TITLE : LIVER DISEASE DETECTOR.

READING THE DATASET:

```
import pandas as pd
a=pd.read_csv("/content/liver.csv")
print(a)
```

```
          Age  Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
     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
     ..   ...     ...              ...               ...                   ...
     578   60       1              0.5               0.1                   500
     579   40       1              0.6               0.1                    98
     580   52       1              0.8               0.2                   245
     581   31       1              1.3               0.5                   184
     582   38       1              1.0               0.3                   216

          Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens  \
     0                          16                          18             6.8
     1                          64                         100             7.5
     2                          60                          68             7.0
     3                          14                          20             6.8
     4                          27                          59             7.3
     ..                        ...                         ...             ...
     578                        20                          34             5.9
     579                        35                          31             6.0
     580                        48                          49             6.4
     581                        29                          32             6.8
     582                        21                          24             7.3

          Albumin  Albumin_and_Globulin_Ratio  Result
     0        3.3                        0.90       0
     1        3.2                        0.74       0
     2        3.3                        0.89       0
     3        3.4                        1.00       0
     4        2.4                        0.40       0
     ..       ...                         ...     ...
     578      1.6                        0.37       1
     579      3.2                        1.10       0
     580      3.2                        1.00       0
     581      3.4                        1.00       0
     582      4.4                        1.50       1

     [583 rows x 11 columns]
```

```
import pandas as pd

# Assuming the file is named "liver_data.csv", change it accordingly if your file has a different name
file_path = '/content/liver.csv'  # Adjust the file path accordingly

# Read the dataset into a Pandas DataFrame
df = pd.read_csv(file_path)

# Display the first few rows of the DataFrame to ensure it has been read correctly
df.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotr
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	

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report

# Importing various classifiers from scikit-learn
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
# Assuming the dataset is saved as 'liver_data.csv' in your current directory
df = pd.read_csv('/content/liver.csv')

# Display the first few rows to confirm data is loaded correctly
print(df.head())

# Splitting the dataset into features and target
X = df.drop('Result', axis=1)
y = df['Result']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Creating a dictionary of models
models = {
    'Logistic Regression': LogisticRegression(),
    'SVM': SVC(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'Naive Bayes': GaussianNB()
}

# Function to train and evaluate models
def train_and_evaluate(models, X_train, X_test, y_train, y_test):
    for name, model in models.items():
        # Training the model
        model.fit(X_train, y_train)
        # Making predictions
        y_pred = model.predict(X_test)
        # Evaluating the model
        accuracy = accuracy_score(y_test, y_pred)
        print(f"{name} Accuracy: {accuracy:.4f}")
        print(f"Classification Report for {name}:\n{classification_report(y_test, y_pred)}\n")

# Train and evaluate all models
train_and_evaluate(models, X_train, X_test, y_train, y_test)
```

```
   Age  Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
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

   Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens  \
0                        16                          18             6.8
1                        64                         100             7.5
2                        60                          68             7.0
3                        14                          20             6.8
4                        27                          59             7.3

   Albumin  Albumin_and_Globulin_Ratio  Result
0      3.3                        0.90       0
1      3.2                        0.74       0
2      3.3                        0.89       0
3      3.4                        1.00       0
4      2.4                        0.40       0
```

```
Logistic Regression Accuracy: 0.7607
Classification Report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.79      0.93      0.85        87
           1       0.57      0.27      0.36        30

    accuracy                           0.76       117
   macro avg       0.68      0.60      0.61       117
weighted avg       0.73      0.76      0.73       117


SVM Accuracy: 0.7436
Classification Report for SVM:
              precision    recall  f1-score   support

           0       0.74      1.00      0.85        87
           1       0.00      0.00      0.00        30

    accuracy                           0.74       117
   macro avg       0.37      0.50      0.43       117
weighted avg       0.55      0.74      0.63       117


K-Nearest Neighbors Accuracy: 0.6581
Classification Report for K-Nearest Neighbors:
              precision    recall  f1-score   support

           0       0.81      0.71      0.76        87
           1       0.38      0.50      0.43        30

    accuracy                           0.66       117
   macro avg       0.59      0.61      0.59       117
weighted avg       0.69      0.66      0.67       117


Decision Tree Accuracy: 0.7179
```

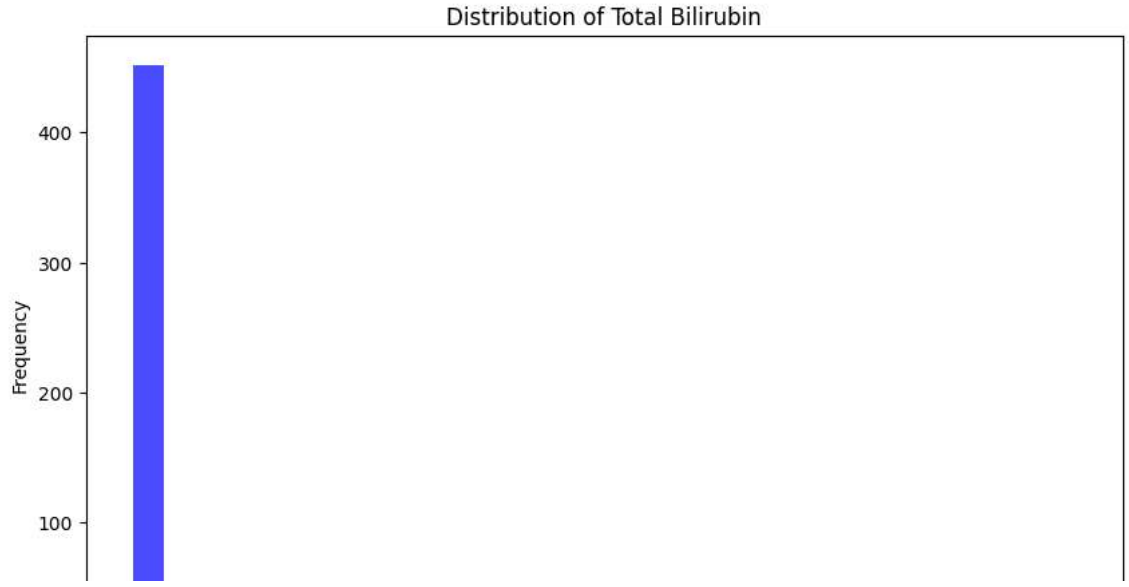
CLASSIFICATION ALGORITHMS IN MACHINE LEARNING:

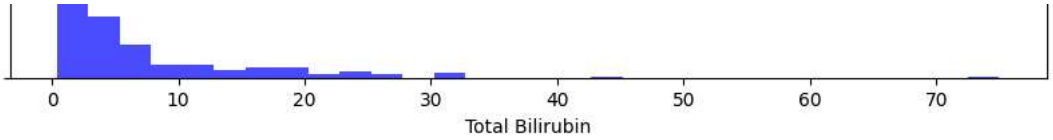
1.SUPPORT VECTOR MACHINES(SVM) MODEL:

1.A)ACCURACY:

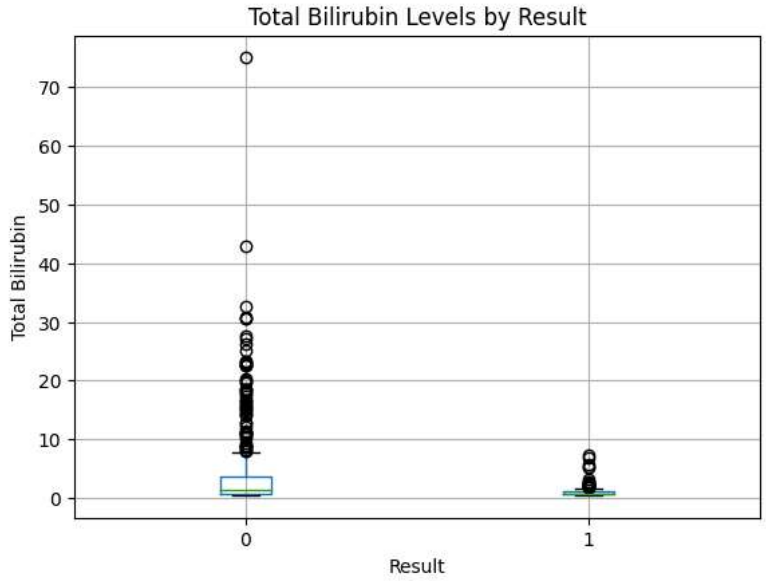
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataframe
print(data.head())

# Basic statistics of the dataset
print(data.describe())

# Visualize the distribution of Total Bilirubin
plt.figure(figsize=(10, 6))
plt.hist(data['Total_Bilirubin'], bins=30, color='blue', alpha=0.7)
plt.title('Distribution of Total Bilirubin')
plt.xlabel('Total Bilirubin')
plt.ylabel('Frequency')
plt.show()

# Box plot for comparing Total Bilirubin across Result categories (0 and 1)
plt.figure(figsize=(10, 6))
data.boxplot(column='Total_Bilirubin', by='Result')
plt.title('Total Bilirubin Levels by Result')
plt.suptitle('')
plt.xlabel('Result')
plt.ylabel('Total Bilirubin')
plt.show()

# Scatter plot of Total Bilirubin vs. Direct Bilirubin
plt.figure(figsize=(10, 6))
plt.scatter(data['Total_Bilirubin'], data['Direct_Bilirubin'], alpha=0.5)
plt.title('Total Bilirubin vs Direct Bilirubin')
plt.xlabel('Total Bilirubin')
plt.ylabel('Direct Bilirubin')
plt.grid(True)
plt.show()

# Prepare data for SVM
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a Support Vector Machine classifier
model = SVC(kernel='linear')

# Train the model
model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = model.predict(X_test_scaled)

# Print accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
print(classification_report(y_test, y_pred))
```

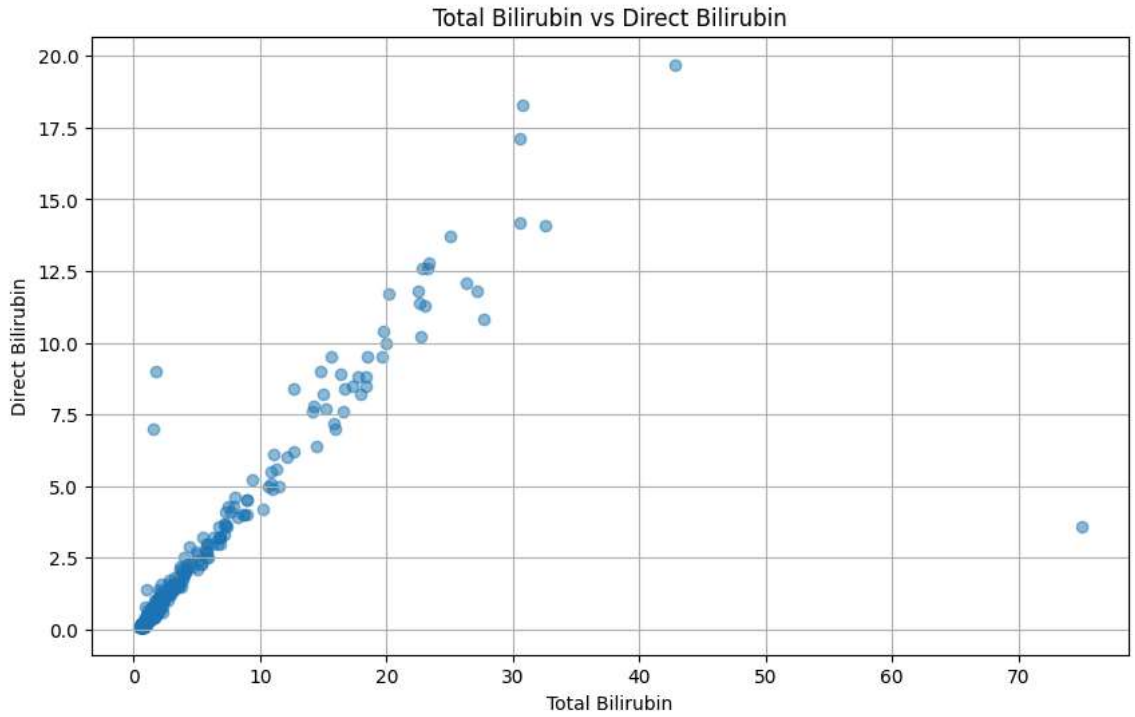
```
   Age  Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
0   65       0              0.7               0.1                   187
1   62       1             10.9               5.5                   699
2   62       1              7.3               4.1                   490
3   58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

   Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens  \
0                        16                          18             6.8
1                        64                         100             7.5
2                        60                          68             7.0
3                        14                          20             6.8
4                        27                          59             7.3

   Albumin  Albumin_and_Globulin_Ratio  Result
0      3.3                        0.90       0
1      3.2                        0.74       0
2      3.3                        0.89       0
3      3.4                        1.00       0
4      2.4                        0.40       0
              Age      Gender  Total_Bilirubin  Direct_Bilirubin  \
count  583.000000  583.000000       583.000000        583.000000
mean    44.746141    0.756432         3.298799          1.486106
std     16.189833    0.429603         6.209522          2.808498
min      4.000000    0.000000         0.400000          0.100000
25%     33.000000    1.000000         0.800000          0.200000
50%     45.000000    1.000000         1.000000          0.300000
75%     58.000000    1.000000         2.600000          1.300000
max     90.000000    1.000000        75.000000         19.700000

       Alkaline_Phosphotase  Alamine_Aminotransferase  \
count            583.000000                583.000000
mean             290.576329                 80.713551
std              242.937989                182.620356
min               63.000000                 10.000000
25%              175.500000                 23.000000
50%              208.000000                 35.000000
75%              298.000000                 60.500000
max             2110.000000               2000.000000

       Aspartate_Aminotransferase  Total_Protiens     Albumin  \
count                  583.000000      583.000000  583.000000
mean                   109.910806        6.483190    3.141852
std                    288.918529        1.085451    0.795519
min                     10.000000        2.700000    0.900000
25%                     25.000000        5.800000    2.600000
50%                     42.000000        6.600000    3.100000
75%                     87.000000        7.200000    3.800000
max                   4929.000000        9.600000    5.500000

       Albumin_and_Globulin_Ratio      Result
count                  583.000000  583.000000
mean                     0.950858    0.286449
std                      0.321751    0.452490
min                      0.300000    0.000000
25%                      0.700000    0.000000
50%                      0.950000    0.000000
75%                      1.100000    1.000000
max                      2.800000    1.000000
```






```
<Figure size 1000x600 with 0 Axes>
```



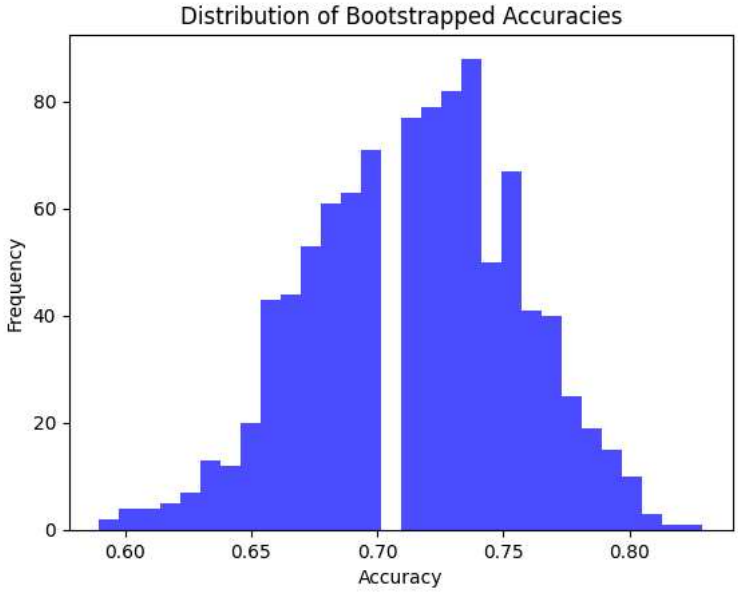



```
Accuracy: 0.74
              precision    recall  f1-score   support

           0       0.74      1.00      0.85        87
           1       0.00      0.00      0.00        30

    accuracy                           0.74       117
   macro avg       0.37      0.50      0.43       117
weighted avg       0.55      0.74      0.63       117

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are il
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are il
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are il
  _warn_prf(average, modifier, msg_start, len(result))
```

1.B)UNCERTAINTY WITH BOOTSTRAPPING:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.utils import resample
import numpy as np
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]
y = data['Result']

# Number of bootstrap samples
n_iterations = 1000
# Store scores
bootstrapped_scores = []

for i in range(n_iterations):
    # Bootstrap sample
    X_sample, y_sample = resample(X, y)
    # Split the bootstrap sample into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_sample, y_sample, test_size=0.2)
    # Standardize features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    # Create and train SVM classifier
    model = SVC(kernel='linear')
    model.fit(X_train_scaled, y_train)
    # Evaluate the model
    y_pred = model.predict(X_test_scaled)
    score = accuracy_score(y_test, y_pred)
    bootstrapped_scores.append(score)

# Calculate the empirical confidence interval and standard deviation
accuracy_mean = np.mean(bootstrapped_scores)
accuracy_std = np.std(bootstrapped_scores)
confidence_lower = np.percentile(bootstrapped_scores, 2.5)
confidence_upper = np.percentile(bootstrapped_scores, 97.5)

print(f"Bootstrap Mean Accuracy: {accuracy_mean:.2f}")
print(f"Bootstrap Accuracy Std Dev: {accuracy_std:.2f}")
print(f"95% Confidence Interval: [{confidence_lower:.2f}, {confidence_upper:.2f}]")

# Plotting the distribution of accuracies
plt.hist(bootstrapped_scores, bins=30, color='blue', alpha=0.7)
plt.title('Distribution of Bootstrapped Accuracies')
plt.xlabel('Accuracy')
plt.ylabel('Frequency')
plt.show()
```

```
Bootstrap Mean Accuracy: 0.71
Bootstrap Accuracy Std Dev: 0.04
95% Confidence Interval: [0.63, 0.79]
```

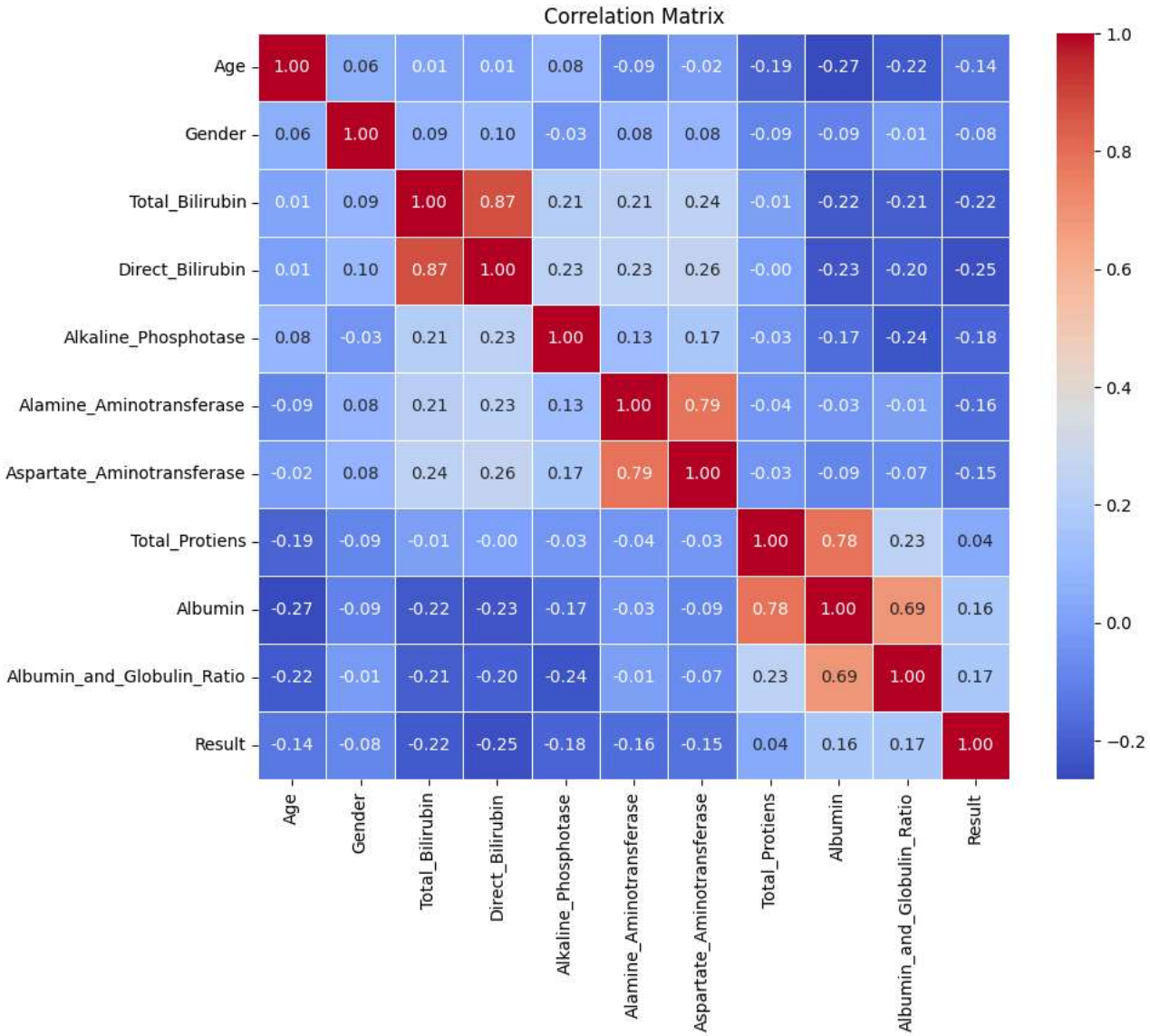



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import numpy as np

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Define a function to perform bootstrap
def bootstrap_accuracy(data, n_iterations=1000):
    accuracies = []
    scaler = StandardScaler()  # Initialize the scaler outside the loop for efficiency
    for _ in range(n_iterations):
        # Resample the data with replacement
        bootstrap_sample = data.sample(n=len(data), replace=True)
        X_bootstrap = bootstrap_sample.drop(columns=['Result'])  # Assuming 'Result' is the target
        y_bootstrap = bootstrap_sample['Result']

        # Split the bootstrap sample
        X_train_bootstrap, X_test_bootstrap, y_train_bootstrap, y_test_bootstrap = train_test_split(
            X_bootstrap, y_bootstrap, test_size=0.2, random_state=42)

        # Standardize features
        X_train_bootstrap_scaled = scaler.fit_transform(X_train_bootstrap)
        X_test_bootstrap_scaled = scaler.transform(X_test_bootstrap)

        # Train an SVM model
        model = SVC(kernel='linear')
        model.fit(X_train_bootstrap_scaled, y_train_bootstrap)

        # Predict on the test set
        y_pred_bootstrap = model.predict(X_test_bootstrap_scaled)

        # Calculate accuracy and store it
        accuracy_bootstrap = accuracy_score(y_test_bootstrap, y_pred_bootstrap)
        accuracies.append(accuracy_bootstrap)

    return accuracies

# Perform bootstrap and calculate uncertainty
bootstrap_accuracies = bootstrap_accuracy(data)
uncertainty = np.std(bootstrap_accuracies)
print("Uncertainty:", uncertainty)
```

```
Uncertainty: 0.04192460943207283
```

1.C)CO-RELATION MATRIX:

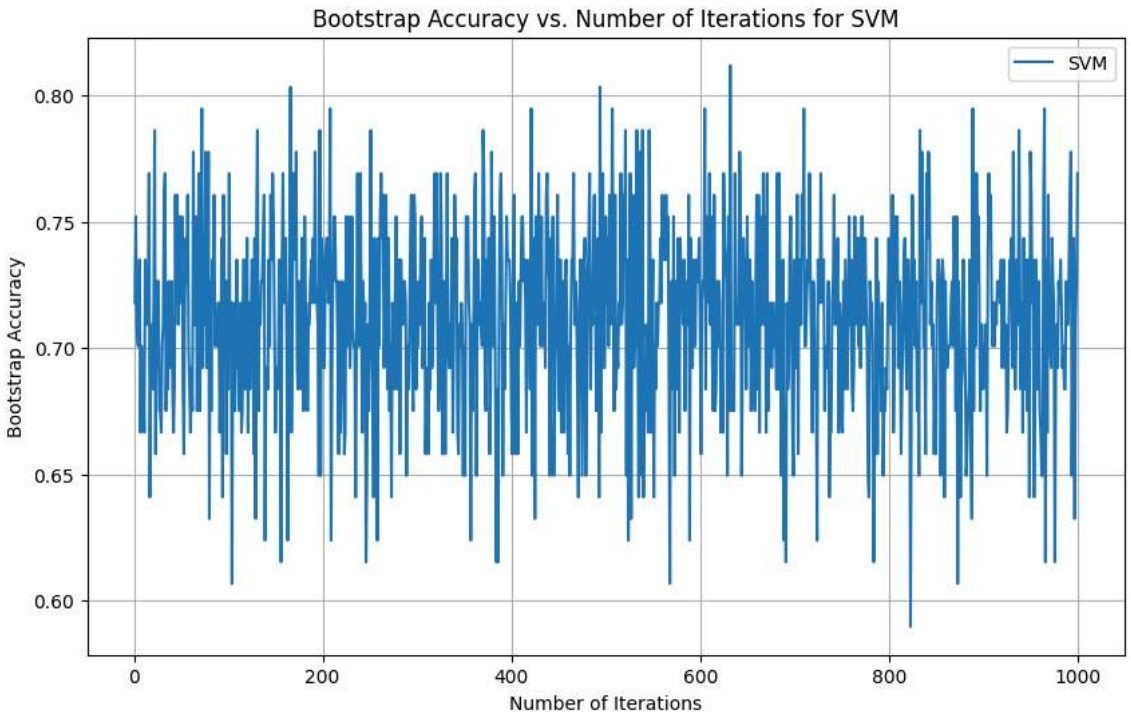
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Calculate the correlation matrix
correlation_matrix = data.corr()

# Visualize the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```



1.D)Graph between Bootstrap Accuracy vs. Number of Iterations:

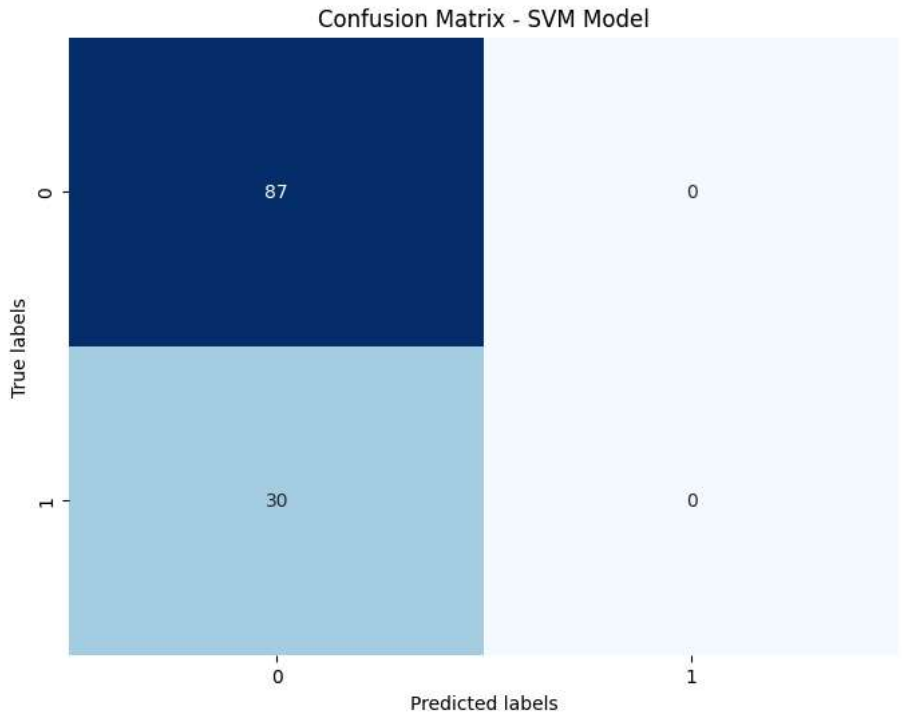
```
import pandas as pd
```

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC  # Support Vector Classifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Number of bootstrap samples
n_iterations = 1000
# Store scores
bootstrapped_scores_SVM = []

# Run bootstrap for SVM
for i in range(n_iterations):
    # Prepare train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    # Standardize features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    # Create a Support Vector Machine classifier
    model_SVM = SVC(kernel='linear')  # You can change the kernel and other parameters as needed
    # Train the model
    model_SVM.fit(X_train_scaled, y_train)
    # Make predictions
    y_pred_SVM = model_SVM.predict(X_test_scaled)
    # Evaluate the model
    score_SVM = accuracy_score(y_test, y_pred_SVM)
    bootstrapped_scores_SVM.append(score_SVM)

# Plotting the graph
plt.figure(figsize=(10, 6))
plt.plot(range(1, n_iterations + 1), bootstrapped_scores_SVM, label='SVM')
plt.title('Bootstrap Accuracy vs. Number of Iterations for SVM')
plt.xlabel('Number of Iterations')
plt.ylabel('Bootstrap Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```



1.E)CONFUSION MATRIX:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]
y = data['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create the SVM classifier
model_SVM = SVC(kernel='linear')  # You can change the kernel if needed

# Train the model
model_SVM.fit(X_train_scaled, y_train)

# Make predictions
y_pred_SVM = model_SVM.predict(X_test_scaled)

# Compute confusion matrix
conf_matrix_SVM = confusion_matrix(y_test, y_pred_SVM)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_SVM, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.title('Confusion Matrix - SVM Model')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

# Print accuracy
accuracy_SVM = accuracy_score(y_test, y_pred_SVM)
print(f"Accuracy: {accuracy_SVM:.2f}")
```




```
Accuracy: 0.74
```

1.F)ERROR RATES:

```
import pandas as pd
import numpy as np
from sklearn.svm import SVC
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split

# Load dataset
data = pd.read_csv('/content/liver.csv')  # Update the path to where your liver disease dataset is located

# Split features and target
X = data.drop('Result', axis=1)  # Assuming the target variable is named 'Result'
y = data['Result']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize SVM classifier
svm = SVC()

# Train the SVM classifier
svm.fit(X_train, y_train)

# Predictions
y_pred = svm.predict(X_test)

# Error calculations
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
# Note: MAPE may not be meaningful if y_test contains zero and is generally not used for classification problems.
# Replacing with Classification Accuracy for a more appropriate metric in classification scenario.
accuracy = np.mean(y_pred == y_test) * 100
mae = mean_absolute_error(y_test, y_pred)

print("SVM:")
print("RMSE:", rmse)
print("Accuracy:", accuracy)  # Using Accuracy instead of MAPE
print("MAE:", mae)
```

```
SVM:
RMSE: 0.5063696835418333
Accuracy: 74.35897435897436
MAE: 0.2564102564102564
```

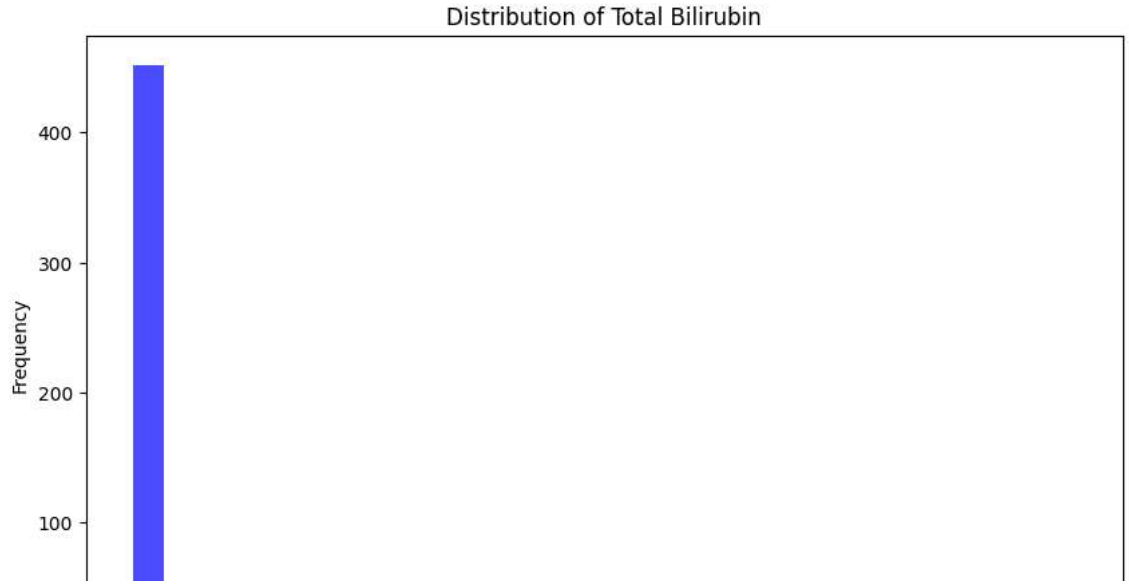
2.LOGISTIC REGRESSION:

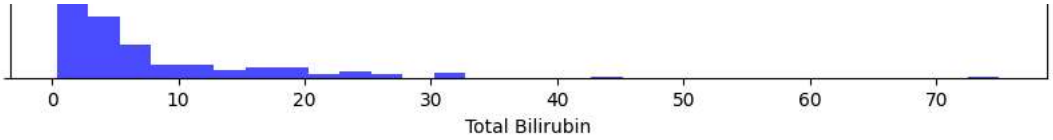
2.A)ACCURACY:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataframe
print(data.head())

# Basic statistics of the dataset
print(data.describe())

# Visualize the distribution of Total Bilirubin
plt.figure(figsize=(10, 6))
plt.hist(data['Total_Bilirubin'], bins=30, color='blue', alpha=0.7)
plt.title('Distribution of Total Bilirubin')
plt.xlabel('Total Bilirubin')
plt.ylabel('Frequency')
plt.show()

# Box plot for comparing Total Bilirubin across Result categories (0 and 1)
plt.figure(figsize=(10, 6))
data.boxplot(column='Total_Bilirubin', by='Result')
plt.title('Total Bilirubin Levels by Result')
plt.suptitle('')
plt.xlabel('Result')
plt.ylabel('Total Bilirubin')
plt.show()

# Scatter plot of Total Bilirubin vs. Direct Bilirubin
plt.figure(figsize=(10, 6))
plt.scatter(data['Total_Bilirubin'], data['Direct_Bilirubin'], alpha=0.5)
plt.title('Total Bilirubin vs Direct Bilirubin')
plt.xlabel('Total Bilirubin')
plt.ylabel('Direct Bilirubin')
plt.grid(True)
plt.show()

# Prepare data for logistic regression
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a Logistic Regression model
model = LogisticRegression()

# Train the model
model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = model.predict(X_test_scaled)

# Print accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

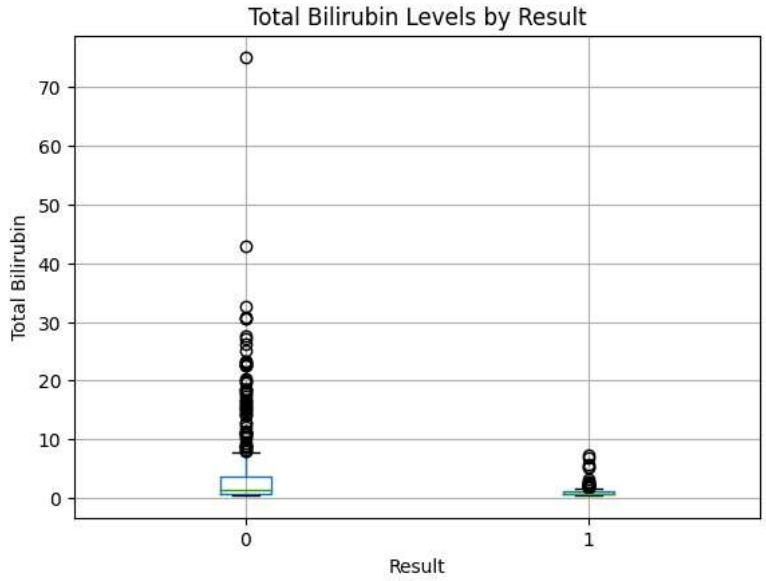
```
# Print classification report
print(classification_report(y_test, y_pred))
```

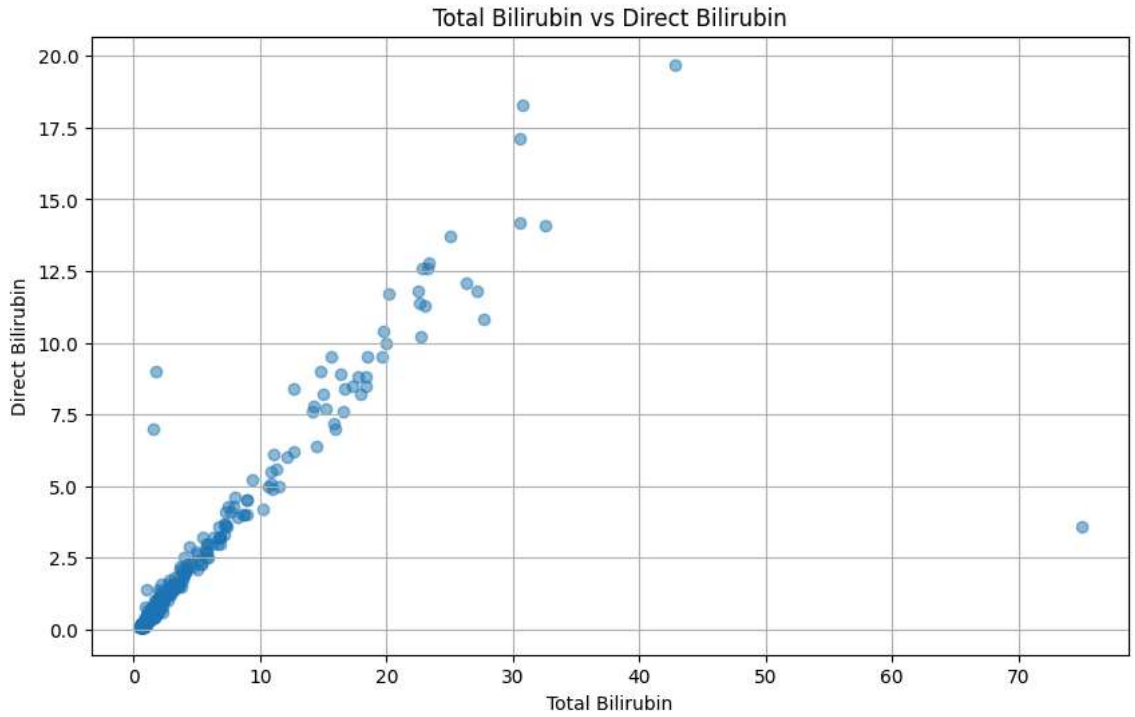
```
   Age  Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
0   65       0              0.7               0.1                   187
1   62       1             10.9               5.5                   699
2   62       1              7.3               4.1                   490
3   58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

   Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens  \
0                        16                          18             6.8
1                        64                         100             7.5
2                        60                          68             7.0
3                        14                          20             6.8
4                        27                          59             7.3

   Albumin  Albumin_and_Globulin_Ratio  Result
0      3.3                        0.90       0
1      3.2                        0.74       0
2      3.3                        0.89       0
3      3.4                        1.00       0
4      2.4                        0.40       0
              Age      Gender  Total_Bilirubin  Direct_Bilirubin  \
count  583.000000  583.000000       583.000000        583.000000
mean    44.746141    0.756432         3.298799          1.486106
std     16.189833    0.429603         6.209522          2.808498
min      4.000000    0.000000         0.400000          0.100000
25%     33.000000    1.000000         0.800000          0.200000
50%     45.000000    1.000000         1.000000          0.300000
75%     58.000000    1.000000         2.600000          1.300000
max     90.000000    1.000000        75.000000         19.700000

       Alkaline_Phosphotase  Alamine_Aminotransferase  \
count            583.000000                583.000000
mean             290.576329                 80.713551
std              242.937989                182.620356
min               63.000000                 10.000000
25%              175.500000                 23.000000
50%              208.000000                 35.000000
75%              298.000000                 60.500000
max             2110.000000               2000.000000

       Aspartate_Aminotransferase  Total_Protiens     Albumin  \
count                  583.000000      583.000000  583.000000
mean                   109.910806        6.483190    3.141852
std                    288.918529        1.085451    0.795519
min                     10.000000        2.700000    0.900000
25%                     25.000000        5.800000    2.600000
50%                     42.000000        6.600000    3.100000
75%                     87.000000        7.200000    3.800000
max                   4929.000000        9.600000    5.500000

       Albumin_and_Globulin_Ratio      Result
count                  583.000000  583.000000
mean                     0.950858    0.286449
std                      0.321751    0.452490
min                      0.300000    0.000000
25%                      0.700000    0.000000
50%                      0.950000    0.000000
75%                      1.100000    1.000000
max                      2.800000    1.000000
```






```
<Figure size 1000x600 with 0 Axes>
```



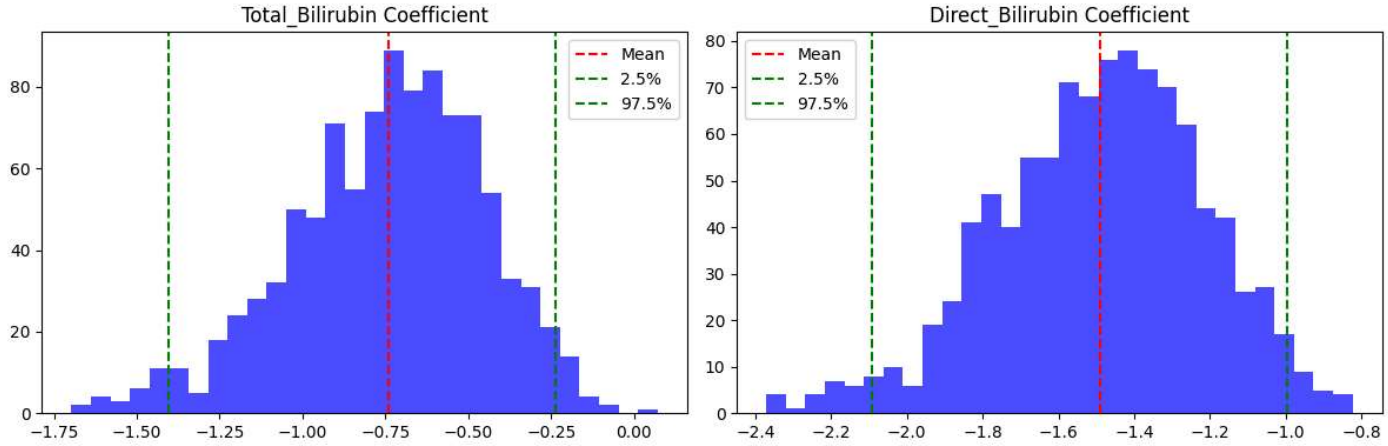



```
Accuracy: 0.74
              precision    recall  f1-score   support

           0       0.74      1.00      0.85        87
           1       0.00      0.00      0.00        30

    accuracy                           0.74       117
   macro avg       0.37      0.50      0.43       117
weighted avg       0.55      0.74      0.63       117

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are il
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are il
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are il
  _warn_prf(average, modifier, msg_start, len(result))
```

2.B)UNCERTAINTY WITH BOOTSTRAPPING:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.utils import resample
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data for logistic regression
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Number of bootstrap samples
n_iterations = 1000
coef_data = np.zeros((n_iterations, X_train.shape[1]))

# Perform bootstrapping
for i in range(n_iterations):
    # Resample the training data with replacement
    X_train_resampled, y_train_resampled = resample(X_train_scaled, y_train, random_state=i)

    # Create and fit the logistic regression model to the bootstrapped dataset
    model = LogisticRegression()
    model.fit(X_train_resampled, y_train_resampled)

    # Store the coefficients
    coef_data[i, :] = model.coef_[0]

# Calculating the mean and confidence intervals for the coefficients
coef_mean = np.mean(coef_data, axis=0)
coef_std_error = np.std(coef_data, axis=0)
lower_bounds = np.percentile(coef_data, 2.5, axis=0)
upper_bounds = np.percentile(coef_data, 97.5, axis=0)

# Display results
for idx, feature_name in enumerate(X.columns):
    print(f"{feature_name} Coefficient Mean: {coef_mean[idx]:.4f}, Std error: {coef_std_error[idx]:.4f}")
    print(f"95% Confidence Interval: ({lower_bounds[idx]:.4f}, {upper_bounds[idx]:.4f})")

# Optionally, plot the distributions of the coefficients
fig, axes = plt.subplots(nrows=1, ncols=len(X.columns), figsize=(12, 4))
for idx, ax in enumerate(axes):
    ax.hist(coef_data[:, idx], bins=30, alpha=0.7, color='blue')
    ax.axvline(x=coef_mean[idx], color='red', linestyle='--', label='Mean')
    ax.axvline(x=lower_bounds[idx], color='green', linestyle='--', label='2.5%')
    ax.axvline(x=upper_bounds[idx], color='green', linestyle='--', label='97.5%')
    ax.set_title(f'{X.columns[idx]} Coefficient')
    ax.legend()

plt.tight_layout()
plt.show()
```

```
Total_Bilirubin Coefficient Mean: -0.7407, Std error: 0.2909
95% Confidence Interval: (-1.4037, -0.2368)
Direct_Bilirubin Coefficient Mean: -1.4898, Std error: 0.2724
95% Confidence Interval: (-2.0916, -0.9968)
```

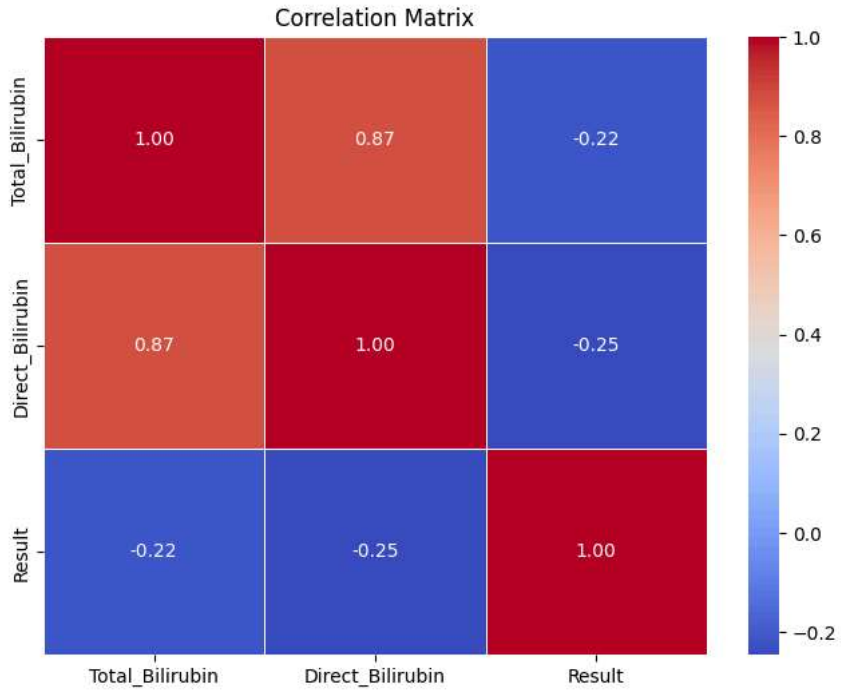



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import numpy as np

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Define a function to perform bootstrap
def bootstrap_accuracy(data, n_iterations=1000):
    accuracies = []
    scaler = StandardScaler()  # Initialize the scaler outside the loop for efficiency
    for _ in range(n_iterations):
        # Resample the data with replacement
        bootstrap_sample = data.sample(n=len(data), replace=True)
        X_bootstrap = bootstrap_sample.drop(columns=['Result'])  # Assuming 'Result' is the target
        y_bootstrap = bootstrap_sample['Result']

        # Split the bootstrap sample
        X_train_bootstrap, X_test_bootstrap, y_train_bootstrap, y_test_bootstrap = train_test_split(
            X_bootstrap, y_bootstrap, test_size=0.2, random_state=42)

        # Standardize features
        X_train_bootstrap_scaled = scaler.fit_transform(X_train_bootstrap)
        X_test_bootstrap_scaled = scaler.transform(X_test_bootstrap)

        # Train a Logistic Regression model
        model = LogisticRegression()
        model.fit(X_train_bootstrap_scaled, y_train_bootstrap)

        # Predict on the test set
        y_pred_bootstrap = model.predict(X_test_bootstrap_scaled)

        # Calculate accuracy and store it
        accuracy_bootstrap = accuracy_score(y_test_bootstrap, y_pred_bootstrap)
        accuracies.append(accuracy_bootstrap)

    return accuracies

# Perform bootstrap and calculate uncertainty
bootstrap_accuracies = bootstrap_accuracy(data)
uncertainty = np.std(bootstrap_accuracies)
print("Uncertainty:", uncertainty)
```

```
Uncertainty: 0.0432132821779703
```

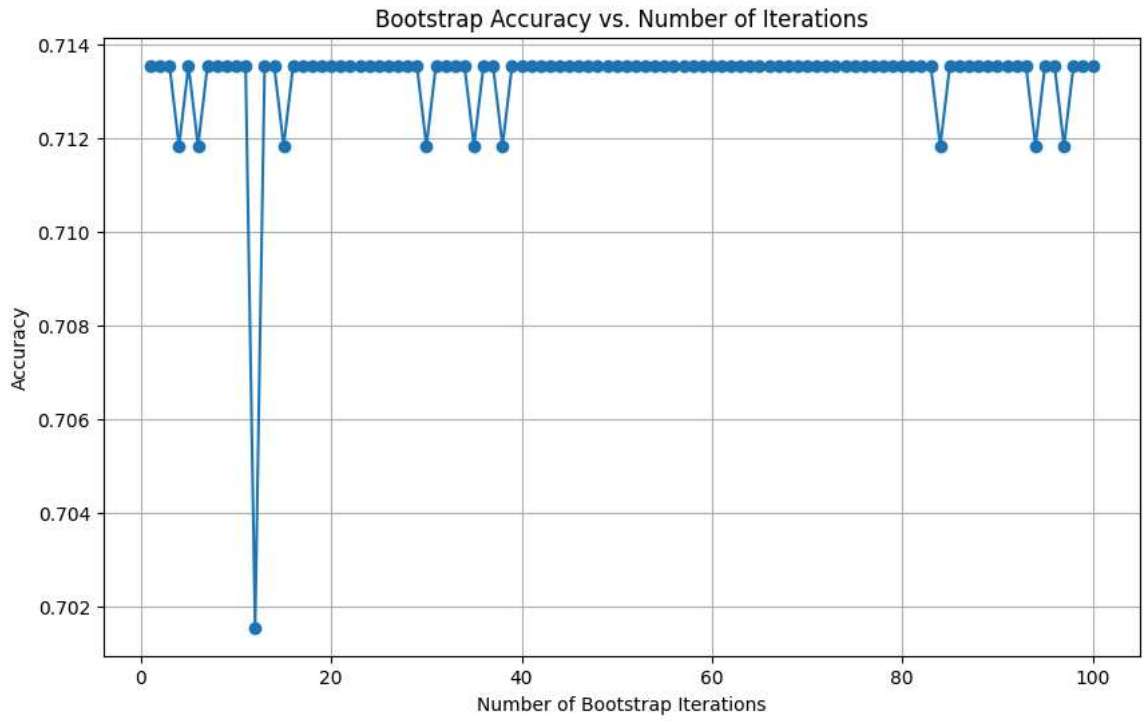
2.C)CO-RELATION MATRIX:

```
import seaborn as sns

# Concatenate the features and target variable to create a single dataframe
df = pd.concat([X, y], axis=1)

# Calculate the correlation matrix
corr_matrix = df.corr()

# Plot the correlation matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



2.D)Graph between Bootstrap Accuracy vs. Number of Iterations:

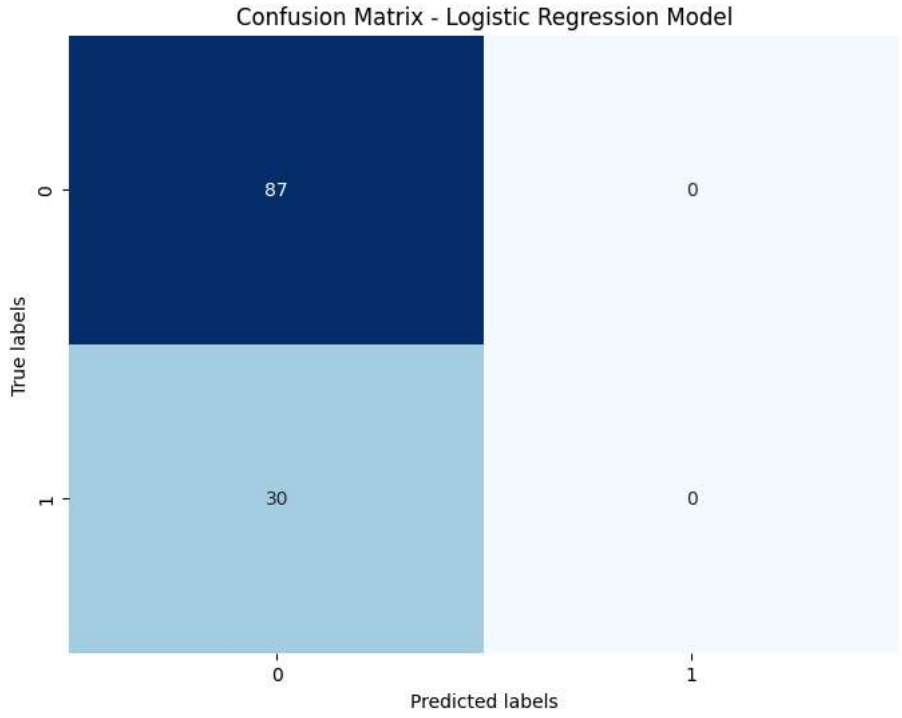
```
import numpy as np
from sklearn.utils import resample
from sklearn.metrics import accuracy_score

# Number of bootstrap samples to create
n_iterations = 100
# Store the accuracy for each bootstrap
accuracies = []

# Loop through the specified number of bootstrap samples
for i in range(n_iterations):
    # Resample the data with replacement
    X_boot, y_boot = resample(X, y)
    # Create a new logistic regression model
    model = LogisticRegression(solver='liblinear')
    # Fit the model with the bootstrap sample
    model.fit(X_boot, y_boot)
    # Predict on the original data (out-of-bag)
    y_pred = model.predict(X)
    # Calculate accuracy
    accuracy = accuracy_score(y, y_pred)
    # Append the accuracy to the list
    accuracies.append(accuracy)

# Plotting the results
plt.figure(figsize=(10, 6))
plt.plot(range(1, n_iterations + 1), accuracies, marker='o', linestyle='-')
plt.title('Bootstrap Accuracy vs. Number of Iterations')
plt.xlabel('Number of Bootstrap Iterations')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```



2.E)CONFUSION MATRIX:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
```

```
file_path = ('/content/liver.csv')
data = pd.read_csv(file_path)

# Prepare data
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a Logistic Regression classifier
model_LR = LogisticRegression()

# Train the model
model_LR.fit(X_train_scaled, y_train)

# Make predictions
y_pred_LR = model_LR.predict(X_test_scaled)

# Compute confusion matrix
conf_matrix_LR = confusion_matrix(y_test, y_pred_LR)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_LR, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.title('Confusion Matrix - Logistic Regression Model')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

# Print accuracy
accuracy_LR = accuracy_score(y_test, y_pred_LR)
print(f"Accuracy: {accuracy_LR:.2f}")
```




```
Accuracy: 0.74
```

2.F)ERROR RATES:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split

# Load dataset
data = pd.read_csv('/content/liver.csv')  # Update the path to where your liver disease dataset is located

# Split features and target
X = data.drop('Result', axis=1)  # Assuming the target variable is named 'Result'
y = data['Result']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize Logistic Regression classifier
log_reg = LogisticRegression()

# Train the Logistic Regression classifier
log_reg.fit(X_train, y_train)

# Predictions
y_pred = log_reg.predict(X_test)

# Error calculations
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
# Note: MAPE may not be meaningful if y_test contains zero and is generally not used for classification problems.
# Replacing with Classification Accuracy for a more appropriate metric in classification scenario.
accuracy = np.mean(y_pred == y_test) * 100
mae = mean_absolute_error(y_test, y_pred)

print("Logistic Regression:")
print("RMSE:", rmse)
print("Accuracy:", accuracy)  # Using Accuracy instead of MAPE
print("MAE:", mae)
```

```
Logistic Regression:
RMSE: 0.5229763603684908
Accuracy: 72.64957264957265
MAE: 0.27350427350427353
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: 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(
```

3.K-NEAREST NEIGHBORS(KNN) MODEL:

3.A)ACCURACY:

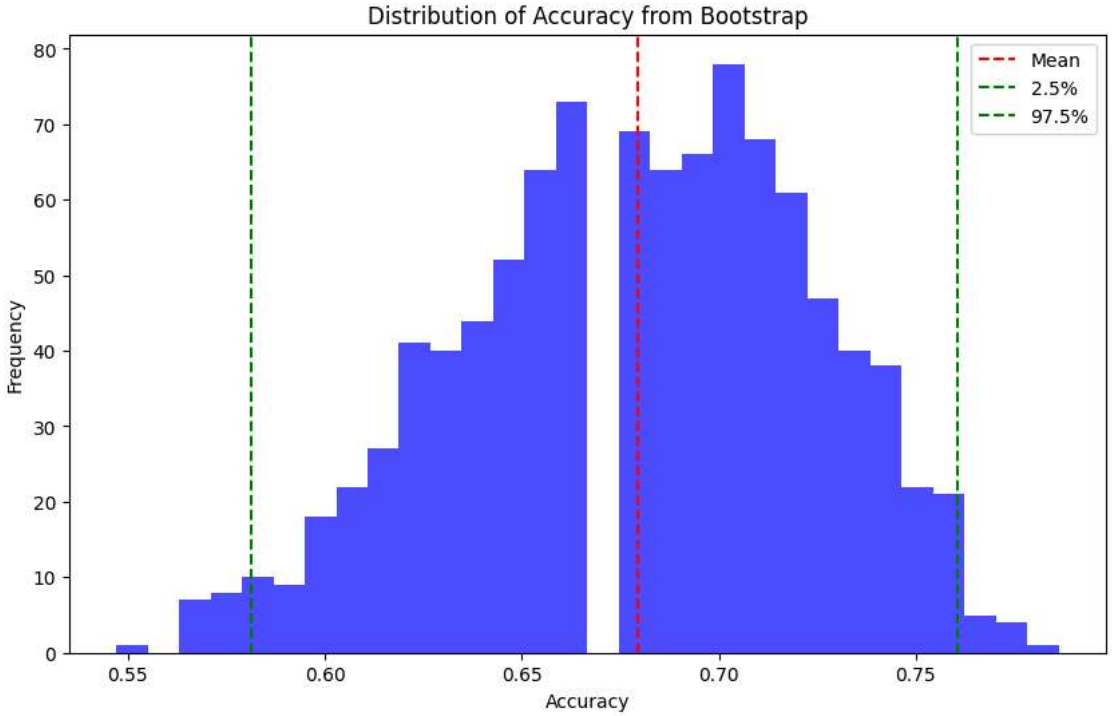
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataframe
print(data.head())

# Basic statistics of the dataset
print(data.describe())

# Visualize the distribution of Total Bilirubin
plt.figure(figsize=(10, 6))
plt.hist(data['Total_Bilirubin'], bins=30, color='blue', alpha=0.7)
plt.title('Distribution of Total Bilirubin')
plt.xlabel('Total Bilirubin')
plt.ylabel('Frequency')
plt.show()

# Box plot for comparing Total Bilirubin across Result categories (0 and 1)
plt.figure(figsize=(10, 6))
data.boxplot(column='Total_Bilirubin', by='Result')
plt.title('Total Bilirubin Levels by Result')
plt.suptitle('')
plt.xlabel('Result')
plt.ylabel('Total Bilirubin')
plt.show()

# Scatter plot of Total Bilirubin vs. Direct Bilirubin
plt.figure(figsize=(10, 6))
plt.scatter(data['Total_Bilirubin'], data['Direct_Bilirubin'], alpha=0.5)
plt.title('Total Bilirubin vs Direct Bilirubin')
plt.xlabel('Total Bilirubin')
plt.ylabel('Direct Bilirubin')
plt.grid(True)
plt.show()

# Prepare data for K-nearest neighbor
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a K-nearest neighbor classifier with k=5 (you can adjust k as needed)
k = 5
model = KNeighborsClassifier(n_neighbors=k)

# Train the model
model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = model.predict(X_test_scaled)

# Print accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
print(classification_report(y_test, y_pred))
```

```
   Age  Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
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

   Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens  \
0                        16                          18             6.8
1                        64                         100             7.5
2                        60                          68             7.0
3                        14                          20             6.8
4                        27                          59             7.3

   Albumin  Albumin_and_Globulin_Ratio  Result
0      3.3                        0.90       0
1      3.2                        0.74       0
2      3.3                        0.89       0
3      3.4                        1.00       0
4      2.4                        0.40       0
               Age      Gender  Total_Bilirubin  Direct_Bilirubin  \
count  583.000000  583.000000       583.000000        583.000000
mean    44.746141    0.756432         3.298799          1.486106
std     16.189833    0.429603         6.209522          2.808498
min      4.000000    0.000000         0.400000          0.100000
25%     33.000000    1.000000         0.800000          0.200000
50%     45.000000    1.000000         1.000000          0.300000
75%     58.000000    1.000000         2.600000          1.300000
max     90.000000    1.000000        75.000000         19.700000

       Alkaline_Phosphotase  Alamine_Aminotransferase  \
count            583.000000                583.000000
mean             290.576329                 80.713551
std              242.937989                182.620356
min               63.000000                 10.000000
25%              175.500000                 23.000000
50%              208.000000                 35.000000
75%              298.000000                 60.500000
max             2110.000000               2000.000000

       Aspartate_Aminotransferase  Total_Protiens     Albumin  \
count                  583.000000      583.000000  583.000000
mean                   109.910806        6.483190    3.141852
std                    288.918529        1.085451    0.795519
min                     10.000000        2.700000    0.900000
25%                     25.000000        5.800000    2.600000
50%                     42.000000        6.600000    3.100000
75%                     87.000000        7.200000    3.800000
max                   4929.000000        9.600000    5.500000

       Albumin_and_Globulin_Ratio      Result
count                  583.000000  583.000000
mean                     0.950858    0.286449
std                      0.321751    0.452490
min                      0.300000    0.000000
25%                      0.700000    0.000000
50%                      0.950000    0.000000
```

3.B)UNCERTAINTY WITH BOOTSTRAPING:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.utils import resample
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data for KNN
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Number of bootstrap samples
n_iterations = 1000
accuracy_data = []

# Perform bootstrapping
for i in range(n_iterations):
    # Resample the training data with replacement
    X_train_resampled, y_train_resampled = resample(X_train_scaled, y_train, random_state=i)

    # Create and fit the KNN model to the bootstrapped dataset
    model = KNeighborsClassifier(n_neighbors=5)
    model.fit(X_train_resampled, y_train_resampled)

    # Evaluate the model on the original test set
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_data.append(accuracy)

# Calculating the mean accuracy and confidence intervals
accuracy_mean = np.mean(accuracy_data)
accuracy_std_error = np.std(accuracy_data)
lower_bound = np.percentile(accuracy_data, 2.5)
upper_bound = np.percentile(accuracy_data, 97.5)

# Display results
print(f"Mean Accuracy: {accuracy_mean:.4f}, Std error: {accuracy_std_error:.4f}")
print(f"95% Confidence Interval: ({lower_bound:.4f}, {upper_bound:.4f})")

# Plot the distribution of accuracies
plt.figure(figsize=(10, 6))
plt.hist(accuracy_data, bins=30, alpha=0.7, color='blue')
plt.axvline(x=accuracy_mean, color='red', linestyle='--', label='Mean')
plt.axvline(x=lower_bound, color='green', linestyle='--', label='2.5%')
plt.axvline(x=upper_bound, color='green', linestyle='--', label='97.5%')
plt.title('Distribution of Accuracy from Bootstrap')
plt.xlabel('Accuracy')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

```
Mean Accuracy: 0.6794, Std error: 0.0449
95% Confidence Interval: (0.5812, 0.7607)
```

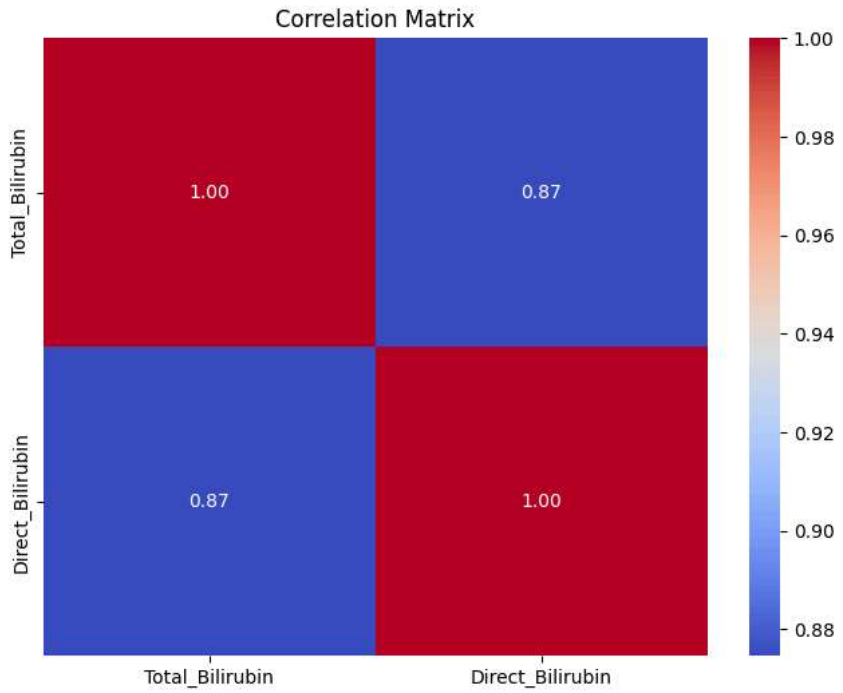


```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import numpy as np

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Define a function to perform bootstrap
def bootstrap_accuracy(data, n_iterations=1000):
    accuracies = []
    scaler = StandardScaler()  # Initialize the scaler outside the loop for efficiency
    for _ in range(n_iterations):
        # Resample the data with replacement
        bootstrap_sample = data.sample(n=len(data), replace=True)
        X_bootstrap = bootstrap_sample.drop(columns=['Result'])  # Assuming 'Result' is the target
        y_bootstrap = bootstrap_sample['Result']

        # Split the bootstrap sample
        X_train_bootstrap, X_test_bootstrap, y_train_bootstrap, y_test_bootstrap = train_test_split(
            X_bootstrap, y_bootstrap, test_size=0.2, random_state=42)

        # Standardize features
        X_train_bootstrap_scaled = scaler.fit_transform(X_train_bootstrap)
        X_test_bootstrap_scaled = scaler.transform(X_test_bootstrap)

        # Train a KNN model
        model = KNeighborsClassifier()
        model.fit(X_train_bootstrap_scaled, y_train_bootstrap)

        # Predict on the test set
        y_pred_bootstrap = model.predict(X_test_bootstrap_scaled)

        # Calculate accuracy and store it
        accuracy_bootstrap = accuracy_score(y_test_bootstrap, y_pred_bootstrap)
        accuracies.append(accuracy_bootstrap)

    return accuracies

# Perform bootstrap and calculate uncertainty
bootstrap_accuracies = bootstrap_accuracy(data)
uncertainty = np.std(bootstrap_accuracies)
print("Uncertainty:", uncertainty)
```

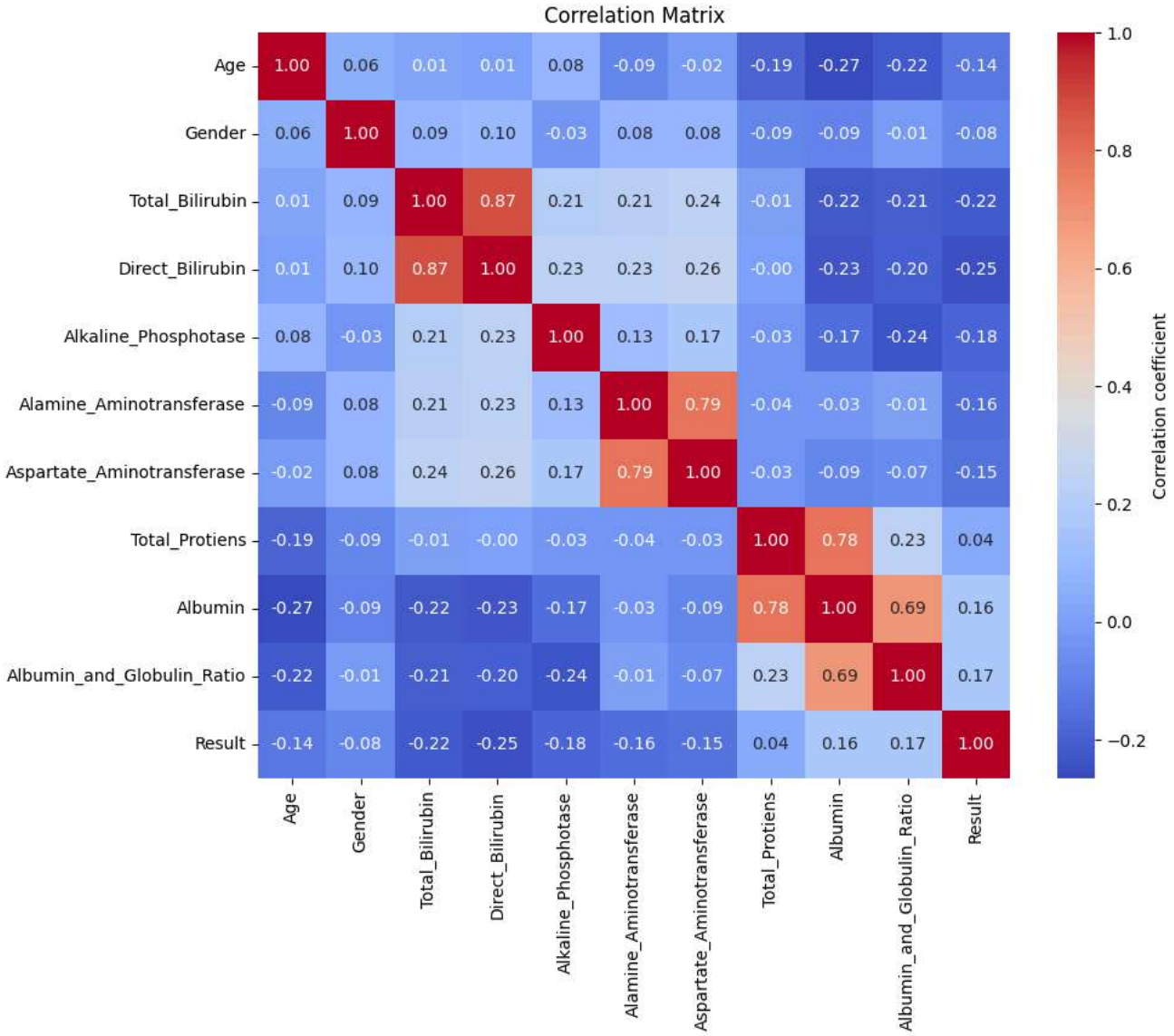
```
Uncertainty: 0.04223446654529389
```

3.C)Co-Relation Matrix:

```
import seaborn as sns

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data for KNN
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here

# Plot correlation matrix using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(X.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

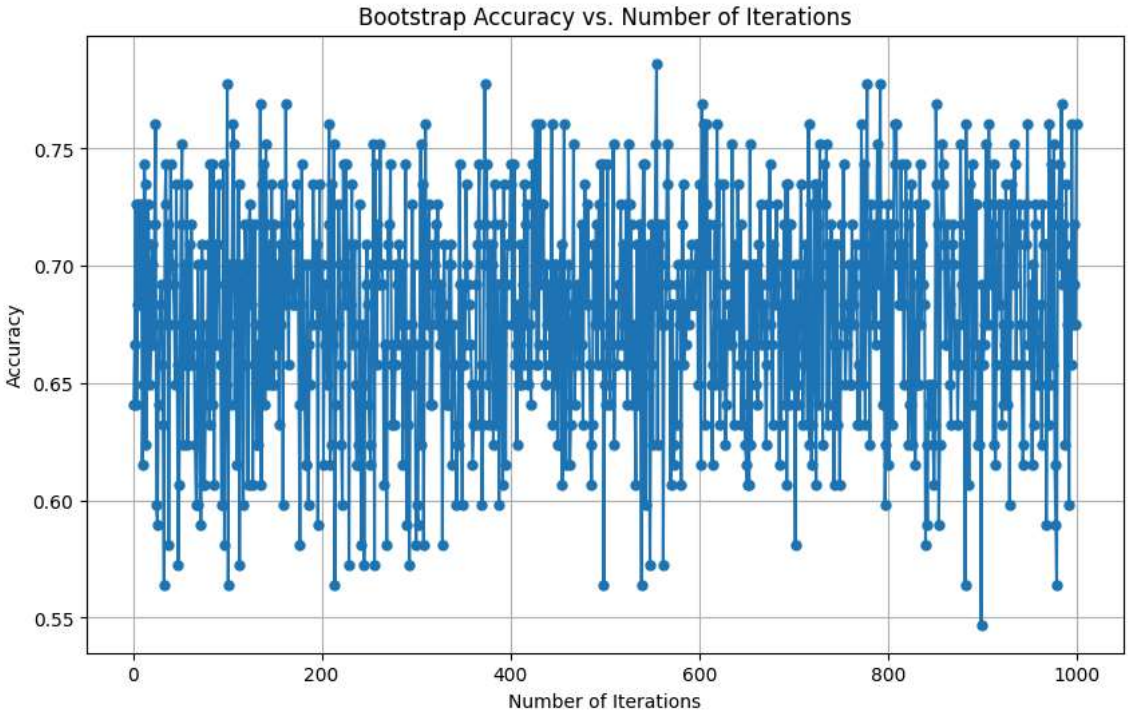



```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Calculate the correlation matrix
correlation_matrix = data.corr()

# Plot the correlation matrix using seaborn for better visualization
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar_kws={'label': 'Correlation coefficient'})
plt.title('Correlation Matrix')
plt.xticks(rotation=90)
plt.yticks(rotation=0)  # Keeps the y-axis labels horizontal
plt.show()
```



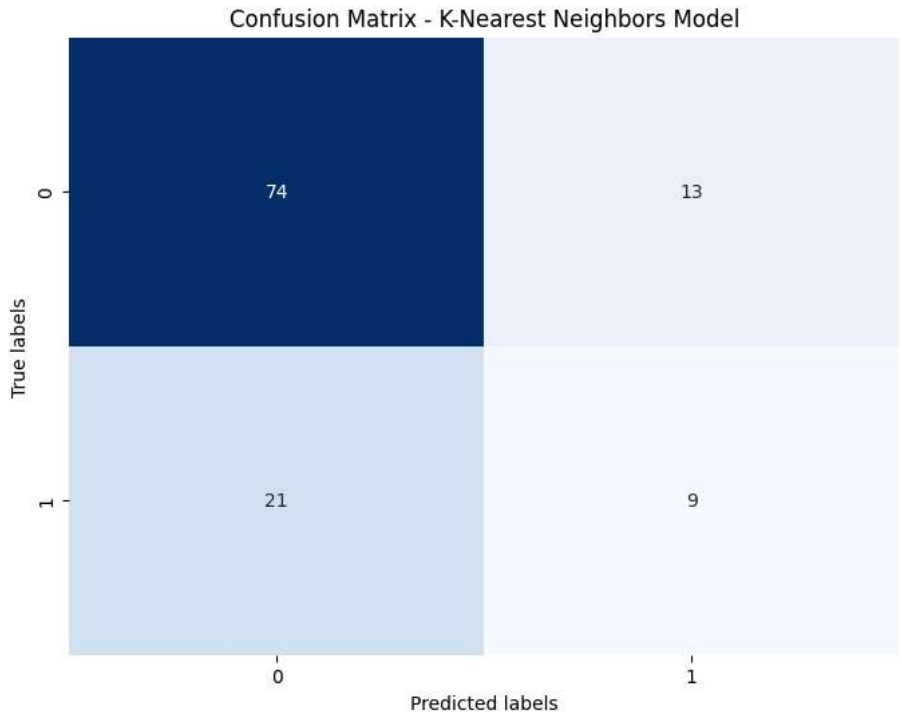
3.D)Graph between Bootstrap Accuracy vs. Number of Iterations:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.utils import resample
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data for KNN
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Number of bootstrap samples and KNN settings
n_iterations = 1000
accuracy_scores = []

# Perform bootstrapping
for i in range(n_iterations):
    # Resample the training data with replacement
    X_train_resampled, y_train_resampled = resample(X_train_scaled, y_train, random_state=i)

    # Create and fit the KNN model to the bootstrapped dataset
    model = KNeighborsClassifier()
    model.fit(X_train_resampled, y_train_resampled)

    # Evaluate the model
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores.append(accuracy)

# Plotting Bootstrap Accuracy vs. Number of Iterations
plt.figure(figsize=(10, 6))
plt.plot(range(1, n_iterations + 1), accuracy_scores, marker='o', linestyle='-', markersize=5)
plt.title('Bootstrap Accuracy vs. Number of Iterations')
plt.xlabel('Number of Iterations')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```



3.E)CONFUSION MATRIX:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a K-Nearest Neighbors classifier
model_KNN = KNeighborsClassifier()

# Train the model
model_KNN.fit(X_train_scaled, y_train)

# Make predictions
y_pred_KNN = model_KNN.predict(X_test_scaled)

# Compute confusion matrix
conf_matrix_KNN = confusion_matrix(y_test, y_pred_KNN)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_KNN, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.title('Confusion Matrix - K-Nearest Neighbors Model')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

# Print accuracy
accuracy_KNN = accuracy_score(y_test, y_pred_KNN)
print(f"Accuracy: {accuracy_KNN:.2f}")
```




```
Accuracy: 0.71
```

3.F)ERROR RATES:

```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split

# Load dataset
data = pd.read_csv('/content/liver.csv')  # Update the path to where your liver disease dataset is located

# Split features and target
X = data.drop('Result', axis=1)  # Assuming the target variable is named 'Result'
y = data['Result']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)  # You can adjust the number of neighbors

# Train the KNN classifier
knn.fit(X_train, y_train)

# Predictions
y_pred = knn.predict(X_test)

# Error calculations
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
# Note: MAPE may not be meaningful if y_test contains zero and is generally not used for classification problems.
# Replacing with Classification Accuracy for a more appropriate metric in classification scenario.
accuracy = np.mean(y_pred == y_test) * 100
mae = mean_absolute_error(y_test, y_pred)

print("KNN:")
print("RMSE:", rmse)
print("Accuracy:", accuracy)  # Using Accuracy instead of MAPE
print("MAE:", mae)
```

```
KNN:
RMSE: 0.5623515948579823
Accuracy: 68.37606837606837
MAE: 0.3162393162393162
```

4.NAIVE BAYES MODEL:

4.A)ACCURACY:

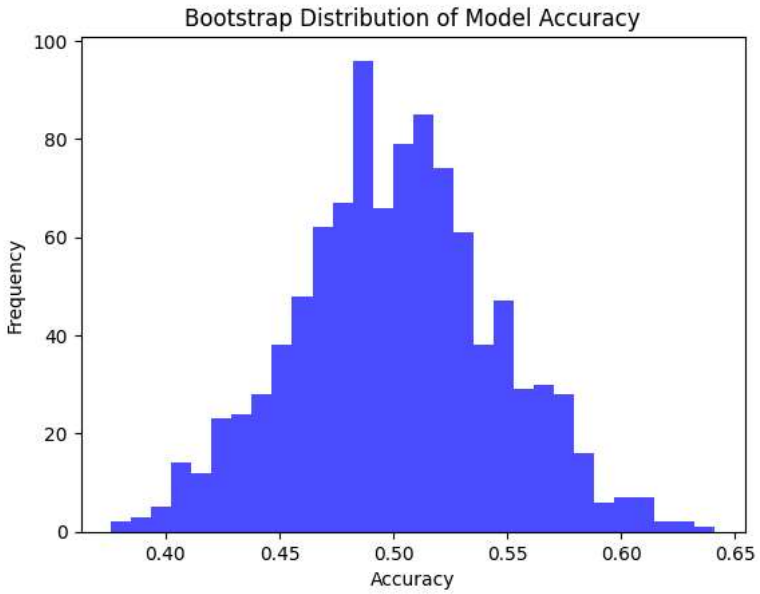
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataframe
print(data.head())

# Basic statistics of the dataset
print(data.describe())

# Visualize the distribution of Total Bilirubin
plt.figure(figsize=(10, 6))
plt.hist(data['Total_Bilirubin'], bins=30, color='blue', alpha=0.7)
plt.title('Distribution of Total Bilirubin')
plt.xlabel('Total Bilirubin')
plt.ylabel('Frequency')
plt.show()

# Box plot for comparing Total Bilirubin across Result categories (0 and 1)
plt.figure(figsize=(10, 6))
data.boxplot(column='Total_Bilirubin', by='Result')
plt.title('Total Bilirubin Levels by Result')
plt.suptitle('')
plt.xlabel('Result')
plt.ylabel('Total Bilirubin')
plt.show()

# Scatter plot of Total Bilirubin vs. Direct Bilirubin
plt.figure(figsize=(10, 6))
plt.scatter(data['Total_Bilirubin'], data['Direct_Bilirubin'], alpha=0.5)
plt.title('Total Bilirubin vs Direct Bilirubin')
plt.xlabel('Total Bilirubin')
plt.ylabel('Direct Bilirubin')
plt.grid(True)
plt.show()

# Prepare data for Naive Bayes
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a Gaussian Naive Bayes classifier
model = GaussianNB()

# Train the model
model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = model.predict(X_test_scaled)

# Print accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
print(classification_report(y_test, y_pred))
```

```
   Age  Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
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

   Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens  \
0                        16                          18             6.8
1                        64                         100             7.5
2                        60                          68             7.0
3                        14                          20             6.8
4                        27                          59             7.3

   Albumin  Albumin_and_Globulin_Ratio  Result
0      3.3                        0.90       0
1      3.2                        0.74       0
2      3.3                        0.89       0
3      3.4                        1.00       0
4      2.4                        0.40       0
               Age      Gender  Total_Bilirubin  Direct_Bilirubin  \
count  583.000000  583.000000       583.000000        583.000000
mean    44.746141    0.756432         3.298799          1.486106
std     16.189833    0.429603         6.209522          2.808498
min      4.000000    0.000000         0.400000          0.100000
25%     33.000000    1.000000         0.800000          0.200000
50%     45.000000    1.000000         1.000000          0.300000
75%     58.000000    1.000000         2.600000          1.300000
max     90.000000    1.000000        75.000000         19.700000

       Alkaline_Phosphotase  Alamine_Aminotransferase  \
count            583.000000                583.000000
mean             290.576329                 80.713551
std              242.937989                182.620356
min               63.000000                 10.000000
25%              175.500000                 23.000000
50%              208.000000                 35.000000
75%              298.000000                 60.500000
max             2110.000000               2000.000000

       Aspartate_Aminotransferase  Total_Protiens     Albumin  \
count                  583.000000      583.000000  583.000000
mean                   109.910806        6.483190    3.141852
std                    288.918529        1.085451    0.795519
min                     10.000000        2.700000    0.900000
25%                     25.000000        5.800000    2.600000
50%                     42.000000        6.600000    3.100000
75%                     87.000000        7.200000    3.800000
max                   4929.000000        9.600000    5.500000

       Albumin_and_Globulin_Ratio      Result
count                  583.000000  583.000000
mean                     0.950858    0.286449
std                      0.321751    0.452490
min                      0.300000    0.000000
25%                      0.700000    0.000000
50%                      0.950000    0.000000
```

4.B) *UNCERTAINTY* WITH BOOTSTRAPPING:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.utils import resample
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Number of bootstrap samples
n_iterations = 1000
# Store scores
bootstrapped_scores = []

# Run bootstrap
for i in range(n_iterations):
    # Prepare train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    # Standardize features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    # Create a Gaussian Naive Bayes classifier
    model = GaussianNB()
    # Train the model
    model.fit(X_train_scaled, y_train)
    # Make predictions
    y_pred = model.predict(X_test_scaled)
    # Evaluate the model
    score = accuracy_score(y_test, y_pred)
    bootstrapped_scores.append(score)

# Calculating the 95% confidence interval
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower = max(0.0, np.percentile(bootstrapped_scores, p))
p = (alpha+((1.0-alpha)/2.0)) * 100
upper = min(1.0, np.percentile(bootstrapped_scores, p))
print(f"{alpha*100:.1f}% confidence interval {lower:.2f} and {upper:.2f}")

# Plotting the bootstrap distributions
plt.hist(bootstrapped_scores, bins=30, color='blue', alpha=0.7)
plt.title('Bootstrap Distribution of Model Accuracy')
plt.xlabel('Accuracy')
plt.ylabel('Frequency')
plt.show()
```

```
95.0% confidence interval 0.42 and 0.58
```




```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import numpy as np

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Define a function to perform bootstrap
def bootstrap_accuracy(data, n_iterations=1000):
    accuracies = []
    scaler = StandardScaler()  # Initialize the scaler outside the loop for efficiency
    for _ in range(n_iterations):
        # Resample the data with replacement
        bootstrap_sample = data.sample(n=len(data), replace=True)
        X_bootstrap = bootstrap_sample.drop(columns=['Result'])  # Assuming 'Result' is the target
        y_bootstrap = bootstrap_sample['Result']

        # Split the bootstrap sample
        X_train_bootstrap, X_test_bootstrap, y_train_bootstrap, y_test_bootstrap = train_test_split(
            X_bootstrap, y_bootstrap, test_size=0.2, random_state=42)

        # Standardize features
        X_train_bootstrap_scaled = scaler.fit_transform(X_train_bootstrap)
        X_test_bootstrap_scaled = scaler.transform(X_test_bootstrap)

        # Train a Naive Bayes model
        model = GaussianNB()
        model.fit(X_train_bootstrap_scaled, y_train_bootstrap)

        # Predict on the test set
        y_pred_bootstrap = model.predict(X_test_bootstrap_scaled)

        # Calculate accuracy and store it
        accuracy_bootstrap = accuracy_score(y_test_bootstrap, y_pred_bootstrap)
        accuracies.append(accuracy_bootstrap)

    return accuracies

# Perform bootstrap and calculate uncertainty
bootstrap_accuracies = bootstrap_accuracy(data)
uncertainty = np.std(bootstrap_accuracies)
print("Uncertainty:", uncertainty)
```

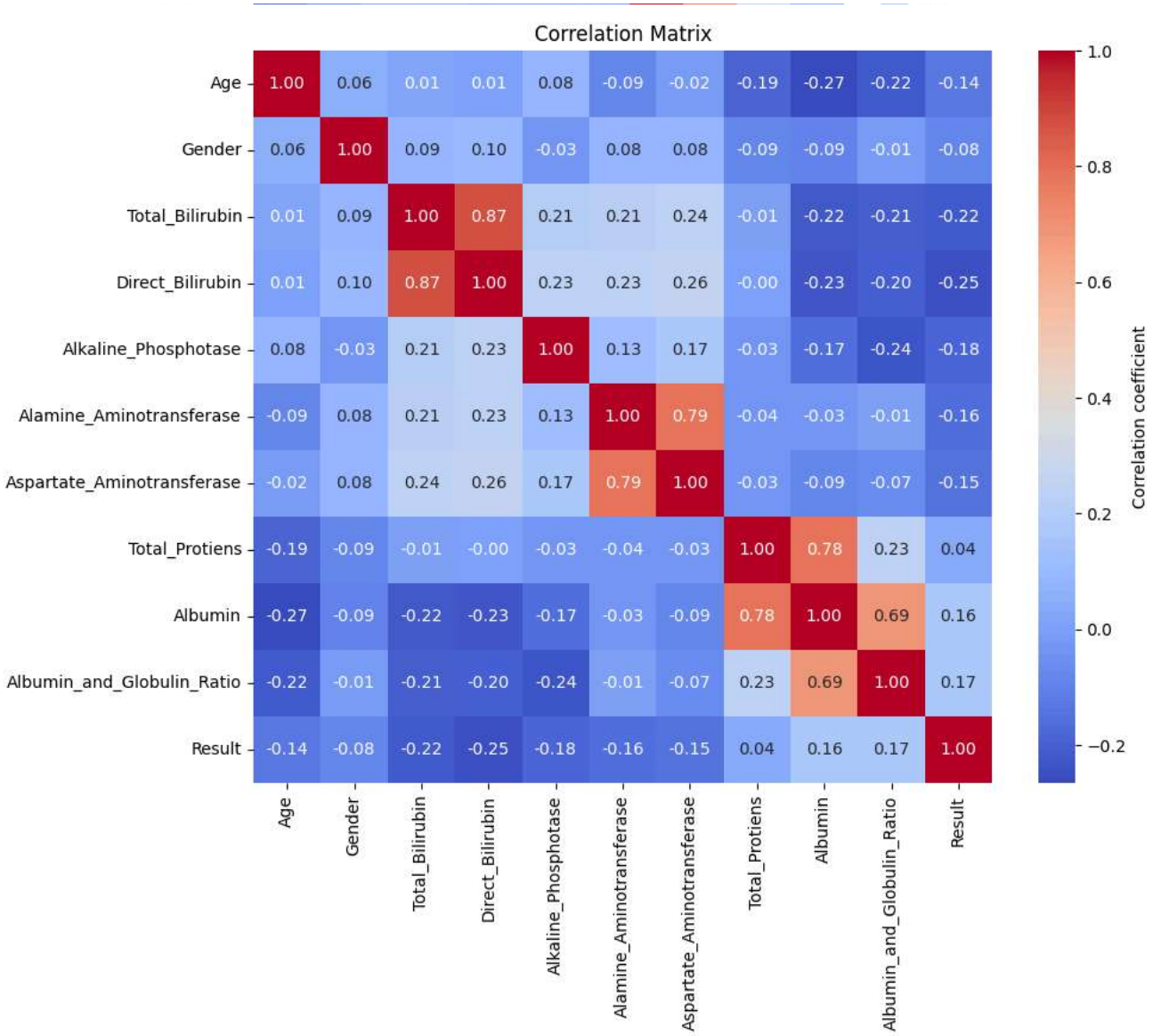
```
Uncertainty: 0.04912820512820513
```

4.C)CO-RELATION MATRIX:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Compute the correlation matrix
correlation_matrix = data.corr()

# Print the correlation matrix
print("Correlation matrix:")
print(correlation_matrix)

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title('Correlation Matrix')
plt.show()
```

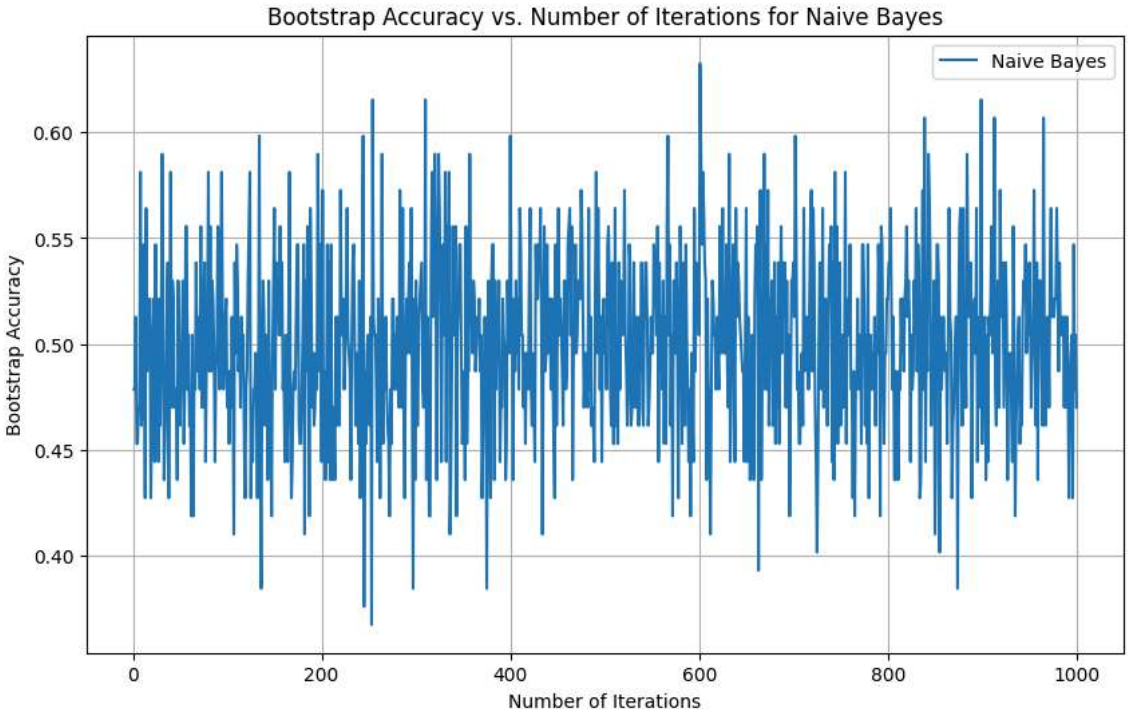
```
Correlation matrix:
                                 Age    Gender  Total_Bilirubin  \
Age                         1.000000  0.056560         0.011763
Gender                      0.056560  1.000000         0.089291
Total_Bilirubin             0.011763  0.089291         1.000000
Direct_Bilirubin            0.007529  0.100436         0.874618
Alkaline_Phosphotase        0.080425 -0.027496         0.206669
Alamine_Aminotransferase   -0.086883  0.082332         0.214065
Aspartate_Aminotransferase -0.019910  0.080336         0.237831
Total_Protiens             -0.187461 -0.089121        -0.008099
Albumin                    -0.265924 -0.093799        -0.222250
Albumin_and_Globulin_Ratio -0.217727 -0.010419        -0.208634
Result                     -0.137351 -0.082416        -0.220208

                            Direct_Bilirubin  Alkaline_Phosphotase  \
Age                                 0.007529              0.080425
Gender                              0.100436             -0.027496
Total_Bilirubin                     0.874618              0.206669
Direct_Bilirubin                    1.000000              0.234939
Alkaline_Phosphotase                0.234939              1.000000
Alamine_Aminotransferase            0.233894              0.125680
Aspartate_Aminotransferase          0.257544              0.167196
Total_Protiens                     -0.000139             -0.028514
Albumin                            -0.228531             -0.165453
Albumin_and_Globulin_Ratio         -0.202861             -0.237148
Result                             -0.246046             -0.184866

                            Alamine_Aminotransferase  \
Age                                        -0.086883
Gender                                      0.082332
Total_Bilirubin                             0.214065
Direct_Bilirubin                            0.233894
Alkaline_Phosphotase                        0.125680
Alamine_Aminotransferase                    1.000000
Aspartate_Aminotransferase                  0.791966
Total_Protiens                             -0.042518
Albumin                                    -0.029742
Albumin_and_Globulin_Ratio                 -0.006212
Result                                     -0.163416

                            Aspartate_Aminotransferase  Total_Protiens  \
Age                                          -0.019910       -0.187461
Gender                                        0.080336       -0.089121
Total_Bilirubin                               0.237831       -0.008099
Direct_Bilirubin                              0.257544       -0.000139
Alkaline_Phosphotase                          0.167196       -0.028514
Alamine_Aminotransferase                      0.791966       -0.042518
Aspartate_Aminotransferase                    1.000000       -0.025645
Total_Protiens                               -0.025645        1.000000
Albumin                                      -0.085290        0.784053
Albumin_and_Globulin_Ratio                   -0.072295        0.233893
Result                                       -0.151934        0.035008

                             Albumin  Albumin_and_Globulin_Ratio    Result
Age                        -0.265924                  -0.217727 -0.137351
Gender                     -0.093799                  -0.010419 -0.082416
Total_Bilirubin            -0.222250                  -0.208634 -0.220208
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Calculate the correlation matrix
correlation_matrix = data.corr()

# Plot the correlation matrix using seaborn for better visualization
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar_kws={'label': 'Correlation coefficient'})
plt.title('Correlation Matrix')
plt.xticks(rotation=90)
plt.yticks(rotation=0)  # Keeps the y-axis labels horizontal
plt.show()
```



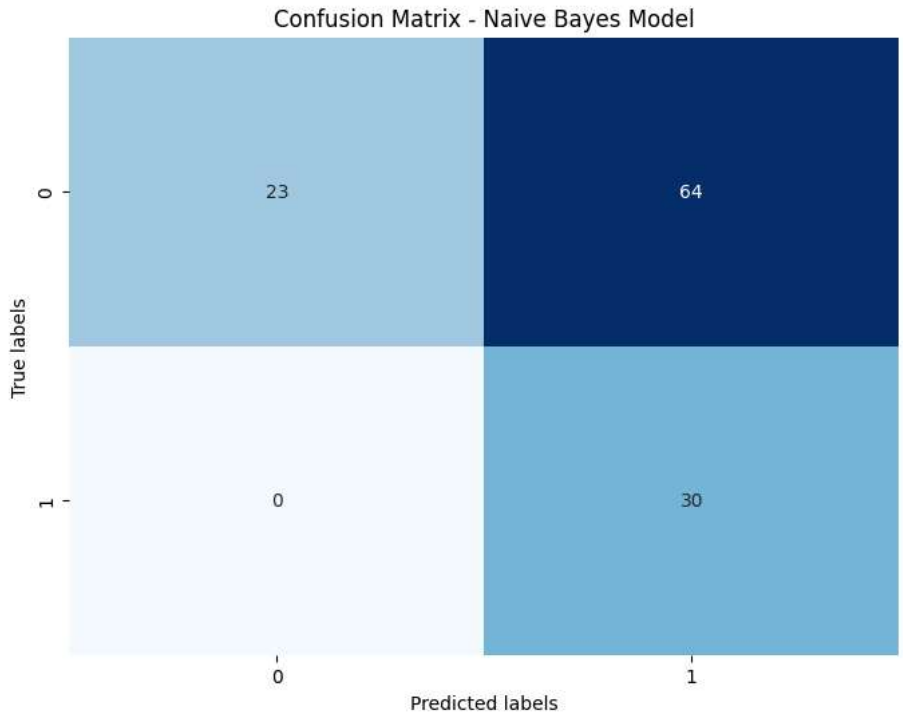
4.D)Graph between Bootstrap Accuracy vs. Number of Iterations:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Number of bootstrap samples
n_iterations = 1000
# Store scores
bootstrapped_scores_NB = []

# Run bootstrap for Naive Bayes
for i in range(n_iterations):
    # Prepare train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    # Standardize features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    # Create a Gaussian Naive Bayes classifier
    model_NB = GaussianNB()
    # Train the model
    model_NB.fit(X_train_scaled, y_train)
    # Make predictions
    y_pred_NB = model_NB.predict(X_test_scaled)
    # Evaluate the model
    score_NB = accuracy_score(y_test, y_pred_NB)
    bootstrapped_scores_NB.append(score_NB)

# Plotting the graph
plt.figure(figsize=(10, 6))
plt.plot(range(1, n_iterations + 1), bootstrapped_scores_NB, label='Naive Bayes')
plt.title('Bootstrap Accuracy vs. Number of Iterations for Naive Bayes')
plt.xlabel('Number of Iterations')
plt.ylabel('Bootstrap Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```



4.E) *CONFUSION* MATRIX:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a Gaussian Naive Bayes classifier
model_NB = GaussianNB()

# Train the model
model_NB.fit(X_train_scaled, y_train)

# Make predictions
y_pred_NB = model_NB.predict(X_test_scaled)

# Compute confusion matrix
conf_matrix_NB = confusion_matrix(y_test, y_pred_NB)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_NB, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.title('Confusion Matrix - Naive Bayes Model')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

# Print accuracy
accuracy_NB = accuracy_score(y_test, y_pred_NB)
print(f"Accuracy: {accuracy_NB:.2f}")
```




```
Accuracy: 0.45
```

4.F)ERROR RATES:

```
import pandas as pd
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split

# Load dataset
data = pd.read_csv('/content/liver.csv')  # Update the path to where your liver disease dataset is located

# Split features and target
X = data.drop('Result', axis=1)  # Assuming the target variable is named 'Result'
y = data['Result']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize Naive Bayes classifier
nb = GaussianNB()

# Train the Naive Bayes classifier
nb.fit(X_train, y_train)

# Predictions
y_pred = nb.predict(X_test)

# Calculate errors and accuracy
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
accuracy = np.mean(y_pred == y_test) * 100  # Calculating accuracy
mae = mean_absolute_error(y_test, y_pred)

print("Naive Bayes:")
print("RMSE:", rmse)
print("Accuracy:", accuracy)  # Displaying accuracy instead of MAPE
print("MAE:", mae)
```

```
Naive Bayes:
RMSE: 0.6918326955503611
Accuracy: 52.13675213675214
MAE: 0.47863247863247865
```

5.DECISION TREES MODEL:

5.A)ACCURACY:

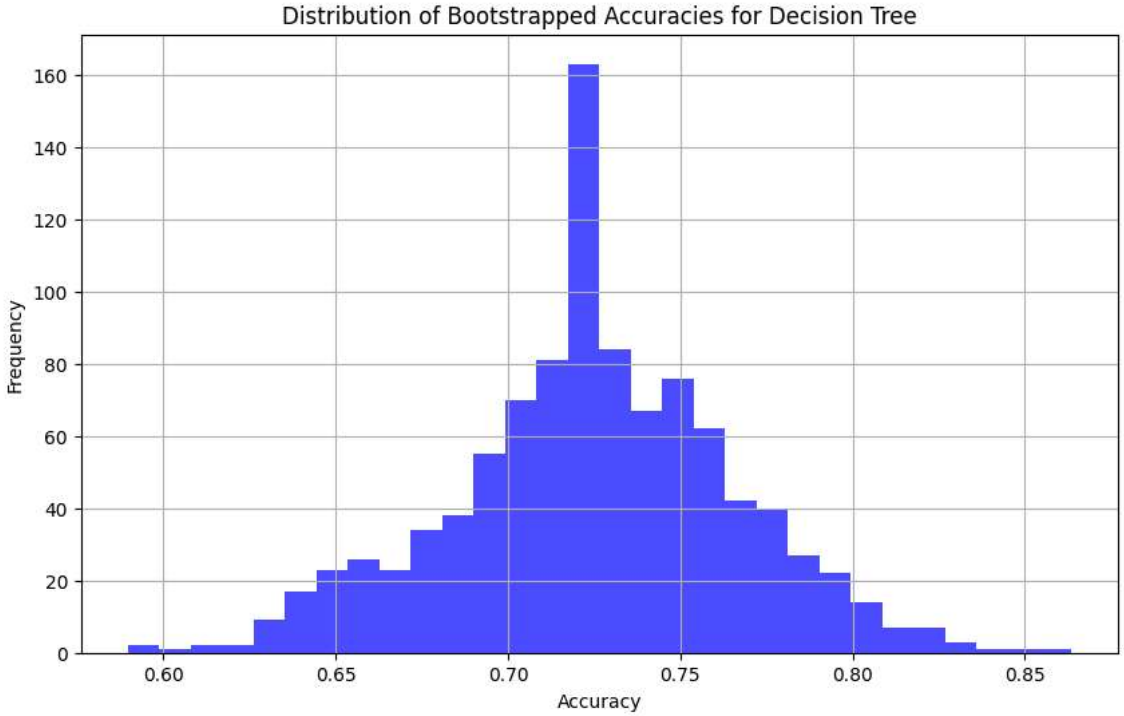
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataframe
print(data.head())

# Basic statistics of the dataset
print(data.describe())

# Visualize the distribution of Total Bilirubin
plt.figure(figsize=(10, 6))
plt.hist(data['Total_Bilirubin'], bins=30, color='blue', alpha=0.7)
plt.title('Distribution of Total Bilirubin')
plt.xlabel('Total Bilirubin')
plt.ylabel('Frequency')
plt.show()

# Box plot for comparing Total Bilirubin across Result categories (0 and 1)
plt.figure(figsize=(10, 6))
data.boxplot(column='Total_Bilirubin', by='Result')
plt.title('Total Bilirubin Levels by Result')
plt.suptitle('')
plt.xlabel('Result')
plt.ylabel('Total Bilirubin')
plt.show()

# Scatter plot of Total Bilirubin vs. Direct Bilirubin
plt.figure(figsize=(10, 6))
plt.scatter(data['Total_Bilirubin'], data['Direct_Bilirubin'], alpha=0.5)
plt.title('Total Bilirubin vs Direct Bilirubin')
plt.xlabel('Total Bilirubin')
plt.ylabel('Direct Bilirubin')
plt.grid(True)
plt.show()

# Prepare data for Decision Tree
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a Decision Tree classifier
model = DecisionTreeClassifier()

# Train the model
model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = model.predict(X_test_scaled)

# Print accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
print(classification_report(y_test, y_pred))
```

```
   Age  Gender  Total_Bilirubin  Direct_Bilirubin  Alkaline_Phosphotase  \
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

   Alamine_Aminotransferase  Aspartate_Aminotransferase  Total_Protiens  \
0                        16                          18             6.8
1                        64                         100             7.5
2                        60                          68             7.0
3                        14                          20             6.8
4                        27                          59             7.3

   Albumin  Albumin_and_Globulin_Ratio  Result
0      3.3                        0.90       0
1      3.2                        0.74       0
2      3.3                        0.89       0
3      3.4                        1.00       0
4      2.4                        0.40       0
               Age      Gender  Total_Bilirubin  Direct_Bilirubin  \
count  583.000000  583.000000       583.000000        583.000000
mean    44.746141    0.756432         3.298799          1.486106
std     16.189833    0.429603         6.209522          2.808498
min      4.000000    0.000000         0.400000          0.100000
25%     33.000000    1.000000         0.800000          0.200000
50%     45.000000    1.000000         1.000000          0.300000
75%     58.000000    1.000000         2.600000          1.300000
max     90.000000    1.000000        75.000000         19.700000

       Alkaline_Phosphotase  Alamine_Aminotransferase  \
count            583.000000                583.000000
mean             290.576329                 80.713551
std              242.937989                182.620356
min               63.000000                 10.000000
25%              175.500000                 23.000000
50%              208.000000                 35.000000
75%              298.000000                 60.500000
max             2110.000000               2000.000000

       Aspartate_Aminotransferase  Total_Protiens     Albumin  \
count                  583.000000      583.000000  583.000000
mean                   109.910806        6.483190    3.141852
std                    288.918529        1.085451    0.795519
min                     10.000000        2.700000    0.900000
25%                     25.000000        5.800000    2.600000
50%                     42.000000        6.600000    3.100000
75%                     87.000000        7.200000    3.800000
max                   4929.000000        9.600000    5.500000

       Albumin_and_Globulin_Ratio      Result
count                  583.000000  583.000000
mean                     0.950858    0.286449
std                      0.321751    0.452490
min                      0.300000    0.000000
25%                      0.700000    0.000000
50%                      0.950000    0.000000
```

5.B)UNCERTAINTY WITH BOOTSTRAPPING:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Number of bootstrap samples
n_iterations = 1000
# Store scores
bootstrapped_scores = []

# Run bootstrap
for i in range(n_iterations):
    # Resample data with replacement
    X_resampled, y_resampled = resample(X, y)
    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=i)

    # Standardize features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Create a Decision Tree classifier
    model = DecisionTreeClassifier()

    # Train the model
    model.fit(X_train_scaled, y_train)

    # Make predictions
    y_pred = model.predict(X_test_scaled)

    # Evaluate the model
    score = accuracy_score(y_test, y_pred)
    bootstrapped_scores.append(score)

# Calculate the uncertainty of the model
bootstrapped_scores = np.array(bootstrapped_scores)
print(f"Mean accuracy: {bootstrapped_scores.mean():.2f}")
print(f"Standard deviation of accuracy: {bootstrapped_scores.std():.2f}")

# Plotting the distribution of bootstrapped accuracies
plt.figure(figsize=(10, 6))
plt.hist(bootstrapped_scores, bins=30, alpha=0.7, color='blue')
plt.title('Distribution of Bootstrapped Accuracies for Decision Tree')
plt.xlabel('Accuracy')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

```
Mean accuracy: 0.73
Standard deviation of accuracy: 0.04
```

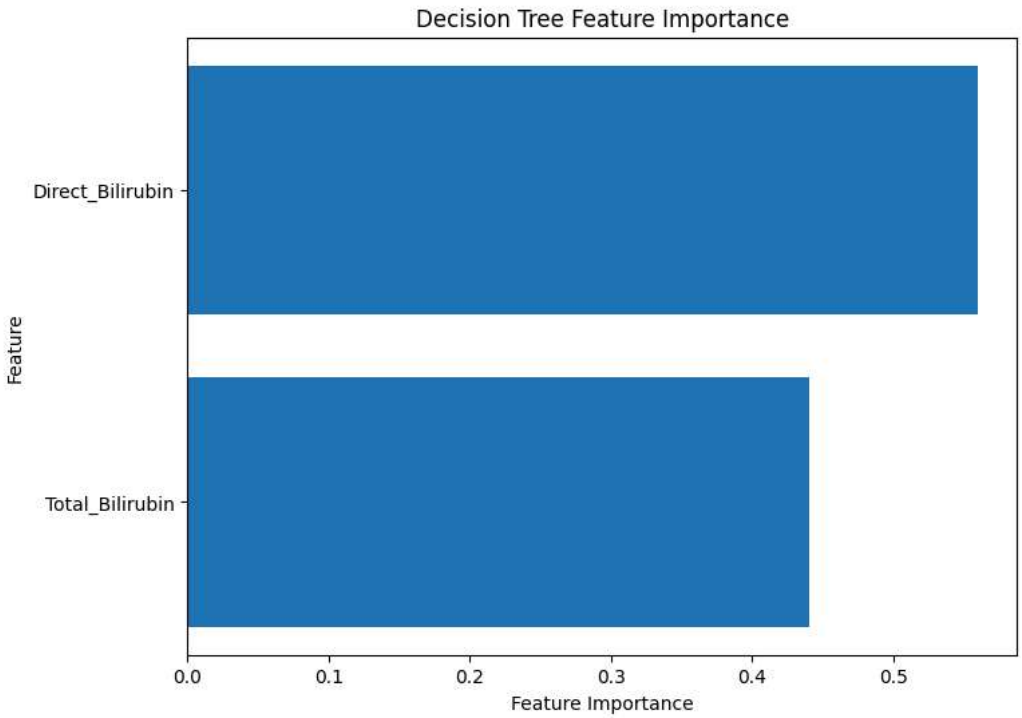


```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import numpy as np

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Define a function to perform bootstrap
def bootstrap_accuracy(data, n_iterations=1000):
    accuracies = []
    scaler = StandardScaler()  # Initialize the scaler outside the loop for efficiency
    for _ in range(n_iterations):
        # Resample the data with replacement
        bootstrap_sample = data.sample(n=len(data), replace=True)
        X_bootstrap = bootstrap_sample.drop(columns=['Result'])  # Assuming 'Result' is the target
        y_bootstrap = bootstrap_sample['Result']

        # Split the bootstrap sample
        X_train_bootstrap, X_test_bootstrap, y_train_bootstrap, y_test_bootstrap = train_test_split(
            X_bootstrap, y_bootstrap, test_size=0.2, random_state=42)

        # Standardize features
        X_train_bootstrap_scaled = scaler.fit_transform(X_train_bootstrap)
        X_test_bootstrap_scaled = scaler.transform(X_test_bootstrap)

        # Train a Decision Tree model
        model = DecisionTreeClassifier()
        model.fit(X_train_bootstrap_scaled, y_train_bootstrap)

        # Predict on the test set
        y_pred_bootstrap = model.predict(X_test_bootstrap_scaled)

        # Calculate accuracy and store it
        accuracy_bootstrap = accuracy_score(y_test_bootstrap, y_pred_bootstrap)
        accuracies.append(accuracy_bootstrap)

    return accuracies

# Perform bootstrap and calculate uncertainty
bootstrap_accuracies = bootstrap_accuracy(data)
uncertainty = np.std(bootstrap_accuracies)
print("Uncertainty:", uncertainty)
```

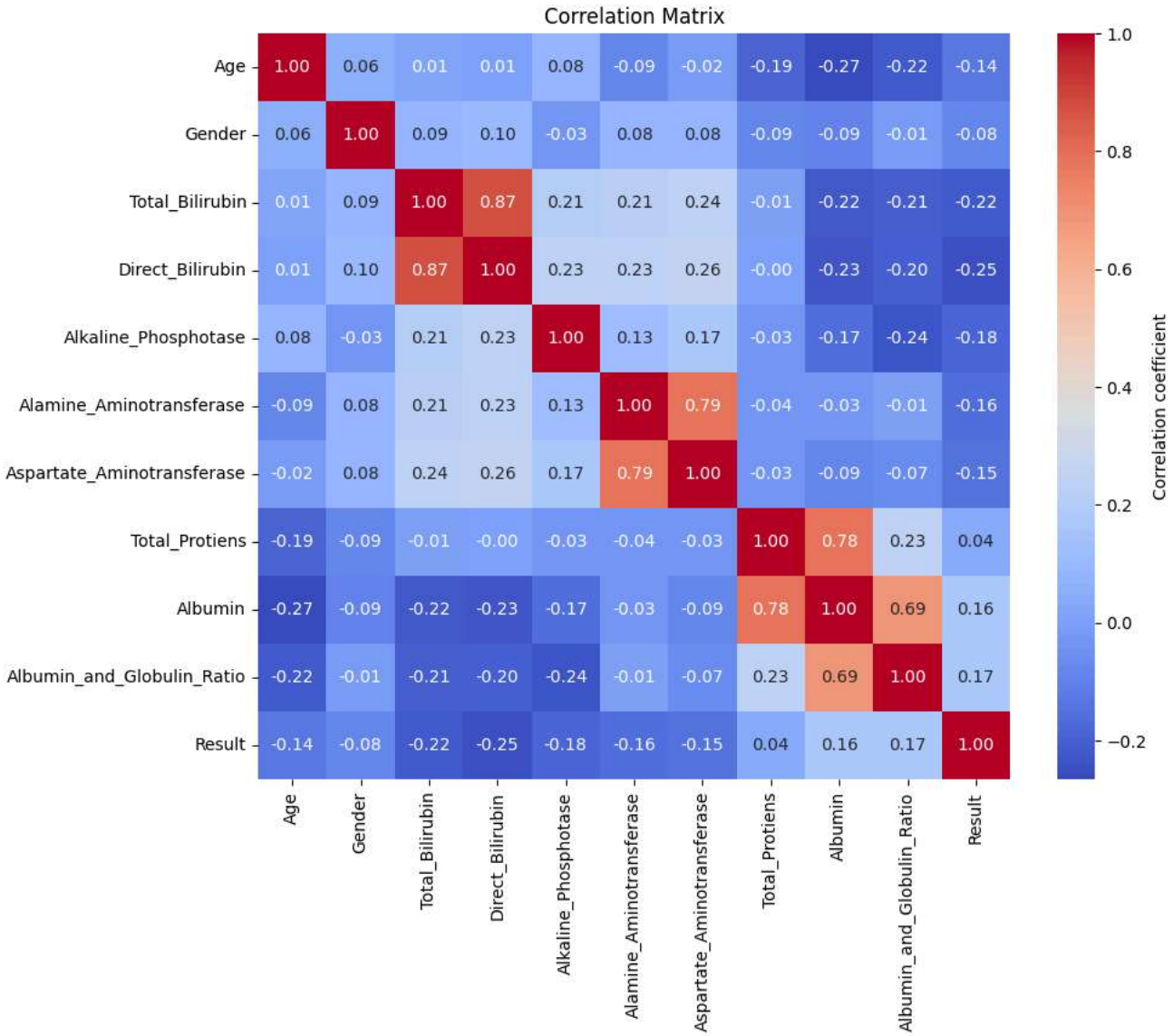
```
Uncertainty: 0.03643909515591756
```

5.C)CO-RELATION MATRIX:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data for Decision Tree
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a Decision Tree classifier
model = DecisionTreeClassifier()

# Train the model
model.fit(X_train_scaled, y_train)

# Plot feature importance
feature_importance = model.feature_importances_
feature_names = X.columns
plt.figure(figsize=(8, 6))
plt.barh(feature_names, feature_importance)
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Decision Tree Feature Importance')
plt.show()
```

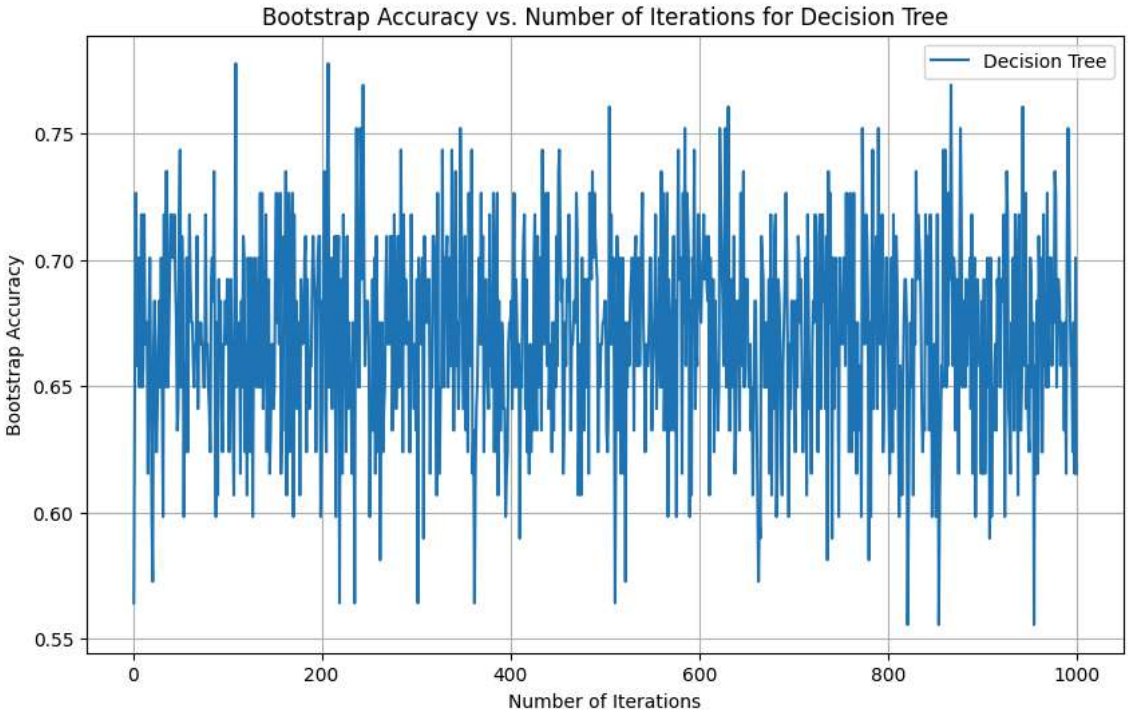


```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Calculate the correlation matrix
correlation_matrix = data.corr()

# Plot the correlation matrix using seaborn for better visualization
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar_kws={'label': 'Correlation coefficient'})
plt.title('Correlation Matrix')
plt.xticks(rotation=90)
plt.yticks(rotation=0)  # Keeps the y-axis labels horizontal
plt.show()
```



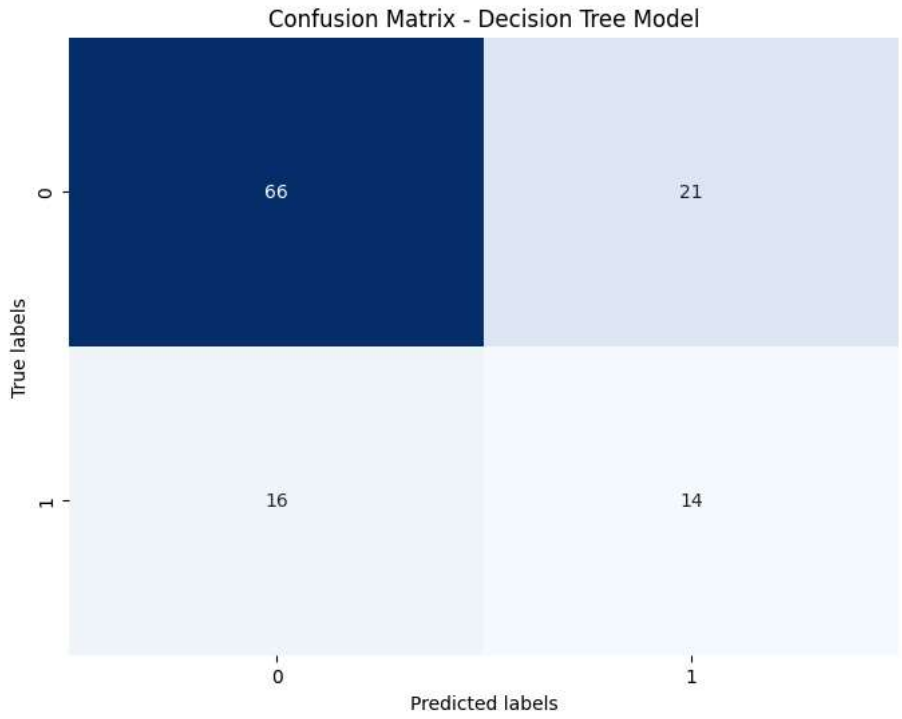
5.D)Graph between Bootstrap Accuracy vs. Number of Iterations:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Number of bootstrap samples
n_iterations = 1000
# Store scores
bootstrapped_scores_DT = []

# Run bootstrap for Decision Tree
for i in range(n_iterations):
    # Prepare train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    # Standardize features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    # Create a Decision Tree classifier
    model_DT = DecisionTreeClassifier()
    # Train the model
    model_DT.fit(X_train_scaled, y_train)
    # Make predictions
    y_pred_DT = model_DT.predict(X_test_scaled)
    # Evaluate the model
    score_DT = accuracy_score(y_test, y_pred_DT)
    bootstrapped_scores_DT.append(score_DT)

# Plotting the graph
plt.figure(figsize=(10, 6))
plt.plot(range(1, n_iterations + 1), bootstrapped_scores_DT, label='Decision Tree')
plt.title('Bootstrap Accuracy vs. Number of Iterations for Decision Tree')
plt.xlabel('Number of Iterations')
plt.ylabel('Bootstrap Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```



5.E)CONFUSION MATRIX:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Create a Decision Tree classifier
model_DT = DecisionTreeClassifier()

# Train the model
model_DT.fit(X_train_scaled, y_train)

# Make predictions
y_pred_DT = model_DT.predict(X_test_scaled)

# Compute confusion matrix
conf_matrix_DT = confusion_matrix(y_test, y_pred_DT)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_DT, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.title('Confusion Matrix - Decision Tree Model')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

# Print accuracy
accuracy_DT = accuracy_score(y_test, y_pred_DT)
print(f"Accuracy: {accuracy_DT:.2f}")
```




```
Accuracy: 0.68
```

5.F)ERROR RATES:

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split

# Load dataset
data = pd.read_csv('/content/liver.csv')  # Update the path to where your liver disease dataset is located

# Split features and target
X = data.drop('Result', axis=1)  # Assuming the target variable is named 'Result'
y = data['Result']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize Decision Tree classifier
tree = DecisionTreeClassifier(random_state=42)

# Train the Decision Tree classifier
tree.fit(X_train, y_train)

# Predictions
y_pred = tree.predict(X_test)

# Calculate errors and accuracy
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
accuracy = np.mean(y_pred == y_test) * 100  # Calculating accuracy
mae = mean_absolute_error(y_test, y_pred)

print("Decision Tree:")
print("RMSE:", rmse)
print("Accuracy:", accuracy)  # Displaying accuracy instead of MAPE
print("MAE:", mae)
```

```
Decision Tree:
RMSE: 0.5390716933750933
Accuracy: 70.94017094017094
MAE: 0.2905982905982906
```

ACCURACY OF ALL MODELS:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Features and target
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # Add more features if desired
y = data['Result']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize models
models = {
    "Support Vector Machine": SVC(kernel='linear'),
    "Logistic Regression": LogisticRegression(),
    "K-Nearest Neighbors": KNeighborsClassifier(n_neighbors=5),
    "Gaussian Naive Bayes": GaussianNB(),
    "Decision Tree": DecisionTreeClassifier()
}

# Train each model, make predictions and calculate accuracy
results = {}
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    results[name] = accuracy

# Convert results to DataFrame for display
results_df = pd.DataFrame(list(results.items()), columns=['Model', 'Accuracy'])
print(results_df)
```

```
                     Model  Accuracy
0  Support Vector Machine  0.743590
1     Logistic Regression  0.743590
2     K-Nearest Neighbors  0.709402
3    Gaussian Naive Bayes  0.452991
4           Decision Tree  0.675214
```

UNCERTAINTY OF EACH MODEL WITH BOOTSTRAPPING:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import resample
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt

# Load the dataset
file_path = '/content/liver.csv'
data = pd.read_csv(file_path)

# Prepare data
X = data[['Total_Bilirubin', 'Direct_Bilirubin']]  # You can add more features here
y = data['Result']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Number of bootstrap samples
n_iterations = 1000

# Models to train
models = {
    "SVC": SVC(kernel='linear'),
    "Logistic Regression": LogisticRegression(),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "Gaussian Naive Bayes": GaussianNB(),
    "Decision Tree": DecisionTreeClassifier()
}

results = {}

# Bootstrap loop
for name, model in models.items():
    if name == "Logistic Regression":
        coef_data = np.zeros((n_iterations, X_train_scaled.shape[1]))
    else:
        accuracies = []

    for i in range(n_iterations):
        # Resample the training data
        X_resampled, y_resampled = resample(X_train_scaled, y_train)
        # Train the model
        model.fit(X_resampled, y_resampled)

        if name == "Logistic Regression":
            # Store coefficients
            coef_data[i, :] = model.coef_[0]
        else:
            # Predict and evaluate accuracy
            y_pred = model.predict(X_test_scaled)
            accuracy = accuracy_score(y_test, y_pred)
            accuracies.append(accuracy)

    if name == "Logistic Regression":
        coef_mean = np.mean(coef_data, axis=0)
        coef_std = np.std(coef_data, axis=0)
        lower_bounds = np.percentile(coef_data, 2.5, axis=0)
        upper_bounds = np.percentile(coef_data, 97.5, axis=0)
        results[name] = (coef_mean, coef_std, lower_bounds, upper_bounds)
    else:
        accuracy_mean = np.mean(accuracies)
        accuracy_std = np.std(accuracies)
        lower_bound = np.percentile(accuracies, 2.5)
        upper_bound = np.percentile(accuracies, 97.5)
        results[name] = (accuracy_mean, accuracy_std, lower_bound, upper_bound)
```