ML LAB ASSIGNMENT 3 DIABETES KNN

Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset. Dataset link : https://www.kaggle.com/datasets/abdallamahgoub/diabete

```
# importing important packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Reading the data
data = pd.read_csv('/content/diabetes.csv')
```

Double-click (or enter) to edit

```
data.head()
```

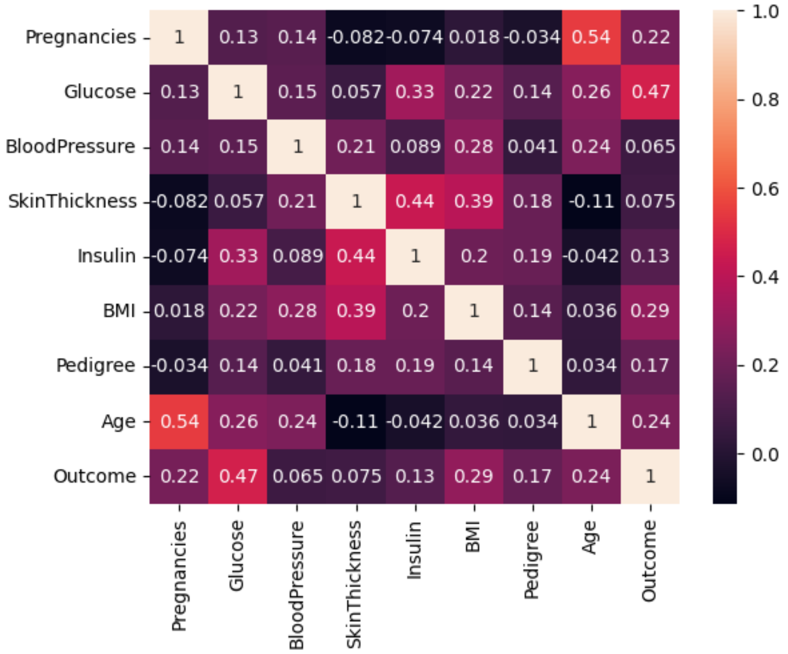
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree	Age	O
0	6	148	72	35	0	33.6	0.627	50	
1	1	85	66	29	0	26.6	0.351	31	
2	8	183	64	0	0	23.3	0.672	32	
3	1	89	66	23	94	28.1	0.167	21	
4	0	137	40	35	168	43.1	2.288	33	

```
# checking the Correlation between other columns and 'Outcome'
data.corr()['Outcome']
```

```
Pregnancies      0.221898
Glucose          0.466581
BloodPressure    0.065068
SkinThickness    0.074752
Insulin          0.130548
BMI              0.292695
Pedigree         0.173844
Age              0.238356
Outcome          1.000000
Name: Outcome, dtype: float64
```

```
# Heatmap shows relation between columns
sns.heatmap(data.corr(), annot=True)
```

```
<Axes: >
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree	Age	Outcome
Pregnancies	1	0.13	0.14	-0.082	-0.074	0.018	-0.034	0.54	0.22
Glucose	0.13	1	0.15	0.057	0.33	0.22	0.14	0.26	0.47
BloodPressure	0.14	0.15	1	0.21	0.089	0.28	0.041	0.24	0.065
SkinThickness	-0.082	0.057	0.21	1	0.44	0.39	0.18	-0.11	0.075
Insulin	-0.074	0.33	0.089	0.44	1	0.2	0.19	-0.042	0.13
BMI	0.018	0.22	0.28	0.39	0.2	1	0.14	0.036	0.29
Pedigree	-0.034	0.14	0.041	0.18	0.19	0.14	1	0.034	0.17
Age	0.54	0.26	0.24	-0.11	-0.042	0.036	0.034	1	0.24
Outcome	0.22	0.47	0.065	0.075	0.13	0.29	0.17	0.24	1

```
# Copying every column except 'Outcome'
X = data.drop('Outcome', axis=1)
```

```
# copying column "Outcome" to y df
y = data['Outcome']
```

```
# Importing train_test_split
from sklearn.model_selection import train_test_split
```

```
# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

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

```
from sklearn.neighbors import KNeighborsClassifier
```

```
# Creating object of StandardScalar()
scaler = StandardScaler()
```

```
# Creating object of KNN Classifier
knn = KNeighborsClassifier()
```

```
operations = [('scaler', scaler), ('knn', knn)]
```

```
from sklearn.pipeline import Pipeline
```

```
# Creating a pipeline instructions with given operations
# as initialised above
pipe = Pipeline(operations)
```

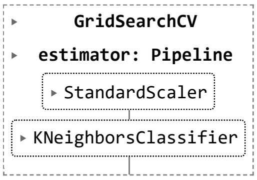
```
from sklearn.model_selection import GridSearchCV
```

```
k_values = list(range(1, 20))
```

```
# passing scoring metric
param_grid = {'knn__n_neighbors':k_values}
```

```
# passing hyperparameters, model and pipline to GridSearchCV
# GridSearchCV will then train the model on all possible combinations of hyperparameter values
# and select the model that produces the best score on the scoring metric.
full_classifier = GridSearchCV(pipe, param_grid, cv=5, scoring='accuracy')
```

```
# Training the model
full_classifier.fit(X_train, y_train)
```



▸ GridSearchCV

▸ estimator: Pipeline

▸ StandardScaler

▸ KNeighborsClassifier

```
# Gives the hyperparameter values of the best model found by the grid search
full_classifier.best_estimator_.get_params()
```

```
{'memory': None,
 'steps': [('scaler', StandardScaler()),
  ('knn', KNeighborsClassifier(n_neighbors=7))],
 'verbose': False,
 'scaler': StandardScaler(),
 'knn': KNeighborsClassifier(n_neighbors=7),
 'scaler__copy': True,
 'scaler__with_mean': True,
 'scaler__with_std': True,
 'knn__algorithm': 'auto',
 'knn__leaf_size': 30,
 'knn__metric': 'minkowski',
 'knn__metric_params': None,
 'knn__n_jobs': None,
 'knn__n_neighbors': 7,
 'knn__p': 2,
 'knn__weights': 'uniform'}
```

```
# predicting the values
y_pred = full_classifier.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

```
# Getting the accuracy score
accuracy_score(y_test, y_pred)
```

```
    0.696969696969697
```

```
# Error rate is the percentage of predictions that are incorrect
# and calculated as follows :
# Error_rate = 1 - accuracy

print(1 - accuracy_score(y_test, y_pred))
```

```
    0.303030303030303
```

```
# Getting confusion matrix
confusion_matrix(y_test, y_pred)
```

```
    array([[121,  30],
           [ 40,  40]])
```

```
# Getting classification report
print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.75      0.80      0.78       151
               1       0.57      0.50      0.53        80

        accuracy                           0.70       231
       macro avg       0.66      0.65      0.65       231
    weighted avg       0.69      0.70      0.69       231
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

```
# Calculate the precision score
precision = precision_score(y_test, y_pred)

# Calculate the recall score
recall = recall_score(y_test, y_pred)

# Print the results
print("Precision:", precision)
print("Recall:", recall)
```

```
    Precision: 0.5714285714285714
    Recall: 0.5
```