Predict the Burned area of Forest Fires

Group No: 10

Participants List:

1) Sri Lalana
2) Parsis Presswala
3) Trapti Khandelwal

Problem Statement: Forest fire causes serious damage to the Flora and fauna of a country. This is one of major environmental concerns. The aim of this project is to predict the burned area of forest fires, in the northeast region of Portugal, by using meteorological and other data.

Description and Analysis of the solutions provided:

Firstly, we started with including the basic libraries such as:

```
import pandas as pd

from sklearn import linear_model

import matplotlib. pyplot as plt
```

And then we read the csv file where data is stored by creating a dataframe as:

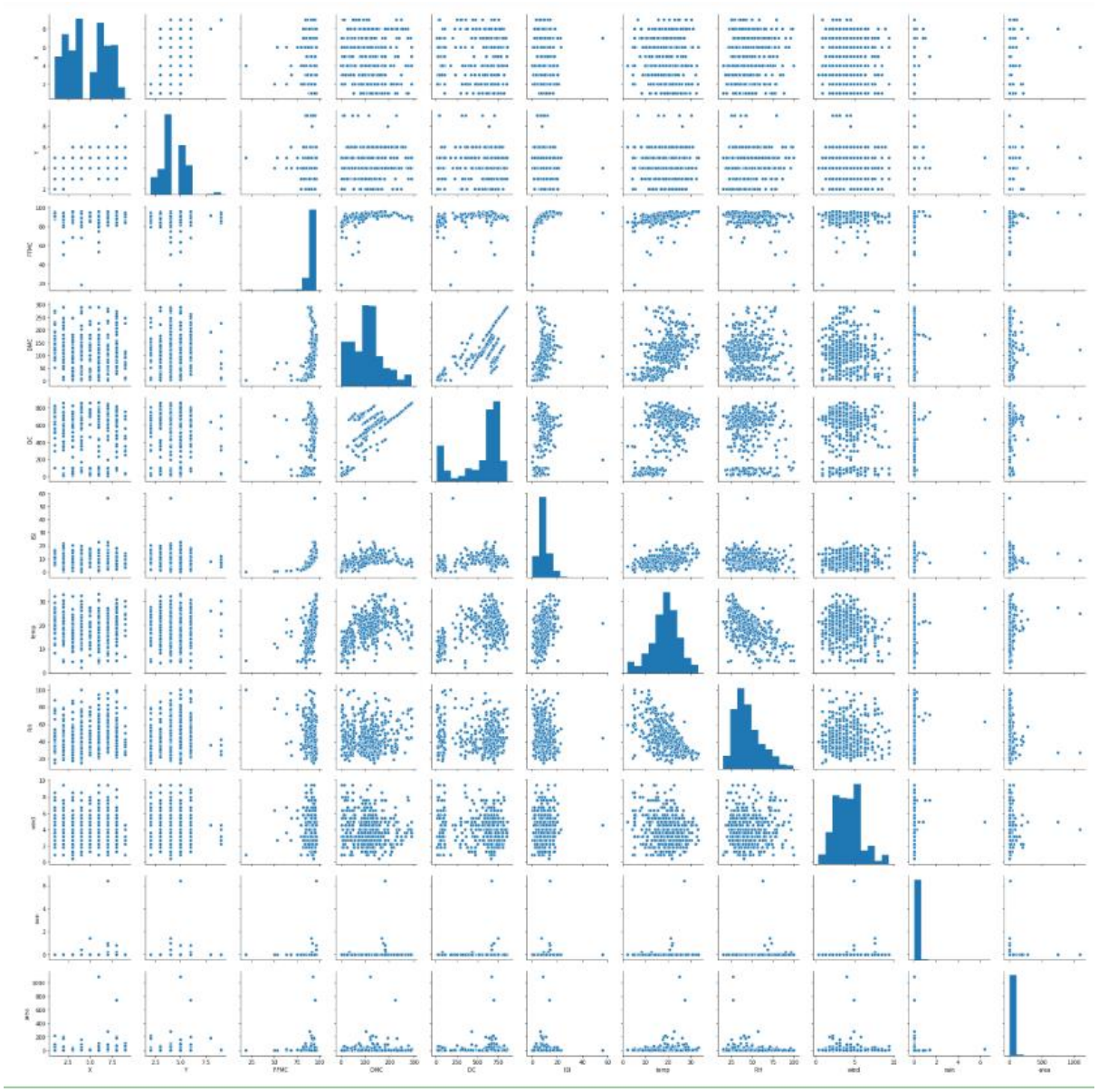
```
df=pd. read_csv('forestfires.csv')

df. head ()
```

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0

For any regression problem as soon as we get the data we need to check for assumptions of linear regression.

1)Linearity**:** The relationship between the independent and dependent variables should be linear. The linearity assumption can best be tested with scatter plots.

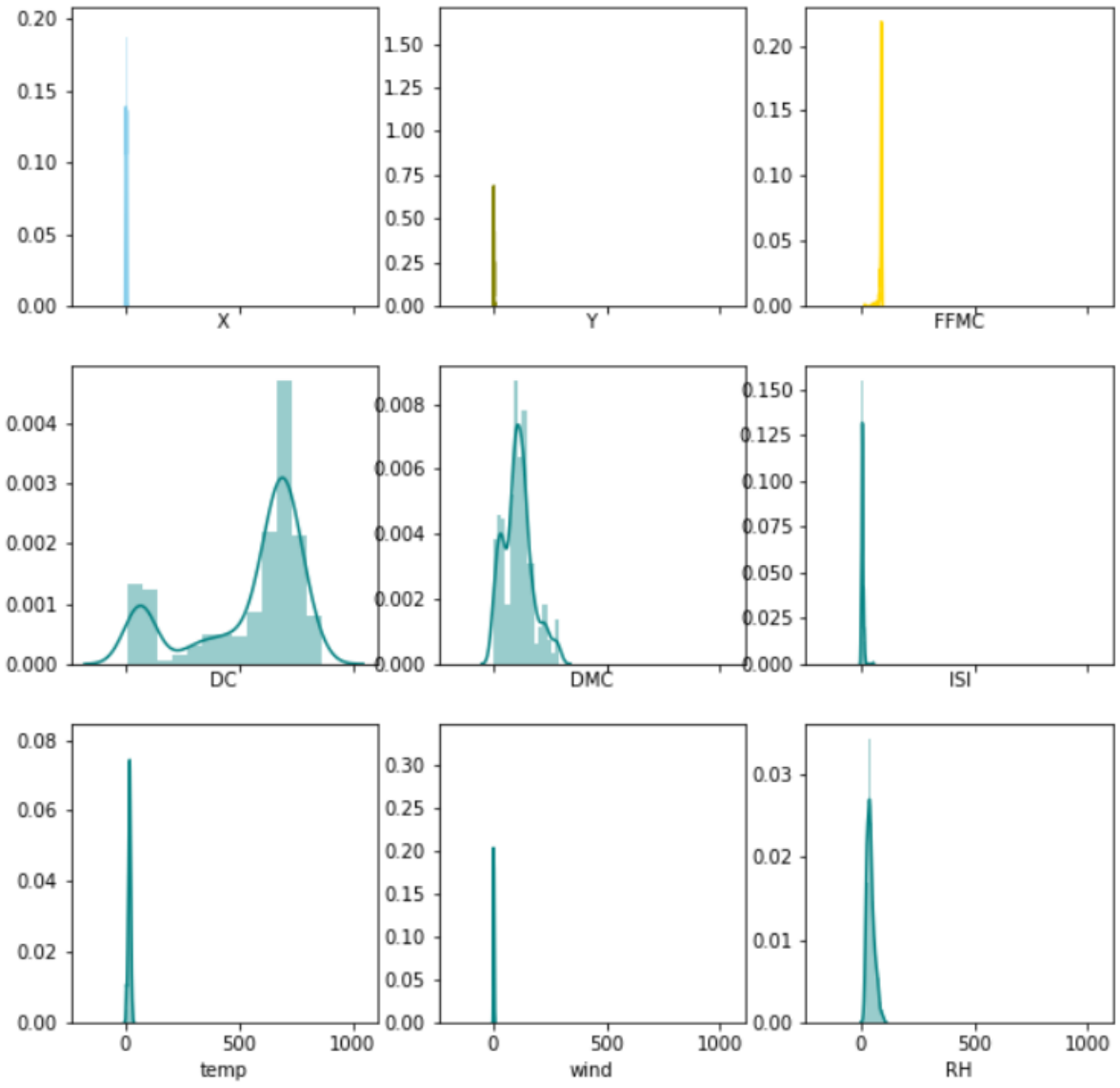
```
import seaborn as sns

sns.pairplot(df)
```



Conclusion: From the above Scatter plot we can see that the data has no linear relationship between the variables. So, the first assumption of the linearity is failed.

2)Normality: The linear regression analysis requires all variables to be multivariate normal. This assumption can best be checked with a histogram or a Q-Q-Plot.

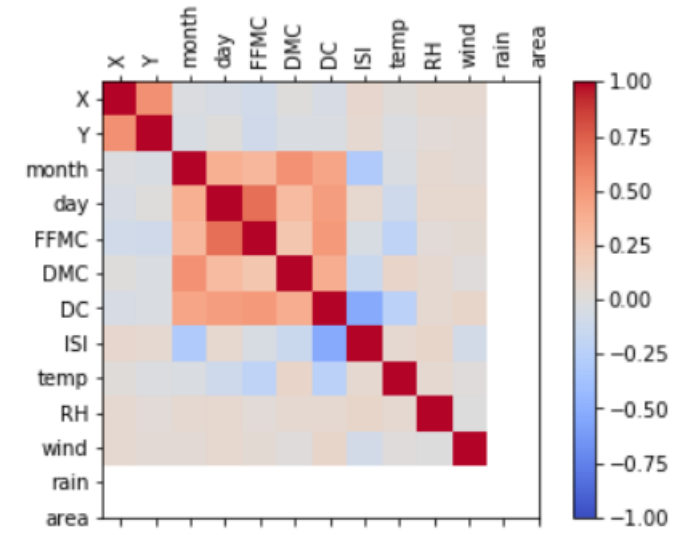
Histogram:



Conclusion: We can see in the above histograms that almost all are in the shape of leptokurtic and few distributions have multimodal. Inorder to satisfy the normality condition, shape of the distribution should be symmetric and peakedness should be mesokurtic. Clearly none of the above distributions are satisfying these conditions. So, the data failed in normality assumption.

3)Multicollinearity: Linear regression assumes that there is little or no multicollinearity in the data. It can be tested through VIF, Correlation matrix.

Heat Map:



Correlation Matrix:

	X	Y	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
X	1.000000	0.539548	-0.021039	-0.048384	-0.085916	0.006210	-0.051258	0.085223	0.018798	0.065387	0.063385
Y	0.539548	1.000000	-0.046308	0.007782	-0.101178	-0.024488	-0.024103	0.062221	-0.020341	0.033234	0.044873
FFMC	-0.021039	-0.046308	1.000000	0.382619	0.330512	0.531805	0.431532	-0.300995	-0.028485	0.056702	0.040122
DMC	-0.048384	0.007782	0.382619	1.000000	0.682192	0.305128	0.469594	0.073795	-0.105342	0.074790	0.072994
DC	-0.085916	-0.101178	0.330512	0.682192	1.000000	0.229154	0.496208	-0.039192	-0.203466	0.035861	0.049383
ISI	0.006210	-0.024488	0.531805	0.305128	0.229154	1.000000	0.394287	-0.132517	0.106826	0.067668	0.008258
temp	-0.051258	-0.024103	0.431532	0.469594	0.496208	0.394287	1.000000	-0.527390	-0.227116	0.069491	0.097844
RH	0.085223	0.062221	-0.300995	0.073795	-0.039192	-0.132517	-0.527390	1.000000	0.069410	0.099751	-0.075519
wind	0.018798	-0.020341	-0.028485	-0.105342	-0.203466	0.106826	-0.227116	0.069410	1.000000	0.061119	0.012317
rain	0.065387	0.033234	0.056702	0.074790	0.035861	0.067668	0.069491	0.099751	0.061119	1.000000	-0.007366
area	0.063385	0.044873	0.040122	0.072994	0.049383	0.008258	0.097844	-0.075519	0.012317	-0.007366	1.000000

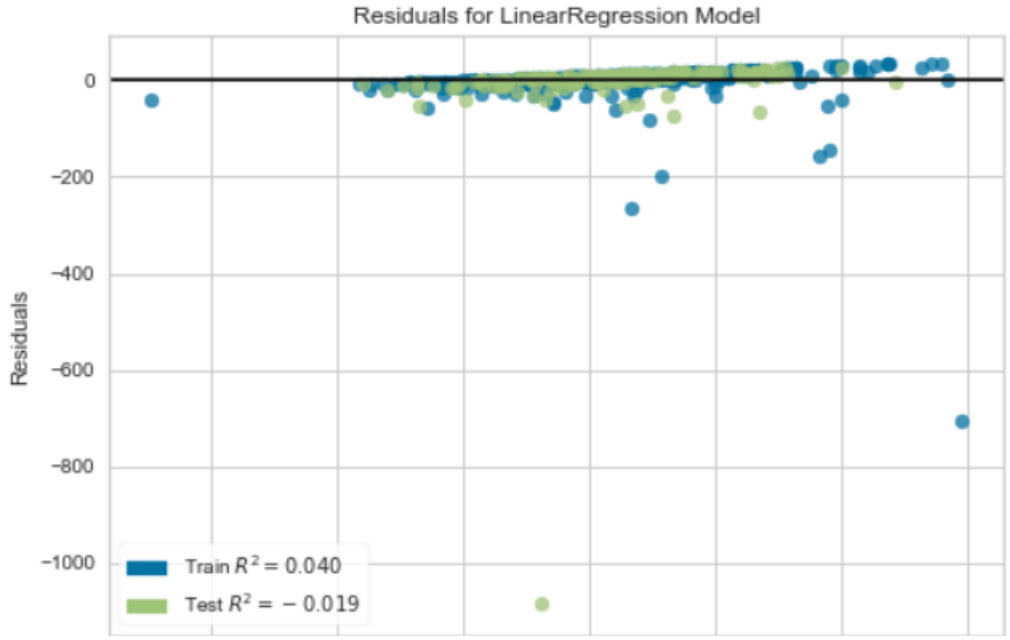
VIF:

	VIF Factor	features
0	106.0	FFMC
1	9.2	DMC
2	35.2	DC
3	7.5	ISI
4	36.3	temp
5	15.1	RH
6	7.4	wind
7	1.0	rain
8	31.6	aug
9	2.1	dec
10	3.0	feb
11	1.1	jan
12	5.6	jul
13	2.9	jun
14	6.8	mar
15	1.3	may
16	1.1	nov
17	3.2	oct
18	31.2	sep

Conclusion: With VIF > 5 there is an indication that multicollinearity may be present; with VIF > 10 there is certainly multicollinearity among the variables. We can see that there few variables that are greater than 10, centering the data the simplest way to address the problem is to remove independent variables with high VIF values

4)Homoscedasticity: We can check this assumption with Residual Plot.

```
from yellowbrick.regressor import ResidualsPlot
visualizer = ResidualsPlot(reg, hist=False)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```



Conclusion: The above figure shows the presence of heteroskedasticity. Since the variance of errors is not constant throughout the line.

Since, we do not have null values in our dataset, we are adding 10% of null values to the data.

df = df. mask (np. random. random(df.shape) < .1)

df. head (5)

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	7.000000	4.28731	NaN	fri	NaN	26.2	94.300000	5.1	8.2	51.000000	4.003965	NaN	0.0
1	4.678959	4.00000	oct	NaN	90.6	NaN	669.100000	6.7	NaN	33.000000	NaN	0.02407	0.0
2	7.000000	NaN	oct	sat	90.6	43.7	553.153037	6.7	14.6	33.000000	NaN	0.00000	0.0
3	8.000000	6.00000	oct	sat	91.7	NaN	77.500000	9.0	8.3	97.000000	4.000000	0.20000	0.0
4	8.000000	6.00000	mar	sun	89.3	NaN	102.200000	9.6	11.4	44.378855	1.800000	NaN	0.0

As we can see in that above table that there are some Nan values which we got after adding 10% of null values to our dataset.

The missing values in the dataset are filled using fillna--ffill and bfill.

```
## filling of the missing values
df['X'].fillna(df['X'].mean(),inplace=True)
df['Y'].fillna(df['Y'].mean(),inplace=True)
df['FFMC'].fillna(df['FFMC'].mean(),inplace=True)
df['DMC'].fillna(df['DMC'].mean(),inplace=True)
df['DC'].fillna(df['DC'].mean(),inplace=True)
df['ISI'].fillna(df['ISI'].mean(),inplace=True)
df['temp'].fillna(df['temp'].mean(),inplace=True)
df['RH'].fillna(df['RH'].mean(),inplace=True)
df['wind'].fillna(df['wind'].mean(),inplace=True)
df['rain'].fillna(df['rain'].mean(),inplace=True)
df['area'].fillna(df['area'].mean(),inplace=True)
```

```
df.fillna(method='ffill',inplace=True)
df.fillna(method='bfill',inplace=True)
```

And for the ‘month’ column we have categorical values which we converted into numeric using get_dummies i.e, one hot encoding technique.

We have also eliminated the columns like ‘X’,’Y’ and ‘day’ as they are not providing any significance to our analysis.

	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area	aug	dec	feb	jan	jul	jun	mar	may	nov	oct	sep
0	90.84994	26.200000	94.300000	5.1	8.200000	51.000000	4.003965	0.024337	0.0	0	0	0	0	0	0	0	0	0	1	0
1	90.60000	110.986215	669.100000	6.7	18.764602	33.000000	4.050999	0.024070	0.0	0	0	0	0	0	0	0	0	0	1	0
2	90.60000	43.700000	553.153037	6.7	14.600000	33.000000	4.050999	0.000000	0.0	0	0	0	0	0	0	0	0	0	1	0
3	91.70000	110.986215	77.500000	9.0	8.300000	97.000000	4.000000	0.200000	0.0	0	0	0	0	0	0	0	0	0	1	0
4	89.30000	110.986215	102.200000	9.6	11.400000	44.378855	1.800000	0.024337	0.0	0	0	0	0	0	0	1	0	0	0	0

The above presented data is the final dataset after cleaning.

Now we have taken all the independent variables into one dataframe named as X and the dependent column i.e, area into another dataframe as y.

```
X=df3.loc[:, df3.columns != 'area']
```

```
y=df3['area']
```

Training and Splitting of the data into 70:30 ratio:

```
from sklearn. model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

Even though linearity assumptions are not satisfied we are just trying to check R score for this type of data using linear regression.

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
# create a regressor object
reg = LinearRegression()


seed = 10

test_size = 0.3
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = test_size, random_state = seed)
reg.fit(X_train,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Finding of RMSE and RScore of train and test data.

```
from sklearn.metrics import mean_squared_error,r2_score
from math import sqrt


predictions = reg.predict(X_test)
rmse = sqrt(mean_squared_error(y_test,predictions))
print("RMSE of test data",rmse)

pred_train=reg.predict(X_train)
rmse_train= sqrt(mean_squared_error(y_train,pred_train))
print("RMSE of test data",rmse_train)




from sklearn.metrics import mean_squared_error, r2_score
r2=reg.score(X_train,y_train)
print("The R Score for test data is:",r2)
```

```
RMSE of test data 88.07951720551468
RMSE of test data 45.18894615306514
The R Score for test data is: 0.038134881482222105
```

Conclusion: The R Score value is very less, almost close to zero for linear regression which implies that this model is unable to explain the variations in the data. So, we can consider that linear regression model is inefficient in predicting the burned area of forest fire for this dataset.

Since our data is not satisfying the linearity assumptions so we can go with other regression methods like Ridge regression and lasso regression which are also called "Regularization methods" in order to reduce the overfitting by adding of little bias to our model.

Ridge

```
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score
rr = Ridge(alpha=5)
rr.fit(X_train, y_train)
pred_train_rr= rr.predict(X_train)
print("RMSE of train data:",np.sqrt(mean_squared_error(y_train,pred_train_rr)))


pred_test_rr= rr.predict(X_test)
print("RMSE of test data:",np.sqrt(mean_squared_error(y_test,pred_test_rr)))

```

```
RMSE of train data: 47.3163218402544
RMSE of test data: 87.13819665082845
```

```
print("R score of train data;",r2_score(y_train, pred_train_rr))
```

```
R score of train data; 0.035124596020331134
```

```
print("R Score of test data:",r2_score(y_test, pred_test_rr))
```

```
R Score of test data: 0.014007270500274949
```

Lasso

```
from sklearn.linear_model import Lasso
model_lasso = Lasso(alpha=5)
model_lasso.fit(X_train, y_train)
pred_train_lasso= model_lasso.predict(X_train)
print('RMSE of train data:',np.sqrt(mean_squared_error(y_train,pred_train_lasso)))


pred_test_lasso= model_lasso.predict(X_test)
print('RMSE of test data:',np.sqrt(mean_squared_error(y_test,pred_test_lasso)))

```

```
RMSE of train data: 47.52113392544899
RMSE of test data: 87.39383859971166
```

```
print('R score of train data:',r2_score(y_train, pred_train_lasso))
```

```
R score of train data: 0.02675345313020039
```

```
print('R score of test data:',r2_score(y_test, pred_test_lasso))
```

```
R score of test data: 0.008213465827488609
```

Conclusion: Even after applying the regularization techniques there no change in the R score of test data.¶

Nonlinear regression:

When the linearity assumptions are not satisfied then the non-linear regression algorithms come into picture that are able to capture the non-linearity within the data. Here we have presented some Tree based methods which comes under nonlinear regression.¶

- Random forest
- Decision Tress
- KNN (bagging)
- XGBoost
- Gradiant Boosting
- Support Vector Machine(using kernels)

In Nonlinear techniques, we have done feature scaling i.e, Standard Scalar inorder to transform the data. Even though Standard Scalar is mostly used in classification, we have used it in regression as it is showing some effect on our R score.

Random Forest usually increases predictive power of algorithm and also prevents overfitting. It is the most simply and widely used nonlinear algorithm.

Random Forest

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train= sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Import the model we are using
from sklearn.ensemble import RandomForestRegressor
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
# Train the model on training data
rf.fit(X_train, y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=1000,
                      n_jobs=None, oob_score=False, random_state=42, verbose=0,
                      warm_start=False)
```

```
predictions = rf.predict(X_test)
pred_train=rf.predict(X_train)
```

```
from sklearn.metrics import mean_squared_error
from math import sqrt

rmse = sqrt(mean_squared_error(y_test,predictions))
print("RMSE of test data",rmse)
```

```
RMSE of test data 87.73526113401071
```

```
rmse_train= sqrt(mean_squared_error(y_train,pred_train))
print("RMSE of test data",rmse_train)
```

```
RMSE of test data 19.806496060611288
```

```
r=rf.score(X_test, y_test)
print("R score of test data:",r)
```

```
R score of test data: 0.00044908061671278254
```

Conclusion: As we can see in the above obtained values that difference between RMSE of train and test data is vary vast. And the value of R Score is almost zero which indicates that this model is ineffective for predicting the burned area of the forest fire.

Decision Trees:

```
## Standarizing the variables
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
X_train = sc_x.fit_transform(X_train)
X_test = sc_x.transform(X_test)

from sklearn.tree import DecisionTreeRegressor
reg=DecisionTreeRegressor(max_depth=3,max_leaf_nodes=3)
reg.fit(X_train,y_train)




y_pred = reg.predict(X_test)
print("MSE of test data:",mean_squared_error(y_test, y_pred))
y_t = reg.predict(X_train)
print("MSE of train data:",mean_squared_error(y_train, y_t))

from sklearn.metrics import mean_squared_error
from math import sqrt

rmse = sqrt(mean_squared_error(y_test, y_pred))
print("RMSE of test data",rmse)

rs=reg.score(X_test, y_test)
print("R score of test data:",rs)
```

```
MSE of test data: 7965.008841765115
MSE of train data: 1582.131704757942
RMSE of test data 89.24689821929452
R score of test data: -0.005991013431708936
```

Conclusion: The MSE values of both test and train data are very vast and also there is a no improvement in R score value for Decision Trees. So, this is not a good method for predicting the burned area of forest fire.

Bagging--KNN

```
from sklearn.ensemble import BaggingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import RandomizedSearchCV
```

```
single_estimator = KNeighborsRegressor()
ensemble_estimator = BaggingRegressor(base_estimator = single_estimator) ### single_estimator--knn
```

```
param_dist = {'max_samples': [0.5,1.0],
              'max_features' : [0.6,1.0],
              'oob_score' : [True, False],
              'base_estimator__n_neighbors':[3,5,7],
              'n_estimators': [100]
              }
```

```
random_bag = RandomizedSearchCV(ensemble_estimator,
                                param_distributions = param_dist,
                                cv=3, n_iter = 5,
                                n_jobs=-1)
```

```
random_bag.fit(X_train, y_train)
```

```
C:\Users\DELL\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:813: DeprecationWarning: The default of the `iid` parameter will change from True to False in version 0.22 and will be removed in 0.24. This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

```
random_bag.best_params_    ### 0.5 means 50% samples
```

```
{'oob_score': True,
 'n_estimators': 100,
 'max_samples': 0.5,
 'max_features': 0.6,
 'base_estimator__n_neighbors': 7}
```

```
best_random_bag = random_bag.best_estimator_
```

```
y_pred = random_bag.predict(X_test)
y_pred_train=random_bag.predict(X_train)
from sklearn.metrics import r2_score, mean_squared_error

print ("RMSE of test data: ",np.sqrt(mean_squared_error(y_test, y_pred)))
```

```
RMSE of test data:  88.20597729189615
```

```
print ("RMSE of train data: ",np.sqrt(mean_squared_error(y_train, y_pred_train)))
print ("R-score of train data",r2_score(y_train, y_pred_train))
```

```
RMSE of train data:  44.79670302602571
R-score of train data 0.13514881483717311
```

```
print ("R-score of test data",r2_score(y_test, y_pred))
```

```
R-score of test data -0.0103052486123969
```

Conclusion: Here we have tried using KNN regression with ensemble technique called bagging and also used the Randomised Search CV method in order to get the best parameters and best estimator. Even after doing this there is no improvement in our model.

Ensemble techniques:XGBoost

```
import xgboost as xgb
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np

X=df3.loc[:, df3.columns != 'area']
y=df3['area']


##XGBoost regressor object by calling the XGBRegressor() class from the XGBoost library with the hyper-parameters passed as


xg_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.3, learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 10)


xg_reg.fit(X_train,y_train)

preds = xg_reg.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, preds))
print("RMSE of test data: %f" % (rmse))


pred_train=xg_reg.predict(X_train)
rmse_train = np.sqrt(mean_squared_error(y_train, pred_train))
print("RMSE of train data: %f" % (rmse_train))


s=xg_reg.score(X_test,y_test)
print("R Score:",s)
```

Conclusion: In XgBoost, we got R Score as -0. 00640.Here we tried using one of the ensemble techniques known as Boosting (XgBoost). We have used the objective parameter as Linear and randomly we ha ve chosen the alpha value as 10. Increasing of alpha value makes the model more conservative.

Gradiant Boosting:

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np

reg1= GradientBoostingRegressor(n_estimators=100, learning_rate=1.0, max_depth=1)
reg1.fit(X_train, y_train)
preds = reg1.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, preds))
print("RMSE of test data: %f" % (rmse))

pred_train=reg1.predict(X_train)
rmse_train = np.sqrt(mean_squared_error(y_train, pred_train))
print("RMSE of train data: %f" % (rmse_train))

s1=reg1.score(X_train,y_train)
print("R score of train data:",s1)

s=reg.score(X_test,y_test)
print("R Score of test data :",s)
```

```
RMSE of test data: 90.195706
RMSE of train data: 33.774254
R score of train data: 0.49881935183490445
R Score of test data : -0.031858486774096084
```

Conclusion: We have then tried another boosting method called Gradiant boosting, but there is no improvement in our model.

Support Vector Machine:

SVM is a supervised machine learning algorithm which can be used for classification or regression problems. It uses a technique called the kernel trick to transform your data and then based on these transformations it finds an optimal boundary between the possible outputs.

SVM

```
#feature scaling
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
X_train = sc_x.fit_transform(X_train)
X_test = sc_x.transform(X_test)


#importing svr from svm
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf',C=100) #here, we are setting the kernel to rbf which is a default kernel

#fitting the data
regressor.fit(X_train,y_train)
pred  = regressor.predict(X_test)
#checking the R score
from sklearn.metrics import r2_score
print("RScore of test data:",regressor.score(X_test,y_test))



```

```
RScore of test data: -0.02302810224396068
```

```
##### SVR with polynomial kernel
regressor = SVR(kernel = 'poly',degree=.5,C=100,gamma='auto',epsilon=1.0,coef0=1)
regressor.fit(X_train,y_train)
pred = regressor.predict(X_test)
print("RScore of test data:",regressor.score(X_test,y_test))

```

```
RScore of test data: -0.02795675849963386
```

```
######## SVR with Linear Kernel
regressor = SVR(kernel = 'linear',C=100,gamma='auto',epsilon=1.0)
regressor.fit(X_train,y_train)
pred = regressor.predict(X_test)
print("RScore of test data:",regressor.score(X_test,y_test))

```

```
RScore of test data: -0.023126894179711366
```

```
####### SVR with gamma kernel
regressor = SVR(gamma = 'auto')
regressor.fit(X_train,y_train)
pred = regressor.predict(X_test)
print("RScore of test data:",regressor.score(X_test,y_test))

```

```
RScore of test data: -0.026096080441121883
```

```
####### SVR with sigmoid
regressor = SVR(kernel = 'sigmoid',C=10,epsilon=1.0)
regressor.fit(X_train,y_train)
pred = regressor.predict(X_test)
print("RScore of test data:",regressor.score(X_test,y_test))

```

```
RScore of test data: -0.027858613754495384
```

Conclusion: In this, we have tried using different kernels such as linear, RBF, sigmoid, polynomial etc. Almost all these methods are giving us the same results.

Best model: None of the above Statistical learning methods are giving us the better results for our model. Whenever we restart the kernel there is a huge change in R Score value. From all methods we can consider Random Forest as a better method as the fluctuations in R score value is less in it.

FINAL CONCLUSION: As we can see that both linear and nonlinear techniques are not giving appropriate result for the dataset, which clearly indicates that the dataset taken is unable to fin d the patterns.

Lessons learnt from the project: From this project we have learnt that not all the problems can be solved by using Machine learning techniques. So, this can be considered as one of those situat ions where Machine learning algorithms cannot work.