PHASE 4 GROUP 10 PROJECT - TIME SERIES MODELING USING ZILLOW HOUSING DATA

Student Names

- Loise Mburuga
- Amina Saidi
- Shilton Soi
- Victor Keya
- Winnie Osolo

INTRODUCTION

Industries across the world have been exposed to dynamic trends in the market place. The Real Estate Industry has not been spared and has been forced to adapt the emerging trends to remain relevant. As a consultant for a real estate investment firm, we have been tasked to determine the best 5 zip codes to live in. The project aims to forecast real estate prices using historical prices to predict future prices. We shall set out to discover areas with the highest profitability using data from Zillow Research.

BUSINESS UNDERSTANDING

The dataset contains historical data on home prices across various zip codes, along with other relevant features such as location characteristics and market trends. The challenge lies in analyzing this data to determine which zip codes offer the most promising investment opportunities based on criteria such as price appreciation, market sta

PROBLEM STATEMENT

The real-estate investment firm needs a data-driven approach to identify the top 5 zip codes for investment. To achieve this, the firm requires accurate forecasts of future real estate prices to make profitable investment decisions. This involves:

- Forecasting real estate prices for various zip codes using historical data.
- Defining the criteria for the "best" zip codes, considering profit margins, risk, and investment horizons.
- Providing a comprehensive recommendation with a rationale that addresses potential ambiguities and aligns with the firm's investment goals.

OBJECTIVE

1. Trend Analysis

To Identify long-term trends in property values across different regions.

Purpose: To help investors understand how property values have changed over time and predict future trends.

2. Regional Comparison

To Compare property value changes across different Zip codes.

Purpose: To identify the top 5 zip codes which have seen the highest appreciation in property values, helping investors and policymakers allocate resources more effectively.

3. Seasonal Patterns

To detect any seasonal patterns in property values.

Purpose: To assist real estate agents and investors in timing their buying and selling activities.

Success Metrics

The model will be considered a success if it achieves a low RMSE and ROI is adequate

TABLE OF CONTENTS

1. Data Inspection
2. Data cleaning
3. Exploratory Data Analysis
4. Modelling
5. Evaluation
6. Conclusion
7. Recommendation

A. DATA INSPECTION

In [1]:
```
# Import the necessarry libraries for data analysis and visualization
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from statsmodels.tools.sm_exceptions import ConvergenceWarning
warnings.simplefilter('ignore', ConvergenceWarning)
sns.set()

from sklearn.metrics import mean_squared_error
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from math import sqrt
from matplotlib.pylab import rcParams
```

In [2]:
```
# Load and preview the dataset

df = pd.read_csv('zillow_data.csv')
df
```

Out[2]:

	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
...	...	...	...	...	...	...	...	...
14718	58333	1338	Ashfield	MA	Greenfield Town	Franklin	14719	94600.0
14719	59107	3293	Woodstock	NH	Claremont	Grafton	14720	92700.0
14720	75672	40404	Berea	KY	Richmond	Madison	14721	57100.0
14721	93733	81225	Mount Crested Butte	CO	NaN	Gunnison	14722	191100.0
14722	95851	89155	Mesquite	NV	Las Vegas	Clark	14723	176400.0

14723 rows × 272 columns

In [3]:
```
# Checking the first 5 rows
df.head()
```

Out[3]:

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	1996-0
0	84654	60657	Chicago	IL	Chicago	Cook	1	334200.0	335400.
1	90668	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	235700.0	236900.
2	91982	77494	Katy	TX	Houston	Harris	3	210400.0	212200.
3	84616	60614	Chicago	IL	Chicago	Cook	4	498100.0	500900.
4	93144	79936	El Paso	TX	El Paso	El Paso	5	77300.0	77300.

5 rows × 272 columns

In [4]:
```
#Checking the last 5 rows
df.tail()
```

Out[4]:

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

5 rows × 272 columns

In [5]:
```
#checking statistical summary
df.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	
RegionID	14723.0	81075.010052	31934.118525	58196.0	67174.5	78007.0	90920.5	1
RegionName	14723.0	48222.348706	29359.325439	1001.0	22101.5	46106.0	75205.5	
SizeRank	14723.0	7362.000000	4250.308342	1.0	3681.5	7362.0	11042.5	
1996-04	13684.0	118299.123063	86002.509608	11300.0	68800.0	99500.0	143200.0	3(
1996-05	13684.0	118419.044139	86155.673905	11500.0	68900.0	99500.0	143300.0	3
...	...	...	...	...	...	...	...	
2017-12	14723.0	281095.320247	367045.388033	14300.0	129900.0	193400.0	313400.0	18
2018-01	14723.0	282657.060382	369572.741938	14100.0	130600.0	194100.0	315100.0	18
2018-02	14723.0	284368.688447	371773.905107	13900.0	131050.0	195000.0	316850.0	18
2018-03	14723.0	286511.376757	372461.234695	13800.0	131950.0	196700.0	318850.0	18
2018-04	14723.0	288039.944305	372054.396908	13800.0	132400.0	198100.0	321100.0	17

268 rows × 8 columns

In [6]:
```
#Renaming RegionName to ZipCode
df = df.copy()
# Rename the column using inplace=True
df.rename(columns={'RegionName': 'ZipCode'}, inplace=True)

# Verify the changes
print(df.columns)
print(df.head())
```

```
Index(['RegionID', 'ZipCode', '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)
   RegionID  ZipCode      City State              Metro CountyName  SizeRank
\
0     84654    60657   Chicago    IL            Chicago       Cook         1
1     90668    75070  McKinney    TX  Dallas-Fort Worth     Collin         2
2     91982    77494      Katy    TX            Houston     Harris         3
3     84616    60614   Chicago    IL            Chicago       Cook         4
4     93144    79936   El Paso    TX            El Paso    El Paso         5

    1996-04   1996-05   1996-06  ...  2017-07  2017-08  2017-09  2017-10  \
0  334200.0  335400.0  336500.0  ...  1005500  1007500  1007800  1009600
1  235700.0  236900.0  236700.0  ...   308000   310000   312500   314100
2  210400.0  212200.0  212200.0  ...   321000   320600   320200   320400
3  498100.0  500900.0  503100.0  ...  1289800  1287700  1287400  1291500
4   77300.0   77300.0   77300.0  ...   119100   119400   120000   120300

   2017-11  2017-12  2018-01  2018-02  2018-03  2018-04
0  1013300  1018700  1024400  1030700  1033800  1030600
1   315000   316600   318100   319600   321100   321800
2   320800   321200   321200   323000   326900   329900
3  1296600  1299000  1302700  1306400  1308500  1307000
4   120300   120300   120300   120500   121000   121500

[5 rows x 272 columns]
```

In [7]:
```
df = pd.melt(df,
                  id_vars=['RegionID', 'ZipCode', 'City', 'State', 'Metro', '
                  var_name='time',
                  value_name='value')

df.head()
```

Out[7]:

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

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3901595 entries, 0 to 3901594
Data columns (total 9 columns):
 #   Column      Dtype  
---  ------      -----  
 0   RegionID    int64  
 1   ZipCode     int64  
 2   City        object 
 3   State       object 
 4   Metro       object 
 5   CountyName  object 
 6   SizeRank    int64  
 7   time        object 
 8   value       float64
dtypes: float64(1), int64(3), object(5)
memory usage: 267.9+ MB
```

The columns have 3 data types:

The data contains 272 columns and 14723 rows.

In [9]:
```
df.describe()
```

Out[9]:

	RegionID	ZipCode	SizeRank	value
count	3.901595e+06	3.901595e+06	3.901595e+06	3.744704e+06
mean	8.107501e+04	4.822235e+04	7.362000e+03	2.076064e+05
std	3.193304e+04	2.935833e+04	4.250165e+03	2.400207e+05
min	5.819600e+04	1.001000e+03	1.000000e+00	1.130000e+04
25%	6.717400e+04	2.210100e+04	3.681000e+03	9.790000e+04
50%	7.800700e+04	4.610600e+04	7.362000e+03	1.476000e+05
75%	9.092100e+04	7.520600e+04	1.104300e+04	2.372000e+05
max	7.538440e+05	9.990100e+04	1.472300e+04	1.931490e+07

In [10]:
```
df.shape
```

Out[10]:
```
(3901595, 9)
```

B. DATA CLEANING

In [11]:
```
# Finding null values
df.isnull().sum()
```

Out[11]:
```
RegionID          0
ZipCode           0
City              0
State             0
Metro        276395
CountyName        0
SizeRank          0
time              0
value        156891
dtype: int64
```

In [12]:
```
# Calculate the percentage of missing values for each column
missing_percentage = df.isnull().mean() * 100
missing_percentage
```

Out[12]:
```
RegionID      0.000000
ZipCode       0.000000
City          0.000000
State         0.000000
Metro         7.084154
CountyName    0.000000
SizeRank      0.000000
time          0.000000
value         4.021202
dtype: float64
```

In [13]:
```
df.drop('Metro', axis=1, inplace=True)
```

In [14]:
```
df1 = df.dropna(subset= ['value'])
df1.isnull().sum()
```

Out[14]:
```
RegionID      0
ZipCode       0
City          0
State         0
CountyName    0
SizeRank      0
time          0
value         0
dtype: int64
```

In [15]:
```
df1.shape
```

Out[15]:
```
(3744704, 8)
```

In [16]:
```
# Checking for duplicated entries
df1.duplicated().sum()
```

Out[16]:
```
0
```

In [17]:
```
# Checking for placeholders
# Define a comprehensive list of potential placeholder values
common_placeholders = ["", "na", "n/a", "nan", "none", "null", "-", "--", "?"

# Loop through each column and check for potential placeholders

found_placeholder = False
for column in df1.columns:
  unique_values = df1[column].unique()
  for value in unique_values:
    if pd.isna(value) or (isinstance(value, str) and value.strip().lower() in
      count = (df1[column] == value).sum()
      print(f"Column '{column}': Found {count} occurrences of potential place

      found_placeholder = True

if not found_placeholder:
  print("No potential placeholders found in the DataFrame.")
```

```
No potential placeholders found in the DataFrame.
```

In [18]:
```
df1.head()
```

Out[18]:

	RegionID	ZipCode	City	State	CountyName	SizeRank	time	value
0	84654	60657	Chicago	IL	Cook	1	1996-04	334200.0
1	90668	75070	McKinney	TX	Collin	2	1996-04	235700.0
2	91982	77494	Katy	TX	Harris	3	1996-04	210400.0
3	84616	60614	Chicago	IL	Cook	4	1996-04	498100.0
4	93144	79936	El Paso	TX	El Paso	5	1996-04	77300.0

In [19]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3744704 entries, 0 to 3901594
Data columns (total 8 columns):
 #   Column      Dtype  
---  ------      -----  
 0   RegionID    int64  
 1   ZipCode     int64  
 2   City        object 
 3   State       object 
 4   CountyName  object 
 5   SizeRank    int64  
 6   time        object 
 7   value       float64
dtypes: float64(1), int64(3), object(4)
memory usage: 257.1+ MB
```

In [20]:
```
# Convert 'time' column to datetime
df1.loc[:, 'time'] = pd.to_datetime(df1['time'], format='%Y-%m')
```

In [21]:
```
from datetime import datetime, timedelta
# Define the most recent date in the dataset
most_recent_date = df1['time'].max()

# Calculate the date 10 years ago from the most recent date
ten_years_ago = most_recent_date - timedelta(days=10*365)

# Filter the DataFrame for the last 10 years
df1_last_10_years = df1[df1['time'] >= ten_years_ago]

print("Filtered DataFrame for the last 10 years:\n", df1_last_10_years)
#df1_last_10_years = df2
```

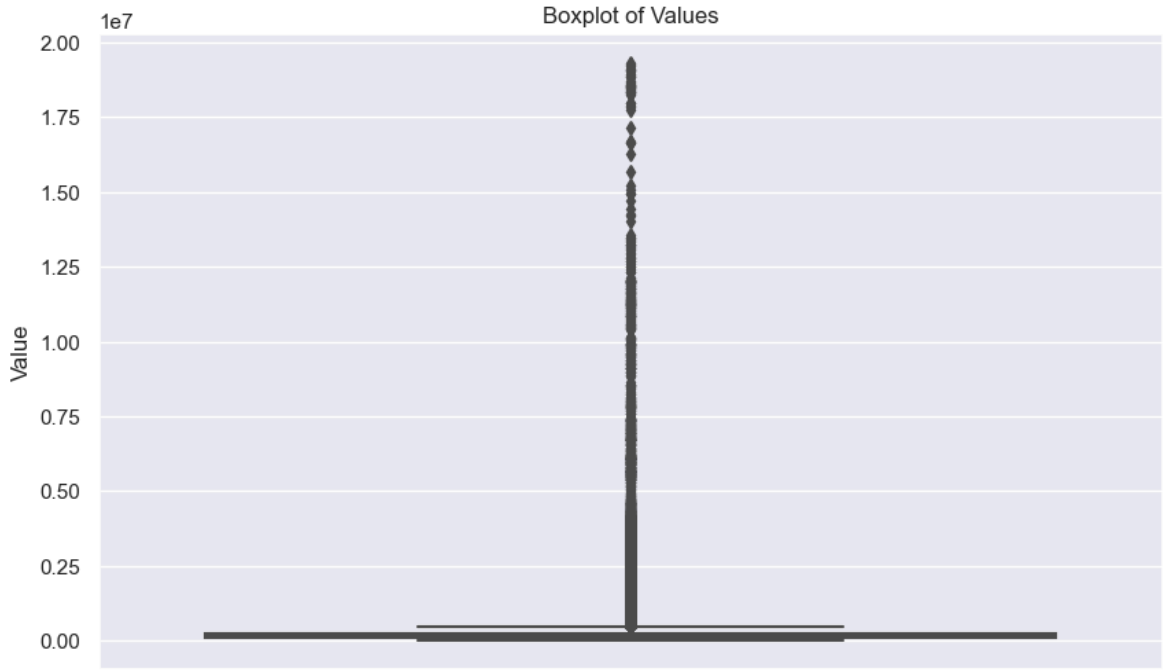
```
Filtered DataFrame for the last 10 years:
          RegionID  ZipCode                 City State CountyName  SizeRank
\
2134835     84654    60657              Chicago    IL       Cook         1
2134836     90668    75070            McKinney    TX     Collin         2
2134837     91982    77494                Katy    TX     Harris         3
2134838     84616    60614              Chicago    IL       Cook         4
2134839     93144    79936              El Paso    TX    El Paso         5
...           ...      ...                  ...   ...        ...       ...
3901590     58333     1338             Ashfield    MA   Franklin     14719
3901591     59107     3293            Woodstock    NH    Grafton     14720
3901592     75672    40404                Berea    KY    Madison     14721
3901593     93733    81225  Mount Crested Butte    CO   Gunnison     14722
3901594     95851    89155              Mesquite    NV      Clark     14723

                        time      value
2134835  2008-05-01 00:00:00   859000.0
2134836  2008-05-01 00:00:00   209500.0
2134837  2008-05-01 00:00:00   248300.0
2134838  2008-05-01 00:00:00  1135600.0
2134839  2008-05-01 00:00:00   127300.0
...                      ...        ...
3901590  2018-04-01 00:00:00   209300.0
3901591  2018-04-01 00:00:00   225800.0
3901592  2018-04-01 00:00:00   133400.0
3901593  2018-04-01 00:00:00   664400.0
3901594  2018-04-01 00:00:00   357200.0

[1742922 rows x 8 columns]
```

In [22]:
```
df2 = df1_last_10_years
df2.info()
```

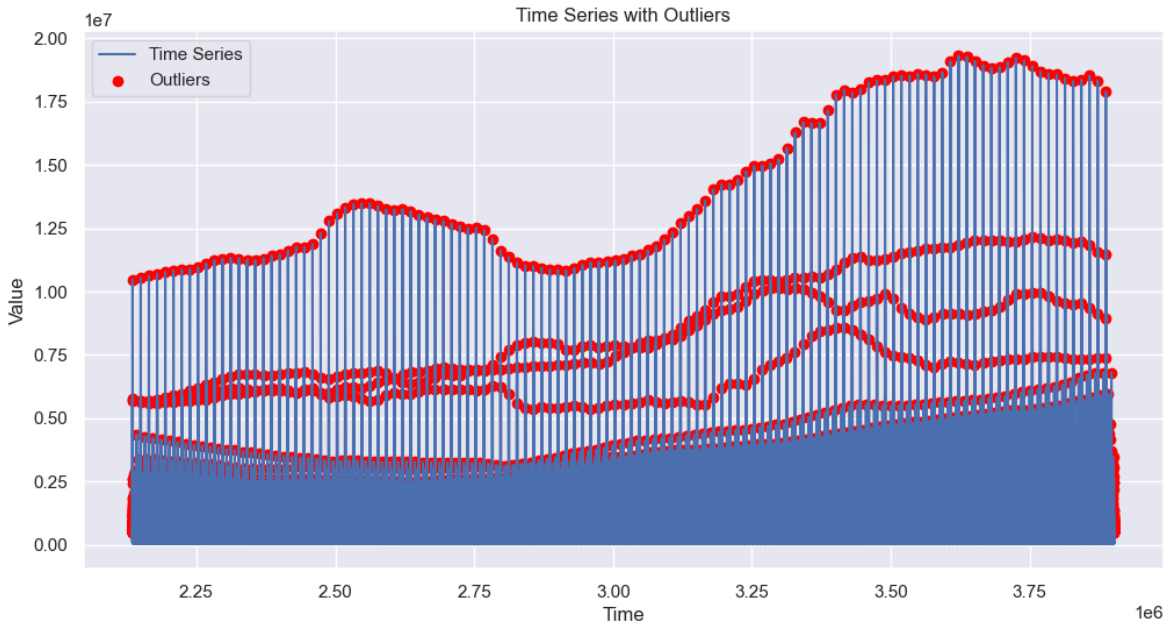
```
<class 'pandas.core.frame.DataFrame'>
Index: 1742922 entries, 2134835 to 3901594
Data columns (total 8 columns):
 #   Column      Dtype  
---  ------      -----  
 0   RegionID    int64  
 1   ZipCode     int64  
 2   City        object 
 3   State       object 
 4   CountyName  object 
 5   SizeRank    int64  
 6   time        object 
 7   value       float64
dtypes: float64(1), int64(3), object(4)
memory usage: 119.7+ MB
```

Checking For Outliers

In [23]:
```
# Create a boxplot for the 'value' column
plt.figure(figsize=(10, 6))
sns.boxplot(y=df2['value'])
plt.title('Boxplot of Values')
plt.ylabel('Value')
plt.show()
```



In [24]:
```
# Outlier detection
Q1 = df2['value'].quantile(0.25)
Q3 = df2['value'].quantile(0.75)
IQR = Q3 - Q1
outliers = df2[(df2['value'] < (Q1 - 1.5 * IQR)) | (df2['value'] > (Q3 + 1.5

plt.figure(figsize=(12, 6))
plt.plot(df2.index, df2['value'], label='Time Series')
plt.scatter(outliers.index, outliers['value'], color='red', label='Outliers')
plt.title('Time Series with Outliers')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend()
plt.show()
```



Checking for outliers in the value column. There was no significant outlier in the data.

Using ROI to identify most suitable zipcodes to invest in

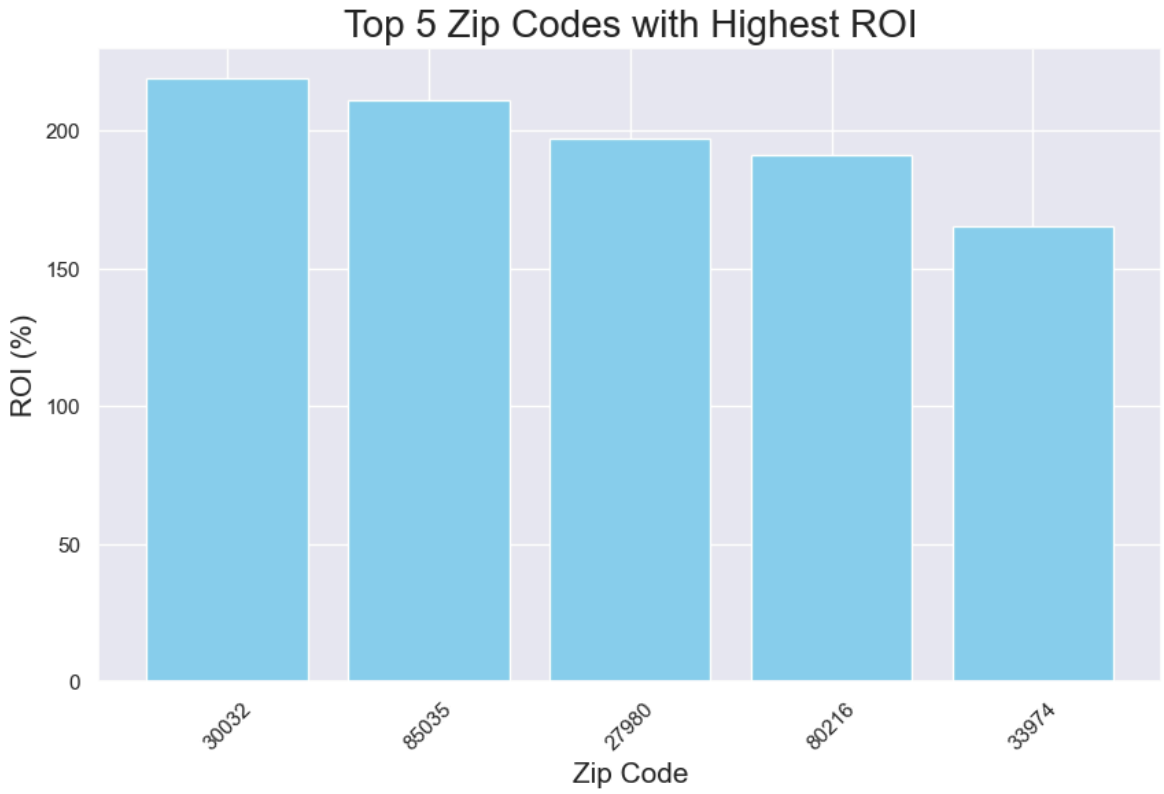
We used Return on Investment(ROI) to identify the top 5 zipcodes that would be most profitable to invest in.

In [25]:
```
# Calculate the initial and final values for each ZipCode
initial_values = df2.groupby('ZipCode').first()['value']
final_values = df2.groupby('ZipCode').last()['value']

# Calculate ROI for each ZipCode
roi = (final_values - initial_values) / initial_values * 100

# Convert the Series to a DataFrame for easier handling
roi_df = roi.reset_index()
roi_df.columns = ['ZipCode', 'ROI']

# Identify the top 5 zip codes with the highest ROI
top_5_zipcodes = roi_df.nlargest(5, 'ROI')

top_5_zipcodes
```

Out[25]:

	ZipCode	ROI
4748	30032	219.014085
12468	85035	211.411992
4288	27980	197.314050
12007	80216	191.176471
5694	33974	165.645161

In [26]:
```
# Plot the bar graph for the top 5 zip codes
plt.figure(figsize=(10, 6))
plt.bar(top_5_zipcodes['ZipCode'].astype(str), top_5_zipcodes['ROI'], color='
plt.xlabel('Zip Code', fontsize=15)
plt.ylabel('ROI (%)', fontsize=15)
plt.title('Top 5 Zip Codes with Highest ROI', fontsize=20)
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



In [27]:
```
merged_df = pd.merge(df2, top_5_zipcodes, on='ZipCode')
merged_df
```

Out[27]:

	RegionID	ZipCode	City	State	CountyName	SizeRank	time	value	ROI
0	94751	85035	Phoenix	AZ	Maricopa	1947	2010-07-01 00:00:00	51700.0	211.411992
1	94751	85035	Phoenix	AZ	Maricopa	1947	2010-08-01 00:00:00	51800.0	211.411992
2	94751	85035	Phoenix	AZ	Maricopa	1947	2010-09-01 00:00:00	51100.0	211.411992
3	94751	85035	Phoenix	AZ	Maricopa	1947	2010-10-01 00:00:00	50500.0	211.411992
4	94751	85035	Phoenix	AZ	Maricopa	1947	2010-11-01 00:00:00	49700.0	211.411992
...	...	...	...	...	...	...	...	...	...
345	69649	27980	Hertford	NC	Perquimans	13410	2017-12-01 00:00:00	122200.0	197.314050
346	69649	27980	Hertford	NC	Perquimans	13410	2018-01-01 00:00:00	126600.0	197.314050
347	69649	27980	Hertford	NC	Perquimans	13410	2018-02-01 00:00:00	130600.0	197.314050
348	69649	27980	Hertford	NC	Perquimans	13410	2018-03-01 00:00:00	137300.0	197.314050
349	69649	27980	Hertford	NC	Perquimans	13410	2018-04-01 00:00:00	143900.0	197.314050

350 rows × 9 columns

In [28]:
```
df3 = merged_df
df3
```

Out[28]:

	RegionID	ZipCode	City	State	CountyName	SizeRank	time	value	ROI
0	94751	85035	Phoenix	AZ	Maricopa	1947	2010-07-01 00:00:00	51700.0	211.411992
1	94751	85035	Phoenix	AZ	Maricopa	1947	2010-08-01 00:00:00	51800.0	211.411992
2	94751	85035	Phoenix	AZ	Maricopa	1947	2010-09-01 00:00:00	51100.0	211.411992
3	94751	85035	Phoenix	AZ	Maricopa	1947	2010-10-01 00:00:00	50500.0	211.411992
4	94751	85035	Phoenix	AZ	Maricopa	1947	2010-11-01 00:00:00	49700.0	211.411992
...	...	...	...	...	...	...	...	...	...
345	69649	27980	Hertford	NC	Perquimans	13410	2017-12-01 00:00:00	122200.0	197.314050
346	69649	27980	Hertford	NC	Perquimans	13410	2018-01-01 00:00:00	126600.0	197.314050
347	69649	27980	Hertford	NC	Perquimans	13410	2018-02-01 00:00:00	130600.0	197.314050
348	69649	27980	Hertford	NC	Perquimans	13410	2018-03-01 00:00:00	137300.0	197.314050
349	69649	27980	Hertford	NC	Perquimans	13410	2018-04-01 00:00:00	143900.0	197.314050

350 rows × 9 columns

C. RESAMPLING

Monythly data

In [29]:
```
# Convert 'time' to datetime
df3['time'] = pd.to_datetime(df3['time'])
# Extract year
df3['year'] = df3['time'].dt.year

# Group by year and calculate mean
df_monthly = df3.groupby(['RegionID', 'ZipCode', 'City', 'State', 'CountyName

# C# Convert 'time' column to datetime format with only the year
df_monthly['time'] = pd.to_datetime(df_monthly['time'], format='%Y')

# Set the 'time' column as the index
df_monthly.set_index('time', inplace=True)

# Sort the DataFrame by the datetime index (optional but recommended)
df_monthly.sort_index(inplace=True)

# Display the DataFrame to verify the changes
df_monthly
```

Out[29]:

time	RegionID	ZipCode	City	State	CountyName	SizeRank	value	ROI
2010-07-01	94751	85035	Phoenix	AZ	Maricopa	1947	51700.0	211.411992
2010-08-01	94751	85035	Phoenix	AZ	Maricopa	1947	51800.0	211.411992
2010-09-01	94751	85035	Phoenix	AZ	Maricopa	1947	51100.0	211.411992
2010-10-01	94751	85035	Phoenix	AZ	Maricopa	1947	50500.0	211.411992
2010-11-01	94751	85035	Phoenix	AZ	Maricopa	1947	49700.0	211.411992
...	...	...	...	...	...	...	...	...
2018-04-01	70817	30032	Candler-Mcafee	GA	Dekalb	843	135900.0	219.014085
2018-04-01	94751	85035	Phoenix	AZ	Maricopa	1947	161000.0	211.411992
2018-04-01	93292	80216	Denver	CO	Denver	7303	267300.0	191.176471
2018-04-01	69649	27980	Hertford	NC	Perquimans	13410	143900.0	197.314050
2018-04-01	399584	33974	Lehigh Acres	FL	Lee	7663	164700.0	165.645161

350 rows × 8 columns

In [30]:
```
# Resample the data to monthly frequency and calculate the mean for each grou
resampled_df = df_monthly.groupby(['RegionID', 'ZipCode', 'City', 'State', 'C
resampled_df
```

Out[30]:

	RegionID	ZipCode	City	State	CountyName	SizeRank	time	value	ROI
0	69649	27980	Hertford	NC	Perquimans	13410	2014-01-31	48400.0	197.314050
1	69649	27980	Hertford	NC	Perquimans	13410	2014-02-28	48800.0	197.314050
2	69649	27980	Hertford	NC	Perquimans	13410	2014-03-31	49500.0	197.314050
3	69649	27980	Hertford	NC	Perquimans	13410	2014-04-30	50400.0	197.314050
4	69649	27980	Hertford	NC	Perquimans	13410	2014-05-31	50100.0	197.314050
...	...	...	...	...	...	...	...	...	...
345	399584	33974	Lehigh Acres	FL	Lee	7663	2017-12-31	160900.0	165.645161
346	399584	33974	Lehigh Acres	FL	Lee	7663	2018-01-31	161200.0	165.645161
347	399584	33974	Lehigh Acres	FL	Lee	7663	2018-02-28	162100.0	165.645161
348	399584	33974	Lehigh Acres	FL	Lee	7663	2018-03-31	163400.0	165.645161
349	399584	33974	Lehigh Acres	FL	Lee	7663	2018-04-30	164700.0	165.645161

350 rows × 9 columns

In [31]:
```
df4 =df_monthly
df4
```

Out[31]:

time	RegionID	ZipCode	City	State	CountyName	SizeRank	value	ROI
2010-07-01	94751	85035	Phoenix	AZ	Maricopa	1947	51700.0	211.411992
2010-08-01	94751	85035	Phoenix	AZ	Maricopa	1947	51800.0	211.411992
2010-09-01	94751	85035	Phoenix	AZ	Maricopa	1947	51100.0	211.411992
2010-10-01	94751	85035	Phoenix	AZ	Maricopa	1947	50500.0	211.411992
2010-11-01	94751	85035	Phoenix	AZ	Maricopa	1947	49700.0	211.411992
...	...	...	...	...	...	...	...	...
2018-04-01	70817	30032	Candler-Mcafee	GA	Dekalb	843	135900.0	219.014085
2018-04-01	94751	85035	Phoenix	AZ	Maricopa	1947	161000.0	211.411992
2018-04-01	93292	80216	Denver	CO	Denver	7303	267300.0	191.176471
2018-04-01	69649	27980	Hertford	NC	Perquimans	13410	143900.0	197.314050
2018-04-01	399584	33974	Lehigh Acres	FL	Lee	7663	164700.0	165.645161

350 rows × 8 columns

Quarterly data

In [32]:
```
# Extract year and quarter
df_monthly['year'] = df_monthly.index.year
df_monthly['quarter'] = df_monthly.index.quarter

# Group by year, quarter, and other columns, then calculate mean
df_quarterly = df_monthly.groupby(['RegionID', 'ZipCode', 'City', 'State', 'C

# Display the result
df_quarterly
```

Out[32]:

	RegionID	ZipCode	City	State	CountyName	SizeRank	year	quarter	value
0	69649	27980	Hertford	NC	Perquimans	13410	2014	1	48900.000000
1	69649	27980	Hertford	NC	Perquimans	13410	2014	2	49866.666667
2	69649	27980	Hertford	NC	Perquimans	13410	2014	3	49366.666667
3	69649	27980	Hertford	NC	Perquimans	13410	2014	4	53933.333333
4	69649	27980	Hertford	NC	Perquimans	13410	2015	1	57933.333333
...	...	...	...	...	...	...	...	...	...
115	399584	33974	Lehigh Acres	FL	Lee	7663	2017	2	150700.000000
116	399584	33974	Lehigh Acres	FL	Lee	7663	2017	3	157366.666667
117	399584	33974	Lehigh Acres	FL	Lee	7663	2017	4	160733.333333
118	399584	33974	Lehigh Acres	FL	Lee	7663	2018	1	162233.333333
119	399584	33974	Lehigh Acres	FL	Lee	7663	2018	2	164700.000000

120 rows × 10 columns

Yearly Data

In [33]:
```
# Yearly data
# Extract year
df3['year'] = df3['time'].dt.year

# Group by year and calculate mean
df_yearly = df3.groupby(['RegionID', 'ZipCode', 'City', 'State', 'CountyName'

# Display the result
df_yearly
```

Out[33]:

	RegionID	ZipCode	City	State	CountyName	SizeRank	year	value	R(
0	69649	27980	Hertford	NC	Perquimans	13410	2014	50516.666667	197.3140
1	69649	27980	Hertford	NC	Perquimans	13410	2015	59183.333333	197.3140
2	69649	27980	Hertford	NC	Perquimans	13410	2016	73758.333333	197.3140
3	69649	27980	Hertford	NC	Perquimans	13410	2017	104283.333333	197.3140
4	69649	27980	Hertford	NC	Perquimans	13410	2018	134600.000000	197.3140
5	70817	30032	Candler-Mcafee	GA	Dekalb	843	2013	44800.000000	219.0140
6	70817	30032	Candler-Mcafee	GA	Dekalb	843	2014	53783.333333	219.0140
7	70817	30032	Candler-Mcafee	GA	Dekalb	843	2015	71350.000000	219.0140
8	70817	30032	Candler-Mcafee	GA	Dekalb	843	2016	87241.666667	219.0140
9	70817	30032	Candler-Mcafee	GA	Dekalb	843	2017	104441.666667	219.0140
10	70817	30032	Candler-Mcafee	GA	Dekalb	843	2018	130675.000000	219.0140
11	93292	80216	Denver	CO	Denver	7303	2013	103116.666667	191.1764
12	93292	80216	Denver	CO	Denver	7303	2014	130758.333333	191.1764
13	93292	80216	Denver	CO	Denver	7303	2015	165466.666667	191.1764
14	93292	80216	Denver	CO	Denver	7303	2016	204800.000000	191.1764
15	93292	80216	Denver	CO	Denver	7303	2017	241900.000000	191.1764
16	93292	80216	Denver	CO	Denver	7303	2018	260725.000000	191.1764
17	94751	85035	Phoenix	AZ	Maricopa	1947	2010	50666.666667	211.4119
18	94751	85035	Phoenix	AZ	Maricopa	1947	2011	47441.666667	211.4119
19	94751	85035	Phoenix	AZ	Maricopa	1947	2012	52833.333333	211.4119
20	94751	85035	Phoenix	AZ	Maricopa	1947	2013	72816.666667	211.4119
21	94751	85035	Phoenix	AZ	Maricopa	1947	2014	90766.666667	211.4119
22	94751	85035	Phoenix	AZ	Maricopa	1947	2015	107933.333333	211.4119
23	94751	85035	Phoenix	AZ	Maricopa	1947	2016	128791.666667	211.4119
24	94751	85035	Phoenix	AZ	Maricopa	1947	2017	145833.333333	211.4119
25	94751	85035	Phoenix	AZ	Maricopa	1947	2018	160200.000000	211.4119
26	399584	33974	Lehigh Acres	FL	Lee	7663	2011	62650.000000	165.6451
27	399584	33974	Lehigh Acres	FL	Lee	7663	2012	71508.333333	165.6451
28	399584	33974	Lehigh Acres	FL	Lee	7663	2013	84375.000000	165.6451
29	399584	33974	Lehigh Acres	FL	Lee	7663	2014	99458.333333	165.6451

	RegionID	ZipCode	City	State	CountyName	SizeRank	year	value	R
30	399584	33974	Lehigh Acres	FL	Lee	7663	2015	116583.333333	165.64516
31	399584	33974	Lehigh Acres	FL	Lee	7663	2016	134958.333333	165.64516
32	399584	33974	Lehigh Acres	FL	Lee	7663	2017	153541.666667	165.64516
33	399584	33974	Lehigh Acres	FL	Lee	7663	2018	162850.000000	165.64516

D. EXPLORATORY DATA ANALYSIS

In this section we will consider:

- Univatiate Analysis,
- Bivariate
- Multivariate Analysis

Univariate Analysis

In [34]:
```
df4 = df_monthly
df4
```

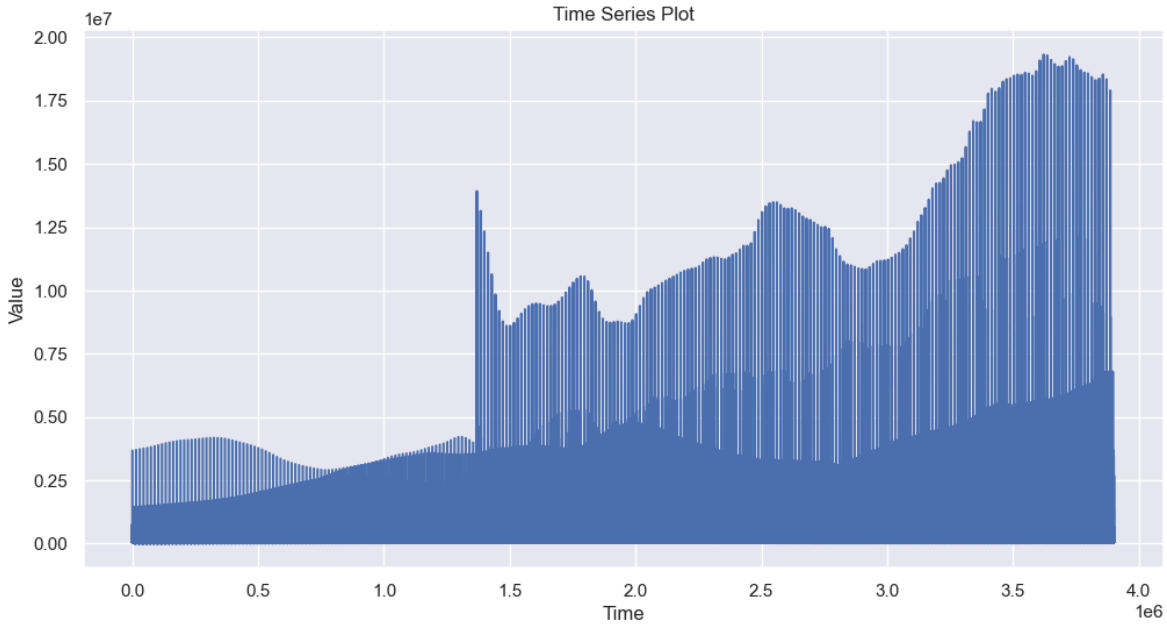
Out[34]:

time	RegionID	ZipCode	City	State	CountyName	SizeRank	value	ROI	year
2010-07-01	94751	85035	Phoenix	AZ	Maricopa	1947	51700.0	211.411992	2010
2010-08-01	94751	85035	Phoenix	AZ	Maricopa	1947	51800.0	211.411992	2010
2010-09-01	94751	85035	Phoenix	AZ	Maricopa	1947	51100.0	211.411992	2010
2010-10-01	94751	85035	Phoenix	AZ	Maricopa	1947	50500.0	211.411992	2010
2010-11-01	94751	85035	Phoenix	AZ	Maricopa	1947	49700.0	211.411992	2010
...	...	...	...	...	...	...	...	...	...
2018-04-01	70817	30032	Candler-Mcafee	GA	Dekalb	843	135900.0	219.014085	2018
2018-04-01	94751	85035	Phoenix	AZ	Maricopa	1947	161000.0	211.411992	2018
2018-04-01	93292	80216	Denver	CO	Denver	7303	267300.0	191.176471	2018
2018-04-01	69649	27980	Hertford	NC	Perquimans	13410	143900.0	197.314050	2018
2018-04-01	399584	33974	Lehigh Acres	FL	Lee	7663	164700.0	165.645161	2018

350 rows × 10 columns

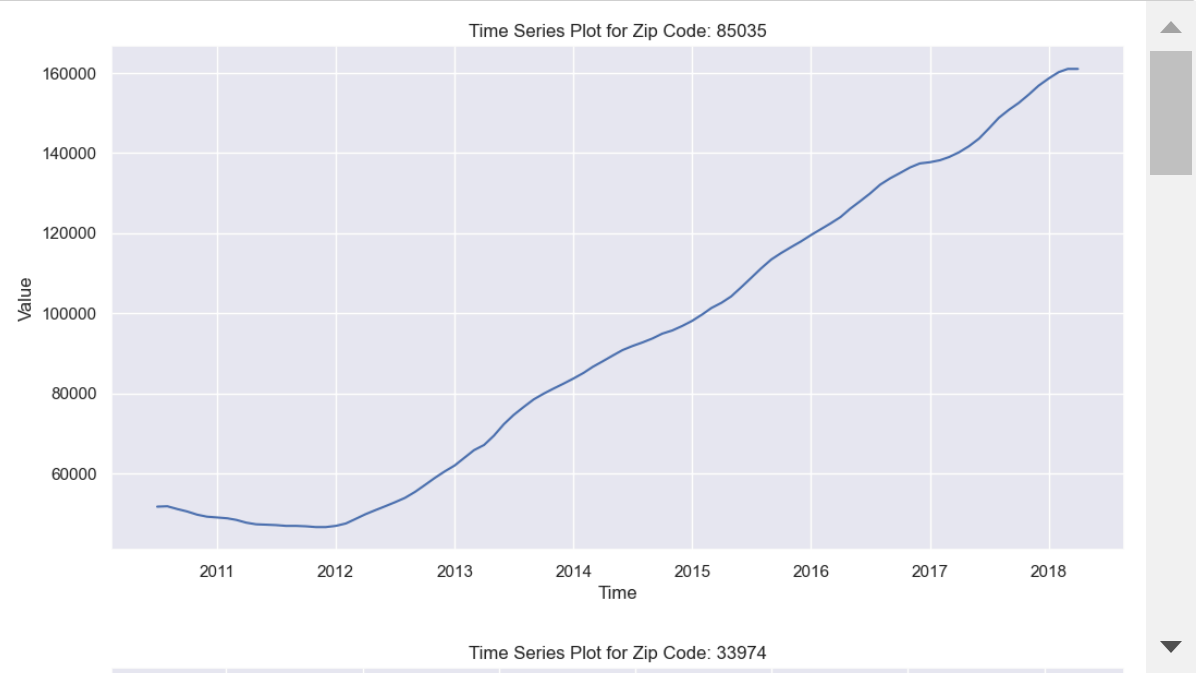
In [35]:
```
# Basic exploration
print(df4.describe())
plt.figure(figsize=(12, 6))
plt.plot(df['value'])
plt.title('Time Series Plot')
plt.xlabel('Time')
plt.ylabel('Value')
plt.show()
```

```
            RegionID       ZipCode      SizeRank          value          ROI
\
count     350.000000    350.000000    350.000000     350.000000  350.000000
mean   158206.582857  54599.422857   5785.685714  106954.857143  196.154492
std    134105.458049  25993.639240   4235.608232   50311.521997   19.312984
min     69649.000000  27980.000000    843.000000   42600.000000  165.645161
25%     70817.000000  30032.000000   1947.000000   67150.000000  191.176471
50%     94751.000000  33974.000000   7303.000000   96700.000000  197.314050
75%     94751.000000  85035.000000   7663.000000  134825.000000  211.411992
max    399584.000000  85035.000000  13410.000000  267300.000000  219.014085

              year     quarter
count   350.000000  350.000000
mean   2014.777143    2.480000
std       1.962848    1.132181
min    2010.000000    1.000000
25%    2013.000000    1.000000
50%    2015.000000    2.000000
75%    2016.000000    3.000000
max    2018.000000    4.000000
```



Change in House Prices for the top 5 Zip Codes

In [36]:
```
import matplotlib.pyplot as plt

# Get unique zip codes
zip_codes = df4['ZipCode'].unique()

# Plot each zip code in a separate figure
for zip_code in zip_codes:
    # Filter the DataFrame for each zip code
    df_zip = df4[df4['ZipCode'] == zip_code]

    # Create a new figure for each zip code
    plt.figure(figsize=(12, 6))
    plt.plot(df_zip.index, df_zip['value'])
    plt.title(f'Time Series Plot for Zip Code: {zip_code}')
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.grid(True)

    # Show the plot
    plt.show()
```



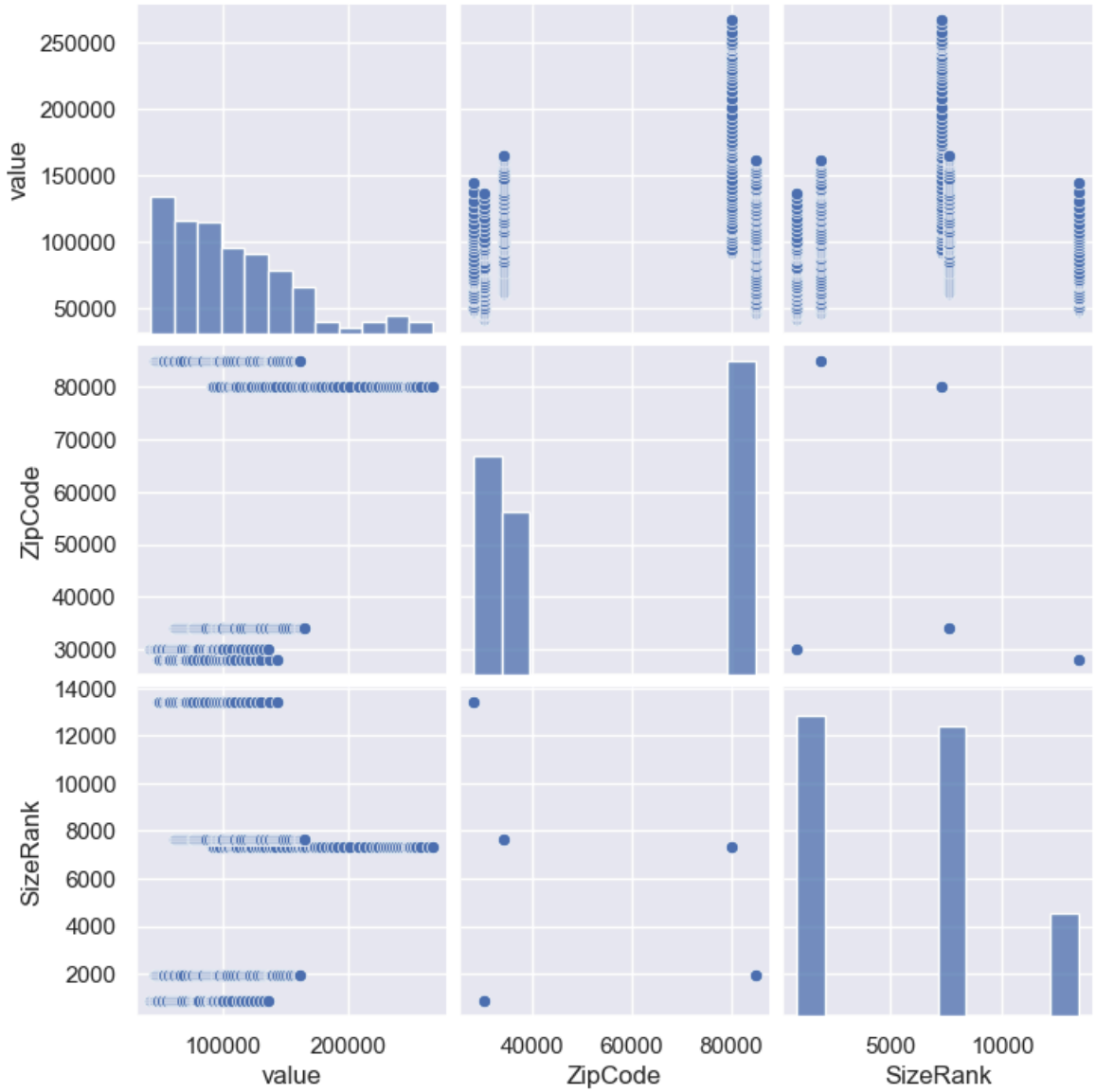
There is a notable positive trend/slope where the home value is increasing over time for each of the individual zip codes

In [37]:
```
# Univariate Analysis
print("Univariate Analysis:")
print(df4.describe(include='all'))

# Plot histograms for numeric variables
numeric_columns = df4.select_dtypes(include='number').columns
for col in numeric_columns:
    plt.figure(figsize=(10, 6))
    sns.histplot(df4[col], kde=True)
    plt.title(f'Histogram of {col}')
    plt.show()
```

```
Univariate Analysis:
             RegionID       ZipCode     City State CountyName      SizeRa
nk  \
count      350.000000    350.000000      350   350        350    350.0000
00
unique            NaN           NaN        5     5          5           N
aN
top               NaN           NaN  Phoenix    AZ   Maricopa           N
aN
freq              NaN           NaN       94    94         94           N
aN
mean    158206.582857  54599.422857      NaN   NaN        NaN   5785.6857
14
std     134105.458049  25993.639240      NaN   NaN        NaN   4235.6082
32
min      69649.000000  27980.000000      NaN   NaN        NaN    843.0000
00
25%      70817.000000  30032.000000      NaN   NaN        NaN   1947.0000
00
```

- Most of the properties range between $50,000 and 100,000$. From there there is a steep decline in the number of properties as value increases.
- Most properties registered an ROI of between 210% - 220%

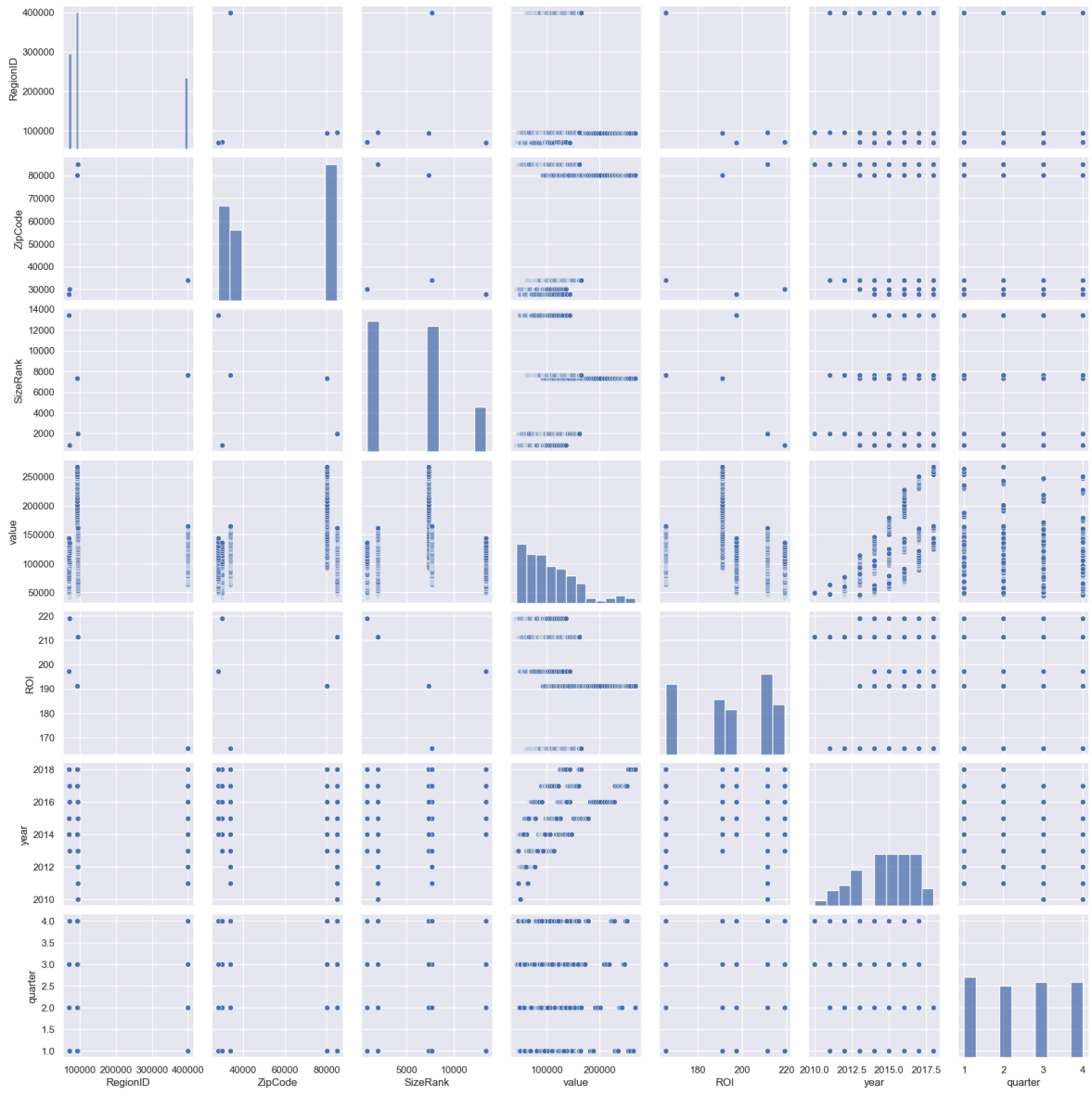
In [38]:
```
# Bivariate Analysis: Scatter plots and Correlation
# Ensure 'ZipCode' is numeric if necessary
df4['ZipCode'] = pd.to_numeric(df4['ZipCode'], errors='coerce')

# Create a copy of the DataFrame to avoid SettingWithCopyWarning
df4_cleaned = df4.copy()

# Replace any infinite values in the DataFrame to NaN
df4_cleaned.replace([np.inf, -np.inf], np.nan, inplace=True)

# List of numeric columns
numeric_columns = ['value', 'ZipCode', 'SizeRank']

# Drop any rows with NaN values in the numeric columns to avoid issues in pai
df4_cleaned = df4_cleaned[numeric_columns].dropna()

# Plot scatter plots for numeric variable pairs
sns.pairplot(df4_cleaned)
plt.show()

# Calculate correlation matrix
correlation_matrix = df4_cleaned.corr()

# Display the correlation matrix
print(correlation_matrix)
```



	value	ZipCode	SizeRank
value	1.000000	0.324033	0.109149
ZipCode	0.324033	1.000000	-0.395074
SizeRank	0.109149	-0.395074	1.000000

- Zip codes in the 80000 range have the higher valued properties while tge 40000 range mostly has the lower valued properties

In [39]:
```
# Multivariate Analysis: Pair Plot
sns.pairplot(df4)
plt.show()

# Multiple Linear Regression to predict 'value' based on other numeric variab
import statsmodels.api as sm

# Define the dependent and independent variables
X = df4[['ZipCode', 'RegionID', 'SizeRank']]
y = df4['value']

# Add a constant to the independent variables matrix
X = sm.add_constant(X)

# Fit the regression model
model = sm.OLS(y, X).fit()

# Print the regression model summary
model_summary = model.summary()
```



E. FEATURE ENGINEERING

1. Annual Features and Visualization per zipcode
2. Quarterly features and Visualization per zipcode
3. Monthly Features and Visualization per zipcode
4. Date-time Features
5. Window Features
6. Expanding Features

Visualization of Each Zipcode to identify different features of our data

In [40]:
```
df4.head()
```

Out[40]:

time	RegionID	ZipCode	City	State	CountyName	SizeRank	value	ROI	year	q
2010-07-01	94751	85035	Phoenix	AZ	Maricopa	1947	51700.0	211.411992	2010	
2010-08-01	94751	85035	Phoenix	AZ	Maricopa	1947	51800.0	211.411992	2010	
2010-09-01	94751	85035	Phoenix	AZ	Maricopa	1947	51100.0	211.411992	2010	
2010-10-01	94751	85035	Phoenix	AZ	Maricopa	1947	50500.0	211.411992	2010	
2010-11-01	94751	85035	Phoenix	AZ	Maricopa	1947	49700.0	211.411992	2010	

1. Annual/ Yearly Features

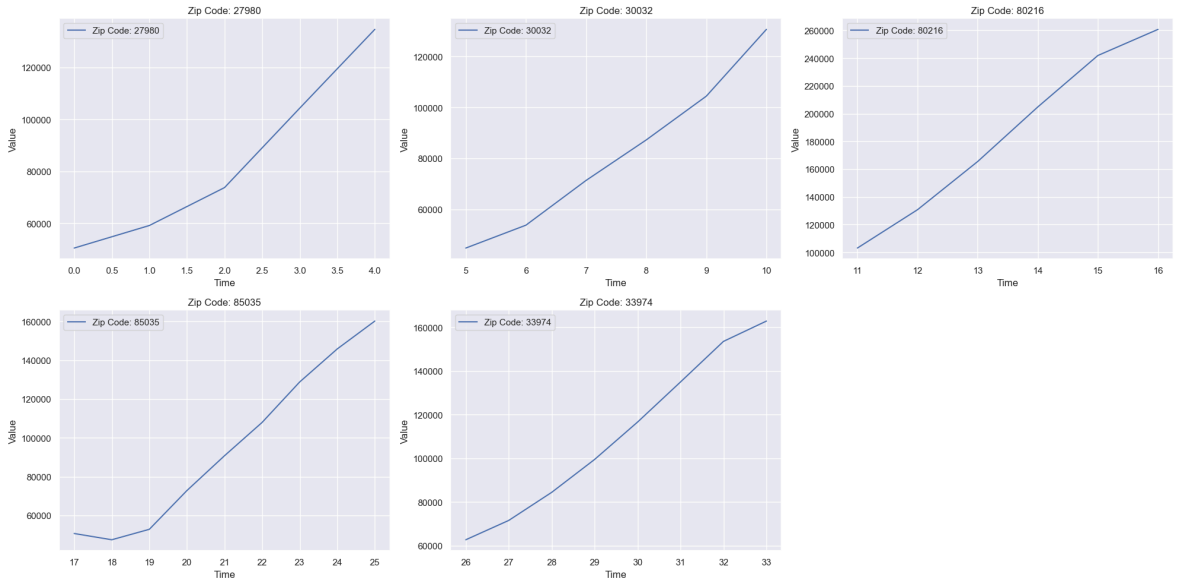
In [41]:
```
df_yearly.head()
```

Out[41]:

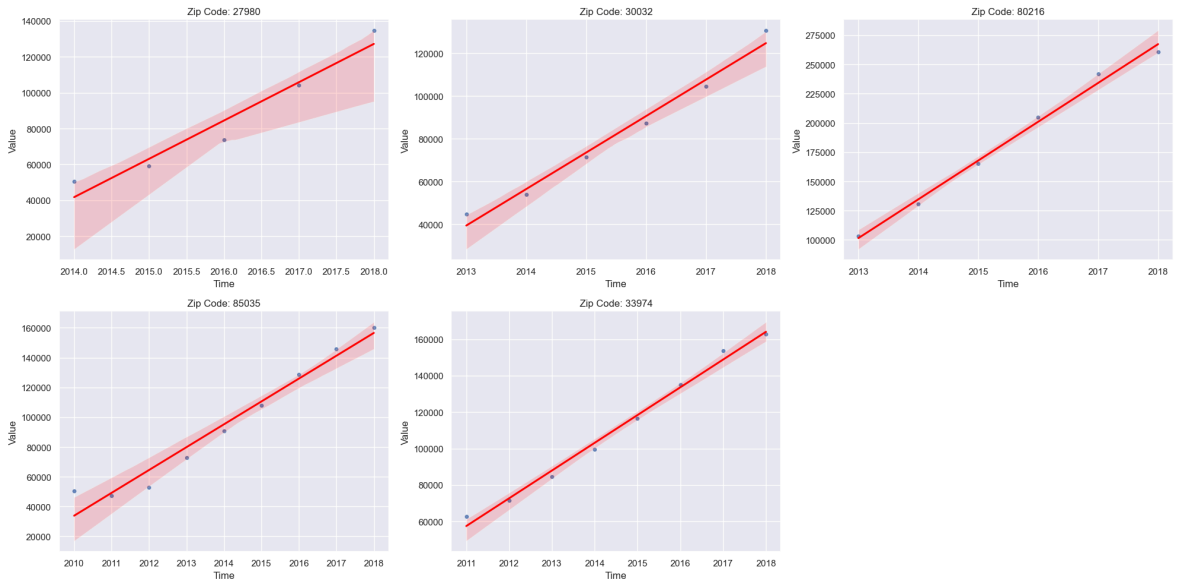
	RegionID	ZipCode	City	State	CountyName	SizeRank	year	value	ROI
0	69649	27980	Hertford	NC	Perquimans	13410	2014	50516.666667	197.31405
1	69649	27980	Hertford	NC	Perquimans	13410	2015	59183.333333	197.31405
2	69649	27980	Hertford	NC	Perquimans	13410	2016	73758.333333	197.31405
3	69649	27980	Hertford	NC	Perquimans	13410	2017	104283.333333	197.31405
4	69649	27980	Hertford	NC	Perquimans	13410	2018	134600.000000	197.31405

In [42]:
```
# Plotting the time series for different selected zipcodes
# Get unique zip codes
zip_codes = df_yearly['ZipCode'].unique()

# Create subplots
plt.figure(figsize=(12, 8))

# Plot each zip code in a separate line plot
for zip_code in zip_codes:
    df_zip = df_yearly[df_yearly['ZipCode'] == zip_code]
    plt.plot(df_zip['year'], df_zip['value'], label=f'Zip Code: {zip_code}')

# Customize the plot
plt.title('Time Series of Value by Zip Code')
plt.xlabel('Year')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
```

In [43]:
```
# Plotting the time series for different selected zipcodes
# Get unique zip codes
zip_codes = df_yearly['ZipCode'].unique()

# Define the number of rows and columns for subplots
num_zip_codes = len(zip_codes)
num_cols = 3  # Number of columns for subplots
num_rows = (num_zip_codes + num_cols - 1) // num_cols  # Calculate the number

# Create subplots
fig, axs = plt.subplots(num_rows, num_cols, figsize=(20, 5 * num_rows))
axs = axs.flatten()  # Flatten the array of axes for easy iteration

# Plot each zip code in a separate subplot
for i, zip_code in enumerate(zip_codes):
    # Filter the DataFrame for each zip code
    df_zip = df_yearly[df_yearly['ZipCode'] == zip_code]

    # Plotting the 'value' column
    axs[i].plot(df_zip.index, df_zip['value'], label=f'Zip Code: {zip_code}')
    axs[i].set_title(f'Zip Code: {zip_code}')
    axs[i].set_xlabel('Time')
    axs[i].set_ylabel('Value')
    axs[i].legend()

# Remove any empty subplots
for j in range(i + 1, num_rows * num_cols):
    fig.delaxes(axs[j])

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```



In [44]:
```
#Visualization for different ZipCodes with their trendline

# Get unique zip codes
zip_codes = df_yearly['ZipCode'].unique()

# Define the number of rows and columns for subplots
num_zip_codes = len(zip_codes)
num_cols = 3  # Number of columns for subplots
num_rows = (num_zip_codes + num_cols - 1) // num_cols  # Calculate the number

# Create subplots
fig, axs = plt.subplots(num_rows, num_cols, figsize=(20, 5 * num_rows))
axs = axs.flatten()  # Flatten the array of axes for easy iteration

# Plot each zip code in a separate subplot
for i, zip_code in enumerate(zip_codes):
    # Filter the DataFrame for each zip code
    df_zip = df_yearly[df_yearly['ZipCode'] == zip_code]

    # Convert DatetimeIndex to numeric timestamps (seconds since epoch)
    x = df_zip.index.astype(int) // 10**9  # Convert nanoseconds to seconds

    # Scatter plot with trend line using seaborn
    sns.regplot(x=df_zip ['year'], y=df_zip['value'], ax=axs[i], scatter_kws=

    axs[i].set_title(f'Zip Code: {zip_code}')
    axs[i].set_xlabel('Time')
    axs[i].set_ylabel('Value')

# Remove any empty subplots
for j in range(i + 1, num_rows * num_cols):
    fig.delaxes(axs[j])

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```



The home value is increasing over time, hence a positive trendlinein our data for the individual zipcodes.

2. Quarterly Features

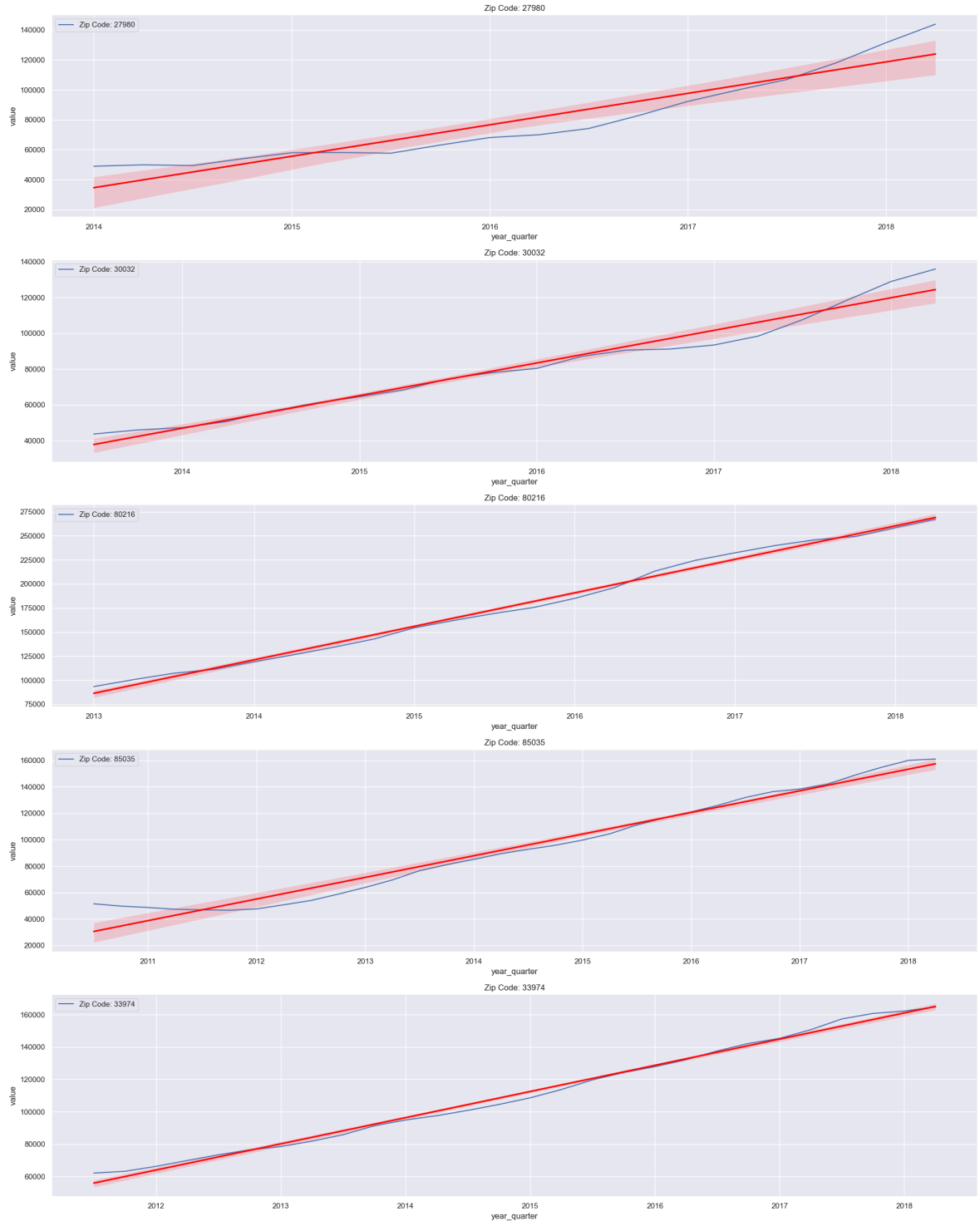
Quarterly visualization of our data to determine seasonality and trend

In [45]:
```
df_quarterly.head()
```

Out[45]:

	RegionID	ZipCode	City	State	CountyName	SizeRank	year	quarter	value	
0	69649	27980	Hertford	NC	Perquimans	13410	2014	1	48900.000000	197
1	69649	27980	Hertford	NC	Perquimans	13410	2014	2	49866.666667	197
2	69649	27980	Hertford	NC	Perquimans	13410	2014	3	49366.666667	197
3	69649	27980	Hertford	NC	Perquimans	13410	2014	4	53933.333333	197
4	69649	27980	Hertford	NC	Perquimans	13410	2015	1	57933.333333	197

In [46]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Ensure the quarter is numeric
df_quarterly['quarter'] = df_quarterly['quarter'].astype(int)

# Create a new column combining year and quarter as a numeric value
df_quarterly['year_quarter'] = df_quarterly['year'] + (df_quarterly['quarter'

# Get unique zip codes
zip_codes = df_quarterly['ZipCode'].unique()

# Define the number of rows and columns for subplots
num_zip_codes = len(zip_codes)
num_cols = 1 # Number of columns for subplots
num_rows = (num_zip_codes + num_cols - 1) // num_cols  # Calculate the number

# Create subplots
fig, axs = plt.subplots(num_rows, num_cols, figsize=(20, 5 * num_rows))
axs = axs.flatten()  # Flatten the array of axes for easy iteration

# Plot each zip code in a separate subplot
for i, zip_code in enumerate(zip_codes):
    # Filter the DataFrame for each zip code
    df_zip = df_quarterly[df_quarterly['ZipCode'] == zip_code]

    # Plotting the 'value' column
    axs[i].plot(df_zip['year_quarter'], df_zip['value'], label=f'Zip Code: {z
    axs[i].set_title(f'Zip Code: {zip_code}')
    axs[i].set_xlabel('Year and Quarter')
    axs[i].set_ylabel('Value')
    axs[i].legend()

    # Adding a trend line (linear regression) using seaborn
    sns.regplot(data=df_zip, x='year_quarter', y='value', ax=axs[i], scatter=

# Remove any empty subplots
for j in range(i + 1, num_rows * num_cols):
    fig.delaxes(axs[j])

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```

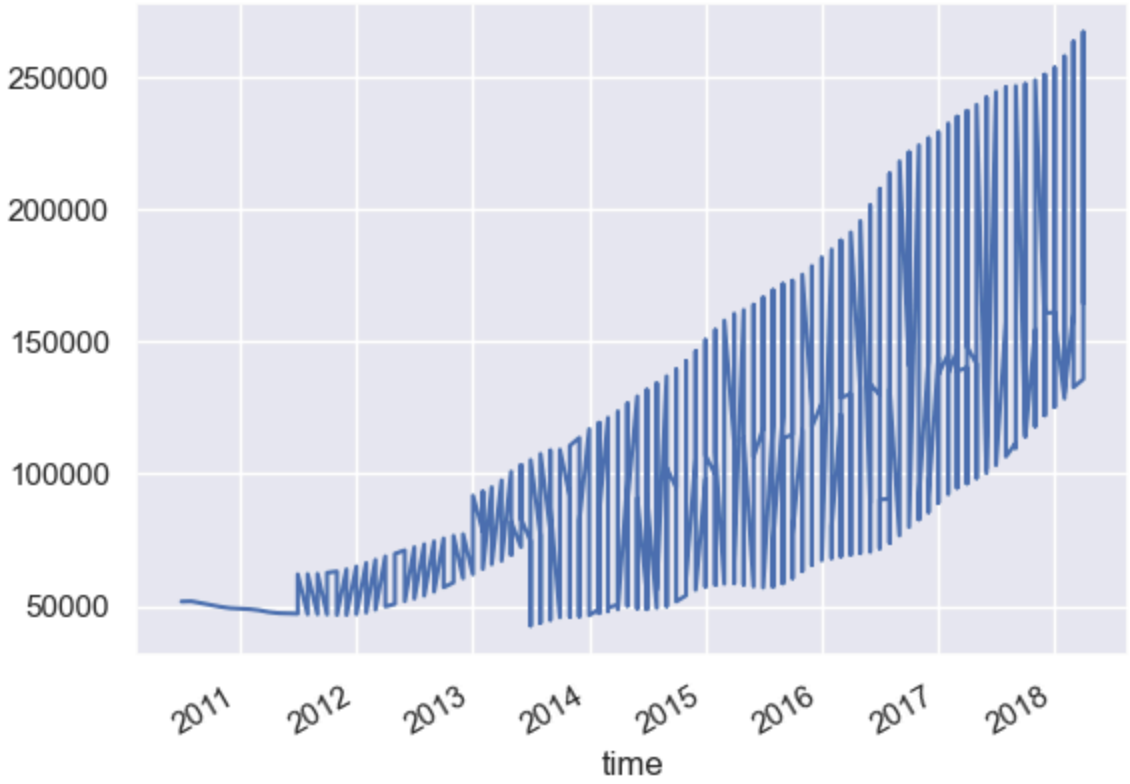


3. Monthly Features

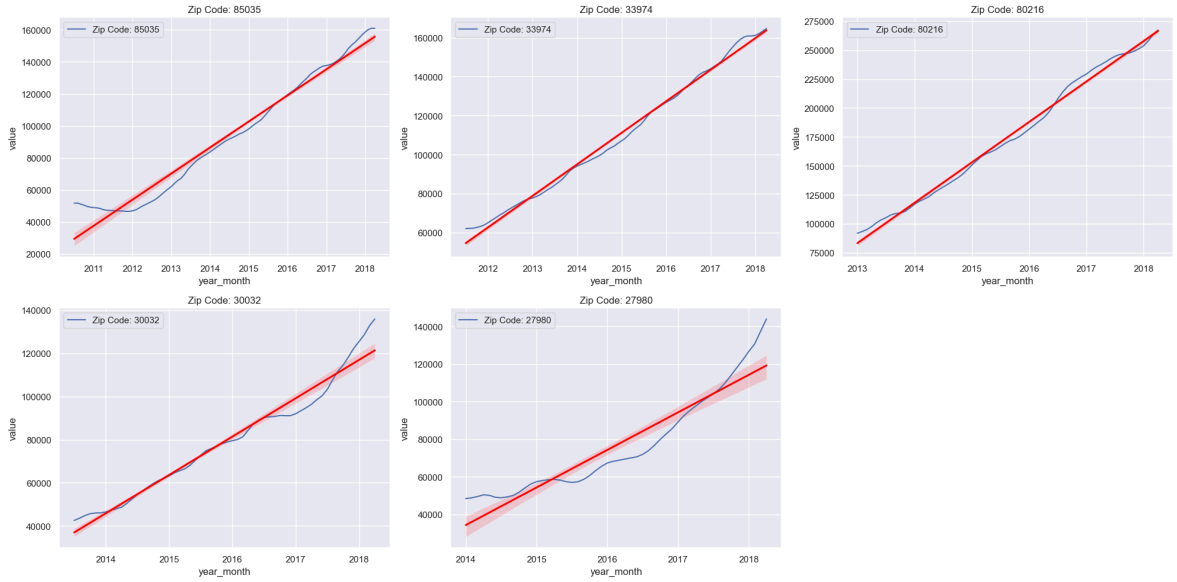
Monthly data Visualization for each zipcode

In [47]:
```
df_monthly['value'].plot()
```

Out[47]: <Axes: xlabel='time'>



In [48]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Convert index to PeriodIndex to handle periods (months) as numeric values
df_monthly.index = df_monthly.index.to_period('M')

# Create a new column combining year and month as a numeric value
df_monthly['year_month'] = df_monthly.index.year + (df_monthly.index.month -

# Get unique zip codes
zip_codes = df_monthly['ZipCode'].unique()

# Define the number of rows and columns for subplots
num_zip_codes = len(zip_codes)
num_cols = 3  # Number of columns for subplots
num_rows = (num_zip_codes + num_cols - 1) // num_cols  # Calculate the number

# Create subplots
fig, axs = plt.subplots(num_rows, num_cols, figsize=(20, 5 * num_rows))
axs = axs.flatten()  # Flatten the array of axes for easy iteration

# Plot each zip code in a separate subplot
for i, zip_code in enumerate(zip_codes):
    # Filter the DataFrame for each zip code
    df_zip = df_monthly[df_monthly['ZipCode'] == zip_code]

    # Plotting the 'value' column against 'year_month'
    axs[i].plot(df_zip['year_month'], df_zip['value'], label=f'Zip Code: {zip_
    axs[i].set_title(f'Zip Code: {zip_code}')
    axs[i].set_xlabel('Year and Month')
    axs[i].set_ylabel('Value')
    axs[i].legend()

    # Adding a trend line (linear regression) using seaborn
    sns.regplot(data=df_zip, x='year_month', y='value', ax=axs[i], scatter=Fa

# Remove any empty subplots
for j in range(i + 1, num_rows * num_cols):
    fig.delaxes(axs[j])

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```



4. Date Time Features

In [49]:
```
df4.head()
```

Out[49]:

time	RegionID	ZipCode	City	State	CountyName	SizeRank	value	ROI	year	q
2010-07	94751	85035	Phoenix	AZ	Maricopa	1947	51700.0	211.411992	2010	
2010-08	94751	85035	Phoenix	AZ	Maricopa	1947	51800.0	211.411992	2010	
2010-09	94751	85035	Phoenix	AZ	Maricopa	1947	51100.0	211.411992	2010	
2010-10	94751	85035	Phoenix	AZ	Maricopa	1947	50500.0	211.411992	2010	
2010-11	94751	85035	Phoenix	AZ	Maricopa	1947	49700.0	211.411992	2010	

In [50]:
```
features = df4.copy()
df4.index.name = 'time'
```

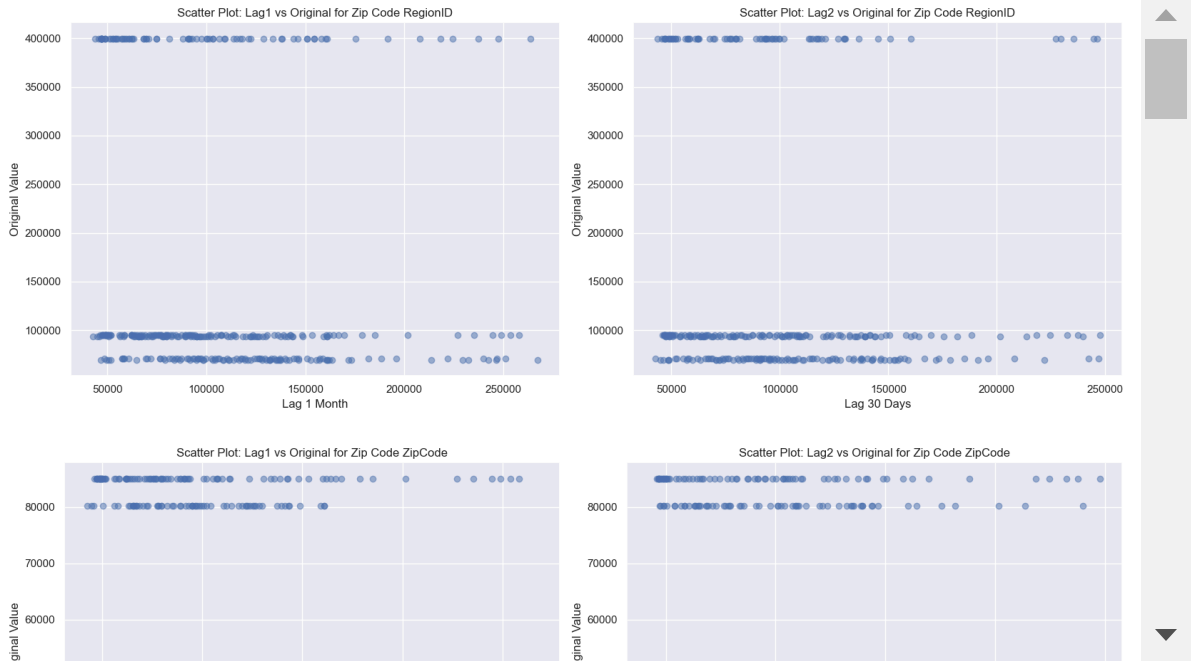
5.Lag Features

In [51]:
```
# Create lag features (e.g., previous month value)
features['lag1'] = df4['value'].shift(1)
features['lag2'] = df4['value'].shift(30)
features.head(12)
```

Out[51]:

time	RegionID	ZipCode	City	State	CountyName	SizeRank	value	ROI	year	q
2010-07	94751	85035	Phoenix	AZ	Maricopa	1947	51700.0	211.411992	2010	
2010-08	94751	85035	Phoenix	AZ	Maricopa	1947	51800.0	211.411992	2010	
2010-09	94751	85035	Phoenix	AZ	Maricopa	1947	51100.0	211.411992	2010	
2010-10	94751	85035	Phoenix	AZ	Maricopa	1947	50500.0	211.411992	2010	
2010-11	94751	85035	Phoenix	AZ	Maricopa	1947	49700.0	211.411992	2010	
2010-12	94751	85035	Phoenix	AZ	Maricopa	1947	49200.0	211.411992	2010	
2011-01	94751	85035	Phoenix	AZ	Maricopa	1947	49000.0	211.411992	2011	
2011-02	94751	85035	Phoenix	AZ	Maricopa	1947	48800.0	211.411992	2011	
2011-03	94751	85035	Phoenix	AZ	Maricopa	1947	48400.0	211.411992	2011	
2011-04	94751	85035	Phoenix	AZ	Maricopa	1947	47700.0	211.411992	2011	
2011-05	94751	85035	Phoenix	AZ	Maricopa	1947	47300.0	211.411992	2011	
2011-06	94751	85035	Phoenix	AZ	Maricopa	1947	47200.0	211.411992	2011	

In [52]:
```
import matplotlib.pyplot as plt

# Loop through each zip code and create scatter plots
for zip_code in df4.columns:
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

    # Scatter plot for lag1 vs original
    ax1.scatter(features['lag1'], df4[zip_code].values, alpha=0.5)
    ax1.set_title(f'Scatter Plot: Lag1 vs Original for Zip Code {zip_code}')
    ax1.set_xlabel('Lag 1 Month')
    ax1.set_ylabel('Original Value')

    # Scatter plot for lag2 vs original
    ax2.scatter(features['lag2'], df4[zip_code].values, alpha=0.5)
    ax2.set_title(f'Scatter Plot: Lag2 vs Original for Zip Code {zip_code}')
    ax2.set_xlabel('Lag 30 Days')
    ax2.set_ylabel('Original Value')

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



6. Window Features

Through the moving average we will be able to smoothen out series and also use the mean in foecasting

In [53]:
```
# To identify the monthy home value mean
features['Roll_mean'] = df_yearly['value'].rolling(window = 1).mean()
features.head(15)
```

Out[53]:

time	RegionID	ZipCode	City	State	CountyName	SizeRank	value	ROI	year	c
2010-07	94751	85035	Phoenix	AZ	Maricopa	1947	51700.0	211.411992	2010	
2010-08	94751	85035	Phoenix	AZ	Maricopa	1947	51800.0	211.411992	2010	
2010-09	94751	85035	Phoenix	AZ	Maricopa	1947	51100.0	211.411992	2010	
2010-10	94751	85035	Phoenix	AZ	Maricopa	1947	50500.0	211.411992	2010	
2010-11	94751	85035	Phoenix	AZ	Maricopa	1947	49700.0	211.411992	2010	
2010-12	94751	85035	Phoenix	AZ	Maricopa	1947	49200.0	211.411992	2010	
2011-01	94751	85035	Phoenix	AZ	Maricopa	1947	49000.0	211.411992	2011	
2011-02	94751	85035	Phoenix	AZ	Maricopa	1947	48800.0	211.411992	2011	
2011-03	94751	85035	Phoenix	AZ	Maricopa	1947	48400.0	211.411992	2011	
2011-04	94751	85035	Phoenix	AZ	Maricopa	1947	47700.0	211.411992	2011	
2011-05	94751	85035	Phoenix	AZ	Maricopa	1947	47300.0	211.411992	2011	
2011-06	94751	85035	Phoenix	AZ	Maricopa	1947	47200.0	211.411992	2011	
2011-07	94751	85035	Phoenix	AZ	Maricopa	1947	47100.0	211.411992	2011	
2011-07	399584	33974	Lehigh Acres	FL	Lee	7663	62000.0	165.645161	2011	
2011-08	94751	85035	Phoenix	AZ	Maricopa	1947	46900.0	211.411992	2011	

In [54]:
```
# Calculate rolling mean per ZipCode
df_yearly['Roll_mean'] = df_yearly.groupby('ZipCode')['value'].rolling(window

# Display the first 15 rows
features = df_yearly.head(15)
features
```

Out[54]:

	RegionID	ZipCode	City	State	CountyName	SizeRank	year	value	R
0	69649	27980	Hertford	NC	Perquimans	13410	2014	50516.666667	197.3140
1	69649	27980	Hertford	NC	Perquimans	13410	2015	59183.333333	197.3140
2	69649	27980	Hertford	NC	Perquimans	13410	2016	73758.333333	197.3140
3	69649	27980	Hertford	NC	Perquimans	13410	2017	104283.333333	197.3140
4	69649	27980	Hertford	NC	Perquimans	13410	2018	134600.000000	197.3140
5	70817	30032	Candler-Mcafee	GA	Dekalb	843	2013	44800.000000	219.0140
6	70817	30032	Candler-Mcafee	GA	Dekalb	843	2014	53783.333333	219.0140
7	70817	30032	Candler-Mcafee	GA	Dekalb	843	2015	71350.000000	219.0140
8	70817	30032	Candler-Mcafee	GA	Dekalb	843	2016	87241.666667	219.0140
9	70817	30032	Candler-Mcafee	GA	Dekalb	843	2017	104441.666667	219.0140
10	70817	30032	Candler-Mcafee	GA	Dekalb	843	2018	130675.000000	219.0140
11	93292	80216	Denver	CO	Denver	7303	2013	103116.666667	191.1764
12	93292	80216	Denver	CO	Denver	7303	2014	130758.333333	191.1764
13	93292	80216	Denver	CO	Denver	7303	2015	165466.666667	191.1764
14	93292	80216	Denver	CO	Denver	7303	2016	204800.000000	191.1764

In [55]:
```
# To identify the maximum sales value for a rolling window of three
features.loc[:,'Roll_max']= features['value'].rolling(window =3).max()
features.head(10)
```

Out[55]:

	RegionID	ZipCode	City	State	CountyName	SizeRank	year	value	RO
0	69649	27980	Hertford	NC	Perquimans	13410	2014	50516.666667	197.31405(
1	69649	27980	Hertford	NC	Perquimans	13410	2015	59183.333333	197.31405(
2	69649	27980	Hertford	NC	Perquimans	13410	2016	73758.333333	197.31405(
3	69649	27980	Hertford	NC	Perquimans	13410	2017	104283.333333	197.31405(
4	69649	27980	Hertford	NC	Perquimans	13410	2018	134600.000000	197.31405(
5	70817	30032	Candler-Mcafee	GA	Dekalb	843	2013	44800.000000	219.014085
6	70817	30032	Candler-Mcafee	GA	Dekalb	843	2014	53783.333333	219.014085
7	70817	30032	Candler-Mcafee	GA	Dekalb	843	2015	71350.000000	219.014085
8	70817	30032	Candler-Mcafee	GA	Dekalb	843	2016	87241.666667	219.014085
9	70817	30032	Candler-Mcafee	GA	Dekalb	843	2017	104441.666667	219.014085

7. Expanding Features

We will be considering the mean, max am]nd min fromm the start of the data

In [56]:
```
# Calculate expanding statistics per ZipCode
features.loc[:,'Expand_mean']= features['value'].expanding().mean()
features.loc[:,'Expand_max']= features['value'].expanding().max()
features.loc[:,'Expand_min']= features['value'].expanding().min()
features.head(10)
```

Out[56]:

	RegionID	ZipCode	City	State	CountyName	SizeRank	year	value	RO
0	69649	27980	Hertford	NC	Perquimans	13410	2014	50516.666667	197.31405(
1	69649	27980	Hertford	NC	Perquimans	13410	2015	59183.333333	197.31405(
2	69649	27980	Hertford	NC	Perquimans	13410	2016	73758.333333	197.31405(
3	69649	27980	Hertford	NC	Perquimans	13410	2017	104283.333333	197.31405(
4	69649	27980	Hertford	NC	Perquimans	13410	2018	134600.000000	197.31405(
5	70817	30032	Candler-Mcafee	GA	Dekalb	843	2013	44800.000000	219.014085
6	70817	30032	Candler-Mcafee	GA	Dekalb	843	2014	53783.333333	219.014085
7	70817	30032	Candler-Mcafee	GA	Dekalb	843	2015	71350.000000	219.014085
8	70817	30032	Candler-Mcafee	GA	Dekalb	843	2016	87241.666667	219.014085
9	70817	30032	Candler-Mcafee	GA	Dekalb	843	2017	104441.666667	219.014085

In [57]:
```
#Month on month growth rate
features.loc[:, 'mom_growth_rate'] = features.groupby('ZipCode')['value'].pct_
features
```

Out[57]:

	RegionID	ZipCode	City	State	CountyName	SizeRank	year	value	R(
0	69649	27980	Hertford	NC	Perquimans	13410	2014	50516.666667	197.3140
1	69649	27980	Hertford	NC	Perquimans	13410	2015	59183.333333	197.3140
2	69649	27980	Hertford	NC	Perquimans	13410	2016	73758.333333	197.3140
3	69649	27980	Hertford	NC	Perquimans	13410	2017	104283.333333	197.3140
4	69649	27980	Hertford	NC	Perquimans	13410	2018	134600.000000	197.3140
5	70817	30032	Candler-Mcafee	GA	Dekalb	843	2013	44800.000000	219.0140
6	70817	30032	Candler-Mcafee	GA	Dekalb	843	2014	53783.333333	219.0140
7	70817	30032	Candler-Mcafee	GA	Dekalb	843	2015	71350.000000	219.0140
8	70817	30032	Candler-Mcafee	GA	Dekalb	843	2016	87241.666667	219.0140
9	70817	30032	Candler-Mcafee	GA	Dekalb	843	2017	104441.666667	219.0140
10	70817	30032	Candler-Mcafee	GA	Dekalb	843	2018	130675.000000	219.0140
11	93292	80216	Denver	CO	Denver	7303	2013	103116.666667	191.1764
12	93292	80216	Denver	CO	Denver	7303	2014	130758.333333	191.1764
13	93292	80216	Denver	CO	Denver	7303	2015	165466.666667	191.1764
14	93292	80216	Denver	CO	Denver	7303	2016	204800.000000	191.1764

In [58]:
```
# Calculate year-over-year growth rate
features.loc[:,'yoy_growth_rate'] = features.groupby('ZipCode')['value'].pct_
features
```

Out[58]:

	RegionID	ZipCode	City	State	CountyName	SizeRank	year	value	R(
0	69649	27980	Hertford	NC	Perquimans	13410	2014	50516.666667	197.3140
1	69649	27980	Hertford	NC	Perquimans	13410	2015	59183.333333	197.3140
2	69649	27980	Hertford	NC	Perquimans	13410	2016	73758.333333	197.3140
3	69649	27980	Hertford	NC	Perquimans	13410	2017	104283.333333	197.3140
4	69649	27980	Hertford	NC	Perquimans	13410	2018	134600.000000	197.3140
5	70817	30032	Candler-Mcafee	GA	Dekalb	843	2013	44800.000000	219.0140
6	70817	30032	Candler-Mcafee	GA	Dekalb	843	2014	53783.333333	219.0140
7	70817	30032	Candler-Mcafee	GA	Dekalb	843	2015	71350.000000	219.0140
8	70817	30032	Candler-Mcafee	GA	Dekalb	843	2016	87241.666667	219.0140
9	70817	30032	Candler-Mcafee	GA	Dekalb	843	2017	104441.666667	219.0140
10	70817	30032	Candler-Mcafee	GA	Dekalb	843	2018	130675.000000	219.0140
11	93292	80216	Denver	CO	Denver	7303	2013	103116.666667	191.1764
12	93292	80216	Denver	CO	Denver	7303	2014	130758.333333	191.1764
13	93292	80216	Denver	CO	Denver	7303	2015	165466.666667	191.1764
14	93292	80216	Denver	CO	Denver	7303	2016	204800.000000	191.1764

In [59]:
```
# Calculate rolling standard deviation
rolling_std = features.groupby('ZipCode')['value'].rolling(window=12).std()

# Assign the values from rolling_std directly to df4['rolling_std_12']
features.loc[:,'rolling_std_12'] = rolling_std.values

# Display the updated DataFrame
features
```

Out[59]:

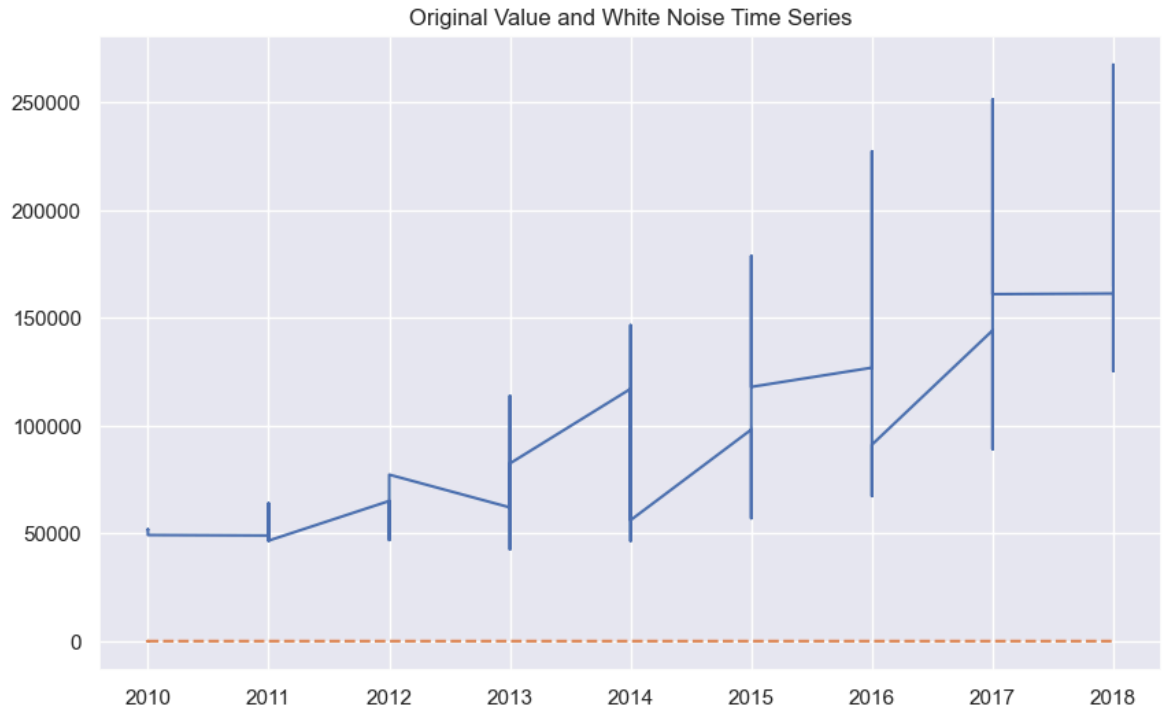
	RegionID	ZipCode	City	State	CountyName	SizeRank	year	value	R(
0	69649	27980	Hertford	NC	Perquimans	13410	2014	50516.666667	197.3140
1	69649	27980	Hertford	NC	Perquimans	13410	2015	59183.333333	197.3140
2	69649	27980	Hertford	NC	Perquimans	13410	2016	73758.333333	197.3140
3	69649	27980	Hertford	NC	Perquimans	13410	2017	104283.333333	197.3140
4	69649	27980	Hertford	NC	Perquimans	13410	2018	134600.000000	197.3140
5	70817	30032	Candler-Mcafee	GA	Dekalb	843	2013	44800.000000	219.0140
6	70817	30032	Candler-Mcafee	GA	Dekalb	843	2014	53783.333333	219.0140
7	70817	30032	Candler-Mcafee	GA	Dekalb	843	2015	71350.000000	219.0140
8	70817	30032	Candler-Mcafee	GA	Dekalb	843	2016	87241.666667	219.0140
9	70817	30032	Candler-Mcafee	GA	Dekalb	843	2017	104441.666667	219.0140
10	70817	30032	Candler-Mcafee	GA	Dekalb	843	2018	130675.000000	219.0140
11	93292	80216	Denver	CO	Denver	7303	2013	103116.666667	191.1764
12	93292	80216	Denver	CO	Denver	7303	2014	130758.333333	191.1764
13	93292	80216	Denver	CO	Denver	7303	2015	165466.666667	191.1764
14	93292	80216	Denver	CO	Denver	7303	2016	204800.000000	191.1764

In [60]:
```
features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RegionID         15 non-null     int64  
 1   ZipCode          15 non-null     int64  
 2   City             15 non-null     object 
 3   State            15 non-null     object 
 4   CountyName       15 non-null     object 
 5   SizeRank         15 non-null     int64  
 6   year             15 non-null     int32  
 7   value            15 non-null     float64
 8   ROI              15 non-null     float64
 9   Roll_mean        15 non-null     float64
 10  Roll_max         13 non-null     float64
 11  Expand_mean      15 non-null     float64
 12  Expand_max       15 non-null     float64
 13  Expand_min       15 non-null     float64
 14  mom_growth_rate  12 non-null     float64
 15  yoy_growth_rate  0 non-null      float64
 16  rolling_std_12   0 non-null      float64
dtypes: float64(10), int32(1), int64(3), object(3)
memory usage: 2.1+ KB
```

In [61]:
```
# Generate a white noise time series
np.random.seed(42)  # For reproducibility
white_noise = np.random.normal(size=len(df4))

# Add the white noise series to the df4 DataFrame
df4['white_noise'] = white_noise

# Plot the original 'value' series and the white noise series
plt.figure(figsize=(10, 6))

plt.plot(df4['year'], df4['value'], label='Original Value')
plt.plot(df4['year'], df4['white_noise'], label='White Noise', linestyle='--'

plt.title('Original Value and White Noise Time Series')
plt
```

Out[61]:
```
<module 'matplotlib.pyplot' from '/opt/anaconda3/lib/python3.11/site-package
s/matplotlib/pyplot.py'>
```



There is a clear trend indicating that there is autocorellation and therefore our model may be able to predict future values.While autocorrelation indicates predictability to some extent, several factors can influence a model's actual predictive performance. Therefore we proceed with more advanced models

F. STATIONARITY CHECKS

1. Checking for Stationarity with Augmented Dicky-Fuller Test
2. Removal of Stationarity through Differencing

Checking for Stationarity with Dickey-Fuller Test

In [62]:
```
from statsmodels.tsa.stattools import adfuller
# Perform ADF test for each zip code and print the results
# Get unique zip codes
zip_codes = df4['ZipCode'].unique()

# Loop through each zip code and perform the ADF test
for zip_code in zip_codes:
    df_zip = df4[df4['ZipCode'] == zip_code]
    result = adfuller(df_zip['value'].dropna())
    print(f'ADF test for Zip Code: {zip_code}')
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print(f'   {key}: {value}')
    if result[1] > 0.05:
        print('Fail to reject the null hypothesis. Data is not stationary.\n'
    else:
        print('Reject the null hypothesis. Data is stationary.\n')
    print('-' * 50)
```

```
ADF test for Zip Code: 85035
ADF Statistic: -0.2922970967838854
p-value: 0.9265882308703993
Critical Values:
   1%: -3.506944401824286
   5%: -2.894989819214876
   10%: -2.584614550619835
Fail to reject the null hypothesis. Data is not stationary.

--------------------------------------------------
ADF test for Zip Code: 33974
ADF Statistic: 2.0092423743781547
p-value: 0.9986848409662372
Critical Values:
   1%: -3.5194805351545413
   5%: -2.9003945086747343
   10%: -2.5874984279778395
Fail to reject the null hypothesis. Data is not stationary.

--------------------------------------------------
ADF test for Zip Code: 80216
ADF Statistic: 0.22053086887540657
p-value: 0.973389882426609
Critical Values:
   1%: -3.5443688564814813
   5%: -2.9110731481481484
   10%: -2.5931902777777776
Fail to reject the null hypothesis. Data is not stationary.

--------------------------------------------------
ADF test for Zip Code: 30032
ADF Statistic: 1.4914051710654614
p-value: 0.9974922753502519
Critical Values:
   1%: -3.560242358792829
   5%: -2.9178502070837
   10%: -2.5967964150943397
Fail to reject the null hypothesis. Data is not stationary.

--------------------------------------------------
ADF test for Zip Code: 27980
ADF Statistic: 2.75951791797398
p-value: 1.0
Critical Values:
   1%: -3.60098336718852
   5%: -2.9351348158036012
   10%: -2.6059629803688282
Fail to reject the null hypothesis. Data is not stationary.

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

The Augmented Dickey-Fuller (ADF) test is used to determine whether a unit root is present in a time series dataset.

- ADF Statistic: The ADF statistic in this case is are mostly positive with one negative.
- p-value: The p-value associated with the test is between 0.9 and 1.

- Critical Values: Critical values are thresholds that the ADF statistic must exceed for different confidence levels (1%, 5%, and 10%).

Interpretation: ADF Statistic: The more negative the ADF statistic (further from zero), the stronger the evidence against the null hypothesis (the presence of a unit root). In this case, the ADF statistic is significantly negative, indicating strong evidence against the null hypothesis.

p-value: The p-value of 0.0 (or very close to zero) suggests strong evidence against the null hypothesis. Typically, a p-value less than 0.05 (or the significance level chosen) indicates that you can reject the null hypothesis. Here, with a p-value of 0.0, we can confidently reject the null hypothesis of a unit root.

Critical Values: These values serve as benchmarks against which the ADF statistic is compared. Since the ADF statistic (-19.26) is much lower than the critical values at all significance levels (1%, 5%, and 10%), this further supports rejecting the null hypothesis.

Conclusion: Based on the ADF test results:

For all the zip codes analyzed, the ADF test results show very high p-values, well above the common significance levels (1%, 5%, and 10%). This indicates that we fail to reject the null hypothesis for all the zip codes. In simpler terms, the data for all the zip codes tested is not stationary. Non-stationarity implies that the statistical properties of the series such as mean, variance, and autocorrelation are not constant over time, which is often a challenge for time series analysis and forecasting models.

Removal of Stationarity through Differencing

In [63]:
```
# Apply differencing and show ADF test results after differencing
for zip_code in zip_codes:
    df_zip = df4[df4['ZipCode'] == zip_code]
    result = adfuller(df_zip['value'].dropna())
    adf_statistic = result[0]

    # Adjust the critical value based on your ADF test results
    if adf_statistic > -3.52:  # Adjust this threshold as per your results
        # Perform differencing on train_data for this zip code
        df4.loc[df4['ZipCode'] == zip_code, 'value'] = df4.loc[df4['ZipCode']
        # Perform ADF test after differencing
        df_zip_diff = df4[df4['ZipCode'] == zip_code].dropna()  # Remove NaNs
        result_diff = adfuller(df_zip_diff['value'])

        print(f"Differencing applied to Zip Code {zip_code}")
        print('New ADF Statistic:', result_diff[0])
        print('New p-value:', result_diff[1])
        print('New Critical Values:')
        for key, value in result_diff[4].items():
            print(f'   {key}: {value}')
        print('-' * 50)
    else:
        print(f"Zip Code {zip_code} is already stationary (ADF Statistic {adf
```

```
Differencing applied to Zip Code 85035
New ADF Statistic: -2.305322575524256
New p-value: 0.17025708106464033
New Critical Values:
   1%: -3.506944401824286
   5%: -2.894989819214876
   10%: -2.584614550619835
--------------------------------------------------
Differencing applied to Zip Code 33974
New ADF Statistic: -4.509761282276268
New p-value: 0.00018880628565507945
New Critical Values:
   1%: -3.518281134660583
   5%: -2.899878185191432
   10%: -2.5872229937594873
--------------------------------------------------
Differencing applied to Zip Code 80216
New ADF Statistic: -2.750709077285009
New p-value: 0.06566685447917704
New Critical Values:
   1%: -3.5443688564814813
   5%: -2.9110731481481484
   10%: -2.5931902777777776
--------------------------------------------------
Differencing applied to Zip Code 30032
New ADF Statistic: -2.4735930809440236
New p-value: 0.1220323187327027
New Critical Values:
   1%: -3.560242358792829
   5%: -2.9178502070837
   10%: -2.5967964150943397
--------------------------------------------------
Differencing applied to Zip Code 27980
New ADF Statistic: 2.217691828352101
New p-value: 0.9988967491937786
New Critical Values:
   1%: -3.60098336718852
   5%: -2.9351348158036012
   10%: -2.6059629803688282
--------------------------------------------------
```

In [64]:
```
df4
```

Out[64]:

time	RegionID	ZipCode	City	State	CountyName	SizeRank	value	ROI	year	q
2010-07	94751	85035	Phoenix	AZ	Maricopa	1947	NaN	211.411992	2010	
2010-08	94751	85035	Phoenix	AZ	Maricopa	1947	100.0	211.411992	2010	
2010-09	94751	85035	Phoenix	AZ	Maricopa	1947	-700.0	211.411992	2010	
2010-10	94751	85035	Phoenix	AZ	Maricopa	1947	-600.0	211.411992	2010	
2010-11	94751	85035	Phoenix	AZ	Maricopa	1947	-800.0	211.411992	2010	
...	...	...	...	...	...	...	...	...	...	
2018-04	70817	30032	Candler-Mcafee	GA	Dekalb	843	3100.0	219.014085	2018	
2018-04	94751	85035	Phoenix	AZ	Maricopa	1947	0.0	211.411992	2018	
2018-04	93292	80216	Denver	CO	Denver	7303	3600.0	191.176471	2018	
2018-04	69649	27980	Hertford	NC	Perquimans	13410	6600.0	197.314050	2018	
2018-04	399584	33974	Lehigh Acres	FL	Lee	7663	1300.0	165.645161	2018	

350 rows × 12 columns

G. SEASONALITY CHECKS

1. Seasonality Check
2. Seasonality Elimination

In [65]:
```
df4.head()
```

Out[65]:

time	RegionID	ZipCode	City	State	CountyName	SizeRank	value	ROI	year	qu
2010-07	94751	85035	Phoenix	AZ	Maricopa	1947	NaN	211.411992	2010	
2010-08	94751	85035	Phoenix	AZ	Maricopa	1947	100.0	211.411992	2010	
2010-09	94751	85035	Phoenix	AZ	Maricopa	1947	-700.0	211.411992	2010	
2010-10	94751	85035	Phoenix	AZ	Maricopa	1947	-600.0	211.411992	2010	
2010-11	94751	85035	Phoenix	AZ	Maricopa	1947	-800.0	211.411992	2010	

In [66]:
```
# Pivot the DataFrame to have time as index and ZipCodes as columns with thei
df_pivot = df4.pivot_table(index='time', columns='ZipCode', values='value')

# Display the transformed DataFrame
df_pivot
```

Out[66]:

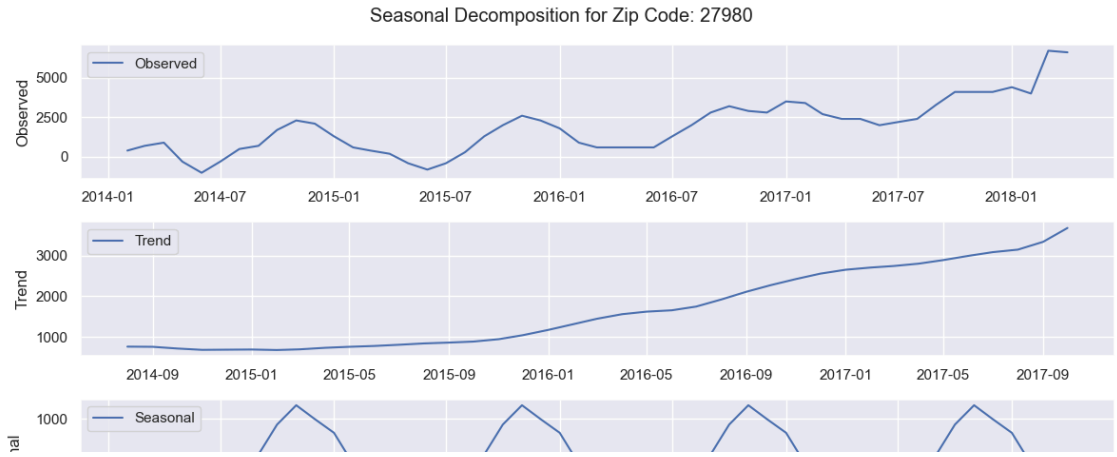
time	ZipCode	27980	30032	33974	80216	85035
2010-08		NaN	NaN	NaN	NaN	100.0
2010-09		NaN	NaN	NaN	NaN	-700.0
2010-10		NaN	NaN	NaN	NaN	-600.0
2010-11		NaN	NaN	NaN	NaN	-800.0
2010-12		NaN	NaN	NaN	NaN	-500.0
...		...	...	...	...	...
2017-12		4100.0	4000.0	0.0	2100.0	2200.0
2018-01		4400.0	3100.0	300.0	2600.0	1800.0
2018-02		4000.0	3200.0	900.0	4300.0	1600.0
2018-03		6700.0	4200.0	1300.0	5600.0	800.0
2018-04		6600.0	3100.0	1300.0	3600.0	0.0

93 rows × 5 columns

1. Check For Seasonality

In [67]:
```
import matplotlib.pyplot as plt
import numpy as np
from statsmodels.tsa.seasonal import seasonal_decompose

def check_seasonality(df_pivot, zip_code):
    df_zip = df_pivot[zip_code].dropna()  # Select the specific zip code's da

    # Perform seasonal decomposition
    decomposition = seasonal_decompose(df_zip, model='additive', period=12)

    # Check for seasonality based on the standard deviation of the seasonal c
    if np.std(decomposition.seasonal) > 0.5:
        print(f"Seasonality: {zip_code} is Seasonal")
    else:
        print(f"Seasonality: {zip_code} is not Seasonal")

    # Plot the decomposition
    fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(12, 8))

    # Convert PeriodIndex to datetime for plotting if needed
    if isinstance(decomposition.observed.index, pd.PeriodIndex):
        observed_index = decomposition.observed.index.to_timestamp()
    else:
        observed_index = decomposition.observed.index

    ax1.plot(observed_index, decomposition.observed, label='Observed')
    ax1.set_ylabel('Observed')
    ax1.legend(loc='upper left')

    ax2.plot(observed_index, decomposition.trend, label='Trend')
    ax2.set_ylabel('Trend')
    ax2.legend(loc='upper left')

    ax3.plot(observed_index, decomposition.seasonal, label='Seasonal')
    ax3.set_ylabel('Seasonal')
    ax3.legend(loc='upper left')

    ax4.plot(observed_index, decomposition.resid, label='Residual')
    ax4.set_ylabel('Residual')
    ax4.legend(loc='upper left')

    fig.suptitle(f'Seasonal Decomposition for Zip Code: {zip_code}')
    plt.tight_layout()
    plt.show()

# Loop through each zip code and check for seasonality and plot decomposition
for zip_code in df_pivot.columns:
    check_seasonality(df_pivot, zip_code)
```

```
Seasonality: 27980 is Seasonal
```



Eliminating Trend and Seasonality using Decomposing the Time Series

The purpose is to identify trends and seasonality detection. The residual component obtaine from decomposition represents the random variations or noise in the data that are not explained by the trend or seasonal patterns. Analyzing residuals helps in assessing the goodness of fit of our model and in detecting any remaining patterns that may need further investigation. Decomposition also provides insights into the structure of the time series, which can guide the selection of appropriate models for forecasting.

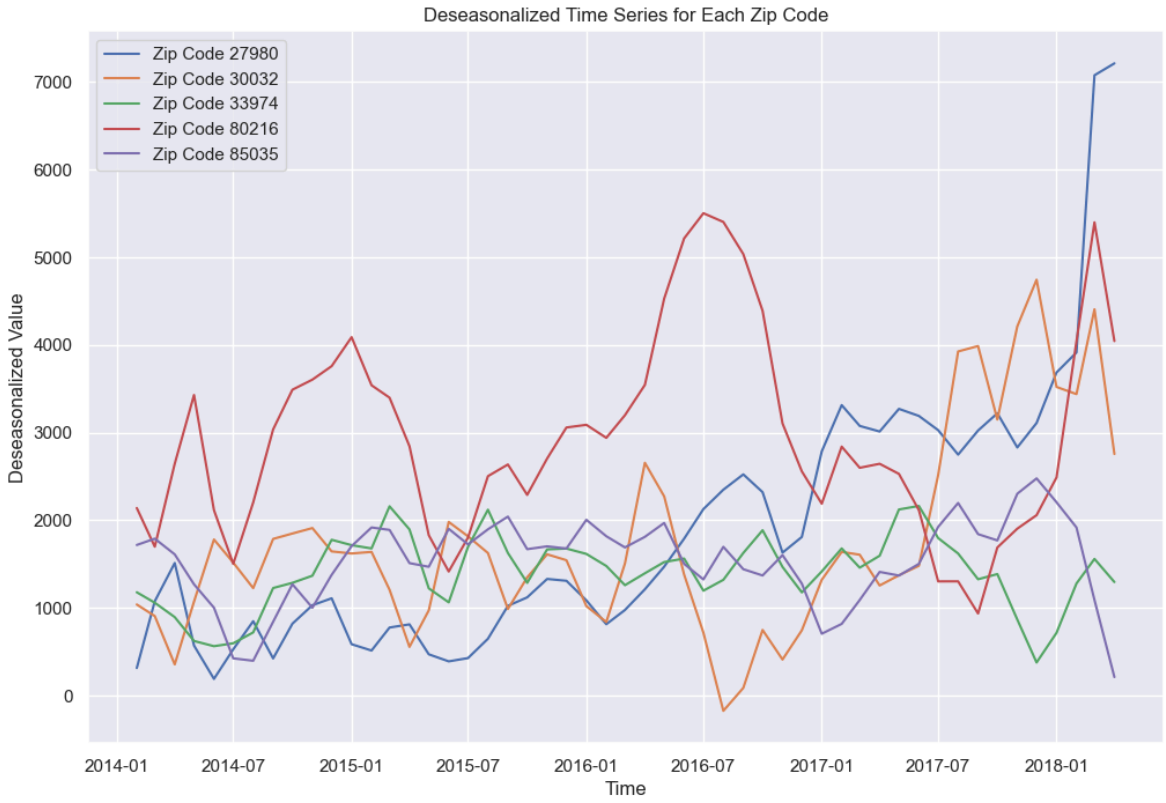
1. Original Time Series Plot: Interpretation: A basic line plot of the original time series data against time. This plot helps visualize the overall trend and any apparent seasonality or irregularities.
2. Trend Component Plot: Interpretation: This plot shows the extracted trend component after decomposition. It helps visualize the long-term direction or movement of the data, ignoring seasonal fluctuations and noise. A clear trend can indicate whether the series is increasing, decreasing, or staying relatively stable over time. OUr trund has flactuations over time.
3. Seasonal Component Plot: Interpretation: Shows the seasonal pattern extracted from the data. It reveals repetitive patterns that occur within specific time intervals, in our case it is monthly. Understanding the seasonal component is crucial for selecting seasonal models like SARIMA (Seasonal Autoregressive Integrated Moving Average).
4. Residual Plot: Interpretation: This plot displays the residual component obtained after removing the trend and seasonal components from the original data. Residuals ideally should exhibit no clear patterns or trends, indicating that the model has effectively captured the systematic variation in the data. Patterns in residuals might suggest further adjustments or insights into the data structure.

Overall Interpretation:

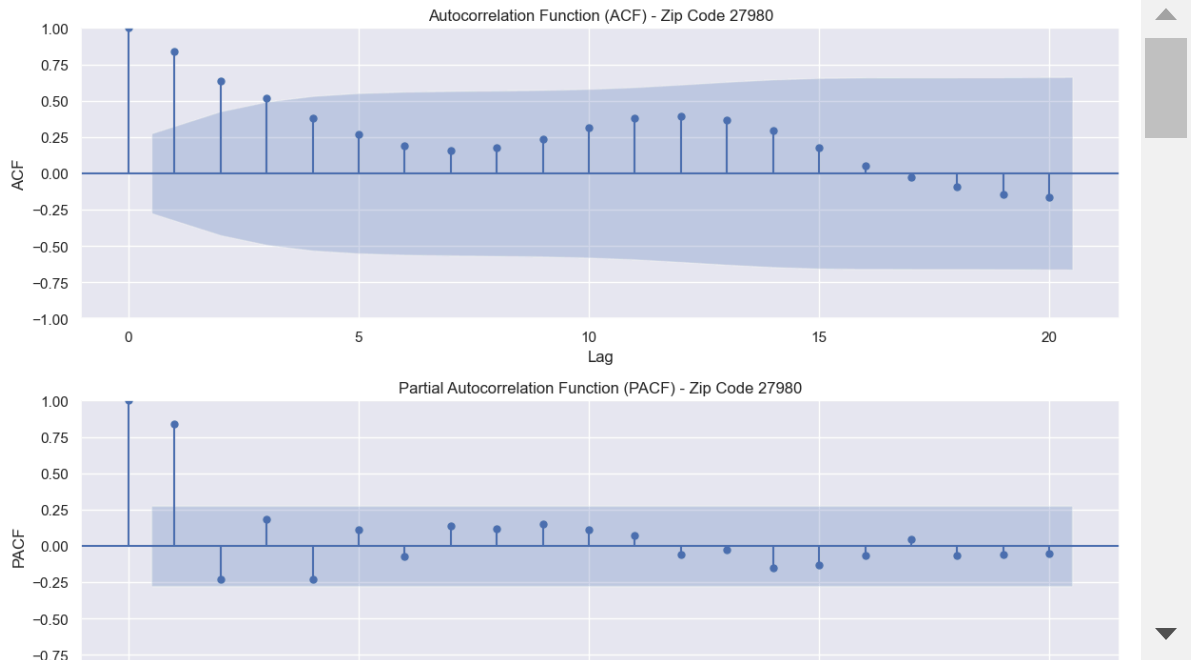
There is seasonality in all the trends for individual zipcodes Trend: Helps understand the overall direction and long-term behavior of the data. Seasonality: Identifies repetitive patterns within the data, guiding the selection of seasonal models. Residuals: Assess model adequacy by examining the randomness and lack of patterns in residuals. Model Selection: Visualizations aid in choosing appropriate time series models (e.g., ARIMA, SARIMA) that best capture the observed patterns and provide accurate forecasts.

2. Removal of Seasonlity

In [68]:
```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

# Drop missing values from the DataFrame
df_pivot.dropna(inplace=True)

# Convert index to datetime if it's in Period format
df_pivot.index = df_pivot.index.to_timestamp()

# Check for missing values
missing_values = df_pivot.isnull().sum().sum()
if missing_values > 0:
    print(f"There are still {missing_values} missing values in the DataFrame.

# Initialize a figure for plotting
plt.figure(figsize=(12, 8))

# Loop through each zip code and plot deseasonalized series with trend line
for zip_code in df_pivot.columns:
    # Perform seasonal decomposition
    decomposition = seasonal_decompose(df_pivot[zip_code], model='additive',

    # Deseasonalize by subtracting the seasonal component
    deseasonalized = df_pivot[zip_code] - decomposition.seasonal

    # Plot deseasonalized series with trend line
    plt.plot(df_pivot.index, deseasonalized, label=f'Zip Code {zip_code}')

plt.title('Deseasonalized Time Series for Each Zip Code')
plt.xlabel('Time')
plt.ylabel('Deseasonalized Value')
plt.legend()
plt.show()
```


Deseasonalized Time Series for Each Zip Code
Zip Code 27980
Zip Code 30032
Zip Code 33974
Zip Code 80216
Zip Code 85035
Deseasonalized Value
0
1000
2000
3000
4000
5000
6000
7000
2014-01
2014-07
2015-01
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01
Time

In [69]:
```
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# Assuming df_pivot is your DataFrame with time as index and zip codes as col

# Loop through each zip code and plot ACF and PACF
for zip_code in df_pivot.columns:
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))

    # Plot ACF
    plot_acf(df_pivot[zip_code].dropna(), lags=20, ax=ax1)
    ax1.set_title(f'Autocorrelation Function (ACF) - Zip Code {zip_code}')
    ax1.set_xlabel('Lag')
    ax1.set_ylabel('ACF')

    # Plot PACF
    plot_pacf(df_pivot[zip_code].dropna(), lags=20, ax=ax2)
    ax2.set_title(f'Partial Autocorrelation Function (PACF) - Zip Code {zip_c
    ax2.set_xlabel('Lag')
    ax2.set_ylabel('PACF')

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



ACF measures the correlation between a time series and its lagged values. This helps in identifying patterns such as **seasonality and trends** in the data, **Model selection** by indicating the number of lagged terms needed in autoregressive (AR) models and **Stationarity**, where ACF can indicate if the time series is stationary (constant mean and variance over time), which is crucial for many time series models. PACF measures the correlation between a time series and its lagged values, removing the effects of intermediate lags. PACF helps in identifying the order of the autoregressive (AR) term in ARIMA models. It shows the direct relationship between observations at different time steps without the influence

of other time points. Model Diagnosis: PACF complements ACF by providing additional insights into the specific effects of lagged terms on the current value, aiding in model diagnosis and selection.

Interpretation: In both plots, significant spikes beyond the confidence intervals suggest strong

Type *Markdown* and LaTeX: α^2

In [70]:
```
features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RegionID         15 non-null     int64  
 1   ZipCode          15 non-null     int64  
 2   City             15 non-null     object 
 3   State            15 non-null     object 
 4   CountyName       15 non-null     object 
 5   SizeRank         15 non-null     int64  
 6   year             15 non-null     int32  
 7   value            15 non-null     float64
 8   ROI              15 non-null     float64
 9   Roll_mean        15 non-null     float64
 10  Roll_max         13 non-null     float64
 11  Expand_mean      15 non-null     float64
 12  Expand_max       15 non-null     float64
 13  Expand_min       15 non-null     float64
 14  mom_growth_rate  12 non-null     float64
 15  yoy_growth_rate  0 non-null      float64
 16  rolling_std_12   0 non-null      float64
dtypes: float64(10), int32(1), int64(3), object(3)
memory usage: 2.1+ KB
```

G. MODELING

1. ARIMA MODEL

In [71]:
```
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

ts_85035 = df4[df4['ZipCode'] == 85035]['value'].diff().dropna()
ts_33974 = df4[df4['ZipCode'] == 33974]['value'].diff().dropna()
ts_80216 = df4[df4['ZipCode'] == 80216]['value'].diff().dropna()
ts_30032 = df4[df4['ZipCode'] == 30032]['value'].diff().dropna()
ts_27980 = df4[df4['ZipCode'] == 27980]['value'].diff().dropna()
```

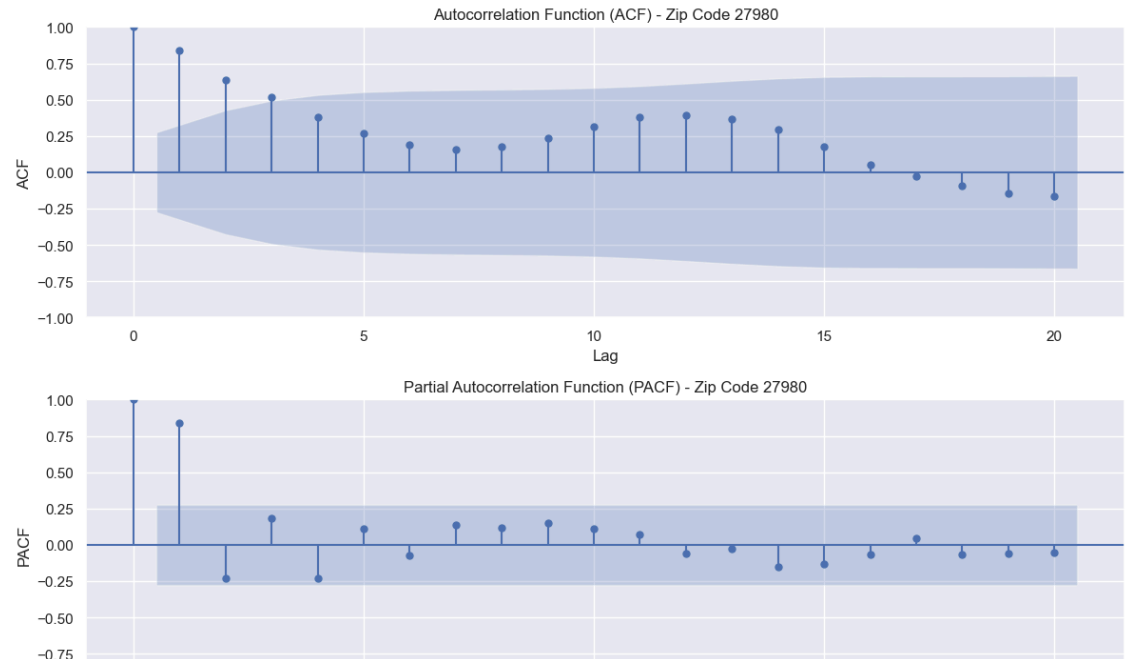
In [72]:
```
from sklearn.model_selection import TimeSeriesSplit
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Example function for cross-validation
def rolling_cv_arima(data, order, tscv):
    rmse_scores = []
    mae_scores = []

    for train_index, test_index in tscv.split(data):
        train, test = data.iloc[train_index], data.iloc[test_index]

       # Fit the ARIMA model
        model = ARIMA(train, order=order)
        model_fit = model.fit()

        # Forecast the test period
        forecast = model_fit.forecast(steps=len(test))

        # Calculate evaluation metrics
        rmse = np.sqrt(mean_squared_error(test, forecast))
        mae = mean_absolute_error(test, forecast)

        rmse_scores.append(rmse)
        mae_scores.append(mae)

    return rmse_scores, mae_scores
tscv = TimeSeriesSplit(n_splits=5)
order = (1, 1, 1)

rmse_scores, mae_scores = rolling_cv_arima(ts_85035, order, tscv)

print(f"Average RMSE: {np.mean(rmse_scores)}, Average MAE: {np.mean(mae_score
```

```
Average RMSE: 361.09702246613034, Average MAE: 287.9688104088012
```

In [73]:
```
from sklearn.model_selection import TimeSeriesSplit
import numpy as np
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error, mean_absolute_error
import pandas as pd

def rolling_cv_sarima(data, order, seasonal_order, tscv):
    rmse_scores = []
    mae_scores = []

    for train_index, test_index in tscv.split(data):
        train, test = data.iloc[train_index], data.iloc[test_index]

        # Fit the SARIMA model
        model = SARIMAX(train, order=order, seasonal_order=seasonal_order)
        model_fit = model.fit(disp=False)

        # Forecast the test period
        forecast = model_fit.forecast(steps=len(test))

        # Handle potential NaNs in test and forecast
        test_clean = test.dropna()
        # Convert forecast to a Series with the same index as test_clean
        forecast_clean = pd.Series(forecast.values, index=test.index).reindex

        # Calculate evaluation metrics if there are valid data points
        if len(test_clean) > 0:
            rmse = np.sqrt(mean_squared_error(test_clean, forecast_clean))
            mae = mean_absolute_error(test_clean, forecast_clean)

            rmse_scores.append(rmse)
            mae_scores.append(mae)

    return rmse_scores, mae_scores
# Perform time series split cross-validation
tscv = TimeSeriesSplit(n_splits=5)
order = (1, 1, 1)
seasonal_order = (1, 1, 1, 12)

rmse_scores, mae_scores = rolling_cv_sarima(ts_85035, order, seasonal_order,

print("RMSE scores:", rmse_scores)
print("MAE scores:", mae_scores)
```

```
RMSE scores: [461.88798942244284, 438.0424087838746, 240.19371051213798, 38
5.7600035278502, 505.93466750083917]
MAE scores: [379.99941537618474, 307.1628227286645, 190.32387500901459, 326.
42038095845186, 425.00003778773385]
```

In [74]:
```
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf


# Loop through each zip code and plot ACF and PACF
for zip_code in df_pivot.columns:
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))

    # Plot ACF
    plot_acf(df_pivot[zip_code].dropna(), lags=20, ax=ax1)
    ax1.set_title(f'Autocorrelation Function (ACF) - Zip Code {zip_code}')
    ax1.set_xlabel('Lag')
    ax1.set_ylabel('ACF')

    # Plot PACF
    plot_pacf(df_pivot[zip_code].dropna(), lags=20, ax=ax2)
    ax2.set_title(f'Partial Autocorrelation Function (PACF) - Zip Code {zip_c
    ax2.set_xlabel('Lag')
    ax2.set_ylabel('PACF')

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



In [75]:
```
pip install pmdarima
```

```
Requirement already satisfied: pmdarima in /opt/anaconda3/lib/python3.11/sit
e-packages (2.0.4)
Requirement already satisfied: joblib>=0.11 in /opt/anaconda3/lib/python3.1
1/site-packages (from pmdarima) (1.2.0)
Requirement already satisfied: Cython!=0.29.18,!=0.29.31,>=0.29 in /opt/anac
onda3/lib/python3.11/site-packages (from pmdarima) (3.0.10)
Requirement already satisfied: numpy>=1.21.2 in /opt/anaconda3/lib/python3.1
1/site-packages (from pmdarima) (1.26.4)
Requirement already satisfied: pandas>=0.19 in /opt/anaconda3/lib/python3.1
1/site-packages (from pmdarima) (2.1.4)
Requirement already satisfied: scikit-learn>=0.22 in /opt/anaconda3/lib/pyth
on3.11/site-packages (from pmdarima) (1.2.2)
Requirement already satisfied: scipy>=1.3.2 in /opt/anaconda3/lib/python3.1
1/site-packages (from pmdarima) (1.11.4)
Requirement already satisfied: statsmodels>=0.13.2 in /opt/anaconda3/lib/pyt
hon3.11/site-packages (from pmdarima) (0.14.0)
Requirement already satisfied: urllib3 in /opt/anaconda3/lib/python3.11/site
-packages (from pmdarima) (2.0.7)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /opt/anaconda
3/lib/python3.11/site-packages (from pmdarima) (68.2.2)
Requirement already satisfied: packaging>=17.1 in /opt/anaconda3/lib/python
3.11/site-packages (from pmdarima) (23.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/anaconda3/lib/
python3.11/site-packages (from pandas>=0.19->pmdarima) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /opt/anaconda3/lib/python3.1
1/site-packages (from pandas>=0.19->pmdarima) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /opt/anaconda3/lib/python3.
11/site-packages (from pandas>=0.19->pmdarima) (2023.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3/lib/py
thon3.11/site-packages (from scikit-learn>=0.22->pmdarima) (2.2.0)
Requirement already satisfied: patsy>=0.5.2 in /opt/anaconda3/lib/python3.1
1/site-packages (from statsmodels>=0.13.2->pmdarima) (0.5.3)
Requirement already satisfied: six in /opt/anaconda3/lib/python3.11/site-pac
kages (from patsy>=0.5.2->statsmodels>=0.13.2->pmdarima) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [76]:
```
# using auto arima to find the best p,d,q for our model
from pmdarima import auto_arima

# Use auto_arima directly instead of through the 'pm' alias
model_85035 = auto_arima(ts_85035, trace=True, error_action='ignore', suppres
print(model_85035.summary())
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=1328.937, Time=0.03 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=1342.124, Time=0.00 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=1339.606, Time=0.03 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=1333.679, Time=0.03 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=1340.124, Time=0.00 sec
 ARIMA(1,0,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.11 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=1332.476, Time=0.07 sec
 ARIMA(3,0,2)(0,0,0)[0] intercept   : AIC=1330.869, Time=0.04 sec
 ARIMA(2,0,3)(0,0,0)[0] intercept   : AIC=1330.866, Time=0.06 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=1332.459, Time=0.03 sec
 ARIMA(1,0,3)(0,0,0)[0] intercept   : AIC=1332.984, Time=0.03 sec
 ARIMA(3,0,1)(0,0,0)[0] intercept   : AIC=1333.543, Time=0.04 sec
 ARIMA(3,0,3)(0,0,0)[0] intercept   : AIC=inf, Time=0.10 sec
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=1326.936, Time=0.02 sec
 ARIMA(1,0,2)(0,0,0)[0]             : AIC=1330.719, Time=0.04 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=1330.477, Time=0.03 sec
 ARIMA(3,0,2)(0,0,0)[0]             : AIC=1328.871, Time=0.04 sec
 ARIMA(2,0,3)(0,0,0)[0]             : AIC=1328.866, Time=0.06 sec
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=1330.472, Time=0.02 sec
 ARIMA(1,0,3)(0,0,0)[0]             : AIC=1330.986, Time=0.02 sec
 ARIMA(3,0,1)(0,0,0)[0]             : AIC=1331.508, Time=0.02 sec
 ARIMA(3,0,3)(0,0,0)[0]             : AIC=inf, Time=0.09 sec

Best model:  ARIMA(2,0,2)(0,0,0)[0]
Total fit time: 0.934 seconds
                               SARIMAX Results
==============================================================================
==
Dep. Variable:                        y   No. Observations:
92
Model:               SARIMAX(2, 0, 2)   Log Likelihood                -658.4
68
Date:                Wed, 03 Jul 2024   AIC                           1326.9
36
Time:                        22:16:20   BIC                           1339.5
45
Sample:                    09-30-2010   HQIC                          1332.0
25
                         - 04-30-2018
Covariance Type:                  opg
==============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
ar.L1         -0.9939      0.192     -5.178      0.000      -1.370      -0.6
18
ar.L2         -0.6192      0.138     -4.486      0.000      -0.890      -0.3
49
ma.L1          1.4171      0.156      9.073      0.000       1.111       1.7
23
ma.L2          0.7935      0.140      5.677      0.000       0.520       1.0
68
sigma2      9.689e+04   1.57e+04      6.182      0.000    6.62e+04    1.28e+
05
```

```
==============================================================================
=======
Ljung-Box (L1) (Q):                   0.08   Jarque-Bera (JB):
0.08
Prob(Q):                              0.78   Prob(JB):
0.96
Heteroskedasticity (H):               1.37   Skew:
-0.06
Prob(H) (two-sided):                  0.38   Kurtosis:
3.08
==============================================================================
=======

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

In [77]:

```
from sklearn.model_selection import TimeSeriesSplit

# Initialize TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5)

# Assuming ts_85035 is a pandas DataFrame with a datetime index
X = ts_85035

# Generating the splits
for train_index, test_index in tscv.split(X):
    train_85035, test_85035 = X.iloc[train_index], X.iloc[test_index]

    # You can print the indices or the actual split data
    print("TRAIN:")
    print(train_85035)
    print("TEST:")
    print(test_85035)
    print("-" * 80)
```

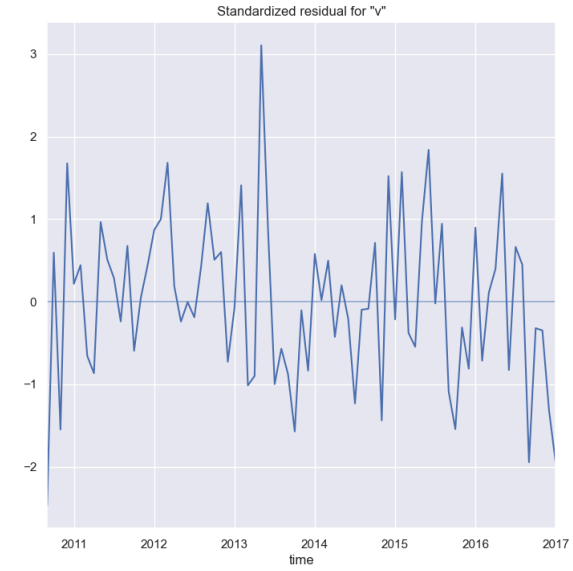
```
TRAIN:
time
2010-09   -800.0
2010-10    100.0
2010-11   -200.0
2010-12    300.0
2011-01    300.0
2011-02      0.0
2011-03   -200.0
2011-04   -300.0
2011-05    300.0
2011-06    300.0
2011-07      0.0
2011-08   -100.0
2011-09    200.0
2011-10   -100.0
2011-11   -100.0
2011-12    200.0
2012-01    300.0
Freq: M, Name: value, dtype: float64
```

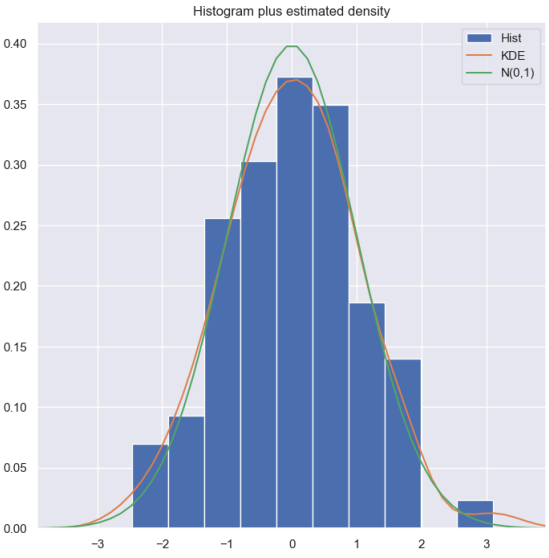
In [78]:
```
# initializing ARIMA model
ARIMAmodel = ARIMA(train_85035, order=(2, 0, 1))
```

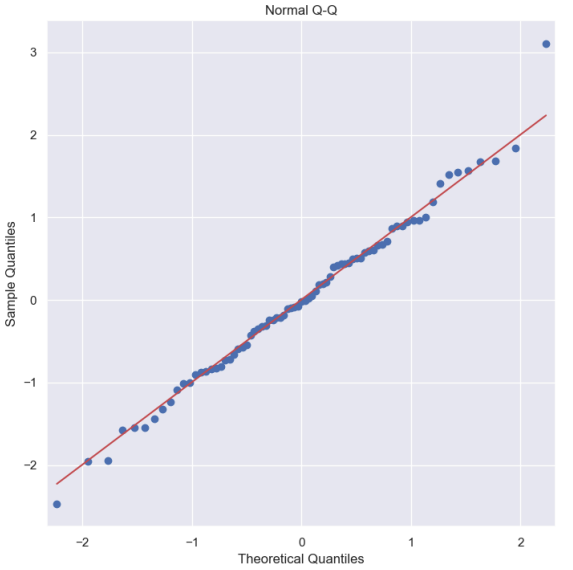
In [79]:
```
# fit the model
ARIMAmodel = ARIMAmodel.fit()
```

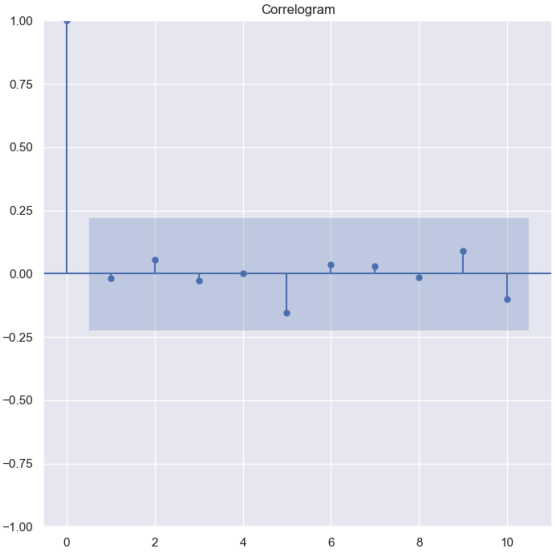
In [80]:
```
# printing the model summary
print(ARIMAmodel.summary())
ARIMAmodel.plot_diagnostics(figsize=(18,18))
plt.show()
```

```
                               SARIMAX Results
==========================================================================
==
Dep. Variable:                   value   No. Observations:
77
Model:                 ARIMA(2, 0, 1)   Log Likelihood                -548.7
18
Date:                Wed, 03 Jul 2024   AIC                           1107.4
35
Time:                        22:16:20   BIC                           1119.1
54
Sample:                    09-30-2010   HQIC                          1112.1
23
                         - 01-31-2017
Covariance Type:                 opg
==========================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
--------------------------------------------------------------------------
--
const          3.1223     35.625      0.088      0.930     -66.702      72.9
46
ar.L1         -0.3117      0.238     -1.311      0.190      -0.778       0.1
54
ar.L2         -0.2640      0.151     -1.744      0.081      -0.561       0.0
33
ma.L1          0.5413      0.267      2.026      0.043       0.018       1.0
65
sigma2      9.021e+04   1.53e+04      5.903      0.000    6.03e+04     1.2e+
05
==========================================================================
=======
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):
0.50
Prob(Q):                              0.88   Prob(JB):
0.78
Heteroskedasticity (H):               1.35   Skew:
0.15
Prob(H) (two-sided):                  0.45   Kurtosis:
3.26
==========================================================================
=======

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


Standardized residual for "v"
3
2
1
0
−1
−2
2011
2012
2013
2014
2015
2016
2017
time


Histogram plus estimated density
Hist
KDE
N(0,1)
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−3
−2
−1
0
1
2
3


Normal Q-Q
Sample Quantiles
3
2
1
0
−1
−2
−2
−1
0
1
2
Theoretical Quantiles


Correlogram
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10

In [81]:
```
#Define the range of parameters to search over
p_range = range(0, 3)
d_range = range(0, 2)
q_range = range(0, 3)

# Initialize variables to store best model and minimum AIC
best_model = None
best_aic = float("inf")

# Perform grid search
for p in p_range:
    for d in d_range:
        for q in q_range:
            # Fit ARIMA model
            model = auto_arima(ts_85035, start_p=p, d=d, start_q=q,
                               max_p=5, max_d=5, max_q=5,
                               seasonal=False,
                               suppress_warnings=True,
                               error_action='ignore',
                               stepwise=True)

            # Get model AIC (you can use BIC or other metrics as well)
            aic = model.aic()

            # Check if current model is better than the best found so far
            if aic < best_aic:
                best_aic = aic
                best_model = model
                best_params = (p, d, q)

# Print the best model summary and parameters
print("Best ARIMA model parameters (p, d, q):", best_params)
print(best_model.summary())
```

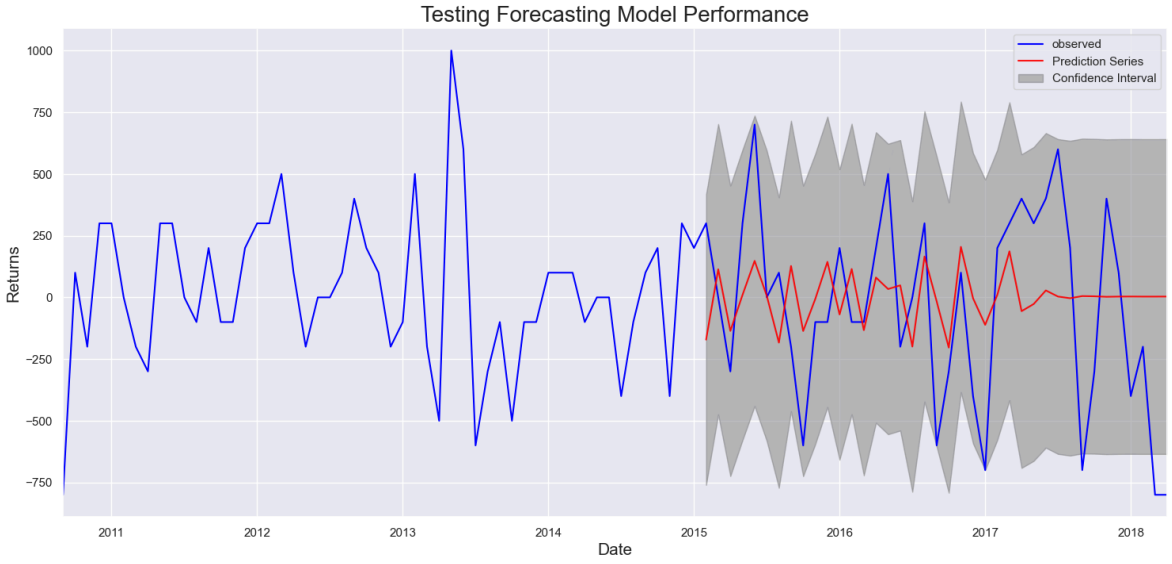
```
Best ARIMA model parameters (p, d, q): (2, 0, 2)
                               SARIMAX Results
==========================================================================
==
Dep. Variable:                        y   No. Observations:
92
Model:               SARIMAX(2, 0, 2)   Log Likelihood                -658.4
68
Date:                Wed, 03 Jul 2024   AIC                           1326.9
36
Time:                        22:16:27   BIC                           1339.5
45
Sample:                    09-30-2010   HQIC                          1332.0
25
                         - 04-30-2018
Covariance Type:                  opg
==========================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
--------------------------------------------------------------------------
--
ar.L1         -0.9939      0.192     -5.178      0.000      -1.370      -0.6
18
ar.L2         -0.6192      0.138     -4.486      0.000      -0.890      -0.3
49
ma.L1          1.4171      0.156      9.073      0.000       1.111       1.7
23
ma.L2          0.7935      0.140      5.677      0.000       0.520       1.0
68
sigma2      9.689e+04   1.57e+04      6.182      0.000    6.62e+04    1.28e+
05
==========================================================================
=======
Ljung-Box (L1) (Q):                   0.08   Jarque-Bera (JB):
0.08
Prob(Q):                              0.78   Prob(JB):
0.96
Heteroskedasticity (H):               1.37   Skew:
-0.06
Prob(H) (two-sided):                  0.38   Kurtosis:
3.08
==========================================================================
=======

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

In [82]:
```
# creating a table of the upper and lower limits
pred = ARIMAmodel.get_prediction(start=pd.to_datetime('2015-02'), end=pd.to_d
pred_conf = pred.conf_int()
pred_conf.head()
```

Out[82]:

	lower value	upper value
2015-02	-760.120155	417.216517
2015-03	-474.884033	702.452638
2015-04	-724.533161	452.803511
2015-05	-579.076739	598.259933
2015-06	-440.876254	736.460418

In [83]:
```
# Plot real vs predicted values along with confidence interval

rcParams['figure.figsize'] = 18, 8

# Plot observed values
ax = ts_85035.plot(label='observed',c="blue")

# Plot predicted values
pred.predicted_mean.plot(ax=ax, label='Prediction Series', alpha=0.9,c="red")

# Plot the range for confidence intervals
ax.fill_between(pred_conf.index,
                pred_conf.iloc[:, 0],
                pred_conf.iloc[:, 1], color='gray', alpha=0.5,label = 'Confid

# Set axes labels
ax.set_xlabel('Date',fontsize=15)
ax.set_ylabel('Returns',fontsize=15)
ax.set_title('Testing Forecasting Model Performance',fontsize=20)
plt.legend()

plt.show()
```



In [84]:
```
pred = ARIMAmodel.get_prediction(start=test_85035.index.min(), end=pd.to_date

from sklearn.metrics import mean_squared_error
rmse = mean_squared_error(test_85035, pred.predicted_mean, squared=False)
print("Root Mean Squared Error:", rmse)
```

```
Root Mean Squared Error: 458.3160991714306
```

In [85]:
```
ARIMA_MODEL = ARIMA(ts_85035,
                    order=(2,0,1),
                    enforce_stationarity=False,
                    enforce_invertibility=False)


full_output = ARIMA_MODEL.fit()

print(full_output.summary())
```

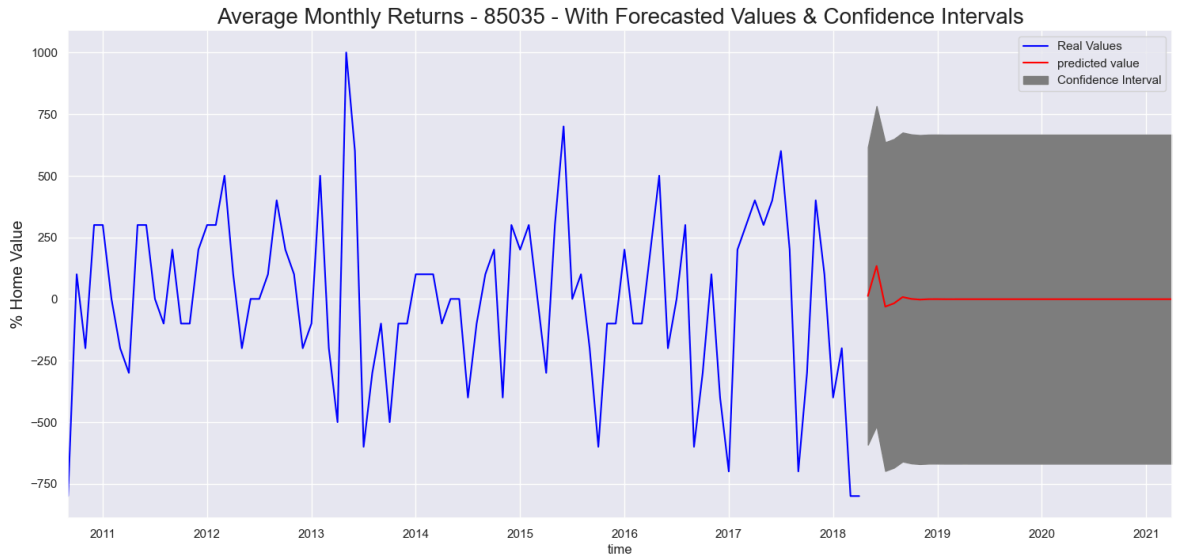
```
                                SARIMAX Results
==========================================================================
==
Dep. Variable:                   value   No. Observations:
92
Model:                 ARIMA(2, 0, 1)   Log Likelihood                -643.5
75
Date:                Wed, 03 Jul 2024   AIC                           1297.1
49
Time:                        22:16:27   BIC                           1309.6
48
Sample:                    09-30-2010   HQIC                          1302.1
90
                         - 04-30-2018
Covariance Type:                 opg
==========================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
--------------------------------------------------------------------------
--
const         -1.0864     38.263     -0.028      0.977     -76.080      73.9
07
ar.L1         -0.2076      0.212     -0.980      0.327      -0.623       0.2
07
ar.L2         -0.1721      0.125     -1.381      0.167      -0.416       0.0
72
ma.L1          0.6014      0.190      3.166      0.002       0.229       0.9
74
sigma2      9.492e+04   1.33e+04      7.135      0.000    6.88e+04    1.21e+
05
==========================================================================
=======
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
0.83
Prob(Q):                              0.97   Prob(JB):
0.66
Heteroskedasticity (H):               2.40   Skew:
0.03
Prob(H) (two-sided):                  0.02   Kurtosis:
3.47
==========================================================================
=======

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

In [86]:
```
# Getting a forecast for the next 36 months after the last recorded date on ou
forecast = full_output.get_forecast(36)
future_prediction = forecast.conf_int()
future_prediction['Price'] = forecast.predicted_mean
future_prediction.columns = ['lower','upper','prediction']
future_prediction.head()
```

Out[86]:

	lower	upper	prediction
2018-05	-591.706153	615.964176	12.129011
2018-06	-515.337968	782.596121	133.629077
2018-07	-698.138082	635.493825	-31.322128
2018-08	-684.867372	648.888813	-17.989279
2018-09	-659.852195	675.100769	7.624287

In [87]:
```
# Plotting our Forecast

fig, ax = plt.subplots()
ts_85035.plot(ax=ax,label='Real Values',c="blue")


future_prediction['prediction'].plot(ax=ax,label='predicted value',c="red")

ax.fill_between(x= future_prediction.index, y1= future_prediction['lower'],
                y2= future_prediction['upper'],color='gray',
                label='Confidence Interval')
ax.legend()
plt.ylabel("% Home Value",fontsize=15)
plt.title('Average Monthly Returns - 85035 - With Forecasted Values & Confide
plt.show()
```



Forecasting for Every ZipCode

In [93]:
```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA


# Dictionary of zip codes with their (p, d, q) values
zip_codes_params = {
    27980: (1, 1, 3),
    30032: (2, 1, 2),
    33974: (1, 1, 1),
    80216: (1, 1, 2),
    85035: (1, 1, 2)
}

# Dictionary of time series data for each zip code
zip_codes_data = {
    27980: ts_27980,
    30032: ts_30032,
    33974: ts_33974,
    80216: ts_80216,
    85035: ts_85035
}

# Looping over each zip code and fit the ARIMA model
for zip_code, (p, d, q) in zip_codes_params.items():
    ts_data = zip_codes_data[zip_code]

    if isinstance(ts_data.index, pd.PeriodIndex):
        ts_data.index = ts_data.index.to_timestamp()

    if d > 0:
        ts_data = ts_data.diff().dropna()

    train_data = ts_data.iloc[:-12]
    test_data = ts_data.iloc[-12:]

    try:
        # Fit the ARIMA model
        model = ARIMA(train_data, order=(p, d, q))
        model_fit = model.fit()

        # Print model summary to inspect details
        print(f"Model summary for Zip Code {zip_code}:")
        print(model_fit.summary())

        # Generate forecasts
        forecast_result = model_fit.get_forecast(steps=len(test_data))

        # Extract forecast mean and confidence intervals
        forecast = forecast_result.predicted_mean
        conf_int = forecast_result.conf_int()

        # Create DataFrame to hold the forecast and the confidence intervals
        forecast_df = pd.DataFrame(index=test_data.index)
        forecast_df['Prediction'] = forecast
        forecast_df['Lower Limit'] = conf_int.iloc[:, 0]
        forecast_df['Upper Limit'] = conf_int.iloc[:, 1]
```

```
        # Print forecast DataFrame to inspect
        print(f"Forecast for Zip Code: {zip_code}")
        print(forecast_df.head())

        # Plotting the forecast with confidence intervals
        plt.figure(figsize=(12, 6))
        plt.plot(train_data.index, train_data, label='Training Data')
        plt.plot(test_data.index, test_data, label='Actual Data')
        plt.plot(forecast_df.index, forecast_df['Prediction'], label='Predict
        plt.fill_between(forecast_df.index, forecast_df['Lower Limit'], forec
        plt.legend(loc='upper left')
        plt.title(f'ARIMA Model Forecast for Zip Code: {zip_code}')
        plt.show()

    except Exception as e:
        print(f"An error occurred for Zip Code {zip_code}: {e}")
```

```
Model summary for Zip Code 27980:
                               SARIMAX Results
==========================================================================
=====
Dep. Variable:                   value   No. Observations:
37
Model:                 ARIMA(1, 1, 3)   Log Likelihood                -28
1.517
Date:                Wed, 03 Jul 2024   AIC                            57
3.033
Time:                        22:23:03   BIC                            58
0.951
Sample:                    04-01-2014   HQIC                           57
5.797
                         - 04-01-2017
Covariance Type:                 opg
==========================================================================
=====
                 coef    std err          z      P>|z|      [0.025
0.975]
```

In [94]:
```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA

# Dictionary of zip codes with their (p, d, q) values
zip_codes_params = {
    27980: (1, 1, 3),
    30032: (2, 1, 2),
    33974: (1, 1, 1),
    80216: (1, 1, 2),
    85035: (1, 1, 2)
}

# Dictionary of time series data for each zip code
zip_codes_data = {
    27980: ts_27980,
    30032: ts_30032,
    33974: ts_33974,
    80216: ts_80216,
    85035: ts_85035
}

# Function to perform rolling origin cross-validation
def rolling_origin_validation(ts_data, p, d, q, forecast_steps=12):
    errors = []
    predictions = []

    # Perform rolling origin cross-validation
    for i in range(len(ts_data) - forecast_steps):
        train_data = ts_data.iloc[:i + forecast_steps]
        test_data = ts_data.iloc[i + forecast_steps:i + forecast_steps + fore

        try:
            # Fit the ARIMA model
            model = ARIMA(train_data, order=(p, d, q))
            model_fit = model.fit()

            # Make forecasts
            forecast_result = model_fit.forecast(steps=forecast_steps)
            forecast = forecast_result[0]

            # Calculate error (e.g., Mean Absolute Error)
            error = (forecast - test_data.values).mean()
            errors.append(error)
            predictions.append(forecast)

        except Exception as e:
            print(f"An error occurred: {e}")
            errors.append(None)
            predictions.append(None)

    return errors, predictions

# Looping over each zip code and perform cross-validation
for zip_code, (p, d, q) in zip_codes_params.items():
    ts_data = zip_codes_data[zip_code]
```

```
if isinstance(ts_data.index, pd.PeriodIndex):
    ts_data.index = ts_data.index.to_timestamp()

if d > 0:
    ts_data = ts_data.diff().dropna()

try:
    # Perform cross-validation
    errors, predictions = rolling_origin_validation(ts_data, p, d, q)

    # Print errors or further analysis
    print(f"Cross-validation errors for Zip Code {zip_code}:")
    print(errors)

    # Optionally, you can aggregate errors or perform further analysis

except Exception as e:
    print(f"An error occurred for Zip Code {zip_code}: {e}")
```

```
Cross-validation errors for Zip Code 27980:
[156.1442999047256, -47.835079245255464, -151.68642688057412, 354.4446854049
8207, -455.3071085895331, -233.13850926849454, -463.82660007292276, -274.418
333874083, 3.5949630666546, -137.6161711807738, 584.1597571269276, 36.765994
64679165, 75.7649667911033, 578.9451752836449, -730.137538911213, 812.239121
9856833, -24.68841482379374, 122.17858107601857, 40.95276956741464, -395.823
49474831176, -394.77189346976996, 490.65132862634664, 410.1602705066002, -13
3.93787725409047, -128.54432029307893, 102.55632822028576, -284.257751559993
04, 7.828889052232716, 392.3027114741574, -53.56843108838976, 548.7730109161
454, 161.4285351737539, -44.68061226919745, 170.24495071028423, 122.61299958
141974, -302.20701695075013, 2056.0485942992536]
Cross-validation errors for Zip Code 30032:
[327.3708879959959, -792.2141816102585, 97.0648041794994, -129.596644856099
6, 397.0884756159749, 18.75927386245727, 95.46716644727235, 88.4286496543375
5, -308.68307352181864, -219.95839417381617, 477.81674369587716, 545.7918456
851935, 433.08561018520237, 298.01285146192623, -27.085042022770967, 62.2854
6231719119, 3.9916544262616562, -98.58298464471629, -434.4820978394896, 536.
6437374137926, -558.2117930263051, 608.1158135152355, 764.6368601601343, 35
5.70258547485065, 326.82775155649495, -99.80248246343172, -58.24131448635042
6, -22.79910270212569, -480.9875798821049, -608.8537938709038, -297.26234082
448656, -33.7053234174727, -195.88034283358172, 84.58230723710994, -323.2909
800961867, -440.61444354825426, 63.87513277857943, 867.5427202595719, 593.02
11101987993, 36.75451418851708, 666.1878415708976, 982.4077018454382, 1519.2
709584587146]
Cross-validation errors for Zip Code 33974:
[-102.33536672978205, -66.30783639954986, 30.183358068990362, 186.7071848076
6786, -231.93387795152023, -119.96548468447178, -7.364684622932134, 67.76871
623671883, 50.15289323461866, 110.79283153947817, -239.92995318916732, 143.5
487673207284, -79.51812890359292, -129.18155585420786, 303.6218346975692, 11
1.80305283623102, -218.59137317671335, -41.93959415160771, -152.492418602607
02, -21.376755648980218, 34.83910626492783, 64.35123691944551, -112.90893303
015373, -51.034440078429725, 34.647271537184196, 327.09610083415174, -180.10
504275182507, -109.65923182919404, 167.8407721334295, -27.91937347274724, -1
19.79245887842778, 187.9600317400187, 17.36716863413567, -56.15340344617576
6, -165.62634930859363, 95.18089058925845, 191.48229159535103, -21.099861842
041616, -157.56958289959024, 64.91904558645264, -8.121140317033905, -36.5623
4332753606, -3.1734146167268023, -92.84416969360404, -42.66907792503368, 97.
12677310334414, 45.74175176880912, -66.24347496189466, 53.168250769458105, 1
02.24384556504599, 46.05171876022985, -41.52323504758143, -83.3698089417623
2, -35.066993820003745, 16.603295526752806, -44.19124291841096, 25.744253156
28955, 159.278346942574, 4.265170121148426, -44.120014853522996, -18.4801721
4183361, -76.83802086097583, -121.88616352127974, -163.21753263428366, -4.32
0790696270706, 275.13233553248676, 428.2526853718015]
Cross-validation errors for Zip Code 80216:
[662.4295991202097, 60.97398623742288, 779.9090647226786, -1409.506335279458
4, 525.5715859635152, -313.7968973062341, 272.8224158354731, -186.0436448638
5944, 358.5513411491849, -221.1985268082491, 121.3598130430721, -279.9839154
494632, 113.43438627146217, -365.66245509377296, -48.846663658790696, 11.741
230363057033, 23.81890363496501, -49.63861705626368, -32.914201479637846, -7
9.74123236221727, 560.208746658829, -96.15865615953128, 47.04050351928807, -
94.79458205314033, 85.45327684393526, -84.86020039615141, 337.7823216437368
5, 38.13620529062079, 33.12966902173631, -188.56237113356033, 11.82167918095
603, -213.6519993053241, -18.38324278653543, -33.4610644143764, -123.5966530
7267385, 129.0862739854708, -340.4300192410057, 177.3408883098106, 228.06975
759552688, 259.040541799101, 34.86964380430764, 213.42700547098167, 48.39380
216898696, 335.130645547655, 564.1365070314687, 636.2669911493117, 837.09700
66007509, 2019.782976737486, 3206.3240035159733]
```

```
Cross-validation errors for Zip Code 85035:
[42.05756543033359, 174.87615423208362, 145.21200620600885, 32.7489506576501
6, 2.3777417742092744, -22.726455123263804, -18.83971982183986, 226.23851923
435177, 207.9921110939123, 105.17656069872699, 188.4893340600922, 110.715459
88349682, 120.2908482163394, 59.98508188768627, 84.17773474421158, -27.32408
897713079, 46.583454201273376, 62.15555615760079, -85.14771834663088, 15.287
863400030915, 373.5249189096835, -32.40674763445713, 58.6736909258619, -29.4
21535566139976, 49.41941902309227, -169.22987156061257, 236.46897433235222,
-175.2041844663123, 182.2296349615707, -132.2228451931937, 143.4983185789828
6, -89.48435127521593, 106.34807418510138, -130.69296206420435, -6.708142845
766663, 26.56724163086786, 49.167368413904455, 98.49801836714774, -105.19551
077598112, 129.4450894246269, -65.72132003485376, 144.94547174543027, -90.87
916659265119, 53.21913373458684, -16.980979964339934, 167.46584581256775, -1
22.51124147674402, 132.1860375288263, -91.67621314286778, 92.6263479855096,
-77.8647633471774, 106.5561149269904, 51.308232066132405, -9.62018250773845
4, -27.913662832723308, 36.679289293100936, 30.702333960745175, -118.4551571
2458473, 55.83382095251886, -39.854765734451384, -34.85069227516093, 89.0964
8098757225, -45.45527761350517, -70.72587419363886, -29.56320587534449, 173.
11423586354917, -6.714367433106038, 235.2146290535229, -14.52828013789261, 2
68.00790582021864, 62.53070476413565, 198.36519769023084, -147.8031440158960
7, 270.2252081710019, 127.74034487495396, 325.95908397592723, -4.56722549974
027, 460.4656987493852, -152.35960322806199]
```

In [95]:
```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Looping over each zip code and fit the ARIMA model
results = {} # Initialize a dictionary to store results
for zip_code, (p, d, q) in zip_codes_params.items():

    try:

        # Calculate and store metrics for each zipcode
        mae = mean_absolute_error(test_data, forecast)
        mse = mean_squared_error(test_data, forecast)
        rmse = np.sqrt(mse)

        results[zip_code] = {'MAE': mae, 'MSE': mse, 'RMSE': rmse}

    except Exception as e:
        print(f"An error occurred for Zip Code {zip_code}: {e}")

# Print the evaluation metrics for each zipcode
for zip_code, metrics in results.items():
    print(f"Metrics for Zip Code {zip_code}:")
    for metric_name, value in metrics.items():
        print(f"{metric_name}: {value}")
```

```
Metrics for Zip Code 27980:
MAE: 383.64083826112096
MSE: 217832.58881139953
RMSE: 466.7253890794881
Metrics for Zip Code 30032:
MAE: 383.64083826112096
MSE: 217832.58881139953
RMSE: 466.7253890794881
Metrics for Zip Code 33974:
MAE: 383.64083826112096
MSE: 217832.58881139953
RMSE: 466.7253890794881
Metrics for Zip Code 80216:
MAE: 383.64083826112096
MSE: 217832.58881139953
RMSE: 466.7253890794881
Metrics for Zip Code 85035:
MAE: 383.64083826112096
MSE: 217832.58881139953
RMSE: 466.7253890794881
```

2. SARIMA Model

In [96]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Create a list to store forecast results for each zip code
forecast_results = []

# Loop through each unique zip code
for zip_code in df4['ZipCode'].unique():
    # Filter the data for the current zip code
    df_zip = df4[df4['ZipCode'] == zip_code]

    # Group by year and calculate mean value
    df_yearly = df_zip.groupby('year').agg({'value': 'mean'})

    # Define the SARIMA model
    model = SARIMAX(df_yearly['value'], order=(1, 1, 1), seasonal_order=(0, 0

    # Fit the model
    results = model.fit(disp=False)

    # Make predictions
    df_yearly['forecast'] = results.predict(start=0, end=len(df_yearly)-1, dy

    # Store the forecast results
    forecast_results.append(df_yearly[['value', 'forecast']].reset_index())

    # Plot the original data and the forecast for the current zip code
    plt.figure(figsize=(10, 6))
    plt.plot(df_yearly.index, df_yearly['value'], label='Original')
    plt.plot(df_yearly.index, df_yearly['forecast'], label='Forecast', linest
    plt.title(f'SARIMA Model Forecast for Zip Code {zip_code}')
    plt.xlabel('Year')
    plt.ylabel('Value')
    plt.legend()
    plt.show()
```

```
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/tsa/base/tsa_mode
l.py:473: ValueWarning: An unsupported index was provided and will be ign
ored when e.g. forecasting.
  self._init_dates(dates, freq)
/opt/anaconda3/lib/python3.11/site-packages/statsmodels/tsa/base/tsa_mode
l.py:473: ValueWarning: An unsupported index was provided and will be ign
ored when e.g. forecasting.
  self._init_dates(dates, freq)
```

SARIMA Model Forecast for Zip Code 85035

Conclusion

Based on ROI, the best zipcodes with their corresponding cities and states to invest in are as follows:

- 30032 Candler Mcafee GA, ROI - 219.01%
- 85035 Phoenix AZ, ROI - 211.41%
- 27980 Hertford NC, ROI - 197.31%
- 80216 Denver CO, ROI - 191.18%
- 33974 LeHigh Acres FL, ROI - 165.65%

Recommendation

1. All the seslected zip codes(top 5) have a promising predicted value, as they are in the positive.
2. Based on the above graph, we can forecast our top five recommendations and their expected ROI after three years.
3. The investor can then decide to invest in any of the recommended zip codes.

In [ ]: