Analysis of Rolling Sales Data - Manhattan (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 [2]:

```
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 [3]:
```
#Loading prepped data
df = pd.read_csv('datasets/rollingsales_manhattan.xls_prepped_bare.csv')
df.reset_index(drop=True, inplace=True)
df.sort_values('SALE DATE')
```

Out[3]:

	TAX CLASS AT PRESENT	ZIP CODE	SALE PRICE	SALE DATE
1401	2	10010	567000	2020-04-01
1148	2	10003	750000	2020-04-01
613	2	10013	21551257	2020-04-01
8281	2	10024	390000	2020-04-01
7558	2C	10023	1999000	2020-04-01
...	...	...	...	...
1084	2	10038	2239682	2021-03-31
1423	2	10010	580000	2021-03-31
3284	2	10002	480000	2021-03-31
1480	2	10010	2269679	2021-03-31
6039	2	10075	1950000	2021-03-31

9234 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 [4]:
```
# 1. Convert 'SALE DATE' column to datetime format

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

In [5]:
```
# 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[5]:

	SALE PRICE
SALE DATE	
2021-02-09	2385000
2020-07-16	4350000
2020-11-24	3672530
2020-06-03	249508
2020-06-16	1250000

In [6]:
```
# 3. Group the sales data by daily average

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

In [7]:
```
# 4. We see here number of rows went down from 13171 to 270. 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  270 non-null    float64
dtypes: float64(1)
memory usage: 5.7 KB
```

In [8]:
```
#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[8]: 95

In [9]:
```
# 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[9]:

	SALE PRICE
SALE DATE	
2020-04-01	2.651838e+06
2020-04-02	1.899093e+06
2020-04-03	2.315087e+06
2020-04-04	1.369242e+06
2020-04-05	0.000000e+00
...	...
2021-03-27	0.000000e+00
2021-03-28	0.000000e+00
2021-03-29	1.530709e+06
2021-03-30	1.889714e+06
2021-03-31	6.265608e+06

365 rows × 1 columns

In [10]:
```
# 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
```

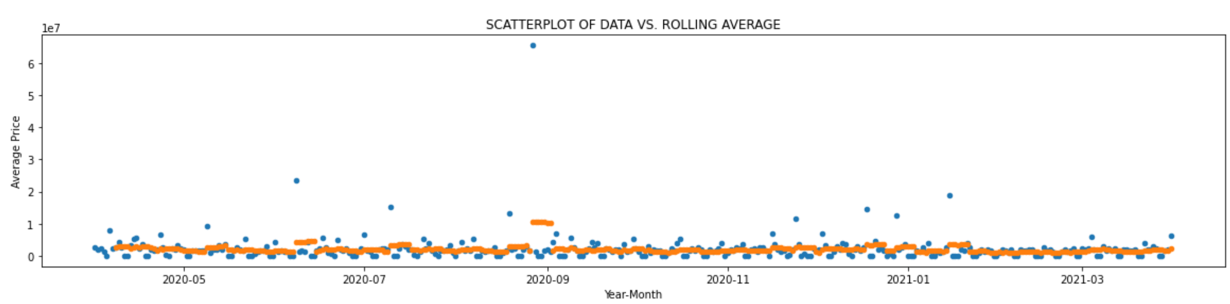
Out[10]:

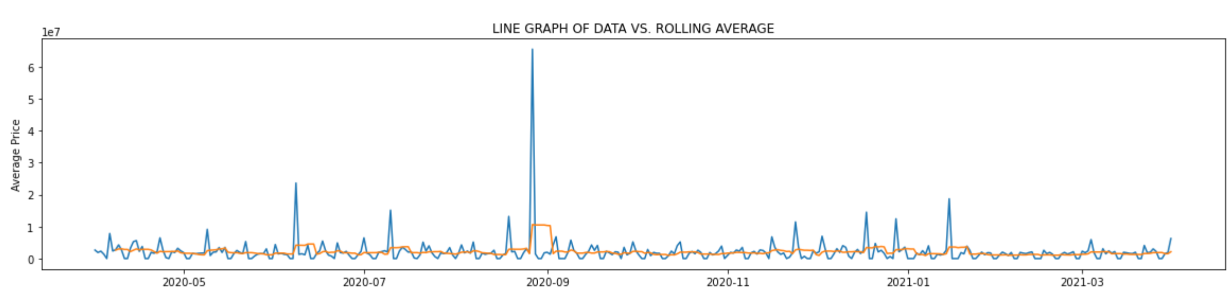
SALE DATE	SALE PRICE	roll_avg
2020-04-01	2.651838e+06	NaN
2020-04-02	1.899093e+06	NaN
2020-04-03	2.315087e+06	NaN
2020-04-04	1.369242e+06	NaN
2020-04-05	0.000000e+00	NaN
...	...	...
2021-03-27	0.000000e+00	1.884169e+06
2021-03-28	0.000000e+00	1.884169e+06
2021-03-29	1.530709e+06	1.516929e+06
2021-03-30	1.889714e+06	1.519312e+06
2021-03-31	6.265608e+06	2.134675e+06

365 rows × 2 columns

In [11]:
```
#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[11]: `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 [12]:

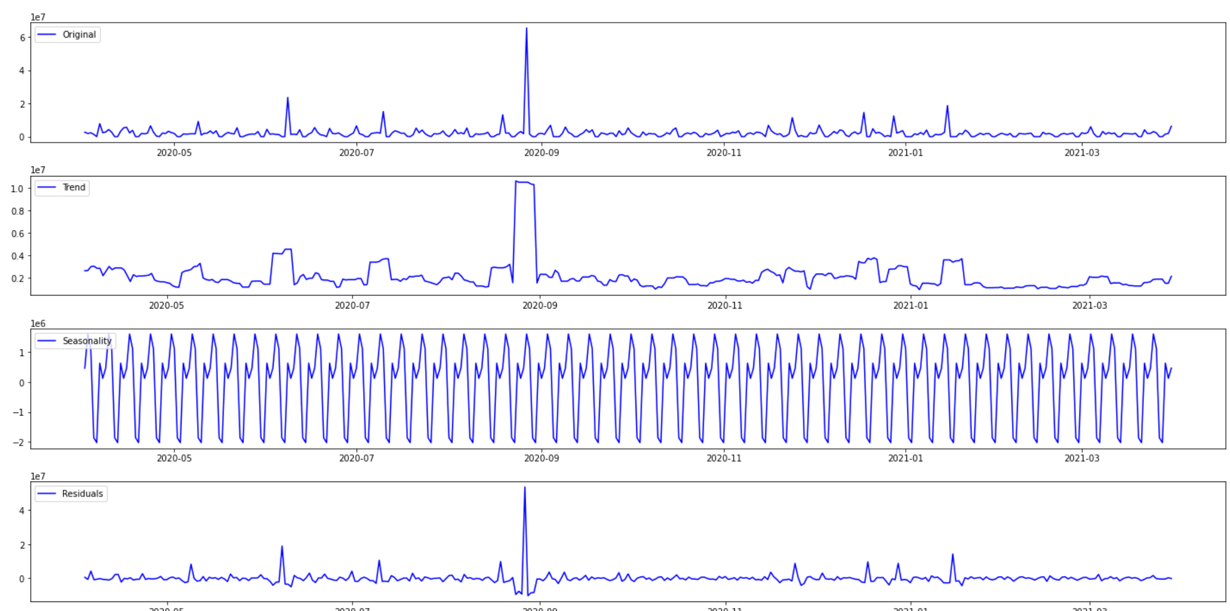
```
# Statsmodels decomposition

# 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 [13]:

```
register_matplotlib_converters()
```

In [14]:

```
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 [15]:
```
# 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)
```

```
(-5.808970891091768, 4.447417926275216e-07, 7, 357, {'1%': -3.44880082033912,
'5%': -2.869670179576637, '10%': -2.5711012838861036}, 11575.736328325731)

Test Statistic                 -5.808971e+00
p-value                         4.447418e-07
#Lags Used                      7.000000e+00
Number of Observations Used     3.570000e+02
Critical Value (1%)            -3.448801e+00
Critical Value (5%)            -2.869670e+00
Critical Value (10%)           -2.571101e+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 **-5.808971**, 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

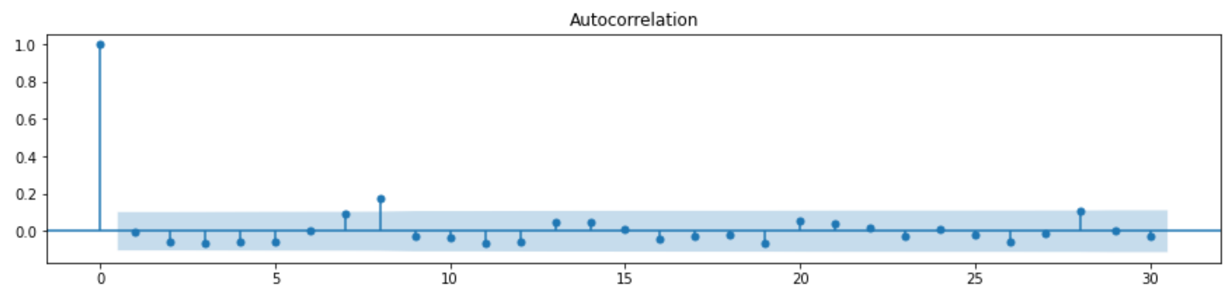
- Our current p-value is **4.447418e-07** or **0.0000004447418** which is REALLY close to zero.
    - *This means: p-value <= 0.05:*
    - *We **REJECT** the null hypothesis! The data does not have a unit root and is stationary*

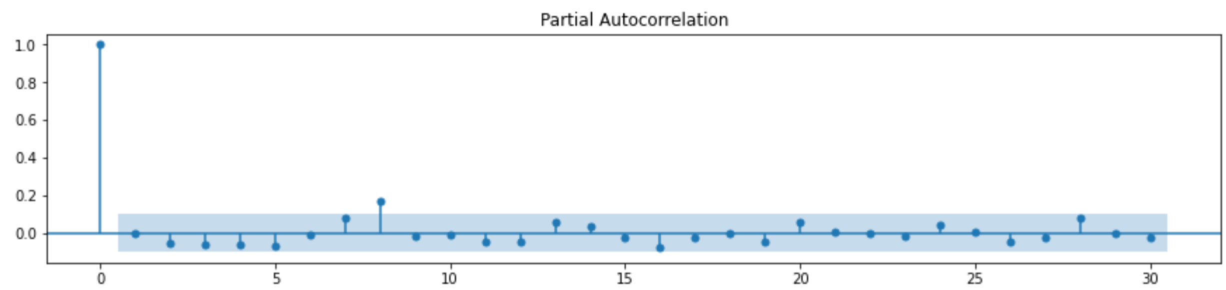
5. ARMA MODELING

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 [16]:
```
# 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 [17]:
```
# 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[17]:

ARMA Model Results

Dep. Variable:	SALE PRICE	**No. Observations:**	365
Model:	ARMA(5, 5)	**Log Likelihood**	-6070.918
Method:	css-mle	**S.D. of innovations**	4005740.737
Date:	Sun, 20 Jun 2021	**AIC**	12165.835
Time:	14:40:49	**BIC**	12212.634
Sample:	04-01-2020	**HQIC**	12184.434
	- 03-31-2021		

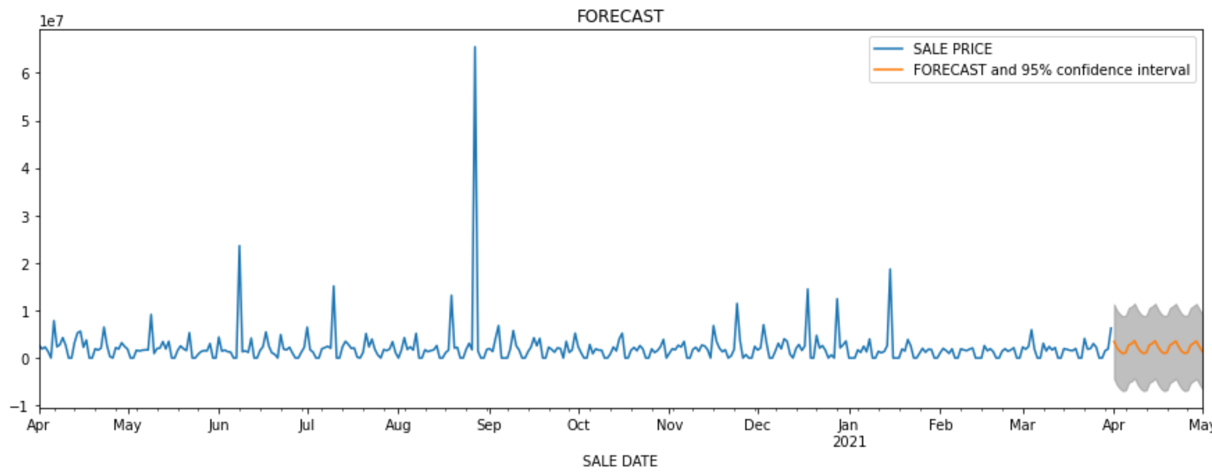
	coef	std err	z	P>|z|	[0.025	0.975]
const	2.161e+06	1.86e+05	11.601	0.000	1.8e+06	2.53e+06
ar.L1.SALE PRICE	-0.0523	nan	nan	nan	nan	nan
ar.L2.SALE PRICE	0.4786	nan	nan	nan	nan	nan
ar.L3.SALE PRICE	-0.6661	0.000	-3228.837	0.000	-0.667	-0.666
ar.L4.SALE PRICE	-0.7110	0.000	-1844.497	0.000	-0.712	-0.710
ar.L5.SALE PRICE	0.4546	0.004	103.932	0.000	0.446	0.463
ma.L1.SALE PRICE	0.0146	0.047	0.312	0.755	-0.077	0.106
ma.L2.SALE PRICE	-0.5352	0.027	-20.101	0.000	-0.587	-0.483
ma.L3.SALE PRICE	0.6791	0.014	50.007	0.000	0.652	0.706
ma.L4.SALE PRICE	0.7053	0.030	23.524	0.000	0.647	0.764
ma.L5.SALE PRICE	-0.5356	0.046	-11.716	0.000	-0.625	-0.446

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	-0.9102	-0.4484j	1.0147	-0.4271
AR.2	-0.9102	+0.4484j	1.0147	0.4271
AR.3	0.6239	-0.7815j	1.0000	-0.1428
AR.4	0.6239	+0.7815j	1.0000	0.1428
AR.5	2.1366	-0.0000j	2.1366	-0.0000
MA.1	-0.8985	-0.4390j	1.0000	-0.4277
MA.2	-0.8985	+0.4390j	1.0000	0.4277
MA.3	0.6234	-0.7819j	1.0000	-0.1429
MA.4	0.6234	+0.7819j	1.0000	0.1429
MA.5	1.8669	-0.0000j	1.8669	-0.0000

In [19]:
```
#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>
```

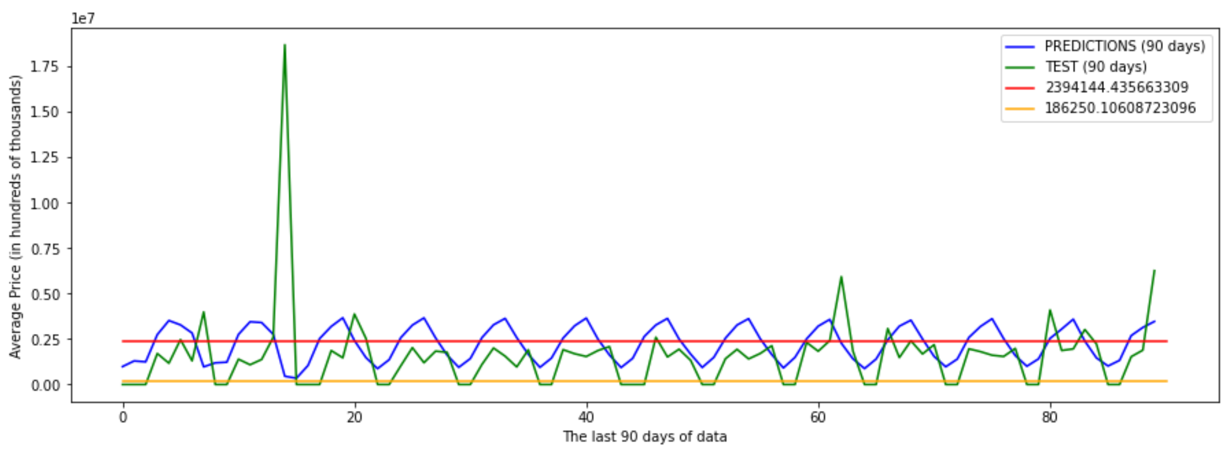


6. Error analysis of ARMA model

In [28]:
```
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, 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 (in hundreds of thousands)")
```

```
Length of Predictions :  90
Length of Test data :  90
RMSE :  2394144.435663309
Standard Error :  186250.10608723096
```

Out[28]: `Text(0, 0.5, 'Average Price (in hundreds of thousands)')`



Observation:

RMSE is on the higher side of the data point values. This might indicate the higher price volatility in manhattan versus other boroughs.

- RMSE is 239414.4
- Standard error is 186250.1

6a. Testing parameters to improve ARMA model

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

In [31]:
```
# Instantiate & fit model with statsmodels
#p = num lags - ACF
p = 7

# q = lagged forecast errors - PACF
q = 7

#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[31]:

ARMA Model Results

Dep. Variable:	SALE PRICE	**No. Observations:**	365
Model:	ARMA(5, 5)	**Log Likelihood**	-6070.918
Method:	css-mle	**S.D. of innovations**	4005740.737
Date:	Sun, 20 Jun 2021	**AIC**	12165.835
Time:	14:47:26	**BIC**	12212.634
Sample:	04-01-2020	**HQIC**	12184.434
	- 03-31-2021		

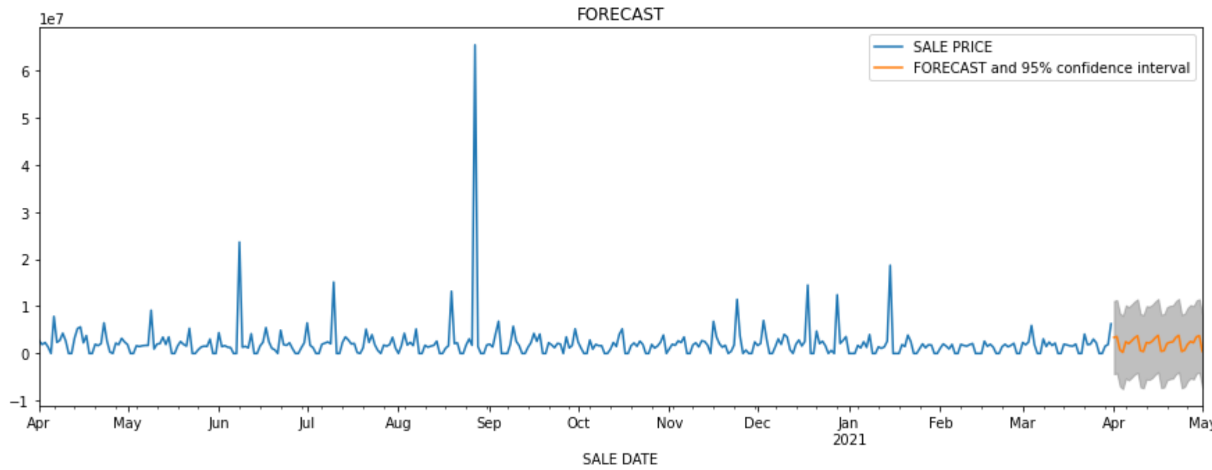
	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	2.161e+06	1.86e+05	11.601	0.000	1.8e+06	2.53e+06
ar.L1.SALE PRICE	-0.0523	nan	nan	nan	nan	nan
ar.L2.SALE PRICE	0.4786	nan	nan	nan	nan	nan
ar.L3.SALE PRICE	-0.6661	0.000	-3228.837	0.000	-0.667	-0.666
ar.L4.SALE PRICE	-0.7110	0.000	-1844.497	0.000	-0.712	-0.710
ar.L5.SALE PRICE	0.4546	0.004	103.932	0.000	0.446	0.463
ma.L1.SALE PRICE	0.0146	0.047	0.312	0.755	-0.077	0.106
ma.L2.SALE PRICE	-0.5352	0.027	-20.101	0.000	-0.587	-0.483
ma.L3.SALE PRICE	0.6791	0.014	50.007	0.000	0.652	0.706
ma.L4.SALE PRICE	0.7053	0.030	23.524	0.000	0.647	0.764
ma.L5.SALE PRICE	-0.5356	0.046	-11.716	0.000	-0.625	-0.446

Roots

	Real	**Imaginary**	**Modulus**	**Frequency**
AR.1	-0.9102	-0.4484j	1.0147	-0.4271
AR.2	-0.9102	+0.4484j	1.0147	0.4271
AR.3	0.6239	-0.7815j	1.0000	-0.1428
AR.4	0.6239	+0.7815j	1.0000	0.1428
AR.5	2.1366	-0.0000j	2.1366	-0.0000
MA.1	-0.8985	-0.4390j	1.0000	-0.4277
MA.2	-0.8985	+0.4390j	1.0000	0.4277
MA.3	0.6234	-0.7819j	1.0000	-0.1429
MA.4	0.6234	+0.7819j	1.0000	0.1429
MA.5	1.8669	-0.0000j	1.8669	-0.0000

In [33]:
```
#plot of ARMA model
plt.figure(figsize=(20,10))
fig, ax = plt.subplots()
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>
```

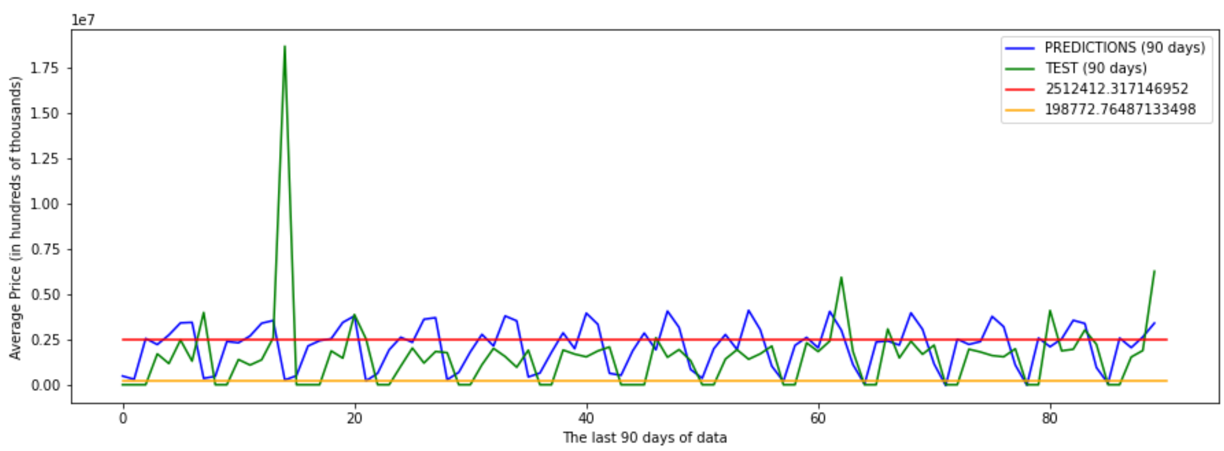


6a - Error Analysis of new model

In [35]:
```
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, 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 (in hundreds of thousands)")
```

```
Length of Predictions :  90
Length of Test data :  90
RMSE :  2512412.317146952
Standard Error :  198772.76487133498
```

Out[35]: `Text(0, 0.5, 'Average Price (in hundreds of thousands)')`



Observation:

- Here RMSE is higher than original model. We will stick with original 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 [36]:
```
#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']=='MANHATTAN']
excel_df.reset_index(drop=True, inplace=True)
```

In [37]:
```
#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 [38]:
```
#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()
```

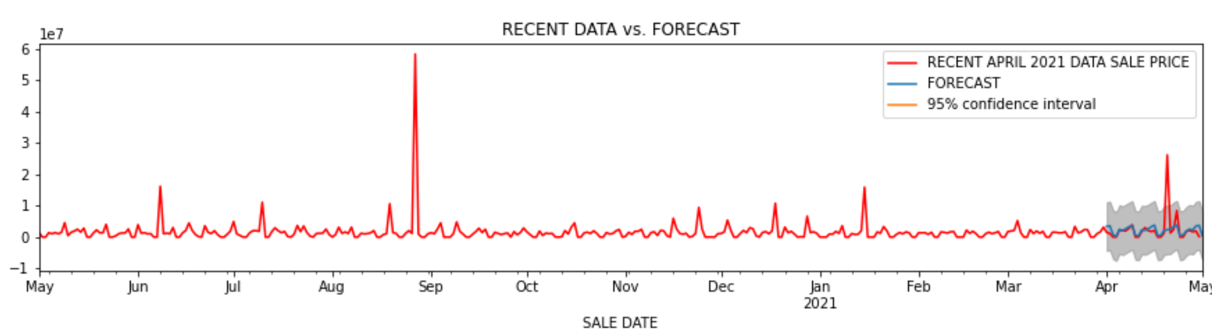
In [39]:
```
# Again, if I drop NaN here, it will change the dates which will affect the plott
# I decide to fillna(0) similar to origional rolling data

excel_price_date = excel_price_date.fillna(0)
```

In [40]:
```
excel_price_date.head()
```

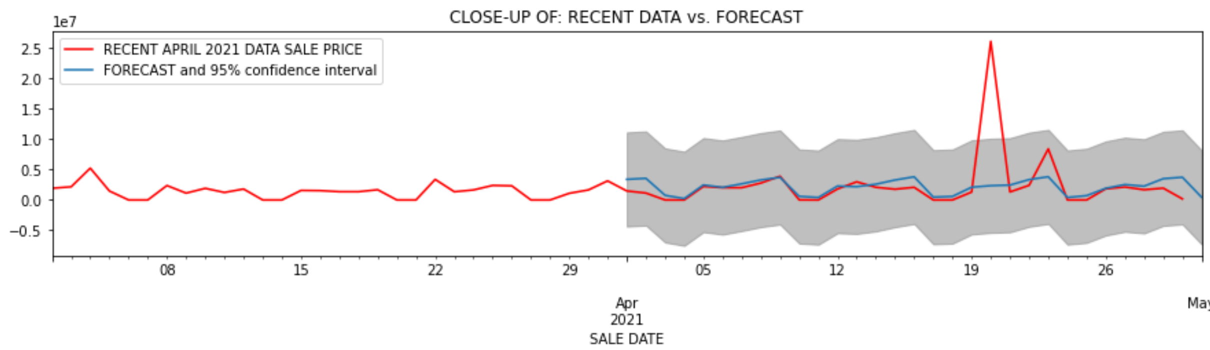
Out[40]:

	SALE PRICE
SALE DATE	
2020-05-01	1.395860e+06
2020-05-02	0.000000e+00
2020-05-03	0.000000e+00
2020-05-04	1.401291e+06
2020-05-05	1.049441e+06

In [41]:
```
# 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 [42]:
```
# 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()
```



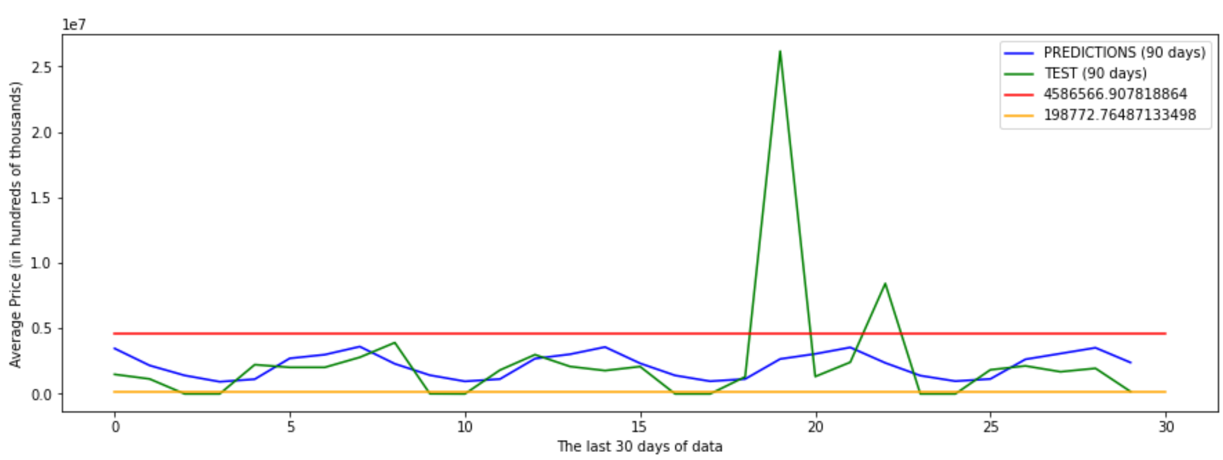
Observation

- We see that the model looks like it fits well versus the test data of 4/1/2021 until 4/31/2021
- There is a spike in April 2021, probably a building got sold for millions
    - This will affect our RMSE

In [47]:
```
#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, 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 :  4586566.907818864
Standard Error :  198772.76487133498
```

Out[47]: Text(0, 0.5, 'Average Price (in hundreds of thousands)')



Observation

1. RMSE is higher because of the spike in sales

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
    - Espescially for Manhattan, there is a predictable fluctuation
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
```