Homework 03: You will run Logistic Regression, K-Nearest Neighbor, Decision Tree, and Random Forest Classifier to predict survival for the Titanic Dataset.

Then, you will check and print the performance of your model.

First, get all your required packages. Note: the list below is not exhaustive, if you need more packages, please import them as needed.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import sklearn.metrics as metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

We are going to obtain the Titanic dataset from DataScienceDojo's github page. Thank you to them for the readily available data.

Here is the link: https://github.com/datasciencedojo/datasets/blob/master/titanic.csv

Import the file as a DataFrame called `titanic`.

```
# https://github.com/datasciencedojo/datasets/blob/master/titanic.csv

titanic = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets
titanic.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599

`Survived` is your target variable, also known as your dependent variable.

Your attributes/features/independent variables will help predict survival on the Titanic.

But first, you need to preprocess the data.

- Note: `Survived` is your target variable (**Y**).
- `Pclass`, `Sex`, `Age`, `SibSp`, `Parch`, and `Fare` will certainly be important predictors of whether a passenger survived or not. Hence, they will be included in your attributes list (**X**).
- The `Name`, `Ticket`, and `Cabin` are not useful features. Someone's name has no bearing on whether they survive or not. Similarly, a ticket number is just a unique identifier for a passenger - it is not meaningful, ordered data. So we can drop these 3. **NOTE**: NEVER drop variables from the original dataset. Either create a new df for relevant features, or create a copy of titanic and then drop the ones you do not want.
- Where they `Embarked` can be meaningful, but the data is a string variable. Let us convert it to an integer. This can be done with `np.where` or with **label encoding**. I will help you with this step. You have to do the rest of the preprocessing steps.

```
titanic.Embarked.unique()
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
titanic['embarked'] = le.fit_transform(titanic['Embarked'])

titanic.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599

Perfect! Now You can use Pclass, Sex, Age, SibSp, Parch, Fare, and embarked as your features.

Step 1: First shuffle your dataset. Step 2: Create X and Y arrays. You can refer to the class file for this. Y is the target (single column), X comprises all the relevant features.

```
titanic = titanic.sample(frac=1, random_state=42).reset_index(drop=True)
```

```
X = titanic[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'embarked']]
y = titanic['Survived']
```

1. Logistic Regression

Use this model to predict survival on the Titanic.

Part I:

First, use a 80:20 train-test split. Run your logistic regression prediction model.

Then, report the accuracy, precision, recall, f-score, sensitivity, specificity, and the confusion matrix. Plot the ROC curve.

Part II: Repeat all the above with a 60:40 split.

Compare the results between Part 1 and Part 2. Which split gave you better results in your opinion?

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer

# X = titanic[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'embarked']]
y = titanic['Survived']

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

# Convert 'Sex' column to numerical values *before* imputation
X_train['Sex'] = np.where(X_train['Sex'] == 'female', 1, 0)
X_test['Sex'] = np.where(X_test['Sex'] == 'female', 1, 0)

# Impute missing values using the mean
imputer = SimpleImputer(strategy='mean')  # or strategy='median', etc.
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Convert back to DataFrame for easier handling if needed
X_train = pd.DataFrame(X_train, columns=['Pclass', 'Sex', 'Age', 'SibSp', 'Parch'
X_test = pd.DataFrame(X_test, columns=['Pclass', 'Sex', 'Age', 'SibSp', 'Parch',


# Now you can proceed with fitting the model
model = LogisticRegression(max_iter=1000)  # Increase max_iter if needed
model.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sco
```

```
# Assuming X and y are already defined as in your previous code

# --- 80:20 Split ---
X_train_80, X_test_80, y_train_80, y_test_80 = train_test_split(X, y, test_size=0
X_train_80['Sex'] = np.where(X_train_80['Sex'] == 'female', 1, 0)
X_test_80['Sex'] = np.where(X_test_80['Sex'] == 'female', 1, 0)
imputer_80 = SimpleImputer(strategy='mean')
X_train_80 = imputer_80.fit_transform(X_train_80)
X_test_80 = imputer_80.transform(X_test_80)
model_80 = LogisticRegression(max_iter=1000)
model_80.fit(X_train_80, y_train_80)
y_pred_80 = model_80.predict(X_test_80)

# --- 60:40 Split ---
X_train_60, X_test_60, y_train_60, y_test_60 = train_test_split(X, y, test_size=0
X_train_60['Sex'] = np.where(X_train_60['Sex'] == 'female', 1, 0)
X_test_60['Sex'] = np.where(X_test_60['Sex'] == 'female', 1, 0)
imputer_60 = SimpleImputer(strategy='mean')
X_train_60 = imputer_60.fit_transform(X_train_60)
X_test_60 = imputer_60.transform(X_test_60)
model_60 = LogisticRegression(max_iter=1000)
model_60.fit(X_train_60, y_train_60)
y_pred_60 = model_60.predict(X_test_60)

# --- Evaluation and Comparison ---
def evaluate_model(y_true, y_pred, split_name):
    print(f"--- {split_name} Split ---")
    print("Accuracy:", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred))
    print("Recall:", recall_score(y_true, y_pred))
    print("F1-score:", f1_score(y_true, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))
    print("Classification Report:\n", classification_report(y_true, y_pred))
    print("\n")

evaluate_model(y_test_80, y_pred_80, "80:20")
evaluate_model(y_test_60, y_pred_60, "60:40")

# --- Conclusion ---
# Compare the evaluation metrics (accuracy, precision, recall, F1-score)
# for both splits and decide which one performed better.
# Consider the trade-off between having more data for training (80:20)
# and having a larger test set for more robust evaluation (60:40).
```

```
--- 80:20 Split ---
Accuracy: 0.7430167597765364
Precision: 0.6428571428571429
Recall: 0.6818181818181818
F1-score: 0.6617647058823529
Confusion Matrix:
 [[88 25]
 [21 45]]
Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.78      0.79       113
           1       0.64      0.68      0.66        66

    accuracy                           0.74       179
   macro avg       0.73      0.73      0.73       179
weighted avg       0.75      0.74      0.74       179



--- 60:40 Split ---
Accuracy: 0.7759103641456583
Precision: 0.6845637583892618
Recall: 0.7555555555555555
F1-score: 0.7183098591549296
Confusion Matrix:
 [[175  47]
 [ 33 102]]
Classification Report:
               precision    recall  f1-score   support

           0       0.84      0.79      0.81       222
           1       0.68      0.76      0.72       135

    accuracy                           0.78       357
   macro avg       0.76      0.77      0.77       357
weighted avg       0.78      0.78      0.78       357
```

Now K Nearest Neighbors:

2. KNN: Use this model to predict survival on the Titanic.

Part I:

Use a 80:20 train-test split. Run your KNN choosing 3 nearest neighbors.

Then, report the accuracy, precision, recall, f-score, sensitivity, specificity, and the confusion matrix. Plot the ROC curve.

Part II: Repeat the above with neighbors = 5.

Compare the results between Part 1 and Part 2. Which neighbor selection gave you better results?

```
# start working here. Feel free to use several separate blocks of code.


# convert Sex to 1 and 0 where 1 is female
titanic['Sex'] = np.where(titanic['Sex'] == 'female', 1, 0)
X = titanic[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'embarked']]
y = titanic['Survived']
```

Finally, Tree-Methods:

3. Decision Tree: Use this model to predict survival on the Titanic.

Part I:

Use a 80:20 train-test split.

Then, report the accuracy, precision, recall, f-score, sensitivity, specificity, and the confusion matrix. **Plot** the ROC curve.

Part II: Repeat the above with a 50:50 train test split.

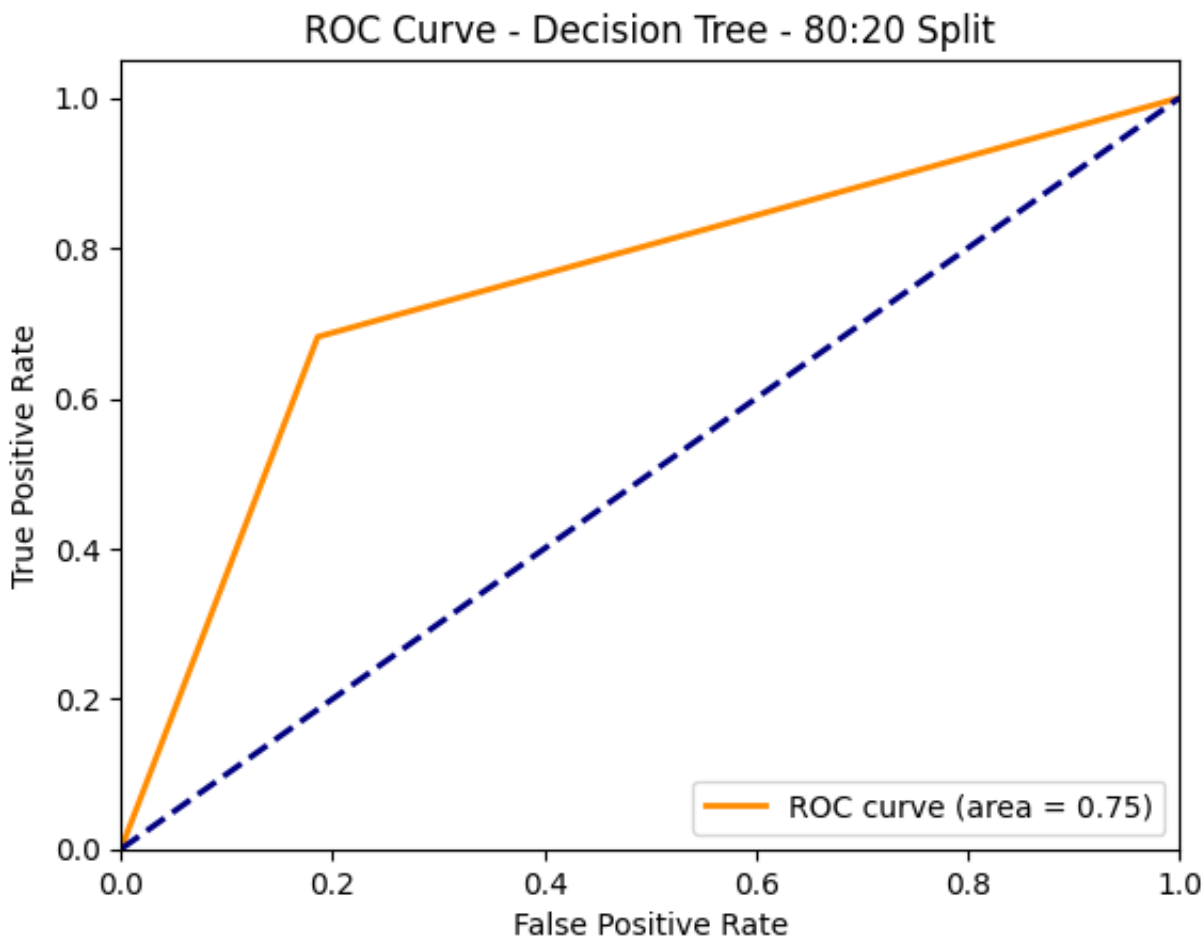
Compare the results between Part 1 and Part 2. Which split gave you better results?

Try a Random Forest Classifier as well. Works very similarly to how a decision tree does.

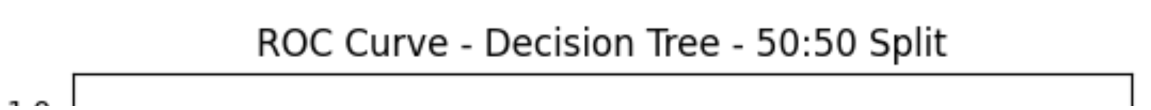
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sco
import matplotlib.pyplot as plt

# Assuming X and y are already defined

# --- Decision Tree ---

# 80:20 Split
X_train_80, X_test_80, y_train_80, y_test_80 = train_test_split(X, y, test_size=0
dt_80 = DecisionTreeClassifier(random_state=42)
dt_80.fit(X_train_80, y_train_80)
y_pred_dt_80 = dt_80.predict(X_test_80)

# 50:50 Split
X_train_50, X_test_50, y_train_50, y_test_50 = train_test_split(X, y, test_size=0
dt_50 = DecisionTreeClassifier(random_state=42)
dt_50.fit(X_train_50, y_train_50)
y_pred_dt_50 = dt_50.predict(X_test_50)

# --- Random Forest ---

# 80:20 Split
rf_80 = RandomForestClassifier(random_state=42)
rf_80.fit(X_train_80, y_train_80)
y_pred_rf_80 = rf_80.predict(X_test_80)

# 50:50 Split
rf_50 = RandomForestClassifier(random_state=42)
rf_50.fit(X_train_50, y_train_50)
y_pred_rf_50 = rf_50.predict(X_test_50)

# --- Evaluation ---

def evaluate_model(y_true, y_pred, split_name, model_name):
    print(f"--- {model_name} - {split_name} Split ---")
    print("Accuracy:", accuracy_score(y_true, y_pred))
    print("Precision:", precision_score(y_true, y_pred))
    print("Recall:", recall_score(y_true, y_pred))
    print("F1-score:", f1_score(y_true, y_pred))
```

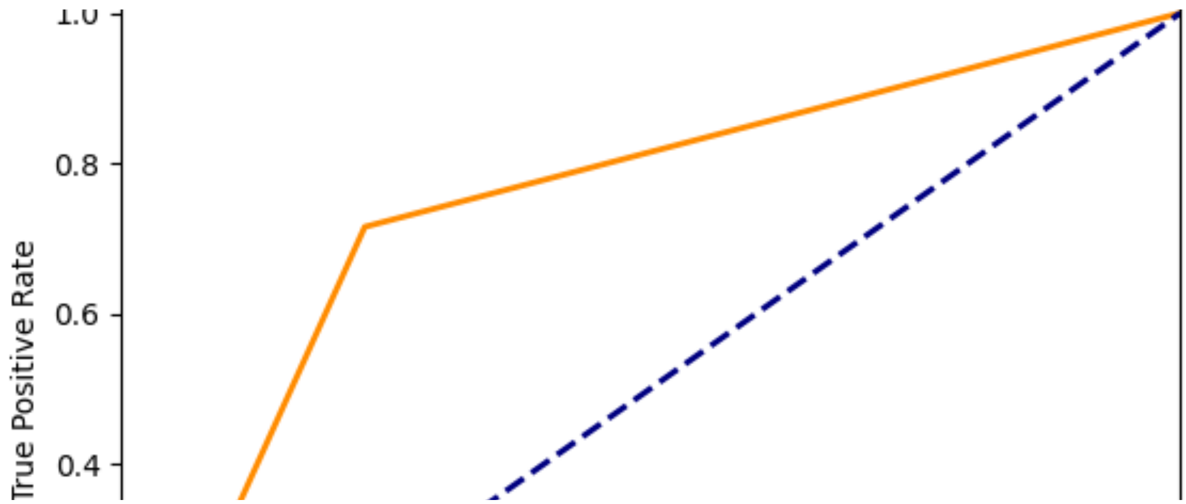
```
    # ... (add sensitivity, specificity, confusion matrix calculations here) ...
    print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))
    print("Classification Report:\n", classification_report(y_true, y_pred))
    print("\n")

    # ROC Curve
    fpr, tpr, thresholds = roc_curve(y_true, y_pred)
    roc_auc = auc(fpr, tpr)
    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)'
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve - {model_name} - {split_name} Split')
    plt.legend(loc="lower right")
    plt.show()

evaluate_model(y_test_80, y_pred_dt_80, "80:20", "Decision Tree")
evaluate_model(y_test_50, y_pred_dt_50, "50:50", "Decision Tree")
evaluate_model(y_test_80, y_pred_rf_80, "80:20", "Random Forest")
evaluate_model(y_test_50, y_pred_rf_50, "50:50", "Random Forest")

# --- Conclusion ---
# Compare the evaluation metrics and ROC curves for both splits
# and both models to determine which combination performed better.
```

```
--- Decision Tree - 80:20 Split ---
Accuracy: 0.7653631284916201
Precision: 0.6818181818181818
Recall: 0.6818181818181818
F1-score: 0.6818181818181818
Confusion Matrix:
 [[92 21]
 [21 45]]
Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.81      0.81       113
           1       0.68      0.68      0.68        66

    accuracy                           0.77       179
   macro avg       0.75      0.75      0.75       179
weighted avg       0.77      0.77      0.77       179
```




```
--- Decision Tree - 50:50 Split ---
Accuracy: 0.7488789237668162
Precision: 0.6612903225806451
Recall: 0.7151162790697675
F1-score: 0.6871508379888268
Confusion Matrix:
 [[211  63]
 [ 49 123]]
Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.77      0.79       274
           1       0.66      0.72      0.69       172

    accuracy                           0.75       446
   macro avg       0.74      0.74      0.74       446
weighted avg       0.75      0.75      0.75       446
```




1.0
0.8
0.6
0.4
True Positive Rate