Veronicah Kimani

MACHINE LEARNING

1. Introduction

We need machine learning to train a model. Machine learning, a subset of artificial intelligence and computer science, employs data and algorithms to enable artificial intelligence systems to mimic human learning processes, progressively enhancing their accuracy over time. In a general machine learning framework, the decision process revolves around utilizing algorithms to analyze input data, which can be either labeled or unlabeled, to make predictions or classifications regarding underlying patterns within the dataset. An error function serves to evaluate the model's predictions by comparing them to known examples, aiding in assessing the accuracy of the model. Subsequently, through a model optimization process, adjustments are made to the model's parameters, typically iteratively, aiming to minimize the discrepancy between predicted outcomes and actual observations. This iterative refinement continues autonomously until a predefined threshold of accuracy is attained.

1.1 Machine learning algorithm

A machine learning algorithm is a set of rules or process used by AI to perform a given task. It is like a recipe for an AI system. It tells the system how to analyze data, find patterns, and make predictions. By following these rules, the AI learns from the data it's given, helping it to make better decisions or predictions in the future.

1.2 Model Training in Machine Learning

Model training in machine learning is like teaching a computer by showing it lots of examples. We give the computer a bunch of data and let it figure out the best way to make predictions or decisions based on that data. It keeps adjusting and learning from its mistakes until it gets really good at it.

1.3 Types Of Machine Learning

There are three types of machine learning;

- **Supervised machine learning**. This involves training a model on a labeled dataset where each example has a correct answer.
- **Unsupervised machine learning.** This model learns from data that isn't labeled. Tries to find patterns and relationships in the data on its own. No target variable.
- **Reinforcement machine learning**. This model involves training agents to make decisions by trying things out in an environment. Agents learn through trial and error, getting feedback in the form of rewards or penalties.

1.3.1 Supervised machine learning

The model learns to make predictions by understanding patterns in the labeled data .In this we have the target variable. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.

Supervised learning is used in image recognition, sentiment analysis, detecting if an email is a spam or not and diagnostics analysis.

It is also classified into two.

- ✓ Regression. Used where there is a relationship between the input variable(s) and output variable. It applies to numerical / categorical variables. For example predicting the performance of students at school, house prices, predicting temperatures etc.
- ✓ Classification. This applies where the output variable is categorical. Example is detecting if an email is spam or not, classifying images like for example crocodile or alligator.

1.3.2 Supervised machine learning Algorithm

- ➢ Logistic Regression: Helps determine if an input belongs to a specific group or not.
- ➢ Support Vector Machines (SVM): Creates coordinates for objects in an n-dimensional space and utilizes a hyperplane to group objects by shared characteristics.

- Naive Bayes: Assumes variable independence and classifies objects based on feature probabilities.
- Decision Trees: Classifies inputs by navigating through the tree's nodes to determine their category.
- Linear Regression: Identifies relationships between the variable of interest and inputs to predict values.
- k Nearest Neighbors (kNN): Groups the closest objects and determines their common characteristics.
- Random Forest: Comprises multiple decision trees from random data subsets for more accurate predictions.
- Boosting algorithms: Ensemble learning techniques (e.g., Gradient Boosting Machine, XGBoost, LightGBM) that combine predictions from various algorithms, accounting for previous algorithm errors.

1.3.3 Describing Machine learning algorithms

2. Machine learning on diabetics data

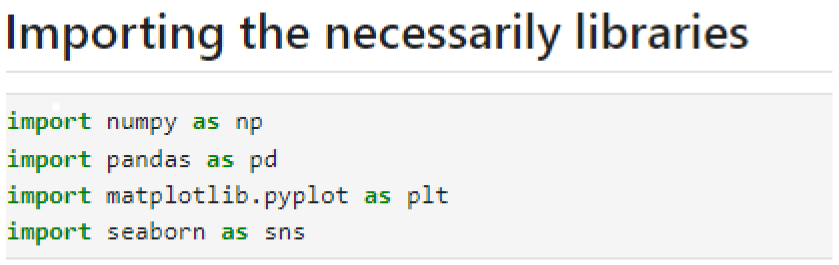
2.1 Objective

The main aim is to predict whether or not a patient is diabetic, based on certain diagnostic measurements included in the dataset.

2.2 Implementation

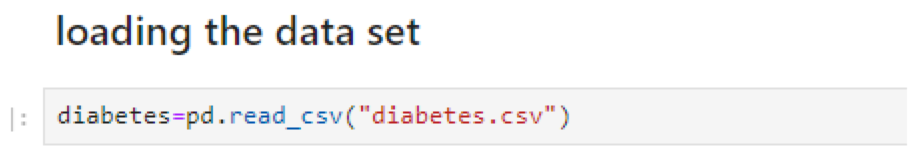
To achieve our objective, we must undertake several tasks. Firstly, we will conduct data preprocessing and cleaning to prepare the data for model training.

2.2.1 loading the dataset



Importing the necessarily libraries

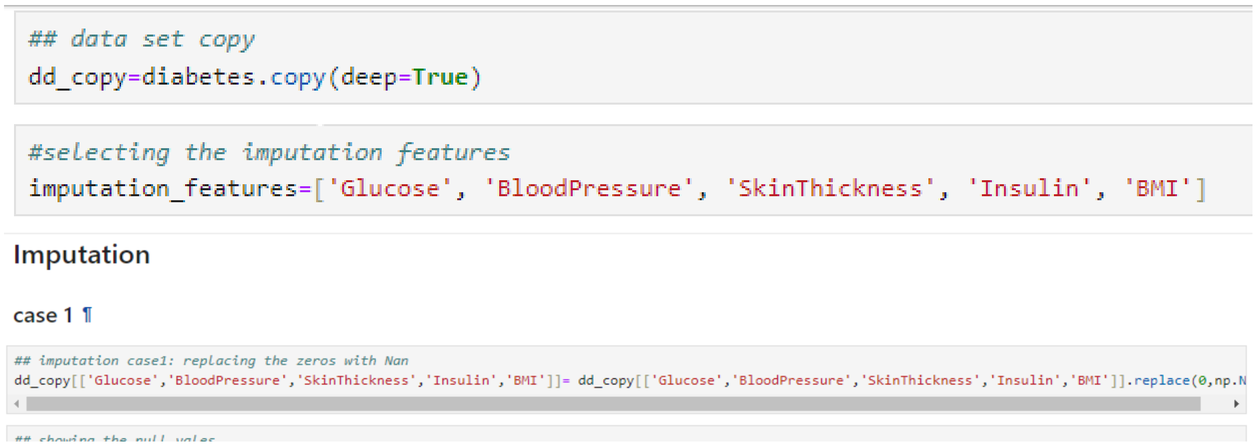
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```



loading the data set

```
[ ]: diabetes=pd.read_csv("diabetes.csv")
```

2.2.2 Data preprocessing



```
## data set copy
dd_copy=diabetes.copy(deep=True)
```

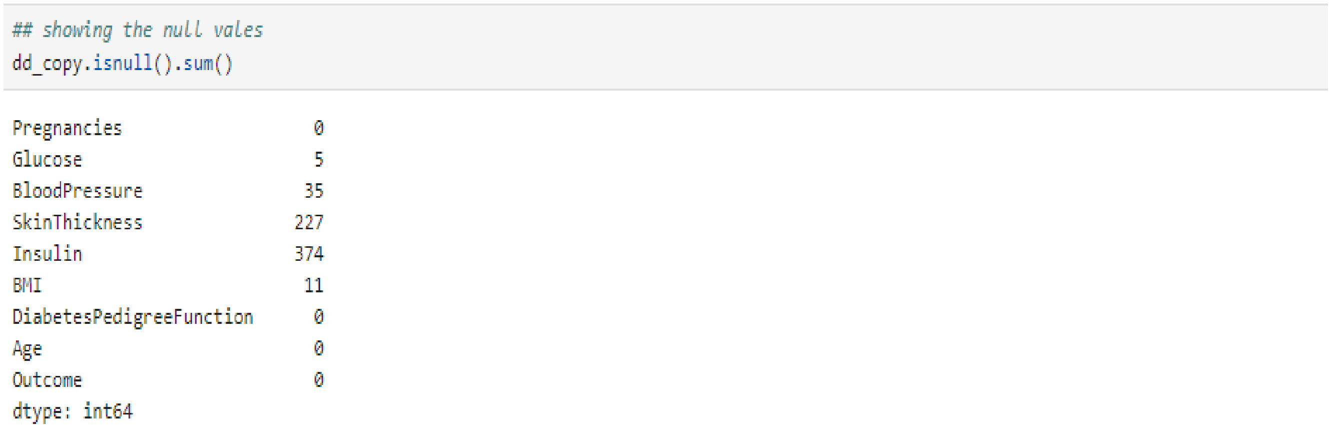
```
#selecting the imputation features
imputation_features=['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
```

Imputation

case 1 ¶

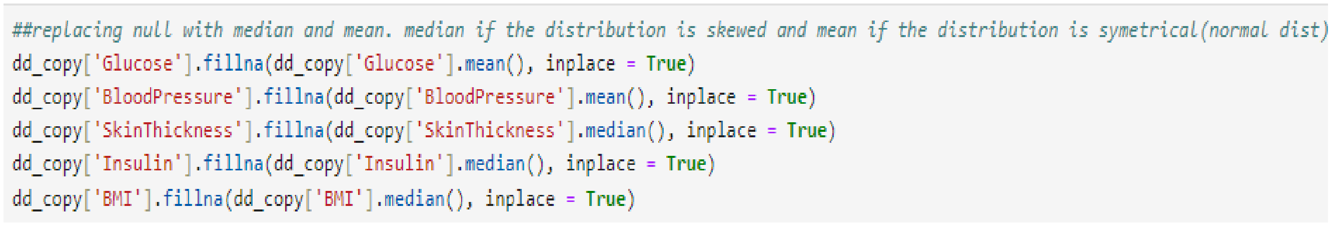
```
## imputation case1: replacing the zeros with Nan
dd_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]= dd_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.N
```

```
## showing the null vales
```

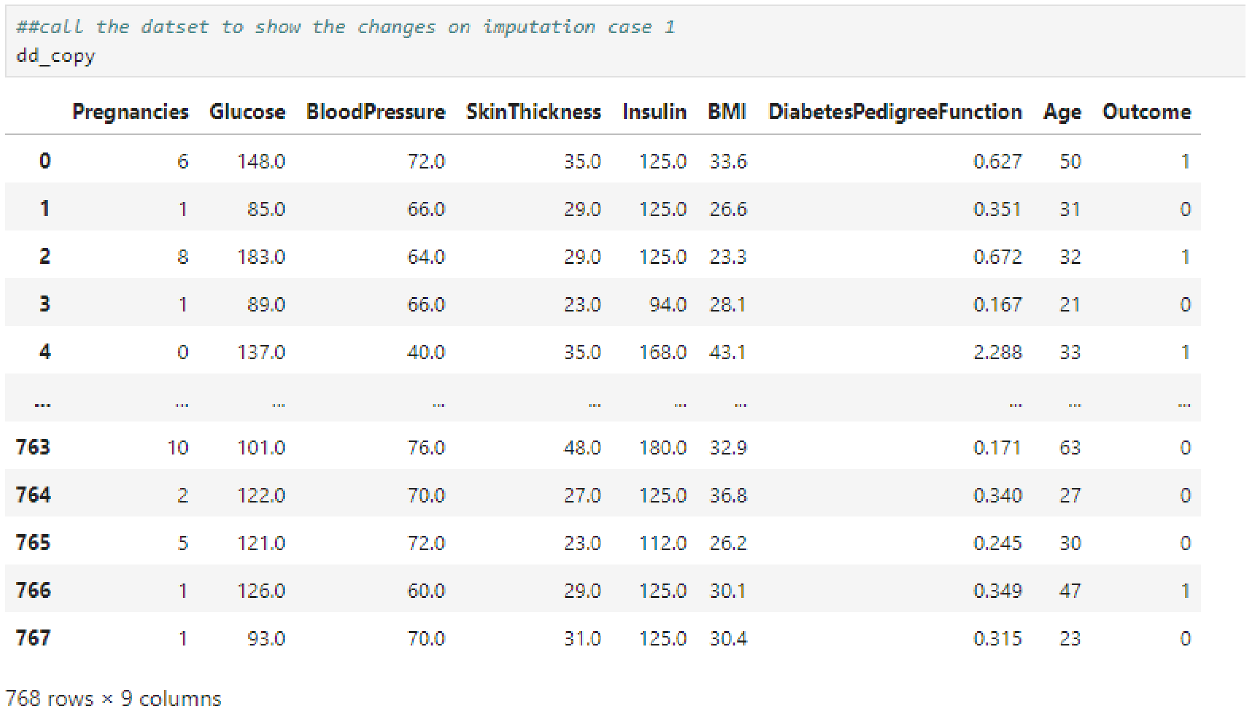


```
## showing the null vales
dd_copy.isnull().sum()
```

```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```



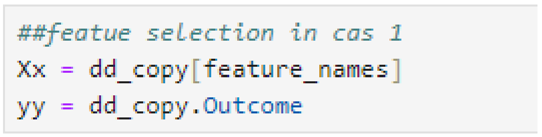
```
##replacing null with median and mean. median if the distribution is skewed and mean if the distribution is symetrical(normal dist)
dd_copy['Glucose'].fillna(dd_copy['Glucose'].mean(), inplace = True)
dd_copy['BloodPressure'].fillna(dd_copy['BloodPressure'].mean(), inplace = True)
dd_copy['SkinThickness'].fillna(dd_copy['SkinThickness'].median(), inplace = True)
dd_copy['Insulin'].fillna(dd_copy['Insulin'].median(), inplace = True)
dd_copy['BMI'].fillna(dd_copy['BMI'].median(), inplace = True)
```



```
##call the datset to show the changes on imputation case 1
dd_copy
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.0	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.0	29.0	125.0	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101.0	76.0	48.0	180.0	32.9	0.171	63	0
764	2	122.0	70.0	27.0	125.0	36.8	0.340	27	0
765	5	121.0	72.0	23.0	112.0	26.2	0.245	30	0
766	1	126.0	60.0	29.0	125.0	30.1	0.349	47	1
767	1	93.0	70.0	31.0	125.0	30.4	0.315	23	0

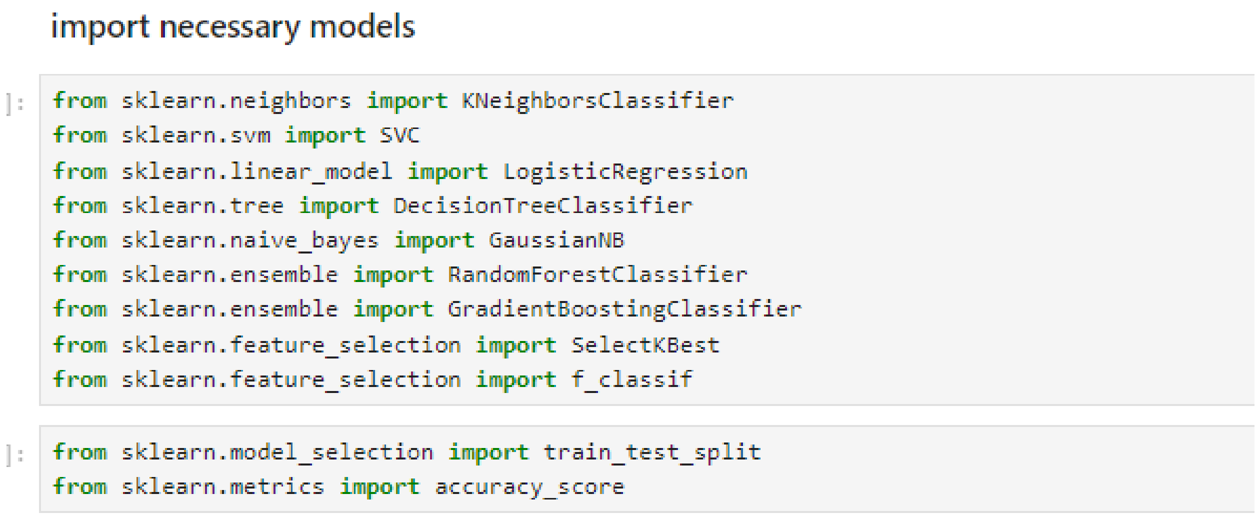
768 rows × 9 columns



```
##featue selection in cas 1
Xx = dd_copy[feature_names]
yy = dd_copy.Outcome
```

2.2.3 model training

We import libraries for model training

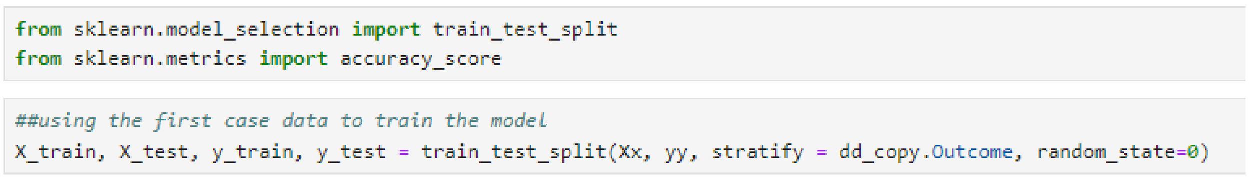


import necessary models

```
]: from sklearn.neighbors import KNeighborsClassifier
   from sklearn.svm import SVC
   from sklearn.linear_model import LogisticRegression
   from sklearn.tree import DecisionTreeClassifier
   from sklearn.naive_bayes import GaussianNB
   from sklearn.ensemble import RandomForestClassifier
   from sklearn.ensemble import GradientBoostingClassifier
   from sklearn.feature_selection import SelectKBest
   from sklearn.feature_selection import f_classif
```

```
]: from sklearn.model_selection import train_test_split
   from sklearn.metrics import accuracy_score
```

Before training the data we split the data into training data and test data.

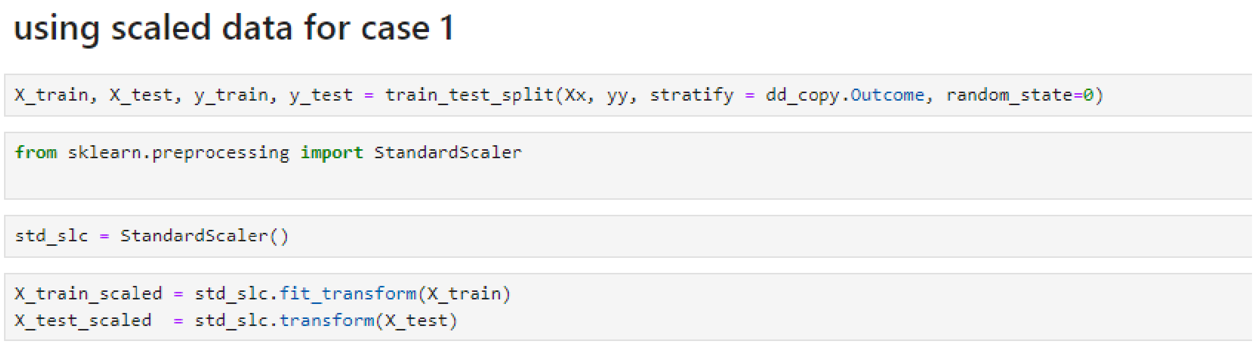


```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
##using the first case data to train the model
X_train, X_test, y_train, y_test = train_test_split(Xx, yy, stratify = dd_copy.Outcome, random_state=0)
```

2.3.4 Data Normalization

Machine learning algorithms are influenced by the ranges of data. This is to ensure that no single feature has undue influence by standardizing the distance calculations used by the algorithm. Scaling methods, such as standardization, normalization, and Min-max scaling, are employed to achieve this goal.



using scaled data for case 1

```
X_train, X_test, y_train, y_test = train_test_split(Xx, yy, stratify = dd_copy.Outcome, random_state=0)
```

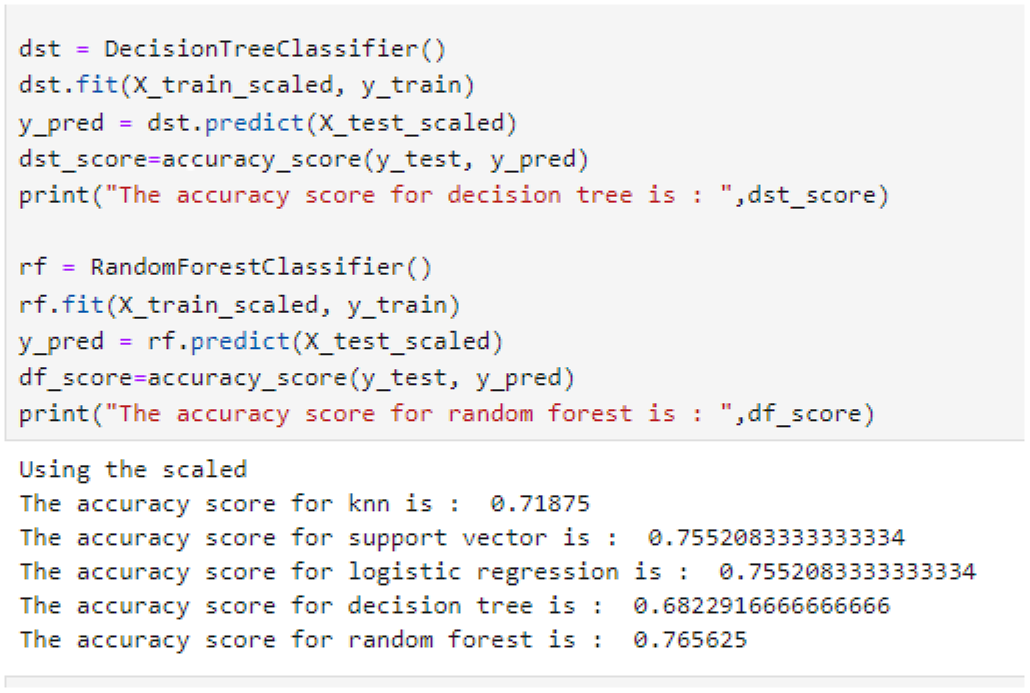
```
from sklearn.preprocessing import StandardScaler
```

```
std_slc = StandardScaler()
```

```
X_train_scaled = std_slc.fit_transform(X_train)
X_test_scaled  = std_slc.transform(X_test)
```

2.3.5 Model Training

```
print("Using the scaled")
knn = KNeighborsClassifier()
knn.fit(X_train_scaled, y_train)
y_pred = knn.predict(X_test_scaled)
knn_score=accuracy_score(y_test, y_pred)
print("The accuracy score for knn is : ",knn_score)

svm = SVC(C=0.5, kernel='linear')
svm.fit(X_train_scaled, y_train)
y_pred = svm.predict(X_test_scaled)
svm_score=accuracy_score(y_test, y_pred)
print("The accuracy score for support vector is : ",svm_score)

logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train_scaled, y_train)
y_pred = logreg_model.predict(X_test_scaled)
logreg_score=accuracy_score(y_test, y_pred)
print("The accuracy score for logistic regression is : ",logreg_score)

dst = DecisionTreeClassifier()
dst.fit(X_train_scaled, y_train)
y_pred = dst.predict(X_test_scaled)
dst_score=accuracy_score(y_test, y_pred)
print("The accuracy score for decision tree is : ",dst_score)
```



```
dst = DecisionTreeClassifier()
dst.fit(X_train_scaled, y_train)
y_pred = dst.predict(X_test_scaled)
dst_score=accuracy_score(y_test, y_pred)
print("The accuracy score for decision tree is : ",dst_score)

rf = RandomForestClassifier()
rf.fit(X_train_scaled, y_train)
y_pred = rf.predict(X_test_scaled)
df_score=accuracy_score(y_test, y_pred)
print("The accuracy score for random forest is : ",df_score)
```

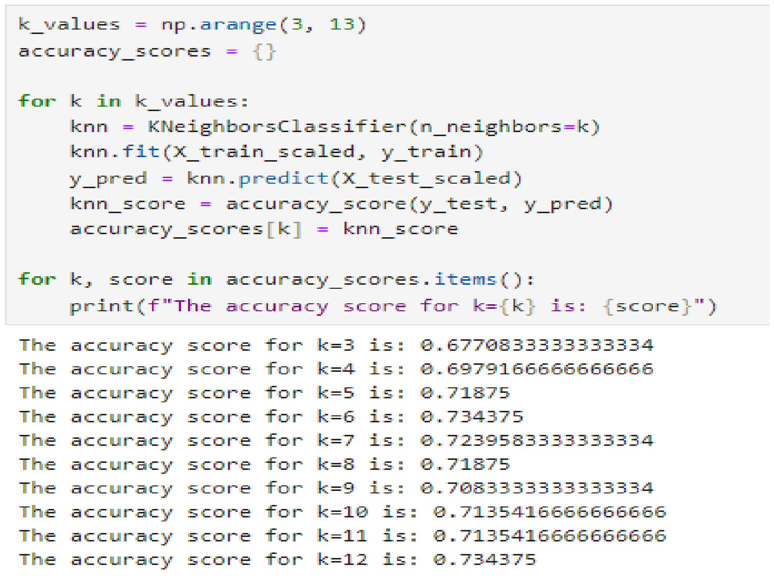
```
Using the scaled
The accuracy score for knn is :  0.71875
The accuracy score for support vector is :  0.7552083333333334
The accuracy score for logistic regression is :  0.7552083333333334
The accuracy score for decision tree is :  0.6822916666666666
The accuracy score for random forest is :  0.765625
```

Random forest has the highest accuracy score. This implies that it’s the best model as at this point.

2.3.6 Model Tuning

This is where we tune in hyper parameters to improve the accuracy scores

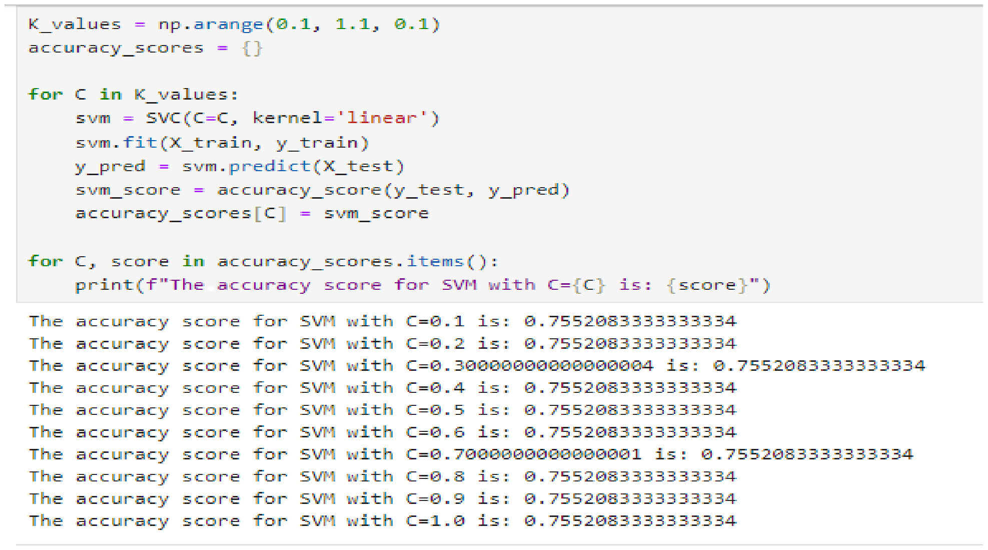
In the following code, we are changing the values of n_neighors from 3 to 13



```
k_values = np.arange(3, 13)
accuracy_scores = {}

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)
    knn_score = accuracy_score(y_test, y_pred)
    accuracy_scores[k] = knn_score

for k, score in accuracy_scores.items():
    print(f"The accuracy score for k={k} is: {score}")
```

```
The accuracy score for k=3 is: 0.6770833333333334
The accuracy score for k=4 is: 0.6979166666666666
The accuracy score for k=5 is: 0.71875
The accuracy score for k=6 is: 0.734375
The accuracy score for k=7 is: 0.7239583333333334
The accuracy score for k=8 is: 0.71875
The accuracy score for k=9 is: 0.7083333333333334
The accuracy score for k=10 is: 0.7135416666666666
The accuracy score for k=11 is: 0.7135416666666666
The accuracy score for k=12 is: 0.734375
```

The accuracy scores increases as value of n neighbor increases.

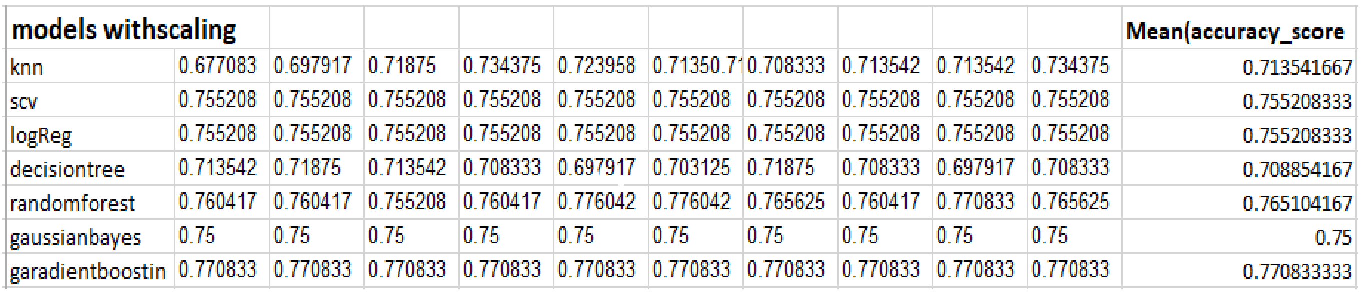


```
K_values = np.arange(0.1, 1.1, 0.1)
accuracy_scores = {}

for C in K_values:
    svm = SVC(C=C, kernel='linear')
    svm.fit(X_train, y_train)
    y_pred = svm.predict(X_test)
    svm_score = accuracy_score(y_test, y_pred)
    accuracy_scores[C] = svm_score

for C, score in accuracy_scores.items():
    print(f"The accuracy score for SVM with C={C} is: {score}")
```

```
The accuracy score for SVM with C=0.1 is: 0.7552083333333334
The accuracy score for SVM with C=0.2 is: 0.7552083333333334
The accuracy score for SVM with C=0.30000000000000004 is: 0.7552083333333334
The accuracy score for SVM with C=0.4 is: 0.7552083333333334
The accuracy score for SVM with C=0.5 is: 0.7552083333333334
The accuracy score for SVM with C=0.6 is: 0.7552083333333334
The accuracy score for SVM with C=0.7000000000000001 is: 0.7552083333333334
The accuracy score for SVM with C=0.8 is: 0.7552083333333334
The accuracy score for SVM with C=0.9 is: 0.7552083333333334
The accuracy score for SVM with C=1.0 is: 0.7552083333333334
```

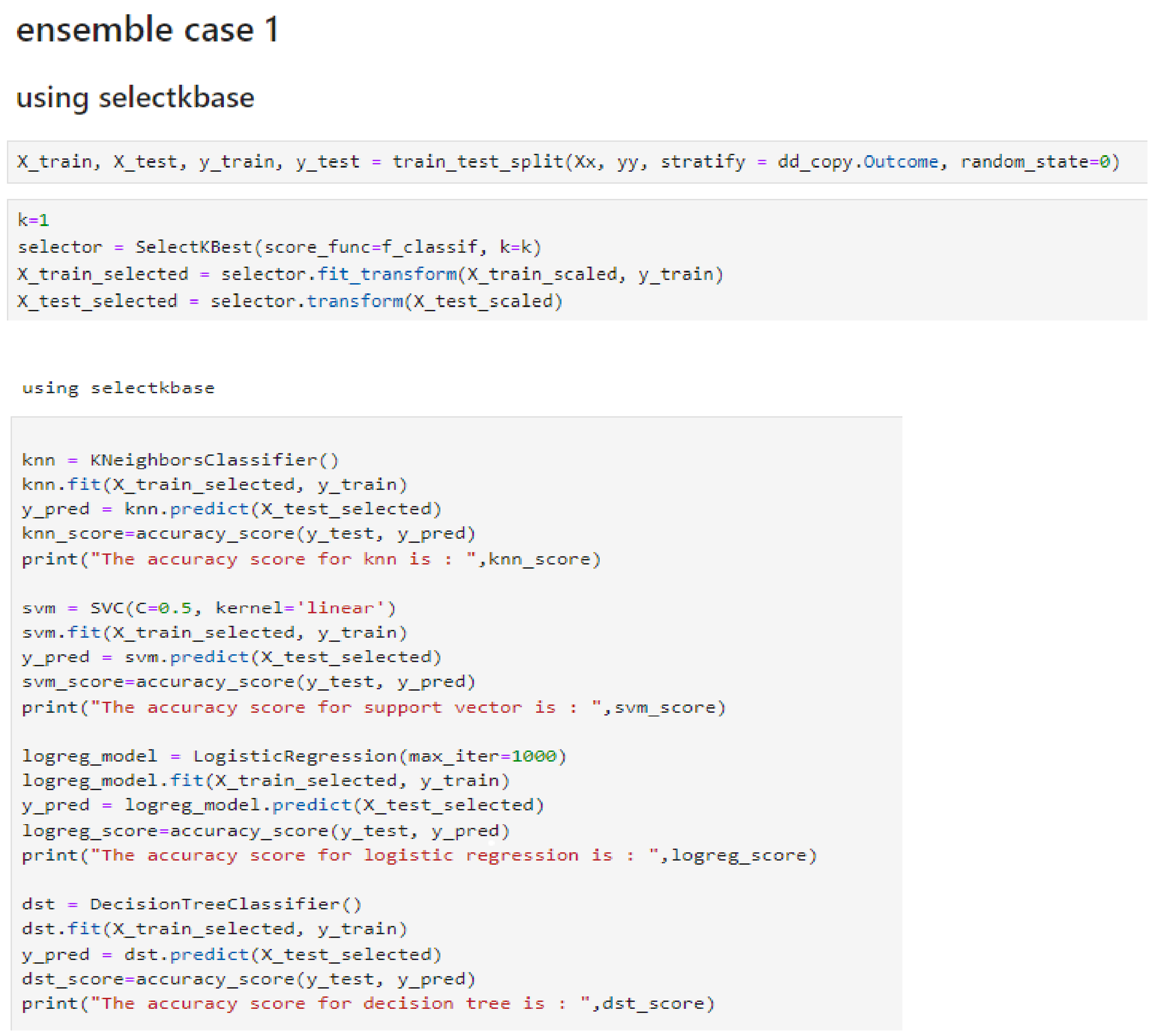
This is the same for all the c’s

Veronicah Kimani



models withscaling											Mean(accuracy_score
knn	0.677083	0.697917	0.71875	0.734375	0.723958	0.71350.71	0.708333	0.713542	0.713542	0.734375	0.713541667
scv	0.755208	0.755208	0.755208	0.755208	0.755208	0.755208	0.755208	0.755208	0.755208	0.755208	0.755208333
logReg	0.755208	0.755208	0.755208	0.755208	0.755208	0.755208	0.755208	0.755208	0.755208	0.755208	0.755208333
decisiontree	0.713542	0.71875	0.713542	0.708333	0.697917	0.703125	0.71875	0.708333	0.697917	0.708333	0.708854167
randomforest	0.760417	0.760417	0.755208	0.760417	0.776042	0.776042	0.765625	0.760417	0.770833	0.765625	0.765104167
gaussianbayes	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
garadientboostin	0.770833	0.770833	0.770833	0.770833	0.770833	0.770833	0.770833	0.770833	0.770833	0.770833	0.770833333

Model training using selectkbest metric



ensemble case 1

using selectkbase

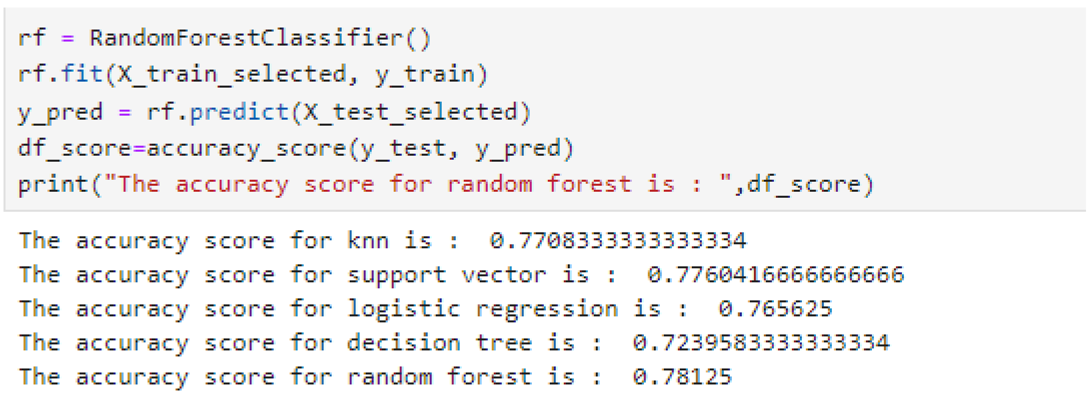
```
X_train, X_test, y_train, y_test = train_test_split(Xx, yy, stratify = dd_copy.Outcome, random_state=0)
```

```
k=1
selector = SelectKBest(score_func=f_classif, k=k)
X_train_selected = selector.fit_transform(X_train_scaled, y_train)
X_test_selected = selector.transform(X_test_scaled)
```

```
using selectkbase
```

```
knn = KNeighborsClassifier()
knn.fit(X_train_selected, y_train)
y_pred = knn.predict(X_test_selected)
knn_score=accuracy_score(y_test, y_pred)
print("The accuracy score for knn is : ",knn_score)

svm = SVC(C=0.5, kernel='linear')
svm.fit(X_train_selected, y_train)
y_pred = svm.predict(X_test_selected)
svm_score=accuracy_score(y_test, y_pred)
print("The accuracy score for support vector is : ",svm_score)

logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train_selected, y_train)
y_pred = logreg_model.predict(X_test_selected)
logreg_score=accuracy_score(y_test, y_pred)
print("The accuracy score for logistic regression is : ",logreg_score)

dst = DecisionTreeClassifier()
dst.fit(X_train_selected, y_train)
y_pred = dst.predict(X_test_selected)
dst_score=accuracy_score(y_test, y_pred)
print("The accuracy score for decision tree is : ",dst_score)
```

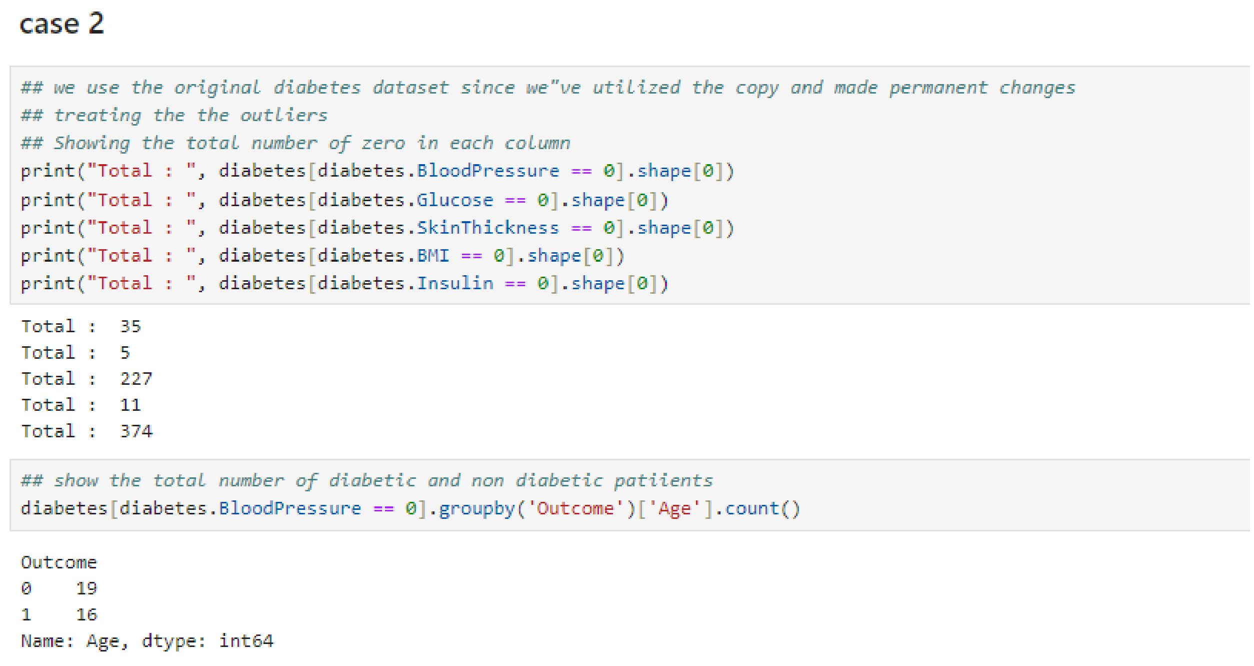
Veronicah Kimani



```
rf = RandomForestClassifier()
rf.fit(X_train_selected, y_train)
y_pred = rf.predict(X_test_selected)
df_score=accuracy_score(y_test, y_pred)
print("The accuracy score for random forest is : ",df_score)
```

```
The accuracy score for knn is :  0.7708333333333334
The accuracy score for support vector is :  0.7760416666666666
The accuracy score for logistic regression is :  0.765625
The accuracy score for decision tree is :  0.7239583333333334
The accuracy score for random forest is :  0.78125
```

Model training case 2.



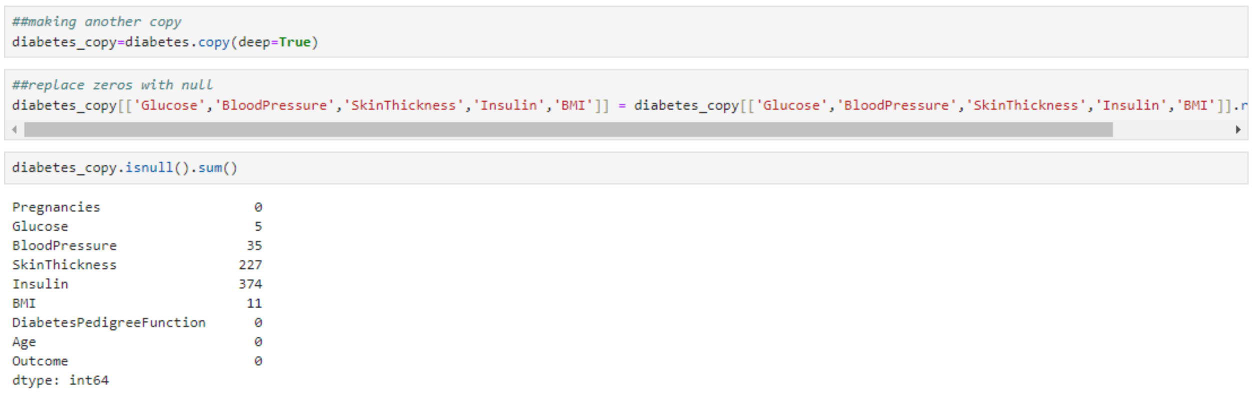
case 2

```
## we use the original diabetes dataset since we"ve utilized the copy and made permanent changes
## treating the the outliers
## Showing the total number of zero in each column
print("Total : ", diabetes[diabetes.BloodPressure == 0].shape[0])
print("Total : ", diabetes[diabetes.Glucose == 0].shape[0])
print("Total : ", diabetes[diabetes.SkinThickness == 0].shape[0])
print("Total : ", diabetes[diabetes.BMI == 0].shape[0])
print("Total : ", diabetes[diabetes.Insulin == 0].shape[0])
```

```
Total :  35
Total :  5
Total :  227
Total :  11
Total :  374
```

```
## show the total number of diabetic and non diabetic patiients
diabetes[diabetes.BloodPressure == 0].groupby('Outcome')['Age'].count()
```

```
Outcome
0    19
1    16
Name: Age, dtype: int64
```

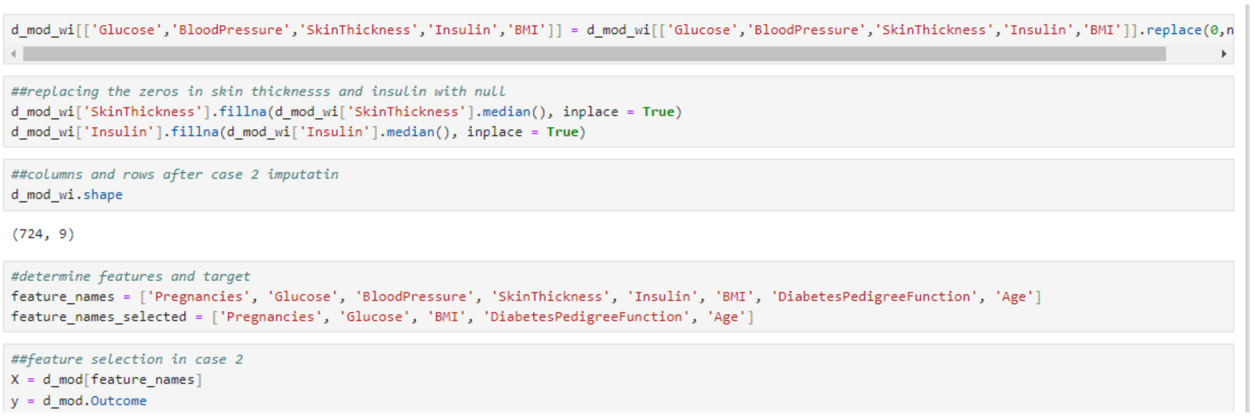


```
##making another copy
diabetes_copy=diabetes.copy(deep=True)
```

```
##replace zeros with null
diabetes_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = diabetes_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].r
```

```
diabetes_copy.isnull().sum()
```

```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```



```
d_mod_wi[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = d_mod_wi[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,n
```

```
##replacing the zeros in skin thicknesss and insulin with null
d_mod_wi['SkinThickness'].fillna(d_mod_wi['SkinThickness'].median(), inplace = True)
d_mod_wi['Insulin'].fillna(d_mod_wi['Insulin'].median(), inplace = True)
```

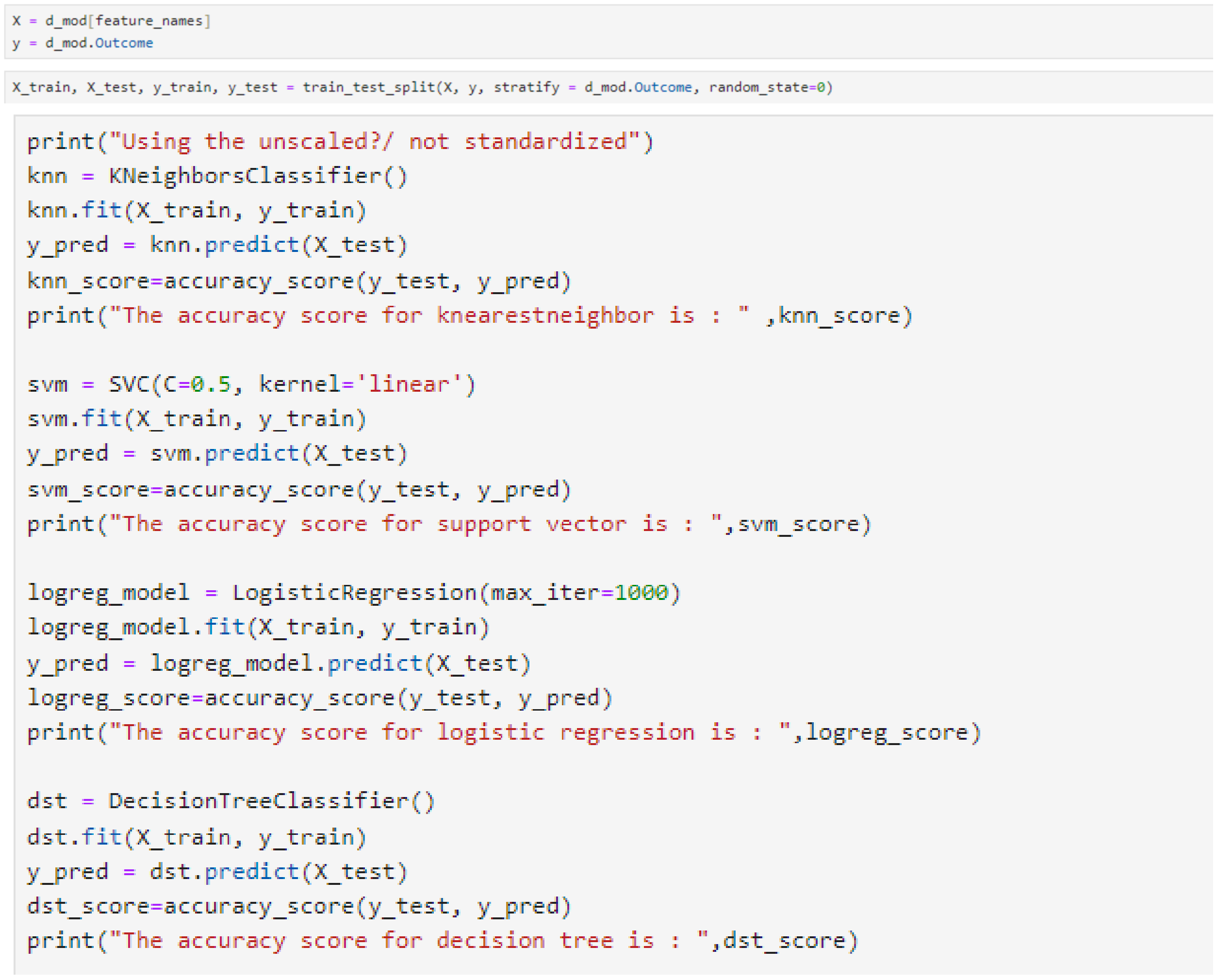
```
##columns and rows after case 2 imputatin
d_mod_wi.shape
```

```
(724, 9)
```

```
#determine features and target
feature_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
feature_names_selected = ['Pregnancies', 'Glucose', 'BMI', 'DiabetesPedigreeFunction', 'Age']
```

```
##feature selection in case 2
X = d_mod[feature_names]
y = d_mod.Outcome
```

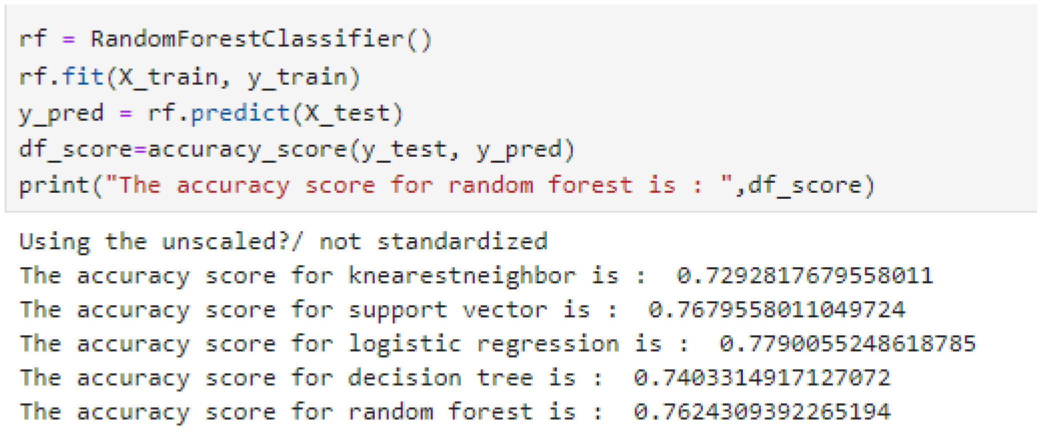
Training the model for case 2



```
X = d_mod[feature_names]
y = d_mod.Outcome
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = d_mod.Outcome, random_state=0)
```

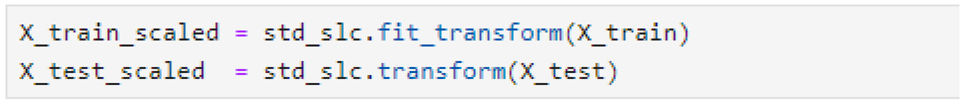
```
print("Using the unscaled?/ not standardized")
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
knn_score=accuracy_score(y_test, y_pred)
print("The accuracy score for knearestneighbor is : " ,knn_score)

svm = SVC(C=0.5, kernel='linear')
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
svm_score=accuracy_score(y_test, y_pred)
print("The accuracy score for support vector is : ",svm_score)

logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train, y_train)
y_pred = logreg_model.predict(X_test)
logreg_score=accuracy_score(y_test, y_pred)
print("The accuracy score for logistic regression is : ",logreg_score)

dst = DecisionTreeClassifier()
dst.fit(X_train, y_train)
y_pred = dst.predict(X_test)
dst_score=accuracy_score(y_test, y_pred)
print("The accuracy score for decision tree is : ",dst_score)
```



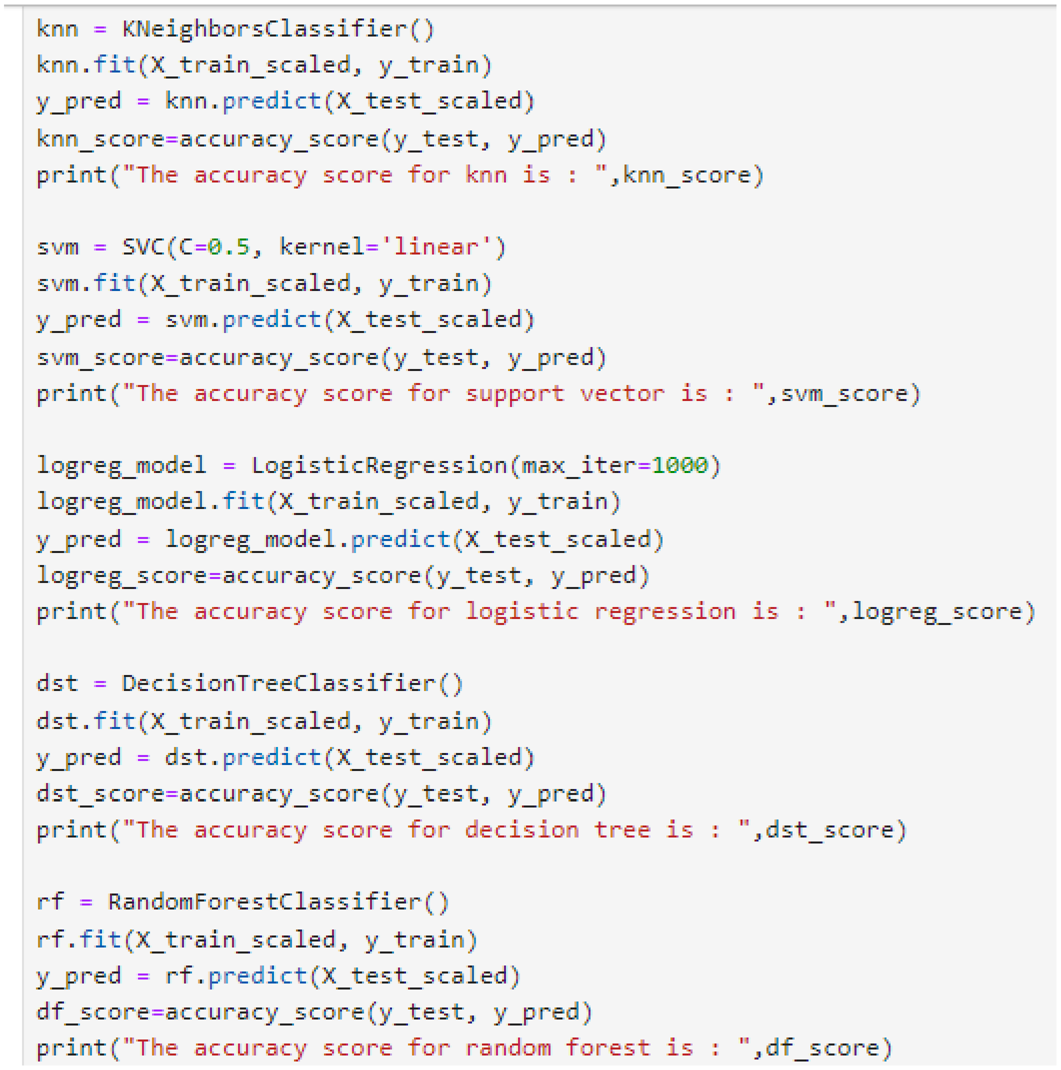
```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
df_score=accuracy_score(y_test, y_pred)
print("The accuracy score for random forest is : ",df_score)
```

```
Using the unscaled?/ not standardized
The accuracy score for knearestneighbor is :  0.7292817679558011
The accuracy score for support vector is :  0.7679558011049724
The accuracy score for logistic regression is :  0.7790055248618785
The accuracy score for decision tree is :  0.7403314917127072
The accuracy score for random forest is :  0.7624309392265194
```

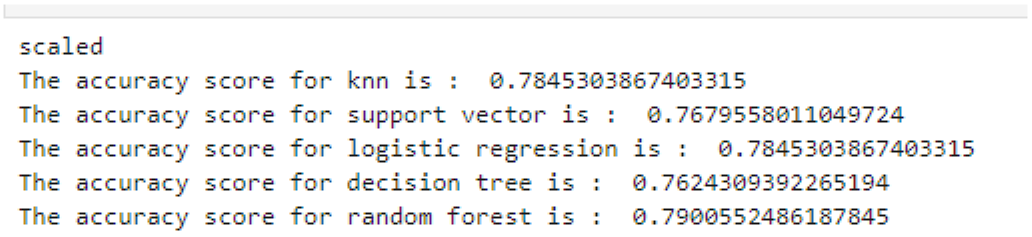
Using scaled data



```
X_train_scaled = std_slc.fit_transform(X_train)
X_test_scaled  = std_slc.transform(X_test)
```



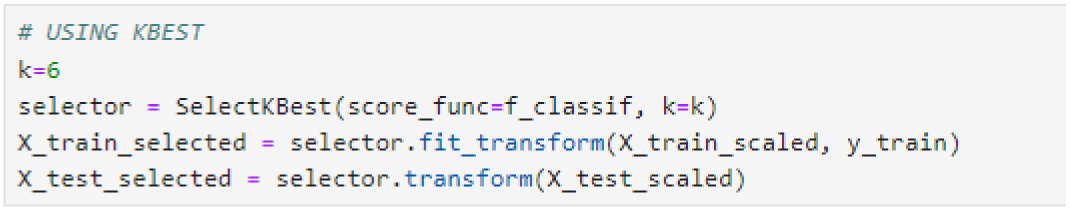
```
knn = KNeighborsClassifier()
knn.fit(X_train_scaled, y_train)
y_pred = knn.predict(X_test_scaled)
knn_score=accuracy_score(y_test, y_pred)
print("The accuracy score for knn is : ",knn_score)

svm = SVC(C=0.5, kernel='linear')
svm.fit(X_train_scaled, y_train)
y_pred = svm.predict(X_test_scaled)
svm_score=accuracy_score(y_test, y_pred)
print("The accuracy score for support vector is : ",svm_score)

logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train_scaled, y_train)
y_pred = logreg_model.predict(X_test_scaled)
logreg_score=accuracy_score(y_test, y_pred)
print("The accuracy score for logistic regression is : ",logreg_score)

dst = DecisionTreeClassifier()
dst.fit(X_train_scaled, y_train)
y_pred = dst.predict(X_test_scaled)
dst_score=accuracy_score(y_test, y_pred)
print("The accuracy score for decision tree is : ",dst_score)

rf = RandomForestClassifier()
rf.fit(X_train_scaled, y_train)
y_pred = rf.predict(X_test_scaled)
df_score=accuracy_score(y_test, y_pred)
print("The accuracy score for random forest is : ",df_score)
```



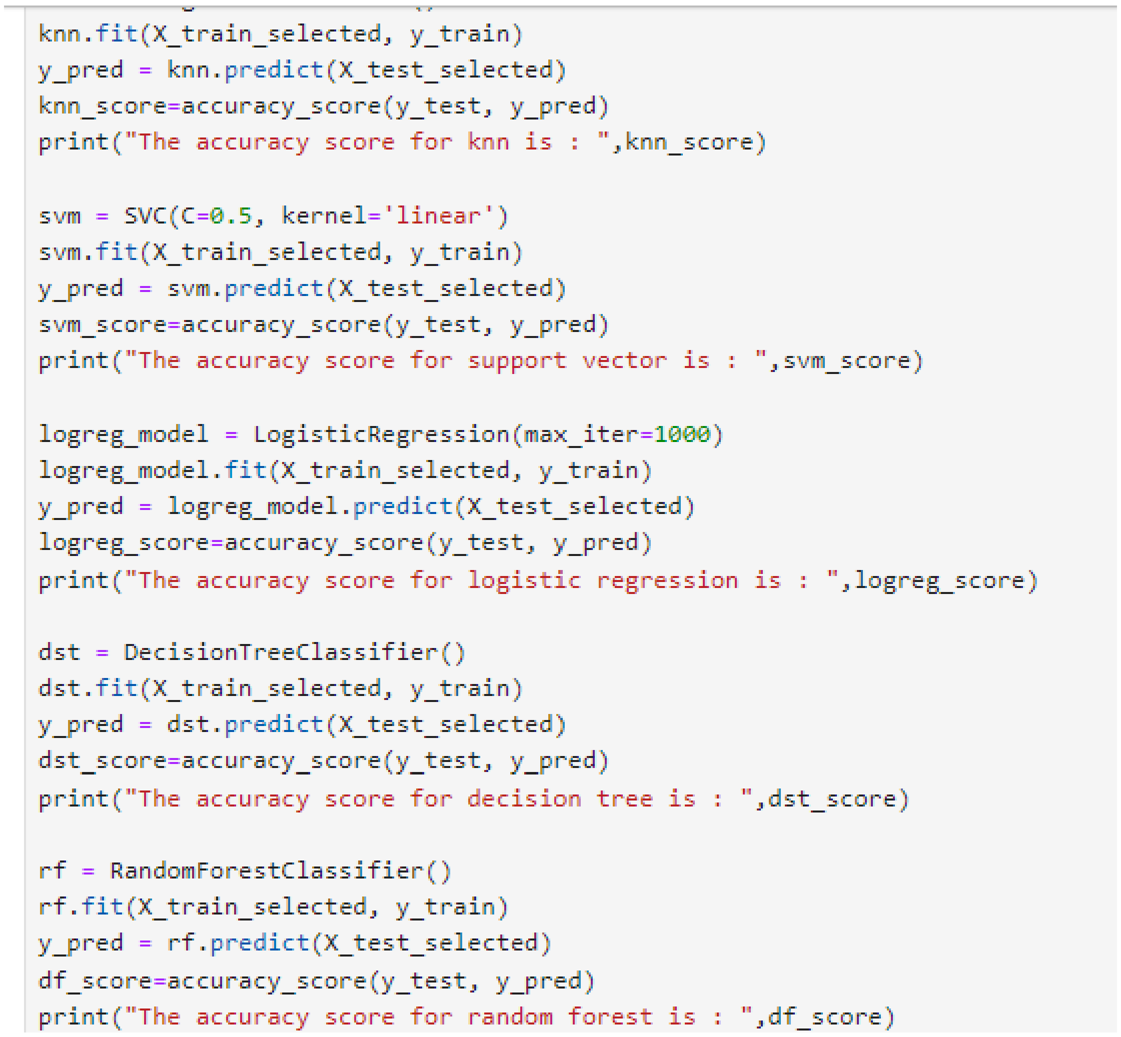
```
scaled
The accuracy score for knn is :  0.7845303867403315
The accuracy score for support vector is :  0.7679558011049724
The accuracy score for logistic regression is :  0.7845303867403315
The accuracy score for decision tree is :  0.7624309392265194
The accuracy score for random forest is :  0.7900552486187845
```

With scaled data, the random forest algorithm is the highest at 79% followed by knn and logistic regression.

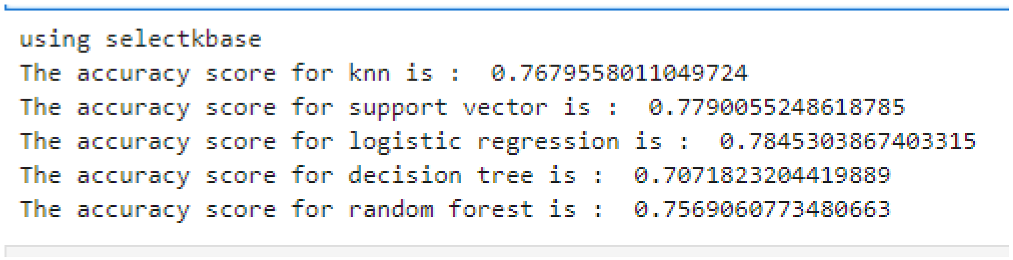
Using selectkbest



```
# USING KBEST
k=6
selector = SelectKBest(score_func=f_classif, k=k)
X_train_selected = selector.fit_transform(X_train_scaled, y_train)
X_test_selected = selector.transform(X_test_scaled)
```



```
knn.fit(X_train_selected, y_train)
y_pred = knn.predict(X_test_selected)
knn_score=accuracy_score(y_test, y_pred)
print("The accuracy score for knn is : ",knn_score)

svm = SVC(C=0.5, kernel='linear')
svm.fit(X_train_selected, y_train)
y_pred = svm.predict(X_test_selected)
svm_score=accuracy_score(y_test, y_pred)
print("The accuracy score for support vector is : ",svm_score)

logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train_selected, y_train)
y_pred = logreg_model.predict(X_test_selected)
logreg_score=accuracy_score(y_test, y_pred)
print("The accuracy score for logistic regression is : ",logreg_score)

dst = DecisionTreeClassifier()
dst.fit(X_train_selected, y_train)
y_pred = dst.predict(X_test_selected)
dst_score=accuracy_score(y_test, y_pred)
print("The accuracy score for decision tree is : ",dst_score)

rf = RandomForestClassifier()
rf.fit(X_train_selected, y_train)
y_pred = rf.predict(X_test_selected)
df_score=accuracy_score(y_test, y_pred)
print("The accuracy score for random forest is : ",df_score)
```



```
using selectkbase
The accuracy score for knn is :  0.7679558011049724
The accuracy score for support vector is :  0.7790055248618785
The accuracy score for logistic regression is :  0.7845303867403315
The accuracy score for decision tree is :  0.7071823204419889
The accuracy score for random forest is :  0.7569060773480663
```

The logistic regression gives the highest score with k=6, followed by support vector

machine algorithm with an accuracy score of 78.45

We keep adjusting the k for at least 5 times to determine the model that gives the highest accuracy score of 80%