PHSAE 4 PROJECT - TEAM 3

Introduction

This is the team 3 phase 3 project notebook.Our group members include :

- Anthony Ngatia
- Jessyca Aperi
- Joy Chepchumba
- Naomi Rotich

In this notebook we shall be anaylsing the **Zillow Housing Dataset using a Time Series MOdel**

Overview

In these project we shall seek to do the following :

- Load the dataset
- Understand the dataset
- Choose our target variable.
- Prepare the dataset (Example : Cleaning the dataset , checking for multicollinearity)
- Encode our categorical variables
- Make several models
- Evaluate our models
- Use our models for prediction
- Come up with relevant findings.

1). Business Understanding

Real estate investment is a lucrative and dynamic industry that requires careful analysis and decision-making. The fictional real estate investment firm is seeking guidance on identifying the top 5 zip codes for investment opportunities. To address this question, historical data from Zillow Research is utilized.

i) Background:

Real estate investment is a lucrative and dynamic industry that requires careful analysis and decision-making. The fictional real estate investment firm is seeking guidance on identifying the top 5 zip codes for investment opportunities. To address this question, historical data from Zillow Research is utilized. The dataset contains information on various attributes, including RegionID, RegionName, City, State, Metro, SizeRank, CountyName, and value (real estate prices).

ii). Main Objective:

The main objective of this project is to identify the top 5 zip codes that offer the best investment potential in terms of real estate prices. By analyzing historical trends and patterns, the project aims to provide actionable insights to the investment firm, enabling them to make informed decisions on where to allocate their resources.

Specific Objectives:

- Analyze Historical Data: The project involves analyzing the historical data of real estate prices across different zip codes. This includes understanding the trends, patterns, and fluctuations in property values over time.
- Identify Promising Zip Codes: Using the analysis of historical data, the project aims to identify the zip codes that have shown consistent growth, stability, or potential for future appreciation. These zip codes are considered the most favorable for investment.
- Consider Location Factors: In addition to the historical performance, the project also takes into account location-specific factors such as city, state, and metro. This information helps assess the overall desirability and attractiveness of the investment opportunities.
- Evaluate Market SizeRank: The SizeRank attribute provides insights into the relative size and competitiveness of the real estate market in each zip code. This factor helps gauge the potential opportunities and risks associated with investing in a particular area.

2). Data Understanding

The dataset contains information on various attributes, including RegionID, RegionName, City, State, Metro, SizeRank, CountyName, and value (real estate prices). Our dataset is the Zillow Housing Dataset which was sourced from Zillow Research Page.

In order to understand how our dataset looks like lets get a preview of this data by loading it.

Importing the Zillow Housing Dataset

```
#Importing data libraries
import numpy as np
import pandas as pd

#importing visualisation libraries
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

#importing math libraries
from math import sqrt

#Importing modeling libraries
from statsmodels.tsa.seasonal import seasonal_decompose
from dateutil.parser import parse
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from matplotlib.pylab import rcParams
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error as MSE
```

```
#importing the dataset
df = pd.read_csv('/content/zillow_data.csv')
df.head()
```

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	1996-06	...	2017-07	2017-08	2017-09	201
0	84654	60657	Chicago	IL	Chicago	Cook	1	334200.0	335400.0	336500.0	...	1005500	1007500	1007800	100
1	90668	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	235700.0	236900.0	236700.0	...	308000	310000	312500	31
2	91982	77494	Katy	TX	Houston	Harris	3	210400.0	212200.0	212200.0	...	321000	320600	320200	32
3	84616	60614	Chicago	IL	Chicago	Cook	4	498100.0	500900.0	503100.0	...	1289800	1287700	1287400	129
4	93144	79936	El Paso	TX	El Paso	El Paso	5	77300.0	77300.0	77300.0	...	119100	119400	120000	12

5 rows × 272 columns

```
#Investigating for the shape of the dataset
df.shape
```

```
(14723, 272)
```

The dataset contains 14723 rows and 272 columns

```
#Describing the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14723 entries, 0 to 14722
Columns: 272 entries, RegionID to 2018-04
dtypes: float64(219), int64(49), object(4)
memory usage: 30.6+ MB
```

The dataset contains data types of; float, integers and strings

3). Data Preparation

Checking for missing values

```
df.isna().sum()
```

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

```
#Displaying the rows with missing values.
df[df.isnull().any(axis=1)].iloc[:-1]
```

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	1996-06	...	2017-07	2017-08	20
20	61625	10011	New York	NY	New York	New York	21	NaN	NaN	NaN	...	12137600	12112600	120
36	61796	10456	New York	NY	New York	Bronx	37	NaN	NaN	NaN	...	357900	357100	3
105	84613	60611	Chicago	IL	Chicago	Cook	106	NaN	NaN	NaN	...	1475200	1473900	14
151	69340	27410	Greensboro	NC	NaN	Guilford	152	137100.0	136600.0	136000.0	...	212900	213200	2
156	62048	11238	New York	NY	New York	Kings	157	NaN	NaN	NaN	...	2673300	2696700	27
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
14706	59046	3215	Waterville Valley	NH	Claremont	Grafton	14707	NaN	NaN	NaN	...	786000	780900	7
14707	69681	28039	East Spencer	NC	Charlotte	Rowan	14708	NaN	NaN	NaN	...	27300	26400	
14708	99401	97733	Crescent	OR	Klamath Falls	Klamath	14709	NaN	NaN	NaN	...	197700	203700	2
14710	59210	3812	Bartlett	NH	NaN	Carroll	14711	80900.0	80800.0	80800.0	...	215500	217000	2
14717	62697	12720	Bethel	NY	NaN	Sullivan	14718	62500.0	62600.0	62700.0	...	122200	122700	1

1827 rows × 272 columns

Dealing with Missing Values

We used Linear Interpolation to handle the numerical missing values

```
# Checking for missing values in each column
missing_columns = df.columns[df.isnull().any()].tolist()

print(missing_columns)
```

```
['Metro', '1996-04', '1996-05', '1996-06', '1996-07', '1996-08', '1996-09', '1996-10', '1996-11', '1996-12', '1997-01', '1997-02', '1997
```

```
# Iterate over the columns with missing values
for col in missing_columns:
    # Perform interpolation using linear method
    df[col] = df[col].interpolate(method='linear')

# Print the updated dataset with interpolated values
```

```
print(df)
```

```
       RegionID  RegionName                 City State             Metro  \
0         84654       60657              Chicago    IL           Chicago
1         90668       75070             McKinney    TX  Dallas-Fort Worth
2         91982       77494                 Katy    TX           Houston
3         84616       60614              Chicago    IL           Chicago
4         93144       79936              El Paso    TX           El Paso
...         ...         ...                  ...   ...               ...
14718     58333        1338             Ashfield    MA   Greenfield Town
14719     59107        3293            Woodstock    NH         Claremont
14720     75672       40404                Berea    KY          Richmond
14721     93733       81225  Mount Crested Butte    CO               NaN
14722     95851       89155             Mesquite    NV         Las Vegas

      CountyName  SizeRank   1996-04   1996-05   1996-06  ...  2017-07  \
0           Cook         1  334200.0  335400.0  336500.0  ...  1005500
1         Collin         2  235700.0  236900.0  236700.0  ...   308000
2         Harris         3  210400.0  212200.0  212200.0  ...   321000
3           Cook         4  498100.0  500900.0  503100.0  ...  1289800
4        El Paso         5   77300.0   77300.0   77300.0  ...   119100
...          ...       ...       ...       ...       ...  ...      ...
14718   Franklin     14719   94600.0   94300.0   94000.0  ...   216800
14719    Grafton     14720   92700.0   92500.0   92400.0  ...   202100
14720    Madison     14721   57100.0   57300.0   57500.0  ...   121800
14721   Gunnison     14722  191100.0  192400.0  193700.0  ...   662800
14722      Clark     14723  176400.0  176300.0  176100.0  ...   333800

       2017-08  2017-09  2017-10  2017-11  2017-12  2018-01  2018-02  2018-03  \
0      1007500  1007800  1009600  1013300  1018700  1024400  1030700  1033800
1       310000   312500   314100   315000   316600   318100   319600   321100
2       320600   320200   320400   320800   321200   321200   323000   326900
3      1287700  1287400  1291500  1296600  1299000  1302700  1306400  1308500
4       119400   120000   120300   120300   120300   120300   120500   121000
...        ...      ...      ...      ...      ...      ...      ...      ...
14718   217700   218600   218500   218100   216400   213100   209800   209200
14719   208400   212200   215200   214300   213100   213700   218300   222700
14720   122800   124600   126700   128800   130600   131700   132500   133000
14721   671200   682400   695600   695500   694700   706400   705300   681500
14722   336400   339700   343800   346800   348900   350400   353000   356000

       2018-04
0      1030600
1       321800
2       329900
3      1307000
4       121500
...        ...
14718   209300
14719   225800
14720   133400
14721   664400
14722   357200

[14723 rows x 272 columns]
```

```
df = df.drop('Metro', axis = 1)
```

```
#Confirming the missing values are not present
df[df.isnull().any(axis=1)].iloc[:-1]
```

RegionID	RegionName	City	State	CountyName	SizeRank	1996-04	1996-05	1996-06	1996-07	...	2017-07	2017-08	2017-09	2017-10	2017-11	2017-12	2018 0

0 rows × 271 columns

Reshaping our dataset from wide to long format

```
#Converting our dataframe time column from float to datetime format

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

```
    """

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


# Reshaping our dataset from Wide to Long Format

def melt_data(data):
    melted = pd.melt(data, id_vars=['RegionID', 'RegionName', 'City', 'State', 'SizeRank', 'CountyName'], var_name='time')
    melted['time'] = pd.to_datetime(melted['time'], infer_datetime_format=True)
    melted = melted.dropna(subset=['value'])
    return melted.groupby(['RegionID', 'RegionName', 'City', 'State', 'SizeRank', 'CountyName', 'time']).aggregate({'value': 'mean'}).reset_i
```

```
#Printing the first five rows of the long format dataset.
df1 = melt_data(df)
df1.head()
```

	RegionID	RegionName	City	State	SizeRank	CountyName	time	value
0	58196	1001	Agawam	MA	5851	Hampden	1996-04-01	113100.0
1	58196	1001	Agawam	MA	5851	Hampden	1996-05-01	112800.0
2	58196	1001	Agawam	MA	5851	Hampden	1996-06-01	112600.0
3	58196	1001	Agawam	MA	5851	Hampden	1996-07-01	112300.0
4	58196	1001	Agawam	MA	5851	Hampden	1996-08-01	112100.0

```
#Descriptive Statistics for the Long format dataset
df1.describe()
```

	RegionID	RegionName	SizeRank	value
count	3.901595e+06	3.901595e+06	3.901595e+06	3.901595e+06
mean	8.107501e+04	4.822235e+04	7.362000e+03	2.060636e+05
std	3.193304e+04	2.935833e+04	4.250165e+03	2.368017e+05
min	5.819600e+04	1.001000e+03	1.000000e+00	1.130000e+04
25%	6.717400e+04	2.210100e+04	3.681000e+03	9.770000e+04
50%	7.800700e+04	4.610600e+04	7.362000e+03	1.469000e+05
75%	9.092100e+04	7.520600e+04	1.104300e+04	2.354000e+05
max	7.538440e+05	9.990100e+04	1.472300e+04	1.931490e+07

```
# Checking for missing values in the long format dataframe
df1.isna().sum()
```

```
RegionID      0
RegionName    0
City          0
State         0
SizeRank      0
CountyName    0
time          0
value         0
dtype: int64
```

```
#checking for duplicates in the Long Format Dataframe
df1.duplicated().sum()
```

```
0
```

4). Exploratory Data Analysis

Grouping the data by months

```
#Setting the time column as the index
df1.set_index('time', inplace = True)
```

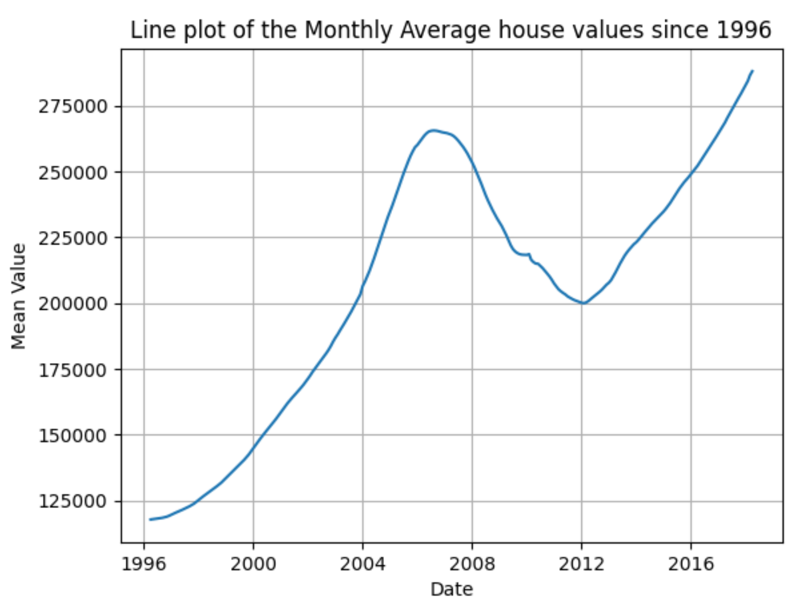
```
#Resampling using monthly buckets
monthlyvalue = df1.resample('MS')
month_mean = monthlyvalue.mean()
month_mean.tail()
```

time	RegionID	RegionName	SizeRank	value
2017-12-01	81075.010052	48222.348706	7362.0	281095.320247
2018-01-01	81075.010052	48222.348706	7362.0	282657.060382
2018-02-01	81075.010052	48222.348706	7362.0	284368.688447
2018-03-01	81075.010052	48222.348706	7362.0	286511.376757
2018-04-01	81075.010052	48222.348706	7362.0	288039.944305

Visualising the Dataframe

Grouping per Month and plottting

```
# Plotting the monthly housing average
plt.figure()
plt.plot(month_mean.index, month_mean['value'])
plt.title('Line plot of the Monthly Average house values since 1996')
plt.xlabel('Date')
plt.ylabel('Mean Value')
plt.grid(True)
plt.show()
```

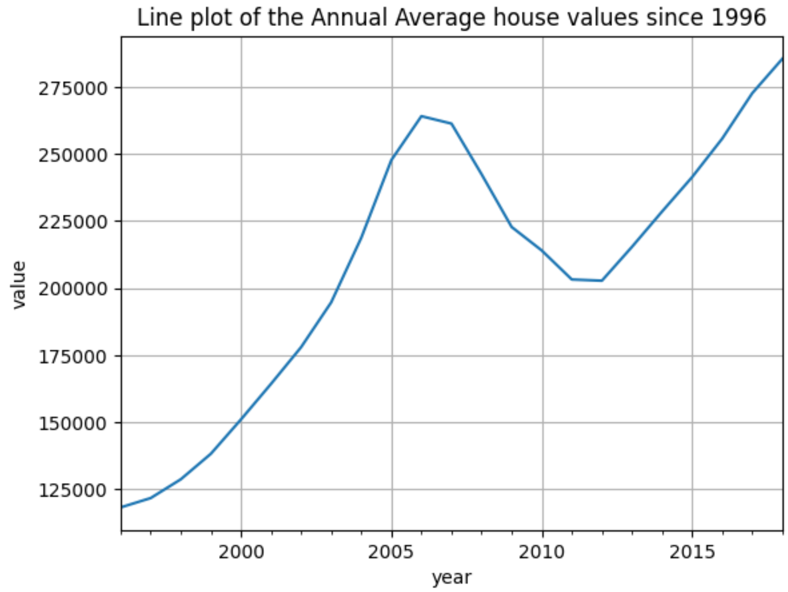


Grouping per Year and plottting

```
#Setting the year as the index
df1['Year'] = df1.index.year
```

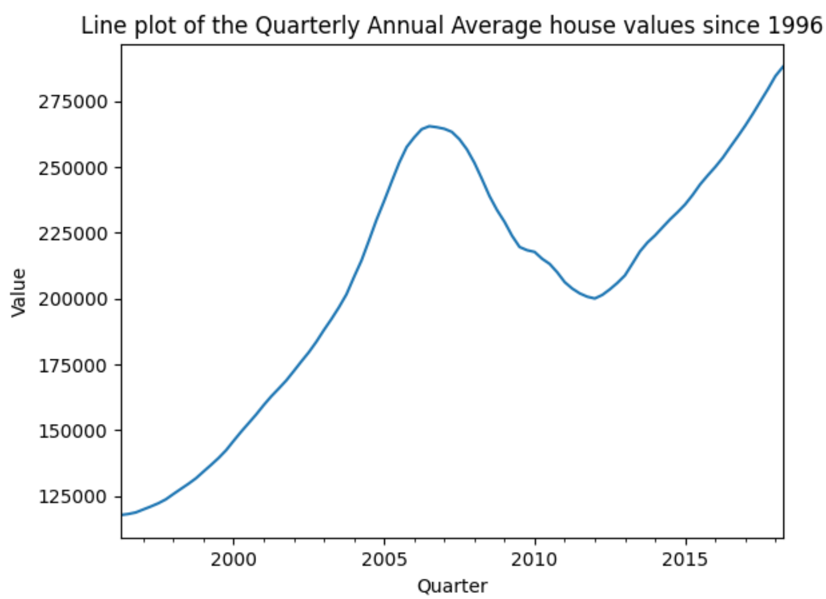
```
#Plotting the Housing annual average
df1_yearly = df1['value'].resample('A').mean()
df1_yearly.plot();
plt.title('Line plot of the Annual Average house values since 1996')
plt.xlabel('year')
```

```
plt.ylabel('value')
plt.grid()
```



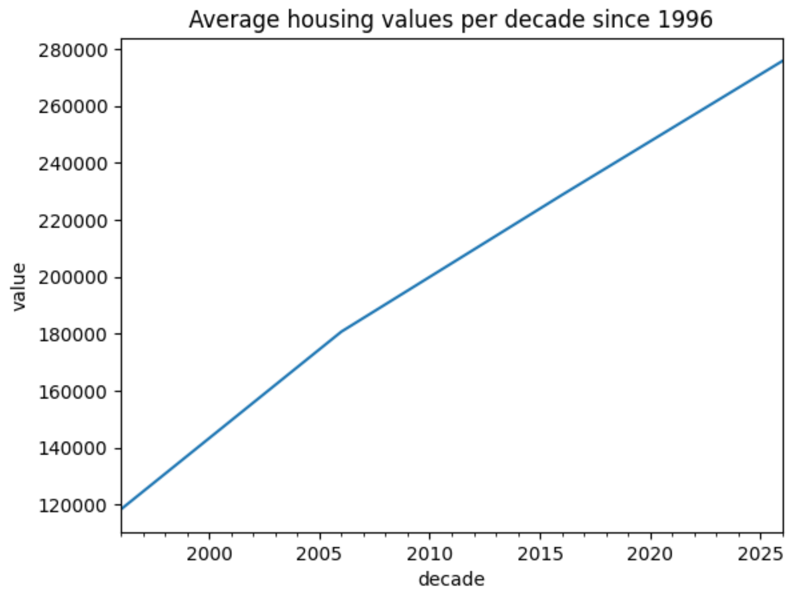
Grouping per Quarter and plotting

```
#Plotting the Quarterly annual housing average
df1_quarterly = df1['value'].resample('Q').mean()
df1_quarterly.plot();
plt.title('Line plot of the Quarterly Annual Average house values since 1996')
plt.xlabel('Quarter')
plt.ylabel('Value')
plt.show()
```



Grouping per Decade and plotting

```
#Plotting the Average housing values per decade
df1_decade = df1['value'].resample('10Y').mean()
df1_decade.plot();
plt.title('Average housing values per decade since 1996')
plt.xlabel('decade')
plt.ylabel('value')
plt.show()
```



We proceeded to group our dataframe with the Region Name and Value columns only inorder to determine the Region Names with the highest mean_values.

```
# Group the data by the 'RegionName' and 'Year' columns and calculate the mean value:
grouped_data = df1.groupby(['RegionName'])['value'].mean()
grouped_data.head()
```

```
RegionName
1001    174509.811321
1002    273152.452830
1005    172650.943396
1007    217938.113208
1008    175319.622642
Name: value, dtype: float64
```

```
#Resetting the index
grouped_new_df = grouped_data.reset_index()
grouped_new_df.columns = ['RegionName', 'mean_value']
grouped_new_df.head()
```

	RegionName	mean_value
0	1001	174509.811321
1	1002	273152.452830
2	1005	172650.943396
3	1007	217938.113208
4	1008	175319.622642

```
#Sorting the grouped dataframe using the mean_value, in descending order
grouped_new = grouped_new_df.sort_values(by = 'mean_value', ascending = False)
grouped_new.head()
```

	RegionName	mean_value
1405	10021	8.438275e+06
1403	10011	5.444482e+06
1406	10128	5.085436e+06
1404	10014	4.507875e+06
13590	94027	3.487129e+06

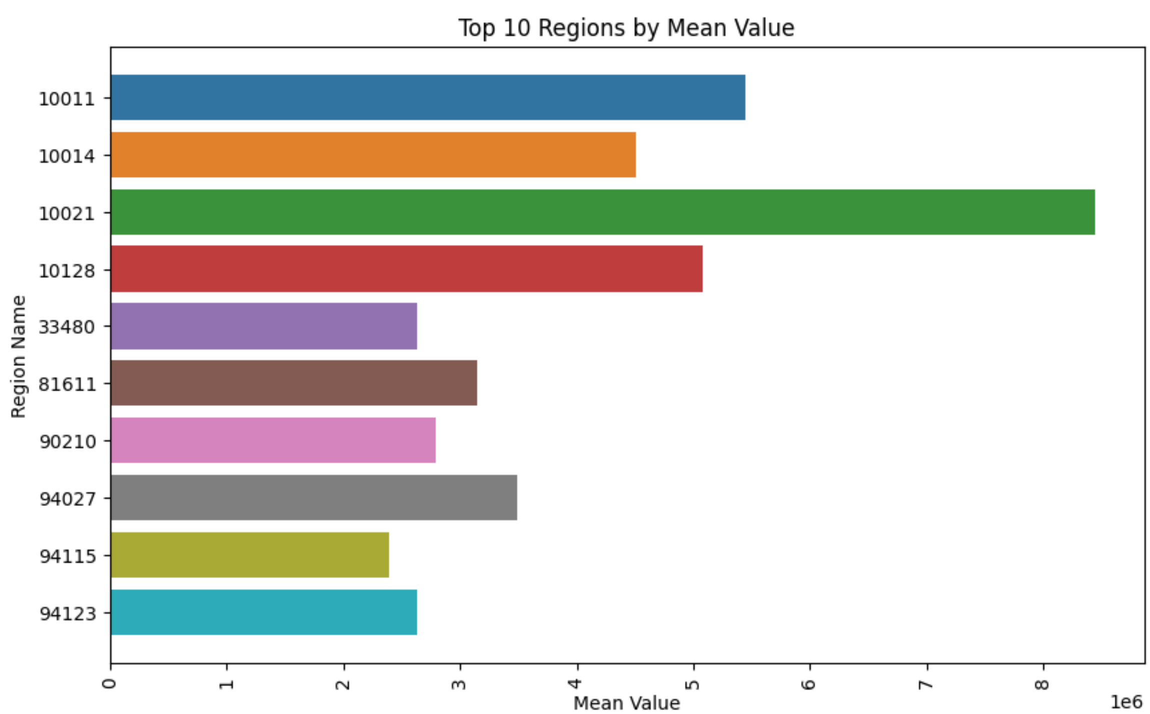
```
# slicing the top 10 values
top_10_df = grouped_new.head(10)
top_10_df
```

	RegionName	mean_value
1405	10021	8.438275e+06
1403	10011	5.444482e+06
1406	10128	5.085436e+06
1404	10014	4.507875e+06
13590	94027	3.487129e+06
12180	81611	3.147124e+06
12902	90210	2.789977e+06
5528	33480	2.634498e+06
13621	94123	2.630977e+06
13615	94115	2.399030e+06

The output above contains a list of the top 10 Region Names according to the mean_value

We proceeded to visualise this result below with a horizontal bar graph.

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


plt.figure(figsize=(10, 6))
sns.barplot(x= 'mean_value', y='RegionName', data=top_10_df, orient='h')
plt.xlabel('Mean Value')
plt.ylabel('Region Name')
plt.title('Top 10 Regions by Mean Value')
plt.xticks(rotation=90)
plt.autoscale(enable=True)
plt.show()
```



We chose our metric of investment as Return of Investment (ROI). Defined using the formula below:

```
ROI = Net Investment Gain/Cost of Investment x 100
```

Considering the investment firm is new in the market, we opted to model our dataset using the last 3 years.

```
# Subsetting the data to include only the years 2015 to 2018
start_date = '2015-04-01'
end_date = '2018-04-01'
subset_df = df1.loc[start_date:end_date]

# Print the subsetted DataFrame
print(subset_df)
```

```
            RegionID  RegionName         City State  SizeRank  CountyName  \
time
2015-04-01     58196        1001       Agawam    MA      5851     Hampden
2015-05-01     58196        1001       Agawam    MA      5851     Hampden
2015-06-01     58196        1001       Agawam    MA      5851     Hampden
2015-07-01     58196        1001       Agawam    MA      5851     Hampden
2015-08-01     58196        1001       Agawam    MA      5851     Hampden
...              ...         ...          ...   ...       ...         ...
2017-12-01    753844       29486  Summerville    SC      3188  Dorchester
2018-01-01    753844       29486  Summerville    SC      3188  Dorchester
2018-02-01    753844       29486  Summerville    SC      3188  Dorchester
2018-03-01    753844       29486  Summerville    SC      3188  Dorchester
2018-04-01    753844       29486  Summerville    SC      3188  Dorchester

               value  Year
time
2015-04-01  192200.0  2015
2015-05-01  192400.0  2015
2015-06-01  192100.0  2015
2015-07-01  191500.0  2015
2015-08-01  191000.0  2015
...              ...   ...
2017-12-01  182700.0  2017
2018-01-01  183300.0  2018
2018-02-01  184400.0  2018
2018-03-01  186500.0  2018
2018-04-01  188300.0  2018

[544751 rows x 8 columns]
```

```
# Grouping the melted dataframe by 'RegionName' and 'Year' and calculating the mean
grouped_df =pd.DataFrame(subset_df.groupby(['RegionName', 'Year',])['value'].mean())
grouped_df.columns = ['mean_value']
grouped_df.head(10)
```

		mean_value
RegionName	**Year**	
1001	**2015**	192322.222222
	2016	199033.333333
	2017	212866.666667
	2018	222425.000000
1002	**2015**	316555.555556
	2016	316950.000000
	2017	333133.333333
	2018	348950.000000
1005	**2015**	176522.222222
	2016	191550.000000

Grouping the dataset and Calculating the ROI

```
# Group the melted dataframe by 'RegionName' and calculate the mean value for the entire timeframe
grouped_df = subset_df.groupby('RegionName')['value'].mean().reset_index()
grouped_df.columns = ['RegionName', 'mean_value']

# Calculate ROI by taking percent change of the mean 'value' column
grouped_df['ROI3'] = grouped_df['mean_value'].pct_change(periods=3) * 100

# Drop the first row since it will have NaN value for ROI
grouped_df = grouped_df.dropna()

# Sort the DataFrame by ROI in descending order
```

```
grouped_df = grouped_df.sort_values('ROI3', ascending=False)

# Print the resulting DataFrame with ROI value
print(grouped_df.head(10))
```

```
       RegionName    mean_value          ROI3
1405        10021  1.844019e+07  5782.191013
1403        10011  1.162878e+07  5578.042150
5038        31561  2.391924e+06  3637.381757
1404        10014  9.444808e+06  2915.375522
12180       81611  4.106662e+06  1667.189644
5703        34102  2.665270e+06  1255.234588
1774        11975  2.934378e+06  1234.414906
12181       81615  2.622397e+06  1019.635130
12809       89413  2.030743e+06   956.295953
14060       96141  6.153486e+05   949.453791
```

We selected the top 10 RegionNames based on the ROI value as listed below;

- 10021
- 10011
- 31561
- 10014
- 81611
- 34102
- 11975
- 81615
- 89413
- 96141

```
# Select the top 10 rows
top_10_df = grouped_df.head(10)
```

```
# plotting Region names with the highest ROI
# Plot the bar chart
plt.figure(figsize=(10, 6))
sns.barplot(x=top_10_df['RegionName'], y= top_10_df['ROI3'])
plt.xlabel('Region Name')
plt.ylabel('ROI3')
plt.title('Bar chart of the Top 10 Region Names by ROI')
plt.xticks(rotation=90)
plt.autoscale(enable=True)
plt.show()
```

Bar chart of the Top 10 Region Names by ROI

Displaying the location of the top 10 Region Names

```
#Get Location Names
best10_zipcodes = list(top_10_df.RegionName.values)
for i in best10_zipcodes:
    city = df1[df1['RegionName']==i].City.values[0]
    state = df1[df1['RegionName']==i].State.values[0]
    print(f'Zipcode : {i} \nLocation: {city}, {state}\n')
```

```
Zipcode : 10021
Location: New York, NY

Zipcode : 10011
Location: New York, NY

Zipcode : 31561
Location: Sea Island, GA

Zipcode : 10014
Location: New York, NY

Zipcode : 81611
Location: Aspen, CO

Zipcode : 34102
Location: Naples, FL

Zipcode : 11975
Location: Wainscott, NY

Zipcode : 81615
Location: Snowmass Village, CO

Zipcode : 89413
Location: Glenbrook, NV

Zipcode : 96141
Location: Homewood, CA
```

```
#Creating a lost of the top 10 Region Names
region_name_list = top_10_df['RegionName'].unique().tolist()
region_name_list
```

```
[10021, 10011, 31561, 10014, 81611, 34102, 11975, 81615, 89413, 96141]
```

```
# Create a list of region names you want to filter
region_names =[10021, 10011, 31561, 10014, 81611,34102, 11975, 81615, 89413, 96141]

# Filter the original DataFrame based on the region names
filtered_df = subset_df[subset_df['RegionName'].isin(region_names)]

# Create a new DataFrame with only the 'RegionName' and 'value' columns
new_df = filtered_df.loc[:, ['RegionName', 'value']]

# Print the new DataFrame
print(new_df.tail(10))
print(len(new_df))
print(new_df['RegionName'].unique())
```

```
            RegionName     value
time
2017-07-01       96141  671300.0
2017-08-01       96141  671500.0
2017-09-01       96141  666500.0
2017-10-01       96141  667100.0
2017-11-01       96141  671800.0
2017-12-01       96141  675000.0
2018-01-01       96141  675000.0
2018-02-01       96141  677500.0
2018-03-01       96141  684400.0
2018-04-01       96141  689700.0
370
[10011 10014 10021 11975 31561 34102 81611 81615 89413 96141]
```

```
#Creating a new df for the yearly sampled values and setting the index as time
new_df['Year'] = new_df.index.year
new_df.head()
```

time	RegionName	value	Year
2015-04-01	10011	10572500.0	2015
2015-05-01	10011	10569500.0	2015
2015-06-01	10011	10674900.0	2015
2015-07-01	10011	10848100.0	2015
2015-08-01	10011	11131200.0	2015

```
# Drop the year column
new_df.drop('Year',axis=1,inplace=True)
new_df.head()
```

time	RegionName	value
2015-04-01	10011	10572500.0
2015-05-01	10011	10569500.0
2015-06-01	10011	10674900.0
2015-07-01	10011	10848100.0
2015-08-01	10011	11131200.0

Calculating the Descriptive Statistics for the top 10 Region Names in the yearly sampled dataset

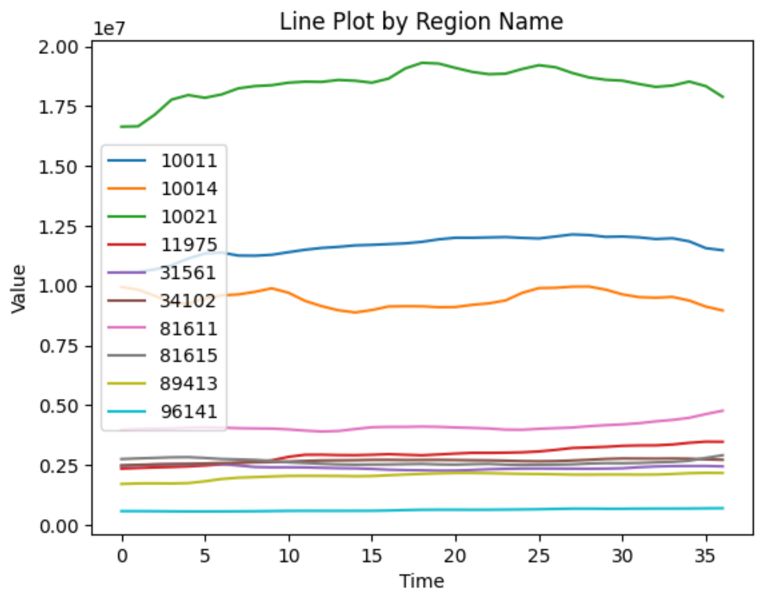
```
for region_name1 in new_df['RegionName'].unique():
    region_data = new_df[new_df['RegionName'] == region_name1]

    print(f'Value descriptive statistics for region name {region_name1}:')
    print(region_data['value'].describe())
    print()
```

```
Value descriptive statistics for region name 10011:
count    3.700000e+01
mean     1.162878e+07
std      4.438828e+05
min      1.056950e+07
25%      1.137880e+07
50%      1.173390e+07
75%      1.199880e+07
max      1.213760e+07
Name: value, dtype: float64

Value descriptive statistics for region name 10014:
count    3.700000e+01
mean     9.444808e+06
std      3.304039e+05
min      8.876100e+06
25%      9.135300e+06
50%      9.458600e+06
75%      9.701500e+06
max      9.958800e+06
Name: value, dtype: float64

Value descriptive statistics for region name 10021:
count    3.700000e+01
mean     1.844019e+07
std      6.335105e+05
min      1.664400e+07
25%      1.830710e+07
50%      1.852730e+07
75%      1.885970e+07
max      1.931490e+07
Name: value, dtype: float64

Value descriptive statistics for region name 11975:
count    3.700000e+01
```

```
mean     2.934378e+06
std      3.292249e+05
min      2.343300e+06
25%      2.692000e+06
50%      2.938700e+06
75%      3.207900e+06
max      3.473300e+06
Name: value, dtype: float64

Value descriptive statistics for region name 31561:
count    3.700000e+01
mean     2.391924e+06
std      8.126152e+04
min      2.262200e+06
25%      2.338800e+06
50%      2.378800e+06
75%      2.453900e+06
max      2.542700e+06
Name: value, dtype: float64

Value descriptive statistics for region name 34102:
count    3.700000e+01
```

Checking for trend

```
#checking for trends in the dataset
for region_name, region_data in new_df.groupby('RegionName'):
  values = region_data['value'].values.flatten()  # Flatten the values to create a 1-dimensional array
  plt.plot(values, label=region_name)  # Plot the values with region name as label


plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Line Plot by Region Name')
plt.legend()
plt.show()
```



Visualising using line plots for each of the Region Names in the top 10

```
# Group the dataframe by 'RegionName'
grouped_df1 = new_df.groupby('RegionName')

# Create a dictionary to store the separate dataframes for each region name
region_dataframes = {}

# Iterate over each group and create a separate dataframe for each region name
for region_name, region_group in grouped_df1:
    region_dataframes[region_name] = region_group.copy()

# Access the separate dataframes for each region name
```

```
for region_name, region_df in region_dataframes.items():
    print(f"Dataframe for Region Name: {region_name}")
    print(region_df.head())
    print()
```

```
Dataframe for Region Name: 10011
            RegionName       value
time
2015-04-01       10011  10572500.0
2015-05-01       10011  10569500.0
2015-06-01       10011  10674900.0
2015-07-01       10011  10848100.0
2015-08-01       10011  11131200.0

Dataframe for Region Name: 10014
            RegionName      value
time
2015-04-01       10014  9938600.0
2015-05-01       10014  9827500.0
2015-06-01       10014  9571200.0
2015-07-01       10014  9278700.0
2015-08-01       10014  9252000.0

Dataframe for Region Name: 10021
            RegionName       value
time
2015-04-01       10021  16644000.0
2015-05-01       10021  16659500.0
2015-06-01       10021  17149200.0
2015-07-01       10021  17775200.0
2015-08-01       10021  17965800.0

Dataframe for Region Name: 11975
            RegionName      value
time
2015-04-01       11975  2343300.0
2015-05-01       11975  2371200.0
2015-06-01       11975  2394400.0
2015-07-01       11975  2413700.0
2015-08-01       11975  2437600.0

Dataframe for Region Name: 31561
            RegionName      value
time
2015-04-01       31561  2453900.0
2015-05-01       31561  2482800.0
2015-06-01       31561  2511200.0
2015-07-01       31561  2530500.0
2015-08-01       31561  2531200.0

Dataframe for Region Name: 34102
            RegionName      value
time
2015-04-01       34102  2481500.0
2015-05-01       34102  2502200.0
2015-06-01       34102  2522100.0
2015-07-01       34102  2529700.0
2015-08-01       34102  2541600.0

Dataframe for Region Name: 81611
            RegionName      value
time
2015-04-01       81611  3956500.0
```

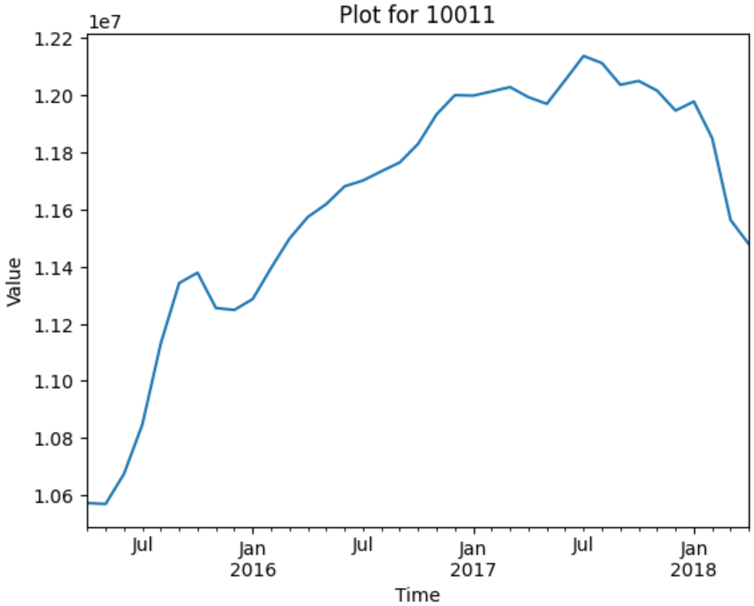
Since we want to plot the line plots of time against value, we proceeded to drop the RegionNames

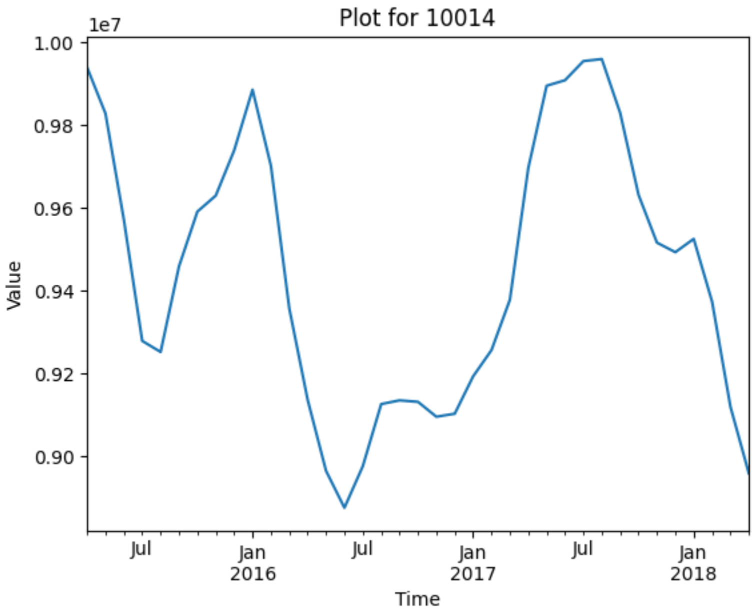
```
# Drop the column in each dataframe
# Iterate over each dataframe in the dictionary and drop the 'RegionName' column
for region_name, region_df in region_dataframes.items():
    region_df.drop('RegionName', axis=1, inplace=True)

# Access the modified dataframes
for region_name, region_df in region_dataframes.items():
    print(f"Dataframe for Region Name: {region_name}")
    print(region_df.head())
    print()
```

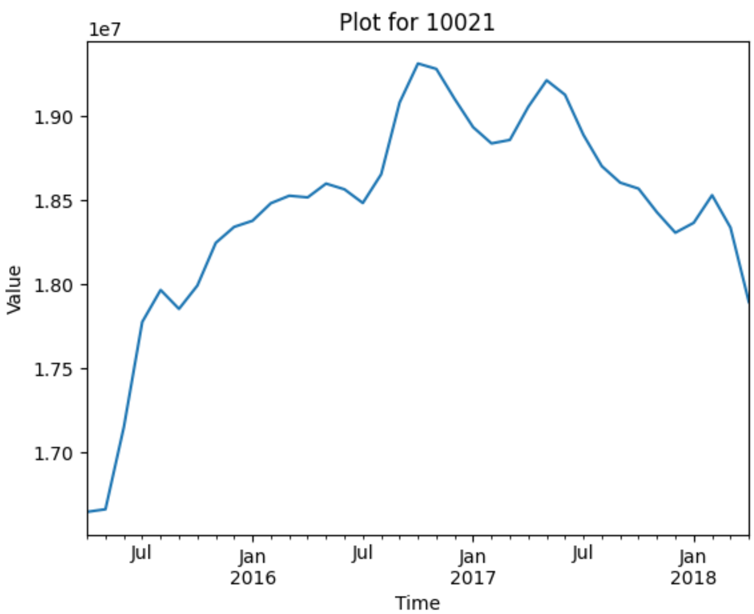
```
Dataframe for Region Name: 10011
                 value
time
```

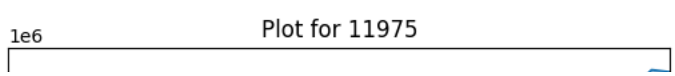
```
2015-04-01  10572500.0
2015-05-01  10569500.0
2015-06-01  10674900.0
2015-07-01  10848100.0
2015-08-01  11131200.0

Dataframe for Region Name: 10014
                value
time
2015-04-01  9938600.0
2015-05-01  9827500.0
2015-06-01  9571200.0
2015-07-01  9278700.0
2015-08-01  9252000.0

Dataframe for Region Name: 10021
                 value
time
2015-04-01  16644000.0
2015-05-01  16659500.0
2015-06-01  17149200.0
2015-07-01  17775200.0
2015-08-01  17965800.0

Dataframe for Region Name: 11975
                value
time
2015-04-01  2343300.0
2015-05-01  2371200.0
2015-06-01  2394400.0
2015-07-01  2413700.0
2015-08-01  2437600.0

Dataframe for Region Name: 31561
                value
time
2015-04-01  2453900.0
2015-05-01  2482800.0
2015-06-01  2511200.0
2015-07-01  2530500.0
2015-08-01  2531200.0

Dataframe for Region Name: 34102
                value
time
2015-04-01  2481500.0
2015-05-01  2502200.0
2015-06-01  2522100.0
2015-07-01  2529700.0
2015-08-01  2541600.0

Dataframe for Region Name: 81611
                value
time
2015-04-01  3956500.0
```

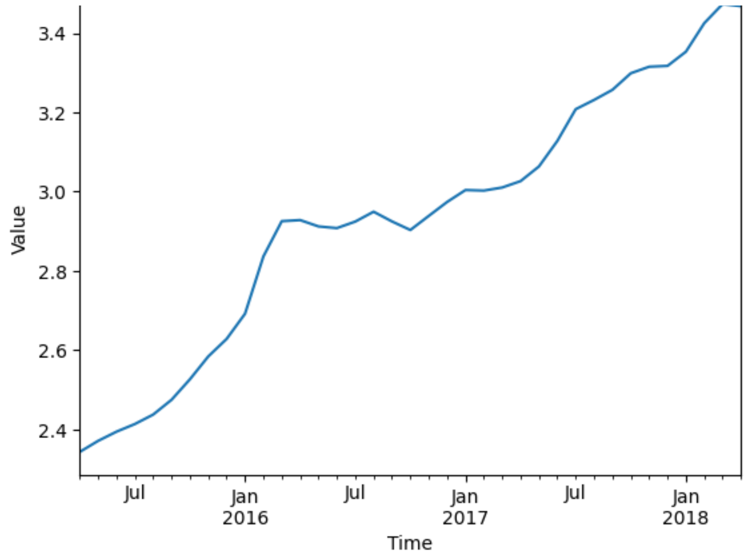
```
#Iterating over the list of RegionNames and plotting for each

for key, df in region_dataframes.items():
    # Plot the 'value' column for the current dataframe
    df['value'].plot()
    plt.title(f'Plot for {key}')
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.show()
```


Plot for 10011
1e7
1.22
1.20
1.18
1.16
1.14
1.12
1.10
1.08
1.06
Value
Jul
Jan
2016
Jul
Jan
2017
Jul
Jan
2018
Time

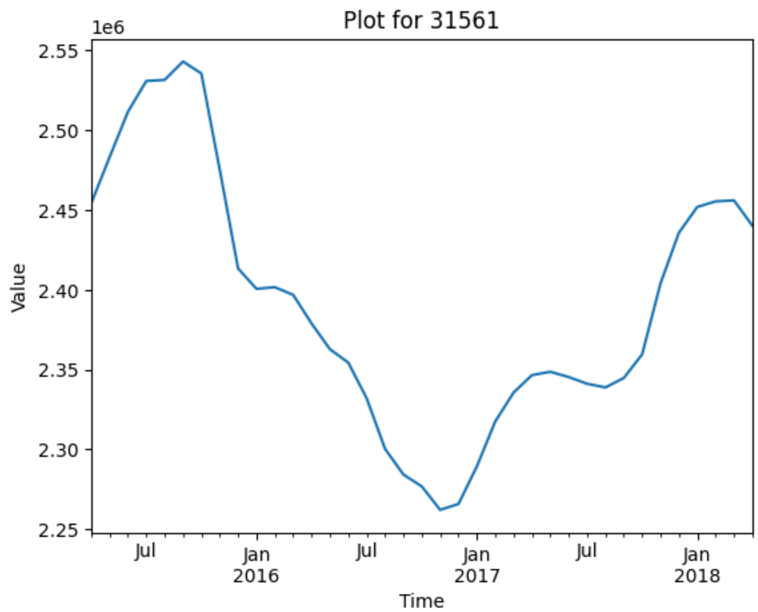

Plot for 10014
1e7
1.00
0.98
0.96
0.94
0.92
0.90
Value
Jul
Jan
2016
Jul
Jan
2017
Jul
Jan
2018
Time


Plot for 10021
1e7
1.90
1.85
1.80
1.75
1.70
Value
Jul
Jan
2016
Jul
Jan
2017
Jul
Jan
2018
Time

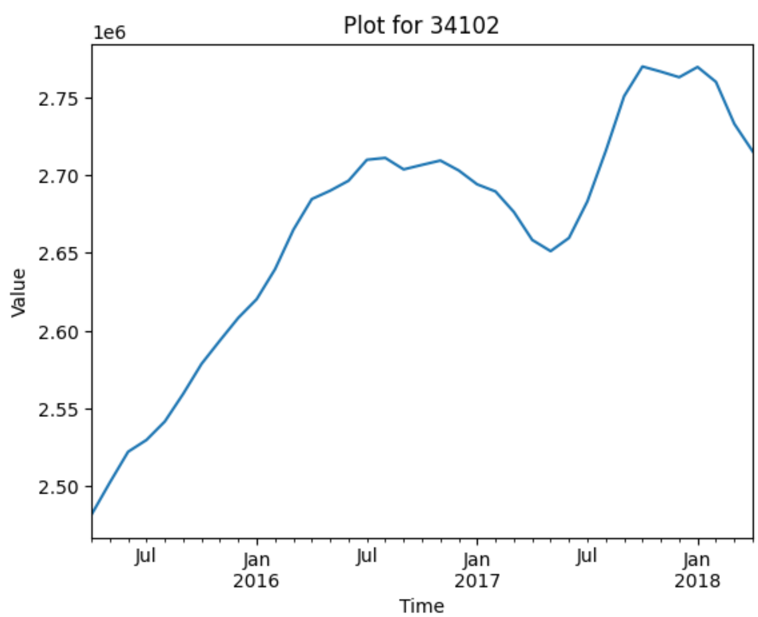

Plot for 11975
1e6



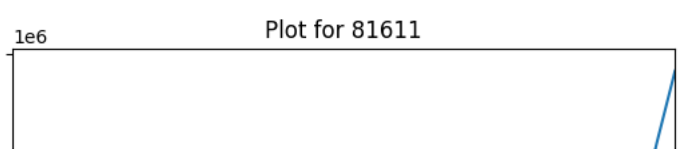
Value
Time
Jul
Jan
2016
2017
2018



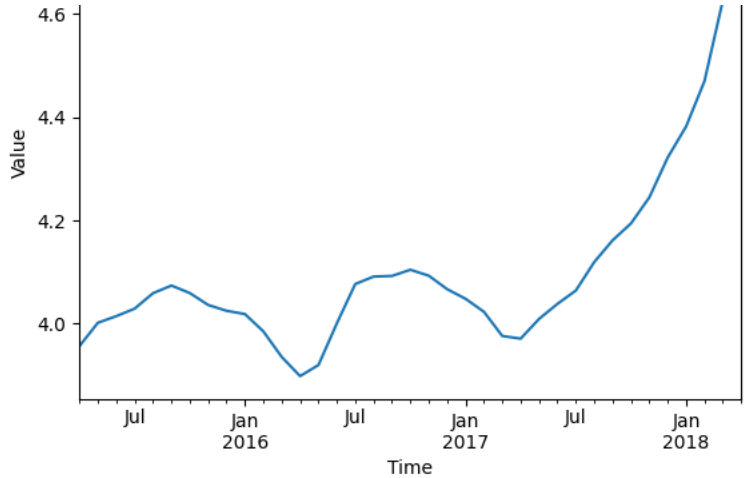
Plot for 31561
1e6
Value
Time

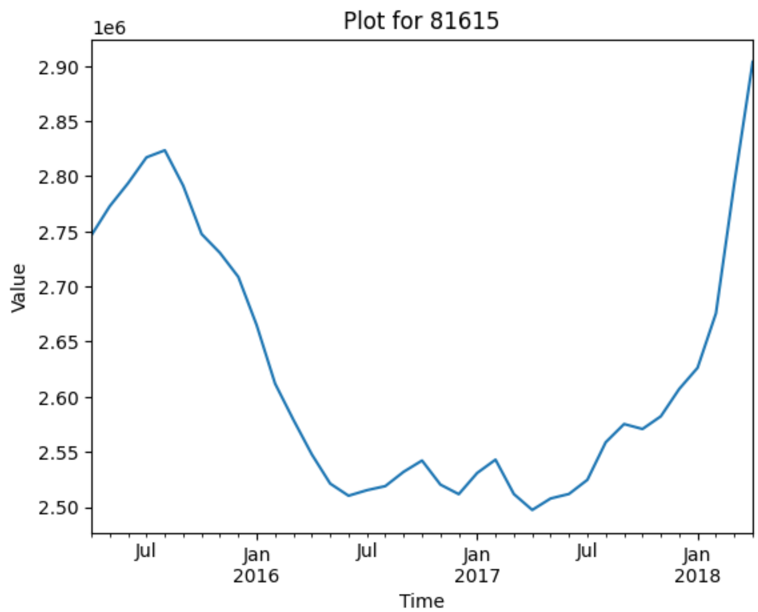


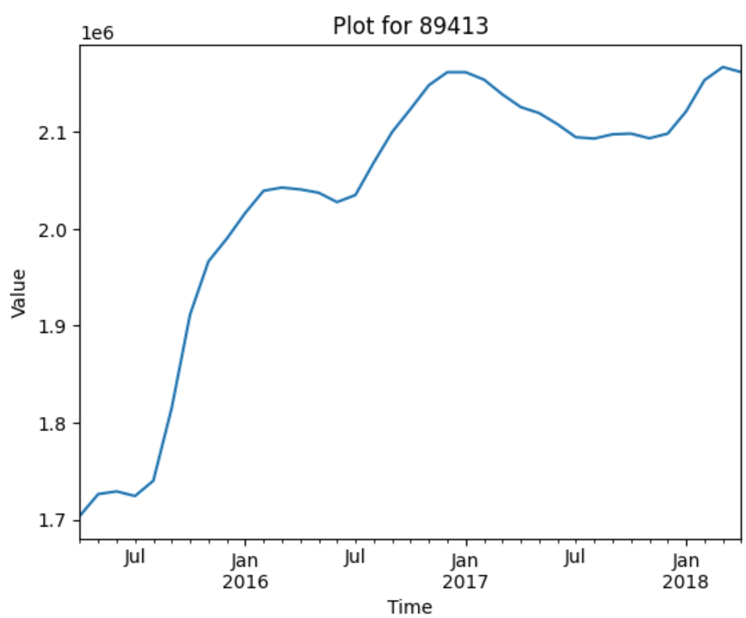
Plot for 34102
1e6
Value
Time

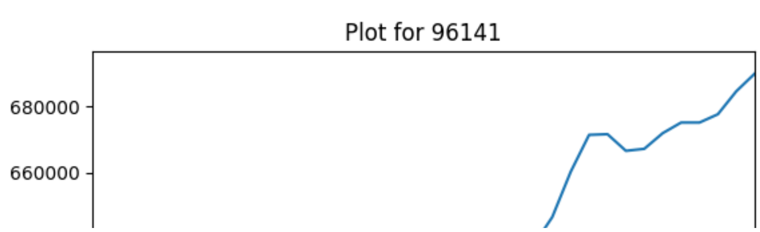


Plot for 81611
1e6
4.8


4.6
4.4
4.2
4.0
Value
Jul
Jan 2016
Jul
Jan 2017
Jul
Jan 2018
Time


Plot for 81615
1e6
2.90
2.85
2.80
2.75
2.70
2.65
2.60
2.55
2.50
Value
Jul
Jan 2016
Jul
Jan 2017
Jul
Jan 2018
Time

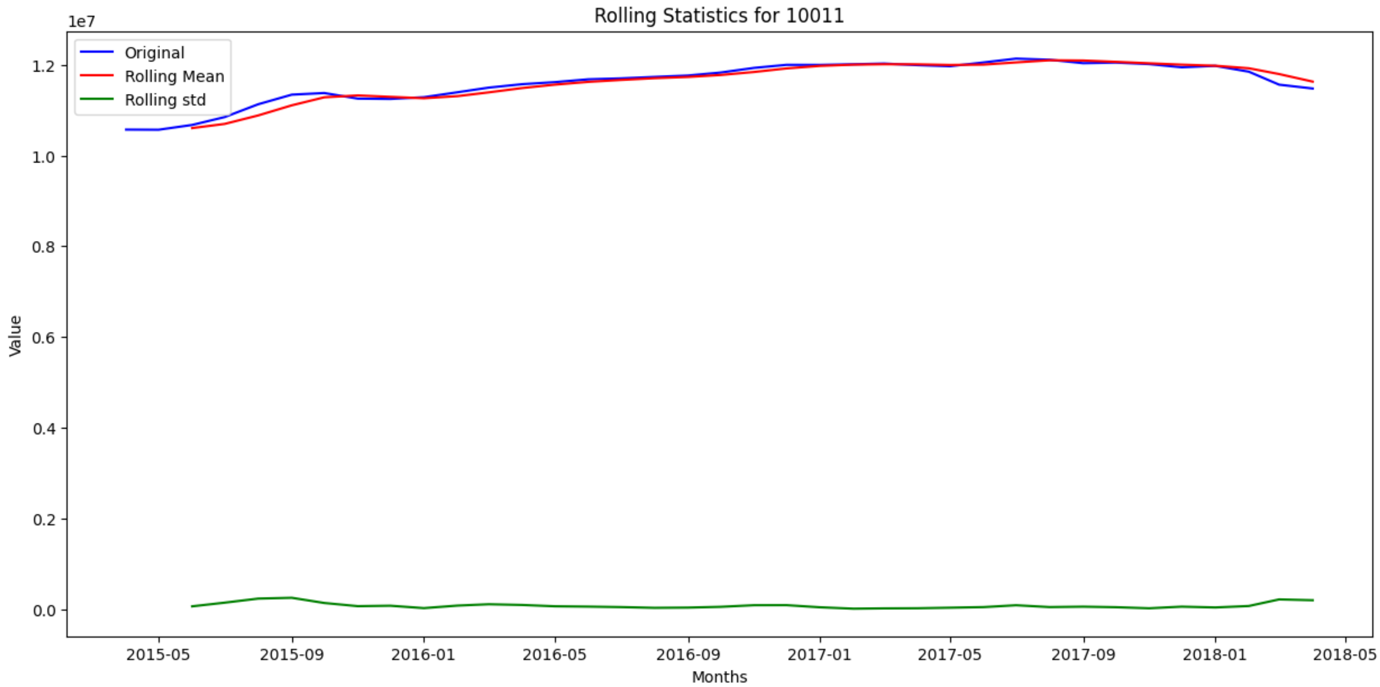

Plot for 89413
1e6
2.1
2.0
1.9
1.8
1.7
Value
Jul
Jan 2016
Jul
Jan 2017
Jul
Jan 2018
Time

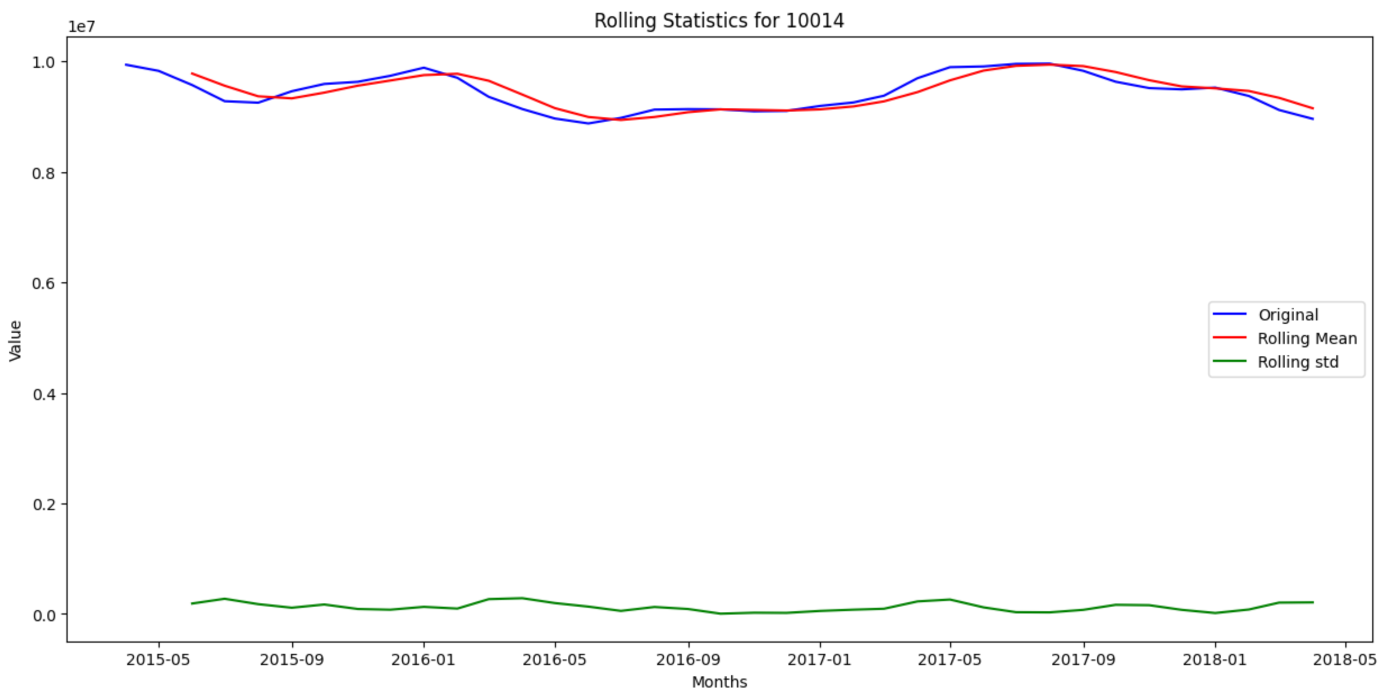

Plot for 96141
680000
660000

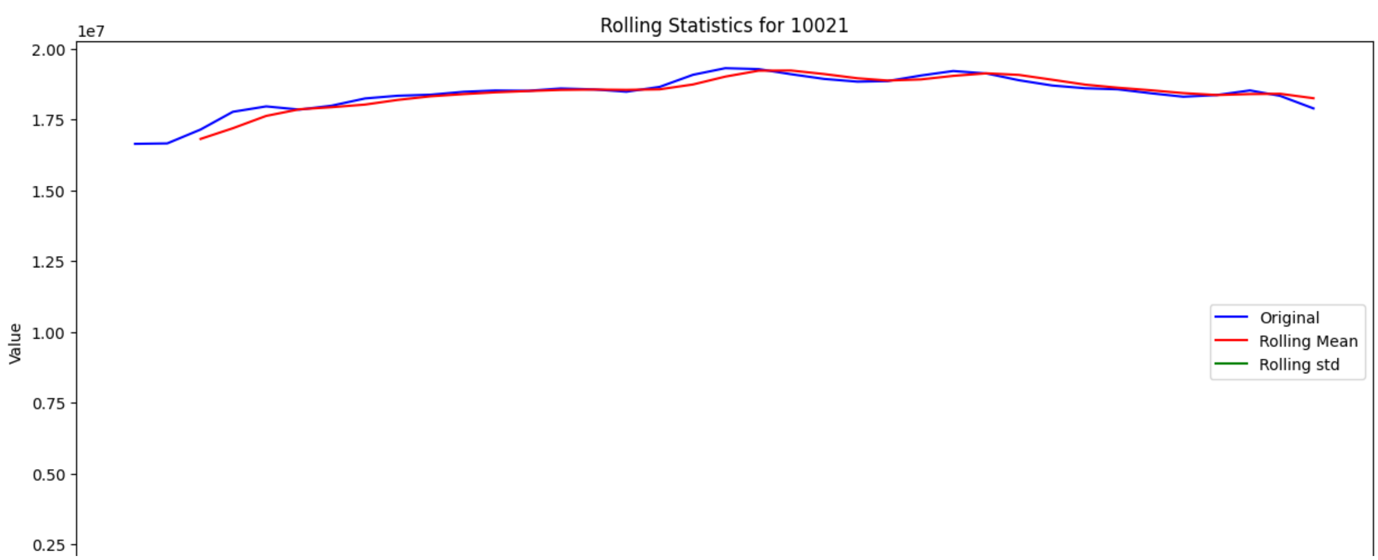
640000

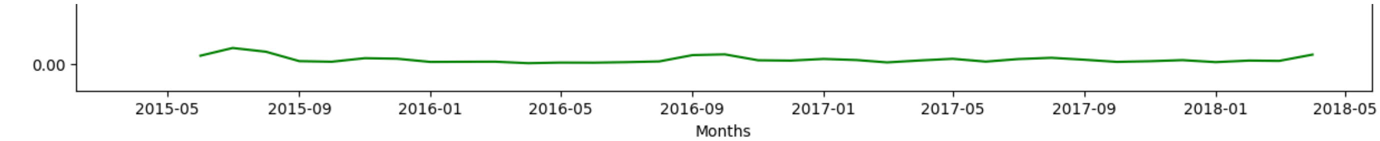
Performing Rolling Statistics

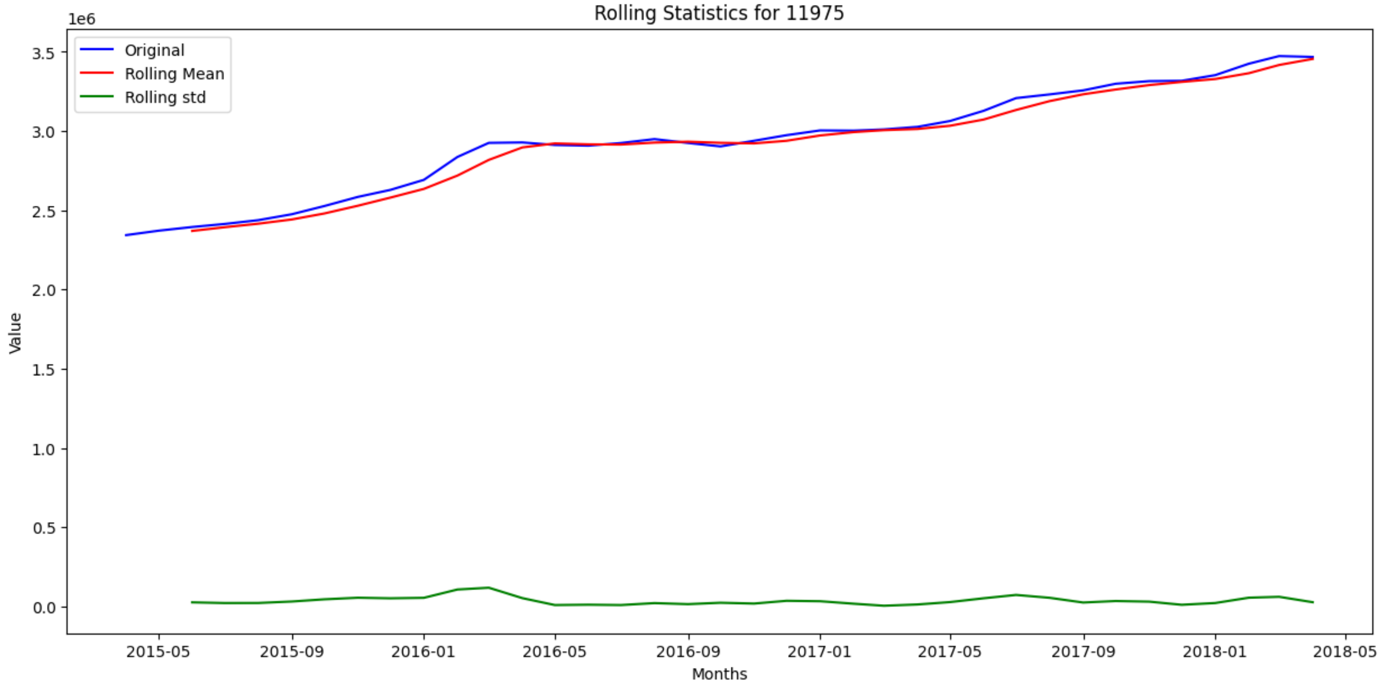
```
# Define the window size for rolling calculations
window_size = 3

for key, df in region_dataframes.items():
    # Calculate the rolling mean and standard deviation for the 'value' column
    rolling_mean = df['value'].rolling(window=window_size).mean()
    rolling_std = df['value'].rolling(window=window_size).std()

    # Plot the original values, rolling mean, and rolling standard deviation
    fig = plt.figure(figsize=(15, 7))
    plt.plot(df['value'], c='blue', label='Original')
    plt.plot(rolling_mean, c='red', label='Rolling Mean')
    plt.plot(rolling_std, c='green', label='Rolling std')
    plt.xlabel('Months')
    plt.ylabel('Value')
    plt.title(f"Rolling Statistics for {key}")
    plt.legend()
    plt.show()
```

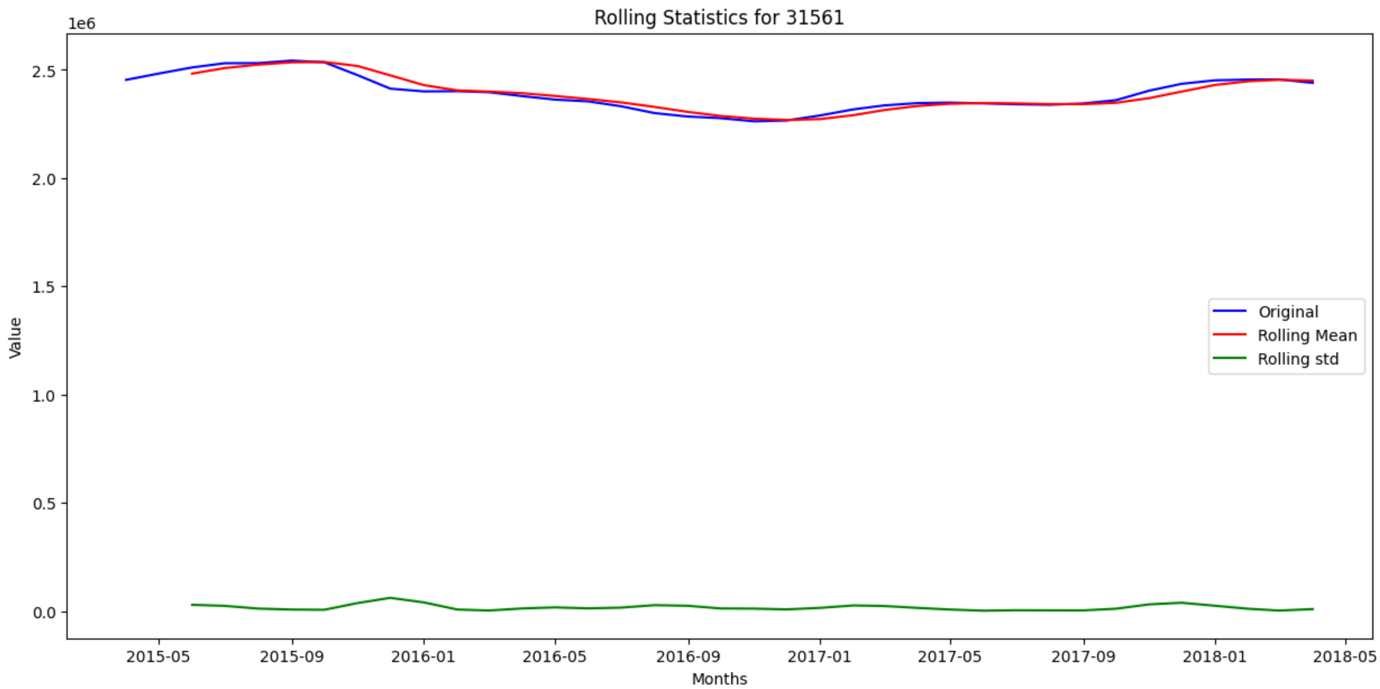

Rolling Statistics for 10011
1e7
Original
Rolling Mean
Rolling std
Value
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01
2018-05
Months

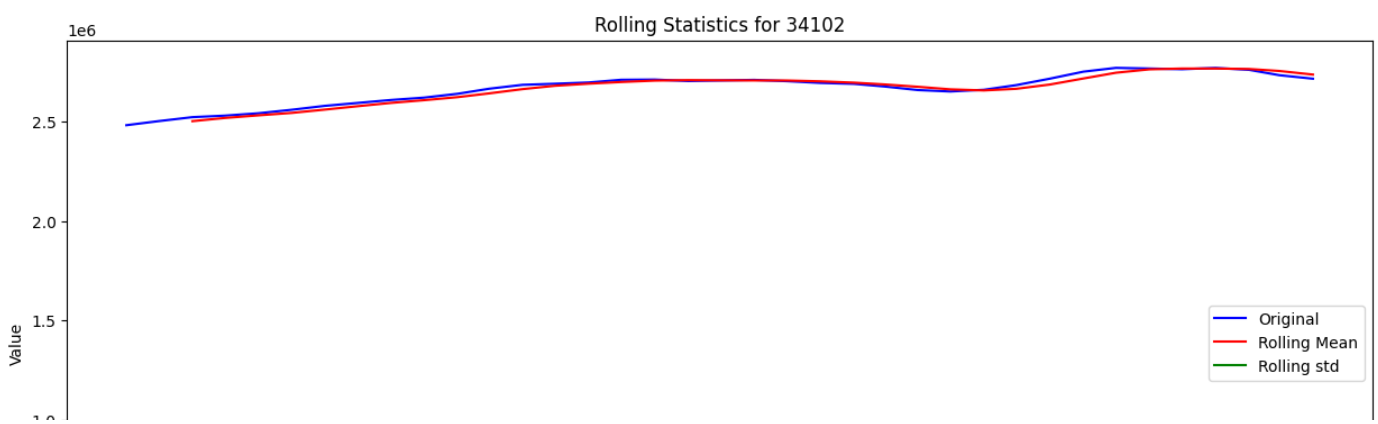

Rolling Statistics for 10014
1e7
Original
Rolling Mean
Rolling std
Value
1.0
0.8
0.6
0.4
0.2
0.0
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01
2018-05
Months

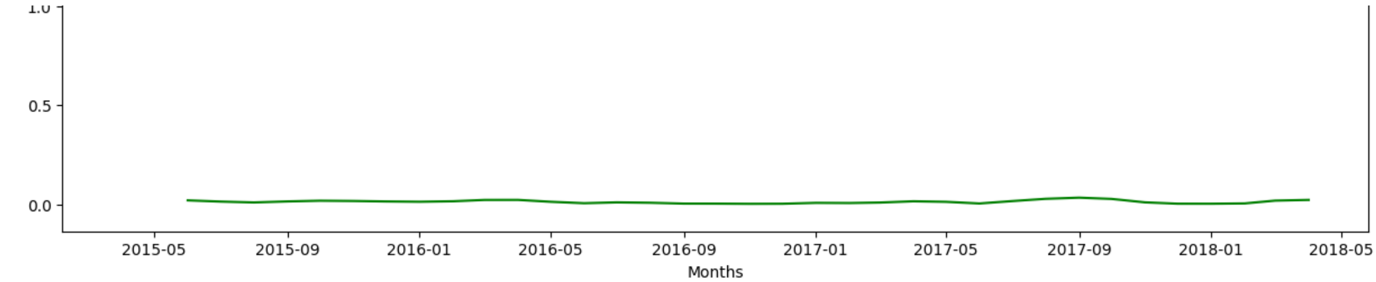

Rolling Statistics for 10021
1e7
Original
Rolling Mean
Rolling std
Value
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25

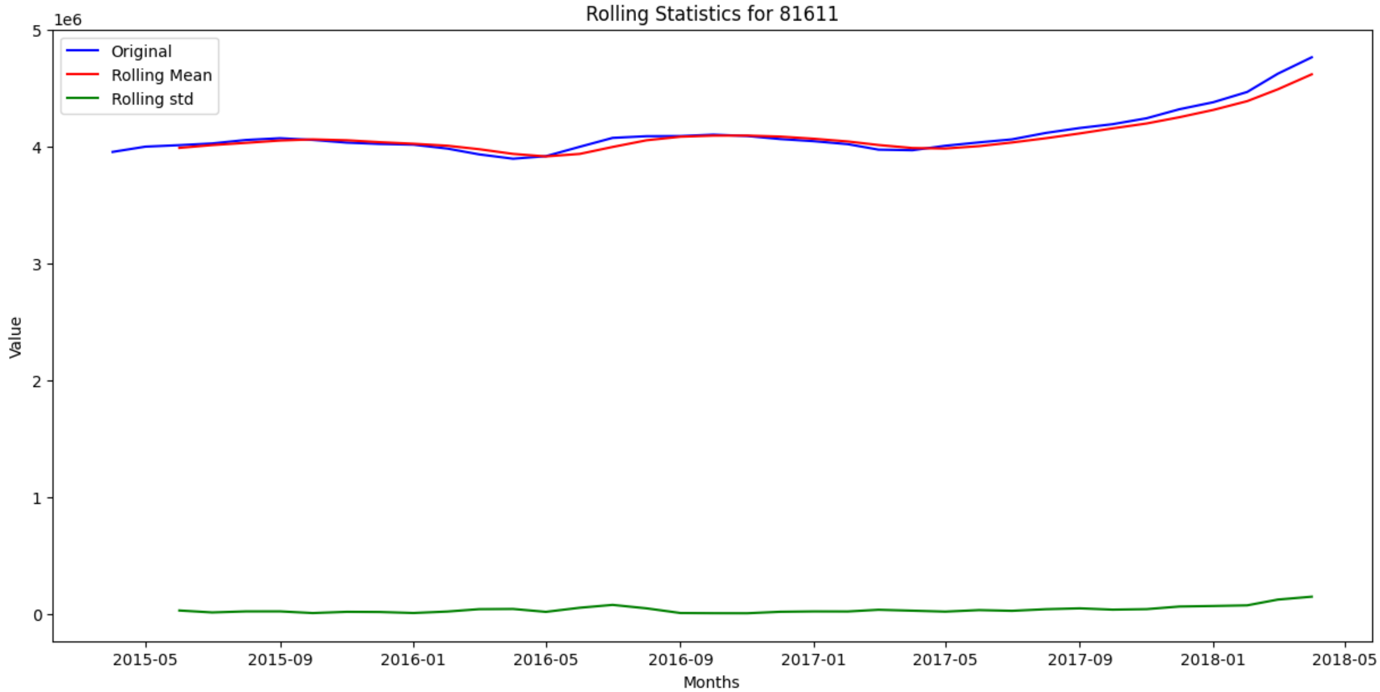

0.00
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01
2018-05
Months

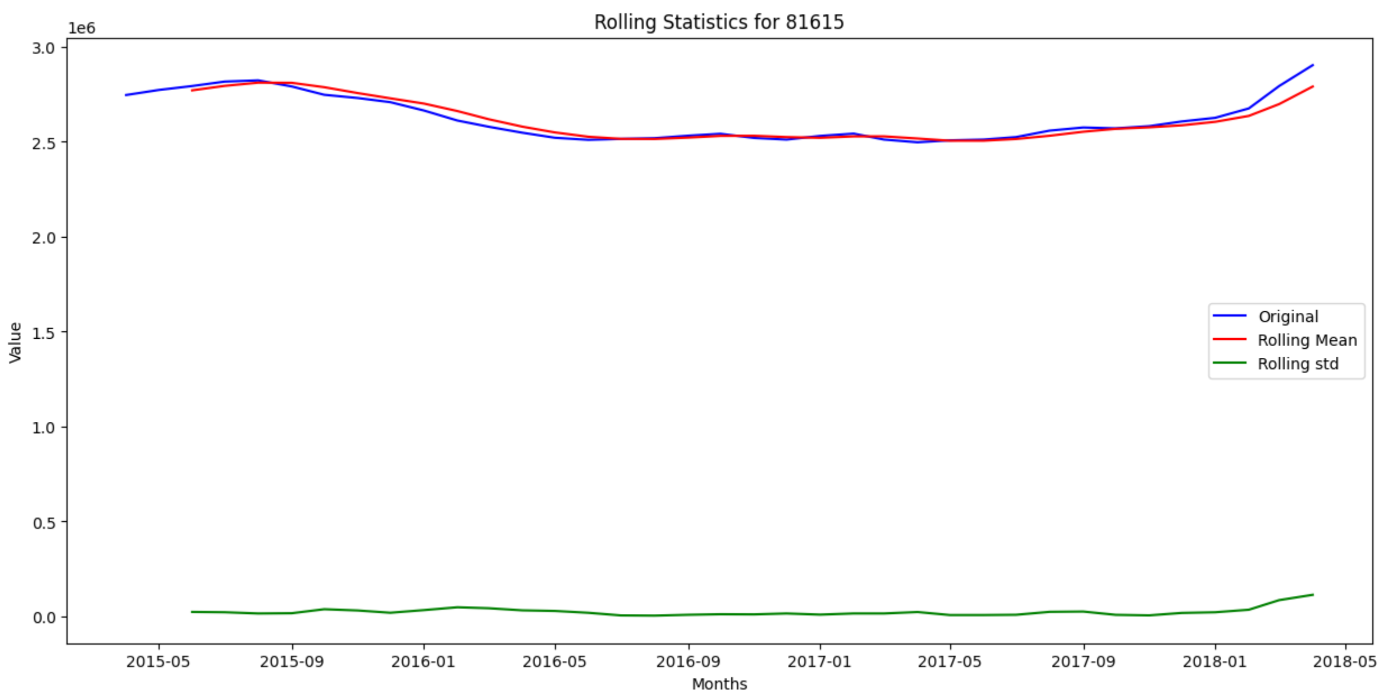

Rolling Statistics for 11975
1e6
Original
Rolling Mean
Rolling std
Value
Months

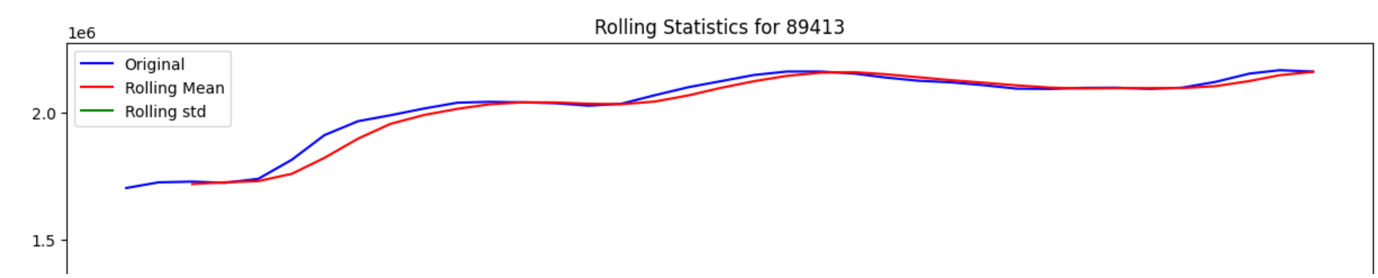

Rolling Statistics for 31561
1e6
Original
Rolling Mean
Rolling std
Value
Months

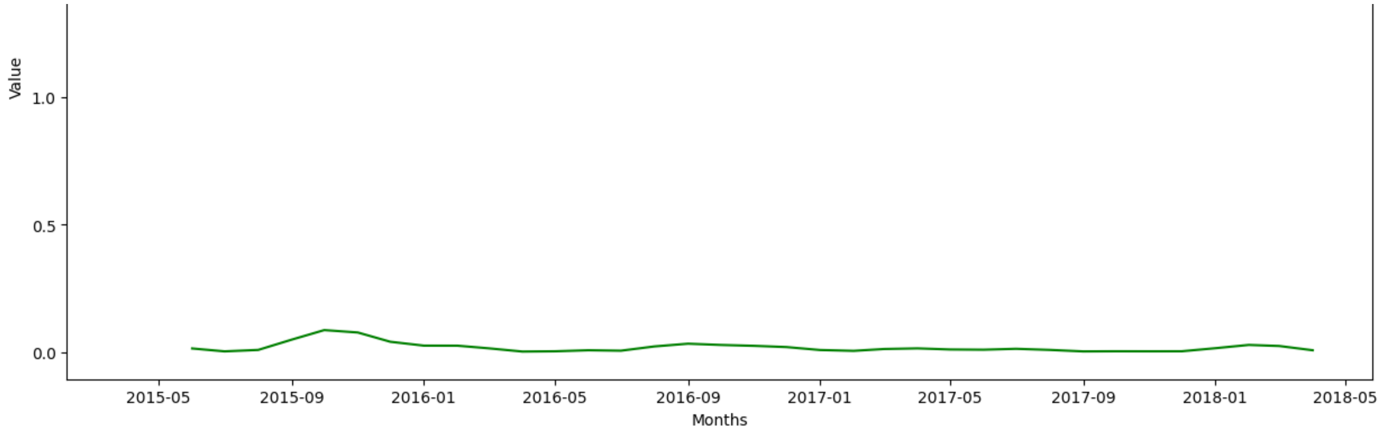

Rolling Statistics for 34102
1e6
Original
Rolling Mean
Rolling std
Value

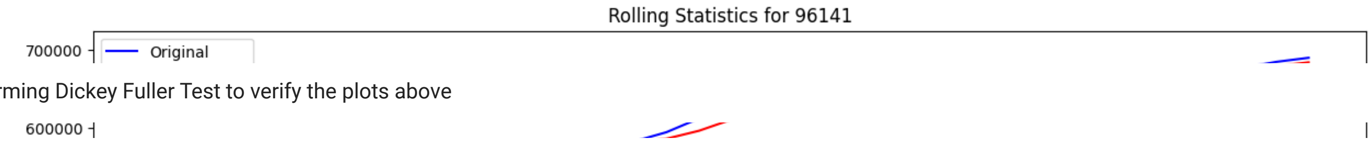

1.0
0.5
0.0
2015-05 2015-09 2016-01 2016-05 2016-09 2017-01 2017-05 2017-09 2018-01 2018-05
Months


Rolling Statistics for 81611
1e6
Original
Rolling Mean
Rolling std
Value
5
4
3
2
1
0
2015-05 2015-09 2016-01 2016-05 2016-09 2017-01 2017-05 2017-09 2018-01 2018-05
Months


Rolling Statistics for 81615
1e6
Original
Rolling Mean
Rolling std
Value
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2015-05 2015-09 2016-01 2016-05 2016-09 2017-01 2017-05 2017-09 2018-01 2018-05
Months


Rolling Statistics for 89413
1e6
Original
Rolling Mean
Rolling std
2.0
1.5





Performing Dickey Fuller Test to verify the plots above

```
#Dickey-Fuller test to verify your visual result.
from statsmodels.tsa.stattools import adfuller

for key, df in region_dataframes.items():
    # Perform Dickey-Fuller test
    print(f'Results of Dickey-Fuller Test for {key}:')
    dftest = adfuller(df['value'])

    # Extract and display test results
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic', 'p-value', '#Lags Used', 'Number of Observations Used'])
    for k, v in dftest[4].items():
        dfoutput[f'Critical Value ({k})'] = v
    print(dfoutput)
    print('\n')
```

```
Results of Dickey-Fuller Test for 10011:
Test Statistic                  0.837760
p-value                         0.992222
#Lags Used                     10.000000
Number of Observations Used    26.000000
Critical Value (1%)            -3.711212
Critical Value (5%)            -2.981247
Critical Value (10%)           -2.630095
dtype: float64


Results of Dickey-Fuller Test for 10014:
Test Statistic                 -3.831754
p-value                         0.002599
#Lags Used                      8.000000
Number of Observations Used    28.000000
Critical Value (1%)            -3.688926
Critical Value (5%)            -2.971989
Critical Value (10%)           -2.625296
dtype: float64


Results of Dickey-Fuller Test for 10021:
Test Statistic                 -0.322143
p-value                         0.922309
#Lags Used                      8.000000
Number of Observations Used    28.000000
Critical Value (1%)            -3.688926
Critical Value (5%)            -2.971989
Critical Value (10%)           -2.625296
dtype: float64


Results of Dickey-Fuller Test for 11975:
Test Statistic                 -1.169849
p-value                         0.686430
#Lags Used                      2.000000
Number of Observations Used    34.000000
Critical Value (1%)            -3.639224
Critical Value (5%)            -2.951230
Critical Value (10%)           -2.614447
dtype: float64
```

```
Results of Dickey-Fuller Test for 31561:
Test Statistic                 -2.571365
p-value                         0.099061
#Lags Used                      1.000000
Number of Observations Used    35.000000
Critical Value (1%)            -3.632743
Critical Value (5%)            -2.948510
Critical Value (10%)           -2.613017
dtype: float64


Results of Dickey-Fuller Test for 34102:
Test Statistic                 -2.339657
```

Interpreting the results:

- 10011 - The test statistic (0.837760) is greater than all the critical values, therefore we fail to reject the null hypothesis of non-stationarity. The data does not provide sufficient evidence to suggest that the time series is stationary.
- 10014 - The test statistic (-3.831754) is lower than the critical values, and the p-value (0.002599) is less than 0.05, therefore we reject the null hypothesis of non-stationarity. The data provides sufficient evidence to suggest that the time series is stationary.
- 10021 - The test statistic (-0.322143) is greater than the critical values, and the p-value (0.922309) is greater than 0.05, we fail to reject the null hypothesis of non-stationarity. The data does not provide sufficient evidence to suggest that the time series is stationary.
- 11975 - The test statistic (-1.169849) is greater than the critical values, and the p-value (0.686430) is greater than 0.05, we fail to reject the null hypothesis of non-stationarity. The data does not provide sufficient evidence to suggest that the time series is stationary.
- 31561 - Since the test statistic (-2.571365) is greater than the critical values, and the p-value (0.099061) is greater than 0.05, we fail to reject the null hypothesis of non-stationarity. The data does not provide sufficient evidence to suggest that the time series is stationary.
- 34102 - Since the test statistic (-2.339657) is greater than the critical values, and the p-value (0.159496) is greater than 0.05, we fail to reject the null hypothesis of non-stationarity. The data does not provide sufficient evidence to suggest that the time series is stationary.
- 81611 - Since the test statistic (2.577411) is greater than the critical values, and the p-value (0.999071) is greater than 0.05, we fail to reject the null hypothesis of non-stationarity. The data does not provide sufficient evidence to suggest that the time series is stationary.
- 81615 - Since the test statistic (-0.671517) is greater than the critical values, and the p-value (0.853990) is greater than 0.05, we fail to reject the null hypothesis of non-stationarity. The data does not provide sufficient evidence to suggest that the time series is stationary.
- 89416 - Since the test statistic (-2.852473) is greater than the critical values, and the p-value (0.051160) is greater than 0.05, we fail to reject the null hypothesis of non-stationarity. The data does not provide sufficient evidence to suggest that the time series is stationary.
- 96141 - Since the test statistic (-2.464834) is greater than the critical values, and the p-value (0.124267) is greater than 0.05, we fail to reject the null hypothesis of non-stationarity. The data does not provide sufficient evidence to suggest that the time series is stationary.

Based on the above interpretation, all other Zipnames contain non-stationarity apart from 10014. We proceed to perform detrending.

Checking for Stationarity

```
# Defining the check_stationarity function
def stationarity_check(TS):

    # Import adfuller
    from statsmodels.tsa.stattools import adfuller

    # Calculate rolling statistics
    roll_mean = TS.rolling(window=5, center=False).mean()
    roll_std = TS.rolling(window=5, center=False).std()

    # Perform the Dickey Fuller test
    dftest = adfuller(TS)
    # Plot rolling statistics:
    fig = plt.figure(figsize=(12,6))
    orig = plt.plot(TS, color='blue',label='Original')
    mean = plt.plot(roll_mean, color='red', label='Rolling Mean')
    std = plt.plot(roll_std, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    # Print Dickey-Fuller test results
```

```
    print('Results of Dickey-Fuller Test: \n')

    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic', 'p-value',
                                             '#Lags Used', 'Number of Observations Used'])
    for key, value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)

    return None
```

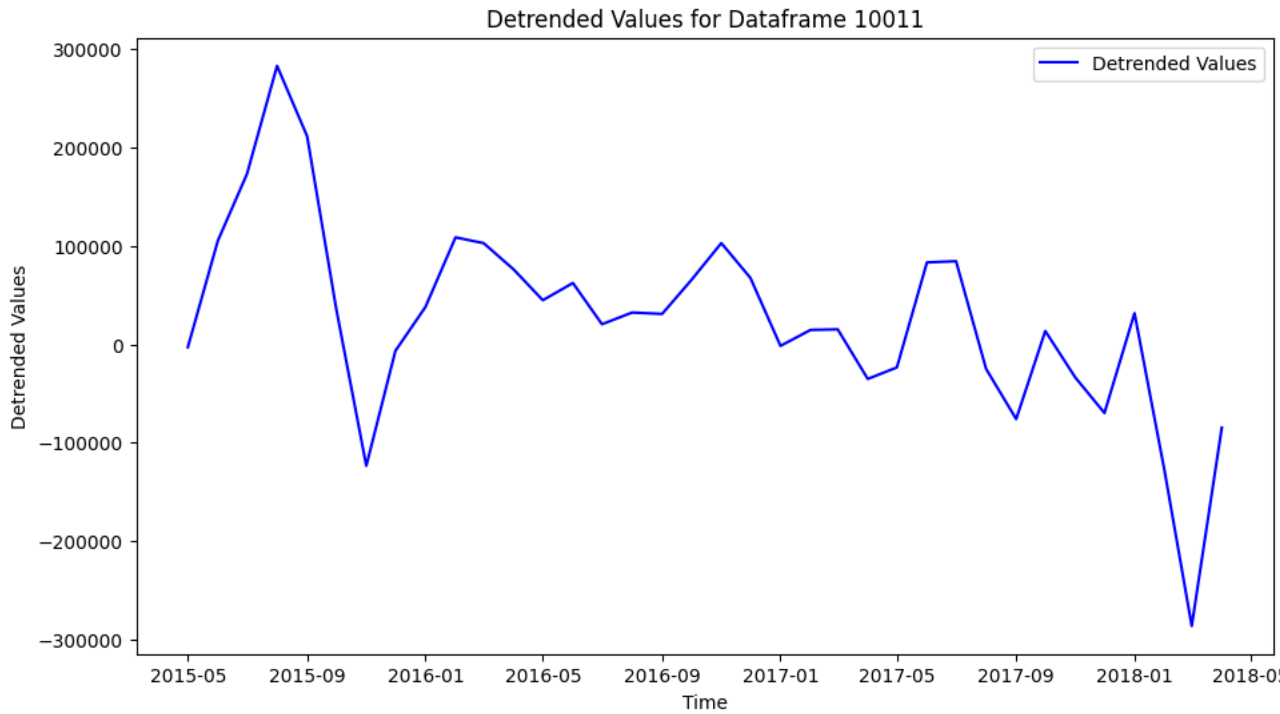
```
import statsmodels.api as sm
# Detrending the specified dataframes
# List of dataframe names to detrend
dataframe_names = [10011, 10021, 31561, 81611, 34102, 11975, 81615, 89413, 96141]

# Loop over the dataframe names
for dataframe_name in dataframe_names:
    # Get the dataframe for the dataframe name
    dataframe = region_dataframes[dataframe_name]

    # Calculate the difference between each observation and its value 12 months ago
    data_diff = dataframe.diff(periods=1)

    # Drop the missing values
    data_diff.dropna(inplace=True)

    ## Plot the detrended data
    fig = plt.figure(figsize=(11, 6))
    plt.plot(data_diff, color='blue', label='Detrended Values')
    plt.title(f"Detrended Values for Dataframe {dataframe_name}")
    plt.xlabel('Time')
    plt.ylabel('Detrended Values')
    plt.legend()
    plt.show()

    # Perform the stationarity check
    print(f"Stationarity Check for Dataframe {dataframe_name}:")
    stationarity_check(data_diff)
    #adf_result = sm.tsa.stattools.adfuller(data_diff)
    #p_value = adf_result[1]
    #print('P_value:',p_value)
    print()
```

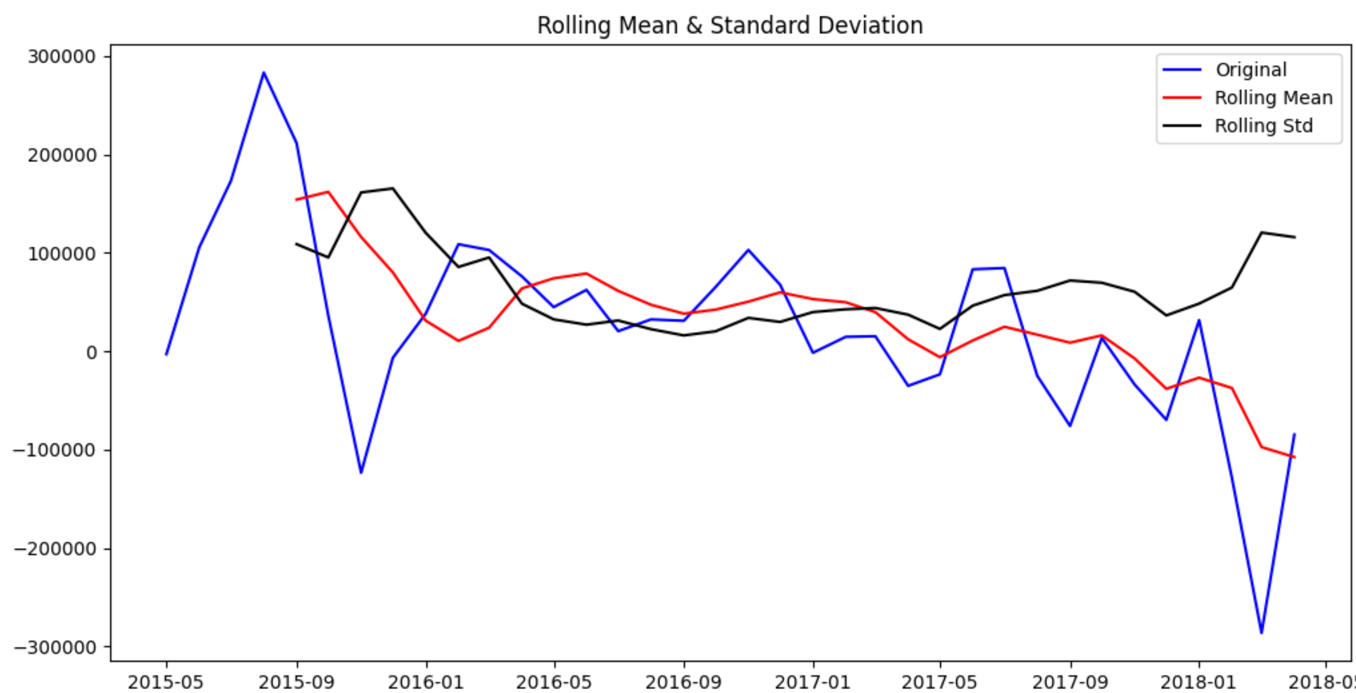
Show hidden output

Performing Detrending of our dataset and storing in a new dictionary

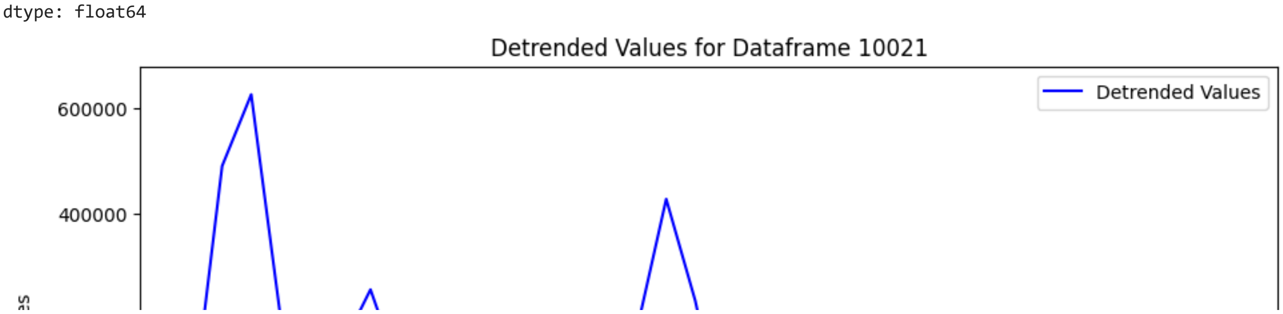
```
# Import adfuller
from statsmodels.tsa.stattools import adfuller

## Create a dictionary to store the detrended dataframes
detrended_dataframes = {}

dataframe_names = [10011, 10021, 31561, 81611, 34102, 11975, 81615, 89413, 96141]

# Loop over the dataframe names
for dataframe_name in dataframe_names:
    # Get the dataframe for the dataframe name
    dataframe = region_dataframes[dataframe_name]

    # Calculate the difference between each observation and its value 12 months ago
    data_diff = dataframe.diff(periods=1)

    # Drop the missing values
    data_diff.dropna(inplace=True)

    ## Plot the detrended data
    fig = plt.figure(figsize=(11, 6))
    plt.plot(data_diff, color='blue', label='Detrended Values')
    plt.title(f"Detrended Values for Dataframe {dataframe_name}")
    plt.xlabel('Time')
    plt.ylabel('Detrended Values')
```

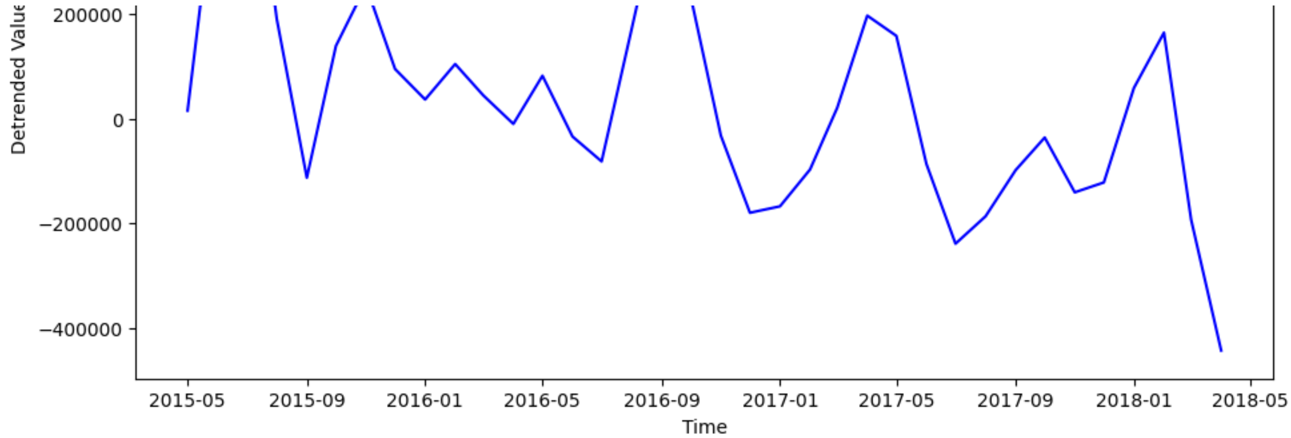
```
plt.legend()
plt.show()

# Perform the stationarity check
def stationarity_check(TS):
    # Calculate rolling statistics
    roll_mean = TS.rolling(window=5, center=False).mean()
    roll_std = TS.rolling(window=5, center=False).std()

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

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

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

    return None

# Call the stationarity_check function
stationarity_check(data_diff)

# Store the detrended dataframe in the detrended_dataframes dictionary
detrended_dataframes[dataframe_name] = data_diff
```

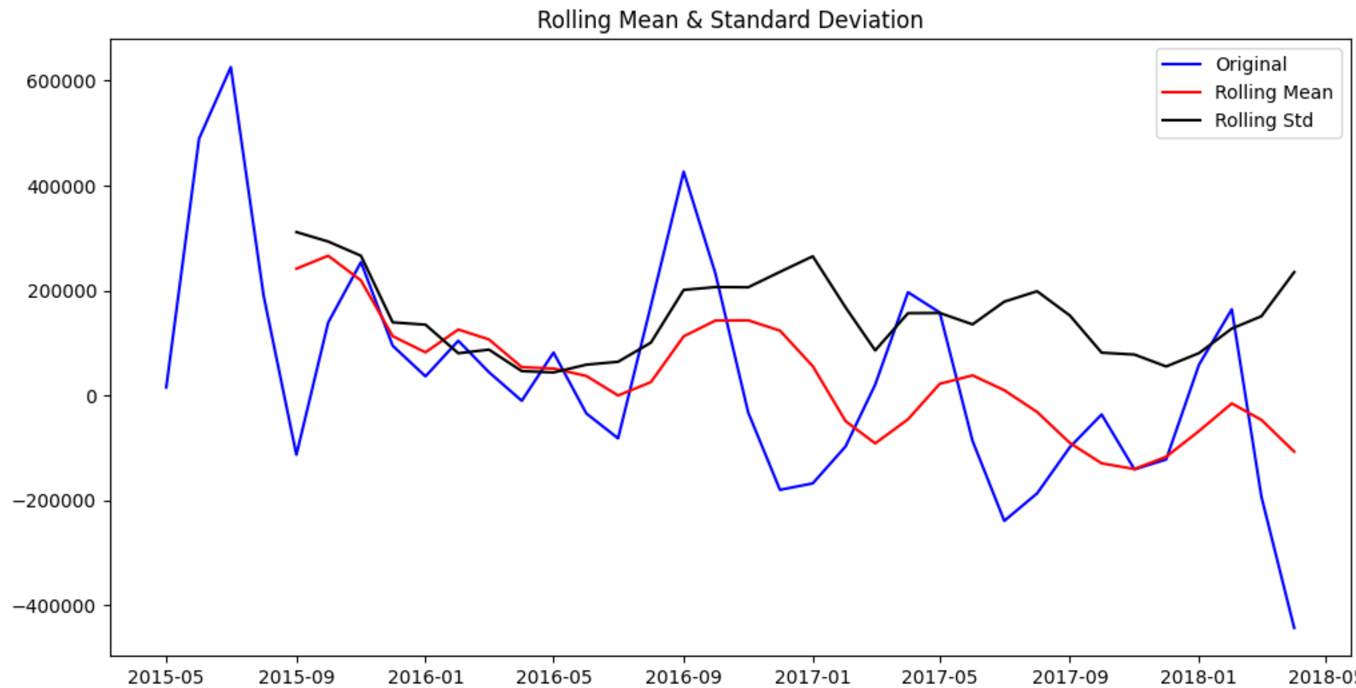




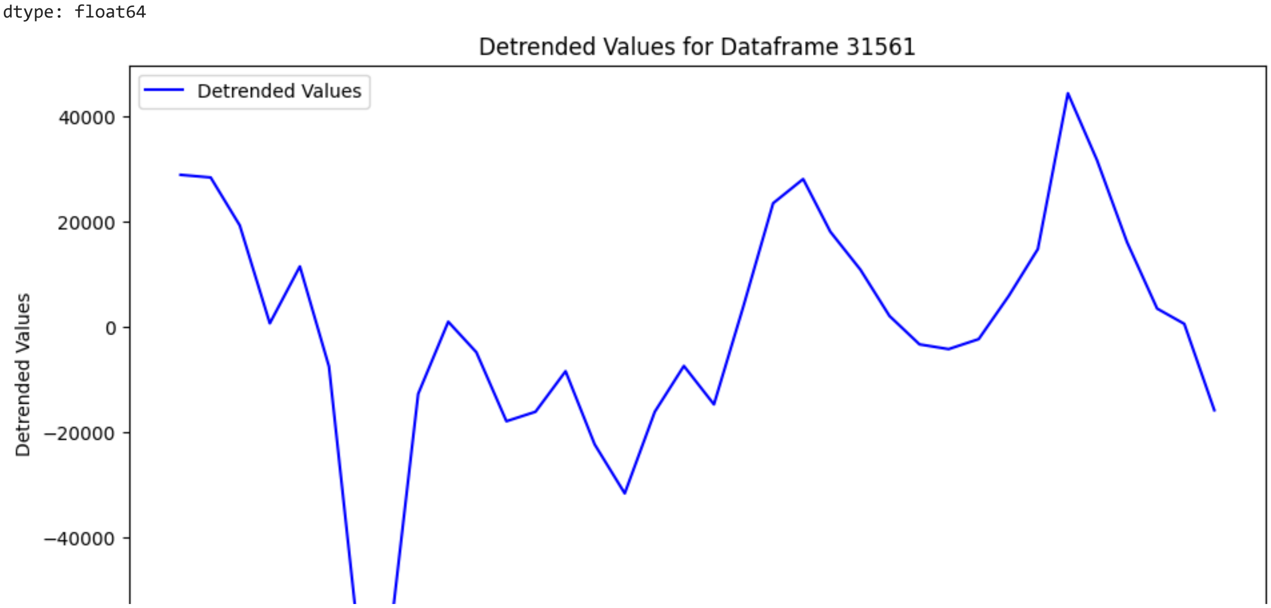

```
Results of Dickey-Fuller Test:

Test Statistic                  2.625870
p-value                         0.999079
#Lags Used                     10.000000
Number of Observations Used    25.000000
Critical Value (1%)            -3.723863
Critical Value (5%)            -2.986489
Critical Value (10%)           -2.632800
dtype: float64
```

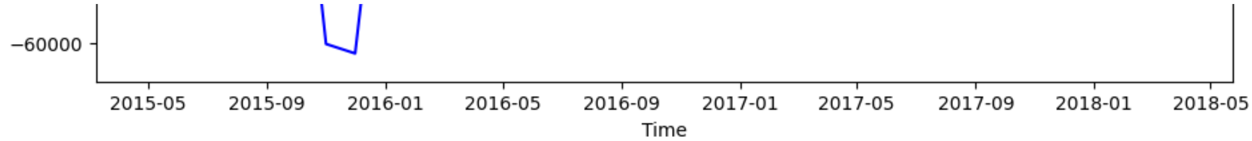


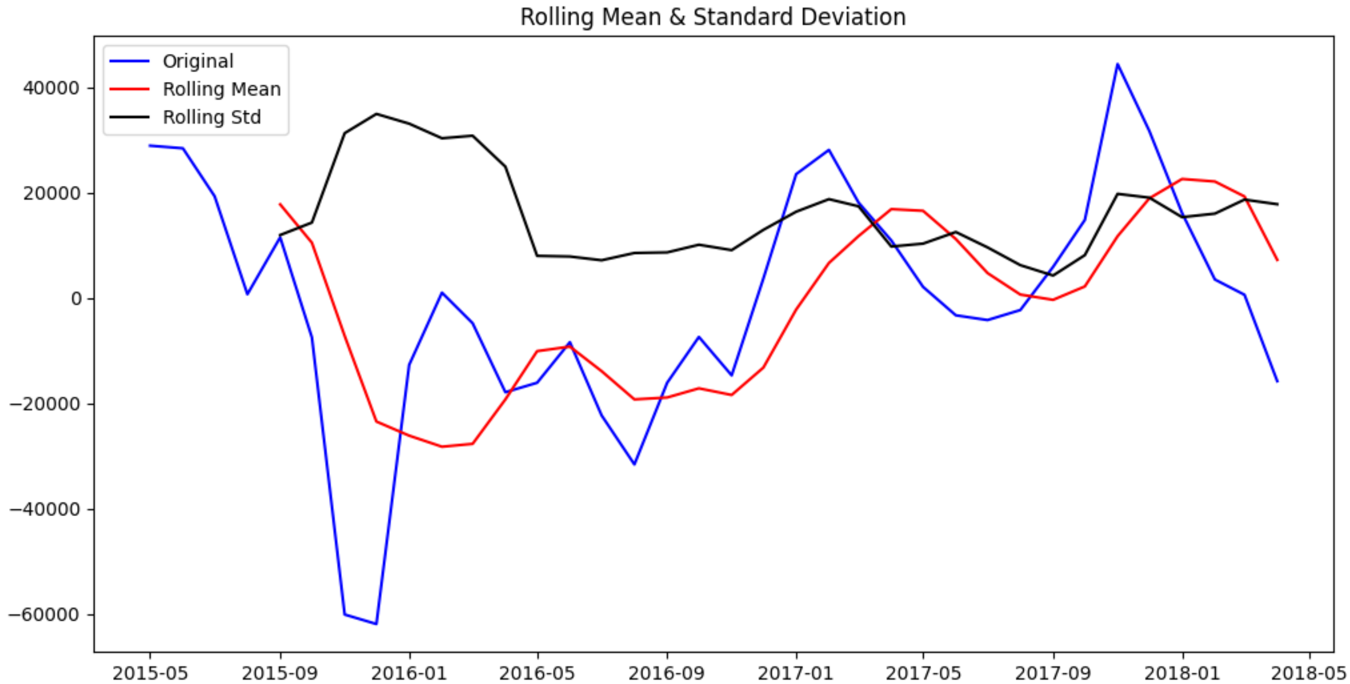




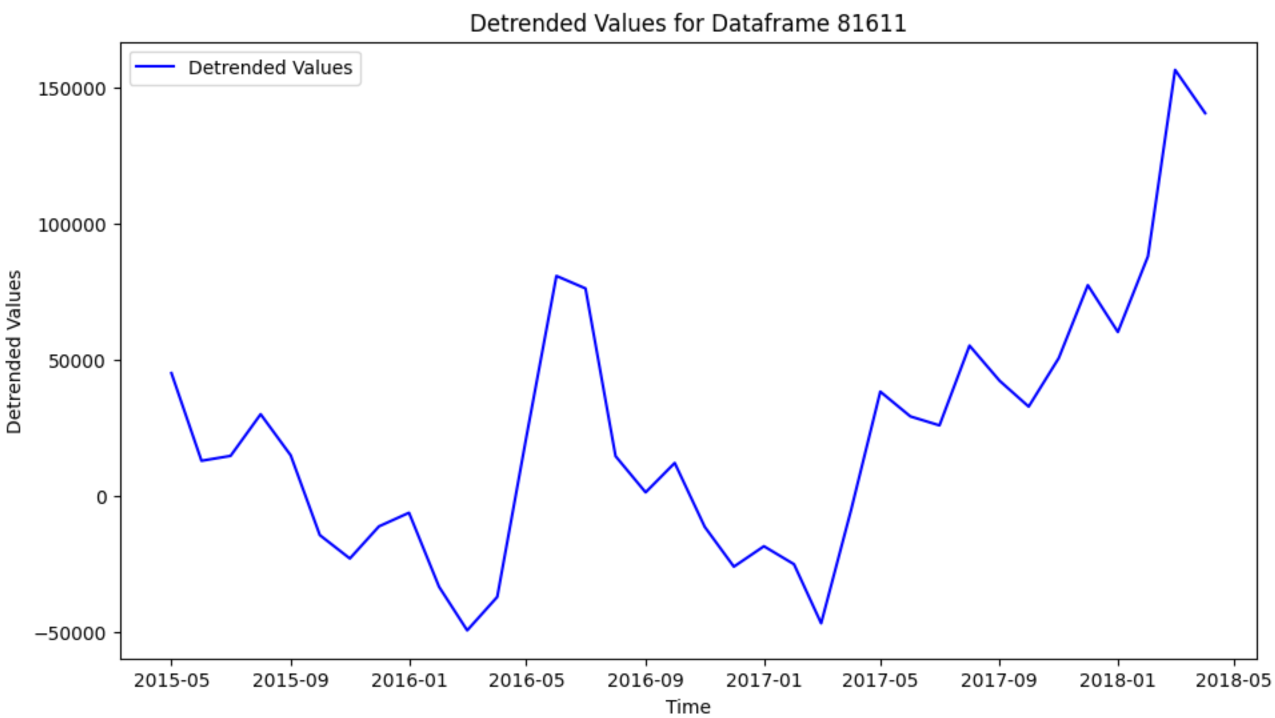

```
Results of Dickey-Fuller Test:

Test Statistic                 -1.093233
p-value                         0.717763
#Lags Used                      7.000000
Number of Observations Used    28.000000
Critical Value (1%)            -3.688926
Critical Value (5%)            -2.971989
Critical Value (10%)           -2.625296
dtype: float64
```

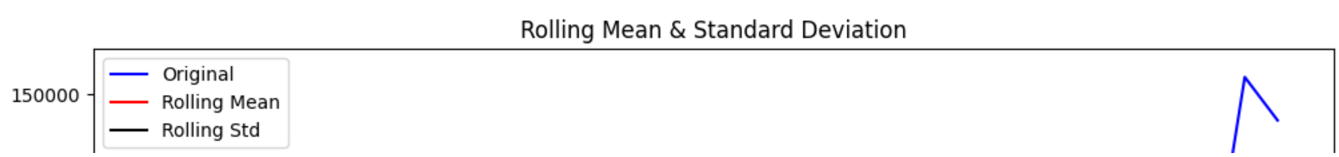


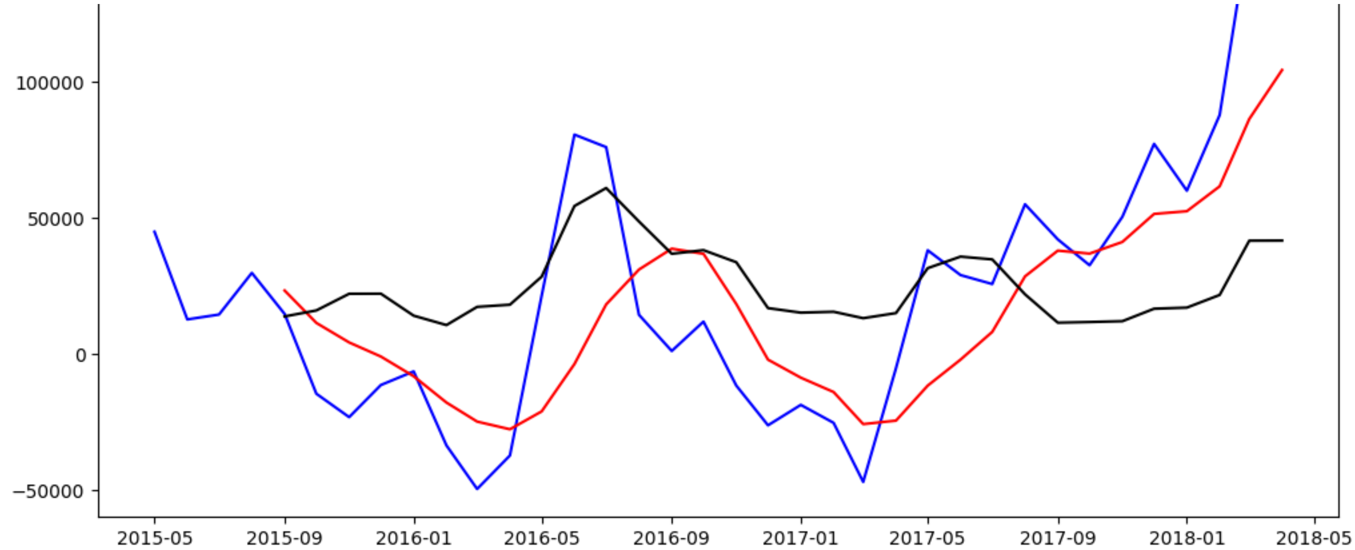




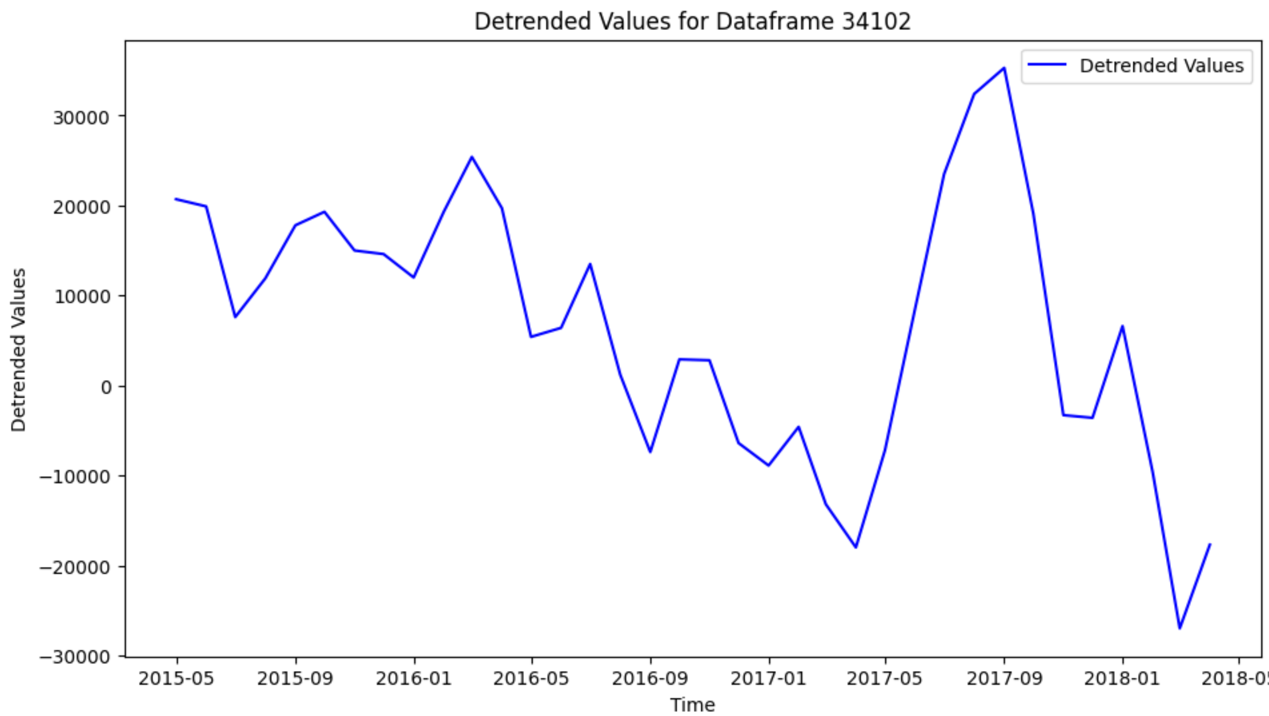

```
Results of Dickey-Fuller Test:

Test Statistic                  -3.532639
p-value                          0.007187
#Lags Used                       1.000000
Number of Observations Used     34.000000
Critical Value (1%)             -3.639224
Critical Value (5%)             -2.951230
Critical Value (10%)            -2.614447
dtype: float64
```

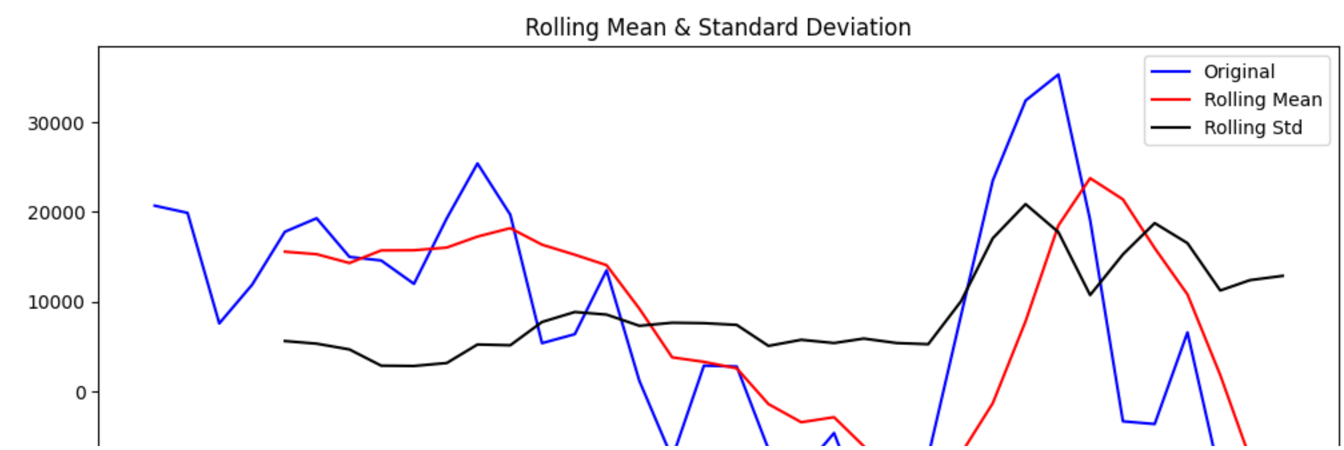


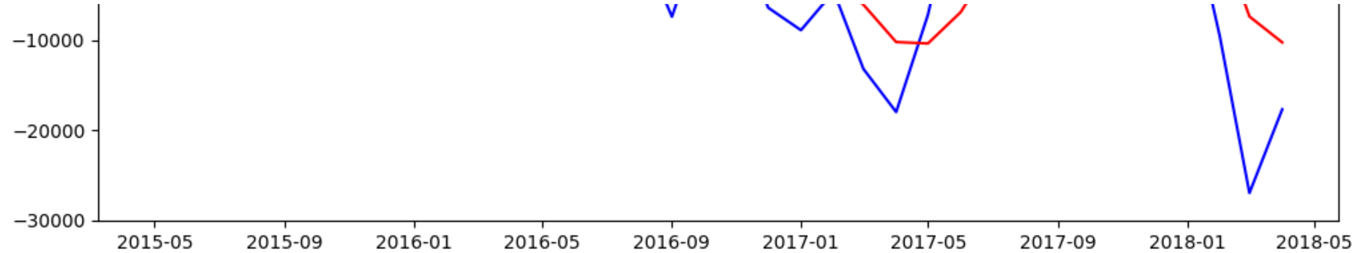




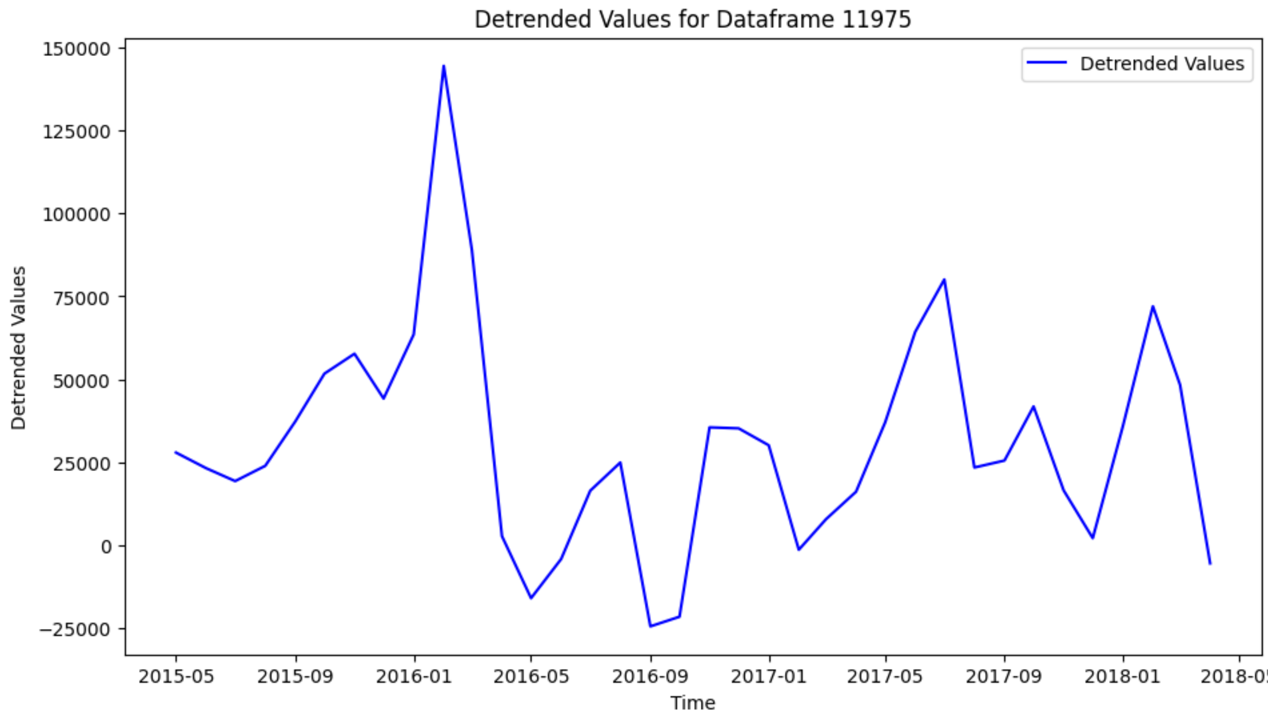

```
Results of Dickey-Fuller Test:

Test Statistic                   0.235991
p-value                          0.974209
#Lags Used                       8.000000
Number of Observations Used     27.000000
Critical Value (1%)             -3.699608
Critical Value (5%)             -2.976430
Critical Value (10%)            -2.627601
dtype: float64
```

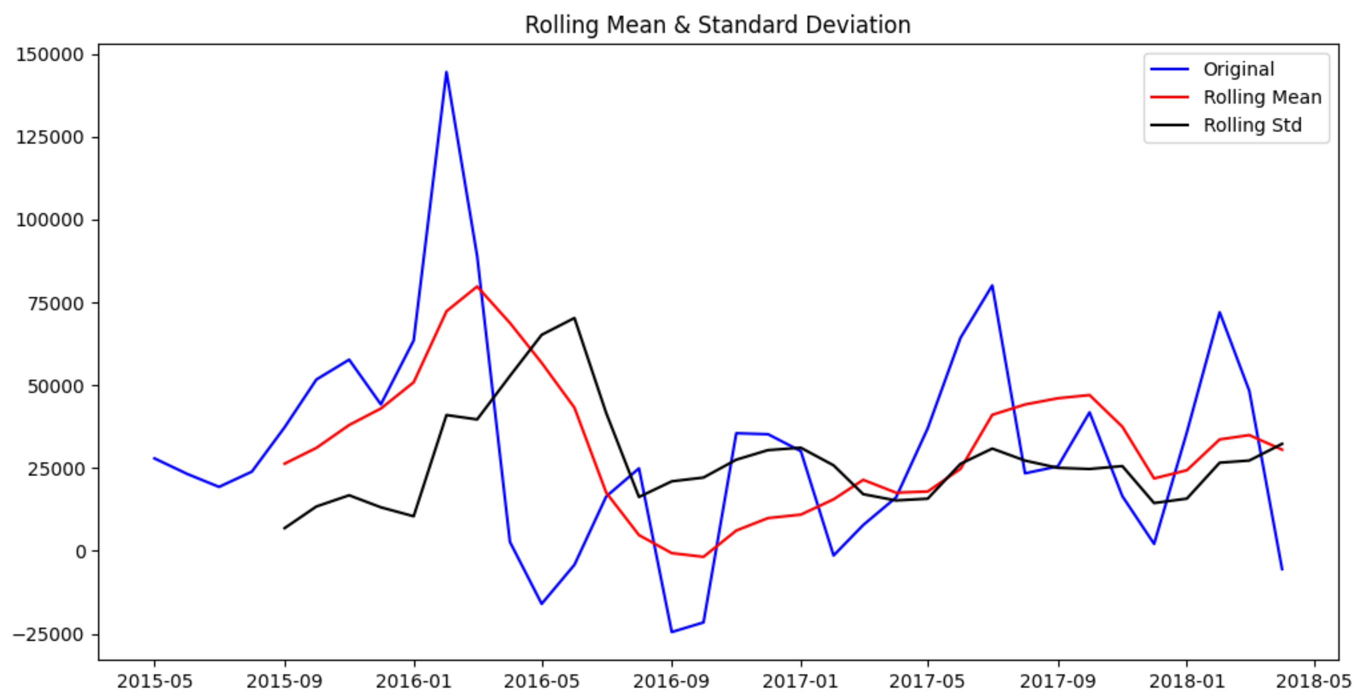






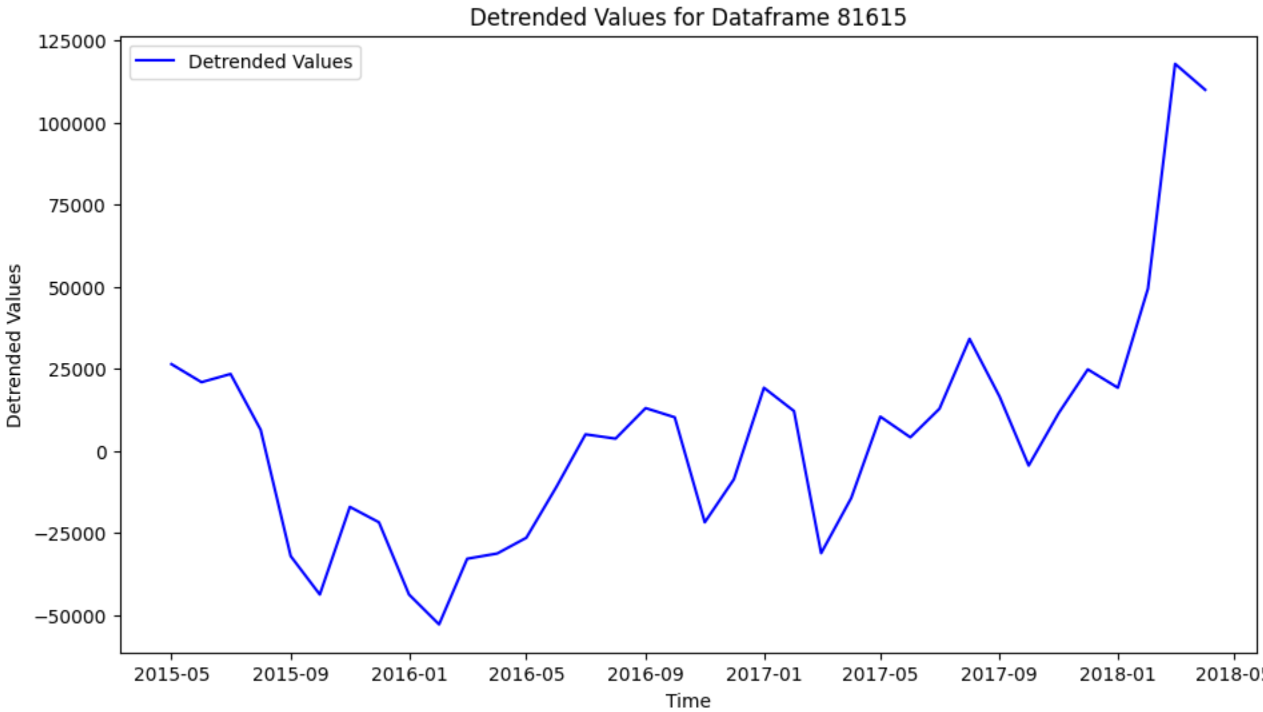

```
Results of Dickey-Fuller Test:

Test Statistic                  -2.949263
p-value                          0.039912
#Lags Used                       3.000000
Number of Observations Used     32.000000
Critical Value (1%)             -3.653520
Critical Value (5%)             -2.957219
Critical Value (10%)            -2.617588
dtype: float64
```

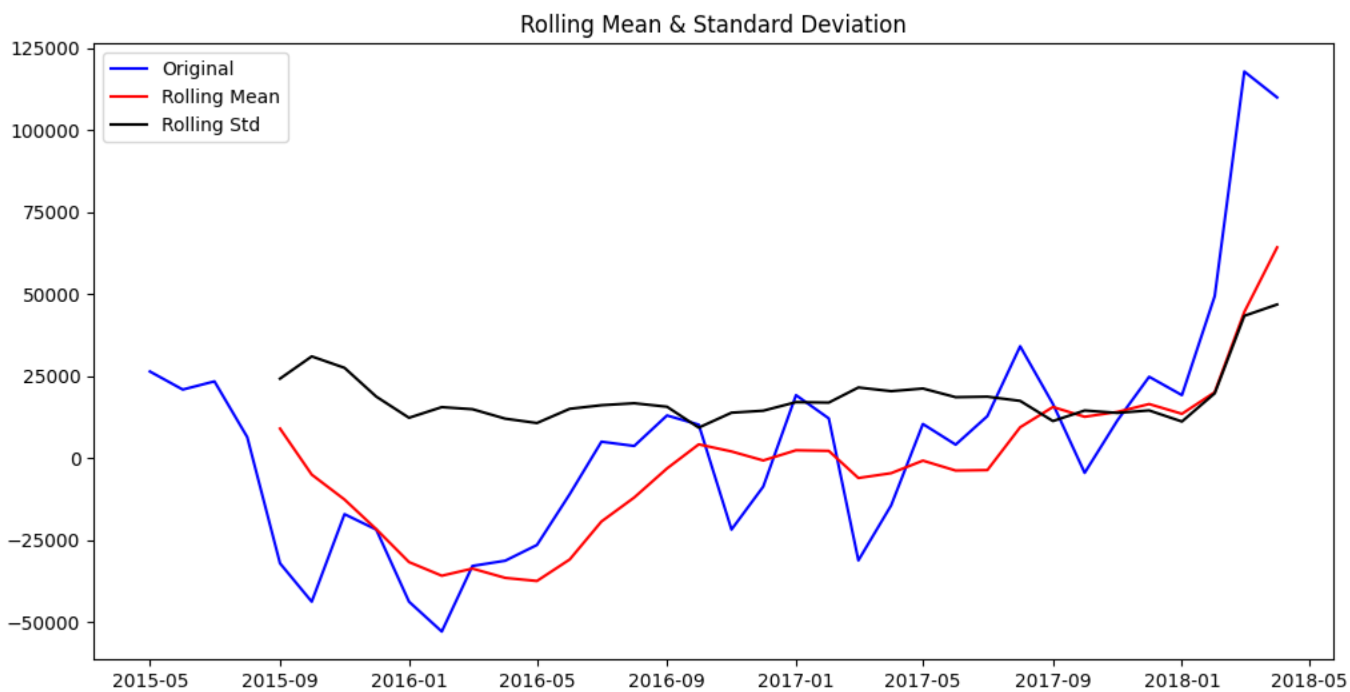




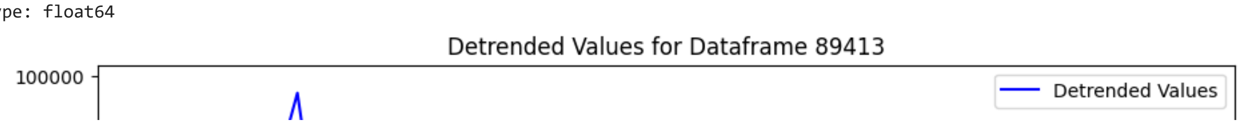

```
Results of Dickey-Fuller Test:
```

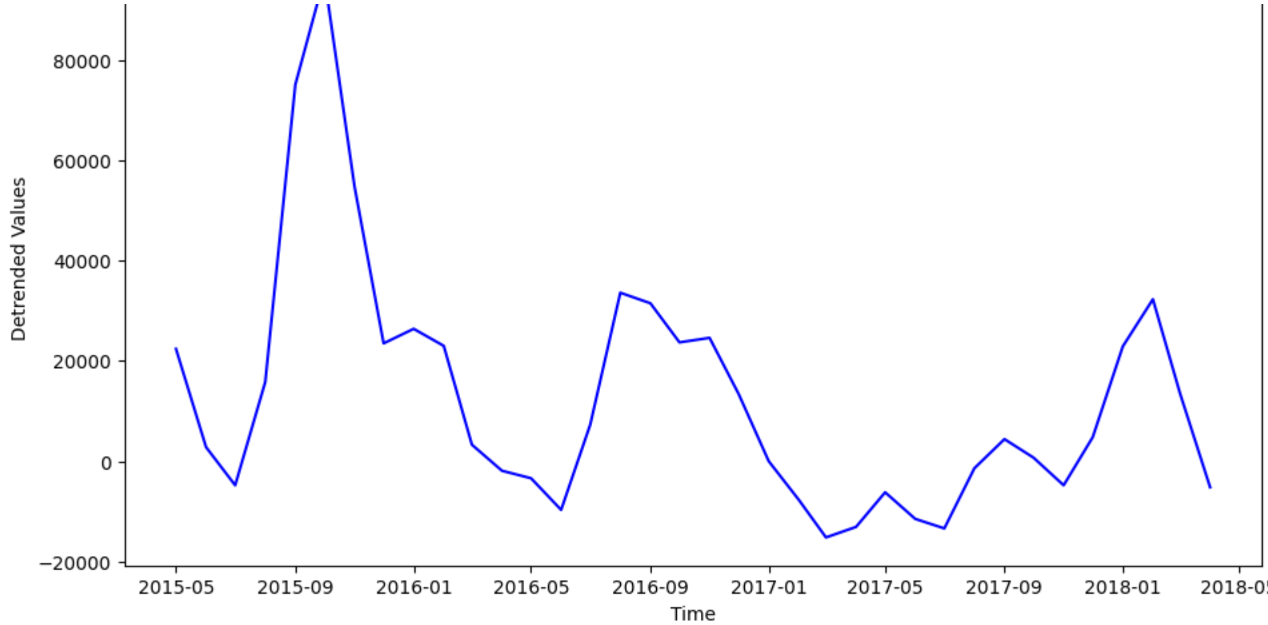
```
Test Statistic                 -4.431406
p-value                         0.000261
#Lags Used                      1.000000
Number of Observations Used    34.000000
Critical Value (1%)            -3.639224
Critical Value (5%)            -2.951230
Critical Value (10%)           -2.614447
dtype: float64
```

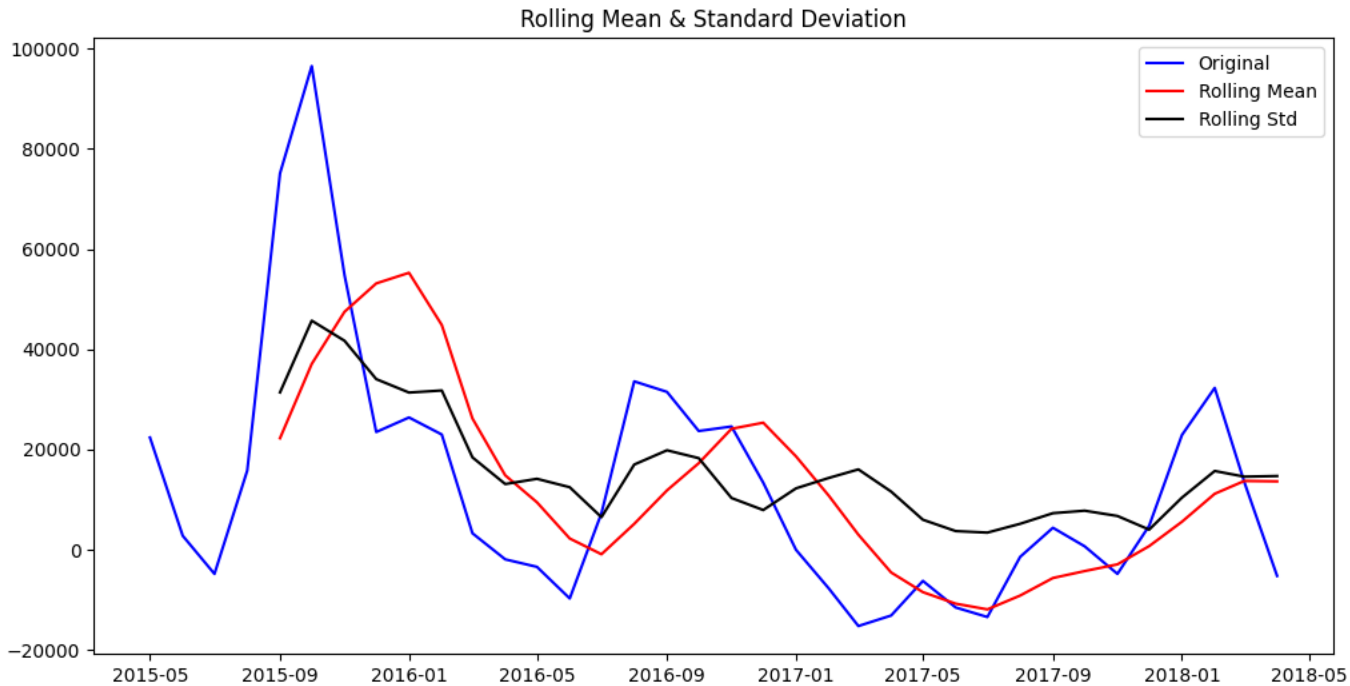




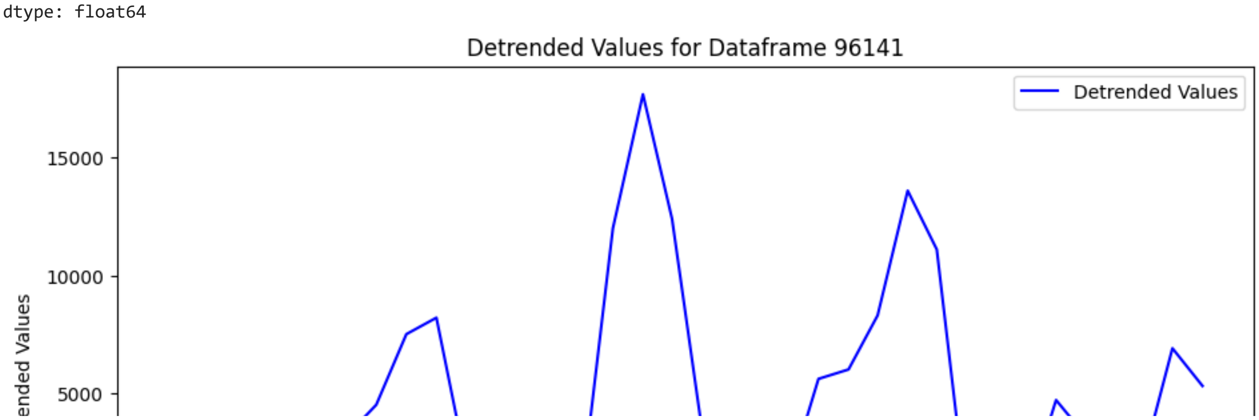

```
Results of Dickey-Fuller Test:

Test Statistic                  0.605864
p-value                         0.987759
#Lags Used                      5.000000
Number of Observations Used    30.000000
Critical Value (1%)            -3.669920
Critical Value (5%)            -2.964071
Critical Value (10%)           -2.621171
dtype: float64
```








```
Results of Dickey-Fuller Test:

Test Statistic                 -3.585082
p-value                         0.006053
#Lags Used                      5.000000
Number of Observations Used    30.000000
Critical Value (1%)            -3.669920
Critical Value (5%)            -2.964071
Critical Value (10%)           -2.621171
dtype: float64
```




```
#Displaying the detrended dataframe keys
detrended_dataframes.keys()
```

Show hidden output

```
# Iterating over the items in the detrended_dataframes dictionary
for dataframe_name, detrended_dataframe in detrended_dataframes.items():
    print(f"Dataframe Name: {dataframe_name}")
    print(detrended_dataframe.head())  # Print the head of the detrended dataframe
print()
```

```
Dataframe Name: 10011
               value
time
2015-05-01   -3000.0
2015-06-01  105400.0
2015-07-01  173200.0
2015-08-01  283100.0
2015-09-01  211500.0
Dataframe Name: 10021
               value
time
2015-05-01   15500.0
2015-06-01  489700.0
2015-07-01  626000.0
2015-08-01  190600.0
2015-09-01 -112700.0
Dataframe Name: 31561
              value
time
2015-05-01  28900.0
2015-06-01  28400.0
2015-07-01  19300.0
2015-08-01    700.0
2015-09-01  11500.0
Dataframe Name: 81611
              value
time
2015-05-01  45000.0
2015-06-01  12800.0
2015-07-01  14600.0
2015-08-01  29900.0
2015-09-01  14800.0
Dataframe Name: 34102
              value
time
2015-05-01  20700.0
2015-06-01  19900.0
2015-07-01   7600.0
2015-08-01  11900.0
2015-09-01  17800.0
Dataframe Name: 11975
              value
time
2015-05-01  27900.0
2015-06-01  23200.0
2015-07-01  19300.0
2015-08-01  23900.0
2015-09-01  37300.0
Dataframe Name: 81615
              value
time
2015-05-01  26400.0
2015-06-01  20900.0
2015-07-01  23400.0
2015-08-01   6400.0
2015-09-01 -32100.0
Dataframe Name: 89413
              value
```

```
# Accessing the separate dataframes for each region name
for region_name, region_df in region_dataframes.items():
    print(f"Dataframe name: {region_name}")
    print(region_df.head())
    print()
```

```
Dataframe name: 10011
                 value
time
2015-04-01  10572500.0
2015-05-01  10569500.0
2015-06-01  10674900.0
2015-07-01  10848100.0
2015-08-01  11131200.0

Dataframe name: 10014
                value
time
2015-04-01  9938600.0
2015-05-01  9827500.0
2015-06-01  9571200.0
2015-07-01  9278700.0
2015-08-01  9252000.0

Dataframe name: 10021
                 value
time
2015-04-01  16644000.0
2015-05-01  16659500.0
2015-06-01  17149200.0
2015-07-01  17775200.0
2015-08-01  17965800.0

Dataframe name: 11975
                value
time
2015-04-01  2343300.0
2015-05-01  2371200.0
2015-06-01  2394400.0
2015-07-01  2413700.0
2015-08-01  2437600.0

Dataframe name: 31561
                value
time
2015-04-01  2453900.0
2015-05-01  2482800.0
2015-06-01  2511200.0
2015-07-01  2530500.0
2015-08-01  2531200.0

Dataframe name: 34102
                value
time
2015-04-01  2481500.0
2015-05-01  2502200.0
2015-06-01  2522100.0
2015-07-01  2529700.0
2015-08-01  2541600.0

Dataframe name: 81611
                value
time
2015-04-01  3956500.0
```

Performing Deseasonalizing of our dataset and storing in a new dictionary

```
#Deseasonalizing the dataset

deseasonalized_data = {}

# Iterate over each key (region) in the region_dataframes dictionary
for region_name, data_diff in detrended_dataframes.items():
    # Time Series Decomposition
    result_mul = seasonal_decompose(data_diff['value'], model='additive', extrapolate_trend='freq', period=int(len(data_diff) / 2))

    # Deseasonalize
    deseasonalized = data_diff['value'].values / result_mul.seasonal

    # Plot
```

```
    plt.figure()
    plt.plot(deseasonalized)
    plt.title(f'Housing Values Deseasonalised - {region_name}', fontsize=14)
    plt.xlabel('Years')
    plt.ylabel('Value')
    plt.xticks(rotation = 45)
    plt.show()

    # Store the deseasonalized values in the dictionary
    deseasonalized_data[region_name] = deseasonalized
```

Show hidden output

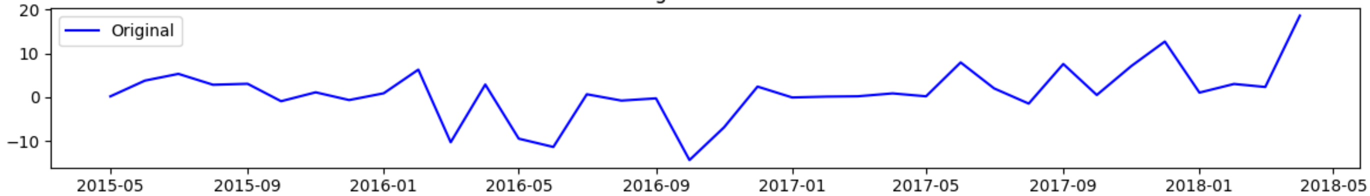
```
#Printing the deseasonalized data keys
deseasonalized_data.keys()
```

```
dict_keys([10011, 10021, 31561, 81611, 34102, 11975, 81615, 89413, 96141])
```

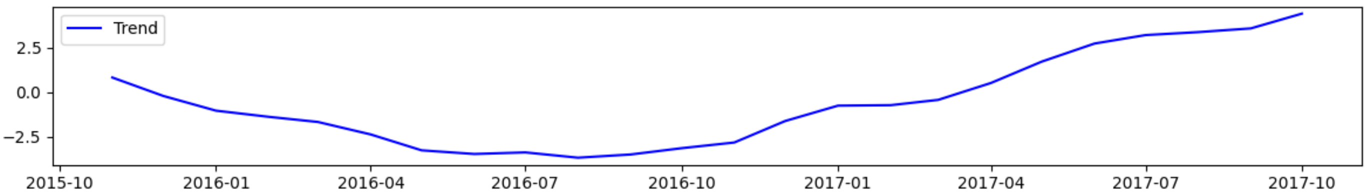
Performing Seasonal decomposition on the dictionary

```
# Iterate over each dataframe in the region_dataframes dictionary
for region_name, region_df in deseasonalized_data.items():
    # Perform seasonal decomposition
    decomposition = seasonal_decompose(region_df)

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

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

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

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

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

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

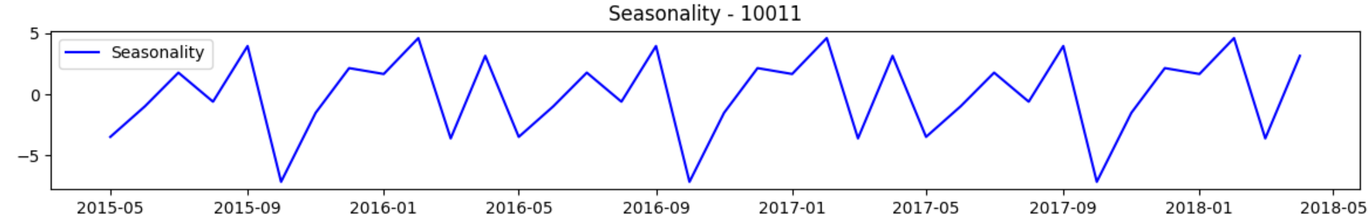
Original - 10011



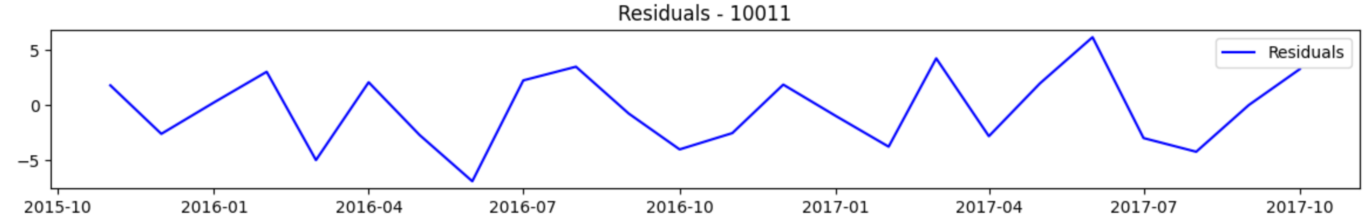
Trend - 10011



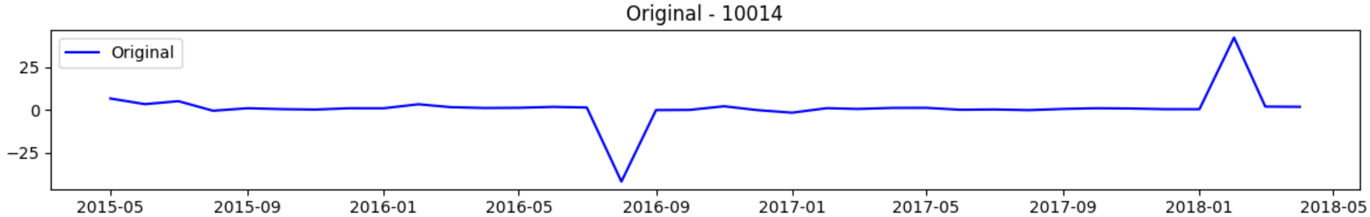
Seasonality - 10011



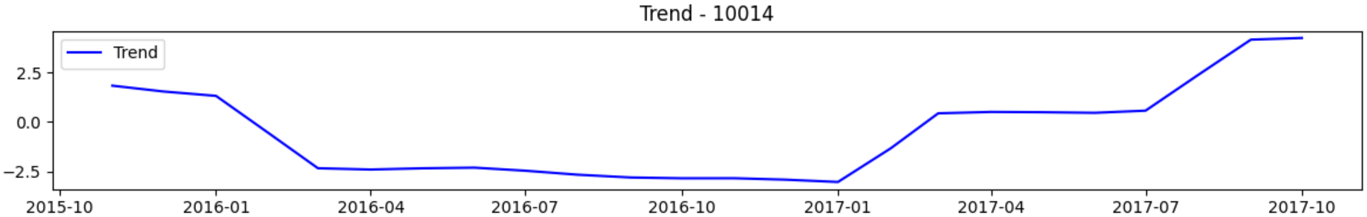
Residuals - 10011



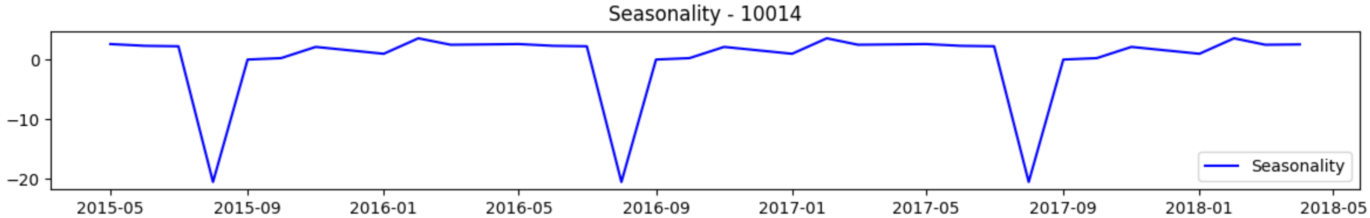
Original - 10014



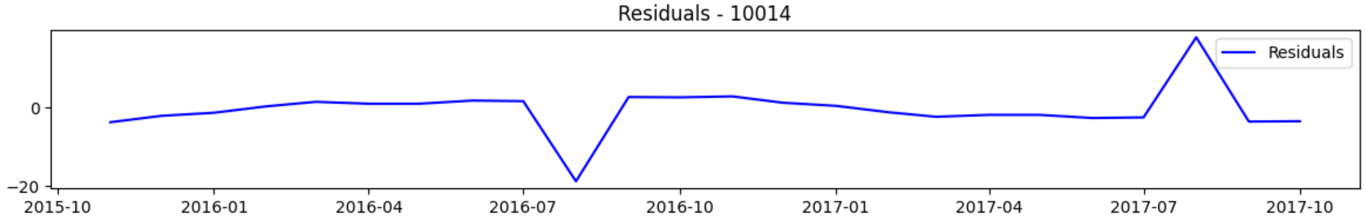
Trend - 10014



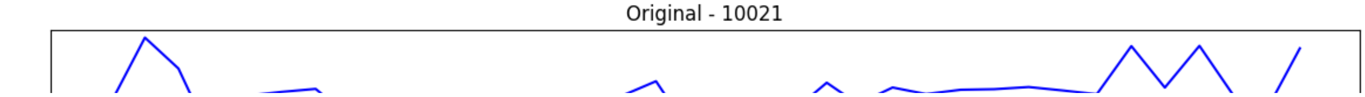
Seasonality - 10014

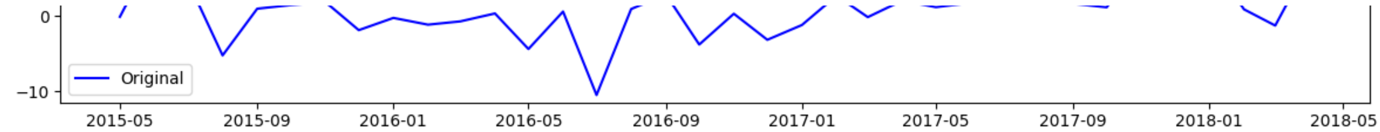


Residuals - 10014

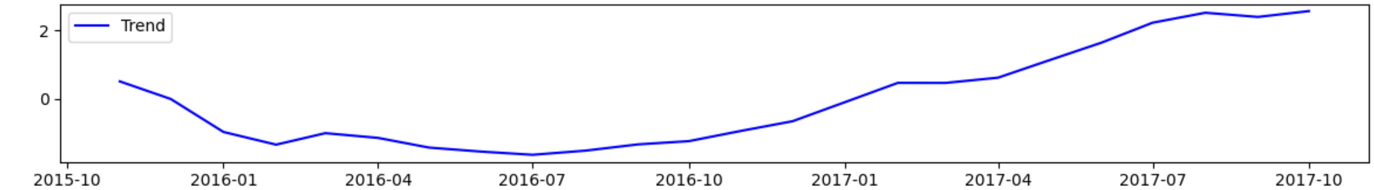


Original - 10021

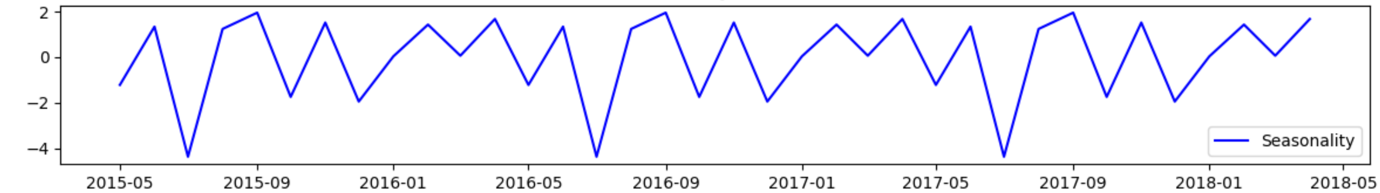




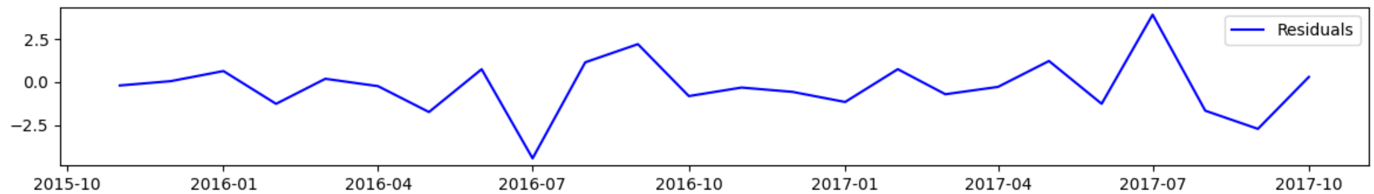
Trend - 10021



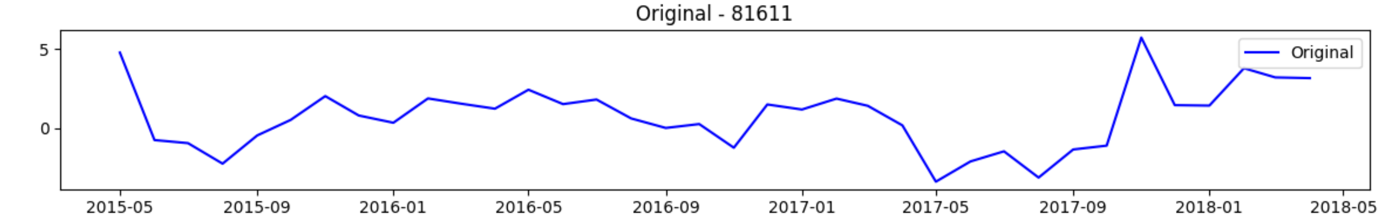
Seasonality - 10021



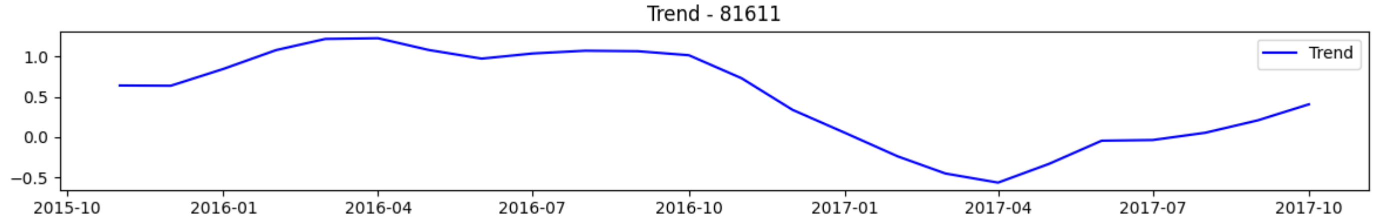
Residuals - 10021



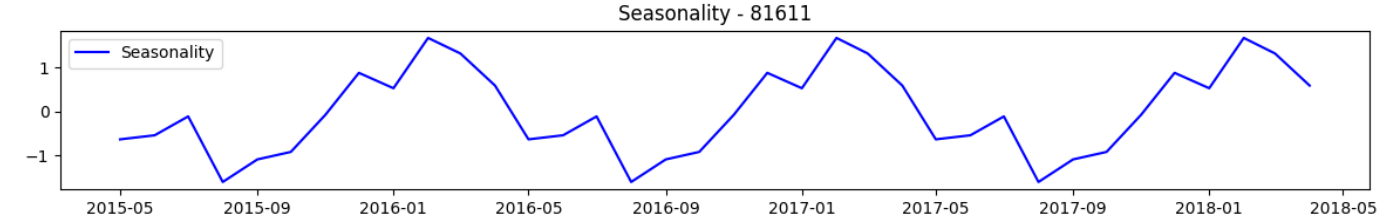
Original - 81611



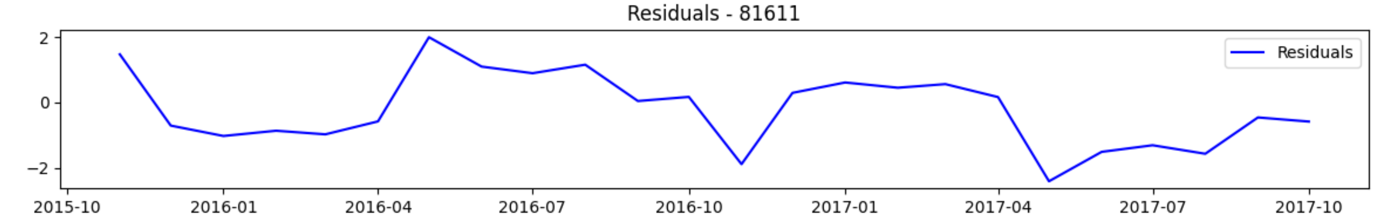
Trend - 81611



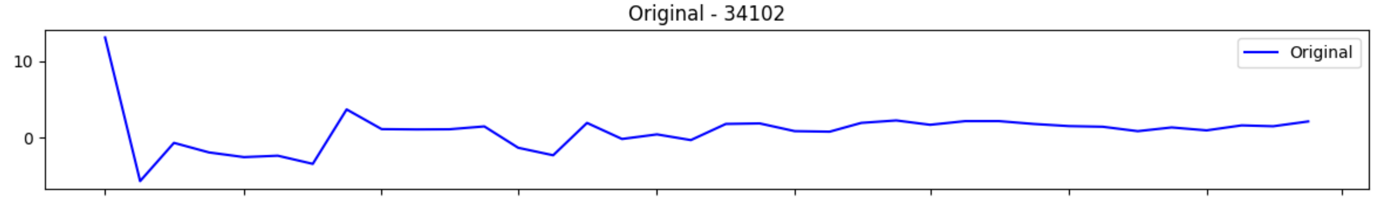
Seasonality - 81611



Residuals - 81611

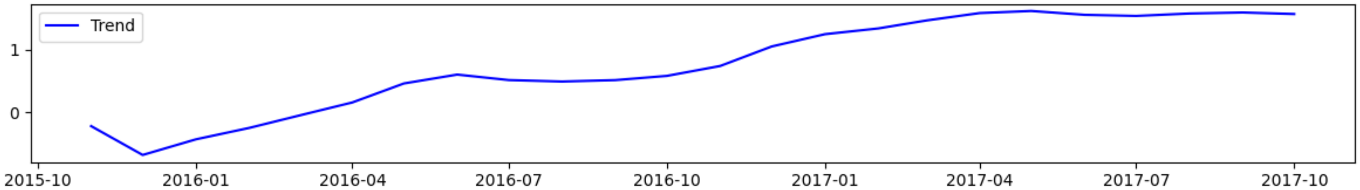


Original - 34102

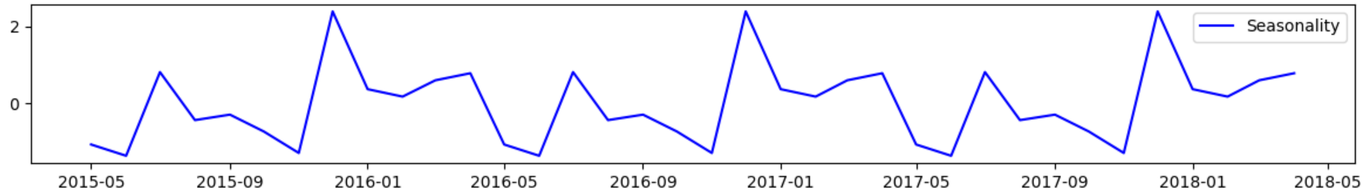


2015-05 2015-09 2016-01 2016-05 2016-09 2017-01 2017-05 2017-09 2018-01 2018-05

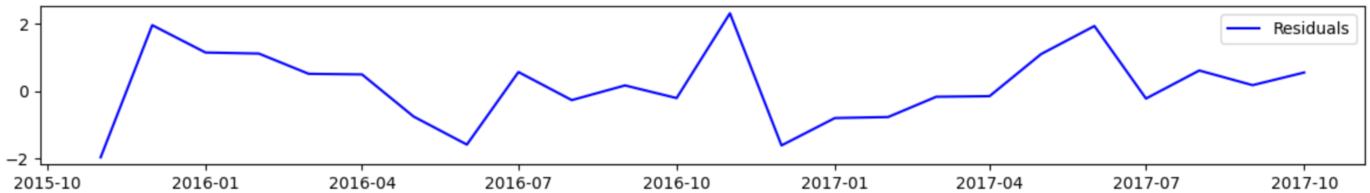
Trend - 34102



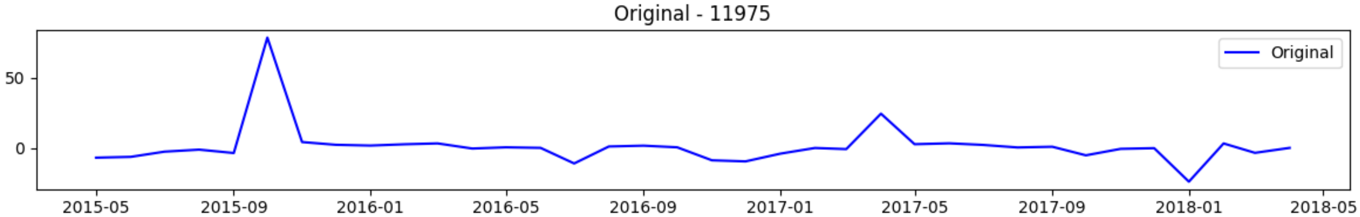
Seasonality - 34102



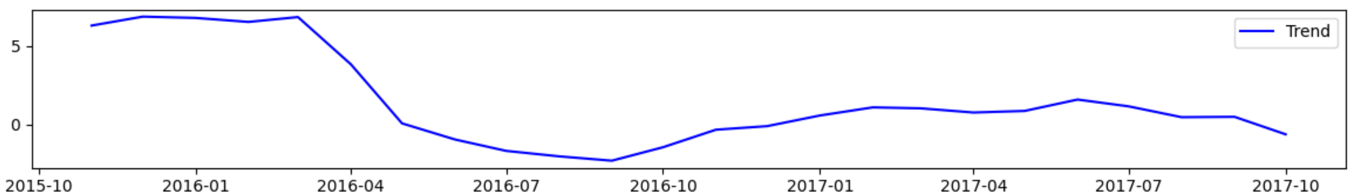
Residuals - 34102



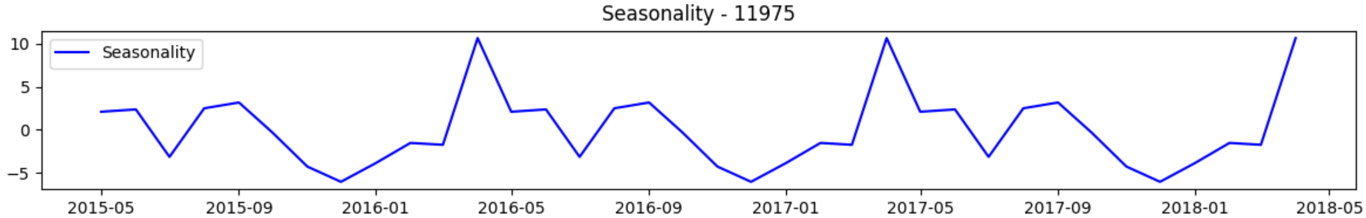
Original - 11975



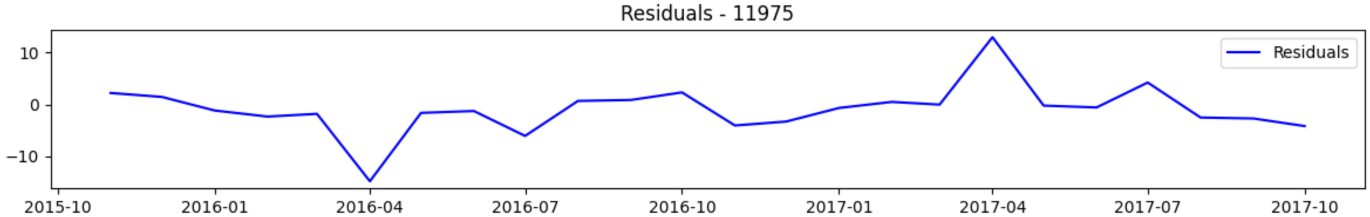
Trend - 11975



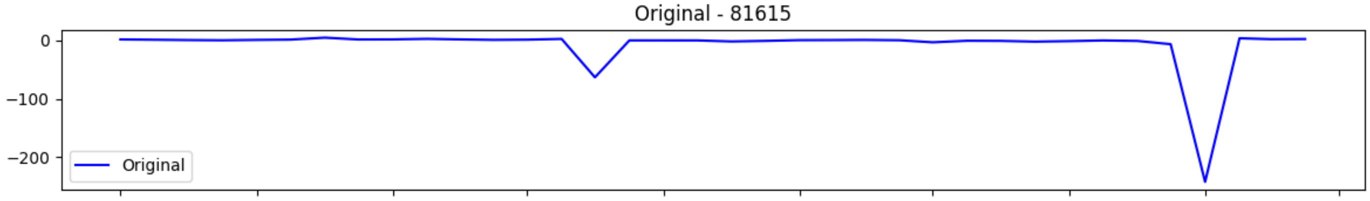
Seasonality - 11975



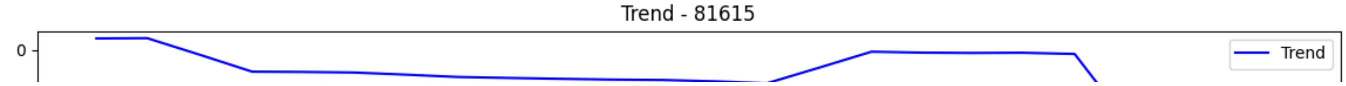
Residuals - 11975

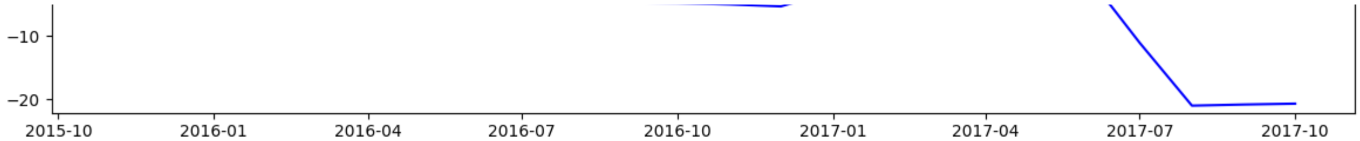


Original - 81615

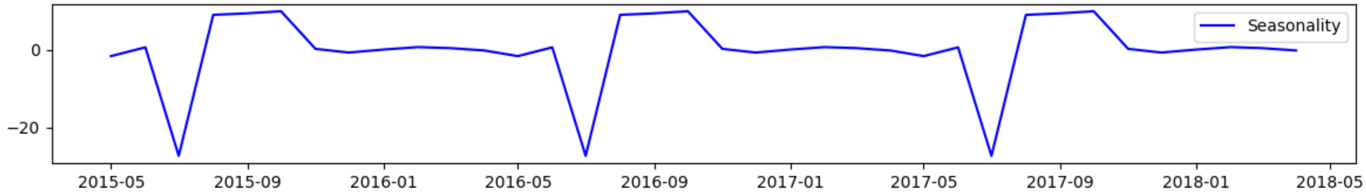


Trend - 81615

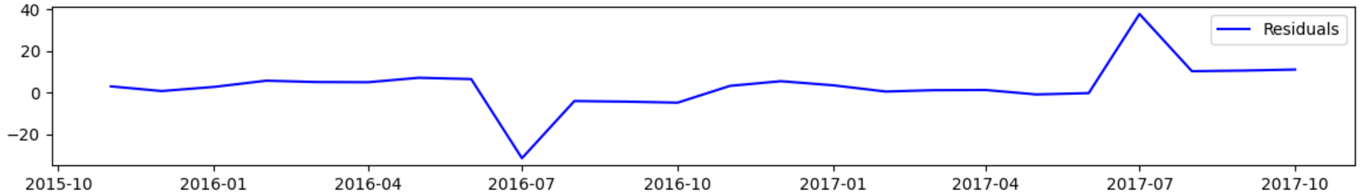




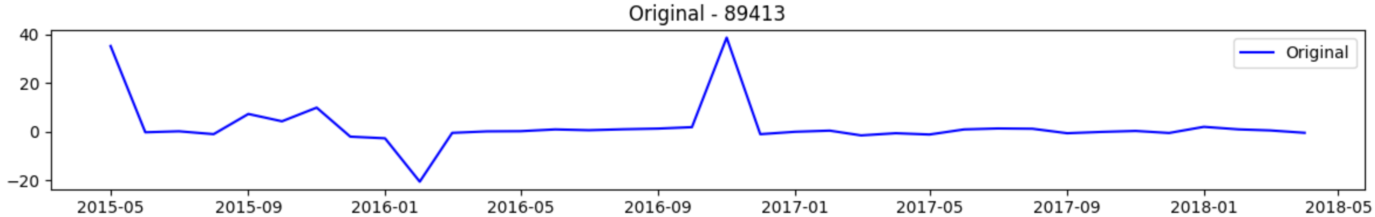
Seasonality - 81615



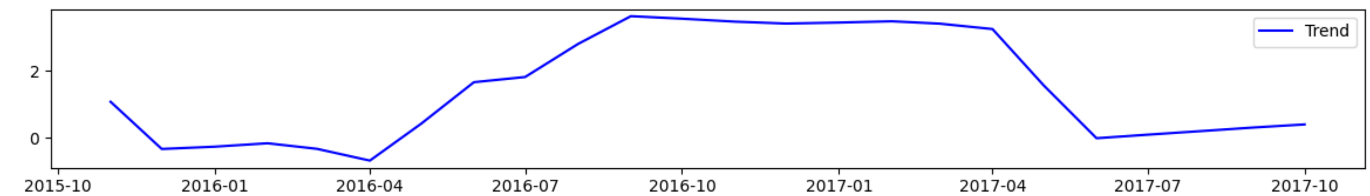
Residuals - 81615



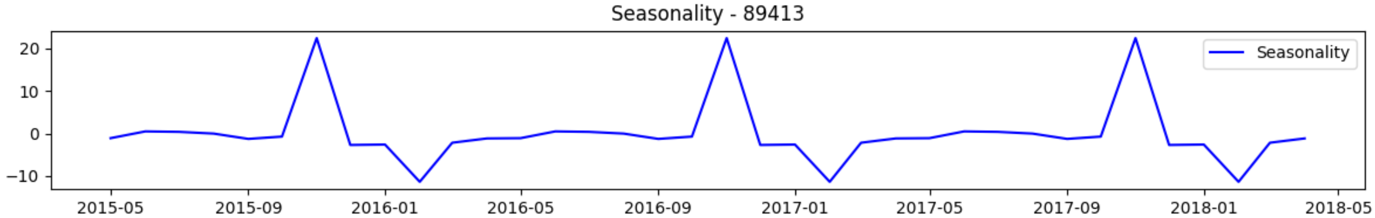
Original - 89413



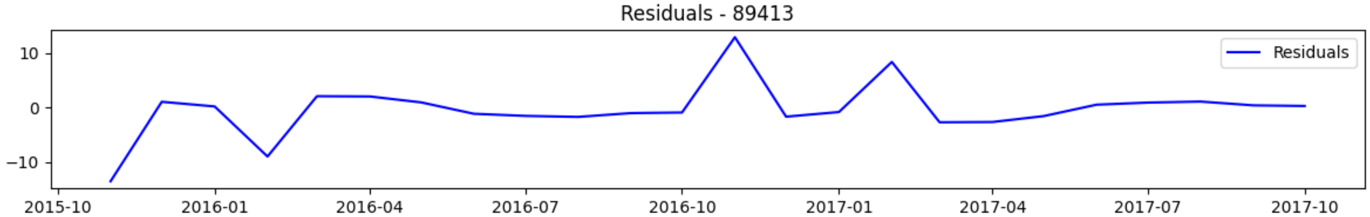
Trend - 89413



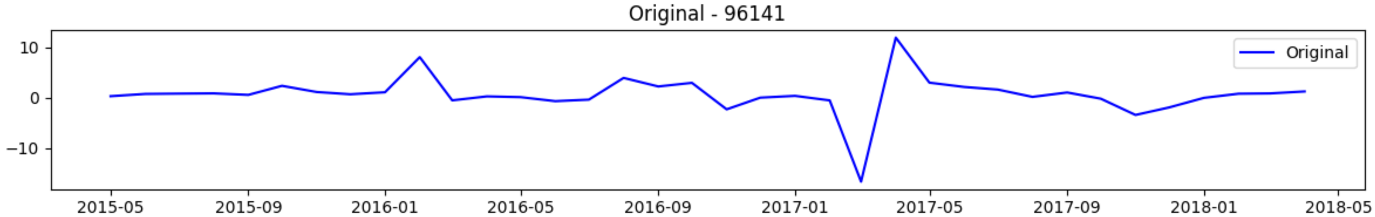
Seasonality - 89413



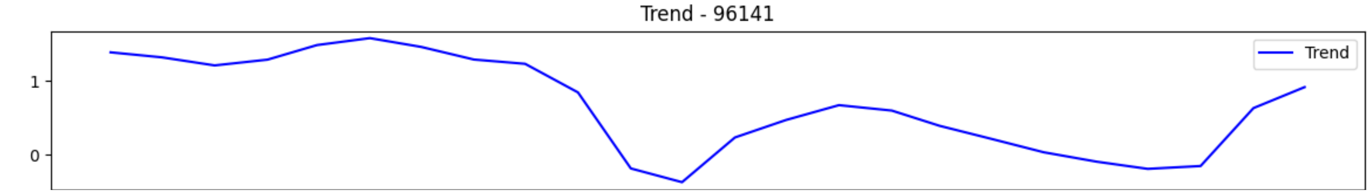
Residuals - 89413

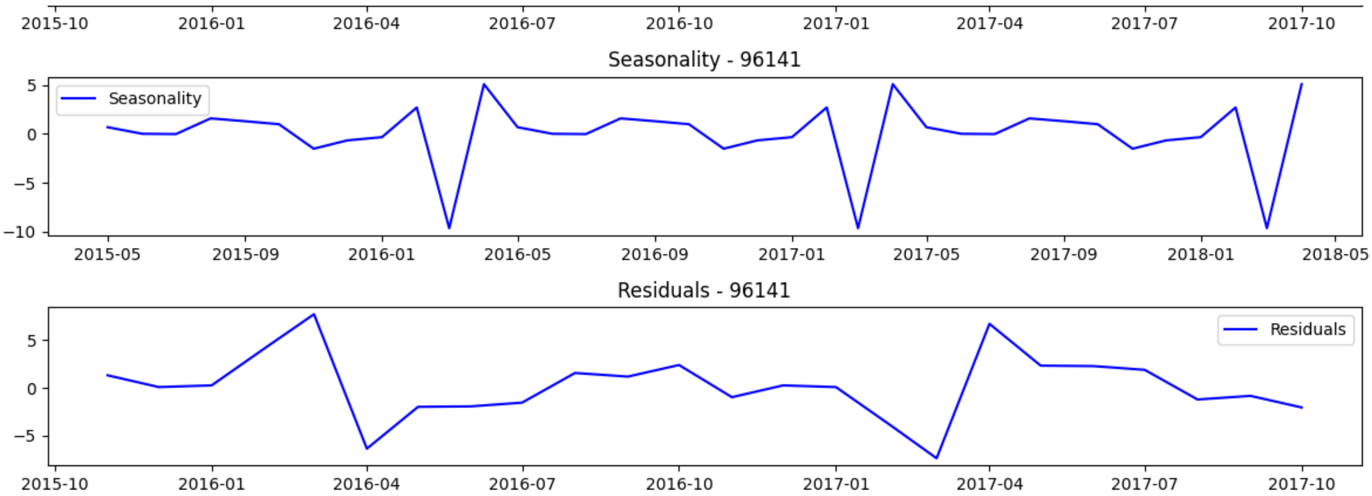


Original - 96141



Trend - 96141





Checking for stationarity of residuals using Seasonal Decompose

```
# Iterate over each dataframe in the region_dataframes dictionary
for region_name, region_df in deseasonalized_data.items():
    # Perform seasonal decomposition
    decomposition = seasonal_decompose(region_df)

    # Gather the residuals
    residual = decomposition.resid

    # Drop missing values from residuals
    ts_log_decompose = residual
    ts_log_decompose.dropna(inplace=True)

    # Perform the stationarity check
    print(f"Stationarity Check for {region_name}:")
    stationarity_check(ts_log_decompose)
    print()
```

Show hidden output

```
#Creating separate dictionaries for the individual zipnames
ts_10011 = deseasonalized_data[10011]
ts_10014 = region_dataframes[10014]
ts_10021 = deseasonalized_data[10021]
ts_81611 = deseasonalized_data[81611]
ts_31561 = deseasonalized_data[31561]
ts_34102 = deseasonalized_data[34102]
ts_81611 = deseasonalized_data[81611]
ts_81615 = deseasonalized_data[81615]
ts_89413 = deseasonalized_data[89413]
ts_96141 = deseasonalized_data[96141]
```

PLotting for the seasonal decomposition of the stationary Region Name 10014

```
    # Perform seasonal decomposition
    decomposition = seasonal_decompose(ts_10014)

    # Gather the residuals
```