Project Name : - Algerian Forest Fire Dataset EDA , FE & Logistic Regression Algorithm.

1) Problem statement .

- This dataset comprises of Algerian Forest Fire Dataset taken from UCI .
- Link of the dataset is as follows :- https://archive.ics.uci.edu/ml/datasets/Algerian+Forest+Fires+Dataset++ .
- This Model Predicts using Logistic Regression Algorithm that whether there will be a fire or not in the Algerian Forest on the basis of various given circumstances in the data .

2) Data Collection.

- This dataset includes 244 instances that regroup a data of 2 regions of Algeria,namely the Brjajia region located in the northeast of Algeria and the Sidi Bel-abbes region located in the northwest of Algeria .
- 122 instances for each region .
- The Period is from June 2012 to September 2012.The Dataset includes 11 attributes and 1 output attribute i.e. Classes
- The data consists of 14 column and 246 rows.

2.1 Import Data and Required Packages

Importing Necessary Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from warnings import filterwarnings
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
filterwarnings('ignore')
%matplotlib inline
```

Loading the Algerian Forest Fire Dataset

In [2]:

```
df=pd.read_csv("Algerian_forest_fires_dataset_UPDATE.csv",header=1)
```

In [3]:

```
df.head()
```

Out[3]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
0	01	06	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	not fire

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	not fire
2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire
3	04	06	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	not fire
4	05	06	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	not fire

Attribute Information :-

Period Covered

- 1. Date : (DD/MM/YYYY) Day, month ('june' to 'september'), year (2012)

Weather data observations

- 1. Temp : temperature noon (temperature max) in Celsius degrees: 22 to 42
- 1. RH : Relative Humidity in %: 21 to 90
- 1. Ws :Wind speed in km/h: 6 to 29
- 1. Rain: total day in mm: 0 to 16.8

FWI Components

- 1. Fine Fuel Moisture Code (FFMC) index from the FWI system: 28.6 to 92.5
- 1. Duff Moisture Code (DMC) index from the FWI system: 1.1 to 65.9
- 1. Drought Code (DC) index from the FWI system: 7 to 220.4
- 1. Initial Spread Index (ISI) index from the FWI system: 0 to 18.5
- 1. Buildup Index (BUI) index from the FWI system: 1.1 to 68
- 1. Fire Weather Index (FWI) Index: 0 to 31.1
- 1. Classes: two classes, namely â€œFireâ€□ and â€œnot Fireâ€□

In [4]:

```
df.tail()
```

Out[4]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
241	26	09	2012	30	65	14	0	85.4	16	44.5	4.5	16.9	6.5	fire
242	27	09	2012	28	87	15	4.4	41.1	6.5	8	0.1	6.2	0	not fire
243	28	09	2012	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	not fire
244	29	09	2012	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	not fire
245	30	09	2012	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	not fire

In [5]:

```
df.shape
```

Out[5]:

```
(246, 14)
```

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 246 entries, 0 to 245
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
```

```
---  ------        --------------  -----
 0   day           246 non-null    object
 1   month         245 non-null    object
 2   year          245 non-null    object
 3   Temperature   245 non-null    object
 4    RH           245 non-null    object
 5    Ws           245 non-null    object
 6   Rain          245 non-null    object
 7   FFMC          245 non-null    object
 8   DMC           245 non-null    object
 9   DC            245 non-null    object
 10  ISI           245 non-null    object
 11  BUI           245 non-null    object
 12  FWI           245 non-null    object
 13  Classes       244 non-null    object
dtypes: object(14)
memory usage: 27.0+ KB
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
day            0
month          1
year           1
Temperature    1
 RH            1
 Ws            1
Rain           1
FFMC           1
DMC            1
DC             1
ISI            1
BUI            1
FWI            1
Classes        2
dtype: int64
```

In [8]:
```
df.describe()
```

Out[8]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
count	246	245	245	245	245	245	245	245	245	245	245	245	245	244
unique	33	5	2	20	63	19	40	174	167	199	107	175	128	9
top	01	07	2012	35	64	14	0	88.9	7.9	8	1.1	3	0.4	fire
freq	8	62	244	29	10	43	133	8	5	5	8	5	12	131

In [9]:
```
df.iloc[121:125,:]
```

Out[9]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
121	30	09	2012	25	78	14	1.4	45	1.9	7.5	0.2	2.4	0.1	not fire
122	Sidi-Bel Abbes Region Dataset	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
123	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
124	01	06	2012	32	71	12	0.7	57.1	2.5	8.2	0.6	2.8	0.2	not fire

In [ ]:

In [ ]:

2.2 Data Cleaning

Dropping row no 122 specifying region name & 123 respecifying the header

In [10]:
```
df.drop([122,123],inplace=True)
```

Resetting the index and dropping the index column

In [11]:
```
df.reset_index(inplace=True)
df.drop('index',axis=1,inplace=True)
```

In [ ]:

Creating a new column called Region reprenting [0:- Bejaia and 1- Sidi Bel-abbes]

In [12]:
```
df.loc[:122,"Region"]=0
df.loc[122:,"Region"]=1
```

Checking the Column Headers

In [13]:
```
df.columns
```

Out[13]:
```
Index(['day', 'month', 'year', 'Temperature', ' RH', ' Ws', 'Rain ', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes  ', 'Region'],
      dtype='object')
```

Removing unnecessary space in column headers using str.strip()

In [14]:
```
df.columns=df.columns.str.strip()
df.columns
```

Out[14]:
```
Index(['day', 'month', 'year', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes', 'Region'],
      dtype='object')
```

Dropping rows with null values

In [15]:
```
df.dropna(inplace=True)
```

Converting the necessary column dataye to int

In [16]:
```
df.dtypes
```

Out[16]:
```
day            object
month          object
year           object
Temperature    object
RH             object
Ws             object
Rain           object
```

```
FFMC            object
DMC             object
DC              object
ISI             object
BUI             object
FWI             object
Classes         object
Region         float64
dtype: object
```

In [17]:
```python
df[['day', 'month', 'year', 'Temperature', 'RH', 'Ws',"Region"]]=df[['day', 'month', 'year
```

In [18]:
```python
df.dtypes
```

Out[18]:
```
day              int32
month            int32
year             int32
Temperature      int32
RH               int32
Ws               int32
Rain            object
FFMC            object
DMC             object
DC              object
ISI             object
BUI             object
FWI             object
Classes         object
Region           int32
dtype: object
```

Values in df[Classes] has unnecessary spaces that are removed by str.strip()

In [19]:
```python
df.Classes.unique()
```

Out[19]:
```
array(['not fire   ', 'fire   ', 'fire', 'fire ', 'not fire', 'not fire ',
       'not fire     ', 'not fire    '], dtype=object)
```

In [20]:
```python
df.Classes=df.Classes.str.strip()
df.Classes.unique()
```

Out[20]:
```
array(['not fire', 'fire'], dtype=object)
```

Converting the Necessary Column Datatype to Float

In [21]:
```python
df.columns
```

Out[21]:
```
Index(['day', 'month', 'year', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes', 'Region'],
      dtype='object')
```

In [22]:
```python
df[['Rain', 'FFMC','DMC', 'DC', 'ISI', 'BUI', 'FWI']]=df[['Rain', 'FFMC','DMC', 'DC', 'ISI
```

In [23]:
```python
df.dtypes
```

Out[23]:
```
day                int32
month              int32
year               int32
Temperature        int32
```

```
RH                int32
Ws                int32
Rain            float64
FFMC            float64
DMC             float64
DC              float64
ISI             float64
BUI             float64
FWI             float64
Classes          object
Region            int32
dtype: object
```

Dropping the year column as the data is for the same year

In [24]:

```
df1=df.drop(['year'],axis=1)
```

DataFrame Description

In [25]:

```
df1.describe().T
```

Out[25]:

	count	mean	std	min	25%	50%	75%	max
day	243.0	15.761317	8.842552	1.0	8.00	16.0	23.00	31.0
month	243.0	7.502058	1.114793	6.0	7.00	8.0	8.00	9.0
Temperature	243.0	32.152263	3.628039	22.0	30.00	32.0	35.00	42.0
RH	243.0	62.041152	14.828160	21.0	52.50	63.0	73.50	90.0
Ws	243.0	15.493827	2.811385	6.0	14.00	15.0	17.00	29.0
Rain	243.0	0.762963	2.003207	0.0	0.00	0.0	0.50	16.8
FFMC	243.0	77.842387	14.349641	28.6	71.85	83.3	88.30	96.0
DMC	243.0	14.680658	12.393040	0.7	5.80	11.3	20.80	65.9
DC	243.0	49.430864	47.665606	6.9	12.35	33.1	69.10	220.4
ISI	243.0	4.742387	4.154234	0.0	1.40	3.5	7.25	19.0
BUI	243.0	16.690535	14.228421	1.1	6.00	12.4	22.65	68.0
FWI	243.0	7.035391	7.440568	0.0	0.70	4.2	11.45	31.1
Region	243.0	0.497942	0.501028	0.0	0.00	0.0	1.00	1.0

3. Exploratory Data Analysis

Encoding not fire as 0 and Fire as 1

In [26]:

```
set(df1.Classes)
```

Out[26]:

```
{'fire', 'not fire'}
```

In [27]:

```
label_encoder = preprocessing.LabelEncoder()
# Encode labels in column 'Classes'.
df1 ['Classes']= label_encoder.fit_transform(df1 ['Classes'])
df1.head()
```

Out[27]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Region
0	1	6	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	1	0
1	2	6	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	1	0
2	3	6	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	1	0
3	4	6	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	1	0
4	5	6	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	1	0

In [28]:

```
set(df1.Classes)
```

Out[28]:

```
{0, 1}
```

In [29]:

```
df1.corr()
```

Out[29]:

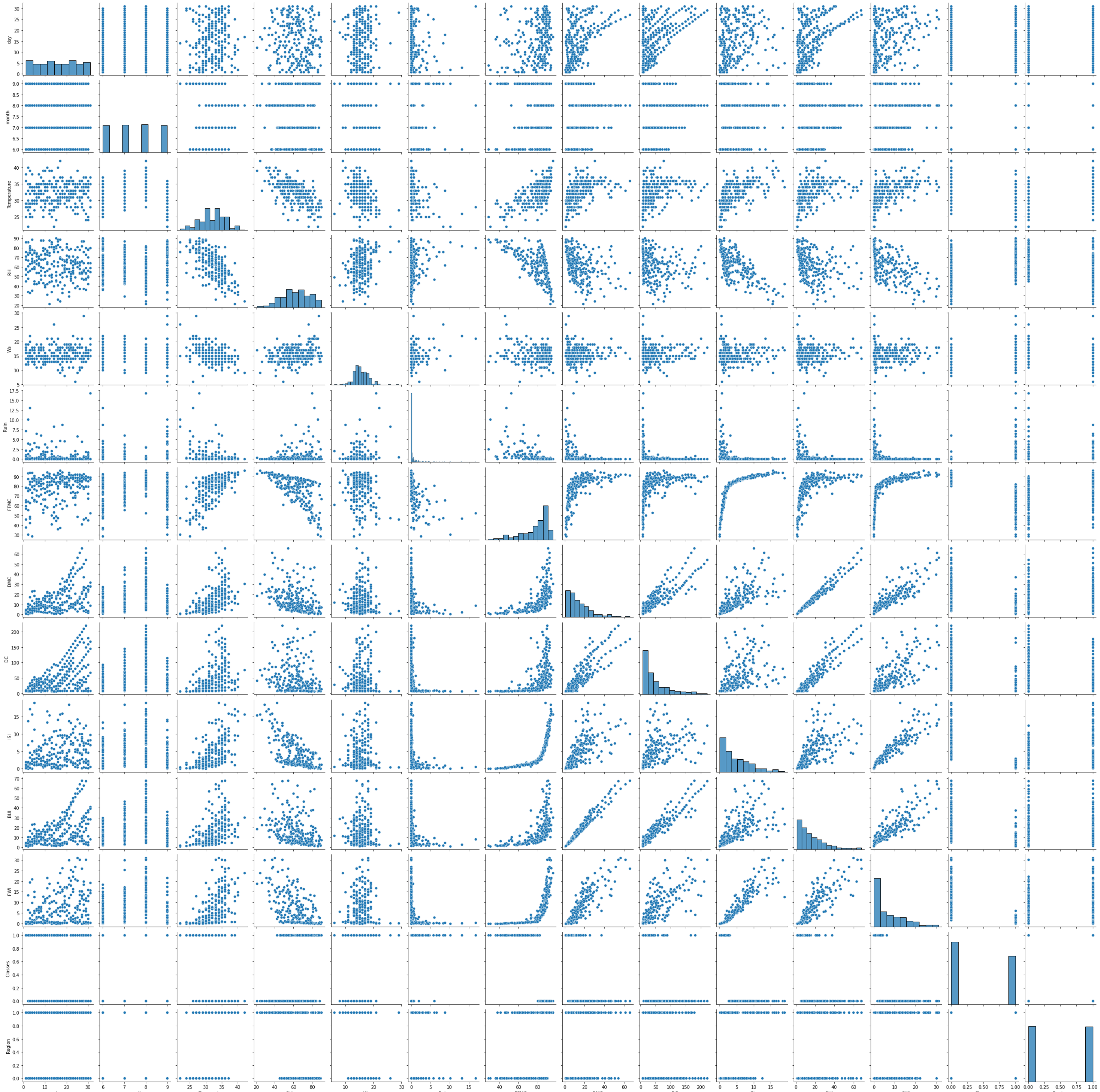
	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	
day	1.000000	-0.000369	0.097227	-0.076034	0.047812	-0.112523	0.224956	0.491514	0.527952	0.1
month	-0.000369	1.000000	-0.056781	-0.041252	-0.039880	0.034822	0.017030	0.067943	0.126511	0.0
Temperature	0.097227	-0.056781	1.000000	-0.651400	-0.284510	-0.326492	0.676568	0.485687	0.376284	0.6
RH	-0.076034	-0.041252	-0.651400	1.000000	0.244048	0.222356	-0.644873	-0.408519	-0.226941	-0.6
Ws	0.047812	-0.039880	-0.284510	0.244048	1.000000	0.171506	-0.166548	-0.000721	0.079135	0.0
Rain	-0.112523	0.034822	-0.326492	0.222356	0.171506	1.000000	-0.543906	-0.288773	-0.298023	-0.3
FFMC	0.224956	0.017030	0.676568	-0.644873	-0.166548	-0.543906	1.000000	0.603608	0.507397	0.7
DMC	0.491514	0.067943	0.485687	-0.408519	-0.000721	-0.288773	0.603608	1.000000	0.875925	0.6
DC	0.527952	0.126511	0.376284	-0.226941	0.079135	-0.298023	0.507397	0.875925	1.000000	0.5
ISI	0.180543	0.065608	0.603871	-0.686667	0.008532	-0.347484	0.740007	0.680454	0.508643	1.0
BUI	0.517117	0.085073	0.459789	-0.353841	0.031438	-0.299852	0.592011	0.982248	0.941988	0.6
FWI	0.350781	0.082639	0.566670	-0.580957	0.032368	-0.324422	0.691132	0.875864	0.739521	0.9
Classes	-0.202840	-0.024004	-0.516015	0.432161	0.069964	0.379097	-0.769492	-0.585658	-0.511123	-0.7
Region	0.000821	0.001857	0.269555	-0.402682	-0.181160	-0.040013	0.222241	0.192089	-0.078734	0.2

In [30]:

```
sns.pairplot(df1)
```

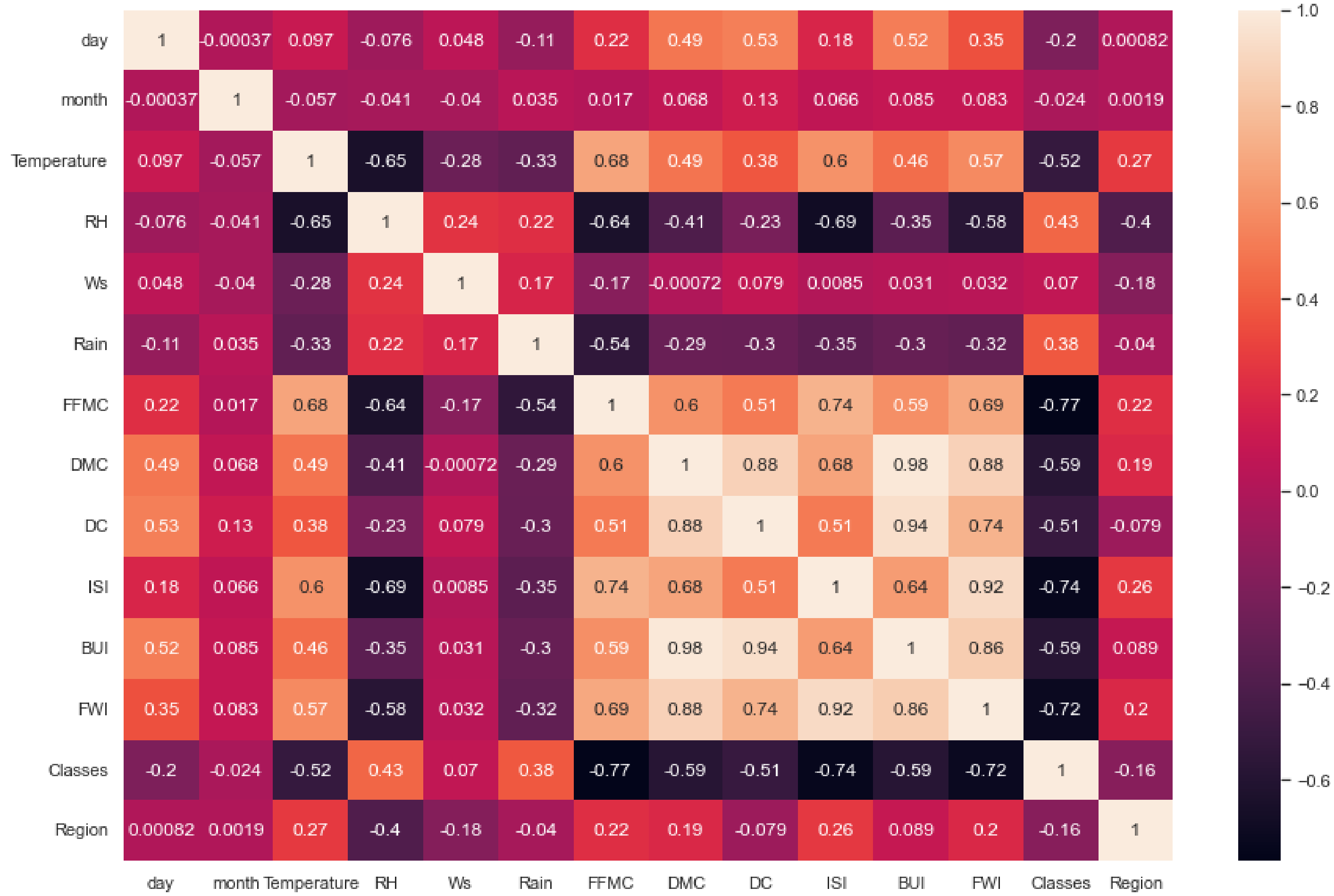
Out[30]:

```
<seaborn.axisgrid.PairGrid at 0x2772371e340>
```



In [31]:
```python
sns.set(rc={'figure.figsize':(15,10)})
sns.heatmap(df1.corr(),annot=True)
```

Out[31]: <AxesSubplot:>



Report

- RH is negatively corelated with Temperature, FFMC and ISI
- Rain is negatively correlated with Temperature and FFMC,DMC, ISI and BUI

Histogram

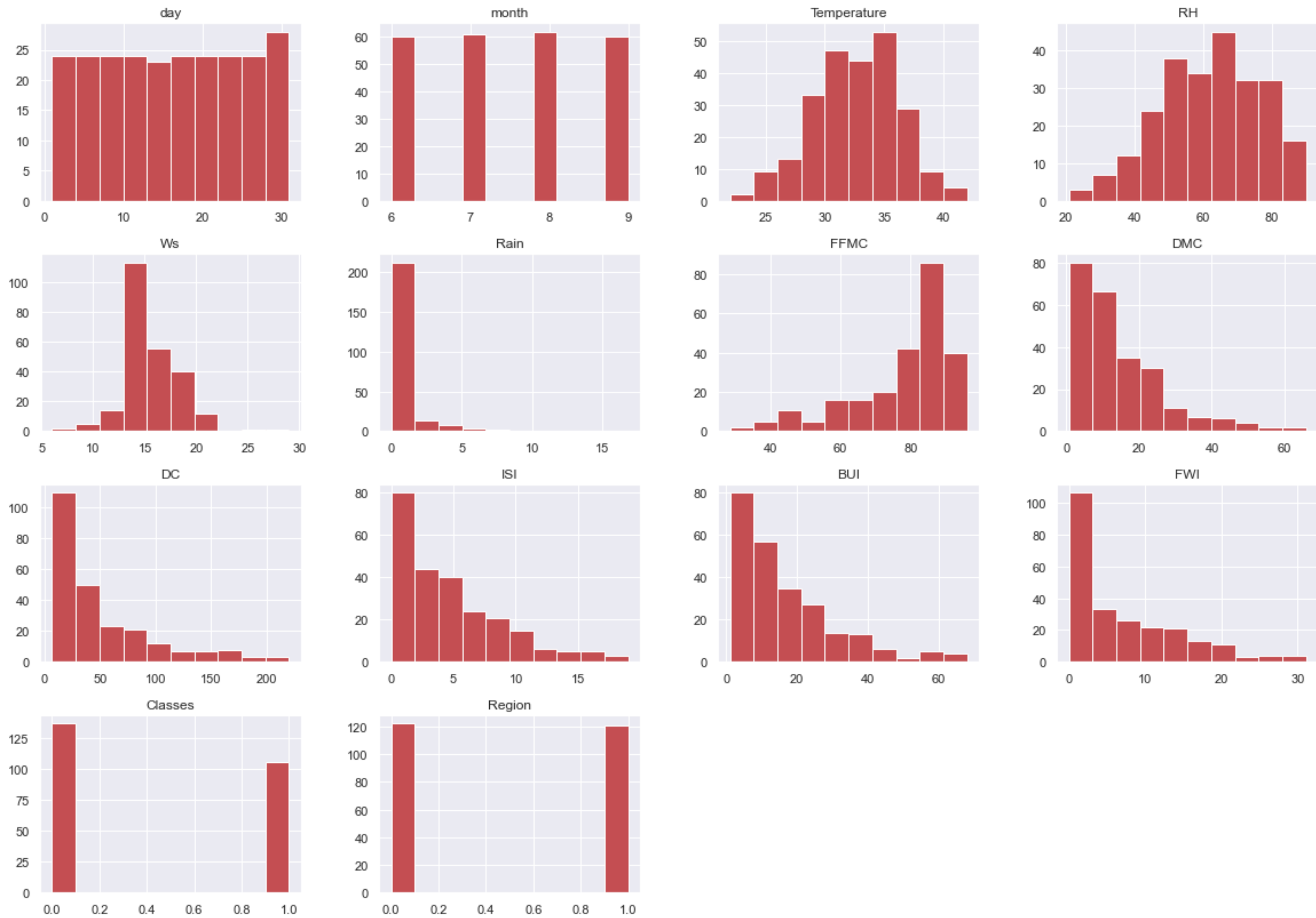
-A histogram is basically used to represent data provided in a form of sme groups

In [32]:

```
df1.hist(figsize=(20,14),color='r')
```

Out[32]:

```
array([[<AxesSubplot:title={'center':'day'}>,
        <AxesSubplot:title={'center':'month'}>,
        <AxesSubplot:title={'center':'Temperature'}>,
        <AxesSubplot:title={'center':'RH'}>],
       [<AxesSubplot:title={'center':'Ws'}>,
        <AxesSubplot:title={'center':'Rain'}>,
        <AxesSubplot:title={'center':'FFMC'}>,
        <AxesSubplot:title={'center':'DMC'}>],
       [<AxesSubplot:title={'center':'DC'}>,
        <AxesSubplot:title={'center':'ISI'}>,
        <AxesSubplot:title={'center':'BUI'}>,
        <AxesSubplot:title={'center':'FWI'}>],
       [<AxesSubplot:title={'center':'Classes'}>,
        <AxesSubplot:title={'center':'Region'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```



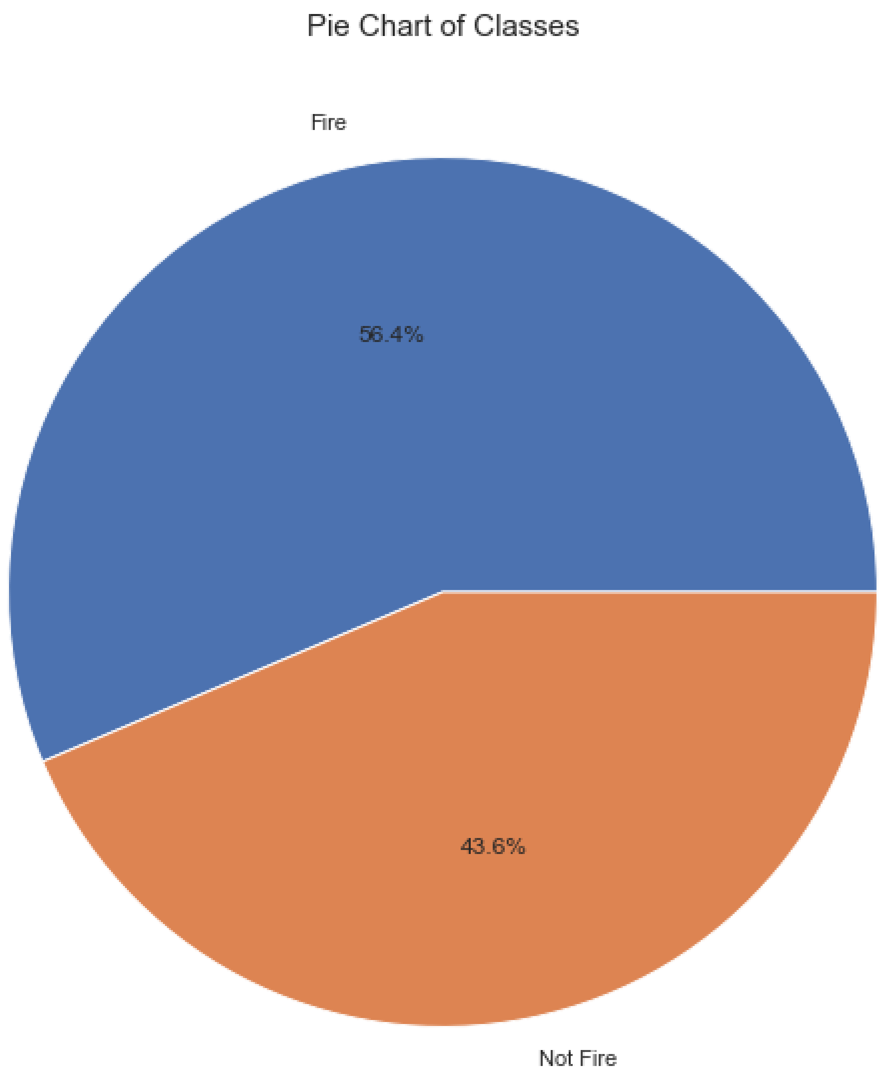
Percentage for Pie Chart

In [33]:
```
percentage=df.Classes.value_counts(normalize=True)*100
percentage
```

Out[33]:
```
fire         56.378601
not fire     43.621399
Name: Classes, dtype: float64
```

Plotting Pie chart

In [34]:
```
classes_labels=['Fire','Not Fire']
plt.figure(figsize=(15,10))
plt.pie(percentage,labels=classes_labels,autopct="%1.1f%%")
plt.title("Pie Chart of Classes",fontsize=15)
plt.show()
```



Model Building Using Logistic Regression

In [35]:

```
df1
```

Out[35]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Region
0	1	6	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	1	0
1	2	6	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	1	0
2	3	6	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	1	0
3	4	6	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	1	0
4	5	6	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
239	26	9	30	65	14	0.0	85.4	16.0	44.5	4.5	16.9	6.5	0	1
240	27	9	28	87	15	4.4	41.1	6.5	8.0	0.1	6.2	0.0	1	1
241	28	9	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	1	1
242	29	9	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	1	1
243	30	9	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	1	1

243 rows × 14 columns

In [36]:

```
X = df1[['Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC',
        'ISI', 'BUI', 'FWI','Region']]
X
```

Out[36]:

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region
0	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0
1	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0
2	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0
3	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	0
4	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	0
...	...	...	...	...	...	...	...	...	...	...	...
239	30	65	14	0.0	85.4	16.0	44.5	4.5	16.9	6.5	1
240	28	87	15	4.4	41.1	6.5	8.0	0.1	6.2	0.0	1
241	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	1
242	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	1
243	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	1

243 rows × 11 columns

In [37]:

```
y=df1['Classes']
y
```

Out[37]:

```
0      1
1      1
2      1
3      1
4      1
      ..
239    0
240    1
241    1
242    1
243    1
Name: Classes, Length: 243, dtype: int32
```

In [38]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=32,test_size=.33)
```

Feature Scaling

In [39]:

```
def Feature_Scaling(X_train, X_test):
    scaler = StandardScaler()
    X_train_after_Standardisation = scaler.fit_transform(X_train)
    X_test_after_Standardisation = scaler.transform(X_test)
    return X_train_after_Standardisation, X_test_after_Standardisation
```

In [40]:

```
X_train_after_Standardisation,X_test_after_Standardisation=Feature_Scaling(X_train, X_test
```

In [41]:
```
logistic_regression=LogisticRegression()
```

In [42]:
```
logistic_regression.fit(X_train_after_Standardisation,y_train)
```

Out[42]:
```
▾ LogisticRegression
LogisticRegression()
```

In [43]:
```
print('Intercept is :',logistic_regression.intercept_)
print('Coefficient is :',logistic_regression.coef_)
```

```
Intercept is : [-1.69998774]
Coefficient is : [[-0.1914346   0.02785304  0.02704621 -0.21299985 -2.34099273  0.26915864
   0.11560254 -2.323554   -0.26909869 -1.80866713 -0.09778616]]
```

In [44]:
```
print("Training Score:",logistic_regression.score(X_train_after_Standardisation, y_train))
print("Test Score:",logistic_regression.score(X_test_after_Standardisation,y_test))
```

```
Training Score: 0.9814814814814815
Test Score: 0.9629629629629629
```

In [45]:
```
Logistic_Regression_Prediction=logistic_regression.predict(X_test_after_Standardisation)
```

In [46]:
```
accuracy_score(y_test,Logistic_Regression_Prediction)
```

Out[46]:
```
0.9629629629629629
```

In [47]:
```
Actual_predicted = pd.DataFrame({'Actual': y_test, 'Predicted': Logistic_Regression_Predic
Actual_predicted['Report']=abs(Actual_predicted['Actual']-Actual_predicted['Predicted'])
Actual_predicted['Classes']= np.where(Actual_predicted['Report']== 0,'Matched','Unmatched'
Actual_predicted_group_df=Actual_predicted.groupby(['Classes']).agg({'Classes':['count']})
Actual_predicted_group_df.reset_index()
```

Out[47]:

		Classes
		count
0	Matched	78
1	Unmatched	3

Evaluation of a Classification Model

In machine learning, once we have a result of the classification problem, how do we measure how accurate our classification is? For a regression problem, we have different metrics like R Squared score, Mean Squared Error etc. what are the metrics to measure the credibility of a classification model?

Metrics In a regression problem, the accuracy is generally measured in terms of the difference in the actual values and the predicted values. In a classification problem, the credibility of the model is measured using the confusion matrix generated, i.e., how accurately the true positives and true negatives were predicted. The different metrics used for this purpose are:

- Accuracy
- Recall
- Precision
- F1 Score
- Specifity
- AUC(Area Under the Curve)
- RUC(Receiver Operator Characteristic)

Confusion Matrix

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model.

In [49]:

```
# Confusion Matrix
conf_mat = confusion_matrix(y_test,Logistic_Regression_Prediction)
conf_mat
```

Out[49]:

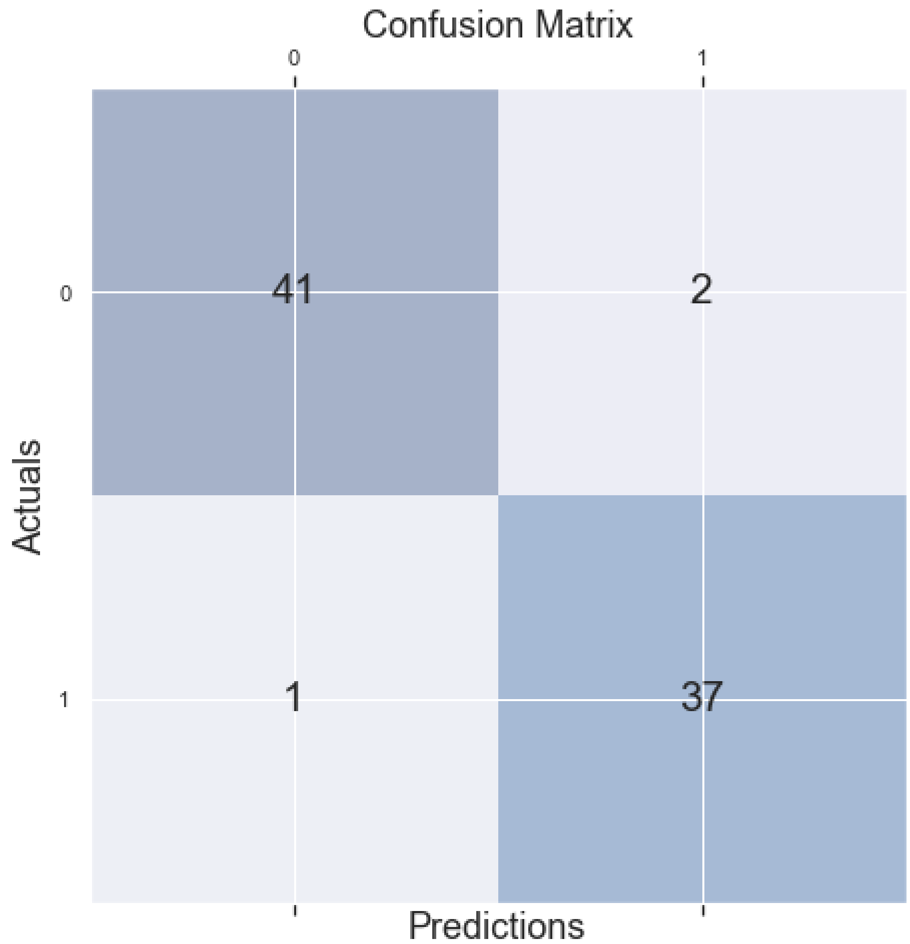
```
array([[41,  2],
       [ 1, 37]], dtype=int64)
```

Plotting Confusion Matrix

In [50]:

```
conf_matrix = confusion_matrix(y_true=y_test, y_pred=Logistic_Regression_Prediction)
#
# Print the confusion matrix using Matplotlib
#
fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix, cmap=plt.cm.Blues, alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```



Splitting the Confusion Matrix

In [51]:
```
true_positive = conf_mat[0][0]
false_positive = conf_mat[0][1]
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]
```

Accuracy

The mathematical formula is :

Accuracy= $\frac{(TP+TN)}{(TP+TN+FP+FN)}$

Or, it can be said that it's defined as the total number of correct classifications divided by the total number of classifications.

In [52]:
```
# Breaking down the formula for Accuracy
Accuracy = (true_positive + true_negative) / (true_positive +false_positive + false_negati
Accuracy
```

Out[52]:
```
0.9629629629629629
```

Our Model has an accuracy of 96%

Precision

Precision is a measure of amongst all the positive predictions, how many of them were actually positive.

Mathematically,

Precision= $\frac{TP}{(TP+FP)}$

In [53]:

```
Precision = true_positive/(true_positive+false_positive)
Precision
```

Out[53]:

```
0.9534883720930233
```

Our model has an Precision of 95%

Recall or Sensitivity

The mathematical formula is:

Recall= $\frac{TP}{(TP+FN)}$

Or, as the name suggests, it is a measure of: from the total number of positive results how many positives were correctly predicted by the model.

It shows how relevant the model is, in terms of positive results only.

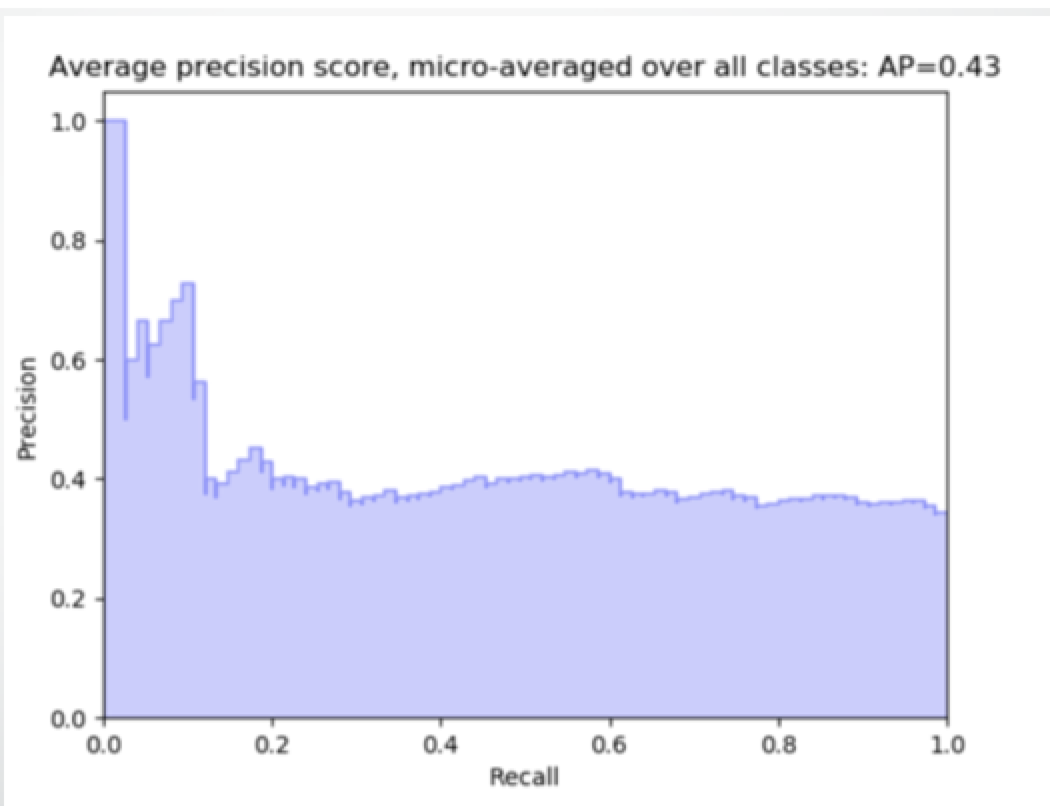
In [54]:

```
Recall = true_positive/(true_positive+false_negative)
Recall
```

Out[54]:

```
0.9761904761904762
```

Our Model has an Recall of 98%



As observed from the graph, with an increase in the Recall, there is a drop in Precision of the model.

So the question is - what to go for? Precision or Recall?

Well, the answer is: it depends on the business requirement.

For example, if you are predicting fire, you need a 100 % recall. But suppose you are predicting whether a person is innocent or not, you need 100% precision.

Can we maximise both at the same time? No

So, there is a need for a better metric then?

Yes. And it's called an *F1 Score*

F1 Score

F1 Score

From the previous examples, it is clear that we need a metric that considers both Precision and Recall for evaluating a model. One such metric is the F1 score.

F1 score is defined as the harmonic mean of Precision and Recall.

The mathematical formula is: F1 score= $\frac{2*((Precision*Recall)}{(Precision+Recall))}$

In [55]:

```
F1_Score = 2*(Recall * Precision) / (Recall + Precision)
F1_Score
```

Out[55]:

```
0.9647058823529412
```

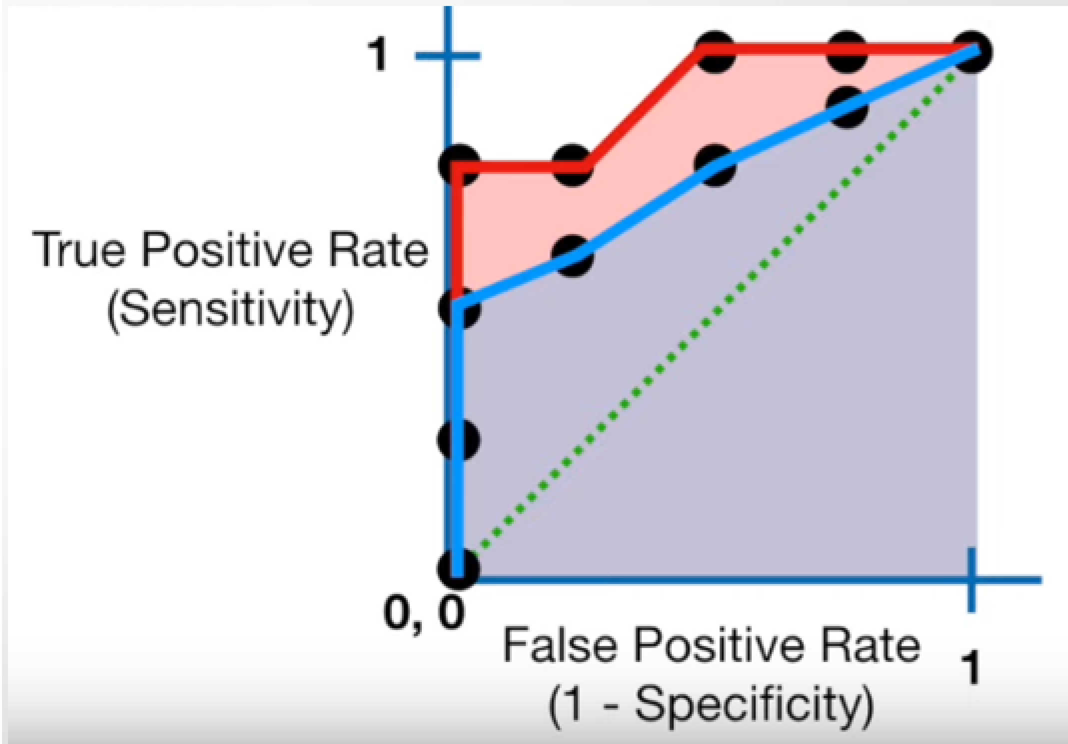
Our Model has an F1 Score of 96%

ROC & AUC

What is the significance of Roc curve and AUC?

In real life, we create various models using different algorithms that we can use for classification purpose. We use AUC to determine which model is the best one to use for a given dataset. Suppose we have created Logistic regression, SVM as well as a clustering model for classification purpose. We will calculate AUC for all the models seperately. The model with highest AUC value will be the best model to use.

AUC(Area Under Curve)



- It helps us to choose the best model amongst the models for which we have plotted the ROC curves
- The best model is the one which encompasses the maximum area under it.
- In the adjacent diagram, amongst the two curves, the model that resulted in the red one should be chosen as it clearly covers more area than the blue one

In [56]:

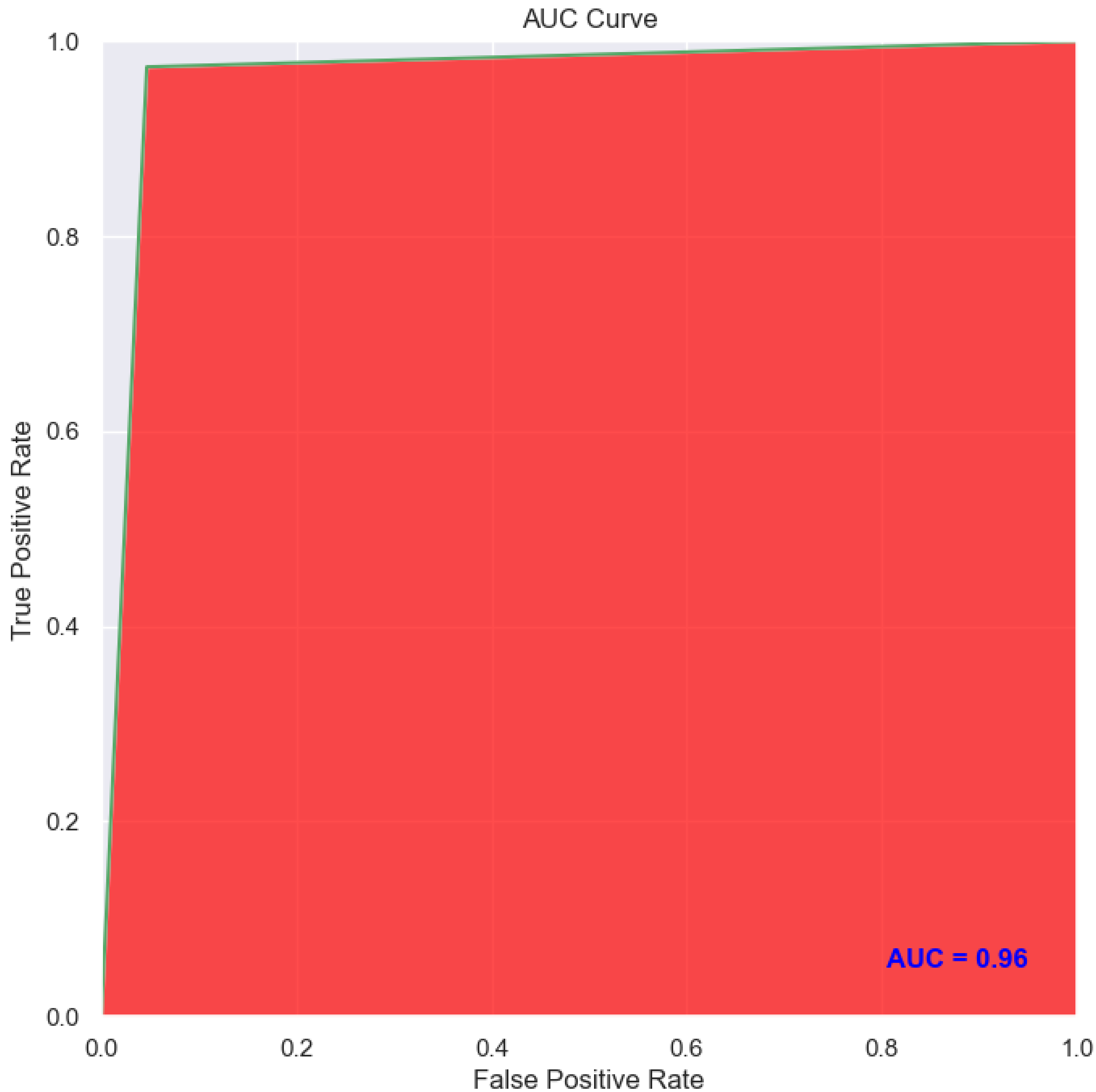
```
auc = roc_auc_score(y_test, Logistic_Regression_Prediction)
auc
```

Out[56]:

```
0.9635862913096697
```

In [57]:

```
false_positive_rate, true_positive_rate, thresolds = metrics.roc_curve(y_test, Logistic_Re
plt.figure(figsize=(10, 8), dpi=100)
plt.axis('scaled')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.title("AUC Curve")
plt.plot(false_positive_rate, true_positive_rate, 'g')
plt.fill_between(false_positive_rate, true_positive_rate, facecolor='red', alpha=0.7)
plt.text(0.95, 0.05, 'AUC = %0.2f' % auc, ha='right', fontsize=12, weight='bold', color='b
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



ROC(Receiver Operator Characteristic)

We know that the classification algorithms work on the concept of probability of occurrence of the possible outcomes. A probability value lies between 0 and 1. Zero means that there is no probability of occurrence and one means that the occurrence is certain.

But while working with real-time data, it has been observed that we seldom get a perfect 0 or 1 value. Instead of that, we get different decimal values lying between 0 and 1. Now the question is if we are not getting binary probability values how are we actually determining the class in our classification problem?

There comes the concept of Threshold. A threshold is set, any probability value below the threshold is a negative outcome, and anything more than the threshold is a favourable or the positive outcome. For Example, if the threshold is 0.5, any probability value below 0.5 means a negative or an unfavourable outcome and any value above 0.5 indicates a positive or favourable outcome.

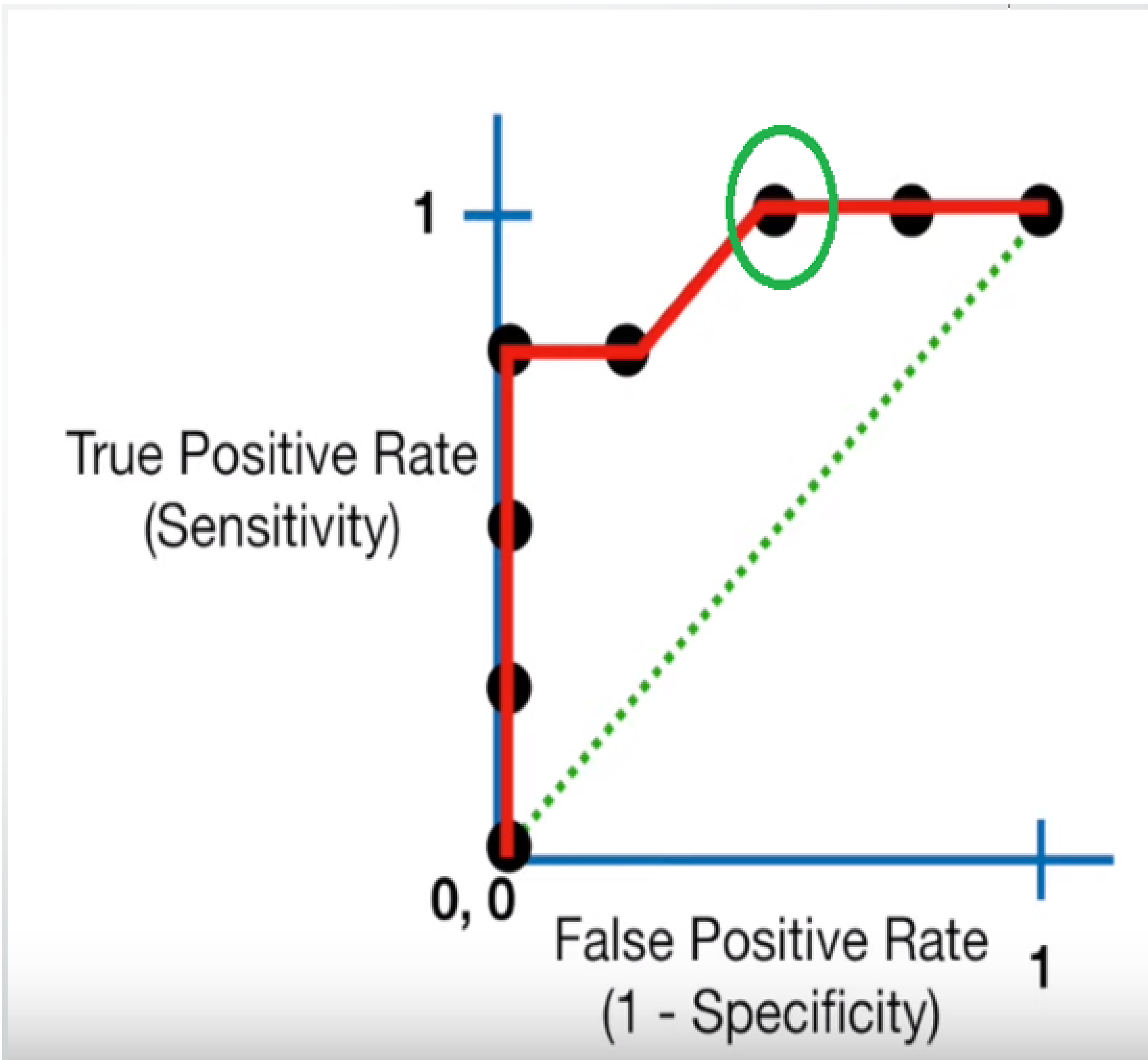
Now, the question is, what should be an ideal threshold?

The following diagram shows a typical logistic regression curve.

- The horizontal lines represent the various values of thresholds ranging from 0 to 1.
- Let's suppose our classification problem was to identify the obese people from the given data.
- The green markers represent obese people and the red markers represent the non-obese people.
- Our confusion matrix will depend on the value of the threshold chosen by us.
- For Example, if 0.25 is the threshold then

```
TP(actually obese)=3
TN(Not obese)=2
FP(Not obese but predicted obese)=2(the two red squares above the 0.25 line)
FN(Obese but predicted as not obese )=1(Green circle below 0.25line  )
```

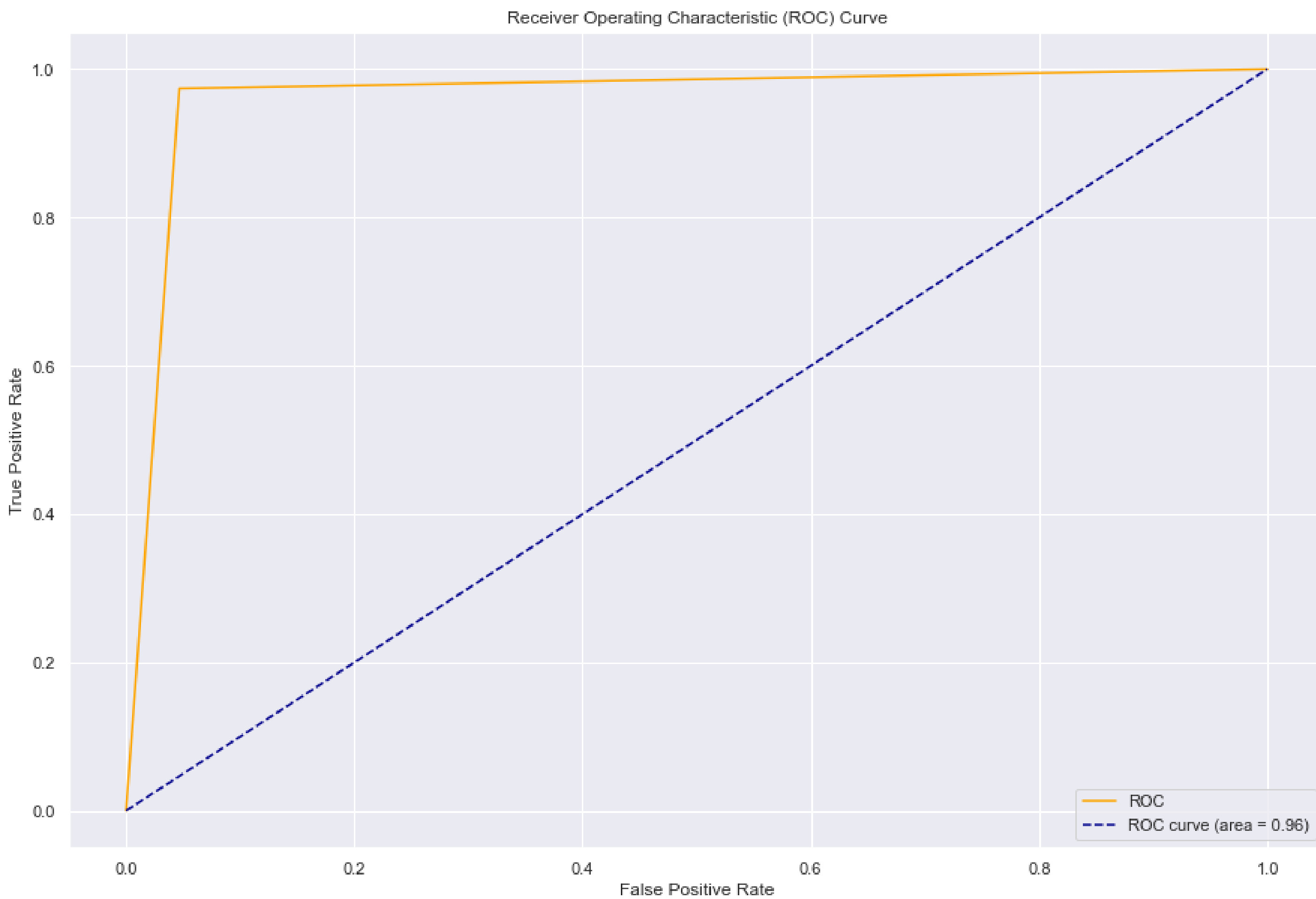
A typical ROC curve looks like the following figure.



- Mathematically, it represents the various confusion matrices for various thresholds. Each black dot is one confusion matrix.
- The green dotted line represents the scenario when the true positive rate equals the false positive rate.
- As evident from the curve, as we move from the rightmost dot towards left, after a certain threshold, the false positive rate decreases.
- After some time, the false positive rate becomes zero.
- The point encircled in green is the best point as it predicts all the values correctly and keeps the False positive as a minimum.
- But that is not a rule of thumb. Based on the requirement, we need to select the point of a threshold.
- The ROC curve answers our question of which threshold to choose.

```
In [58]: fpr, tpr, thresholds = roc_curve(y_test, Logistic_Regression_Prediction)
```

```
In [65]: plt.plot(fpr, tpr, color='orange', label='ROC')
         plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve (area = %0.2f)'
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.title('Receiver Operating Characteristic (ROC) Curve')
         plt.legend()
         plt.show()
```



But we have a confusion!!

Let’s suppose that we used different classification algorithms, and different ROCs for the corresponding algorithms have been plotted.

The question is: which algorithm to choose now?

The answer is to calculate the area under each ROC curve.

Advantages & Disadvantages of Logistic Regression

Advantages of Logisitic Regression

- It is very simple and easy to implement.
- The output is more informative than other classification algorithms
- It expresses the relationship between independent and dependent variables
- Very effective with linearly seperable data

Disadvantages of Logisitic Regression

- Not effective with data which are not linearly seperable
- Not as powerful as other classification models
- Multiclass classifications are much easier to do with other algorithms than logisitic regression
- It can only predict categorical outcomes

Saving the Model

In [60]:

```
import pickle
# Saving the model file
with open( 'modelForPrediction.sav', 'wb') as f:
    pickle.dump(logistic_regression,f)
```