LAB Manual

**Experiment No.01**

PART B

| Roll No. C009 | Name: Samarth Borade |
| --- | --- |
| Class : BTI SEM 10 | Batch : B1 |
| Date of Experiment: 15/12/23 | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student:**

**​​#Samarth Borade**

**#Roll No: C009**

**#BTI SEM 10**

**#Batch: B1**

**#Exp No: 1**

**#Aim: Implementation of Perceptron from scratch**

**# Modules used for data handling and linear algebra operations.**

**import pandas as pd**

**import numpy as np**

**# Modules used for data visualization**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**sns.set\_style()**

**# Modules used for encoding the categorical variables.**

**from sklearn.preprocessing import OneHotEncoder**

**df = pd.read\_csv("crx.csv",*header*=None)**

**df.head()**

**df.columns = df.iloc[0]**

**df = df[1:]**

**df.head()**

**df.info()**

**df.dtypes.value\_counts().plot(*kind*="bar",*title*="Types of Data",*xlabel*="Data Type",*ylabel*="No.of columns",*rot*=0,*color*=["crimson","orange"])**

**plt.show()**

**cat\_cols = []**

**num\_cols = []**

**for i in df.columns:**

**if df[i].dtype == "O":**

**cat\_cols.append(i)**

**else:**

**num\_cols.append(i)**

**null\_freq = []**

**for i in df.columns:**

**f = dict(df[i].value\_counts())**

**if "?" in f.keys():**

**null\_freq.append(f["?"]\*100/len(df))**

**else:**

**null\_freq.append(0)**

**pd.Series(dict(zip(df.columns,null\_freq))).plot(*kind*="bar",*rot*=0,*title*="Missing Value Frequency",*xlabel*="Column Name",*ylabel*="Percentage of missing values",*color*=["orange","crimson"])**

**plt.show()**

**df['a16'].value\_counts().plot(*kind*="bar",*title*="Class Distribution",*xlabel*="Status of Credit Card Approval",*ylabel*="Frequency of the Status",*color*=["crimson","orange"],**

***rot*=0)**

**plt.show()**

**df = df.replace({"?":None})**

**df = df.dropna()**

**numerical\_columns = df.select\_dtypes(*include*=['float64', 'int64']).columns**

**Q1 = df[numerical\_columns].quantile(0.25)**

**Q3 = df[numerical\_columns].quantile(0.75)**

**IQR = Q3 - Q1**

**threshold = 1.5**

**# Remove rows with values outside the acceptable range**

**df = df[~((df[numerical\_columns] < (Q1 - threshold \* IQR)) | (df[numerical\_columns] > (Q3 + threshold \* IQR))).any(*axis*=1)]**

**encoder = OneHotEncoder(*sparse*=False)**

**for i in cat\_cols:**

**df[i] = encoder.fit\_transform(df[i].values.reshape(-1,1))**

**from sklearn.model\_selection import train\_test\_split**

**X\_train,X\_test,y\_train,y\_test = train\_test\_split(df,**

**df['a16'],**

***test\_size* = 0.10,**

***train\_size*=0.90,*random\_state* = 0**

**)**

**X\_train.pop('a16')**

**X\_test.pop('a16')**

**class Perceptron:**

**# Initialising the required parameters for the perceptron.**

**def \_\_init\_\_(*self*, *X*, *y*, *learning\_rate*, *epochs* : int):**

***self*.X = *X***

***self*.y = *y***

***self*.learning\_rate = *learning\_rate***

***self*.epochs = *epochs***

**# Activation function.**

**def \_\_activation\_function(*self*,*x*):**

**return 1.0 if (*x* > 0) else 0.0**

**# The model training or fitting by updating weights.**

**def fit(*self*):**

**n\_rows,n\_cols = *self*.X.shape**

***self*.weights = np.zeros((n\_cols + 1, 1))**

**for epoch in range(*self*.epochs):**

**for index, features in enumerate(*self*.X.values):**

**feature\_transposed = np.insert(features, 0, 1).reshape(-1,1)**

**predicted\_target = *self*.\_\_activation\_function(np.dot(feature\_transposed.T, *self*.weights))**

**flag = np.squeeze(predicted\_target) - *self*.y[index]**

**if flag != 0:**

***self*.weights += *self*.learning\_rate\*((*self*.y[index] - predicted\_target)\*feature\_transposed)**

**# Predicting on a single instance.**

**def predict(*self*, *X\_test*):**

**return *self*.\_\_activation\_function(np.dot(p.weights.reshape(1,-1)[0],*X\_test*))**

**# Predicting on a larger number of instances and returning accuracy.**

**def test(*self*, *test\_data*, *y*):**

**x = []**

**for i in range(len(*test\_data*.values)):**

**X\_test = np.array(*test\_data*.iloc[i])**

**x.append(p.predict(np.insert(X\_test,0,1)) == p.y[i])**

**return sum(x)\*100/len(*test\_data*)**

**y = np.array(pd.DataFrame(y\_train).reset\_index().drop(["index"],*axis*=1))**

**X = pd.DataFrame(X\_train).reset\_index().drop(["index"],*axis*=1)**

**p = Perceptron(X, y, 0.5, 50)**

**p.fit()**

**p.test(pd.DataFrame(X\_test).reset\_index().drop(["index"],*axis*=1),**

**np.array(pd.DataFrame(y\_test).reset\_index().drop(["index"],*axis*=1)))**

**train\_acc = []**

**test\_acc = []**

**epochs = []**

**for i in range(20,200,20):**

**y = np.array(pd.DataFrame(y\_train).reset\_index().drop(["index"],*axis*=1))**

**X = pd.DataFrame(X\_train).reset\_index().drop(["index"],*axis*=1)**

**p = Perceptron(X, y, 0.5, i)**

**p.fit()**

**train\_acc.append(p.test(X,y))**

**test\_acc.append(p.test(pd.DataFrame(X\_test).reset\_index().drop(["index"],*axis*=1),**

**np.array(pd.DataFrame(y\_test).reset\_index().drop(["index"],*axis*=1))))**

**epochs.append(i)**

**plt.figure(*figsize*=(10,10))**

**plt.plot(epochs,train\_acc)**

**plt.plot(epochs,test\_acc)**

**plt.xlabel("Number of Epochs")**

**plt.ylabel("Accuracy of the model")**

**plt.legend(['Train Accuracy',"Test Accuracy"])**

**plt.show()**

**tuning = pd.DataFrame({"epochs":np.array(epochs).squeeze(),"train accuracy":np.array(train\_acc).squeeze(), "test accuracy":np.array(test\_acc).squeeze()})**

**def diff(*row*):**

**return *row*[1] - *row*[2]**

**tuning["Difference"] = tuning.apply(diff,*axis*=1)**

**tuning.sort\_values(*by*="Difference")**

**For Dataset 2:**

# Cirrhosis Patient Survival Prediction

df = pd.read\_csv("cirrhosis.csv",*header*=None)

from sklearn.model\_selection import train\_test\_split

X\_train,X\_test,y\_train,y\_test = train\_test\_split(df,

df['Stage'],

*test\_size* = 0.10,

*train\_size*=0.90,

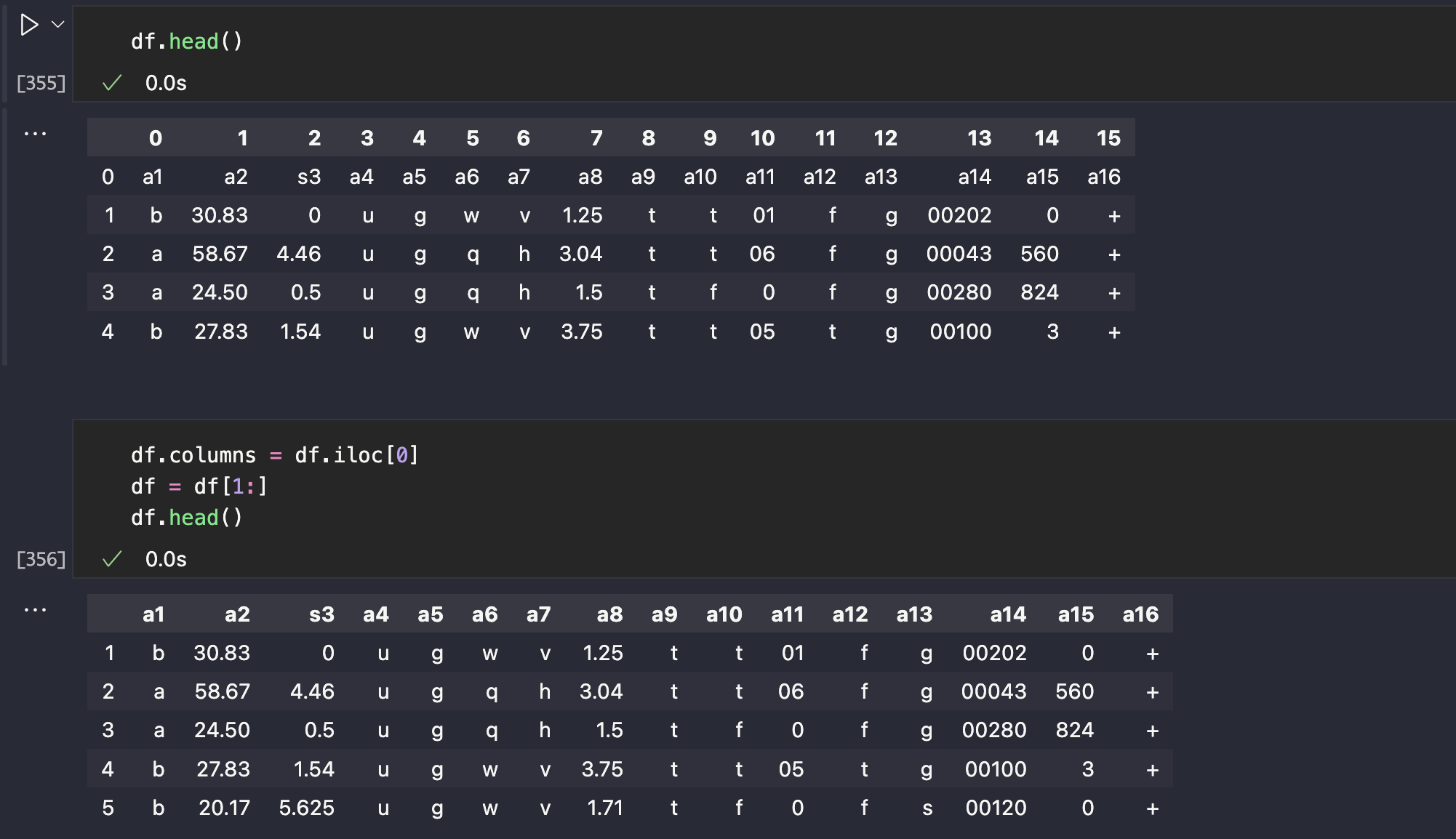
*random\_state* = 0)

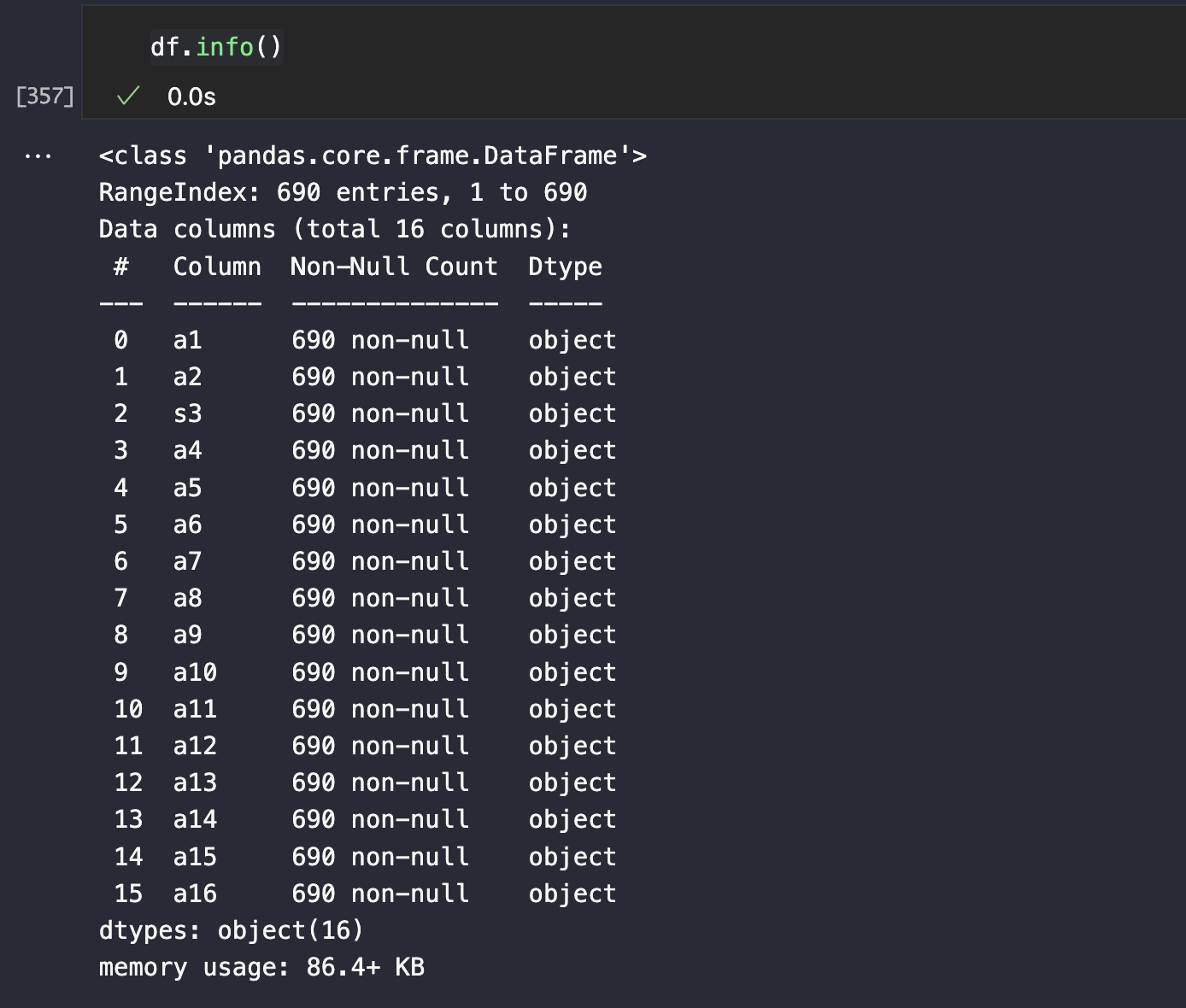
X\_train.pop('Stage')

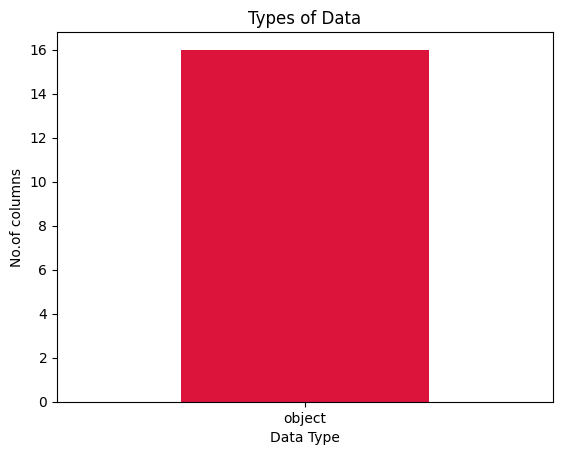
X\_test.pop('Stage')

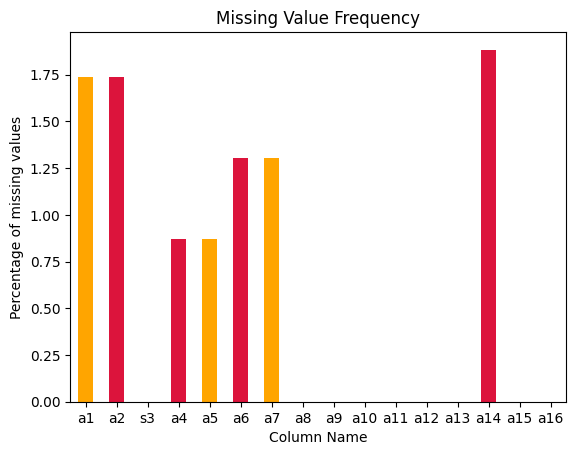
**B.2 Input and Output:**

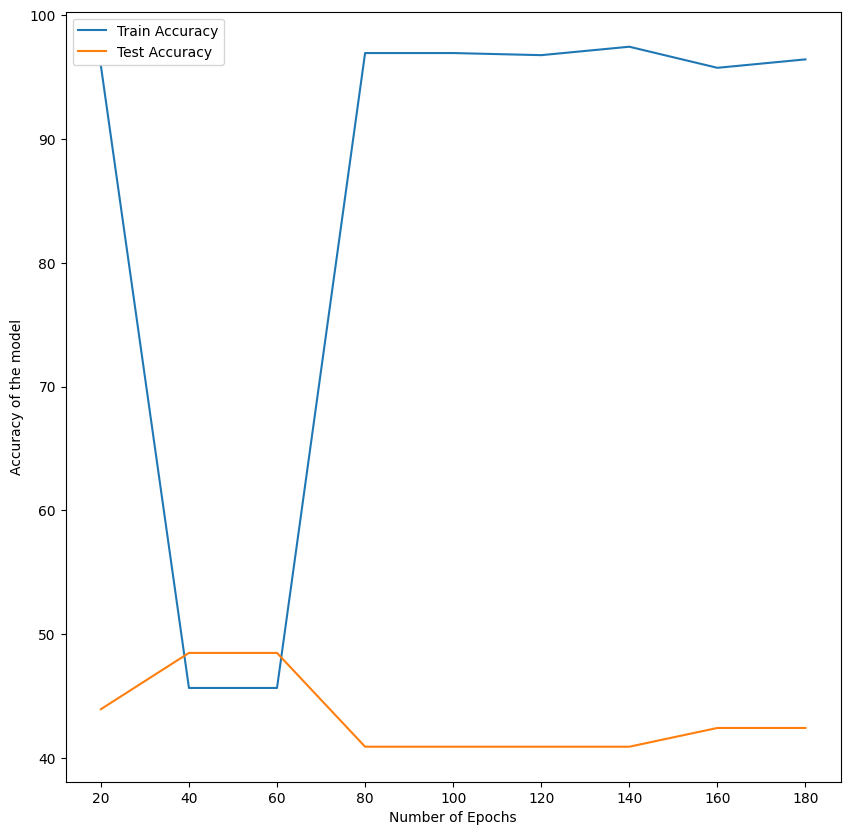
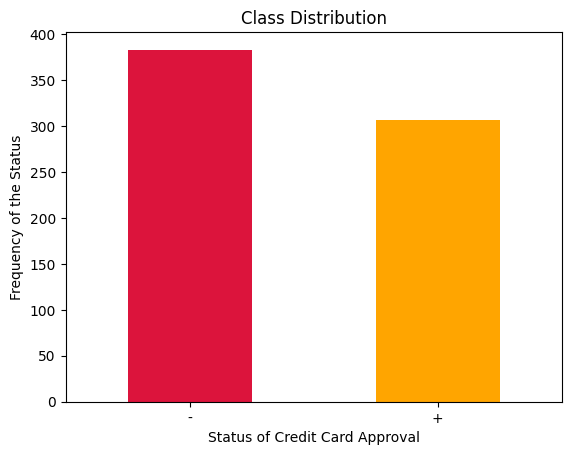
**Input Dataset 1:**

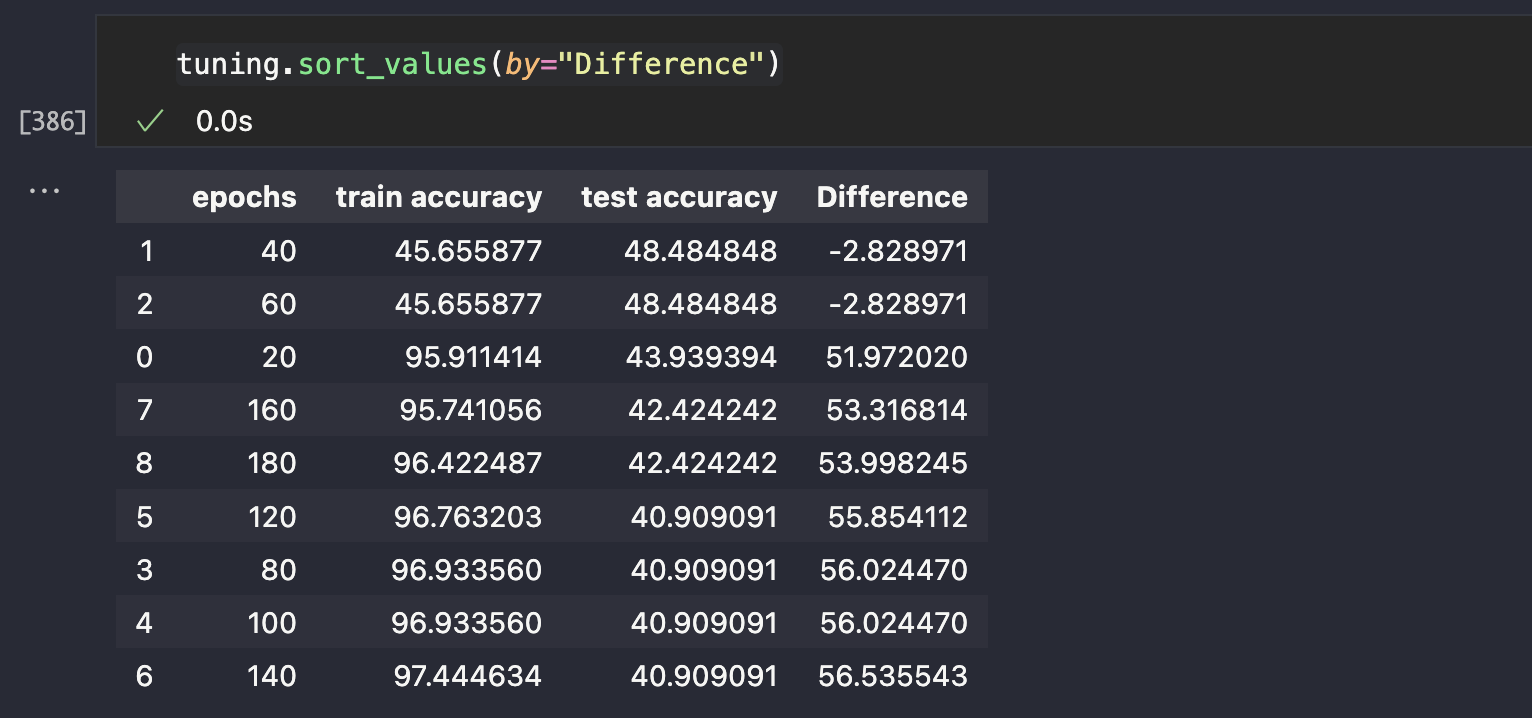
****

****

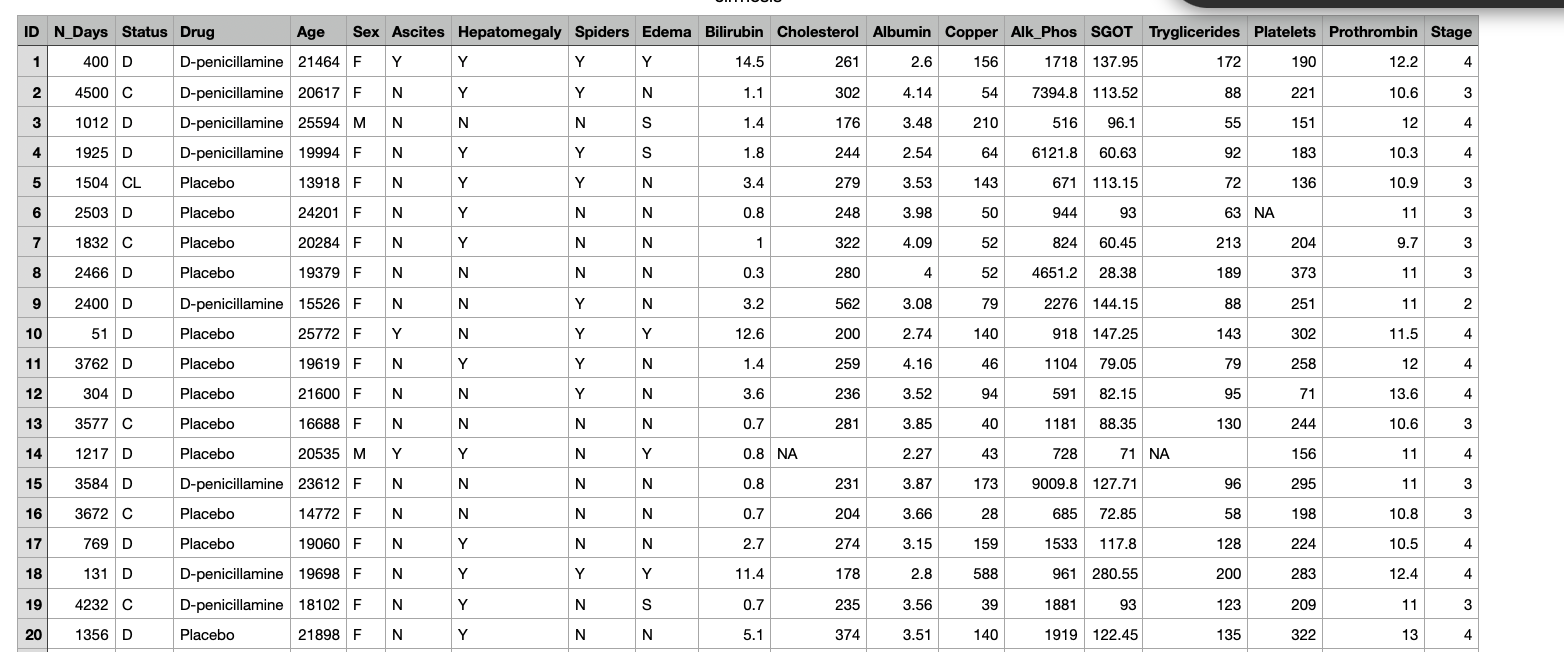
****

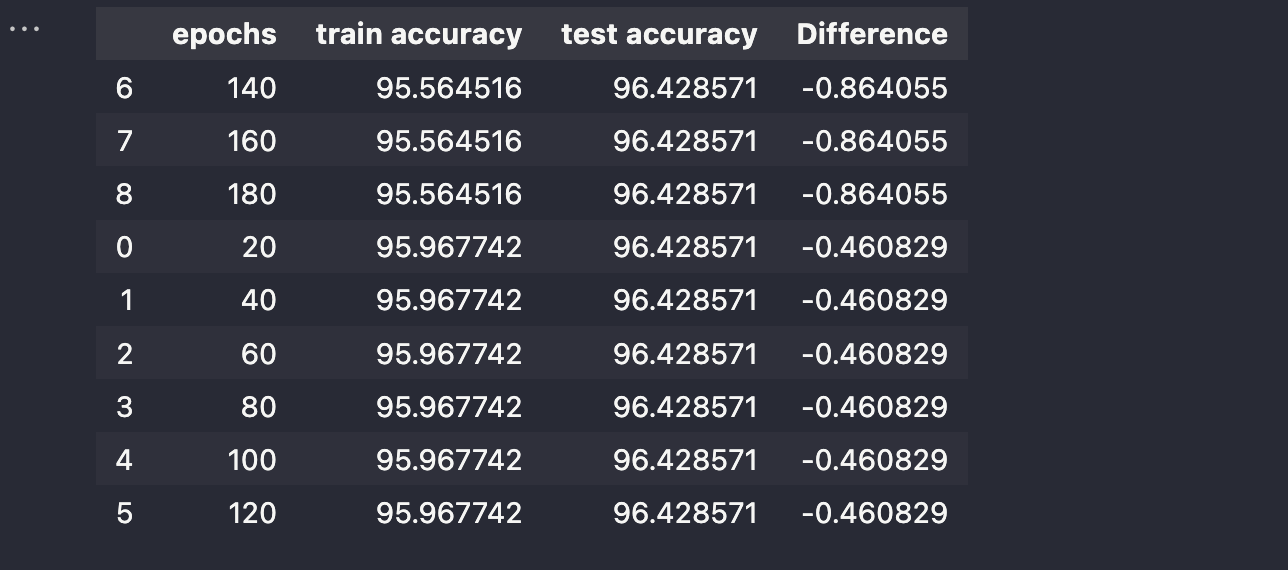
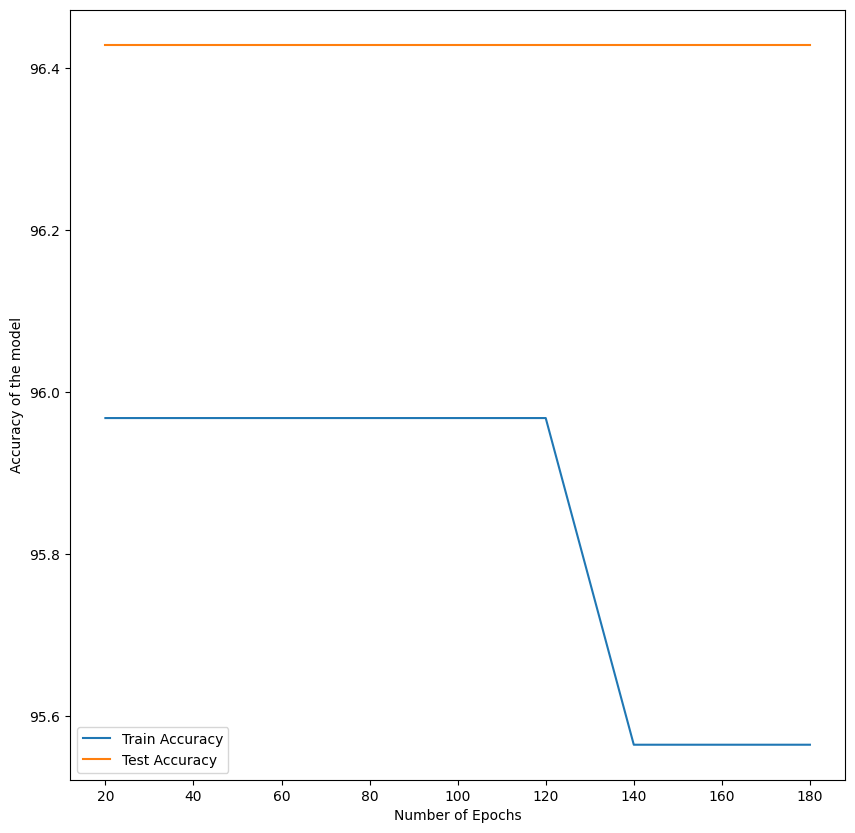
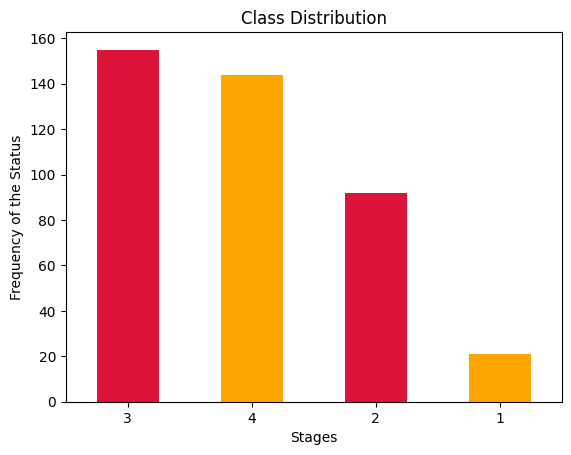
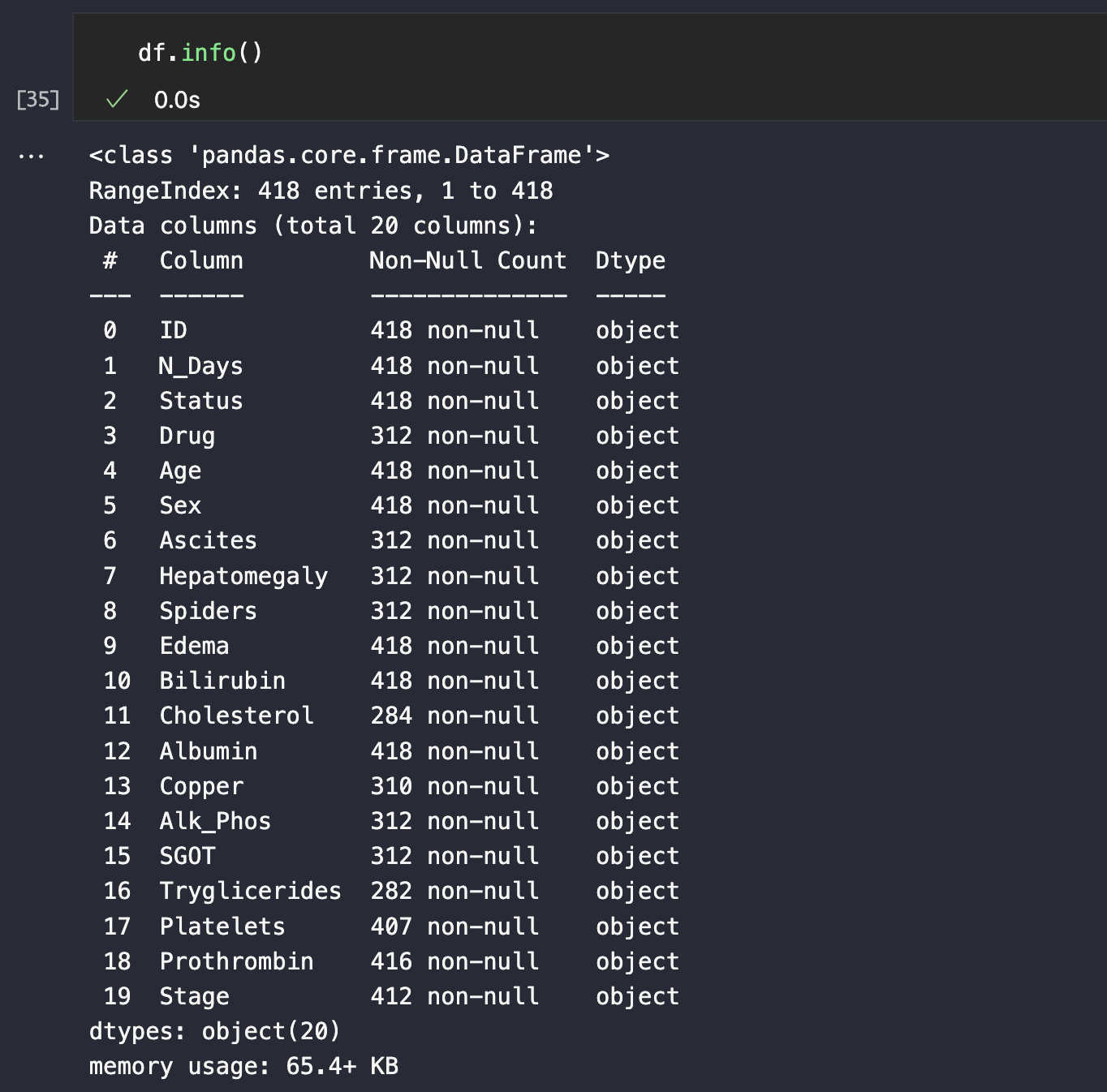
****

****

****

**Dataset 2:**

****

****

**B.3 Observations and learning:**

* Optimal performance was observed in the model trained for 40 epochs during testing.
* Constraints in adjusting hyperparameters arose due to the Perceptron being coded from scratch.
* The model trained for 80 epochs was excluded from consideration due to its lower test accuracy compared to its training accuracy, a scenario not aligned with ideal model behavior.

**B.4 Conclusion:**

The process of implementing the Perceptron for Cirrhosis Patient Survival Prediction deepened my understanding of its workings and their application in real-world scenarios. This successful optimization underscores the potential of tailored models in addressing specific medical predictive tasks with improved accuracy and reliability . Moreover, noteworthy improvements were achieved in Dataset 2, further affirming the efficacy of the optimized Perceptron model in diverse dataset scenarios.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*