1 Zipcodes for Millennials: Where You Should Invest in a Home

As some of the youngest millennials begin to turn 30, the prospect of home ownership is becoming more of a priority for investment. According to the Zillow Consumer Housing Trends Report for 2021, 26 percent of homebuyers are between the ages of 30 and 39. The majority of millennials fall within this age range, with even the youngest millennials turning 30 each year.

This project seeks to identify the top five zipcodes for millennials to invest in a home.

2 Project Summary

This time series modeling project seeks to solve the business problem of recommending the top five best zipcodes to first time millennial home buyers. The data used in this modeling comes from the Zillow Research Page and includes monthly median home values for over 14,000 zipcodes in the United States dating back to 1996. This data is appropriate and well-suited to address the business problem because it comes from a reputable source and includes median home values dating back to 1996, allowing for vast historical trends.

In preparing the data, I began by calculating the cumulative Return on Investment based on 1996 and 2018 values as well as the ROI for the most recent five years, based on 2012 and 2018. I chose to focus on ROI as the metric to ascertain "best" zipcode because it's a common metric the clients will understand and dilutes any influence cost of living may have on the selection of zipcodes. For data preparation, I used `pandas` , `matplotlib` , and `datetime` .

For the time series modeling, I used `statsmodel` SARIMAX for SARIMA modeling and `pmdarima.arima` to use `auto_arima` in order to calculate the best pdq and pdqs values.

For the SARIMA modeling, the pdq and pdqs values were selected based on which combination had the lowest AIC of all the combinations; in addition, I set stationarity and invertibility both to false (since the SARIMA model already takes into account seasonality, we do not need to enforce stability).

The model that performed the best was the SARIMA model for Katy, TX (77449) with a Mean Absolute Error of 4466, by far the smallest MAE of the five models conducted. In validating the predictions, I created visualizations of the validation set and predicted values.

3 Business Understanding

My clients are first-time homebuyers who are millennials. I know my clients have incomes in the average, middle-of-the-road range and seeking to know what return on their investment may they see in the first two years.

To measure "best" zipcode based on our client profile, I will evaluate long-term (22 years) and short-term (5 years) return on investment and will forecast median home values out two years.

I will provide clients a range of recommendations, from more safe investments (with less variation in forecast) to more risky investments (with more variation in forecast). Further, I'll provide clients a measure error, so that they may make as informed decision as they can.

3.1 Data Understanding

The data used in this time series modeling comes from the [Zillow Research Page (https://www.zillow.com/research/data/)](https://www.zillow.com/research/data/) and includes monthly median home values for over 14,000 zipcodes in the United States dating back to 1996.

3.2 Evaluation of "Best" Zipcodes

To narrow down our list of prospective zipcodes, I focus on cumulative ROI (over the 22 years of the full dataset) and the ROI of the most recent five years.

When making predictions using the model, I evaluate its performance using the mean absolute error, so that I can convey to the clients the dollar amount predictions may be off by.

Lastly, I also include the median home value as a data point to consider for investment as homebuyers will having varying levels of cash on hand with which to make their investment.

4 Data Preparation

4.1 Load Libraries

In [1]:
```python
# Basics
import pandas as pd
import numpy as np

# Visualization
import matplotlib.pyplot as plt
plt.style.use('ggplot')

# Modeling
import statsmodels.api as sm
import itertools
from sklearn.metrics import mean_absolute_error
from statsmodels.tsa.statespace.sarimax import SARIMAX
from pmdarima.arima import auto_arima

# Warnings
import warnings
warnings.filterwarnings("ignore")
```
executed in 7.40s, finished 20:33:16 2022-06-29

4.2 Load Housing Data

In [2]:
```python
df = pd.read_csv('zillow_data.csv')
```
executed in 1.15s, finished 20:33:18 2022-06-29

In [3]:
```python
df.head()
```
executed in 151ms, finished 20:33:18 2022-06-29

Out[3]:

	RegionID	RegionName	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	1996-06	...	2017-07	2017-08	2017-0
0	84654	60657	Chicago	IL	Chicago	Cook	1	334200.0	335400.0	336500.0	...	1005500	1007500	100780
1	90668	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	235700.0	236900.0	236700.0	...	308000	310000	31250
2	91982	77494	Katy	TX	Houston	Harris	3	210400.0	212200.0	212200.0	...	321000	320600	32020
3	84616	60614	Chicago	IL	Chicago	Cook	4	498100.0	500900.0	503100.0	...	1289800	1287700	128740
4	93144	79936	El Paso	TX	El Paso	El Paso	5	77300.0	77300.0	77300.0	...	119100	119400	12000

5 rows × 272 columns

In [4]:
```python
# RegionID refers to zipcode, so rename for convenience

df = df.drop('RegionID', axis = 1)
df = df.rename({'RegionName': 'zipcode'}, axis = 1)
```
executed in 120ms, finished 20:33:18 2022-06-29

In [5]:
```python
df.head()
```
executed in 84ms, finished 20:33:18 2022-06-29

Out[5]:

	zipcode	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	1996-06	1996-07	...	2017-07	2017-08	2017-09	20
0	60657	Chicago	IL	Chicago	Cook	1	334200.0	335400.0	336500.0	337600.0	...	1005500	1007500	1007800	10
1	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	235700.0	236900.0	236700.0	235400.0	...	308000	310000	312500	3
2	77494	Katy	TX	Houston	Harris	3	210400.0	212200.0	212200.0	210700.0	...	321000	320600	320200	3
3	60614	Chicago	IL	Chicago	Cook	4	498100.0	500900.0	503100.0	504600.0	...	1289800	1287700	1287400	12
4	79936	El Paso	TX	El Paso	El Paso	5	77300.0	77300.0	77300.0	77300.0	...	119100	119400	120000	1

5 rows × 271 columns

4.3 Exploratory Data Analysis

4.3.1 Preprocessing

To help refine our dataset, I want to compute the cumulative percent change of home values for the whole dataset, from 1996 to 2018 as well as the most recent percent change in home values over the most recent five years (2012-2018).

In [6]:
```
# Find ROI - first, make copy of df
df1 = df.copy()

# Add column with percent change between 2018 and 1996 median value
df1['Cumulative Percent Change'] = (df1['2018-04'] / df1['1996-04']) * 100

# Add column with percent change between 2018 and 2012 median value
df1['5-year Percent Change'] = (df1['2018-04'] / df1['2012-04']) * 100

df1.head()
```
executed in 164ms, finished 20:33:18 2022-06-29

Out[6]:

	zipcode	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	1996-06	1996-07	...	2017-09	2017-10	2017-11	20
0	60657	Chicago	IL	Chicago	Cook	1	334200.0	335400.0	336500.0	337600.0	...	1007800	1009600	1013300	10
1	75070	McKinney	TX	Dallas-Fort Worth	Collin	2	235700.0	236900.0	236700.0	235400.0	...	312500	314100	315000	3
2	77494	Katy	TX	Houston	Harris	3	210400.0	212200.0	212200.0	210700.0	...	320200	320400	320800	3
3	60614	Chicago	IL	Chicago	Cook	4	498100.0	500900.0	503100.0	504600.0	...	1287400	1291500	1296600	12
4	79936	El Paso	TX	El Paso	El Paso	5	77300.0	77300.0	77300.0	77300.0	...	120000	120300	120300	1

5 rows × 273 columns

In [7]:
```
df2 = df1.copy()
```
executed in 45ms, finished 20:33:18 2022-06-29

I created individual dataframes for each computed value, so that I could trim each dataframe to only include the Interquartile Range.

I am only including data values within the IQR to meet the business needs of millennials who are first-time homebuyers. Home values above the 75th percentile and below the 25th percentile will not be appropriate homes to include for prospects.

In [8]:
```
df3 = df2[['zipcode',
           'Metro',
           'Cumulative Percent Change']]

df4 = df2[['zipcode',
           'Metro',
           '5-year Percent Change']]
```
executed in 209ms, finished 20:33:18 2022-06-29

In [9]:
```
df3.head(10)
```
executed in 31ms, finished 20:33:19 2022-06-29

Out[9]:

	zipcode	Metro	Cumulative Percent Change
0	60657	Chicago	308.378217
1	75070	Dallas-Fort Worth	136.529487
2	77494	Houston	156.796578
3	60614	Chicago	262.397109
4	79936	El Paso	157.179819
5	77084	Houston	172.947368
6	10467	New York	273.315893
7	60640	Chicago	361.570439
8	77449	Houston	184.696017
9	94109	San Francisco	497.845953

In [10]:
```
Q1 = df3.quantile(0.25)
Q3 = df3.quantile(0.75)
IQR = Q3 - Q1

df3_trimmed = df3[~((df3 < (Q1 - 1.5 * IQR)) |(df3 > (Q3 + 1.5 * IQR))).any(axis=1)]
cum_pct_top10 = df3_trimmed.head(10).sort_values('Cumulative Percent Change', ascending=False)
cum_pct_top10
```
executed in 66ms, finished 20:33:19 2022-06-29

Out[10]:

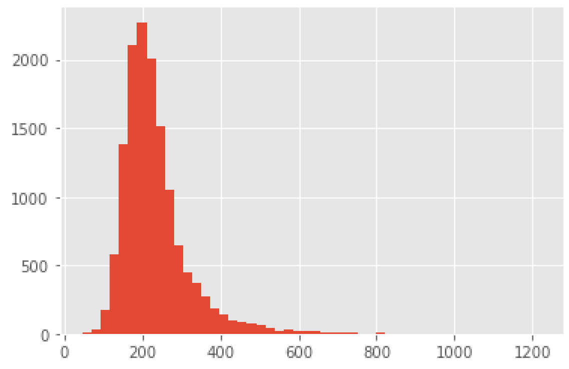
	zipcode	Metro	Cumulative Percent Change
7	60640	Chicago	361.570439
0	60657	Chicago	308.378217
6	10467	New York	273.315893
3	60614	Chicago	262.397109
11	32162	The Villages	249.306931
8	77449	Houston	184.696017
5	77084	Houston	172.947368
4	79936	El Paso	157.179819
2	77494	Houston	156.796578
1	75070	Dallas-Fort Worth	136.529487

To verify trimming the data of the bottom 25th and upper 75th percentiles, I check the means of each subset and visualize the distributions.

In [11]:
```
df3['Cumulative Percent Change'].mean()
```
executed in 24ms, finished 20:33:19 2022-06-29

Out[11]:
```
232.56052088963128
```

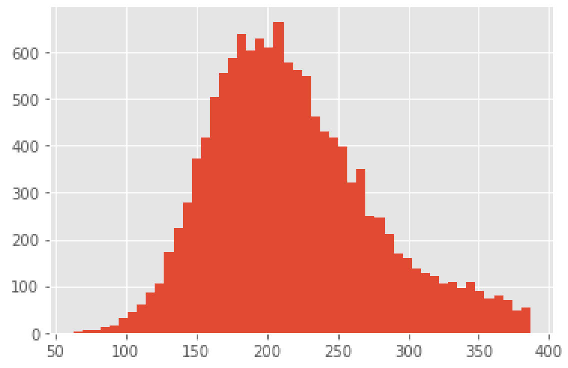
In [12]:
```
df3['Cumulative Percent Change'].hist(bins=50);
```
executed in 536ms, finished 20:33:19 2022-06-29



In [13]:
```
df3_trimmed['Cumulative Percent Change'].mean()

# Mean ROI 218.09
```
executed in 10ms, finished 20:33:19 2022-06-29

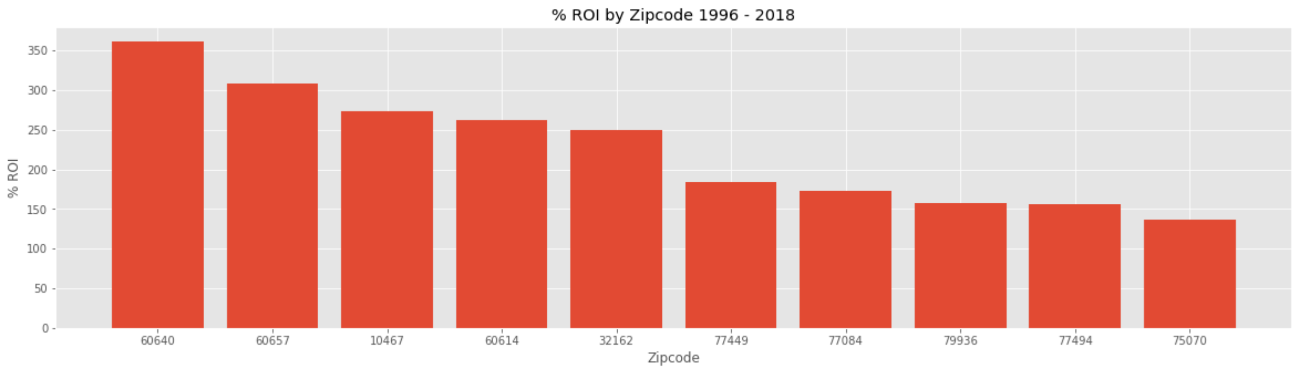
Out[13]:
```
218.0907895856348
```

In [14]:
```
df3_trimmed['Cumulative Percent Change'].hist(bins=50);
```
executed in 480ms, finished 20:33:20 2022-06-29



4.3.2 Visualizing Cumulative ROI

In [15]:
```
cum_pct_top10['zipcode'] = cum_pct_top10['zipcode'].astype(str)

# Plotting the historical data
fig, ax = plt.subplots(figsize=(20,5))
plt.bar(cum_pct_top10.zipcode, cum_pct_top10['Cumulative Percent Change'])
plt.title('% ROI by Zipcode 1996 - 2018')
plt.xlabel('Zipcode')
plt.ylabel('% ROI')
plt.show()
```
executed in 407ms, finished 20:33:20 2022-06-29



In [16]:
```
Q1 = df4.quantile(0.25)
Q3 = df4.quantile(0.75)
IQR = Q3 - Q1

df4_trimmed = df4[~((df4 < (Q1 - 1.5 * IQR)) |(df4 > (Q3 + 1.5 * IQR))).any(axis=1)]
yrs5_pct_top10 = df4_trimmed.head(10).sort_values('5-year Percent Change', ascending=False)
yrs5_pct_top10
```
executed in 70ms, finished 20:33:20 2022-06-29

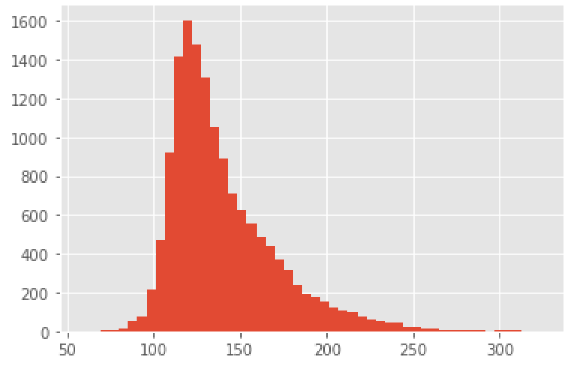
Out[16]:

	zipcode	Metro	5-year Percent Change
9	94109	San Francisco	165.538047
1	75070	Dallas-Fort Worth	159.464817
8	77449	Houston	152.951389
5	77084	Houston	146.696429
7	60640	Chicago	141.708907
6	10467	New York	141.565041
3	60614	Chicago	139.324166
0	60657	Chicago	139.213832
2	77494	Houston	130.705230
4	79936	El Paso	107.712766

In [17]:
```
df4['5-year Percent Change'].mean()
```
executed in 25ms, finished 20:33:20 2022-06-29

Out[17]:
```
139.91164731554213
```

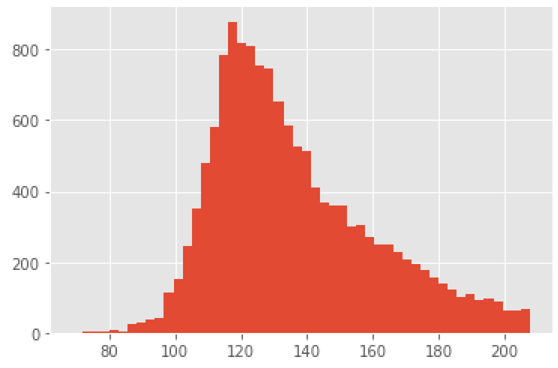
In [18]:
```
df4['5-year Percent Change'].hist(bins=50);
```
executed in 471ms, finished 20:33:21 2022-06-29



In [19]:
```
df4_trimmed['5-year Percent Change'].mean()
```
executed in 13ms, finished 20:33:21 2022-06-29

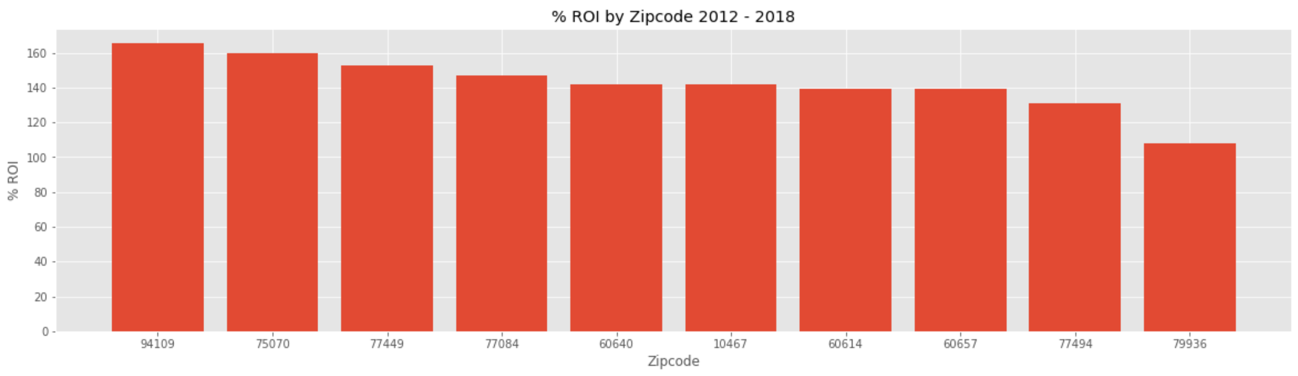
Out[19]:
```
136.1420128346487
```

In [20]:
```
df4_trimmed['5-year Percent Change'].hist(bins=50);
```
executed in 415ms, finished 20:33:21 2022-06-29



4.3.3 Visualizing Most Recent 5-Year ROI

In [21]:
```
yrs5_pct_top10['zipcode'] = yrs5_pct_top10['zipcode'].astype(str)

# Plotting the historical data
fig, ax = plt.subplots(figsize=(20,5))
plt.bar(yrs5_pct_top10.zipcode, yrs5_pct_top10['5-year Percent Change'])
plt.title('% ROI by Zipcode 2012 - 2018')
plt.xlabel('Zipcode')
plt.ylabel('% ROI')
plt.show()
```
executed in 433ms, finished 20:33:22 2022-06-29



In [22]:
```
cum_pct_top10['zipcode'] = cum_pct_top10['zipcode'].astype(int)
yrs5_pct_top10['zipcode'] = yrs5_pct_top10['zipcode'].astype(int)

from functools import reduce
dfs = [cum_pct_top10, yrs5_pct_top10]

# Combine into one dataframe ONLY those zipcodes that are on each of the Top 10 lists
df_combo = reduce(lambda left,right: pd.merge(left,right,on='zipcode'), dfs)
```
executed in 16ms, finished 20:33:22 2022-06-29

This shows not only the zipcodes with the highest ROIs over the long run, from 1996 to 2018, but also those zipcodes with the highest ROIs over the past five years.

In [23]:
```
df_combo
```
executed in 33ms, finished 20:33:22 2022-06-29

Out[23]:

	zipcode	Metro_x	Cumulative Percent Change	Metro_y	5-year Percent Change
0	60640	Chicago	361.570439	Chicago	141.708907
1	60657	Chicago	308.378217	Chicago	139.213832
2	10467	New York	273.315893	New York	141.565041
3	60614	Chicago	262.397109	Chicago	139.324166
4	77449	Houston	184.696017	Houston	152.951389
5	77084	Houston	172.947368	Houston	146.696429
6	79936	El Paso	157.179819	El Paso	107.712766
7	77494	Houston	156.796578	Houston	130.705230
8	75070	Dallas-Fort Worth	136.529487	Dallas-Fort Worth	159.464817

In [24]:
```
df_combo.drop(columns = ['Metro_y', 'Metro_x'], inplace=True)
```
executed in 10ms, finished 20:33:22 2022-06-29

In [25]:
```
# Add a column summing cumulative ROI and 5-yr ROI
df_combo['Total Change'] = df_combo['Cumulative Percent Change'] + df_combo['5-year Percent Change']
```
executed in 7ms, finished 20:33:22 2022-06-29

In [26]:
```
# Sort zipcodes based on total change and save top 5 zipcodes into a subset
df_small = df_combo.drop(columns = ['Cumulative Percent Change',
                                    '5-year Percent Change']).sort_values('Total Change',
                                                                           ascending=False).head()
```
executed in 13ms, finished 20:33:22 2022-06-29

In [27]:
```
df_small
```
executed in 18ms, finished 20:33:22 2022-06-29

Out[27]:

	zipcode	Total Change
0	60640	503.279345
1	60657	447.592049
2	10467	414.880933
3	60614	401.721275
4	77449	337.647406

In [28]:
```
# Save the zipcodes in a list
zip_list = df_small['zipcode']
```
executed in 8ms, finished 20:33:22 2022-06-29

In [29]:
```
# Create a new dataframe from the original dataframe including only those zipcodes in the top 5
df_draft = df[df.zipcode.isin(zip_list)]

df_draft.head(10)
```
executed in 49ms, finished 20:33:22 2022-06-29

Out[29]:

	zipcode	City	State	Metro	CountyName	SizeRank	1996-04	1996-05	1996-06	1996-07	...	2017-07	2017-08	2017-09	201
0	60657	Chicago	IL	Chicago	Cook	1	334200.0	335400.0	336500.0	337600.0	...	1005500	1007500	1007800	100
3	60614	Chicago	IL	Chicago	Cook	4	498100.0	500900.0	503100.0	504600.0	...	1289800	1287700	1287400	1291
6	10467	New York	NY	New York	Bronx	7	152900.0	152700.0	152600.0	152400.0	...	394400	400000	407300	411
7	60640	Chicago	IL	Chicago	Cook	8	216500.0	216700.0	216900.0	217000.0	...	798000	787100	776100	77
8	77449	Katy	TX	Houston	Harris	9	95400.0	95600.0	95800.0	96100.0	...	166800	167400	168400	16

5 rows × 271 columns

4.3.4 Melt Data Function

In [30]:
```
# This function was provided for us with the dataset at the beginning of this project
def melt_data(df):
    """
    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.
    """
    melted = pd.melt(df, id_vars=['zipcode', 'City', 'State', 'Metro', 'CountyName', 'SizeRank'], var_na
    melted['time'] = pd.to_datetime(melted['time'])
    melted = melted.dropna(subset=['value'])
    return melted.groupby('time').aggregate({'value':'mean'})
```
executed in 12ms, finished 20:33:22 2022-06-29

4.3.5 Prospective Zipcodes For Analysis

In [31]:
```
# Using the provided melt function, create a dataframe of the prospective zipcodes
# For each zipcode, melt the data

df_prospects = pd.DataFrame()
for i in df_draft['zipcode']:
    x = melt_data(df_draft[df_draft['zipcode'] == i])
    df_prospects = pd.concat([df_prospects, x], axis=1)
    df_prospects.rename(columns = {'value':i}, inplace = True)

# Display results
df_prospects.head()
```
executed in 157ms, finished 20:33:22 2022-06-29

Out[31]:

	60657	60614	10467	60640	77449
time					
1996-04-01	334200.0	498100.0	152900.0	216500.0	95400.0
1996-05-01	335400.0	500900.0	152700.0	216700.0	95600.0
1996-06-01	336500.0	503100.0	152600.0	216900.0	95800.0
1996-07-01	337600.0	504600.0	152400.0	217000.0	96100.0
1996-08-01	338500.0	505500.0	152300.0	217100.0	96400.0

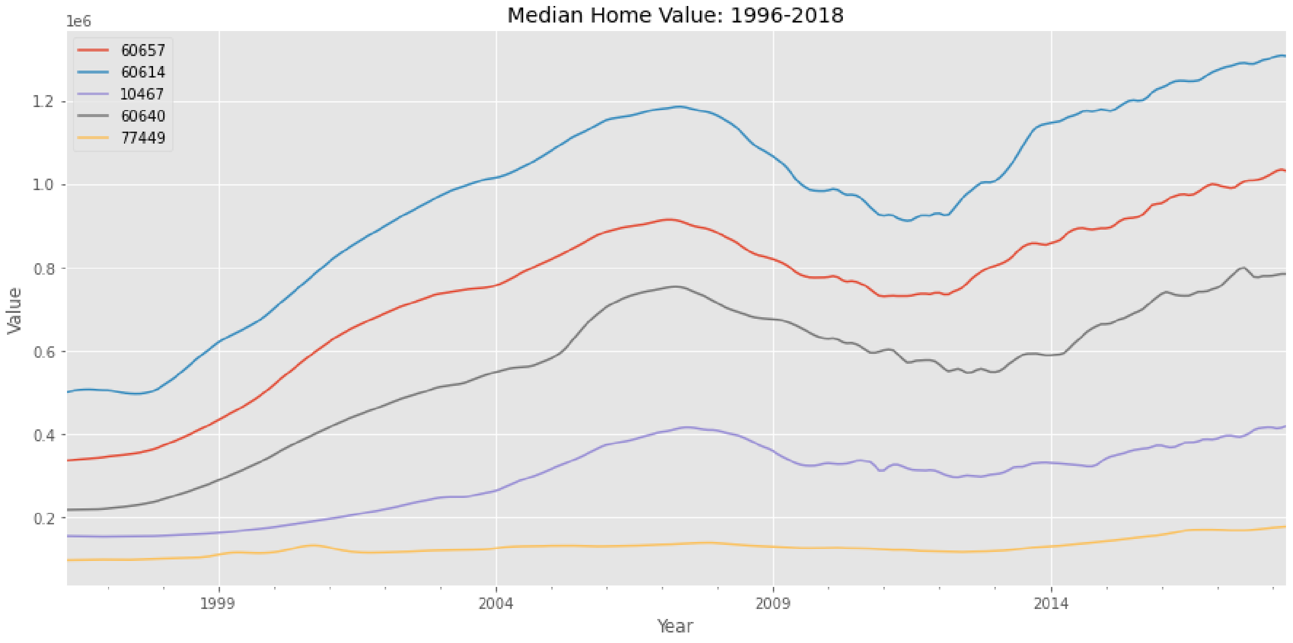
4.3.6 More on the Top Five Zipcodes

After completing this initial subsetting to narrow our list of 14,000 zipcodes down to 5, there are some initial observations.

- Three of the five zipcodes are in Chicago: Lakeview, Depaul/Lincoln Park, and Uptown
- One zipcode is in the Bronx (Van Cortland Park)
- One zipcode is an up-and-coming western suburb (Katy, TX) of Houston

In [32]:
```
# Ensure time series frequency is set as monthly
df_prospects = df_prospects.asfreq('MS')
```
executed in 15ms, finished 20:33:22 2022-06-29

In [33]:
```
df_prospects.plot(figsize=(15, 7))
plt.xlabel('Year')
plt.ylabel('Value')
plt.title('Median Home Value: 1996-2018')
plt.show();
```
executed in 534ms, finished 20:33:23 2022-06-29



For modeling, I subset the data to only include values beginning with 2011. This minimizes the outlier affect of the 2008-2009 housing crash on predictions and forecasts.

In [34]:
```
# For modeling, only include datapoints beginning in 2011 so as to not include
# the effects of the housing market crash
df_prospects_postcrash = df_prospects['2011-01-31':]
```
executed in 22ms, finished 20:33:23 2022-06-29

5 Modeling

In [35]:
```
# Create cutoff for train-validation set
cutoff = round(df_prospects_postcrash.shape[0]*0.8)
```
executed in 4ms, finished 20:33:23 2022-06-29

A note on the usage of the SARIMA Model:

The SARIMA model stands for "Seasonal AutoRegressive Integrated Moving Average" model, so in one swoop, the model can conduct auto-regression, moving average, the integration or differencing of the two, and include the seasonal component of the data.

A note on the SARIMA Parameters: Per the formula SARIMA(p,d,q)x(P,D,Q,s), the parameters for these types of models are as follows:

- p and seasonal P: indicate number of autoregressive terms (lags of the stationarized series, based on past data points)
- d and seasonal D: indicate differencing that must be done to stationarize series (accounting for the overall 'trend' in the data)
- q and seasonal Q: indicate number of moving average terms (lags of the forecast errors, based on noise of past data points)
- s: indicates seasonal length in the data (in this case, 12 months)

This project utilized a grid search of all the different model iterations with different combinations of pdq and pdqs. For the final model, the pdq and pdqs values were selected based on the smallest AIC value - the AIC is a metric that is an estimator of out-of-sample prediction error; a lower AIC score indicates a more predictive model.

A note on SARIMA Model performance evaluation:

Use MAE

5.1 Lakeview (Chicago, IL, 60657)

5.1.1 Lakeview: SARIMA Parameter Search

In [36]:
```
# Define train and test sets according to the index found above
train1 = df_prospects_postcrash[60657][:cutoff]
valid1 = df_prospects_postcrash[60657][cutoff:]

# 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))]

# Run a grid with pdq and seasonal pdq parameters calculated above and get the best AIC value
ans = []

for comb in pdq:
    for combs in pdqs:
        try:
            mod = sm.tsa.statespace.SARIMAX(train1,
                                            order = comb,
                                            seasonal_order = combs,
                                            enforce_stationarity = False,
                                            enforce_invertibility = False)
            output = mod.fit()
            ans.append([comb, combs, output.aic])
            print('ARIMA {} x {}12 : AIC Calculated = {}'.format(comb, combs, output.aic))
        except:
            continue
```

executed in 9.35s, finished 20:33:32 2022-06-29

```
ARIMA (0, 0, 1) x (0, 0, 0, 12)12 : AIC Calculated = 2006.7057360154313
ARIMA (0, 0, 1) x (0, 0, 1, 12)12 : AIC Calculated = 6341989.634026975
ARIMA (0, 0, 1) x (0, 1, 0, 12)12 : AIC Calculated = 1420.3937640029985
ARIMA (0, 0, 1) x (0, 1, 1, 12)12 : AIC Calculated = 1083.490379181547
ARIMA (0, 0, 1) x (1, 0, 0, 12)12 : AIC Calculated = 1697.946411013325
ARIMA (0, 0, 1) x (1, 0, 1, 12)12 : AIC Calculated = 1758.3234456782975
ARIMA (0, 0, 1) x (1, 1, 0, 12)12 : AIC Calculated = 1098.2457705699221
ARIMA (0, 0, 1) x (1, 1, 1, 12)12 : AIC Calculated = 1076.251022346739
ARIMA (0, 1, 0) x (0, 0, 0, 12)12 : AIC Calculated = 1364.9194448579628
ARIMA (0, 1, 0) x (0, 0, 1, 12)12 : AIC Calculated = 1108.7640841889174
ARIMA (0, 1, 0) x (0, 1, 0, 12)12 : AIC Calculated = 1115.8627379158047
ARIMA (0, 1, 0) x (0, 1, 1, 12)12 : AIC Calculated = 870.4590795345358
ARIMA (0, 1, 0) x (1, 0, 0, 12)12 : AIC Calculated = 1128.020229338298
ARIMA (0, 1, 0) x (1, 0, 1, 12)12 : AIC Calculated = 1113.3167271042014
ARIMA (0, 1, 0) x (1, 1, 0, 12)12 : AIC Calculated = 887.9023841191629
ARIMA (0, 1, 0) x (1, 1, 1, 12)12 : AIC Calculated = 870.7754660614501
ARIMA (0, 1, 1) x (0, 0, 0, 12)12 : AIC Calculated = 1277.4102853049785
ARIMA (0, 1, 1) x (0, 0, 1, 12)12 : AIC Calculated = 1051.2659505002964
ARIMA (0, 1, 1) x (0, 1, 0, 12)12 : AIC Calculated = 1082.0689124094793
```

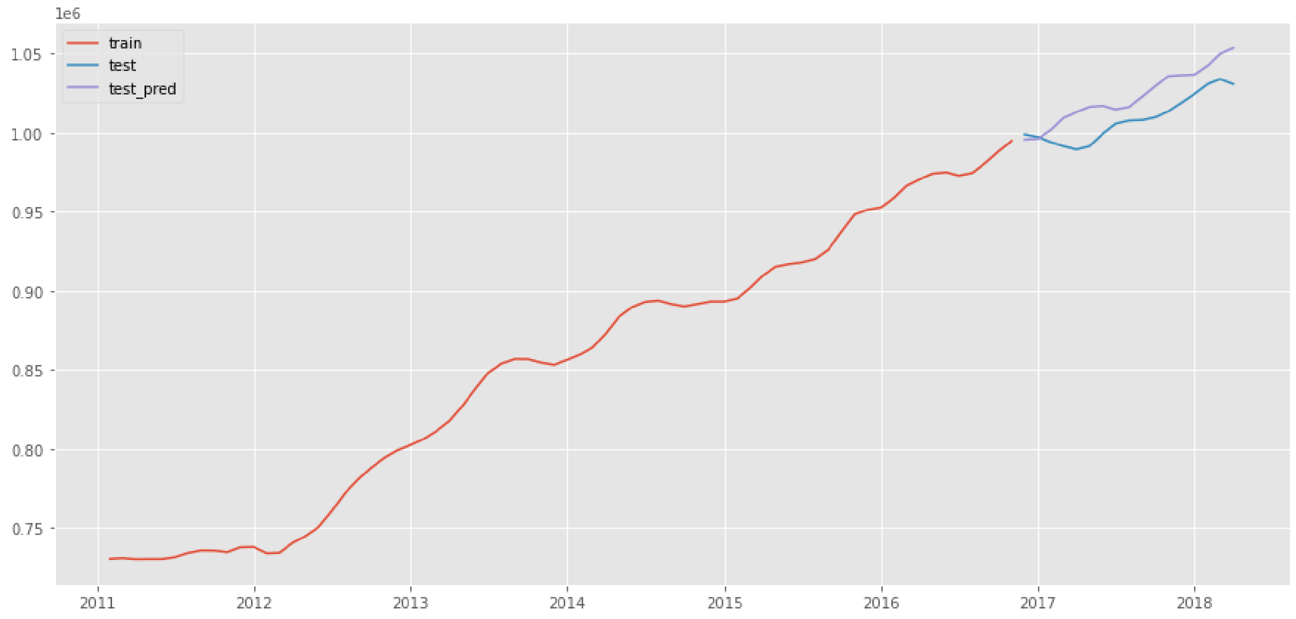
In [37]:
```python
# Find the parameters with minimal AIC value
ans_df = pd.DataFrame(ans, columns = ['pdq', 'pdqs', 'aic'])
ans_df.loc[ans_df['aic'].idxmin()]
```
executed in 64ms, finished 20:33:32 2022-06-29

Out[37]:
```
pdq          (1, 1, 1)
pdqs     (1, 1, 1, 12)
aic            810.964
Name: 63, dtype: object
```

5.1.2 Lakeview SARIMA Predictions

In [38]:
```python
# Plug the optimal parameter values into a new SARIMAX model
sarima1 = sm.tsa.statespace.SARIMAX(train1,
                                        order=(1, 1, 1),
                                        seasonal_order=(1, 1, 1, 12),
                                        enforce_stationarity=False,
                                        enforce_invertibility=False).fit()

y_hat_test1 = sarima1.predict(start=valid1.index[0], end=valid1.index[-1], typ='levels')

# Calculate the mean absolute error from residuals
mae1 = np.mean(np.abs(sarima1.resid))
print('The Mean Absolute Error of our predictions: $%.2f' % mae1)

fig, ax = plt.subplots(figsize=(15,7))
ax.plot(train1, label='train')
ax.plot(valid1, label='test')
ax.plot(y_hat_test1, label='test_pred')

plt.legend();
```
executed in 739ms, finished 20:33:33 2022-06-29

```
The Mean Absolute Error of our predictions: $44097.36
```

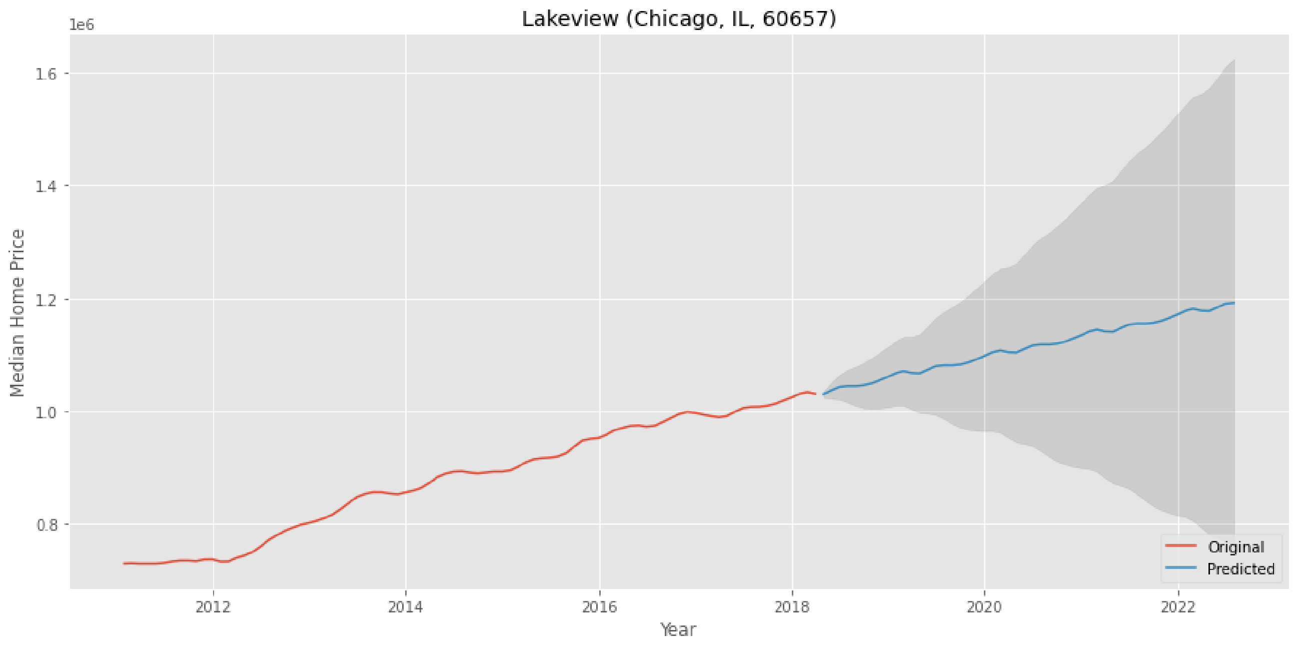


5.1.3 Lakeview SARIMA Forecast

```
In [39]: sarima_mod1 = sm.tsa.statespace.SARIMAX(df_prospects_postcrash[60657],
                                         order=(1, 1, 1),
                                         seasonal_order=(1, 1, 0, 12),
                                         enforce_stationarity=False,
                                         enforce_invertibility=False).fit()

         forecast1 = sarima_mod1.get_forecast(steps=52).summary_frame()
         forecast1_mean = round(forecast1['mean'][51])
         low_int1 =  round(forecast1['mean_ci_lower'][51])
         high_int1 = round(forecast1['mean_ci_upper'][51])
         ci_delta1 = round(high_int1 - low_int1)

         print(f'Lakeview (Chicago, IL, 60657):')
         print(f'95% confidence that the true future value is between ${low_int1} and ${high_int1}')
         print(f'Confidence range: ${ci_delta1}')

         fig, ax = plt.subplots(figsize=(15, 7))
         plt.plot(df_prospects_postcrash[60657])
         plt.plot(forecast1['mean'])
         ax.fill_between(forecast1.index, forecast1['mean_ci_lower'],
                         forecast1['mean_ci_upper'], color='k', alpha=0.1)
         plt.title('Lakeview (Chicago, IL, 60657)')
         plt.legend(['Original','Predicted'], loc='lower right')
         plt.xlabel('Year')
         plt.ylabel('Median Home Price')
         plt.show()
```

executed in 610ms, finished 20:33:34 2022-06-29

```
Lakeview (Chicago, IL, 60657):
95% confidence that the true future value is between $759441 and $1624075
Confidence range: $864634
```



5.2 Depaul/Lincoln Park (Chicago, IL, 60614)

5.2.1 Depaul/Lincoln Park SARIMA Parameters

In [40]:
```
# Define train and test sets according to the index found above
train2 = df_prospects_postcrash[60614][:cutoff]
valid2 = df_prospects_postcrash[60614][cutoff:]

# 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))]

# Run a grid with pdq and seasonal pdq parameters calculated above and get the best AIC value
ans = []

for comb in pdq:
    for combs in pdqs:
        try:
            mod = sm.tsa.statespace.SARIMAX(train2,
                                            order = comb,
                                            seasonal_order = combs,
                                            enforce_stationarity = False,
                                            enforce_invertibility = False)
            output = mod.fit()
            ans.append([comb, combs, output.aic])
            print('ARIMA {} x {}12 : AIC Calculated = {}'.format(comb, combs, output.aic))
        except:
            continue
```
executed in 7.55s, finished 20:33:41 2022-06-29

```
ARIMA (0, 0, 0) x (1, 0, 0, 12)12 : AIC Calculated = 1399.837238924396
ARIMA (0, 0, 0) x (1, 0, 1, 12)12 : AIC Calculated = 1404.5086705571828
ARIMA (0, 0, 0) x (1, 1, 0, 12)12 : AIC Calculated = 1137.4301906062833
ARIMA (0, 0, 0) x (1, 1, 1, 12)12 : AIC Calculated = 1319.8918430424421
ARIMA (0, 0, 1) x (0, 0, 0, 12)12 : AIC Calculated = 855950.437719476
ARIMA (0, 0, 1) x (0, 0, 1, 12)12 : AIC Calculated = 1791.842601995287
ARIMA (0, 0, 1) x (0, 1, 0, 12)12 : AIC Calculated = 1466.4182159313573
ARIMA (0, 0, 1) x (0, 1, 1, 12)12 : AIC Calculated = 1091.193721719454
ARIMA (0, 0, 1) x (1, 0, 0, 12)12 : AIC Calculated = 1886.2629349250155
ARIMA (0, 0, 1) x (1, 0, 1, 12)12 : AIC Calculated = 1798.67251388005
ARIMA (0, 0, 1) x (1, 1, 0, 12)12 : AIC Calculated = 1133.49407396651
ARIMA (0, 0, 1) x (1, 1, 1, 12)12 : AIC Calculated = 1114.742513308918
ARIMA (0, 1, 0) x (0, 0, 0, 12)12 : AIC Calculated = 1408.5218500587516
ARIMA (0, 1, 0) x (0, 0, 1, 12)12 : AIC Calculated = 1153.3662413042834
ARIMA (0, 1, 0) x (0, 1, 0, 12)12 : AIC Calculated = 1164.3648064420859
ARIMA (0, 1, 0) x (0, 1, 1, 12)12 : AIC Calculated = 919.9461406382939
ARIMA (0, 1, 0) x (1, 0, 0, 12)12 : AIC Calculated = 1171.380750935906
ARIMA (0, 1, 0) x (1, 0, 1, 12)12 : AIC Calculated = 1207.6592403271172
ARIMA (0, 1, 0) x (1, 1, 0, 12)12 : AIC Calculated = 923.6934083768965
```

In [41]:
```
# Find the parameters with minimal AIC value
ans_df = pd.DataFrame(ans, columns = ['pdq', 'pdqs', 'aic'])
ans_df.loc[ans_df['aic'].idxmin()]
```
executed in 9ms, finished 20:33:41 2022-06-29

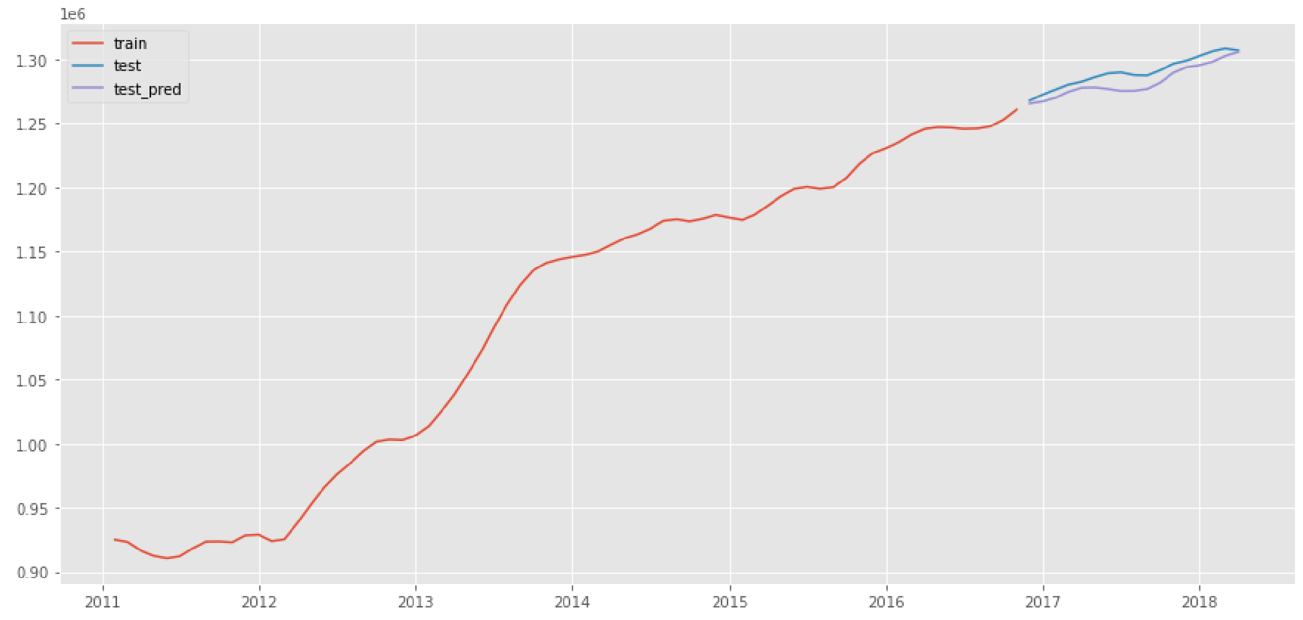
Out[41]:
```
pdq         (1, 1, 1)
pdqs    (0, 1, 1, 12)
aic            817.82
Name: 59, dtype: object
```

5.2.2 Depaul/Lincoln Park SARIMA Predictions

In [42]:
```
# Plug the optimal parameter values into a new SARIMAX model
sarima2 = sm.tsa.statespace.SARIMAX(train2,
                                    order=(1, 1, 1),
                                    seasonal_order=(0, 1, 1, 12),
                                    enforce_stationarity=False,
                                    enforce_invertibility=False).fit()

y_hat_test2 = sarima2.predict(start=valid2.index[0], end=valid2.index[-1],typ='levels')

# Calculate the mean absolute error from residuals
mae2 = np.mean(np.abs(sarima2.resid))
print('The Mean Absolute Error of our predictions: $%.2f' % mae2)

fig, ax = plt.subplots(figsize=(15,7))
ax.plot(train2, label='train')
ax.plot(valid2, label='test')
ax.plot(y_hat_test2, label='test_pred')

plt.legend();
```

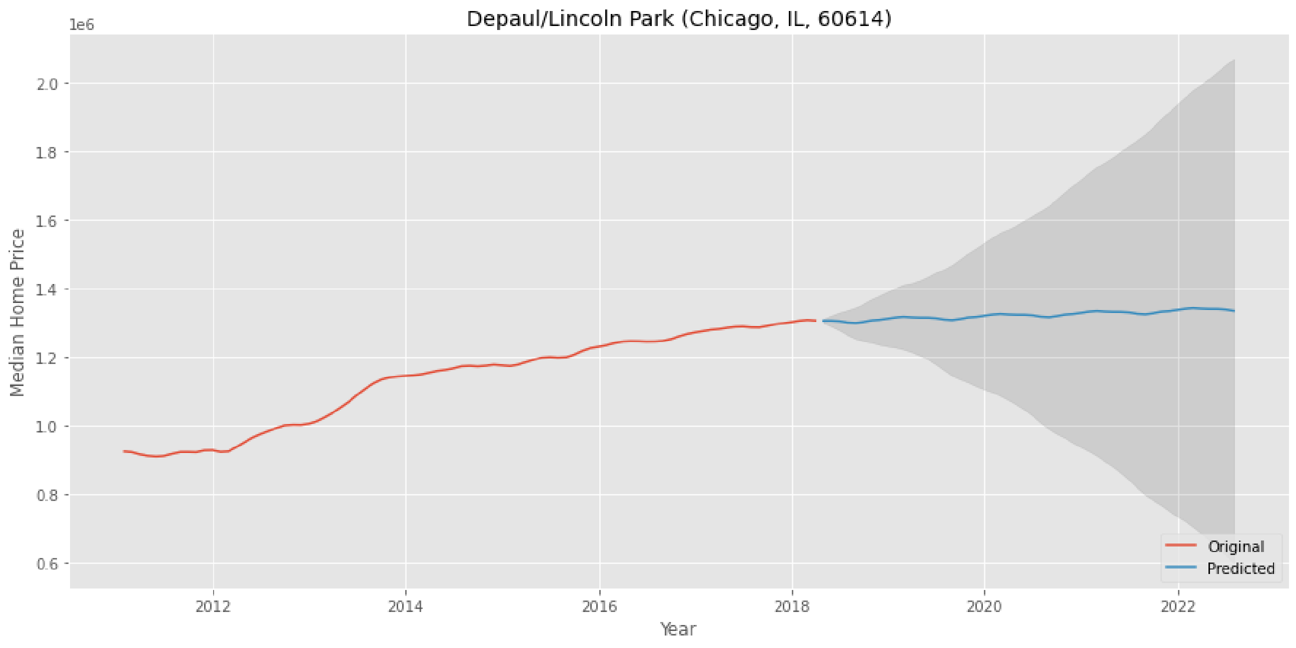
executed in 549ms, finished 20:33:42 2022-06-29

```
The Mean Absolute Error of our predictions: $58447.74
```



5.2.3 Depaul/Lincoln Park SARIMA Forecast

In [43]:
```
# Plug the optimal parameter values into a new SARIMAX model
sarima_mod2 = sm.tsa.statespace.SARIMAX(df_prospects_postcrash[60614],
                                        order=(1, 1, 1),
                                        seasonal_order=(0, 1, 1, 12),
                                        enforce_stationarity=False,
                                        enforce_invertibility=False).fit()

forecast2 = sarima_mod2.get_forecast(steps=52).summary_frame()
forecast2_mean = round(forecast2['mean'][51])
low_int2 =  round(forecast2['mean_ci_lower'][51])
high_int2 = round(forecast2['mean_ci_upper'][51])
ci_delta2 = round(high_int2 - low_int2)


fig, ax = plt.subplots(figsize=(15, 7))
plt.plot(df_prospects_postcrash[60614])
plt.plot(forecast2['mean'])
ax.fill_between(forecast2.index, forecast2['mean_ci_lower'],
                    forecast2['mean_ci_upper'], color='k', alpha=0.1)
plt.title('Depaul/Lincoln Park (Chicago, IL, 60614)')
plt.legend(['Original','Predicted'], loc='lower right')
plt.xlabel('Year')
plt.ylabel('Median Home Price')
plt.show()


print(f'Depaul/Lincoln Park (Chicago, IL, 60614):')
print(f'95% confidence that the true future value is between ${low_int2} and ${high_int2}')
print(f'Confidence range: ${ci_delta2}')
```
executed in 488ms, finished 20:33:42 2022-06-29



```
Depaul/Lincoln Park (Chicago, IL, 60614):
95% confidence that the true future value is between $598924 and $2071511
Confidence range: $1472587
```

5.3 The Bronx/Van Cortland Park (NYC, 10467)

5.3.1 The Bronx/Van Cortland Park SARIMA Parameters

In [44]:
```
# Define train and test sets according to the index found above
train3 = df_prospects_postcrash[10467][:cutoff]
valid3 = df_prospects_postcrash[10467][cutoff:]

# 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))]

# Run a grid with pdq and seasonal pdq parameters calculated above and get the best AIC value
ans = []

for comb in pdq:
    for combs in pdqs:
        try:
            mod = sm.tsa.statespace.SARIMAX(train3,
                                            order = comb,
                                            seasonal_order = combs,
                                            enforce_stationarity = False,
                                            enforce_invertibility = False)
            output = mod.fit()
            ans.append([comb, combs, output.aic])
            print('ARIMA {} x {}12 : AIC Calculated = {}'.format(comb, combs, output.aic))
        except:
            continue
```
executed in 8.29s, finished 20:33:51 2022-06-29

```
ARIMA (0, 0, 0) x (0, 0, 0, 12)12 : AIC Calculated = 1952.3672728672839
ARIMA (0, 0, 0) x (0, 0, 1, 12)12 : AIC Calculated = 1583.4201218543678
ARIMA (0, 0, 0) x (0, 1, 0, 12)12 : AIC Calculated = 1309.0548619613517
ARIMA (0, 0, 0) x (0, 1, 1, 12)12 : AIC Calculated = 1039.726698118213
ARIMA (0, 0, 0) x (1, 0, 0, 12)12 : AIC Calculated = 1311.8286858517333
ARIMA (0, 0, 0) x (1, 0, 1, 12)12 : AIC Calculated = 1237.7435111901484
ARIMA (0, 0, 0) x (1, 1, 0, 12)12 : AIC Calculated = 1060.6419205782759
ARIMA (0, 0, 0) x (1, 1, 1, 12)12 : AIC Calculated = 1387.2904202657098
ARIMA (0, 0, 1) x (0, 0, 0, 12)12 : AIC Calculated = 1877.539889925758
ARIMA (0, 0, 1) x (0, 0, 1, 12)12 : AIC Calculated = 1634.6304059180486
ARIMA (0, 0, 1) x (0, 1, 0, 12)12 : AIC Calculated = 1305.501830364878
ARIMA (0, 0, 1) x (0, 1, 1, 12)12 : AIC Calculated = 982.1226532895585
ARIMA (0, 0, 1) x (1, 0, 0, 12)12 : AIC Calculated = 1587.5621437692635
ARIMA (0, 0, 1) x (1, 0, 1, 12)12 : AIC Calculated = 1633.8028461525878
ARIMA (0, 0, 1) x (1, 1, 0, 12)12 : AIC Calculated = 1024.4794397743367
ARIMA (0, 0, 1) x (1, 1, 1, 12)12 : AIC Calculated = 993.7973554635779
ARIMA (0, 1, 0) x (0, 0, 0, 12)12 : AIC Calculated = 1280.7702445508414
ARIMA (0, 1, 0) x (0, 0, 1, 12)12 : AIC Calculated = 1057.6955430858395
ARIMA (0, 1, 0) x (0, 1, 0, 12)12 : AIC Calculated = 1098.1084869550364
```

In [45]:
```
# Find the parameters with minimal AIC value
ans_df = pd.DataFrame(ans, columns = ['pdq', 'pdqs', 'aic'])
ans_df.loc[ans_df['aic'].idxmin()]
```
executed in 90ms, finished 20:33:51 2022-06-29

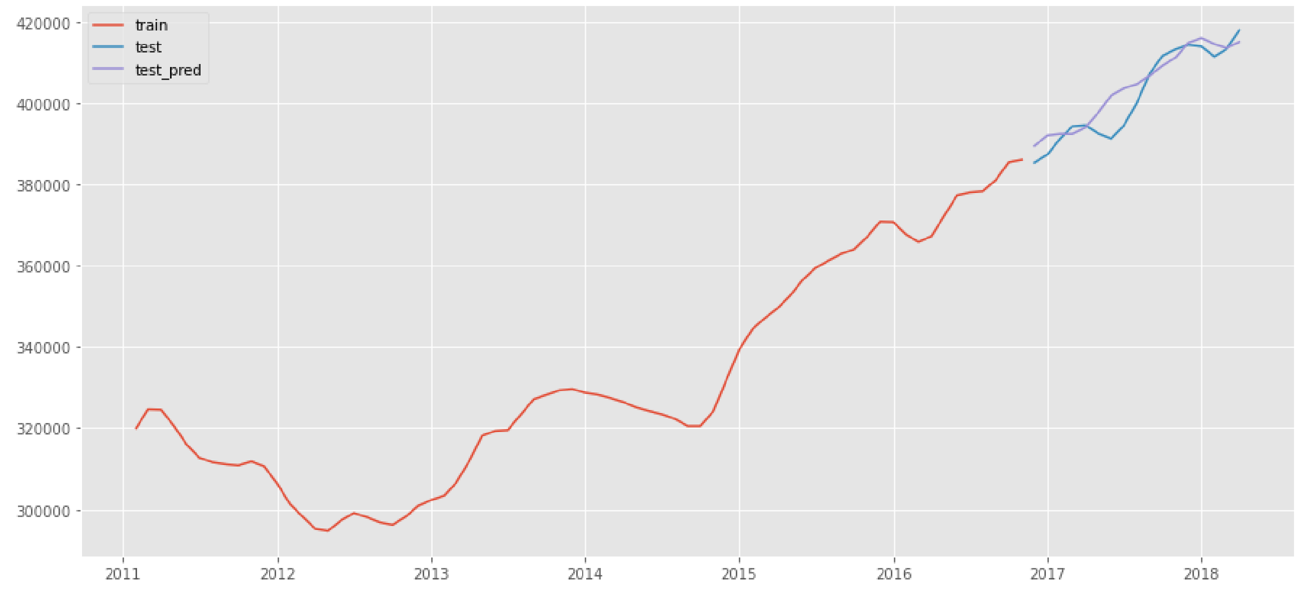
Out[45]:
```
pdq          (1, 1, 1)
pdqs     (1, 1, 1, 12)
aic            793.027
Name: 63, dtype: object
```

5.3.2 The Bronx/Van Cortland Park SARIMA Predictions

In [46]:
```
# Plug the optimal parameter values into a new SARIMAX model
sarima3 = sm.tsa.statespace.SARIMAX(train3,
                                        order=(1, 1, 1),
                                        seasonal_order=(1, 1, 1, 12),
                                        enforce_stationarity=False,
                                        enforce_invertibility=False).fit()

y_hat_test3 = sarima3.predict(start=valid3.index[0], end=valid3.index[-1],typ='levels')

# Calculate the mean absolute error from residuals
mae3 = np.mean(np.abs(sarima3.resid))
print('The Mean Absolute Error of our predictions: $%.2f' % mae3)

fig, ax = plt.subplots(figsize=(15,7))
ax.plot(train3, label='train')
ax.plot(valid3, label='test')
ax.plot(y_hat_test3, label='test_pred')

plt.legend();
```

executed in 888ms, finished 20:33:52 2022-06-29

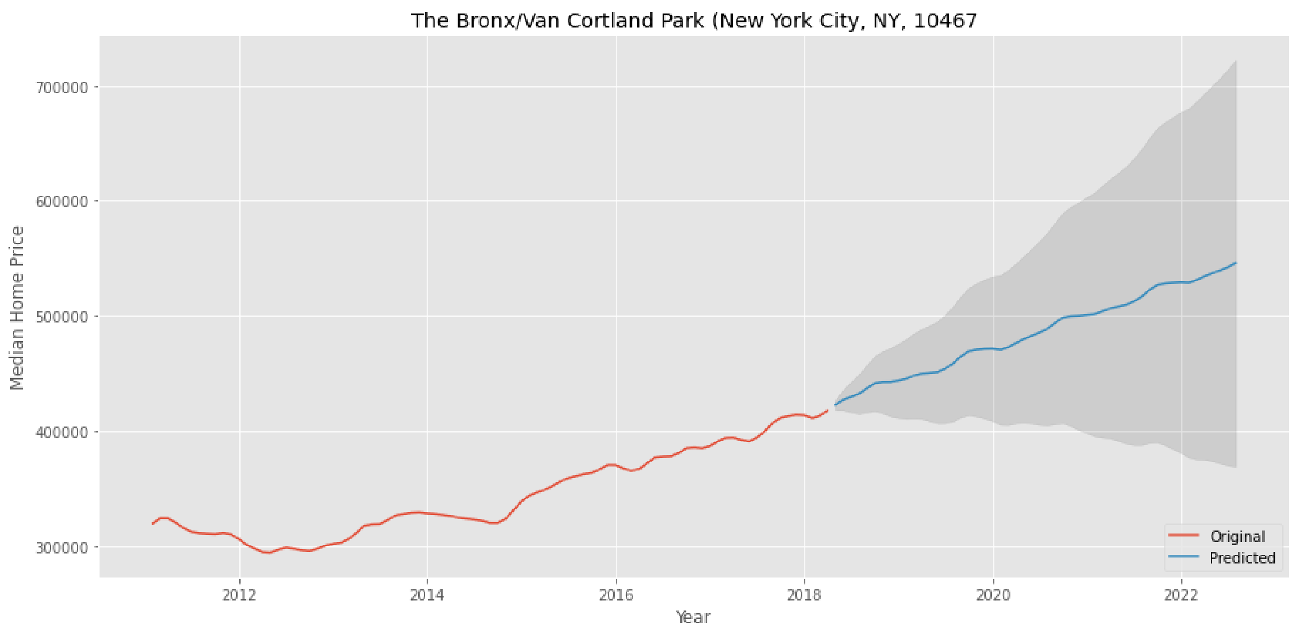
```
The Mean Absolute Error of our predictions: $13507.14
```



5.3.3 The Bronx/Van Cortland Park SARIMA Forecast

In [47]:
```python
# Plug the optimal parameter values into a new SARIMAX model
sarima_mod3 = sm.tsa.statespace.SARIMAX(df_prospects_postcrash[10467],
                                        order=(1, 1, 1),
                                        seasonal_order=(1, 1, 1, 12),
                                        enforce_stationarity=False,
                                        enforce_invertibility=False).fit()

forecast3 = sarima_mod3.get_forecast(steps=52).summary_frame()
forecast3_mean = round(forecast3['mean'][51])
low_int3 =  round(forecast3['mean_ci_lower'][51])
high_int3 = round(forecast3['mean_ci_upper'][51])
ci_delta3 = round(high_int3 - low_int3)


fig, ax = plt.subplots(figsize=(15, 7))
plt.plot(df_prospects_postcrash[10467])
plt.plot(forecast3['mean'])
ax.fill_between(forecast3.index, forecast3['mean_ci_lower'],
                    forecast3['mean_ci_upper'], color='k', alpha=0.1)
plt.title('The Bronx/Van Cortland Park (New York City, NY, 10467')
plt.legend(['Original','Predicted'], loc='lower right')
plt.xlabel('Year')
plt.ylabel('Median Home Price')
plt.show()


print(f'The Bronx/Van Cortland Park (New York City, NY, 10467')
print(f'95% confidence that the true future value is between ${low_int3} and ${high_int3}')
print(f'Confidence range: ${ci_delta3}')
```

executed in 826ms, finished 20:33:52 2022-06-29



```
The Bronx/Van Cortland Park (New York City, NY, 10467
95% confidence that the true future value is between $369452 and $722262
Confidence range: $352810
```

5.4 Uptown (Chicago, IL, 60640)

5.4.1 Uptown SARIMA Parameters

In [48]:

```python
# Define train and test sets according to the index found above
train4 = df_prospects_postcrash[60640][:cutoff]
valid4 = df_prospects_postcrash[60640][cutoff:]

# 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))]

# Run a grid with pdq and seasonal pdq parameters calculated above and get the best AIC value
ans = []

for comb in pdq:
    for combs in pdqs:
        try:
            mod = sm.tsa.statespace.SARIMAX(train4,
                                            order = comb,
                                            seasonal_order = combs,
                                            enforce_stationarity = False,
                                            enforce_invertibility = False)
            output = mod.fit()
            ans.append([comb, combs, output.aic])
            print('ARIMA {} x {}12 : AIC Calculated = {}'.format(comb, combs, output.aic))
        except:
            continue
```

executed in 10.7s, finished 20:34:03 2022-06-29

```
ARIMA (0, 0, 1) x (0, 0, 0, 12)12 : AIC Calculated = 1964.4957780810198
ARIMA (0, 0, 1) x (0, 0, 1, 12)12 : AIC Calculated = 1610.5366534878785
ARIMA (0, 0, 1) x (0, 1, 0, 12)12 : AIC Calculated = 1403.156086031353
ARIMA (0, 0, 1) x (0, 1, 1, 12)12 : AIC Calculated = 1071.7127455999955
ARIMA (0, 0, 1) x (1, 0, 0, 12)12 : AIC Calculated = 1661.8267915010729
ARIMA (0, 0, 1) x (1, 0, 1, 12)12 : AIC Calculated = 1606.592266911814
ARIMA (0, 0, 1) x (1, 1, 0, 12)12 : AIC Calculated = 1111.9536017717792
ARIMA (0, 0, 1) x (1, 1, 1, 12)12 : AIC Calculated = 1460.71812309697
ARIMA (0, 1, 0) x (0, 0, 0, 12)12 : AIC Calculated = 1372.0620449574562
ARIMA (0, 1, 0) x (0, 0, 1, 12)12 : AIC Calculated = 1119.43760204724
ARIMA (0, 1, 0) x (0, 1, 0, 12)12 : AIC Calculated = 1143.75312950116
ARIMA (0, 1, 0) x (0, 1, 1, 12)12 : AIC Calculated = 891.7533152334679
ARIMA (0, 1, 0) x (1, 0, 0, 12)12 : AIC Calculated = 1140.3103629959555
ARIMA (0, 1, 0) x (1, 0, 1, 12)12 : AIC Calculated = 1121.285881640366
ARIMA (0, 1, 0) x (1, 1, 0, 12)12 : AIC Calculated = 915.455965994411
ARIMA (0, 1, 0) x (1, 1, 1, 12)12 : AIC Calculated = 893.7961496788989
ARIMA (0, 1, 1) x (0, 0, 0, 12)12 : AIC Calculated = 1284.7383897239986
ARIMA (0, 1, 1) x (0, 0, 1, 12)12 : AIC Calculated = 1051.0887477918243
ARIMA (0, 1, 1) x (0, 1, 0, 12)12 : AIC Calculated = 1137.8560215028676
```

In [49]:

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

executed in 180ms, finished 20:34:03 2022-06-29

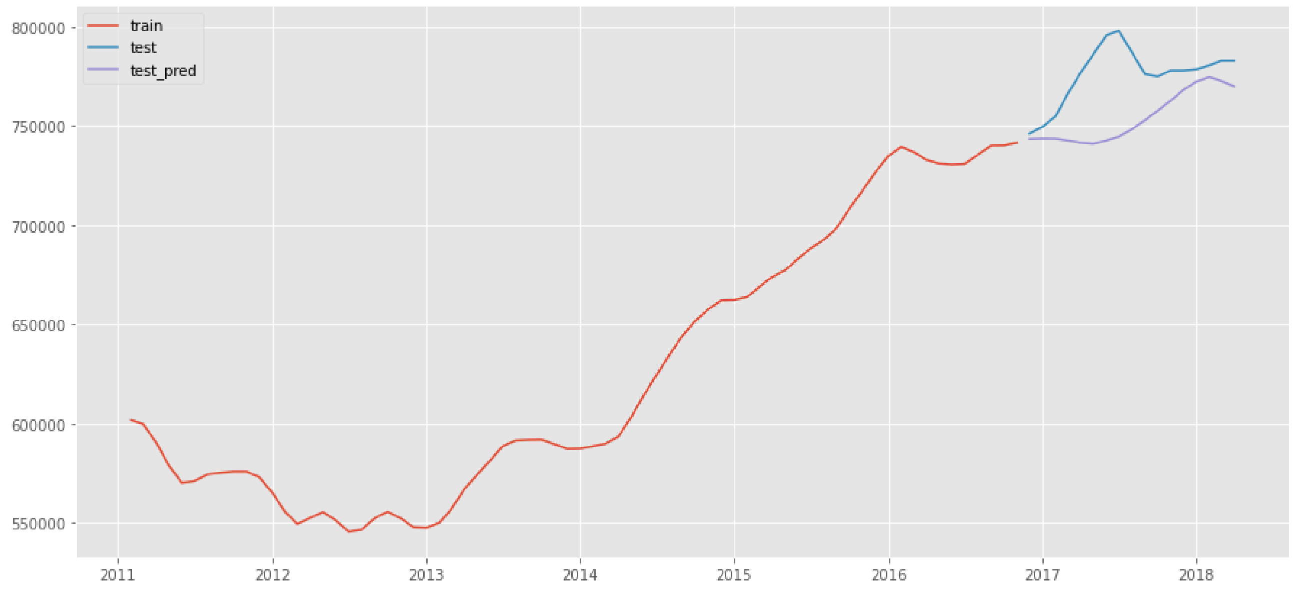
Out[49]:

```
pdq          (1, 1, 1)
pdqs     (1, 1, 1, 12)
aic            827.021
Name: 63, dtype: object
```

5.4.2 Uptown SARIMA Predictions

In [50]:
```
# Plug the optimal parameter values into a new SARIMAX model
sarima4 = sm.tsa.statespace.SARIMAX(train4,
                                        order=(1, 1, 1),
                                        seasonal_order=(1, 1, 1, 12),
                                        enforce_stationarity=False,
                                        enforce_invertibility=False).fit()

y_hat_test4 = sarima4.predict(start=valid4.index[0], end=valid4.index[-1],typ='levels')

# Calculate the mean absolute error from residuals
mae4 = np.mean(np.abs(sarima4.resid))
print('The Mean Absolute Error of our predictions: $%.2f' % mae4)

fig, ax = plt.subplots(figsize=(15,7))
ax.plot(train4, label='train')
ax.plot(valid4, label='test')
ax.plot(y_hat_test4, label='test_pred')

plt.legend();
```
executed in 664ms, finished 20:34:04 2022-06-29

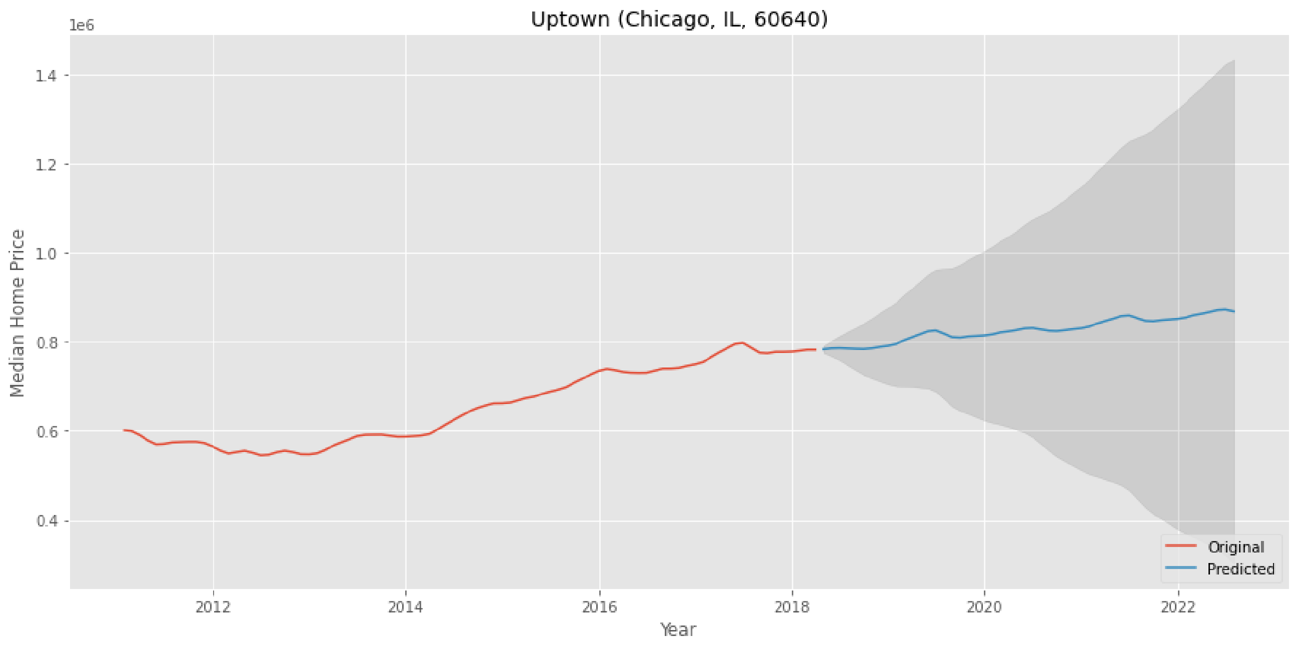
```
The Mean Absolute Error of our predictions: $27626.26
```



5.4.3 Uptown SARIMA Forecast

In [51]:
```
# Plug the optimal parameter values into a new SARIMAX model
sarima_mod4 = sm.tsa.statespace.SARIMAX(df_prospects_postcrash[60640],
                                        order=(1, 1, 1),
                                        seasonal_order=(1, 1, 1, 12),
                                        enforce_stationarity=False,
                                        enforce_invertibility=False).fit()

forecast4 = sarima_mod4.get_forecast(steps=52).summary_frame()
forecast4_mean = round(forecast4['mean'][51])
low_int4 =  round(forecast4['mean_ci_lower'][51])
high_int4 = round(forecast4['mean_ci_upper'][51])
ci_delta4 = round(high_int4 - low_int4)


fig, ax = plt.subplots(figsize=(15, 7))
plt.plot(df_prospects_postcrash[60640])
plt.plot(forecast4['mean'])
ax.fill_between(forecast4.index, forecast4['mean_ci_lower'],
                    forecast4['mean_ci_upper'], color='k', alpha=0.1)
plt.title('Uptown (Chicago, IL, 60640)')
plt.legend(['Original','Predicted'], loc='lower right')
plt.xlabel('Year')
plt.ylabel('Median Home Price')
plt.show()


print(f'Uptown (Chicago, IL, 60640):')
print(f'95% confidence that the true future value is between ${low_int4} and ${high_int4}')
print(f'Confidence range: ${ci_delta4}')
```

executed in 576ms, finished 20:34:05 2022-06-29



```
Uptown (Chicago, IL, 60640):
95% confidence that the true future value is between $301783 and $1435313
Confidence range: $1133530
```

5.5 Katy, TX (Houston Suburb, 77449)

5.5.1 Katy SARIMA Parameters

In [52]:
```
# Define train and test sets according to the index found above
train5 = df_prospects_postcrash[77449][:cutoff]
valid5 = df_prospects_postcrash[77449][cutoff:]

# 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))]

# Run a grid with pdq and seasonal pdq parameters calculated above and get the best AIC value
ans = []

for comb in pdq:
    for combs in pdqs:
        try:
            mod = sm.tsa.statespace.SARIMAX(train5,
                                            order = comb,
                                            seasonal_order = combs,
                                            enforce_stationarity = False,
                                            enforce_invertibility = False)
            output = mod.fit()
            ans.append([comb, combs, output.aic])
            print('ARIMA {} x {}12 : AIC Calculated = {}'.format(comb, combs, output.aic))
        except:
            continue
```

executed in 8.78s, finished 20:34:13 2022-06-29

```
ARIMA (0, 0, 1) x (0, 0, 0, 12)12 : AIC Calculated = 1755.2721107835443
ARIMA (0, 0, 1) x (0, 0, 1, 12)12 : AIC Calculated = 153642.6886173263
ARIMA (0, 0, 1) x (0, 1, 0, 12)12 : AIC Calculated = 1184.021585548715
ARIMA (0, 0, 1) x (0, 1, 1, 12)12 : AIC Calculated = 1378.80769398341
ARIMA (0, 0, 1) x (1, 0, 0, 12)12 : AIC Calculated = 1584.023564548756
ARIMA (0, 0, 1) x (1, 0, 1, 12)12 : AIC Calculated = 1434.4227372141138
ARIMA (0, 0, 1) x (1, 1, 0, 12)12 : AIC Calculated = 966.3449723616714
ARIMA (0, 0, 1) x (1, 1, 1, 12)12 : AIC Calculated = 918.2601881472069
ARIMA (0, 1, 0) x (0, 0, 0, 12)12 : AIC Calculated = 1138.1563976052746
ARIMA (0, 1, 0) x (0, 0, 1, 12)12 : AIC Calculated = 911.5750037857814
ARIMA (0, 1, 0) x (0, 1, 0, 12)12 : AIC Calculated = 894.7654870553334
ARIMA (0, 1, 0) x (0, 1, 1, 12)12 : AIC Calculated = 700.0027750408709
ARIMA (0, 1, 0) x (1, 0, 0, 12)12 : AIC Calculated = 911.6955462585789
ARIMA (0, 1, 0) x (1, 0, 1, 12)12 : AIC Calculated = 898.1256662081054
ARIMA (0, 1, 0) x (1, 1, 0, 12)12 : AIC Calculated = 715.4130498353594
ARIMA (0, 1, 0) x (1, 1, 1, 12)12 : AIC Calculated = 701.9351045943639
ARIMA (0, 1, 1) x (0, 0, 0, 12)12 : AIC Calculated = 1048.9281117728722
ARIMA (0, 1, 1) x (0, 0, 1, 12)12 : AIC Calculated = 842.2430349200495
ARIMA (0, 1, 1) x (0, 1, 0, 12)12 : AIC Calculated = 853.9128386339045
```

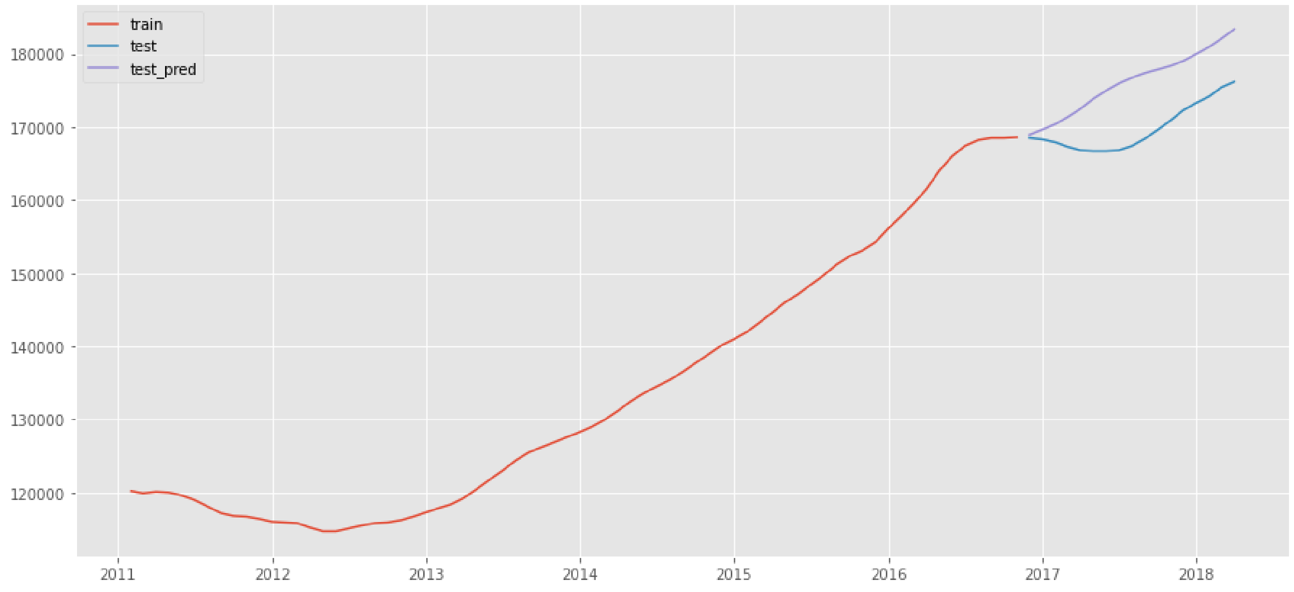
In [53]:
```
# Find the parameters with minimal AIC value
ans_df = pd.DataFrame(ans, columns = ['pdq', 'pdqs', 'aic'])
ans_df.loc[ans_df['aic'].idxmin()]
```
executed in 41ms, finished 20:34:13 2022-06-29

Out[53]:
```
pdq          (1, 1, 1)
pdqs     (1, 1, 1, 12)
aic            615.691
Name: 63, dtype: object
```

5.5.2 Katy SARIMA Predictions

In [54]:
```
# Plug the optimal parameter values into a new SARIMAX model
sarima5 = sm.tsa.statespace.SARIMAX(train5,
                                    order=(1, 1, 1),
                                    seasonal_order=(1, 1, 1, 12),
                                    enforce_stationarity=False,
                                    enforce_invertibility=False).fit()

y_hat_test5 = sarima5.predict(start=valid5.index[0], end=valid5.index[-1],typ='levels')

# Calculate the mean absolute error from residuals
mae5 = np.mean(np.abs(sarima5.resid))
print('The Mean Absolute Error of our predictions: $%.2f' % mae5)

fig, ax = plt.subplots(figsize=(15,7))
ax.plot(train5, label='train')
ax.plot(valid5, label='test')
ax.plot(y_hat_test5, label='test_pred')

plt.legend();
```
executed in 1.49s, finished 20:34:15 2022-06-29

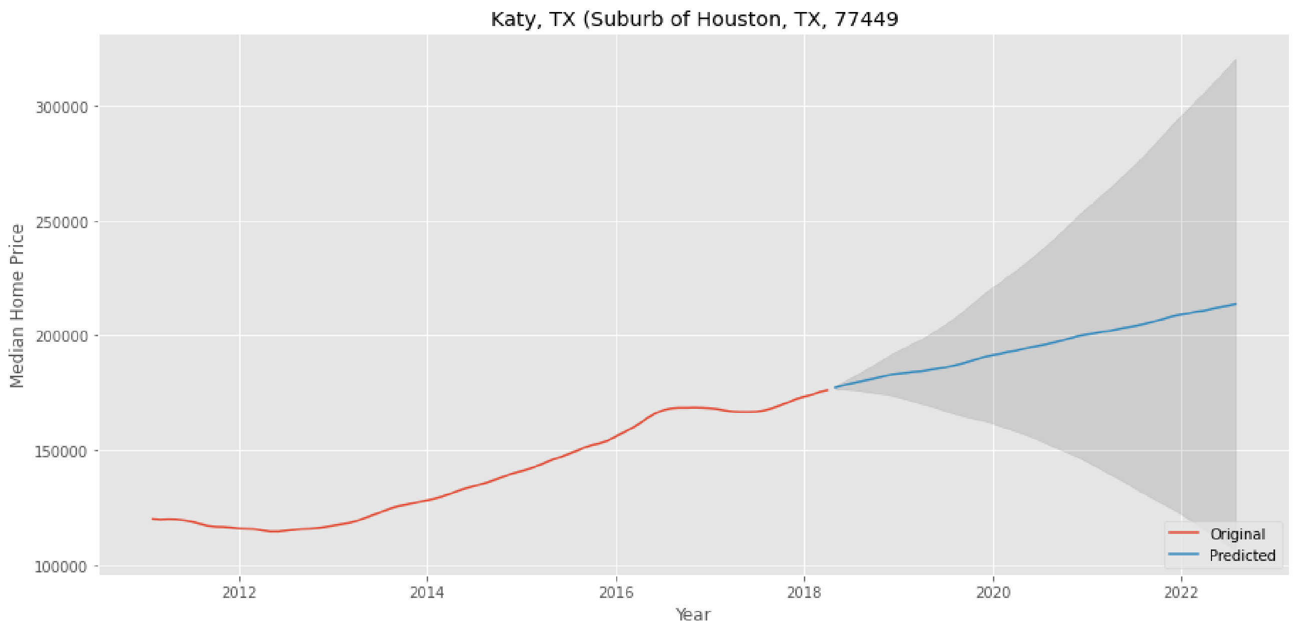
```
The Mean Absolute Error of our predictions: $4466.01
```



5.5.3 Katy SARIMA Forecast

```
In [55]: # Plug the optimal parameter values into a new SARIMAX model
         sarima_mod5 = sm.tsa.statespace.SARIMAX(df_prospects_postcrash[77449],
                                                 order=(1, 1, 1),
                                                 seasonal_order=(1, 1, 1, 12),
                                                 enforce_stationarity=False,
                                                 enforce_invertibility=False).fit()

         forecast5 = sarima_mod5.get_forecast(steps=52).summary_frame()
         forecast5_mean = round(forecast5['mean'][51])
         low_int5 =  round(forecast5['mean_ci_lower'][51])
         high_int5 = round(forecast5['mean_ci_upper'][51])
         ci_delta5 = round(high_int5 - low_int5)


         fig, ax = plt.subplots(figsize=(15, 7))
         plt.plot(df_prospects_postcrash[77449])
         plt.plot(forecast5['mean'])
         ax.fill_between(forecast5.index, forecast5['mean_ci_lower'],
                         forecast5['mean_ci_upper'], color='k', alpha=0.1)
         plt.title('Katy, TX (Suburb of Houston, TX, 77449')
         plt.legend(['Original','Predicted'], loc='lower right')
         plt.xlabel('Year')
         plt.ylabel('Median Home Price')
         plt.show()


         print(f'Katy, TX (Suburb of Houston, TX, 77449:')
         print(f'95% confidence that the true future value is between ${low_int5} and ${high_int5}')
         print(f'Confidence range: ${ci_delta5}')
```

executed in 709ms, finished 20:34:16 2022-06-29



```
Katy, TX (Suburb of Houston, TX, 77449:
95% confidence that the true future value is between $106555 and $320536
Confidence range: $213981
```

6 Evaluation

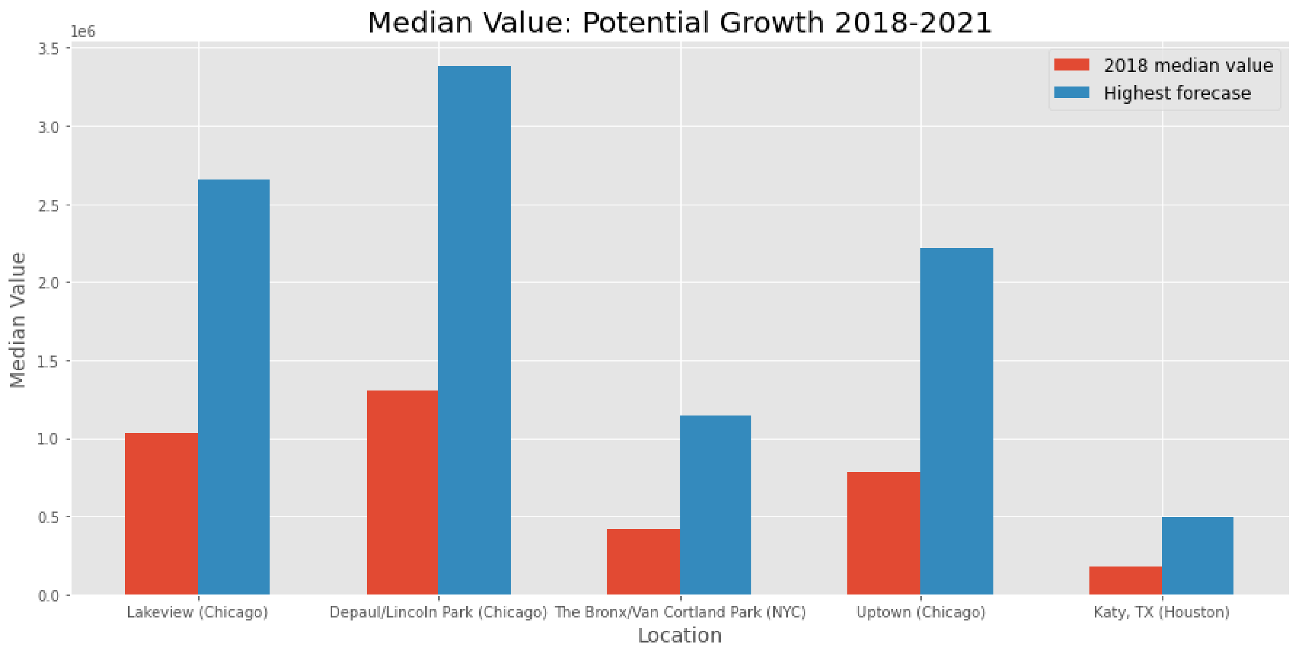
In [56]:
```
d = {'zipcode': ['60657', '60614', '10467', '60640', '77449'],
     'city': ['Lakeview (Chicago)',
              'Depaul/Lincoln Park (Chicago)',
              'The Bronx/Van Cortland Park (NYC)',
              'Uptown (Chicago)',
              'Katy, TX (Houston)'],
     '2018 median value': [1030600, 1307000, 417900, 782800, 176200],
     'mae': [mae1, mae2, mae3, mae4, mae5],
     'low_conf': [low_int1, low_int2, low_int3, low_int4, low_int5],
     'high_conf': [high_int1, high_int2, high_int3, high_int4, high_int5],
     'forecast range': [ci_delta1, ci_delta2, ci_delta3, ci_delta4, ci_delta5]}

df_results = pd.DataFrame(data=d)
df_results['low_end'] = df_results['2018 median value'] + df_results['low_conf']
df_results['high_end'] = df_results['2018 median value'] + df_results['high_conf']
df_results['Best case ROI%'] = round(((df_results['high_end'] - df_results['2018 median value']) /
                                     df_results['2018 median value']) * 100, 2)
```
executed in 46ms, finished 20:34:16 2022-06-29

In [57]:
```
df_results
```
executed in 41ms, finished 20:34:16 2022-06-29

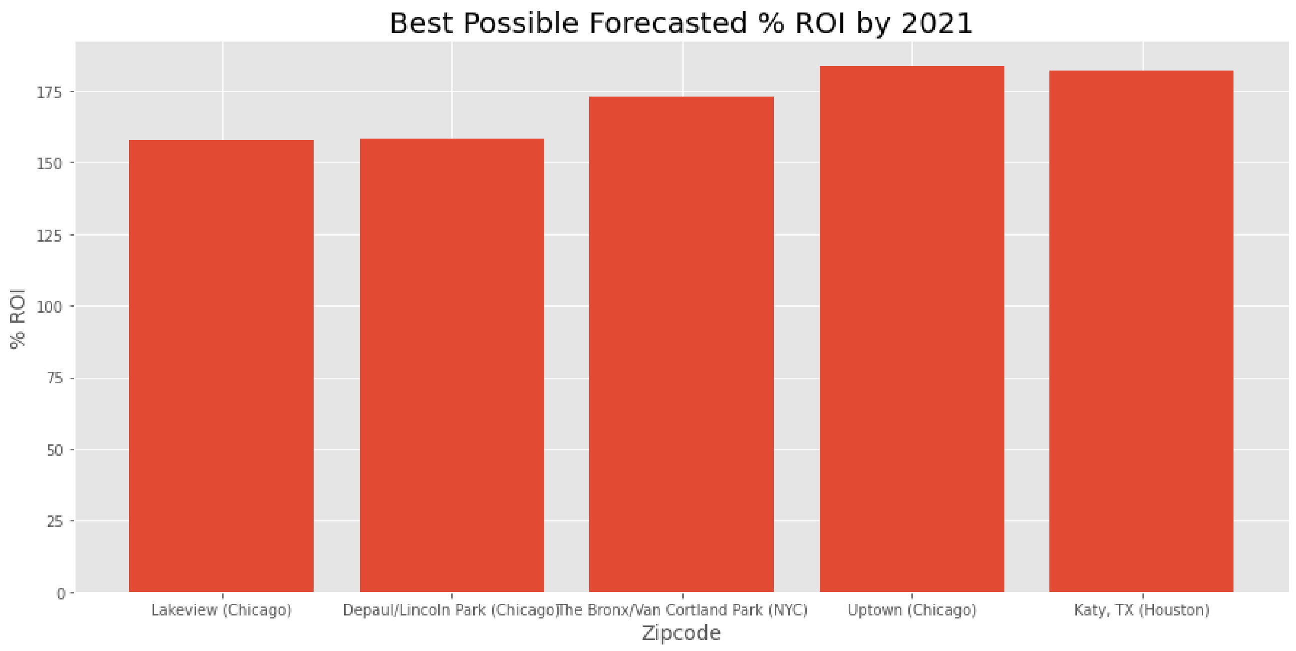
Out[57]:

	zipcode	city	2018 median value	mae	low_conf	high_conf	forecast range	low_end	high_end	Best case ROI%
0	60657	Lakeview (Chicago)	1030600	44097.363722	759441	1624075	864634	1790041	2654675	157.59
1	60614	Depaul/Lincoln Park (Chicago)	1307000	58447.741423	598924	2071511	1472587	1905924	3378511	158.49
2	10467	The Bronx/Van Cortland Park (NYC)	417900	13507.136939	369452	722262	352810	787352	1140162	172.83
3	60640	Uptown (Chicago)	782800	27626.262986	301783	1435313	1133530	1084583	2218113	183.36
4	77449	Katy, TX (Houston)	176200	4466.014997	106555	320536	213981	282755	496736	181.92

In [58]:
```
#create the base axis to add the bars to
fig, ax = plt.subplots(figsize = (15,7));

#extract the labels
label = df_results['city']

#use this to create x ticks to add the data to
x = np.arange(len(label))

#set a width for each bar
width = 0.3

rect1 = ax.bar(x - width/2, df_results['2018 median value'],
               width = width, label = '2018 median value')

rect2 = ax.bar(x + width/2, df_results['high_end'],
               width = width, label = 'Highest forecase')

#add the labels to the axis
ax.set_ylabel('Median Value', fontsize = 14)
ax.set_xlabel('Location', fontsize = 14)
ax.set_title('Median Value: Potential Growth 2018-2021', fontsize = 20)

#set the ticks
ax.set_xticks(x)
ax.set_xticklabels(label)

#add the legend
#using the labels of the bars

ax.legend(fontsize = 12);
```
executed in 747ms, finished 20:34:17 2022-06-29



In [59]:
```
# Plotting the forecasted ROI data
fig, ax = plt.subplots(figsize=(15,7))
plt.bar(df_results.city, df_results['Best case ROI%'])
plt.title('Best Possible Forecasted % ROI by 2021', fontsize = 20)
plt.xlabel('Zipcode', fontsize = 14)
plt.ylabel('% ROI', fontsize = 14)

plt.show()
```
executed in 563ms, finished 20:34:17 2022-06-29



1. The zipcode model with the smallest MAE of only 4,466 was Katy, TX (77449)
2. The zipcode forecast with the smallest 95% confidence interval range (thus perhaps the less risky) is also Katy, TX, with a range of only 213,981
3. On the contrary, the zipcode model with the highest MAE was Depaul/Lincoln Park of Chicago (60614) at 58,447
4. Depaul/Lincoln Park also has the widest confidence interval range, incidating a more risky option, with a range of 1,472,587

7 Recommendations

For those clients who may be more risk-averse, possibly with less cash on hand, Katy, TX (77449) and The Bronx/Van Cortland Park (10467) may be the best zipcodes in which to invest because:

- In Katy, the median home value in 2018 was 176,200 and our model predictions only deviated by as much as 4500. Further, forecasted Katy home values two years out have a less varied range than other zipcodes. Thus, in two years, we can expect home values to increase on the low end to 282,755 and the high end 496,736.
- In the Bronx, the median home value in 2018 was 417,900 and our model predictions deviated by as much as 13,507. Further, forecasted home values in the Bronx two years out have a less varied range than other zipcodes. Thus, in two years, we can expect home values to increase on the low end to 787,352 and the high end 1,140,162.

For those clients who may be more willing to take a risk, and perhaps with more cash on hand, the Lakeview and Depaul/Lincoln Park neighborhoods of Chicago may be the best zipcobes in which to invest because:

- In Lakeview, the median home value in 2018 was 1,030,600. In making our predictions, our model deviated by as much as 44,097. Forecasted Lakeview home values two years out have a more varied range than other zipcodes. Thus, in two years, we can expect home values to increase on the low end to 1,790,041 and the high end 2,654,675.
- In Depaul/Lincoln Park, the median home value in 2018 was 1,307,000. In making our predictions, our model deviated by as much as 58,447. Forecasted Depaul/Lincoln Park home values two years out have a more varied range than other zipcodes. Thus, in two years, we can expect home values to increase on the low end to 1,905,924 and the high end 3,368,511.

For clients looking for a good, middle-of-the-road investment zipcode with an above average median home value and moderate error in predictions, the Uptown neighborhood of Chicago would be a sound option. In Uptown, the median home value in 2018 was 782,800. In making our predictions, our model deviated by as much as 27,626. Forecasted Uptown home values two years out do have a more varied range than other zipcodes, which we may note of in terms of potential risk. Thus, in two years, we can expect home values to increase on the low end to 1,084,583 and the high end 2,218,113.

8 Future Work

I can draw additional data on median incomes, population size, housing demand, cost of living, community attributes, etc. to better ascertain zipcodes of interest worth modeling.