student

June 20, 2021

Deducing Investment Opportunity for a Real Estate Investment Company

By: Tamjid Ahsan

As phase 4 project of Flatiron Data Science Bootcamp. * Student pace: Full Time * Scheduled project review date/time: June 24, 2021, 05:00 PM [DST] * Instructor name: James Irving _____

OVERVIEW

New York City is among the most expensive and competitive housing markets in the USA. It was impacted severely by the COVID-19 with high job loss. NYC is among the top impacted areas of the country. New York has been recovering from the economic impacts of the pandemic as of mid 2021. The strong buyer demand has also changed the dynamics of the residential real estate sales market that had been cooling for nearly three years.

NYC, however, is still a buyer's real estate market and buyers may have an opportunity to get some heavy discounts.

Many industry experts have been predicting a strong property appreciation in New York starting from 2021. 2021 is should be a great year for property owners. Different business sectors have been opening up in different ways and at differing speeds with relaxing COVID-19 policies. The current trends show that the New York housing market will be hyperactive in the peak home-buying season.

Home prices are still low compared to where they were last year, just before the pandemic hit New York City. Most buyers aren't paying sellers' asking prices. In April 2021, the New York real estate market (statewide) showed strong sales due to pent-up buyer demand, according to the most recent housing report released by the New York State Association of REALTORS®. Closed and pending sales remained strong in April of 2021, marking the eighth consecutive month of sales growth in year-over-year comparisons. Since 2012, the NYC home values have appreciated by nearly 52% as per Zillow Home Value Index.

This makes New York as one of the best real estate market for homes to get into as the house prices are relatively low, high buyer power and huge inventory of homes for sale to choose from and a projected uptrend in price leading to higher return on investment.

Ref: Norada Real Estate, NY Post.

BUSINESS PROBLEM

XYZ, Inc. LLC is a (read: fictional) private equity investment company based on Queens, New York. They want to invest in the housing market for relatively short term, three years. They want to isolate and invest in properties with the highest return on investment potential based

on geographical location close to their operation base in Queens, as they want to cluster their investment based on location. For this analysis, all 55 zipcodes of Queens county of New York city, NY were considered.

This analysis will recommend top five zipcodes with with return on investment potential with some insights, which will aid the top management of the company to make an educated decision on where to invest.

Source: image generated by author using plotly, and online gif maker.

Methodology

- Zillow House Value dataset is used. (more info on OBTAIN section)
- Data Science Process of the O.S.E.M.N. framework is adapted for this analysis
- Several analysis techniques were used such as conventional time series method such as `ARIMA` and `SARIMAX` by `statsmodels` on all zipcodes.
- not including `white nosie` or `random walk models`.
- forecasting procedure implemented by Facebook, Inc. named `Prophet` for a handful of zipcodes, can be found in APPENDIX.
- Implementation of recurrent neural network (RNN - LTMS and GRU) and transfer learning (combining `SARIMAX` and `RNN`) is a work in progress.

IMPORTS

- custom functions are used, can be found in `./imports_and_functions/functions.py`
- most of the imports and notebook formatting used in this analysis is in `./imports_and_functions/packages.py`
- those are also available in the APPENDIX section.

[1]:
```
%load_ext autoreload
%autoreload 2
```

[1]:
```
import imports_and_functions as fn
from imports_and_functions.packages import *

%matplotlib inline
mpl.rcParams['figure.facecolor'] = '#232323'
```

1 OBTAIN

- Main dataset:
  - Zillow Home Value Index (ZHVI): A smoothed, seasonally adjusted measure of the typical home value and market changes across a given region and housing type. It reflects the typical value for homes in the 35th to 65th percentile range. This data is used for the Time Series analysis obtained form Zillow Research. This data is separated by zipcode.

A copy of that file renamed as `zillow_raw_2021.csv` can be found here. Explanation of methodology can be found here.

- GeoJson:
  - GeoJson file used to generate map is sourced from here provided by Open Data Delaware. A copy of that can be found at `./data/ny_new_york_zip_codes_geo.min.json` in this repository.
- Zipcodes with Neighborhood information
- This file was obtained from here. A copy of this can be found at `./data/nyc-zip-codes.txt` in this repository.

1.1 Zillow Dataset information

[2]:
```
# leading data
df = pd.read_csv('./data/zillow_raw_2021.csv')
```

[3]:
```
df.dtypes
```

[3]:
```
RegionID        int64
SizeRank        int64
RegionName      int64
RegionType     object
StateName      object
               …
2020-12-31    float64
2021-01-31    float64
2021-02-28    float64
2021-03-31    float64
2021-04-30    float64
Length: 313, dtype: object
```

[4]:
```
# display dataset
df
```

[4]:

	RegionID	SizeRank	RegionName	…	2021-02-28	2021-03-31	2021-04-30
0	61639	0	10025	…	1092196.0	1104487.0	1121360.0
1	84654	1	60657	…	515983.0	517482.0	519569.0
2	61637	2	10023	…	1120821.0	1118048.0	1120428.0
3	91982	3	77494	…	365462.0	370763.0	376914.0
4	84616	4	60614	…	656818.0	659416.0	662782.0
…	…	…	…	…	…	…	
30837	87060	34430	66045	…	225272.0	229287.0	233174.0
30838	90986	34430	75599	…	49377.0	49610.0	50222.0
30839	58379	34430	1470	…	399243.0	398659.0	401526.0
30840	58117	35187	822	…	171867.0	171381.0	172291.0
30841	58110	35187	801	…	38316.0	37639.0	36916.0

[30842 rows x 313 columns]

[5]:
```
df.columns
```

[5]:
```
Index(['RegionID', 'SizeRank', 'RegionName', 'RegionType', 'StateName',
       'State', 'City', 'Metro', 'CountyName', '1996-01-31',
       …
       '2020-07-31', '2020-08-31', '2020-09-30', '2020-10-31', '2020-11-30',
       '2020-12-31', '2021-01-31', '2021-02-28', '2021-03-31', '2021-04-30'],
      dtype='object', length=313)
```

[6]:
```
# information
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30842 entries, 0 to 30841
Columns: 313 entries, RegionID to 2021-04-30
dtypes: float64(304), int64(3), object(6)
memory usage: 73.7+ MB
```

[6]:

	RegionID	SizeRank	RegionName	…	2021-02-28	2021-03-31	2021-04-30
0	61639	0	10025	…	1092196.0	1104487.0	1121360.0
1	84654	1	60657	…	515983.0	517482.0	519569.0
2	61637	2	10023	…	1120821.0	1118048.0	1120428.0
3	91982	3	77494	…	365462.0	370763.0	376914.0
4	84616	4	60614	…	656818.0	659416.0	662782.0

[5 rows x 313 columns]

[7]:
```
# no duplicate zipcodes
df.RegionName.value_counts().value_counts()
```

[7]:
```
1    30842
Name: RegionName, dtype: int64
```

Column explainer:

Column Name	Expaination	Range
RegionID	Unique Region Identifyer	from 58001 to 753844
SizeRank	Ranked by Population	from 0 to 35187
RegionName	Zipcode	30842 unique values
RegionType	Type of location	constant value of “Zip”
StateName	Name of State	51 unique values
State	Name of State	51 unique values
City	City name	15005 unique values
Metro	Metromoliton area	862 unique values

Column Name	Expaination	Range
CountyName	Name of county	1758 unique values
Rest of the colums	dates	from Jan 31, 1996 to Apr 30, 2021

- steps are not shown to find out those values as this dataset will be sliced to focus only on Queens County, NY as per the goal of this analysis.

2 SCRUB & EXPLORE

2.1 Selecting Queens, New York data

[8]:
```
## selecting NY zipcodes
df_ny = df.loc[(df['State'] == 'NY')
               & (df['CountyName'] == 'Queens County')].reset_index()
df_ny.drop(['index'], axis=1, inplace=True)
df_ny
```

[8]:

	RegionID	SizeRank	RegionName	…	2021-02-28	2021-03-31	2021-04-30
0	62087	21	11375	…	421097.0	416426.0	411762.0
1	62088	110	11377	…	434951.0	433844.0	435544.0
2	62067	122	11355	…	466919.0	465487.0	464428.0
3	62093	167	11385	…	703008.0	704874.0	707296.0
4	62085	184	11373	…	413289.0	415127.0	416809.0
5	62084	279	11372	…	390607.0	388914.0	389908.0
6	62004	393	11101	…	833444.0	827952.0	822370.0
7	62080	496	11368	…	406164.0	404570.0	403652.0
8	62066	581	11354	…	416733.0	414523.0	411965.0
9	62086	732	11374	…	414567.0	413708.0	412868.0
10	62120	858	11434	…	415070.0	414574.0	414619.0
11	62179	947	11691	…	556678.0	558419.0	560278.0
12	62118	1271	11432	…	652898.0	653655.0	655697.0
13	62121	1713	11435	…	376961.0	378580.0	378927.0
14	62006	2121	11103	…	752601.0	751708.0	751877.0
15	62077	2154	11365	…	705044.0	705832.0	706928.0
16	62079	2222	11367	…	417844.0	417757.0	418022.0
17	62069	2809	11357	…	820842.0	821812.0	822467.0
18	62076	3171	11364	…	510305.0	509562.0	508134.0
19	62070	3506	11358	…	897735.0	897941.0	897168.0
20	62007	3574	11104	…	466181.0	467021.0	464044.0
21	62090	4200	11379	…	763210.0	764520.0	766161.0
22	62099	4385	11413	…	575442.0	577542.0	581338.0
23	62106	4719	11420	…	600657.0	602680.0	604603.0
24	62100	4821	11414	…	506686.0	507868.0	509471.0
25	62098	5130	11412	…	575170.0	577937.0	581654.0

```
26      62073      5133       11361  …   795332.0   795355.0   795151.0
27      62089      5391       11378  …   737800.0   737990.0   738171.0
28      62107      5414       11421  …   606259.0   607544.0   607973.0
29      62105      5538       11419  …   612966.0   616095.0   620214.0
30      62119      5589       11433  …   551655.0   554616.0   557119.0
31      62072      5676       11360  …   499535.0   497718.0   496283.0
32      62104      5756       11418  …   607392.0   609746.0   612122.0
33      62101      5769       11415  …   310914.0   304934.0   297538.0
34      62109      6259       11423  …   625767.0   630411.0   638006.0
35      62081      6436       11369  …   696911.0   701237.0   707477.0
36      62182      6529       11694  …   640819.0   644862.0   648065.0
37      62108      6570       11422  …   572342.0   571554.0   572027.0
38      62103      6841       11417  …   612457.0   615680.0   619167.0
39      62113      6886       11427  …   647574.0   651496.0   657234.0
40      62180      7098       11692  …   266741.0   272745.0   275388.0
41      62082      7324       11370  …   715316.0   715999.0   715961.0
42      62068      7633       11356  …   728028.0   730516.0   733839.0
43      62115      7895       11429  …   577540.0   580048.0   583710.0
44      62074      8062       11362  …   591694.0   591611.0   591892.0
45      62097      8120       11411  …   568897.0   568245.0   568945.0
46      62112      8198       11426  …   652363.0   655331.0   657969.0
47      62114      8554       11428  …   611535.0   616593.0   623049.0
48      62181      8812       11693  …   350591.0   355157.0   356097.0
49      61979      8912       11004  …   400391.0   405799.0   409080.0
50      62102      9277       11416  …   609687.0   612198.0   614847.0
51      62122      9306       11436  …   532509.0   534761.0   535380.0
52      62078      9616       11366  …   883587.0   882933.0   882687.0
53      62075     12001       11363  …   932673.0   931076.0   928875.0
54      62116     25178       11430  …   642995.0   641942.0   642980.0

[55 rows x 313 columns]
```

[9]:
```
# these colums are not needed anymore as they are not going to be used in the␣
 ↪analysis
df_ny.drop(columns=['RegionID', 'SizeRank', 'RegionType', 'StateName'],
           inplace=True)
```

2.2 wide format to long

[10]:
```
# the data is in wide format
# converting it to long format for analysis
df_ny_reshaped = fn.melt_data(df_ny)
df_ny_reshaped
```

[10]:
```
       RegionName State      City  …     CountyName       date     value
0           11375    NY  New York  …  Queens County 1996-01-31  129189.0
1           11377    NY  New York  …  Queens County 1996-01-31  111191.0
```

```
2           11355    NY  New York  …  Queens County 1996-01-31  120873.0
3           11385    NY  New York  …  Queens County 1996-01-31  215700.0
4           11373    NY  New York  …  Queens County 1996-01-31  114715.0
…             …   …     …   …          …         …       …
16715       11416    NY  New York  …  Queens County 2021-04-30  614847.0
16716       11436    NY  New York  …  Queens County 2021-04-30  535380.0
16717       11366    NY  New York  …  Queens County 2021-04-30  882687.0
16718       11363    NY  New York  …  Queens County 2021-04-30  928875.0
16719       11430    NY  New York  …  Queens County 2021-04-30  642980.0

[16720 rows x 7 columns]
```

[11]:
```
df_ny_reshaped.dtypes
```

[11]:
```
RegionName             int64
State                 object
City                  object
Metro                 object
CountyName            object
date          datetime64[ns]
value                float64
dtype: object
```

All column is recognized as they should be except `RegionName`. Those are int s that represent zipcode.

2.3 dtype casting

Casting 'RegionName' column as `str`

[12]:
```
df_ny_reshaped['RegionName'] = df_ny_reshaped['RegionName'].astype('str')
df_ny_reshaped.set_index('date', inplace=True)
```

[13]:
```
df_ny_reshaped.dtypes
```

[13]:
```
RegionName     object
State          object
City           object
Metro          object
CountyName     object
value         float64
dtype: object
```

[14]:
```
df_ny_reshaped
```

[14]:
```
           RegionName State  …     CountyName     value
date                         …
1996-01-31      11375    NY  …  Queens County  129189.0
```

```
1996-01-31      11377    NY  …  Queens County  111191.0
1996-01-31      11355    NY  …  Queens County  120873.0
1996-01-31      11385    NY  …  Queens County  215700.0
1996-01-31      11373    NY  …  Queens County  114715.0
…               …    …   …          …          …
2021-04-30      11416    NY  …  Queens County  614847.0
2021-04-30      11436    NY  …  Queens County  535380.0
2021-04-30      11366    NY  …  Queens County  882687.0
2021-04-30      11363    NY  …  Queens County  928875.0
2021-04-30      11430    NY  …  Queens County  642980.0

[16720 rows x 6 columns]
```

2.4 Transforming

For ease of use in analysis. Setting zipcodes as column head and not selecting any information rather than `date` as index and `value` as series.

[15]:
```
# list of all zipcodes
zipcode_list = df_ny_reshaped['RegionName'].unique().tolist()
print('Total zipcodes: ', len(zipcode_list))

# converting to dict
TS = {}
for zipcode in zipcode_list:
    temp_df = df_ny_reshaped.groupby('RegionName').get_group(
        zipcode).sort_index()['value']
    TS[zipcode] = temp_df

# sanity check
print('Keys in the dict: ', len(TS.keys()))
print('All zipcodes accounted for: ', len(zipcode_list) == len(TS.keys()))
```

```
Total zipcodes:  55
Keys in the dict:  55
All zipcodes accounted for:  True
```

[16]:
```
# converting to pandas.DataFrame
ts_df = pd.DataFrame(TS)
ts_df
```

[16]:

	11375	11377	11355	…	11366	11363	11430
date				…			
1996-01-31	129189.0	111191.0	120873.0	…	227326.0	346968.0	253175.0
1996-02-29	128476.0	110784.0	120431.0	…	227169.0	347413.0	252204.0
1996-03-31	128105.0	110823.0	120228.0	…	226699.0	348278.0	251766.0
1996-04-30	127360.0	110714.0	119757.0	…	225982.0	349253.0	251324.0
1996-05-31	127055.0	110794.0	119493.0	…	225593.0	350497.0	251464.0

```
…              …        …        …  …        …        …        …
2020-12-31  431259.0  432383.0  471009.0  …  883397.0  934384.0  642500.0
2021-01-31  427765.0  434379.0  469184.0  …  884268.0  934303.0  643663.0
2021-02-28  421097.0  434951.0  466919.0  …  883587.0  932673.0  642995.0
2021-03-31  416426.0  433844.0  465487.0  …  882933.0  931076.0  641942.0
2021-04-30  411762.0  435544.0  464428.0  …  882687.0  928875.0  642980.0

[304 rows x 55 columns]
```

2.5 setting frequency of the date

[17]:
```
# not recognized
print(ts_df.index.freq)
```

```
None
```

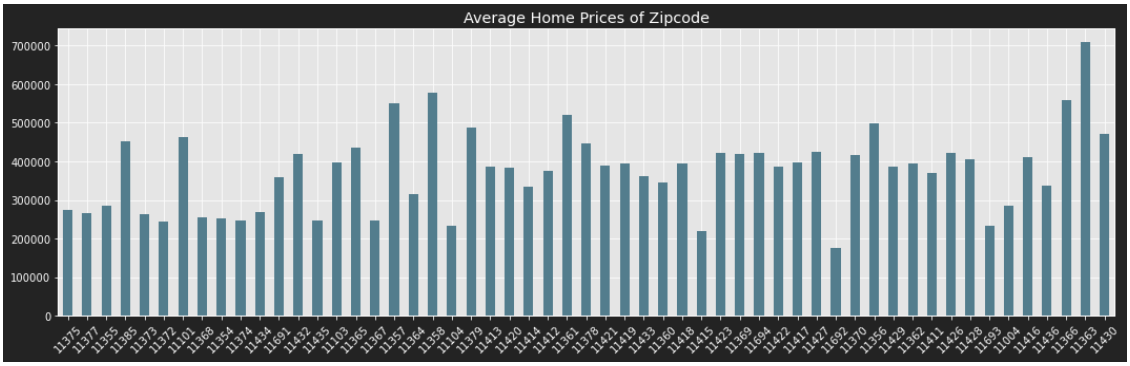
[18]:
```
# setting as month end
ts_df.index.freq = 'M'
ts_df.index.freq
```

[18]:
```
<MonthEnd>
```

2.6 EDA

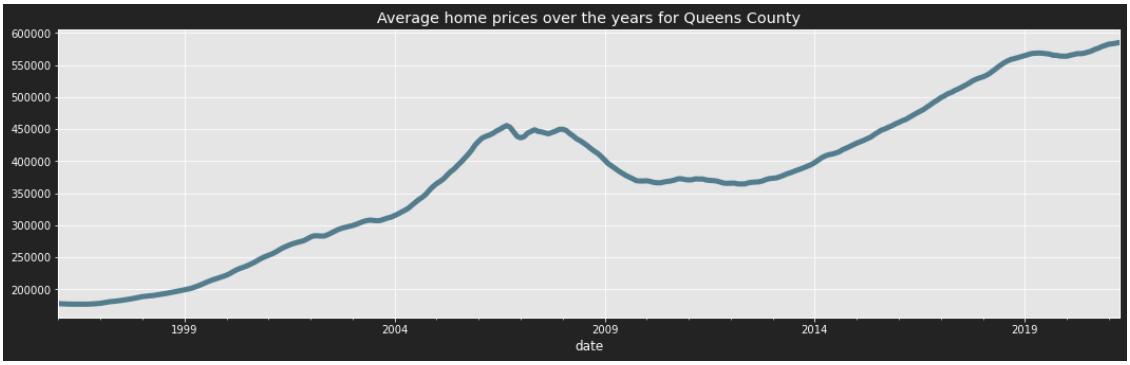
2.6.1 Average home price by zipcode

[19]:
```
ts_df.mean().plot(kind='bar',
                  figsize=(18, 5),
                  title='Average Home Prices of Zipcode',
                  color='#537d8d')
plt.tick_params('x', labelrotation=45)
```



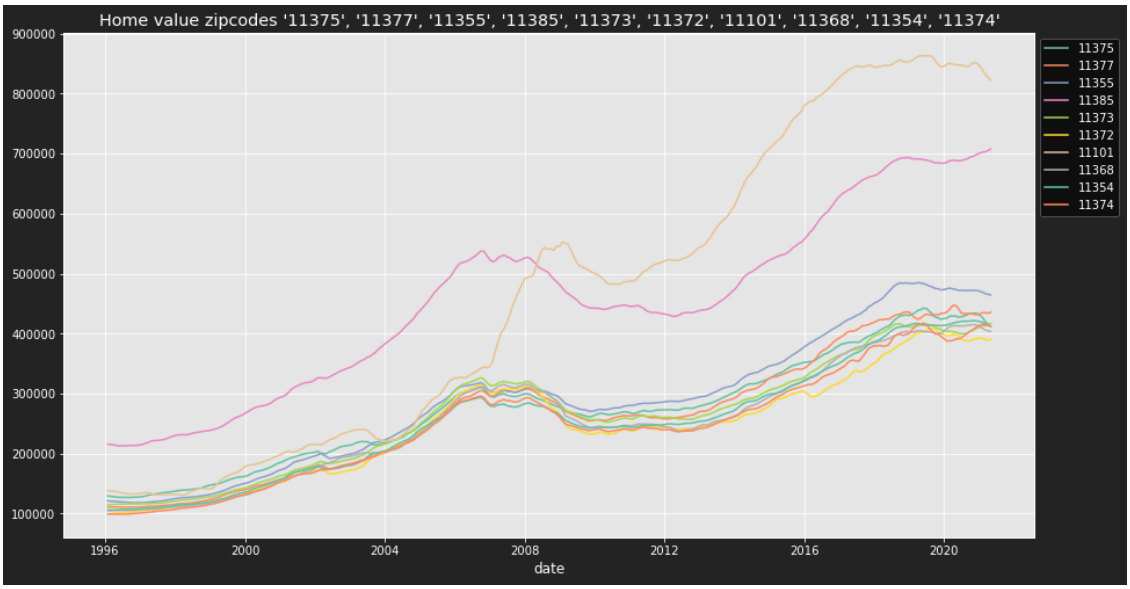
2.6.2 Average home price over the years

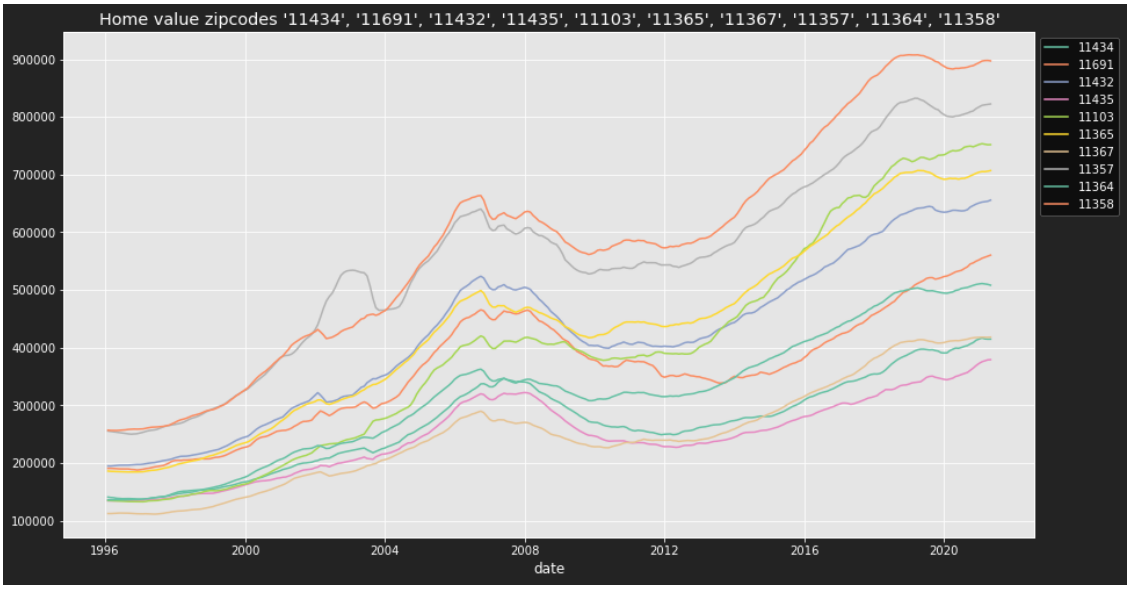
[20]:
```
ts_df.mean(1).plot(
    figsize=(18, 5),
    title='Average home prices over the years for Queens County',
    lw=5,
    color='#537d8d')
plt.show()
```

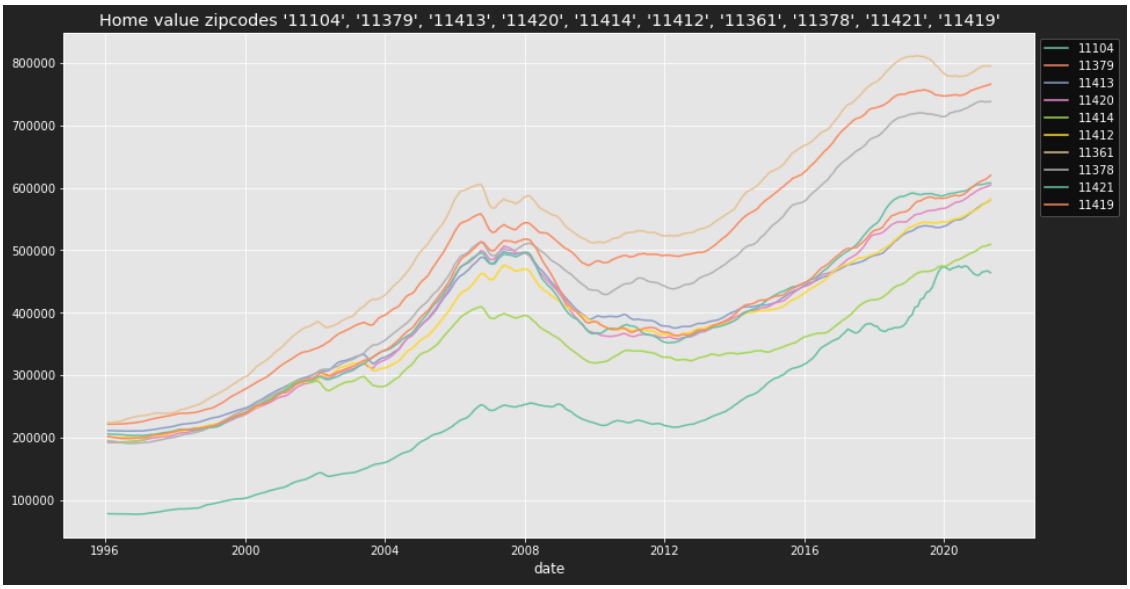


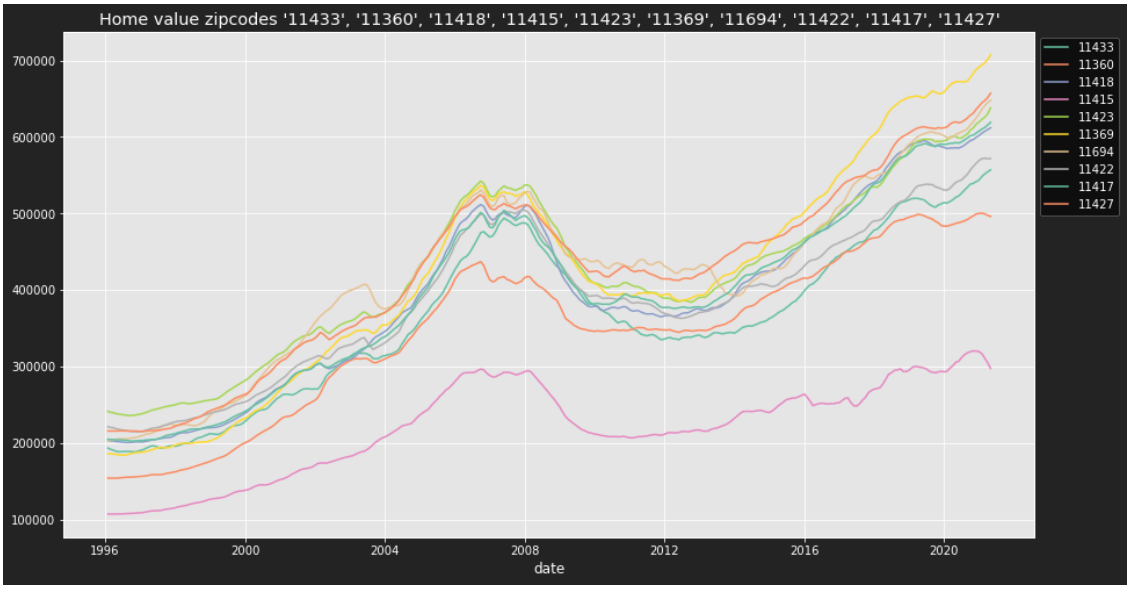
2.6.3 Time Series

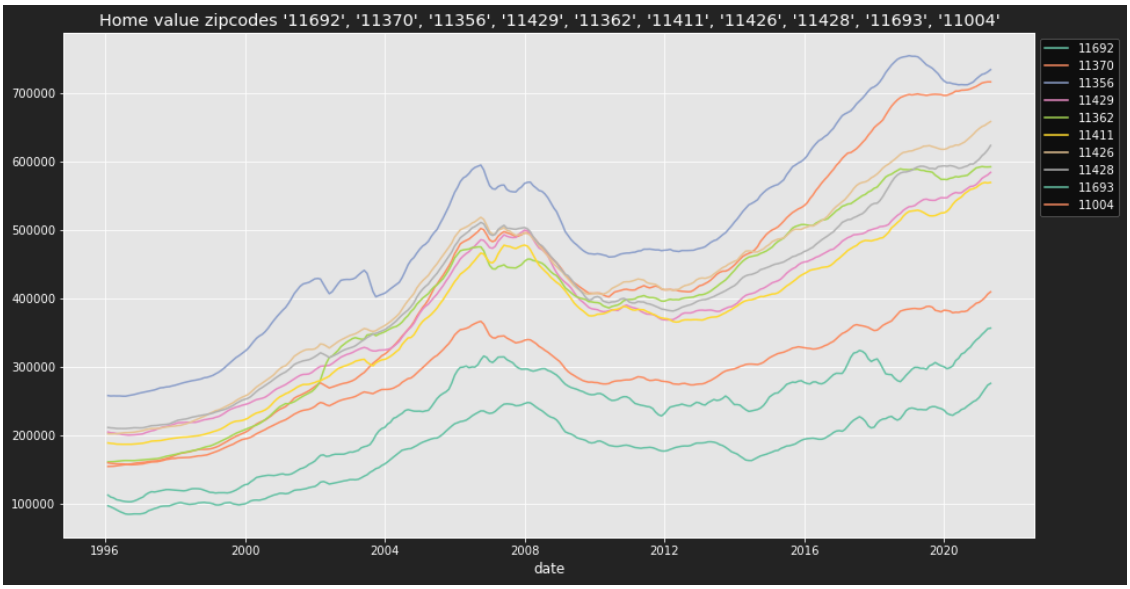
[21]:
```
n = 0
jump = 10
while n < len(ts_df.columns):
    slice_ = list(ts_df.columns[n:n + jump])
    fig, ax = plt.subplots(figsize=(15, 8))
    sns.lineplot(data=ts_df[slice_],
                 ax=ax,
                 palette='Set2',
                 estimator=None,
                 dashes=False)
    ax.set_title(f'Home value zipcodes {str(slice_)[1:-1]}')
    plt.legend(bbox_to_anchor=(1, 1), loc="upper left")
    n = n + jump
```

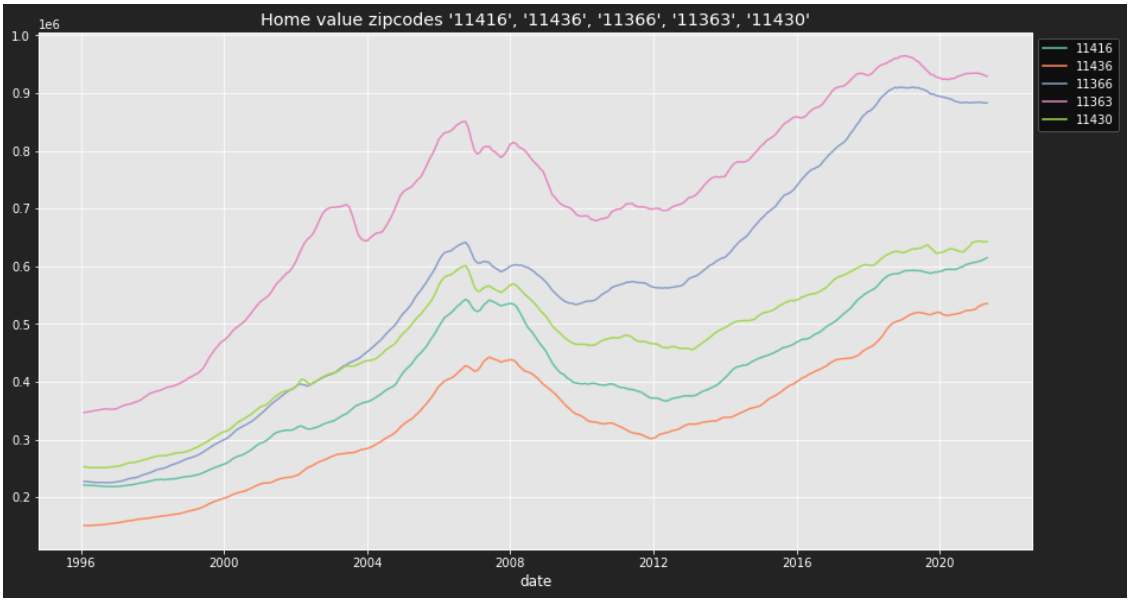

Home value zipcodes '11375', '11377', '11355', '11385', '11373', '11372', '11101', '11368', '11354', '11374'
900000
800000
700000
600000
500000
400000
300000
200000
100000
1996
2000
2004
2008
2012
2016
2020
date
11375
11377
11355
11385
11373
11372
11101
11368
11354
11374


Home value zipcodes '11434', '11691', '11432', '11435', '11103', '11365', '11367', '11357', '11364', '11358'
900000
800000
700000
600000
500000
400000
300000
200000
100000
1996
2000
2004
2008
2012
2016
2020
date
11434
11691
11432
11435
11103
11365
11367
11357
11364
11358


Home value zipcodes '11104', '11379', '11413', '11420', '11414', '11412', '11361', '11378', '11421', '11419'
800000
700000
600000
500000
400000
300000
200000
100000
1996
2000
2004
2008
2012
2016
2020
date
11104
11379
11413
11420
11414
11412
11361
11378
11421
11419


Home value zipcodes '11433', '11360', '11418', '11415', '11423', '11369', '11694', '11422', '11417', '11427'
700000
600000
500000
400000
300000
200000
100000
1996
2000
2004
2008
2012
2016
2020
date
11433
11360
11418
11415
11423
11369
11694
11422
11417
11427



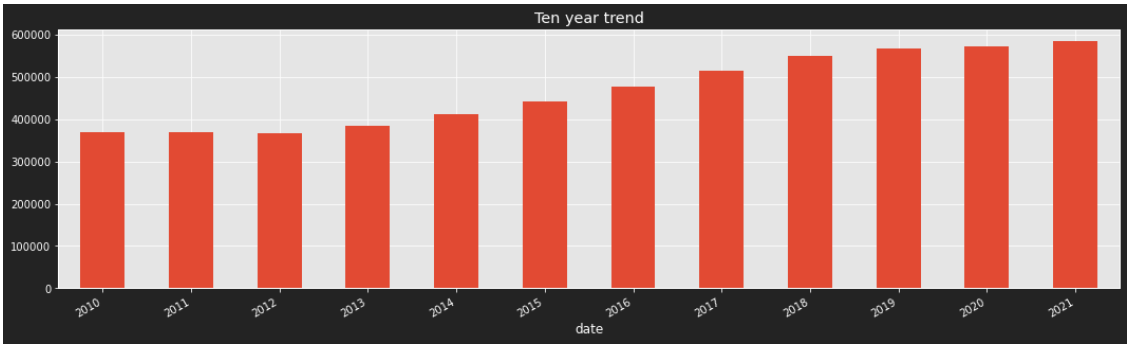


[22]:
```
# plotly line chart, its had to isolate individual time series
px.line(ts_df, template='plotly_dark')
```

comment

2.6.4 Recent trend

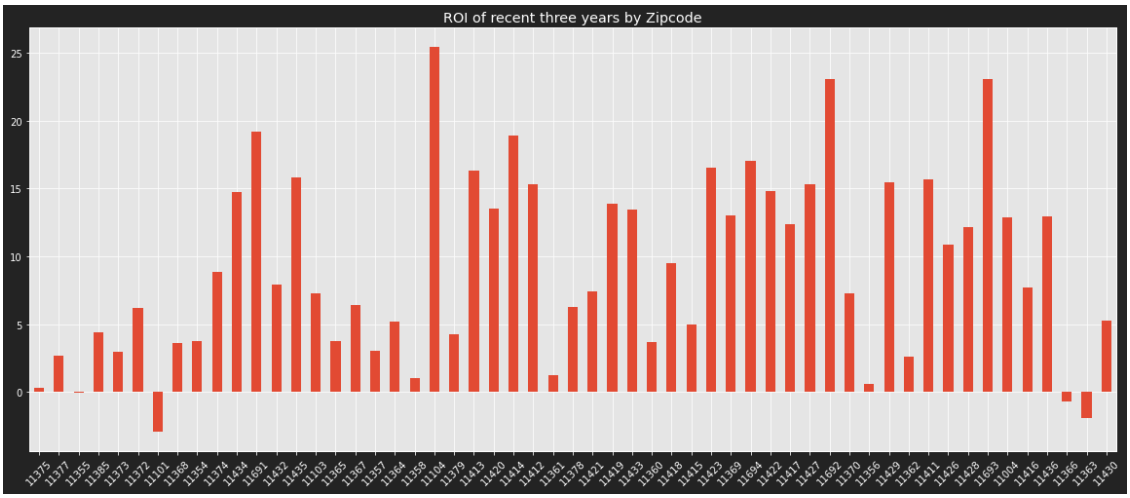
[23]:
```
fig, ax = plt.subplots()
ts_df['2010-1-31':].mean(1).resample('A').mean().
 ↪plot(kind='bar',figsize=(18,5),ax=ax)
ax.set_xticklabels(ts_df['2010-1-31':].mean(1).resample('A').mean().index.
 ↪strftime("%Y"))
fig.autofmt_xdate()
ax.set_title('Ten year trend')
plt.show()
```



comment

2.6.5 Three year ROI

[24]:
```
(((ts_df.iloc[-1, :] - ts_df.iloc[-37, :]) / ts_df.iloc[-37, :]) * 100).plot(
    kind='bar', figsize=(20, 8), title='ROI of recent three years by Zipcode')
plt.tick_params('x', labelrotation=45)
```



comment

2.6.6 Map of zipcodes

[25]:
```
# slicing last date's value
df_join = ts_df[-1:].T.reset_index()
df_join= df_join.rename(columns={'index':'Zipcode'})
df_join['values'] = df_join.iloc[:,1]
df_join = df_join[['Zipcode','values']]
# loading location data
# this data is sourced by the author by scraping web for location info
location_lat_long = pd.read_csv('./data/lat_long.csv')
# some cleaing is required as per scraping overlook
location_lat_long.long = location_lat_long.long.apply(lambda x: x*-1)
location_lat_long = location_lat_long.rename(columns={'index':'Zipcode'})
location_lat_long.Zipcode = location_lat_long.Zipcode.astype('str')
# data for map
map_df = pd.merge(location_lat_long,df_join, how='inner', on='Zipcode')
```

[26]:
```
fig = px.scatter_mapbox(
    map_df,
    lat=map_df.lat,
    lon=map_df.long,
    color='Zipcode',
    zoom=11,
    size='values',
    height=1200,
    template='plotly_dark',
    title='Zipcode location with House Value, as of 2021',
    center={
        'lat': map_df[map_df['Zipcode'] == '11418']['lat'].values[0],
        'lon': map_df[map_df['Zipcode'] == '11418']['long'].values[0]
    })
fig.update_layout(mapbox_style="stamen-toner")  # # "carto-positron"
fig.update_layout(margin={"r": 0, "l": 0, "b": 1})
fig.show()
```

Comment

3 MODEL

3.1 Model on test Zipcode

[39]:
```
# all zipcodes for analysis
print(zipcode_list)
```

```
['11375', '11377', '11355', '11385', '11373', '11372', '11101', '11368',
'11354', '11374', '11434', '11691', '11432', '11435', '11103', '11365', '11367',
```

```
'11357', '11364', '11358', '11104', '11379', '11413', '11420', '11414', '11412',
'11361', '11378', '11421', '11419', '11433', '11360', '11418', '11415', '11423',
'11369', '11694', '11422', '11417', '11427', '11692', '11370', '11356', '11429',
'11362', '11411', '11426', '11428', '11693', '11004', '11416', '11436', '11366',
'11363', '11430']
```

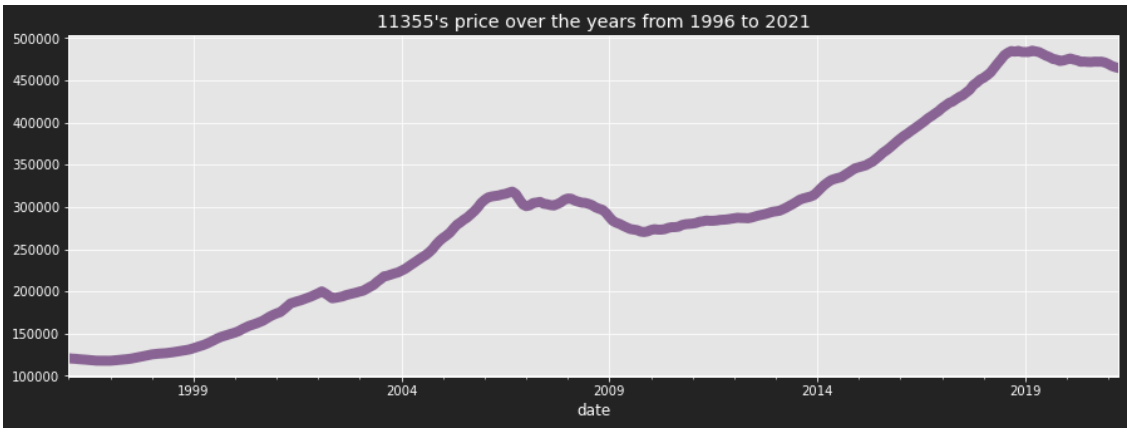
[40]:
```
# testing one zipcode randomly
zipcode = random.choice(zipcode_list)
```

[41]:
```
print('\033[1m \033[91m' +
      f"""Selected Zipcode : {ts_df[zipcode].name}""")
ts_df[zipcode]
```

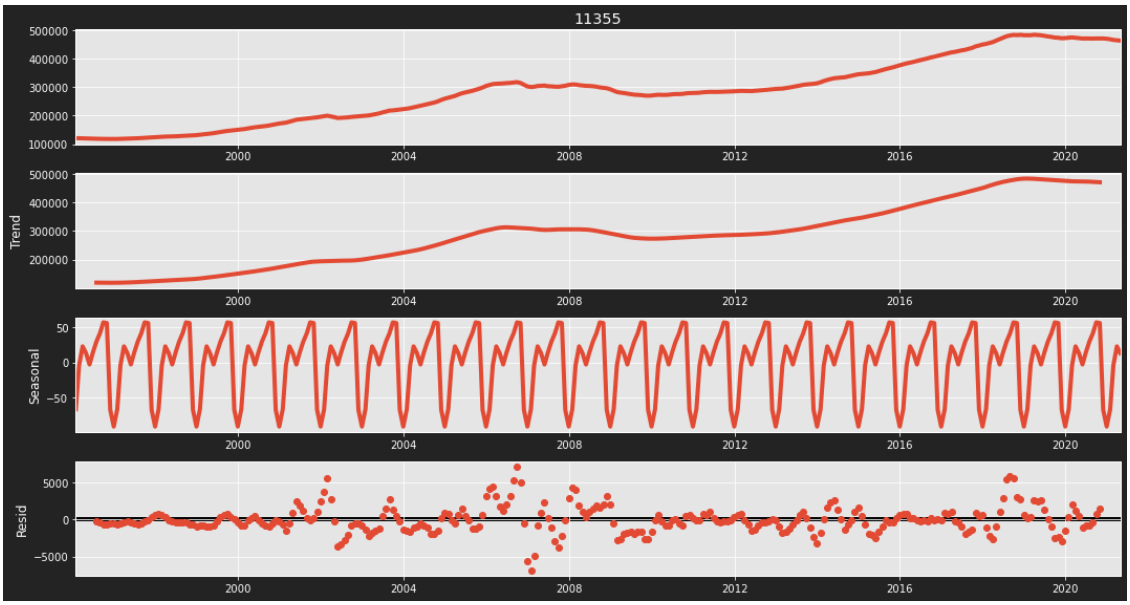
```
Selected Zipcode : 11355
```

[41]:
```
date
1996-01-31    120873.0
1996-02-29    120431.0
1996-03-31    120228.0
1996-04-30    119757.0
1996-05-31    119493.0
                 …
2020-12-31    471009.0
2021-01-31    469184.0
2021-02-28    466919.0
2021-03-31    465487.0
2021-04-30    464428.0
Freq: M, Name: 11355, Length: 304, dtype: float64
```

[42]:
```
# timeline
ts_df[zipcode].plot(
    figsize=(15, 5),
    legend=0,
    color='#886393',
    lw=8,
    title=
    f"""{zipcode}'s price over the years from {
        str(ts_df[zipcode].index[0]).split(" ")[0].split("-")[0]} to {
        str(ts_df[zipcode].index[-1]).split(" ")[0].split("-")[0]}"""
);
```

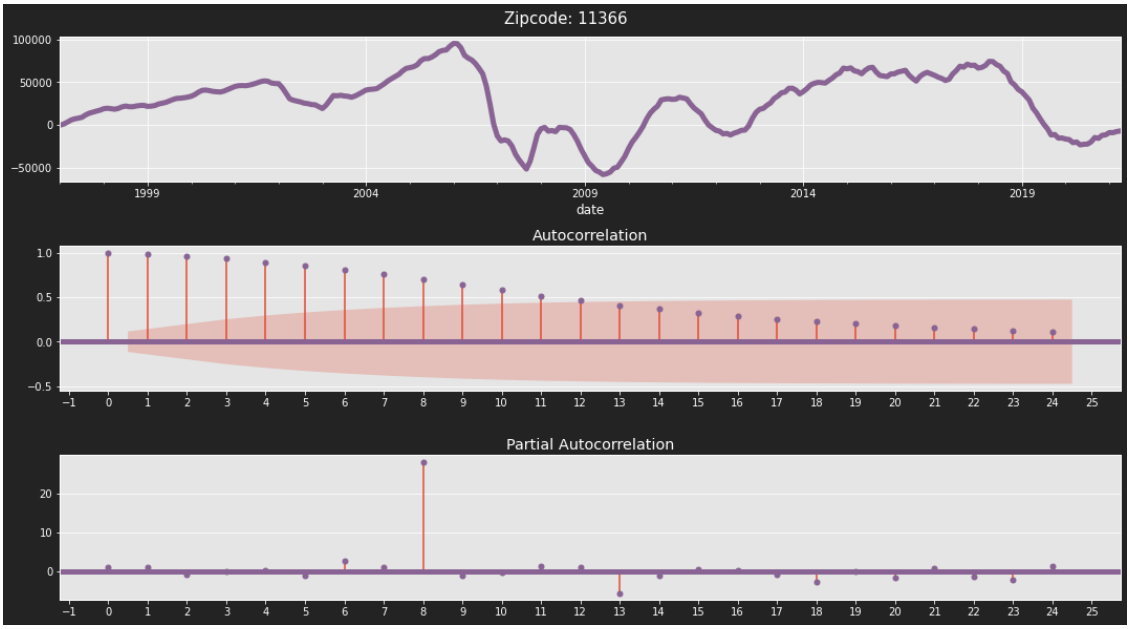


[43]:
```
with mpl.rc_context():
    mpl.rc('figure', figsize=(15,8))
    mpl.rc('lines', linewidth = 4)
    tsa.seasonal_decompose(ts_df[zipcode].dropna()).plot();
```



Can spot a recent uptrend and a highly seasonal pattern for all models tested.

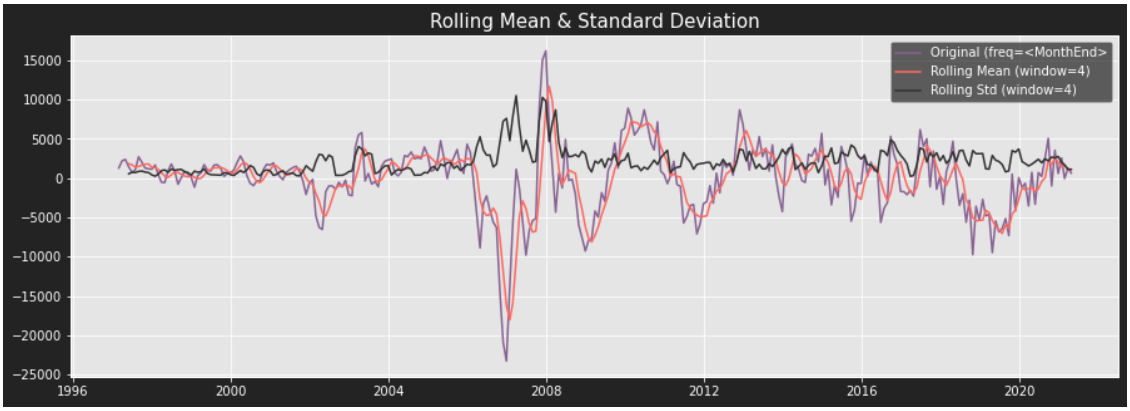
[26]:
```
## finding optimal p, d, q for ARIMA Models
# this data is higly seasonal data monthly so diffrenciating 12 times
fn.plot_acf_pacf(ts_df[zipcode].diff(12).dropna());
```



[27]:
```
fn.plot_acf_pacf(ts_df[zipcode].diff(1).dropna());
```

Observation varies from zipcode to zipcode for acf pacf plots, but usually diffrenciating 1 time then 12 times usually makes the data stationary.

[28]:
```
fn.stationarity_check((ts_df[zipcode].diff(1)).diff(12).dropna(), window=4)
```



[28]:

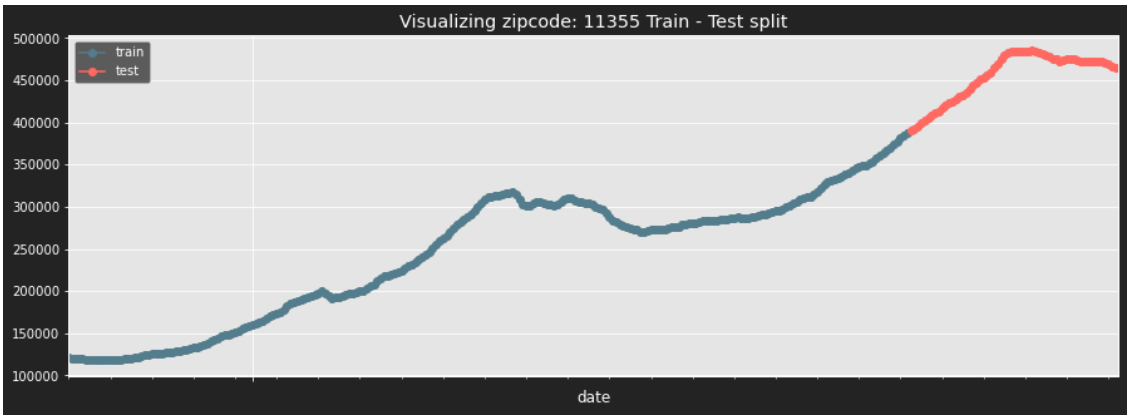
	Test Statistic	#Lags Used	…	p<.05	Stationary?
ADF Result	-3.571854	16	…	True	True

[1 rows x 6 columns]

3.1.1 Base MODEL

Model 1

[79]:
```
## train test split
train_size = 0.8
split_idx = round(len(ts_df[zipcode].dropna()) * train_size)

## Split
train = ts_df[zipcode].dropna().iloc[:split_idx]
test = ts_df[zipcode].dropna().iloc[split_idx:]

## Visualize the train-test split split
fig, ax = plt.subplots(figsize=(15, 5))
kws = dict(ax=ax, marker='o')
train.plot(**kws, label='train',color='#537d8d')
test.plot(**kws, label='test',color='#ff6961')
ax.legend()
plt.title(f'Visualizing zipcode: {zipcode} Train - Test split')
plt.show()
```



[33]:
```
# selected params for testing
# time series is not stationary, starting params
p = 0
d = 1
q = 0
# seasonality in data, starting with SARIMAX model
P = 0
D = 0
Q = 0
m = 12
```

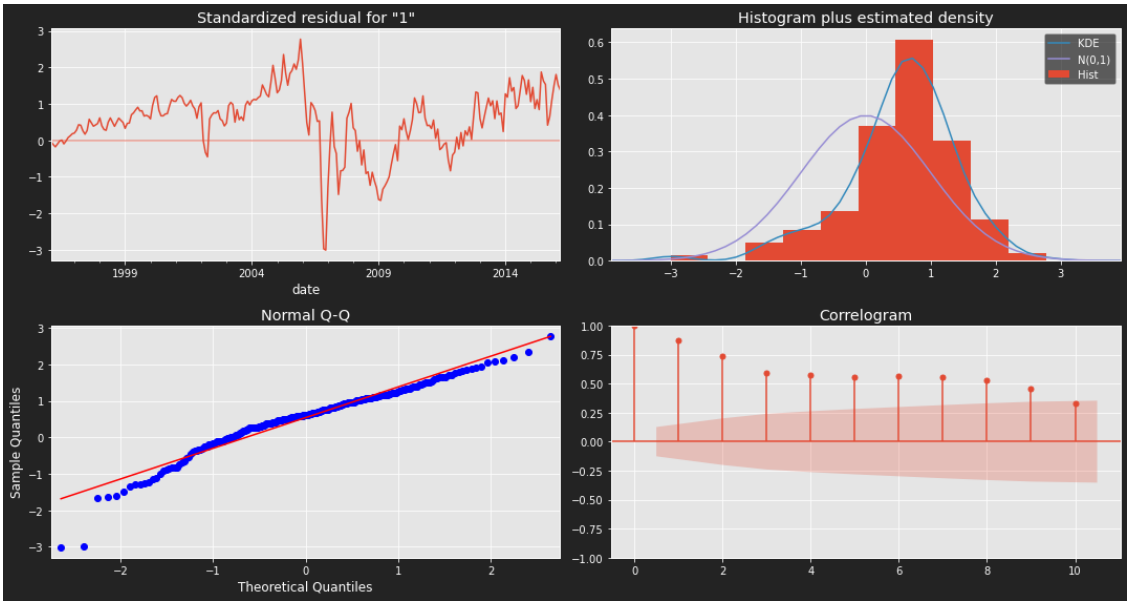
[34]:
```
order = (p, d, q)
seasonal_order = (P, D, Q, m)
```

[35]:
```
fn.model_builder(train, test, order, seasonal_order, zipcode);
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                SARIMAX Results
==========================================================================================
Dep. Variable:                          11366   No. Observations:                  243
Model:                 SARIMAX(0, 1, 0)   Log Likelihood               -2352.849
Date:                  Sun, 20 Jun 2021   AIC                             4707.698
Time:                          02:19:06   BIC                             4711.187
Sample:                      01-31-1996   HQIC                            4709.103
                           - 03-31-2016
Covariance Type:                    opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sigma2      1.624e+07   1.59e+06     10.230      0.000    1.31e+07    1.94e+07
==============================================================================
```

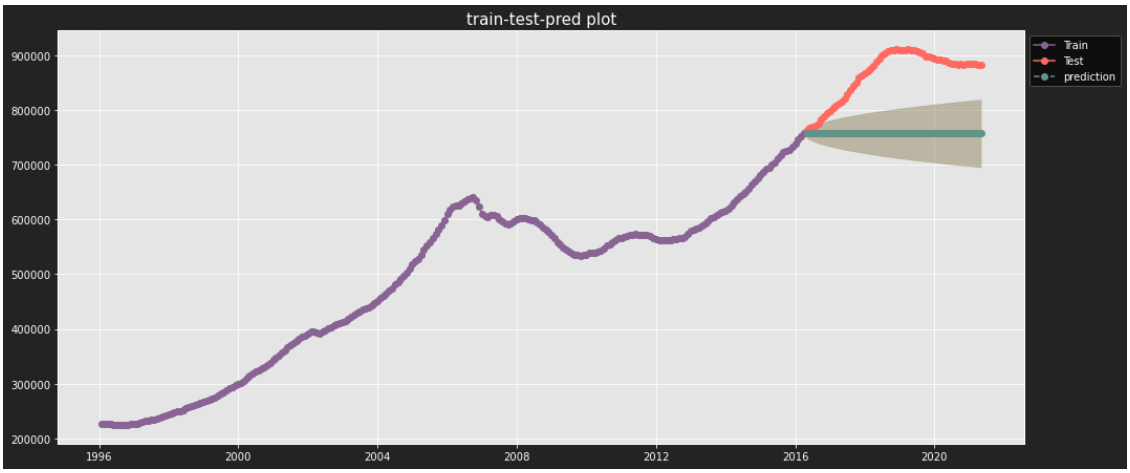
```
Ljung-Box (L1) (Q):                   187.09   Jarque-Bera (JB):                 77.
 ↪12
Prob(Q):                                0.00   Prob(JB):                         0.
 ↪00
Heteroskedasticity (H):                 1.90   Skew:                            -0.
 ↪90
Prob(H) (two-sided):                    0.00   Kurtosis:                         5.
 ↪10
================================================================================

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

Model Diagonostics of 11366



Performance on test data of 11366

```
Root Mean Squared Error of test and prediction: 117273.7829249342
Mean Squared Error: 13753140161.52459
Mean Absolute Error: 108136.80327868853
```



model fails to predict on test data. tuning parameters once again

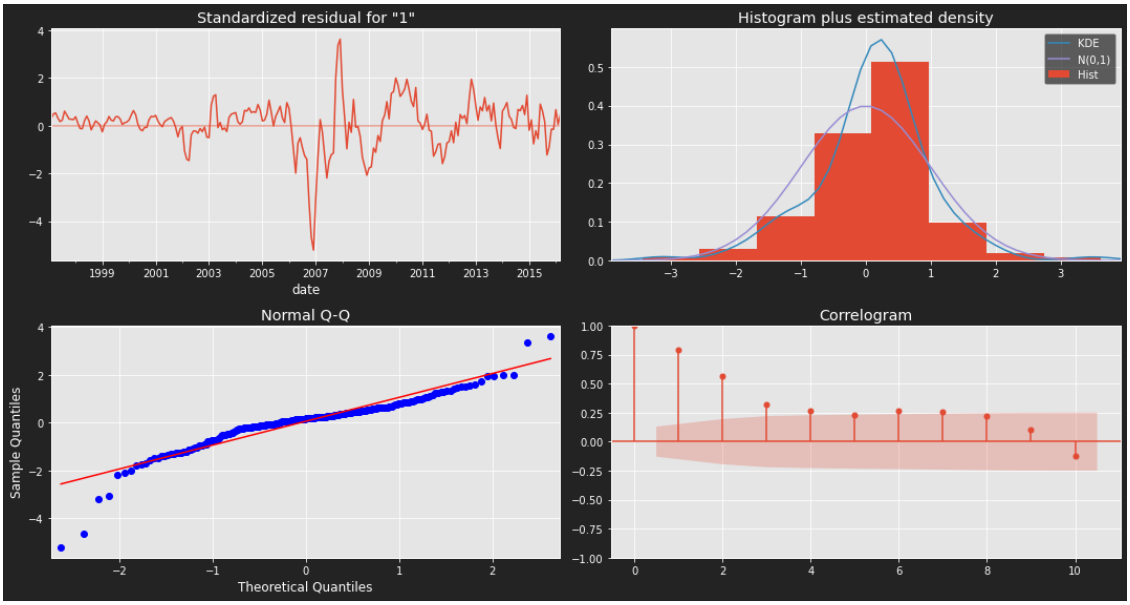
Model 2

[36]:
```
# selected params for testing
# time series is not stationary, starting params
p = 0
d = 1
q = 0
# seasonality in data, starting with SARIMAX model
P = 0
D = 1
Q = 0
m = 12
order = (p,d,q)
seasonal_order = (P,D,Q,m)
fn.model_builder(train, test, order, seasonal_order, zipcode);
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
```

```
==========================================================================================
Dep. Variable:                                11366   No. Observations:                ␣
 ↪      243
Model:             SARIMAX(0, 1, 0)x(0, 1, 0, 12)   Log Likelihood                   ␣
 ↪-2259.993
Date:                            Sun, 20 Jun 2021   AIC                              ␣
 ↪ 4521.987
Time:                                    02:20:09   BIC                              ␣
 ↪ 4525.425
Sample:                                01-31-1996   HQIC                             ␣
 ↪ 4523.374
                                     - 03-31-2016
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sigma2          2e+07   9.55e+05     20.945      0.000    1.81e+07    2.19e+07
===================================================================================
Ljung-Box (L1) (Q):                 146.71   Jarque-Bera (JB):                381.
 ↪04
Prob(Q):                              0.00   Prob(JB):                          0.
 ↪00
Heteroskedasticity (H):               4.13   Skew:                             -1.
 ↪12
Prob(H) (two-sided):                  0.00   Kurtosis:                          8.
 ↪89
===================================================================================

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

```
Model Diagonostics of 11366
```

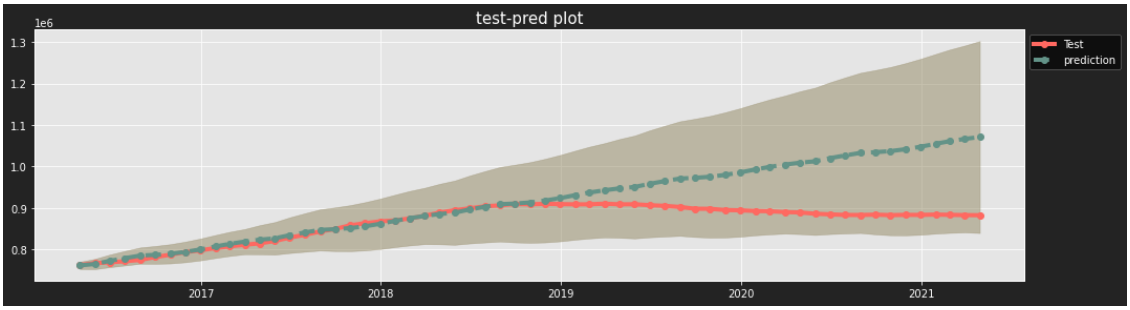


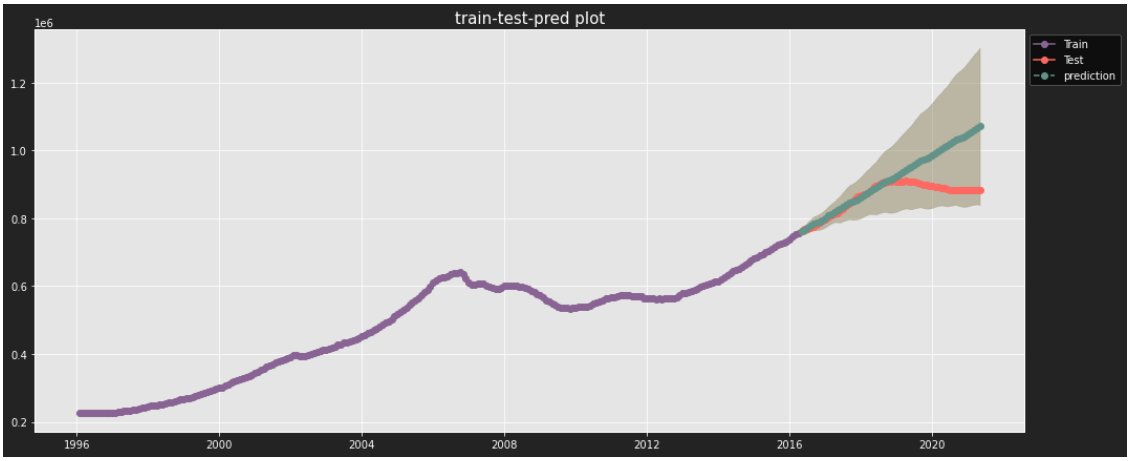
Performance on test data of 11366

Root Mean Squared Error of test and prediction: 80886.0900725799

Mean Squared Error: 6542559567.229508

Mean Absolute Error: 51838.83606557377





looking good on the prediction, but residual plots are not good and prediction is off towards later part of prediction, thus model can be better.

3.1.2 grid searching using pmdarima

BEST MODEL

Grid searching using pyramidarima for best p, d, q, P, D, Q, m for using in a SARIMA model using predefined conditions and shows model performance for predicting in the future.

Predefined parameters: - d and D is calculated using ndiffs using 'adf'(Augmented Dickey–Fuller test for Unit Roots) for d and 'ocsb' (Osborn, Chui, Smith, and Birchenhall Test for Seasonal Unit Roots) for D. - parameters for auto_arima model: - start_p = 0; The starting value of p, the order (or number of time lags) of the auto-regressive ("AR") model. - d = d; The order of first-differencing, - start_q = 0; order of the moving-average ("MA") model, - max_p = 3, max value for p - max_q = 3, max value for q - start_P = 0; the order of the auto-regressive portion of the seasonal model, - D = D; The order of the seasonal differencing, - start_Q = 0; the order of the moving-average portion of the seasonal model, - max_P = 3, max value of P - max_Q = 3, max value for Q - m = 12; The period for seasonal differencing, - refers to the number of periods in each season., - seasonal = True; this data is seasonal, - stationary = False; data is not stationary, - information_criterion = 'oob', optimizing on `out-of-bag` sample validation on a scoring metric, - other information criterias did not perform well - out_of_sample_size = 12, step hold out for validation, - scoring = 'mse', validation metric, - method = 'lbfgs'; limited-memory Broyden-Fletcher-Goldfarb-Shanno with optional box constraints, BFGS is in the family of quasi-Newton-Raphson methods that approximates the `bfgs` using a limited amount of computer memory.

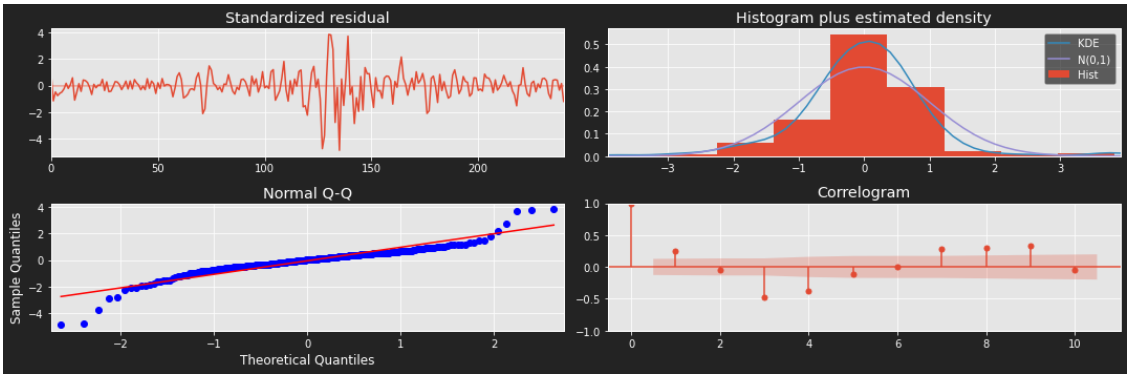
all other parameters were left at default.

[139]:
```
fn.grid_search(ts_df[zipcode], train, test, display_roi_results=True);
```

```
Performing stepwise search to minimize oob
 ARIMA(0,2,0)(0,0,0)[12]             : OOB=51453071.833, Time=0.02 sec
 ARIMA(1,2,0)(1,0,0)[12]             : OOB=47229900.474, Time=0.09 sec
```

```
 ARIMA(0,2,1)(0,0,1)[12]             : OOB=47436844.932, Time=0.13 sec
 ARIMA(1,2,0)(0,0,0)[12]             : OOB=51276016.007, Time=0.04 sec
 ARIMA(1,2,0)(2,0,0)[12]             : OOB=46280232.257, Time=0.42 sec
 ARIMA(1,2,0)(3,0,0)[12]             : OOB=46992892.533, Time=0.47 sec
 ARIMA(1,2,0)(2,0,1)[12]             : OOB=52690534.769, Time=1.18 sec
 ARIMA(1,2,0)(1,0,1)[12]             : OOB=53608838.451, Time=0.42 sec
 ARIMA(1,2,0)(3,0,1)[12]             : OOB=52704169.892, Time=2.27 sec
 ARIMA(0,2,0)(2,0,0)[12]             : OOB=45580256.338, Time=0.21 sec
 ARIMA(0,2,0)(1,0,0)[12]             : OOB=46784245.158, Time=0.07 sec
 ARIMA(0,2,0)(3,0,0)[12]             : OOB=47248893.180, Time=0.91 sec
 ARIMA(0,2,0)(2,0,1)[12]             : OOB=inf, Time=1.65 sec
 ARIMA(0,2,0)(1,0,1)[12]             : OOB=53758361.353, Time=0.65 sec
 ARIMA(0,2,0)(3,0,1)[12]             : OOB=inf, Time=3.31 sec
 ARIMA(0,2,1)(2,0,0)[12]             : OOB=46387541.922, Time=0.24 sec
 ARIMA(1,2,1)(2,0,0)[12]             : OOB=46448364.410, Time=0.35 sec
 ARIMA(0,2,0)(2,0,0)[12] intercept   : OOB=41762075.998, Time=0.40 sec
 ARIMA(0,2,0)(1,0,0)[12] intercept   : OOB=42644760.495, Time=0.12 sec
 ARIMA(0,2,0)(3,0,0)[12] intercept   : OOB=42404654.908, Time=0.74 sec
 ARIMA(0,2,0)(2,0,1)[12] intercept   : OOB=49999328.412, Time=1.12 sec
 ARIMA(0,2,0)(1,0,1)[12] intercept   : OOB=50788751.326, Time=0.55 sec
 ARIMA(0,2,0)(3,0,1)[12] intercept   : OOB=49998552.190, Time=2.08 sec
 ARIMA(1,2,0)(2,0,0)[12] intercept   : OOB=43428318.787, Time=1.52 sec
 ARIMA(0,2,1)(2,0,0)[12] intercept   : OOB=43328201.535, Time=0.47 sec
 ARIMA(1,2,1)(2,0,0)[12] intercept   : OOB=41224251.618, Time=0.68 sec
 ARIMA(1,2,1)(1,0,0)[12] intercept   : OOB=42115585.600, Time=0.24 sec
 ARIMA(1,2,1)(3,0,0)[12] intercept   : OOB=41890189.116, Time=1.22 sec
 ARIMA(1,2,1)(2,0,1)[12] intercept   : OOB=51051177.156, Time=2.28 sec
 ARIMA(1,2,1)(1,0,1)[12] intercept   : OOB=51902405.478, Time=0.80 sec
 ARIMA(1,2,1)(3,0,1)[12] intercept   : OOB=51055749.348, Time=5.36 sec
 ARIMA(2,2,1)(2,0,0)[12] intercept   : OOB=41869365.612, Time=1.78 sec
 ARIMA(1,2,2)(2,0,0)[12] intercept   : OOB=19128076.718, Time=1.40 sec
 ARIMA(1,2,2)(1,0,0)[12] intercept   : OOB=19833965.051, Time=0.52 sec
 ARIMA(1,2,2)(3,0,0)[12] intercept   : OOB=19619926.418, Time=2.57 sec
 ARIMA(1,2,2)(2,0,1)[12] intercept   : OOB=30732167.772, Time=2.76 sec
 ARIMA(1,2,2)(1,0,1)[12] intercept   : OOB=31088448.032, Time=1.66 sec
 ARIMA(1,2,2)(3,0,1)[12] intercept   : OOB=30618268.101, Time=4.61 sec
 ARIMA(0,2,2)(2,0,0)[12] intercept   : OOB=41641094.081, Time=0.79 sec
 ARIMA(2,2,2)(2,0,0)[12] intercept   : OOB=30359326.510, Time=2.07 sec
 ARIMA(1,2,3)(2,0,0)[12] intercept   : OOB=20388427.300, Time=1.59 sec
 ARIMA(0,2,3)(2,0,0)[12] intercept   : OOB=37875331.820, Time=0.85 sec
 ARIMA(2,2,3)(2,0,0)[12] intercept   : OOB=23280260.111, Time=2.74 sec
 ARIMA(1,2,2)(2,0,0)[12]             : OOB=42948617.326, Time=0.88 sec

Best model:  ARIMA(1,2,2)(2,0,0)[12] intercept
Total fit time: 54.273 seconds
 Model Diagonostics of 11417

<class 'statsmodels.iolib.summary.Summary'>
```

```
"""
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                   y   No. Observations:                  ␣
 ↪       243
Model:             SARIMAX(1, 2, 2)x(2, 0, [], 12)   Log Likelihood                  ␣
 ↪ -2108.641
Date:                             Fri, 18 Jun 2021   AIC                              ␣
 ↪  4231.282
Time:                                     13:51:27   BIC                              ␣
 ↪  4255.676
Sample:                                          0   HQIC                             ␣
 ↪  4241.110
                                               - 243
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     13.6332     41.466      0.329      0.742     -67.639      94.905
ar.L1          0.7545      0.377      2.000      0.045       0.015       1.494
ma.L1         -0.7661      0.384     -1.995      0.046      -1.519      -0.013
ma.L2         -0.0172      0.015     -1.144      0.253      -0.047       0.012
ar.S.L12      -0.0190      0.011     -1.683      0.092      -0.041       0.003
ar.S.L24       0.0097      0.058      0.166      0.868      -0.104       0.124
sigma2      2.252e+06   1.23e+05     18.370      0.000    2.01e+06    2.49e+06
==========================================================================================
Ljung-Box (L1) (Q):                   14.92   Jarque-Bera (JB):                355.
 ↪62
Prob(Q):                              0.00   Prob(JB):                          0.
 ↪00
Heteroskedasticity (H):               1.54   Skew:                             -0.
 ↪59
Prob(H) (two-sided):                  0.05   Kurtosis:                          8.
 ↪83
==========================================================================================

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

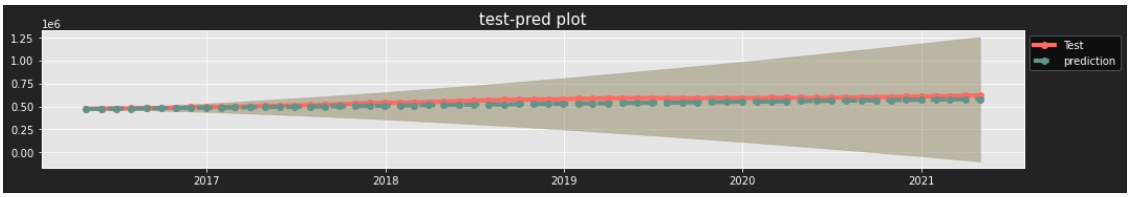


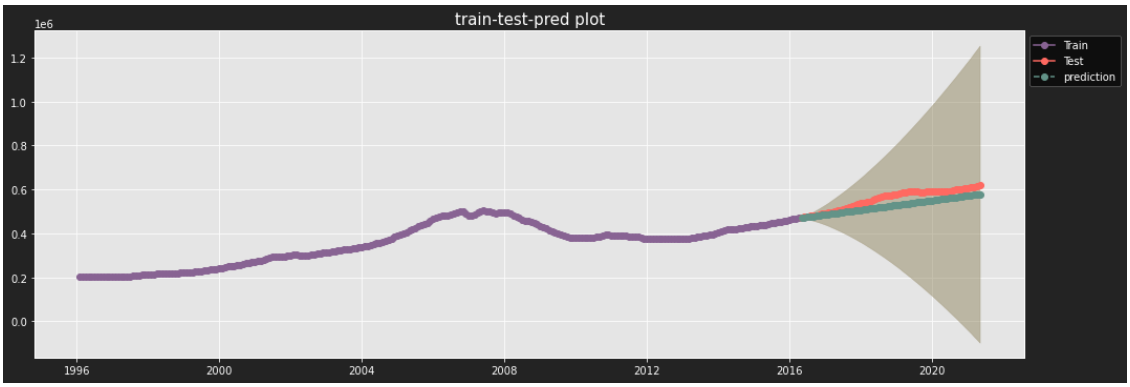
Performance on test data of 11417

Root Mean Squared Error of test and prediction: 35232.19339655126

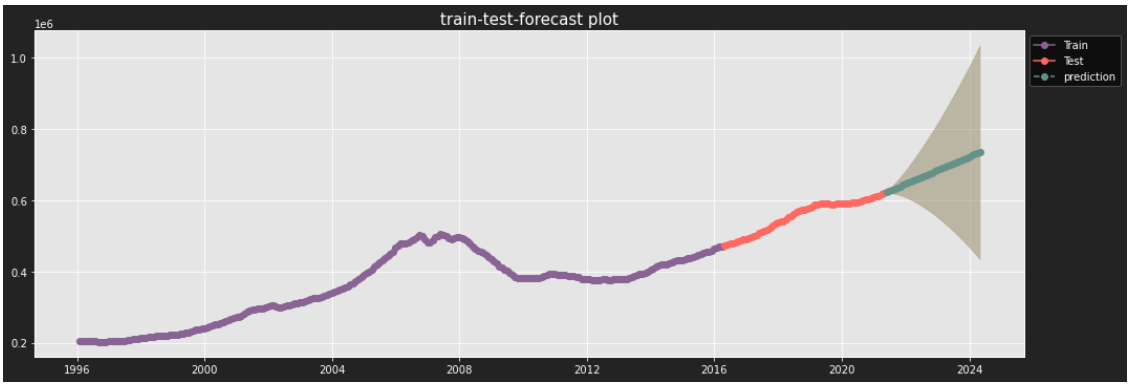
Mean Squared Error: 1241307451.5319903

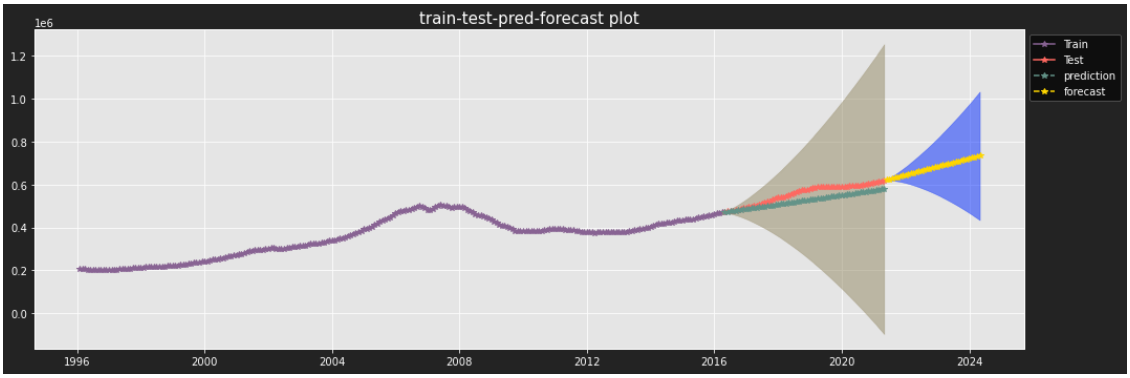
Mean Absolute Error: 31029.564416600693





Forecast of 11417





	zipcode	mean_forecasted_roi	…	upper_forecasted_roi	std_forecasted_roi
0	11417	18.57	…	66.95	48.38

[1 rows x 5 columns]

Model looks good in fitting and predicting with some long tailed residuals at both end. It can capture the future but with less certainty. This is expected as determinant house price is a combination of other factors which were not considered, e.g., loan interest rate, recent development and other external factors.

I am going to consider these parameters as the best one for this type of model. This can be improved by using SARIMAX model by using some of those factors as exog, but this increased model complexity and data needed for model as the exog's true data or a proxy is needed for prediction in the future.

3.2 All Zipcodes

[63]:
```
# looping through all the zipcodes with the best parameters of grid search
# my asumption is that those zipcodes are fairly similar in a broader sense
# as they share the same city. thus concluding that same model should be good
# enough to capture their nuances. later their performance will be evaluated.
```

```
results_, roi_ = fn.model_loop(ts_df,
                               zipcode_list[:2],
                               show_grid_search_steps=False,
                               display_details=False)
```

```
Working on #1 out of 2 zipcodes.
Working on: 11375
--------------------------------------------------------------------------------
----------
Working on #2 out of 2 zipcodes.
Working on: 11377
--------------------------------------------------------------------------------
----------
Looping completed.
```

This loop takes around 1.5 hours to run. For demo, output of only 5 zipdcodes are shown. Output of the loop is saved and used for the following parts. Those are not available in the repository as that file exceeds GitHubs file size limit by a lot.

= NOTE: What can be done?

[4]:
```
# filename='model'
# joblib.dump({'Results':results_, 'ROI': roi_}, f'./model/{filename}.joblib')
```

[27]:
```
lod = joblib.load('./model/model.joblib')
```

[28]:
```
results_ = lod['Results']
roi_ = lod['ROI']
```

[29]:
```
# # this is in percentage form
# roi_
```

3.3 High return Zipcodes

Criteria for selecting best zipcode: - Return on investment after three years

Cost is assumed to be the last true value of the median price of the zipcode, i.e., value on April 30, 2021. And revenue is assumed to the mean forecasted value after three years, i.e., 36 steps in the future. Then standard deviation is taken of the return on investment on upper confidence level and lower confidence level respectively as a proxy of risk of investment.

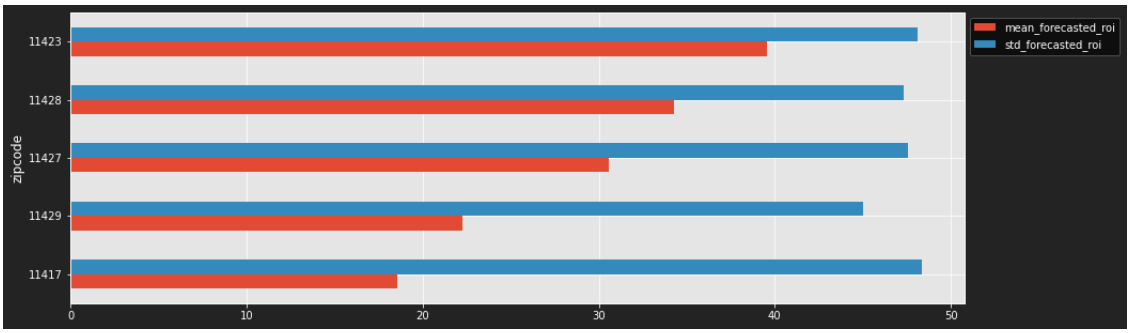
Top five zipcodes based on best 15 (27 percent of all available zipcode) ROI and then selecting top 5 of the based on lowest risk, i.e., the risk proxy mentioned above.

[30]:
```
# in tabuler form
((roi_.set_index('zipcode')[[
    'mean_forecasted_roi', 'std_forecasted_roi'
]].sort_values(by='mean_forecasted_roi')[-15:].sort_values(
    by='std_forecasted_roi')[:5]) * 100)
```

[30]:
```
         mean_forecasted_roi  std_forecasted_roi
zipcode
11429              22.230379           45.001385
11428              34.241021           47.287097
11427              30.558596           47.554807
11423              39.547861           48.105119
11417              18.564516           48.358635
```

[31]:
```
# visualizing
((roi_.set_index('zipcode')[[
    'mean_forecasted_roi', 'std_forecasted_roi'
]].sort_values(by='mean_forecasted_roi', ascending=True)[-15:].sort_values(
    by='std_forecasted_roi', ascending=False)[-5:]) * 100).sort_values(
        by='mean_forecasted_roi', ascending=True).plot(kind='barh',
                                                       figsize=(15, 5))
plt.legend(bbox_to_anchor=(1, 1), loc="upper left")
```

[31]:
```
<matplotlib.legend.Legend at 0x2943b8901c0>
```



[32]:
```
# these are the best zipcodes
best_investments = ((roi_.set_index('zipcode')[[
    'mean_forecasted_roi', 'std_forecasted_roi'
]].sort_values(by='mean_forecasted_roi')[-15:].sort_values(
    by='std_forecasted_roi')[:5]) * 100).index

best_investments
```

[32]:
```
Index(['11429', '11428', '11427', '11423', '11417'], dtype='object',
name='zipcode')
```

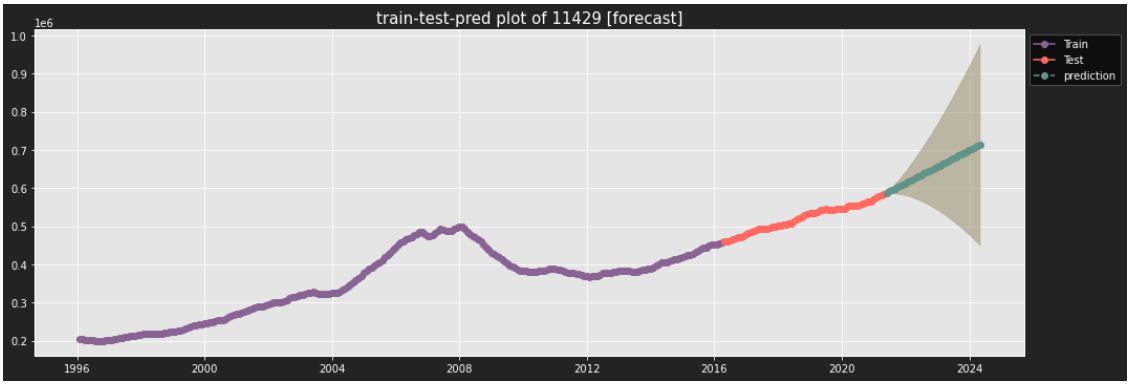
[33]:
```
# visualizing model performance of those five zipcodes
fn.model_report(best_investments,
                results_,
                show_model_performance=False,
                show_train_fit=False,
```

```
                show_prediction=True,
                show_detailed_prediction=True)
```

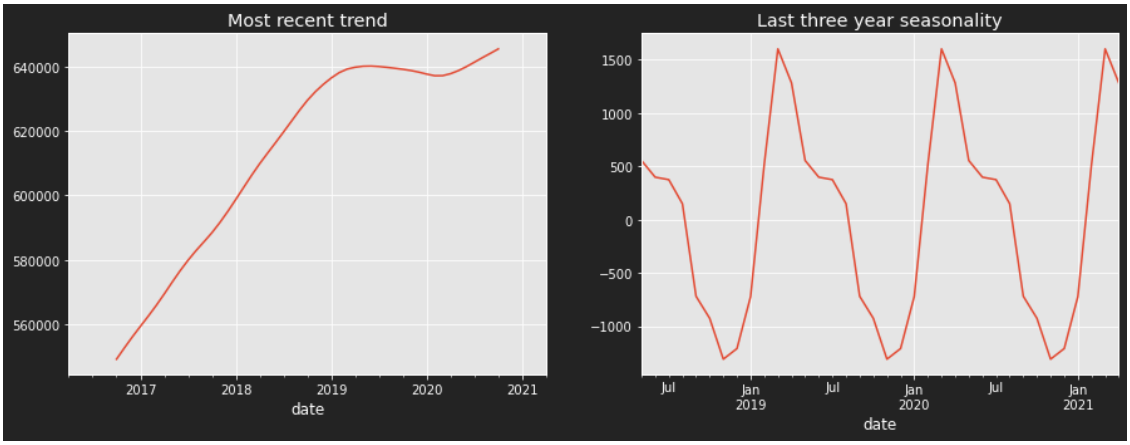
Report of 11429

Model Used:

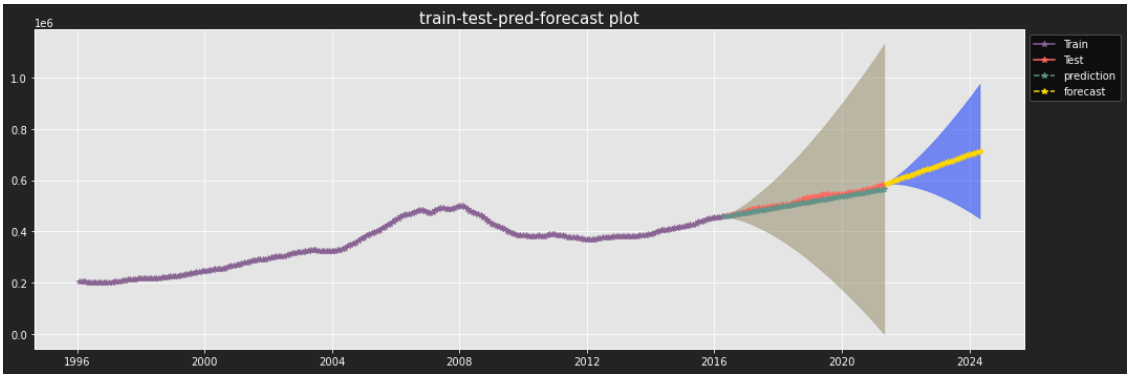
```
ARIMA(order=(3, 2, 1), out_of_sample_size=12, scoring_args={},
      seasonal_order=(2, 0, 0, 12), suppress_warnings=True)
```



Insights:



Overall model performance and projected ROI:



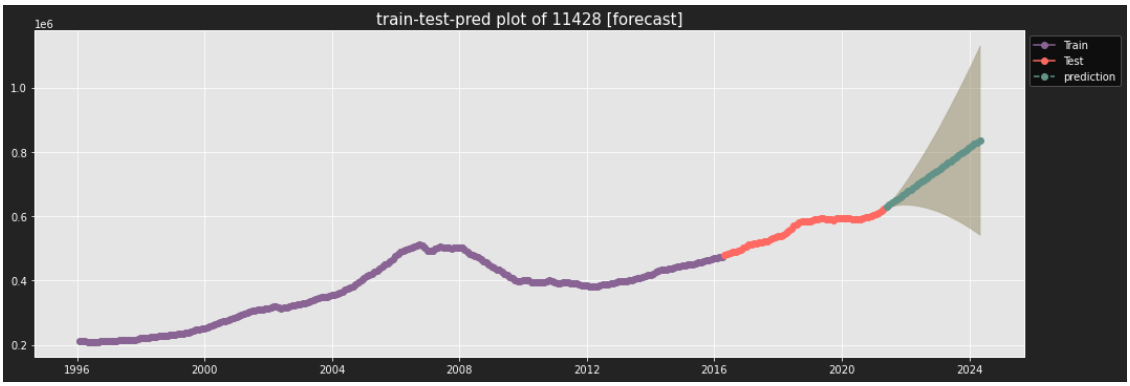
	zipcode	mean_forecasted_roi	…	upper_forecasted_roi	std_forecasted_roi
0	11429	22.23	…	67.23	45.0

[1 rows x 5 columns]

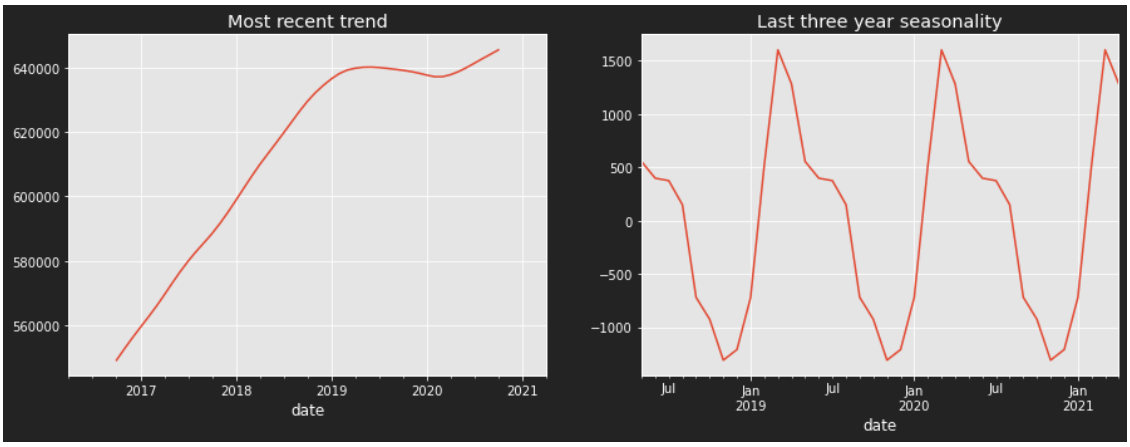
Report of 11428

Model Used:

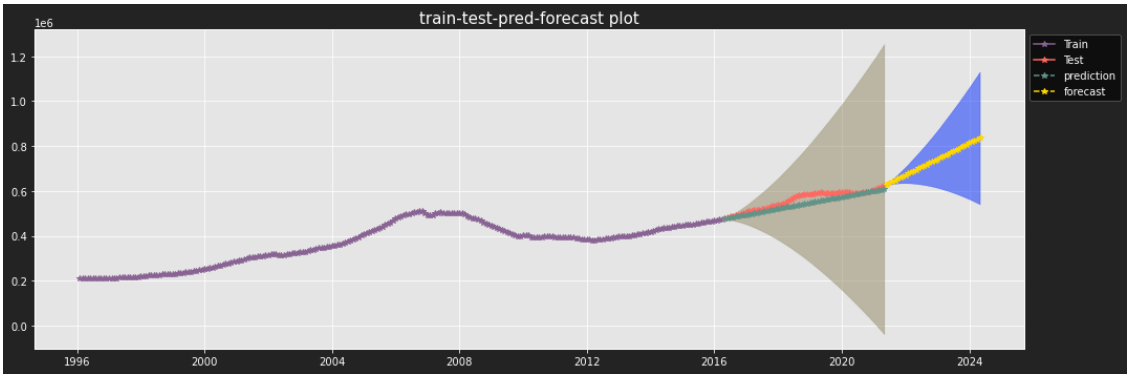
```
ARIMA(order=(1, 2, 3), out_of_sample_size=12, scoring_args={},
      seasonal_order=(2, 0, 0, 12), suppress_warnings=True)
```



Insights:



Overall model performance and projected ROI:



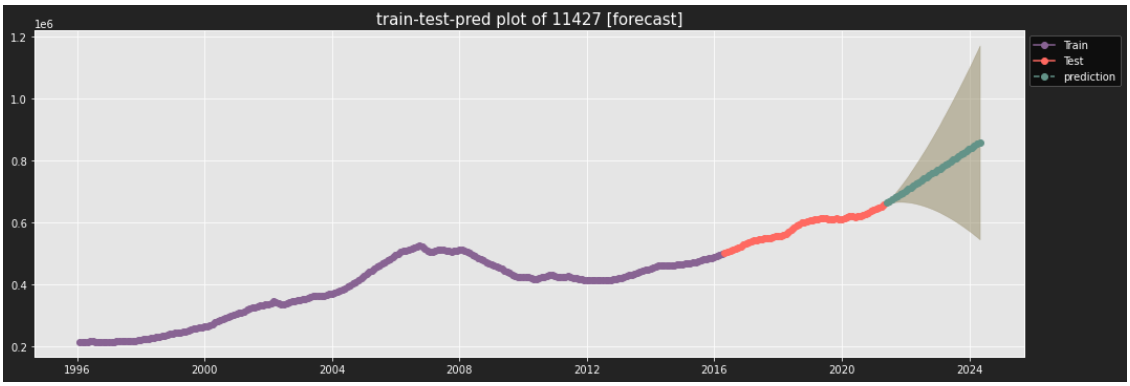
	zipcode	mean_forecasted_roi	…	upper_forecasted_roi	std_forecasted_roi
0	11428	34.24	…	81.53	47.29

[1 rows x 5 columns]

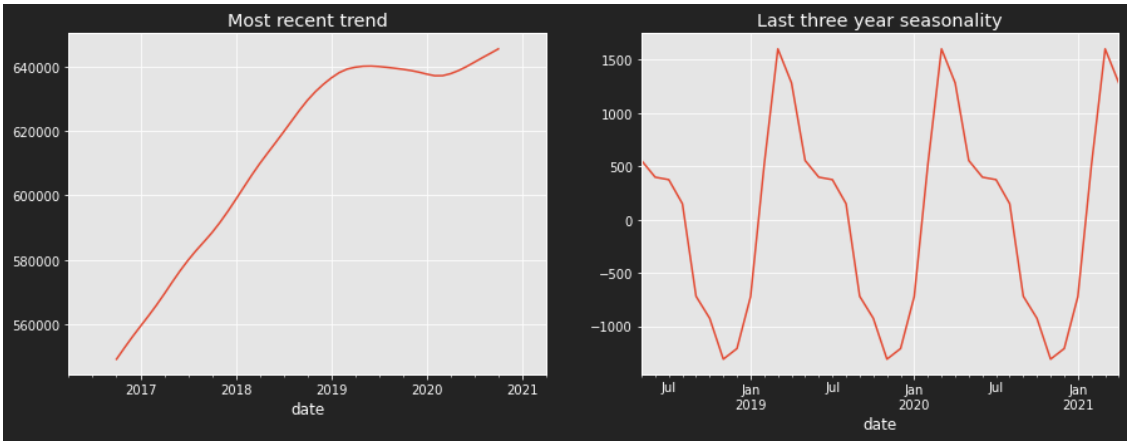
Report of 11427

Model Used:

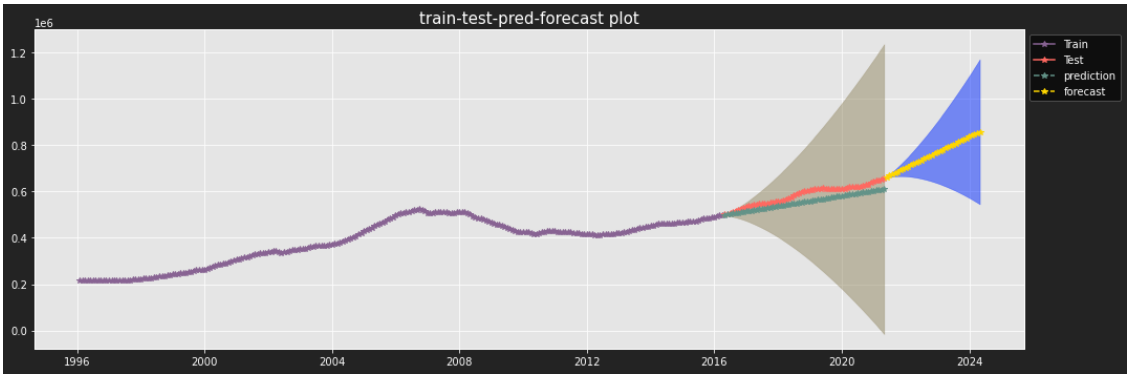
```
ARIMA(order=(1, 2, 3), out_of_sample_size=12, scoring_args={},
      seasonal_order=(1, 0, 3, 12), suppress_warnings=True)
```



Insights:



Overall model performance and projected ROI:



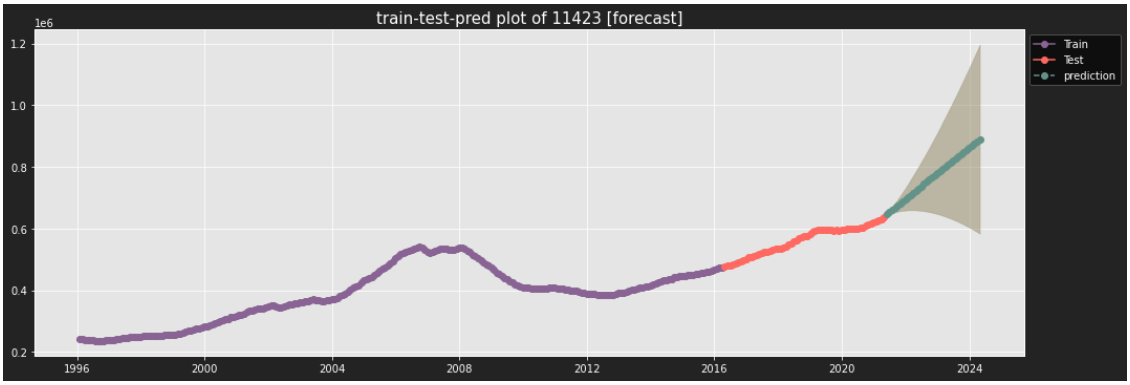
	zipcode	mean_forecasted_roi	…	upper_forecasted_roi	std_forecasted_roi
0	11427	30.56	…	78.11	47.55

[1 rows x 5 columns]

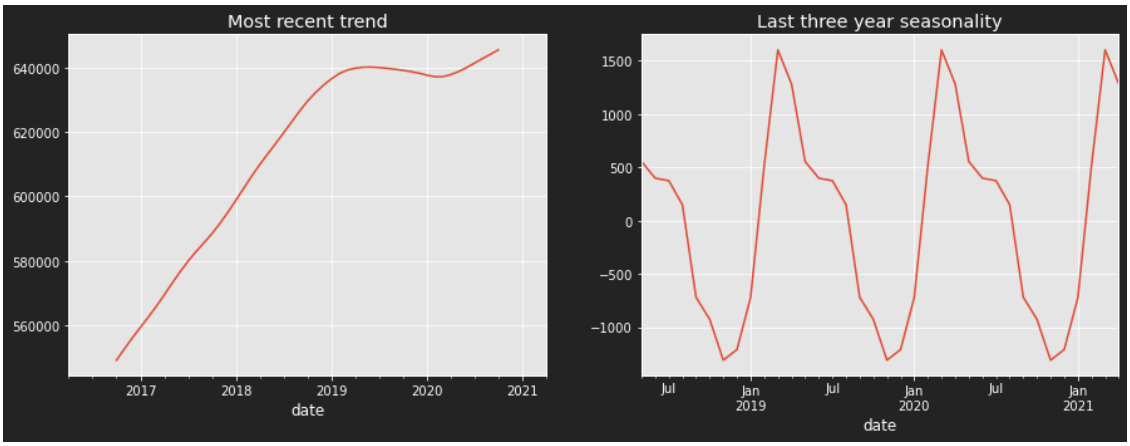
Report of 11423

Model Used:

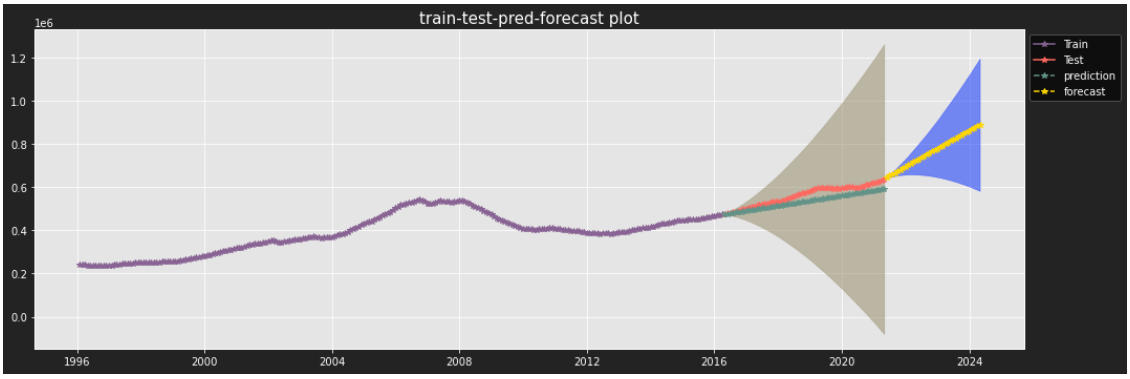
```
ARIMA(order=(3, 2, 1), out_of_sample_size=12, scoring_args={},
      seasonal_order=(0, 0, 1, 12), suppress_warnings=True)
```



Insights:



Overall model performance and projected ROI:



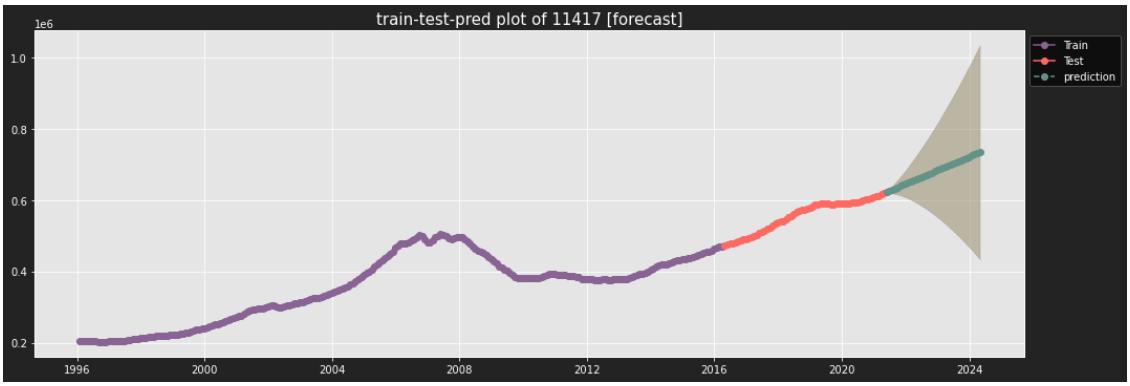
	zipcode	mean_forecasted_roi	…	upper_forecasted_roi	std_forecasted_roi
0	11423	39.55	…	87.65	48.11

[1 rows x 5 columns]

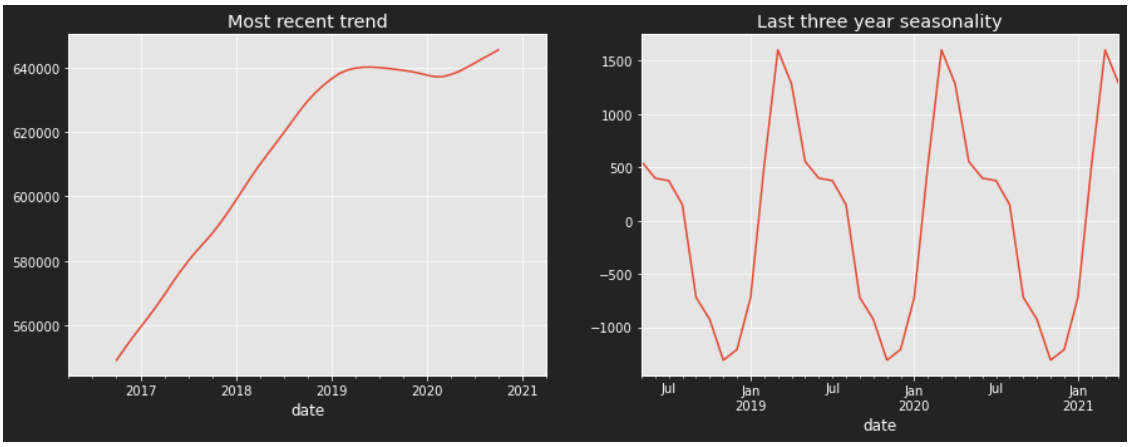
Report of 11417

Model Used:

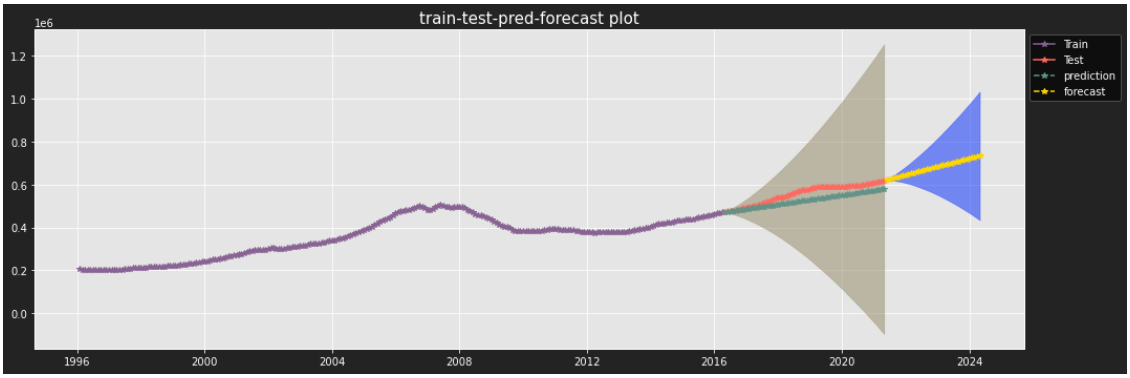
```
ARIMA(order=(1, 2, 2), out_of_sample_size=12, scoring_args={},
      seasonal_order=(2, 0, 0, 12), suppress_warnings=True)
```



Insights:



Overall model performance and projected ROI:




```
  zipcode  mean_forecasted_roi  …  upper_forecasted_roi  std_forecasted_roi
0   11417                18.56  …                 66.92               48.36

[1 rows x 5 columns]
```

[375]:
```
# statistical informations
fn.output_df(best_investments, results_)
```

[375]:
```
              aic  …  three_year_projected_upper_roi
ZipCode            …
11429     4188.75  …                           67.23
11428     4232.73  …                           81.53
11427     4193.11  …                           78.11
11423     4247.35  …                           87.65
11417     4231.28  …                           66.92

[5 rows x 12 columns]
```

3.3.1 Visual

[376]:
```
# Getting Neighbouhood name
code_match = pd.read_csv('./data/nyc-zip-codes.txt')
# from : `https://github.com/erikgregorywebb/nyc-housing/blob/master/Data/
 ↪nyc-zip-codes.csv`
code_match['ZipCode'] = code_match['ZipCode'].astype('str')
roi_viz = pd.merge(roi_, code_match, left_on='zipcode', right_on='ZipCode')
roi_viz.drop(columns=['Borough', 'ZipCode'], inplace=True)
# display(roi_viz)
roi_viz[roi_viz.columns.values[1:-1]] = roi_viz[
    roi_viz.columns.values[1:-1]].apply(lambda x: round(x * 100, 2), axis=1)
# roi_viz
```

[378]:
```
# # Putting things on a map
# fn.map_zipcodes_return(roi_viz, plot_style='static')
# # # Folium version
# # fn.zip_code_map(roi_)
```

img

3.4 Investigating Zipcodes with poor return

[55]:
```
# Isolating low returning to find out poor model fit
investigation = roi_.set_index('zipcode')[[
    'mean_forecasted_roi', 'std_forecasted_roi'
]].sort_values(by='mean_forecasted_roi')[:4].index
investigation
```

[55]:
```
Index(['11415', '11375', '11101', '11354'], dtype='object', name='zipcode')
```

[56]:
```
roi_.set_index('zipcode')[[
    'mean_forecasted_roi', 'std_forecasted_roi'
]].sort_values(by='mean_forecasted_roi')[:4]
```

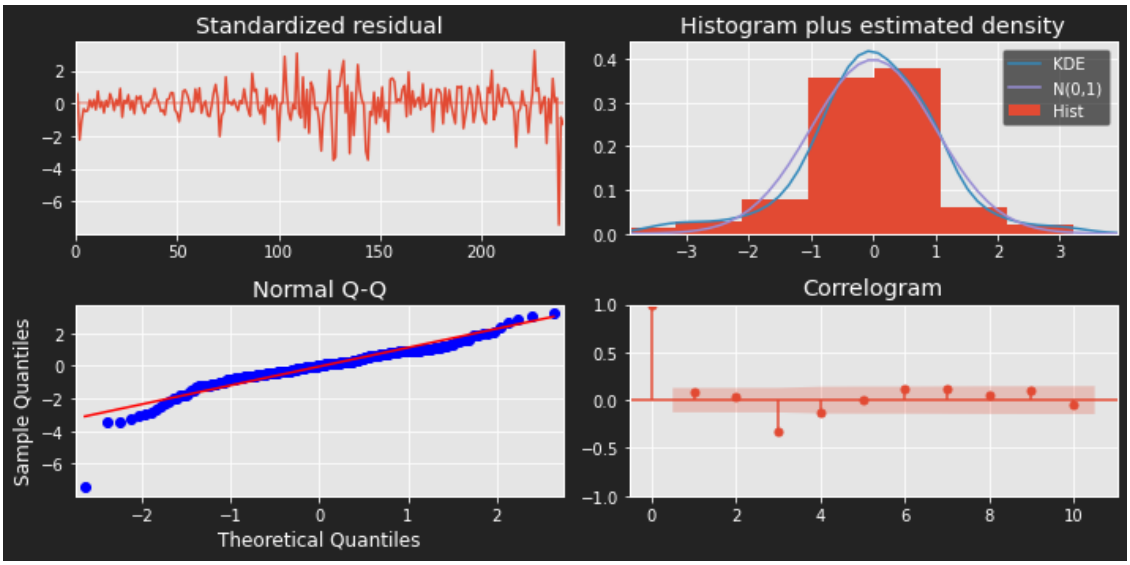
[56]:

zipcode	mean_forecasted_roi	std_forecasted_roi
11415	-0.710355	0.790114
11375	-0.407783	0.759214
11101	-0.234924	0.729505
11354	-0.171130	0.398002

[60]:
```
# inspecting model report to isolate bad models
fn.model_report(investigation,
                results_,
                show_model_performance=True,
                show_train_fit=True,
                show_prediction=True,test_conf_int=True,
```

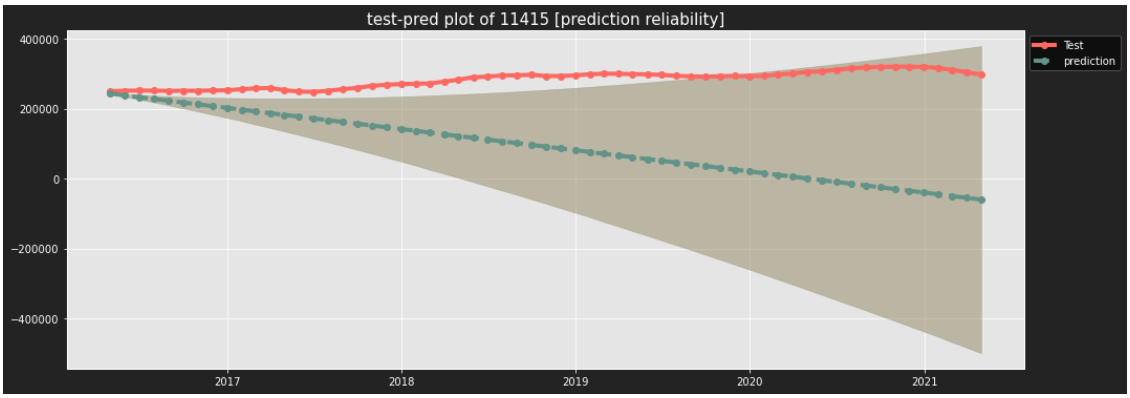
```
                show_detailed_prediction=True)
```

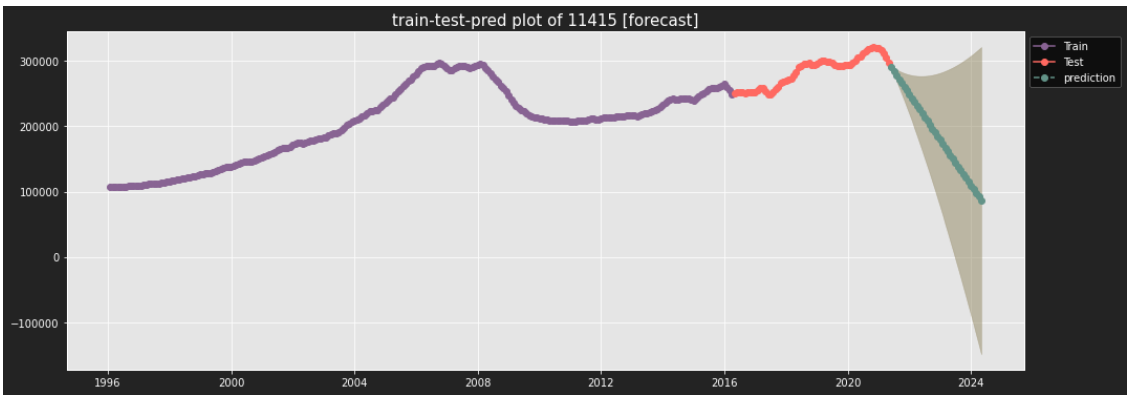
```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
 Report of 11415
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
 Model Used:

ARIMA(order=(1, 2, 3), out_of_sample_size=12, scoring_args={},
      seasonal_order=(0, 0, 0, 12), suppress_warnings=True,
      with_intercept=False)

                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  243
Model:               SARIMAX(1, 2, 3)   Log Likelihood               -2003.765
Date:                Sun, 20 Jun 2021   AIC                           4017.531
Time:                        03:21:00   BIC                           4034.955
Sample:                             0   HQIC                          4024.551
                                - 243
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4475      0.250      1.788      0.074      -0.043       0.938
ma.L1         -0.4490      0.254     -1.768      0.077      -0.947       0.049
ma.L2         -0.0145      0.031     -0.469      0.639      -0.075       0.046
ma.L3         -0.0663      0.017     -3.900      0.000      -0.100      -0.033
sigma2      6.965e+05   2.24e+04     31.097      0.000    6.53e+05     7.4e+05
==============================================================================
===
Ljung-Box (L1) (Q):                   1.86   Jarque-Bera (JB):
573.21
Prob(Q):                              0.17   Prob(JB):
0.00
Heteroskedasticity (H):               4.21   Skew:
-1.30
Prob(H) (two-sided):                  0.00   Kurtosis:
10.09
==============================================================================
===

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



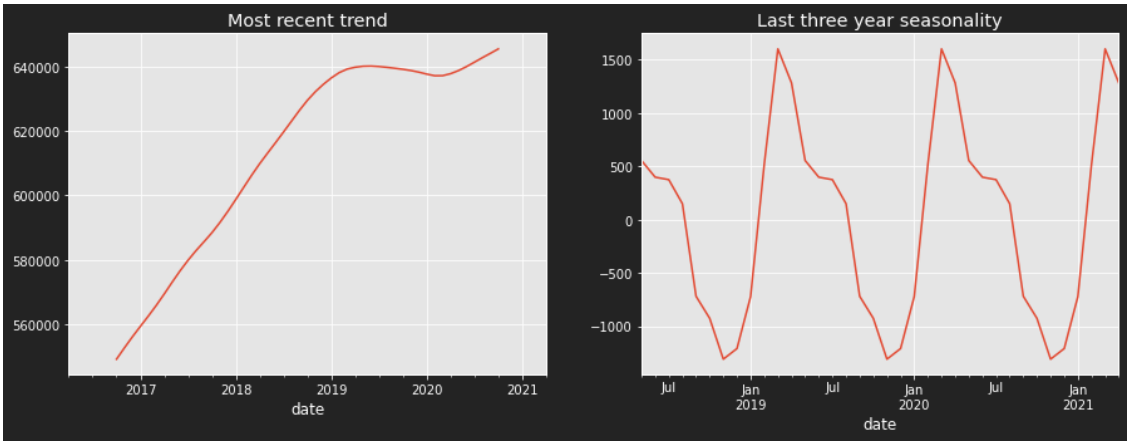
Prediction:

```
Root Mean Squared Error of test and prediction: 222405.8425298457
Mean Squared Error: 49464358791.41052
Mean Absolute Error: 192689.9140215953
```

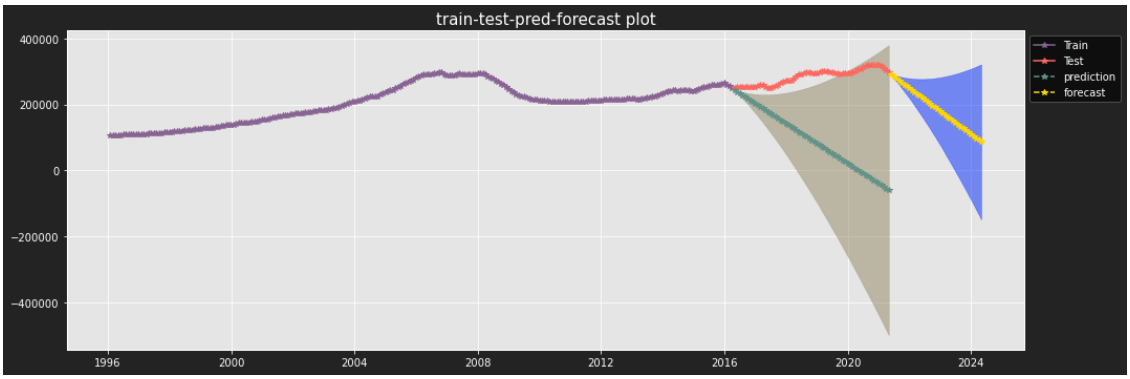




Insights:



Overall model performance and projected ROI:



	zipcode	mean_forecasted_roi	…	upper_forecasted_roi	std_forecasted_roi
0	11415	-71.04	…	7.98	79.01

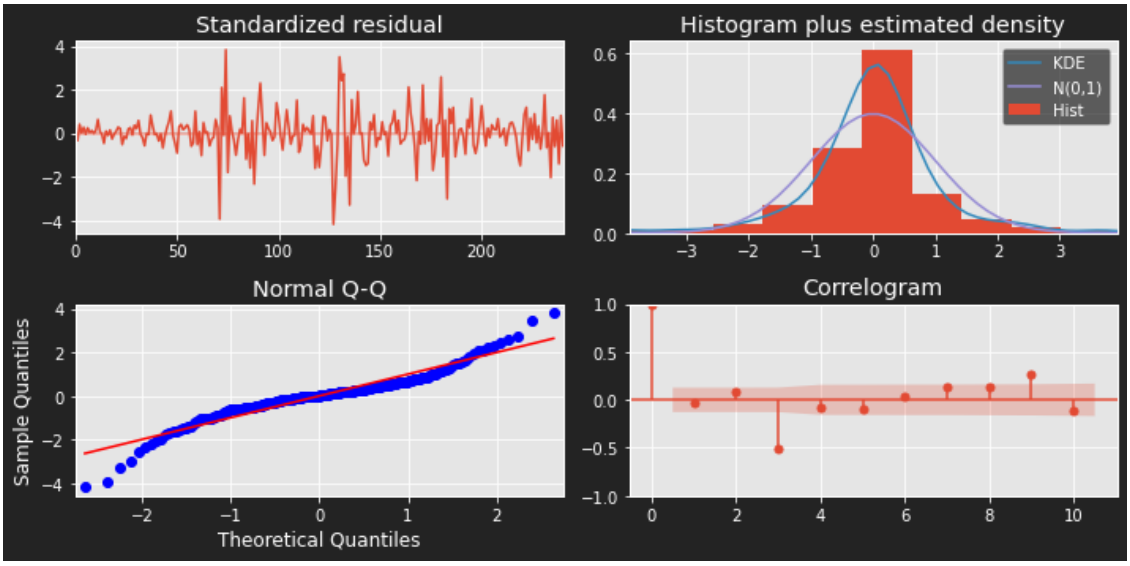
[1 rows x 5 columns]

Report of 11375

Model Used:

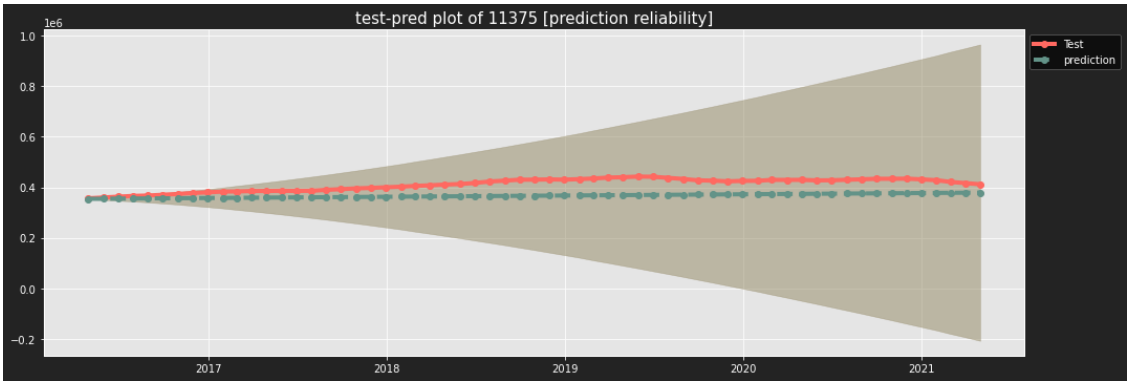
```
ARIMA(order=(1, 2, 0), out_of_sample_size=12, scoring_args={},
      seasonal_order=(0, 0, 0, 12), suppress_warnings=True,
```

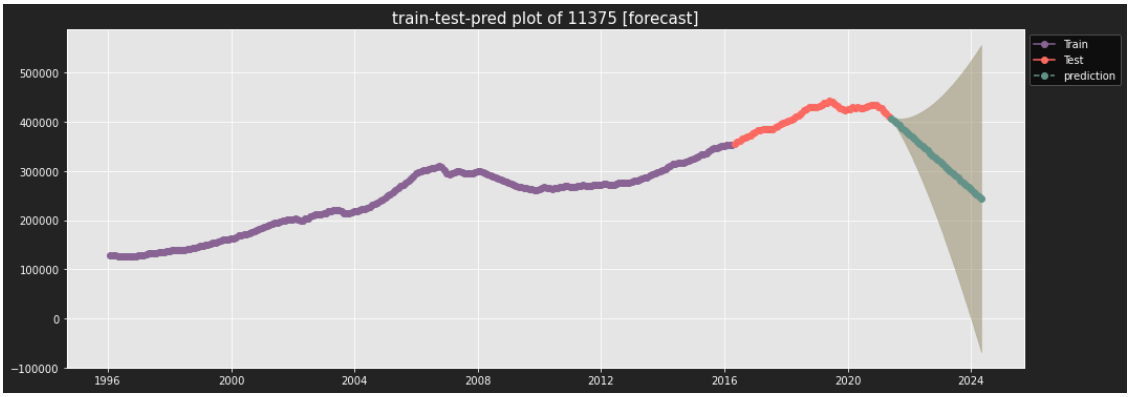
```
      with_intercept=False)

                               SARIMAX Results
==============================================================================
Dep. Variable:                          y   No. Observations:                  243
Model:                 SARIMAX(1, 2, 0)   Log Likelihood               -2025.933
Date:                Sun, 20 Jun 2021   AIC                           4055.867
Time:                        03:21:03   BIC                           4062.837
Sample:                             0   HQIC                          4058.675
                                  - 243
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0092      0.042     -0.218      0.827      -0.092       0.073
sigma2      1.175e+06   6.36e+04     18.479      0.000    1.05e+06     1.3e+06
===================================================================================
===
Ljung-Box (L1) (Q):                   0.29   Jarque-Bera (JB):
140.75
Prob(Q):                              0.59   Prob(JB):
0.00
Heteroskedasticity (H):               1.42   Skew:
-0.27
Prob(H) (two-sided):                  0.12   Kurtosis:
6.70
===================================================================================
===

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



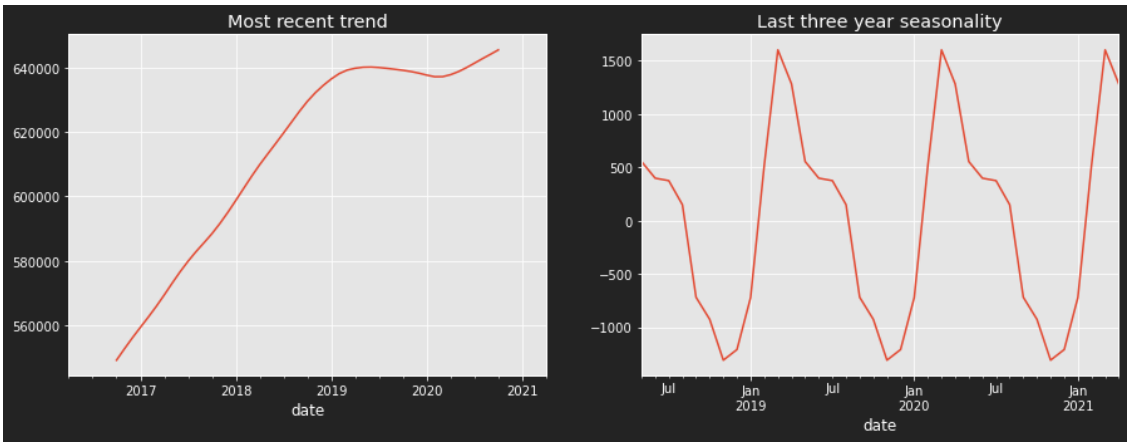
Prediction:

```
Root Mean Squared Error of test and prediction: 47935.52415165556
Mean Squared Error: 2297814475.693954
Mean Absolute Error: 44126.14249132199
```

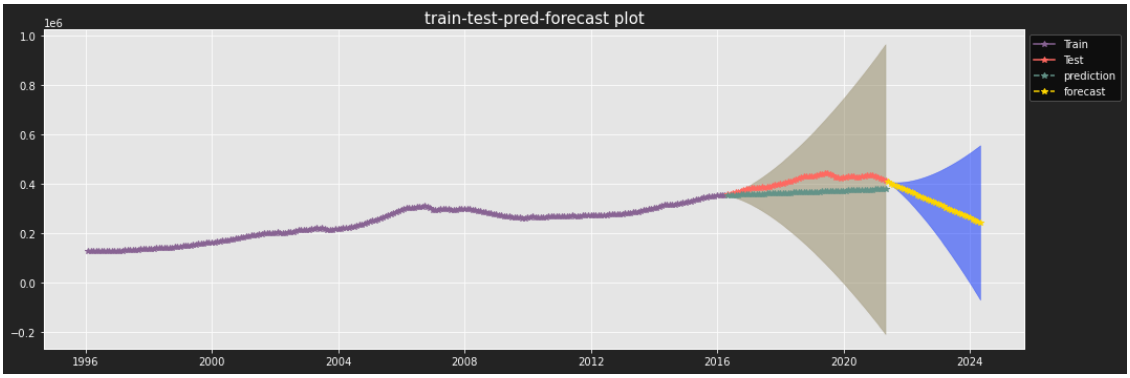




Insights:



Overall model performance and projected ROI:



	zipcode	mean_forecasted_roi	…	upper_forecasted_roi	std_forecasted_roi
0	11375	-40.78	…	35.14	75.92

[1 rows x 5 columns]

Report of 11101

Model Used:

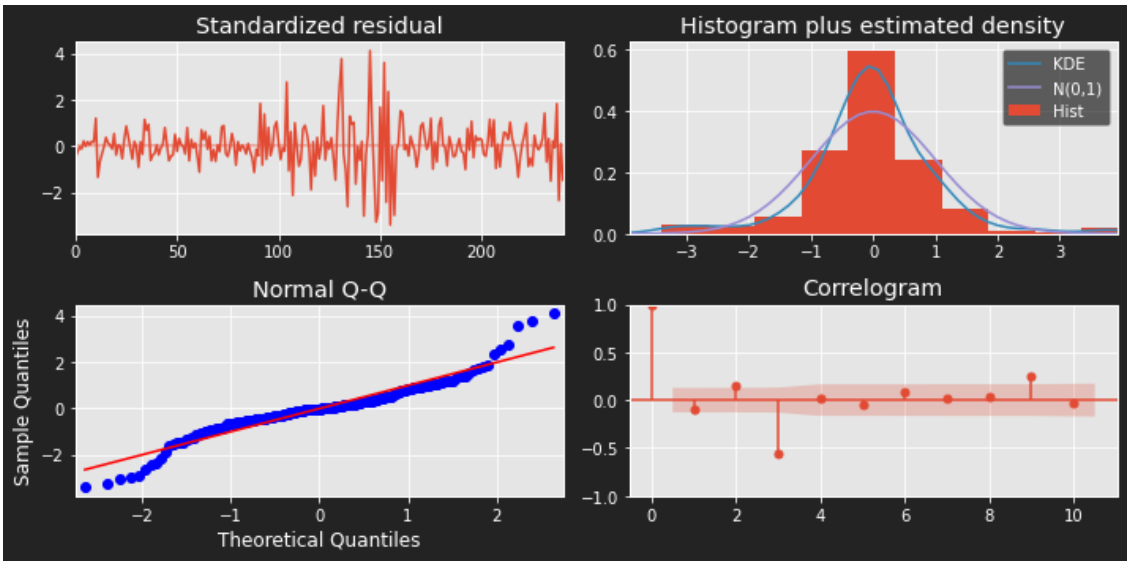
```
ARIMA(order=(1, 2, 0), out_of_sample_size=12, scoring_args={},
      seasonal_order=(3, 0, 2, 12), suppress_warnings=True)
```

SARIMAX Results

==========

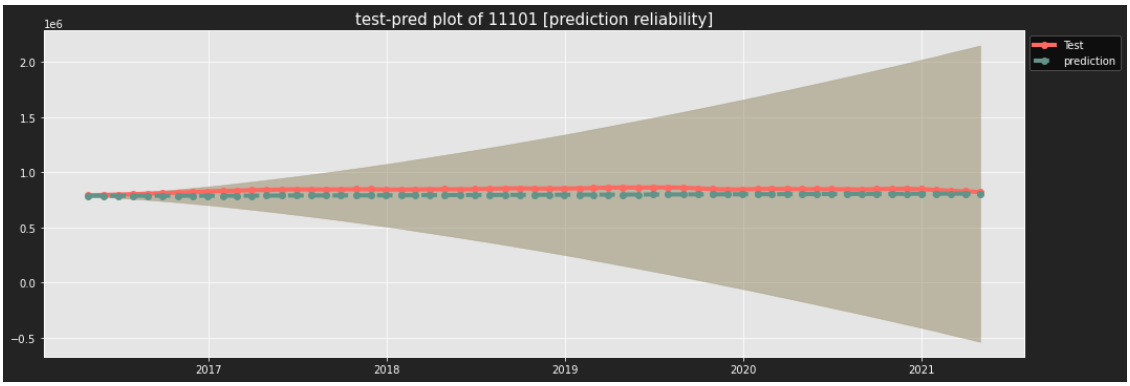
Dep. Variable: y No. Observations:

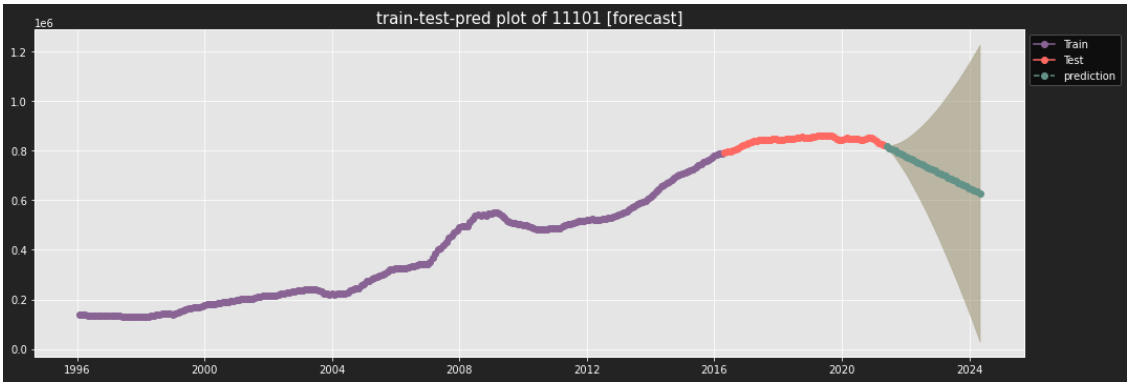
```
243
Model:             SARIMAX(1, 2, 0)x(3, 0, [1, 2], 12)   Log Likelihood
-2233.012
Date:                                 Sun, 20 Jun 2021   AIC
4482.024
Time:                                         03:21:04   BIC
4509.902
Sample:                                              0   HQIC
4493.255
                                                 - 243
Covariance Type:                                   opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     22.2906   1399.221      0.016      0.987   -2720.132    2764.713
ar.L1         -0.0169      0.032     -0.524      0.600      -0.080       0.046
ar.S.L12      -0.0818     28.146     -0.003      0.998     -55.247      55.083
ar.S.L24       0.3049     19.397      0.016      0.987     -37.713      38.323
ar.S.L36       0.0160      0.860      0.019      0.985      -1.669       1.701
ma.S.L12       0.0421     28.146      0.001      0.999     -55.123      55.207
ma.S.L24      -0.3113     18.309     -0.017      0.986     -36.196      35.573
sigma2      6.554e+06      0.465   1.41e+07      0.000    6.55e+06    6.55e+06
==============================================================================
===
Ljung-Box (L1) (Q):                   1.90   Jarque-Bera (JB):
123.05
Prob(Q):                              0.17   Prob(JB):
0.00
Heteroskedasticity (H):               2.14   Skew:
0.13
Prob(H) (two-sided):                  0.00   Kurtosis:
6.49
==============================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
[2] Covariance matrix is singular or near-singular, with condition number
9.53e+22. Standard errors may be unstable.
```



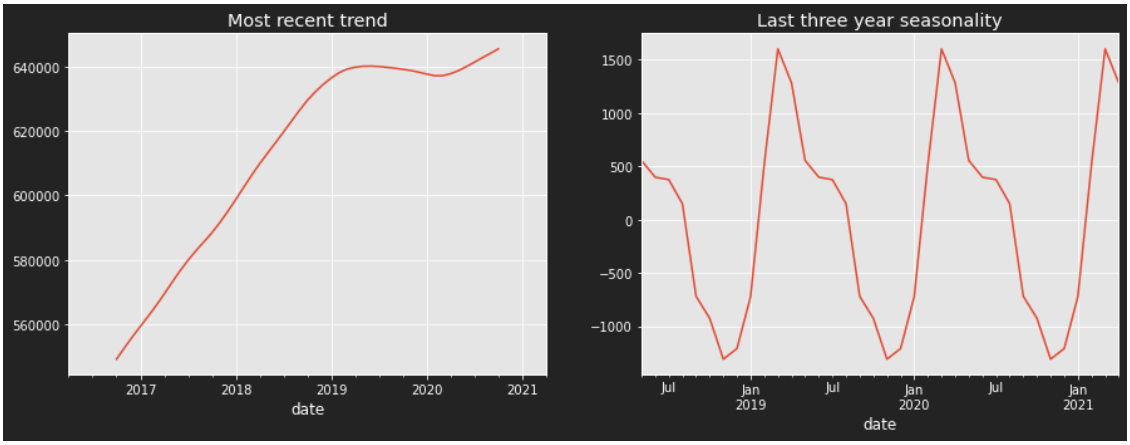
Prediction:

```
Root Mean Squared Error of test and prediction: 48000.75378597782
Mean Squared Error: 2304072364.0220637
Mean Absolute Error: 45685.20620194418
```

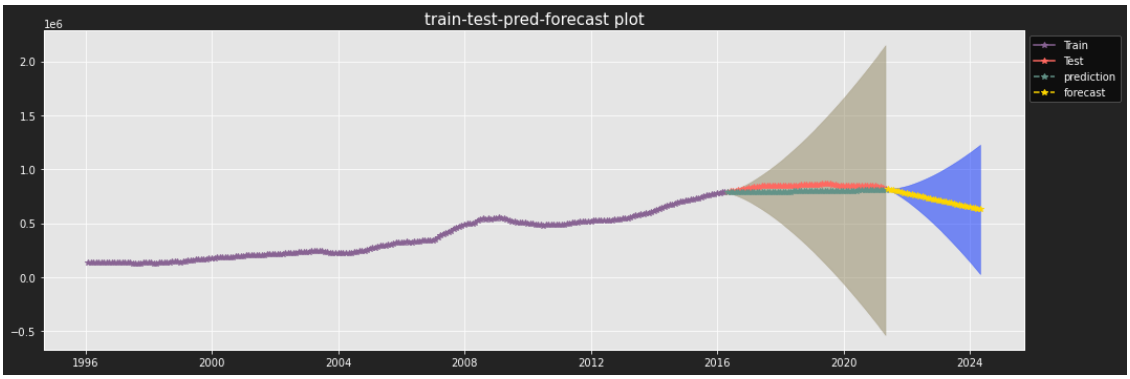




Insights:



Overall model performance and projected ROI:



	zipcode	mean_forecasted_roi	…	upper_forecasted_roi	std_forecasted_roi
0	11101	-23.49	…	49.46	72.95

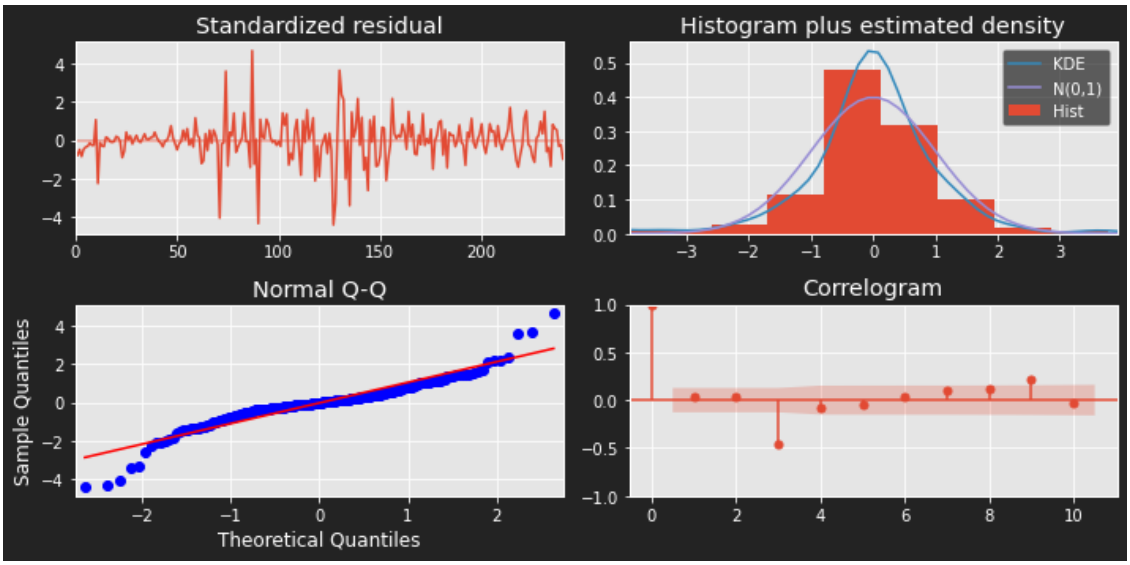
[1 rows x 5 columns]

Report of 11354

Model Used:

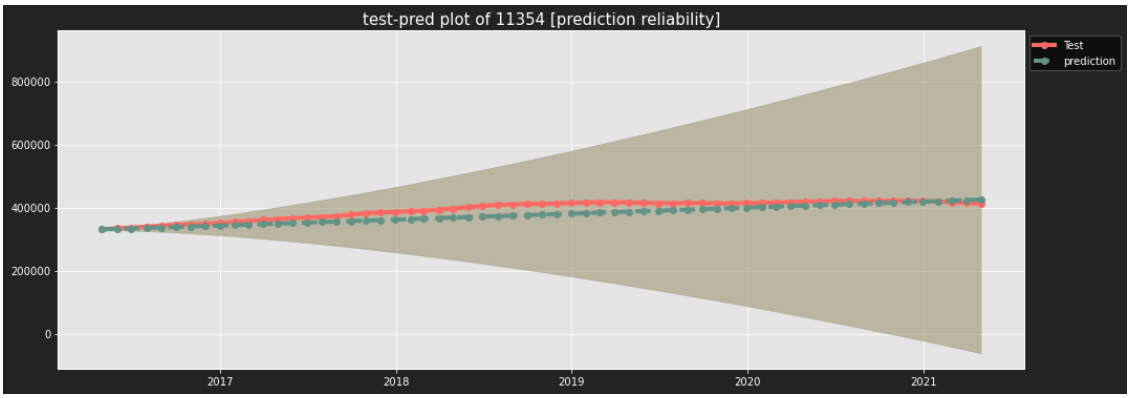
```
ARIMA(order=(1, 2, 1), out_of_sample_size=12, scoring_args={},
      seasonal_order=(0, 0, 2, 12), suppress_warnings=True,
```

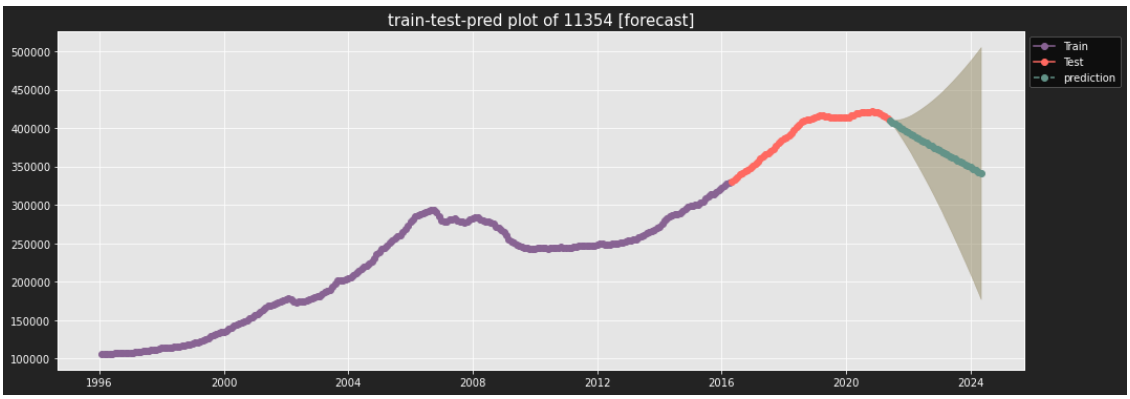
```
      with_intercept=False)
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                        y   No. Observations:
243
Model:             SARIMAX(1, 2, 1)x(0, 0, [1, 2], 12)   Log Likelihood
-1992.361
Date:                                 Sun, 20 Jun 2021   AIC
3994.721
Time:                                         03:21:06   BIC
4012.145
Sample:                                             0   HQIC
4001.741
                                                - 243
Covariance Type:                                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.8180      0.217      3.778      0.000       0.394       1.242
ma.L1         -0.8547      0.205     -4.178      0.000      -1.256      -0.454
ma.S.L12      -0.0549      0.013     -4.280      0.000      -0.080      -0.030
ma.S.L24      -0.0162      0.024     -0.679      0.497      -0.063       0.031
sigma2      7.528e+05    3.4e+04     22.110      0.000    6.86e+05     8.2e+05
==============================================================================
===
Ljung-Box (L1) (Q):                   0.19   Jarque-Bera (JB):
220.53
Prob(Q):                              0.66   Prob(JB):
0.00
Heteroskedasticity (H):               0.85   Skew:
-0.35
Prob(H) (two-sided):                  0.46   Kurtosis:
7.63
==============================================================================
===

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



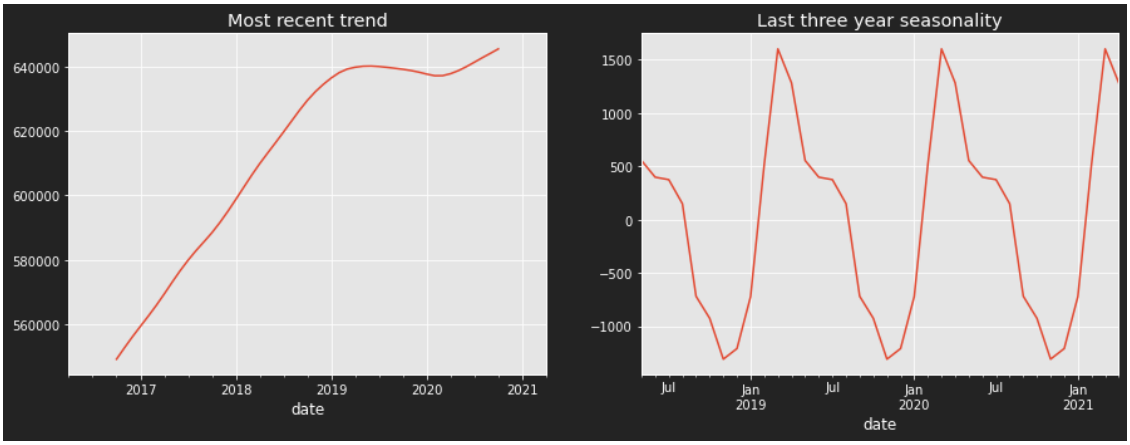
Prediction:

```
Root Mean Squared Error of test and prediction: 20263.737033321657
Mean Squared Error: 410619038.55561155
Mean Absolute Error: 17275.420481853587
```

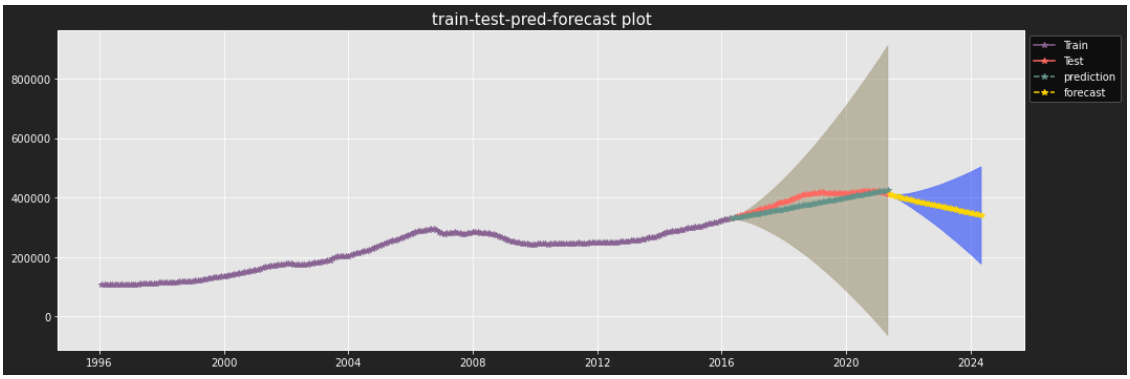




Insights:



Overall model performance and projected ROI:



	zipcode	mean_forecasted_roi	…	upper_forecasted_roi	std_forecasted_roi
0	11354	-17.11	…	22.69	39.8

[1 rows x 5 columns]

[58]:
```
# statistical information
fn.output_df(investigation, results_) # risk is the relative standard deviation␣
 ↪of ROI
```

[58]:

ZipCode	aic	…	three_year_projected_upper_roi
11415	4017.53	…	7.98
11375	4055.87	…	35.14

```
11101    4482.02  …                              49.46
11354    3994.72  …                              22.69

[4 rows x 12 columns]
```

Model fit is good.

3.5 Under performing models

[49]:
```
# Isolating high variance of prediction of test data and prediction to find out␣
 ↪poor model fit
under_performing = fn.output_df(zipcode_list, results_)
under_performing['und_per_model'] = abs(under_performing.test_roi -
                                        under_performing.pred_roi)
```

[50]:
```
under_performing.sort_values(by='und_per_model')[-8:]
```

[50]:
```
              aic      bic  …  three_year_projected_upper_roi  und_per_model
ZipCode                     …
11101    4482.02  4509.90  …                            49.46          20.03
11354    3994.72  4012.15  …                            22.69          20.43
11428    4232.73  4260.60  …                            81.53          20.72
11423    4247.35  4271.74  …                            87.65          21.34
11693    4509.50  4516.48  …                            10.15          23.09
11375    4055.87  4062.84  …                            35.14          40.40
11104    4041.80  4055.73  …                            84.23          42.01
11415    4017.53  4034.95  …                             7.98          75.36

[8 rows x 13 columns]
```

[51]:
```
und_per_model = under_performing.sort_values(by='und_per_model')[-8:].index
```

[52]:
```
fn.model_report(und_per_model,
                results_,
                show_model_performance=True,
                show_train_fit=True,
                show_prediction=True,test_conf_int=True,
                show_detailed_prediction=True)
```

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
 Report of 11101
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
 Model Used:

ARIMA(order=(1, 2, 0), out_of_sample_size=12, scoring_args={},
      seasonal_order=(3, 0, 2, 12), suppress_warnings=True)
```

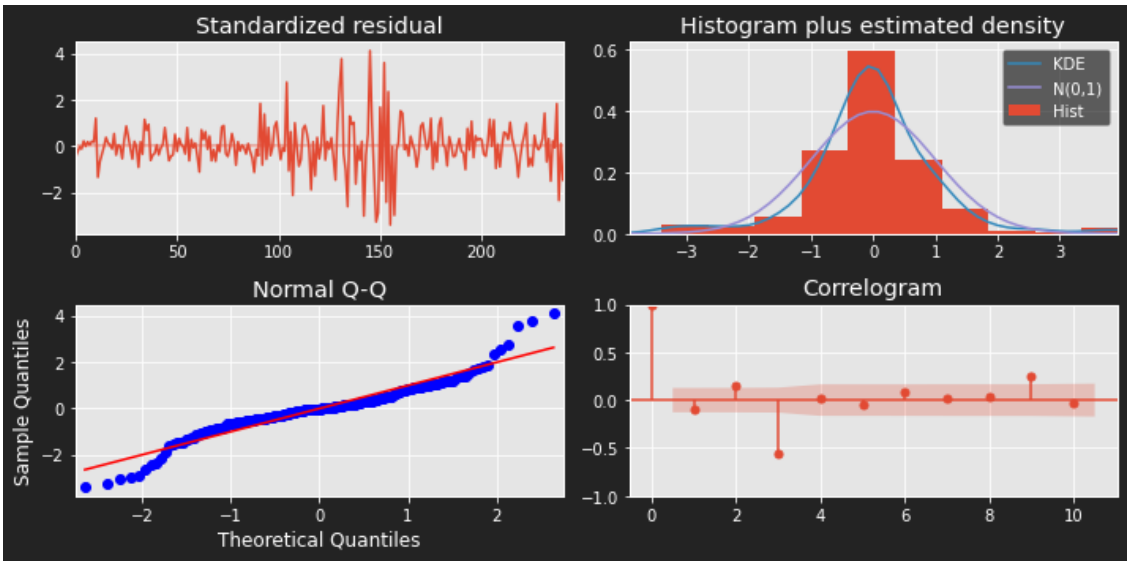
SARIMAX Results

Dep. Variable:	y	No. Observations:	243
Model:	SARIMAX(1, 2, 0)x(3, 0, [1, 2], 12)	Log Likelihood	-2233.012
Date:	Sun, 20 Jun 2021	AIC	4482.024
Time:	03:12:11	BIC	4509.902
Sample:	0	HQIC	4493.255
	- 243		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	22.2906	1399.221	0.016	0.987	-2720.132	2764.713
ar.L1	-0.0169	0.032	-0.524	0.600	-0.080	0.046
ar.S.L12	-0.0818	28.146	-0.003	0.998	-55.247	55.083
ar.S.L24	0.3049	19.397	0.016	0.987	-37.713	38.323
ar.S.L36	0.0160	0.860	0.019	0.985	-1.669	1.701
ma.S.L12	0.0421	28.146	0.001	0.999	-55.123	55.207
ma.S.L24	-0.3113	18.309	-0.017	0.986	-36.196	35.573
sigma2	6.554e+06	0.465	1.41e+07	0.000	6.55e+06	6.55e+06

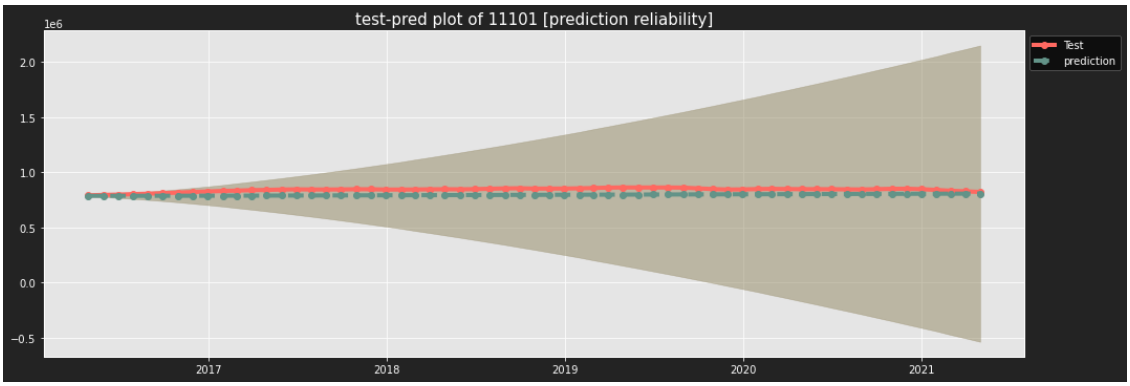
Ljung-Box (L1) (Q):	1.90	Jarque-Bera (JB):	123.05
Prob(Q):	0.17	Prob(JB):	0.00
Heteroskedasticity (H):	2.14	Skew:	0.13
Prob(H) (two-sided):	0.00	Kurtosis:	6.49

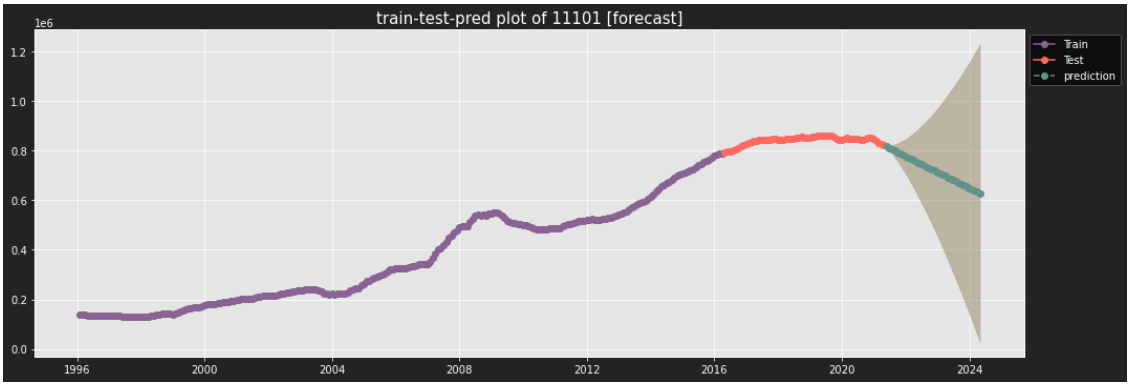
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
[2] Covariance matrix is singular or near-singular, with condition number 9.53e+22. Standard errors may be unstable.



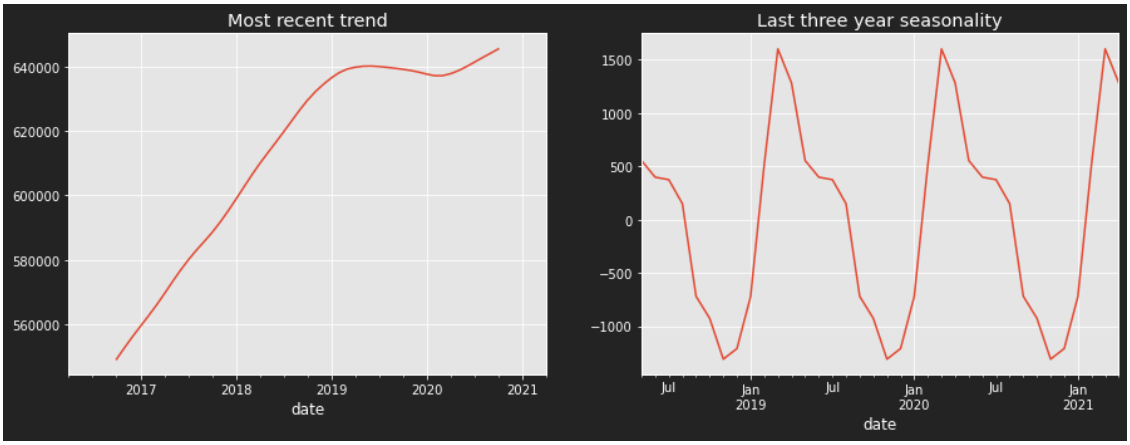
Prediction:

```
Root Mean Squared Error of test and prediction: 48000.75378597782
Mean Squared Error: 2304072364.0220637
Mean Absolute Error: 45685.20620194418
```

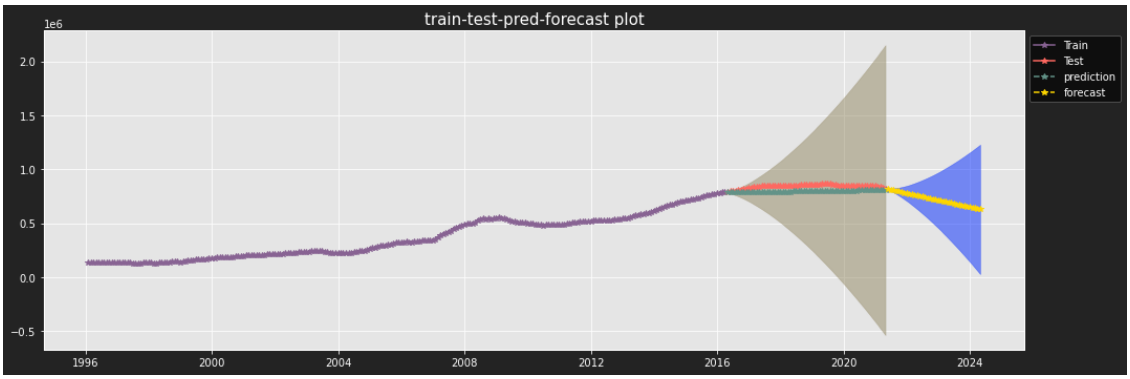




Insights:



Overall model performance and projected ROI:



	zipcode	mean_forecasted_roi	…	upper_forecasted_roi	std_forecasted_roi
0	11101	-23.49	…	49.46	72.95

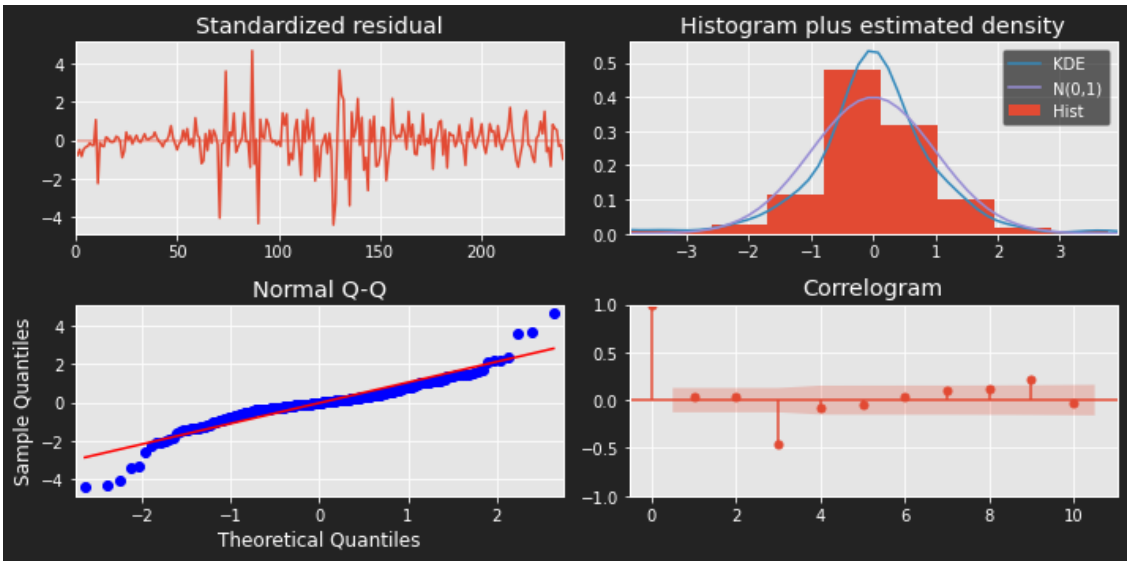
[1 rows x 5 columns]

Report of 11354

Model Used:

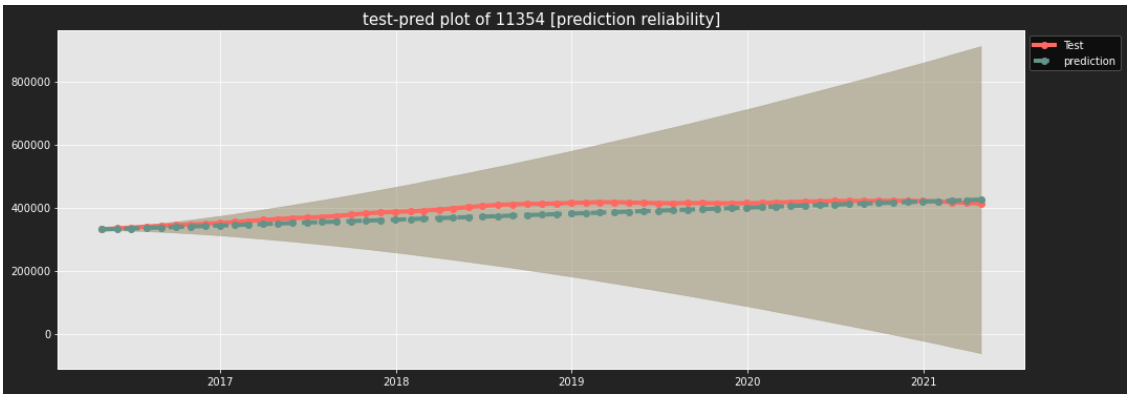
```
ARIMA(order=(1, 2, 1), out_of_sample_size=12, scoring_args={},
      seasonal_order=(0, 0, 2, 12), suppress_warnings=True,
```

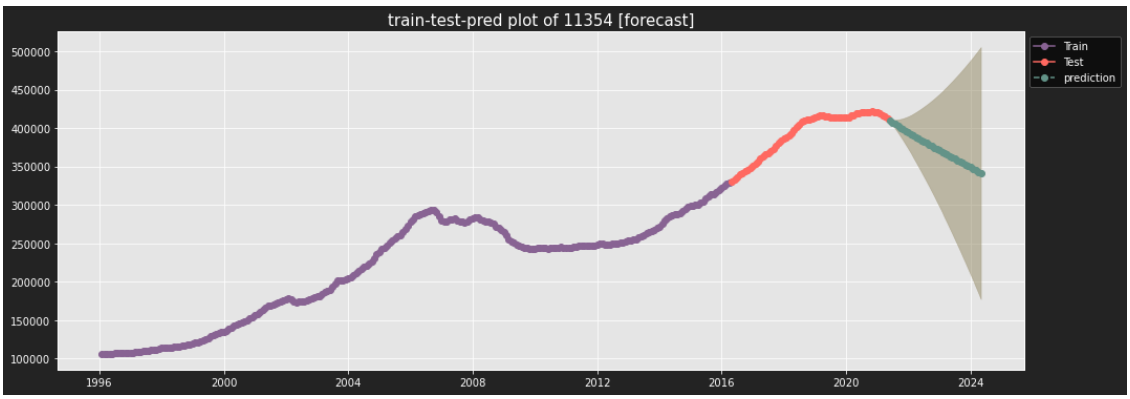
```
      with_intercept=False)
                                      SARIMAX Results
==========================================================================================
Dep. Variable:                                        y   No. Observations:
243
Model:             SARIMAX(1, 2, 1)x(0, 0, [1, 2], 12)   Log Likelihood
-1992.361
Date:                                 Sun, 20 Jun 2021   AIC
3994.721
Time:                                         03:12:13   BIC
4012.145
Sample:                                              0   HQIC
4001.741
                                                 - 243
Covariance Type:                                   opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.8180      0.217      3.778      0.000       0.394       1.242
ma.L1         -0.8547      0.205     -4.178      0.000      -1.256      -0.454
ma.S.L12      -0.0549      0.013     -4.280      0.000      -0.080      -0.030
ma.S.L24      -0.0162      0.024     -0.679      0.497      -0.063       0.031
sigma2      7.528e+05    3.4e+04     22.110      0.000    6.86e+05     8.2e+05
==============================================================================
===
Ljung-Box (L1) (Q):                   0.19   Jarque-Bera (JB):
220.53
Prob(Q):                              0.66   Prob(JB):
0.00
Heteroskedasticity (H):               0.85   Skew:
-0.35
Prob(H) (two-sided):                  0.46   Kurtosis:
7.63
==============================================================================
===

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



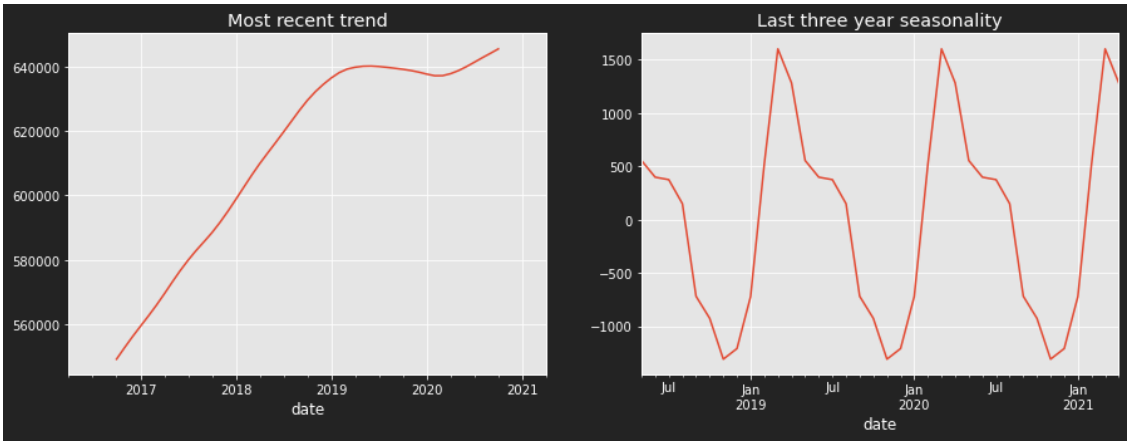
Prediction:

```
Root Mean Squared Error of test and prediction: 20263.737033321657
Mean Squared Error: 410619038.55561155
Mean Absolute Error: 17275.420481853587
```

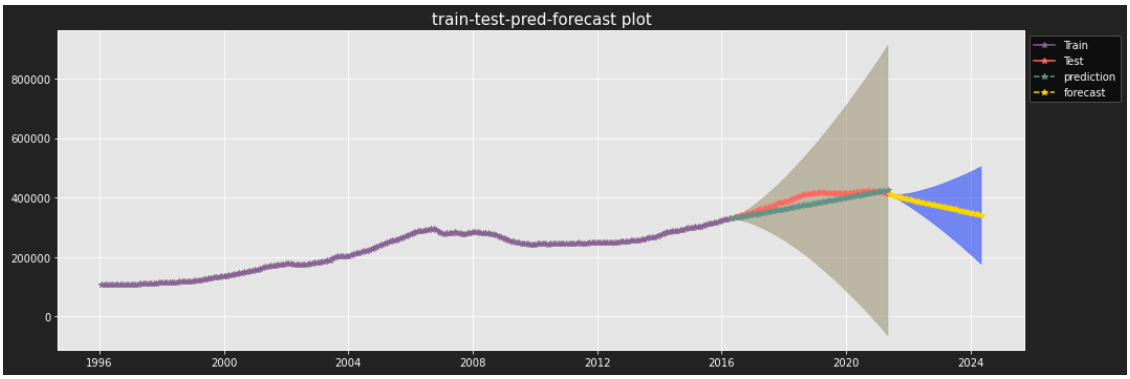




Insights:



Overall model performance and projected ROI:




```
  zipcode  mean_forecasted_roi  …  upper_forecasted_roi  std_forecasted_roi
0   11354                -17.11  …                 22.69                39.8

[1 rows x 5 columns]
```

Report of 11428

Model Used:

```
ARIMA(order=(1, 2, 3), out_of_sample_size=12, scoring_args={},
      seasonal_order=(2, 0, 0, 12), suppress_warnings=True)
```

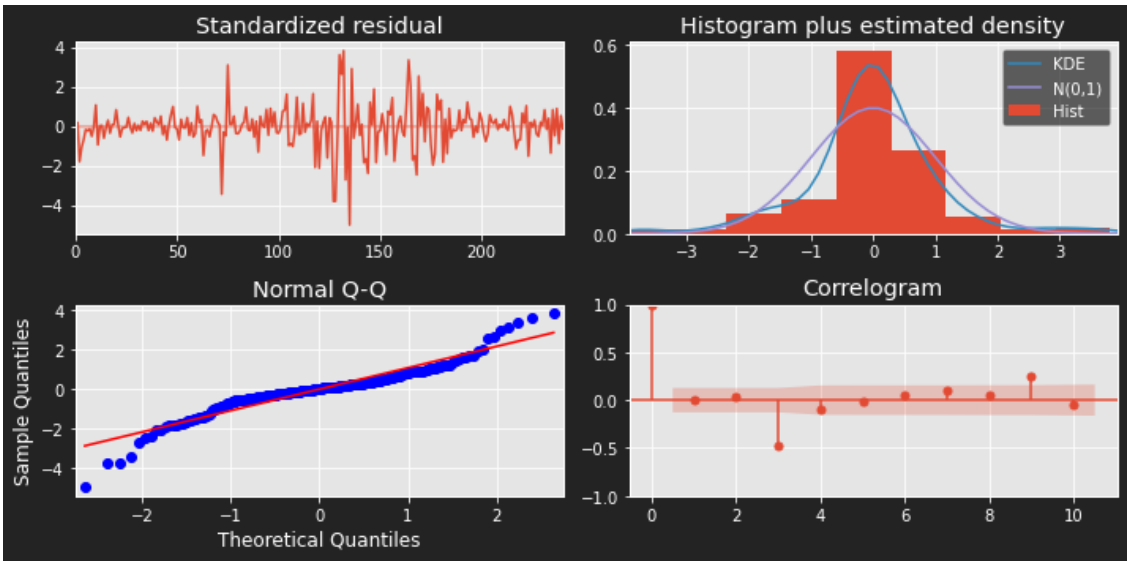
SARIMAX Results

Dep. Variable:	y	No. Observations:	243
Model:	SARIMAX(1, 2, 3)x(2, 0, [], 12)	Log Likelihood	-2108.363
Date:	Sun, 20 Jun 2021	AIC	4232.725
Time:	03:12:15	BIC	4260.603
Sample:	0	HQIC	4243.957
	- 243		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	17.4917	40.570	0.431	0.666	-62.024	97.007
ar.L1	0.5195	0.277	1.873	0.061	-0.024	1.063
ma.L1	-0.5368	0.286	-1.877	0.061	-1.097	0.024
ma.L2	-0.0136	0.042	-0.320	0.749	-0.096	0.069
ma.L3	-0.0292	0.019	-1.528	0.127	-0.067	0.008
ar.S.L12	-0.0179	0.010	-1.763	0.078	-0.038	0.002
ar.S.L24	0.0071	0.028	0.256	0.798	-0.047	0.061
sigma2	1.946e+06	9.74e+04	19.970	0.000	1.75e+06	2.14e+06

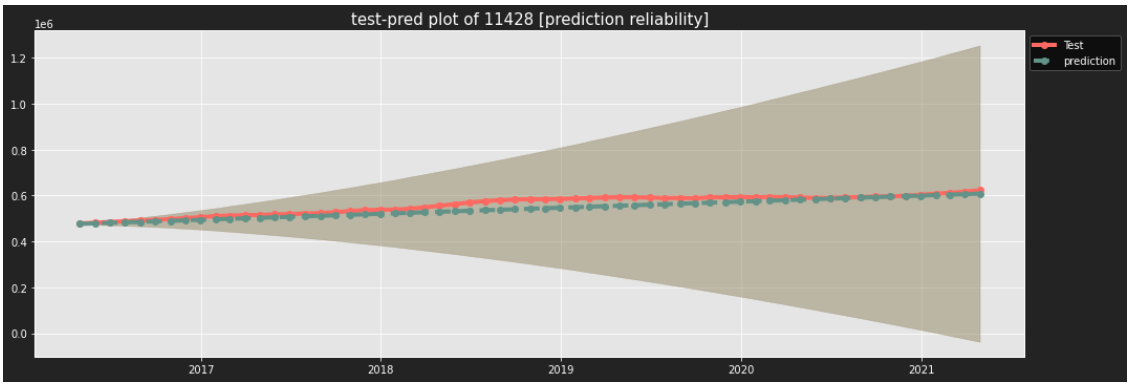
Ljung-Box (L1) (Q):	0.02	Jarque-Bera (JB):	160.18
Prob(Q):	0.90	Prob(JB):	0.00
Heteroskedasticity (H):	1.91	Skew:	-0.34
Prob(H) (two-sided):	0.00	Kurtosis:	6.94

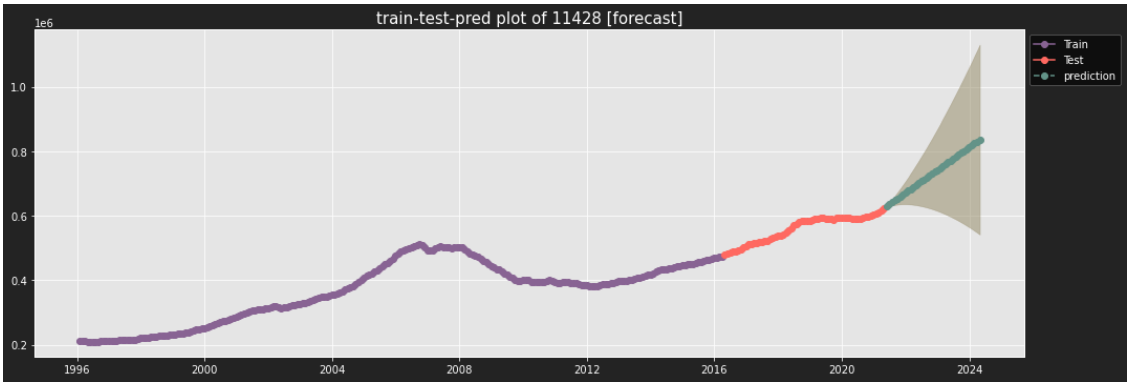
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).



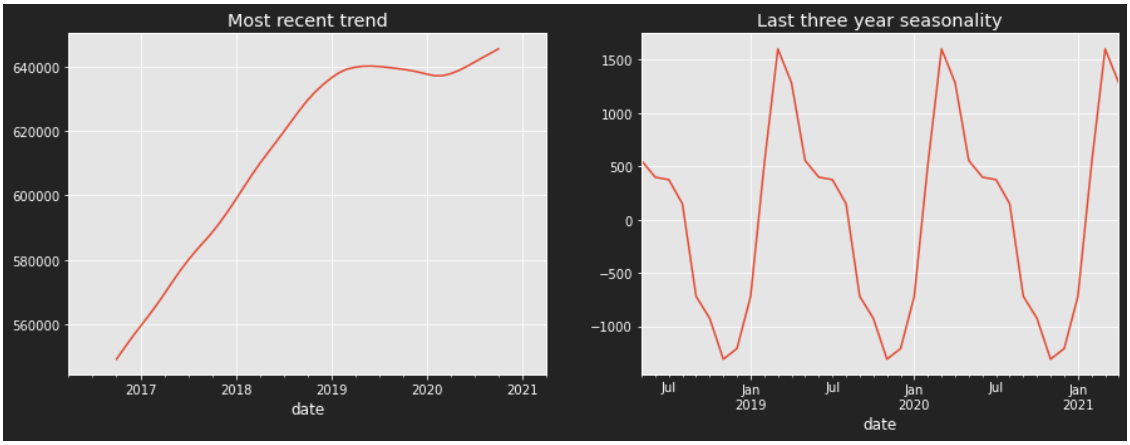
Prediction:

```
Root Mean Squared Error of test and prediction: 21826.030233377074
Mean Squared Error: 476375595.7482901
Mean Absolute Error: 17786.74206469421
```

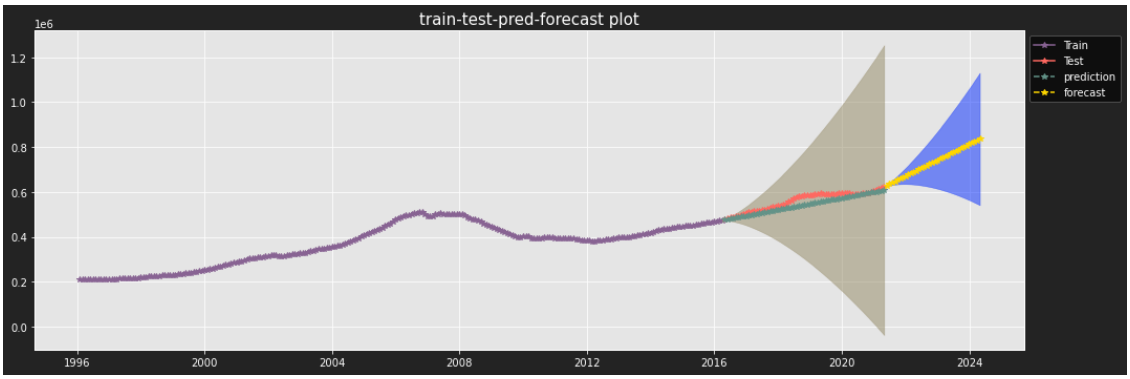




Insights:



Overall model performance and projected ROI:



	zipcode	mean_forecasted_roi	…	upper_forecasted_roi	std_forecasted_roi
0	11428	34.24	…	81.53	47.29

[1 rows x 5 columns]

Report of 11423

Model Used:

```
ARIMA(order=(3, 2, 1), out_of_sample_size=12, scoring_args={},
      seasonal_order=(0, 0, 1, 12), suppress_warnings=True)
```

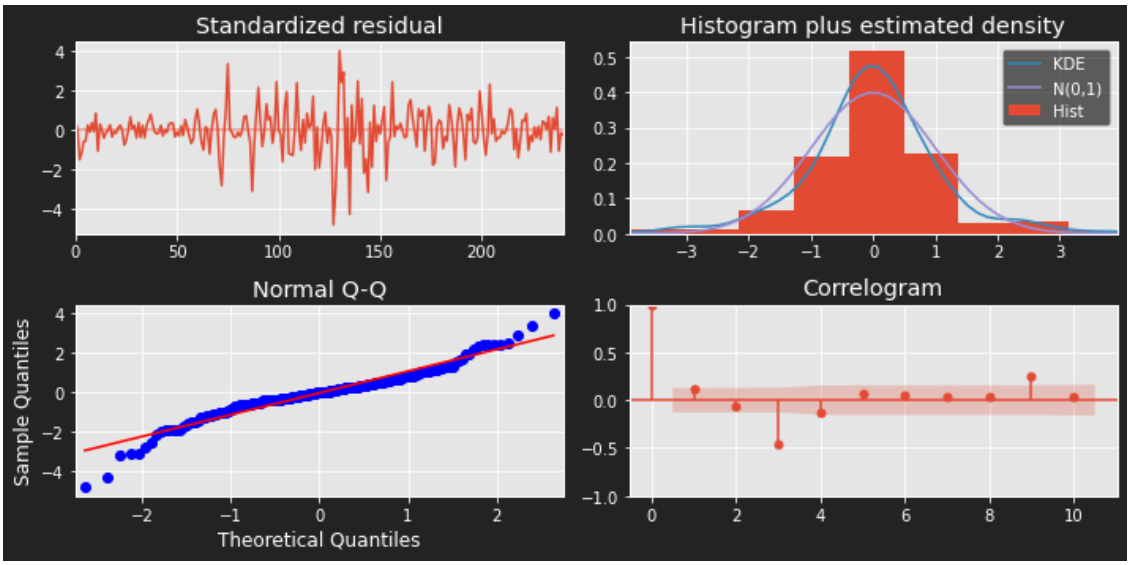
SARIMAX Results

Dep. Variable:	y	No. Observations:	243
Model:	SARIMAX(3, 2, 1)x(0, 0, 1, 12)	Log Likelihood	-2116.675
Date:	Sun, 20 Jun 2021	AIC	4247.351
Time:	03:12:16	BIC	4271.744
Sample:	0	HQIC	4257.178
	- 243		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	11.5646	37.425	0.309	0.757	-61.787	84.916
ar.L1	0.5965	0.314	1.897	0.058	-0.020	1.213
ar.L2	-0.0115	0.028	-0.418	0.676	-0.065	0.042
ar.L3	-0.0231	0.021	-1.081	0.280	-0.065	0.019
ma.L1	-0.6076	0.321	-1.894	0.058	-1.236	0.021
ma.S.L12	-0.0129	0.009	-1.478	0.139	-0.030	0.004
sigma2	1.988e+06	1.02e+05	19.399	0.000	1.79e+06	2.19e+06

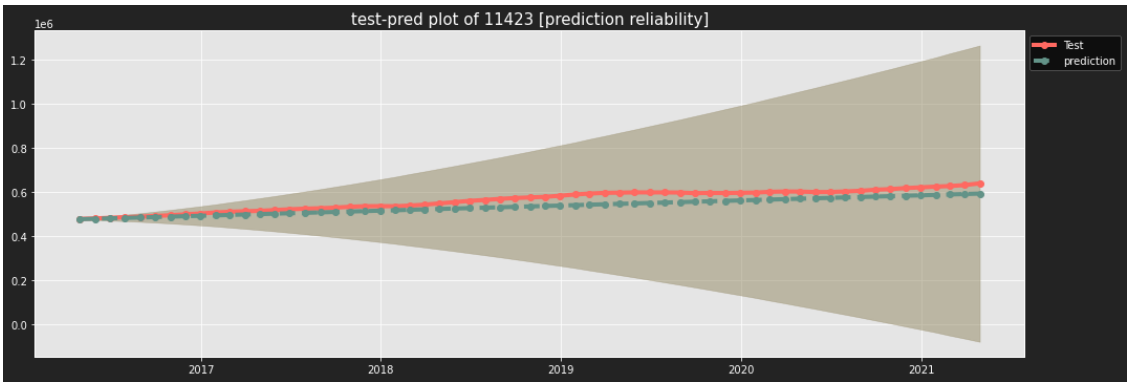
Ljung-Box (L1) (Q):	3.21	Jarque-Bera (JB):	102.79
Prob(Q):	0.07	Prob(JB):	0.00
Heteroskedasticity (H):	1.36	Skew:	-0.35
Prob(H) (two-sided):	0.18	Kurtosis:	6.12

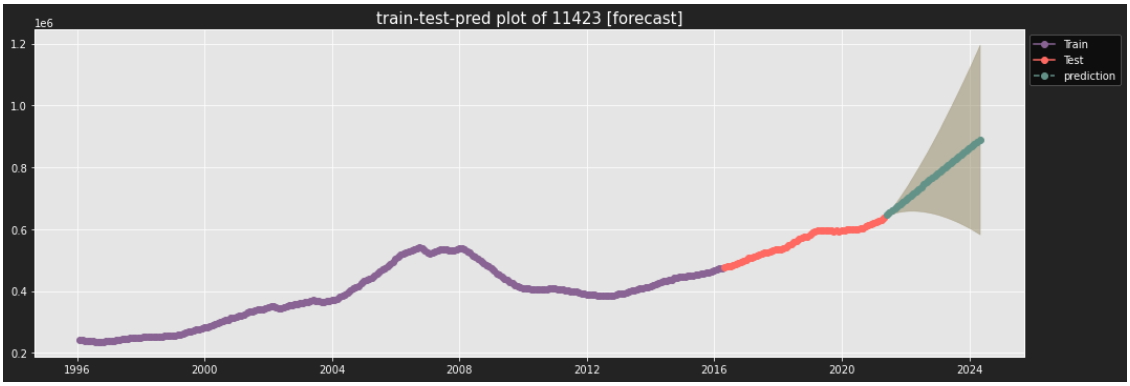
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).



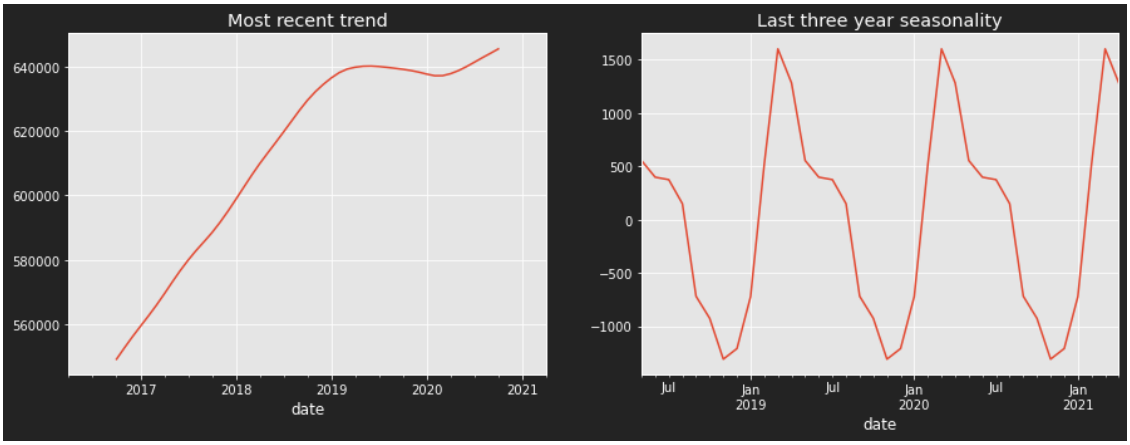
Prediction:

```
Root Mean Squared Error of test and prediction: 31539.042281473074
Mean Squared Error: 994711188.0325463
Mean Absolute Error: 28175.240673024087
```

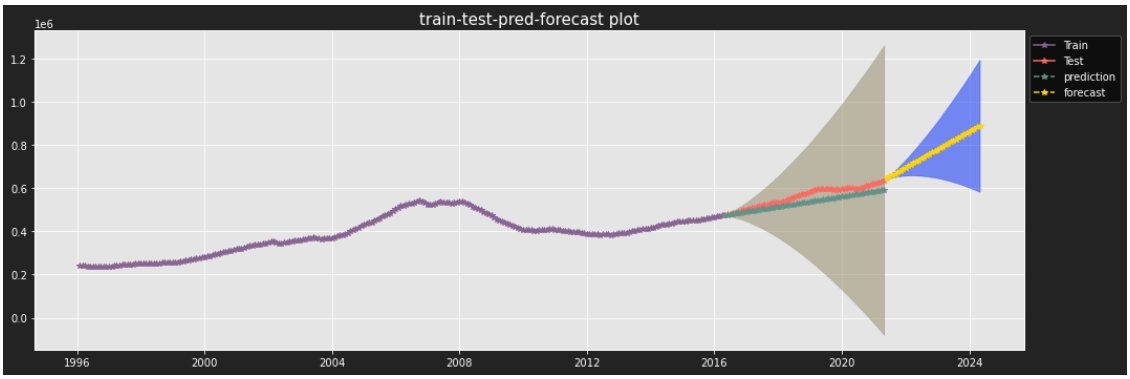




Insights:



Overall model performance and projected ROI:



	zipcode	mean_forecasted_roi	…	upper_forecasted_roi	std_forecasted_roi
0	11423	39.55	…	87.65	48.11

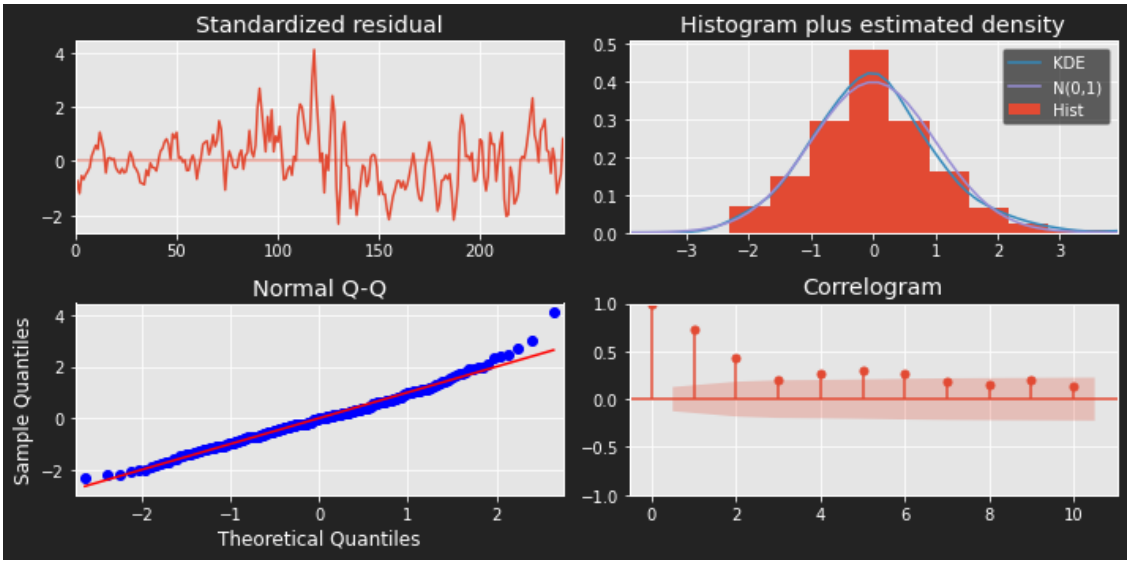
[1 rows x 5 columns]

Report of 11693

Model Used:

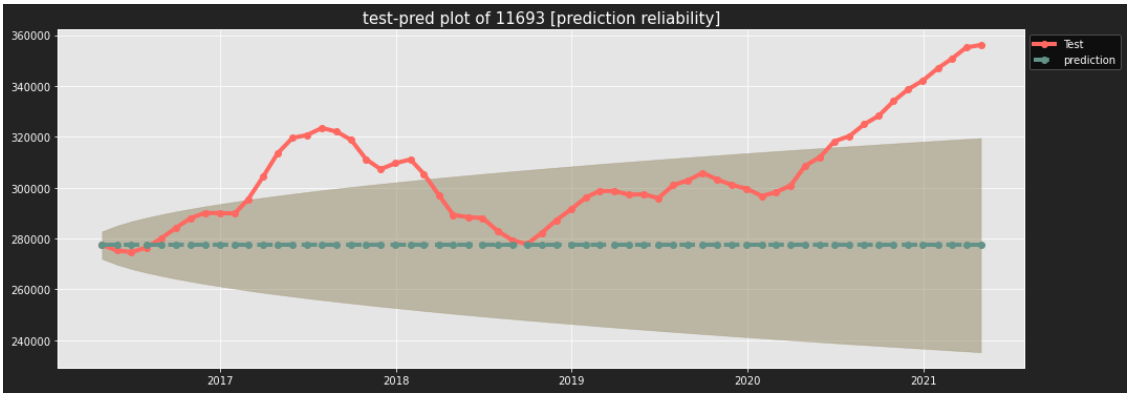
```
ARIMA(order=(0, 1, 0), out_of_sample_size=12, scoring_args={},
      seasonal_order=(0, 0, 0, 12), suppress_warnings=True)
```

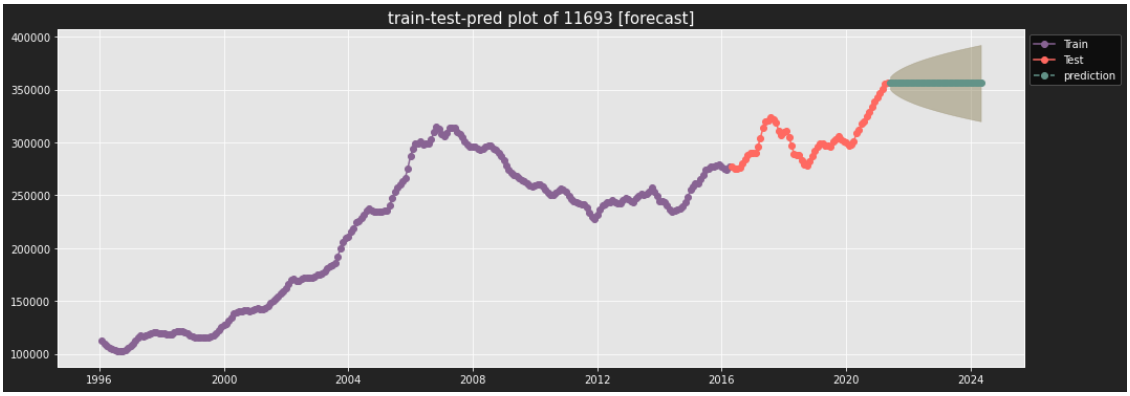
```
                               SARIMAX Results
==========================================================================================
Dep. Variable:                            y   No. Observations:                  243
Model:                 SARIMAX(0, 1, 0)   Log Likelihood               -2252.749
Date:                  Sun, 20 Jun 2021   AIC                           4509.499
Time:                          03:12:18   BIC                           4516.477
Sample:                               0   HQIC                          4512.310
                                  - 243
Covariance Type:                    opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept    649.6043    181.926      3.571      0.000     293.035    1006.173
sigma2      7.222e+06    5.7e+05     12.676      0.000    6.11e+06    8.34e+06
==============================================================================
===
Ljung-Box (L1) (Q):                 130.57   Jarque-Bera (JB):
18.71
Prob(Q):                              0.00   Prob(JB):
0.00
Heteroskedasticity (H):               3.09   Skew:
0.48
Prob(H) (two-sided):                  0.00   Kurtosis:
3.96
==============================================================================
===

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



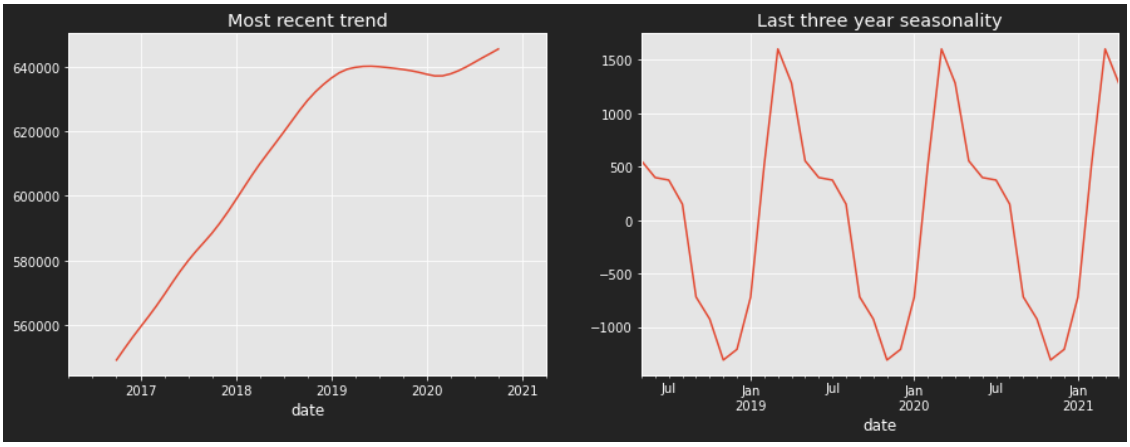
Prediction:

```
Root Mean Squared Error of test and prediction: 33917.68629186611
Mean Squared Error: 1150409443.3934426
Mean Absolute Error: 27405.196721311477
```

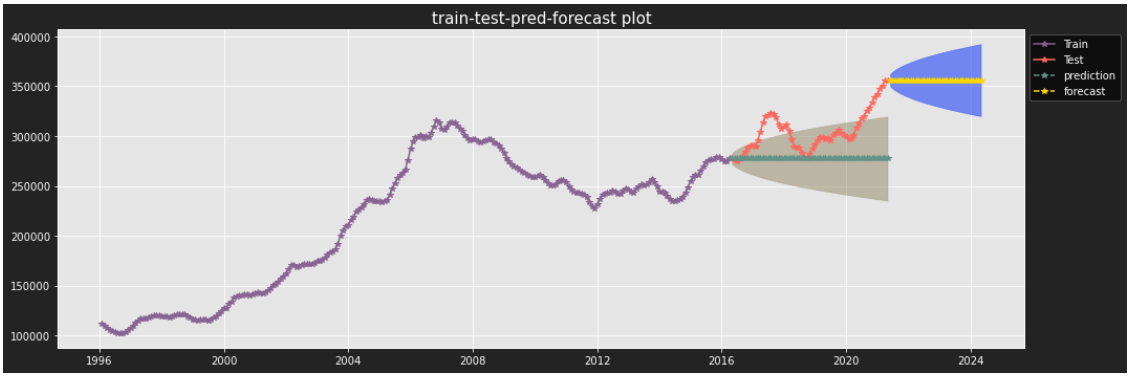




Insights:



Overall model performance and projected ROI:



	zipcode	mean_forecasted_roi	…	upper_forecasted_roi	std_forecasted_roi
0	11693	0.0	…	10.15	10.15

[1 rows x 5 columns]

Report of 11375

Model Used:

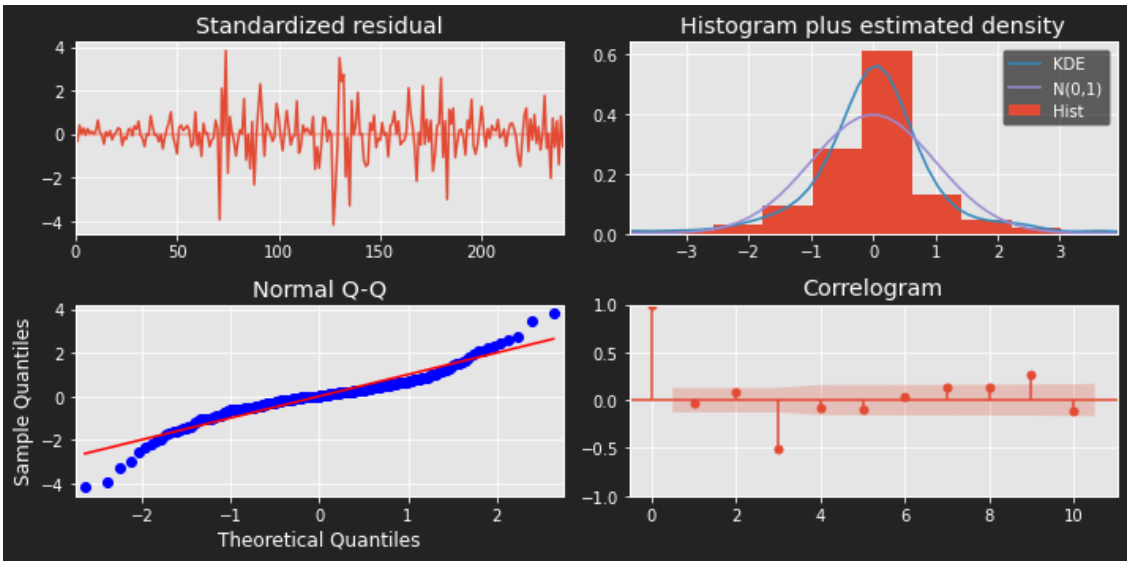
```
ARIMA(order=(1, 2, 0), out_of_sample_size=12, scoring_args={},
      seasonal_order=(0, 0, 0, 12), suppress_warnings=True,
      with_intercept=False)
```

SARIMAX Results

Dep. Variable:	y	No. Observations:	243
Model:	SARIMAX(1, 2, 0)	Log Likelihood	-2025.933
Date:	Sun, 20 Jun 2021	AIC	4055.867
Time:	03:12:19	BIC	4062.837
Sample:	0	HQIC	4058.675
	- 243		
Covariance Type:	opg		

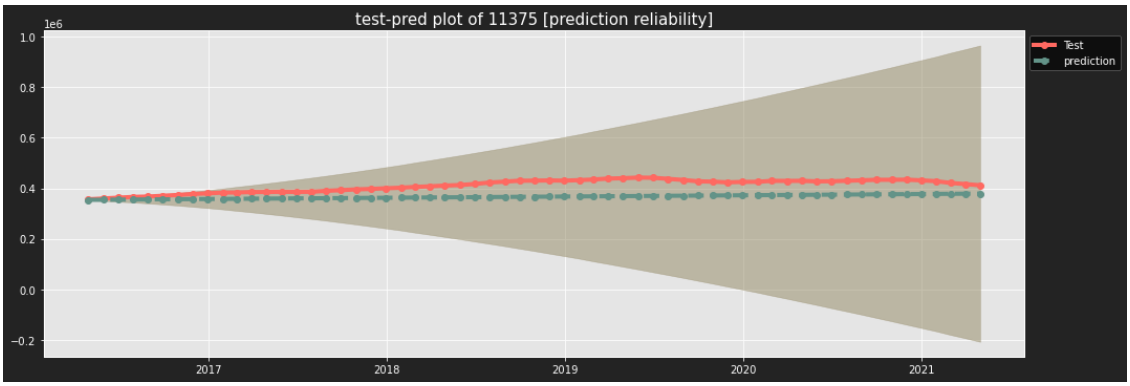
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.0092	0.042	-0.218	0.827	-0.092	0.073
sigma2	1.175e+06	6.36e+04	18.479	0.000	1.05e+06	1.3e+06

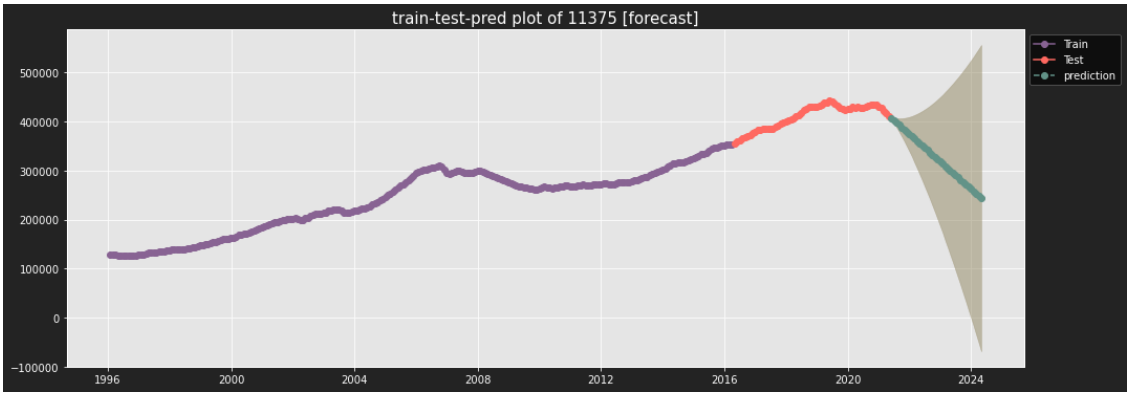
```
===
Ljung-Box (L1) (Q):                   0.29   Jarque-Bera (JB):
140.75
Prob(Q):                              0.59   Prob(JB):
0.00
Heteroskedasticity (H):               1.42   Skew:
-0.27
Prob(H) (two-sided):                  0.12   Kurtosis:
6.70
================================================================================
===

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



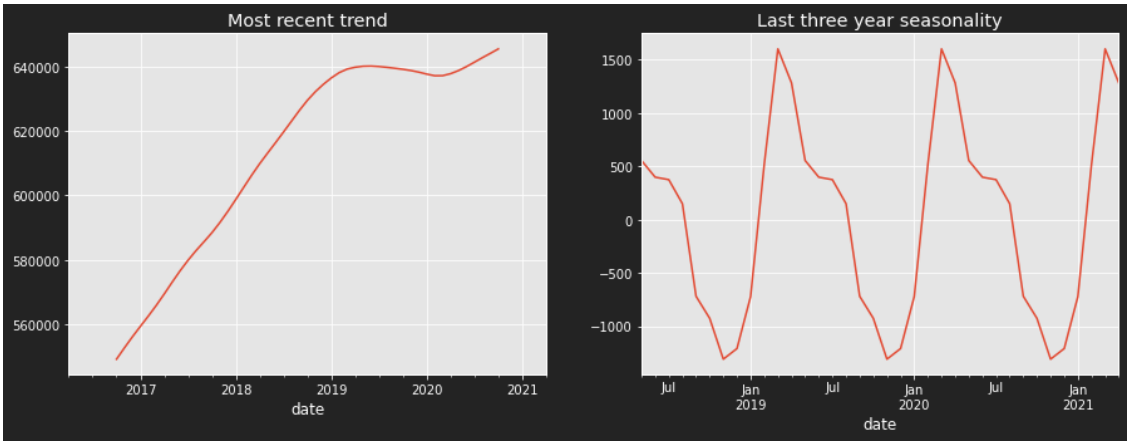
Prediction:

```
Root Mean Squared Error of test and prediction: 47935.52415165556
Mean Squared Error: 2297814475.693954
Mean Absolute Error: 44126.14249132199
```

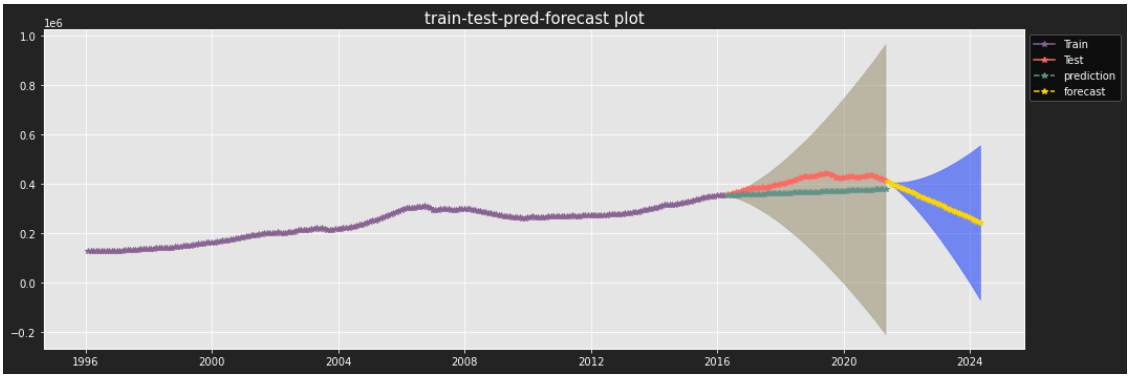




Insights:



Overall model performance and projected ROI:




```
  zipcode  mean_forecasted_roi  …  upper_forecasted_roi  std_forecasted_roi
0   11375               -40.78  …                 35.14               75.92

[1 rows x 5 columns]
```

Report of 11104

Model Used:

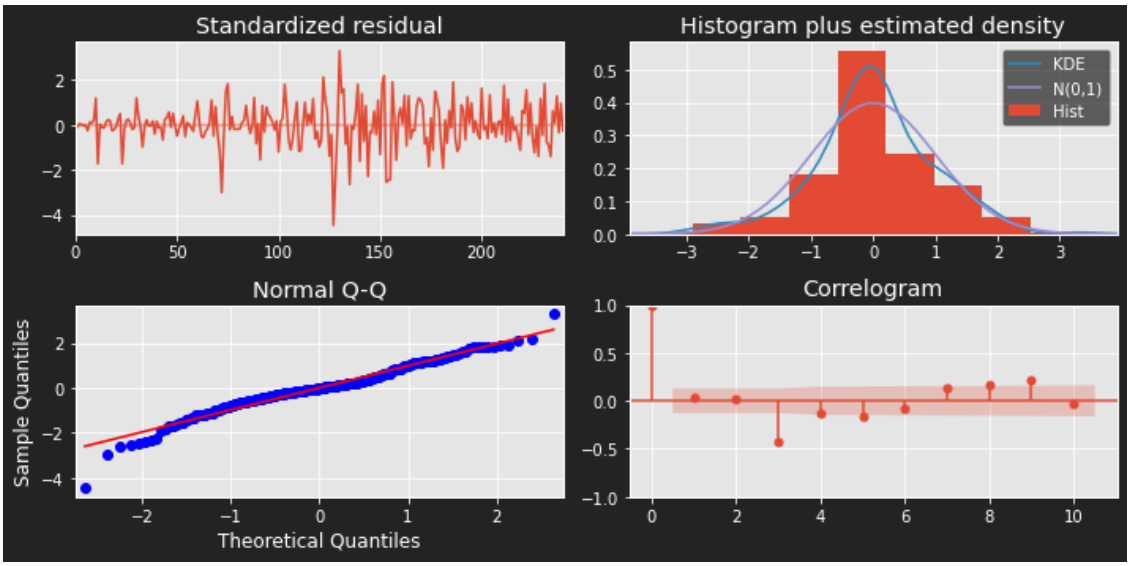
```
ARIMA(order=(0, 2, 0), out_of_sample_size=12, scoring_args={},
      seasonal_order=(0, 0, 2, 12), suppress_warnings=True)
```

SARIMAX Results

Dep. Variable:	y	No. Observations:	243
Model:	SARIMAX(0, 2, 0)x(0, 0, [1, 2], 12)	Log Likelihood	-2016.898
Date:	Sun, 20 Jun 2021	AIC	4041.795
Time:	03:12:21	BIC	4055.734
Sample:	0	HQIC	4047.411
	- 243		
Covariance Type:	opg		

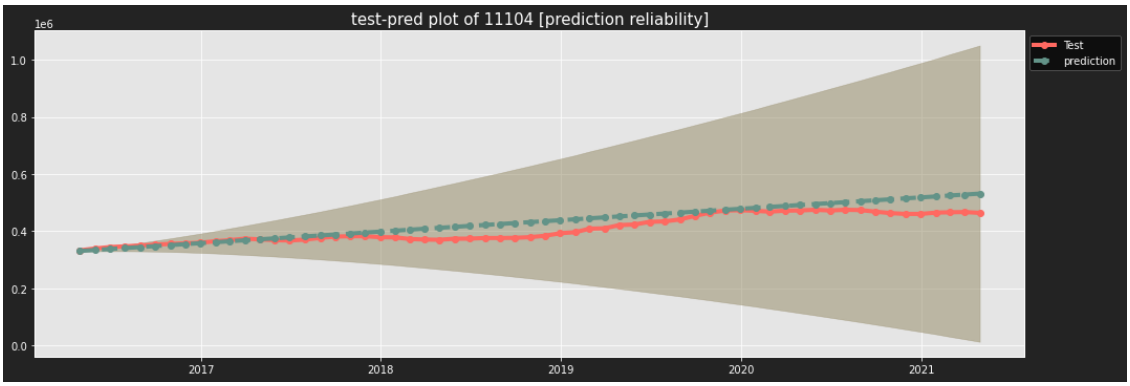
	coef	std err	z	P>|z|	[0.025	0.975]
intercept	6.4236	64.715	0.099	0.921	-120.416	133.263
ma.S.L12	-0.0658	0.018	-3.674	0.000	-0.101	-0.031
ma.S.L24	-0.0371	0.018	-2.082	0.037	-0.072	-0.002

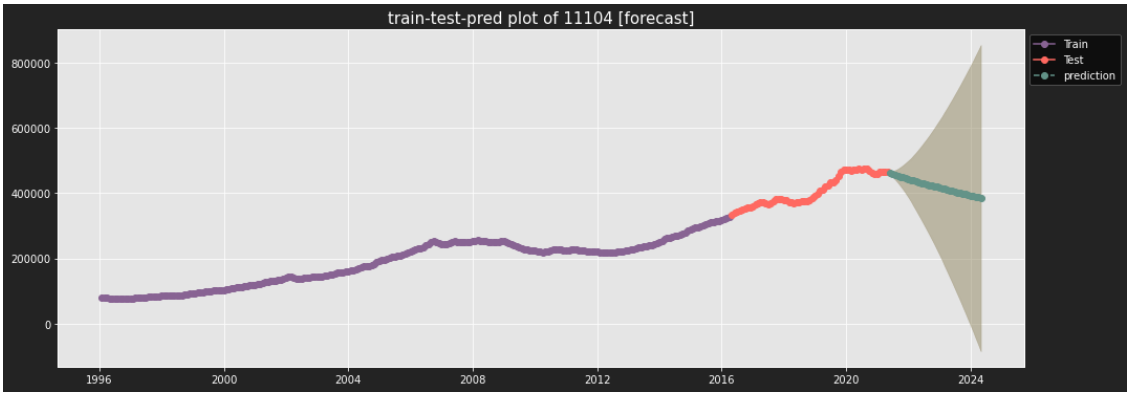
```
sigma2      1.123e+06   7.93e+04     14.161       0.000     9.68e+05     1.28e+06
===================================================================================
===
Ljung-Box (L1) (Q):                   0.35   Jarque-Bera (JB):
49.63
Prob(Q):                              0.56   Prob(JB):
0.00
Heteroskedasticity (H):               1.99   Skew:
-0.44
Prob(H) (two-sided):                  0.00   Kurtosis:
5.04
===================================================================================
===

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



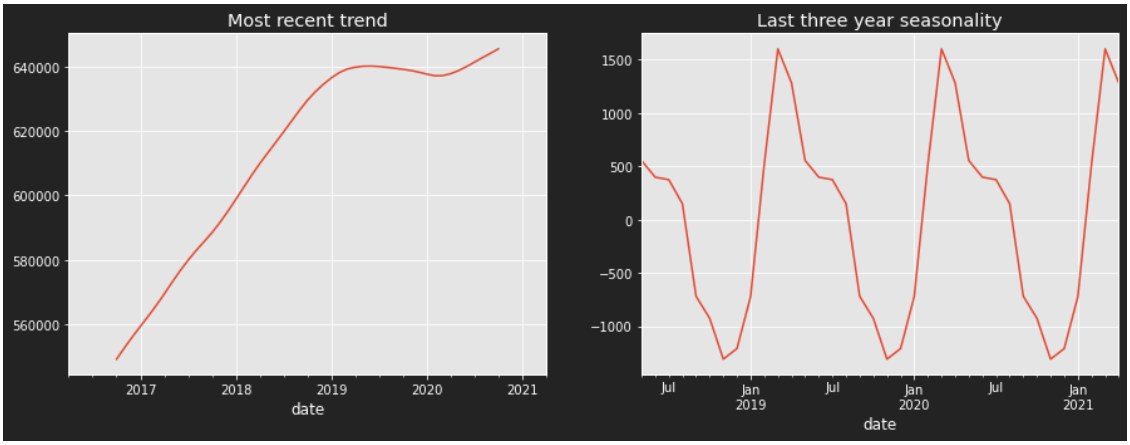
Prediction:

```
Root Mean Squared Error of test and prediction: 31972.81205732297
Mean Squared Error: 1022260710.8528973
Mean Absolute Error: 25470.88689787593
```

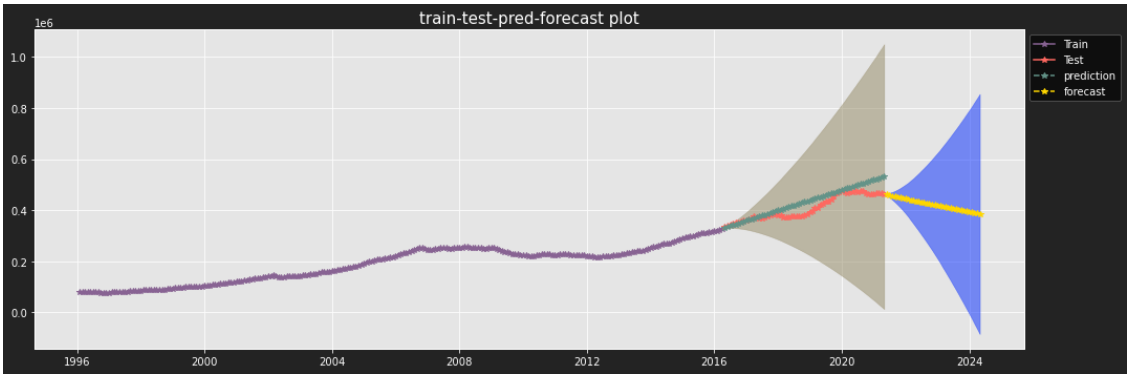




Insights:



Overall model performance and projected ROI:




```
  zipcode  mean_forecasted_roi  …  upper_forecasted_roi  std_forecasted_roi
0   11104                -17.07  …                 84.23               101.3

[1 rows x 5 columns]
```

Report of 11415

Model Used:

```
ARIMA(order=(1, 2, 3), out_of_sample_size=12, scoring_args={},
      seasonal_order=(0, 0, 0, 12), suppress_warnings=True,
      with_intercept=False)
```

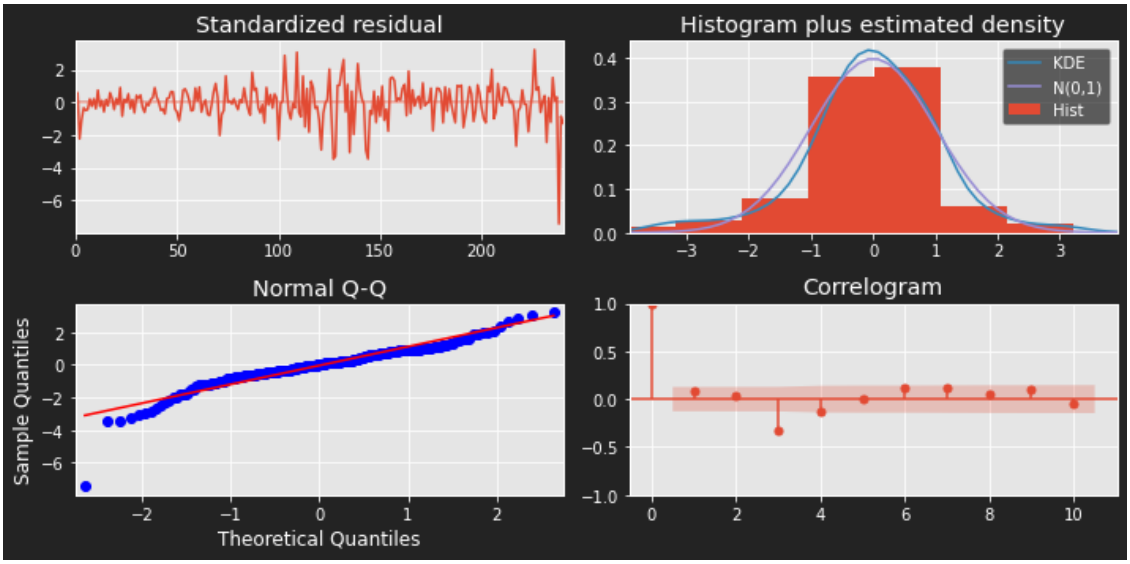
SARIMAX Results

Dep. Variable:	y	No. Observations:	243
Model:	SARIMAX(1, 2, 3)	Log Likelihood	-2003.765
Date:	Sun, 20 Jun 2021	AIC	4017.531
Time:	03:12:23	BIC	4034.955
Sample:	0	HQIC	4024.551
	- 243		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.4475	0.250	1.788	0.074	-0.043	0.938
ma.L1	-0.4490	0.254	-1.768	0.077	-0.947	0.049
ma.L2	-0.0145	0.031	-0.469	0.639	-0.075	0.046
ma.L3	-0.0663	0.017	-3.900	0.000	-0.100	-0.033
sigma2	6.965e+05	2.24e+04	31.097	0.000	6.53e+05	7.4e+05

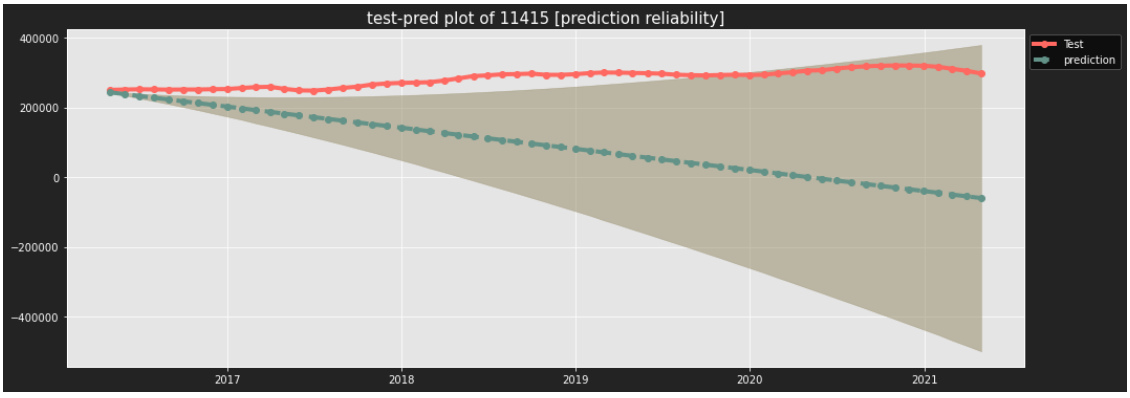
Ljung-Box (L1) (Q):	1.86	Jarque-Bera (JB):	

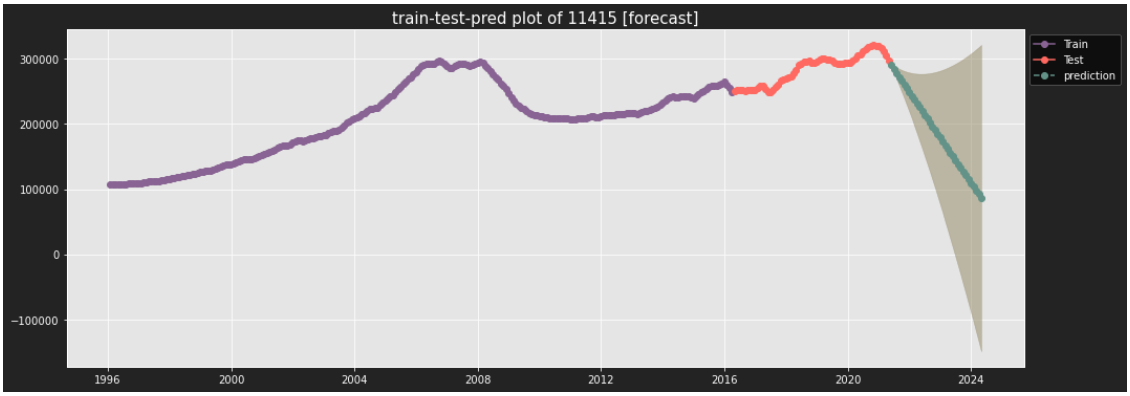
```
573.21
Prob(Q):                                0.17   Prob(JB):
0.00
Heteroskedasticity (H):                 4.21   Skew:
-1.30
Prob(H) (two-sided):                    0.00   Kurtosis:
10.09
================================================================================
===

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



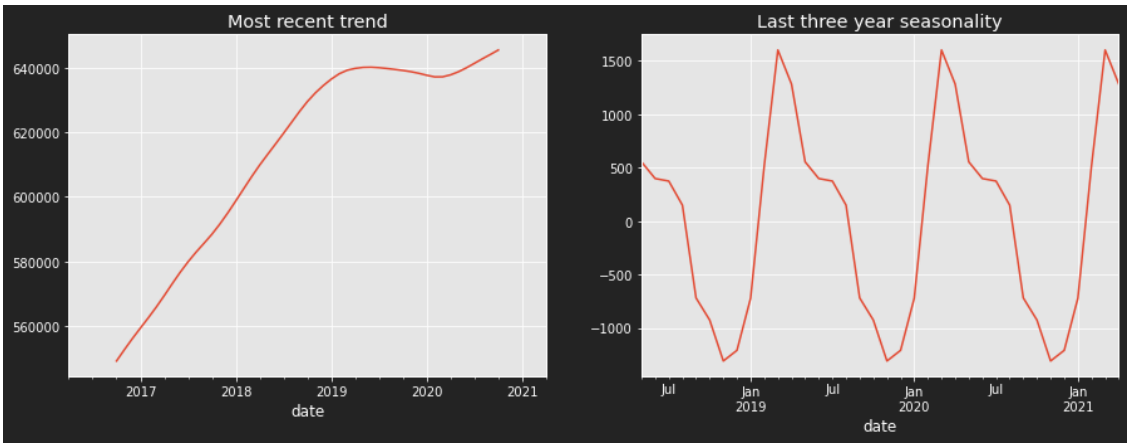
Prediction:

```
Root Mean Squared Error of test and prediction: 222405.8425298457
Mean Squared Error: 49464358791.41052
Mean Absolute Error: 192689.9140215953
```

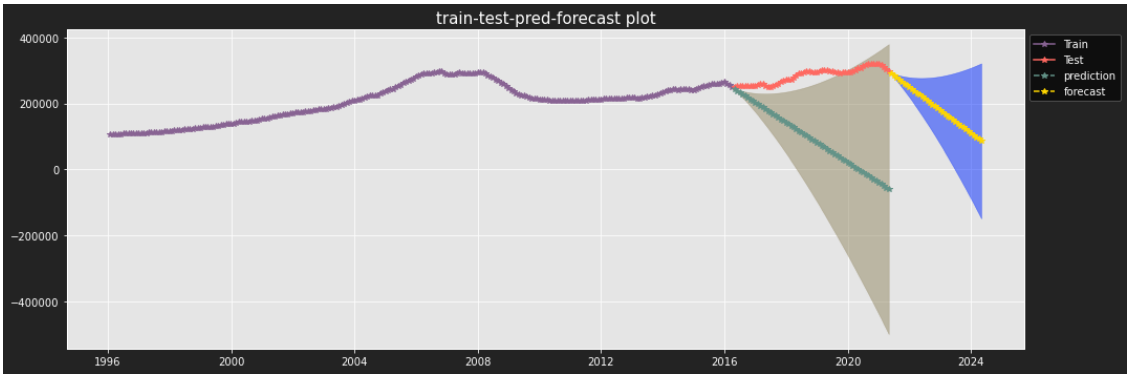




Insights:



Overall model performance and projected ROI:



```
  zipcode  mean_forecasted_roi  …  upper_forecasted_roi  std_forecasted_roi
0   11415               -71.04  …                  7.98               79.01

[1 rows x 5 columns]
```

[63]:
```
fn.output_df(und_per_model, results_)
```

[63]:
```
             aic  …  three_year_projected_upper_roi
ZipCode           …
11101    4482.02  …                            49.46
11354    3994.72  …                            22.69
11428    4232.73  …                            81.53
11423    4247.35  …                            87.65
11693    4509.50  …                            10.15
11375    4055.87  …                            35.14
11104    4041.80  …                            84.23
11415    4017.53  …                             7.98

[8 rows x 12 columns]
```

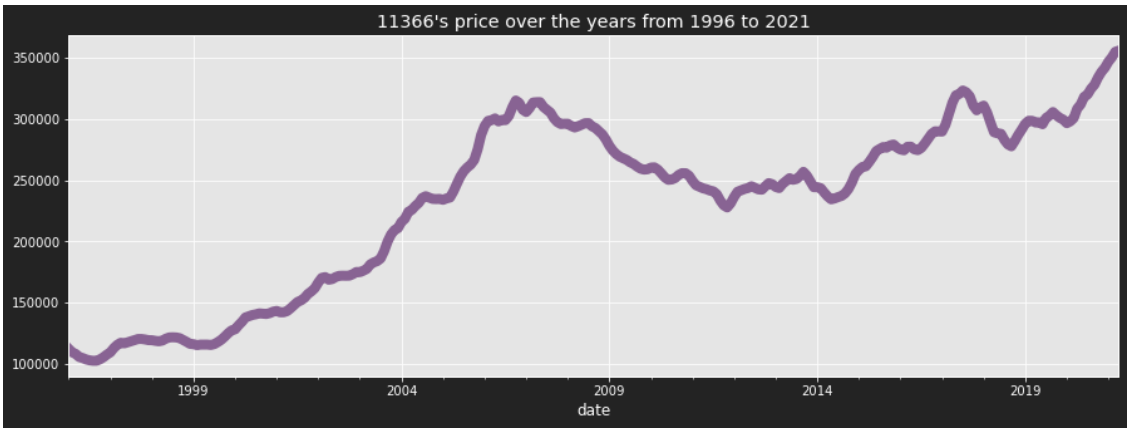
3.5.1 Actual under performing models

[64]:
```
# based on model fit and satistical information, diffrence between test roi and predicted ROI
actual_under_performing_models = ['11693', '11415']
```

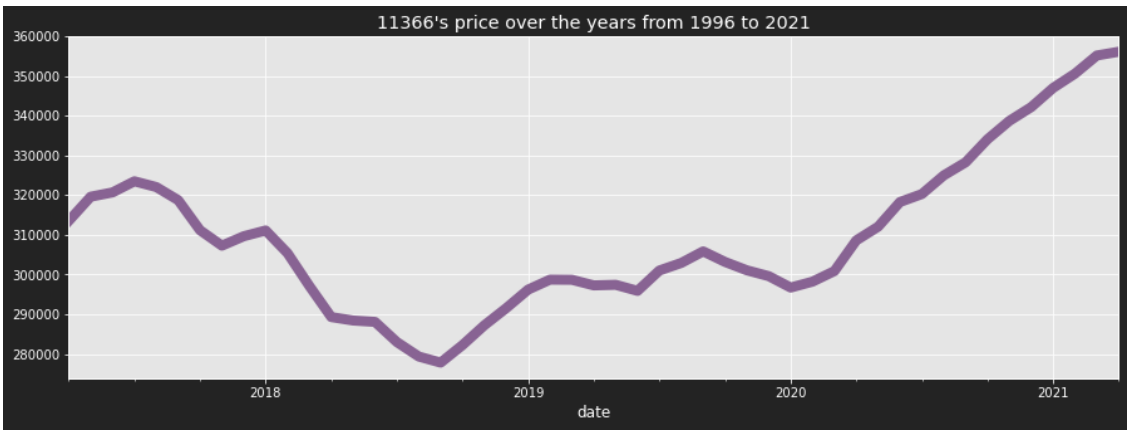
Zipcode 11693

[65]:
```
zip_11693 = ts_df['11693']
```

[66]:
```
zip_11693.plot(
    figsize=(15, 5),
    legend=0,
    color='#886393',
    lw=8,
    title=
    f"""{zipcode}'s price over the years from {
        str(zip_11693.index[0]).split(" ")[0].split("-")[0]} to {
        str(zip_11693.index[-1]).split(" ")[0].split("-")[0]}"""
);
```

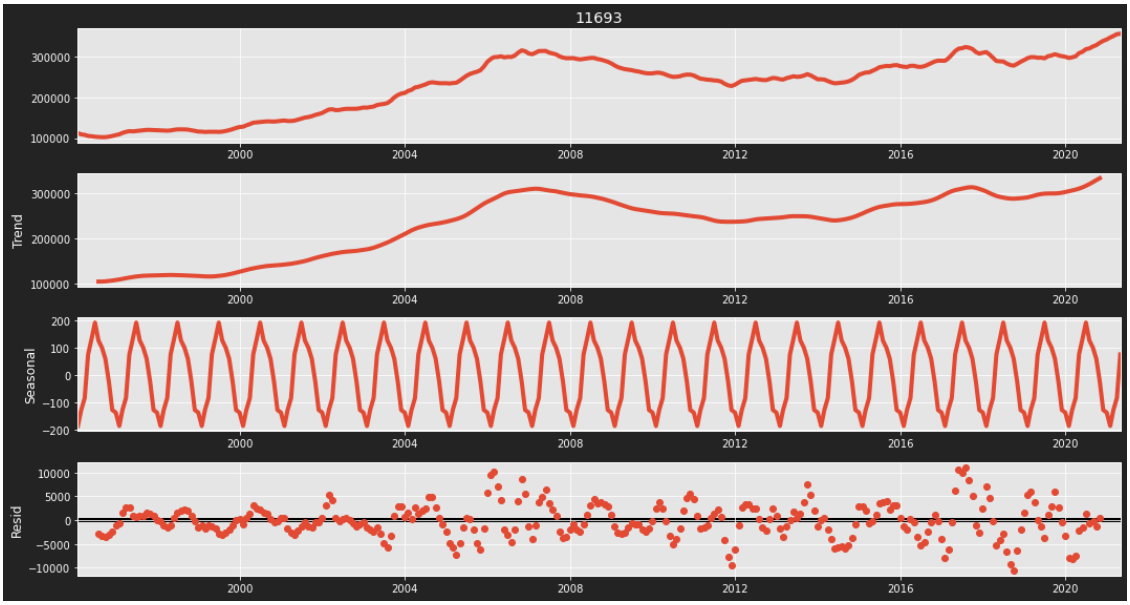


[67]:
```
zip_11693['2017-04-30':].plot(
    figsize=(15, 5),
    legend=0,
    color='#886393',
    lw=8,
    title=
    f"""{zipcode}'s price over the years from {
        str(zip_11693.index[0]).split(" ")[0].split("-")[0]} to {
        str(zip_11693.index[-1]).split(" ")[0].split("-")[0]}"""
);
```

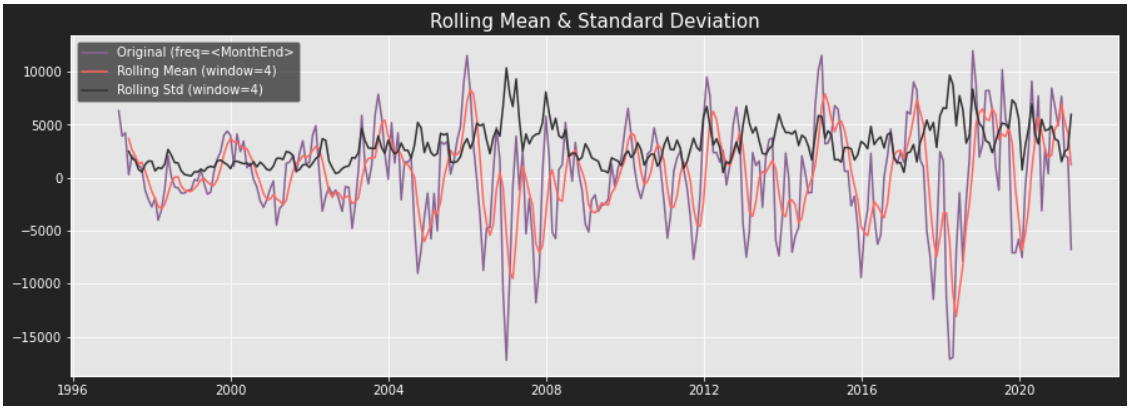


Stat test

[68]:
```
with mpl.rc_context():
    mpl.rc('figure', figsize=(15,8))
    mpl.rc('lines', linewidth = 4)
    tsa.seasonal_decompose(zip_11693.dropna()).plot();
```



[69]:
```
fn.stationarity_check((zip_11693.diff(1)).diff(12).dropna(), window=4)
```



[69]:

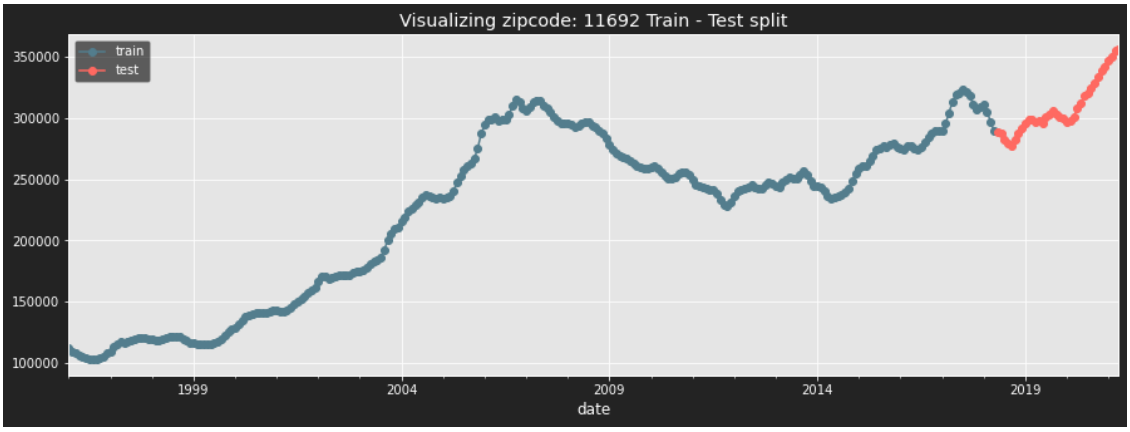
	Test Statistic	#Lags Used	…	p<.05	Stationary?
ADF Result	-4.893798	16	…	True	True

[1 rows x 6 columns]

Split A different split should be able to catch the trend. Setting the test split as same as the forecast period.

[66]:
```
## train test split
num_of_steps = 36
```

```
## Split
test_11693 = zip_11693.iloc[(len(zip_11693) - num_of_steps):]
train_11693 = zip_11693.iloc[:(len(zip_11693) - num_of_steps)]

## Visualize the train-test split split
fig, ax = plt.subplots(figsize=(15, 5))
kws = dict(ax=ax, marker='o')
train_11693.plot(**kws, label='train', color='#537d8d')
test_11693.plot(**kws, label='test', color='#ff6961')
ax.legend()
plt.title(f'Visualizing zipcode: {zipcode} Train - Test split')
plt.show()
```

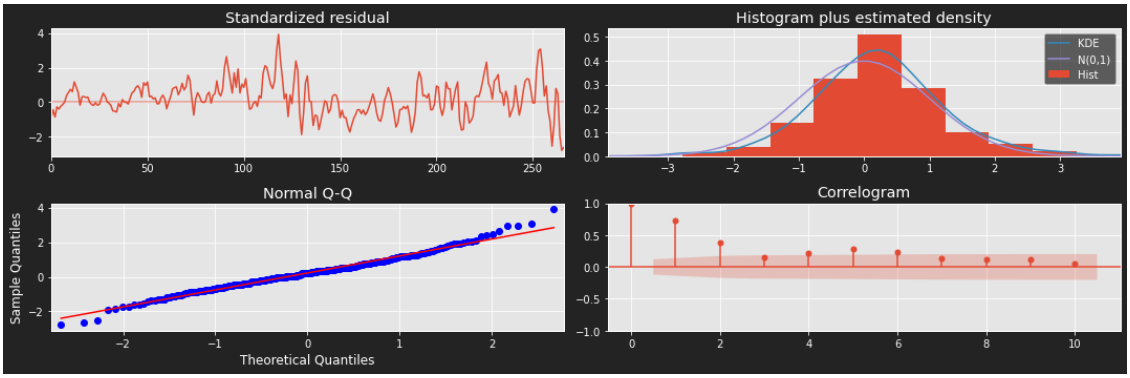


Model Grid search

[67]:
```
_, pred_11693, forecast_11693 = fn.grid_search(zip_11693, train_11693,␣
↪test_11693)
```

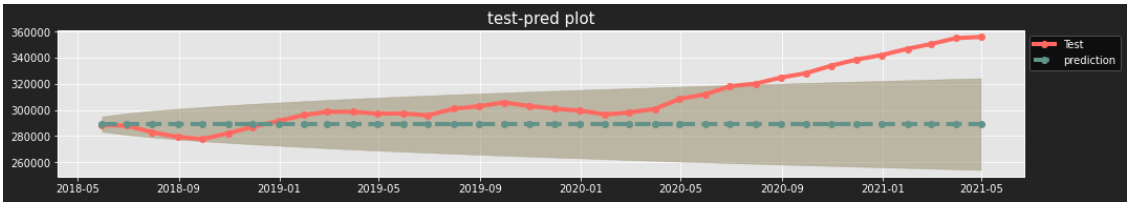
```
Performing stepwise search to minimize oob
 ARIMA(0,1,0)(0,0,0)[12] intercept   : OOB=212972519.409, Time=0.04 sec
 ARIMA(1,1,0)(1,0,0)[12] intercept   : OOB=167508922.491, Time=0.40 sec
 ARIMA(0,1,1)(0,0,1)[12] intercept   : OOB=195628397.442, Time=0.65 sec
 ARIMA(0,1,0)(0,0,0)[12]             : OOB=106277638.583, Time=0.02 sec
 ARIMA(0,1,0)(1,0,0)[12] intercept   : OOB=204759111.488, Time=0.14 sec
 ARIMA(0,1,0)(0,0,1)[12] intercept   : OOB=207361041.602, Time=0.14 sec
 ARIMA(0,1,0)(1,0,1)[12] intercept   : OOB=205287350.523, Time=0.28 sec
 ARIMA(1,1,0)(0,0,0)[12] intercept   : OOB=171788163.079, Time=0.14 sec
 ARIMA(0,1,1)(0,0,0)[12] intercept   : OOB=200441096.505, Time=0.19 sec
 ARIMA(1,1,1)(0,0,0)[12] intercept   : OOB=235101216.260, Time=0.24 sec

Best model:  ARIMA(0,1,0)(0,0,0)[12]
Total fit time: 2.255 seconds
 Model Diagonostics of 11693
```

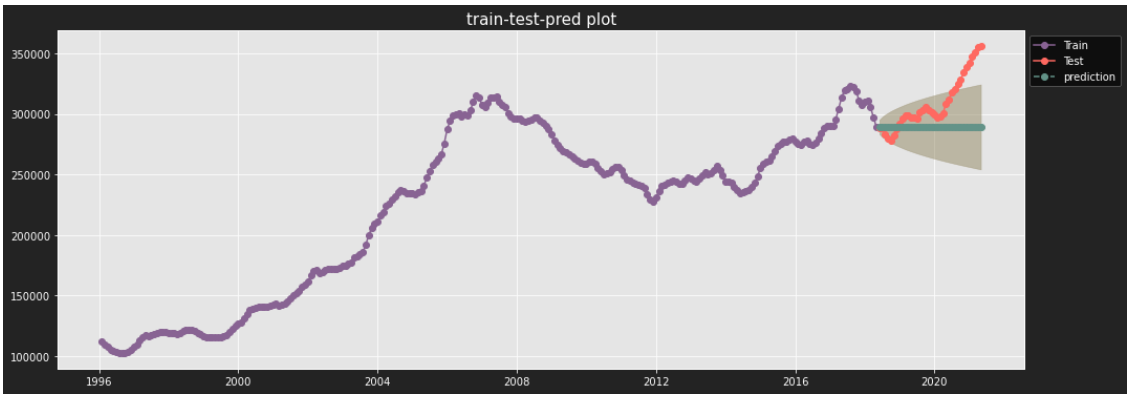
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                SARIMAX Results
==========================================================================================
Dep. Variable:                           y   No. Observations:                  268
Model:                 SARIMAX(0, 1, 0)   Log Likelihood               -2515.908
Date:                  Sat, 19 Jun 2021   AIC                           5033.816
Time:                          03:57:03   BIC                           5037.403
Sample:                               0   HQIC                          5035.257
                                  - 268
Covariance Type:                    opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sigma2      8.883e+06   6.06e+05     14.656      0.000     7.7e+06     1.01e+07
===================================================================================
Ljung-Box (L1) (Q):                 140.75   Jarque-Bera (JB):                 15.
 ↪46
Prob(Q):                              0.00   Prob(JB):                          0.
 ↪00
Heteroskedasticity (H):               4.13   Skew:                              0.
 ↪23
Prob(H) (two-sided):                  0.00   Kurtosis:                          4.
 ↪08
===================================================================================

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



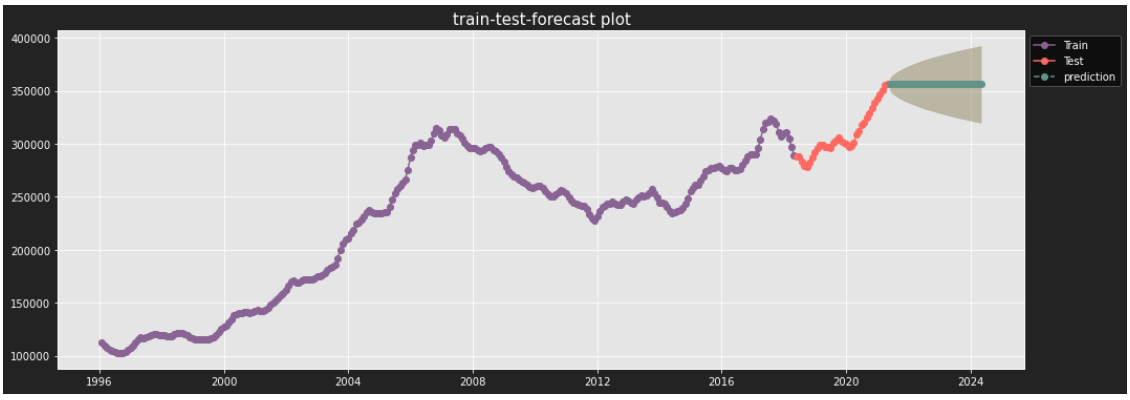

```
Performance on test data of 11693
Root Mean Squared Error of test and prediction: 29175.89351816553
Mean Squared Error: 851232762.5833334
```

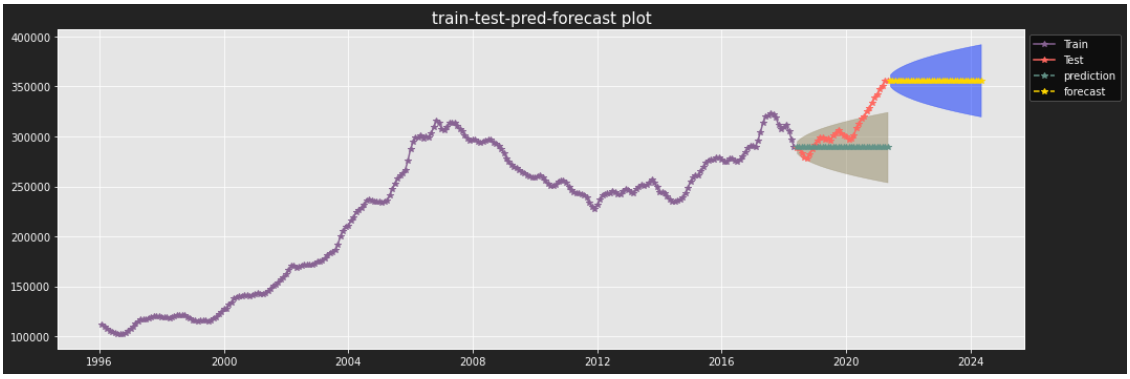
```
Mean Absolute Error: 21410.63888888889
```






```
Forecast of 11693
```



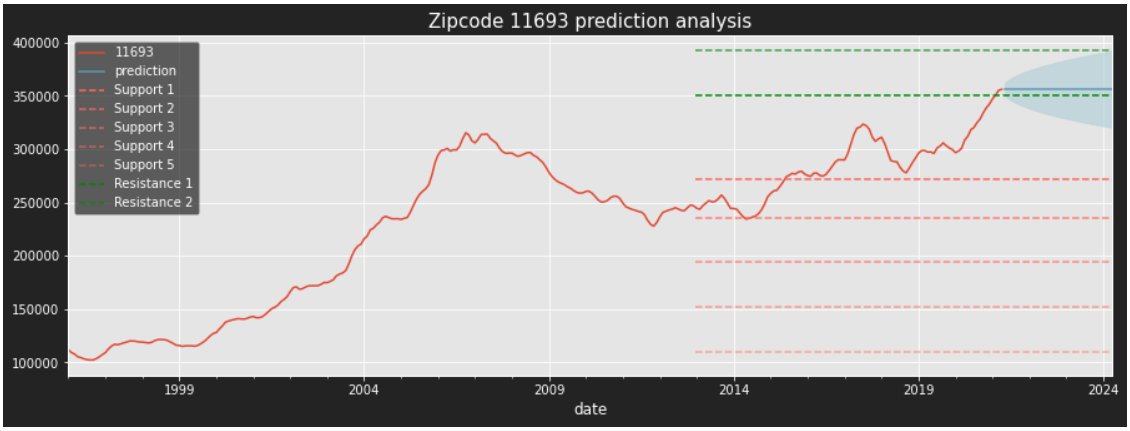


	zipcode	mean_forecasted_roi	…	upper_forecasted_roi	std_forecasted_roi
0	11693	0.0	…	10.15	10.15

[1 rows x 5 columns]

a different test split yielded better model fit.

[68]:
```
fn.prediction_analysis(zip_11693, test_11693, forecast_11693);
```



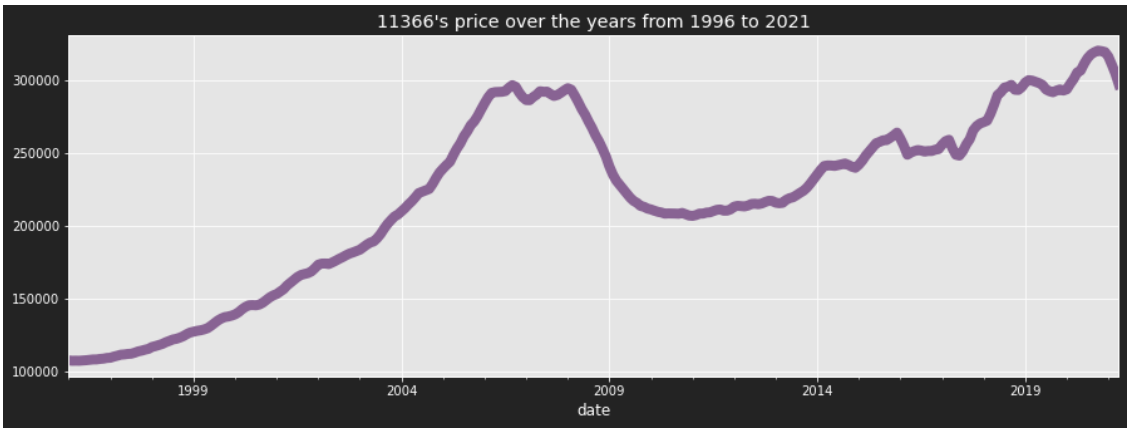
Inference

Even with modified train test split it does not change forecasted ROI by much. For that reason, not appending results to decision dataframe. Valid for short term. This zipcode is experiencing a recent uptrend as it recently crossed resistance level 1. Does not make sense to invest here as ROI can be lower as the investment required is high and to make a substantial profit gain market has to break the resistance once more. Dollar for dollar return is lower for high value property.

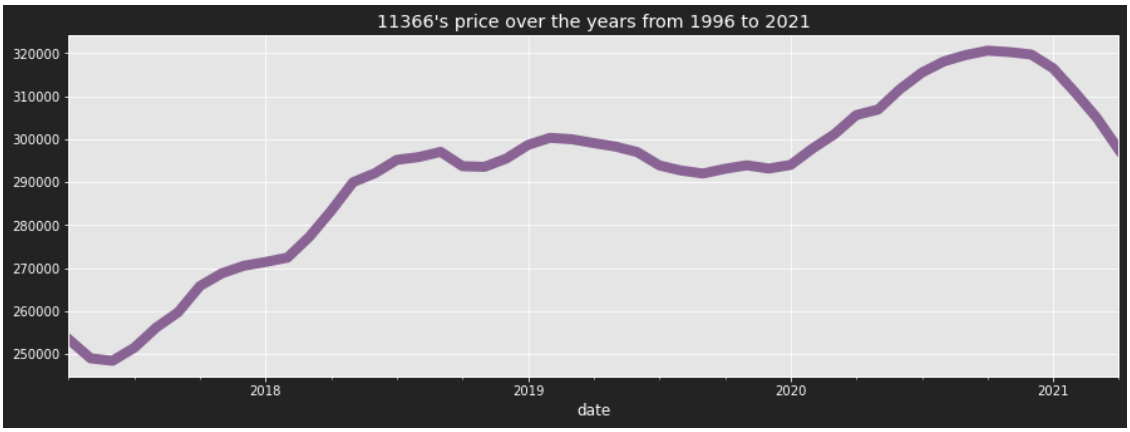
Zipcode 11415

[70]:
```
zip_11415 = ts_df['11415']
```

[71]:
```
zip_11415.plot(
    figsize=(15, 5),
    legend=0,
    color='#886393',
    lw=8,
    title=
    f"""{zipcode}'s price over the years from {
        str(zip_11415.index[0]).split(" ")[0].split("-")[0]} to {
        str(zip_11415.index[-1]).split(" ")[0].split("-")[0]}"""
);
```

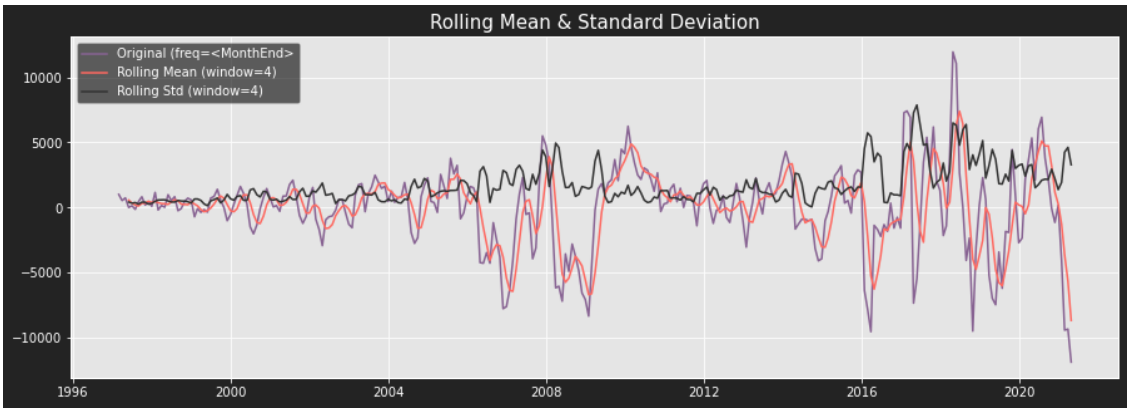


[72]:
```
zip_11415['2017-04-30':].plot(
    figsize=(15, 5),
    legend=0,
    color='#886393',
    lw=8,
    title=
    f"""{zipcode}'s price over the years from {
        str(zip_11415.index[0]).split(" ")[0].split("-")[0]} to {
        str(zip_11415.index[-1]).split(" ")[0].split("-")[0]}"""
);
```



Stat test

[73]:
```
fn.stationarity_check((zip_11415.diff(1)).diff(12).dropna(), window=4)
```



[73]:

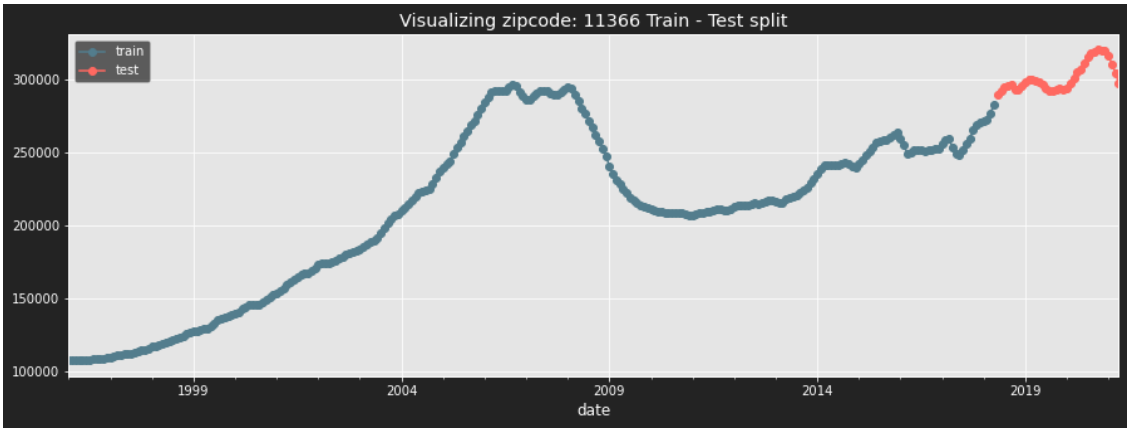
	Test Statistic	#Lags Used	…	p<.05	Stationary?
ADF Result	-3.48066	15	…	True	True

[1 rows x 6 columns]

Train test split

[77]:
```
## train test split
num_of_steps = 36

## Split
test_11415 = zip_11415.iloc[(len(zip_11415) - num_of_steps):]
train_11415 = zip_11415.iloc[:(len(zip_11415) - num_of_steps)]

## Visualize the train-test split split
```

```
fig, ax = plt.subplots(figsize=(15, 5))
kws = dict(ax=ax, marker='o')
train_11415.plot(**kws, label='train', color='#537d8d')
test_11415.plot(**kws, label='test', color='#ff6961')
ax.legend()
plt.title(f'Visualizing zipcode: {zipcode} Train - Test split')
plt.show()
```



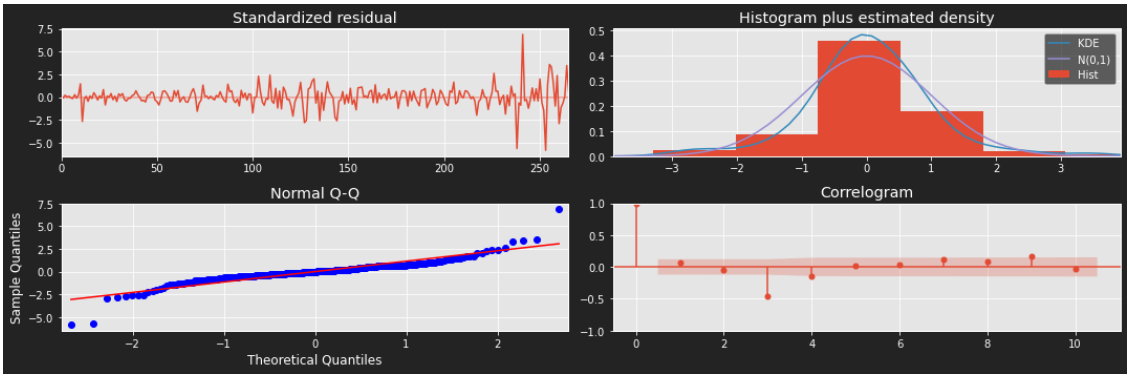
Grid search

[78]:
```
_, pred_11415, forecast_11415 = fn.grid_search(zip_11415, train_11415,
                                               test_11415)
```

```
Performing stepwise search to minimize oob
 ARIMA(0,2,0)(0,0,0)[12]             : OOB=3403438654.083, Time=0.03 sec
 ARIMA(1,2,0)(1,0,0)[12]             : OOB=3371344893.365, Time=0.12 sec
 ARIMA(0,2,1)(0,0,1)[12]             : OOB=3362765293.950, Time=0.25 sec
 ARIMA(0,2,1)(0,0,0)[12]             : OOB=3416692636.606, Time=0.08 sec
 ARIMA(0,2,1)(1,0,1)[12]             : OOB=3382774863.564, Time=0.34 sec
 ARIMA(0,2,1)(0,0,2)[12]             : OOB=3386204870.523, Time=0.31 sec
 ARIMA(0,2,1)(1,0,0)[12]             : OOB=3371235806.550, Time=0.12 sec
 ARIMA(0,2,1)(1,0,2)[12]             : OOB=3369716458.455, Time=1.07 sec
 ARIMA(0,2,0)(0,0,1)[12]             : OOB=3337498686.094, Time=0.11 sec
 ARIMA(0,2,0)(1,0,1)[12]             : OOB=3361153218.577, Time=0.38 sec
 ARIMA(0,2,0)(0,0,2)[12]             : OOB=3370846369.203, Time=0.24 sec
 ARIMA(0,2,0)(1,0,0)[12]             : OOB=3338240025.115, Time=0.10 sec
 ARIMA(0,2,0)(1,0,2)[12]             : OOB=3355782500.149, Time=1.55 sec
 ARIMA(1,2,0)(0,0,1)[12]             : OOB=3365197245.015, Time=0.19 sec
 ARIMA(1,2,1)(0,0,1)[12]             : OOB=3366870425.522, Time=0.22 sec
 ARIMA(0,2,0)(0,0,1)[12] intercept   : OOB=3455047832.075, Time=0.19 sec

Best model:  ARIMA(0,2,0)(0,0,1)[12]
Total fit time: 5.321 seconds
```

Model Diagonostics of 11415

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                   y   No. Observations:                 ␣
 ↪        268
Model:             SARIMAX(0, 2, 0)x(0, 0, [1], 12)   Log Likelihood                 ␣
 ↪  -2274.147
Date:                              Sun, 20 Jun 2021   AIC                              ␣
 ↪   4552.295
Time:                                     03:45:21    BIC                              ␣
 ↪   4559.462
Sample:                                          0    HQIC                             ␣
 ↪   4555.174
                                               - 268
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.S.L12      -0.0763      0.008     -9.279      0.000      -0.092      -0.060
sigma2      1.145e+06   3.36e+04     34.120      0.000    1.08e+06    1.21e+06
==============================================================================
Ljung-Box (L1) (Q):                   1.16   Jarque-Bera (JB):                938.
 ↪07
Prob(Q):                              0.28   Prob(JB):                          0.
 ↪00
Heteroskedasticity (H):               7.82   Skew:                             -0.
 ↪02
Prob(H) (two-sided):                  0.00   Kurtosis:                         12.
 ↪20
==============================================================================

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

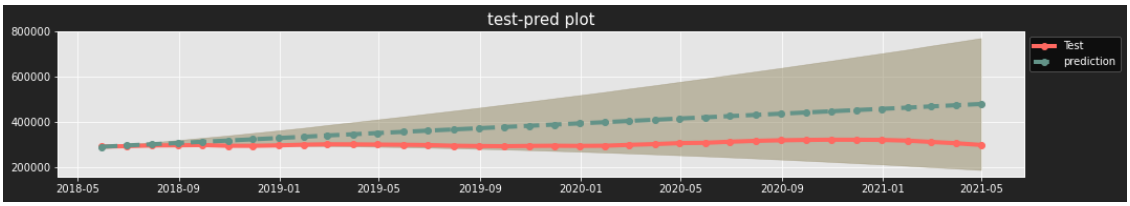


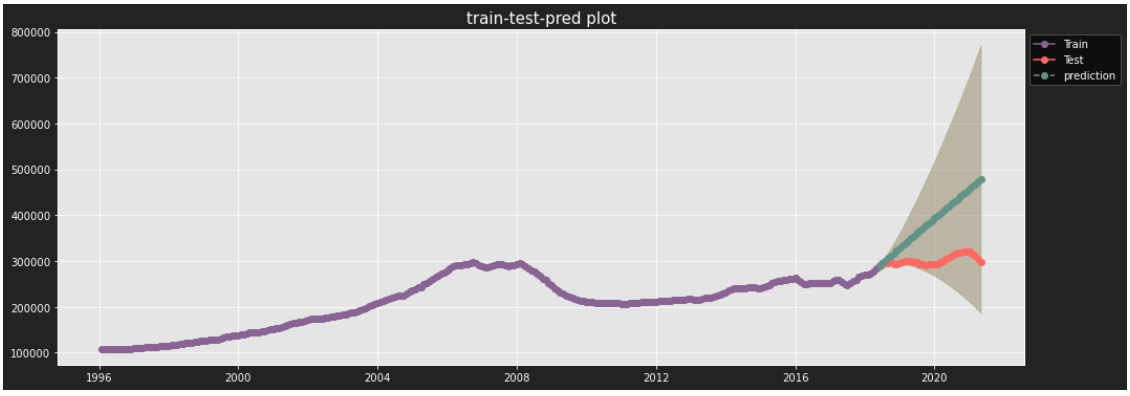
Performance on test data of 11415

Root Mean Squared Error of test and prediction: 96770.79761390286

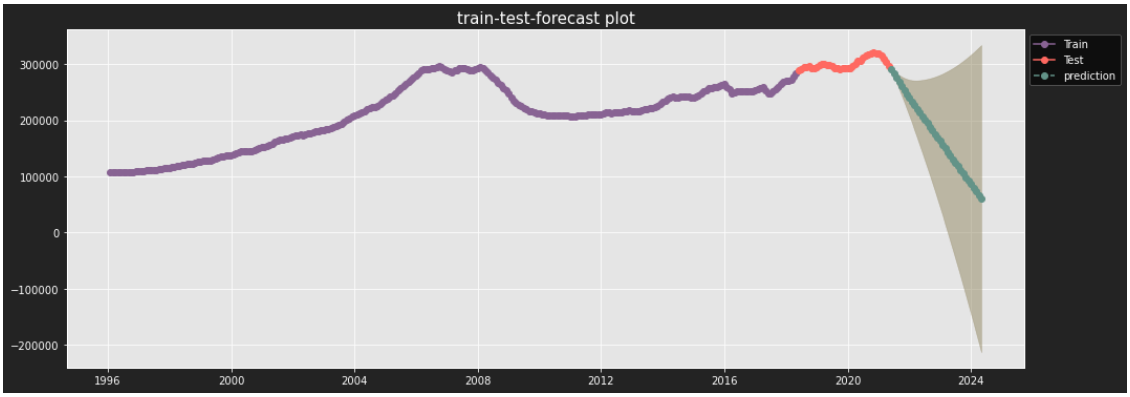
Mean Squared Error: 9364587270.830948

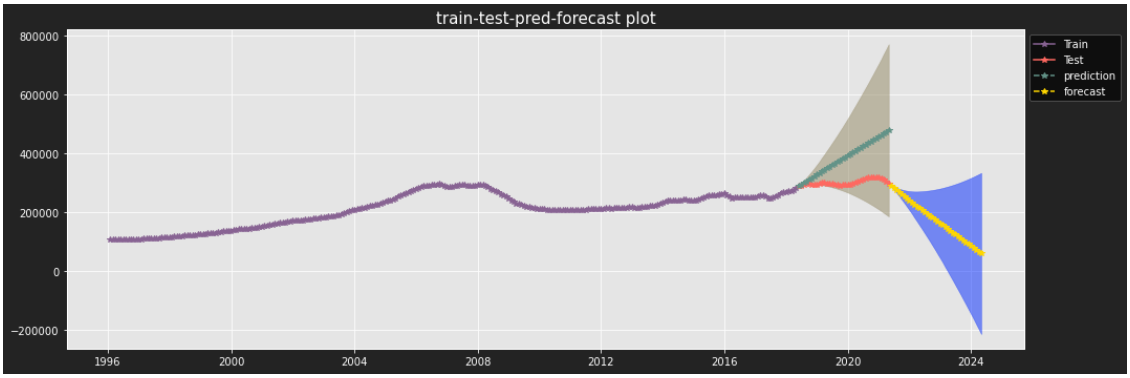
Mean Absolute Error: 83097.8291422973





Forecast of 11415

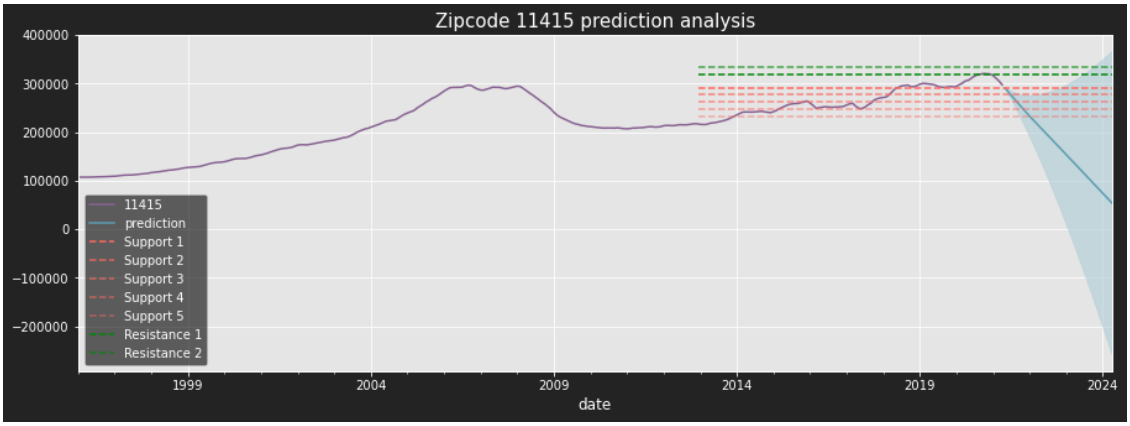




	zipcode	mean_forecasted_roi	…	upper_forecasted_roi	std_forecasted_roi
0	11415	-79.66	…	12.38	92.04

[1 rows x 5 columns]

[76]:
```
fn.prediction_analysis(zip_11415, test_11415, forecast_11415);
```



A better model fit but impact on model in not that good. Not expected to go below support level, but still not a good investment prospect.

[ ]:

4 INTERPRET

[80]:
```
# final
fn.output_df(best_investments, results_)
```

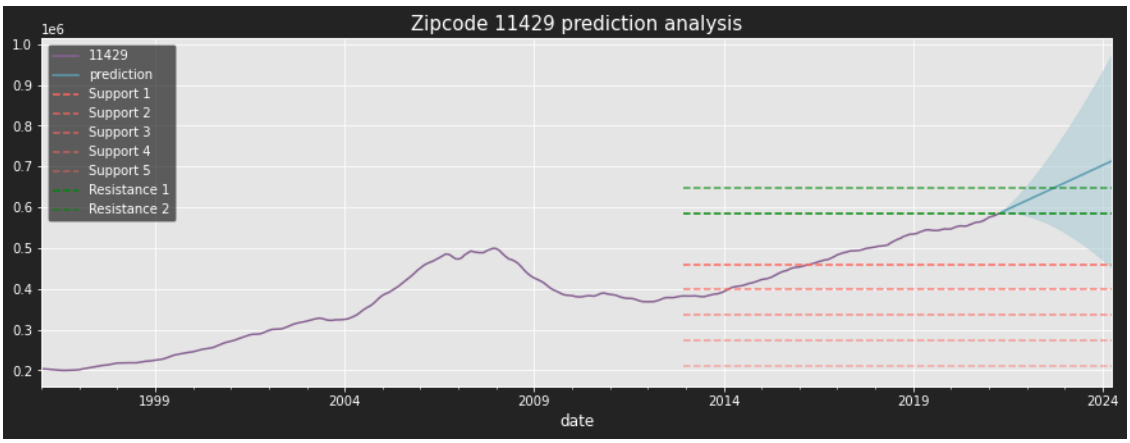
[80]:
```
             aic  …  three_year_projected_upper_roi
ZipCode           …
11429    4188.75  …                            67.23
11428    4232.73  …                            81.53
11427    4193.11  …                            78.11
11423    4247.35  …                            87.65
11417    4231.28  …                            66.92

[5 rows x 12 columns]
```

[34]:
```
best_investments
```

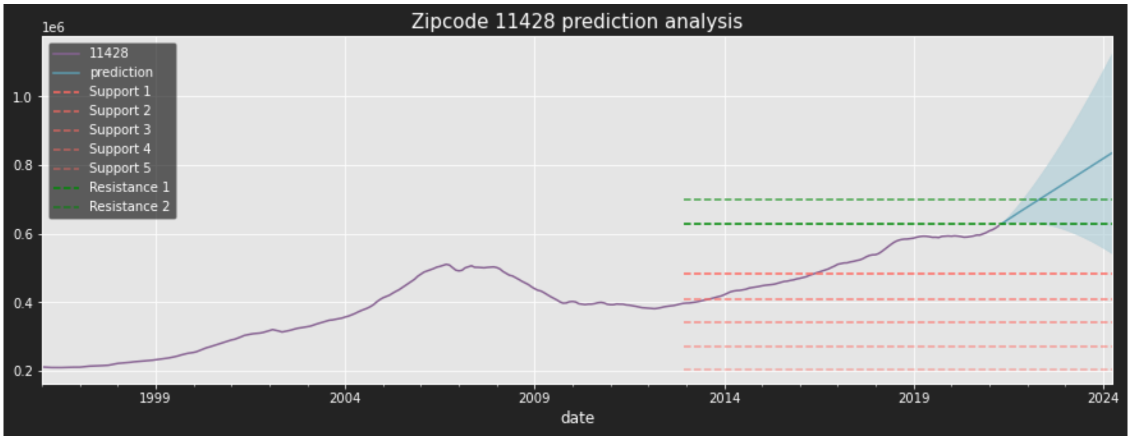
[34]:
```
Index(['11429', '11428', '11427', '11423', '11417'], dtype='object',
name='zipcode')
```

[381]:
```
print('Best investment opportunities')
for item in best_investments:
    print(f'{"+"*140}')
    fn.prediction_analysis(ts_df[item], results_[item]['test'],
                           results_[item]['pred_df'])
```

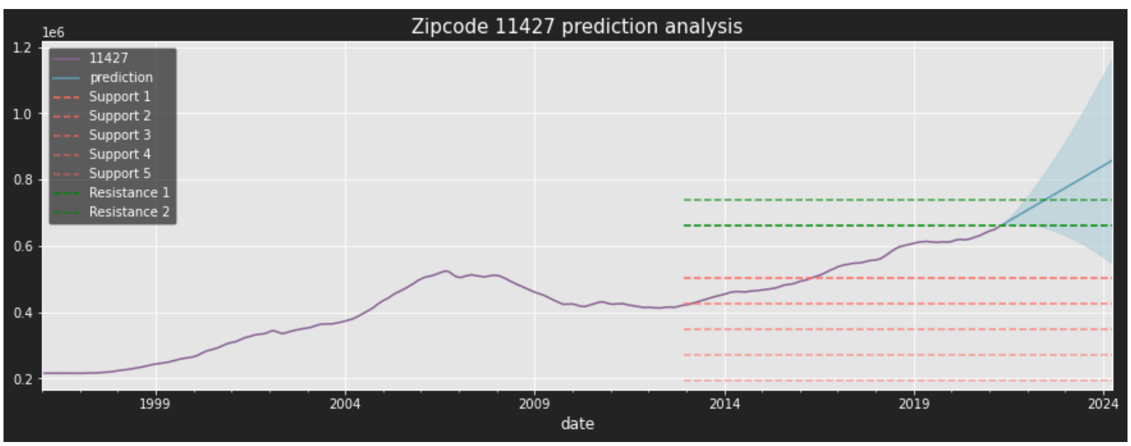
```
Best investment opportunities
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```



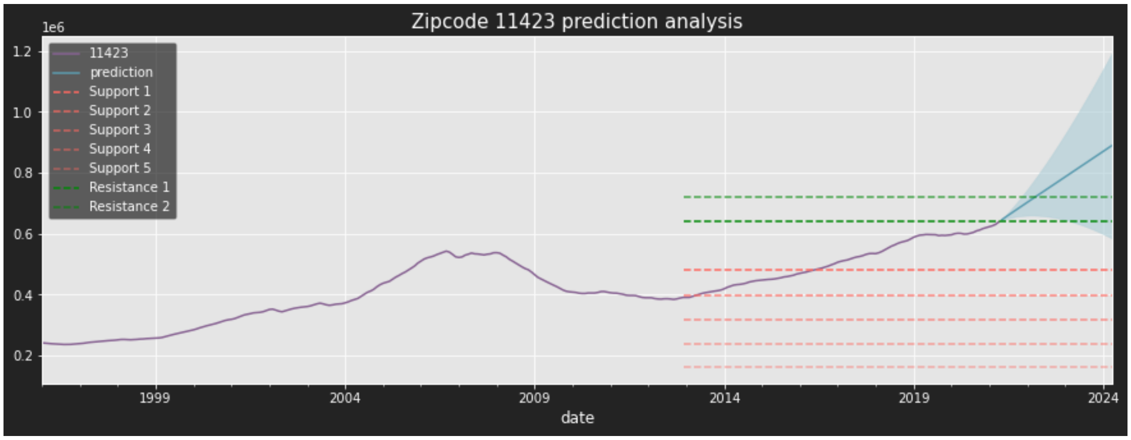
++


Zipcode 11428 prediction analysis
1e6
11428
prediction
Support 1
Support 2
Support 3
Support 4
Support 5
Resistance 1
Resistance 2
1.0
0.8
0.6
0.4
0.2
1999
2004
2009
2014
2019
2024
date

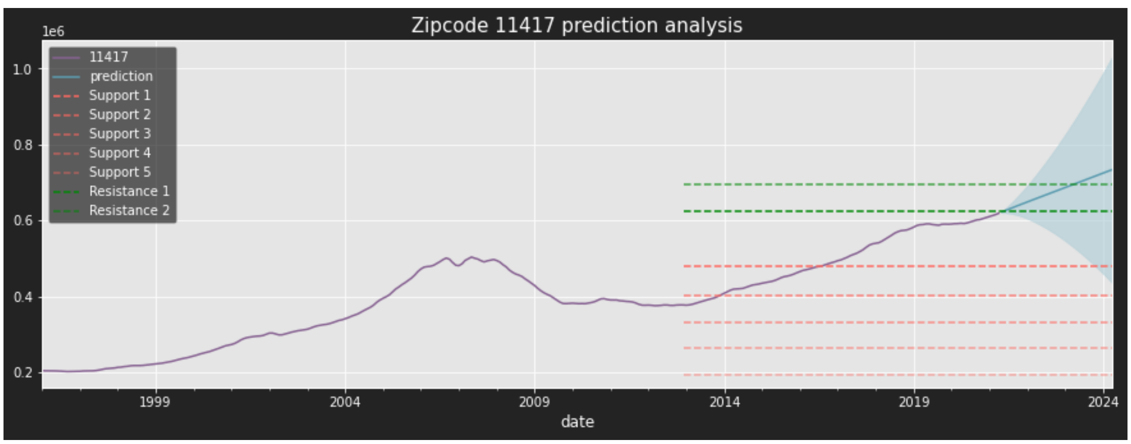
++


Zipcode 11427 prediction analysis
1e6
11427
prediction
Support 1
Support 2
Support 3
Support 4
Support 5
Resistance 1
Resistance 2
1.2
1.0
0.8
0.6
0.4
0.2
1999
2004
2009
2014
2019
2024
date

++



++
++



5 RECOMMENDATION

Invest in following zip codes: - 11429 - 11428 - 11427 - 11423 - 11417

Stay away from these, they are in a bubble : - 11693 - 11415

Rule of thumb - Go southeast part of Queens for good investment opportunity. - Some of the house are overvalued, and awaits correction, be careful of those houses.

6 CONCLUSION

Although modeling process is adequate, there are some caveats. - This analysis does not consider Time Value of Money, one of major driver for any financial decision making process. - Model generalization can be a issue. Analysis of individual models were not performed. All of the model

were run on on a loop and then searched for possible issue based of different metrics, e.g., RMSE, true versus prediction accuracy. - In general time series models are heavily contingent on model train test split, and recent trend. All of models were split on by 80-20 train-test ratio. There might be so issue of such generalization present in some of the model. Two of them were identified and dealt with, but without significant change in decision criteria. There might be some unidentified ones.

7 NEXT STEPS

- add variables to model, using a `SARIMAX` model
- interest rate
- other qualitative indicators
- try other models
  - RNN
  - Prophet
  - use transfer learning

8 APPENDIX

8.1 Show py file content

[35]:
```
fn.show_py_file_content(file='./imports_and_functions/packages.py')
```

```
import plotly.express as px
from datetime import datetime
import seaborn as sns
import joblib
from jupyterthemes import jtplot
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import random
import pmdarima as pm
import statsmodels.tsa.api as tsa
import pandas as pd
pd.set_option('display.max_columns', 0)

# OG plot
jtplot.reset()
# jtplot.style(theme='monokai', context='notebook', ticks='True', grid='False')
plt.style.use('ggplot')

# https://matplotlib.org/stable/tutorials/introductory/customizing.html
font = {'weight': 'normal', 'size': 8}
text = {'color': 'white'}
```

```
axes = {'labelcolor': 'white'}
xtick = {'color': 'white'}
ytick = {'color': 'white'}
legend = {'facecolor': 'black', 'framealpha': 0.6}

# mpl.rcParams['figure.facecolor'] = '#232323' # matplotlib over-writes this
mpl.rc('legend', **legend)
mpl.rc('text', **text)
mpl.rc('xtick', **xtick)
mpl.rc('ytick', **ytick)
mpl.rc('axes', **axes)
mpl.rc('font', **font)
```

[36]:
```
fn.show_py_file_content(file='./imports_and_functions/functions.py')
```

```
# imports
import plotly.express as px
from sklearn.metrics import r2_score
from pmdarima.arima.utils import ndiffs
from pmdarima.arima.utils import nsdiffs
import folium
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib as mpl
import statsmodels.tsa.api as tsa
from IPython.display import display, Markdown
import numpy as np
import pmdarima as pm
import warnings
import json
from statsmodels.tools.eval_measures import rmse, mse
from sklearn.metrics import mean_absolute_error as mae
from statsmodels.tools.sm_exceptions import ConvergenceWarning
warnings.simplefilter('ignore', UserWarning)
warnings.simplefilter('ignore', ConvergenceWarning)


# functions


def forecast_to_df(forecast, zipcode):
    """
    Creates dataframe from statsmodels.tsa model obeject forecast

    ### Helper function ###

    Parameters:
```

```
    ===========
    forecast = output from model forecast;
    zipcode = str; zipcode name,
    """
    test_pred = forecast.conf_int()
    test_pred[zipcode] = forecast.predicted_mean
    test_pred.columns = ['lower', 'upper', 'prediction']
    return test_pred


def model_error_report(test, pred_df, show_report=False):
    """
    Generates model reports; rmse, mse, mae

    ### Helper function ###

    Parameters:
    ===========
    test = array like; no default; test y,
    pred_df = predicted y with confidance interval.
    """
    rmse_ = rmse(pred_df['prediction'], test)
    mse_ = mse(pred_df['prediction'], test)
    mae_ = mae(test, pred_df['prediction'])
    if show_report:
        print(f'Root Mean Squared Error of test and prediction: {rmse_}')
        print(f'Mean Squared Error: {mse_}')
        print(f'Mean Absolute Error: {mae_}')
    return rmse_, mse_, mae_


def plot_test_pred(test, pred_df, figsize=(15, 5), conf_int=True):
    """
    plots test and prediction

    ### Helper function ###

    returns matplotlib fig and ax object.

    Parameters:
    ===========
    test = array like; no default; test y,
    pred_df = predicted y with confidance interval.
    figsize = tuple of int or float; deafult = (15, 5), figure size control,
    conf_int = bool; default = True, plots confidance interval
    """
    fig, ax = plt.subplots(figsize=figsize)
    ax.plot(test, label='Test', marker='o', color='#ff6961', lw=4)
```

```
    ax.plot(pred_df['prediction'], label='prediction',
            ls='--', marker='o', color='#639388', lw=4)
    if conf_int:
        ax.fill_between(
            x=pred_df.index, y1=pred_df['lower'], y2=pred_df['upper'], color='#938863', alpha=
    ax.legend()
    ax.legend(bbox_to_anchor=(1, 1), loc="upper left")
    fig.tight_layout()
    rmse_, mse_, mae_ = model_error_report(test, pred_df, show_report=True)

    return fig, ax


def plot_train_test_pred(train, test, pred_df, figsize=(15, 5), color_by_train_test=True):
    """
    plots test and prediction

    ### Helper function ###

    Parameters:
    ===========
    train = array like; no default; train y,
    test = array like; no default; test y,
    pred_df = pandas.DataFrame; no default; predicted y with confidance interval,
    figsize = tuple of int or float; deafult = (15, 5), figure size control,
    color_by_train_test = bool; default = True, seperates train and test data by color.

    returns matplotlib fig and ax object.
    """
    fig, ax = plt.subplots(figsize=figsize)
    # diffrentiate train test split by color
    if color_by_train_test:
        color_list = ['#886393', '#ff6961']
    else:
        color_list = ['#886393', '#886393']
    # train
    ax.plot(train, label='Train', marker='o', color=color_list[0])
    # test
    ax.plot(test, label='Test', marker='o', color=color_list[1])
    # prediction
    ax.plot(pred_df['prediction'], label='prediction',
            ls='--', marker='o', color='#639388')
    ax.fill_between(
        x=pred_df.index, y1=pred_df['lower'], y2=pred_df['upper'], color='#938863', alpha=.5)
    ax.legend()
    ax.legend(bbox_to_anchor=(1, 1), loc="upper left")
    fig.tight_layout()
    return fig, ax
```

```
def plot_train_test_pred_forecast(train, test, pred_df_test, pred_df, zipcode, figsize=(15, 5)
    """
    plots test and prediction

    ### Helper function ###

    Parameters:
    ===========
    train = array like; no default; train y,
    test = array like; no default; test y,
    pred_df_test = array like; no default; predicted y on train y with confidance interval,
    pred_df = array like; no default; predicted y on all data with confidance interval,
    figsize = tuple of int or float; deafult = (15, 5), figure size control,
    color_by_train_test = bool; default = True, seperates train and test data by color.

    returns matplotlib fig and ax object. and roi information as pandas.DataFrame object
    """
    fig, ax = plt.subplots(figsize=figsize)
    kws = dict(marker='*')
    if color_by_train_test:
        color_list = ['#886393', '#ff6961']
    else:
        color_list = ['#886393', '#886393']
    # train
    ax.plot(train, label='Train', **kws, color=color_list[0])
    # test
    ax.plot(test, label='Test', **kws, color=color_list[1])
    # prediction
    ax.plot(pred_df_test['prediction'],
            label='prediction',
            ls='--',
            **kws,
            color='#639388')
    ax.fill_between(x=pred_df_test.index,
                    y1=pred_df_test['lower'],
                    y2=pred_df_test['upper'],
                    color='#938863', alpha=.5)
    # forecast
    ax.plot(pred_df['prediction'],
            label='forecast',
            ls='--',
            **kws,
            color='#ffd700')
    ax.fill_between(x=pred_df.index,
                    y1=pred_df['lower'],
                    y2=pred_df['upper'],
```

```
                    color='#0028ff', alpha=.5)
    ax.legend(bbox_to_anchor=(1, 1), loc="upper left")
    ax.set_title('train-test-pred-forecast plot', size=15)
    fig.tight_layout()
    plt.show()
    # ROI
    mean_roi = (pred_df[-1:]['prediction'][0] - test[-1:][0])/test[-1:][0]
    lower_roi = (pred_df[-1:]['lower'][0] - test[-1:][0])/test[-1:][0]
    upper_roi = (pred_df[-1:]['upper'][0] - test[-1:][0])/test[-1:][0]
    std_roi = np.std([lower_roi, upper_roi])
    roi_df = pd.DataFrame([{
        'zipcode': zipcode,
        'mean_forecasted_roi': round(mean_roi*100, 2),
        'lower_forecasted_roi': round(lower_roi*100, 2),
        'upper_forecasted_roi': round(upper_roi*100, 2),
        'std_forecasted_roi': round(std_roi*100, 2)
    }])
    display(roi_df)
    return fig, ax, roi_df


def plot_acf_pacf(ts, figsize=(15, 8), lags=24):
    """
    Plots acf and pacf of a time series

    Parameters:
    ===========
    ts = array like; no default; time series data,
    figsize = tuple of int or float; deafult = (15, 8), figure size control,
    lags = int; default = 24, lag input of plot_acf() of statsmodels.tsa

    returns matplotlib fig and ax object.
    """
    fig, ax = plt.subplots(nrows=3, figsize=figsize)
    # Plot ts
    ts.plot(ax=ax[0], color='#886393', lw=5)
    # Plot acf, pacf
    plot_acf(ts, ax=ax[1], lags=lags, color='#886393', lw=5)
    plot_pacf(ts, ax=ax[2], lags=lags, method='ld', color='#886393', lw=5)
    fig.tight_layout()
    fig.suptitle(f"Zipcode: {ts.name}", y=1.02, fontsize=15)
    for a in ax[1:]:
        a.xaxis.set_major_locator(
            mpl.ticker.MaxNLocator(min_n_ticks=lags, integer=True))
        a.xaxis.grid()
    plt.show()
    return fig, ax
```

```
def adfuller_test_df(ts, index=['AD Fuller Results']):
    """
    Adapted from https://github.com/learn-co-curriculum/dsc-removing-trends-lab/tree/solution
    Returns the AD Fuller Test Results and p-values for the null hypothesis that there the
    data is non-stationary (that there is a unit root in the data).

    ### helper function ###

    Parameters:
    ===========
    ts = array like; no default; time series data,

    Returns:
    ========
    report of the test as pandas.DataFrame object.
    """

    df_res = tsa.stattools.adfuller(ts)

    names = ['Test Statistic', 'p-value',
             '#Lags Used', '# of Observations Used']
    res = dict(zip(names, df_res[:4]))

    res['p<.05'] = res['p-value'] < .05
    res['Stationary?'] = res['p<.05']

    if isinstance(index, str):
        index = [index]
    res_df = pd.DataFrame(res, index=index)
    res_df = res_df[['Test Statistic', '#Lags Used',
                     '# of Observations Used', 'p-value', 'p<.05',
                     'Stationary?']]
    return res_df


def stationarity_check(TS, window=8, plot=True, figsize=(15, 5), index=['ADF Result']):
    """
    Adapted from https://github.com/learn-co-curriculum/dsc-removing-trends-lab/tree/solution
    Checks stationarity of the time series.

    Parameters:
    ===========
    TS = array like; no default; time series data, ,
    window = int; default = 8, rolling window input.
    plot = bool; default = True, displays plot
    figsize = tuple of int or float; deafult = (15, 5), figure size control,
    index= list containing str; default = ['ADF Result']. dataframe index input.
```

```
    Returns:
    ========
    report of the test as pandas.DataFrame object.
    """
    # Calculate rolling statistics
    roll_mean = TS.rolling(window=window, center=False).mean()
    roll_std = TS.rolling(window=window, center=False).std()

    # Perform the Dickey Fuller Test
    dftest = adfuller_test_df(TS, index=index)

    if plot:

        # Building in contingency if not a series with a freq
        try:
            freq = TS.index.freq
        except:
            freq = 'N/A'

        # Plot rolling statistics:
        fig = plt.figure(figsize=figsize)
        plt.plot(TS, color='#886393', label=f'Original (freq={freq}')
        plt.plot(roll_mean,
                 color='#ff6961',
                 label=f'Rolling Mean (window={window})')
        plt.plot(roll_std,
                 color='#333333',
                 label=f'Rolling Std (window={window})')
        plt.legend(loc='best')
        plt.title('Rolling Mean & Standard Deviation', size=15)
        # display(dftest)
        plt.show(block=False)

    return dftest


def melt_data(df):
    """
    Converts dataframe from wide form to long form.
    ### pre-defined function ###
    Returns:
    ========
    panda.DataFrame
    """
    melted = pd.melt(df,
                     id_vars=['RegionName', 'State',
                              'City', 'Metro', 'CountyName'],
```

```
                    var_name='date')
    melted['date'] = pd.to_datetime(melted['date'], infer_datetime_format=True)
    melted = melted.dropna(subset=['value'])
    return melted


def model_builder(train, test, order, seasonal_order, zipcode, figsize=(15, 8), show_summary=Tr
    """
    builds a model with statsmodels.tsa.SARIMAX and prints information.

    Parameters:
    ===========
    train = array like; no default; train y,
    test = array like; no default; test y,
    order = tuple of three int; no default, p, d, q of SARIMAX model,
    seasonal_order = tuple of four int; no default, P, D, Q, m of SARIMAX model,
    zipcode = str; zipcode name,
    figsize = tuple of int or float; deafult = (15, 8), figure size control,
    show_summary = bool; default = True, show SARIMAX summary,
    show_diagnostics = bool; default = True, show model diagonistics reports,
    show_prediction = bool; default = True, show prediction,

    Returns:
    ========
    SARIMAX model object,
    prediction of the model for steps of length of test.
    """
    # model
    model = tsa.SARIMAX(train, order=order,
                        seasonal_order=seasonal_order).fit()
    if show_summary:
        display(model.summary())
    print('\033[1m \033[5;30;47m' +
          f'{" "*70}Model Diagonostics of {zipcode}{" "*70}'+'\033[0m')
    if show_diagnostics:
        model.plot_diagnostics(figsize=figsize)
        plt.tight_layout()
        plt.show()
    # forecast
    forecast = model.get_forecast(steps=len(test))
    pred_df = forecast_to_df(forecast, zipcode)
    if show_prediction:
        print('\033[1m \033[5;30;47m' +
              f'{" "*70}Performance on test data of {zipcode}{" "*70}'+'\033[0m')
        _, ax = plot_test_pred(
            test, pred_df, figsize=(figsize[0], figsize[1]/2))
        ax.set_title('test-pred plot', size=15)
        _, ax = plot_train_test_pred(
```

```
            train, test, pred_df, figsize=(figsize[0], figsize[1]-2))
        ax.set_title('train-test-pred plot', size=15)
        plt.show()

    return model, pred_df


def grid_search(ts, train, test, forecast_steps=36, figsize=(15, 5), trace=True, display_result
    """
    grid searching using pyramid arima for best p, d, q, P, D, Q, m for
    a SARIMA model using predefined conditions and shows model performance
     for predicting in the future.

    ### predefined options ###
    # d and D is calculated using ndiffs using 'adf'(Augmented Dickey-Fuller test for Unit Roo
     for d and 'ocsb' (Osborn, Chui, Smith, and Birchenhall Test for Seasonal Unit Roots) for
    # parameters for auto_arima model:
    start_p=0; The starting value of p, the order (or number of time lags) of the auto-regress
    d=d; The order of first-differencing,
    start_q=0; order of the moving-average ("MA") model,
    max_p=3, max value for p
    max_q=3, max value for q
    start_P=0; the order of the auto-regressive portion of the seasonal model,
    D=D; The order of the seasonal differencing,
    start_Q=0; the order of the moving-average portion of the seasonal model,
    max_P=3, max value of P
    max_Q=3, max value for Q
    m=12; The period for seasonal differencing,
                refers to the number of periods in each season.,
    seasonal=True; this data is seasonal,
    stationary=False; data is not stationary,
    information_criterion='oob', optimizing on `out-of-bag` sample validation on a scoring met
                other information criterias did not perform well
    out_of_sample_size=12, step hold out for validation,
    scoring='mse', validation metric,
    method='lbfgs'; limited-memory Broyden-Fletcher-Goldfarb-Shanno with optional box constrai
                BFGS is  in the family of quasi-Newton-Raphson methods that
                approximates the `bfgs` using a limited amount of computer memory.

    Parameters:
    ===========
    ts = array like; no default; time series y,
    train = array like; no default; train y,
    test = array like; no default; test y,
    forecast_steps = int; default = 36, steps to forecast into future,
    figsize = tuple of int or float; deafult = (15, 8), figure size control,
    trace = bool; default = True,
    display_results = bool; default = True,
```

```
    display_roi_results = bool; default = True,

    Returns:
    ========
    auto_model model object created by pmdautoarima,
    predictions of test,
    forecast of ts for selected steps.
    """

    d = ndiffs(train, alpha=0.05, test='adf', max_d=4)
    D = nsdiffs(train, m=12, test='ocsb', max_D=4)

    auto_model = pm.auto_arima(y=train,
                               X=None,
                               start_p=0,
                               d=d,
                               start_q=0,
                               max_p=3,
                               max_q=3,
                               start_P=0,
                               D=D,
                               start_Q=0,
                               max_P=3,
                               max_Q=3,
                               m=12,
                               seasonal=True,
                               stationary=False,
                               information_criterion='oob',
                               stepwise=True,
                               suppress_warnings=True,
                               error_action='warn',
                               trace=trace,
                               out_of_sample_size=12,
                               scoring='mse',
                               method='lbfgs',
                               )
    # display results of grid search
    zipcode = ts.name
    if display_results:
        print('\033[1m \033[5;30;47m' +
              f'{" "*70}Model Diagonostics of {zipcode}{" "*70}'+'\033[0m')
        display(auto_model.summary())
        auto_model.plot_diagnostics(figsize=figsize)
        plt.tight_layout()
        plt.show()
    # fitting model on train data with the best params found by grid search
    best_model = tsa.SARIMAX(train,
                             order=auto_model.order,
```

```
                                seasonal_order=auto_model.seasonal_order, maxiter=500,
                                enforce_invertibility=False).fit()
        forecast = best_model.get_forecast(steps=len(test))
        pred_df_test = pd.DataFrame([forecast.conf_int(
        ).iloc[:, 0], forecast.conf_int().iloc[:, 1], forecast.predicted_mean]).T
        pred_df_test.columns = ["lower", 'upper', 'prediction']
        if display_results:
            print('\033[1m \033[5;30;47m' +
                  f'{" "*70}Performance on test data of {zipcode}{" "*70}'+'\033[0m')
            _, ax = plot_test_pred(
                test, pred_df_test, figsize=(figsize[0], figsize[1]/2))
            ax.set_title('test-pred plot', size=15)

            _, ax = plot_train_test_pred(
                train, test, pred_df_test, figsize=figsize)
            ax.set_title('train-test-pred plot', size=15)
            plt.show()

        # fitting on entire data
        best_model_all = tsa.SARIMAX(ts,
                                     order=auto_model.order,
                                     seasonal_order=auto_model.seasonal_order, maxiter=500,
                                     enforce_invertibility=False).fit()
        forecast = best_model_all.get_forecast(steps=forecast_steps)
        pred_df = pd.DataFrame([forecast.conf_int(
        ).iloc[:, 0], forecast.conf_int().iloc[:, 1], forecast.predicted_mean]).T
        pred_df.columns = ["lower", 'upper', 'prediction']
        if display_results:
            print('\033[1m \033[5;30;47m' +
                  f'{" "*70}Forecast of {zipcode}{" "*70}'+'\033[0m')
            _, ax = plot_train_test_pred(train, test, pred_df, figsize=figsize)
            ax.set_title('train-test-forecast plot', size=15)
            plt.show()
        if display_roi_results:
            plot_train_test_pred_forecast(
                train, test, pred_df_test, pred_df, zipcode, figsize=figsize)

        return auto_model, pred_df_test, pred_df


    def model_loop(ts_df,
                   zipcode_list,
                   train_size=.8, show_grid_search_steps=True,
                   forecast_steps=36, figsize=(15, 5),
                   display_details=False):
        """
        Loops through provided zipcodes as list with grid_search function and
        stores output using provided train test split.
```

```
    Parameters:
    ===========
    ts_df = pandas.DataFrame, all zipcode inforamation
    zipcode_list = list, zipcodes to loop
    train_size=.8,
    show_grid_search_steps = bool; default = True, display gid search steps,
    display_details = bool; default = False, show forecast breakdown.
    forecast_steps = int; default = 36, steps to forecast into future.
    figsize = tuple of int or float; deafult = (15, 8), figure size control,

    Returns:
    ========
    Result dict, and ROI dataframe
    """
    # store results
    RESULTS = {}
    # store ROI information
    ROI = pd.DataFrame(columns=[
        'zipcode', 'mean_forecasted_roi', 'lower_forecasted_roi',
        'upper_forecasted_roi', 'std_forecasted_roi'
    ])

    # loop counter
    n = 0
    for zipcode in zipcode_list:
        # loop counter
        n = n + 1
        len_ = len(zipcode_list)
        print(f"""Working on #{n} out of {len_} zipcodes.""")
        print('Working on:', zipcode)
        # make empty dicts for storing data
        temp_dict = {}
        # temp_dict_1 = {}

        # make a copy of time series data
        ts = ts_df[zipcode].dropna().copy()
        # train-test split
        train_size = train_size
        split_idx = round(len(ts) * train_size)
        # split
        train = ts.iloc[:split_idx]
        test = ts.iloc[split_idx:]

        # Get best params using auto_arima on train-test data
        if display_details:
            display_results_gs = True
            display_roi_results_gs = True
```

```
            else:
                display_results_gs = False
                display_roi_results_gs = False

            model, pred_df_test, pred_df = grid_search(ts,
                                                       train,
                                                       test,
                                                       forecast_steps=forecast_steps,
                                                       figsize=figsize,
                                                       trace=show_grid_search_steps,
                                                       display_results=display_results_gs,
                                                       display_roi_results=display_roi_results_gs)

            # storing data in RESULTS
            temp_dict['model'] = model
            temp_dict['train'] = train
            temp_dict['test'] = test
            temp_dict['pred_df_test'] = pred_df_test
            temp_dict['pred_df'] = pred_df

            RESULTS[zipcode] = temp_dict

            # storing data in ROI
            mean_roi = (pred_df[-1:]['prediction'][0] - test[-1:][0])/test[-1:][0]
            lower_roi = (pred_df[-1:]['lower'][0] - test[-1:][0])/test[-1:][0]
            upper_roi = (pred_df[-1:]['upper'][0] - test[-1:][0])/test[-1:][0]
            std_roi = np.std([lower_roi, upper_roi])

            roi_df = pd.DataFrame([{
                'zipcode': zipcode,
                'mean_forecasted_roi': mean_roi,
                'lower_forecasted_roi': lower_roi,
                'upper_forecasted_roi': upper_roi,
                'std_forecasted_roi': std_roi
            }])
            ROI = ROI.append(roi_df, ignore_index=True)
            print('-' * 90, end='\n')
        print('Looping completed.')
        return RESULTS, ROI


def zip_code_map(roi_df):
    """
    Returns an interactive map of zip codes colorized to reflect
    expected return on investment using folium.
    """
    geojason = json.load(
        open('./data/ny_new_york_zip_codes_geo.min.json', 'r'))
```

```
    zip_code_map = folium.Map(
        location=[40.7027, -73.7890], width=1330, height=820, zoom_start=11)
    folium.Choropleth(
        geo_data=geojason,
        name='choropleth',
        data=roi_df,
        columns=['zipcode', 'mean_forecasted_roi'],
        key_on='feature.properties.ZCTA5CE10',
        fill_color='BuGn',
        fill_opacity=0.7,
        nan_fill_opacity=0
    ).add_to(zip_code_map)
    return zip_code_map


def map_zipcodes_return(df, plot_style='interactive', geojson_file_path='./data/ny_new_york_zi
    """
    GeoJson sourced from:
    Returns an map of zip codes colorized to reflect
    expected return on investment using plotly express.
    ### pre-defined function ###

    Parameters:
    ===========
    df = pandas.DataFrame; no default, return on investment dataframe,
    plot_style = str; default = 'interactive',
                available options:
                -  'interactive'
                - 'static'
                - 'dash'
    geojson_file_path = geojson; default = './data/ny_new_york_zip_codes_geo.min.json',
                    file path of geojson file.
    """
    import plotly.express as px
    geojason = json.load(open(geojson_file_path, 'r'))
    for feature in geojason['features']:
        feature['id'] = feature['properties']['ZCTA5CE10']
    fig = px.choropleth_mapbox(data_frame=df,
                               geojson=geojason,
                               locations='zipcode',
                               color='mean_forecasted_roi',
                               mapbox_style="stamen-terrain",
                               # range_color=[-20,20],
                               zoom=10.5, color_continuous_scale=['#FF3D70', '#FFE3E8', '#039E
                               color_continuous_midpoint=0, hover_name='Neighborhood',
                               hover_data=[
                                   'mean_forecasted_roi', 'lower_forecasted_roi',
                                   'upper_forecasted_roi', 'std_forecasted_roi'
```

```
                                ],
                                title='Zipcode by Average Price',
                                opacity=.85,
                                height=800,
                                center={
                                    'lat': 40.7027,
                                    'lon': -73.7890
                                })
    fig.update_geos(fitbounds='locations', visible=True)
    fig.update_layout(margin={"r": 0, "l": 0, "b": 0})
    if plot_style == 'interactive':
        fig.show()
    if plot_style == 'static':
        # import plotly.io as plyIo
        img_bytes = fig.to_image(format="png", width=1400, height=800, scale=1)
        from IPython.display import Image
        display(Image(img_bytes))
    if plot_style == 'dash':
        import dash
        import dash_core_components as dcc
        import dash_html_components as html
        app = dash.Dash()
        app.layout = html.Div([dcc.Graph(figure=fig)])
        app.run_server(debug=True, use_reloader=False)


def model_report(zipcode_list, results_, show_model_performance=True, show_train_fit=True, sho
    """
    ### predefined funtion ###
    For visualizing reports by using results from model loop.


    ##############################  OUTPUT CONTROL  ##############################
    ###########################################################################
    show_model_performance = True  # Performance metrics & Diagonistics plots
    show_train_fit = True          # test and prediction fit
    show_prediction = True         # forecast in the future
    #-------------------------------------------------------------------------#
    show_detailed_prediction = True  # forecast in the future, whith prediction #
    #terminology: ###  prediction is to judge model performance   ###############
    ###################################   &   ###################################
    ################   forecast is prediction of unknown  ######################
    ###########################################################################
    """
    for best_zipcode in zipcode_list:
        # display models
        print(f'{"-"*157}')
        print('\033[1m \033[5;30;47m' +
```

```
            f'{" "*70}Report of {best_zipcode}{" "*70}' + '\033[0m')
        print(f'{"-"*157}')
        print('\033[1m \033[91m' + 'Model Used:')
        display(results_[best_zipcode]['model'])
        if show_model_performance:
            # model performance
            print(results_[best_zipcode]['model'].summary())
            results_[best_zipcode]['model'].plot_diagnostics(figsize=(10, 5))
            plt.tight_layout()
            plt.show()
        # extracting information from results dict
        train = results_[best_zipcode]['train']
        test = results_[best_zipcode]['test']
        pred_df_test = results_[best_zipcode]['pred_df_test']
        pred_df = results_[best_zipcode]['pred_df']
        if show_train_fit:
            print('\033[1m \033[1;33;40m' + 'Prediction:')
            print('\033[0m')
            # plot train fit
            _, ax = plot_test_pred(test, pred_df_test, conf_int=test_conf_int)
            ax.set_title(f'test-pred plot of {best_zipcode} [prediction reliability]',
                         size=15)
            plt.show()
        if show_prediction:
            # plot_train_test_pred
            _, ax = plot_train_test_pred(train, test, pred_df)
            ax.set_title(f'train-test-pred plot of {best_zipcode} [forecast]',
                         size=15)
            plt.show()
        if show_detailed_prediction:
            print('\033[1m \033[1;33;40m' + 'Insights:'+'\033[1m')
            # train_test_pred_forecast
            fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(15, 5))
            ax1 = tsa.seasonal_decompose(results_['11432']['test']).trend.plot(
                title='Most recent trend', ax=ax1)
            ax2 = tsa.seasonal_decompose(
                results_['11432']['test'][-36:]).seasonal.plot(
                    title='Last three year seasonality', ax=ax2)
            plt.show()
            print('\033[1m \033[1;33;40m' +
                  'Overall model performance and projected ROI:')
            _, ax, _ = plot_train_test_pred_forecast(train, test, pred_df_test,
                                                     pred_df, best_zipcode)
            ax.set_title(f'train-test-pred-forecast plot of {best_zipcode}',
                         size=15)
        plt.show()
    pass
```

```
def output_df(zipcode_list, results_):
    """
    # data processing function #
    creats a dataframe for decision making.
    """
    def roi(end, beg):
        """Return on investment calculation"""
        x = (end-beg)/beg
        x = (x*100).round(2)
        return x

    def relative_standard_deviation(lower, upper):
        """
        Standard deviation expressed in percent and is obtained by multiplying the standard
        deviation by 100 and dividing this product by the average.

        ----------------------------------------------------------------------------
        Reference: https://www.chem.tamu.edu/class/fyp/keeney/stddev.pdf
        """
        x = abs(upper-lower)
        y = (np.std(x)/np.mean(x))*100
        return y

    output = {}
    for item in zipcode_list:
        temp_dict = {}
        temp_dict['aic'] = results_[item]['model'].aic().round(2)
        temp_dict['bic'] = results_[item]['model'].bic().round(2)
        temp_dict['oob'] = results_[item]['model'].oob().round(2)
        rmse_o, mse_o, mae_o = model_error_report(results_[item]['test'], results_[
                                                    item]['pred_df_test'], show_report=False)
        temp_dict['rmse'] = rmse_o.round()
        temp_dict['mse'] = mse_o.round()
        temp_dict['mse'] = mae_o.round()
        temp_dict['r2'] = r2_score(results_[item]['test'], results_[
                                   item]['pred_df_test']['prediction']).round(3)
        temp_dict['test_roi'] = roi(results_[
                                    item]['test'][-1], results_[item]['test'][-(len(results_[it
        temp_dict['pred_roi'] = roi(results_[
                                    item]['pred_df']['prediction'][-1], results_[item]['pred_d
        temp_dict['three_year_projected_mean_roi'] = roi(
            results_[item]['pred_df'][-1:]['prediction'][0], results_[item]['test'][-1:][0])
        temp_dict['risk'] = round(relative_standard_deviation(
            results_[item]['pred_df']['lower'], results_[item]['pred_df']['upper']), 2)
        temp_dict['three_year_projected_lower_roi'] = roi(
            results_[item]['pred_df'][-1:]['lower'][0], results_[item]['test'][-1:][0])
        temp_dict['three_year_projected_upper_roi'] = roi(
            results_[item]['pred_df'][-1:]['upper'][0], results_[item]['test'][-1:][0])
```

```
        output[item] = temp_dict
    out = pd.DataFrame(output).T
    out.index.name = "ZipCode"
    return out


def prediction_analysis(ts, test, forecast):
    """
    Creats forecast and time series data with 2 resistance and 5 support level.

    Parameters:
    ===========
    ts = array like; no default; time series y,
    test = array like; no default; test y,
    forecast = pandas.DataFrame; predicted y with confidance interval.
    """
    HIGH = test.max()
    LOW = test.min()
    CLOSE = test.mean()

    PP = (HIGH + LOW + CLOSE) / 3
    S1 = 2 * PP - HIGH
    S2 = PP - (HIGH - LOW)
    S3 = PP * 2 - (2 * HIGH - LOW)
    S4 = PP * 3 - (3 * HIGH - LOW)
    S5 = PP * 4 - (4 * HIGH - LOW)
    R1 = 2 * PP - LOW
    R2 = PP + (HIGH - LOW)

    fig, ax = plt.subplots(figsize=(15, 5))
    # TS
    ts.plot(ax=ax, color='#886393')
    # forecast
    forecast['prediction'].plot(ax=ax, color='#5d9db1')
    ax.fill_between(forecast.index,
                    forecast.lower,
                    forecast.upper,
                    alpha=.6,
                    color='#accdd7')
    # support and resistance
    kws = dict(color='#ff6961', xmin=.6, ls='dashed')
    kws_1 = dict(color='#008807', xmin=.6, ls='dashed')
    plt.axhline(S1, **kws, label='Support 1')
    plt.axhline(S2, **kws, label='Support 2', alpha=.8)
    plt.axhline(S3, **kws, label='Support 3', alpha=.7)
    plt.axhline(S4, **kws, label='Support 4', alpha=.6)
    plt.axhline(S5, **kws, label='Support 5', alpha=.5)
    plt.axhline(R1, **kws_1, label='Resistance 1')
```

```
    plt.axhline(R2, **kws_1, label='Resistance 2', alpha=.7)
    plt.title(f'Zipcode {ts.name} prediction analysis', fontsize=15)
    plt.legend()
    plt.show()
    return fig, ax


def show_py_file_content(file='./imports_and_functions/functions.py'):
    """
    displays content of a py file output formatted as python code in jupyter notebook.

    Parameter:
    ==========
    file = `str`; default: './imports_and_functions/functions.py',
                path to the py file.
    """
    with open(file, 'r', encoding="utf8") as f:
        x = f"""```python
{f.read()}```"""
        display(Markdown(x))


def map_based_on_zipcode(map_df, mapbox_style="open-street-map"):
    """
    plots map

    Parameters:
    ===========
    mapbox_style = str; options are following:
        > "white-bg" yields an empty white canvas which results in no external HTTP requests

        > "carto-positron", "carto-darkmatter", "stamen-terrain",
          "stamen-toner" or "stamen-watercolor" yield maps composed of raster tiles
          from various public tile servers which do not require signups or access tokens

        > "open-street-map" does work
    """
    fig = px.scatter_mapbox(
        map_df,
        lat=map_df.lat,
        lon=map_df.long,
        color='Zipcode',
        zoom=11,
        size='values',
        height=1200,
        title='Zipcode location',
        center={
            'lat': map_df[map_df['Zipcode'] == '11418']['lat'].values[0],
```

```
            'lon': map_df[map_df['Zipcode'] == '11418']['long'].values[0]
        })
    # use "stamen-toner" or "carto-positron"
    fig.update_layout(mapbox_style=mapbox_style)
    fig.update_layout(margin={"r": 0, "l": 0, "b": 1})
    # fig.update_traces(marker=dict(size=20),
    #                   selector=dict(mode='markers'))
    fig.show()
    pass
```

8.2 Prophet

[72]:
```
try:
    from fbprophet import Prophet
except:
    !pip install fbprophet
    # for troubleshooting: https://facebook.github.io/prophet/docs/installation.
 ↪html#python
from statsmodels.tools.eval_measures import rmse
```

[61]:
```
actual_under_performing_models
```

[61]:
```
['11693', '11415']
```

[147]:
```
zipcode=actual_under_performing_models[0]
```

[148]:
```
ts_df[zipcode]
```

[148]:
```
date
1996-01-31    112014.0
1996-02-29    109195.0
1996-03-31    107898.0
1996-04-30    105353.0
1996-05-31    104549.0
                 …
2020-12-31    342164.0
2021-01-31    346986.0
2021-02-28    350591.0
2021-03-31    355157.0
2021-04-30    356097.0
Freq: M, Name: 11693, Length: 304, dtype: float64
```

[149]:
```
ts_proph = ts_df[zipcode]
```

[150]:
```
ts_proph = ts_proph.reset_index()
```

[151]:
```
ts_proph.columns = ['ds','y']
```

[152]:
```
ts_proph
```

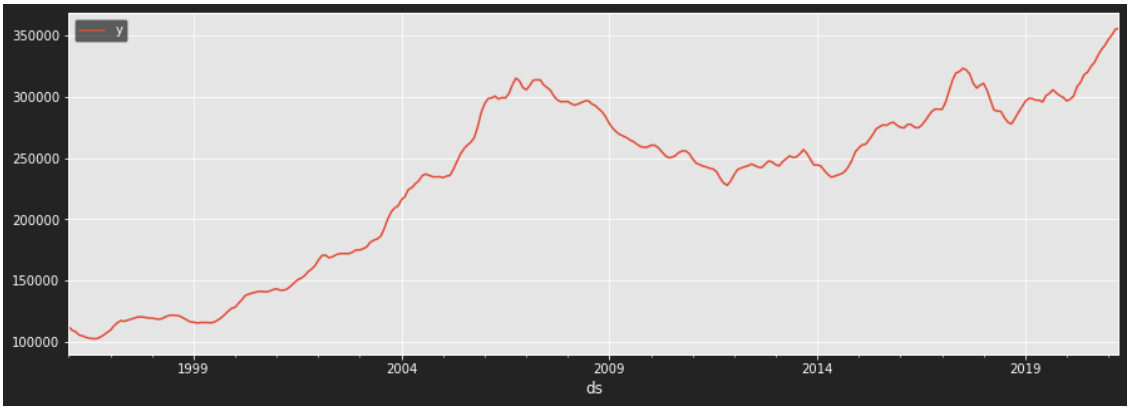
[152]:
```
            ds         y
0   1996-01-31  112014.0
1   1996-02-29  109195.0
2   1996-03-31  107898.0
3   1996-04-30  105353.0
4   1996-05-31  104549.0
..         …       …
299 2020-12-31  342164.0
300 2021-01-31  346986.0
301 2021-02-28  350591.0
302 2021-03-31  355157.0
303 2021-04-30  356097.0

[304 rows x 2 columns]
```

[66]:
```
ts_proph.dtypes
```

[66]:
```
ds    datetime64[ns]
y            float64
dtype: object
```

[67]:
```
ts_proph.plot(x='ds', y='y', figsize=(15, 5))
```

[67]:
```
<AxesSubplot:xlabel='ds'>
```

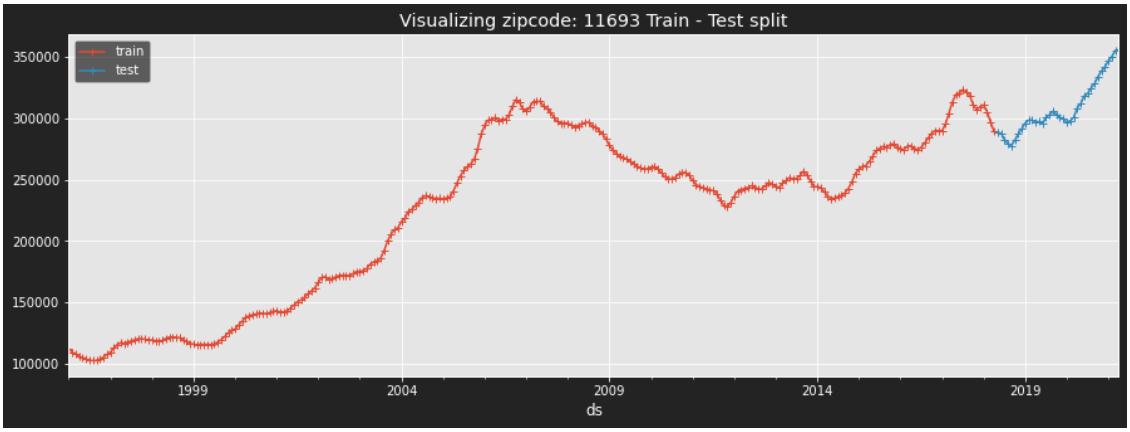


[68]:
```
len(ts_proph)
```

[68]:
```
304
```

[69]:
```
train = ts_proph.iloc[:len(ts_proph) - 36]
test = ts_proph.iloc[len(ts_proph) - 36:]
```

[70]:
```
## Visualize the train-test split split
fig, ax = plt.subplots(figsize=(15, 5))
kws = dict(x='ds',y='y', ax=ax, marker='+')
train.plot(**kws, label='train')
test.plot(**kws, label='test')
ax.legend()
plt.title(f'Visualizing zipcode: {zipcode} Train - Test split')
plt.show()
```



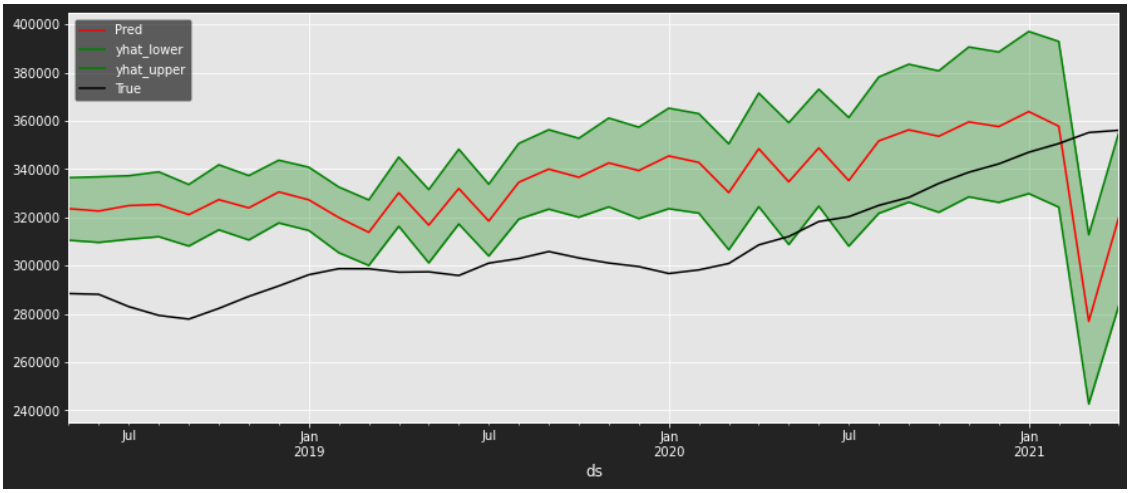
[73]:
```
m = Prophet(interval_width=0.95,
            weekly_seasonality=False,
            daily_seasonality=False,seasonality_mode='additive')
m.add_seasonality(name='monthly', period=30.5, fourier_order=5)
m.fit(train)
future = m.make_future_dataframe(periods=36, freq='MS')
forecast = m.predict(future)
```

[74]:
```
# forecast.tail()
```

[75]:
```
ax = forecast[len(ts_proph) - 36:].plot(x='ds',
                                        y='yhat',
                                        label='Pred',
                                        legend=True,
                                        figsize=(15, 6),
                                        color='red')
forecast.iloc[len(ts_proph) - 36:].plot(x='ds',
                                        y='yhat_lower',
                                        ax=ax,
                                        color='g')
forecast.iloc[len(ts_proph) - 36:].plot(x='ds',
                                        y='yhat_upper',
                                        ax=ax,
```

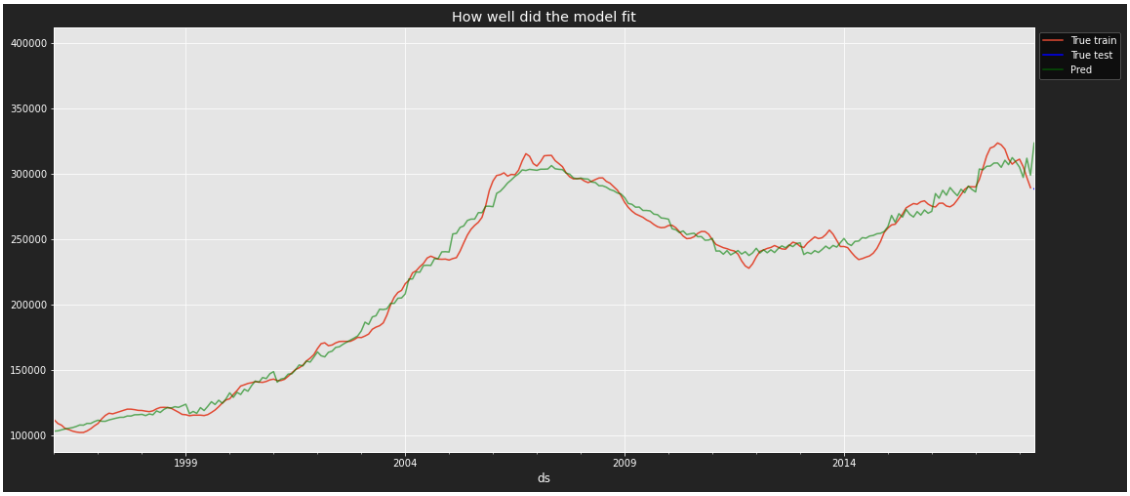
```
                                        color='g')
ax.fill_between(x=forecast.iloc[len(ts_proph) - 36:]['ds'].values,
                y1=forecast.iloc[len(ts_proph) - 36:]['yhat_lower'],
                y2=forecast.iloc[len(ts_proph) - 36:]['yhat_upper'],
                color='g',
                alpha=.3)
test.plot(x='ds', y='y', label='True', legend=True, ax=ax, color='k')
```

[75]: <AxesSubplot:xlabel='ds'>

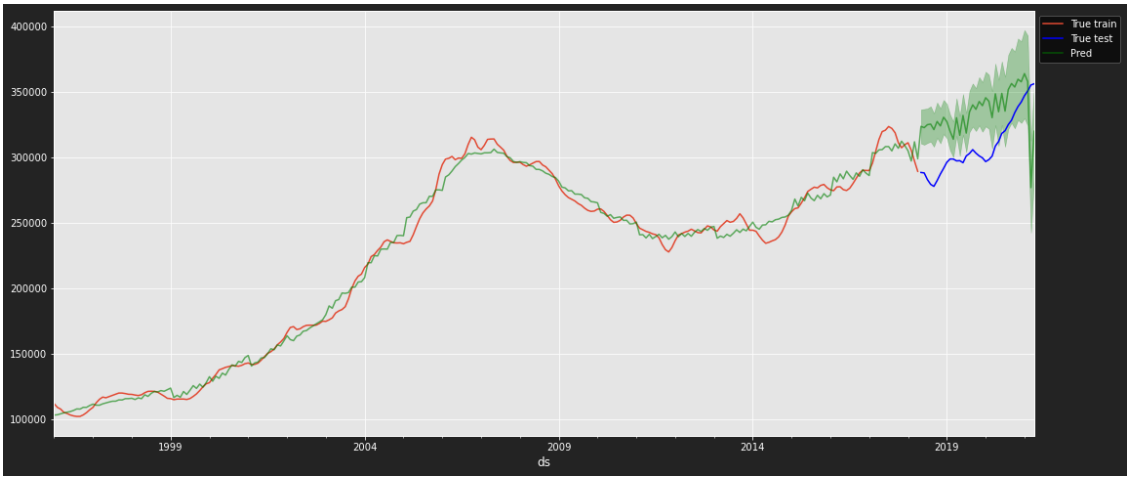


[76]:
```
ax = train.plot(x='ds',
                y='y',
                label='True train',
                legend=True,
                figsize=(18, 8))
test.plot(x='ds', y='y', label='True test', legend=True, ax=ax, color='b')
forecast.plot(x='ds',
              y='yhat',
              label='Pred',
              legend=True,
              color='g',
              ax=ax,
              alpha=.6)
ax.fill_between(x=forecast.iloc[len(ts_proph) - 36:]['ds'].values,
                y1=forecast.iloc[len(ts_proph) - 36:]['yhat_lower'],
                y2=forecast.iloc[len(ts_proph) - 36:]['yhat_upper'],
                color='g',
                alpha=.3)

ax.set_xlim((forecast.iloc[0]['ds'], forecast.iloc[len(ts_proph) - 36]['ds']))
```

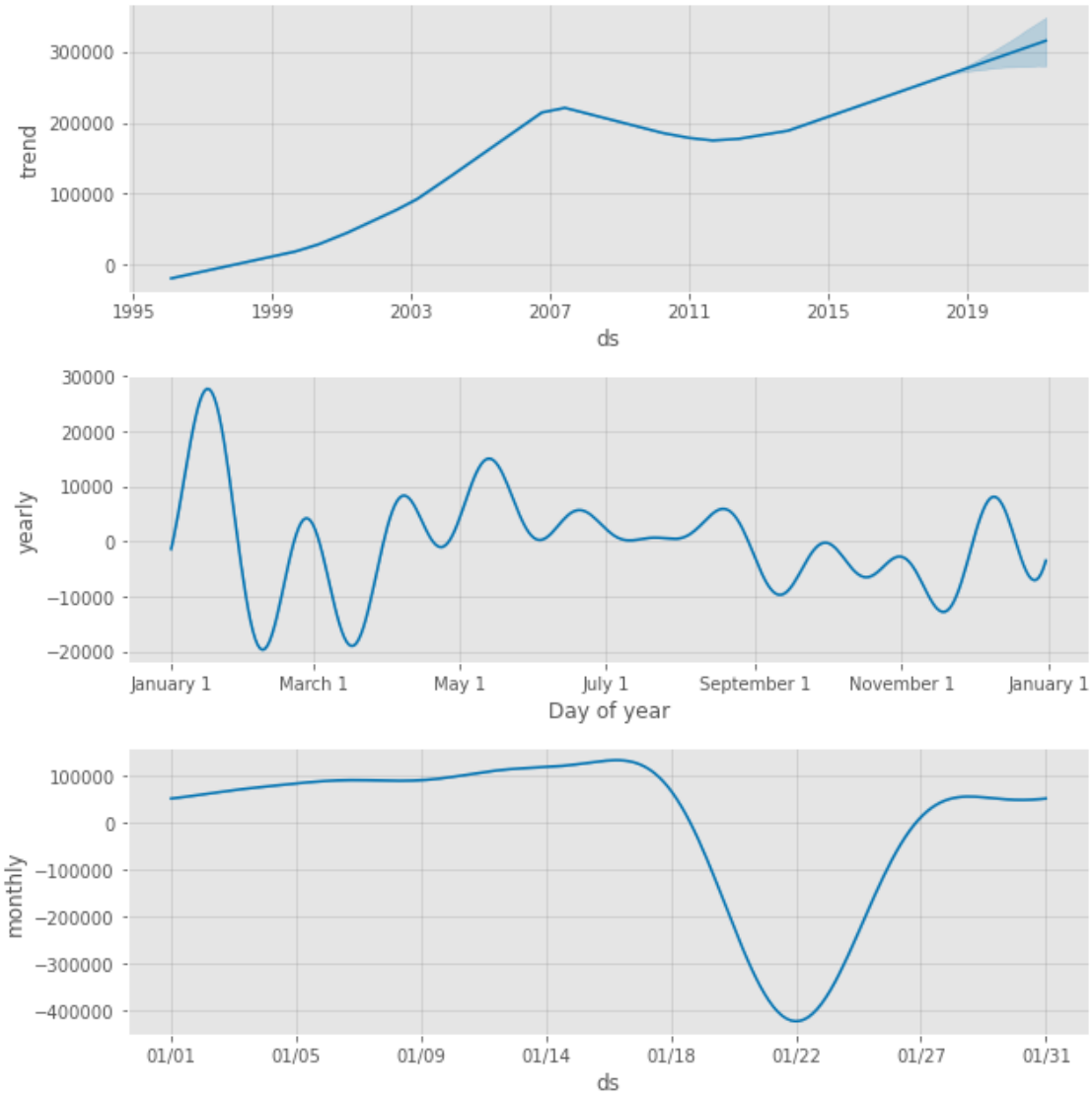
```
ax.legend(bbox_to_anchor=(1, 1), loc="upper left")
plt.title('How well did the model fit')
plt.show()
```



[77]:
```
ax = train.plot(x='ds',
                y='y',
                label='True train',
                legend=True,
                figsize=(18, 8))
test.plot(x='ds', y='y', label='True test', legend=True, ax=ax, color='b')
forecast.plot(x='ds',
              y='yhat',
              label='Pred',
              legend=True,
              color='g',
              ax=ax,
              alpha=.6)
ax.fill_between(x=forecast.iloc[len(ts_proph) - 36:]['ds'].values,
                y1=forecast.iloc[len(ts_proph) - 36:]['yhat_lower'],
                y2=forecast.iloc[len(ts_proph) - 36:]['yhat_upper'],
                color='g',
                alpha=.3)

# ax.set_xlim((forecast.iloc[len(ts_proph) - 36]['ds'], forecast.
 ↪iloc[-1]['ds']))
ax.legend(bbox_to_anchor=(1, 1), loc="upper left")
plt.show()
```



[78]:
```
with plt.style.context('ggplot'):
    m.plot_components(forecast)
```



[62]:
```
rmse(forecast.iloc[len(ts_proph) - 36:]['yhat'], test['y'])
```

[62]:
```
18461.86878802158
```

[63]:
```
test.mean(numeric_only=True)
```

[63]:
```
y    591781.972222
dtype: float64
```

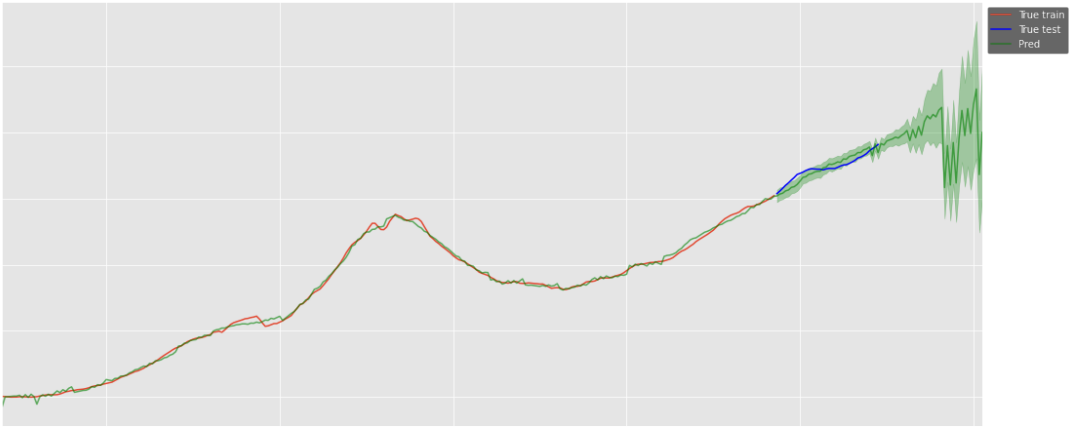
[64]:
```
test.std(numeric_only=True)
```

[64]:
```
y    11367.179322
dtype: float64
```

[ ]:

[53]:
```
m = Prophet(interval_width=0.95,
            weekly_seasonality=False,
            daily_seasonality=False,seasonality_mode='multiplicative')
m.add_seasonality(name='monthly', period=30.5, fourier_order=5)
m.fit(ts_proph)
future = m.make_future_dataframe(periods=36, freq='MS')
forecast = m.predict(future)
```

[56]:
```
ax = train.plot(x='ds',
                y='y',
                label='True train',
                legend=True,
                figsize=(18, 8))
test.plot(x='ds', y='y', label='True test', legend=True, ax=ax, color='b')
forecast.plot(x='ds',
              y='yhat',
              label='Pred',
              legend=True,
              color='g',
              ax=ax,
              alpha=.6)
ax.fill_between(x=forecast.iloc[len(ts_proph) - 36:]['ds'].values,
                y1=forecast.iloc[len(ts_proph) - 36:]['yhat_lower'],
                y2=forecast.iloc[len(ts_proph) - 36:]['yhat_upper'],
                color='g',
                alpha=.3)

# ax.set_xlim((forecast.iloc[len(ts_proph) - 36]['ds'], forecast.
 ↪iloc[-1]['ds']))
ax.legend(bbox_to_anchor=(1, 1), loc="upper left")
plt.show()
```



Not good yet

[ ]:

[ ]:

[ ]:

8.3 RNN

[ ]:
```
# gluon package
```

9 Todo:

9.1 Coding

- commenting
- grid parameter tuning option in function

9.2 EDA

- ploty line plot
- map

9.3 presentation

[ ]: