1

```
: import random
import csv

[14]: # Dataset
data = []
with open ('enjoysport.csv') as file:
reader = csv.reader(file)
for row in reader:
data.append(row)
print(data)

[['sunny', 'warm', 'normal', 'strong', 'warm', 'same', 'Yes'], ['sunny', 'warm','high', 'strong', 'warm', 'same',
'Yes'], ['rainy', 'cold', 'high', 'strong','warm', 'change', 'No'], ['sunny', 'warm', 'high', 'strong', 'cool', 'change',
'Yes']]

[15]: # No of attributes
n = len(data[0])-1
hypothesis = data[0].copy()[:-1]
print('Initial hypothesis', hypothesis)

Initial hypothesis ['sunny', 'warm', 'normal', 'strong', 'warm', 'same']

[16]: # Find S algorithm
for i in range (0, len(data)):
if data[i][n] == 'Yes':
for j in range (0, n):
if hypothesis[j] != '?' and hypothesis[j] != data[i][j]:
hypothesis[j] = '?'
print('Hypothesis after {} iteration {}'.format(i+1, hypothesis))

Hypothesis after 1 iteration ['sunny', 'warm', 'normal', 'strong', 'warm','same']
Hypothesis after 2 iteration ['sunny', 'warm', '?', 'strong', 'warm', 'same']
Hypothesis after 3 iteration ['sunny', 'warm', '?', 'strong', 'warm', 'same']
1
Hypothesis after 4 iteration ['sunny', 'warm', '?', 'strong', '?', '?']
[17]: print('Final Hypothesis : ', hypothesis)
```

2

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

[46]: # Loading dataset
X = []
y = []
```

```
with open ('c1.csv') as file:
reader = csv.reader(file)
for row in reader:
# Select every column except last column
X.append(row[:-1])
# Select last column
y.append(row[-1])
[47]: # Candidate Elimination algorithm
def learn (X, y):
# Number of attributes
n = len(X[0])
# Specific hypothesis
specific = X[0].copy()
# General hypothesis
general = [['?' for _ in range(n)] for _ in range(n)]
for i, h in enumerate(X):
if y[i] == 'Y':
for x in range (n):
if h[x] != specific[x]:
1
specific[x] = '?'
general[x][x] = '?'
elif y[i] == 'N':
for x in range (n):
if h[x] != specific[x]: general[x][x] = specific[x]
else: general[x][x] = '?'

# Remove elements from general hypothesis if its equal to [ '?', '?', '?', .
,→..., '?' ]

indices = [i for i, val in enumerate(general) if val == (['?'] * n)]
for _ in indices: general.remove(['?'] * n)
return specific, general
[48]: specific, general = learn(X,y)
print('Specific hypothesis', specific, sep='\n')
print()
print('General hypothesis', general, sep='\n')
```

3

```
import math
import csv
import pandas as pd
import numpy as np


def load():
  data = csv.reader(open('prog3 .csv', 'r'));
  d = list(data)
  h = d.pop(0)
  return d,h

class Node:
  def __init__(self, attribute):
    self.attribute = attribute
    self.children = []
    self.answer = ""

def sub(data, col, delete):
  dic = {}
  # print(col, data[col])
  coldata = [row[col] for row in data]
  attr = list(set(coldata))

  for i in attr : dic[i] = []

  for i in range(len(data)):
    key = data[i][col]
    if delete : del data[i][col]
    dic[key].append(data[i])

  return attr ,dic

def entropy (row):
  attr = list(set(row))
  if len(attr) == 1: return 0;

  arr = [0] * len(attr)
  for i in range(len(attr)) : arr[i] = sum([1 for x in row if x == attr[i]])/(len(row)*1.0)
  ans = 0
  for i in arr : ans += -1*i*math.log(i,2)
  return ans

def compute_gain (data, col):
  # print("hi" ,col)
  attr, dic = sub(data, col, delete=False)
  total_entropy = entropy([row[-1] for row in data])

  for i in range(len(attr)):
    ratio = len(dic[attr[i]])/(len(data)*1.0)
    entro = entropy([row[-1] for row in dic[attr[i]]])
    total_entropy -= ratio*entro

  return total_entropy


def build_tree(data, header):
  y = [row[-1] for row in data]
  if len(set(y)) == 1:
    node = Node("")
    node.answer = y[0]
    return node

  n = len(data[0])-1
  gains = [compute_gain(data , c) for c in range(n)]
  split = gains.index(max(gains))
  node = Node(header[split])
  fea = header[:split] + header[split+1:]

  attr, dic = sub(data, split, delete=True)

  for i in range(len(attr)):
    child = build_tree(dic[attr[i]] , fea)
    node.children.append((attr[i] , child))

  return node


def print_tree (node, level):
  if node.answer != "":
    print("---"*level, node.answer)
    return
  print("---"*level, node.attribute)
  for value, n in node.children:
    print("---"*(level+1), value)
    print_tree(n, level+2)

data , header = load()


node = build_tree(data, header)

print_tree(node, 0)

data

header
```

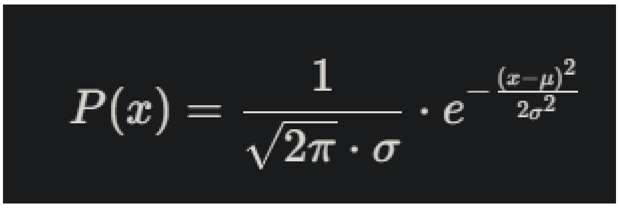
4. Back propagation algorithm

```
import numpy as np
from numpy import random as ran


# defining this
iter = 1000
rate = 0.2
input = 2
hid = 3
out = 1

wh = ran.uniform(size = (input , hid))
bh = ran.uniform(size = (1 , hid))
wout = ran.uniform(size = (hid , out))
bout = ran.uniform(size = (1, out))

# definnin input and output
x = np.array(([2,9], [1,5], [3,6]) , dtype = float)
y = np.array(([92] , [86] , [89]), dtype = float)

# normalize the input
x = x/np.amax(x,axis=0)
y = y/100

# activation finction
```

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

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

for _ in range(iter):
  ah = np.dot(x, wh) + bh
  h_act = sigma(ah)
  aout = np.dot(h_act, wout) + bout
  o_act = sigma(aout)

  e_out = y - o_act
  d_out = e_out * sigma_rev(o_act)

  e_h = d_out.dot(wout.T)
  d_h = e_h * sigma_rev(h_act)

  wout += h_act.T.dot(d_out)
  wh += x.T.dot(d_h)


print("Normalised Input",x, sep='\n')
print()
print("Actual Output", y, sep='\n')
print()
print("Predicted Output", o_act, sep='\n')
```

5. Naive bayesian classifier

```
import csv , math, random
import statistics as st

def load():
  d = csv.reader(open('prog5.csv', 'r'))
  data = list(d)
  data.pop(0)
  for i in range(len(data)):
    data[i] = [float(x) for x in data[i]]

  return data

def data_split(data):
  test = []
  test_size = int(len(data)*(0.2))
  for i in range(test_size):
    indi = random.randrange(test_size)
    test.append(data.pop(indi))

  return data,test

def separate(data):
  dic = {}
  for i in range(len(data)):
    if data[i][-1] not in dic:
      dic[data[i][-1]] = []
    dic[data[i][-1]].append(data[i])
  return dic

def mean_cal(data):
  arr = [(st.mean(x) , st.stdev(x)) for x in zip(*data)]
  del arr[-1]
  return arr

def summ(data):
  dic = separate(data)
  sum = {}
  for a, b in dic.items():
    sum[a] = mean_cal(b)
  return sum
```



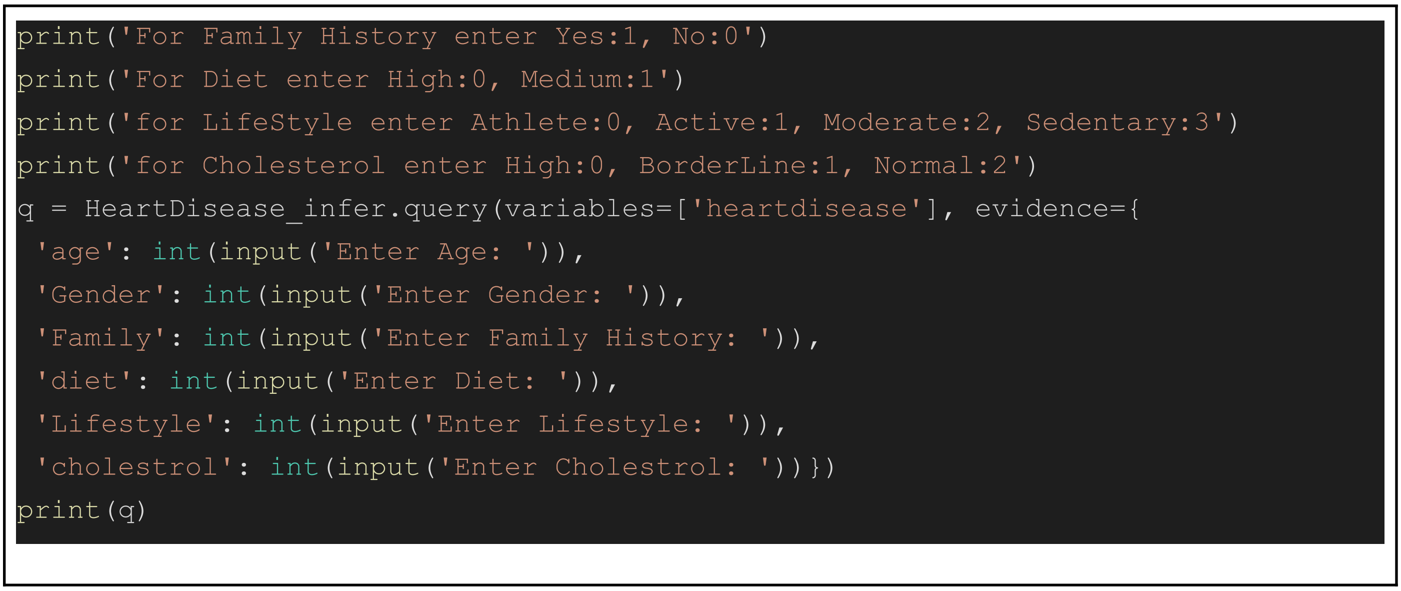
$$P(x) = \frac{1}{\sqrt{2\pi}\cdot\sigma}\cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

```
def find_prob(x,mean,std):
  exp = math.exp((-1)*((  math.pow((x-mean), 2)  )/(2*math.pow(std,2))))
  return (1 / (math.sqrt(2*math.pi) * mean )) * exp

def prob_row(summary, x):
  dic = {}
  # print(summary)
  for cls,value in summary.items():
    dic[cls] = 1
    for i in range(len(value)):
      dic[cls] -= find_prob(x[i] , value[i][0], value[i][1])
  # print(dic)
  return dic

def pred_row(Sum , X):
  # print(Sum)
  # print(X)
  dic = prob_row(Sum,X)
  # print("hi" , dic)
  if dic[0] > dic[1]:
    return 1

  return 0

def predict(Sum , test_set):
  pred = []
  # print(Sum)
  # print(test_set)
  for i in range(len(test_set)):
    pred.append(pred_row(Sum , test_set[i]))
  return pred
```

```
def get_accuracy(test_set, pred):
  ans = 0
  for i in range(len(test_set)):
    if test_set[i][-1] == pred[i]:
      ans+=1
  print(ans)
  return (float(ans)/ float(len(test_set)))*100

data = load()
train_set, test_set = data_split(data)
print(len(train_set))
len(test_set)

summarize = summ(train_set)
summarize

prediction = predict(summarize , test_set)

accuracy = get_accuracy(test_set , prediction)
accuracy
```

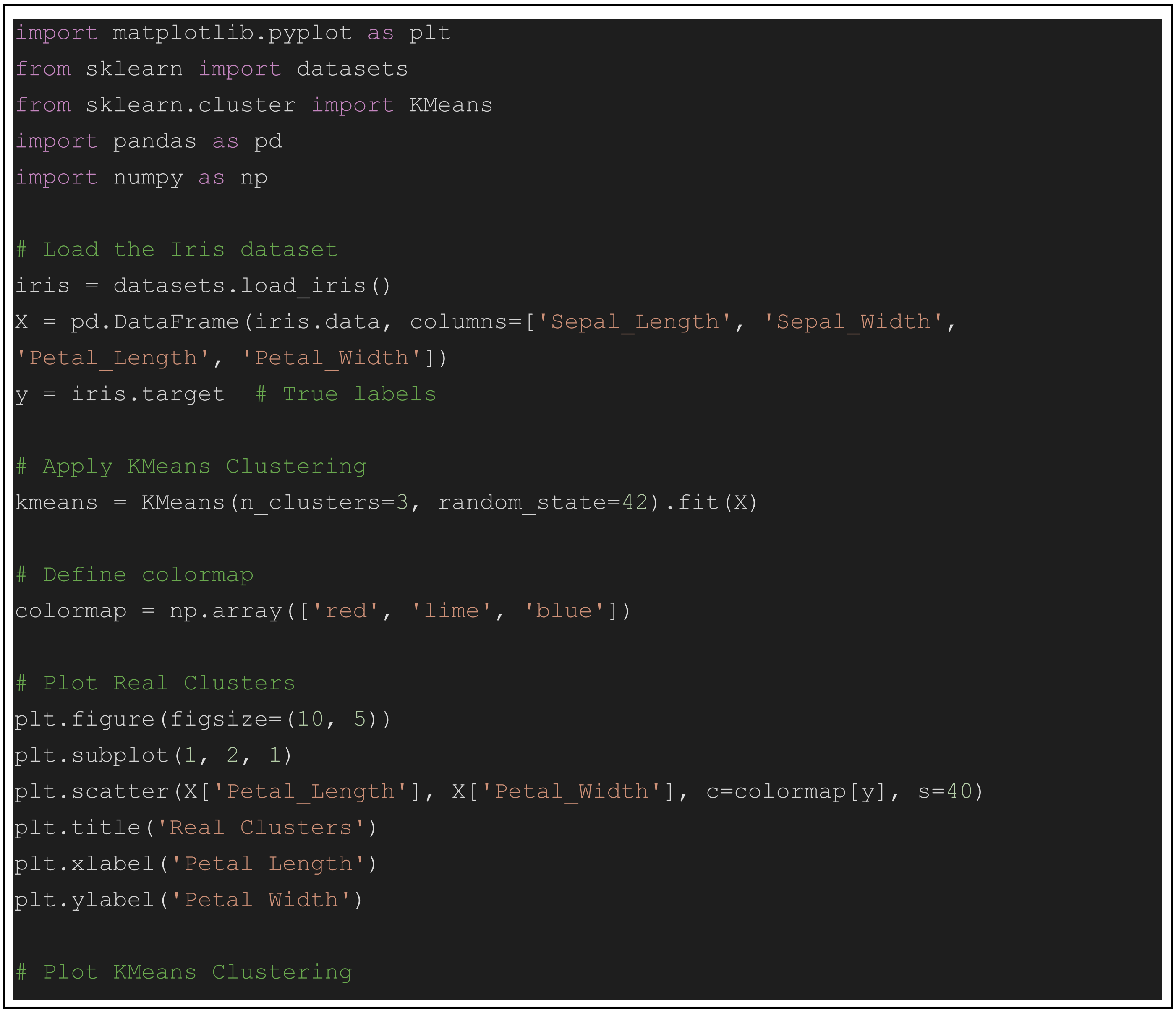
6. Heart disease

```
import pandas as pd
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
data = pd.read_csv("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)
```

7: KmeansCluster



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

# Load the Iris dataset
iris = datasets.load_iris()
X = pd.DataFrame(iris.data, columns=['Sepal_Length', 'Sepal_Width',
'Petal_Length', 'Petal_Width'])
y = iris.target  # True labels

# Apply KMeans Clustering
kmeans = KMeans(n_clusters=3, random_state=42).fit(X)

# Define colormap
colormap = np.array(['red', 'lime', 'blue'])

# Plot Real Clusters
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.scatter(X['Petal_Length'], X['Petal_Width'], c=colormap[y], s=40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# Plot KMeans Clustering
```

```
plt.subplot(1, 2, 2)
plt.scatter(X['Petal_Length'], X['Petal_Width'], c=colormap[kmeans.labels_],
s=40)
plt.title('KMeans Clustering')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

# plt.tight_layout()
# plt.show()
```

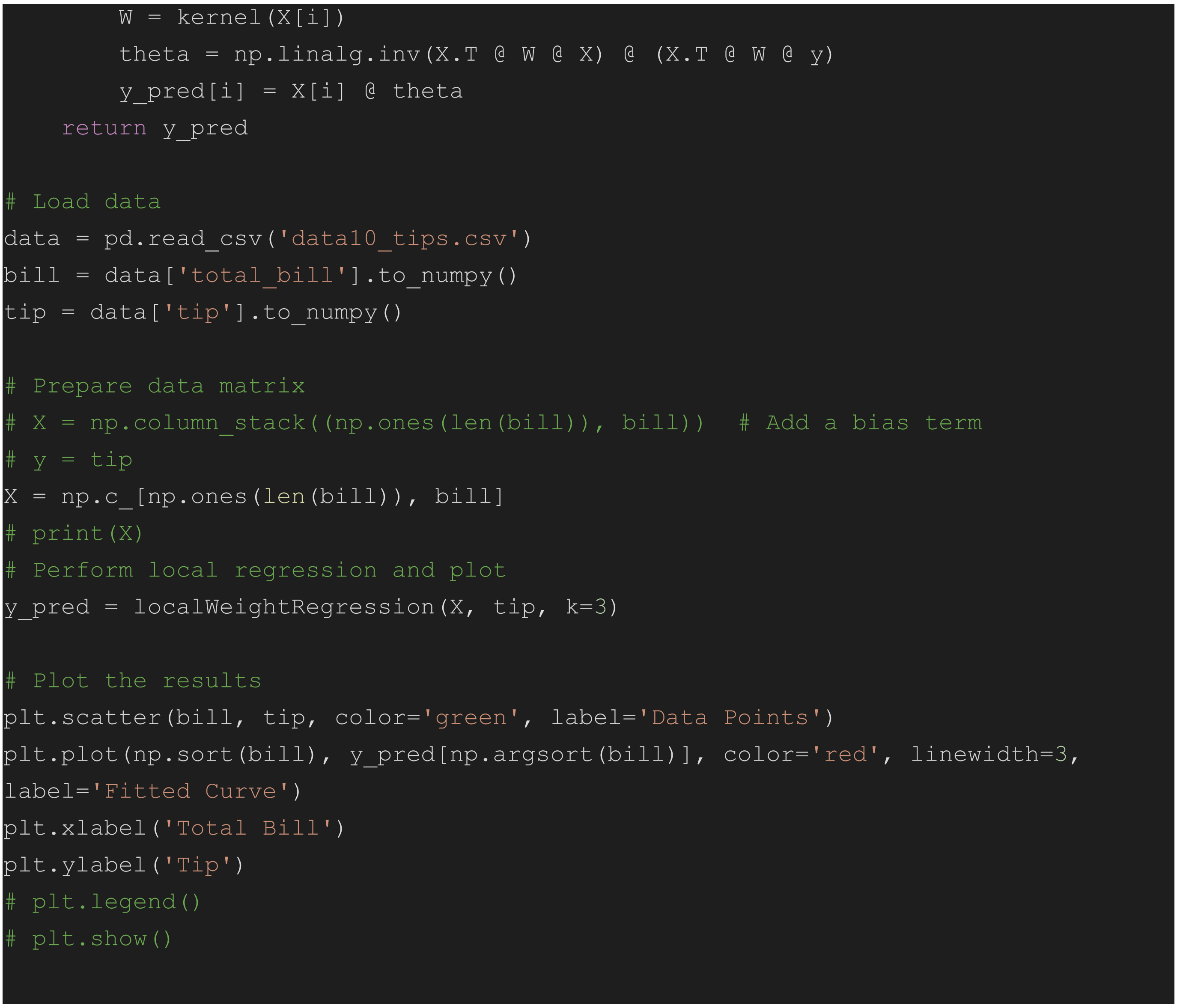
8: kNN neighbors

```
# import the required packages
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets

# Load dataset
iris=datasets.load_iris()
print("Iris Data set loaded...")
# Split the data into train and test samples
x_train, x_test, y_train, y_test =
train_test_split(iris.data,iris.target,test_size=0.1)
print("Dataset is split into training and testing...")
print("Size of trainng data and its label",len(x_train),
x_train.shape,y_train.shape)
print("Size of trainng data and its label",x_test.shape, y_test.shape)
# Prints Label no. and their names
# print(iris)
for i in range(len(iris.target_names)):
    print("Label", i , "-",str(iris.target_names[i]))

# Create object of KNN classifier
classifier = KNeighborsClassifier(6)

# Perform Training
classifier.fit(x_train, y_train)
# Perform testing
y_pred=classifier.predict(x_test)

# Display the results
print("Results of Classification using K-nn with K=1")
for r in range(0,len(x_test)):
    print(" Sample:", str(x_test[r]), "Actual-label:", str(y_test[r]), " Predicted-label:",str(y_pred[r]))
print("Classification Accuracy :" , classifier.score(x_test,y_test));
```

9. Locally weighted regression algorithm

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.nonparametric.smoothers_lowess import lowess

# Load data
data = pd.read_csv('data10_tips.csv')
bill = data['total_bill'].to_numpy()
tip = data['tip'].to_numpy()

# Apply lowess
smoothed = lowess(tip, bill, frac=0.3)  # `frac` controls the smoothing parameter

# Extract smoothed values
smoothed_x, smoothed_y = smoothed[:, 0], smoothed[:, 1]

# Plot the results
plt.scatter(bill, tip, color='green', label='Data Points')
plt.plot(smoothed_x, smoothed_y, color='red', linewidth=2, label='Fitted Curve')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.legend()
plt.show()
```

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

def localWeightRegression(X, y, k):
    def kernel(point):
        weights = np.exp(-np.square(X[:, 1] - point[1]) / (2 * k**2))
        return np.diag(weights)

    y_pred = np.zeros(len(X))
    for i in range(len(X)):
```



```
        W = kernel(X[i])
        theta = np.linalg.inv(X.T @ W @ X) @ (X.T @ W @ y)
        y_pred[i] = X[i] @ theta
    return y_pred

# Load data
data = pd.read_csv('data10_tips.csv')
bill = data['total_bill'].to_numpy()
tip = data['tip'].to_numpy()

# Prepare data matrix
# X = np.column_stack((np.ones(len(bill)), bill))  # Add a bias term
# y = tip
X = np.c_[np.ones(len(bill)), bill]
# print(X)
# Perform local regression and plot
y_pred = localWeightRegression(X, tip, k=3)

# Plot the results
plt.scatter(bill, tip, color='green', label='Data Points')
plt.plot(np.sort(bill), y_pred[np.argsort(bill)], color='red', linewidth=3,
label='Fitted Curve')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
# plt.legend()
# plt.show()
```