**Practical No. 1** Aim: To perform the cross validation techniques. Theory:

Cross validation is a technique used in machine learning to evaluate the performance of a model on unseen dataThe main purpose of cross validation is to prevent overfitting, which occurs when a model is trained too well on the training data and performs poorly on new, unseen data. By evaluating the model on multiple validation sets, cross validation provides a more realistic estimate of the model’s generalization performance, i.e., its ability to perform well on new, unseen data.

There are several types as follow:

1. K-fold cross-validation
2. Holdout cross-validation
3. Stratified k-fold cross-validation
4. Leave-P-out cross validation
5. Leave one out
6. Monte Carlo Cross Validation
7. Time Series

# Code and Output:

1. K-fold cross-validation

# Code -

from sklearn.datasets import load\_iris

from sklearn.model\_selection import cross\_val\_score, KFold from sklearn.linear\_model import LogisticRegression iris=load\_iris()

X=iris.data Y=iris.target lr=LogisticRegression()

k\_fold= KFold (n\_splits=7)

score = cross\_val\_score(lr,X,Y,cv=k\_fold) print("Cross Validation Scores: {}".format(score))

print("Average Cross Validation {}".format(score.mean()))

# Output -

Cross Validation Scores: [1. 1. 1.

0.80952381 0.95238095 0.85714286

0.9047619 ]

Average Cross Validation 0.9319727891156463

1. Holdout cross-validation

# Code -

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression from sklearn.metrics import accuracy\_score

Irls= load\_iris() X=iris.data Y=iris.target Ir=LogisticRegression()

x\_train, x\_test, y\_train , y\_test=train\_test\_split(X, Y, test\_size=0.25, random\_state=42)

lr.fit(x\_train,y\_train) results=lr.predict(x\_test) print ("Training

accuracy:{}".format(accuracy\_score(lr.predict(x\_train),y\_train))) print("Testing accuracy:{}".format(accuracy\_score(results,y\_test)))

# Output -

Training accuracy:0.9642857142857143 Testing accuracy:1.0

1. Stratified k-fold cross-validation

# Code -

from sklearn.model\_selection import ShuffleSplit,cross\_val\_score from sklearn.datasets import load\_iris

from sklearn.linear\_model import LogisticRegression Ir=LogisticRegression() shuffle\_split\_cv=ShuffleSplit(test\_size=0.3, train\_size=0.5,n\_splits=6)

scores=cross\_val\_score(lr,iris.data,iris.target,cv=shuffle\_split\_ cv)

print("Cross Validation Scores:{}".format(scores))

print("Average Cross Validation score:{}".format(scores .mean()))

# Output -

Cross Validation Scores:[0.97777778 1. 0.91111111 0.

97777778 0.95555556 0.95555556]

Average Cross Validation score:0.9629629629629629

1. Leave-P-out cross validation

# Code -

from sklearn.datasets import load\_iris

from sklearn.model\_selection import cross\_val\_score,

StratifiedKFold

from sklearn.linear\_model import LogisticRegression iris=load\_iris()

X=iris.data Y=iris.target Ir=LogisticRegression()

st\_kf = StratifiedKFold(n\_splits=3) score=cross\_val\_score(lr,X,Y,cv=st\_kf) print("Cross Validation Scores:{}".format(score))

print("Average Cross Validation score:{}".format(score.mean()))

# Output -

Cross Validation Scores:[0.98 0.96 0.98] Average Cross Validation score:0.9733333333333333

1. Leave one out

# Code -

from sklearn.model\_selection import LeavePOut,cross\_val\_score from sklearn.datasets import load\_iris

from sklearn.ensemble import RandomForestClassifier iris=load\_iris()

X=iris.data Y=iris.target lpo=LeavePOut(p=1) lpo.get\_n\_splits(X)

tree= RandomForestClassifier(n\_estimators=5,max\_depth=3,n\_jobs=- 1)

score =cross\_val\_score(tree,X,Y,cv=lpo) print("Cross Validation Scores-{}".format(score))

print("Average Cross Validation score:{}".format(score.mean()))

# Output -

Cross Validation Scores-[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.

1. 1. 1. 1. 1. 1. 1. 1. 1. 1.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. |
| 1. 1. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1. 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. |
| 0. 1. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0. 1. | 1. | 1. | 1. | 0. | 1. | 1. | 1. | 1. | 1. | 0. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. |
| 1. 1. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1. 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 0. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. |
| 1. 0. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1. 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 0. | 1. | 1. | 1. | 0. | 0. | 1. | 1. | 1. | 1. | 1. | 1. | 1. |
| 1. 1. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1. 1. | 1. | 1. | 1. | 1.] |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Average Cross Validation score:0.94

1. Monte Carlo Cross Validation

# Code –

from sklearn.datasets import load\_iris

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import LeaveOneOut, cross\_val\_score iris=load\_iris()

X=iris.data Y=iris.target

leave\_one\_out=LeaveOneOut() rfc=RandomForestClassifier(n\_estimators=7,max\_depth=4,n\_jobs=-1) score=cross\_val\_score(rfc,X,Y,cv=leave\_one\_out)

print("Cross Validation Scores:{}".format(score)) print("Average Cross Validation score:{}".format(score.mean()))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Output –** | | | | | | | | |  | | | | | | | | | | | |
| Cross Validation Scores:[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. | | | | | | | | | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. |
| 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. | | | | | | | | | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. |
| 1. 1. | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| 1. 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 0. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. |
| 0. 1. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1. 1. | 1. | 1. | 1. | 0. | 1. | 1. | 1. | 1. | 1. | 0. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. |
| 1. 1. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1. 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 0. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. |
| 1. 0. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1. 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 1. | 0. | 0. | 1. | 1. | 1. | 0. | 1. | 1. | 1. |
| 1. 1. |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1. 1. | 1. | 1. | 1. | 1.] |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Average Cross Validation score:0.94

1. Time Series

# Code –

import numpy as np

from sklearn.model\_selection import TimeSeriesSplit

X = np.array([[1,2],[3,4],[1,2],[3,4],[1,2],[3,4],[1,1],[2,2]]) Y = np.array([1,2,3,4,5,6,7,8])

tscv = TimeSeriesSplit(n\_splits=7) print(tscv)

i=1

for train\_index, test\_index in tscv.split(X): print(f"TRAIN {i}:",train\_index,"TEST:",test\_index) X\_train, X\_test = X[train\_index], X[test\_index] y\_train, y\_test =Y[train\_index], Y[test\_index] i+=1

# Output-

TimeSeriesSplit(gap=0, max\_train\_size=None, n\_splits=7, test\_size=No ne)

TRAIN 1: [0] TEST: [1]

TRAIN 2: [0 1] TEST: [2]

TRAIN 3: [0 1 2] TEST: [3]

TRAIN 4: [0 1 2 3] TEST: [4]

TRAIN 5: [0 1 2 3 4] TEST: [5]

TRAIN 6: [0 1 2 3 4 5] TEST: [6]

TRAIN 7: [0 1 2 3 4 5 6] TEST: [7]

**Practical No. 2 Aim:** To perform the K- Nearest Neigbhors Algorithm

# Theory:

K-Nearest Neighbours is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection.

Distance metrics used in KNN

* Euclidean Distance
* Manhattan Distance
* Minkowski Distance.

How to choose the value of k for KNN Algorithm?

The value of k is very crucial in the KNN algorithm to define the number of neighbors in the algorithm. The value of k in the k-nearest neighbors (k-NN) algorithm should be chosen based on the input data. If the input data has more outliers or noise, a higher value of k would be better. It is recommended to choose an odd value for k to avoid ties in classification. Cross-validation methods can help in selecting the best k value for the given dataset.

# Code & Output:

**Code -**

import pandas as pd import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

from sklearn.neighbors import KneighborsClassifier from sklearn.model\_selection import train\_test\_split

x,y=make\_blobs(n\_samples=500,n\_features=2,centers=4,cluster\_std=1

.5, random\_state=4) plt.style.use(“seaborn”) plt.figure(figsize=(10,10))

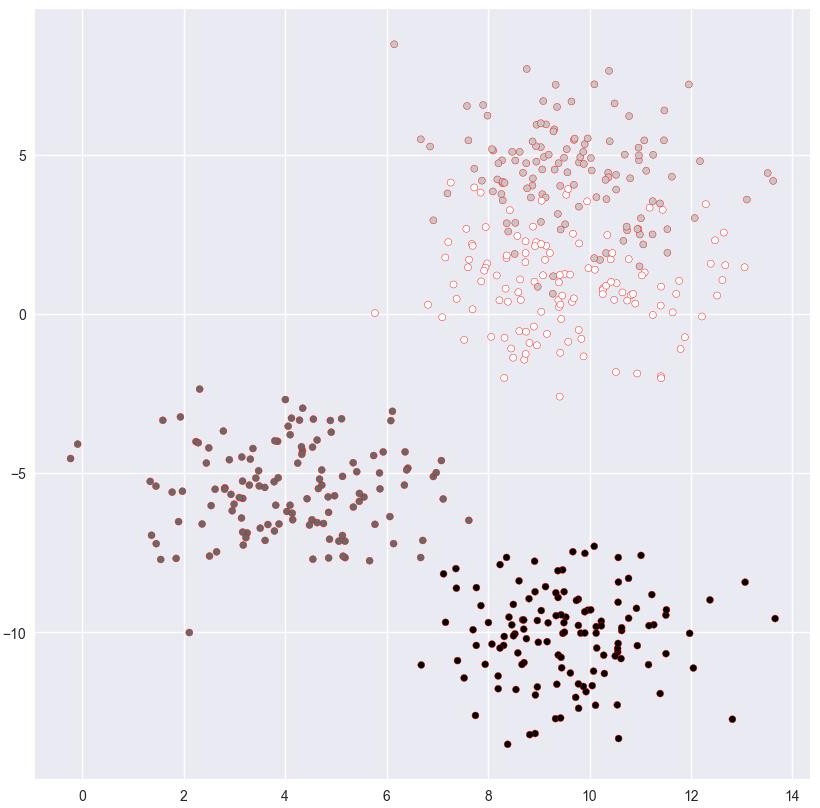
plt.scatter(x[:,0],x[:,1],c=y,marker=’.’,s=100,edgecolor=’red’) plt.show()

x\_train, x\_test, y\_train, y\_test=train\_test\_split(x,y,random\_state=0) knn5=KneighborsClassifier(n\_neighbors=5) knn1=KneighborsClassifier(n\_neighbors=1) knn5.fit(x\_train,y\_train) knn1.fit(x\_train,y\_train) y\_pred\_5=knn5.predict(x\_test) y\_pred\_1=knn1.predict(x\_test)

from sklearn.metrics import accuracy\_score

print(“Accuracy score with 5: “,accuracy\_score(y\_test,y\_pred\_5)\*100) print(“Accuracy score with 1: “,accuracy\_score(y\_test,y\_pred\_1)\*100)

|  |  |
| --- | --- |
| **Output -** |  |
| Accuracy score with 5: | 93.60000000000001 |
| Accuracy score with 1: | 90.4 |



**Practical No. 3 Aim:** To perform the simple linear regression

# Theory:

Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent features. When the number of the independent feature, is 1 then it is known as Univariate Linear regression, and in the case of more than one feature, it is known as multivariate linear regression. The goal of the algorithm is to find the best linear equation that can predict the value of the dependent variable based on the independent variables. The equation provides a straight line that represents the relationship between the dependent and independent variables. The slope of the line indicates how much the dependent variable changes for a unit change in the independent variable(s).

# Code & Output:

**Code**-

import numpy as np

from sklearn.linear\_model import LinearRegression x=np.array([5,15,25,35,45,55]).reshape((-1,1)) y=np.array([5, 20, 14, 32,22,38])

model=LinearRegression() model.fit(x,y) r\_sq=model.score(x,y)

print (f” coefficient of determination: {r\_sq}”) print (f”intercept: {model.intercept\_}”)

print (f”scope: {model.coef\_}”) y\_pred=model.predict(x)

print (f”predicted response: \n{y\_pred}”) y\_pred=model.intercept\_ + model.coef\_\*x print (f”predicted response: \n{y\_pred}”)

# Output -

coefficient of determination: 0.7158756137479542 intercept: 5.633333333333329

scope: [0.54] predicted response:

[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 3

5.33333333]

predicted response:

[[ 8.33333333][13.73333333][19.13333333]

[24.53333333][29.93333333][35.33333333]]

**Practical No. 4 Aim**: To perfom the multiple linear regression

# Code & Output:

**Code** -

import numpy as np

from sklearn.linear\_model import LinearRegression x=[[0,1],[5,1],[15,2],[25,5],[35,11],[45,15],[55,18]] y=[4,5,20,14,32,22,38]

x,y = np.array(x),np.array(y) model= LinearRegression().fit(x,y) r\_sq = model.score(x,y)

print(f"Coeffiecent of determinantion : {r\_sq}") print(f"Intercept : {model.intercept\_}") print(f"Score : {model.coef\_}")

y\_pred = model.predict(x) print(f"Predicted response : \t{y\_pred}")

# Output-

Coeffiecent of determinantion : 0.7979631287564747 Intercept : 4.355292792792792

Score : [ 0.75056306 -0.57713964]

Predicted response : [ 3.77815315 7.53096847 14.45945946 20.23367117 2

4.27646396 29.47353604

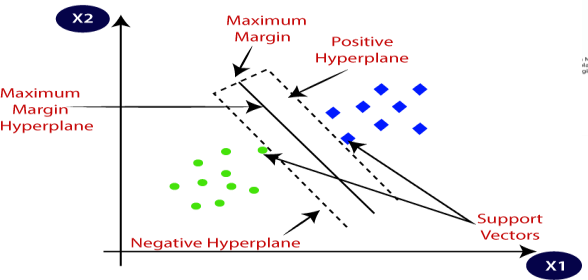
35.24774775]

# Practical No. 5

**Aim**: To perform the SVM algorithm on given dataset (Social Networks Ads)

# Theory:

Support Vector Machine(SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well it’s best suited for classification. The objective of the 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.



Use Of kernel function

The SVM kernel is a function that takes low-dimensional input space and transforms it into higher-dimensional space, ie it converts nonseparable problems to separable problems. It is mostly useful in non-linear separation problems.

# Code & Output:

**Code** –

import numpy as np

import matplotlib.pyplot as plt import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler from sklearn.svm import SVC

dataset=pd.read\_csv ('Social\_Network\_ads.csv') x=dataset.iloc[:,[2,3]].values y=dataset.iloc[:, 4].values

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y, test\_size=0.25, random\_state=0)

sc=StandardScaler() x\_train=sc.fit\_transform(x\_train) x\_test=sc. transform(x\_test)

classifier=SVC (kernel='rbf', random\_state=0) classifier.fit (x\_train,y\_train) y\_pred=classifier.predict(x\_test)

from sklearn.metrics import confusion\_matrix, accuracy\_score cm =confusion\_matrix(y\_test,y\_pred)

print (cm) accuracy\_score(y\_test,y\_pred)

# Output -

[[64 4]

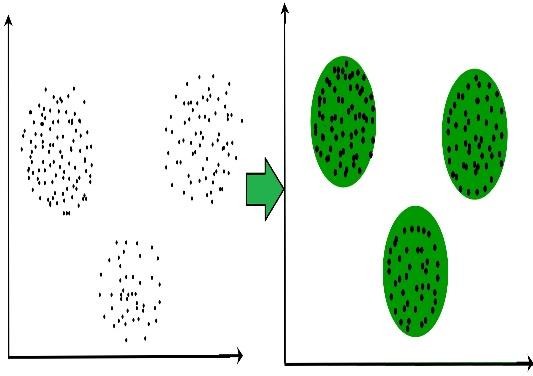
[ 3 29]]

0.93

# Practical No. 6

**Aim**: To perform K Means Clustering algorithm on income dataset.

# Theory:

Clustering:- Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups.

K-Means Clustering:- It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way

that each dataset belongs only one group that has similar properties. It select K centroids, assign each data point to their closest centroid, for K clusters and then calculate the variance and

place a new centroid of each cluster.

# Code & Output:

**Code** -

import numpy as np import pandas as pd

from sklearn.preprocessing import StandardScaler from sklearn.preprocessing import MinMaxScaler from sklearn.cluster import KMeans

from sklearn.cluster import AgglomerativeClustering df=pd.read\_csv("/content/Mall\_Customers.csv") df.head()

df=df.drop(['CustomerID'],axis=1)

df.rename(columns = {'Annual Income (k$)' : 'Income(k$/yr)', 'Spending Score (1-100)' : 'SpendScore(1-100)'}, inplace=True)

X = df[['Income(k$/yr)', 'SpendScore(1-100)']].copy() X.head()

scaler = StandardScaler()

X\_kmeans = scaler.fit\_transform(X)

wcss = []

for cluster in range(1,11):

kmeans = KMeans(n\_clusters = cluster, init = 'k-means++', random\_state = 42)

kmeans.fit(X\_kmeans) wcss.append(kmeans.inertia\_)

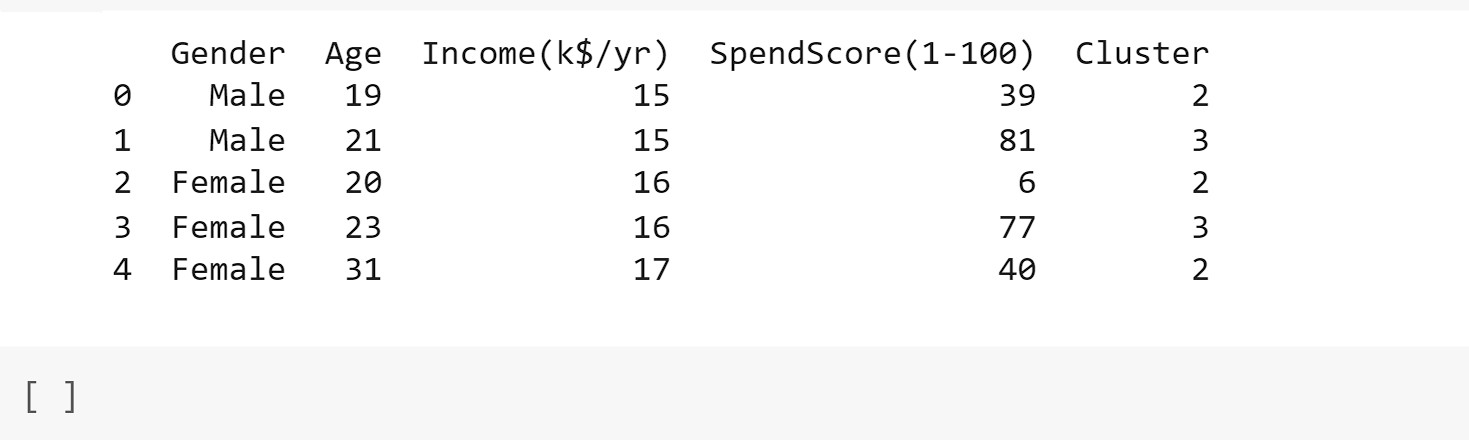
clusters = KMeans(n\_clusters = 5, init = 'k-means++', random\_state = 42)

y\_kmeans = clusters.fit\_predict(X\_kmeans) y\_kmeans

df\_Kmeans = df.copy() ## making a copy of original dataframe

df\_Kmeans['Cluster'] = y\_kmeans ## appending the cluster column print(df\_Kmeans.head(5))

# Output -



**Practical No. 7 Aim:** Stock Price Prediction using MLPClassifier.

# Theory:

Multilayer Perceptron (MLP) is the most fundamental type of neural network architecture when compared to other major types such as Convolutional Neural Network (CNN),

Recurrent Neural Network (RNN), Autoencoder (AE) and Generative Adversarial Network (GAN).

In an MLP, perceptrons (neurons) are stacked in multiple layers. Every node on each layer is connected to all other nodes on the next layer. There is no connection between nodes within a single layer. An MLP is a Fully (Densely) Connected Neural Network (FCNN).

# Code & Output:

**Code** –

import numpy as np import pandas as pd

data\_train = pd.read\_csv("/content/NFLX.csv") data\_train.head()

isna\_df = data\_train.isnull().sum() isna\_df[isna\_df > 0]

print("Max:", isna\_df.max()) print(isna\_df[isna\_df == isna\_df.max()]) isna\_df.max()/data\_train.size X=data\_train.drop(['Date','Adj Close'],axis=1) y=data\_train['Adj Close']

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.neural\_network import MLPRegressor

regr = MLPRegressor(random\_state=1, max\_iter=500).fit(x\_train, y\_train)

regr.predict(x\_test[:2])

# Output:

array([385.00662251, 548.14237429])

# Practical No. 8

**Aim:** To perform LSTM algorithm to predict stock price.

# Theory:

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. This is a behavior required in complex problem domains like machine translation, speech recognition, and more. LSTMs are a complex area of deep learning. It can be hard to get your hands around what LSTMs are, and how terms like bidirectional and sequence-to-sequence relate to the field.

The Long Short Term Memory architecture was motivated by an analysis of error flow in existing RNNs which found that long time lags were inaccessible to existing architectures, because backpropagated error either blows up or decays exponentially. An LSTM layer

consists of a set of recurrently connected blocks, known as memory blocks. These blocks can be thought of as a differentiable version of the memory chips in a digital computer.

# Code & Output:

**Code** -

# Import Required Libraries import math

from keras.models import Sequential

from keras.layers import Dense, LSTM, Dropout from sklearn.preprocessing import MinMaxScaler from sklearn.metrics import mean\_squared\_error import pandas as pd

import numpy as np

# importing the training set

data\_train = pd.read\_csv("/content/Google\_Stock\_Price\_Train.csv") data\_train.head(10)

# Preprocessing Data

data = data\_train.loc[:,["Open"]].values print(data)

# Reshape

data = data.reshape(-1,1) data = data.astype("float32")

data.shape # scaling

scaler = MinMaxScaler(feature\_range=(0,1)) data = scaler.fit\_transform(data)

# train test split

train\_size = int(len(data) \* 0.5) test\_size = len(data) - train\_size train = data[0:train\_size, :]

test = data[train\_size:len(data), :]

print("train size: {}, test size: {} ".format(len(train), len(test)))

# timesteps timesteps = 5 x\_data = [] y\_data = []

for i in range(len(train) - timesteps -1): a = train[i:(i+timesteps), 0] x\_data.append(a)

y\_data.append(train[i + timesteps, 0]) x\_train = np.array(x\_data)

y\_train = np.array(y\_data) x\_data = []

y\_data = []

for i in range(len(test) - timesteps -1): a = test[i:(i+timesteps), 0] x\_data.append(a)

y\_data.append(test[i + timesteps, 0]) x\_test = np.array(x\_data)

y\_test = np.array(y\_data)

x\_train = np.reshape(x\_train, (x\_train.shape[0], 1, x\_train.shape[1]))

x\_test = np.reshape(x\_test, (x\_test.shape[0], 1 , x\_test.shape[1])) ## Create LSTM model

model = Sequential()

model.add(LSTM(100, input\_shape = (1, timesteps))) # 50 LSTM neuron (block)

model.add(Dense(units=1))

model.compile(loss = "mean\_squared\_error", optimizer= "adam") model.fit(x\_train, y\_train, epochs = 50, batch\_size = 1)

# predictions

train\_pred = model.predict(x\_train) test\_pred = model.predict(x\_test) # invert predictions

train\_pred = scaler.inverse\_transform(train\_pred) y\_train = scaler.inverse\_transform([y\_train]) test\_pred = scaler.inverse\_transform(test\_pred) y\_test = scaler.inverse\_transform([y\_test])

# calculate root mean squared error

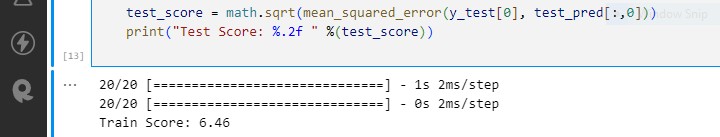
train\_score = math.sqrt(mean\_squared\_error(y\_train[0], train\_pred[:,0]))

print("Train Score: %.2f " %(train\_score))

test\_score = math.sqrt(mean\_squared\_error(y\_test[0], test\_pred[:,0]))

print("Test Score: %.2f " %(test\_score))

# Output:



**Practical No.9**

**Aim:** Find the information gained from each column for the given dataset.

# Theory:

Entropy is a scientific concept as well as a measurable physical property that is most commonly associated with a state of disorder, randomness, or uncertainty.

Gini Index or Impurity measures the probability for a random instance being misclassified when chosen randomly. The lower the Gini Index, the better the lower the likelihood of misclassification.

# Code & Output:

**Code** -

# Import libraries of python import pandas as pd

import os import math

# Read the CSV dataset using pandas df=pd.read\_csv('data.csv')

# Entropy

def entropy(labels):

# Initialize the entropy to 0. entropy = 0.0

# Iterate over all possible labels. for label in set(labels):

# Calculate the probability of the label. probability = labels.count(label) / len(labels)

# Calculate the contribution of the label to the entropy. entropy -= probability \* math.log2(probability)

# Return the entropy. return entropy

# Print entropy for each columns of dataset for name in columns:

print(entropy(list(df[name])))

# Gini Index

def gini\_index(labels):

# Initialize the Gini impurity to 1. gini = 1.0

# Iterate over all possible labels. for label in set(labels):

# Calculate the probability of the label. probability = labels.count(label) / len(labels)

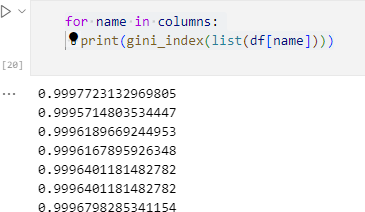
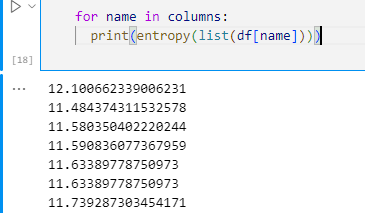
# Calculate the contribution of the label to the Gini impurity. gini -= probability \*\* 2

# Return the Gini impurity. return gini

# Print Gini index for each columns of dataset for name in columns:

print(gini\_index(list(df[name])))

# Output:



**Practical No. 10 Aim**: To predict Stock Price using SVR algorithm.

# Theory:

SVR gives us the flexibility to define how much error is acceptable in our model and will find an appropriate line (or hyperplane in higher dimensions) to fit the data.

In contrast to OLS, the objective function of SVR is to minimize the coefficients — more specifically, the l2-norm of the coefficient vector — not the squared error.

# Code & Output:

**Code** -

# Import needed libraries import pandas as pd import numpy as np

from sklearn.svm import SVR

from sklearn.model\_selection import train\_test\_split # Load the data

df = pd.read\_csv("EW-MAX.csv") df.head()

# Prepare the data

X = df[["Volume", "High", "Low", "Close","Adj\_Close"]] y = df["Open"]

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25)

# Create the SVR model svr = SVR(kernel='rbf') # Train the model svr.fit(X\_train, y\_train) # Predict on the test set

y\_pred = svr.predict(X\_test) # Evaluate the model

print(svr.score(X\_test, y\_test))

# Output:

