# Timeseries - Forecasting Real Estate Prices

By: Michael Holthouser

## Business Understanding

**World Wide Real Estate Investments** is looking for their new real estate investment opportunity, and they have their eyes set on Idaho. During the coronavirus pandemic, the state of Idaho had seen the largest net gain in population than any other state. World Wide Real Estate Investments wants get a piece of this pie before it's too late.

I have been tasked by World Wide Real Estate Investments to recommend the top 5 zip codes in Idaho. My responsibility is to forecast the top 5 zip codes in terms of **ROI (return of investment)** for the company to invest in.

## Data Selection

I had chosen to filter the data to only looking at zip codes in **Boise, Idaho**. I chose Boise as the "best" area to analyze not only because it is the largest city in Idaho, but because millennials are quickly relocating there. "In 2018, Boise was the fastest-growing city according to Forbes." Many Californians are moving to Boise because of the cost of living is much more affordable than California, a better work-life balance, and access to nature. Business Insider (https://www.businessinsider.com/why-millennials-are-moving-from-california-to-boise-idaho-2019-12).

### Outline

- Introduction and data cleaning
- Boise ZipCode analysis
- Time series modeling
- Conclusion

### Relevant packages and libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib
import xlrd
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.graphics.tsaplots import plot_acf
from sklearn.metrics import r2_score
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from pandas.plotting import autocorrelation_plot, lag_plot
import warnings
warnings.filterwarnings('ignore')
import itertools
import statsmodels.api as sm
from matplotlib.pylab import rcParams
plt.style.use('ggplot')
```

### Column Descriptions

- **`RegionID`** , numerical. This is a unique Id for the regions.
- **`RegionName`** , numerical. This field contains the zip code of the region.
- **`City`** , string. This column provides the specific city name of the housing data.
- **`State`** , string. This column provides the specific state name.
- **`Metro`** , string. This provides the name of the metro city that surrounds that region.
- **`CountyName`** , string. This is the county name for that region.
- **`SizeRank`** , string. This is the ranking done based on the urbanization of the area.
- **`Dates`** , The next several columns provide the month and the year with the median price of a real estate.

### Load the dataset

In [2]:
```
df = pd.read_csv('zillow_data.csv')
df.head()
```

Out[2]:

| | RegionID | RegionName | City | State | Metro | CountyName | SizeRank | 1996-04 | 1996-05 | 1996-06 | ... | 2017-07 | 2017-08 | 2017-09 | 2017-10 | 2017-11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 84654 | 60657 | Chicago | IL | Chicago | Cook | 1 | 334200.0 | 335400.0 | 336500.0 | ... | 1005500 | 1007500 | 1007800 | 1009600 | 1013300 |
| **1** | 90668 | 75070 | McKinney | TX | Dallas-Fort Worth | Collin | 2 | 235700.0 | 236900.0 | 236700.0 | ... | 308000 | 310000 | 312500 | 314100 | 315000 |
| **2** | 91982 | 77494 | Katy | TX | Houston | Harris | 3 | 210400.0 | 212200.0 | 212200.0 | ... | 321000 | 320600 | 320200 | 320400 | 320800 |
| **3** | 84616 | 60614 | Chicago | IL | Chicago | Cook | 4 | 498100.0 | 500900.0 | 503100.0 | ... | 1289800 | 1287700 | 1287400 | 1291500 | 1296600 |
| **4** | 93144 | 79936 | El Paso | TX | El Paso | El Paso | 5 | 77300.0 | 77300.0 | 77300.0 | ... | 119100 | 119400 | 120000 | 120300 | 120300 |

5 rows × 272 columns

- Initial thoughts on the dataset:
  - **RegionID**, **Metro**, **CountyName**, and **SizeRank** will be eventually removed from the dataset.

## Data Processing and Cleaning

### Check the shape of the dataframe

In [3]:
```
df.shape
```

Out[3]: (14723, 272)

- There are 14,723 rows and 272 columns contained in this dataset.

### Filter for Idaho ZipCodes only

- The investment company is interested in fast growing areas. Idaho is on the top of the list, with populations from California, Hawaii, Illinois, and West Virginia are moving there because of its affordability.
- Boise also has a rapidly growing tech sector.
- [World Population Review (https://worldpopulationreview.com/state-rankings/fastest-growing-states)](https://worldpopulationreview.com/state-rankings/fastest-growing-states)

In [4]:
```
id_df = df.loc[df['State'] == 'ID'].reset_index()
id_df.head(3)
```

Out[4]:

| | index | RegionID | RegionName | City | State | Metro | CountyName | SizeRank | 1996-04 | 1996-05 | ... | 2017-07 | 2017-08 | 2017-09 | 2017-10 | 2017-11 | 2017-12 | 201 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 448 | 94103 | 83301 | Twin Falls | ID | Twin Falls | Twin Falls | 449 | 94100.0 | 94100.0 | ... | 164300 | 164700 | 164900 | 166400 | 170000 | 173800 | 1764 |
| **1** | 596 | 94282 | 83709 | Boise | ID | Boise City | Ada | 597 | 118900.0 | 118800.0 | ... | 242200 | 251800 | 261200 | 263500 | 261100 | 257000 | 2552 |
| **2** | 1178 | 94272 | 83686 | Nampa | ID | Boise City | Canyon | 1179 | 99500.0 | 99400.0 | ... | 187100 | 188700 | 191300 | 193500 | 194500 | 195100 | 1963 |

3 rows × 273 columns

### Rename RegionName to ZipCode

- Per the column descriptions, RegionName actually means ZipCode. For reading purposes I shall rename the column to ZipCode to avoid any confusion.

In [5]:
```
id_df.rename(columns={'RegionName':'ZipCode'}, inplace=True)
id_df.head(1)
```

Out[5]:

| | index | RegionID | ZipCode | City | State | Metro | CountyName | SizeRank | 1996-04 | 1996-05 | ... | 2017-07 | 2017-08 | 2017-09 | 2017-10 | 2017-11 | 2017-12 | 2018-01 | 201 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 448 | 94103 | 83301 | Twin Falls | ID | Twin Falls | Twin Falls | 449 | 94100.0 | 94100.0 | ... | 164300 | 164700 | 164900 | 166400 | 170000 | 173800 | 176400 | 1778 |

1 rows × 273 columns

### Drop RegionID and index from dataset

- RegionID per the descriptions is a unique Identfier for each region and has no significance to the research.
- Date will eventually become our new index

In [6]:
```
id_df.drop(columns=['RegionID', 'index'], inplace=True)
id_df.head(1)
```

Out[6]:

| | ZipCode | City | State | Metro | CountyName | SizeRank | 1996-04 | 1996-05 | 1996-06 | 1996-07 | ... | 2017-07 | 2017-08 | 2017-09 | 2017-10 | 2017-11 | 2017-12 | 2018-01 | 201 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 83301 | Twin Falls | ID | Twin Falls | Twin Falls | 449 | 94100.0 | 94100.0 | 94200.0 | 94200.0 | ... | 164300 | 164700 | 164900 | 166400 | 170000 | 173800 | 176400 | 1778 |

1 rows × 271 columns

### How many ZipCodes are in Idaho?

In [7]:
```
print(f'Number of ID zipcodes: {len(id_df)}')
```

```
Number of ID zipcodes: 110
```

- This was merely for curiosities sake, to see how many ZipCodes are within the state of Idaho.

### Check for missing data

In [8]:
```
id_df.isna().any()
```

Out[8]:
```
ZipCode       False
City          False
State         False
Metro          True
CountyName    False
              ...
2017-12       False
2018-01       False
2018-02       False
2018-03       False
2018-04       False
Length: 271, dtype: bool
```

- It appears we do have some missing data that will need to be delt with.

### Reshape from wide to long format

- Takes the zillow_data dataset in wide form or a subset of the zillow_dataset.
  Returns a long-form datetime dataframe with the datetime column names as the index and the values as the 'values' column.
- If more than one row is passes in the wide-form dataset, the values column will be the mean of the values from the datetime columns in all of the rows.

In [9]:
```
def melt_data(df):
    melted = pd.melt(df, id_vars=['ZipCode', 'City', 'State', 'CountyName', 'Metro', 'SizeRank'], var_name='Date')
    melted['Date'] = pd.to_datetime(melted['Date'], infer_datetime_format=True)
    melted = melted.dropna(subset=['value'])
    return melted
```

In [10]:
```
melted_id = melt_data(id_df)
melted_id.head(3)
```

Out[10]:

| | ZipCode | City | State | CountyName | Metro | SizeRank | Date | value |
|---|---|---|---|---|---|---|---|---|
| **0** | 83301 | Twin Falls | ID | Twin Falls | Twin Falls | 449 | 1996-04-01 | 94100.0 |
| **1** | 83709 | Boise | ID | Ada | Boise City | 597 | 1996-04-01 | 118900.0 |
| **2** | 83686 | Nampa | ID | Canyon | Boise City | 1179 | 1996-04-01 | 99500.0 |

### Check info details of the new dataframe

In [11]:
```
1 melted_id.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 27363 entries, 0 to 29149
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   ZipCode     27363 non-null  int64
 1   City        27363 non-null  object
 2   State       27363 non-null  object
 3   CountyName  27363 non-null  object
 4   Metro       22012 non-null  object
 5   SizeRank    27363 non-null  int64
 6   Date        27363 non-null  datetime64[ns]
 7   value       27363 non-null  float64
dtypes: datetime64[ns](1), float64(1), int64(2), object(4)
memory usage: 1.9+ MB
```

- From the info details we can see that the columns have their appropriate data types, and Metro is the only column to have missing data.

### Set Date as the new index

- Setting the datetime as the index is important with time series data. It makes filtering for data by datetime super convient and easy.

In [12]:
```
1 melted_id.set_index('Date', inplace=True)
2 melted_id.head(1)
```

Out[12]:

| Date | ZipCode | City | State | CountyName | Metro | SizeRank | value |
|---|---|---|---|---|---|---|---|
| **1996-04-01** | 83301 | Twin Falls | ID | Twin Falls | Twin Falls | 449 | 94100.0 |

### Filter dataframe to only contain records from 2012-01 till 2018-04 to account for the housing bubble crisis

- In 2008 a financial crisis occurred leading the collapse of a house bubble. The market didn't recover till late 2012.
- The Declines in residential investment preceded the Great Recession and were followed by a reduction in household spending and business investment.
- Wikipedia (https://en.wikipedia.org/wiki/Subprime_mortgage_crisis)

In [13]:
```
1 post_bubble = melted_id.loc['2012-01-01':'2018-04-01']
```

### Resample date index to 'MS' (month start)

- This will allow me to to perform calculations or generate calculations on a monthly time scale.

In [14]:
```
1 post_bubble.index
```

Out[14]:
```
DatetimeIndex(['2012-01-01', '2012-01-01', '2012-01-01', '2012-01-01',
               '2012-01-01', '2012-01-01', '2012-01-01', '2012-01-01',
               '2012-01-01', '2012-01-01',
               ...
               '2018-04-01', '2018-04-01', '2018-04-01', '2018-04-01',
               '2018-04-01', '2018-04-01', '2018-04-01', '2018-04-01',
               '2018-04-01', '2018-04-01'],
              dtype='datetime64[ns]', name='Date', length=8360, freq=None)
```

In [15]:
```
post_bubble.resample('MS').mean().index
```

Out[15]:
```
DatetimeIndex(['2012-01-01', '2012-02-01', '2012-03-01', '2012-04-01',
               '2012-05-01', '2012-06-01', '2012-07-01', '2012-08-01',
               '2012-09-01', '2012-10-01', '2012-11-01', '2012-12-01',
               '2013-01-01', '2013-02-01', '2013-03-01', '2013-04-01',
               '2013-05-01', '2013-06-01', '2013-07-01', '2013-08-01',
               '2013-09-01', '2013-10-01', '2013-11-01', '2013-12-01',
               '2014-01-01', '2014-02-01', '2014-03-01', '2014-04-01',
               '2014-05-01', '2014-06-01', '2014-07-01', '2014-08-01',
               '2014-09-01', '2014-10-01', '2014-11-01', '2014-12-01',
               '2015-01-01', '2015-02-01', '2015-03-01', '2015-04-01',
               '2015-05-01', '2015-06-01', '2015-07-01', '2015-08-01',
               '2015-09-01', '2015-10-01', '2015-11-01', '2015-12-01',
               '2016-01-01', '2016-02-01', '2016-03-01', '2016-04-01',
               '2016-05-01', '2016-06-01', '2016-07-01', '2016-08-01',
               '2016-09-01', '2016-10-01', '2016-11-01', '2016-12-01',
               '2017-01-01', '2017-02-01', '2017-03-01', '2017-04-01',
               '2017-05-01', '2017-06-01', '2017-07-01', '2017-08-01',
               '2017-09-01', '2017-10-01', '2017-11-01', '2017-12-01',
               '2018-01-01', '2018-02-01', '2018-03-01', '2018-04-01'],
              dtype='datetime64[ns]', name='Date', freq='MS')
```

### Preview Head and Tail of the post_bubble dataframe

In [16]:
```
post_bubble.head()
```

Out[16]:

| Date | ZipCode | City | State | CountyName | Metro | SizeRank | value |
|---|---|---|---|---|---|---|---|
| **2012-01-01** | 83301 | Twin Falls | ID | Twin Falls | Twin Falls | 449 | 117200.0 |
| **2012-01-01** | 83709 | Boise | ID | Ada | Boise City | 597 | 146100.0 |
| **2012-01-01** | 83686 | Nampa | ID | Canyon | Boise City | 1179 | 113700.0 |
| **2012-01-01** | 83704 | Boise | ID | Ada | Boise City | 1344 | 123800.0 |
| **2012-01-01** | 83854 | Post Falls | ID | Kootenai | Coeur d'Alene | 1407 | 169700.0 |

In [17]:
```
post_bubble.tail()
```

Out[17]:

| Date | ZipCode | City | State | CountyName | Metro | SizeRank | value |
|---|---|---|---|---|---|---|---|
| **2018-04-01** | 83434 | Menan | ID | Jefferson | Idaho Falls | 14361 | 183600.0 |
| **2018-04-01** | 83813 | Cocolalla | ID | Bonner | Sandpoint | 14431 | 279900.0 |
| **2018-04-01** | 83803 | Bayview | ID | Kootenai | Coeur d'Alene | 14609 | 260100.0 |
| **2018-04-01** | 83846 | Wallace | ID | Shoshone | NaN | 14683 | 83800.0 |
| **2018-04-01** | 83821 | Coolin | ID | Bonner | Sandpoint | 14704 | 540400.0 |

### Check if missing values still remain

In [18]:
```
post_bubble.isna().sum()
```

Out[18]:
```
ZipCode          0
City             0
State            0
CountyName       0
Metro         1748
SizeRank         0
value            0
dtype: int64
```

- Metro still has 1,748 missing values. With that being said, I will drop the Metro column since it does not have any effect on our numerical calculations.

In [19]:
```
# Drop Metro column
post_bubble.drop(columns=['Metro'], inplace=True)
```

### Check info() to ensure missing value are no longer present

In [20]:
```
1 post_bubble.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8360 entries, 2012-01-01 to 2018-04-01
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   ZipCode     8360 non-null   int64  
 1   City        8360 non-null   object 
 2   State       8360 non-null   object 
 3   CountyName  8360 non-null   object 
 4   SizeRank    8360 non-null   int64  
 5   value       8360 non-null   float64
dtypes: float64(1), int64(2), object(3)
memory usage: 457.2+ KB
```

# EDA : Exploratory Data Analysis

### Filter for ZipCodes located within Boise

In [21]:
```
1 boise = post_bubble.loc[post_bubble['City'] == 'Boise']
2 boise
```

Out[21]:

| Date | ZipCode | City | State | CountyName | SizeRank | value |
|---|---|---|---|---|---|---|
| **2012-01-01** | 83709 | Boise | ID | Ada | 597 | 146100.0 |
| **2012-01-01** | 83704 | Boise | ID | Ada | 1344 | 123800.0 |
| **2012-01-01** | 83706 | Boise | ID | Ada | 1784 | 156600.0 |
| **2012-01-01** | 83705 | Boise | ID | Ada | 2963 | 111100.0 |
| **2012-01-01** | 83702 | Boise | ID | Ada | 3636 | 244200.0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **2018-04-01** | 83702 | Boise | ID | Ada | 3636 | 441000.0 |
| **2018-04-01** | 83713 | Boise | ID | Ada | 3820 | 252800.0 |
| **2018-04-01** | 83703 | Boise | ID | Ada | 5765 | 264200.0 |
| **2018-04-01** | 83716 | Boise | ID | Ada | 5996 | 305600.0 |
| **2018-04-01** | 83712 | Boise | ID | Ada | 8622 | 433500.0 |

684 rows × 6 columns

In [22]:
```
1 boise['ZipCode'].nunique()
```

Out[22]: 9

In [23]:
```
1 boise['ZipCode'].unique()
```

Out[23]: array([83709, 83704, 83706, 83705, 83702, 83713, 83703, 83716, 83712])

- Within Boise, there are exactly **9** ZipCodes. This is very convenient so we do not have to do anymore filtering to narrow down our sample size of ZipCodes.

In [24]:
```
1 boise['CountyName'].unique()
```

Out[24]: array(['Ada'], dtype=object)

The whole city of boise lies in the County of Ada.


### Size Rank

- Next lets take a look at the SizeRank of each zip code with in Boise.

In [25]:
```
group_rnk = boise.groupby('ZipCode').agg({"SizeRank": "first"})
```

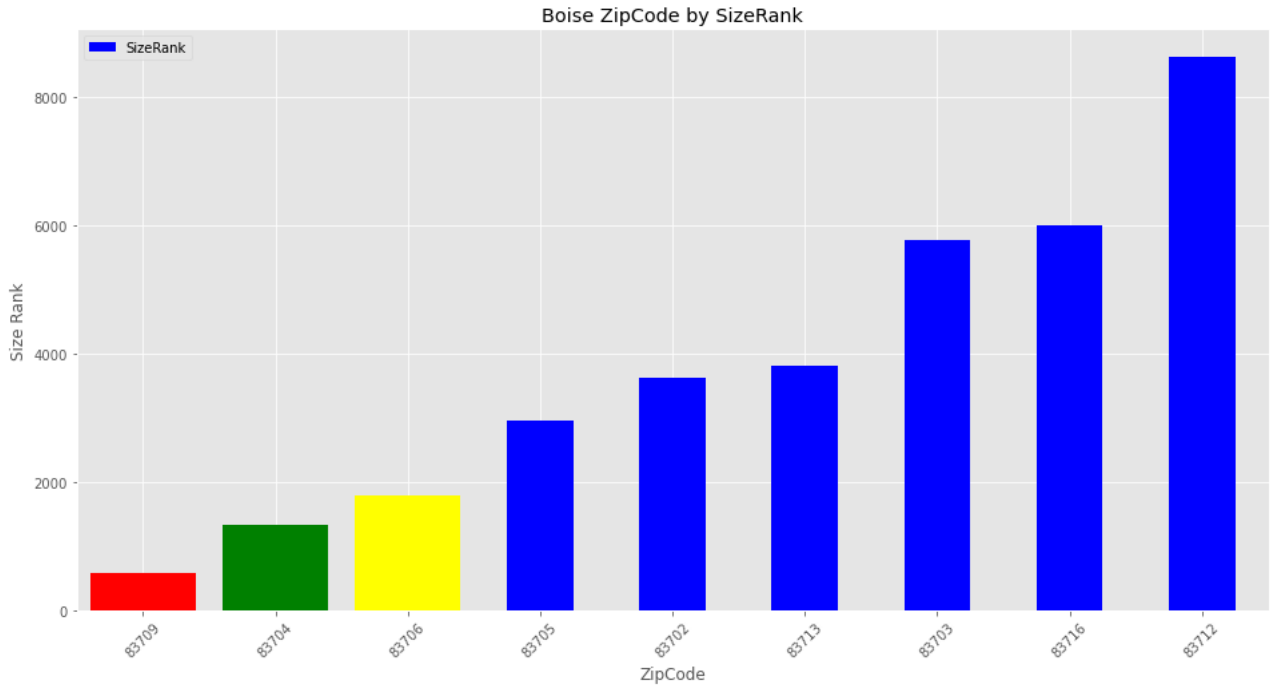
In [26]:
```
group_rnk = group_rnk.sort_values("SizeRank")
group_rnk
```

Out[26]:

| ZipCode | SizeRank |
|---|---|
| **83709** | 597 |
| **83704** | 1344 |
| **83706** | 1784 |
| **83705** | 2963 |
| **83702** | 3636 |
| **83713** | 3820 |
| **83703** | 5765 |
| **83716** | 5996 |
| **83712** | 8622 |

- Looks their is one ZipCode that clearly stands out amongst the rest in terms of SizeRank **83709**. **83704** and **83706** are moderately clearly rank higher than the rest of the ZipCodes as well. I will be interesting to see if that play a role in the "best" ZipCodes in our final model.

In [27]:
```
group_rnk.plot.bar(color='blue', figsize=(16,8))

# Assign different colors to first three bars
colors = ['red', 'green', 'yellow']
for i in range(3):
    plt.bar(i, group_rnk.iloc[i], color=colors[i])

plt.title('Boise ZipCode by SizeRank')
plt.ylabel('Size Rank')
plt.xticks(rotation=45)
plt.show();
```




- Looking at the graph above, the smaller the number, the higher the rank. I put the top three ZipCodes in different colors to stand out from the rest.

### ZipCodes grouped by mean value

In [28]:
```
group_val = boise.groupby('ZipCode').agg({'value': 'mean'})
group_val
```

Out[28]:

| ZipCode | value |
|---|---|
| **83702** | 319838.157895 |
| **83703** | 190135.526316 |
| **83704** | 164397.368421 |
| **83705** | 154701.315789 |
| **83706** | 200582.894737 |
| **83709** | 197040.789474 |
| **83712** | 333967.105263 |
| **83713** | 195805.263158 |
| **83716** | 252846.052632 |

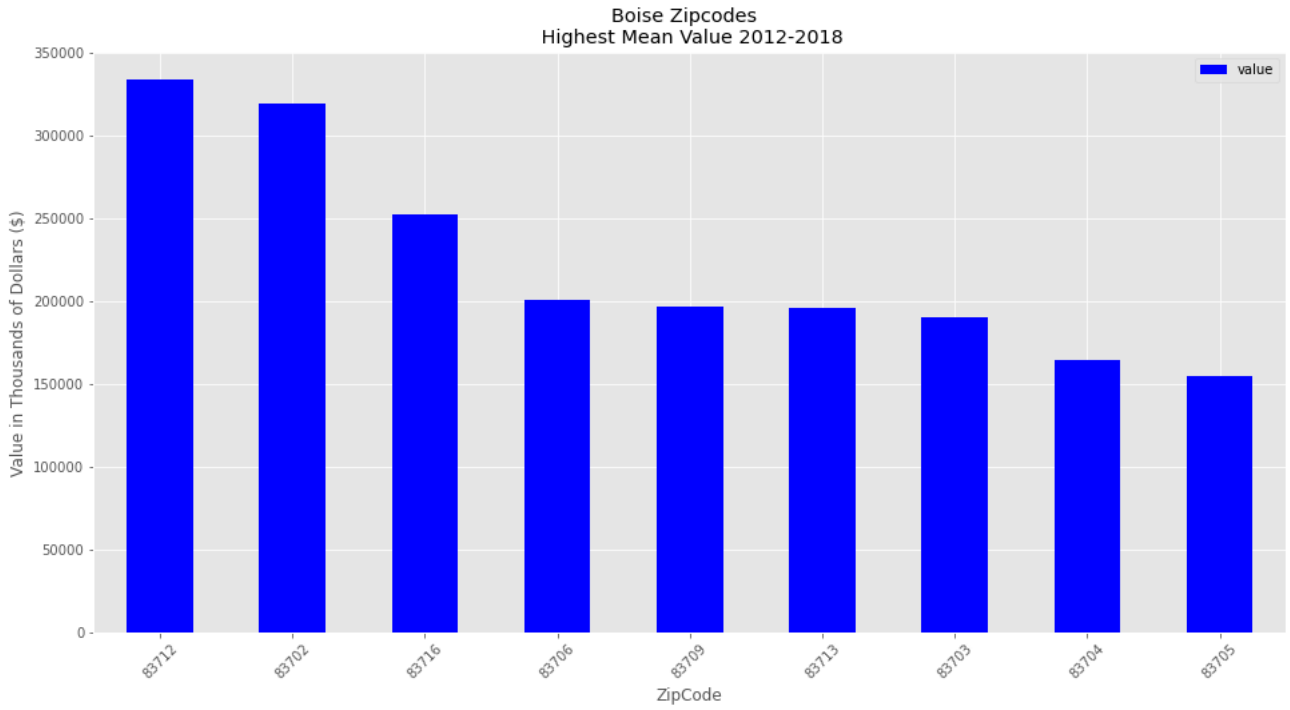
In [29]:
```
group_val = group_val.sort_values('value', ascending=False)
group_val
```

Out[29]:

| ZipCode | value |
|---|---|
| **83712** | 333967.105263 |
| **83702** | 319838.157895 |
| **83716** | 252846.052632 |
| **83706** | 200582.894737 |
| **83709** | 197040.789474 |
| **83713** | 195805.263158 |
| **83703** | 190135.526316 |
| **83704** | 164397.368421 |
| **83705** | 154701.315789 |

- This is a list of Boise ZipCodes and the mean value of the homes in descending order. 83709, 83704, and 83706 were the top three highest ranked zipcodes in terms of SizeRank. However in terms of mean value, they did not land in the top 3. The top three ZipCodes in terms of mean value were **83712**, **83702**, **83716**.

In [30]:
```
group_val.plot.bar(color='blue', figsize=(16,8))

plt.title('Boise Zipcodes  \n Highest Mean Value 2012-2018')
plt.ylabel('Value in Thousands of Dollars ($)')
plt.xticks(rotation=45)
plt.show();
```

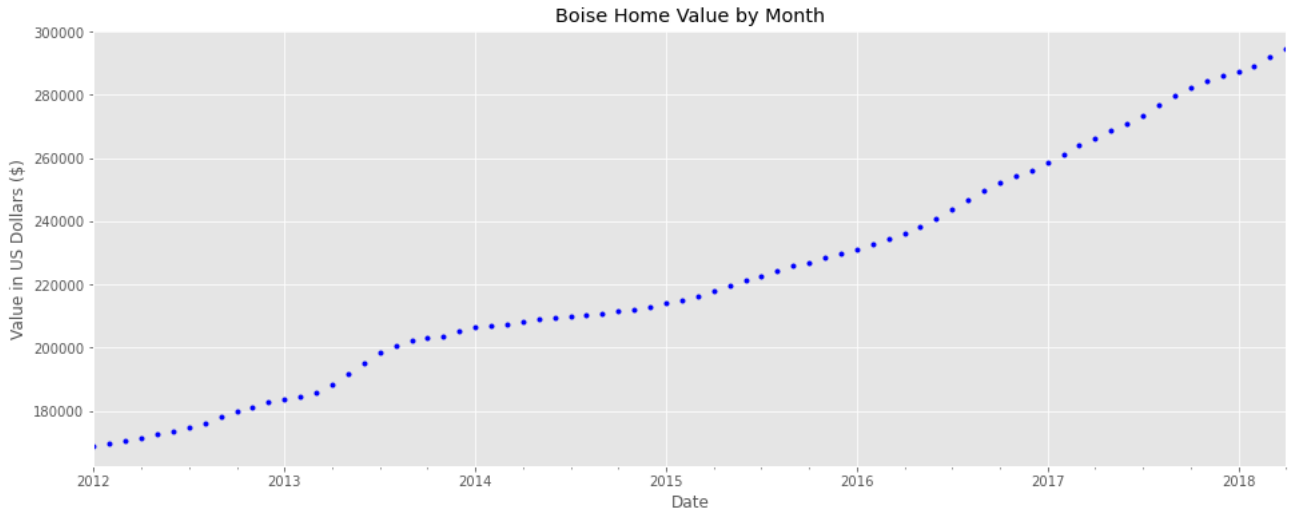



### Drop SizeRank column

- SizeRank will not have any influence in my findings, therefore it will be dropped from the dataset.

In [31]:
```
boise.drop(columns=['SizeRank'], inplace=True)
```

In [32]:
```
monthly_data = boise.resample('MS').mean()['value']
monthly_data.plot(figsize=(16,6), style='.b')
plt.title('Boise Home Value by Month')
plt.ylabel('Value in US Dollars ($)')
plt.show();

plt.savefig('boise_home_value_monthly')
```




```
<Figure size 432x288 with 0 Axes>
```

- From 2012 there has been an increase in home value every year. Several factors can lead to this:
  - Homes in Boise are affordable compared to homes in more densly populated areas.
  - The law of supply and demand. More and more of the population is moving to Idaho driving the cost of homes up.
  - With more of the population relocating to Boise, it drives the price up.

### Closer look at Boise ZipCodes

- Let's take a deeper look into the top 5 zipcodes in terms of mean value.
- I will list them in order from largest mean value of homes to the lowest mean value homes.

In [33]:
```
group_val
```

Out[33]:

| ZipCode | value |
|---|---|
| **83712** | 333967.105263 |
| **83702** | 319838.157895 |
| **83716** | 252846.052632 |
| **83706** | 200582.894737 |
| **83709** | 197040.789474 |
| **83713** | 195805.263158 |
| **83703** | 190135.526316 |
| **83704** | 164397.368421 |
| **83705** | 154701.315789 |

### 83712

In [34]:
```
boise.loc[boise['ZipCode'] == 83712]
```

Out[34]:

| Date | ZipCode | City | State | CountyName | value |
|---|---|---|---|---|---|
| **2012-01-01** | 83712 | Boise | ID | Ada | 254500.0 |
| **2012-02-01** | 83712 | Boise | ID | Ada | 255900.0 |
| **2012-03-01** | 83712 | Boise | ID | Ada | 257200.0 |
| **2012-04-01** | 83712 | Boise | ID | Ada | 258800.0 |
| **2012-05-01** | 83712 | Boise | ID | Ada | 261000.0 |
| **...** | ... | ... | ... | ... | ... |
| **2017-12-01** | 83712 | Boise | ID | Ada | 417900.0 |
| **2018-01-01** | 83712 | Boise | ID | Ada | 424400.0 |
| **2018-02-01** | 83712 | Boise | ID | Ada | 428900.0 |
| **2018-03-01** | 83712 | Boise | ID | Ada | 431800.0 |
| **2018-04-01** | 83712 | Boise | ID | Ada | 433500.0 |

76 rows × 5 columns

- The mean value of a home located in this ZipCode was **254,500** dollars in 2012.
- Six year later in 2018, the mean value of a home in this ZipCode grew to **433,500** dollars.

### 83702

In [35]:
```
boise.loc[boise['ZipCode'] == 83702]
```

Out[35]:

| Date | ZipCode | City | State | CountyName | value |
|---|---|---|---|---|---|
| **2012-01-01** | 83702 | Boise | ID | Ada | 244200.0 |
| **2012-02-01** | 83702 | Boise | ID | Ada | 245800.0 |
| **2012-03-01** | 83702 | Boise | ID | Ada | 247800.0 |
| **2012-04-01** | 83702 | Boise | ID | Ada | 249200.0 |
| **2012-05-01** | 83702 | Boise | ID | Ada | 250800.0 |
| **...** | ... | ... | ... | ... | ... |
| **2017-12-01** | 83702 | Boise | ID | Ada | 416600.0 |
| **2018-01-01** | 83702 | Boise | ID | Ada | 423600.0 |
| **2018-02-01** | 83702 | Boise | ID | Ada | 429300.0 |
| **2018-03-01** | 83702 | Boise | ID | Ada | 436000.0 |
| **2018-04-01** | 83702 | Boise | ID | Ada | 441000.0 |

76 rows × 5 columns

- The mean value of a home located in this ZipCode was **244,200** dollars in 2012.
- Six year later in 2018, the mean value of a home in this ZipCode grew to **441,000** dollars.

### 83716

In [36]:
```
1 boise.loc[boise['ZipCode'] == 83716]
```

Out[36]:

| Date | ZipCode | City | State | CountyName | value |
|---|---|---|---|---|---|
| **2012-01-01** | 83716 | Boise | ID | Ada | 200900.0 |
| **2012-02-01** | 83716 | Boise | ID | Ada | 201900.0 |
| **2012-03-01** | 83716 | Boise | ID | Ada | 203000.0 |
| **2012-04-01** | 83716 | Boise | ID | Ada | 204100.0 |
| **2012-05-01** | 83716 | Boise | ID | Ada | 205500.0 |
| **...** | ... | ... | ... | ... | ... |
| **2017-12-01** | 83716 | Boise | ID | Ada | 299300.0 |
| **2018-01-01** | 83716 | Boise | ID | Ada | 299000.0 |
| **2018-02-01** | 83716 | Boise | ID | Ada | 300000.0 |
| **2018-03-01** | 83716 | Boise | ID | Ada | 303000.0 |
| **2018-04-01** | 83716 | Boise | ID | Ada | 305600.0 |

76 rows × 5 columns

- The mean value of a home located in this ZipCode was **200,900** dollars in 2012.
- Six year later in 2018, the mean value of a home in this ZipCode grew to **305,600** dollars.

### 83706

In [37]:
```
1 boise.loc[boise['ZipCode'] == 83706]
```

Out[37]:

| Date | ZipCode | City | State | CountyName | value |
|---|---|---|---|---|---|
| **2012-01-01** | 83706 | Boise | ID | Ada | 156600.0 |
| **2012-02-01** | 83706 | Boise | ID | Ada | 156900.0 |
| **2012-03-01** | 83706 | Boise | ID | Ada | 157000.0 |
| **2012-04-01** | 83706 | Boise | ID | Ada | 156900.0 |
| **2012-05-01** | 83706 | Boise | ID | Ada | 156700.0 |
| **...** | ... | ... | ... | ... | ... |
| **2017-12-01** | 83706 | Boise | ID | Ada | 262500.0 |
| **2018-01-01** | 83706 | Boise | ID | Ada | 262300.0 |
| **2018-02-01** | 83706 | Boise | ID | Ada | 263300.0 |
| **2018-03-01** | 83706 | Boise | ID | Ada | 265700.0 |
| **2018-04-01** | 83706 | Boise | ID | Ada | 267500.0 |

76 rows × 5 columns

- The mean value of a home located in this ZipCode was **156,600** dollars in 2012.
- Six year later in 2018, the mean value of a home in this ZipCode grew to **267,500** dollars.

### 83709

In [38]:
```
1 boise.loc[boise['ZipCode'] == 83709]
```

Out[38]:

| Date | ZipCode | City | State | CountyName | value |
|---|---|---|---|---|---|
| **2012-01-01** | 83709 | Boise | ID | Ada | 146100.0 |
| **2012-02-01** | 83709 | Boise | ID | Ada | 146400.0 |
| **2012-03-01** | 83709 | Boise | ID | Ada | 147000.0 |
| **2012-04-01** | 83709 | Boise | ID | Ada | 147800.0 |
| **2012-05-01** | 83709 | Boise | ID | Ada | 148900.0 |
| **...** | ... | ... | ... | ... | ... |
| **2017-12-01** | 83709 | Boise | ID | Ada | 257000.0 |
| **2018-01-01** | 83709 | Boise | ID | Ada | 255200.0 |
| **2018-02-01** | 83709 | Boise | ID | Ada | 255100.0 |
| **2018-03-01** | 83709 | Boise | ID | Ada | 256200.0 |
| **2018-04-01** | 83709 | Boise | ID | Ada | 256600.0 |

76 rows × 5 columns

- The mean value of a home located in this ZipCode was **146,100** dollars in 2012.
- Six year later in 2018, the mean value of a home in this ZipCode grew to **256,600** dollars.

# Sarima Modeling:

### Helper Functions:

- I have created some functions to help with the modeling process.

In [39]:
```
## Finding the best Sarimax
def find_best_SARIMAX(data, pdq, pdqs):
    ans = []
    for comb in pdq:
        for combs in pdqs:
            try:
                mod = sm.tsa.statespace.SARIMAX(data,
                                                order=comb,
                                                seasonal_order=combs,
                                                enforce_stationarity=False,
                                                enforce_invertibility=False)

                output = mod.fit()
                ans.append([comb, combs, output.aic])
                print('ARIMA {} x {}: AIC Calculated={}'.format(comb, combs, output.aic))
            except:
                continue
    return ans

## Fit sarimax model
def fit_SARIMAX_model(data, order, seasonal_order):
    model = sm.tsa.statespace.SARIMAX(data,
                                      order=order,
                                      seasonal_order=seasonal_order,
                                      enforce_stationarity=False,
                                      enforce_invertibility=False)
    output = model.fit()
    output.plot_diagnostics(figsize=(15, 18))
    return output

## ACF and PACF funtions
# acf and pacf function

def plot_acf_pacf(ts, figsize=(12,10), lags = 24, zipcode = 'add'):
    '''Plots the ACF and PACF of the time series.'''
    fig,ax = plt.subplots(nrows=3,
                          figsize = figsize)

    #plot time series
    ts.plot(ax=ax[0])

    #plot acf, pacf
    plot_acf(ts,ax=ax[1], lags=lags)
    plot_pacf(ts, ax=ax[2], lags=lags)
    fig.tight_layout()

    fig.suptitle(f'Zipcode: {zipcode}',y=1.1, fontsize=20)

    for a in ax[1:]:
        a.xaxis.set_major_locator(matplotlib.ticker.MaxNLocator(min_n_ticks=lags, integer = True))
        a.xaxis.grid()
    return fig,ax


```

### Drop irrelevant columns for the modeling process

In [40]:
```
boise.drop(columns=['City', 'State', 'CountyName'], inplace=True)
```

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

Out[41]:

| Date | ZipCode | value |
|---|---|---|
| **2012-01-01** | 83709 | 146100.0 |
| **2012-01-01** | 83704 | 123800.0 |
| **2012-01-01** | 83706 | 156600.0 |
| **2012-01-01** | 83705 | 111100.0 |
| **2012-01-01** | 83702 | 244200.0 |

- For the modeling process, city, state, and CountyName carry no weight and are not necessary for the dataset.

### Grouping Boise values by ZipCode

In [42]:
```
boise_grouped = boise.groupby(['ZipCode'], as_index = True).resample('MS').sum()
boise_grouped.head()
```

Out[42]:

| | | ZipCode | value |
|---|---|---|---|
| **ZipCode** | **Date** | | |
| **83702** | **2012-01-01** | 83702 | 244200.0 |
| | **2012-02-01** | 83702 | 245800.0 |
| | **2012-03-01** | 83702 | 247800.0 |
| | **2012-04-01** | 83702 | 249200.0 |
| | **2012-05-01** | 83702 | 250800.0 |

### Drop ZipCode column

In [43]:
```
boise_grouped.drop(columns=['ZipCode'], inplace=True)
boise_grouped.head()
```

Out[43]:

| | | value |
|---|---|---|
| **ZipCode** | **Date** | |
| **83702** | **2012-01-01** | 244200.0 |
| | **2012-02-01** | 245800.0 |
| | **2012-03-01** | 247800.0 |
| | **2012-04-01** | 249200.0 |
| | **2012-05-01** | 250800.0 |

- Now the dataframe is formatted appropriately for modeling.

## 83709

In [44]:
```
seven_zero_nine = boise_grouped.loc[83709]
seven_zero_nine.head()
```

Out[44]:

| | value |
|---|---|
| **Date** | |
| **2012-01-01** | 146100.0 |
| **2012-02-01** | 146400.0 |
| **2012-03-01** | 147000.0 |
| **2012-04-01** | 147800.0 |
| **2012-05-01** | 148900.0 |

### Find the cut off to determine 80/20 split for the training and test data

In [45]:
```
# find the index which allows us to split off 20% of the data
cutoff = round(seven_zero_nine.shape[0]*0.8)
cutoff
```

Out[45]: 61

- at index 61 is where we will begin the test set. From 0 through 60 will be the considered training data

In [46]:
```
1 seven_zero_nine[61:]
```

Out[46]:

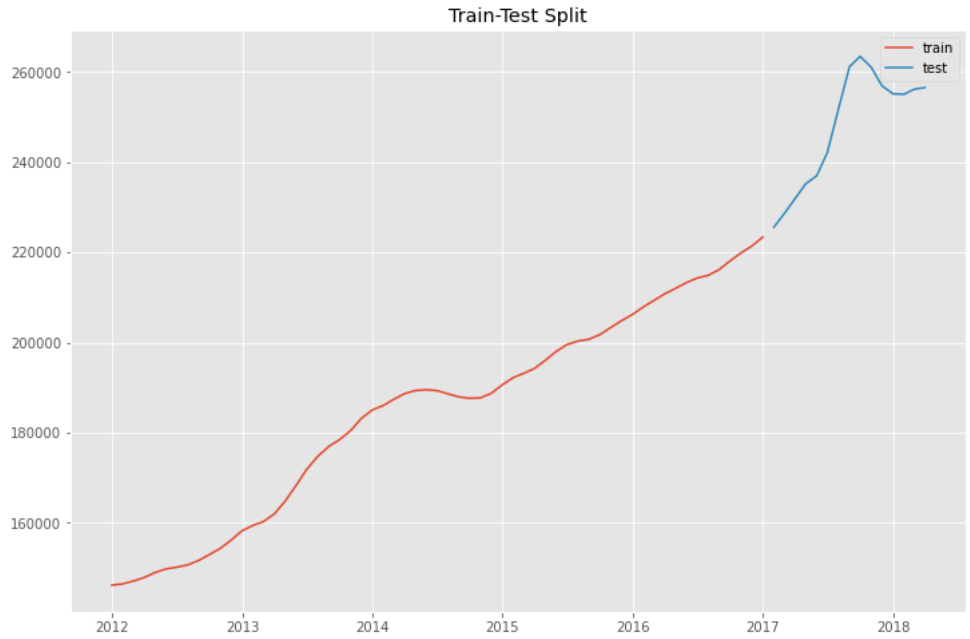
| Date | value |
|---|---|
| 2017-02-01 | 225600.0 |
| 2017-03-01 | 228500.0 |
| 2017-04-01 | 231900.0 |
| 2017-05-01 | 235200.0 |
| 2017-06-01 | 237000.0 |
| 2017-07-01 | 242200.0 |
| 2017-08-01 | 251800.0 |
| 2017-09-01 | 261200.0 |
| 2017-10-01 | 263500.0 |
| 2017-11-01 | 261100.0 |
| 2017-12-01 | 257000.0 |
| 2018-01-01 | 255200.0 |
| 2018-02-01 | 255100.0 |
| 2018-03-01 | 256200.0 |
| 2018-04-01 | 256600.0 |

- Our training data will be from 2012-01-01 and end and 2017-01-01. The test data begins at 2017-02-01 and ends at 2018-04-01.

### Create train/test variables

In [47]:
```
1 train = seven_zero_nine[:cutoff]
2
3 test = seven_zero_nine[cutoff:]
4
5 fig, ax = plt.subplots(figsize=(12, 8))
6 ax.plot(train, label='train')
7 ax.plot(test, label='test')
8 ax.set_title('Train-Test Split');
9 plt.legend()
```

Out[47]: <matplotlib.legend.Legend at 0x7faf4cb69400>

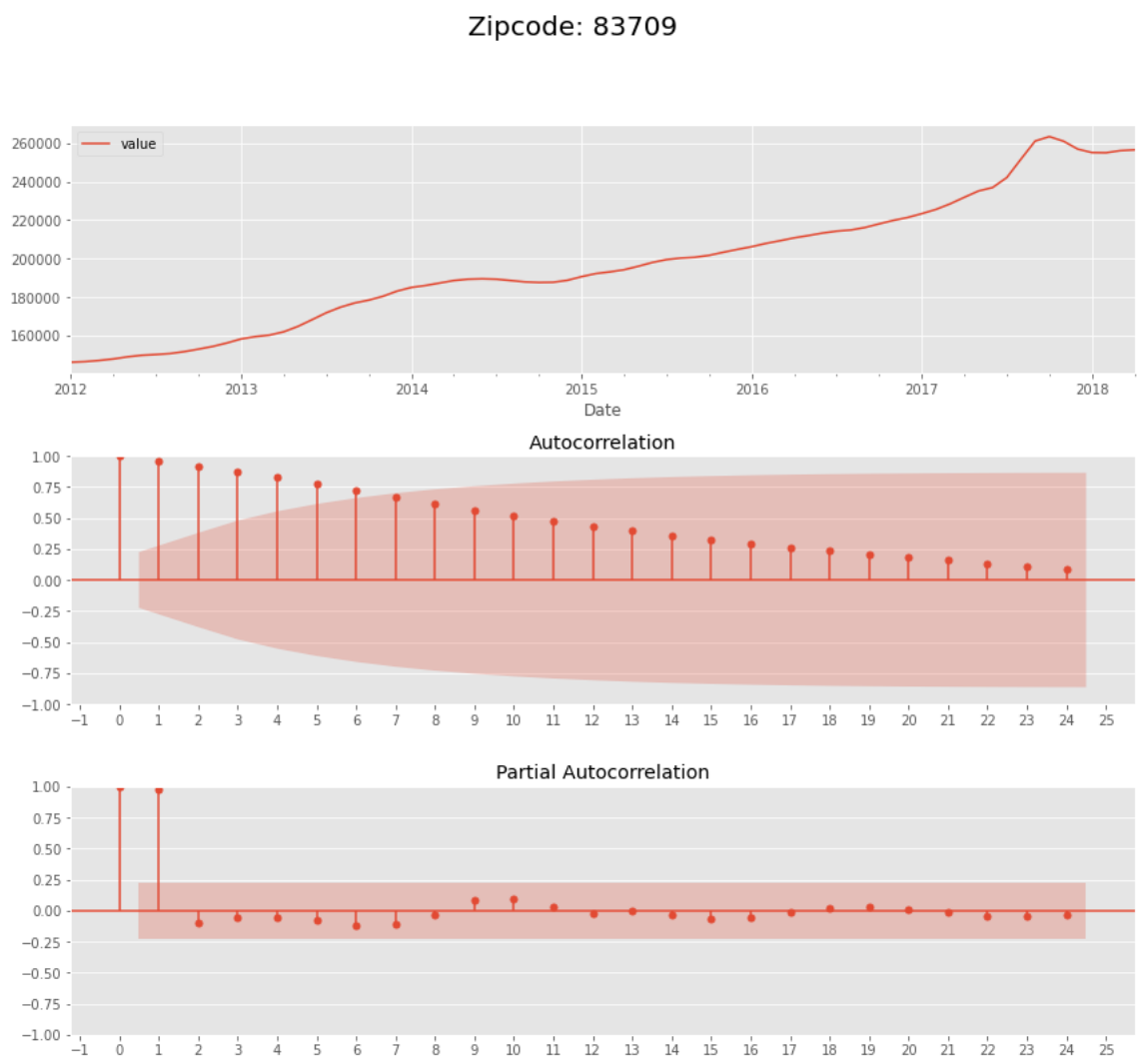


- Looking at this graph, there is a steady increase from 2012 till early 2013 with a slightly faster rate of increase from early 2013 to about mid 2014, then decreases slightly before leveling off and steadily increasing again by late 2014.
- From mid 2014 to 2017 the mean value consistently increases at a steady rate.
- 2017 shows the fastest rate of increase in home values spiking in late 2017. After the spike in late 2017 there was a steep decrease, but by 2018 the value of home began to steadily increase once again.

### ACF and PACF

- The time series trends positively.
- The ACF graph shows a slight decrease over time due to trend.
- The PACF graph shows a spike at 1 which would suggest an AR value of 1.

In [48]:
```
plot_acf_pacf(seven_zero_nine, zipcode= '83709')
plt.show()
```




### Parameter selection for the ARIMA time series model

- **Number of AR (Auto-Regressive) terms (p)**
  - p is the auto-regressive part of the model. It allows us to incorporate the effect of past values into our model.
- **Number of Differences (d)**
  - d is the Integrated component of an ARIMA model. This value is concerned with the amount of differencing as it identifies the number of lag values to subtract from the current observation.
- **Number of MA (Moving Average) terms (q)**
  - q is the moving average part of the model which is used to set the error of the model as a linear combination of the error values observed at previous time points in the past.

In [49]:
```
# Define the p, d and q parameters to take any value between 0 and 2
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets (use 12 for frequency)
pdqs = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
```

- The first step towards fitting an ARIMA model is to find the values of ARIMA(p,d,q)(P,D,Q)s that produce the desired output.

### AIC (Akaike Information Criterion) as Regularization Measure

- we are interested in finding the model that yields the lowest AIC value.

In [50]:
```
ans = find_best_SARIMAX(train, pdq, pdqs)
```

```
ARIMA (0, 0, 0) x (0, 0, 0, 12): AIC Calculated=1628.6554715891596
ARIMA (0, 0, 0) x (0, 0, 1, 12): AIC Calculated=276321.00350357394
ARIMA (0, 0, 0) x (0, 1, 0, 12): AIC Calculated=1074.7608067932592
ARIMA (0, 0, 0) x (0, 1, 1, 12): AIC Calculated=800.9896953931972
ARIMA (0, 0, 0) x (1, 0, 0, 12): AIC Calculated=1015.3058393458962
ARIMA (0, 0, 0) x (1, 0, 1, 12): AIC Calculated=1039.164538655018
ARIMA (0, 0, 0) x (1, 1, 0, 12): AIC Calculated=795.0130995893325
ARIMA (0, 0, 0) x (1, 1, 1, 12): AIC Calculated=761.2725664089535
ARIMA (0, 0, 1) x (0, 0, 0, 12): AIC Calculated=23782.178214837102
ARIMA (0, 0, 1) x (0, 0, 1, 12): AIC Calculated=254534.87908446955
ARIMA (0, 0, 1) x (0, 1, 0, 12): AIC Calculated=1027.846007475738
ARIMA (0, 0, 1) x (0, 1, 1, 12): AIC Calculated=762.0917158645034
ARIMA (0, 0, 1) x (1, 0, 0, 12): AIC Calculated=1286.4535423892971
ARIMA (0, 0, 1) x (1, 0, 1, 12): AIC Calculated=1236.1569881197602
ARIMA (0, 0, 1) x (1, 1, 0, 12): AIC Calculated=800.1584136836245
ARIMA (0, 0, 1) x (1, 1, 1, 12): AIC Calculated=758.1167459909087
ARIMA (0, 1, 0) x (0, 0, 0, 12): AIC Calculated=1037.0464920501315
ARIMA (0, 1, 0) x (0, 0, 1, 12): AIC Calculated=824.9434636123818
ARIMA (0, 1, 0) x (0, 1, 0, 12): AIC Calculated=824.8494735434424
ARIMA (0, 1, 0) x (0, 1, 1, 12): AIC Calculated=980.0806091296117
ARIMA (0, 1, 0) x (1, 0, 0, 12): AIC Calculated=833.3761968624317
ARIMA (0, 1, 0) x (1, 0, 1, 12): AIC Calculated=817.1518779709622
ARIMA (0, 1, 0) x (1, 1, 0, 12): AIC Calculated=601.3823566845822
ARIMA (0, 1, 0) x (1, 1, 1, 12): AIC Calculated=1454.053847230823
ARIMA (0, 1, 1) x (0, 0, 0, 12): AIC Calculated=951.643156553008
ARIMA (0, 1, 1) x (0, 0, 1, 12): AIC Calculated=754.9049202376855
ARIMA (0, 1, 1) x (0, 1, 0, 12): AIC Calculated=792.4092775549668
ARIMA (0, 1, 1) x (0, 1, 1, 12): AIC Calculated=2519.536553885142
ARIMA (0, 1, 1) x (1, 0, 0, 12): AIC Calculated=793.2338258328464
ARIMA (0, 1, 1) x (1, 0, 1, 12): AIC Calculated=756.8931846106722
ARIMA (0, 1, 1) x (1, 1, 0, 12): AIC Calculated=604.3402516909151
ARIMA (0, 1, 1) x (1, 1, 1, 12): AIC Calculated=2540.15874632189
ARIMA (1, 0, 0) x (0, 0, 0, 12): AIC Calculated=986.8759088969648
ARIMA (1, 0, 0) x (0, 0, 1, 12): AIC Calculated=795.2114600429827
ARIMA (1, 0, 0) x (0, 1, 0, 12): AIC Calculated=843.6819700081599
ARIMA (1, 0, 0) x (0, 1, 1, 12): AIC Calculated=629.8167010253956
ARIMA (1, 0, 0) x (1, 0, 0, 12): AIC Calculated=795.3487799278976
ARIMA (1, 0, 0) x (1, 0, 1, 12): AIC Calculated=796.1589712219247
ARIMA (1, 0, 0) x (1, 1, 0, 12): AIC Calculated=613.3112781773798
ARIMA (1, 0, 0) x (1, 1, 1, 12): AIC Calculated=615.3068869668492
ARIMA (1, 0, 1) x (0, 0, 0, 12): AIC Calculated=917.0073505404949
ARIMA (1, 0, 1) x (0, 0, 1, 12): AIC Calculated=820.685631512905
ARIMA (1, 0, 1) x (0, 1, 0, 12): AIC Calculated=819.892751204544
ARIMA (1, 0, 1) x (0, 1, 1, 12): AIC Calculated=628.5997181035417
ARIMA (1, 0, 1) x (1, 0, 0, 12): AIC Calculated=753.2116200033856
ARIMA (1, 0, 1) x (1, 0, 1, 12): AIC Calculated=738.609969458576
ARIMA (1, 0, 1) x (1, 1, 0, 12): AIC Calculated=604.5428457421394
ARIMA (1, 0, 1) x (1, 1, 1, 12): AIC Calculated=585.618715237089
ARIMA (1, 1, 0) x (0, 0, 0, 12): AIC Calculated=907.8999127516361
ARIMA (1, 1, 0) x (0, 0, 1, 12): AIC Calculated=730.6017989015269
ARIMA (1, 1, 0) x (0, 1, 0, 12): AIC Calculated=758.5822504260461
ARIMA (1, 1, 0) x (0, 1, 1, 12): AIC Calculated=1308.7426246273735
ARIMA (1, 1, 0) x (1, 0, 0, 12): AIC Calculated=730.584686960153
ARIMA (1, 1, 0) x (1, 0, 1, 12): AIC Calculated=731.2545558130004
ARIMA (1, 1, 0) x (1, 1, 0, 12): AIC Calculated=539.2190901051512
ARIMA (1, 1, 0) x (1, 1, 1, 12): AIC Calculated=1332.9068010407093
ARIMA (1, 1, 1) x (0, 0, 0, 12): AIC Calculated=885.7059466278977
ARIMA (1, 1, 1) x (0, 0, 1, 12): AIC Calculated=698.4845601618254
ARIMA (1, 1, 1) x (0, 1, 0, 12): AIC Calculated=742.4089703082338
ARIMA (1, 1, 1) x (0, 1, 1, 12): AIC Calculated=1323.27190807759
ARIMA (1, 1, 1) x (1, 0, 0, 12): AIC Calculated=712.3134523731013
ARIMA (1, 1, 1) x (1, 0, 1, 12): AIC Calculated=699.9369177292551
ARIMA (1, 1, 1) x (1, 1, 0, 12): AIC Calculated=540.2963262609073
ARIMA (1, 1, 1) x (1, 1, 1, 12): AIC Calculated=1342.9506530575902
```

In [51]:
```
# Find the parameters with minimal AIC value
ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'aic'])
ans_df.loc[ans_df['aic'].idxmin()]
```

Out[51]:
```
pdq         (1, 1, 0)
pdqs    (1, 1, 0, 12)
aic           539.219
Name: 54, dtype: object
```

The output of our code suggests that `ARIMA (1, 1, 0) x (1, 1, 0, 12)` yields the lowest AIC value of `539.219`. We should therefore consider this to be optimal option out of all the models we have considered.

### Fitting the time series model - ARIMA

In [52]:
```
output = fit_SARIMAX_model(seven_zero_nine, (1, 1, 0), (1, 1, 0, 12))
output.summary()
```

Out[52]:

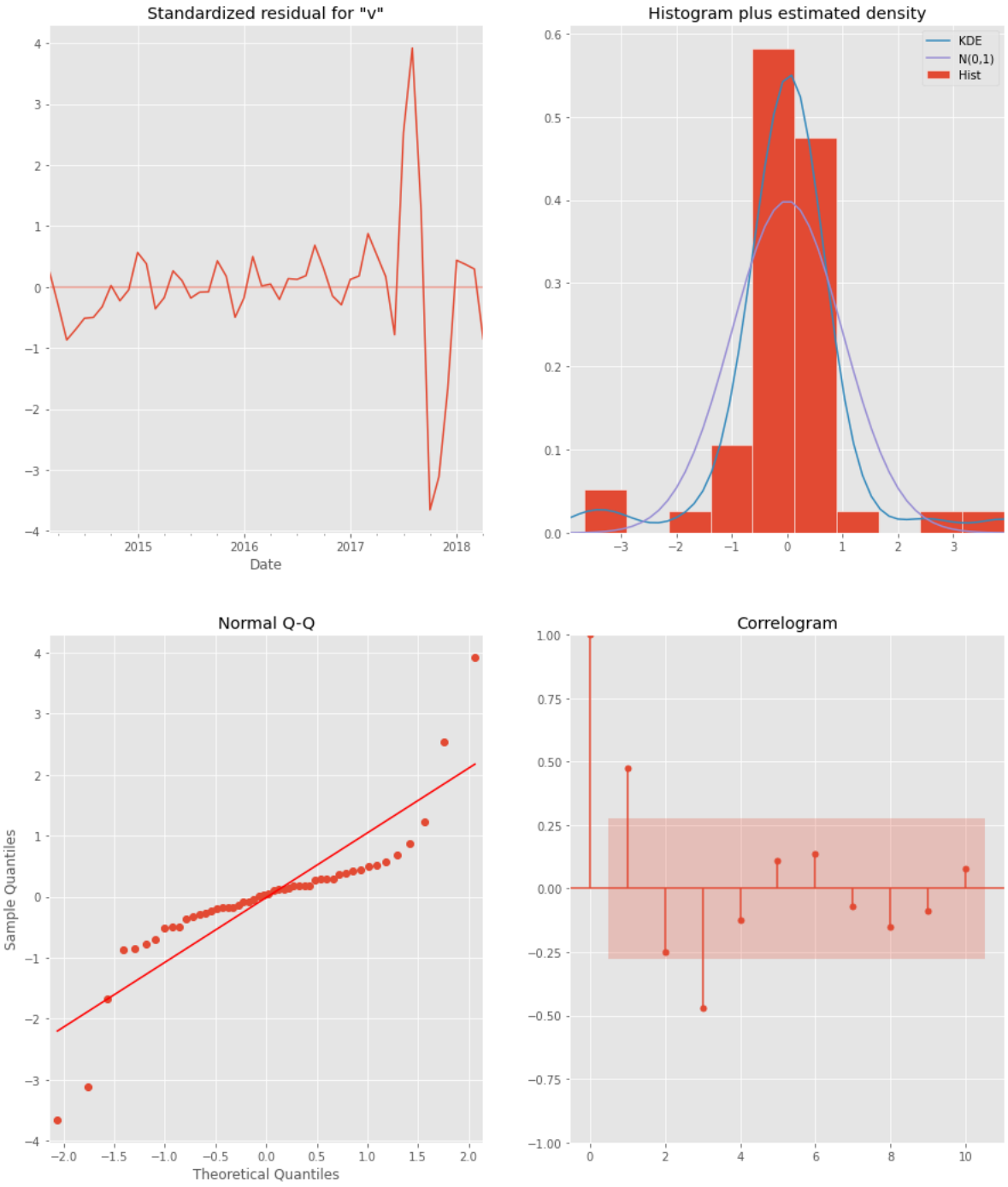
SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | value | **No. Observations:** | 76 |
| **Model:** | SARIMAX(1, 1, 0)x(1, 1, 0, 12) | **Log Likelihood** | -440.323 |
| **Date:** | Tue, 14 Feb 2023 | **AIC** | 886.646 |
| **Time:** | 12:57:28 | **BIC** | 892.382 |
| **Sample:** | 01-01-2012 | **HQIC** | 888.830 |
| | - 04-01-2018 | | |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **ar.L1** | 0.7469 | 0.047 | 15.954 | 0.000 | 0.655 | 0.839 |
| **ar.S.L12** | -0.4713 | 0.584 | -0.807 | 0.420 | -1.615 | 0.673 |
| **sigma2** | 2.302e+06 | 2.23e+05 | 10.306 | 0.000 | 1.86e+06 | 2.74e+06 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 11.85 | **Jarque-Bera (JB):** | 71.63 |
| **Prob(Q):** | 0.00 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 18.58 | **Skew:** | -0.06 |
| **Prob(H) (two-sided):** | 0.00 | **Kurtosis:** | 8.86 |

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




- **Standardized residual**: There appears to be some seasonality till about mid 2017 where there appears to be some random noise.
- **Histogram**: The histogram shows that the distribution appears to be pretty close to normal.
- **QQ-plot**: QQ-plot flares slightly away from the line.
- **Correlogram**: There are two spikes outside of the shaded area. This means there could be some missed seasonality component.

### Validating the model: One-step ahead forecasting

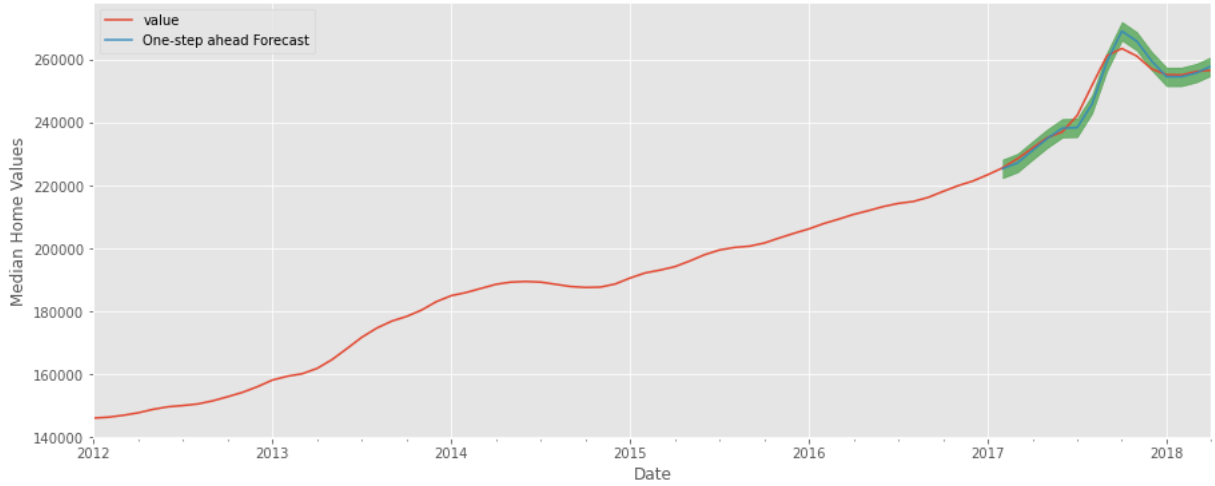
- In the cell below:
  - Get the predictions from 1st february 2017 till 2018 (end of time series)
  - Get the confidence intervals for all predictions
  - For get_predictions(), set the dynamic parameter to False to ensure that we produce one-step ahead forecasts, meaning that forecasts at each point are generated using the full history up to that point

In [53]:
```
# Get predictions starting from 01-01-2017 and calculate confidence intervals
pred = output.get_prediction(start=pd.to_datetime('2017-02-01'), dynamic=False)
pred_conf = pred.conf_int()
```

- We shall now plot the real and forecasted values of the time series to assess how well we did:

  -Plot the observed values from the dataset, starting at 2012 -Use .predicted_mean.plot() method to plot predictions -Plot the confidence intervals overlapping the predicted values

In [54]:
```
# Plot real vs predicted values along with confidence interval

rcParams['figure.figsize'] = 15, 6

# Plot observed values
ax = seven_zero_nine['2012-01-01':].plot(label='observed')

# Plot predicted values
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=0.9)

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

# Set axes labels
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')
plt.legend()

plt.show()
```




- For the most part, the forecasts align with the true values as seen above, with an overall increase trend.

### Accuracy of our forecast with Root Mean Squared Error

In [55]:
```
# Get the real and predicted values
seven_zero_nine_forecasted = pred.predicted_mean
seven_zero_nine_truth = seven_zero_nine.loc['2017-02-01':].value

# Compute the mean square error
mse = np.sqrt((seven_zero_nine_forecasted - seven_zero_nine_truth) ** 2).mean()
print('The Root Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

```
The Root Mean Squared Error of our forecasts is 2083.69
```

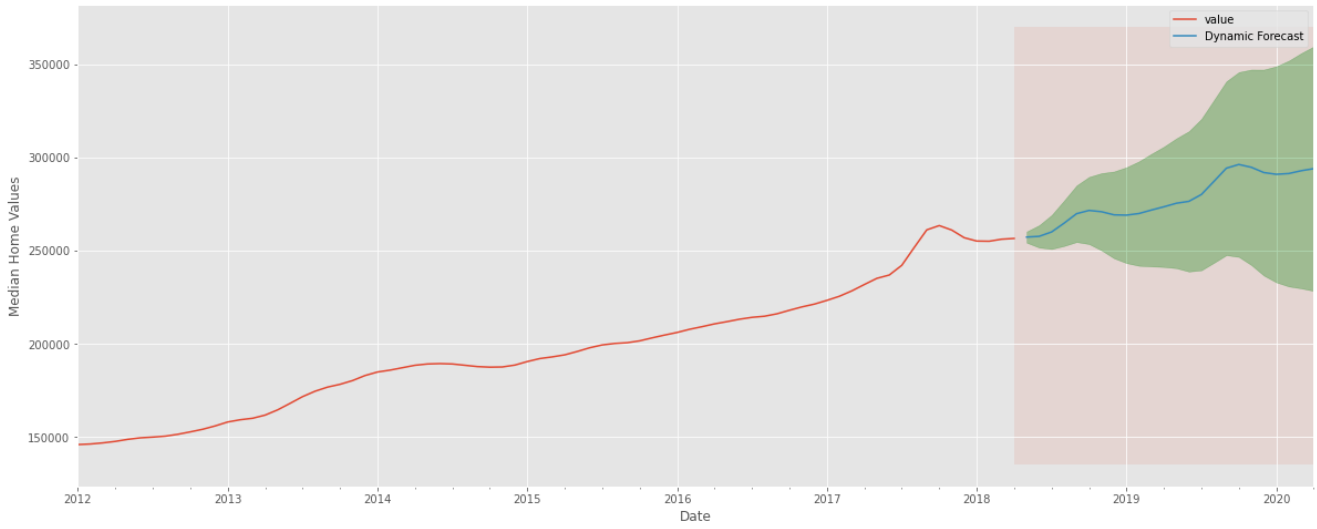
- The RMSE of 2083.69 in this case would indicate that, on average, the model's predictions for the mean value of homes in Boise, ID deviate by 2083.69 dollars from the actual values.
- This model also had an AIC of 539.219

### Dynamic Forecasting

- We can achieve a deeper insight into model's predictive power using dynamic forecasts. In this case, we only use information from the time series up to a certain point, and after that, forecasts are generated using values from previous forecasted time points.

In [56]:
```
# Get dynamic predictions with confidence intervals as above
pred_dynamic = output.get_forecast(steps=24)
pred_dynamic_conf = pred_dynamic.conf_int()
```

In [57]:
```
# Plot the dynamic forecast with confidence intervals as above
# Plot the dynamic forecast with confidence intervals.

ax = seven_zero_nine['2012':].plot(label='observed', figsize=(20, 8))

pred_dynamic.predicted_mean.plot(label='Dynamic Forecast', ax=ax)

ax.fill_between(pred_dynamic_conf.index,
                pred_dynamic_conf.iloc[:, 0],
                pred_dynamic_conf.iloc[:, 1], color='g', alpha=.3)

ax.fill_betweenx(ax.get_ylim(), pd.to_datetime('2020-04-01'), seven_zero_nine_forecasted.index[-1], alpha=.1, zorde

ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')

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




- Looking at the image above, it appears that the value of homes in boise, ID will continue to increase. With two slight spikes in value.

### Producing and visualizing forecasts

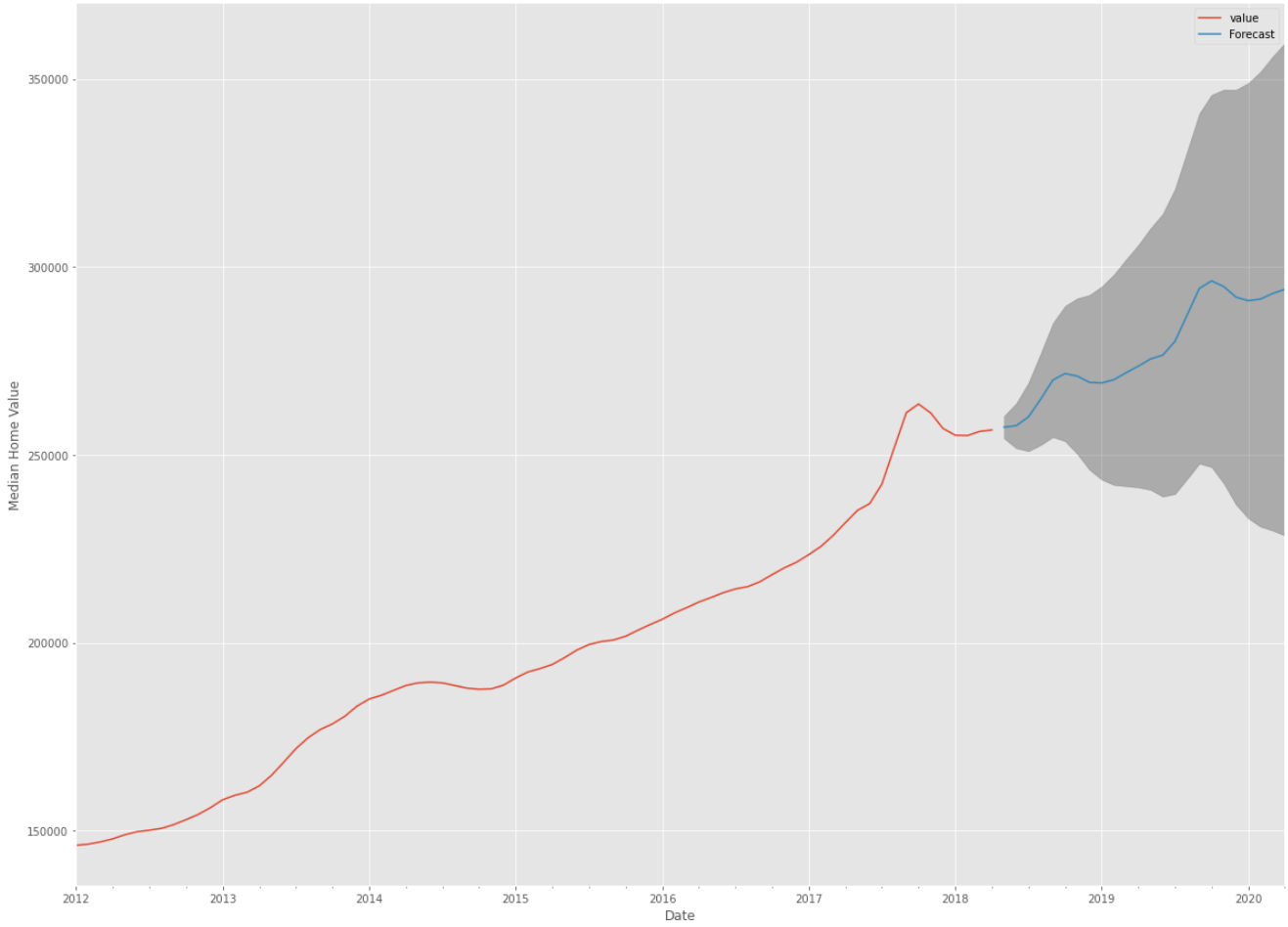
In [58]:
```
# Get forecast 24 steps ahead in future.
prediction = output.get_forecast(steps=24)

# Get confidence intervals of forecasts
pred_conf = prediction.conf_int()

pred_conf['mean'] = prediction.predicted_mean

pred_conf.head()
```

Out[58]:

| | lower value | upper value | mean |
|---|---|---|---|
| **2018-05-01** | 254364.521911 | 260312.443102 | 257338.482507 |
| **2018-06-01** | 251742.710158 | 263714.993448 | 257728.851803 |
| **2018-07-01** | 250972.636202 | 269172.886769 | 260072.761485 |
| **2018-08-01** | 252598.332322 | 276954.895535 | 264776.613929 |
| **2018-09-01** | 254716.358742 | 285024.362467 | 269870.360604 |

- The output of this code can now be used to plot the time series and forecasts of its future values.

In [59]:
```
# Plot future predictions with confidence intervals
ax = seven_zero_nine.plot(label='observed', figsize=(20, 15))
prediction.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_conf.index,
                pred_conf.iloc[:, 0],
                pred_conf.iloc[:, 1], color='k', alpha=0.25)
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Value')

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




## Average Return on Investment (ROI)

- The first predicted mean value represents the cost of the home in the beginning of the real estate investment. The ROI is calculated by subtracting the cost from the averaged predicted values, then dividing that value by the cost. This is a 2 year ROI.

In [60]:
```
#percentage over time is now the mean
#mean change percentage
cost = pred_conf.iloc[0]['mean']
roi = (pred_conf - cost)/ abs(cost) * 100

#pred_conf['result'] = pred_conf[''] - pred_conf['']

roi
```

Out[60]:

| | lower value | upper value | mean |
|---|---|---|---|
| **2018-05-01** | -1.155661 | 1.155661 | 0.000000 |
| **2018-06-01** | -2.174479 | 2.477869 | 0.151695 |
| **2018-07-01** | -2.473725 | 4.598770 | 1.062522 |
| **2018-08-01** | -1.841990 | 7.622806 | 2.890408 |
| **2018-09-01** | -1.018940 | 10.758546 | 4.869803 |
| **2018-10-01** | -1.445465 | 12.542187 | 5.548361 |
| **2018-11-01** | -2.765034 | 13.322710 | 5.278838 |
| **2018-12-01** | -4.407287 | 13.671126 | 4.631920 |
| **2019-01-01** | -5.413099 | 14.551467 | 4.569184 |
| **2019-02-01** | -5.969221 | 15.783902 | 4.907341 |
| **2019-03-01** | -6.094453 | 17.357308 | 5.631427 |
| **2019-04-01** | -6.222499 | 18.845698 | 6.311600 |
| **2019-05-01** | -6.472887 | 20.571375 | 7.049244 |
| **2019-06-01** | -7.173841 | 22.064507 | 7.445333 |
| **2019-07-01** | -6.902131 | 24.639964 | 8.868916 |
| **2019-08-01** | -5.354983 | 28.526466 | 11.585742 |
| **2019-09-01** | -3.756620 | 32.452608 | 14.347994 |
| **2019-10-01** | -4.124835 | 34.372218 | 15.123691 |
| **2019-11-01** | -5.826009 | 34.902983 | 14.538487 |
| **2019-12-01** | -8.005313 | 34.891775 | 13.443231 |
| **2020-01-01** | -9.420533 | 35.577838 | 13.078653 |
| **2020-02-01** | -10.278644 | 36.754262 | 13.237809 |
| **2020-03-01** | -10.680160 | 38.322385 | 13.821113 |
| **2020-04-01** | -11.201801 | 39.708322 | 14.253260 |

## Results:

In [61]:
```
data = {'Zipcode':  ['83709', '83704', '83706', '83705', '83702', '83713', '83703', '83716', '83712']}

results = pd.DataFrame(data, columns = ['Zipcode'])
results['Two_YR_ROI_Percentage'] = 1
results.head()
```

Out[61]:

| | Zipcode | Two_YR_ROI_Percentage |
|---|---|---|
| **0** | 83709 | 1 |
| **1** | 83704 | 1 |
| **2** | 83706 | 1 |
| **3** | 83705 | 1 |
| **4** | 83702 | 1 |

In [62]:
```
results.Two_YR_ROI_Percentage[0] = roi['mean'][-1]
results.head()
```

Out[62]:

| | Zipcode | Two_YR_ROI_Percentage |
|---|---|---|
| **0** | 83709 | 14 |
| **1** | 83704 | 1 |
| **2** | 83706 | 1 |
| **3** | 83705 | 1 |
| **4** | 83702 | 1 |

- For area code 83709, the forecasted 2 year ROI is 14%.

## 83704

In [63]:
```
seven_zero_four = boise_grouped.loc[83704]
```

In [64]:
```
# find the index which allows us to split off 20% of the data
cutoff = round(seven_zero_four.shape[0]*0.8)
cutoff
```

Out[64]: 61

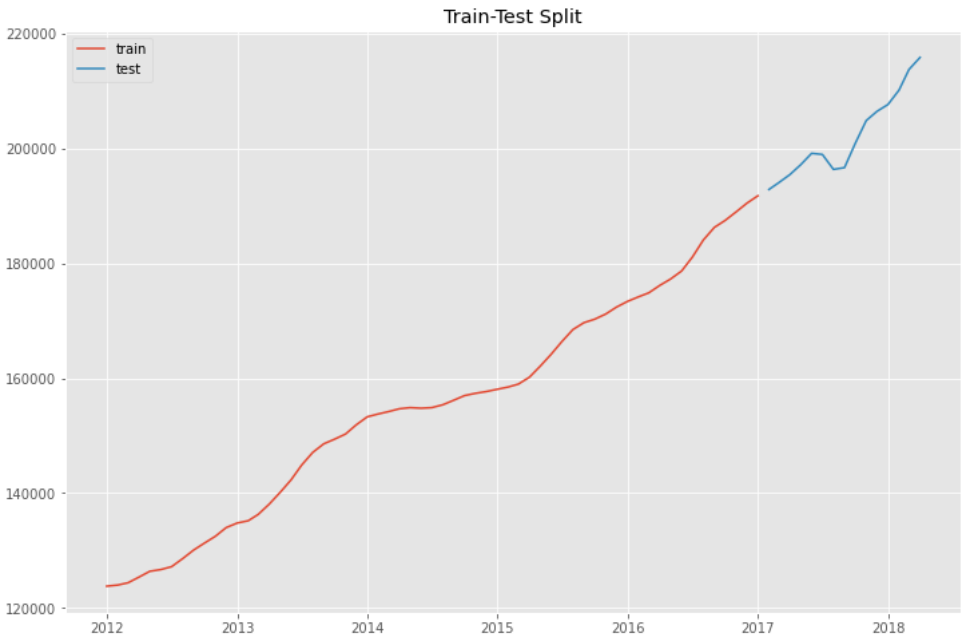
In [65]:
```
# Test Data
seven_zero_four[61:]
```

Out[65]:

| Date | value |
|---|---|
| **2017-02-01** | 192900.0 |
| **2017-03-01** | 194100.0 |
| **2017-04-01** | 195500.0 |
| **2017-05-01** | 197200.0 |
| **2017-06-01** | 199200.0 |
| **2017-07-01** | 199000.0 |
| **2017-08-01** | 196400.0 |
| **2017-09-01** | 196700.0 |
| **2017-10-01** | 200900.0 |
| **2017-11-01** | 204900.0 |
| **2017-12-01** | 206500.0 |
| **2018-01-01** | 207700.0 |
| **2018-02-01** | 210200.0 |
| **2018-03-01** | 213800.0 |
| **2018-04-01** | 215900.0 |

In [66]:
```
train = seven_zero_four[:cutoff]

test = seven_zero_four[cutoff:]

fig, ax = plt.subplots(figsize=(12, 8))
ax.plot(train, label='train')
ax.plot(test, label='test')
ax.set_title('Train-Test Split');
plt.legend()
```

Out[66]: <matplotlib.legend.Legend at 0x7faf4d129610>




In [67]:
```
seven_zero_four = boise_grouped.loc[83704]
seven_zero_four.tail()
```

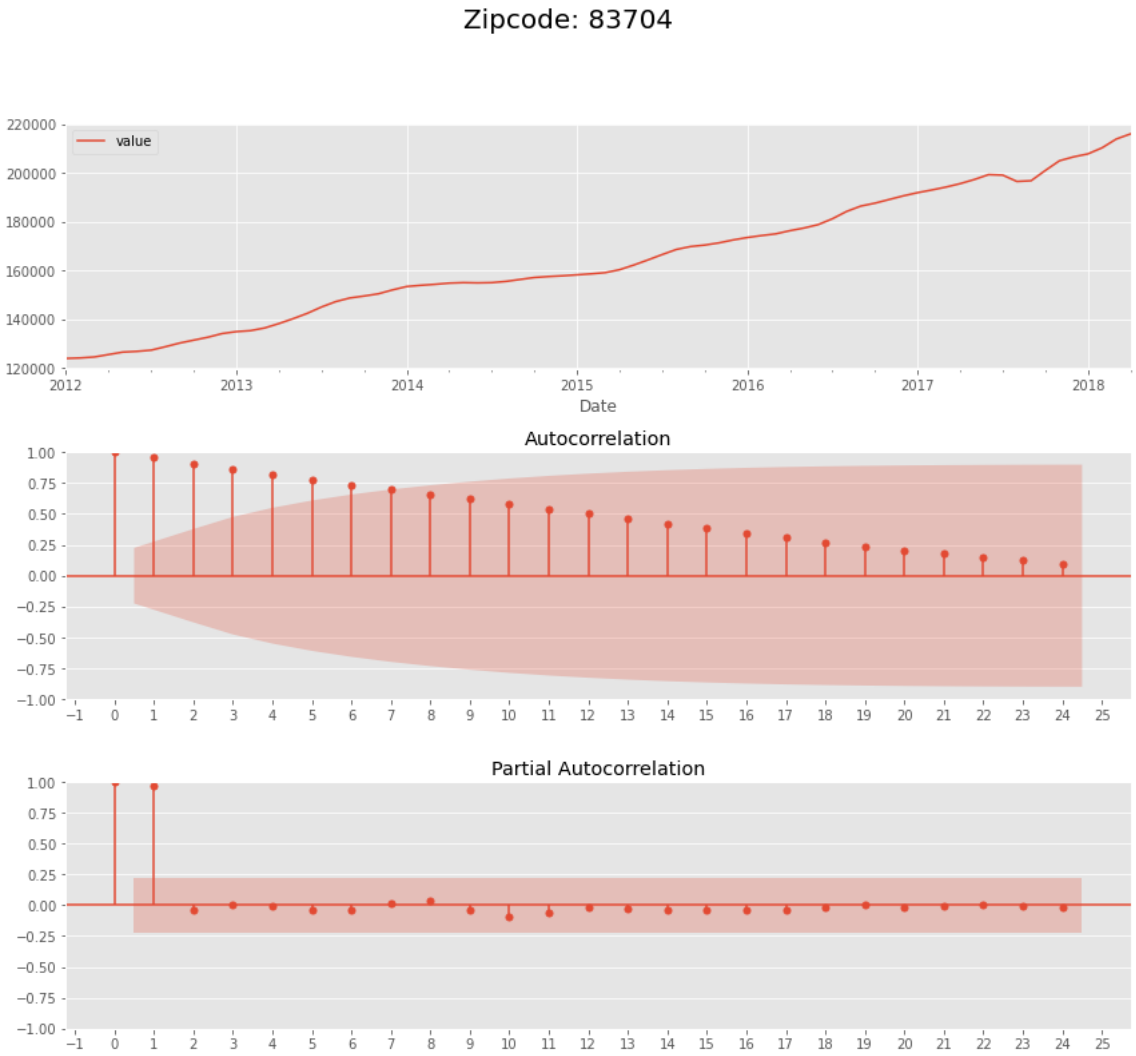
Out[67]:

| Date | value |
|---|---|
| **2017-12-01** | 206500.0 |
| **2018-01-01** | 207700.0 |
| **2018-02-01** | 210200.0 |
| **2018-03-01** | 213800.0 |
| **2018-04-01** | 215900.0 |

## ACF & PACF

- The time series trends positively. In about mid 2017 there is short decrease in value for a couple of months, then begins to even out before continuing to increase positively.
- The ACF graph shows a slight decrease over time due to trend.
- The PACF graph shows a spike at 1 which would suggest an AR value of 1.

In [68]:
```
plot_acf_pacf(seven_zero_four, zipcode= '83704')
plt.show()
```




### Parameter selection for the ARIMA time series model

In [69]:
```
# Define the p, d and q parameters to take any value between 0 and 2
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets (use 12 for frequency)
pdqs = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
```

In [70]:
```
ans = find_best_SARIMAX(train, pdq, pdqs)
```

```
ARIMA (0, 0, 0) x (0, 0, 0, 12): AIC Calculated=1607.8068912305614
ARIMA (0, 0, 0) x (0, 0, 1, 12): AIC Calculated=201216.94563344726
ARIMA (0, 0, 0) x (0, 1, 0, 12): AIC Calculated=1055.7336468410306
ARIMA (0, 0, 0) x (0, 1, 1, 12): AIC Calculated=790.286002275818
ARIMA (0, 0, 0) x (1, 0, 0, 12): AIC Calculated=969.3897307762956
ARIMA (0, 0, 0) x (1, 0, 1, 12): AIC Calculated=942.8285520625384
ARIMA (0, 0, 0) x (1, 1, 0, 12): AIC Calculated=772.0238291440097
ARIMA (0, 0, 0) x (1, 1, 1, 12): AIC Calculated=748.657893502281
ARIMA (0, 0, 1) x (0, 0, 0, 12): AIC Calculated=1541.674790879383
ARIMA (0, 0, 1) x (0, 0, 1, 12): AIC Calculated=184194.45359179587
ARIMA (0, 0, 1) x (0, 1, 0, 12): AIC Calculated=1044.306385389932
ARIMA (0, 0, 1) x (0, 1, 1, 12): AIC Calculated=731.6803447451764
ARIMA (0, 0, 1) x (1, 0, 0, 12): AIC Calculated=1269.3783030610316
ARIMA (0, 0, 1) x (1, 0, 1, 12): AIC Calculated=1288.9348190471153
ARIMA (0, 0, 1) x (1, 1, 0, 12): AIC Calculated=791.8922385598272
ARIMA (0, 0, 1) x (1, 1, 1, 12): AIC Calculated=747.0763479134038
ARIMA (0, 1, 0) x (0, 0, 0, 12): AIC Calculated=1019.0066713579278
ARIMA (0, 1, 0) x (0, 0, 1, 12): AIC Calculated=819.3651888637335
ARIMA (0, 1, 0) x (0, 1, 0, 12): AIC Calculated=792.6276053591022
ARIMA (0, 1, 0) x (0, 1, 1, 12): AIC Calculated=1421.7106079256346
ARIMA (0, 1, 0) x (1, 0, 0, 12): AIC Calculated=806.6965694595962
ARIMA (0, 1, 0) x (1, 0, 1, 12): AIC Calculated=791.6996543011568
ARIMA (0, 1, 0) x (1, 1, 0, 12): AIC Calculated=584.9921368883723
ARIMA (0, 1, 0) x (1, 1, 1, 12): AIC Calculated=1153.7711857583968
ARIMA (0, 1, 1) x (0, 0, 0, 12): AIC Calculated=933.8450036496301
ARIMA (0, 1, 1) x (0, 0, 1, 12): AIC Calculated=749.5072212307059
ARIMA (0, 1, 1) x (0, 1, 0, 12): AIC Calculated=760.0618511280023
ARIMA (0, 1, 1) x (0, 1, 1, 12): AIC Calculated=2594.3018082818576
ARIMA (0, 1, 1) x (1, 0, 0, 12): AIC Calculated=778.4480381577587
ARIMA (0, 1, 1) x (1, 0, 1, 12): AIC Calculated=725.2649202362264
ARIMA (0, 1, 1) x (1, 1, 0, 12): AIC Calculated=579.1988981592182
ARIMA (0, 1, 1) x (1, 1, 1, 12): AIC Calculated=2359.490256894773
ARIMA (1, 0, 0) x (0, 0, 0, 12): AIC Calculated=956.2236228823836
ARIMA (1, 0, 0) x (0, 0, 1, 12): AIC Calculated=801.4232122698758
ARIMA (1, 0, 0) x (0, 1, 0, 12): AIC Calculated=809.8635795721527
ARIMA (1, 0, 0) x (0, 1, 1, 12): AIC Calculated=609.9082376646966
ARIMA (1, 0, 0) x (1, 0, 0, 12): AIC Calculated=773.2075810681451
ARIMA (1, 0, 0) x (1, 0, 1, 12): AIC Calculated=794.2295909267458
ARIMA (1, 0, 0) x (1, 1, 0, 12): AIC Calculated=592.9616433015347
ARIMA (1, 0, 0) x (1, 1, 1, 12): AIC Calculated=608.5349642112011
ARIMA (1, 0, 1) x (0, 0, 0, 12): AIC Calculated=887.6297218041024
ARIMA (1, 0, 1) x (0, 0, 1, 12): AIC Calculated=822.2261172394884
ARIMA (1, 0, 1) x (0, 1, 0, 12): AIC Calculated=785.3389871953822
ARIMA (1, 0, 1) x (0, 1, 1, 12): AIC Calculated=556.3606532786101
ARIMA (1, 0, 1) x (1, 0, 0, 12): AIC Calculated=729.0072939786273
ARIMA (1, 0, 1) x (1, 0, 1, 12): AIC Calculated=716.1906504875227
ARIMA (1, 0, 1) x (1, 1, 0, 12): AIC Calculated=578.1409450371561
ARIMA (1, 0, 1) x (1, 1, 1, 12): AIC Calculated=556.2709192540602
ARIMA (1, 1, 0) x (0, 0, 0, 12): AIC Calculated=896.1936602477444
ARIMA (1, 1, 0) x (0, 0, 1, 12): AIC Calculated=717.6188669733247
ARIMA (1, 1, 0) x (0, 1, 0, 12): AIC Calculated=734.4026241400607
ARIMA (1, 1, 0) x (0, 1, 1, 12): AIC Calculated=1388.571316999189
ARIMA (1, 1, 0) x (1, 0, 0, 12): AIC Calculated=715.6985562684744
ARIMA (1, 1, 0) x (1, 0, 1, 12): AIC Calculated=714.6037624362529
ARIMA (1, 1, 0) x (1, 1, 0, 12): AIC Calculated=525.5287207492049
ARIMA (1, 1, 0) x (1, 1, 1, 12): AIC Calculated=1326.5135275117366
ARIMA (1, 1, 1) x (0, 0, 0, 12): AIC Calculated=869.094225256748
ARIMA (1, 1, 1) x (0, 0, 1, 12): AIC Calculated=684.256541573081
ARIMA (1, 1, 1) x (0, 1, 0, 12): AIC Calculated=714.3446344875362
ARIMA (1, 1, 1) x (0, 1, 1, 12): AIC Calculated=1327.842530076638
ARIMA (1, 1, 1) x (1, 0, 0, 12): AIC Calculated=698.4167053796357
ARIMA (1, 1, 1) x (1, 0, 1, 12): AIC Calculated=684.327358320981
ARIMA (1, 1, 1) x (1, 1, 0, 12): AIC Calculated=526.7972383746151
ARIMA (1, 1, 1) x (1, 1, 1, 12): AIC Calculated=1122.090761895089
```

In [71]:
```
# Find the parameters with minimal AIC value
ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'aic'])
ans_df.loc[ans_df['aic'].idxmin()]
```

Out[71]:
```
pdq         (1, 1, 0)
pdqs    (1, 1, 0, 12)
aic           525.529
Name: 54, dtype: object
```

The output of our code suggests that `ARIMA (1, 1, 0) x (1, 1, 0, 12)` yields the lowest AIC value of `525.529`. We should therefore consider this to be optimal option out of all the models we have considered.

In [72]:
```
output = fit_SARIMAX_model(seven_zero_four, (1, 1, 0), (1, 1, 0, 12))
output.summary()
```

Out[72]:

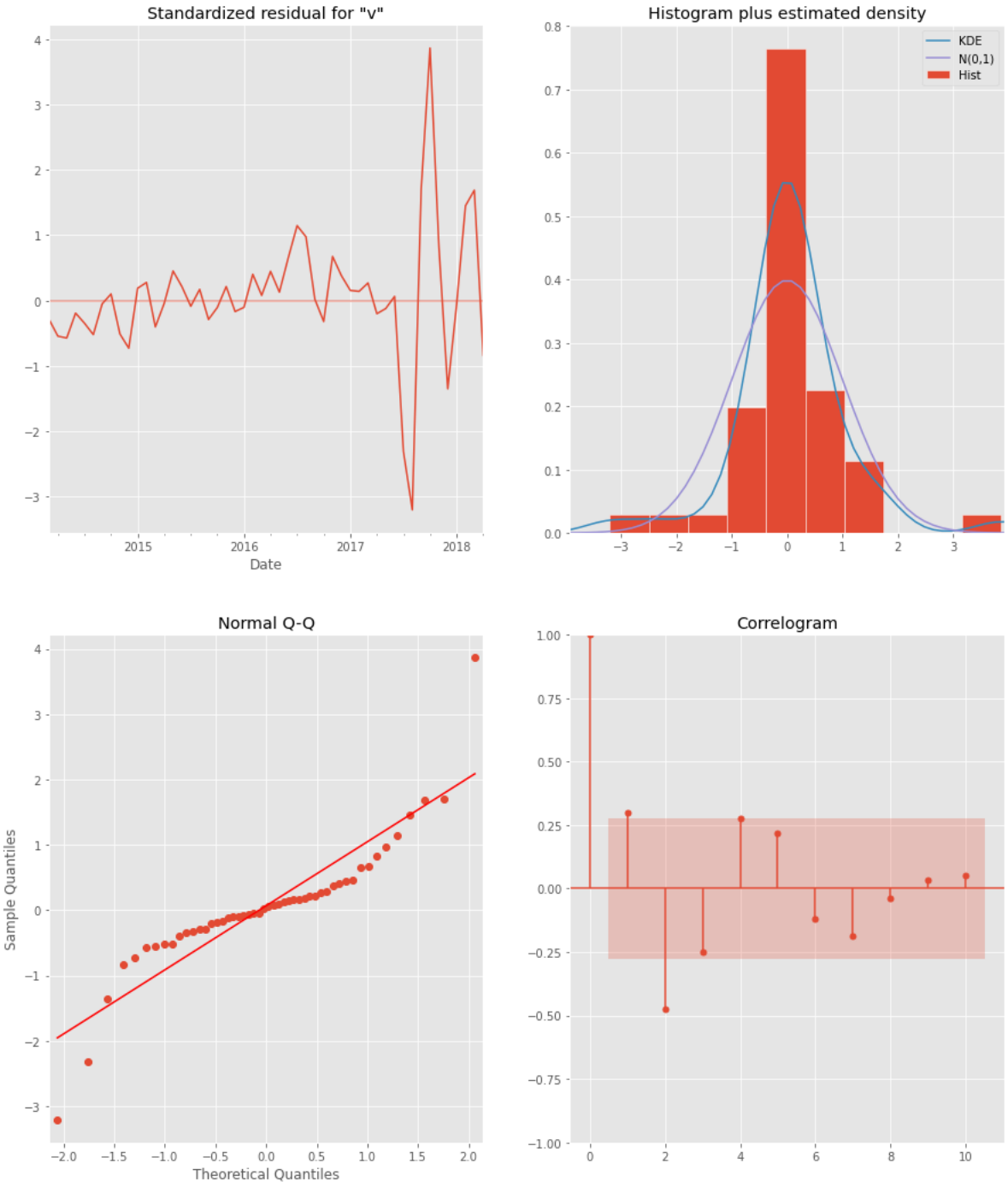
SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | value | **No. Observations:** | 76 |
| **Model:** | SARIMAX(1, 1, 0)x(1, 1, 0, 12) | **Log Likelihood** | -418.896 |
| **Date:** | Tue, 14 Feb 2023 | **AIC** | 843.792 |
| **Time:** | 12:57:39 | **BIC** | 849.528 |
| **Sample:** | 01-01-2012 | **HQIC** | 845.976 |
| | - 04-01-2018 | | |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **ar.L1** | 0.6002 | 0.090 | 6.637 | 0.000 | 0.423 | 0.778 |
| **ar.S.L12** | -0.7801 | 0.398 | -1.961 | 0.050 | -1.560 | -0.001 |
| **sigma2** | 1.147e+06 | 1.42e+05 | 8.072 | 0.000 | 8.68e+05 | 1.43e+06 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 4.73 | **Jarque-Bera (JB):** | 58.37 |
| **Prob(Q):** | 0.03 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 16.92 | **Skew:** | 0.33 |
| **Prob(H) (two-sided):** | 0.00 | **Kurtosis:** | 8.25 |

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



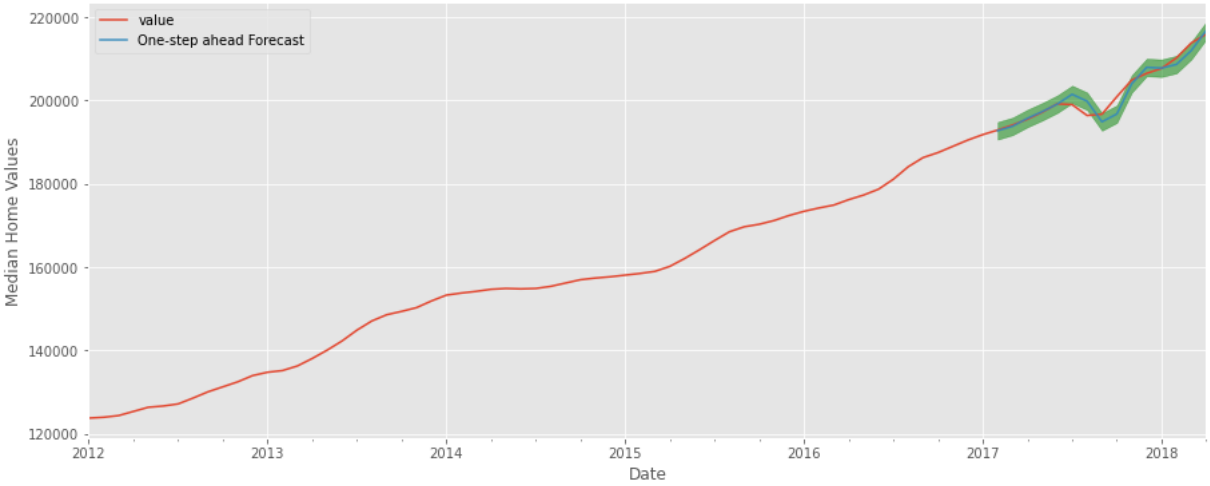

- **Standardized residual**: There appears to be some seasonality till about mid 2017 where there appears to be some random noise.
- **Histogram**: The histogram shows that the distribution appears to be pretty close to normal.
- **QQ-plot**: QQ-plot flares slightly away from the line.
- **Correlogram**: There are two spikes outside of the shaded area. This means there could be some missed seasonality component.

- For our time series model, we see that each weight has a **p-value** lower or close to 0.05, so its reasonable to retain all of them in our model.

### Validating the model: One-step ahead forecasting

In [73]:
```
# Get predictions starting from 01-01-2017 and calculate confidence intervals
pred = output.get_prediction(start=pd.to_datetime('2017-02-01'), dynamic=False)
pred_conf = pred.conf_int()
```

In [74]:
```
# Plot real vs predicted values along with confidence interval

rcParams['figure.figsize'] = 15, 6

# Plot observed values
ax = seven_zero_four['2012-01-01':].plot(label='observed')

# Plot predicted values
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=0.9)

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

# Set axes labels
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')
plt.legend()

plt.show()
```




- The forecasts align with the true values as seen above, with an overall increase trend.

### Accuracy of our forecast with Root Mean Squared Error

In [75]:
```
# Get the real and predicted values
seven_zero_four_forecasted = pred.predicted_mean
seven_zero_four_truth = seven_zero_four.loc['2017-02-01':].value

# Compute the mean square error
mse = np.sqrt((seven_zero_four_forecasted - seven_zero_four_truth) ** 2).mean()
print('The Root Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

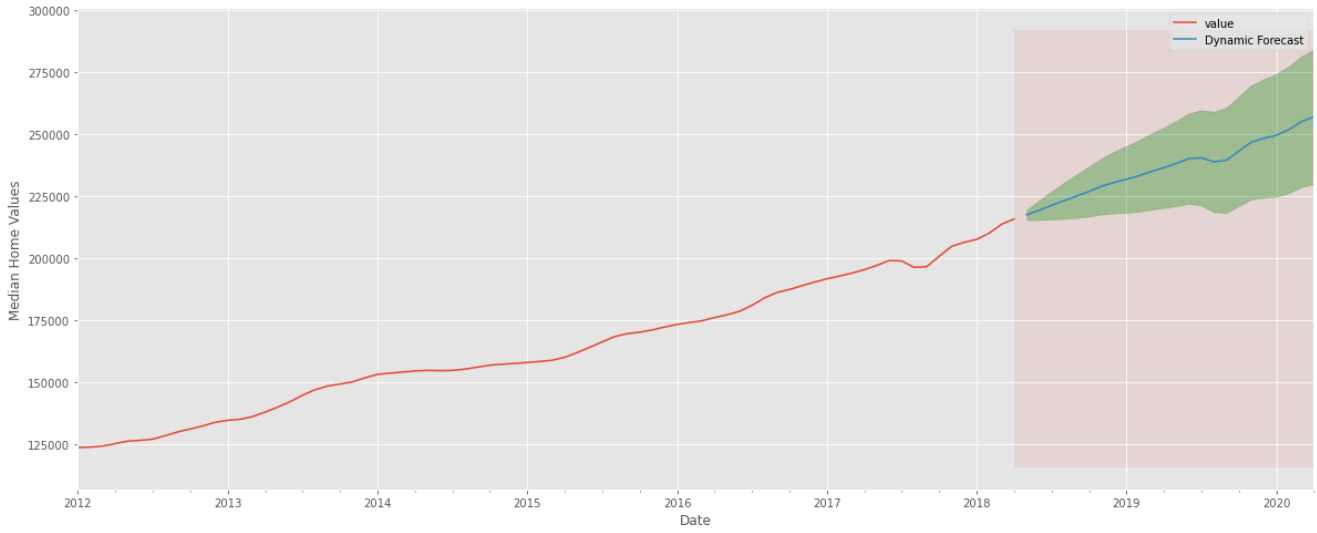
```
The Root Mean Squared Error of our forecasts is 1292.15
```

- The RMSE of 1292.15 in this case would indicate that, on average, the model's predictions for the mean value of homes in Boise, ID deviate by 1292.15 dollars from the actual values.
- AIC = 525.529

### Dynamic Forecasting

In [76]:
```
# Get dynamic predictions with confidence intervals as above
pred_dynamic = output.get_forecast(steps=24)
pred_dynamic_conf = pred_dynamic.conf_int()
```

In [77]:
```
# Plot the dynamic forecast with confidence intervals as above
# Plot the dynamic forecast with confidence intervals.

ax = seven_zero_four['2012':].plot(label='observed', figsize=(20, 8))

pred_dynamic.predicted_mean.plot(label='Dynamic Forecast', ax=ax)

ax.fill_between(pred_dynamic_conf.index,
                pred_dynamic_conf.iloc[:, 0],
                pred_dynamic_conf.iloc[:, 1], color='g', alpha=.3)

ax.fill_betweenx(ax.get_ylim(), pd.to_datetime('2020-04-01'), seven_zero_four_forecasted.index[-1], alpha=.1, zorde

ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')

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




- The dynamic forecast shows a positive trend into the future.

### Producing and visualizing forecasts

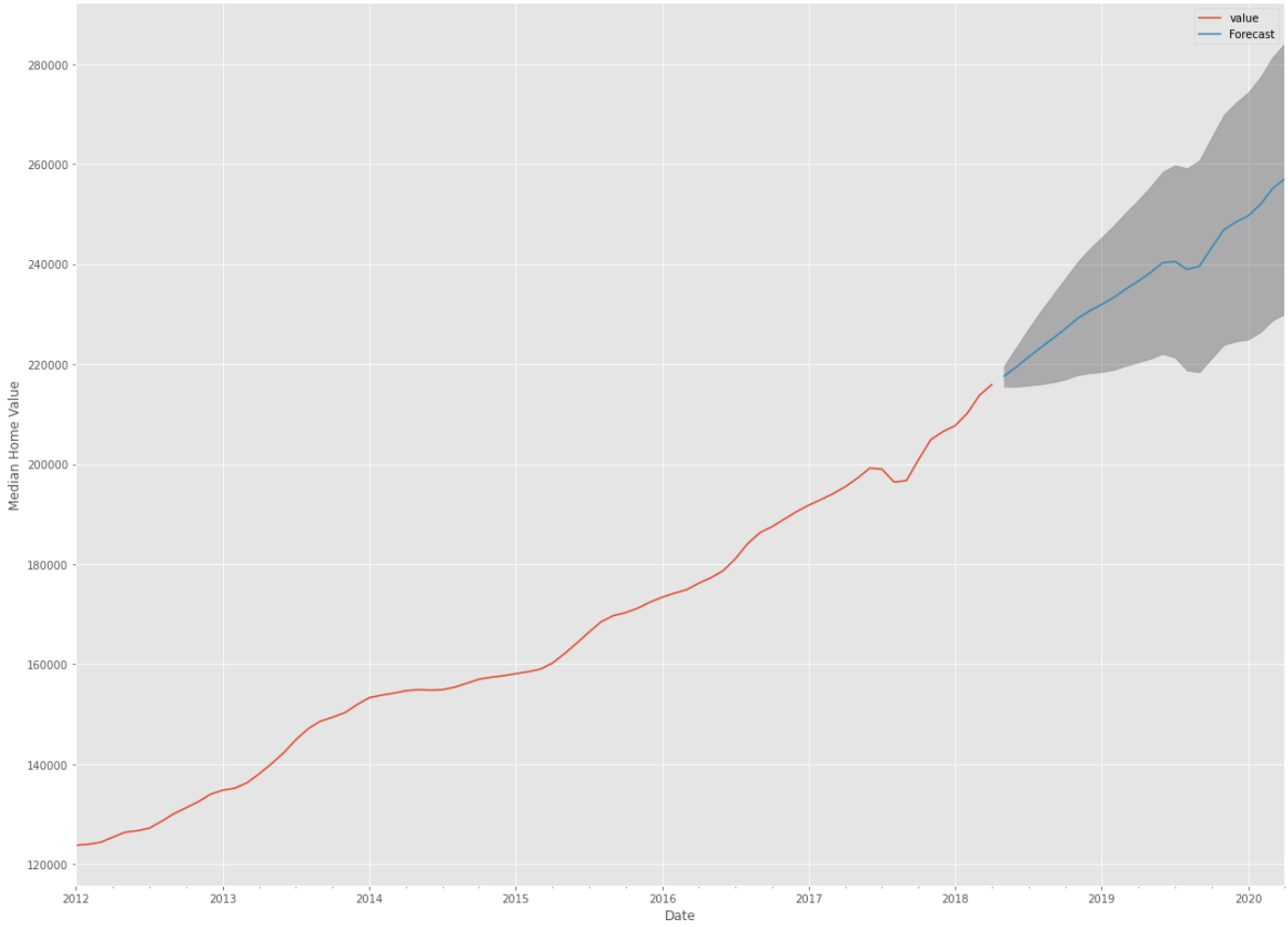
In [78]:
```
# Get forecast 24 steps ahead in future.
prediction = output.get_forecast(steps=24)

# Get confidence intervals of forecasts
pred_conf = prediction.conf_int()

pred_conf['mean'] = prediction.predicted_mean

pred_conf.head()
```

Out[78]:

| | lower value | upper value | mean |
|---|---|---|---|
| **2018-05-01** | 215499.946345 | 219697.973857 | 217598.960101 |
| **2018-06-01** | 215450.392427 | 223372.081200 | 219411.236813 |
| **2018-07-01** | 215695.969161 | 227119.293309 | 221407.631235 |
| **2018-08-01** | 215962.556031 | 230591.298888 | 223276.927460 |
| **2018-09-01** | 216348.040131 | 233891.261392 | 225119.650762 |

- The output of this code can now be used to plot the time series and forecasts of its future values.

In [79]:
```
# Plot future predictions with confidence intervals
ax = seven_zero_four.plot(label='observed', figsize=(20, 15))
prediction.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_conf.index,
                pred_conf.iloc[:, 0],
                pred_conf.iloc[:, 1], color='k', alpha=0.25)
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Value')

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




- Our forecasts show that the time series is expected to continue increasing at a steady pace.

## Average Return on Investment (ROI)

In [80]:
```
#percentage over time is now the mean
#mean change percentage
cost = pred_conf.iloc[0]['mean']
roi = (pred_conf - cost)/ abs(cost) * 100

#pred_conf['result'] = pred_conf[''] - pred_conf['']

roi
```

Out[80]:

| | lower value | upper value | mean |
|---|---|---|---|
| **2018-05-01** | -0.964625 | 0.964625 | 0.000000 |
| **2018-06-01** | -0.987398 | 2.653101 | 0.832852 |
| **2018-07-01** | -0.874540 | 4.375174 | 1.750317 |
| **2018-08-01** | -0.752028 | 5.970772 | 2.609372 |
| **2018-09-01** | -0.574874 | 7.487307 | 3.456216 |
| **2018-10-01** | -0.313526 | 8.968820 | 4.327647 |
| **2018-11-01** | 0.079639 | 10.479806 | 5.279722 |
| **2018-12-01** | 0.269516 | 11.700877 | 5.985197 |
| **2019-01-01** | 0.381417 | 12.770852 | 6.576134 |
| **2019-02-01** | 0.582535 | 13.868129 | 7.225332 |
| **2019-03-01** | 0.956201 | 15.085193 | 8.020697 |
| **2019-04-01** | 1.272083 | 16.199156 | 8.735619 |
| **2019-05-01** | 1.606523 | 17.427952 | 9.517238 |
| **2019-06-01** | 2.042379 | 18.792955 | 10.417667 |
| **2019-07-01** | 1.708126 | 19.387738 | 10.547932 |
| **2019-08-01** | 0.509597 | 19.100252 | 9.804925 |
| **2019-09-01** | 0.360992 | 19.836589 | 10.098790 |
| **2019-10-01** | 1.630320 | 21.961922 | 11.796121 |
| **2019-11-01** | 2.860210 | 24.018745 | 13.439478 |
| **2019-12-01** | 3.189451 | 25.147007 | 14.168229 |
| **2020-01-01** | 3.363205 | 26.093576 | 14.728390 |
| **2020-02-01** | 4.027966 | 27.506812 | 15.767389 |
| **2020-03-01** | 5.130457 | 29.335276 | 17.232866 |
| **2020-04-01** | 5.687920 | 30.597932 | 18.142926 |

## Results:

In [81]:
```
results.Two_YR_ROI_Percentage[1] = roi['mean'][-1]
results.head()
```

Out[81]:

| | Zipcode | Two_YR_ROI_Percentage |
|---|---|---|
| **0** | 83709 | 14 |
| **1** | 83704 | 18 |
| **2** | 83706 | 1 |
| **3** | 83705 | 1 |
| **4** | 83702 | 1 |

## 83706

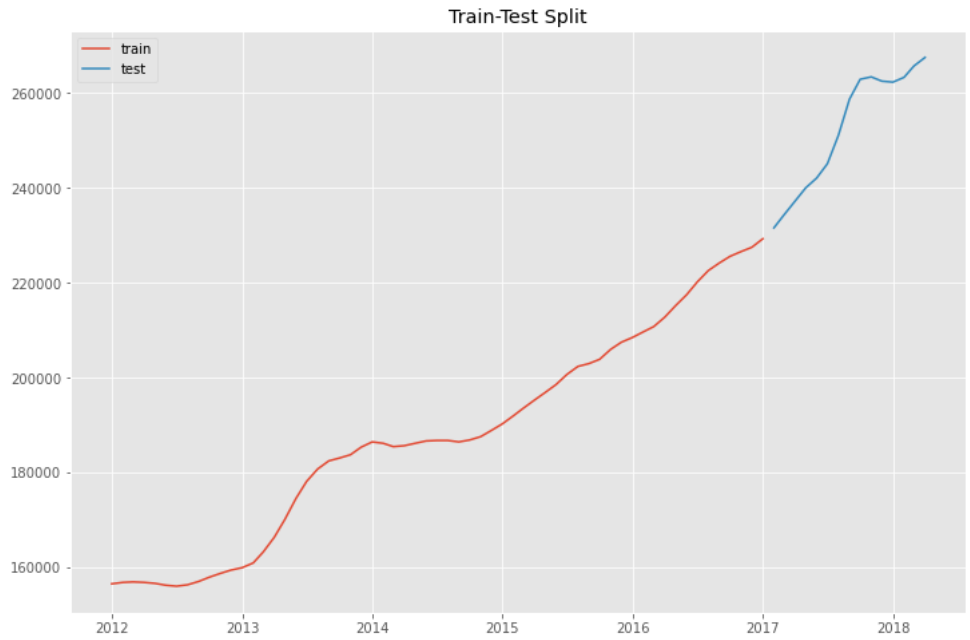
In [82]:
```
seven_zero_six = boise_grouped.loc[83706]
seven_zero_six.tail()
```

Out[82]:

| Date | value |
|---|---|
| 2017-12-01 | 262500.0 |
| 2018-01-01 | 262300.0 |
| 2018-02-01 | 263300.0 |
| 2018-03-01 | 265700.0 |
| 2018-04-01 | 267500.0 |

In [83]:
```
# find the index which allows us to split off 20% of the data
cutoff = round(seven_zero_six.shape[0]*0.8)


train = seven_zero_six[:cutoff]

test = seven_zero_six[cutoff:]

fig, ax = plt.subplots(figsize=(12, 8))
ax.plot(train, label='train')
ax.plot(test, label='test')
ax.set_title('Train-Test Split');
plt.legend()
```

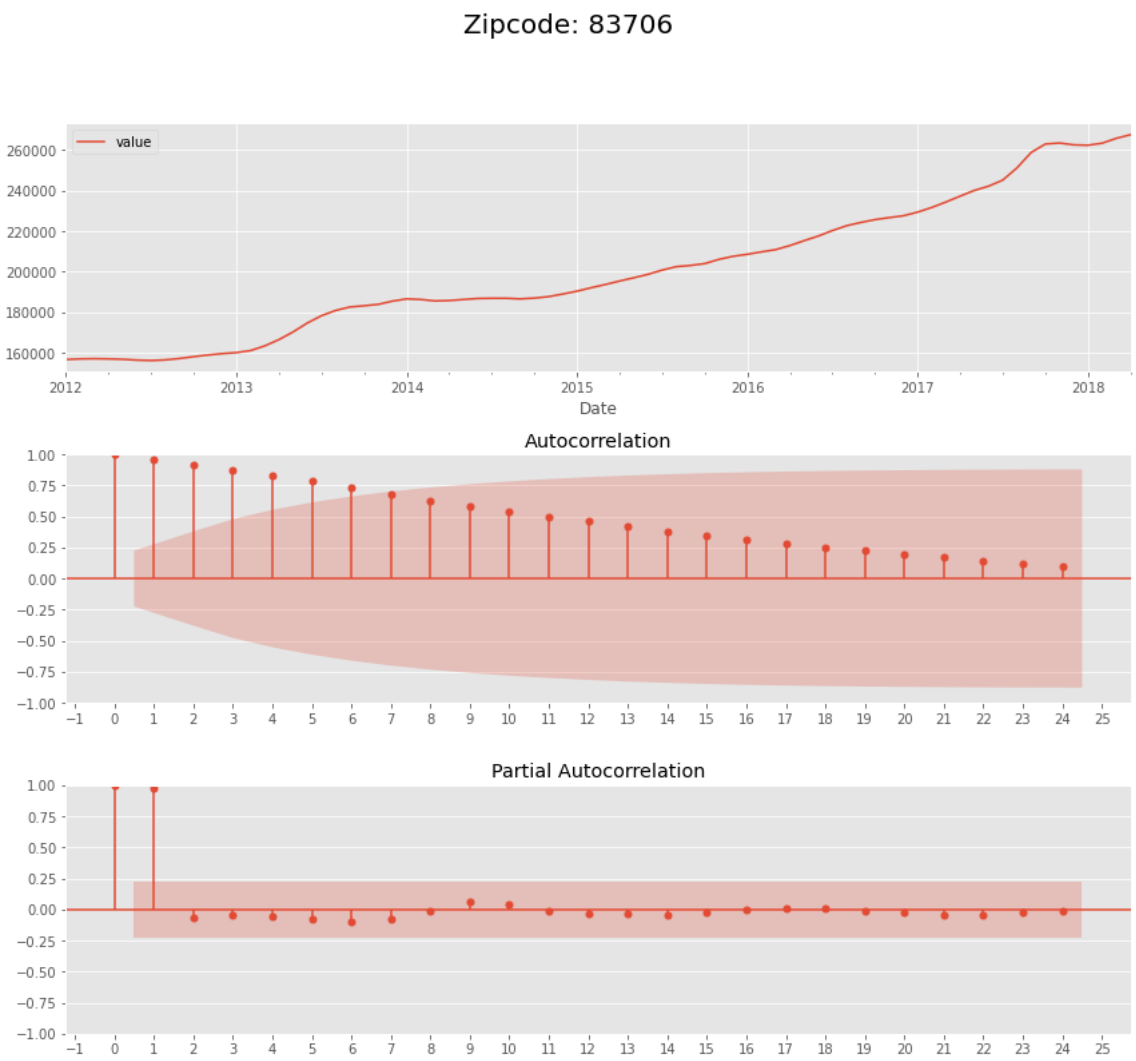
Out[83]: <matplotlib.legend.Legend at 0x7faf4d17e8b0>




### AIC & PACF

- The time series trends positively.
- The ACF graph shows a slight decrease over time due to trend.
- The PACF graph shows a spike at 1 which would suggest an AR value of 1.

In [84]:
```
plot_acf_pacf(seven_zero_six, zipcode= '83706')
plt.show()
```




### Parameter selection for the ARIMA time series model

In [85]:
```
# Define the p, d and q parameters to take any value between 0 and 2
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets (use 12 for frequency)
pdqs = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
```

In [86]:
```
ans = find_best_SARIMAX(train, pdq, pdqs)
```

```
ARIMA (0, 0, 0) x (0, 0, 0, 12): AIC Calculated=1630.600370690789
ARIMA (0, 0, 0) x (0, 0, 1, 12): AIC Calculated=299778.7119828826
ARIMA (0, 0, 0) x (0, 1, 0, 12): AIC Calculated=1074.319386740556
ARIMA (0, 0, 0) x (0, 1, 1, 12): AIC Calculated=802.7936429817904
ARIMA (0, 0, 0) x (1, 0, 0, 12): AIC Calculated=1015.1359610457447
ARIMA (0, 0, 0) x (1, 0, 1, 12): AIC Calculated=1044.5326279235276
ARIMA (0, 0, 0) x (1, 1, 0, 12): AIC Calculated=805.4105282911407
ARIMA (0, 0, 0) x (1, 1, 1, 12): AIC Calculated=782.2271278487251
ARIMA (0, 0, 1) x (0, 0, 0, 12): AIC Calculated=26984.317663229787
ARIMA (0, 0, 1) x (0, 0, 1, 12): AIC Calculated=305650.9822364844
ARIMA (0, 0, 1) x (0, 1, 0, 12): AIC Calculated=1017.1010262769502
ARIMA (0, 0, 1) x (0, 1, 1, 12): AIC Calculated=1170.6040240415607
ARIMA (0, 0, 1) x (1, 0, 0, 12): AIC Calculated=1287.8919971259484
ARIMA (0, 0, 1) x (1, 0, 1, 12): AIC Calculated=1237.5358546981583
ARIMA (0, 0, 1) x (1, 1, 0, 12): AIC Calculated=802.0785741871218
ARIMA (0, 0, 1) x (1, 1, 1, 12): AIC Calculated=760.5616316606822
ARIMA (0, 1, 0) x (0, 0, 0, 12): AIC Calculated=1041.8107051210181
ARIMA (0, 1, 0) x (0, 0, 1, 12): AIC Calculated=837.9633728434573
ARIMA (0, 1, 0) x (0, 1, 0, 12): AIC Calculated=846.4349141918692
ARIMA (0, 1, 0) x (0, 1, 1, 12): AIC Calculated=1016.6200447998885
ARIMA (0, 1, 0) x (1, 0, 0, 12): AIC Calculated=852.6383125058783
ARIMA (0, 1, 0) x (1, 0, 1, 12): AIC Calculated=836.6605600984924
ARIMA (0, 1, 0) x (1, 1, 0, 12): AIC Calculated=601.9772713729914
ARIMA (0, 1, 0) x (1, 1, 1, 12): AIC Calculated=1274.7082943519729
ARIMA (0, 1, 1) x (0, 0, 0, 12): AIC Calculated=963.6577537815411
ARIMA (0, 1, 1) x (0, 0, 1, 12): AIC Calculated=770.3479731293045
ARIMA (0, 1, 1) x (0, 1, 0, 12): AIC Calculated=813.4235158270116
ARIMA (0, 1, 1) x (0, 1, 1, 12): AIC Calculated=1713.0023171565635
ARIMA (0, 1, 1) x (1, 0, 0, 12): AIC Calculated=808.4468150646474
ARIMA (0, 1, 1) x (1, 0, 1, 12): AIC Calculated=772.0500872585386
ARIMA (0, 1, 1) x (1, 1, 0, 12): AIC Calculated=616.7253580447365
ARIMA (0, 1, 1) x (1, 1, 1, 12): AIC Calculated=2620.6104864974222
ARIMA (1, 0, 0) x (0, 0, 0, 12): AIC Calculated=1007.4613307466244
ARIMA (1, 0, 0) x (0, 0, 1, 12): AIC Calculated=870.5560035112358
ARIMA (1, 0, 0) x (0, 1, 0, 12): AIC Calculated=865.2178314781836
ARIMA (1, 0, 0) x (0, 1, 1, 12): AIC Calculated=639.1211169238202
ARIMA (1, 0, 0) x (1, 0, 0, 12): AIC Calculated=796.4553286401134
ARIMA (1, 0, 0) x (1, 0, 1, 12): AIC Calculated=799.7520240047911
ARIMA (1, 0, 0) x (1, 1, 0, 12): AIC Calculated=603.5728506248646
ARIMA (1, 0, 0) x (1, 1, 1, 12): AIC Calculated=626.835080380123
ARIMA (1, 0, 1) x (0, 0, 0, 12): AIC Calculated=948.2060918734034
ARIMA (1, 0, 1) x (0, 0, 1, 12): AIC Calculated=1199.9311512671538
ARIMA (1, 0, 1) x (0, 1, 0, 12): AIC Calculated=836.8733634072065
ARIMA (1, 0, 1) x (0, 1, 1, 12): AIC Calculated=587.008126765004
ARIMA (1, 0, 1) x (1, 0, 0, 12): AIC Calculated=771.81686088334
ARIMA (1, 0, 1) x (1, 0, 1, 12): AIC Calculated=762.3443527615427
ARIMA (1, 0, 1) x (1, 1, 0, 12): AIC Calculated=618.9652867777445
ARIMA (1, 0, 1) x (1, 1, 1, 12): AIC Calculated=600.3647241049706
ARIMA (1, 1, 0) x (0, 0, 0, 12): AIC Calculated=920.7898652850994
ARIMA (1, 1, 0) x (0, 0, 1, 12): AIC Calculated=744.6734199517789
ARIMA (1, 1, 0) x (0, 1, 0, 12): AIC Calculated=774.5397489879541
ARIMA (1, 1, 0) x (0, 1, 1, 12): AIC Calculated=1430.5385531161887
ARIMA (1, 1, 0) x (1, 0, 0, 12): AIC Calculated=744.5106113682418
ARIMA (1, 1, 0) x (1, 0, 1, 12): AIC Calculated=744.9177488107817
ARIMA (1, 1, 0) x (1, 1, 0, 12): AIC Calculated=551.1898329802726
ARIMA (1, 1, 0) x (1, 1, 1, 12): AIC Calculated=1534.9053896414075
ARIMA (1, 1, 1) x (0, 0, 0, 12): AIC Calculated=902.0616227007927
ARIMA (1, 1, 1) x (0, 0, 1, 12): AIC Calculated=715.747454252641
ARIMA (1, 1, 1) x (0, 1, 0, 12): AIC Calculated=755.3211954968688
ARIMA (1, 1, 1) x (0, 1, 1, 12): AIC Calculated=1174.8063096818555
ARIMA (1, 1, 1) x (1, 0, 0, 12): AIC Calculated=745.5369961710842
ARIMA (1, 1, 1) x (1, 0, 1, 12): AIC Calculated=717.6801741481116
ARIMA (1, 1, 1) x (1, 1, 0, 12): AIC Calculated=550.2860748526953
ARIMA (1, 1, 1) x (1, 1, 1, 12): AIC Calculated=1295.0592920664537
```

In [87]:
```
# Find the parameters with minimal AIC value
ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'aic'])
ans_df.loc[ans_df['aic'].idxmin()]
```

Out[87]:
```
pdq         (1, 1, 1)
pdqs    (1, 1, 0, 12)
aic           550.286
Name: 62, dtype: object
```

- The output of our code suggests that `ARIMA (1, 1, 1) x (1, 1, 0, 12)` yields the lowest AIC value of `550.286`. We should therefore consider this to be optimal option out of all the models we have considered.

In [88]:
```
output = fit_SARIMAX_model(seven_zero_six, (1, 1, 1), (1, 1, 0, 12))
output.summary()
```

Out[88]:

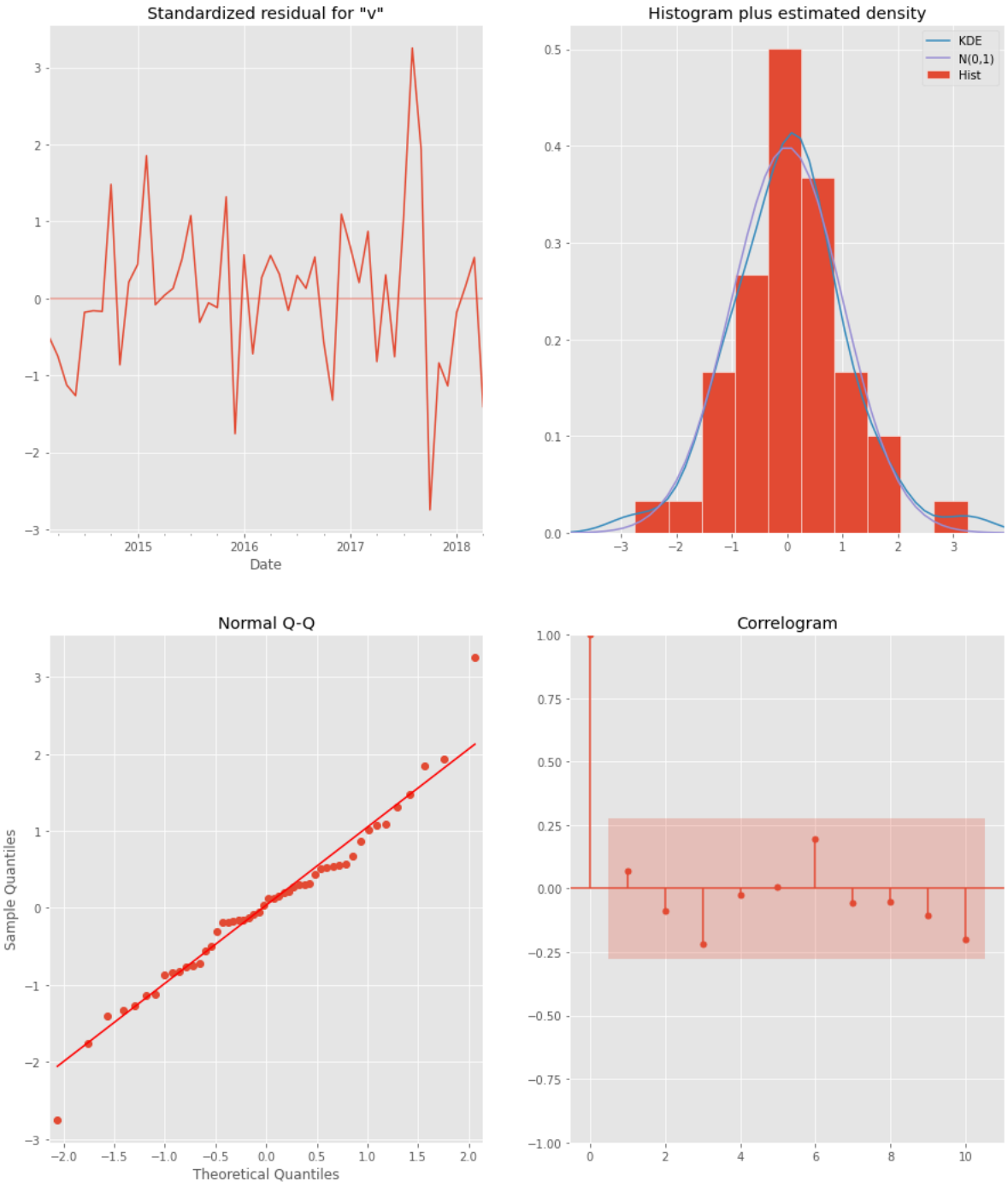
SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | value | **No. Observations:** | 76 |
| **Model:** | SARIMAX(1, 1, 1)x(1, 1, [], 12) | **Log Likelihood** | -409.431 |
| **Date:** | Tue, 14 Feb 2023 | **AIC** | 826.862 |
| **Time:** | 12:57:49 | **BIC** | 834.510 |
| **Sample:** | 01-01-2012 | **HQIC** | 829.774 |
| | - 04-01-2018 | | |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **ar.L1** | 0.6709 | 0.087 | 7.748 | 0.000 | 0.501 | 0.841 |
| **ma.L1** | 0.6866 | 0.142 | 4.846 | 0.000 | 0.409 | 0.964 |
| **ar.S.L12** | 0.0290 | 0.027 | 1.085 | 0.278 | -0.023 | 0.082 |
| **sigma2** | 7.358e+05 | 1.41e+05 | 5.227 | 0.000 | 4.6e+05 | 1.01e+06 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 0.25 | **Jarque-Bera (JB):** | 4.34 |
| **Prob(Q):** | 0.62 | **Prob(JB):** | 0.11 |
| **Heteroskedasticity (H):** | 2.62 | **Skew:** | 0.27 |
| **Prob(H) (two-sided):** | 0.05 | **Kurtosis:** | 4.34 |

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

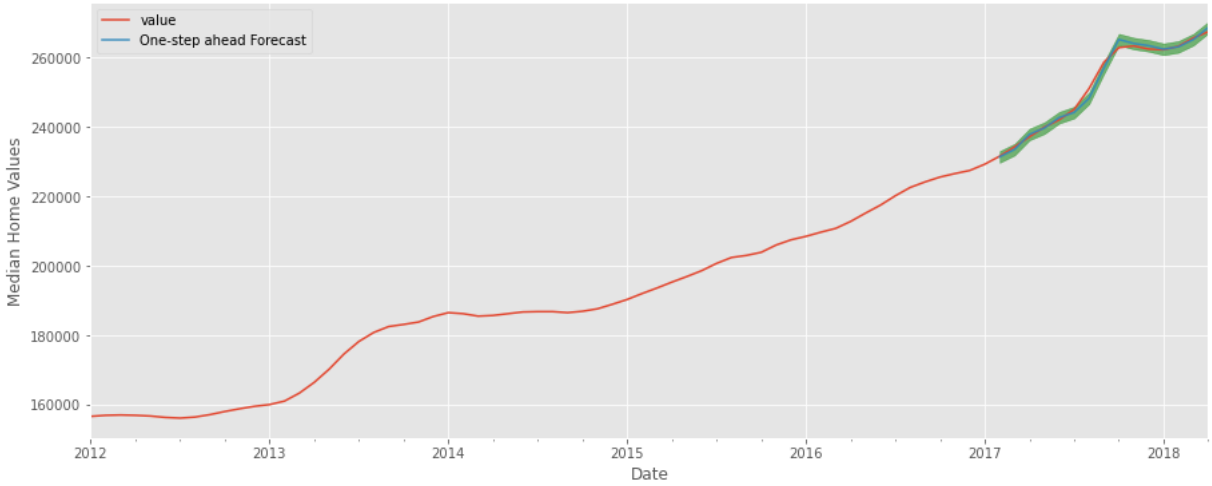



- **Standardized residual**: Seems to show random noise without and distinct pattern.
- **Histogram**: The histogram shows that the distribution appears to be to normaly distributed.
- **QQ-plot**: QQ-plot flares slightly away from the line in a couple of areas but for the most part follows the line.
- **Correlogram**: The points in the correlogram fall inside the box and do not indicate a missed seasonality component.

### Validating the model: One-step ahead forecasting

In [89]:
```
# Get predictions starting from 01-01-1998 and calculate confidence intervals
pred = output.get_prediction(start=pd.to_datetime('2017-02-01'), dynamic=False)
pred_conf = pred.conf_int()
```

In [90]:
```
# Plot real vs predicted values along with confidence interval

rcParams['figure.figsize'] = 15, 6

# Plot observed values
ax = seven_zero_six['2012-01-01':].plot(label='observed')

# Plot predicted values
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=0.9)

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

# Set axes labels
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')
plt.legend()

plt.show()
```




- The forecasts align with the true values as seen above, with overall increase trend.

### Accuracy of our forecast with Root Mean Squared Error

In [91]:
```
# Get the real and predicted values
seven_zero_six_forecasted = pred.predicted_mean
seven_zero_six_truth = seven_zero_six.loc['2017-02-01':].value

# Compute the mean square error
mse = np.sqrt((seven_zero_six_forecasted - seven_zero_six_truth) ** 2).mean()
print('The Root Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

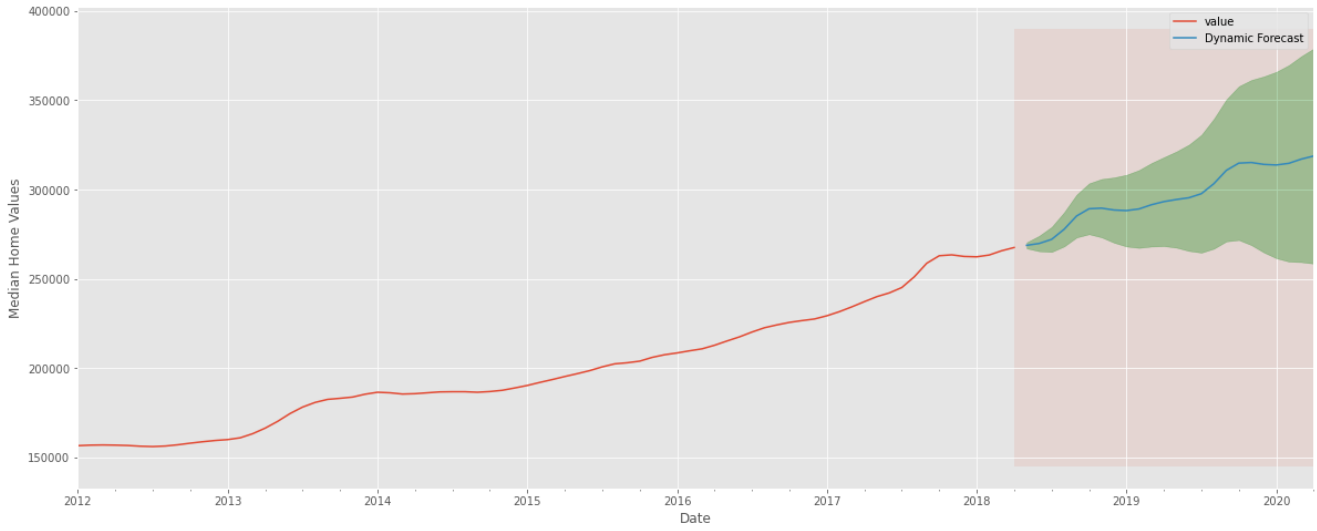
```
The Root Mean Squared Error of our forecasts is 924.03
```

- The RMSE of 924.03 in this case would indicate that, on average, the model's predictions for the mean value of homes in Boise, ID deviate by 924.03 dollars from the actual values.

### Dynamic Forecasting

In [92]:
```
# Get dynamic predictions with confidence intervals as above
pred_dynamic = output.get_forecast(steps=24)
pred_dynamic_conf = pred_dynamic.conf_int()
```

In [93]:
```
# Plot the dynamic forecast with confidence intervals as above
# Plot the dynamic forecast with confidence intervals.

ax = seven_zero_six['2012':].plot(label='observed', figsize=(20, 8))

pred_dynamic.predicted_mean.plot(label='Dynamic Forecast', ax=ax)

ax.fill_between(pred_dynamic_conf.index,
                pred_dynamic_conf.iloc[:, 0],
                pred_dynamic_conf.iloc[:, 1], color='g', alpha=.3)

ax.fill_betweenx(ax.get_ylim(), pd.to_datetime('2020-04-01'), seven_zero_six_forecasted.index[-1], alpha=.1, zorder=-1)

ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')

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




- The dynamic forecast shows an increasing trend in home values.

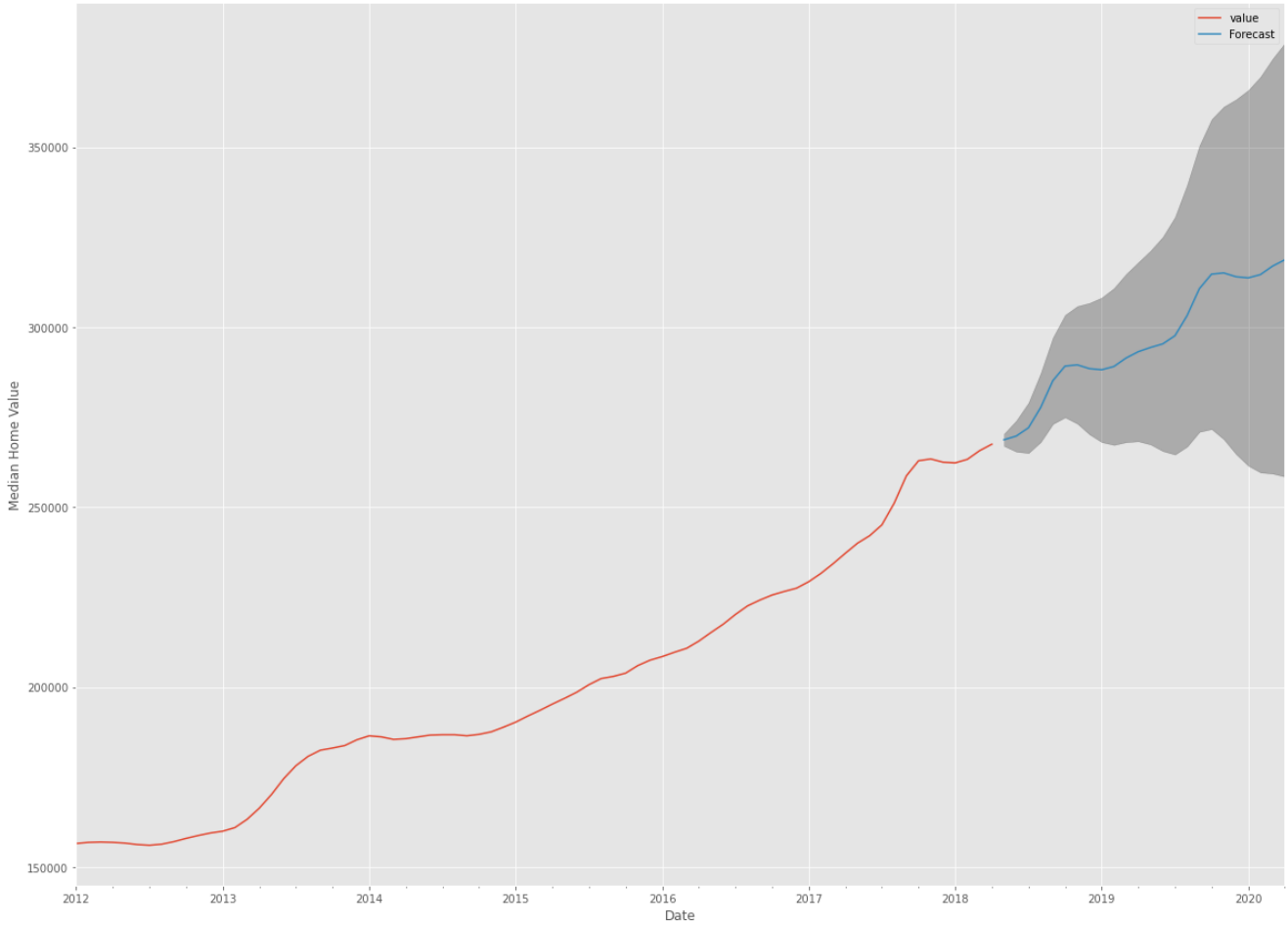
### Producing and visualizing forecasts

In [94]:
```
# Get forecast 24 steps ahead in future.
prediction = output.get_forecast(steps=24)

# Get confidence intervals of forecasts
pred_conf = prediction.conf_int()

pred_conf['mean'] = prediction.predicted_mean

pred_conf.head()
```

Out[94]:

| | lower value | upper value | mean |
|---|---|---|---|
| **2018-05-01** | 267046.698817 | 270409.114955 | 268727.906886 |
| **2018-06-01** | 265454.251462 | 274064.814728 | 269759.533095 |
| **2018-07-01** | 265074.858118 | 279035.817331 | 272055.337724 |
| **2018-08-01** | 268128.361334 | 287234.822867 | 277681.592100 |
| **2018-09-01** | 273165.512592 | 297104.400456 | 285134.956524 |

In [95]:
```
# Plot future predictions with confidence intervals
ax = seven_zero_six.plot(label='observed', figsize=(20, 15))
prediction.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_conf.index,
                pred_conf.iloc[:, 0],
                pred_conf.iloc[:, 1], color='k', alpha=0.25)
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Value')

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




- Our forecasts show that the time series is expected to continue increasing at a steady pace.

## Average Return on Investment (ROI)

In [96]:
```
#percentage over time is now the mean
#mean change percentage
cost = pred_conf.iloc[0]['mean']
roi = (pred_conf - cost)/ abs(cost) * 100

#pred_conf['result'] = pred_conf[''] - pred_conf['']

roi
```

Out[96]:

| | lower value | upper value | mean |
|---|---|---|---|
| **2018-05-01** | -0.625617 | 0.625617 | 0.000000 |
| **2018-06-01** | -1.218204 | 1.985989 | 0.383892 |
| **2018-07-01** | -1.359386 | 3.835817 | 1.238216 |
| **2018-08-01** | -0.223105 | 6.886860 | 3.331878 |
| **2018-09-01** | 1.651338 | 10.559563 | 6.105451 |
| **2018-10-01** | 2.327615 | 12.909408 | 7.618512 |
| **2018-11-01** | 1.677395 | 13.813428 | 7.745412 |
| **2018-12-01** | 0.564219 | 14.145734 | 7.354977 |
| **2019-01-01** | -0.230355 | 14.699824 | 7.234735 |
| **2019-02-01** | -0.520230 | 15.673372 | 7.576571 |
| **2019-03-01** | -0.235618 | 17.146682 | 8.455532 |
| **2019-04-01** | -0.146595 | 18.358907 | 9.106156 |
| **2019-05-01** | -0.472454 | 19.554839 | 9.541192 |
| **2019-06-01** | -1.155713 | 20.976206 | 9.910247 |
| **2019-07-01** | -1.515341 | 23.024843 | 10.754751 |
| **2019-08-01** | -0.685475 | 26.371260 | 12.842892 |
| **2019-09-01** | 0.829397 | 30.398376 | 15.613887 |
| **2019-10-01** | 1.115251 | 33.134414 | 17.124833 |
| **2019-11-01** | 0.058930 | 34.440204 | 17.249567 |
| **2019-12-01** | -1.466051 | 35.180489 | 16.857219 |
| **2020-01-01** | -2.672112 | 36.143002 | 16.735445 |
| **2020-02-01** | -3.369476 | 37.522009 | 17.076266 |
| **2020-03-01** | -3.486271 | 39.395723 | 17.954726 |
| **2020-04-01** | -3.792029 | 41.001492 | 18.604731 |

## Results:

In [97]:
```
results.Two_YR_ROI_Percentage[2] = roi['mean'][-1]
results.head()
```

Out[97]:

| | Zipcode | Two_YR_ROI_Percentage |
|---|---|---|
| **0** | 83709 | 14 |
| **1** | 83704 | 18 |
| **2** | 83706 | 18 |
| **3** | 83705 | 1 |
| **4** | 83702 | 1 |

## 83705

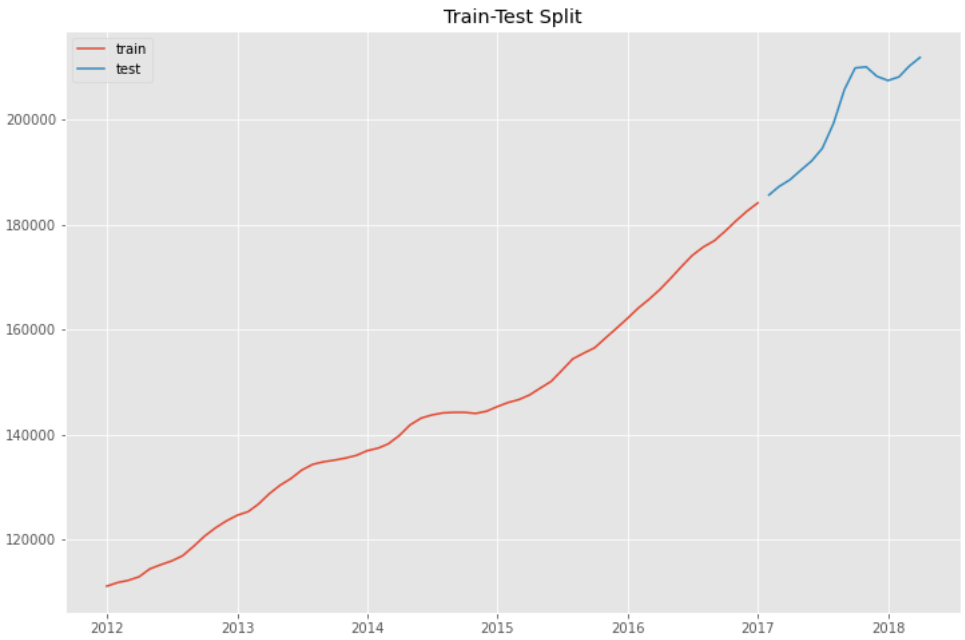
In [98]:
```
seven_zero_five = boise_grouped.loc[83705]
seven_zero_five.tail()
```

Out[98]:

| Date | value |
|---|---|
| 2017-12-01 | 208200.0 |
| 2018-01-01 | 207400.0 |
| 2018-02-01 | 208100.0 |
| 2018-03-01 | 210100.0 |
| 2018-04-01 | 211800.0 |

In [99]:
```
# find the index which allows us to split off 20% of the data
cutoff = round(seven_zero_five.shape[0]*0.8)


train = seven_zero_five[:cutoff]

test = seven_zero_five[cutoff:]

fig, ax = plt.subplots(figsize=(12, 8))
ax.plot(train, label='train')
ax.plot(test, label='test')
ax.set_title('Train-Test Split');
plt.legend()
```

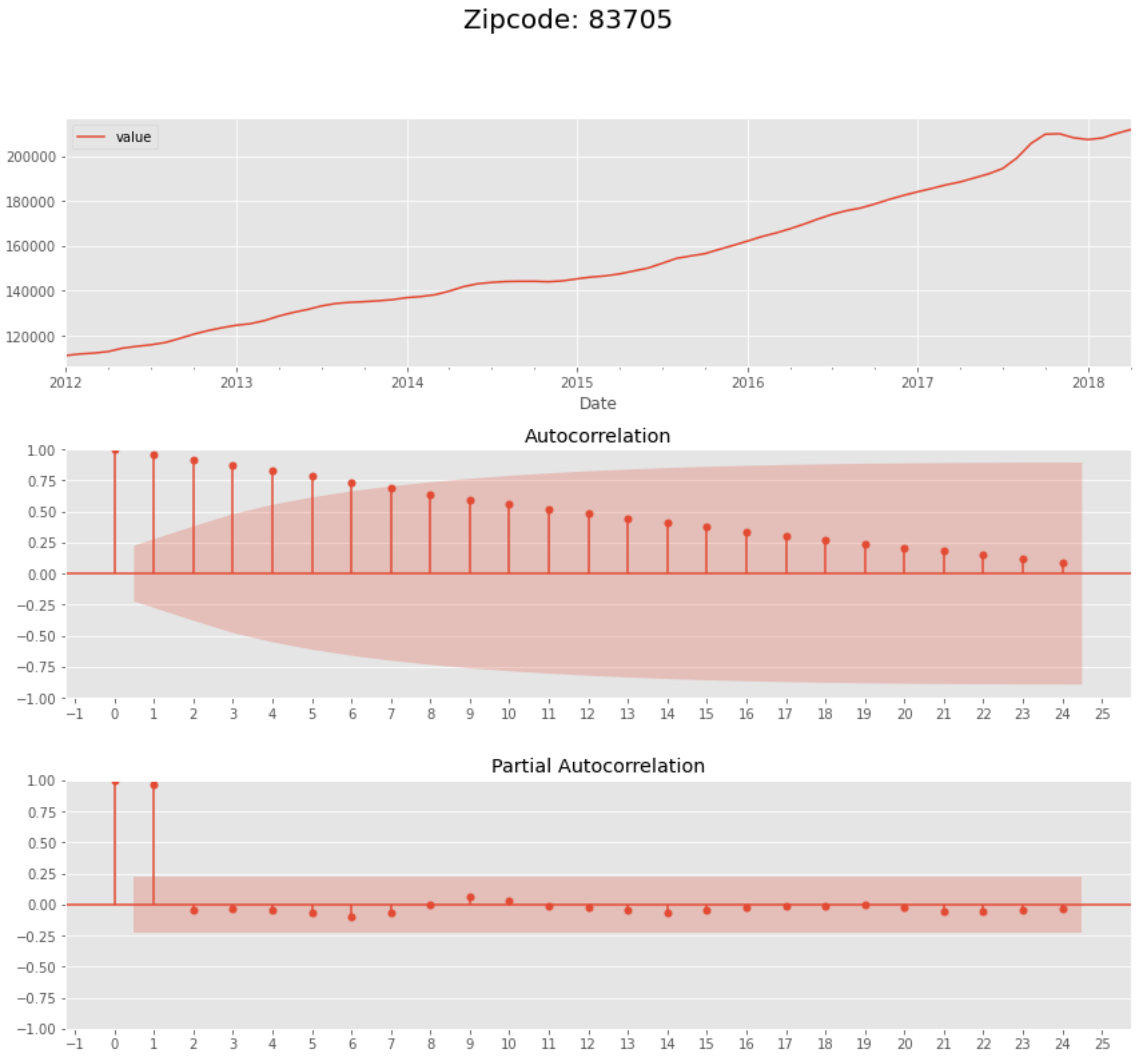
Out[99]: <matplotlib.legend.Legend at 0x7faf4c2dc580>




### ACF & PACF

- The time series trends positively.
- The ACF graph shows a slight decrease over time due to trend.
- The PACF graph shows a spike at 1 which would suggest an AR value of 1.

In [100]:
```
plot_acf_pacf(seven_zero_five, zipcode= '83705')
plt.show()
```




### Parameter selection for the ARIMA time series model

In [101]:
```
# Define the p, d and q parameters to take any value between 0 and 2
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets (use 12 for frequency)
pdqs = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
```

In [102]:
```
ans = find_best_SARIMAX(train, pdq, pdqs)
```

```
ARIMA (0, 0, 0) x (0, 0, 0, 12): AIC Calculated=1598.6711863510395
ARIMA (0, 0, 0) x (0, 0, 1, 12): AIC Calculated=1259.8810035897745
ARIMA (0, 0, 0) x (0, 1, 0, 12): AIC Calculated=1061.5899381339345
ARIMA (0, 0, 0) x (0, 1, 1, 12): AIC Calculated=798.6665275186123
ARIMA (0, 0, 0) x (1, 0, 0, 12): AIC Calculated=967.237043989679
ARIMA (0, 0, 0) x (1, 0, 1, 12): AIC Calculated=948.1033366151078
ARIMA (0, 0, 0) x (1, 1, 0, 12): AIC Calculated=766.7850898512145
ARIMA (0, 0, 0) x (1, 1, 1, 12): AIC Calculated=735.3933651281499
ARIMA (0, 0, 1) x (0, 0, 0, 12): AIC Calculated=1532.718444392994
ARIMA (0, 0, 1) x (0, 0, 1, 12): AIC Calculated=1281.0365609013172
ARIMA (0, 0, 1) x (0, 1, 0, 12): AIC Calculated=1048.3382864487999
ARIMA (0, 0, 1) x (0, 1, 1, 12): AIC Calculated=752.5803288352085
ARIMA (0, 0, 1) x (1, 0, 0, 12): AIC Calculated=1261.9918225960093
ARIMA (0, 0, 1) x (1, 0, 1, 12): AIC Calculated=1212.7094372378833
ARIMA (0, 0, 1) x (1, 1, 0, 12): AIC Calculated=762.6890665620614
ARIMA (0, 0, 1) x (1, 1, 1, 12): AIC Calculated=749.0113771759944
ARIMA (0, 1, 0) x (0, 0, 0, 12): AIC Calculated=1022.5612005633831
ARIMA (0, 1, 0) x (0, 0, 1, 12): AIC Calculated=789.2835719237621
ARIMA (0, 1, 0) x (0, 1, 0, 12): AIC Calculated=771.3744707636863
ARIMA (0, 1, 0) x (0, 1, 1, 12): AIC Calculated=1252.0145943340835
ARIMA (0, 1, 0) x (1, 0, 0, 12): AIC Calculated=788.0576363967268
ARIMA (0, 1, 0) x (1, 0, 1, 12): AIC Calculated=775.327794133499
ARIMA (0, 1, 0) x (1, 1, 0, 12): AIC Calculated=588.6195391706319
ARIMA (0, 1, 0) x (1, 1, 1, 12): AIC Calculated=1164.0567167922725
ARIMA (0, 1, 1) x (0, 0, 0, 12): AIC Calculated=944.1115544775629
ARIMA (0, 1, 1) x (0, 0, 1, 12): AIC Calculated=728.8315163375156
ARIMA (0, 1, 1) x (0, 1, 0, 12): AIC Calculated=742.7754373784599
ARIMA (0, 1, 1) x (0, 1, 1, 12): AIC Calculated=1218.357035061413
ARIMA (0, 1, 1) x (1, 0, 0, 12): AIC Calculated=778.9693316758885
ARIMA (0, 1, 1) x (1, 0, 1, 12): AIC Calculated=730.2026950882163
ARIMA (0, 1, 1) x (1, 1, 0, 12): AIC Calculated=565.410377454018
ARIMA (0, 1, 1) x (1, 1, 1, 12): AIC Calculated=1236.4092537026816
ARIMA (1, 0, 0) x (0, 0, 0, 12): AIC Calculated=939.0994007483871
ARIMA (1, 0, 0) x (0, 0, 1, 12): AIC Calculated=756.705666964361
ARIMA (1, 0, 0) x (0, 1, 0, 12): AIC Calculated=786.3967437588874
ARIMA (1, 0, 0) x (0, 1, 1, 12): AIC Calculated=586.0990732625852
ARIMA (1, 0, 0) x (1, 0, 0, 12): AIC Calculated=758.1871346191732
ARIMA (1, 0, 0) x (1, 0, 1, 12): AIC Calculated=758.6943534125326
ARIMA (1, 0, 0) x (1, 1, 0, 12): AIC Calculated=585.0031021800146
ARIMA (1, 0, 0) x (1, 1, 1, 12): AIC Calculated=586.8829846142268
ARIMA (1, 0, 1) x (0, 0, 0, 12): AIC Calculated=885.4441081226115
ARIMA (1, 0, 1) x (0, 0, 1, 12): AIC Calculated=707.3450065000909
ARIMA (1, 0, 1) x (0, 1, 0, 12): AIC Calculated=763.9114484429819
ARIMA (1, 0, 1) x (0, 1, 1, 12): AIC Calculated=547.2691433066082
ARIMA (1, 0, 1) x (1, 0, 0, 12): AIC Calculated=721.2342928674689
ARIMA (1, 0, 1) x (1, 0, 1, 12): AIC Calculated=709.0071832494253
ARIMA (1, 0, 1) x (1, 1, 0, 12): AIC Calculated=566.6411954082168
ARIMA (1, 0, 1) x (1, 1, 1, 12): AIC Calculated=550.7702651333877
ARIMA (1, 1, 0) x (0, 0, 0, 12): AIC Calculated=889.8738014590778
ARIMA (1, 1, 0) x (0, 0, 1, 12): AIC Calculated=710.3038709617047
ARIMA (1, 1, 0) x (0, 1, 0, 12): AIC Calculated=734.7419987029332
ARIMA (1, 1, 0) x (0, 1, 1, 12): AIC Calculated=925.8077301285429
ARIMA (1, 1, 0) x (1, 0, 0, 12): AIC Calculated=710.8690279788876
ARIMA (1, 1, 0) x (1, 0, 1, 12): AIC Calculated=712.526479932625
ARIMA (1, 1, 0) x (1, 1, 0, 12): AIC Calculated=533.6060731841527
ARIMA (1, 1, 0) x (1, 1, 1, 12): AIC Calculated=1202.735996335872
ARIMA (1, 1, 1) x (0, 0, 0, 12): AIC Calculated=874.1192299289086
ARIMA (1, 1, 1) x (0, 0, 1, 12): AIC Calculated=688.4880083890959
ARIMA (1, 1, 1) x (0, 1, 0, 12): AIC Calculated=717.8258282108985
ARIMA (1, 1, 1) x (0, 1, 1, 12): AIC Calculated=1003.5542605142444
ARIMA (1, 1, 1) x (1, 0, 0, 12): AIC Calculated=705.3570412337272
ARIMA (1, 1, 1) x (1, 0, 1, 12): AIC Calculated=688.7866432165918
ARIMA (1, 1, 1) x (1, 1, 0, 12): AIC Calculated=543.263431481523
ARIMA (1, 1, 1) x (1, 1, 1, 12): AIC Calculated=1254.3947031262555
```

### Fitting the time series model - ARIMA

In [103]:
```
# Find the parameters with minimal AIC value
ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'aic'])
ans_df.loc[ans_df['aic'].idxmin()]
```

Out[103]:
```
pdq         (1, 1, 0)
pdqs    (1, 1, 0, 12)
aic           533.606
Name: 54, dtype: object
```

- The output of our code suggests that `ARIMA (1, 1, 0) x (1, 1, 0, 12)` yields the lowest AIC value of `533.606` . We should therefore consider this to be optimal option out of all the models we have considered.

In [104]:
```
output = output = fit_SARIMAX_model(seven_zero_five, (1, 1, 0), (1, 1, 0, 12))
output.summary()
```

Out[104]:

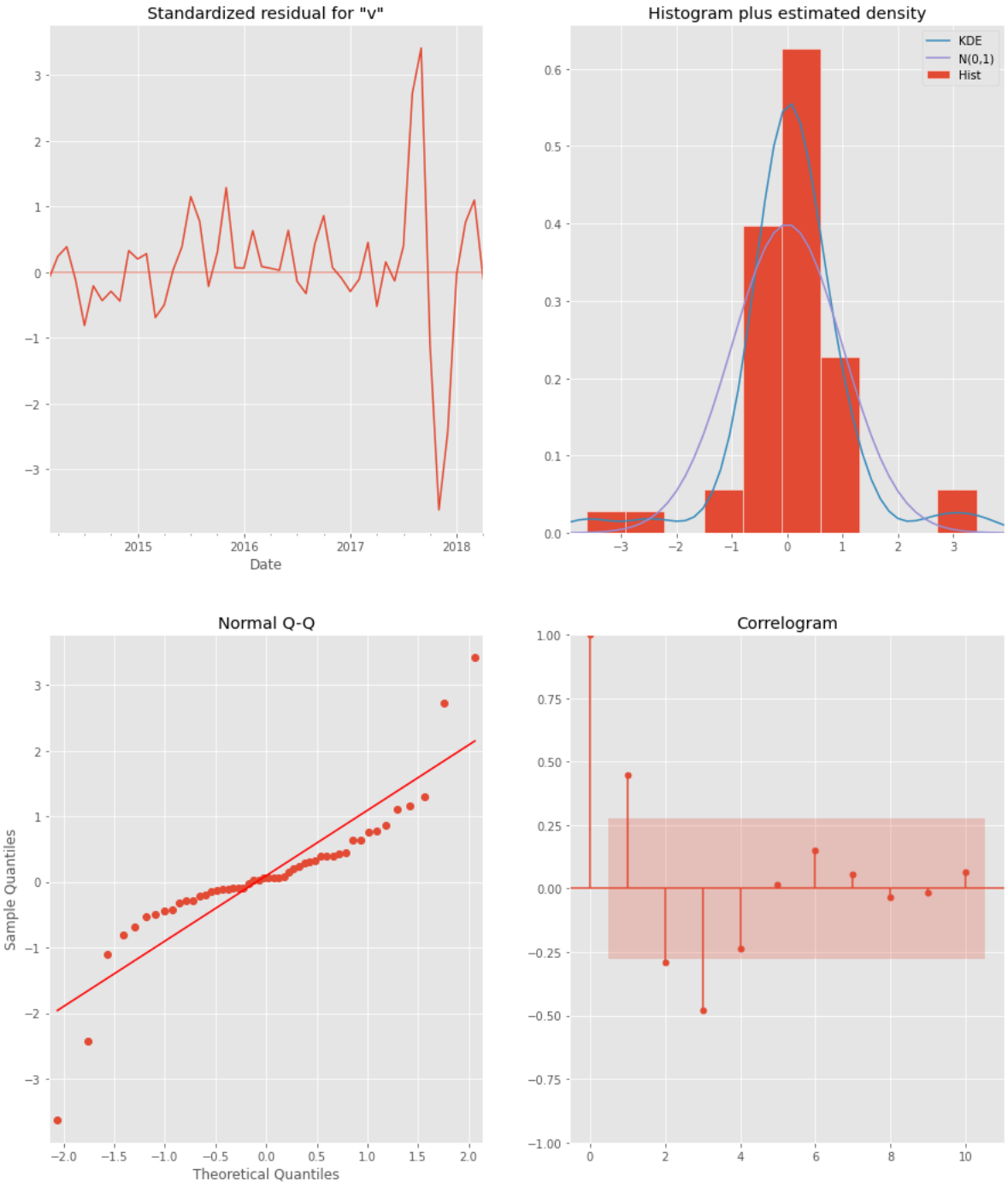
SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | value | **No. Observations:** | 76 |
| **Model:** | SARIMAX(1, 1, 0)x(1, 1, 0, 12) | **Log Likelihood** | -415.398 |
| **Date:** | Tue, 14 Feb 2023 | **AIC** | 836.795 |
| **Time:** | 12:58:00 | **BIC** | 842.531 |
| **Sample:** | 01-01-2012 | **HQIC** | 838.980 |
| | - 04-01-2018 | | |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **ar.L1** | 0.6892 | 0.063 | 10.969 | 0.000 | 0.566 | 0.812 |
| **ar.S.L12** | -0.3692 | 0.312 | -1.183 | 0.237 | -0.981 | 0.242 |
| **sigma2** | 9.632e+05 | 1.12e+05 | 8.565 | 0.000 | 7.43e+05 | 1.18e+06 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 10.50 | **Jarque-Bera (JB):** | 59.55 |
| **Prob(Q):** | 0.00 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 10.95 | **Skew:** | -0.22 |
| **Prob(H) (two-sided):** | 0.00 | **Kurtosis:** | 8.33 |

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

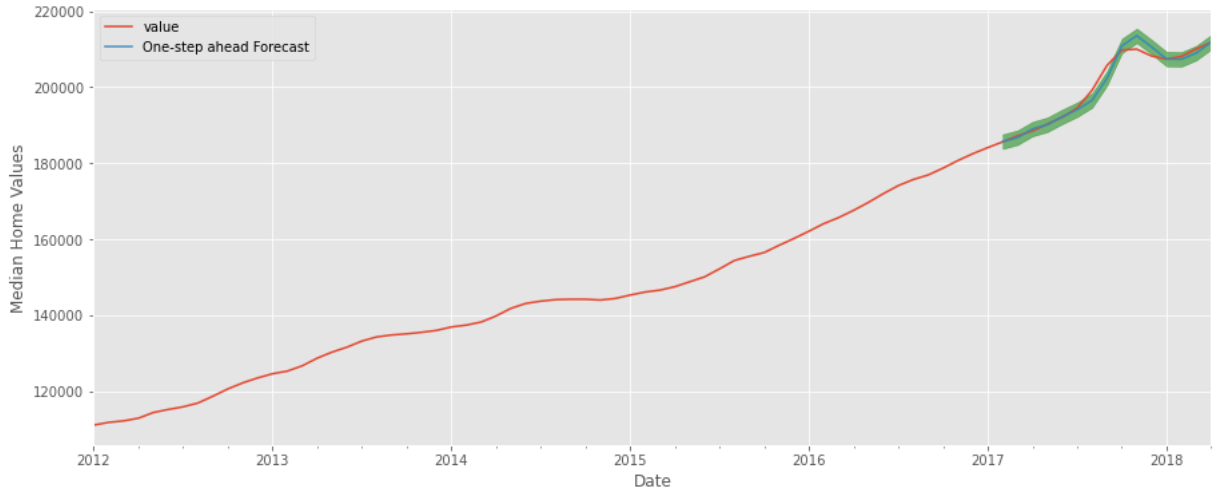



- **Standardized residual**: This looks pretty random for the most part. Maybe so a small amount of seasonlity.
- **Histogram**: The histogram shows that the distribution appears to be pretty close to normal.
- **QQ-plot**: QQ-plot flares away from the line.
- **Correlogram**: There are three spikes outside of the shaded area. This means there could be some missed seasonality component.

### Validating the model: One-step ahead forecasting

In [105]:
```
# Get predictions starting from 01-01-2017 and calculate confidence intervals
pred = output.get_prediction(start=pd.to_datetime('2017-02-01'), dynamic=False)
pred_conf = pred.conf_int()
```

In [106]:
```
# Plot real vs predicted values along with confidence interval

rcParams['figure.figsize'] = 15, 6

# Plot observed values
ax = seven_zero_five['2012-01-01':].plot(label='observed')

# Plot predicted values
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=0.9)

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

# Set axes labels
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')
plt.legend()

plt.show()
```




- The forecasted line for the most part follows the true values. It does however continue on a positive trend.

### Accuracy of our forecast with Root Mean Squared Error

In [107]:
```
# Get the real and predicted values
seven_zero_five_forecasted = pred.predicted_mean
seven_zero_five_truth = seven_zero_five.loc['2017-02-01':].value

# Compute the mean square error
mse = np.sqrt((seven_zero_five_forecasted - seven_zero_five_truth) ** 2).mean()
print('The Root Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

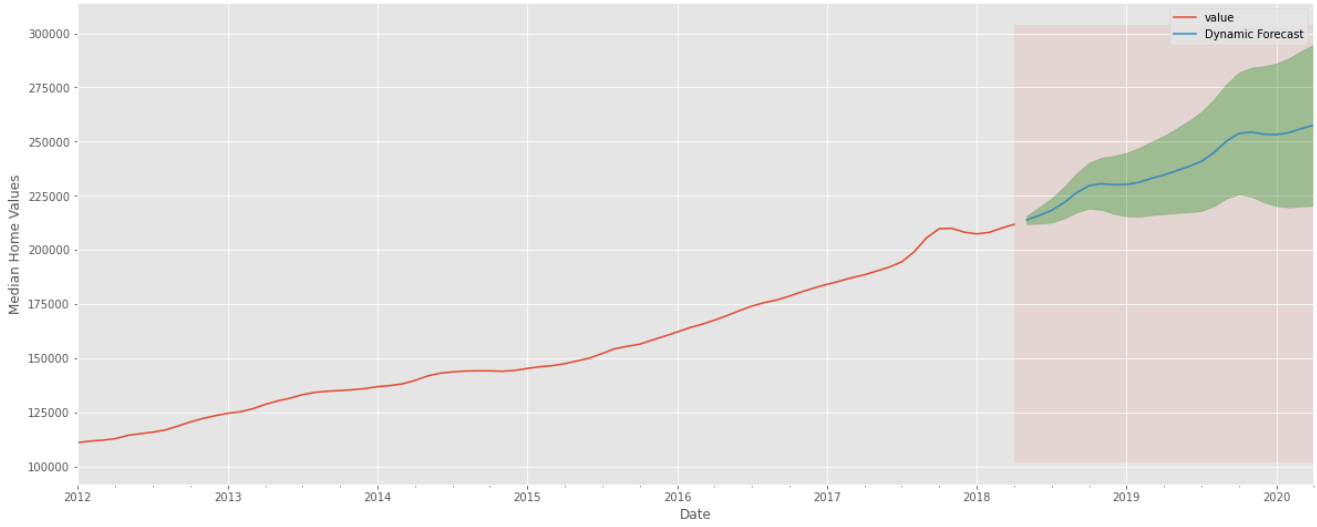
```
The Root Mean Squared Error of our forecasts is 1115.22
```

- The RMSE of 1115.22 in this case would indicate that, on average, the model's predictions for the mean value of homes in Boise, ID deviate by 1115.22 dollars from the actual values.
- AIC = 533.606

### Dynamic Forecasting

In [108]:
```
# Get dynamic predictions with confidence intervals as above
pred_dynamic = output.get_forecast(steps=24)
pred_dynamic_conf = pred_dynamic.conf_int()
```

In [109]:
```
# Plot the dynamic forecast with confidence intervals as above
# Plot the dynamic forecast with confidence intervals.

ax = seven_zero_five['2012':].plot(label='observed', figsize=(20, 8))

pred_dynamic.predicted_mean.plot(label='Dynamic Forecast', ax=ax)

ax.fill_between(pred_dynamic_conf.index,
                pred_dynamic_conf.iloc[:, 0],
                pred_dynamic_conf.iloc[:, 1], color='g', alpha=.3)

ax.fill_betweenx(ax.get_ylim(), pd.to_datetime('2020-04-01'), seven_zero_five_forecasted.index[-1], alpha=.1, zorde

ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')

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




- Our forecasts show that the time series is expected to continue increasing at a steady pace.

### Producing and visualizing forecasts

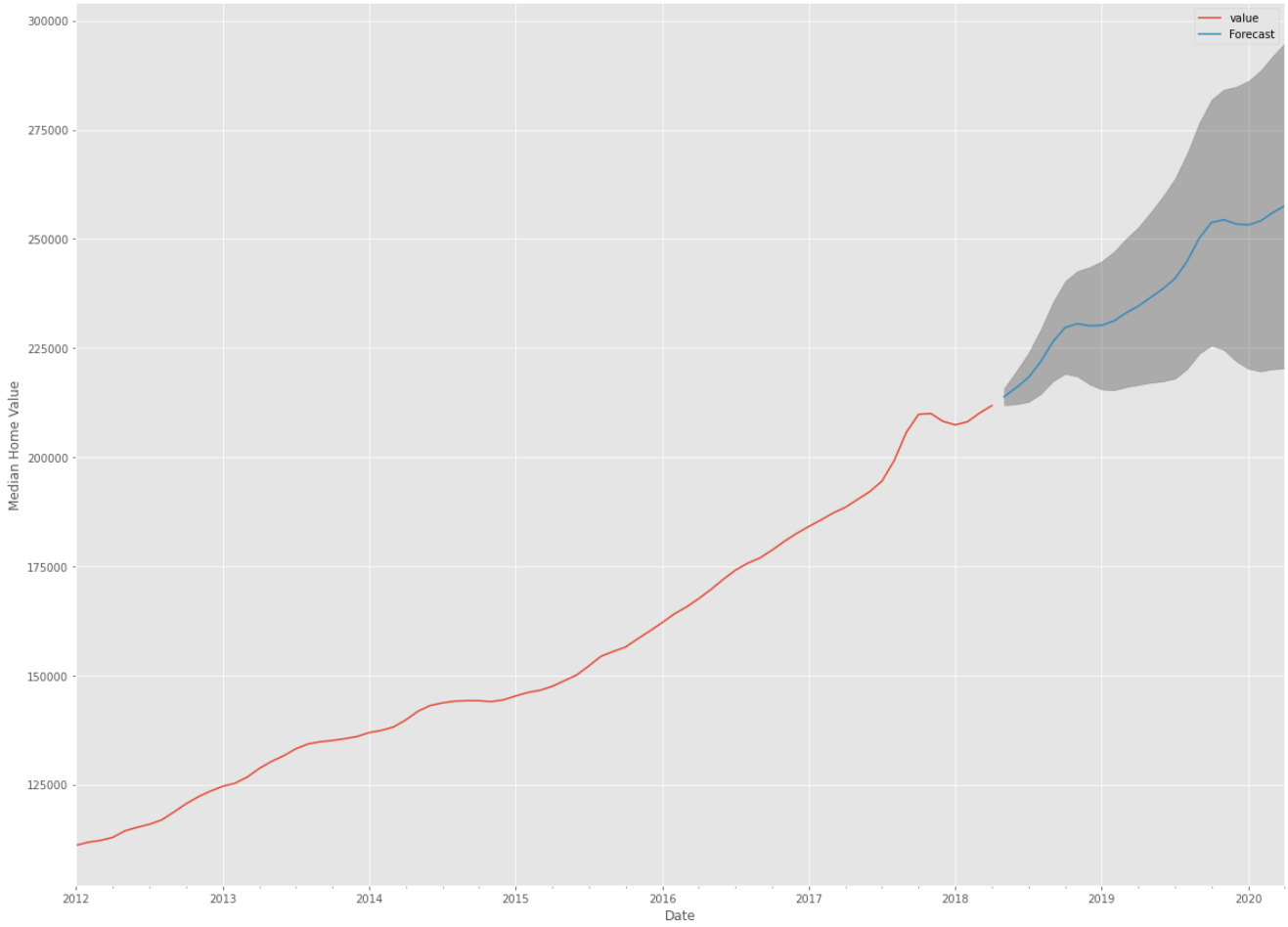
In [110]:
```
# Get forecast 24 steps ahead in future.
prediction = output.get_forecast(steps=24)

# Get confidence intervals of forecasts
pred_conf = prediction.conf_int()

pred_conf['mean'] = prediction.predicted_mean

pred_conf.head()
```

Out[110]:

| | lower value | upper value | mean |
|---|---|---|---|
| **2018-05-01** | 211910.167681 | 215757.347867 | 213833.757774 |
| **2018-06-01** | 212127.166794 | 219679.108589 | 215903.137692 |
| **2018-07-01** | 212630.490825 | 223871.060433 | 218250.775629 |
| **2018-08-01** | 214460.169873 | 229232.565868 | 221846.367871 |
| **2018-09-01** | 217368.804405 | 235465.406908 | 226417.105656 |

In [111]:

```
# Plot future predictions with confidence intervals
ax = seven_zero_five.plot(label='observed', figsize=(20, 15))
prediction.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_conf.index,
                pred_conf.iloc[:, 0],
                pred_conf.iloc[:, 1], color='k', alpha=0.25)
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Value')

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




- Our forecasts show that the time series is expected to continue increasing at a steady pace.

## Average Return on Investment (ROI)

In [112]:
```
#percentage over time is now the mean
#mean change percentage
cost = pred_conf.iloc[0]['mean']
roi = (pred_conf - cost)/ abs(cost) * 100

#pred_conf['result'] = pred_conf[''] - pred_conf['']

roi
```

Out[112]:

| | lower value | upper value | mean |
|---|---|---|---|
| **2018-05-01** | -0.899573 | 0.899573 | 0.000000 |
| **2018-06-01** | -0.798092 | 2.733596 | 0.967752 |
| **2018-07-01** | -0.562711 | 4.693975 | 2.065632 |
| **2018-08-01** | 0.292944 | 7.201299 | 3.747121 |
| **2018-09-01** | 1.653175 | 10.116106 | 5.884640 |
| **2018-10-01** | 2.455728 | 12.371867 | 7.413797 |
| **2018-11-01** | 2.188090 | 13.460535 | 7.824312 |
| **2018-12-01** | 1.338497 | 13.878360 | 7.608428 |
| **2019-01-01** | 0.787942 | 14.515383 | 7.651663 |
| **2019-02-01** | 0.697154 | 15.541207 | 8.119180 |
| **2019-03-01** | 1.037869 | 16.935740 | 8.986805 |
| **2019-04-01** | 1.265614 | 18.161788 | 9.713701 |
| **2019-05-01** | 1.504003 | 19.746175 | 10.625089 |
| **2019-06-01** | 1.651579 | 21.441980 | 11.546780 |
| **2019-07-01** | 1.938760 | 23.369270 | 12.654015 |
| **2019-08-01** | 2.980476 | 26.072382 | 14.526429 |
| **2019-09-01** | 4.630379 | 29.364101 | 16.997240 |
| **2019-10-01** | 5.502715 | 31.836982 | 18.669849 |
| **2019-11-01** | 5.021688 | 32.905090 | 18.963389 |
| **2019-12-01** | 3.827626 | 33.205266 | 18.516446 |
| **2020-01-01** | 2.997016 | 33.814196 | 18.405606 |
| **2020-02-01** | 2.719317 | 34.923470 | 18.821393 |
| **2020-03-01** | 2.943222 | 36.484828 | 19.714025 |
| **2020-04-01** | 3.049621 | 37.882544 | 20.466083 |

## Results:

In [113]:
```
results.Two_YR_ROI_Percentage[3] = roi['mean'][-1]
results.head()
```

Out[113]:

| | Zipcode | Two_YR_ROI_Percentage |
|---|---|---|
| **0** | 83709 | 14 |
| **1** | 83704 | 18 |
| **2** | 83706 | 18 |
| **3** | 83705 | 20 |
| **4** | 83702 | 1 |

- ZipCode 83705 has the greatest 2 year ROI thus far.

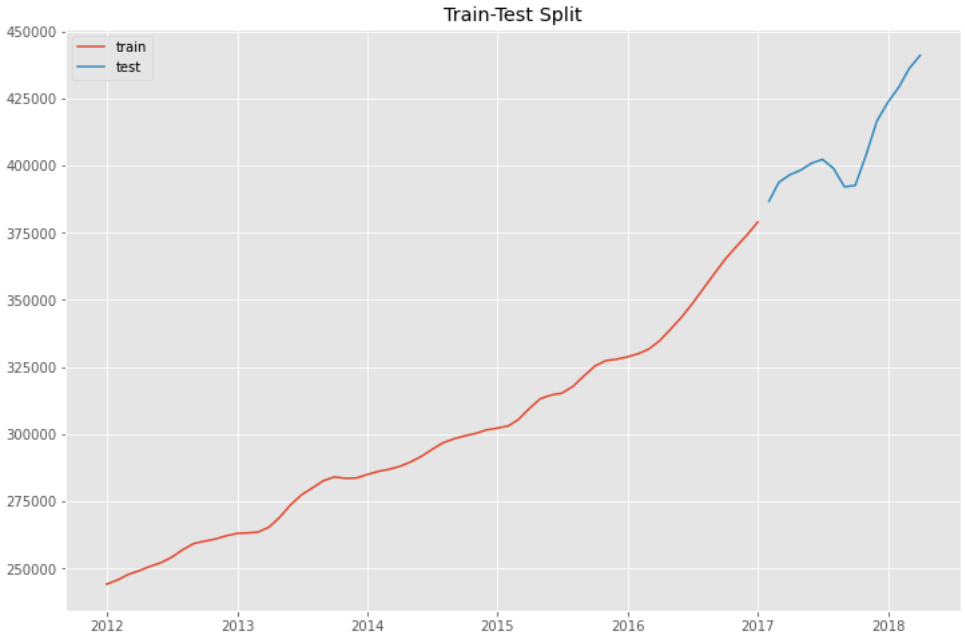
## 83702

In [114]:
```
seven_zero_two = boise_grouped.loc[83702]
seven_zero_two.head()
```

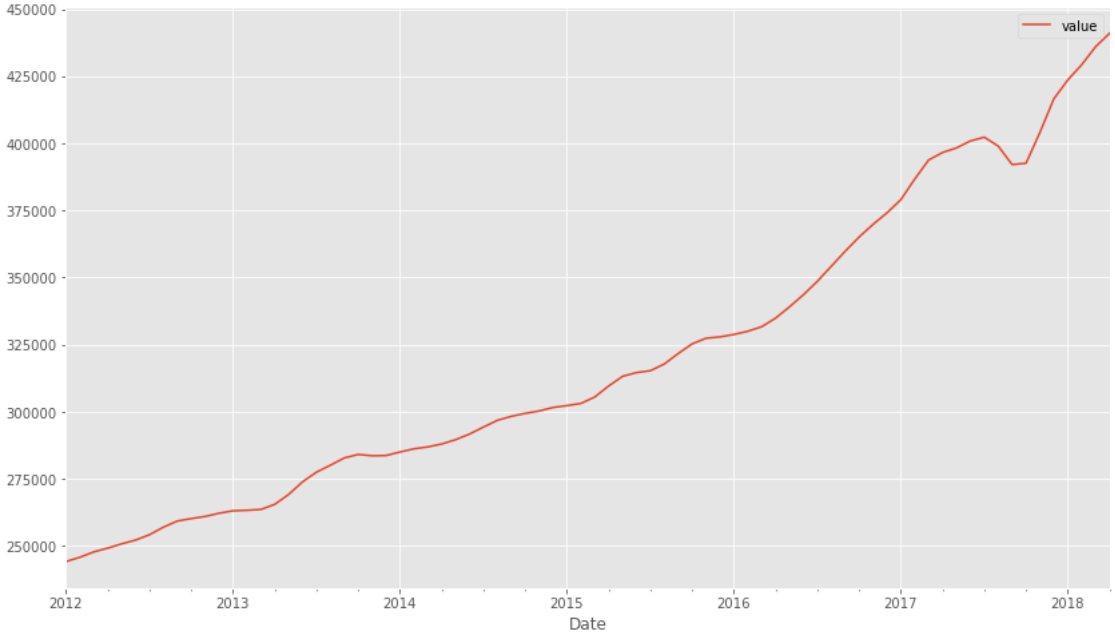
Out[114]:

| Date | value |
|---|---|
| 2012-01-01 | 244200.0 |
| 2012-02-01 | 245800.0 |
| 2012-03-01 | 247800.0 |
| 2012-04-01 | 249200.0 |
| 2012-05-01 | 250800.0 |

In [115]:
```
# find the index which allows us to split off 20% of the data
cutoff = round(seven_zero_two.shape[0]*0.8)


train = seven_zero_two[:cutoff]

test = seven_zero_two[cutoff:]

fig, ax = plt.subplots(figsize=(12, 8))
ax.plot(train, label='train')
ax.plot(test, label='test')
ax.set_title('Train-Test Split')
plt.legend();
```



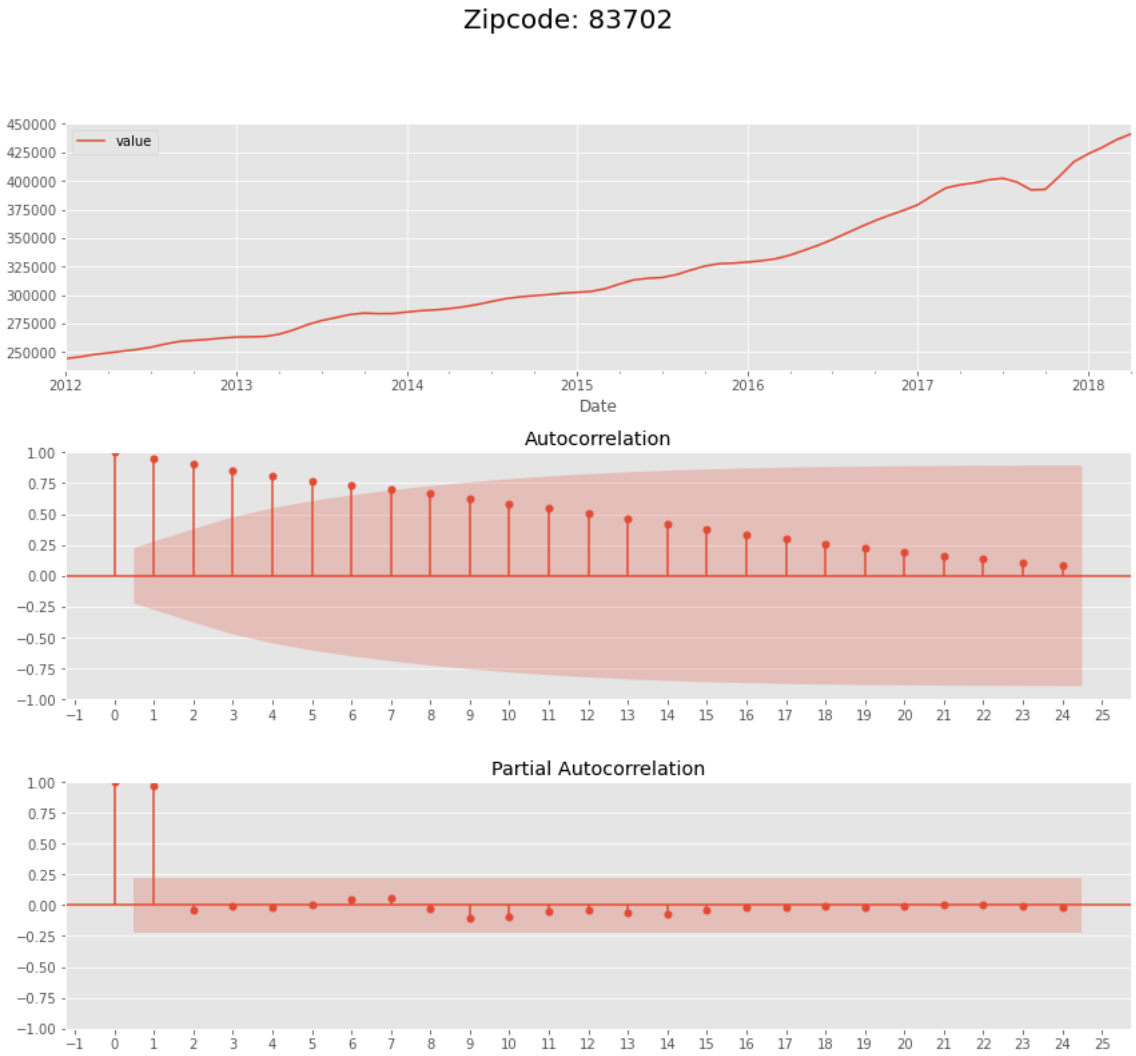
In [116]:
```
1 seven_zero_two.plot(figsize=(14,8))
2 plt.show()
```




### ACF & PACF

- The time series trends positively, with a small decline in mid 2017. However, it increases a couple of months later.
- The ACF graph shows a slight decrease over time due to trend.
- The PACF graph shows a spike at 1 which would suggest an AR value of 1.

In [117]:
```
plot_acf_pacf(seven_zero_two, zipcode= '83702')
plt.show()
```




### Parameter selection for the ARIMA time series model

In [118]:
```
# Define the p, d and q parameters to take any value between 0 and 2
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets (use 12 for frequency)
pdqs = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
```

In [119]:
```
ans = find_best_SARIMAX(train, pdq, pdqs)
```

```
ARIMA (0, 0, 0) x (0, 0, 0, 12): AIC Calculated=1686.1456433389528
ARIMA (0, 0, 0) x (0, 0, 1, 12): AIC Calculated=784354.6742572291
ARIMA (0, 0, 0) x (0, 1, 0, 12): AIC Calculated=1113.9412483107267
ARIMA (0, 0, 0) x (0, 1, 1, 12): AIC Calculated=840.6797209181717
ARIMA (0, 0, 0) x (1, 0, 0, 12): AIC Calculated=1004.0643976827877
ARIMA (0, 0, 0) x (1, 0, 1, 12): AIC Calculated=987.5110341334399
ARIMA (0, 0, 0) x (1, 1, 0, 12): AIC Calculated=775.1590882069529
ARIMA (0, 0, 0) x (1, 1, 1, 12): AIC Calculated=1081.5161967960755
ARIMA (0, 0, 1) x (0, 0, 0, 12): AIC Calculated=62944.822576005405
ARIMA (0, 0, 1) x (0, 0, 1, 12): AIC Calculated=799657.1425231297
ARIMA (0, 0, 1) x (0, 1, 0, 12): AIC Calculated=1066.4150991736715
ARIMA (0, 0, 1) x (0, 1, 1, 12): AIC Calculated=976.2209874984102
ARIMA (0, 0, 1) x (1, 0, 0, 12): AIC Calculated=1333.2210704768204
ARIMA (0, 0, 1) x (1, 0, 1, 12): AIC Calculated=1281.0418984325484
ARIMA (0, 0, 1) x (1, 1, 0, 12): AIC Calculated=829.7538044306043
ARIMA (0, 0, 1) x (1, 1, 1, 12): AIC Calculated=785.747441237041
ARIMA (0, 1, 0) x (0, 0, 0, 12): AIC Calculated=1103.4354200525486
ARIMA (0, 1, 0) x (0, 0, 1, 12): AIC Calculated=889.1546171608577
ARIMA (0, 1, 0) x (0, 1, 0, 12): AIC Calculated=842.4155071420322
ARIMA (0, 1, 0) x (0, 1, 1, 12): AIC Calculated=1559.954966431972
ARIMA (0, 1, 0) x (1, 0, 0, 12): AIC Calculated=861.3960528518612
ARIMA (0, 1, 0) x (1, 0, 1, 12): AIC Calculated=846.8132280237962
ARIMA (0, 1, 0) x (1, 1, 0, 12): AIC Calculated=648.1841107346748
ARIMA (0, 1, 0) x (1, 1, 1, 12): AIC Calculated=1456.1819699361467
ARIMA (0, 1, 1) x (0, 0, 0, 12): AIC Calculated=1016.5575170191906
ARIMA (0, 1, 1) x (0, 0, 1, 12): AIC Calculated=798.494036387871
ARIMA (0, 1, 1) x (0, 1, 0, 12): AIC Calculated=814.1682125682428
ARIMA (0, 1, 1) x (0, 1, 1, 12): AIC Calculated=2784.511263775066
ARIMA (0, 1, 1) x (1, 0, 0, 12): AIC Calculated=864.6805082564774
ARIMA (0, 1, 1) x (1, 0, 1, 12): AIC Calculated=800.4600459344251
ARIMA (0, 1, 1) x (1, 1, 0, 12): AIC Calculated=619.6990541875172
ARIMA (0, 1, 1) x (1, 1, 1, 12): AIC Calculated=662.061567861753
ARIMA (1, 0, 0) x (0, 0, 0, 12): AIC Calculated=1044.4404144815378
ARIMA (1, 0, 0) x (0, 0, 1, 12): AIC Calculated=843.9737863483576
ARIMA (1, 0, 0) x (0, 1, 0, 12): AIC Calculated=850.5138992750171
ARIMA (1, 0, 0) x (0, 1, 1, 12): AIC Calculated=641.5654865327434
ARIMA (1, 0, 0) x (1, 0, 0, 12): AIC Calculated=853.1076115056369
ARIMA (1, 0, 0) x (1, 0, 1, 12): AIC Calculated=845.7870028210443
ARIMA (1, 0, 0) x (1, 1, 0, 12): AIC Calculated=634.7802560813294
ARIMA (1, 0, 0) x (1, 1, 1, 12): AIC Calculated=644.5627377173593
ARIMA (1, 0, 1) x (0, 0, 0, 12): AIC Calculated=973.6145114971706
ARIMA (1, 0, 1) x (0, 0, 1, 12): AIC Calculated=780.2169927952372
ARIMA (1, 0, 1) x (0, 1, 0, 12): AIC Calculated=832.0541565169091
ARIMA (1, 0, 1) x (0, 1, 1, 12): AIC Calculated=595.2001237355449
ARIMA (1, 0, 1) x (1, 0, 0, 12): AIC Calculated=798.8792913874559
ARIMA (1, 0, 1) x (1, 0, 1, 12): AIC Calculated=781.7016954156697
ARIMA (1, 0, 1) x (1, 1, 0, 12): AIC Calculated=611.455225050599
ARIMA (1, 0, 1) x (1, 1, 1, 12): AIC Calculated=654.6174961736443
ARIMA (1, 1, 0) x (0, 0, 0, 12): AIC Calculated=976.268068494525
ARIMA (1, 1, 0) x (0, 0, 1, 12): AIC Calculated=787.0253840180616
ARIMA (1, 1, 0) x (0, 1, 0, 12): AIC Calculated=809.5346642807192
ARIMA (1, 1, 0) x (0, 1, 1, 12): AIC Calculated=1349.9241024986595
ARIMA (1, 1, 0) x (1, 0, 0, 12): AIC Calculated=786.6655684049889
ARIMA (1, 1, 0) x (1, 0, 1, 12): AIC Calculated=788.526742630533
ARIMA (1, 1, 0) x (1, 1, 0, 12): AIC Calculated=589.7095867896571
ARIMA (1, 1, 0) x (1, 1, 1, 12): AIC Calculated=1478.2045377730656
ARIMA (1, 1, 1) x (0, 0, 0, 12): AIC Calculated=958.178449378822
ARIMA (1, 1, 1) x (0, 0, 1, 12): AIC Calculated=751.8142977389185
ARIMA (1, 1, 1) x (0, 1, 0, 12): AIC Calculated=826.734397512325
ARIMA (1, 1, 1) x (0, 1, 1, 12): AIC Calculated=1488.3756756229807
ARIMA (1, 1, 1) x (1, 0, 0, 12): AIC Calculated=762.128881705292
ARIMA (1, 1, 1) x (1, 0, 1, 12): AIC Calculated=747.5853764862821
ARIMA (1, 1, 1) x (1, 1, 0, 12): AIC Calculated=590.1183471797394
ARIMA (1, 1, 1) x (1, 1, 1, 12): AIC Calculated=1220.1993736065292
```

### Fitting the time series model - ARIMA

In [120]:
```
# Find the parameters with minimal AIC value
ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'aic'])
ans_df.loc[ans_df['aic'].idxmin()]
```

Out[120]:
```
pdq         (1, 1, 0)
pdqs    (1, 1, 0, 12)
aic            589.71
Name: 54, dtype: object
```

- The output of our code suggests that `ARIMA (1, 1, 0) x (1, 1, 0, 12)` yields the lowest AIC value of `589.71`. We should therefore consider this to be optimal option out of all the models we have considered.

In [121]:
```
output = fit_SARIMAX_model(seven_zero_two, (1, 1, 0), (1, 1, 0, 12))
output.summary()
```

Out[121]:

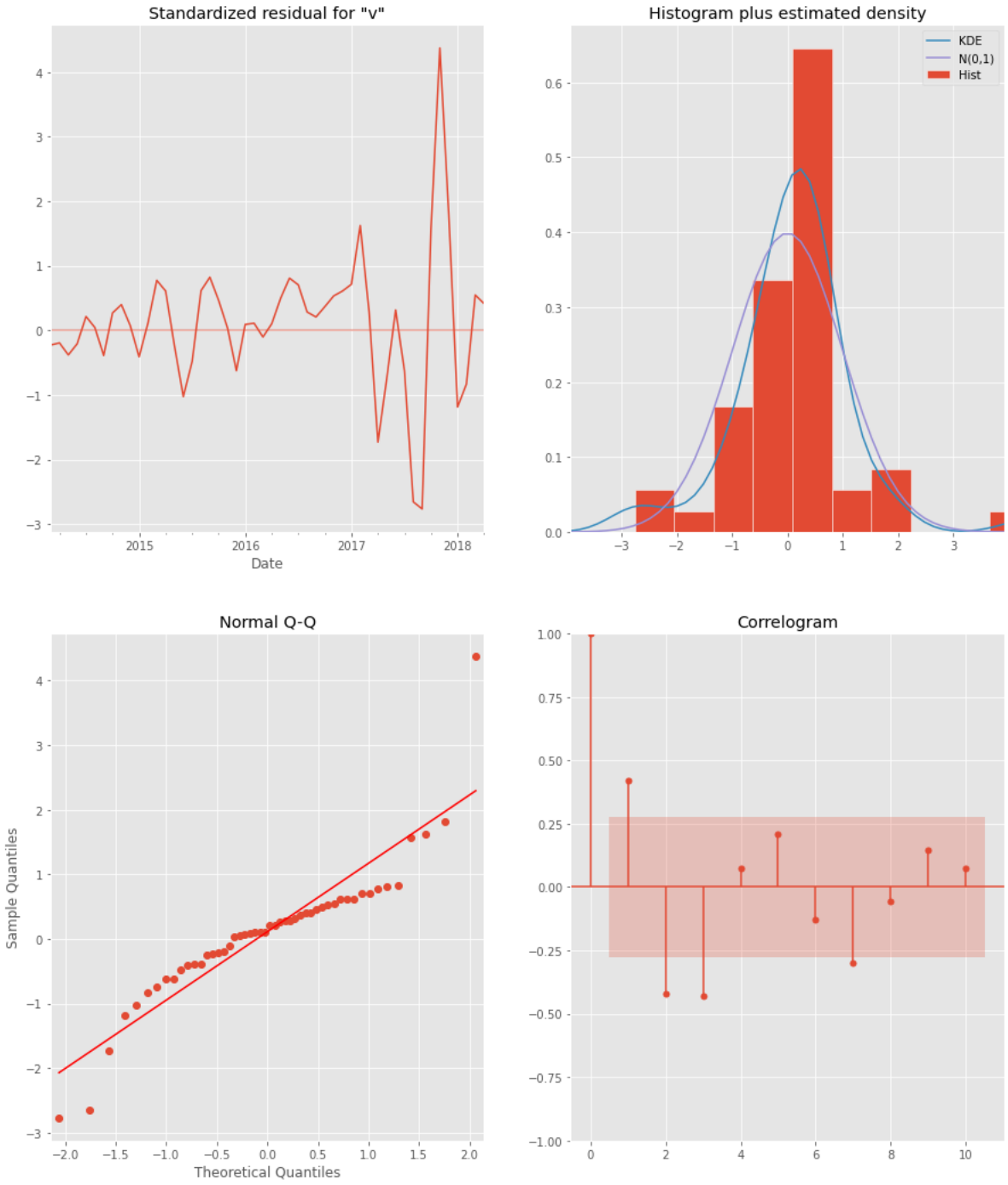
SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | value | **No. Observations:** | 76 |
| **Model:** | SARIMAX(1, 1, 0)x(1, 1, 0, 12) | **Log Likelihood** | -463.705 |
| **Date:** | Tue, 14 Feb 2023 | **AIC** | 933.410 |
| **Time:** | 12:58:11 | **BIC** | 939.146 |
| **Sample:** | 01-01-2012 | **HQIC** | 935.595 |
| | - 04-01-2018 | | |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **ar.L1** | 0.6703 | 0.060 | 11.107 | 0.000 | 0.552 | 0.789 |
| **ar.S.L12** | -0.3657 | 0.354 | -1.032 | 0.302 | -1.060 | 0.329 |
| **sigma2** | 5.823e+06 | 7.65e+05 | 7.612 | 0.000 | 4.32e+06 | 7.32e+06 |

| | | | |
|---|---|---|---|
| **Ljung-Box (L1) (Q):** | 9.36 | **Jarque-Bera (JB):** | 52.17 |
| **Prob(Q):** | 0.00 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 15.53 | **Skew:** | 0.56 |
| **Prob(H) (two-sided):** | 0.00 | **Kurtosis:** | 7.88 |

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

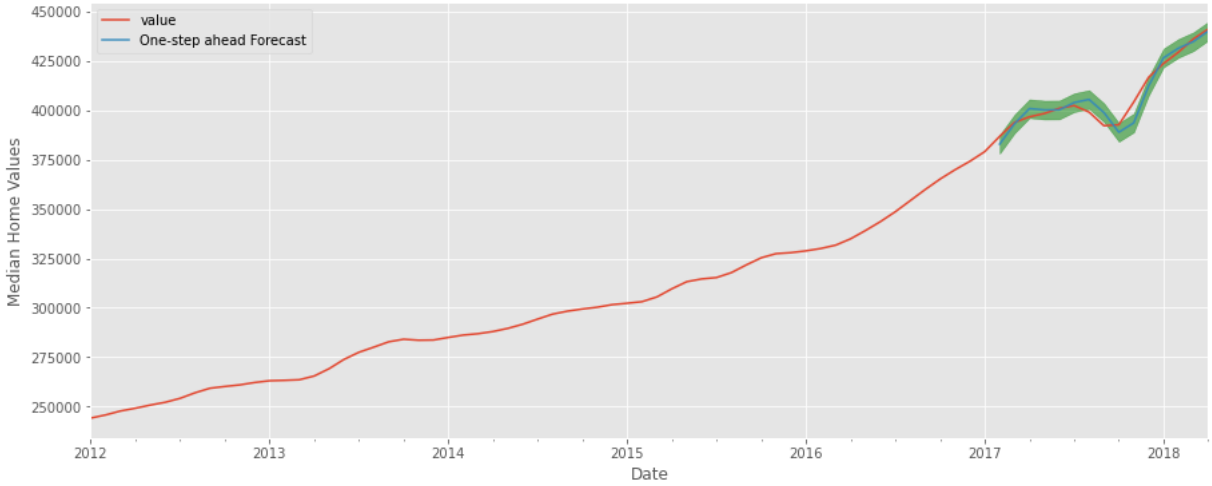



- **Standardized residual**: Shows noise rather than a pattern.
- **Histogram**: The histogram shows that the distribution appears to be pretty close to normal.
- **QQ-plot**: QQ-plot flares away from the line.
- **Correlogram**: There are four spikes outside of the shaded area. This means there could be some missed seasonality component.

### Validating the model: One-step ahead forecasting

In [122]:
```
# Get predictions starting from 01-01-2017 and calculate confidence intervals
pred = output.get_prediction(start=pd.to_datetime('2017-02-01'), dynamic=False)
pred_conf = pred.conf_int()
```

In [123]:
```
# Plot real vs predicted values along with confidence interval

rcParams['figure.figsize'] = 15, 6

# Plot observed values
ax = seven_zero_two['2012-01-01':].plot(label='observed')

# Plot predicted values
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=0.9)

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

# Set axes labels
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')
plt.legend()

plt.show()
```




- The forecasted line shows an upwards trend. The lines are close but not exact.

### Accuracy of our forecast with Root Mean Squared Error

In [124]:
```
# Get the real and predicted values
seven_zero_two_forecasted = pred.predicted_mean
seven_zero_two_truth = seven_zero_two.loc['2017-02-01':].value

# Compute the mean square error
mse = np.sqrt((seven_zero_two_forecasted - seven_zero_two_truth) ** 2).mean()
print('The Root Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

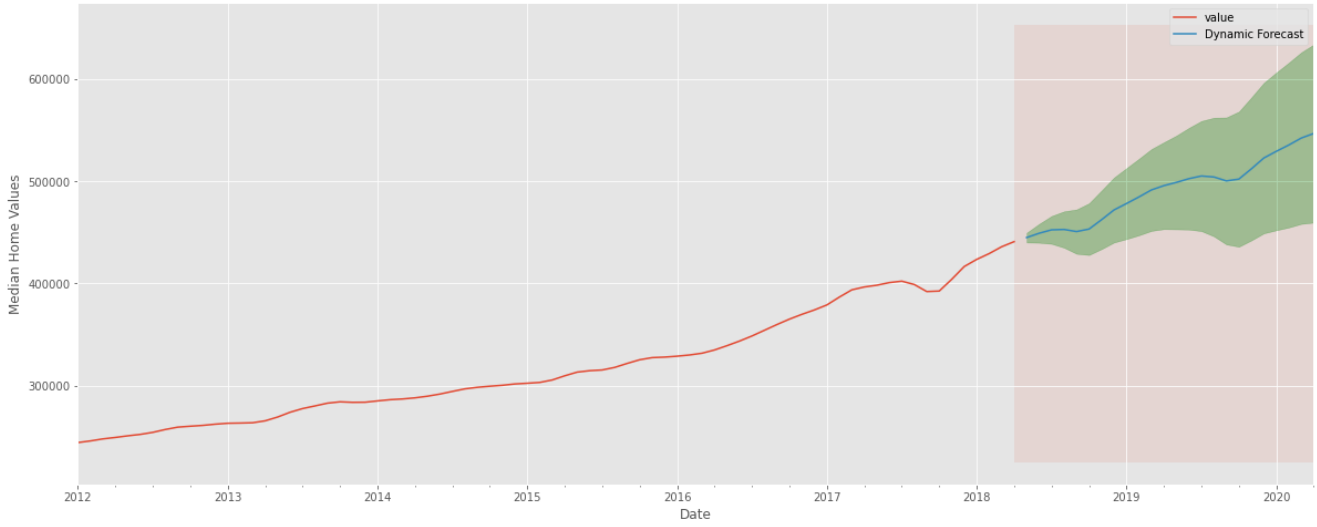
```
The Root Mean Squared Error of our forecasts is 3457.07
```

- The RMSE of 3457.07 in this case would indicate that, on average, the model's predictions for the mean value of homes in Boise, ID deviate by 3457.07 dollars from the actual values.
- AIC = 589.71

### Dynamic Forecasting

In [125]:
```
# Get dynamic predictions with confidence intervals as above
pred_dynamic = output.get_forecast(steps=24)
pred_dynamic_conf = pred_dynamic.conf_int()
```

In [126]:
```
# Plot the dynamic forecast with confidence intervals as above
# Plot the dynamic forecast with confidence intervals.

ax = seven_zero_two['2012':].plot(label='observed', figsize=(20, 8))

pred_dynamic.predicted_mean.plot(label='Dynamic Forecast', ax=ax)

ax.fill_between(pred_dynamic_conf.index,
                pred_dynamic_conf.iloc[:, 0],
                pred_dynamic_conf.iloc[:, 1], color='g', alpha=.3)

ax.fill_betweenx(ax.get_ylim(), pd.to_datetime('2020-04-01'), seven_zero_two_forecasted.index[-1], alpha=.1, zorder

ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')

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




- Our forecasts show that the time series is expected to continue increasing at a steady pace.

### Producing and visualizing forecasts

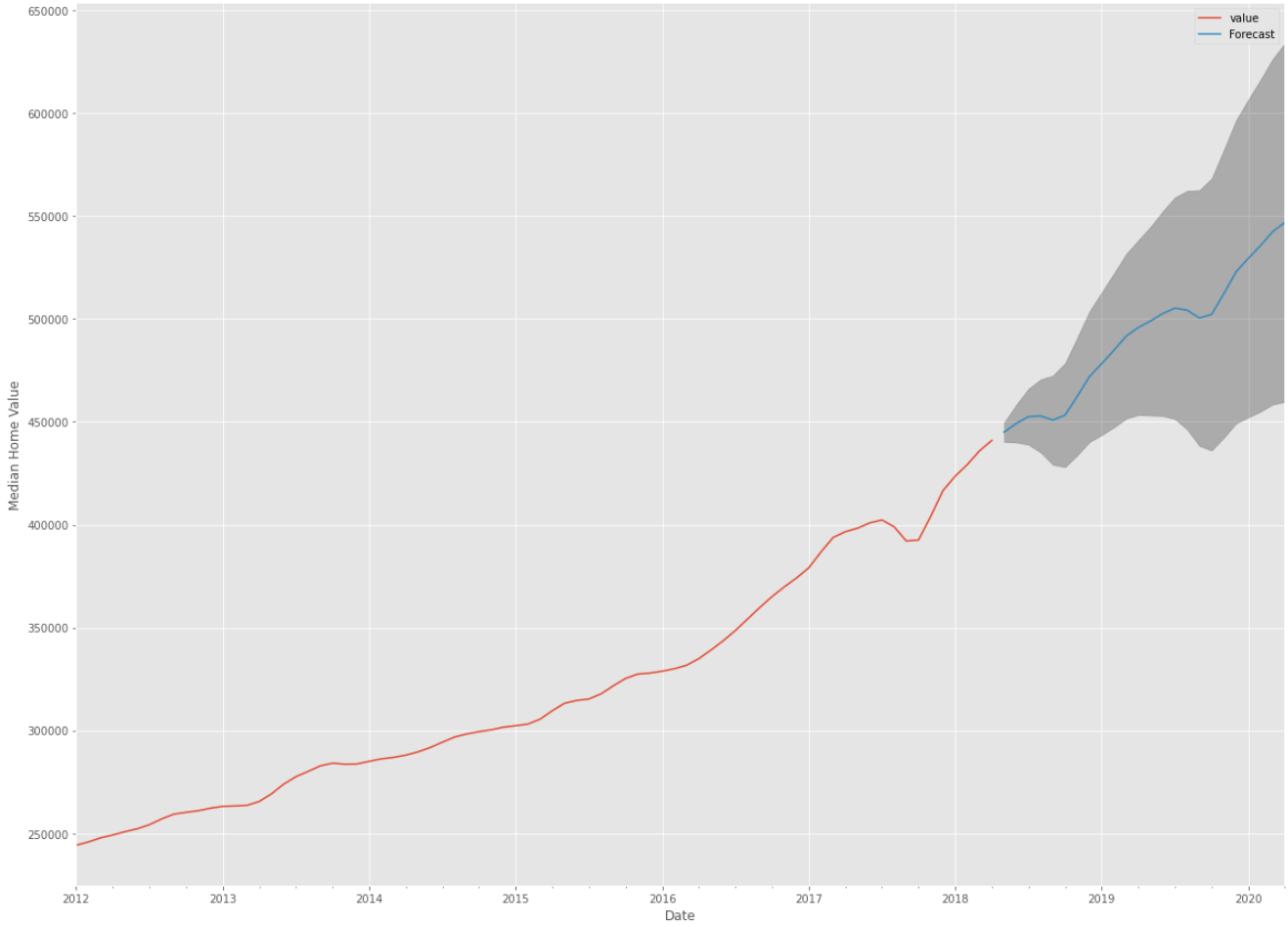
In [127]:
```
# Get forecast 24 steps ahead in future.
prediction = output.get_forecast(steps=24)

# Get confidence intervals of forecasts
pred_conf = prediction.conf_int()

pred_conf['mean'] = prediction.predicted_mean

pred_conf.head()
```

Out[127]:

| | lower value | upper value | mean |
|---|---|---|---|
| **2018-05-01** | 440261.073496 | 449720.562312 | 444990.817904 |
| **2018-06-01** | 440000.681972 | 458416.003897 | 449208.342935 |
| **2018-07-01** | 438931.468256 | 466155.167789 | 452543.318022 |
| **2018-08-01** | 435125.715234 | 470699.376898 | 452912.546066 |
| **2018-09-01** | 429176.702979 | 472546.464773 | 450861.583876 |

In [128]:

```
# Plot future predictions with confidence intervals
ax = seven_zero_two.plot(label='observed', figsize=(20, 15))
prediction.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_conf.index,
                pred_conf.iloc[:, 0],
                pred_conf.iloc[:, 1], color='k', alpha=0.25)
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Value')

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




- Our forecasts show that the time series is expected to continue increasing at a steady pace.

## Average Return on Investment (ROI)

In [129]:
```
#percentage over time is now the mean
#mean change percentage
cost = pred_conf.iloc[0]['mean']
roi = (pred_conf - cost)/ abs(cost) * 100

#pred_conf['result'] = pred_conf[''] - pred_conf['']

roi
```

Out[129]:

| | lower value | upper value | mean |
|---|---|---|---|
| **2018-05-01** | -1.062886 | 1.062886 | 0.000000 |
| **2018-06-01** | -1.121402 | 3.016958 | 0.947778 |
| **2018-07-01** | -1.361680 | 4.756132 | 1.697226 |
| **2018-08-01** | -2.216923 | 5.777323 | 1.780200 |
| **2018-09-01** | -3.553807 | 6.192408 | 1.319300 |
| **2018-10-01** | -3.818827 | 7.554791 | 1.867982 |
| **2018-11-01** | -2.521042 | 10.364116 | 3.921537 |
| **2018-12-01** | -1.070764 | 13.221792 | 6.075514 |
| **2019-01-01** | -0.315333 | 15.292547 | 7.488607 |
| **2019-02-01** | 0.521144 | 17.363539 | 8.942341 |
| **2019-03-01** | 1.483460 | 19.489591 | 10.486526 |
| **2019-04-01** | 1.879243 | 20.987046 | 11.433144 |
| **2019-05-01** | 1.832510 | 22.456012 | 12.144261 |
| **2019-06-01** | 1.773802 | 24.147831 | 12.960817 |
| **2019-07-01** | 1.439485 | 25.665299 | 13.552392 |
| **2019-08-01** | 0.286662 | 26.382529 | 13.334595 |
| **2019-09-01** | -1.493212 | 26.444643 | 12.475716 |
| **2019-10-01** | -1.998993 | 27.729358 | 12.865182 |
| **2019-11-01** | -0.615559 | 30.841692 | 15.113066 |
| **2019-12-01** | 0.945824 | 34.067680 | 17.506752 |
| **2020-01-01** | 1.616760 | 36.340141 | 18.978450 |
| **2020-02-01** | 2.236573 | 38.501545 | 20.369059 |
| **2020-03-01** | 3.023903 | 40.774492 | 21.899198 |
| **2020-04-01** | 3.318374 | 42.502777 | 22.910575 |

## Results:

In [130]:
```
results.Two_YR_ROI_Percentage[4] = roi['mean'][-1]
results.head()
```

Out[130]:

| | Zipcode | Two_YR_ROI_Percentage |
|---|---|---|
| **0** | 83709 | 14 |
| **1** | 83704 | 18 |
| **2** | 83706 | 18 |
| **3** | 83705 | 20 |
| **4** | 83702 | 22 |

- ZipCode 83702 possess the new highest 2 year ROI

## 83713

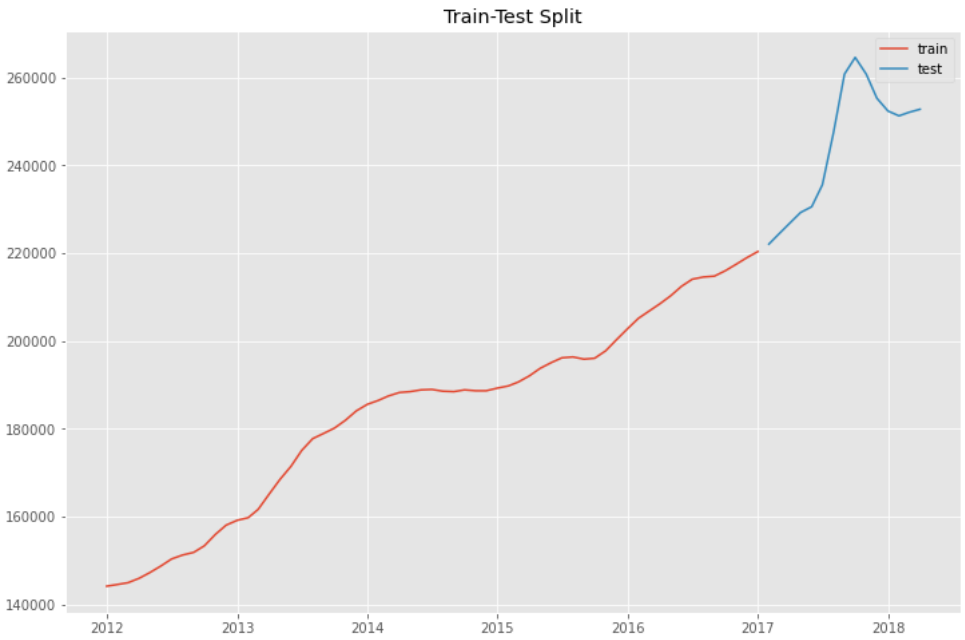
In [131]:
```
seven_one_three = boise_grouped.loc[83713]
seven_one_three.head()
```

Out[131]:

| Date | value |
|---|---|
| 2012-01-01 | 144200.0 |
| 2012-02-01 | 144600.0 |
| 2012-03-01 | 145000.0 |
| 2012-04-01 | 146000.0 |
| 2012-05-01 | 147300.0 |

In [132]:
```
# find the index which allows us to split off 20% of the data
cutoff = round(seven_one_three.shape[0]*0.8)


train = seven_one_three[:cutoff]

test = seven_one_three[cutoff:]

fig, ax = plt.subplots(figsize=(12, 8))
ax.plot(train, label='train')
ax.plot(test, label='test')
ax.set_title('Train-Test Split');
plt.legend()
```

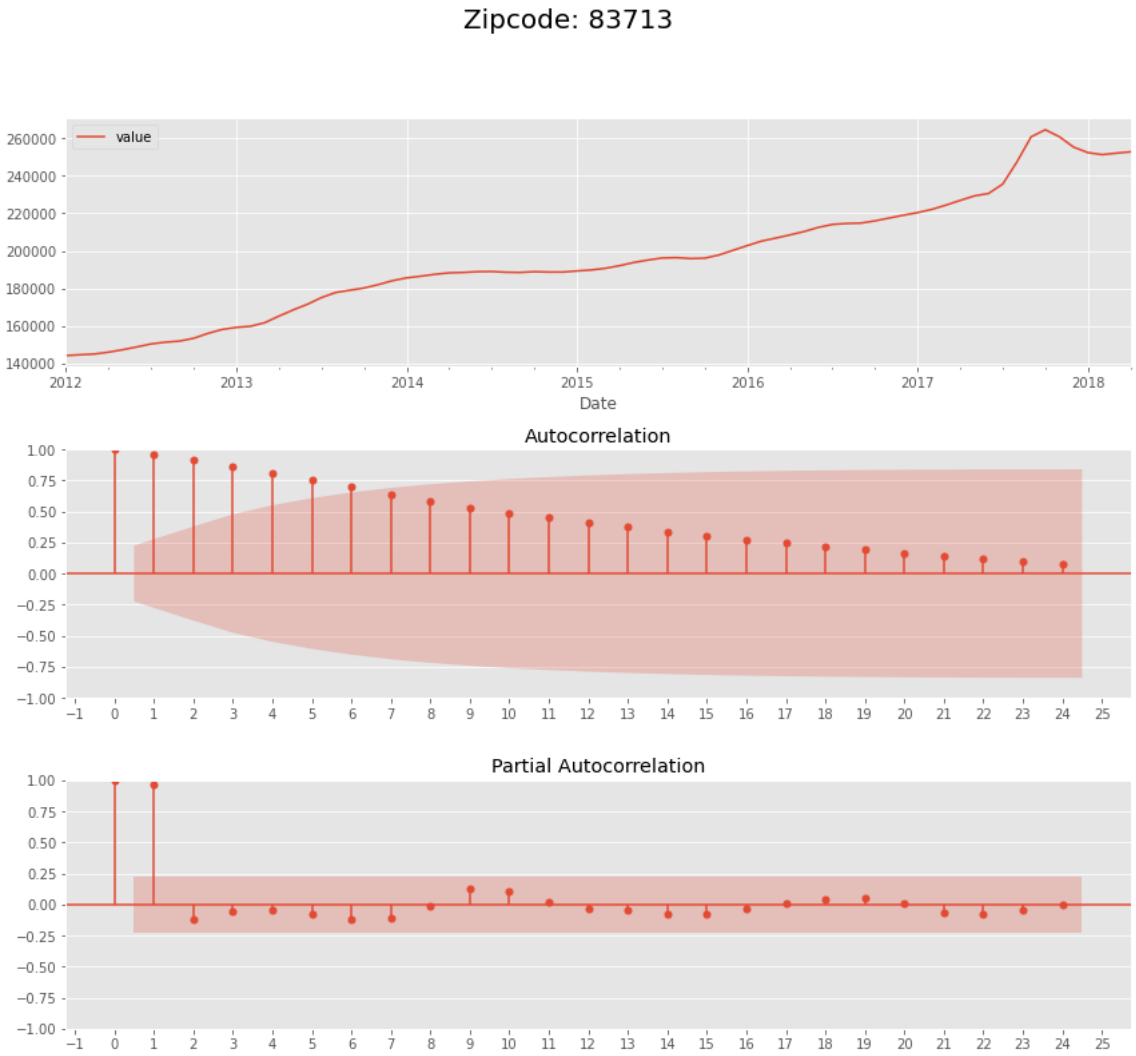
Out[132]: <matplotlib.legend.Legend at 0x7faf4b914d30>




### ACF & PACF

- The time series trends positively. Where some zipcodes showed decrease in mid 2017, this zipcode showed an increased spike in mid 2017.
- The ACF graph shows a slight decrease over time due to trend.
- The PACF graph shows a spike at 1 which would suggest an AR value of 1.

In [133]:
```
plot_acf_pacf(seven_one_three, zipcode= '83713')
plt.show()
```




### Parameter selection for the ARIMA time series model

In [134]:
```
# Define the p, d and q parameters to take any value between 0 and 2
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets (use 12 for frequency)
pdqs = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
```

In [*]:
```
ans = find_best_SARIMAX(train, pdq, pdqs)
```

```
ARIMA (0, 0, 0) x (0, 0, 0, 12): AIC Calculated=1628.1425960843765
ARIMA (0, 0, 0) x (0, 0, 1, 12): AIC Calculated=275970.8688547062
ARIMA (0, 0, 0) x (0, 1, 0, 12): AIC Calculated=1075.7792495234523
ARIMA (0, 0, 0) x (0, 1, 1, 12): AIC Calculated=796.6440676122734
ARIMA (0, 0, 0) x (1, 0, 0, 12): AIC Calculated=1028.1752237483383
ARIMA (0, 0, 0) x (1, 0, 1, 12): AIC Calculated=993.4179579789773
ARIMA (0, 0, 0) x (1, 1, 0, 12): AIC Calculated=797.3903141187725
ARIMA (0, 0, 0) x (1, 1, 1, 12): AIC Calculated=776.4722922906702
ARIMA (0, 0, 1) x (0, 0, 0, 12): AIC Calculated=1561.6861731671518
ARIMA (0, 0, 1) x (0, 0, 1, 12): AIC Calculated=253338.04676286696
ARIMA (0, 0, 1) x (0, 1, 0, 12): AIC Calculated=1064.4872980677856
ARIMA (0, 0, 1) x (0, 1, 1, 12): AIC Calculated=1158.278358913908
ARIMA (0, 0, 1) x (1, 0, 0, 12): AIC Calculated=1377.3622075095016
ARIMA (0, 0, 1) x (1, 0, 1, 12): AIC Calculated=1307.7725987404674
ARIMA (0, 0, 1) x (1, 1, 0, 12): AIC Calculated=812.0646389167508
ARIMA (0, 0, 1) x (1, 1, 1, 12): AIC Calculated=1153.033365670271
ARIMA (0, 1, 0) x (0, 0, 0, 12): AIC Calculated=1039.822529653008
ARIMA (0, 1, 0) x (0, 0, 1, 12): AIC Calculated=822.241550418185
ARIMA (0, 1, 0) x (0, 1, 0, 12): AIC Calculated=824.2157863666222
ARIMA (0, 1, 0) x (0, 1, 1, 12): AIC Calculated=1420.1184458656155
ARIMA (0, 1, 0) x (1, 0, 0, 12): AIC Calculated=832.0068310074762
ARIMA (0, 1, 0) x (1, 0, 1, 12): AIC Calculated=816.4388375788468
ARIMA (0, 1, 0) x (1, 1, 0, 12): AIC Calculated=616.302039921576
ARIMA (0, 1, 0) x (1, 1, 1, 12): AIC Calculated=1177.9454532801378
ARIMA (0, 1, 1) x (0, 0, 0, 12): AIC Calculated=956.6688508483841
ARIMA (0, 1, 1) x (0, 0, 1, 12): AIC Calculated=756.3563585979199
ARIMA (0, 1, 1) x (0, 1, 0, 12): AIC Calculated=793.4643293158201
ARIMA (0, 1, 1) x (0, 1, 1, 12): AIC Calculated=1111.2273971715215
ARIMA (0, 1, 1) x (1, 0, 0, 12): AIC Calculated=796.2276104975115
ARIMA (0, 1, 1) x (1, 0, 1, 12): AIC Calculated=757.4402455402285
```

### Fitting the time series model - ARIMA

In [*]:
```
# Find the parameters with minimal AIC value
ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'aic'])
ans_df.loc[ans_df['aic'].idxmin()]
```

- The output of our code suggests that `ARIMA (1, 1, 0) x (1, 1, 0, 12)` yields the lowest AIC value of `554.909`. We should therefore consider this to be optimal option out of all the models we have considered.

In [*]:
```
output = fit_SARIMAX_model(seven_one_three, (1, 1, 0), (1, 1, 0, 12))
output.summary()
```

- **Standardized residual**: Pattern seems to be random noise.
- **Histogram**: The histogram shows that the distribution appears to be pretty close to normal.
- **QQ-plot**: QQ-plot flares away from the line.
- **Correlogram**: There are two spikes outside of the shaded area. This means there could be some missed seasonality component.

### Validating the model: One-step ahead forecasting

In [*]:
```
# Get predictions starting from 01-01-2017 and calculate confidence intervals
pred = output.get_prediction(start=pd.to_datetime('2017-02-01'), dynamic=False)
pred_conf = pred.conf_int()
```

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

rcParams['figure.figsize'] = 15, 6

# Plot observed values
ax = seven_one_three['2012-01-01':].plot(label='observed')

# Plot predicted values
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=0.9)

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

# Set axes labels
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')
plt.legend()

plt.show()
```

- Forcasted line follows the true values till about mid 2017, then begins to vary off the line till 2018. However, it still maintains the upward trend.

### Accuracy of our forecast with Root Mean Squared Error

In [*]:
```
# Get the real and predicted values
seven_one_three_forecasted = pred.predicted_mean
seven_one_three_truth = seven_one_three.loc['2017-02-01':].value

# Compute the mean square error
mse = np.sqrt((seven_one_three_forecasted - seven_one_three_truth) ** 2).mean()
print('The Root Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

- The RMSE of 2711.26 in this case would indicate that, on average, the model's predictions for the mean value of homes in Boise, ID deviate by 2711.26 dollars from the actual values.
- AIC = 554.909

### Dynamic Forecasting

In [*]:
```
# Get dynamic predictions with confidence intervals as above
pred_dynamic = output.get_forecast(steps=24)
pred_dynamic_conf = pred_dynamic.conf_int()
```

In [*]:
```
# Plot the dynamic forecast with confidence intervals as above
# Plot the dynamic forecast with confidence intervals.

ax = seven_one_three['2012':].plot(label='observed', figsize=(20, 8))

pred_dynamic.predicted_mean.plot(label='Dynamic Forecast', ax=ax)

ax.fill_between(pred_dynamic_conf.index,
                pred_dynamic_conf.iloc[:, 0],
                pred_dynamic_conf.iloc[:, 1], color='g', alpha=.3)

ax.fill_betweenx(ax.get_ylim(), pd.to_datetime('2020-04-01'), seven_one_three_forecasted.index[-1], alpha=.1, zorde

ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')

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

- Our forecasts show that the time series is expected to continue increasing at a steady pace.

### Producing and visualizing forecasts

In [*]:
```
# Get forecast 24 steps ahead in future.
prediction = output.get_forecast(steps=24)

# Get confidence intervals of forecasts
pred_conf = prediction.conf_int()

pred_conf['mean'] = prediction.predicted_mean

pred_conf.head()
```

In [*]:
```
# Plot future predictions with confidence intervals
ax = seven_one_three.plot(label='observed', figsize=(20, 15))
prediction.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_conf.index,
                pred_conf.iloc[:, 0],
                pred_conf.iloc[:, 1], color='k', alpha=0.25)
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Value')

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

- - Our forecasts show that the time series is expected to continue increasing at a steady pace.

## Average Return on Investment (ROI)

In [*]:
```
#percentage over time is now the mean
#mean change percentage
cost = pred_conf.iloc[0]['mean']
roi = (pred_conf - cost)/ abs(cost) * 100

#pred_conf['result'] = pred_conf[''] - pred_conf['']

roi
```

## Results:

In [*]:
```
results.Two_YR_ROI_Percentage[5] = roi['mean'][-1]
results
```

## 83703

In [*]:
```
seven_zero_three = boise_grouped.loc[83703]
seven_zero_three.head()
```

In [*]:
```
# find the index which allows us to split off 20% of the data
cutoff = round(seven_zero_three.shape[0]*0.8)


train = seven_zero_three[:cutoff]

test = seven_zero_three[cutoff:]

fig, ax = plt.subplots(figsize=(12, 8))
ax.plot(train, label='train')
ax.plot(test, label='test')
ax.set_title('Train-Test Split')
plt.legend();
```

### ACF & PACF

- The time series trends positively.
- The ACF graph shows a slight decrease over time due to trend.
- The PACF graph shows a spike at 1 which would suggest an AR value of 1.

In [*]:
```
plot_acf_pacf(seven_zero_three, zipcode= '83703')
plt.show()
```

### Parameter selection for the ARIMA time series model

In [*]:
```
# Define the p, d and q parameters to take any value between 0 and 2
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets (use 12 for frequency)
pdqs = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
```

In [*]:
```
ans = find_best_SARIMAX(train, pdq, pdqs)
```

### Fitting the time series model - ARIMA

In [*]:
```
# Find the parameters with minimal AIC value
ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'aic'])
ans_df.loc[ans_df['aic'].idxmin()]
```

- The output of our code suggests that `ARIMA (1, 1, 0) x (1, 1, 0, 12)` yields the lowest AIC value of `544.537`. We should therefore consider this to be optimal option out of all the models we have considered.

In [*]:
```
output = fit_SARIMAX_model(seven_zero_three, (1, 1, 0), (1, 1, 0, 12))
output.summary()
```

- **Standardized residual**: Pattern seems to be random.
- **Histogram**: The histogram shows that the distribution appears to be pretty close to normal.
- **QQ-plot**: QQ-plot flares away from the line.
- **Correlogram**: There are three spikes outside of the shaded area. This means there could be some missed seasonality component.

### Validating the model: One-step ahead forecasting

In [*]:
```
# Get predictions starting from 01-01-2017 and calculate confidence intervals
pred = output.get_prediction(start=pd.to_datetime('2017-02-01'), dynamic=False)
pred_conf = pred.conf_int()
```

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

rcParams['figure.figsize'] = 15, 6

# Plot observed values
ax = seven_zero_three['2012-01-01':].plot(label='observed')

# Plot predicted values
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=0.9)

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

# Set axes labels
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')
plt.legend()

plt.show()
```

- The forecast deviates from the true value line slightly. But continues an upward positive trend.

### Accuracy of our forecast with Root Mean Squared Error

In [*]:
```
# Get the real and predicted values
seven_zero_three_forecasted = pred.predicted_mean
seven_zero_three_truth = seven_zero_three.loc['2017-02-01':].value

# Compute the mean square error
mse = np.sqrt((seven_zero_three_forecasted - seven_zero_three_truth) ** 2).mean()
print('The Root Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

- The RMSE of 1495.92 in this case would indicate that, on average, the model's predictions for the mean value of homes in Boise, ID deviate by 1495.92 dollars from the actual values.

- AIC = 544.537

### Dynamic Forecasting

In [*]:
```
# Get dynamic predictions with confidence intervals as above
pred_dynamic = output.get_forecast(steps=24)
pred_dynamic_conf = pred_dynamic.conf_int()
```

In [*]:
```
# Plot the dynamic forecast with confidence intervals as above
# Plot the dynamic forecast with confidence intervals.

ax = seven_zero_three['2012':].plot(label='observed', figsize=(20, 8))

pred_dynamic.predicted_mean.plot(label='Dynamic Forecast', ax=ax)

ax.fill_between(pred_dynamic_conf.index,
                pred_dynamic_conf.iloc[:, 0],
                pred_dynamic_conf.iloc[:, 1], color='g', alpha=.3)

ax.fill_betweenx(ax.get_ylim(), pd.to_datetime('2020-04-01'), seven_zero_three_forecasted.index[-1], alpha=.1, zord

ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')

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

- Our forecasts show that the time series is expected to continue increasing at a steady pace.

### Producing and visualizing forecasts

In [*]:
```
# Get forecast 24 steps ahead in future.
prediction = output.get_forecast(steps=24)

# Get confidence intervals of forecasts
pred_conf = prediction.conf_int()

pred_conf['mean'] = prediction.predicted_mean

pred_conf.head()
```

In [*]:
```
# Plot future predictions with confidence intervals
ax = seven_zero_three.plot(label='observed', figsize=(20, 15))
prediction.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_conf.index,
                pred_conf.iloc[:, 0],
                pred_conf.iloc[:, 1], color='k', alpha=0.25)
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Value')

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

- Our forecasts show that the time series is expected to continue increasing at a steady pace.

## Average Return on Investment (ROI)

In [*]:
```
#percentage over time is now the mean
#mean change percentage
cost = pred_conf.iloc[0]['mean']
roi = (pred_conf - cost)/ abs(cost) * 100

#pred_conf['result'] = pred_conf[''] - pred_conf['']

roi
```

## Results:

In [*]:
```
results.Two_YR_ROI_Percentage[6] = roi['mean'][-1]
results
```

- ZipCode 83703 now has the highest 2 year ROI.

## 83716

In [*]:
```
seven_one_six = boise_grouped.loc[83716]
seven_one_six.head()
```

In [*]:
```
# find the index which allows us to split off 20% of the data
cutoff = round(seven_one_six.shape[0]*0.8)


train = seven_one_six[:cutoff]

test = seven_one_six[cutoff:]

fig, ax = plt.subplots(figsize=(12, 8))
ax.plot(train, label='train')
ax.plot(test, label='test')
ax.set_title('Train-Test Split');
plt.legend()
```

### ACF & PACF

- The time series trends positively.
- The ACF graph shows a slight decrease over time due to trend.
- The PACF graph shows a spike at 1 which would suggest an AR value of 1.

In [*]:
```
plot_acf_pacf(seven_one_six, zipcode= '83716')
plt.show()

```

### Parameter selection for the ARIMA time series model

In [*]:
```
# Define the p, d and q parameters to take any value between 0 and 2
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets (use 12 for frequency)
pdqs = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
```

In [*]:
```
ans = find_best_SARIMAX(train, pdq, pdqs)
```

### Fitting the time series model - ARIMA

In [*]:
```
# Find the parameters with minimal AIC value
ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'aic'])
ans_df.loc[ans_df['aic'].idxmin()]
```

The output of our code suggests that `ARIMA (1, 1, 0) x (1, 1, 0, 12)` yields the lowest AIC value of `582.942` . We should therefore consider this to be optimal option out of all the models we have considered.

In [*]:
```
output = fit_SARIMAX_model(seven_one_six, (1, 1, 0), (1, 1, 0, 12))
output.summary()
```

- **Standardized residual**: Pattern appears to be random.
- **Histogram**: The histogram shows that the distribution appears to be pretty close to normal.
- **QQ-plot**: QQ-plot flares slightly away from the line.
- **Correlogram**: There is one spike outside of the shaded area. This means there could be some missed seasonality component.

### Validating the model: One-step ahead forecasting

In [*]:
```
# Get predictions starting from 01-01-2017 and calculate confidence intervals
pred = output.get_prediction(start=pd.to_datetime('2017-02-01'), dynamic=False)
pred_conf = pred.conf_int()
```

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

rcParams['figure.figsize'] = 15, 6

# Plot observed values
ax = seven_one_six['2012-01-01':].plot(label='observed')

# Plot predicted values
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=0.9)

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

# Set axes labels
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')
plt.legend()

plt.show()
```

- The forecasted line varies off the true line slightly.

### Accuracy of our forecast with Root Mean Squared Error

In [*]:
```
# Get the real and predicted values
seven_one_six_forecasted = pred.predicted_mean
seven_one_six_truth = seven_one_six.loc['2017-02-01':].value

# Compute the mean square error
mse = np.sqrt((seven_one_six_forecasted - seven_one_six_truth) ** 2).mean()
print('The Root Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

- The RMSE of 1309.59 in this case would indicate that, on average, the model's predictions for the mean value of homes in Boise, ID deviate by 1309.59 dollars from the actual values.
- AIC = 582.942

### Dynamic Forecasting

In [*]:
```
# Get dynamic predictions with confidence intervals as above
pred_dynamic = output.get_forecast(steps=24)
pred_dynamic_conf = pred_dynamic.conf_int()
```

In [*]:
```
# Plot the dynamic forecast with confidence intervals as above
# Plot the dynamic forecast with confidence intervals.

ax = seven_one_six['2012':].plot(label='observed', figsize=(20, 8))

pred_dynamic.predicted_mean.plot(label='Dynamic Forecast', ax=ax)

ax.fill_between(pred_dynamic_conf.index,
                pred_dynamic_conf.iloc[:, 0],
                pred_dynamic_conf.iloc[:, 1], color='g', alpha=.3)

ax.fill_betweenx(ax.get_ylim(), pd.to_datetime('2020-04-01'), seven_one_six_forecasted.index[-1], alpha=.1, zorder=

ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')

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

- Our forecasts show that the time series is expected to continue increasing at a steady pace.

### Producing and visualizing forecasts

In [*]:
```
# Get forecast 24 steps ahead in future.
prediction = output.get_forecast(steps=24)

# Get confidence intervals of forecasts
pred_conf = prediction.conf_int()

pred_conf['mean'] = prediction.predicted_mean

pred_conf.head()
```

In [*]:
```
# Plot future predictions with confidence intervals
ax = seven_one_six.plot(label='observed', figsize=(20, 15))
prediction.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_conf.index,
                pred_conf.iloc[:, 0],
                pred_conf.iloc[:, 1], color='k', alpha=0.25)
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Value')

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

- Our forecasts show that the time series is expected to continue increasing at a steady pace.

## Average Return on Investment (ROI)

In [*]:
```
#percentage over time is now the mean
#mean change percentage
cost = pred_conf.iloc[0]['mean']
roi = (pred_conf - cost)/ abs(cost) * 100

#pred_conf['result'] = pred_conf[''] - pred_conf['']

roi
```

## Results:

In [*]:
```
results.Two_YR_ROI_Percentage[7] = roi['mean'][-1]
results
```

## 83712

In [*]:
```
seven_one_two = boise_grouped.loc[83712]
seven_one_two.head()
```

In [*]:
```
# find the index which allows us to split off 20% of the data
cutoff = round(seven_one_two.shape[0]*0.8)


train = seven_one_two[:cutoff]

test = seven_one_two[cutoff:]

fig, ax = plt.subplots(figsize=(12, 8))
ax.plot(train, label='train')
ax.plot(test, label='test')
ax.set_title('Train-Test Split');
plt.legend()
```

### ACF & PACF

- The time series trends positively.
- The ACF graph shows a slight decrease over time due to trend.
- The PACF graph shows a spike at 1 which would suggest an AR value of 1.

In [*]:
```
plot_acf_pacf(seven_one_two, zipcode = '83712')
plt.show()
```

### Parameter selection for the ARIMA time series model

In [*]:
```
# Define the p, d and q parameters to take any value between 0 and 2
p = d = q = range(0, 2)

# Generate all different combinations of p, q and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, q and q triplets (use 12 for frequency)
pdqs = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
```

In [*]:
```
ans = find_best_SARIMAX(train, pdq, pdqs)
```

### Fitting the time series model - ARIMA

In [*]:
```
# Find the parameters with minimal AIC value
ans_df = pd.DataFrame(ans, columns=['pdq', 'pdqs', 'aic'])
ans_df.loc[ans_df['aic'].idxmin()]
```

The output of our code suggests that `ARIMA (1, 1, 1) x (1, 1, 0, 12)` yields the lowest AIC value of `603.165`. We should therefore consider this to be optimal option out of all the models we have considered.

In [*]:
```
output = fit_SARIMAX_model(seven_one_two, (1, 1, 1), (1, 1, 0, 12))
output.summary()
```

- **Standardized residual**: The pattern appears to be random.
- **Histogram**: The histogram shows that the distribution appears to be pretty close to normal.
- **QQ-plot**: QQ-plot flares away from the line.
- **Correlogram**: The points in the correlogram fall inside the box and do not indidcate a missed seasonality component.

### Validating the model: One-step ahead forecasting

In [*]:
```
# Get predictions starting from 01-01-2017 and calculate confidence intervals
pred = output.get_prediction(start=pd.to_datetime('2017-02-01'), dynamic=False)
pred_conf = pred.conf_int()
```

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

rcParams['figure.figsize'] = 15, 6

# Plot observed values
ax = seven_one_two['2012-01-01':].plot(label='observed')

# Plot predicted values
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=0.9)

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

# Set axes labels
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')
plt.legend()

plt.show()
```

### Accuracy of our forecast with Root Mean Squared Error

In [*]:
```
# Get the real and predicted values
seven_one_two_forecasted = pred.predicted_mean
seven_one_two_truth = seven_one_two.loc['2017-02-01':].value

# Compute the mean square error
mse = np.sqrt((seven_one_two_forecasted - seven_one_two_truth) ** 2).mean()
print('The Root Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

- The RMSE of 2530.95 in this case would indicate that, on average, the model's predictions for the mean value of homes in Boise, ID deviate by 2530.95 dollars from the actual values.
- AIC = 603.165

### Dynamic Forecasting

In [*]:
```
# Get dynamic predictions with confidence intervals as above
pred_dynamic = output.get_forecast(steps=24)
pred_dynamic_conf = pred_dynamic.conf_int()
```

In [*]:
```
# Plot the dynamic forecast with confidence intervals as above
# Plot the dynamic forecast with confidence intervals.

ax = seven_one_two['2012':].plot(label='observed', figsize=(20, 8))

pred_dynamic.predicted_mean.plot(label='Dynamic Forecast', ax=ax)

ax.fill_between(pred_dynamic_conf.index,
                pred_dynamic_conf.iloc[:, 0],
                pred_dynamic_conf.iloc[:, 1], color='g', alpha=.3)

ax.fill_betweenx(ax.get_ylim(), pd.to_datetime('2020-04-01'), seven_one_two_forecasted.index[-1], alpha=.1, zorder=

ax.set_xlabel('Date')
ax.set_ylabel('Median Home Values')

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

### Producing and visualizing forecasts

In [*]:
```
# Get forecast 24 steps ahead in future.
prediction = output.get_forecast(steps=24)

# Get confidence intervals of forecasts
pred_conf = prediction.conf_int()

pred_conf['mean'] = prediction.predicted_mean

pred_conf.head()
```

In [*]:
```
# Plot future predictions with confidence intervals
ax = seven_one_two.plot(label='observed', figsize=(20, 15))
prediction.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_conf.index,
                pred_conf.iloc[:, 0],
                pred_conf.iloc[:, 1], color='k', alpha=0.25)
ax.set_xlabel('Date')
ax.set_ylabel('Median Home Value')

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

## Average Return on Investment (ROI)

In [*]:
```
#percentage over time is now the mean
#mean change percentage
cost = pred_conf.iloc[0]['mean']
roi = (pred_conf - cost)/ abs(cost) * 100

#pred_conf['result'] = pred_conf[''] - pred_conf['']

roi
```

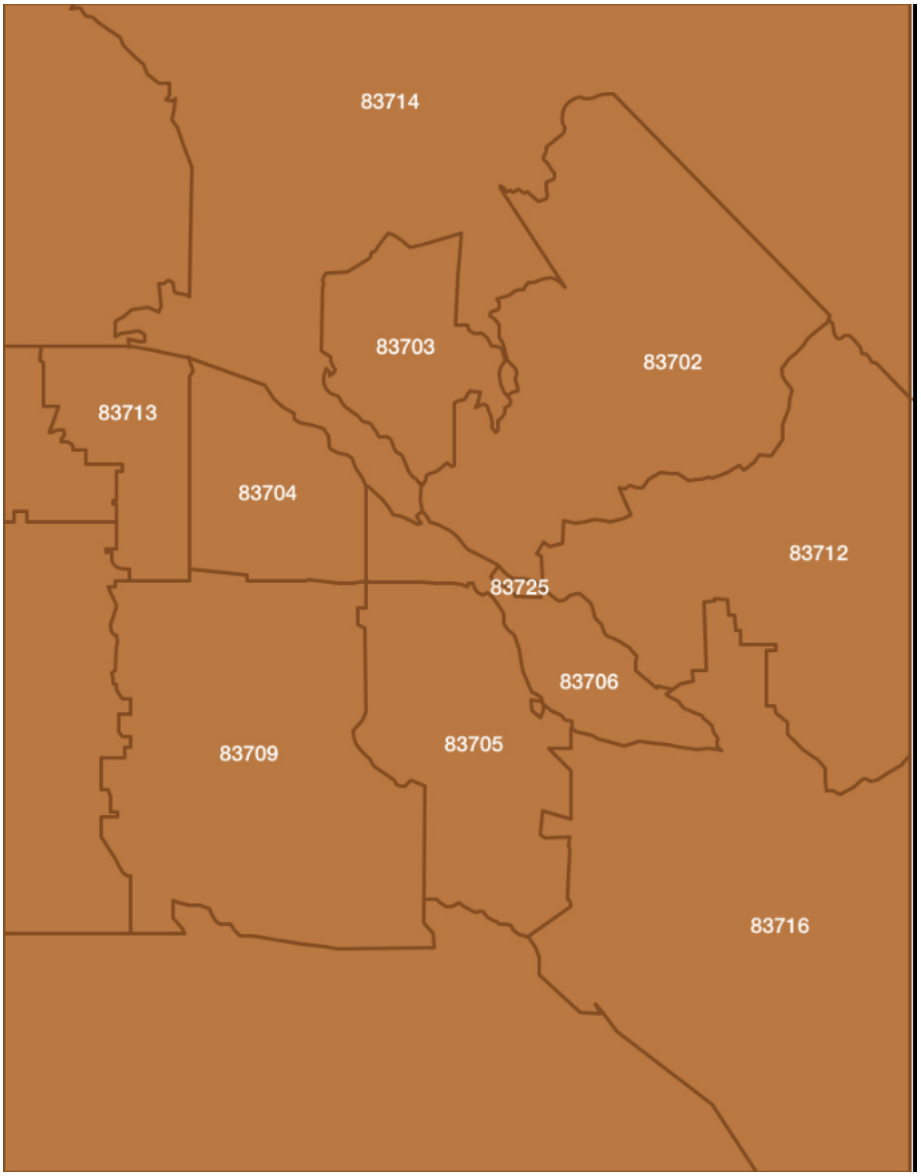
## Results:

In [*]:
```
results.Two_YR_ROI_Percentage[8] = roi['mean'][-1]
results
```

## Top 5 ZipCodes

In [*]:
```
top_5 = results.sort_values(by='Two_YR_ROI_Percentage', ascending=False).head()
top_5.reset_index(inplace=True)
top_5
```

### Image of Boise ZipCodes




# Conclusion

## ZipCode Recommendations

- The ROI percentage represents the appreciation in property value of a home in Boise, ID over a 2 year span. I believe even though we are just analyzing the ROI in the short term, the value of homes in Boise will continue to rise.
- Though all top 5 zip codes have forecasted to be profitable, I recommend investing in properties in the 83703 first. The inital invest is low and the profit margin is excellent, and doesn't show signs of slowing down.

1. **83703**
2. **83702**
3. **83705**
4. **83704**
5. **83706**

# Future Work

- In the future, I would like to look into other cities in Idaho to see how they compare to the state's capitol. Additionally, future work could include using a greater range than 0 and 2 on pdq parameters in SARIMAX modeling.