Week-1

Source code:

```
from collections import defaultdict
class Graph:
    def __init__(self):
        self.value = defaultdict(list)
    def drawGraph(self, parent, child):
        self.value[parent].append(child)
    def DFS(self, start):
        visited = []
        stack = [start]
        print("DFS Traversal:", end=" ")
        while stack:
            s = stack.pop()
            if s not in visited:
                print(s, end=" ")
                visited.append(s)
                for neighbor in reversed(self.value[s]):
                    if neighbor not in visited:
                        stack.append(neighbor)

    def BFS(self, start):
        visited = []
        queue = [start]

        print("\nBFS Traversal:", end=" ")

        while queue:
            x = queue.pop(0)
            if x not in visited:
```

```
            print(x, end=" ")
            visited.append(x)

            for neighbor in self.value[x]:
                if neighbor not in visited:
                    queue.append(neighbor)

# Create a graph and add edges.
g = Graph()
g.drawGraph(1, 4)
g.drawGraph(1, 2)
g.drawGraph(2, 3)
g.drawGraph(2, 6)
g.drawGraph(4, 5)
g.drawGraph(4, 7)
g.drawGraph(7, 96)

# Perform DFS and BFS traversals.
g.DFS(1)
g.BFS(1)
```

output:

```
DFS Traversal: 1 4 5 7 96 2 3 6
BFS Traversal: 1 4 2 5 7 3 6 96
```

Week-3a:

Source code:

```
from sys import maxsize
from itertools import permutations

V=4
```

```
def travellingSalespersonProblem(graph,s):
    vertex=[]
    for i in range(V):
        if i!=s:
            vertex.append(i)


    min_path = maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:
        print(i)
        current_pathweight=0
        k=s
        for j in i:
            current_pathweight += graph[k][j]
            k=j
        current_pathweight += graph[k][s]
        min_path = min(min_path, current_pathweight)
    return min_path
if __name__ == "__main__":
    graph = [[0,10,15,20],[10,0,35,25],[15,35,0,30],[20,25,30,0]]
    s=0
    print(travellingSalespersonProblem(graph,s))
```

output:

```
(1, 2, 3)
(1, 3, 2)
(2, 1, 3)
(2, 3, 1)
(3, 1, 2)
(3, 2, 1)
80
```

Week-3b:

Source code:

```
colors=['red','blue','green','yellow','black']
states=['andhra','karnataka','tamilnadu','kerela']
neighbors={}
neighbors['andhra']=['karnataka','tamilnadu']
neighbors['karnataka']=['andhra','tamilnadu','kerela']
neighbors['tamilnadu']=['andhra','kerela','karnataka']
neighbors['kerela']=['karnataka','tamilnadu']
colors_of_states={}
def promising(state,color):
    for neighbor in neighbors.get(state):
        colors_of_neighbor=colors_of_states.get(neighbor)
        if colors_of_neighbor==color:
            return False
    return True

def get_color_for_state(state):
    for color in colors:
        if promising(state,color):
            return color

def main():
    for state in states:
        colors_of_states[state]=get_color_for_state(state)
    print(colors_of_states)

main()
```

output:

```
{'andhra': 'red', 'karnataka': 'blue', 'tamilnadu': 'green', 'kerela': 'red'}
```

Week-4:

Source code:

```
from sympy import symbols, Or, Not , Implies, satisfiable

Rain = symbols('Rain')

Harry_Visited_Hagrid = symbols('Harry_Visited_Hagrid')

Harry_Visited_Dumbledore = symbols('Harry_Visited_Dumbledore')

sentence_1 = Implies(Not(Rain), Harry_Visited_Hagrid)

sentence_2 = Or (Harry_Visited_Hagrid,Harry_Visited_Dumbledore) &
Not(Harry_Visited_Hagrid & Harry_Visited_Dumbledore)

sentence_3 = Harry_Visited_Dumbledore

knowledge_base = sentence_1 & sentence_2 & sentence_3

solution = satisfiable(knowledge_base, all_models=True)

for model in solution:

  if model[Rain]:

     print("It rained today.")

  else:

       print("It did not rain today.")
```

output:

it rained today

week-5:

source code:

```
import numpy

from pomegranate import *

guest = DiscreteDistribution({'A':1./3,'B':1./3,'C':1./3})

prize = DiscreteDistribution({'A':1./3,'B':1./3,'C':1./3})

monty = ConditionalProbabilityTable(

   [['A','A','A',0.0],

    ['A','A','B',0.5],

    ['A','A','C',0.5],
```

```
    ['A','B','A',0.0],
    ['A','B','B',0.0],
    ['A','B','C',1.0],
    ['A','C','A',0.0],
    ['A','C','B',1.0],
    ['A','C','C',0.0],
    ['B','A','A',0.0],
    ['B','A','B',0.0],
    ['B','A','C',1.0],
    ['B','B','A',0.5],
    ['B','B','B',0.0],
   ['B','B','C',0.5],
    ['B','C','A',1.0],
    ['B','C','B',0.0],
    ['B','C','C',0.0],
  ['C','A','A',0.0],
    ['C','A','B',1.0],
   ['C','A','C',0.0],
    ['C','B','A',1.0],
  ['C','B','B',0.0],
    ['C','B','C',0.0],
  ['C','C','A',0.5],
   ['C','C','B',0.5],
   ['C','C','C',0.0]], [guest,prize])
s1 = State(guest, name="guest")
s2 = State(prize, name="prize")
s3 = State(monty, name="monty")
model = BayesianNetwork("monty Hall Problem")
model.add_states(s1,s2,s3)
model.add_edge(s1,s3)
```

```
model.add_edge(s2,s3)
model.bake()
print(model.probability([['A','B','C'],['A','A','C'],['A','C','C']]))
print(model.predict([['A',None,'C'],['A','A',None],[None,'B','A']]))
```

week-6:

source code:

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

#create state space and initial state probabilities
states=['sleeping','eating','pooping']
hidden_states=['healthy','sick']
pi=[0.5,0.5]

state_space=pd.Series(pi, index=hidden_states,name='states')
print(state_space)

a_df=pd.DataFrame(columns=hidden_states,index=hidden_states)
a_df.loc[hidden_states[0]]=[0.7,0.3]
a_df.loc[hidden_states[1]]=[0.4,0.6]
print(a_df)

observable_states=states
b_df=pd.DataFrame(columns=observable_states,index=hidden_states)
b_df.loc[hidden_states[0]]=[0.2,0.6,0.2]
b_df.loc[hidden_states[1]]=[0.4,0.1,0.5]
print(b_df)
```

```
def HMM(obsq,a_df,b_df,pi,states,hidden_states):
    hidst=list(itertools.product(hidden_states,repeat=len(obsq)))
    print(hidst)
    sum=0
    for k in hidst:
        prod=1
        for j in range(len(k)):
            c=0
            for i in obsq:
                if c==0:
                    prod*=a_df[i][k[j]]*pi[hidden_states.index(k[j])]
                    c=1
                else:
                    prod*=b_df[k[j]][k[j-1]]*a_df[i][k[j]]
        sum+=prod
        c=0
    return sum
def vertibi(obsq,a_df,b_df,pi,states,hidden_states):
    sum=0
    hidst=list(itertools.product(hidden_states,repeat=len(obsq)))
    for k in hidst:
        sum1=0
        prod=1
        for j in range(len(k)):
            c=0
            for i in obsq:
                if c==0:
                    prod*=a_df[i][k[j]]*pi[hidden_states.index(k[j])]
                    c=1
                else:
```

```
                prod*=b_df[k[j]][k[j-1]]*a_df[i][k[j]]
        c=0
        sum1+=prod
        if(sum1>sum):
            sum=sum1
            hs=k
    return sum,hs


obsq=['eating','sleeping','sleeping']
print(HMM(obsq,b_df,a_df,pi,states,hidden_states))
print(vertibi(obsq,b_df,a_df,pi,states,hidden_states))
```

output:

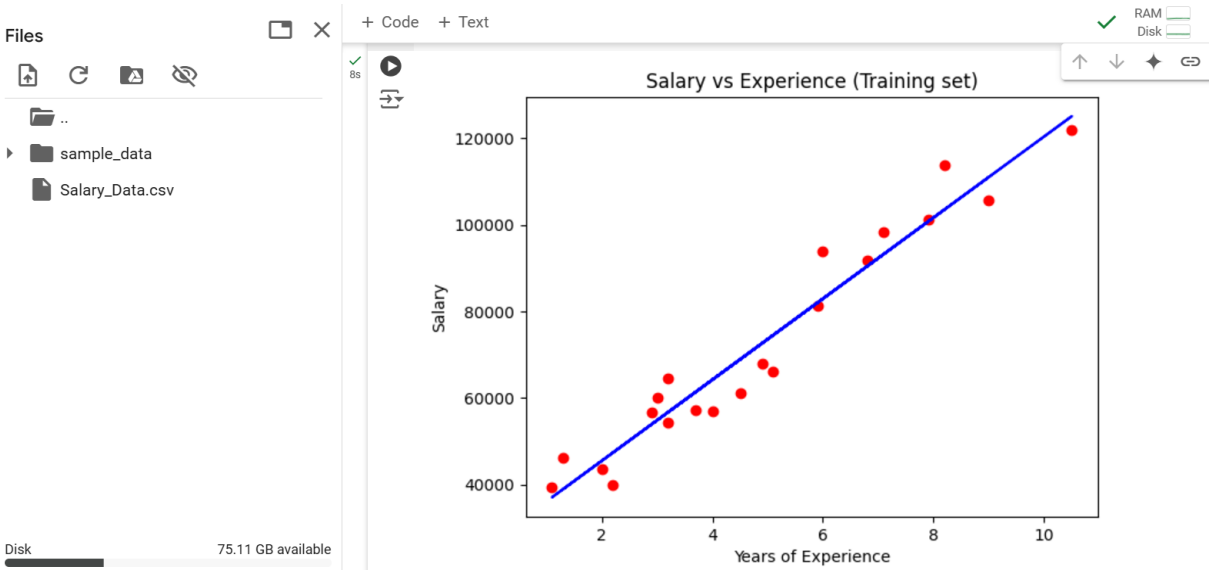
```
healthy    0.5
sick       0.5
Name: states, dtype: float64
        healthy sick
healthy     0.7  0.3
sick        0.4  0.6
        sleeping eating pooping
healthy      0.2    0.6     0.2
sick         0.4    0.1     0.5
[('healthy', 'healthy', 'healthy'), ('healthy', 'healthy', 'sick'), ('healthy', 'sick', 'healthy'), ('healthy', 'sick', 'sick'), ('sick', 'healthy', 'healthy'), ('sick', 'healthy', 'sick'), ('sick', 'sick', 'healthy'), ('sick', 'sick', 'sick')]
2.635148159999999e-07
(2.0329747199999986e-07, ('healthy', 'healthy', 'healthy'))
```

week-7:

source code:

```
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
dataset = pd.read_csv('Salary_Data.csv')
dataset.head()
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
y_pred
y_test
plt.scatter(X_train, y_train, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue')
plt.title("Salary vs Experience (Training set)")
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.show()
```



Week-9:

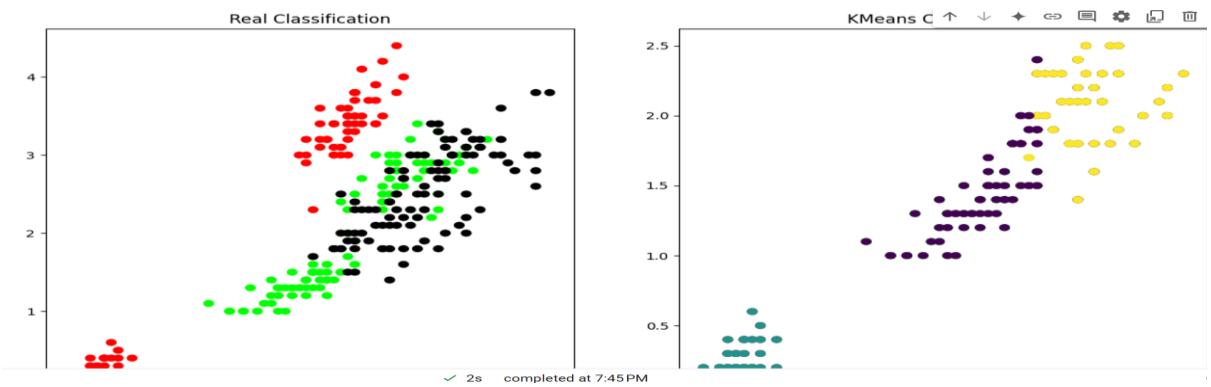
Source code:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import pandas as pd
import numpy as np

iris = datasets.load_iris()
x = pd.DataFrame(iris.data)
x.columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
y = pd.DataFrame(iris.target, columns=['target'])

plt.figure(figsize=(14, 7))
colormap = np.array(['red', 'lime', 'black'])

plt.subplot(1, 2, 1)
plt.scatter(x.sepal_length, x.sepal_width, c=colormap[y.target], s=40)
plt.title('Sepal')
```

```
plt.subplot(1, 2, 2)
plt.scatter(x.petal_length, x.petal_width, c=colormap[y.target], s=40)
plt.title('Petal')

model = KMeans(n_clusters=3)
model.fit(x)
print(model.labels_)

plt.subplot(1, 2, 1)
plt.scatter(x.petal_length, x.petal_width, c=colormap[y.target], s=40)
plt.title('Real Classification')

plt.subplot(1, 2, 2)
plt.scatter(x.petal_length, x.petal_width, c=model.labels_, s=40)
plt.title('KMeans Classification')

plt.show()
```

output:

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```



Week-10:

Source code:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
iris = load_iris()
print(iris.keys())
df = pd.DataFrame(iris['data'])
print(df)
print(iris['target_names'])
print(iris['feature_names'])
print(iris['target'])
X = df
y = iris['target']
from sklearn.neighbors import KNeighborsClassifier
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
y_pred = knn.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print("Correct prediction", accuracy_score(y_test, y_pred))
print("Wrong prediction", (1 - accuracy_score(y_test, y_pred)))
y_test_train = knn.predict(X_train)
cm1 = confusion_matrix(y_train, y_test_train)
print(cm1)
```

output:

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename',
'data_module'])
     0    1    2    3
```

```
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
..   ...  ...  ...  ...
145  6.7  3.0  5.2  2.3
146  6.3  2.5  5.0  1.9
147  6.5  3.0  5.2  2.0
148  6.2  3.4  5.4  2.3
149  5.9  3.0  5.1  1.8

[150 rows x 4 columns]
['setosa' 'versicolor' 'virginica']
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Correct prediction 0.98
Wrong prediction 0.020000000000000018
[[31  0  0]
 [ 0 33  2]
 [ 0  2 32]]
```

Week-11:

Source code:

```
import numpy as np
```

```
x = np.array([[2, 9], [1, 5], [3, 6]], dtype=float)
print(x)

y = np.array([[92], [86], [89]], dtype=float)
y = y / 100
print(y)

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def derivatives_sigmoid(x):
    return x * (1 - x)

epoch = 1000
lr = 0.01
input_layer_neurons = 2
hidden_layer_neurons = 2
output_neurons = 1

wh = np.random.uniform(size=(input_layer_neurons, hidden_layer_neurons))
bh = np.random.uniform(size=(1, hidden_layer_neurons))
wout = np.random.uniform(size=(hidden_layer_neurons, output_neurons))
bout = np.random.uniform(size=(1, output_neurons))

for i in range(epoch):
    hinp1 = np.dot(x, wh)
    hinp = hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1 = np.dot(hlayer_act, wout)
    outinp = outinp1 + bout
```

```
    output = sigmoid(outinp)
    EO = (y - output)
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)
    d_hiddenlayer = EH * hiddengrad
    wout += hlayer_act.T.dot(d_output) * lr
    bout += np.sum(d_output, axis=0, keepdims=True) * lr
    wh += x.T.dot(d_hiddenlayer) * lr
    bh += np.sum(d_hiddenlayer, axis=0, keepdims=True) * lr
    print("Actual output:" + str(y))
    print("Predicted output:" + str(output))
    print("Error:"+str(EO))
```

output:

```
[0.87169281]
 [0.87292951]]
Error:[[ 0.04699477]
 [-0.01169281]
 [ 0.01707049]]
Actual output:[[0.92]
 [0.86]
 [0.89]]
Predicted output:[[0.87302452]
 [0.87171213]
 [0.87294881]]
Error:[[ 0.04697548]
 [-0.01171213]
 [ 0.01705119]]
Actual output:[[0.92]
```

[0.86]

[0.89]]

Predicted output:[[0.87304378]

[0.87173142]

[0.87296808]]

Error:[[0.04695622]

[-0.01173142]

[0.01703192]]

Actual output:[[0.92]

[0.86]

[0.89]]

Predicted output:[[0.87306302]

[0.87175068]

[0.87298732]]

Error:[[0.04693698]

[-0.01175068]

[0.01701268]]

.

.

.

.

Week-12:

Source code:

```
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC
cancer = load_breast_cancer()
x = cancer.data[:, :2]
y = cancer.target
```

```
svm = SVC(kernel="rbf", gamma=0.5, C=1.0)
svm.fit(x, y)

DecisionBoundaryDisplay.from_estimator(
    svm,
    x,
    response_method="predict",
    cmap=plt.cm.Spectral,
    alpha=0.8,
    xlabel=cancer.feature_names[0],
    ylabel=cancer.feature_names[1]
)

plt.scatter(x[:, 0], x[:, 1],
            c=y,
            s=20, edgecolors="k")
plt.show()
```

output: