Overview

Implementing the logistic regression unit, a single-neuron neural network.

Write an alternative activation function, i.e., replace the sigmoid non-linearity with ReLu or tan, and then use Log Loss Function and code to do stochastic gradient descent (SGD) for optimization.

Implementing as a set of functions is ideal.

Using any Titanic dataset for a classification task.

Providing the results' accuracy and F1 score is the last step.

Importing packages required

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Loading the dataset

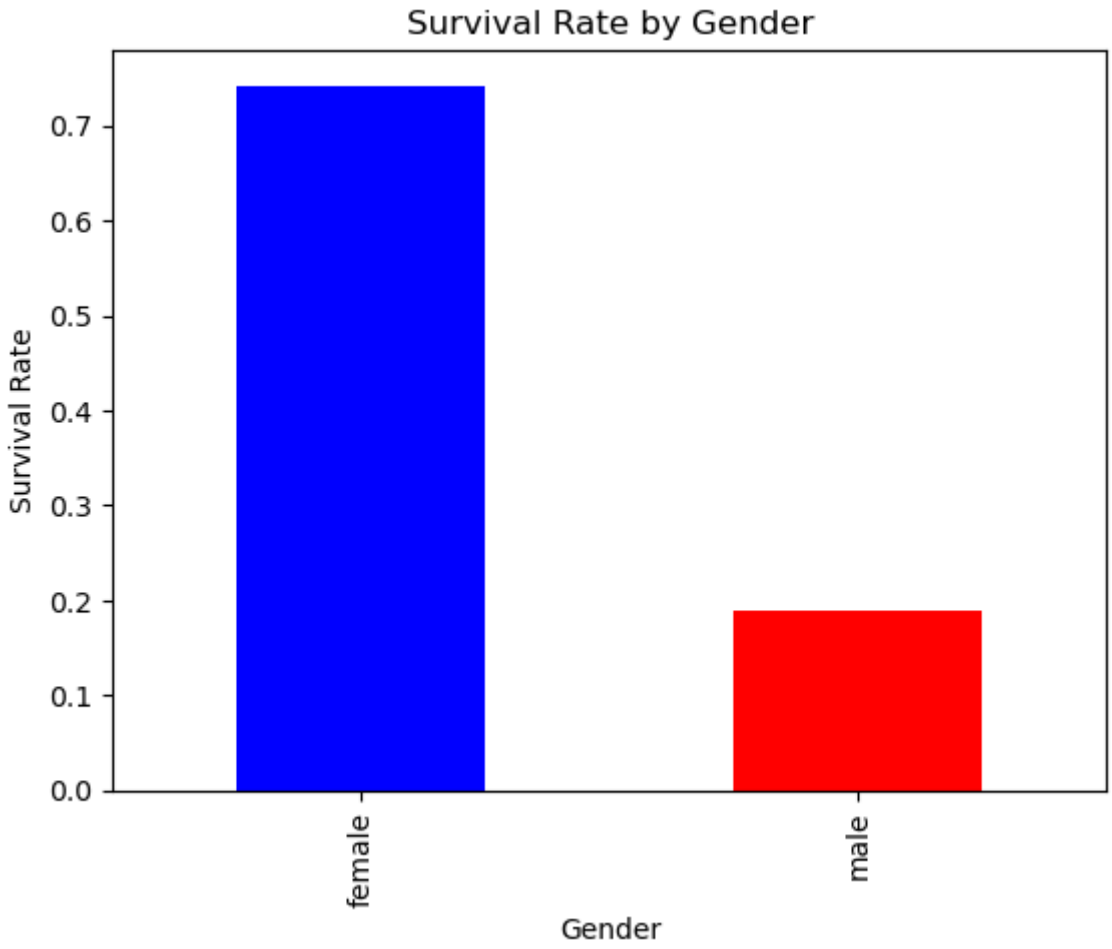
In [2]:
```
df = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/ma

df.head()
```

Out[2]:

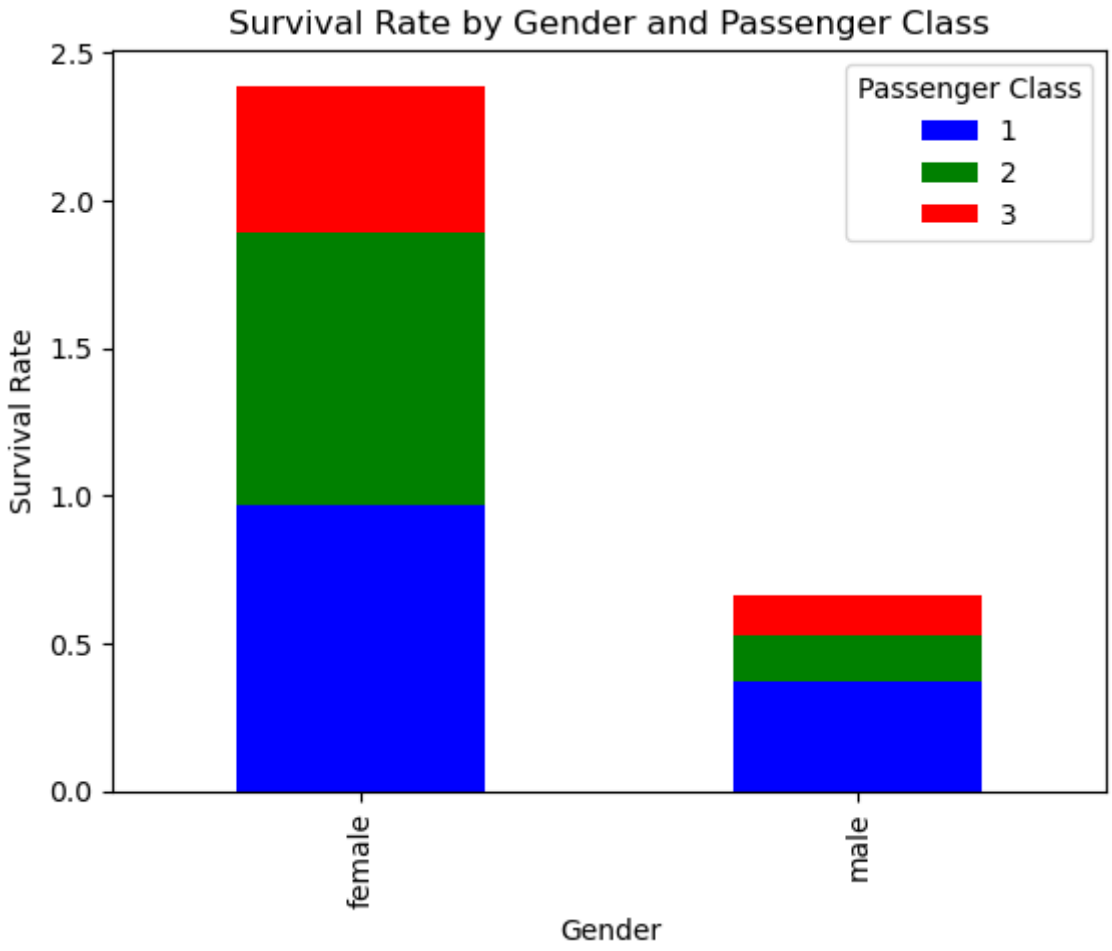
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Ca
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1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

Data Visualization

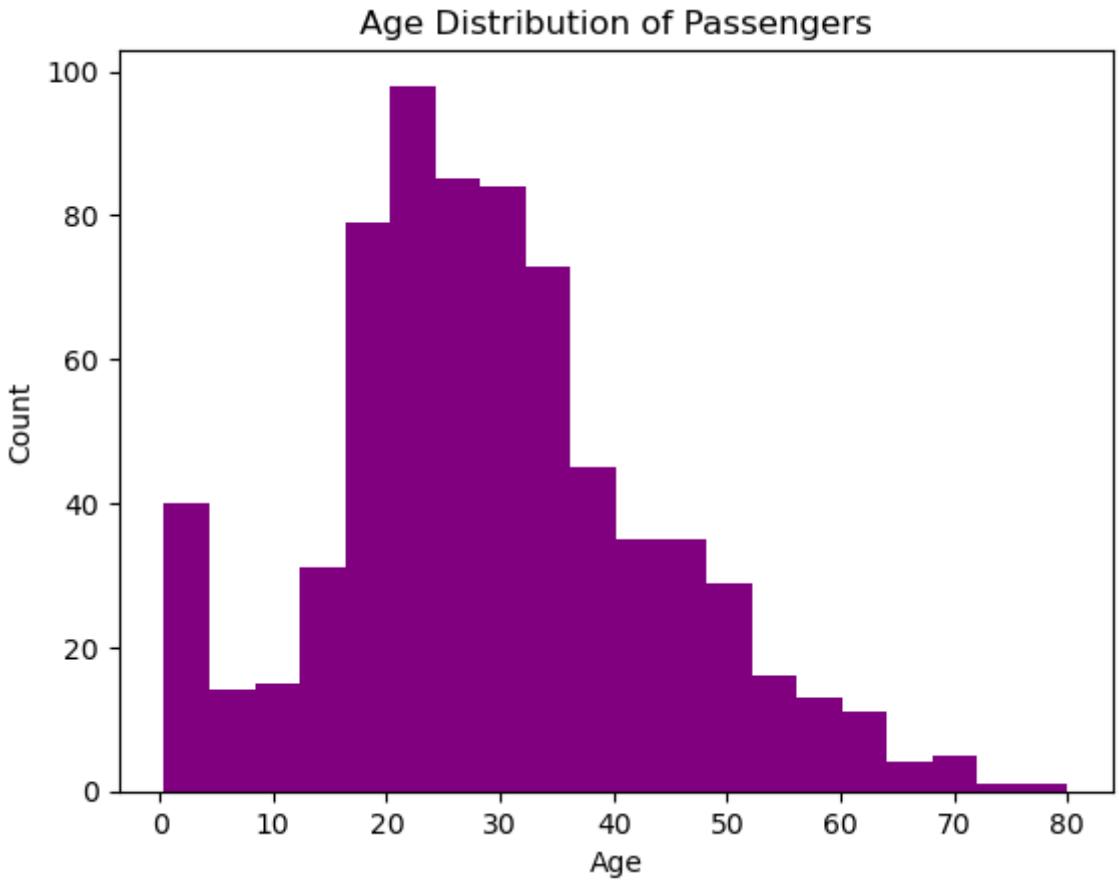
In [3]:
```
# Calculate the survival rate by gender
survival_rate = df.groupby("Sex")["Survived"].mean()

# Plot the survival rate by gender
survival_rate.plot(kind="bar", color=["b", "r"])
plt.xlabel("Gender")
plt.ylabel("Survival Rate")
plt.title("Survival Rate by Gender")
plt.show()
```



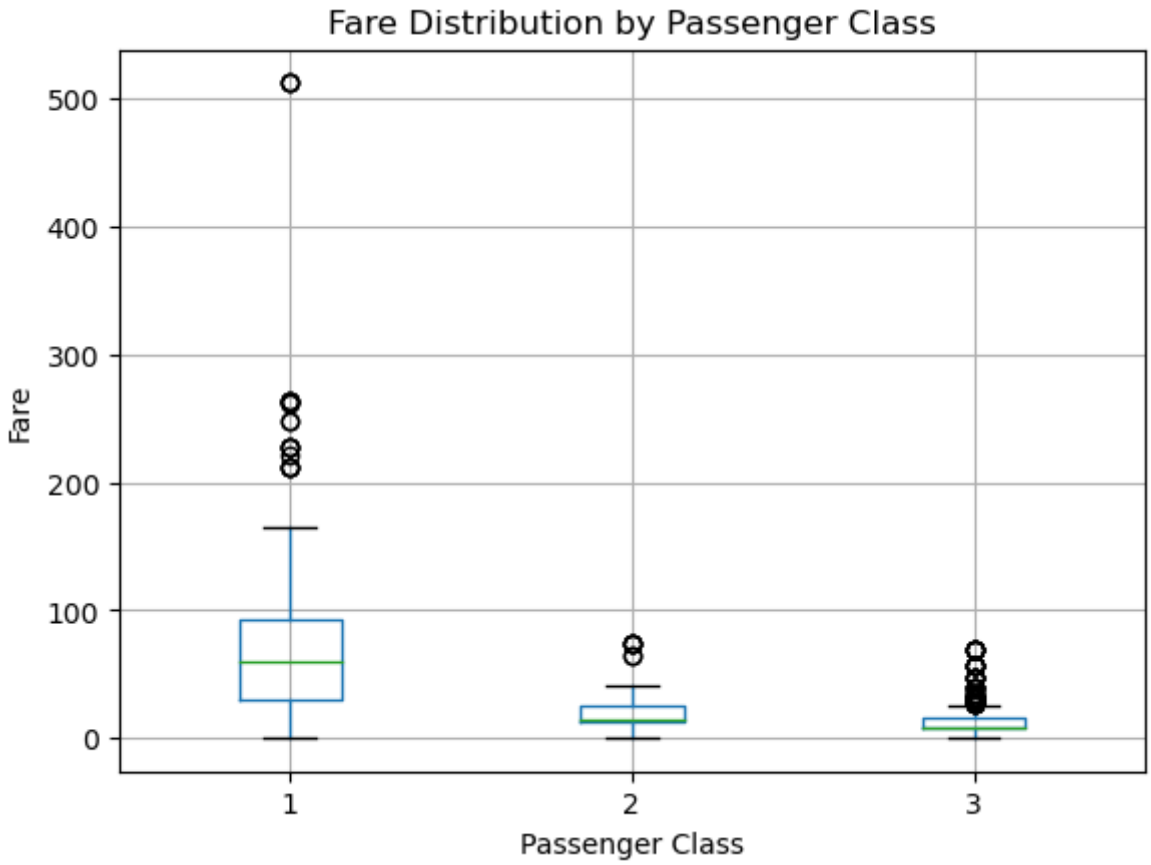
In [4]:
```
# Calculate the survival rate by gender and passenger class
survival_rate = df.groupby(["Sex", "Pclass"])["Survived"].mean().unstack()

# Plot the survival rate by gender and passenger class
survival_rate.plot(kind="bar", stacked=True, color=["b", "g", "r"])
plt.xlabel("Gender")
plt.ylabel("Survival Rate")
plt.title("Survival Rate by Gender and Passenger Class")
plt.legend(title="Passenger Class", loc="best")
plt.show()
```



In [5]:
```
# Plot the age distribution of passengers
plt.hist(df["Age"].dropna(), bins=20, color="purple")
plt.xlabel("Age")
plt.ylabel("Count")
plt.title("Age Distribution of Passengers")
plt.show()
```



In [6]:
```
# Plot the fare distribution by passenger class
df.boxplot(column="Fare", by="Pclass")
plt.xlabel("Passenger Class")
plt.ylabel("Fare")
plt.title("Fare Distribution by Passenger Class")
plt.suptitle("")
plt.show()
```



Defining the ReLU activation function using pandas and numpy package

In [7]:
```
def relu_function(z):
    return np.maximum(0, z)
```

The Log Loss function definition

In [8]:
```
def log_loss_function(y_true, y_predict):
    return -((y_true * np.log(y_predict)) + ((1 - y_true) * np.log(1 - y_predic
```

The function for stochastic gradient descent (SGD) optimization is defined.

In [9]:
```
def sgd_function(X, y_true, weights_generated, rate_of_learning):
    y_pred = relu_function(np.dot(X, weights_generated))
    error = y_pred - y_true
    gradient = np.dot(X.T, error) / len(X)
    weights_generated -= rate_of_learning * gradient
    return weights_generated
```

Create the training function for the logistic regression model with SGD.

In [10]:
```
def logistic_regression(X, y, activation_function=relu_function, rate_of_learni
    X = np.hstack((np.ones((X.shape[0], 1)), X))

    weights_generated = np.zeros(X.shape[1])

    # Looping over the specified number of epochs
    for epoch in range(epochs):
        # Looping over each training example
        for i in range(X.shape[0]):
            # Performing SGD update
            weights_generated = sgd_function(X[i], y[i], weights_generated, rat

    # Returning the learned weights
    return weights_generated
```

Checking how many null and NaN are there in the dataset

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [12]:
```
df.isna().sum()
```

Out[12]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Performing data wrangling

1. Dropping non-relevant columns
2. Convert categorical variables (ones with different categories) to one-hot encoding (labeling them)
3. Filling missing values with mean (average values)
4. Spliting the dataset into features and labels

5. Scaling the features
6. Spliting the titanic dataset into training and testing sets (80% and 20% respectively)

In [13]:
```
df = df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1)

df = pd.get_dummies(df, columns=['Sex', 'Embarked'])

df = df.fillna(df.mean())

X = df.drop('Survived', axis=1).values
y = df['Survived'].values

X = (X - X.mean()) / X.std()

np.random.seed(42)
idx = np.arange(X.shape[0])
np.random.shuffle(idx)
split_idx = int(X.shape[0] * 0.8)
X_train, X_test = X[idx[:split_idx]], X[idx[split_idx:]]
y_train, y_test = y[idx[:split_idx]], y[idx[split_idx:]]
```

checking if the no. of columns of the train and test data is same or not

In [14]:
```
print(X_test.shape)

print(y_test.shape)

print(X_train.shape)

print(y_train.shape)
```

```
(179, 10)
(179,)
(712, 10)
(712,)
```

Training this logistic regression model

In [15]:
```
weights_generated = logistic_regression(X_train, y_train, activation_function=r
```

Making predictions on the test set using Numpy package and using ReLu activation function instead of sigmoid

In [16]:
```
X_test = np.hstack((np.ones((X_test.shape[0], 1)), X_test))
y_pred = np.round(relu_function(np.dot(X_test, weights_generated)))
```

Calculating accuracy and F1 score

In [17]:
```
TP = np.sum((y_test == 1) & (y_pred == 1))
print(TP)
TN = np.sum((y_test == 0) & (y_pred == 0))
print(TN)
```

```
FP = np.sum((y_test == 0) & (y_pred == 1))
print(FP)
FN = np.sum((y_test == 1) & (y_pred == 0))
print(FN)

accuracy = (TP + TN) / (TP + TN + FP + FN)
print(accuracy)
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1 = 2 * precision * recall / (precision + recall)
print('Accuracy:', accuracy)
print('F1 Score:', f1)
```

```
33
104
10
32
0.7653631284916201
Accuracy: 0.7653631284916201
F1 Score: 0.611111111111111
```

This model is built with the accuracy of 76.5% and F1 score of 0.61.

some inferences drawn from this model

The Titanic dataset is a well-known example of a binary classification issue, where the objective is to predict a passenger's likelihood of survival based on characteristics like age, gender, ticket class, etc. Using this dataset, predictions can be made using a single neural network model. These are some conclusions that can be made using a single Titanic dataset neural network model:

The weights gained by the neural network can be used to determine which features are most crucial for predicting survival. For instance, the neural network might discover that having a higher ticket class and being female are both significant predictors of survival.

Relationships that are not linear: Unlike linear models like logistic regression, the neural network may learn non-linear relationships between the features and the target variable. The neural network might discover, for instance, that having a high ticket class and being young are important predictors of survival, which may not be clear from a simple linear relationship.

Overfitting: A neural network that has been trained for a long time or too intricately may overfit the training set and underperform when presented with fresh data. Comparing the model's performance on the training and validation sets can reveal this. The model may be overfitting if it performs significantly better on the training set than the validation set.

Generalization: Even if the distribution of the incoming data is slightly different, a neural network that has been trained successfully can nevertheless adapt well to it. By assessing the model on a held-out test set that wasn't used during training, this can be verified.

Ultimately, a single neural network model can offer insightful information on the connections between the features and the goal variable as well as the model's generalization capabilities. Yet it's vital to remember that neural networks have a tendency to overfit and don't always produce outcomes that are easy to understand.