analysis_merged_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 = "merged"
fig_size = (20,5)
```

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

[ ]:
```
data_path = "../data/crime_occurrence_per_day.csv"
target = "Count"
date = "Date"
```

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

[ ]:
```
df_by_day=df_by_day_nyc.join(df_by_day_chi,how='inner')
```

[ ]:
```
df_by_day[target]=df_by_day[target]+df_by_day['count']
df_by_day.drop('count',axis=1,inplace=True)
```

[ ]:
```
df_by_day
```

[ ]:
```
            Count
2006-01-01   2469
2006-01-02   1648
2006-01-03   2057
2006-01-04   2389
2006-01-05   2503
…             …
2021-12-27    809
2021-12-28    816
2021-12-29    866
2021-12-30    877
2021-12-31    935

[5844 rows x 1 columns]
```

1.2 Profiling

1.2.1 By day

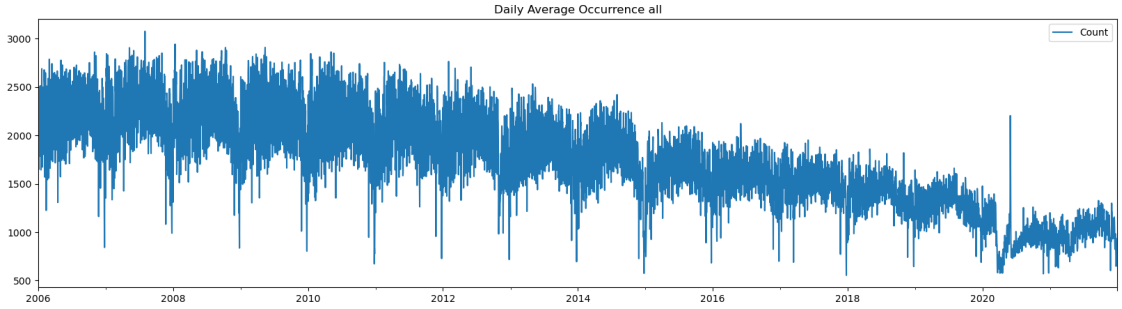
[ ]:
```
df_by_day.head()
```

[ ]:
```
            Count
2006-01-01   2469
2006-01-02   1648
2006-01-03   2057
2006-01-04   2389
2006-01-05   2503
```

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

[ ]:
```
             Count
count  5844.000000
mean   1779.790726
std     524.189021
min     555.000000
25%    1393.000000
50%    1766.000000
75%    2204.000000
max    3077.000000
```

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



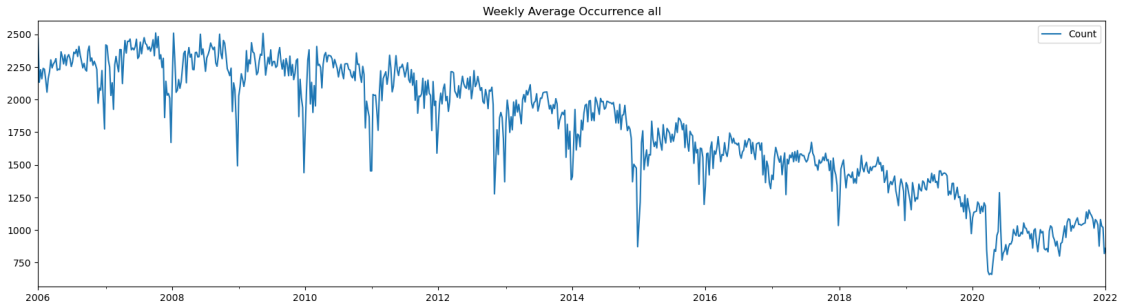
[ ]:
```
df_by_day[target].sort_values(ascending=False).head()
```

[ ]:
```
2007-08-01    3077
2008-01-11    2944
2009-05-13    2910
2008-10-10    2909
2007-05-09    2907
Name: Count, dtype: int64
```

1.2.2 By week

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

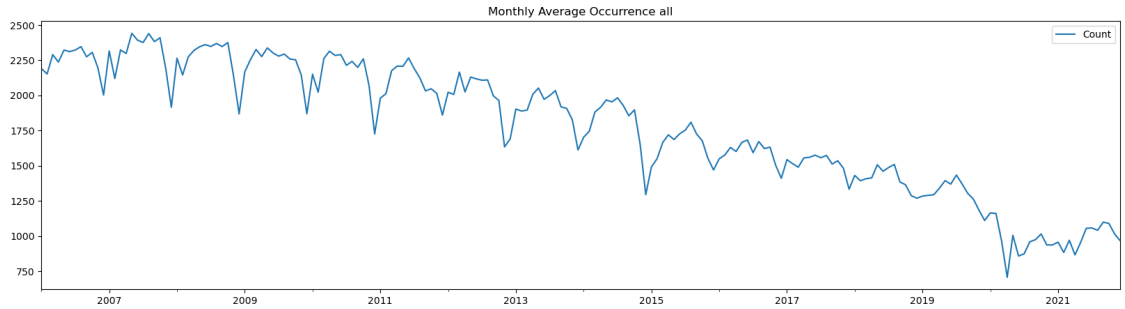
[ ]:
```
df_by_week.plot(
    figsize=fig_size,
    title="Weekly Average Occurrence " + crime)
plt.show()
```



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 : 0.27241211934971554
p-value : 0.9760427583407253
#Lags Used : 14
Number of Observations Used : 177
weak evidence against null hypothesis, time series has a unit root, indicating
it is non-stationary
```

1.3.2 Checking seasonality

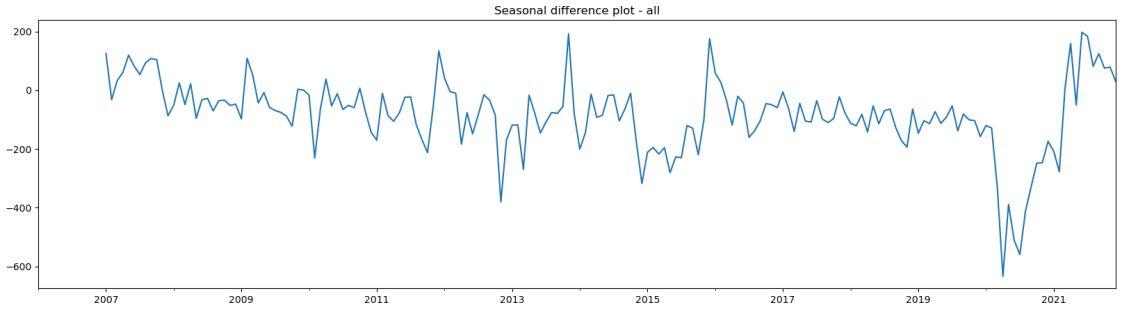
[ ]:
```
df_by_month['seasonal_first_difference'] = df_by_month[target] -␣
↪df_by_month[target].shift(12)
```

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

```
ADF Test Statistic : -3.2567890935200645
p-value : 0.016924009462096865
#Lags Used : 12
Number of Observations Used : 167
strong evidence against the null hypothesis(Ho), reject the null hypothesis.
Data has no unit root and is stationary
```

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

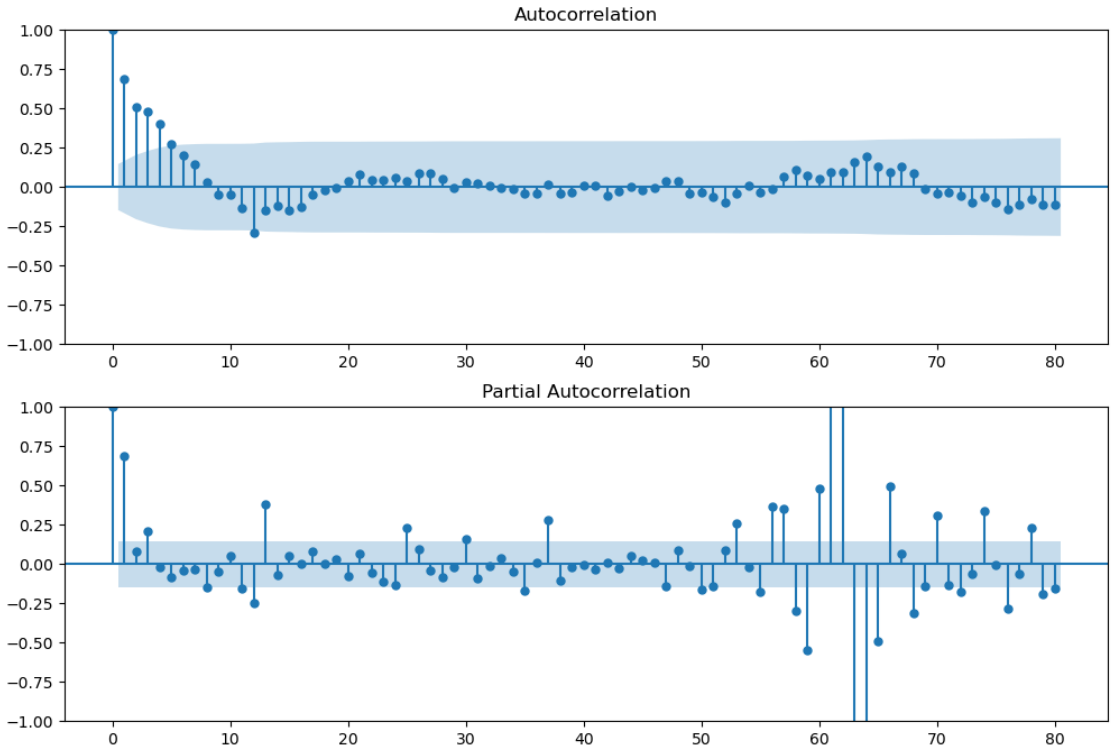
[ ]: <Axes: title={'center': 'Seasonal difference plot - all'}>



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(
```

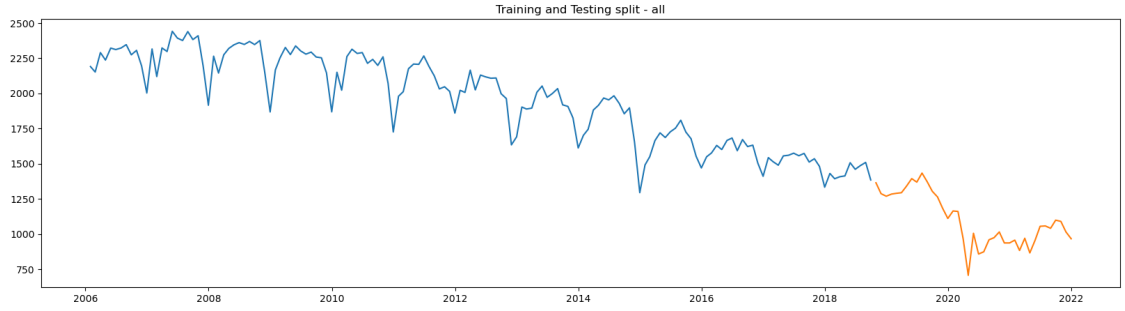


1.3.4 Implementing Seasonal Arima Model

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

[ ]:
```
(0.01, False)
```

[ ]:
```
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=1666.692, Time=0.13 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=1628.914, Time=0.61 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=1593.796, Time=3.48 sec
 ARIMA(0,1,1)(0,1,0)[12]             : AIC=1640.942, Time=0.32 sec
 ARIMA(0,1,1)(1,1,1)[12]             : AIC=1595.530, Time=3.47 sec
 ARIMA(0,1,1)(0,1,2)[12]             : AIC=1595.482, Time=23.93 sec
 ARIMA(0,1,1)(1,1,0)[12]             : AIC=1616.456, Time=0.96 sec
 ARIMA(0,1,1)(1,1,2)[12]             : AIC=1597.439, Time=39.78 sec
 ARIMA(0,1,0)(0,1,1)[12]             : AIC=1608.538, Time=1.19 sec
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=1583.009, Time=2.45 sec
 ARIMA(1,1,1)(0,1,0)[12]             : AIC=1629.880, Time=0.67 sec
 ARIMA(1,1,1)(1,1,1)[12]             : AIC=1584.982, Time=3.12 sec
 ARIMA(1,1,1)(0,1,2)[12]             : AIC=1584.976, Time=36.54 sec
 ARIMA(1,1,1)(1,1,0)[12]             : AIC=1601.333, Time=2.49 sec
 ARIMA(1,1,1)(1,1,2)[12]             : AIC=1586.025, Time=51.95 sec
 ARIMA(1,1,0)(0,1,1)[12]             : AIC=1602.511, Time=1.59 sec
 ARIMA(2,1,1)(0,1,1)[12]             : AIC=1584.226, Time=2.85 sec
 ARIMA(1,1,2)(0,1,1)[12]             : AIC=1584.340, Time=2.71 sec
 ARIMA(0,1,2)(0,1,1)[12]             : AIC=1583.643, Time=2.09 sec
 ARIMA(2,1,0)(0,1,1)[12]             : AIC=1597.415, Time=1.40 sec
 ARIMA(2,1,2)(0,1,1)[12]             : AIC=inf, Time=2.75 sec
 ARIMA(1,1,1)(0,1,1)[12] intercept   : AIC=1580.828, Time=1.49 sec
 ARIMA(1,1,1)(0,1,0)[12] intercept   : AIC=1629.942, Time=0.18 sec
 ARIMA(1,1,1)(1,1,1)[12] intercept   : AIC=1582.823, Time=1.84 sec
 ARIMA(1,1,1)(0,1,2)[12] intercept   : AIC=1582.823, Time=12.54 sec
 ARIMA(1,1,1)(1,1,0)[12] intercept   : AIC=1601.306, Time=0.89 sec
 ARIMA(1,1,1)(1,1,2)[12] intercept   : AIC=1584.773, Time=10.29 sec
```

```
 ARIMA(0,1,1)(0,1,1)[12] intercept   : AIC=1594.727, Time=0.38 sec
 ARIMA(1,1,0)(0,1,1)[12] intercept   : AIC=1604.136, Time=0.46 sec
 ARIMA(2,1,1)(0,1,1)[12] intercept   : AIC=1582.294, Time=1.16 sec
 ARIMA(1,1,2)(0,1,1)[12] intercept   : AIC=1582.362, Time=1.20 sec
 ARIMA(0,1,0)(0,1,1)[12] intercept   : AIC=1610.307, Time=0.28 sec
 ARIMA(0,1,2)(0,1,1)[12] intercept   : AIC=1582.349, Time=0.59 sec
 ARIMA(2,1,0)(0,1,1)[12] intercept   : AIC=1598.953, Time=0.59 sec
 ARIMA(2,1,2)(0,1,1)[12] intercept   : AIC=inf, Time=1.09 sec

Best model:  ARIMA(1,1,1)(0,1,1)[12] intercept
Total fit time: 217.498 seconds
```

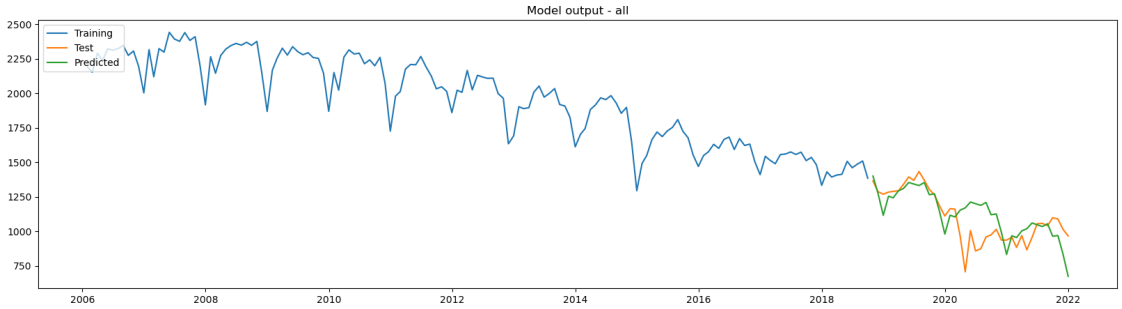
[ ]:
```
model.summary()
```

[ ]:
```
<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
-785.414
Date:                            Sun, 23 Apr 2023   AIC
1580.828
Time:                                    01:35:11   BIC
1595.536
Sample:                                01-31-2006   HQIC
1586.805
                                     - 09-30-2018
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     -0.4874      0.213     -2.287      0.022      -0.905      -0.070
ar.L1          0.4357      0.098      4.431      0.000       0.243       0.628
ma.L1         -0.9128      0.064    -14.363      0.000      -1.037      -0.788
ma.S.L12      -0.7262      0.075     -9.679      0.000      -0.873      -0.579
sigma2      4047.9679    407.216      9.941      0.000    3249.838    4846.097
==========================================================================================
===
Ljung-Box (L1) (Q):                   0.07   Jarque-Bera (JB):
14.45
Prob(Q):                              0.79   Prob(JB):
0.00
Heteroskedasticity (H):               1.47   Skew:
-0.13
```

```
Prob(H) (two-sided):                  0.19   Kurtosis:
4.55
================================================================================
===

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']
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())
```

[ ]:
```
150.03960532693114
```