analysis_nyc_all

April 23, 2023

1 Analysis Template

1.1 Preprocess

[ ]:

```
# resolve dependency
# !pip install pmdarima
```

[ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from pandas.plotting import autocorrelation_plot
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
import statsmodels.api as sm
from pmdarima.arima import ADFTest , auto_arima
%matplotlib inline
```

[ ]:

```
data_path = "../data/nypd_all.csv"
crime = "all"
target = "count"
date = "date"
city = "nyc"
fig_size = (20,5)
```

[ ]:

```
df_by_day = pd.read_csv(data_path)
df_by_day[date] = pd.to_datetime(df_by_day[date])
df_by_day.set_index(date, inplace=True)
```

1.2 Profiling

1.2.1 By day

[ ]:

```
df_by_day.head()
```

[ ]:

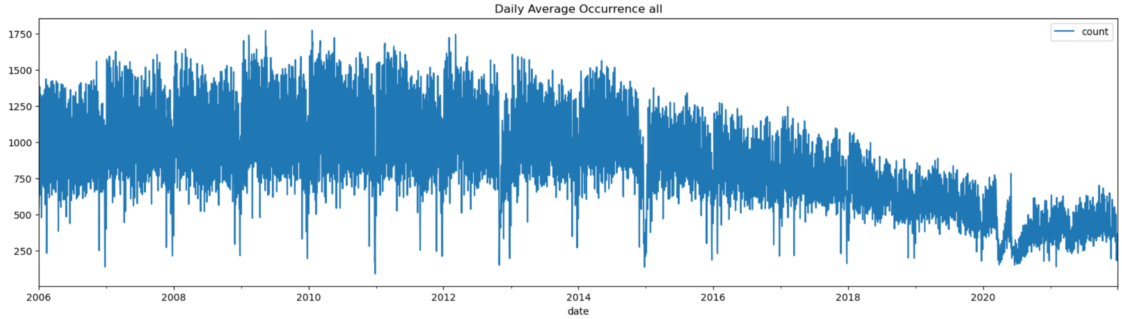
```
            count
date
2006-01-01    551
2006-01-02    618
```

```
2006-01-03     899
2006-01-04    1229
2006-01-05    1383
```

[ ]:
```
df_by_day.describe()
```

[ ]:
```
             count
count  5844.000000
mean    906.862936
std     352.693246
min      90.000000
25%     642.750000
50%     869.000000
75%    1210.000000
max    1772.000000
```

[ ]:
```
df_by_day.plot(figsize=fig_size, title="Daily Average Occurrence " + crime)
plt.show()
```



[ ]:
```
df_by_day[target].sort_values(ascending=False).head()
```

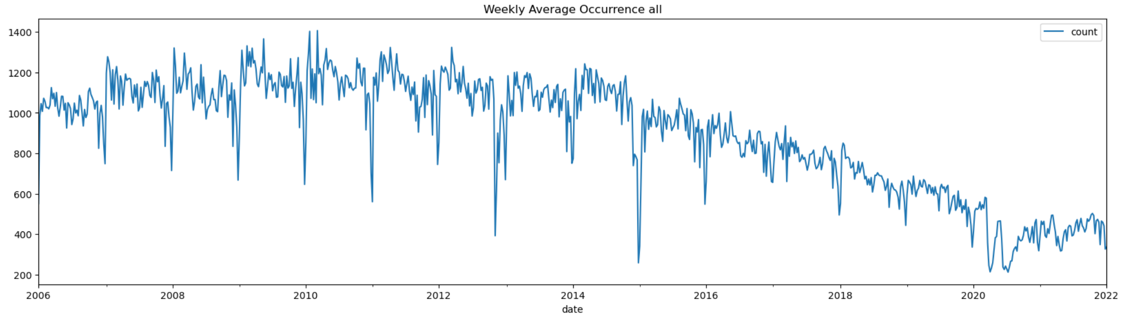
[ ]:
```
date
2010-01-20    1772
2009-05-13    1770
2012-03-07    1744
2009-02-11    1738
2012-02-01    1722
Name: count, dtype: int64
```

1.2.2 By week

[ ]:
```
df_by_week = pd.DataFrame(df_by_day[target].resample('W').mean())
```

[ ]:

```
df_by_week.plot(
    figsize=fig_size,
    title="Weekly Average Occurrence " + crime)
plt.show()
```



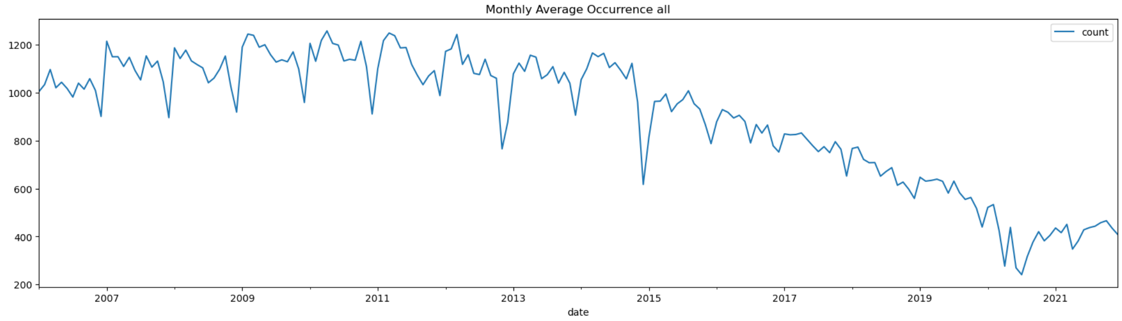
1.2.3 By month

[ ]:

```
df_by_month = pd.DataFrame(df_by_day[target].resample('M').mean())
```

[ ]:

```
df_by_month.plot(
    figsize=fig_size,
    title="Monthly Average Occurrence " + crime)
plt.show()
```



1.3 Analysis

[ ]:

```
#Ho: It is non stationary
#H1: It is stationary

def adfuller_test(count):
    result=adfuller(count)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of␣
↪Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null␣
↪hypothesis. Data has no unit root and is stationary")
    else:
        print("weak evidence against null hypothesis, time series has a unit␣
↪root, indicating it is non-stationary ")
```

1.3.1 Checking stationary

[ ]:

```
adfuller_test(df_by_month[target])
```

```
ADF Test Statistic : 1.0639355763319358
p-value : 0.9948967641659721
#Lags Used : 12
Number of Observations Used : 179
weak evidence against null hypothesis, time series has a unit root, indicating
it is non-stationary
```

1.3.2 Checking seasonality

[ ]:

```
df_by_month['seasonal_first_difference'] = df_by_month[target] -␣
↪df_by_month[target].shift(12)
```

[ ]:

```
adfuller_test(df_by_month['seasonal_first_difference'].dropna())
```

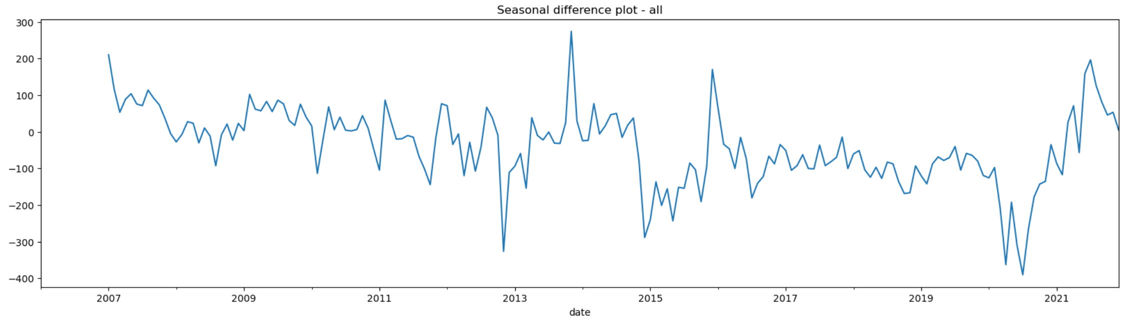
```
ADF Test Statistic : -2.348504655413488
p-value : 0.15680087793764524
#Lags Used : 14
Number of Observations Used : 165
weak evidence against null hypothesis, time series has a unit root, indicating
it is non-stationary
```

[ ]:

```
df_by_month['seasonal_first_difference'].plot(figsize=fig_size, title='Seasonal␣
↪difference plot - ' + crime)
```

[ ]:

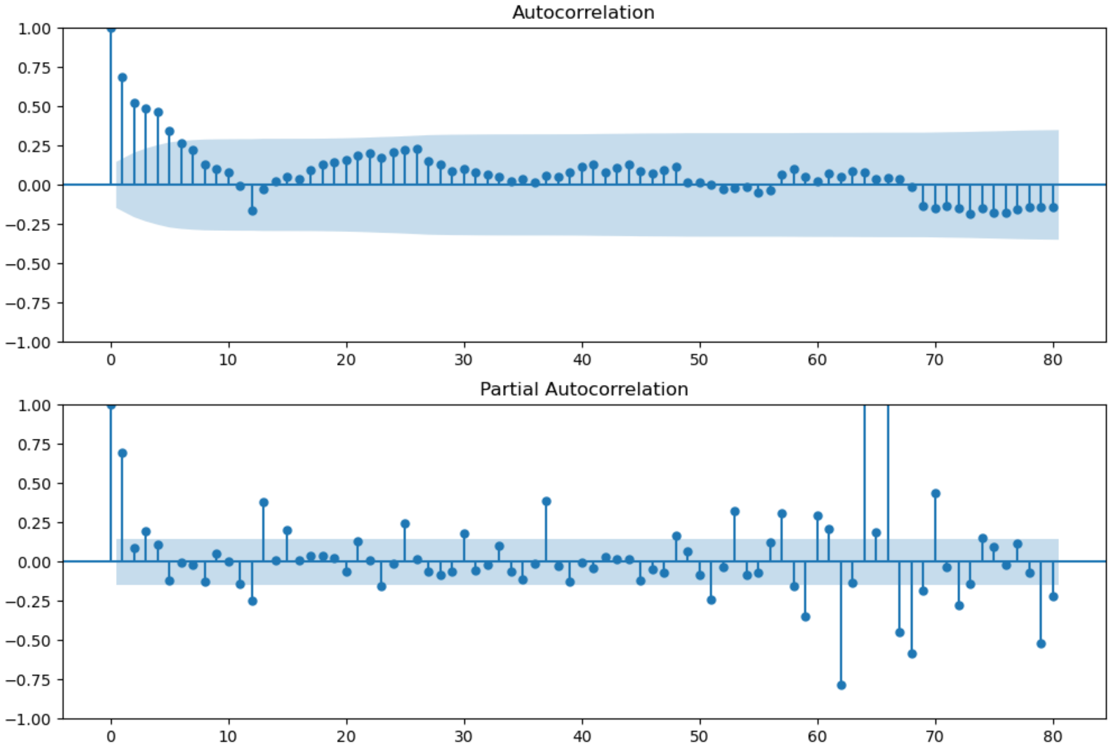
```
<Axes: title={'center': 'Seasonal difference plot - all'}, xlabel='date'>
```



1.3.3 Auto Regressive Model

```
[ ]: fig = plt.figure(figsize=(12,8))
     ax1 = fig.add_subplot(211)
     fig = sm.graphics.tsa.plot_acf(df_by_month['seasonal_first_difference'].iloc[13:
      ↪],lags=80,ax=ax1)
     ax2 = fig.add_subplot(212)
     fig = sm.graphics.tsa.plot_pacf(df_by_month['seasonal_first_difference'].
      ↪iloc[13:],lags=80,ax=ax2)
```

```
/Users/xuyanchong/opt/anaconda3/lib/python3.9/site-
packages/statsmodels/graphics/tsaplots.py:348: FutureWarning: The default method
'yw' can produce PACF values outside of the [-1,1] interval. After 0.13, the
default will change tounadjusted Yule-Walker ('ywm'). You can use this method
now by setting method='ywm'.
  warnings.warn(
```



1.3.4 Implementing Seasonal Arima Model

[ ]:

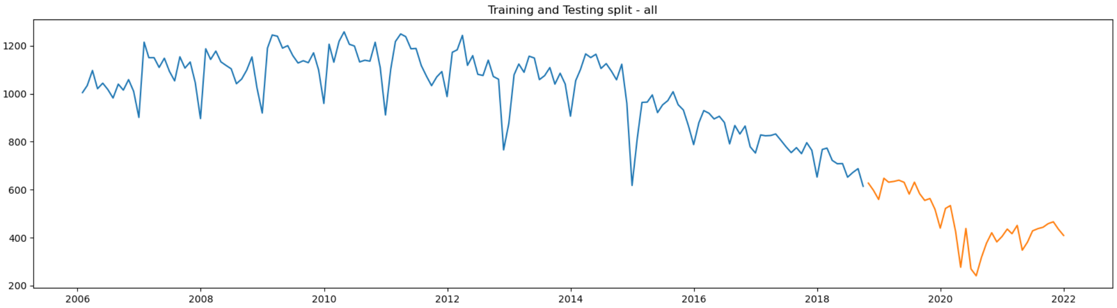
```
adf_test=ADFTest(alpha=0.05)
adf_test.should_diff(df_by_month[target])
```

[ ]:

```
(0.34024313664207795, True)
```

[ ]:

```
start=int(df_by_month.shape[0]*0.8)
train=df_by_month[:start]
test=df_by_month[start:]
plt.figure(figsize=fig_size)
plt.plot(train[target])
plt.plot(test[target])
plt.title('Training and Testing split - '+ crime)
plt.show()
```



```
[ ]: model=auto_arima(train[target],start_p=0,d=1,start_q=0,
               max_p=10,max_d=10,max_q=10, start_P=0,
               D=1, start_Q=0, max_P=10,max_D=10,
               max_Q=10, m=12, seasonal=True,
               error_action='warn',trace=True,
               supress_warnings=True,stepwise=True,
               random_state=20,n_fits=50)
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=1622.760, Time=0.08 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=1586.372, Time=0.19 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=1547.962, Time=0.95 sec
 ARIMA(0,1,1)(0,1,0)[12]             : AIC=1604.855, Time=0.18 sec
 ARIMA(0,1,1)(1,1,1)[12]             : AIC=1549.961, Time=2.77 sec
 ARIMA(0,1,1)(0,1,2)[12]             : AIC=1549.961, Time=22.27 sec
 ARIMA(0,1,1)(1,1,0)[12]             : AIC=1574.771, Time=0.77 sec
 ARIMA(0,1,1)(1,1,2)[12]             : AIC=1551.942, Time=35.76 sec
 ARIMA(0,1,0)(0,1,1)[12]             : AIC=1558.275, Time=1.23 sec
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=1541.301, Time=3.07 sec
 ARIMA(1,1,1)(0,1,0)[12]             : AIC=1595.370, Time=0.67 sec
 ARIMA(1,1,1)(1,1,1)[12]             : AIC=1543.173, Time=3.06 sec
 ARIMA(1,1,1)(0,1,2)[12]             : AIC=1543.136, Time=27.32 sec
 ARIMA(1,1,1)(1,1,0)[12]             : AIC=1564.086, Time=2.82 sec
 ARIMA(1,1,1)(1,1,2)[12]             : AIC=1543.053, Time=35.19 sec
 ARIMA(1,1,0)(0,1,1)[12]             : AIC=1553.718, Time=2.18 sec
 ARIMA(2,1,1)(0,1,1)[12]             : AIC=1542.745, Time=3.54 sec
 ARIMA(1,1,2)(0,1,1)[12]             : AIC=1542.778, Time=3.29 sec
 ARIMA(0,1,2)(0,1,1)[12]             : AIC=1542.011, Time=2.56 sec
 ARIMA(2,1,0)(0,1,1)[12]             : AIC=1549.121, Time=2.79 sec
 ARIMA(2,1,2)(0,1,1)[12]             : AIC=inf, Time=4.58 sec
 ARIMA(1,1,1)(0,1,1)[12] intercept   : AIC=inf, Time=2.63 sec

Best model:  ARIMA(1,1,1)(0,1,1)[12]
Total fit time: 157.926 seconds
```

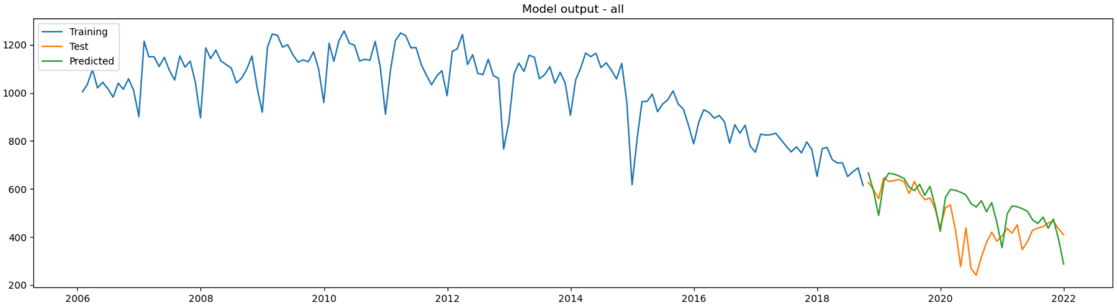
[ ]:
```
model.summary()
```

[ ]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
================================================================================
==========
Dep. Variable:                                  y   No. Observations:
153
Model:             SARIMAX(1, 1, 1)x(0, 1, 1, 12)   Log Likelihood
-766.650
Date:                            Sun, 23 Apr 2023   AIC
1541.301
Time:                                    01:34:04   BIC
1553.067
Sample:                                01-31-2006   HQIC
1546.082
                                     - 09-30-2018
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4540      0.124      3.669      0.000       0.211       0.697
ma.L1         -0.8218      0.098     -8.371      0.000      -1.014      -0.629
ma.S.L12      -0.7483      0.074    -10.074      0.000      -0.894      -0.603
sigma2      3101.2810    235.584     13.164      0.000    2639.545    3563.017
================================================================================
===
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):
188.69
Prob(Q):                              0.85   Prob(JB):
0.00
Heteroskedasticity (H):               2.80   Skew:
-1.08
Prob(H) (two-sided):                  0.00   Kurtosis:
8.26
================================================================================
===

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

[ ]:
```
prediction = pd.DataFrame(model.predict(n_periods = train.shape[0]),index=test.
  ↪index)
prediction.columns = ['predicted_crime']
```

```python
plt.figure(figsize=fig_size)
plt.plot(train[target],label="Training")
plt.plot(test[target],label="Test")
plt.plot(prediction,label="Predicted")
plt.legend(loc = 'upper left')
plt.savefig('../output/%s_%s_pred.jpg' % (city,crime))
plt.title('Model output - '+crime)
plt.show()
```



```
[ ]: np.sqrt(np.square(np.subtract(test[target].values,prediction['predicted_crime'].
     ↪values)).mean())
```

```
[ ]: 113.3564046298324
```