Modelling and Forecasting of Rainfall Time Series

Presented By-

Ayush Agarwal - 170187

Instructor - *Prof. Amit Mitra*

INTRODUCTION:

Time series analysis comprises methods for analyzing time series data in order to extract meaningful statistics and other characteristics of the data. Time series forecasting is the use of a model to predict future values based on previously observed values.

Time series are widely used for non-stationary data, like economic, weather, stock price, and retail sales in this post. We will demonstrate an approach for forecasting of rainfall time series.

About the data-

We analysed monthly rainfall data of states of India in mm.
https://www.kaggle.com/rajanand/rainfall-in-india

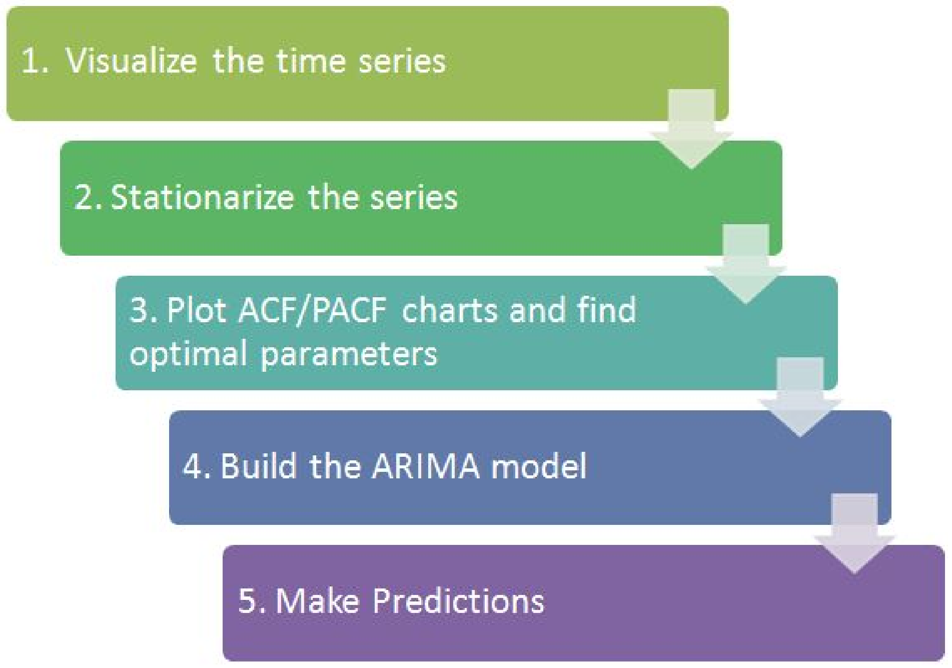
- The data contains- rainfall in mm of each month over 114 years (1901 - 2015) for various states of India.
- Each row specifies a state and a year- and each column specifies the rainfall in mm for each month.
- In this project we are using the data for Lakhshadweep.
- *All the intermediate missing values are interpolated.*

THE PROCESS:

We used Box Jenkins Algorithm for data analysis and forecasting.

It comprises of three steps:

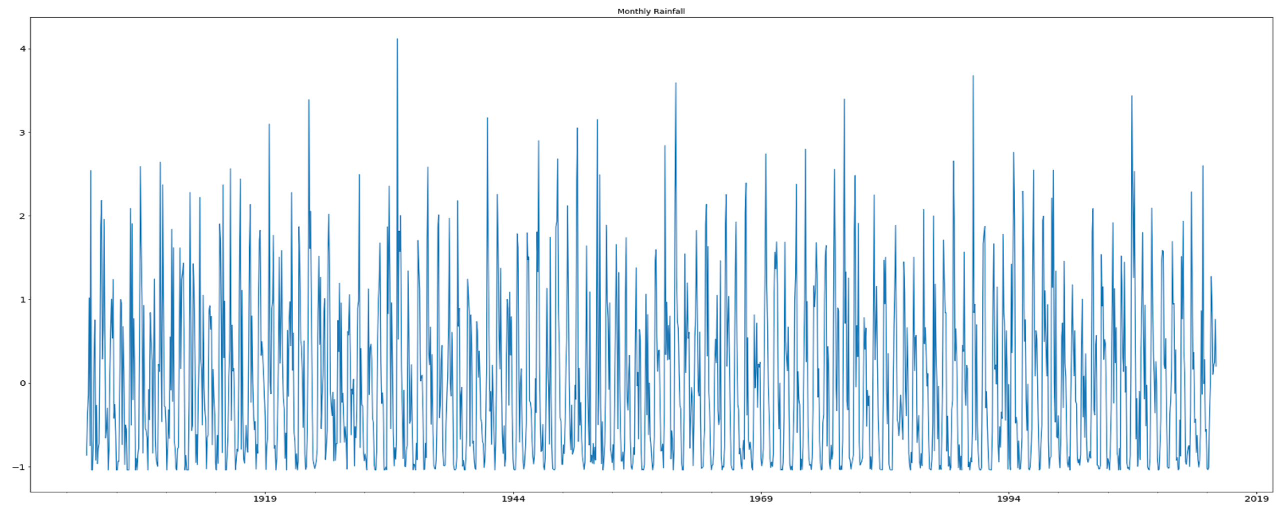
1. **Identification**. Use the data and all related information to help select a subclass of model that may best summarize the data.
   a. We will apply proper transformation until the data is stationarized.
   b. Plot the ACF and PACF of the above stationarized data for finding proper parameters of the time series process.
2. **Estimation**. Use the data to train the parameters of the model (i.e. the coefficients).
3. **Diagnostic Checking**. Evaluate the fitted model in the context of the available data and check for areas where the model may be improved.


1. Visualize the time series
2. Stationarize the series
3. Plot ACF/PACF charts and find optimal parameters
4. Build the ARIMA model
5. Make Predictions

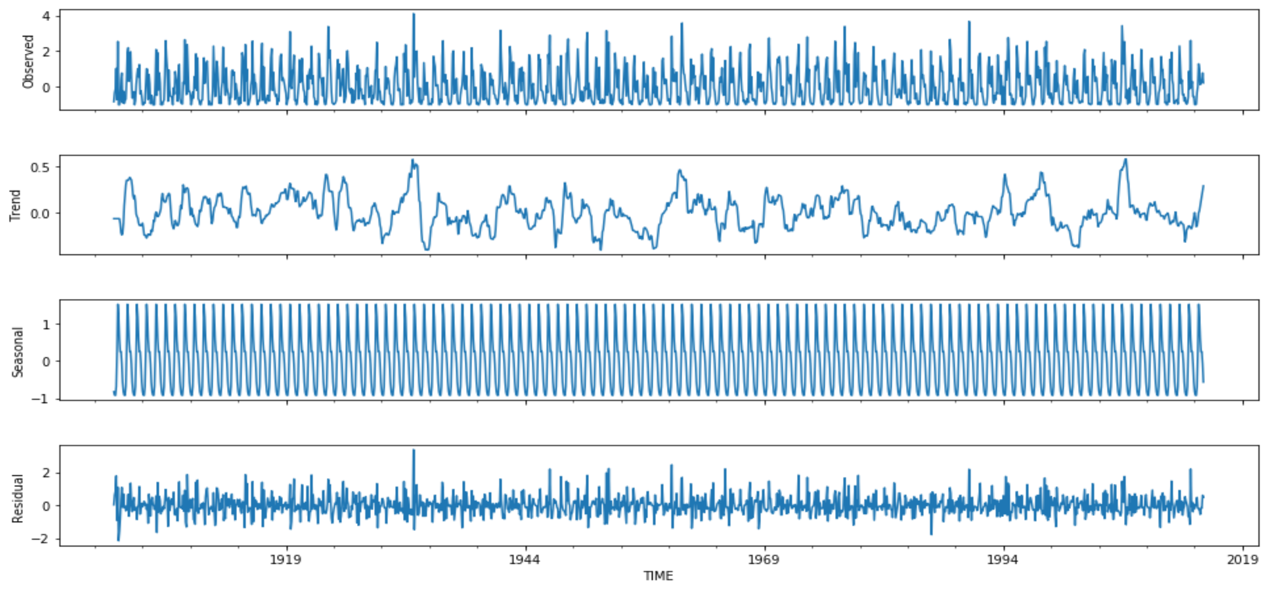
After modelling, we will perform residual analysis to check whether we have extracted all the patterns and trends from our time series data.

1. Visualize the data:

The first step is to visualize the data to understand what type of model we should use. We will check for the overall trend in our data. Also, look for any seasonal trends. This is important for deciding which type of model to use. We have used the predefined library functions to decompose the time series into trend, seasonal and residual components.



Monthly rainfall(normal data)



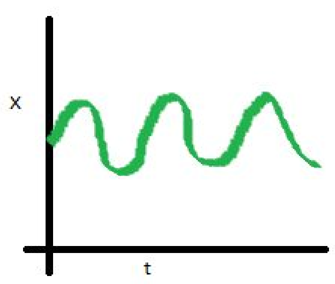
Decomposition of data

2. Stationarize the data:

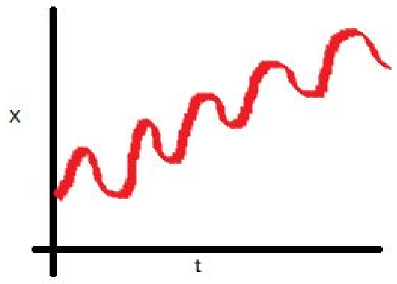
Definition-- $\{X_t\}$ is said to be a stationary process if for every n, and every admissible $t_1, t_2, \ldots t_n$ and any integer k, the joint distribution of $\{X_{t1}, X_{t2}, \ldots, X_{tn}\}$ is identical to the joint distribution of $\{X_{t1+k}, X_{t2+k}, \ldots, X_{tn+k}\}$.

Now, we will try to visualize what is meant by stationary time series process

1. The mean of the series should not be a function of time. The red graph below is not stationary because the mean increases over time.

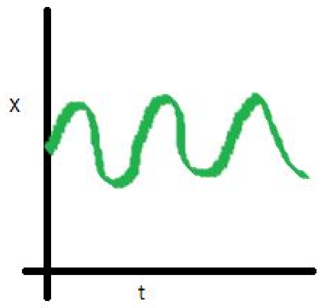


Stationary series

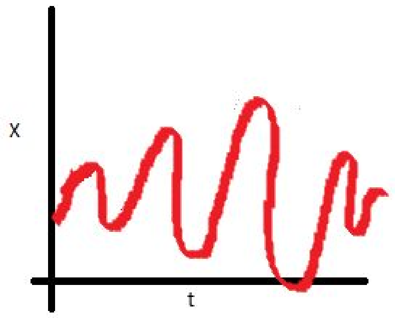


Non-Stationary series

2. The variance of the series should not be a function of time. This property is known as homoscedasticity. Notice in the red graph the varying spread of data over time.

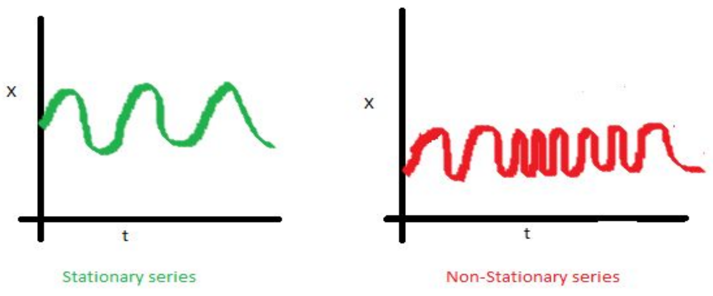


Stationary series



Non-Stationary series

3. Finally, the covariance of the i th term and the (i + m) th term should not be a function of time. In the following graph, you will notice the spread becomes closer as the time increases. Hence, the covariance is not constant with time for the 'red series'.



Now a very natural question is that, why do we need a stationary time series? The reason being, when we run a linear regression the assumption is that all of the observations are all independent of each other. In a time series, however, we know that observations are time dependent. It turns out that a lot of nice results that hold for independent random variables (law of large numbers and central limit theorem to name a couple) hold for stationary random variables. So by making the data stationary, we can actually apply regression techniques to this time dependent variable.

There are two ways you can check the stationarity of a time series. The first is by looking at the data. By visualizing the data it should be easy to identify a changing mean or variation in the data. For a more accurate assessment there is the **Dickey-Fuller test.**

Dickey Fuller Test

In statistics, the Dickey–Fuller test ,tests the null hypothesis that a unit root is present in an autoregressive model. The alternative hypothesis is different depending on which version of the test is used, but is usually stationarity or trend-stationarity.

Hence our null hypothesis H0 is not stationary against our alt. hypothesis H1 which is data stationary .

The Dickey-Fuller test is testing if $\phi=1$ in this model of the data:

$$y_t=\alpha+\beta t+\phi y_{t-1}+e_t$$

which is written as:

$$\Delta y_t=y_t-y_{t-1}=\alpha+\beta t + \gamma y_{t-1}+e_t$$

Where y_t is your data.It is written this way so we can do a linear regression of Δy_t against t and y_{t-1} and test if γ is different from 0. If $\gamma=0$, then we have a random walk process. If not and $-1<1+\gamma<1$, then we have a stationary process.

We applied the test to check whether our hypothesis is true or not, we look for a p-value in the test, and if the p-value is less than a specific significant level often 0.05 or 0.01, we reject our null hypothesis and thus making our time series stationary.

So now we need to transform the data to make it more stationary. There are various transformations you can do to stationarize the data.

1. **Deflation** - Merely applies a constant discount factor to the previous data. It usually helps in stabilizing variance.

2. **Logarithmic** - Converts multiplicative patterns to additive patterns and/or linearises exponential growth. Converts absolute change to percentage changes. Often stabilizes the variance of data with compound growth, regardless of whether deflation is also used.

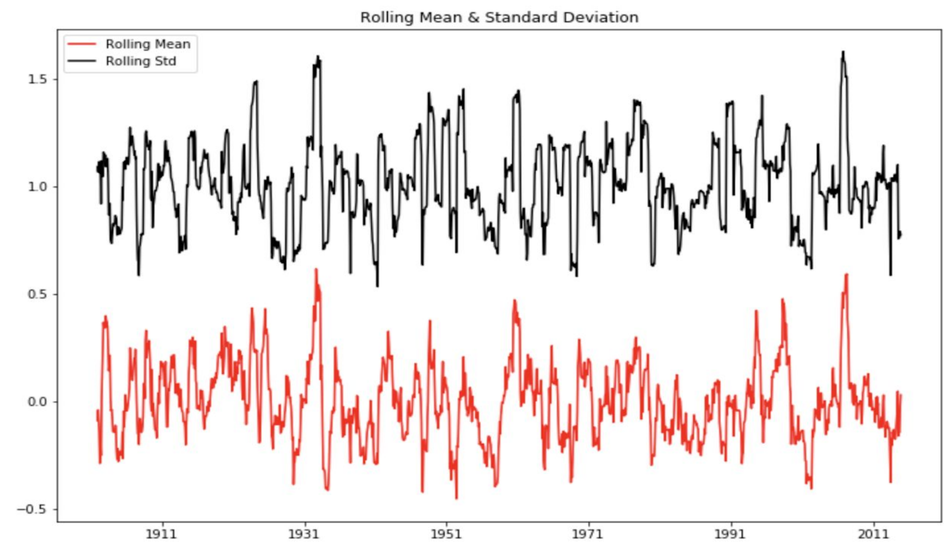
3. **First Difference** - Converts "levels" to "changes". (X_t -X_{t-1})

4. **Seasonal Difference** - Convert "levels" to "seasonal changes".(X_t -X_{t-s})

5. **Seasonal Adjustment -** Remove a constant seasonal pattern from a series (either multiplicative or additive).

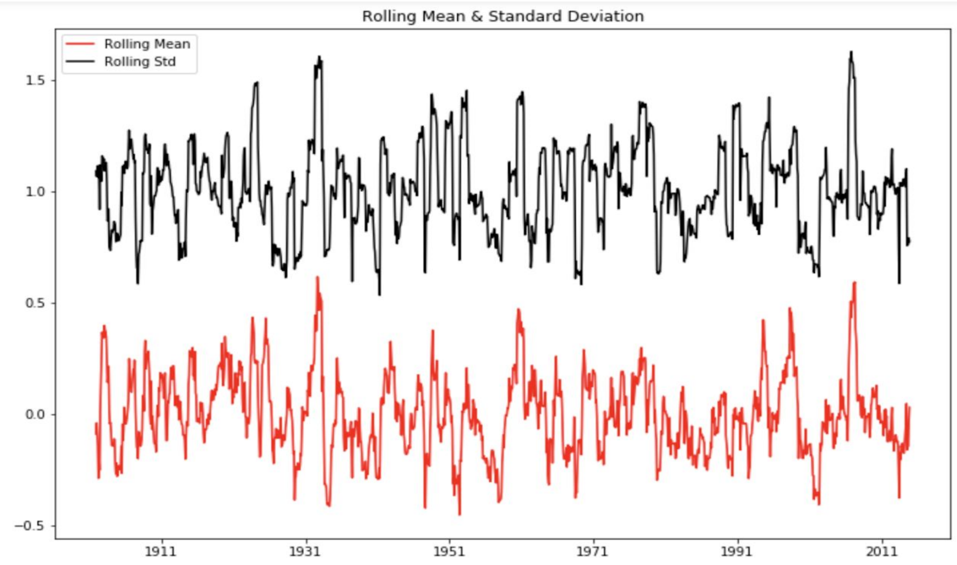
Now we will apply various transformations recursively until we obtain a stationary time series according to the Dickey-Fuller test.

In this project, we are just going to use First Differencing and seasonal Differencing.



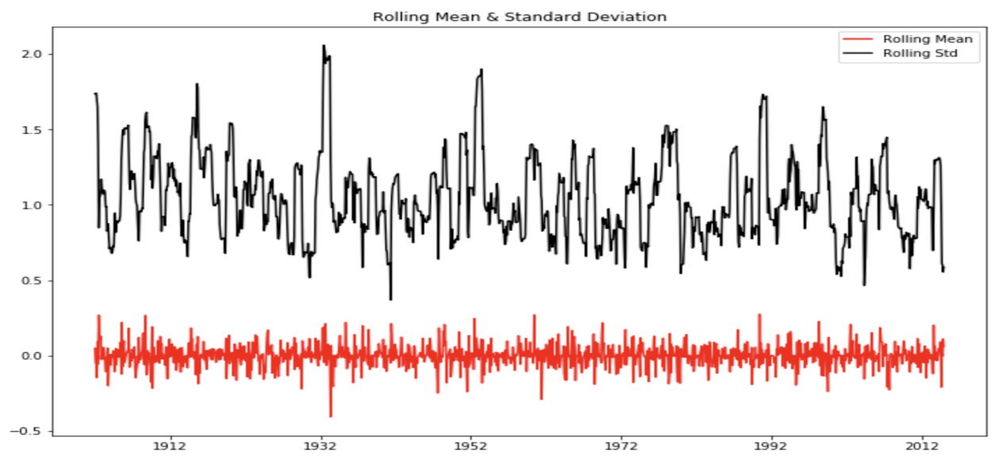
```
Results of Dickey-Fuller Test:
Test Statistic                 -6.358154e+00
p-value                         2.512770e-08
#Lags Used                      2.300000e+01
Number of Observations Used     1.344000e+03
Critical Value (1%)            -3.435225e+00
Critical Value (5%)            -2.863693e+00
Critical Value (10%)           -2.567916e+00
dtype: float64
```

Dickey fuller for normal data(AIC)



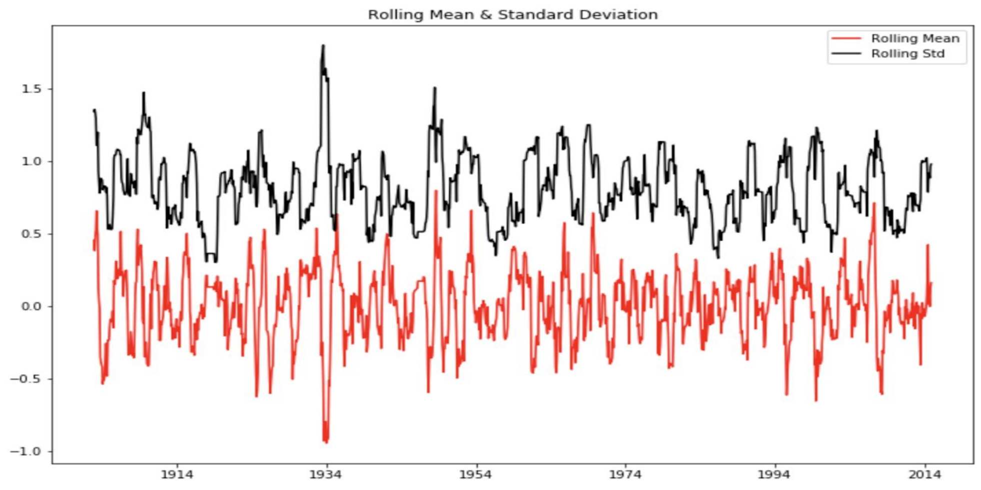
```
Results of Dickey-Fuller Test:
Test Statistic                 -6.358154e+00
p-value                         2.512770e-08
#Lags Used                      2.300000e+01
Number of Observations Used     1.344000e+03
Critical Value (1%)            -3.435225e+00
Critical Value (5%)            -2.863693e+00
Critical Value (10%)           -2.567916e+00
dtype: float64
```

Dickey fuller for normal data(BIC)



```
Results of Dickey-Fuller Test:
Test Statistic                 -1.547800e+01
p-value                         2.578198e-28
#Lags Used                      2.300000e+01
Number of Observations Used     1.343000e+03
Critical Value (1%)            -3.435229e+00
Critical Value (5%)            -2.863694e+00
Critical Value (10%)           -2.567917e+00
dtype: float64
```

Dickey fuller for first differenced data(AIC)



```
Results of Dickey-Fuller Test:
Test Statistic                 -1.190225e+01
p-value                         5.539866e-22
#Lags Used                      2.400000e+01
Number of Observations Used     1.331000e+03
Critical Value (1%)            -3.435273e+00
Critical Value (5%)            -2.863714e+00
Critical Value (10%)           -2.567927e+00
dtype: float64
```

Dickey fuller for seasonal differenced data(AIC)

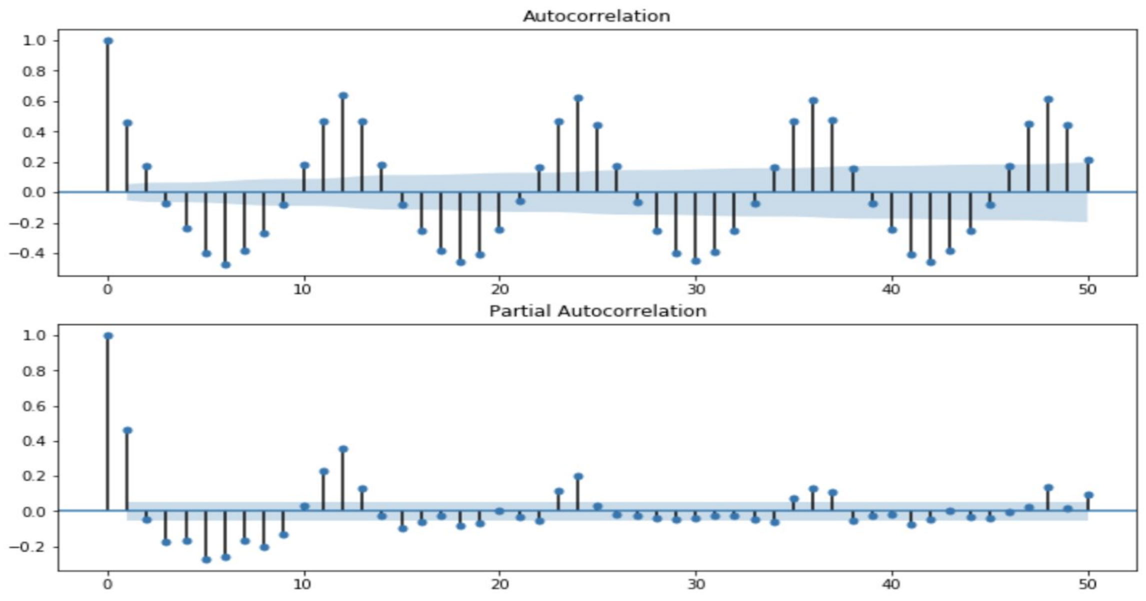
After performing D-F test we saw that our original data was already stationary and hence there was no need to apply any transformations-- as this would only increase the variance of our time series data which would hamper our model. The differencing was shown only for completion of our analysis

3. Plot the ACF and PACF charts and find the optimal parameters

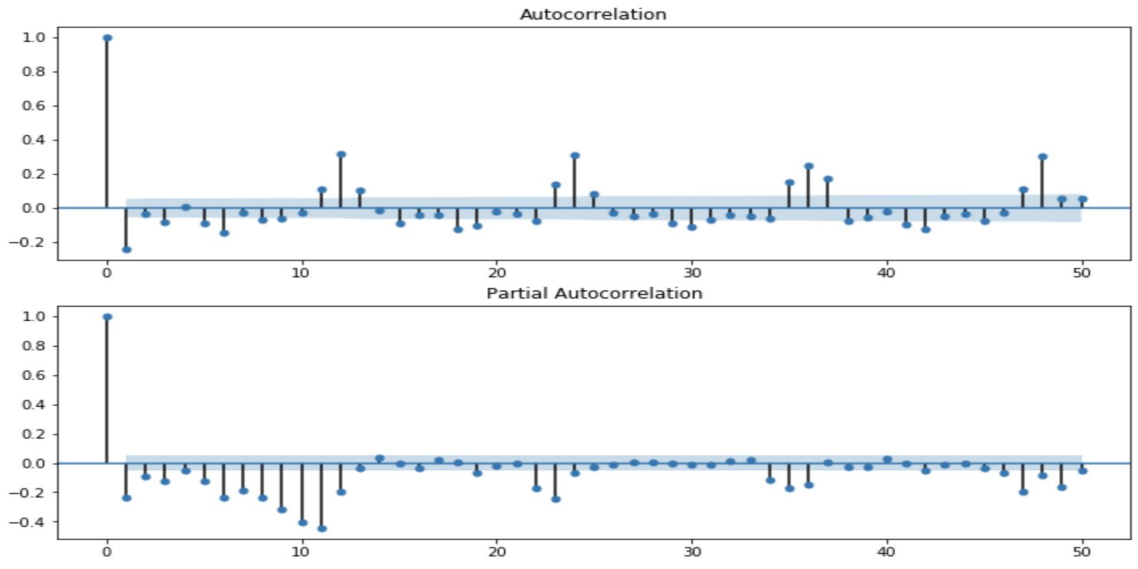
- **Autocorrelation Function (ACF)**. The plot summarizes the correlation of an observation with lag values. The x-axis shows the lag and the y-axis shows the correlation coefficient between -1 and 1 for negative and positive correlation.
- **Partial Autocorrelation Function (PACF)**. The plot summarizes the correlations for an observation with lag values that is not accounted for by prior lagged observations.

Some useful patterns you may observe on these plots are:

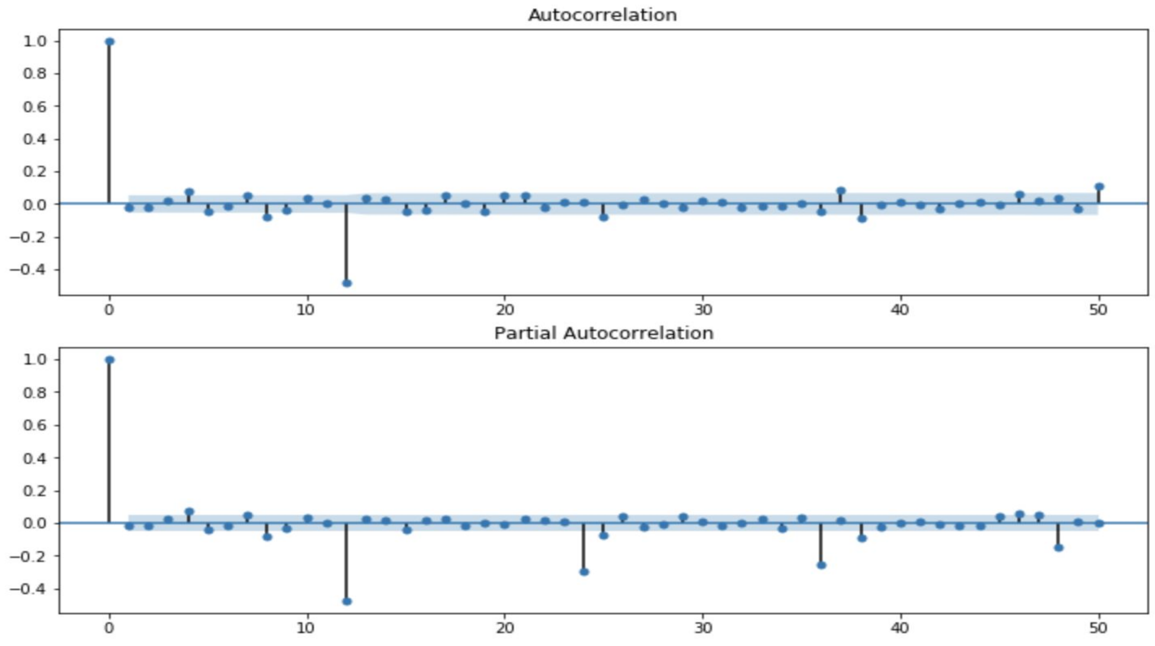
- The model is AR if the ACF trails off after a lag and has a hard cut-off in the PACF after a lag. This lag is taken as the value for *p*.
- The model is MA if the PACF trails off after a lag and has a hard cut-off in the ACF after the lag. This lag value is taken as the value for *q*.
- The model is a mix of AR and MA if both the ACF and PACF tail off.



ACF and PACF of normal data



ACF and PACF of first differenced data



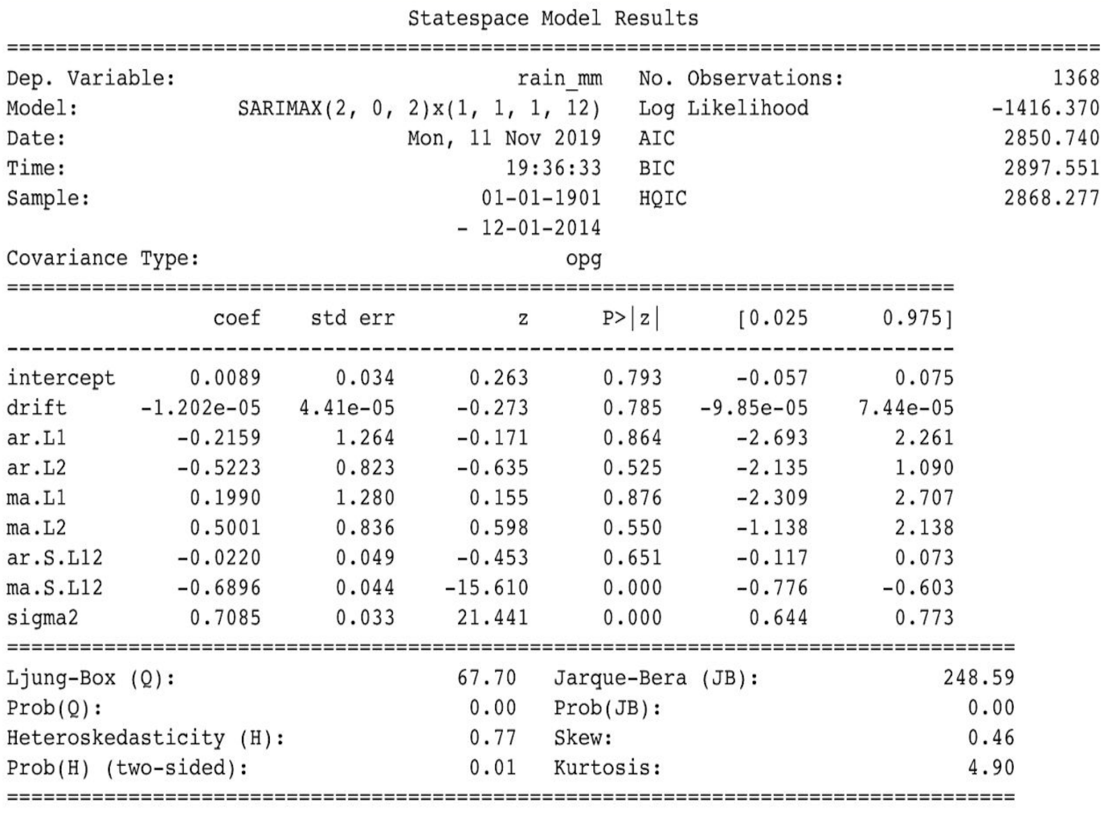
ACF and PACF of seasonal differenced data

Here we can observe a slow decay of the ACF and PACF at multiple lags of 12, which are significant. This supports the earlier assertion of seasonality in the data, hence, the need for seasonal differencing with period of 12. The seasonally differenced data is stationary as above.

From Figure, a spike at 12 in the ACF is significant but no other is significant at lags multiple of 12, the pacf shows an exponential decay in the seasonal lags; that is 12, 24, 36 etc. Thus, the seasonal part of the model has a moving average term of order 1 and an autoregressive term of 1. For the non-seasonal part, the ACF tails off after lag 2 and the PACF cuts off after lag 2. Therefore, the non-seasonal part has an autoregressive term of 2 and a moving average term of 2. Based on the features portrayed by the plots, an initial SARIMA (2,0,2)×(1,1,1,12) model is proposed.

First we tried to model a SARIMA (2,0,2)×(1,0,1,12) model which should be our natural choice, but using SARIMA (2,0,2)×(1,1,1,12) model gave better results. Hence we kept the latter

4. Build Model:



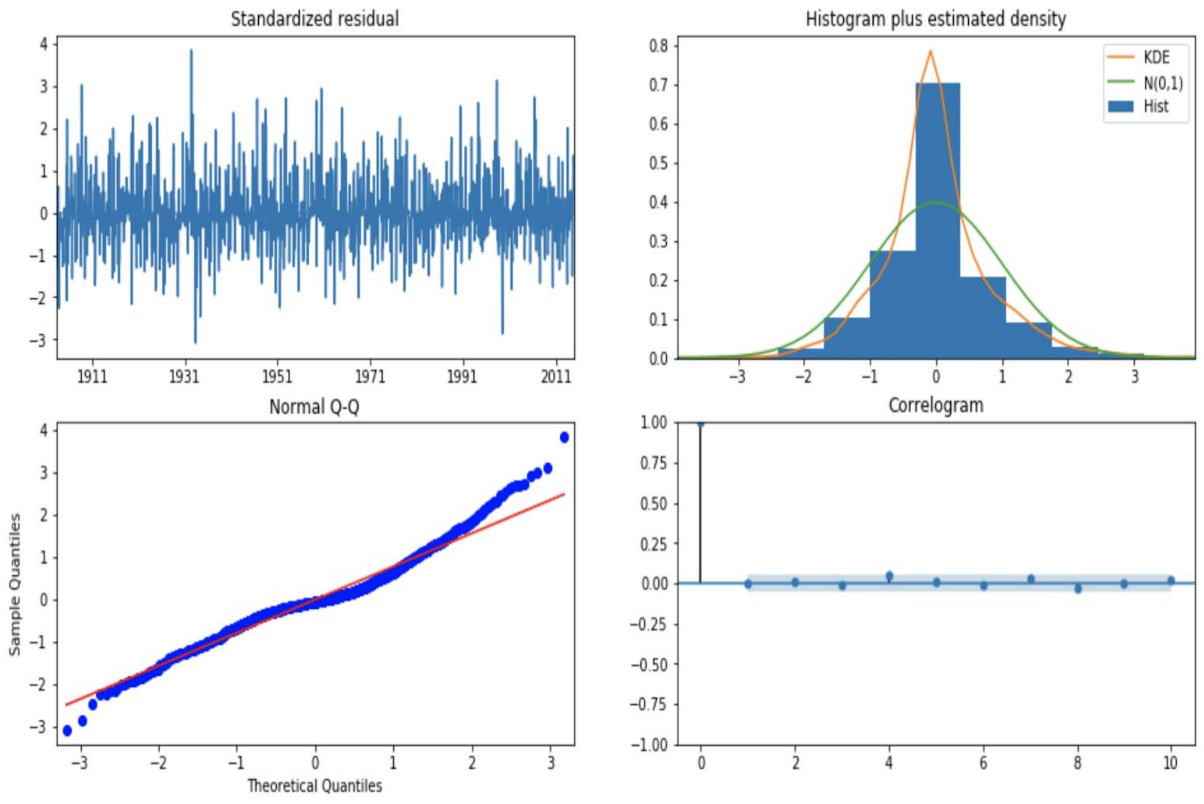
```
                                     Statespace Model Results
==========================================================================================
Dep. Variable:                                  rain_mm   No. Observations:           1368
Model:             SARIMAX(2, 0, 2)x(1, 1, 1, 12)   Log Likelihood           -1416.370
Date:                              Mon, 11 Nov 2019   AIC                       2850.740
Time:                                      19:36:33   BIC                       2897.551
Sample:                                  01-01-1901   HQIC                      2868.277
                                       - 12-01-2014
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0089      0.034      0.263      0.793      -0.057       0.075
drift      -1.202e-05   4.41e-05     -0.273      0.785   -9.85e-05    7.44e-05
ar.L1         -0.2159      1.264     -0.171      0.864      -2.693       2.261
ar.L2         -0.5223      0.823     -0.635      0.525      -2.135       1.090
ma.L1          0.1990      1.280      0.155      0.876      -2.309       2.707
ma.L2          0.5001      0.836      0.598      0.550      -1.138       2.138
ar.S.L12      -0.0220      0.049     -0.453      0.651      -0.117       0.073
ma.S.L12      -0.6896      0.044    -15.610      0.000      -0.776      -0.603
sigma2         0.7085      0.033     21.441      0.000       0.644       0.773
===================================================================================
Ljung-Box (Q):                       67.70   Jarque-Bera (JB):               248.59
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.77   Skew:                             0.46
Prob(H) (two-sided):                  0.01   Kurtosis:                         4.90
===================================================================================
```

We used statsmodel library to build the model ,

We took a set of integers and tried for different values of p,q and i and found that for the above shown process, the AIC and BIC values came out to be small and hence it is the required model which came out to be same as we anticipated using ACF and PACF plots.

The above table also displays the values of parameters.

Residual Analysis :



- **Histogram** of residuals are used to show whether variance is normally
- distributed or not, a symmetric well shaped histogram which is evenly distributed around 0, indicates that the normality assumption about the residuals is likely to be true. If the histogram indicates that random error is not normally distributed then it suggests that the model underlying assumptions may have been violated.

- **Normal Q-Q** , or quantile-quantile plot is a graphical tool to help us assess if a set of data plausibly came from some theoretical distribution such as a Normal or exponential. A Q-Q plot is a scatterplot created by plotting two sets of quantiles against one another. If both sets of quantiles came from the same distribution, we should see the points forming a line that's roughly straight. We can observe that for our Q-Q plots both the quantiles are overlapping

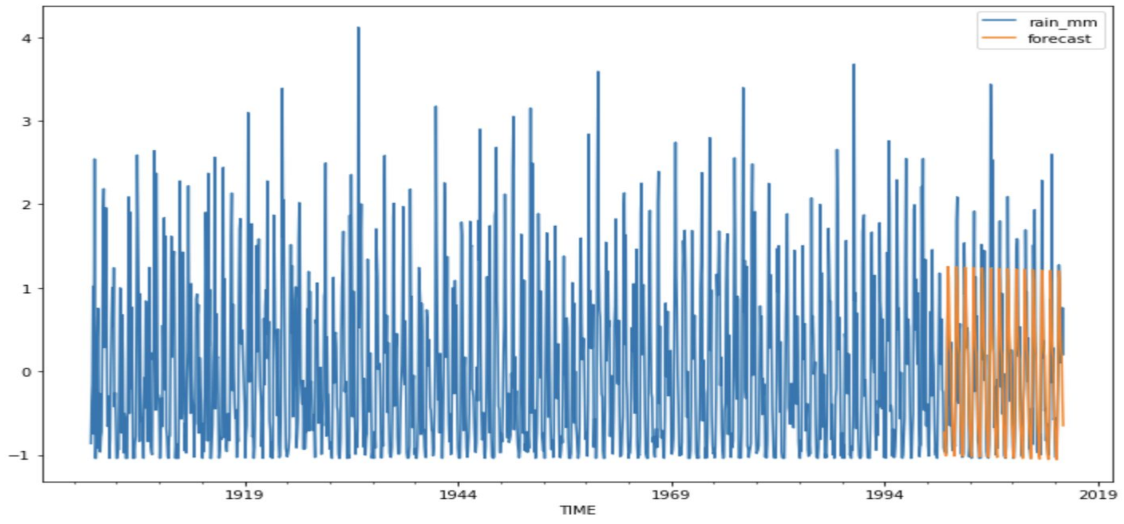
- In the **standardized residual plot** we can see that the residuals appear to be just fluctuating about x-axis without any kind of trend or pattern and hence showing the randomness of this residuals.

- A correlogram (also called Auto Correlation Function ACF Plot or Autocorrelation plot) is a visual way to show serial correlation in data that changes over time (i.e. time series data). Serial correlation (also called autocorrelation) is where an error at one point in time travels to a subsequent point in time. Correlograms can give you a good idea of whether or not pairs of data show autocorrelation. We can observe that our residual data follows a plot similar to WN sequence.

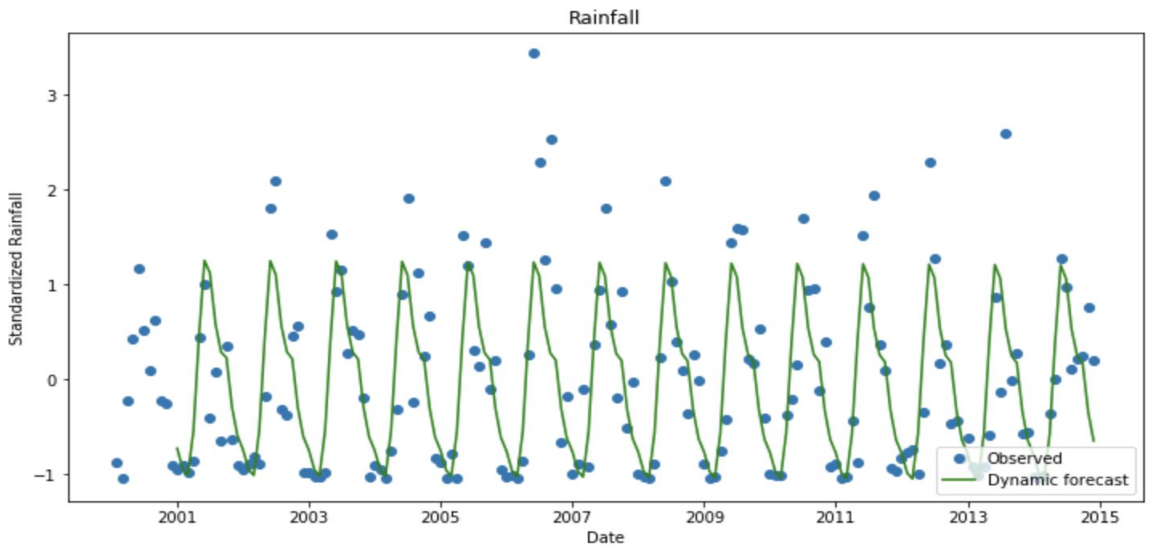
Hence based on these plots, we can conclude that there are no further trend that can be extracted from these residuals and that it follows a distribution similar to a normal WN process.

5. Make Predictions:

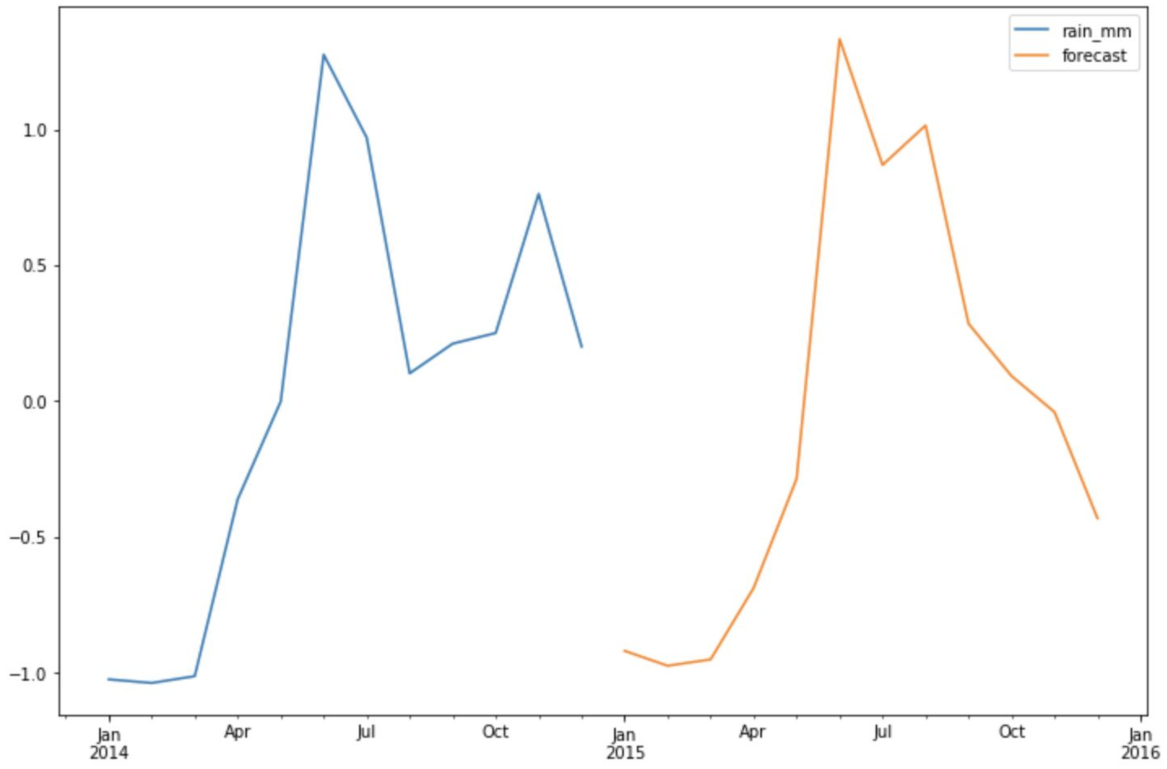
Now that we have a model built, we want to use it to make forecasts. First we use the model to forecast for time periods that we already have data for, so we can understand how accurate are the forecasts.



The magnified image of the above prediction is-



Now we will use the predict function to create forecast values for these newly added time periods and plot them.



```
# coding: utf-8

# In[ ]:



get_ipython().run_line_magic('matplotlib', 'inline')
import math
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import datetime
from dateutil.relativedelta import relativedelta
import seaborn as sns
import statsmodels.api as sm
from statsmodels.tsa.stattools import acf
from statsmodels.tsa.stattools import pacf
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller


# In[255]:



df =
pd.read_csv('/Users/harshitgarg/Downloads/rainfall.csv'
, index_col=0)
df1 = df.loc['LAKSHADWEEP'] #Selecting a particular
state for rainfall characteristics
df1 = df1[['YEAR', 'JAN', 'FEB', 'MAR', 'APR', 'MAY',
'JUN', 'JUL', 'AUG', 'SEP',
       'OCT', 'NOV', 'DEC']]
df1.index.name=None
df1.reset_index(inplace=True)

df1 = df1.interpolate(method ='linear', limit_direction
='backward', limit = 100) #interpolating NaN values

start = datetime.datetime.strptime("1901-01-01",
"%Y-%m-%d")
date_list = [start + relativedelta(months=x) for x in
range(0,114*12)]  #indexing by month+year
df2 = pd.DataFrame(date_list,columns = ['TIME'])

zeroes = [0.00]*(114*12)
df1 = df1[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN',
'JUL', 'AUG',
       'SEP', 'OCT', 'NOV', 'DEC']]
df2['rain_mm'] = zeroes

k = 0
for i in range(114):
    for j in list(df1.iloc[i]):
        df2.at[k,'rain_mm'] = j
        k = k + 1
df2.set_index(['TIME'], inplace=True)
a = math.sqrt(np.var(df2['rain_mm']))
b = np.mean(df2['rain_mm'])
df2['rain_mm'] = (df2['rain_mm'] - b)/a  #standardizing
the data
```

```
# In[268]:


df2.rain_mm.plot(figsize=(32,16), title= 'Monthly
Rainfall', fontsize=14)
plt.savefig('month_ridership.png', bbox_inches='tight')
#Plot data


# In[257]:


decomposition =
seasonal_decompose(df2.rain_mm,freq=12,extrapolate_tren
d = 1)  #decompose using additive method
fig = plt.figure()
fig = decomposition.plot()
fig.set_size_inches(15, 8)


# In[258]:


trend = decomposition.trend
seasonal = decomposition.seasonal
print(trend)


# In[259]:
```

```
def test_stationarity(timeseries,lag):

    #Determing rolling statistics
    #pd.rolling_mean(timeseries, window=12)
    rolmean = timeseries.rolling(12).mean()   #moving
average
    rolstd = timeseries.rolling(12).std()

    #Plot rolling statistics:
    fig = plt.figure(figsize=(12, 8))
    mean = plt.plot(rolmean, color='red',
label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label =
'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show()

    #Perform Dickey-Fuller test:
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag = lag)
    dfoutput = pd.Series(dftest[0:4], index=['Test
Statistic','p-value','#Lags Used','Number of
Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
test_stationarity(df2.rain_mm,'AIC')
test_stationarity(df2.rain_mm,'BIC')


# In[260]:


#Though we have achieved stationarity we still show
application of first difference and seasonal difference
#first difference
df2['first_difference'] = df2.rain_mm -
df2.rain_mm.shift(1)
test_stationarity(df2.first_difference.dropna(inplace=F
alse),'AIC')


# In[261]:


#seasonal difference
df2['seasonal_difference'] = df2.rain_mm -
df2.rain_mm.shift(12)
test_stationarity(df2.seasonal_difference.dropna(inplac
e=False),'AIC')
print(df2)


# In[262]:


fig = plt.figure(figsize=(12,8))
```

```
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(df2.rain_mm.iloc[:],
lags=50, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(df2.rain_mm.iloc[:],
lags=50, ax=ax2)

fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig =
sm.graphics.tsa.plot_acf(df2.first_difference.iloc[1:],
lags=50, ax=ax1)
ax2 = fig.add_subplot(212)
fig =
sm.graphics.tsa.plot_pacf(df2.first_difference.iloc[1:]
, lags=50, ax=ax2)

fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig =
sm.graphics.tsa.plot_acf(df2.seasonal_difference.iloc[1
3:], lags=50, ax=ax1)
ax2 = fig.add_subplot(212)
fig =
sm.graphics.tsa.plot_pacf(df2.seasonal_difference.iloc[
13:], lags=50, ax=ax2)



# In[263]:
```

```
mod = sm.tsa.statespace.SARIMAX(df2.rain_mm,
trend='ct', order=(2,0,2),
seasonal_order=(1,1,1,12),enforce_invertibility=False,e
nforce_stationarity=False)
results = mod.fit()
print(results.summary()) #model fit


# In[264]:


df2['forecast'] = results.predict(start = 1200, end=
1368, dynamic= True)  #comparing prediction with actual
df2[['rain_mm', 'forecast']].plot(figsize=(12, 8))
plt.savefig('ts_df_predict.png', bbox_inches='tight')


# In[265]:


npredict = 168  #data zoomed
fig, ax = plt.subplots(figsize=(12,6))
npre = 12
ax.set(title='Rainfall', xlabel='Date',
ylabel='Standardized Rainfall')
ax.plot(df2.index[-npredict-npre+1:],
df2.ix[-npredict-npre+1:, 'rain_mm'], 'o',
label='Observed')
ax.plot(df2.index[-npredict-npre+1:],
df2.ix[-npredict-npre+1:, 'forecast'], 'g',
label='Dynamic forecast')
legend = ax.legend(loc='lower right')
legend.get_frame().set_facecolor('w')
plt.savefig('ts_predict_compare.png',
bbox_inches='tight')


# In[266]:


start = datetime.datetime.strptime("2015-01-01",
"%Y-%m-%d")  #adding new data for predicition
date_list = [start + relativedelta(months=x) for x in
range(0,12)]
future = pd.DataFrame(index=date_list, columns=
df.columns)
df2 = pd.concat([df2, future])


# In[267]:


df2['forecast'] = results.predict(start = 1368, end =
1379, dynamic= True)  #forecasting for new data
df2[['rain_mm', 'forecast']].ix[-24:].plot(figsize=(12,
8))
plt.savefig('ts_predict_future.png',
bbox_inches='tight')
```

----------x---------------------x----------------x-----------------x-----------