Time series Forecasting using ARIMA

We followed the Box-Jenkins method of applying this ARIMA algorithm on the data we have With the Data being preprocessed and ready for the forecasting and prediction. So now it is the machine learning part of the Capstone.

Since our data is a time series data we have chosen a machine learning algorithm which best suit the time series Data. The ARIMA (Autoregressive integrated moving average). Both of these models are fitted to time series data either to better understand the data or to predict future points in the series (forecasting).

Starting first by visualizing the data by decomposition

Getting the Data from the data set created for crude oil and canadian dollar

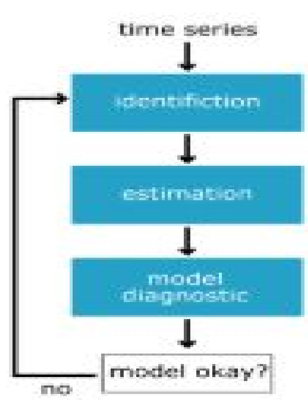
```
df_cont=pd.concat([df['crude price'],df_cad['Cad price']],axis=1)
df_cont.dropna().head()
```

	crude price	Cad price
Date		
1983-04-07	29.45	0.8087
1983-04-08	29.90	0.8093
1983-04-11	29.80	0.8095
1983-04-12	30.40	0.8103
1983-04-13	30.45	0.8103

```
## Spliting the data to train and test data
df_train=df_cont.loc['2000':'2016']
df_test=df_cont.loc['2017':]
```

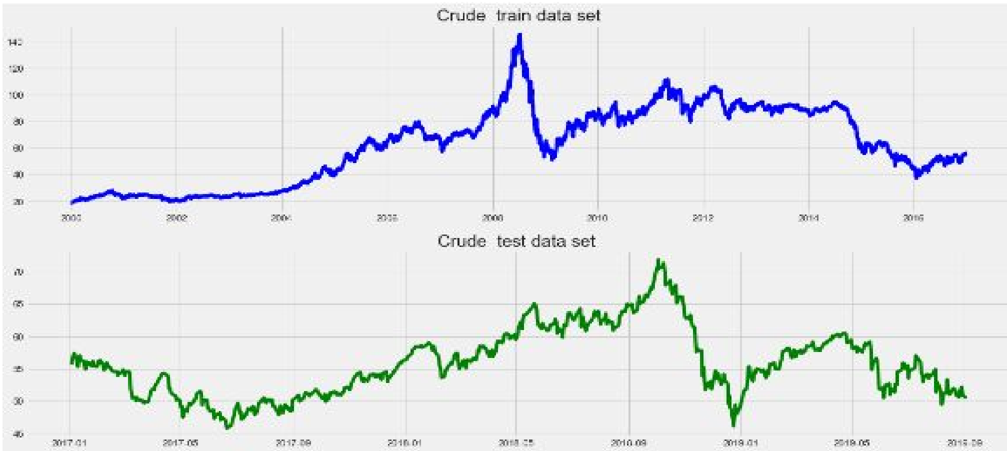
```
##df_crude=df_cont.drop(['Cad price'],axis=1).loc['2016':'01-01-2017']
df_crude=df_cont.drop((['Cad price']),axis=1)
df_crude_train=df_train.drop(['Cad price'],axis=1)
df_crude_test=df_test.drop(['Cad price'],axis=1)
```

Box-Jenkins method



1- Identification step:

A- Plot time series



B- Apply the adfuller test:

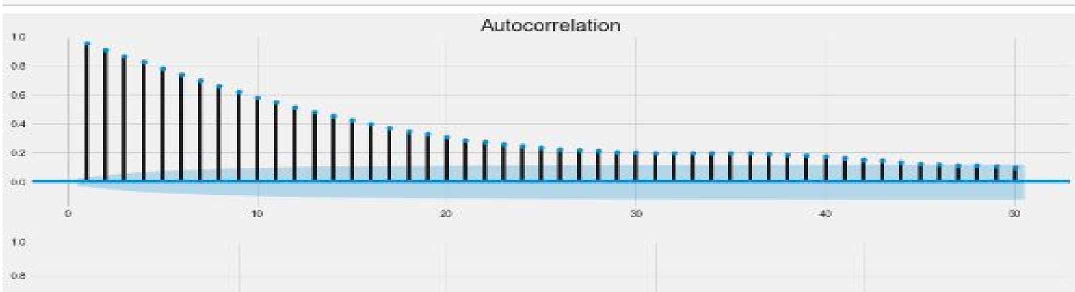
The first test before the modeling is to check whether the time series is stationary or not in case it is not stationary we have to change it to stationary series .one way is the Adfuller test where the following test shows that the time series is not stationary and we should make it stationary where the p-value is 0.4 > a 0.05 to reject the null hypothesis.

```
adf = adfuller(df_crude['crude price'])
print('ADF Statistic: {}'.format(adf[0]))
print('p-value: {}'.format(adf[1]))
print('Critical Values:')
for key, value in adf[4].items():
        print('\t{}: {}'.format(key, value))
```

```
ADF Statistic: -1.7232418286062188
p-value: 0.41917911729762747
Critical Values:
        1%: -3.4310759087573603
        5%: -2.8618608046125042
        10%: -2.5669407592642446
```

Testing for the seasonality of the DATA

```
# ploting the ACF  for Crude oil Detrended data
bf=df_crude-df_crude.rolling(30).mean()
bf=bf.dropna()
fig,(ax1,ax2)=plt.subplots(2,1,figsize=(16,8))
plot_acf(bf,lags=50,zero=False ,ax=ax1)
plt.show()
```



The above plot proves that data is not seasonal

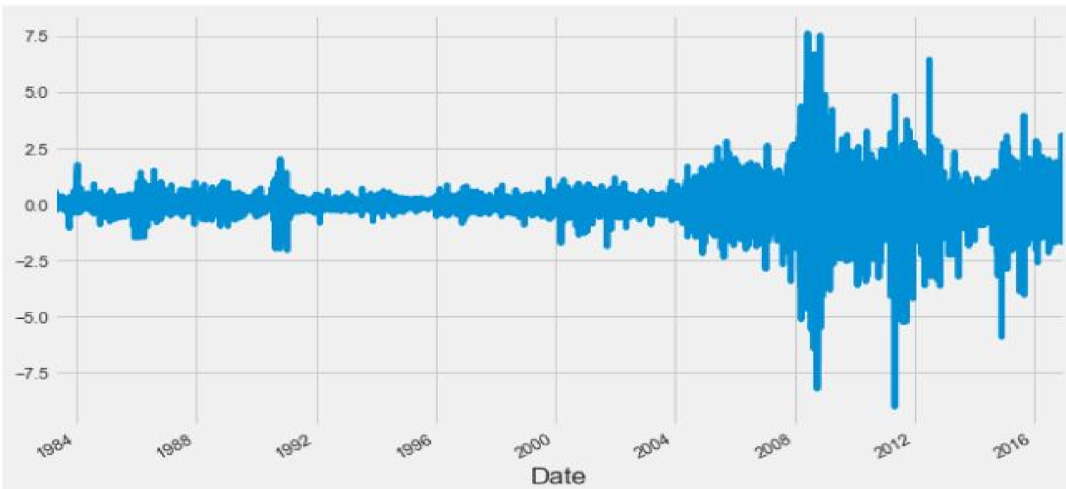
C- Transforming Data by Differencing:

Taking the first difference to make the data stationary

```
df_diff=df_train['crude price'].diff()
df_diff=df_diff.dropna()
```

```
df_diff.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x28ffd431cf8>
```



Making sure that the first difference is enough for the data to be stationary .

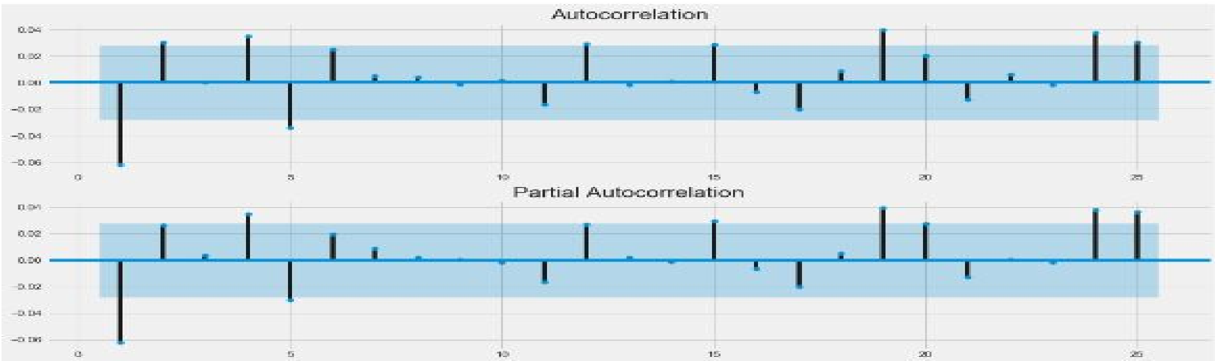
```
# printing the adfuller of the df_diff
result = adfuller(df_diff)
print('ADF Statistic: {}'.format(result[0]))
print('p-value: {}'.format(result[1]))
print('Critical Values:')
for key, value in result[4].items():
        print('\t{}: {}'.format(key, value))
```

```
ADF Statistic: -10.65760348397638
p-value: 4.492253381546068e-19
Critical Values:
        1%: -3.4316937565980177
        5%: -2.862133791594171
        10%: -2.5670860765896855
```

D- Plotting the ACF and PACF

Plotting the ACF and PACF would be directional to select the value of the order p and q for the ARIMA model

```
# ploting the ACF and PCF for Crude oil
fig,(ax1,ax2)=plt.subplots(2,1,figsize=(16,8))
plot_acf(df_diff,lags=25,zero=False ,ax=ax1)
plot_pacf(df_diff,lags=25,zero=False,ax=ax2)
plt.show()
```



2- Estimation to pick best p and q

Looping over a range of values for p and q to choose the best fitting Model

```
order_aic_bic=[]
for p in range(4):
    for q in range (4):
        # fit the model
        model=SARIMAX(df_diff,order=(p,0,q))
        results= model.fit()
        order_aic_bic.append((p,q,results.aic,results.bic))
```

```
order_df=pd.DataFrame(order_aic_bic,columns=['p','q','AIC','BIC'])
order_df.sort_values('AIC').head()
```

	p	q	AIC	BIC
15	3	3	12798.275994	12849.069977
11	2	3	12803.860288	12848.305023
14	3	2	12803.881509	12848.326243
5	1	1	12808.339248	12833.736239
13	3	1	12808.344614	12846.440100

```
# since the Lowest AIC corresponds to p of 3 and q of 3 and  first level difference
model=SARIMAX(df_crude_train,order=(3,1,3))
result= model.fit()
print(result.summary())
```

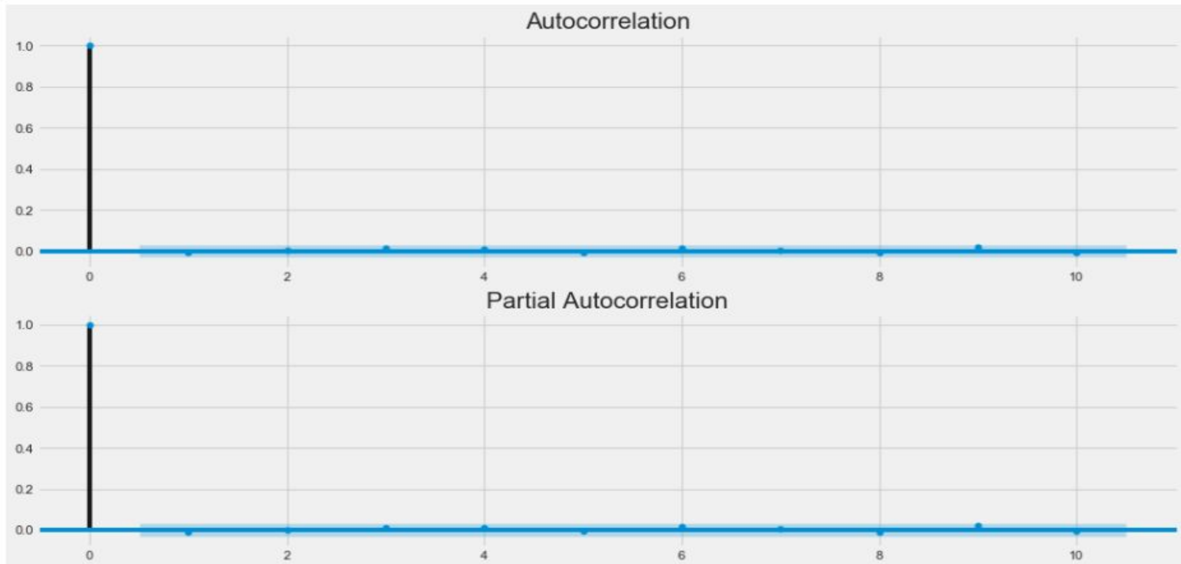
```
                           Statespace Model Results
==============================================================================
Dep. Variable:            crude price   No. Observations:                 4228
Model:               SARIMAX(3, 1, 3)   Log Likelihood               -6390.561
Date:                Mon, 09 Sep 2019   AIC                          12795.123
Time:                        22:04:44   BIC                          12839.567
Sample:                             0   HQIC                         12810.833
                               - 4228
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.8484      0.040    -21.222      0.000      -0.927      -0.770
ar.L2         -0.9320      0.015    -62.491      0.000      -0.961      -0.903
ar.L3         -0.8483      0.038    -22.361      0.000      -0.923      -0.774
ma.L1          0.8008      0.043     18.641      0.000       0.717       0.885
ma.L2          0.9195      0.015     62.775      0.000       0.891       0.948
ma.L3          0.8234      0.041     19.966      0.000       0.743       0.904
sigma2         1.2061      0.012     99.550      0.000       1.182       1.230
===================================================================================
Ljung-Box (Q):                       85.68   Jarque-Bera (JB):             10284.56
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               4.89   Skew:                            -0.46
Prob(H) (two-sided):                  0.00   Kurtosis:                        10.59
===================================================================================

Warnings:
```

3- Model Diagnostic

Plotting the Residual of the model

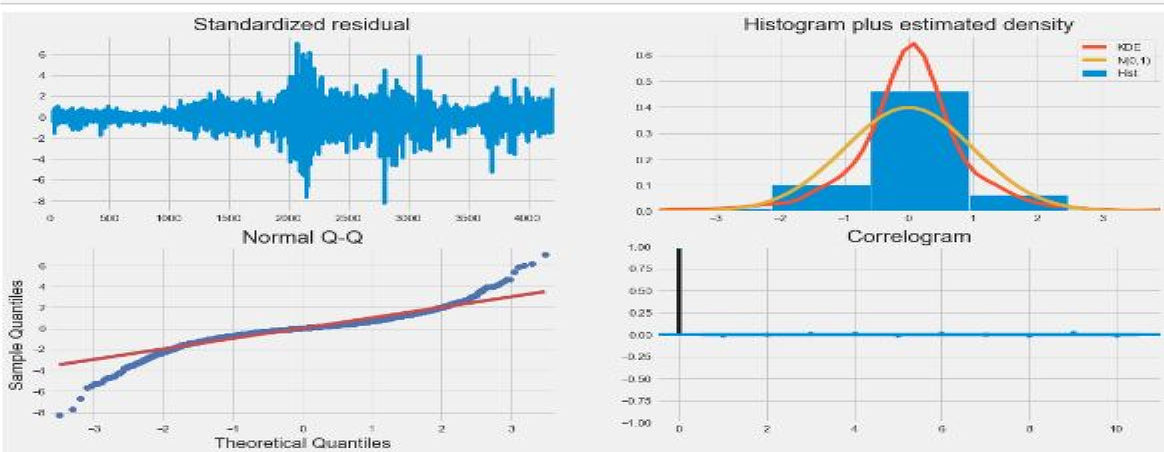
```
res = result.resid
fig,ax = plt.subplots(2,1,figsize=(15,8))
fig = sm.graphics.tsa.plot_acf(res, lags=10, ax=ax[0])
fig = sm.graphics.tsa.plot_pacf(res, lags=10, ax=ax[1])
plt.show()
```



The above plot of the residuals shows that there is no correlation between the residuals which means that the model did not miss any aspect of trend or seasonality

Also a validation step would be applying or calling the plot_diagnostic on the fitted model which shows a validated model based on the normal distribution of the data

```
result.plot_diagnostics()
plt.show()
```



The diagnostic test states that the model is a good fit for the data where the residuals are normally distributed as the Histogram shows as for the Normal Q-Q shows the residuals are normally distributed for the correlogram shows there is no correlation among the residuals

Since everything looks fine now we are to predict and forecast the prices of crude oil.

```
## let us now forecast in sample
forecast1=result.get_prediction(start=-50)
mean_forecast1=forecast1.predicted_mean
confidence_intervals= forecast1.conf_int()
lower_limits1= confidence_intervals.loc[:,'lower crude price']
upper_limits1= confidence_intervals.loc[:,'upper crude price']
```

```
plt.plot(df_crude.index,df_crude,color='blue')
plt.plot(mean_forecast1.index,mean_forecast1,color='red')
plt.fill_between(lower_limits1.index,lower_limits1,upper_limits1,color='pink')
plt.show()
```

Here below is the mean average error for the forecast

```
# CALCULATING THE Mean absolute error of the residuals
residuals =result.resid
mae=np.mean(np.abs(residuals))
mae
```

```
0.7259000819075213
```

```
## forecasting the future
import scipy.stats
from sklearn.metrics import mean_squared_error
forecast2=result.get_forecast(steps=10)
mean_forecast2=forecast2.predicted_mean
confidence_intervals= forecast2.conf_int()
lower_limits2 = confidence_intervals.loc[:,'lower crude price']
upper_limits2 = confidence_intervals.loc[:,'upper crude price']
```

```
print(mean_forecast2)
```

```
4228    56.623144
4229    56.568890
4230    56.587315
4231    56.594134
4232    56.617200
4233    56.575647
4234    56.583617
4235    56.596016
4236    56.613317
4237    56.580323
dtype: float64
```