Time Series

To start off our report, let's first understand what is time series? Time series is the data which changes over a period. An example of this would be stock price or sales growth.

There are a few models which allow us to analyze Time series. These models allow us to forecast the observation based on the historical data of the previous time phases for the same observation.

A small condition to this prediction method is to know whether the data is stationary or not, if it is **not** stationary then we apply the differencing factor and conduct the tests once more to check for its prediction and how accurate it may be.

1. Different types of Variation

Four kinds of Variation have impact on Time series. These are:

Seasonal Trend:

A short-term trend with in a year that may be caused by climatic changes, holidays, temperature or rainfall etc. This type is easy to understand or measure. Also we can say that seasonal variations are the changes that can repeat themselves within a fixed period of time.

Secular Trend:

A long-term trend that is more than 5 years. The trend may be Increasing or Decreasing trend. Secular Trend can be linear and Non-linear.

Cyclic Variation:

Cyclic Variation also called a business cycle is a long-term trend where changes occur within 3 to 10 years. It is periodic in nature and have peak and troughs. These variations are not regular like seasonal variations.

Irregular Variations:

It is a short-term trend that cannot be predicted and are caused by sudden causes like floods, earth quakes, war etc. It is obtained by removing trend and cyclic variations from the original set of time series data.

If we explain irregular variation in terms of probability such as by using method of moving average or using AR (autoregressive) models, cyclic variations are left in residuals.

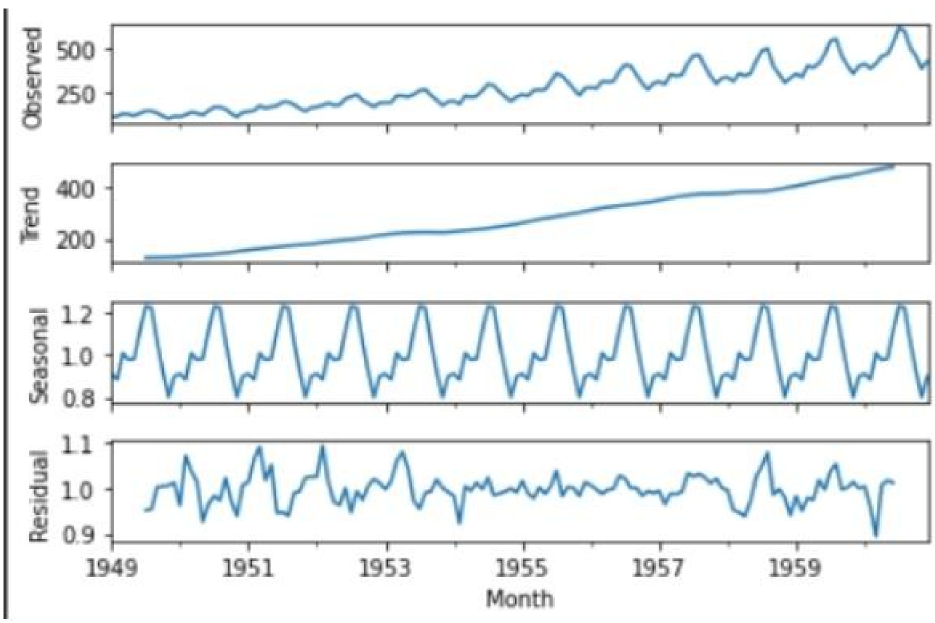
Now we will get a plot by doing seasonal decomposition on data. It is done by first importing library.

```
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
data_Mul_Decompose = seasonal_decompose(data,model="multiplicative")

data_Mul_Decompose.plot()
plt.show()
```

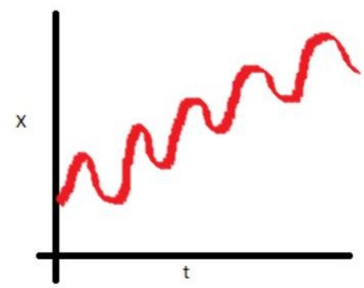
Output:



We get the residual by differencing trend and seasonal variation from the original data. Residual is the leftover value, and it shows how the points of data deviate from the model.

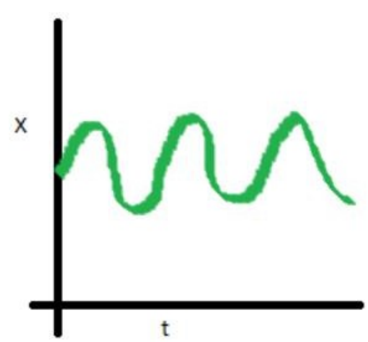
2. Dickey-Fuller test

A lot of tests are available to test if your time series stationary or non-stationary. DT or **Dicky Fuller test** is the test which determines and sets apart stationary from non-stationary data. But before going to discuss dicky fuller test we check graphically if our graph shows curve according to the below figure then our series will be nonstationary.



Non-Stationary series

And if our graph curve is moving in such a way that the average and time both increase together then series will be stationery and graph looks like this:



Stationary series

Let us now discuss the dicky fuller test. We conduct the **Dicky fuller** test using **Adfuller** from **statsmodel.**

```
from statsmodels.tsa.stattools import adfuller
```

Adfuller function calculate ADF statistics, lags, p-value and critical value. We notice p-value, if p-value is below 0.05 then series will be stationary on the other hand if value will be greater than or equal to 0.05 then the series shows nonstationary behavior. Critical values are the probability of a series to be stationary for example, at 99% its -3.48, 95% -2.88 and 90% it's -2.57.

```
result = adfuller(series, autolag='AIC')

print(f'ADF Statistic: {result[0]}')
print(f'n_lags: {result[1]}')
print(f'p-value: {result[1]}')
for key, value in result[4].items():
    print('Critial Values:')
    print(f'   {key}, {value}')
```

```
ADF Statistic: 0.8153688792060472
n_lags: 0.991880243437641
p-value: 0.991880243437641
Critial Values:
   1%, -3.4816817173418295
Critial Values:
   5%, -2.8840418343195267
Critial Values:
   10%, -2.578770059171598
```

3. ACF & PACF

- ACF (Auto-Correlation Function)
  - The correlation coefficients between lagged values can be plotted to form the autocorrelated function also known as correlogram.
- PACF (Partial Auto-Correlation Function)
  - PACF is a plot that shows the correlation of a series with its lagged values.

You can determine the ARIMA model's AR and MA components using the ACF and PACF plots. ACF and PACF plots can be used to determine both seasonal and non-seasonal AR and MA components.ACF and PACF are used to find values for P, D, Q in ARIMA model.

ARIMA model has 3 components.

P order of AR.

Q order of MA

Integrated ~~ Difference (D)

Auto Regression is a correlation with its past values. P from PACF Plot; if p=3 that means Yt is dependent on past 3 periods. Moving Average has the effects of previous error terms in it. Q from ACF plot. For P current value of y is dependent on how many previous lagged values of current Y. For Q future values of Y is dependent of previous lagged values of white noise i.e., the irregular component. white noise is just the error. The term error here refers to the difference between the actual predicted value. so, we take into consideration the error also to predict the future value.

For I Integrated means no of times we difference the data then we must integrated it back to get the original series back.

For AR we have the formula:

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} + \epsilon_1$$

(T-1) is the lag1 of the series, beta1 represents the coefficient of lag1 that the model estimates, and alpha represents the intercept.

For MA we have the formula

$$Y_t = \alpha + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + \ldots + \phi_q \epsilon_{t-q}$$

Integrated ~~ Differenced

Integrated means no of times we difference the data then we have to integrated it back to get the original series back.

We difference to remove trend and seasonality to it stationary series as only after making a series stationary we can implement AR and MA.

D is order: How many times we difference the data.

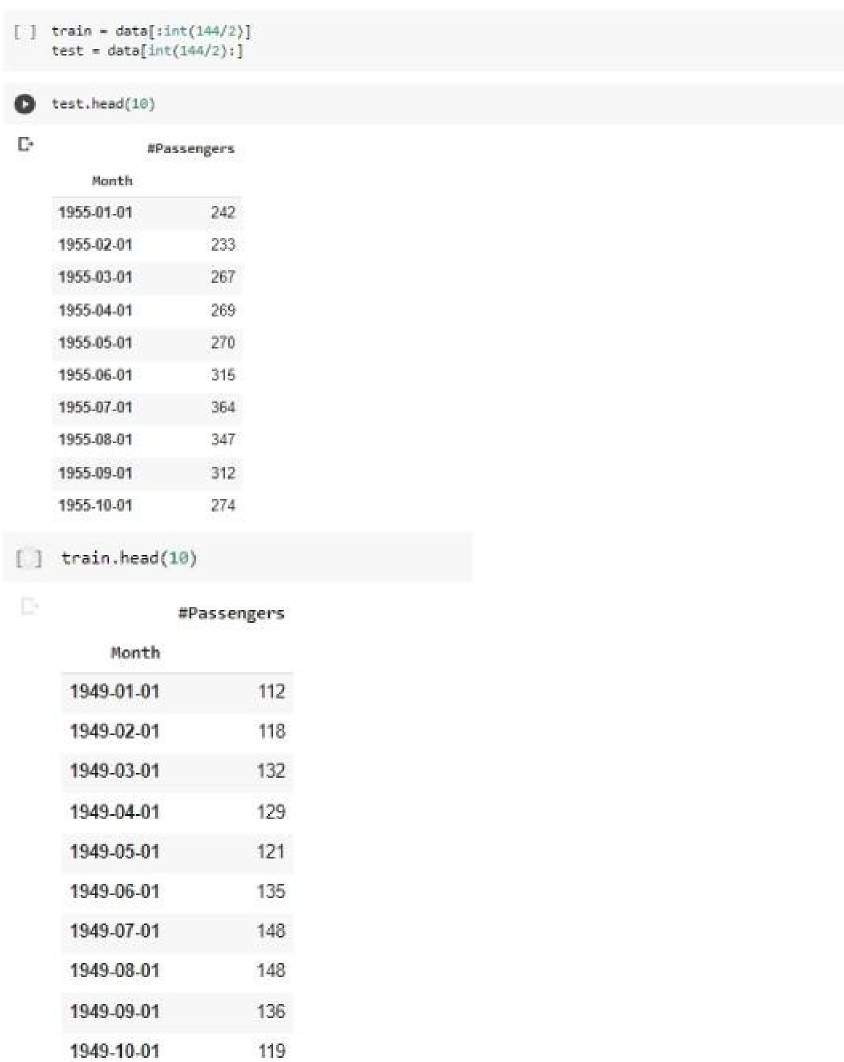
NOTE: If both P and Q are positive, then the plots are not helpful in determining what values of P and Q are appropriate.

4. ARIMA model

ARIMA model is one of the most Used models to predict the outcome of the future events Which have Occurred with respect to Time. This model Is used when we have Nonstationary data.

First if it’s confirmed our data is non-Stationary then we have to performance of our model For which we have to train our model and then test it with known values. Then we will find the difference in between.

1) We are going to Split our data set into Test and Train data.



```
train = data[:int(144/2)]
test = data[int(144/2):]
```

```
test.head(10)
```

Month	#Passengers
1955-01-01	242
1955-02-01	233
1955-03-01	267
1955-04-01	269
1955-05-01	270
1955-06-01	315
1955-07-01	364
1955-08-01	347
1955-09-01	312
1955-10-01	274

```
train.head(10)
```

Month	#Passengers
1949-01-01	112
1949-02-01	118
1949-03-01	132
1949-04-01	129
1949-05-01	121
1949-06-01	135
1949-07-01	148
1949-08-01	148
1949-09-01	136
1949-10-01	119

Both the Split-ed data's are showing Above.

2) Analyzing the values from the ACF and PACF graphs are difficult And then we still don't know if it order P,D,Q gives the most Efficient model for which we have created all possible combinations of orders and then find the combination of P,D,Q with the least Root Mean Square Error.

```
[ ] import itertools
    import warnings
    warnings.filterwarnings("ignore")

[ ] p = range(0,8)
    d = range(0,2)
    q = range(0,8)

[ ] combination = list(itertools.product(p,d,q))

[ ] import statsmodels.api as sm

[ ] from sklearn.metrics import mean_squared_error
    from statsmodels.tsa.arima_model import ARIMA

[ ] orders = []
    rmse = []
    for i in combination:
      try:
        model = ARIMA(train,order = i).fit()
        pred = model.predict(start = len(train),end = len(da
        error = np.sqrt(mean_squared_error(test,pred))
        rmse.append(error)
        orders.append(i)
      except:
        continue

[ ] p,d,q = orders[rmse.index(min(rmse))]
    print(p,d,q)
    print(min(rmse))
```

This is the screen Shot giving us the most Efficient Model which has order p = 3, D = 0 , Q = 3, With the Minimum Root mean square value of 61.9. And in the End printing all possible combination P,D,Q.

```
[ ] p,d,q = orders[rmse.index(min(rmse))]
    print(p,d,q)
    print(min(rmse))

    rmse
    orders

    3 0 3
    61.92346825647696
    [(0, 0, 1),
     (0, 1, 1),
     (0, 1, 2),
     (0, 1, 3),
     (0, 1, 4),
     (0, 1, 6),
     (1, 0, 0),
     (1, 0, 1),
     (1, 0, 2),
     (1, 0, 3),
     (1, 0, 4),
     (1, 1, 0),
```

3) Now Fitting/Training the model with training data and getting the prediction with testing data After which printed the Predicted data.

```
[ ] from statsmodels.tsa.arima_model import ARIMA

    finalModel = ARIMA(train,order = (p,d,q)).fit()

    prediction = finalModel.predict(start = len(train),end = len(data)-1)
```

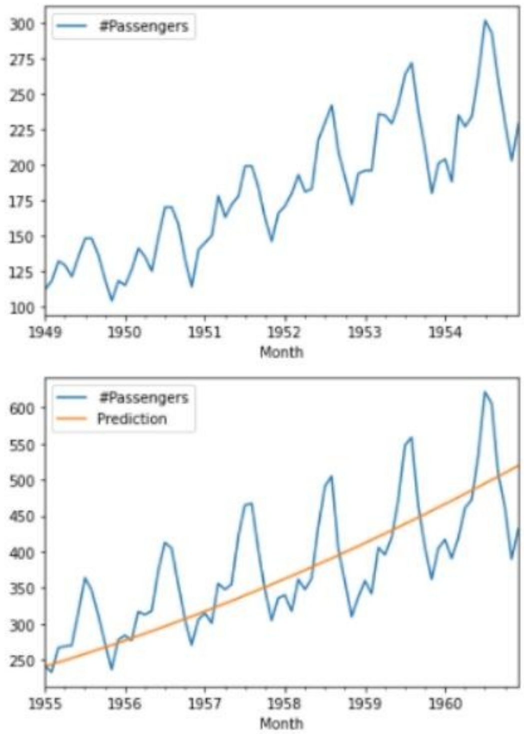
```
[ ] prediction.head(10)

    1955-01-01    241.148778
    1955-02-01    243.826013
    1955-03-01    246.797654
    1955-04-01    249.541605
    1955-05-01    252.578909
    1955-06-01    255.389379
    1955-07-01    258.492149
    1955-08-01    261.368936
    1955-09-01    264.536972
    1955-10-01    267.479870
    Freq: MS, dtype: float64
```

4) Plotting the Data. First Graph contains the graph of Training Data using which we got our prediction on training data. And Printing training and Prediction both.

```
train.plot(legend = True , label = "Train")
test.plot(legend = True , label = "Test")
prediction.plot(legend = True , label = "Prediction")

plt.show()
```



We Can see that the Prediction is also Accurate in the form of Trend Line. ARIMA don’t give us the Seasonal Variation.

5. SARIMA Model

Just like ARIMA that is used to predict in a linear pattern we use seasonal ARIMA that predicts the output in a seasonal manner. In SARIMA model four new hyper parameters are included in existing ARIMA model for the conversion that are seasonal (p,d,q) or seasonal order and frequency.

Mathematical Equation:

$$\underbrace{(1-\phi_1 B)}_{\begin{pmatrix}\text{Non-seasonal}\\ \text{AR(1)}\end{pmatrix}}\ \underbrace{(1-\Phi_1 B^4)}_{\begin{pmatrix}\text{Seasonal}\\ \text{AR(1)}\end{pmatrix}}\ \underbrace{(1-B)}_{\begin{pmatrix}\text{Non-seasonal}\\ \text{difference}\end{pmatrix}}\ \underbrace{(1-B^4)}_{\begin{pmatrix}\text{Seasonal}\\ \text{difference}\end{pmatrix}} y_t = \underbrace{(1+\theta_1 B)}_{\begin{pmatrix}\text{Non-seasonal}\\ \text{MA(1)}\end{pmatrix}}\ \underbrace{(1+\Theta_1 B^4)}_{\begin{pmatrix}\text{Seasonal}\\ \text{MA(1)}\end{pmatrix}} e_t.$$

Coding SARIMA MODEL:

First of all we need normal (p,d,f) that we have obtained using acf and pacf. Then we will import the built in SARIMAX function from python's statsmodels library.

```
[28] from statsmodels.tsa.statespace.sarimax import SARIMAX
```

Now we will initialize pdq with a range and make a list using itertools.Here we are gonna use nested for loop for the combined possible outcomes for our seasonal and non seasonal pdf and

$\text{ARIMA}\ (p, d, q) \times (P, D, Q)S$

s(frequency).

For this purpose we used try except condition so that if the result will return false it will continue the next iteration until all the optimal outcomes are obtained and stored in a list.

```
[29] p = range(0, 3)
     d = range(1,2)
     q = range(0, 3)
     pdq = list(itertools.product(p, d, q))
     seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
     print('Examples of parameter combinations for Seasonal ARIMA...')
     print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
     print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
     print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
     print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))

     for param in pdq:
         for param_seasonal in seasonal_pdq:
             try:
                 mod = sm.tsa.statespace.SARIMAX(y,
                                                 order=param,
                                                 seasonal_order=param_seasonal,
                                                 enforce_stationarity=False,
                                                 enforce_invertibility=False)
                 results = mod.fit()
                 print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
             except:
                 continue
```

Which yields the result of possible combinations that are

```
Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 1, 1) x (0, 1, 1, 12)
SARIMAX: (0, 1, 1) x (0, 1, 2, 12)
SARIMAX: (0, 1, 2) x (1, 1, 0, 12)
SARIMAX: (0, 1, 2) x (1, 1, 1, 12)
```

Now we choose 12 as frequency because it is seasonal data over 12 months (1 year). Then we choose one of the possible combinations from above and printed the summary using model. Fit() and print command

```
[30] mod = sm.tsa.statespace.SARIMAX(train,
                                     order=(0,1,1),
                                     seasonal_order=(0,1,1,12),
                                     enforce_stationarity=False,
                                     enforce_invertibility=False)
     results=mod.fit()
     print(results.summary())
```

We will obtain the following result

```
[30] mod = sm.tsa.statespace.SARIMAX(train,
                                     order=(0,1,1),
                                     seasonal_order=(0,1,1,12),
                                     enforce_stationarity=False,
                                     enforce_invertibility=False)

     results=mod.fit()
     print(results.summary())
```

For the prediction purpose we will use the length of training data as the starting point and length of data till the end as the ending point.

```
[31] pred = results.predict(start=(len(train)), end=(len(data)-1))
```

To check fo the correctness we use pred.tail() that shows the number of passengers accordingly to the date.

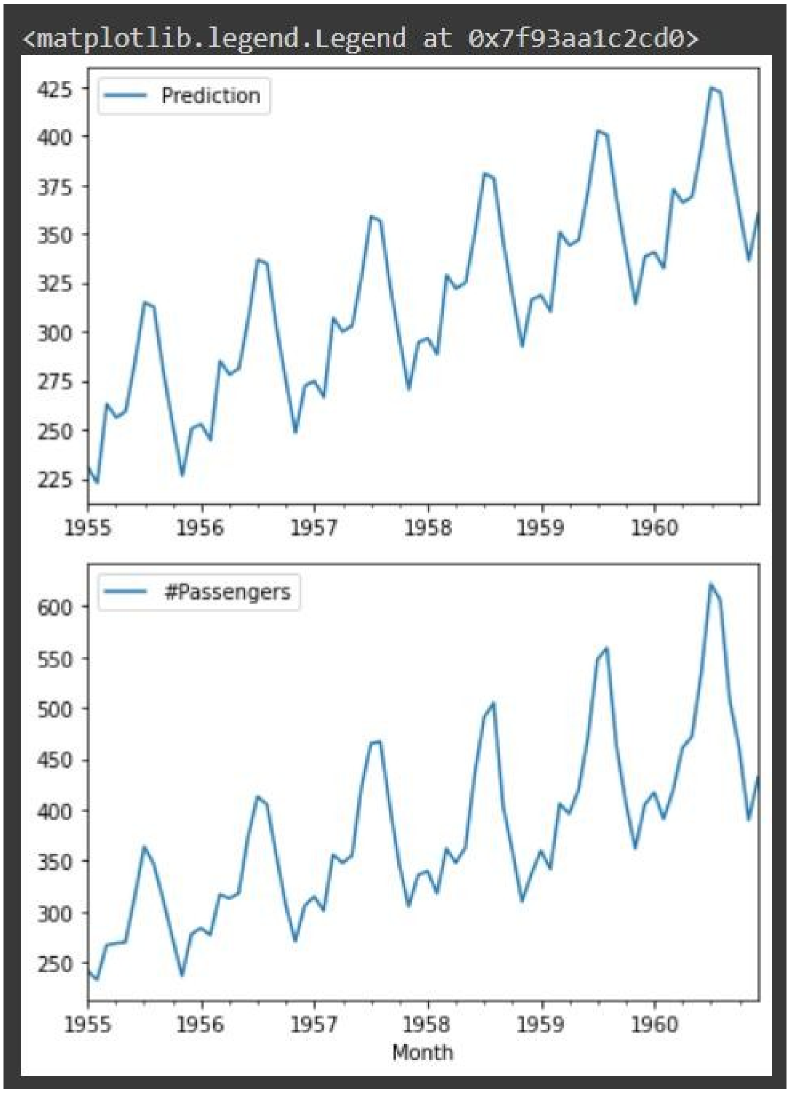
```
[32] pred.tail()

     1960-08-01    422.163567
     1960-09-01    389.334430
     1960-10-01    361.977391
     1960-11-01    336.063736
     1960-12-01    360.128563
     Freq: MS, dtype: float64
```

Now we will plot the Prediction and test using matplotlib library .

```
pred.plot(legend = True , label = "Prediction")
test.plot(legend = True , label = "Test")
plt.legend()
```

The graphs that we obtain are as follows:



Summary:

From the above graphs we can clearly see that the test results we obtained are seasonal unlike ARIMA model that gives linear prediction results.

6. ARMA model

ARMA is combined from AR (Auto-Regressive) and MA (Moving Average). It means when we are predicting future, we will consider impact of previous lags as well as residual too.

$$\mathbf{Yt = \beta_1 * y_{t-1} + \alpha_1 * \varepsilon_{t-1} + \beta_2 * y_{t-2} + \alpha_2 * \varepsilon_{t-2} + \beta_3 * y_{t-3} + \alpha_3 * \varepsilon_{t-3} + \ldots\ldots\ldots\ldots\ldots + \beta_k * y_{t-k} + \alpha_k * \varepsilon_{t-k}}$$

Here, **β** represents coefficients of AR and a represents coefficients of MA.

ARMA is used to predict future using stationary data. We first apply Dicky fuller test, if result is less than 0.5 it means our data is stationary and now, we can apply ARMA on it. After that we find ACF and PACF to get orders of AR and MA. ACF is used to find order of AR and PACF is used to find order of MA.

- **Information about data:**

	A	B
1	Date	Total
2	1/1/1986	9034
3	2/1/1986	9596
4	3/1/1986	10558
5	4/1/1986	9002
6	5/1/1986	9239
7	6/1/1986	8951
8	7/1/1986	9668
9	8/1/1986	10188
10	9/1/1986	9896
11	########	10649
12	########	8917
13	########	8196
14	1/1/1987	10768
15	2/1/1987	12220
16	3/1/1987	14463
17	4/1/1987	12944
18	5/1/1987	11001
19	6/1/1987	11000
20	7/1/1987	11876
21	8/1/1987	13021
22	9/1/1987	13494

We have data of total catfish sale in years. This data is stationary which means we can apply ARMA model on it. Data is so much huge so, we just have pasted screenshot of some piece of data.

Code Screenshots:

```
] import pandas as pd
  import numpy as np
  import matplotlib.pyplot as plt
  from datetime import datetime
  from datetime import timedelta
  from pandas.plotting import register_matplotlib_converters
  from statsmodels.tsa.stattools import acf, pacf
  from statsmodels.tsa.arima_model import ARMA
  register_matplotlib_converters()
  from time import time
  from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
  from statsmodels.tsa.stattools import adfuller
```

First, we include all necessary libraries needed to apply ARMA.

```
[ ] def parser(s):
        return datetime.strptime(s, '%Y-%m-%d')
    #read data
    catfish_sales = pd.read_csv('catfish.csv', parse_dates=[0], index_col=0, squeeze=True, date_parser=parser)
```

```
[ ] catfish_sales   #Displaying data

    Date
    1986-01-01     9034
    1986-02-01     9596
    1986-03-01    10558
    1986-04-01     9002
    1986-05-01     9239
                   ...
    2012-08-01    14442
    2012-09-01    13422
    2012-10-01    13795
    2012-11-01    13352
    2012-12-01    12716
    Name: Total, Length: 324, dtype: int64
```

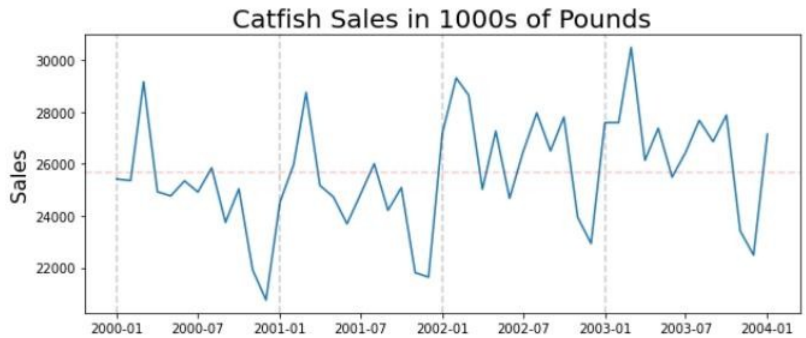
Now, we read our stationary data of catfish sales in years.

```
[ ] start_date = datetime(2000,1,1)
    end_date = datetime(2004,1,1)
    lim_catfish_sales = catfish_sales[start_date:end_date]
```

Here we limit our data from 2000 till 2004 becase we have huge amount of data so, we will perform our operations on this piece of data.

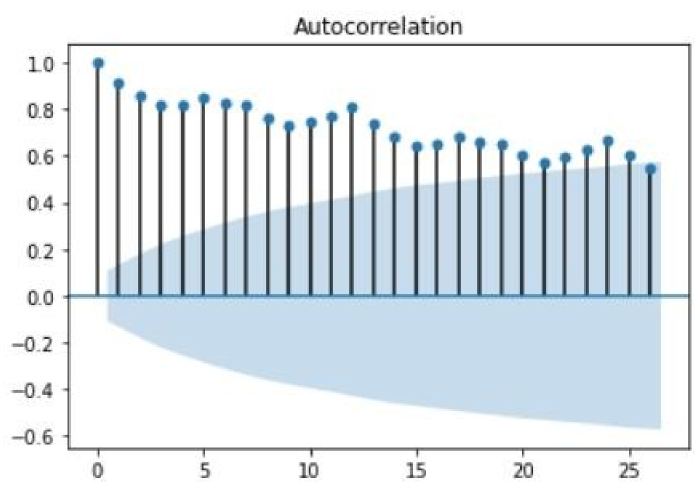
```
plt.figure(figsize=(10,4))
plt.plot(lim_catfish_sales)
plt.title('Catfish Sales in 1000s of Pounds', fontsize=20)
plt.ylabel('Sales', fontsize=16)
for year in range(start_date.year,end_date.year):
    plt.axvline(pd.to_datetime(str(year)+'-01-01'), color='k', linestyle='--', alpha=0.2)
plt.axhline(lim_catfish_sales.mean(), color='r', alpha=0.2, linestyle='--')
```

```
<matplotlib.lines.Line2D at 0x7fd5cc85cfd0>
```



Here graph of data is displayed. Red line indicates the mean of data.

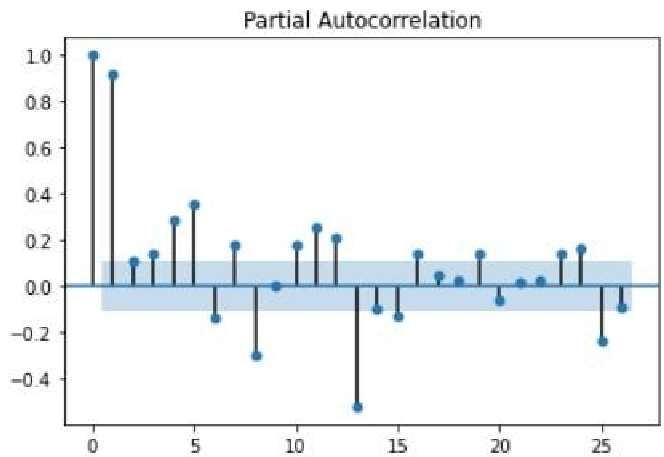
```
[ ] #ACF
    ACF_plot = plot_acf(catfish_sales)    #Based on ACF we should start with MA(1)
```



W

e find ACF of our data here which is 1.

```
pacf_plot = plot_pacf(catfish_sales)     #Based on PACF we should start with AR(4)
```



W

e have found PACF here which is 4.

```
[ ] #Training Data
    train_end = datetime(2003,7,1)
    test_end = datetime(2004,1,1)

    train_data = first_diff[:train_end]
    test_data = first_diff[train_end + timedelta(days=1):test_end]
```

```
[ ] # define model
    model = ARMA(train_data, order=(4,1))
```

```
[ ] #fit the model
    start = time()
    model_fit = model.fit()
    end = time()
    print('Model Fitting Time:', end - start)

    Model Fitting Time: 0.5325267314910889
```

First of all, we have trained our data from year 2003 till year 2004. After training we have applied ARMA model with order 4 of AR due to PACF and order 1 of MA due to ACF.

Next, we fit our data at end of training.

```
[ ] #summary of the model
    print(model_fit.summary())
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                  Total   No. Observations:                   42
Model:                     ARMA(4, 1)   Log Likelihood                -376.584
Method:                       css-mle   S.D. of innovations           1850.781
Date:                Mon, 18 Apr 2022   AIC                            767.167
Time:                        17:55:39   BIC                            779.331
Sample:                    02-01-2000   HQIC                           771.626
                         - 07-01-2003
===============================================================================
                  coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------
const          37.2955    129.751      0.287      0.775    -217.012     291.603
ar.L1.Total    -0.8666      0.185     -4.692      0.000      -1.229      -0.505
ar.L2.Total    -0.4236      0.166     -2.547      0.015      -0.750      -0.098
ar.L3.Total    -0.5584      0.156     -3.579      0.001      -0.864      -0.253
ar.L4.Total    -0.6144      0.126     -4.894      0.000      -0.861      -0.368
ma.L1.Total     0.5197      0.219      2.370      0.023       0.090       0.950
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            0.4929           -1.0592j            1.1683           -0.1807
AR.2            0.4929           +1.0592j            1.1683            0.1807
AR.3           -0.9473           -0.5431j            1.0920           -0.4171
AR.4           -0.9473           +0.5431j            1.0920            0.4171
MA.1           -1.9240           +0.0000j            1.9240            0.5000
```

This is summary of our data. ar.L1, ar.L2, ar.L3, ar.L4 indicates lags of our data which is negative and ma.L1 indicates Moving Average at Lag 1 which is positive. There is also more information about model, their orders, method etc.

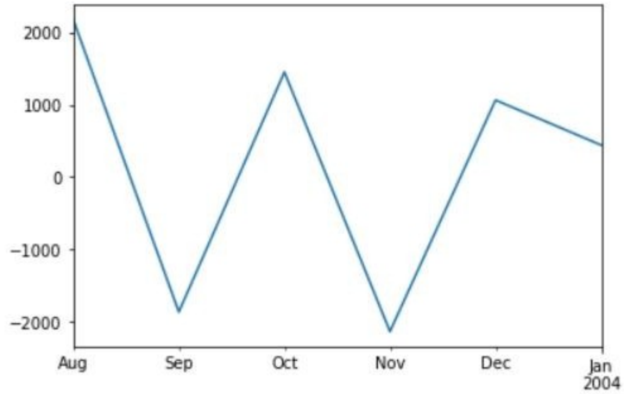
```
[ ] #Applying ARMA:
    #get prediction start and end dates
    pred_start_date = test_data.index[0]
    pred_end_date = test_data.index[-1]
```

Here we set our starting and ending dates according to which, our model has to predict future.

```
[ ] #get the predictions and residuals
    predictions = model_fit.predict(start=pred_start_date, end=pred_end_date)
    residuals = test_data - predictions
```

Here we get prediction of our data and also can get residual by subtracting test data with predicted data.

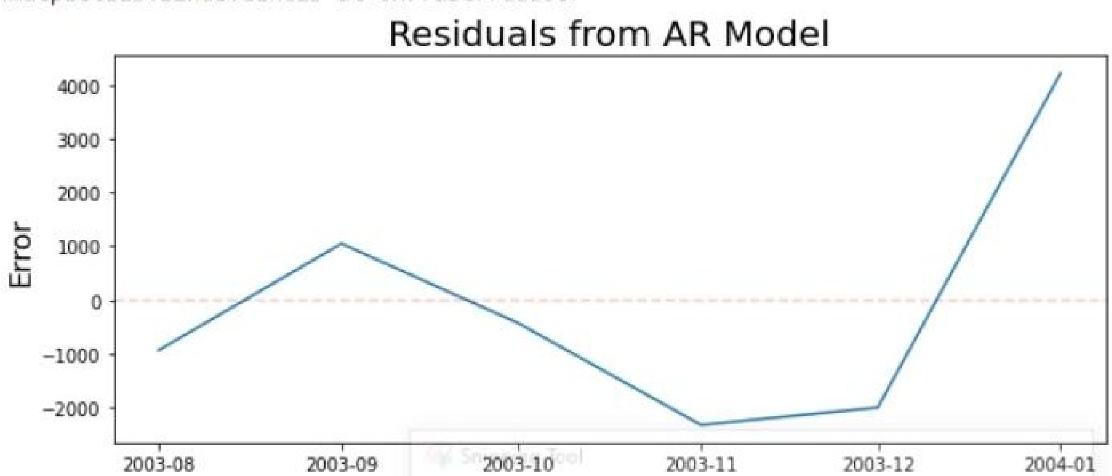
```
[ ] predictions.plot()
    plt.show()
```



This is the prediction of our model till year 2004, January.

```
[ ] plt.figure(figsize=(10,4))
    plt.plot(residuals)
    plt.title('Residuals from AR Model', fontsize=20)
    plt.ylabel('Error', fontsize=16)
    plt.axhline(0, color='r', linestyle='--', alpha=0.2)
```

```
<matplotlib.lines.Line2D at 0x7fd5c77eaa90>
```



Th

is is the residual of our model which we have calculated above.

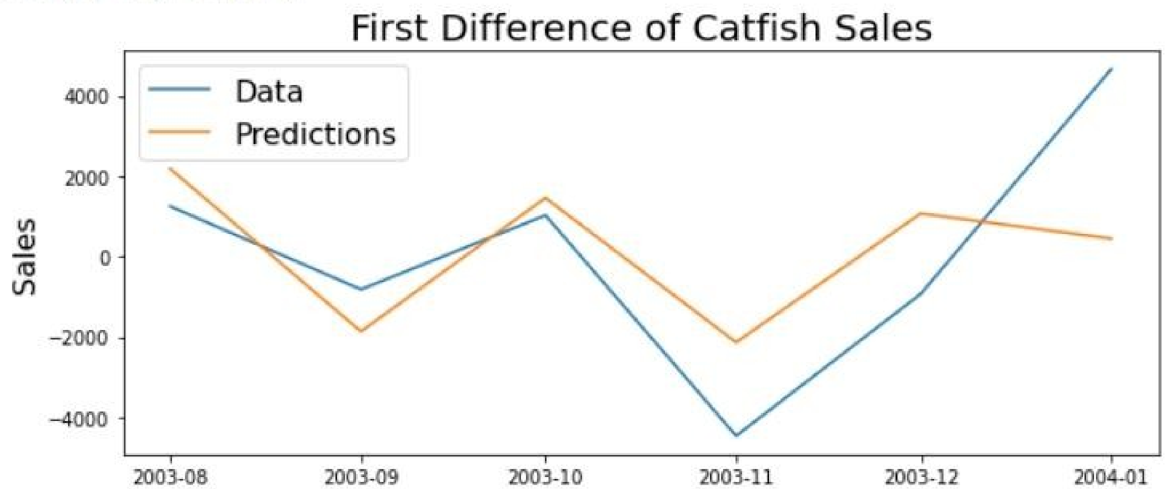
```
plt.figure(figsize=(10,4))

plt.plot(test_data)
plt.plot(predictions)

plt.legend(('Data', 'Predictions'), fontsize=16)

plt.title('First Difference of Catfish Sales', fontsize=20)
plt.ylabel('Sales', fontsize=16)
```

```
Text(0, 0.5, 'Sales')
```



Blue lines represent our actual data and orange lines represent predictions.We can easily see the difference between both data and check that how much our prediction is accurate. Root mean squared error according to residual is stated below.

```
print('Root Mean Squared Error:', np.sqrt(np.mean(residuals**2)))
```

```
Root Mean Squared Error: 2210.2690575778024
```

7. SARMA MODEL

The SARMA model deals with the ARMA model however in the context of seasonality. The python code for this was not implemented since there were not that many sources and the implementation was a bit more difficult than the rest.

The SARMA can also be written as:

$$(1 - \Phi B^{12})X_t = (1 + \Theta B^{12})Z_t,$$

Seasonal ARMA represents time series in terms of its historic values at lag which is equal to the length of the period and also incorporates the seasonality into the model.

If seasonal and non-seasonal operators are combined, a model is obtained as follows.

$$\Phi(B^h)\phi(B)X_t = \Theta(B^h)\theta(B)Z_t,$$

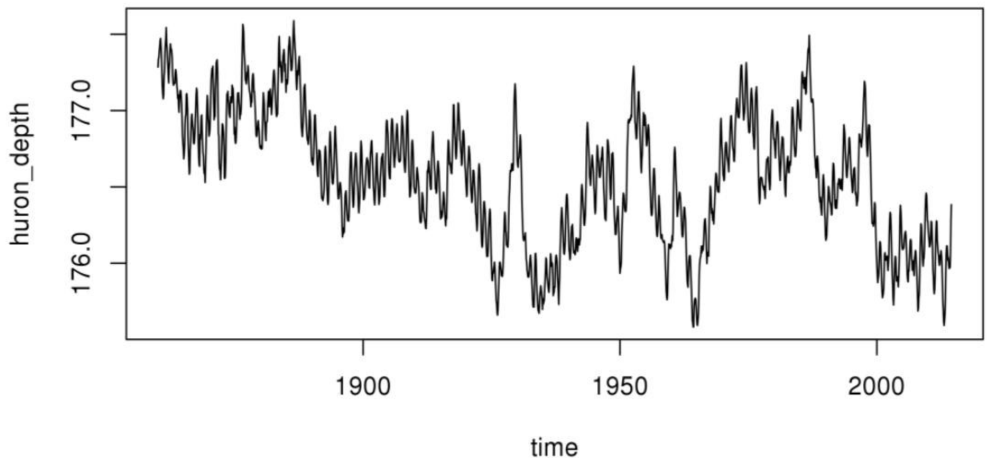
This model is known as the mixed SARMA and it denoted by.

$$ARMA(p, q) \times (P, Q)_h.$$

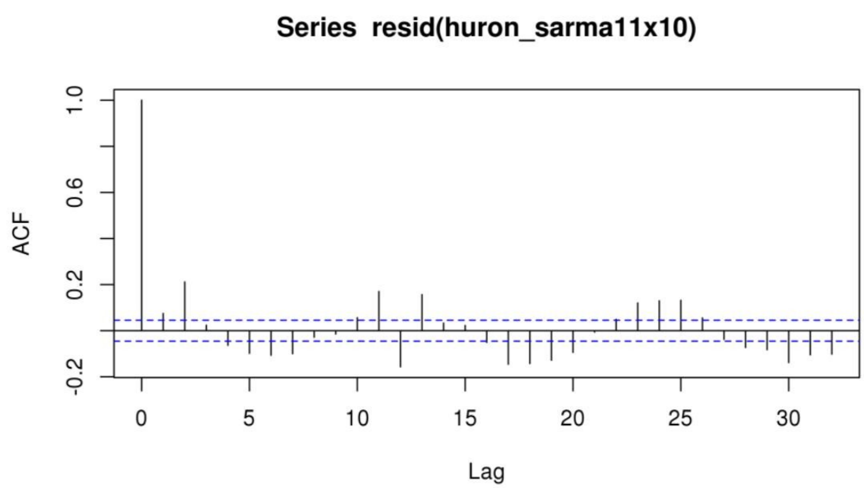
It is in SARMA Model where the AR and MA polynomials are factored into a monthly and annual polynomial . The annual polynomial may also be refered to as the the seasonal polynomial.

Thus, everything we learned about ARMA models also applies to SARMA.

example to better understand this model is taken from github and is explained as follows:



The above diagram represents reading the csv file with the data set and then fitting it into a model.



The diagram above represents the residual analysis after the mathematical calculations take place.