Importing the Dependencies

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from imblearn.over_sampling import RandomOverSampler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import warnings
warnings.filterwarnings('ignore')
```

Loading the Dataset

```
# Load the diabetes dataset to a pandas DataFrame

diabetes_dataset = pd.read_csv('/content/drive/MyDrive/Data Set/diabetes.csv')
diabetes_dataset
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
0	6	148	72	35	0	33.6	0.
1	1	85	66	29	0	26.6	0.
2	8	183	64	0	0	23.3	0.
3	1	89	66	23	94	28.1	0.
4	0	137	40	35	168	43.1	2.
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	0.
764	2	122	70	27	0	36.8	0.
765	5	121	72	23	112	26.2	0.
766	1	126	60	0	0	30.1	0.
767	1	93	70	31	0	30.4	0.

768 rows × 9 columns

Cleaning the Dataset

```
# Finding Missing Data

diabetes_dataset.isna().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
# Finding Data Types

diabetes_dataset.dtypes
```

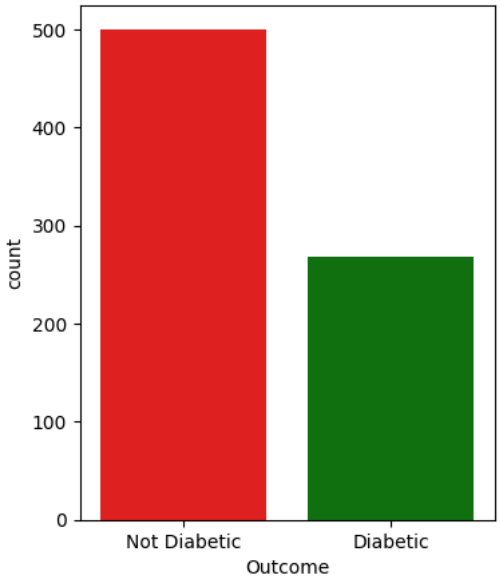
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
```

```
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

```
# Data Set Information

diabetes_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
# Graph shows peoples having diabetes &  not having diabetes

fig, ax = plt.subplots(figsize=(4, 5))
sns.countplot(x='Outcome', data=diabetes_dataset, ax=ax, palette=['red', 'green'])

ax.set_xticklabels(['Not Diabetic', 'Diabetic'])

plt.show()
```



Separating X & Y

```
# Separate the data and labels

X = diabetes_dataset.drop(columns='Outcome', axis=1)
y = diabetes_dataset['Outcome']
```

```
# Scaling using Standard Scaler

sc = StandardScaler()
X = sc.fit_transform(X)
```

```
# Using Random Over Sampling

ros = RandomOverSampler(random_state=42)
X,y = ros.fit_resample(X,y)
```

Training the Model

```
# Train-test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10)
```

Grid Search for all classifiers

```
# Finding the best parameters

# rf = RandomForestClassifier()
# params = {'n_estimators': [100, 200, 300],'criterion':['gini','entropy']}
# clf_rf = GridSearchCV(rf, params, cv=10, scoring='accuracy')
# clf_rf.fit(X_train, y_train)
# print("Best Parameters (Random Forest):", clf_rf.best_params_)
# Best Parameters (Random Forest): {criterion = 'entropy', n_estimators = 20}

# xgb = XGBClassifier()
# params = {'n_estimators': [100, 200, 300]}
# clf_xgb = GridSearchCV(xgb, params, cv=10, scoring='accuracy')
# clf_xgb.fit(X_train, y_train)
# print("Best Parameters (XGBoost):", clf_xgb.best_params_)
# Best Parameters (XGBoost): {'n_estimators': 200}

# knn = KNeighborsClassifier()
# params = {'n_neighbors': [3, 5, 7, 9],'weights': ['uniform', 'distance'],'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']}
# clf_knn = GridSearchCV(knn, params, cv=10, scoring='accuracy')
# clf_knn.fit(X_train, y_train)
# print("Best Parameters (KNeighbors):", clf_knn.best_params_)
# Best Parameters (KNeighbors): {'algorithm': 'auto', 'n_neighbors': 3, 'weights': 'uniform'}

# dtc = DecisionTreeClassifier()
# params = {'criterion': ['gini', 'entropy']}
# clf_dt = GridSearchCV(dtc, params, cv=10, scoring='accuracy')
# clf_dt.fit(X_train, y_train)
# print("Best Parameters (Decision-Tree):", clf_dt.best_params_)
# Best Parameters (Decision-Tree): {'criterion': 'entropy'}

# svc = SVC()
# params = {'C': [0.1, 1, 10],'kernel': ['linear', 'rbf'],'gamma': ['scale', 'auto']}
# clf_svc = GridSearchCV(svc, params, cv=10, scoring='accuracy')
# clf_svc.fit(X_train, y_train)
# print("Best Parameters (SVC):", clf_svc.best_params_)
# Best Parameters (SVC): {'C': 1, 'gamma': 'auto', 'kernel': 'rbf'}
```

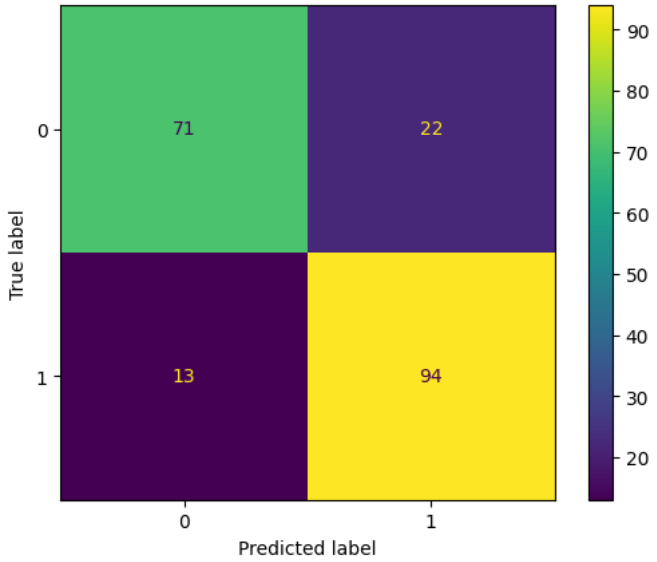
Classifiers

```
# Random Forest Classifier

rf = RandomForestClassifier(criterion = 'entropy', n_estimators = 20, random_state=2)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
print(classification_report(y_test,y_pred))
print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.91      0.84      0.87        93
               1       0.87      0.93      0.90       107

        accuracy                           0.89       200
       macro avg       0.89      0.88      0.88       200
    weighted avg       0.89      0.89      0.88       200

    <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f625ad8f250>
```

```
# XGBoost Classifier

xgb = XGBClassifier(n_estimators = 200)
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
print(classification_report(y_test,y_pred))
print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.85      0.76      0.80        93
               1       0.81      0.88      0.84       107

        accuracy                           0.82       200
       macro avg       0.83      0.82      0.82       200
    weighted avg       0.83      0.82      0.82       200

    <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f625ad8f250>
```

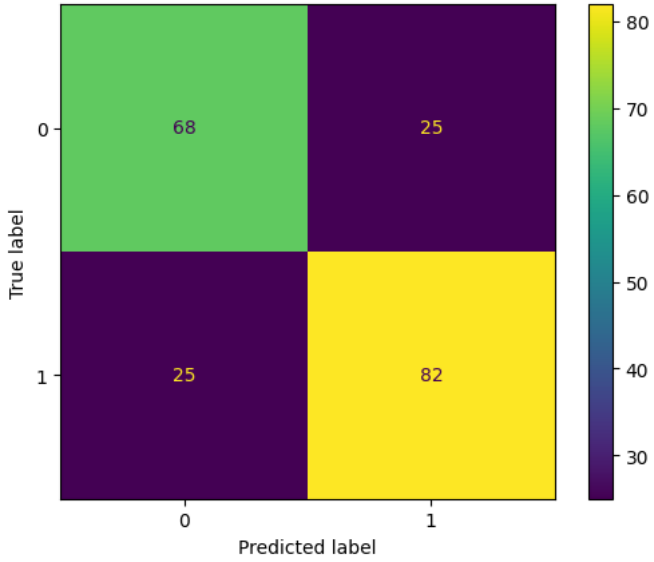



```
# KNeighbors Classifier

knn = KNeighborsClassifier(algorithm = 'auto', n_neighbors = 3, weights = 'uniform')
knn.fit(X_train,y_train)
y_pred = knn.predict(X_test)
print(classification_report(y_test,y_pred))
print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.77      0.74      0.75        93
               1       0.78      0.80      0.79       107

        accuracy                           0.78       200
       macro avg       0.77      0.77      0.77       200
    weighted avg       0.77      0.78      0.77       200

    <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f625b0f8a30>
```


```
# Decision-Tree Classifier

dtc = DecisionTreeClassifier(criterion = 'entropy', max_depth = 5, min_samples_leaf = 2, min_samples_split = 2)
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)
print(classification_report(y_test,y_pred))
print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.73      0.73      0.73        93
               1       0.77      0.77      0.77       107

        accuracy                           0.75       200
       macro avg       0.75      0.75      0.75       200
    weighted avg       0.75      0.75      0.75       200

    <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f625b434d30>
```

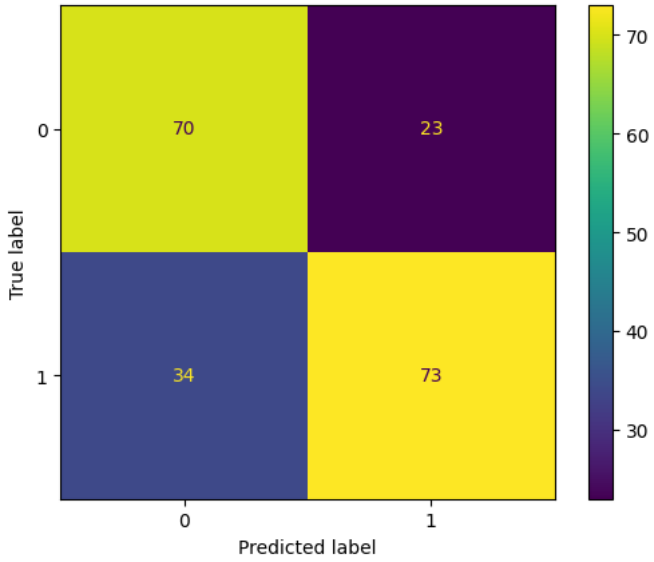



```
# SVC classifier

svc = SVC(C=1, gamma = 'auto', kernel = 'rbf')
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)
print(classification_report(y_test,y_pred))
print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.77      0.71      0.74        93
               1       0.76      0.81      0.79       107

        accuracy                           0.77       200
       macro avg       0.77      0.76      0.76       200
    weighted avg       0.77      0.77      0.76       200

    <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f625b312f50>
```


```
# GaussianNB classifier

gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)
print(classification_report(y_test,y_pred))
print(ConfusionMatrixDisplay.from_predictions(y_test,y_pred))
```

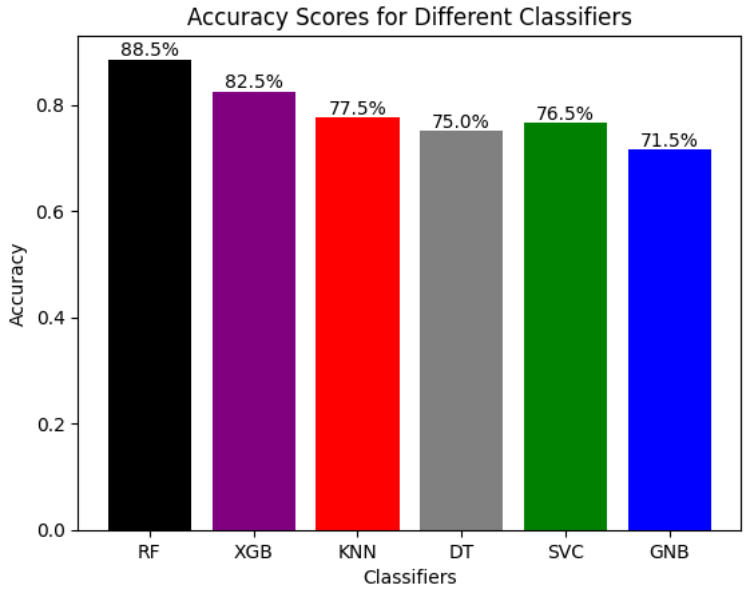
```
                  precision    recall  f1-score   support

               0       0.67      0.75      0.71        93
               1       0.76      0.68      0.72       107

        accuracy                           0.71       200
       macro avg       0.72      0.72      0.71       200
    weighted avg       0.72      0.71      0.72       200

    <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f625e256980>
```



Best Classifier

```
classifiers = [rf, xgb, knn, dtc, svc, gnb]
accuracy = []

for classifier in classifiers:
    y_pred = classifier.predict(X_test)
    accuracy_score = classification_report(y_test, y_pred, output_dict=True)['accuracy']
    accuracy.append(accuracy_score)

new_labels = ['RF', 'XGB', 'KNN', 'DT', 'SVC', 'GNB']
colors = ['black', 'purple', 'red', 'gray', 'green', 'blue']

plt.bar(new_labels, accuracy, color=colors)
plt.xlabel('Classifiers')
plt.ylabel('Accuracy')
plt.title('Accuracy Scores for Different Classifiers')
plt.xticks(rotation=0)  # Rotates the x-axis labels for better visibility

# Add labels inside each bar
for i in range(len(accuracy)):
    plt.text(i, accuracy[i], str(round(accuracy[i] * 100, 2)) + '%', ha='center', va='bottom')
```

```
plt.show()
```



Making a Predictive System

```
input_data = [2,146,70,38,360,28,0.337,29]

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# Scale the input data using the same StandardScaler instance used for training
input_data_scaled = sc.transform(input_data_reshaped)

# Making a Predictive System using Random Forest Classifier
prediction = rf.predict(input_data_scaled)
print(prediction)

if prediction[0] == 0:
    print('The person is not diabetic')
else:
    print('The person is diabetic')
```

```
 [1]
 The person is diabetic
```

Saving the Trained Model

```
import pickle


# Loading scaler file
scaler = pickle.load(open('scaler.pkl','rb'))


# Loading the saved model
loaded_model = pickle.load(open('trained_model.pkl', 'rb'))


input_data = [2,146,70,38,360,28,0.337,29]

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# Scale the input data using the same StandardScaler instance used for training
input_data_scaled = sc.transform(input_data_reshaped)

prediction = loaded_model.predict(input_data_scaled)
print(prediction)
```

```
if (prediction[0] == 0):
  print('The person is not diabetic')
else:
  print('The person is diabetic')
```

```
[1]
The person is diabetic
```

✓ 0s completed at 1:51 PM