analysis_chicago_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/crime_occurrence_per_day.csv"
crime = "all"
target = "Count"
date = "Date"
city = "chicago"
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()
```

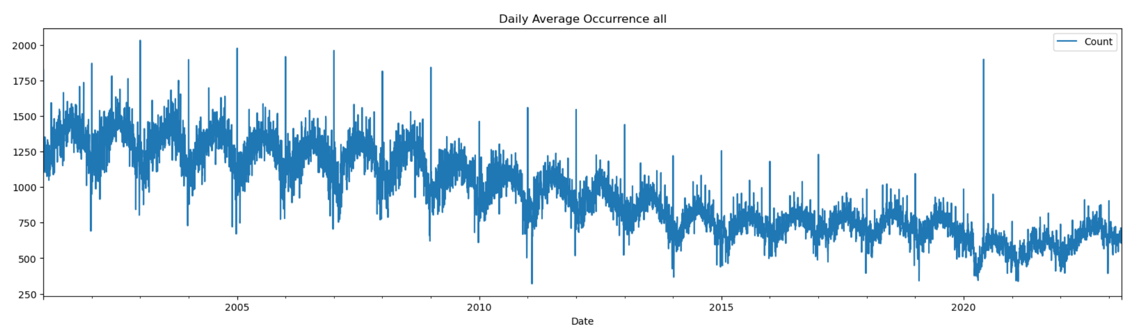
[ ]:
```
            Count
Date
2001-01-01   1825
2001-01-02   1143
```

```
2001-01-03    1151
2001-01-04    1166
2001-01-05    1267
```

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

[ ]:
```
             Count
count  8132.000000
mean    956.034801
std     285.218720
min     320.000000
25%     717.000000
50%     913.000000
75%    1207.000000
max    2033.000000
```

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



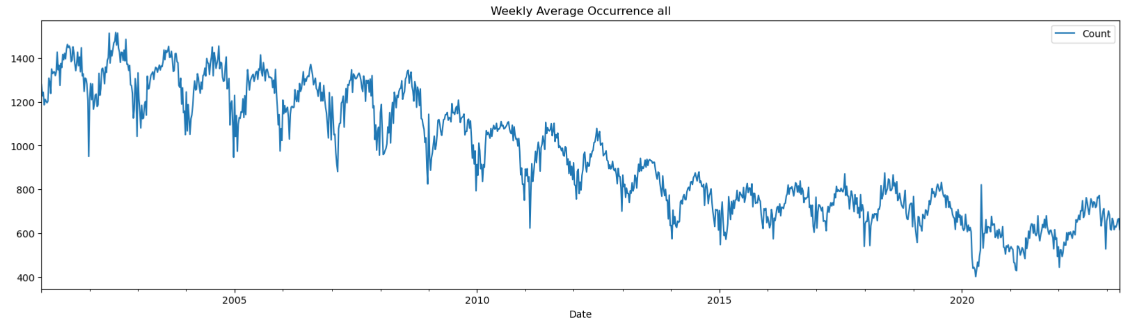
[ ]:
```
df_by_day[target].sort_values(ascending=False).head()
```

[ ]:
```
Date
2003-01-01    2033
2005-01-01    1977
2007-01-01    1961
2006-01-01    1918
2020-05-31    1899
Name: Count, dtype: int64
```

1.2.2 By week

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

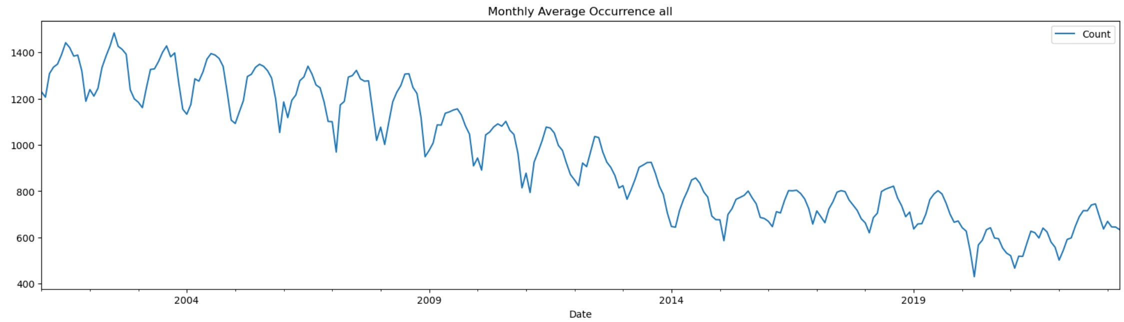
[ ]:
```
df_by_week.plot(
    figsize=fig_size,
    title="Weekly Average Occurrence " + crime)
plt.show()
```



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 : -1.3090286063052534
p-value : 0.6250316590534809
#Lags Used : 15
Number of Observations Used : 252
weak evidence against null hypothesis, time series has a unit root, indicating
it is non-stationary
```

1.3.2 Checking seasonality

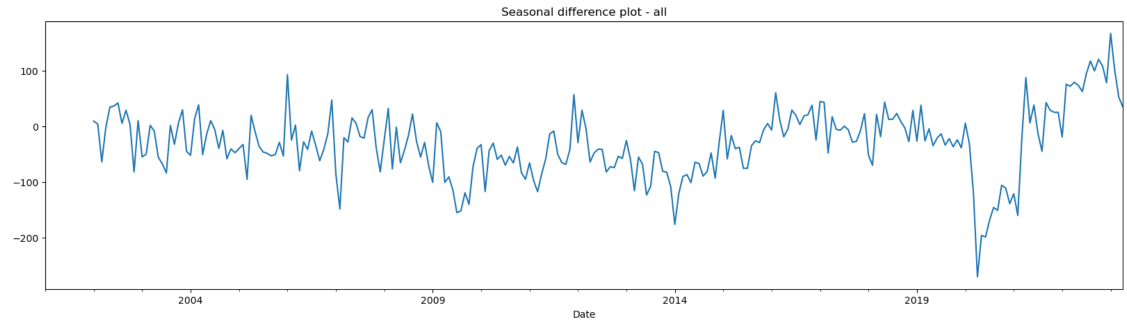
[ ]:
```
df_by_month['seasonal_first_difference'] = df_by_month[target] -␣
↪df_by_month[target].shift(12)
```

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

```
ADF Test Statistic : -2.912294910803625
p-value : 0.043936833679229 69
#Lags Used : 12
Number of Observations Used : 243
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)
```

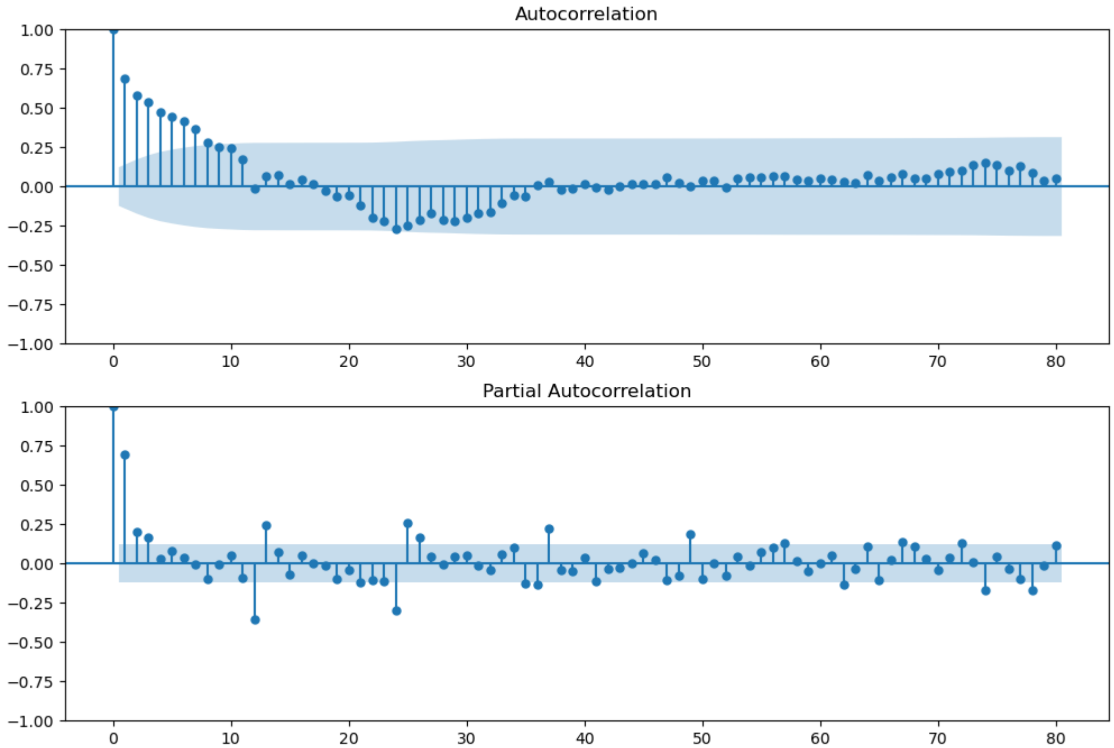
[ ]:
```
<Axes: title={'center': 'Seasonal difference plot - all'}, xlabel='Date'>
```



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(
```

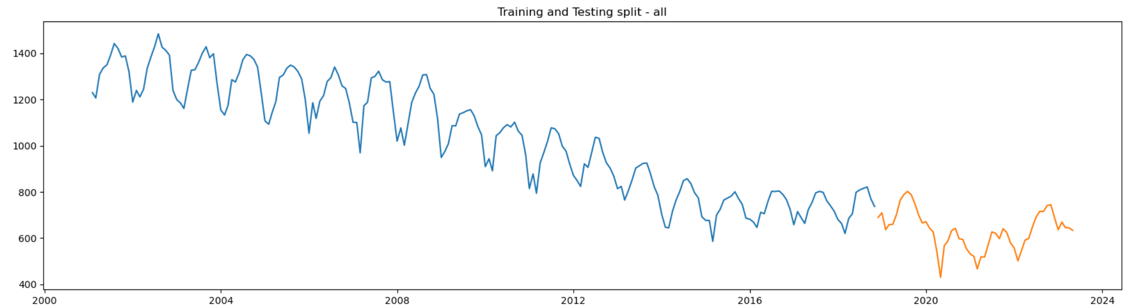


1.3.4 Implementing Seasonal Arima Model

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

[ ]: (0.01, 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()
```



[ ]:
```python
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=2112.466, Time=0.04 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=2045.289, Time=0.49 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=1983.008, Time=3.07 sec
 ARIMA(0,1,1)(0,1,0)[12]             : AIC=2048.790, Time=0.19 sec
 ARIMA(0,1,1)(1,1,1)[12]             : AIC=1984.906, Time=3.66 sec
 ARIMA(0,1,1)(0,1,2)[12]             : AIC=1984.841, Time=26.05 sec
 ARIMA(0,1,1)(1,1,0)[12]             : AIC=2017.890, Time=0.75 sec
 ARIMA(0,1,1)(1,1,2)[12]             : AIC=inf, Time=37.85 sec
 ARIMA(0,1,0)(0,1,1)[12]             : AIC=2028.476, Time=1.61 sec
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=1979.815, Time=3.28 sec
 ARIMA(1,1,1)(0,1,0)[12]             : AIC=2046.407, Time=0.47 sec
 ARIMA(1,1,1)(1,1,1)[12]             : AIC=1981.654, Time=5.32 sec
 ARIMA(1,1,1)(0,1,2)[12]             : AIC=1981.561, Time=27.43 sec
 ARIMA(1,1,1)(1,1,0)[12]             : AIC=2015.549, Time=1.34 sec
 ARIMA(1,1,1)(1,1,2)[12]             : AIC=inf, Time=55.11 sec
 ARIMA(1,1,0)(0,1,1)[12]             : AIC=2004.148, Time=2.41 sec
 ARIMA(2,1,1)(0,1,1)[12]             : AIC=1981.810, Time=3.78 sec
 ARIMA(1,1,2)(0,1,1)[12]             : AIC=1981.812, Time=3.06 sec
 ARIMA(0,1,2)(0,1,1)[12]             : AIC=1980.112, Time=1.82 sec
 ARIMA(2,1,0)(0,1,1)[12]             : AIC=1996.287, Time=1.70 sec
 ARIMA(2,1,2)(0,1,1)[12]             : AIC=1983.547, Time=2.83 sec
 ARIMA(1,1,1)(0,1,1)[12] intercept   : AIC=1981.803, Time=1.64 sec

Best model:  ARIMA(1,1,1)(0,1,1)[12]
Total fit time: 183.924 seconds
```

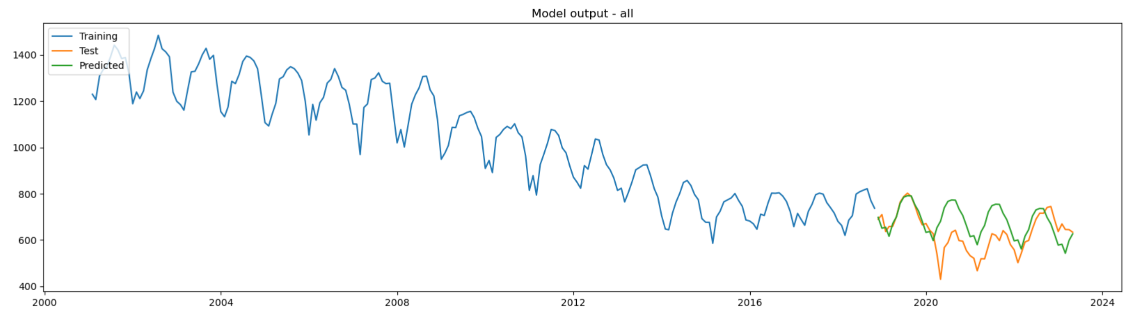
[ ]:
```
model.summary()
```

[ ]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
================================================================================
==========
Dep. Variable:                                  y   No. Observations:
214
Model:             SARIMAX(1, 1, 1)x(0, 1, 1, 12)   Log Likelihood
-985.908
Date:                            Sun, 23 Apr 2023   AIC
1979.815
Time:                                    01:34:43   BIC
1993.029
Sample:                                01-31-2001   HQIC
1985.162
                                     - 10-31-2018
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.2290      0.116      1.967      0.049       0.001       0.457
ma.L1         -0.7716      0.075    -10.324      0.000      -0.918      -0.625
ma.S.L12      -0.6947      0.055    -12.529      0.000      -0.803      -0.586
sigma2      1021.9363     82.140     12.441      0.000     860.945    1182.927
================================================================================
===
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
14.24
Prob(Q):                              0.99   Prob(JB):
0.00
Heteroskedasticity (H):               0.62   Skew:
0.15
Prob(H) (two-sided):                  0.05   Kurtosis:
4.27
================================================================================
===

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())
```

[ ]: 89.9247972781049