analysis_nyc_type2

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_larceny.csv"
crime = "type2"
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()
```

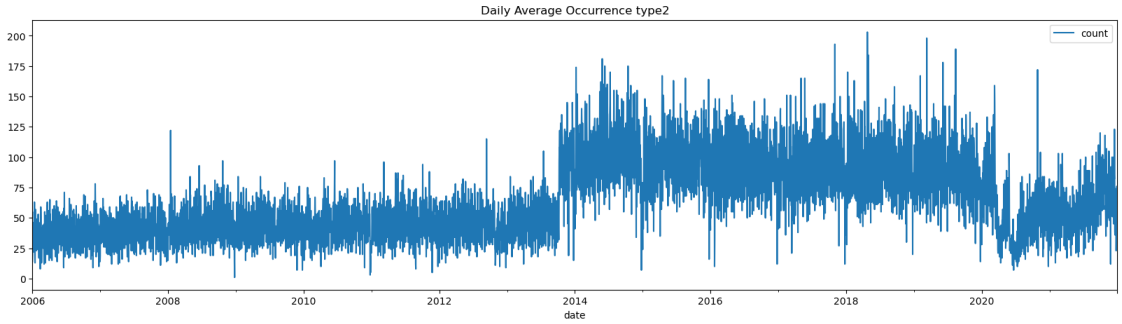
[ ]:
```
            count
date
2006-01-01     14
2006-01-02     29
```

```
2006-01-03     36
2006-01-04     40
2006-01-05     46
```

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

[ ]:
```
             count
count  5844.000000
mean     65.869952
std      32.442862
min       1.000000
25%      41.000000
50%      58.000000
75%      89.000000
max     203.000000
```

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



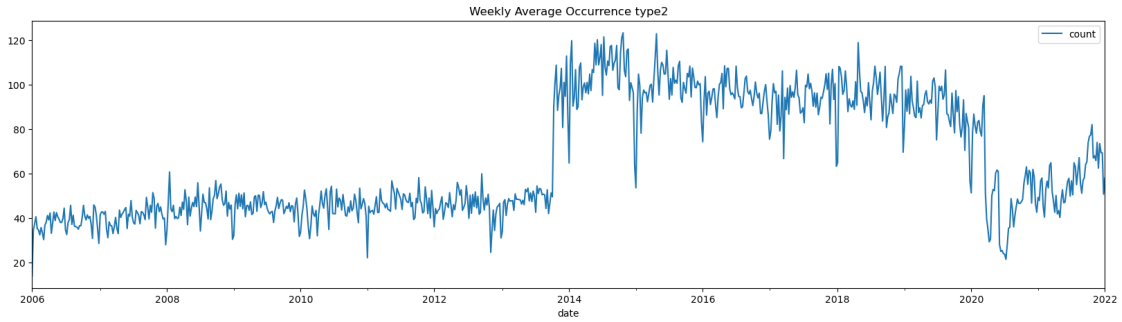
[ ]:
```
df_by_day[target].sort_values(ascending=False).head()
```

[ ]:
```
date
2018-04-25    203
2019-03-11    198
2017-11-01    193
2019-08-14    189
2018-05-01    184
Name: count, dtype: int64
```

1.2.2 By week

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

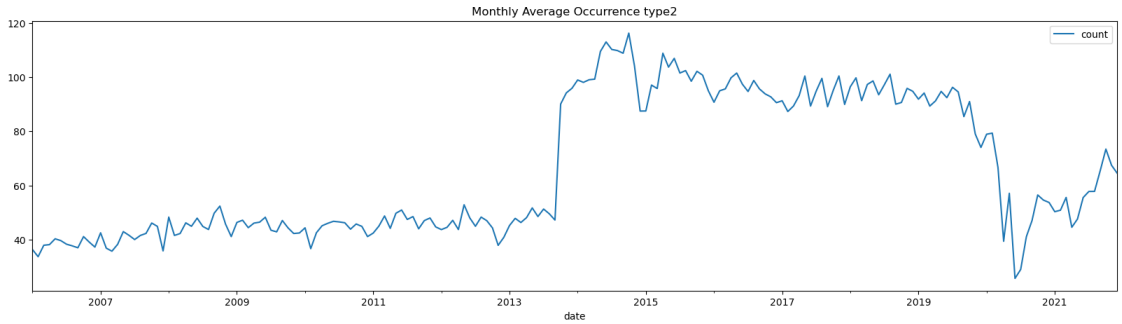
[ ]:
```
df_by_week.plot(
    figsize=fig_size,
    title="Weekly Average Occurrence " + crime)
plt.show()
```



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 : -1.7402553693199236
p-value : 0.4105000692820011
#Lags Used : 3
Number of Observations Used : 188
weak evidence against null hypothesis, time series has a unit root, indicating
it is non-stationary
```

1.3.2 Checking seasonality

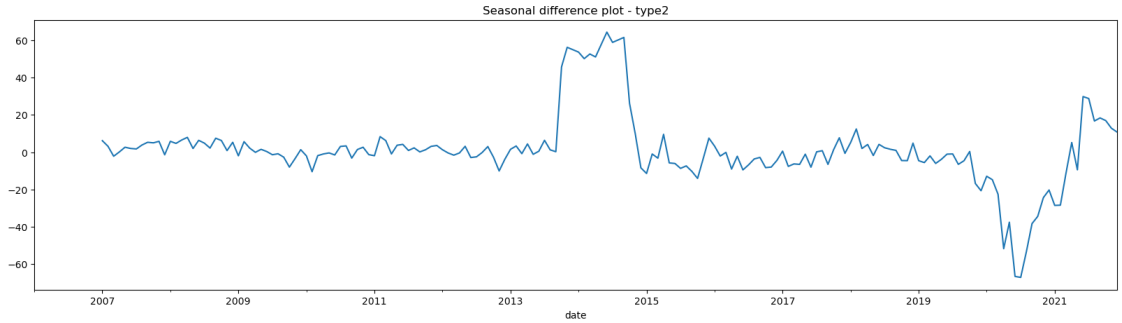
[ ]:
```
df_by_month['seasonal_first_difference'] = df_by_month[target] -␣
↪df_by_month[target].shift(12)
```

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

```
ADF Test Statistic : -2.54028837463391
p-value : 0.1059768507355136
#Lags Used : 12
Number of Observations Used : 167
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)
```

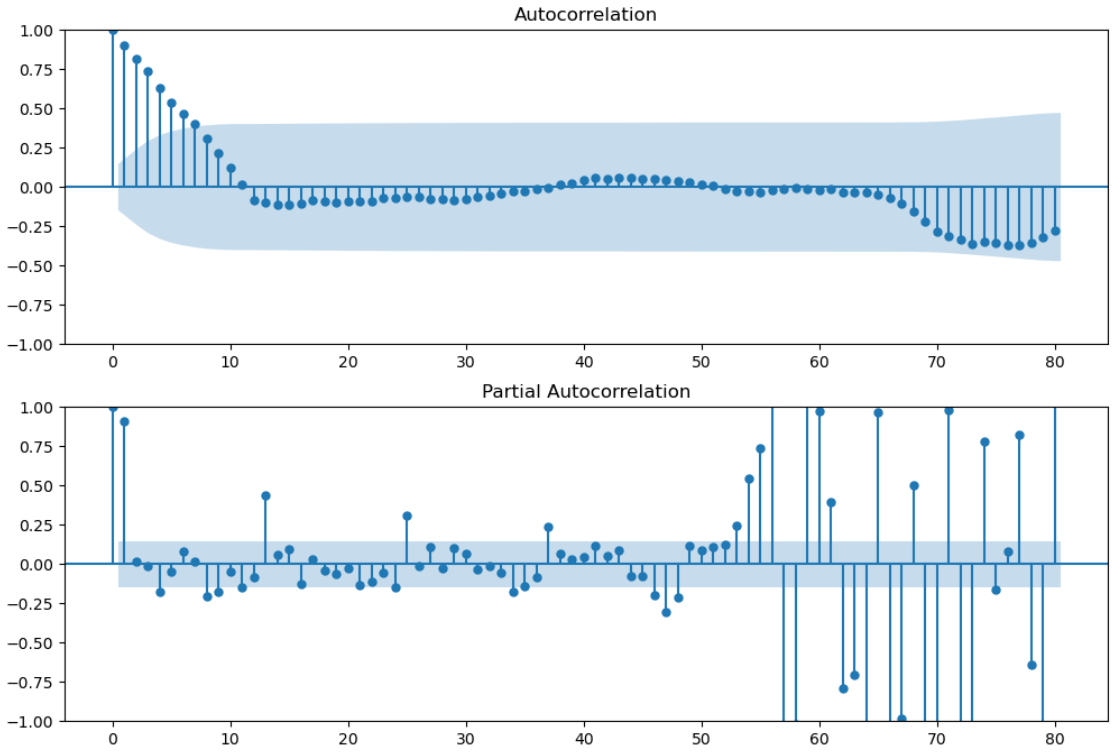
[ ]:
```
<Axes: title={'center': 'Seasonal difference plot - type2'}, xlabel='date'>
```



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(
```

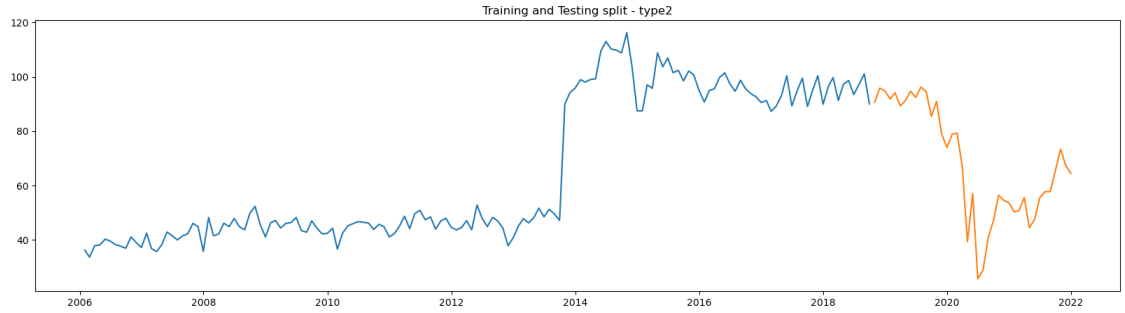


1.3.4 Implementing Seasonal Arima Model

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

[ ]:
```
(0.7932110139413808, 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=954.090, Time=0.06 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=926.233, Time=0.11 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=899.144, Time=0.38 sec
 ARIMA(0,1,1)(0,1,0)[12]             : AIC=956.049, Time=0.03 sec
 ARIMA(0,1,1)(1,1,1)[12]             : AIC=900.900, Time=0.56 sec
 ARIMA(0,1,1)(0,1,2)[12]             : AIC=900.905, Time=2.80 sec
 ARIMA(0,1,1)(1,1,0)[12]             : AIC=926.187, Time=0.12 sec
 ARIMA(0,1,1)(1,1,2)[12]             : AIC=902.899, Time=5.78 sec
 ARIMA(0,1,0)(0,1,1)[12]             : AIC=898.906, Time=0.20 sec
 ARIMA(0,1,0)(1,1,1)[12]             : AIC=900.434, Time=0.31 sec
 ARIMA(0,1,0)(0,1,2)[12]             : AIC=900.442, Time=3.10 sec
 ARIMA(0,1,0)(1,1,0)[12]             : AIC=924.565, Time=0.10 sec
 ARIMA(0,1,0)(1,1,2)[12]             : AIC=902.434, Time=30.05 sec
 ARIMA(1,1,0)(0,1,1)[12]             : AIC=899.431, Time=2.47 sec
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=900.931, Time=3.22 sec
 ARIMA(0,1,0)(0,1,1)[12] intercept   : AIC=900.901, Time=1.51 sec

Best model:  ARIMA(0,1,0)(0,1,1)[12]
Total fit time: 50.826 seconds
```

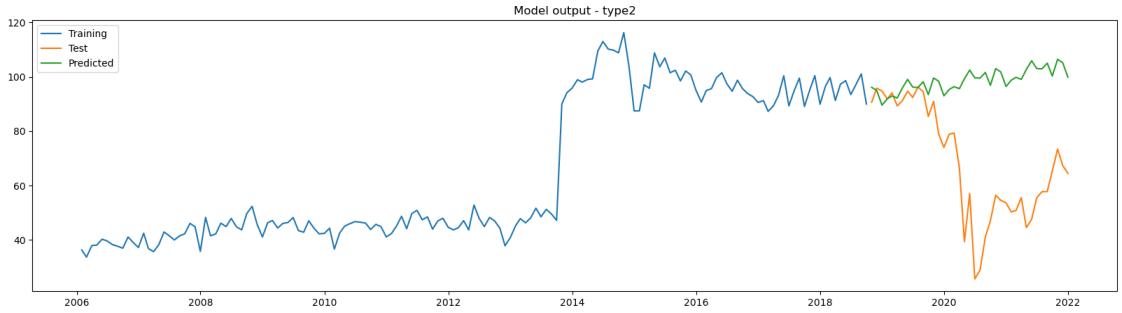
[ ]:
```
model.summary()
```

[ ]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
==========================================================================================
```

```
============
Dep. Variable:                                   y   No. Observations:
153
Model:             SARIMAX(0, 1, 0)x(0, 1, [1], 12)   Log Likelihood
-447.453
Date:                              Sun, 23 Apr 2023   AIC
898.906
Time:                                      01:32:03   BIC
904.789
Sample:                                  01-31-2006   HQIC
901.296
                                       - 09-30-2018
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.S.L12      -0.8152      0.102     -7.956      0.000      -1.016      -0.614
sigma2        31.8544      1.930     16.507      0.000      28.072      35.637
==============================================================================
===
Ljung-Box (L1) (Q):                   1.44   Jarque-Bera (JB):
2264.70
Prob(Q):                              0.23   Prob(JB):
0.00
Heteroskedasticity (H):               3.85   Skew:
2.60
Prob(H) (two-sided):                  0.00   Kurtosis:
22.01
==============================================================================
===

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())
```

[ ]: 37.80735738802341