VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF INFORMATION TECHNOLOGY

FACULTY OF INFORMATION SYSTEMS

PROJECT REPORT

DATA MINING

PREDICT THE INCIDENCE OF DIABETES

Instructors:	Ph.D Cao Thi Nhan
	MSc. Vu Minh Sang
Class:	IS252.N21.HTCL

Members:

Le Thi Kieu Lam(leader)	21522275
Tran Phan Thanh Thao	21522610
Nguyen Quoc Huy	21522158
Do Dinh Dang Khoa	21522218

Ho Chi Minh City, June 5, 2024

ACKNOWLEDGEMENT

First of all, we would like to express our sincere thanks to all of the teachers and students of the University of Information Technology - Vietnam National University Ho Chi Minh City and the teachers of the Information System Faculty for helping us to have the basic knowledge that is the basis for implementing this topic.

In particular, the authors would like to send our deepest gratitude and gratitude to Ph.D. Cao Thi Nhan – theoretical instructor of Data Mining, and MSc. Vu Minh Sang – practical instructor of Data Mining. They have guided directly, corrected and contributed many valuable comments to help the authors successfully complete the course report.

During the project implementation period, the authors applied the accumulated knowledge and combined it with learning and researching new knowledge. Since then, the authors make the most of what has been collected to complete a report on the best project. However, during the implementation process, the author group cannot avoid shortcomings. Therefore, the author group is looking forward to receiving suggestions from instructors to improve the knowledge that the authors have learned and is the luggage for the author group to carry out other topics in the future.

Sincerely thank you!

Ho Chi Minh City, June 2024

Le Thi Kieu Lam

Tran Phan Thanh Thao

Nguyen Quoc Huy

Do Dinh Dang Khoa

LECTURE COMMENTS

TABLE OF CONTENTS

CHAPTER I: INTRODUCTION

1. Dataset

Machine Learning, with its superior capabilities in classification and prediction, has revolutionized numerous industries. A prime example is its application in weather forecasting, which has brought tangible benefits to daily life. Building on these successes, our team aims to harness the potential of Machine Learning to address another pressing issue in modern society: predicting the prevalence of diabetes.

Equipped with a diverse and comprehensive dataset on risk factors associated with diabetes, we are confident that applying advanced Machine Learning algorithms will enable the development of a highly accurate predictive model. This will not only open up new avenues in diabetes research and prevention but also contribute to the advancement of the field of preventive medicine as a whole.

Dataset Link:

https://www.kaggle.com/datasets/prosperchuks/health-dataset?select=diabetes_data.csv

2. Reason for choosing the topic

The topic "Predicting Diabetes Prevalence Using Machine Learning Algorithms Based on the diabetes_data.csv Dataset" is chosen due to the urgency of the challenges posed by diabetes globally and in Vietnam specifically. Diabetes affects hundreds of millions worldwide, and its prevalence in Vietnam is increasing, particularly among young people and urban populations, placing a significant burden on the healthcare system and society. Early detection of diabetes risk is seen as key to timely intervention, lifestyle changes, and effective treatment, reducing complications and improving patients' quality of life.

The feasibility of this project is ensured by the availability of the "diabetes_data.csv" dataset, which provides diverse information on risk factors associated with diabetes. Coupled with the rapid advancement of Machine Learning (ML) algorithms, which have proven effective in analyzing medical data and predicting diseases, the development of a diabetes prediction model is entirely feasible. Furthermore, the support from the data science community and open-source ML libraries will be valuable resources and tools for this research.

The diabetes prediction model is expected to have significant practical applications. It can assist healthcare professionals in assessing risk, making diagnostic decisions, and selecting appropriate treatment plans for individual patients. Additionally, developing applications based on the predictive model will enable users to self-assess their risk, raise awareness, and encourage healthy lifestyle changes. Early prediction and intervention can not only reduce the prevalence of diabetes and its complications but also contribute to improving public health and reducing the burden on the healthcare system.

This research is expected to uncover new relationships between risk factors and diabetes, contributing to the understanding of the disease and its development mechanisms. Simultaneously, the application and improvement of ML algorithms will create a more accurate and effective predictive model compared to traditional methods. The research findings will be shared with the scientific community, promoting the development of ML-based disease prediction and bringing practical benefits to the community and society.

3. Description of the dataset

The is a clean dataset of 70,692 survey responses to the CDC's BRFSS2015. It has an equal 50-50 split of respondents with no diabetes and with either prediabetes or diabetes. The target variable Diabetes_binary has 2 classes. 0 is for no diabetes, and 1 is for prediabetes or diabetes. This dataset has 21 feature variables and is balanced.

Diabetes is among the most prevalent chronic diseases in the United States, impacting millions of Americans each year and exerting a significant financial burden on the economy. Diabetes is a serious chronic disease in which individuals lose the ability to effectively regulate levels of glucose in the blood, and can lead to reduced quality of life and life expectancy. After different foods are broken down into sugars during digestion, the sugars are then released into the bloodstream. This signals the pancreas to release insulin. Insulin helps enable cells within the body to use those sugars in the bloodstream for energy. Diabetes is generally characterized by either the body not making enough insulin or being unable to use the insulin that is made as effectively as needed.

Complications like heart disease, vision loss, lower-limb amputation, and kidney disease are associated with chronically high levels of sugar remaining in the bloodstream for those with diabetes. While there is no cure for diabetes, strategies like losing weight, eating healthily, being active, and receiving medical treatments can mitigate the harms of this disease in many patients. Early diagnosis can lead to lifestyle changes and more effective treatment, making predictive models for diabetes risk important tools for public and public health officials.

The scale of this problem is also important to recognize. The Centers for Disease Control and Prevention has indicated that as of 2018, 34.2 million Americans have diabetes and 88 million have prediabetes. Furthermore, the CDC estimates that 1 in 5 diabetics, and roughly 8 in 10 prediabetics are unaware of their risk. While there are different types of diabetes, type II diabetes is the most common form and its prevalence varies by age, education, income, location, race, and other social determinants of health. Much of the burden of the disease falls on those of lower socioeconomic status as well. Diabetes also places a massive burden on the economy, with diagnosed diabetes costs of roughly $327 billion dollars and total costs with undiagnosed diabetes and prediabetes approaching $400 billion dollars annually.

Dataset includes:

- Attributes: 18 (columns)

- Records: 70692

Data description table

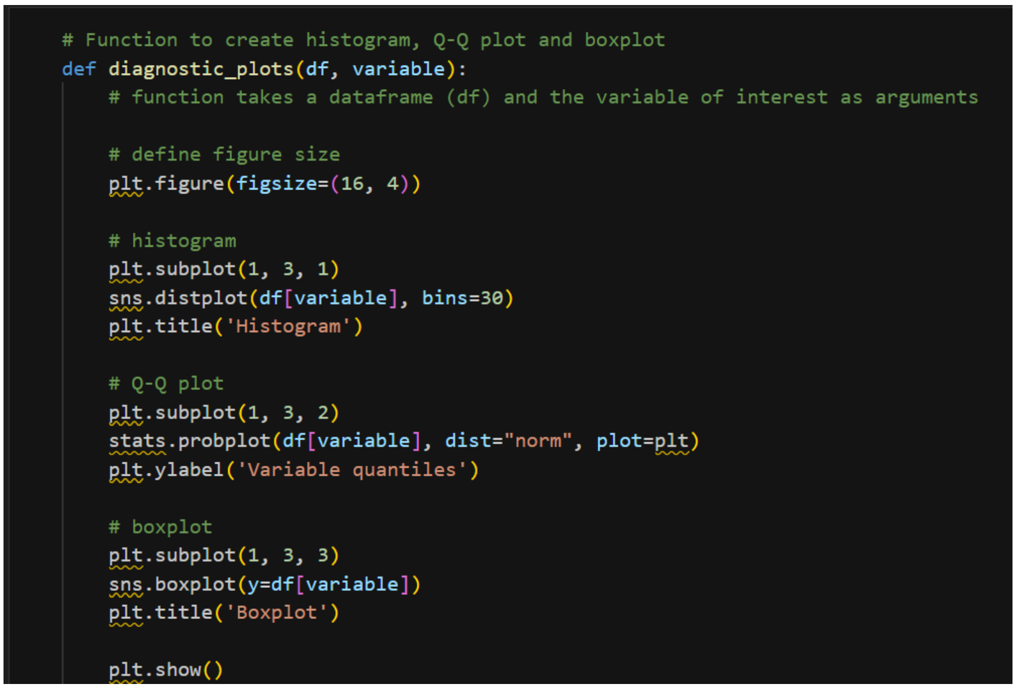
STT	FIELD NAME	DESCRIPTION	TYPE
1	Age	The age of the individual.	INT
2	Sex	The biological sex of the individual (1:male; 0: female).	BIT
3	HighChol	Indicates whether the individual has high cholesterol (0=no high cholesterol; 1 = high cholesterol).	BIT
4	CholCheck	Indicates whether the individual has had their cholesterol checked(0 = no cholesterol check in 5 years; 1 = yes cholesterol check in 5 years).	BIT
5	BMI	The Body Mass Index of the individual, a measure of body fat based on height and weight.	INT
6	Smoker	Have you smoked at least 100 cigarettes in your entire life? [Note: 5 packs = 100 cigarettes](0 = no 1 = yes).	BIT
7	HeartDiseaseorAttack	Coronary heart disease (CHD) or myocardial infarction (MI) (0 = no 1 = yes).	BIT
8	PhysActivity	Physical activity in past 30 days - not including job (0 = no 1 = yes).	BIT
9	Fruits	Consume Fruit 1 or more times per day (0 = no; 1 = yes).	BIT
10	Veggies	Consume Vegetables 1 or more times per day (0 = no; 1 = yes).	BIT
11	HvyAlcoholConsump	Indicates whether the individual engages in heavy alcohol consumption. Adult men >=14 drinks per week and adult women>=7 drinks per week (0 = no 1 = yes).	BIT
12	GenHlth	Would you say that in general your health is: scale 1-5 1 = excellent 2 = very good 3 = good 4 = fair 5 = poor.	INT

13	MentHlth	Days of poor mental health scale 1-30 days.	INT
14	PhysHlth	Physical illness or injury days in past 30 days scale 1-30.	INT
15	DiffWalk	Do you have serious difficulty walking or climbing stairs?(0 = no 1 = yes).	BIT
16	Stroke	If you ever had a stroke(0 = no, 1 = yes).	BIT
17	HighBP	Indicates whether the individual has high blood pressure(0 = no high, BP 1 = high BP).	BIT
18	Diabetes	Indicates whether the individual has diabetes(0 = no diabetes, 1 = diabetes).	BIT

CHAPTER II: DATA VISUALIZATION

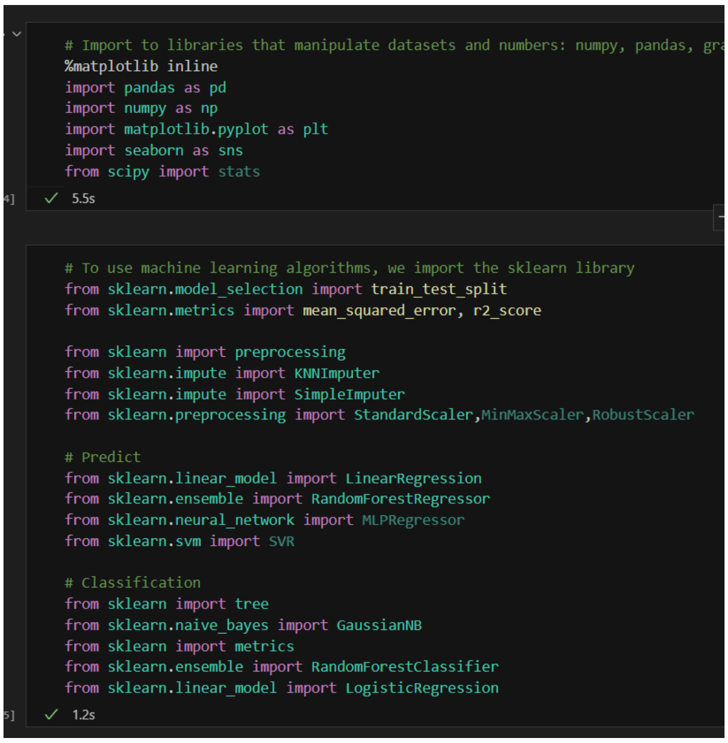
1. Import libraries and dataset

a. Define function



```
# Function to create histogram, Q-Q plot and boxplot
def diagnostic_plots(df, variable):
    # function takes a dataframe (df) and the variable of interest as arguments

    # define figure size
    plt.figure(figsize=(16, 4))

    # histogram
    plt.subplot(1, 3, 1)
    sns.distplot(df[variable], bins=30)
    plt.title('Histogram')

    # Q-Q plot
    plt.subplot(1, 3, 2)
    stats.probplot(df[variable], dist="norm", plot=plt)
    plt.ylabel('Variable quantiles')

    # boxplot
    plt.subplot(1, 3, 3)
    sns.boxplot(y=df[variable])
    plt.title('Boxplot')

    plt.show()
```

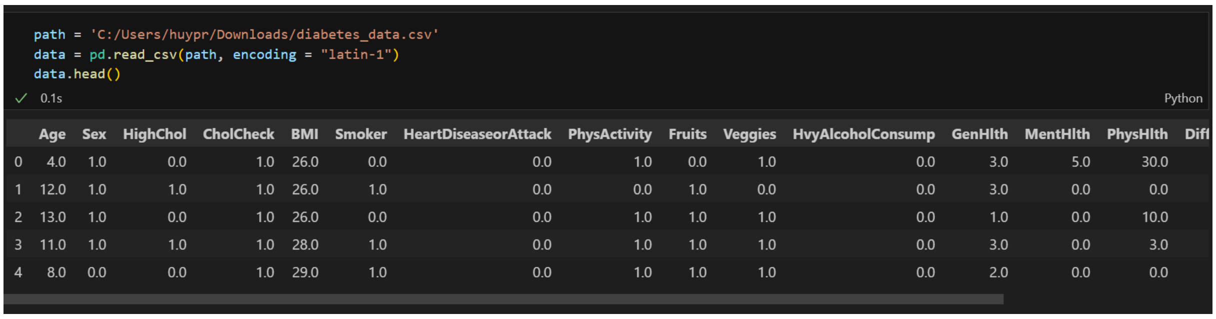
b. Import necessary libraries



```
# Import to libraries that manipulate datasets and numbers: numpy, pandas, gra
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```
✓ 5.5s

```
# To use machine learning algorithms, we import the sklearn library
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

from sklearn import preprocessing
from sklearn.impute import KNNImputer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler,MinMaxScaler,RobustScaler

# Predict
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.svm import SVR

# Classification
from sklearn import tree
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```
✓ 1.2s

c. Read Dataset



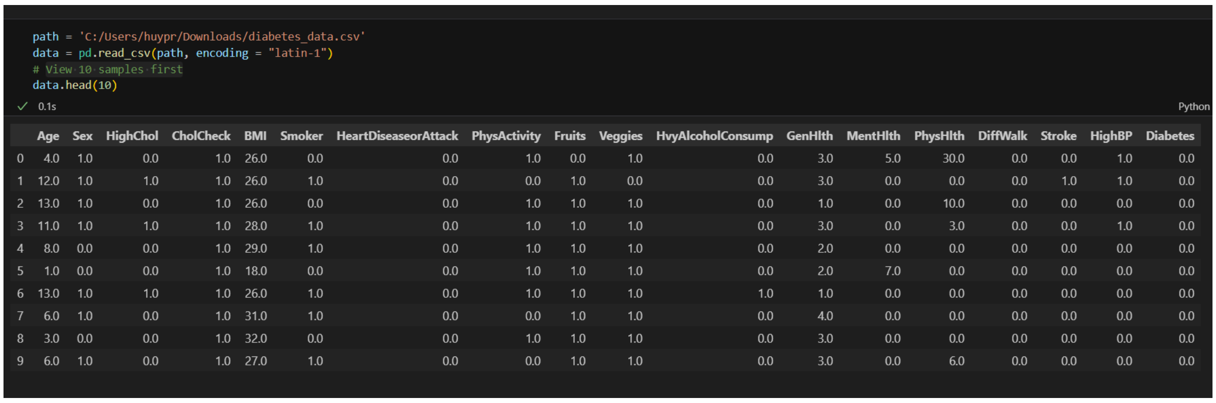
```
path = 'C:/Users/huypr/Downloads/diabetes_data.csv'
data = pd.read_csv(path, encoding = "latin-1")
data.head()
```

✓ 0.1s Python

	Age	Sex	HighChol	CholCheck	BMI	Smoker	HeartDiseaseorAttack	PhysActivity	Fruits	Veggies	HvyAlcoholConsump	GenHlth	MentHlth	PhysHlth	Diffl
0	4.0	1.0	0.0	1.0	26.0	0.0	0.0	1.0	0.0	1.0	0.0	3.0	5.0	30.0	
1	12.0	1.0	1.0	1.0	26.0	1.0	0.0	0.0	1.0	0.0	0.0	3.0	0.0	0.0	
2	13.0	1.0	0.0	1.0	26.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	10.0	
3	11.0	1.0	1.0	1.0	28.0	1.0	0.0	1.0	1.0	1.0	0.0	3.0	0.0	3.0	
4	8.0	0.0	0.0	1.0	29.0	1.0	0.0	1.0	1.0	1.0	0.0	2.0	0.0	0.0	

2. Exploratory Data Analysis

- View 10 samples first

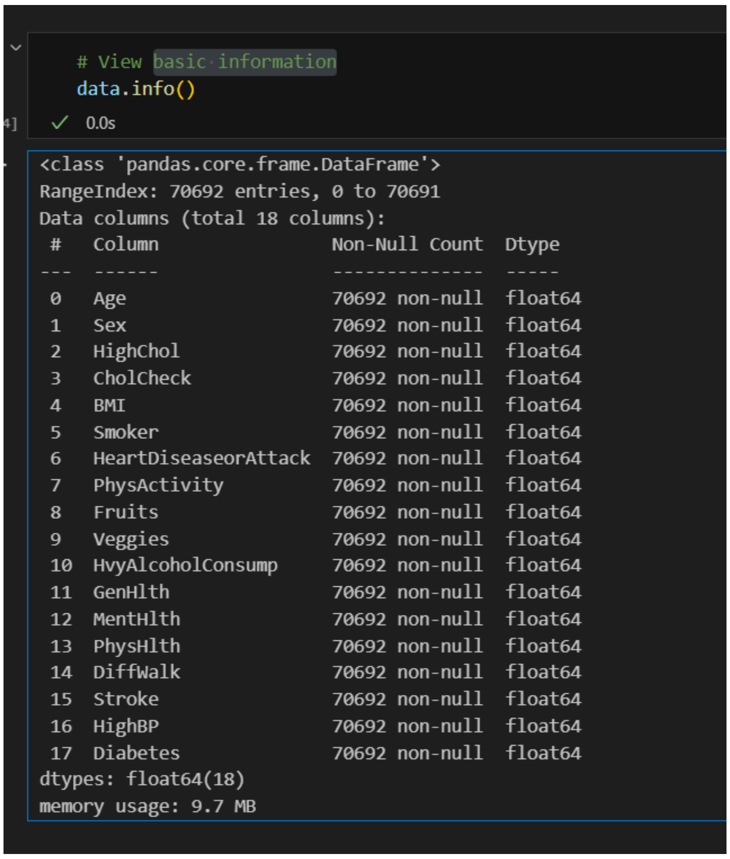


```
path = 'C:/Users/huypr/Downloads/diabetes_data.csv'
data = pd.read_csv(path, encoding = "latin-1")
# View 10 samples first
data.head(10)
```

✓ 0.1s Python

	Age	Sex	HighChol	CholCheck	BMI	Smoker	HeartDiseaseorAttack	PhysActivity	Fruits	Veggies	HvyAlcoholConsump	GenHlth	MentHlth	PhysHlth	DiffWalk	Stroke	HighBP	Diabetes
0	4.0	1.0	0.0	1.0	26.0	0.0	0.0	1.0	0.0	1.0	0.0	3.0	5.0	30.0	0.0	0.0	1.0	0.0
1	12.0	1.0	1.0	1.0	26.0	1.0	0.0	0.0	1.0	0.0	0.0	3.0	0.0	0.0	0.0	1.0	1.0	0.0
2	13.0	1.0	0.0	1.0	26.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	10.0	0.0	0.0	0.0	0.0
3	11.0	1.0	1.0	1.0	28.0	1.0	0.0	1.0	1.0	1.0	0.0	3.0	0.0	3.0	0.0	0.0	1.0	0.0
4	8.0	0.0	0.0	1.0	29.0	1.0	0.0	1.0	1.0	1.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0
5	1.0	0.0	0.0	1.0	18.0	0.0	0.0	1.0	1.0	1.0	0.0	2.0	7.0	0.0	0.0	0.0	0.0	0.0
6	13.0	1.0	1.0	1.0	26.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
7	6.0	1.0	0.0	1.0	31.0	1.0	0.0	0.0	1.0	1.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0
8	3.0	0.0	0.0	1.0	32.0	0.0	0.0	1.0	1.0	1.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0
9	6.0	1.0	0.0	1.0	27.0	1.0	0.0	0.0	1.0	1.0	0.0	3.0	0.0	6.0	0.0	0.0	0.0	0.0

- View basic information

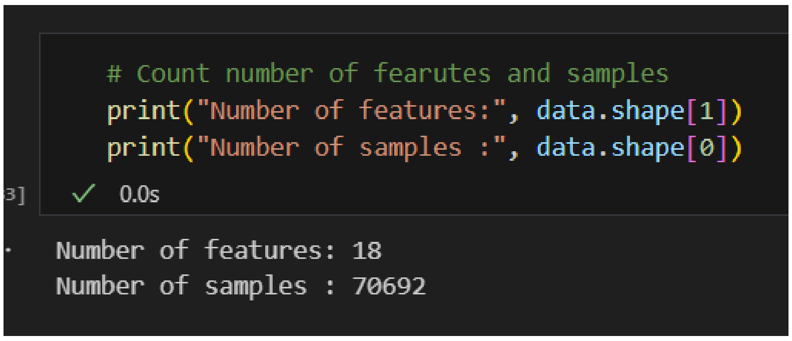


```
# View basic information
data.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70692 entries, 0 to 70691
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Age                   70692 non-null  float64
 1   Sex                   70692 non-null  float64
 2   HighChol              70692 non-null  float64
 3   CholCheck             70692 non-null  float64
 4   BMI                   70692 non-null  float64
 5   Smoker                70692 non-null  float64
 6   HeartDiseaseorAttack  70692 non-null  float64
 7   PhysActivity          70692 non-null  float64
 8   Fruits                70692 non-null  float64
 9   Veggies               70692 non-null  float64
 10  HvyAlcoholConsump     70692 non-null  float64
 11  GenHlth               70692 non-null  float64
 12  MentHlth              70692 non-null  float64
 13  PhysHlth              70692 non-null  float64
 14  DiffWalk              70692 non-null  float64
 15  Stroke                70692 non-null  float64
 16  HighBP                70692 non-null  float64
 17  Diabetes              70692 non-null  float64
dtypes: float64(18)
memory usage: 9.7 MB
```

- Count number of features and samples



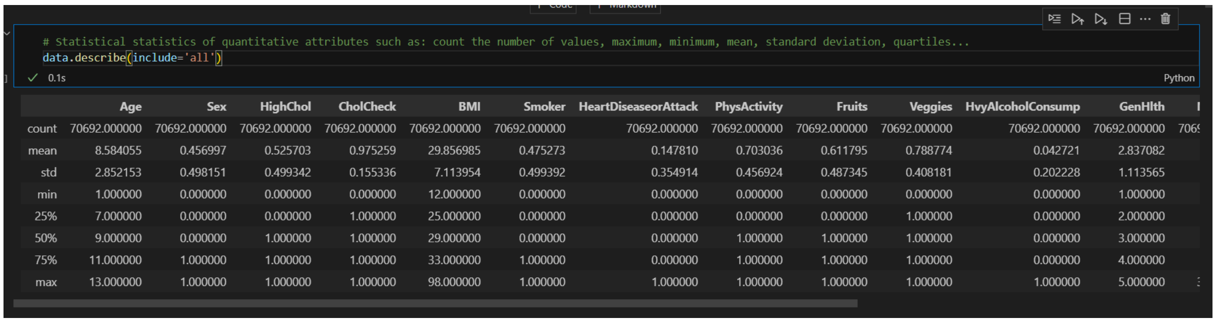
```
# Count number of fearutes and samples
print("Number of features:", data.shape[1])
print("Number of samples :", data.shape[0])
```

```
0.0s
```

```
Number of features: 18
Number of samples : 70692
```

➩ This dataset have 18 features and 70692 samples

- Count number of features and samples Statistical statistics of quantitative attributes such as: count the number of values, maximum, minimum, mean, standard deviation, quartiles...

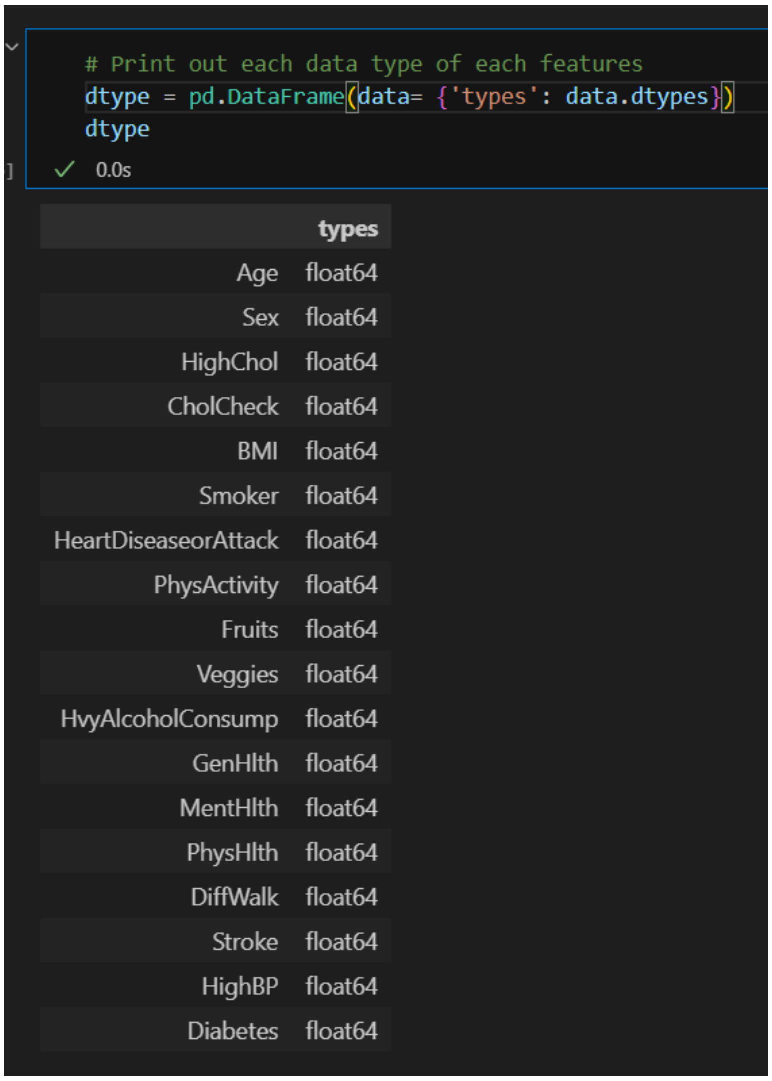


```
# Statistical statistics of quantitative attributes such as: count the number of values, maximum, minimum, mean, standard deviation, quartiles...
data.describe(include='all')
```

```
0.1s
```

	Age	Sex	HighChol	CholCheck	BMI	Smoker	HeartDiseaseorAttack	PhysActivity	Fruits	Veggies	HvyAlcoholConsump	GenHlth
count	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000	70692.000000
mean	8.584055	0.456997	0.525703	0.975259	29.856985	0.475273	0.147810	0.703036	0.611795	0.788774	0.042721	2.837082
std	2.852153	0.498151	0.499342	0.155336	7.113954	0.499392	0.354914	0.456924	0.487345	0.408181	0.202228	1.113565
min	1.000000	0.000000	0.000000	0.000000	12.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	7.000000	0.000000	0.000000	1.000000	25.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	2.000000
50%	9.000000	0.000000	1.000000	1.000000	29.000000	0.000000	0.000000	1.000000	1.000000	1.000000	0.000000	3.000000
75%	11.000000	1.000000	1.000000	1.000000	33.000000	1.000000	0.000000	1.000000	1.000000	1.000000	0.000000	4.000000
max	13.000000	1.000000	1.000000	1.000000	98.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	5.000000

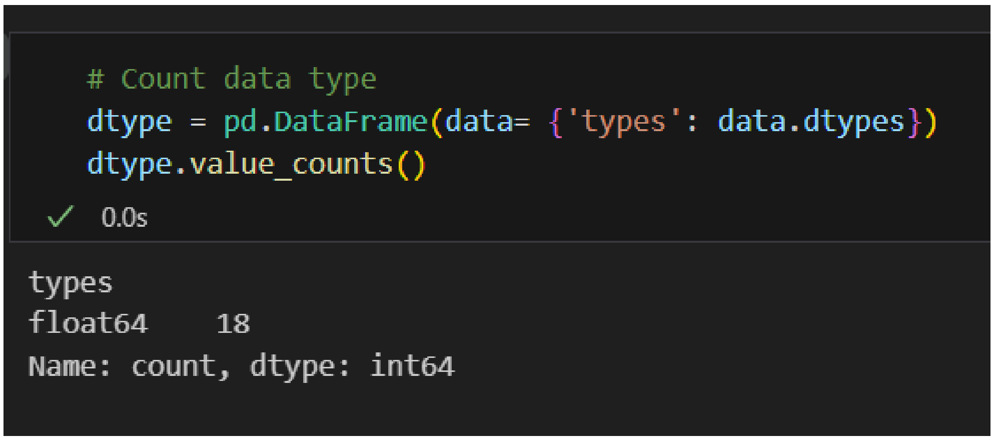
- Print out each data type of each features



```
# Print out each data type of each features
dtype = pd.DataFrame(data= {'types': data.dtypes})
dtype
✓ 0.0s
```

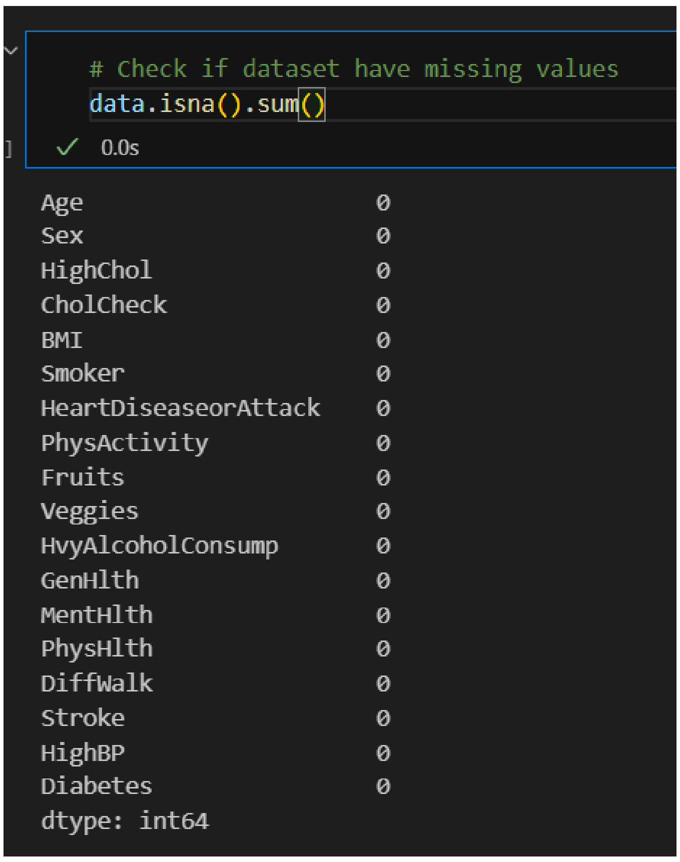
	types
Age	float64
Sex	float64
HighChol	float64
CholCheck	float64
BMI	float64
Smoker	float64
HeartDiseaseorAttack	float64
PhysActivity	float64
Fruits	float64
Veggies	float64
HvyAlcoholConsump	float64
GenHlth	float64
MentHlth	float64
PhysHlth	float64
DiffWalk	float64
Stroke	float64
HighBP	float64
Diabetes	float64

- Count data type



```
# Count data type
dtype = pd.DataFrame(data= {'types': data.dtypes})
dtype.value_counts()
✓ 0.0s

types
float64    18
Name: count, dtype: int64
```

➪ We see there are 18 features of type float
- Check if dataset have missing values



```
# Check if dataset have missing values
data.isna().sum()
```

✓ 0.0s

```
Age                     0
Sex                     0
HighChol                0
CholCheck               0
BMI                     0
Smoker                  0
HeartDiseaseorAttack    0
PhysActivity            0
Fruits                  0
Veggies                 0
HvyAlcoholConsump       0
GenHlth                 0
MentHlth                0
PhysHlth                0
DiffWalk                0
Stroke                  0
HighBP                  0
Diabetes                0
dtype: int64
```

⇨ We can see that no feature contains missing value

CHAPTER III: PROBLEMS AND DATA PREPROCESSING

1. Problems

Predicting the prevalence of diabetes based on various factors and attributes within the dataset.

- Input:

The dataset consists of 17 attributes with 70,692 rows of data.

COLUMN NAME	TYPE
Age	INT
Sex	BIT
HighChol	BIT
CholCheck	BIT
BMI	INT
Smoker	BIT
HeartDiseaseorAttack	BIT
PhysActivity	BIT
Fruits	BIT
Veggies	BIT
HvyAlcoholConsump	BIT
GenHlth	INT
MentHlth	INT
PhysHlth	INT
DiffWalk	BIT
Stroke	BIT
HighBP	BIT

- Output:

Determine the value for the diabetes column to predict the incidence of diabetes based on 2 values: 0 - no disease, 1 - yes.

COLUMN NAME	TYPE
Diabetes	BIT

2. Data Preprocessing

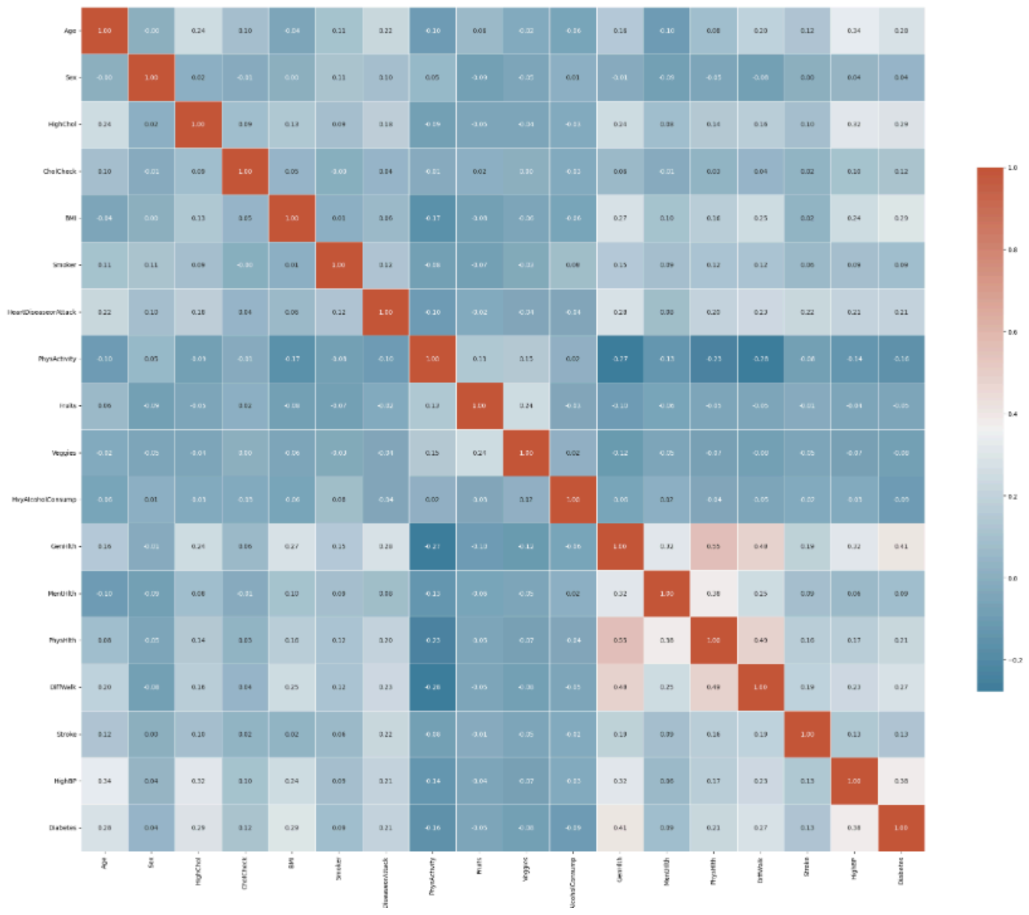
a. Feature Encoding

Firstly, we use the correlation matrix to see the relation of features in the dataset.

```
memory usage: 9.7 MB
```

```
[5]: corr = data.corr()
     fig, ax = plt.subplots(figsize= (30,30))
     sns.heatmap(corr, mask=np.zeros_like(corr, dtype=bool), cmap=sns.diverging_palette(230, 20, as_cmap=True),
                 square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True, fmt=".2f")
```

```
[5]: <Axes: >
```



This is correlation between target ‘Diabetes’ with other features.

```
# Correlation between target with other features
corr = data.corr()
corr['Diabetes']
```

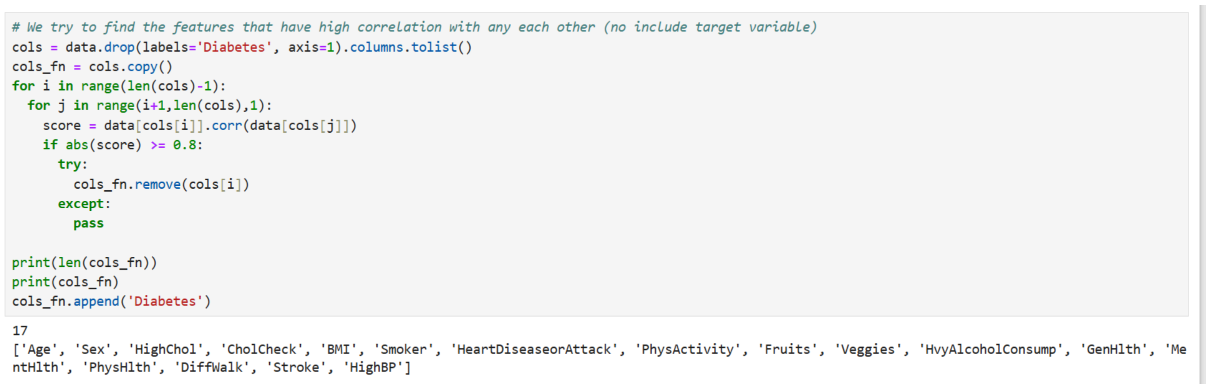
```
Age                     0.278738
Sex                     0.044413
HighChol                0.289213
CholCheck               0.115382
BMI                     0.293373
Smoker                  0.085999
HeartDiseaseorAttack    0.211523
PhysActivity           -0.158666
Fruits                 -0.054077
Veggies                -0.079293
HvyAlcoholConsump      -0.094853
GenHlth                 0.407612
MentHlth                0.087029
PhysHlth                0.213081
DiffWalk                0.272646
Stroke                  0.125427
HighBP                  0.381516
Diabetes                1.000000
Name: Diabetes, dtype: float64
```

When checking, we found that this dataset does not contain Null values. So we skip the Handling missing data step.

b. Feature selection

We use threshold by correlation matrix to choose important feature.

We find the features that have high correlation with any each other (no include target variable). This is the result:



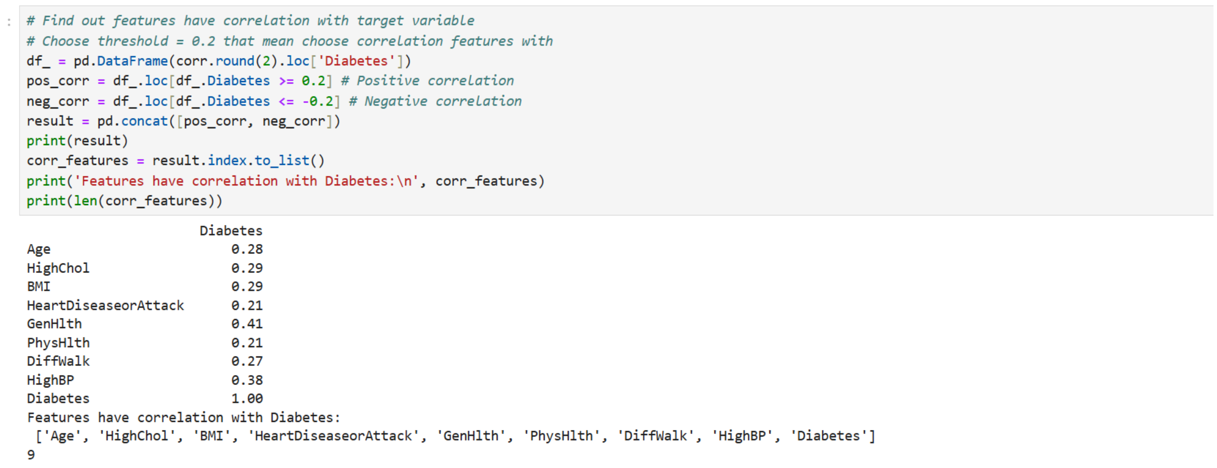
```
# We try to find the features that have high correlation with any each other (no include target variable)
cols = data.drop(labels='Diabetes', axis=1).columns.tolist()
cols_fn = cols.copy()
for i in range(len(cols)-1):
  for j in range(i+1,len(cols),1):
    score = data[cols[i]].corr(data[cols[j]])
    if abs(score) >= 0.8:
      try:
        cols_fn.remove(cols[i])
      except:
        pass

print(len(cols_fn))
print(cols_fn)
cols_fn.append('Diabetes')
```

```
17
['Age', 'Sex', 'HighChol', 'CholCheck', 'BMI', 'Smoker', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies', 'HvyAlcoholConsump', 'GenHlth', 'Me
ntHlth', 'PhysHlth', 'DiffWalk', 'Stroke', 'HighBP']
```

After that, we find out features have correlation with target variable 'Diabetes'.

We choose threshold = 0.2 that mean choose correlation features with 'Diabetes'.

The result has eight features: Age, HighChol, BMI, HeartDiseaseorAttack, GenHlth, PhysHlth, DiffWalk, HighBP.



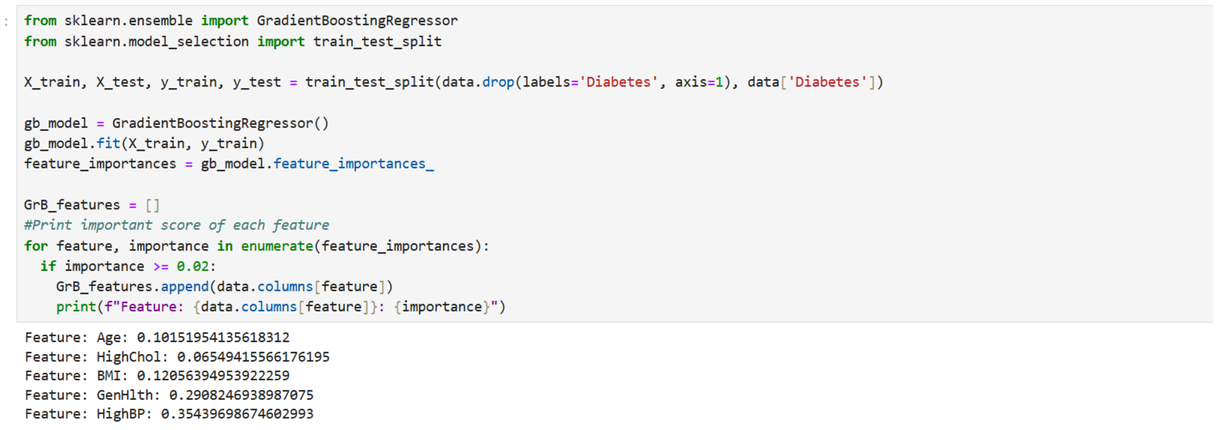
```
# Find out features have correlation with target variable
# Choose threshold = 0.2 that mean choose correlation features with
df_ = pd.DataFrame(corr.round(2).loc['Diabetes'])
pos_corr = df_.loc[df_.Diabetes >= 0.2] # Positive correlation
neg_corr = df_.loc[df_.Diabetes <= -0.2] # Negative correlation
result = pd.concat([pos_corr, neg_corr])
print(result)
corr_features = result.index.to_list()
print('Features have correlation with Diabetes:\n', corr_features)
print(len(corr_features))
```

```
                      Diabetes
Age                       0.28
HighChol                  0.29
BMI                       0.29
HeartDiseaseorAttack      0.21
GenHlth                   0.41
PhysHlth                  0.21
DiffWalk                  0.27
HighBP                    0.38
Diabetes                  1.00
Features have correlation with Diabetes:
 ['Age', 'HighChol', 'BMI', 'HeartDiseaseorAttack', 'GenHlth', 'PhysHlth', 'DiffWalk', 'HighBP', 'Diabetes']
9
```

Using GradientBoosting to choose important feature:

We use Gradient Boosting to find important attributes that greatly affect the dataset. This is the result:

There are five features: Age, HighChol, BMI, GenHlth, HighBP.

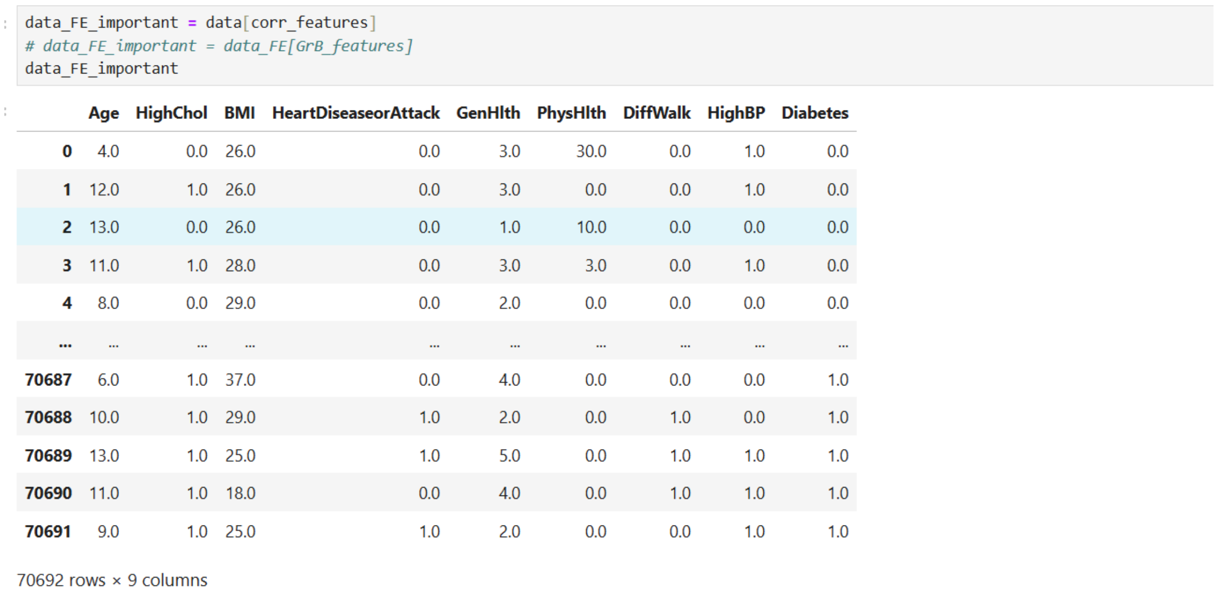

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data.drop(labels='Diabetes', axis=1), data['Diabetes'])

gb_model = GradientBoostingRegressor()
gb_model.fit(X_train, y_train)
feature_importances = gb_model.feature_importances_

GrB_features = []
#Print important score of each feature
for feature, importance in enumerate(feature_importances):
  if importance >= 0.02:
    GrB_features.append(data.columns[feature])
    print(f"Feature: {data.columns[feature]}: {importance}")
```

```
Feature: Age: 0.10151954135618312
Feature: HighChol: 0.06549415566176195
Feature: BMI: 0.12056394953922259
Feature: GenHlth: 0.2908246938987075
Feature: HighBP: 0.35439698674602993
```

After running many tests to evaluate the performance of the predictive models, we found that the accuracy of this method is significantly lower than that of using threshold by correlation matrix.

We decided to use the threshold according to the correlation matrix for feature selection.

We proceed to store the processed data in the ‘data_FE_important’.


```
data_FE_important = data[corr_features]
# data_FE_important = data_FE[GrB_features]
data_FE_important
```

	Age	HighChol	BMI	HeartDiseaseorAttack	GenHlth	PhysHlth	DiffWalk	HighBP	Diabetes
0	4.0	0.0	26.0	0.0	3.0	30.0	0.0	1.0	0.0
1	12.0	1.0	26.0	0.0	3.0	0.0	0.0	1.0	0.0
2	13.0	0.0	26.0	0.0	1.0	10.0	0.0	0.0	0.0
3	11.0	1.0	28.0	0.0	3.0	3.0	0.0	1.0	0.0
4	8.0	0.0	29.0	0.0	2.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...
70687	6.0	1.0	37.0	0.0	4.0	0.0	0.0	0.0	1.0
70688	10.0	1.0	29.0	1.0	2.0	0.0	1.0	0.0	1.0
70689	13.0	1.0	25.0	1.0	5.0	0.0	1.0	1.0	1.0
70690	11.0	1.0	18.0	0.0	4.0	0.0	1.0	1.0	1.0
70691	9.0	1.0	25.0	1.0	2.0	0.0	0.0	1.0	1.0

70692 rows × 9 columns

c. Outlier Processing

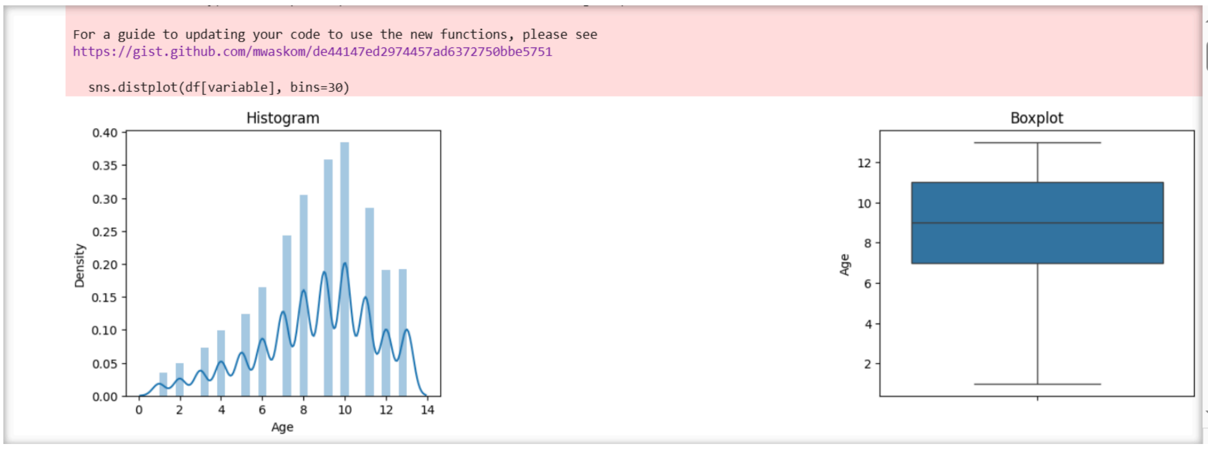
Define function

We define functions to create a histogram, boxplot to determine whether outliers appear or not (we do not use Q-Q plot because most of the attributes in this dataset only have the value 0 or 1).

```
# Function to create histogram, Q-Q plot and boxplot
def diagnostic_plots(df, variable):
    # function takes a dataframe (df) and the variable of interest as arguments

    # define figure size
    plt.figure(figsize=(16, 4))

    # histogram
    plt.subplot(1, 3, 1)
    sns.distplot(df[variable], bins=30)
    plt.title('Histogram')

    # boxplot
    plt.subplot(1, 3, 3)
    sns.boxplot(y=df[variable])
    plt.title('Boxplot')

    plt.show()
```

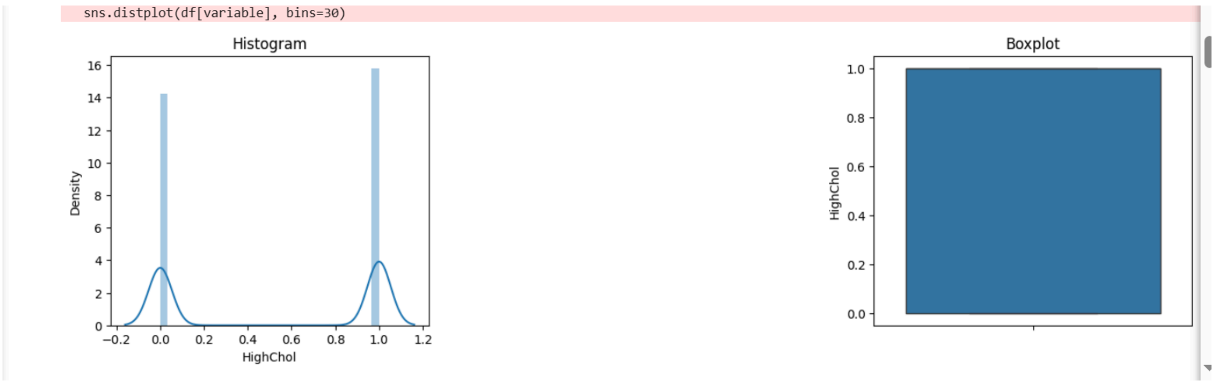
```
#Visualize distribution of each column and find outliers
for col in data_FE_important.columns:
  diagnostic_plots(data_FE_important, col)
```

Visualize distribution of each column and find outliers. This is the result of nine features:



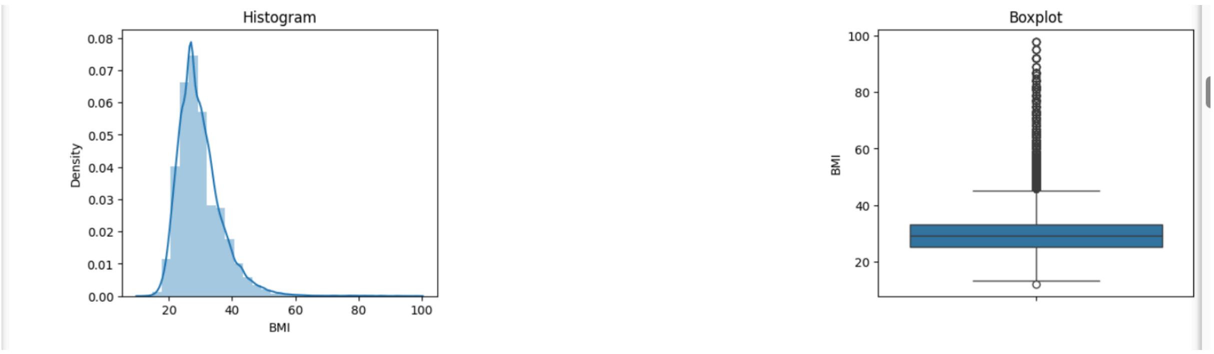
```
For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[variable], bins=30)
```

Age

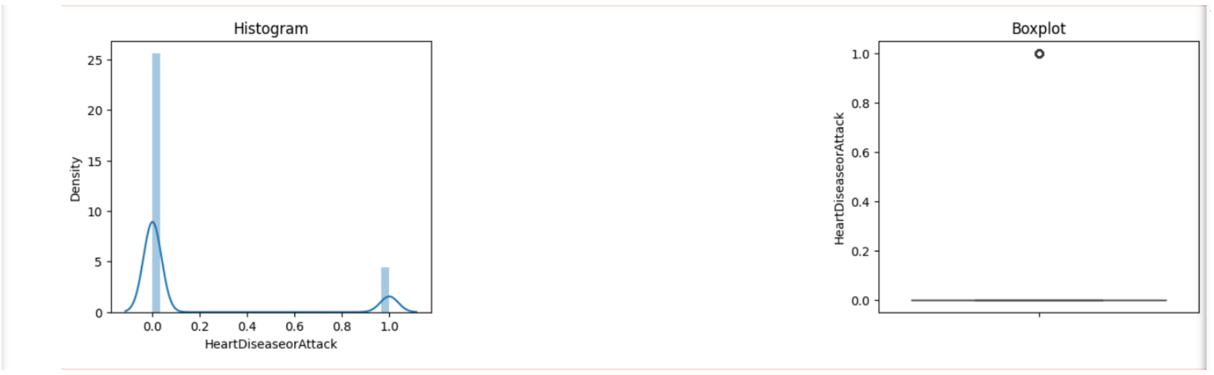


```
  sns.distplot(df[variable], bins=30)
```

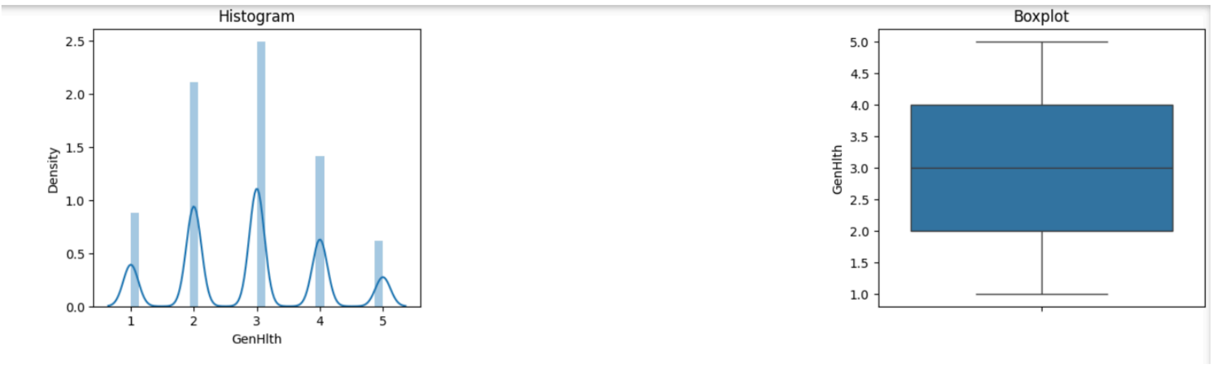
HighChol



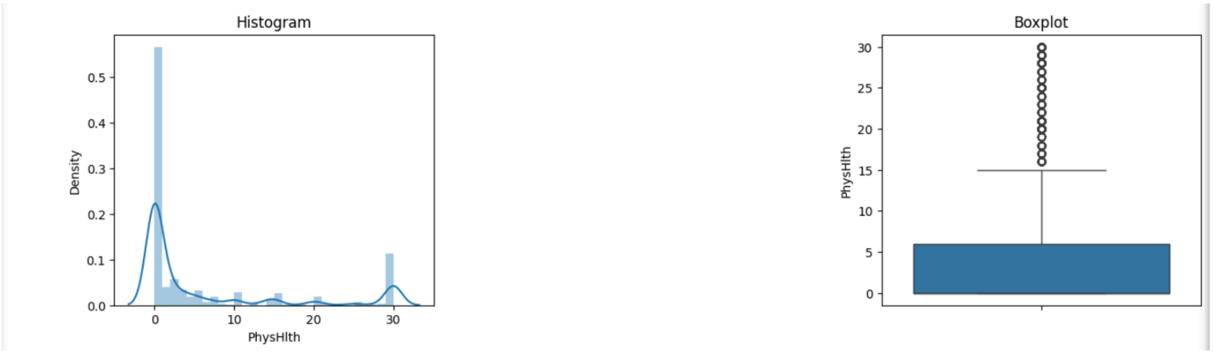
BMI



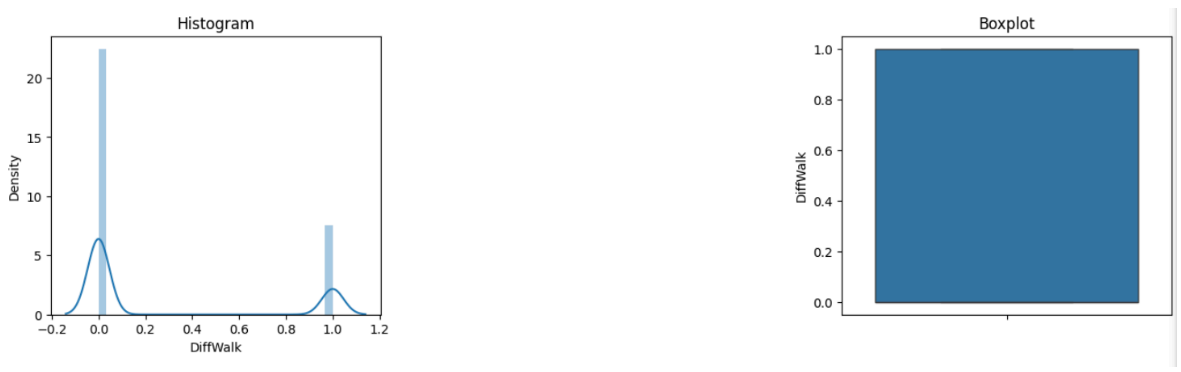
HeartDiseaseorAttack



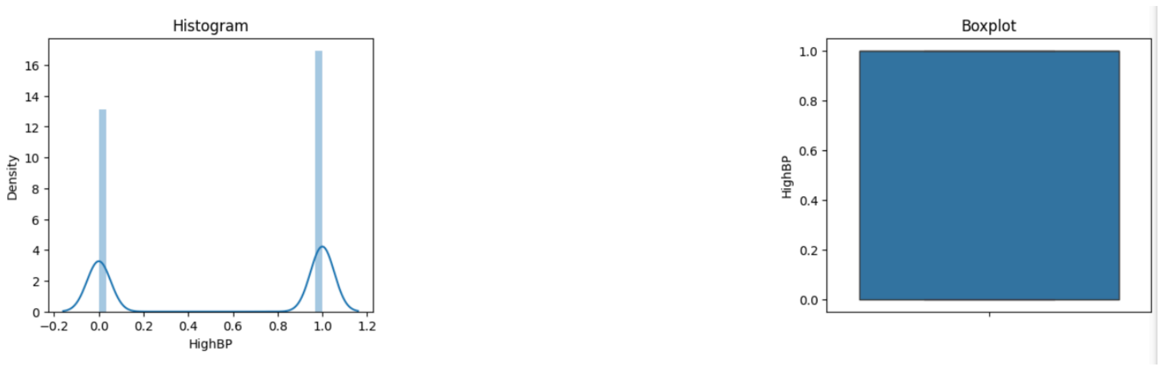
GenHlth



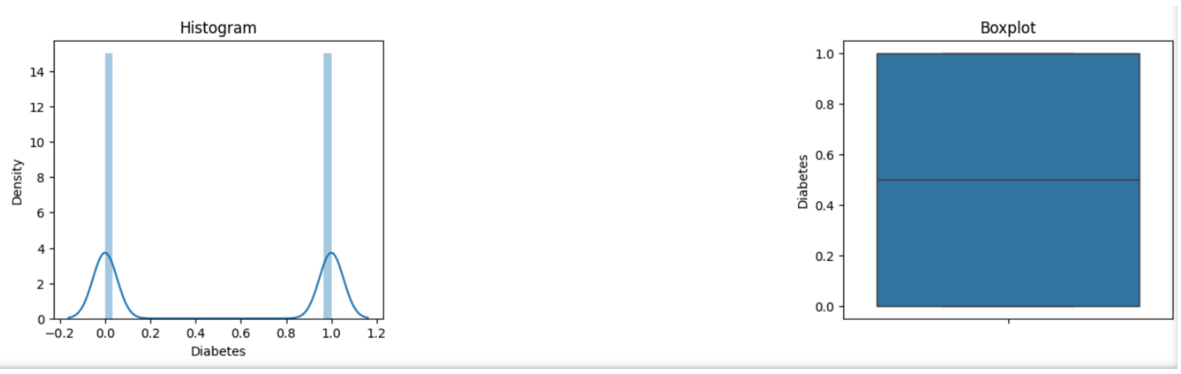
PhysHlth



DiffWalk



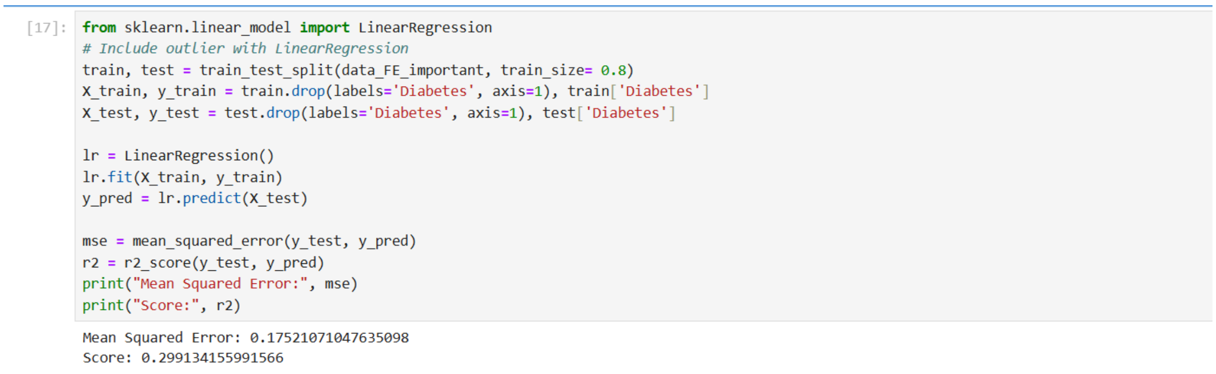
HighBP



Diabetes

We test whether the values affect the accuracy of the algorithm or not.

Use LinearRegression to conduct testing. First, do LinearRegression with outliers. Accuracy of this algorithm is 0.299 (1).



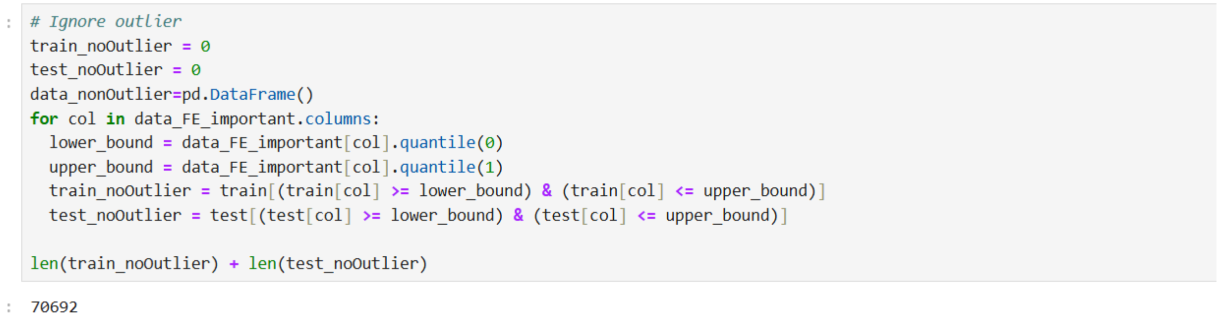
```
[17]: from sklearn.linear_model import LinearRegression
      # Include outlier with LinearRegression
      train, test = train_test_split(data_FE_important, train_size= 0.8)
      X_train, y_train = train.drop(labels='Diabetes', axis=1), train['Diabetes']
      X_test, y_test = test.drop(labels='Diabetes', axis=1), test['Diabetes']

      lr = LinearRegression()
      lr.fit(X_train, y_train)
      y_pred = lr.predict(X_test)

      mse = mean_squared_error(y_test, y_pred)
      r2 = r2_score(y_test, y_pred)
      print("Mean Squared Error:", mse)
      print("Score:", r2)
```

```
Mean Squared Error: 0.17521071047635098
Score: 0.299134155991566
```

Ignore outlier

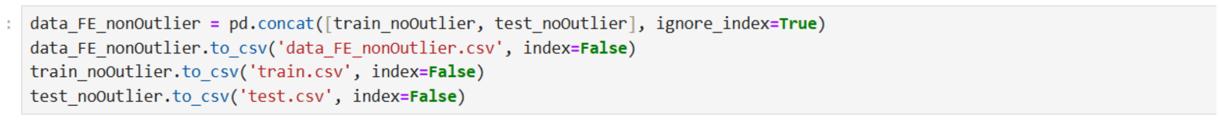
We split the outlier-removed dataset and store it into csv files:

- ‘data_FE_nonOutlier.csv’
- ‘train.csv’
- ‘test.csv’



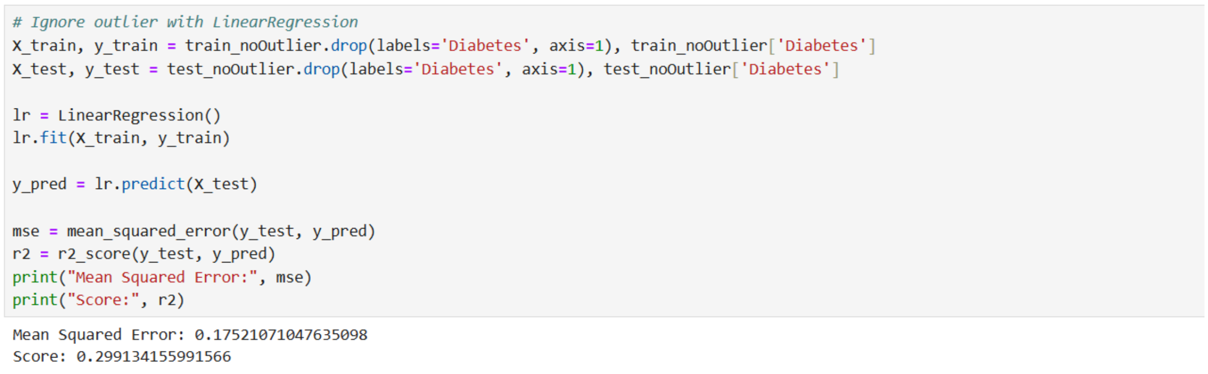
```
# Ignore outlier
train_noOutlier = 0
test_noOutlier = 0
data_nonOutlier=pd.DataFrame()
for col in data_FE_important.columns:
  lower_bound = data_FE_important[col].quantile(0)
  upper_bound = data_FE_important[col].quantile(1)
  train_noOutlier = train[(train[col] >= lower_bound) & (train[col] <= upper_bound)]
  test_noOutlier = test[(test[col] >= lower_bound) & (test[col] <= upper_bound)]

len(train_noOutlier) + len(test_noOutlier)
```

```
70692
```



```
data_FE_nonOutlier = pd.concat([train_noOutlier, test_noOutlier], ignore_index=True)
data_FE_nonOutlier.to_csv('data_FE_nonOutlier.csv', index=False)
train_noOutlier.to_csv('train.csv', index=False)
test_noOutlier.to_csv('test.csv', index=False)
```

We do LinearRegression with dataset that ignores outlier. Accuracy of this algorithm is 0.299 (2).



```
# Ignore outlier with LinearRegression
X_train, y_train = train_noOutlier.drop(labels='Diabetes', axis=1), train_noOutlier['Diabetes']
X_test, y_test = test_noOutlier.drop(labels='Diabetes', axis=1), test_noOutlier['Diabetes']

lr = LinearRegression()
lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("Score:", r2)
```

```
Mean Squared Error: 0.17521071047635098
Score: 0.299134155991566
```

(1),(2) ⇒ Outliers do not affect the accuracy of the algorithm.

Conclusion: after outlier processing, we compare performance of include oulier and ignore outlier in base model -> Then, include outlier have better performance so that, we evaluate outlier in current dataset out of quantile is not really outliers because of not enough large samples data.

d. Standard Scaler

We conduct standard scaler with nine selected features to apply the algorithm.

```
final_train = pd.read_csv('train.csv')
final_test = pd.read_csv('test.csv')

len(final_train) + len(final_test)
```

```
70692
```

```
from joblib import dump

sd_scale_train = final_train.drop(labels=['Diabetes'], axis=1)
sd_scale_test = final_test.drop(labels=['Diabetes'], axis=1)

SS = StandardScaler()
SS.fit(sd_scale_train)
cols = sd_scale_train.columns
sd_scale_train = pd.DataFrame(SS.transform(sd_scale_train), columns=cols)
sd_scale_test = pd.DataFrame(SS.transform(sd_scale_test), columns=cols)

dump(SS, 'std_scaler.bin', compress=True)

sd_scale_train.head(5)
```

This is the result:

	Age	HighChol	BMI	HeartDiseaseorAttack	GenHlth	PhysHlth	DiffWalk	HighBP
0	-1.260792	0.951268	1.421384	-0.415531	1.043125	2.391941	-0.582870	-1.137115
1	-0.207069	0.951268	-0.823680	2.406558	1.941213	1.896750	1.715648	0.879419
2	-0.207069	-1.051228	0.298852	-0.415531	0.145036	2.391941	-0.582870	0.879419
3	-2.314514	-1.051228	2.403599	-0.415531	0.145036	-0.579205	-0.582870	0.879419
4	-1.260792	-1.051228	1.281067	2.406558	0.145036	-0.381129	1.715648	-1.137115

```
sd_scale_test.head(5)
```

	Age	HighChol	BMI	HeartDiseaseorAttack	GenHlth	PhysHlth	DiffWalk	HighBP
0	0.144172	0.951268	0.298852	-0.415531	-0.753052	-0.579205	-0.582870	0.879419
1	0.846654	0.951268	0.439168	-0.415531	1.941213	2.391941	1.715648	0.879419
2	0.495413	0.951268	0.158536	-0.415531	0.145036	-0.579205	-0.582870	0.879419
3	0.144172	0.951268	-0.823680	-0.415531	-0.753052	-0.381129	-0.582870	-1.137115
4	-0.207069	-1.051228	1.140751	-0.415531	0.145036	-0.579205	-0.582870	0.879419

We check by using LinearRegression whether the processed data set affects the accuracy of the algorithm or not. Accuracy is 0.299.

There is no change in accuracy after using the standard scaler.

```
# Try again with LinearRegression
X_train, y_train = sd_scale_train, final_train['Diabetes']
X_test, y_test = sd_scale_test, final_test['Diabetes']

lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)

r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)

print("Mean Squared Error:", mse)
print("Score:", r2)
```

```
Mean Squared Error: 0.17521071047635087
Score: 0.29913415599156645
```

Complete data preprocessing.

CHAPTER IV: ALGORITHM

1. Reason for choosing the algorithm

a. Random Forest

- **Handles Mixed Data Types:** Dataset includes a mix of numerical (Age, BMI), categorical (Sex, Smoker), and binary (HeartDiseaseorAttack) variables. Random forest excels at handling diverse data types without requiring extensive preprocessing.
- **Robust to Outliers and Missing Values:** Random forest is less sensitive to outliers and missing values, which are common in real-world healthcare datasets. The algorithm's ensemble nature helps mitigate the impact of these issues.
- **Feature Importance Evaluation:** Random forest provides a measure of feature importance, indicating which variables (e.g., BMI, HighBP) contribute most significantly to predicting diabetes. This can offer valuable insights into the risk factors associated with the disease.
- **High Accuracy and Generalization:** Random forest often achieves high predictive accuracy and generalizes well to unseen data. It is less prone to overfitting compared to individual decision trees, thanks to the ensemble approach.
- **Non-linear Relationships:** The algorithm can capture complex, non-linear relationships between variables and the outcome (diabetes). This is crucial in medical data, where interactions between risk factors can be intricate.
- **Scalability:** Random forest is computationally efficient and can handle large datasets with numerous features, making it suitable for your dataset with 18 attributes and over 70,000 rows.
- **Interpretability:** While not as easily interpretable as individual decision trees, random forest can still provide insights into the decision-making process through feature importance analysis and partial dependence plots.
- **Proven Success in Healthcare:** Random forest has been successfully applied to various healthcare problems, including disease prediction and diagnosis, demonstrating its suitability for your task.

b. Decision Tree

- **Interpretability:** Decision trees are highly interpretable models. They mimic human decision-making processes by splitting the data based on the features' values. This makes them particularly useful when transparency and understanding of the model's decisions are important.
- **Handling Non-linear Relationships:** Decision trees can capture non-linear relationships between features and the target variable. They do not assume

linearity in the data, making them suitable for datasets with complex relationships.

- **Feature Importance:** Decision trees provide a measure of feature importance. This allows you to identify which features have the most significant impact on the model's predictions, which can be valuable for feature selection and understanding the data.
- **Robustness to Outliers:** Decision trees are robust to outliers and noise in the data. They partition the feature space into regions, and outliers typically have less impact on the model's performance compared to other algorithms.
- **Mixed Data Types:** Decision trees can handle both numerical and categorical data without the need for feature scaling or one-hot encoding. This simplifies the preprocessing steps and can be advantageous when dealing with datasets containing a mix of data types.
- **Easy to Visualize:** Decision trees can be visualized graphically, allowing you to understand the model's structure and decision-making process easily. This visualization can be helpful for communicating with stakeholders or debugging the model.
- **Ensemble Methods:** Decision trees can be used as base learners in ensemble methods like Random Forests and Gradient Boosting, which often result in improved predictive performance compared to individual decision trees.

c. K-Nearest Neighbors (KNN)

- **Simplicity and Interpretability:** KNN is a relatively simple algorithm to understand and implement. Its predictions are based on the majority class of the k-nearest neighbors, making it easy to explain the reasoning behind its decisions. This can be valuable in healthcare settings where interpretability is often crucial.
- **Non-parametric Nature:** KNN is a non-parametric algorithm, meaning it doesn't make strong assumptions about the underlying data distribution. This can be beneficial in medical datasets where relationships between variables and outcomes may not be linear or easily modeled by parametric methods.
- **Handles Mixed Data Types:** KNN can handle a mix of numerical and categorical variables, which is relevant to your dataset with attributes like Age (numerical), Sex (categorical), and Smoker (binary). You may need to standardize numerical features and encode categorical ones, but the algorithm itself can accommodate these types.
- **No Training Phase:** Unlike many other algorithms, KNN doesn't require a separate training phase. This can be advantageous when dealing with limited datasets or when you want to quickly explore the data and generate initial predictions.
- **Adaptability to New Data:** KNN is inherently adaptive to new data points. When a new sample arrives, its neighbors are identified, and the prediction

is made based on the majority class among them. This means the model can be easily updated without retraining the entire algorithm.

- **Effective for Small Datasets:** KNN can perform well on smaller datasets, like yours, where there may not be enough samples to train complex models like neural networks effectively.

2. Application of classification algorithms to data set

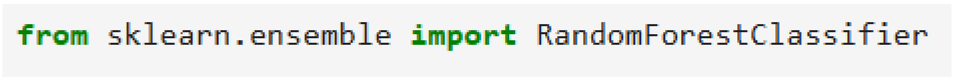
a. Random Forest Model

i. Concept

Random Forest is a supervised machine learning model widely used in Classification and Regression problems. It builds decision trees on different samples, decision trees will give the best type of each tree, RF will take those best classes and average them to get the best result at the end. together.

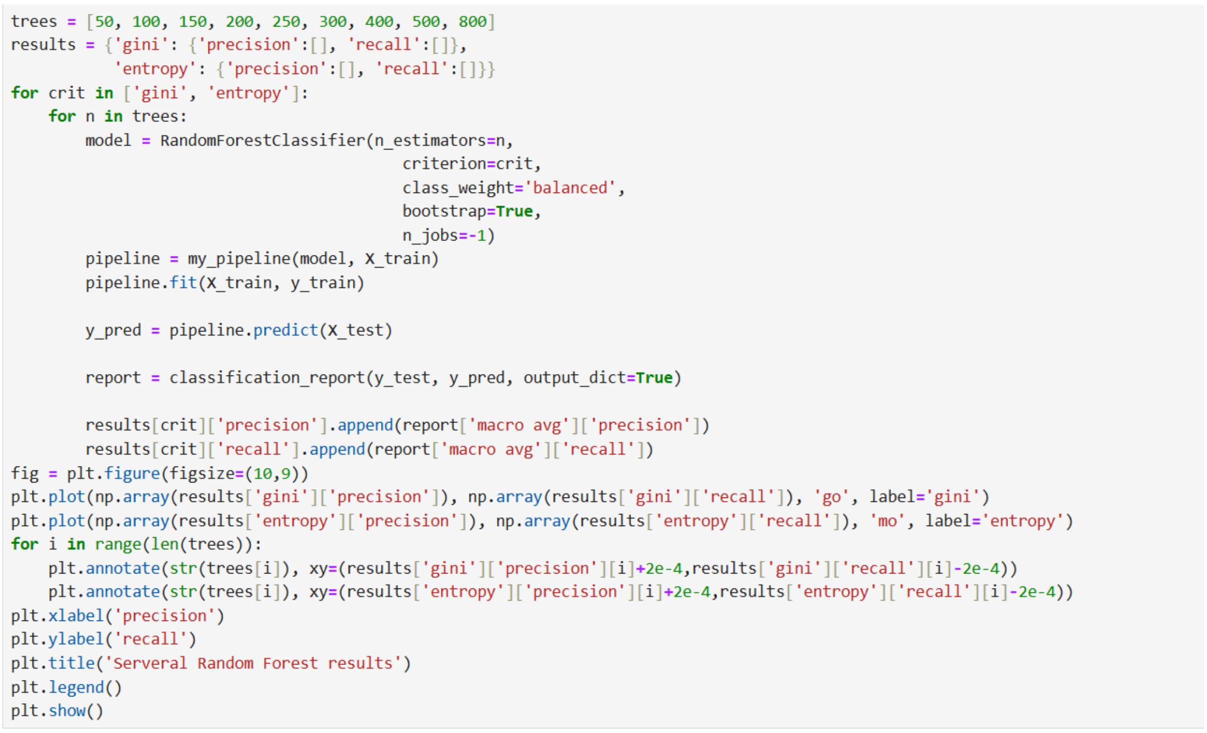
ii. Random Forest Algorithm

Import library



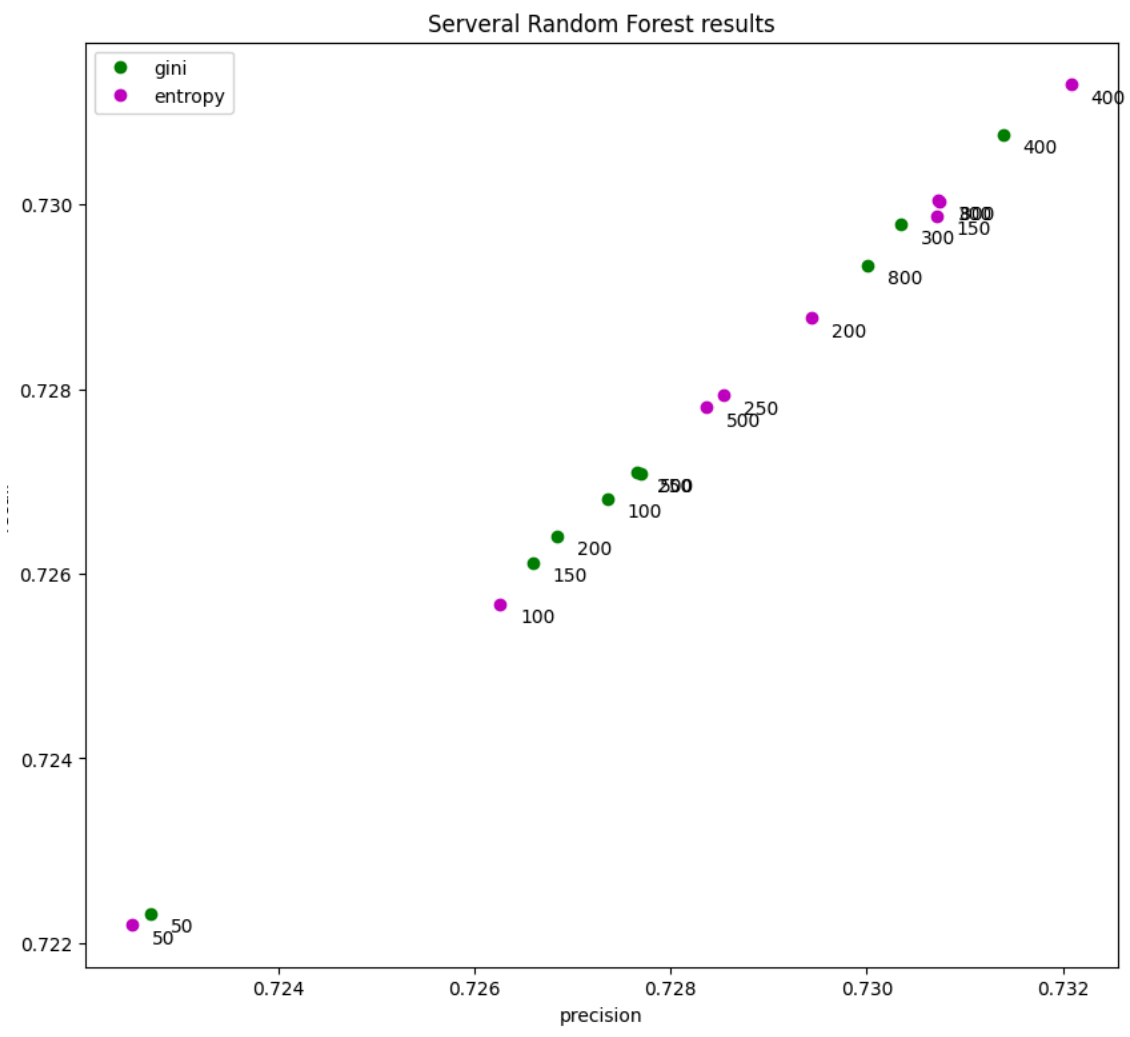
```
from sklearn.ensemble import RandomForestClassifier
```

- We created a function to forecast n-estimators to use in the random forest algorithm that will produce the highest accuracy.



```
trees = [50, 100, 150, 200, 250, 300, 400, 500, 800]
results = {'gini': {'precision':[], 'recall':[]},
           'entropy': {'precision':[], 'recall':[]}}
for crit in ['gini', 'entropy']:
    for n in trees:
        model = RandomForestClassifier(n_estimators=n,
                                      criterion=crit,
                                      class_weight='balanced',
                                      bootstrap=True,
                                      n_jobs=-1)
        pipeline = my_pipeline(model, X_train)
        pipeline.fit(X_train, y_train)

        y_pred = pipeline.predict(X_test)

        report = classification_report(y_test, y_pred, output_dict=True)

        results[crit]['precision'].append(report['macro avg']['precision'])
        results[crit]['recall'].append(report['macro avg']['recall'])
fig = plt.figure(figsize=(10,9))
plt.plot(np.array(results['gini']['precision']), np.array(results['gini']['recall']), 'go', label='gini')
plt.plot(np.array(results['entropy']['precision']), np.array(results['entropy']['recall']), 'mo', label='entropy')
for i in range(len(trees)):
    plt.annotate(str(trees[i]), xy=(results['gini']['precision'][i]+2e-4,results['gini']['recall'][i]-2e-4))
    plt.annotate(str(trees[i]), xy=(results['entropy']['precision'][i]+2e-4,results['entropy']['recall'][i]-2e-4))
plt.xlabel('precision')
plt.ylabel('recall')
plt.title('Serveral Random Forest results')
plt.legend()
plt.show()
```

This is the result:



- Define Random Forest Classifier:

```
# Define RandomForestClassifier model
model_RF = RandomForestClassifier(n_estimators=500,
                                  class_weight='balanced',
                                  bootstrap=True,
                                  n_jobs=-1)
```

- Create and fit pipeline to training data:

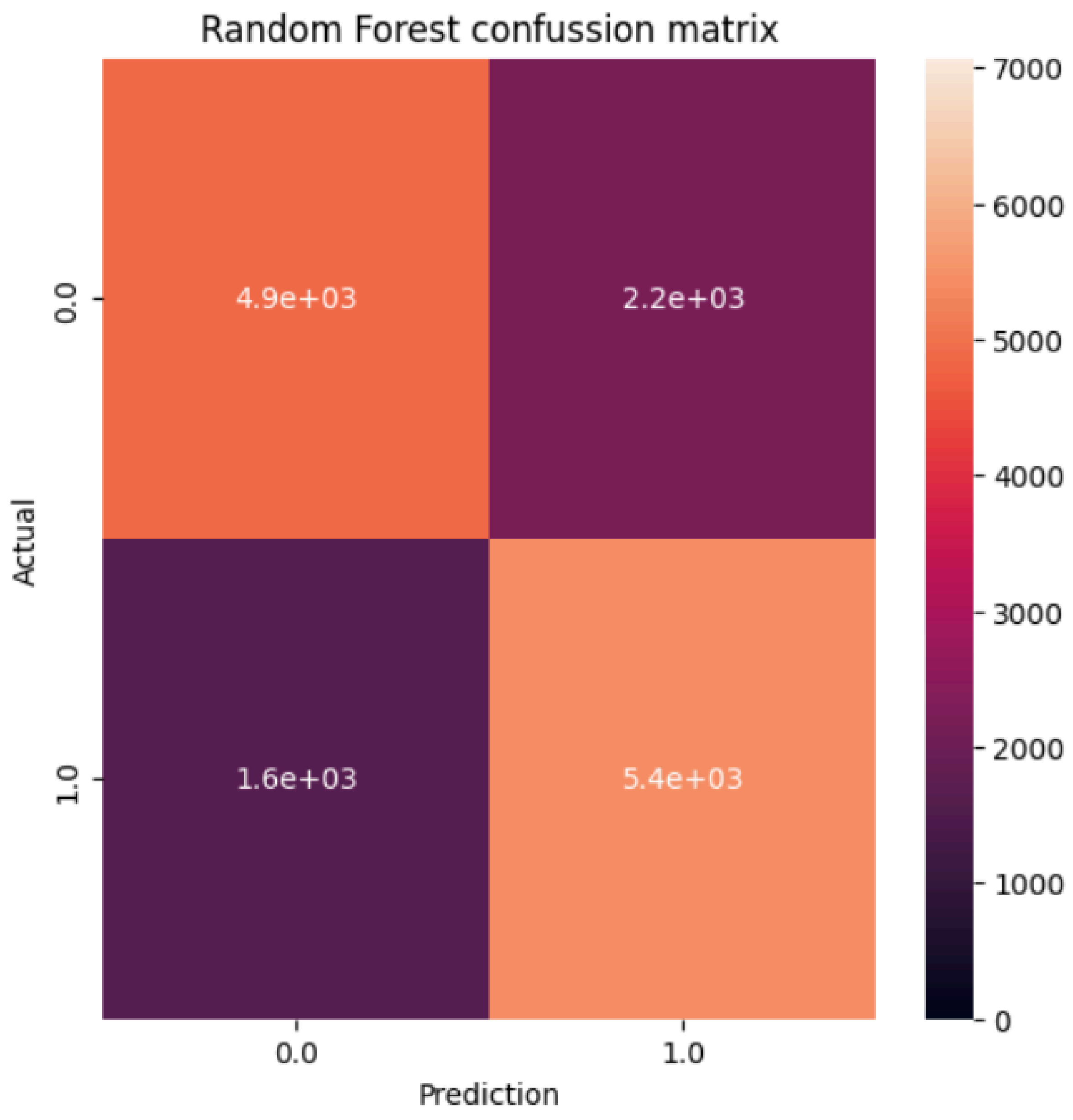
```
# Create pipeline using the defined model
pipeline_RF = my_pipeline(model_RF, X_train)

# Fit the pipeline on training data
pipeline_RF.fit(X_train, y_train)
```

- Make predictions:

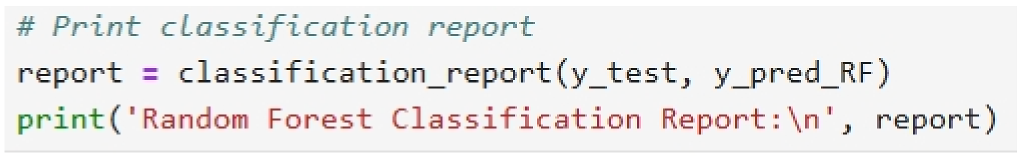
```
# Make predictions on test data
y_pred_RF = pipeline_RF.predict(X_test)
```

- Create and plot confusion matrix:

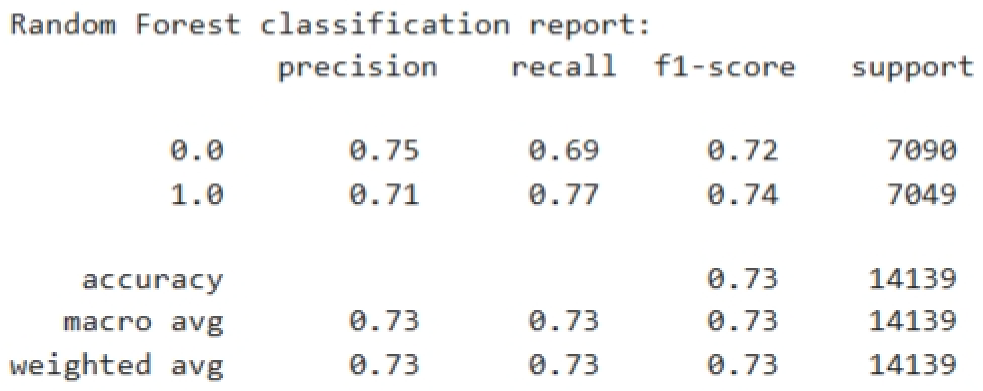
```
# Create confusion matrix
confusion_matrix = pd.crosstab(y_test, y_pred_RF, rownames=['Actual'], colnames=['Prediction'])

# Plot confusion matrix
plt.figure(figsize=(6,6))
sns.heatmap(confusion_matrix, annot=True, vmin=0, vmax=int(len(y_test)/2))
plt.title('Random Forest Confusion Matrix')
plt.show()
```



- Print classification report:



```python
# Print classification report
report = classification_report(y_test, y_pred_RF)
print('Random Forest Classification Report:\n', report)
```



```
Random Forest classification report:
              precision    recall  f1-score   support

         0.0       0.75      0.69      0.72      7090
         1.0       0.71      0.77      0.74      7049

    accuracy                           0.73     14139
   macro avg       0.73      0.73      0.73     14139
weighted avg       0.73      0.73      0.73     14139
```

b. K-Nearest Neighbors (KNN)

i. Concept

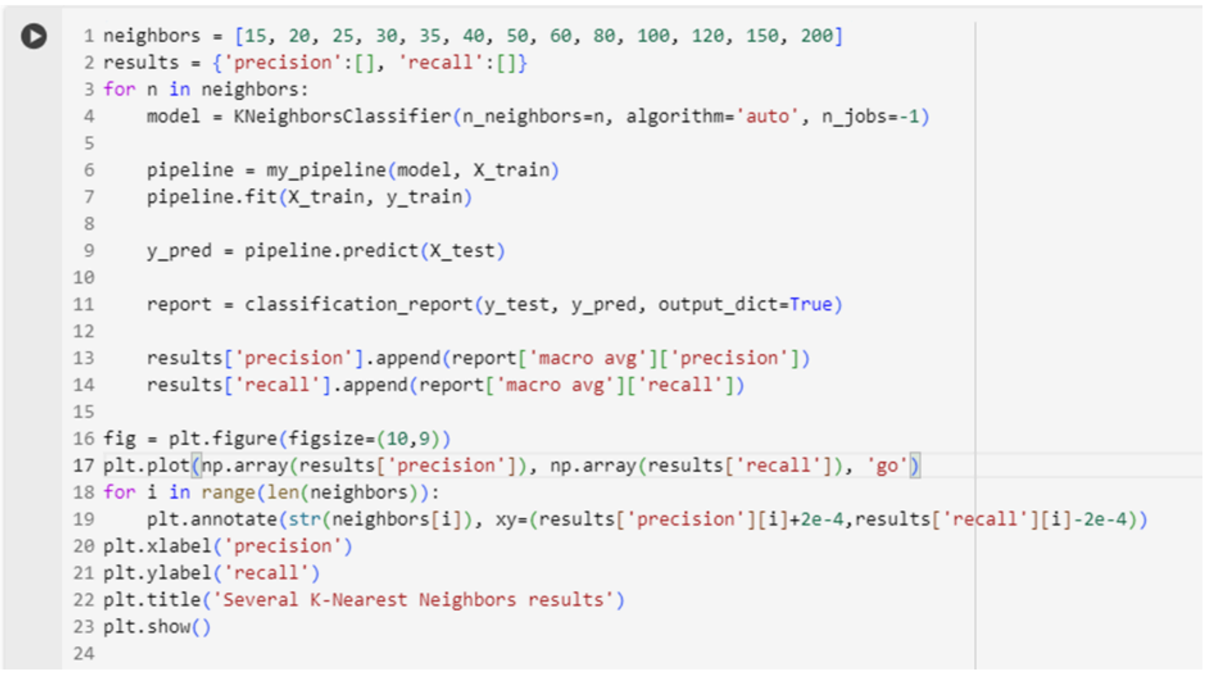
KNN is a supervised machine learning algorithm used to solve Classification and Regression problems. KNN tries to predict the correct class for the testing data by calculating the distance between the testing data and all the training points. Then select K the number of points with values that fit the testing data and calculate the probability that the testing data belongs to the training data classes K and the probability class that the highest groups will be selected.

ii. KNN Algorithm

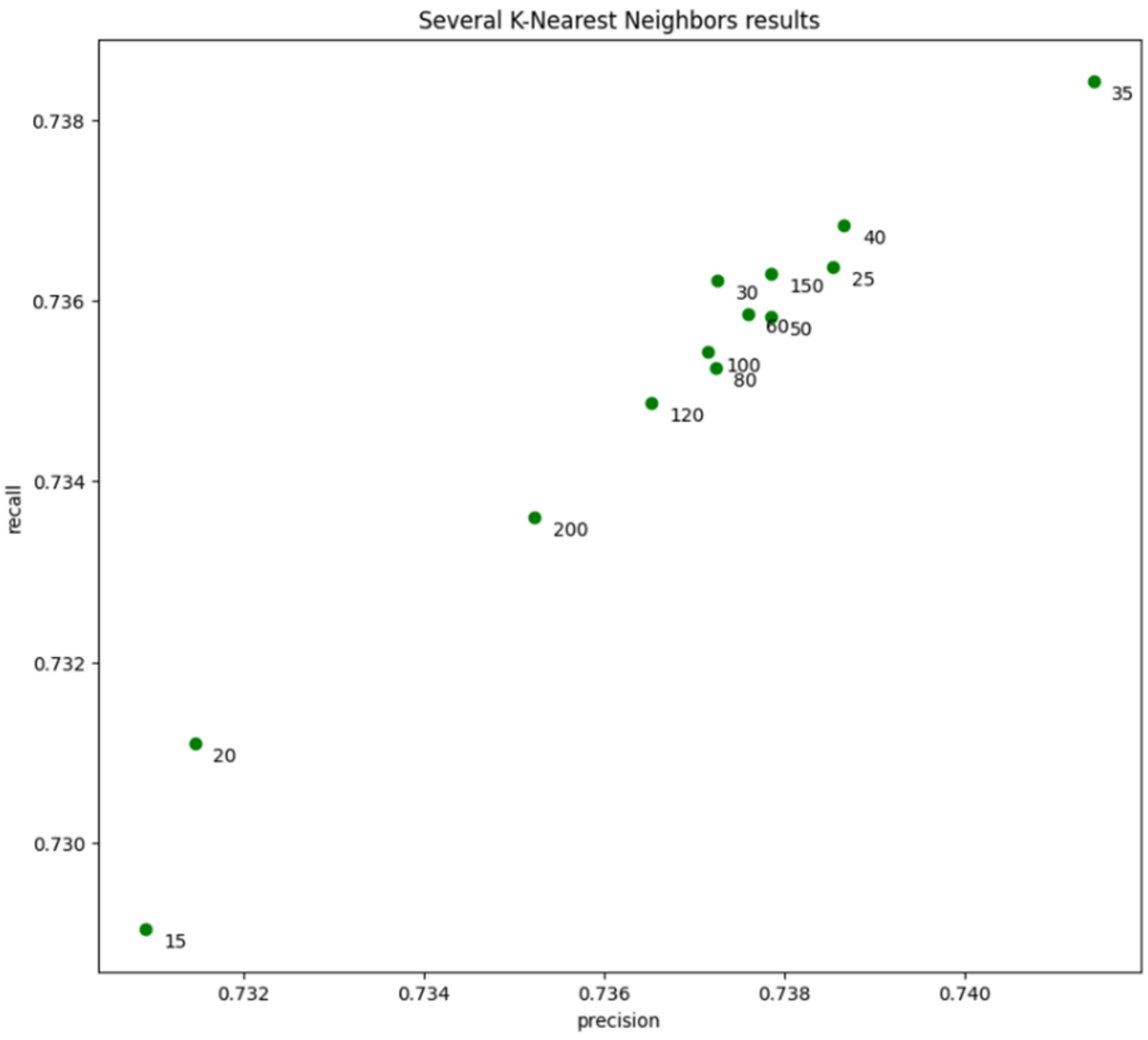
To build a K-Nearest Neighbors (KNN) model, we will use the KNeighborsClassifier function from the sklearn library to initialize the model. The parameters of the initialization function are as follows:

- **n_neighbors** (integer, default is 5): The number of nearest neighbors to use for predicting the label of a new data point. Here, we will experiment with values ranging from 15 to 200.
- **algorithm** (values include 'auto', 'ball_tree', 'kd_tree', 'brute', default is 'auto'): The algorithm used to compute the nearest neighbors. 'Auto' will attempt to select the most appropriate algorithm based on the value of n_neighbors and the nature of the data.
- **n_jobs** (integer, default is 1): The number of CPU cores used for computation. A value of -1 uses all available cores.

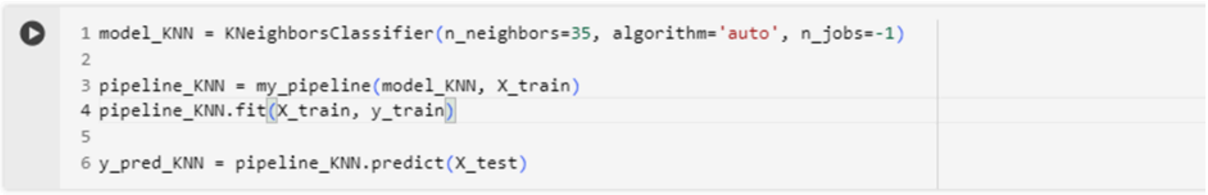
Next, we create a range of n_neighbors values to experiment with and iterate through these values to find the optimal one.



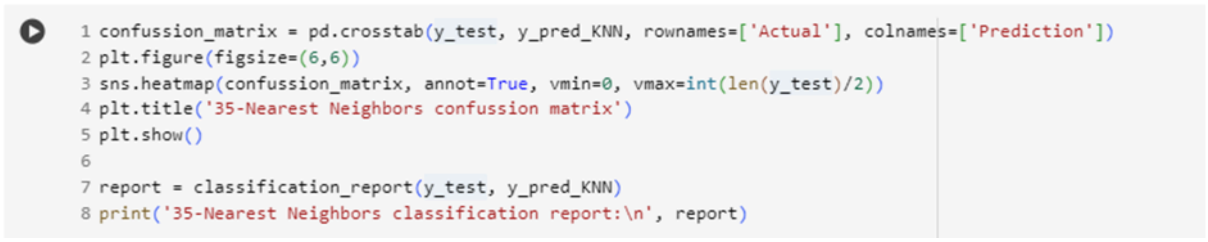
```python
neighbors = [15, 20, 25, 30, 35, 40, 50, 60, 80, 100, 120, 150, 200]
results = {'precision':[], 'recall':[]}
for n in neighbors:
    model = KNeighborsClassifier(n_neighbors=n, algorithm='auto', n_jobs=-1)

    pipeline = my_pipeline(model, X_train)
    pipeline.fit(X_train, y_train)

    y_pred = pipeline.predict(X_test)

    report = classification_report(y_test, y_pred, output_dict=True)

    results['precision'].append(report['macro avg']['precision'])
    results['recall'].append(report['macro avg']['recall'])

fig = plt.figure(figsize=(10,9))
plt.plot(np.array(results['precision']), np.array(results['recall']), 'go')
for i in range(len(neighbors)):
    plt.annotate(str(neighbors[i]), xy=(results['precision'][i]+2e-4,results['recall'][i]-2e-4))
plt.xlabel('precision')
plt.ylabel('recall')
plt.title('Several K-Nearest Neighbors results')
plt.show()

```



Initialize the KNN model with the best n_neighbors value.

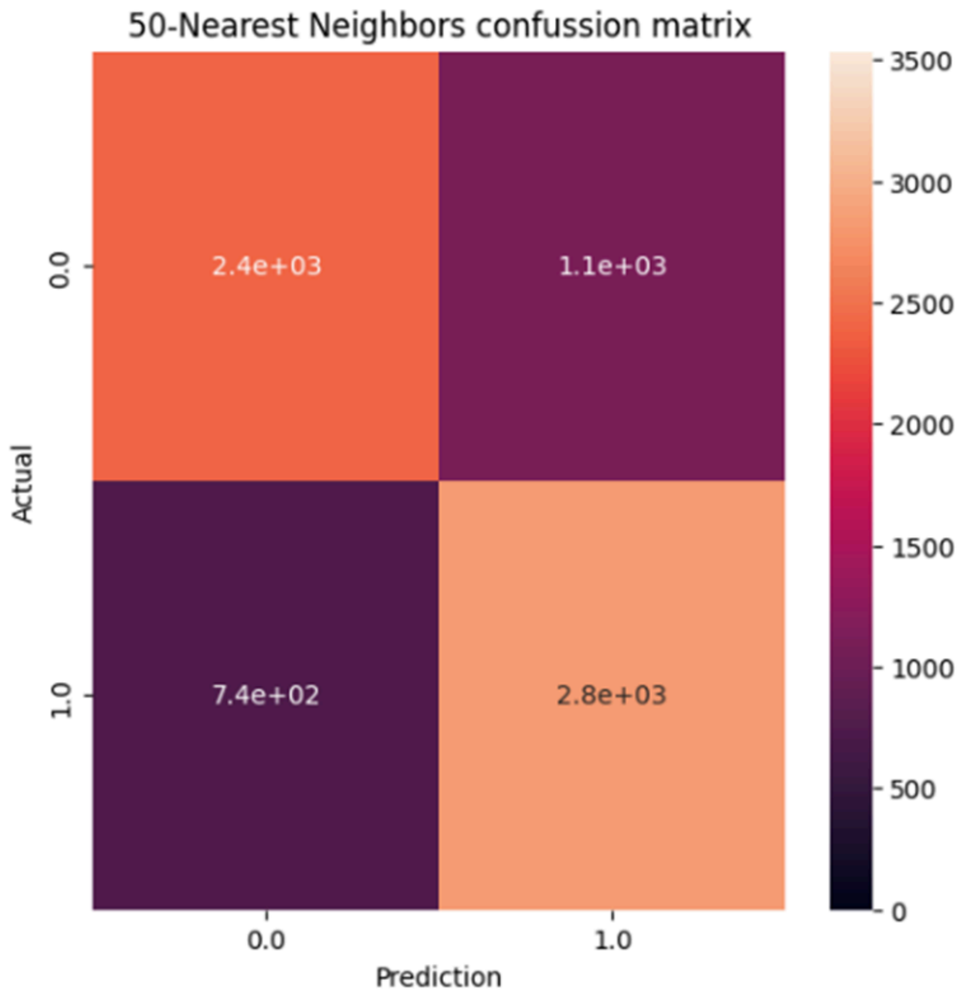


```python
model_KNN = KNeighborsClassifier(n_neighbors=35, algorithm='auto', n_jobs=-1)

pipeline_KNN = my_pipeline(model_KNN, X_train)
pipeline_KNN.fit(X_train, y_train)

y_pred_KNN = pipeline_KNN.predict(X_test)
```



```python
confussion_matrix = pd.crosstab(y_test, y_pred_KNN, rownames=['Actual'], colnames=['Prediction'])
plt.figure(figsize=(6,6))
sns.heatmap(confussion_matrix, annot=True, vmin=0, vmax=int(len(y_test)/2))
plt.title('35-Nearest Neighbors confussion matrix')
plt.show()

report = classification_report(y_test, y_pred_KNN)
print('35-Nearest Neighbors classification report:\n', report)
```

The results are as follows:



```
50-Nearest Neighbors classification report:
              precision    recall  f1-score   support

         0.0       0.76      0.68      0.72      3504
         1.0       0.72      0.79      0.75      3566

    accuracy                           0.74      7070
   macro avg       0.74      0.74      0.74      7070
weighted avg       0.74      0.74      0.74      7070
```

c. Decision Tree Model

i. Concept

Decision Tree is a popular supervised machine learning algorithm used for both classification and regression tasks. It operates by creating a tree-like structure, where each node in the tree represents a decision based on an input feature, and each branch represents the outcome of that decision. Decision trees are used to predict output values for new data by traversing from the root node to a leaf node based on the values of input features.

ii. Decision Tree Algorithm

- Import libraries:

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn import preprocessing
from sklearn import metrics
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
```

- Defining the my_pipeline function:

```python
def my_pipeline(model, x):
    scaler = preprocessing.MinMaxScaler()
    preprocessor = ColumnTransformer(transformers=[('norm', scaler, x.columns)])
    return Pipeline(steps=[('preprocessor', preprocessor), ('model', model)])
```

- Creating the Decision Tree model:

```python
model_DT = DecisionTreeClassifier(class_weight='balanced')
```

- Create pipeline and Fit pipeline to training data:

```python
pipeline_DT = my_pipeline(model_DT, X_train)
pipeline_DT.fit(X_train, y_train)
```

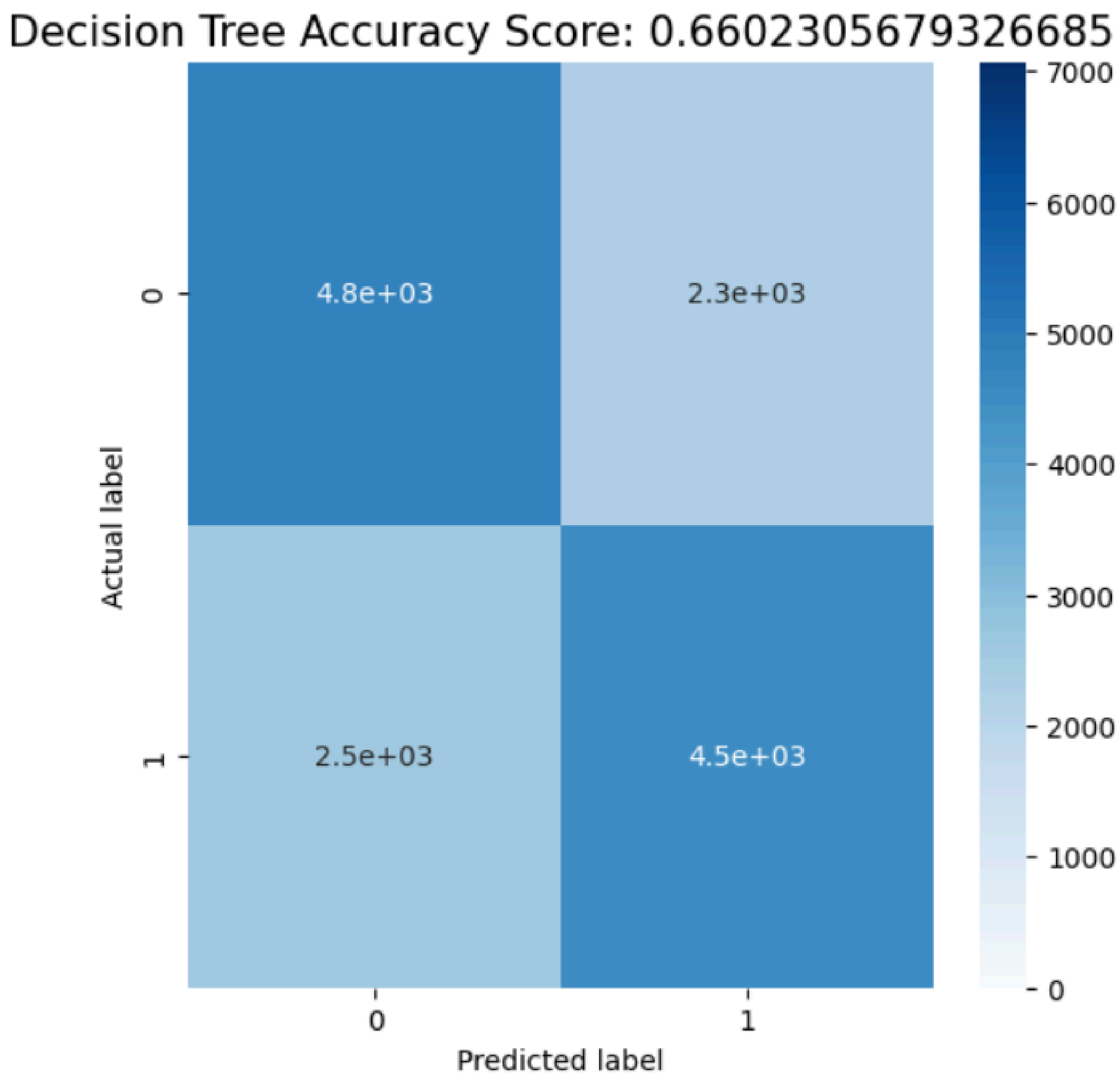
- Make predictions:

```python
y_pred_DT = pipeline_DT.predict(X_test)
```

- Create and plot Confusion Matrix:

```python
# Creating confusion matrix
confusion_matrix = pd.crosstab(y_test, y_pred_DT, rownames=['Actual'], colnames=['Predicted'])

# Plotting confusion matrix
plt.figure(figsize=(6, 6))
sb.heatmap(confusion_matrix, annot=True, vmin=0, vmax=int(len(y_test)/2), cmap='Blues')
tree_score = metrics.accuracy_score(y_test, y_pred_DT)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.title('Decision Tree Accuracy Score: {0}'.format(tree_score), size=15)
plt.show()
```

This is the result:



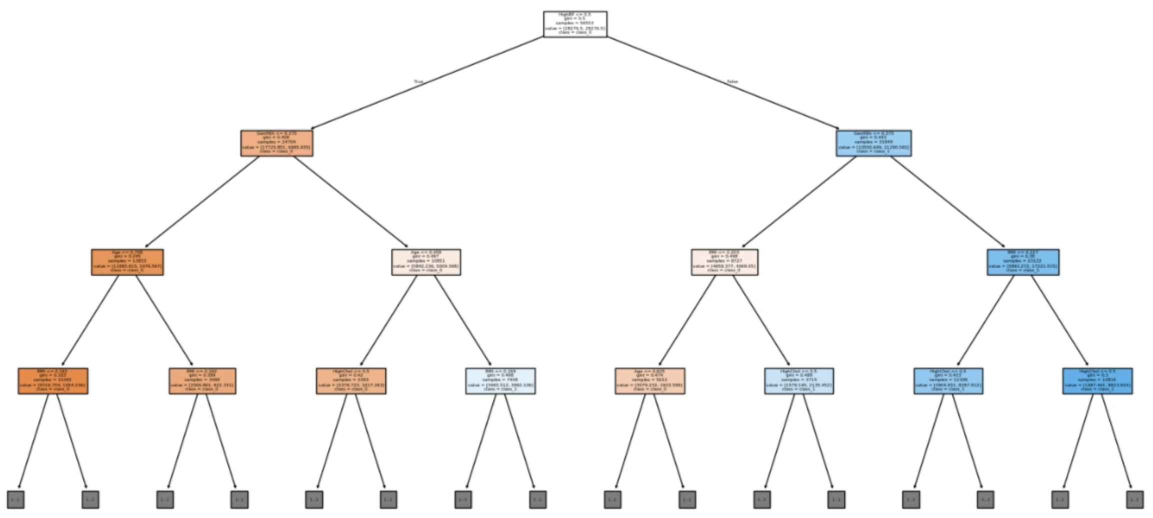
- Print Classification Report:

```python
# Printing classification report
report = classification_report(y_test, y_pred_DT)
print('Decision Tree classification report:\n', report)
```

```
Decision Tree classification report:
              precision    recall  f1-score   support

           0       0.66      0.68      0.67      7090
           1       0.67      0.64      0.65      7049

    accuracy                           0.66     14139
   macro avg       0.66      0.66      0.66     14139
weighted avg       0.66      0.66      0.66     14139
```

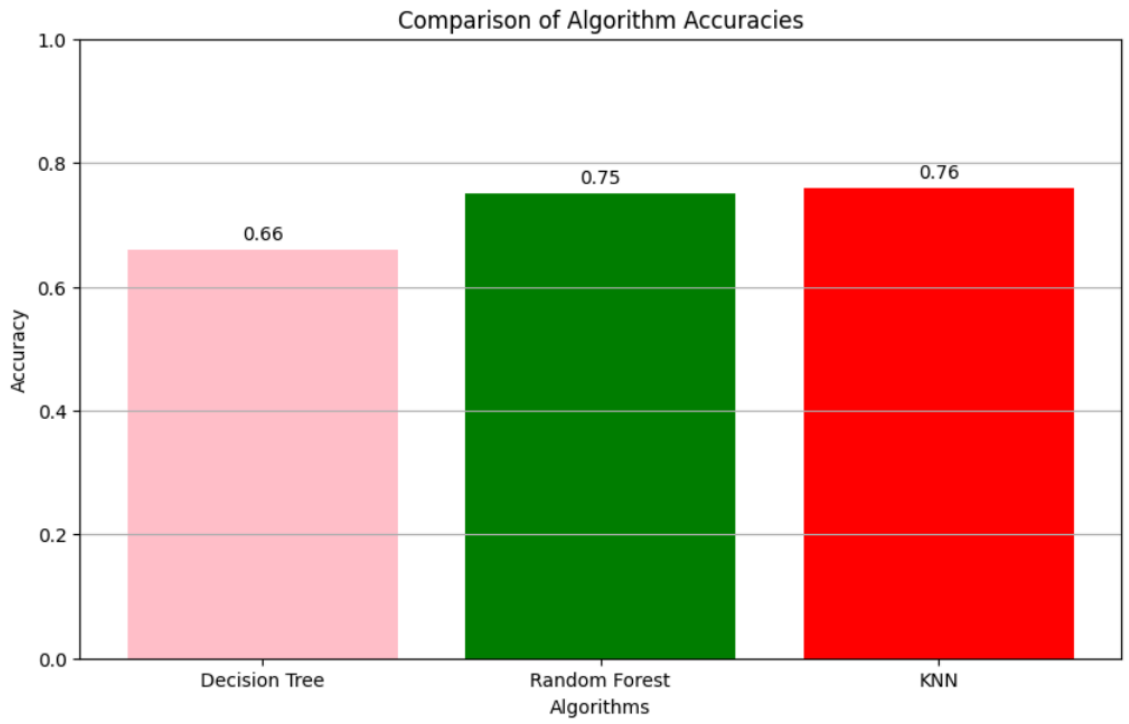
- Visualize Decision Tree:

```python
from sklearn.tree import plot_tree

# Visualizing the decision tree
plt.figure(figsize=(20,10))
plot_tree(pipeline_DT.named_steps['model'], feature_names=X_train.columns, class_names=['class_0', 'class_1'], filled=True, max_depth=3)
plt.show()
```



CHAPTER V: RESULT

1. Comparison



We use the histogram to compare the accuracy of the three algorithms.

Sorting accuracy in descending order: KNN > Random forest > Decision Tree.

2. Assessment

KNN:

- KNN has the highest accuracy among these three algorithms.
- This may indicate that KNN is doing a good job of classifying the likelihood of having diabetes in the dataset.
- However, KNN can have difficulty as the size of the data increases because it calculates the distance between data points, leading to higher computational costs.

Random forest:

- Random Forest has very close accuracy to KNN.
- Random Forest is a powerful algorithm and can handle large and feature-rich data sets without overfitting.
- Random Forest also provides estimates of the importance of features, which can be very useful in gaining insight into the data.

Decision Tree:

- Decision Tree has the lowest accuracy among the three algorithms.
- Decision Tree is a simple and straightforward algorithm, but it can be overfitting if not pruned properly.

- Despite its lower accuracy, Decision Trees can be useful in providing an intuitive view of how to make decisions based on data characteristics.

General assessment

- KNN had the highest accuracy, showing strong classification capabilities for this dataset.
- Random Forest also has good performance and is often favored for its ability to handle large and complex data as well as its better resistance to overfitting.
- Decision Tree has the lowest accuracy but is simple and easy to understand, suitable for problems that need an easy explanation of how to make decisions.

⇒ In short, Considering KNN and Random Forest for problems that require high accuracy. If the requirement requires an easy-to-understand and intuitive model, Decision Tree may be the right choice.

CHAPTER VI: CONCLUSION

1. **Advantages**
    - The dataset has been cleaned and balanced, containing no null values.
    - KNN: Accuracy after being normalized reaches the highest value.
    - Random forest: Accuracy is quite high (nearly equal to the accuracy of the KNN algorithm).
    - Decision Tree: Decision Tree models are easy to visualize and interpret, even for non-technical people.It often trains and predicts very quickly.
    - Work Rating: Know how to learn, data mining (pre-processing, classification by learned algorithms) evaluate results, work in groups, build models to predict by dataset.
2. **Disadvantages**
    - KNN: The running time of the algorithm is quite long and must be done many times to improve the accuracy of the algorithm.
    - Random forest: The algorithm's running time is relatively long and must be performed many times to improve the accuracy of the algorithm.
    - Decision Tree: The accuracy of this algorithm is the lowest among the three algorithms.
    - Work Rating: Because of the time in a semester, the team has not yet found a way to optimize for the highest accuracy, and has not clearly understood the Python language.Some members have poor work attitudes and miss deadlines.
3. **Development**
    - Re-implement algorithms many times, learn the latest methods to improve algorithm accuracy.
    - Learn and test other algorithms to find the most suitable and accurate algorithm.
    - Study algorithms to reduce columns in data preprocessing to choose which columns make the algorithm have the highest accuracy.

REFERENCES

[1] Theoretical lecture slides by Ph.D Cao Thi Nhan.

[2] Practical guide by MSc. Vu Minh Sang.

[3] Scikit – learn : https://scikit-learn.org/stable/

[4] Random forest knowledge: https://ibm.co/4e9r5v5

[5] KNN knowledge: https://ibm.co/3yF752W

[7] Link to dataset: Link dataset