LOGISTICS REGRESSION-MACHINE LEARNING: SUPERVISED LEARNING

Predict Titanic Survival

The RMS Titanic set sail on its maiden voyage in 1912, crossing the Atlantic from Southampton, England to New York City. The ship never completed the voyage, sinking to the bottom of the Atlantic Ocean after hitting an iceberg, bringing down 1,502 of 2,224 passengers onboard.

In this project you will create a Logistic Regression model that predicts which passengers survived the sinking of the Titanic, based on features like age and class.

The data we will be using for training our model is provided by Kaggle. Feel free to make the model better on your own and submit it to the Kaggle Titanic competition!

If you get stuck during this project or would like to see an experienced developer complete it, check out the **project walkthrough video** which can be found in the "get help" menu in the bottom-right of this window.

16/16Complete

Mark the tasks as complete by checking them off

Load the Data

1.

The file `passengers.csv` contains the data of `892` passengers onboard the Titanic when it sank that fateful day. Let's begin by loading the data into a pandas DataFrame named `passengers`. Print `passengers` and inspect the columns. What features could we use to predict survival?

Hint

Use `pandas`' `.read_csv()` method to load the data with the below syntax:

```
pd.read_csv('file_name')
```

Clean the Data

2.

Given the saying, "women and children first," `Sex` and `Age` seem like good features to predict survival. Let's map the text values in the `Sex` column to a numerical value. Update `Sex` such that all values `female` are replaced with `1` and all values `male` are replaced with `0`.

Hint

We can update the `Sex` column in `passengers` with the following syntax:

```
passengers['Sex'] = expression_for_new_values
```

To map the values in `Sex` we can use `pandas`' `.map()` method with the below syntax:

```
data_frame(['column']).map({'initial_value_1':'updated_value_1','initial_value_2':'updated_value_2'})
```

3.

Let’s take a look at `Age`. Print `passengers['Age'].values`. You can see we have multiple missing values, or `nan`s. Fill all the empty `Age` values in passengers with the mean age.
Hint

The `.fillna()` method allows us to fill all the missing values in a column with the below syntax:

```
data_frame(['column']).fillna(value='value_to_replace_nan',inplace=True)
```

- `data_frame` is the name of the DataFrame
- `column` is name of the column in which we want to fill missing values
- `value_to_replace_nan` is the value that will replace the missing values
- `inplace=True` fills the missing values in our DataFrame rather than returning a new DataFrame

4.

Given the strict class system onboard the Titanic, let’s utilize the `Pclass` column, or the passenger class, as another feature. Create a new column named `FirstClass` that stores `1` for all passengers in first class and `0` for all other passengers.
Hint

We can create the `FirstClass` column in `passengers` with the following syntax:

```
passengers['FirstClass'] = expression_for_column_values
```

To set the value of `FirstClass` for each passenger to `1` or `0`, we can use `pandas`’ `.apply()` method on `Pclass` as shown below:

```
passengers['Pclass'].apply(lambda x: 1 if x == 1 else 0)
```

5.

Create a new column named `SecondClass` that stores `1` for all passengers in second class and `0` for all other passengers.

Print `passengers` and inspect the DataFrame to ensure all the updates have been made.
Hint

We can create the `SecondClass` column in `passengers` with the following syntax:

```
passengers['SecondClass'] = expression_for_column_values
```

To set the value of `SecondClass` for each passenger to `1` or `0`, we can use `pandas`’ `.apply()` method on `Pclass` as shown below:

```
passengers['Pclass'].apply(lambda x: 1 if x == 2 else 0)
```

Select and Split the Data

6.

Now that we have cleaned our data, let's select the columns we want to build our model on. Select columns `Sex`, `Age`, `FirstClass`, and `SecondClass` and store them in a variable named `features`. Select column `Survived` and store it a variable named `survival`.
Hint

We can select an individual column from a `pandas` DataFrame using the following syntax:

```
selection = data_frame['column_to_select']
```

To select multiple columns from a DataFrame:

```
selection =  data_frame[['column_1','column_2', ..., 'column_n']]
```

7.

Split the data into training and test sets using `sklearn`'s `train_test_split()` method. We'll use the training set to train the model and the test set to evaluate the model.
Hint

We can create training and test sets using `train_test_split()` with the below syntax:

```
train_test_split(features,labels,test_size = 0_to_1_test_size)
```

- `features` is the feature data
- `labels` are the labels for the feature data
- `0_to_1_test_size` is a number from 0 to 1 indicating the percentage of data that should be saved for the test set

`train_test_split()` will return, in this order, the training features (`X_train`), the testing features (`X_test`), the training labels (`y_train`), and the testing labels (`y_test`).

Normalize the Data

8.

Since `sklearn`'s Logistic Regression implementation uses *Regularization*, we need to scale our feature data. Create a `StandardScaler` object, `.fit_transform()` it on the training features, and `.transform()` the test features.
Hint

We can create a `StandardScaler` object with the below syntax:

```
scaler = StandardScaler()
```

To determine the scaling factors and apply the scaling to the feature data:

```
train_features = scaler.fit_transform(train_features)
```

To apply the scaling to the test data:

```
test_features = scaler.transform(test_features)
```

Create and Evaluate the Model

9.

Create a `LogisticRegression` model with `sklearn` and `.fit()` it on the training data.

Fitting the model will perform gradient descent to find the feature coefficients that minimize the log-loss for the training data.

Hint

To create a `LogisticRegression` model in `sklearn`:

```
model = LogisticRegression()
```

To `.fit()` the model to training data:

```
model.fit(X_train, y_tain)
```

10.

`.score()` the model on the training data and print the training score.

Scoring the model on the training data will run the data through the model and make final classifications on survival for each passenger in the training set. The score returned is the percentage of correct classifications, or the accuracy.

Hint

To score an `sklearn` model on the training data:

```
model.score(X_train, y_train)
```

11.

`.score()` the model on the test data and print the test score.

Similarly, scoring the model on the testing data will run the data through the model and make final classifications on survival for each passenger in the test set.

How well did your model perform?

Hint

To score an `sklearn` model on the testing data:

```
model.score(X_test, y_test)
```

12.

Print the feature coefficients determined by the model. Which feature is most important in predicting survival on the sinking of the Titanic?
Hint

To print the coefficients of the model, access the `.coef_` attribute:

```
print(model.coef_)
```

To print each feature with its respectice coefficient value, you can use the following expression:

```
print(list(zip(['Sex','Age','FirstClass','SecondClass'],model.coef_[0])))
```

Predict with the Model

13.

Let’s use our model to make predictions on the survival of a few fateful passengers. Provided in the code editor is information for 3rd class passenger `Jack` and 1st class passenger `Rose`, stored in `NumPy` arrays. The arrays store 4 feature values, in the following order:

- `Sex`, represented by a `0` for male and `1` for female
- `Age`, represented as an integer in years
- `FirstClass`, with a `1` indicating the passenger is in first class
- `SecondClass`, with a `1` indicating the passenger is in second class

A third array, `You`, is also provided in the code editor with empty feature values. Uncomment the line containing `You` and update the array with your information, or the information for some fictitious passenger. Make sure to enter all values as floats with a `.`!
Hint

If you or your fictitious passenger identify as non-binary, the `Sex` value can be `0.5`.

If you or your fictitious passenger are sailing in 1st class, the `FirstClass` value should be `1` and the `SecondClass` value should be `0`.

If you or your fictitious passenger are sailing in 2nd class, the `FirstClass` value should be `0` and the `SecondClass` value should be `1`.

If you or your fictitious passenger are sailing in 3rd class, the `FirstClass` value should be `0` and the `SecondClass` value should be `0`.

14.

Combine `Jack`, `Rose`, and `You` into a single `NumPy` array named `sample_passengers`.
Hint

To combine the arrays for each passenger into a single `NumPy` array, use the following syntax:

```
combined_arrays = np.array([ ___ , ___, ___ ])
```

where the blanks are replaced with `Jack`, `Rose`, and `You`.

15.

Since our Logistic Regression model was trained on scaled feature data, we must also scale the feature data we are making predictions on. Using the `StandardScaler` object created earlier, apply its `.transform()` method to `sample_passengers` and save the result to `sample_passengers`.

Print `sample_passengers` to view the scaled features.
Hint

To scale the features of our sample passengers according to the same scaling as the training data, use the following syntax:

```
sample_passengers = scaler.transform(sample_passengers)
```

16.

Who will survive, and who will sink? Use your model's `.predict()` method on `sample_passengers` and print the result to find out.

Want to see the probabilities that led to these predictions? Call your model's `.predict_proba()` method on `sample_passengers` and print the result. The 1st column is the probability of a passenger perishing on the Titanic, and the 2nd column is the probability of a passenger surviving the sinking (which was calculated by our model to make the final classification decision).

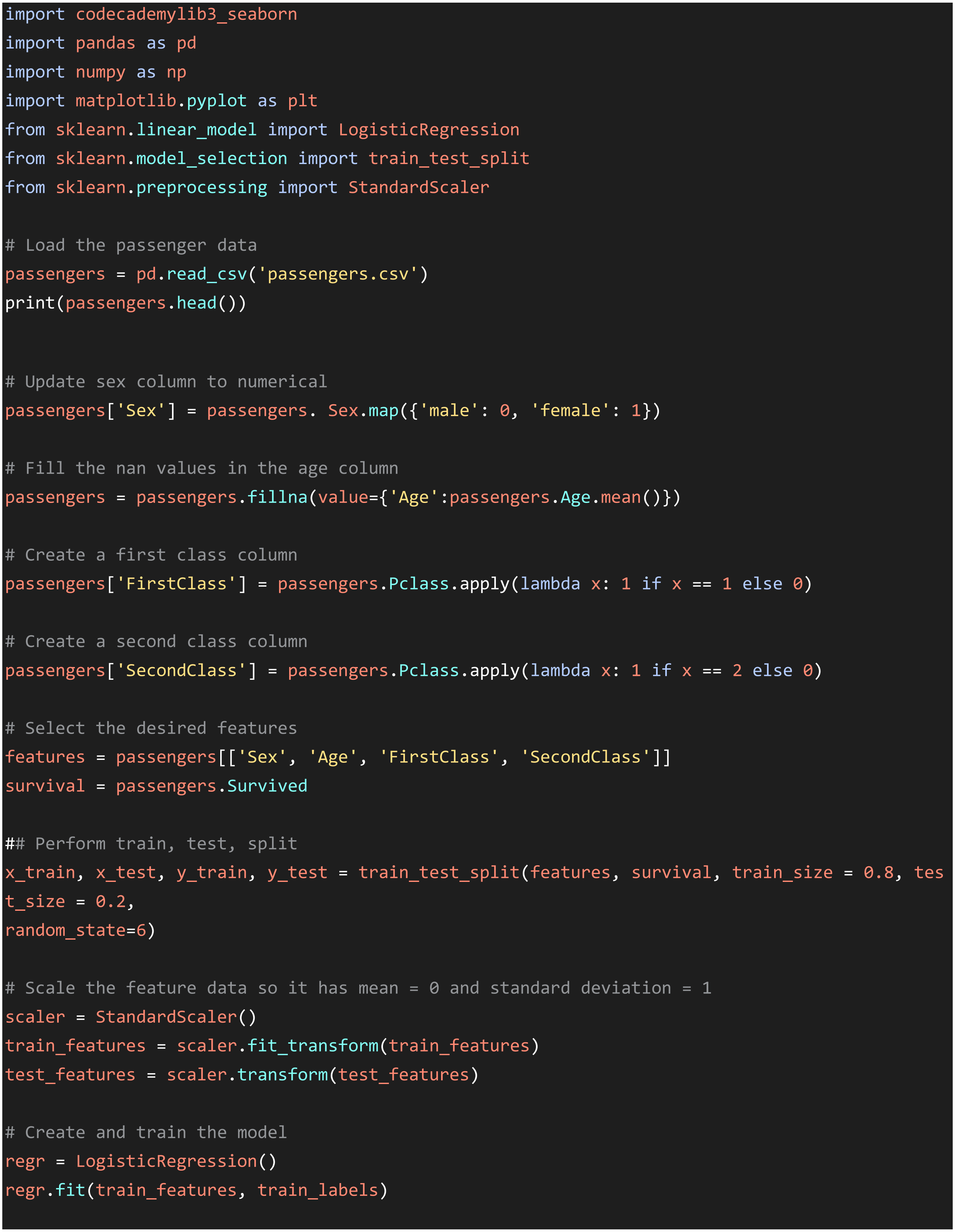
Hint
To make survival classifications on `sample_passengers`:

```
model.predict(sample_passengers)
```

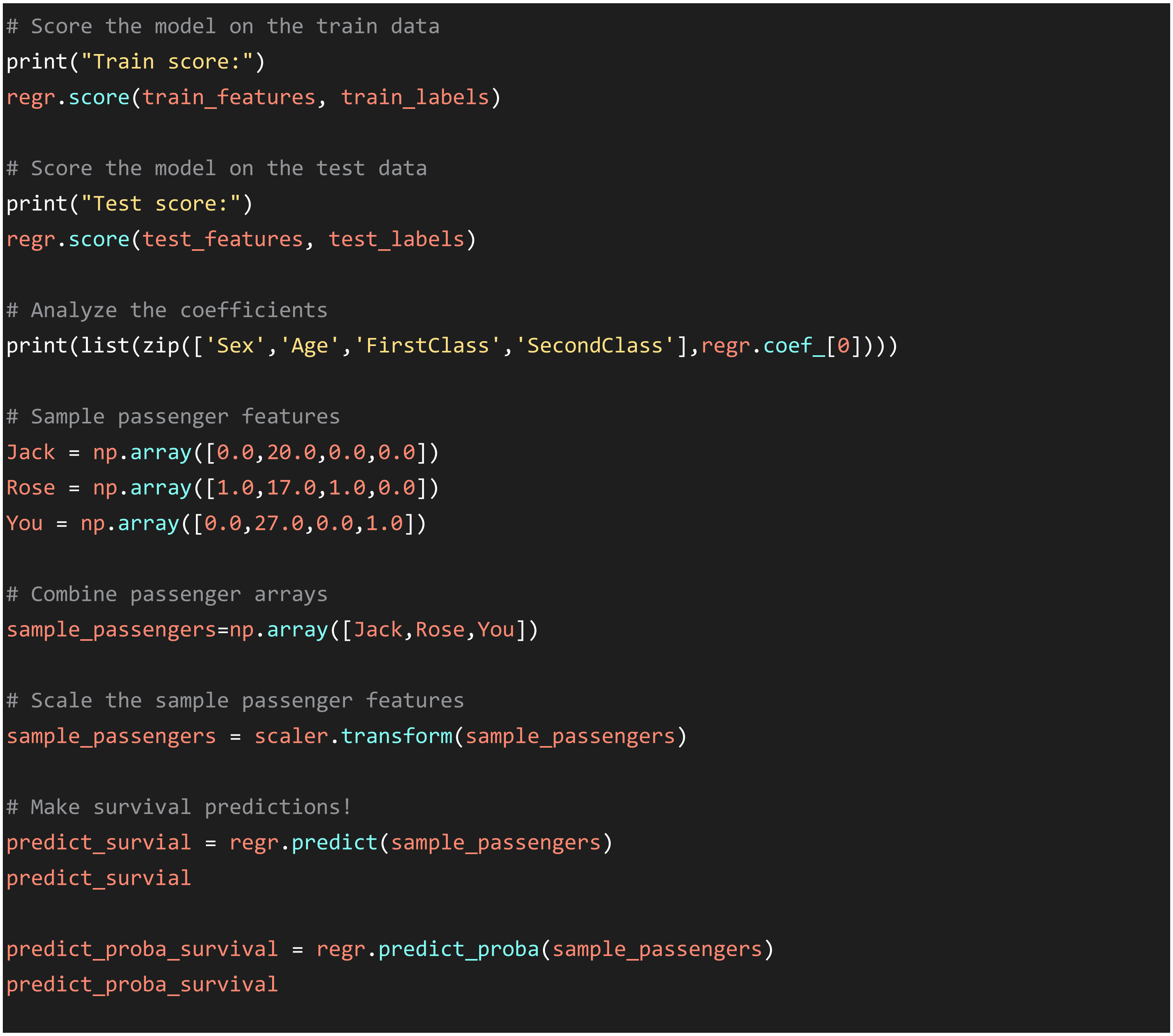
To predict the probabilities of survival for `sample_passengers`:

```
model.predict_proba(sample_passengers)
```

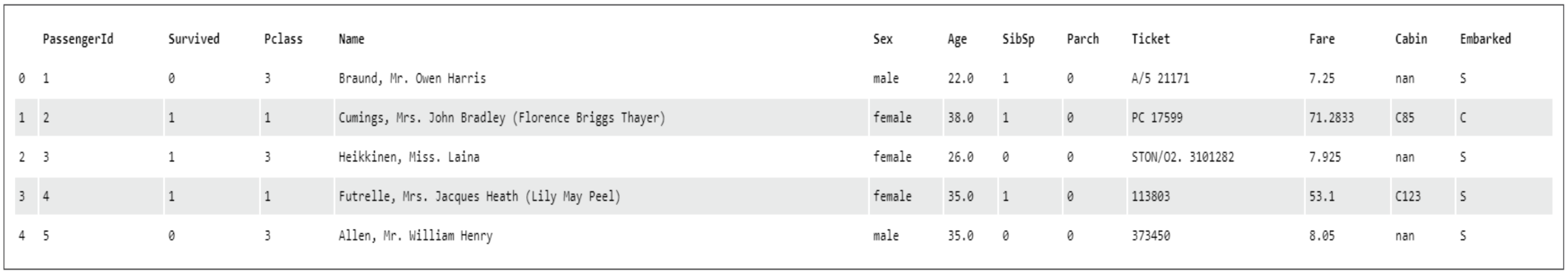
From the model's predictions, it appears `Rose` has a better fate than `Jack`. How did `You` fare?



```
import codecademylib3_seaborn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the passenger data
passengers = pd.read_csv('passengers.csv')
print(passengers.head())


# Update sex column to numerical
passengers['Sex'] = passengers. Sex.map({'male': 0, 'female': 1})

# Fill the nan values in the age column
passengers = passengers.fillna(value={'Age':passengers.Age.mean()})

# Create a first class column
passengers['FirstClass'] = passengers.Pclass.apply(lambda x: 1 if x == 1 else 0)

# Create a second class column
passengers['SecondClass'] = passengers.Pclass.apply(lambda x: 1 if x == 2 else 0)

# Select the desired features
features = passengers[['Sex', 'Age', 'FirstClass', 'SecondClass']]
survival = passengers.Survived

## Perform train, test, split
x_train, x_test, y_train, y_test = train_test_split(features, survival, train_size = 0.8, tes
t_size = 0.2,
random_state=6)

# Scale the feature data so it has mean = 0 and standard deviation = 1
scaler = StandardScaler()
train_features = scaler.fit_transform(train_features)
test_features = scaler.transform(test_features)

# Create and train the model
regr = LogisticRegression()
regr.fit(train_features, train_labels)
```



```python
# Score the model on the train data
print("Train score:")
regr.score(train_features, train_labels)

# Score the model on the test data
print("Test score:")
regr.score(test_features, test_labels)

# Analyze the coefficients
print(list(zip(['Sex','Age','FirstClass','SecondClass'],regr.coef_[0])))

# Sample passenger features
Jack = np.array([0.0,20.0,0.0,0.0])
Rose = np.array([1.0,17.0,1.0,0.0])
You = np.array([0.0,27.0,0.0,1.0])

# Combine passenger arrays
sample_passengers=np.array([Jack,Rose,You])

# Scale the sample passenger features
sample_passengers = scaler.transform(sample_passengers)

# Make survival predictions!
predict_survial = regr.predict(sample_passengers)
predict_survial

predict_proba_survival = regr.predict_proba(sample_passengers)
predict_proba_survival
```



	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	7.25	nan	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	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	7.925	nan	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05	nan	S