Week-1

1.a. Python Program to find factorial of a Number

Source Code:

```
def factorial(n):
    if (n==1 or n==0):
        return 1
    else:
        return n * factorial(n - 1)
```

```
num = int(input('Enter the Number : '))
```

Enter the Number : 6

```
fact = factorial(num)
```

```
print("Factorial of {} is {}".format(num,fact))
```

Factorial of 6 is 720

1.b. Python Program to find Second Largest Element in an Array

Source Code:

```
def findseclargest(arr):
    largest = arr[0]
    sec_largest = arr[0]
    for i in range(len(arr)):
        if( arr[i] > largest ):
            largest = arr[i]
    for i in range(len(arr)):
        if(arr[i] > sec_largest and arr[i]!=largest):
            sec_largest = arr[i]
    return sec_largest
```

```
x = int(input("How many elements you want to enter into the list : "))
```

```
How many elements you want to enter into the list : 6
```

```
l=list()
for i in range(1,x+1):
    y = int(input("Enter the Element{} :".format(i)))
    l.append(y)
```

```
Enter the Element1 :8
Enter the Element2 :9
Enter the Element3 :7
Enter the Element4 :5
Enter the Element5 :8
Enter the Element6 :6
```

```
second_largest = findseclargest(l)
print("The Second Largest Element in Array {} is {}".format(l,second_largest))
```

```
The Second Largest Element in Array [8, 9, 7, 5, 8, 6] is 8
```

1.c. Python Program for Multiplication of Two Matrices

Source Code:

```
A=[ ]
B=[ ]
```

```
m = int(input("Enter the No of Rows of Matrix A : "))
n = int(input("Enter the No of Columns of Matrix A : "))
print("Enter the Matrix A Elements : \n")
for i in range(m):
    row = [ ]
    for j in range(n):
        row.append(int(input()))
    A.append(row)
```

```
Enter the No of Rows of Matrix A : 2
Enter the No of Columns of Matrix A : 2
Enter the Matrix A Elements :

4
5
6
8
```

```
p = int(input("Enter the No of Rows of Matrix B : "))
q = int(input("Enter the No of Columns of Matrix B : "))
print("Enter the Matrix A Elements : \n")
for i in range(p):
    row=[ ]
    for j in range(q):
        row.append(int(input()))
    B.append(row)
```

```
Enter the No of Rows of Matrix B : 2
Enter the No of Columns of Matrix B : 4
Enter the Matrix A Elements :

9
8
7
5
2
4
6
3
```

```
print("Matrix - A :")
for row in A:
   print(row)
print("\nMatrix - B :")
for row in B:
   print(row)
```

```
Matrix - A :
[4, 5]
[6, 8]

Matrix - B :
[9, 8, 7, 5]
[2, 4, 6, 3]
```

```
if(n==p):
   C=[]
   for i in range(m):
      row=[]
      for j in range(q):
         row.append(0)
      C.append(row)
   for i in range(m):
      for j in range(q):
         for k in range(p):
            C[i][j] += A[i][k] * B[k][j]

   print("The Matrix Muliplication of Two Matrices is \n")
   for row in C:
      print(row)
else:
   print("Matrix Multiplication Not Possible")
```

```
The Matrix Muliplication of Two Matrices is

[46, 52, 58, 35]
[70, 80, 90, 54]
```

1.d. Python Program to print reverse of a number

Source Code:

```
x = int(input("Enter a Number : "))
rev = 0
num=x
```

Enter a Number : 1245

```
while num != 0:
    rem = num % 10
    rev = rev * 10 + rem
    num //= 10
```

```
print("Reverse of Number {} is {} ".format(x,rev))
```

Reverse of Number 1245 is 5421

1.e.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.

Dataset:

A1 f_x Sky

	A	B	C	D	E	F	G	H	I
1	Sky	Temp	Humidity	Wind	Water	Forecast	Enjoy		
2	Sunny	Warm	Normal	Strong	Warm	Same	Yes		
3	Sunny	Warm	High	Strong	Warm	Same	Yes		
4	Rainy	Cold	High	Strong	Warm	Change	No		
5	Sunny	Warm	High	Strong	Cool	Same	Yes		
6									
7									
8									
9									
10									
11									

Source Code:

```
import pandas as pd
import numpy as np
```

```
data = pd.read_csv('dataset.csv')
print(data)
```

```
Sky   Temp  Humidity    Wind  Water  Forecast  Enjoy
0   Sunny  Warm    Normal  Strong  Warm      Same    Yes
1   Sunny  Warm      High  Strong  Warm      Same    Yes
2   Rainy  Cold      High  Strong  Warm    Change     No
3   Sunny  Warm      High  Strong  Cool      Same    Yes
```

```
concepts = np.array (data) [ : , : -1]
print(concepts)
```

```
[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Same']]
```

```
targets=np.array(data)[:,-1:]
print(targets)
```

```
[['Yes']
 ['Yes']
 ['No']
 ['Yes']]
```

```
def train(c,t):
    for i,val in enumerate(t):
        if(val=="Yes"):
            sp = c[i].copy()
            break
        else:
            return "There are No positives"
    for i,val in enumerate(c):
        if(t[i]=="Yes"):
            for x in range(len(sp)):
                if(val[x]!=sp[x]):
                        sp[x]="?"
    return sp;
```

```
print("The Most Specific Hypothesis : ",train(concepts,targets))
```

```
The Most Specific Hypothesis :  ['Sunny' 'Warm' '?' 'Strong' '?' 'Same']
```

Week-2

2a. Implement Simple linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

Dataset:

	A	B
1	YearsExperience	Salary
2	1.1	39343
3	1.3	46205
4	1.5	37731
5	2	43525
6	2.2	39891
7	2.9	56642
8	3	60150
9	3.2	54445
10	3.2	64445
11	3.7	57189
12	3.9	63218
13	4	55794
14	4	56957
15	4.1	57081
16	4.5	61111
17	4.9	67938
18	5.1	66029
19	5.3	83088
20	5.9	81363
21	6	93940
22	6.8	91738

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
dataset=pd.read_csv('Salary_Data.csv')
```

```
X=dataset.iloc[:,:-1].values
y=dataset.iloc[:, -1].values
print(X)
print(y)
```

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

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=1/3, random_state=0)
print(X_train)
print(X_test)
print(y_train)
print(y_test)
```

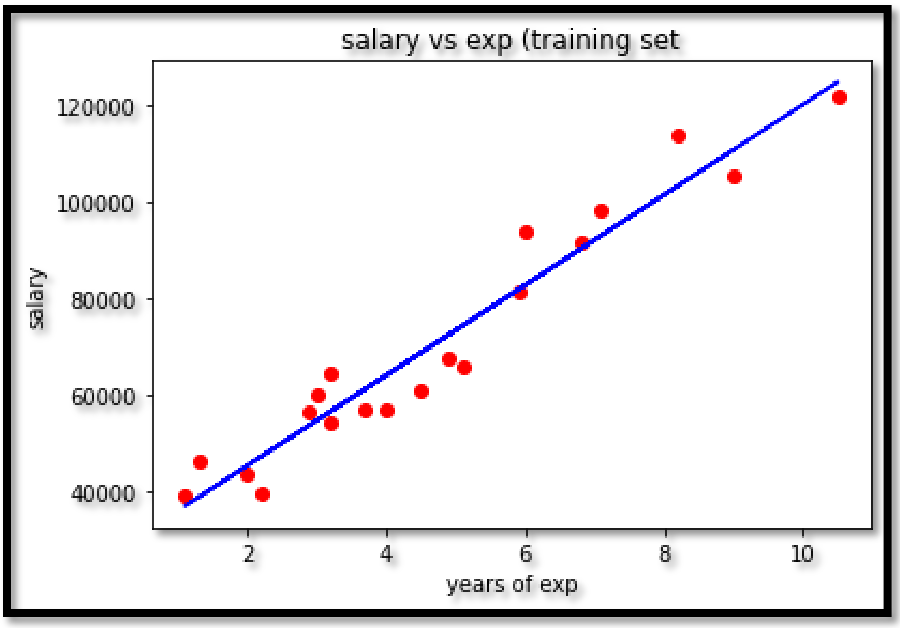
```
[[ 2.9]
 [ 5.1]
 [ 3.2]
 [ 4.5]
 [ 8.2]
 [ 6.8]
 [ 1.3]
 [10.5]
 [ 3. ]
 [ 2.2]
 [ 5.9]
 [ 6. ]
 [ 3.7]
 [ 3.2]
 [ 9. ]
 [ 2. ]
```

```
 [ 1.1]
 [ 7.1]
 [ 4.9]
 [ 4. ]]
[[ 1.5]
 [10.3]
 [ 4.1]
 [ 3.9]
 [ 9.5]
 [ 8.7]
 [ 9.6]
 [ 4. ]
 [ 5.3]
 [ 7.9]]
[ 56642.  66029.  64445.  61111. 113812.  91738.  46205. 121872.  60150.
  39891.  81363.  93940.  57189.  54445. 105582.  43525.  39343.  98273.
  67938.  56957.]
[ 37731. 122391.  57081.  63218. 116969. 109431. 112635.  55794.  83088.
 101302.]
```

```
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(X_train, y_train)
```

```
LinearRegression()
```

```
#LinearRegression(copy_X=(True), fit_intercept=(True), n_jobs=None, normalize=(False))
y_pred=regressor.predict(X_test)
plt.scatter(X_train,y_train,color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue')
plt.title('salary vs exp (training set')
plt.xlabel('years of exp')
plt.ylabel('salary')
plt.show()
```



2b. Implement Mulitple linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

Dataset:

	R&D Spend	Administration	Marketing Spend	State	Profit
2	165349.2	136897.8	471784.1	New York	192261.83
3	162597.7	151377.59	443898.53	California	191792.06
4	153441.51	101145.55	407934.54	Florida	191050.39
5	144372.41	118671.85	383199.62	New York	182901.99
6	142107.34	91391.77	366168.42	Florida	166187.94
7	131876.9	99814.71	362861.36	New York	156991.12
8	134615.46	147198.87	127716.82	California	156122.51
9	130298.13	145530.06	323876.68	Florida	155752.6
10	120542.52	148718.95	311613.29	New York	152211.77
11	123334.88	108679.17	304981.62	California	149759.96
12	101913.08	110594.11	229160.95	Florida	146121.95
13	100671.96	91790.61	249744.55	California	144259.4
14	93863.75	127320.38	249839.44	Florida	141585.52
15	91992.39	135495.07	252664.93	California	134307.35
16	119943.24	156547.42	256512.92	Florida	132602.65
17	114523.61	122616.84	261776.23	New York	129917.04
18	78013.11	121597.55	264346.06	California	126992.93
19	94657.16	145077.58	282574.31	New York	125370.37
20	91749.16	114175.79	294919.57	Florida	124266.9
21	86419.7	153514.11	0	New York	122776.86
22	76253.86	113867.3	298664.47	California	118474.03
23	78389.47	153773.43	299737.29	New York	111313.02
24	73994.56	122782.75	303319.26	Florida	110352.25

Source Code:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
# Importing the dataset
dataset = pd.read_csv('MLR_dataset.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
print(X)
```

```
[[165349.2 136897.8 471784.1 'New York']
 [162597.7 151377.59 443898.53 'California']
 [153441.51 101145.55 407934.54 'Florida']
 [144372.41 118671.85 383199.62 'New York']
 [142107.34 91391.77 366168.42 'Florida']
 [131876.9 99814.71 362861.36 'New York']
 [134615.46 147198.87 127716.82 'California']
 [130298.13 145530.06 323876.68 'Florida']
 [120542.52 148718.95 311613.29 'New York']
```

```
 [123334.88 108679.17 304981.62 'California']
 [101913.08 110594.11 229160.95 'Florida']
 [100671.96 91790.61 249744.55 'California']
 [93863.75 127320.38 249839.44 'Florida']
 [91992.39 135495.07 252664.93 'California']
 [119943.24 156547.42 256512.92 'Florida']
 [114523.61 122616.84 261776.23 'New York']
 [78013.11 121597.55 264346.06 'California']
 [94657.16 145077.58 282574.31 'New York']
 [91749.16 114175.79 294919.57 'Florida']
 [86419.7 153514.11 0.0 'New York']
 [76253.86 113867.3 298664.47 'California']
 [78389.47 153773.43 299737.29 'New York']
 [73994.56 122782.75 303319.26 'Florida']
 [67532.53 105751.03 304768.73 'Florida']
 [77044.01 99281.34 140574.81 'New York']
 [64664.71 139553.16 137962.62 'California']
 [75328.87 144135.98 134050.07 'Florida']
 [72107.6 127864.55 353183.81 'New York']
 [66051.52 182645.56 118148.2 'Florida']
 [65605.48 153032.06 107138.38 'New York']
 [61994.48 115641.28 91131.24 'Florida']
 [61136.38 152701.92 88218.23 'New York']
 [63408.86 129219.61 46085.25 'California']
 [55493.95 103057.49 214634.81 'Florida']
 [46426.07 157693.92 210797.67 'California']
 [46014.02 85047.44 205517.64 'New York']
 [28663.76 127056.21 201126.82 'Florida']
 [44069.95 51283.14 197029.42 'California']
 [20229.59 65947.93 185265.1 'New York']
 [38558.51 82982.09 174999.3 'California']
 [28754.33 118546.05 172795.67 'California']
 [27892.92 84710.77 164470.71 'Florida']
 [23640.93 96189.63 148001.11 'California']
 [15505.73 127382.3 35534.17 'New York']
 [22177.74 154806.14 28334.72 'California']
 [1000.23 124153.04 1903.93 'New York']
 [1315.46 115816.21 297114.46 'Florida']
 [0.0 135426.92 0.0 'California']
 [542.05 51743.15 0.0 'New York']
 [0.0 116983.8 45173.06 'California']]
```

```
# Encoding categorical data
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])],
remainder='passthrough')
X = np.array(ct.fit_transform(X))
print(X)
```

```
[[0.0 0.0 1.0 165349.2 136897.8 471784.1]
 [1.0 0.0 0.0 162597.7 151377.59 443898.53]
 [0.0 1.0 0.0 153441.51 101145.55 407934.54]
```

```
 [0.0 0.0 1.0 144372.41 118671.85 383199.62]
 [0.0 1.0 0.0 142107.34 91391.77 366168.42]
 [0.0 0.0 1.0 131876.9 99814.71 362861.36]
 [1.0 0.0 0.0 134615.46 147198.87 127716.82]
 [0.0 1.0 0.0 130298.13 145530.06 323876.68]
 [0.0 0.0 1.0 120542.52 148718.95 311613.29]
 [1.0 0.0 0.0 123334.88 108679.17 304981.62]
 [0.0 1.0 0.0 101913.08 110594.11 229160.95]
 [1.0 0.0 0.0 100671.96 91790.61 249744.55]
 [0.0 1.0 0.0 93863.75 127320.38 249839.44]
 [1.0 0.0 0.0 91992.39 135495.07 252664.93]
 [0.0 1.0 0.0 119943.24 156547.42 256512.92]
 [0.0 0.0 1.0 114523.61 122616.84 261776.23]
 [1.0 0.0 0.0 78013.11 121597.55 264346.06]
 [0.0 0.0 1.0 94657.16 145077.58 282574.31]
 [0.0 1.0 0.0 91749.16 114175.79 294919.57]
 [0.0 0.0 1.0 86419.7 153514.11 0.0]
 [1.0 0.0 0.0 76253.86 113867.3 298664.47]
 [0.0 0.0 1.0 78389.47 153773.43 299737.29]
 [0.0 1.0 0.0 73994.56 122782.75 303319.26]
 [0.0 1.0 0.0 67532.53 105751.03 304768.73]
 [0.0 0.0 1.0 77044.01 99281.34 140574.81]
 [1.0 0.0 0.0 64664.71 139553.16 137962.62]
 [0.0 1.0 0.0 75328.87 144135.98 134050.07]
 [0.0 0.0 1.0 72107.6 127864.55 353183.81]
 [0.0 1.0 0.0 66051.52 182645.56 118148.2]
 [0.0 0.0 1.0 65605.48 153032.06 107138.38]
 [0.0 1.0 0.0 61994.48 115641.28 91131.24]
 [0.0 0.0 1.0 61136.38 152701.92 88218.23]
 [1.0 0.0 0.0 63408.86 129219.61 46085.25]
 [0.0 1.0 0.0 55493.95 103057.49 214634.81]
 [1.0 0.0 0.0 46426.07 157693.92 210797.67]
 [0.0 0.0 1.0 46014.02 85047.44 205517.64]
 [0.0 1.0 0.0 28663.76 127056.21 201126.82]
 [1.0 0.0 0.0 44069.95 51283.14 197029.42]
 [0.0 0.0 1.0 20229.59 65947.93 185265.1]
 [1.0 0.0 0.0 38558.51 82982.09 174999.3]
 [1.0 0.0 0.0 28754.33 118546.05 172795.67]
 [0.0 1.0 0.0 27892.92 84710.77 164470.71]
 [1.0 0.0 0.0 23640.93 96189.63 148001.11]
 [0.0 0.0 1.0 15505.73 127382.3 35534.17]
 [1.0 0.0 0.0 22177.74 154806.14 28334.72]
 [0.0 0.0 1.0 1000.23 124153.04 1903.93]
 [0.0 1.0 0.0 1315.46 115816.21 297114.46]
 [1.0 0.0 0.0 0.0 135426.92 0.0]
 [0.0 0.0 1.0 542.05 51743.15 0.0]
 [1.0 0.0 0.0 0.0 116983.8 45173.06]]
```

```
# Splitting the dataset into the Training set and Test set
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)
```

```
# Training the Multiple Linear Regression model on the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

```
LinearRegression()
```

```
# Predicting the Test set results
y_pred = regressor.predict(X_test)
np.set_printoptions(precision=2)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[103015.2  103282.38]
 [132582.28 144259.4 ]
 [132447.74 146121.95]
 [ 71976.1   77798.83]
 [178537.48 191050.39]
 [116161.24 105008.31]
 [ 67851.69  81229.06]
 [ 98791.73  97483.56]
 [113969.44 110352.25]
 [167921.07 166187.94]]
```

Week-3

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 a nd apply this knowledge to classify a new sample

Source Code:

```
import numpy as np
import pandas as pd
```

```
eps = np.finfo(float).eps
```

```
outlook =
'overcast,overcast,overcast,overcast,rainy,rainy,rainy,rainy,rainy,sunny,sunny,sunny,sunny,sunny'
.split(',')


temp = 'hot,cool,mild,hot,mild,cool,cool,mild,mild,hot,hot,mild,cool,mild'.split(',')


humidity =
'high,normal,high,normal,high,normal,normal,normal,high,high,high,high,normal,normal'.split(',')


windy =
'FALSE,TRUE,TRUE,FALSE,FALSE,FALSE,TRUE,FALSE,TRUE,FALSE,TRUE,FALSE,FA
LSE,TRUE'.split(',')


play = 'yes,yes,yes,yes,yes,yes,no,yes,no,no,no,no,yes,yes'.split(',')
```

```
#dataframe
# dataset file is comma-separated
# first row are the predictors + target, columns of dataframe

# making a dataframe
# dataframe = pd.DataFrame(dataset,columns = list(dataset))
dataset ={'outlook':outlook,'temp':temp,'humidity':humidity,'windy':windy,'play':play}

dataframe = pd.DataFrame(dataset,columns=['outlook','temp','humidity','windy','play'])
dataframe
```

outlook	temp	humidity	windy	play	
0	overcast	hot	high	FALSE	yes
1	overcast	cool	normal	TRUE	yes
2	overcast	mild	high	TRUE	yes
3	overcast	hot	normal	FALSE	yes
4	rainy	mild	high	FALSE	yes
5	rainy	cool	normal	FALSE	yes
6	rainy	cool	normal	TRUE	no
7	rainy	mild	normal	FALSE	yes
8	rainy	mild	high	TRUE	no
9	sunny	hot	high	FALSE	no
10	sunny	hot	high	TRUE	no
11	sunny	mild	high	FALSE	no
12	sunny	cool	normal	FALSE	yes
13	sunny	mild	normal	TRUE	yes

```
# Entropy of the target attribute values
def find_entropy(df):
    target = df.keys()[-1]  # The last dataframe column is the target attribute (playGolf)
    entropy = 0
    values = df[target].unique()

    # for each value in the target playGolf attribute values
    for value in values:
        # ratio of values occurring and entropy
        fraction = df[target].value_counts()[value] / len(df[target])
        entropy += -fraction * np.log2(fraction)
    return entropy
```

```
# Entropy of attribute values
def find_entropy_attribute(df, attribute):
    target = df.keys()[-1]
    target_variables = df[target].unique()  # unique values in target playGolf attribute (Yes, No)
```

```
    variables = df[attribute].unique() # Identify Sunny, Overcast, Rainy

    # attribute entropy         # Variables=[sunny, sunny....5, overcast1.....overcast4, Rainy1...Ra5 ]
    entropy2 = 0
    # for each attribute value in attribute values
    for variable in variables:
        # value entropy
        entropy = 0
        # for each target value in target values (yes/no)
        for target_variable in target_variables:
            # frequency of attribute and target values (boolean indexing, pandas dataframe filtering)
            num = len(df[attribute][df[attribute] == variable][df[target] == target_variable])
            den = len(df[attribute][df[attribute] == variable])
            fraction = num / (den + eps)
            entropy += -fraction * np.log2(fraction + eps)
        fraction2 = den / len(df)
        entropy2 += -fraction2 * entropy
    return abs(entropy2)
```

```
def bestClassifier(df):
    # Entropy_att = []
    # information gain array for all attributes
    IG = []
    # for all attributes excluding target
    for key in df.keys()[:-1]:
        # Entropy_att.append(find_entropy_attribute(df,key))
        # calculate and record information gain value
        IG.append(find_entropy(df) - find_entropy_attribute(df, key)) #0.940 -0.693= 0.247
    return df.keys()[:-1][np.argmax(IG)]  # IG[0.247, 0.029, 0.152, 0.048 ]
```

```
def get_subtable(df, node, value):
    return df[df[node] == value].reset_index(drop=True)
```

```
def ID3split(df, tree=None):
    target = df.keys()[-1]
    # Here we build our decision tree
```

```python
    # Get attribute with maximum information gain
    node = bestClassifier(df) # 0.247
    # Get distinct value of that attribute e.g Salary is node and Low,Med and High are values
    attributeValues = np.unique(df[node])
    # Create an empty dictionary to create tree (recursive-friendly definition)
    if tree is None:            #  Outlook ->root node attribute
        tree = {}
        tree[node] = {}
    # following loop recursively calls ID3split to create and add to the tree
    # it runs till the tree is pure (leaf (result) node branches are added to the tree)
    for value in attributeValues:
        # get the subtable from current node based on the value
        subtable = get_subtable(df, node, value)
        # get the most common target value in the subtable
        targetValues, counts = np.unique(subtable[target], return_counts=True)
        # if the subtable is empty, assign the leaf node to the most common target value
        if len(counts) == 1:
            tree[node][value] = targetValues[0]
        else:
            # recursively call ID3 to create subtrees
            tree[node][value] = ID3split(subtable)  # Calling the function recursively
    return tree
```

```python
decisionTree = ID3split(dataframe)
print(decisionTree)
```

```
{'outlook': {'overcast': 'yes', 'rainy': {'windy': {'FALSE': 'yes
', 'TRUE': 'no'}}, 'sunny': {'humidity': {'high': 'no', 'normal':
'yes'}}}}
```