Index

Vision and Mission of the Institute

Vision

To induce higher planes of learning by imparting technical education with

- International standards
- Applied research
- Creative Ability
- Value based instruction and to emerge as a premiere institute

Mission

Achieving academic excellence by providing globally acceptable technical education by forecasting technology through

- Innovative Research and development
- Industry Institute Interaction
- Empowered Manpower

Vision and Mission of the Department

Vision

To be a department with high repute and focused on quality education

Mission

M1	To provide an environment for the development of professionals with knowledge and skills
M2	To promote innovative learning
M3	To promote innovative ideas towards society
M4	To foster trainings with institutional collaborations
M5	To involve in the development of software applications for societal needs

Program Educational Objectives (PEOs)

PEO1	Graduates will be skilled in Mathematics, Science & modern engineering tools to solve real life problems.
PEO2	Excel in the IT industry with the attained knowledge and skills or pursue higher studies to acquire emerging technologies and become an entrepreneur.
PEO3	Accomplish a successful career and nurture as a responsible professional with ethics and human values.

Program Specific Outcomes (PSOs)

PSO1	Apply mathematical foundations, algorithmic and latest computing tools and techniques to design computer-based systems to solve engineering problems
PSO2	Apply knowledge of Engineering and develop software-based applications for research and development in the areas of relevance under realistic constraints.
PSO3	Apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product.

Program Outcomes (POs)

Engineering Graduates will be able to:

1. **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem Analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

MACHINE LEARNING USING PYTHON LAB

III B. Tech (IT) II SEMESTER (R20)

CO – Blooms Taxonomy Mapping

S.No	CO No.	Course Outcomes	Blooms Taxonomy
1	CO1	Implement procedures for the Machine Learning algorithms.	Application
2	CO2	Design and Develop Python programs for various Learning algorithms.	Application
3	CO3	Apply appropriate data sets to the Machine Learning algorithms.	Application
4	CO4	Develop Machine Learning algorithms to solve real world problems.	Application
5	CO5	Design and carry out an experimental evaluation of algorithms and state the conclusions.	Application
6	CO6	Apply Deep Learning to solve complex problems.	Application

LIST OF EXPERIMENTS

Experiment-1:
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.

Experiment-2:
For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Experiment-3:
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.

Experiment-4:
Exercises to solve the real-world problems using the following machine learning methods: a) Linear Regression b) Logistic Regression c) Binary Classifier

Experiment-5:
Develop a program for Bias, Variance, Remove duplicates, Cross Validation

Experiment-6:
Write a program to implement Categorical Encoding, One-hot Encoding

Experiment-7:
Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

Experiment-8:
Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

Experiment-9:
Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Experiment-10:
Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

Experiment-11:
Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

Experiment-12:
Exploratory Data Analysis for Classification using Pandas or Matplotlib.

Experiment-13:
Write a Python program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

Experiment-14:
Write a program to Implement Support Vector Machines and Principle Component Analysis

Experiment-15:
Write a program to Implement Principle Component Analysis.

Experiment-1:

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.

FIND-S Algorithm

The find-S algorithm is a machine learning concept algorithm. The find-S technique identifies the hypothesis that best matches by considering only positive cases. In machine learning, a hypothesis H is a function that best describes the target in supervised learning.

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x

   For each attribute constraint a_i in h

   If the constraint a_i is satisfied by x

   Then do nothing

   Else

   replace a_i in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

Training Examples:

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Program

```
import csv
a = []
with open('E:/Materials/Machine Learning/ML Lab Manual/1 enjoysport.csv', 'r') as csvfile:
    for row in csv.reader(csvfile):
        a.append(row)
    print(a)

print("\n The total number of training instances are : ",len(a))
num_attribute = len(a[0])-1
print("\n The initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)

for i in range(1, len(a)):
    if a[i][num_attribute] != 'No':
        print ("\nVector {} instance is: ".format(i),a[i])
        for j in range(0, num_attribute):
            if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
                hypothesis[j] = a[i][j]
            else:
                hypothesis[j] = '?'
    print("\n The hypothesis for the training instance {} is :\n" .format(i+1),hypothesis)
print("\n The Maximally specific hypothesis for the training instance is ")
print(hypothesis)
```

Output:

```
Vector 1 instance is:  ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm',
'Same', 'Yes']

 The hypothesis for the training instance 2 is :
 ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

Vector 2 instance is:  ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same',
'Yes']

 The hypothesis for the training instance 3 is :
 ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

 The hypothesis for the training instance 4 is :
 ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']

Vector 4 instance is:  ['Sunny', 'Warm', 'High', 'Strong', 'Cool',
'Change', 'Yes']

 The hypothesis for the training instance 5 is :
 ['Sunny', 'Warm', '?', 'Strong', '?', '?']

 The Maximally specific hypothesis for the training instance is
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

Experiment-2:

For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

CANDIDATE-ELIMINATION Algorithm

This algorithm incrementally builds the version space given a hypothesis space H and a set E of examples. The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example. The candidate elimination algorithm does this by updating the general and specific boundary for each new example.

- You can consider this as an extended form of Find-S algorithm.
- Consider both positive and negative examples.
- Actually, positive examples are used here as Find-S algorithm (Basically they are generalize from the specification).
- While negative example is specified from generalize form.

The CANDIDATE-ELIMINTION algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

1. Initialize G to the set of maximally general hypotheses in H
2. Initialize S to the set of maximally specific hypotheses in H
3. For each training example d, do
    a. If d is a positive example
        i. Remove from G any hypothesis inconsistent with d
        ii. For each hypothesis s in S that is not consistent with d
            1. Remove s from S
            2. Add to S all minimal generalizations h of s such that
                a. h is consistent with d, and some member of G is more general than h
            3. Remove from S any hypothesis that is more general than another hypothesis in S
    b. If d is a negative example

i. Remove from S any hypothesis inconsistent with d

ii. For each hypothesis g in G that is not consistent with d

1. Remove g from G
2. Add to G all minimal specializations h of g such that
   a. h is consistent with d, and some member of S is more specific than h
3. Remove from G any hypothesis that is less general than another hypothesis in G

Training Examples:

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Program:

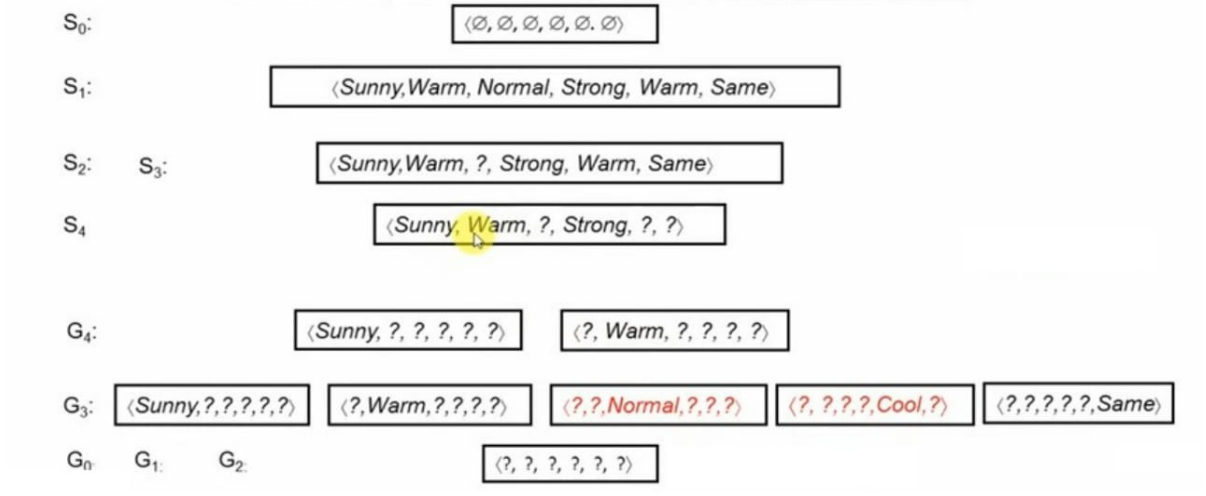
```
import numpy as np
import pandas as pd

data = pd.DataFrame(data=pd.read_csv('E:/Materials/Machine Learning/ML Lab Manual/1 enjoysport.csv'))
concepts = np.array(data.iloc[:,0:-1])
print('Concepts are\n',concepts)
target = np.array(data.iloc[:,-1])
print('Targets are\n',target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
```

```
    print("Initialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print(general_h)
    for i, h in enumerate(concepts):
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'
                #print(specific_h)
        print(specific_h)
        if target[i] == "No":
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print("Steps of Candidate Elimination Algorithm",i+1)
        print('Values of specific_h\n',specific_h)
        #print('Values of general_h\n',general_h)
    indices = [i for i, val in enumerate(general_h) if val ==['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

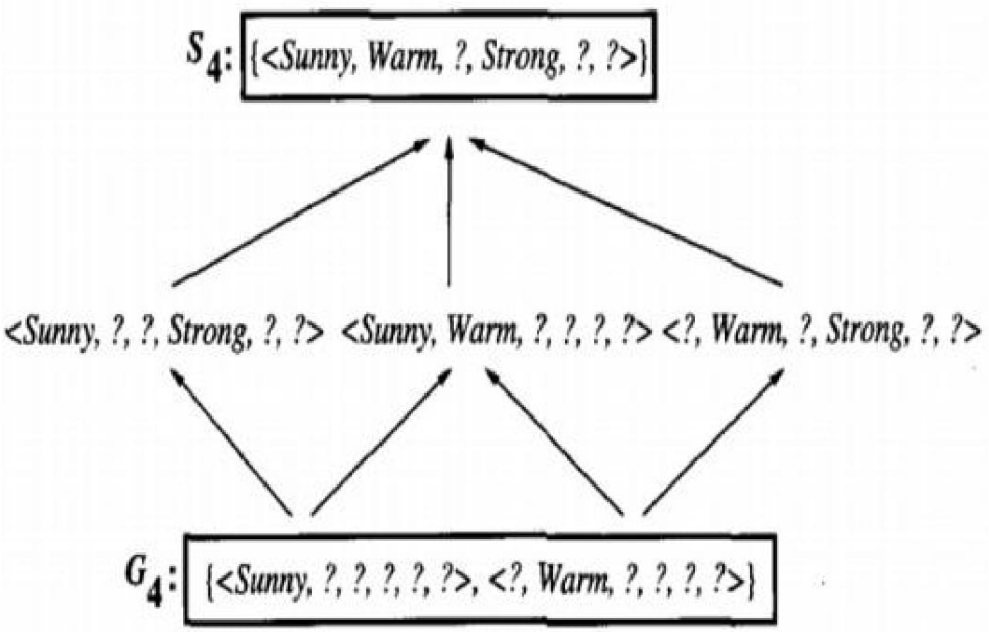
Output:

```
Initialization of specific_h and general_h
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Steps of Candidate Elimination Algorithm 1
Values of specific_h
 ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
Steps of Candidate Elimination Algorithm 2
Values of specific_h
 ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
Steps of Candidate Elimination Algorithm 3
Values of specific_h
 ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
['Sunny' 'Warm' '?' 'Strong' '?' '?']
Steps of Candidate Elimination Algorithm 4
Values of specific_h
 ['Sunny' 'Warm' '?' 'Strong' '?' '?']
Final Specific_h:
['Sunny' 'Warm' '?' 'Strong' '?' '?']
Final General_h:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes



Finally, the result is produced by synchronizing the General hypothesis and Specific hypothesis.



Experiment-3:

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.

ID3(Examples, Target_attribute, Attributes)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

- Create a Root node for the tree
- If all Examples are positive, Return the single-node tree Root, with label = +
- If all Examples are negative, Return the single-node tree Root, with label = -
- If Attributes is empty, Return the single-node tree Root, with label = most common value of Target_attribute in Examples
- Otherwise Begin
  - A ← the attribute from Attributes that best* classifies Examples
  - The decision attribute for Root ← A
  - For each possible value, v_i, of A,
    - Add a new tree branch below *Root*, corresponding to the test A = v_i
    - Let *Examples* v_i, be the subset of Examples that have value v_i for *A*
    - If *Examples* v_i , is empty
      - Then below this new branch add a leaf node with label = most common value of Target_attribute in Examples
      - Else below this new branch add the subtree

        ID3(*Examples* v_i, Targe_tattribute, Attributes – {A}))
- End
- Return Root

ENTROPY:

Entropy measures the impurity of a collection of examples.

$$Entropy\ (S) \equiv -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus}$$

Where, $p+$ is the proportion of positive examples in S

$p-$ is the proportion of negative examples in S.

INFORMATION GAIN:

- ***Information gain,*** is the expected reduction in entropy caused by partitioning the examples according to this attribute.
- The information gain, Gain(S, A) of an attribute A, relative to a collection of examples S is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Gini Index

Gini index and information gain both of these methods are used to select from the n attributes of the dataset which attribute would be placed at the root node or the internal node.

If a data set D contains examples from n classes, gini index, $gini(D)$ is defined as

$$gini(D) = 1 - \sum_{j=1} p_j^2$$

where p_j is the relative frequency of class j in D

Program

```
# Import required packages
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

```
from google.colab import drive
drive.mount('/content/drive')
data =pd.read_csv("drive/My Drive/Colab Notebooks/3 id3.csv")data.head()
data.shape

data['Outlook'] = data['Outlook'].astype('category')
data['Temperature'] = data['Temperature'].astype('category')
data['Humidity'] = data['Humidity'].astype('category')
data['Wind'] = data['Wind'].astype('category')
data['Answer'] = data['Answer'].astype('category')
# Convert Categorical features to Numeric features
data['Outlook'] = data['Outlook'].cat.codes
data['Temperature'] = data['Temperature'].cat.codes
data['Humidity'] = data['Humidity'].cat.codes
data['Wind'] = data['Wind'].cat.codes
data['Answer'] = data['Answer'].cat.codes

data.head()
# Splitting the dataset into train and test
X = data.values[:, 0:4]
Y = data.values[:, 4]
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3, random_state = 100)
print (X.shape)
print (Y.shape)

# Decision Tree classifier using Gini Index
#Training Decision Tree with giniIndex.
# Creating the classifier object
dt_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100)
# Performing training
dt_gini.fit(X_train, y_train)
```

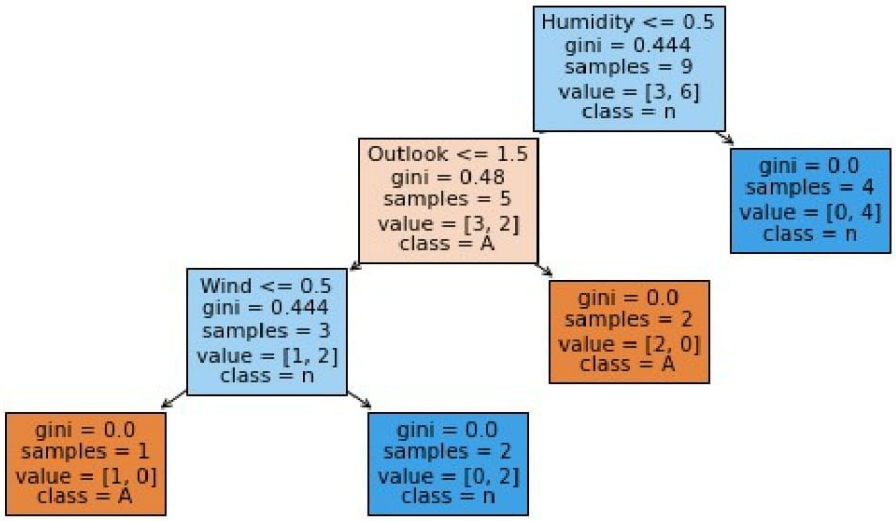
```
# Prediction using gini
print("Train Results Using Gini Index:")
y_train_pred_gini = dt_gini.predict(X_train)
print("Confusion Matrix: \n",confusion_matrix(y_train, y_train_pred_gini))
print ("Accuracy : ", accuracy_score(y_train, y_train_pred_gini)*100)
print("Report : ", classification_report(y_train, y_train_pred_gini))

# Prediction using gini
print("Test Results Using Gini Index:")
y_pred_gini = dt_gini.predict(X_test)
print("Confusion Matrix: \n",confusion_matrix(y_test, y_pred_gini))
print ("Accuracy : ", accuracy_score(y_test, y_pred_gini)*100)
print("Report : ", classification_report(y_test, y_pred_gini))

from sklearn import tree
text_rep = tree.export_text (dt_gini)
print (text_rep)

from matplotlib import pyplot as plt
fig = plt.figure(figsize=(10,5))
x = tree.plot_tree(dt_gini,
                   feature_names=['Outlook', 'Temperature', 'Humidity', 'Wind'] ,
                   class_names='Answer',
                   filled=True)
```

Output:

	Outlook	Temperature	Humidity	Wind	Answer
0	sunny	hot	high	weak	no
1	sunny	hot	high	strong	no
2	overcast	hot	high	weak	yes
3	rain	mild	high	weak	yes
4	rain	cool	normal	weak	yes

	Outlook	Temperature	Humidity	Wind	Answer
0	2	1	0	1	0
1	2	1	0	0	0
2	0	1	0	1	1
3	1	2	0	1	1
4	1	0	1	1	1

```
Train Results Using Gini Index:
Confusion Matrix:
 [[3 0]
 [0 6]]
Accuracy :  100.0
Report :                precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       1.00      1.00      1.00         6

    accuracy                           1.00         9
   macro avg       1.00      1.00      1.00         9
weighted avg       1.00      1.00      1.00         9
```



Experiment-4:

Exercises to solve the real-world problems using the following machine learning methods:
a) Linear Regression b) Logistic Regression c) Binary Classifier

a) Linear Regression

Linear regression is probably one of the most important and widely used regression techniques. It's among the simplest regression methods. One of its main advantages is the ease of interpreting results. When implementing linear regression of some dependent variable $y$on the set of independent variables x= (x1…xn), where n is the number of predictors, you assume a linear relationship between y and x

$$y= \beta0 + \beta1x1 + \ldots\ldots + \beta nxn + \epsilon.$$

This equation is the regression equation. β0, β1, … βn are the regression coefficients, and ε is the random error. Linear regression calculates the estimators of the regression coefficients or simply the predicted weights.

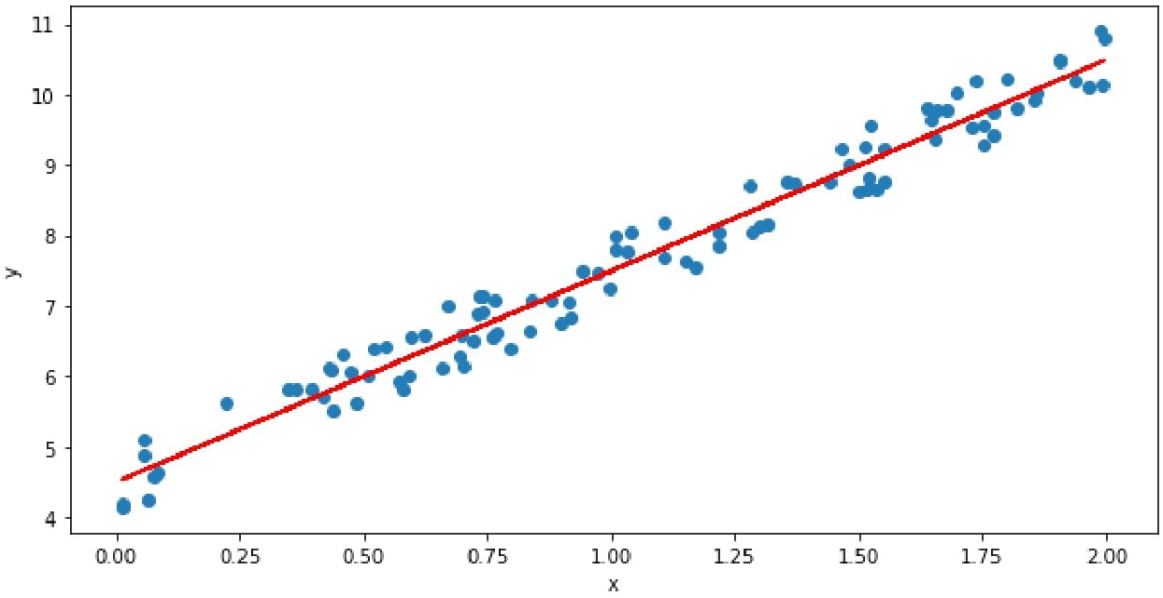
Let's start with the simplest case, which is simple linear regression. There are five basic steps when you're implementing linear regression:

1. Import the packages and classes you need.
2. Provide data to work with and eventually do appropriate transformations.
3. Create a regression model and fit it with existing data.
4. Check the results of model fitting to know whether the model is satisfactory.
5. Apply the model for predictions.

Code:

```
import numpy as np
x = 2 * np.random.rand(100,1)
y = 4 + 3 * x + np.random.rand(100,1)

from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit (x,y)
```

```
print ('Intercept value:',lr.intercept_)
print ('Coefficient value:', lr.coef_)

y_pred = lr.predict (x)
plt.figure(figsize=(10,5))
plt.xlabel('x')
plt.ylabel('y')
plt.scatter (x,y)
plt.plot (x,y_pred,color='red')
```



b) Logistic Regression

Logistic regression is a statistical method that is used for building machine learning models where the dependent variable is binary. Logistic regression is used to describe data and the relationship between one dependent variable and one or more independent variables. The independent variables can be nominal, ordinal, or of interval type.

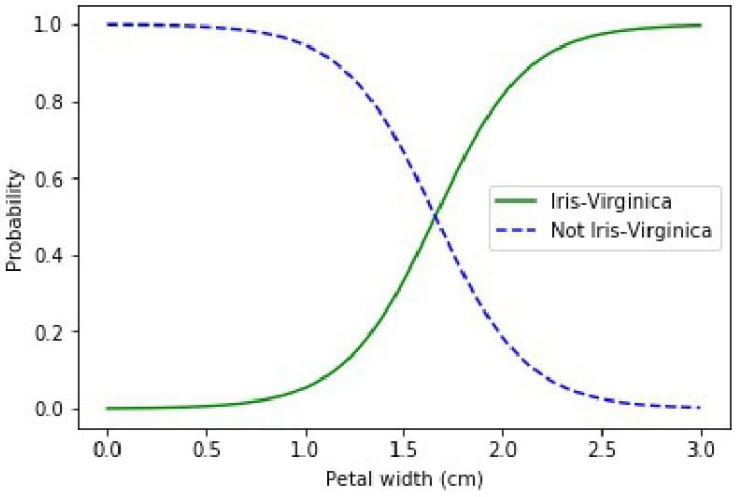
The name “logistic regression” is derived from the concept of the logistic function that it uses. The logistic function is also known as the sigmoid function. The value of this logistic function lies between zero and one.

Logistic function or sigmoid function to calculate probability in logistic regression. The logistic function is a simple S-shaped curve used to convert data into a value between 0 and 1.

Code: Classifier for iris dataset based on petal widths

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

iris = datasets.load_iris()
print (iris.keys())

print (iris["feature_names"])
iris

iris['target']

x = iris["data"][:,3:] # Consider only  petal width (cm) feature
y = (iris['target'] == 2).astype(np.int)
x.shape, y.shape

from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression ()
log_reg.fit (x,y)

print ('Logistic Regression Accuracy Score:', log_reg.score (x,y))

y_pred = log_reg.predict (x)
y_pred

from sklearn.metrics import confusion_matrix, accuracy_score
print (confusion_matrix (y,y_pred))

accuracy_score(y,y_pred)
```

```
x_new = np.linspace (0,3, 1000).reshape (-1, 1)

y_proba = log_reg.predict_proba (x_new)

plt.plot (x_new, y_proba[:,1],"g", label = "Iris-Virginica")
plt.plot (x_new, y_proba[:,0],"b--", label = "Not Iris-Virginica")
plt.xlabel ("Petal width (cm)")
plt.ylabel ("Probability")
plt.legend()
```



Experiment-5:

Develop a program for Bias, Variance, Remove duplicates, Cross Validation

Solution:

Bias – This is the difference between predicted values and expected results. A machine learning model with a low bias is a perfect model and a model with a high bias is expected with a high error rate on the training and test sets.

Variance – This is the variability of your model's predictions over different sets of data. A machine learning model with high variance indicates that the model may work well on the data it was trained on, but it will not generalize well on the dataset it has never seen before.

Reducing the bias can easily be achieved by increasing the variance. Conversely, reducing the variance can easily be achieved by increasing the bias. This relationship is generally referred to as the **bias-variance trade-off**. The goal is to balance or achieve an optimum model complexity between these two concepts that would not ***underfit*** or ***overfit***.

The ***bias_variance_decomp()*** function that can estimate the bias and variance for a model over several samples. This function includes the following parameters:

estimator: A Regressor or classifier object that performs a fit or predicts method

X_train : The training dataset

y_train : The targets that correspond with the X_train examples

X_test : The test dataset used for computing the average loss, bias, and variance that corresponds with the X_train examples

y_test : The targets that correspond with the y_test examples

loss: The loss function for performing the bias-variance decomposition

num_rounds: Total number of rounds for performing the bias-variance decomposition

Code:

```
# Load required packages
from mlxtend.evaluate import bias_variance_decomp
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.utils import shuffle
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

data = pd.read_csv("5 student.csv")
data = data[["G1", "G2", "G3", "studytime", "failures", "absences"]]
data.head()
predict = "G3"
x = np.array(data.drop([predict], 1))
y = np.array(data[predict])
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2)
print (xtrain.shape, ytrain.shape, xtest.shape, ytest.shape)

lr = LinearRegression()
lr.fit(xtrain, ytrain)
y_pred = lr.predict(xtest)

mse, bias, variance = bias_variance_decomp (lr, xtrain, ytrain, xtest, ytest,
                                    loss='mse', num_rounds=150)
print ("MSE : %.3f" %mse)
print ("Average Bias : %.3f" %bias)
print ("Average Variance : %.3f " %variance)
```

Output:

	G1	G2	G3	studytime	failures	absences
0	5	6	6	2	0	6
1	5	5	6	2	0	4
2	7	8	10	2	3	10
3	15	14	15	3	0	2
4	6	10	10	2	0	4

```
316, 5) (316,) (79, 5) (79,)

MSE : 4.062
Average Bias : 4.003
Average Variance : 0.059
```

Remove duplicates – Pandas **drop_duplicates()** method helps in removing duplicates from the Pandas Dataframe.

DataFrame.drop_duplicates (subset=None, keep='first', inplace=False)

Parameter	Value	Description
subset	column label(s)	Optional. A String, or a list, containing the columns to use when looking for duplicates. If not specified, all columns are being used.
keep	'first' 'last' False	Optional, default 'first'. Specifies which duplicate to keep. If False, drop ALL duplicates
inplace	True False	Optional, default False. True: the removing is done on the current DataFrame. False: returns a copy where the removing is done.

Code:

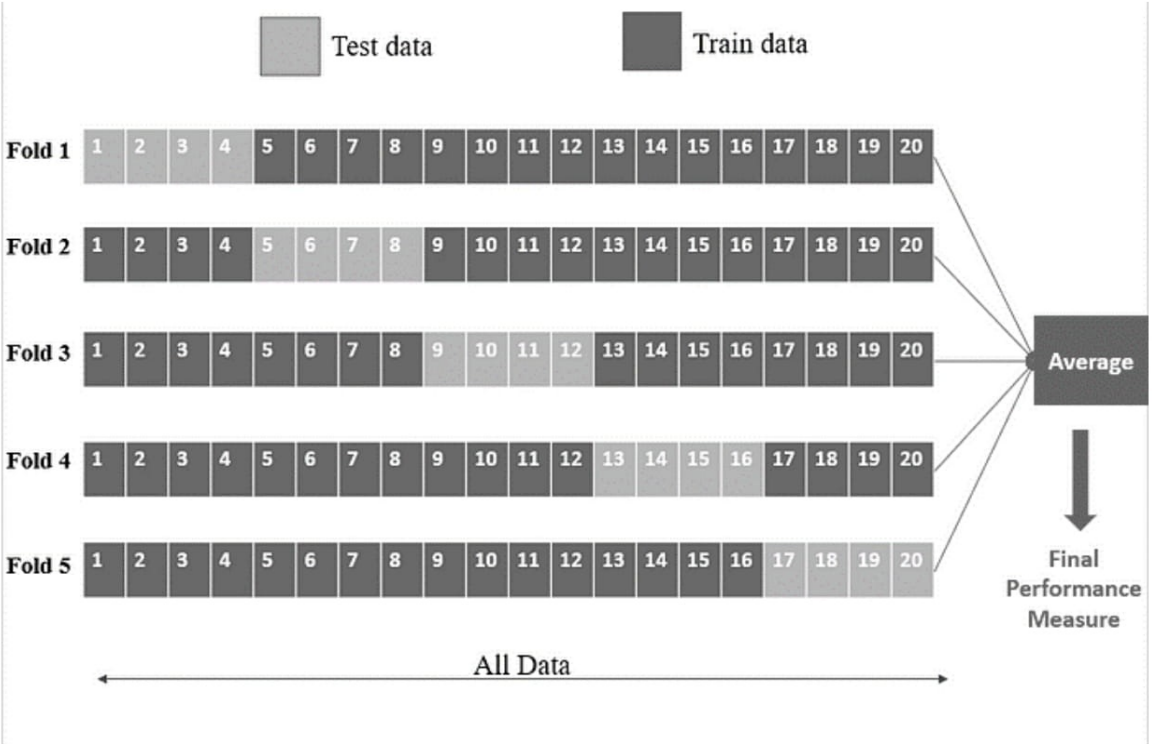
```
import pandas as pd
data = {'F1': [1, 1, 1, 2], 'F2': [2, 2, 2, 3], 'F3': [3, 3, 4, 5]}
df = pd.DataFrame(data)
print ('Source DataFrame:\n', df)

# Drop Duplicate Rows keeping the First One
result_df = df.drop_duplicates()
print ('Result DataFrame:\n', result_df)
```

```
# Delete All Duplicate Rows from DataFrame
result_df = df.drop_duplicates(keep=False)
print ('Result DataFrame:\n', result_df)

# Identify Duplicate Rows based on Specific Columns (by default keeps the first record)
result_df = df.drop_duplicates(subset=['F1', 'F2'])
print ('Result DataFrame:\n', result_df)

# Identify Duplicate Rows based on Specific Columns and keep the last record
result_df = df.drop_duplicates(subset=['F1', 'F2'], keep='last')
print ('Result DataFrame:\n', result_df)
```

```
Source DataFrame:
    F1  F2  F3
0   1   2   3
1   1   2   3
2   1   2   4
3   2   3   5

Result DataFrame:
    F1  F2  F3
0   1   2   3
2   1   2   4
3   2   3   5


Result DataFrame:
    F1  F2  F3
2   1   2   4
3   2   3   5


Result DataFrame:
    F1  F2  F3
0   1   2   3
3   2   3   5


Result DataFrame:
    F1  F2  F3
2   1   2   4
3   2   3   5
```

Cross validation: K-fold cross-validation is a superior technique to validate the performance of a machine learning model. It evaluates the model using different portions of the data set as the validation set. Below are the steps for it:

1. Randomly split your entire dataset into ***k*** "folds".
2. For each k-fold in your dataset, build your model on ***k – 1*** folds of the dataset. Then, test the model to check the effectiveness for ***k***th fold.
3. Record the error you see on each of the predictions.
4. Repeat this until each of the k-folds has served as the test set.
5. The average of your ***k*** recorded errors is called the cross-validation error and will serve as your performance metric for the model.



Each data will be considered one time in test set and k-1 times in training set enhancing the effectiveness of this method. Each fold will give different accuracy and final accuracy will be average of all these 5 accuracies.

Code:

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.model_selection import KFold, cross_val_score

x, y = datasets.load_iris(return_X_y=True)
print ("Size of independent and dependent features: ", x.shape, y.shape)

dtclf = DecisionTreeClassifier(random_state=30)

cv_feature = KFold(n_splits = 4)
cv_scores = cross_val_score(dtclf, x, y, cv = cv_feature)

print ("Cross Validation Scores: ", cv_scores)
print ("Average CV Score: ", cv_scores.mean())
print ("Number of CV Scores used in Average: ", len(cv_scores))
```

Output:

```
Size of independent and dependent features:  (150, 4) (150,)

Cross Validation Scores:  [1.         0.92105263 0.89189189 0.7027027]
Average CV Score:  0.8789118065433854
Number of CV Scores used in Average:  4
```

Experiment-6:

Write a program to implement Categorical Encoding, One-hot Encoding

Solution:

In many Machine-learning or Data Science activities, the data set might contain text or categorical values (basically non-numerical values). For example, color feature having values like red, orange, blue, white etc. Meal plan having values like breakfast, lunch, snacks, dinner, tea etc. But, most of the Machine Learning algorithms cannot work with categorical data and needs to be converted into numerical data.

There are many ways to convert categorical values into numerical values. Each approach has its own trade-offs and impact on the data set. The two main methods are:

- Label-Encoder and
- One-Hot-Encoder

Both of these encoders are part of SciKit-learn library and are used to convert text or categorical data into numerical data.

Label encoder

This is a very simple approach that converts each value in a column to a number.

One-Hot encoder

Label encoding has the disadvantage that the numeric values can be misinterpreted by algorithms as having some sort of hierarchy/order in them. This issue can be solved using 'One-Hot Encoding'. In this, each category value is converted into a new column and assigned a 1 or 0 value to the column. It can cause the number of columns to expand greatly if you have many unique values in a category column.

Label encoding – Using category codes approach

```
# import required libraries
import pandas as pd
import numpy as np
# creating initial dataframe
bridge_types = ('Arch','Beam','Truss','Cantilever','Tied Arch','Suspension','Cable')
bridge_df = pd.DataFrame(bridge_types, columns=['Bridge_Types'])
# converting type of columns to 'category'
```

```
bridge_df['Bridge_Types'] = bridge_df['Bridge_Types'].astype('category')
# Assigning numerical values and storing in another column
bridge_df['Bridge_Types_Cat'] = bridge_df['Bridge_Types'].cat.codes
bridge_df
```

	Bridge_Types	Bridge_Types_Cat
0	Arch	0
1	Beam	1
2	Truss	6
3	Cantilever	3
4	Tied Arch	5
5	Suspension	4
6	Cable	2

Label encoding – Using Sci-kit learn library approach

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
# creating initial dataframe
bridge_types = ('Arch','Beam','Truss','Cantilever','Tied Arch','Suspension','Cable')
bridge_df = pd.DataFrame(bridge_types, columns=['Bridge_Types'])
# creating instance of labelencoder
labelencoder = LabelEncoder()
# Assigning numerical values and storing in another column
bridge_df['Bridge_Types_Cat'] = labelencoder.fit_transform(bridge_df['Bridge_Types'])
bridge_df
```

	Bridge_Types	Bridge_Types_Cat
0	Arch	0
1	Beam	1
2	Truss	6
3	Cantilever	3
4	Tied Arch	5
5	Suspension	4
6	Cable	2

One-Hot encoding – Using Sci-kit learn library approach

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder
# creating instance of one-hot-encoder
enc = OneHotEncoder(handle_unknown='ignore')
# passing bridge-types-cat column (label encoded values of bridge_types)
enc_df = pd.DataFrame(enc.fit_transform(bridge_df[['Bridge_Types_Cat']]).toarray())
# merge with main df bridge_df on key values
bridge_df = bridge_df.join(enc_df)
bridge_df
```

	Bridge_Types	Bridge_Types_Cat	0	1	2	3	4	5	6
0	Arch	0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1	Beam	1	0.0	1.0	0.0	0.0	0.0	0.0	0.0
2	Truss	6	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	Cantilever	3	0.0	0.0	0.0	1.0	0.0	0.0	0.0
4	Tied Arch	5	0.0	0.0	0.0	0.0	0.0	1.0	0.0
5	Suspension	4	0.0	0.0	0.0	0.0	1.0	0.0	0.0
6	Cable	2	0.0	0.0	1.0	0.0	0.0	0.0	0.0

One-Hot encoding – Using dummies values approach

```
import pandas as pd
import numpy as np
# creating initial dataframe
bridge_types = ('Arch','Beam','Truss','Cantilever','Tied Arch','Suspension','Cable')
bridge_df = pd.DataFrame(bridge_types, columns=['Bridge_Types'])
bridge_df
# generate binary values using get_dummies
dum_df = pd.get_dummies(bridge_df, columns=["Bridge_Types"], prefix=["Type_is"] )
dum_df
# merge with main df bridge_df on key values
```

```
bridge_df = bridge_df.join(dum_df)
bridge_df
```

	Bridge_Types
0	Arch
1	Beam
2	Truss
3	Cantilever
4	Tied Arch
5	Suspension
6	Cable

	Type_is_Arch	Type_is_Beam	Type_is_Cable	Type_is_Cantilever	Type_is_Suspension	Type_is_Tied Arch	Type_is_Truss
0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0
2	0	0	0	0	0	0	1
3	0	0	0	1	0	0	0
4	0	0	0	0	0	1	0
5	0	0	0	0	1	0	0
6	0	0	1	0	0	0	0

	Bridge_Types	Type_is_Arch	Type_is_Beam	Type_is_Cable	Type_is_Cantilever	Type_is_Suspension	Type_is_Tied Arch	Type_is_Truss
0	Arch	1	0	0	0	0	0	0
1	Beam	0	1	0	0	0	0	0
2	Truss	0	0	0	0	0	0	1
3	Cantilever	0	0	0	1	0	0	0
4	Tied Arch	0	0	0	0	0	1	0
5	Suspension	0	0	0	0	1	0	0
6	Cable	0	0	1	0	0	0	0

Experiment-7:

Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

Solution:

```
import numpy as np
X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) # maximum of X array longitudinally
y = y/100
print (X)
print (y)

# Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))
# Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

# Variable initialization
epoch=5000 #Setting training iterations / epochs
lr=0.1 #Setting learning rate
inputlayer_neurons = 2 #number of input features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer

# Initialize Weight and Bias for hidden and output layers
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
```

```
bout=np.random.uniform(size=(1,output_neurons))

# Draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
    # Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+ bout
    output = sigmoid(outinp)

    # Back Propagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)

    # How much hidden layer wts contributed to error
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad

    # Dotproduct of nextlayererror and currentlayerop
    wout += hlayer_act.T.dot(d_output) *lr
    wh += X.T.dot(d_hiddenlayer) *lr

print("Input: \n" + str(X))
print("Actual Output: \n" + str(y))
print("Predicted Output: \n" ,output)
```

Output:

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]

Actual Output:
[[0.92]
 [0.86]
 [0.89]]

Predicted Output:
 [[0.89471252]
 [0.87605844]
 [0.89876295]]
```

Experiment-8:

Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

Solution:

KNN Classification algorithm is used to solve the classification model problems. K-nearest neighbor or K-NN algorithm basically creates an imaginary boundary to classify the data. When new data points come in, the algorithm will try to predict that to the nearest of the boundary line. Therefore, larger k value means smother curves of separation resulting in less complex models. Whereas, smaller k value tends to over fit the data and resulting in complex models. It's very important to have the right k-value when analyzing the dataset to avoid over fitting and under fitting of the dataset.

Pipeline of the solution

- The **k-nearest neighbor algorithm** is imported from the scikit-learn package.
- Create feature and target variables.
- Split data into training and test data.
- Generate a **k-NN** model using **neighbor's** value.
- Train or fit the data into the model.
- Predict the future.

Program

```
# Import required packages
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import datasets
# Load Iris dataset
iris = datasets.load_iris()
#The following variables x contains the four features and y contains the class labels
x = iris.data
y = iris.target
```

```
print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print (x)
print ('class: 0-Iris-Setosa, 1-Iris-Versicolour, 2-Iris-Virginica')
print (y)
# Split the dataset into train and test data in 70:30 ratio.
# The train data contains 105 samples and test data contains 45 samples
x_train, x_test, y_train, y_test = train_test_split (x,y, test_size=0.3)
# Train the model using KNN classifier with neighbours =5
knnclf = KNeighborsClassifier (n_neighbors = 5)
knnclf.fit (x_train,y_train)
# Make predictions for test data
y_pred = knnclf.predict (x_test)
print ("---------------------------------------------------------------------")
print("\nConfusion Matrix for Test data:\n",metrics.confusion_matrix(y_test, y_pred))
print ("---------------------------------------------------------------------")
print("\nKNN Classification Report:\n",metrics.classification_report(y_test, y_pred))
print ("---------------------------------------------------------------------")
print('Test data Accuracy of the KNNclassifieris %0.2f' %
metrics.accuracy_score(y_test,y_pred))
print ("---------------------------------------------------------------------")
# Print both correct and wrong predictions
i = 0
print ("\n---------------------------------------------------------------------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))
print ("---------------------------------------------------------------------")
for label in y_test:
    print ('%-25s %-25s' % (label, y_pred[i]), end="")
    if (label == y_pred[i]):
        print (' %-25s' % ('Correct'))
    else:
        print (' %-25s' % ('Wrong'))
```

```
  i = i + 1
--------------------------------------------------------------------------
Confusion Matrix:
 [[16  0  0]
 [ 0 14  0]
 [ 0  1 14]]
--------------------------------------------------------------------------

KNN Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.93      1.00      0.97        14
           2       1.00      0.93      0.97        15

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45

--------------------------------------------------------------------------
Accuracy of the KNN classifier is 0.98
--------------------------------------------------------------------------


--------------------------------------------------------------------------
Original Label            Predicted Label           Correct/Wrong
--------------------------------------------------------------------------
1                         1                         Correct
1                         1                         Correct
2                         2                         Correct
2                         2                         Correct
0                         0                         Correct
2                         2                         Correct
1                         1                         Correct
0                         0                         Correct
0                         0                         Correct
1                         1                         Correct
2                         2                         Correct
2                         2                         Correct
2                         2                         Correct
...
0                         0                         Correct
2                         1                         Wrong
0                         0                         Correct
```

Experiment-9:

Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

Solution:

Locally Weighted Regression

1. Nonparametric regression is a category of regression analysis in which the predictor does not take a predetermined form but is constructed according to information derived from the data (training examples).
2. Nonparametric regression requires larger sample sizes than regression based on parametric models. Because larger the data available, accuracy will be high.

Locally Weighted Linear Regression

1. Locally weighted regression is called local because the function is approximated based a only on data near the query point, weighted because the contribution of each training example is weighted by its distance from the query point.
2. Query point is nothing but the point nearer to the target function, which will help in finding the actual position of the target function.

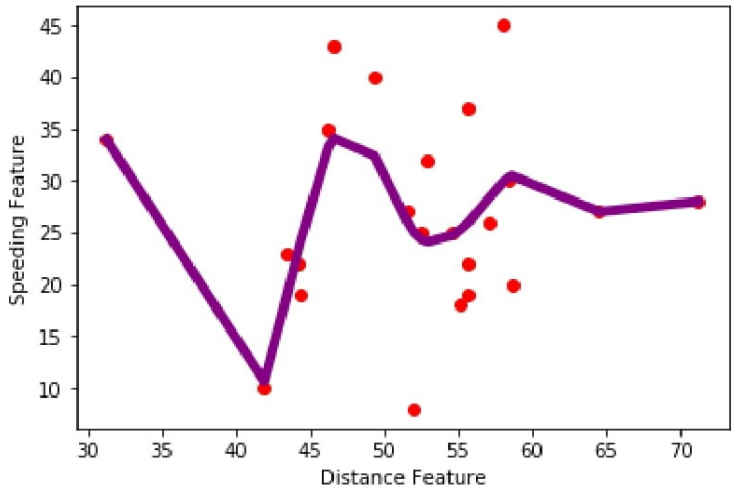
Code:

```
# Import requred packages
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def kernel(point,xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np1.eye((m)))
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.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 = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

def graphplot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest xsort = X[sortindex][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill,tip, color='red')
    ax.plot(xsort[:,1],ypred[sortindex], color = 'purple', linewidth=5)
    plt.xlabel('Distance Feature')
    plt.ylabel('Speeding Feature')
    plt.show()

# Load data set
data = pd.read_csv('11kmeansdata.csv')
df = np.array(data.Distance_Feature)
sf = np.array(data.Speeding_Feature)

# Preparing and add 1 in sf
df = np.mat(df)
sf = np.mat(sf)
m = np.shape(df)[1]
```

```
one = np.mat(np.ones(m))
X= np.hstack((one.T,df.T))

# Set k here
ypred = localWeightRegression(X,sf,2)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]

graphplot (X, ypred)
```

Output:



Experiment-10:

Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

Solution:

Naïve Bayes assumes conditional independence over the training dataset. The classifier separates data into different classes according to the Bayes' Theorem. But assumes that the relationship between all input features in a class is independent. Hence, the model is called naïve. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

Bayes Theorem $\longrightarrow P(A \mid x_1, \ldots, x_n) = \frac{P(x_1, \ldots, x_n \mid A)\, P(A)}{P(x_1, \ldots, x_n)}$

$\downarrow$

Naïve Bayes $\longrightarrow P(A \mid x_1, \ldots, x_n) = P(x_1 \mid A) \cdot P(x_2 \mid A) \cdot P(x_i \mid A)\, P(A)$

Code:

```
# Import required packages
import pandas as pd

msg=pd.read_csv('E:/Materials/Machine Learning/III-II IT ML Lab Manual R20 Jan 2023/10
naivetext.csv',names=['message','label'])
print('The dimensions of the dataset',msg.shape)

msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
y=msg.labelnum
print(X)
print(y)
```

```
#splitting the dataset into train and test data
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,y)
print ('\n The total number of Training Data :',ytrain.shape)
print ('\n The total number of Test Data :',ytest.shape)

#Output of count vectorizer is a sparse matrix
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
xtrain_dtm = count_vect.fit_transform(xtrain)
xtest_dtm=count_vect.transform(xtest)
print('\n The words or Tokens in the text documents \n')
print(count_vect.get_feature_names())

xtrain
xtrain_dtm

df=pd.DataFrame(xtrain_dtm.toarray(),columns=count_vect.get_feature_names())

df.head(12)
# Training Naive Bayes (NB) classifier on training data.
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(xtrain_dtm,ytrain)
train_predicted = clf.predict(xtrain_dtm)
test_predicted = clf.predict(xtest_dtm)

# Printing accuracy, Confusion matrix, Precision and Recall
from sklearn import metrics
print('\n Accuracy of the classifer is for Train data:',
metrics.accuracy_score(ytrain,train_predicted))
print('\n Accuracy of the classifer is for Test data:', metrics.accuracy_score(ytest,test_predicted))
```

```
print('\n Confusion matrix for Train data')
print(metrics.confusion_matrix(ytrain,train_predicted))
print('\n The value of Precision' , metrics.precision_score(ytrain,train_predicted))
print('\n The value of Recall' , metrics.recall_score(ytrain,train_predicted))

print('\n Confusion matrix for Test data')
print(metrics.confusion_matrix(ytest,predicted))
print('\n The value of Precision' , metrics.precision_score(ytest,predicted))
print('\n The value of Recall' , metrics.recall_score(ytest,predicted))
```

Output:

```
0                       I love this sandwich   1
1                  This is an amazing place   1
2          I feel very good about these beers   1
3                       This is my best work   1
4                       What an awesome view   1
5              I do not like this restaurant   0
6                   I am tired of this stuff   0
7                     I can't deal with this   0
8                       He is my sworn enemy   0
9                        My boss is horrible   0
10                 This is an awesome place   1
11    I do not like the taste of this juice   0
12                            I love to dance   1
13         I am sick and tired of this place   0
14                      What a great holiday   1
15            That is a bad locality to stay   0
16            We will have good fun tomorrow   1
17          I went to my enemy's house today   0
The words or Tokens in the text documents

['about', 'am', 'amazing', 'an', 'awesome', 'bad', 'beers', 'best', 'can',
'dance', 'deal', 'do', 'enemy', 'feel', 'fun', 'good', 'great', 'have',
'he', 'holiday', 'is', 'juice', 'like', 'locality', 'love', 'my', 'not',
'of', 'place', 'restaurant', 'stay', 'stuff', 'sworn', 'taste', 'that',
'the', 'these', 'this', 'tired', 'to', 'tomorrow', 'very', 'view', 'we',
'what', 'will', 'with', 'work']
Confusion matrix for Train data
[[6 0]
 [0 7]]
Confusion matrix for Test data
[[2 1]
 [0 2]]
```

Experiment-11:

Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

Solution:

K-Means algorithm

1. First, we need to specify the number of clusters, K, need to be generated by this algorithm.
2. Next, randomly select K data points and assign each data point to a cluster. In simple words, classify the data based on the number of data points.
3. Now it will compute the cluster centroids.
4. Next, keep iterating the following until we find optimal centroid which is the assignment of data points to the clusters that are not changing any more −
   a. First, the sum of squared distance between data points and centroids would be computed.
   b. Now, we have to assign each data point to the cluster that is closer than other cluster (centroid).
   c. At last compute the centroids for the clusters by taking the average of all data points of that cluster.

Program

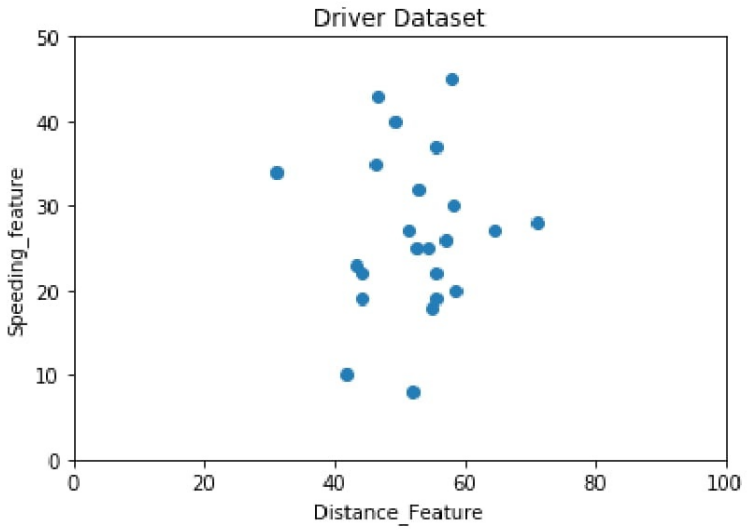
```
# Import required packages
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans

from google.colab import drive
drive.mount('/content/drive')
data =pd.read_csv("drive/My Drive/Colab Notebooks/11kmeansdata.csv")
df1=pd.DataFrame(data)
df1
```

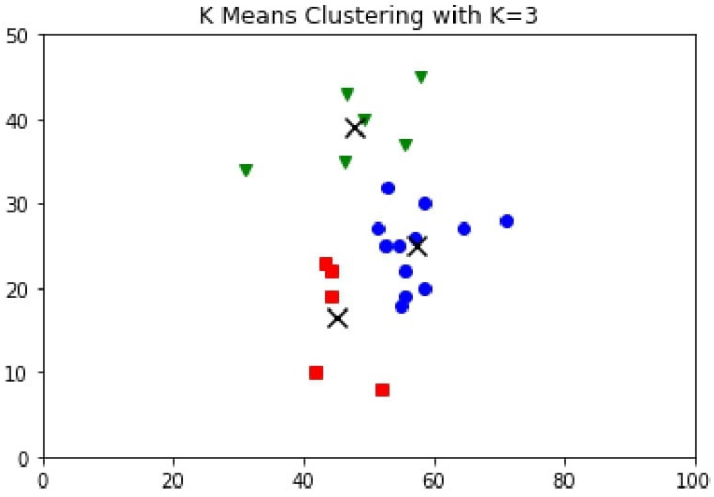
```
f1 = df1['Distance_Feature'].values
f2 = df1['Speeding_Feature'].values
X=np.matrix(list(zip(f1,f2)))

plt.plot()
plt.xlim([0, 100])
plt.ylim([0, 50])
plt.title('Driver Dataset')
plt.ylabel('Speeding_feature')
plt.xlabel('Distance_Feature')
plt.scatter(f1,f2)
plt.show()

# KMeans algorithm
#K = 3
kmeans_model = KMeans(n_clusters=3).fit(X)

# Cluster labels for the given input records
kmeans_model.labels_

# Centers for the three clusters
kmeans_model.cluster_centers_

# The transform() method measures the distance from each instance to every centroid
kmeans_model.transform(X)

colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']

plt.xlim([0, 100])
```

```
plt.ylim([0, 50])
plt.title("K Means Clustering with K=3")
for i, l in enumerate(kmeans_model.labels_):
    plt.plot(f1[i], f2[i], color=colors[l], marker=markers[l],ls='None')

centroids = kmeans_model.cluster_centers_
plt.scatter (centroids[:,0], centroids[:,1],
            marker='x', s=100, linewidths=3,
            color='k', zorder=10)
plt.show()
```

Output

	Driver_ID	Distance_Feature	Speeding_Feature
0	3423311935	71.24	28
1	3423313212	52.53	25
2	3423313724	64.54	27
3	3423311373	55.69	22
4	3423310999	54.58	25
5	3423313857	41.91	10
6	3423312432	58.64	20
7	3423311434	52.02	8
8	3423311328	31.25	34
9	3423312488	44.31	19
10	3423311254	49.35	40




K Means Clustering with K=3
50
40
30
20
10
0
0
20
40
60
80
100

Experiment-12:

Exploratory Data Analysis for Classification using Pandas or Matplotlib.

Solution:

Exploratory Data Analysis is the first step of any data science project. It gives an idea of which set of variables will best serve as the input to a Machine Learning / Deep Learning model. In statistics, exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. By the name itself, we can get to know that it is a step in which we need to explore the data set.

An essential step before moving on to machine learning or data modelling is exploratory data analysis. By doing this, we can learn whether the characteristics chosen are suitable for modelling, whether all the features are necessary, and whether there are any connections, depending on which we can either return to the Data Pre-processing stage or proceed to modelling.

Exploratory Data Analysis can be used for both supervised and unsupervised machine learning modelling once it is finished and conclusions have been formed. Once the exploratory data analysis is complete, we will have a variety of plots, heat maps, frequency distribution, graphs, correlation matrices, and hypothesis that anybody can use to understand what data is all about and what insights gained from examining the data set. There are many steps for conducting exploratory data analysis.

1. Description of data
2. Handling missing data
3. Handling outliers
4. Understanding relationships and new insights through plots

1. Description of data:

In Pandas, we can apply describe() on a DataFrame which helps in generating descriptive statistics that summarize the central tendency, dispersion, and shape of a dataset's distribution, excluding NaN values. The result's index will include count, mean, std, min, max as well as lower, 50 and upper percentiles. By default, the lower percentile is 25 and the upper percentile is 75. The 50 percentile is the same as the median.

```
# Import required packages
import pandas as pd
```

```
from google.colab import files
uploaded = files.upload()
import io
boston_df = pd.read_csv (io.BytesIO(uploaded['12 HousingData (1).csv']), index_col=0,
parse_dates=True)
boston_df.describe()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.6
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.1
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.7
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.9
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.3
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.9
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.9

2. Handling missing data:

The real-world data can either be missing during data extraction or collection due to several reasons. Some of the techniques to handle missing data are

Drop NULL or missing values – This method reduces the quality of our model as it reduces sample size because it works by deleting all other observations where any of the variables is missing.

```
# Drop all rows that contain null values
boston_df = boston_df.dropna()
print ("Data set size after droping rows that contain null values:", boston_df.shape)
```

Fill Missing Values – This is a process whereby missing values are replaced with a test statistic like mean, median or mode of the particular feature the missing value belongs to.

```
# Fill the missing value in AGE feature with 30
boston_df ['AGE'] = boston_df ['AGE'].fillna(30)
```

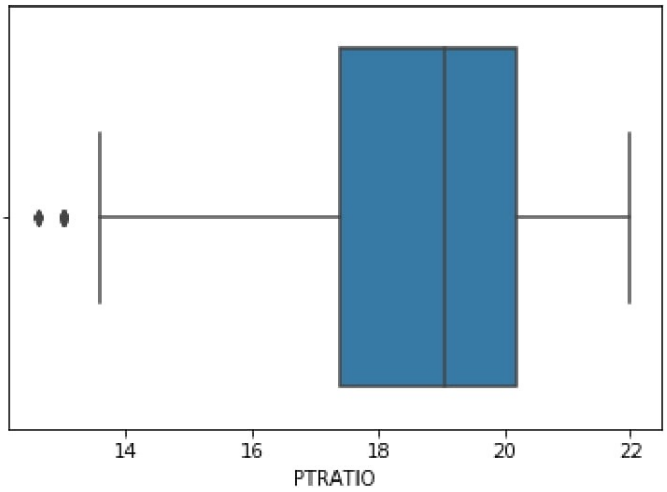
Predict Missing values with an ML Algorithm – Depending on the class of data that is missing, one can either use a regression or classification model to predict missing data.

3. Handling outliers

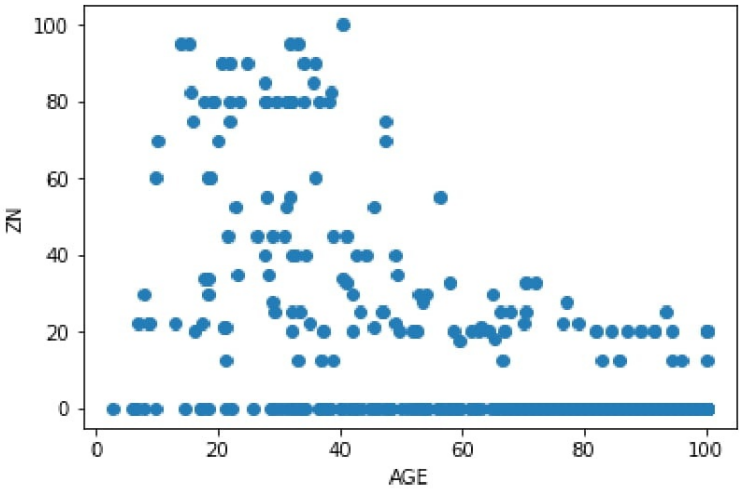
Outliers can be a result of a mistake during data collection or it can be just an indication of variance in your data. Methods used for detecting and handling outliers:

- *BoxPlot* – graphically shows groups of numerical data through their quartiles. The box extends from the Q1 to Q3 quartile values of the data, with a line at the median (Q2).
- *Scatterplot* – The data are displayed as a collection of points, each having the value of one variable determining the position on the horizontal axis and the value of the other variable determining the position on the vertical axis.
- *Z-score* – the signed number of standard deviations by which the value of an observation or data point is above the mean value of what is being observed or measured. In most of the cases a threshold of 3 or -3 is used i.e if the Z-score value is greater than or less than 3 or -3 respectively, that data point will be identified as outliers.
- IQR(Inter-Quartile Range) – is a measure of statistical dispersion, being equal to the difference between 75th and 25th percentiles, or between upper and lower quartiles.

```
# BoxPlot for PTRATIO
import seaborn as sns
sns.boxplot (x = boston_df ['PTRATIO'])
```




```
# Scatter plot between AGE and ZN
import matplotlib.pyplot as plt
plt.scatter(boston_df['AGE'] , boston_df['ZN'])
plt.xlabel('AGE')
plt.ylabel('ZN')
plt.show()
```




```
# Z-Score
from scipy import stats
import numpy as np
z = np.abs (stats.zscore (boston_df))
print (z)
boston_df_zscore = boston_df [(z < 3).all (axis = 1)]
boston_df_zscore.shape
(415, 13)

# Inter Quartile Range (IQR)
q1 = boston_df.quantile (0.25)
q3 = boston_df.quantile (0.75)
iqr = q3 - q1
print (iqr)

CRIM        3.595038
ZN         12.500000
INDUS      12.910000
CHAS        0.000000
NOX         0.175000
RM          0.738000
AGE        49.050000
DIS         3.088250
RAD        20.000000
TAX       387.000000
PTRATIO     2.800000
```

```
B           20.847500
LSTAT       10.005000
dtype: float64
```

Remove all the outliers in our dataset

boston_df_iqr = boston_df [~((boston_df < (q1 - 1.5 * iqr)) |

(boston_df > (q3 + 1.5 * iqr))).any (axis=1)]

boston_df_iqr.shape

```
(274, 13)
```

4. Understanding relationships and new insights through plots

The Heat Map procedure shows the distribution of a quantitative variable over all combinations. The correlation between two random variables is a number that runs from -1 through 0 to +1 and indicates a strong inverse relationship, no relationship, and a strong direct relationship, respectively.

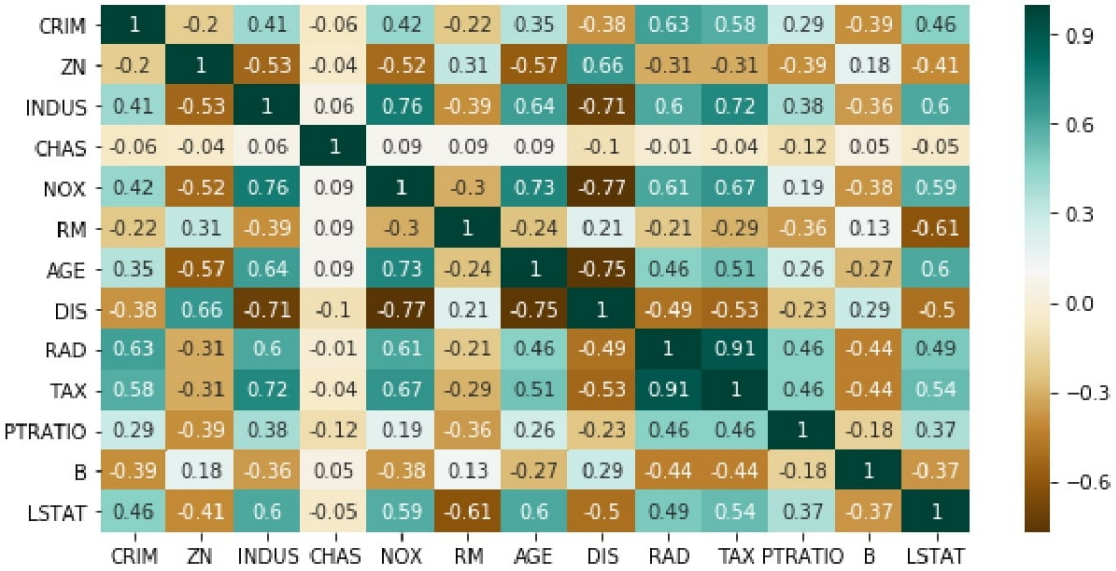
Correlation matrix

Correlation matrix

corr_matrix = boston_df.corr().round(2)

plt.figure(1, figsize=(10,5))

sns.heatmap (data = corr_matrix, annot = True, cmap='BrBG')



Experiment-13:

Write a Python program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

Solution:

Bayesian Network – A Bayesian Network is a Probabilistic Graphical Modelling (PGM) technique used to compute uncertainties by using the concept of probability. This is also known as Belief Networks, Bayesian Networks are used to model uncertainties by using Directed Acyclic Graphs (DAG).

Bayesian Networks have given shape to complex problems that provide limited information and resources. It's being implemented in the most advancing technologies such as Artificial Intelligence and Machine Learning.

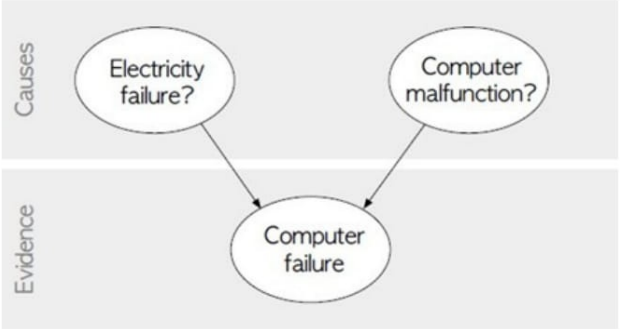
Conditional Probability of an event X is the probability that the event will occur given that an event Y has already occurred.

p(X| Y) is the probability of event X occurring, given that event, Y occurs.

- If X and Y are dependent events then the expression for conditional probability is given by: P (X| Y) = P (X and Y) / P (Y)
- If X and Y are independent events then the expression for conditional probability is given by: P(X| Y) = P (X)

Example

Suppose we attempt to turn on our computer, but the computer does not start (observation/evidence). We would like to know which of the possible causes of computer failure is more likely. In this simplified illustration, we assume only two possible causes of this misfortune: ***electricity failure*** and ***computer malfunction***. The corresponding directed acyclic graph is



The goal is to calculate the posterior conditional probability distribution of each of the possible unobserved causes given the observed evidence, i.e. P [Cause | Evidence].

Dataset:

The Cleveland database contains 76 attributes, but experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "Heartdisease" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

Code:

```
# Import required packages
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
from google.colab import drive
drive.mount('/content/drive')

#read Cleveland Heart Disease data
heartDisease =pd.read_csv("drive/My Drive/Colab Notebooks/13 heart.csv")
heartDisease = heartDisease.replace('?',np.nan)
# Display the data
print('Sample instances from the dataset are given below')
heartDisease.head()

# Display the Attributes names and datatyes
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
# Create Model- Bayesian Network
model =  BayesianModel([('age','heartdisease'),('sex','heartdisease'),( 'exang','heartdisease'),
                ('cp','heartdisease'),('heartdisease', 'restecg'),('heartdisease','chol')])
```

```
# Learning Conditional Probability Distributions (CPDs) using Maximum Likelihood Estimators
print('\n Learning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

# Inferenceing with Bayesian Network
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)

# Computing the Probability of HeartDisease given restecg
print('\n1.Probability of HeartDisease given evidence= restecg :2')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':2})
print(q1)

# Computing the Probability of HeartDisease given cp
print('\n2.Probability of HeartDisease given evidence= cp:4')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':4})
print(q2)

# Computing the Probability of HeartDisease given restecg, cp, exang
print('\n3.Probability of HeartDisease given evidence= restecg:2, cp:2, exang:1')
q3=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':2,'cp':2,
'exang':1})
print(q3)
```

Output:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0

1.Probability of HeartDisease given evidence= restecg :2

heartdisease	phi(heartdisease)
heartdisease(0)	0.2697
heartdisease(1)	0.2195
heartdisease(2)	0.1522
heartdisease(3)	0.1763
heartdisease(4)	0.1822

2. Probability of HeartDisease given evidence= cp:3

heartdisease	phi(heartdisease)
heartdisease(0)	0.4754
heartdisease(1)	0.1809
heartdisease(2)	0.1262
heartdisease(3)	0.1136
heartdisease(4)	0.1038

3. Probability of HeartDisease given evidence= restecg:2, cp:2, exang:1

heartdisease	phi(heartdisease)
heartdisease(0)	0.1773
heartdisease(1)	0.1914
heartdisease(2)	0.1654
heartdisease(3)	0.2095
heartdisease(4)	0.2564

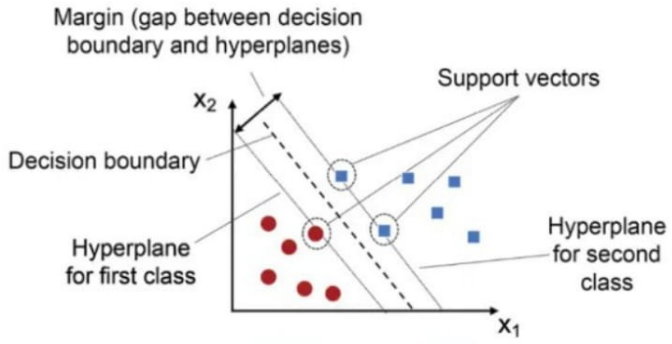
Experiment-14:

Write a program to Implement Support Vector Machines.

Solution:

Support Vector Machine (SVM) is a supervised machine learning algorithm that can be used for both classification and regression tasks. SVM for classification is termed as ***Support Vector Classification*** (SVC) and SVM for regression is termed as ***Support Vector Regression*** (SVR). The idea behind SVM classifier is to find a hyperplane that maximally separates the data points of different classes. In other words, we are looking for the largest margin between the two classes. The logic behind having decision boundaries with large margins is that they tend to have a lower generalization error, whereas models with small margins are more prone to overfitting.

Given labeled training data (supervised learning), the SVM classification algorithm outputs an optimal hyperplane which categorizes new examples into different classes. This hyperplane is then used to make predictions on new data points.



The blue square points represent one class and the red dots represent another class. The black line is the decision boundary learned by an SVM. As you can see, the SVM has placed the boundary in such a way as to maximize the margin between the two classes. The following steps will be covered for training the model using SVM while using Python code:

1. Load the data
2. Create training and test split
3. Perform feature scaling
4. Instantiate an SVC classifier
5. Fit the model
6. Measure the model performance

Code:

```
# Import required packages
# Basic packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
 # Sklearn modules & classes
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.metrics import confusion_matrix

from google.colab import drive
drive.mount('/content/drive')
# Import Customer dataset
ds =pd.read_csv("drive/My Drive/Colab Notebooks/14 custdata.csv")
ds.head()

# Extract Age, EstimatedSalary as Independent and Purchased as Dependent Variable
x= ds.iloc[:, [2,3]].values
y= ds.iloc[:, 4].values

# Display shape of x & y
print ("Independent features:", x.shape)
print ("Dependent features:", y.shape)

# Splitting the dataset into training and test set in 75:25 ratio.
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)

# Use Standard Scaler for feature Scaling
```

```
st= StandardScaler()
x_train= st.fit_transform(x_train)
x_test= st.transform(x_test)

# Create an instance of Support Vector Classifier
svc_cl = SVC(kernel='linear', random_state=0)
svc_cl.fit(x_train, y_train)

# Predict the train set result
y_train_pred = svc_cl.predict(x_train)

# Creating the Confusion matrix
cm= confusion_matrix(y_train, y_train_pred)
print ("Train data Confusion matrix:\n",cm)
print('\n The value of Precision' , metrics.precision_score(y_train,y_train_pred))
print('\n The value of Recall' , metrics.recall_score(y_train,y_train_pred))
print('\n The value of F1 Score' , metrics.f1_score(y_train,y_train_pred))

# Predict the test set result
y_test_pred = svc_cl.predict(x_test)

# Creating the Confusion matrix
cm= confusion_matrix(y_test, y_test_pred)
print ("Train data Confusion matrix:\n",cm)
print('\n The value of Precision' , metrics.precision_score(y_test,y_test_pred))
print('\n The value of Recall' , metrics.recall_score(y_test,y_test_pred))
print('\n The value of F1 Score' , metrics.f1_score(y_test,y_test_pred))

# Visualize the result of SVC
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
```

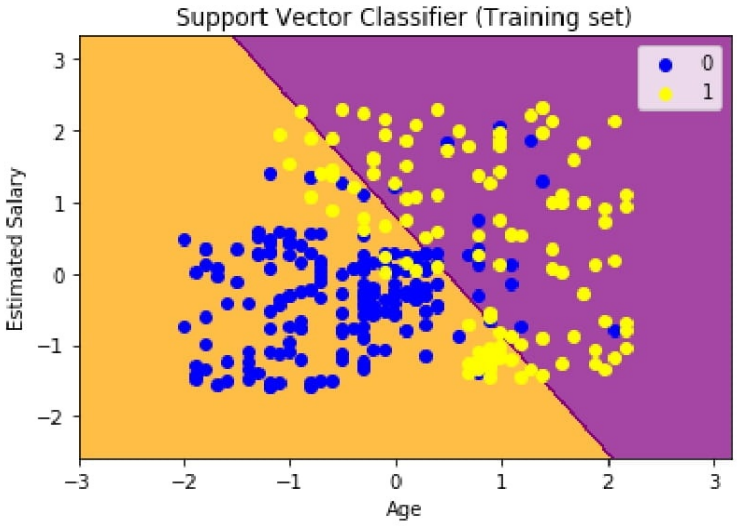
```
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1,
                     stop = x_set[:, 0].max() + 1, step  =0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
plt.contourf(x1, x2, svc_cl.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('orange', 'purple')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
    c = ListedColormap(('blue', 'yellow'))(i), label = j)
plt.title('Support Vector Classifier (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

Output:

	User ID	Gender	Age	Estimated Salary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
Train data Confusion matrix:
 [[173  16]
 [ 37  74]]

 The value of Precision 0.8222222222222222

 The value of Recall 0.6666666666666666

 The value of F1 Score 0.7363184079601989

Train data Confusion matrix:
 [[66  2]
 [ 8 24]]

 The value of Precision 0.9230769230769231
```

```
The value of Recall 0.75

The value of F1 Score 0.8275862068965517
```



Experiment-15:

Write a program to Implement Principle Component Analysis.

Solution:

Principle Component Analysis (PCA) is an Unsupervised Machine Learning task used to reduce the number of features in a data collection while retaining as much information as feasible. PCA can be mainly used for Dimensionality Reduction and also for important feature selection. When there are many input attributes, it is difficult to visualize the data. Basically, it refers to the fact that a higher number of attributes in a dataset adversely affects the accuracy and training time of the machine learning model.

PCA is pre-processing task that is carried out before applying any ML algorithm. PCA is based on "orthogonal linear transformation" which is a mathematical technique to project the attributes of a data set onto a new coordinate system. The attribute which describes the most variance is called the ***first principal component*** and is placed at the first coordinate. Similarly, the attribute which stands second in describing variance is called a ***second principal component*** and so on. The complete dataset can be expressed in terms of principal components. Generally, more than 90% of the variance is explained by two/three principal components. PCA, thus converts data from high dimensional space to low dimensional space by selecting the most important attributes that capture maximum information about the dataset. Some *basic* terminology to understand PCA

Variance – for calculating the variation of data distributed across dimensionality of graph

Covariance – calculating dependencies and relationship between features

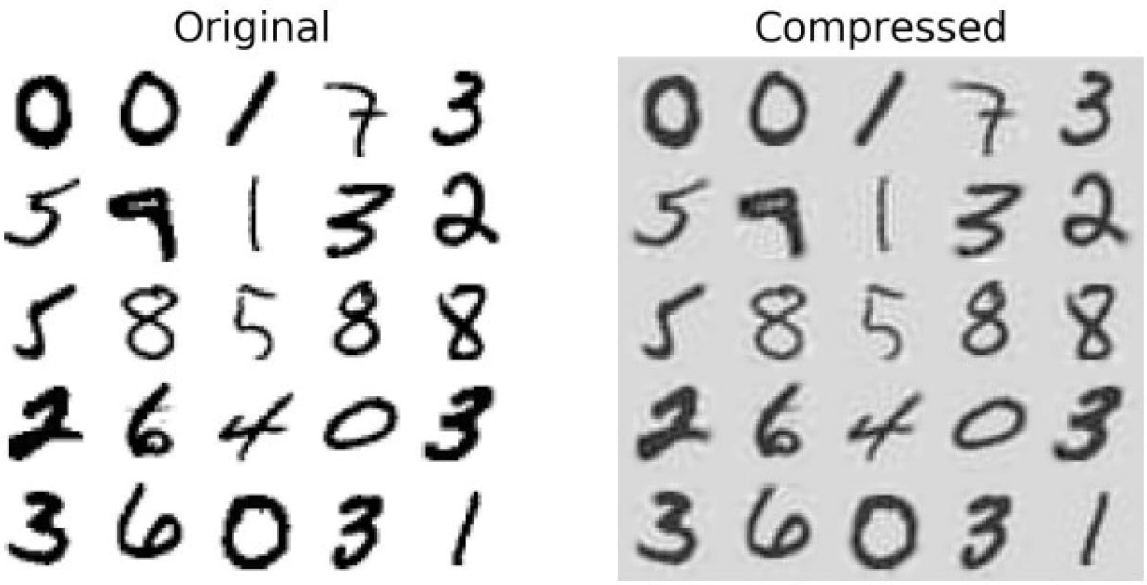
Standardizing data – Scaling our dataset within a specific range for unbiased output

Code:

Applying PCA to the MNIST dataset while preserving 95% of its variance.

```
# Import required packages
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
```

```
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
mnist.target = mnist.target.astype(np.uint8)
mnist["data"].shape
mnist["data"]

mnist["target"].shape
mnist["target"]

X = mnist["data"]
y = mnist["target"]
X_train, X_test, y_train, y_test = train_test_split(X, y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape

pca = PCA(n_components=0.95)
pca.fit(X_train)

print ("No. of features after reduction:", pca.n_components_)

# Each instance will have just over 154 features, instead of the original 784 features
pca = PCA(n_components=154)
X_reduced = pca.fit_transform (X_train)
# inverse_transform() method used to decompress from 154 back to 784 dimensions
X_recovered = pca.inverse_transform (X_reduced)

X_reduced.shape
X_recovered.shape
```


Original
Compressed