainty affecting the housing market negatively.

However, after 2013, the housing market started to stabilize. Measures were taken to address the effects of the [financial crisis (https://www.sciencedirect.com/science/article/pii/S1572308910000343)](https://www.sciencedirect.com/science/article/pii/S1572308910000343), and economic conditions began to improve gradually. These improvements, along with factors such as increased consumer confidence, lower interest rates, and a recovery in the overall economy, contributed to the upward trend in housing prices

5. Modeling

5.1 Preparing Data for Modelling

5.1.1 Splitting the Data

In [26]:

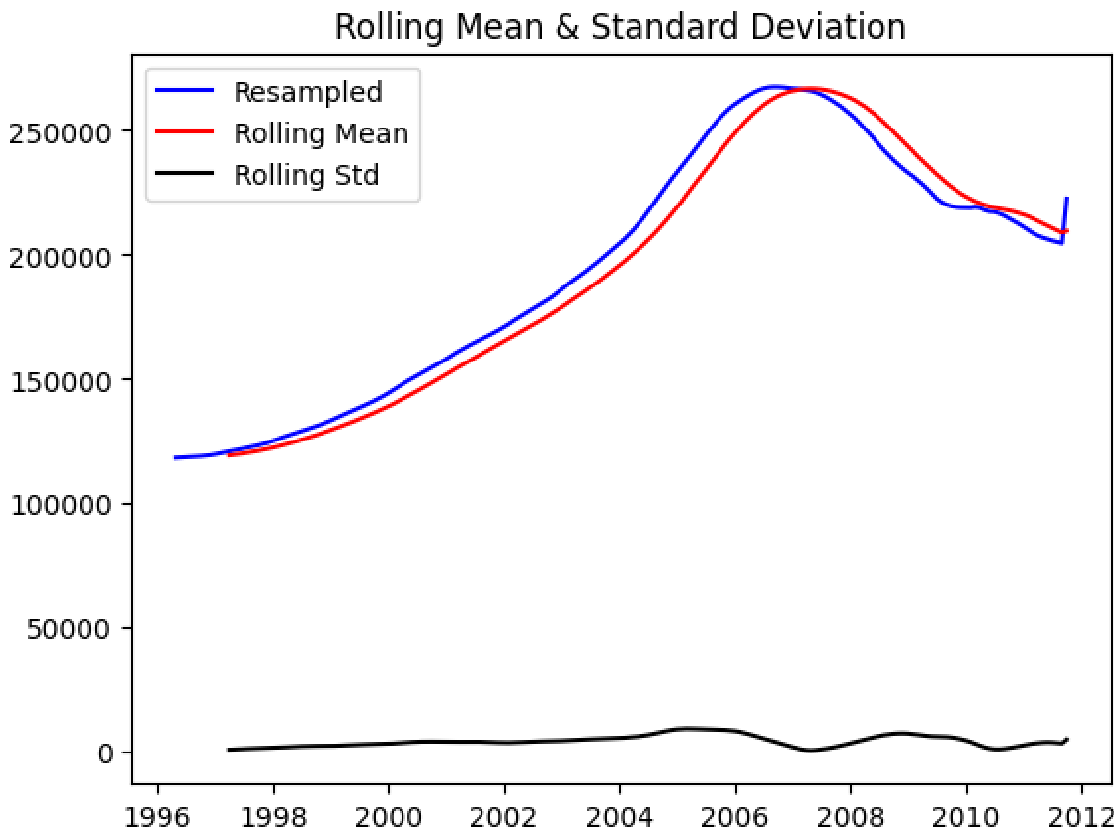
```
# split the data
df = ts.sort_index()

# Calculate the index to split the dataset
split_index = int(0.7 * len(df))

# Split the dataset
train_set = df.iloc[:split_index]
test_set = df.iloc[split_index:]

# Print the sizes of the train and test sets
print("Train set size:", len(train_set))
print("Test set size:", len(test_set))
```

```
Train set size: 2538382
Test set size: 1087878
```

5.1.2 Checking Stationarity

In [27]:

```
train_set= train_set.resample('M').mean().fillna(method='ffill')

def stationarity_check(TS):
    """
    Check the stationarity of a resampled time series using the Dickey-Fuller tes
    Parameters:
    TS (pandas.Series): the time series to check for stationarity.
    resample_freq (str): the frequency at which to resample the time series, e.g.
    Returns:
    None: prints the Dickey-Fuller test results and the plot of the rolling mean
    """
    # Calculate rolling statistics
    roll_mean = TS.rolling(window=12).mean()
    roll_std = TS.rolling(window=12).std()
    # Perform the Dickey-Fuller test
    dftest = adfuller(TS)
    # Plot rolling statistics
    plt.plot(TS, color='blue', label='Resampled')
    plt.plot(roll_mean, color='red', label='Rolling Mean')
    plt.plot(roll_std, color='black', label='Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show()
    # Print Dickey-Fuller test results
    print('Results of Dickey-Fuller Test:')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic', 'p-value', '#Lags
    for key, value in dftest[4].items():
        dfoutput['Critical Value (%s)' % key] = value
    print(dfoutput)

from statsmodels.tsa.stattools import adfuller

stationarity_check(train_set)
```



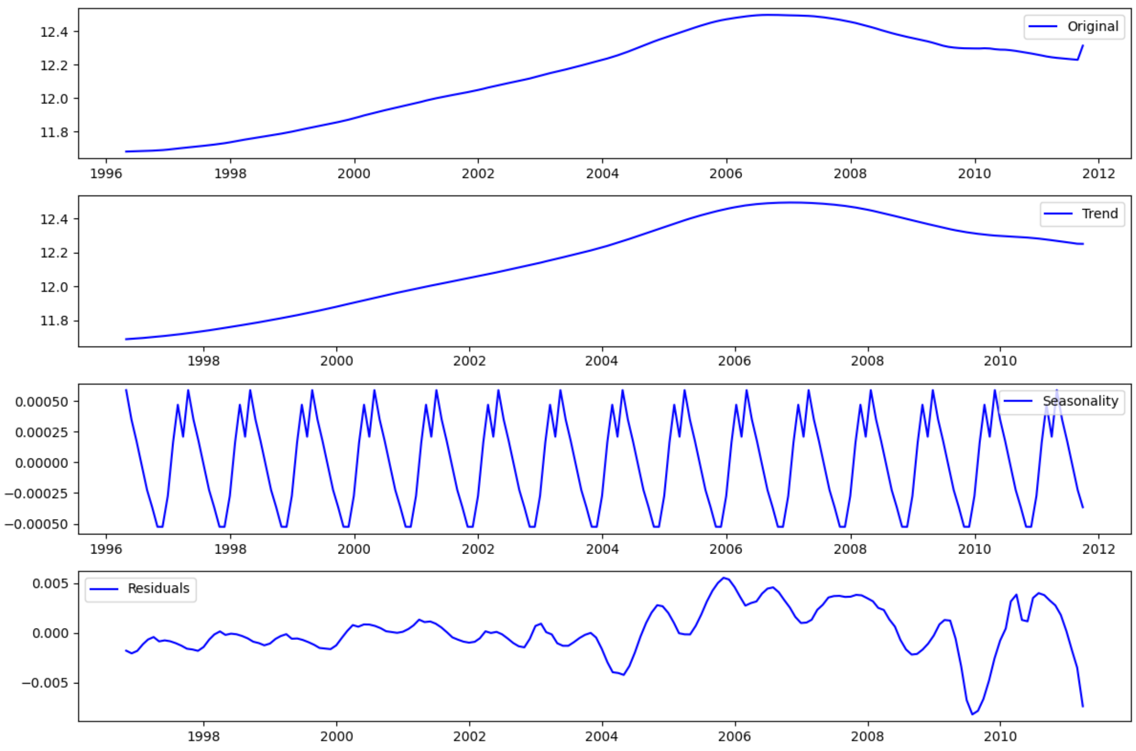

```
Results of Dickey-Fuller Test:
Test Statistic                  -0.361852
p-value                          0.916276
#Lags Used                       1.000000
Number of Observations Used    184.000000
Critical Value (1%)             -3.466398
Critical Value (5%)             -2.877380
Critical Value (10%)            -2.575214
dtype: float64
```

From the above plot we can see that the data is not stationary since the rolling mean is not constant over time. We can confirm this using the adfuller test. The p-value is greater than 0.05 thus we fail to reject the null hypothesis,the data is not stationary.

5.1.3 Check Seasonality

In [28]:

```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(np.log(train_set),period=12)

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

# Plot gathered statistics
plt.figure(figsize=(12,8))
plt.subplot(411)
plt.plot(np.log(train_set), 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()
```



This makes it easier to identify a changing mean or variation in our data. From the decomposition plot it clearly shows an upward trend in our series with seasonality and minimal variation.We will need to detrend our data because if seasonality and trend are part of the time series then there will be effects in the forecast value

5.1.4 Detrending data

Since our dataset has both the trend and the seasonal component, we'll use differencing to detrend our data since it deals with both seasonality and trend.

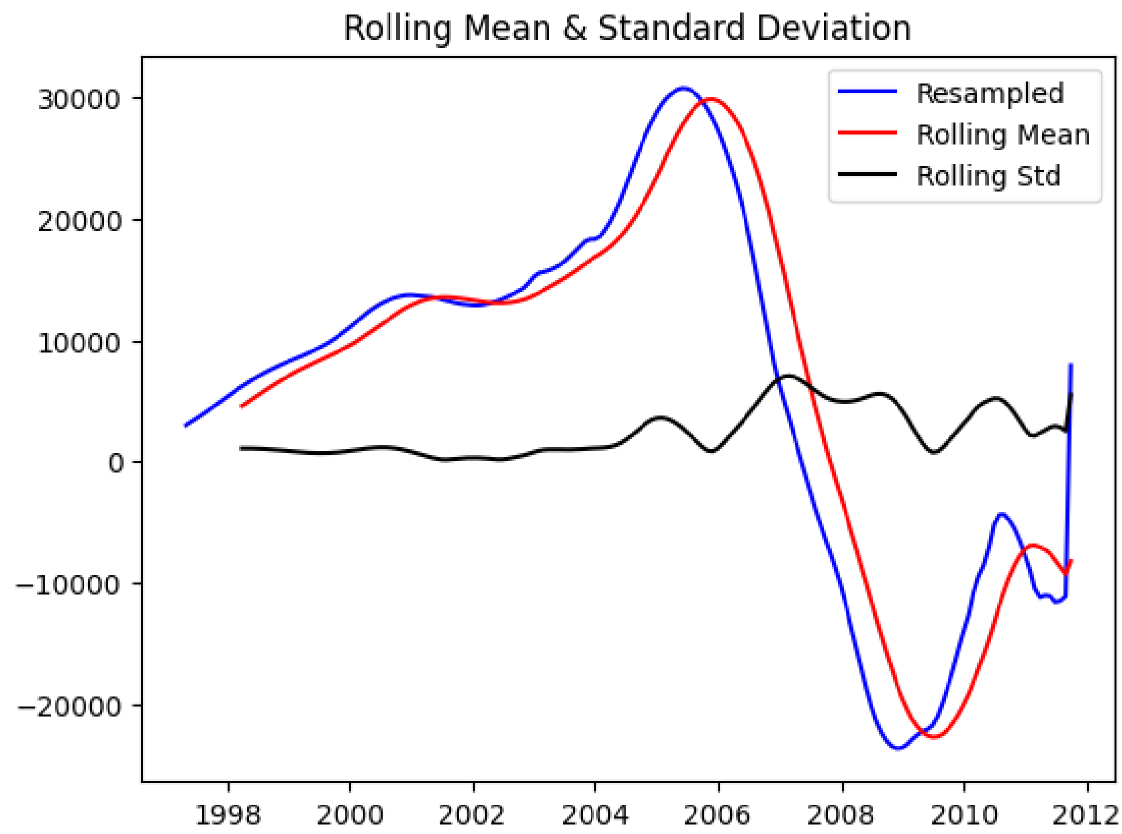
In [29]:

```
def calculate_data_diff(train_set,period):
    data_diff = train_set.diff(periods=period).dropna()
    return stationarity_check(data_diff)
```

let's check for stationarity to see if the differencing by 1 year makes the data stationary.

In [30]:

```
calculate_data_diff(train_set,12)
```

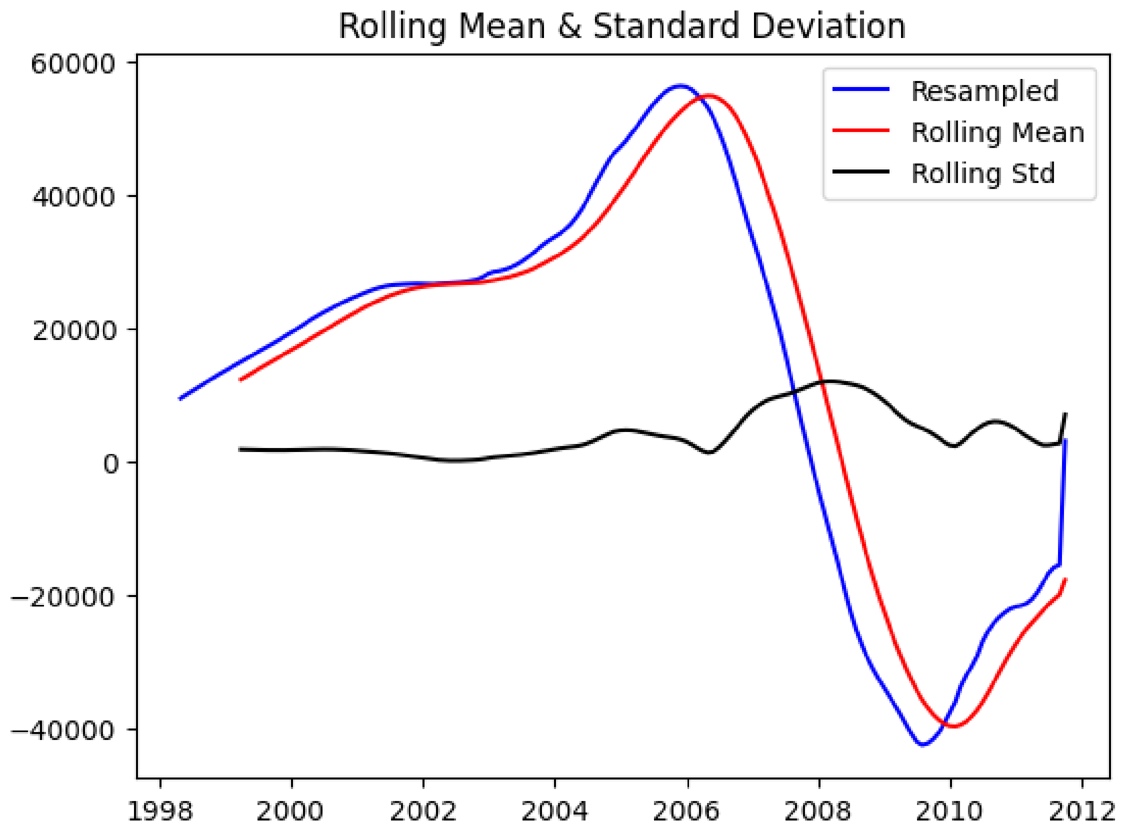



```
Results of Dickey-Fuller Test:
Test Statistic                  -2.514293
p-value                          0.112035
#Lags Used                      14.000000
Number of Observations Used    159.000000
Critical Value (1%)             -3.472161
Critical Value (5%)             -2.879895
Critical Value (10%)            -2.576557
dtype: float64
```

The p value is still greater than 0.05 showing that the data is not stationary.

In [31]:

```
calculate_data_diff(train_set,24)
```

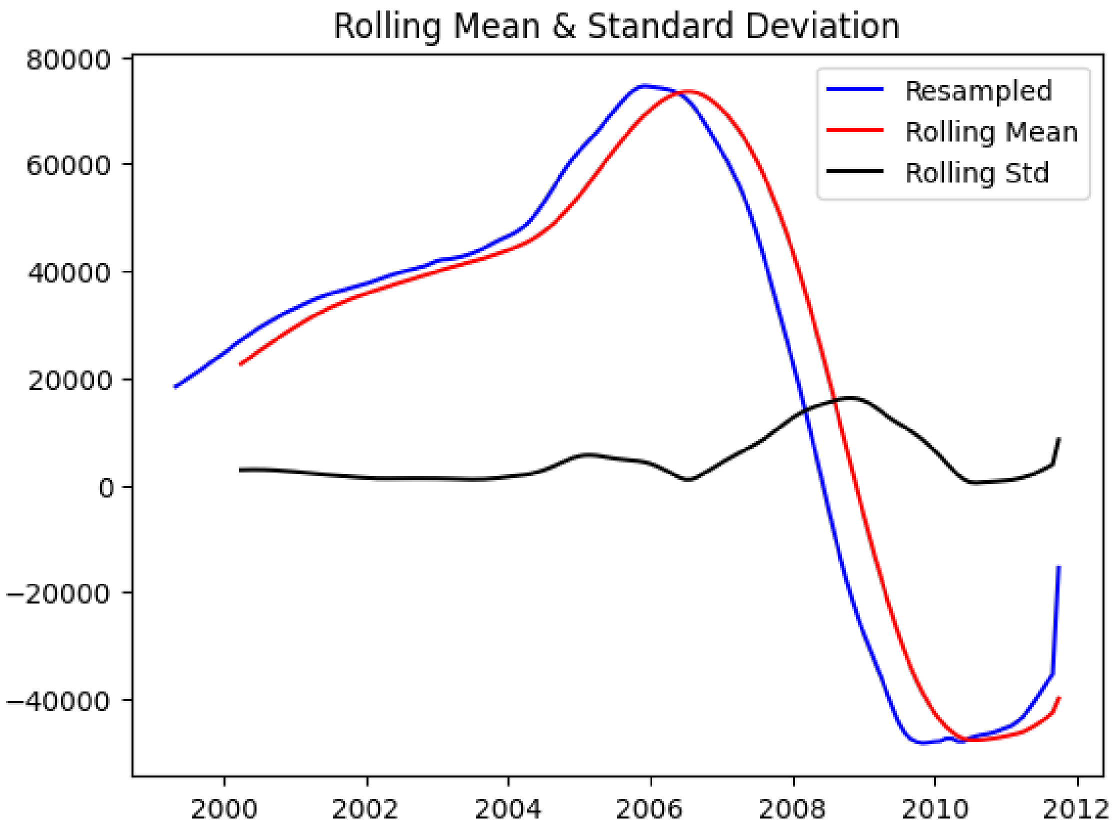



```
Results of Dickey-Fuller Test:
Test Statistic                  -2.786414
p-value                          0.060241
#Lags Used                       3.000000
Number of Observations Used    158.000000
Critical Value (1%)             -3.472431
Critical Value (5%)             -2.880013
Critical Value (10%)            -2.576619
dtype: float64
```

The p value is still greater than 0.05 showing that the data is not stationary.

In [32]:

```
calculate_data_diff(train_set,36)
```

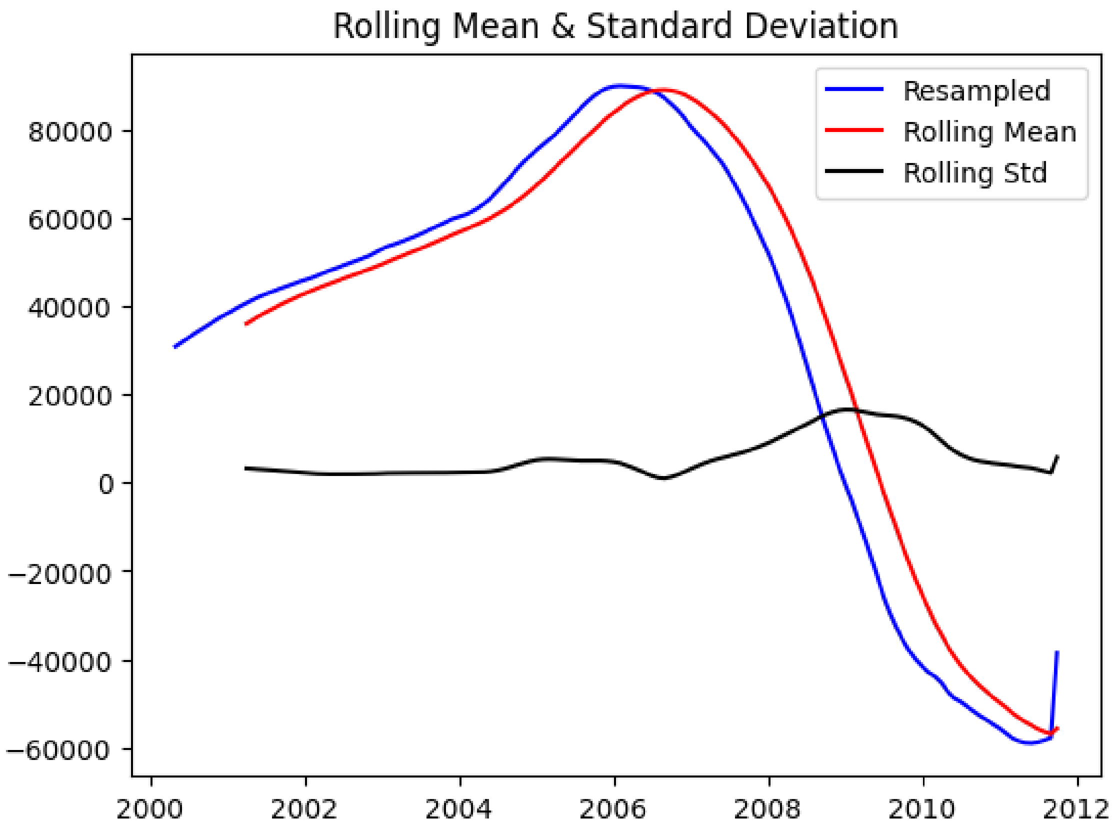



```
Results of Dickey-Fuller Test:
Test Statistic                  -2.473729
p-value                          0.121998
#Lags Used                       1.000000
Number of Observations Used    148.000000
Critical Value (1%)             -3.475325
Critical Value (5%)             -2.881275
Critical Value (10%)            -2.577293
dtype: float64
```

The p value is still greater than 0.05 showing that the data is not stationary.

In [33]:

```
calculate_data_diff(train_set,48)
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -3.123698
p-value                          0.024848
#Lags Used                       1.000000
Number of Observations Used    136.000000
Critical Value (1%)             -3.479372
Critical Value (5%)             -2.883037
Critical Value (10%)            -2.578234
dtype: float64
```

The p value is now less than 0.05 showing that the data is finally stationary.

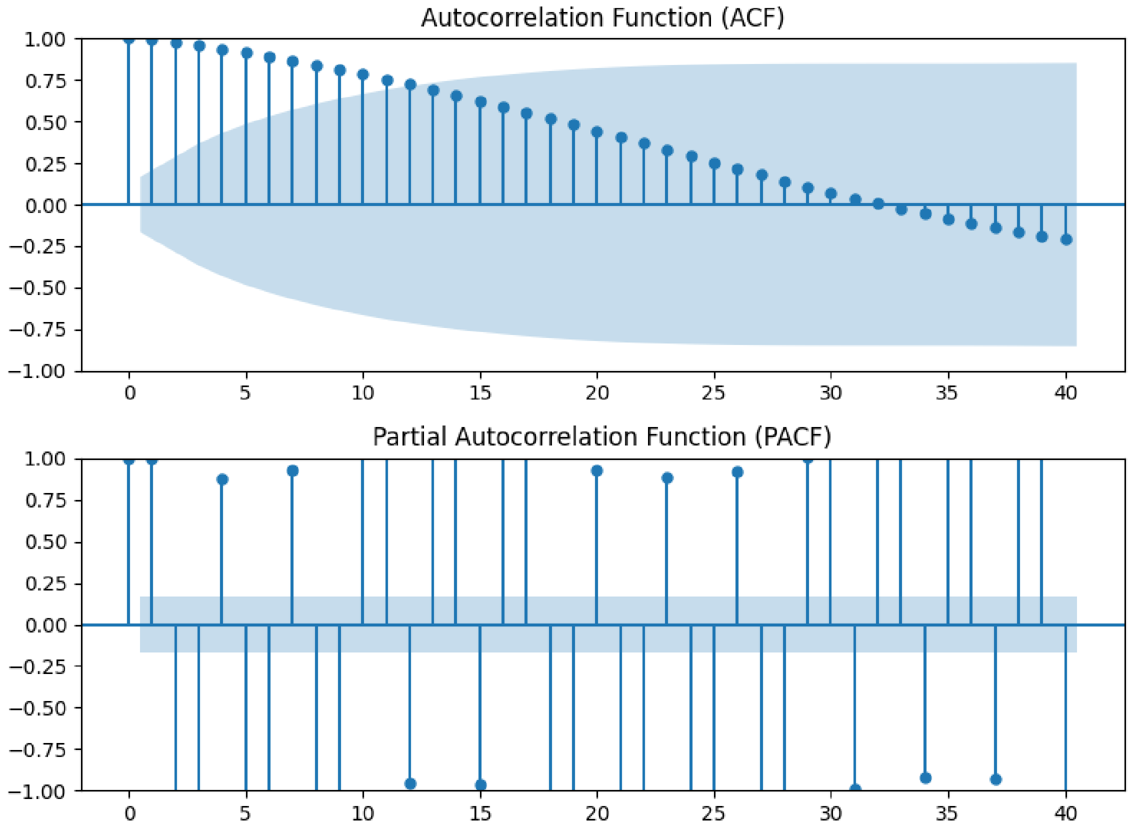
In [34]:

```
data_diff = train_set.diff(periods=48).dropna()
```

5.1.5 Plotting ACF and PACF

In [35]:

```
# Plot the PACF
# Plot the ACF
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf

def plot_acf_pacf(data):
    fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(8, 6))
    plot_acf(data, ax=axes[0], lags=40)
    plot_pacf(data, ax=axes[1], lags=40)
    axes[0].set_title('Autocorrelation Function (ACF)', fontsize=12)
    axes[1].set_title('Partial Autocorrelation Function (PACF)', fontsize=12)
    plt.tight_layout()
    plt.show()

plot_acf_pacf(data_diff)
```



From the acf plot, lags between 1 and 14 are in the statistically significant region meaning time periods within that span can affect present values. The plot shows a significant peak at a particular lag and decays exponentially afterward suggesting the presence of a seasonal pattern and the presence of an autoregressive (AR) process.

From the pacf plot it shows significant spikes at multiple lags but decays afterward, it suggests the presence of a mixed autoregressive-moving average (ARMA) process.

From the above we conclude that some of the models we'll fit are AR model and AR(I)MA model.

5.2 Building Models

As seen earlier, the acf plot suggests the presence of an autoregressive (AR) process thus our baseline

5.2.1 AR Model(Base Model)

In [36]:

```
train_values = data_diff['value']

# Define the order of the autoregressive model
order = 1
from statsmodels.tsa.ar_model import AutoReg

# Fit the autoregressive model
model = AutoReg(train_values, order)
model_fit = model.fit()
model_fit.summary()
```

Out[36]:

AutoReg Model Results

Dep. Variable:	value	**No. Observations:**	138
Model:	AutoReg(1)	**Log Likelihood**	-1267.409
Method:	Conditional MLE	**S.D. of innovations**	2520.527
Date:	Wed, 21 Jun 2023	**AIC**	2540.818
Time:	16:20:28	**BIC**	2549.578
Sample:	05-31-2000	**HQIC**	2544.378
	- 09-30-2011		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-752.9139	266.424	-2.826	0.005	-1275.096	-230.732
value.L1	1.0072	0.005	219.782	0.000	0.998	1.016

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	0.9928	+0.0000j	0.9928	0.0000

5.2.2 ARIMA Model

In [37]:

```
# Fit an ARMA model
from statsmodels.tsa.arima.model import ARIMA

mod_arma = ARIMA(train_values, order=(1,0,1))
res_arma = mod_arma.fit()

# Print out summary information on the fit
print(res_arma.summary())
```

```
                                   SARIMAX Results
==================================================================================
===========
Dep. Variable:                  value   No. Observations:
138
Model:                 ARIMA(1, 0, 1)   Log Likelihood
-1242.011
Date:                Wed, 21 Jun 2023   AIC
2492.021
Time:                        16:20:29   BIC
2503.730
Sample:                    04-30-2000   HQIC
2496.779
                         - 09-30-2011
Covariance Type:                  opg
==================================================================================
===========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------------------
-----------
const        3.37e+04   8.28e+04      0.407      0.684   -1.29e+05
1.96e+05
ar.L1          0.9960      0.006    166.077      0.000       0.984
1.008
ma.L1          0.9745      0.117      8.344      0.000       0.746
1.203
sigma2      3.598e+06   5.14e+04     70.045      0.000     3.5e+06
3.7e+06
==================================================================================
================
Ljung-Box (L1) (Q):                  12.44   Jarque-Bera (JB):
27100.44
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):              69.79   Skew:
6.88
Prob(H) (two-sided):                  0.00   Kurtosis:
70.26
==================================================================================
================

Warnings:
[1] Covariance matrix calculated using the outer product of gradien
ts (complex-step).
[2] Covariance matrix is singular or near-singular, with condition
number 2.34e+16. Standard errors may be unstable.
```

Between AR(1) and ARMA(1,1), ARMA(1,1) has the lowest AIC AND BIC. Let's use auto arima to determine the best order.

5.2.3 Auto-ARIMA to determine best order.

In [51]:

```
from pmdarima.arima import auto_arima
stepwise_model = auto_arima(train_values, start_p=1, d=0, start_q=1,
                            max_p=10, max_d=0, max_q = 13,
                            seasonal=True, stationary=False,
                            information_criterion='aic',
                            stepwise=False, suppress_warning=True)
stepwise_model.fit(train_values)
order = stepwise_model.order
print("Order (p, d, q):", order)
```

```
Order (p, d, q): (2, 0, 0)
```

In [52]:

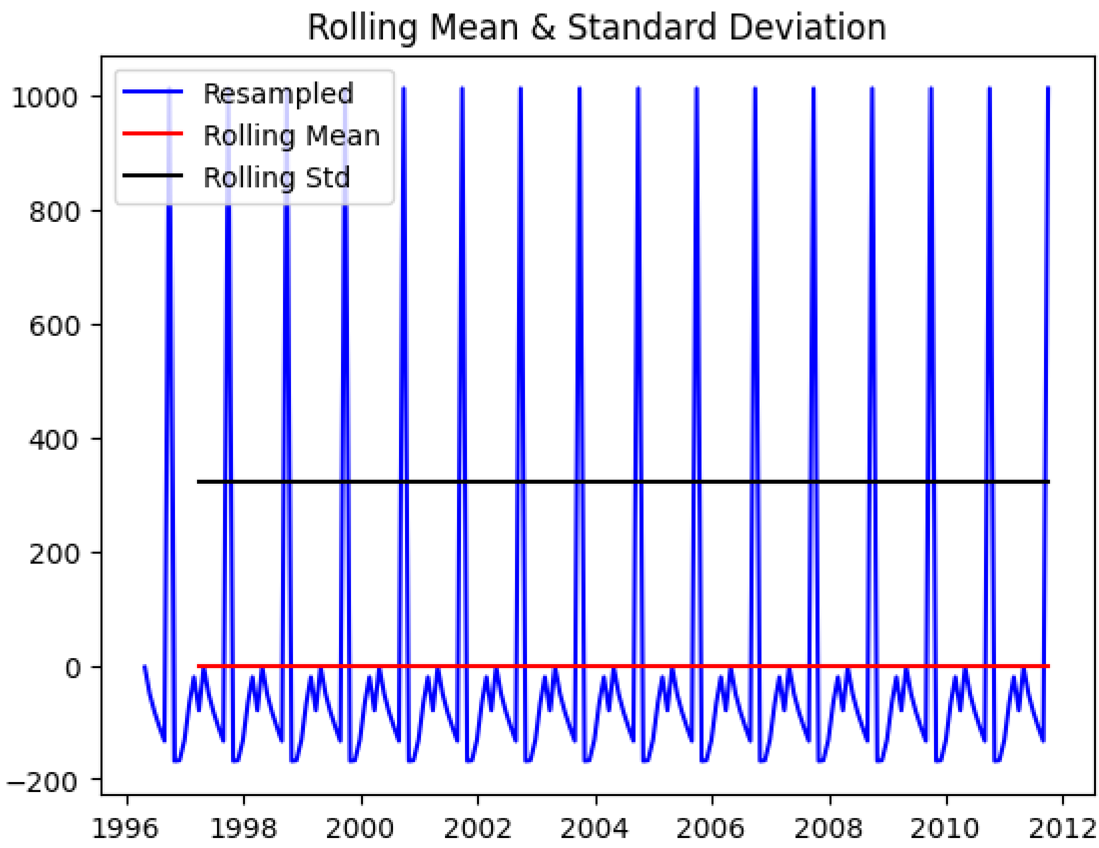
```
mod_arma = ARIMA(train_values, order=(2, 0, 0))
res_arma = mod_arma.fit()

# Print out summary information on the fit
print(res_arma.summary())
```

```
                               SARIMAX Results
==============================================================================
===========
Dep. Variable:                  value   No. Observations:
138
Model:                 ARIMA(2, 0, 0)   Log Likelihood
-1216.079
Date:                Wed, 21 Jun 2023   AIC
2440.157
Time:                        22:23:36   BIC
2451.866
Sample:                    04-30-2000   HQIC
2444.916
                         - 09-30-2011
Covariance Type:                  opg
==============================================================================
===========
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----------
const       3.37e+04   1.82e-08   1.85e+12      0.000    3.37e+04
3.37e+04
ar.L1         1.9546      0.053     36.769      0.000       1.850
2.059
ar.L2        -0.9622      0.053    -18.184      0.000      -1.066
-0.858
sigma2     2.435e+06   1.19e-08   2.04e+14      0.000    2.43e+06
2.43e+06
==============================================================================
================
Ljung-Box (L1) (Q):                   0.05   Jarque-Bera (JB):
84540.49
Prob(Q):                              0.83   Prob(JB):
0.00
Heteroskedasticity (H):             332.50   Skew:
10.71
Prob(H) (two-sided):                  0.00   Kurtosis:
122.35
==============================================================================
================

Warnings:
[1] Covariance matrix calculated using the outer product of gradien
ts (complex-step).
[2] Covariance matrix is singular or near-singular, with condition
number 3.12e+29. Standard errors may be unstable.
```

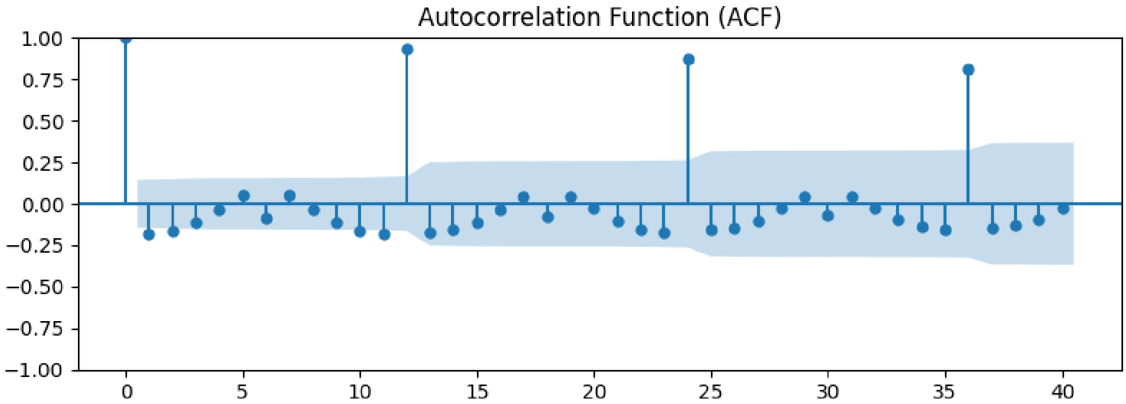
5.2.4 SARIMA

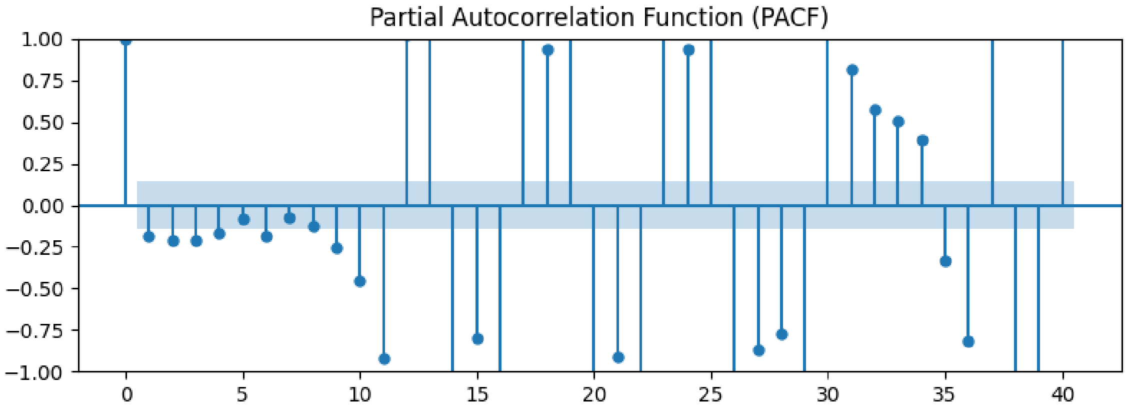
In [39]:

```
result = seasonal_decompose(train_set, model='additive',extrapolate_trend='freq')
seasonal = result.seasonal
stationarity_check(seasonal)
plot_acf_pacf(seasonal)
```




```
Results of Dickey-Fuller Test:
Test Statistic                 -1.045709e+15
p-value                         0.000000e+00
#Lags Used                      1.300000e+01
Number of Observations Used     1.720000e+02
Critical Value (1%)            -3.468952e+00
Critical Value (5%)            -2.878495e+00
Critical Value (10%)           -2.575809e+00
dtype: float64
```


Autocorrelation Function (ACF)
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
35
40


Partial Autocorrelation Function (PACF)
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
35
40

In [40]:

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
model_seasonal = SARIMAX(train_values, order=(4,0,2), seasonal_order=(4,0,2,12))
model_fit_seaonal = model_seasonal.fit()
model_fit_seaonal.summary()
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =           13     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  6.57149D+02    |proj g|=  1.29301D+03

 This problem is unconstrained.


At iterate    5    f=  9.51274D+00    |proj g|=  7.05494D-01

At iterate   10    f=  8.91001D+00    |proj g|=  5.99975D-02

At iterate   15    f=  8.83060D+00    |proj g|=  8.26675D-03

At iterate   20    f=  8.82743D+00    |proj g|=  3.98014D-03

At iterate   25    f=  8.82599D+00    |proj g|=  4.55485D-04

At iterate   30    f=  8.82593D+00    |proj g|=  1.92707D-03

At iterate   35    f=  8.82574D+00    |proj g|=  2.03400D-04

At iterate   40    f=  8.82574D+00    |proj g|=  2.29462D-04

At iterate   45    f=  8.82573D+00    |proj g|=  9.44331D-04

At iterate   50    f=  8.82571D+00    |proj g|=  5.96706D-04

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
   13     50     55      1     0     0   5.967D-04   8.826D+00
  F =   8.8257124868072339

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT
```

```
/home/moringa/.local/lib/python3.8/site-packages/statsmodels/base/m
odel.py:604: ConvergenceWarning: Maximum Likelihood optimization fa
iled to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

Out[40]:

SARIMAX Results

Dep. Variable:	value	**No. Observations:**	138
Model:	SARIMAX(4, 0, 2)x(4, 0, 2, 12)	**Log Likelihood**	-1217.948
Date:	Wed, 21 Jun 2023	**AIC**	2461.897
Time:	16:20:53	**BIC**	2499.951
Sample:	04-30-2000	**HQIC**	2477.361
	- 09-30-2011		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.1950	25.878	0.008	0.994	-50.525	50.915
ar.L2	1.6315	31.131	0.052	0.958	-59.384	62.647
ar.L3	-0.0475	19.604	-0.002	0.998	-38.471	38.376
ar.L4	-0.7966	20.424	-0.039	0.969	-40.828	39.235
ma.L1	1.7968	24.986	0.072	0.943	-47.174	50.768
ma.L2	0.8883	19.943	0.045	0.964	-38.199	39.976
ar.S.L12	-0.0333	23.116	-0.001	0.999	-45.339	45.272
ar.S.L24	0.8398	27.816	0.030	0.976	-53.679	55.358
ar.S.L36	0.0588	7.816	0.008	0.994	-15.260	15.377
ar.S.L48	-0.0621	7.708	-0.008	0.994	-15.170	15.046
ma.S.L12	-0.0022	24.369	-9.05e-05	1.000	-47.764	47.760
ma.S.L24	-0.7289	26.866	-0.027	0.978	-53.385	51.927
sigma2	2.593e+06	0.002	1.64e+09	0.000	2.59e+06	2.59e+06

Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	91976.33
Prob(Q):	0.90	**Prob(JB):**	0.00
Heteroskedasticity (H):	114.02	**Skew:**	11.06
Prob(H) (two-sided):	0.00	**Kurtosis:**	127.53

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 6.27e+24. Standard errors may be unstable.

6. Evaluation of Models

Our success metric is the Root Mean Squared Error.

In [41]:

```
from sklearn.metrics import mean_squared_error

test= test_set.resample('M').mean().fillna(method='ffill')

def check_rmse(model, test):
  n_test = test.shape[0]
  pred = model.forecast(steps=n_test)
  rmse = np.sqrt(mean_squared_error(test, pred))
  return np.round(rmse, 4)

## sarima model
check_rmse(model_fit_seaonal, test)
```

Out[41]:

```
210985.1432
```

In [42]:

```
## arma model
check_rmse(res_arma, test)
```

Out[42]:

```
209344.1877
```

In [43]:

```
## arma model
check_rmse(model_fit, test)
```

Out[43]:

```
335359.5424
```

The ARMA model has the lowest AIC, BIC and RMSE hence we will use this model for forecasting.

7. Forecasting

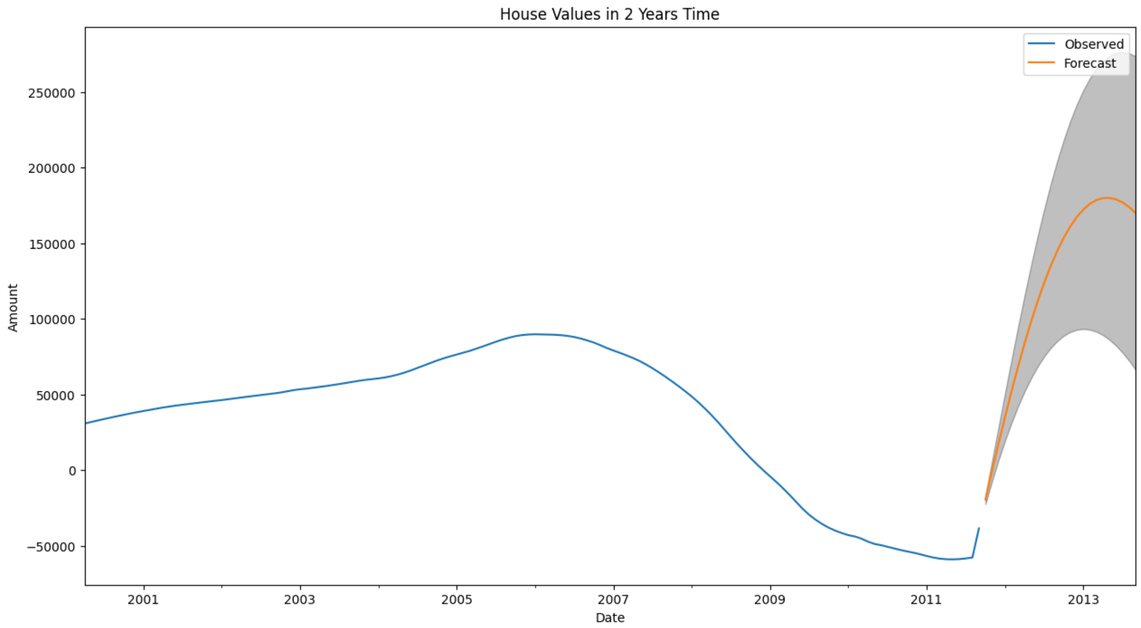
7.1 Two Year Forecast

In [44]:

```
def plot_forecast(data_diff, results, steps):
    # Get the forecast
    pred_fut = results.get_forecast(steps=steps)
    # Get confidence intervals of forecasts
    pred_ci = pred_fut.conf_int()
    # Create the plot
    ax = data_diff.plot(label='Observed', figsize=(15, 8))
    pred_fut.predicted_mean.plot(ax=ax, label='Forecast')
    ax.fill_between(pred_ci.index, pred_ci.iloc[:, 0], pred_ci.iloc[:, 1], color=
    # Adjust the title based on the steps parameter
    years = int(steps / 12)
    ax.set_title(f'House Values in {years} Years Time')
    # Set the labels for x and y axes
    ax.set_xlabel('Date')
    ax.set_ylabel('Amount')
    # Display the legend and show the plot
    plt.legend()
    plt.show()
```

In [48]:

```
plot_forecast(train_values, res_arma, 24)
```

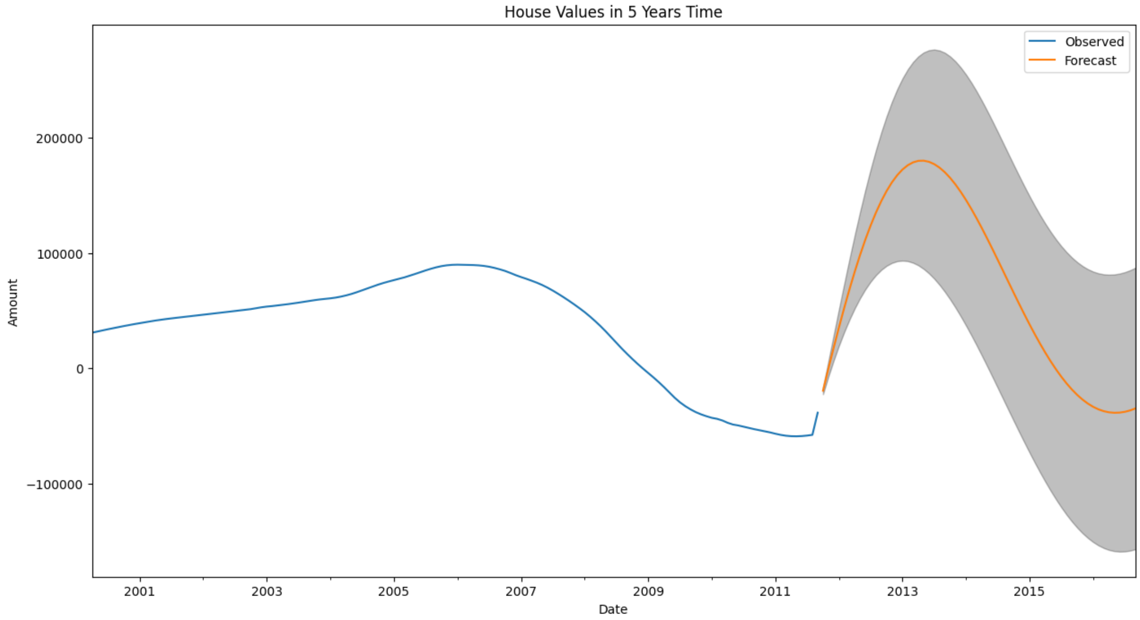


The model predicts a high increment in the house prices in the next 1 year

7.2 Five Year Forecast

In [49]:

```
plot_forecast(train_values, res_arma, 60)
```

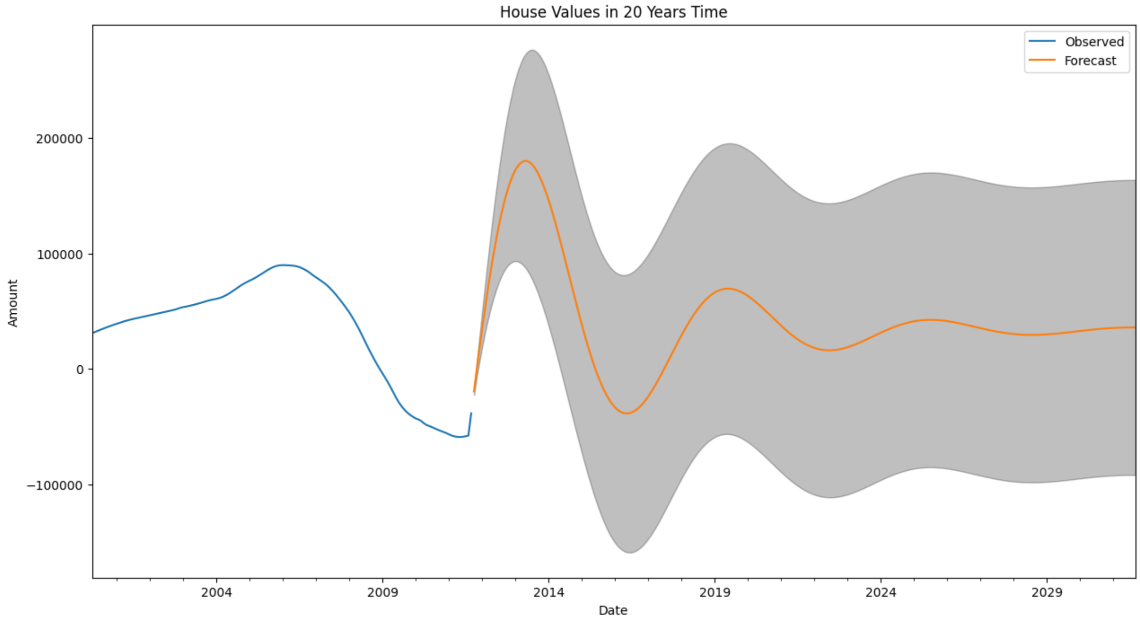


The model predicts a high increment in the house prices in the next 1 year followed by a decrement in the next 3 years.

7.3 Twenty Year Forecast

In [50]:

```
plot_forecast(train_values, res_arma, 240)
```



The model anticipates a significant rise in house prices within the next year, followed by a substantial decline over the course of the following three years. Subsequently, there is projected to be a moderate increase in prices for the subsequent three years,

followed by a smaller decrease over the next four years. Finally, the subsequent years are expected to experience minimal

fluctuations in house prices.

The model anticipates a significant rise in house prices within the next year, followed by a substantial decline over the course

of the following three years. Subsequently, there is projected to be a moderate increase in prices for the subsequent three years,

followed by a smaller decrease over the next four years. Finally, the subsequent years are expected to experience minimal

fluctuations in house prices.

8. Conclusion

- The housing market in New York, Los Angeles, and other top cities, states, metros, and counties demonstrates high activity and a

  significant number of houses.
- Return on investment (ROI) and house prices exhibit positively skewed distributions, with outliers indicating higher returns and

  extremely high-priced houses.
- There is a strong positive relationship between ROI and the coefficient of variation (CV), implying that higher risk is

  associated with higher returns.
- New York consistently stands out as the location with the highest ROI across different levels of analysis, such as states,

  metros, and counties.
- San Francisco, Los Angeles, and other specific cities, states, metros, and counties offer a combination of high ROI and lower

  risk, making them attractive for real estate investment.

9. Challenges:

- Data availability and quality may pose challenges in obtaining comprehensive and accurate information on housing markets across

  different locations.
- Market dynamics and trends can change rapidly, so it is important to regularly update and reassess your analysis and

  recommendations to account for any shifts in the housing market.

- Economic and regulatory factors can impact the housing market, requiring careful monitoring and adaptation of investment

  strategies.
- Local market knowledge and understanding of specific nuances in each location are essential for making informed investment

  decisions.

10. Recommendations & Next Steps.

10.1 Recomendations

1. Consider investing in real estate in New York, particularly in cities like San Francisco and Los Angeles, which have shown high

   ROI and relatively lower risk.
2. Focus on short-term investment opportunities to capitalize on the predicted high increment in house prices for the next year.
3. Adopt a conservative approach during the anticipated high decrement period of the following three years and carefully assess

   market conditions before making major investments.
4. Consider selling properties at their peak value before the reduced increment period to maximize profits and mitigate potential

   losses.
5. Diversify the portfolio across different locations and property types to spread risk and minimize exposure to market downturns.
6. Monitor the real estate market closely, staying updated on trends, economic indicators, and regulatory developments to make

   informed decisions.
7. Mitigate risks through thorough due diligence, assessing property quality, and conducting proper market research.

10.2 Next Steps

- Continuously monitor and evaluate the performance of your real estate investments, adjusting your strategies as needed based on

  market conditions and changing investment goals.
- Stay updated with market trends and economic indicators that can impact the housing market, such as interest rates, employment

  rates, and local infrastructure developments.
- Consider consulting with real estate professionals or experts familiar with the target markets to gain valuable insights and

  guidance for investment decisions.
- Evaluate the potential rental market in the selected locations, as rental income can contribute to overall returns and provide

stability during market fluctuations.