Importing Library

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

Importing Dataset

In [169]:
```
df=pd.read_csv(r"C:\Users\Shubham\Desktop\Data Science\Data Science Class\Stats a
df.head()
```

Out[169]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase
0	65	Female	0.7	0.1	187	16
1	62	Male	10.9	5.5	699	64
2	62	Male	7.3	4.1	490	60
3	58	Male	1.0	0.4	182	14
4	72	Male	3.9	2.0	195	27

Data Discription

In [170]:
```
df.Dataset.value_counts()
```

Out[170]:
```
1    416
2    167
Name: Dataset, dtype: int64
```

In [171]:
```
df.info()
```

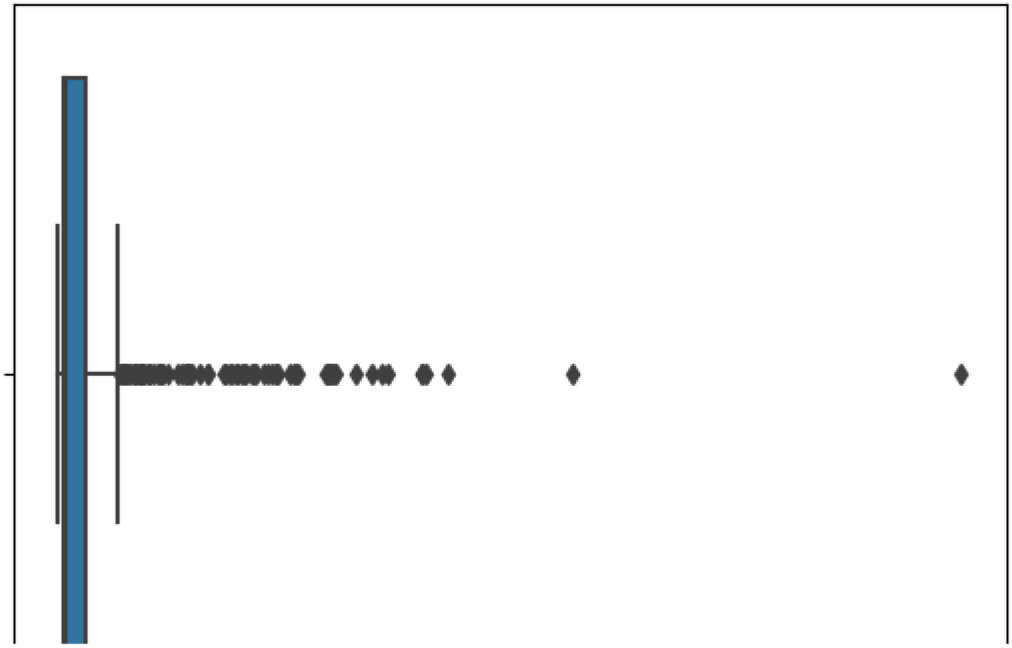
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Age                         583 non-null    int64  
 1   Gender                      583 non-null    object 
 2   Total_Bilirubin             583 non-null    float64
 3   Direct_Bilirubin            583 non-null    float64
 4   Alkaline_Phosphotase        583 non-null    int64  
 5   Alamine_Aminotransferase    583 non-null    int64  
 6   Aspartate_Aminotransferase  583 non-null    int64  
 7   Total_Protiens              583 non-null    float64
 8   Albumin                     583 non-null    float64
 9   Albumin_and_Globulin_Ratio  579 non-null    float64
 10  Dataset                     583 non-null    int64  
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
```

Checking Null Values

In [172]:
```
df.isna().sum()
```

Out[172]:
```
Age                           0
Gender                        0
Total_Bilirubin               0
Direct_Bilirubin              0
Alkaline_Phosphotase          0
Alamine_Aminotransferase      0
Aspartate_Aminotransferase    0
Total_Protiens                0
Albumin                       0
Albumin_and_Globulin_Ratio    4
Dataset                       0
dtype: int64
```

In [173]:
```
def boxplots(col):
    sns.boxplot(df[col])
    plt.show()

for i in list(df.select_dtypes(exclude=['object']).columns)[1:]:
    boxplots(i)
```



In [174]:
```
df.Albumin_and_Globulin_Ratio=df.Albumin_and_Globulin_Ratio.fillna(df.Albumin_and
```

In [175]:
```
df.isna().sum()
```

Out[175]:
```
Age                           0
Gender                        0
Total_Bilirubin               0
Direct_Bilirubin              0
Alkaline_Phosphotase          0
Alamine_Aminotransferase      0
Aspartate_Aminotransferase    0
Total_Protiens                0
Albumin                       0
Albumin_and_Globulin_Ratio    0
Dataset                       0
dtype: int64
```

Encoding Concept

In [176]:
```
df.Gender=df.Gender.astype('category')
df.Gender=df.Gender.cat.codes
```

In [177]:
```
df.head()
```

Out[177]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase
0	65	0	0.7	0.1	187	16
1	62	1	10.9	5.5	699	64
2	62	1	7.3	4.1	490	60
3	58	1	1.0	0.4	182	14
4	72	1	3.9	2.0	195	27

Spliting the data into independent variable and dependent variable

In [ ]:

In [178]:
```
x=df.iloc[:,0:-1]
y=df.iloc[:,-1]
```

In [179]:
```
x.head()
```

Out[179]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase
0	65	0	0.7	0.1	187	16
1	62	1	10.9	5.5	699	64
2	62	1	7.3	4.1	490	60
3	58	1	1.0	0.4	182	14
4	72	1	3.9	2.0	195	27

In [180]:
```
y.head()
```

Out[180]:
```
0    1
1    1
2    1
3    1
4    1
Name: Dataset, dtype: int64
```

Handling Imbalacne Data

In [181]:
```
import imblearn
```

In [182]:
```
from imblearn.over_sampling import RandomOverSampler
ros=RandomOverSampler()
x_sam,y_sam=ros.fit_resample(x,y)
```

In [183]:
```
y_sam.value_counts()
```

Out[183]:
```
1    416
2    416
Name: Dataset, dtype: int64
```

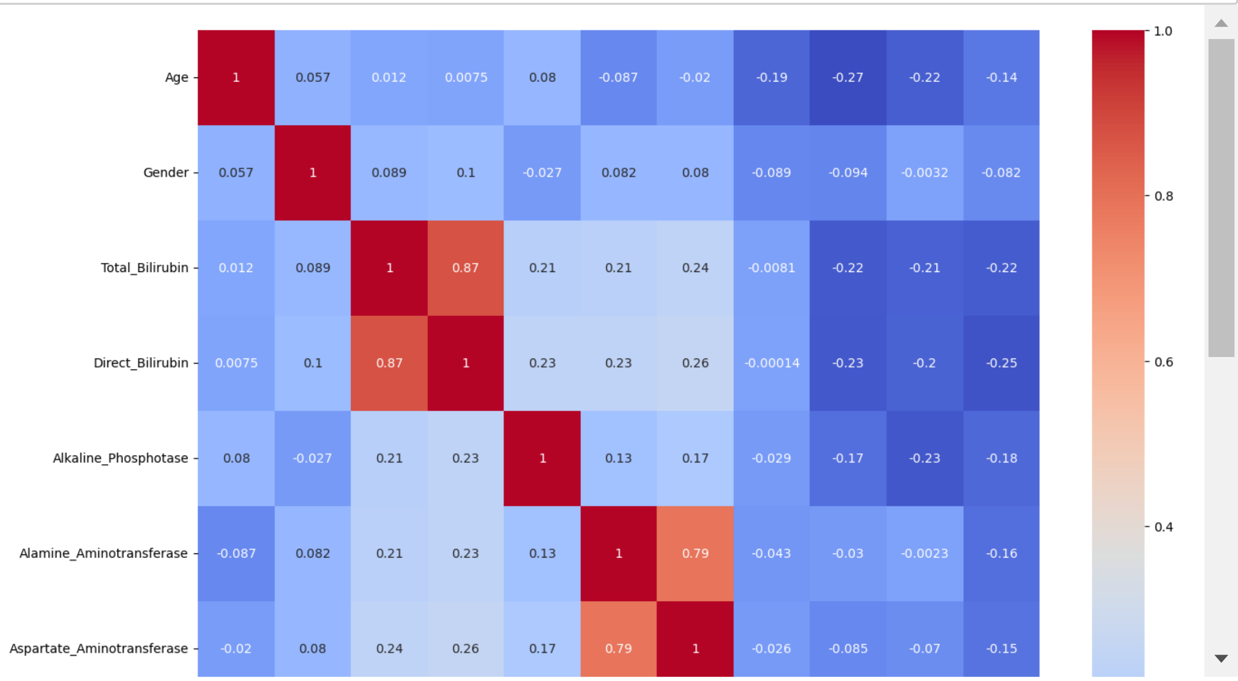
In [184]:
```
y.value_counts()
```

Out[184]:
```
1    416
2    167
Name: Dataset, dtype: int64
```

Data Preprocessing

In [ ]:

In [185]:
```
plt.figure(figsize=(15,15))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```



Split the data into training and test for building the model and for prediction

In [ ]:

In [186]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_sam,y_sam,train_size=0.35,randor
```

In [ ]:

1. KNeighborsClassifier Model

In [187]:
```
from sklearn.neighbors import KNeighborsClassifier
```

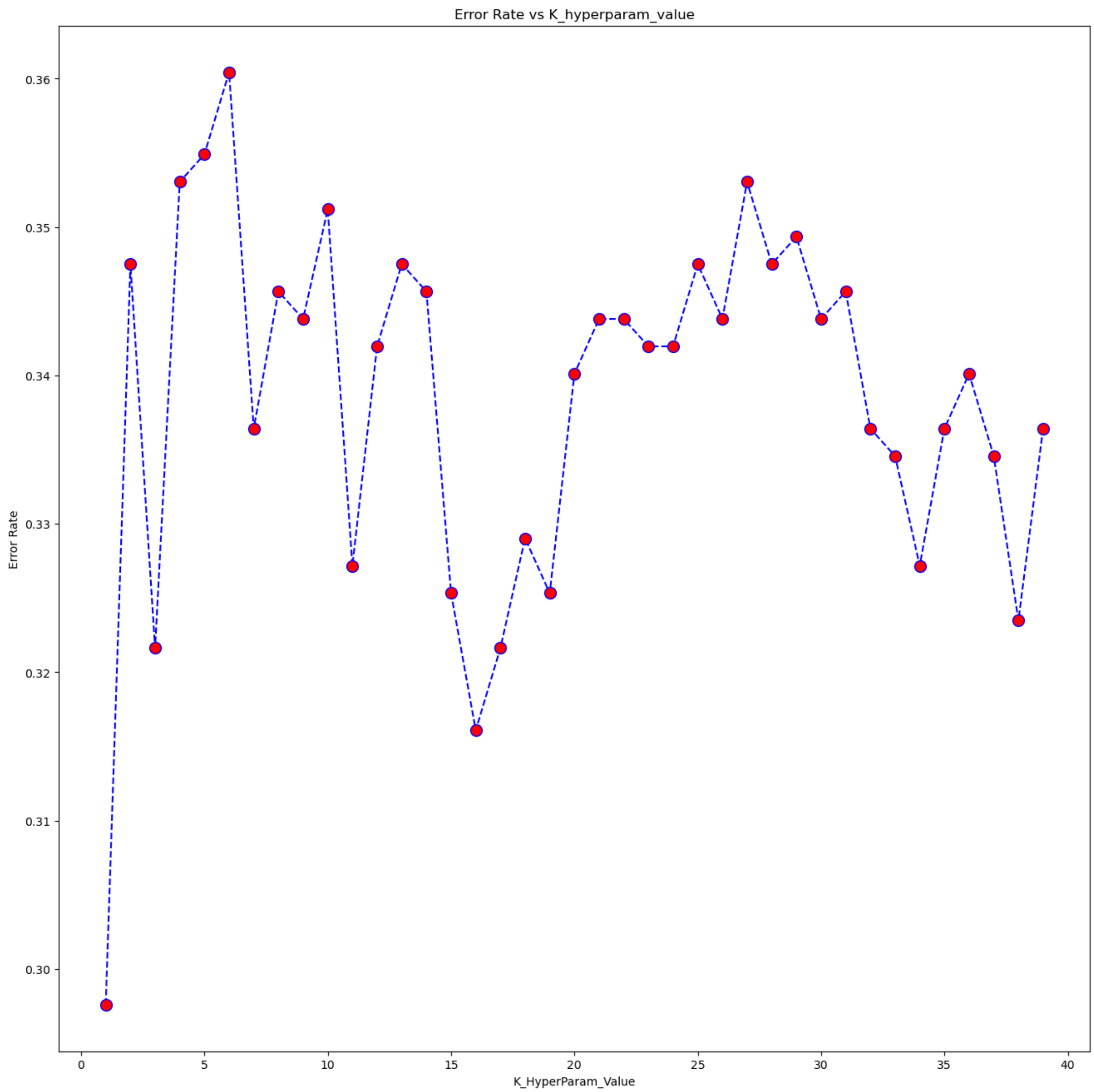
In [188]:
```
error_rate  = []

for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train, y_train)
    pred_i = knn.predict(x_test)
    error_rate.append(np.mean(pred_i != y_test))
```

In [189]:
```
error_rate
```

Out[189]:
```
[0.2975970425138632,
 0.34750462107208874,
 0.32162661737523107,
 0.35304990757855825,
 0.35489833641404805,
 0.36044362292051757,
 0.3364140480591497,
 0.3456561922365989,
 0.3438077634011091,
 0.3512014787430684,
 0.32717190388170053,
 0.3419593345656192,
 0.34750462107208874,
 0.3456561922365989,
 0.32532347504621073,
 0.31608133086876156,
 0.32162661737523107,
 0.3290203327171904,
 0.32532347504621073,
 0.34011090573012936,
 0.3438077634011091,
 0.3438077634011091,
 0.3419593345656192,
 0.3419593345656192,
 0.34750462107208874,
 0.3438077634011091,
 0.35304990757855825,
 0.34750462107208874,
 0.34935304990757854,
 0.3438077634011091,
 0.3456561922365989,
 0.3364140480591497,
 0.3345656192236599,
 0.32717190388170053,
 0.3364140480591497,
 0.34011090573012936,
 0.3345656192236599,
 0.3234750462107209,
 0.3364140480591497]
```

Plotting Error Rate vs K_hyperparam_value Graph

In [190]:
```
plt.figure(figsize=(16,16))
plt.plot(range(1,40), error_rate, color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title("Error Rate vs K_hyperparam_value")
plt.xlabel("K_HyperParam_Value")
plt.ylabel("Error Rate")
plt.show()
```



Finding the N valuse based on the above data

In [191]:
```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(x_train, y_train)
```

Out[191]:
```
KNeighborsClassifier(n_neighbors=1)
```

Predict the data

In [ ]:

In [192]:
```
y_pred_train = knn.predict(x_train)
y_pred_test = knn.predict(x_test)
```

Evaluate the model

In [ ]:

In [193]:
```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_sc
```

In [194]:
```
print(confusion_matrix(y_train, y_pred_train))
print("***************"*5)
print(confusion_matrix(y_test, y_pred_test))
```

```
[[146   0]
 [  0 145]]
******************************************************************************
*
[[178  92]
 [ 69 202]]
```

In [195]:
```
print(classification_report(y_train, y_pred_train))
print("***************"*5)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       146
           2       1.00      1.00      1.00       145

    accuracy                           1.00       291
   macro avg       1.00      1.00      1.00       291
weighted avg       1.00      1.00      1.00       291

******************************************************************************
*
              precision    recall  f1-score   support

           1       0.72      0.66      0.69       270
           2       0.69      0.75      0.72       271

    accuracy                           0.70       541
   macro avg       0.70      0.70      0.70       541
weighted avg       0.70      0.70      0.70       541
```

In [196]:
```
print("Training Accuracy :", accuracy_score(y_train, y_pred_train))
print("****************"*5)
print("Test Accuracy :", accuracy_score(y_test, y_pred_test))
```

```
Training Accuracy : 1.0
*******************************************************************************
*
Test Accuracy : 0.7024029574861368
```

Cross Validation approach - K-Fold Method

In [ ]:

In [197]:
```
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(knn, x_train, y_train, cv=10)
test_accuracy = cross_val_score(knn, x_test, y_test, cv=10)
print("Training Accuracy after CV :", training_accuracy.mean())
print("***********************"*5)
print("Test Accuracy after CV :", test_accuracy.mean())
```

```
Training Accuracy after CV : 0.7222988505747128
*******************************************************************************
************************************
Test Accuracy after CV : 0.7984175084175085
```

In [ ]:

2. BaggingClassifier Model

In [198]:
```
from sklearn.ensemble import BaggingClassifier
bagging=BaggingClassifier()
bagging.fit(x_train,y_train)
```

Out[198]:
```
BaggingClassifier()
```

Predict the data

In [ ]:

In [199]:
```
y_pred_train_bgg=bagging.predict(x_train)
y_pred_test_bgg=bagging.predict(x_test)
```

Evaluate the model

In [ ]:

```
```

In [200]:

```
from sklearn.metrics import confusion_matrix, classification_report,accuracy_scor
```

In [201]:

```
print(confusion_matrix(y_train,y_pred_train_bgg))
print(confusion_matrix(y_test,y_pred_test_bgg))
```

```
[[145   1]
 [  1 144]]
[[194  76]
 [ 45 226]]
```

In [202]:

```
print(classification_report(y_train,y_pred_train_bgg))
print(classification_report(y_test,y_pred_test_bgg))
```

```
              precision    recall  f1-score   support

           1       0.99      0.99      0.99       146
           2       0.99      0.99      0.99       145

    accuracy                           0.99       291
   macro avg       0.99      0.99      0.99       291
weighted avg       0.99      0.99      0.99       291

              precision    recall  f1-score   support

           1       0.81      0.72      0.76       270
           2       0.75      0.83      0.79       271

    accuracy                           0.78       541
   macro avg       0.78      0.78      0.78       541
weighted avg       0.78      0.78      0.78       541
```

In [203]:

```
print(accuracy_score(y_train,y_pred_train_bgg))
print(accuracy_score(y_test,y_pred_test_bgg))
```

```
0.993127147766323
0.7763401109057301
```

Cross Validation approach - K-Fold Method

In [ ]:

```
```

In [204]:
```
training_accuracy = cross_val_score(bagging, x_train, y_train, cv=10)
test_accuracy = cross_val_score(bagging, x_test, y_test, cv=10)
print("Training Accuracy after CV :", training_accuracy.mean())
print("***********************"*5)
print("Test Accuracy after CV :", test_accuracy.mean())
```

```
Training Accuracy after CV : 0.7185057471264369
*******************************************************************************
************************************
Test Accuracy after CV : 0.8188888888888888
```

In [ ]:

3. RandomForestClassifier Model

In [205]:
```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=200,criterion='entropy',bootstrap=True,oob
rf.fit(x_train,y_train)
```

Out[205]:
```
RandomForestClassifier(criterion='entropy', n_estimators=200)
```

Predict the data

In [206]:
```
y_pred_train_rf=rf.predict(x_train)
y_pred_test_rf=rf.predict(x_test)
```

Evaluate the model

In [ ]:

In [207]:
```
print(confusion_matrix(y_train,y_pred_train_rf))
print(confusion_matrix(y_test,y_pred_test_rf))
```

```
[[146   0]
 [  0 145]]
[[180  90]
 [ 48 223]]
```

In [208]:
```
print(classification_report(y_train,y_pred_train_rf))
print(classification_report(y_test,y_pred_test_rf))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00       146
           2       1.00      1.00      1.00       145

    accuracy                           1.00       291
   macro avg       1.00      1.00      1.00       291
weighted avg       1.00      1.00      1.00       291

              precision    recall  f1-score   support

           1       0.79      0.67      0.72       270
           2       0.71      0.82      0.76       271

    accuracy                           0.74       541
   macro avg       0.75      0.74      0.74       541
weighted avg       0.75      0.74      0.74       541
```

In [209]:
```
print(accuracy_score(y_train,y_pred_train_rf))
print(accuracy_score(y_test,y_pred_test_rf))
```

```
1.0
0.744916820702403
```

Cross Validation approach - K-Fold Method

In [ ]:

In [210]:
```
training_accuracy = cross_val_score(rf, x_train, y_train, cv=10)
test_accuracy = cross_val_score(rf, x_test, y_test, cv=10)
print("Training Accuracy after CV :", training_accuracy.mean())
print("**********************"*5)
print("Test Accuracy after CV :", test_accuracy.mean())
```

```
Training Accuracy after CV : 0.7357471264367816
**************************************************************************************************************
Test Accuracy after CV : 0.8040404040404041
```

In [ ]:

In [ ]: