DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS

MACHINE LEARNING LAB

LAB REPORT

III SEMESTER

Faculty In-Charge
Dr. R Saraswathi
Associate Professor & HoD

SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(Autonomous)
(Approved by AICTE, New Delhi, Affiliated to JNTUA, Anantapuramu)
Murukambattu, Chittoor-517127

2024-2025

Appropriate datasets from the following repository can be utilized:

1. https://www.kaggle.com/datasets

2. http://sci2s.ugr.es/keel/datasets.php#sub1

List of Experiments:

1. Demonstrate how do you structure data in Machine Learning.

2.Implement Data Preprocessing techniques on real time dataset.

3. Implement Feature subset selection techniques.

4.Implement and demonstrate Simple Linear Regression. Use the appropriate data set.

5. Implementation of Multiple Linear Regression for House Price Prediction using sklearn.

6. Demonstration of Regularization-LASSO, Ridge regression using appropriate data set.

7. Implementation of Logistic Regression using sklearn.

8. Write a program to demonstrate the working of the decision tree algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

9. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

10.Implement SVM with different kernel methods.

11. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set.

Print both correct and wrong predictions.

12. Write a program to demonstrate the working of K-Means Clustering.

1. Demonstrate how do you structure data in Machine Learning.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
data={
    'Name':['John','Asha','Kumar','Priya','Ali'],
    'CGPA':[7.0,8.5,6.0,9.0,5.5],
    'Internships':[1,2,0,3,0],
    'Placed':['Yes','Yes','No','Yes','No']
}
df=pd.DataFrame(data)
df=df.drop('Name',axis=1)
encoder=LabelEncoder()
df['Placed']=encoder.fit_transform(df['Placed'])
X=df[['CGPA','Internships']]
y=df['Placed']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.4,random_state=42)
print("Training Features:\n",X_train)
print("\nTraining Labels:\n",y_train
```

OUTPUT:

Training Features:

	CGPA	Internships
2	6.0	0
0	7.0	1
3	9.0	3

Training Labels:

2 0
0 1
3 1
Name: Placed, dtype: int32

2. To Implement Data Preprocessing techniques on real time dataset.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.preprocessing import MinMaxScaler
df=pd.read_csv(r'Z:\iris1.csv')
print("original dataset:")
print(df)
print("original dataset(first 5 records):")
print(df.head())
df_mean=df.fillna(df.mean(numeric_only=True))
print("\n after filling with mean(first 5 records):")
print(df_mean.head())
df_median=df.fillna(df.median(numeric_only=True))
print("\n after filling with median(first 5 records):")
print(df_median.head())
df_zeros=df.fillna(0)
print("\n after filling with zero(first 5 records):")
print(df_zeros.head())
df_pad=df.fillna(method='pad')
print("\n after fowardfill(pad)(first 5 records):")
print(df_pad.head())
df_backfill=df.fillna(method='bfill')
print("\n after backfill(first 5 records):")
```

```
print(df_backfill.head())
df_mean.to_csv("output_mean.csv",index=False)
df_median.to_csv("output_median.csv",index=False)
df_zeros.to_csv("output_zeros.csv",index=False)
df_pad.to_csv("output_pad.csv",index=False)
df_backfill.to_csv("output_backfill.csv",index=False)
label_encoder=LabelEncoder()
df['species_encoded']=label_encoder.fit_transform(df['species'])
X=df.drop(['species','species_encoded'],axis=1)
y=df['species_encoded']
scaler=StandardScaler()
X_scaled=scaler.fit_transform(X)
X_train,X_test,y_train,y_test=train_test_split(X_scaled,y,test_size=0.3,random_state=42)
print("\n scaled Training Data:\n",X_train[:3])
print("\n Taining Labels:\n",y_train[:3])
print("\n preprocessing completed! Files saved as:")
print("-output_mean.csv")
print("-output_median.csv")
print("-output_zeros.csv")
print("-output_pad.csv")
print("-output_backfill.csv")
print("-output_scaling.csv")
```

OUTPUT:

original dataset:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	NaN	NaN	NaN	NaN	NaN
2	4.7	3.2	1.3	0.2	setosa
3	NaN	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	virginica
6	NaN	NaN	NaN	NaN	NaN
7	5.0	3.2	1.5	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
9	4.9	3.1	1.5	0.1	virginica
10	5.4	3.7	1.5	0.2	virginica
11	4.8	3.4	1.3	0.2	setosa
12	4.8	NaN	1.4	0.1	virginica
13	4.3	3.0	1.1	0.1	virginica
14	5.8	1.3	1.2	0.2	virginica
15	5.7	4.4	1.5	0.4	virginica

original dataset(first 5 records):

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	NaN	NaN	NaN	NaN	NaN
2	4.7	3.2	1.3	0.2	setosa
3	NaN	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

after filling with mean(first 5 records):

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.100000	3.500000	1.400000	0.200000	setosa
1	5.023077	3.253846	1.407143	0.207143	NaN
2	4.700000	3.200000	1.300000	0.200000	setosa
3	5.023077	3.100000	1.500000	0.200000	setosa
4	5.000000	3.600000	1.400000	0.200000	setosa

after filling with median(first 5 records):

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	5.0	3.2	1.4	0.2	NaN
2	4.7	3.2	1.3	0.2	setosa
3	5.0	3.1	1.5	0.2	setosa

```
4          5.0          3.6          1.4          0.2  setosa

 after filling with zero(first 5 records):
   sepal_length  sepal_width  petal_length  petal_width species
0          5.1          3.5          1.4          0.2  setosa
1          0.0          0.0          0.0          0.0       0
2          4.7          3.2          1.3          0.2  setosa
3          0.0          3.1          1.5          0.2  setosa
4          5.0          3.6          1.4          0.2  setosa

 after fowardfill(pad)(first 5 records):
   sepal_length  sepal_width  petal_length  petal_width species
0          5.1          3.5          1.4          0.2  setosa
1          5.1          3.5          1.4          0.2  setosa
2          4.7          3.2          1.3          0.2  setosa
3          4.7          3.1          1.5          0.2  setosa
4          5.0          3.6          1.4          0.2  setosa
after backfill(first 5 records):
   sepal_length  sepal_width  petal_length  petal_width species
0          5.1          3.5          1.4          0.2  setosa
1          4.7          3.2          1.3          0.2  setosa
2          4.7          3.2          1.3          0.2  setosa
3          5.0          3.1          1.5          0.2  setosa
4          5.0          3.6          1.4          0.2  setosa

 scaled Training Data:
 [[-0.51042417  0.21176572 -0.74535599 -0.08084521]
 [-1.42566752 -0.51269596 -0.0496904  -0.08084521]
 [-0.28161334 -0.22291129  0.64597519 -1.21267813]]

 Taining Labels:
 11    0
8     0
9     1
Name: species_encoded, dtype: int32

 preprocessing completed! Files saved as:
-output_mean.csv
-output_median.csv
-output_zeros.csv
-output_pad.csv
-output_backfill.csv
-output_scaling.csv
```

3. To Implement Feature subset selection techniques.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

from sklearn.feature_selection import SelectKBest,f_classif, RFE

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear_model import LogisticRegression

df=pd.read_csv(r'Z:\iris3.csv')

X=df.drop('target',axis=1)

y=df['target']

le=LabelEncoder()

y=le.fit_transform(y)

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)

print("Original Features:",list(X.columns))

print()

selector=SelectKBest(score_func=f_classif,k=2)

X_train_Kbest=selector.fit_transform(X_train,y_train)

selected_Kbest=X.columns[selector.get_support()]

print("SelectKbest Selected Features:",list(selected_Kbest))

print()

model=LogisticRegression(max_iter=200)

rfe=RFE(model,n_features_to_select=2)

X_train_rfe=rfe.fit_transform(X_train,y_train)

selected_rfe=X.columns[rfe.get_support()]

print("RFE Selected Features:",list(selected_rfe))
```

```
rf=RandomForestClassifier(n_estimators=100,random_state=42)
rf.fit(X_train,y_train)
importances=rf.feature_importances_
selected_rf=X.columns[importances.argsort()[-2:]]
print("RandomForest Selected Features:",list(selected_rf))
```

OUTPUT:

Original Features: ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']

SelectKbest Selected Features: ['petal_length', 'petal_width']

RFE Selected Features: ['petal_length', 'petal_width']

RandomForest Selected Features: ['petal_width', 'petal_length']

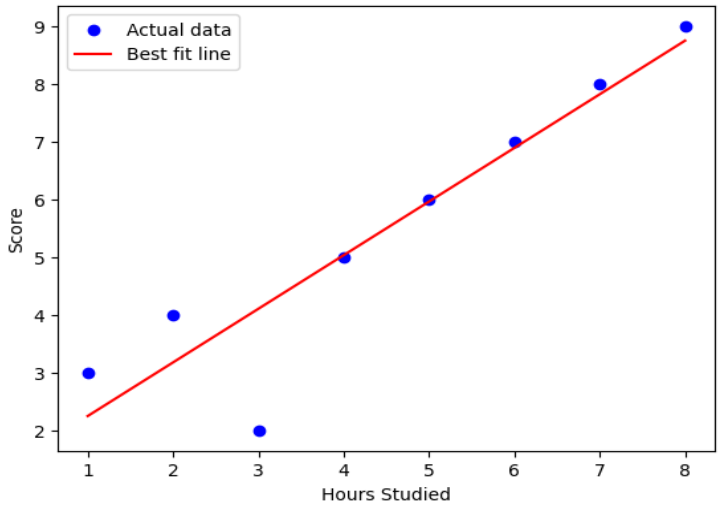
4. Implement and demonstrate Simple Linear Regression. Use the appropriate data set.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
data=pd.read_csv(r'F:\data1.csv')
print(data)
x=data[['Hours']]
y=data['Score']
model=LinearRegression()
model.fit(x,y)
y_pred=model.predict(x)
print("Slope (m):",model.coef_[0])
print("Intercept (c):",model.intercept_)
print("Mean Squared Error:",mean_squared_error(y,y_pred))
print("R2 Score:",r2_score(y,y_pred))
plt.scatter(X,y,color='blue',label='Actual data')
plt.plot(X,y_pred,color='red',label='Best fit line')
plt.xlabel("Hours Studied")
plt.ylabel("Score")
plt.legend()
plt.show()
```

OUTPUT

	Hours	Score
0	1	3
1	2	4
2	3	2
3	4	5
4	5	6
5	6	7
6	7	8
7	8	9

Slope (m): 0.9285714285714285
Intercept (c): 1.321428571428572
Mean Squared Error: 0.7232142857142857
R^2 Score: 0.8622448979591837



5. Implementation of Multiple Linear Regression for House Price Prediction using sklearn.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
import matplotlib.pyplot as plt
df=pd.read_csv(r'F:\house_prices.csv')
print("First 5 rows of dataset:")
print(df.head())
x=df[['Area','Bedrooms','Bathrooms','Age']]
y=df['Price']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
model=LinearRegression()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
print("\nPredicted Prices(first 10):",y_pred[:10])
print("Actual Prices(first 10): ",list(y_test.values[:10]))
print("\n Model Performance:")
print("Mean Squared Error(MSE):",mean_squared_error(y_test,y_pred))
print("R_Squared (R2score):",r2_score(y_test,y_pred))
print("\nModel Coefficients:")
for feature,coef in zip(x.columns,model.coef_):
print(f"{feature}:{coef}")
print("Intercept:",model.intercept_)
```

```
plt.scatter(y_test,y_pred,color="blue",alpha=0.6,label="Predictions")
plt.plot([y.min(),y.max()],[y.min(),y.max()],color="red",linewidth=2,label="Perfect Fit")
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted House Prices")
plt.legend()
plt.show()
```

OUTPUT

First 5 rows of dataset:

	Area	Bedrooms	Bathrooms	Age	Price
0	1200	2	1	15	200000
1	1500	3	2	12	250000
2	1800	3	2	10	280000
3	2000	4	2	8	320000
4	2200	4	3	6	350000

Predicted Prices(first 10): [192463.9869364 777726.9612811 721944.52080368 247650.77072673]

Actual Prices(first 10): [200000, 780000, 720000, 250000]

Model Performance:

Mean Squared Error(MSE): 17814559.3116829

R_Squared (R2score): 0.999744296268963

Model Coefficients:

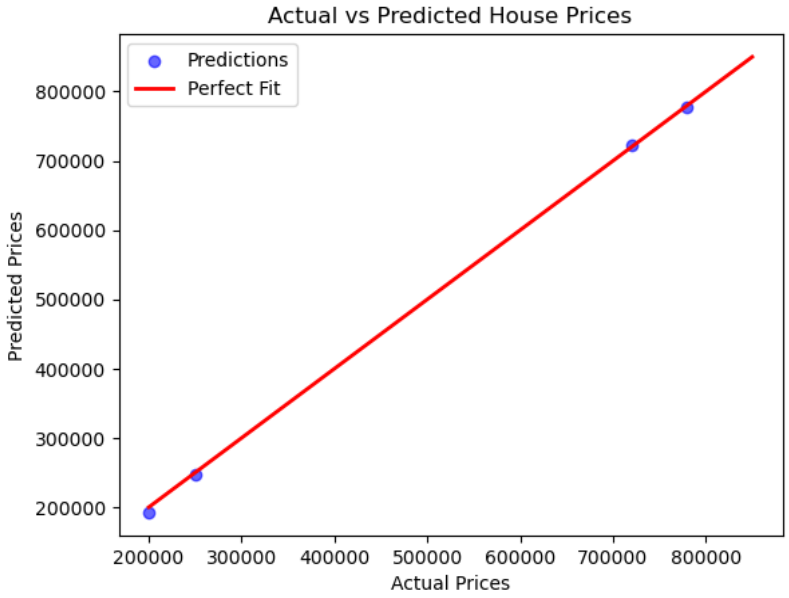
Area:104.33674830038743

Bedrooms:14912.013654397677

Bathrooms:4953.986156869679

Age:-1339.9198296482562

Intercept: 52580.672954993846



6. Demonstration of Regularization-LASSO, Ridge regression using appropriate data set.

```
Import  numpy  as  np
import  pandas as pd
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.metrics import mean_squared_error
data=load_diabetes()
x=pd.DataFrame(data.data,columns=data.feature_names)
y=data.target
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
lin_reg=LinearRegression()
lin_reg.fit(x_train,y_train)
y_pred_lin=lin_reg.predict(x_test)
print("Linear Regression MSE:",mean_squared_error(y_test,y_pred_lin))
ridge=Ridge (alpha=1.0)
```

```
ridge.fit(x_train,y_train)

y_pred_ridge=ridge.predict(x_test)

print("Ridge Regression MSE:",mean_squared_error(y_test,y_pred_ridge))

lasso=Lasso(alpha=0.1,max_iter=10000)

lasso.fit(x_train,y_train)

y_pred_lasso=lasso.predict(x_test)

print("Lasso Regression MSE:",mean_squared_error(y_test,y_pred_lasso))

coef_df=pd.DataFrame({

    "Feature":x.columns,

    "Linear":lin_reg.coef_,

    "Ridge":ridge.coef_,

    "Lasso":lasso.coef_

})

print("\n Coefficient comparison:\n",coef_df)
```

OUTPUT

```
Linear Regression MSE: 2900.1936284934804
Ridge Regression MSE: 3077.41593882723
Lasso Regression MSE: 2798.1934851697188

 Coefficient comparison:
   Feature      Linear       Ridge       Lasso
0     age   37.904021   45.367377    0.000000
1     sex -241.964362  -76.666086 -152.664779
2     bmi  542.428759  291.338832  552.697775
3      bp  347.703844  198.995817  303.365158
4      s1 -931.488846   -0.530310  -81.365007
```

```
5      s2  518.062277  -28.577050   -0.000000
6      s3  163.419983 -144.511905 -229.255776
7      s4  275.317902  119.260066    0.000000
8      s5  736.198859  230.221608  447.919525
9      s6   48.670657  112.149830   29.642617
```

7. To Implementation of Logistic Regression using sklearn.

```
import pandas as pd

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn import metrics

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

pima=pd.read_csv(r'f:\diabetes.csv')

s=pima.copy(deep=True)

x=s[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFu
nction','Age']]

y=s['Outcome']

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1)

logreg=LogisticRegression(max_iter=200)

logreg.fit(x_train,y_train)
```

```
y_pred=logreg.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test,y_pred))
print("Precision:",metrics.precision_score(y_test,y_pred))
print("Recall:",metrics.recall_score(y_test,y_pred))
cnf_matrix=metrics.confusion_matrix(y_test,y_pred)
print("confusion matrix:\n",cnf_matrix)
tn,fp,fn,tp=cnf_matrix.ravel()
accuracy=(tp+tn)/(tp+tn+fp+fn)
precision=tp/(tp+fp)
recall=tp/(tp+fn)
sensitivity=recall
specificity=tn/(tn+fp)
f_measure= 2 * (precision * recall)/(precision + recall)
print(f"specificity:{specificity:.3f}")
print(f"F_measure:{f_measure:.3f}")
sns.regplot(x=s["Glucose"],y=s["Outcome"],data=s, logistic=True)
plt.figure(figsize=(12,10))
cor=x_train.corr()
sns.heatmap(cor,annot=True,cmap=plt.cm.Reds)
plt.show()
y_vals=(accuracy,precision,recall,sensitivity,specificity,f_measure)
metrics_names=('accuracy','precision','recall','sensitivity','specificity','f_measure')
index=np.arange(len(metrics_names))
plt.bar(index,y_vals,color=['red','blue','yellow','black','green','cyan'])
plt.xticks(index,metrics_names)
```

```
plt.xlabel('Metrics')

plt.ylabel('Values')

plt.title('Model Performance')

plt.show()
```

OUTPUT

Accuracy: 0.7835497835497836

Precision: 0.7777777777777778

Recall: 0.5764705882352941

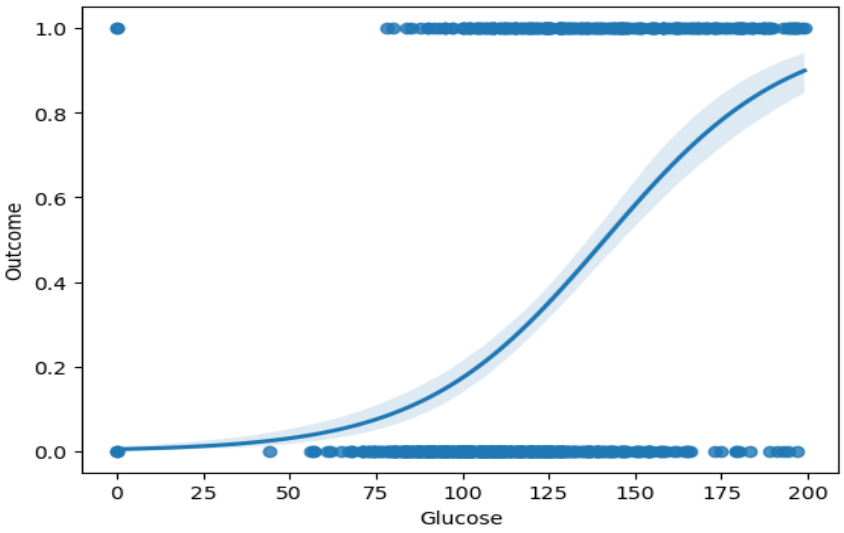
confusion matrix:

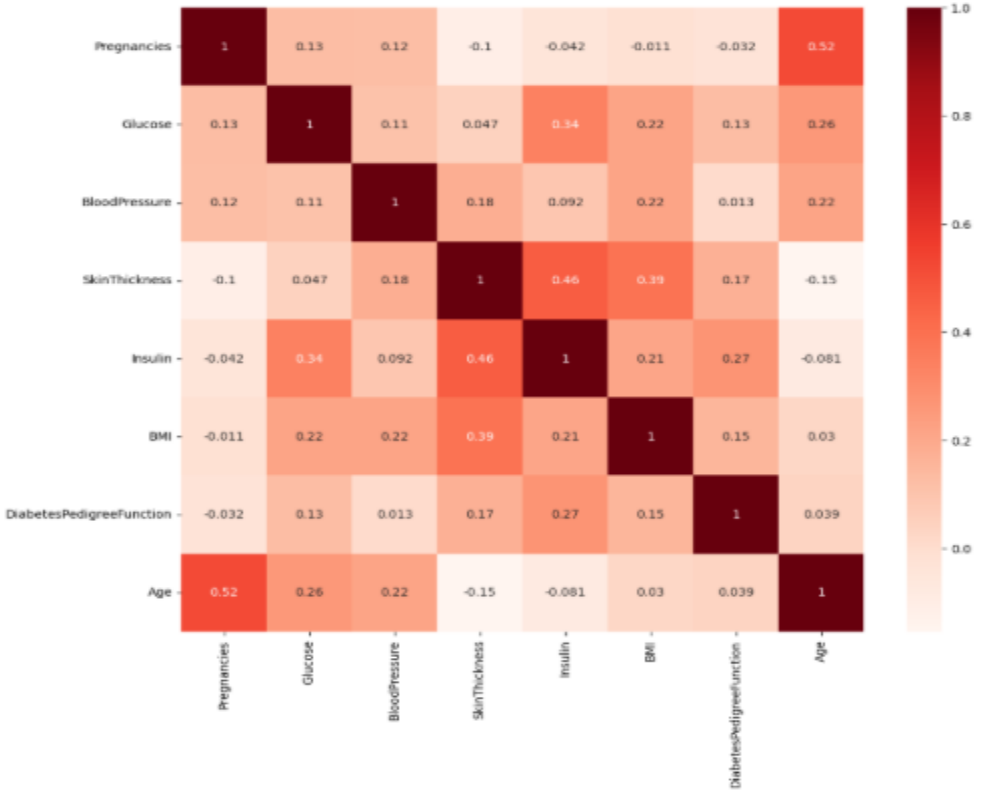
[[132 14]

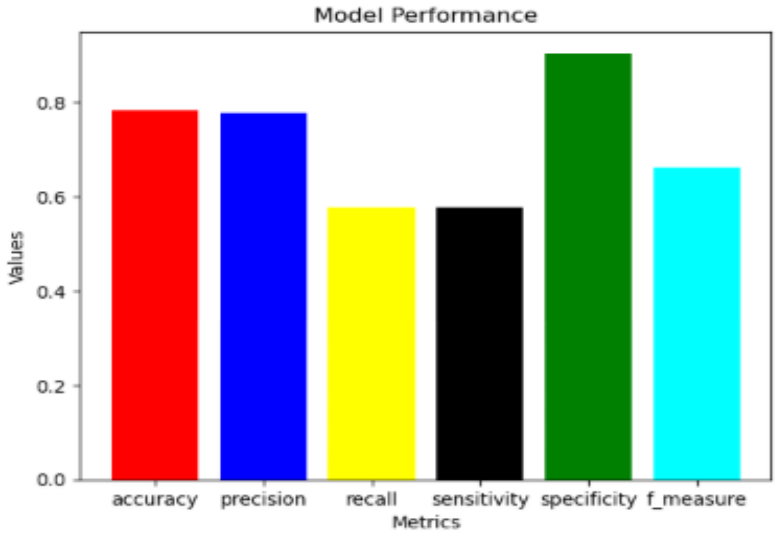
[36 49]]

specificity:0.904

F_measure:0.662




Pregnancies 1 0.13 0.12 -0.1 -0.042 -0.011 -0.032 0.52
Glucose 0.13 1 0.11 0.047 0.34 0.22 0.13 0.26
BloodPressure 0.12 0.11 1 0.18 0.092 0.22 0.013 0.22
SkinThickness -0.1 0.047 0.18 1 0.46 0.39 0.17 -0.15
Insulin -0.042 0.34 0.092 0.46 1 0.21 0.27 -0.081
BMI -0.011 0.22 0.22 0.39 0.21 1 0.15 0.03
DiabetesPedigreeFunction -0.032 0.13 0.013 0.17 0.27 0.15 1 0.039
Age 0.52 0.26 0.22 -0.15 -0.081 0.03 0.039 1
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
1.0
0.8
0.6
0.4
0.2
0.0


Model Performance
Values
0.0
0.2
0.4
0.6
0.8
accuracy
precision
recall
sensitivity
specificity
f_measure
Metrics

8. To demonstrate the Decision Tree Algorithm and to classify a new sample

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
pima=pd.read_csv(r'f:\diabetes.csv')
s=pima.copy(deep=True)
x=s.loc[:,['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']]
y=s.loc[:,'Outcome']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1)
x_train,x_test,y_train,y_test
clf=DecisionTreeClassifier()
clf=clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test,y_pred))
print("Precision:",metrics.precision_score(y_test,y_pred))
print("Recall:",metrics.recall_score(y_test,y_pred))
cnf_matrix=metrics.confusion_matrix(y_test,y_pred)
cnf_matrix
```

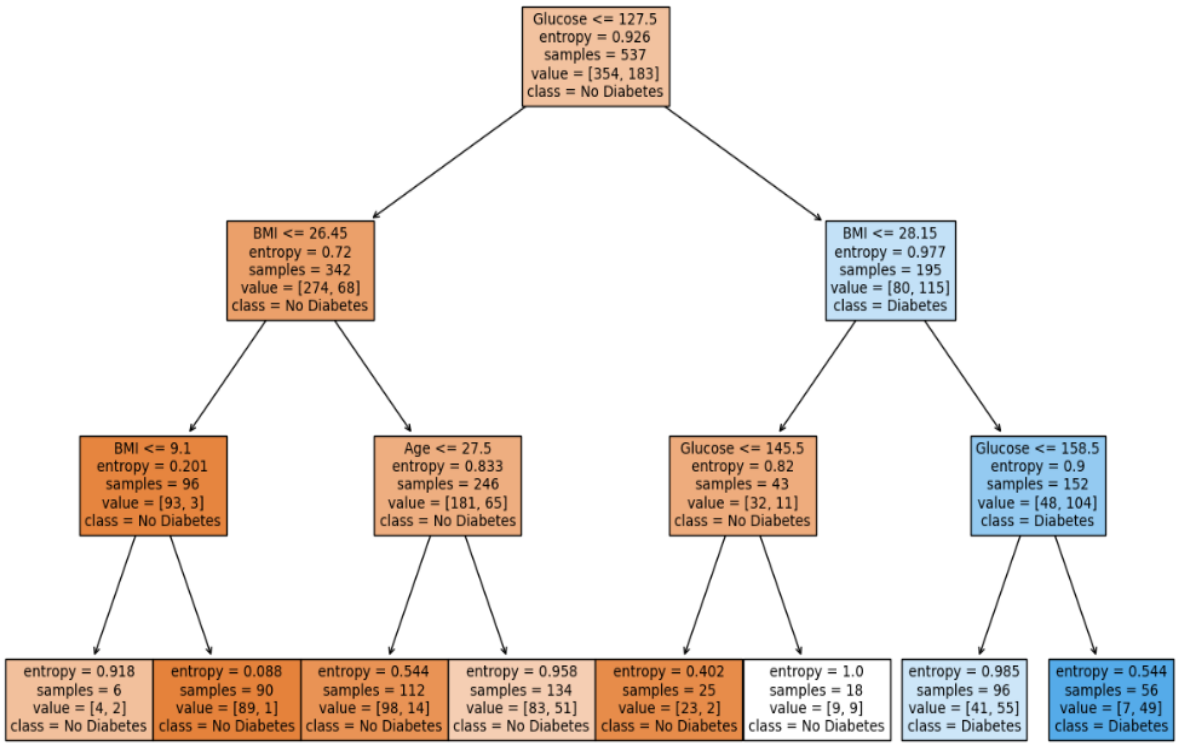
```
clf=DecisionTreeClassifier(criterion="entropy",max_depth=3)

clf=clf.fit(x_train,y_train)
plt.figure(figsize=(15,10))
plot_tree(clf,filled=True,feature_names=x.columns,class_names=['No Diabetes','Diabetes'])
plt.show()
```

OUTPUT

Accuracy: 0.7142857142857143

Precision: 0.6301369863013698

Recall: 0.5411764705882353



9. To implement the Naïve Bayesian Classifier

```
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn import metrics
pima=pd.read_csv(r'F:\diabetes.csv')
s=pima.copy(deep=True)
model=GaussianNB()
x=s.loc[:,['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']]
y=s.loc[:,'Outcome']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1)
model=model.fit(x_train,y_train)
y_pred=model.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test,y_pred))
print("Precision:",metrics.precision_score(y_test,y_pred))
print("recall:",metrics.recall_score(y_test,y_pred))
```

OUTPUT

```
Accuracy: 0.7835497835497836
Precision: 0.7464788732394366
recall: 0.6235294117647059
```

10.To implement SVM with different kernel Methods

```
import pandas as pd
from sklearn.model_selection import  train_test_split
from sklearn.svm  import SVC
from sklearn  import metrics
data=pd.read_csv(r'F:\diabetes.csv')
X=data.drop("Outcome",axis=1)
y=data["Outcome"]
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=42)
kernels=['linear','poly','rbf','sigmoid']
for kernel in kernels:
    print(f"\nSVM with {kernel} kernel")
    print("-" * 30)
model=SVC(kernel=kernel,gamma='scale')
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
accuracy=metrics.accuracy_score(y_test,y_pred)
print("Accuracy:",round(accuracy*100,2),"%")
```

OUTPUT

SVM with linear kernel

SVM with poly kernel

SVM with rbf kernel

SVM with sigmoid kernel
---

11. To implement K-Nearest Neighbour Algorithm to classify the correct and wrong prediction

```
import warnings
warnings.filterwarnings("ignore",category=FutureWarning)
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
iris=load_iris()
x=iris.data
y=iris.target
df=pd.DataFrame(x, columns=iris.feature_names)
df['target']=y
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.5,random_state=42)
k=15
model=KNeighborsClassifier(n_neighbors=k)
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
accuracy=accuracy_score(y_test,y_pred)
print("Accuracy:",accuracy*100,"%\n")
print("=== Correct Predictions ===")
```

```
for i in range(len(y_test)):
    if y_test[i]==y_pred[i]:

print(f"Sample{i}:Actual={iris.target_names[y_test[i]]},predicted={iris.target_names[y_pred[i]]
}")
print("\n ===Wrong Predictions ===")
for i in range(len(y_test)):
    if y_test[i]!=y_pred[i]:
        print(f"Sample
{i}:Actual={iris.target_names[y_test[i]]},predicted={iris.target_names[y_pred[i]]}")
```

OUTPUT

Accuracy: 97.33333333333334 %

=== Correct Predictions ===
Sample0:Actual=versicolor,predicted=versicolor
Sample1:Actual=setosa,predicted=setosa
Sample2:Actual=virginica,predicted=virginica
Sample3:Actual=versicolor,predicted=versicolor
Sample4:Actual=versicolor,predicted=versicolor
Sample5:Actual=setosa,predicted=setosa
Sample6:Actual=versicolor,predicted=versicolor
Sample7:Actual=virginica,predicted=virginica
Sample8:Actual=versicolor,predicted=versicolor
Sample9:Actual=versicolor,predicted=versicolor
Sample10:Actual=virginica,predicted=virginica
Sample11:Actual=setosa,predicted=setosa
Sample12:Actual=setosa,predicted=setosa
Sample13:Actual=setosa,predicted=setosa
Sample14:Actual=setosa,predicted=setosa
Sample15:Actual=versicolor,predicted=versicolor
Sample16:Actual=virginica,predicted=virginica
Sample17:Actual=versicolor,predicted=versicolor
Sample18:Actual=versicolor,predicted=versicolor
Sample19:Actual=virginica,predicted=virginica
Sample20:Actual=setosa,predicted=setosa
Sample21:Actual=virginica,predicted=virginica
Sample22:Actual=setosa,predicted=setosa
Sample23:Actual=virginica,predicted=virginica
Sample24:Actual=virginica,predicted=virginica
Sample25:Actual=virginica,predicted=virginica
Sample26:Actual=virginica,predicted=virginica
Sample27:Actual=virginica,predicted=virginica
Sample28:Actual=setosa,predicted=setosa
Sample29:Actual=setosa,predicted=setosa
Sample30:Actual=setosa,predicted=setosa
Sample31:Actual=setosa,predicted=setosa
Sample32:Actual=versicolor,predicted=versicolor
Sample33:Actual=setosa,predicted=setosa
Sample34:Actual=setosa,predicted=setosa
Sample35:Actual=virginica,predicted=virginica
Sample36:Actual=versicolor,predicted=versicolor
Sample37:Actual=setosa,predicted=setosa

Sample38:Actual=setosa,predicted=setosa
Sample39:Actual=setosa,predicted=setosa
Sample40:Actual=virginica,predicted=virginica
Sample41:Actual=versicolor,predicted=versicolor
Sample42:Actual=versicolor,predicted=versicolor
Sample43:Actual=setosa,predicted=setosa
Sample44:Actual=setosa,predicted=setosa
Sample45:Actual=versicolor,predicted=versicolor
Sample46:Actual=virginica,predicted=virginica
Sample47:Actual=virginica,predicted=virginica
Sample48:Actual=versicolor,predicted=versicolor
Sample49:Actual=virginica,predicted=virginica
Sample50:Actual=versicolor,predicted=versicolor
Sample51:Actual=virginica,predicted=virginica
Sample52:Actual=versicolor,predicted=versicolor
Sample53:Actual=setosa,predicted=setosa
Sample54:Actual=virginica,predicted=virginica
Sample55:Actual=versicolor,predicted=versicolor
Sample56:Actual=setosa,predicted=setosa
Sample57:Actual=setosa,predicted=setosa
Sample58:Actual=setosa,predicted=setosa
Sample59:Actual=versicolor,predicted=versicolor
Sample61:Actual=setosa,predicted=setosa
Sample62:Actual=setosa,predicted=setosa
Sample63:Actual=setosa,predicted=setosa
Sample64:Actual=versicolor,predicted=versicolor
Sample65:Actual=setosa,predicted=setosa
Sample66:Actual=versicolor,predicted=versicolor
Sample67:Actual=virginica,predicted=virginica
Sample68:Actual=setosa,predicted=setosa
Sample69:Actual=versicolor,predicted=versicolor
Sample70:Actual=virginica,predicted=virginica
Sample71:Actual=setosa,predicted=setosa
Sample73:Actual=virginica,predicted=virginica
Sample74:Actual=versicolor,predicted=versicolor

===Wrong Predictions ===
Sample 60:Actual=virginica,predicted=versicolor
Sample 72:Actual=virginica,predicted=versicolor

12. To demonstrate the working of K-Means Clustering

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
pd1=pd.read_csv(r'F:\diabetes.csv')
x=np.array(pd1.drop(['Outcome'],axis=1).astype(float))
kmeans=KMeans(n_clusters=2)
y=kmeans.fit(x)
data=kmeans.predict(x)
ss=silhouette_score(x,kmeans.labels_)
print(ss)
```

OUTPUT

```
0.5687897205830247
```