Expt	Aim / Title	CO
1	Implement Inference with Bayesian Network in Python.	CO 1
2	Build a Cognitive Healthcare application.	CO 2
3	Build Cognitive computing Insurance application.	CO 2
4	Implement Fuzzy Membership Functions.	CO 3
5	Implement Fuzzy set Properties.	CO 3
6	Design a Fuzzy control system using Fuzzy tool. (e.g. Washing Machine Controller/ Water Purifier Controller/ Domestic shower Controller etc.)	CO 3
7	Implement Image Classification System Application using Deep Learning.	CO 4
8	Implement Image Caption Generator Application using Deep Learning.	CO 4
9	Implement Ada-Boosting supervised learning algorithm.	CO 5
10	Implement Random forest supervised learning algorithm	CO 5
11	Mini Project on trends and applications in Data Science. e. g. Build text/ image/ video/ audio based DS Applications such as a. Chatbot b. Document Classification c. Sentiment Analysis d. Bounding Box Detection e. Music/Video Genre Classification	CO 6

Agnel Charities

Fr. C. Rodrigues Institute of Technology, Vashi

Department of Information Technology

LAB MANUAL

Class: VII IT

Course: ITL701 Data Science LAB

Course Coordinator: Dr. Vaishali V. Bodade

Experiment List

Experiment No 1

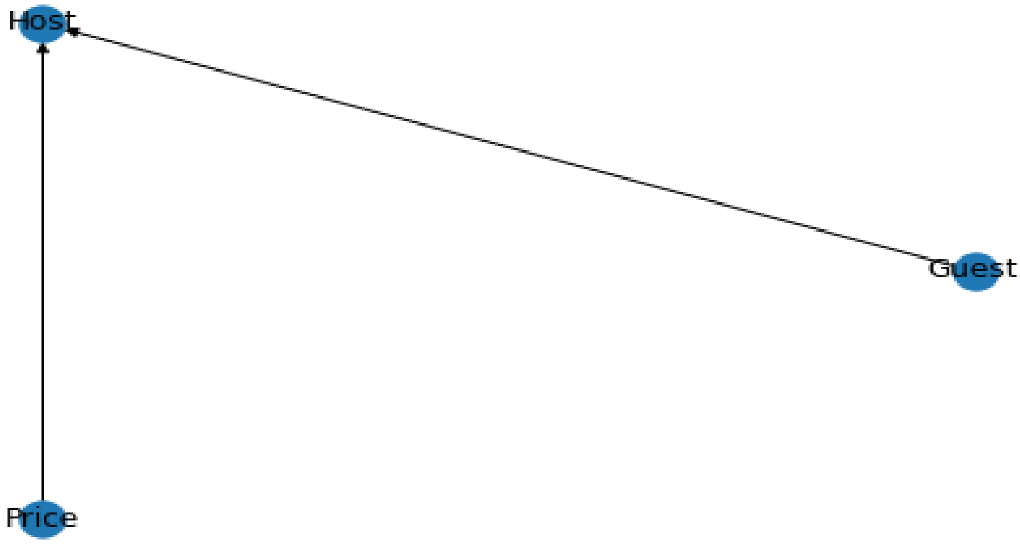
Aim: Implement Inferencing with Bayesian Network in Python.
CO: Implement reasoning with uncertainty.
Theory:
A Bayesian network is a probabilistic graphical model that represents a set of variables and their conditional dependencies via a directed acyclic graph (DAG). Bayesian networks are ideal for taking an event that occurred and predicting the likelihood that any one of several possible known causes was the contributing factor. For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

Bayesian Inferencing

Code :
```
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
import networkx as nx
import pylab as plt

# Defining Bayesian Structure
model = BayesianNetwork([('Guest', 'Host'), ('Price', 'Host')])

# Defining the CPDs:
cpd_guest = TabularCPD('Guest', 3, [[0.33], [0.33], [0.33]])
cpd_price = TabularCPD('Price', 3, [[0.33], [0.33], [0.33]])
cpd_host = TabularCPD('Host', 3, [[0, 0, 0, 0, 0.5, 1, 0, 1, 0.5],
                    [0.5, 0, 1, 0, 0, 0, 1, 0, 0.5],
                    [0.5, 1, 0, 1, 0.5, 0, 0, 0, 0]],
            evidence=['Guest', 'Price'], evidence_card=[3, 3])

# Associating the CPDs with the network structure.
model.add_cpds(cpd_guest, cpd_price, cpd_host)

model.check_model()
# Infering the posterior probability
from pgmpy.inference import VariableElimination

infer = VariableElimination(model)
posterior_p = infer.query(['Host'], evidence={'Guest': 2, 'Price': 2})
print(posterior_p)
pos = nx.circular_layout(model)
nx.draw(model, pos, with_labels=True)
plt.savefig('model.png')
plt.show()
```

```
plt.close()
```

Output:



Conclusion: Hence, we successfully implemented inferencing with the Bayesian networks in python.

Experiment No 2

Aim: Building a Cognitive Healthcare application.
CO: Explore use cases of Cognitive Computing
Theory:
Cognitive computing is the use of computerized models to simulate the human thought process. Computers are faster than humans at processing and calculating, but they have yet to master some tasks, such as understanding natural language and recognizing objects in an image. Cognitive computing is an attempt to have computers mimic the way a human brain works. It used techniques like self-learning algorithms, data analysis and pattern recognition to teach computing systems.

- Advantages-

1. Better data analysis.
2. Efficient processing.
3. Helps to include humans without completely automating a process.

- Disadvantages-

1. Security is a concern as cognitive computing handles sensitive data.
2. The cognitive computing systems require very lengthy development cycles.

- Cognitive computing in healthcare-

Heart Stroke Prediction
Code:

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import csv
from pandas import DataFrame
from datetime import datetime
from dateutil.relativedelta import relativedelta #used to represent intervals of timeframe
import time
import os
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
warnings.filterwarnings('ignore')
```

In [3]:
```
data = pd.read_csv('C:\Users\admin\Downloads\HeartStrokePrediction/train_data_stroke.csv',low_memory=False,skipinitialspace=True)
ata = pd.read_csv('C:\Users\admin\Downloads\HeartStrokePrediction/test_data.csv',low_memory=False,skipinitialspace=True)
```

```
[ ]: print(test_data.shape)
     print(train_data.shape)
     print(test_data.columns)
     print(train_data.columns)
```

```
(18601, 11)
(43400, 12)
Index(['id', 'gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
       'smoking_status'],
      dtype='object')
Index(['id', 'gender', 'age', 'hypertension', 'heart_disease',
       'Marital_Status', 'Work Profile', 'Residence_type', 'avg_glucose_level',
       'BMI', 'smoking_status', 'stroke'],
      dtype='object')
```

Since there is no stroke column available in test dataset , we will be considering only Train dataset for whole analysis

```
[ ]: data_stroke =train_data
```

```
[ ]: data_stroke.head()
```

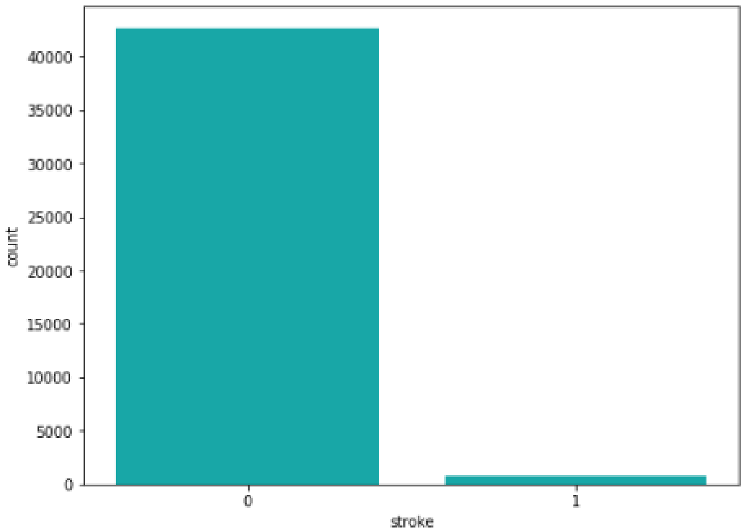
0	30669	Male	3.0	0	0	No	children	Rural	95.12	18.0	NaN	0
1	30468	Male	58.0	1	0	Yes	Private	Urban	87.96	39.2	never smoked	0
2	16523	Female	8.0	0	0	No	Private	Urban	110.89	17.6	NaN	0
3	56543	Female	70.0	0	0	Yes	Private	Rural	69.04	35.9	formerly smoked	0
4	46136	Male	14.0	0	0	No	Never_worked	Rural	161.28	19.1	NaN	0

```
In [ ]: data_stroke.isnull().sum() #null values for each of the feature variables
```

```
Out[8]: id                      0
        gender                  0
        age                     0
        hypertension            0
        heart_disease           0
        Marital_Status          0
        Work Profile            0
        Residence_type          0
        avg_glucose_level       0
        BMI                  1462
        smoking_status      13292
        stroke                  0
        dtype: int64
```

A countplot shows the counts of observations in each categorical bin using bars.

```
: f, ax = plt.subplots(figsize=(8, 6))
  sns.countplot(x="stroke", data=data_stroke, color="c")
  plt.show()
```



Handling Missing Data

```
data_stroke['BMI'].fillna(data_stroke['BMI'].mean(),inplace=True) #Filling the missing values of BMI with the mean values
```

Handling the Categorical columns

```
from sklearn.preprocessing import LabelEncoder
# instantiate labelencoder object
labelEncoder = LabelEncoder()
data_stroke['gender'] = labelEncoder.fit_transform(data_stroke['gender'])
data_stroke['Marital_Status'] = labelEncoder.fit_transform(data_stroke['Marital_Status'])
data_stroke['Work Profile'] = labelEncoder.fit_transform(data_stroke['Work Profile'])
data_stroke['Residence_type'] = labelEncoder.fit_transform(data_stroke['Residence_type'])
```

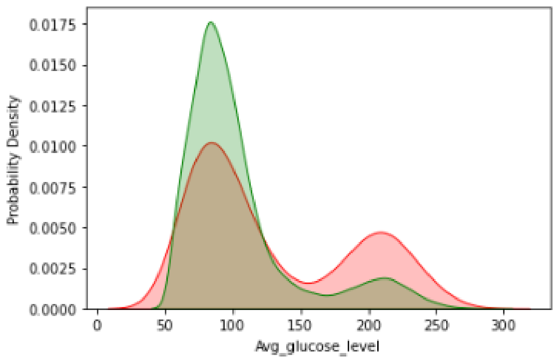
```
data_stroke.isnull().sum()
```

```
id                      0
gender                  0
age                     0
hypertension            0
heart_disease           0
Marital_Status          0
Work Profile            0
Residence_type          0
avg_glucose_level       0
BMI                     0
smoking_status      13292
stroke                  0
dtype: int64
```

This density estimate plot compares the probability density of patients with or without stroke on the range of ages

```
sns.kdeplot(data_stroke.loc[(data_stroke['stroke']==1),
            'avg_glucose_level'], color='r', shade=True, Label='Stroke')

sns.kdeplot(data_stroke.loc[(data_stroke['stroke']==0),
            'avg_glucose_level'], color='g', shade=True, Label='No Stroke')

plt.xlabel('Avg_glucose_level')
plt.ylabel('Probability Density')
```

```
Text(0, 0.5, 'Probability Density')
```



Handling Missing Data

```
data_stroke['BMI'].fillna(data_stroke['BMI'].mean(),inplace=True) #Filling the missing values of BMI with the mean values
```

Handling the Categorical columns

```
from sklearn.preprocessing import LabelEncoder
# instantiate labelencoder object
labelEncoder = LabelEncoder()
data_stroke['gender'] = labelEncoder.fit_transform(data_stroke['gender'])
data_stroke['Marital_Status'] = labelEncoder.fit_transform(data_stroke['Marital_Status'])
data_stroke['Work Profile'] = labelEncoder.fit_transform(data_stroke['Work Profile'])
data_stroke['Residence_type'] = labelEncoder.fit_transform(data_stroke['Residence_type'])
```

```
data_stroke.isnull().sum()
```

```
id                      0
gender                  0
age                     0
hypertension            0
heart_disease           0
Marital_Status          0
Work Profile            0
Residence_type          0
avg_glucose_level       0
BMI                     0
smoking_status      13292
stroke                  0
dtype: int64
```

Split the data as train and test

```
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
df_data_stroke = shuffle(df_data_stroke)
X = df_data_stroke.drop('stroke', axis = 1)
y = df_data_stroke['stroke']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=101)
```

```
print('X Train dataset shapes',X_train.shape)
print('Y Train dataset shapes',y_train.shape)
print('X Test dataset shapes',X_test.shape)
print('Y Test dataset shapes',y_test.shape)
```

Applying Logistic Regression Model

```
from sklearn.linear_model import LogisticRegression
```

```
logRe = LogisticRegression()
logRe.fit(X_train,y_train)
```

```
predictions = logRe.predict(X_test)
```

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

```
print(classification_report(y_test,predictions))
logRe.score(X_test, y_test)
```

In [68]:

```
prediction1 = logRe.predict([[0,45,0,1,1,2,0,90,27]])
print(prediction1)
```

```
[0]
```

Conclusion: Hence, we successfully built a Cognitive Healthcare application.

Experiment No 3

Aim: Cognitive computing in Insurance.
CO: Explore use cases of Cognitive Computing
Theory:
Cognitive computing is the use of computerized models to simulate the human thought process. Computers are faster than humans at processing and calculating, but they have yet to master some tasks, such as understanding natural language and recognizing objects in an image. Cognitive computing is an attempt to have computers mimic the way a human brain works. It used techniques like self-learning algorithms, data analysis and pattern recognition to teach computing systems.

HEALTH INSURANCE APPLICATION

Code:

```
In [1]: import pandas as pd
```

```
In [2]: data = pd.read_csv('insurance.csv')
```

1. Display Top 5 Rows of The Dataset

```
In [3]: data.head()
```

Out[3]:

	Age	Gender	BMI	Children	Existing illness	Surgical procedure	Alcohol consumption	Covid 19	Health Premium Cost
0	19	female	27.900	0	yes	no	yes	no	16884.92400
1	18	male	33.770	1	no	no	yes	no	1725.55230
2	28	male	33.000	3	no	no	yes	no	4449.46200
3	33	male	22.705	0	no	no	yes	no	21984.47061
4	32	male	28.880	0	no	no	yes	no	3866.85520

2. Check Last 5 Rows of The Dataset

```
In [4]: data.tail()
```

Out[4]:

	Age	Gender	BMI	Children	Existing illness	Surgical procedure	Alcohol consumption	Covid 19	Health Premium Cost
1333	50	male	30.97	3	no	no	no	yes	10600.5483
1334	18	female	31.92	0	no	no	yes	yes	2205.9808
1335	18	female	36.85	0	no	no	yes	yes	1629.8335
1336	21	female	25.80	0	no	no	yes	yes	2007.9450
1337	61	female	29.07	0	yes	no	yes	yes	29141.3603

3. Find Shape of Our Dataset (Number of Rows And Number of Columns)

```
In [5]: data.shape
```

```
Out[5]: (1338, 9)
```

```
In [6]: print("Number of Rows: ", data.shape[0])
        print("Number of Columns: ", data.shape[1])

Number of Rows:  1338
Number of Columns:  9
```

4. Get Information About Our Dataset (Total Number Rows, Total Number of Columns, Datatypes of Each Column And Memory Requirement)

```
In [7]: data.info
```

Out[7]:
```
<bound method DataFrame.info of       Age  Gender     BMI  Children Existing illness Surgical procedure  \
0      19  female  27.900         0              yes                 no
1      18    male  33.770         1               no                 no
2      28    male  33.000         3               no                 no
3      33    male  22.705         0               no                 no
4      32    male  28.880         0               no                 no
...   ...     ...     ...       ...              ...                ...
1333   50    male  30.970         3               no                 no
1334   18  female  31.920         0               no                 no
1335   18  female  36.850         0               no                 no
1336   21  female  25.800         0               no                 no
1337   61  female  29.070         0              yes                 no

     Alcohol consumption Covid 19  Health Premium Cost
0                    yes       no          16884.92400
1                    yes       no           1725.55230
2                    yes       no           4449.46200
3                    yes       no          21984.47061
4                    yes       no           3866.85520
...                  ...      ...                  ...
1333                  no      yes          10600.54830
1334                 yes      yes           2205.98080
1335                 yes      yes           1629.83350
1336                 yes      yes           2007.94500
1337                 yes      yes          29141.36030

[1338 rows x 9 columns]>
```

5.Check Null Values In The Dataset

In [8]:
```
data.isnull()
```

Out[8]:

	Age	Gender	BMI	Children	Existing illness	Surgical procedure	Alcohol consumption	Covid 19	Health Premium Cost
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
1333	False	False	False	False	False	False	False	False	False
1334	False	False	False	False	False	False	False	False	False
1335	False	False	False	False	False	False	False	False	False
1336	False	False	False	False	False	False	False	False	False
1337	False	False	False	False	False	False	False	False	False

1338 rows × 9 columns

In [9]:
```
data.isnull().sum()
```

Out[9]:
```
Age                   0
Gender                0
BMI                   0
Children              0
Existing illness      0
Surgical procedure    0
```

```
Surgical procedure     0
Alcohol consumption    0
Covid 19               0
Health Premium Cost    0
dtype: int64
```

6. Get Overall Statistics About The Dataset

In [10]:
```
data.describe(include="all")
```

Out[10]:

	Age	Gender	BMI	Children	Existing illness	Surgical procedure	Alcohol consumption	Covid 19	Health Premium Cost
count	1338.000000	1338	1338.000000	1338.000000	1338	1338	1338	1338	1338.000000
unique	NaN	2	NaN	NaN	2	2	2	2	NaN
top	NaN	male	NaN	NaN	no	no	yes	no	NaN
freq	NaN	676	NaN	NaN	1064	680	921	1220	NaN
mean	39.207025	NaN	30.663397	1.094918	NaN	NaN	NaN	NaN	13270.422265
std	14.049960	NaN	6.098187	1.205493	NaN	NaN	NaN	NaN	12110.011237
min	18.000000	NaN	15.960000	0.000000	NaN	NaN	NaN	NaN	1121.873900
25%	27.000000	NaN	26.296250	0.000000	NaN	NaN	NaN	NaN	4740.287150
50%	39.000000	NaN	30.400000	1.000000	NaN	NaN	NaN	NaN	9382.033000
75%	51.000000	NaN	34.693750	2.000000	NaN	NaN	NaN	NaN	16639.912515
max	64.000000	NaN	53.130000	5.000000	NaN	NaN	NaN	NaN	63770.428010

In [11]:
```
data.head()
```

Out[11]:

	Age	Gender	BMI	Children	Existing illness	Surgical procedure	Alcohol consumption	Covid 19	Health Premium Cost
0	19	female	27.900	0	yes	no	yes	no	16884.92400
1	18	male	33.770	1	no	no	yes	no	1725.55230
2	28	male	33.000	3	no	no	yes	no	4449.46200
3	33	male	22.705	0	no	no	yes	no	21984.47061
4	32	male	28.880	0	no	no	yes	no	3866.85520

7. Covert Columns From String ['sex','illness', 'region'] To Numerical Values

In [12]:
```
data['Gender'].unique()
```

Out[12]:
```
array(['female', 'male'], dtype=object)
```

In [13]:
```
data['Gender']=data['Gender'].map({'female':0, 'male':1})
```

In [14]:
```
data.head()
```

Out[14]:

	Age	Gender	BMI	Children	Existing illness	Surgical procedure	Alcohol consumption	Covid 19	Health Premium Cost
0	19	0	27.900	0	yes	no	yes	no	16884.92400
1	18	1	33.770	1	no	no	yes	no	1725.55230
2	28	1	33.000	3	no	no	yes	no	4449.46200
3	33	1	22.705	0	no	no	yes	no	21984.47061
4	32	1	28.880	0	no	no	yes	no	3866.85520

In [15]:
```
data['Existing illness']=data['Existing illness'].map({'yes':1, 'no':0})
```

In [16]:
```
data.head()
```

Out[16]:

	Age	Gender	BMI	Children	Existing illness	Surgical procedure	Alcohol consumption	Covid 19	Health Premium Cost
0	19	0	27.900	0	1	no	yes	no	16884.92400
1	18	1	33.770	1	0	no	yes	no	1725.55230
2	28	1	33.000	3	0	no	yes	no	4449.46200
3	33	1	22.705	0	0	no	yes	no	21984.47061
4	32	1	28.880	0	0	no	yes	no	3866.85520

In [17]:
```
data['Surgical procedure']=data['Surgical procedure'].map({'yes':1, 'no':0})
```

In [18]:
```
data.head()
```

Out[18]:

	Age	Gender	BMI	Children	Existing illness	Surgical procedure	Alcohol consumption	Covid 19	Health Premium Cost
0	19	0	27.900	0	1	0	yes	no	16884.92400
1	18	1	33.770	1	0	0	yes	no	1725.55230
2	28	1	33.000	3	0	0	yes	no	4449.46200
3	33	1	22.705	0	0	0	yes	no	21984.47061
4	32	1	28.880	0	0	0	yes	no	3866.85520

In [19]:
```
data['Alcohol consumption']=data['Alcohol consumption'].map({'yes':1, 'no':0})
```

In [20]:
```
data.head()
```

Out[20]:

	Age	Gender	BMI	Children	Existing illness	Surgical procedure	Alcohol consumption	Covid 19	Health Premium Cost
0	19	0	27.900	0	1	0	1	no	16884.92400
1	18	1	33.770	1	0	0	1	no	1725.55230
2	28	1	33.000	3	0	0	1	no	4449.46200
3	33	1	22.705	0	0	0	1	no	21984.47061
4	32	1	28.880	0	0	0	1	no	3866.85520

In [21]:
```
data['Covid 19']=data['Covid 19'].map({'yes':1, 'no':0})
```

In [22]:
```
data.head()
```

Out[14]:

	Age	Gender	BMI	Children	Existing illness	Surgical procedure	Alcohol consumption	Covid 19	Health Premium Cost
0	19	0	27.900	0	yes	no	yes	no	16884.92400
1	18	1	33.770	1	no	no	yes	no	1725.55230
2	28	1	33.000	3	no	no	yes	no	4449.46200
3	33	1	22.705	0	no	no	yes	no	21984.47061
4	32	1	28.880	0	no	no	yes	no	3866.85520

In [15]:
```
data['Existing illness']=data['Existing illness'].map({'yes':1, 'no':0})
```

In [16]:
```
data.head()
```

Out[16]:

	Age	Gender	BMI	Children	Existing illness	Surgical procedure	Alcohol consumption	Covid 19	Health Premium Cost
0	19	0	27.900	0	1	no	yes	no	16884.92400
1	18	1	33.770	1	0	no	yes	no	1725.55230
2	28	1	33.000	3	0	no	yes	no	4449.46200
3	33	1	22.705	0	0	no	yes	no	21984.47061
4	32	1	28.880	0	0	no	yes	no	3866.85520

In [17]:
```
data['Surgical procedure']=data['Surgical procedure'].map({'yes':1, 'no':0})
```

In [18]:
```
data.head()
```

Out[18]:

	Age	Gender	BMI	Children	Existing illness	Surgical procedure	Alcohol consumption	Covid 19	Health Premium Cost
0	19	0	27.900	0	1	0	yes	no	16884.92400
1	18	1	33.770	1	0	0	yes	no	1725.55230
2	28	1	33.000	3	0	0	yes	no	4449.46200
3	33	1	22.705	0	0	0	yes	no	21984.47061
4	32	1	28.880	0	0	0	yes	no	3866.85520

In [19]:
```
data['Alcohol consumption']=data['Alcohol consumption'].map({'yes':1, 'no':0})
```

In [20]:
```
data.head()
```

Out[20]:

	Age	Gender	BMI	Children	Existing illness	Surgical procedure	Alcohol consumption	Covid 19	Health Premium Cost
0	19	0	27.900	0	1	0	1	no	16884.92400
1	18	1	33.770	1	0	0	1	no	1725.55230
2	28	1	33.000	3	0	0	1	no	4449.46200
3	33	1	22.705	0	0	0	1	no	21984.47061
4	32	1	28.880	0	0	0	1	no	3866.85520

In [21]:
```
data['Covid 19']=data['Covid 19'].map({'yes':1, 'no':0})
```

In [22]:
```
data.head()
```

```
1336     2007.94500
1337    29141.36030
Name: Health Premium Cost, Length: 1338, dtype: float64
```

9. Train/Test split

```
1. Split data into two-part: a training set and a testing set
2. Train the model(s) on the training set
3. Test the Model(s) on the Testing set
```

In [28]:
```
from sklearn.model_selection import train_test_split
```

In [29]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [30]:
```
X_train
```

Out[30]:

	Age	Gender	BMI	Children	Existing illness	Surgical procedure	Alcohol consumption	Covid 19
560	46	0	19.950	2	0	1	1	0
1285	47	0	24.320	0	0	0	0	1
1142	52	0	24.860	0	0	1	1	0
969	39	0	34.320	5	0	0	1	0
486	54	0	21.470	3	0	1	1	0
...	...	...	...	...	...	...	...	...
1095	18	0	31.350	4	0	0	1	0
1130	39	0	23.870	5	0	1	1	0
1294	58	1	25.175	0	0	0	0	1
860	37	0	47.600	2	1	1	1	0
1126	55	1	29.900	0	0	1	1	0

1070 rows × 8 columns

In [31]:
```
y_train
```

Out[31]:
```
560      9193.83850
1285     8534.67180
1142    27117.99378
969      8596.82780
486     12475.35130
           ...
1095     4561.18850
1130     8582.30230
1294    11931.12525
860     46113.51100
1126    10214.63600
Name: Health Premium Cost, Length: 1070, dtype: float64
```

10. Import the models

In [32]:
```
from sklearn.linear_model import LinearRegression
from sklearn .svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

11. Model Training

In [33]:
```
lr = LinearRegression()
lr.fit(X_train, y_train)
svm = SVR()
svm.fit(X_train, y_train)
rf = RandomForestRegressor()
rf.fit(X_train, y_train)
gr = GradientBoostingRegressor()
gr.fit(X_train, y_train)
```

Out[33]:
```
GradientBoostingRegressor()
```

12. Prediction on Test Data

In [34]:
```
y_pred1 = lr.predict(X_test)
y_pred2 = svm.predict(X_test)
y_pred3 = rf.predict(X_test)
y_pred4 = gr.predict(X_test)

df1 = pd.DataFrame({'Actual':y_test, 'Lr':y_pred1, 'svm':y_pred2, 'rf':y_pred3, 'gr':y_pred4 })
```

In [35]:
```
df1
```

Out[35]:

	Actual	Lr	svm	rf	gr
764	9095.06825	8568.184849	9550.831680	10171.191204	10439.830888
887	5272.17580	6986.496486	9500.534344	5413.430396	6000.080317
890	29330.98315	36810.053376	9643.561212	28041.405384	28087.387080
1293	9301.89355	9738.990517	9556.979036	9541.952180	9804.243725
259	33750.29180	26809.036731	9432.205209	34508.495010	33914.083059
...	...	...	...	...	...
109	47055.53210	39183.183850	9643.535205	47119.145996	45728.193623
575	12222.89830	11712.841785	9621.198401	12744.005759	12974.656144
535	6067.12675	7268.798433	9511.013943	5918.815996	6433.325944
543	63770.42801	40967.939605	9603.303277	46753.764604	48114.743060
846	9872.70100	12589.536958	9589.616516	10219.634006	10666.795603

268 rows × 5 columns

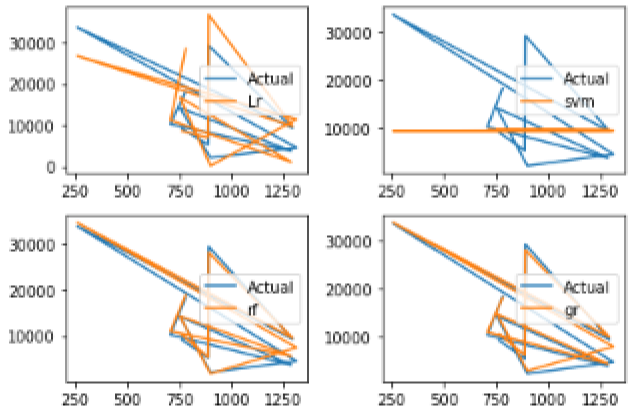
13. Compare Performance Visually

In [36]:
```
import matplotlib.pyplot as plt
```

In [36]:
```
import matplotlib.pyplot as plt
```

In [37]:
```
plt.subplot(221)
plt.plot(df1['Actual'].iloc[0:11],label='Actual')
plt.plot(df1['Lr'].iloc[0:11],label="Lr")
plt.legend()



plt.subplot(222)
plt.plot(df1['Actual'].iloc[0:11],label='Actual')
plt.plot(df1['svm'].iloc[0:11],label="svm")
plt.legend()



plt.subplot(223)
plt.plot(df1['Actual'].iloc[0:11],label='Actual')
plt.plot(df1['rf'].iloc[0:11],label="rf")
plt.legend()



plt.subplot(224)
plt.plot(df1['Actual'].iloc[0:11],label='Actual')
plt.plot(df1['gr'].iloc[0:11],label="gr")

plt.tight_layout()

plt.legend()
```

Out[37]: <matplotlib.legend.Legend at 0x1c324c710d0>



14. Evaluating the Algorithm

In [38]:
```
from sklearn import metrics
```

In [39]:
```
score1 = metrics.r2_score(y_test, y_pred1)
score2 = metrics.r2_score(y_test, y_pred2)
score3 = metrics.r2_score(y_test, y_pred3)
score4 = metrics.r2_score(y_test, y_pred4)
```

In [40]:
```
print(score1, score2, score3, score4)
```

```
0.7804872179173783 -0.07241871334749583 0.8577374153144108 0.8778850690766052
```

In [41]:
```
s1 = metrics.median_absolute_error(y_test, y_pred1)
s2 = metrics.median_absolute_error(y_test, y_pred2)
s3 = metrics.median_absolute_error(y_test, y_pred3)
s4 = metrics.median_absolute_error(y_test, y_pred4)
```

In [42]:
```
print(s1, s2, s3, s4)
```

```
2740.509599168465 5338.754597359515 1282.5546911500169 1478.0088964120223
```

In [43]:
```
s1 = metrics.mean_absolute_error(y_test, y_pred1)
s2 = metrics.mean_absolute_error(y_test, y_pred2)
s3 = metrics.mean_absolute_error(y_test, y_pred3)
s4 = metrics.mean_absolute_error(y_test, y_pred4)
```

In [44]:
```
print(s1, s2, s3, s4)
```

```
4222.113163128142 8596.512729434484 2654.9843461708956 2430.7104125443407
```

15. Predict Charges For New Customer

In [45]:
```
data = {'Age': 40,
        'Gender': 1,
        'BMI': 26.7,
        'Children': 3,
        'Existing illness': 1,
        'Surgical procedure': 0,
        'Alcohol consumption': 1,
        'Covid 19': 1}

df = pd.DataFrame(data, index=[0])
df
```

Out[45]:

	Age	Gender	BMI	Children	Existing illness	Surgical procedure	Alcohol consumption	Covid 19
0	40	1	26.7	3	1	0	1	1

In [46]:
```
new_pred = gr.predict(df)
print(new_pred)
```

```
[21787.82941338]
```

16. Save Model Usign Joblib

In [47]:
```
gr = GradientBoostingRegressor()
gr.fit(X,y)
```

Out[47]:
```
GradientBoostingRegressor()
```

In [48]:
```
import joblib
```

In [49]:
```
joblib.dump(gr, 'model_joblib_gr')
```

Out[49]:
```
['model_joblib_gr']
```

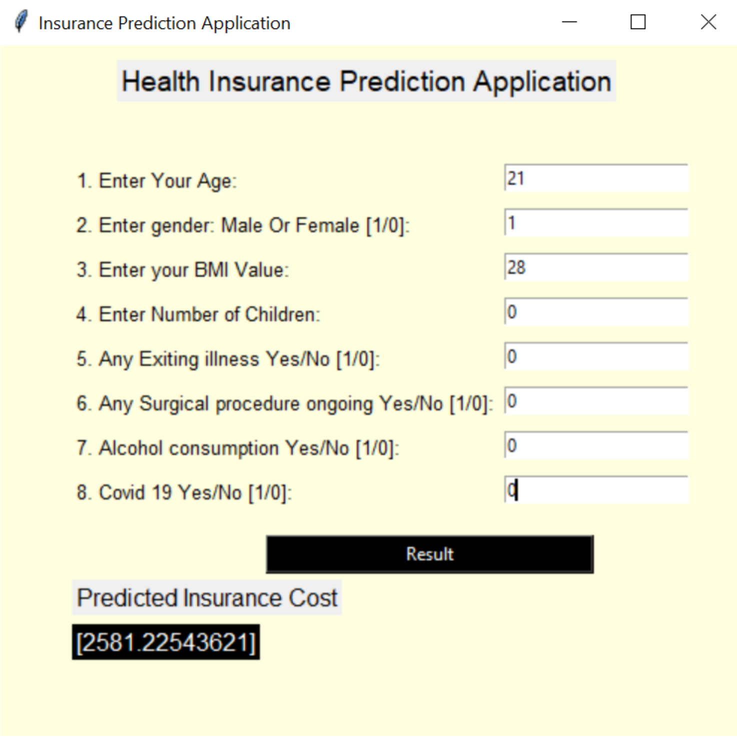
In [50]:
```
model = joblib.load('model_joblib_gr')
```

In [51]:
```
model.predict(df)
```

Out[51]:
```
array([21612.22992953])
```

GUI

In [ ]:
```
from tkinter import *
import joblib

def show_entry():

    p1 = float(e1.get())
    p2 = float(e2.get())
    p3 = float(e3.get())
    p4 = float(e4.get())
    p5 = float(e5.get())
    p6 = float(e6.get())
    p7 = float(e7.get())
    p8 = float(e8.get())

    model = joblib.load('model_joblib_gr')
    result = model.predict([[p1,p2,p3,p4,p5,p6,p7,p8]])

    label_9 = Label(master, text="Predicted Insurance Cost",font=("bold", 12)).place(x=50,y=360)

    label_10 = Label(master, text= result,font=("bold", 12),bg="black", fg="white").place(x=50,y=390)


master = Tk()
master.geometry('500x500')
master.configure(bg="light yellow")
master.title("Insurance Prediction Application")


label_0 = Label(master, text="Health Insurance Prediction Application",font=("bold", 14),justify = "center", fg = "black").place

label_1 = Label(master, text="1. Enter Your Age:",font=("bold", 10),bg="light yellow").place(x=50,y=80)
e1 = Entry(master)
e1.place(x=340,y=80)

label_2 = Label(master, text="2. Enter gender: Male Or Female [1/0]:",font=("bold", 10),bg="light yellow").place(x=50,y=110)
e2 = Entry(master)
e2.place(x=340,y=110)

label_3 = Label(master, text="3. Enter your BMI Value:",font=("bold", 10),bg="light yellow").place(x=50,y=140)
e3 = Entry(master)
e3.place(x=340,y=140)

label_4 = Label(master, text="4. Enter Number of Children:",font=("bold", 10),bg="light yellow").place(x=50,y=170)
e4 = Entry(master)
e4.place(x=340,y=170)

label_5 = Label(master, text="5. Any Exiting illness Yes/No [1/0]:",font=("bold", 10),bg="light yellow").place(x=50,y=200)
e5 = Entry(master)
e5.place(x=340,y=200)

label_6 = Label(master, text="6. Any Surgical procedure ongoing Yes/No [1/0]:",font=("bold", 10),bg="light yellow").place(x=50,y
e6 = Entry(master)
e6.place(x=340,y=230)

label_7 = Label(master, text="7. Alcohol consumption Yes/No [1/0]:",font=("bold", 10),bg="light yellow").place(x=50,y=260)
e7 = Entry(master)
e7.place(x=340,y=260)

label_8 = Label(master, text="8. Covid 19 Yes/No [1/0]:",font=("bold", 10),bg="light yellow").place(x=50,y=290)
e8 = Entry(master)
e8.place(x=340,y=290)
```



Conclusion: - Hence we successfully implemented cognitive computing Insurance application.

Experiment No 4

Aim: Implementation of Fuzzy Membership Functions.
CO: Implement a fuzzy controller system.
Theory:
Fuzzy membership function is used to convert the crisp input provided to the fuzzy inference system. a membership function for a fuzzy set A on the universe of discourse X is defined as $\mu A: X \rightarrow [0, 1]$, where each element of X is mapped to a value between 0 and 1. This value is called membership value or degree of membership. Fuzzy membership function is the graphical way of visualizing degree of membership of any value in given fuzzy set.

Code:

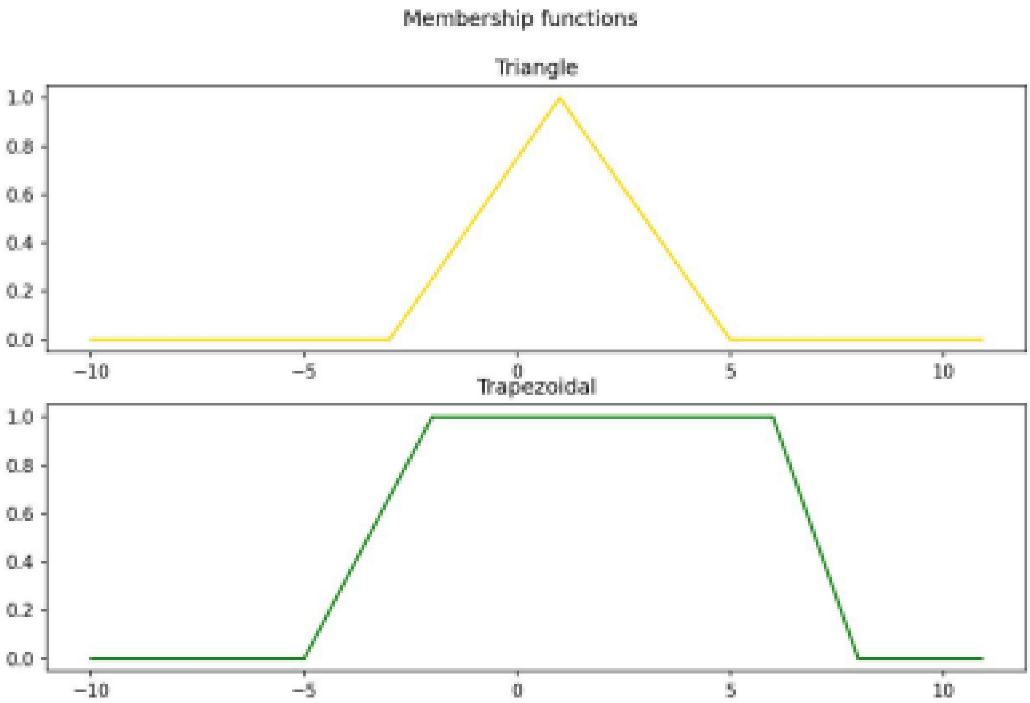
membershipFunction.py

```
import matplotlib.pyplot as plt

import membershipFunctions as membFuncts


# Setting a domain field.

xaxis = { 'xmin':-10, 'xmax':11 }


# Creating the values for the domain of all our membership functions.

xrange = membFuncts.getAxisValues(xaxis, 0.1)


trianglePoints = {'a':-3, 'b':1, 'c': 5}

trapezoidalPoints = {'a':-5, 'b': -2, 'c':6, 'd':8}


triangleFunction = membFuncts.triangleFunction(trianglePoints, xrange)

trapezoidalFunction = membFuncts.trapezoidalFunction(trapezoidalPoints, xrange)


# Setting the graphs grid size.

fig, axs = plt.subplots(2, 1, figsize=(10, 6), sharey=True)


axs[0].plot(xrange, triangleFunction, 'gold' )

axs[1].plot(xrange, trapezoidalFunction, 'green' )


axs[0].title.set_text('Triangle')

axs[1].title.set_text('Trapezoidal')
```

```
plt.suptitle('Membership functions')


plt.show()
```

graphics.py

```
import matplotlib.pyplot as plt

import membershipFunctions as membFuncts


# Setting a domain field.

xaxis = { 'xmin':-10, 'xmax':11 }


# Creating the values for the domain of all our membership functions.

xrange = membFuncts.getAxisValues(xaxis, 0.1)


# Functions information

trianglePoints = {'a':-3, 'b':1, 'c': 5}

trapezoidalPoints = {'a':-5, 'b': -2, 'c':6, 'd':8}


# Obtaining the graph values of each membership function.

triangleFunction = membFuncts.triangleFunction(trianglePoints, xrange)

trapezoidalFunction = membFuncts.trapezoidalFunction(trapezoidalPoints, xrange)


# Setting the graphs grid size.

fig, axs = plt.subplots(2, 1, figsize=(10, 6), sharey=True)


axs[0].plot(xrange, triangleFunction, 'gold' )

axs[1].plot(xrange, trapezoidalFunction, 'green' )


axs[0].title.set_text('Triangle')

axs[1].title.set_text('Trapezoidal')

plt.suptitle('Membership functions')

plt.show()
```



Conclusion: Successfully implemented fuzzy membership function.

Experiment No 5

Aim: Implementation of fuzzy set Properties.

CO: Implement a fuzzy controller system.

Theory:

Properties of fuzzy set helps us to simplify many mathematical fuzzy set operations. Sets are collection of unordered, district elements.

Code:

```
# Example to Demonstrate the

# Union of Two Fuzzy Sets

A = dict()

B = dict()

C = dict()

A = {"a": 0.2, "b": 0.3, "c": 0.6, "d": 0.6}

B = {"a": 0.9, "b": 0.9, "c": 0.4, "d": 0.5}

C = {"a": 0.7, "b": 0.5, "c": 0.2, "d": 0.1}

print('The First Fuzzy Set is :', A)

print('The Second Fuzzy Set is :', B)

print('The Third Fuzzy Set is :', C)

def union(A, B):

 if A == {}:

 return B

 elif B == {}:

 return A

 Res = dict()

 for A_key, B_key in zip(A, B):

 A_value = A[A_key]

 B_value = B[B_key]
 if A_value > B_value:  Res[A_key]

= A_value  else:

 Res[B_key] = B_value  return Res



def intersection(A, B):

 if A == {} or B =={}:

 return {}

 Res = dict()

 for A_key, B_key in zip(A, B):
```

```
A_value = A[A_key]  B_value =
B[B_key]


 if A_value < B_value:  Res[A_key]
= A_value  else:
 Res[B_key] = B_value  return Res


def complement(A):
 Res = dict()
 for key in A:
 Res[key] = 1 - A[key]
 return Res


def trans(A, B, C):
 for key in A:
 if key not in B:
 return False
 for key in B:
 if key not in C:
 return False
 return True

print("FUZZY SET PROPERTIES")
print("1. Involution")
invol = complement(complement(A))
print(invol)
print("")
print("2. Commutativity")
aub = union(A, B)
bua = union(B, A)
print("AUB : ", aub)
```

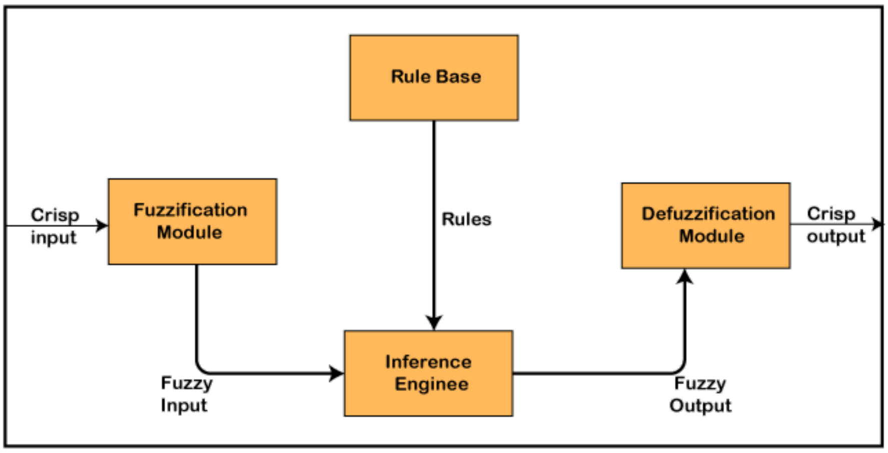
```
print("BUA : ", bua)

print("")

print("3. Associativity")

asc1 = union(A , union(B, C)) asc2

= union(union(A, B), C)

print("AU(BUC) : ", asc1)

print("(AUB)UC : ", asc2)
print("")

print("4. Distributivity")

lhs = union(A, intersection(B, C))

rhs = intersection(union(A, B), union(A, C))

print("AU(B∩C) : ", lhs)

print("( A ∪ B ) ∩ ( A ∪ C ) : ", rhs)

print("")

print("5. Absorption")

print(intersection(A, union(A, B)))

print("")

print("6. Idempotency")

print(union(A, A))

print("")

print("7. Identity")

print(union(A,{}))

print(intersection(A, {}))

print("")

print("8. Transitivity")

print(trans(A,B,C))

print("")

print("9. De Morgan's Law")

lhs = complement(union(A, B))

rhs = intersection(complement(A), complement(B))

print("(A U B)' : ", lhs)

print("A' ∩ B' : ", rhs)
```
Output:

```
The First Fuzzy Set is : {'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
The Second Fuzzy Set is : {'a': 0.9, 'b': 0.9, 'c': 0.4, 'd': 0.5}
The Third Fuzzy Set is : {'a': 0.7, 'b': 0.5, 'c': 0.2, 'd': 0.1}
FUZZY SET PROPERTIES
1. Involution
{'a': 0.19999999999999996, 'b': 0.30000000000000004, 'c': 0.6, 'd': 0.6}

2. Commutativity
AUB :  {'a': 0.9, 'b': 0.9, 'c': 0.6, 'd': 0.6}
BUA :  {'a': 0.9, 'b': 0.9, 'c': 0.6, 'd': 0.6}

3. Associativity
AU(BUC) :  {'a': 0.9, 'b': 0.9, 'c': 0.6, 'd': 0.6}
(AUB)UC :  {'a': 0.9, 'b': 0.9, 'c': 0.6, 'd': 0.6}

4. Distributivity
AU(BnC) :  {'a': 0.7, 'b': 0.5, 'c': 0.6, 'd': 0.6}
( A U B ) n ( A U C ) :  {'a': 0.7, 'b': 0.5, 'c': 0.6, 'd': 0.6}

5. Absorption
{'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}

6. Idempotency
{'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}

7. Identity
{'a': 0.2, 'b': 0.3, 'c': 0.6, 'd': 0.6}
{}

8. Transitivity
True

9. De Morgan's Law
(A U B)' :  {'a': 0.09999999999999998, 'b': 0.09999999999999998, 'c': 0.4, 'd': 0.4}
A' n B' :  {'a': 0.09999999999999998, 'b': 0.09999999999999998, 'c': 0.4, 'd': 0.4}
```

Conclusion: Successfully implemented fuzzy set properties.

Experiment No 6

Aim: Design of a Fuzzy control system using Fuzzy tool.
CO: Implement a fuzzy controller system.
Theory:
Fuzzy Control System:
A control system is an arrangement of physical components designed to alter another physical system so that this system exhibits certain desired characteristics
Architecture of fuzzy control system:



- Steps in designing FLC-

1. Identification of variables.
2. Fuzzy subset configuration.
3. Fuzzy rule base configuration.
4. Fuzzifcation.
5. Defuzzification.

Code:

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

###Declaring Constants

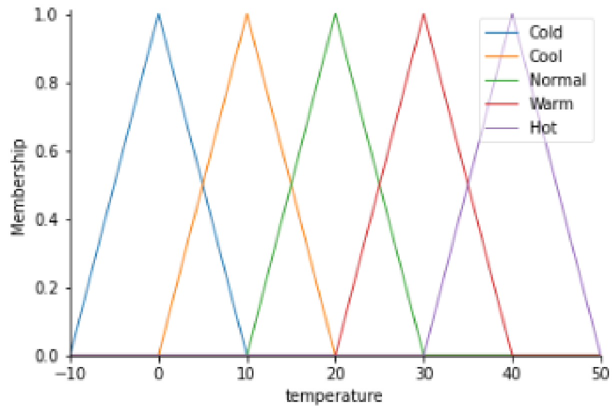
```
HUMIDITY = 'humidity'
SPRINKLER_DURATION = 'sprinkler_duration'
TEMPRATURE = 'temperature'

# Temperature's fuzzy linguistics
COLD = 'Cold'
COOL = 'Cool'
NORMAL = 'Normal'
WARM = 'Warm'
HOT = 'Hot'

# Humidity's fuzzy linguistics
DRY = 'Dry'
MOIST = 'Moist'
WET = 'Wet'

# Sprinkler duration's fuzzy linguistics
SHORT = 'Short'
MEDIUM = 'Medium'
LONG = 'Long'
```

###Setting The Antecedents & Consequent

```
temperature = ctrl.Antecedent(np.arange(-10,55,5), TEMPRATURE)
humidity = ctrl.Antecedent(np.arange(0,105,5), HUMIDITY)
sprinkler_duration = ctrl.Consequent(np.arange(0,105,5), SPRINKLER_DURATION)
```

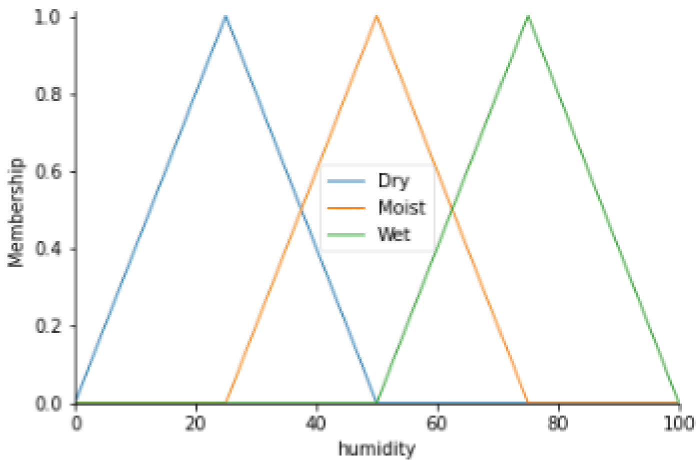
Assigning Membership Values To Temperature Using Triangular Membership Function

```
cold_parameter = [-10,0,10]
cool_parameter = [0,10,20]
normal_parameter = [10,20,30]
warm_parameter = [20,30,40]
hot_parameter = [30,40,50]

temperature[COLD] = fuzz.trimf(temperature.universe, cold_parameter)
temperature[COOL] = fuzz.trimf(temperature.universe, cool_parameter)
temperature[NORMAL] = fuzz.trimf(temperature.universe, normal_parameter)
temperature[WARM] = fuzz.trimf(temperature.universe, warm_parameter)
temperature[HOT] = fuzz.trimf(temperature.universe, hot_parameter)
temperature.view()
```



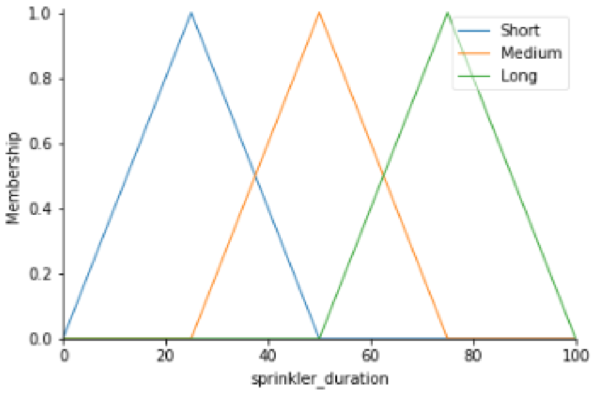
Assigning Membership Values To Humidity Using Triangular Membership Function

18]:
```
dry_parameter = [0,25,50]
moist_parameter = [25,50,75]
wet_parameter = [50,75,100]

humidity[DRY] = fuzz.trimf(humidity.universe, dry_parameter)
humidity[MOIST] = fuzz.trimf(humidity.universe, moist_parameter)
humidity[WET] = fuzz.trimf(humidity.universe, wet_parameter)
humidity.view()
```



Assigning Membership Values To Sprinkler Duration Using Triangular Membership Function

```python
short_parameter = [0,25,50]
medium_parameter = [25,50,75]
long_parameter = [50,75,100]

sprinkler_duration[SHORT] = fuzz.trimf(sprinkler_duration.universe, short_parameter)
sprinkler_duration[MEDIUM] = fuzz.trimf(sprinkler_duration.universe, medium_parameter)
sprinkler_duration[LONG] = fuzz.trimf(sprinkler_duration.universe, long_parameter)
sprinkler_duration.view()
```



Setting The Rules For Performance

H \ T	Cold	Cool	Normal	Warm	Hot
Dry	S	S	M	L	L
Moist	S	S	M	M	L
Wet	S	S	S	M	L

```python
rule1 = ctrl.Rule(humidity[DRY] & temperature[COLD], sprinkler_duration[SHORT])
rule2 = ctrl.Rule(humidity[DRY] & temperature[COOL], sprinkler_duration[SHORT])
rule3 = ctrl.Rule(humidity[DRY] & temperature[NORMAL], sprinkler_duration[MEDIUM])
rule4 = ctrl.Rule(humidity[DRY] & temperature[WARM], sprinkler_duration[LONG])
rule5 = ctrl.Rule(humidity[DRY] & temperature[HOT], sprinkler_duration[LONG])
rule6 = ctrl.Rule(humidity[MOIST] & temperature[COLD], sprinkler_duration[SHORT])
rule7 = ctrl.Rule(humidity[MOIST] & temperature[COOL], sprinkler_duration[SHORT])
rule8 = ctrl.Rule(humidity[MOIST] & temperature[NORMAL], sprinkler_duration[MEDIUM])
rule9 = ctrl.Rule(humidity[MOIST] & temperature[WARM], sprinkler_duration[MEDIUM])
rule10 = ctrl.Rule(humidity[MOIST] & temperature[HOT], sprinkler_duration[LONG])
rule11 = ctrl.Rule(humidity[WET] & temperature[COLD], sprinkler_duration[SHORT])
rule12 = ctrl.Rule(humidity[WET] & temperature[COOL], sprinkler_duration[SHORT])
rule13 = ctrl.Rule(humidity[WET] & temperature[NORMAL], sprinkler_duration[SHORT])
rule14 = ctrl.Rule(humidity[WET] & temperature[WARM], sprinkler_duration[MEDIUM])
rule15 = ctrl.Rule(humidity[WET] & temperature[HOT], sprinkler_duration[LONG])
```

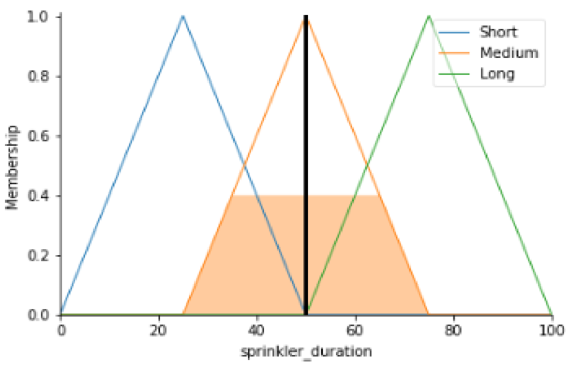
###Evaluating The Result Of A Student Using Fuzzy Logic

```python
sprinkler_analysis.input[TEMPRATURE] = 20
sprinkler_analysis.input[HUMIDITY] = 10

sprinkler_analysis.compute()
print(f'Evaluated Result: {str(round(sprinkler_analysis.output[SPRINKLER_DURATION], 2))} Min')
```

```
Evaluated Result: 50.0 Min
```

####Visualizating The Result

```python
sprinkler_duration.view(sim=sprinkler_analysis)
```



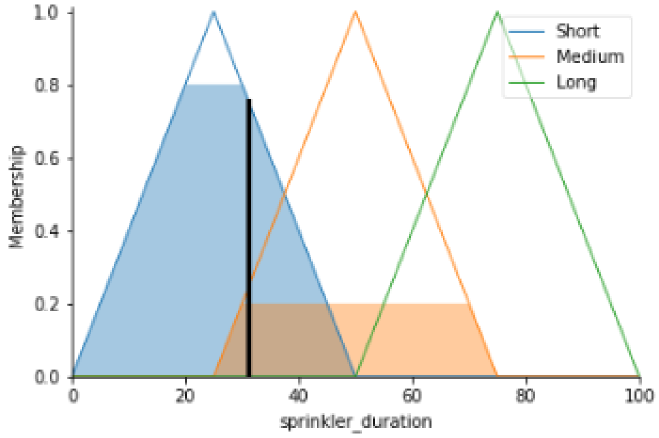
###Evaluating The Result Of A Student Using Fuzzy Logic

```
sprinkler_analysis.input[TEMPRATURE] = 12
sprinkler_analysis.input[HUMIDITY] = 50

sprinkler_analysis.compute()
print(f'Evaluated Result: {str(round(sprinkler_analysis.output[SPRINKLER_DURATION], 2))} Min')
```

```
Evaluated Result: 31.03 Min
```

####Visualizating The Result

```
sprinkler_duration.view(sim=sprinkler_analysis)
```



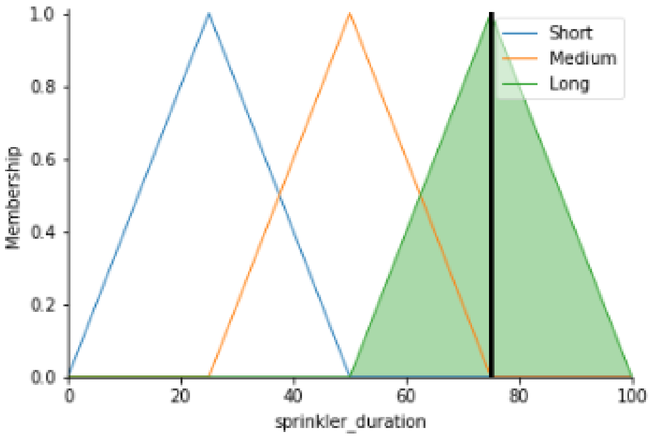
###Evaluating The Result Of A Student Using Fuzzy Logic

```
sprinkler_analysis.input[TEMPRATURE] = 40
sprinkler_analysis.input[HUMIDITY] = 50

sprinkler_analysis.compute()
print(f'Evaluated Result: {str(round(sprinkler_analysis.output[SPRINKLER_DURATION], 2))} Min')
```

```
Evaluated Result: 75.0 Min
```

####Visualizating The Result

```
sprinkler_duration.view(sim=sprinkler_analysis)
```



Conclusion: Successfully designed fuzzy logic controller using fuzzy tool.

Experiment No 7

Aim: Implementing Deep Learning Application Image Classification System.

CO: Develop real life applications using learning concepts.

Theory:

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers.

Code:

```
import gradio as gr
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import pathlib
dataset_url =
"https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url, untar=True) data_dir =
pathlib.Path(data_dir)
img_height,img_width=180,180
batch_size=32
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
data_dir,
validation_split=0.2,
subset="training",
seed=123,
image_size=(img_height, img_width),
batch_size=batch_size)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
data_dir,
validation_split=0.2,
subset="validation",
seed=123,
image_size=(img_height, img_width),
batch_size=batch_size)
class_names = train_ds.class_names
class_names[0] = "Daisy originated from India"
class_names[1] = "Dandelion originated from India"
class_names[2] = "Roses originated from India"
class_names[3] = "sunflower originated from India"
class_names[4] = "Tulips originated from India"
print(class_names)
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
for i in range(9):
ax = plt.subplot(3, 3, i + 1)
plt.imshow(images[i].numpy().astype("uint8"))
```

```
plt.title(class_names[labels[i]])
plt.axis("off")
num_classes = 5

model = Sequential([
layers.experimental.preprocessing.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
layers.Conv2D(16, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Conv2D(32, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Conv2D(64, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Flatten(),
layers.Dense(128, activation='relu'),
layers.Dense(num_classes,activation='softmax')
])
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
)
epochs=10
history = model.fit(
train_ds,
validation_data=val_ds,
epochs=epochs
)
def predict_image(img):
img_4d=img.reshape(-1,180,180,3)
prediction=model.predict(img_4d)[0]
return {class_names[i]: float(prediction[i]) for i in range(5)}
from typing import NamedTuple
image = gr.inputs.Image(shape=(180,180))

label = gr.outputs.Label(num_top_classes=1)

gr.Interface(fn=predict_image, inputs=image, outputs=[label], interpretation='default',
title='Image Classification').launch(debug='True')
```

Output :-

Conclusion: Successfully implemented image classification using deep learning.

Experiment No 8

Aim: Implementing Deep Learning Application Image Caption Generation.

CO: Develop real life applications using learning concepts.

Theory:
Image caption generator is a process of recognizing the context of an image and annotating it with relevant captions using deep learning, and computer vision. It includes the labeling of an image with English keywords with the help of datasets provided during model training. Imagenet dataset is used to train the CNN model called Xception.

Result already given in experiment 7.

Conclusion: Successfully implemented image caption generator using deep learning.

Experiment No 9

Aim: Implementation of supervised learning algorithm Ada-Boosting.
CO: Develop real life applications using learning concepts. Evaluate performance of applications.
Theory:
Algorithm:

1. Initially, Adaboost selects a training subset randomly.

2. It iteratively trains the AdaBoost machine learning model by selecting the training set based on the accurate prediction of the last training.
3. It assigns the higher weight to wrong classified observations so that in the next iteration these observations will get the high probability for classification.
4. Also, it assigns the weight to the trained classifier in each iteration according to the accuracy of the classifier. The more accurate classifier will get high weight.
5. This process iterates until the complete training data fits without any error or until reached to the specified maximum number of estimators.
6. To classify, perform a "vote" across all of the learning algorithms you built.

Code:

```
# Load libraries
from sklearn.ensemble import AdaBoostClassifier from sklearn import datasets
# Import train_test_split function
from sklearn.model_selection import train_test_split
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Load data
iris = datasets.load_iris()
X = iris.data y = iris.target
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and
30% test
# Create adaboost classifer object
abc = AdaBoostClassifier(n_estimators=50, learning_rate=1)
# Train Adaboost Classifer
model = abc.fit(X_train, y_train)
#Predict the response for test dataset
y_pred = model.predict(X_test)
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
# Load libraries
from sklearn.ensemble import AdaBoostClassifier
# Import Support Vector Classifier
from sklearn.svm import SVC
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
svc=SVC(probability=True, kernel='linear')
```

 # Create adaboost classifer object

abc =AdaBoostClassifier(n_estimators=50, base_estimator=svc,learning_rate=1)

 # Train Adaboost Classifer

model = abc.fit(X_train, y_train)

#Predict the response for test dataset

y_pred = model.predict(X_test)

 # Model Accuracy, how often is the classifier correct?

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

Conclusion: - Hence we successfully implemented Ada-boosting supervised learning algorithm.

Experiment No 10

Aim: Implementation of supervised learning algorithm Random forests.

CO: Develop real life applications using learning concepts. Evaluate performance of applications.

Theory:

Random forest is a type of supervised machine learning algorithm based on ensemble learning.

Algorithm:

1. Pick N random records from the dataset.
2. Build a decision tree based on these N records.
3. Choose the number of trees you want in your algorithm and repeat steps 1 and 2.
4. In case of a regression problem, for a new record, each tree in the forest predicts a value for Y (output). The final value can be calculated by taking the average of all the values predicted by all the trees in forest. Or, in case of a classification problem, each tree in the

forest predicts the category to which the new record belongs. Finally, the new record is assigned to the category that wins the majority vote.

Code:

```
import pandas as pd
import numpy as np
dataset = pd.read_csv('D:\Datasets\petrol_consumption.csv')
dataset.head()
```

	Petrol_tax	Average_income	Paved_Highways	Population_Driver_licence(%)	Petrol_Consumption
0	9.0	3571	1976	0.525	541
1	9.0	4092	1250	0.572	524
2	9.0	3865	1586	0.580	561
3	7.5	4870	2351	0.529	414
4	8.0	4399	431	0.544	410

```
X = dataset.iloc[:, 0:4].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=0.2,
random_state=0) # Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators=20, random_state=0)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
```

```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:',

np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```
The output will look something like this:

Mean Absolute Error: 51.765

Mean Squared Error: 4216.16675

Root Mean Squared Error: 64.932016371

Conclusion: Successfully implemented Random Forests algorithm from scratch.

Experiment 11

Mini Project on trends and applications in Data Science.
e.g.
Chatbot
Document Classification
Sentiment Analysis