Analysis of Rolling Sales Data - Staten Island (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 [261]:

```
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 [262]:
```
#Loading prepped data
df = pd.read_csv('datasets/rollingsales_statenisland.xls_prepped_bare.csv')
df.reset_index(drop=True, inplace=True)
df.sort_values('SALE DATE')
```

Out[262]:

	TAX CLASS AT PRESENT	ZIP CODE	SALE PRICE	SALE DATE
613	1	10314.0	575000	2020-04-01
2164	1	10305.0	580000	2020-04-01
2222	1	10306.0	547000	2020-04-02
547	1	10314.0	500000	2020-04-02
3446	1	10309.0	905000	2020-04-02
...	...	...	...	...
3118	1	10302.0	379999	2021-03-15
2376	2	10301.0	270000	2021-03-18
331	2	10305.0	255000	2021-03-24
3372	1	10305.0	435000	2021-03-29
3329	1	10305.0	435000	2021-03-29

4515 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.
    - 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 [263]:
```
# 1. Convert 'SALE DATE' column to datetime format

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

In [264]:
```
# 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[264]:

SALE DATE	SALE PRICE
2020-10-02	315000
2020-06-24	450000
2020-07-02	525000
2021-01-21	455000
2020-10-15	720000

In [265]:
```
# 3. Group the sales data by daily average

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

In [266]:
```
# 4. We see here number of rows went down 258. Why wasn't it 365 rows to represen
df_price_date.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 363 entries, 2020-04-01 to 2021-03-29
Freq: D
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   SALE PRICE  258 non-null    float64
dtypes: float64(1)
memory usage: 5.7 KB
```

In [267]:
```
#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[267]: 105

In [268]:
```
# 4. Instead of dropping the rows, I decided to fill NaN with 0 to reflect no sa
df_price_date['SALE PRICE'].fillna(0, inplace=True)
df_price_date
```

Out[268]:

SALE DATE	SALE PRICE
2020-04-01	577500.000000
2020-04-02	650666.666667
2020-04-03	519414.285714
2020-04-04	0.000000
2020-04-05	0.000000
...	...
2021-03-25	0.000000
2021-03-26	0.000000
2021-03-27	0.000000
2021-03-28	0.000000
2021-03-29	435000.000000

363 rows × 1 columns

In [269]:
```
# 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
```

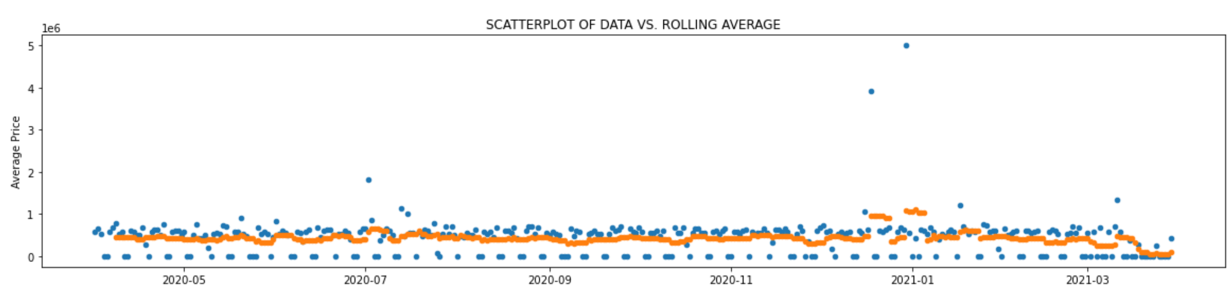
Out[269]:

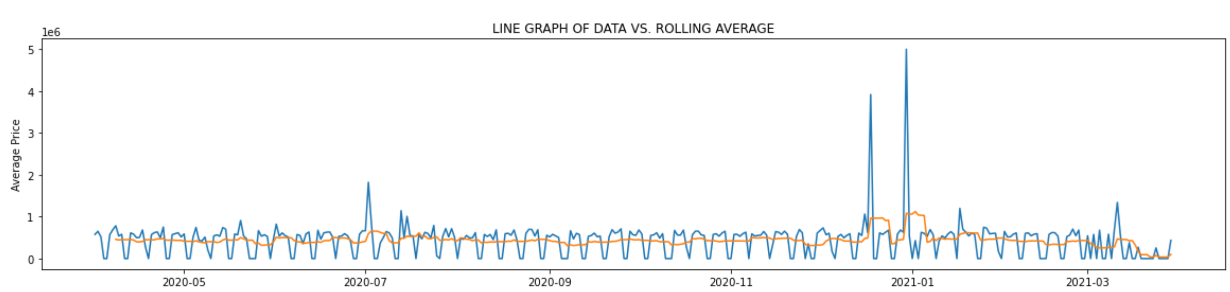
	SALE PRICE	roll_avg
SALE DATE		
2020-04-01	577500.000000	NaN
2020-04-02	650666.666667	NaN
2020-04-03	519414.285714	NaN
2020-04-04	0.000000	NaN
2020-04-05	0.000000	NaN
...	...	...
2021-03-25	0.000000	36428.571429
2021-03-26	0.000000	36428.571429
2021-03-27	0.000000	36428.571429
2021-03-28	0.000000	36428.571429
2021-03-29	435000.000000	98571.428571

363 rows × 2 columns

In [270]:
```
#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[270]: `Text(0, 0.5, 'Average Price')`





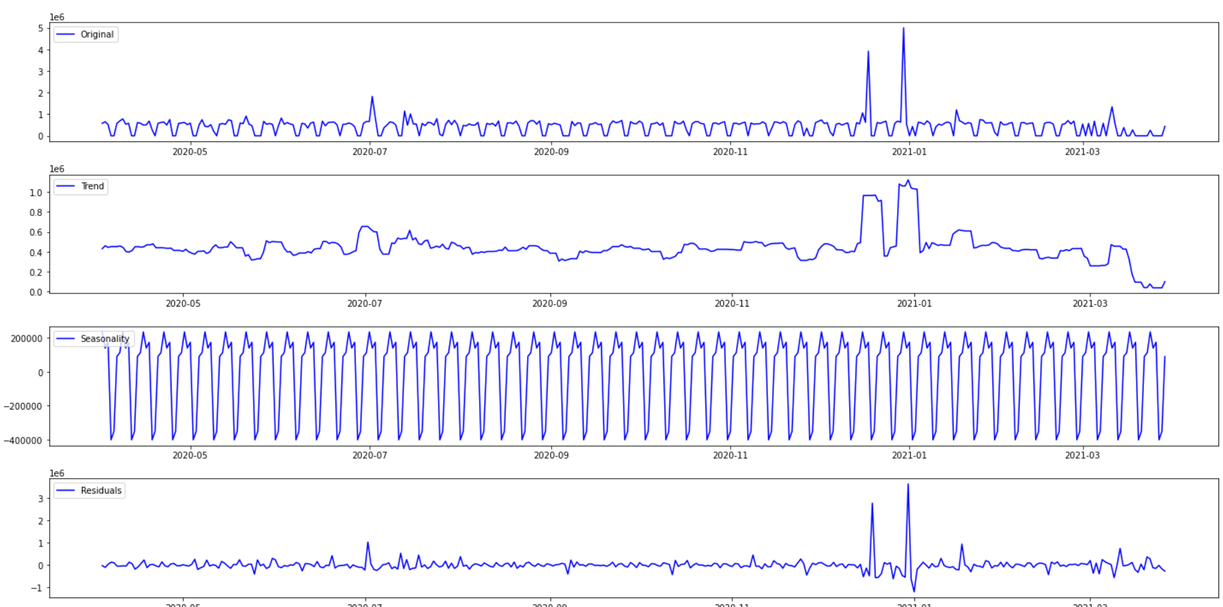
Observation

```
- 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
- Near the end of the last month, there is a drop in sales
```

In [271]:
```
# 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 [272]:
```
register_matplotlib_converters()
```

In [273]:
```
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:

- 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 [274]:

```python
# 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)
```

```
(-2.077159536291176, 0.25379748112932804, 13, 349, {'1%': -3.449226932880019,
'5%': -2.869857365438656, '10%': -2.571201085130664}, 9854.55967670128)

Test Statistic                  -2.077160
p-value                          0.253797
#Lags Used                      13.000000
Number of Observations Used    349.000000
Critical Value (1%)             -3.449227
Critical Value (5%)             -2.869857
Critical Value (10%)            -2.571201
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 **-2.077160**, this is greater than the critical values for 1% and 5%.
    - *We fail to reject the null hypothesis that the time series is not stationary!*

P-Value Analysis

- Our current p-value is **0.253797**
    - *This means: p-value > 0.05: Fail to reject the null hypothesis (H0), the data has a unit root and is non-stationary.*

4a. Inducing Stationarity

In [275]:
```
df_price_date_diff= df_price_date.diff(periods=7)
df_price_date_diff
```

Out[275]:

	SALE PRICE	roll_avg
SALE DATE		
2020-04-01	NaN	NaN
2020-04-02	NaN	NaN
2020-04-03	NaN	NaN
2020-04-04	NaN	NaN
2020-04-05	NaN	NaN
...	...	...
2021-03-25	-270000.0	-137499.857143
2021-03-26	0.0	-56428.428571
2021-03-27	0.0	-56428.428571
2021-03-28	0.0	-56428.428571
2021-03-29	435000.0	60000.000000

363 rows × 2 columns

In [276]:
```
df_price_date_diff.dropna(inplace=True)
```

In [277]:
```
df_price_date_diff.index.unique()
```

Out[277]:
```
DatetimeIndex(['2020-04-14', '2020-04-15', '2020-04-16', '2020-04-17',
               '2020-04-18', '2020-04-19', '2020-04-20', '2020-04-21',
               '2020-04-22', '2020-04-23',
               ...
               '2021-03-20', '2021-03-21', '2021-03-22', '2021-03-23',
               '2021-03-24', '2021-03-25', '2021-03-26', '2021-03-27',
               '2021-03-28', '2021-03-29'],
              dtype='datetime64[ns]', name='SALE DATE', length=350, freq=None)
```

In [278]:
```
dftest = adfuller(df_price_date_diff['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)
```

```
(-5.182300371217239, 9.55386441131336e-06, 16, 333, {'1%': -3.450141065277327,
'5%': -2.870258846235788, '10%': -2.571415151457764}, 9517.916871772755)

Test Statistic                 -5.182300
p-value                         0.000010
#Lags Used                     16.000000
Number of Observations Used   333.000000
Critical Value (1%)            -3.450141
Critical Value (5%)            -2.870259
Critical Value (10%)           -2.571415
dtype: float64
```

Results of ADF Test

Test Statistic vs. Critical Values

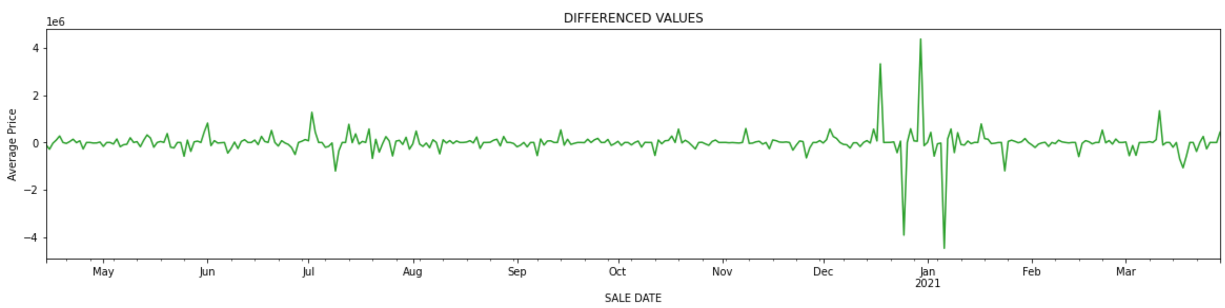
- Initial test shows Test Statistic of **-5.182300**, this is greater than the critical values for 1% and 5%.
    - *We **REJECT** the null hypothesis! The data does not have a unit root and is stationary*

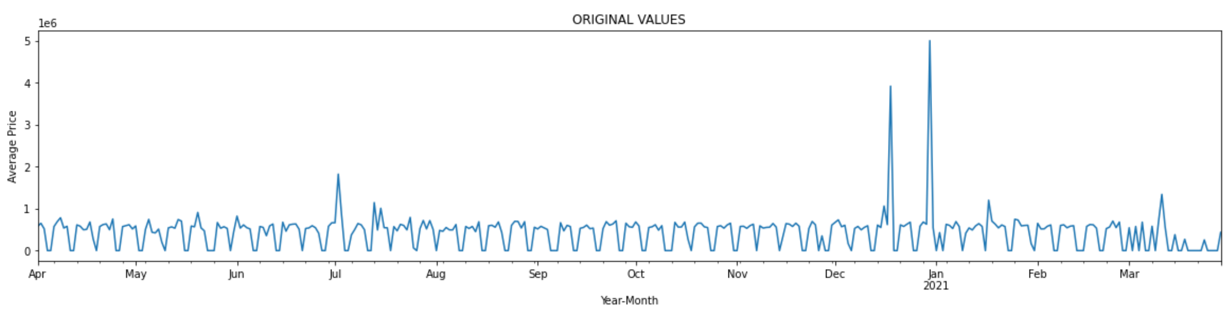
P-Value Analysis

- Our current p-value is **0.000010**
  - *This means: p-value <= 0.05:*
  - *We **REJECT** the null hypothesis! The data does not have a unit root and is stationary*

In [279]:
```
plt.figure(figsize=(20, 4))

plt.xlabel("Year-Month")
plt.ylabel("Average Price")

#s=20 to keep dots small in size
df_price_date_diff['SALE PRICE'].plot(color="tab:green", title="DIFFERENCED VALUI

plt.figure(figsize=(20, 4))
plt.title("LINE GRAPH OF DATA VS. ROLLING AVERAGE")
df_price_date['SALE PRICE'].plot(color="tab:blue", title="ORIGINAL VALUES");
plt.xlabel("Year-Month")
plt.ylabel("Average Price")
```

Out[279]: Text(0, 0.5, 'Average Price')



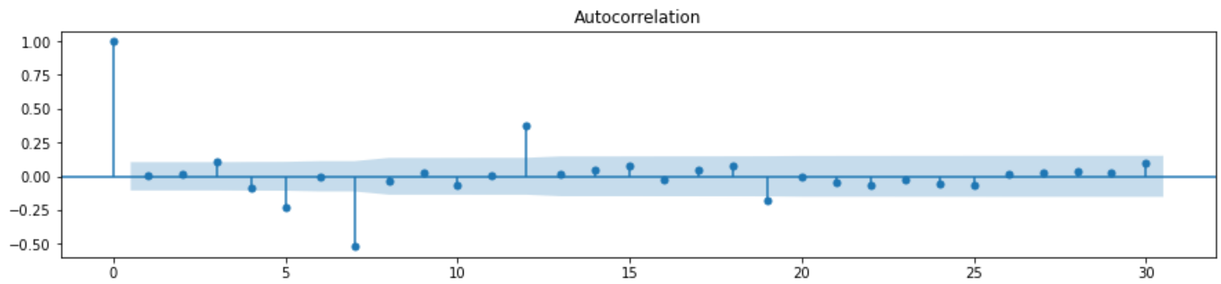


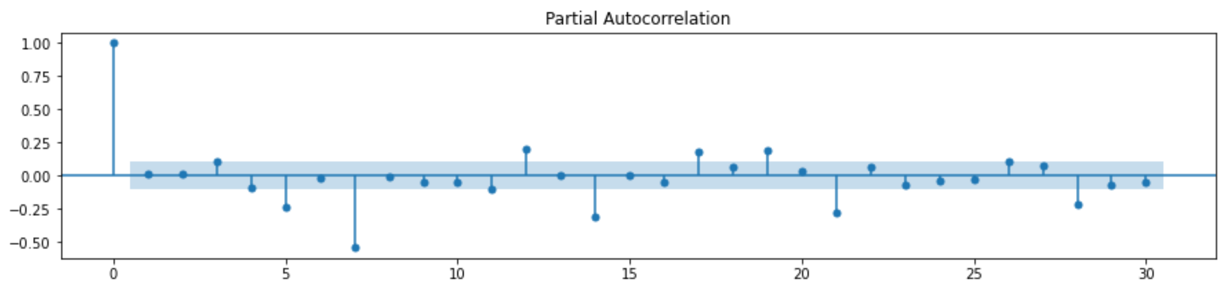
5. ARMA MODELING

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 [280]:
```
# ACF AND PACF

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

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





In [281]:
```
df_price_date_diff  = df_price_date_diff.drop(columns=['roll_avg'])
```

In [282]:
```
# Instantiate & fit model with statsmodels
#p = num lags - ACF
p = 9

# q = lagged forecast errors - PACF
q = 7

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

Out[282]:

ARMA Model Results

Dep. Variable:	SALE PRICE	**No. Observations:**	350
Model:	ARMA(9, 7)	**Log Likelihood**	-4966.356
Method:	css-mle	**S.D. of innovations**	339269.774
Date:	Sun, 20 Jun 2021	**AIC**	9968.712
Time:	20:05:55	**BIC**	10038.155
Sample:	04-14-2020	**HQIC**	9996.352
	- 03-29-2021		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-7170.9984	8893.773	-0.806	0.420	-2.46e+04	1.03e+04
ar.L1.SALE PRICE	-0.1311	0.073	-1.806	0.071	-0.273	0.011
ar.L2.SALE PRICE	-0.1018	0.071	-1.441	0.149	-0.240	0.037
ar.L3.SALE PRICE	-0.0039	0.066	-0.059	0.953	-0.133	0.125
ar.L4.SALE PRICE	-0.1398	0.062	-2.268	0.023	-0.261	-0.019
ar.L5.SALE PRICE	-0.2069	0.060	-3.434	0.001	-0.325	-0.089
ar.L6.SALE PRICE	-0.0739	0.061	-1.211	0.226	-0.193	0.046
ar.L7.SALE PRICE	-0.1509	0.060	-2.514	0.012	-0.269	-0.033
ar.L8.SALE PRICE	-0.1087	0.064	-1.694	0.090	-0.234	0.017
ar.L9.SALE PRICE	-0.0622	0.062	-1.005	0.315	-0.184	0.059
ma.L1.SALE PRICE	0.1578	0.040	3.905	0.000	0.079	0.237
ma.L2.SALE PRICE	0.0994	0.049	2.026	0.043	0.003	0.196
ma.L3.SALE PRICE	0.1116	0.049	2.263	0.024	0.015	0.208
ma.L4.SALE PRICE	0.1532	0.048	3.159	0.002	0.058	0.248
ma.L5.SALE PRICE	0.1735	0.044	3.899	0.000	0.086	0.261
ma.L6.SALE PRICE	0.1177	0.036	3.305	0.001	0.048	0.188
ma.L7.SALE PRICE	-0.8636	0.037	-23.364	0.000	-0.936	-0.791

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.0017	-0.5620j	1.1486	-0.0814
AR.2	1.0017	+0.5620j	1.1486	0.0814
AR.3	0.5136	-1.2602j	1.3608	-0.1884
AR.4	0.5136	+1.2602j	1.3608	0.1884
AR.5	-1.3540	-0.0000j	1.3540	-0.5000
AR.6	-0.5401	-1.2211j	1.3352	-0.3163
AR.7	-0.5401	+1.2211j	1.3352	0.3163
AR.8	-1.1715	-1.1630j	1.6507	-0.3756
AR.9	-1.1715	+1.1630j	1.6507	0.3756
MA.1	-0.9033	-0.4290j	1.0000	-0.4294
MA.2	-0.9033	+0.4290j	1.0000	0.4294

MA.3	-0.2379	-0.9713j	1.0000	-0.2882
MA.4	-0.2379	+0.9713j	1.0000	0.2882
MA.5	0.6304	-0.7763j	1.0000	-0.1414
MA.6	0.6304	+0.7763j	1.0000	0.1414
MA.7	1.1578	-0.0000j	1.1578	-0.0000

In [283]:

```
df_price_date_diff
```

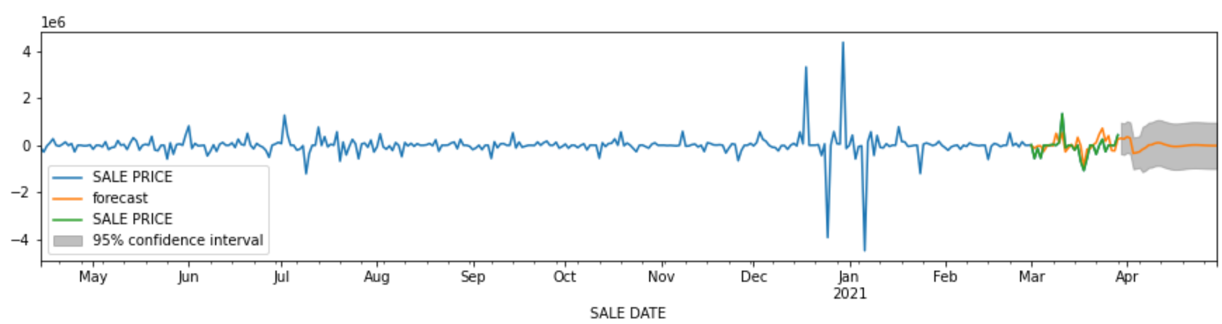
Out[283]:

SALE DATE	SALE PRICE
2020-04-14	-106773.777778
2020-04-15	-285580.000000
2020-04-16	-31959.818182
2020-04-17	102306.611111
2020-04-18	275000.000000
...	...
2021-03-25	-270000.000000
2021-03-26	0.000000
2021-03-27	0.000000
2021-03-28	0.000000
2021-03-29	435000.000000

350 rows × 1 columns

In [284]:

```
fig, ax = plt.subplots()
ax = df_price_date_diff.loc['2020-04-14':].plot(ax=ax)
fig = ar.plot_predict('2021-3-01', '2021-04-30', dynamic=False, ax=ax, plot_insam
plt.show()
```

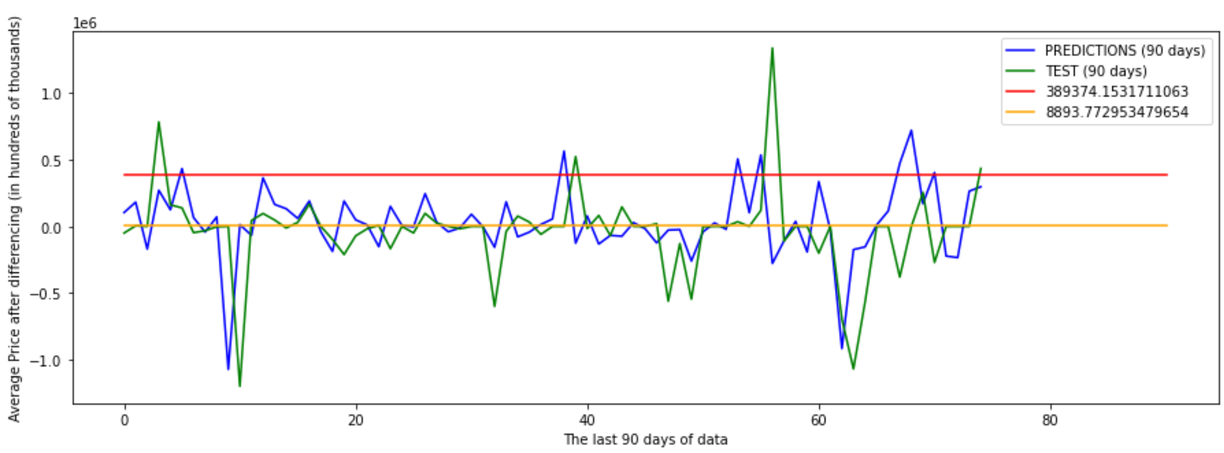


6. Error analysis of ARMA model

In [285]:
```
predictions = list(ar.predict(276, 350))
test = list(df_price_date_diff['SALE PRICE'][275:350])

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, color='red')

x=[0,90]
y=[stderr,stderr]

plt.plot(x,y, label=stderr, color='orange')

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

```
Length of Predictions :  75
Length of Test data :  75
RMSE :  389374.1531711063
Standard Error :  8893.772953479654
```

Out[285]: Text(0, 0.5, 'Average Price after differencing (in hundreds of thousands)')



Observation:

RMSE is on the higher side of the data but this is after differencing

- RMSE is 389374.15
- Standard error is 8893.7

6a. Testing parameters to improve ARMA model

- I will try p of 19 per ACF
- I will try q of 22 per PACF

In [286]:
```
# Instantiate & fit model with statsmodels
#p = num lags - ACF
p = 4

# q = lagged forecast errors - PACF
q = 4

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

Out[286]:

ARMA Model Results

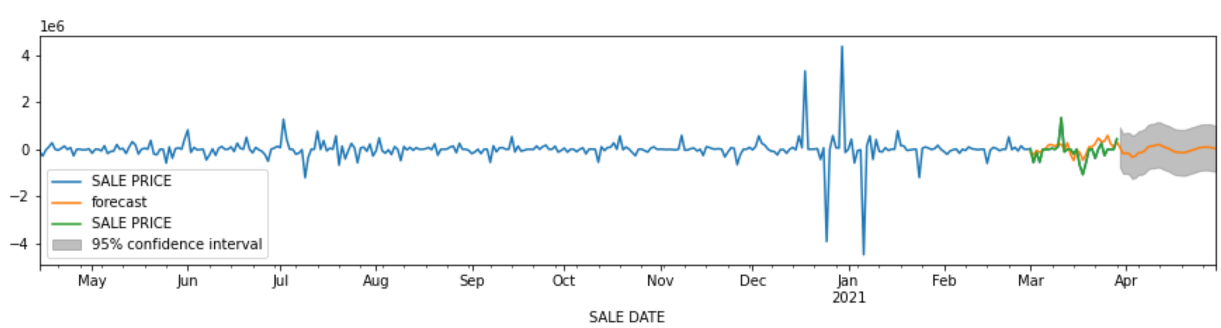
Dep. Variable:	SALE PRICE	**No. Observations:**	350
Model:	ARMA(4, 4)	**Log Likelihood**	-5041.738
Method:	css-mle	**S.D. of innovations**	431192.169
Date:	Sun, 20 Jun 2021	**AIC**	10103.475
Time:	20:05:59	**BIC**	10142.055
Sample:	04-14-2020	**HQIC**	10118.831
	- 03-29-2021		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-7170.9985	1.45e+04	-0.496	0.620	-3.55e+04	2.12e+04
ar.L1.SALE PRICE	1.2226	0.064	19.091	0.000	1.097	1.348
ar.L2.SALE PRICE	-0.7237	0.071	-10.263	0.000	-0.862	-0.585
ar.L3.SALE PRICE	0.8579	0.062	13.911	0.000	0.737	0.979
ar.L4.SALE PRICE	-0.7065	0.045	-15.779	0.000	-0.794	-0.619
ma.L1.SALE PRICE	-1.3127	0.064	-20.499	0.000	-1.438	-1.187
ma.L2.SALE PRICE	1.0104	0.048	21.230	0.000	0.917	1.104
ma.L3.SALE PRICE	-1.3671	0.039	-35.216	0.000	-1.443	-1.291
ma.L4.SALE PRICE	0.8884	0.065	13.643	0.000	0.761	1.016

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-0.3547	-1.0840j	1.1405	-0.3003
AR.2	-0.3547	+1.0840j	1.1405	0.3003
AR.3	0.9618	-0.4037j	1.0431	-0.0632
AR.4	0.9618	+0.4037j	1.0431	0.0632
MA.1	-0.2438	-0.9698j	1.0000	-0.2892
MA.2	-0.2438	+0.9698j	1.0000	0.2892
MA.3	1.0133	-0.3145j	1.0609	-0.0479
MA.4	1.0133	+0.3145j	1.0609	0.0479

In [287]:
```
fig, ax = plt.subplots()
ax = df_price_date_diff.loc['2020-04-14':].plot(ax=ax)
fig = ar1.plot_predict('2021-3-01', '2021-04-30', dynamic=False, ax=ax, plot_ins
plt.show()
```



6a - Error Analysis of new model

In [288]:
```
predictions = list(ar1.predict(276, 350))
test = list(df_price_date_diff['SALE PRICE'][275:350])

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, color='red')

x=[0,90]
y=[stderr,stderr]

plt.plot(x,y, label=stderr, color='orange')

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

```
Length of Predictions :  75
Length of Test data :  75
RMSE :  376348.778022551
Standard Error :  14469.487968645191
```

Out[288]:
```
Text(0, 0.5, 'Average Price after differencing (in hundreds of thousands)')
```

Observation:

- Here RMSE is lower than original model, but standard error is higher....I will stick with 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 [289]:
```
#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']=='STATEN ISLAND']
excel_df.reset_index(drop=True, inplace=True)
```

In [290]:
```
#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 [291]:
```
#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 [292]:
```
# Again, if I drop NaN here, it will change the dates which will affect the plott
# I decide to fillna(0) similar to origional rolling data

excel_price_date = excel_price_date.fillna(0)
```

In [293]:
```
excel_price_date.head()
```

Out[293]:

	SALE PRICE
SALE DATE	
2020-05-01	379892.352941
2020-05-02	0.000000
2020-05-03	0.000000
2020-05-04	368376.181818
2020-05-05	557939.375000

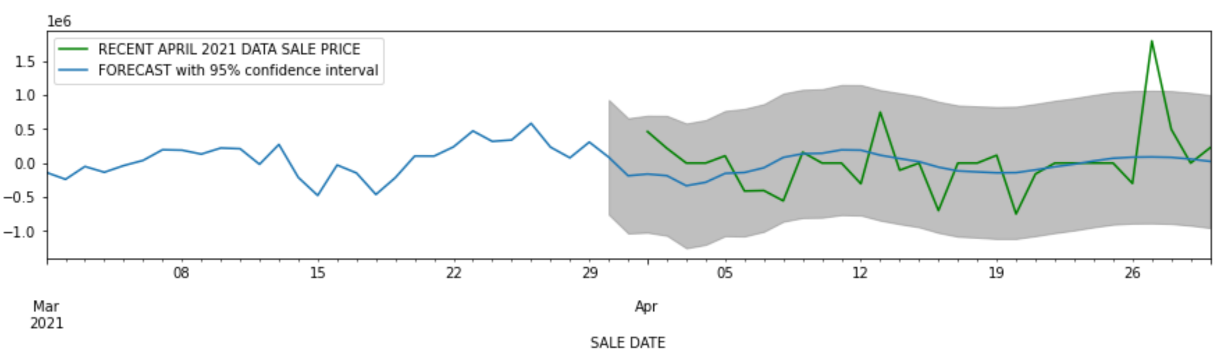
In [294]:
```
excel_price_date_diff= excel_price_date.diff(periods=7)
excel_price_date_diff.dropna(inplace=True)
excel_price_date_diff
```

Out[294]:

	SALE PRICE
SALE DATE	
2020-05-08	3.928715e+04
2020-05-09	5.000000e+04
2020-05-10	0.000000e+00
2020-05-11	-2.387255e+04
2020-05-12	-2.619994e+05
...	...
2021-04-26	-3.000000e+05
2021-04-27	1.800000e+06
2021-04-28	4.980000e+05
2021-04-29	0.000000e+00
2021-04-30	2.250000e+05

358 rows × 1 columns

In [295]:
```
fig, ax = plt.subplots()
ax = excel_price_date_diff.loc['2021-04-01':].plot(ax=ax, color='green')
fig = ar1.plot_predict('2021-3-01', '2021-04-30', dynamic=False, ax=ax, plot_ins

handles, labels = ax.get_legend_handles_labels()
labels = ['RECENT APRIL 2021 DATA SALE PRICE', 'FORECAST with 95% confidence int
ax.legend(handles, labels)
plt.show()
```



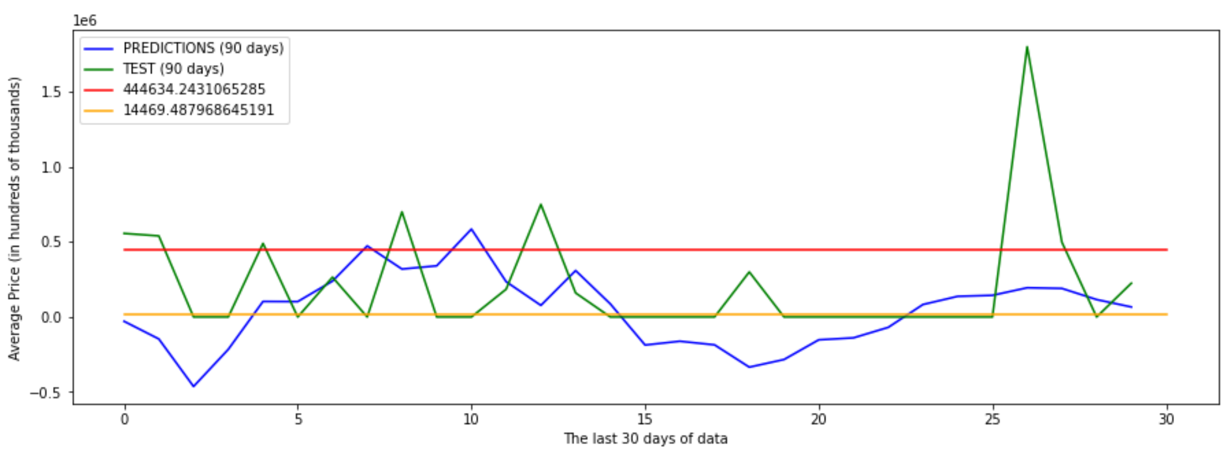
Observation

- The model does not look like it fits well

In [296]:
```
#RMSE, Standard error

# last 30 days of data
predictions = list(ar1.predict(336, 365))
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, color='red')

x=[0,30]
y=[stderr,stderr]
plt.plot(x,y, label=stderr, color='orange')

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 :  444634.2431065285
Standard Error :  14469.487968645191
```

Out[296]: `Text(0, 0.5, 'Average Price (in hundreds of thousands)')`



Observation

1. RMSE is alot higher here when comparing the new month data with predicted values
    - this is probably due to the large sales that occured during the last month

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
    - There is predicable fluctuation in Staten Island
4. We can look at the top 10 building permit heavy locations further
```