Diabetes classification

Team Members	SEC
1-Ahmed Mohamed Ali	1
2-Assem Saber Mohamed	3
3-Rawan Abdel-Aziz Ahmed	3
4-Aya Magdy Galal	2
5-Reham Mostafa Ali	3
6-Sara Reda Moatamed	3

Department AI

OverView

- our dataset contains 8 features (Gender-Age- -hypertension-Smokinghistory-HbA1c_level-bmi-heart disease-blood_glucose_level) and its goal is to classify the patients as either having diabetes or not based on their input features

Introduction

The dataset would likely to be split into a training set and a testing set, with the training set used to fit the machine learning models and the testing set used to evaluate the performance of the models. The different machine learning algorithms you mentioned, such as random forest, SVM, KNN, naive Bayes, decision tree, and logistic regression, would be used to classify the patients as either having diabetes or not based on their input features.

Preprocessing

The purpose of preprocessing is to transform the raw data into a format that is suitable for machine learning models, making it easier for them to learn and make accurate predictions.

-Label encoding

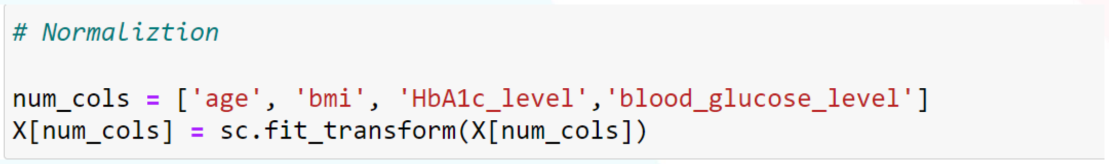
The purpose of label encoding in machine learning is to transform categorical variables into numerical values that can be used as input to machine learning algorithms, which can improve the performance of the resulting models.

```
# labelencoding
X['gender']=lb.fit_transform(X['gender'])
X
```

-Normalization

The purpose of normalization in machine learning is to avoid bias towards variables with larger scales, which can cause the algorithm to overlook smaller but still important features. Normalization can also help to improve the efficiency of machine learning algorithms and prevent numerical errors that may occur when large differences in feature scales are present.

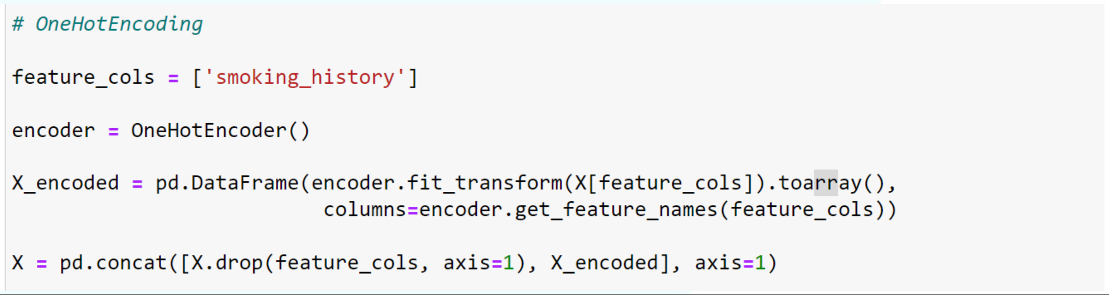
Here we're using the Standard Scalar technique for this purpose.



```
# Normaliztion

num_cols = ['age', 'bmi', 'HbA1c_level','blood_glucose_level']
X[num_cols] = sc.fit_transform(X[num_cols])
```

-One-hot encoding

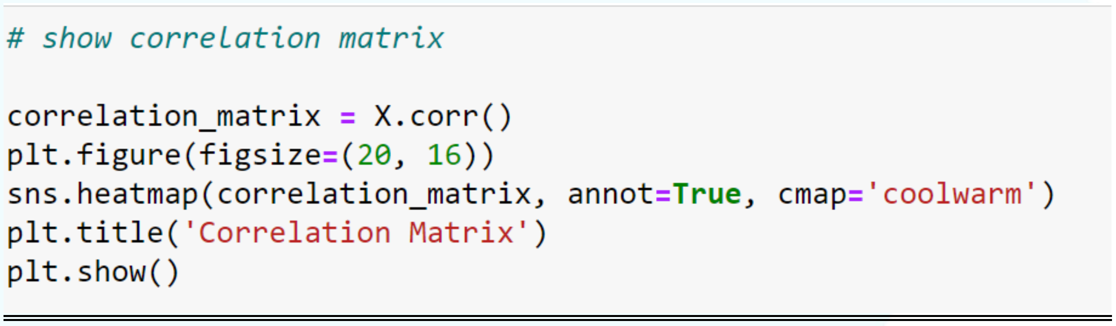
The purpose of one-hot encoding in this project is to convert categorical variables into a numerical format that can be used as input for machine learning models, categorical variables take on a limited number of discrete values. One-hot encoding creates a separate binary feature for each category, where each feature indicates whether or not the instance belongs to that category. This way, the machine learning algorithm can learn the relationship between the features and the target variable without any numerical bias towards a particular category.



```
# OneHotEncoding

feature_cols = ['smoking_history']

encoder = OneHotEncoder()

X_encoded = pd.DataFrame(encoder.fit_transform(X[feature_cols]).toarray(),
                          columns=encoder.get_feature_names(feature_cols))

X = pd.concat([X.drop(feature_cols, axis=1), X_encoded], axis=1)
```

-Correlation Matrix

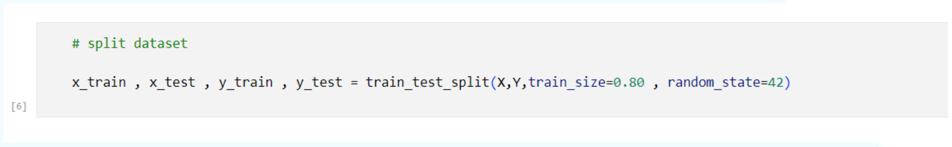
The correlation matrix is an important tool in machine learning for addressing overfitting because it helps to identify highly correlated features in a dataset. When two or more features are highly correlated, they contain redundant information and can lead to overfitting, you can identify which features are correlated and potentially remove one or more of them from the dataset. This can help to reduce the complexity of the model and improve its generalization performance on new, unseen data.



```
# show correlation matrix

correlation_matrix = X.corr()
plt.figure(figsize=(20, 16))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

Splitting-data

The purpose of this machine learning project is addressing dataset that contains the classification of diabetes disease which is to evaluate the performance of the model on unseen data. By separating the data into training and testing sets, we can use the training set to fit the machine learning model and the testing set to evaluate the accuracy of the model's predictions on unseen data.

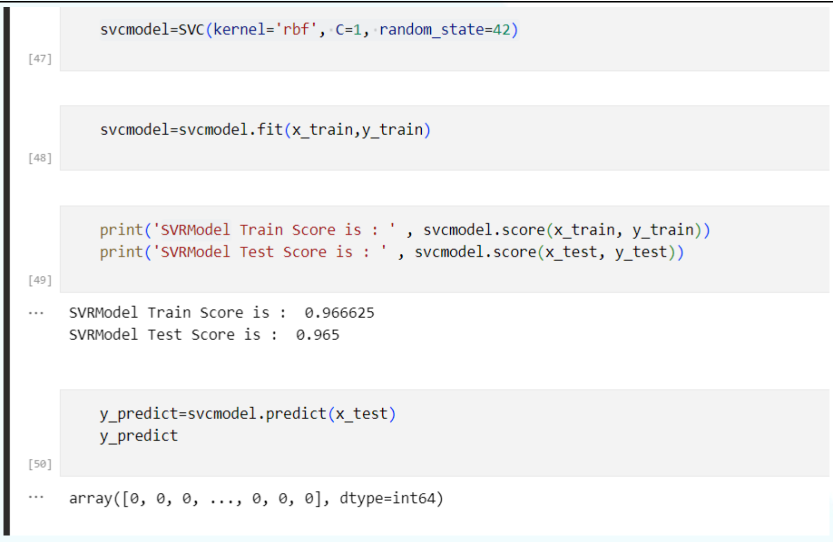


```
[6]
# split dataset

x_train , x_test , y_train , y_test = train_test_split(X,Y,train_size=0.80 , random_state=42)
```

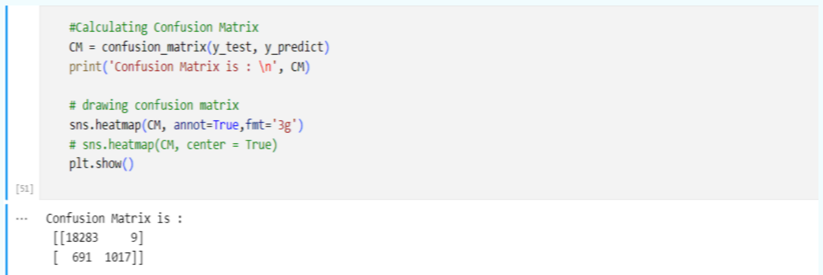
Support Vector Machine (SVM) algorithm

-The SVC algorithm works by finding a hyperplane that separates the data into two classes, in this case, patients with diabetes and patients without diabetes.

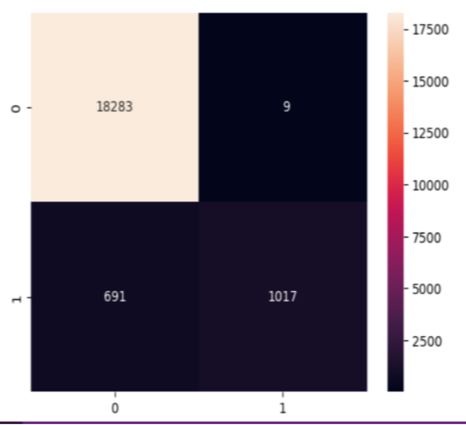
-The hyperplane that is chosen by the algorithm maximizes the margin between the two classes, making it a useful algorithm for binary classification problems. The SVC algorithm can also handle non-linearly separable data by using a kernel function to transform the data into a higher-dimensional space where it can be linearly separated.



```
[47]  svcmodel=SVC(kernel='rbf', C=1, random_state=42)

[48]  svcmodel=svcmodel.fit(x_train,y_train)

[49]  print('SVRModel Train Score is : ' , svcmodel.score(x_train, y_train))
      print('SVRModel Test Score is : ' , svcmodel.score(x_test, y_test))

...   SVRModel Train Score is :  0.966625
      SVRModel Test Score is :  0.965

[50]  y_predict=svcmodel.predict(x_test)
      y_predict

...   array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

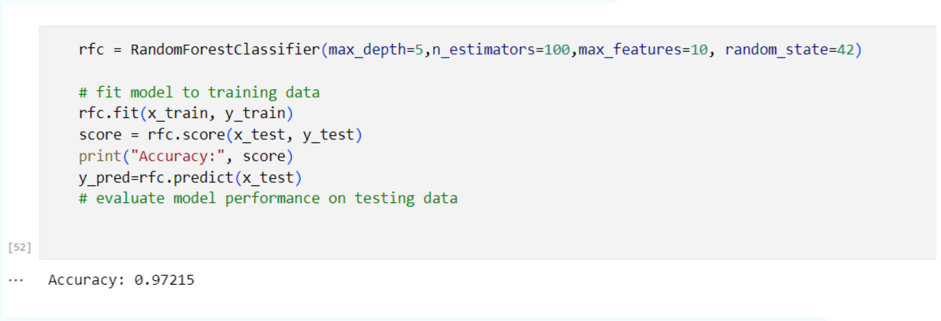


```
#Calculating Confusion Matrix
CM = confusion_matrix(y_test, y_predict)
print('Confusion Matrix is : \n', CM)

# drawing confusion matrix
sns.heatmap(CM, annot=True,fmt='3g')
# sns.heatmap(CM, center = True)
plt.show()
[51]

... Confusion Matrix is :
 [[18283     9]
 [  691  1017]]
```



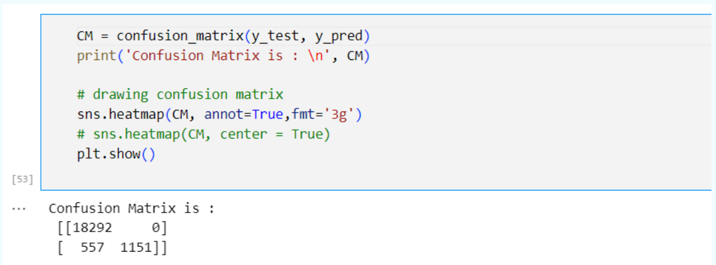
Random Forest

Random Forest algorithm works by creating multiple decision trees, each of which is trained on a random subset of the data and a random subset of the features. The output of each decision tree is then combined to make a final prediction. This ensemble approach helps to reduce overfitting and improve the generalization performance of the model.



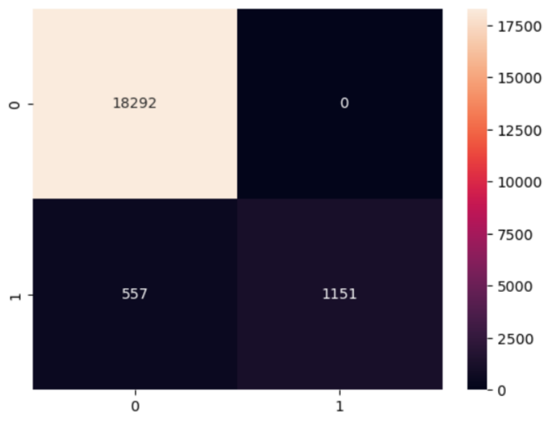
```
rfc = RandomForestClassifier(max_depth=5,n_estimators=100,max_features=10, random_state=42)

# fit model to training data
rfc.fit(x_train, y_train)
score = rfc.score(x_test, y_test)
print("Accuracy:", score)
y_pred=rfc.predict(x_test)
# evaluate model performance on testing data
```

```
Accuracy: 0.97215
```



```
CM = confusion_matrix(y_test, y_pred)
print('Confusion Matrix is : \n', CM)

# drawing confusion matrix
sns.heatmap(CM, annot=True,fmt='3g')
# sns.heatmap(CM, center = True)
plt.show()
```

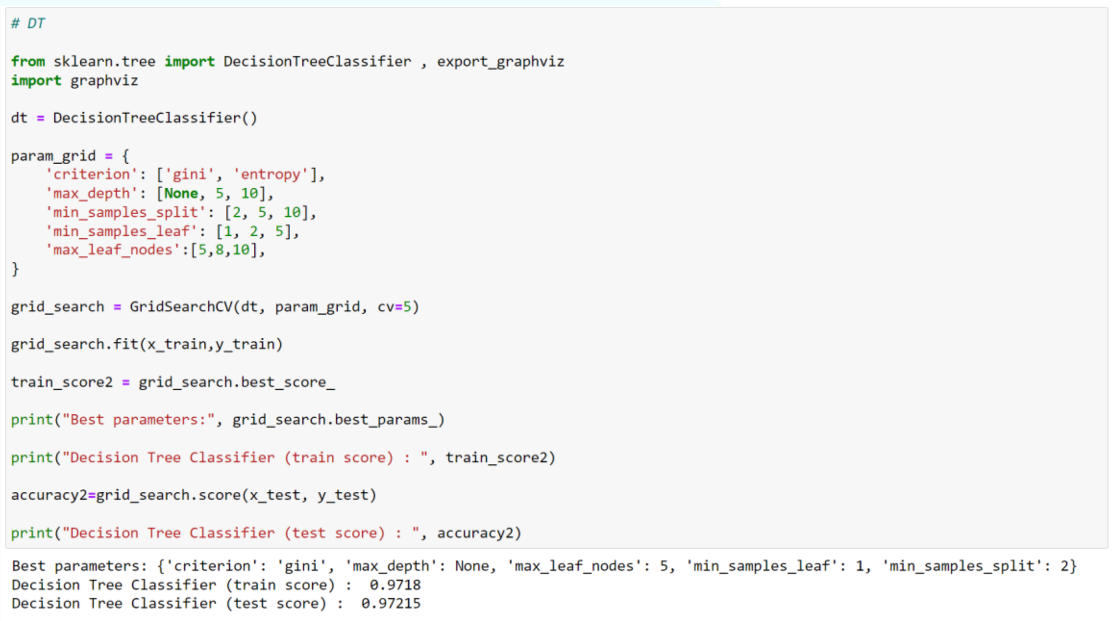
```
Confusion Matrix is :
 [[18292     0]
 [  557  1151]]
```



DecisionTreeClassifier algorithm

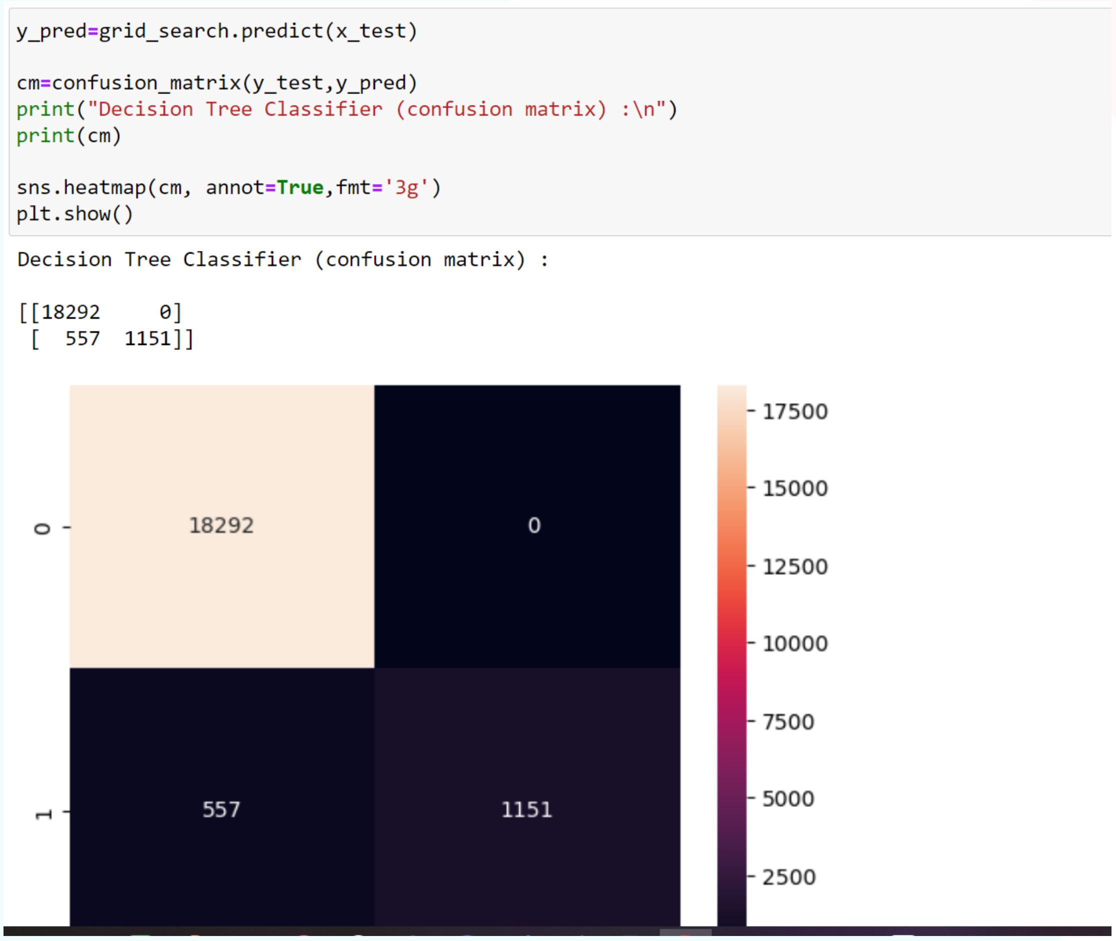
-The purpose of using the Decision Tree Classifier algorithm can be to build a model that can accurately predict whether a patient has diabetes or not based on various features such as glucose level, blood pressure, BMI, age, etc.

-The Decision Tree Classifier algorithm works by recursively partitioning the data based on the features that provide the most information gain at each step. This process results in a tree-like model, where each internal node represents a decision based on a feature, and each leaf node represents a class label.



```
# DT

from sklearn.tree import DecisionTreeClassifier , export_graphviz
import graphviz

dt = DecisionTreeClassifier()

param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5],
    'max_leaf_nodes':[5,8,10],
}

grid_search = GridSearchCV(dt, param_grid, cv=5)

grid_search.fit(x_train,y_train)

train_score2 = grid_search.best_score_

print("Best parameters:", grid_search.best_params_)

print("Decision Tree Classifier (train score) : ", train_score2)

accuracy2=grid_search.score(x_test, y_test)

print("Decision Tree Classifier (test score) : ", accuracy2)
```

```
Best parameters: {'criterion': 'gini', 'max_depth': None, 'max_leaf_nodes': 5, 'min_samples_leaf': 1, 'min_samples_split': 2}
Decision Tree Classifier (train score) :  0.9718
Decision Tree Classifier (test score) :  0.97215
```

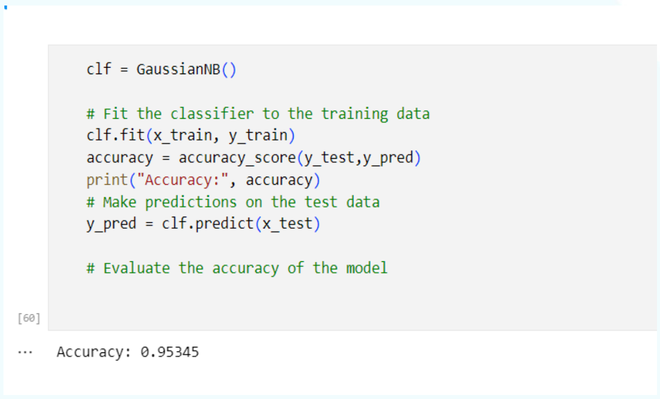

```
y_pred=grid_search.predict(x_test)

cm=confusion_matrix(y_test,y_pred)
print("Decision Tree Classifier (confusion matrix) :\n")
print(cm)

sns.heatmap(cm, annot=True,fmt='3g')
plt.show()
```

```
Decision Tree Classifier (confusion matrix) :

[[18292     0]
 [  557  1151]]
```

Naive Bayes Algorithm

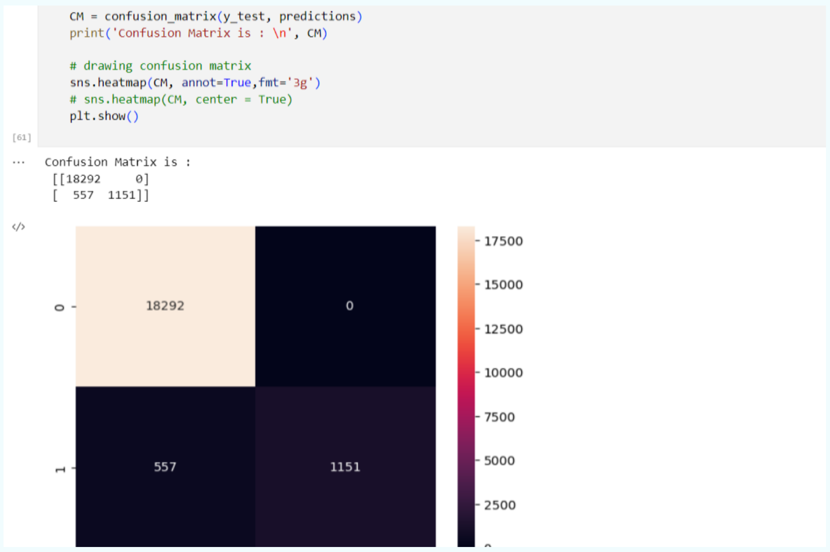
The Naive Bayes algorithm can be used to calculate the probability of a patient having diabetes based on certain features such as age, BMI, blood pressure, etc. It assumes that all features are independent of each other, hence the "naive" assumption.

- This algorithm is fast, requires a small amount of training data, and is relatively robust to irrelevant features in the dataset. It can also handle both categorical and continuous data. However, the Naive Bayes algorithm assumes that the features are independent, which may not always be true in real-world scenarios.



```
clf = GaussianNB()

# Fit the classifier to the training data
clf.fit(x_train, y_train)
accuracy = accuracy_score(y_test,y_pred)
print("Accuracy:", accuracy)
# Make predictions on the test data
y_pred = clf.predict(x_test)

# Evaluate the accuracy of the model
```

```
Accuracy: 0.95345
```

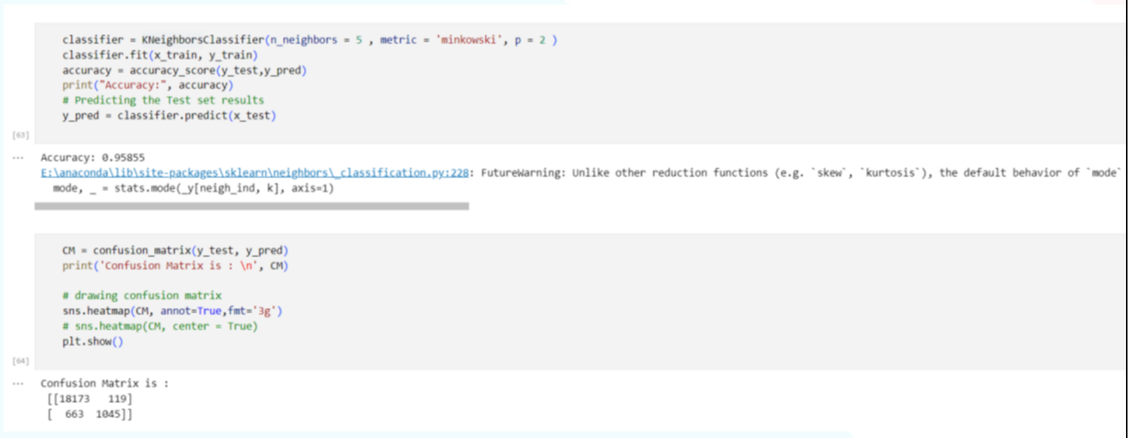


```
CM = confusion_matrix(y_test, predictions)
print('Confusion Matrix is : \n', CM)

# drawing confusion matrix
sns.heatmap(CM, annot=True,fmt='3g')
# sns.heatmap(CM, center = True)
plt.show()
```

```
Confusion Matrix is :
 [[18292     0]
 [  557  1151]]
```

K-Nearest Neighbors (KNN) Algorithm

KNN is a type of supervised learning algorithm that works by finding the k number of training data points that are closest to a new test data point and using the class labels of those k nearest points to classify the new data point. In the case of diabetes classification, the KNN algorithm can be used to predict whether a patient has diabetes or not based on features such as age, BMI, glucose level, and blood pressure. The KNN algorithm can be tuned by changing the value of k to achieve better classification accuracy.

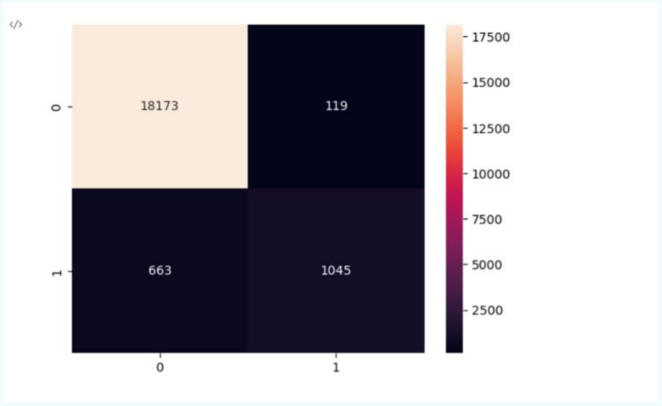


```
[63]
    classifier = KNeighborsClassifier(n_neighbors = 5 , metric = 'minkowski', p = 2 )
    classifier.fit(x_train, y_train)
    accuracy = accuracy_score(y_test,y_pred)
    print("Accuracy:", accuracy)
    # Predicting the Test set results
    y_pred = classifier.predict(x_test)
```

```
Accuracy: 0.95855
E:\anaconda\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

```
[64]
    CM = confusion_matrix(y_test, y_pred)
    print('Confusion Matrix is : \n', CM)

    # drawing confusion matrix
    sns.heatmap(CM, annot=True,fmt='3g')
    # sns.heatmap(CM, center = True)
    plt.show()
```

```
Confusion Matrix is :
 [[18173   119]
 [  663  1045]]
```



Logistic Regression Algorithm

Logistic Regression can be used to predict whether a patient has diabetes or not based on various input features such as glucose level, BMI, age, etc.

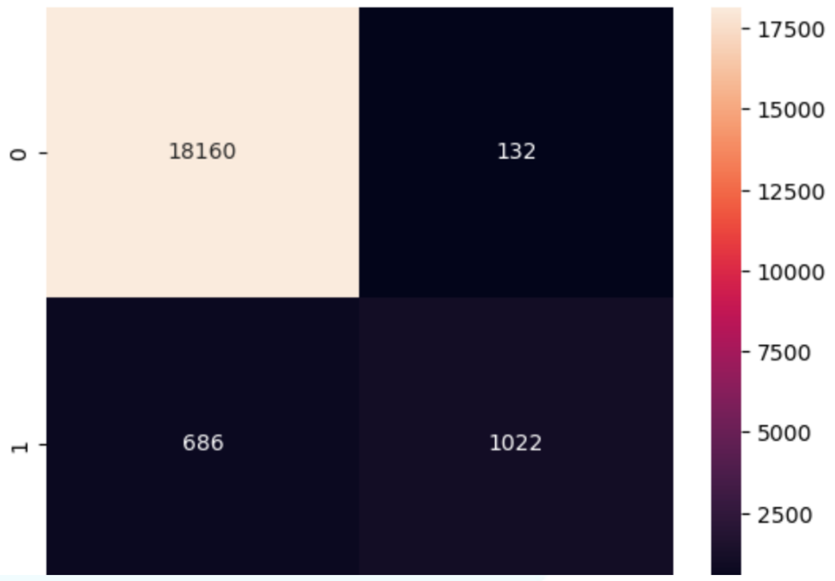
Logistic Regression works based on one or more predicator variables by using logistic function (sigmoid function). The logistic regression algorithm estimates the parameters by minimizing a cost function such as least squares method (LSM). Once the model is trained, it can be used to make predictions on new, unseen data by computing the probability of the event occurring for each instance and applying decision threshold to classify the instance into one of the *two classes.*

```python
# Logistic Regression

param_grid = {
    # 'penalty': ['l1', 'l2'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100]
}

from sklearn.linear_model import LogisticRegression

classifier=LogisticRegression(random_state=42)
grid_search = GridSearchCV(classifier, param_grid, cv=5)

grid_search.fit(x_train, y_train)

best_params = grid_search.best_params_
train_score1 = grid_search.best_score_

print("Best hyperparameters: ", best_params)
print("Logistic Regression Classifier (train score) : ", train_score1)

accuracy1=grid_search.score(x_test, y_test)

print(f"Logistic Regression Classifier (test score) : {accuracy1}")
```

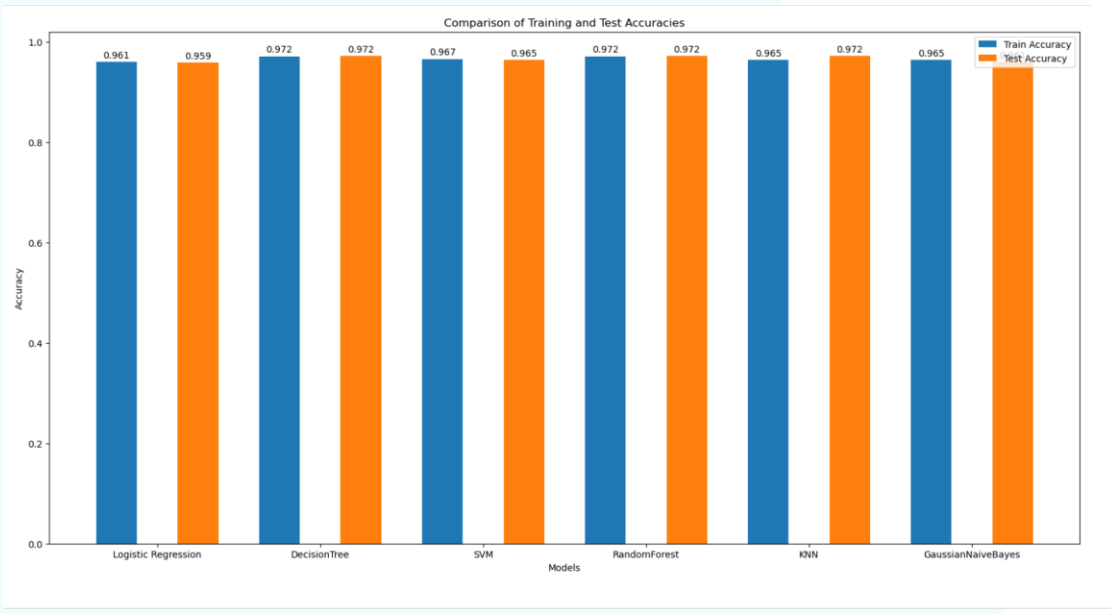
```
Best hyperparameters:  {'C': 0.01}
Logistic Regression Classifier (train score) :  0.9606875
Logistic Regression Classifier (test score) : 0.9591
```

```python
y_pred=grid_search.predict(x_test)
cm=confusion_matrix(y_test,y_pred)
print("Logistic Regression Classifier (confusion matrix) :\n")
print(cm)

sns.heatmap(cm, annot=True,fmt='3g')
plt.show()
```

```
Logistic Regression Classifier (confusion matrix) :

[[18160   132]
 [  686  1022]]
```



Here is a graph showing the performance of each machine learning model that is been used on the dataset.


Comparison of Training and Test Accuracies
Accuracy
Logistic Regression
DecisionTree
SVM
RandomForest
KNN
GaussianNaiveBayes
Models
Train Accuracy
Test Accuracy