import libraries

In [35]:
```
# Libraries for data handling and visualization
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Display plots in the notebook
%matplotlib inline
```

In [36]:
```
# Importing preprocessing and model selection tools
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# Importing machine learning models
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

# Importing evaluation metrics
from sklearn.metrics import precision_score, f1_score, recall_score
```

read and load the data

In [37]:
```
# Loading the liver patient dataset
patient = pd.read_csv("/content/indian_liver_patient.csv")

# Displaying number of samples and features
print(f"Total number of samples: {patient.shape[0]}. Total number of features in
```

```
Total number of samples: 583. Total number of features in each sample: 11.
```

In [6]:
```
patient.head()
```

Out[6]:

	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	

In [7]:
```
patient.tail()
```

Out[7]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Amin
578	60	Male	0.5	0.1	500	
579	40	Male	0.6	0.1	98	
580	52	Male	0.8	0.2	245	
581	31	Male	1.3	0.5	184	
582	38	Male	1.0	0.3	216	

In [9]:
```
patient.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
```

In [10]:
```
patient.describe()
```

Out[10]:

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Amin
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	

In [11]:
```
patient.shape
```

Out[11]: (583, 11)

In [12]:
```
#commence data preprocessing

#check to see if there are duplicates
''' Duplicates are removed as it's most likely these entries has been inputed tw
patient_duplicate = patient[patient.duplicated(keep = False)]
# keep = False gives you all rows with duplicate entries
patient_duplicate
```

Out[12]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Amin
18	40	Female	0.9	0.3	293	
19	40	Female	0.9	0.3	293	
25	34	Male	4.1	2.0	289	
26	34	Male	4.1	2.0	289	
33	38	Female	2.6	1.2	410	
34	38	Female	2.6	1.2	410	
54	42	Male	8.9	4.5	272	
55	42	Male	8.9	4.5	272	
61	58	Male	1.0	0.5	158	
62	58	Male	1.0	0.5	158	
105	36	Male	5.3	2.3	145	
106	36	Male	5.3	2.3	145	
107	36	Male	0.8	0.2	158	
108	36	Male	0.8	0.2	158	
137	18	Male	0.8	0.2	282	
138	18	Male	0.8	0.2	282	
142	30	Male	1.6	0.4	332	
143	30	Male	1.6	0.4	332	
157	72	Male	0.7	0.1	196	
158	72	Male	0.7	0.1	196	
163	39	Male	1.9	0.9	180	
164	39	Male	1.9	0.9	180	
173	31	Male	0.6	0.1	175	
174	31	Male	0.6	0.1	175	
200	49	Male	0.6	0.1	218	
201	49	Male	0.6	0.1	218	

In [13]:
```
patient = patient[~patient.duplicated(subset = None, keep = 'first')]
# Here, keep = 'first' ensures that only the first row is taken into the final d
# The '~' sign tells pandas to keep all values except the 13 duplicate values
patient.shape
```

Out[13]: (570, 11)

In [14]:
```
#checking if there are any NULL values in our Dataset
patient.isnull().values.any()
```

Out[14]: np.True_

In [15]:
```
#removing null values
# display number of null values by column# display number of null values by colu
print(data.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
```

In [16]:
```
# We can see that the column 'Albumin_and_Globulin_Ratio' has 4 missing values
# One way to deal with them can be to just directly remove these 4 values
print ("length before removing NaN values:%d"%len(patient))
patient_2 = patient[pd.notnull(patient['Albumin_and_Globulin_Ratio'])]
print ("length after removing NaN values:%d"%len(patient_2))
```

```
length before removing NaN values:570
length after removing NaN values:566
```

In [38]:
```
# Remove rows with missing values
new_patient = patient.dropna(axis=0, how='any')
```

In [39]:
```
# Verify removal of missing values
new_patient.isnull().values.any()
```

Out[39]: np.False_

In [19]:
```
patient.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 570 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Age                         570 non-null    int64
 1   Gender                      570 non-null    object
 2   Total_Bilirubin             570 non-null    float64
 3   Direct_Bilirubin            570 non-null    float64
 4   Alkaline_Phosphotase        570 non-null    int64
 5   Alamine_Aminotransferase    570 non-null    int64
 6   Aspartate_Aminotransferase  570 non-null    int64
 7   Total_Protiens              570 non-null    float64
 8   Albumin                     570 non-null    float64
 9   Albumin_and_Globulin_Ratio  566 non-null    float64
 10  Dataset                     570 non-null    int64
dtypes: float64(5), int64(5), object(1)
memory usage: 53.4+ KB
```

In [20]:
```
'''
The Albumin-Globulin Ratio feature has four missing values, as seen above. Here,

A constant value that has meaning within the domain, such as 0, distinct from al
A value from another randomly selected record, or the immediately next or previo
A mean, median or mode value for the column.
A value estimated by another predictive model.
But here, since a very small fraction of values are missing, we choose to drop t
'''
#Transform our data
le = preprocessing.LabelEncoder()
le.fit(['Male','Female'])
patient.loc[:,'Gender'] = le.transform(patient['Gender'])

#Remove rows with missing values
patient = patient.dropna(how = 'any', axis = 0)

#Also transform Selector variable into usual conventions followed
patient['Dataset'] = patient['Dataset'].map({2:0, 1:1})
```

In [21]:
```
patient.head()
```

Out[21]:

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

In [22]:
```
#features characteristics to determine if feature scaling is necessary
patient.describe()
```

Out[22]:

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Amin
count	566.000000	566.000000	566.000000	566.000000	
mean	44.886926	3.338869	1.505830	292.567138	
std	16.274893	6.286728	2.841485	245.936559	
min	4.000000	0.400000	0.100000	63.000000	
25%	33.000000	0.800000	0.200000	176.000000	
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	

In [23]:
```
#split the data into test and train samples
X_train, X_test, y_train, y_test = train_test_split(patient, patient['Dataset'],
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

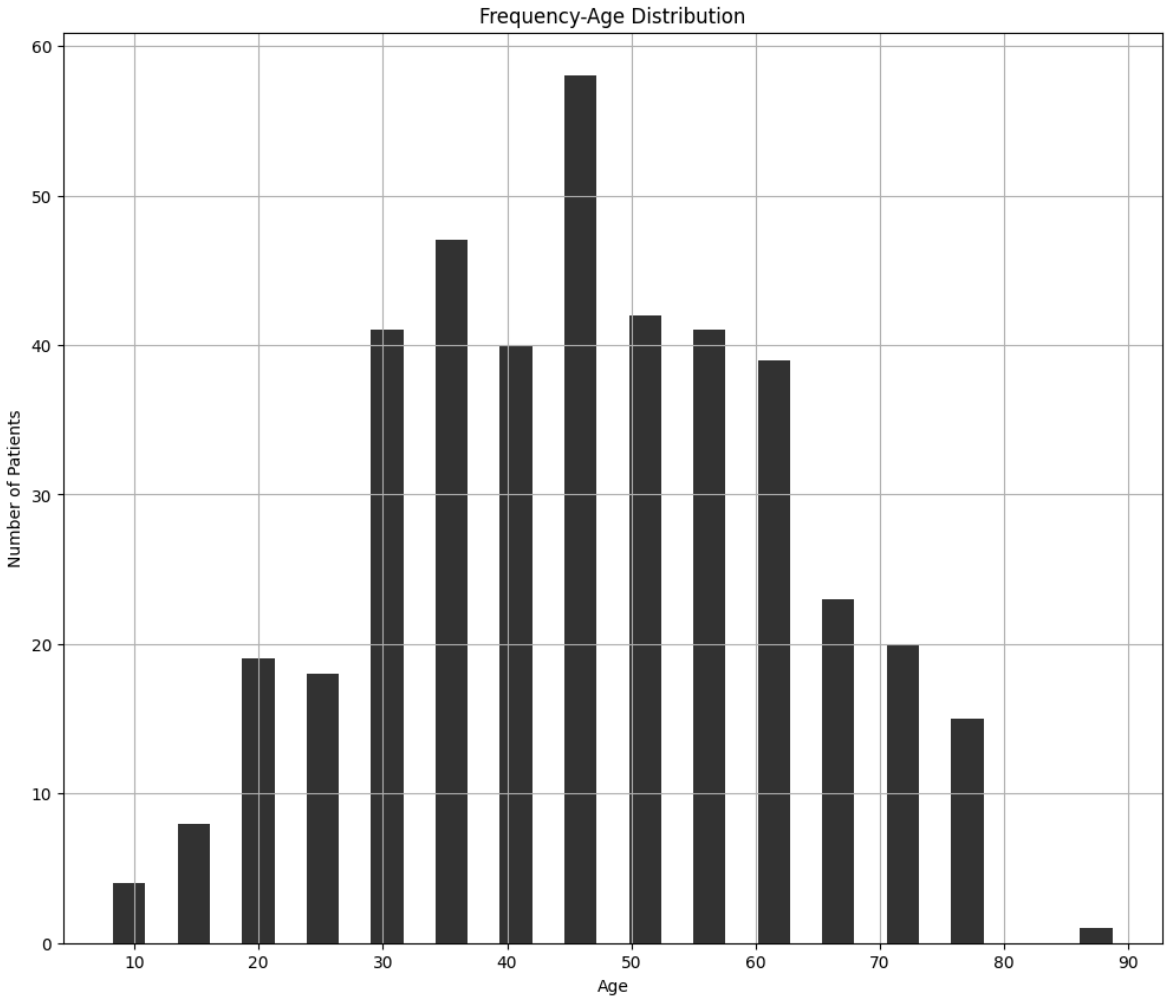
In [24]:
```
#Exploratory Data Analysis

#Determining the healthy-affected split
print("Positive records:", patient['Dataset'].value_counts().iloc[0])
print("Negative records:", patient['Dataset'].value_counts().iloc[1])
```

```
Positive records: 404
Negative records: 162
```

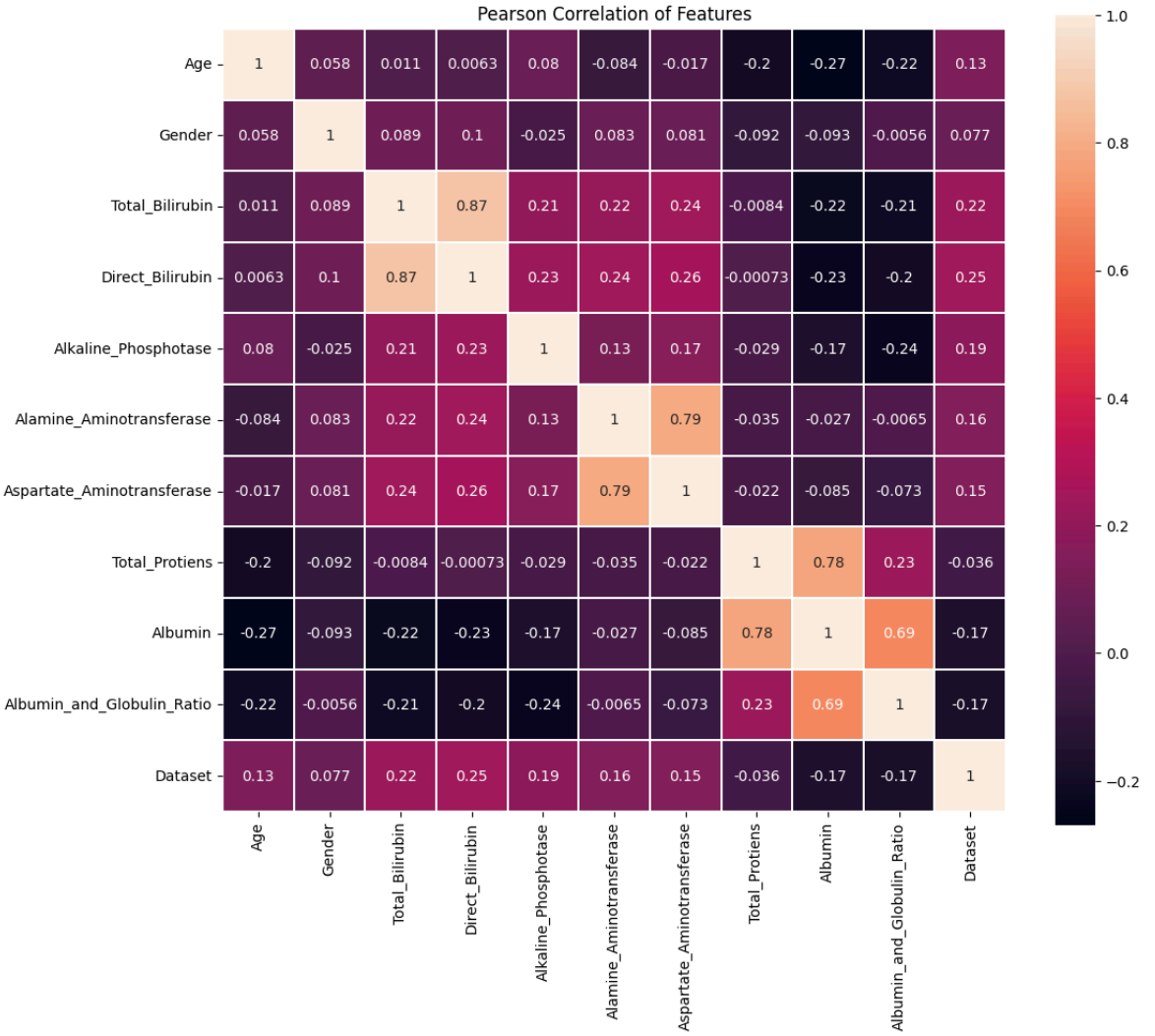
In [25]:
```
#The above output confirms that we have 414 positive and 165 negative records. T

#Determine statistics based on age
plt.figure(figsize=(12, 10))
plt.hist(patient[patient['Dataset'] == 1]['Age'], bins = 16, align = 'mid', rwid
plt.xlabel('Age')
plt.ylabel('Number of Patients')
plt.title('Frequency-Age Distribution')
plt.grid(True)
plt.savefig('fig1')
plt.show()
```



In [27]:
```
#Looking at the age vs. frequency graph, we can observe that middle-aged people
#as seen by the bar sizes at ages 60-80.

#correlation-matrix
plt.subplots(figsize=(12, 10))
plt.title('Pearson Correlation of Features')
# Draw the heatmap using seaborn
sns.heatmap(patient.corr(),linewidths=0.25, vmax=1.0, square=True,annot=True)
plt.savefig('fig2')
plt.show()
```



In [28]:
```
'''
The correlation matrix gives us the relationship between two features. As seen a

1.Total Bilirubin and Direct Bilirubin(0.87)
2.Sgpt Alamine Aminotransferase and Sgot Aspartate Aminotransferase(0.79)
3.Albumin and Total Proteins(0.78)
4.Albumin and Albumin-Globulin Ratio(0.69)
'''

'''
#Using Classification Algorithms
Let us now evaluate the performance of various classifiers on this dataset.
For the sake of understanding as to how feature scaling affects classifier perfo
we will train models using both scaled and unscaled data.
Since we are interested in capturing records of people who have been tested posi
we will base our classifier evaluation metric on precision and recall instead of
We could also use F1 score, since it takes into account both precision and recal
'''

#Logistic Regression: Using normal data
logreg = LogisticRegression(C = 0.1).fit(X_train, y_train)
print("Logistic Regression Classifier on unscaled test data:")
print("Accuracy:", logreg.score(X_test, y_test))
print("Precision:", precision_score(y_test, logreg.predict(X_test)))
print("Recall:", recall_score(y_test, logreg.predict(X_test)))
print("F-1 score:", f1_score(y_test, logreg.predict(X_test)))
```

```
Logistic Regression Classifier on unscaled test data:
Accuracy: 0.9859154929577465
Precision: 0.9805825242718447
Recall: 1.0
F-1 score: 0.9901960784313726
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:465: Co
nvergenceWarning: lbfgs failed to converge (status=1):
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(
```

In [29]:
```
#Using feature-scaled data
logreg_scaled = LogisticRegression(C = 0.1).fit(X_train_scaled, y_train)
print("Logistic Regression Classifier on scaled test data:")
print("Accuracy:", logreg_scaled.score(X_test_scaled, y_test))
print("Precision:", precision_score(y_test, logreg_scaled.predict(X_test_scaled)
print("Recall:", recall_score(y_test, logreg_scaled.predict(X_test_scaled)))
print("F-1 score:", f1_score(y_test, logreg_scaled.predict(X_test_scaled)))
```

```
Logistic Regression Classifier on scaled test data:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F-1 score: 1.0
```

In [30]:
```
'''
Well! The performance has definitely improved by feature scaling, though not dra
as there was already very little scope of improvement.
Let us look at other classifiers and analyse how they react to scaling.
'''

#SVM Classifier with RBF kernel: Using normal data
svc_clf = SVC(C = 0.1, kernel = 'rbf').fit(X_train, y_train)
print("SVM Classifier on unscaled test data:")
print("Accuracy:", svc_clf.score(X_test, y_test))
print("Precision:", precision_score(y_test, svc_clf.predict(X_test)))
print("Recall:", recall_score(y_test, svc_clf.predict(X_test)))
print("F-1 score:", f1_score(y_test, svc_clf.predict(X_test)))
```

```
SVM Classifier on unscaled test data:
Accuracy: 0.7112676056338029
Precision: 0.7112676056338029
Recall: 1.0
F-1 score: 0.831275720164609
```

In [31]:
```
#Using scaled data
svc_clf_scaled = SVC(C = 0.1, kernel = 'rbf').fit(X_train_scaled, y_train)
print("SVM Classifier on scaled test data:")
print("Accuracy:", svc_clf_scaled.score(X_test_scaled, y_test))
print("Precision:", precision_score(y_test, svc_clf_scaled.predict(X_test_scaled
print("Recall:", recall_score(y_test, svc_clf_scaled.predict(X_test_scaled)))
print("F-1 score:", f1_score(y_test, svc_clf_scaled.predict(X_test_scaled)))
```

```
SVM Classifier on scaled test data:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F-1 score: 1.0
```

In [33]:
```
#Random Forest Classifier: using normal data
rfc = RandomForestClassifier(n_estimators = 20)
rfc.fit(X_train, y_train)
print("SVM Classifier on unscaled test data:")
print("Accuracy:", rfc.score(X_test, y_test))
print("Precision:", precision_score(y_test, rfc.predict(X_test)))
print("Recall:", recall_score(y_test, rfc.predict(X_test)))
print("F-1 score:", f1_score(y_test, rfc.predict(X_test)))
```

```
SVM Classifier on unscaled test data:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F-1 score: 1.0
```

In [34]:
```
#using scaled data
rfc_scaled = RandomForestClassifier(n_estimators = 20)
rfc_scaled.fit(X_train_scaled, y_train)
print("Random Forest Classifier on scaled test data:")
print("Accuracy:", rfc_scaled.score(X_test_scaled, y_test))
print("Precision:", precision_score(y_test, rfc_scaled.predict(X_test_scaled)))
print("Recall:", recall_score(y_test, rfc_scaled.predict(X_test_scaled)))
print("F-1 score:", f1_score(y_test, rfc_scaled.predict(X_test_scaled)))
```

```
Random Forest Classifier on scaled test data:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F-1 score: 1.0
```