Week-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.

Aim: To 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.

Description: The find-S algorithm is a basic concept learning algorithm in machine learning. The find-S algorithm finds the most specific hypothesis that fits all the positive examples. We have to note here that the algorithm considers only those positive training example. The find-S algorithm starts with the most specific hypothesis and generalizes this hypothesis each time it fails to classify an observed positive training data. Hence, the Find-S algorithm moves from the most specific hypothesis to the most general hypothesis.

The most general hypothesis—that every day is a positive example—is represented by

$$\langle ?, ?, ?, ?, ?, ? \rangle$$

and the most specific possible hypothesis—that *no* day is a positive example—is represented by

$$\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

FIND-S: Finding a Maximally Specific Hypothesis

FIND-S Algorithm

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

Notation

The set of items over which the concept is defined is called the set of ***instances***, which we denote by X.
Example: X is the set of all possible days, each represented by the attributes: Sky, AirTemp, Humidity, Wind, Water, and Forecast

The concept or function to be learned is called the ***target concept***, which we denote by c.

c can be any Boolean valued function defined over the instances X

$$c : X \longrightarrow \{O, 1\}$$

Example: The target concept corresponds to the value of the attribute ***EnjoySport*** (i.e., c(x) = 1 if ***EnjoySport*** = Yes, and c(x) = 0 if ***EnjoySport*** = No).

- Instances for which c(x) = 1 are called positive examples, or members of the target concept.
- Instances for which c(x) = 0 are called negative examples, or non-members of the target concept.
- The ordered pair (x, c(x)) to describe the training example consisting of the instance x and its target concept value c(x).
- ***D*** to denote the set of available training examples
- The symbol ***H*** to denote the set of all possible hypotheses that the learner may consider regarding the identity of the target concept. Each hypothesis ***h*** in ***H*** represents a Boolean-valued function defined over X

$$h : X \longrightarrow \{O, 1\}$$

- The goal of the learner is to find a hypothesis h such that h(x) = c(x) for all x in ***X***.

EnjoySport.csv:

sky	airtemp	humidity	wind	water	forecast	enjoysport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	Yes

Program:

```
import random
import csv
attributes=[["Sunny","Rainy"],["Warm","Cold"],["Normal","High"],["Strong","Weak"],["Wa
rm","Cool"],["Same","Change"]]
num_attributes=len(attributes)
print("The most general hypothesis: ['?','?','?','?','?','?']")
print("The most general hypothesis: ['0','0','0','0','0','0']")
a=[]
print("The given training dataset: ")
with open('/content/Week-1.csv','r') as csvFile:
  reader=csv.reader(csvFile)
  for row in reader:
    a.append(row)
    print(row)
print("The initial value of hypothesis: ")
hypothesis=['0']*num_attributes
print(hypothesis)
for j in range(0,num_attributes):
  hypothesis[j]=a[0][j]
print("FIND-S: Finding a Maximality Specific Hypothesis")
for i in range(0,len(a)):
  if a[i][num_attributes]=="yes":
    for j in range(0,num_attributes):
      if a[i][j]!=hypothesis[j]:
        hypothesis[j]='?'
      else:
        hypothesis[j]=a[i][j]
  print("For training example no: {0} the hypothesis is ".format(i),hypothesis)
print("The Maximally Specific Hypothesis for a given training examples:")
print(hypothesis)
```

Output:

```
The most general hypothesis: ['?','?','?','?','?','?']
The most general hypothesis: ['0','0','0','0','0','0']
The given training dataset:
['sky', 'airtemp', 'humidity', 'wind', 'water', 'forecast', 'enjoysport']
['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'yes']
['sunny', 'warm', 'high', 'strong', 'warm', 'same', 'yes']
['rainy', 'cold', 'high', 'strong', 'warm', 'change', 'no']
['sunny', 'warm', 'high', 'strong', 'cool', 'change', 'yes']
The initial value of hypothesis:
['0', '0', '0', '0', '0', '0']
FIND-S: Finding a Maximality Specific Hypothesis
For training example no: 0 the hypothesis is  ['sky', 'airtemp', 'humidity', 'wind', 'water', 'forecast']
For training example no: 1 the hypothesis is  ['?', '?', '?', '?', '?', '?']
For training example no: 2 the hypothesis is  ['?', '?', '?', '?', '?', '?']
For training example no: 3 the hypothesis is  ['?', '?', '?', '?', '?', '?']
For training example no: 4 the hypothesis is  ['?', '?', '?', '?', '?', '?']
The Maximally Specific Hypothesis for a given training examples:
['?', '?', '?', '?', '?', '?']
```

Week-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.

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

Description: The candidate Elimination algorithm finds all hypotheses that match all the given training examples. Unlike in Find-S algorithm and List-then-Eliminate algorithm, it goes through both negative and positive examples, eliminating any inconsistent hypothesis.
It 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.

The key idea in the CANDIDATE-ELIMINATION algorithm is to output a description of the set of all ***hypotheses consistent with the training examples***

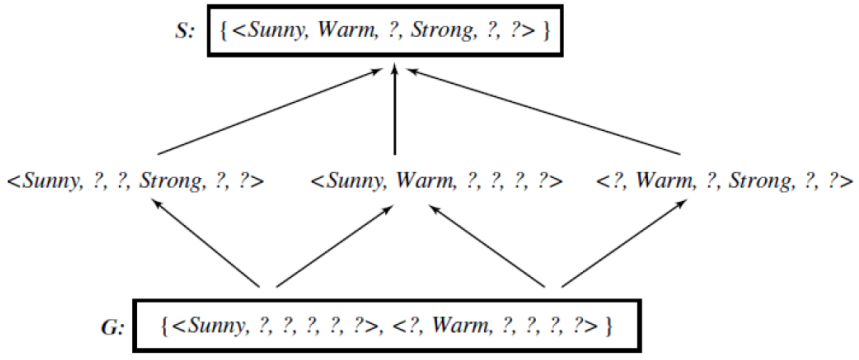
Representation

- ***Definition:*** A hypothesis h is **consistent** with **a** set of training examples ***D*** if **and** only if ***h(x) = c(x)*** for each example (x, ***c(x)***) in ***D.***

$$Consistent(h, D) \equiv (\forall \langle x, c(x)\rangle \in D)\ h(x) = c(x))$$

Note difference between definitions of *consistent* and *satisfies*

- an example *x* is said to ***satisfy*** hypothesis ***h*** when ***h(x)*** = 1, regardless of whether *x* is a positive or negative example of the target concept.
- an example *x* is said to ***consistent*** with hypothesis ***h*** iff ***h(x) = c(x)***



- A version space with its general and specific boundary sets.
- The version space includes all six hypotheses shown here, but can be represented more simply by ***S*** and ***G.***
- Arrows indicate instance of the *more-general-than* relation. This is the version space for the *Enjoysport* concept learning
- problem and training examples described in below table

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

EnjoySport.csv:

sky	airtemp	humidity	wind	water	forecast	enjoysport
sunny	warm	normal	strong	warm	same	yes
sunny	warm	high	strong	warm	same	yes
rainy	cold	high	strong	warm	change	no
sunny	warm	high	strong	cool	change	Yes

Program:

```
import numpy as np
import pandas as pd
data=pd.DataFrame(data=pd.read_csv('/content/enjoysport.csv'))
concepts=np.array(data.iloc[:,0:-1])
print(concepts)
target=np.array(data.iloc[:,-1])
print(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]='?'
    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(specific_h)
    print(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:

```
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
['yes' 'yes' 'no' 'yes']
Initialization of specific_h and general_h:
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Steps of candidate elimination algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Steps of candidate elimination algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Steps of candidate elimination algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
Steps of candidate elimination algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

Week-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.

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.

Description: ID3 Algorithm is used in machine learning for building decision trees from a given dataset. It was developed in 1986 by Ross Quinlan. It is a greedy algorithm that builds a decision tree by recursively partitioning the data set into smaller and smaller subsets until all data points in each subset belong to the same class. It employs a top-down approach, recursively selecting features to split the dataset based on information gain.

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, vi, of A,
        - Add a new tree branch below Root, corresponding to the test A = vi
        - Let Examples vi, be the subset of Examples that have value vi for A
        - If Examples vi , 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 vi, 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 and 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:

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

Training Dataset:

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Program:

```
import pandas as pd
import math
def id3(df, target_attribute_name, attribute_names, default_class=None):
    # Base cases for recursion
    # If all instances have the same class, return that class
    if len(set(df[target_attribute_name])) == 1:
        return df[target_attribute_name].iloc[0]
    # If attribute_names is empty, return the default class
    elif len(attribute_names) == 0:
        return default_class
    else:
        # Calculate information gain for each attribute
        gains = {attribute_name: information_gain(df, attribute_name, target_attribute_name)
for attribute_name in attribute_names}
        # Choose the attribute with the highest information gain
        best_attribute = max(gains, key=gains.get)
        # Create an empty tree
        tree = {best_attribute: {}}
        # Remove the best attribute from the list of attributes
        remaining_attributes = [attr for attr in attribute_names if attr != best_attribute]
        # Recursively build the tree for each value of the best attribute
        for value in df[best_attribute].unique():
            subset = df[df[best_attribute] == value]
            subtree = id3(subset, target_attribute_name, remaining_attributes, default_class)
            tree[best_attribute][value] = subtree
        return tree
# Define functions for entropy and information gain
def entropy(probs):
    return sum([-prob * math.log(prob, 2) for prob in probs if prob != 0])
def entropy_of_list(a_list):
    total_instances = len(a_list)
    class_counts = a_list.value_counts()
```

```
    probs = class_counts / total_instances
    return entropy(probs)
def information_gain(df, split_attribute_name, target_attribute_name):
    total_entropy = entropy_of_list(df[target_attribute_name])
    subset_entropy =
df.groupby(split_attribute_name)[target_attribute_name].apply(entropy_of_list)
    subset_sizes = df.groupby(split_attribute_name).size()
    weighted_entropy = (subset_entropy * subset_sizes / len(df)).sum()
    return total_entropy - weighted_entropy
# Read the dataset
df = pd.read_csv('/content/id3.csv')
# Get attribute names and remove the target attribute
attribute_names = list(df.columns)
target_attribute_name = 'Answer'
attribute_names.remove(target_attribute_name)
# Build the decision tree
tree = id3(df, target_attribute_name, attribute_names)
# Print the resultant decision tree
print("Decision Tree:")
print(tree)
```

Output:

```
Decision Tree:
{'Outlook': {'sunny': {'Humidity': {'high': 'no', 'normal': 'yes'}}, 'overcast': 'yes', 'rain':
{'Wind': {'weak': 'yes', 'strong': 'no'}}}}
```

Week-4: Exercises to solve the real-world problems using Linear Regression.

Aim: To write a program to solve the real-world problems using Linear Regression.

Description: Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent features. When the number of the independent feature, is 1 then it is known as Univariate Linear regression, and in the case of more than one feature, it is known as multivariate linear regression. There are two main types of linear regression:

Simple Linear Regression: This is the simplest form of linear regression, and it involves only one independent variable and one dependent variable. The equation for simple linear regression

$$y = \beta_0 + \beta_1 X$$

Y is the dependent variable, X is the independent variable, β0 is the intercept, β1 is the slope.

Multiple Linear Regression: This involves more than one independent variable and one dependent variable. The equation for multiple linear regression is:

$$y = \beta_0 + \beta_1 X + \beta_2 X + ...\beta_n X$$

Y is the dependent variable, X1, X2, …, Xp are the independent variables, β0 is the intercept, β1, β2, …, βn are the slopes. The goal of the algorithm is to find the best Fit Line equation that can predict the values based on the independent variables.

Program and Output:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
IceCream=pd.read_csv('/content/IceCreamData.csv')
print(IceCream)
```

```
     Temperature     Revenue
0      24.566884  534.799028
1      26.005191  625.190122
2      27.790554  660.632289
3      20.595335  487.706960
4      11.503498  316.240194
..           ...         ...
495    22.274899  524.746364
496    32.893092  755.818399
497    12.588157  306.090719
498    22.362402  566.217304
499    28.957736  655.660388

[500 rows x 2 columns]
```

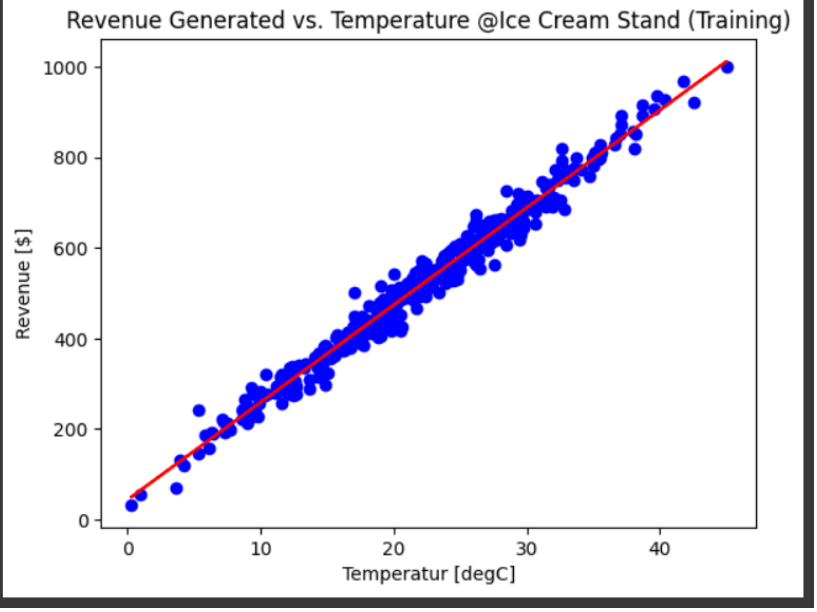
```
# Divide the data into “Attributes” and “labels”
X = IceCream[['Temperature']]
y = IceCream['Revenue']
# Split 80% of the data to the training set while 20% of the data to test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
# Create a Linear Regression model and fit it
regressor =LinearRegression(fit_intercept=True)
regressor.fit(X_train,y_train)
print('Linear Model Coeff (m) =' , regressor.coef_)
print('Linear Model Coeff (b) =' , regressor.intercept_)
```

```
# Predicting the data
y_predict=regressor.predict(X_test)
print(y_predict)
```

```
Linear Model Coeff (m) = [21.5133908]
Linear Model Coeff (b) = 43.73357869209326
[698.3385558  653.32331149 664.73027451 450.5192845  665.47469743
 441.36861407 584.06540609 623.82532723 667.48717467 468.72433832
 546.82733151 443.41191785 622.95162777 377.64639971 367.0607334
 945.67057977 893.79551974 694.45445099 546.05047608 420.58523672
 391.08500303 597.0141581  283.23582775 655.50055011 380.98796154
 412.31810124 371.05055651 510.23910289 479.70270426 456.68206658
 640.1157508  281.65224383 314.1894674  470.01363777 559.72453055
 539.75091165 307.72368191 508.65180339 571.43237276 732.25599161
 440.44010989 494.39422767 567.56536766 443.94181482 914.46632525
 603.19341879 541.83315574 199.94980451 694.04258508 351.09960842
 189.49123987 576.80689646 216.55393778 468.15141951 461.80905978
 448.43970076 494.89418532 801.3758273  331.24527072 540.42751209
 661.1953557  526.66690494 360.66507037 451.46656256 621.57729407
 254.83395119 290.1749214  525.5900171  656.68802152 663.1062835
 740.96627734 184.48524774 593.42653041 148.41501952 485.97744998
 611.03624804 664.50658946 473.51664017 785.34682628 422.11909846
 169.76879503 820.72328003 434.39990573 325.82688811 660.07484042
 586.46853445 415.89811147 651.95510136 865.22669518 265.88922879
 577.32110608  43.73357869 901.24435059 621.87554173 759.07316169
 465.78060018 758.74558525 711.30125473 394.45680968 559.53716333]
```

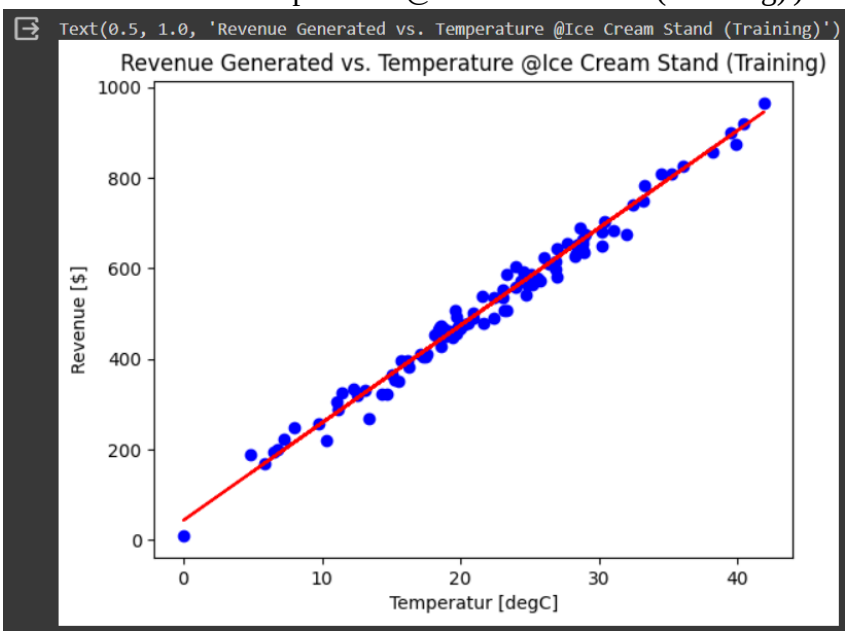
```
# Scatter plot on Training Data
plt.scatter(X_train,y_train,color='blue')
plt.plot(X_train,regressor.predict(X_train),color='red')
plt.ylabel('Revenue [$]')
plt.xlabel('Temperatur [degC]')
plt.title('Revenue Generated vs. Temperature @Ice Cream Stand (Training)')
```

```
Text(0.5, 1.0, 'Revenue Generated vs. Temperature @Ice Cream Stand (Training)')
```




```
# Scatter plot on Testing Data
plt.scatter(X_test,y_test,color='blue')
plt.plot(X_test,regressor.predict(X_test),color='red')
plt.ylabel('Revenue [$]')
plt.xlabel('Temperatur [degC]')
```

plt.title('Revenue Generated vs. Temperature @Ice Cream Stand (Training)')




```
# Prediction the revenve using Temperature Value directly
print('---------0---------')
Temp = -0
Revenue = regressor.predict([[Temp]])
print(Revenue)
print('--------35----------')
Temp = 35
Revenue = regressor.predict([[Temp]])
print(Revenue)
print('--------55----------')
Temp = 55
Revenue = regressor.predict([[Temp]])
print(Revenue)
```

```
---------0---------
[43.73357869]
--------35----------
[796.70225678]
--------55----------
[1226.97007282]
```

Week-5: Exercises to solve the real-world problems using Logistic Regression

Aim: To write a program to solve the real-world problem using Logistic Regression.

Description: Logistic regression is used for binary classification where we use sigmoid function, that takes input as independent variables and produces a probability value between 0 and 1. Logistic regression predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1. In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

Types of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

Binomial: In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.

Multinomial: In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep".

Ordinal: In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

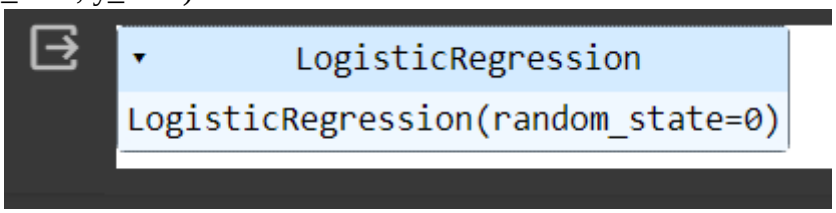
Program and Output:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset = pd.read_csv('/content/Social_Network_Ads.csv')
print(dataset)
```

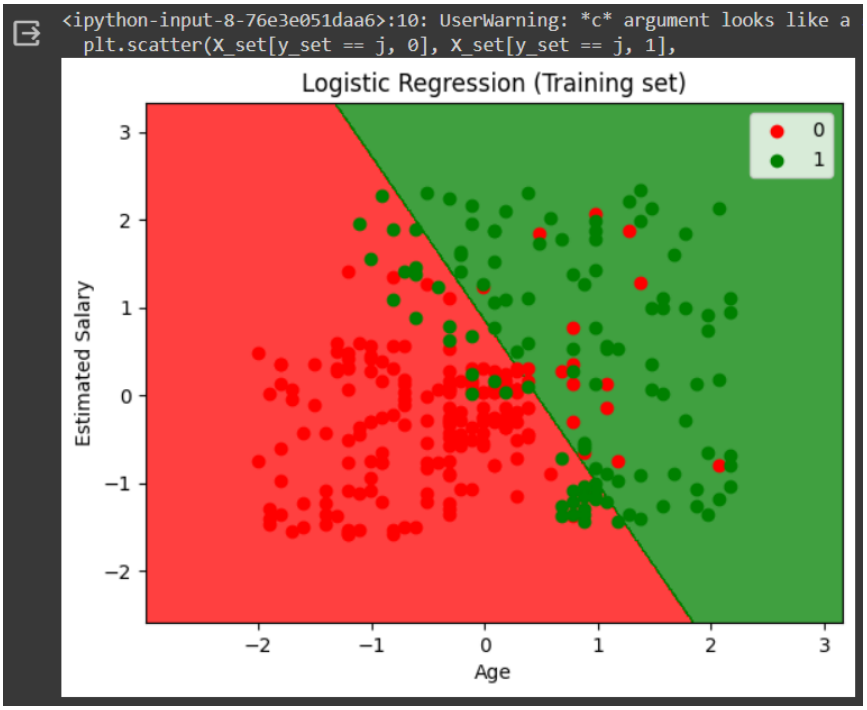
```
      User ID  Gender  Age  EstimatedSalary  Purchased
0    15624510    Male   19            19000          0
1    15810944    Male   35            20000          0
2    15668575  Female   26            43000          0
3    15603246  Female   27            57000          0
4    15804002    Male   19            76000          0
..        ...     ...  ...              ...        ...
395  15691863  Female   46            41000          1
396  15706071    Male   51            23000          1
397  15654296  Female   50            20000          1
398  15755018    Male   36            33000          0
399  15594041  Female   49            36000          1

[400 rows x 5 columns]
```

```
X = dataset.iloc[:, [2,3]].values
y = dataset.iloc[:, 4].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
```

```
X_test = sc_X.transform(X_test)
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)
```

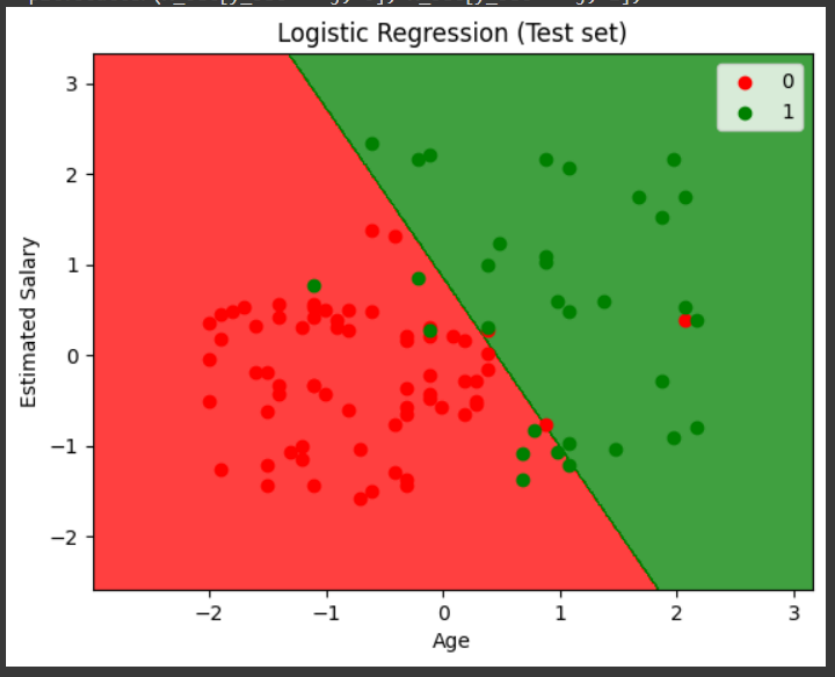



```
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step =
0.01))
plt.contourf(X1,                    X2,                    classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
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(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```




```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
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, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
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(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
<ipython-input-9-bfb0df682ba5>:10: UserWarning: *c* argument looks like a
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```

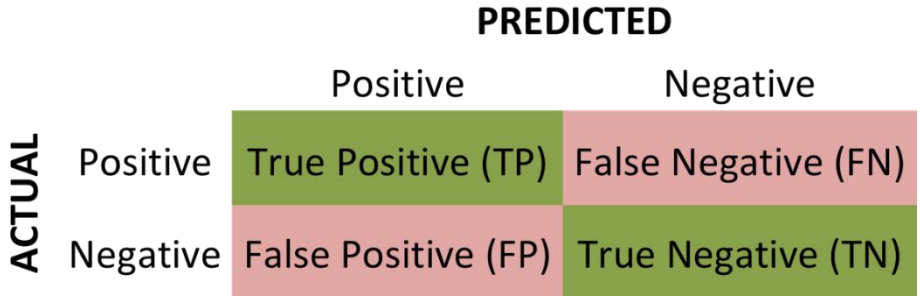


Week-6: Exercises to solve the real-world problems using Binary Classifier.

Aim: To write a program to solve the real-world problem using Binary Classifier.

Description: Binary Classification is a type of machine learning algorithm used to classify data into one of two categories. It predicts a binary outcome, where the result can either be positive or negative. For example, binary classification can be used to predict if a customer will buy a product or not, or if an email is spam or not. We can evaluate a binary classifier based on the following parameters:
True Positive (TP): The patient is diseased and the model predicts "diseased"
False Positive (FP): The patient is healthy but the model predicts "diseased"
True Negative (TN): The patient is healthy and the model predicts "healthy"
False Negative (FN): The patient is diseased and the model predicts "healthy"
After obtaining these values, we can compute the accuracy score of the binary classifier as follows:

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$



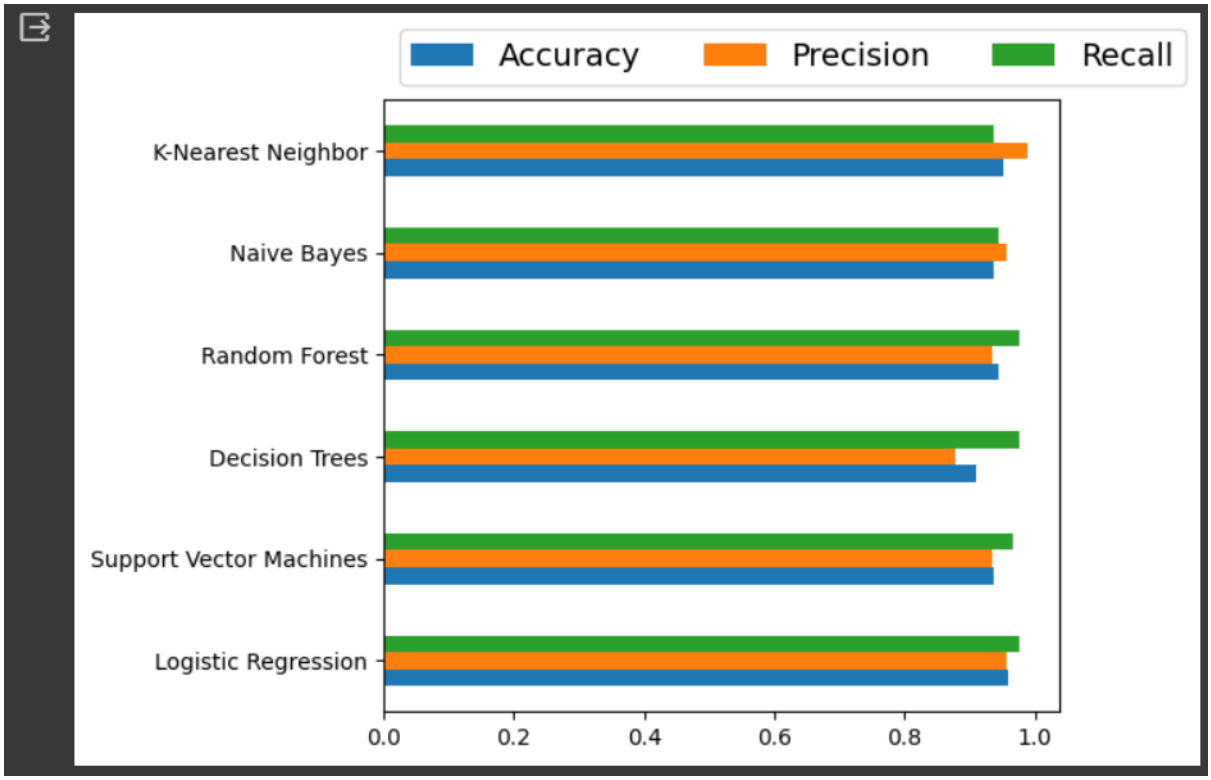
Program and Output:

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
dataset=load_breast_cancer(as_frame=True)
X=dataset['data']
y=dataset['target']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)
from sklearn.preprocessing import StandardScaler
ss_train=StandardScaler()
X_train=ss_train.fit_transform(X_train)
ss_test=StandardScaler()
X_test=ss_test.fit_transform(X_test)
models={}
# Logistic Regression
from sklearn.linear_model import LogisticRegression
models['Logistic Regression']=LogisticRegression()
# Support Vector Machines
```

```
from sklearn.svm import LinearSVC
models['Support Vector Machines']=LinearSVC()
# Decision Trees
from sklearn.tree import DecisionTreeClassifier
models['Decision Trees']=DecisionTreeClassifier()
# Random Forest
from sklearn.ensemble import RandomForestClassifier
models['Random Forest']=RandomForestClassifier()
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
models['Naive Bayes']=GaussianNB()
# K-Nearest Neighbors
from sklearn.neighbors import KNeighborsClassifier
models['K-Nearest Neighbor']=KNeighborsClassifier()
from sklearn.metrics import accuracy_score,precision_score,recall_score
accuracy,precision,recall={},{},{}
for key in models.keys():
  models[key].fit(X_train,y_train)
  predictions=models[key].predict(X_test)
  accuracy[key]=accuracy_score(predictions,y_test)
  precision[key]=precision_score(predictions,y_test)
  recall[key]=recall_score(predictions,y_test)
import pandas as pd
df_model=pd.DataFrame(index=models.keys(),columns=['Accuracy','Precision','Recall'])
df_model['Accuracy']=accuracy.values()
df_model['Precision']=precision.values()
df_model['Recall']=recall.values()
df_model
```

	Accuracy	Precision	Recall
Logistic Regression	0.958042	0.955556	0.977273
Support Vector Machines	0.937063	0.933333	0.965517
Decision Trees	0.909091	0.877778	0.975309
Random Forest	0.944056	0.933333	0.976744
Naive Bayes	0.937063	0.955556	0.945055
K-Nearest Neighbor	0.951049	0.988889	0.936842

```
ax=df_model.plot.barh()
ax.legend(
    ncol=len(models.keys()),
    bbox_to_anchor=(0,1),
    loc="lower left",
    prop={'size':14}
)
plt.tight_layout()
```



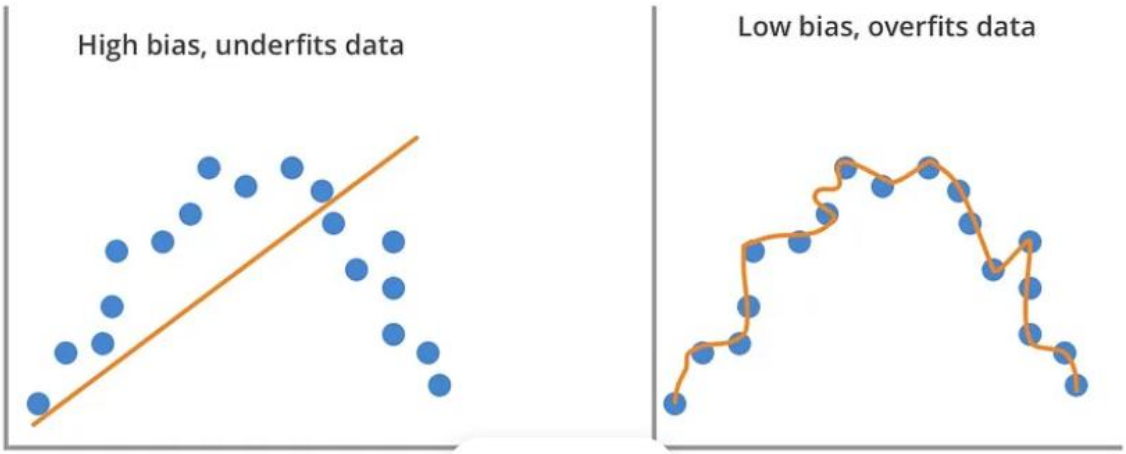

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,predictions)
TN,FP,FN,TP=confusion_matrix(y_test,predictions).ravel()
print("True Positive(TP):",TP)
print("False Positive(FP):",FP)
print("True Negative(TN):",TN)
print("False Negative(FN):",FN)
accuracy=(TP+TN)/(TP+FP+TN+FN)
print("Accuracy of the binary classifier = {:0.3f}".format(accuracy))
```

```
True Positive(TP): 89
False Positive(FP): 6
True Negative(TN): 47
False Negative(FN): 1
Accuracy of the binary classifier = 0.951
```

Week-7: Develop a program for Bias, Variance, Remove duplicates, Cross Validation
Aim: To develop a program for Bias, Variance, Remove duplicates, Cross Validation
Description:

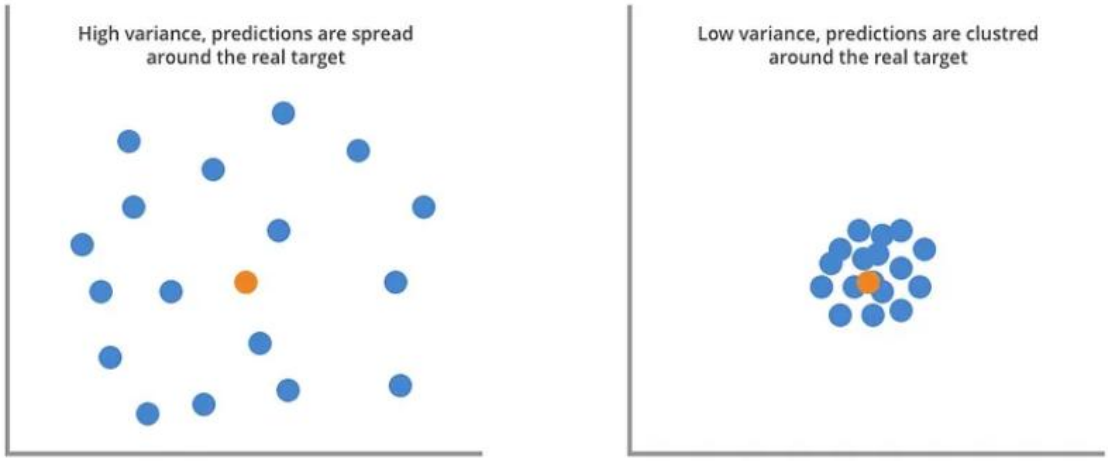
Bias

Bias is about the ability to capture the true patterns in the dataset.



Variance

Variance captures the range of predictions for each data record.



Cross Validation: Cross validation is a technique used in machine learning to evaluate the performance of a model on unseen data. It involves dividing the available data into multiple folds or subsets, using one of these folds as a validation set, and training the model on the remaining folds. This process is repeated multiple times, each time using a different fold as the validation set. Finally, the results from each validation step are averaged to produce a more robust estimate of the model's performance.

Program:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
```

```
iris_data = pd.read_csv("https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv")
print("Original dataset:")
print(iris_data.head())
# Remove duplicates
iris_data_no_duplicates = iris_data.drop_duplicates()
# Split dataset into features and target variable
X = iris_data_no_duplicates.drop(columns=['species'])
y = iris_data_no_duplicates['species']
# Convert categorical labels into numerical values
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
# Calculate bias and variance
bias = np.mean((y_test - y_pred) ** 2)
variance = np.mean(np.var(y_pred, axis=0))
print("Bias:", bias)
print("Variance:", variance)
# Cross-validation
scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
accuracy_cv = scores.mean()
print("Accuracy using cross-validation:", accuracy_cv)
```

Output:

```
Original dataset:
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
Bias: 0.06666666666666667
Variance: 0.6622222222222225
Accuracy using cross-validation: 0.9590804597701149
```

Week-8: Write a program to implement One-hot Encoding.

Aim: To write a program to implement One-hot Encoding.

Description: One-hot encoding is a technique in machine learning that turns categorical data, like colors (red, green, blue), into numerical data for machines to understand. It creates new binary columns for each category, with a 1 marking the presence of that category and 0s elsewhere. This allows machine learning algorithms to process the information in categorical data without misinterpreting any order between the categories. The advantages of using one hot encoding include:

- It allows the use of categorical variables in models that require numerical input.
- It can improve model performance by providing more information to the model about the categorical variable.
- It can help to avoid the problem of ordinality, which can occur when a categorical variable has a natural ordering (e.g. “small”, “medium”, “large”).

Program:

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
data = {'Employee id': [10, 20, 15, 25, 30],
        'Gender': ['M', 'F', 'F', 'M', 'F'],
        'Remarks': ['Good', 'Nice', 'Good', 'Great', 'Nice'],
        }
df = pd.DataFrame(data)
print(f"Employee data : \n{df}")
#Here we extract the columns with object datatype as they are the categorical columns
categorical_columns = df.select_dtypes(include=['object']).columns.tolist()
encoder = OneHotEncoder(sparse_output=False)
# Apply one-hot encoding to the categorical columns
one_hot_encoded = encoder.fit_transform(df[categorical_columns])
one_hot_df =pd.DataFrame(one_hot_encoded,
                         columns=encoder.get_feature_names_out(categorical_columns))
df_encoded = pd.concat([df, one_hot_df], axis=1)
df_encoded = df_encoded.drop(categorical_columns, axis=1)
print(f"Encoded Employee data : \n{df_encoded}")
```

Output:

```
Employee data :
   Employee id Gender Remarks
0           10      M    Good
1           20      F    Nice
2           15      F    Good
3           25      M   Great
4           30      F    Nice
Encoded Employee data :
   Employee id  Gender_F  Gender_M  Remarks_Good  Remarks_Great  Remarks_Nice
0           10       0.0       1.0           1.0            0.0           0.0
1           20       1.0       0.0           0.0            0.0           1.0
2           15       1.0       0.0           1.0            0.0           0.0
3           25       0.0       1.0           0.0            1.0           0.0
4           30       1.0       0.0           0.0            0.0           1.0
```

Week-9: Write a program to implement Categorical Encoding.

Aim: To write a program to implement Categorical Encoding.

Description:

A combination of numerical as well as categorical variables. A machine can only understand the numbers. It cannot understand the text.

Categorical Encoding: Categorical Encoding refers to the process of converting categorical (or qualitative) data into a numerical format that machine learning algorithms can understand. Categorical data represents categories or groups and does not have a natural numerical representation. In machine learning, it's common to encounter datasets with categorical features such as colors, shapes, cities, or labels.

Label Encoding: Label Encoding is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical order.

Program:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
data = {
    'Color': ['Red', 'Blue', 'Green', 'Red', 'Green'],
    'Size': ['Small', 'Large', 'Medium', 'Medium', 'Small'],
    'Shape': ['Circle', 'Square', 'Triangle', 'Circle', 'Square'],
    'Label': ['A', 'B', 'C', 'A', 'B']
}
df = pd.DataFrame(data)
print("Original dataset:")
print(df)
label_encoder = LabelEncoder()
df_encoded = df.copy()
for col in df.columns:
    if df[col].dtype == 'object':
        df_encoded[col] = label_encoder.fit_transform(df[col])
print("\nAfter Label Encoding:")
print(df_encoded)
```

Output:

```
Original dataset:
   Color    Size     Shape Label
0    Red   Small    Circle     A
1   Blue   Large    Square     B
2  Green  Medium  Triangle     C
3    Red  Medium    Circle     A
4  Green   Small    Square     B

After Label Encoding:
   Color  Size  Shape  Label
0      2     2      0      0
1      0     0      1      1
2      1     1      2      2
3      2     1      0      0
4      1     2      1      1
```

Week-10: Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

Aim: To build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.

Description: Backpropagation is an iterative algorithm, that helps to minimize the cost function by determining which weights and biases should be adjusted. During every epoch, the model learns by adapting the weights and biases to minimize the loss by moving down toward the gradient of the error. Thus, it involves the two most popular optimization algorithms, such as gradient descent or stochastic gradient descent.

Program:

```
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)
y = y/100
def sigmoid (x):          return 1/(1 + np.exp(-x))
def derivatives_sigmoid(x):    return x * (1 - x)
epoch=5000
lr=0.1
inputlayer_neurons = 2
hiddenlayer_neurons = 3
output_neurons = 1
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))
for i in range(epoch):
  hinp1=np.dot(X,wh);hinp=hinp1 + bh;hlayer_act = sigmoid(hinp);
  outinp1=np.dot(hlayer_act,wout);outinp= outinp1+ bout;output = sigmoid(outinp);
  EO = y-output;outgrad = derivatives_sigmoid(output);d_output = EO* outgrad;
  EH = d_output.dot(wout.T);hiddengrad = derivatives_sigmoid(hlayer_act);
  d_hiddenlayer = EH * hiddengrad;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:
 [[0.89473707]
 [0.87952092]
 [0.89547986]]
```

Week-11: Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

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

Description: KNN is one of the most basic yet essential classification algorithms in machine learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection. It is widely disposable in real-life scenarios since it is non-parametric, meaning it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data). We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

Program:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
iris=datasets.load_iris()
x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target,test_size=0.1)
for i in range(len(iris.target_names)):
  print("Label", i , "-",str(iris.target_names[i]))
classifier = KNeighborsClassifier(n_neighbors=2)
classifier.fit(x_train, y_train)
y_pred=classifier.predict(x_test)
print("Results of Classification using K-nn with K=1 ")
for r in range(0,len(x_test)):
  print(" Sample:", str(x_test[r]), " Actual-label:", str(y_test[r])," Predicted-label:", str(y_pred[r]))
  print("Classification Accuracy :" , classifier.score(x_test,y_test))
```

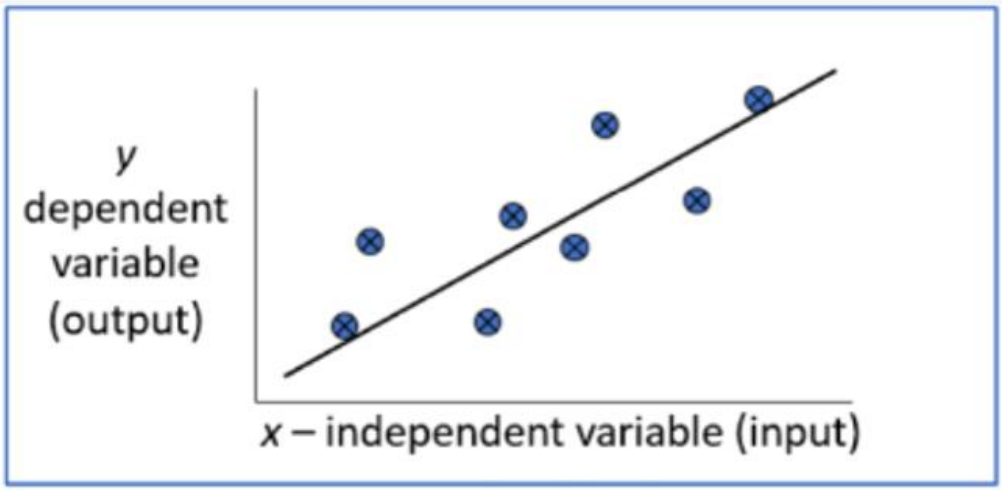
Output:

```
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
Results of Classification using K-nn with K=1
 Sample: [7.7 2.6 6.9 2.3]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.8666666666666667
 Sample: [4.8 3.1 1.6 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.8666666666666667
 Sample: [4.3 3.  1.1 0.1]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.8666666666666667
 Sample: [5.2 3.4 1.4 0.2]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.8666666666666667
 Sample: [6.5 3.  5.2 2. ]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.8666666666666667
 Sample: [5.5 2.4 3.7 1. ]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.8666666666666667
 Sample: [5.7 2.9 4.2 1.3]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.8666666666666667
 Sample: [6.4 2.9 4.3 1.3]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.8666666666666667
 Sample: [6.3 3.3 6.  2.5]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.8666666666666667
 Sample: [6.  2.7 5.1 1.6]  Actual-label: 1  Predicted-label: 2
Classification Accuracy : 0.8666666666666667
 Sample: [6.2 2.2 4.5 1.5]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.8666666666666667
 Sample: [5.  3.5 1.3 0.3]  Actual-label: 0  Predicted-label: 0
Classification Accuracy : 0.8666666666666667
 Sample: [6.  2.2 5.  1.5]  Actual-label: 2  Predicted-label: 1
Classification Accuracy : 0.8666666666666667
 Sample: [6.3 2.7 4.9 1.8]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.8666666666666667
 Sample: [5.5 2.3 4.  1.3]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.8666666666666667
```

Week-12: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

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

Description: LWLR manifests as a non-parametric regression algorithm that discerns the connection between a dependent variable and several independent variables. LWLR functions on the premise that the association between the dependent and independent variables adheres to linearity; however, this relationship is allowed to exhibit variability across distinct sections within the dataset. This is achieved by employing an individual linear regression model for each prediction, employing a weighted least squares technique.



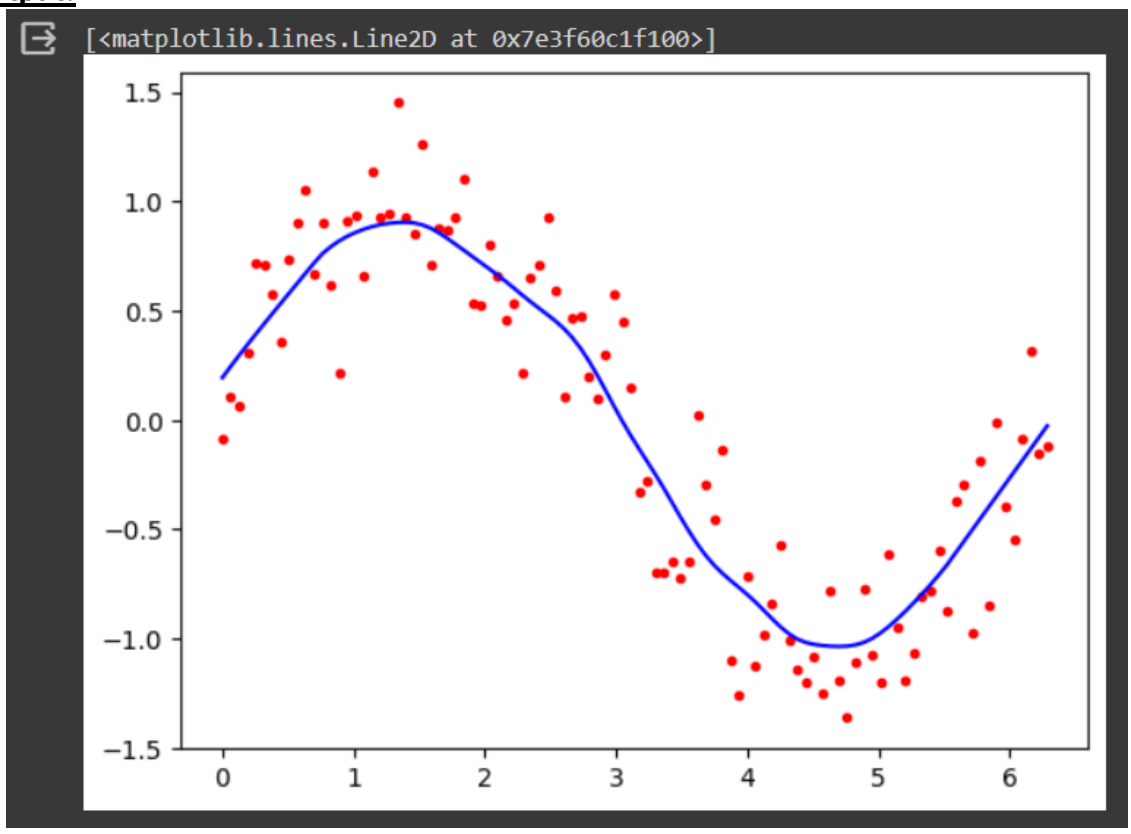
Loess Regression: Loess regression is a nonparametric technique that uses local weighted regression to fit a smooth curve through points in a scatter plot. Loess curves are can reveal trends and cycles in data that might be difficult to model with a parametric curve.

Program:

```
from math import ceil
import numpy as np
from scipy import linalg
def lowess(x, y, f, iterations):
  n = len(x)
  r = int(ceil(f * n))
  h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]
  w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)
  w = (1 - w ** 3) ** 3
  yest = np.zeros(n)
  delta = np.ones(n)
  for iteration in range(iterations):
    for i in range(n):
      weights = delta * w[:, i]
      b = np.array([np.sum(weights * y), np.sum(weights * y * x)])
```

```
    A = np.array([[np.sum(weights), np.sum(weights * x)],[np.sum(weights * x),
np.sum(weights * x * x)]])
      beta = linalg.solve(A, b)
      yest[i] = beta[0] + beta[1] * x[i]
    residuals = y - yest
    s = np.median(np.abs(residuals))
    delta = np.clip(residuals / (6.0 * s), -1, 1)
    delta = (1 - delta ** 2) ** 2
  return yest
import math
import matplotlib.pyplot as plt
n = 100
x = np.linspace(0, 2 * math.pi, n)
y = np.sin(x) + 0.3 * np.random.randn(n)
f = 0.25
iterations=3
yest = lowess(x, y, f, iterations)
plt.plot(x,y,"r.")
plt.plot(x,yest,"b-")
```

Output:



Augmented Experiments

Experiment-13: 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.

Aim: 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.

Program:

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']
twenty_train = fetch_20newsgroups (subset='train', categories=categories, shuffle=True)
twenty_test = fetch_20newsgroups (subset='test', categories=categories, shuffle=True)
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_tf = count_vect.fit_transform(twenty_train.data)
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_tf)
X_train_tfidf.shape
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn import metrics
mod = MultinomialNB()
mod.fit(X_train_tfidf, twenty_train.target)
X_test_tf = count_vect.transform(twenty_test.data)
X_test_tfidf = tfidf_transformer.transform(X_test_tf)
predicted = mod.predict(X_test_tfidf)
print("Accuracy:", accuracy_score(twenty_test.target, predicted))
print(classification_report(twenty_test.target, predicted,
target_names=twenty_test.target_names))
print("confusion matrix is \n", metrics.confusion_matrix(twenty_test.target, predicted))
```

Output:

```
Accuracy: 0.8348868175765646
                        precision    recall  f1-score   support

           alt.atheism       0.97      0.60      0.74       319
         comp.graphics       0.96      0.89      0.92       389
               sci.med       0.97      0.81      0.88       396
soc.religion.christian       0.65      0.99      0.78       398

              accuracy                           0.83      1502
             macro avg       0.89      0.82      0.83      1502
          weighted avg       0.88      0.83      0.84      1502

confusion matrix is
 [[192   2   6 119]
 [  2 347   4  36]
 [  2  11 322  61]
 [  2   2   1 393]]
```

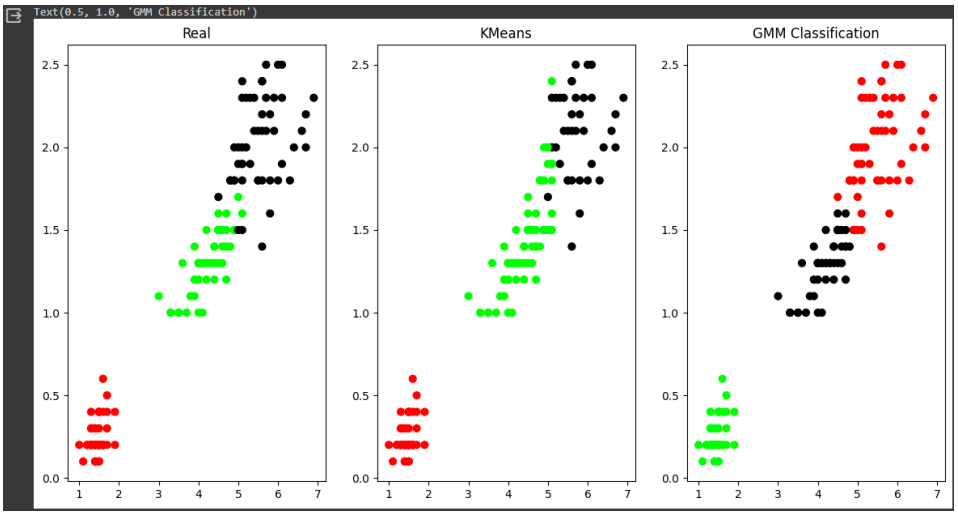
Experiment-14: Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

Aim: 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.

Program:

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
from sklearn.mixture import GaussianMixture
from sklearn.datasets import load_iris
import sklearn.metrics as sm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dataset=load_iris()
X=pd.DataFrame (dataset.data)
X.columns=['Sepal_Length', 'Sepal_width', 'Petal_Length', 'Petal_width']
y=pd.DataFrame(dataset.target)
y.columns=['Targets']
plt.figure(figsize=(14,7))
colormap=np.array(['red', 'lime', 'black'])
plt.subplot(1,3,1)
plt.scatter(X.Petal_Length,X.Petal_width,c=colormap[y.Targets],s=40)
plt.title('Real')
plt.subplot(1,3,2)
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
plt.scatter(X.Petal_Length, X. Petal_width, c=colormap[predY], s=40)
plt.title('KMeans')
scaler=preprocessing.StandardScaler()
scaler.fit(X)
xsa=scaler.transform(X)
xs=pd.DataFrame(xsa,columns=X.columns)
gmm=GaussianMixture(n_components=3)
gmm.fit(xs)
y_cluster_gmm=gmm.predict(xs)
plt.subplot(1,3,3)
plt.scatter(X.Petal_Length, X.Petal_width, c=colormap[y_cluster_gmm],s=40)
plt.title('GMM Classification')
```

Output:


Text(0.5, 1.0, 'GMM Classification')
Real
KMeans
GMM Classification