Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies 2 in CpE 1

ACTIVITY NO.	TITLE
Name	Beato, Danica Marie L.
Section	CPE32S3
Date Performed:	02/21/2024
Date Submitted:	02/21/2024
Instructor:	Engr. Roman Richard

PART 1:

1. Importing Libraries and Data
2. Plot the Data

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
```

```
titanicTest = '/content/titanic_test.csv'
titanicTrain = '/content/titanic_train.csv'
testFrame = pd.read_csv(titanicTest)
trainFrame = pd.read_csv(titanicTrain)
```

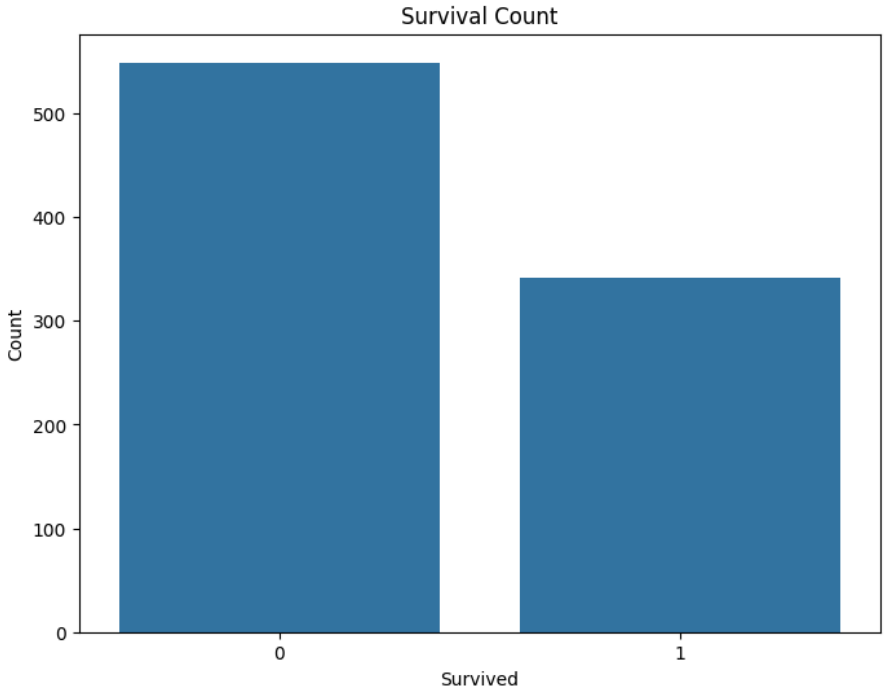
```
testFrame.head()
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emb
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	

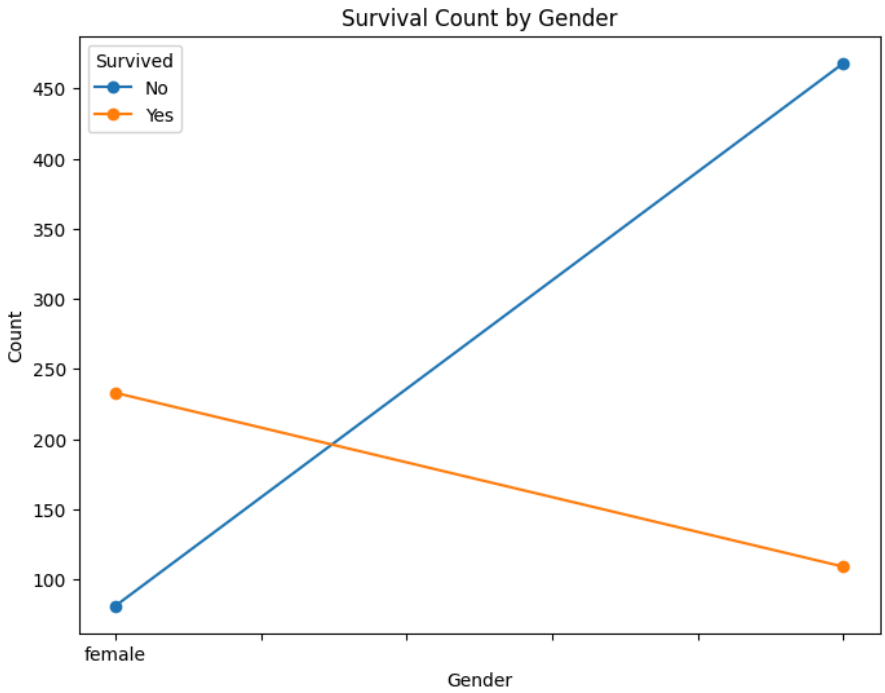
```
trainFrame.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs	female	38.0	1	0	PC 17599	71.2833

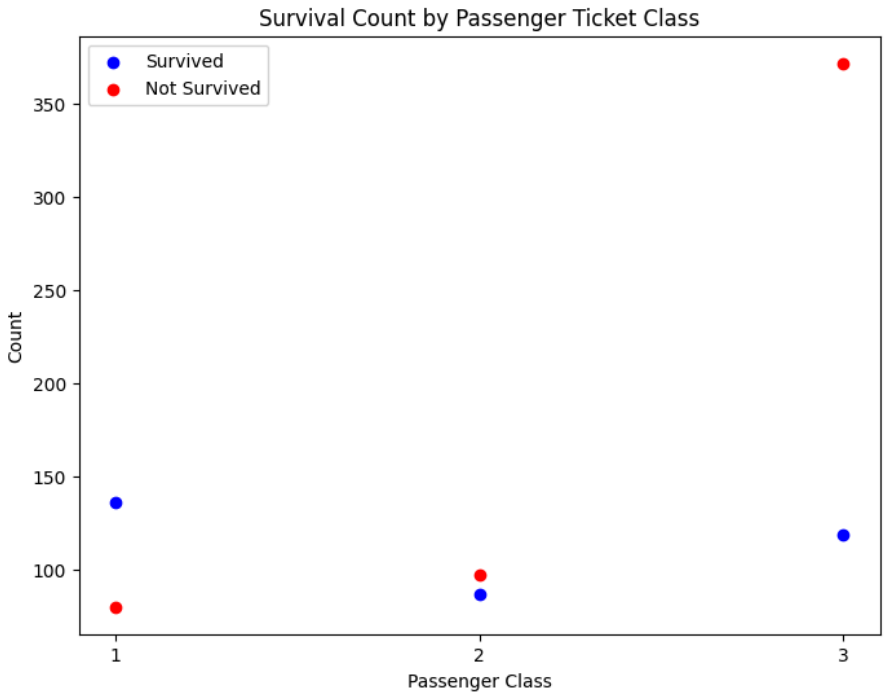
```
# This graph displays the overall survival count based on the given data
plt.figure(figsize=(8, 6))
sns.countplot(x='Survived', data=trainFrame)
plt.title('Survival Count')
plt.xlabel('Survived')
plt.ylabel('Count')
plt.show()
```




```
# This plot displays the survival count by gender
survivalGender = trainFrame.groupby(['Sex', 'Survived']).size().unstack()

survivalGender.plot(kind='line', marker='o', figsize=(8, 6))

plt.title('Survival Count by Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.legend(title='Survived', labels=['No', 'Yes'])

plt.show()
```




```
# This scatter plot shows the survival count by a passengers' ticket class
survivalPclass = trainFrame.groupby(['Pclass', 'Survived']).size().unstack()

passengerClass = survivalPclass.index
survivedCounts = survivalPclass[1]
notSurvCounts = survivalPclass[0]

plt.figure(figsize=(8, 6))
plt.scatter(passengerClass, survivedCounts, label='Survived', color='blue')
plt.scatter(passengerClass, notSurvCounts, label='Not Survived', color='red')

plt.title('Survival Count by Passenger Ticket Class')
plt.xlabel('Passenger Class')
plt.ylabel('Count')
plt.xticks(passengerClass)
plt.legend()

plt.show()
```



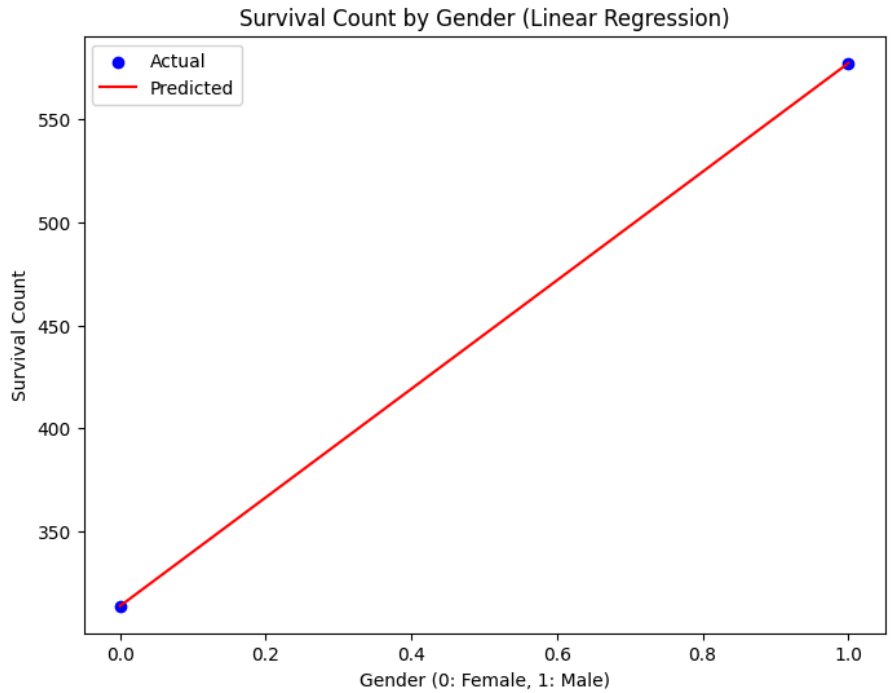

```
testFrame.corr(method = 'pearson')
```

```
<ipython-input-27-e0b61ddca6a3>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version
  testFrame.corr(method = 'pearson')
```

	PassengerId	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.026751	-0.034102	0.003818	0.043080	0.008211
Pclass	-0.026751	1.000000	-0.492143	0.001087	0.018721	-0.577147
Age	-0.034102	-0.492143	1.000000	-0.091587	-0.061249	0.337932
SibSp	0.003818	0.001087	-0.091587	1.000000	0.306895	0.171539
Parch	0.043080	0.018721	-0.061249	0.306895	1.000000	0.230046
Fare	0.008211	-0.577147	0.337932	0.171539	0.230046	1.000000

3. Perform Simple Linear Regression

```
# This is a simple linear regression plot that displays survival count of passengers by gender

survivalGender = trainFrame.groupby(['Sex', 'Survived']).size().unstack()

gender = np.array([0, 1]).reshape(-1, 1)  # 0 for female, 1 for male
survivalCount = survivalGender.sum(axis=1).values.reshape(-1, 1)

model = LinearRegression()
model.fit(gender, survivalCount)

survCountPred = model.predict(gender)

# Plot the data and linear regression line
plt.figure(figsize=(8, 6))
plt.scatter(gender, survivalCount, color='blue', label='Actual')
plt.plot(gender, survCountPred, color='red', label='Predicted')
plt.title('Survival Count by Gender (Linear Regression)')
plt.xlabel('Gender (0: Female, 1: Male)')
plt.ylabel('Survival Count')
plt.legend()
plt.show()

print("Intercept:", model.intercept_[0])
print("Coefficient (slope):", model.coef_[0][0])
```




```
Intercept: 314.0
Coefficient (slope): 262.99999999999994
```

PART 2:

Part 1: Create a Decision Tree Classifier

In this part of the lab, you will create a decision tree classifier that will learn from a labelled dataset.

The dataset contains the names and demographic details for each passenger. In addition, details of the passengers' trip are included. From this data, we can be build a decision tree that illustrates the factors that contributed to survivability, or lack of it, for the voyage.

The datasets contain the following variables:

With the data above, what kinds of questions can we ask about the factors that contributed to passengers surviving or perishing in the Titanic disaster?

> Based on the given variables, I identified at least 5 questions about the factors which may have had contributed to the passengers survival in the Titanic, listed below are the questions:
>
> - Did the passengers' age contributed to their survival?
> - Did the passengers' ticket class affect their survival rates?
> - Is there a correlation between their gender and survival rate?
> - Did having family members (siblings/spouse/parents/children) onboard affect a passengers' survival?

Step 1: Create the Dataframe

a) Import pandas and the csv file

First, import pandas and create a dataframe from the Titanic training data set, which is stored in the titanictrain.csv file. Use the pd.read_csv() method.

```
# Import pandas and the csv file
import pandas as pd
training = pd.read_csv("/content/titanic_train.csv")
```

b) Verify the import and take a look at the data.

```
# Verify the import and take a look at the data
training.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
```

Are there missing values in the data set?

> Yes, based on the values shown from the data. At least 3 variables: age, cabin, and embarked have missing values.

```
# View the first few rows of the data
training.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs	female	38.0	1	0	PC 17599	71.2833

Step 2: Prepare the Data for the Decision Tree Model

a) Replace string data with numeric labels

We will use scikit-learn to create the decision trees. The decision tree model we will be using can only handle numeric data. The values for the Gender variable must be transformed into numeric representations. 0 will be used to represent "male" and 1 will represent "female."

In this code, a lambda expression is used with the apply() dataframe method. This lambda expression represents a function that uses a conditional statement to replace the text values in the columns with the appropriate numeric value. The lambda statement can be interpreted as "if the parameter toLabel equals 'male', return 0, if the value is something else, return 1." The apply() method will execute this function on the values in every row of the "Gender" column of the dataframe.

```
# a. Replace string data with numeric labels
training["Sex"] = training["Sex"].apply(lambda toLabel: 0 if toLabel == 'male' else 1)
```

b) Verify that the Gender variable has been changed.

The output should show values of 0 or 1 for the Gender variable in the dataset.

```
# b. Verify that the Gender variable has been changed
training.head()
```

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

c) Address Missing Values in the Dataset

The output of the info() method above indicated that about 180 observations are missing the age value. The age value is important to our analysis. We must address these missing values in some way. While not ideal, we can replace these missing age values with the mean of the ages for the entire dataset.

This is done by using the fillna() method on the "Age" column in the dataset. The fillna() method will change the original dataframe by using the inplace = True argument.

```
# c. Address Missing Values in the Dataset
training["Age"].fillna(training["Age"].mean(), inplace=True)
```

d) Verify that the values have been replaced.

```
# d. Verify that the values have been replaced
training.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    int64  
 5   Age          891 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(6), object(4)
memory usage: 83.7+ KB
```

What is the value that was used to replace the missing ages?

```
training.describe()
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	0.352413	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	0.477990	13.002015	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	0.000000	22.000000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	0.000000	29.699118	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	1.000000	35.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	1.000000	80.000000	8.000000	6.000000	512.329200

Step 3: Train and Score the Decision Tree Model

a) Create an array object with the variable that will be the target for the model.

The purpose of the model is to classify passengers as survivors or victims. The dataset identifies survivors and victims. The model will learn which input variable values are most likely to belong to victims and survivors, and then use that information to classify passengers from a unique test data set.

```
# a. Create an array object with the variable that will be the target for the model
y_target = training["Survived"].values
```

b) Create an array of the values that will be the input for the model.

Only some of the features of the data are useful for creating the classifier tree. We create a list of the columns from the data that we want the classifier to use as the input variables and then create an array using the column name from that variable. The variable X_input holds the values for all the features that the model will use to learn how to make the classifications. After the model is trained, we will use this variable to assign these labels to the test data set.

```
# b. Create an array of the values that will be the input for the model
columns = ["Fare", "Pclass", "Sex", "Age", "SibSp"]
X_input = training[list(columns)].values
```

c) Create the learned model.

Import the decision tree module from the sklearn machine learning library. Create the classifier object clf_train. Then, use the fit() method of the classifier object, with the X_input and y_target variables as parameters, to train the model.

```
# c. Create the learned model
from sklearn import tree
clf_train = tree.DecisionTreeClassifier(criterion="entropy", max_depth=3)
clf_train = clf_train.fit(X_input, y_target)
```

d) Evaluate the model

Use the score() method of the decision tree object to display the percentage accuracy of the assignments made by the classifier. It takes the input and target variables as arguments.

```
# d. Evaluate the model
clf_train.score(X_input,y_target)
```

```
0.8226711560044894
```

Step 4: Visualize the Tree

a) Create the intermediate file output

Import the sklearn.externals.six StringIO module which is used to output the characteristics of the decision tree to a file. We will create a Graphviz dot file which will allow us to export the results of the classifier into a format that can be converted into a graphic.

```
# a. Create the intermediate file output
with open("/content/titanic_train.csv", 'w') as f:
  f = tree.export_graphviz(clf_train, out_file=f, feature_names=columns)
```

b) Install Graphviz

To visualize the decision tree, Graphviz needs to be installed from a terminal. The installation requires that a prompt be answered, which can't be done from a notebook code cell. Use the apt-get install graphviz command from the terminal command line to install this software.

```
!apt-get install graphviz
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
graphviz is already the newest version (2.42.2-6).
0 upgraded, 0 newly installed, 0 to remove and 35 not upgraded.
```

c) Convert the intermediate file to a graphic

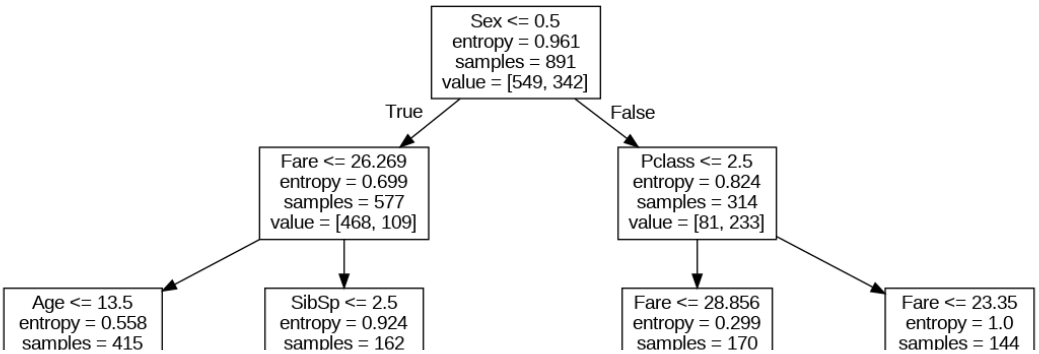
The dot file that was created above can be converted to a .png file with the graphiz dot renderer. This is a shell command, so use ! before it to run it from this noteblook. The new titanic.png graphic file should appear in the directory that contains this notebook

```
#run the Graphviz dot command to convert the .dot file to .png
!dot -Tpng /content/titanic_train.csv -o .titanic.png
```

d) Display the image

Now we will import the Image module from the IPython.display library. This will allow us to open and display an external graphics file on the notebook page. The Image function is used to display the file, with the .png file name as argument

```
#import the Image module from the Ipython.display libary
from IPython.display import Image
#display the decison tree graphic
Image(".titanic.png")
```



e.) Interpret the tree

From the tree, we can see several things. First, at the root of the tree is the Gender variable, indicating that it is the single most important factor in making the classification. The branches to the left are for Gender = 0 or male. Each root and intermediate node contains the decision factor, the entropy, and the number of passengers who fit the critierion at that point in the tree. For example, the root node indicates that there are 891 observations that make up the learning data set. At the next level, we can see that 577 people were male, and 314 were female. In the third level, at the far right, we can see that 415 people were male and paid a fare of less than 26.2686.

Finally, the leaf nodes for that intermediate node indicate that 15 of these passengers were below the age of 13.5, and the other 400 were older than that age.

Finally, the elements in the value array indicate survival. The first value is the number of people who died, and the second is the number of survivors for each criterion. The root node tells us that out of our sample, 549 people died and 342 survived.

Entropy is a measure of noise in the decision. Noise can be viewed as uncertainty. For example, in nodes in which the decision results in equal values in the survival value array, the entropy is at its highest possible value, which is 1.0. This means that the model was unable to definitively make the classification decision based on the input variables. For values of very low entropy, the decision was much more clear cut, and the difference in the number of survivors and victims is much higher.

Part 2: Apply the Decision Tree Model

In this part of the lab, we will use the results of the learned decision tree model to label an unlabelled dataset of Titanic passengers. The decision tree will evaluate the features of each observation and label the observation as survived (label = 1) or died (label = 0).

Step 1: Import and Prepare the Data

In this step, you will import and prepare the data for analysis.

a) Import the data.

Name the dataframe "testing" and import the file titanic-test.csv.

```
#import the file into the 'testing' dataframe.
testing = pd.read_csv("/content/titanic_test.csv")
```

How many records are in the data set?

> There are 418 records in the titanic test data set.

```
print(testFrame.shape[0])
```

```
418
```

b) Use a lambda expression to replace the "male" and "female" values with 0 for male and 1 for female..

```
#replace the Gender labels in the testing dataframe
testing["Sex"] = testing["Sex"].apply(lambda toLabel: 0 if toLabel ==
'male' else 1)



#Use the fillna method of the testing dataframe column "Age"
#to replace missing values with the mean of the age values.
testing["Age"].fillna(testing["Age"].mean(), inplace=True)


testing.info()
testing.head()
```

```
<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    int64
 4   Age          418 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(5), object(4)
memory usage: 36.0+ KB
```

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

Step 2: Label the testing dataset

In this step, you will apply the learned model to the testing dataset.

a) Create the array of input variables from the testing data set

```
#create the variable X_input to hold the features that the classifier will use
X_input = testing[list(columns)].values
```

b) Apply the model to the testing data set.

Use the predict() method of the clf_train object that was trained to label the observations in the testing data set with the most likely survival classification. Provide the array of input variables from the testing data set as the parameter for this method.

```
#apply the model to the testing data and store the result in a pandas dataframe.
#Use X_input as the argurment for the predict() method of the clf_train classifier object

X_input = testing[list(columns)].fillna(testing[list(columns)].mean()).values
target_labels = clf_train.predict(X_input)

#convert the target array into a pandas dataframe using the pd.DataFrame() method and target as argument
target_labels = pd.DataFrame({'Est_Survival':target_labels, 'Name':testing['Name']})
```

c) Evaluate the accuracy of the estimated labels

The ground truth for the survival of each passenger can be found in another file called all_data.csv. To select only the passengers contained in the testing dataset, we merge the target_labels dataframe and the all_data dataframe on the field Name. We then compare the estimated label with the ground truth dataframe and compute the accuracy of the learned model.

```
# Load data for all passengers in the variable all_data
all_data = pd.read_csv("/content/titanic_all.csv")

# Merging using the field Name as key, selects only the rows of the two datasets that refer to the same passenger
testing_results = pd.merge(target_labels, all_data[['Name','Survived']], on=['Name'])

# Compute the accuracy as a ratio of matching observations to total osbervations. Store this in in the variable acc.
acc = np.sum(testing_results['Est_Survival'] == testing_results['Survived']) /float(len(testing_results))

print("Accuracy: ", acc)
```

```
Accuracy:  0.7682619647355163
```

Part 3: Evaluate the Decision Tree Model

Step 1: Import the data

This time we will import the data from a csv file, but we will specify the columns that we want to have appear in the dataframe. We will do this by passing an array-like list of column names to the read_csv() method usecols parameter. Use the following columns: 'Survived', 'Fare', 'Pclass', 'Gender', 'Age', and 'SibSP'. Each should be in quotes and the list should be square brackets. Name this dataframe all_data.

```
#import the titanic_all.csv file into a dataframe called all_data. Specify thelist of columns to import.

all_data = pd.read_csv("/content/titanic_all.csv", usecols=['Survived','Pclass','Gender','Age','SibSp','Fare'])

#View info for the new dataframe
all_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1308 entries, 0 to 1307
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Survived  1308 non-null   int64  
 1   Pclass    1308 non-null   int64  
 2   Gender    1308 non-null   object 
 3   Age       1045 non-null   float64
 4   SibSp     1308 non-null   int64  
 5   Fare      1308 non-null   float64
dtypes: float64(2), int64(3), object(1)
memory usage: 61.4+ KB
```

How many records are in the data set?

> There are 1308 records in the data test.

```
print("No. of Records:", all_data.shape[0])
```

```
No. of Records: 1308
```

Which important variables(s) are missing values and how many are missing?

> There are about 200+ missing values for the age variable.

Step 2: Prepare the data.

a) Remove the "male" and "female" strings and replace them with 0 and 1 respectively.

```
#Label the gender variable with 0 and 1
all_data["Gender"] = all_data["Gender"].apply(lambda toLabel: 0 if toLabel =='male' else 1)
```

c) Replace the missing age values with the mean of the age of all members of the data set

```
#replace missing Age values with the mean age

all_data["Age"].fillna(all_data["Age"].mean(), inplace=True)

all_data.head()
```

	Survived	Pclass	Gender	Age	SibSp	Fare
0	1	1	1	29.0000	0	211.3375
1	1	1	0	0.9167	1	151.5500
2	0	1	1	2.0000	1	151.5500
3	0	1	0	30.0000	1	151.5500
4	0	1	1	25.0000	1	151.5500

Step 2: Create the input and output variables for the training and testing data

The sklearn library includes modules that help with model selection. We will import from sklearn.model_selection the train_test_split() method. This method will automatically split the entire dataset, returning in total four numpy arrays, two for the features (test and validation) and two for the labels (test and validation). One parameter of the method specifies the proportion of observations to use for testing and training. Another parameter specifies a seed value that will be used to randomize assignment of the observation to testing or training. This is used so that another user can replicate your work by receiving the same assignments of observations to datasets. The syntax of the method is:

```
train_test_split(input_X, target_y, test_size=0.4, random_state=0)
```

40% of the data will be used for testing. The random seed is set to 0.

The method returns four values. These values are the input varibles for training and testing data and the target variables for the training and testing data in that order.

a) Designate the input variables and output variables and generate the arrays.

```
#Import train_test_split() from the sklearn.model_selection libary
from sklearn.model_selection import train_test_split

#create the input and target variables as uppercase X and lowercase y. Reuse the columns variable.
columns = ["Fare", "Pclass", "Gender", "Age", "SibSp"]
X = all_data[list(columns)].values
y = all_data["Survived"].values

#generate the four testing and training data arrays with the train_test_split() method
X_train,X_test,y_train,y_test=train_test_split(X, y, test_size=0.40, random_state=0)
```

b) Train the model and fit it to the testing data.

Now the model can be fit again. The model will be trained using only the training datat, as selected by the train_test_split function.

```
#create the training decision tree object
```