# PART B

|  |  |  |
| --- | --- | --- |
| (PART B : TO BE COMPL | ETED BY STU | DENTS) |

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. C050 | Name: Nisha Kini |
| Class : BTI | Batch : EB2 |
| Date of Experiment: 20.12.23 | Date of Submission: 21.12.23 |
| Grade : |  |

**B.1 Documentation written by student:**

**Name: Nisha Kini**

**Roll No: C050**

**Aim: Study the Architecture of Multi-Layer Perceptron (MLP) Using Python**

**:**

**CRX.CSV**

Code:

#

#try displaying hidden layer of perceptron in diagram, architecture of mlp.

# 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 preprocessing

from sklearn.preprocessing import OneHotEncoder

# Modules used for Machine Learning models.

from sklearn.linear\_model import Perceptron

from sklearn.neural\_network import MLPClassifier

# Modules used for hyperparameter tuning.

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import RandomizedSearchCV

# Models used for evaluating the model.

from sklearn import metrics

from sklearn.model\_selection import cross\_val\_score

# Suppressing the warnings.

import warnings

warnings.filterwarnings('ignore')

df = pd.read\_csv("/crx.csv")

df.head()

df.info()

"""## Data Type Distribution"""

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()

encoder = OneHotEncoder(sparse=False)

for i in cat\_cols:

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

df = df.reset\_index()

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')

clf = Perceptron(random\_state=0)

X\_train.columns = X\_train.columns.astype(str)

clf.fit(X\_train, y\_train)

X\_test.columns = X\_test.columns.astype(str)

y\_pred\_train = clf.predict(X\_train)

y\_pred\_test = clf.predict(X\_test)

metrics.accuracy\_score(y\_true=y\_train,y\_pred=y\_pred\_train)

metrics.accuracy\_score(y\_true=y\_test,y\_pred=y\_pred\_test)

clf = MLPClassifier(random\_state=1, max\_iter=300).fit(X\_train, y\_train)

y\_pred\_train = clf.predict(X\_train)

y\_pred\_test = clf.predict(X\_test)

metrics.accuracy\_score(y\_true=y\_train,y\_pred=y\_pred\_train)

metrics.accuracy\_score(y\_true=y\_test,y\_pred=y\_pred\_test)

df

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import StandardScaler

# Assuming df is your original DataFrame with both numerical and categorical columns

# Replace missing values ('?') with NaN and drop rows with NaN values

df.replace('?', np.nan, inplace=True)

df.dropna(inplace=True)

# Extract features and target variable

X = df.drop('a16', axis=1)

y = df['a16'].map({'-': 0, '+': 1}) # Assuming binary classification

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the numerical features

scaler = StandardScaler()

X\_train\_numeric = scaler.fit\_transform(X\_train[['a2', 's3', 'a8', 'a11', 'a14', 'a15']])

X\_test\_numeric = scaler.transform(X\_test[['a2', 's3', 'a8', 'a11', 'a14', 'a15']])

# Combine the standardized numerical features with the one-hot encoded categorical features for training set

X\_train\_numeric\_df = pd.DataFrame(X\_train\_numeric, columns=['a2', 's3', 'a8', 'a11', 'a14', 'a15'])

X\_train\_processed = pd.concat([X\_train\_numeric\_df, X\_train.drop(['a2', 's3', 'a8', 'a11', 'a14', 'a15'], axis=1)], axis=1)

# Combine the standardized numerical features with the one-hot encoded categorical features for testing set

X\_test\_numeric\_df = pd.DataFrame(X\_test\_numeric, columns=['a2', 's3', 'a8', 'a11', 'a14', 'a15'])

X\_test\_processed = pd.concat([X\_test\_numeric\_df, X\_test.drop(['a2', 's3', 'a8', 'a11', 'a14', 'a15'], axis=1)], axis=1)

y

df

# Assuming df is your DataFrame with both numerical and categorical columns

# Replace missing values ('?') with NaN and drop rows with NaN values

df.replace('?', np.nan, inplace=True)

df.dropna(inplace=True)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Build the MLP model with a hidden layer

model = Sequential()

model.add(Dense(units=64, activation='relu', input\_dim=X.shape[1])) # Input layer

model.add(Dense(units=32, activation='relu')) # Hidden layer

model.add(Dense(units=1, activation='sigmoid')) # Output layer

# Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Display the architecture of the MLP

model.summary()

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import plot\_model

# Assuming X\_train has the input features and y\_train has the labels

# You should replace these with your actual dataset

# Define the MLP model

model = Sequential()

model.add(Dense(units=64, activation='relu', input\_shape=(X\_train.shape[1],))) # Input layer

model.add(Dense(units=32, activation='relu')) # Hidden layer

model.add(Dense(units=1, activation='sigmoid')) # Output layer

# Compile the model (you should replace 'binary\_crossentropy' and 'adam' with appropriate values)

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Display the architecture of the MLP

plot\_model(model, to\_file='mlp\_architecture.png', show\_shapes=True, show\_layer\_names=True)

"""## Comparative Inferences

- The Perceptron model gives an accuracy of 67% and 75% on the test and train dataset respectively.

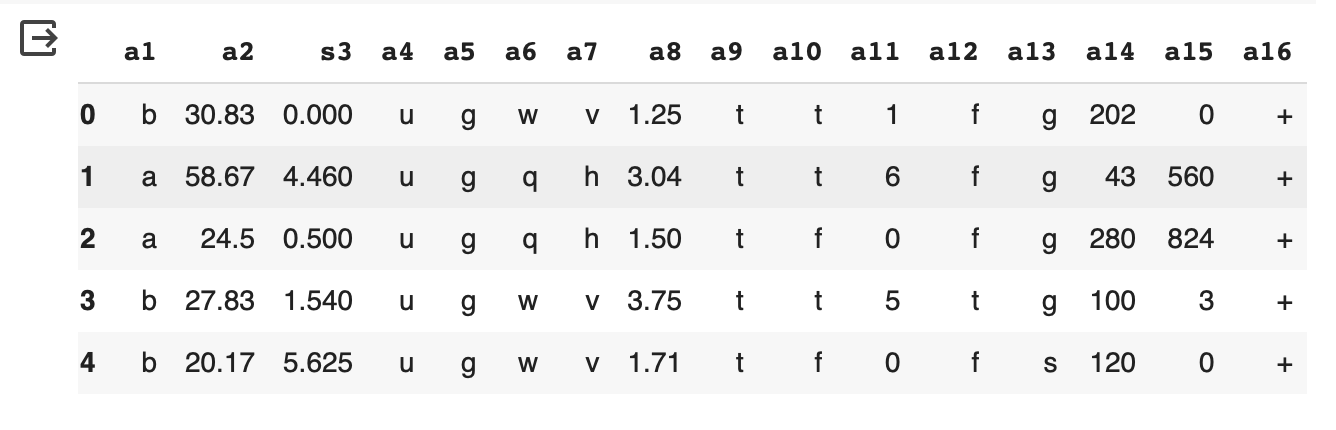
- The MLP model gives an accuracy of 78% and 79% on the test and train dataset respectively.

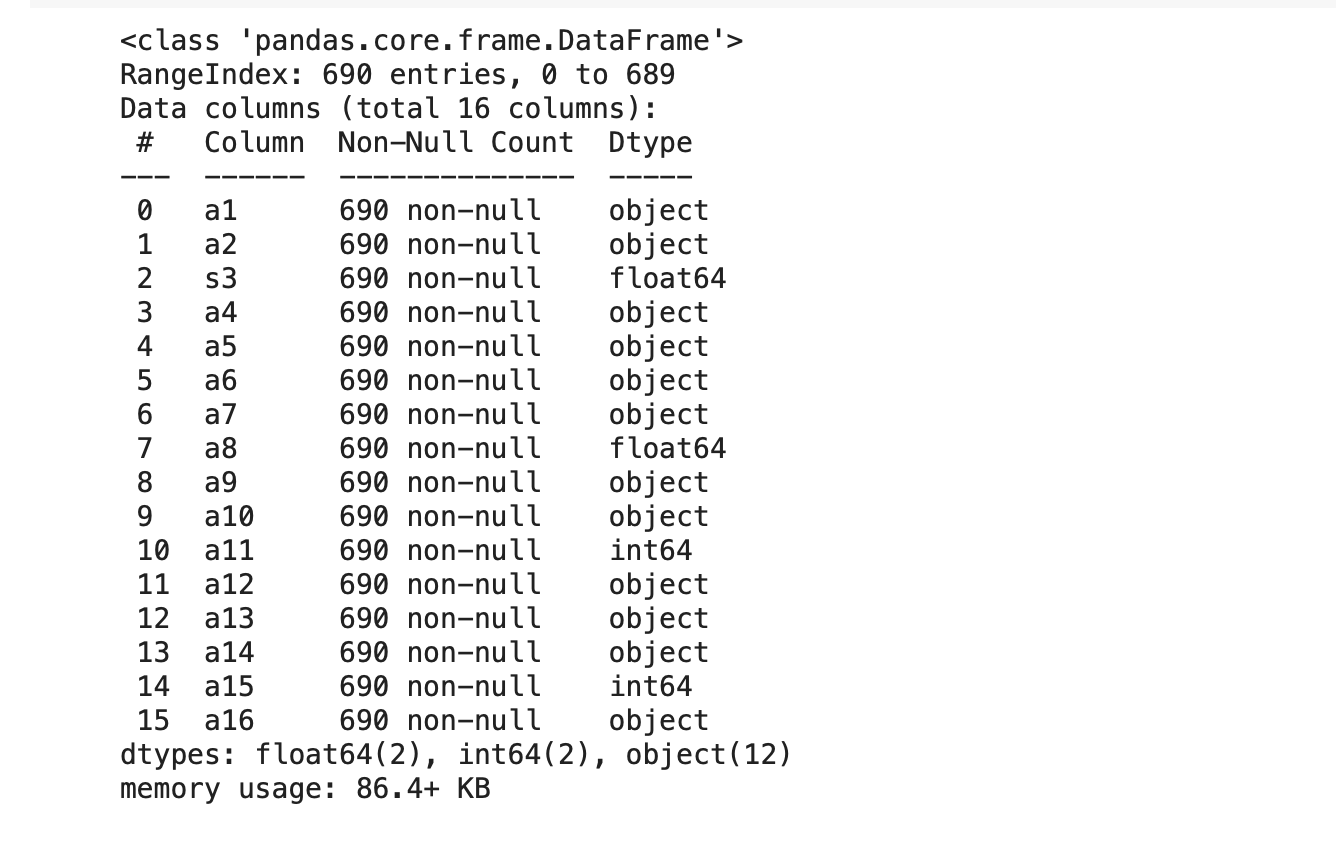
- Addition of hidden layers gives better results indicating the dataset is not linearly separable.

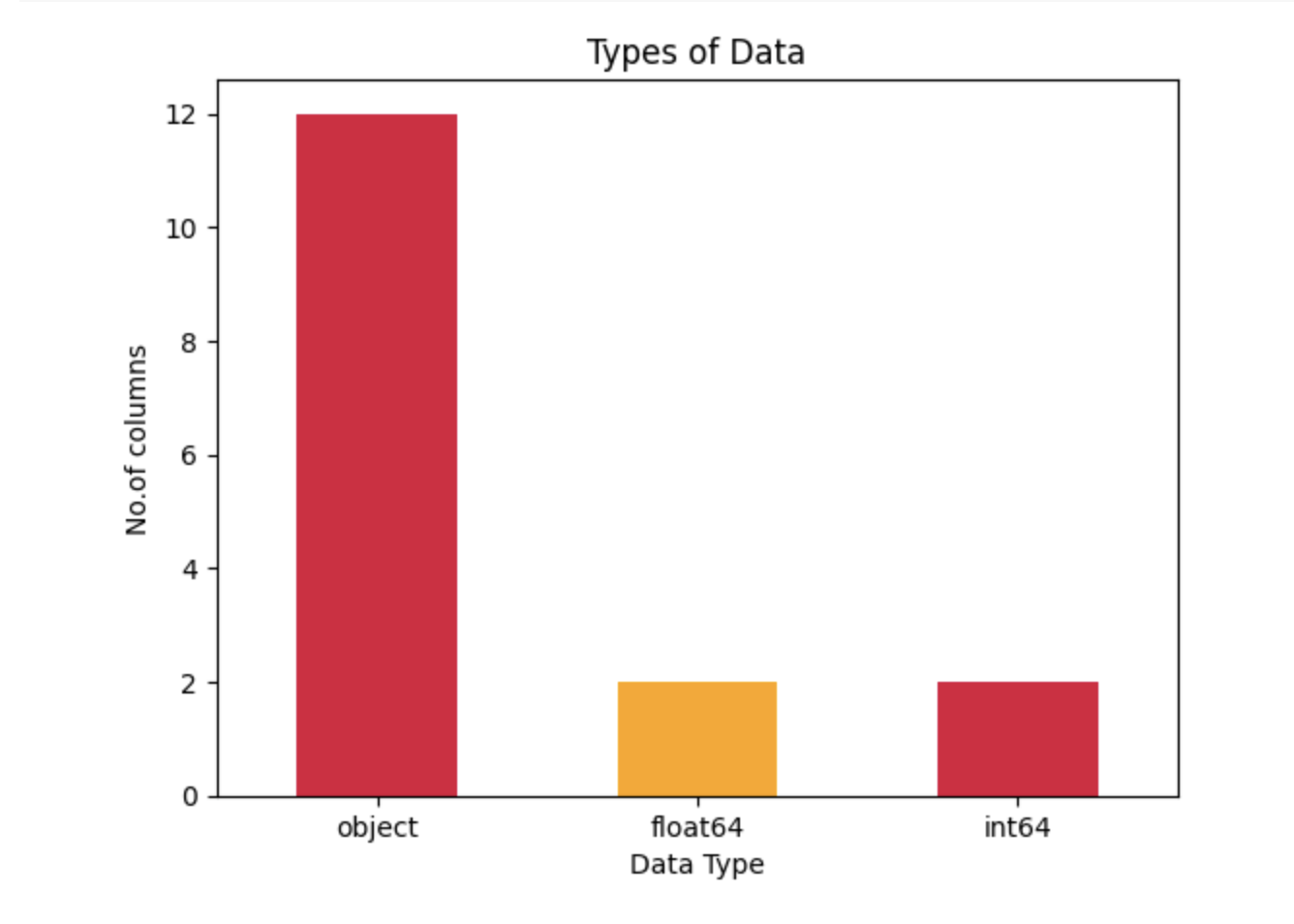
- The MLP is able to learn a non-linear decision boundary compared to Perceptron which is a linear classifier.

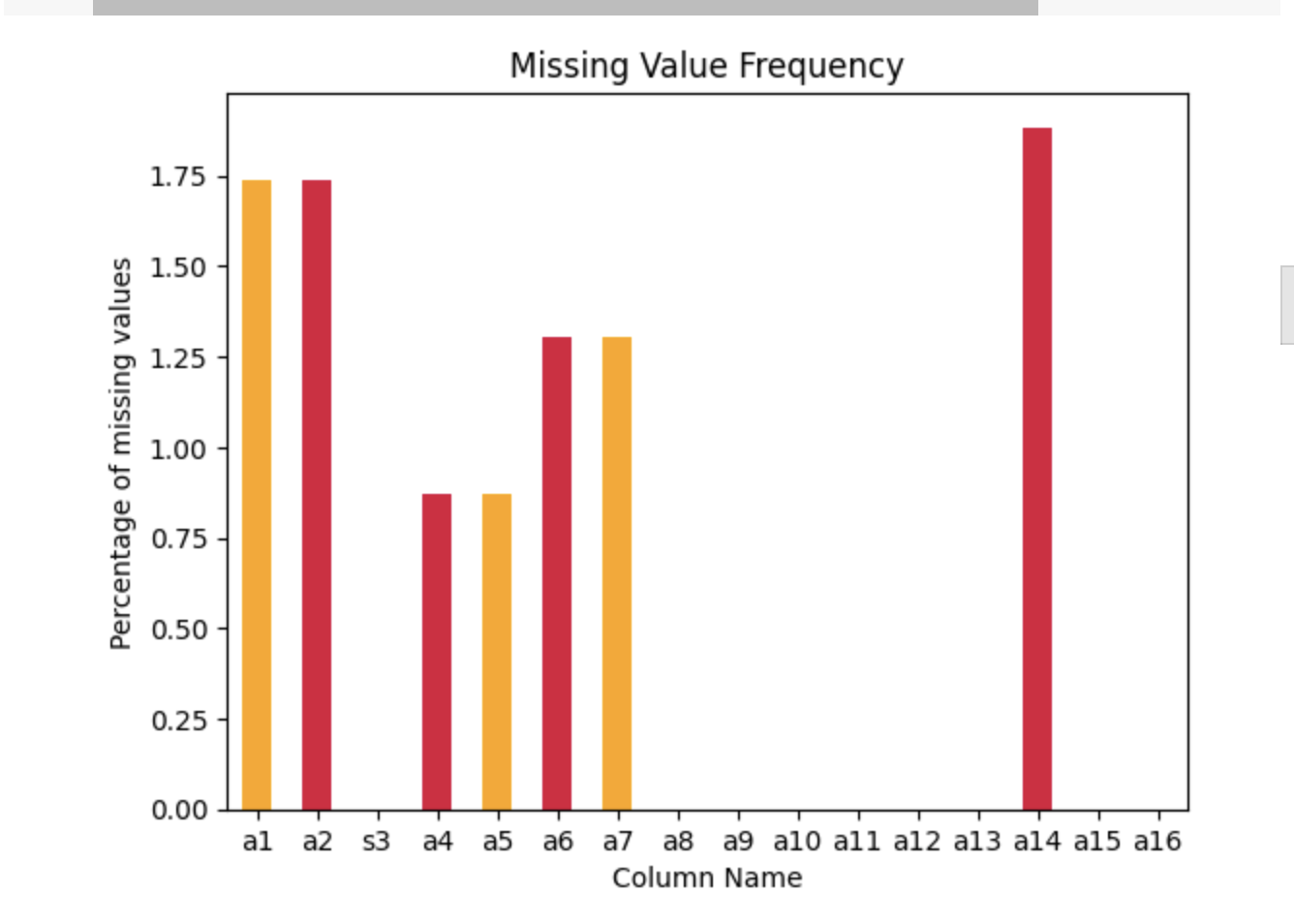
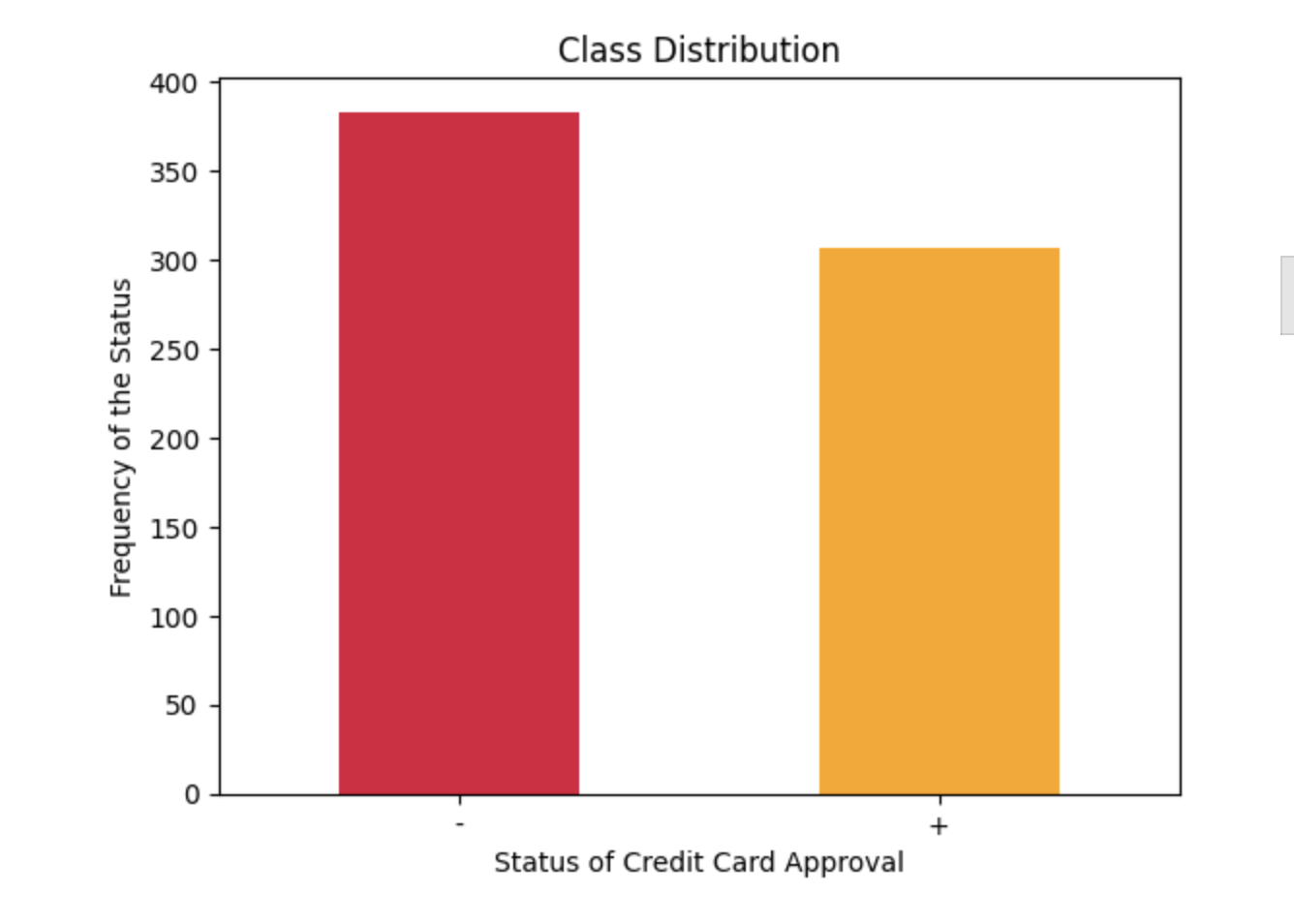
"""

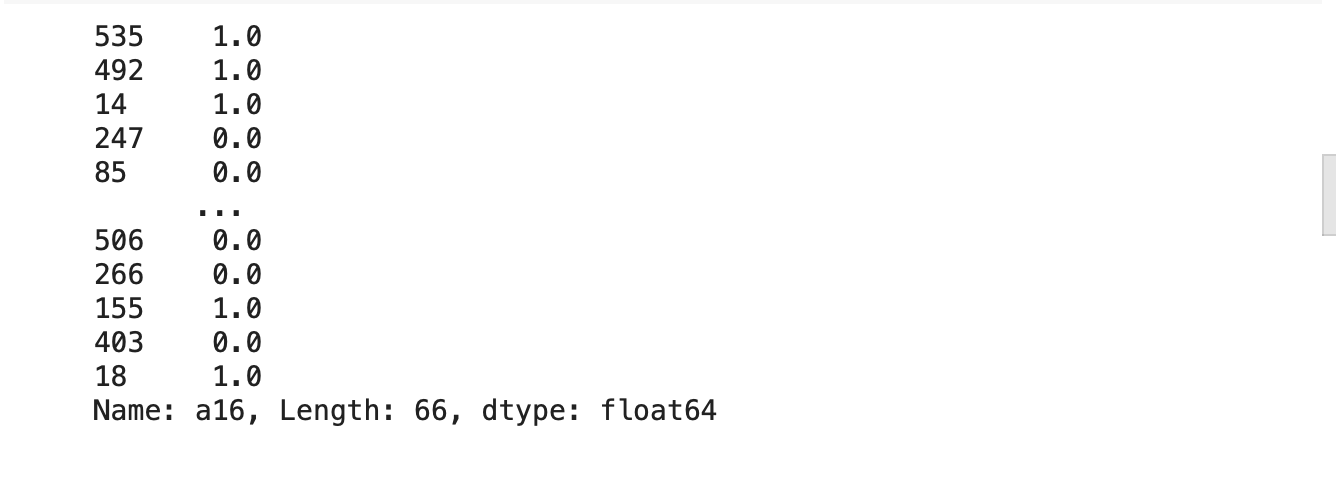
Snippets:

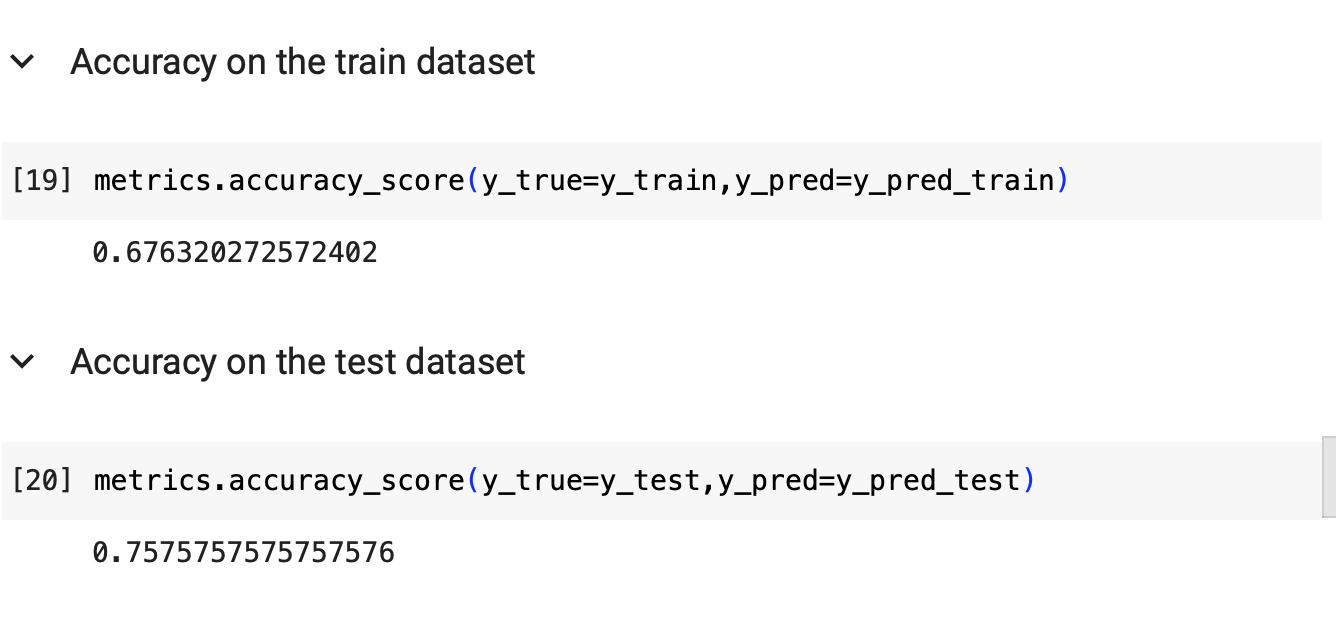
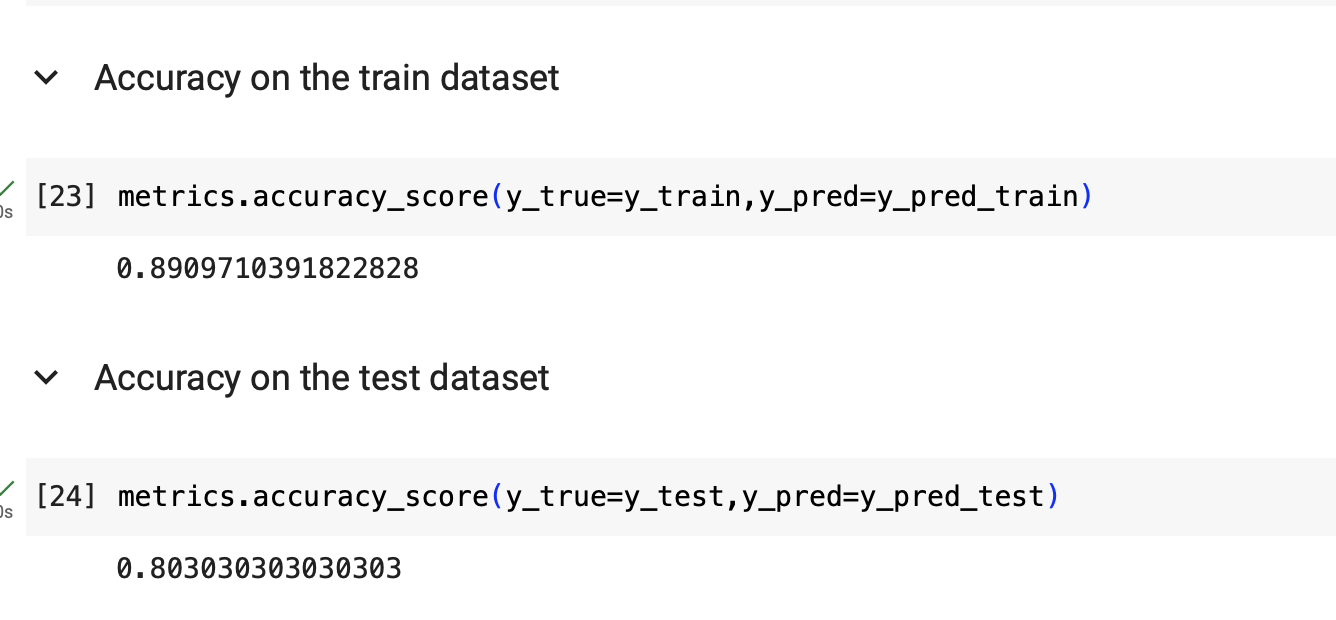
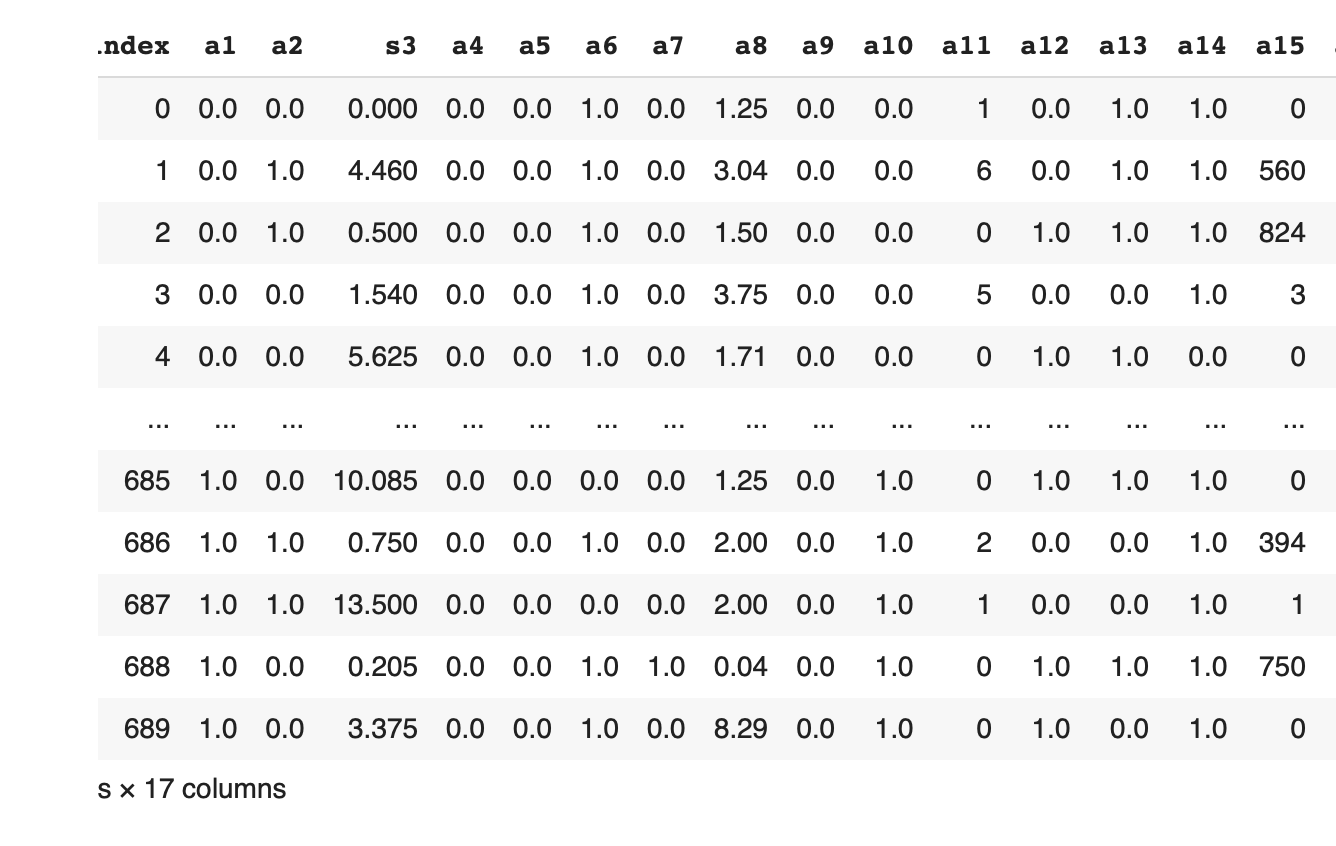
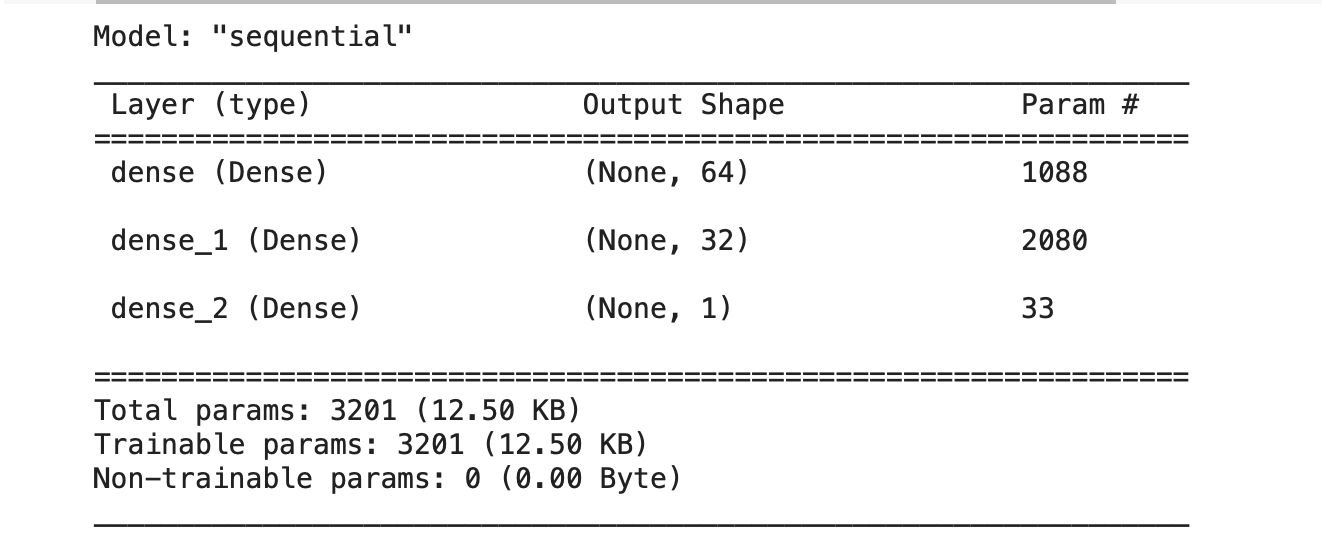


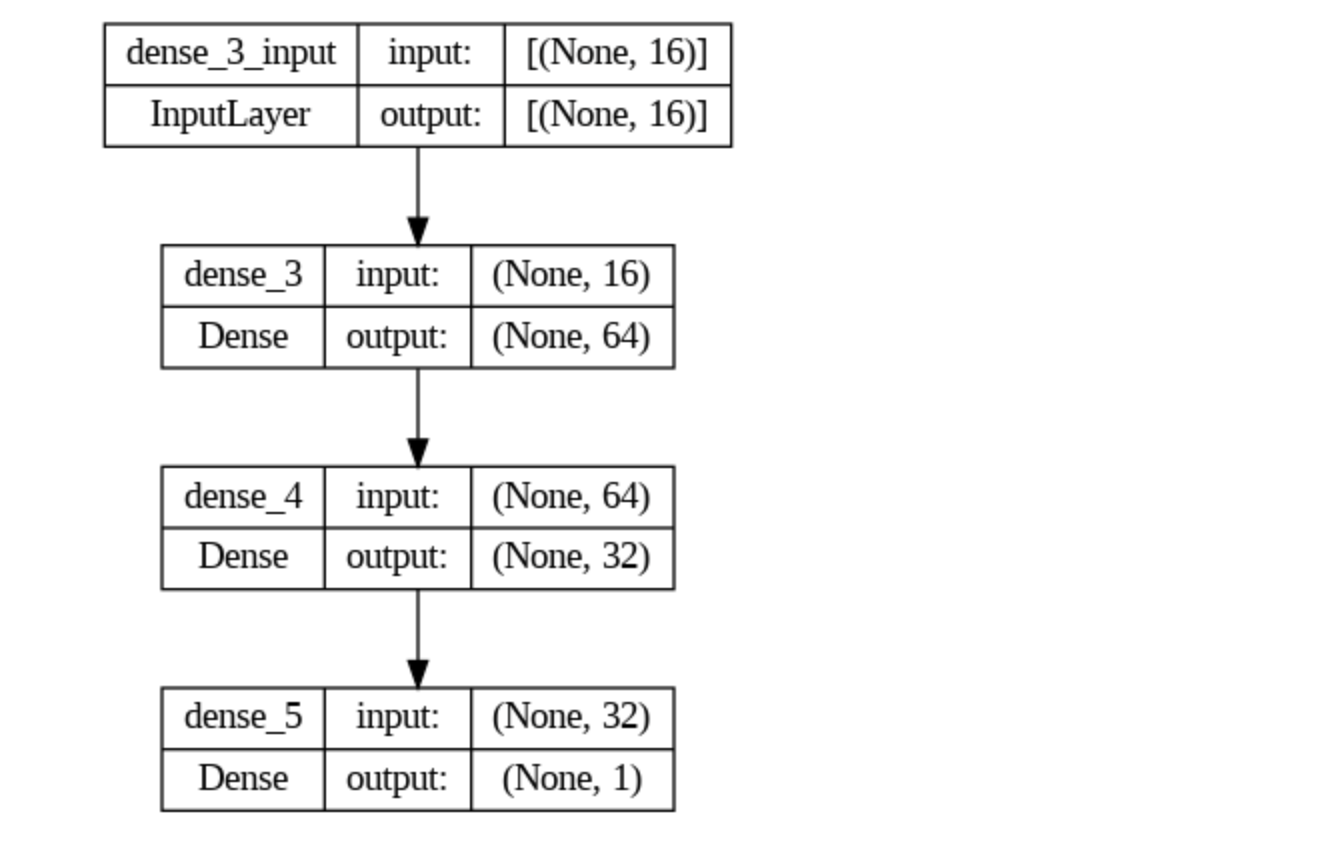




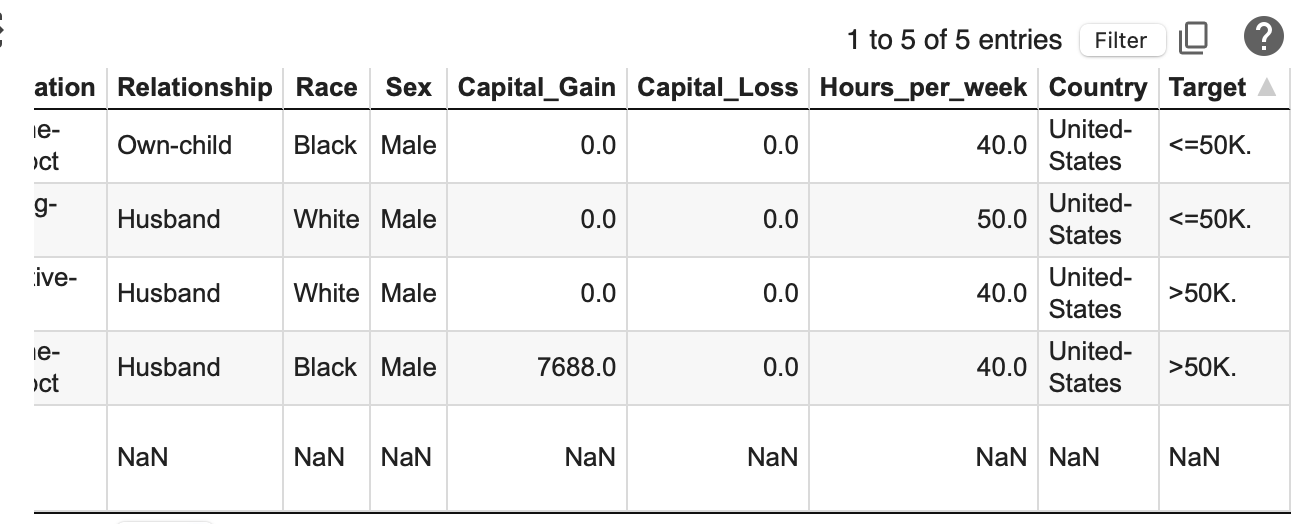
 

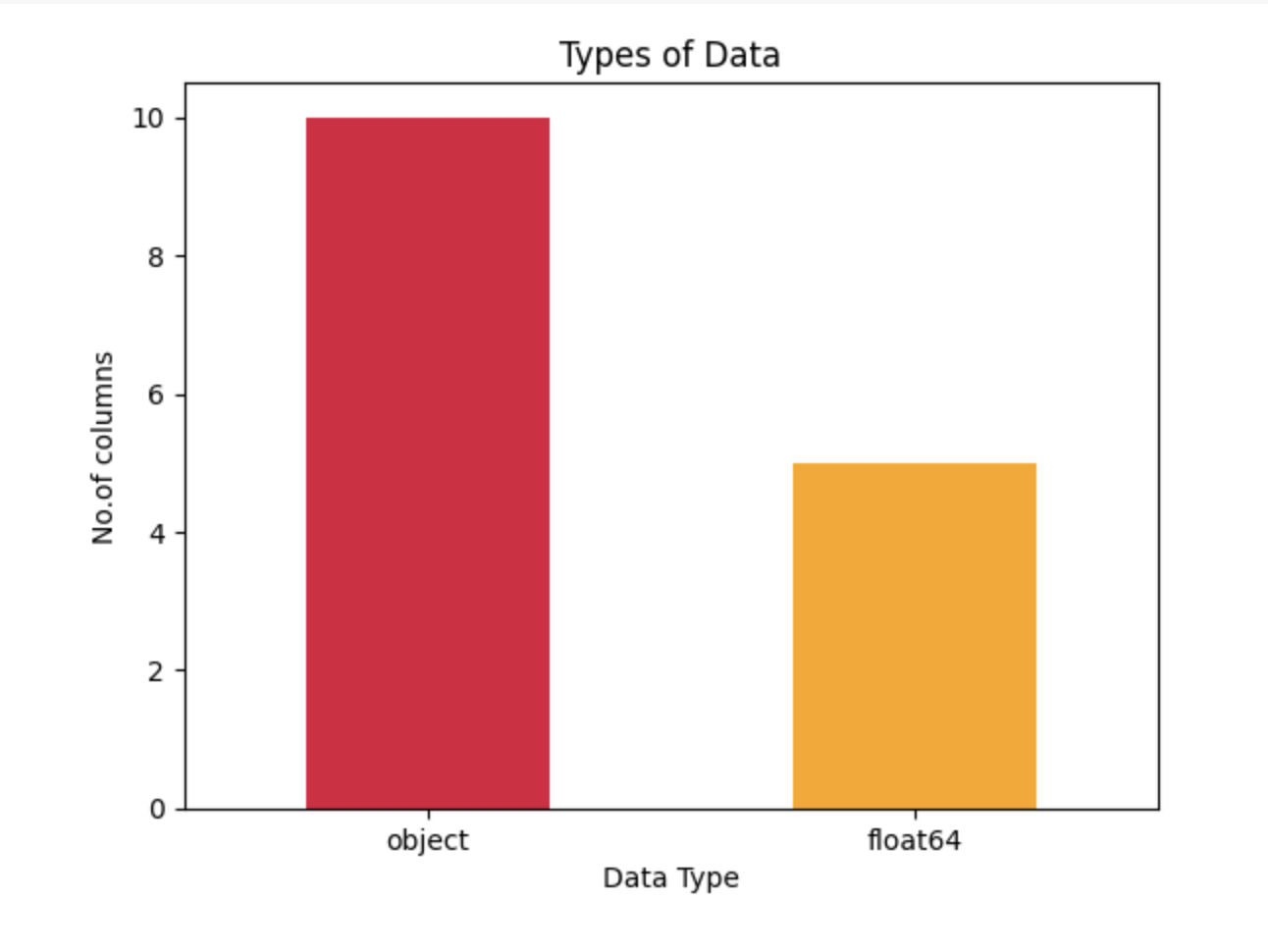


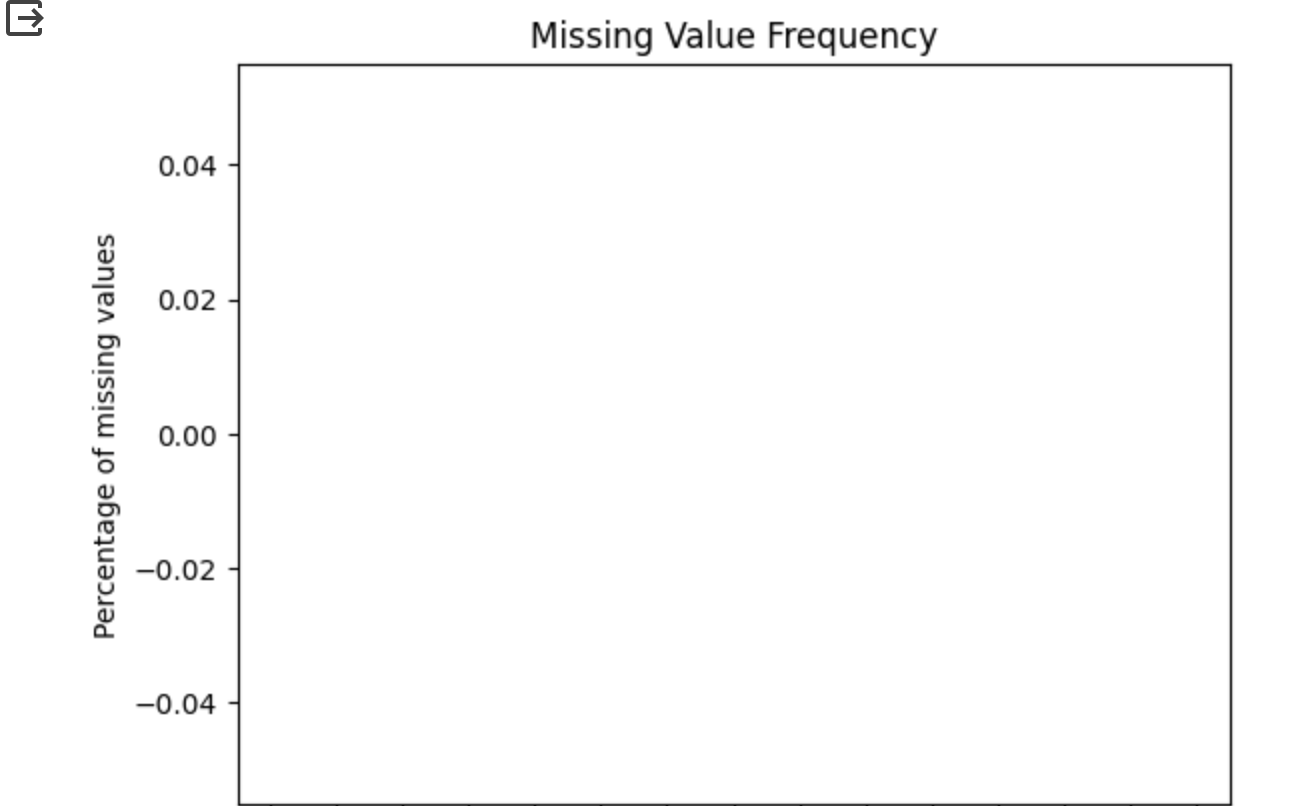
   

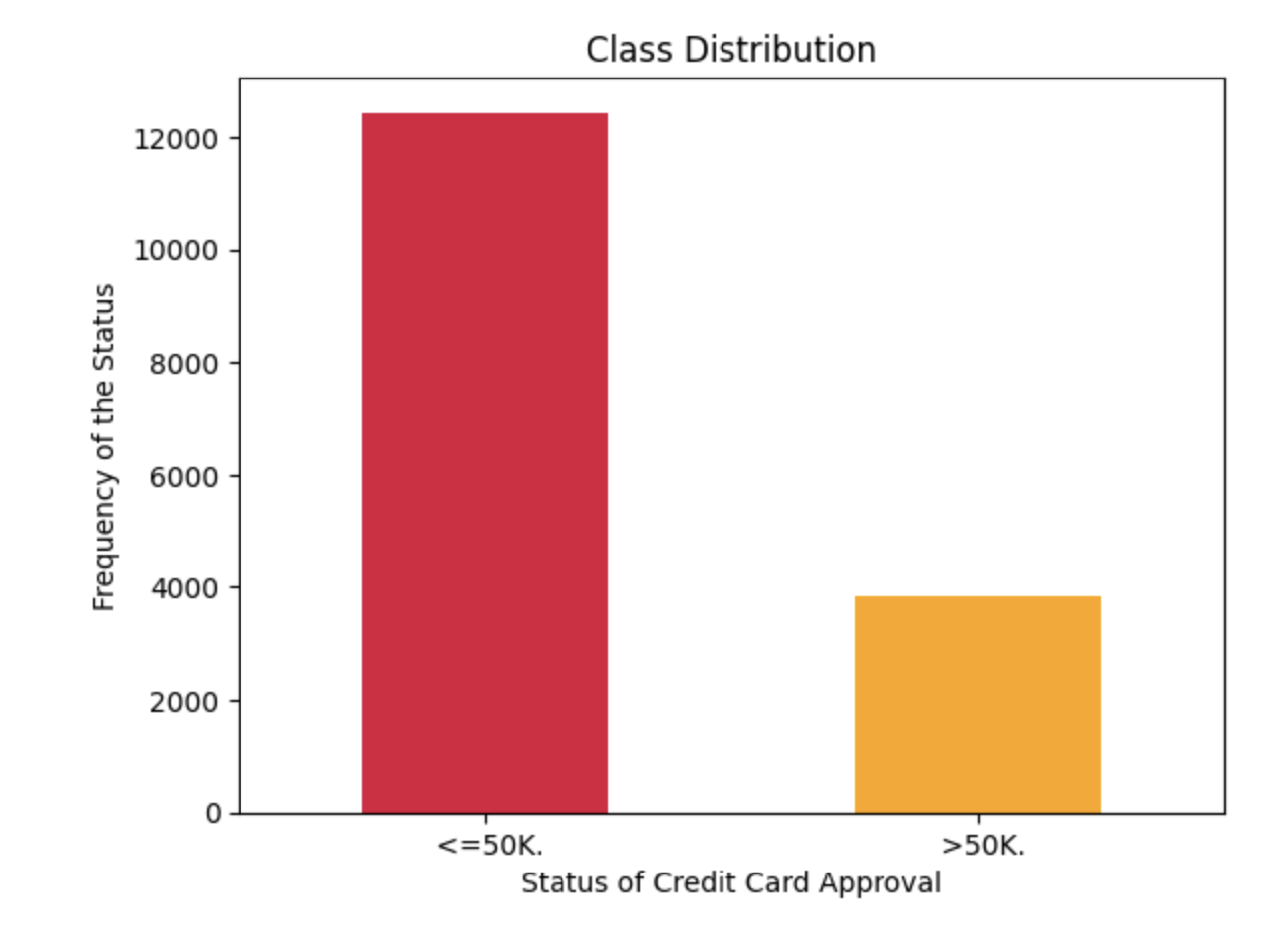


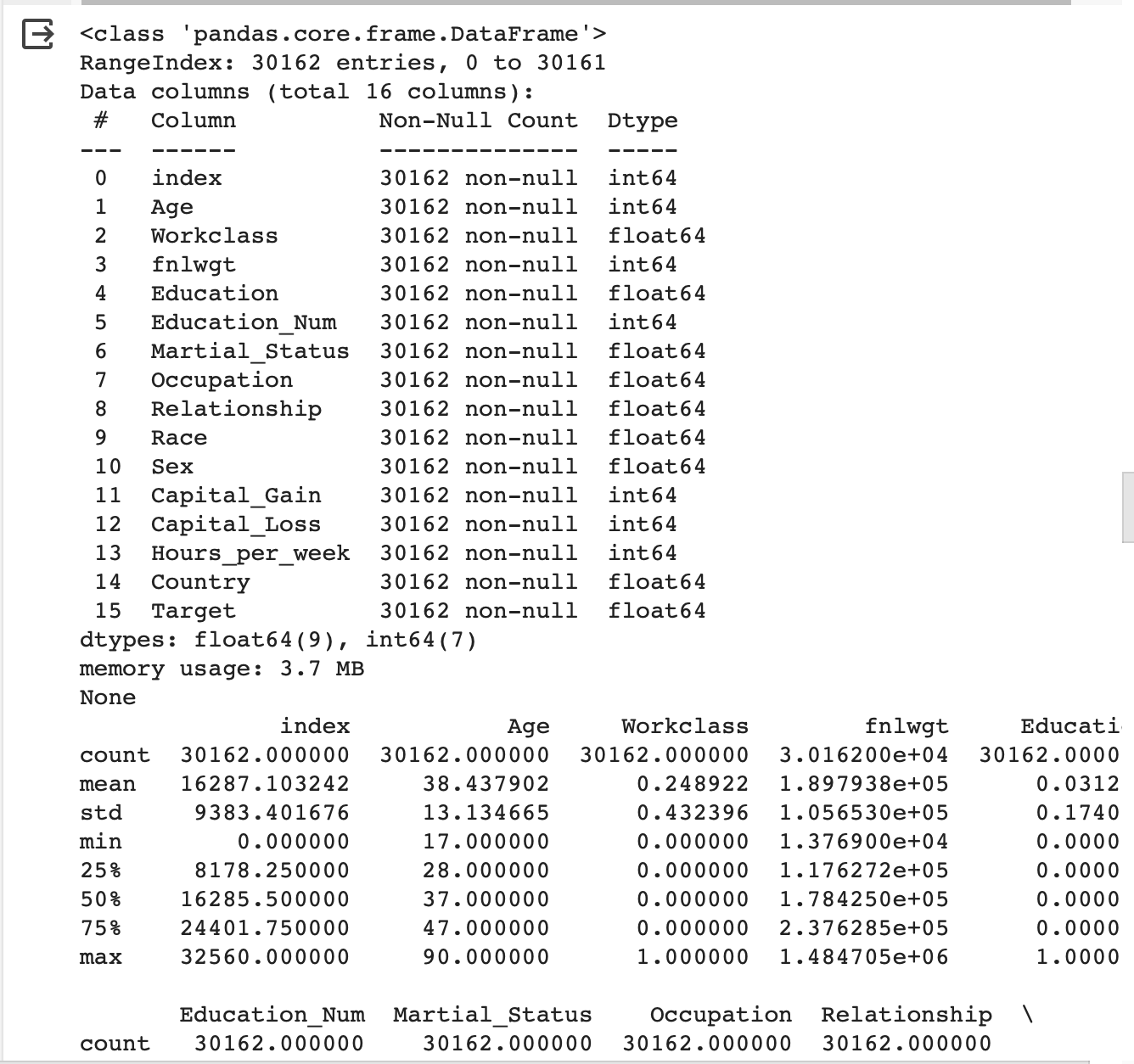
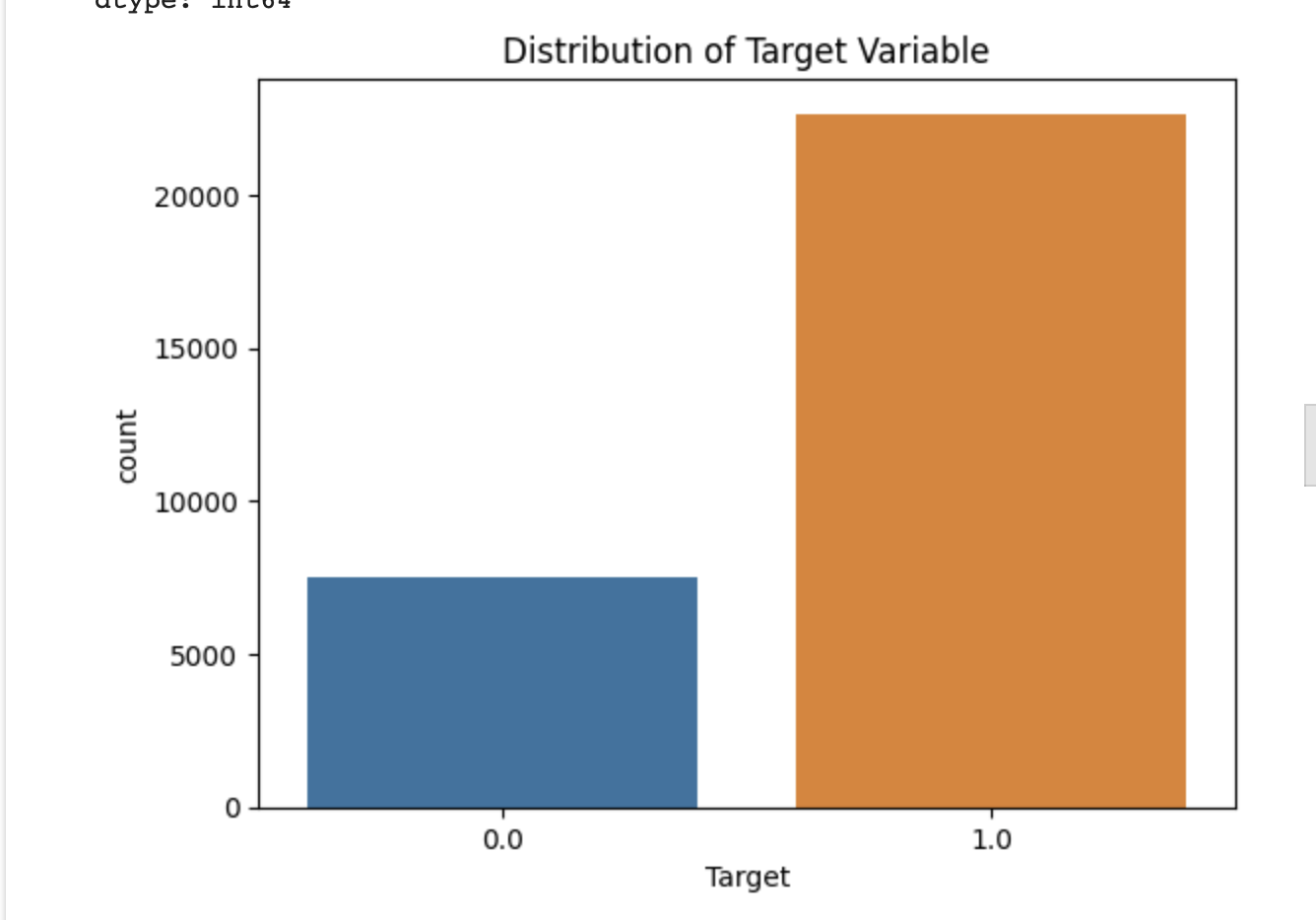
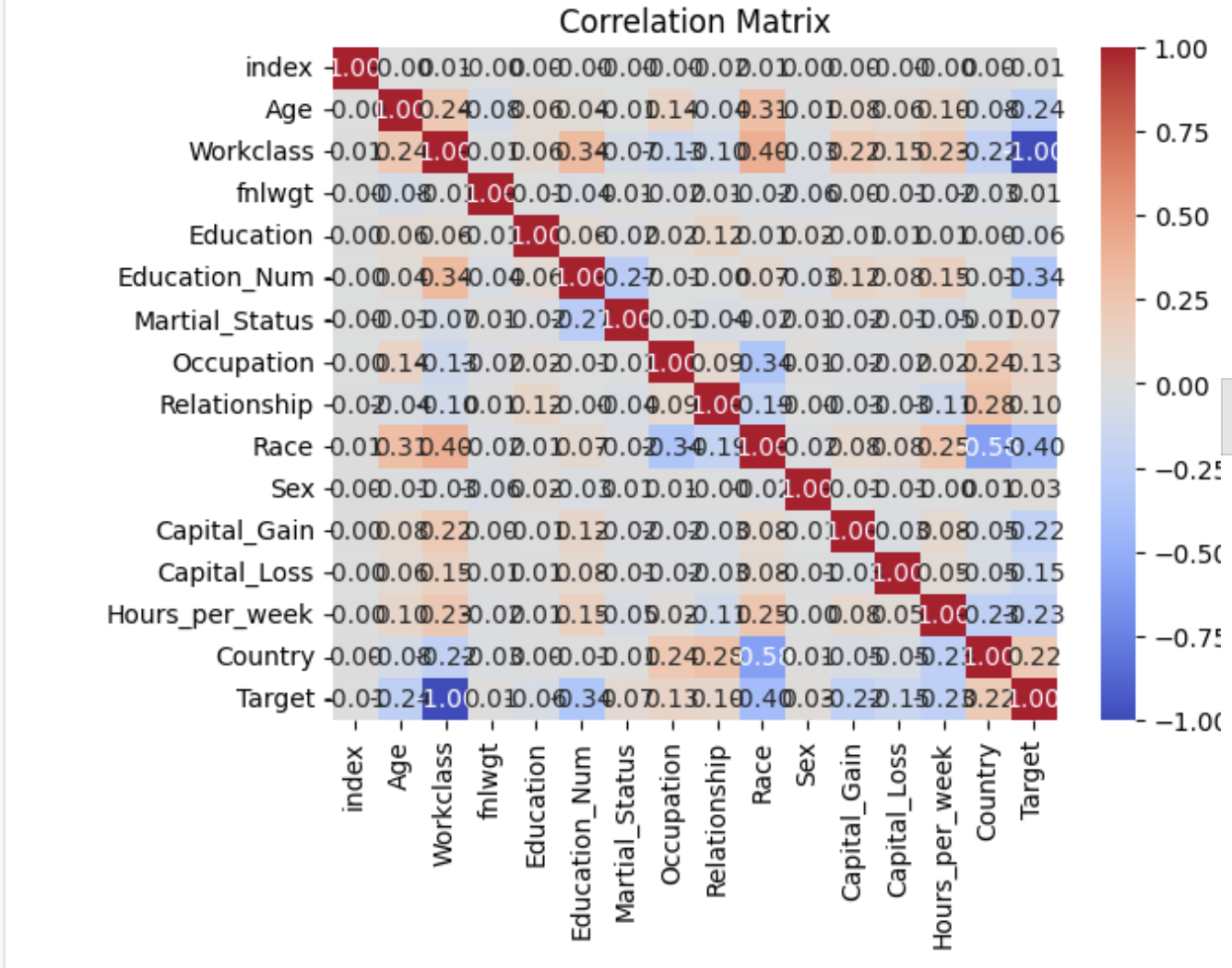
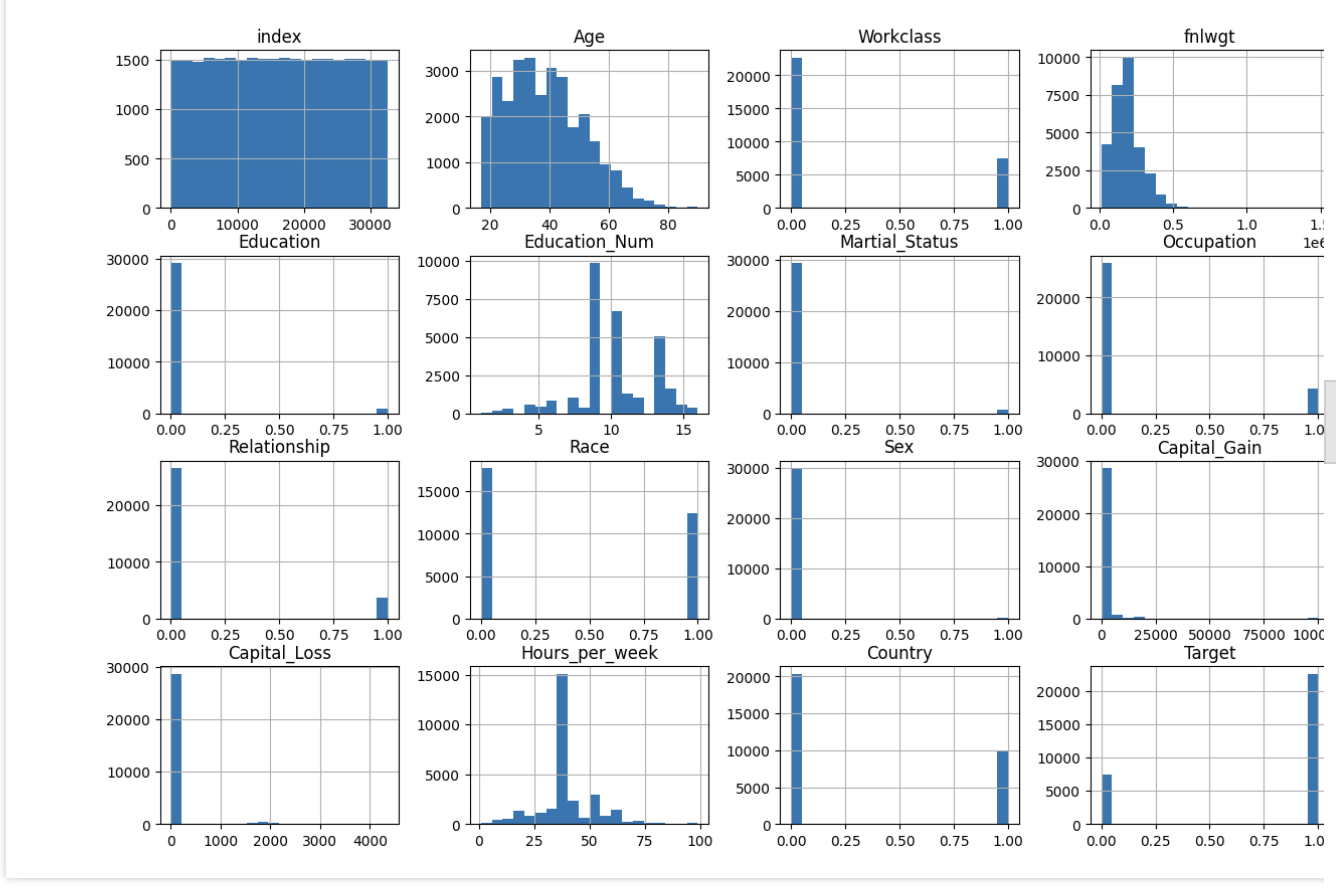
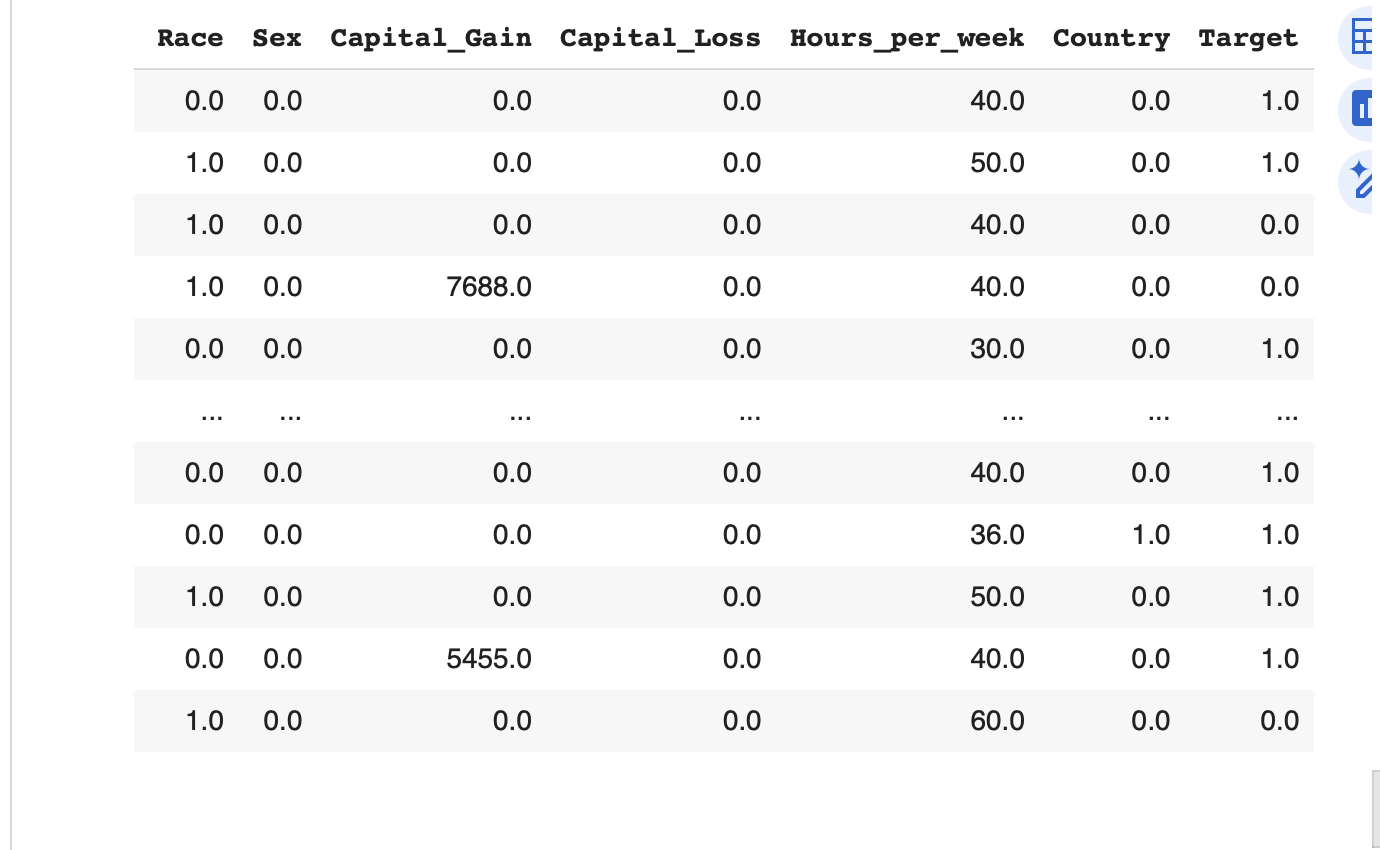
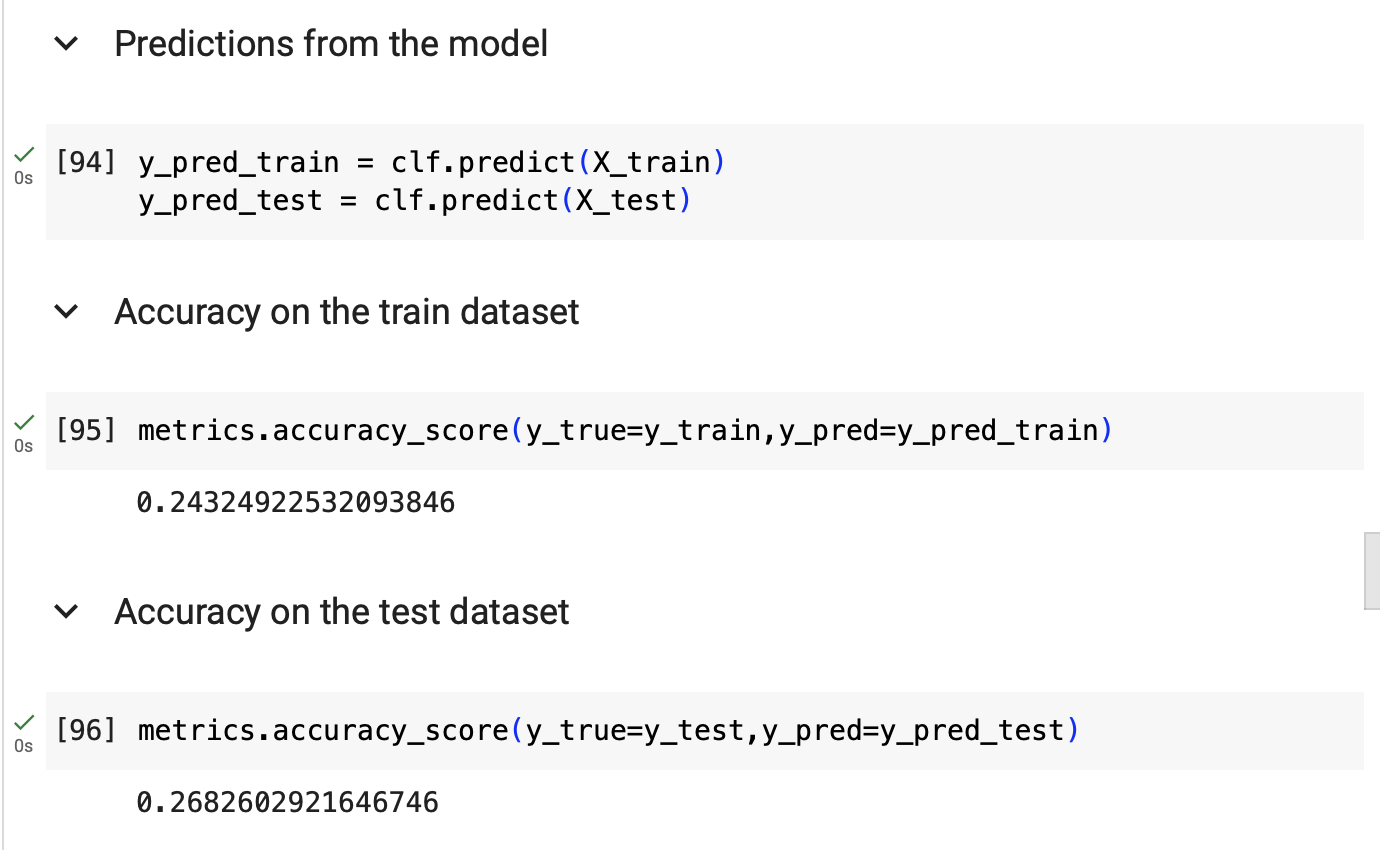
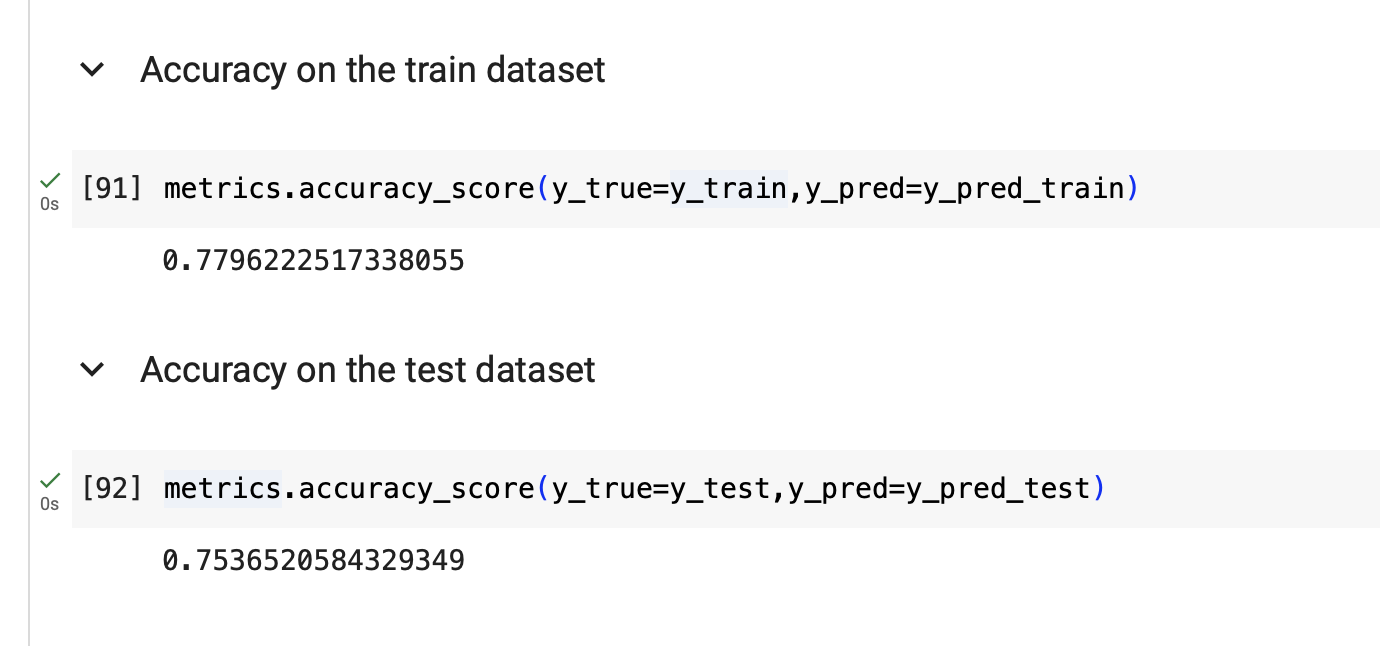
**Adult Census.csv**

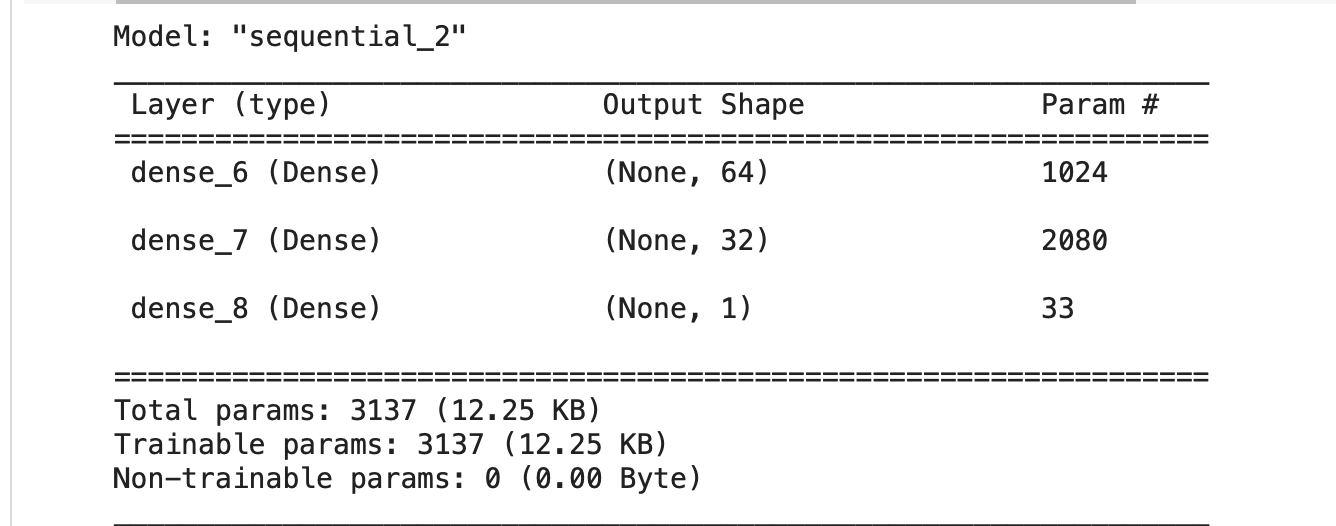
****

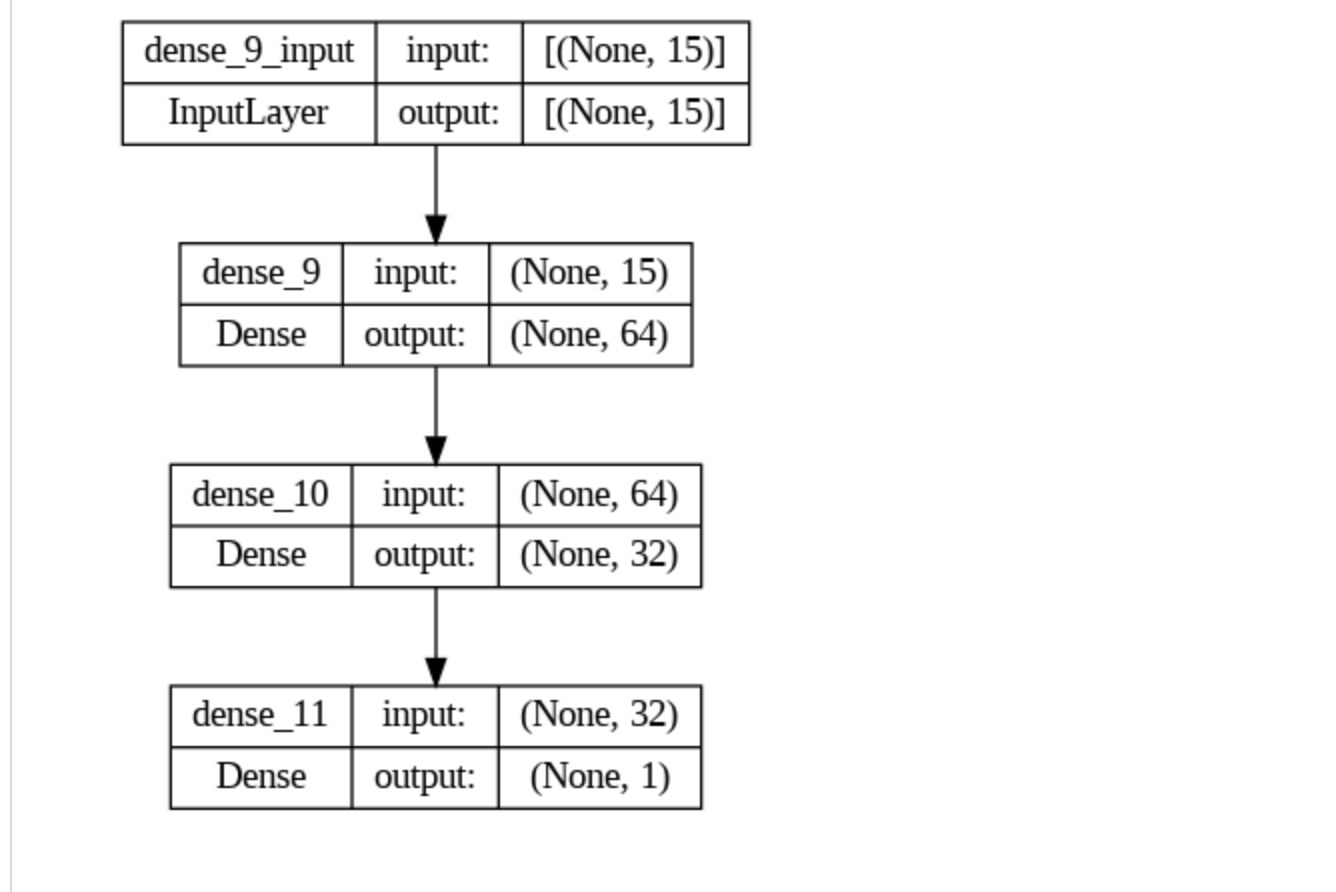
****

****

****

**** ****     



****

1. **Observation and Learning:**
   * The code represents a comprehensive preprocessing pipeline that addresses various aspects of data quality and feature engineering.
   * Adjusting parameters, such as the number of features to select or outlier detection thresholds, is essential and depends on the specific characteristics of the dataset.
   * Understanding the dataset and its context is crucial for making informed decisions during preprocessing.
   * Regularly validating and iterating on preprocessing steps can lead to better model performance.
2. **Conclusion:**

In conclusion, the study of the architecture of Multi-Layer Perceptron (MLP) using Python has provided valuable insights into the intricate workings of neural networks. Through the exploration of various aspects, from basic model construction to advanced configurations, the study aimed to equip participants with a robust understanding of MLPs.