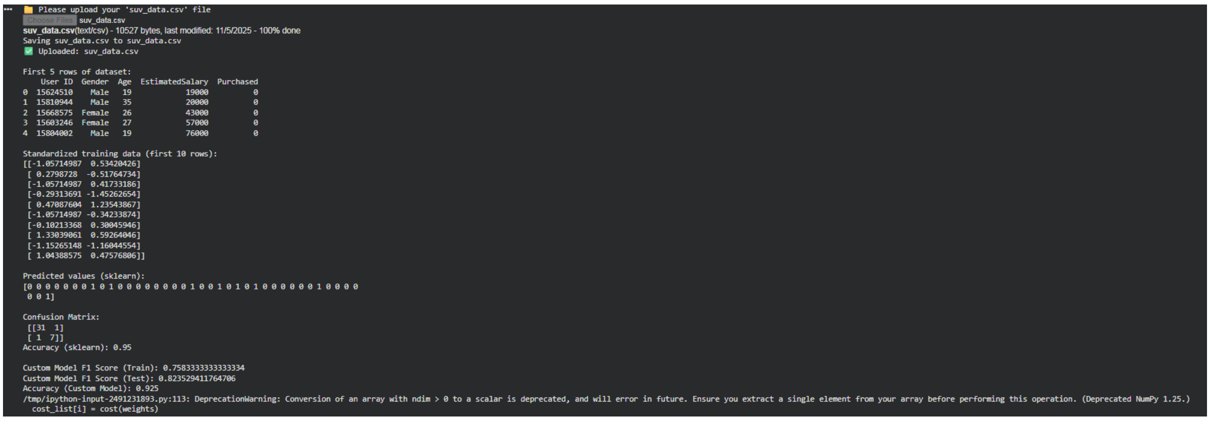
```
# ==============================
# Logistic Regression Experiment
# ==============================

# Step 1: Import required libraries
import pandas as pd
import numpy as np
from numpy import log, dot, exp, shape
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

# Step 2: Upload dataset to Colab
from google.colab import files
print("📁 Please upload your 'suv_data.csv' file")
uploaded = files.upload()   # Upload file manually
filename = list(uploaded.keys())[0]
print(f"✅ Uploaded: {filename}")

# Step 3: Load the dataset
data = pd.read_csv(filename)
print("\nFirst 5 rows of dataset:")
print(data.head())

# Step 4: Prepare independent and dependent variables
x = data.iloc[:, [2, 3]].values   # Age, EstimatedSalary
y = data.iloc[:, 4].values        # Purchased

# Step 5: Split data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.10, random_state=0)
```

```
# Step 6: Standardize features
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
print("\nStandardized training data (first 10 rows):")
print(x_train[0:10, :])

# Step 7: Logistic Regression using sklearn
classifier = LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)

print("\nPredicted values (sklearn):")
print(y_pred)

cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", cm)
print("Accuracy (sklearn):", accuracy_score(y_test, y_pred))

# =========================
# USER DEFINED IMPLEMENTATION
# =========================

# Step 8: Re-split (for consistency)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.10, random_state=0)

# Step 9: Standardization (manual)
def Std(input_data):
  mean0 = np.mean(input_data[:, 0])
  sd0 = np.std(input_data[:, 0])
```

```
    mean1 = np.mean(input_data[:, 1])
    sd1 = np.std(input_data[:, 1])
    return lambda x: ((x[0]-mean0)/sd0, (x[1]-mean1)/sd1)

my_std = Std(x)
my_std(x_train[0])

def standardize(X_tr):
    for i in range(shape(X_tr)[1]):
        X_tr[:, i] = (X_tr[:, i] - np.mean(X_tr[:, i])) / np.std(X_tr[:, i])

def F1_score(y, y_hat):
    tp = tn = fp = fn = 0
    for i in range(len(y)):
        if y[i] == 1 and y_hat[i] == 1:
            tp += 1
        elif y[i] == 1 and y_hat[i] == 0:
            fn += 1
        elif y[i] == 0 and y_hat[i] == 1:
            fp += 1
        elif y[i] == 0 and y_hat[i] == 0:
            tn += 1
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    f1_score = 2 * precision * recall / (precision + recall)
    return f1_score

# Step 10: User-defined Logistic Regression Class
class LogisticRegressionCustom:
    def sigmoid(self, z):
        return 1 / (1 + exp(-z))
    def initialize(self, X):
        weights = np.zeros((shape(X)[1] + 1, 1))
        X = np.c_[np.ones((shape(X)[0], 1)), X]
        return weights, X

    def fit(self, X, y, alpha=0.001, iterations=400):
        weights, X = self.initialize(X)

        def cost(theta):
            z = dot(X, theta)
            cost0 = y.T.dot(log(self.sigmoid(z)))
            cost1 = (1 - y).T.dot(log(1 - self.sigmoid(z)))
            cost = -((cost1 + cost0)) / len(y)
            return cost

        cost_list = np.zeros(iterations,)
        for i in range(iterations):
            weights = weights - alpha * dot(X.T, self.sigmoid(dot(X, weights)) - np.reshape(y, (len(y), 1)))
            cost_list[i] = cost(weights)
        self.weights = weights
        return cost_list

    def predict(self, X):
        z = dot(self.initialize(X)[1], self.weights)
        lis = []
        for i in self.sigmoid(z):
            lis.append(1 if i > 0.5 else 0)
        return lis

# Step 11: Train and test custom model
```

```
standardize(x_train)

standardize(x_test)

obj1 = LogisticRegressionCustom()

obj1.fit(x_train, y_train)

y_pred = obj1.predict(x_test)

y_trainn = obj1.predict(x_train)


# Step 12: Evaluate model

f1_score_tr = F1_score(y_train, y_trainn)

f1_score_te = F1_score(y_test, y_pred)

print("\nCustom Model F1 Score (Train):", f1_score_tr)

print("Custom Model F1 Score (Test):", f1_score_te)


conf_mat = confusion_matrix(y_test, y_pred)

accuracy = (conf_mat[0, 0] + conf_mat[1, 1]) / sum(sum(conf_mat))

print("Accuracy (Custom Model):", accuracy)
```



```
📁 Please upload your 'suv_data.csv' file
suv_data.csv
suv_data.csv(text/csv) - 10527 bytes, last modified: 11/5/2025 - 100% done
Saving suv_data.csv to suv_data.csv
✅ Uploaded: suv_data.csv

First 5 rows of dataset:
    User ID  Gender  Age  EstimatedSalary  Purchased
0  15624510    Male   19            19000          0
1  15810944    Male   35            20000          0
2  15668575  Female   26            43000          0
3  15603246  Female   27            57000          0
4  15804002    Male   19            76000          0

Standardized training data (first 10 rows):
[[-1.05714987  0.53420426]
 [ 0.2798728  -0.51764734]
 [-1.05714987  0.41733186]
 [-0.29313691 -1.45262654]
 [ 0.47087604  1.23543867]
 [-1.05714987 -0.34233874]
 [-0.10213368  0.30045946]
 [ 1.33039061  0.59264046]
 [-1.15265148 -1.16044554]
 [ 1.04388575  0.47576806]]

Predicted values (sklearn):
[0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0
 0 0 1]

Confusion Matrix:
 [[31  1]
 [ 1  7]]
Accuracy (sklearn): 0.95

Custom Model F1 Score (Train): 0.7583333333333334
Custom Model F1 Score (Test): 0.823529411764706
Accuracy (Custom Model): 0.925
/tmp/ipython-input-2491231893.py:113: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated NumPy 1.25.)
  cost_list[i] = cost(weights)
```