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.

Aim: To implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

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.

Important Representation:

- ? indicates that any value is acceptable for the attribute.
- specify a single required value (e.g., Cold) for the attribute.
- ϕ indicates that no value is acceptable.
- The most general hypothesis is represented by: {?, ?, ?, ?, ?, ?}
- The most specific hypothesis is represented by: {ϕ, ϕ, ϕ, ϕ, ϕ, ϕ}

Steps involved in FIND-S algorithm:

- Start with the most specific hypothesis.
  h = {ϕ, ϕ, ϕ, ϕ, ϕ, ϕ}
- Take the next example and if it is negative, then no changes occur to the hypothesis.
- If the example is positive and we find that our initial hypothesis is too specific then we update our current hypothesis to a general condition.

Training Data:

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('/content/weather 4.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(0, len(a)):
  if a[i][num_attribute] == 'yes':
    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  traininginstance is ")
print(hypothesis)
```

Output:

```
 The total number of training instances are :  5

 The initial hypothesis is :
['0', '0', '0', '0', '0', '0']

 The hypothesis for the training instance 1 is :
 ['0', '0', '0', '0', '0', '0']

 The hypothesis for the training instance 2 is :
 ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

 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']

 The hypothesis for the training instance 5 is :
 ['sunny', 'warm', '?', 'strong', '?', '?']

  The  Maximally  specific  hypothesis  for  the  traininginstance 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.

Aim: To give 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.

Description:

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

Initialize G to the set of maximally general hypotheses in H

Initialize S to the set of maximally specific hypotheses in H

For each training example d, do

- If d is a positive example
  - Remove from G any hypothesis inconsistent with d
  - For each hypothesis s in S that is not consistent with d
    - Remove s from S
    - Add to S all minimal generalizations h of s such that
      - ❖ h is consistent with d, and some member of G is more general than h
    - Remove from S any hypothesis that is more general than another hypothesis in S
- If d is a negative example
  - Remove from S any hypothesis inconsistent with d
  - For each hypothesis g in G that is not consistent with d
    - Remove g from G
    - Add to G all minimal specializations h of g such that
      - ❖ h is consistent with d, and some member of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

Training Data:

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('/content/weather 4.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] ='?'
        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(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']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
 steps of Candidate Elimination Algorithm 2
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
['sunny' 'warm' '?' 'strong' 'warm' 'same']
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' '?' 'strong' 'warm' 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' '?' 'same']
['sunny' 'warm' '?' 'strong' '?' '?']
['sunny' 'warm' '?' 'strong' '?' '?']
 steps of Candidate Elimination Algorithm 4
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

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.

Aim: 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:

Decision Tree

- The Decision Tree Basically is an inverted tree, with each node representing features and attributes.
- While the leaf node represents the output
- Except for the leaf node, the remaining nodes act as decision making nodes.
- Basic Terms in Decision tree

Algorithms

- CART (Gini Index)
- ID3 (Entropy, Information Gain)

Note :- Here we will understand the ID3 algorithm

Algorithm Concepts

1. To understand this concept, we take an example, assuming we have a data set .
2. Based on this data, we have to find out if we can play someday or not.
3. We have four attributes in the data-set. Now how do we decide which attribute we should put on the root node?
4. For this, we will Calculate the information gain of all the attributes (Features), which will have maximum information will be our root node.

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: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 Data:

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:

```
[48] import pandas as pd
     from pandas import DataFrame
     df_tennis = pd.read_csv('/book.csv')
     print( df_tennis)
```

Output:

```
    Day   Outlook Temperature Humidity    Wind PlayTennis
0    D1     Sunny         Hot     High    Weak         No
1    D2     Sunny         Hot     High  Strong         No
2    D3  Overcast         Hot     High    Weak        Yes
3    D4      Rain        Mild     High    Weak        Yes
4    D5      Rain        Cool   Normal    Weak        Yes
5    D6      Rain        Cool   Normal  Strong         No
6    D7  Overcast        Cool   Normal  Strong        Yes
7    D8     Sunny        Mild     High    Weak         No
8    D9     Sunny        Cool   Normal    Weak        Yes
9   D10      Rain        Mild   Normal    Weak        Yes
10  D11     Sunny        Mild   Normal  Strong        Yes
11  D12  Overcast        Mild     High  Strong        Yes
12  D13  Overcast         Hot   Normal    Weak        Yes
13  D14      Rain        Mild     High  Strong         No
```

Program:

```
[61] def entropy(probs):
         import math
         return sum( [-prob*math.log(prob, 2) for prob in probs] )
     def entropy_of_list(a_list):
         from collections import Counter
         cnt = Counter(x for x in a_list)
         num_instances = len(a_list)*1.0
         print("\n Number of Instances of the Current Sub Class is {0}:".format(num_instances ))
         probs = [x / num_instances for x in cnt.values()]
         print("\n Classes:",min(cnt),max(cnt))
         print(" \n Probabilities of Class {0} is {1}:".format(min(cnt),min(probs)))
         print(" \n Probabilities of Class {0} is {1}:".format(max(cnt),max(probs)))
         return entropy(probs)
     print("\n  INPUT DATA SET FOR ENTROPY CALCULATION:\n", df_tennis['PlayTennis'])
     total_entropy = entropy_of_list(df_tennis['PlayTennis'])
```

Output:

```
  INPUT DATA SET FOR ENTROPY CALCULATION:
 0       No
1       No
2      Yes
3      Yes
4      Yes
5       No
6      Yes
7       No
8      Yes
9      Yes
10     Yes
11     Yes
12     Yes
13      No
Name: PlayTennis, dtype: object

 Number of Instances of the Current Sub Class is 14.0:

 Classes: No Yes

 Probabilities of Class No is 0.35714285714285715:

 Probabilities of Class Yes is 0.6428571428571429:
```

Program:

```
[65] def information_gain(df, split_attribute_name, target_attribute_name, trace=0):
         print("Information Gain Calculation of ",split_attribute_name)
         df_split = df.groupby(split_attribute_name)
         nobs = len(df.index) * 1.0
         df_agg_ent = df_split.agg({target_attribute_name : [entropy_of_list, lambda x: len(x)/nobs] })[target_attribute_name]
         df_agg_ent.columns = ['Entropy', 'PropObservations']
         new_entropy = sum( df_agg_ent['Entropy'] * df_agg_ent['PropObservations'] )
         old_entropy = entropy_of_list(df[target_attribute_name])
         return old_entropy - new_entropy
```

```
[64] def id3(df, target_attribute_name, attribute_names, default_class=None):
         from collections import Counter
         cnt = Counter(x for x in df[target_attribute_name])
         if len(cnt) == 1:
             return next(iter(cnt))
         elif df.empty or (not attribute_names):
             return default_class
         else:
             default_class = max(cnt.keys())
             gainz = [information_gain(df, attr, target_attribute_name) for attr in attribute_names]
             index_of_max = gainz.index(max(gainz))
             best_attr = attribute_names[index_of_max]
             tree = {best_attr:{}}
             remaining_attribute_names = [i for i in attribute_names if i != best_attr]
             for attr_val, data_subset in df.groupby(best_attr):
                 subtree = id3(data_subset,
                               target_attribute_name,
                               remaining_attribute_names,
                               default_class)
                 tree[best_attr][attr_val] = subtree
             return tree
```

```
[68] attribute_names = list(df_tennis.columns)
     print("List of Attributes:", attribute_names)
     attribute_names.remove('PlayTennis') #Remove the class attribute
     print("Predicting Attributes:", attribute_names)
```

```
List of Attributes: ['Day', 'Outlook', 'Temperature', 'Humidity', 'Wind', 'PlayTennis']
Predicting Attributes: ['Day', 'Outlook', 'Temperature', 'Humidity', 'Wind']
```

```
[77] def classify(instance, tree, default=None):
         attribute = next(iter(tree))
         if instance[attribute] in tree[attribute].keys():
             result = tree[attribute][instance[attribute]]
             if isinstance(result, dict):
                 return classify(instance, result)
             else:
                 return result
         else:
             return default
     df_tennis['predicted'] = df_tennis.apply(classify, axis=1, args=(tree,'No') )
     print(df_tennis['predicted'])
     df_tennis[['PlayTennis', 'predicted']]
```

Output:

```
0     No
1     No
2    Yes
3    Yes
4    Yes
5     No
6    Yes
7     No
8    Yes
9    Yes
10   Yes
11   Yes
12   Yes
13    No
Name: predicted, dtype: object
```

	PlayTennis	predicted
0	No	No
1	No	No
2	Yes	Yes
3	Yes	Yes
4	Yes	Yes
5	No	No
6	Yes	Yes
7	No	No
8	Yes	Yes
9	Yes	Yes
10	Yes	Yes
11	Yes	Yes
12	Yes	Yes
13	No	No

Program:

```
training_data = df_tennis.iloc[1:-4]
test_data  = df_tennis.iloc[-4:]
train_tree = id3(training_data, 'PlayTennis', attribute_names)

test_data['predicted2'] = test_data.apply(
                                    classify,
                                    axis=1,
                                    args=(train_tree,'Yes') )


print ('\n\n Accuracy is : ' + str( sum(test_data['PlayTennis']==test_data['predicted2'] ) / (1.0*len(test_data.index)) ))
```

Output:

```
 Accuracy is : 0.75
```

Experiment – 4

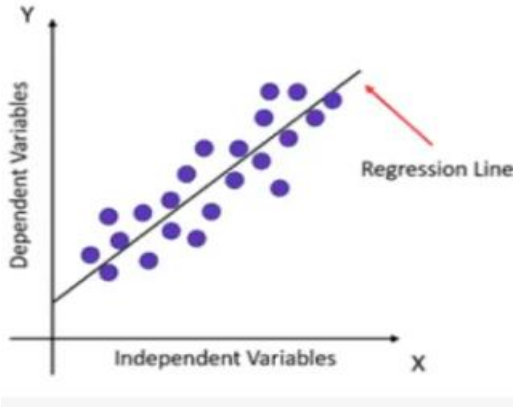
Exercises to solve the real-world problems using Linear Regression

Aim: Exercises to solve the real-world problems using Linear Regression with python.

Description:

Linear Regression is an algorithm that belongs to supervised Machine Learning. It tries to apply relations that will predict the outcome of an event based on the independent variable data points. The relation is usually a straight line that best fits the different data points as close as possible. The output is of a continuous form, i.e., numerical value. For example, the output could be revenue or sales in currency, the number of products sold, etc. In the above example, the independent variable can be single or multiple.

Linear Regression Equation



Linear regression can be expressed mathematically as:

$y= \beta 0+ \beta\ 1x+ \varepsilon$

Here,

- Y= Dependent Variable
- X= Independent Variable
- β 0= intercept of the line
- $\beta 1$ = Linear regression coefficient (slope of the line)
- ε = random error

Linear Regression Model

Since the Linear Regression algorithm represents a linear relationship between a dependent (y) and one or more independent (y) variables, it is known as Linear Regression. This means it finds how the value of the dependent variable changes according to the change in the value of the independent variable. The relation between independent and dependent variables is a straight line with a slope.

Types of Linear Regression

- Simple Linear Regression
- Multiple Linear Regression
- Non - Linear Regression

Training Data:

Car_Name	Year	Selling	Present	Kms	Fuel	Seller	Transmission	Owner
bugati	2014	200	150	26000	Petrol	Individual	Manual	2
shift	2007	6	3	40000	CNG	Individual	Manual	4
audi	2020	80	59	30000	Diesel	Dealer	Manual	4
kicks	2022	20	15	24000	Petrol	Dealer	Automatic	0
bmw	2016	90	50	50000	Diesel	Dealer	Automatic	6
ritz	2014	9	5	44000	Petrol	Individual	Manual	3
rolls royce	2006	300	250	2000	CNG	Dealer	Automatic	0
range rover	2021	45	30	10000	Diesel	Dealer	Automatic	4
nano	2008	3	2	5000	CNG	Individual	Automatic	0
harrier	2022	40	40	60000	Diesel	Individual	Automatic	0

Program:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
book = pd.read_csv('/content/Book car.csv')
print(book)
book.info()
```

Output:

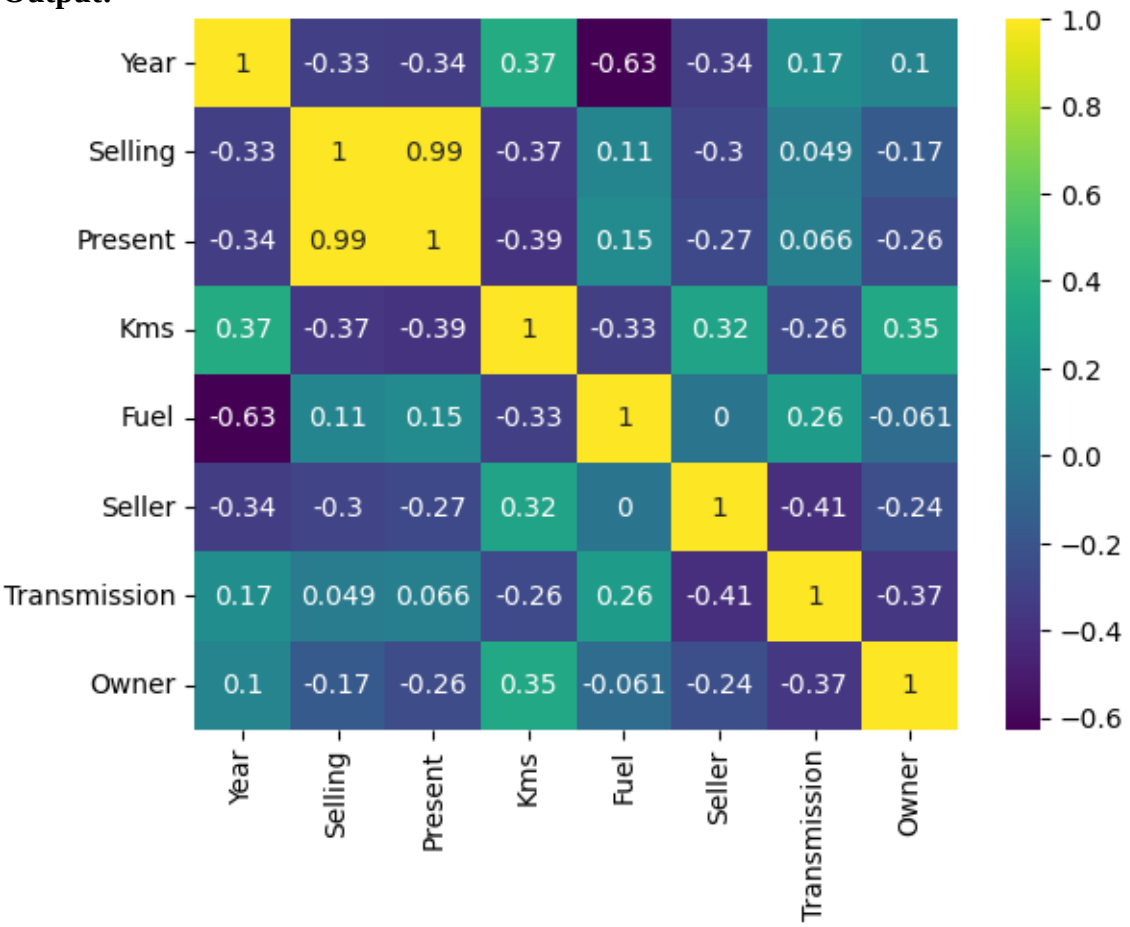
```
      Car_Name  Year  Selling  Present     Kms    Fuel       Seller
0       bugati  2014      200      150   26000  Petrol   Individual
1        shift  2007        6        3   40000     CNG   Individual
2         audi  2020       80       59   30000  Diesel       Dealer
3        kicks  2022       20       15   24000  Petrol       Dealer
4          bmw  2016       90       50   50000  Diesel       Dealer
5         ritz  2014        9        5   44000  Petrol   Individual
6  rolls royce  2006      300      250    2000     CNG       Dealer
7  range rover  2021       45       30   10000  Diesel       Dealer
8         nano  2008        3        2    5000     CNG   Individual
9      harrier  2022       40       40   60000  Diesel   Individual

  Transmission  Owner
0       Manual      2
1       Manual      4
2       Manual      4
3    Automatic      0
4    Automatic      6
5       Manual      3
6    Automatic      0
7    Automatic      4
8    Automatic      0
9    Automatic      0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Car_Name       10 non-null     object
 1   Year           10 non-null     int64
 2   Selling        10 non-null     int64
 3   Present        10 non-null     int64
 4   Kms            10 non-null     int64
 5   Fuel           10 non-null     object
 6   Seller         10 non-null     object
 7   Transmission   10 non-null     object
 8   Owner          10 non-null     int64
dtypes: int64(5), object(4)
memory usage: 848.0+ bytes
```

Program:

```
[24] book.replace({'Fuel':{'Petrol':0,'Diesel':1,'CNG':2}},inplace=True)
     book.replace({'Seller':{'Dealer':0,'Individual':1}},inplace=True)
     book.replace({'Transmission':{'Manual':0,'Automatic':1}},inplace=True)
```

```
[25] corrMatrix = book.corr()
     sns.heatmap(corrMatrix, annot=True, cmap="viridis")
     plt.show()
```

Output:



Program:

```
[30] X = book.drop(['Car_Name','Selling'],axis=1)
     Y = book['Selling']
     X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=42)
     lin_reg_model = LinearRegression()
     lin_reg_model.fit(X_train,Y_train)
     training_data_prediction = lin_reg_model.predict(X_train)
     train_error_score = metrics.r2_score(Y_train, training_data_prediction)
     print("R squared Error - Training : ", train_error_score)
```

Output:

```
R squared Error - Training :  1.0
```

Program:

```
Y_pred = lin_reg_model.predict(X_test)
test_error_score = metrics.r2_score(Y_test, Y_pred)
print("R squared Error - Test: ", test_error_score)
```

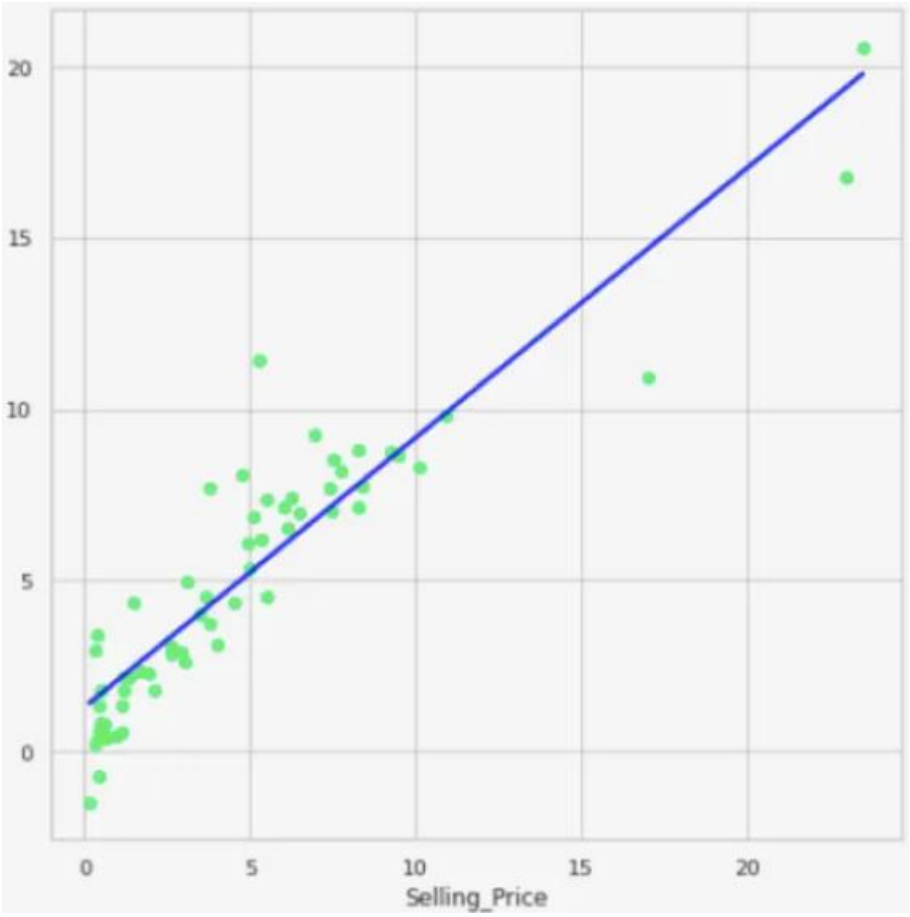
Output:

```
R squared Error - Test:  -769.1751803526794
```

Program:

```
[33] sns.regplot(Y_test, Y_pred,scatter_kws={"color": "green"}, line_kws={"color": "blue"})
```

Output:



Experiment – 5

Exercises to solve the real-world problems using Logistic Regression

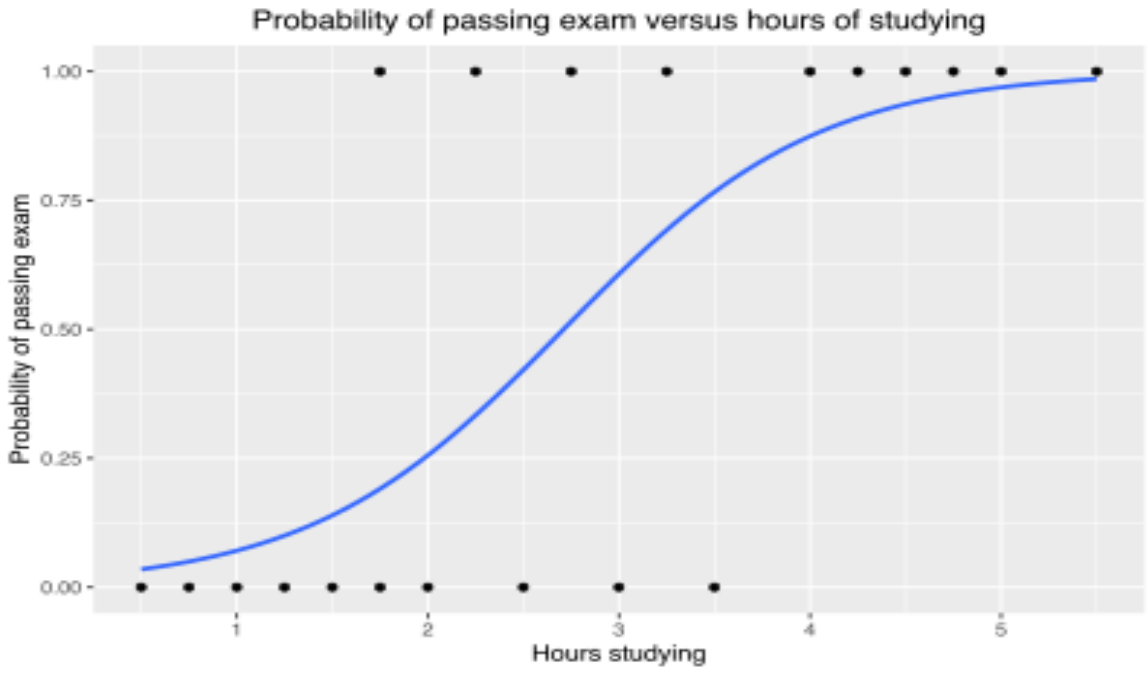
Aim: Exercises to solve the real-world problems using Logistic Regression

Description:

Logistic regression is a fundamental classification technique. It belongs to the group of linear classifiers and is somewhat similar to polynomial and **linear regression**. Logistic regression is fast and relatively uncomplicated, and it's convenient for you to interpret the results. Although it's essentially a method for binary classification, it can also be applied to multiclass problems. This type of statistical model (also known as *logit model*) is often used for classification and predictive analytics. Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1. In logistic regression, a logit transformation is applied on the odds—that is, the probability of success divided by the probability of failure. This is also commonly known as the log odds, or the natural logarithm of odds, and this logistic function is represented by the following formulas:

Logit(pi) = 1/(1+ exp(-pi))

ln(pi/(1-pi)) = Beta_0 + Beta_1*X_1 + … + B_k*K_k



Program:

```
[5]  import matplotlib.pyplot as plt
     import numpy as np
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import classification_report, confusion_matrix
     x = np.arange(10).reshape(-1, 1)
     y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
     print("x")
     print(x)
     print("y")
     print(y)
```

Output:

```
x
[[0]
 [1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]
 [9]]
y
[0 0 0 0 1 1 1 1 1 1]
```

Program:

```
[12] model = LogisticRegression(solver='liblinear', random_state=0)
     model.fit(x, y)
     LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                        intercept_scaling=1, l1_ratio=None, max_iter=100,
                        multi_class='warn', n_jobs=None, penalty='l2',
                        random_state=0, solver='liblinear', tol=0.0001, verbose=0,
                        warm_start=False)
     model = LogisticRegression(solver='liblinear', random_state=0).fit(x, y)
     model.classes_

     array([0, 1])

[13] model.intercept_

     array([-1.04608067])

[14] model.coef_

     array([[0.51491375]])

[15] model.predict_proba(x)

     array([[0.74002157, 0.25997843],
            [0.62975524, 0.37024476],
            [0.5040632 , 0.4959368 ],
            [0.37785549, 0.62214451],
            [0.26628093, 0.73371907],
            [0.17821501, 0.82178499],
            [0.11472079, 0.88527921],
            [0.07186982, 0.92813018],
            [0.04422513, 0.95577487],
            [0.02690569, 0.97309431]])

[16] model.predict(x)

     array([0, 0, 0, 1, 1, 1, 1, 1, 1, 1])
```

```
[17] model.score(x, y)
```

```
0.9
```

```
[18] confusion_matrix(y, model.predict(x))
```

```
array([[3, 1],
       [0, 6]])
```

```
cm = confusion_matrix(y, model.predict(x))

fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cm)
ax.grid(False)
ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted 0s', 'Predicted 1s'))
ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual 0s', 'Actual 1s'))
ax.set_ylim(1.5, -0.5)
for i in range(2):
    for j in range(2):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='red')
plt.show()
```

Output:

Program:

```
[22] print(classification_report(y, model.predict(x)))
```

Output:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	4
1	1.00	1.00	1.00	6
accuracy			1.00	10
macro avg	1.00	1.00	1.00	10
weighted avg	1.00	1.00	1.00	10

Experiment – 6

Exercises to solve the real-world problems using Binary Classifier

Aim: Exercises to solve the real-world problems using Binary Classifier with python.

Description:

Binary Classifier

Binary classification refers to those classification tasks that have two class labels.
Examples include:

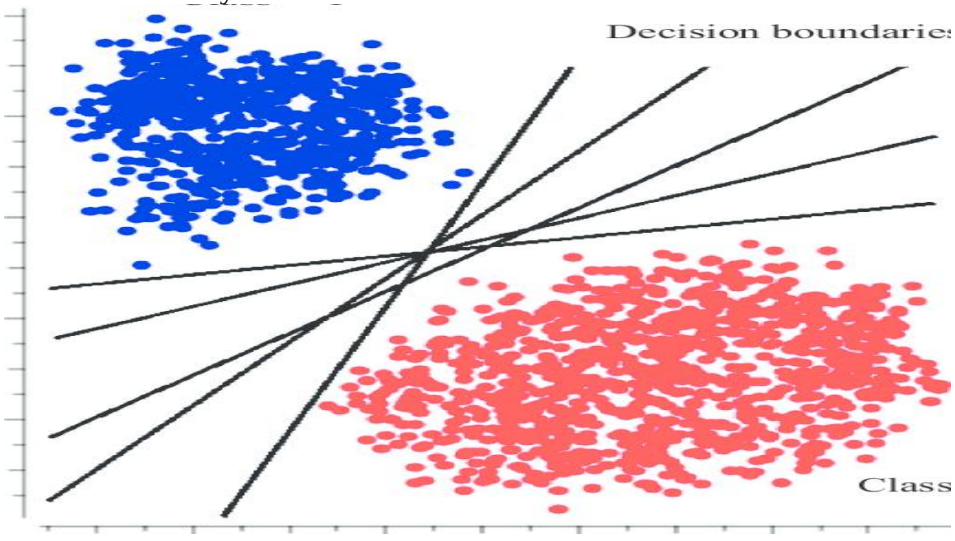
- Email spam detection (spam or not).
- Churn prediction (churn or not).
- Conversion prediction (buy or not).

Typically, binary classification tasks involve one class that is the normal state and another class that is the abnormal state.

For example, "*not spam*" is the normal state and "*spam*" is the abnormal state. Another example is "*cancer not detected*" is the normal state of a task that involves a medical test and "*cancer detected*" is the abnormal state.
It is common to model a binary classification task with a model that predicts a Bernoulli probability distribution for each example.
The Bernoulli distribution is a discrete probability distribution that covers a case where an event will have a binary outcome as either a 0 or 1.

Popular algorithms that can be used for binary classification include:

- Logistic Regression
- k-Nearest Neighbors
- Decision Trees
- Support Vector Machine
- Naive Bayes

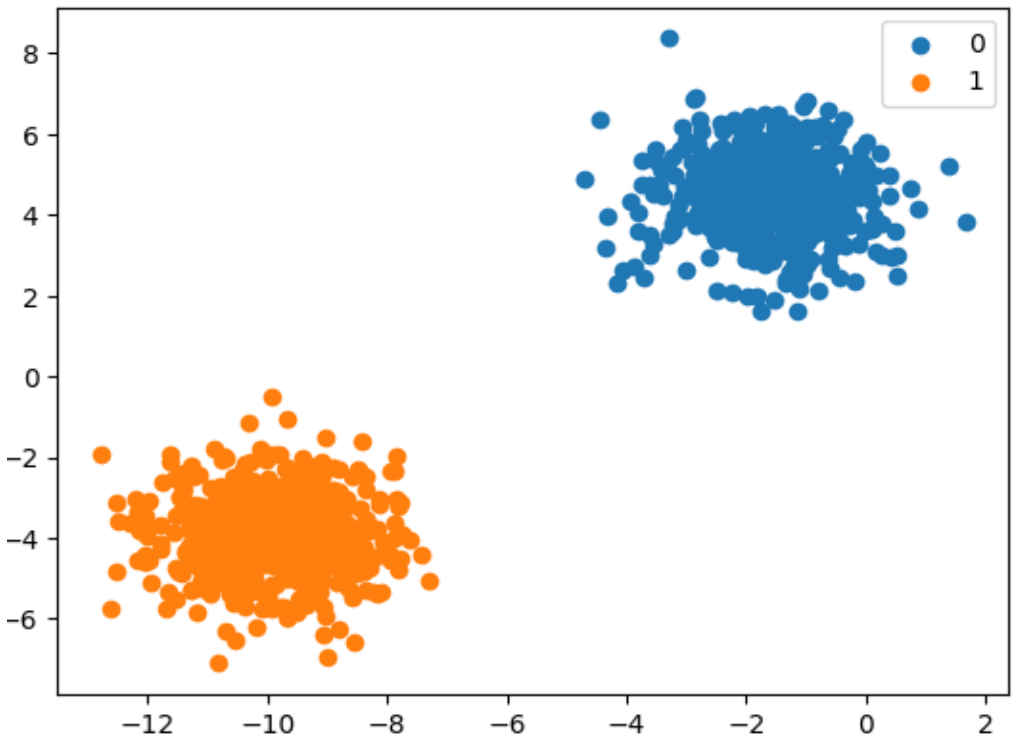


Program:

```
[16] from numpy import where
     from collections import Counter
     from sklearn.datasets import make_blobs
     from matplotlib import pyplot
     X, y = make_blobs(n_samples=1000, centers=2, random_state=1)
     print(X.shape, y.shape)
     counter = Counter(y)
     print(counter)
     for i in range(10):
       print(X[i], y[i])
     for label, _ in counter.items():
       row_ix = where(y == label)[0]
       pyplot.scatter(X[row_ix, 0], X[row_ix, 1], label=str(label))
     pyplot.legend()
     pyplot.show()
```

Output:

```
(1000, 2) (1000,)
Counter({0: 500, 1: 500})
[-3.05837272  4.48825769] 0
[-8.60973869 -3.72714879] 1
[1.37129721 5.23107449] 0
[-9.33917563 -2.9544469 ] 1
[-11.57178593  -3.85275513] 1
[-11.42257341  -4.85679127] 1
[-10.44518578  -3.76476563] 1
[-10.44603561  -3.26065964] 1
[-0.61947075  3.48804983] 0
[-10.91115591  -4.5772537 ] 1
```



Experiment – 7

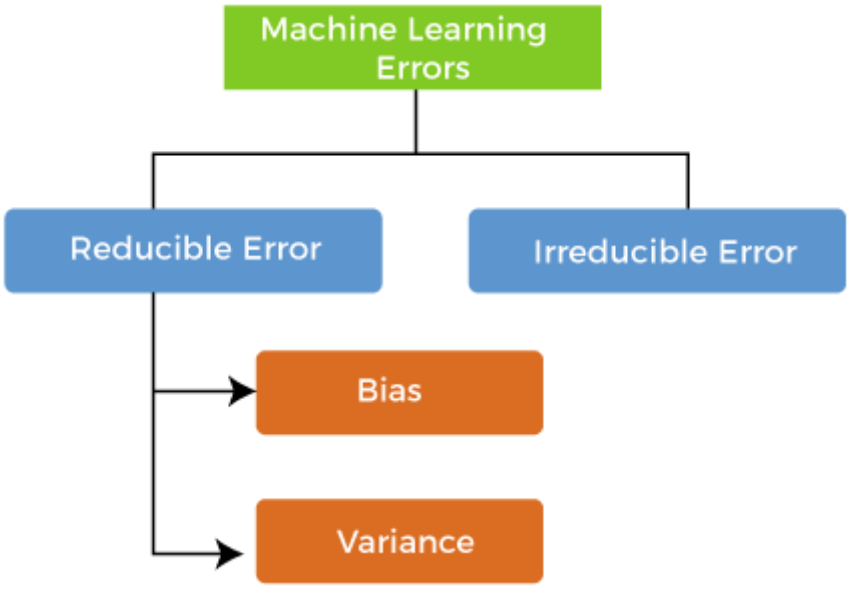
Develop a program for Bias, Variance, Remove duplicates, Cross Validation

Aim: Develop a program for Bias, Variance, Remove duplicates, Cross Validation using python.

Description:

In machine learning, an error is a measure of how accurately an algorithm can make predictions for the previously unknown dataset. On the basis of these errors, the machine learning model is selected that can perform best on the particular dataset. There are mainly two types of errors in machine learning, which are:

- **Reducible errors:** These errors can be reduced to improve the model accuracy. Such errors can further be classified into bias and Variance.
- **Irreducible errors:** These errors will always be present in the model



1.Bias:

While making predictions, a difference occurs between prediction values made by the model and actual values/expected values, and this difference is known as bias errors or Errors due to bias.

- **Low Bias:** A low bias model will make fewer assumptions about the form of the target function.
- **High Bias:** A model with a high bias makes more assumptions, and the model becomes unable to capture the important features of our dataset. **A high bias model also cannot perform well on new data.**

2.Variance Error:

The variance would specify the amount of variation in the prediction if the different training data was used. In simple words, variance tells that how much a random variable is different from its expected value. Ideally, a model should not vary too much from one training dataset to another, which means the algorithm should be good in understanding the hidden mapping between inputs and output variables. Variance errors are either of low variance or high variance. Low variance means there is a small variation in the prediction of the target function with changes in the training data set. At the same time, High variance shows a large variation in the prediction of the target function with changes in the training dataset.

3.Removing Duplicates

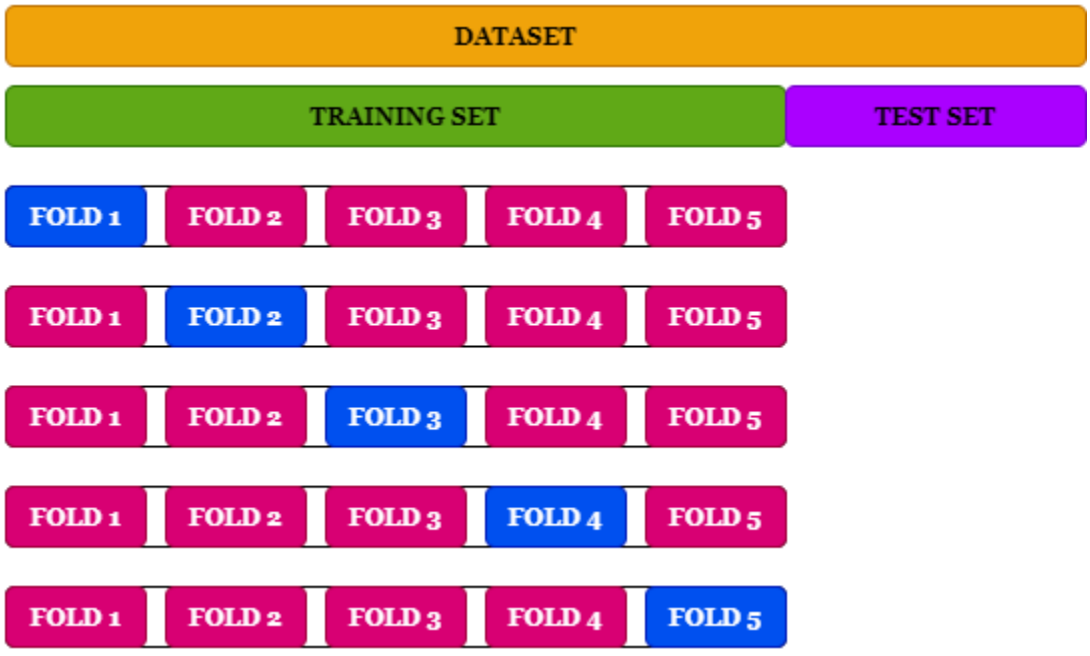
Duplicate entries are problematic for multiple reasons. An entry appearing more than once receives disproportionate weight during training. Models that succeed on frequent entries only *look* like they perform well. Duplicate entries can ruin the split between train, validation, and test sets where identical entries are not all in the same set. This can lead to biased performance estimates that result in disappointing the model in production.

4.Cross Validation:

Cross-validation is a technique for evaluating ML models by training several ML models on subsets of the available input data and evaluating them on the complementary subset of the data. Use cross-validation to detect overfitting, i.e., failing to generalize a pattern.

Methods used for Cross-Validation:

- Validation Set Approach
- Leave-P-out cross-validation
- Leave one out cross-validation
- K-fold cross-validation
- Stratified k-fold cross-validation



Program:

```
%pip install mlxtend --upgrade
```

Output:

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: mlxtend in /usr/local/lib/python3.9/dist-packages (0.14.0)
Collecting mlxtend
  Downloading mlxtend-0.22.0-py2.py3-none-any.whl (1.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 17.2 MB/s eta 0:00:00
Requirement already satisfied: matplotlib>=3.0.0 in /usr/local/lib/python3.9/dist-packages (from mlxtend) (3.7.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packages (from mlxtend) (67.7.2)
Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.9/dist-packages (from mlxtend) (1.2.0)
Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.9/dist-packages (from mlxtend) (1.5.3)
Requirement already satisfied: numpy>=1.16.2 in /usr/local/lib/python3.9/dist-packages (from mlxtend) (1.22.4)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.9/dist-packages (from mlxtend) (1.2.2)
Requirement already satisfied: scipy>=1.2.1 in /usr/local/lib/python3.9/dist-packages (from mlxtend) (1.10.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (4.39.3)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (8.4.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (0.11.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (1.0.7)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (3.0.9)
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (5.12.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (23.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib>=3.0.0->mlxtend) (1.4.4)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.24.2->mlxtend) (2022.7.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.9/dist-packages (from scikit-learn>=1.0.2->mlxtend) (3.1.0)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from importlib-resources>=3.2.0->matplotlib>=3.0.0->mlxtend) (3.15.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from python-dateutil>=2.7->matplotlib>=3.0.0->mlxtend) (1.16.0)
Installing collected packages: mlxtend
  Attempting uninstall: mlxtend
    Found existing installation: mlxtend 0.14.0
    Uninstalling mlxtend-0.14.0:
      Successfully uninstalled mlxtend-0.14.0
Successfully installed mlxtend-0.22.0
```

Program: (Bias, Variance)

```
[3] from pandas import read_csv
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression
    from mlxtend.evaluate import bias_variance_decomp
    # load dataset
    url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
    dataframe = read_csv(url, header=None)
    # separate into inputs and outputs
    data = dataframe.values
    X, y = data[:, :-1], data[:, -1]
    # split the data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
    # define the model
    model = LinearRegression()
    # estimate bias and variance
    mse, bias, var = bias_variance_decomp(model, X_train, y_train, X_test, y_test, loss='mse', num_rounds=200, random_seed=1)
    # summarize results
    print('MSE: %.3f' % mse)
    print('Bias: %.3f' % bias)
    print('Variance: %.3f' % var)
```

Output:

```
MSE: 22.418
Bias: 20.744
Variance: 1.674
```

Program:(Removing Duplicates)

```
[4]  import pandas as pd
     data = {
       "A": ["TeamA", "TeamB", "TeamB", "TeamC", "TeamA"],
       "B": [50, 40, 40, 30, 50],
       "C": [True, False, False, False, True]
     }
     df = pd.DataFrame(data)
     display(df.drop_duplicates())
```

Output:

	A	B	C
0	TeamA	50	True
1	TeamB	40	False
3	TeamC	30	False

Program:(Cross Validation)

```
[8]  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)

     clf = DecisionTreeClassifier(random_state=42)

     k_folds = KFold(n_splits = 5)

     scores = cross_val_score(clf, X, y, cv = k_folds)

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

Output:

```
Cross Validation Scores:  [1.         1.         0.83333333 0.93333333 0.8       ]
Average CV Score:  0.9133333333333333
Number of CV Scores used in Average:  5
```

Experiment – 8

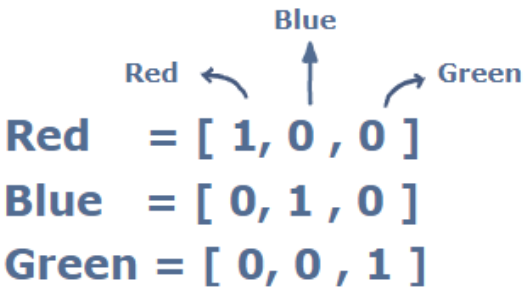
Write a program to implement One-hot Encoding

Aim: Write a program to implement One-hot Encoding

Description:

Most of the existing machine learning algorithms cannot be executed on categorical data. Instead, the categorical data needs to first be converted to numerical data. **One-hot encoding** is one of the techniques used to perform this conversion. This method is mostly used when deep learning techniques are to be applied to sequential classification problems.

One-hot encoding is essentially the representation of categorical variables as binary vectors. These categorical values are first mapped to integer values. Each integer value is then represented as a binary vector that is all 0s (*except* the index of the integer which is marked as 1).



Program:

```
import numpy as np
colors = ["red", "green", "yellow", "red", "blue"]
total_colors = ["red", "green", "blue", "black", "yellow"]
mapping = {}
for x in range(len(total_colors)):
  mapping[total_colors[x]] = x

one_hot_encode = []

for c in colors:
  arr = list(np.zeros(len(total_colors), dtype = int))
  arr[mapping[c]] = 1
  one_hot_encode.append(arr)

print(one_hot_encode)
```

Output:

```
[[1, 0, 0, 0, 0], [0, 1, 0, 0, 0], [0, 0, 0, 0, 1], [1, 0, 0, 0, 0], [0, 0, 1, 0, 0]]
```

Experiment – 9

Write a program to implement Categorical Encoding.

Aim: Write a program to implement Categorical Encoding using python.

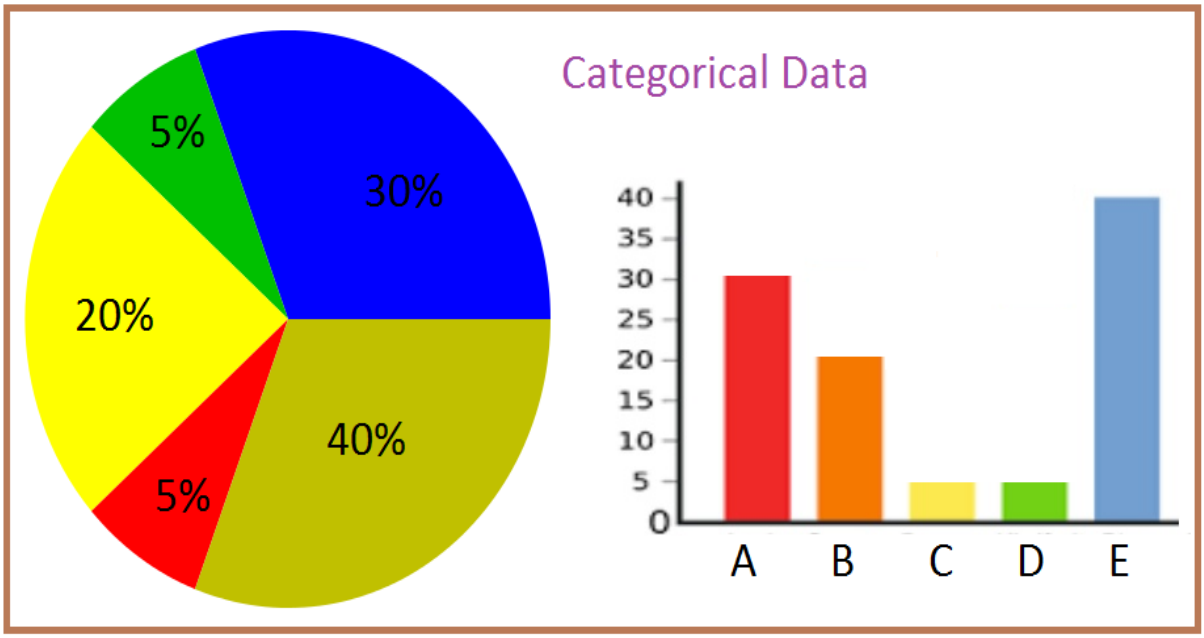
Description:

The **Sunbird** library is the best option for feature engineering purposes. In this library, you will get various techniques to handle missing values, outliers, categorical encoding, normalization and standardization, feature selection techniques, etc.

Categorical Encoding

Categorical data is a common type of non-numerical data that contains label values and not numbers. Some examples include:

Colors: White, Black, Green. Cities: Mumbai, Pune, Delhi. Gender: Male, Female.



Categorical data is a collection of information that is divided into groups. i.e., if an organization or agency is trying to get a biodata of its employees, the resulting data is referred to as categorical. This data is called categorical because it may be grouped according to the variables present in the biodata such as sex, state of residence, etc. Categorical data can take on numerical values (such as “1” indicating Yes and “2” indicating No), but those numbers don’t have mathematical meaning. One can neither add them together nor subtract them from each other.

Types of Categorical Data

- Nominal Data
- Ordinal Data

Program:

```
[3] pip install sunbird
```

Output:

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: sunbird in /usr/local/lib/python3.9/dist-packages (0.0.4)
Requirement already satisfied: sklearn in /usr/local/lib/python3.9/dist-packages (from sunbird) (0.0.post4)
Requirement already satisfied: seaborn in /usr/local/lib/python3.9/dist-packages (from sunbird) (0.12.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (from sunbird) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.9/dist-packages (from sunbird) (1.10.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.9/dist-packages (from sunbird) (1.5.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas->sunbird) (2022.7.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.9/dist-packages (from pandas->sunbird) (2.8.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in /usr/local/lib/python3.9/dist-packages (from seaborn->sunbird) (3.7.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn->sunbird) (8.4.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn->sunbird) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn->sunbird) (4.39.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn->sunbird) (1.4.4)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn->sunbird) (3.0.9)
```

Program:

```
[2] import pandas as pd
    data = {'Subject': ['s1', 's2', 's3', 's1', 's4',
                        's3', 's2', 's1', 's2', 's4', 's1'],
            'Target': [1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0]}
    df = pd.DataFrame(data)
    df
```

Output:

	Subject	Target
0	s1	1
1	s2	0
2	s3	1
3	s1	1
4	s4	1
5	s3	0
6	s2	0
7	s1	1
8	s2	1
9	s4	1
10	s1	0

Experiment – 10

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

Aim:

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

Description:

Backpropagation neural network is used to improve the accuracy of neural network and make them capable of self-learning. Backpropagation means "backward propagation of errors". Here error is spread into the reverse direction in order to achieve better performance. Backpropagation is an algorithm for supervised learning of artificial neural networks that uses the gradient descent method to minimize the cost function. It searches for optimal weights that optimize the mean-squared distance between the predicted and actual labels.

Artificial neural networks (ANNs) are powerful tools for machine learning with applications in manyareas including speech recognition, image classification, medical diagnosis, and spam filtering. It hasbeen shown that ANNs can approximate any function to any degree of accuracy given enough neuronsand training time. However, there is no guarantee on the number of neurons required or the time it will take to train them. These are the main disadvantages of using ANNs. Here we develop the Back Propagation algorithm which learns the weights for a multilayer network, given a network witha fixed set of units and interconnections. It employs gradient descent to attempt to minimize the squared error between the network output values and target values for these outputs.

BACK PROPAGATION ALGORITHM:

Multiple layer perceptron are effectively applied to handle tricky problems if trained with a vastly accepted algorithm identified as the back-propagation algorithm (error) in a supervised manner. It functions on learning law with error-correction. It is also a simplified version for the least mean square(LMS) filtering algorithm which is equally popular to error back-propagation algorithm.

Program:

```
[14] 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) :
       hinpl=np.dot (x, wh)
       hinp=hinpl + bh
       hlayer_act = sigmoid (hinp)
       outinpl=np.dot (hlayer_act,wout)
       outinp=outinpl+ 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:
 [[0.77276286]
 [0.76538704]
 [0.778016  ]]
```

Experiment – 11

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

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

Description:

K-Nearest Neighbor Algorithm

Training algorithm:

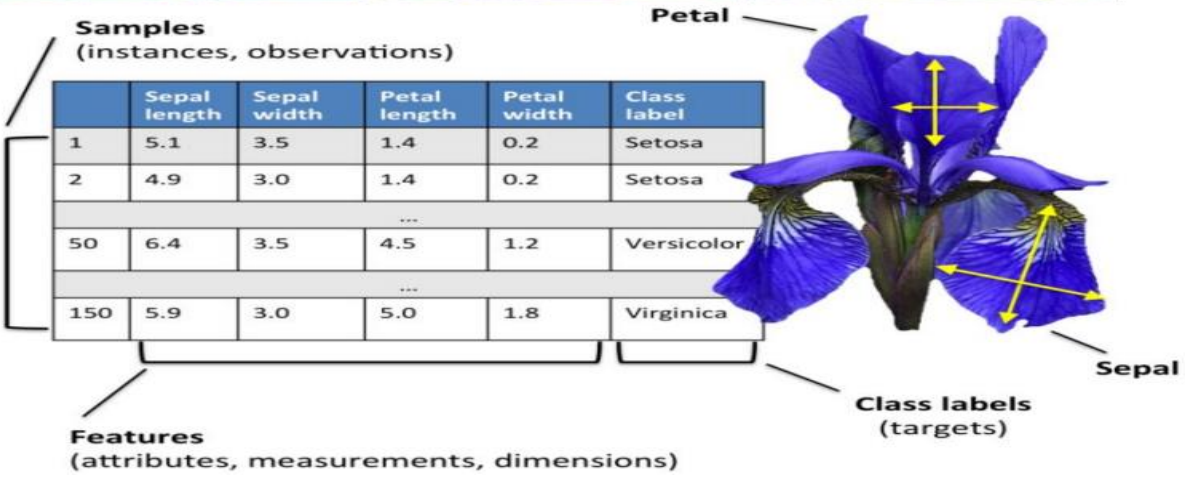
- For each training example (x, f (x)), add the example to the list training examples Classification algorithm:
    - Given a query instance x_q to be classified,
        - Let $x_1 \ldots x_k$ denote the k instances from training examples that are nearest to x_q
        - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

- Where, $f(x_i)$ function to calculate the mean value of the k nearest training examples.

Data Set:

Iris Plants Dataset: Dataset contains 150 instances (50 in each of three classes) Number of Attributes: 4 numeric, predictive attributes and the Class.



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
			...		
50	6.4	3.5	4.5	1.2	Versicolor
			...		
150	5.9	3.0	5.0	1.8	Virginica

Program:

```
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

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
dataset = pd.read_csv("/content/8-dataset.csv", names=names)
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
print(X.head())
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)

classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)

ypred = classifier.predict(Xtest)

i = 0
print ("\n-------------------------------------------------------------------------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))
print ("-------------------------------------------------------------------------")
for label in ytest:
    print ('%-25s %-25s' % (label, ypred[i]), end="")
    if (label == ypred[i]):
        print (' %-25s' % ('Correct'))
    else:
        print (' %-25s' % ('Wrong'))
    i = i + 1
print ("-------------------------------------------------------------------------")
print("\nConfusion Matrix:\n",metrics.confusion_matrix(ytest, ypred))
print ("-------------------------------------------------------------------------")
print("\nClassification Report:\n",metrics.classification_report(ytest, ypred))
print ("-------------------------------------------------------------------------")
print('Accuracy of the classifer is %0.2f' % metrics.accuracy_score(ytest,ypred))
print ("-------------------------------------------------------------------------")
```

Output:

```
   sepal-length  sepal-width  petal-length  petal-width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2
```

```
--------------------------------------------------------------------------------
Original Label            Predicted Label           Correct/Wrong
--------------------------------------------------------------------------------
Iris-virginica            Iris-virginica            Correct
Iris-virginica            Iris-virginica            Correct
Iris-setosa               Iris-setosa               Correct
Iris-virginica            Iris-virginica            Correct
Iris-setosa               Iris-setosa               Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-setosa               Iris-setosa               Correct
Iris-setosa               Iris-setosa               Correct
Iris-setosa               Iris-setosa               Correct
Iris-virginica            Iris-virginica            Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-setosa               Iris-setosa               Correct
Iris-versicolor           Iris-versicolor           Correct
Iris-setosa               Iris-setosa               Correct
Iris-versicolor           Iris-versicolor           Correct
--------------------------------------------------------------------------------

Confusion Matrix:
 [[7 0 0]
 [0 4 0]
 [0 0 4]]
--------------------------------------------------------------------------------


Classification Report:
                  precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       1.00      1.00      1.00         4
 Iris-virginica       1.00      1.00      1.00         4

       accuracy                           1.00        15
      macro avg       1.00      1.00      1.00        15
   weighted avg       1.00      1.00      1.00        15

--------------------------------------------------------------------------------
Accuracy of the classifer is 1.00
--------------------------------------------------------------------------------
```

Experiment – 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:

Implementation of the non-parametric Locally Weighted Regression algorithm in order to fit data points.

Description:

Locally Weighted Regression –

- 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).
- 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 –

- Locally weighted regression is called local because the function is approximated based aonly on data near the query point, weighted because the contribution of each training example is weighted by its distance from the query point.

- Query point is nothing but the point nearer to the target function , which will help in finding the actual position of the target function.

Let us consider the case of locally weighted regression in which the target function f is

1. Minimize the squared error over just the k nearest neighbors:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2$$

2. Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some decreasing function K of its distance from x_q:

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

3. Combine 1 and 2:

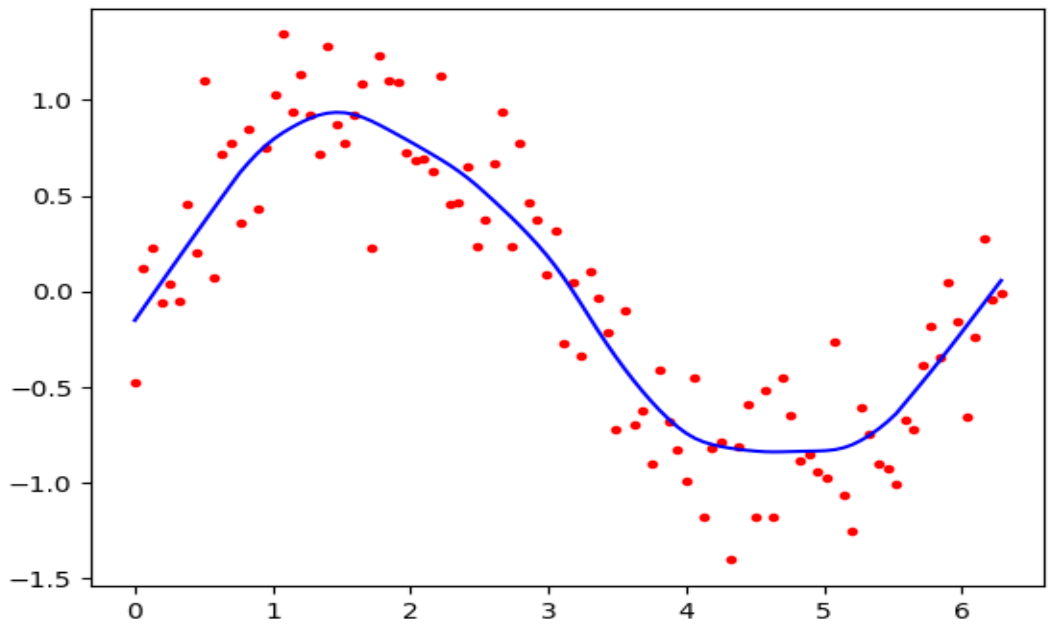
$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

Program:

```
[3]  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
     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)

     import matplotlib.pyplot as plt
     plt.plot(x,y,"r.")
     plt.plot(x,yest,"b-")
```

Output:

```
[<matplotlib.lines.Line2D at 0x7f7515eb2b50>]
```



Augmented Experiments

Experiment – 13

Assume 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. Calculating the accuracy, precision, and recall for the 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)
#print(len(twenty_train.data))
#print(len(twenty_test.data))
#print(twenty_train.target_names)
#print("\n".join(twenty_train.data[0].split("\n")))
#print(twenty_train.target[0])
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)
#print(X_train_tfidf)
X_train_tfidf.shape
#print(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

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

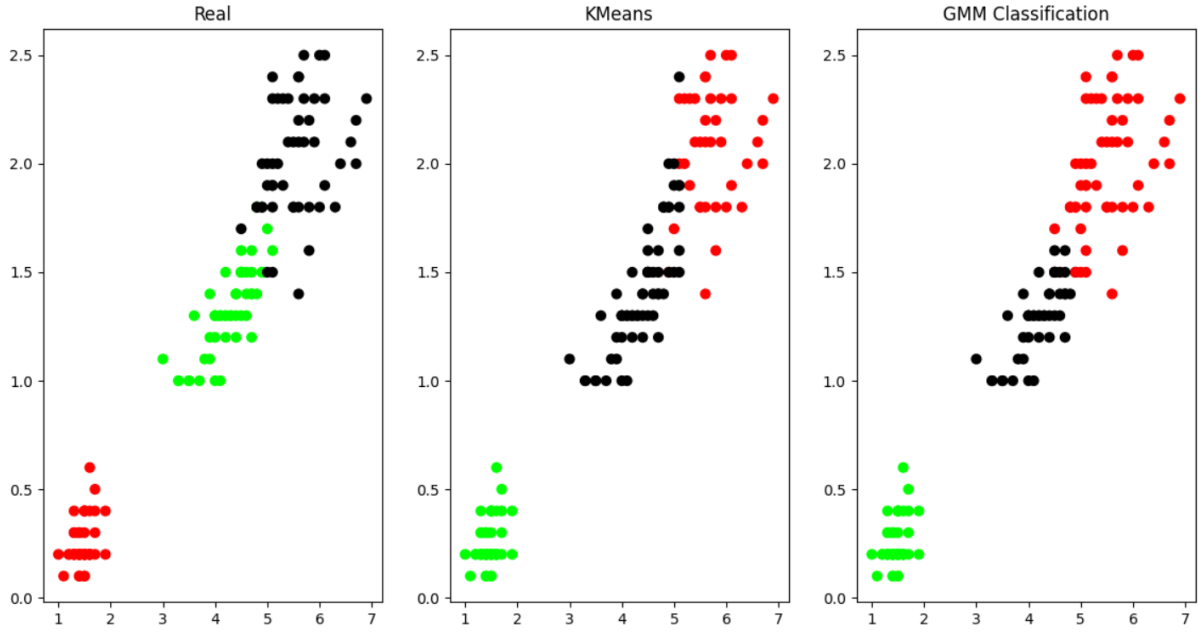
Aim: Implement the non-parametric Locally Weighted Regression algorithm to fit data points. Select appropriate data set for your experiment and draw graphs.

Program:

```
[1]  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()
     # print(dataset)
     X=pd.DataFrame(dataset.data)
     X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
     y=pd.DataFrame(dataset.target)
     y.columns=['Targets']
     # print(X)
     plt.figure(figsize=(14,7))
     colormap=np.array(['red','lime','black'])

     # REAL PLOT
     plt.subplot(1,3,1)
     plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y.Targets],s=40)
     plt.title('Real')

     # K-PLOT
     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')
  # GMM PLOT
  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')
```



Experiment – 15

Write a 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. You can use Java/Python ML library classes/API.

Aim: a 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. You can use Java/Python ML library classes/API.

Program:

```
import pandas as pd
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination

data = pd.read_csv("/content/Book heart.csv")
heart_disease = pd.DataFrame(data)
print(heart_disease)

model = BayesianModel([
    ('age', 'Lifestyle'),
    ('Gender', 'Lifestyle'),
    ('Family', 'heartdisease'),
    ('diet', 'cholestrol'),
    ('Lifestyle', 'diet'),
    ('cholestrol', 'heartdisease'),
    ('diet', 'cholestrol')
])

model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)
HeartDisease_infer = VariableElimination(model)
print('For Age enter SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4')
print('For Gender enter Male:0, Female:1')
print('For Family History enter Yes:1, No:0')
print('For Diet enter High:0, Medium:1')
print('for LifeStyle enter Athlete:0, Active:1, Moderate:2, Sedentary:3')
print('for Cholesterol enter High:0, BorderLine:1, Normal:2')

q = HeartDisease_infer.query(variables=['heartdisease'], evidence={
    'age': int(input('Enter Age: ')),
    'Gender': int(input('Enter Gender: ')),
    'Family': int(input('Enter Family History: ')),
    'diet': int(input('Enter Diet: ')),
    'Lifestyle': int(input('Enter Lifestyle: ')),
    'cholestrol': int(input('Enter Cholestrol: '))
})

print(q)
```

Output:

	age	Gender	Family	diet	Lifestyle	cholestrol	heartdisease
0	0	0	1	1	3	0	1
1	0	1	1	1	3	0	1
2	1	0	0	0	2	1	1
3	4	0	1	1	3	2	0
4	3	1	1	0	0	2	0
5	2	0	1	1	1	0	1
6	4	0	1	0	2	0	1
7	0	0	1	1	3	0	1
8	3	1	1	0	0	2	0
9	1	1	0	0	0	2	1
10	4	1	0	1	2	0	1
11	4	0	1	1	3	2	0
12	2	1	0	0	0	0	0
13	2	0	1	1	1	0	1
14	3	1	1	0	0	1	0
15	0	0	1	0	0	2	1
16	1	1	0	1	2	1	1
17	3	1	1	1	0	1	0
18	4	0	1	1	3	2	0

```
For Age enter SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4
For Gender enter Male:0, Female:1
For Family History enter Yes:1, No:0
For Diet enter High:0, Medium:1
for LifeStyle enter Athlete:0, Active:1, Moderate:2, Sedentary:3
for Cholesterol enter High:0, BorderLine:1, Normal:2

Enter Age: 0
Enter Gender: 0
Enter Family History: 0
Enter Diet: 0
Enter Lifestyle: 3
Enter Cholestrol: 0
```

heartdisease	phi(heartdisease)
heartdisease(0)	0.5000
heartdisease(1)	0.5000

```
Finding Elimination Order: : : 0it [00:00, ?it/s]
0it [00:00, ?it/s]
```

Experiment – 16

Consider Patient Dataset. Apply linear classification technique (SVM) to identify the rate of heart patients.

Aim: Considering Patient Dataset to apply linear classification technique (SVM) to identify the rate of heart patients in the hospital.

Program:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import hvplot.pandas
from scipy import stats

%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")
data = pd.read_csv("/content/Book1 heart.csv")
data.head()
```

Output:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Program:

```
data.shape
```

Output:

```
(5, 14)
```

Program:

```
pd.set_option("display.float", "{:.2f}".format)
data.describe()
```

Output:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00	5.00
mean	50.80	0.60	1.40	129.00	255.40	0.20	0.60	170.00	0.20	1.72	1.20	0.00	1.80	1.00
std	11.19	0.55	1.14	10.25	57.60	0.45	0.55	14.20	0.45	1.19	1.10	0.00	0.45	0.00
min	37.00	0.00	0.00	120.00	204.00	0.00	0.00	150.00	0.00	0.60	0.00	0.00	1.00	1.00
25%	41.00	0.00	1.00	120.00	233.00	0.00	0.00	163.00	0.00	0.80	0.00	0.00	2.00	1.00
50%	56.00	1.00	1.00	130.00	236.00	0.00	1.00	172.00	0.00	1.40	2.00	0.00	2.00	1.00
75%	57.00	1.00	2.00	130.00	250.00	0.00	1.00	178.00	0.00	2.30	2.00	0.00	2.00	1.00
max	63.00	1.00	3.00	145.00	354.00	1.00	1.00	187.00	1.00	3.50	2.00	0.00	2.00	1.00

Program:

```
data.target.value_counts()
```

Output:

```
1    5
Name: target, dtype: int64
```

Program:

```
data.isna().sum()
```

Output:

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Program:

```
categorical_val = []
continous_val = []
for column in data.columns:
    if len(data[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)
```

```
categorical_val
```

Output:

```
['age',
 'sex',
 'cp',
 'trestbps',
 'chol',
 'fbs',
 'restecg',
 'thalach',
 'exang',
 'oldpeak',
 'slope',
 'ca',
 'thal',
 'target']
```

Program:

```
for i, column in enumerate(categorical_val, 1):
  plt.subplot(3, 3, i)
  data[data["target"] == 0][column].hist(bins=35, color='blue', label='Have Heart Disease = NO', alpha=0.6)
  data[data["target"] == 1][column].hist(bins=35, color='red', label='Have Heart Disease = YES', alpha=0.6)
  plt.legend()
  plt.xlabel(column)
```

Output: