INDEX

PROGRAM NO: 1 **DATE :19-08-22**

AIM: Predict class label of a given data point using KNN

SOURCE CODE

```
from sklearn.neighbors import KNeighborsClassifier
x1=[7,7,3,1]
x2=[7,4,4,4]
target=['bad', 'bad', 'Good','Good']
from sklearn import preprocessing
le= preprocessing.LabelEncoder()
target_encoded=le.fit_transform(target)
print(target_encoded)
features=zip(x1,x2)
features =list(features)
features
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(features,target)
print(knn.predict([[3,7]]))
```

OUTPUT:

```
[1 1 0 0]
['Good']
```

PROGRAM NO:2 **DATE :24-08-22**

AIM: Predict the class label of an unseen observation using Naïve_bayes

SOURCE CODE

```
weather= ['Sunny', 'Sunny','Overcast', 'Rainy', 'Rainy', 'Rainy','Overcast', 'Sunny', 'Sunny',
'Rainy', 'Sunny', 'Overcast','Overcast', 'Rainy']
temp=['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild','Cool', 'Mild', 'Mild', 'Mild', 'Hot',
'Mild']
play=['No','No', 'Yes', 'Yes','Yes', 'No', 'Yes','No', 'Yes','Yes','Yes', 'Yes', 'Yes', 'No']
from sklearn import preprocessing
#creating labelEncoder
le =preprocessing.LabelEncoder()
# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
print(weather_encoded)
temp_encoded=le.fit_transform(temp)
label=le.fit_transform(play)
print("Temp:",temp_encoded)
print("Play:",label)
features=zip(weather_encoded,temp_encoded)
features=list(features)
features
from sklearn.naive_bayes import GaussianNB
#Create a Gaussian Classifier
model = GaussianNB()
#Train the model using the training sets
model.fit(features, label)
#Predict Output
predicted= model.predict([[0,2]]) #0:Overcast, 2:Mild
print("Predicted Value:", predicted)
```

OUTPUT:

```
[2 2 0 1 1 1 0 2 2 1 2 0 0 1]
Temp: [1 1 1 2 0 0 0 2 0 2 2 2 1 2]
Play: [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Predicted Value: [1]
```

PROGRAM NO:3 **DATE :25-08-22**

AIM: Calculate accuracy of KNN using iris dataset

SOURCE CODE

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
irisData= load_iris()
print("Features: ", irisData.feature_names)
print("Labels: ", irisData.target_names)
X= irisData.data
y= irisData.target
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
knn= KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
print(knn.predict([[7.7,2.6,6.9,2.3]]))
y_pred=knn.predict(X_test)
print(y_pred)
from sklearn.metrics import accuracy_score
ac= accuracy_score(y_test,y_pred)
print(ac)
```

OUTPUT:

```
Features:  ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Labels:  ['setosa' 'versicolor' 'virginica']
[2]
[1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
0.9666666666666667
```

PROGRAM NO:4 **DATE :13-09-22**

AIM: Predict the accuracy of standard data set using Naïve_bayes

SOURCE CODE

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
irisData=load_iris()
print("Features: ", irisData. feature_names)
print("Labels: ", irisData. target_names)
X= irisData.data
y=irisData. target
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size = 0.2, random_state=42)
from sklearn.naive_bayes import GaussianNB
gnb=GaussianNB()
gnb.fit(X_train, y_train)
print(gnb.predict([[ 7.7,2.6,6.9,2.3]]))
y_predl= gnb.predict(X_test)
print(y_predl)
from sklearn.metrics import accuracy_score
acl= accuracy_score(y_test,y_predl)
print(acl)
```

OUTPUT:

```
Features:  ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Labels:  ['setosa' 'versicolor' 'virginica']
[2]
[1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
1.0
```

PROGRAM NO: 5 **DATE :29-09-22**

AIM: Data Visualization

SOURCE CODE

```
1)import matplotlib.pyplot as plt
x = [1,4,3]
y =[2,4,1]
plt.plot(x, y, linewidth=4)
plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.title('My first graph!')
plt.show()
```

```
2) import matplotlib.pyplot as plt
x=[1,2,3]
y= [2,4,1]
plt.bar(x, y)
plt.xlabel('x -axis')
plt.ylabel('y - axis')
plt.title('My first graph!')
plt.show()
```

```
3) from matplotlib import pyplot as plt
y=[10, 5, 8, 4, 2,2,2,5]
plt.hist(y)
plt.show()
```

```
4) from matplotlib import pyplot as plt
x = [5, 2, 9,4, 71]
y=[10, 5, 8, 4, 2]
plt.scatter(x, y)
plt.show()
```

```
5) import plotly.express as px
fig=px.line(x=[1, 2, 3], y=[2, 4, 6])
fig.show()
```

```
6) import plotly.express as px
df=px.data.iris()
print(df)
fig= px.line(df, x="species", y="petal_width")
fig.show()
```

```
7) import plotly.express as px
df=px.data.iris()
fig=px.bar(df, x="sepal_width", y="sepal_length")
fig.show()
```

```
8) import plotly.express as px
df= px.data.iris()
fig=px.histogram(df, x="sepal_length", y="petal_width")
fig.show()
```

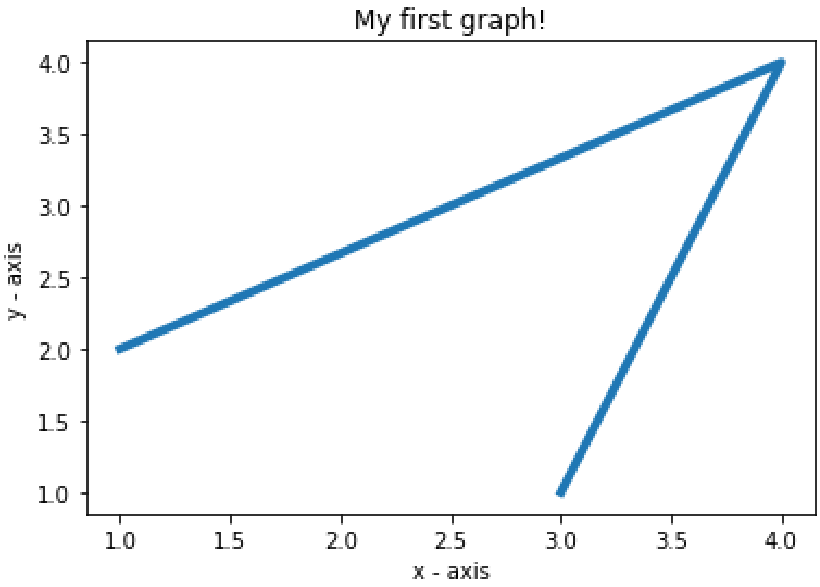
```
9) import plotly.express as px
df= px.data.iris()
fig=px.scatter(df, x="species", y="petal_width")
fig.show()
```

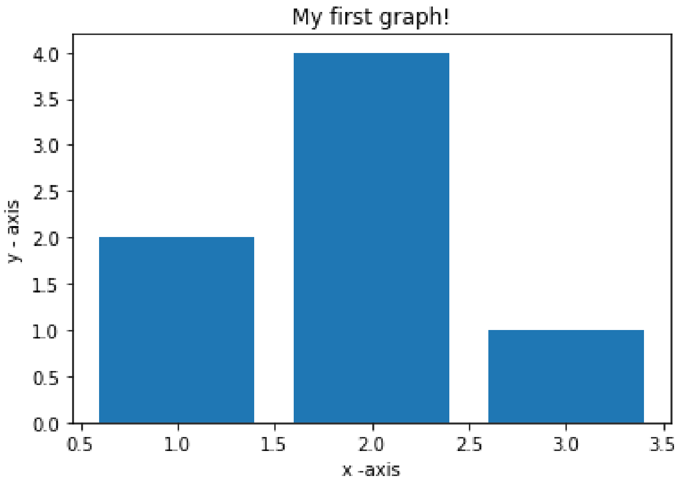
```
10) import plotly.express as px
df=px.data.iris()
fig= px.scatter(df, x="species", y="petal_width",
                size="petal_length",color="species")
fig.show()
```

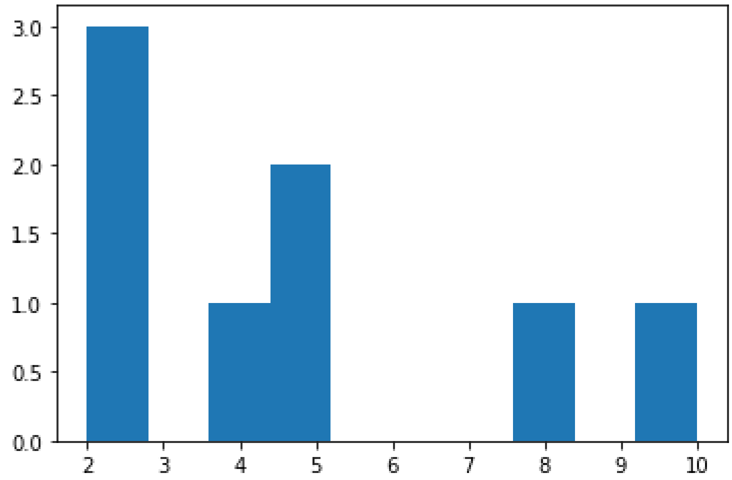
```
11) import plotly.express as px
df= px.data.iris()
```

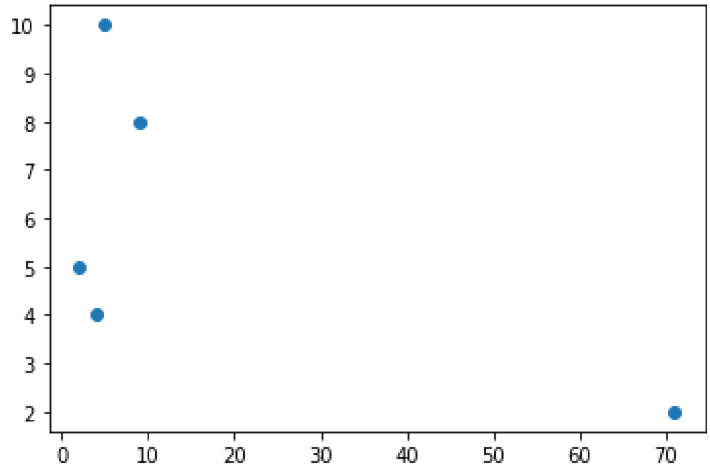
```
fig = px.scatter_3d(df, x = 'sepal_width',
                    y= 'sepal_length',
                    z='petal_width',
                    color ='species')
fig.show()
```

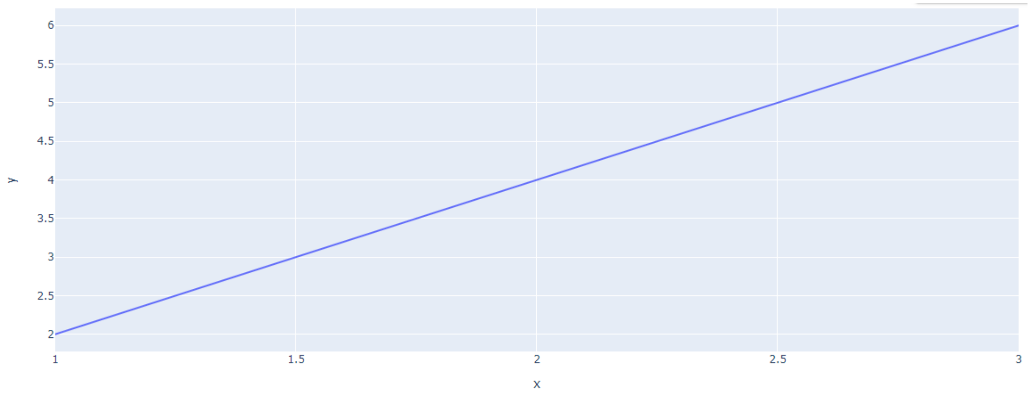
OUTPUT:





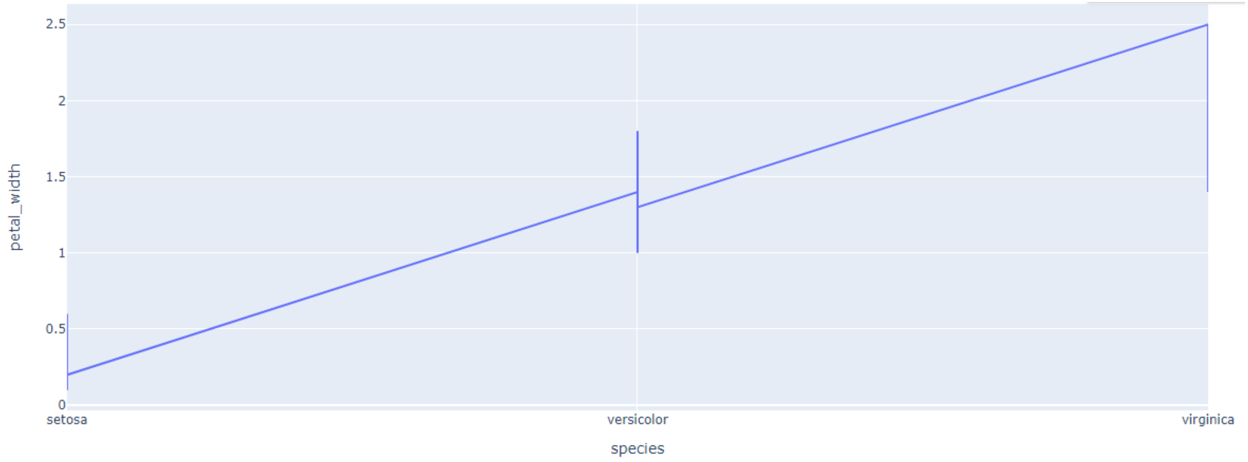


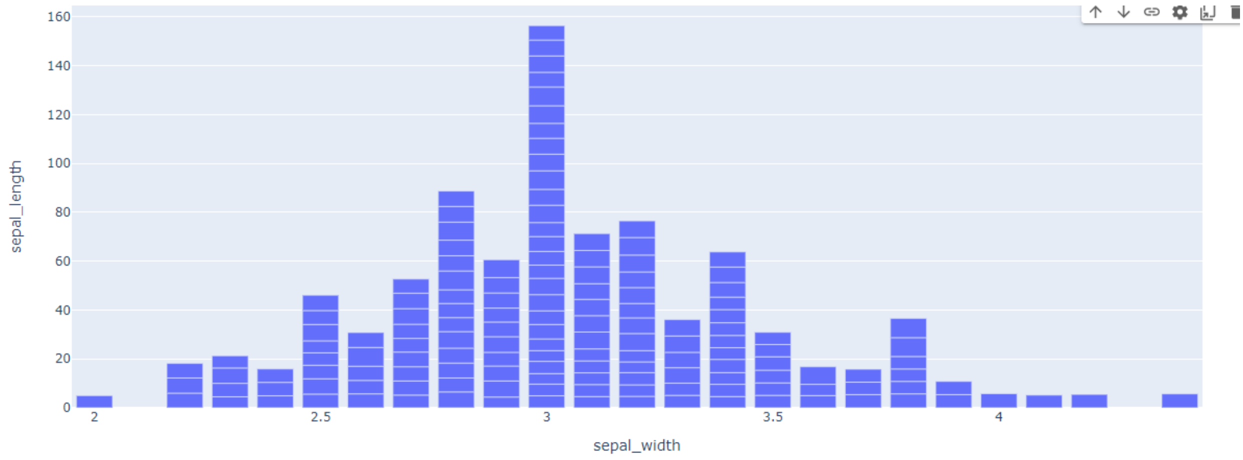


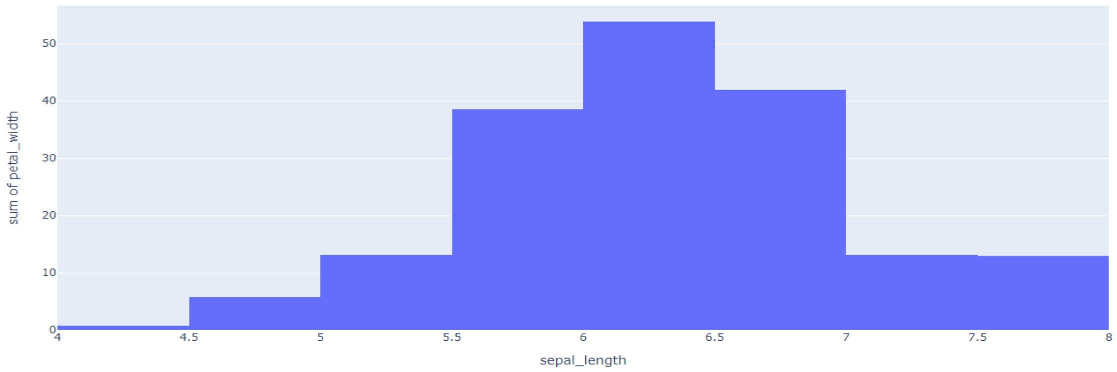


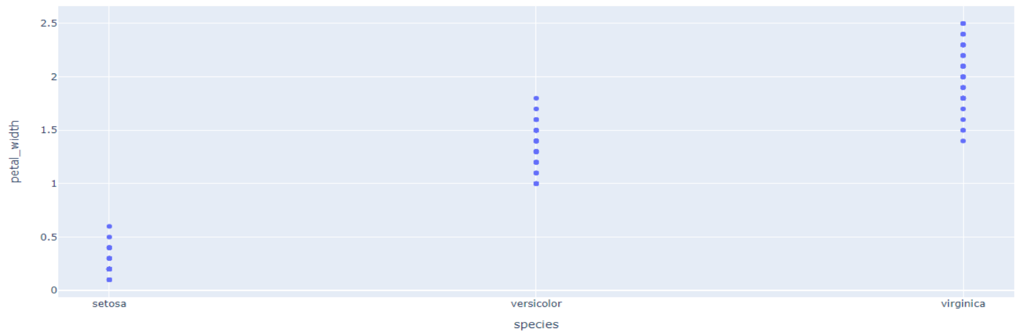
```
     sepal_length  sepal_width  petal_length  petal_width    species  \
0             5.1          3.5           1.4          0.2     setosa
1             4.9          3.0           1.4          0.2     setosa
2             4.7          3.2           1.3          0.2     setosa
3             4.6          3.1           1.5          0.2     setosa
4             5.0          3.6           1.4          0.2     setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  virginica
```

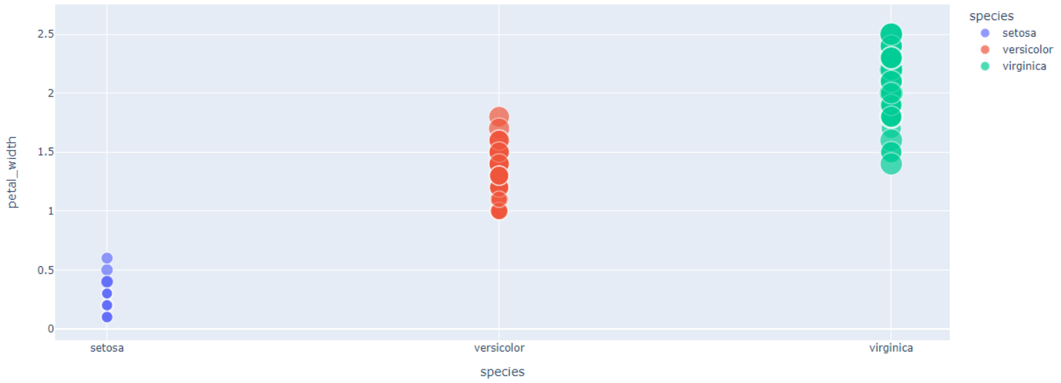
```
146          6.3          2.5          5.0          1.9  virginica
147          6.5          3.0          5.2          2.0  virginica
148          6.2          3.4          5.4          2.3  virginica
149          5.9          3.0          5.1          1.8  virginica

     species_id
0             1
1             1
2             1
3             1
4             1
..          ...
145           3
146           3
147           3
148           3
149           3

[150 rows x 6 columns]
```

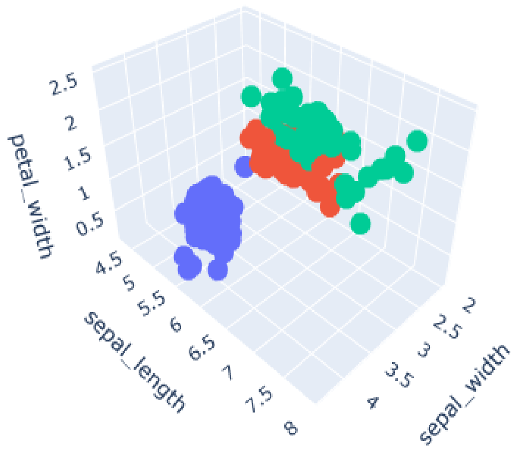





sum of petal_width
50
40
30
20
10
0
4
4.5
5
5.5
6
6.5
7
7.5
8
sepal_length


petal_width
2.5
2
1.5
1
0.5
0
setosa
versicolor
virginica
species


petal_width
2.5
2
1.5
1
0.5
0
setosa
versicolor
virginica
species
species
setosa
versicolor
virginica


petal_width
2.5
2
1.5
1
0.5
sepal_length
4.5
5
5.5
6
6.5
7
7.5
8
sepal_width
2
2.5
3
3.5
4

PROGRAM NO: 6 **DATE :06-10-22**

AIM: Generate classification report and confusion matrix of diabetes dataset using Naïve Bayes algorithm

SOURCE CODE

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
import pandas as pd
df=pd.read_csv("./diabetes.csv")
df.head()
X=df.drop("Outcome",axis=1)
y=df["Outcome"]
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size = 0.2, random_state=42)
gnb=GaussianNB()
gnb.fit(X_train, y_train)
y_predl=gnb.predict(X_test)
print(y_predl)
from sklearn.metrics import accuracy_score
acl=accuracy_score(y_test,y_predl)
print(acl)
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
matrix = confusion_matrix(y_test,y_predl)
print("confusion matrix: In",matrix)
cr=classification_report(y_test,y_predl)
print( "Classification Report In ",cr)
```

OUTPUT:

```
[0 0 0 0 1 1 0 1 0 1 0 1 1 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 1 1 1 1
 0 1 0 0 0 0 1 0 1 1 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 1 1 0 1 0 1 0 1 1 0 0 0
 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 0 1 0
 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 1 1 1 1 1 0 1 1 0 0 1 1 0 0 0 0 1 0 0 0 0
 0 1 0 0 1 0]
0.7662337662337663
confusion matrix: In [[79 20]
 [16 39]]
Classification Report In               precision    recall  f1-score   support

           0       0.83      0.80      0.81        99
           1       0.66      0.71      0.68        55

    accuracy                           0.77       154
   macro avg       0.75      0.75      0.75       154
weighted avg       0.77      0.77      0.77       154
```

PROGRAM NO: 7 **DATE :13-10-22**

AIM: Generate classification report and confusion matrix of diabetes dataset using KNN algorithm

SOURCE CODE

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
df=pd.read_csv("./diabetes.csv")
x=df.drop("Outcome",axis =1)
y=df["Outcome"]
x_train,x_test,y_train,y_test= train_test_split(x,y, test_size=0.2,random_state=42)
knn = KNeighborsClassifier (n_neighbors=7)
knn.fit(x_train,y_train)
y_pred=knn.predict(x_test)
from sklearn.metrics import accuracy_score
ac= accuracy_score(y_test,y_pred)
print(ac)
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
matrix= confusion_matrix(y_test,y_pred)
print("confusion matrix: \n", matrix)
cr=classification_report(y_test,y_pred)
print("Classification Report in'\n",cr)
```

OUTPUT:

```
0.6883116883116883
confusion matrix:
 [[72 27]
 [21 34]]
Classification Report in'

precision    recall  f1-score   support

           0       0.77      0.73      0.75        99
           1       0.56      0.62      0.59        55

    accuracy                           0.69       154
   macro avg       0.67      0.67      0.67       154
weighted avg       0.70      0.69      0.69       154
```

PROGRAM NO:8 **DATE :18-10-22**

AIM: Implement decision tree algorithm, find accuracy in pima india diabetes dataset

SOURCE CODE

```
import pandas as pd
df=pd.read_csv("./diabetes.csv")
col = df.columns
print(col)
X=df.drop("Outcome",axis=1)
y=df["Outcome"]
display(X.shape,y.shape)
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=50,test_size=0.25)
classifier=DecisionTreeClassifier()
classifier.fit(X_train,y_train)
y_pred=classifier.predict(X_test)
from sklearn.metrics import accuracy_score
print('Accuraccy on train data using gini
:',accuracy_score(y_train,y_pred=classifier.predict(X_train)))
print('Accuracy on test data using gini:',accuracy_score(y_test,y_pred))
```

OUTPUT:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
(768, 8)
(768,)
Accuraccy on train data using gini : 1.0
Accuracy on test data using gini: 0.6822916666666666
```

PROGRAM NO: 9 **DATE :19-10-22**

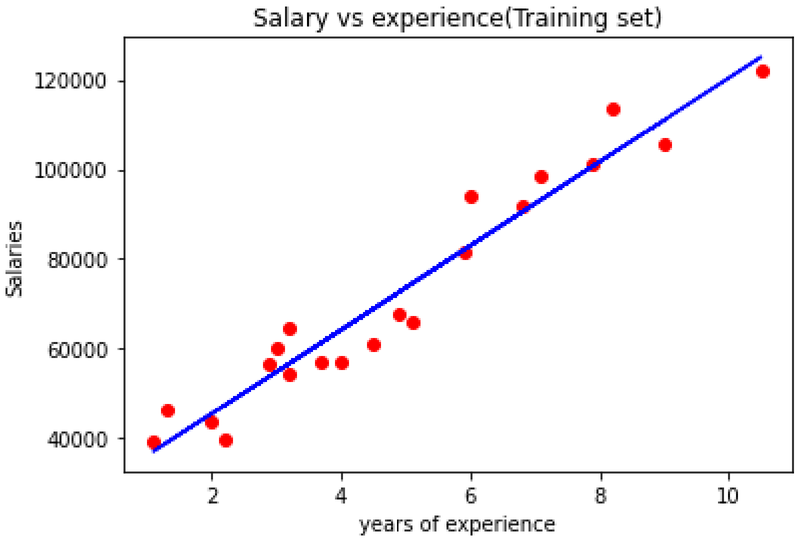
AIM: Implement Simple Linear Regression using Salary dataset

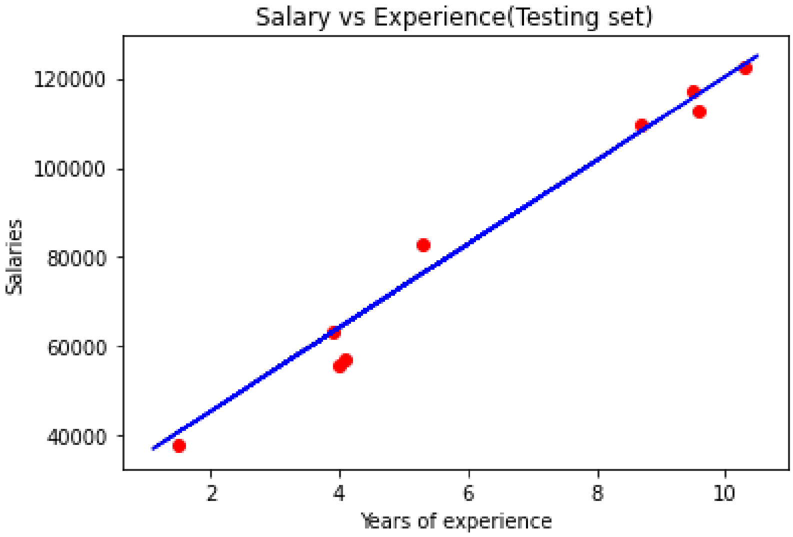
SOURCE CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
dataset=pd.read_csv("./Salary_Data.csv")
dataset.head()
X=dataset.iloc[:,:-1].values
y=dataset.iloc[:,1].values
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(X_train,y_train)
y_pred=regressor.predict(X_test)
y_pred
print('coefficient:',regressor.coef_)
print('intercept:',regressor.intercept_)
print(y_test)
plt.scatter(X_train,y_train,color='red')
plt.plot(X_train,regressor.predict(X_train),color='blue')
plt.title("Salary vs experience(Training set)")
plt.xlabel("years of experience")
plt.ylabel("Salaries")
plt.show()
plt.scatter(X_test,y_test,color='red')
plt.plot(X_train,regressor.predict(X_train),color='blue')
plt.title("Salary vs Experience(Testing set)")
plt.xlabel("Years of experience")
plt.ylabel("Salaries")
plt.show()
```

OUTPUT:

```
coefficient: [9360.26128619]
intercept: 26777.391341197625
[ 37731 122391  57081  63218 116969 109431 112635  55794  83088]
```





PROGRAM NO:10 DATE :21-10-22

AIM: Implement Multilinear regression using Advertising dataset

SOURCE CODE

```
import pandas as pd
dst=pd.read_csv('./advertising.csv')
dst.head()
x=dst.iloc[:,:-1]
y=dst.iloc[:,-1]
x.head()
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=100)
from sklearn.linear_model import LinearRegression
mr=LinearRegression()
mr.fit(x_train,y_train)
print('intercept:',mr.intercept_)
print("coefficients:")
list(zip(x,mr.coef_))
y_pred=mr.predict(x_test)
print("prediction:{}".format(y_pred))
```

OUTPUT:

```
intercept: 4.334595861728431
coefficients:
prediction:[ 9.35221067 20.96344625 16.48851064 20.10971005 21.67148354 16.16054424
 13.5618056  15.39338129 20.81980757 21.00537077 12.29451311 20.70848608
  8.17367308 16.82471534 10.48954832  9.99530649 16.34698901 14.5758119
 17.23065133 12.56890735 18.55715915 12.12402775 20.43312609 17.78017811
 16.73623408 21.60387629 20.13532087 10.82559967 19.12782848 14.84537816
 13.13597397  9.07757918 12.07834143 16.62824427  8.41792841 14.0456697
  9.92050209 14.26101605 16.76262961 17.17185467 18.88797595 15.50165469
 15.78688377 16.86266686 13.03405813 10.47673934 10.6141644  20.85264977
 10.1517568   6.88471443 17.88702583 18.16013938 12.55907083 16.28189561
 18.98024679 11.33714913  5.91026916 10.06159509 17.62383031 13.19628335]
```

PROGRAM NO:11 **DATE :26-10-22**

AIM: Implement SVM classification using diabetes dataset

SOURCE CODE

```
import pandas as pd
ds=pd.read_csv('./diabetes.csv')
ds.head()
x=ds.iloc[:,:-1]
x.head()
y=ds.iloc[:,-1]
y.head()
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=100)
from sklearn.svm import SVC
classifier=SVC(kernel='linear')
classifier.fit(x_train,y_train)
y_pred= classifier.predict(x_test)
from sklearn.metrics import accuracy_score
ac=accuracy_score(y_test,y_pred)
print("Accuracy Score:",ac)
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
print("Confusion Matrix:\n",cm)
from sklearn.metrics import classification_report
cr=classification_report(y_test,y_pred)
print("Classification Report:\n",cr)
```

OUTPUT:

```
Accuracy Score: 0.7445887445887446
Confusion Matrix:
 [[124  26]
 [ 33  48]]
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.83      0.81       150
           1       0.65      0.59      0.62        81

    accuracy                           0.74       231
   macro avg       0.72      0.71      0.71       231
weighted avg       0.74      0.74      0.74       231
```

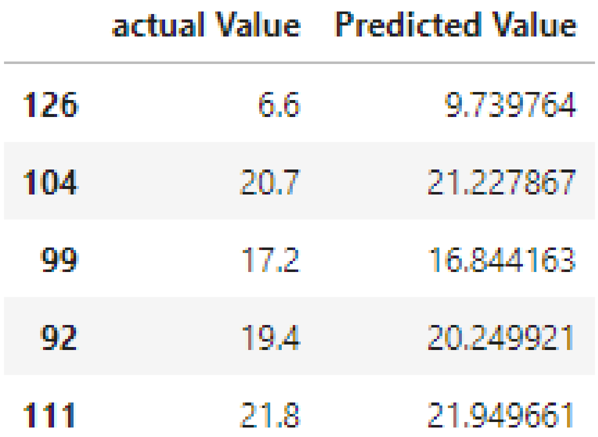
PROGRAM NO:12 **DATE :27-10-22**

AIM: Implement SVM regression using Advertising dataset

SOURCE CODE

```
import pandas as pd
dataset=pd.read_csv("/content/advertising.csv")
x=dataset.iloc[:,:-1]
y=dataset.iloc[:,-1]
dataset.head()
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=100)
from sklearn.svm import SVR
Re=SVR(kernel='linear')
Re.fit(x_train,y_train)
y_pred=Re.predict(x_test)
diff=pd.DataFrame({'actual Value':y_test,'Predicted Value':y_pred})
diff.head()
```

OUTPUT:



	actual Value	Predicted Value
126	6.6	9.739764
104	20.7	21.227867
99	17.2	16.844163
92	19.4	20.249921
111	21.8	21.949661

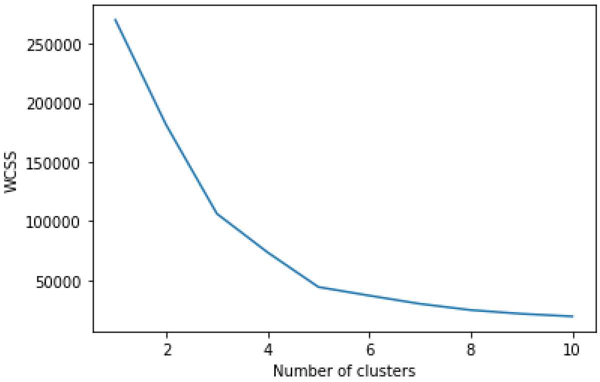
PROGRAM NO:13 **DATE :02-11-22**

AIM: Implement Kmeans using Mall customer dataset

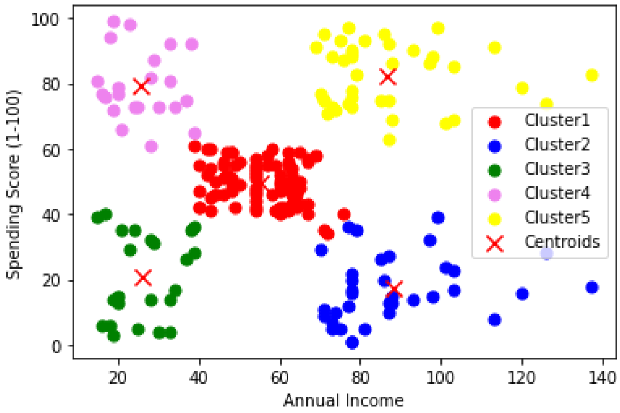
SOURCE CODE

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn
dataset=pd.read_csv('Mall_Customers.csv')
X= dataset.iloc[:, [3, 4]].values
from sklearn.cluster import KMeans
wcss=[]
for i in range(1, 11):
  kmeans=KMeans(n_clusters=i,init='k-means++',random_state=42)
  kmeans.fit(X)
  wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
plt.plot(range(1, 11), wcss)
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
kmeans=KMeans(n_clusters =5, init = "k-means++", random_state = 42)
y_kmeans = kmeans.fit_predict(X)
print(y_kmeans)
plt.scatter(X[y_kmeans== 0, 0], X[y_kmeans ==0, 1], s = 50, c = 'red', label = 'Cluster1')
plt.scatter(X[y_kmeans== 1, 0], X[y_kmeans==1,1], s= 50, c = 'blue', label = 'Cluster2')
plt.scatter(X[y_kmeans== 2, 0], X[y_kmeans== 2, 1], s= 50, c = 'green', label = 'Cluster3')
plt.scatter(X[y_kmeans== 3, 0], X[y_kmeans== 3, 1], s= 50, c = 'violet', label = 'Cluster4')
plt.scatter(X[y_kmeans== 4, 0], X[y_kmeans== 4, 1], s= 50, c = 'yellow', label = 'Cluster5')
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=100,marker='x',c='red'
,label='Centroids')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

OUTPUT:



```
[2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2
 3 2 3 2 3 2 0 2 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 4 1 4 0 4 1 4 1 4 0 4 1 4 1 4 1 4 1 4 0 4 1 4 1 4
 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1
 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4]
```



PROGRAM NO:14 **DATE :03-11-22**

AIM: Implement Web scraping using python

SOURCE CODE

```
!pip install autoscraper
from autoscraper import AutoScraper
url="https://www.geeksforgeeks.org/what-is-web-scraping-and-how-to-use-it/"
wanted_list=['Self-built Web Scrapers']
Scraper=AutoScraper()
result=Scraper.build(url,wanted_list)
print(result)
```

OUTPUT:

```
 ['Web Scraping', 'crawler', 'Self-built Web Scrapers', 'Browser extensions Web
Scrapers', 'Cloud Web Scrapers']
```

PROGRAM NO:15 **DATE :09-11-22**

AIM: Implement problem on Natural Language Processing-part of speech, tagging Ngram using NLTK.

SOURCE CODE

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize,sent_tokenize
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
stop_words = set(stopwords.words('english'))
txt ="Hello. MCA S3 is fantastic. We learn many new concepts and implement them in
our practical exams. "
"Ist of all the data science is a new paper. "
tokenized= sent_tokenize(txt)
for i in tokenized:
  wordsList= nltk.word_tokenize(i)
  wordsList= [w for w in wordsList if not w in stop_words]
  tagged = nltk.pos_tag(wordsList)
  print(tagged)

def generate_N_grams(text,ngram=1):
  words=[word for word in text.split(" ") if word not in set(stopwords.words('english'))]
  print("Sentence after removing stopwords:",words)
  temp=zip(*[words[i:] for i in range(0,ngram)])
  ans=[''.join(ngram) for ngram in temp]
  return ans
generate_N_grams("The sun rises in the east",2)
generate_N_grams("The sun rises in the east",3)
generate_N_grams("The sun rises in the east",4)
```

OUTPUT:

```
 [('Hello', 'NNP'), ('.', '.')]
```

```
[('MCA', 'NNP'), ('S3', 'NNP'), ('fantastic', 'JJ'), ('.', '.')]
[('We', 'PRP'), ('learn', 'VBP'), ('many', 'JJ'), ('new', 'JJ'), ('concepts', 'NNS'), ('implement', 'JJ'),
('practical', 'JJ'), ('exams', 'NN'), ('.', '.')]
[('Ist', 'NNP'), ('data', 'NNS'), ('science', 'NN'), ('new', 'JJ'), ('paper', 'NN'), ('.', '.')]

Sentence after removing stopwords: ['The', 'sun', 'rises', 'east']
['Thesun', 'sunrises', 'riseseast']
Sentence after removing stopwords: ['The', 'sun', 'rises', 'east']
['Thesunrises', 'sunriseseast']
Sentence after removing stopwords: ['The', 'sun', 'rises', 'east']
['Thesunriseseast']
```

PROGRAM NO:16 **DATE :13-11-22**

AIM: Program on CNN to classify images from any standard sataset in the public domain using the keras framework

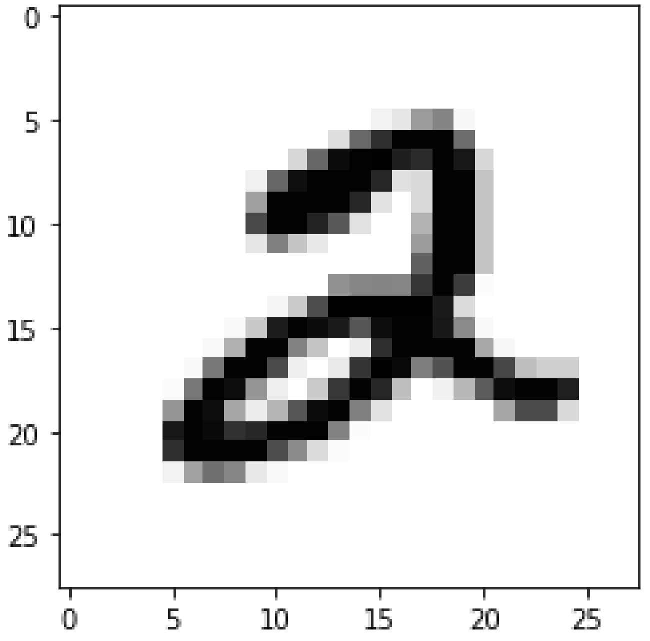
SOURCE CODE

```
from keras. datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils
(X_train, y_train), (X_test, y_test)= mnist.load_data()
print("X_train shape",X_train. shape)
print("y_train shape", y_train.shape)
print("X_test shape", X_test. shape)
print("y_test shape", y_test.shape)
import matplotlib.pyplot as plt
plt.imshow(X_train[5], cmap=plt.cm.binary)
print(y_train[5])
X_train= X_train.reshape(60000, 784)
X_test= X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test= X_test.astype('float32')
X_train/= 255
X_test/=255
X_train.shape
n_classes = 10
Y_train= np_utils.to_categorical(y_train, n_classes)
Y_test=np_utils.to_categorical(y_test, n_classes)
model = Sequential()
model.add(Dense(100,input_shape=(784,), activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
model.compile(loss='categorical_crossentropy',metrics=['accuracy'], optimizer='adam')
model. fit(X_train, Y_train, batch_size=100, epochs=10)
test_loss, test_acc= model.evaluate(X_test, Y_test)
print("TEST ACCURACY",round(test_acc,3))
print("TEST LOSS",round(test_loss,3))
```

OUTPUT:

```
X_train shape (60000, 28, 28)
y_train shape (60000,)
X_test shape (10000, 28, 28)
y_test shape (10000,)
2
Model: "sequential_1"
_________________________________________________________________
 Layer (type)              Output Shape            Param #
=================================================================
 dense_2 (Dense)           (None, 100)              78500

 dense_3 (Dense)           (None, 10)              1010

=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
600/600 [==============================] - 3s 4ms/step - loss: 0.3592 - accuracy: 0.9001
Epoch 2/10
600/600 [==============================] - 2s 4ms/step - loss: 0.1646 - accuracy: 0.9534
Epoch 3/10
600/600 [==============================] - 2s 4ms/step - loss: 0.1170 - accuracy: 0.9672
Epoch 4/10
600/600 [==============================] - 2s 4ms/step - loss: 0.0915 - accuracy: 0.9736
Epoch 5/10
600/600 [==============================] - 2s 4ms/step - loss: 0.0759 - accuracy: 0.9777
Epoch 6/10
```

```
600/600 [==============================] - 2s 4ms/step - loss: 0.0628 - accuracy:
0.9818
Epoch 7/10
600/600 [==============================] - 3s 5ms/step - loss: 0.0527 - accuracy:
0.9846
Epoch 8/10
600/600 [==============================] - 2s 4ms/step - loss: 0.0453 - accuracy:
0.9870
Epoch 9/10
600/600 [==============================] - 2s 4ms/step - loss: 0.0394 - accuracy:
0.9883
Epoch 10/10
600/600 [==============================] - 2s 4ms/step - loss: 0.0339 - accuracy:
0.9904
313/313 [==============================] - 1s 2ms/step - loss: 0.0828 - accuracy:
0.9748
TEST ACCURACY 0.975
TEST LOSS 0.083
```


0
5
10
15
20
25
0
5
10
15
20
25