UNIVERSITY OF MUMBAI

DEPARTMENT OF COMPUTER SCIENCE

M.SC (Computer Science)

CERTIFICATE

Certified that the work entered in this journal was done in the computer laboratory by the student

Mr. Saurabh Vishwas Kadam

Seat No. 112056

Roll No.32 of the Class - **MSc computer Science**

First Year Sem 2

Semester during the year **2021-2022** in a Satisfactory manner.

Practical 1

AIM :- Implement Linear Regression (Diabetes Dataset)

THEORY : -

Logistic regression is a classification model in machine learning, extensively used in clinical analysis. It uses probabilistic estimations which helps in understanding the relationship between the dependent variable and one or more independent variables. Diabetes, being one of the most common diseases around the world, when detected early, may prevent the progression of the disease and avoid other complications. In this work, we design a prediction model, that predicts whether a patient has diabetes, based on certain diagnostic measurements included in the dataset, and explore various techniques to boost the performance and accuracy.

CODE/OUTPUT : -

```
# Import Dependencies

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets,linear_model,metrics

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score

import seaborn as sns


# Load the diabetes dataset

diabetes=datasets.load_diabetes()


# X - feature vectors

# y - Target values

X=diabetes.data

y=diabetes.target


# splitting X and y into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
```

```
random_state=1)

# Create linear regression object
lin_reg=linear_model.LinearRegression()

# Train the model using train and test data
lin_reg.fit(X_train,y_train)
```

```
Out[6]: LinearRegression()
```

```
# Predict values for X_test data
predicted = lin_reg.predict(X_test)

# Regression coefficients
print('\n Coefficients are:\n',lin_reg.coef_)

# Intercept
print('\nIntercept : ',lin_reg.intercept_)

# variance score: 1 means perfect prediction
print('Variance score: ',lin_reg.score(X_test, y_test))
```

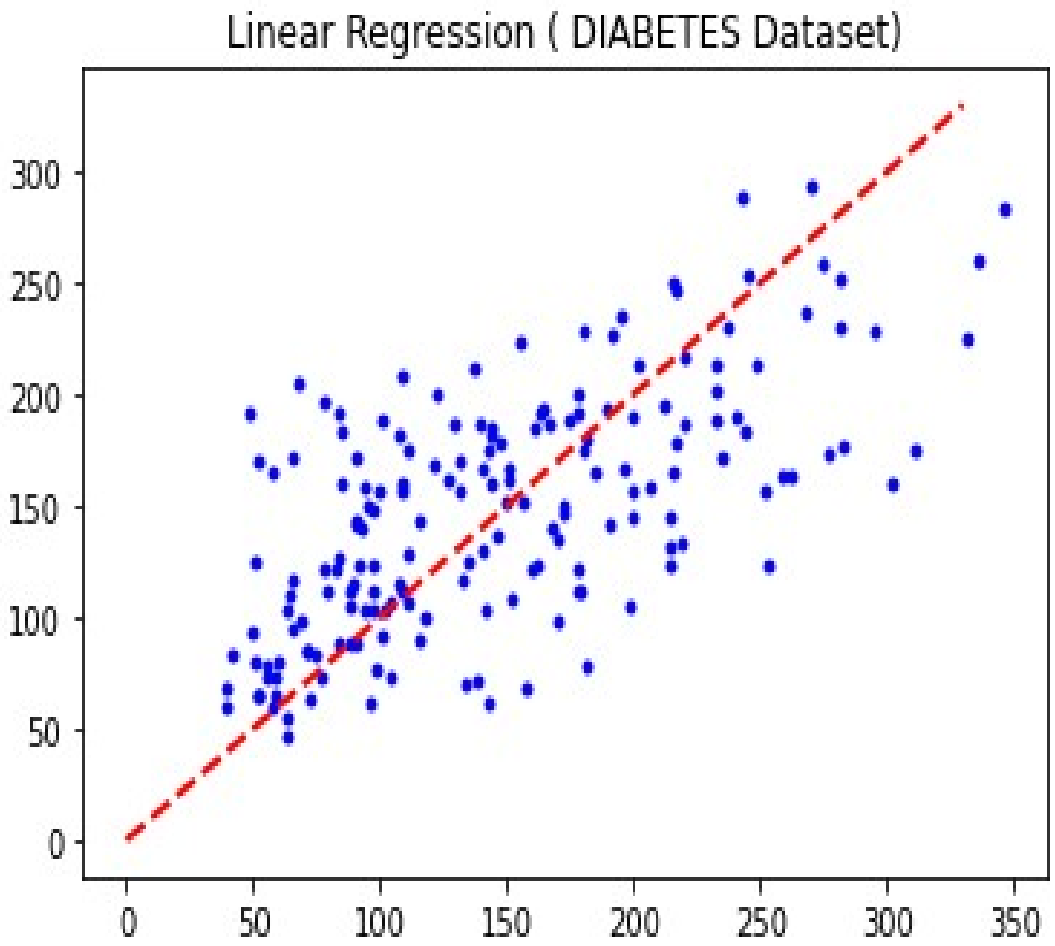
```
Coefficients are:
[ -59.73663337 -215.62170919  599.92621335  291.96724002 -829.65206295
  544.63994617  164.85191153  224.2392528   768.94426062   70.84982207]

Intercept :  152.89009028286725
Variance score:  0.4160439011127657
```

```
# Mean Squared Error
print("Mean squared error: %.2f\n"
      % mean_squared_error(y_test, predicted))
# Original data of X_test
expected = y_test
```

```
Mean squared error: 2962.93
```

Plot a graph for expected and predicted values

```
plt.title('Linear Regression ( DIABETES Dataset)')
plt.scatter(expected,predicted,c='b',marker='.',s=36)
plt.plot(np.linspace(0, 330, 100),np.linspace(0, 330, 100), '--r', linewidth=2)
plt.show()
```



Practical 2

AIM :- Implement Logistic Regression (Iris Dataset)

THEORY: -

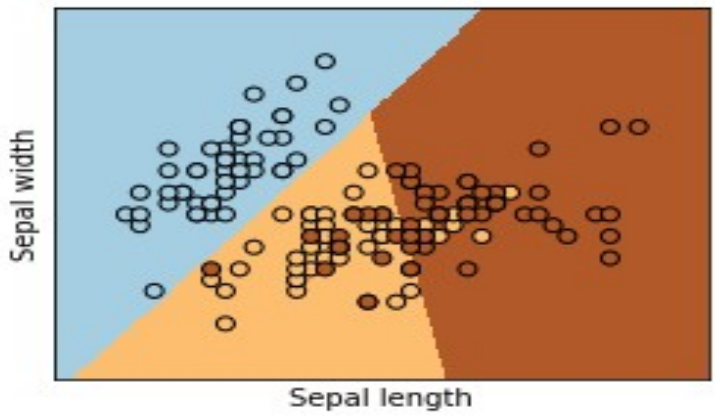
Logistic regression is a model that uses a logistic function to model a dependent variable. Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent variable and one or more nominal, ordinal, interval or ratio-level independent variables. It classifies an iris species as either (virginica, setosa, or versicolor) based off of the pedal length, pedal height, sepal length, and sepal height using a machine learning algorithm called Logistic Regression.

CODE/OUTPUT: -

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression

from sklearn import datasets


# import some data to play with

iris = datasets.load_iris()

X = iris.data[:, :2] # we only take the first two features.

Y = iris.target


# Create an instance of Logistic Regression Classifier and fit the data.

logreg = LogisticRegression(C=1e5)

logreg.fit(X, Y)
```

```
Out[3]: LogisticRegression(C=100000.0)
```

```
# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
h = 0.02 # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors="k", cmap=plt.cm.Paired)
plt.xlabel("Sepal length")
plt.ylabel("Sepal width")
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())
plt.show()
```



Practical 3

AIM :- Implement Multinomial Logistic Regression (Iris Dataset)

THEORY :-

Multinomial logistic regression is a classification method that generalizes logistic regression to multiclass problems, i.e. with more than two possible discrete outcomes. That is, it is a model that is used to predict the probabilities of the different possible outcomes of a categorically distributed dependent variable, given a set of independent variables (which may be real-valued, binary-valued, categorical-valued, etc.).

CODE/OUTPUT: -

```
#Loading the libraries and the data
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
import matplotlib as mpl
import matplotlib.pyplot as plt
import statsmodels.api as sm

#for readable figures
pd.set_option('float_format', '{:f}'.format)
iris = pd.read_csv("./Iris_Data.csv")
iris.head()
```

Out[1]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.100000	3.500000	1.400000	0.200000	Iris-setosa
1	4.900000	3.000000	1.400000	0.200000	Iris-setosa
2	4.700000	3.200000	1.300000	0.200000	Iris-setosa
3	4.600000	3.100000	1.500000	0.200000	Iris-setosa
4	5.000000	3.600000	1.400000	0.200000	Iris-setosa

```
x = iris.drop('species', axis=1)
y = iris['species']
trainX, testX, trainY, testY = train_test_split(x, y, test_size = 0.2)

#Fit the model
log_reg = LogisticRegression(solver='newton-cg', multi_class='multinomial')
log_reg.fit(trainX, trainY)
y_pred = log_reg.predict(testX)

# Model validation
# print the accuracy and error rate:
print('Accuracy: {:.2f}'.format(accuracy_score(testY, y_pred)))
print('Error rate: {:.2f}'.format(1 - accuracy_score(testY, y_pred)))
```

```
Accuracy: 1.00
Error rate: 0.00
```

```
# look at the scores from cross validation:
clf = LogisticRegression(solver='newton-cg', multi_class='multinomial')
scores = cross_val_score(clf, trainX, trainY, cv=5)
scores
```

Out[5]: array([0.91666667, 1. , 0.95833333, 0.91666667, 0.875])

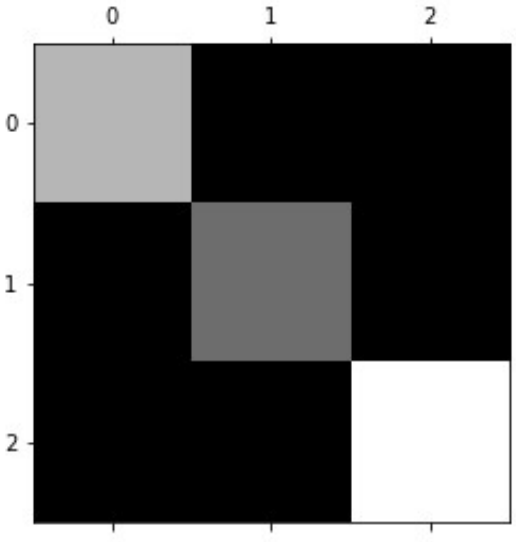
```
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.93 (+/- 0.08)
```

```
#Look at the confusion matrix:
confusion_matrix = confusion_matrix(testY, y_pred)
print(confusion_matrix)
```

```
[[10  0  0]
 [ 0  6  0]
 [ 0  0 14]]
```

```
# If you have many variables, it makes sense to plot the confusion matrix:
plt.matshow(confusion_matrix, cmap=plt.cm.gray)
plt.show()
```



#Calculated probabilities

#get the probabilities of the predicted classes

probability = log_reg.predict_proba(testX)

probability

```
Out[9]: array([[9.77177267e-01, 2.28226180e-02, 1.15291342e-07],
               [3.15508406e-04, 3.08742123e-01, 6.90942369e-01],
               [4.05833517e-06, 2.02355204e-02, 9.79760421e-01],
               [9.65833217e-01, 3.41664723e-02, 3.11152866e-07],
               [7.62413496e-03, 8.48073124e-01, 1.44302741e-01],
               [6.28488743e-05, 1.26754242e-01, 8.73182909e-01],
               [1.45813299e-06, 2.85751813e-02, 9.71423361e-01],
               [9.85612953e-01, 1.43870021e-02, 4.50510430e-08],
               [9.58087024e-05, 2.37604349e-01, 7.62299843e-01],
               [1.48077079e-07, 9.67116139e-03, 9.90328691e-01],
               [7.36532150e-04, 4.38827401e-01, 5.60436066e-01],
               [9.83925007e-01, 1.60749399e-02, 5.31322483e-08],
               [6.16083666e-04, 2.60226818e-01, 7.39157098e-01],
               [4.96184418e-06, 4.20074265e-02, 9.57987612e-01],
               [8.94595259e-08, 1.03011426e-02, 9.89698768e-01],
               [2.85615469e-02, 9.14889387e-01, 5.65490659e-02],
               [1.96617408e-05, 2.40582944e-02, 9.75922044e-01],
               [9.86409941e-01, 1.35900230e-02, 3.60773562e-08],
               [9.38044534e-01, 6.19551257e-02, 3.40320864e-07],
               [9.86999674e-01, 1.30002738e-02, 5.20347448e-08],
               [2.18962552e-05, 5.42286793e-02, 9.45749424e-01],
               [9.70345579e-01, 2.96543110e-02, 1.09581545e-07],
               [1.07061610e-02, 7.63582236e-01, 2.25711603e-01],
               [5.96119296e-03, 8.04958103e-01, 1.89080704e-01],
               [1.98090631e-04, 1.94208015e-01, 8.05593894e-01],
               [9.82592389e-01, 1.74075552e-02, 5.59265185e-08],
               [9.43324589e-01, 5.66750800e-02, 3.31517288e-07],
               [1.74083427e-02, 9.50815074e-01, 3.17765834e-02],
               [1.53811428e-02, 9.09184838e-01, 7.54340190e-02],
               [7.13182512e-06, 1.86770785e-02, 9.81315790e-01]])
```

#Each column here represents a class. The class with the highest probability is the output of the predicted class. Here we can see that the length of the probability data is the same as the length of the test data.

print(probability.shape[0])

print(testX.shape[0])

```
30
30
```

#Output into shape and a readable format

df = pd.DataFrame(log_reg.predict_proba(testX), columns=log_reg.classes_)

df.head()

#with the .classes_ function we get the order of the classes that Python gave.

Out[11]:

	Iris-setosa	Iris-versicolor	Iris-virginica
0	0.977177	0.022823	0.000000
1	0.000316	0.308742	0.690942
2	0.000004	0.020236	0.979760
3	0.965833	0.034166	0.000000
4	0.007624	0.848073	0.144303

#sum of the probabilities must always be 1

df['sum'] = df.sum(axis=1)

df.head()

Out[12]:

	Iris-setosa	Iris-versicolor	Iris-virginica	sum
0	0.977177	0.022823	0.000000	1.000000
1	0.000316	0.308742	0.690942	1.000000
2	0.000004	0.020236	0.979760	1.000000
3	0.965833	0.034166	0.000000	1.000000
4	0.007624	0.848073	0.144303	1.000000

add the predicted classes…

df['predicted_class'] = y_pred

df.head()

Out[13]:

	Iris-setosa	Iris-versicolor	Iris-virginica	sum	predicted_class
0	0.977177	0.022823	0.000000	1.000000	Iris-setosa
1	0.000316	0.308742	0.690942	1.000000	Iris-virginica
2	0.000004	0.020236	0.979760	1.000000	Iris-virginica
3	0.965833	0.034166	0.000000	1.000000	Iris-setosa
4	0.007624	0.848073	0.144303	1.000000	Iris-versicolor

#Actual classes:

df['actual_class'] = testY.to_frame().reset_index().drop(columns='index')

df.head()

Out[14]:

	Iris-setosa	Iris-versicolor	Iris-virginica	sum	predicted_class	actual_class
0	0.977177	0.022823	0.000000	1.000000	Iris-setosa	Iris-setosa
1	0.000316	0.308742	0.690942	1.000000	Iris-virginica	Iris-virginica
2	0.000004	0.020236	0.979760	1.000000	Iris-virginica	Iris-virginica
3	0.965833	0.034166	0.000000	1.000000	Iris-setosa	Iris-setosa
4	0.007624	0.848073	0.144303	1.000000	Iris-versicolor	Iris-versicolor

#Do a plausibility check whether the classes were predicted correctly.

le = preprocessing.LabelEncoder()

df['label_pred'] = le.fit_transform(df['predicted_class'])

df['label_actual'] = le.fit_transform(df['actual_class'])

df.head()

Out[15]:

	Iris-setosa	Iris-versicolor	Iris-virginica	sum	predicted_class	actual_class	label_pred	label_actual
0	0.977177	0.022823	0.000000	1.000000	Iris-setosa	Iris-setosa	0	0
1	0.000316	0.308742	0.690942	1.000000	Iris-virginica	Iris-virginica	2	2
2	0.000004	0.020236	0.979760	1.000000	Iris-virginica	Iris-virginica	2	2
3	0.965833	0.034166	0.000000	1.000000	Iris-setosa	Iris-setosa	0	0
4	0.007624	0.848073	0.144303	1.000000	Iris-versicolor	Iris-versicolor	1	1

#See that the two variables (predicted_class&actual_class) were coded the same and can therefore be continued properly.

targets = df['predicted_class']

integerEncoded = le.fit_transform(targets)

integerMapping=dict(zip(targets,integerEncoded))

integerMapping

Out[17]: {'Iris-setosa': 0, 'Iris-virginica': 2, 'Iris-versicolor': 1}

```
targets = df['actual_class']

integerEncoded = le.fit_transform(targets)

integerMapping=dict(zip(targets,integerEncoded))

integerMapping
```

```
Out[18]: {'Iris-setosa': 0, 'Iris-virginica': 2, 'Iris-versicolor': 1}
```

Plausibility check whether the classes were predicted correctly. If the result of subtraction is 0, it was a correct estimate of the model.

```
df['check'] = df['label_actual'] - df['label_pred']

df.head(7)
```

Out[20]:

	Iris-setosa	Iris-versicolor	Iris-virginica	sum	predicted_class	actual_class	label_pred	label_actual	check
0	0.977177	0.022823	0.000000	1.000000	Iris-setosa	Iris-setosa	0	0	0
1	0.000316	0.308742	0.690942	1.000000	Iris-virginica	Iris-virginica	2	2	0
2	0.000004	0.020236	0.979760	1.000000	Iris-virginica	Iris-virginica	2	2	0
3	0.965833	0.034166	0.000000	1.000000	Iris-setosa	Iris-setosa	0	0	0
4	0.007624	0.848073	0.144303	1.000000	Iris-versicolor	Iris-versicolor	1	1	0
5	0.000063	0.126754	0.873183	1.000000	Iris-virginica	Iris-virginica	2	2	0
6	0.000001	0.028575	0.971423	1.000000	Iris-virginica	Iris-virginica	2	2	0

#For better orientation, we give the observations descriptive names and delete unnecessary columns.

```
df['correct_prediction?'] = np.where(df['check'] == 0, 'True', 'False')

df = df.drop(['label_pred', 'label_actual', 'check'], axis=1)

df.head()
```

Out[21]:

	Iris-setosa	Iris-versicolor	Iris-virginica	sum	predicted_class	actual_class	correct_prediction?
0	0.977177	0.022823	0.000000	1.000000	Iris-setosa	Iris-setosa	True
1	0.000316	0.308742	0.690942	1.000000	Iris-virginica	Iris-virginica	True
2	0.000004	0.020236	0.979760	1.000000	Iris-virginica	Iris-virginica	True
3	0.965833	0.034166	0.000000	1.000000	Iris-setosa	Iris-setosa	True
4	0.007624	0.848073	0.144303	1.000000	Iris-versicolor	Iris-versicolor	True

```
#Use the generated “values” to manually calculate the accuracy again.
true_predictions = df[(df["correct_prediction?"] == 'True')].shape[0]
false_predictions = df[(df["correct_prediction?"] == 'False')].shape[0]
total = df["correct_prediction?"].shape[0]
print('manual calculated Accuracy is:', (true_predictions / total * 100))
```

```
manual calculated Accuracy is: 100.0
```

```
#take finally a look at the probabilities of the mispredicted classes
wrong_pred = df[(df["correct_prediction?"] == 'False')]
wrong_pred
```

Out[23]:

	Iris-setosa	Iris-versicolor	Iris-virginica	sum	predicted_class	actual_class	correct_prediction?

```
#Multinomial Logit with the statsmodel library
#To get the p-values of the model created above we have to use the statsmodel library again.
x = iris.drop('species', axis=1)
y = iris['species']
x = sm.add_constant(x, prepend = False)
mnlogit_mod = sm.MNLogit(y, x)
mnlogit_fit = mnlogit_mod.fit()
print (mnlogit_fit.summary())
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.039662
         Iterations: 35
                          MNLogit Regression Results
==============================================================================
Dep. Variable:                species   No. Observations:                  150
Model:                        MNLogit   Df Residuals:                      140
Method:                           MLE   Df Model:                            8
Date:                Fri, 13 May 2022   Pseudo R-squ.:                  0.9639
Time:                        20:17:46   Log-Likelihood:                -5.9493
converged:                      False   LL-Null:                       -164.79
Covariance Type:            nonrobust   LLR p-value:                7.055e-64
================================================================================================
species=Iris-versicolor      coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------------
sepal_length             -0.4880   2.54e+04  -1.92e-05      1.000   -4.98e+04    4.98e+04
sepal_width             -13.9801   4.06e+04     -0.000      1.000   -7.96e+04    7.96e+04
petal_length             14.4645   1.61e+04      0.001      0.999   -3.16e+04    3.16e+04
petal_width              11.0469   2.62e+04      0.000      1.000   -5.14e+04    5.14e+04
const                    -4.3766   6.94e+04  -6.31e-05      1.000   -1.36e+05    1.36e+05
------------------------------------------------------------------------------------------------
species=Iris-virginica       coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------------
sepal_length             -2.9532   2.54e+04     -0.000      1.000   -4.98e+04    4.98e+04
sepal_width             -20.6610   4.06e+04     -0.001      1.000   -7.96e+04    7.96e+04
petal_length             23.8939   1.61e+04      0.001      0.999   -3.16e+04    3.16e+04
petal_width              29.3330   2.62e+04      0.001      0.999   -5.14e+04    5.14e+04
const                   -47.0144   6.94e+04     -0.001      0.999   -1.36e+05    1.36e+05
================================================================================================
```

```
c:\users\harsh\appdata\local\programs\python\python38\lib\site-packages\statsmodels\base\model.py:604: ConvergenceWarning: Maxi
mum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

Practical 4

AIM :- Implement SVM Classifier (Iris Dataset)

THEORY :-

Support Vector Machine(SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well its best suited for classification. The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

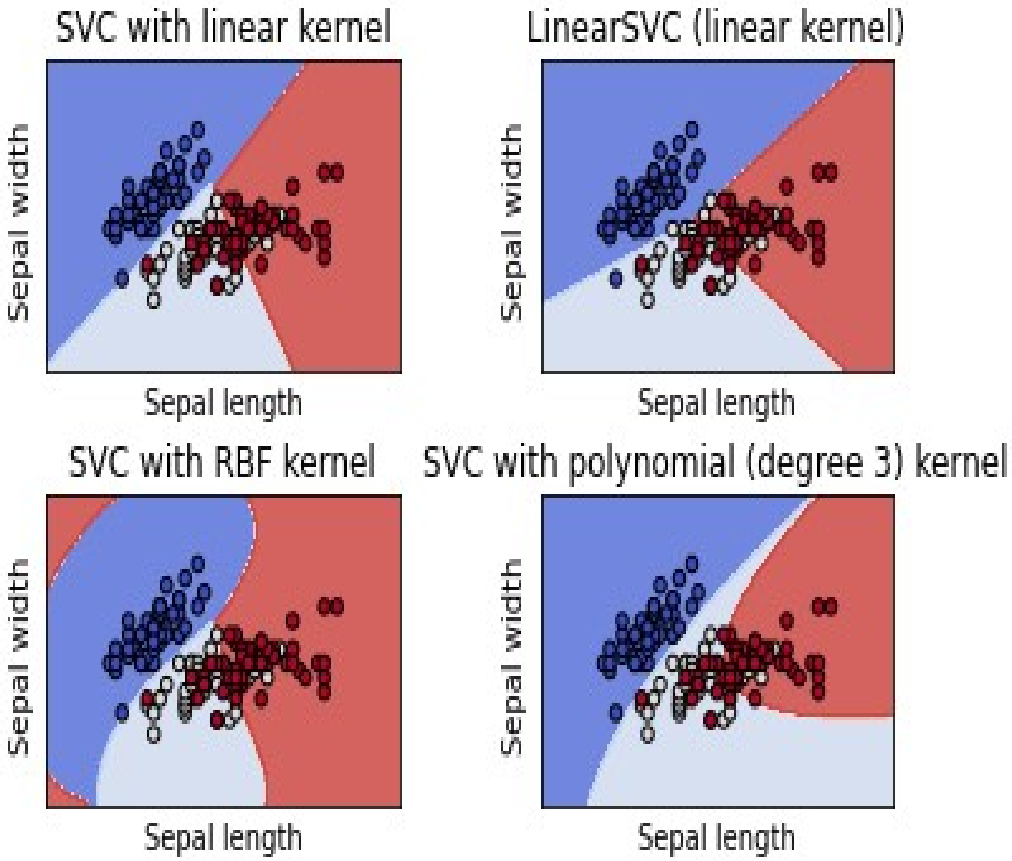
CODE/OUTPUT: -

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets

def make_meshgrid(x, y, h=0.02):
    """Create a mesh of points to plot in

    Parameters
    ----------
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    h: stepsize for meshgrid, optional

    Returns
    -------
    xx, yy :ndarray
    """
x_min, x_max = x.min() - 1, x.max() + 1
```

```
y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy


def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.



    Parameters
    ----------
    ax: matplotlib axes object
clf: a classifier
    xx: meshgridndarray
yy: meshgridndarray
    params: dictionary of params to pass to contourf, optional
    """


    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out



# import some data to play with
iris = datasets.load_iris()
# Take the first two features. We could avoid this by using a two-dim dataset
X = iris.data[:, :2]
y = iris.target
```

```
# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0 # SVM regularization parameter
models = (
svm.SVC(kernel="linear", C=C),
svm.LinearSVC(C=C, max_iter=10000),
svm.SVC(kernel="rbf", gamma=0.7, C=C),
svm.SVC(kernel="poly", degree=3, gamma="auto", C=C),
)
models = (clf.fit(X, y) for clf in models)


# title for the plots
titles = (
    "SVC with linear kernel",
    "LinearSVC (linear kernel)",
    "SVC with RBF kernel",
    "SVC with polynomial (degree 3) kernel",
)

# Set-up 2x2 grid for plotting.
fig, sub = plt.subplots(2, 2)
plt.subplots_adjust(wspace=0.4, hspace=0.4)

X0, X1 = X[:, 0], X[:, 1]
xx, yy = make_meshgrid(X0, X1)
```

```
for clf, title, ax in zip(models, titles, sub.flatten()):
plot_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
ax.scatter(X0, X1, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())
ax.set_xlabel("Sepal length")
ax.set_ylabel("Sepal width")
ax.set_xticks(())
ax.set_yticks(())
ax.set_title(title)
plt.show()
```



Practical 5

AIM :- Train and fine-tune a Decision Tree for the Moons Dataset

THEORY : -

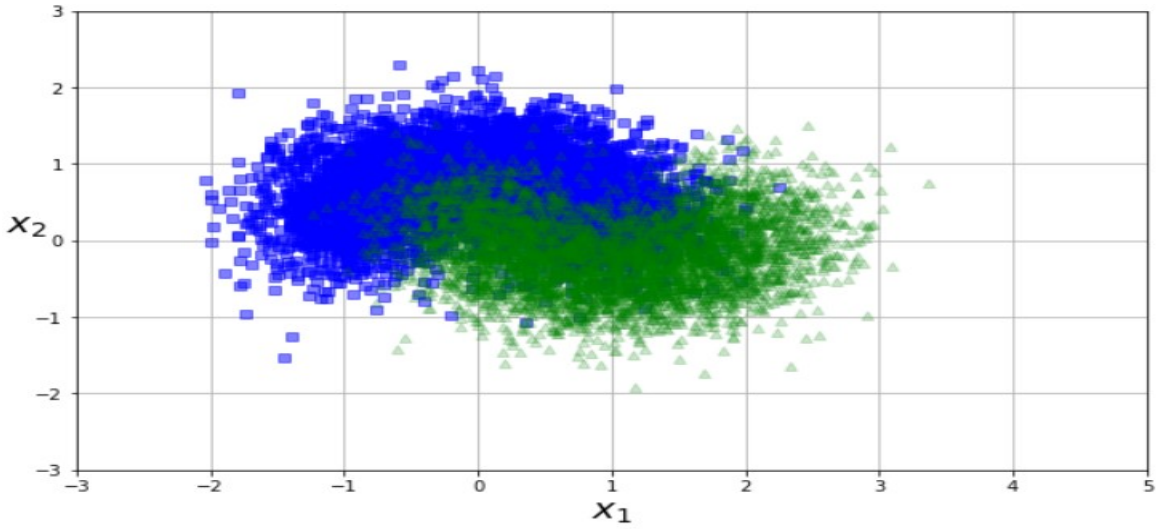
A Decision tree is a flowchart-like structure in which each internal node represents a test on a feature (e.g. whether a coin flip comes up heads or tails) , each leaf node represents a class label (decision taken after computing all features) and branches represent conjunctions of features that lead to those class

Libraries :numpy is used for scientific computing with Python. It is one of the fundamental packages you will use. Matplotlib library is used in Python for plotting graphs. The datasets package is the place from where you will import the make moons dataset. Sklearn library is used fo scientific computing. It has many features related to classification, regression and clustering algorithms including support vector machines.

CODE/OUTPUT: -

```
# GridSearchCV to fine-tune a Decision Tree Classifier

import numpy as np

import matplotlib.pyplot as plt



#Visualization

# This function will help in visualization of our dataset.

def plot_dataset(X, y, axes):

plt.figure(figsize=(10,6))

plt.plot(X[:, 0][y==0], X[:, 1][y==0], "bs",alpha = 0.5)

plt.plot(X[:, 0][y==1], X[:, 1][y==1], "g^",alpha = 0.2)

plt.axis(axes)

plt.grid(True, which='both')

plt.xlabel(r"$x_1$", fontsize=20)

plt.ylabel(r"$x_2$", fontsize=20, rotation=0)
```

```
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=10000, noise=0.4, random_state=21)
plot_dataset(X, y, [-3, 5, -3, 3])
```




```
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)

from sklearn.tree import DecisionTreeClassifier
tree_clf = DecisionTreeClassifier()

from sklearn.model_selection import GridSearchCV
parameter = {
        'criterion' : ["gini", "entropy"],
        'max_leaf_nodes': list(range(2, 50)),
        'min_samples_split': [2, 3, 4]
       }

clf = GridSearchCV(tree_clf, parameter, cv = 5,scoring ="accuracy",return_train_score=True,n_jobs=-1)

clf.fit(X_train, y_train)
```

```
Out[9]: GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
                     param_grid={'criterion': ['gini', 'entropy'],
                                 'max_leaf_nodes': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                                    13, 14, 15, 16, 17, 18, 19, 20, 21,
                                                    22, 23, 24, 25, 26, 27, 28, 29, 30,
                                                    31, ...],
                                 'min_samples_split': [2, 3, 4]},
                     return_train_score=True, scoring='accuracy')
```

Getting the best parameter:

clf.best_params_

```
Out[10]: {'criterion': 'entropy', 'max_leaf_nodes': 29, 'min_samples_split': 2}
```

#look at the training results:

cvres = clf.cv_results_

for mean_score, params in zip(cvres["mean_train_score"], cvres["params"]):

print(mean_score, params)

```
0.774 {'criterion': 'gini', 'max_leaf_nodes': 2, 'min_samples_split': 2}
0.774 {'criterion': 'gini', 'max_leaf_nodes': 2, 'min_samples_split': 3}
0.774 {'criterion': 'gini', 'max_leaf_nodes': 2, 'min_samples_split': 4}
0.821875 {'criterion': 'gini', 'max_leaf_nodes': 3, 'min_samples_split': 2}
0.821875 {'criterion': 'gini', 'max_leaf_nodes': 3, 'min_samples_split': 3}
0.821875 {'criterion': 'gini', 'max_leaf_nodes': 3, 'min_samples_split': 4}
0.8566874999999999 {'criterion': 'gini', 'max_leaf_nodes': 4, 'min_samples_split': 2}
0.8566874999999999 {'criterion': 'gini', 'max_leaf_nodes': 4, 'min_samples_split': 3}
0.8566874999999999 {'criterion': 'gini', 'max_leaf_nodes': 4, 'min_samples_split': 4}
0.8566874999999999 {'criterion': 'gini', 'max_leaf_nodes': 5, 'min_samples_split': 2}
0.8566874999999999 {'criterion': 'gini', 'max_leaf_nodes': 5, 'min_samples_split': 3}
0.8566874999999999 {'criterion': 'gini', 'max_leaf_nodes': 5, 'min_samples_split': 4}
0.8566874999999999 {'criterion': 'gini', 'max_leaf_nodes': 6, 'min_samples_split': 2}
0.8566874999999999 {'criterion': 'gini', 'max_leaf_nodes': 6, 'min_samples_split': 3}
0.8566874999999999 {'criterion': 'gini', 'max_leaf_nodes': 6, 'min_samples_split': 4}
0.8566874999999999 {'criterion': 'gini', 'max_leaf_nodes': 7, 'min_samples_split': 2}
0.8566874999999999 {'criterion': 'gini', 'max_leaf_nodes': 7, 'min_samples_split': 3}
0.8566874999999999 {'criterion': 'gini', 'max_leaf_nodes': 7, 'min_samples_split': 4}
0.85765625 {'criterion': 'gini', 'max_leaf_nodes': 8, 'min_samples_split': 2}
0.85765625 {'criterion': 'gini', 'max_leaf_nodes': 8, 'min_samples_split': 3}
0.85765625 {'criterion': 'gini', 'max_leaf_nodes': 8, 'min_samples_split': 4}
0.85778125 {'criterion': 'gini', 'max_leaf_nodes': 9, 'min_samples_split': 2}
0.85778125 {'criterion': 'gini', 'max_leaf_nodes': 9, 'min_samples_split': 3}
0.85778125 {'criterion': 'gini', 'max_leaf_nodes': 9, 'min_samples_split': 4}
0.8592187499999999 {'criterion': 'gini', 'max_leaf_nodes': 10, 'min_samples_split': 2}
0.8592187499999999 {'criterion': 'gini', 'max_leaf_nodes': 10, 'min_samples_split': 3}
0.8592187499999999 {'criterion': 'gini', 'max_leaf_nodes': 10, 'min_samples_split': 4}
0.86071875 {'criterion': 'gini', 'max_leaf_nodes': 11, 'min_samples_split': 2}
0.86071875 {'criterion': 'gini', 'max_leaf_nodes': 11, 'min_samples_split': 3}
0.86071875 {'criterion': 'gini', 'max_leaf_nodes': 11, 'min_samples_split': 4}
0.8615625 {'criterion': 'gini', 'max_leaf_nodes': 12, 'min_samples_split': 2}
0.8615625 {'criterion': 'gini', 'max_leaf_nodes': 12, 'min_samples_split': 3}
0.8615625 {'criterion': 'gini', 'max_leaf_nodes': 12, 'min_samples_split': 4}
0.8615625 {'criterion': 'gini', 'max_leaf_nodes': 13, 'min_samples_split': 2}
0.8615625 {'criterion': 'gini', 'max_leaf_nodes': 13, 'min_samples_split': 3}
0.8615625 {'criterion': 'gini', 'max_leaf_nodes': 13, 'min_samples_split': 4}
0.8615625 {'criterion': 'gini', 'max_leaf_nodes': 14, 'min_samples_split': 2}
```

```
0.8615625 {'criterion': 'gini', 'max_leaf_nodes': 14, 'min_samples_split': 3}
0.8615625 {'criterion': 'gini', 'max_leaf_nodes': 14, 'min_samples_split': 4}
0.8620625000000001 {'criterion': 'gini', 'max_leaf_nodes': 15, 'min_samples_split': 2}
0.8620625000000001 {'criterion': 'gini', 'max_leaf_nodes': 15, 'min_samples_split': 3}
0.8620625000000001 {'criterion': 'gini', 'max_leaf_nodes': 15, 'min_samples_split': 4}
0.8620625000000001 {'criterion': 'gini', 'max_leaf_nodes': 16, 'min_samples_split': 2}
0.8620625000000001 {'criterion': 'gini', 'max_leaf_nodes': 16, 'min_samples_split': 3}
0.8620625000000001 {'criterion': 'gini', 'max_leaf_nodes': 16, 'min_samples_split': 4}
0.86303125 {'criterion': 'gini', 'max_leaf_nodes': 17, 'min_samples_split': 2}
0.86303125 {'criterion': 'gini', 'max_leaf_nodes': 17, 'min_samples_split': 3}
0.86303125 {'criterion': 'gini', 'max_leaf_nodes': 17, 'min_samples_split': 4}
0.86396875 {'criterion': 'gini', 'max_leaf_nodes': 18, 'min_samples_split': 2}
0.86396875 {'criterion': 'gini', 'max_leaf_nodes': 18, 'min_samples_split': 3}
0.86396875 {'criterion': 'gini', 'max_leaf_nodes': 18, 'min_samples_split': 4}
0.86509375 {'criterion': 'gini', 'max_leaf_nodes': 19, 'min_samples_split': 2}
0.86509375 {'criterion': 'gini', 'max_leaf_nodes': 19, 'min_samples_split': 3}
0.86509375 {'criterion': 'gini', 'max_leaf_nodes': 19, 'min_samples_split': 4}
0.8660937499999999 {'criterion': 'gini', 'max_leaf_nodes': 20, 'min_samples_split': 2}
0.8660937499999999 {'criterion': 'gini', 'max_leaf_nodes': 20, 'min_samples_split': 3}

0.8660937499999999 {'criterion': 'gini', 'max_leaf_nodes': 20, 'min_samples_split': 4}
0.8671875 {'criterion': 'gini', 'max_leaf_nodes': 21, 'min_samples_split': 2}
0.8671875 {'criterion': 'gini', 'max_leaf_nodes': 21, 'min_samples_split': 3}
0.8671875 {'criterion': 'gini', 'max_leaf_nodes': 21, 'min_samples_split': 4}
0.8680625000000001 {'criterion': 'gini', 'max_leaf_nodes': 22, 'min_samples_split': 2}
0.8680625000000001 {'criterion': 'gini', 'max_leaf_nodes': 22, 'min_samples_split': 3}
0.8680625000000001 {'criterion': 'gini', 'max_leaf_nodes': 22, 'min_samples_split': 4}
0.86890625 {'criterion': 'gini', 'max_leaf_nodes': 23, 'min_samples_split': 2}
0.86890625 {'criterion': 'gini', 'max_leaf_nodes': 23, 'min_samples_split': 3}
0.86890625 {'criterion': 'gini', 'max_leaf_nodes': 23, 'min_samples_split': 4}
0.8699999999999999 {'criterion': 'gini', 'max_leaf_nodes': 24, 'min_samples_split': 2}
0.8699999999999999 {'criterion': 'gini', 'max_leaf_nodes': 24, 'min_samples_split': 3}
0.8699999999999999 {'criterion': 'gini', 'max_leaf_nodes': 24, 'min_samples_split': 4}
0.8714062499999999 {'criterion': 'gini', 'max_leaf_nodes': 25, 'min_samples_split': 2}
0.8714062499999999 {'criterion': 'gini', 'max_leaf_nodes': 25, 'min_samples_split': 3}
0.8714062499999999 {'criterion': 'gini', 'max_leaf_nodes': 25, 'min_samples_split': 4}
0.87215625 {'criterion': 'gini', 'max_leaf_nodes': 26, 'min_samples_split': 2}

0.87215625 {'criterion': 'gini', 'max_leaf_nodes': 26, 'min_samples_split': 3}
0.87215625 {'criterion': 'gini', 'max_leaf_nodes': 26, 'min_samples_split': 4}
0.87290625 {'criterion': 'gini', 'max_leaf_nodes': 27, 'min_samples_split': 2}
0.87290625 {'criterion': 'gini', 'max_leaf_nodes': 27, 'min_samples_split': 3}
0.87290625 {'criterion': 'gini', 'max_leaf_nodes': 27, 'min_samples_split': 4}
0.8733124999999999 {'criterion': 'gini', 'max_leaf_nodes': 28, 'min_samples_split': 2}
0.8733124999999999 {'criterion': 'gini', 'max_leaf_nodes': 28, 'min_samples_split': 3}
0.8733124999999999 {'criterion': 'gini', 'max_leaf_nodes': 28, 'min_samples_split': 4}
0.87384375 {'criterion': 'gini', 'max_leaf_nodes': 29, 'min_samples_split': 2}
0.87384375 {'criterion': 'gini', 'max_leaf_nodes': 29, 'min_samples_split': 3}
0.87384375 {'criterion': 'gini', 'max_leaf_nodes': 29, 'min_samples_split': 4}
0.87425 {'criterion': 'gini', 'max_leaf_nodes': 30, 'min_samples_split': 2}
0.87425 {'criterion': 'gini', 'max_leaf_nodes': 30, 'min_samples_split': 3}
0.87425 {'criterion': 'gini', 'max_leaf_nodes': 30, 'min_samples_split': 4}
0.8745312500000001 {'criterion': 'gini', 'max_leaf_nodes': 31, 'min_samples_split': 2}
0.8745312500000001 {'criterion': 'gini', 'max_leaf_nodes': 31, 'min_samples_split': 3}
0.8745312500000001 {'criterion': 'gini', 'max_leaf_nodes': 31, 'min_samples_split': 4}

0.87475 {'criterion': 'gini', 'max_leaf_nodes': 32, 'min_samples_split': 2}
0.87475 {'criterion': 'gini', 'max_leaf_nodes': 32, 'min_samples_split': 3}
0.87475 {'criterion': 'gini', 'max_leaf_nodes': 32, 'min_samples_split': 4}
0.875 {'criterion': 'gini', 'max_leaf_nodes': 33, 'min_samples_split': 2}
0.875 {'criterion': 'gini', 'max_leaf_nodes': 33, 'min_samples_split': 3}
0.875 {'criterion': 'gini', 'max_leaf_nodes': 33, 'min_samples_split': 4}
0.8754062499999999 {'criterion': 'gini', 'max_leaf_nodes': 34, 'min_samples_split': 2}
0.8754062499999999 {'criterion': 'gini', 'max_leaf_nodes': 34, 'min_samples_split': 3}
0.8754062499999999 {'criterion': 'gini', 'max_leaf_nodes': 34, 'min_samples_split': 4}
0.8756562499999999 {'criterion': 'gini', 'max_leaf_nodes': 35, 'min_samples_split': 2}
0.8756562499999999 {'criterion': 'gini', 'max_leaf_nodes': 35, 'min_samples_split': 3}
0.8756562499999999 {'criterion': 'gini', 'max_leaf_nodes': 35, 'min_samples_split': 4}
0.8763125 {'criterion': 'gini', 'max_leaf_nodes': 36, 'min_samples_split': 2}
0.8763125 {'criterion': 'gini', 'max_leaf_nodes': 36, 'min_samples_split': 3}
0.8763125 {'criterion': 'gini', 'max_leaf_nodes': 36, 'min_samples_split': 4}
0.87675 {'criterion': 'gini', 'max_leaf_nodes': 37, 'min_samples_split': 2}
0.87675 {'criterion': 'gini', 'max_leaf_nodes': 37, 'min_samples_split': 3}
0.87675 {'criterion': 'gini', 'max_leaf_nodes': 37, 'min_samples_split': 4}
```

```
0.8774999999999998 {'criterion': 'gini', 'max_leaf_nodes': 38, 'min_samples_split': 2}
0.8774999999999998 {'criterion': 'gini', 'max_leaf_nodes': 38, 'min_samples_split': 3}
0.8774999999999998 {'criterion': 'gini', 'max_leaf_nodes': 38, 'min_samples_split': 4}
0.8778124999999999 {'criterion': 'gini', 'max_leaf_nodes': 39, 'min_samples_split': 2}
0.8778124999999999 {'criterion': 'gini', 'max_leaf_nodes': 39, 'min_samples_split': 3}
0.8778124999999999 {'criterion': 'gini', 'max_leaf_nodes': 39, 'min_samples_split': 4}
0.8778437499999999 {'criterion': 'gini', 'max_leaf_nodes': 40, 'min_samples_split': 2}
0.8778437499999999 {'criterion': 'gini', 'max_leaf_nodes': 40, 'min_samples_split': 3}
0.8778437499999999 {'criterion': 'gini', 'max_leaf_nodes': 40, 'min_samples_split': 4}
0.8781874999999999 {'criterion': 'gini', 'max_leaf_nodes': 41, 'min_samples_split': 2}
0.8781874999999999 {'criterion': 'gini', 'max_leaf_nodes': 41, 'min_samples_split': 3}
0.8781874999999999 {'criterion': 'gini', 'max_leaf_nodes': 41, 'min_samples_split': 4}
0.8784375000000001 {'criterion': 'gini', 'max_leaf_nodes': 42, 'min_samples_split': 2}
0.8784375000000001 {'criterion': 'gini', 'max_leaf_nodes': 42, 'min_samples_split': 3}
0.8784375000000001 {'criterion': 'gini', 'max_leaf_nodes': 42, 'min_samples_split': 4}
0.8786250000000001 {'criterion': 'gini', 'max_leaf_nodes': 43, 'min_samples_split': 2}
0.8786250000000001 {'criterion': 'gini', 'max_leaf_nodes': 43, 'min_samples_split': 3}
0.8786250000000001 {'criterion': 'gini', 'max_leaf_nodes': 43, 'min_samples_split': 4}

0.87890625 {'criterion': 'gini', 'max_leaf_nodes': 44, 'min_samples_split': 2}
0.87890625 {'criterion': 'gini', 'max_leaf_nodes': 44, 'min_samples_split': 3}
0.87890625 {'criterion': 'gini', 'max_leaf_nodes': 44, 'min_samples_split': 4}
0.8789999999999999 {'criterion': 'gini', 'max_leaf_nodes': 45, 'min_samples_split': 2}
0.8789999999999999 {'criterion': 'gini', 'max_leaf_nodes': 45, 'min_samples_split': 3}
0.8789999999999999 {'criterion': 'gini', 'max_leaf_nodes': 45, 'min_samples_split': 4}
0.8793124999999999 {'criterion': 'gini', 'max_leaf_nodes': 46, 'min_samples_split': 2}
0.8793124999999999 {'criterion': 'gini', 'max_leaf_nodes': 46, 'min_samples_split': 3}
0.8793124999999999 {'criterion': 'gini', 'max_leaf_nodes': 46, 'min_samples_split': 4}
0.8797499999999999 {'criterion': 'gini', 'max_leaf_nodes': 47, 'min_samples_split': 2}
0.8797499999999999 {'criterion': 'gini', 'max_leaf_nodes': 47, 'min_samples_split': 3}
0.8797499999999999 {'criterion': 'gini', 'max_leaf_nodes': 47, 'min_samples_split': 4}
0.87990625 {'criterion': 'gini', 'max_leaf_nodes': 48, 'min_samples_split': 2}
0.87990625 {'criterion': 'gini', 'max_leaf_nodes': 48, 'min_samples_split': 3}
0.87990625 {'criterion': 'gini', 'max_leaf_nodes': 48, 'min_samples_split': 4}
0.88009375 {'criterion': 'gini', 'max_leaf_nodes': 49, 'min_samples_split': 2}
0.88009375 {'criterion': 'gini', 'max_leaf_nodes': 49, 'min_samples_split': 3}

0.88009375 {'criterion': 'gini', 'max_leaf_nodes': 49, 'min_samples_split': 4}
0.7714375 {'criterion': 'entropy', 'max_leaf_nodes': 2, 'min_samples_split': 2}
0.7714375 {'criterion': 'entropy', 'max_leaf_nodes': 2, 'min_samples_split': 3}
0.7714375 {'criterion': 'entropy', 'max_leaf_nodes': 2, 'min_samples_split': 4}
0.8254999999999999 {'criterion': 'entropy', 'max_leaf_nodes': 3, 'min_samples_split': 2}
0.8254999999999999 {'criterion': 'entropy', 'max_leaf_nodes': 3, 'min_samples_split': 3}
0.8254999999999999 {'criterion': 'entropy', 'max_leaf_nodes': 3, 'min_samples_split': 4}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 4, 'min_samples_split': 2}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 4, 'min_samples_split': 3}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 4, 'min_samples_split': 4}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 5, 'min_samples_split': 2}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 5, 'min_samples_split': 3}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 5, 'min_samples_split': 4}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 6, 'min_samples_split': 2}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 6, 'min_samples_split': 3}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 6, 'min_samples_split': 4}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 7, 'min_samples_split': 2}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 7, 'min_samples_split': 3}

0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 7, 'min_samples_split': 4}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 8, 'min_samples_split': 2}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 8, 'min_samples_split': 3}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 8, 'min_samples_split': 4}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 9, 'min_samples_split': 2}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 9, 'min_samples_split': 3}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 9, 'min_samples_split': 4}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 10, 'min_samples_split': 2}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 10, 'min_samples_split': 3}
0.8539375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 10, 'min_samples_split': 4}
0.8545312500000002 {'criterion': 'entropy', 'max_leaf_nodes': 11, 'min_samples_split': 2}
0.8545312500000002 {'criterion': 'entropy', 'max_leaf_nodes': 11, 'min_samples_split': 3}
0.8545312500000002 {'criterion': 'entropy', 'max_leaf_nodes': 11, 'min_samples_split': 4}
0.856625 {'criterion': 'entropy', 'max_leaf_nodes': 12, 'min_samples_split': 2}
0.856625 {'criterion': 'entropy', 'max_leaf_nodes': 12, 'min_samples_split': 3}
0.856625 {'criterion': 'entropy', 'max_leaf_nodes': 12, 'min_samples_split': 4}
0.8578437500000001 {'criterion': 'entropy', 'max_leaf_nodes': 13, 'min_samples_split': 2}
```

```
0.8578437500000001 {'criterion': 'entropy', 'max_leaf_nodes': 13, 'min_samples_split': 3}
0.8578437500000001 {'criterion': 'entropy', 'max_leaf_nodes': 13, 'min_samples_split': 4}
0.8611562500000002 {'criterion': 'entropy', 'max_leaf_nodes': 14, 'min_samples_split': 2}
0.8611562500000002 {'criterion': 'entropy', 'max_leaf_nodes': 14, 'min_samples_split': 3}
0.8611562500000002 {'criterion': 'entropy', 'max_leaf_nodes': 14, 'min_samples_split': 4}
0.86234375 {'criterion': 'entropy', 'max_leaf_nodes': 15, 'min_samples_split': 2}
0.86234375 {'criterion': 'entropy', 'max_leaf_nodes': 15, 'min_samples_split': 3}
0.86234375 {'criterion': 'entropy', 'max_leaf_nodes': 15, 'min_samples_split': 4}
0.86234375 {'criterion': 'entropy', 'max_leaf_nodes': 16, 'min_samples_split': 2}
0.86234375 {'criterion': 'entropy', 'max_leaf_nodes': 16, 'min_samples_split': 3}
0.86234375 {'criterion': 'entropy', 'max_leaf_nodes': 16, 'min_samples_split': 4}
0.8647500000000001 {'criterion': 'entropy', 'max_leaf_nodes': 17, 'min_samples_split': 2}
0.8647500000000001 {'criterion': 'entropy', 'max_leaf_nodes': 17, 'min_samples_split': 3}
0.8647500000000001 {'criterion': 'entropy', 'max_leaf_nodes': 17, 'min_samples_split': 4}
0.8662812500000001 {'criterion': 'entropy', 'max_leaf_nodes': 18, 'min_samples_split': 2}
0.8662812500000001 {'criterion': 'entropy', 'max_leaf_nodes': 18, 'min_samples_split': 3}
0.8662812500000001 {'criterion': 'entropy', 'max_leaf_nodes': 18, 'min_samples_split': 4}
```

```
0.8665625 {'criterion': 'entropy', 'max_leaf_nodes': 19, 'min_samples_split': 2}
0.8665625 {'criterion': 'entropy', 'max_leaf_nodes': 19, 'min_samples_split': 3}
0.8665625 {'criterion': 'entropy', 'max_leaf_nodes': 19, 'min_samples_split': 4}
0.8665625 {'criterion': 'entropy', 'max_leaf_nodes': 20, 'min_samples_split': 2}
0.8665625 {'criterion': 'entropy', 'max_leaf_nodes': 20, 'min_samples_split': 3}
0.8665625 {'criterion': 'entropy', 'max_leaf_nodes': 20, 'min_samples_split': 4}
0.86675 {'criterion': 'entropy', 'max_leaf_nodes': 21, 'min_samples_split': 2}
0.86675 {'criterion': 'entropy', 'max_leaf_nodes': 21, 'min_samples_split': 3}
0.86675 {'criterion': 'entropy', 'max_leaf_nodes': 21, 'min_samples_split': 4}
0.8669062500000001 {'criterion': 'entropy', 'max_leaf_nodes': 22, 'min_samples_split': 2}
0.8669062500000001 {'criterion': 'entropy', 'max_leaf_nodes': 22, 'min_samples_split': 3}
0.8669062500000001 {'criterion': 'entropy', 'max_leaf_nodes': 22, 'min_samples_split': 4}
0.8674375 {'criterion': 'entropy', 'max_leaf_nodes': 23, 'min_samples_split': 2}
0.8674375 {'criterion': 'entropy', 'max_leaf_nodes': 23, 'min_samples_split': 3}
0.8674375 {'criterion': 'entropy', 'max_leaf_nodes': 23, 'min_samples_split': 4}
0.86815625 {'criterion': 'entropy', 'max_leaf_nodes': 24, 'min_samples_split': 2}
0.86815625 {'criterion': 'entropy', 'max_leaf_nodes': 24, 'min_samples_split': 3}
0.86815625 {'criterion': 'entropy', 'max_leaf_nodes': 24, 'min_samples_split': 4}
```

```
0.86815625 {'criterion': 'entropy', 'max_leaf_nodes': 25, 'min_samples_split': 2}
0.86815625 {'criterion': 'entropy', 'max_leaf_nodes': 25, 'min_samples_split': 3}
0.86815625 {'criterion': 'entropy', 'max_leaf_nodes': 25, 'min_samples_split': 4}
0.8682812500000001 {'criterion': 'entropy', 'max_leaf_nodes': 26, 'min_samples_split': 2}
0.8682812500000001 {'criterion': 'entropy', 'max_leaf_nodes': 26, 'min_samples_split': 3}
0.8682812500000001 {'criterion': 'entropy', 'max_leaf_nodes': 26, 'min_samples_split': 4}
0.8682812500000001 {'criterion': 'entropy', 'max_leaf_nodes': 27, 'min_samples_split': 2}
0.8682812500000001 {'criterion': 'entropy', 'max_leaf_nodes': 27, 'min_samples_split': 3}
0.8682812500000001 {'criterion': 'entropy', 'max_leaf_nodes': 27, 'min_samples_split': 4}
0.8684375 {'criterion': 'entropy', 'max_leaf_nodes': 28, 'min_samples_split': 2}
0.8684375 {'criterion': 'entropy', 'max_leaf_nodes': 28, 'min_samples_split': 3}
0.8684375 {'criterion': 'entropy', 'max_leaf_nodes': 28, 'min_samples_split': 4}
0.8689375 {'criterion': 'entropy', 'max_leaf_nodes': 29, 'min_samples_split': 2}
0.8689375 {'criterion': 'entropy', 'max_leaf_nodes': 29, 'min_samples_split': 3}
0.8689375 {'criterion': 'entropy', 'max_leaf_nodes': 29, 'min_samples_split': 4}
0.8689375 {'criterion': 'entropy', 'max_leaf_nodes': 30, 'min_samples_split': 2}
0.8689375 {'criterion': 'entropy', 'max_leaf_nodes': 30, 'min_samples_split': 3}
0.8689375 {'criterion': 'entropy', 'max_leaf_nodes': 30, 'min_samples_split': 4}
```

```
0.8689375 {'criterion': 'entropy', 'max_leaf_nodes': 31, 'min_samples_split': 2}
0.8689375 {'criterion': 'entropy', 'max_leaf_nodes': 31, 'min_samples_split': 3}
0.8689375 {'criterion': 'entropy', 'max_leaf_nodes': 31, 'min_samples_split': 4}
0.869125 {'criterion': 'entropy', 'max_leaf_nodes': 32, 'min_samples_split': 2}
0.869125 {'criterion': 'entropy', 'max_leaf_nodes': 32, 'min_samples_split': 3}
0.869125 {'criterion': 'entropy', 'max_leaf_nodes': 32, 'min_samples_split': 4}
0.8692500000000001 {'criterion': 'entropy', 'max_leaf_nodes': 33, 'min_samples_split': 2}
0.8692500000000001 {'criterion': 'entropy', 'max_leaf_nodes': 33, 'min_samples_split': 3}
0.8692500000000001 {'criterion': 'entropy', 'max_leaf_nodes': 33, 'min_samples_split': 4}
0.86934375 {'criterion': 'entropy', 'max_leaf_nodes': 34, 'min_samples_split': 2}
0.86934375 {'criterion': 'entropy', 'max_leaf_nodes': 34, 'min_samples_split': 3}
0.86934375 {'criterion': 'entropy', 'max_leaf_nodes': 34, 'min_samples_split': 4}
0.8694375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 35, 'min_samples_split': 2}
0.8694375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 35, 'min_samples_split': 3}
0.8694375000000001 {'criterion': 'entropy', 'max_leaf_nodes': 35, 'min_samples_split': 4}
0.8699375 {'criterion': 'entropy', 'max_leaf_nodes': 36, 'min_samples_split': 2}
0.8699375 {'criterion': 'entropy', 'max_leaf_nodes': 36, 'min_samples_split': 3}
0.8699375 {'criterion': 'entropy', 'max_leaf_nodes': 36, 'min_samples_split': 4}
0.8701874999999999 {'criterion': 'entropy', 'max_leaf_nodes': 37, 'min_samples_split': 2}
0.8701874999999999 {'criterion': 'entropy', 'max_leaf_nodes': 37, 'min_samples_split': 3}
0.8701874999999999 {'criterion': 'entropy', 'max_leaf_nodes': 37, 'min_samples_split': 4}
0.87034375 {'criterion': 'entropy', 'max_leaf_nodes': 38, 'min_samples_split': 2}
0.87034375 {'criterion': 'entropy', 'max_leaf_nodes': 38, 'min_samples_split': 3}
0.87034375 {'criterion': 'entropy', 'max_leaf_nodes': 38, 'min_samples_split': 4}
0.87053125 {'criterion': 'entropy', 'max_leaf_nodes': 39, 'min_samples_split': 2}
0.87053125 {'criterion': 'entropy', 'max_leaf_nodes': 39, 'min_samples_split': 3}
0.87053125 {'criterion': 'entropy', 'max_leaf_nodes': 39, 'min_samples_split': 4}
0.8706250000000001 {'criterion': 'entropy', 'max_leaf_nodes': 40, 'min_samples_split': 2}
0.8706250000000001 {'criterion': 'entropy', 'max_leaf_nodes': 40, 'min_samples_split': 3}
0.8706250000000001 {'criterion': 'entropy', 'max_leaf_nodes': 40, 'min_samples_split': 4}
0.87109375 {'criterion': 'entropy', 'max_leaf_nodes': 41, 'min_samples_split': 2}
0.87109375 {'criterion': 'entropy', 'max_leaf_nodes': 41, 'min_samples_split': 3}
0.87109375 {'criterion': 'entropy', 'max_leaf_nodes': 41, 'min_samples_split': 4}
0.87121875 {'criterion': 'entropy', 'max_leaf_nodes': 42, 'min_samples_split': 2}
```

```
0.87121875 {'criterion': 'entropy', 'max_leaf_nodes': 42, 'min_samples_split': 3}
0.87121875 {'criterion': 'entropy', 'max_leaf_nodes': 42, 'min_samples_split': 4}
0.8713437500000001 {'criterion': 'entropy', 'max_leaf_nodes': 43, 'min_samples_split': 2}
0.8713437500000001 {'criterion': 'entropy', 'max_leaf_nodes': 43, 'min_samples_split': 3}
0.8713437500000001 {'criterion': 'entropy', 'max_leaf_nodes': 43, 'min_samples_split': 4}
0.87165625 {'criterion': 'entropy', 'max_leaf_nodes': 44, 'min_samples_split': 2}
0.87165625 {'criterion': 'entropy', 'max_leaf_nodes': 44, 'min_samples_split': 3}
0.87165625 {'criterion': 'entropy', 'max_leaf_nodes': 44, 'min_samples_split': 4}
0.8717187500000001 {'criterion': 'entropy', 'max_leaf_nodes': 45, 'min_samples_split': 2}
0.8717187500000001 {'criterion': 'entropy', 'max_leaf_nodes': 45, 'min_samples_split': 3}
0.8717187500000001 {'criterion': 'entropy', 'max_leaf_nodes': 45, 'min_samples_split': 4}
0.8717500000000001 {'criterion': 'entropy', 'max_leaf_nodes': 46, 'min_samples_split': 2}
0.8717500000000001 {'criterion': 'entropy', 'max_leaf_nodes': 46, 'min_samples_split': 3}
0.8717500000000001 {'criterion': 'entropy', 'max_leaf_nodes': 46, 'min_samples_split': 4}
0.8719062500000001 {'criterion': 'entropy', 'max_leaf_nodes': 47, 'min_samples_split': 2}
0.8719062500000001 {'criterion': 'entropy', 'max_leaf_nodes': 47, 'min_samples_split': 3}
0.8719062500000001 {'criterion': 'entropy', 'max_leaf_nodes': 47, 'min_samples_split': 4}
0.87234375 {'criterion': 'entropy', 'max_leaf_nodes': 48, 'min_samples_split': 2}
0.87234375 {'criterion': 'entropy', 'max_leaf_nodes': 48, 'min_samples_split': 3}
0.87234375 {'criterion': 'entropy', 'max_leaf_nodes': 48, 'min_samples_split': 4}
0.87240625 {'criterion': 'entropy', 'max_leaf_nodes': 49, 'min_samples_split': 2}
0.87240625 {'criterion': 'entropy', 'max_leaf_nodes': 49, 'min_samples_split': 3}
0.87240625 {'criterion': 'entropy', 'max_leaf_nodes': 49, 'min_samples_split': 4}
```

#Getting the training score

clf.score(X_train, y_train)

```
Out[12]: 0.867
```

from sklearn.metrics import confusion_matrix

pred = clf.predict(X_train)

confusion_matrix(y_train,pred)

```
Out[13]: array([[3547,  469],
                [ 595, 3389]], dtype=int64)
```

#from the confusion matrix let's get our precision and recall, which are better metrics.

from sklearn.metrics import precision_score, recall_score

pre = precision_score(y_train, pred)

re = recall_score(y_train, pred)

print(f"Precision: {pre} Recall:{re}")

```
Precision: 0.8784344219803006 Recall:0.8506526104417671
```

#we have a higher precision than recall but lets combine the two metrics into F1 score.

from sklearn.metrics import f1_score

f1_score(y_train, pred)

```
Out[15]: 0.8643203264473348
```

#Getting the testing score

clf.score(X_test, y_test)

```
Out[16]: 0.8625
```

Practical 6

AIM :- Train an SVM regressor on the California Housing Dataset

THEORY : -

SVM regressor : Support Vector Regression is a supervised learning algorithm that is used to predict discrete values. Support Vector Regression uses the same principle as the SVMs. The basic idea behind SVR is to find the best fit line. In SVR, the best fit line is the hyperplane that has the maximum number of points

California Housing Dataset : The data contains information from the 1990 California census ,it does provide an accessible introductory dataset for teaching people about the basics of machine learning.

CODE/OUTPUT : -

```
# IMPORT LIBRARIES

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns


test=pd.read_csv("./california_housing_test.csv")

train=pd.read_csv("./california_housing_train.csv")


train.head()
```

Out[3]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.00	37.55	27	6103	1249	3026	1134	4.1591	332400
1	-122.07	37.93	25	7201	1521	3264	1433	3.7433	252100
2	-118.02	33.90	34	2678	511	1540	497	4.4954	202900
3	-121.79	39.73	8	5690	1189	2887	1077	3.0625	116300
4	-120.90	39.93	23	2679	546	1424	529	2.8812	81900

test.tail()

Out[4]:

	Unnamed: 0	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
3397	3398	-118.33	34.09	36	654	186	416	138	3.6953
3398	3399	-117.88	34.09	29	3416	790	2223	728	3.5109
3399	3400	-118.32	34.26	32	3690	791	1804	715	4.4875
3400	3401	-118.12	33.80	35	1835	435	774	418	2.7092
3401	3402	-118.19	33.78	42	1021	300	533	187	1.8036

print(train.info())

print(test.info())

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13598 entries, 0 to 13597
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           13598 non-null  float64
 1   latitude            13598 non-null  float64
 2   housing_median_age  13598 non-null  int64
 3   total_rooms         13598 non-null  int64
 4   total_bedrooms      13598 non-null  int64
 5   population          13598 non-null  int64
 6   households          13598 non-null  int64
 7   median_income       13598 non-null  float64
 8   median_house_value  13598 non-null  int64
dtypes: float64(3), int64(6)
memory usage: 956.2 KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3402 entries, 0 to 3401
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Unnamed: 0          3402 non-null   int64
 1   longitude           3402 non-null   float64
 2   latitude            3402 non-null   float64
 3   housing_median_age  3402 non-null   int64
 4   total_rooms         3402 non-null   int64
 5   total_bedrooms      3402 non-null   int64
 6   population          3402 non-null   int64
 7   households          3402 non-null   int64
 8   median_income       3402 non-null   float64
dtypes: float64(3), int64(6)
memory usage: 239.3 KB
None
```

n_train = train.shape[0]

n_test = test.shape[0]

y = train['median_house_value'].values

data = pd.concat((train, test)).reset_index(drop = True)

data.drop(['longitude','latitude'], axis=1, inplace = True)

#VISUALISING THE DATA

#Visualise the data

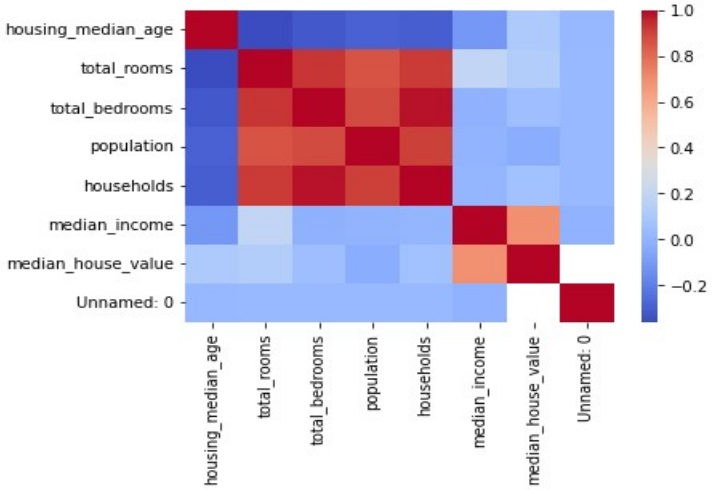
plt.figure()

sns.heatmap(data.corr(), cmap='coolwarm')

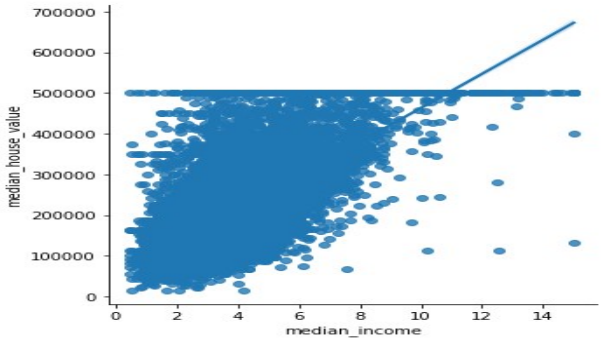
plt.show()

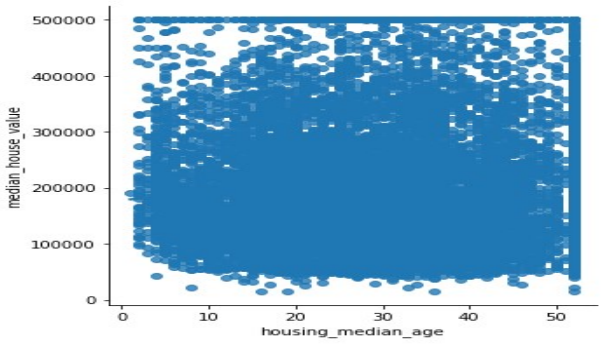
sns.lmplot(x='median_income', y='median_house_value', data=train)

sns.lmplot(x='housing_median_age', y='median_house_value', data=train)




```
Out[7]: <seaborn.axisgrid.FacetGrid at 0x1cf0d13ea00>
```

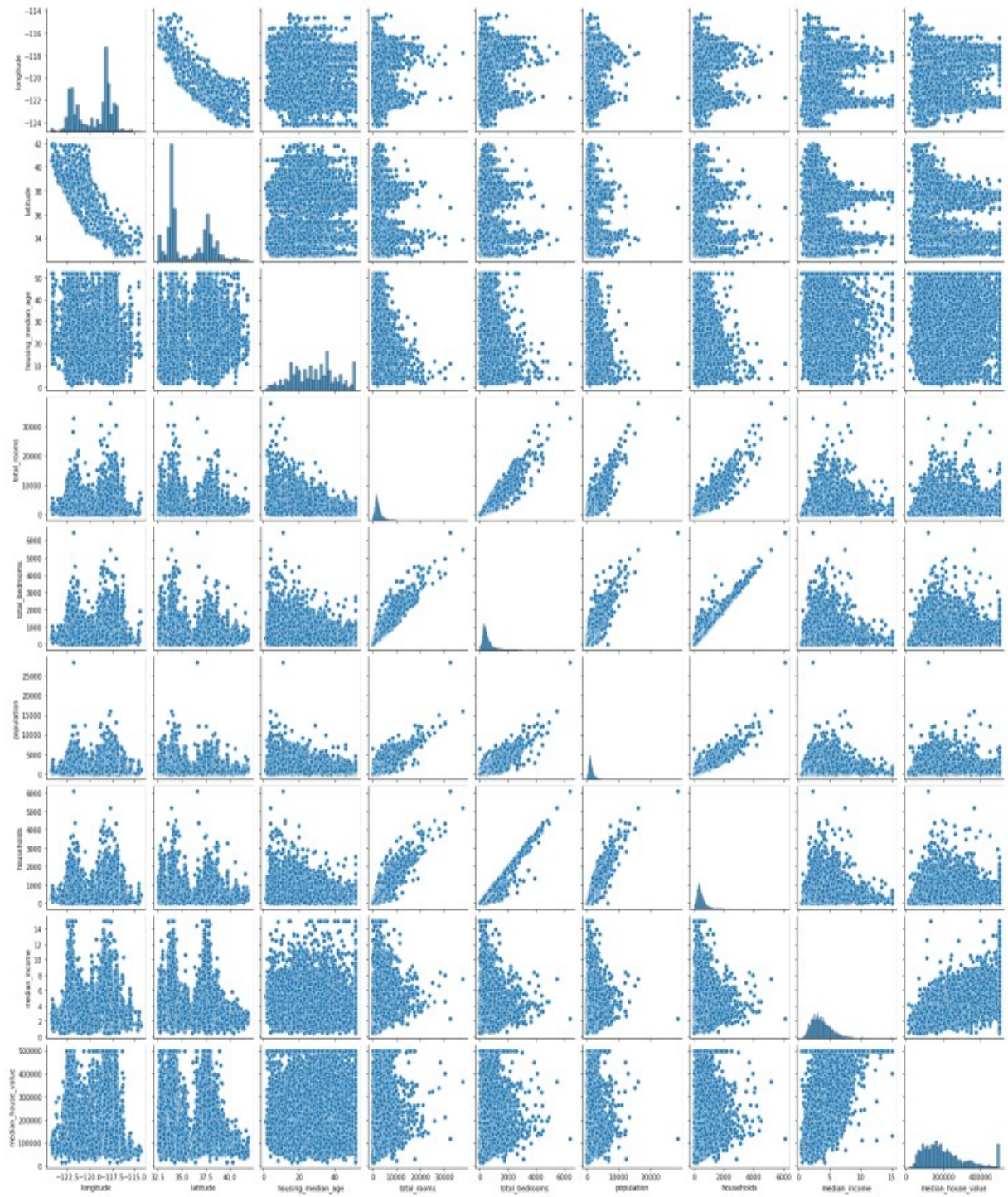




sns.pairplot(train, palette='rainbow')

Out[8]:

<seaborn.axisgrid.PairGrid at 0x1cf0d38d430>



#FEATURE ENGINEERING

#Feature engineering is the process of using domain knowledge to extract features from raw data via data mining techniques.

#Select appropriate features

data = data[['total_rooms', 'total_bedrooms', 'housing_median_age','median_income', 'population', 'households']]

data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17000 entries, 0 to 16999
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   total_rooms         17000 non-null  int64  
 1   total_bedrooms      17000 non-null  int64  
 2   housing_median_age  17000 non-null  int64  
 3   median_income       17000 non-null  float64
 4   population          17000 non-null  int64  
 5   households          17000 non-null  int64  
dtypes: float64(1), int64(5)
memory usage: 797.0 KB
```

data['total_rooms'] = data['total_rooms'].fillna(data['total_rooms'].mean())

data['total_bedrooms'] = data['total_bedrooms'].fillna(data['total_bedrooms'].mean())

data['housing_median_age'] =
data['housing_median_age'].fillna(data['housing_median_age'].mean())

data['median_income'] = data['median_income'].fillna(data['median_income'].mean())

data['population'] = data['population'].fillna(data['population'].mean())

data['households'] = data['households'].fillna(data['households'].mean())

```
train = data[:n_train]
test = data[n_train:]



#FITTING THE MODEL
#Split the dataset into training and testing data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train, y, test_size = 0.2)
y_train = y_train.reshape(-1,1)
y_test = y_test.reshape(-1,1)
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.fit_transform(X_test)
y_train = sc_y.fit_transform(y_train)
y_test = sc_y.fit_transform(y_test)

# Fit the model over the training data
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X_train, y_train)
```

```
c:\users\harsh\appdata\local\programs\python\python38\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarning:
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using rav
el().
  y = column_or_1d(y, warn=True)
```

Out[14]: SVR()

y_pred = regressor.predict(X_test)

y_pred = y_pred.reshape(1,-1)

y_pred = sc_y.inverse_transform(y_pred)

y_pred

```
Out[15]: array([[179219.93127152, 134049.46739203, 287742.07026065, ...,
                 164907.28374701,  90330.41746274, 254711.02129636]])
```

df = pd.DataFrame({'Real Values':sc_y.inverse_transform(y_test.reshape(1,-1)).ravel(),'Predicted Values':y_pred.ravel()})

df

Out[17]:

	Real Values	Predicted Values
0	227900.0	179219.931272
1	81100.0	134049.467392
2	247100.0	287742.070261
3	446200.0	437493.815678
4	135200.0	131255.773320
...	...	...
2715	122500.0	104581.879284
2716	421000.0	353816.134198
2717	500001.0	164907.283747
2718	94300.0	90330.417463
2719	188900.0	254711.021296

2720 rows × 2 columns

Practical 8

AIM :- Implement MLP for classification of handwritten digits (MNIST Dataset)

THEORY : -

MLP : A multilayer perceptron (MLP) is a feedforward artificial neural network that generates a set of outputs from a set of inputs. An MLP is characterized by several layers of input nodes connected as a directed graph between the input and output layers. MLP uses backpropogation for training the network.

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.The database is widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NIST's original datasets. The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset. The original creators of the database keep a list of some of the methods tested on it.In their original paper, they use a support-vector machine to get an error rate of 0.8%.

CODE/OUTPUT : -

```
#import libraries

import tensorflow as tf

from keras.utils import np_utils

from keras.datasets import mnist

import seaborn as sns

from keras.initializers import RandomNormal

%matplotlib notebook


import matplotlib.pyplot as plt

import numpy as np

import time
```

```
#function for updating plots for each each epoch and error

def plt_dynamic (x,vy, ty, ax, colors=['b']):

ax.plot (x, vy, 'b',label='Validation Loss')

ax.plot(x,vy, 'r',label='Trin Loss')

plt.legend()

plt.grid()

#fig.canvas.draw()



#train and test data

(X_train,y_train), (X_test,y_test)=mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 2s 0us/step
11501568/11490434 [==============================] - 2s 0us/step
```

```
X_train=X_train.reshape(X_train.shape [0], X_train.shape [1]*X_train.shape[2])

X_test=X_test.reshape(X_test.shape [0],X_test.shape [1]*X_test.shape [2])



#print('Number of Training Examples:', X_ train. shape [0], 'and each image is of shape (%d,%d)'
%(X_train.shape[1]*X_train.shape[2]))

#print('Number of Test Examples:', X_test.shape[0], 'and each image is of shape (%d,%d)'
%(X_test.shape[1],X_test.shape[2]))

print('Number of Training Examples:',X_train.shape [0], 'and each image is of shape (%d)'
%(X_train.shape[1]))

print('Number of Test Examples:',X_test.shape [0], 'and each image is of shape (%d)'
%(X_test.shape[1]))
```

```
Number of Training Examples: 60000 and each image is of shape (784)
Number of Test Examples: 10000 and each image is of shape (784)
```

example of data point

print(X_train[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   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   3  18  18  18 126 136 175  26 166 255
  247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  36  94 154
  170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0   0   0
    0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82
   82  56  39   0   0   0   0   0   0   0   0   0   0   0   0  18 219 253
  253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0  11 190 253  70   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  35 241
  225 160 108   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
  253 207   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253
  253 201  78   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0  18 171 219 253 253 253 253 195
   80   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0 136 253 253 253 212 135 132  16
    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   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
```

normalize the data

X_train=X_train/255

X_test=X_test/255

example of data after normalization

print(X_train[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.         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.         0.01176471 0.07058824 0.07058824 0.07058824
0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
0.96862745 0.49803922 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.19215686
0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863

   0.32156863 0.21960784 0.15294118 0.         0.         0.
   0.         0.         0.         0.         0.         0.
   0.         0.         0.         0.07058824 0.85882353 0.99215686
   0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
   0.96862745 0.94509804 0.         0.         0.         0.
   0.         0.         0.         0.         0.         0.
   0.         0.         0.         0.         0.         0.
   0.         0.         0.31372549 0.61176471 0.41960784 0.99215686
   0.99215686 0.80392157 0.04313725 0.         0.16862745 0.60392157
   0.         0.         0.         0.         0.         0.
   0.         0.         0.         0.         0.         0.
   0.         0.         0.         0.         0.         0.
   0.         0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
   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.         0.54509804 0.99215686 0.74509804 0.00784314 0.
```

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.04313725
0.74509804 0.99215686 0.2745098  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.         0.1372549  0.94509804
0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.31764706 0.94117647 0.99215686
0.99215686 0.46666667 0.09803922 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
```

```
0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
0.58823529 0.10588235 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
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.         0.97647059 0.99215686 0.97647059 0.25098039 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
0.99215686 0.81176471 0.00784314 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
```

```
0.         0.         0.         0.         0.15294118 0.58039216
0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
0.99215686 0.78823529 0.30588235 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.09019608 0.25882353 0.83529412 0.99215686
0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.07058824 0.67058824
0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
0.31372549 0.03529412 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
```

```
0.21568627 0.6745098  0.88627451 0.99215686 0.99215686 0.99215686
0.99215686 0.95686275 0.52156863 0.04313725 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.53333333 0.99215686
0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
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.         0.
```

```
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         ]
```

class number for each image

print('Class label of first image:',y_train[0])

```
# convert this into 10 decimal vector
Y_train=np_utils.to_categorical(y_train,10)
Y_test=np_utils.to_categorical(y_test,10)
print('After converting the output into a vector:',Y_train[0])
```

```
Class label of first image: 5
After converting the output into a vector: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

```
# buildsoftmax classifier
from keras.models import Sequential
from keras.layers import Dense, Activation

#model=Sequential([Dense(32, input_shape=(784,)),
#Activation('relu'),Dense(10),Activation('softmax')])

#model parameters
output_dim=10
input_dim=X_train.shape[1]
batch_size=128
nb_epoch=20

#start building model
#from keras.layers import Activation, Dense
model=Sequential()
#model.add(Dense(64))
#model.add(Activation('tanh'))
model.add(Dense(output_dim,input_dim=input_dim, activation='softmax'))

print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(60000, 784)
(60000, 10)
(10000, 784)
(10000, 10)
```

Configure the learning process

```
model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
```

train using fit()

```
history = model.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1,
validation_data=(X_test, Y_test))
```

```
Epoch 1/20
469/469 [==============================] - 2s 2ms/step - loss: 1.2983 - accuracy: 0.6842 - val_loss: 0.8128 - val_accuracy: 0.8
332
Epoch 2/20
469/469 [==============================] - 1s 2ms/step - loss: 0.7187 - accuracy: 0.8410 - val_loss: 0.6071 - val_accuracy: 0.8
626
Epoch 3/20
469/469 [==============================] - 1s 2ms/step - loss: 0.5890 - accuracy: 0.8600 - val_loss: 0.5256 - val_accuracy: 0.8
736
Epoch 4/20
469/469 [==============================] - 1s 2ms/step - loss: 0.5271 - accuracy: 0.8692 - val_loss: 0.4799 - val_accuracy: 0.8
812
Epoch 5/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4894 - accuracy: 0.8754 - val_loss: 0.4500 - val_accuracy: 0.8
873
Epoch 6/20
469/469 [==============================] - 1s 3ms/step - loss: 0.4634 - accuracy: 0.8797 - val_loss: 0.4286 - val_accuracy: 0.8
908
Epoch 7/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4441 - accuracy: 0.8828 - val_loss: 0.4125 - val_accuracy: 0.8
925
Epoch 8/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4292 - accuracy: 0.8858 - val_loss: 0.3998 - val_accuracy: 0.8
953
Epoch 9/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4170 - accuracy: 0.8885 - val_loss: 0.3894 - val_accuracy: 0.8
974
Epoch 10/20
469/469 [==============================] - 1s 2ms/step - loss: 0.4070 - accuracy: 0.8907 - val_loss: 0.3809 - val_accuracy: 0.8
977
Epoch 11/20
469/469 [==============================] - 1s 2ms/step - loss: 0.3985 - accuracy: 0.8923 - val_loss: 0.3735 - val_accuracy: 0.8
998
Epoch 12/20
469/469 [==============================] - 1s 2ms/step - loss: 0.3911 - accuracy: 0.8938 - val_loss: 0.3674 - val_accuracy: 0.9
015
Epoch 13/20
469/469 [==============================] - 1s 2ms/step - loss: 0.3847 - accuracy: 0.8951 - val_loss: 0.3617 - val_accuracy: 0.9
015
Epoch 14/20
469/469 [==============================] - 1s 2ms/step - loss: 0.3790 - accuracy: 0.8962 - val_loss: 0.3565 - val_accuracy: 0.9
021
Epoch 15/20
469/469 [==============================] - 1s 2ms/step - loss: 0.3740 - accuracy: 0.8976 - val_loss: 0.3521 - val_accuracy: 0.9
037
Epoch 16/20
469/469 [==============================] - 1s 2ms/step - loss: 0.3694 - accuracy: 0.8991 - val_loss: 0.3483 - val_accuracy: 0.9
049
Epoch 17/20
469/469 [==============================] - 1s 2ms/step - loss: 0.3652 - accuracy: 0.8994 - val_loss: 0.3449 - val_accuracy: 0.9
060
Epoch 18/20
469/469 [==============================] - 1s 2ms/step - loss: 0.3614 - accuracy: 0.9006 - val_loss: 0.3416 - val_accuracy: 0.9
077
Epoch 19/20
469/469 [==============================] - 1s 2ms/step - loss: 0.3580 - accuracy: 0.9015 - val_loss: 0.3384 - val_accuracy: 0.9
080
Epoch 20/20
469/469 [==============================] - 1s 2ms/step - loss: 0.3548 - accuracy: 0.9021 - val_loss: 0.3355 - val_accuracy: 0.9
084
```

```
score=model.evaluate (X_test,Y_test, verbose=0)

print('Test Score',score [0])

print('Test Accuracy',score[1])


'''fig,ax=plt.subplot(1,1)

ax.set_xlabel('epoch');

ax.set_ylabel('Categorical Crossentropy Loss')
```

```
#List the epoc numbers
x=list(range(1, nb_epoch+1))

vy=history.history['val loss']
ty=history.history['loss']
plt_dynamic (x,uy,ty,ax)

'''
```

```
Test Score 0.3355453610420227
Test Accuracy 0.9083999991416931
```

```
Out[26]: "fig,ax=plt.subplot(1,1)\nax.set_xlabel('epoch');\nax.set_ylabel('Categorical Crossentropy Loss')\n\n#List the epoc numbers\nx=
list(range(1, nb_epoch+1))\n\nvy=history.history['val loss']\nty=history.history['loss'] \nplt_dynamic (x,uy,ty,ax)\n\n"
```

```
# multilayer perceptron
model_sigmoid=Sequential()
model_sigmoid.add(Dense(512,activation='sigmoid', input_shape=(input_dim,)))
model_sigmoid.add(Dense(128, activation='sigmoid'))
model_sigmoid.add(Dense(output_dim,activation='softmax'))
model_sigmoid.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 512)               401920

 dense_2 (Dense)             (None, 128)               65664

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 468,874
Trainable params: 468,874
Non-trainable params: 0
```

```
model_sigmoid.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])
```

history = model_sigmoid.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

```
Epoch 1/20
469/469 [==============================] - 6s 11ms/step - loss: 2.2703 - accuracy: 0.2298 - val_loss: 2.2228 - val_accuracy: 0.
2836
Epoch 2/20
469/469 [==============================] - 5s 10ms/step - loss: 2.1768 - accuracy: 0.4245 - val_loss: 2.1195 - val_accuracy: 0.
5760
Epoch 3/20
469/469 [==============================] - 4s 9ms/step - loss: 2.0574 - accuracy: 0.5416 - val_loss: 1.9786 - val_accuracy: 0.5
896
Epoch 4/20
469/469 [==============================] - 4s 9ms/step - loss: 1.8930 - accuracy: 0.6181 - val_loss: 1.7855 - val_accuracy: 0.6
382
Epoch 5/20
469/469 [==============================] - 4s 9ms/step - loss: 1.6886 - accuracy: 0.6687 - val_loss: 1.5656 - val_accuracy: 0.6
840
Epoch 6/20
469/469 [==============================] - 4s 9ms/step - loss: 1.4694 - accuracy: 0.7106 - val_loss: 1.3458 - val_accuracy: 0.7
406
Epoch 7/20
469/469 [==============================] - 5s 10ms/step - loss: 1.2648 - accuracy: 0.7435 - val_loss: 1.1567 - val_accuracy: 0.
7628
Epoch 8/20
469/469 [==============================] - 4s 9ms/step - loss: 1.0956 - accuracy: 0.7682 - val_loss: 1.0080 - val_accuracy: 0.7
882
Epoch 9/20
469/469 [==============================] - 5s 10ms/step - loss: 0.9648 - accuracy: 0.7881 - val_loss: 0.8957 - val_accuracy: 0.
7998
Epoch 10/20
469/469 [==============================] - 5s 10ms/step - loss: 0.8649 - accuracy: 0.8032 - val_loss: 0.8075 - val_accuracy: 0.
8191
Epoch 11/20
469/469 [==============================] - 4s 9ms/step - loss: 0.7871 - accuracy: 0.8158 - val_loss: 0.7395 - val_accuracy: 0.8
271
Epoch 12/20
469/469 [==============================] - 5s 10ms/step - loss: 0.7250 - accuracy: 0.8268 - val_loss: 0.6827 - val_accuracy: 0.
8386
Epoch 13/20
469/469 [==============================] - 4s 10ms/step - loss: 0.6746 - accuracy: 0.8355 - val_loss: 0.6371 - val_accuracy: 0.
8472
Epoch 14/20
469/469 [==============================] - 4s 9ms/step - loss: 0.6329 - accuracy: 0.8435 - val_loss: 0.5993 - val_accuracy: 0.8
521
Epoch 15/20
469/469 [==============================] - 4s 9ms/step - loss: 0.5979 - accuracy: 0.8494 - val_loss: 0.5674 - val_accuracy: 0.8
596
Epoch 16/20
469/469 [==============================] - 4s 9ms/step - loss: 0.5684 - accuracy: 0.8561 - val_loss: 0.5394 - val_accuracy: 0.8
633
Epoch 17/20
469/469 [==============================] - 4s 9ms/step - loss: 0.5432 - accuracy: 0.8605 - val_loss: 0.5168 - val_accuracy: 0.8
685
Epoch 18/20
469/469 [==============================] - 5s 10ms/step - loss: 0.5215 - accuracy: 0.8649 - val_loss: 0.4964 - val_accuracy: 0.
8731
Epoch 19/20
469/469 [==============================] - 4s 9ms/step - loss: 0.5025 - accuracy: 0.8687 - val_loss: 0.4786 - val_accuracy: 0.8
765
Epoch 20/20
469/469 [==============================] - 4s 9ms/step - loss: 0.4860 - accuracy: 0.8716 - val_loss: 0.4634 - val_accuracy: 0.8
793
```

w_after = model_sigmoid.get_weights()

h1_w = w_after[0].flatten().reshape(-1,1)

h2_w = w_after[2].flatten().reshape(-1,1)

out_w = w_after[4].flatten().reshape(-1,1)

fig = plt.figure()

plt.title("Weight matrices after model trained")

plt.subplot (1, 3, 1)

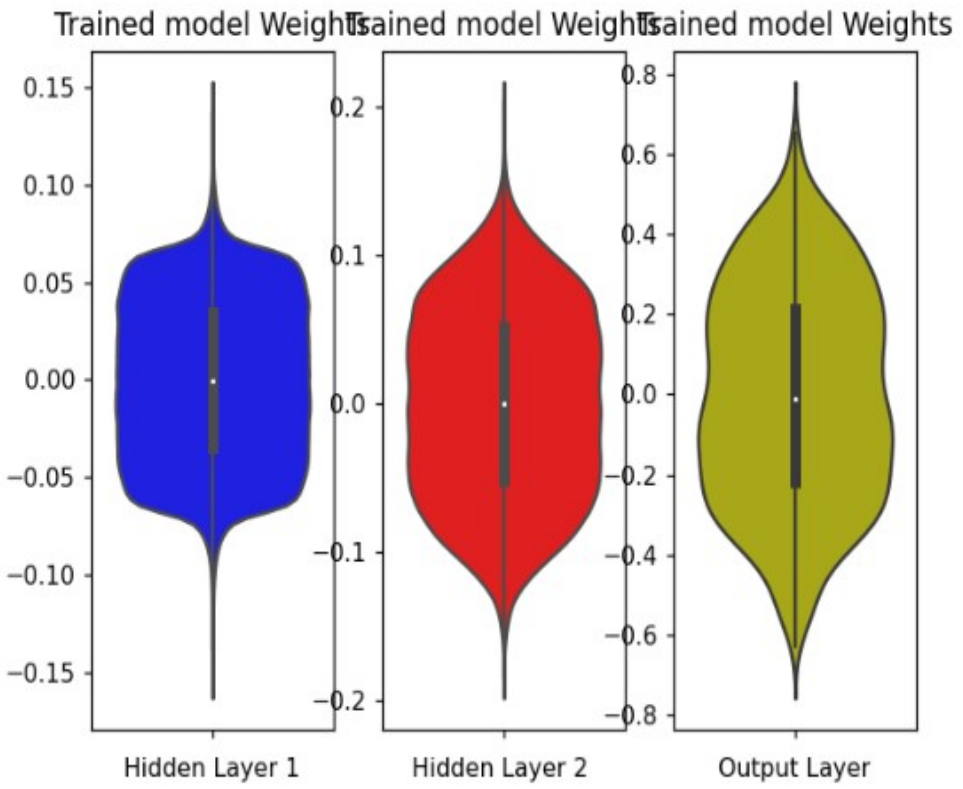
plt.title("Trained model Weights")

ax = sns.violinplot (y=h1_w, color='b')

plt.xlabel('Hidden Layer 1')

```
plt.subplot (1, 3, 2)
plt.title("Trained model Weights")
ax = sns.violinplot (y=h2_w, color='r')
plt.xlabel('Hidden Layer 2 ')

plt.subplot (1, 3, 3)
plt.title("Trained model Weights")
ax = sns.violinplot (y=out_w,color='y')
plt.xlabel('Output Layer')
plt.show()
```

```
IPython.core.display.Javascript object>
```



Practical 9

AIM :- Classification of images of clothing using TensorFlow (Fashion MNIST Dataset)

THEORY :-

A convolution multiplies a matrix of pixels with a filter matrix or 'kernel' and sums up the multiplication values. Then the convolution slides over to the next pixel and repeats the same process until all the image pixels have been covered.

CODE/OUTPUT: -

```
# Classification using Tensor flow on MNIST cloth dataset

#import libraries

import tensorflow as tf


# access the dataset

fashion_mnist=tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data( )



# print the training and test dataset

train_images.shape
```

```
Out[3]: (60000, 28, 28)
```

```
train_images [0]
```

Out[4]:
```
array([[  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,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,
          0,   0,  13,  73,   0,   0,   1,   4,   0,   0,   0,   0,   1,
          1,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
          0,  36, 136, 127,  62,  54,   0,   0,   0,   1,   3,   4,   0,
          0,   3],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,
          0, 102, 204, 176, 134, 144, 123,  23,   0,   0,   0,   0,  12,
         10,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0, 155, 236, 207, 178, 107, 156, 161, 109,  64,  23,  77, 130,
         72,  15],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,
         69, 207, 223, 218, 216, 216, 163, 127, 121, 122, 146, 141,  88,
        172,  66],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   1,   1,   0,
        200, 232, 232, 233, 229, 223, 223, 215, 213, 164, 127, 123, 196,
        229,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        183, 225, 216, 223, 228, 235, 227, 224, 222, 224, 221, 223, 245,
        173,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        193, 228, 218, 213, 198, 180, 212, 210, 211, 213, 223, 220, 243,
        202,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   3,   0,  12,
        219, 220, 212, 218, 192, 169, 227, 208, 218, 224, 212, 226, 197,
        209,  52],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,   0,  99,
        244, 222, 220, 218, 203, 198, 221, 215, 213, 222, 220, 245, 119,
        167,  56],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   4,   0,   0,  55,
        236, 228, 230, 228, 240, 232, 213, 218, 223, 234, 217, 217, 209,
         92,   0],
       [  0,   0,   1,   4,   6,   7,   2,   0,   0,   0,   0,   0, 237,
        226, 217, 223, 222, 219, 222, 221, 216, 223, 229, 215, 218, 255,
         77,   0],
       [  0,   3,   0,   0,   0,   0,   0,   0,   0,  62, 145, 204, 228,
        207, 213, 221, 218, 208, 211, 218, 224, 223, 219, 215, 224, 244,
        159,   0],
       [  0,   0,   0,   0,  18,  44,  82, 107, 189, 228, 220, 222, 217,
        226, 200, 205, 211, 230, 224, 234, 176, 188, 250, 248, 233, 238,
        215,   0],
       [  0,  57, 187, 208, 224, 221, 224, 208, 204, 214, 208, 209, 200,
        159, 245, 193, 206, 223, 255, 255, 221, 234, 221, 211, 220, 232,
        246,   0],
       [  3, 202, 228, 224, 221, 211, 211, 214, 205, 205, 205, 220, 240,
         80, 150, 255, 229, 221, 188, 154, 191, 210, 204, 209, 222, 228,
```

```
[ 98, 233, 198, 210, 222, 229, 229, 234, 249, 220, 194, 215, 217,
  241,  65,  73, 106, 117, 168, 219, 221, 215, 217, 223, 223, 224,
  229,  29],
[ 75, 204, 212, 204, 193, 205, 211, 225, 216, 185, 197, 206, 198,
  213, 240, 195, 227, 245, 239, 223, 218, 212, 209, 222, 220, 221,
  230,  67],
[ 48, 203, 183, 194, 213, 197, 185, 190, 194, 192, 202, 214, 219,
  221, 220, 236, 225, 216, 199, 206, 186, 181, 177, 172, 181, 205,
  206, 115],
[  0, 122, 219, 193, 179, 171, 183, 196, 204, 210, 213, 207, 211,
  210, 200, 196, 194, 191, 195, 191, 198, 192, 176, 156, 167, 177,
  210,  92],
[  0,   0,  74, 189, 212, 191, 175, 172, 175, 181, 185, 188, 189,
  188, 193, 198, 204, 209, 210, 210, 211, 188, 188, 194, 192, 216,
  170,   0],
[  2,   0,   0,   0,  66, 200, 222, 237, 239, 242, 246, 243, 244,
  221, 220, 193, 191, 179, 182, 182, 181, 176, 166, 168,  99,  58,
    0,   0],
[  0,   0,   0,   0,   0,   0,   0,  40,  61,  44,  72,  41,  35,
    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]], dtype=uint8)
```

train_labels

```
Out[5]: array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

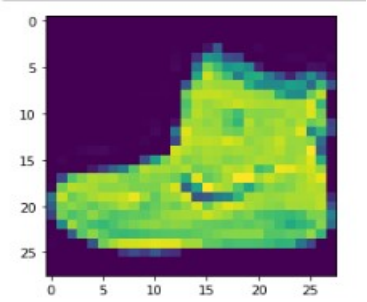
test_images.shape

```
Out[6]: (10000, 28, 28)
```

test_labels.shape

```
Out[7]: (10000,)
```

```
# visualization of training data

import matplotlib.pyplot as plt


# plot the first train_image

plt.imshow(train_images[0])

plt.show()
```




```
# plot multiple images and its label

class_names = ['T-shirt/ top' , 'Trouser' , ' Pullover' , 'Dress ' , 'Coat ' , 'Sandal' , ' Shirt ' , 'Sneaker','Bag','Ankle boot']

plt.figure(figsize=(15,15))


for i in range(25):

plt.subplot(5,5,i+1)

plt.imshow(train_images[i])

plt.xlabel(class_names[train_labels[i]])

plt.xticks([])

plt.yticks([])

   pass
```

plt.show()

scale images to a range of 0 to 1

train_images=train_images/255.0

test_images=test_images/255.0

see the train_image[0] after scaling

train_images[0]

```
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.00392157, 0.        , 0.        ,
 0.05098039, 0.28627451, 0.        , 0.        , 0.00392157,
 0.01568627, 0.        , 0.        , 0.        , 0.        ,
 0.00392157, 0.00392157, 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.01176471, 0.        , 0.14117647,
 0.53333333, 0.49803922, 0.24313725, 0.21176471, 0.        ,
 0.        , 0.        , 0.00392157, 0.01176471, 0.01568627,
 0.        , 0.        , 0.01176471],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.02352941, 0.        , 0.4       ,
 0.8       , 0.69019608, 0.5254902 , 0.56470588, 0.48235294,
 0.09019608, 0.        , 0.        , 0.        , 0.        ,
 0.04705882, 0.03921569, 0.        ],
```

```
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.60784314,
 0.9254902 , 0.81176471, 0.69803922, 0.41960784, 0.61176471,
 0.63137255, 0.42745098, 0.25098039, 0.09019608, 0.30196078,
 0.50980392, 0.28235294, 0.05882353],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.00392157, 0.        , 0.27058824, 0.81176471,
 0.8745098 , 0.85490196, 0.84705882, 0.84705882, 0.63921569,
 0.49803922, 0.4745098 , 0.47843137, 0.57254902, 0.55294118,
 0.34509804, 0.6745098 , 0.25882353],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.00392157,
 0.00392157, 0.00392157, 0.        , 0.78431373, 0.90980392,
 0.90980392, 0.91372549, 0.89803922, 0.8745098 , 0.8745098 ,
 0.84313725, 0.83529412, 0.64313725, 0.49803922, 0.48235294,
 0.76862745, 0.89803922, 0.        ],
```

```
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.71764706, 0.88235294,
 0.84705882, 0.8745098 , 0.89411765, 0.92156863, 0.89019608,
 0.87843137, 0.87058824, 0.87843137, 0.86666667, 0.8745098 ,
 0.96078431, 0.67843137, 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.75686275, 0.89411765,
 0.85490196, 0.83529412, 0.77647059, 0.70588235, 0.83137255,
 0.82352941, 0.82745098, 0.83529412, 0.8745098 , 0.8627451 ,
 0.95294118, 0.79215686, 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.00392157,
 0.01176471, 0.        , 0.04705882, 0.85882353, 0.8627451 ,
 0.83137255, 0.85490196, 0.75294118, 0.6627451 , 0.89019608,
 0.81568627, 0.85490196, 0.87843137, 0.83137255, 0.88627451,
 0.77254902, 0.81960784, 0.20392157],
```

```
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.02352941, 0.        , 0.38823529, 0.95686275, 0.87058824,
 0.8627451 , 0.85490196, 0.79607843, 0.77647059, 0.86666667,
 0.84313725, 0.83529412, 0.87058824, 0.8627451 , 0.96078431,
 0.46666667, 0.65490196, 0.21960784],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.01568627,
 0.        , 0.        , 0.21568627, 0.9254902 , 0.89411765,
 0.90196078, 0.89411765, 0.94117647, 0.90980392, 0.83529412,
 0.85490196, 0.8745098 , 0.91764706, 0.85098039, 0.85098039,
 0.81960784, 0.36078431, 0.        ],
[0.        , 0.        , 0.00392157, 0.01568627, 0.02352941,
 0.02745098, 0.00784314, 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.92941176, 0.88627451, 0.85098039,
 0.8745098 , 0.87058824, 0.85882353, 0.87058824, 0.86666667,
 0.84705882, 0.8745098 , 0.89803922, 0.84313725, 0.85490196,
```

```
 1.        , 0.30196078, 0.        ],
[0.        , 0.01176471, 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.24313725,
 0.56862745, 0.8       , 0.89411765, 0.81176471, 0.83529412,
 0.86666667, 0.85490196, 0.81568627, 0.82745098, 0.85490196,
 0.87843137, 0.8745098 , 0.85882353, 0.84313725, 0.87843137,
 0.95686275, 0.62352941, 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.07058824,
 0.17254902, 0.32156863, 0.41960784, 0.74117647, 0.89411765,
 0.8627451 , 0.87058824, 0.85098039, 0.88627451, 0.78431373,
 0.80392157, 0.82745098, 0.90196078, 0.87843137, 0.91764706,
 0.69019608, 0.7372549 , 0.98039216, 0.97254902, 0.91372549,
 0.93333333, 0.84313725, 0.        ],
[0.        , 0.22352941, 0.73333333, 0.81568627, 0.87843137,
 0.86666667, 0.87843137, 0.81568627, 0.8       , 0.83921569,
 0.81568627, 0.81960784, 0.78431373, 0.62352941, 0.96078431,
 0.75686275, 0.80784314, 0.8745098 , 1.        , 1.        ,
 0.86666667, 0.91764706, 0.86666667, 0.82745098, 0.8627451 ,
 0.90980392, 0.96470588, 0.        ],
```

```
[0.01176471, 0.79215686, 0.89411765, 0.87843137, 0.86666667,
 0.82745098, 0.82745098, 0.83921569, 0.80392157, 0.80392157,
 0.80392157, 0.8627451 , 0.94117647, 0.31372549, 0.58823529,
 1.        , 0.89803922, 0.86666667, 0.7372549 , 0.60392157,
 0.74901961, 0.82352941, 0.8       , 0.81960784, 0.87058824,
 0.89411765, 0.88235294, 0.        ],
[0.38431373, 0.91372549, 0.77647059, 0.82352941, 0.87058824,
 0.89803922, 0.89803922, 0.91764706, 0.97647059, 0.8627451 ,
 0.76078431, 0.84313725, 0.85098039, 0.94509804, 0.25490196,
 0.28627451, 0.41568627, 0.45882353, 0.65882353, 0.85882353,
 0.86666667, 0.84313725, 0.85098039, 0.8745098 , 0.8745098 ,
 0.87843137, 0.89803922, 0.11372549],
[0.29411765, 0.8       , 0.83137255, 0.8       , 0.75686275,
 0.80392157, 0.82745098, 0.88235294, 0.84705882, 0.7254902 ,
 0.77254902, 0.80784314, 0.77647059, 0.83529412, 0.94117647,
 0.76470588, 0.89019608, 0.96078431, 0.9372549 , 0.8745098 ,
 0.85490196, 0.83137255, 0.81960784, 0.87058824, 0.8627451 ,
 0.86666667, 0.90196078, 0.2627451 ],
```

```
[0.18823529, 0.79607843, 0.71764706, 0.76078431, 0.83529412,
 0.77254902, 0.7254902 , 0.74509804, 0.76078431, 0.75294118,
 0.79215686, 0.83921569, 0.85882353, 0.86666667, 0.8627451 ,
 0.9254902 , 0.88235294, 0.84705882, 0.78039216, 0.80784314,
 0.72941176, 0.70980392, 0.69411765, 0.6745098 , 0.70980392,
 0.80392157, 0.80784314, 0.45098039],
[0.        , 0.47843137, 0.85882353, 0.75686275, 0.70196078,
 0.67058824, 0.71764706, 0.76862745, 0.8       , 0.82352941,
 0.83529412, 0.81176471, 0.82745098, 0.82352941, 0.78431373,
 0.76862745, 0.76078431, 0.74901961, 0.76470588, 0.74901961,
 0.77647059, 0.75294118, 0.69019608, 0.61176471, 0.65490196,
 0.69411765, 0.82352941, 0.36078431],
[0.        , 0.        , 0.29019608, 0.74117647, 0.83137255,
 0.74901961, 0.68627451, 0.6745098 , 0.68627451, 0.70980392,
 0.7254902 , 0.7372549 , 0.74117647, 0.7372549 , 0.75686275,
 0.77647059, 0.8       , 0.81960784, 0.82352941, 0.82352941,
 0.82745098, 0.7372549 , 0.7372549 , 0.76078431, 0.75294118,

 0.84705882, 0.66666667, 0.        ],
[0.00784314, 0.        , 0.        , 0.        , 0.25882353,
 0.78431373, 0.87058824, 0.92941176, 0.9372549 , 0.94901961,
 0.96470588, 0.95294118, 0.95686275, 0.86666667, 0.8627451 ,
 0.75686275, 0.74901961, 0.70196078, 0.71372549, 0.71372549,
 0.70980392, 0.69019608, 0.65098039, 0.65882353, 0.38823529,
 0.22745098, 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.15686275, 0.23921569, 0.17254902,
 0.28235294, 0.16078431, 0.1372549 , 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.        ]])
```

```
# create model
model=tf.keras.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
model.add (tf.keras.layers.Dense(128,activation='relu'))
model.add(tf.keras.layers.Dense(10))
model.compile(optimizer= 'adam' ,
        loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=['accuracy'])
```

```
# model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
```

```
# Train the Model
history=model.fit(train_images,train_labels,epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 7s 3ms/step - loss: 0.5013 - accuracy: 0.8233
Epoch 2/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3744 - accuracy: 0.8660
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3352 - accuracy: 0.8777
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3113 - accuracy: 0.8861
Epoch 5/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2937 - accuracy: 0.8915
Epoch 6/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2798 - accuracy: 0.8969
Epoch 7/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2678 - accuracy: 0.9002
Epoch 8/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2582 - accuracy: 0.9028
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2475 - accuracy: 0.9076
Epoch 10/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2390 - accuracy: 0.9108
```

```
loss=history.history['loss']
acc=history.history['accuracy']

epochs_range=range(10)
plt.plot(epochs_range,loss,label= 'Training Loss')
plt.plot(epochs_range,acc,label= 'Training Accuracy')
plt.legend(loc='lower right')
plt.title('Training Loss and Accuracy')
```

```
Out[16]: Text(0.5, 1.0, 'Training Loss and Accuracy')
```

```
# Evaluate the accuracy
test_loss, test_acc=model.evaluate(test_images, test_labels, verbose=2)
print( ' \Test accuracy: ' , test_acc)
```

```
313/313 - 1s - loss: 0.3398 - accuracy: 0.8820 - 818ms/epoch - 3ms/step
 \Test accuracy:  0.8820000290870667
```

```
# Predict
probability_model=tf.keras.Sequential([model,tf.keras.layers.Softmax()])
predictions=probability_model.predict(test_images)
predictions[0]
```

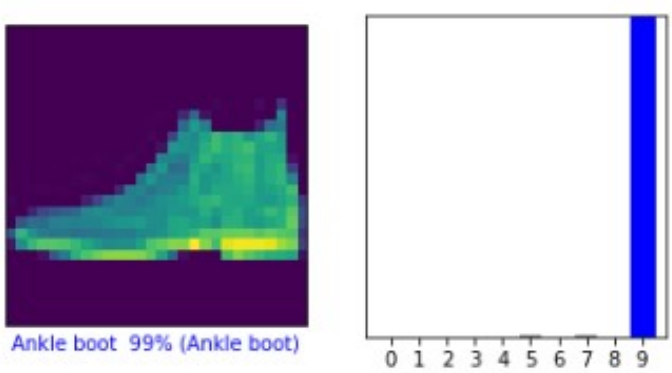
```
Out[18]: array([3.8962352e-07, 1.4807861e-07, 1.2840245e-06, 9.7617547e-10,
                8.7510152e-06, 2.5299708e-03, 1.3094771e-06, 6.3346457e-03,
                3.1729502e-07, 9.9112320e-01], dtype=float32)
```

```
# view prediction
import numpy as np
np.argmax(predictions[0])
```

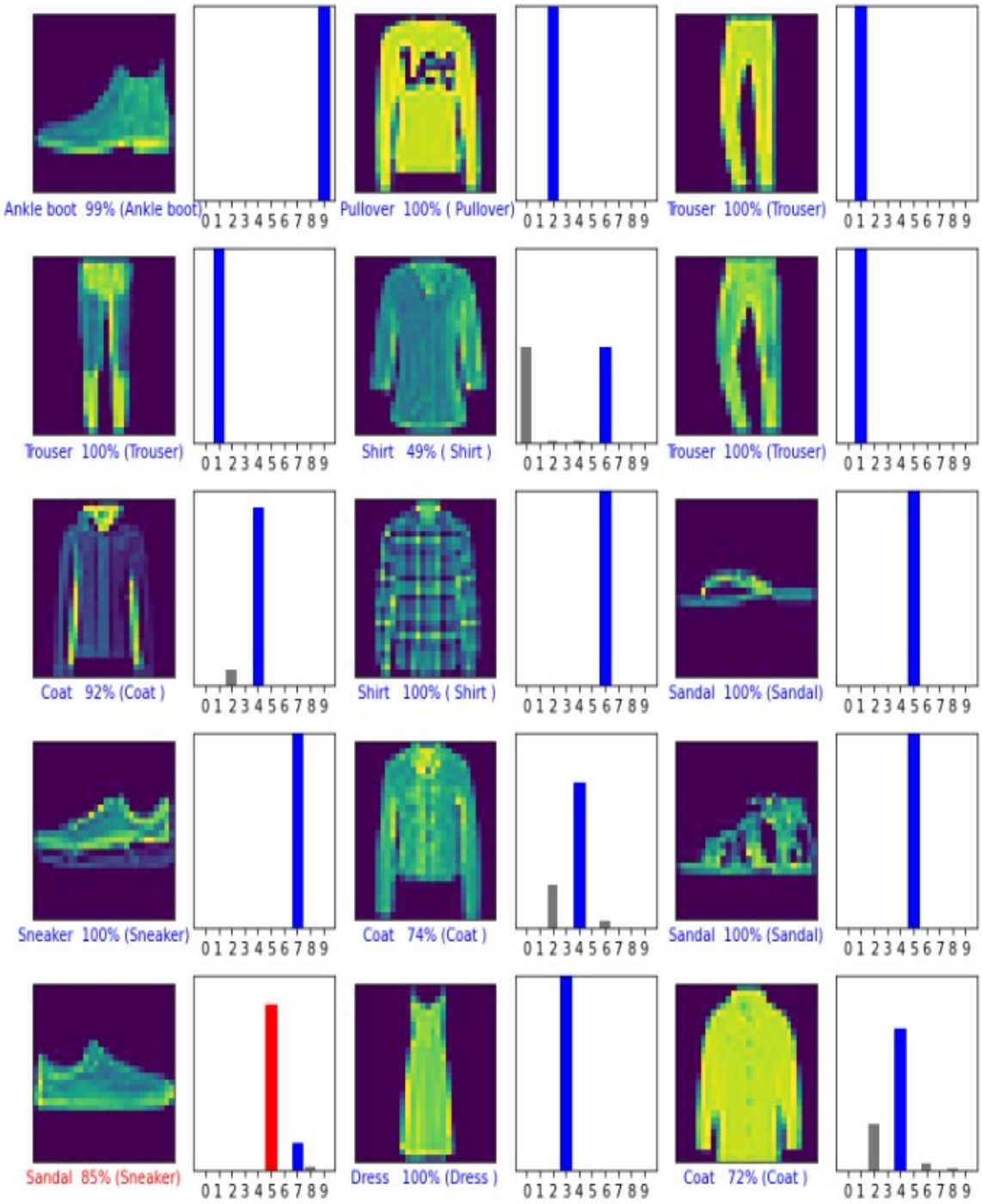
```
Out[19]: 9
```

```
#verify predictions
def plot_image(i, predictions_array, true_label, img) :
true_label, img=true_label [i] ,img[i]
plt.grid(False)
plt.xticks([])
plt.yticks([])
plt.imshow(img)
predicted_label = np.argmax(predictions_array)

   if predicted_label == true_label:
      color = 'blue'
   else:
      color =  'red'
```

```
plt.xlabel("{}  {:2.0f}% ({})".format(class_names[predicted_label],
                           100*np.max(predictions_array),
class_names[true_label]),
                           color=color)
   pass

def plot_value_array(i, predictions_array, true_label):
true_label = true_label[i]
plt.grid(False)
plt.xticks(range(10))
plt.yticks([])
thisplot = plt.bar(range(10), predictions_array, color="#777777")
plt.ylim([0,1])
predicted_label = np.argmax(predictions_array)
thisplot[predicted_label].set_color('red')
thisplot[true_label].set_color('blue')
   pass


# Verify the 0th image prediction
i=0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i,predictions[i],test_labels,test_images)
plt.subplot(1,2,2)
plot_value_array(i,predictions[i],test_labels)
plt.show()
```




```
# plot images with their predictions
num_rows = 5
num_cols = 3
num_images = num_rows * num_cols
plt.figure(figsize=(10,10))
for i in range(num_images):
plt.subplot(num_rows, 2 * num_cols, 2 * i + 1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(num_rows, 2 * num_cols, 2 * i + 2)
plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```


Ankle boot 99% (Ankle boot)
Pullover 100% (Pullover)
Trouser 100% (Trouser)
Trouser 100% (Trouser)
Shirt 49% (Shirt)
Trouser 100% (Trouser)
Coat 92% (Coat)
Shirt 100% (Shirt)
Sandal 100% (Sandal)
Sneaker 100% (Sneaker)
Coat 74% (Coat)
Sandal 100% (Sandal)
Sandal 85% (Sneaker)
Dress 100% (Dress)
Coat 72% (Coat)
0 1 2 3 4 5 6 7 8 9