Dataset

Time	Weather	Temperature	Humidity	Goes
Morning	Sunny	Warm	mild	yes
Evening	Rainy	Cold	Mild	No
Morning	Sunny	Moderate	Normal	Yes
Evening	Sunny	Cold	High	yes

Output

n the attributes are :

[['morning' 'sunny' 'warm' 'mild']

['evening' 'rainy' 'cold' 'mild']

['morning' 'sunny' 'moderate' 'normal']

['evening' 'sunny' 'cold' 'high']]

n the attributes are : ['yes' 'no' 'yes' 'yes']

n the final hypothesis is : ['?' 'sunny' '?' '?']

Experiment: - 01

Date: / /2023

Aim: - Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

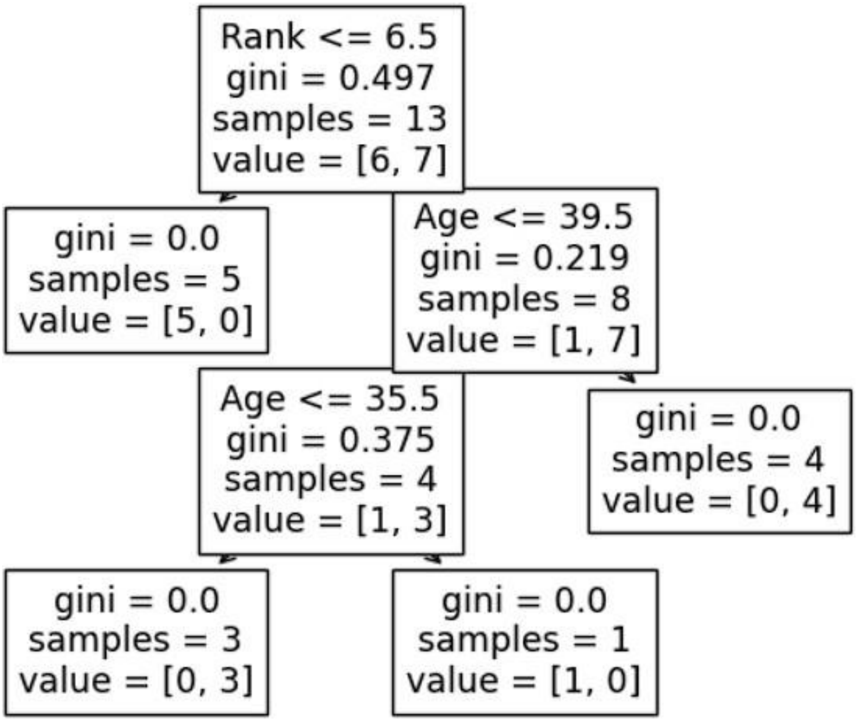
Code:-

```
import pandas as pd
import numpy as np
data = pd.read_csv("exp1.csv")
d = np.array(data)[:,:-1]
print("n the attributes are : ",d)
target = np.array(data)[:,-1]
print("n the attributes are : " , target)
def train(c,t):
    for i , val in enumerate(t):
        if val == "yes":
            specific_hypothesis = c[i].copy()
            break
    for i, val in enumerate(c):
        if t[i] == "yes":
            for x in range (len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = "?"
                else:
                    pass
    return specific_hypothesis
print("n the final hypothesis is : ", train(d,target))
```

Data Set

Age	Experience	Rank	Nationality	Go
36	10	9	UK	NO
42	12	4	USA	NO
23	4	6	N	NO
52	4	4	USA	NO
43	21	8	USA	YES
44	14	5	UK	NO
66	3	7	N	YES
35	14	9	UK	YES
52	13	7	N	YES
35	5	9	N	YES
24	3	5	USA	NO
18	3	7	UK	YES
45	9	9	UK	YES

Output



Experiment: - 02

Date: / /2023

Aim: **-** Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Code: -

```
import sys
import pandas as pd
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
df = pd.read_csv("DTree.csv")
print(df)
d = {'UK' : 0, 'USA' : 1, 'N' : 2}
df ['Nationality'] = df ['Nationality'].map(d)
d = {'YES':1, 'NO':0}
df['Go'] = df['Go'].map(d)
print(df)
features = ['Age','Experince', 'Rank','Nationality']
x = df[features]
y = df['Go']
dtree = DecisionTreeClassifier()
dtree = dtree.fit(x,y)
tree.plot_tree(dtree, feature_names = features)
```

Data set

Years Experience	Salary
1.1	39343
1.3	46205
1.5	37731
2	43525
2.2	39891
2.9	56642
3	60150
3.2	54445
3.2	64445
3.7	57189
3.9	63218
4	55794
4	56957
4.1	57081
4.5	61111
4.9	67938
5.1	66029
5.3	83088
5.9	81363
6	93940
6.8	91738
7.1	98273
7.9	101302
8.2	113812
8.7	109431
9	105582
9.5	116969
9.6	112635
10.3	122391
10.5	121872

Experiment: - 03

Date: / /2023

Aim: - To solve the real-world problems using the following machine learning methods:

(a). Linear regression

(b). Logistic regression

Code: -

(a). Linear regression

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

dataset = pd.read_csv('salary_data.csv')

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

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

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 1/3, random_state =
0)

rom sklearn.linear_model import LinearRegression

regressor = LinearRegression()

regressor.fit(x_train, y_train)

y_pred = regressor.predict(x_test)

y_pred

plt.scatter(x_train, y_train, color = 'blue')

plt.plot(x_train, regressor.predict(x_train), color = 'green')

plt.title('salary vs experiance (training set)')
```

Output

(a).Linear regression

Output

(b).Logistic Regression

Array ([0])

```
plt.xlabel('year of experiance')
plt.ylabel('salary')
plt.show()
```

Code: -

(b). Logistic regression

```
import pandas as pd
import numpy as np
from sklearn import linear_model
x = np.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)

y = np.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear_model.LogisticRegression()
logr.fit(x,y)

predict = logr.predict(np.array([3.46]).reshape(-1,1))
predict
```

Output

MSE: 22.418

Bias: 20.744

Variance: 1.674

Experiment: - 04

Date: / /2023

Aim: - Develop a program for Bias, Variance, Remove duplicates, Cross Validation

Theory:-

The performance of a machine learning model can be characterized in terms of the bias and the variance of the model.

A model with high bias makes strong assumptions about the form of the unknown underlying function that maps inputs to outputs in the dataset, such as linear regression. A model with high variance is highly dependent upon the specifics of the training dataset, such as unpruned decision trees. We desire models with low bias and low variance, although there is often a trade-off between these two concerns.

The bias-variance trade-off is a useful conceptualization for selecting and configuring models, although generally cannot be computed directly as it requires full knowledge of the problem domain, which we do not have. Nevertheless, in some cases, we can estimate the error of a model and divide the error down into bias and variance components, which may provide insight into a given model's behavior.

Code: -

```
# estimate the bias and variance for a regression model
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from mlxtend.evaluate import bias_variance_decomp
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
dataframe = read_csv(url, header=None)
# separate into inputs and outputs
data = dataframe.values
X, y = data[:, :-1], data[:, -1]
# split the data
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)

# define the model

model = LinearRegression()

# estimate bias and variance

mse, bias, var = bias_variance_decomp(model, X_train, y_train, X_test, y_test, loss='mse',
num_rounds=200, random_seed=1)

# summarize results

print('MSE: %.3f' % mse)

print('Bias: %.3f' % bias)

print('Variance: %.3f' % var)
```

Output

(a).Categorical encoding

	City
0	Delhi
1	Mumbai
2	Hyderabad
3	Chennai
4	Bangalore
5	Delhi
6	Hyderabad
7	Bangalore
8	Delhi

(b).One-hot encoding

team	points	0	1	2
A	25	1.0	0.0	0.0
A	12	1.0	0.0	0.0
B	15	0.0	1.0	0.0
B	14	0.0	1.0	0.0
B	19	0.0	1.0	0.0
B	23	0.0	1.0	0.0
C	25	0.0	0.0	1.0
C	29	0.0	0.0	1.0

Experiment: - 05

Date: / /2023

Aim: -Write a program to implement (a) Categorical encoding, (b) One-hot encoding

Theory:-

(a). Categorical Encoding:-

The process of encoding categorical data into numerical data is called "categorical encoding." It involves transforming categorical variables into a numerical format suitable for machine learning models.

(b).One-hot Encoding:-

One-hot encoding is used to convert categorical variables into a format that can be readily used by machine learning algorithms. The basic idea of one-hot encoding is to create new variables that take on values 0 and 1 to represent the original categorical values.

Code: -

(a) Categorical encoding

```
import category encoders as ce

import pandas as pd

data=pd.DataFrame({'City':['Delhi','Mumbai','Hydrabad','Chennai','Bangalore','Delhi','Hydrabad','Ba
ngalore','Delhi']})

encoder=ce.OneHotEncoder(cols='City',handle_unknown='return_nan',return_df=True,use_cat_na
mes=True)

data
```

Code:

(b).One-hot encoding

```
import pandas as pd

df = pd.DataFrame({'team': ['A', 'A', 'B', 'B', 'B', 'B', 'C', 'C'],'points': [25, 12, 15, 14, 19, 23, 25, 29]})

print(df)

from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(handle_unknown='ignore')

encoder_df = pd.DataFrame(encoder.fit_transform(df[['team']]).toarray())

final_df = df.join(encoder_df)

print(final_df)
```

Output

sepal-length sepal-weidth petal-lenght petal-width

[[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

… … … … … … … … …

[6.5 3. 5.2 2.]

[6.2 3.4 5.4 2.3]

[5.9 3. 5.1 1.8]]

class: 0-Iris-sentosa, 1-iris-versicolour, 2-iris-virginica

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 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 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2]

KNeighborsClassifier()

```
Confusion Matrix
[[17  0  0]
 [ 0 15  2]
 [ 0  0 11]]
accuracy metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       1.00      0.88      0.94        17
           2       0.85      1.00      0.92        11

    accuracy                           0.96        45
   macro avg       0.95      0.96      0.95        45
weighted avg       0.96      0.96      0.96        45
```

Experiment: - 06

Date: / /2023

Aim: -

To write a program to implement K-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong prediction.

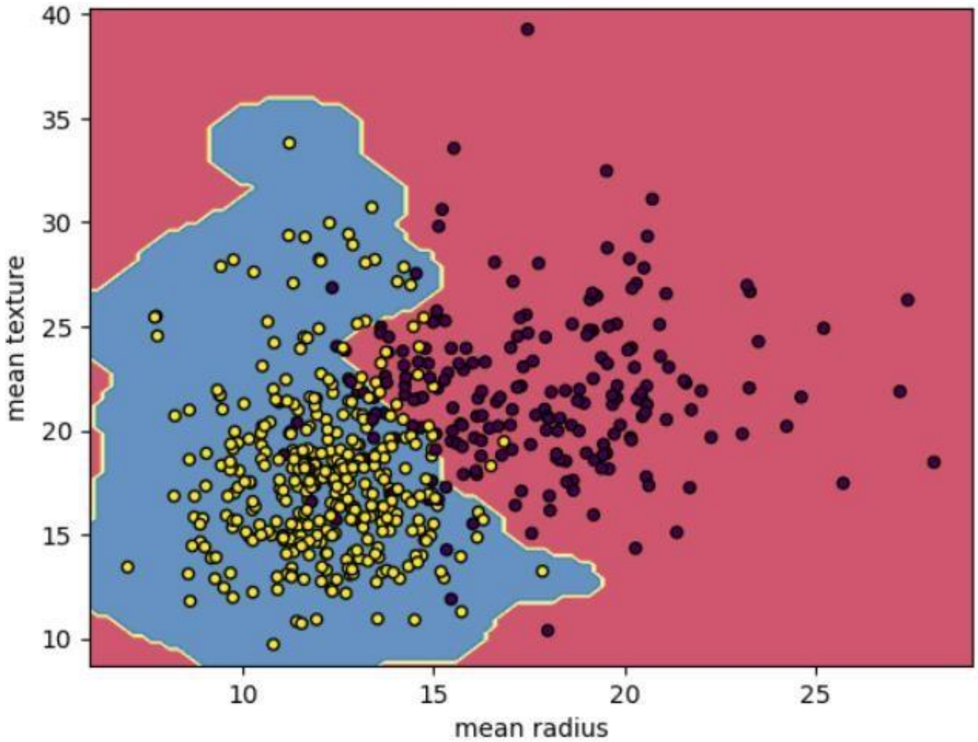
Theory: -

Support Vector Machine (SVM) is a powerful machine learning algorithm used for linear or nonlinear classification, regression, and even outlier detection tasks. SVMs can be used for a variety of tasks, such as text classification, image classification, spam detection, handwriting identification, gene expression analysis, face detection, and anomaly detection.SVM algorithms are very effective as we try to find the maximum separating hyperplane between the different classes available in the target feature.Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression.The main objective of the SVM algorithm is to find the optimal hyperplane in an N-dimensional space that can separate the data points in different classes in the feature space. The hyperplane tries that the margin between the closest points of different classes should be as maximum as possible.

Code: -

```
from sklearn. model _ selection import train_test_split

from sklearn.neighbors import KNeighborsC1assifier

from sklearn. metrics import classification_report, confusion_matrix

from sklearn import datasets

iris = datasets.load_iris()

x = iris.data

y= iris. target

print ( ‘sepal-length’ , ‘sepal-width’, ‘petal-length’, ‘petal-width’ )

print(x)

print(‘ class: 0-lris-sentosa, 1- Iris-Versicolour, 2- Iris-Virginica’ )

print (y)
```

Output



Experiment: - 07

Date: / /2023

Aim: -

Write a program to implement support vector machines and principle component analysis.

Code:

```
from sklearn.datasets import load_breast_cancer

import matplotlib.pyplot as plt

from sklearn.inspection import DecisionBoundaryDisplay

from sklearn.svm import SVC

cancer = load_breast_cancer()

X = cancer.data[:,:2]

y = cancer.target

svm = SVC(kernel = "rbf", gamma = 0.5, C= 1.0)

svm.fit(X,y)

DecisionBoundaryDisplay.from_estimator(svm,X,response_method = "predict",cmap = plt.cm.Spectral,alpha = 0.8, xlabel = cancer.feature_names[0], ylabel = cancer.feature_names[1],)

plt.scatter(X[:,0],X[:,1],c = y, s = 20, edgecolors = "k")

plt.show()
```

Dataset

	Type	Alcohol	Malic	Ash	Alcalinity	Magnesium	Phenols	Flavanoids	Nonflavanoids	Proanthocyanins	Color	Hue	Dilution	Proline
0	1	14.23	1.71	2.43	15.6	127	2.8	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.2	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.4	1050
2	1	13.16	2.36	2.67	18.6	101	2.8	3.24	0.3	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.5	16.8	113	3.85	3.49	0.24	2.18	7.8	0.86	3.45	1480
4	1	13.24	2.59	2.87	21	118	2.8	2.69	0.39	1.82	4.32	1.04	2.93	735

Experiment: - 08

Date: / /2023

Aim: Write a program to implement principle component analysis

Code:

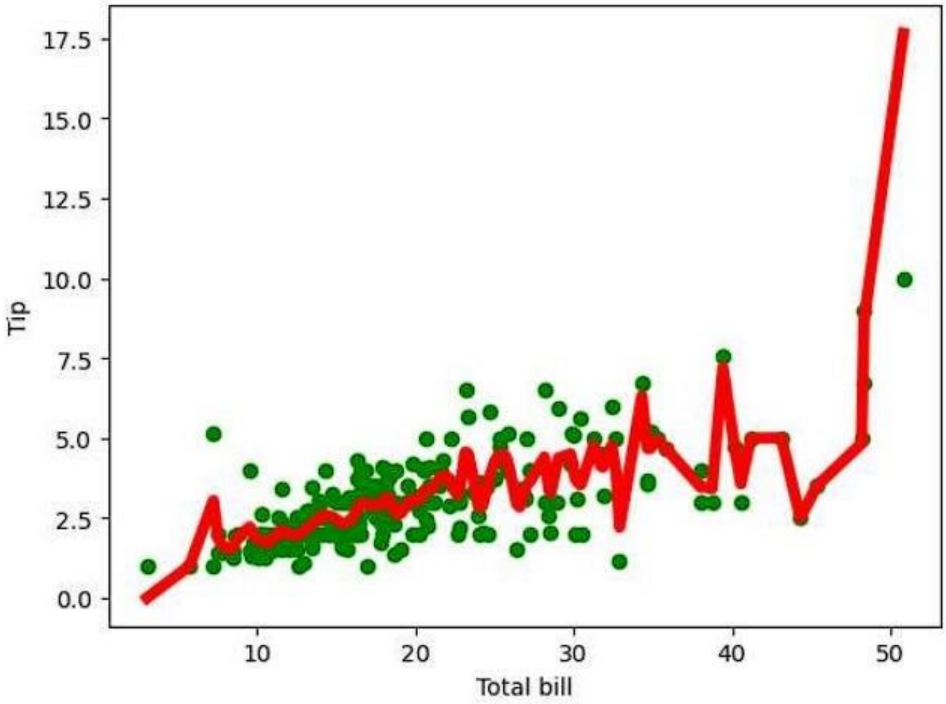
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
import seaborn as sns
from sklearn.decomposition import PCA
import scipy.cluster.hierarchy as sch
df = pd.read_csv("7_wine.csv")
df.head()
y =  df['Type']
df1 = df.iloc[:,1:]
df1.head()
pca = PCA(n_components=13)
principalComponents = pca.fit_transform(df_norm)
PC = range(1, pca.n_components_+1)
plt.bar(PC, pca.explained_variance_ratio_, color='blue')
plt.xlabel('Principal Components')
```

Output

0	1	2	3	4	5	6	7	8	9	10	11	12	
0	3.316751	-1.44346	-0.16574	-0.21563	0.693043	-0.22388	0.596427	0.065139	0.641443	1.020956	-0.45156	0.54081	-0.06624
1	2.209465	0.333393	-2.02646	-0.29136	-0.25766	-0.92712	0.053776	1.024416	-0.30885	0.159701	-0.14266	0.388238	0.003637
2	2.51674	-1.03115	0.982819	0.724902	-0.25103	0.549276	0.424205	-0.34422	-1.17783	0.113361	-0.28667	0.000584	0.021717
3	3.757066	-2.75637	-0.17619	0.567983	-0.31184	0.114431	-0.38334	0.643593	0.052544	0.239413	0.759584	-0.24202	-0.36948
4	1.008908	-0.86983	2.026688	-0.40977	0.298458	-0.40652	0.444074	0.4167	0.326819	-0.07837	-0.52595	-0.21666	-0.07936
...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	-3.37052	-2.21629	-0.34257	1.058527	-0.57416	-1.10879	0.958416	-0.1461	-0.0225	-0.30412	0.139228	0.170786	-0.11443
174	-2.60196	-1.75723	0.207581	0.349496	0.255063	-0.02647	0.146894	-0.55243	-0.09797	-0.20606	0.258198	-0.27943	-0.18737
175	-2.67784	-2.7609	-0.94094	0.312035	1.271355	0.273068	0.679235	0.047024	0.001222	-0.248	0.512492	0.698766	0.072078

```
plt.ylabel('Variance %')

plt.xticks(PC)

pca.explained_variance_ratio_

PCA_components = pd.DataFrame(principalComponents
```

Output



Experiment: - 09

Date: / /2023

Aim: -

Implement the non-parametric Locally weighted regression algorithm in order to fit data point. Select appropriate data set for your experiment and draw graphs.

Code:

```
from numpy import *

from os import listdir

import matplotlib

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np1

import numpy.linalg as np

from scipy.stats.stats import pearsonr


def kernel(point,xmat, k):

    m,n = np1.shape(xmat)

    weights = np1.mat(np1.eye((m)))

    for j in range(m):

        diff = point - X[j]

        weights[j,j] = np1.exp(diff*diff.T/(-2.0*k**2))

    return weights


def localWeight(point,xmat,ymat,k):

    wei = kernel(point,xmat,k)

    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
```

```
 return W

def localWeightRegression(xmat,ymat,k):
    m,n = np1.shape(xmat)
    ypred = np1.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred
data = pd.read_csv('tips.csv')
bill = np1.array(data.total_bill)
tip = np1.array(data.tip)

mbill = np1.mat(bill)
mtip = np1.mat(tip) # mat is used to convert to n dimesiona to 2 dimensional array form
m= np1.shape(mbill)[1]
# print(m) 244 data is stored in m
one = np1.mat(np1.ones(m))
X= np1.hstack((one.T,mbill.T)) # create a stack of bill from ONE
#print(X)
#set k here
ypred = localWeightRegression(X,mtip,0.3)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='green')
```

```
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)

plt.xlabel('Total bill')

plt.ylabel('Tip')

plt.show();
```