Instructions:

1. Submit your python notebooks in zip format with naming convention as:

        RollNo1_RollNo2_RollNo3.zip

2. Cheating of any form will not be tolerated.

Fill your Team details here.

Format: Roll Number

1.
2.
3.

Question: You need to build Logistic Regression from scratch using training set of Titanic dataset.

Import necessary packages.

In [ ]:
```
# Import necessary packages
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

Set seed value to 100.

In [ ]:
```
# Set seed value to 100.
np.random.seed(100)
```

Load the "train.csv" dataset. You will be using the same file for sampling training and testing points.

In [ ]:
```
# Load train.csv dataset
data = pd.read_csv("train.csv")
category = ["Sex","Age","Embarked"]
```

Remove all missing rows and columns from the dataset.

In [ ]:
```
# use dropna() to remove null values from data

data.dropna(inplace=True)
```

Select following features from dataset.

1. Sex
2. Age
3. Embarked

In [ ]:
```
# Select Sex, Age, Survived columns.

data = data[[category[0],category[1],category[2]]]

# Replace class labels with numerical data

data[category[2]] = data[category[2]].replace(to_replace = "C", value
data[category[2]] = data[category[2]].replace(to_replace = "S", value
```

The target variable to be predicted is whether a person will "survive" the Titanic tragedy or not.

Store the target 'Survived' in 'y' variable and other variables in 'X'.

In [ ]:
```
# Store 'Survived' in y variable and other variables in X.
y = data[category[2]]
del(data[category[2]])
X = data
```

The values in the 'Sex' column are 'Male/Female'. So convert them into 1/0 using Label Encoding.

In [ ]:
```
le = LabelEncoder()

X['Sex'] = le.fit_transform(X['Sex'])
```

Split the dataset into train and test with a test size of 20% of total dataset.

In [ ]:
```
# Split dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

Convert X_train, y_train, X_test, y_test into numpy arrays.

In [ ]:
```

X_train = X_train.values
y_train = y_train.values

X_test =  X_test.values
y_test =  y_test.values


```

Define sigmoid function.

1. Input: An array.
2. Output: Sigmoid of Input.

In [ ]:
```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

Define loss function for logistic regression.

y is the label value

In [ ]:
```
def probability(theta, x):
    return sigmoid(np.dot(x, theta)catagorical_features=[0])
```

In [ ]:
```
def loss(theta,x,y):
    m = x.shape[0]
    total_cost = -(1 / m) * np.sum(y * np.log(probability(theta, x)) +
    return total_cost
```

Create a class for Logistic Regression function.

```
Input X, y, NumberOfIterations, LearningRate.
Output: Updated weights.
```

In [ ]:
```
def LogisticRegression(X, y, NumberOfIterations, LearningRate):

    # Initialize weights randomly
    weights = (np.random.rand(2))
    weights = weights.reshape(2,1)
    for i in range(NumberOfIterations):

        # Forward pass
        Z = np.dot(X, weights)
        A = sigmoid(Z)

        # Loss Computation
        J = loss(weights,X,y)

        # Gradient computation
        m = X.shape[0]
        dweights = (1 / m) * np.dot(X.T, A - y)

        # Update weights
        weights = weights- LearningRate*dweights

        # Printing loss after every 100 iterations
        if(i % 100 == 0):
            print('loss:' + str(J) + '\t')
    return weights
```

Define prediction function.

```
Input: X, threshold, weights.
Output: Corresponding labels for data.
```

In [ ]:
```
def predict(X, threshold, weights):
    val = []
    Z = np.dot(X, weights)
    A = sigmoid(Z)
    if(A>threshold):
        val.append(1)
    else:
        val.append(0)
    return val
```

Call LogisticRegression function with following inputs to train on training set.

1. X_train
2. y_train
3. NumberOfIterations = 1000
4. LearningRate = 0.1

In [ ]:
```
# Call  LogisticRegression function and store weights in model.
model = LogisticRegression(X_train,y_train,1000,0.1)
```

Make predictions on testing data. Store predictions in preds variable.

In [ ]:
```
# Store predictions in preds variable.
preds = predict(X_test,0.5,model)
preds = preds.reshape(X_test.shape[0],1)
```

Compute the accuracy on test dataset given y_test (in the beginning)

In [ ]:
```
ctr = 0
l = preds.shape[0]
for i in range(l):
    if preds[i] == y_test[i]:
        ctr = ctr+1
Accuracy = (ctr/l)*100
```

What is the reason behind such low accuracy? What do you think are possible ways of improving it?

In [ ]:
```
# Accuracy: 55.091235%
# Type out the answer below. Due to many non correlated columns, the n
```

Type Here!

In [ ]:

In [ ]: