ForestFire Prediction Model

Project Objective

To predict the chances of forest fires based on a given set of conditions on the data given to us.

Project Scope

- **Requirements** :-
  - **Dataset** - Required a classification type of dataset for previous years forest fires at specific regions.
  - **Python Libraries** - Especially important data science libraries like numpy, pandas, matplotlib, scikit learn, seaborn for analysis and model creation of the project.
  - **IDE / Code Editor** - Preferably Jupyter Notebook but any other editor/IDE can be used.
- **GOALS :-**

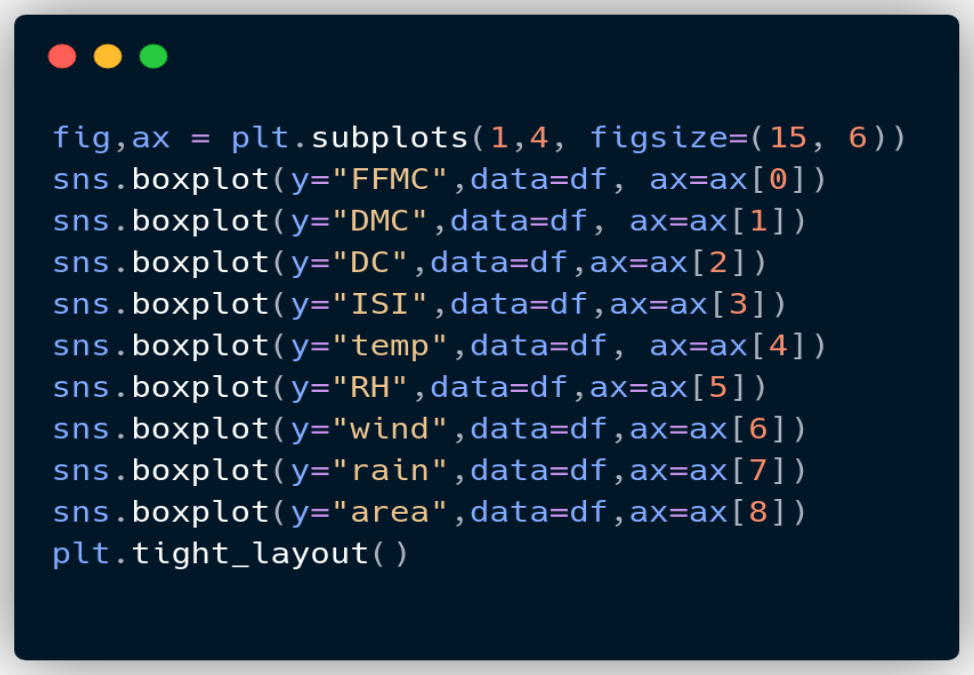
  The goal of this project is to predict whether there will be a forest fire or not and by what chances based on given set of data conditions.
- **Limitations :-**

  As of now, due to the availability of classification dataset calculating chances is difficult but will be done with complex calculations
- **Deliverables :-**
  - **Documentation -** This in-depth documentation will be provided for reference and usage of the product.
  - **Code -** The actual code of the model will be given to implement in other applications.
- **Milestone Planning :-**
  - EXPLORATORY DATA ANALYSIS
  - MODEL CREATION
  - MODEL EVALUATION

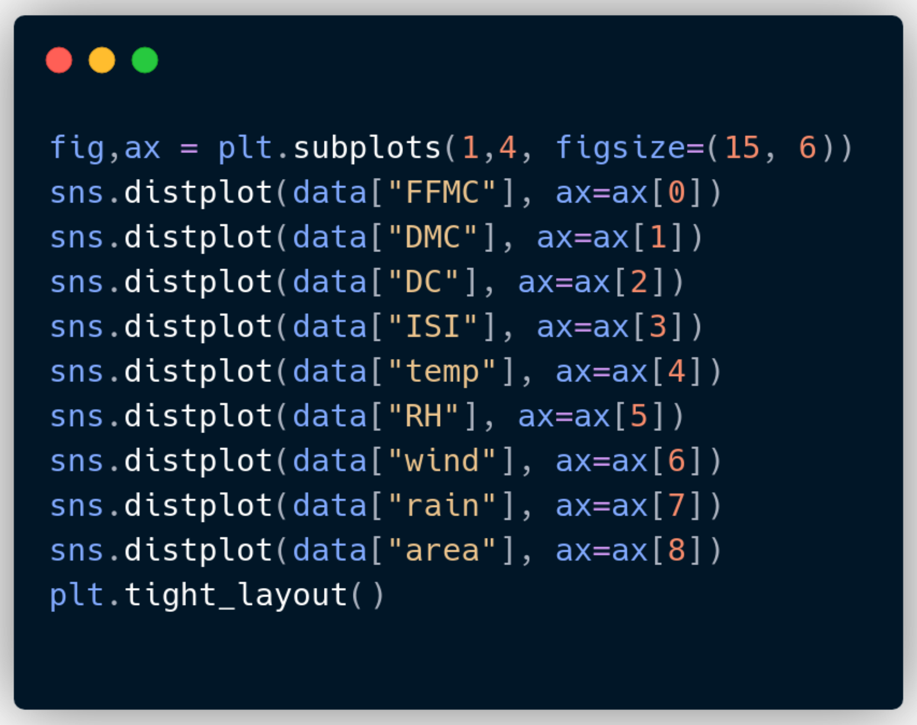
Exploratory Data Analysis

- **Univariate, Bivariate & Multivariate Analysis**

  In this part the analysis of all the variables is done. The variables are analysed in respect to their correlation with the other variables. We need to predict which variable is the independent variable and dependent variable. For independent variable, there are multiple columns. Thus this analysis will be a multivariate analysis. In the given data set columns **FFMC, DMC, DC, ISI, temp, RH, wind, rain & Area.** These variables fall under multivariate analysis. For dependent variable there has to be a new column viz. **Fire** which will be a categorical variable and analysis will be of bivariate analysis with all the independent variables and one dependent variable.

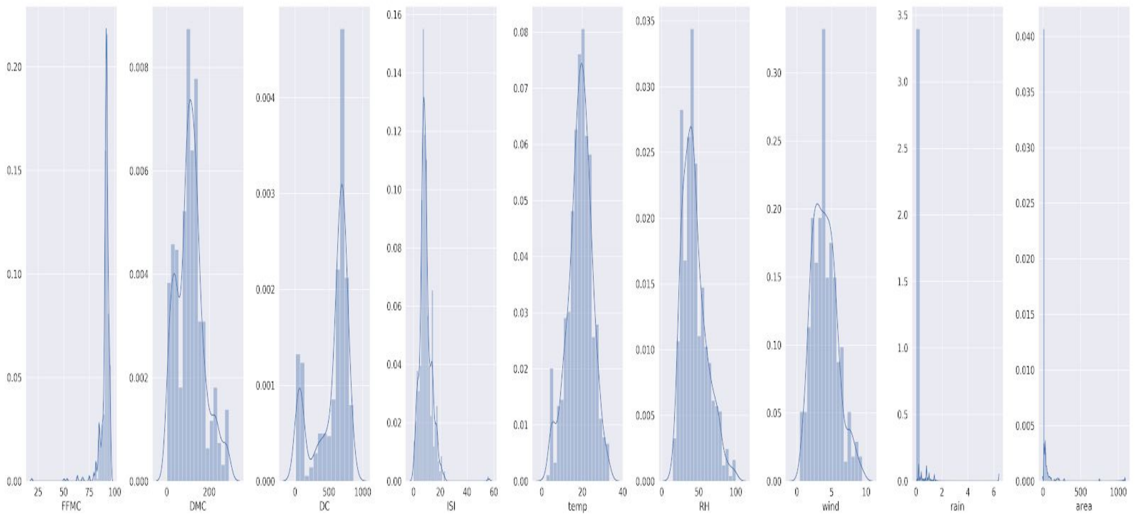


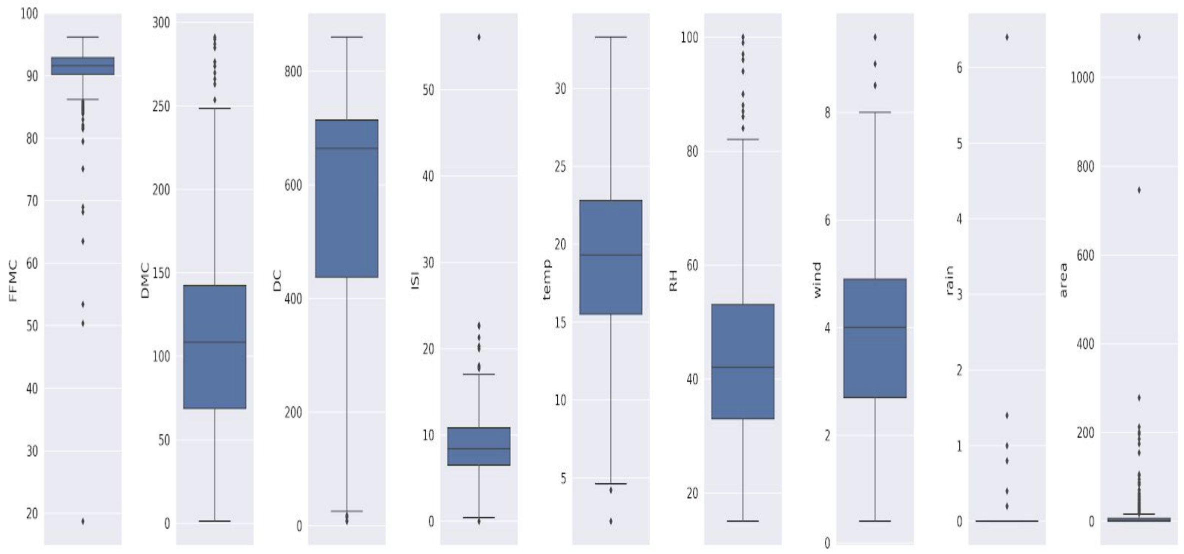
```python
fig,ax = plt.subplots(1,4, figsize=(15, 6))
sns.boxplot(y="FFMC",data=df, ax=ax[0])
sns.boxplot(y="DMC",data=df, ax=ax[1])
sns.boxplot(y="DC",data=df,ax=ax[2])
sns.boxplot(y="ISI",data=df,ax=ax[3])
sns.boxplot(y="temp",data=df, ax=ax[4])
sns.boxplot(y="RH",data=df,ax=ax[5])
sns.boxplot(y="wind",data=df,ax=ax[6])
sns.boxplot(y="rain",data=df,ax=ax[7])
sns.boxplot(y="area",data=df,ax=ax[8])
plt.tight_layout()
```



```python
fig,ax = plt.subplots(1,4, figsize=(15, 6))
sns.distplot(data["FFMC"], ax=ax[0])
sns.distplot(data["DMC"], ax=ax[1])
sns.distplot(data["DC"], ax=ax[2])
sns.distplot(data["ISI"], ax=ax[3])
sns.distplot(data["temp"], ax=ax[4])
sns.distplot(data["RH"], ax=ax[5])
sns.distplot(data["wind"], ax=ax[6])
sns.distplot(data["rain"], ax=ax[7])
sns.distplot(data["area"], ax=ax[8])
plt.tight_layout()
```

The box plots,distribution plots and histograms for the various columns are seen below which further consolidate the above stated analysis.



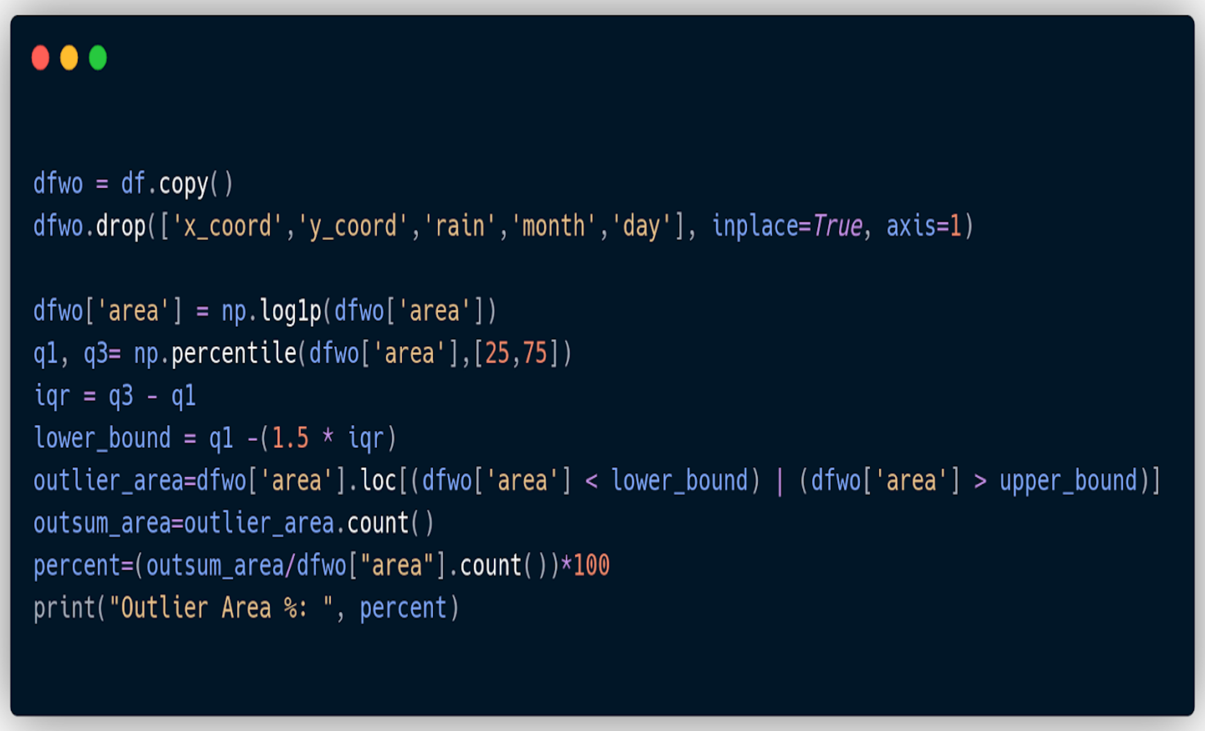


• Missing value Treatment

In the dataset given, all missing values are eliminated and thus no missing value treatment is required. All the data doesn’t contain any **NaN** values and thus the data can be passed on to outlier treatment phase.

• Outlier Treatment

All the independent variables have outliers that are needed to be removed for fine tuning of the model.The outlier values are replaced with the mean of the column.This reduces the percentage of outliers. For the area column, the **logarithmic transformation** is used to remove to remove all outliers. For the other columns it is seen that the percentages of outliers are very small in number therefore they are left as it is.

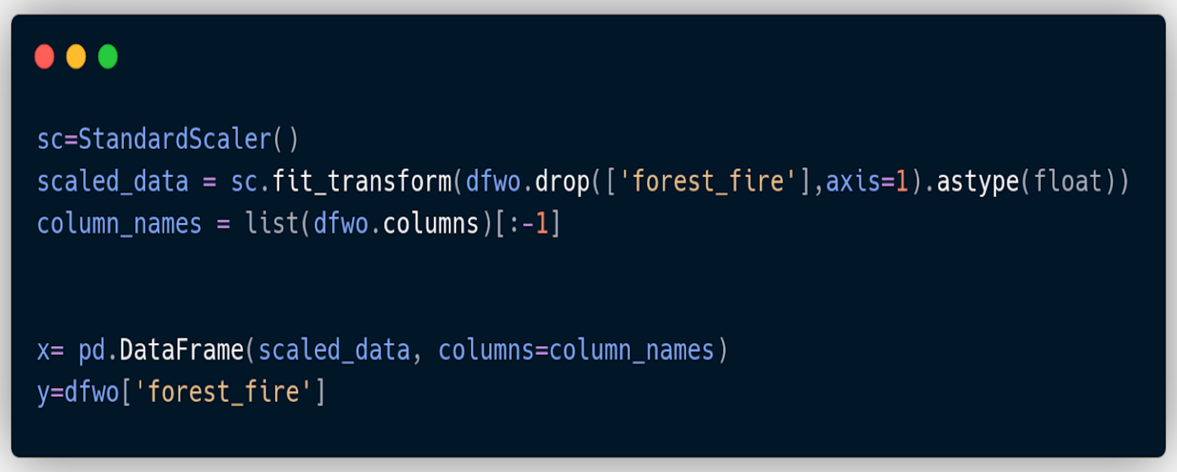


```
dfwo = df.copy()
dfwo.drop(['x_coord','y_coord','rain','month','day'], inplace=True, axis=1)

dfwo['area'] = np.log1p(dfwo['area'])
q1, q3= np.percentile(dfwo['area'],[25,75])
iqr = q3 - q1
lower_bound = q1 -(1.5 * iqr)
outlier_area=dfwo['area'].loc[(dfwo['area'] < lower_bound) | (dfwo['area'] > upper_bound)]
outsum_area=outlier_area.count()
percent=(outsum_area/dfwo["area"].count())*100
print("Outlier Area %: ", percent)
```

- **Variable transformation**

  The variables as seen in the dataset all contain values relating to different weights or scales therefore to effectively correlate among them the columns are scaled using the StandardScaler method from the **Sklearn's** preprocessing library.

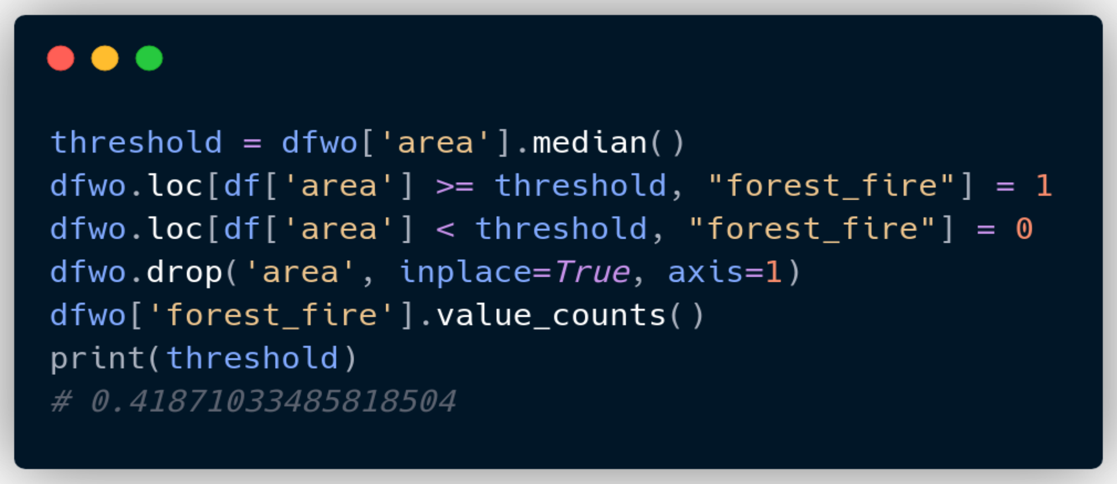


```python
sc=StandardScaler()
scaled_data = sc.fit_transform(dfwo.drop(['forest_fire'],axis=1).astype(float))
column_names = list(dfwo.columns)[:-1]


x= pd.DataFrame(scaled_data, columns=column_names)
y=dfwo['forest_fire']
```

- **Variable creation**

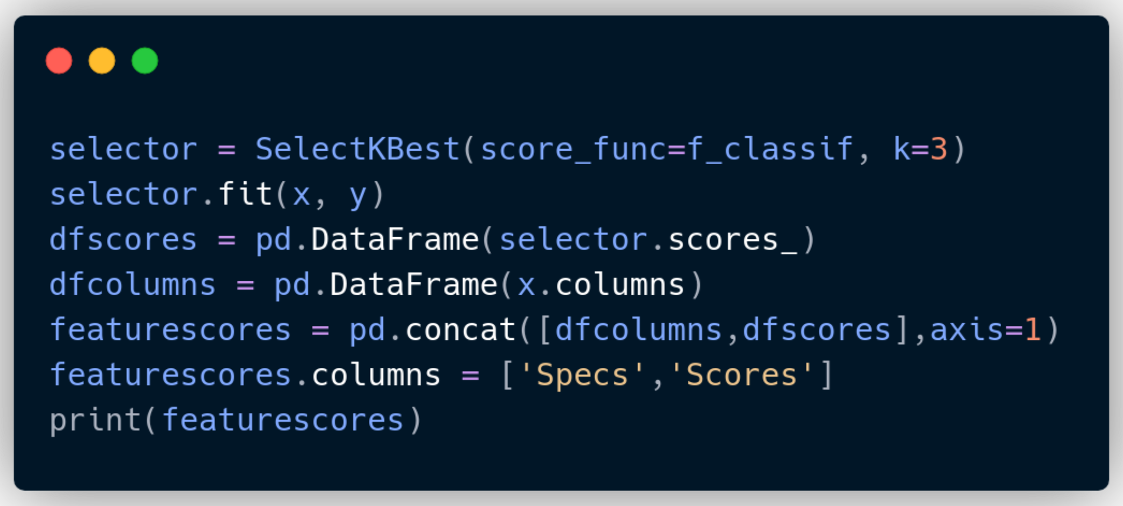
  A new variable is created called “**forest_fire**” this contains the categorical type data which has been calculated from the area variable.The mean of the area column has been used as a threshold after transforming the column with logarithmic values and values above the mean have been selected as ones and below the mean as zeroes.These values have then been fitted to the newly created forest_fire column.



```python
threshold = dfwo['area'].median()
dfwo.loc[df['area'] >= threshold, "forest_fire"] = 1
dfwo.loc[df['area'] < threshold, "forest_fire"] = 0
dfwo.drop('area', inplace=True, axis=1)
dfwo['forest_fire'].value_counts()
print(threshold)
# 0.41871033485818504
```

- ## Selection of input and output features

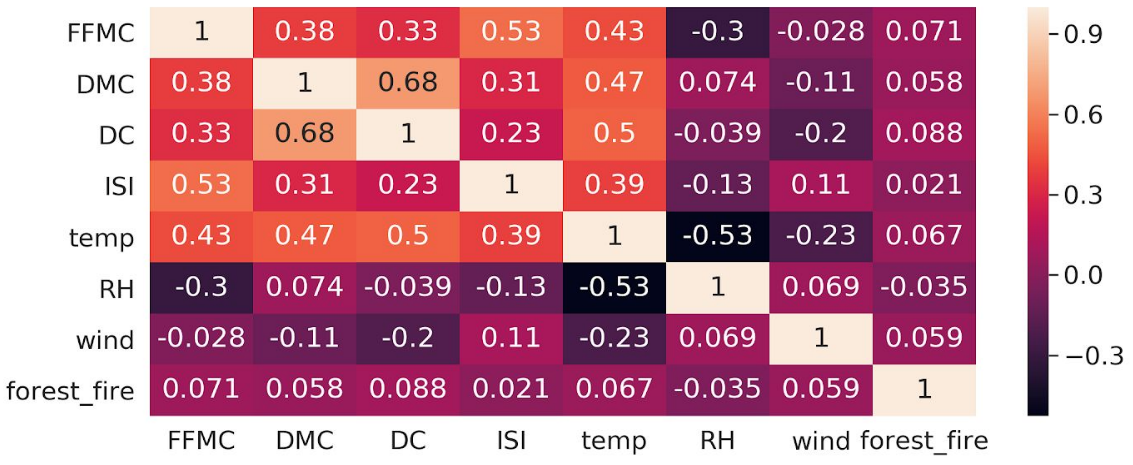
The forest_fire column is selected as the output feature as it has been created with the purpose of telling us whether the forest fire has occurred or not.



```python
selector = SelectKBest(score_func=f_classif, k=3)
selector.fit(x, y)
dfscores = pd.DataFrame(selector.scores_)
dfcolumns = pd.DataFrame(x.columns)
featurescores = pd.concat([dfcolumns,dfscores],axis=1)
featurescores.columns = ['Specs','Scores']
print(featurescores)
```

The input features are ideally selected based off of the score of the **KBased selection** method and the **heatmap** .Therefore in our case except the x coordinate y coordinate and month all other features are selected as input features .

The **heatmap** of the input and output features is seen below

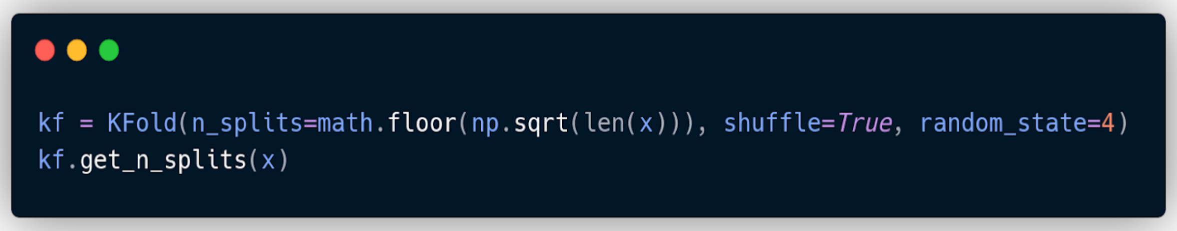


	FFMC	DMC	DC	ISI	temp	RH	wind	forest_fire
FFMC	1	0.38	0.33	0.53	0.43	-0.3	-0.028	0.071
DMC	0.38	1	0.68	0.31	0.47	0.074	-0.11	0.058
DC	0.33	0.68	1	0.23	0.5	-0.039	-0.2	0.088
ISI	0.53	0.31	0.23	1	0.39	-0.13	0.11	0.021
temp	0.43	0.47	0.5	0.39	1	-0.53	-0.23	0.067
RH	-0.3	0.074	-0.039	-0.13	-0.53	1	0.069	-0.035
wind	-0.028	-0.11	-0.2	0.11	-0.23	0.069	1	0.059
forest_fire	0.071	0.058	0.088	0.021	0.067	-0.035	0.059	1

Model Creation

Before the creation of the various models the training set and testing sets are prepared for the input and output features.

To do this the **KFold Cross Validation** is used. To decide the number of splits the square root of the length of the input features is used. The split data are then stored into variables x_train, x_test, y_test, and y_test.



```
kf = KFold(n_splits=math.floor(np.sqrt(len(x))), shuffle=True, random_state=4)
kf.get_n_splits(x)
```

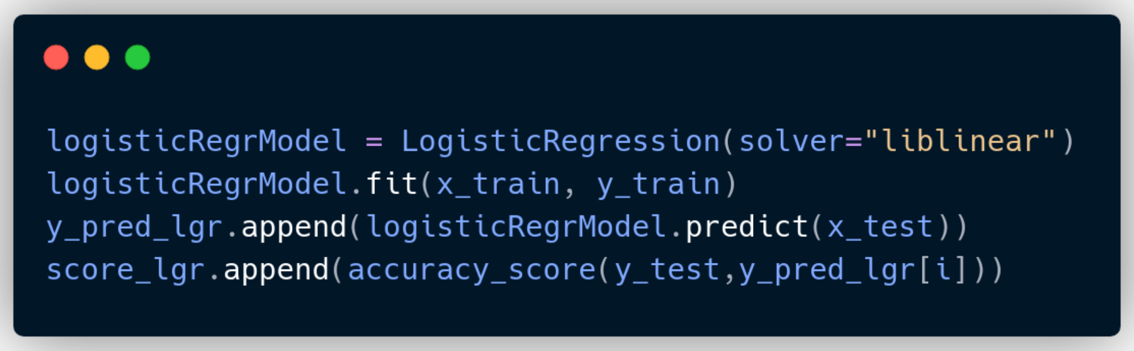
• Logistic regression

Our first model uses logistic regression to predict whether a certain set of weather conditions will give rise to a forest fire or not .

This uses the **LogisticRegression()** method of the sklearn library.

The LogisticRegression.fit() method is passed with arguments x_train and y_train and this accounts for the model training.

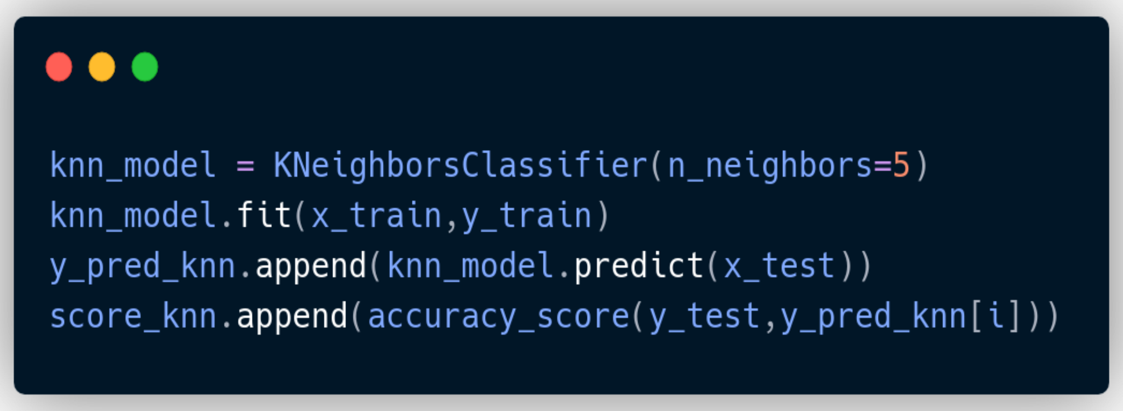
Then to test our model the LogisticRegression.predict method is passed with the x_test values and this output is stored in a variable which is then compared with the y_test values. The accuracy score of the comparison is found using the **accuracy_score()** method and the output is stored in a list .



```
logisticRegrModel = LogisticRegression(solver="liblinear")
logisticRegrModel.fit(x_train, y_train)
y_pred_lgr.append(logisticRegrModel.predict(x_test))
score_lgr.append(accuracy_score(y_test,y_pred_lgr[i]))
```

• KNN model

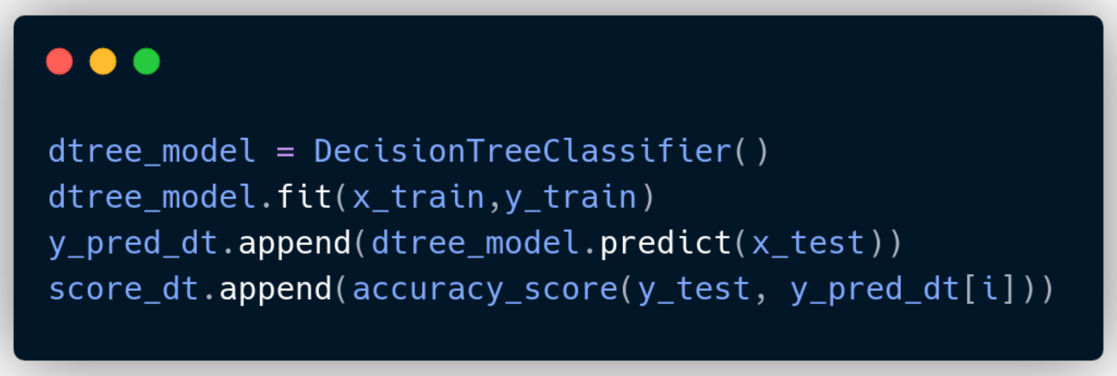
The knn model or the k nearest neighbour model is a model that is solely based on the **distance** of the concerned point from the k-nearest neighbours to it.
It is usually used for classification based problems and is hence very useful for us.In our case we have taken k=5 which means that the 5 nearest points to our test point will be examined to assign a class to our test point.
This model works on the training and testing set created using the method stated above and has yielded an accuracy of **83%**.
This uses the **KNeighborsClassifier()** method from the sklearn library in python.



```
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(x_train,y_train)
y_pred_knn.append(knn_model.predict(x_test))
score_knn.append(accuracy_score(y_test,y_pred_knn[i]))
```

• Decision Tree model

The **decision tree** is a tool that works on a tree like graph for its decisions and its possible consequences. We have used decision tree to predict the outcome value of the forest_fire column. It did a very good job in predicting the outcome with an accuracy score of **73.9%** at a given point of time. This value however is changed on every run. From sklearn library the **DecisionTreeClassifier()** is being taken in our project. The train and test method is done by KFold method as stated above. The max accurate score is kept for confusion matrix.

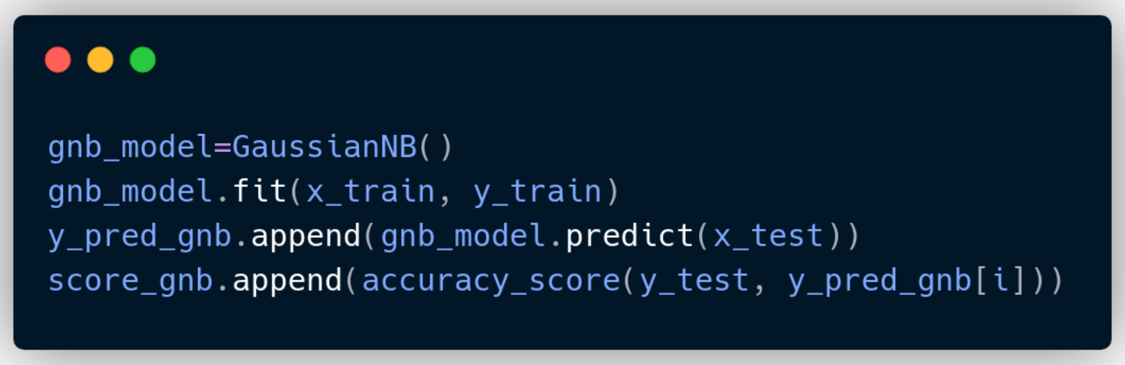


```
dtree_model = DecisionTreeClassifier()
dtree_model.fit(x_train,y_train)
y_pred_dt.append(dtree_model.predict(x_test))
score_dt.append(accuracy_score(y_test, y_pred_dt[i]))
```

- **Gaussian Naive Bayes model**

This model works best for binary and multiclass classification and since our's is a binary model it works good. This model improves upon the naive bayes theorem in which calculations are simplified and the assumption is made that the various features do not interact with each other.The gaussian part signifies working on real valued attributes assuming a gaussian distribution.

This model comes out with an accuracy score of **79.16%** and does so using the **GaussianNB()** method of the sklearn library



```
gnb_model=GaussianNB()
gnb_model.fit(x_train, y_train)
y_pred_gnb.append(gnb_model.predict(x_test))
score_gnb.append(accuracy_score(y_test, y_pred_gnb[i]))
```

Evaluation

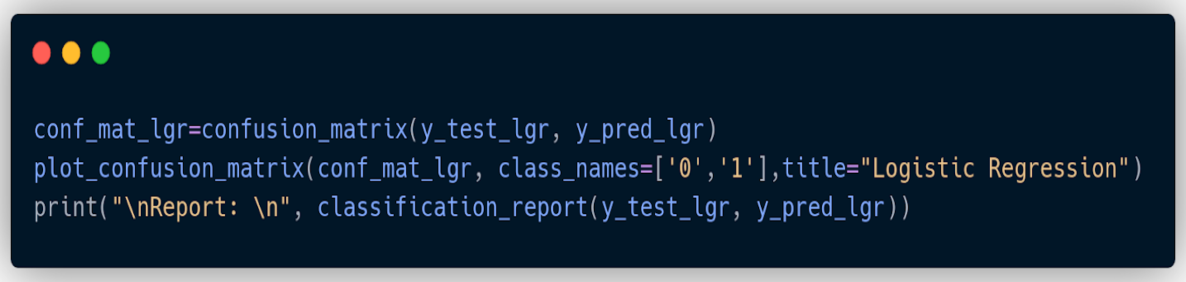
The models created so far have only been checked for their accuracy but a more in-depth analysis into their performance is required and this is discussed in this section

Generally for classification based problems the tool used for evaluation of the model is a confusion matrix and here we have created that

Precision & Recall

- **Logistic regression model**

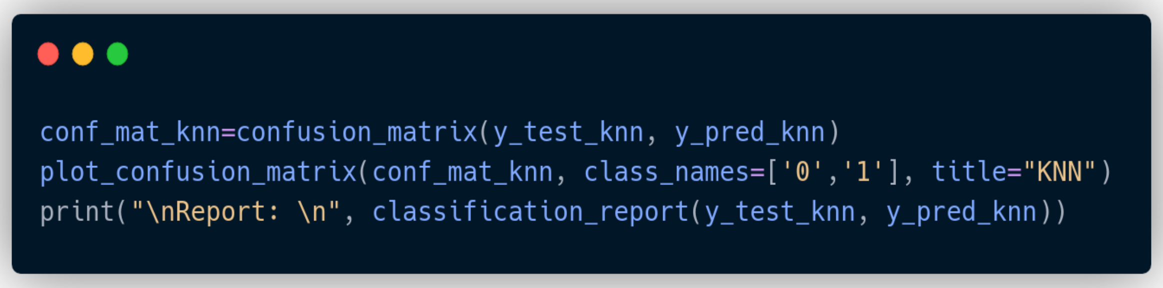
For logistic regression it is seen that when ‘0’ is considered to be positive the precision and recall both are **0.5** which means that the number of false positives and false negatives are same but when the classes are flipped the precision and recall both shoot up to **0.73** . Therefore it can be said that the model is fairly good judging by the recall and precision



```
conf_mat_lgr=confusion_matrix(y_test_lgr, y_pred_lgr)
plot_confusion_matrix(conf_mat_lgr, class_names=['0','1'],title="Logistic Regression")
print("\nReport: \n", classification_report(y_test_lgr, y_pred_lgr))
```

- ## Knn model

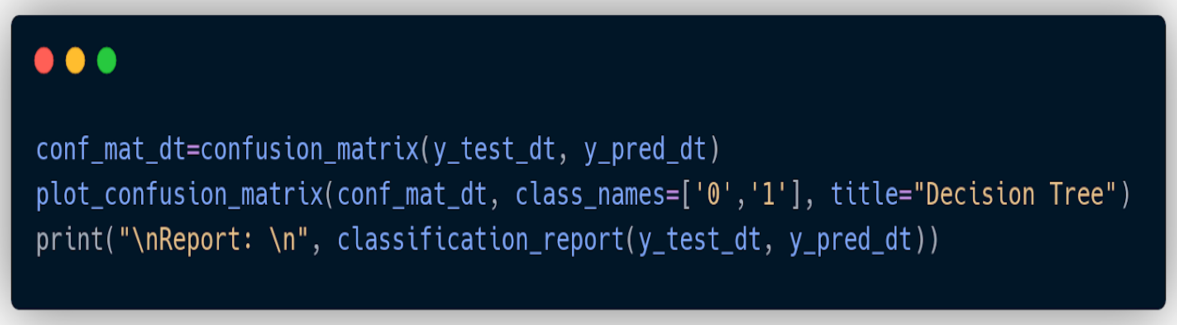
For the knn model when ‘0’ is considered to be positive the precision and recall both are 0.85 and 0.82 when the classes are flipped, thus it can be seen from the recall and precision values that this model performs better than the previous model.



```
conf_mat_knn=confusion_matrix(y_test_knn, y_pred_knn)
plot_confusion_matrix(conf_mat_knn, class_names=['0','1'], title="KNN")
print("\nReport: \n", classification_report(y_test_knn, y_pred_knn))
```

• Decision tree model

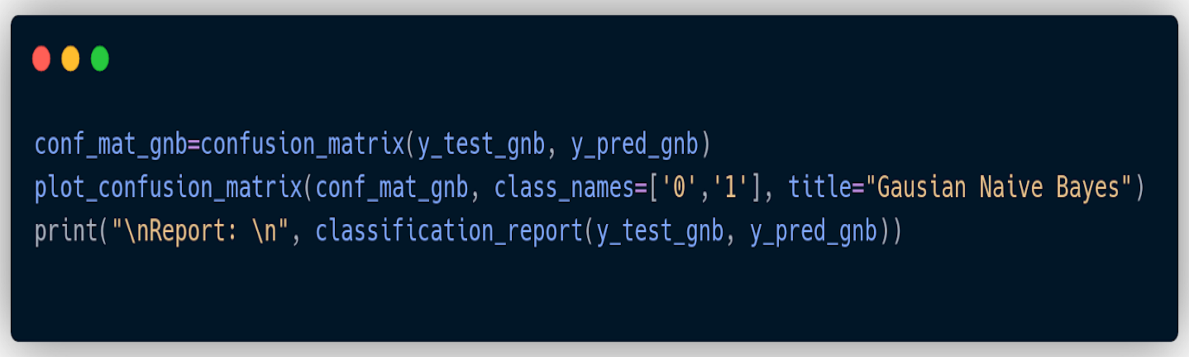
For this model when ‘0’ is taken as positive the precision and recall values are seen as 0.8 and 0.92 respectively and when the classes are reversed the values change to 0.89 to 0.73 respectively thus in this model we can say that the number of false positives and false negatives differ and in the first case when 0 is taken as positive the number of false positives exceed the number of false negatives and exactly opposite with the classes reversed.



```
conf_mat_dt=confusion_matrix(y_test_dt, y_pred_dt)
plot_confusion_matrix(conf_mat_dt, class_names=['0','1'], title="Decision Tree")
print("\nReport: \n", classification_report(y_test_dt, y_pred_dt))
```

• Gaussian naive bayes model

For this model when '0' is taken as positive it is seen that the precision is **0.83** and the recall is **0.56** and **0.78** and **0.93** respectively with the classes reversed as seen in the previous case in the decision tree model the number of false positives and the number of false negatives are not equal, when 0 is positive the number of false positives is more than the number of false negatives and exactly opposite in case of 1.



```
conf_mat_gnb=confusion_matrix(y_test_gnb, y_pred_gnb)
plot_confusion_matrix(conf_mat_gnb, class_names=['0','1'], title="Gausian Naive Bayes")
print("\nReport: \n", classification_report(y_test_gnb, y_pred_gnb))
```

A further inference from the confusion matrix is seen in the inference section.

Ensemble learning

The various models we had created till now were isolated learners and hence are assumed to be weak but here we create strong learners using ensemble learning using the following techniques

- ### Voting classifier

  This uses all the models to categorise the test data points and the prediction we get from the majority of the data points is used as the final prediction
  In our project we have used the **VotingClassifier()** method of the sklearn library to implement this and have displayed the mean vote out of the many instances of the models created on different test and train sets which comes out to be 54.85 percent

- ### Random forest classifier

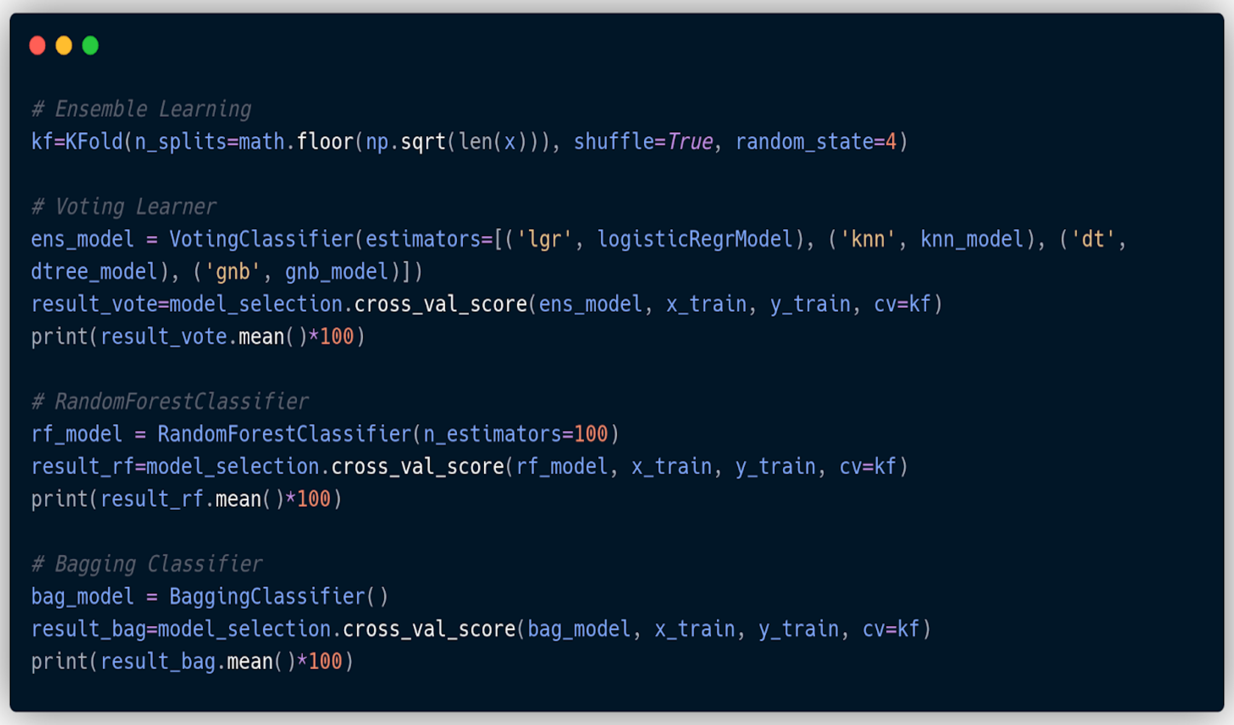
  Here multiple uncorrelated decision trees work together to classify a data point,that is the class with the maximum number of votes from the trees is the one chosen.

In our project this is done using the **RandomForestClassifier()** method of the sklearn library.100 decision trees are created by the classifiers on the seven input features and the classification score is calculated for the various test and train sets which comes out to be **54.86%**

- ## Bagging classifier

Bagging also called bootstrap aggregation works by aggregating the predictions of a model on different random subsets of the data to produce a final prediction.

In our project we have used the **BaggingClasifier()** selected the model to be a decision tree and have created hundred such trees based on different random subsets of our data and calculated the average percentage of the score of our predictions which comes out at **56.46%**.



```
# Ensemble Learning
kf=KFold(n_splits=math.floor(np.sqrt(len(x))), shuffle=True, random_state=4)

# Voting Learner
ens_model = VotingClassifier(estimators=[('lgr', logisticRegrModel), ('knn', knn_model), ('dt',
dtree_model), ('gnb', gnb_model)])
result_vote=model_selection.cross_val_score(ens_model, x_train, y_train, cv=kf)
print(result_vote.mean()*100)

# RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=100)
result_rf=model_selection.cross_val_score(rf_model, x_train, y_train, cv=kf)
print(result_rf.mean()*100)

# Bagging Classifier
bag_model = BaggingClassifier()
result_bag=model_selection.cross_val_score(bag_model, x_train, y_train, cv=kf)
print(result_bag.mean()*100)
```

Inference

From this project, starting from EDA and analysing the data carefully we have learned that forest fire depends on not all features but specific features selected by us after thorough evaluation.

The columns **FFMC, DMC, DC, ISI, temp, RH, wind, rain & Area** are the most important columns in the dataset. After analysis, we have also found that the following dataset is a regression type dataset specifically designed to solve regression type problems.

We need to convert this dataset into a classification type of dataset and that's what we did. We by taking the area column as our main threshold predictor we created a column which will be a classification column named “**forest_fire**”.

The values of this column is set based on the median of the area field. If the value of the area is higher than the median, the corresponding value of the forest_fire column will be 1. The reason for doing this is that we need some area to be burned before we call it a forest fire. Thus we have decided on this logic.
Next up after creating the forest_fire we marked it as **“y”.**

The other unnecessary columns are dropped like X_coord, y_coord,day,date,month, and area.

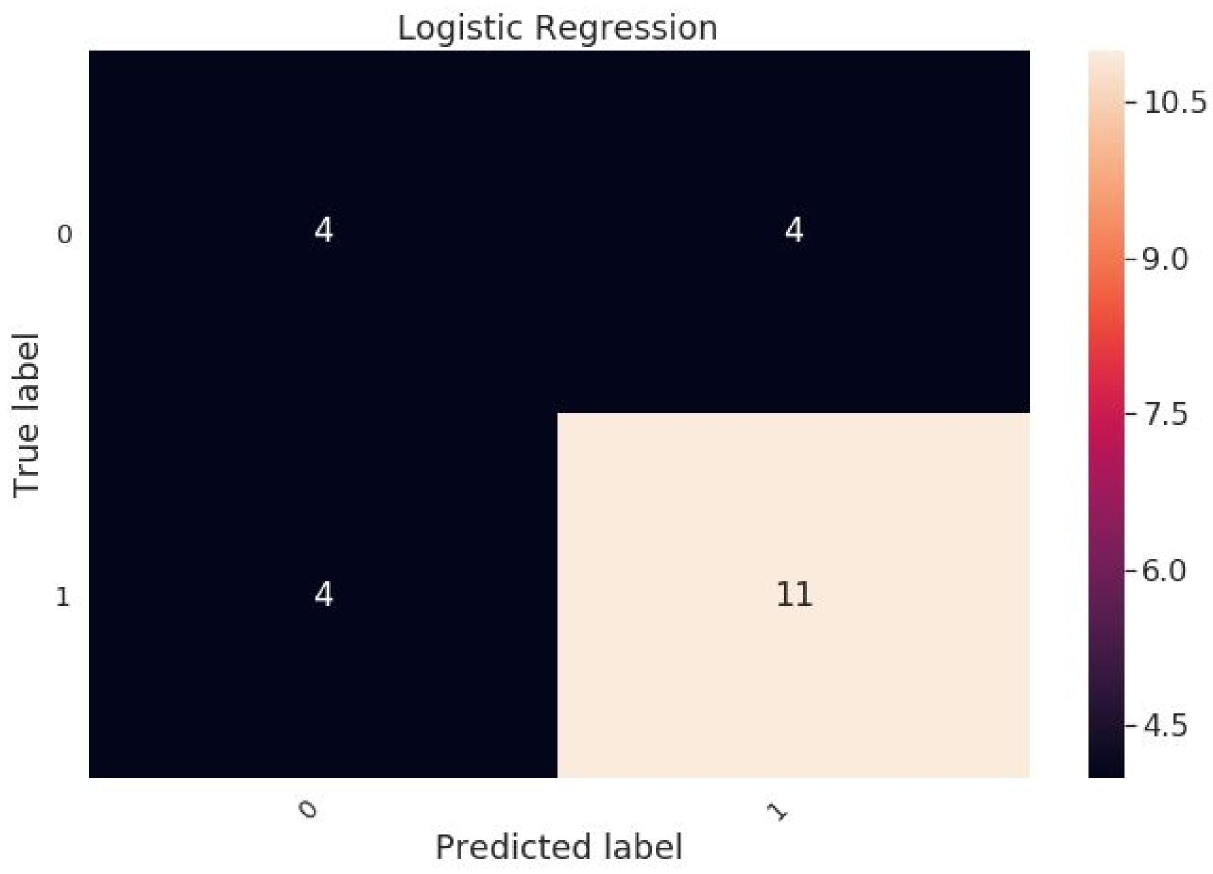
The features are then scaled to make everything on the same range. This is necessary because the weight of the values in each column is different than one another. This might confuse the model resulting in false predictions.

The next thing we did is selecting best features using the **KBased Feature selection** method. The kbased feature gives scores to each feature which helps us decide which feature is most relevant to the model and which is not. This makes sure only the relevant features goes in the model.

However our dataset doesn’t contain that many features and thus we have passed all the features into the model for prediction. SInce too little feature might result into less data feeding to the model and reducing in performance of the model. This concludes our EDA and the data is now ready to be fed in the model. The next step and also the first step in model creation is to split the data into train and test splits.

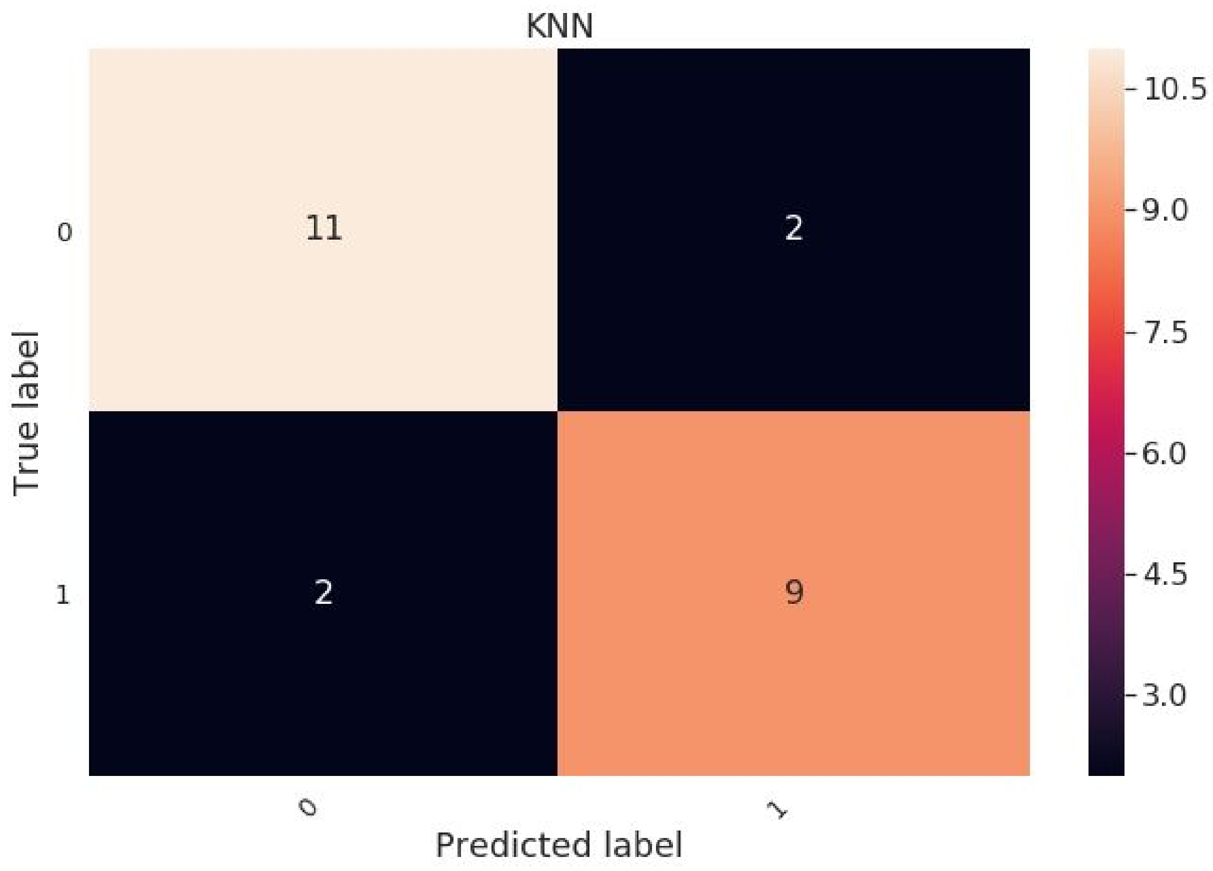
However splitting with hardcoding values not a good practice and thus we have used another technique for this is known as the **KFold Cross - Validation.** This method splits the data in multiple iteration which trains the model very well. The no. of iterations is based on the square root of the data which comes out to be about **22**.

Then the first model we create is the logistic regression model. This model presents us with a **S-Curve** ranging from 0 to 1. This method gave us an accuracy of about **65%**. The following confusion matrix is plotted.



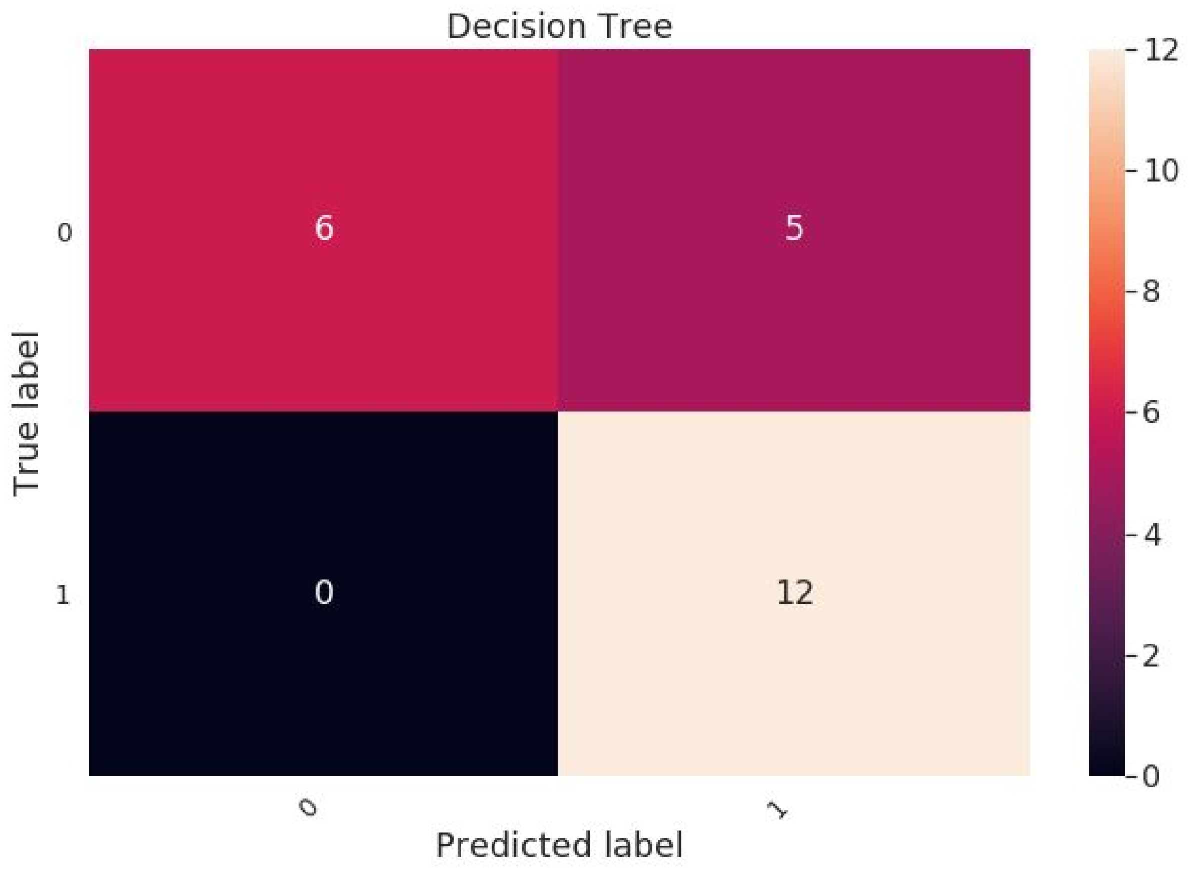
As we can see the values of TP and TN are 4 and 11 respectively. The model is not that accurate as we get 8 false values but still its not bad.

The next model we created is the KNN model which is based on the distance of the concerned datapoint. This model did very well than the logistic model and gave an accuracy of **83.34%**. The following confusion matrix is plotted:



From above we can interpret that the TP and TN values of the model are 11 and 9. This model has very low false predictions and only 4 predictions are wrongly predicted out of 24.

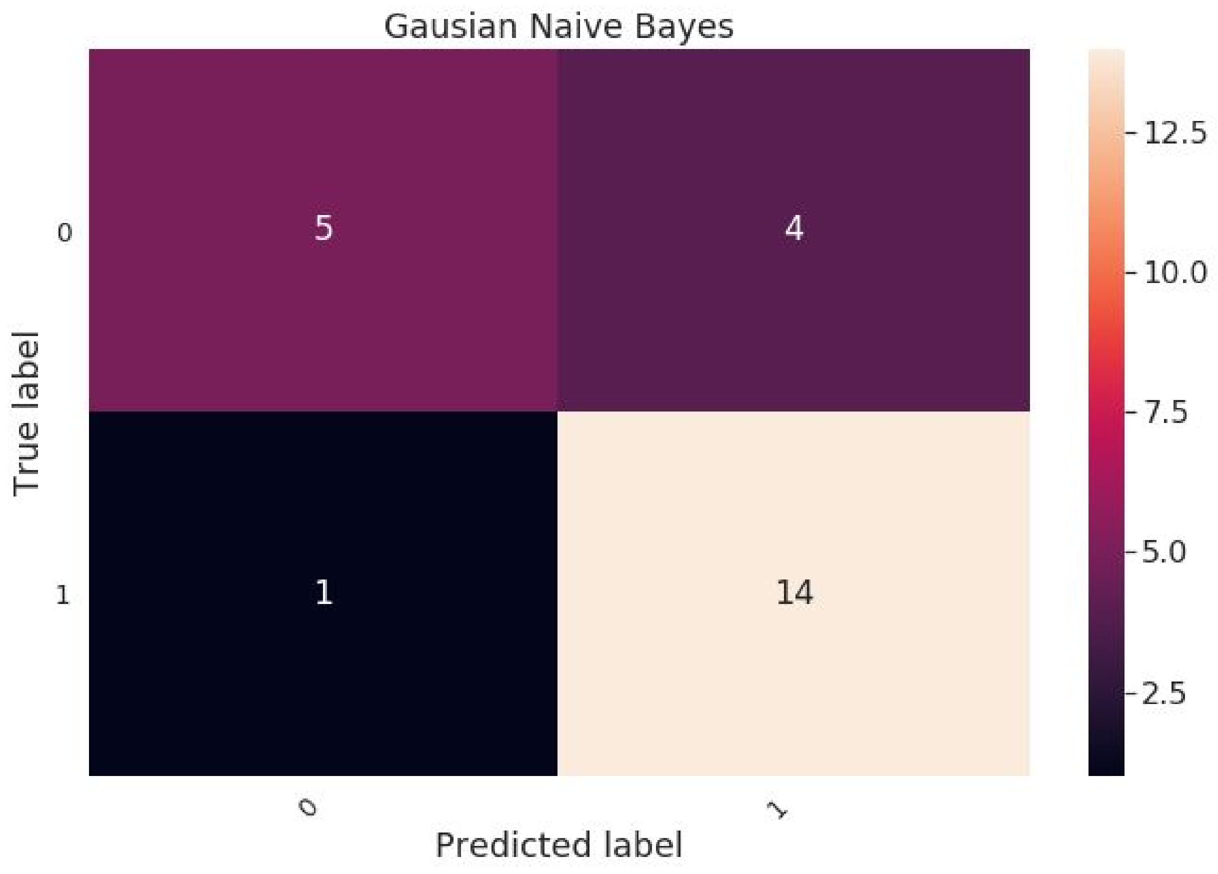
The next model is decision tree which is a bit tricky one. This model did well enough than logistic model but failed to surpass the KNN model. The accuracy of this model is around **73%** and the following confusion matrix is deducted.



From the above image we can analyse that this model also has very less false values. Only 5 values are falsely predicted and TP and TN values are 6 and 12 respectively.

The FInal Model that we have created is the Gaussian Naive Bayes model. This model did surpass the logistic and decision tree but couldn't win against the KNN model. The accuracy of this model came to be around **79%.**

The following confusion matrix is plotted



As from the above matrix, this matrix did well in predicting the truthy values. The values of TP and TN are 5 and 14 respectively. The falsey values came to be 5 out of 24 which is not bad.

After evaluation of all these models we did create one more model which is known as **Ensemble Model.** These models are known to be as strong learners compared to the single individual learners. The three methods for ensemble model are used namely VotingClassifier, RandomForestClassifier and BaggingClassifier. These models predicted with an accuracy score of 55.47%, 57.07% and 56.69% respectively.

The Final conclusion is based on that the model is not that good but it is still better in predicting forest fires in some areas based on the given conditions.

Future Scope

The model can be improved in many areas like

- it can be used to predict the percentage of chances of having a forest fire.
- The model can also be improved on other datasets for prediction.
- The model can move from sklearn to *Tensorflow* for more complex calculations and accurate predictions.

Acknowledgement

I and my team would like to express my special thanks of gratitude to my teacher ***Kaushik Ghosh*** as well as *Globsyn Finishing School* who gave me a golden opportunity to do this wonderful project on the topic Forest Fire Prediction Model, which also helped me and my team in doing a lot of research and we came to know about so many new things. We are really thankful to them.
Secondly we would also like to everyone who helped us a lot in finalizing this project within the limited time frame.

Reference

- Kaggle - For the Dataset
- Medium - For Helpful articles.
- StackOverFlow - For answers to our questions
- Sklearn - For all ML libraries

Project Members: -

Ayan Banerjee
Saswata Chatterjee
Debojotee Dutta
Sudipra Biswas
Chitradip Chakraborty
Amit Kushwaha

Thank You.