Cars Class Multi Class Classification Model

IMPORTING LIBRARIES

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
```

Data Cleaning

```
data = pd.read_csv('cars_class.csv')
```

```
data.shape
```

```
(719, 20)
```

```
data.head()
```

	ID	Comp	Circ	D.Circ	Rad.Ra	Pr.Axis.Ra	Max.L.Ra	Scat.Ra	Elong	Pr.Axis.Rect	Max.L.Rect	Sc.Var.Maxis	Sc.Var.maxis	R
0	1	88	39	70	166	66	7	148	44	19	134	167	332	
1	2	85	35	64	129	57	6	116	57	17	125	138	200	
2	3	91	41	84	141	57	9	149	45	19	143	170	330	
3	4	102	54	98	177	56	10	219	31	25	171	219	706	
4	5	87	39	74	152	58	6	151	44	19	136	174	337	

```
data['Class'].unique()
```

```
array([0, 3, 1, 2])
```

```
data.isnull().sum()
```

```
ID              0
Comp            0
Circ            0
D.Circ          0
Rad.Ra          0
Pr.Axis.Ra      0
Max.L.Ra        0
Scat.Ra         0
Elong           0
Pr.Axis.Rect    0
Max.L.Rect      0
Sc.Var.Maxis    0
Sc.Var.maxis    0
Ra.Gyr          0
Skew.Maxis      0
Skew.maxis      0
Kurt.maxis      0
Kurt.Maxis      0
Holl.Ra         0
Class           0
dtype: int64
```

```
data.describe().style.background_gradient()
```

	ID	Comp	Circ	D.Circ	Rad.Ra	Pr.Axis.Ra	Max.L.Ra	Scat.Ra	Elong	Pr.Axis.Rect	Ma
count	719.000000	719.000000	719.000000	719.000000	719.000000	719.000000	719.000000	719.000000	719.000000	719.000000	71
mean	360.000000	93.435327	44.851182	81.723227	168.579972	61.847010	8.625869	168.137691	41.075104	20.531293	14
std	207.701709	8.111406	6.150286	15.528208	33.809172	8.259136	4.916908	32.937591	7.764459	2.560969	1
min	1.000000	73.000000	33.000000	40.000000	105.000000	47.000000	2.000000	112.000000	26.000000	17.000000	11
25%	180.500000	87.000000	40.000000	70.000000	141.000000	57.000000	6.000000	146.000000	33.000000	19.000000	13
50%	360.000000	93.000000	44.000000	79.000000	166.000000	61.000000	8.000000	157.000000	43.000000	20.000000	14
75%	539.500000	99.000000	49.000000	96.000000	194.500000	65.000000	10.000000	197.500000	46.000000	23.000000	15
max	719.000000	119.000000	59.000000	110.000000	333.000000	138.000000	55.000000	265.000000	61.000000	29.000000	18

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 719 entries, 0 to 718
Data columns (total 20 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   ID            719 non-null    int64
 1   Comp          719 non-null    int64
 2   Circ          719 non-null    int64
 3   D.Circ        719 non-null    int64
 4   Rad.Ra        719 non-null    int64
 5   Pr.Axis.Ra    719 non-null    int64
 6   Max.L.Ra      719 non-null    int64
 7   Scat.Ra       719 non-null    int64
 8   Elong         719 non-null    int64
 9   Pr.Axis.Rect  719 non-null    int64
 10  Max.L.Rect    719 non-null    int64
```

```
 11  Sc.Var.Maxis  719 non-null    int64
 12  Sc.Var.maxis  719 non-null    int64
 13  Ra.Gyr        719 non-null    int64
 14  Skew.Maxis    719 non-null    int64
 15  Skew.maxis    719 non-null    int64
 16  Kurt.maxis    719 non-null    int64
 17  Kurt.Maxis    719 non-null    int64
 18  Holl.Ra       719 non-null    int64
 19  Class         719 non-null    int64
dtypes: int64(20)
memory usage: 112.5 KB
```

```
data.duplicated().sum()
```

```
0
```

```
cols = data.columns
cols
```

```
Index(['ID', 'Comp', 'Circ', 'D.Circ', 'Rad.Ra', 'Pr.Axis.Ra', 'Max.L.Ra',
       'Scat.Ra', 'Elong', 'Pr.Axis.Rect', 'Max.L.Rect', 'Sc.Var.Maxis',
       'Sc.Var.maxis', 'Ra.Gyr', 'Skew.Maxis', 'Skew.maxis', 'Kurt.maxis',
       'Kurt.Maxis', 'Holl.Ra', 'Class'],
      dtype='object')
```

```
#to check different values in each column and its features

for col in cols:

    print(data[col].value_counts())
```

```
for col in cols:

    print(col,data[col].nunique())
```

```
ID 719
```

```
Comp 43
Circ 27
D.Circ 60
Rad.Ra 132
Pr.Axis.Ra 37
Max.L.Ra 21
Scat.Ra 125
Elong 35
Pr.Axis.Rect 13
Max.L.Rect 66
Sc.Var.Maxis 124
Sc.Var.maxis 380
Ra.Gyr 138
Skew.Maxis 39
Skew.maxis 23
Kurt.maxis 38
Kurt.Maxis 29
Holl.Ra 31
Class 4
```

```
for col in cols:

    print(col,data[col].unique())
```

```
Pr.Axis.Rect [19 17 25 20 18 26 24 22 23 21 27 28 29]
Max.L.Rect [134 125 143 171 136 155 129 128 168 132 140 148 149 174 145 160 172 159
 146 173 170 178 130 131 122 139 161 162 150 147 167 127 157 180 158 151
 141 154 138 133 156 186 164 163 126 153 142 144 177 119 124 121 166 176
 152 137 165 169 175 123 135 182 179 120 188 118]
Sc.Var.Maxis [167 138 170 219 174 175 148 188 258 190 146 185 180 168 230 177 204 227
 173 151 225 222 228 210 162 155 158 165 159 206 247 153 193 226 171 214
 166 234 223 181 208 169 161 189 184 172 231 272 256 136 147 197 203 211
 236 186 213 164 154 156 130 262 141 137 179 217 235 202 200 176 232 196
 199 182 265 229 160 149 218 152 163 220 139 221 157 280 209 195 140 216
 212 194 238 183 191 142 266 143 207 224 187 215 150 178 275 237 269 285
 144 134 241 240 264 135 192 145 205 267 246 287 288 320 263 254]
Sc.Var.maxis [ 332  200  330  706  337  381  246  419  866  428  225  382  379  370
  321  347  205  533  371  727  368  264  336  686  679  737  543  299
  274  364  284  319  289  530  731  266  426  305  713  362  596  359
  349  756  314  635  373  546  311  285  586  339  415  275  383  260
  663  318  625  732  404  374  611  346  833  352  229  668  203  322
```

```
   259  445  523  506  661  696  492  307  673  427  402  578  310  324
   240  741  184  776  221  348  197  325  712  367  631  677  711  326
   622  716  485  351  718  425  757  504  366  450  320  342  272  629
   391  666  327  870  237  253  684  258  363  671  335  524  389  583
   607  406  707  653  220  701  265  277  281  928  721  455  517  323
   472  294  212  610  520  340  418  290  526  333  308  535  722  602
   682  208  624  287  361  460  282  341  576  685  252  385  355  703
   487  262  306  301  494  334  408  511  642  680  589  249  658  375
   396  438  648  256  433  338  358  273  376  892  223  331  572  261
   270  369  312  329  279  298  534  691  608  238  545  309  669  670
   434  345  639  254  489  300  388  725  263  440  476  595  600  665
   687  562  354  315  317  465  748  644  255  296  271  657  344  479
   645  956  356  728  401  904  219  512  627  567  987  692  209  601
   518  694  365  674  471  233  681  416  241  697  709  513  574  268
   195  409  700  251  280  710  726  357  597  430  719  575  206  204
   328  650  717  855  558  278  678  196  387  372  508  469  343  393
   527  587  563  459  641  446  675  247  467  227  232  243  708  857
   230  704  766  968  623  923  605  473  458  954  295  350  640  481
   245  390  651  570  218  521  844  612  399  304  676  224  405  478
   207  688  360  250  216  417  621  729  613  683  291  429  667  638
   435  616  413  242  463  982  211  720  316  559  480  752  525  378
   598  636  474  838  466  579  966  422  735  557  505  519  222 1018
   698  414]
Ra.Gyr [143 123 158 223 140 172 112 136 245 148 150 184 169 134 226 190 138 182
 189 201 176 186 220 214 213 146 162 188 120 137 205 209 127 171 202 132
 173 149 181 174 191 179 175 155 121 145 219 218 200 253 199 128 139 152
 183 230 216 217 130 195 177 192 204 164 126 193 119 246 178 142 151 159
 185 153 187 247 168 133 180 157 234 135 124 198 211 224 197 239 154 141
 222 160 129 167 229 170 206 144 210 156 196 240 163 166 242 235 212 194
 161 203 236 250 125 215 244 117 165 260 114 221 208 228 131 147 232 113
 231 261 262 249 118 115 109 238 116 237 207 255]
Skew.Maxis [ 69  65  72  70  74  66  80  67  63  71  73  64  85  77  82  75  68  62
  81  78 118  76  89  83  86  79  87  91  84  61  90  97  59 119 127  88
  99  60 135]
Skew.maxis [ 5  1  9  8  6  3  7 10 16  4  2 15 11  0 14 12 13 21 19 17 20 18 22]
Kurt.maxis [13 23 14 17 33  4  2  3  1  5 11 12 24 22 15  6 20  0 29 28 21  8  7 10
 19 18  9 32 16 27 25 26 38 36 30 35 41 31]
Kurt.Maxis [193 196 189 186 187 184 195 199 198 197 191 182 180 183 190 179 185 181
 204 201 188 194 192 177 178 200 203 202 176]
Holl.Ra [201 203 199 196 193 200 184 202 206 191 197 187 183 198 194 185 189 195
 190 210 209 186 204 207 188 208 205 192 182 211 181]
```

```
Class [0 3 1 2]
```

Data splitting

```
x = data.iloc[:,1:-1]
y = data.loc[:,'Class']
```

```
print(x.shape)
print(y.shape)
```

```
(719, 18)
(719,)
```

```
x.head()
```

	Comp	Circ	D.Circ	Rad.Ra	Pr.Axis.Ra	Max.L.Ra	Scat.Ra	Elong	Pr.Axis.Rect	Max.L.Rect	Sc.Var.Maxis	Sc.Var.maxis	Ra.Gy
0	88	39	70	166	66	7	148	44	19	134	167	332	14
1	85	35	64	129	57	6	116	57	17	125	138	200	12
2	91	41	84	141	57	9	149	45	19	143	170	330	15
3	102	54	98	177	56	10	219	31	25	171	219	706	22
4	87	39	74	152	58	6	151	44	19	136	174	337	14

```
print(y.head())
print(y.unique())
```

```
0    0
1    3
2    3
3    1
4    2
Name: Class, dtype: int64
[0 3 1 2]
```

Training Data to Create a Model

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

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state = 1)
```

```
print('x_train : ', x_train.shape)
print('x_test : ', x_test.shape)
print('y_train : ', y_train.shape)
print('y_test : ', y_test.shape)
```

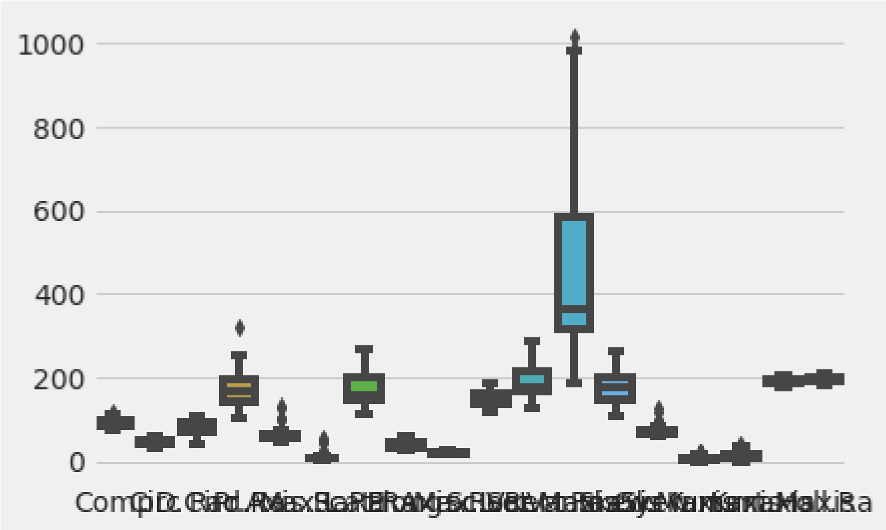
```
x_train :  (575, 18)
x_test :  (144, 18)
y_train :  (575,)
y_test :  (144,)
```

Standardizing the Model

- their is some outliers in the data
- which will cause the problem for predicting the model
- so i am using standard scaler method to reduce the outliers in the data

```
#to check the outliers

sns.boxplot(data = x_train)
```

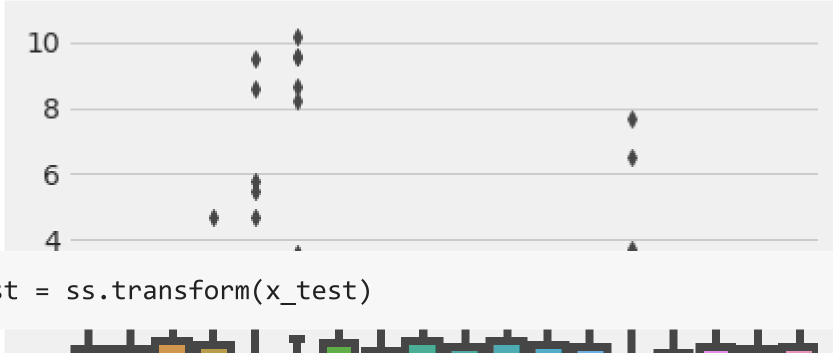
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f673a821990>
```



```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()

x_train = ss.fit_transform(x_train)
```

```
#now the data is standardised
sns.boxplot(data = x_train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f673a003750>
```




```
x_test = ss.transform(x_test)
```

Model Creation

- Apply different ML Model Creation to see the result
- from this we will select which is the best model
- to create a final model

Importing all Model Creation Libraries

```
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC
from sklearn.metrics import f1_score, plot_confusion_matrix,confusion_matrix,precision_score,recall_score,accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
```

```
logreg = LogisticRegression()
logreg.fit(x_train,y_train)
print('Train Score : ',logreg.score(x_train,y_train))
print('Test Score : ',logreg.score(x_test,y_test))
```

```
y_pred = logreg.predict(x_test)
print('F1_Score : ',f1_score(y_test,y_pred,average=None))
print('confusion_matrix : ', confusion_matrix(y_test,y_pred))
print("Precision Score : ",precision_score(y_test,y_pred,average= None))
print("Recall Score :" , recall_score(y_test, y_pred,average= None))
print('Accuracy_Score : ', accuracy_score(y_test, y_pred))
```

```
Train Score :  0.8121739130434783
Test Score :  0.7916666666666666
F1_Score :  [0.92473118 0.52830189 0.68571429 0.91666667]
confusion_matrix :  [[43  0  0  1]
 [ 3 14 11  2]
 [ 1  9 24  1]
 [ 2  0  0 33]]
Precision Score :  [0.87755102 0.60869565 0.68571429 0.89189189]
Recall Score : [0.97727273 0.46666667 0.68571429 0.94285714]
Accuracy_Score :  0.7916666666666666
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (sta
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
svc = SVC()
svc.fit(x_train,y_train)
print('Train Score : ',svc.score(x_train,y_train))
print('Test Score : ',svc.score(x_test,y_test))
y_pred = svc.predict(x_test)
print('F1_Score : ',f1_score(y_test,y_pred,average=None))
print('confusion_matrix : ', confusion_matrix(y_test,y_pred))
print("Precision Score : ",precision_score(y_test,y_pred,average= None))
print("Recall Score :" , recall_score(y_test, y_pred,average= None))
print('Accuracy_Score : ', accuracy_score(y_test, y_pred))
```

```
Train Score :  0.84
Test Score :  0.7708333333333334
F1_Score :  [0.94505495 0.47272727 0.61764706 0.91891892]
confusion_matrix :  [[43  0  0  1]
 [ 2 13 12  3]
 [ 1 12 21  1]
 [ 1  0  0 34]]
Precision Score :  [0.91489362 0.52       0.63636364 0.87179487]
Recall Score : [0.97727273 0.43333333 0.6        0.97142857]
Accuracy_Score :  0.7708333333333334
```

```
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
print('Train Score : ',knn.score(x_train,y_train))
print('Test Score : ',knn.score(x_test,y_test))
y_pred = knn.predict(x_test)
print('F1_Score : ',f1_score(y_test,y_pred,average= None))
print('confusion_matrix : ', confusion_matrix(y_test,y_pred))
print("Precision Score : ",precision_score(y_test,y_pred,average= None))
print("Recall Score :" , recall_score(y_test, y_pred,average= None))
print('Accuracy_Score : ', accuracy_score(y_test, y_pred))
```

```
Train Score :  0.8434782608695652
Test Score :  0.7569444444444444
F1_Score :  [0.93617021 0.47272727 0.63013699 0.87878788]
confusion_matrix :  [[44  0  0  0]
 [ 2 13 14  1]
 [ 1 10 23  1]
 [ 3  2  1 29]]
Precision Score :  [0.88       0.52       0.60526316 0.93548387]
Recall Score : [1.         0.43333333 0.65714286 0.82857143]
Accuracy_Score :  0.7569444444444444
```

```
dtc = DecisionTreeClassifier()
dtc.fit(x_train,y_train)
print('Train Score : ',dtc.score(x_train,y_train))
print('Test Score : ',dtc.score(x_test,y_test))
```

```
y_pred = dtc.predict(x_test)
print('F1_Score : ',f1_score(y_test,y_pred,average= None))
print('confusion_matrix : ', confusion_matrix(y_test,y_pred))
print("Precision Score : ",precision_score(y_test,y_pred,average= None))
print("Recall Score :" , recall_score(y_test, y_pred,average= None))
print('Accuracy_Score : ', accuracy_score(y_test, y_pred))
```

```
Train Score :  1.0
Test Score :  0.6944444444444444
F1_Score :  [0.88172043 0.44827586 0.53125    0.79452055]
confusion_matrix :  [[41  1  0  2]
 [ 3 13 12  2]
 [ 2 11 17  5]
 [ 3  3  0 29]]
Precision Score :  [0.83673469 0.46428571 0.5862069  0.76315789]
Recall Score : [0.93181818 0.43333333 0.48571429 0.82857143]
Accuracy_Score :  0.6944444444444444
```

```
rfc = RandomForestClassifier()
rfc.fit(x_train,y_train)
print('Train Score : ',rfc.score(x_train,y_train))
print('Test Score : ',rfc.score(x_test,y_test))
y_pred = rfc.predict(x_test)
print('F1_Score : ',f1_score(y_test,y_pred,average= None))
print('confusion_matrix : ', confusion_matrix(y_test,y_pred))
print("Precision Score : ",precision_score(y_test,y_pred,average= None))
print("Recall Score :" , recall_score(y_test, y_pred,average= None))
print('Accuracy_Score : ', accuracy_score(y_test, y_pred))
```

```
Train Score :  1.0
Test Score :  0.75
F1_Score :  [0.95652174 0.44444444 0.52941176 0.91891892]
confusion_matrix :  [[44  0  0  0]
 [ 1 12 15  2]
 [ 2 12 18  3]
 [ 1  0  0 34]]
Precision Score :  [0.91666667 0.5        0.54545455 0.87179487]
```

```
Recall Score : [1.         0.4        0.51428571 0.97142857]
Accuracy_Score :  0.75
```

```
gbc = GradientBoostingClassifier()
gbc.fit(x_train,y_train)
print('Train Score : ',gbc.score(x_train,y_train))
print('Test Score : ',gbc.score(x_test,y_test))
y_pred = gbc.predict(x_test)
print('F1_Score : ',f1_score(y_test,y_pred,average= None))
print('confusion_matrix : ', confusion_matrix(y_test,y_pred))
print("Precision Score : ",precision_score(y_test,y_pred,average= None))
print("Recall Score :" , recall_score(y_test, y_pred,average= None))
print('Accuracy_Score : ', accuracy_score(y_test, y_pred))
```

```
Train Score :  0.9982608695652174
Test Score :  0.7361111111111112
F1_Score :  [0.97777778 0.40677966 0.47761194 0.94444444]
confusion_matrix :  [[44  0  0  0]
 [ 1 12 16  1]
 [ 1 16 16  2]
 [ 0  1  0 34]]
Precision Score :  [0.95652174 0.4137931  0.5        0.91891892]
Recall Score : [1.         0.4        0.45714286 0.97142857]
Accuracy_Score :  0.7361111111111112
```

```
gb = GaussianNB()
gb.fit(x_train,y_train)
print('Train Score : ',gb.score(x_train,y_train))
print('Test Score : ',gb.score(x_test,y_test))
y_pred = gb.predict(x_test)
print('F1_Score : ',f1_score(y_test,y_pred,average=None))
print('confusion_matrix : ', confusion_matrix(y_test,y_pred))
print("Precision Score : ",precision_score(y_test,y_pred,average= None))
print("Recall Score :" , recall_score(y_test, y_pred,average= None))
print('Accuracy_Score : ', accuracy_score(y_test, y_pred))
```

```
Train Score :  0.4782608695652174
Test Score :  0.4444444444444444
F1_Score :  [0.19607843 0.47619048 0.44067797 0.53913043]
```

```
confusion_matrix :  [[ 5  6  2 31]
 [ 0 15  7  8]
 [ 0 12 13 10]
 [ 2  0  2 31]]
Precision Score :  [0.71428571 0.45454545 0.54166667 0.3875    ]
Recall Score : [0.11363636 0.5        0.37142857 0.88571429]
Accuracy_Score :  0.4444444444444444
```

```
sgd = SGDClassifier()
sgd.fit(x_train,y_train)
print('Train Score : ',sgd.score(x_train,y_train))
print('Test Score : ',sgd.score(x_test,y_test))
y_pred = sgd.predict(x_test)
print('F1_Score : ',f1_score(y_test,y_pred,average=None))
print('confusion_matrix : ', confusion_matrix(y_test,y_pred))
print("Precision Score : ",precision_score(y_test,y_pred,average= None))
print("Recall Score :" , recall_score(y_test, y_pred,average= None))
print('Accuracy_Score : ', accuracy_score(y_test, y_pred))
```

```
Train Score :  0.7634782608695653
Test Score :  0.7222222222222222
F1_Score :  [0.92473118 0.44444444 0.49180328 0.90140845]
confusion_matrix :  [[43  0  0  1]
 [ 3 14 11  2]
 [ 1 18 15  1]
 [ 2  1  0 32]]
Precision Score :  [0.87755102 0.42424242 0.57692308 0.88888889]
Recall Score : [0.97727273 0.46666667 0.42857143 0.91428571]
Accuracy_Score :  0.7222222222222222
```

Out of all the Model Random Forest classifier is the best Model

- so the final model will be created on Random Forest Classifier

Hyperparameter Tuning

- will be created based on RandomForestClassifier
- to find the best tuning parameters for the model we either use Randomized or Grid Search SV

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

```
param_dist = {'max_depth':[3,6,9], 'min_samples_leaf':[3,5,7],'criterion':['gini','entropy','log_loss'],'n_estimators':[100,10]}
```

```
rscv = RandomizedSearchCV(rfc,param_distributions = param_dist,n_iter = 27,cv = 5)
rscv.fit(x_train,y_train)
print('Best Parameters : ',rscv.best_params_)
print('Best Estimator : ',rscv.best_estimator_)
print('RSCV Test Score : ',rscv.score(x_test,y_test))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py:372: FitFailedWarning:
45 fits failed out of a total of 135.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
45 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_forest.py", line 467, in fit
    for i, t in enumerate(trees)
  File "/usr/local/lib/python3.7/dist-packages/joblib/parallel.py", line 1085, in __call__
    if self.dispatch_one_batch(iterator):
  File "/usr/local/lib/python3.7/dist-packages/joblib/parallel.py", line 901, in dispatch_one_batch
    self._dispatch(tasks)
  File "/usr/local/lib/python3.7/dist-packages/joblib/parallel.py", line 819, in _dispatch
    job = self._backend.apply_async(batch, callback=cb)
  File "/usr/local/lib/python3.7/dist-packages/joblib/_parallel_backends.py", line 208, in apply_async
    result = ImmediateResult(func)
  File "/usr/local/lib/python3.7/dist-packages/joblib/_parallel_backends.py", line 597, in __init__
```

```
    self.results = batch()
  File "/usr/local/lib/python3.7/dist-packages/joblib/parallel.py", line 289, in __call__
    for func, args, kwargs in self.items]
  File "/usr/local/lib/python3.7/dist-packages/joblib/parallel.py", line 289, in <listcomp>
    for func, args, kwargs in self.items]
  File "/usr/local/lib/python3.7/dist-packages/sklearn/utils/fixes.py", line 216, in __call__
    return self.function(*args, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_forest.py", line 185, in _parallel_build_trees
    tree.fit(X, y, sample_weight=curr_sample_weight, check_input=False)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/tree/_classes.py", line 942, in fit
    X_idx_sorted=X_idx_sorted,
  File "/usr/local/lib/python3.7/dist-packages/sklearn/tree/_classes.py", line 352, in fit
    criterion = CRITERIA_CLF[self.criterion](
KeyError: 'log_loss'

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py:972: UserWarning: One or more of the test scores are
 0.73913043 0.73217391 0.73043478 0.6626087  0.65043478 0.6626087
 0.72695652 0.73217391 0.71304348 0.73565217 0.73391304 0.72173913
        nan        nan        nan        nan        nan        nan
        nan        nan        nan]
  category=UserWarning,
Best Parameters :  {'n_estimators': 100, 'min_samples_leaf': 3, 'max_depth': 6, 'criterion': 'gini'}
Best Estimator :  RandomForestClassifier(max_depth=6, min_samples_leaf=3)
RSCV Test Score :  0.7569444444444444
```

```
gs = GridSearchCV(rfc,param_grid=param_dist,cv = 5)
gs.fit(x_train,y_train)
print('Best Parameters : ',gs.best_params_)
print('Best Estimator : ',gs.best_estimator_)
print('RSCV Test Score : ',gs.score(x_test,y_test))
print('RSCV best Score : ',gs.best_score_)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py:372: FitFailedWarning:
90 fits failed out of a total of 270.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.
```

```
Below are more details about the failures:
--------------------------------------------------------------------------------
90 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_forest.py", line 467, in fit
    for i, t in enumerate(trees)
  File "/usr/local/lib/python3.7/dist-packages/joblib/parallel.py", line 1085, in __call__
    if self.dispatch_one_batch(iterator):
  File "/usr/local/lib/python3.7/dist-packages/joblib/parallel.py", line 901, in dispatch_one_batch
    self._dispatch(tasks)
  File "/usr/local/lib/python3.7/dist-packages/joblib/parallel.py", line 819, in _dispatch
    job = self._backend.apply_async(batch, callback=cb)
  File "/usr/local/lib/python3.7/dist-packages/joblib/_parallel_backends.py", line 208, in apply_async
    result = ImmediateResult(func)
  File "/usr/local/lib/python3.7/dist-packages/joblib/_parallel_backends.py", line 597, in __init__
    self.results = batch()
  File "/usr/local/lib/python3.7/dist-packages/joblib/parallel.py", line 289, in __call__
    for func, args, kwargs in self.items]
  File "/usr/local/lib/python3.7/dist-packages/joblib/parallel.py", line 289, in <listcomp>
    for func, args, kwargs in self.items]
  File "/usr/local/lib/python3.7/dist-packages/sklearn/utils/fixes.py", line 216, in __call__
    return self.function(*args, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_forest.py", line 185, in _parallel_build_trees
    tree.fit(X, y, sample_weight=curr_sample_weight, check_input=False)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/tree/_classes.py", line 942, in fit
    X_idx_sorted=X_idx_sorted,
  File "/usr/local/lib/python3.7/dist-packages/sklearn/tree/_classes.py", line 352, in fit
    criterion = CRITERIA_CLF[self.criterion](
KeyError: 'log_loss'

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py:972: UserWarning: One or more of the test scores are
 0.7426087  0.70782609 0.73391304 0.73391304 0.72347826 0.7026087
 0.74608696 0.72695652 0.73565217 0.73565217 0.72173913 0.72695652
 0.65391304 0.66086957 0.67130435 0.66782609 0.66782609 0.67652174
 0.72347826 0.72347826 0.73913043 0.72521739 0.72869565 0.71826087
 0.74086957 0.72347826 0.73913043 0.72347826 0.72347826 0.73391304
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
```

```
        nan        nan        nan        nan        nan        nan]
  category=UserWarning,
Best Parameters :  {'criterion': 'gini', 'max_depth': 9, 'min_samples_leaf': 3, 'n_estimators': 100}
Best Estimator :  RandomForestClassifier(max_depth=9, min_samples_leaf=3)
RSCV Test Score :  0.75
RSCV best Score :  0.7460869565217391
```

Visualize feature scores of the features

```
rfc = RandomForestClassifier(criterion= 'gini', max_depth=9, min_samples_leaf= 3, n_estimators= 100)
rfc.fit(x_train,y_train)
print('Train Score : ',rfc.score(x_train,y_train))
print('Test Score : ',rfc.score(x_test,y_test))
y_pred = rfc.predict(x_test)
print('F1_Score : ',f1_score(y_test,y_pred,average= None))
print('confusion_matrix : ', confusion_matrix(y_test,y_pred))
print("Precision Score : ",precision_score(y_test,y_pred,average= None))
print("Recall Score :" , recall_score(y_test, y_pred,average= None))
print('Accuracy_Score : ', accuracy_score(y_test, y_pred))
```

```
Train Score :  0.9808695652173913
Test Score :  0.7569444444444444
F1_Score :  [0.95652174 0.49122807 0.52307692 0.91891892]
confusion_matrix :  [[44  0  0  0]
 [ 1 14 13  2]
 [ 2 13 17  3]
 [ 1  0  0 34]]
Precision Score :  [0.91666667 0.51851852 0.56666667 0.87179487]
Recall Score : [1.         0.46666667 0.48571429 0.97142857]
Accuracy_Score :  0.7569444444444444
```

```
x_train.shape
```

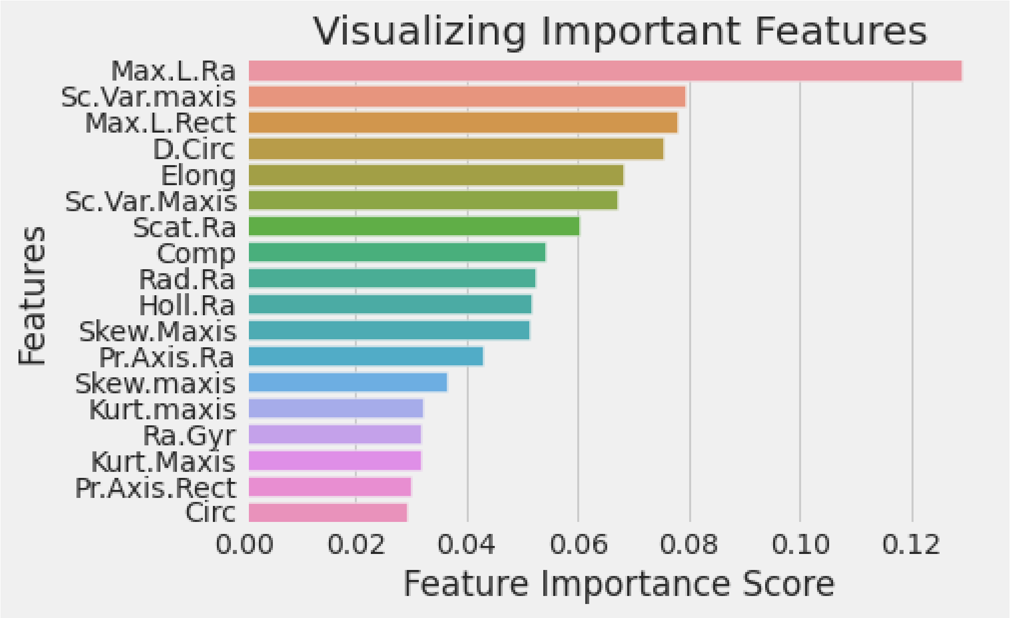
```
(575, 18)
```

```
colss = ['Comp', 'Circ', 'D.Circ', 'Rad.Ra', 'Pr.Axis.Ra', 'Max.L.Ra',
       'Scat.Ra', 'Elong', 'Pr.Axis.Rect', 'Max.L.Rect', 'Sc.Var.Maxis',
       'Sc.Var.maxis', 'Ra.Gyr', 'Skew.Maxis', 'Skew.maxis', 'Kurt.maxis',
       'Kurt.Maxis', 'Holl.Ra']
feature_scores = pd.Series(rfc.feature_importances_, index=colss).sort_values(ascending=False)

feature_scores
```

```
Max.L.Ra        0.128859
Sc.Var.maxis    0.079392
Max.L.Rect      0.077922
D.Circ          0.075444
Elong           0.068048
Sc.Var.Maxis    0.067004
Scat.Ra         0.060081
Comp            0.054147
Rad.Ra          0.052142
Holl.Ra         0.051476
Skew.Maxis      0.051109
Pr.Axis.Ra      0.043071
Skew.maxis      0.036588
Kurt.maxis      0.032019
Ra.Gyr          0.031921
Kurt.Maxis      0.031615
Pr.Axis.Rect    0.029902
Circ            0.029260
dtype: float64
```

```
# Creating a seaborn bar plot

sns.barplot(x=feature_scores, y=feature_scores.index)



# Add labels to the graph

plt.xlabel('Feature Importance Score')
```

```
plt.ylabel('Features')



# Add title to the graph

plt.title("Visualizing Important Features")



# Visualize the graph

plt.show()
```




```
x.head()
```

	Comp	Circ	D.Circ	Rad.Ra	Pr.Axis.Ra	Max.L.Ra	Scat.Ra	Elong	Pr.Axis.Rect	Max.L.Rect	Sc.Var.Maxis	Sc.Var.maxis	Ra.Gy
0	88	39	70	166	66	7	148	44	19	134	167	332	14
1	85	35	64	129	57	6	116	57	17	125	138	200	12
2	91	41	84	141	57	9	149	45	19	143	170	330	15
3	102	54	98	177	56	10	219	31	25	171	219	706	22
4	87	39	74	152	58	6	151	44	19	136	174	337	14

Final Model Building Using Random Forest Model on Selected Features

```
column = ['Kurt.maxis','Ra.Gyr','Kurt.Maxis','Pr.Axis.Rect','Circ']
x = x.drop(column,axis=1)
```

```
x.shape
```

```
(719, 13)
```

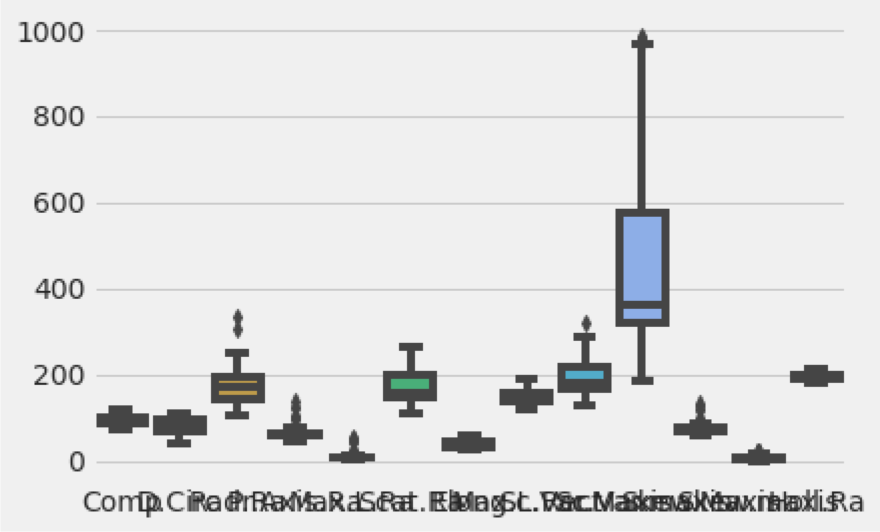
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.20,random_state = 10)
```

```
print('x_train : ', x_train.shape)
print('x_test : ', x_test.shape)
print('y_train : ', y_train.shape)
print('y_test : ', y_test.shape)
```

```
x_train :  (575, 13)
x_test :  (144, 13)
y_train :  (575,)
y_test :  (144,)
```

```
#to check the outliers

sns.boxplot(data = x_train)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f67366deb50>




```
ss = StandardScaler()

x_train = ss.fit_transform(x_train)
x_test = ss.transform(x_test)
```

```
#now the data is standardised
sns.boxplot(data = x_train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6736519310>
```


```
rfc = RandomForestClassifier(criterion= 'gini', max_depth=9, min_samples_leaf= 3, n_estimators= 100)
rfc.fit(x_train,y_train)
y_pred = rfc.predict(x_test)
print('Train Score : ',rfc.score(x_train,y_train))
print('Test Score : ',rfc.score(x_test,y_test))
print('F1_Score : ',f1_score(y_test,y_pred,average= None))
print('confusion_matrix : ', confusion_matrix(y_test,y_pred))
print("Precision Score : ",precision_score(y_test,y_pred,average= None))
print("Recall Score :" , recall_score(y_test, y_pred,average= None))
print('Accuracy_Score : ', accuracy_score(y_test, y_pred))
```

```
Train Score :  0.951304347826087
Test Score :  0.8194444444444444
F1_Score :  [0.97222222 0.63636364 0.56603774 0.96907216]
confusion_matrix :  [[35  0  0  0]
 [ 2 21  9  0]
 [ 0 12 15  1]
 [ 0  1  1 47]]
Precision Score :  [0.94594595 0.61764706 0.6        0.97916667]
Recall Score : [1.         0.65625    0.53571429 0.95918367]
Accuracy_Score :  0.8194444444444444
```

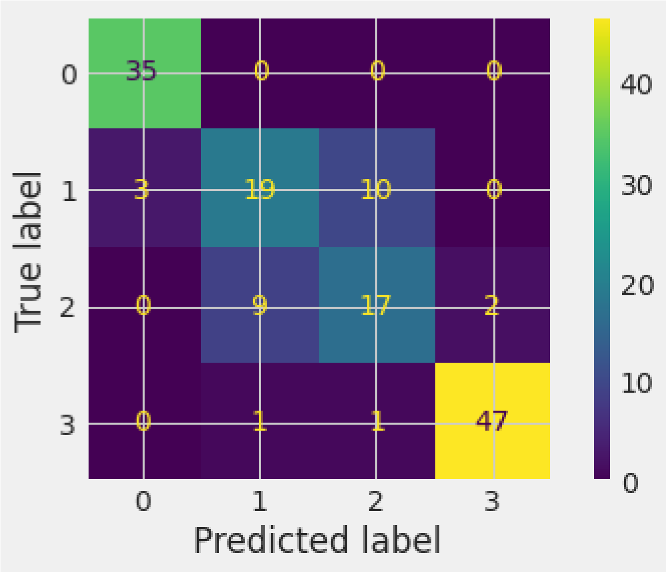
Scores of Random Forest Classifier Final model

- If u see the the tuning model with orginal data we got 75%
- But using feature score method we removed some 5 Unwanted columns
- Now in our model our Score got Inceased by 82%

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, y_pred))
plot_confusion_matrix(rfc,x_test,y_test)
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        35
           1       0.66      0.59      0.62        32
           2       0.61      0.61      0.61        28
           3       0.96      0.96      0.96        49

    accuracy                           0.82       144
   macro avg       0.79      0.79      0.79       144
weighted avg       0.81      0.82      0.82       144

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprec
  warnings.warn(msg, category=FutureWarning)
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f673a5eac90>
```



Results and conclusion

Table of Contents

- In this project, I build a Random Forest Classifier to predict the Class of the cars. I build a models with 100 decision-trees.
- The model accuracy score with Orginal Features the 100 decision-trees is 0.7569 but the same with 100 decision-trees with 13 features after reduction of the least score features the score is 0.8194. So, as expected accuracy increases with number of decision-trees and Important Feature Selection in the model.
- I have used the Random Forest model to find only the important features, build the model using these features and see its effect on accuracy. The most important feature is Max.L.Ra and least important feature is Kurt.maxis,Ra.Gyr,Kurt.Maxis,Pr.Axis.Rect,Circ.
- I have removed the ['Kurt.maxis','Ra.Gyr','Kurt.Maxis','Pr.Axis.Rect','Circ'] variable from the model, rebuild it and checked its accuracy. The accuracy of the model with ['Kurt.maxis','Ra.Gyr','Kurt.Maxis','Pr.Axis.Rect','Circ'] variable removed is 0.8194. The accuracy of the model with all the variables taken into account is 0.7569. So, we can see that the model accuracy has been improved with ['Kurt.maxis','Ra.Gyr','Kurt.Maxis','Pr.Axis.Rect','Circ'] variable removed from the model.
- Confusion matrix and classification report are another tool to visualize the model performance. They yield good performance.