Logistic Regression

Objective(s):

This activity aims to solve classification problem using logistic regression

Intended Learning Outcomes (ILOs):

- Demonstrate how to train and predict classification model using logistic regression.
- Demonstrate how to evaluate the performance of the logistic regression.
- Demonstrate how to visualize the performance of the logistic regression.

Resources:

- Jupyter Notebook
- Titanic

Procedure:

Import the libraries.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Pandas. Use for data manipulation and analysis.
Handles structured data in DataFrames (like an Excel table)
Supports data cleaning, transformation, and filtering
Allows reading and writing from various file formats (CSV, Excel, SQL, JSON)

NumPy (Numerical Python). Intended for efficient numerical computing
Provides powerful N-dimensional arrays (ndarray) for mathematical operations
Optimized for numerical calculations with large datasets
Supports linear algebra, Fourier transforms, and random number generation

Matplotlib. For data visualization
Generates static, animated, and interactive plots
Supports line plots, bar charts, scatter plots, histograms, etc.

Seaborn. Statistical data visualization (built on top of Matplotlib)

Simplifies complex visualizations

Provides pre-styled themes for better aesthetics

Supports correlation heatmaps, violin plots, pair plots, etc.

Load the data using Pandas and check the content of the dataframe

```
train = pd.read_csv('titanic_train.csv')
```

```
train.head()
```

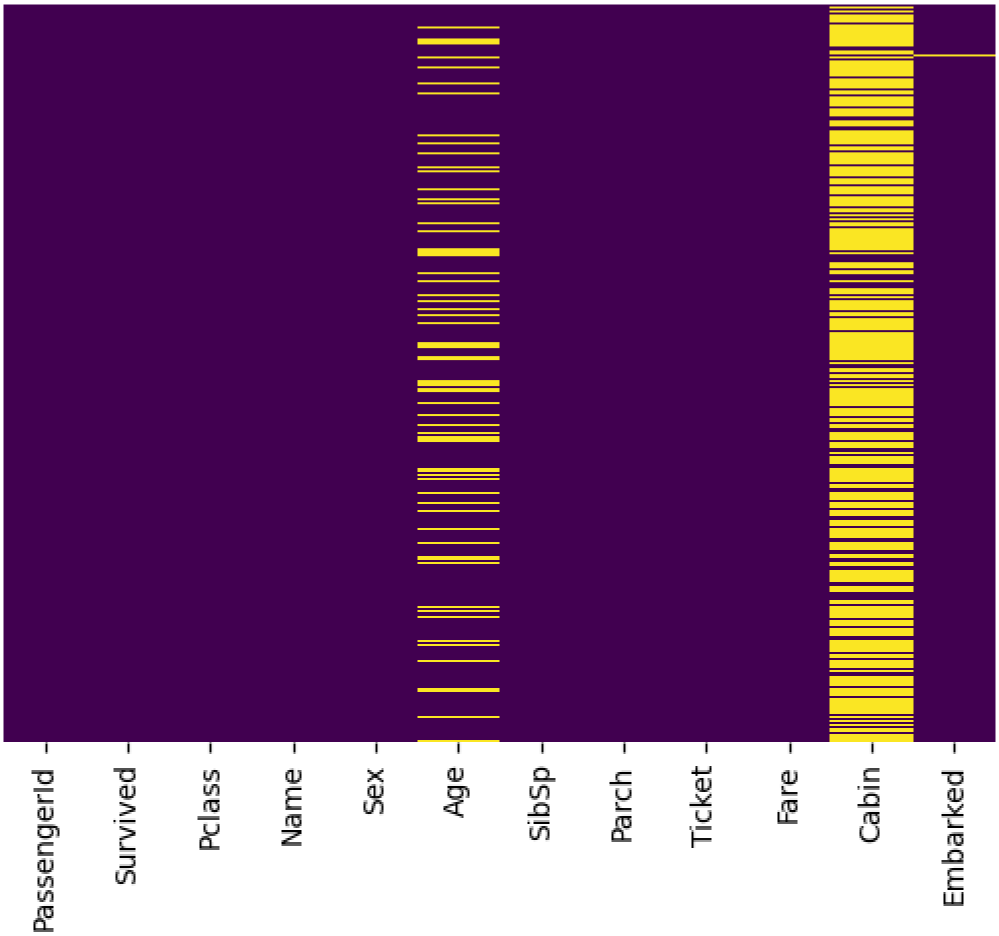
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle,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

Next steps: Generate code with `train` View recommended plots New interactive sheet

Check the missing data. Use seaborn to create a simple heatmap to see where are the missing data

```
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

```
<Axes: >
```

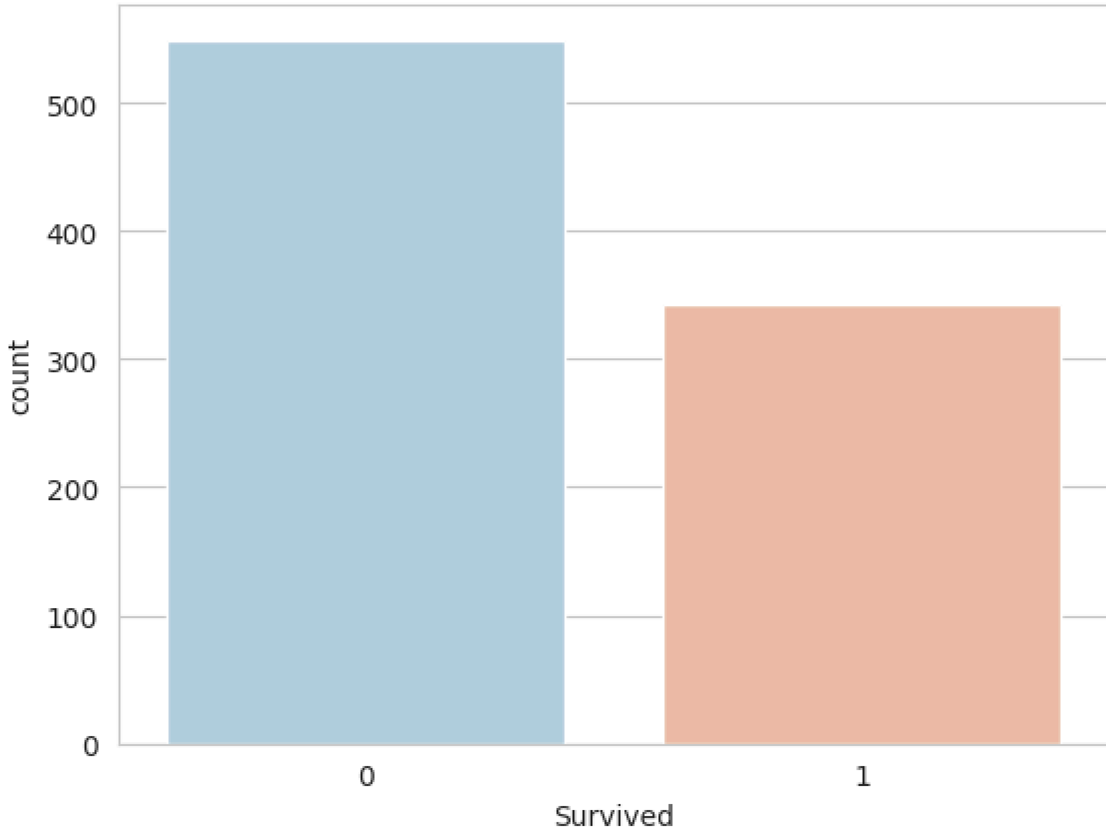


There are 20% of Age data is missing. We need to replace the missing data with some of imputation. The Cabin column are also missing too much of that data.

Use data visualization to analyze the data

```
sns.set_style('whitegrid')
sns.countplot(x='Survived',data=train,palette='RdBu_r')
```

```
<ipython-input-5-05742e5567b5>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.

  sns.countplot(x='Survived',data=train,palette='RdBu_r')
<Axes: xlabel='Survived', ylabel='count'>
```



Interpret the result of the graph

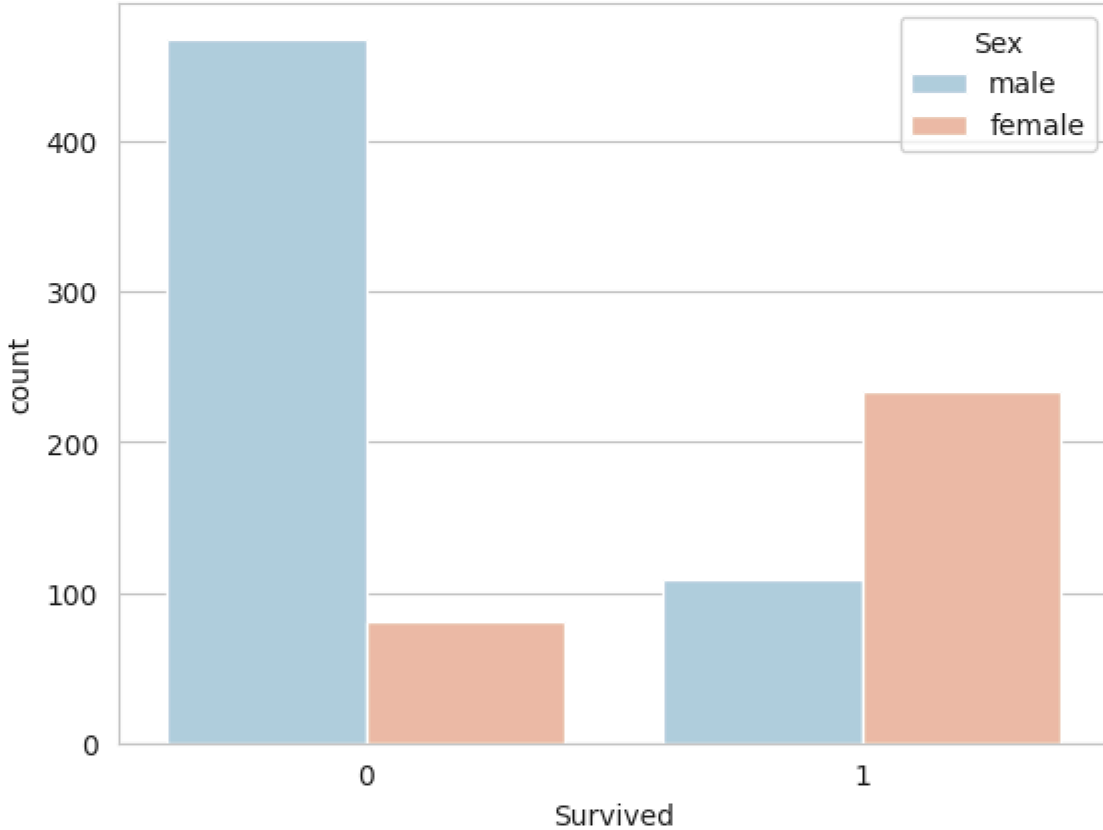
Answer

- The bar for '0' (did not survive) is significantly taller than the bar for '1' (survived). This indicates that there were more passengers who did not survive than those who did.
- The exact counts can be obtained from the y-axis.

In summary, the graph shows that a majority of the passengers in the dataset did not survive.

```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Sex',data=train,palette='RdBu_r')
```

```
<Axes: xlabel='Survived', ylabel='count'>
```



Interpret the result of the graph.

Interpretation:

Survival Disparity: The graph clearly shows a significant disparity in survival rates between males and females.

Males: A large number of males did not survive (0), while a relatively small number survived (1).

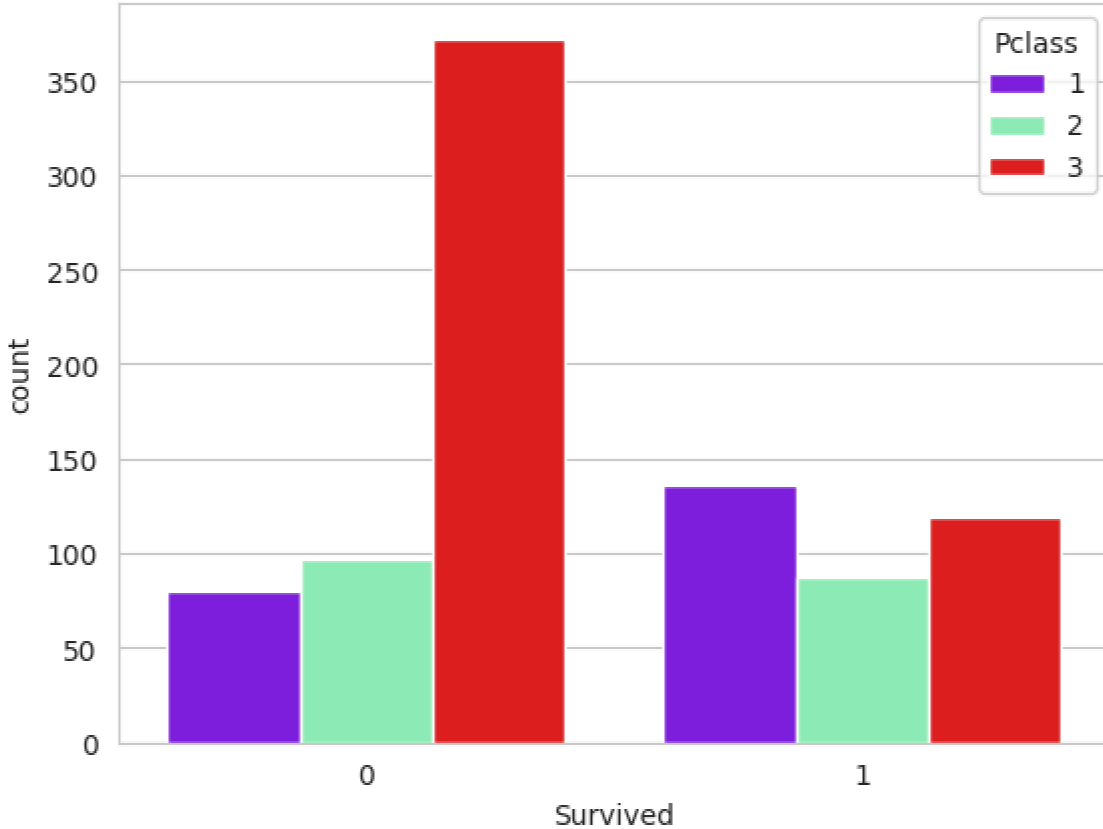
Females: The number of females who survived (1) is considerably higher than the number who did not (0). In fact, more females survived than males.

Gender and Survival: This visualization highlights the strong correlation between gender and survival, suggesting that females had a much higher chance of survival than males.

In simpler terms: More males died than females, and more females survived than males. This indicates that gender played a significant role in determining survival in this dataset.

```
sns.set_style('whitegrid')
sns.countplot(x='Survived',hue='Pclass',data=train,palette='rainbow')
```

```
<Axes: xlabel='Survived', ylabel='count'>
```



Interpret the result.

Interpretation:

Class 3 Disparity: The most striking observation is the significantly higher number of passengers from class 3 who did not survive (0). This class also has the highest overall count of passengers.

Class 1 Survival: Passengers from class 1 had the highest survival rate (1) compared to the other classes.

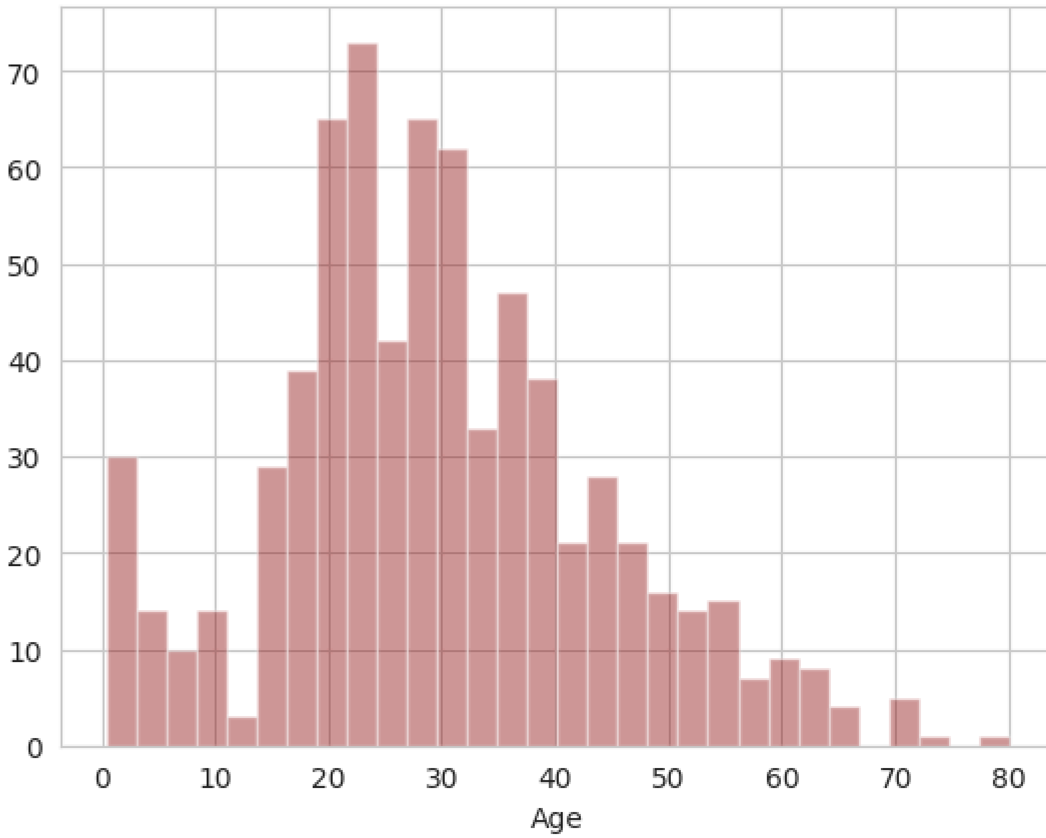
Class 2 Survival: Class 2 passengers also had a higher survival rate than those in class 3, though not as high as class 1.

Pclass and Survival: This visualization clearly shows a strong correlation between passenger class and survival. Passengers in higher classes (1 and 2) had a significantly better chance of survival than those in class 3.

Passengers in the lower class (3) were much more likely to die, while those in the upper classes (1 and 2) had a higher chance of survival. This indicates that passenger class played a significant role in determining survival in this dataset.

```
sns.distplot(train['Age'].dropna(),kde=False,color='darkred',bins=30)
```

```
&lt;ipython-input-8-53c281d34688&gt;:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(train['Age'].dropna(),kde=False,color='darkred',bins=30)
<Axes: xlabel='Age'>
```



Interpret the result of the graph.

Interpretation:

Age Distribution: The graph shows the distribution of ages is skewed to the right, indicating a higher concentration of younger individuals in the dataset.

Peak: There is a noticeable peak in the distribution around the 20-30 age range, suggesting this is the most common age group. Decreasing Frequency: The frequency of individuals decreases as age increases, with relatively few individuals in the older age ranges (60-80).

Missing Values: The .dropna() method was used, meaning any missing values in the 'Age' column were removed before plotting. Therefore, the histogram represents the distribution of available age data.

The dataset contains mostly younger individuals, with a peak in the 20-30 age range. There are fewer older individuals in the dataset.

```
!pip install cufflinks
```

```
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: ipykernel>=4.5.1 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: ipython-genutils~=0.2.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.11
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.11
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/di
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: debugpy>=1.0 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python3.11/di
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: pyzmq>=17 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: tornado>=6.1 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: parso<0.9.0,>=0.8.4 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: wcwidth in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: notebook>=4.4.1 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: jupyter-core>=4.6.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: nbformat in /usr/local/lib/python3.11/dist-packages (f
Requirement already satisfied: nbconvert>=5 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: nbclassic>=0.4.7 in /usr/local/lib/python3.11/dist-pac
```

```
Requirement already satisfied: jupyter-server<3,>=1.8 in /usr/local/lib/python3.11/di
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.11/
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: anyio>=3.1.0 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: websocket-client in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-package
```

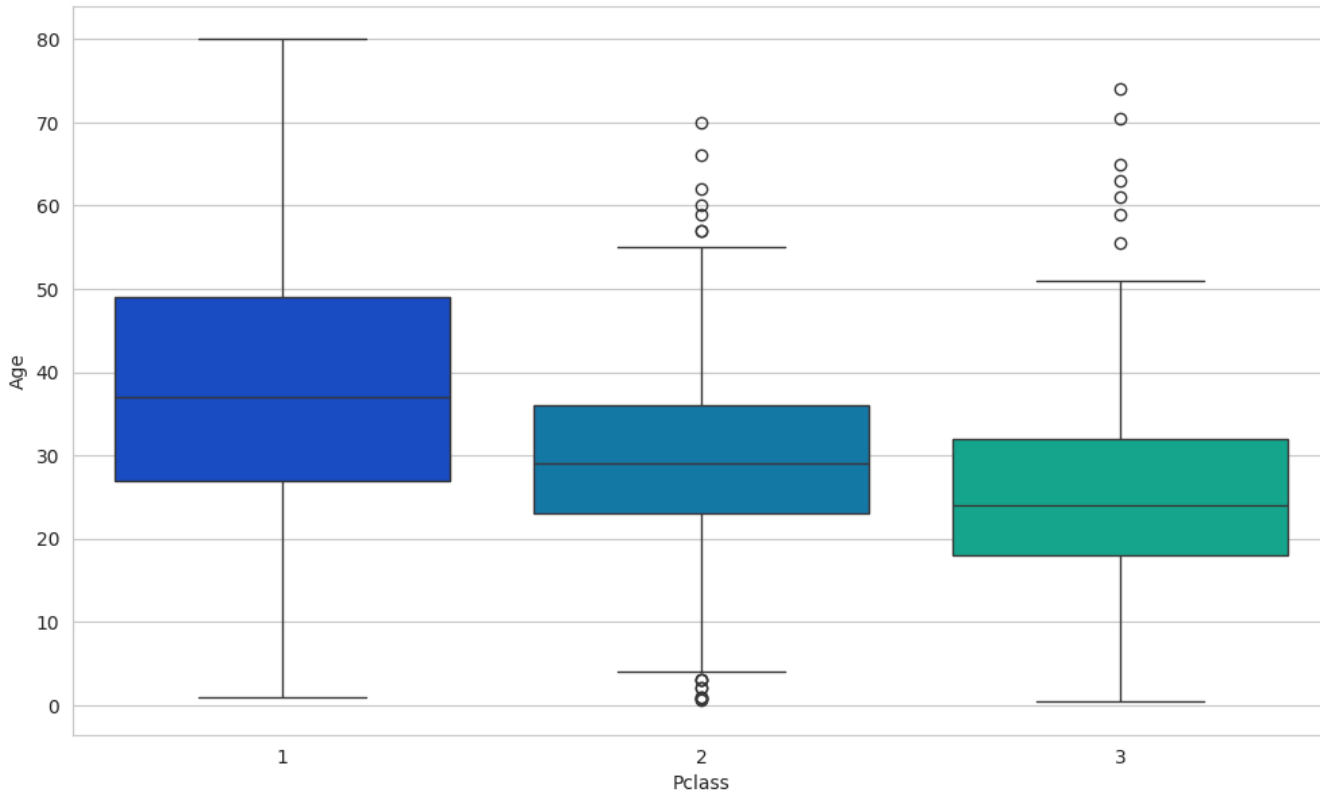
```
import cufflinks as cf
cf.go_offline()
```

```
import plotly.express as px
import pandas as pd

# Assuming you have a pandas DataFrame called 'train'
# with a column named 'Fare'

fig = px.histogram(train, x='Fare', color_discrete_sequence=['green'])  # Using named CSS cc
fig.show()
```

Replace the missing data of the Age column. One way to do this is by filling in the mean age of all the passengers (imputation).

```
plt.figure(figsize=(12, 7))
sns.boxplot(x='Pclass',y='Age',data=train,palette='winter')
```

```
<ipython-input-14-551bc5ec5847>:2: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.


<Axes: xlabel='Pclass', ylabel='Age'>
```

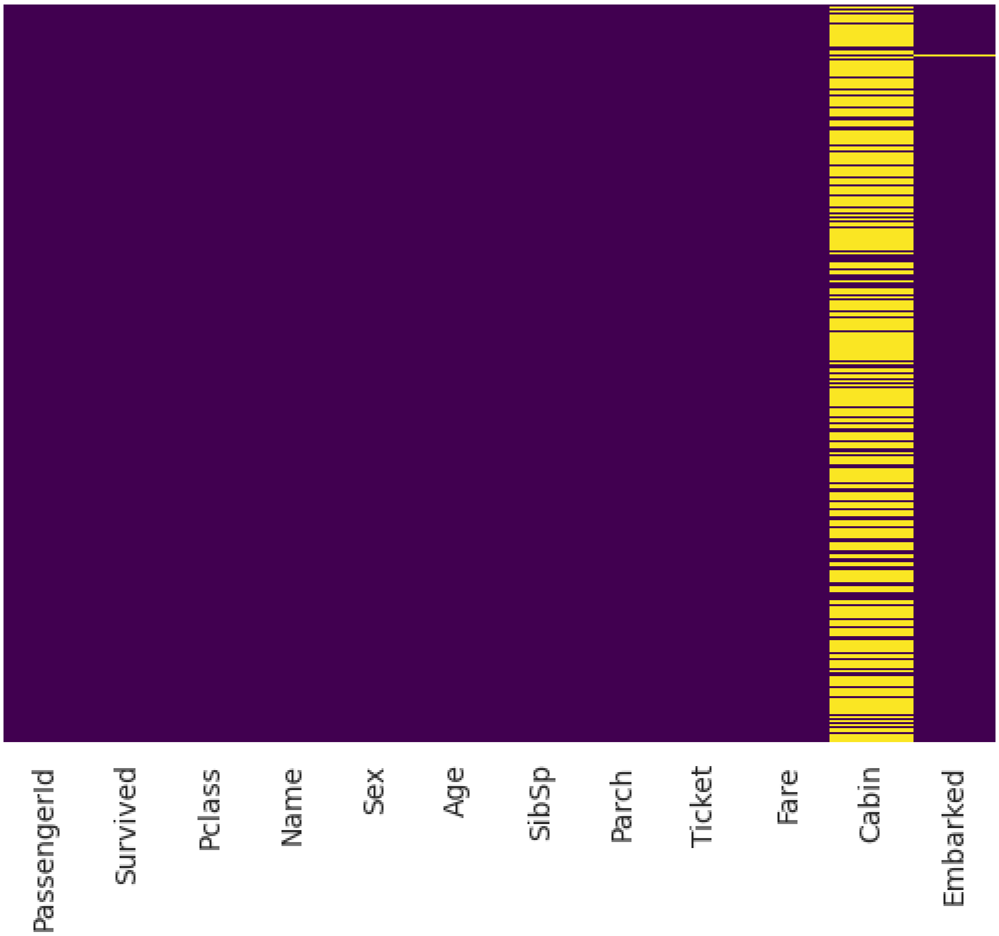


```
#create a function to replace the missing data
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]

    if pd.isnull(Age):

        if Pclass == 1:
            return 37

        elif Pclass == 2:
            return 29

        else:
            return 24

    else:
        return Age


#apply the function
train['Age'] = train[['Age','Pclass']].apply(impute_age,axis=1)
```

```
<ipython-input-15-fd2656a82d91>:3: FutureWarning:

Series.__getitem__ treating keys as positions is deprecated. In a future version, intege

<ipython-input-15-fd2656a82d91>:4: FutureWarning:

Series.__getitem__ treating keys as positions is deprecated. In a future version, intege
```

```
#check the missing data
sns.heatmap(train.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

<Axes: >



Drop the Cabin column and the row in Embarked that is NaN.

```
train.drop('Cabin',axis=1,inplace=True)
```

```
train.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 Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

Next steps: Generate code with `train` View recommended plots New interactive sheet

```
#drop NaN
train.dropna(inplace=True)
```

Convert categorical features to dummy variables using pandas

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 889 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  889 non-null    int64  
 1   Survived     889 non-null    int64  
 2   Pclass       889 non-null    int64  
 3   Name         889 non-null    object 
 4   Sex          889 non-null    object 
 5   Age          889 non-null    float64
 6   SibSp        889 non-null    int64  
 7   Parch        889 non-null    int64  
 8   Ticket       889 non-null    object 
```

```
 9   Fare         889 non-null    float64
 10  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 83.3+ KB
```

```
sex = pd.get_dummies(train['Sex'],drop_first=True)
embark = pd.get_dummies(train['Embarked'],drop_first=True)
```

```
train.drop(['Sex','Embarked','Name','Ticket'],axis=1,inplace=True)
train = pd.concat([train,sex,embark],axis=1)
train.head()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare	male	Q	S
0	1	0	3	22.0	1	0	7.2500	True	False	True
1	2	1	1	38.0	1	0	71.2833	False	False	False
2	3	1	3	26.0	0	0	7.9250	False	False	True
3	4	1	1	35.0	1	0	53.1000	False	False	True
4	5	0	3	35.0	0	0	8.0500	True	False	True

Next steps: Generate code with train | View recommended plots | New interactive sheet

Build the Logistic Regression Model

Split the data into a training set and test set

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
```

```
X_train, X_test, y_train, y_test = train_test_split(train.drop('Survived',axis=1),
                                                    train['Survived'], test_size=0.30,
                                                    random_state=101)
```

Train the model

```
from sklearn.linear_model import LogisticRegression
```

```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: Converger

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
```

```
▾ LogisticRegression
LogisticRegression()
```

Predict the values for the testing data

```
predictions = logmodel.predict(X_test)
```

Check precision,recall,f1-score using classification report

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.79      0.91      0.85       163
           1       0.81      0.62      0.71       104

    accuracy                           0.80       267
   macro avg       0.80      0.77      0.78       267
weighted avg       0.80      0.80      0.79       267
```

Interpret the precision, recall f1-score

Interpretation

Class 0:

Precision: 0.79 (When the model predicted class 0, it was correct 79% of the time)

Recall: 0.91 (The model correctly identified 91% of all actual class 0 instances)

F1-score: 0.85

Support: 163 (There were 163 instances of class 0 in the test set)

Class 1:

Precision: 0.81 (When the model predicted class 1, it was correct 81% of the time)

Recall: 0.62 (The model correctly identified 62% of all actual class 1 instances)

F1-score: 0.71

Support: 104 (There were 104 instances of class 1 in the test set)

Overall Metrics:

Accuracy: 0.80 (The model was correct 80% of the time overall)

Macro Avg: The average of precision, recall, and f1-score without considering class imbalance.

Precision: 0.80

Recall: 0.77

F1-score: 0.78

Weighted Avg: The average of precision, recall, and f1-score weighted by the number of instances in each class.

Precision: 0.80

Recall: 0.80

F1-score: 0.79

Evaluate the accuracy and confusion matrix of the model

```
from sklearn.metrics import confusion_matrix, accuracy_score, roc_auc_score
```

```
print(accuracy_score(y_test, predictions))
```

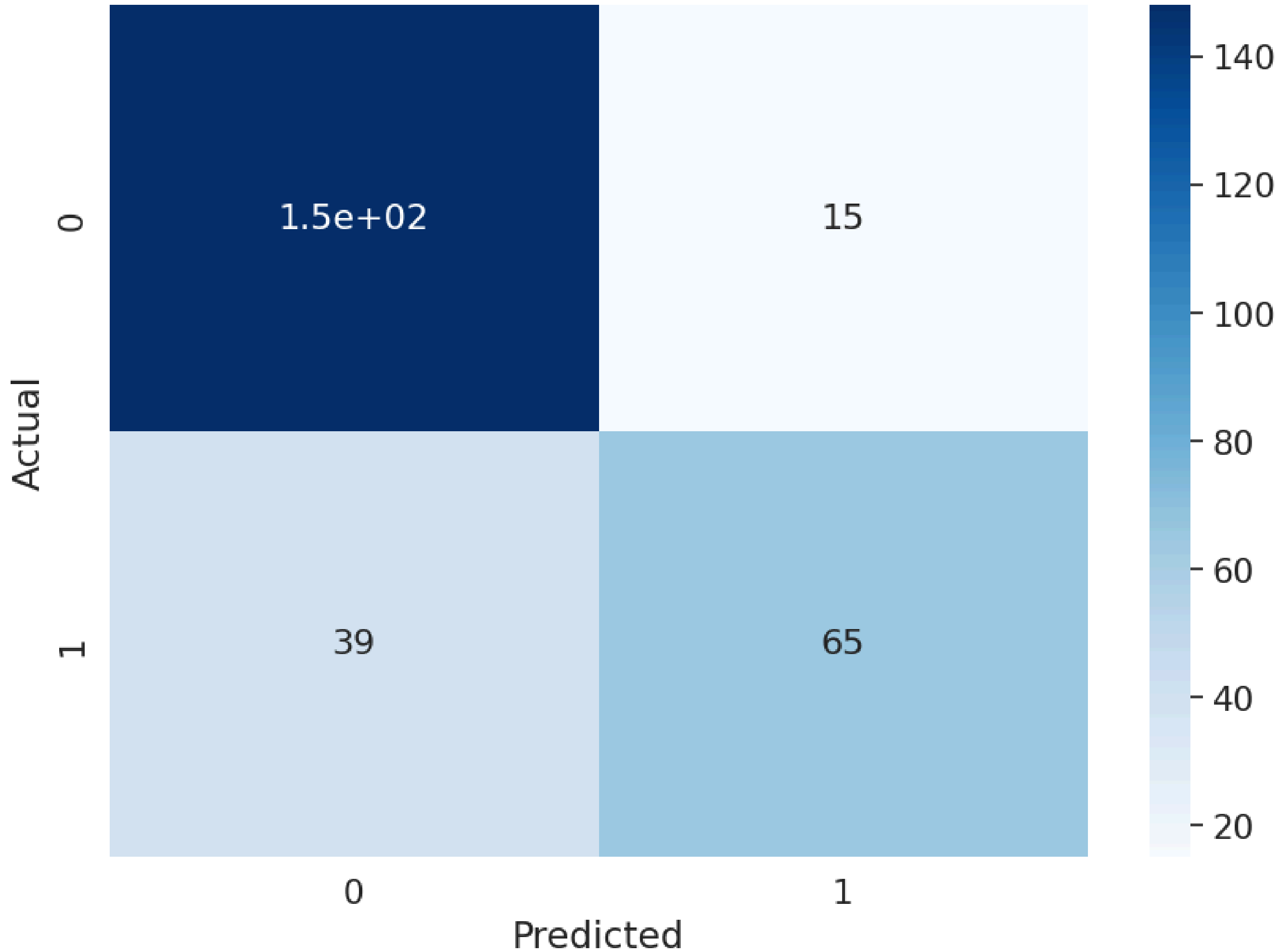
```
0.797752808988764
```

```
confusion_matrix(y_test, predictions)
```

```
array([[148,  15],
       [ 39,  65]])
```

```
data = confusion_matrix(y_test, predictions)
df_cm = pd.DataFrame(data, columns=np.unique(y_test), index = np.unique(predictions))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (10,7))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16})
```

```
<Axes: xlabel='Predicted', ylabel='Actual'>
```



Reflection

Logistic regression emerges as a powerful tool in the realm of binary classification, elegantly predicting the likelihood of an event's occurrence. In the context of survival analysis, it adeptly calculates the probability of a passenger either surviving or not surviving. The sigmoid function plays a pivotal role, transforming linear combinations of features into probabilities bounded between 0 and 1. This probabilistic output is then translated into a binary prediction based on a predetermined threshold. The model's capacity to assign weights to different features illuminates their relative importance in influencing survival outcomes. A higher weight associated with a particular feature signifies a stronger correlation with survival probability.

The analysis of the provided data underscores the significance of certain features in determining survival chances. Gender, as vividly illustrated in the countplot, reveals a striking disparity in survival rates between males and females, with females exhibiting a considerably higher likelihood of survival. Similarly, passenger class emerges as a crucial determinant, with higher-class passengers enjoying a greater probability of survival compared to their lower-class counterparts. While the age distribution histogram provides insights into the dataset's composition, further modeling would be required to precisely quantify age's impact on survival.

In essence, Logistic Regression, by estimating probabilities and weighting features, offers a robust framework for classifying survival, and the analysis of features like gender and passenger class provides valuable insights into the factors influencing survival outcomes.

Supplementary Activity:

- Choose your own dataset
- Import the dataset
- Determine the number of datapoints, columns and data types
- Remove unneccesary columns
- Do data cleaning such as removing empty values(NaN), replacing missing data .
- Perform descriptive statistics such as mean, median and mode
- Perform data visualization
- Solve classification problem using Logistic Regression
- Evaluate the model using classification report, accuracy and confusion matrix

Dataset

WHI_Inflation.csv

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

# 1. Load the Dataset with Tab Delimiter
try:
    df = pd.read_csv("WHI_Inflation.csv", sep='\t')  # Corrected loading with tab delimiter
    print("Dataset loaded successfully!")
except FileNotFoundError:
    print("Error: File 'WHI_Inflation.csv' not found. Please check the file path.")
    exit()

print(df.head()) # Add this line to see the first few rows
print(df.columns) # Add this line to see the column names
```

```
Dataset loaded successfully!
       Country  Year  Rank  Score  GDP per Capita  Social support  \
0  Afghanistan  2015   153  3.575        0.319820        0.302850   
1  Afghanistan  2016   154  3.360        0.382270        0.110370   
2  Afghanistan  2017   141  3.794        0.401477        0.581543   
3  Afghanistan  2018   145  3.632        0.332000        0.537000   
4  Afghanistan  2019   154  3.203        0.350000        0.517000   

   Healthy life expectancy at birth  Freedom to make life choices  Generosity  \
0                          0.303350                       0.23414    0.365100   
1                          0.173440                       0.16430    0.312680   
2                          0.180747                       0.10618    0.311871   
3                          0.255000                       0.08500    0.191000   
4                          0.361000                       0.00000    0.158000   

   Perceptions of corruption  Energy Consumer Price Inflation  \
0                   0.097190                        -4.250000   
1                   0.071120                         2.070000   
2                   0.061158                         4.440000   
3                   0.036000                         1.474185   
4                   0.025000                        -2.494359   

   Food Consumer Price Inflation  GDP deflator Index growth rate  \
0                      -0.840000                        2.665090   
1                       5.670000                       -2.409509   
2                       6.940000                        2.404000   
3                      -1.045952                        2.071208   
4                       3.794770                        6.520928   

   Headline Consumer Price Inflation  Official Core Consumer Price Inflation  \
0                             -0.660                                0.219999   
1                              4.380                                5.192760   
2                              4.976                                5.423228   
3                              0.630                               -0.126033   
4                              2.302                                     NaN   

   Producer Price Inflation Continent  
0                       NaN      Asia  
1                       NaN      Asia  
2                       NaN      Asia  
3                       NaN      Asia  
4                       NaN      Asia  
Index(['Country', 'Year', 'Rank', 'Score', 'GDP per Capita', 'Social support',
       'Healthy life expectancy at birth', 'Freedom to make life choices',
       'Generosity', 'Perceptions of corruption',
       'Energy Consumer Price Inflation', 'Food Consumer Price Inflation',
       'GDP deflator Index growth rate', 'Headline Consumer Price Inflation',
       'Official Core Consumer Price Inflation', 'Producer Price Inflation',
       'Continent'],
      dtype='object')
```

```
import pandas as pd
import numpy as np
```

```
# Load the dataset (you've already done this successfully)
df = pd.read_csv("WHI_Inflation.csv", sep='\t')

# Explore the dataset
print("\n--- Dataset Info ---")
print(f"Number of rows: {df.shape[0]}")
print(f"Number of columns: {df.shape[1]}")
print("\nData types:\n", df.dtypes)

print("\nBasic statistics:\n", df.describe())
```

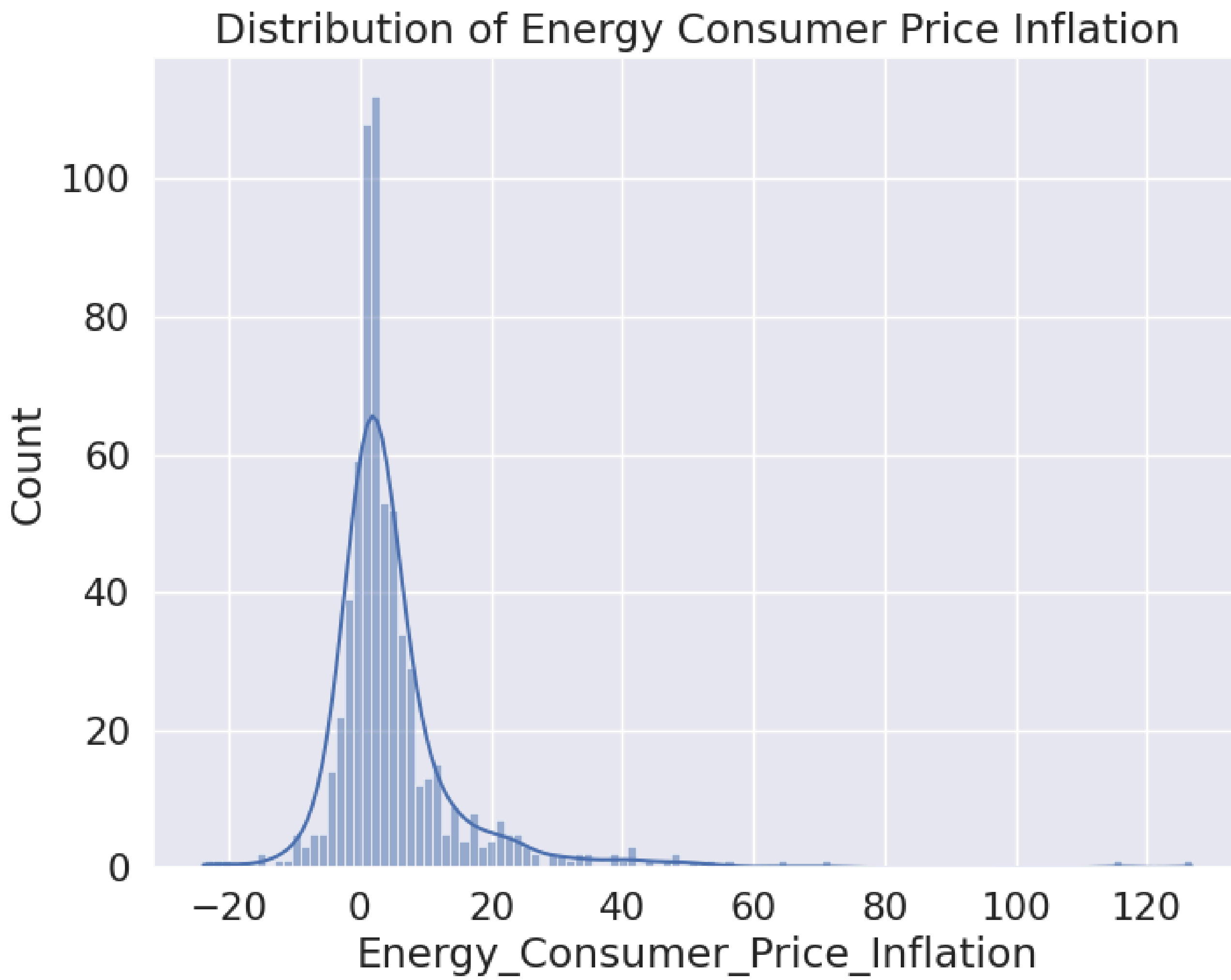
```
Continent                                    object
dtype: object

Basic statistics:
               Year         Rank        Score  GDP per Capita  Social support  \
count  1203.000000  1203.000000  1203.000000     1203.000000     1203.000000   
mean   2018.868662    73.975062     5.503177        2.797294        0.983193   
std       2.551181    44.776420     1.138402        3.547966        0.302705   
min    2015.000000     1.000000     1.859000        0.000000        0.000000   
25%    2017.000000    35.000000     4.624300        0.800500        0.799505   
50%    2019.000000    72.000000     5.546000        1.164920        0.934000   
75%    2021.000000   113.000000     6.346150        1.704000        1.214504   
max    2023.000000   158.000000     7.842000       11.660000        1.644000   

       Healthy life expectancy at birth  Freedom to make life choices  \
count                       1203.000000                   1203.000000   
mean                          21.676503                      0.553649   
std                           30.421780                      0.221108   
min                            0.000000                      0.000000   
25%                            0.582975                      0.405480   
50%                            0.792566                      0.546040   
75%                           59.512076                      0.729000   
```

```
count          1187.000000              1180.000000
mean              6.942309                 7.392601
std              31.771016                25.370648
min             -26.100000                -3.752996
25%               1.353737                 1.392791
50%               3.176209                 3.445729
75%               6.996021                 6.751251
max             812.247463               557.210000

       Official Core Consumer Price Inflation  Producer Price Inflation
count                              720.000000                757.000000
mean                                 3.517293                  5.691213
std                                  5.535930                 13.384842
```

```
# Remove Unnecessary Columns (Adjust as needed)
columns_to_remove = ['Rank', 'Producer Price Inflation']  # Add or remove columns based on y
df = df.drop(columns_to_remove, axis=1, errors='ignore')

# Handle Missing Values
print("\nMissing values per column:\n", df.isnull().sum())
df = df.dropna()  # Remove rows with any missing values (adjust as needed)

# Rename Columns (Example: Remove spaces or special characters)
df.columns = df.columns.str.replace(' ', '_')  # Replace spaces with underscores
df.columns = df.columns.str.replace('[^A-Za-z0-9_]+', '', regex=True) # Remove special chara

print(df.columns) # Print columns to verify changes
```

```
Missing values per column:
 Country                                  0
Year                                     0
Score                                    0
GDP per Capita                           0
Social support                           0
Healthy life expectancy at birth         0
Freedom to make life choices             0
Generosity                               0
Perceptions of corruption                1
Energy Consumer Price Inflation        129
Food Consumer Price Inflation           89
GDP deflator Index growth rate          16
Headline Consumer Price Inflation       23
Official Core Consumer Price Inflation 483
Continent                               44
dtype: int64
Index(['Country', 'Year', 'Score', 'GDP_per_Capita', 'Social_support',
       'Healthy_life_expectancy_at_birth', 'Freedom_to_make_life_choices',
       'Generosity', 'Perceptions_of_corruption',
       'Energy_Consumer_Price_Inflation', 'Food_Consumer_Price_Inflation',
       'GDP_deflator_Index_growth_rate', 'Headline_Consumer_Price_Inflation',
       'Official_Core_Consumer_Price_Inflation', 'Continent'],
      dtype='object')
```

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

# Example: Histogram for Inflation Rate
plt.figure(figsize=(8, 6))
sns.histplot(df['Energy_Consumer_Price_Inflation'], kde=True)
plt.title('Distribution of Energy Consumer Price Inflation')
plt.show()

# Example: Scatter Plot of Happiness Score vs. GDP
plt.figure(figsize=(8, 6))
sns.scatterplot(x='GDP_per_Capita', y='Score', data=df)
plt.title('Happiness Score vs. GDP')
plt.show()
```


Distribution of Energy Consumer Price Inflation
Count
Energy_Consumer_Price_Inflation
100
80
60
40
20
0
−20
0
20
40
60
80
100
120


Happiness Score vs. GDP
Score
8
7
6
5
4


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

# 1. Load the Dataset with Tab Delimiter
try:
    df = pd.read_csv("WHI_Inflation.csv", sep='\t')
    print("Dataset loaded successfully!")
except FileNotFoundError:
    print("Error: File 'WHI_Inflation.csv' not found. Please check the file path.")
    exit()
```

```
Dataset loaded successfully!
```

```
# 2. Explore the Dataset
print("\n--- Dataset Info ---")
print(f"Number of rows: {df.shape[0]}")
print(f"Number of columns: {df.shape[1]}")
print("\nData types:\n", df.dtypes)
```

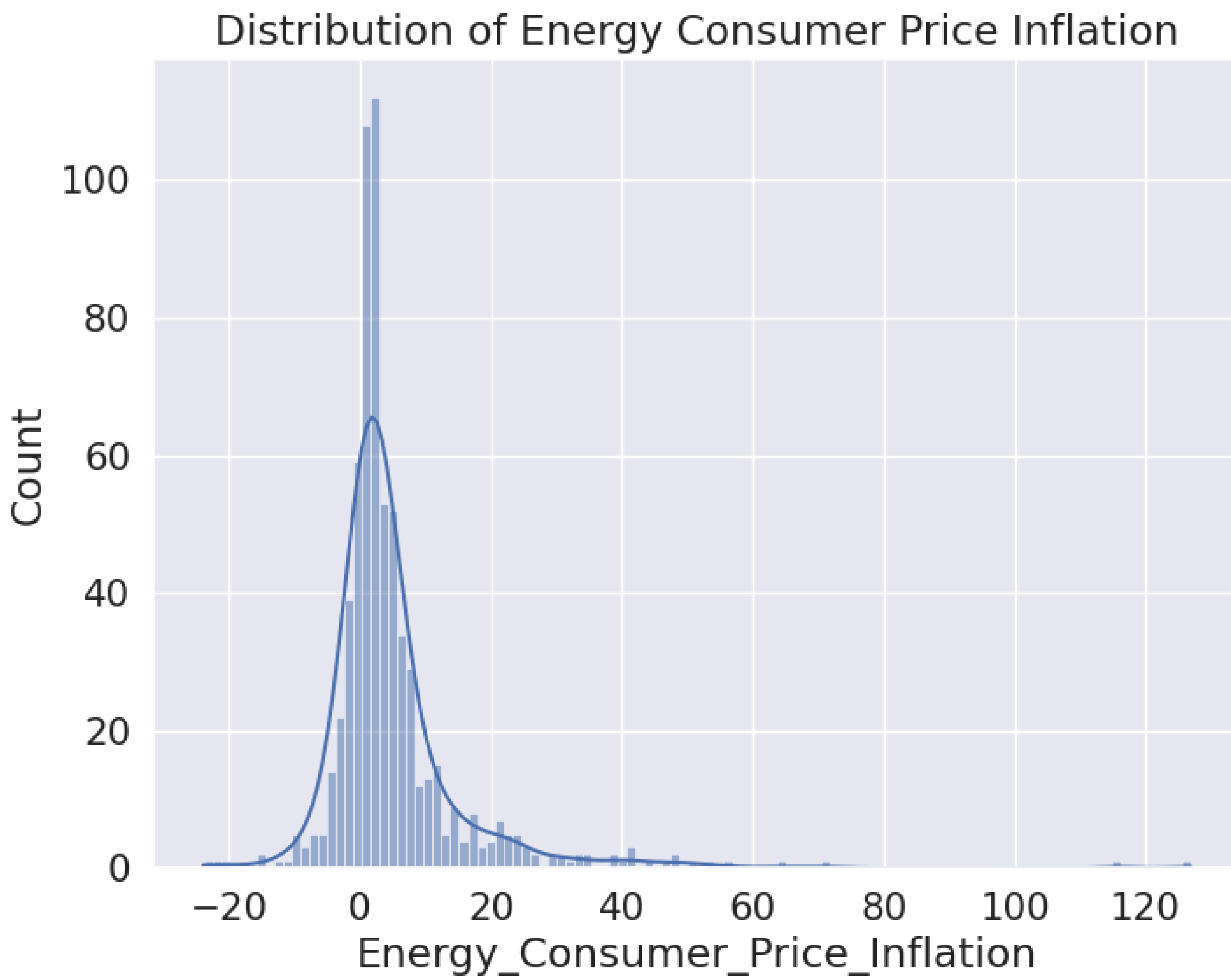
```
--- Dataset Info ---
Number of rows: 1203
Number of columns: 17

Data types:
 Country                                    object
Year                                         int64
Rank                                         int64
Score                                      float64
GDP per Capita                             float64
Social support                             float64
Healthy life expectancy at birth           float64
Freedom to make life choices               float64
```

```
Generosity                                float64
Perceptions of corruption                 float64
Energy Consumer Price Inflation           float64
Food Consumer Price Inflation             float64
GDP deflator Index growth rate            float64
Headline Consumer Price Inflation         float64
Official Core Consumer Price Inflation    float64
Producer Price Inflation                  float64
Continent                                  object
dtype: object
```

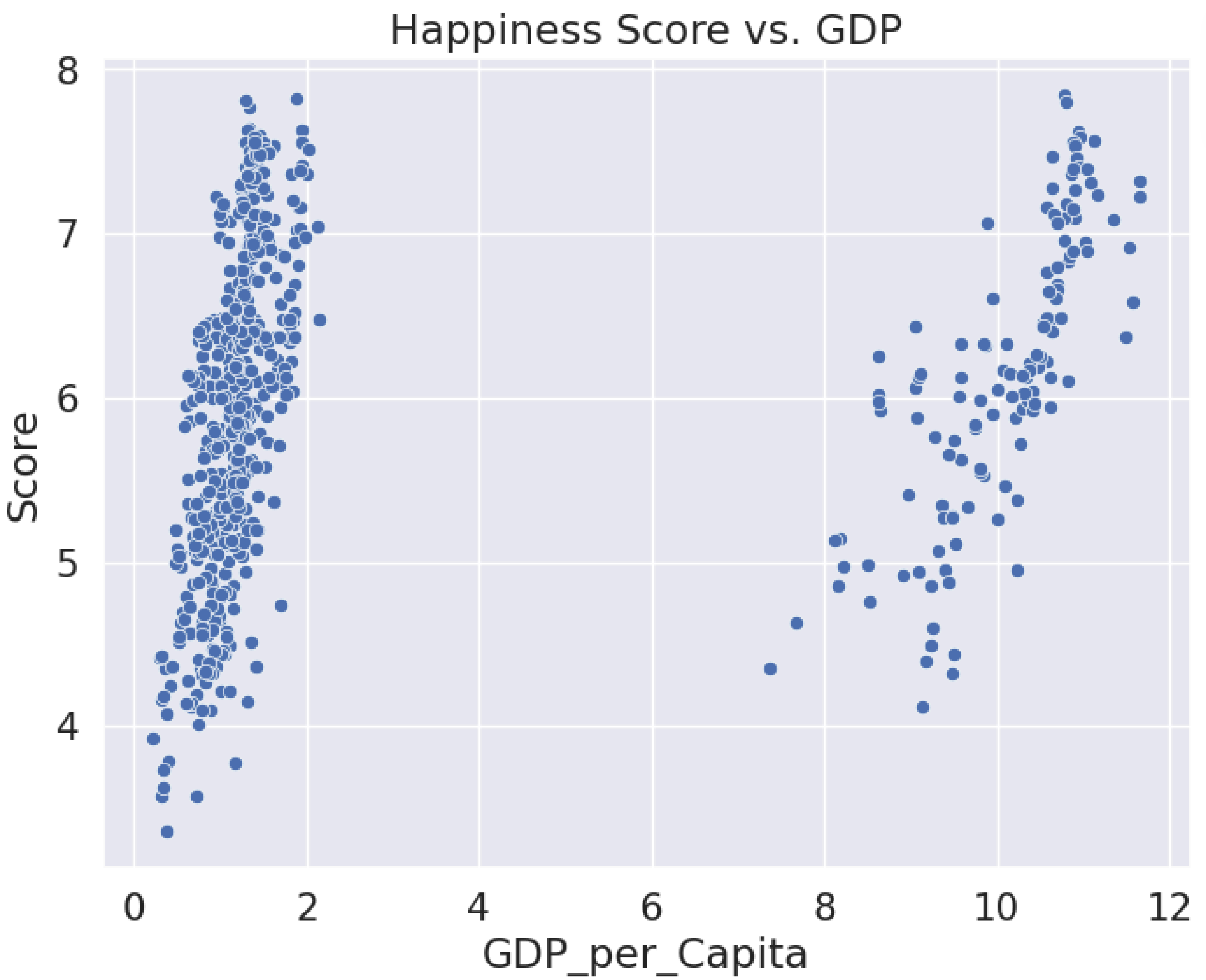
```
# 3. Preprocessing

# Remove Unnecessary Columns (Adjust as needed)
columns_to_remove = ['Rank', 'Producer Price Inflation']
df = df.drop(columns_to_remove, axis=1, errors='ignore')

# Handle Missing Values
print("\nMissing values per column:\n", df.isnull().sum())
df = df.dropna()

# Rename Columns (Example: Remove spaces or special characters)
df.columns = df.columns.str.replace(' ', '_')
df.columns = df.columns.str.replace('[^A-Za-z0-9_]+', '', regex=True)

print(df.columns)
```

```
Missing values per column:
 Country                                     0
Year                                        0
Score                                       0
GDP per Capita                              0
Social support                              0
Healthy life expectancy at birth            0
Freedom to make life choices                0
Generosity                                  0
Perceptions of corruption                   1
Energy Consumer Price Inflation           129
Food Consumer Price Inflation              89
GDP deflator Index growth rate             16
Headline Consumer Price Inflation          23
Official Core Consumer Price Inflation    483
Continent                                  44
dtype: int64
Index(['Country', 'Year', 'Score', 'GDP_per_Capita', 'Social_support',
       'Healthy_life_expectancy_at_birth', 'Freedom_to_make_life_choices',
       'Generosity', 'Perceptions_of_corruption',
       'Energy_Consumer_Price_Inflation', 'Food_Consumer_Price_Inflation',
       'GDP_deflator_Index_growth_rate', 'Headline_Consumer_Price_Inflation',
       'Official_Core_Consumer_Price_Inflation', 'Continent'],
      dtype='object')
```

```
# 4. Data Visualization

#Histogram for Inflation Rate
plt.figure(figsize=(8, 6))
sns.histplot(df['Energy_Consumer_Price_Inflation'], kde=True)
plt.title('Distribution of Energy Consumer Price Inflation')
plt.show()
```




```
#Scatter Plot of Happiness Score vs. GDP
plt.figure(figsize=(8, 6))
sns.scatterplot(x='GDP_per_Capita', y='Score', data=df)
plt.title('Happiness Score vs. GDP')
plt.show()
```




```
# 5. Solve Classification Problem (Logistic Regression)

# Create Happiness Score Ranges (Example)
bins = [0, 4, 6, 10]
labels = ['Low', 'Medium', 'High']
df['Happiness_Range'] = pd.cut(df['Score'], bins=bins, labels=labels, right=False)
```

```
# Prepare Data
X = df.drop(['Happiness_Range', 'Score'], axis=1, errors='ignore')
y = df['Happiness_Range']

# Remove the 'Continent' column (or use one-hot encoding as explained before)
X = X.drop('Continent', axis=1, errors='ignore')

# Encode Target Variable (if needed)
```

```
    if y.dtype == 'object':
        le = LabelEncoder()
        y = le.fit_transform(y)



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

    print(X_train.info()) # Inspect X_train to find the problematic column
```