Analysis of Rolling Sales Data - Queens (04/01/2020 - 03/31/2021)

Steps

I am going to do the following:

```
1. Import necessary modules
2. Load the prepped data per borough
3. Analyze the data for trends and seasonality
4. Dickey-Fuller Tests and preparing data for ARMA modeling
     - Induce stationarity if needed
5. ARMA model of the data
6. Error analysis of the ARMA model
     - Try to improve ARMA model
7. Comparison with latest data
     -Test data from 04/01/2021 - 04/31/2021
8. Observations/Conclusions/Recommendations
```

1. Imports

In [175]:

```
import pandas as pd
from pandas.plotting import register_matplotlib_converters
import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import datetime
from statsmodels.tsa.arima_model import ARMA
from statsmodels.tsa.stattools import adfuller, acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
import numpy as np
from matplotlib.pylab import rcParams
from sklearn.metrics import mean_squared_error
from math import sqrt
import sklearn
import math

#Supress default INFO logging
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import logging
logger = logging.getLogger()
logger.setLevel(logging.CRITICAL)
import logging, sys
warnings.simplefilter(action='ignore', category=FutureWarning)
```

2. Loading the prepared data

Observations:

- Once I loaded the data and sorted it, the SALE DATE values range from 4/1/2020 until 3/31/2021.
- This data was the most recent data when I started working on the project.
- NYC OpenData website updates this data regularly with newer months about every 2-3 months
- The latest data which came out this month gave data up to 4/31/2021, which I can test against the prediction for 30 days

In [176]:
```
#Loading prepped data
df = pd.read_csv('datasets/rollingsales_queens.xls_prepped_bare.csv')
df.reset_index(drop=True, inplace=True)
df.sort_values('SALE DATE')
```

Out[176]:

	TAX CLASS AT PRESENT	ZIP CODE	SALE PRICE	SALE DATE
11776	1	11434	434500	2020-04-01
5303	2	11375	1150000	2020-04-01
10406	4	11418	2500000	2020-04-01
11814	1	11434	358000	2020-04-01
12658	1	11357	720000	2020-04-01
...	...	...	...	...
13114	2	11377	370000	2021-03-31
10094	1	11418	773800	2021-03-31
5566	2	11004	167600	2021-03-31
5156	2	11375	425000	2021-03-31
4230	2	11355	400000	2021-03-31

13171 rows × 4 columns

3. Analyzing the data for trends/seasonality

I do the following steps here to help the data work with the modules:

```
1. Convert 'SALE DATE' column to datetime format
2. Create new dataframe with 'SALE DATE' as the index and 'SALE PRICE' a
s the column
3. Since we have multiple sales per day, I will aggregate the data into
 daily data by taking the daily average of sales
4. Check the data for any nulls/NaNs
    -Decide what to do for Nulls/NaNs
5. Use statsmodels to observe the data for trends and seasonality
```

Observations:

```
- NaN values came into the data after the data got aggregated.
- Upon further inspection, this was due to the 70 days of no sales in th
e origional data.
    - Dropping these rows will result in skewing the data predictions
    - I decided to repalce the NaN values with 0 since no sales were don
e on that day
        -This also preserves the 365 day row length
```

In [177]:
```
# 1. Convert 'SALE DATE' column to datetime format

df['SALE DATE'] = pd.to_datetime(df['SALE DATE'])
```

In [178]:
```
# 2 . Create new dataframe with 'SALE DATE' as the index and 'SALE PRICE' as the

df_price_date = pd.DataFrame(df, columns=['SALE DATE', 'SALE PRICE'])
df_price_date = df_price_date.set_index('SALE DATE')
df_price_date.head()
```

Out[178]:

	SALE PRICE
SALE DATE	
2020-07-16	4121000
2020-08-28	584569
2021-01-11	800000
2020-12-16	300000
2020-06-23	360000

In [179]:
```
# 3. Group the sales data by daily average

df_price_date = df_price_date.resample('D').mean()
```

In [180]:
```
# 4. We see here number of rows went down from 13171 to 295. Why wasn't it 365 ro
df_price_date.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 365 entries, 2020-04-01 to 2021-03-31
Freq: D
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   SALE PRICE  295 non-null    float64
dtypes: float64(1)
memory usage: 5.7 KB
```

In [181]:
```
#Here we see that since we resampled by day, there are NaN values for the days th
df_price_date['SALE PRICE'].isna().sum()
```

Out[181]: 70

In [182]:
```
# 4. Instead of dropping the rows, I decided to fill NaN with 0 to reflect no sal
df_price_date['SALE PRICE'].fillna(0, inplace=True)
df_price_date
```

Out[182]:

	SALE PRICE
SALE DATE	
2020-04-01	961150.000000
2020-04-02	753357.142857
2020-04-03	681724.206897
2020-04-04	0.000000
2020-04-05	0.000000
...	...
2021-03-27	0.000000
2021-03-28	0.000000
2021-03-29	694114.470588
2021-03-30	747610.935484
2021-03-31	602154.750000

365 rows × 1 columns

In [183]:
```
# 5. Checking for trends/seasonality
#Here I check the origional data against its 7-day weekly rolling window to see

df_price_date['roll_avg'] = df_price_date.rolling(window=7).mean()
df_price_date
```

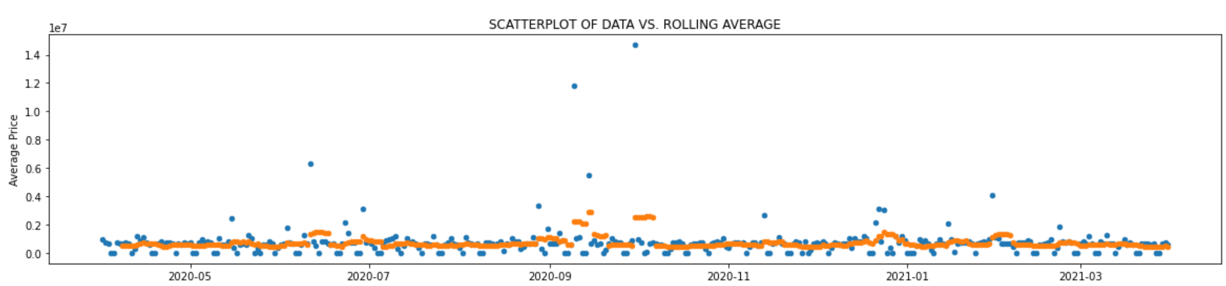
Out[183]:

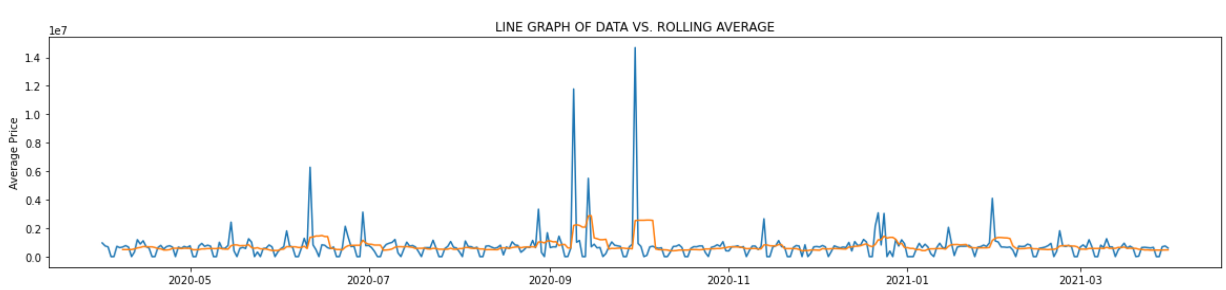
SALE DATE	SALE PRICE	roll_avg
2020-04-01	961150.000000	NaN
2020-04-02	753357.142857	NaN
2020-04-03	681724.206897	NaN
2020-04-04	0.000000	NaN
2020-04-05	0.000000	NaN
...	...	...
2021-03-27	0.000000	460444.131063
2021-03-28	0.000000	456158.416777
2021-03-29	694114.470588	463897.318187
2021-03-30	747610.935484	476893.916114
2021-03-31	602154.750000	472297.028269

365 rows × 2 columns

In [184]:
```
#Plotting the 7-day rolling average against the origional data

plt.figure(figsize=(20, 4))
plt.title("SCATTERPLOT OF DATA VS. ROLLING AVERAGE")
plt.xlabel("Year-Month")
plt.ylabel("Average Price")

#s=20 to keep dots small in size
plt.scatter(df_price_date.index[:365], df_price_date['SALE PRICE'][:365], s=20)
plt.scatter(df_price_date.index[7:], df_price_date['roll_avg'][7:], s=20);
plt.figure(figsize=(20, 4))

plt.title("LINE GRAPH OF DATA VS. ROLLING AVERAGE")
plt.plot(df_price_date.index[:365], df_price_date['SALE PRICE'][:365])
plt.plot(df_price_date.index[7:], df_price_date['roll_avg'][7:]);
plt.xlabel("Year-Month")
plt.ylabel("Average Price")
```

Out[184]:
```
Text(0, 0.5, 'Average Price')
```





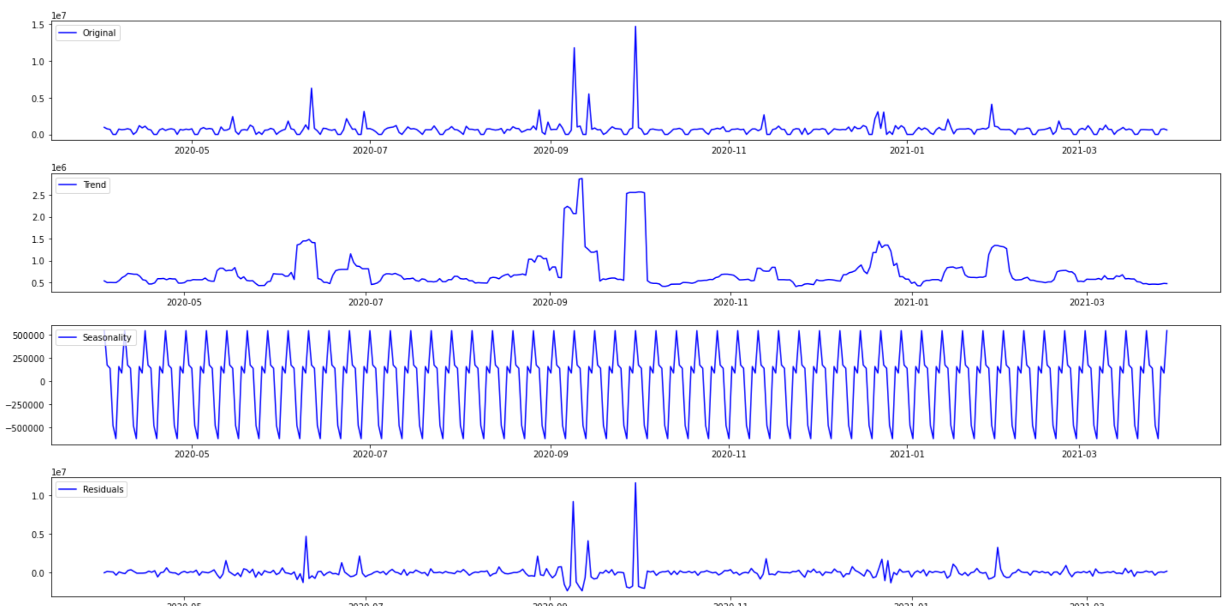
Observation

```
- The spikes in the data where the price goes to the millions or tens of
millions is due to buildings being bought.
- Other than that, the rest are residential properties well under a mill
ion in price
```

In [185]:
```python
# Statsmodels decomposition

# Additive model was chosen here. It would not allow multiplicative with "0" valu
# Period of 7 for weekly lag

decomposition = seasonal_decompose(df_price_date['SALE PRICE'], model='additive'
observed = decomposition.observed
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
```

In [186]:
```python
register_matplotlib_converters()
```

In [187]:
```python
plt.figure(figsize=(20,10))
plt.subplot(411)
plt.plot(observed, label='Original', color="blue")
plt.legend(loc='upper left')
plt.subplot(412)
plt.plot(trend, label='Trend', color="blue")
plt.legend(loc='upper left')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality', color="blue")
plt.legend(loc='upper left')
plt.subplot(414)
plt.plot(residual, label='Residuals', color="blue")
plt.legend(loc='upper left')
plt.tight_layout()
```



Observations:

- A large amount of sales happened between August 2020 and November 2020.
- Looks like there may be some seasonality every month

4. Dickey-Fuller Tests and preparing data for ARMA modeling

```
1. First I will run initial Augmented Dickey Fuller (ADF) test to check
 if the data is already stationary and does not have a unit root.
2. If the data fails the ADF test, I will induce stationarity using the
 following methods:
    - Differencing
    - Logging the data
    - Rolling mean subtraction
```

In [188]:
```
# Initial test
dftest = adfuller(df_price_date['SALE PRICE'])
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used'
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)'%key] = value
print(dftest)
print()
print(dfoutput)
```

```
(-8.565541911900292, 8.466399684197556e-14, 4, 360, {'1%': -3.448645946352023,
'5%': -2.869602139060357, '10%': -2.5710650077160495}, 10680.436655510479)

Test Statistic                 -8.565542e+00
p-value                         8.466400e-14
#Lags Used                      4.000000e+00
Number of Observations Used     3.600000e+02
Critical Value (1%)            -3.448646e+00
Critical Value (5%)            -2.869602e+00
Critical Value (10%)           -2.571065e+00
dtype: float64
```

Augmented Dickey Fuller Test Goals:

Our goal is to induce stationarity and show that the data does not have a unit root.

ADF Test Null Hypothesis: The data has a unit root and is non-stationary.

Requirements for stationarity:

```
1. If p-value <= 0.05: Reject the null hypothesis (H0), the data does no
t have a unit root and is stationary.
    - If p-value > 0.05: Fail to reject the null hypothesis (H0), the da
ta has a unit root and is non-stationary.
2. If the Test Statistic is lower than the critical values, then reject
 the null hypothesis. Data does not have a unity root and is stationary
```

Results of ADF Test

Test Statistic vs. Critical Values

- Initial test shows Test Statistic of **-8.565542**, this is greater than the critical values for 1% and 5%.
    - *We **REJECT** the null hypothesis! The data does not have a unit root and is stationary*

P-Value Analysis

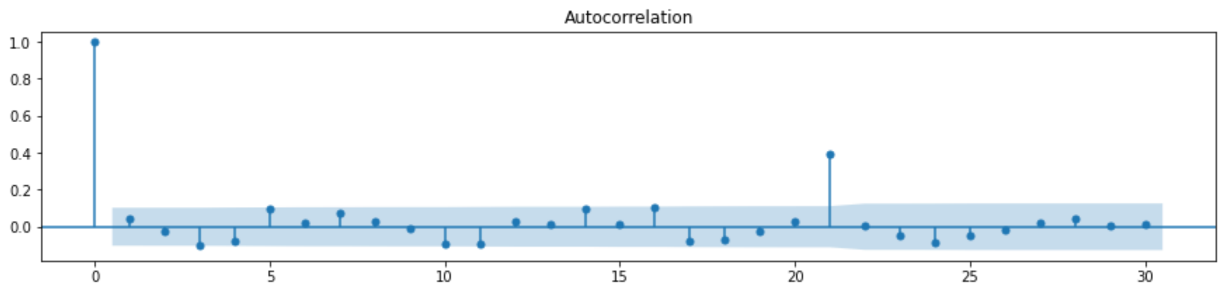
- Our current p-value is **8.466400e-14** or **0.00000000000008466400** which is REALLY close to zero.
    - *This means: p-value <= 0.05:*
    - *We **REJECT** the null hypothesis! The data does not have a unit root and is stationary*

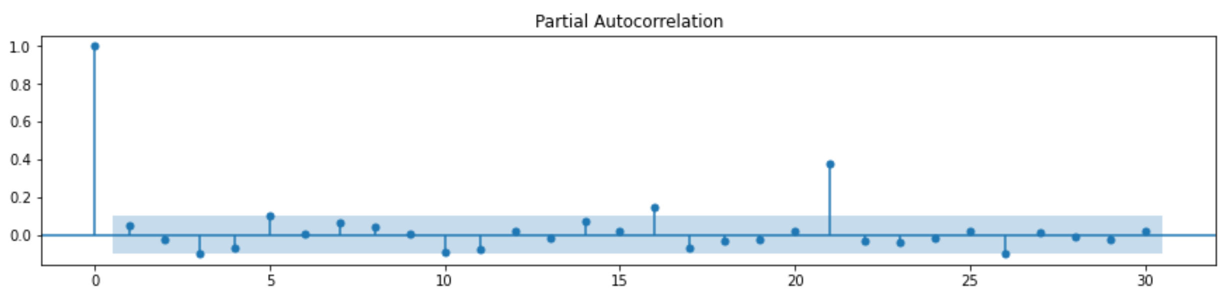
5. ARMA MODELING

Because ADF test shows data was stationary and does not have a unit root, we can proceed with ARMA model setup.

ACF and PACF will be used to determine the parameters.

In [189]:
```
# ACF AND PACF

rcParams['figure.figsize'] = 15, 3
plot_acf(df_price_date['SALE PRICE'], lags=30, alpha=0.05);

rcParams['figure.figsize'] = 15, 3
plot_pacf(df_price_date['SALE PRICE'], lags=30, alpha=0.05);
```





In [190]:
```
# Instantiate & fit model with statsmodels
#p = num lags - ACF
p = 5

# q = lagged forecast errors - PACF
q = 5

#d = number of differences - will compare differenced data RMSE with this model i
# d=


# Fitting ARMA model and summary
ar = ARMA(df_price_date['SALE PRICE'],(p,q)).fit()
ar.summary()
```

Out[190]:

ARMA Model Results

Dep. Variable:	SALE PRICE	**No. Observations:**	365
Model:	ARMA(5, 5)	**Log Likelihood**	-5588.405
Method:	css-mle	**S.D. of innovations**	1062824.560
Date:	Sun, 20 Jun 2021	**AIC**	11200.810
Time:	14:36:00	**BIC**	11247.609
Sample:	04-01-2020	**HQIC**	11219.408
	- 03-31-2021		

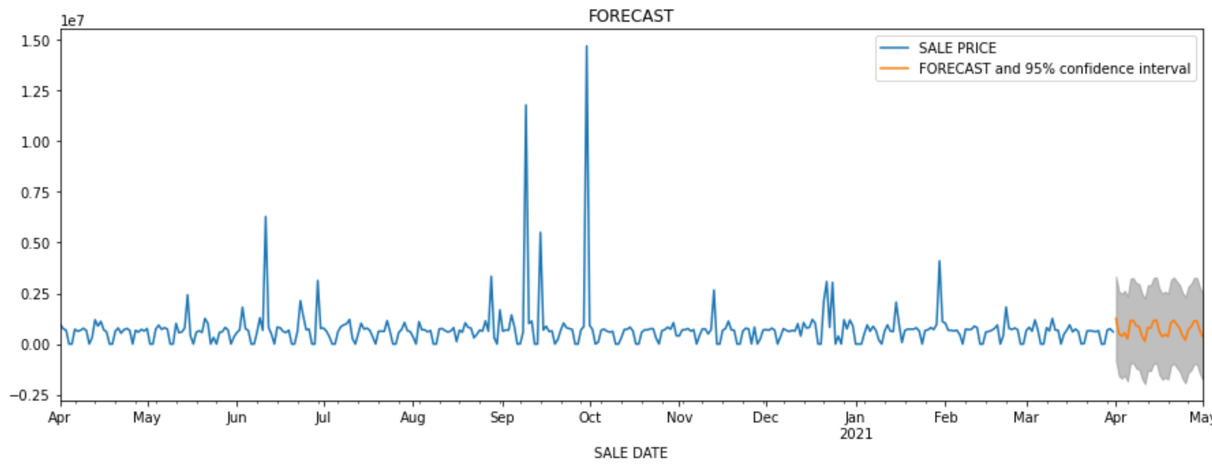
	coef	std err	z	P>|z|	[0.025	0.975]
const	7.33e+05	5.79e+04	12.654	0.000	6.2e+05	8.47e+05
ar.L1.SALE PRICE	-0.9124	nan	nan	nan	nan	nan
ar.L2.SALE PRICE	-0.3866	nan	nan	nan	nan	nan
ar.L3.SALE PRICE	-0.4637	nan	nan	nan	nan	nan
ar.L4.SALE PRICE	-0.9571	nan	nan	nan	nan	nan
ar.L5.SALE PRICE	-0.9011	nan	nan	nan	nan	nan
ma.L1.SALE PRICE	0.9621	0.022	44.568	0.000	0.920	1.004
ma.L2.SALE PRICE	0.4438	0.029	15.290	0.000	0.387	0.501
ma.L3.SALE PRICE	0.4430	0.031	14.354	0.000	0.382	0.503
ma.L4.SALE PRICE	0.9637	0.038	25.625	0.000	0.890	1.037
ma.L5.SALE PRICE	0.9984	0.030	33.307	0.000	0.940	1.057

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	0.6239	-0.7816j	1.0001	-0.1428
AR.2	0.6239	+0.7816j	1.0001	0.1428
AR.3	-0.6407	-0.8175j	1.0387	-0.3558
AR.4	-0.6407	+0.8175j	1.0387	0.3558
AR.5	-1.0285	-0.0000j	1.0285	-0.5000
MA.1	0.6256	-0.7802j	1.0000	-0.1424
MA.2	0.6256	+0.7802j	1.0000	0.1424
MA.3	-0.6074	-0.7944j	1.0000	-0.3539
MA.4	-0.6074	+0.7944j	1.0000	0.3539
MA.5	-1.0016	-0.0000j	1.0016	-0.5000

In [191]:
```
#plot of ARMA model
plt.figure(figsize=(20,10))
fig, ax = plt.subplots()
# ax = df_price_date['SALE_PRICE_LOGGED'].plot(ax=ax, title='FORECAST')
ax = df_price_date['SALE PRICE'].plot(ax=ax, title='FORECAST',figsize=(15,5))
fig = ar.plot_predict(365, 395, dynamic=True, ax=ax, plot_insample=True)

handles, labels = ax.get_legend_handles_labels()
labels = ['SALE PRICE', 'FORECAST and 95% confidence interval']
ax.legend(handles, labels)

plt.show()
```

```
<Figure size 1440x720 with 0 Axes>
```

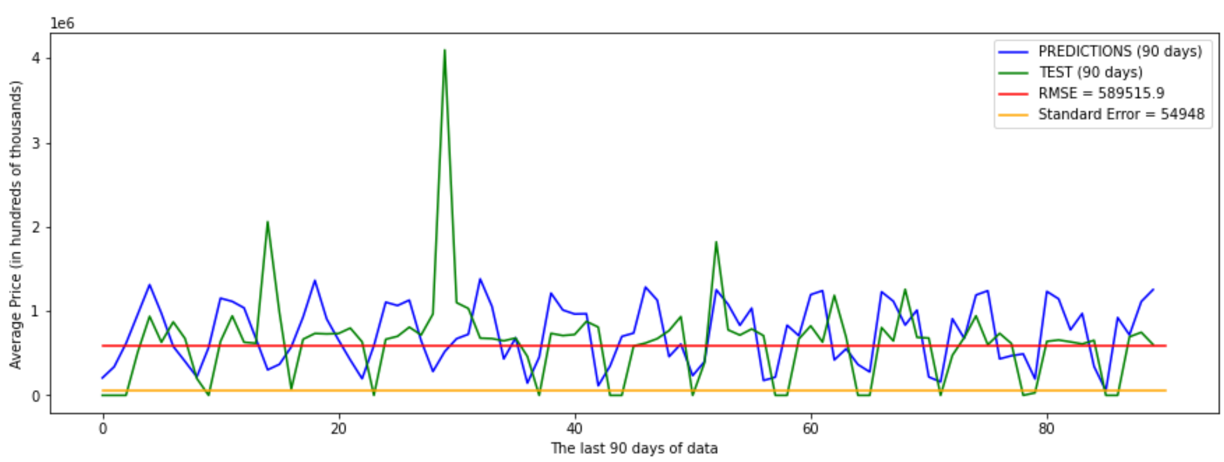


6. Error analysis of ARMA model

In [192]:
```
predictions = list(ar.predict(276, 365))
test = list(df_price_date['SALE PRICE'][275:365])

print("\033[1m" + '\033[4m'+ 'Length of Predictions' + "\033[0m", ': ', len(pred
print("\033[1m" + '\033[4m'+ 'Length of Test data' + "\033[0m", ': ', len(test))

#RMSE
mse = sklearn.metrics.mean_squared_error(test, predictions)
rmse = math.sqrt(mse)
print("\033[1m" + '\033[4m'+ 'RMSE' + "\033[0m", ': ', rmse)

#standard error
stderr = ar.bse.const
print("\033[1m" + '\033[4m'+ 'Standard Error' + "\033[0m", ': ', stderr)

#plot of all
plt.figure(figsize=(15,5))
plt.plot(predictions, label='PREDICTIONS (90 days)', color='blue')
plt.plot(test, label='TEST (90 days)', color='green')

x=[0,90]
y=[rmse,rmse]
plt.plot(x,y, label='RMSE = 589515.9', color='red')

x=[0,90]
y=[stderr,stderr]
plt.plot(x,y, label='Standard Error = 54948', color='orange')

plt.legend(loc='best')
plt.xlabel("The last 90 days of data")
plt.ylabel("Average Price (in hundreds of thousands)")
```

```
Length of Predictions :  90
Length of Test data :  90
RMSE :  589515.9314535034
Standard Error :  57927.81077050332
```

Out[192]: `Text(0, 0.5, 'Average Price (in hundreds of thousands)')`



Observation:

RMSE is not too high or low. Lower values of RMSE indicate better fit. I believe this is a good range and model fits well.

- RMSE is 589515.9314535034
- Standard error is 57927.81077050332

6a. Testing parameters to improve ARMA model

- I will try p of 3 per ACF
- I will try q of 3 per PACF
- I will try d = 7 to difference weekly

In [193]:
```
# Instantiate & fit model with statsmodels
#p = num lags - ACF
p = 3

# q = lagged forecast errors - PACF
q = 3

#d = number of differences
d = 7


# Fitting ARMA model and summary
ar1 = ARMA(df_price_date['SALE PRICE'],(p,d,q)).fit()
ar.summary()
```

Out[193]:

ARMA Model Results

Dep. Variable:	SALE PRICE	**No. Observations:**	365
Model:	ARMA(5, 5)	**Log Likelihood**	-5588.405
Method:	css-mle	**S.D. of innovations**	1062824.560
Date:	Sun, 20 Jun 2021	**AIC**	11200.810
Time:	14:36:16	**BIC**	11247.609
Sample:	04-01-2020	**HQIC**	11219.408
	- 03-31-2021		

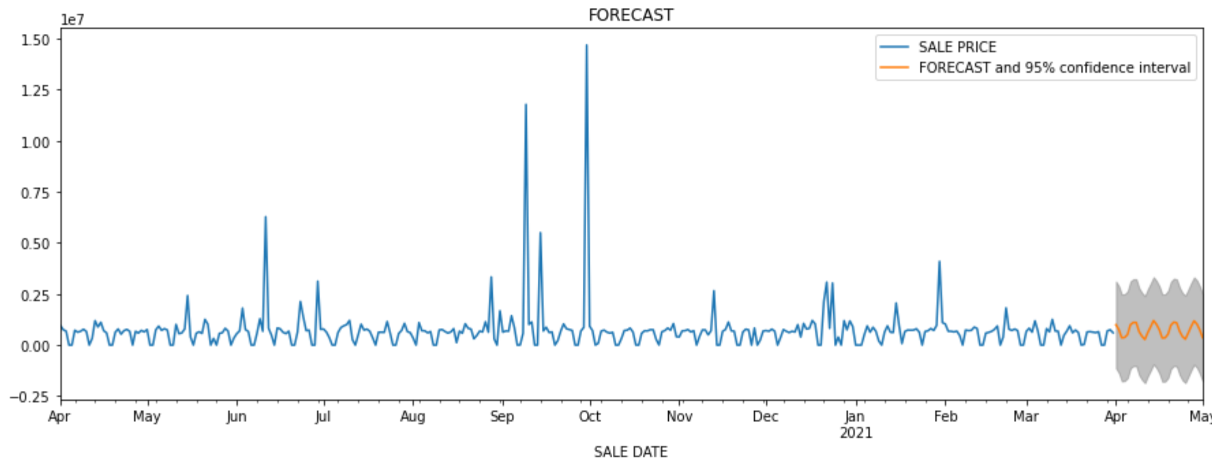
	coef	std err	z	P>|z|	[0.025	0.975]
const	7.33e+05	5.79e+04	12.654	0.000	6.2e+05	8.47e+05
ar.L1.SALE PRICE	-0.9124	nan	nan	nan	nan	nan
ar.L2.SALE PRICE	-0.3866	nan	nan	nan	nan	nan
ar.L3.SALE PRICE	-0.4637	nan	nan	nan	nan	nan
ar.L4.SALE PRICE	-0.9571	nan	nan	nan	nan	nan
ar.L5.SALE PRICE	-0.9011	nan	nan	nan	nan	nan
ma.L1.SALE PRICE	0.9621	0.022	44.568	0.000	0.920	1.004
ma.L2.SALE PRICE	0.4438	0.029	15.290	0.000	0.387	0.501
ma.L3.SALE PRICE	0.4430	0.031	14.354	0.000	0.382	0.503
ma.L4.SALE PRICE	0.9637	0.038	25.625	0.000	0.890	1.037
ma.L5.SALE PRICE	0.9984	0.030	33.307	0.000	0.940	1.057

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	0.6239	-0.7816j	1.0001	-0.1428
AR.2	0.6239	+0.7816j	1.0001	0.1428
AR.3	-0.6407	-0.8175j	1.0387	-0.3558
AR.4	-0.6407	+0.8175j	1.0387	0.3558
AR.5	-1.0285	-0.0000j	1.0285	-0.5000
MA.1	0.6256	-0.7802j	1.0000	-0.1424
MA.2	0.6256	+0.7802j	1.0000	0.1424
MA.3	-0.6074	-0.7944j	1.0000	-0.3539
MA.4	-0.6074	+0.7944j	1.0000	0.3539
MA.5	-1.0016	-0.0000j	1.0016	-0.5000

In [194]:
```
#plot of ARMA model
plt.figure(figsize=(20,10))
fig, ax = plt.subplots()
# ax = df_price_date['SALE_PRICE_LOGGED'].plot(ax=ax, title='FORECAST')
ax = df_price_date['SALE PRICE'].plot(ax=ax, title='FORECAST',figsize=(15,5))
fig = ar1.plot_predict(365, 395, dynamic=True, ax=ax, plot_insample=True)

handles, labels = ax.get_legend_handles_labels()
labels = ['SALE PRICE', 'FORECAST and 95% confidence interval']
ax.legend(handles, labels)

plt.show()
```

```
<Figure size 1440x720 with 0 Axes>
```

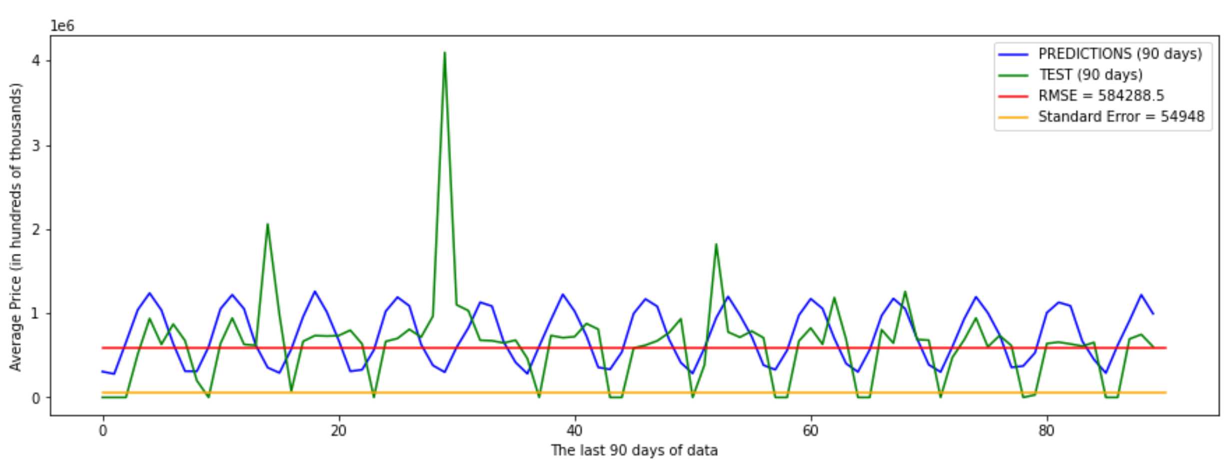


6a - Error Analysis of new model

In [195]:
```
predictions = list(ar1.predict(276, 365))
test = list(df_price_date['SALE PRICE'][275:365])

print("\033[1m" + '\033[4m'+ 'Length of Predictions' + "\033[0m", ': ', len(pred:
print("\033[1m" + '\033[4m'+ 'Length of Test data' + "\033[0m", ': ', len(test))

#RMSE
mse = sklearn.metrics.mean_squared_error(test, predictions)
rmse = math.sqrt(mse)
print("\033[1m" + '\033[4m'+ 'RMSE' + "\033[0m", ': ', rmse)

#standard error
stderr = ar1.bse.const
print("\033[1m" + '\033[4m'+ 'Standard Error' + "\033[0m", ': ', stderr)

#plot of all
plt.figure(figsize=(15,5))
plt.plot(predictions, label='PREDICTIONS (90 days)', color='blue')
plt.plot(test, label='TEST (90 days)', color='green')

x=[0,90]
y=[rmse,rmse]
plt.plot(x,y, label='RMSE = 584288.5', color='red')

x=[0,90]
y=[stderr,stderr]
plt.plot(x,y, label='Standard Error = 54948', color='orange')

plt.legend(loc='best')
plt.xlabel("The last 90 days of data")
plt.ylabel("Average Price (in hundreds of thousands)")
```

```
Length of Predictions :  90
Length of Test data :  90
RMSE :  584288.5171829152
Standard Error :  54947.9592801162
```

Out[195]: Text(0, 0.5, 'Average Price (in hundreds of thousands)')



Observation:

- We see that RMSE looks acceptable. Not too high and not too low. Indicates a good fit.
- Compared to the origional model RMSE is also lower. We can use this model rather than the origional model.

7. Comparing predictions with fresh data from June 2021 dataset (4/1/2021 - 4/31/2021)

Here I do the following:

1. Load data with only specific columns to borough
    - Sale price
    - Sale data

- Borough

2. Clean the data to get rid of issues when plotting/calculating errors
    - This dataset was in .csv format, different from the origional rolling dataset
    - I had to filter the data and change columns from strings to int
    - Change 'SALE DATE' to datetime
    - Resample the data to match origional rolling data
        - aggregate by day

3. Plot the new data versus the predicted data and calculate RMSE

In [196]:
```
#Loading the data and reset the index

excel_df = pd.read_csv('NYC_Citywide_Rolling_Calendar_Sales.csv', usecols=['BOROU
excel_df = excel_df[excel_df['BOROUGH']=='QUEENS']
excel_df.reset_index(drop=True, inplace=True)
```

In [197]:
```
#Fixes to the data

excel_df['SALE PRICE'] = excel_df['SALE PRICE'].str.replace(',','')
excel_df['SALE PRICE'] = excel_df['SALE PRICE'].astype(int)
excel_df['SALE DATE'] = pd.to_datetime(excel_df['SALE DATE'])
```

In [198]:
```
#Create new dataframe and aggregate to days like I did with origional rolling dat

excel_price_date = pd.DataFrame(excel_df, columns=['SALE DATE', 'SALE PRICE'])
excel_price_date = excel_price_date.set_index('SALE DATE')

#aggregate by day
excel_price_date = excel_price_date.resample('D').mean()
```

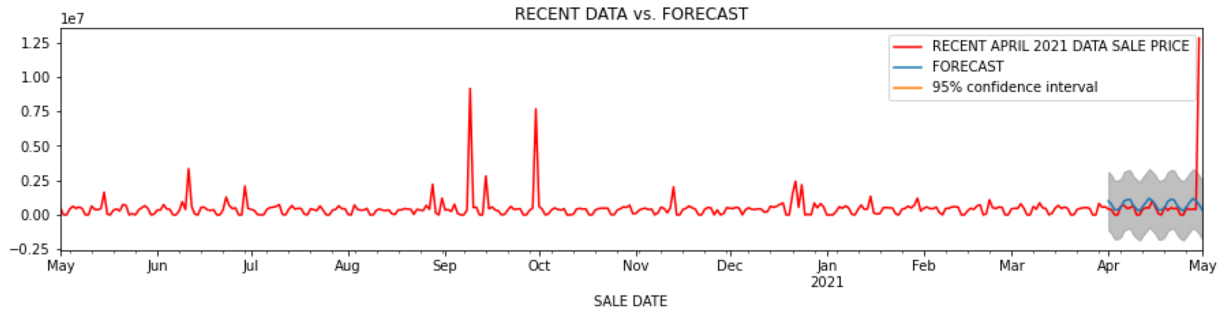
In [199]:
```
# Again, if I drop NaN here, it will change the dates which will affect the ploti
# I decide to fillna(0) similar to origional rolling data

excel_price_date = excel_price_date.fillna(0)
```

In [200]:
```
excel_price_date.head()
```

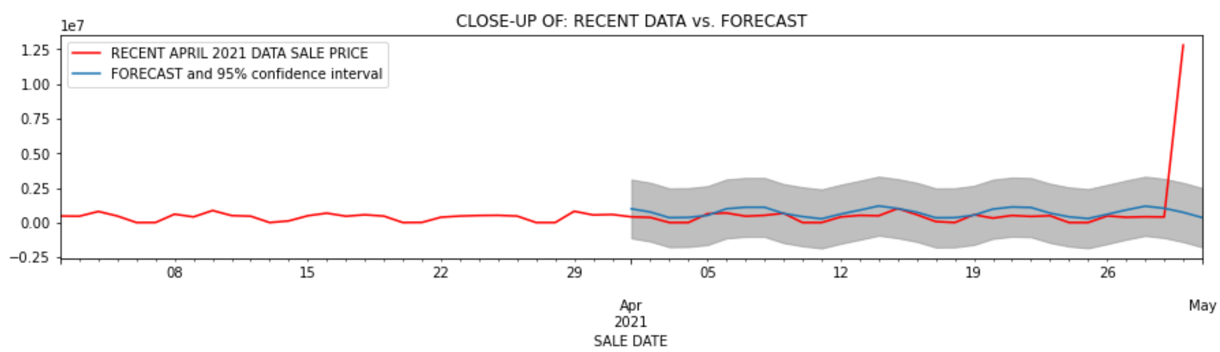
Out[200]:

	SALE PRICE
SALE DATE	
2020-05-01	512087.741379
2020-05-02	0.000000
2020-05-03	0.000000
2020-05-04	409406.285714
2020-05-05	618745.238095

In [201]:
```
# Plotting the data versus the ar.plot_predict values

fig, ax = plt.subplots()

ax = excel_price_date['SALE PRICE'].plot(title='RECENT DATA vs. FORECAST', color
fig = ar1.plot_predict(365, 395, dynamic=True, ax = ax, plot_insample=True)

handles, labels = ax.get_legend_handles_labels()
labels = ['RECENT APRIL 2021 DATA SALE PRICE', 'FORECAST','95% confidence interva
ax.legend(handles, labels)

plt.show()
```



In [206]:
```
# Plotting the data versus the ar.plot_predict values
#Here I do a close up

fig, ax = plt.subplots()

ax = excel_price_date['SALE PRICE'][305:365].plot(title='CLOSE-UP OF: RECENT DAT
fig = ar1.plot_predict(365, 395, dynamic=True, ax = ax, plot_insample=True)

handles, labels = ax.get_legend_handles_labels()
labels = ['RECENT APRIL 2021 DATA SALE PRICE', 'FORECAST and 95% confidence inte
ax.legend(handles, labels)

plt.show()
```



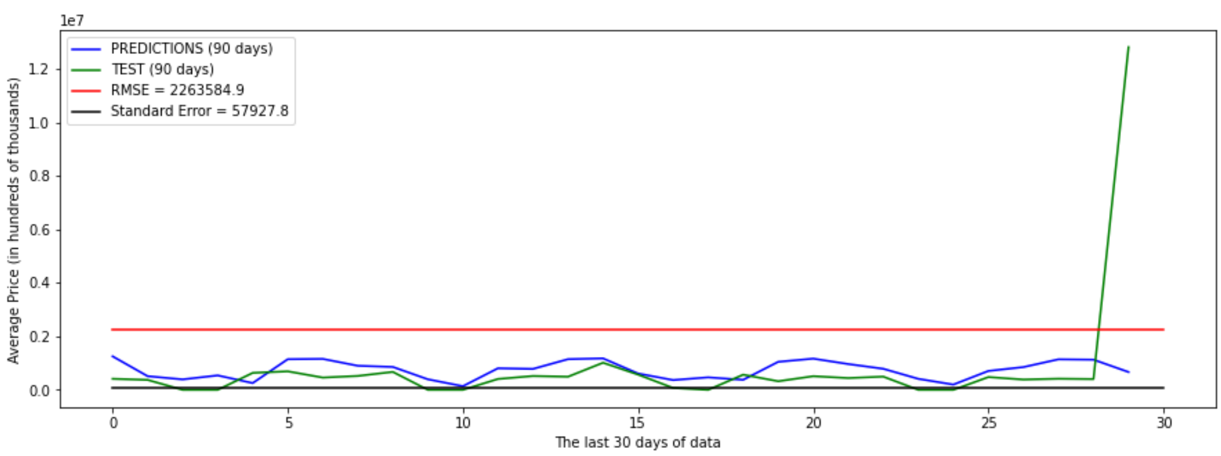
Observation

We see that the model has a decent fit with the test data of the last 30 days. However, the outlier will affect the error

In [203]:
```
#RMSE, Standard error

# last 30 days of data
predictions = list(ar.predict(365, 394))
test = list(excel_price_date['SALE PRICE'][335:365])

print("\033[1m" + '\033[4m'+ 'Length of Predictions' + "\033[0m", ': ', len(pred
print("\033[1m" + '\033[4m'+ 'Length of Test data' + "\033[0m", ': ', len(test))

#RMSE
mse = sklearn.metrics.mean_squared_error(test, predictions)
rmse = math.sqrt(mse)
print("\033[1m" + '\033[4m'+ 'RMSE' + "\033[0m", ': ', rmse)

#standard error
stderr = ar1.bse.const
print("\033[1m" + '\033[4m'+ 'Standard Error' + "\033[0m", ': ', stderr)

#plot of all
plt.figure(figsize=(15,5))
plt.plot(predictions, label='PREDICTIONS (90 days)', color='blue')
plt.plot(test, label='TEST (90 days)', color='green')

x=[0,30]
y=[rmse,rmse]
plt.plot(x,y, label='RMSE = 2263584.9', color='red')

x=[0,30]
y=[stderr,stderr]
plt.plot(x,y, label='Standard Error = 57927.8', color='black')

plt.legend(loc='best')
plt.xlabel("The last 30 days of data")
plt.ylabel("Average Price (in hundreds of thousands)")
```

```
Length of Predictions :  30
Length of Test data :  30
RMSE :  2263584.943411371
Standard Error :  54947.9592801162
```

Out[203]: Text(0, 0.5, 'Average Price (in hundreds of thousands)')



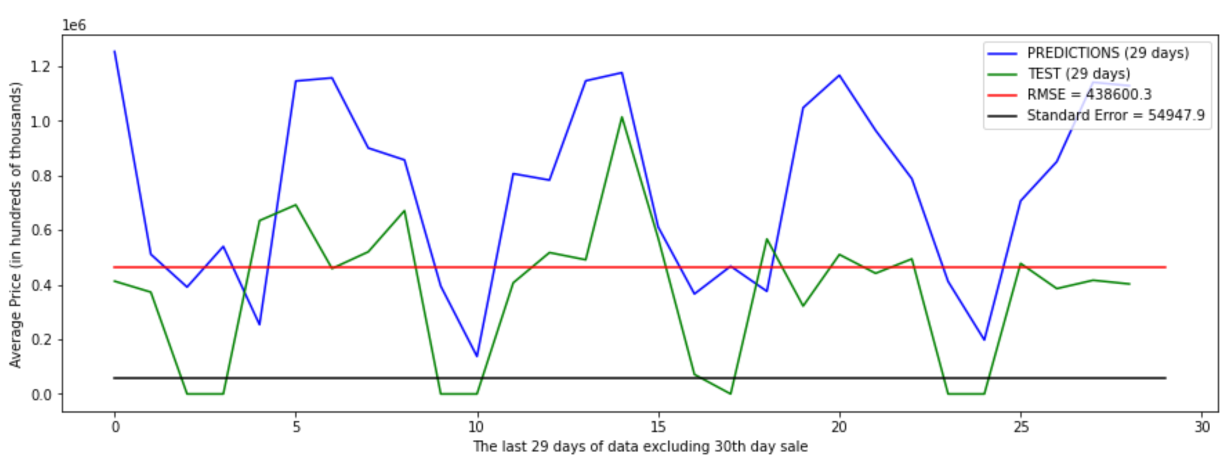
Observation:

- We see that RMSE is much higher due to a large sale that occured near the end of the month.
- Below I remove that outlier and re-check RMSE and Standard error

In [204]:
```
#Predicting ERROR when we remove the last day to get rid of the huge sale, we se


predictions = list(ar.predict(365, 393))
test = list(excel_price_date['SALE PRICE'][335:364])

print("\033[1m" + '\033[4m'+ 'Length of Predictions' + "\033[0m", ': ', len(pred
print("\033[1m" + '\033[4m'+ 'Length of Test data' + "\033[0m", ': ', len(test))

#RMSE
mse = sklearn.metrics.mean_squared_error(test, predictions)
rmse = math.sqrt(mse)
print("\033[1m" + '\033[4m'+ 'RMSE' + "\033[0m", ': ', rmse)

#standard error
stderr = ar.bse.const
print("\033[1m" + '\033[4m'+ 'Standard Error' + "\033[0m", ': ', stderr)

#plot of all
plt.figure(figsize=(15,5))
plt.plot(predictions, label='PREDICTIONS (29 days)', color='blue')
plt.plot(test, label='TEST (29 days)', color='green')

x=[0,29]
y=[rmse,rmse]
plt.plot(x,y, label='RMSE = 438600.3', color='red')

x=[0,29]
y=[stderr,stderr]
plt.plot(x,y, label='Standard Error = 54947.9', color='black')

plt.legend(loc='best')
plt.xlabel("The last 29 days of data excluding 30th day sale")
plt.ylabel("Average Price (in hundreds of thousands)")
```

Length of Predictions : 29
Length of Test data : 29
RMSE : 463989.1499686349
Standard Error : 57927.81077050332

Out[204]: Text(0, 0.5, 'Average Price (in hundreds of thousands)')



Observation

- RMSE is lower and so is the standard error than before. This can indicate a good fit. RMSE is not too high.
- With the outlier removed, RMSE is lower, indicates good fit

8. Observations/Conclusions/Recommendations

```
1. The point of this analysis was to see if the borough was good to inve
st in
2. Based on the model:
    - We can enter to buy or exit to sell based on when the market will
 do well
3. The borough sales look predictable
4. There are unpredictable building sales which are very large amounts i
n the millions to tens of millions
5. We can look at the top 10 building permit heavy locations further
```