```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
```

```
liver_df= pd.read_csv('/content/indian_liver_patient.csv')
liver_df.head()
```

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

Next steps: Generate code with liver_df View recommended plots

```
liver_df.head()
```

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

Next steps: Generate code with liver_df View recommended plots

```
liver_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
```

```
liver_df.describe(include='all')
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Ala
count	583.000000	583	583.000000	583.000000	583.000000	
unique	NaN	2	NaN	NaN	NaN	
top	NaN	Male	NaN	NaN	NaN	
freq	NaN	441	NaN	NaN	NaN	
mean	44.746141	NaN	3.298799	1.486106	290.576329	
std	16.189833	NaN	6.209522	2.808498	242.937989	
min	4.000000	NaN	0.400000	0.100000	63.000000	
25%	33.000000	NaN	0.800000	0.200000	175.500000	
50%	45.000000	NaN	1.000000	0.300000	208.000000	
75%	58.000000	NaN	2.600000	1.300000	298.000000	
max	90.000000	NaN	75.000000	19.700000	2110.000000	

```
liver_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')
```

```
liver_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
```

```
pd.get_dummies(liver_df['Gender'], prefix = 'Gender').head()
```

	Gender_Female	Gender_Male
0	True	False
1	False	True
2	False	True
3	False	True
4	False	True

```
liver_df = pd.concat([liver_df,pd.get_dummies(liver_df['Gender'], prefix = 'Gender')], axis
```

```
liver_df.head()
```

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

Next steps: Generate code with `liver_df` View recommended plots

```
liver_df.describe()
```

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Ami
count	583.000000	583.000000	583.000000	583.000000	
mean	44.746141	3.298799	1.486106	290.576329	
std	16.189833	6.209522	2.808498	242.937989	
min	4.000000	0.400000	0.100000	63.000000	
25%	33.000000	0.800000	0.200000	175.500000	
50%	45.000000	1.000000	0.300000	208.000000	
75%	58.000000	2.600000	1.300000	298.000000	
max	90.000000	75.000000	19.700000	2110.000000	

```
liver_df[liver_df['Albumin_and_Globulin_Ratio'].isnull()]
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Amin
209	45	Female	0.9	0.3	189	
241	51	Male	0.8	0.2	230	
253	35	Female	0.6	0.2	180	
312	27	Male	1.3	0.6	106	

```
liver_df["Albumin_and_Globulin_Ratio"] = liver_df.Albumin_and_Globulin_Ratio.fillna(liver_d
```

```
X = liver_df.drop(['Gender','Dataset'], axis=1)
X.head(3)
```

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransfera
0	65	0.7	0.1	187	
1	62	10.9	5.5	699	
2	62	7.3	4.1	490	

Next steps: Generate code with X View recommended plots

```
y = liver_df['Dataset']
```

```
liver_corr = X.corr()
liver_corr
```

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosp
Age	1.000000	0.011763	0.007529	0.
Total_Bilirubin	0.011763	1.000000	0.874618	0.
Direct_Bilirubin	0.007529	0.874618	1.000000	0.
Alkaline_Phosphotase	0.080425	0.206669	0.234939	1.
Alamine_Aminotransferase	-0.086883	0.214065	0.233894	0.
Aspartate_Aminotransferase	-0.019910	0.237831	0.257544	0.
Total_Protiens	-0.187461	-0.008099	-0.000139	-0.
Albumin	-0.265924	-0.222250	-0.228531	-0.
Albumin_and_Globulin_Ratio	-0.216089	-0.206159	-0.200004	-0.
Gender_Female	-0.056560	-0.089291	-0.100436	0.
Gender_Male	0.056560	0.089291	0.100436	-0.

Next steps: Generate code with `liver_corr` View recommended plots

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, BaggingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=101)
print (X_train.shape)
print (y_train.shape)
print (X_test.shape)
print (y_test.shape)
```

```
(408, 11)
(408,)
(175, 11)
```

```
(175,)
```

Logistic Regression

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

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Converge
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
  n_iter_i = _check_optimize_result(
 ▾ LogisticRegression
 LogisticRegression()
```

```
log_predicted= logreg.predict(X_test)
logreg_score = round(logreg.score(X_train, y_train) * 100, 2)
logreg_score_test = round(logreg.score(X_test, y_test) * 100, 2)


print('Logistic Regression Training Score: \n', logreg_score)
print('Logistic Regression Test Score: \n', logreg_score_test)
print('Coefficient: \n', logreg.coef_)
print('Intercept: \n', logreg.intercept_)
print('Accuracy: \n', accuracy_score(y_test,log_predicted))
print('Confusion Matrix: \n', confusion_matrix(y_test,log_predicted))
print('Classification Report: \n', classification_report(y_test,log_predicted))
```

```
Logistic Regression Training Score:
 72.06
Logistic Regression Test Score:
 68.0
Coefficient:
 [[-0.00994992 -0.0985122  -0.30688724 -0.00082939 -0.01078827 -0.00275598
  -0.23899684  0.40208926  0.59475501  0.2533529   0.0911599 ]]
Intercept:
 [0.36100669]
Accuracy:
 0.68
Confusion Matrix:
 [[107  17]
 [ 39  12]]
Classification Report:
               precision    recall  f1-score   support

           1       0.73      0.86      0.79       124
           2       0.41      0.24      0.30        51
```

```
        accuracy                           0.68       175
       macro avg       0.57      0.55      0.55       175
    weighted avg       0.64      0.68      0.65       175
```

```
coeff_df = pd.DataFrame(X.columns)
coeff_df.columns = ['Feature']
coeff_df["Correlation"] = pd.Series(logreg.coef_[0])
pd.Series(logreg.coef_[0])

coeff_df.sort_values(by='Correlation', ascending=False)
```

	Feature	Correlation
8	Albumin_and_Globulin_Ratio	0.594755
7	Albumin	0.402089
9	Gender_Female	0.253353
10	Gender_Male	0.091160
3	Alkaline_Phosphotase	-0.000829
5	Aspartate_Aminotransferase	-0.002756
0	Age	-0.009950
4	Alamine_Aminotransferase	-0.010788
1	Total_Bilirubin	-0.098512
6	Total_Protiens	-0.238997
2	Direct_Bilirubin	-0.306887

Gaussian Naive Bayes

```
gaussian = GaussianNB()
gaussian.fit(X_train, y_train)
gauss_predicted = gaussian.predict(X_test)
```

```
gauss_score = round(gaussian.score(X_train, y_train) * 100, 2)
gauss_score_test = round(gaussian.score(X_test, y_test) * 100, 2)
print('Gaussian Score: \n', gauss_score)
print('Gaussian Test Score: \n', gauss_score_test)
print('Accuracy: \n', accuracy_score(y_test, gauss_predicted))
print(confusion_matrix(y_test,gauss_predicted))
print(classification_report(y_test,gauss_predicted))
```

```
Gaussian Score: 
 56.13
Gaussian Test Score: 
 53.14
Accuracy: 
 0.5314285714285715
[[44 80]
 [ 2 49]]
              precision    recall  f1-score   support

           1       0.96      0.35      0.52       124
           2       0.38      0.96      0.54        51

    accuracy                           0.53       175
   macro avg       0.67      0.66      0.53       175
weighted avg       0.79      0.53      0.53       175
```

Random Forest

```
random_forest = RandomForestClassifier(max_depth=3,n_estimators=56,criterion='entropy')
random_forest.fit(X_train, y_train)
```

```
▾                         RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=3, n_estimators=56)
```

```
rf_predicted = random_forest.predict(X_test)
```

```
random_forest_score = round(random_forest.score(X_train, y_train) * 100, 2)
random_forest_score_test = round(random_forest.score(X_test, y_test) * 100, 2)
print('Random Forest Score: \n', random_forest_score)
print('Random Forest Test Score: \n', random_forest_score_test)
print('Accuracy: \n', accuracy_score(y_test,rf_predicted))
print(confusion_matrix(y_test,rf_predicted))
print(classification_report(y_test,rf_predicted))
```

```
Random Forest Score: 
 76.23
Random Forest Test Score: 
 70.29
Accuracy: 
 0.7028571428571428
[[117   7]
 [ 45   6]]
              precision    recall  f1-score   support

           1       0.72      0.94      0.82       124
           2       0.46      0.12      0.19        51

    accuracy                           0.70       175
```

```
       macro avg       0.59      0.53      0.50       175
    weighted avg       0.65      0.70      0.63       175
```

```
finX = liver_df[['Total_Protiens','Albumin', 'Gender_Male']]
finX.head(4)
```

	Total_Protiens	Albumin	Gender_Male
0	6.8	3.3	False
1	7.5	3.2	True
2	7.0	3.3	True
3	6.8	3.4	True

Next steps: Generate code with finX View recommended plots

Logistic Regression

```
X_train, X_test, y_train, y_test = train_test_split(finX, y, test_size=0.30, random_state=1
```

```
logreg = LogisticRegression()
```

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

```
▾ LogisticRegression
LogisticRegression()
```

```
log_predicted= logreg.predict(X_test)
```

```
logreg_score = round(logreg.score(X_train, y_train) * 100, 2)
logreg_score_test = round(logreg.score(X_test, y_test) * 100, 2)

# Equation coefficient and Intercept

print('Logistic Regression Training Score: \n', logreg_score)
print('Logistic Regression Test Score: \n', logreg_score_test)
print('Coefficient: \n', logreg.coef_)
print('Intercept: \n', logreg.intercept_)
print('Accuracy: \n', accuracy_score(y_test,log_predicted))
print('Confusion Matrix: \n', confusion_matrix(y_test,log_predicted))
print('Classification Report: \n', classification_report(y_test,log_predicted))
```

```
Logistic Regression Training Score:
 71.08
Logistic Regression Test Score:
 71.43
Coefficient:
 [[-0.58254377  1.08940697 -0.54185122]]
Intercept:
 [-0.20423275]
Accuracy:
 0.7142857142857143
Confusion Matrix:
 [[120   4]
 [ 46   5]]
Classification Report:
               precision    recall  f1-score   support

           1       0.72      0.97      0.83       124
           2       0.56      0.10      0.17        51

    accuracy                           0.71       175
   macro avg       0.64      0.53      0.50       175
weighted avg       0.67      0.71      0.63       175
```

Decision Tree Classifier

```
dt=DecisionTreeClassifier()
```

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

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

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

```
dt_score = round(dt.score(X_train, y_train) * 100, 2)
dt_test = round(dt.score(X_test, y_test) * 100, 2)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

```
0.6457142857142857
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
```

```
array([[100,  24],
       [ 38,  13]])
```

Model evaluation

```
models = pd.DataFrame({
    'Model': [ 'Logistic Regression', 'Gaussian Naive Bayes','Random Forest','Decision Tree']
    'Score': [ logreg_score, gauss_score, random_forest_score,dt_score],
    'Test Score': [ logreg_score_test, gauss_score_test, random_forest_score_test,dt_test]})
models.sort_values(by='Test Score', ascending=False)
```

```
                  Model  Score  Test Score
0   Logistic Regression  71.08       71.43
2         Random Forest  76.23       70.29
3         Decision Tree  93.38       64.57
1  Gaussian Naive Bayes  56.13       53.14
```

```
import pickle
```

```
filename = 'liver.sav'
pickle.dump(round, open(filename, 'wb'))
```

```
# loading the saved model
loaded_model = pickle.load(open('liver.sav', 'rb'))
```

Start coding or generate with AI.