DWDM lab

QUESTION -1

```
# nomral mean

import statistics as st

data1 = [1, 3, 4, 5, 7, 9, 2,4]

normal_mean = st.mean(data1)

print("normal mean : ",normal_mean)


# harmonic mean

print("Harmonic Mean is % s " % (st.harmonic_mean(data1)))

### weighted mean

import pandas as pd


def weighted_average(dataframe, value, weight):
    val = dataframe[value]
    wt = dataframe[weight]
    return (val * wt).sum() / wt.sum()


# creating a dataframe to represent different
# items and their corresponding weight and value
dataframe = pd.DataFrame({'item_name': ['Chocolate', 'Chocolate',
                                        'Chocolate', 'Biscuit',
                                        'Biscuit', 'Biscuit',
                                        'IceCream', 'IceCream',
                                        'IceCream'],
                          'value': [90, 50, 86, 87, 42, 48,
                                    68, 92, 102],
                          'weight': [4, 2, 3, 5, 6, 5, 3, 7,
                                     5]})

# Weighted average of value  grouped by item name
print(dataframe.groupby('item_name').apply(weighted_average,
                                     'value', 'weight'))


# Geometric Mean
# ==============
from scipy.stats.mstats import gmean
arr1 = gmean([1, 3, 27])

print("Geometric Mean is :", arr1)
```

```
# Printing median of the
# random data-set
print("Median of data-set is : % s "
        % (st.median(data1)))


# Printing out mode of given data-set
print("Mode of given data set is % s" % (st.mode(data1)))
```

2 . QUESTION-2

```
# Function will automatically calculate
# it's mean and set it as xbar
print("Variance of sample set is % s"
      %(st.variance(data1)))

# standard deviation
print("standard deviation : ",st.stdev(data1))


#skewness
from scipy.stats import skew
print("skewness = ",dataframe['value'].skew())

# quartiles
import numpy as np

print("quantile = ",np.quantile(data1,.25))

# percentile
print("percentile = ", np.percentile(data1,25))

# range
# using range for iteration
l = [10, 20, 30, 40]
for i in range(len(l)):
    print(l[i], end=" ")
print()


# arange fucn
print(np.arange(1,5))
```

3 . QUESTION - 3

```
# correlation
import numpy as np
from scipy.stats import pearsonr
from scipy.stats import spearmanr
from scipy.stats import kendalltau
```

```
x = np.arange(10, 20)
y = np.array([2, 1, 4, 5, 8, 12, 18, 25, 96, 48])

np.corrcoef(x,y)
# covariance

print(np.cov(x,y))
```

4. QUESTION - 4

```
# box plot
import matplotlib.pyplot as plt

fig = plt.figure(figsize =(10, 7))

plt.boxplot(dataframe['value'])
# dataframe.boxplot(['value'])


# histogram

fig1, axs = plt.subplots(1, 1,
                        figsize =(10, 7),
                        tight_layout = True)
axs.hist(dataframe['value'] , bins=20)

dataframe.hist(['value'])


# pie chart

plt.pie(dataframe['value'],labels=dataframe['item_name'])

# bar chart
ax = fig.add_axes([0,0,1,1])
ax.bar(dataframe['value'],dataframe['item_name'],color ='maroon', width = 0.5)
plt.show()




# plt.style.use('bmh')
# x = md['Age']
# y = md['Obesity']# Bar chart
# plt.title("cancer patients ")
# plt.xlabel('Age', fontsize=16)
# plt.ylabel('Obesity', fontsize=16)
# plt.bar(x,y)
# plt.show()


# heatmap
import seaborn as sn
d = dataframe['value']
```

```
sn.heatmap(d[:, np.newaxis])
sn.heatmap(dataframe.corr(),annot=True)
```

5. QUESTION _ 5

```
# handling missing values
# ======================

# fill with 0
md = df['column_name'].fillna(0) #
print(md)
#

# fill with mean
mean_df = df['age'].fillna(mean)
print(mean_df)

# drop columns

drop = df.dropna()  # rows with null values
drop = df.dropna(how = 'all') #if all values in rows are null
drop = df.dropna(axis='1') #drop columns with atleast 1 null value
drop = df.dropna(axis='1' , subset=['value']) #drop only value column



# Data Smoothing using Binning
# ======================
data = md['Age']
data = data[:50]
data=np.sort(data)
print(data)
b1=np.zeros((10,5))
b2=np.zeros((10,5))
b3=np.zeros((10,5))
b4=np.zeros((10,5))
b5=np.zeros((10,5))
for i in range (0,50,5):
  k=int(i/5)
  mean=(data[i] + data[i+1] + data[i+2] )/5
  for j in range(5):
    b1[k,j]=mean
print("----------------Mean Bin:----------------- \n",b1)
for i in range (0,50,5):
  k=int(i/5)
  for j in range (5):
    if (data[i+j]-data[i]) < (data[i+2]-data[i+j]):
      b3[k,j]=data[i]
    else:
      b3[k,j]=data[i+2]
print("-----------------Boundary Bin:----------------- \n",b3)
```

6. QUESTION-6

```
!pip install apyori
from apyori import apriori
store_data = pd.read_csv('store_data_Association.csv',header=None)
store_data.head()
store_data.shape
records = []
for i in range(0, 7501):
    records.append([str(store_data.values[i,j]) for j in range(0, 20)])
association_rules = apriori(records, min_support=0.004, min_confidence=0.3,min_lift=3,min_length=1)
association_results = list(association_rules)
print(len(association_results))
for item in association_results:

    # first index of the inner list
    # Contains base item and add item
    pair = item[0]
    items = [x for x in pair]
    print("Rule :"+ str(items[0]) + "->" + str(items[1]))

    #shortform - SCL
    #==============
    #second index of the inner list
    print("Support: " + str(item[1]))

    #third index of the list located at 0th
    #of the third index of the inner list

    print("Confidence: " + str(item[2][0][2]))
    print("Lift: " + str(item[2][0][3]))
    print("=====================================")
```

7.QUESTION 13

```
import random
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
def rand(s,e,n):
  arr=[]
  for i in range(n):
    arr.append(random.randint(s,e))
  return arr

x = rand(20,60,30)
y = rand(50,90,30)


print(x)
print(y)


df = pd.DataFrame({
    'x':x,
    'y':y
})

kmeans = KMeans(n_clusters = 2).fit(df)
centeroid = kmeans.cluster_centers_
print("Centeroids : ",centeroid)
plt.scatter(df['x'],df['y'],c=kmeans.labels_.astype(float))
plt.scatter(centeroid[:,0],centeroid[:,1],color="red")
```

classification report

```
from sklearn.metrics import classification_report

# or you can use "x_test" in case of predictions
print(classification_report(predictions,y_test))
```

confusion matrix

```
predictions = clf.predict(X_test)
# use clf.labels_  incase classes_ throws an error
>>> cm = confusion_matrix(y_test, predictions, labels=clf.classes_)
>>> disp = ConfusionMatrixDisplay(confusion_matrix=cm,
...                               display_labels=clf.classes_)
>>> disp.plot()
<...>
>>> plt.show()
```