**Time Series**

**Introduction**

* Crime is a well-known social problem and it is unpredictable. Increase rate of crime affects our society in different ways. It disrupts not only the normal life but also the socio economic developments of society. Therefore, it becomes necessary to analyse crime data to inform the law enforcement agencies about specific and general trends and patterns of crime so that the department can best utilize the scant human and material resources to combat the situation.
* Traditional policing methodologies to prevent and detect criminal elements has not found to significantly deter or preempt the rise of crime in a region. The crime solving agencies can do a better work if they have a good idea of trend and pattern of criminal activities that are happening in a particular area. This can be done by using machine learning by employing different algorithms to find the patterns of the criminal activities.
* Due to rapid development in computerization and digital technique, a large amount of data related with crime are available with law enforcement agencies. Data mining techniques can play an important role to analyse the data and discover knowledge from them. Data mining can help in classification, clustering, evaluation, prediction and trend analysis. Knowledge gained from Data analysis can help law enforcement agencies to predict present crime. In order to understand, analyse huge amount of data, a multidisciplinary approach, data mining is used. Data mining techniques can play an important role to analyse data and discover knowledge from them.
* We analyze Crime to inform law enforcers about general and specific crime trends, patterns, and series in an ongoing, timely manner. To take advantage of the abundance of information existing in law enforcement agencies, the criminal justice system, and public domain. To maximize the use of limited law enforcement resources.

**Crime Analysis**

* Machine learning methods have become important for detection of crime and its prevention. In this presentation, various machine learning methods have been described briefly and also the purpose for which they can be used for crime analysis and prediction.
* Crime Data of Chhattisgarh has been taken and its prediction has been done for next two years. The scope of this presentation is to know effective the machine learning algorithms used in data mining analysis can be at predicting crime patterns.
* This is expected to help the police plan for decision making for deployment of available resources and effectively contribute to build a smarter city.

**Methodology**

* Data collection
* Time series analysis
* Data analysis: Applying algorithm on the crime data and obtain the pattern of data, building statistical models and statistical analysis
* Action: Forecast the data, building machine learning models and decision making.

**Data description**

* analysis of time series in Python by taking an example of crime in Chhattisgarh from date 2015.01.01 to 2019.12.01 and forecast for 2020 and 2021
* Steps involved in Time series analysis:
* Importation of data an plotting
* Check the stationarity /seasonality/trend of data
* Differencing
* Decomposition
* Forecasting

**Limitation of TA analysis**

* Time series model consists of four components: Trend, seasonality, cyclic and Random variation. While prediction of the above data, Random variation which are caused by Natural disaster(s) and strikes/ lockouts have not been taken into account. Since we are going through irregular variation in the form of ‘Pandemics’, the forecast data will be different from the actual data of 2020 and 2021 because of not factoring irregular variation in crime data due to ‘Pandemics’.

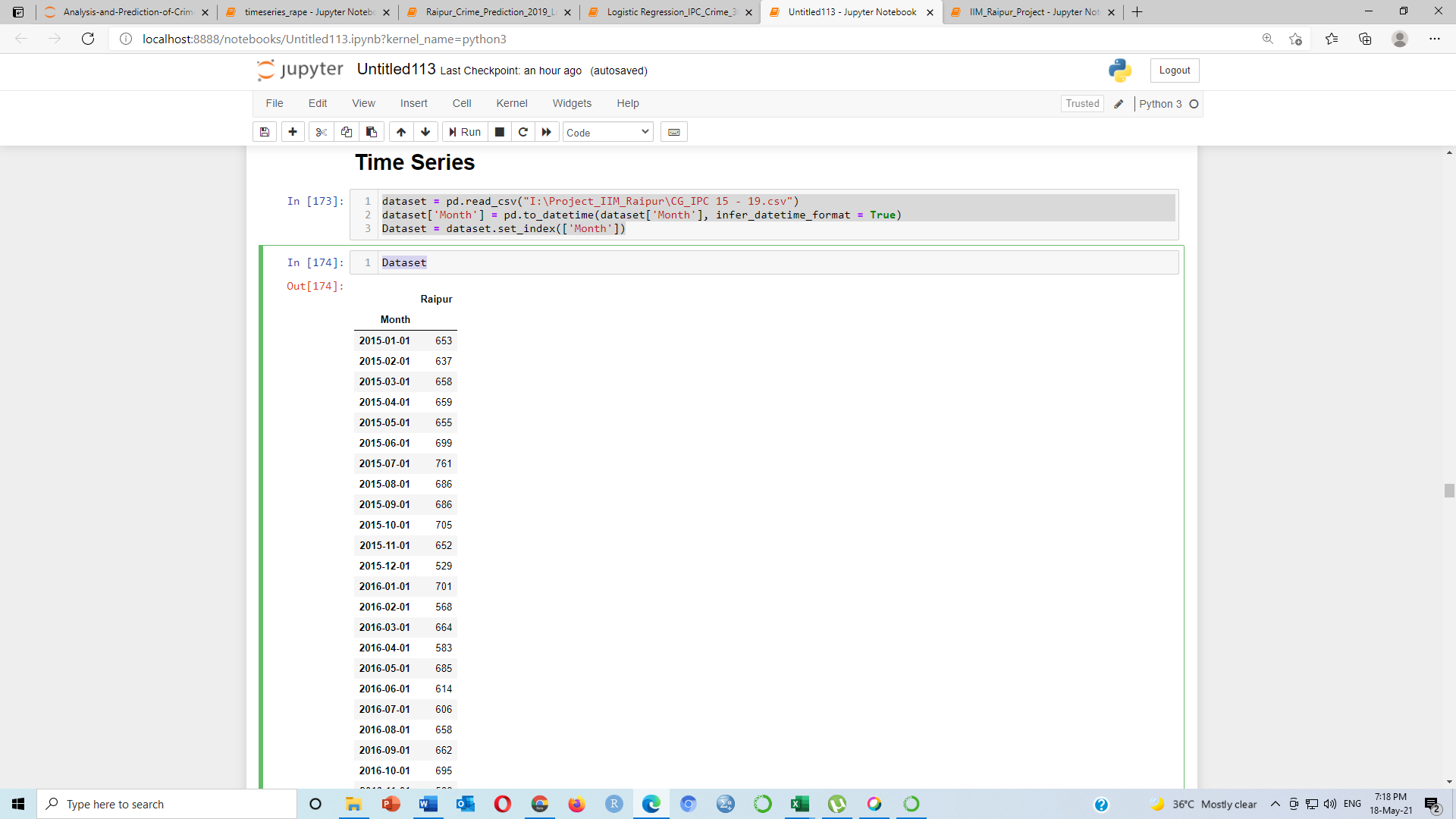
**Codes:**

dataset = pd.read\_csv("CG\_IPC 15 - 19.csv")

dataset['Month'] = pd.to\_datetime(dataset['Month'], infer\_datetime\_format = True)

Dataset = dataset.set\_index(['Month'])

Dataset



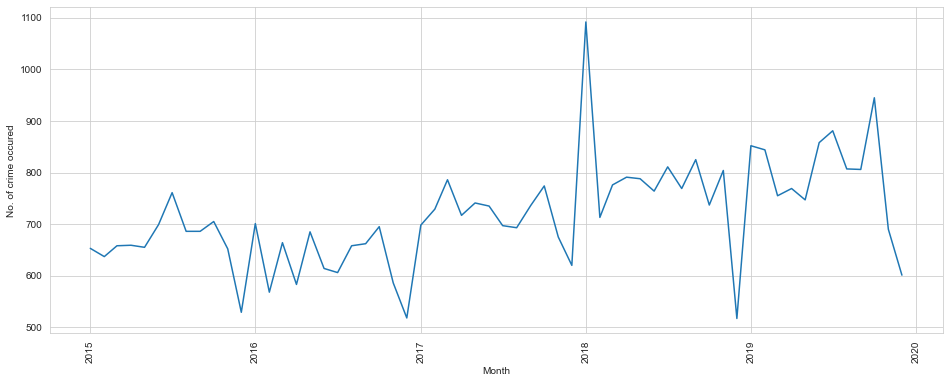
plt.figure(figsize=(16, 6))

plt.xticks(rotation=90)

plt.xlabel("Month")

plt.ylabel("No. of crime occured")

plt.plot(Dataset)



**#Check data is stationary or not through Deckey Fuller Test**

from statsmodels.tsa.stattools import adfuller

print ('Results of Dickey-Fuller Test:')

dftest = adfuller(Dataset['Raipur'], autolag='AIC')

dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','P-value', 'Lags Useed', 'Number of observations Used'])

for key, value in dftest[4].items():

dfoutput['Critical Value (%s)'%key] = value

print(dfoutput)

Results of Dickey-Fuller Test:

Test Statistic -1.902926

P-value 0.330686

Lags Useed 3.000000

Number of observations Used 56.000000

Critical Value (1%) -3.552928

Critical Value (5%) -2.914731

Critical Value (10%) -2.595137

dtype: float64

**here is P-value is not less than 0.05 than the data is not stationary**

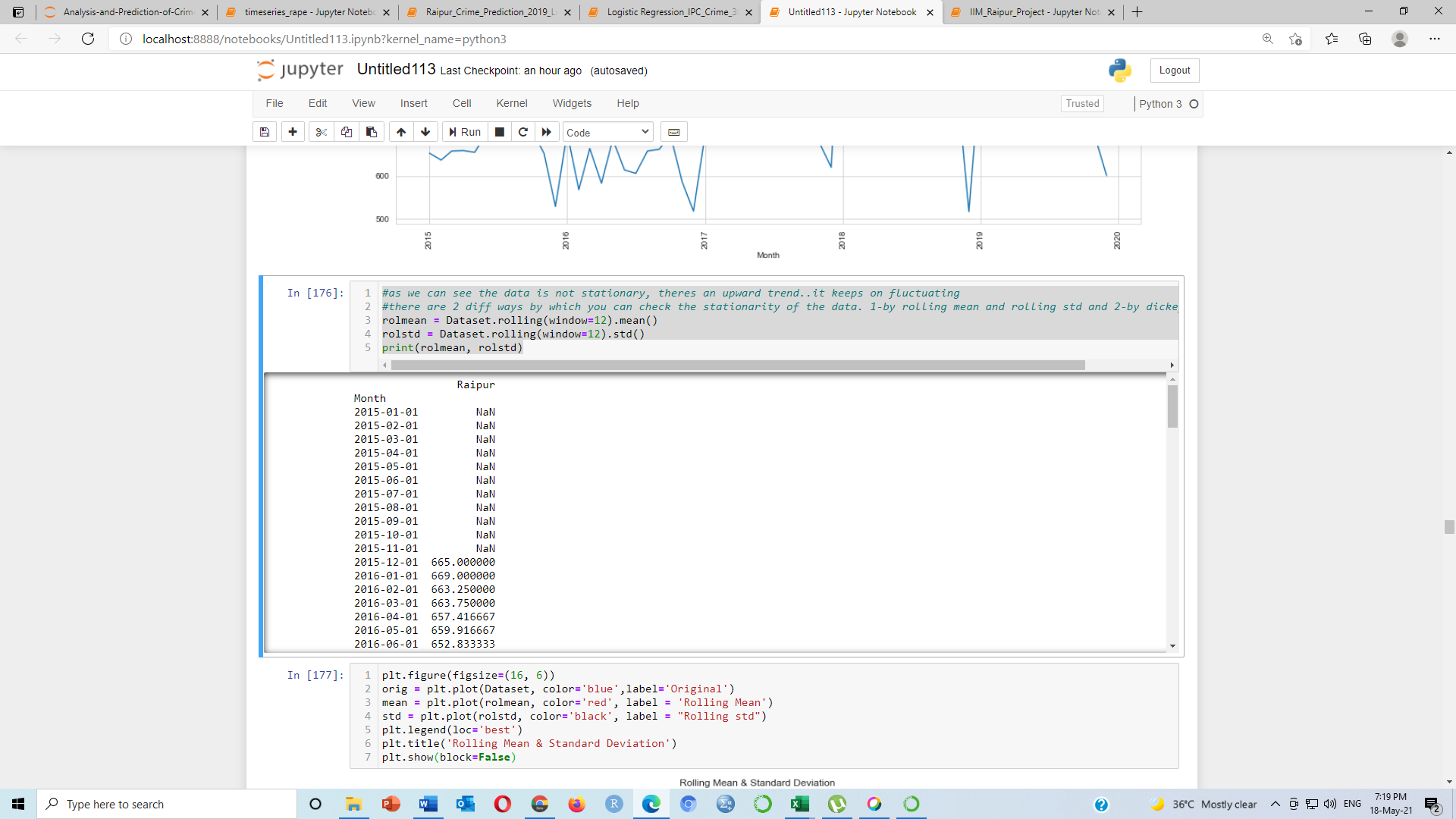
**#as we can see the data is not stationary, there are an upward trend..it keeps on non stationary**

#there are 2 diff ways by which you can check the stationarity of the data. 1-by rolling mean and rolling std and 2-by dickey-fuller test

rolmean = Dataset.rolling(window=12).mean()

rolstd = Dataset.rolling(window=12).std()

print(rolmean, rolstd)



plt.figure(figsize=(16, 6))

orig = plt.plot(Dataset, color='blue',label='Original')

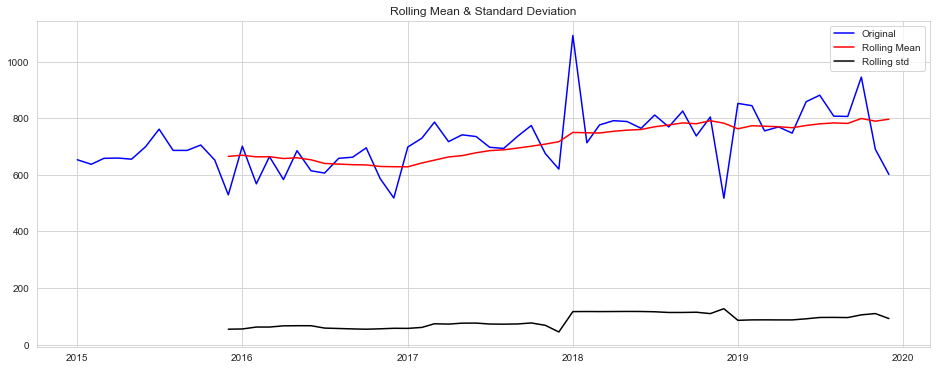
mean = plt.plot(rolmean, color='red', label = 'Rolling Mean')

std = plt.plot(rolstd, color='black', label = "Rolling std")

plt.legend(loc='best')

plt.title('Rolling Mean & Standard Deviation')

plt.show(block=False)



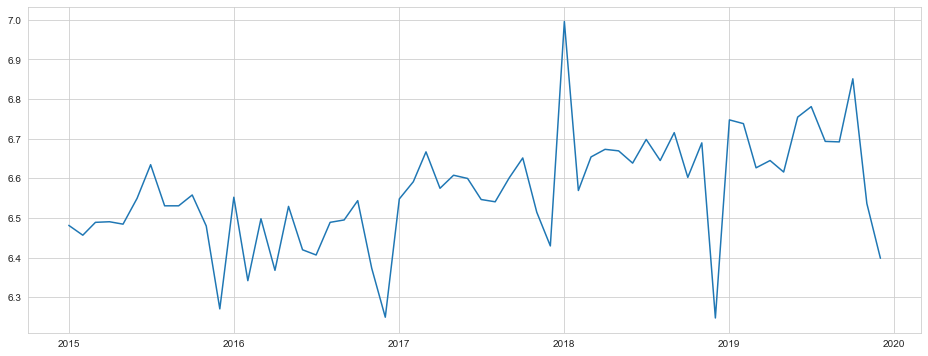
#from the above graph you can see that the std and mean is not constant, it keeps on changing

**#Estimate Trend**

plt.figure(figsize=(16, 6))

Dataset\_log = np.log(Dataset)

plt.plot(Dataset\_log)



**#Calculate moving avg**

movingAverage = Dataset\_log.rolling(window=12).mean()

movingSTD = Dataset\_log.rolling(window=12).std()

plt.figure(figsize=(16, 6))

plt.plot(Dataset\_log)

plt.plot(movingAverage, color='red')



**#finding the difference of the two types**

#the data can be made stationary by diff ways such as taking log ,squaring, cube root etc

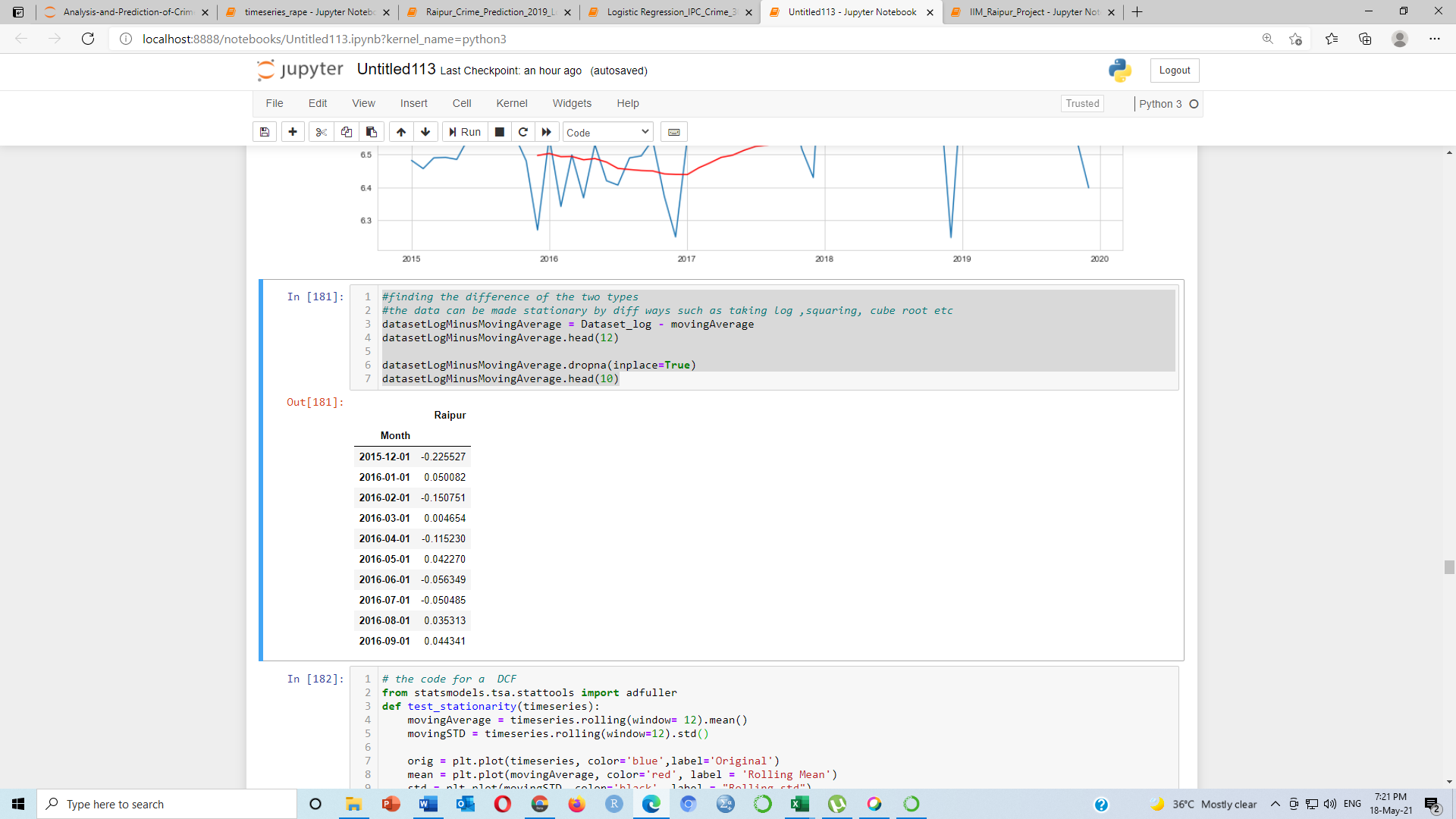
datasetLogMinusMovingAverage = Dataset\_log - movingAverage

datasetLogMinusMovingAverage.head(12)

#Remove NaN values

datasetLogMinusMovingAverage.dropna(inplace=True)

datasetLogMinusMovingAverage.head(10)



**# the code for a DCF Test**

from statsmodels.tsa.stattools import adfuller

def test\_stationarity(timeseries):

movingAverage = timeseries.rolling(window= 12).mean()

movingSTD = timeseries.rolling(window=12).std()

orig = plt.plot(timeseries, color='blue',label='Original')

mean = plt.plot(movingAverage, color='red', label = 'Rolling Mean')

std = plt.plot(movingSTD, color='black', label = "Rolling std")

plt.legend(loc='best')

plt.title('Rolling Mean & Standard Deviation')

plt.show(block=False)

print('Results of D-F Test')

dftest = adfuller(timeseries['Raipur'], autolag='AIC')

dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','P-value', 'Lags Useed', 'Number of observations Used'])

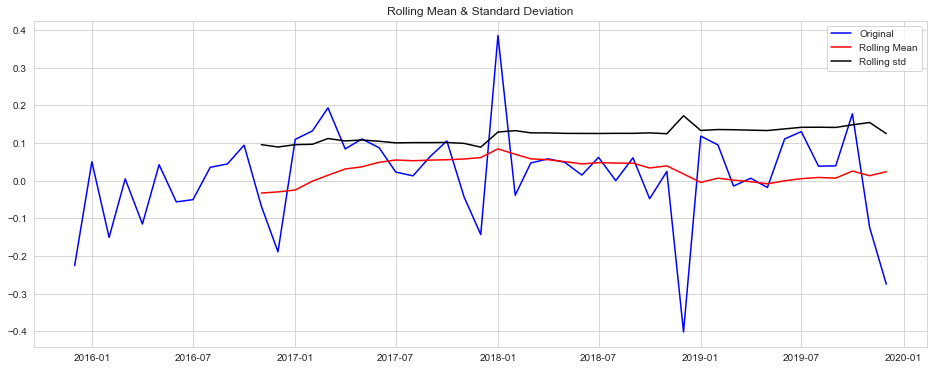
for key, value in dftest[4].items():

dfoutput['Critical Value (%s)'%key] = value

print(dfoutput)

plt.figure(figsize=(16, 6))

test\_stationarity(datasetLogMinusMovingAverage)



Results of D-F Test

Test Statistic -7.086270e+00

P**-value 4.530028e-10**

Lags Useed 0.000000e+00

Number of observations Used 4.800000e+01

Critical Value (1%) -3.574589e+00

dtype: float64

Test Statistic -7.086270e+00

P-value 4.530028e-10

Lags Useed 0.000000e+00

Number of observations Used 4.800000e+01

Critical Value (1%) -3.574589e+00

Critical Value (5%) -2.923954e+00

dtype: float64

Test Statistic -7.086270e+00

P-value 4.530028e-10

Lags Useed 0.000000e+00

Number of observations Used 4.800000e+01

Critical Value (1%) -3.574589e+00

Critical Value (5%) -2.923954e+00

Critical Value (10%) -2.600039e+00

dtype: float64

**#Calculate Data\_Log\_Shifting Difference**

plt.figure(figsize=(16, 6))

datasetLogDiffShifting = Dataset\_log - Dataset\_log.shift(1)

datasetLogDiffShifting.dropna(inplace=True)

test\_stationarity(datasetLogDiffShifting)

Results of D-F Test

Test Statistic -4.006503

**P-value 0.001377**

Lags Useed 10.000000

Number of observations Used 48.000000

Critical Value (1%) -3.574589

dtype: float64

Test Statistic -4.006503

P-value 0.001377

Lags Useed 10.000000

Number of observations Used 48.000000

Critical Value (1%) -3.574589

Critical Value (5%) -2.923954

dtype: float64

Test Statistic -4.006503

P-value 0.001377

Lags Useed 10.000000

Number of observations Used 48.000000

Critical Value (1%) -3.574589

Critical Value (5%) -2.923954

Critical Value (10%) -2.600039

dtype: float64

**Now the P-value is less than 0.05, i.e. data is stationary**

**#Decompose the data**

from statsmodels.tsa.seasonal import seasonal\_decompose

decomposition = seasonal\_decompose(Dataset\_log)

trend = decomposition.trend

seasonal = decomposition.seasonal

residual = decomposition.resid

plt.figure(figsize=(16, 6))

plt.subplot(411)

plt.plot(Dataset\_log, label='Original')

plt.legend(loc='best')

plt.subplot(412)

plt.plot(trend, label='Trend')

plt.legend(loc='best')

plt.subplot(413)

plt.plot(seasonal,label='Seasonality')

plt.subplot(414)

plt.plot(residual,label='Residuals')

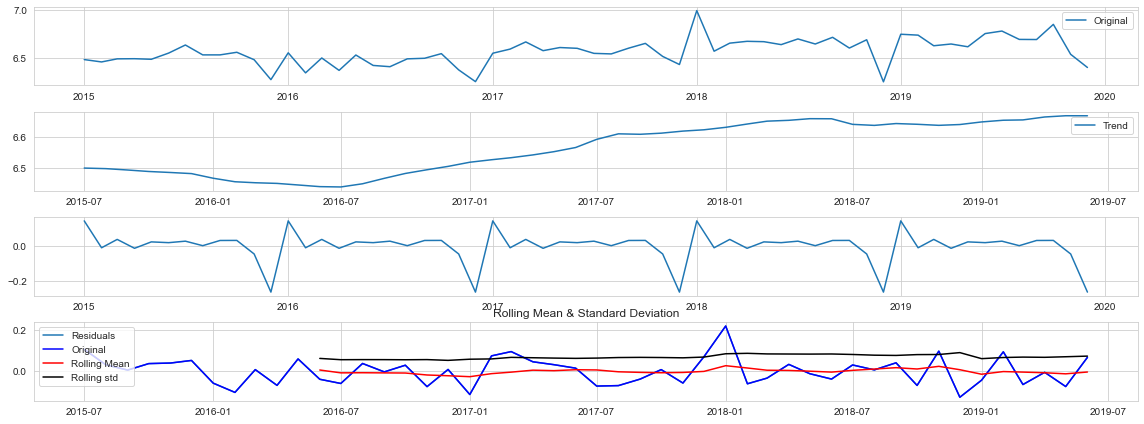
plt.legend(loc='best')

plt.tight\_layout()

decomposedLogData = residual

decomposedLogData.dropna(inplace=True)

test\_stationarity(decomposedLogData )



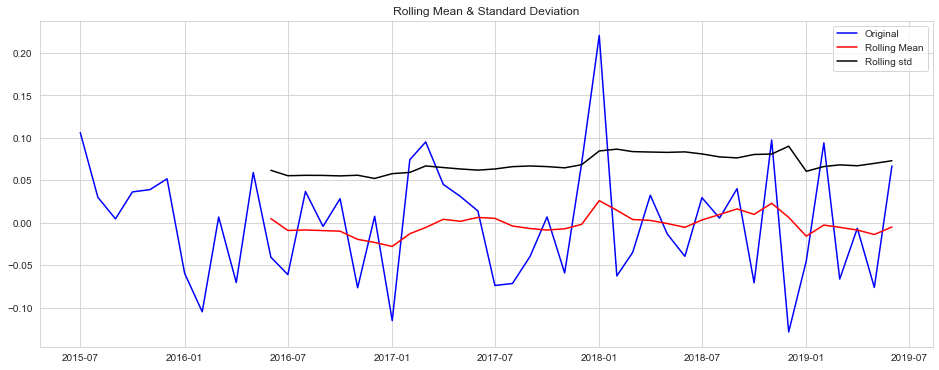
**#Check noise**

plt.figure(figsize=(16, 6))

decomposedLogData = residual

decomposedLogData.dropna(inplace=True)

test\_stationarity(decomposedLogData )



**#Plot ACF and PACF**

from statsmodels.tsa.stattools import acf, pacf

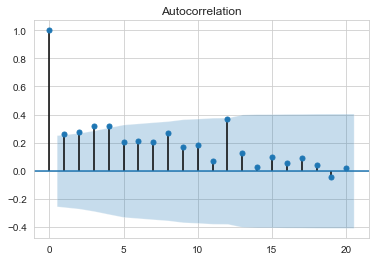
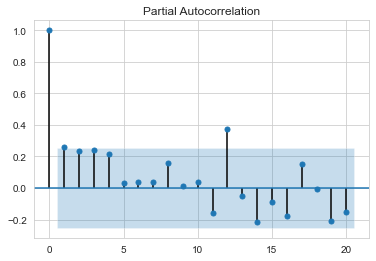
import statsmodels.api as sm

dataset = pd.read\_csv("CG\_IPC 15 - 19.csv", parse\_dates= True, index\_col=0)

sm.graphics.tsa.plot\_acf(dataset.values.squeeze(), lags=20)

sm.graphics.tsa.plot\_pacf(dataset.values.squeeze(), lags=20)

plt.show()

from statsmodels.tsa.stattools import acf, pacf

lag\_acf = acf(datasetLogDiffShifting, nlags=20)

lag\_pacf = pacf(datasetLogDiffShifting, nlags=20, method='ols')

**#plot ACF:**

plt.figure(figsize=(16, 6))

plt.subplot(121)

plt.plot(lag\_acf)

plt.axhline(y=0,linestyle='--',color='gray')

plt.axhline(y=1.96/np.sqrt(len(datasetLogDiffShifting)),linestyle='--',color='gray')

plt.axhline(y=1.96/np.sqrt(len(datasetLogDiffShifting)),linestyle='--',color='gray')

plt.title('Autocorrelation Funtion')

**#plot PACF:**

plt.subplot(122)

plt.plot(lag\_pacf)

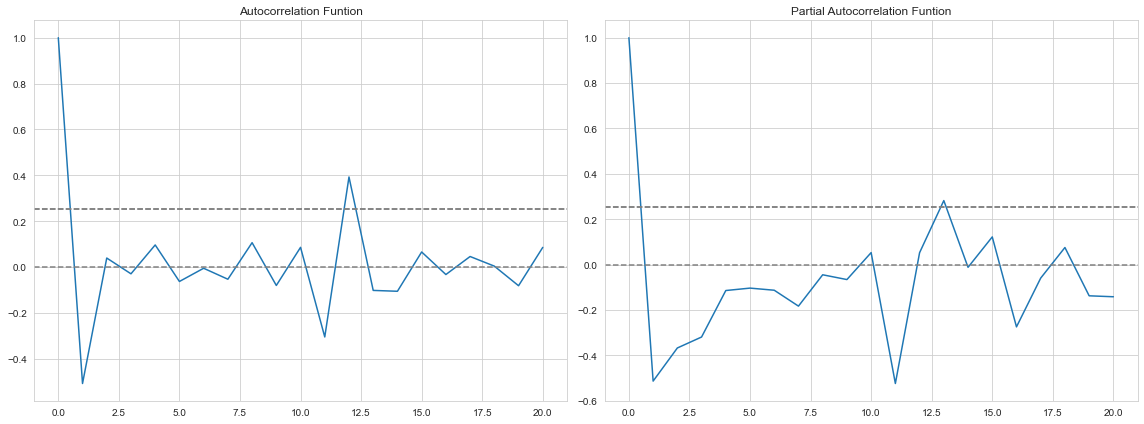
plt.axhline(y=0,linestyle='--',color='gray')

plt.axhline(y=1.96/np.sqrt(len(datasetLogDiffShifting)),linestyle='--',color='gray')

plt.axhline(y=1.96/np.sqrt(len(datasetLogDiffShifting)),linestyle='--',color='gray')

plt.title('Partial Autocorrelation Funtion')

plt.tight\_layout()



**#AR MODEL**

***#AR model with p,d,q value respectively 1,1,0***

from statsmodels.tsa.arima\_model import ARIMA

model = ARIMA(Dataset\_log, order=(1,1,0))

results\_AR = model.fit(disp=-1)

plt.figure(figsize=(16, 6))

plt.plot(datasetLogDiffShifting)

plt.plot(results\_AR.fittedvalues, color='red')

plt.title('RSS: %.4f'% sum((results\_AR.fittedvalues-datasetLogDiffShifting["Raipur"])\*\*2))

print('Plotting AR model')



***#AR model with p,d,q value respectively 0,1,1***

model = ARIMA(Dataset\_log, order=(0,1,1))

results\_AR = model.fit(disp=-1)

plt.figure(figsize=(16, 6))

plt.plot(datasetLogDiffShifting)

plt.plot(results\_AR.fittedvalues, color='red')

plt.title('RSS: %.4f'% sum((results\_AR.fittedvalues-datasetLogDiffShifting["Raipur"])\*\*2))

print('Plotting AR model')



***#MA Model***

***#MA model with p,d,q value respectively 1,1,0***

model = ARIMA(Dataset\_log, order=(1,1,0))

results\_MA = model.fit(disp=-1)

plt.figure(figsize=(16, 6))

plt.plot(datasetLogDiffShifting)

plt.plot(results\_MA.fittedvalues, color='red')

plt.title('RSS: %.4f'% sum((results\_MA.fittedvalues-datasetLogDiffShifting["Raipur"])\*\*2))

print('Plotting MA model')



***#MA model with p,d,q value respectively 0,1,1***

model = ARIMA(Dataset\_log, order=(0,1,1))

results\_MA = model.fit(disp=-1)

plt.figure(figsize=(16, 6))

plt.plot(datasetLogDiffShifting)

plt.plot(results\_MA.fittedvalues, color='red')

plt.title('RSS: %.4f'% sum((results\_MA.fittedvalues-datasetLogDiffShifting["Raipur"])\*\*2))

print('Plotting MA model')



***#ARIMA***

***#ARIMA model with p,d,q value respectively 1,1,1***

model = ARIMA(Dataset\_log, order=(1,1,1))

results\_ARIMA = model.fit(disp=-1)

plt.figure(figsize=(16, 6))

plt.plot(datasetLogDiffShifting)

plt.plot(results\_ARIMA.fittedvalues, color='red')

plt.title('RSS: %.4f'% sum((results\_ARIMA.fittedvalues-datasetLogDiffShifting["Raipur"])\*\*2))

print('Plotting ARIMA model')



***#Now Data transformation to original form***

predictions\_ARIMA\_diff = pd.Series(results\_ARIMA.fittedvalues, copy=True)

print(predictions\_ARIMA\_diff.head())

Month

2015-02-01 0.004051

2015-03-01 0.018278

2015-04-01 -0.005033

2015-05-01 -0.000909

2015-06-01 0.008058

dtype: float64

***#Convert to cumulative sum***

predictions\_ARIMA\_diff\_cumsum = predictions\_ARIMA\_diff.cumsum()

print(predictions\_ARIMA\_diff\_cumsum.head())

Month

2015-02-01 0.004051

2015-03-01 0.022330

2015-04-01 0.017297

2015-05-01 0.016388

2015-06-01 0.024447

dtype: float64

***#Show into Series***

predictions\_ARIMA\_log = pd.Series(Dataset\_log["Raipur"], index=Dataset\_log.index)

predictions\_ARIMA\_log = predictions\_ARIMA\_log.add(predictions\_ARIMA\_diff\_cumsum,fill\_value=0)

predictions\_ARIMA\_log.head()

Month

2015-01-01 6.481577

2015-02-01 6.460821

2015-03-01 6.511535

2015-04-01 6.508021

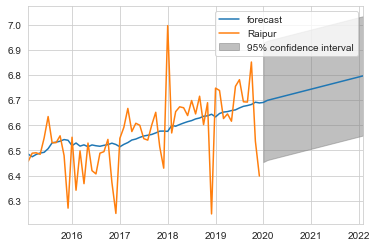
2015-05-01 6.501023

dtype: float64

***#ARIMA Forecasting (****this is not require because the data is seasonality****)***

results\_ARIMA.plot\_predict(1,85)

x=results\_ARIMA.forecast(steps=60)



**SARIMAX**

import statsmodels.api as sm

model = sm.tsa.statespace.SARIMAX(Dataset['Raipur'], order = (1,1,1), seasonal\_order=(1,1,1,12))

results = model.fit()

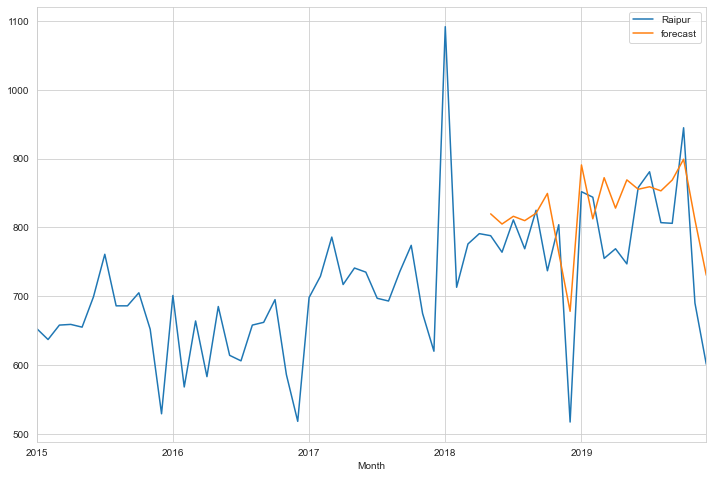
from pandas.tseries.offsets import DateOffset

future\_dates = [Dataset.index[-1]+DateOffset(months=x) for x in range(0,24)]

***#Orange line are Forecast value respect of original value***

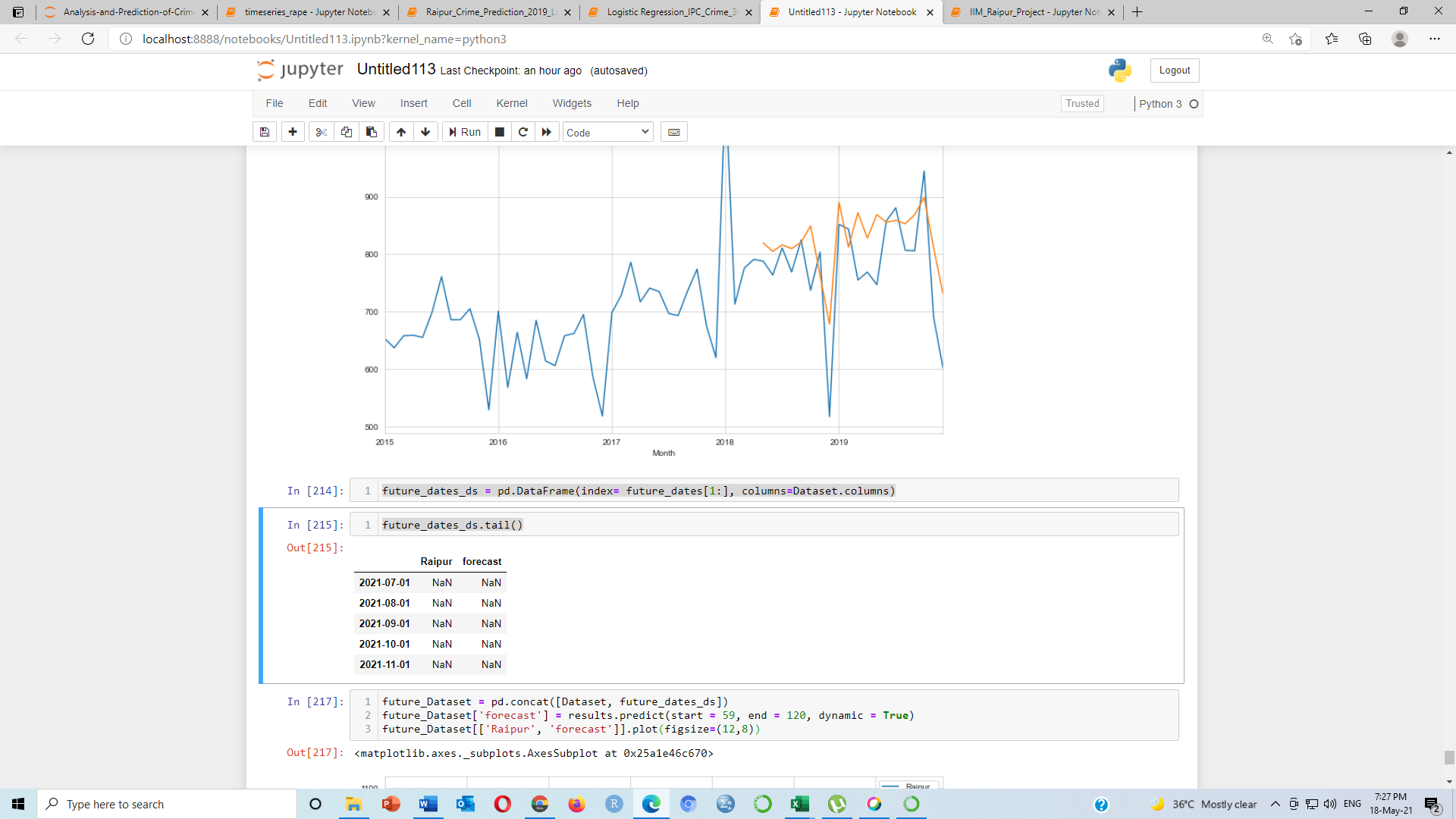
Dataset['forecast']=results.predict(start=40,end=100,dynamic=True)

Dataset[['Raipur','forecast']].plot(figsize=(12,8))



future\_dates\_ds = pd.DataFrame(index= future\_dates[1:], columns=Dataset.columns)

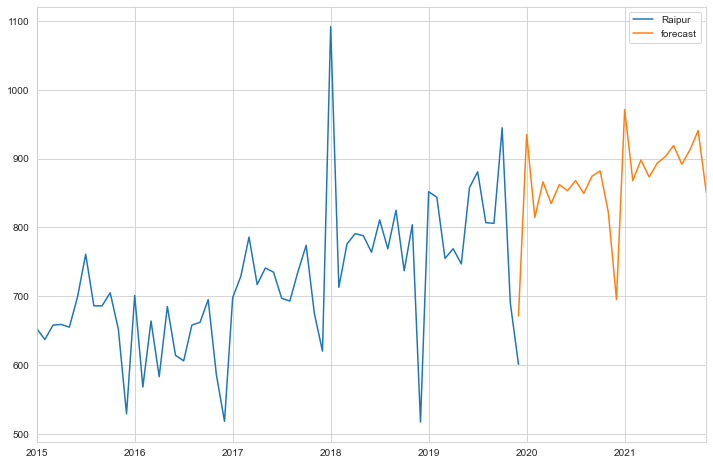
future\_dates\_ds.tail()



future\_Dataset = pd.concat([Dataset, future\_dates\_ds])

future\_Dataset['forecast'] = results.predict(start = 59, end = 120, dynamic = True)

future\_Dataset[['Raipur', 'forecast']].plot(figsize=(12,8))



**future\_Dataset**

| **Raipur** | **forecast** |
| --- | --- |
| **2015-01-01** | 653 | NaN |
| **2015-02-01** | 637 | NaN |
| **2015-03-01** | 658 | NaN |
| **2015-04-01** | 659 | NaN |
| **2015-05-01** | 655 | NaN |
| **...** | ... | ... |
| **2021-07-01** | NaN | 919.035444 |
| **2021-08-01** | NaN | 891.927444 |
| **2021-09-01** | NaN | 912.944987 |
| **2021-10-01** | NaN | 940.907232 |
| **2021-11-01** | NaN | 850.776455 |

83 rows × 2 columns