import pandas as pd

import numpy as np

from sklearn import preprocessing

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

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier, export\_graphviz

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.metrics import confusion\_matrix

import matplotlib.pyplot as plt

from sklearn.metrics import classification\_report

from sklearn.model\_selection import GridSearchCV

The code imports the necessary libraries, including pandas for data manipulation, numpy for numerical computations, sklearn for machine learning, and matplotlib and seaborn for visualization.

temp\_df = pd.read\_csv("D:\Data Mining\dataset.csv")

temp\_df.head()

The dataset is loaded into a pandas DataFrame using the pd.read\_csv function.

temp\_df.dtypes

The df.dtypes function is used to check the data types of the columns in the DataFrame.

temp\_df.isnull().sum()

The df.isnull().sum() function is used to check for missing values in the dataset.

temp\_df.Target.unique()

The df.Target.unique() function is used to check the unique values in the Target column, which is the target variable for the logistic regression model.

temp\_df.drop(temp\_df[temp\_df["Target"]=="Enrolled"].index,inplace=True)

The df.drop function is used to drop rows from the DataFrame based on a condition. In this case, rows with a Target value of Enrolled are dropped.

def f(s):

    if s == 'Graduate':

        return 1;

    if s == 'Dropout':

        return 2;

temp\_df.Target = temp\_df.Target.apply(f)

temp\_df.head()

The function is defined to convert the Target column values to integers. The df.Target.apply(f) function is used to apply this function to the Target column.

temp\_df.shape

The df.shape function is used to check the shape of the DataFrame.

correlation\_matrix=temp\_df.corr()

correlation\_matrix

The df.corr() function is used to calculate the correlation matrix of the DataFrame. The sns.heatmap function is used to visualize the correlation matrix.

import seaborn as sns

import matplotlib.pyplot as plt

# Plot correlation matrix heatmap

plt.figure(figsize=(34, 25))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f")

plt.title('Correlation Matrix')

plt.show()

# In[11]:

# Compute correlation with target variable

correlation\_with\_target = temp\_df.corr()['Target']

# Select columns with correlation between -0.1 and 0.1 with the target variable

relevant\_columns = correlation\_with\_target[(correlation\_with\_target < -0.2) | (correlation\_with\_target > 0.2)].index.tolist()

# Print relevant columns

print("Relevant Columns:", relevant\_columns)

# Subset the DataFrame with relevant columns

df\_relevant = temp\_df[relevant\_columns]

The correlation\_with\_target variable is used to calculate the correlation between each column and the Target column. The relevant\_columns variable is used to select columns with a correlation between -0.2 and 0.2 with the Target column.

df\_relevant

correlation\_matrix=df\_relevant.corr()

correlation\_matrix

The df\_relevant DataFrame is created by selecting only the relevant columns from the original DataFrame.

# ploting scatter plot in every column

sns.pairplot(df\_relevant,hue='Target')

The sns.pairplot function is used to visualize the pairwise relationships between the columns in the df\_relevant DataFrame.

from sklearn.neighbors import LocalOutlierFactor

outlier\_columns = list(df\_relevant.columns[:10])

df\_relevant[outlier\_columns[:6] + ["Target"]].boxplot(by="Target", figsize=(20, 10), layout=(1, 6))

df\_relevant[outlier\_columns[6:] + ["Target"]].boxplot(by="Target", figsize=(20, 10), layout=(1, 4))

The LocalOutlierFactor class from the sklearn.neighbors module is used to detect outliers in the dataset. The df\_relevant[outlier\_columns] DataFrame is created by selecting only the outlier columns from the df\_relevant DataFrame.

from scipy.stats import zscore

z\_scores = zscore(df\_relevant)

outliers = df\_relevant[z\_scores > 3]

df\_relevant = df\_relevant[(z\_scores < 3)]

The zscore function from the scipy.stats module is used to calculate the z-scores of the values in the df\_relevant DataFrame. The df\_relevant[z\_scores > 3] DataFrame is created by selecting only the rows with z-scores greater than 3.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df\_relevant = pd.DataFrame(scaler.fit\_transform(df\_relevant), columns=df\_relevant.columns)

The MinMaxScaler class from the sklearn.preprocessing module is used to scale the values in the df\_relevant DataFrame to a range between 0 and 1.

X = df\_relevant.iloc[:,0:10]

Y = df\_relevant.iloc[:,10]

# In[50]:

X.shape, Y.shape

# In[51]:

X\_train, X\_test, Y\_train, Y\_test= train\_test\_split(X,Y,random\_state=1)

X\_train.shape, X\_test.shape, Y\_train.shape, Y\_test.shape

The dataset is split into training and testing sets using the train\_test\_split function from the sklearn.model\_selection module.

#Logistic Regression

logistic\_regression = LogisticRegression(max\_iter=1000000)

logistic\_regression.fit(X\_train,Y\_train)

logistic\_regression\_train\_prediction = logistic\_regression.predict(X\_train)

logistic\_regression\_test\_prediction = logistic\_regression.predict(X\_test)

cf\_matrix=confusion\_matrix(Y\_test, logistic\_regression\_test\_prediction)

print(cf\_matrix)

# In[53]:

import matplotlib.pyplot as plt

import seaborn as sns

# Plot confusion matrix

plt.figure(figsize=(10, 4))

sns.heatmap(cf\_matrix, annot=True, cmap='Blues', fmt='g', cbar=False)

plt.xlabel('Predicted labels')

plt.ylabel('True labels')

plt.title('Confusion Matrix')

plt.show()

# In[54]:

print(classification\_report(Y\_test, logistic\_regression\_test\_prediction))

In the above code, logistic regression model is trained on default parameters and then the results are shown in the form of confusion matrix.

grid\_logistic\_regression = LogisticRegression(C=2, max\_iter=500, solver='saga')

grid\_logistic\_regression.fit(X\_train,Y\_train)

grid\_logistic\_regression\_train\_prediction = grid\_logistic\_regression.predict(X\_train)

grid\_logistic\_regression\_test\_prediction = grid\_logistic\_regression.predict(X\_test)

cf\_matrix=confusion\_matrix(Y\_test, grid\_logistic\_regression\_test\_prediction)

print(cf\_matrix)

# In[56]:

# Plot confusion matrix

plt.figure(figsize=(10, 4))

sns.heatmap(cf\_matrix, annot=True, cmap='Blues', fmt='g', cbar=False)

plt.xlabel('Predicted labels')

plt.ylabel('True labels')

plt.title('Confusion Matrix')

plt.show()

# In[57]:

print(classification\_report(Y\_test, grid\_logistic\_regression\_test\_prediction))

In the above code, I have tuned several parameters in order to increase the accuracy on training dataset for logistic regression. To know that which specific parameters I have tuned, you can refer our report and ppt.

The same task is repeated for Support Vector Machine and Decision Tree, i.e., firstly I have trained the model with default parameters and after that using GridSearchCV, I have fine-tuned the models by tweaking several parameters, everything is mentioned in the report and ppt. Kindly refer it

Overall, this code performs exploratory data analysis, feature selection, outlier detection, hyperparameter tuning, and model evaluation on a given dataset. The logistic regression model is trained on the relevant features, and its performance is evaluated using various metrics and visualizations. The hyperparameters of the logistic regression model are tuned using Grid Search, and the best model is selected based on its performance on the testing set. The performance of the best model is visualized using various plots and metrics.