```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

import warnings
warnings.filterwarnings('ignore')
sns.set()
plt.style.use('ggplot')
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
```

```
df=pd.read_csv("/content/liver patient.csv")
```

```
df.sample(5)
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_P
185	38	Male	1.5	0.4	298	60	103	
395	45	Male	0.8	0.2	140	24	20	
254	38	Female	0.7	0.1	152	90	21	
67	37	Male	1.8	0.8	215	53	58	
116	48	Male	0.7	0.1	1630	74	149	

```
# Target Column is Dataset
# Supervised or Unsupervised
# Classification
# Binary
```

```
df.shape
```

```
(583, 11)
```

```
df.columns
```

```
Index(['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin',
       'Alkaline_Phosphotase', 'Alamine_Aminotransferase',
       'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',
       'Albumin_and_Globulin_Ratio', 'Dataset'],
      dtype='object')
```

```
df.describe()
```

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_I
count	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	58
mean	44.746141	3.298799	1.486106	290.576329	80.713551	109.910806	
std	16.189833	6.209522	2.808498	242.937989	182.620356	288.918529	
min	4.000000	0.400000	0.100000	63.000000	10.000000	10.000000	
25%	33.000000	0.800000	0.200000	175.500000	23.000000	25.000000	
50%	45.000000	1.000000	0.300000	208.000000	35.000000	42.000000	
75%	58.000000	2.600000	1.300000	298.000000	60.500000	87.000000	
max	90.000000	75.000000	19.700000	2110.000000	2000.000000	4929.000000	

```
# EDA
```

```
df.info()
```

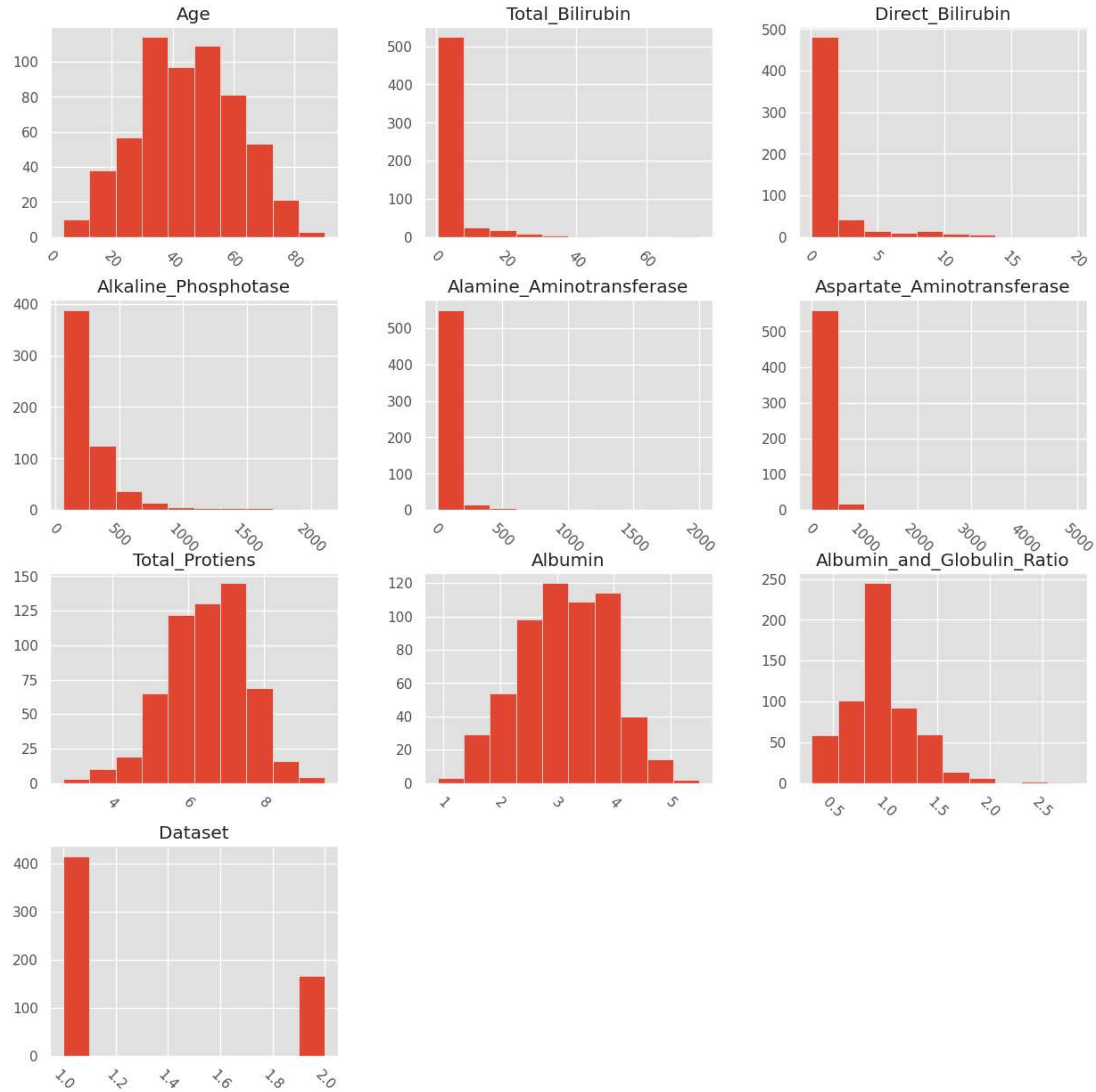
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Age                         583 non-null    int64  
 1   Gender                      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
```

```
df.dtypes[df.dtypes=='object']
```

```
Gender    object
dtype: object
```

Distribution of Numberical Features

```
df.hist(figsize=(15,15), xrot=-45,bins=10)
plt.show()
```


Age
Total_Bilirubin
Direct_Bilirubin
Alkaline_Phosphotase
Alamine_Aminotransferase
Aspartate_Aminotransferase
Total_Protiens
Albumin
Albumin_and_Globulin_Ratio
Dataset

```
df.describe()
```

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_I
count	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	58
mean	44.746141	3.298799	1.486106	290.576329	80.713551	109.910806	
std	16.189833	6.209522	2.808498	242.937989	182.620356	288.918529	
min	4.000000	0.400000	0.100000	63.000000	10.000000	10.000000	
25%	33.000000	0.800000	0.200000	175.500000	23.000000	25.000000	
50%	45.000000	1.000000	0.300000	208.000000	35.000000	42.000000	
75%	58.000000	2.600000	1.300000	298.000000	60.500000	87.000000	
max	90.000000	75.000000	19.700000	2110.000000	2000.000000	4929.000000	

```
def convertdataset(x):
    if x==2:
        return 0
    return 1
df['Dataset'] = df['Dataset'].map(convertdataset)
```

```
df.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Pro
0	65	Female	0.7	0.1	187	16	18	
1	62	Male	10.9	5.5	699	64	100	
2	62	Male	7.3	4.1	490	60	68	
3	58	Male	1.0	0.4	182	14	20	
4	72	Male	3.9	2.0	195	27	59	

Next steps: Generate code with df View recommended plots

```
df.Dataset.value_counts()
```

```
1    416
0    167
Name: Dataset, dtype: int64
```

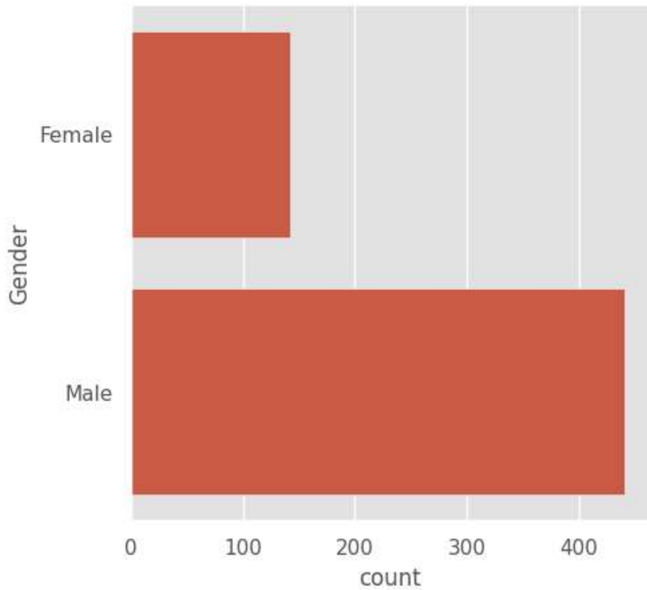
```
df.describe(include=['object'])
```

	Gender
count	583
unique	2
top	Male
freq	441

```
# Bar plots for categorical features
```

```
plt.figure(figsize=(5,5))
sns.countplot(y='Gender', data=df)
```

```
<Axes: xlabel='count', ylabel='Gender'>
```

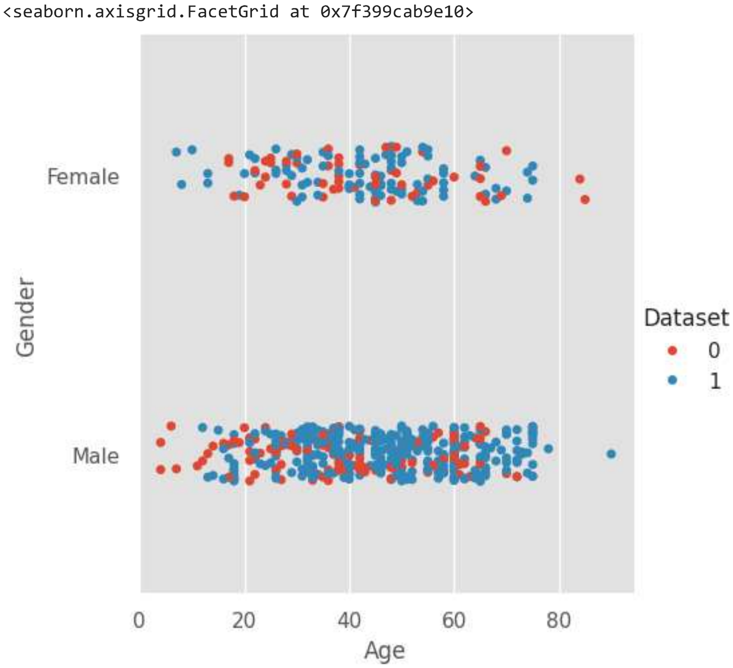



```
df[df['Gender'] == 'Male'][['Dataset','Gender']].head()
```

	Dataset	Gender
1	1	Male
2	1	Male
3	1	Male
4	1	Male
5	1	Male

```
sns.catplot(x="Age", y="Gender", hue="Dataset", data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f399cab9e10>
```




```
df['Gender'].value_counts()
```

```
Male      441
Female    142
Name: Gender, dtype: int64
```

```
# Categorical Value Handling
def convertgender(x):
    if x== 'Male':
        return 0
    else:
        return 1
df['Gender'] = df['Gender'].map(convertgender)
```

```
df.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Pro
0	65	1	0.7	0.1	187	16	18	
1	62	0	10.9	5.5	699	64	100	
2	62	0	7.3	4.1	490	60	68	
3	58	0	1.0	0.4	182	14	20	
4	72	0	3.9	2.0	195	27	59	

Next steps: Generate code with df View recommended plots

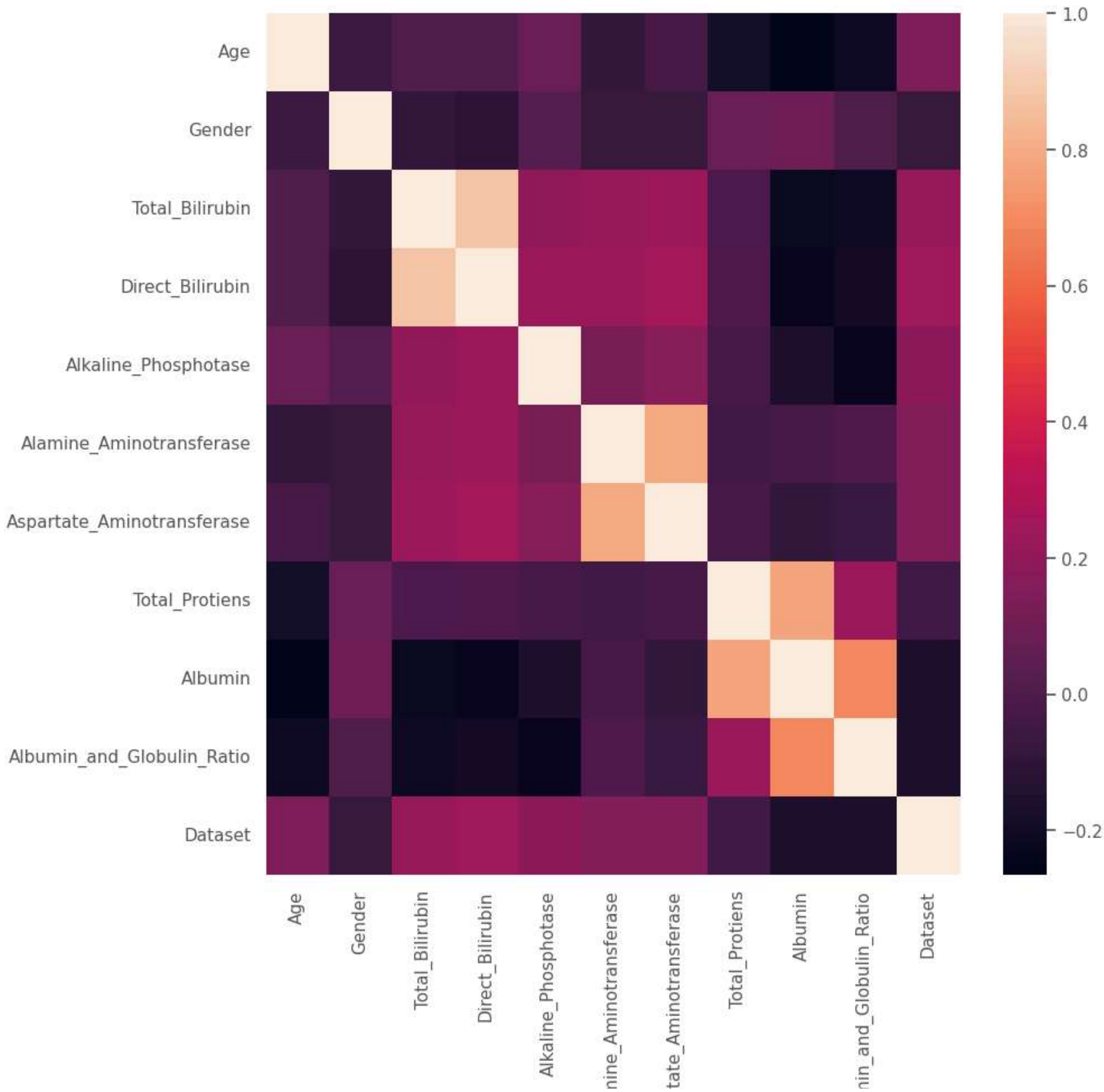
```
df.corr()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspar
Age	1.000000	-0.056560	0.011763	0.007529	0.080425	-0.086883	
Gender	-0.056560	1.000000	-0.089291	-0.100436	0.027496	-0.082332	
Total_Bilirubin	0.011763	-0.089291	1.000000	0.874618	0.206669	0.214065	
Direct_Bilirubin	0.007529	-0.100436	0.874618	1.000000	0.234939	0.233894	
Alkaline_Phosphotase	0.080425	0.027496	0.206669	0.234939	1.000000	0.125680	
Alamine_Aminotransferase	-0.086883	-0.082332	0.214065	0.233894	0.125680	1.000000	
Aspartate_Aminotransferase	-0.019910	-0.080336	0.237831	0.257544	0.167196	0.791966	
Total_Protiens	-0.187461	0.089121	-0.008099	-0.000139	-0.028514	-0.042518	
Albumin	-0.265924	0.093799	-0.222250	-0.228531	-0.165453	-0.029742	
Albumin_and_Globulin_Ratio	-0.216408	0.003424	-0.206267	-0.200125	-0.234166	-0.002375	
Dataset	0.137351	-0.082416	0.220208	0.246046	0.184866	0.163416	

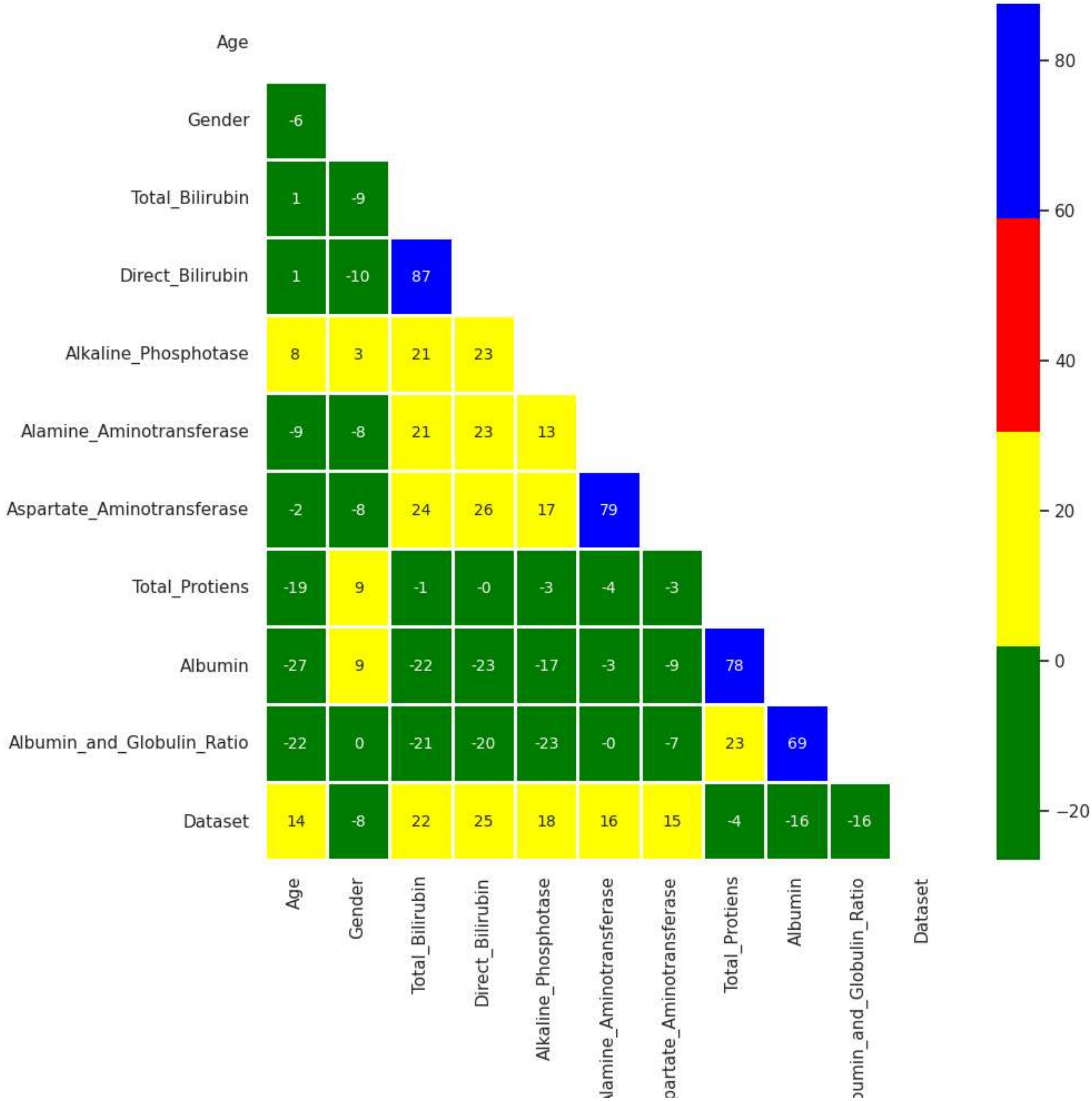
```
# Positive Correlation-> one feature increases other also increases
# Negative Correlation-> one feature increases other decreases
# closer to 0-> weak relationship
```

```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr())
```

```
<Axes: >
```




```
mask = np.zeros_like(df.corr())
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize=(10,10))
with sns.axes_style("white"):
    ax = sns.heatmap(df.corr()*100, mask=mask, fmt = ".0f", annot=True, lw=1, cmap=ListedColormap(['green','yellow','red','blue']))
```




```
df = df.drop_duplicates()
```

```
df.shape
```

```
(570, 11)
```

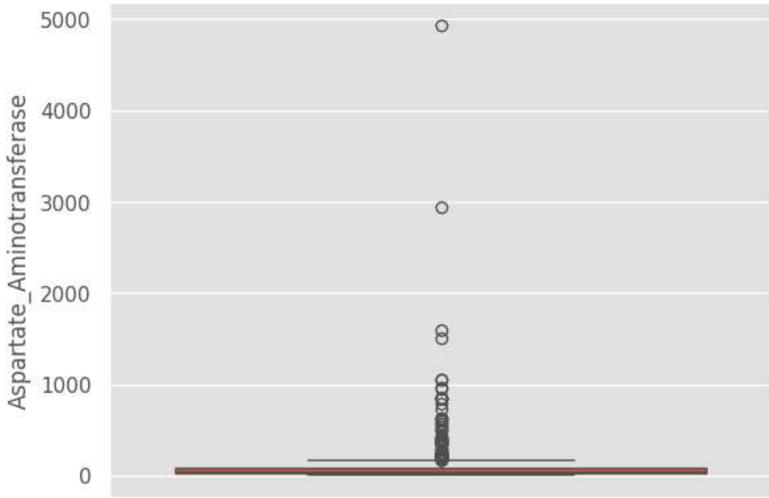
```
# Removing Outlier
```

```
df.columns
```

```
Index(['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin',
       'Alkaline_Phosphotase', 'Alamine_Aminotransferase',
       'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',
       'Albumin_and_Globulin_Ratio', 'Dataset'],
      dtype='object')
```

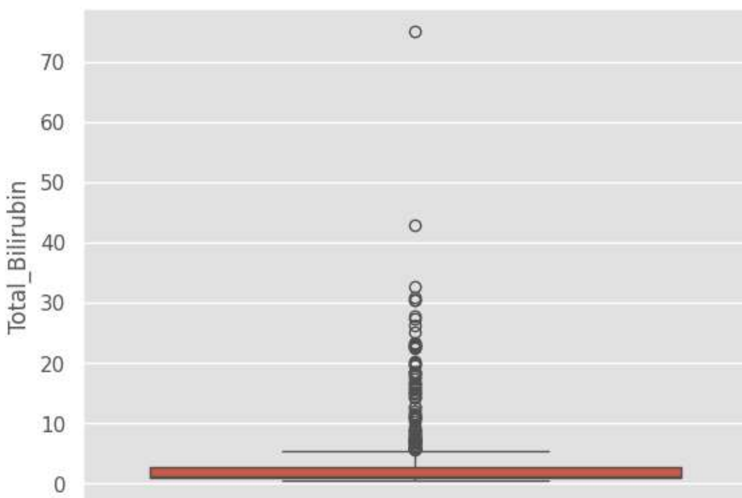
```
sns.boxplot(df.Aspartate_Aminotransferase)
```

```
<Axes: ylabel='Aspartate_Aminotransferase'>
```




```
sns.boxplot(df.Total_Bilirubin)
```

```
<Axes: ylabel='Total_Bilirubin'>
```




```
df.Aspartate_Aminotransferase.sort_values(ascending=False).head()
```

```
135    4929
117    2946
118    1600
207    1500
199    1050
Name: Aspartate_Aminotransferase, dtype: int64
```

```
df = df[df.Aspartate_Aminotransferase<=3000]
```

```
df.shape
```

```
(569, 11)
```

```
df.Aspartate_Aminotransferase.sort_values(ascending=False).head()
```

```
117    2946
118    1600
207    1500
119    1050
199    1050
Name: Aspartate_Aminotransferase, dtype: int64
```

```
df = df[df.Aspartate_Aminotransferase<=2500]
```

```
df.shape
```

```
(568, 11)
```

```
df.isnull().sum()
```

```
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
```

```
df = df.dropna(how='any')
```

```
df.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Pro
0	65	1	0.7	0.1	187	16	18	
1	62	0	10.9	5.5	699	64	100	
2	62	0	7.3	4.1	490	60	68	
3	58	0	1.0	0.4	182	14	20	
4	72	0	3.9	2.0	195	27	59	

Next steps: Generate code with df View recommended plots

```
df.shape
```

```
(564, 11)
```

```
# Machine Learning Model
```

```
# Data Preparation
```

```
y=df.Dataset
X=df.drop('Dataset', axis=1)
```

```
X_train, X_test, y_train , y_test = train_test_split(X,y, test_size=0.2, random_state=0, stratify=y)
```

```
# Data Standardization
```

```
train_mean = X_train.mean()
train_std = X_train.std()
```

```
X_train = (X_train - train_mean) / train_std
```

```
X_train.describe()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotra
count	4.510000e+02	4.510000e+02	4.510000e+02	4.510000e+02	4.510000e+02	4.510000e+02	4.51
mean	1.043757e-16	3.938707e-17	-2.363224e-17	-3.150966e-17	-1.240693e-16	1.575483e-17	-3.93
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.00
min	-2.568370e+00	-5.468521e-01	-4.545818e-01	-4.862972e-01	-9.633546e-01	-4.150881e-01	-4.88
25%	-7.747074e-01	-5.468521e-01	-3.928717e-01	-4.516441e-01	-4.787490e-01	-3.330804e-01	-4.08
50%	2.934809e-02	-5.468521e-01	-3.620167e-01	-4.169909e-01	-3.259454e-01	-2.573809e-01	-3.16
75%	8.952540e-01	-5.468521e-01	-1.151762e-01	-7.045893e-02	5.169765e-02	-9.967374e-02	-7.97
max	2.441515e+00	1.824593e+00	1.105435e+01	6.305729e+00	7.973471e+00	1.011975e+01	8.58

```
X_test = (X_test - train_mean) / train_std
```

```
X_test.describe()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransfer
count	113.000000	113.000000	113.000000	113.000000	113.000000	113.000000	113.000
mean	-0.201633	0.166680	-0.029574	-0.022619	0.166349	-0.035753	-0.004
std	1.019796	1.092491	0.833443	0.911646	1.288436	0.912202	0.926
min	-2.382818	-0.546852	-0.423727	-0.486297	-0.627187	-0.408780	-0.493
25%	-0.836558	-0.546852	-0.392872	-0.451644	-0.443822	-0.326772	-0.402
50%	-0.094353	-0.546852	-0.362017	-0.382338	-0.339043	-0.257381	-0.316
75%	0.462301	1.824593	-0.084321	-0.105112	0.136831	-0.087057	-0.025
max	2.750767	1.824593	3.757133	3.914658	6.401777	7.407188	5.444

```
# Logistic Regression
```

```
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
y_pred = lr.predict(X_test)
```

```
print(accuracy_score(y_train, lr.predict(X_train)))
lr_acc = accuracy_score(y_test, lr.predict(X_test))
print(lr_acc)
print(confusion_matrix(y_test, lr.predict(X_test)))
print(classification_report(y_test, lr.predict(X_test)))
```

```
0.7117516629711752
0.7699115044247787
[[11 21]
 [ 5 76]]
              precision    recall  f1-score   support

           0       0.69      0.34      0.46        32
           1       0.78      0.94      0.85        81

    accuracy                           0.77       113
   macro avg       0.74      0.64      0.66       113
weighted avg       0.76      0.77      0.74       113
```

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
knn.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
knn.predict(X_test)
```

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

```
print(accuracy_score(y_train, lr.predict(X_train)))
knn_acc = accuracy_score(y_test, knn.predict(X_test))
print(knn_acc)
print(confusion_matrix(y_test, knn.predict(X_test)))
print(classification_report(y_test, knn.predict(X_test)))
```

```
0.7117516629711752
0.6637168141592921
[[16 16]
 [22 59]]
              precision    recall  f1-score   support

           0       0.42      0.50      0.46        32
           1       0.79      0.73      0.76        81

    accuracy                           0.66       113
   macro avg       0.60      0.61      0.61       113
weighted avg       0.68      0.66      0.67       113
```

```
svc= SVC(probability=True)
parameters = {
    'gamma':[0.0001, 0.001, 0.01, 0.1],
    'C':[0.01, 0.05, 0.5, 0.1, 1, 10, 15, 20, 30]
}
grid_search = GridSearchCV(svc, parameters)
grid_search.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

```
grid_search.best_params_
```

```
{'C': 0.01, 'gamma': 0.0001}
```

```
grid_search.best_score_
```

```
0.7117460317460318
```

```
svc= SVC(C=0.01, gamma=0.0001,probability=True)
svc.fit(X_train, y_train)
```

```
▾                    SVC
SVC(C=0.01, gamma=0.0001, probability=True)
```

```
print(accuracy_score(y_train, svc.predict(X_train)))
svc_acc = accuracy_score(y_test, svc.predict(X_test))
print(svc_acc)
print(confusion_matrix(y_test, svc.predict(X_test)))
print(classification_report(y_test, svc.predict(X_test)))
```

```
0.7117516629711752
0.7168141592920354
```

```
[[ 0 32]
 [ 0 81]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        32
           1       0.72      1.00      0.84        81

    accuracy                           0.72       113
   macro avg       0.36      0.50      0.42       113
weighted avg       0.51      0.72      0.60       113
```

```
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
print(accuracy_score(y_train, dtc.predict(X_train)))
dtc_acc = accuracy_score(y_test, dtc.predict(X_test))
print(dtc_acc)
print(confusion_matrix(y_test, dtc.predict(X_test)))
print(classification_report(y_test, dtc.predict(X_test)))
```

```
1.0
0.6371681415929203
[[17 15]
 [26 55]]
              precision    recall  f1-score   support

           0       0.40      0.53      0.45        32
           1       0.79      0.68      0.73        81

    accuracy                           0.64       113
   macro avg       0.59      0.61      0.59       113
weighted avg       0.68      0.64      0.65       113
```

```
grid_parameter = {
    'criterion':['gini','entropy'],
    'max_depth':[3,5,7,10,12,15],
    'splitter':['best','random'],
    'min_samples_leaf':[1,2,3,5,7],
    'min_samples_split':[1,2,3,5,7],
    'max_features':['auto','sqrt','log2']
}
grid_seach_dt = GridSearchCV(dtc, grid_parameter, cv=24, n_jobs=-1, verbose=1)
grid_seach_dt.fit(X_train, y_train)
```

```
Fitting 24 folds for each of 1800 candidates, totalling 43200 fits
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

```
grid_seach_dt.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 3,
 'max_features': 'log2',
 'min_samples_leaf': 5,
 'min_samples_split': 2,
 'splitter': 'best'}
```

```
grid_seach_dt.best_score_
```

```
0.7443957115009746
```

```
dtc = DecisionTreeClassifier(criterion='entropy', max_depth=5, max_features='sqrt', min_samples_leaf=7, min_samples_split=3, splitter='best'
dtc.fit(X_train, y_train)
```

```
▾                         DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=5, max_features='sqrt',
                       min_samples_leaf=7, min_samples_split=3)
```

```
print(accuracy_score(y_train, dtc.predict(X_train)))
dtc_acc = accuracy_score(y_test, dtc.predict(X_test))
print(dtc_acc)
print(confusion_matrix(y_test, dtc.predict(X_test)))
print(classification_report(y_test, dtc.predict(X_test)))
```

```
0.7516629711751663
0.6902654867256637
[[ 7 25]
 [10 71]]
              precision    recall  f1-score   support

           0       0.41      0.22      0.29        32
           1       0.74      0.88      0.80        81

    accuracy                           0.69       113
   macro avg       0.58      0.55      0.54       113
weighted avg       0.65      0.69      0.66       113
```

```
# Random  Forest
```

```
rand_clf = RandomForestClassifier(criterion='entropy', max_depth=15, max_features=0.75, min_samples_leaf=7, min_samples_split=3, n_estimator
```

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

```
▾                         RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=15, max_features=0.75,
                       min_samples_leaf=7, min_samples_split=3,
                       n_estimators=130)
```

```
print(accuracy_score(y_train, rand_clf.predict(X_train)))
rand_clf_acc = accuracy_score(y_test, rand_clf.predict(X_test))
print(rand_clf_acc)
print(confusion_matrix(y_test, rand_clf.predict(X_test)))
print(classification_report(y_test, rand_clf.predict(X_test)))
```

```
0.8980044345898004
0.6902654867256637
[[13 19]
 [16 65]]
              precision    recall  f1-score   support

           0       0.45      0.41      0.43        32
           1       0.77      0.80      0.79        81

    accuracy                           0.69       113
   macro avg       0.61      0.60      0.61       113
weighted avg       0.68      0.69      0.69       113
```

```
# Gradient Boosting Classifier
```

```
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier()

parameters = {
    'loss': ['deviance', 'exponential'],
    'learning_rate': [0.001, 0.1, 1, 10],
    'n_estimators': [100, 150, 180, 200]
}

grid_search_gbc = GridSearchCV(gbc, parameters, cv = 20, n_jobs = -1, verbose = 1)
grid_search_gbc.fit(X_train, y_train)
```

```
Fitting 20 folds for each of 32 candidates, totalling 640 fits
```

```
▸ GridSearchCV
▸ estimator: GradientBoostingClassifier
  ▸ GradientBoostingClassifier
```

```
grid_search_gbc.best_params_
```

```
{'learning_rate': 0.001, 'loss': 'deviance', 'n_estimators': 100}
```

```
grid_search_gbc.best_score_
```

```
0.7120553359683793
```

```
gbc = GradientBoostingClassifier(learning_rate=0.001, loss='exponential',n_estimators=100)
gbc.fit(X_train , y_train)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.001, loss='exponential')
```

```
print(accuracy_score(y_train, gbc.predict(X_train)))
gbc_acc = accuracy_score(y_test, gbc.predict(X_test))
print(gbc_acc)
print(confusion_matrix(y_test, gbc.predict(X_test)))
print(classification_report(y_test, gbc.predict(X_test)))
```

```
0.7117516629711752
0.7168141592920354
[[ 0 32]
 [ 0 81]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        32
           1       0.72      1.00      0.84        81

    accuracy                           0.72       113
   macro avg       0.36      0.50      0.42       113
weighted avg       0.51      0.72      0.60       113
```

```
# XGBoost
```

```
from xgboost import XGBClassifier
```

```
xgb = XGBClassifier(objective='binary:logistic', learning_rate = 0.001, max_depth = 100, n_estimators = 300)
xgb.fit(X_train, y_train)
```

```
▾ XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.001, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=100, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=300, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

```
print(accuracy_score(y_train, xgb.predict(X_train)))
xgb_acc = accuracy_score(y_test, xgb.predict(X_test))
print(xgb_acc)
print(confusion_matrix(y_test, xgb.predict(X_test)))
print(classification_report(y_test, xgb.predict(X_test)))
```

```
0.7117516629711752
0.7168141592920354
[[ 0 32]
 [ 0 81]]
              precision    recall  f1-score   support
```

```
              0       0.00      0.00      0.00        32
              1       0.72      1.00      0.84        81

       accuracy                           0.72       113
      macro avg       0.36      0.50      0.42       113
   weighted avg       0.51      0.72      0.60       113
```

```
# Model Comparison
```

```
models = pd.DataFrame({
    'Model':['Logistic Regreesion','KNN', 'SVC', 'Decision Tree Classifier', 'Random Forest Classifier', 'Gradient Boosting Classifer', 'XgB
    'Score':[100*round(lr_acc, 4), 100*round(knn_acc, 4), 100*round(svc_acc, 4), 100*round(dtc_acc, 4), 100*round(rand_clf_acc, 4), 100*rour
})
```

```
models.sort_values(by='Score', ascending=False)
```

	Model	Score
0	Logistic Regreesion	76.99
2	SVC	71.68
5	Gradient Boosting Classifer	71.68
6	XgBoost	71.68
3	Decision Tree Classifier	69.03
4	Random Forest Classifier	69.03
1	KNN	66.37

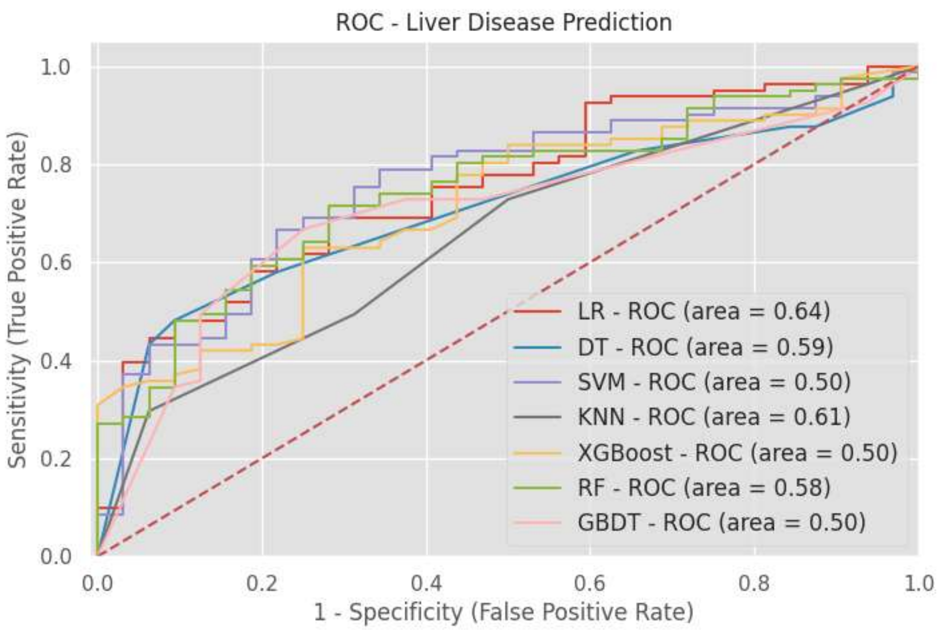
```
import pickle
model = lr_acc
pickle.dump(model, open("liver.pkl","wb"))
```

```
# 85%
# ANN
```

```
from sklearn import metrics
plt.figure(figsize=(8,5))
models = [
{
    'label': 'LR',
    'model': lr,
},
{
    'label': 'DT',
    'model': dtc,
},
{
    'label': 'SVM',
    'model': svc,
},
{
    'label': 'KNN',
    'model': knn,
},
{
    'label': 'XGBoost',
    'model': xgb,
},
{
    'label': 'RF',
    'model': rand_clf,
},
{
    'label': 'GBDT',
    'model': gbc,
}
]
for m in models:
    model = m['model']
    model.fit(X_train, y_train)
```

```
    y_pred=model.predict(X_test)
    fpr1, tpr1, thresholds = metrics.roc_curve(y_test, model.predict_proba(X_test)[:,1])
    auc = metrics.roc_auc_score(y_test,model.predict(X_test))
    plt.plot(fpr1, tpr1, label='%s - ROC (area = %0.2f)' % (m['label'], auc))

plt.plot([0, 1], [0, 1],'r--')
plt.xlim([-0.01, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1 - Specificity (False Positive Rate)', fontsize=12)
plt.ylabel('Sensitivity (True Positive Rate)', fontsize=12)
plt.title('ROC - Liver Disease Prediction', fontsize=12)
plt.legend(loc="lower right", fontsize=12)
plt.savefig("roc_liver.jpeg", format='jpeg', dpi=400, bbox_inches='tight')
plt.show()
```




```
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
models = [
{
    'label': 'LR',
    'model': lr,
},
{
    'label': 'DT',
    'model': dtc,
},
```