CS304 Hw1

Hasan Alp Doyduk
S025015

ÖZYEĞİN
ÜNİVERSİTESİ

Step 1: Load and analyze the data

Step 1.a Load the two datasets

[6]
```python
# First, let's check whether the training and the test files exist in our file.
from os.path import exists
import pandas as pd

train_data = "titanictrain.csv"
test_data = "titanictest.csv"

# - Supress all warnings (Optional)
import warnings
warnings.simplefilter(action='ignore',
                      category=FutureWarning)

if exists(train_data) and exists(test_data):
    print(f"\nBoth {train_data} and {test_data} exists.")
else:
    print("Please set directory to read the files")
```

```
Both titanictrain.csv and titanictest.csv exists.
```

[7]
```python
# Let's load the training and the test data and display their types.
# Hint!: you can use the read_csv from pandas and type built-in method

import pandas as pd
from pandas import read_csv

train_df = pd.read_csv(train_data)
test_df = pd.read_csv(test_data)
```

Step 1.b Display the shape of both training and test data.

The shape of each data shows a summary of the dataset.

For example, "(890, 12)" should be interpreted as 890 samples with ten features, where

890 is the row size (or height of the data) 12 is the column size (or width of the data)

Hint!: You can use the shape method

[8]
```python
print(f"Train shape: {train_df.shape}")
print(f"Test shape: {test_df.shape}")
```

```
Train shape: (891, 12)
Test shape: (418, 11)
```

Step 1.c Display the first five rows of training and test data

```
print("\nTraining data:")
train_df.head()
```

Training data:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Next steps: Generate code with train_df View recommended plots

```
[14] print("\nTest data:")
     test_df.head()
```

Test data:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

Next steps: Generate code with test_df View recommended plots

Step 1.e Determine which column exists in the training set but is missing in the test set

Training and test sets have different number of features (columns).

In Step 1.b you saw that the traind_df has 12 columns bur test_df has 11 columns

We need to find which column is missing in the test data.

Hint!: You can access the features using keys method.

Hint!: Iterate through the keys of training samples, and check if it is in the set of test features.

```
[16] missing_columns = set(train_df.columns) - set(test_df.columns)
     print("Columns missing in test dataset:", missing_columns)
```

```
Columns missing in test dataset: {'Survived'}
```

Step 1.f Observe the datatype of each column using the .info() method

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
[19] train_df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

```
[20] test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
 2   Name         418 non-null    object
 3   Sex          418 non-null    object
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64
 6   Parch        418 non-null    int64
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

Step 1.g. Determine the number and percentage of passengers who survived.

```
survived_passengers = train_df[train_df['Survived'] == 1]
survival_percentage = len(survived_passengers) / len(train_df) * 100
print("Number of passengers who survived:", len(survived_passengers))
print("Percentage of passengers who survived:", survival_percentage)
```

```
Number of passengers who survived: 342
Percentage of passengers who survived: 38.38383838383838
```

Step 2: Prepare the data for classification

Step 2.a Extract the target label (i.e. "Survived") from the training set and assign it to the variable "y_train"

```
[24] y_train = train_df['Survived']
```

Step 2.b Copy the columns "Pclass","Sex", "Age", "Fare" to a new dataframe

Name the new dataframe as x_train Display the top 5 rows of x_train

```
x_train = train_df[['Pclass', 'Sex', 'Age', 'Fare']].copy()
x_train.head()
```

	Pclass	Sex	Age	Fare
0	3	male	22.0	7.2500
1	1	female	38.0	71.2833
2	3	female	26.0	7.9250
3	1	female	35.0	53.1000
4	3	male	35.0	8.0500

Next steps: Generate code with x_train View recommended plots

Step 2.c Add a new feature "FamilySize"

"Sibsp" and "Parch" features are related.

Instead of using them separately, we can use their sum as a feature.

Hint!: We can add the sum as a new column to the new data frame object as new_df['FamilySize'] = Sibsp + Parch

Hint!: We can also limit the sum so that it does not exceed 4. (For this purpose one option is use the apply method.)

Display the first five rows of x_train

```
x_train['FamilySize'] = train_df['SibSp'] + train_df['Parch']
x_train['FamilySize'] = x_train['FamilySize'].clip(upper=4)

print("x_train:")
x_train.head()
```

```
x_train:
```

	Pclass	Sex	Age	Fare	FamilySize
0	3	0	22.0	7.2500	1
1	1	1	38.0	71.2833	1
2	3	1	26.0	7.9250	0
3	1	1	35.0	53.1000	1
4	3	0	35.0	8.0500	0

Next steps: Generate code with x_train | View recommended plots

Step 2.g. Use a pipeline to implement Steps 2.d,e,f. OR implement them one-by-one.

```
[40] fare_mean = x_train['Fare'].mean()
     age_mean = x_train['Age'].mean()

     # Mean
     x_train_mean = x_train.fillna({'Fare': fare_mean, 'Age': age_mean})
```

```
group_means = train_df.groupby(['Sex', 'Pclass']).mean()
group_means.rename(columns={'Age': 'Mean_Age', 'Fare': 'Mean_Fare'}, inplace=True)
group_means.name = "AgeMean"
group_means
```

Sex	Pclass	PassengerId	Survived	Mean_Age	SibSp	Parch	Mean_Fare
female	1	469.212766	0.968085	34.611765	0.553191	0.457447	106.125798
	2	443.105263	0.921053	28.722973	0.486842	0.605263	21.970121
	3	399.729167	0.500000	21.750000	0.895833	0.798611	16.118810
male	1	455.729508	0.368852	41.281386	0.311475	0.278689	67.226127
	2	447.962963	0.157407	30.740707	0.342593	0.222222	19.741782
	3	455.515850	0.135447	26.507589	0.498559	0.224784	12.661633

Next steps: Generate code with group_means | View recommended plots

```python
merged_data = pd.merge(train_df, group_means, on=['Sex', 'Pclass'], how='left')

merged_data['Age'] = merged_data['Age'].fillna(merged_data['Mean_Age'])
merged_data['Fare'] = merged_data['Fare'].fillna(merged_data['Mean_Fare'])

merged_data = merged_data.drop(['Mean_Age', 'Mean_Fare'], axis=1)

print("Merged data:")
merged_data.head()
```

Merged data:

	PassengerId_x	Survived_x	Pclass	Name	Sex	Age	SibSp_x	Parch_x	Ticket	Fare	Cabin	Embarked	PassengerId_y	Survived_y	SibSp_y	Parch_y
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	455.515850	0.135447	0.498559	0.224784
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	469.212766	0.968085	0.553191	0.457447
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	399.729167	0.500000	0.895833	0.798611
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	469.212766	0.968085	0.553191	0.457447
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	455.515850	0.135447	0.498559	0.224784

Next steps: Generate code with merged_data | View recommended plots

```python
from sklearn.impute import KNNImputer

features_to_impute = ['Age', 'Fare']

knn_imputer = KNNImputer()

imputed_data = knn_imputer.fit_transform(train_df[features_to_impute])


train_df['Age'] = imputed_data[:, 0]
train_df['Fare'] = imputed_data[:, 1]

print("Imputed data using KNNImputer:")
train_df.head()
```

Imputed data using KNNImputer:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Next steps: Generate code with train_df | View recommended plots

[51]
```python
# Using MinMaxScaler()

from sklearn.preprocessing import MinMaxScaler

# Select the numerical features to scale
numerical_features = ['Fare', 'Age']

# Create the MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the scaler on the selected features
train_df[numerical_features] = scaler.fit_transform(train_df[numerical_features])

# Display the updated DataFrame
print("Scaled data using MinMaxScaler:")
train_df.head()
```

Scaled data using MinMaxScaler:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	0.271174	1	0	A/5 21171	0.014151	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	0.472229	1	0	PC 17599	0.139136	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	0.321438	0	0	STON/O2. 3101282	0.015469	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	0.434531	1	0	113803	0.103644	C123	S
4	5	0	3	Allen, Mr. William Henry	male	0.434531	0	0	373450	0.015713	NaN	S

Next steps: Generate code with train_df | View recommended plots

Step 3: Train two different ML models and compare their accuracies

Step 3.a Split into training and test set, ratio: 80/20

```
from sklearn.model_selection import train_test_split

x_train_split, x_test_split, y_train_split, y_test_split = train_test_split(x_train, y_train, test_size=0.2, random_state=42)

# Display the shapes of the split data
print("Training features shape:", x_train_split.shape)
print("Test features shape:", x_test_split.shape)
print("Training target shape:", y_train_split.shape)
print("Test target shape:", y_test_split.shape)
```

```
Training features shape: (712, 5)
Test features shape: (179, 5)
Training target shape: (712,)
Test target shape: (179,)
```

Step 3.b Train a logistic regression classifier and test the accuracy

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import OneHotEncoder


# One-Hot Encoding for 'Sex' column
encoder = OneHotEncoder()
x_train_encoded = encoder.fit_transform(x_train[['Sex']])

log_reg = LogisticRegression(max_iter=1000, random_state=42)

# Train the classifier
log_reg.fit(x_train_encoded, y_train)


# One-Hot Encoding for 'Sex' column in the test set
x_test_encoded = encoder.transform(x_test_split[['Sex']])

y_pred_log_reg = log_reg.predict(x_test_encoded)

# Calculate the accuracy
accuracy_log_reg = accuracy_score(y_test_split, y_pred_log_reg)
print("Accuracy of logistic regression classifier:", accuracy_log_reg)
```

```
Accuracy of logistic regression classifier: 0.7821229050279329
```

Step 3.c Train a random forest classifier and test the accuracy

[72]
```
# Train the classifier
random_forest = RandomForestClassifier()
random_forest.fit(x_train_encoded, y_train)

# Predict the labels for the test set
y_pred_random_forest = random_forest.predict(x_test_encoded)
```

Step 3.d Train a logistic regression classifier using 5-fold cross validation.

[76]
```
from sklearn.model_selection import cross_val_score
from sklearn.impute import SimpleImputer
import numpy as np

# Initialize the logistic regression model
logistic_regression = LogisticRegression()

# Initialize the imputer with strategy='mean' to replace missing values with the mean
imputer = SimpleImputer(strategy='mean')

# Fit the imputer on the training data and transform the training data
x_train_imputed = imputer.fit_transform(x_train.drop(columns=['Sex']))

# Perform 5-fold cross-validation
cv_scores = cross_val_score(logistic_regression, np.concatenate((x_train_encoded.toarray(), x_train_imputed), axis=1), y_train, cv=5)

# Calculate the mean accuracy
mean_accuracy_logistic_regression = cv_scores.mean()
print("Mean accuracy of logistic regression classifier (5-fold cross-validation):", mean_accuracy_logistic_regression)
```

```
Mean accuracy of logistic regression classifier (5-fold cross-validation): 0.7901261691042623
```

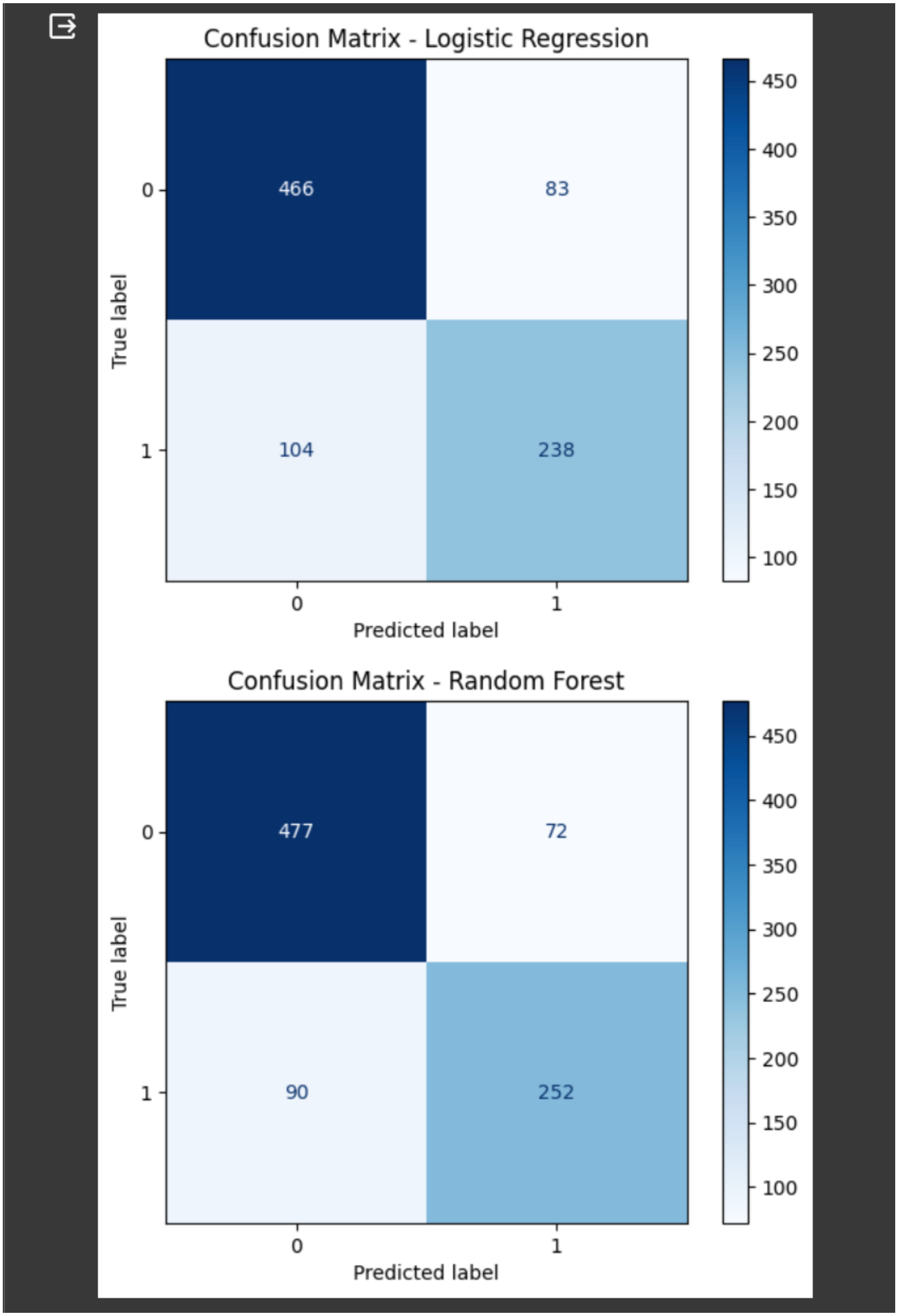
Step 3.e Train a random forest classifier using 5-fold cross validation.

[78]
```
from sklearn.ensemble import RandomForestClassifier

# Initialize the imputer with strategy='mean' to replace missing values with the mean
imputer = SimpleImputer(strategy='mean')

# Fit the imputer on the training data and transform the training data
x_train_imputed = imputer.fit_transform(x_train_encoded)

# Now, you can proceed with cross-validation using RandomForestClassifier
# Initialize the random forest classifier
random_forest = RandomForestClassifier(random_state=42)

# Perform 5-fold cross-validation
cv_scores_rf = cross_val_score(random_forest, x_train_imputed, y_train, cv=5)

# Calculate the mean accuracy
mean_accuracy_random_forest = cv_scores_rf.mean()
print("Mean accuracy of random forest classifier (5-fold cross-validation):", mean_accuracy_random_forest)
```

```
Mean accuracy of random forest classifier (5-fold cross-validation): 0.7867365513778168
```

Step 3.f Inspect the confusion matrices of the two classifiers

```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Convert encoded data to array
x_train_encoded_array = x_train_encoded.toarray()

# Combine encoded data with the remaining features in x_train
x_train_processed = np.concatenate((x_train_encoded_array, x_train.drop(columns=['Sex']).values), axis=1)

# Impute missing values with mean
imputer = SimpleImputer(strategy='mean')
x_train_imputed = imputer.fit_transform(x_train_processed)

# Initialize the logistic regression classifier
logistic_regression = LogisticRegression(max_iter=1000, random_state=42)

# Perform cross-validation with logistic regression
y_pred_log_reg = cross_val_predict(logistic_regression, x_train_imputed, y_train, cv=5)
cm_log_reg = confusion_matrix(y_train, y_pred_log_reg)
disp_log_reg = ConfusionMatrixDisplay(confusion_matrix=cm_log_reg, display_labels=np.unique(y_train))
disp_log_reg.plot(cmap='Blues')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()

# Initialize the random forest classifier
random_forest = RandomForestClassifier(random_state=42)

# Perform cross-validation with random forest
y_pred_rf = cross_val_predict(random_forest, x_train_imputed, y_train, cv=5)
cm_rf = confusion_matrix(y_train, y_pred_rf)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=np.unique(y_train))
disp_rf.plot(cmap='Blues')
plt.title('Confusion Matrix - Random Forest')
plt.show()
```



Confusion Matrix - Logistic Regression
466
83
104
238
True label
Predicted label
0
1
450
400
350
300
250
200
150
100
Confusion Matrix - Random Forest
477
72
90
252
True label
Predicted label
0
1
450
400
350
300
250
200
150
100

Step 3.g Calculate the precision and recall scores of the two classifiers

```
from sklearn.metrics import precision_score, recall_score, f1_score

# Precision and recall scores for logistic regression classifier
precision_log_reg = precision_score(y_train, y_pred_log_reg)
recall_log_reg = recall_score(y_train, y_pred_log_reg)
f1_log_reg = f1_score(y_train, y_pred_log_reg)

print("Logistic Regression Classifier:")
print("Precision:", precision_log_reg)
print("Recall:", recall_log_reg)
print("F1 Score:", f1_log_reg)

# Precision and recall scores for random forest classifier
precision_rf = precision_score(y_train, y_pred_rf)
recall_rf = recall_score(y_train, y_pred_rf)
f1_rf = f1_score(y_train, y_pred_rf)

print("\nRandom Forest Classifier:")
print("Precision:", precision_rf)
print("Recall:", recall_rf)
print("F1 Score:", f1_rf)
```

```
Logistic Regression Classifier:
Precision: 0.7414330218068536
Recall: 0.695906432748538
F1 Score: 0.717948717948718

Random Forest Classifier:
Precision: 0.7777777777777778
Recall: 0.7368421052631579
F1 Score: 0.7567567567567567
```

Step 3.h Draw the precision-recall curves of the two classifiers.

0 sn.

```
from sklearn.metrics import precision_recall_curve

# Assuming you have trained Logistic Regression and Random Forest classifiers
# Fit the Logistic Regression model
logistic_regression.fit(x_train, y_train)

# Fit the Random Forest model
random_forest.fit(x_train, y_train)

# Make predictions for the training data to get the scores
y_scores_lr = logistic_regression.decision_function(x_train)
y_scores_rf = random_forest.predict_proba(x_train)[:, 1]

# Precision-recall curve for Logistic Regression
precision_lr, recall_lr, _ = precision_recall_curve(y_train, y_scores_lr)

# Precision-recall curve for Random Forest
precision_rf, recall_rf, _ = precision_recall_curve(y_train, y_scores_rf)

# Plot the precision-recall curves
plt.figure(figsize=(10, 6))
plt.plot(recall_lr, precision_lr, label='Logistic Regression')
plt.plot(recall_rf, precision_rf, label='Random Forest')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.grid(True)
plt.show()
```