In [ ]:
```
Expt No 7
```

URK21CS1181 ISHAN CHASKAR

In [ ]:
```
Aim: To execute the Performance Analysis on KNN Classification Technique using
data science
```

In [ ]:
```
Description:
K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on
Supervised Learning technique. K-NN algorithm assumes the similarity between the
and available cases and put the new case into the category that is most similar
categories. K-NN algorithm stores all the available data and classifies a new da
the similarity. This means when new data appears then it can be easily classifie
category by using K- NN algorithm.

knn = KNeighborsClassifier(n_neighbor=5)
knn.fit(x_train,y_train)

StandardScaler comes into play when the characteristics of the input dataset dif
their ranges, or simply when they are measured in different units of measure. St
removes the mean and scales the data to the unit variance. However, outliers hav
calculating the empirical mean and standard deviation, which narrows the range c
values.

standard_Scaler = Standard Scaler()
x_train = standard_Scaler.fit_transform(x_train)
x_test = standard_Scaler.transform(x_test)

K-NN algorithm can be used for Regression as well as for Classification but most
Classification problems. K-NN is a non-parametric algorithm, which means it does
assumption on underlying data. It is also called a lazy learner algorithm becaus
from the training set immediately instead it stores the dataset and at the time
performs an action on the dataset.

The kNN algorithm is a supervised machine learning model. That means it predicts
variable using one or multiple independent variables.

y_pred = knn.predict(x_test)
print(y_pred)
print(y_test)
```

In [22]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
```

In [23]:
```
print("URK21CS1181")
df = pd.read_csv("diabetes.csv",index_col=0)
df
```

URK21CS1181

Out[23]:

Name	City	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
Ann	Kovai	6	148	72	35	0	33.6	
Sam	Trichy	1	85	66	29	0	26.6	
George	Madurai	8	183	64	0	0	23.3	
Sam	Banglore	1	89	66	23	94	28.1	
Matt	Banglore	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	...	
Matt	Banglore	10	101	76	48	180	32.9	
Matt	Banglore	2	122	70	27	0	36.8	
Jennifer	Madurai	5	121	72	23	112	26.2	
Matt	Banglore	1	126	60	0	0	30.1	
Ann	Trichy	1	93	70	31	0	30.4	

768 rows × 10 columns

In [24]:
```
print("URK21CS1181")
df1 = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']
df[df1]=df[df1].replace(0,np.nan)
df.head(10)
```

URK21CS1181

Out[24]:

Name	City	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
Ann	Kovai	6	148.0	72.0	35.0	NaN	33.6	
Sam	Trichy	1	85.0	66.0	29.0	NaN	26.6	
George	Madurai	8	183.0	64.0	NaN	NaN	23.3	
Sam	Banglore	1	89.0	66.0	23.0	94.0	28.1	
Matt	Banglore	0	137.0	40.0	35.0	168.0	43.1	
Matt	Banglore	5	116.0	74.0	NaN	NaN	25.6	
Matt	Chennai	3	78.0	50.0	32.0	88.0	31.0	
George	Kovai	10	115.0	NaN	NaN	NaN	35.3	
Jennifer	Banglore	2	197.0	70.0	45.0	543.0	30.5	
Jennifer	Trichy	8	125.0	96.0	NaN	NaN	NaN	

In [25]:
```
print("URK21CS1181")
df.isnull().sum()
```

URK21CS1181

Out[25]:
```
City                         0
Pregnancies                  0
Glucose                      5
BloodPressure               35
SkinThickness              227
Insulin                    374
BMI                         11
DiabetesPedigreeFunction     0
Age                          0
Outcome                      0
dtype: int64
```

In [26]:
```
print("URK21CS1181")
df.fillna(df.mean(),inplace=True)
df.head()
```

URK21CS1181

```
/tmp/ipykernel_4013620/901963660.py:2: FutureWarning: The default value of nume
ric_only in DataFrame.mean is deprecated. In a future version, it will default
to False. In addition, specifying 'numeric_only=None' is deprecated. Select onl
y valid columns or specify the value of numeric_only to silence this warning.
  df.fillna(df.mean(),inplace=True)
```

Out[26]:

Name	City	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabe
Ann	Kovai	6	148.0	72.0	35.00000	155.548223	33.6	
Sam	Trichy	1	85.0	66.0	29.00000	155.548223	26.6	
George	Madurai	8	183.0	64.0	29.15342	155.548223	23.3	
Sam	Banglore	1	89.0	66.0	23.00000	94.000000	28.1	
Matt	Banglore	0	137.0	40.0	35.00000	168.000000	43.1	

In [27]:
```
print("URK21CS1181")
x = df[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin','BMI', 'DiabetesPedigreeFunction', 'Age']]
x
```

URK21CS1181

Out[27]:

Name	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigr
Ann	6	148.0	72.0	35.00000	155.548223	33.6	
Sam	1	85.0	66.0	29.00000	155.548223	26.6	
George	8	183.0	64.0	29.15342	155.548223	23.3	
Sam	1	89.0	66.0	23.00000	94.000000	28.1	
Matt	0	137.0	40.0	35.00000	168.000000	43.1	
...	...	...	...	...	...	...	
Matt	10	101.0	76.0	48.00000	180.000000	32.9	
Matt	2	122.0	70.0	27.00000	155.548223	36.8	
Jennifer	5	121.0	72.0	23.00000	112.000000	26.2	
Matt	1	126.0	60.0	29.15342	155.548223	30.1	
Ann	1	93.0	70.0	31.00000	155.548223	30.4	

768 rows × 8 columns

In [28]:
```
print("URK21CS1181")
y = df[['Outcome']]
y
```

URK21CS1181

Out[28]:

Name	Outcome
Ann	Yes
Sam	No
George	Yes
Sam	No
Matt	Yes
...	...
Matt	No
Matt	No
Jennifer	No
Matt	Yes
Ann	No

768 rows × 1 columns

In [29]:
```
print("URK21CS1181")
y = y.replace({'No':0,'Yes':1})
y
```

URK21CS1181

Out[29]:

Name	Outcome
Ann	1
Sam	0
George	1
Sam	0
Matt	1
...	...
Matt	0
Matt	0
Jennifer	0
Matt	1
Ann	0

768 rows × 1 columns

In [30]:
```
print("URK21CS1181")
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state
print(x_train)
print(x_test)
print(y_train)
print(y_test)
```

```
URK21CS1181
          Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin  \
Name
George             13    129.0      72.405184       30.00000  155.548223
Ann                 4    129.0      86.000000       20.00000  270.000000
Jennifer            3     61.0      82.000000       28.00000  155.548223
Matt                2     81.0      72.000000       15.00000   76.000000
Sam                 0    102.0      75.000000       23.00000  155.548223
...               ...      ...            ...            ...         ...
Ann                 5    139.0      64.000000       35.00000  140.000000
Matt                1     96.0     122.000000       29.15342  155.548223
Jennifer           10    101.0      86.000000       37.00000  155.548223
Jennifer            0    141.0      72.405184       29.15342  155.548223
Jennifer            0    125.0      96.000000       29.15342  155.548223

                BMI  DiabetesPedigreeFunction  Age
Name
George    39.900000                     0.569   44
Ann       35.100000                     0.231   23
Jennifer  34.400000                     0.243   46
Matt      30.100000                     0.547   25
Sam       32.457464                     0.572   21
...             ...                       ...  ...
Ann       28.600000                     0.411   26
Matt      22.400000                     0.207   27
Jennifer  45.600000                     1.136   38
Jennifer  42.400000                     0.205   29
Jennifer  22.500000                     0.262   21

[576 rows x 8 columns]
          Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin  \
Name
Jennifer            6     98.0           58.0       33.00000  190.000000
Ann                 2    112.0           75.0       32.00000  155.548223
George              2    108.0           64.0       29.15342  155.548223
Sam                 8    107.0           80.0       29.15342  155.548223
Jennifer            7    136.0           90.0       29.15342  155.548223
...               ...      ...            ...            ...         ...
Matt                1     84.0           64.0       23.00000  115.000000
Sam                 6    194.0           78.0       29.15342  155.548223
Matt                6    123.0           72.0       45.00000  230.000000
Matt                3     78.0           50.0       32.00000   88.000000
Sam                 3    106.0           72.0       29.15342  155.548223

           BMI  DiabetesPedigreeFunction  Age
Name
Jennifer  34.0                     0.430   43
Ann       35.7                     0.148   21
George    30.8                     0.158   21
Sam       24.6                     0.856   34
Jennifer  29.9                     0.210   50
...        ...                       ...  ...
Matt      36.9                     0.471   28
Sam       23.5                     0.129   59
Matt      33.6                     0.733   34
Matt      31.0                     0.248   26
Sam       25.8                     0.207   27

[192 rows x 8 columns]
          Outcome
```

```
Name
George          1
Ann             0
Jennifer        0
Matt            0
Sam             0
...           ...
Ann             0
Matt            0
Jennifer        1
Jennifer        1
Jennifer        0

[576 rows x 1 columns]
          Outcome
Name
Jennifer        0
Ann             0
George          0
Sam             0
Jennifer        0
...           ...
Matt            0
Sam             1
Matt            0
Matt            1
Sam             0

[192 rows x 1 columns]
```

In [31]:
```
print("URK21CS1181")
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train,y_train)
```

```
URK21CS1181
/home/urk21cs1181/.local/lib/python3.9/site-packages/sklearn/neighbors/_classif
ication.py:228: DataConversionWarning: A column-vector y was passed when a 1d a
rray was expected. Please change the shape of y to (n_samples,), for example us
ing ravel().
  return self._fit(X, y)
```

Out[31]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [32]:
```
print("URK21CS1181")
y_pred = knn.predict(x_test)
print("Confusion Matrix:",confusion_matrix(y_test,y_pred))
print("Accuracy:",accuracy_score(y_test,y_pred))
print("Classification:",classification_report(y_test,y_pred))
print("AUC Score:",roc_auc_score(y_test,y_pred))
```

```
URK21CS1181
Confusion Matrix: [[87 36]
 [29 40]]
Accuracy: 0.6614583333333334
Classification:               precision    recall  f1-score   support

           0       0.75      0.71      0.73       123
           1       0.53      0.58      0.55        69

    accuracy                           0.66       192
   macro avg       0.64      0.64      0.64       192
weighted avg       0.67      0.66      0.66       192

AUC Score: 0.6435136090491339
```

In [33]:
```
print("URK21CS1181")
for i in [5,7,9,11]:
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,y_train)
    y_pred = knn.predict(x_test)
    print("Confusion Matrix:",confusion_matrix(y_test,y_pred))
    print("Accuracy:",accuracy_score(y_test,y_pred))
    print("Classification:",classification_report(y_test,y_pred))
    print("AUC Score:",roc_auc_score(y_test,y_pred))
```

```
URK21CS1181
Confusion Matrix: [[83 40]
 [28 41]]
Accuracy: 0.6458333333333334
Classification:               precision    recall  f1-score   support

           0       0.75      0.67      0.71       123
           1       0.51      0.59      0.55        69

    accuracy                           0.65       192
   macro avg       0.63      0.63      0.63       192
weighted avg       0.66      0.65      0.65       192

AUC Score: 0.6344998232591021
Confusion Matrix: [[85 38]
 [28 41]]
Accuracy: 0.65625
Classification:               precision    recall  f1-score   support

           0       0.75      0.69      0.72       123
           1       0.52      0.59      0.55        69

    accuracy                           0.66       192
   macro avg       0.64      0.64      0.64       192
weighted avg       0.67      0.66      0.66       192

AUC Score: 0.6426299045599151
Confusion Matrix: [[90 33]
 [27 42]]
Accuracy: 0.6875
Classification:               precision    recall  f1-score   support

           0       0.77      0.73      0.75       123
           1       0.56      0.61      0.58        69

    accuracy                           0.69       192
   macro avg       0.66      0.67      0.67       192
weighted avg       0.69      0.69      0.69       192

AUC Score: 0.6702014846235419
Confusion Matrix: [[90 33]
 [28 41]]
Accuracy: 0.6822916666666666
Classification:               precision    recall  f1-score   support

           0       0.76      0.73      0.75       123
           1       0.55      0.59      0.57        69

    accuracy                           0.68       192
   macro avg       0.66      0.66      0.66       192
weighted avg       0.69      0.68      0.68       192

AUC Score: 0.6629551078119477
```

```
/home/urk21cs1181/.local/lib/python3.9/site-packages/sklearn/neighbors/_classif
ication.py:228: DataConversionWarning: A column-vector y was passed when a 1d a
rray was expected. Please change the shape of y to (n_samples,), for example us
ing ravel().
  return self._fit(X, y)
/home/urk21cs1181/.local/lib/python3.9/site-packages/sklearn/neighbors/_classif
ication.py:228: DataConversionWarning: A column-vector y was passed when a 1d a
rray was expected. Please change the shape of y to (n_samples,), for example us
ing ravel().
  return self._fit(X, y)
/home/urk21cs1181/.local/lib/python3.9/site-packages/sklearn/neighbors/_classif
ication.py:228: DataConversionWarning: A column-vector y was passed when a 1d a
rray was expected. Please change the shape of y to (n_samples,), for example us
ing ravel().
  return self._fit(X, y)
/home/urk21cs1181/.local/lib/python3.9/site-packages/sklearn/neighbors/_classif
ication.py:228: DataConversionWarning: A column-vector y was passed when a 1d a
rray was expected. Please change the shape of y to (n_samples,), for example us
ing ravel().
  return self._fit(X, y)
```

In [ ]:
```
j. Analyse and infer for which K value, the classification algorithm provides be
print("URK21CS1181")
```

In [ ]:
```
We can analyze the performance metrics for different k values and infer which
k value providesbetter performance Based on the k values, we can see that:
k=5 provides the highest accuracy and F1 score, but the lowest AUC score.
k = 7 provides the second-hoighest accuracy and F1 score,and the second
-highest AUC score. k = 9 provides the thrid-highest accuracy and F1 score
and the third-highest, AUC score k = 11 provides the lowest accuracy
and F1 score, but the second-highest AUC score. Therefore, we can infer
that k=5 provides better performance for this particular dataset
and classification problem.
```

In [ ]:
```
Result:The programs are executed by using Performance Analysis
on KNN Classification Technique.
```