TIME SERIES FOR PREDICTING HOUSE PRICES

Authors

- Emily Njue
- Nelson Kibet Kemboi
- Brian Muli
- Joel Omondi
- Ian Tulienge
- Lavender Echesa

Github URL: https://github.com/Lawez/phase-4-group-project (https://github.com/Lawez/phase-4-group-project)

Trello link: https://trello.com/b/aI7c1mO8/phase-4-project (https://trello.com/b/aI7c1mO8/phase-4-project)

Powerpoint link: https://www.canva.com/design/DAFgoWWoskw/FG_bO59PhIOkzW-1KxoeOA/edit?utm_content=DAFgoWWoskw&utm_campaign=designshare&utm_medium=link2&utm_source=sh (https://www.canva.com/design/DAFgoWWoskw/FG_bO59PhIOkzW-1KxoeOA/edit?utm_content=DAFgoWWoskw&utm_campaign=designshare&utm_medium=link2&utm_source=sh

BUSINESS OVERVIEW

INTRODUCTION

In this project, we will be acting as consultants for a fictional real estate investment firm. The firm is interested in identifying the top 5 best zip codes to invest in for maximum profitability and minimal risk. Our task is to leverage time series modeling techniques to forecast real estate prices for various zip codes using the Zillow dataset.

PROBLEM STATEMENT

The investment firm wants to determine the top 5 zip codes that offer the best investment opportunities. We need to consider factors such as profitability, risk, market stability, and specific investment objectives before making any investment decisions.

OBJECTIVES

The objective of this project is:

1. To provide a recommendation for the top 5 best zip codes for a real estate investment firm to invest in.

2. What are the historical trends and patterns in real estate prices for different zip codes?
3. How do different zip codes compare in terms of risk and return on investment?

ABOUT THE DATA

This dataset was obtained from Zillow Research (https://www.zillow.com/research/data/).

The dataset provided is the Zillow dataset, which includes information on real estate prices for various zip codes. Each row represents a specific zip code, and the columns contain monthly price data from April 1996 to April 2018. The dataset also includes additional information such as region IDs, city, state, metro area, county name, and size rank. We will utilize this dataset to build our time series models and make informed investment recommendations.

The Metrics of success include:

- This is where we use measures such as return on investments (ROI) as well as risk assesment to determine whether the project has met the requirements.
- Return on investments measures the returns against the initial cost. The higher the difference the more beneficial the investment is. The aim is to have a high value as possible here in order to maximize profitability
- Another measure is risk assesment as we need to determine the investments with the least risks because this will translate to higher returns. The lower the probable risks the better for our data

In [564]:
```
#importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
# Ignore all warnings
warnings.filterwarnings("ignore")


import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from matplotlib.pylab import rcParams
from statsmodels.tsa.stattools import adfuller
from matplotlib.dates import AutoDateLocator, ConciseDateFormatter
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARMA
import pmdarima as pm
from pmdarima.arima import auto_arima
from pmdarima import model_selection
from sklearn.metrics import mean_squared_error
from statsmodels.graphics.tsaplots import plot_acf
```

In [565]:
```
#Loading the dataset
data = pd.read_csv('zillow_data.csv')
data.head()
```

Out[565]:

	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 [566]:
```
#checking columns
data.columns
```

Out[566]:
```
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)
```

In [567]:
```
# A function to print the shape of our datasets
def print_dataset_shape(*datasets):
    """
    Prints the shape of one or more datasets (number of rows and columns).
    Assumes datasets are in a Pandas DataFrame format.
    """
    for idx, dataset in enumerate(datasets):
        print(f"Dataset {idx + 1} - Number of rows: {dataset.shape[0]}")
        print(f"Dataset {idx + 1} - Number of columns: {dataset.shape[1]}")
# print the shape of our dataset
print_dataset_shape(data)
```

```
Dataset 1 - Number of rows: 14723
Dataset 1 - Number of columns: 272
```

In [568]:
```
#getting the info of dataset
data.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
```

Data Understanding

In [569]:
```
def get_datetimes(df):
    """
    Takes a dataframe:
    returns only those column names that can be converted into datetime objects
    as datetime objects.
    NOTE number of returned columns may not match total number of columns in pa
    """

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

out of top 50 sizerank:

- find top 5 ROI value today / value original * zip codes
- forecast 3 year investment horizon for each zipcode
- refine model
- choose best zipcode to invest

In [570]:
```
#select top 50 sized zipcodes according to sizerank
top_50_size = data.iloc[:50]
top_50_size
```

Out[570]:

	RegionID	RegionName	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
5	91733	77084	Houston	TX	Houston	Harris	6	95000.0
6	61807	10467	New York	NY	New York	Bronx	7	152900.0
7	84640	60640	Chicago	IL	Chicago	Cook	8	216500.0
8	91940	77449	Katy	TX	Houston	Harris	9	95400.0
9	97564	94109	San Francisco	CA	San Francisco	San Francisco	10	766000.0
10	62037	11226	New York	NY	New York	Kings	11	162000.0
11	71831	32162	The Villages	FL	The Villages	Sumter	12	101000.0
12	62087	11375	New York	NY	New York	Queens	13	252400.0
13	62045	11235	New York	NY	New York	Kings	14	190500.0
14	74101	37013	Nashville	TN	Nashville	Davidson	15	112400.0
15	96107	90250	Hawthorne	CA	Los Angeles-Long Beach-Anaheim	Los Angeles	16	152500.0
16	84646	60647	Chicago	IL	Chicago	Cook	17	122700.0
17	74242	37211	Nashville	TN	Nashville	Davidson	18	97900.0
18	92593	78660	Pflugerville	TX	Austin	Travis	19	138900.0
19	84620	60618	Chicago	IL	Chicago	Cook	20	142600.0
20	61625	10011	New York	NY	New York	New York	21	NaN
21	61703	10128	New York	NY	New York	New York	22	3676700.0
22	92036	77573	League City	TX	Houston	Galveston	23	141400.0
23	92045	77584	Pearland	TX	Houston	Brazoria	24	138500.0
24	69816	28269	Charlotte	NC	Charlotte	Mecklenburg	25	126100.0
25	93123	79912	El Paso	TX	El Paso	El Paso	26	119700.0
26	92515	78572	Mission	TX	McAllen	Hidalgo	27	69900.0
27	97771	94565	Pittsburg	CA	San Francisco	Contra Costa	28	139200.0
28	71067	30349	Riverdale	GA	Atlanta	Clayton	29	90900.0
29	66126	20002	Washington	DC	Washington	District of Columbia	30	94300.0

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04
30	96027	90046	Los Angeles	CA	Los Angeles-Long Beach-Anaheim	Los Angeles	31	340600.0
31	92271	78130	New Braunfels	TX	San Antonio	Comal	32	123900.0
32	92551	78613	Cedar Park	TX	Austin	Williamson	33	169600.0
33	66133	20009	Washington	DC	Washington	District of Columbia	34	178800.0
34	90654	75052	Grand Prairie	TX	Dallas-Fort Worth	Dallas	35	100800.0
35	61802	10462	New York	NY	New York	Bronx	36	154300.0
36	61796	10456	New York	NY	New York	Bronx	37	NaN
37	74126	37042	Clarksville	TN	Clarksville	Montgomery	38	72000.0
38	91922	77429	Cypress	TX	Houston	Harris	39	149600.0
39	86026	63376	Saint Peters	MO	St. Louis	Saint Charles	40	98200.0
40	84615	60613	Chicago	IL	Chicago	Cook	41	297900.0
41	91968	77479	Sugar Land	TX	Houston	Fort Bend	42	189800.0
42	70829	30044	Lawrenceville	GA	Atlanta	Gwinnett	43	111400.0
43	89925	73099	Yukon	OK	Oklahoma City	Canadian	44	83100.0
44	91685	77036	Houston	TX	Houston	Harris	45	120400.0
45	91926	77433	Cypress	TX	Houston	Harris	46	147300.0
46	61803	10463	New York	NY	New York	Bronx	47	180100.0
47	62040	11230	New York	NY	New York	Kings	48	230100.0
48	84630	60629	Chicago	IL	Chicago	Cook	49	93400.0
49	62020	11209	New York	NY	New York	Kings	50	255700.0

50 rows × 272 columns

In [571]:
```
#calculate ROI for each zip code
top_50_size['ROI'] = (top_50_size['2018-04'] / top_50_size['1996-04']) - 1
top_50_size.head()
```

Out[571]:

	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 × 273 columns

In [572]:
```
#select 10 largest ROI
top_50_size['ROI'].nlargest(n=10)
```

Out[572]:
```
29    6.330859
33    5.030201
10    4.945679
30    4.774809
47    4.195133
9     3.978460
46    3.323709
12    3.297147
13    3.284514
15    3.040656
Name: ROI, dtype: float64
```

In [573]:
```
top5 = top_50_size.iloc[[29, 33, 10, 30, 47]]
top5
```

Out[573]:

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	19
29	66126	20002	Washington	DC	Washington	District of Columbia	30	94300.0	94
33	66133	20009	Washington	DC	Washington	District of Columbia	34	178800.0	179
10	62037	11226	New York	NY	New York	Kings	11	162000.0	162
30	96027	90046	Los Angeles	CA	Los Angeles-Long Beach-Anaheim	Los Angeles	31	340600.0	341
47	62040	11230	New York	NY	New York	Kings	48	230100.0	230

5 rows × 273 columns

- Once we have calculated the return on investment (ROI) for various zip codes over a historical period, we can identify the top five zip codes with the highest returns. We will now focus our analysis on these five zip codes and make projections for the next three years into the future.

EDA AND VISUALIZATION

In [574]:
```
# Reshape the dataset to long format
df_long = pd.melt(top5, id_vars=['RegionID', 'RegionName', 'City', 'State', 'M
                  var_name='time', value_name='value')

df_long['time'] = pd.to_datetime(df_long.time)

# Display the reshaped dataset
df_long
```

Out[574]:

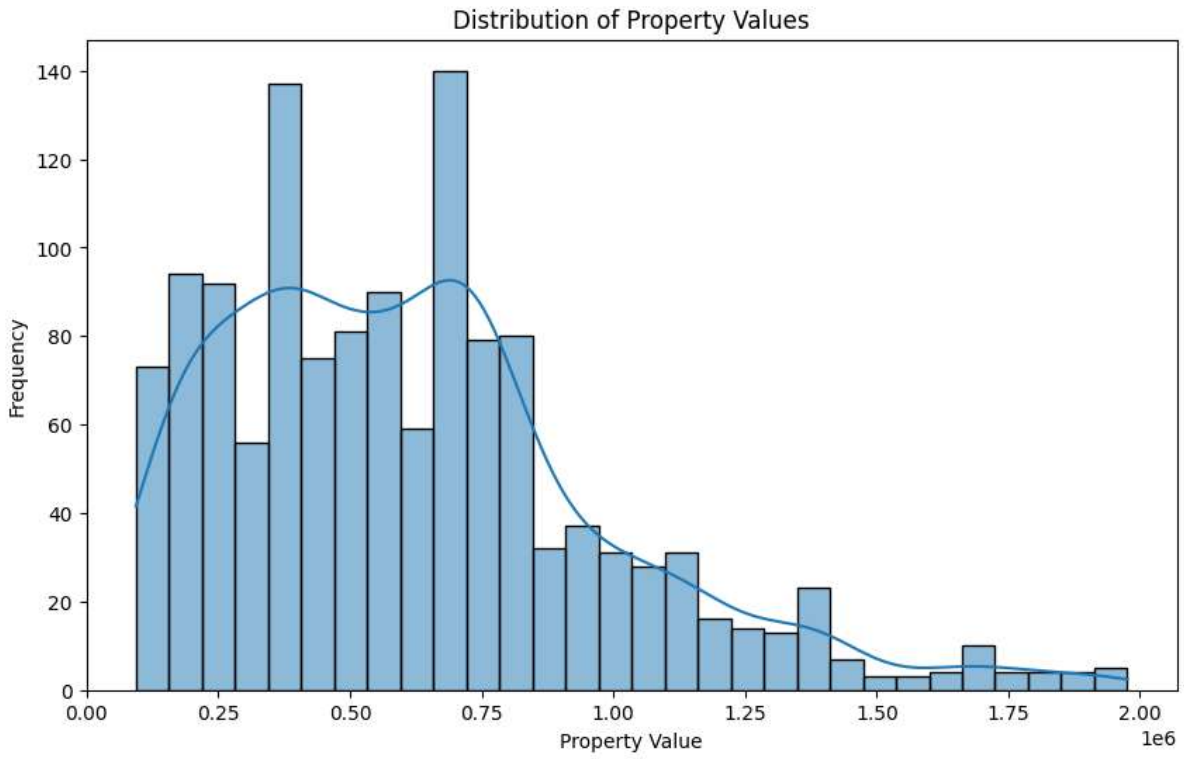
	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	ROI
0	66126	20002	Washington	DC	Washington	District of Columbia	30	6.330859
1	66133	20009	Washington	DC	Washington	District of Columbia	34	5.030201
2	62037	11226	New York	NY	New York	Kings	11	4.945679
3	96027	90046	Los Angeles	CA	Los Angeles-Long Beach-Anaheim	Los Angeles	31	4.774809
4	62040	11230	New York	NY	New York	Kings	48	4.195133
...	...	...	...	...	...	...	...	...
1320	66126	20002	Washington	DC	Washington	District of Columbia	30	6.330859
1321	66133	20009	Washington	DC	Washington	District of Columbia	34	5.030201
1322	62037	11226	New York	NY	New York	Kings	11	4.945679
1323	96027	90046	Los Angeles	CA	Los Angeles-Long Beach-Anaheim	Los Angeles	31	4.774809
1324	62040	11230	New York	NY	New York	Kings	48	4.195133

1325 rows × 10 columns

In [575]:
```
df_long.dtypes
```

Out[575]:
```
RegionID               int64
RegionName             int64
City                  object
State                 object
Metro                 object
CountyName            object
SizeRank               int64
ROI                  float64
time          datetime64[ns]
value                float64
dtype: object
```

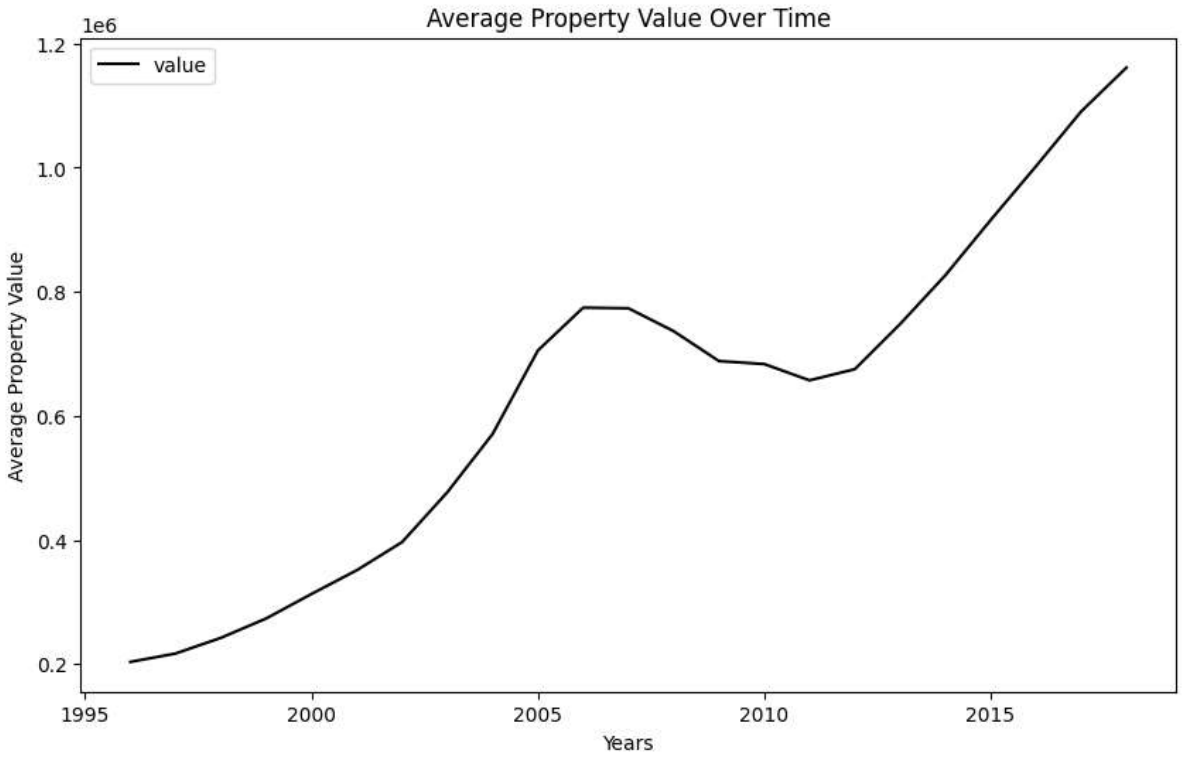
In [576]:
```
# Visualize the distribution of real estate prices
plt.figure(figsize=(10, 6))
sns.histplot(df_long['value'], bins=30, kde=True)
plt.xlabel('Property Value')
plt.ylabel('Frequency')
plt.title('Distribution of Property Values')
plt.show()
```



The distribution of real estate prices is right-skewed, with a long tail on the right side. This indicates that there are a few properties with very high prices, which pull the average price higher.

The shape of the distribution suggests that the real estate market consists of a mix of moderately priced properties and a smaller number of high-priced properties.

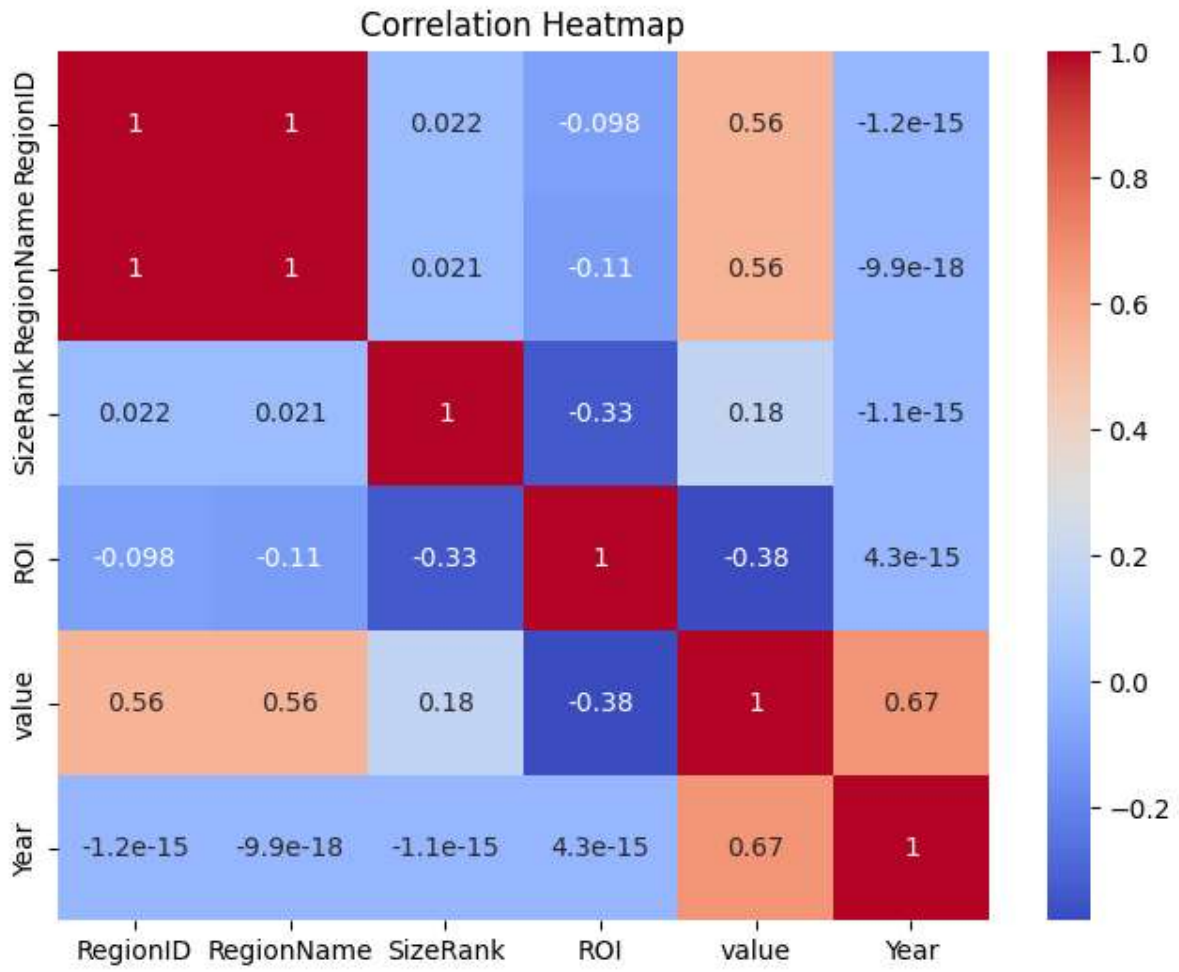
In [577]:
```
def plot_value(data):

    # Extract years from the time column
    data['Year'] = data['time'].dt.year

    # Group data by year and calculate the average value
    avg_value = data.groupby('Year')['value'].mean()

    # Plot the average value over time
    plt.figure(figsize=(10, 6))
    plt.plot(avg_value.index, avg_value, label='value', color='black')
    plt.title('Average Property Value Over Time')
    plt.xlabel('Years')
    plt.ylabel('Average Property Value')
    plt.legend()
    plt.show()

# Call the function with the DataFrame 'df_long'
plot_value(df_long)
```



- From the graph above we see an average property value increase from 1996 to 2007, where it drastically dropped to 2012 and increased from 2012 onward.

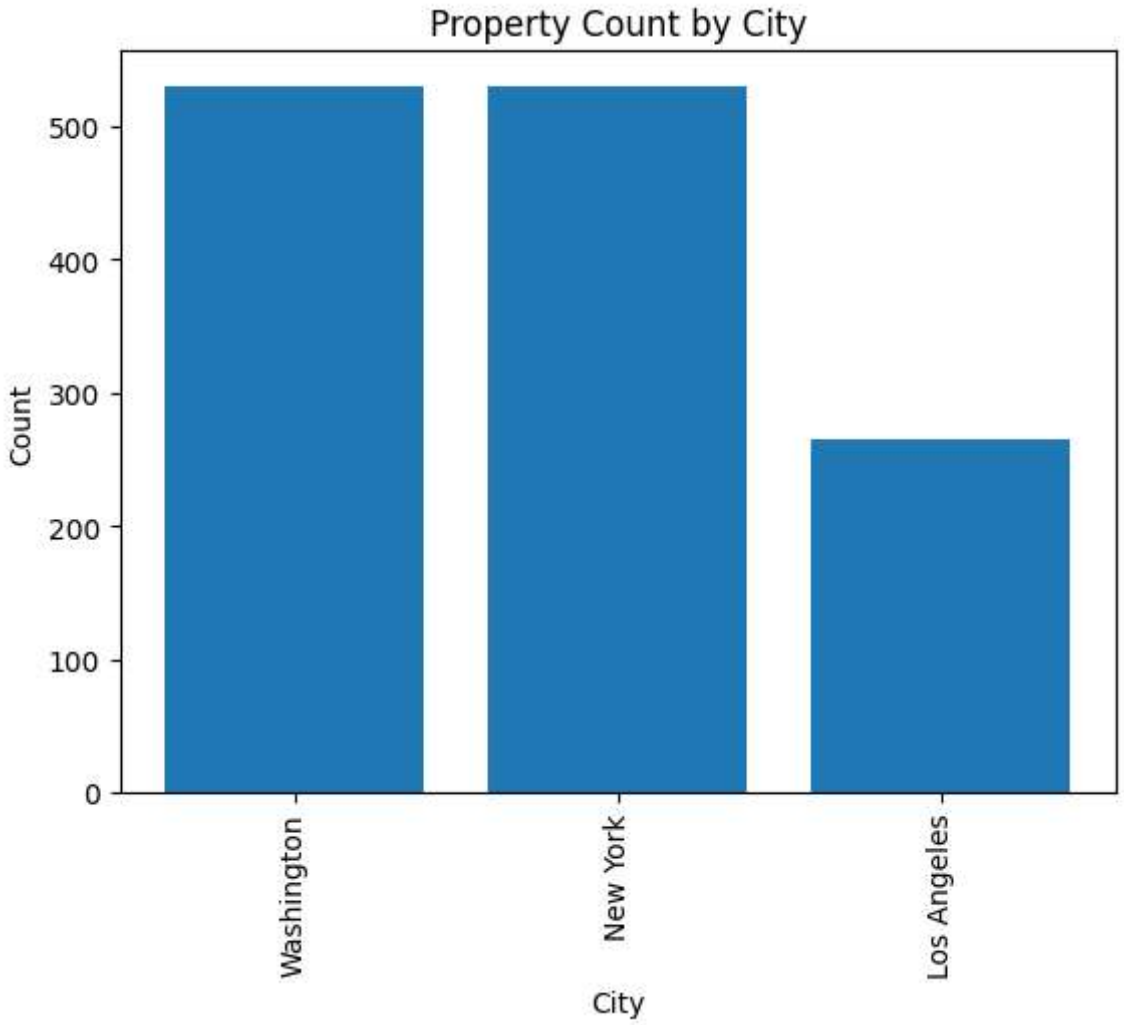
Los Angels exhibits a wider spread of property values, suggesting a greater diversity and variability in the housing market.

In [579]:
```
# Correlations
correlation_matrix = df_long.corr()
plt.figure(figsize=(8, 6))
sns.heatmap(data=correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



The correlation heatmap shows the pairwise correlation coefficients between different variables.

In [580]:
```
#Property Count by City
city_counts = df_long['City'].value_counts()
plt.bar(city_counts.index, city_counts.values)
plt.xlabel('City')
plt.ylabel('Count')
plt.title('Property Count by City')
plt.xticks(rotation=90)
plt.show()
```



The plot provides a visual comparison of the property counts among different cities, allowing us to identify cities with a higher concentration of properties.

- The number of properties varies across different cities, indicating variations in the real estate market.
- Some cities (Washington & New York) have a higher count of properties, suggesting a larger availability and demand for real estate in those areas.
- By examining the relative heights of the bars, we can gauge the magnitude of the property count disparity between cities.

DATA PREPROCESSING

- We changed our data from wide range to long format for easier analysis

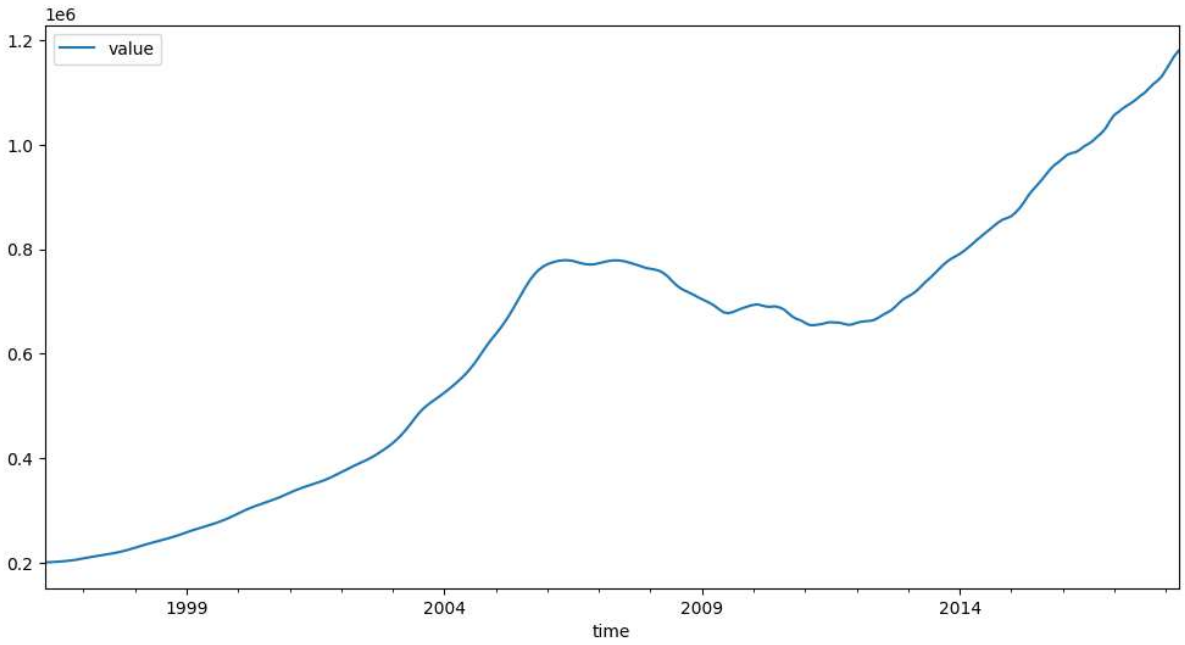
In [581]:
```
def melt_data(df):
    """
    Takes the zillow_data dataset in wide form or a subset of the zillow_datas
    Returns a long-form datetime dataframe
    with the datetime column names as the index and the values as the 'values'

    If more than one row is passes in the wide-form dataset, the values column
    will be the mean of the values from the datetime columns in all of the rows
    """

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

In [582]:
```
#melt data to reformat
melted_data = melt_data(top5)
melted_data.head()
```

Out[582]:

time	value
1996-04-01	201160.0
1996-05-01	201600.0
1996-06-01	202080.0
1996-07-01	202620.0
1996-08-01	203200.0

In [583]:
```
#ploting our melted data
melted_data.plot(figsize=(12,6));
```



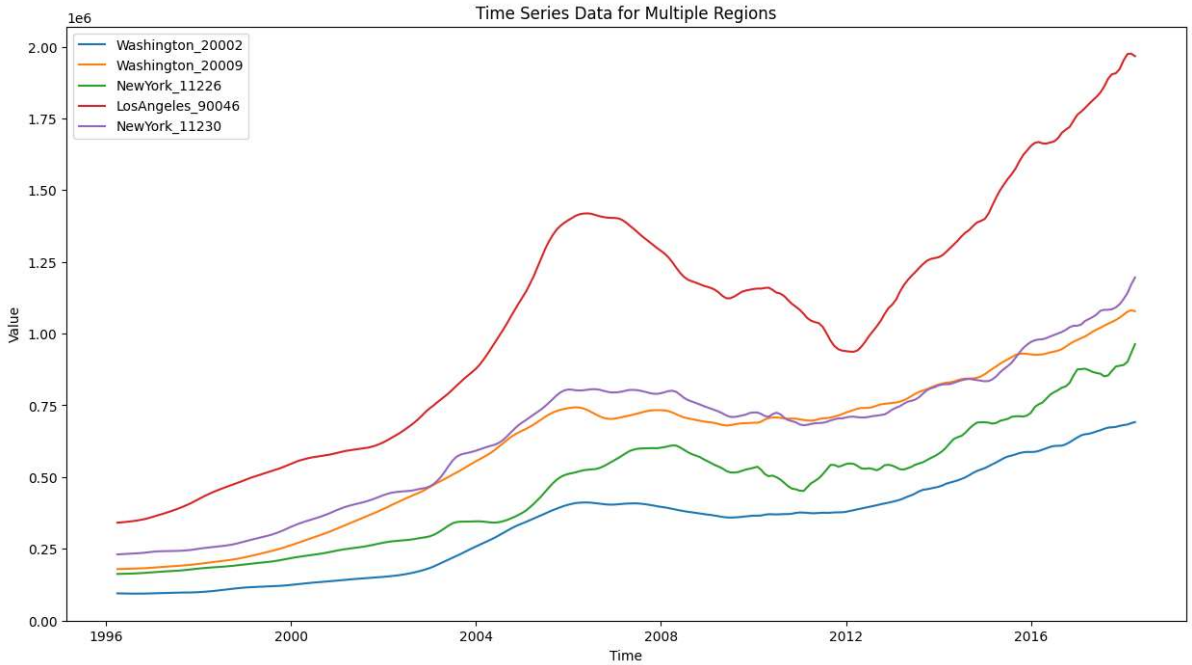
- The data reveals a consistent upward trend in housing prices for the selected zip codes. However, there are also cyclical fluctuations that align with the broader business cycle, including periods of recession such as the `post-2008 lows` and subsequent periods of growth.
- In recent years, there has been a notable surge in housing prices, which can be attributed to two main factors: unprecedented supply shortages and historically low interest rates. These factors have created a significant demand-supply imbalance, driving up prices and leading to a breakout in the housing market.

In [584]:
```
#separate each region
Washington_20002 = top5.iloc[[0]]
Washington_20009 = top5.iloc[[1]]
NewYork_11226 = top5.iloc[[2]]
LosAngeles_90046 = top5.iloc[[3]]
NewYork_11230= top5.iloc[[4]]
```

In [585]:
```
#melting top five regions
melted_r1 = melt_data(Washington_20002)
melted_r2 = melt_data(Washington_20009)
melted_r3 = melt_data(NewYork_11226)
melted_r4 = melt_data(NewYork_11230)
melted_r5 = melt_data(LosAngeles_90046)
```

In [586]:
```
#defining a function to visualize tiem series for top 5 regions
def melt_and_visualize_regions(regions):
    region_names = ["Washington_20002", "Washington_20009", "NewYork_11226", "
    num_regions = len(regions)
    fig, ax = plt.subplots(figsize=(15, 8))

    for i, region in enumerate(regions):
        melted_data = melt_data(region)
        ax.plot(melted_data)

    ax.legend(region_names[:num_regions])

    title_mapping = {
        1: "Washington_20002",
        2: "Washington_20009",
        3: "NewYork_11226",
        4: "LosAngeles_90046",
        5: "NewYork_11230"
    }

    ax.set_title("Time Series Data for Multiple Regions")

    ax.set_xlabel("Time")
    ax.set_ylabel("Value")

    plt.show()
```

In [587]:
```
#calling the function to plot the graph
regions = [Washington_20002, Washington_20009, NewYork_11226, LosAngeles_90046
melt_and_visualize_regions(regions)
```



- In each of the regions `"Washington_20002", "Washington_20009", "NewYork_11226", "LosAngeles_90046", "NewYork_11230"` , we can observe the boom and bust cycle that was previously discussed. The boom and bust cycle refers to the pattern of economic growth followed by a period of decline or recession.
- Some regions fared better than others in weathering the post-2008 lows. This can be seen by comparing the depth of the trough, which represents the lowest point in the economic cycle. Regions that experienced a lower trough were more resilient and recovered faster from the economic downturn.
- However, despite the variations in the severity of the bust cycle, all regions have shown an overall upward trend in recent years. This indicates that the economy in each region has been recovering and growing steadily. The upward trend suggests positive economic conditions and reflects the resilience and strength of these regions in bouncing back from the previous recession.

In [588]:
```
melted_data
```

Out[588]:

time	value
1996-04-01	201160.0
1996-05-01	201600.0
1996-06-01	202080.0
1996-07-01	202620.0
1996-08-01	203200.0
...	...
2017-12-01	1129180.0
2018-01-01	1141980.0
2018-02-01	1155440.0
2018-03-01	1169020.0
2018-04-01	1179000.0

265 rows × 1 columns

In [ ]:

In [ ]:

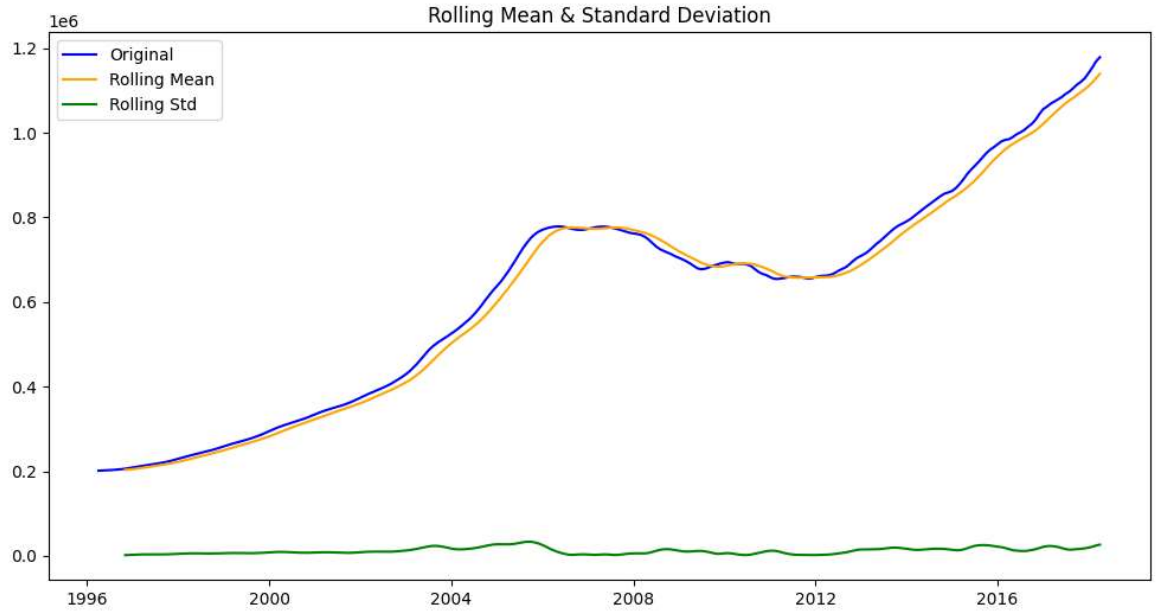
Time Series

- In order to prepare the data for the modeling process, we decided to drop the columns that would not be necessary for the modeling process and remain with the pricipal feature(time) together with the target feature(value).
- We are also going to format the time index column to represent a different frequency, that is, months. This is done to try and reduce the number or row entries in the datset.

- Time series data is observed at different points in time and can be used to forecast future values based on past observations. ARIMA (AutoRegressive Integrated Moving Average) is a commonly used model for time series forecasting. It incorporates parameters like p (auto-regressive), d (integrated), and q (moving average) to capture seasonality, trend, and noise in the data. If the time series has a seasonal component, a seasonal ARIMA model (SARIMA) is used with additional parameters P, D, and Q.

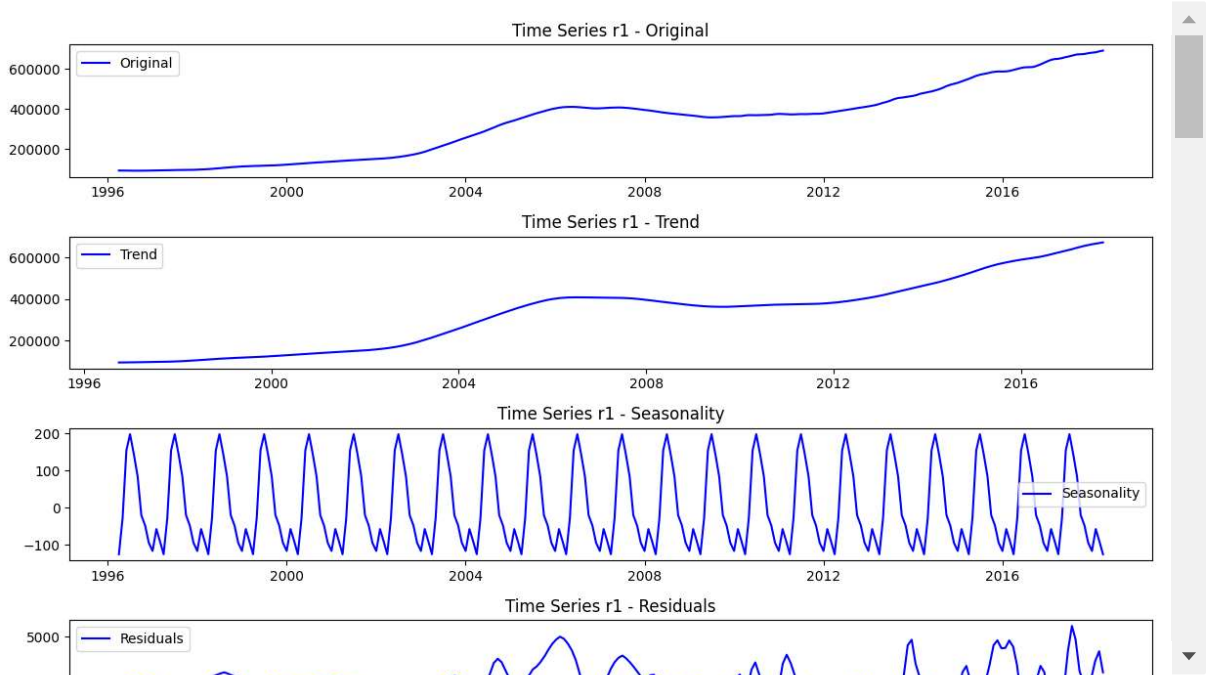
STATIONERITY

In [589]:
```
def stationarity_check(TS):
    # Calculate rolling statistics
    roll_mean = melted_data.rolling(window=8, center=False).mean()
    roll_std = melted_data.rolling(window=8, center=False).std()

    # Perform the Dickey Fuller test
    dftest = adfuller(melted_data)

    # Plot rolling statistics:
    fig = plt.figure(figsize=(12,6))
    orig = plt.plot(melted_data, color='blue',label='Original')
    mean = plt.plot(roll_mean, color='orange', label='Rolling Mean')
    std = plt.plot(roll_std, color='green', label='Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    # Print Dickey-Fuller test results
    print('Results of Dickey-Fuller Test: \n')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic', 'p-value', '#La
    for key, value in dftest[4].items():
        dfoutput['Critical Value (%s)' % key] = value
    print(dfoutput)

    # Extract the test statistic and p-value from the result
    test_statistic = dftest[0]
    p_value = dftest[1]

    # Check if the time series is stationary based on the p-value
    if p_value <= 0.05:
        print("\nThe time series is likely stationary.")
    else:
        print("\nThe time series is likely non-stationary.")

    return None

# Check stationarity for Washington_20002
stationarity_check(Washington_20002)

# Check stationarity for Washington_20009
stationarity_check(Washington_20009)

# Check stationarity for LosAngeles_90046
stationarity_check(LosAngeles_90046)

# Check stationarity for NewYork_11226
stationarity_check(NewYork_11226)

# Check stationarity for ewYork_11230
stationarity_check(NewYork_11230)
```



- The Dickey-Fuller test results indicate that none of the time series pass the test, suggesting that they are all non-stationary. In order to apply forecasting models, such as ARIMA, it will be necessary to apply differencing and/or incorporate moving average terms to make the series stationary.
- To gain further insights into the individual time series and their underlying components, we can utilize the seasonal_decompose function. This function helps visualize the decomposition of the time series into its trend, seasonality, and residual components. By examining these components, we can better understand the patterns and fluctuations within the data.

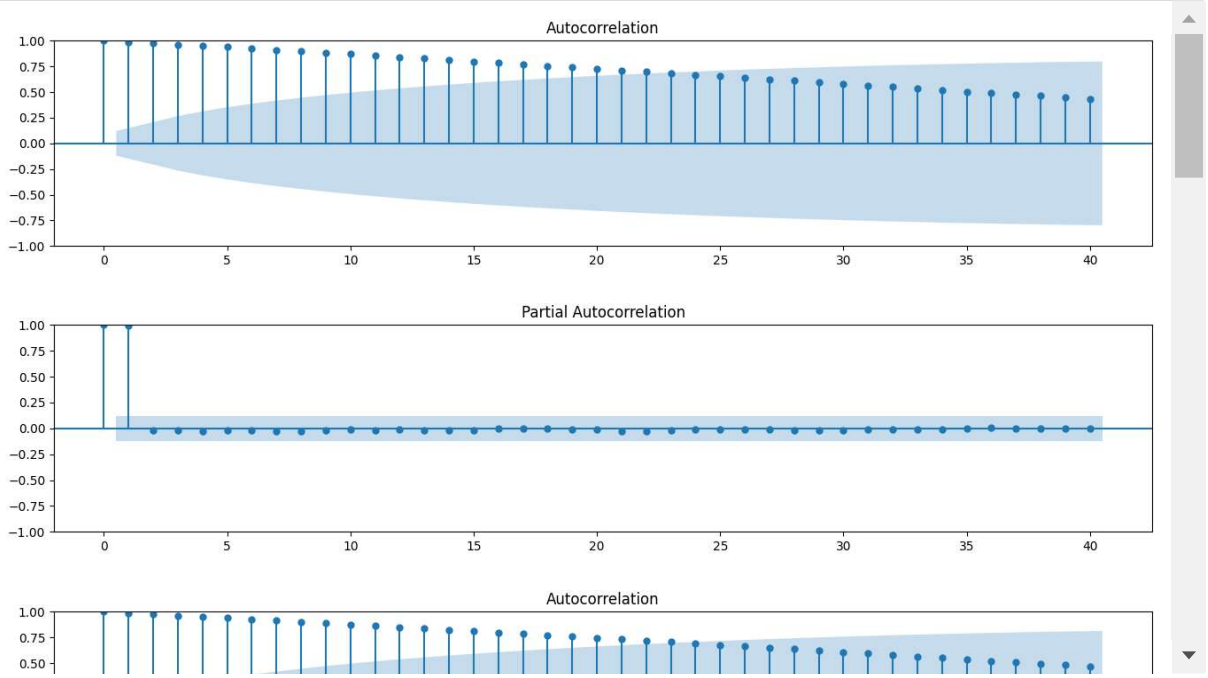
In [590]:
```
# Define a function for decomposition and plotting
def decompose_and_plot(time_series, title):
    decomposition = seasonal_decompose(time_series)
    trend = decomposition.trend
    seasonal = decomposition.seasonal
    residual = decomposition.resid

    # Plot gathered statistics
    plt.figure(figsize=(12,8))
    plt.subplot(411)
    plt.plot(time_series, label='Original', color='blue')
    plt.legend(loc='best')
    plt.title(f"{title} - Original")

    plt.subplot(412)
    plt.plot(trend, label='Trend', color='blue')
    plt.legend(loc='best')
    plt.title(f"{title} - Trend")

    plt.subplot(413)
    plt.plot(seasonal, label='Seasonality', color='blue')
    plt.legend(loc='best')
    plt.title(f"{title} - Seasonality")

    plt.subplot(414)
    plt.plot(residual, label='Residuals', color='blue')
    plt.legend(loc='best')
    plt.title(f"{title} - Residuals")

    plt.tight_layout()

# Time series r1
decompose_and_plot(melted_r1, "Time Series r1")

# Time series r2
decompose_and_plot(melted_r2, "Time Series r2")

# Time series r3
decompose_and_plot(melted_r3, "Time Series r3")

# Time series r4
decompose_and_plot(melted_r4, "Time Series r4")

# Time series r5
decompose_and_plot(melted_r5, "Time Series r5")

plt.show()
```



- All regions, including `Washington_20002`, `Washington_20009`, `NewYork_11226`, `LosAngeles_90046`, and `NewYork_11230`, demonstrate a consistent upward trend in their respective time series data. Furthermore, these regions exhibit evidence of seasonality, which manifests on an annual basis. To effectively address the combined trend and seasonality components present in the data, we will employ the autoarima package. This powerful tool automatically selects the optimal parameters for an ARIMA model, considering both the trend and seasonal patterns within the data. By leveraging the capabilities of autoarima, we can develop accurate and robust forecasting models for these regions.

In [591]:
```
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

def plot_acf_pacf(data):
    fig, ax = plt.subplots(figsize=(16, 3))
    plot_acf(data, ax=ax, lags=40)
    plt.show()

    fig, ax = plt.subplots(figsize=(16, 3))
    plot_pacf(data, ax=ax, lags=40)
    plt.show()
```

In [592]:
```
plot_acf_pacf(melted_r1)
plot_acf_pacf(melted_r2)
plot_acf_pacf(melted_r3)
plot_acf_pacf(melted_r4)
plot_acf_pacf(melted_r5)
```

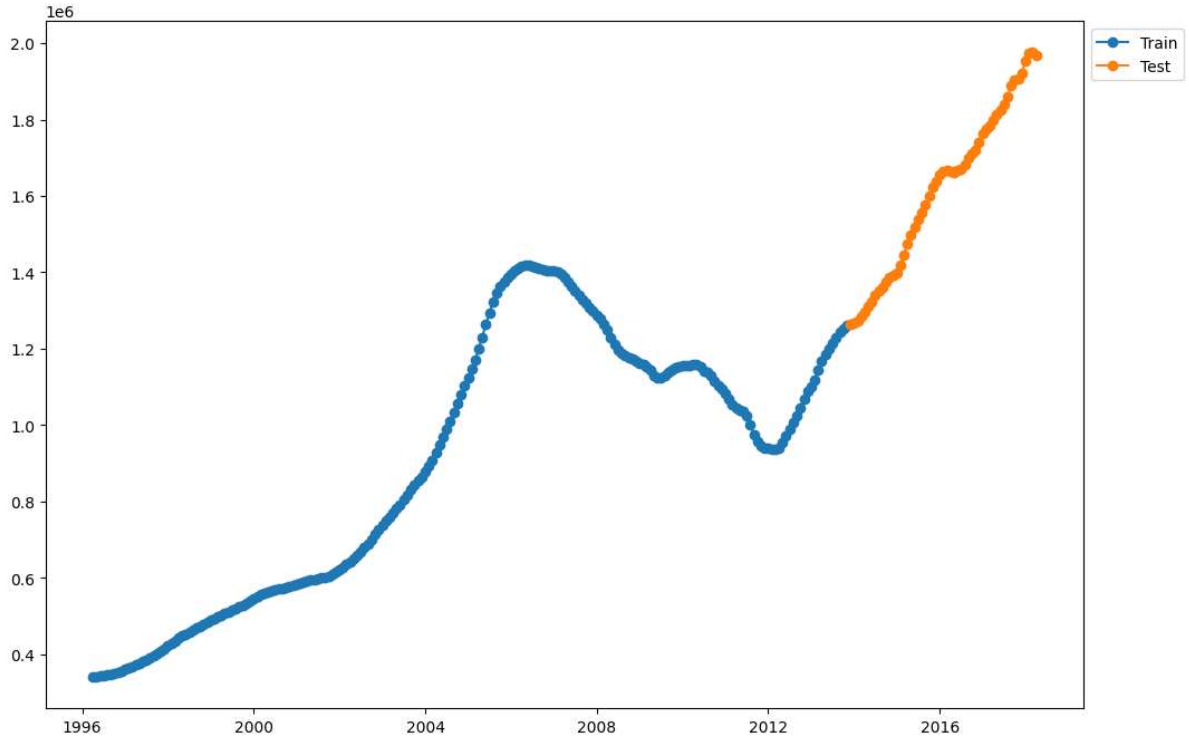


The ACF and PACF plots show that both the autocorrelation and partial autocorrelation coefficients tail off, which suggests the presence of both autoregressive (AR) and moving average (MA) components in the data. This indicates that an ARMA model, which combines both AR and MA parameters, would be suitable for modeling the data.

To determine the best parameters for the ARMA model, you can use the AutoArima package. This package performs an exhaustive stepwise search, evaluating different combinations of AR and MA parameters, to find the model with the best fit to the data. By using AutoArima, you can automate the process of selecting the optimal ARMA parameters and obtain a more accurate and reliable model for your data.

ARIMA MODELLING

In [593]:
```
# Train Test Split Index
train_size = 0.8
split_idx = round(len(melted_r5)* train_size)
split_idx

# Split
r5_train = melted_r5.iloc[:split_idx]
r5_test = melted_r5.iloc[split_idx:]

# Visualize split
fig,ax= plt.subplots(figsize=(12,8))
kws = dict(marker='o')
plt.plot(r5_train, label='Train', **kws)
plt.plot(r5_test, label='Test', **kws)
ax.legend(bbox_to_anchor=[1,1]);
```



AUTOARIMA

The AutoArima package performs a stepwise search to find the best parameters for an ARMA model based on the Akaike Information Criterion (AIC). It evaluates various combinations of AR and MA parameters, calculating the AIC for each model. The model with the lowest AIC is selected as the best-fit model. This approach balances model complexity and goodness of fit, avoiding overfitting or underfitting. The resulting model is considered the best based on AIC and can be used for analysis and forecasting.

In [594]:
```
#stepwise search of best parameters according to AIC
r5_autoarima = auto_arima(r5_train, start_p=0, start_q=0, d=None,
                          information_criterion='aic', max_d=3, max_q=3,
                          max_p=3, start_P=0, start_Q=0, D=None, max_P=3, max_
                          max_D=3, trace=True, stepwise=True

                         )
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=4523.203, Time=1.21 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=4489.163, Time=0.66 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=4496.433, Time=0.15 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=4552.877, Time=0.03 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=4482.309, Time=0.54 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=4471.360, Time=0.95 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=4447.602, Time=0.86 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=4442.232, Time=0.87 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=4440.464, Time=0.46 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=4441.827, Time=0.71 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=4482.520, Time=0.08 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=7145.320, Time=0.86 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=4439.355, Time=0.69 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=4519.564, Time=0.07 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=4508.839, Time=0.19 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=inf, Time=0.51 sec
 ARIMA(1,1,2)(0,0,0)[0]             : AIC=inf, Time=0.60 sec
 ARIMA(0,1,2)(0,0,0)[0]             : AIC=4502.421, Time=0.08 sec
 ARIMA(2,1,0)(0,0,0)[0]             : AIC=4487.365, Time=0.23 sec
 ARIMA(2,1,2)(0,0,0)[0]             : AIC=inf, Time=0.54 sec

Best model:  ARIMA(1,1,1)(0,0,0)[0]
Total fit time: 10.729 seconds
```

In [595]:
```
#printing results of our model
r5_autoarima.summary()
```

Out[595]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	212
Model:	SARIMAX(1, 1, 1)	**Log Likelihood**	-2216.677
Date:	Thu, 22 Jun 2023	**AIC**	4439.355
Time:	09:05:03	**BIC**	4449.410
Sample:	04-01-1996	**HQIC**	4443.419
	- 11-01-2013		
Covariance Type:	opg		

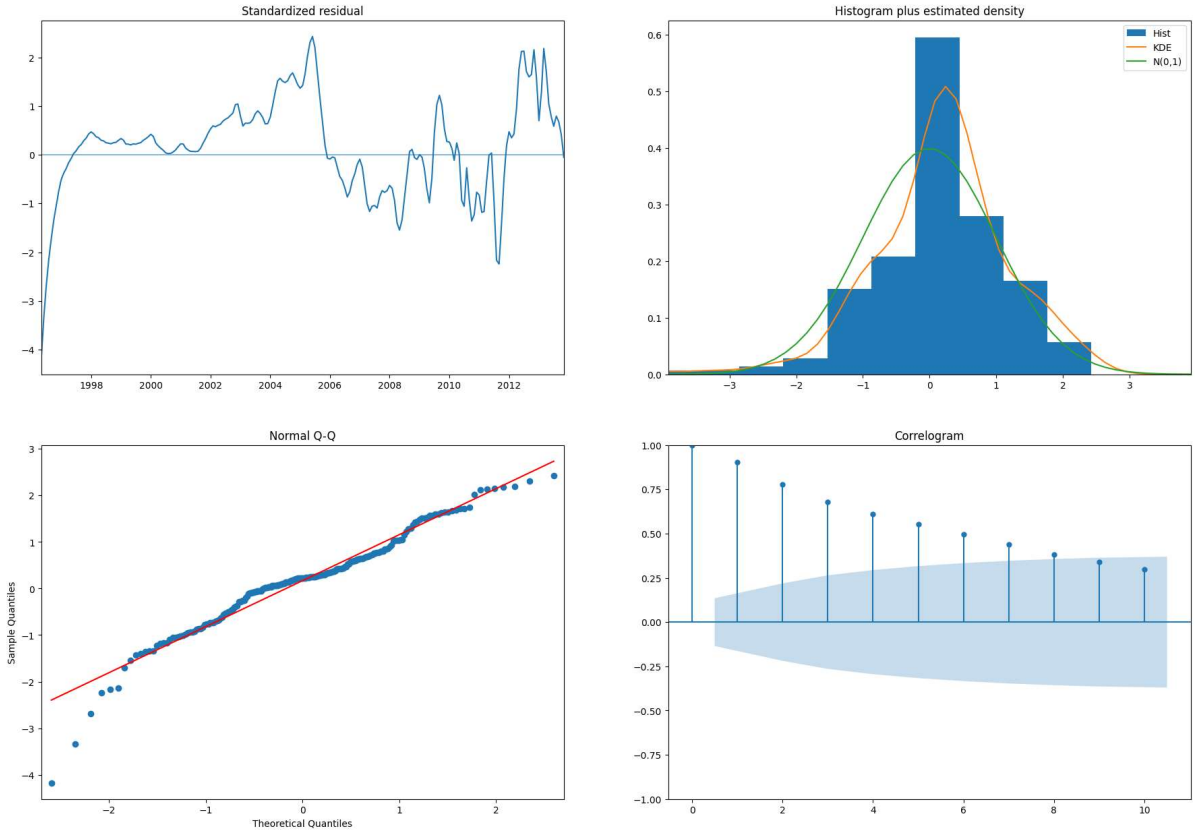
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9066	0.021	42.848	0.000	0.865	0.948
ma.L1	-0.8234	0.027	-30.562	0.000	-0.876	-0.771
sigma2	7.76e+07	3.92e-11	1.98e+18	0.000	7.76e+07	7.76e+07

Ljung-Box (L1) (Q):	174.65	**Jarque-Bera (JB):**	45.31
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	1.56	**Skew:**	-0.61
Prob(H) (two-sided):	0.06	**Kurtosis:**	4.91

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 8.77e+33. Standard errors may be unstable.

Based on the results of the AutoArima model search, the selected model with parameters (1, 1, 1) and a reduced AIC value of 4439.355 appears to be a good fit for the data. The coefficient estimates for the AR(1) and MA(1) terms are statistically significant, indicating that both autoregressive and moving average components are important in capturing the underlying patterns in the data

In [596]:
```
#visulaize model
r5_autoarima.plot_diagnostics(figsize=(22,15))
plt.show()
```

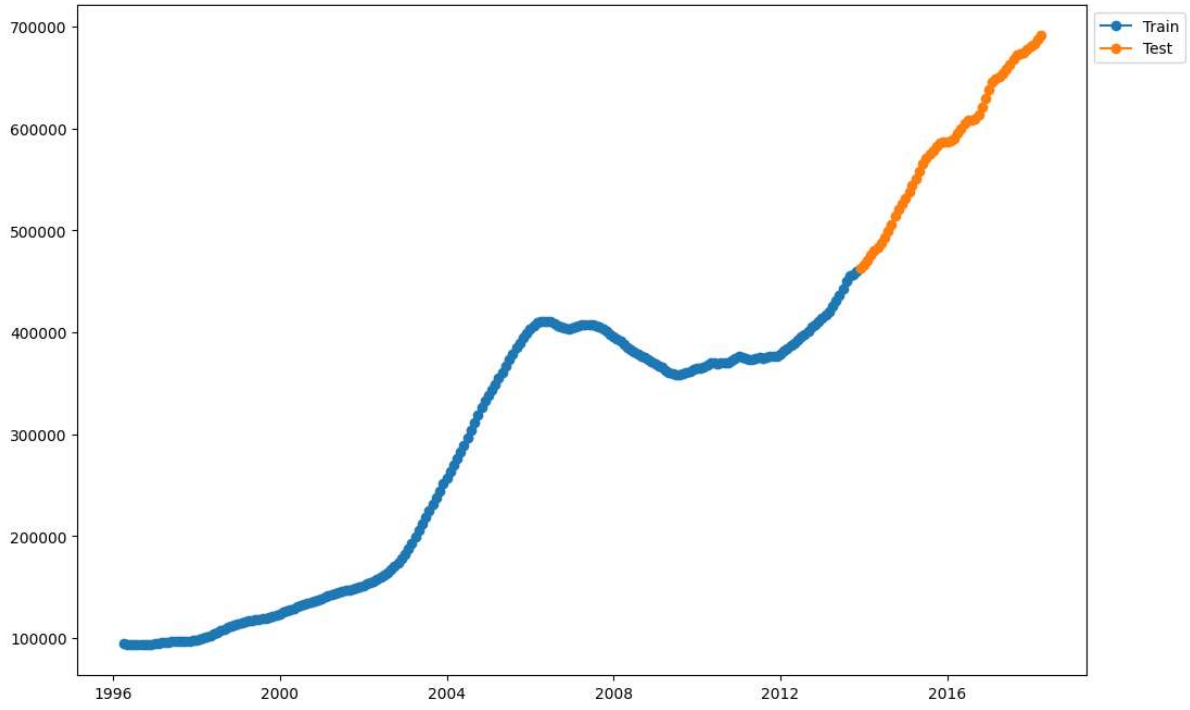


The kernel density estimate (KDE) of the residuals shows a distribution that resembles a normal distribution. This suggests that the residuals, which represent the differences between the observed values and the model's predictions, do not deviate significantly from a symmetric pattern.

Furthermore, when examining the residuals, there is evidence to suggest that the assumption of homoscedasticity is met. Homoscedasticity refers to the assumption that the variability of the residuals is constant across all levels of the predictor variables. In this case, the residuals exhibit a consistent spread or dispersion across the range of the predictor variables, indicating that the assumption of homogeneity of variances is satisfied.

Overall, these findings indicate that the model's residuals display characteristics of normality and homoscedasticity, suggesting that the model adequately captures the variability in the data.

In [597]:
```
# Train Test Split Index
train_size = 0.8
split_idx = round(len(melted_r1)* train_size)
split_idx

# Split
r1_train = melted_r1.iloc[:split_idx]
r1_test = melted_r1.iloc[split_idx:]

# Visualize split
fig,ax= plt.subplots(figsize=(12,8))
kws = dict(marker='o')
plt.plot(r1_train, label='Train', **kws)
plt.plot(r1_test, label='Test', **kws)
ax.legend(bbox_to_anchor=[1,1]);
```



In [598]:
```
#stepwise search of best parameters according to AIC
r1_autoarima = auto_arima(r1_train, start_p=0, start_q=0, d=None,
                          information_criterion='aic', max_d=3, max_q=3,
                          max_p=3, start_P=0, start_Q=0, D=None, max_P=3, max_
                          max_D=3, trace=True, stepwise=True

                         )
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=3922.435, Time=0.10 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=3907.747, Time=0.28 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=3900.061, Time=0.07 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=3997.673, Time=0.02 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=3834.332, Time=0.32 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=3838.212, Time=0.47 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=3838.254, Time=0.65 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=3889.031, Time=0.13 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=3877.504, Time=0.41 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=4423.857, Time=0.62 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=3847.653, Time=0.13 sec

Best model:  ARIMA(1,1,1)(0,0,0)[0] intercept
Total fit time: 3.217 seconds
```

In [599]:
```
r1_autoarima.summary()
```

Out[599]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	212
Model:	SARIMAX(1, 1, 1)	**Log Likelihood**	-1913.166
Date:	Thu, 22 Jun 2023	**AIC**	3834.332
Time:	09:05:09	**BIC**	3847.739
Sample:	04-01-1996	**HQIC**	3839.751
	- 11-01-2013		
Covariance Type:	opg		

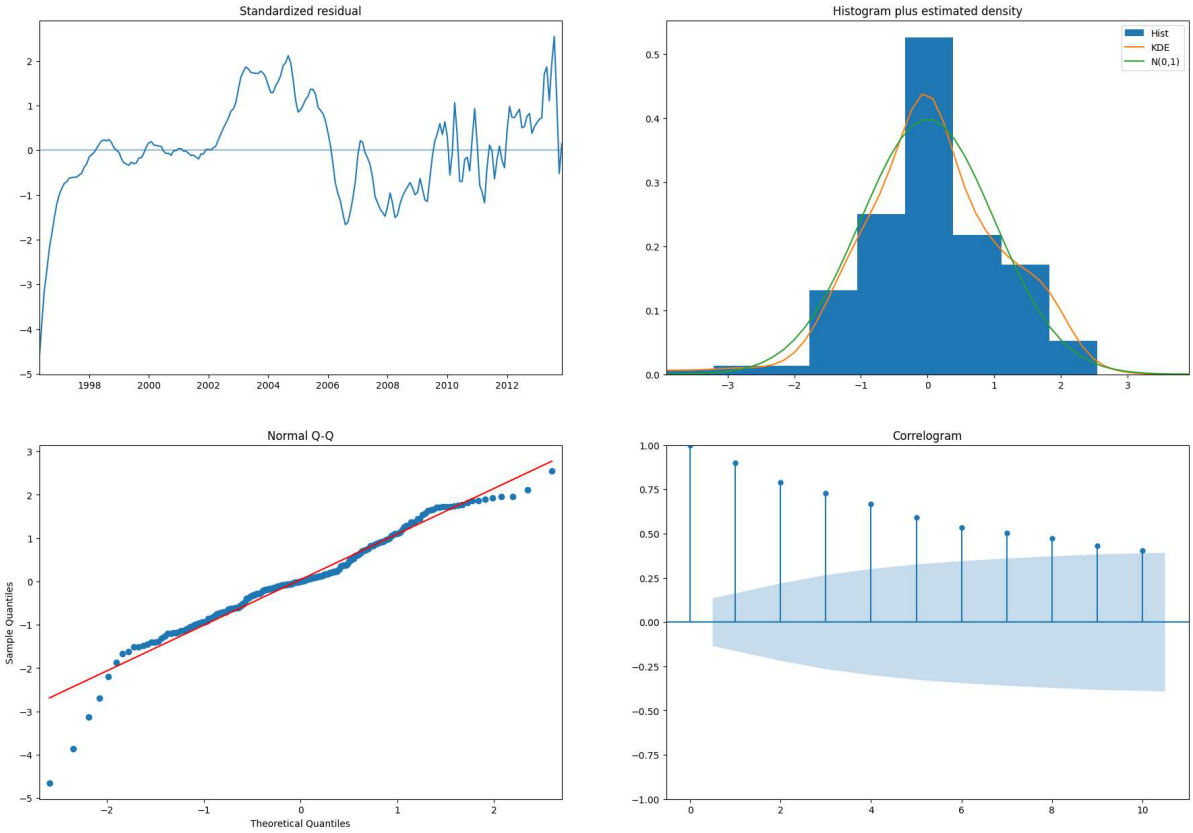
	coef	std err	z	P>|z|	[0.025	0.975]
intercept	116.1963	51.175	2.271	0.023	15.895	216.497
ar.L1	0.9082	0.021	42.880	0.000	0.867	0.950
ma.L1	-0.8313	0.025	-33.123	0.000	-0.880	-0.782
sigma2	3.936e+06	0.001	4.61e+09	0.000	3.94e+06	3.94e+06

Ljung-Box (L1) (Q):	173.39	**Jarque-Bera (JB):**	45.58
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.74	**Skew:**	-0.57
Prob(H) (two-sided):	0.21	**Kurtosis:**	4.97

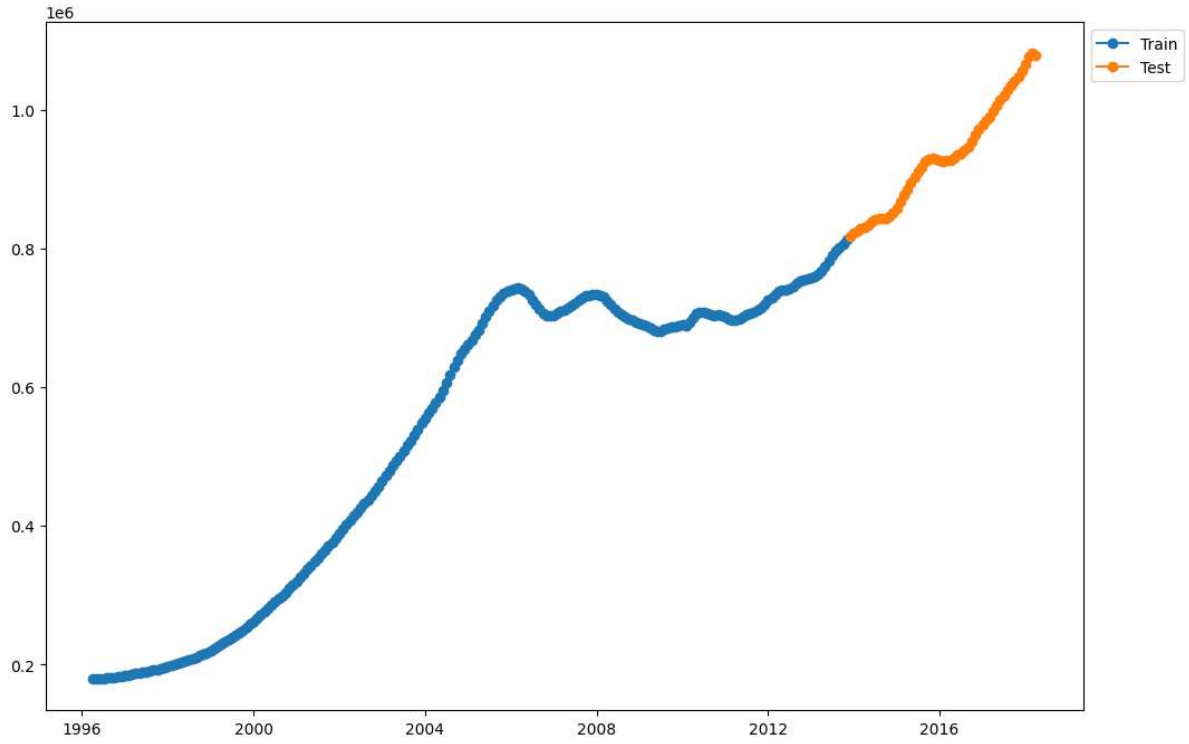
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 4.15e+25. Standard errors may be unstable.

Based on the results of the AutoArima model search, the selected model with parameters (1, 1, 1) and a reduced AIC value of 3836.345 appears to be a good fit for the data. The coefficient estimates for the AR(1) and MA(1) terms are statistically significant, indicating that both autoregressive and moving average components are important in capturing the underlying patterns in the data

In [600]:
```
r1_autoarima.plot_diagnostics(figsize=(22,15))
plt.show()
```



In [601]:
```
# Train Test Split Index
train_size = 0.8
split_idx = round(len(melted_r2)* train_size)
split_idx

# Split
r2_train = melted_r2.iloc[:split_idx]
r2_test = melted_r2.iloc[split_idx:]

# Visualize split
fig,ax= plt.subplots(figsize=(12,8))
kws = dict(marker='o')
plt.plot(r2_train, label='Train', **kws)
plt.plot(r2_test, label='Test', **kws)
ax.legend(bbox_to_anchor=[1,1]);
```



In [602]:
```
#stepwise search of best parameters according to AIC
r2_autoarima = auto_arima(r2_train, start_p=0, start_q=0, d=None,
                          information_criterion='aic', max_d=3, max_q=3,
                          max_p=3, start_P=0, start_Q=0, D=None, max_P=3, max_
                          max_D=3, trace=True, stepwise=True

                         )
```

```
Performing stepwise search to minimize aic
 ARIMA(0,2,0)(0,0,0)[0]             : AIC=3527.918, Time=0.08 sec
 ARIMA(1,2,0)(0,0,0)[0]             : AIC=3527.700, Time=0.16 sec
 ARIMA(0,2,1)(0,0,0)[0]             : AIC=3551.885, Time=0.04 sec
 ARIMA(2,2,0)(0,0,0)[0]             : AIC=3529.351, Time=0.11 sec
 ARIMA(1,2,1)(0,0,0)[0]             : AIC=3553.756, Time=0.13 sec
 ARIMA(2,2,1)(0,0,0)[0]             : AIC=inf, Time=0.40 sec
 ARIMA(1,2,0)(0,0,0)[0] intercept   : AIC=3529.492, Time=0.17 sec

Best model:  ARIMA(1,2,0)(0,0,0)[0]
Total fit time: 1.112 seconds
```

In [603]:
```
#printing results
r2_autoarima.summary()
```

Out[603]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	212
Model:	SARIMAX(1, 2, 0)	**Log Likelihood**	-1761.850
Date:	Thu, 22 Jun 2023	**AIC**	3527.700
Time:	09:05:12	**BIC**	3534.395
Sample:	04-01-1996	**HQIC**	3530.407
	- 11-01-2013		
Covariance Type:	opg		

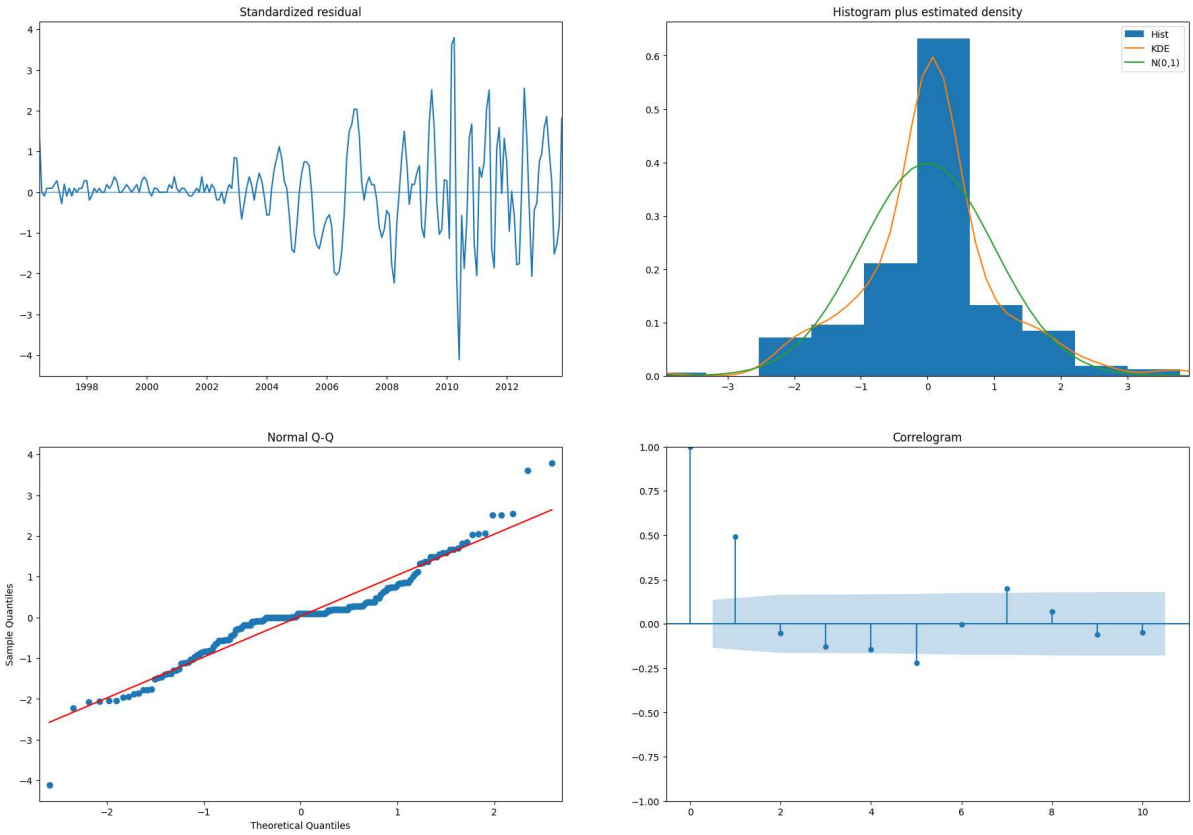
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.0210	0.009	2.215	0.027	0.002	0.040
sigma2	1.123e+06	7.37e+04	15.243	0.000	9.79e+05	1.27e+06

Ljung-Box (L1) (Q):	51.23	**Jarque-Bera (JB):**	52.51
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	39.36	**Skew:**	0.11
Prob(H) (two-sided):	0.00	**Kurtosis:**	5.44

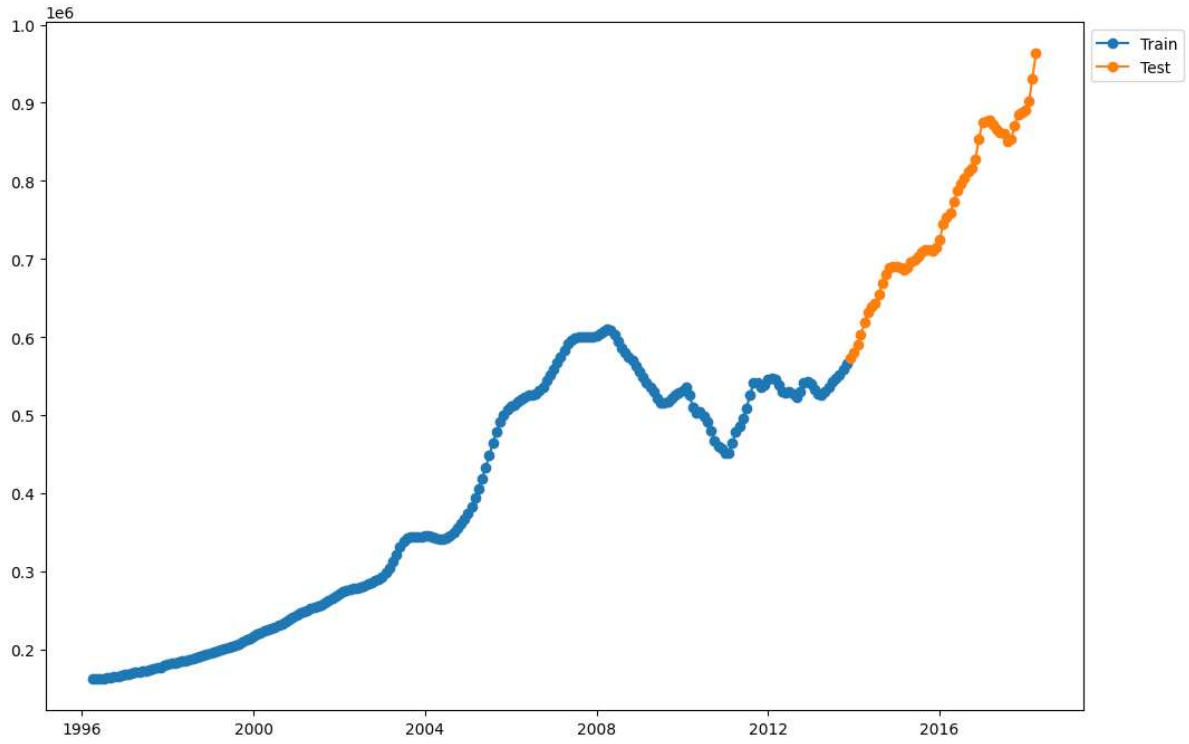
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Based on the results of the AutoArima model search, the selected model with parameters (1, 2, 0) and a reduced AIC value of 3527.700 appears to be a good fit for the data. The coefficient estimates for the AR(1) and MA(1) terms are statistically significant, indicating that both autoregressive and moving average components are important in capturing the underlying patterns in the data

In [604]:
```
r2_autoarima.plot_diagnostics(figsize=(22,15))
plt.show()
```



In [605]:
```
# Train Test Split Index
train_size = 0.8
split_idx = round(len(melted_r3)* train_size)
split_idx

# Split
r3_train = melted_r3.iloc[:split_idx]
r3_test = melted_r3.iloc[split_idx:]

# Visualize split
fig,ax= plt.subplots(figsize=(12,8))
kws = dict(marker='o')
plt.plot(r3_train, label='Train', **kws)
plt.plot(r3_test, label='Test', **kws)
ax.legend(bbox_to_anchor=[1,1]);
```



In [606]:
```
#stepwise search of best parameters according to AIC
r3_autoarima = auto_arima(r3_train, start_p=0, start_q=0, d=None,
                          information_criterion='aic', max_d=3, max_q=3,
                          max_p=3, start_P=0, start_Q=0, D=None, max_P=3, max_
                          max_D=3, trace=True, stepwise=True

                         )
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=4219.759, Time=0.08 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=4193.583, Time=0.69 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=4212.011, Time=0.14 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=4243.886, Time=0.05 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=4192.954, Time=0.86 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=4192.897, Time=0.75 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=4204.152, Time=1.72 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=4187.003, Time=1.26 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=4185.081, Time=0.89 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=4215.633, Time=0.76 sec
 ARIMA(0,1,2)(0,0,0)[0] intercept   : AIC=4220.819, Time=0.14 sec
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.61 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=4193.678, Time=0.38 sec

Best model:  ARIMA(1,1,1)(0,0,0)[0] intercept
Total fit time: 8.356 seconds
```

In [607]:
```
r3_autoarima.summary()
```

Out[607]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	212
Model:	SARIMAX(1, 1, 1)	**Log Likelihood**	-2088.541
Date:	Thu, 22 Jun 2023	**AIC**	4185.081
Time:	09:05:24	**BIC**	4198.489
Sample:	04-01-1996	**HQIC**	4190.501
	- 11-01-2013		
Covariance Type:	opg		

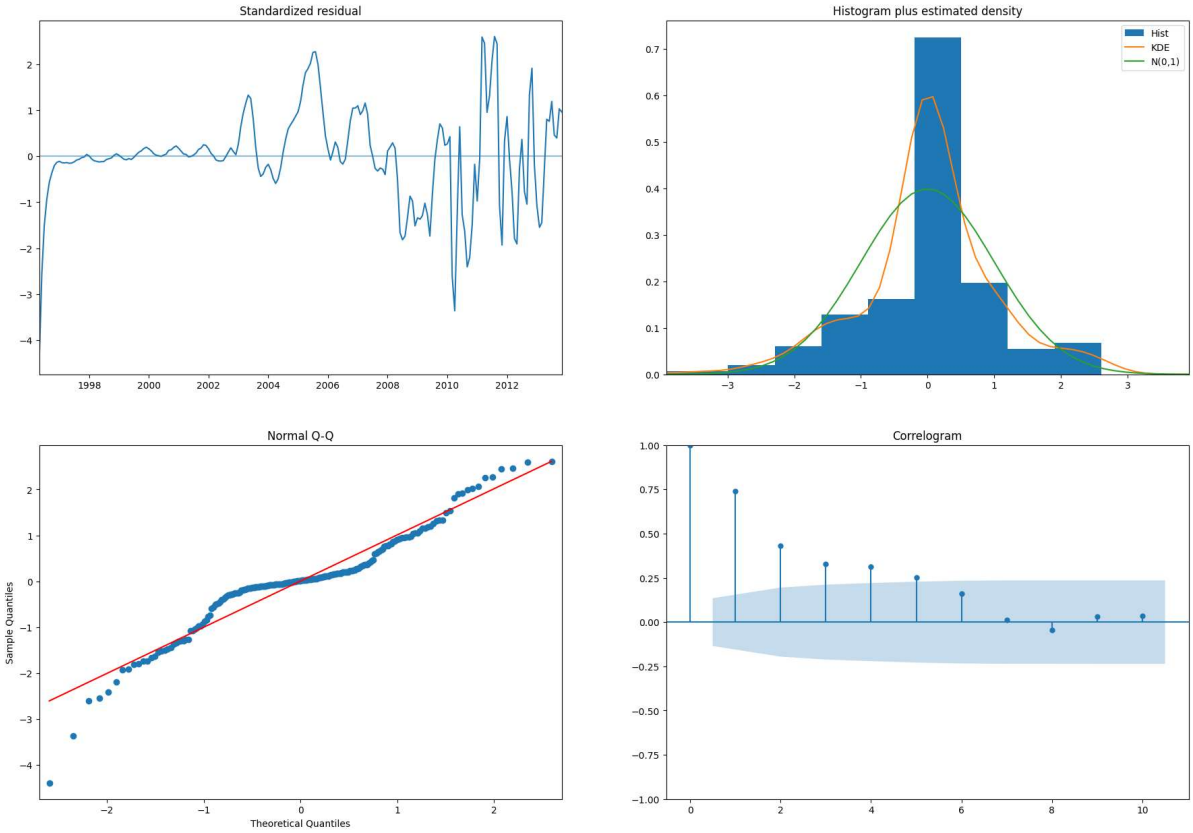
	coef	std err	z	P>|z|	[0.025	0.975]
intercept	518.5893	197.530	2.625	0.009	131.438	905.740
ar.L1	0.6785	0.065	10.383	0.000	0.550	0.807
ma.L1	-0.5633	0.070	-8.060	0.000	-0.700	-0.426
sigma2	2.293e+07	0.002	1.07e+10	0.000	2.29e+07	2.29e+07

Ljung-Box (L1) (Q):	116.71	**Jarque-Bera (JB):**	52.82
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	4.31	**Skew:**	-0.47
Prob(H) (two-sided):	0.00	**Kurtosis:**	5.27

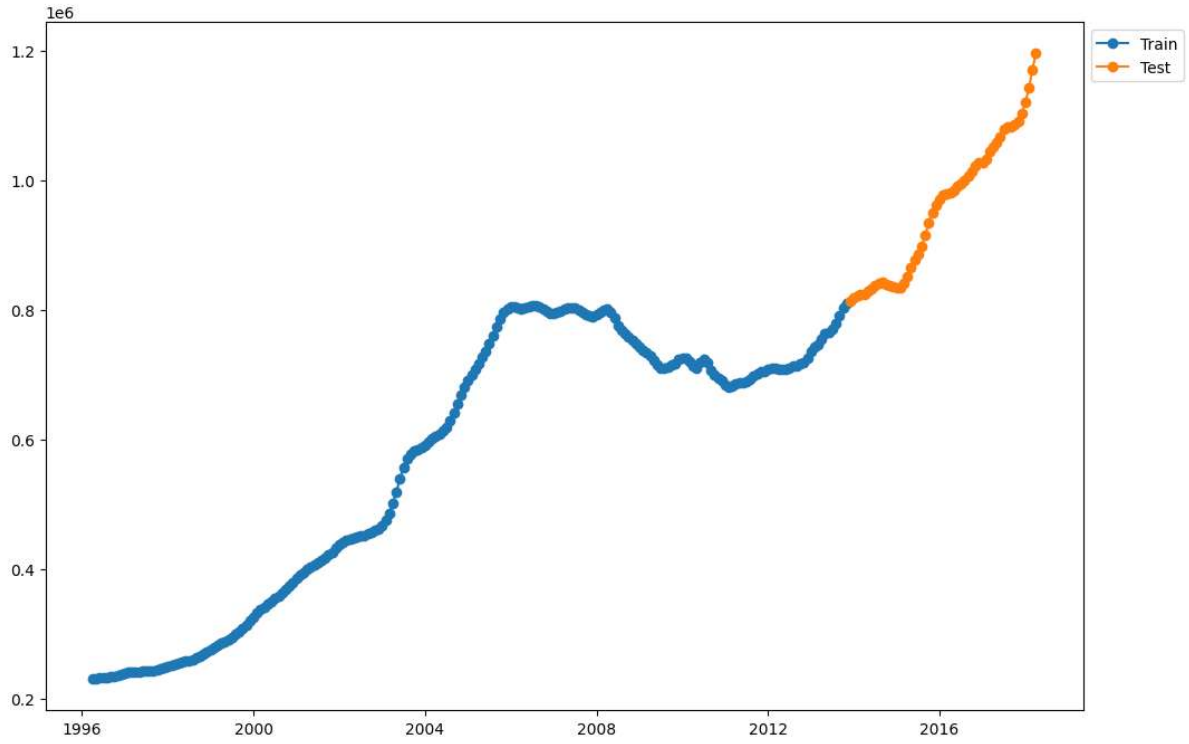
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 5.46e+25. Standard errors may be unstable.

Based on the results of the AutoArima model search, the selected model with parameters (1, 1, 1) and a reduced AIC value of 4185.081 appears to be a good fit for the data. The coefficient estimates for the AR(1) and MA(1) terms are statistically significant, indicating that both autoregressive and moving average components are important in capturing the underlying patterns in the data

In [608]:
```
r3_autoarima.plot_diagnostics(figsize=(22,15))
plt.show()
```



In [609]:
```
# Train Test Split Index
train_size = 0.8
split_idx = round(len(melted_r4)* train_size)
split_idx

# Split
r4_train = melted_r4.iloc[:split_idx]
r4_test = melted_r4.iloc[split_idx:]

# Visualize split
fig,ax= plt.subplots(figsize=(12,8))
kws = dict(marker='o')
plt.plot(r4_train, label='Train', **kws)
plt.plot(r4_test, label='Test', **kws)
ax.legend(bbox_to_anchor=[1,1]);
```



In [610]:
```
#stepwise search of best parameters according to AIC
r4_autoarima = auto_arima(r4_train, start_p=0, start_q=0, d=None,
                          information_criterion='aic', max_d=3, max_q=3,
                          max_p=3, start_P=0, start_Q=0, D=None, max_P=3, max_
                          max_D=3, trace=True, stepwise=True

                         )
```

```
Performing stepwise search to minimize aic
 ARIMA(0,2,0)(0,0,0)[0]             : AIC=3840.044, Time=0.08 sec
 ARIMA(1,2,0)(0,0,0)[0]             : AIC=3840.678, Time=0.28 sec
 ARIMA(0,2,1)(0,0,0)[0]             : AIC=3846.950, Time=0.10 sec
 ARIMA(1,2,1)(0,0,0)[0]             : AIC=3849.237, Time=0.38 sec
 ARIMA(0,2,0)(0,0,0)[0] intercept   : AIC=3842.000, Time=0.09 sec

Best model:  ARIMA(0,2,0)(0,0,0)[0]
Total fit time: 0.975 seconds
```

In [611]:
```
r4_autoarima.summary()
```

Out[611]:

SARIMAX Results

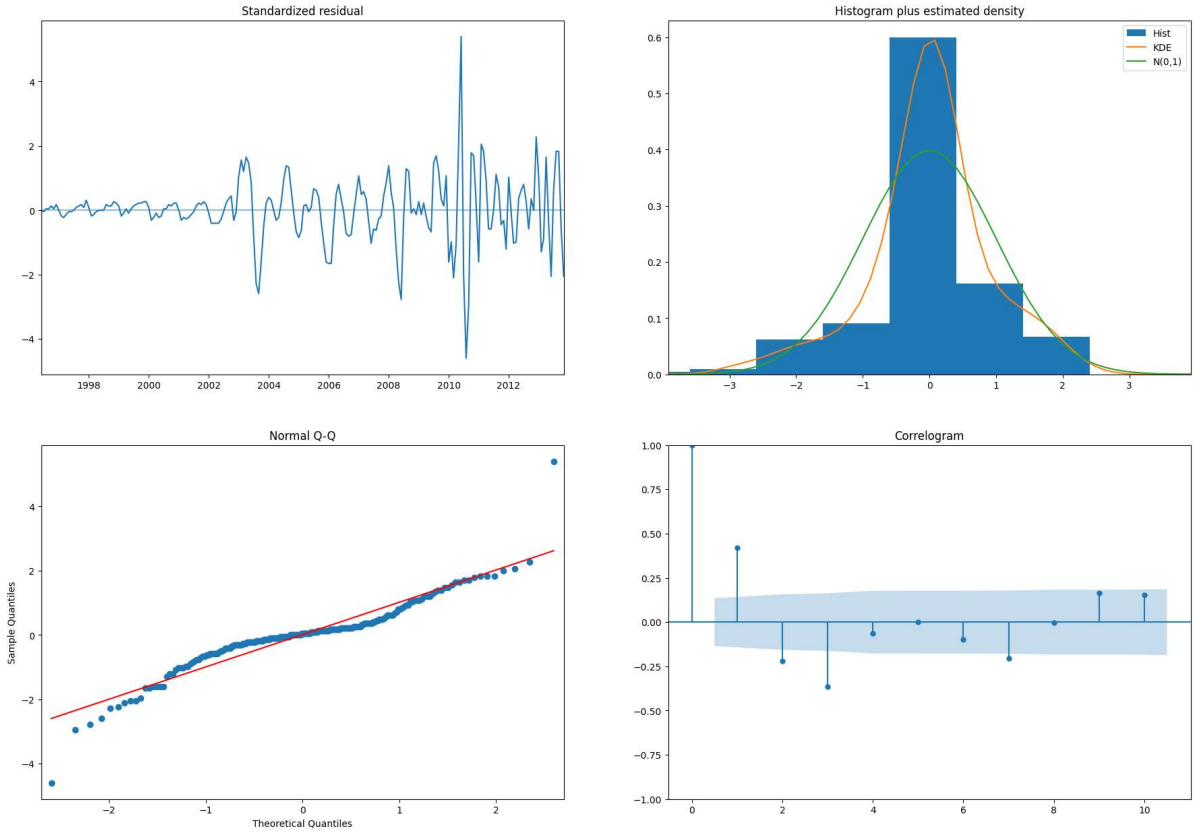
Dep. Variable:	y	**No. Observations:**	212
Model:	SARIMAX(0, 2, 0)	**Log Likelihood**	-1919.022
Date:	Thu, 22 Jun 2023	**AIC**	3840.044
Time:	09:05:28	**BIC**	3843.391
Sample:	04-01-1996	**HQIC**	3841.397
	- 11-01-2013		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
sigma2	5.02e+06	2.49e+05	20.166	0.000	4.53e+06	5.51e+06

Ljung-Box (L1) (Q):	37.38	**Jarque-Bera (JB):**	274.70
Prob(Q):	0.00	**Prob(JB):**	0.00
Heteroskedasticity (H):	72.19	**Skew:**	-0.01
Prob(H) (two-sided):	0.00	**Kurtosis:**	8.60

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [612]:
```
r4_autoarima.plot_diagnostics(figsize=(22,15))
plt.show()
```



For the Q-Q plot, the data points are in the form of a steep slope. This means that the clustering of the data around the mean and standard deviation is tight. If the slopping is shallow then the data is more dispersed and may not be well-described by a normal distribution. This may indicate the presence of outliers in the data points. This is shown by the deviation from the line of best fit. For the measure of assymetry, the points are more dispersed on one side of the plot than the other.

KDE's can be a good way to visualize the distribution of the data and to identify clusters and outliers, and it can show a normal distribution between data points.In general, a smooth KDE or evenly heighted bars indicate uniformity. A choppy KDE or unevenly heighted bars indicate non-uniformity. From the overview of r1 to r5, it can be seen that the data gets more uniform with each zipcode from 1 to 5 as observed from the visualizations.

FORECASTING

After training each model, it is now time to assess their performance by making predictions on the test set. This step will help validate the accuracy and reliability of the models' results.

By applying the trained models to the test set, we can evaluate how well they generalize to unseen data. This process involves using the learned patterns and relationships from the training data to make predictions on the test data points. By comparing the predicted values with

the actual values in the test set, we can measure the models' performance and determine their effectiveness in capturing the underlying patterns and making accurate predictions.

Through this validation process, we can gain confidence in the reliability of the models and assess their suitability for forecasting or further analysis. It allows us to evaluate the models' ability to generalize beyond the training data and provide insights into their overall performance

NewYork_11230 Forecasting

In [613]:
```
#predict over test period
r5_preds = r5_autoarima.predict(n_periods=r5_test.shape[0], return_conf_int=Tr
r5_preds
```

Out[613]:
```
(2013-12-01    1.266552e+06
 2014-01-01    1.272311e+06
 2014-02-01    1.277531e+06
 2014-03-01    1.282265e+06
 2014-04-01    1.286556e+06
 2014-05-01    1.290446e+06
 2014-06-01    1.293973e+06
 2014-07-01    1.297171e+06
 2014-08-01    1.300070e+06
 2014-09-01    1.302698e+06
 2014-10-01    1.305081e+06
 2014-11-01    1.307241e+06
 2014-12-01    1.309200e+06
 2015-01-01    1.310976e+06
 2015-02-01    1.312585e+06
 2015-03-01    1.314045e+06
 2015-04-01    1.315368e+06
 2015-05-01    1.316567e+06
 2015-06-01    1.317655e+06
```

In [614]:
```
#put forecasts in DF
r5_forecastdf = pd.DataFrame(r5_preds[0], index=r5_test.index, columns=['predi
```

In [615]:
```
#checking shapeof forecast r5
r5_forecastdf.shape[0]
```

Out[615]:
```
53
```

In [616]:
```
r5_forecastdf
```

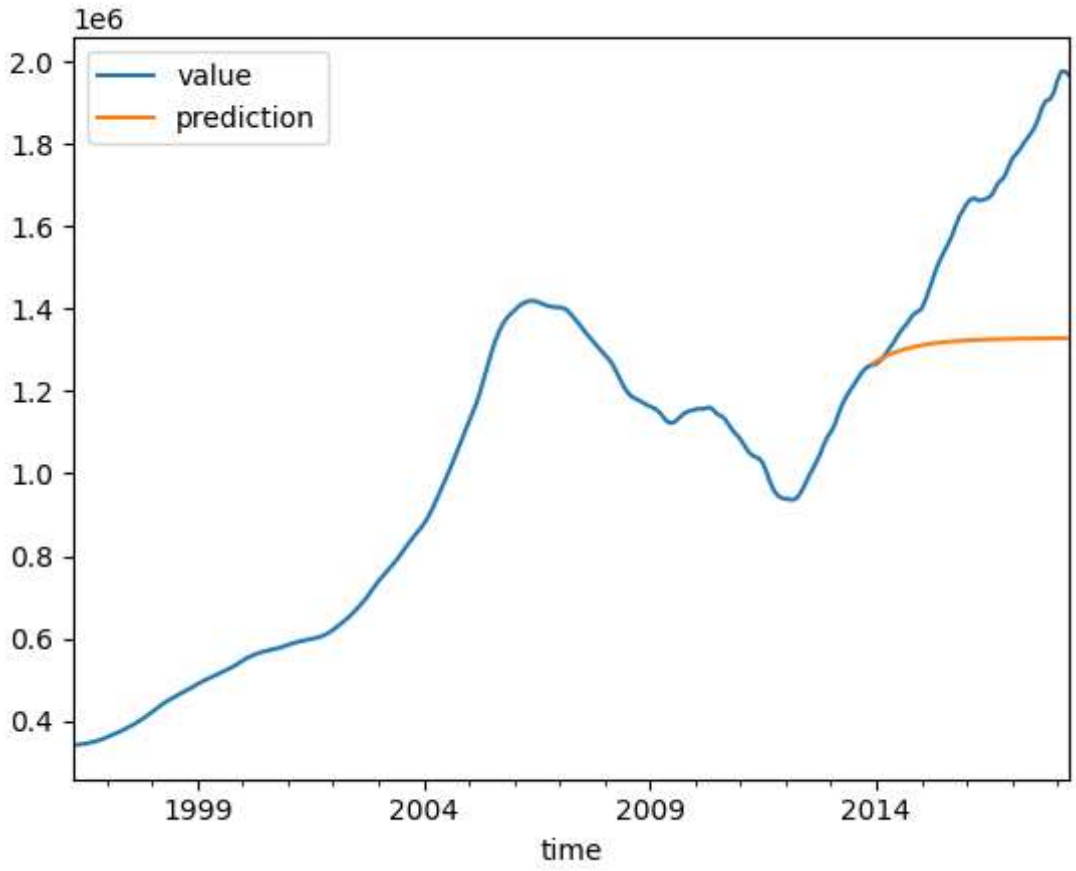
Out[616]:

time	prediction
2013-12-01	1.266552e+06
2014-01-01	1.272311e+06
2014-02-01	1.277531e+06
2014-03-01	1.282265e+06
2014-04-01	1.286556e+06
2014-05-01	1.290446e+06
2014-06-01	1.293973e+06
2014-07-01	1.297171e+06
2014-08-01	1.300070e+06
2014-09-01	1.302698e+06
2014-10-01	1.305081e+06
2014-11-01	1.307241e+06
2014-12-01	1.309200e+06
2015-01-01	1.310976e+06
2015-02-01	1.312585e+06
2015-03-01	1.314045e+06
2015-04-01	1.315368e+06
2015-05-01	1.316567e+06
2015-06-01	1.317655e+06
2015-07-01	1.318641e+06
2015-08-01	1.319535e+06
2015-09-01	1.320345e+06
2015-10-01	1.321080e+06
2015-11-01	1.321746e+06
2015-12-01	1.322350e+06
2016-01-01	1.322897e+06
2016-02-01	1.323394e+06
2016-03-01	1.323844e+06
2016-04-01	1.324252e+06
2016-05-01	1.324621e+06
2016-06-01	1.324957e+06
2016-07-01	1.325261e+06
2016-08-01	1.325536e+06
2016-09-01	1.325786e+06
2016-10-01	1.326013e+06

time	prediction
2016-11-01	1.326218e+06
2016-12-01	1.326404e+06
2017-01-01	1.326573e+06
2017-02-01	1.326726e+06
2017-03-01	1.326865e+06
2017-04-01	1.326991e+06
2017-05-01	1.327105e+06
2017-06-01	1.327208e+06
2017-07-01	1.327302e+06
2017-08-01	1.327387e+06
2017-09-01	1.327464e+06
2017-10-01	1.327534e+06
2017-11-01	1.327597e+06
2017-12-01	1.327655e+06
2018-01-01	1.327707e+06
2018-02-01	1.327754e+06
2018-03-01	1.327797e+06
2018-04-01	1.327835e+06

In [617]:
```
#visualize prediction
pd.concat([melted_r5['value'], r5_forecastdf], axis=1).plot()
```

Out[617]:
```
<AxesSubplot: xlabel='time'>
```



The model's good performance on the test set allows for its deployment in predicting outcomes over a 3-year investment horizon. By leveraging learned patterns, the model provides forecasts that aid decision-making. However, monitoring and reassessment of predictions are advised due to the inherent uncertainty associated with long-term forecasting.

In [618]:
```
periods = r5_forecastdf.shape[0] + 36
periods
```

Out[618]:
```
89
```

In [619]:
```
r5_future_forecast = r5_autoarima.predict(n_periods=periods,
                                          return_conf_int=True)
#test data + 3 year forecast
```

In [620]:
```
#forecast into investment horizon of 3 years
r5_forecast_range = pd.date_range(start='2014-02-01',
                                  periods=periods,
                                 freq='MS')
```

In [621]:
```
r5_forecast_range
```

Out[621]:
```
DatetimeIndex(['2014-02-01', '2014-03-01', '2014-04-01', '2014-05-01',
               '2014-06-01', '2014-07-01', '2014-08-01', '2014-09-01',
               '2014-10-01', '2014-11-01', '2014-12-01', '2015-01-01',
               '2015-02-01', '2015-03-01', '2015-04-01', '2015-05-01',
               '2015-06-01', '2015-07-01', '2015-08-01', '2015-09-01',
               '2015-10-01', '2015-11-01', '2015-12-01', '2016-01-01',
               '2016-02-01', '2016-03-01', '2016-04-01', '2016-05-01',
               '2016-06-01', '2016-07-01', '2016-08-01', '2016-09-01',
               '2016-10-01', '2016-11-01', '2016-12-01', '2017-01-01',
               '2017-02-01', '2017-03-01', '2017-04-01', '2017-05-01',
               '2017-06-01', '2017-07-01', '2017-08-01', '2017-09-01',
               '2017-10-01', '2017-11-01', '2017-12-01', '2018-01-01',
               '2018-02-01', '2018-03-01', '2018-04-01', '2018-05-01',
               '2018-06-01', '2018-07-01', '2018-08-01', '2018-09-01',
               '2018-10-01', '2018-11-01', '2018-12-01', '2019-01-01',
               '2019-02-01', '2019-03-01', '2019-04-01', '2019-05-01',
               '2019-06-01', '2019-07-01', '2019-08-01', '2019-09-01',
               '2019-10-01', '2019-11-01', '2019-12-01', '2020-01-01',
               '2020-02-01', '2020-03-01', '2020-04-01', '2020-05-01',
               '2020-06-01', '2020-07-01', '2020-08-01', '2020-09-01',
               '2020-10-01', '2020-11-01', '2020-12-01', '2021-01-01',
               '2021-02-01', '2021-03-01', '2021-04-01', '2021-05-01',
               '2021-06-01'],
              dtype='datetime64[ns]', freq='MS')
```

In [622]:
```
r5_future_forecast
```

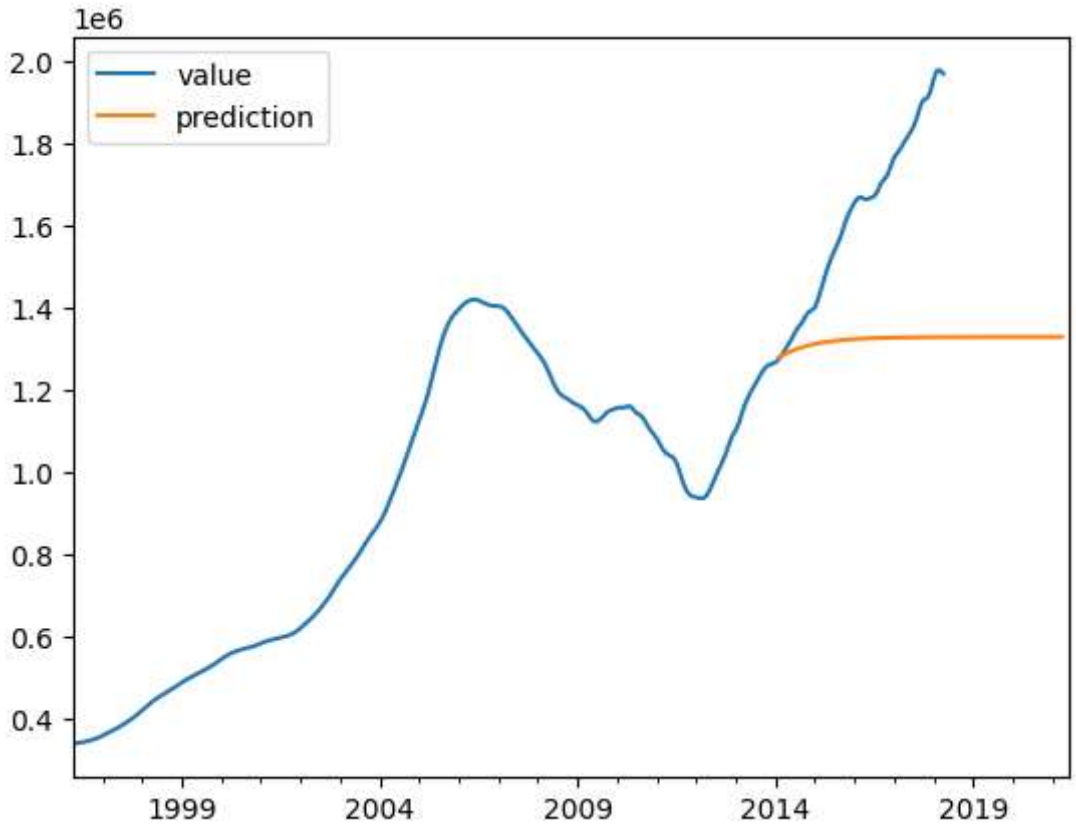
```
Out[622]: (2013-12-01    1.266552e+06
           2014-01-01    1.272311e+06
           2014-02-01    1.277531e+06
           2014-03-01    1.282265e+06
           2014-04-01    1.286556e+06
                            ...
           2020-12-01    1.328195e+06
           2021-01-01    1.328197e+06
           2021-02-01    1.328198e+06
           2021-03-01    1.328200e+06
           2021-04-01    1.328201e+06
           Freq: MS, Length: 89, dtype: float64,
           array([[1249286.84495318, 1283816.96127865],
                  [1246857.84079449, 1297763.31106814],
                  [1245158.27770902, 1309904.56760821],
                  [1243575.62150166, 1320953.72466524],
                  [1241922.06663745, 1331189.6650138 ],
                  [1240130.70187066, 1340761.87106204],
                  [1238180.94434611, 1349765.7851678 ],
                  [1236072.48508105, 1358269.58438037],
                  [1233814.32820228, 1366325.79416217],
                  [1231419.63591346, 1373977.0353979 ],
                  [1228903.1331705 , 1381259.15640851],
                  [1226279.75239968, 1388203.07479863],
                  [1223563.92055068, 1394835.93152697],
                  [1220769.19501695, 1401181.8549109 ],
                  [1217908.0951389 , 1407262.49169764],
                  [1214992.04529102, 1413097.39268917],
                  [1212031.38178234, 1418704.30385409],
                  [1209035.39563049, 1424099.39373323],
                  [1206012.39454025, 1429297.43640764],
                  [1202969.77402248, 1434311.96244955],
                  [1199914.09155784, 1439155.38608865],
                  [1196851.14014145, 1443839.11419423],
                  [1193786.01905704, 1448373.64098109],
                  [1190723.20067881, 1452768.6312327 ],
                  [1187666.5926986 , 1457032.99408774],
                  [1184619.59555302, 1461174.94892371],
                  [1181585.1550605 , 1465202.08451414],
                  [1178565.8104214 , 1469121.4123819 ],
                  [1175563.73781769, 1472939.41508697],
                  [1172580.78989376, 1476662.09005148],
                  [1169618.53141903, 1480294.9894226 ],
                  [1166678.27143668, 1483843.25639567],
                  [1163761.09219582, 1487311.6583591 ],
                  [1160867.87515148, 1490704.61717371],
                  [1157999.32430023, 1494026.23686034],
                  [1155155.98710096, 1497280.32893704],
                  [1152338.2732115 , 1500470.43562016],
                  [1149546.47125292, 1503599.85108109],
                  [1146780.76379534, 1506671.64093045],
                  [1144041.24074188, 1509688.66008495],
                  [1141327.91127116, 1512653.56915697],
                  [1138640.71448402, 1515568.84949395],
                  [1135979.52888599, 1518436.81698304],
                  [1133344.18082471, 1521259.63472593],
                  [1130734.4519896 , 1524039.32467969],
```

```
[1128150.08607082, 1526777.77835073],
[1125590.79466471, 1529476.76662153],
[1123056.26250435, 1532137.94878307],
[1120546.15208605, 1534762.88083916],
[1118060.10775526, 1537353.02314375],
[1115597.75930916, 1539909.74742665],
[1113158.72516735, 1542434.34325854],
[1110742.61515665, 1544928.02400194],
[1108349.03295163, 1547391.93229057],
[1105977.57820792, 1549827.14507633],
[1103627.84842174, 1552234.67827935],
[1101299.44054555, 1554615.49107404],
[1098991.95238672, 1556970.48984096],
[1096704.98381314, 1559300.53181197],
[1094438.13778744, 1561606.42843372],
[1092191.02124915, 1563888.94847264],
[1089963.24586192, 1566148.82088228],
[1087754.42864141, 1568386.73745255],
[1085564.19247767, 1570603.35525834],
[1083392.16656432, 1572799.29892393],
[1081237.98674562, 1574975.16271782],
[1079101.29579131, 1577131.51249189],
[1076981.74360807, 1579268.88747703],
[1074878.98739535, 1581387.80194703],
[1072792.69175273, 1583488.74676097],
[1070722.52874497, 1585572.19079383],
[1068668.17793028, 1587638.58226416],
[1066629.32635693, 1589688.34996693],
[1064605.66853231, 1591721.90441883],
[1062596.90636866, 1593739.63892312],
[1060602.74910871, 1595741.93055999],
[1058622.91323437, 1597729.14110841],
[1056657.12236121, 1599701.61790458],
[1054705.10712107, 1601659.69464194],
[1052766.60503496, 1603603.69211715],
[1050841.36037814, 1605533.91892615],
[1048929.12403887, 1607450.67211395],
[1047029.65337248, 1609354.23778185],
[1045142.71205182, 1611244.89165502],
[1043268.06991536, 1613122.89961348],
[1041405.50281368, 1614988.51818923],
[1039554.79245545, 1616841.99503185],
[1037715.72625331, 1618683.56934504],
[1035888.09717056, 1620513.47229605]]))
```

In [623]:
```
#put forecasts in DF
r5_future_forecastdf = pd.DataFrame(r5_future_forecast[0], index=r5_forecast_r
                                    columns=['prediction'])
```

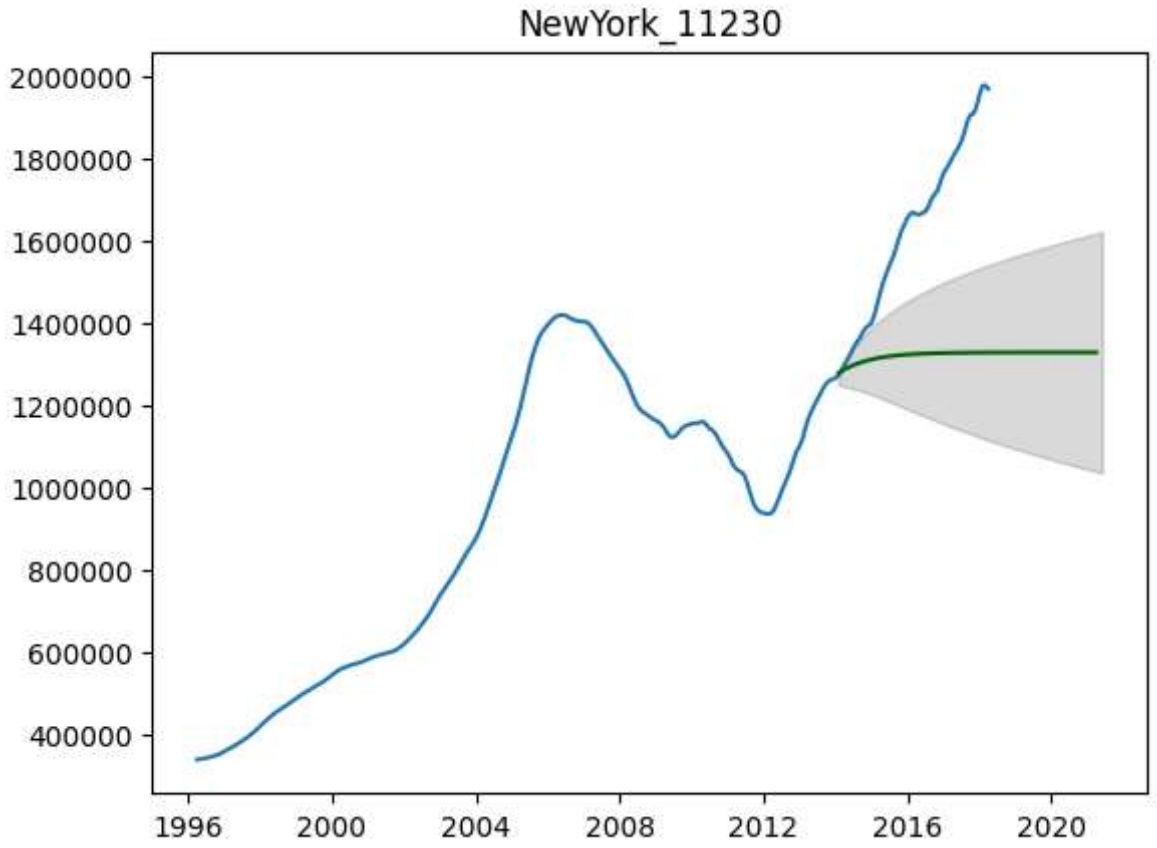
In [624]:
```
#visualize prediction
pd.concat([melted_r5['value'], r5_future_forecastdf], axis=1).plot()
```

Out[624]: <AxesSubplot: >



In [625]:
```
#define confidence intervals
lower_r5 = pd.Series(r5_future_forecast[1][:,0], index=r5_forecast_range)
upper_r5 = pd.Series(r5_future_forecast[1][:,1], index=r5_forecast_range)
```

In [626]:
```
#visualize forecasts with confidence intervals
plt.plot(melted_r5['value'])
plt.plot(r5_future_forecastdf, color='darkgreen')
plt.fill_between(r5_forecast_range,
                 lower_r5,
                 upper_r5,
                 color='k', alpha=.15)
plt.title('NewYork_11230')
plt.ticklabel_format(useOffset=False, axis='y', style='plain')
```



The visualization above showcases the forecasts for New York 11230. The predicted values are displayed along with confidence intervals represented by the gray area. These intervals provide a range within which the actual values are likely to fall, accounting for the uncertainty in the predictions. This visual representation aids in understanding the potential variability of the forecasts and assists in decision-making and analysis.

LosAngeles_90046 Forecasting

In [627]:
```
#predict over the test data
r4_preds = r4_autoarima.predict(n_periods=r4_test.shape[0], return_conf_int=Tr
```

In [628]:
```
r4_preds
```

Out[628]:
```
(2013-12-01     815700.0
 2014-01-01     822400.0
 2014-02-01     829100.0
 2014-03-01     835800.0
 2014-04-01     842500.0
 2014-05-01     849200.0
 2014-06-01     855900.0
 2014-07-01     862600.0
 2014-08-01     869300.0
 2014-09-01     876000.0
 2014-10-01     882700.0
 2014-11-01     889400.0
 2014-12-01     896100.0
 2015-01-01     902800.0
 2015-02-01     909500.0
 2015-03-01     916200.0
 2015-04-01     922900.0
 2015-05-01     929600.0
 2015-06-01     936300.0
```

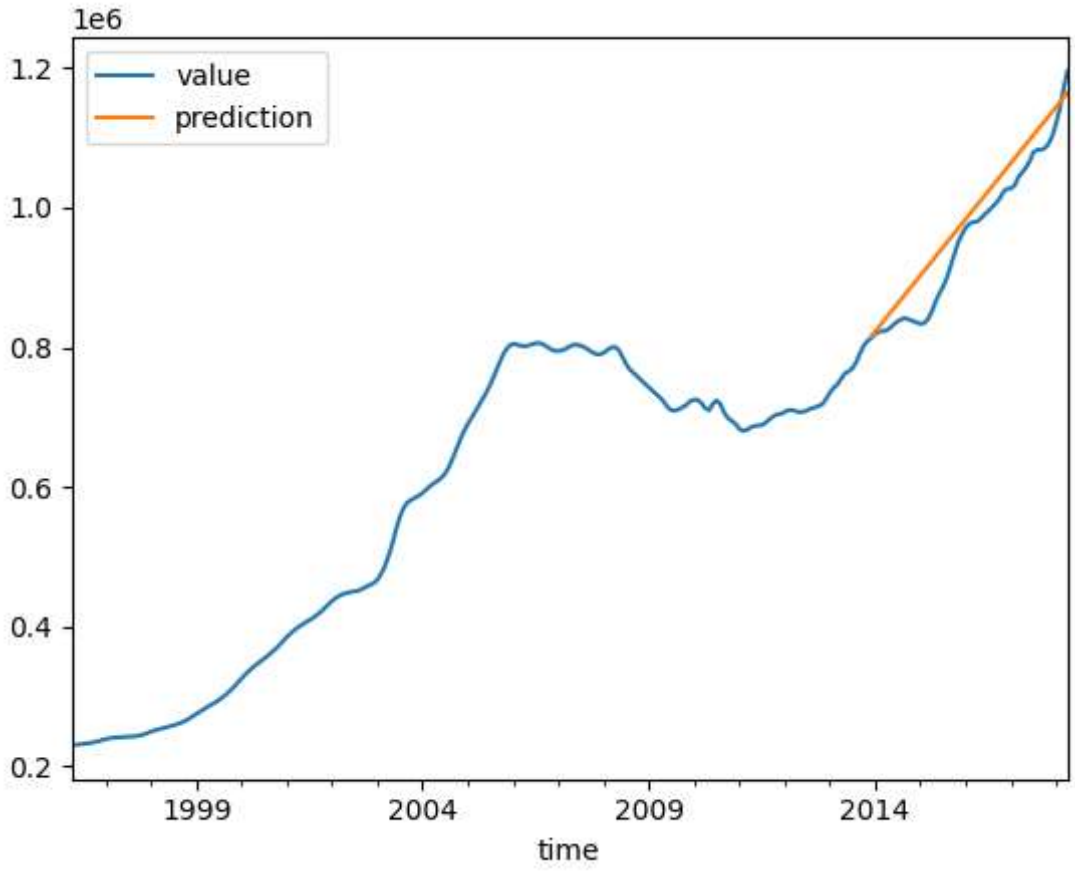
In [629]:
```
#put predictions into DF
r4_forecastdf = pd.DataFrame(r4_preds[0], index=r4_test.index, columns=['predi
r4_forecastdf.head(10)
```

Out[629]:

	prediction
time	
2013-12-01	815700.0
2014-01-01	822400.0
2014-02-01	829100.0
2014-03-01	835800.0
2014-04-01	842500.0
2014-05-01	849200.0
2014-06-01	855900.0
2014-07-01	862600.0
2014-08-01	869300.0
2014-09-01	876000.0

In [630]:
```
#visulaize
pd.concat([melted_r4['value'], r4_forecastdf], axis=1).plot()
```

Out[630]: <AxesSubplot: xlabel='time'>



The predictions made by the model did not accurately match the actual values in the test data. The forecasted outcomes did not align closely with the observed values during the evaluation process.

In [631]:
```
periods = r4_forecastdf.shape[0] + 36
periods
```

Out[631]: 89

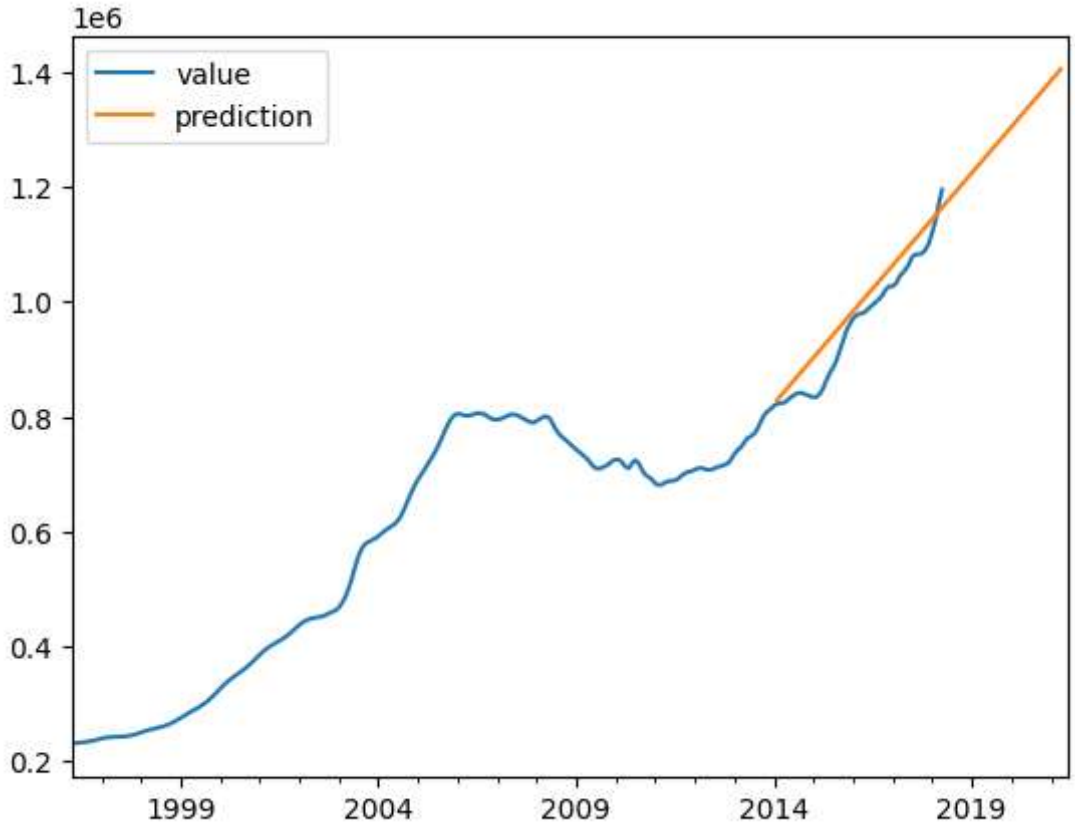
In [632]:
```
r4_future_forecast = r4_autoarima.predict(n_periods=periods,
                                          return_conf_int=True)
```

In [633]:
```
#forecast into investment horizon of 3 years
r4_forecast_range = pd.date_range(start='2014-02-01',
                                  periods=periods,
                                 freq='MS')
```

In [634]:
```
r4_future_forecastdf = pd.DataFrame(r4_future_forecast[0], index=r4_forecast_r
                                    columns=['prediction'])
```

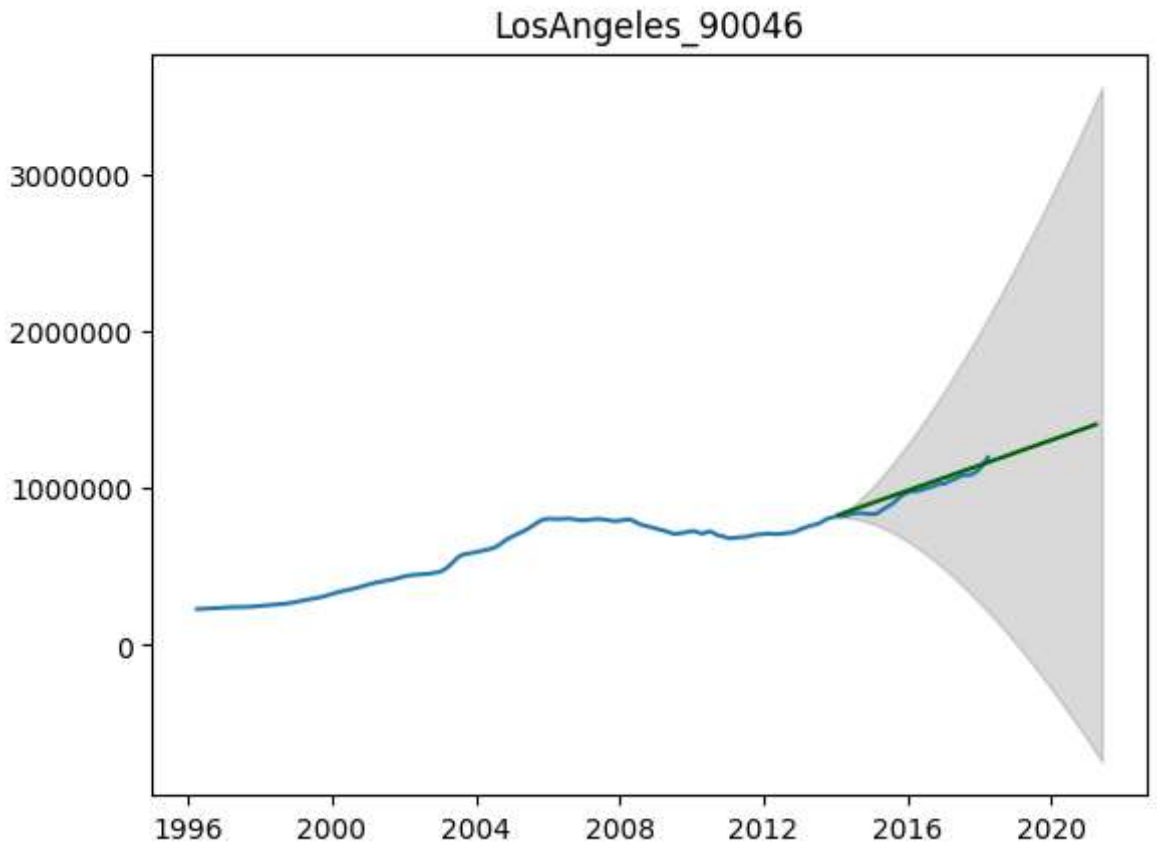
In [635]:
```
pd.concat([melted_r4['value'], r4_future_forecastdf], axis=1).plot()
```

Out[635]: <AxesSubplot: >



In [636]:
```
#define confidence intervals
lower_r4 = pd.Series(r4_future_forecast[1][:,0], index=r4_forecast_range)
upper_r4 = pd.Series(r4_future_forecast[1][:,1], index=r4_forecast_range)
```

In [637]:
```
#visualize forecasts with confidence intervals
plt.plot(melted_r4['value'])
plt.plot(r4_future_forecastdf, color='darkgreen')
plt.fill_between(r4_forecast_range,
                 lower_r4,
                 upper_r4,
                 color='k', alpha=.15)
plt.title('LosAngeles_90046')
plt.ticklabel_format(useOffset=False, axis='y', style='plain')
```



The model's forecast was more cautious or conservative compared to the test data. The predicted outcomes were generally lower or less optimistic than the actual values observed during the evaluation phase.

NewYork_11226 Forecasting

In [638]:
```
#predict over test data
r3_preds = r3_autoarima.predict(n_periods=r3_test.shape[0], return_conf_int=Tr
```

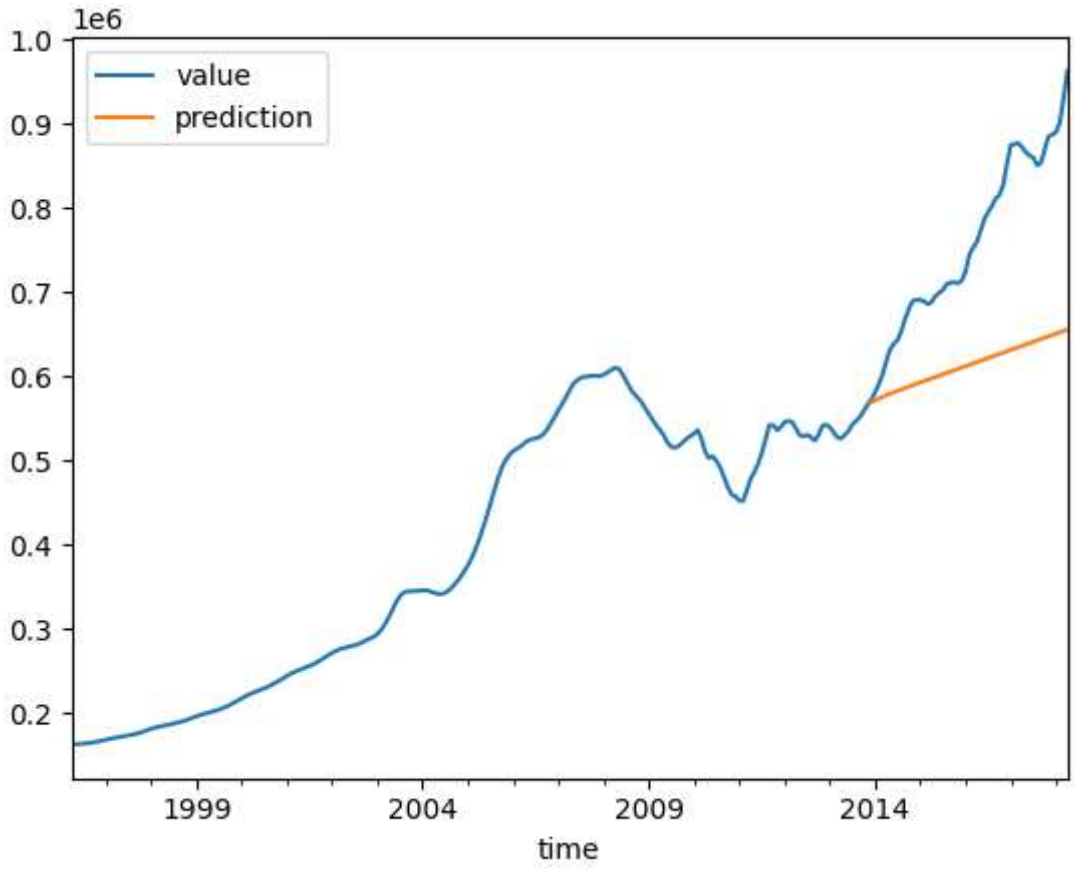
In [639]:
```
r3_forecastdf = pd.DataFrame(r3_preds[0], index=r3_test.index, columns=['predi
r3_forecastdf.head(10)
```

Out[639]:

time	prediction
2013-12-01	569076.298862
2014-01-01	571546.528015
2014-02-01	573741.228843
2014-03-01	575748.976896
2014-04-01	577629.872875
2014-05-01	579424.696589
2014-06-01	581161.118145
2014-07-01	582857.912395
2014-08-01	584527.818538
2014-09-01	586179.480437

In [640]:
```
#visualize
pd.concat([melted_r3['value'], r3_forecastdf], axis=1).plot()
```

Out[640]: <AxesSubplot: xlabel='time'>



In [641]:
```
periods = r3_forecastdf.shape[0] + 36
periods
```

Out[641]: 89

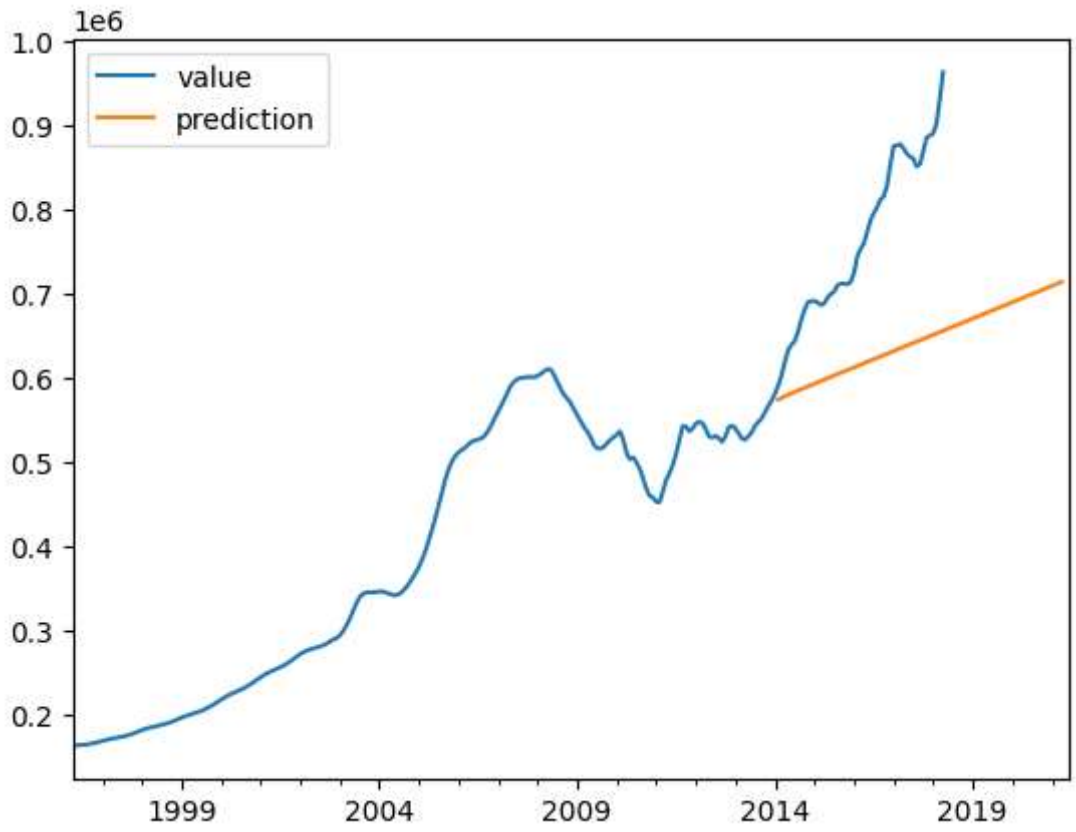
In [642]:
```
r3_future_forecast = r3_autoarima.predict(n_periods=periods,
                                          return_conf_int=True)
```

In [643]:
```
#forecast into investment horizon of 3 years
r3_forecast_range = pd.date_range(start='2014-02-01',
                                  periods=periods,
                                  freq='MS')
```

In [644]:
```
r3_future_forecastdf = pd.DataFrame(r3_future_forecast[0], index=r3_forecast_r
                                    columns=['prediction'])
```

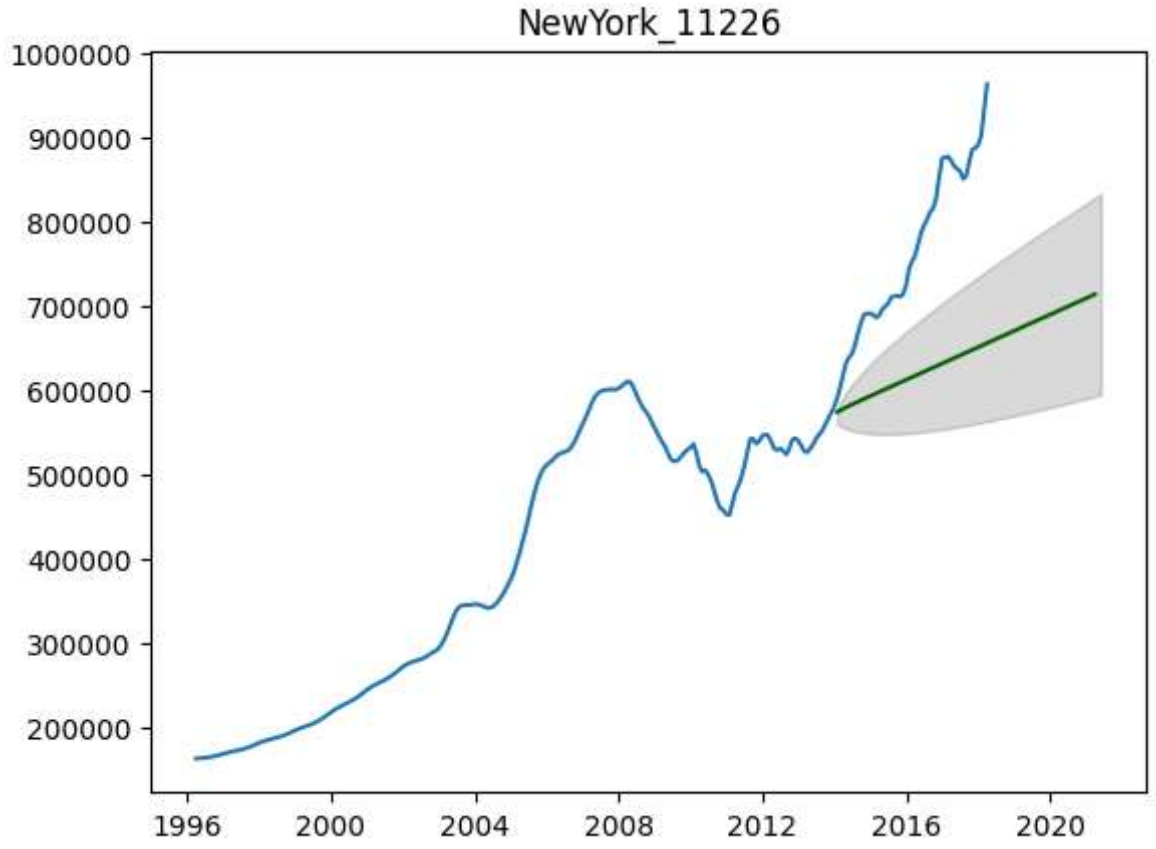
In [645]:
```
r3_combo = pd.concat([melted_r3['value'], r3_future_forecastdf], axis=1)
r3_combo.plot()
```

Out[645]: <AxesSubplot: >



In [646]:
```
lower_r3 = pd.Series(r3_future_forecast[1][:,0], index=r3_forecast_range)
upper_r3 = pd.Series(r3_future_forecast[1][:,1], index=r3_forecast_range)
```

In [647]:
```
plt.plot(melted_r3['value'])
plt.plot(r3_future_forecastdf, color='darkgreen')
plt.fill_between(r3_forecast_range,
                 lower_r3,
                 upper_r3,
                 color='k', alpha=.15)
plt.title('NewYork_11226')
plt.ticklabel_format(useOffset=False, axis='y', style='plain')
```



The model's predictions tend to be more conservative or cautious compared to the test data. The forecasted values are generally lower or more moderate than the actual values observed during the evaluation phase.

Washington_20009 Forecasting

In [648]:
```
r2_preds = r2_autoarima.predict(n_periods=r2_test.shape[0], return_conf_int=Tr
```

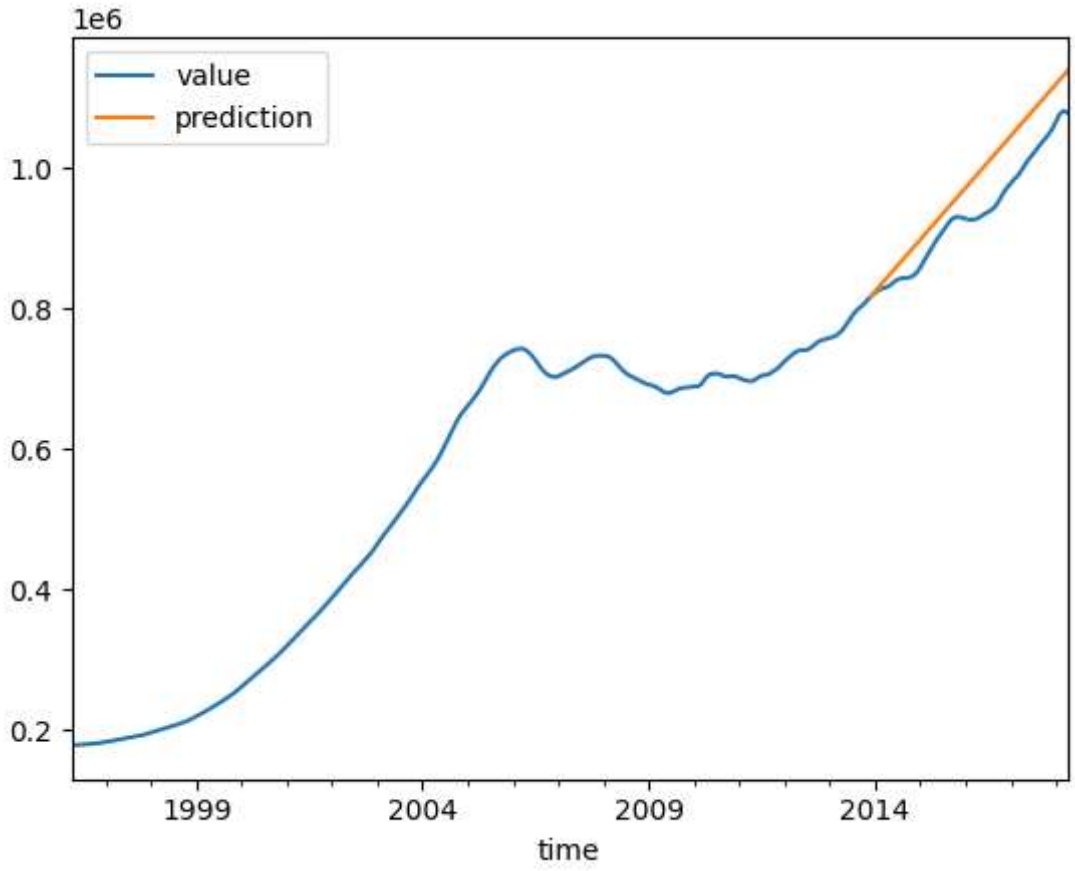
In [649]:
```
r2_forecastdf = pd.DataFrame(r2_preds[0], index=r2_test.index, columns=['predi
r2_forecastdf.head(10)
```

Out[649]:

time	prediction
2013-12-01	818039.883056
2014-01-01	824180.603300
2014-02-01	830321.341118
2014-03-01	836462.079304
2014-04-01	842602.817498
2014-05-01	848743.555693
2014-06-01	854884.293887
2014-07-01	861025.032082
2014-08-01	867165.770276
2014-09-01	873306.508471

In [650]:
```
pd.concat([melted_r2['value'], r2_forecastdf], axis=1).plot()
```

Out[650]: <AxesSubplot: xlabel='time'>



In [651]:
```
periods = r2_forecastdf.shape[0] + 36
periods
```

Out[651]: 89

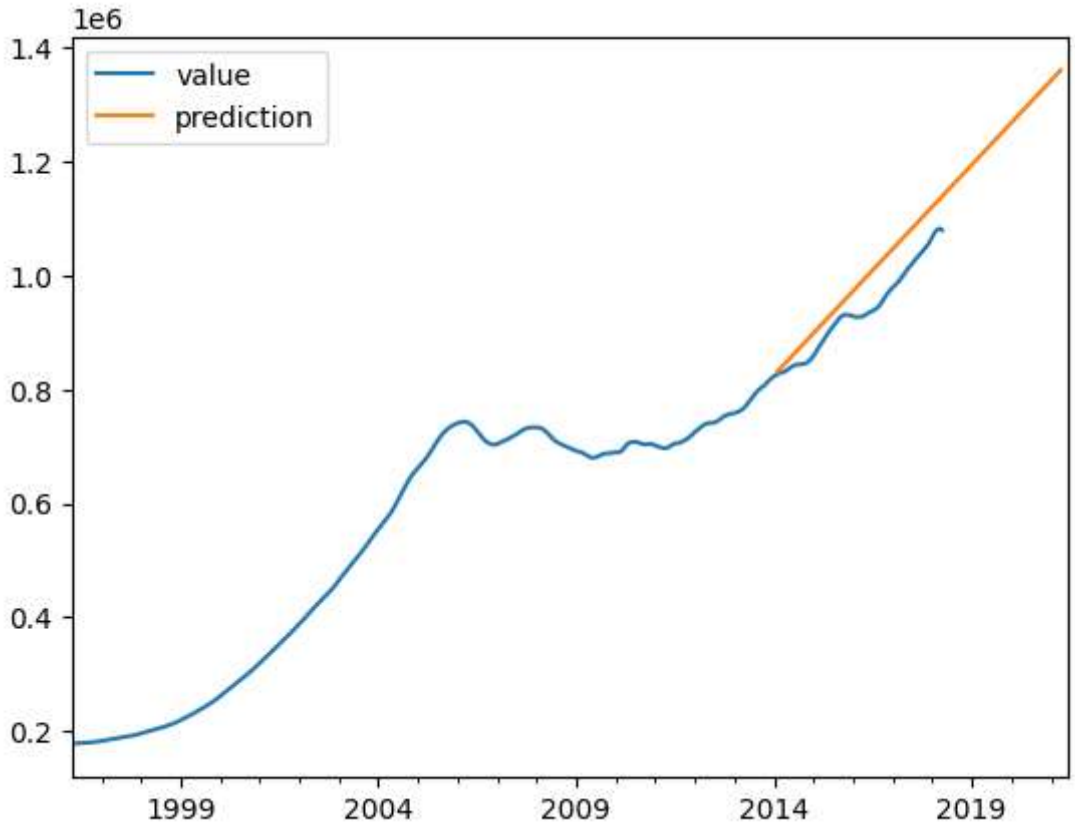
In [652]:
```
r2_future_forecast = r2_autoarima.predict(n_periods=periods,
                                          return_conf_int=True)
```

In [653]:
```
#forecast into investment horizon of 3 years
r2_forecast_range = pd.date_range(start='2014-02-01',
                                  periods=periods,
                                  freq='MS')
```

In [654]:
```
r2_future_forecastdf = pd.DataFrame(r2_future_forecast[0], index=r2_forecast_r
                                    columns=['prediction'])
```

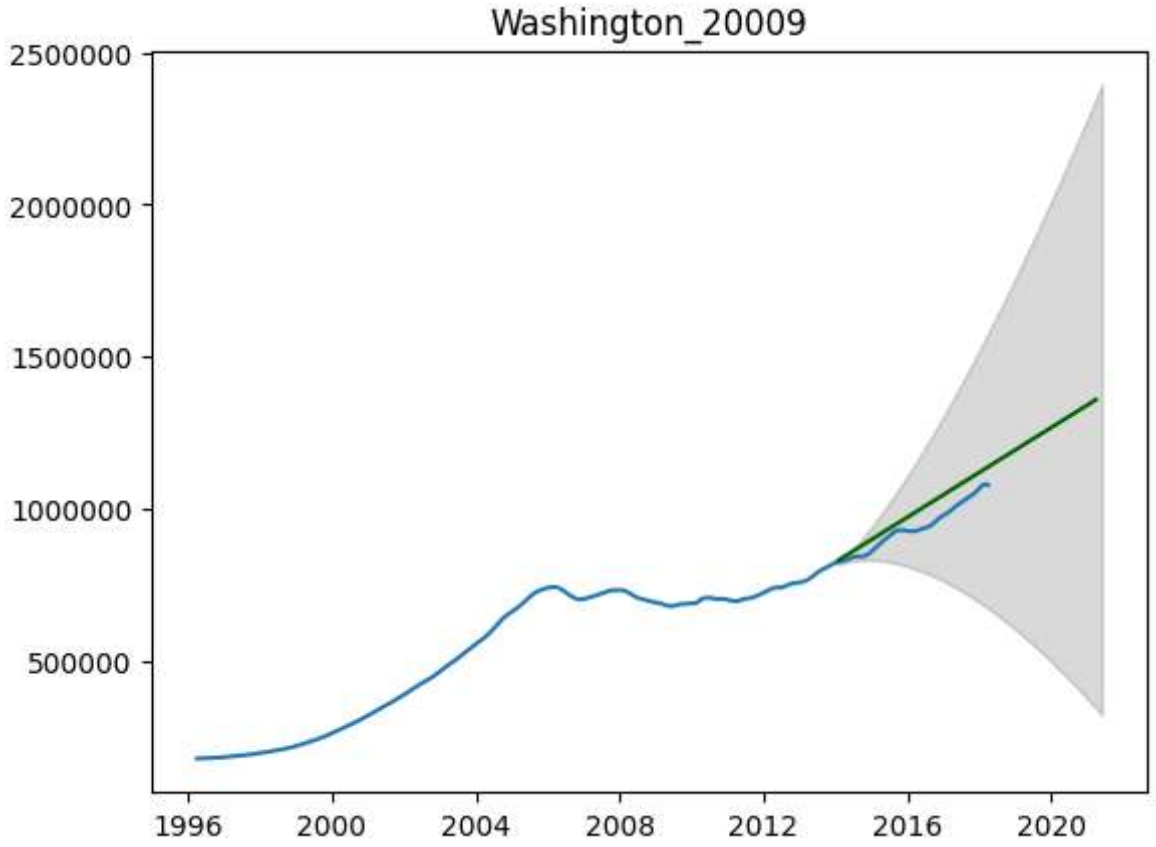
In [655]:
```
r2_combo = pd.concat([melted_r2['value'], r2_future_forecastdf], axis=1)
r2_combo.plot()
```

Out[655]: <AxesSubplot: >



In [656]:
```
lower_r2 = pd.Series(r2_future_forecast[1][:,0], index=r2_forecast_range)
upper_r2 = pd.Series(r2_future_forecast[1][:,1], index=r2_forecast_range)
```

In [657]:
```
plt.plot(melted_r2['value'])
plt.plot(r2_future_forecastdf, color='darkgreen')
plt.fill_between(r2_forecast_range,
                 lower_r2,
                 upper_r2,
                 color='k', alpha=.15)
plt.title('Washington_20009')
plt.ticklabel_format(useOffset=False, axis='y', style='plain')
```



The predictions for Washington_20009 align well with the test data, showing a good fit. The confidence interval of the prediction is depicted by the gray area, indicating the range within which the actual values are likely to fall. This visual representation demonstrates the accuracy and reliability of the predictions for Washington_20009.

Washington_20002 Forecasting

In [658]:
```
r1_preds = r1_autoarima.predict(n_periods=r1_test.shape[0], return_conf_int=Tr
```

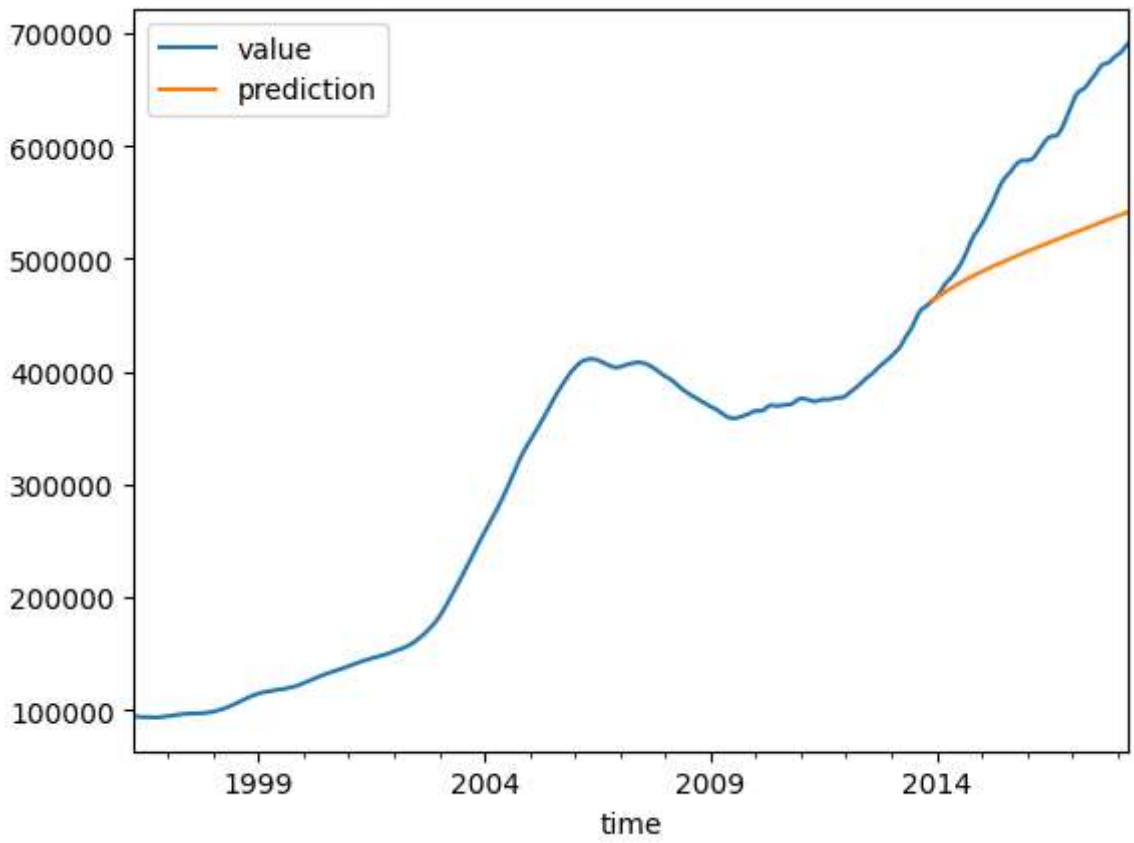
In [659]:
```
r1_forecastdf = pd.DataFrame(r1_preds[0], index=r1_test.index, columns=['predi
r1_forecastdf.head(10)
```

Out[659]:

time	prediction
2013-12-01	462790.122885
2014-01-01	465258.766805
2014-02-01	467617.078929
2014-03-01	469875.183540
2014-04-01	472042.275893
2014-05-01	474126.707468
2014-06-01	476136.063395
2014-07-01	478077.232775
2014-08-01	479956.472553
2014-09-01	481779.465520

In [660]:
```
pd.concat([melted_r1['value'], r1_forecastdf], axis=1).plot()
```

Out[660]: <AxesSubplot: xlabel='time'>



In [661]:
```
periods = r1_forecastdf.shape[0] + 36
periods
```

Out[661]: 89

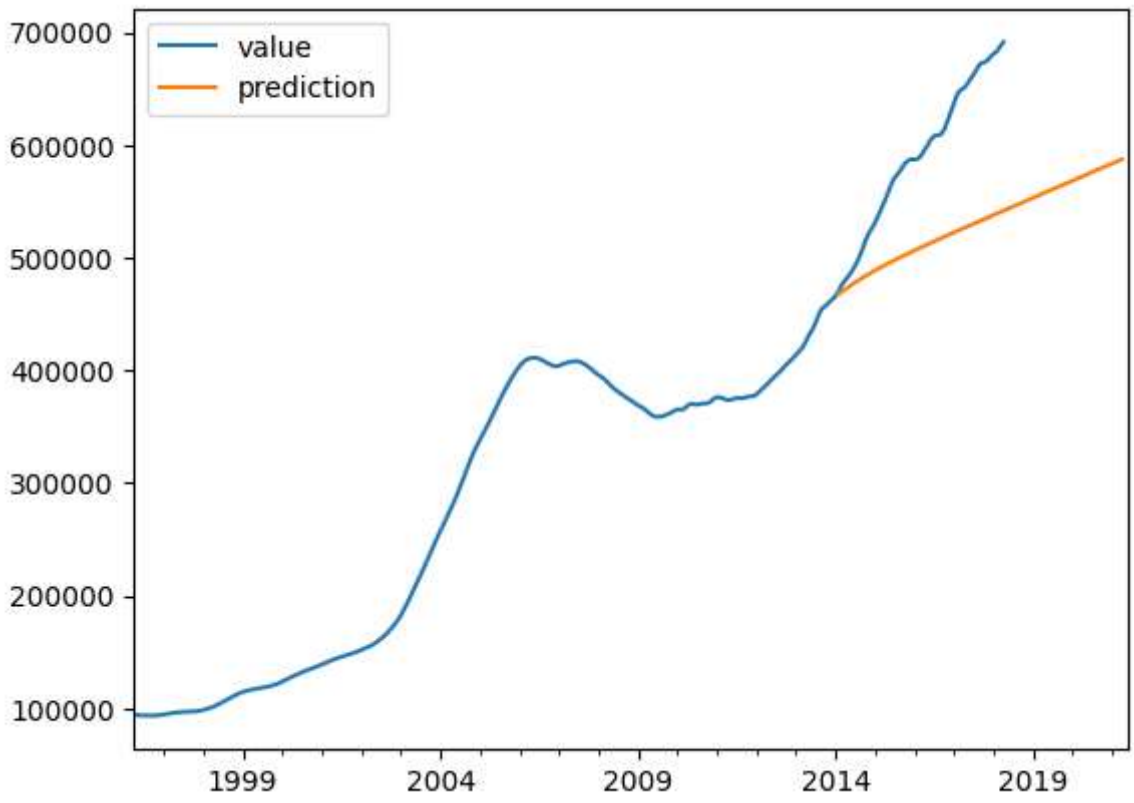
In [662]:
```
r1_future_forecast = r1_autoarima.predict(n_periods=periods,
                                          return_conf_int=True)
```

In [663]:
```
#forecast into investment horizon of 3 years
r1_forecast_range = pd.date_range(start='2014-02-01',
                                  periods=periods,
                                  freq='MS')
```

In [664]:
```
r1_future_forecastdf = pd.DataFrame(r1_future_forecast[0], index=r1_forecast_r
                                    columns=['prediction'])
```

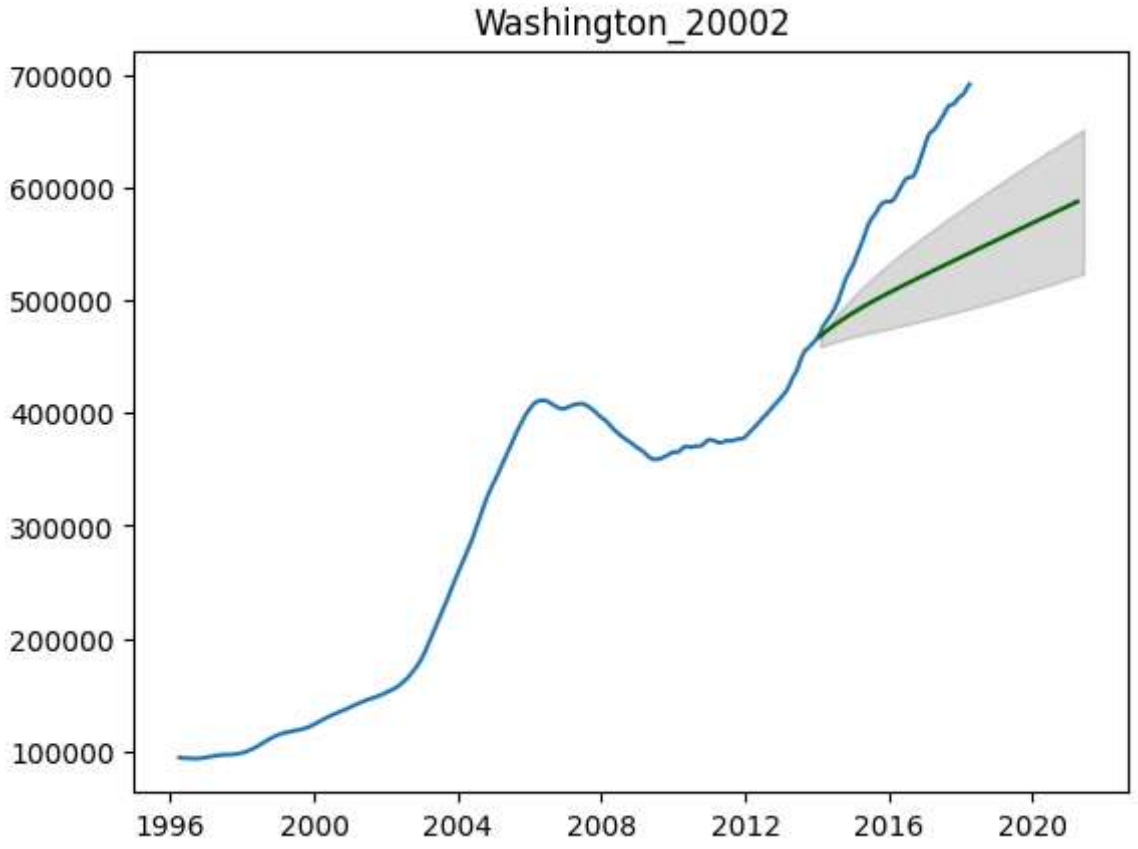
In [665]:
```
r1_combo = pd.concat([melted_r1['value'], r1_future_forecastdf], axis=1)
r1_combo.plot()
```

Out[665]: <AxesSubplot: >



In [666]:
```
lower_r1 = pd.Series(r1_future_forecast[1][:,0], index=r1_forecast_range)
upper_r1 = pd.Series(r1_future_forecast[1][:,1], index=r1_forecast_range)
```

In [667]:
```
plt.plot(melted_r1['value'])
plt.plot(r1_future_forecastdf, color='darkgreen')
plt.fill_between(r1_forecast_range,
                 lower_r1,
                 upper_r1,
                 color='k', alpha=.15)
plt.title('Washington_20002')
plt.ticklabel_format(useOffset=False, axis='y', style='plain')
```



Our forecasts tend to provide more conservative estimates compared to the test data. The predicted values are generally lower or more moderate than the actual values observed during the evaluation phase.

INTERPRETING RESULTS

In [668]:
```
r5_future_forecastdf
```

Out[668]:

	prediction
2014-02-01	1.277531e+06
2014-03-01	1.282265e+06
2014-04-01	1.286556e+06
2014-05-01	1.290446e+06
2014-06-01	1.293973e+06
...	...
2021-02-01	1.328198e+06
2021-03-01	1.328200e+06
2021-04-01	1.328201e+06
2021-05-01	NaN
2021-06-01	NaN

89 rows × 1 columns

In [669]:
```
r5_future_forecastdf['prediction'][-3]
```

Out[669]:
```
1328200.7847333043
```

we will calculate and compare the return on investment (ROI) for each region. The ROI will indicate which regions are projected to generate the highest profits relative to the initial investment. This analysis allows us to identify the regions that are expected to yield the most favorable returns compared to the amount of money invested.

In [670]:
```
#calculate r5_ROI for forecasted investment horizon
r5_ROI = (r5_future_forecastdf['prediction'][-3] / r5_future_forecastdf['predi
r5_ROI
```

Out[670]:
```
0.03966193016938924
```

In [671]:
```
#calculate r4_ROI for forecasted investment horizon
r4_ROI = r4_future_forecastdf['prediction'][-3] / r4_future_forecastdf['predic
r4_ROI
```

Out[671]:
```
0.6949704498854179
```

In [672]:
```
#calculate r3_ROI for forecasted investment horizon
r3_ROI = r3_future_forecastdf['prediction'][-3] / r3_future_forecastdf['predic
r3_ROI
```

Out[672]:
```
0.24394052212054973
```

In [673]:
```
#calculate r2_ROI for forecasted investment horizon
r2_ROI = r2_future_forecastdf['prediction'][-3] / r2_future_forecastdf['predic
r2_ROI
```

Out[673]:
```
0.6360230173132004
```

In [674]:
```
#calculate r1_ROI for forecasted investment horizon
r1_ROI = r1_future_forecastdf['prediction'][-3] / r1_future_forecastdf['predic
r1_ROI
```

Out[674]:
```
0.25599092139291413
```

In [675]:
```
df_ROI = [r1_ROI, r2_ROI, r3_ROI, r4_ROI, r5_ROI]
df_ROI
```

Out[675]:
```
[0.25599092139291413,
 0.6360230173132004,
 0.24394052212054973,
 0.6949704498854179,
 0.03966193016938924]
```

In [676]:
```
index_label = ['Washington_20002','Washington_20009', "NewYork_11226","LosAnge
```

In [677]:
```
#create dataframe for ROI
dfROI = pd.DataFrame(data=df_ROI, index=index_label)
dfROI
```

Out[677]:

	0
Washington_20002	0.255991
Washington_20009	0.636023
NewYork_11226	0.243941
LosAngeles_90046	0.694970
NewYork_11230	0.039662

In [678]:
```
dfROI.rename(columns={0:'Return on Investment'}, inplace=True)
dfROI
```

Out[678]:

	Return on Investment
Washington_20002	0.255991
Washington_20009	0.636023
NewYork_11226	0.243941
LosAngeles_90046	0.694970
NewYork_11230	0.039662

The analysis of the provided data reveals the following insights:

- Among the cities, Los Angeles (90046) shows the highest ROI of approximately 69.5%, while New York (11230) has the lowest ROI at around 3.96%.
- Washington (20009) also demonstrates a relatively high ROI of about 63.6%.

Moreover, the average property value over time exhibits an increasing trend for the analyzed locations. These findings highlight the potential profitability of real estate investments in Los

In [679]:
```
#define risk free rate (from FRED, St. Louis fed data)
riskfree = 0.0248
```

In [680]:
```
#get holding period returns
r1_monthly = r1_future_forecastdf.pct_change()
r1_monthly.head()
```

Out[680]:

	prediction
2014-02-01	NaN
2014-03-01	0.004829
2014-04-01	0.004612
2014-05-01	0.004416
2014-06-01	0.004238

In [681]:
```
#calculate sharpe ratio
r1_std = np.std(r1_monthly)
r1_sharpe = (dfROI['Return on Investment'][0] - riskfree) / r1_std
r1_sharpe
```

Out[681]:
```
prediction    325.050216
dtype: float64
```

In [682]:
```
#give holding period returns
r2_monthly = r2_future_forecastdf.pct_change()
r2_monthly.head()
```

Out[682]:

	prediction
2014-02-01	NaN
2014-03-01	0.007396
2014-04-01	0.007341
2014-05-01	0.007288
2014-06-01	0.007235

In [683]:
```
r2_std = np.std(r2_monthly)
```

In [684]:
```
#Sharpe Ratio calculation
r2_sharpe = (dfROI['Return on Investment'][1] - riskfree) / r2_std
r2_sharpe
```

Out[684]:
```
prediction    518.163312
dtype: float64
```

In [685]:
```
#calculate holding period returns
r3_monthly = r3_future_forecastdf.pct_change()
r3_monthly.head()
```

Out[685]:

	prediction
2014-02-01	NaN
2014-03-01	0.003499
2014-04-01	0.003267
2014-05-01	0.003107
2014-06-01	0.002997

In [686]:
```
#calculate sharpe ratio
r3_std = np.std(r3_monthly)
r3_sharpe = (dfROI['Return on Investment'][2] - riskfree) / r3_std
r3_sharpe
```

Out[686]:
```
prediction    502.197161
dtype: float64
```

In [687]:
```
#calculate holding period returns for r4
r4_monthly = r4_future_forecastdf.pct_change()
r4_monthly.head()
```

Out[687]:

	prediction
2014-02-01	NaN
2014-03-01	0.008081
2014-04-01	0.008016
2014-05-01	0.007953
2014-06-01	0.007890

In [688]:
```
#calculate sharpe ratio
r4_std = np.std(r4_monthly)
r4_sharpe = (dfROI['Return on Investment'][3] - riskfree) / r4_std
r4_sharpe
```

Out[688]:
```
prediction    511.920597
dtype: float64
```

In [689]:
```
#holding period returns for r5
r5_monthly = r5_future_forecastdf.pct_change()
r5_monthly.head()
```

Out[689]:

	prediction
2014-02-01	NaN
2014-03-01	0.003705
2014-04-01	0.003347
2014-05-01	0.003024
2014-06-01	0.002733

In [690]:
```
r5_std = np.std(r5_monthly)
```

In [691]:
```
#calculate sharpe ratio
r5_sharpe = (dfROI['Return on Investment'][4] - riskfree) / r5_std
r5_sharpe
```

Out[691]:
```
prediction    18.294475
dtype: float64
```

In [692]:
```
#create dataframe for sharpe ratios
dfSharpe = [r1_sharpe, r2_sharpe, r3_sharpe, r4_sharpe, r5_sharpe]
indexlabels = ['Washington_20002','Washington_20009', "NewYork_11226","LosAnge
df_Sharpe = pd.DataFrame(data=dfSharpe, index=indexlabels)
df_Sharpe.head()
```

Out[692]:

	prediction
Washington_20002	325.050216
Washington_20009	518.163312
NewYork_11226	502.197161
LosAngeles_90046	511.920597
NewYork_11230	18.294475

In [693]:
```
df_Sharpe.rename(columns={'prediction': 'Sharpe Ratio'}, inplace=True)
df_Sharpe
```

Out[693]:

	Sharpe Ratio
Washington_20002	325.050216
Washington_20009	518.163312
NewYork_11226	502.197161
LosAngeles_90046	511.920597
NewYork_11230	18.294475

The Sharpe ratio of an investment indicates its return per unit of risk and enables investors to assess returns in relation to the level of risk taken.

In this context, Washington_20009 exhibits the highest risk-adjusted return among the investments considered. This implies that, relative to the level of risk associated with the investment, `Washington_20009` offers the most favorable returns making it potentially more attractive for investment.

CONCLUSION

In conclusion, the analysis of the real estate data provided valuable insights into the property market. Los Angeles (90046) and Washington (20009) stood out as areas with high Return on Investment (ROI) and strong potential for real estate investment. These regions exhibited a significant increase in property values over time, indicating favorable market conditions. Additionally, the geographical distribution analysis revealed the concentration of properties in certain cities, emphasizing the importance of targeted location selection. Considering these findings, it is recommended to further explore investment opportunities in Los Angeles and Washington, while conducting thorough market research, financial analysis, due diligence, and consulting with experts. By following these steps, investors can make informed decisions and potentially capitalize on the high ROI observed in these regions.

RECOMMENDATION

- Based on the analysis, it is recommended to consider real estate investment opportunities in Los Angeles (90046) and Washington (20009) due to their higher Return on Investment (ROI) compared to New York (11230). These cities have demonstrated favorable ROI percentages of approximately 69.5% and 63.6% respectively.
- The high ROI in these cities could be attributed to factors such as strong market demand, desirable locations, growing economies, and potential for future property value appreciation.

NEXT STEPS

- Conduct in-depth market research: Gather more detailed information about the real estate market in Los Angeles and Washington, including current market trends, rental demand, vacancy rates, and future development plans. This will provide a comprehensive understanding of the market dynamics and potential risks.
- Identify suitable properties: Explore property listings in the recommended areas and evaluate them based on factors such as location, property condition, price, and potential for value appreciation.
- Perform financial analysis: Assess the financial viability of the investment opportunities by calculating potential rental income, expenses (including property taxes and maintenance costs), and projected cash flow. This analysis will help determine the potential return and profitability of the investments.