Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Importing Data

In [2]:
```
df=pd.read_csv(r"C:\Users\Shubham\Desktop\Data Science\Data Science Class\Stats a
df.head()
```

Out[2]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.893369	0
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.892193	0
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.846389	0
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296	0.963322	0
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713	0.971955	0

5 rows × 21 columns

Describing Data

In [3]:
```
df.columns
```

Out[3]:
```
Index(['meanfreq', 'sd', 'median', 'Q25', 'Q75', 'IQR', 'skew', 'kurt',
       'sp.ent', 'sfm', 'mode', 'centroid', 'meanfun', 'minfun', 'maxfun',
       'meandom', 'mindom', 'maxdom', 'dfrange', 'modindx', 'label'],
      dtype='object')
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3168 entries, 0 to 3167
Data columns (total 21 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   meanfreq  3168 non-null   float64
 1   sd        3168 non-null   float64
 2   median    3168 non-null   float64
 3   Q25       3168 non-null   float64
 4   Q75       3168 non-null   float64
 5   IQR       3168 non-null   float64
 6   skew      3168 non-null   float64
 7   kurt      3168 non-null   float64
 8   sp.ent    3168 non-null   float64
 9   sfm       3168 non-null   float64
 10  mode      3168 non-null   float64
 11  centroid  3168 non-null   float64
 12  meanfun   3168 non-null   float64
 13  minfun    3168 non-null   float64
 14  maxfun    3168 non-null   float64
 15  meandom   3168 non-null   float64
 16  mindom    3168 non-null   float64
 17  maxdom    3168 non-null   float64
 18  dfrange   3168 non-null   float64
 19  modindx   3168 non-null   float64
 20  label     3168 non-null   object 
dtypes: float64(20), object(1)
memory usage: 519.9+ KB
```

Checking Null Values

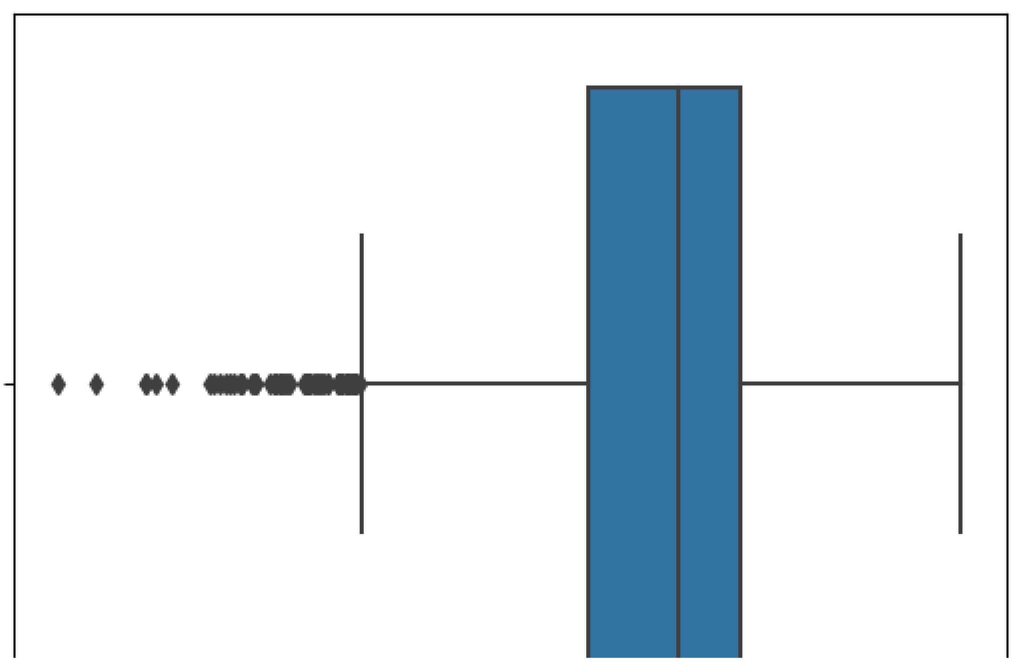
In [5]:
```
df.isna().sum()
```

Out[5]:
```
meanfreq    0
sd          0
median      0
Q25         0
Q75         0
IQR         0
skew        0
kurt        0
sp.ent      0
sfm         0
mode        0
centroid    0
meanfun     0
minfun      0
maxfun      0
meandom     0
mindom      0
maxdom      0
dfrange     0
modindx     0
label       0
dtype: int64
```

Checking Outliers

In [6]:
```
def boxplots(col):
    sns.boxplot(df[col])
    plt.show()

for i in list(df.select_dtypes(exclude=['object']).columns)[0:]:
    boxplots(i)
```



As a natural part of the population we won't remove outliers

Encoding Concept

In [7]:
```
df.label=df.label.astype('category')
df.label=df.label.cat.codes
```

In [8]:
```
x=df.drop(["label"],axis=1)
y=df['label']
```

In [9]:
```
x.head()
```

Out[9]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.893369	0
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.892193	0
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.846389	0
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296	0.963322	0
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713	0.971955	0

In [10]:
```
y.head()
```

Out[10]:
```
0    1
1    1
2    1
3    1
4    1
Name: label, dtype: int8
```

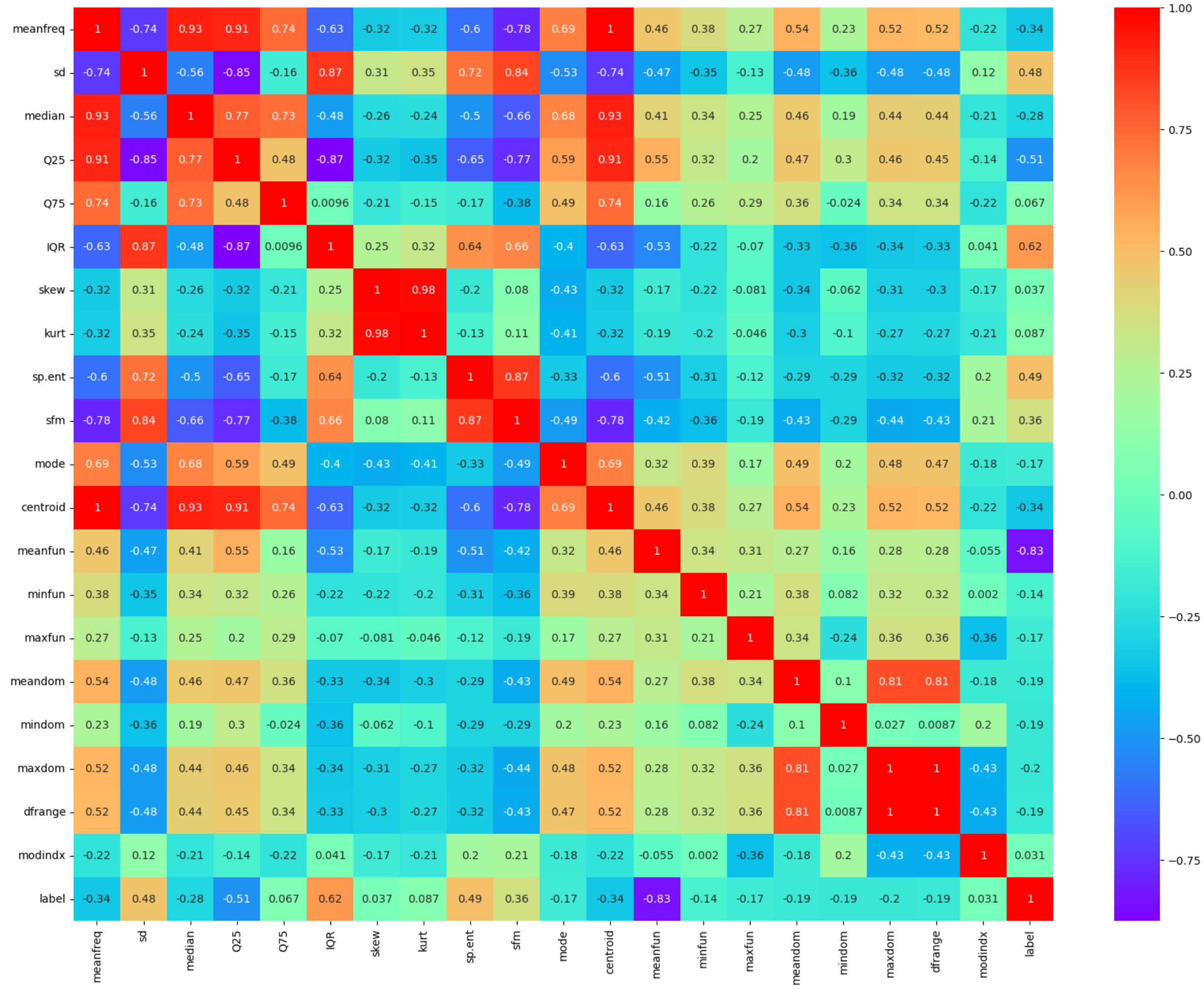
Checking if the data is balanced

In [11]:
```
y.value_counts()
```

Out[11]:
```
1    1584
0    1584
Name: label, dtype: int64
```

Checking Colinearity

In [12]:
```
plt.figure(figsize=(20,15))
sns.heatmap(df.corr(),annot=True, cmap='rainbow')
plt.show()
```



split the training data into train and test

In [13]:
```
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_st
```

Support Vector Machine Model

In [14]:
```
from sklearn.svm import SVC
```

kernel - linear

In [16]:
```
svm_linear=SVC(kernel='linear',probability = True)
svm_linear.fit(x_train,y_train)
y_pred_train_linear=svm_linear.predict(x_train)
y_pred_test_linear=svm_linear.predict(x_test)
```

kernel - sigmoid

In [17]:
```
svm_sigmoid=SVC(kernel='sigmoid')
svm_sigmoid.fit(x_train,y_train)
y_pred_train_sigmoid=svm_sigmoid.predict(x_train)
y_pred_test_sigmoid=svm_sigmoid.predict(x_test)
```

kernel - poly

In [18]:
```
svm_poly=SVC(kernel='poly')
svm_poly.fit(x_train,y_train)
y_pred_train_poly=svm_poly.predict(x_train)
y_pred_test_poly=svm_poly.predict(x_test)
```

kernel - rbf

In [19]:
```
svm_rbf=SVC(kernel='rbf')
svm_rbf.fit(x_train,y_train)
y_pred_train_rbf=svm_rbf.predict(x_train)
y_pred_test_rbf=svm_rbf.predict(x_test)
```

Evulating the Data

In [20]:
```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_sco
```

In [21]:
```
print("Training Accuracy - Linear :", accuracy_score(y_train, y_pred_train_linear
print("*************"*5)
print("Test Accuracy - Linear :", accuracy_score(y_test, y_pred_test_linear))
print("*************"*5)
print("Training Accuracy - sigmoid :", accuracy_score(y_train, y_pred_train_sigmo
print("*************"*5)
print("Test Accuracy - sigmoid :", accuracy_score(y_test, y_pred_test_sigmoid))
print("*************"*5)
print("Training Accuracy - poly :", accuracy_score(y_train, y_pred_train_poly))
print("*************"*5)
print("Test Accuracy - poly :", accuracy_score(y_test, y_pred_test_poly))
print("*************"*5)
print("Training Accuracy - rbf :", accuracy_score(y_train, y_pred_train_rbf))
print("*************"*5)
print("Test Accuracy - rbf :", accuracy_score(y_test, y_pred_test_rbf))
```

```
Training Accuracy - Linear : 0.9222573007103394
*****************************************************************
Test Accuracy - Linear : 0.9148264984227129
*****************************************************************
Training Accuracy - sigmoid : 0.3701657458563536
*****************************************************************
Test Accuracy - sigmoid : 0.3722397476340694
*****************************************************************
Training Accuracy - poly : 0.5197316495659037
*****************************************************************
Test Accuracy - poly : 0.5031545741324921
*****************************************************************
Training Accuracy - rbf : 0.6764009471191792
*****************************************************************
Test Accuracy - rbf : 0.6829652996845426
```

Since Linear SVM gives higher accuracy we will continue with that

In [22]:
```
print("Training Accuracy - Linear :", classification_report(y_train, y_pred_trair
print("*************"*5)
print("Test Accuracy - Linear :", classification_report(y_test, y_pred_test_linea
```

```
Training Accuracy - Linear :               precision    recall  f1-score   supp
ort

           0       0.98      0.86      0.92      1273
           1       0.88      0.98      0.93      1261

    accuracy                           0.92      2534
   macro avg       0.93      0.92      0.92      2534
weighted avg       0.93      0.92      0.92      2534

*****************************************************************
Test Accuracy - Linear :               precision    recall  f1-score   support

           0       0.97      0.85      0.91       311
           1       0.87      0.98      0.92       323

    accuracy                           0.91       634
   macro avg       0.92      0.91      0.91       634
weighted avg       0.92      0.91      0.91       634
```

cross validation method

In [23]:
```
from sklearn.model_selection import cross_val_score
train_accuracy = cross_val_score(svm_linear, x_train, y_train, cv=10)
test_accuracy = cross_val_score(svm_linear, x_test, y_test, cv=10)
print("Training accuracy :", train_accuracy)
print("************"*5)
print("Training Mean Accuracy :", train_accuracy.mean())
print("************"*5)
print("Training Max Accuracy :", train_accuracy.max())


print("Test accuracy :", test_accuracy)
print("************"*5)
print("Test Mean Accuracy :", test_accuracy.mean())
print("************"*5)
print("Test Max Accuracy :", test_accuracy.max())
```

```
Training accuracy : [0.94488189 0.9015748  0.92519685 0.9015748  0.92490119 0.9
0513834
 0.8972332  0.9486166  0.93280632 0.92094862]
************************************************************
Training Mean Accuracy : 0.9202872615231398
************************************************************
Training Max Accuracy : 0.9486166007905138
Test accuracy : [0.890625   0.890625   0.9375     0.828125   0.87301587 0.80952
381
 0.92063492 0.87301587 0.88888889 0.87301587]
************************************************************
Test Mean Accuracy : 0.8784970238095239
************************************************************
Test Max Accuracy : 0.9375
```

Grid Search CV(Hyperperameter Tuning)

In [24]:
```
from sklearn.model_selection import GridSearchCV
```

In [25]:
```
SVC()
```

Out[25]:
```
SVC()
```

In [26]:
```
param_grid = {'C':[0,1,2,10,100], 'gamma':[1,0.1,0.001,0.01]}
grid = GridSearchCV(SVC(), param_grid, refit=True)
grid.fit(x_train, y_train)
grid_predict = grid.predict(x_test)
print(accuracy_score(y_test, grid_predict))
```

```
0.9211356466876972
```

In [ ]:

Building Voting Class Model- It combines various models and produce higer accurcy

In [ ]:

LogisticRegression

In [27]:
```
from sklearn.linear_model import LogisticRegression
logit = LogisticRegression()
logit.fit(x_train, y_train)
```

Out[27]:
```
LogisticRegression()
```

In [28]:
```
y_pred_logit_train = logit.predict(x_train)
y_pred_logit_test = logit.predict(x_test)
```

In [29]:
```
print("Training Accuracy - Logistic :", accuracy_score(y_train, y_pred_logit_tra
print("*************"*5)
print("Test Accuracy - Logistic :", accuracy_score(y_test, y_pred_logit_test))
```

```
Training Accuracy - Logistic : 0.9048934490923441
*****************************************************************
Test Accuracy - Logistic : 0.8943217665615142
```

DecisionTreeClassifier

In [30]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(x_train, y_train)
```

Out[30]:
```
DecisionTreeClassifier()
```

In [31]:
```
y_pred_dtree_train = dtree.predict(x_train)
y_pred_dtree_test = dtree.predict(x_test)
```

In [32]:
```
print("Training Accuracy - Dtree :", accuracy_score(y_train, y_pred_dtree_train))
print("*************"*5)
print("Test Accuracy - dtree :", accuracy_score(y_test, y_pred_dtree_test))
```

```
Training Accuracy - Dtree : 1.0
*****************************************************************
Test Accuracy - dtree : 0.9747634069400631
```

KNN

In [33]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(x_train, y_train)
```

Out[33]:
```
KNeighborsClassifier()
```

In [34]:
```
y_pred_knn_train = knn.predict(x_train)
y_pred_knn_test = knn.predict(x_test)
```

In [35]:
```
print("Training Accuracy - knn :", accuracy_score(y_train, y_pred_knn_train))
print("*************"*5)
print("Test Accuracy - knn :", accuracy_score(y_test, y_pred_knn_test))
```

```
Training Accuracy - knn : 0.8113654301499605
*****************************************************************
Test Accuracy - knn : 0.722397476340694
```

Using Voting Method

In [36]:
```
estimators = [('SVM_Linear',svm_linear),('SVM_Sigmoid',svm_sigmoid),('SVM_Poly',
              ("SVM_RBF" ,svm_rbf),('Logistic',logit),('Dtree', dtree),('KNN',knr
```

In [37]:
```
from sklearn.ensemble import VotingClassifier
```

Hard Voting

In [38]:
```
voting_hard = VotingClassifier(estimators = estimators, voting='hard')
v_train_accuracy = cross_val_score(voting_hard, x_train, y_train, cv=10, scoring=
v_test_accuracy = cross_val_score(voting_hard, x_test, y_test, cv=10, scoring='ac
print(np.round(np.mean(v_train_accuracy),2))
print()
print(np.round(np.mean(v_test_accuracy),2))
```

```
0.9

0.79
```

Since there a big diiference in the accuracy in train and test data we have overfitting problem

So we will only take linear SVM and try

In [39]:
```
estimators1 = [('SVM_Linear',svm_linear),('Logistic',logit),('Dtree', dtree),('KN
```

In [40]:
```
voting_hard1 = VotingClassifier(estimators = estimators1, voting='hard')
voting_hard1.fit(x_train, y_train)
```

Out[40]:
```
VotingClassifier(estimators=[('SVM_Linear',
                              SVC(kernel='linear', probability=True)),
                             ('Logistic', LogisticRegression()),
                             ('Dtree', DecisionTreeClassifier()),
                             ('KNN', KNeighborsClassifier())])
```

In [41]:
```
y_pred_train_voting = voting_hard1.predict(x_train)
y_pred_test_voting = voting_hard1.predict(x_test)
```

In [42]:
```
print("Training Accuracy - voting_hard :", accuracy_score(y_train, y_pred_train_v
print("*************"*5)
print("Test Accuracy - voting_hard :", accuracy_score(y_test, y_pred_test_voting)
```

```
Training Accuracy - voting_hard : 0.9593528018942383
*****************************************************************
Test Accuracy - voting_hard : 0.9258675078864353
```

In [ ]:

Soft Voting

In [44]:
```
voting_soft = VotingClassifier(estimators = estimators1, voting='soft')
v_train_accuracy1 = cross_val_score(voting_soft, x_train, y_train, cv=10, scoring
v_test_accuracy1 = cross_val_score(voting_soft, x_test, y_test, cv=10, scoring='a
print("Training Accuracy - voting_soft :", np.round(np.mean(v_train_accuracy1),2)
print("*************"*5)
print("Test Accuracy - voting_soft :",np.round(np.mean(v_test_accuracy1),2))
```

```
Training Accuracy - voting_soft : 0.94
*****************************************************************
Test Accuracy - voting_soft : 0.91
```

In [ ]: