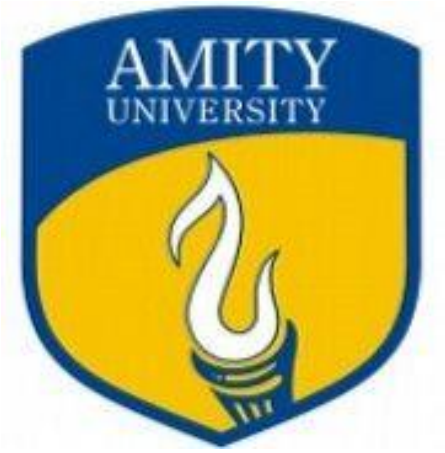
Lab File
Fundamentals of Machine Learning



Amity School of Engineering and Technology
Amity University Uttar Pradesh

Submitted To:
Mr. Deepak Gaur
ASET

Submitted By:
Kuldeep Dwivedi
A2305218477
B.Tech (6CSE7Y)

INDEX

Experiment No	Objective of the Experiment	Faculty Signature
1	Perform basic operations on matrices (like addition, subtraction, multiplication) and display specific rows or columns of the matrix.	

Experiment 1

Objective
Perform basic operations on matrices (like addition, subtraction, multiplication) and display specific rows or columns of the matrix.

Theory
In python matrix can be implemented as 2D list or 2D Array. Forming matrix from latter, gives the additional functionalities for performing various operations in matrix. These operations and array are defines in module "**numpy**".
Operation on Matrix:
add():- This function is used to perform **element wise matrix addition**.
subtract():- This function is used to perform **element wise matrix subtraction**.
divide():- This function is used to perform **element wise matrix division.**
multiply():- This function is used to perform **element wise matrix multiplication**.

Program

```
import numpy as np

x = np.array([[3,4],[6,7]])
y = np.array([[10,12],[21,23]])

#Matrix Addition
print("The element wise addition of matrix is: ")
print()
print(np.add(x,y))

#Matrix Subtraction
print("The element wise subtraction of matrix is: ")
print()
print(np.add(x,y))

#Matrix Division
print("The element wise division of matrix is: ")
print()
print(np.add(x,y))

#Matrix Multiplication
print("The element wise multiplication of matrix is: ")
print()
print(np.add(x,y))
```

Output

```
>>>
=== RESTART: C:/Users/hp/AppData/Local/Programs/Python/Python38-32/matrix.py ===
The element wise addition of matrix is:

[[13 16]
 [27 30]]
The element wise subtraction of matrix is:

[[13 16]
 [27 30]]
The element wise division of matrix is:

[[13 16]
 [27 30]]
The element wise multiplication of matrix is:

[[13 16]
 [27 30]]
>>>
```

Experiment 4

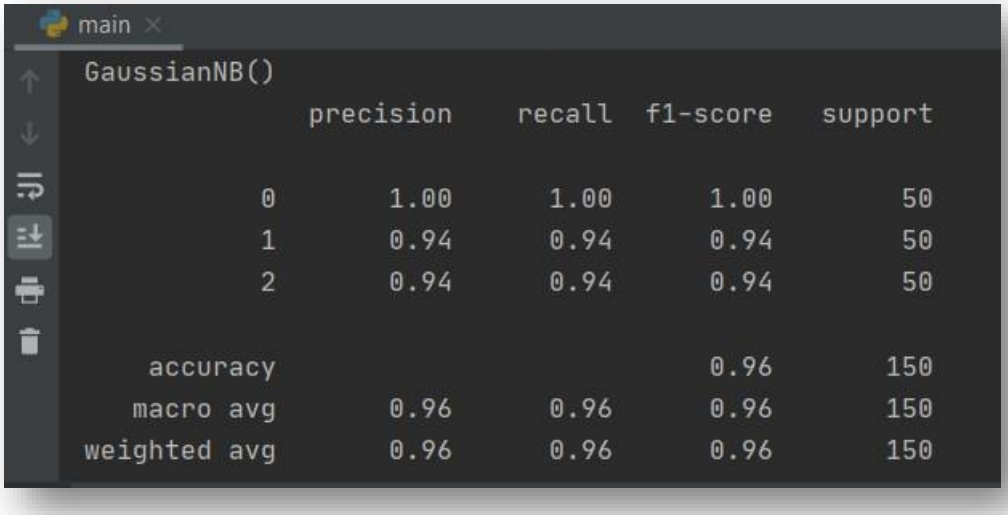
Objective
Implement and evaluate a classification using Naïve Bayes.

Theory:

Program:

```
from sklearn import datasets

from sklearn import metrics

from sklearn.naive_bayes import GaussianNB


dataset=datasets.load_iris()


model=GaussianNB()

model.fit(dataset.data, dataset.target)


print(model)

GaussianNB(priors=None)


expected=dataset.target

predicted=model.predict(dataset.data)


print(metrics.classification_report(expected, predicted))

print(metrics.confusion_matrix(expected, predicted))
```

Output:


main
GaussianNB()
precision recall f1-score support
0 1.00 1.00 1.00 50
1 0.94 0.94 0.94 50
2 0.94 0.94 0.94 50
accuracy 0.96 150
macro avg 0.96 0.96 0.96 150
weighted avg 0.96 0.96 0.96 150

Experiment 4

Objective

Implement Linear Regression problem. For example, based on a dataset comprising of existing set of prices and area/size of the houses, predict the estimated price of a given house.

Theory:

Linear Regression: Linear regression is used for finding linear relationship between target and one or more predictors. There are two types of linear regression- Simple and Multiple.

Simple Linear Regression

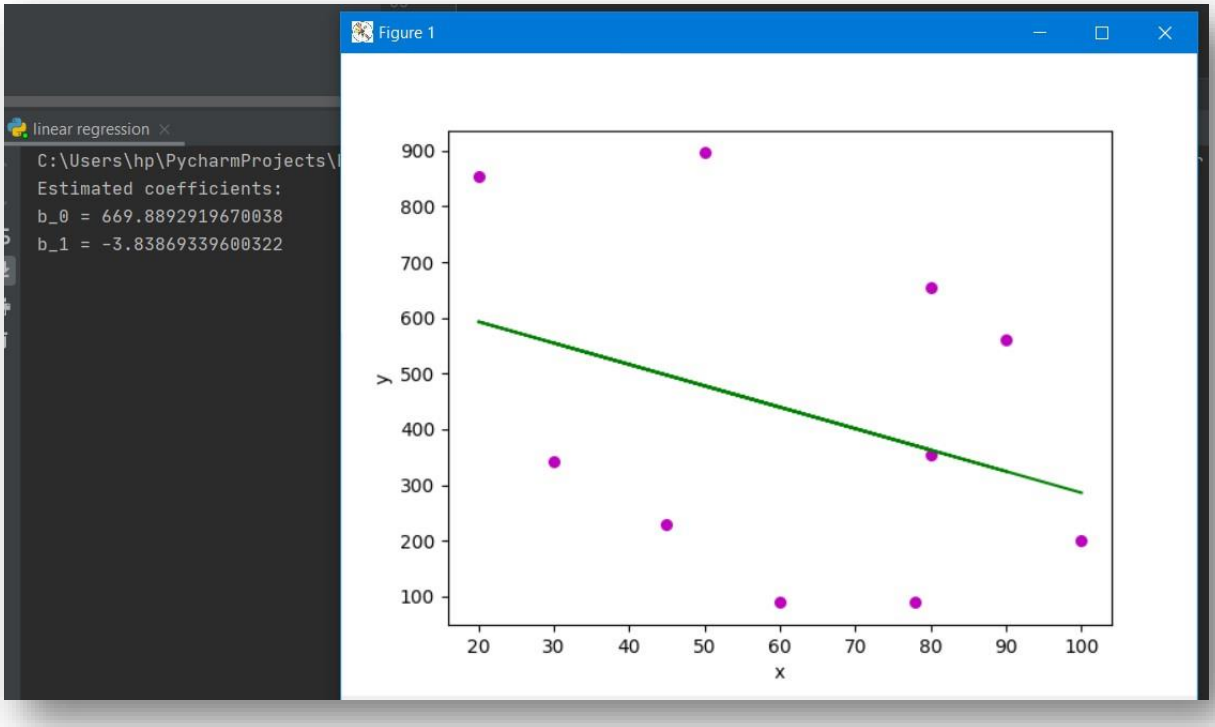
Simple linear regression is useful for finding relationship between two continuous variables. One is predictor or independent variable and other is response or dependent variable. It looks for statistical relationship but not deterministic relationship. Relationship between two variables is said to be deterministic if one variable can be accurately expressed by the other. For example, using temperature in degree Celsius it is possible to accurately predict Fahrenheit. Statistical relationship is not accurate in determining relationship between two variables. For example, relationship between height and weight.

Program:

```
import numpy as np
import matplotlib.pyplot as plt

def estimate_coef(x, y):
  # number of observations/points
  n = np.size(x)

  # mean of x and y vector
  m_x, m_y = np.mean(x), np.mean(y)

  # calculating cross-deviation and deviation about x
  SS_xy = np.sum(y*x) - n*m_y*m_x
  SS_xx = np.sum(x*x) - n*m_x*m_x

  # calculating regression coefficients
  b_1 = SS_xy / SS_xx
  b_0 = m_y - b_1*m_x

  return(b_0, b_1)

def plot_regression_line(x, y, b):
  # plotting the actual points as scatter plot
  plt.scatter(x, y, color = "m",
       marker = "o", s = 30)

  # predicted response vector
  y_pred = b[0] + b[1]*x

  # plotting the regression line
  plt.plot(x, y_pred, color = "g")
```

```
    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()

def main():
    # observations
    x = np.array([100, 80, 45, 20, 60, 30, 80, 50, 78, 90])
    y = np.array([200, 354, 230, 854, 89, 342, 654, 896, 90, 560])

    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {} \
      \nb_1 = {}".format(b[0], b[1]))

    # plotting regression line
    plot_regression_line(x, y, b)

if __name__ == "__main__":
    main()
```

Output:



Experiment 6

Objective
Implementation of Principle Component Analysis Algorithm.

Theory:

Principal Component Analysis (PCA) is an unsupervised, non-parametric statistical technique primarily used for dimensionality reduction in machine learning.

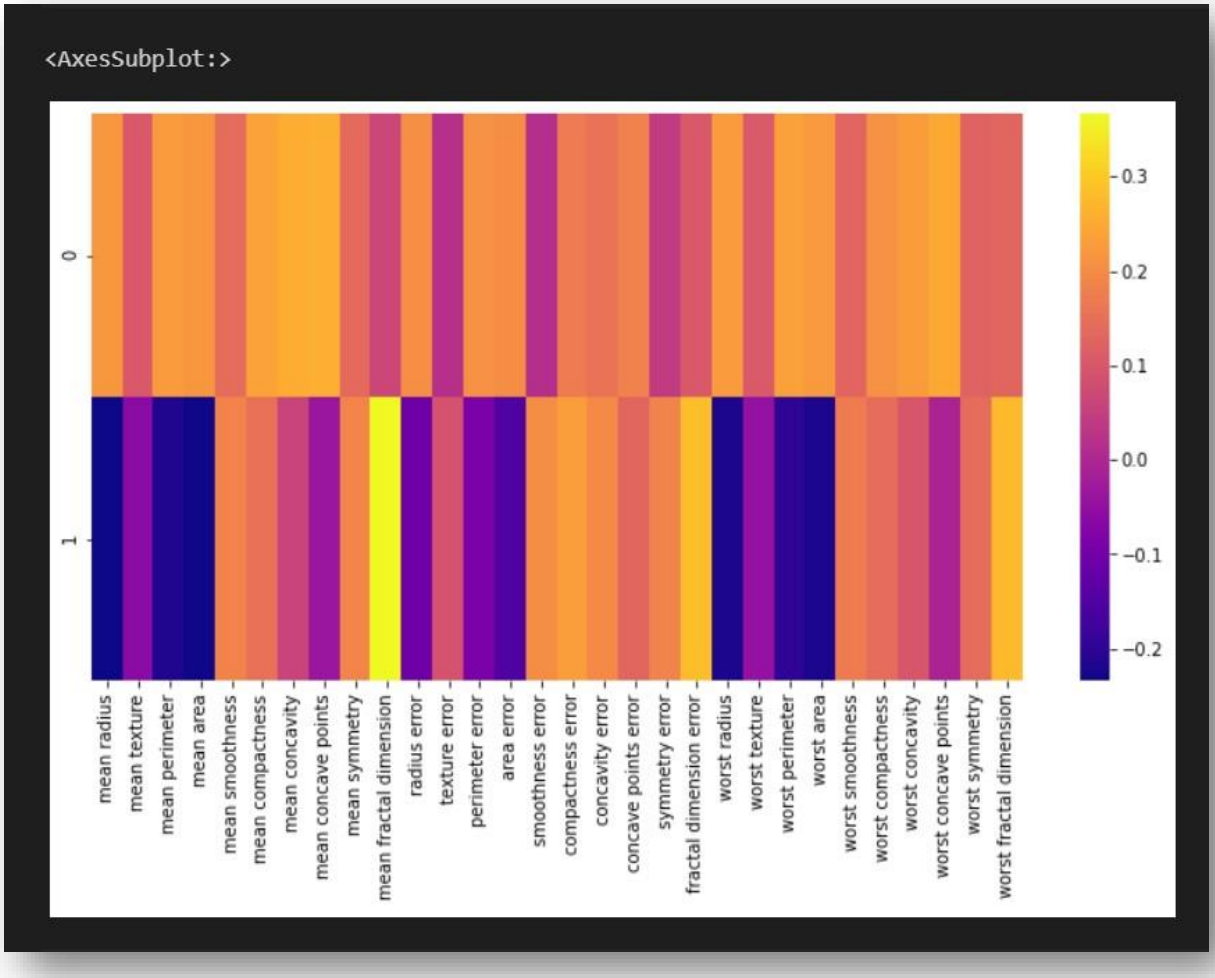
High dimensionality means that the dataset has a large number of features. The primary problem associated with high-dimensionality in the machine learning field is model overfitting, which reduces the ability to generalize beyond the examples in the training set.

Program:

```
%matplotlib inline

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Import cancer data set from sklearn

from sklearn.datasets import load_breast_cancer

cancer=load_breast_cancer()

cancer.keys()

# print target class

print(cancer['target_names'])

# printing of the dataset

df=pd.DataFrame(cancer['data'], columns=cancer['feature_names'])

df.head()

#Scaler the dataset: So that each feature has single unit variance

from sklearn.preprocessing import StandardScaler
```

```
scaler=StandardScaler()

scaler.fit(df)

scaled_data=scaler.transform(df)

#Import PCA and specify PCA components

from sklearn.decomposition import PCA

pca=PCA(n_components=2)

pca.fit(scaled_data)

#Transform this data to first 2 principal components

x_pca=pca.transform(scaled_data)

print(scaled_data.shape)

print(x_pca.shape)

#plot these two dimenstions

plt.figure(figsize=(8,6))

plt.scatter(x_pca[:,0],x_pca[:,1], c=cancer['target'], cmap='plasma')

plt.xlabel('First Principal Component')

plt.ylabel('Second Principal Component')

#Component corrosponds to combination of original features, components itself stored as attr
ibute of the fitted PCA

pca.components_

#in this numpy output each row represent a principal component, each column relate back to
original feature

# visualize using heatmap funtion: This heatmap and color bar basically represent correlation
between various features and princpal components

df_comp=pd.DataFrame(pca.components_,columns=cancer['feature_names'])

plt.figure(figsize=(12,6))

sns.heatmap(df_comp,cmap='plasma')
```

Output:


<AxesSubplot:>
0
1
mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension
0.3
0.2
0.1
0.0
−0.1
−0.2

Experiment 7

Objective
Implementation of L1 and L2 Regularization for overfitting and underfitting problem.

Theory:
Regularization: In mathematics, statistics, finance, computer science, particularly in machine learning and inverse problems, **regularization** is the process of adding information in order to solve an ill-posed problem or to prevent overfitting. **Regularization** applies to objective functions in ill-posed optimization problems.

Program:

```
%matplotlib inline

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt


# Read the dataset

dataset=pd.read_csv('d:\\data\\melb_data.csv')

dataset.head()

# printing of unique values in dataset

dataset.nunique()

# Shape of the dataset

dataset.shape

# Choose columns of your interest

cols_to_use=['Suburb','Rooms', 'Type','Method', 'SellerG', 'Regionname', 'Propertycount', 'Dis
tance', 'CouncilArea', 'Bedroom2', 'Bathroom', 'Car', 'Landsize','BuildingArea', 'Price']


dataset=dataset[cols_to_use]

dataset.head()

#check now shape of dataset

dataset.shape

#Checking for null values

dataset.isnull().sum()

# representing null values in heatmap function
```

```
sns.heatmap(dataset.isnull(), yticklabels=False, cmap='plasma')

# Remove of the null values

dataset.dropna(inplace=True)

# representing null values in heatmap function

sns.heatmap(dataset.isnull(), yticklabels=False, cmap='plasma')

#Checking for null values

dataset.isnull().sum()

dataset.head()

# Need dummy encoding for every column whose values are strings using pandas library

dataset=pd.get_dummies(dataset)

dataset.head()

# Define dependent and independent variables x and y

X=dataset.drop('Price', axis=1)

y# use of train_test_split from sklearn library
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=2)

=dataset['Price']

# import regular linear regression model

from sklearn.linear_model import LinearRegression

# create instance for model

reg=LinearRegression()

#fit the model

reg.fit(X_train, y_train)

# check score of model on test data

reg.score(X_test, y_test)

# check score of model on training data

reg.score(X_train, y_train)


# So, test data gave score in -
ve even, whereas training data gave score of 77%, so here arise the problem of overfitting of
dataset.
```

```
#import lasso model from sklearn
from sklearn import linear_model
lasso_reg=linear_model.Lasso(alpha=50, max_iter=100, tol=0.1)

lasso_reg.fit(X_train, y_train)
# now check score on testing values using Lasso model
lasso_reg.score(X_test, y_test)

## now check score on training values using Lasso model
lasso_reg.score(X_train, y_train)

# So, accuracy improved here on this dataset.
 So, accuracy improved here on this dataset.
# import Ridge model (L2 Regularizatio) from sklearn
from sklearn.linear_model import Ridge
ridge_reg=linear_model.Ridge(alpha=50, max_iter=100, tol=0.1)

ridge_reg.fit(X_train, y_train)
# now check score on testing values using Ridge model
ridge_reg.score(X_test, y_test)

# So, accuracy improved here on this dataset.
# now check score on training values using Lasso model
ridge_reg.score(X_train, y_train)

# So, accuracy improved here on this dataset.
```

Output:

```
# now check score on testing values using Ridge model
ridge_reg.score(X_test, y_test)

# So, accuracy improved here on this dataset.
```

```
0.5871285478332213
```

```
# now check score on training values using Lasso model
ridge_reg.score(X_train, y_train)

# So, accuracy improved here on this dataset.
```

```
0.6967602449724921
```

So, at last with using Lasso (L1 Regularization) and Ridge (L2 Regualrization), Accuracy of model increased.

Experiment 9

Objective:
Implementation of Neural network classifier.

Theory:

Neural Network Classifier: A neural network consists of units (neurons), arranged in layers, which convert an input vector into some output. Each unit takes an input, applies a (often nonlinear) function to it and then passes the output on to the next layer. Generally the networks are defined to be feed-forward: a unit feeds its output to all the units on the next layer, but there is no feedback to the previous layer. Weightings are applied to the signals passing from one unit to another, and it is these weightings which are tuned in the training phase to adapt a neural network to the particular problem at hand. This is the learning phase.

Program:

```
import numpy as np

import pandas as pd

import tensorflow as tf

tf.__version__

from google.colab import files

uploaded = files.upload()

import io

dataset = pd.read_csv(io.BytesIO(uploaded['d:\\data\\Churn_Modelling.csv']))

X = dataset.iloc[:, 3:-1].values

y = dataset.iloc[:, -1].values

print(X)

print(y)

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

X[:, 2] = le.fit_transform(X[:, 2])

print(X)

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passt
hrough')
```

```
X = np.array(ct.fit_transform(X))
print(X)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
print(X)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
ann = tf.keras.models.Sequential()
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

Experiment 9

Objective
Implement and evaluate a classification using SVM.

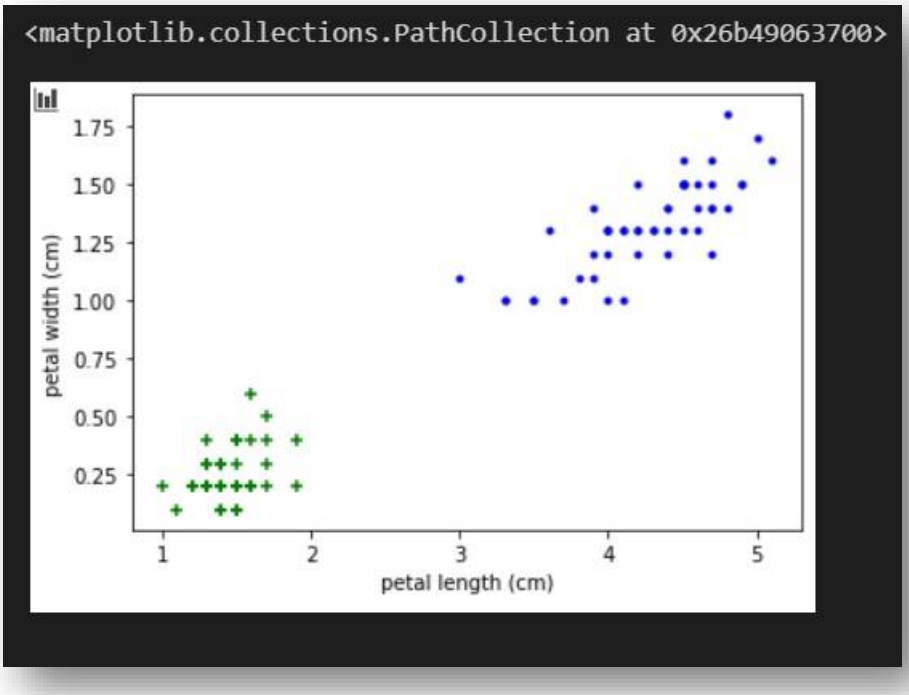
Theory:
An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).
Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems.

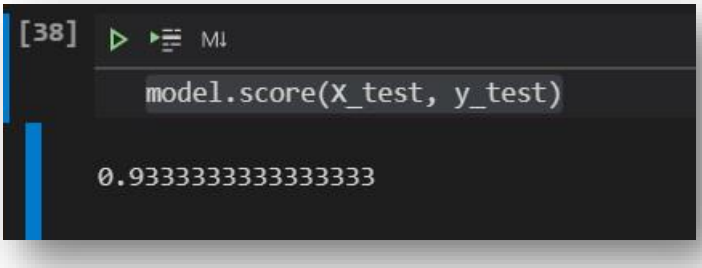
Program:

```
import pandas as pd

from sklearn.datasets import load_iris

iris=load_iris()

dir(iris)
iris.feature_names
df=pd.DataFrame(iris.data, columns=iris.feature_names)
df.head()
# Append traget column in this data frame
df['target'] = iris.target
df.head()
# Possible value of target is 0,1 and 3.
iris.target_names
# Check how many are setosa, versicolor or virginica
df[df.target==0].head()
# Check how many are setosa, versicolor or virginica
df[df.target==1].head()
# Check how many are setosa, versicolor or virginica
df[df.target==2].head()
# Add flower name in the dataset
df['flower_name'] = df.target.apply(lambda x: iris.target_names[x])
df.head()
from matplotlib import pyplot as plt
%matplotlib inline
# Divide the dataset into three separate data frames
df0=df[df.target==0]
df1=df[df.target==1]
df2=df[df.target==2]
df0.head()
df1.head()
df2.head()
# Scatter plot of these data frames
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
```

```
plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'], color='green', marker='+')
plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'], color='blue', marker='.')
# Plot petal lenth and petal width also
plt.xlabel('petal length (cm)')
plt.ylabel('petal width (cm)')
plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'], color='green', marker='+')
plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'], color='blue', marker='.')
from sklearn.model_selection import train_test_split
# drop target and flower name from the dataset as these two are not features
X=df.drop(['target','flower_name'], axis='columns')
X.head()
y=df.target
y
# split the dataset
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.2)
len(X_test)
len(X_train)
# import SVM
from sklearn.svm import SVC
model=SVC()
# fit the model
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

Output:




[38]
M↓
model.score(X_test, y_test)
0.9333333333333333