DATA MIING ASSIGNMENT 3 (PART 1)

Q1) The Apriori Algorithm employs an iterative approach which is known as level wise search, where k itemsets are used to explore k+1 itemsets. It is easy to implement and can be used for large datasets unlike the process of manually calculating the support of every subset of an itemset to find frequent itemsets. Moreover according to the Apriori Property:-

> "All non-empty subsets of a frequent itemset must also be frequent".

Hence it decreases the size of the new k+1 itemsets, which in turn reduces the complexity of our algorithm making it a better choice over manually calculating support of subsets to obtain frequent itemsets.

Q2) If L_1 denotes the set of frequent 1 item sets, for $K \geq 2$ every K-itemset will be a superset of an itemset in L_1. This stands true because of the Apriori Property which says that, "All non-empty subsets of a frequent itemset must also be frequent". For example:-

If $Y \subseteq Z$ and $X \subseteq Y$

> Here if Y is frequent (support Y is greater than some threshold S_Y) and $X \subseteq Y$, then X is also frequent. ($S_X > S_Y >$ some threshold) a frequent-1 itemset and 'Y' is its superset with ($K \geq 2$).

↳ Due to the Apriori Property.

According to the Apriori algorithm, every k+1 itemset is derived from L_K by checking the support threshold of the itemsets, hence always L_{k+1} will be derived from L_K with pruning / joining operations, making L_{k+1} a superset of L_K.

Q3

l_1 l_2 l_3 l_4 l_5

Let $L_2 = \{\{1,2\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,5\}\}$

Compute C_3 that is obtained by joing every pair of joinable itemsets from L_2.

An itemset is said to be joinable with another itemset when the first element of both the itemsets are same.

l_1 and l_2 are joinable if, (length of itemset)

$l_1[i] = l_2[i]$, $l_1[k-1] < l_2[k-1]$, and $i < k-1$

∴

$C_3 = \{\{1,2,4\}, \{2,3,4\}\}$

{1,2,4} joined l_1 and l_2

{2,3,4} joined l_3 and l_4

Q4

Here

S_1 denotes the support of

A → {boarding pass, passport} ⇒ {flight}

and S_2 denotes

B → {boarding pass} ⇒ {flight}

It can be seen that B is a subset of A. Hence the number of times 'A' will occur in a transaction dataset will always be less than or equal to the times B occurs. (as B can also occur individu)

For example:-

B	Boarding Pass	Passport
T_1	1	1
T_2	7	0
T_3	2	4
T_4	0	0
T_5	9	11
T_6	0	7
T_7	0	8

From this table we can see that the number of times Boarding Pass occurs is '4' times.

The number of times Boarding Pass and Passport both occur together which is 'A' is '3'.

Support of an itemset is calculated by

$S = \frac{N_A}{N}$ which number of times the itemset occurs in the transaction set divided by total number of transactions.

$\therefore \quad S_2 = \frac{N_B}{N} = \frac{4}{7} = 0.571$

$S_1 = \frac{N_A}{N}$ (Both occuring) $= \frac{3}{7} = 0.428.$

$\therefore$ Every time the support of a subset of a data will be be greater than or equal to the support of its superset as the subset will always occur with superset or may occur individually.

Hence $\boxed{S_2 > S_1}$

Q5.

```
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]
```

The support of any itemset is given by :-

$$\frac{N_A}{N} = \frac{\text{Number of times A occurs in a transaction}}{\text{Total number of transactions}}$$

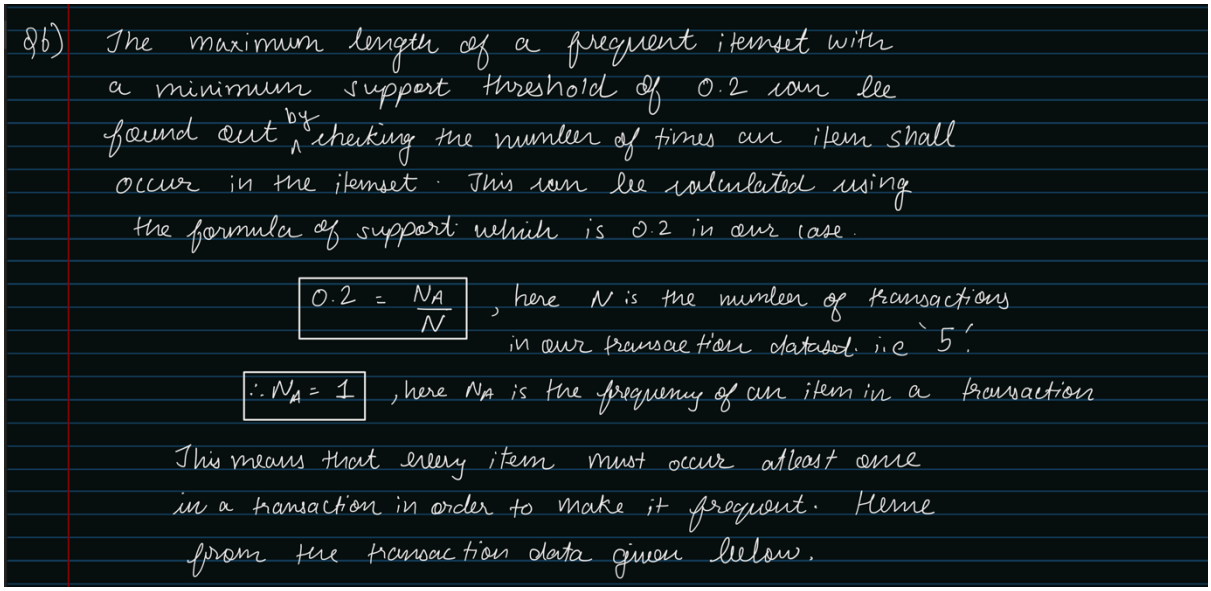
From the dataset given above the {eggs} occurs `4 times.

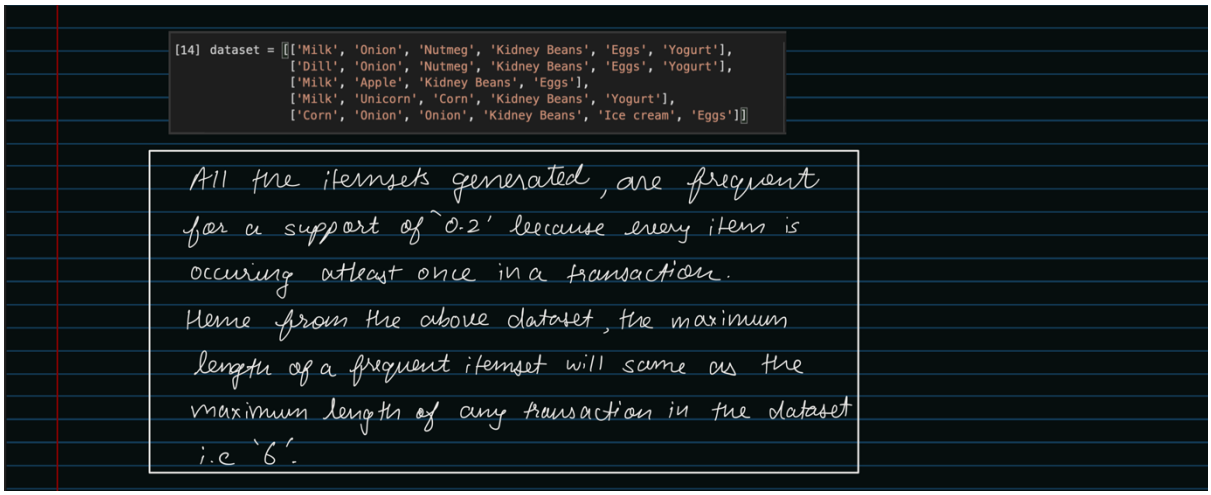
$\boxed{N_A = 4}$

and the total number of transactions are `5'. $\boxed{N = 5}$

Hence the support for {eggs} $= \frac{N_A}{N}$

$\therefore \boxed{S = \frac{4}{5} = 0.8}$





Hence we obtained the frequent itemsets for a minimum support threshold of 0.2.

Answer:- The maximum length of a frequent itemset with a minimum support threshold of 0.2 is 6.

Question 7:-

```
from mlxtend.frequent_patterns import association_rules

strong_rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0)
df = strong_rules
df
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Kidney Beans)	(Eggs)	1.0	0.8	0.8	0.80	1.00	0.00	1.0
1	(Eggs)	(Kidney Beans)	0.8	1.0	0.8	1.00	1.00	0.00	inf
2	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
3	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6
4	(Milk)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
5	(Kidney Beans)	(Milk)	1.0	0.6	0.6	0.60	1.00	0.00	1.0
6	(Onion)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
7	(Kidney Beans)	(Onion)	1.0	0.6	0.6	0.60	1.00	0.00	1.0
8	(Kidney Beans)	(Yogurt)	1.0	0.6	0.6	0.60	1.00	0.00	1.0
9	(Yogurt)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
10	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
11	(Onion, Eggs)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
12	(Kidney Beans, Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6
13	(Onion)	(Kidney Beans, Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
14	(Kidney Beans)	(Onion, Eggs)	1.0	0.6	0.6	0.60	1.00	0.00	1.0
15	(Eggs)	(Onion, Kidney Beans)	0.8	0.6	0.6	0.75	1.25	0.12	1.6

Converting the datframe columns "antecedents" and "consequents" to string in order to use df.query.

```
df["antecedents"] = df["antecedents"].astype("unicode")
df["consequents"] = df["consequents"].astype("unicode")
df
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	frozenset({'Kidney Beans'})	frozenset({'Eggs'})	1.0	0.8	0.8	0.80	1.00	0.00	1.0
1	frozenset({'Eggs'})	frozenset({'Kidney Beans'})	0.8	1.0	0.8	1.00	1.00	0.00	inf
2	frozenset({'Onion'})	frozenset({'Eggs'})	0.6	0.8	0.6	1.00	1.25	0.12	inf
3	frozenset({'Eggs'})	frozenset({'Onion'})	0.8	0.6	0.6	0.75	1.25	0.12	1.6
4	frozenset({'Milk'})	frozenset({'Kidney Beans'})	0.6	1.0	0.6	1.00	1.00	0.00	inf
5	frozenset({'Kidney Beans'})	frozenset({'Milk'})	1.0	0.6	0.6	0.60	1.00	0.00	1.0
6	frozenset({'Onion'})	frozenset({'Kidney Beans'})	0.6	1.0	0.6	1.00	1.00	0.00	inf
7	frozenset({'Kidney Beans'})	frozenset({'Onion'})	1.0	0.6	0.6	0.60	1.00	0.00	1.0
8	frozenset({'Kidney Beans'})	frozenset({'Yogurt'})	1.0	0.6	0.6	0.60	1.00	0.00	1.0
9	frozenset({'Yogurt'})	frozenset({'Kidney Beans'})	0.6	1.0	0.6	1.00	1.00	0.00	inf
10	frozenset({'Onion', 'Kidney Beans'})	frozenset({'Eggs'})	0.6	0.8	0.6	1.00	1.25	0.12	inf
11	frozenset({'Onion', 'Eggs'})	frozenset({'Kidney Beans'})	0.6	1.0	0.6	1.00	1.00	0.00	inf
12	frozenset({'Kidney Beans', 'Eggs'})	frozenset({'Onion'})	0.8	0.6	0.6	0.75	1.25	0.12	1.6
13	frozenset({'Onion'})	frozenset({'Kidney Beans', 'Eggs'})	0.6	0.8	0.6	1.00	1.25	0.12	inf
14	frozenset({'Kidney Beans'})	frozenset({'Onion', 'Eggs'})	1.0	0.6	0.6	0.60	1.00	0.00	1.0
15	frozenset({'Eggs'})	frozenset({'Onion', 'Kidney Beans'})	0.8	0.6	0.6	0.75	1.25	0.12	1.6

Creating a function to calculate the Kulczynski measure of two itemsets. The Kulczynski measure KA,B ∈ [0, 1] of the itemsets A ⊆ I and B ⊆ I such that A ∩ B = ∅ is given by:-
K (A,B) = (VA⇒B + VB⇒A)/2 The Kulczynski measure KA,B can be interpreted as the average between the confidence that A ⇒ B and the confidence that B⇒A.

```
[11] def Kulczynski(a,b):
       a = frozenset(a) # converting input sets to frozensets
       b = frozenset(b)
       x = df.query(f'antecedents == "{a}"' and f'consequents == "{b}"')
       c1 = x['confidence'].values[0]   # extracting the confidence values for sets a and b (input) from our dataframe
       print(f'Confidence of A⇒B',c1)
       y = df.query(f'antecedents == "{b}"' and f'consequents == "{a}"')
       c2= y['confidence'].values[0]
       print(f'Confidence of B⇒A',c2)
       print('The Kulczynski measure is given by',(c1+c2)/2) # taking out average of the two confidence values
```

```
# setting our inputs to the function
a = ['Onion']
b = ['Kidney Beans','Eggs']
```

```
[13] # running the functions on the desired itemsets A={Onion}  and B={Kidney Beans,Eggs}
     Kulczynski(a,b)
```

```
Confidence of A⇒B 1.0
Confidence of B⇒A 0.7499999999999999
The Kulczynski measure is given by 0.875
```

Question 8

```
[34] dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
                ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
                ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
                ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
                ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]
```

The imbalance ratio I A,B is the ratio between the absolute difference between the support count of A and the support count of B and the number of transactions that contain A, B, or both A and B. The formula for imbalance ratio is given by:-
$I_{A,B} = (|N_A - N_B|)/(N_A + N_B - N_{A \cup B})$

```
# creating a function to calculate the imbalance ratio

def imbalance_ratio(a,b):
  count_a = 0
  count_b = 0
  count_ab = 0
  ab = a+b
  for i in range(len(dataset)):
    if set(a) <= set(dataset[i]): # checking if the set a is present in our dataset and adding a counter to get its frequency
      count_a = count_a + 1
    if set(b) <= set(dataset[i]): # checking if the set a is present in our dataset and adding a counter to get its frequency
      count_b = count_b + 1
    if set(ab) <= set(dataset[i]): # checking if the both set a and set b are present in our dataset and adding a counter to get its frequency
      count_ab = count_ab + 1
  imb_ratio = abs((count_a - count_b)/(count_a + count_b - count_ab)) #calculating the imbalance ratio with the above mentioned formula
  print(f'The imbalance ratio of {a} and {b} is:- ',imb_ratio)
```

```
[36] A = ['Onion']
     B = ['Kidney Beans','Eggs']
     imbalance_ratio(A,B)
```

```
The imbalance ratio of ['Onion'] and ['Kidney Beans', 'Eggs'] is:-  0.25
```

Creating a function to calculate the Kulczynski measure of two itemsets. The Kulczynski measure $K_{A,B} \in [0, 1]$ of the itemsets $A \subseteq I$ and $B \subseteq I$ such that $A \cap B = \varnothing$ is given by:-
$K(A,B) = (V_{A \Rightarrow B} + V_{B \Rightarrow A})/2$ The Kulczynski measure $K_{A,B}$ can be interpreted as the average between the confidence that $A \Rightarrow B$ and the confidence that $B \Rightarrow A$.

```
def Kulczynski(a,b):
  a = frozenset(a) # converting input sets to frozensets
  b = frozenset(b)
  x = df.query(f'antecedents == "{a}"' and f'consequents == "{b}"')
  c1 = x['confidence'].values[0]   # extracting the confidence values for sets a and b (input) from our dataframe
  print(f'Confidence of A⇒B',c1)
  y = df.query(f'antecedents == "{b}"' and f'consequents == "{a}"')
  c2= y['confidence'].values[0]
  print(f'Confidence of B⇒A',c2)
  print('The Kulczynski measure is given by',(c1+c2)/2) # taking out average of the two confidence values
```

+ Code + Text

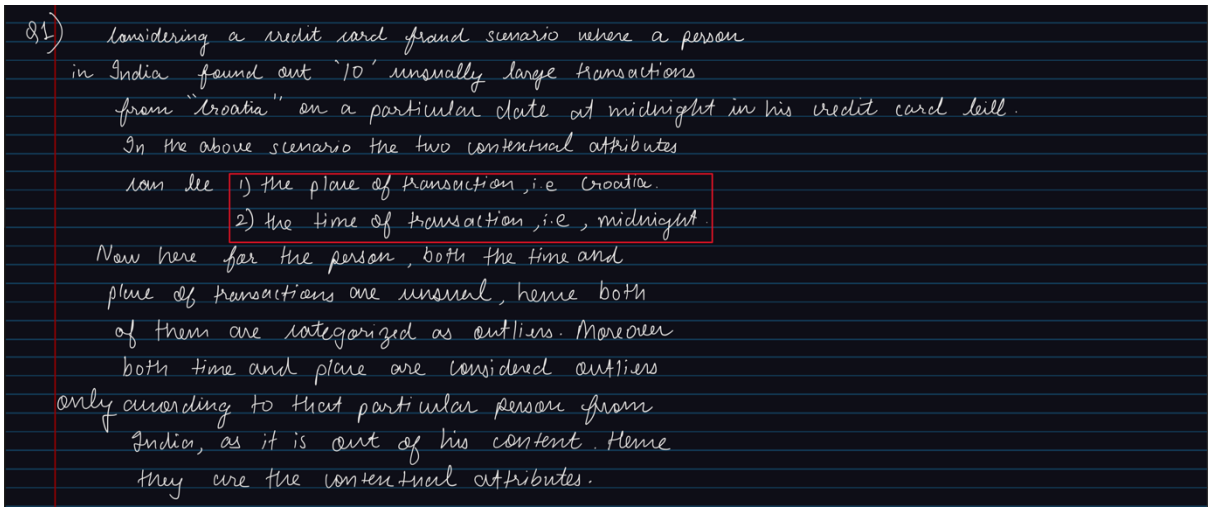
```
[32] # setting our inputs to the function
     a = ['Onion']
     b = ['Kidney Beans','Eggs']
```

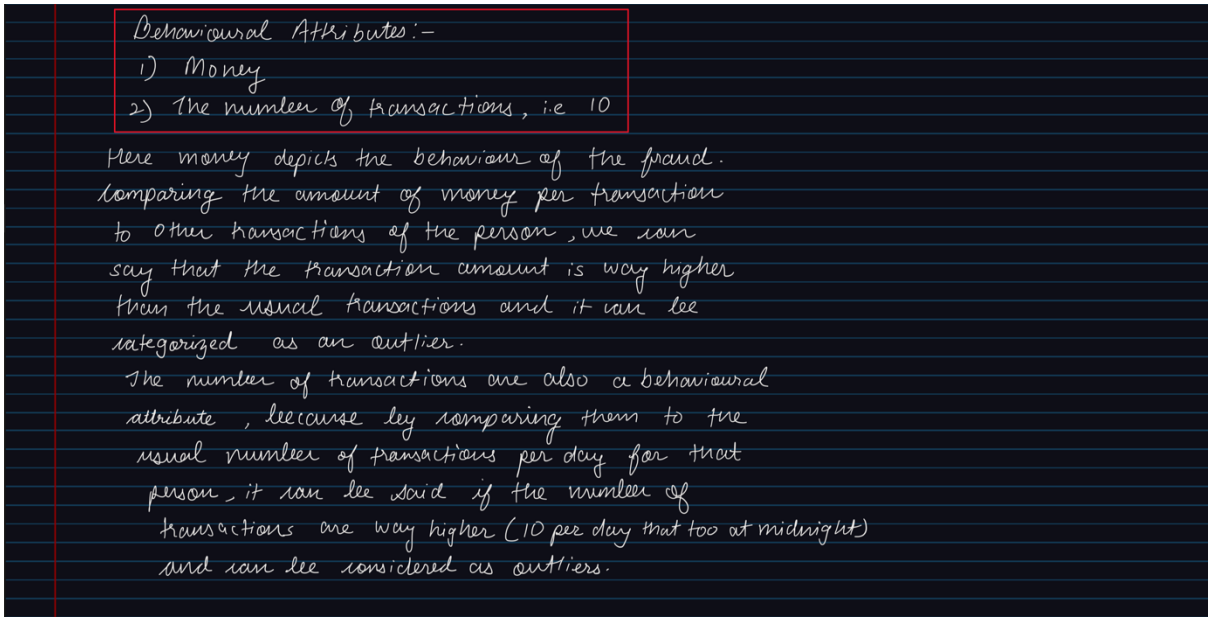
```
[33] # running the functions on the desired itemsets A={Onion}  and B={Kidney Beans,Eggs}
     Kulczynski(a,b)
```

```
Confidence of A⇒B 1.0
Confidence of B⇒A 0.7499999999999999
The Kulczynski measure is given by 0.875
```

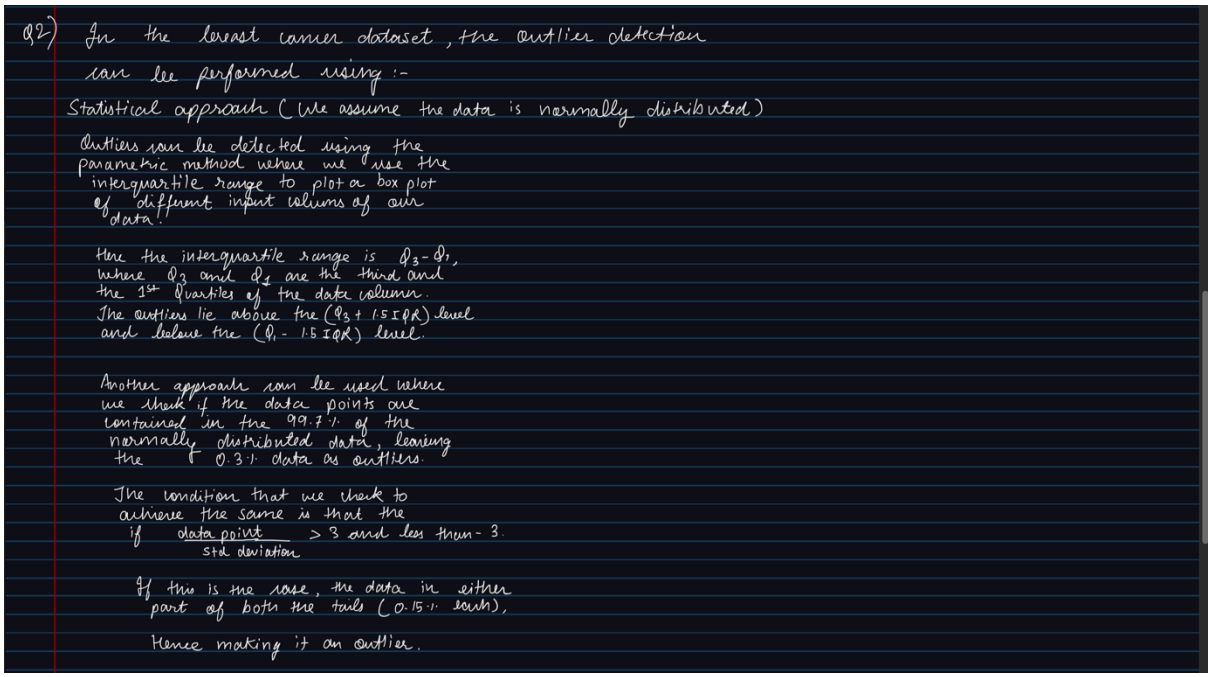
DATA MIING ASSIGNMENT 3 (PART 2)

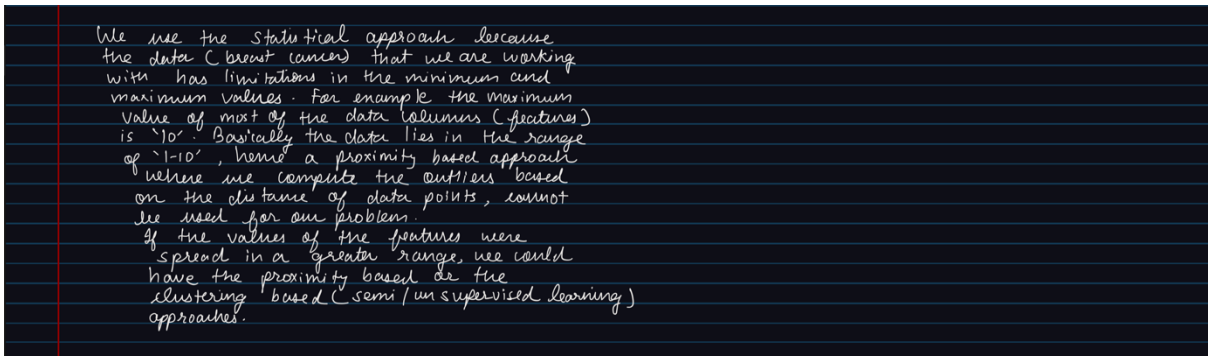
QUESTION 1:-





Question 2:-





Question 3 :-

Calculating the mean and standard deviation of the precipitation data

```
[8] import numpy as np
    precipitation =  [22.93, 20.69, 25.75, 23.84, 25.34, 3.25, 23.55, 28.28, 23.72, 22.42, 26.83, 23.82]

    np_precipitation = np.array(precipitation)

    mean = np.mean(np_precipitation)
    sd = np.std(np_precipitation)
    print('The mean and standard deviation of the data are:-')
    print('1)Mean:-',mean,'2)Standard Deviation:- ',sd)
```

```
The mean and standard deviation of the data are:-
1)Mean:- 22.534999999999997 2)Standard Deviation:-  6.130045540885756
```

To check that the a datapoint is an outlier, it needs to satisfy a condition where the distance of the datapoint form mean divided by the standard deviation is greater than 3. This is because we know that 99.7% of the data in a normal distribution is in the μ+3σ and μ-3σ region, hence the above condition holds true where the distance of the datapoint form mean divided by the standard deviation should be greater than 3 or ;less than -3. That means the datapoint is from that 0.3% of the data on both the tails. (0.15% on each)

```
# making a list in order to store the outlier values
outliers = []

# a for loop to check the above mentioned condition for outliers
for i in range(len(precipitation)):
  if((precipitation[i] - mean)/(sd) > 3 or (precipitation[i] -mean) < -3):
    outliers.append(precipitation[i])

# printing the calculated outliers
print('The outliers values of precipitaion from the abouve data are:-',outliers, '(in mm)')
```

```
The outliers values of precipitaion from the abouve data are:- [3.25] (in mm)
```

Hence we have found out the outlier from the above data using the statistical approach and it came out to be 3.25

QUESTION 4 :-

Loading the stocks dataset and calculating the percentage change in the daily closing price of each stock

Q4

```
from pandas import read_csv

# Loading the dataset
df = read_csv('stocks.csv', header=None)

# Extracting the values from the dataframe
data = df.values

# Split dataset into input and output elements
X, y = data[:, :-1], data[:, -1]

# Summarize the shape of the dataset
print(X.shape, y.shape)
```

```
(2519, 3) (2519,)
```

```
import numpy as np

N,d = stocks.shape
# Compute delta, which denotes the percentage of changes in the daily closing price of each stock
delta = pd.DataFrame(100*np.divide(stocks.iloc[1:,:].values-stocks.iloc[:N-1,:].values, stocks.iloc[:N-1,:].values),
                     columns=stocks.columns, index=stocks.iloc[1:].index)
delta.head()
```

```
from sklearn.svm import OneClassSVM

ee = OneClassSVM(nu=0.01,gamma='auto')
yhat = ee.fit_predict(delta) # Perform fit on input data and returns labels for that input data.

print(yhat) # Print labels: -1 for outliers and 1 for inliers.
```

```
[1 1 1 ... 1 1 1]
```

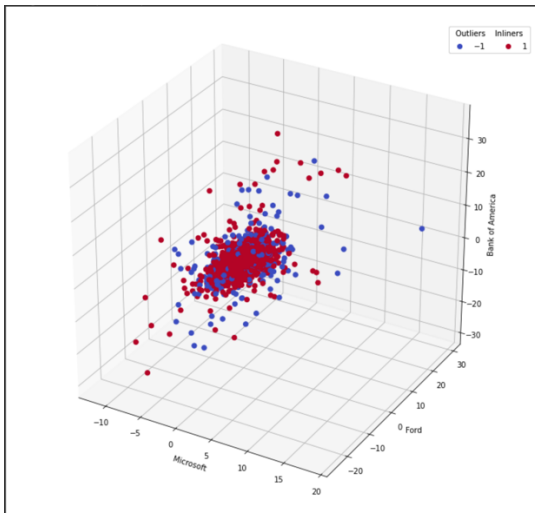
After fitting our data in the one class svm classifier, we separate out the outliers (in order to visualise outliers and inliners separately) using the code below. Thereafter we plot a 3-d scatter plot with all our columns according the color of the predicted values (yhat). This contains both the outliers and Inliners.

```
outliers_svm = []
for i in range(len(yhat)):
  if yhat[i] == -1:
    outliers_svm.append(yhat[i])
len(outliers_svm)
print(len(outliers_svm)/len(yhat)*100, '% of the data is consisted of outliers')
```

```
17.798967024235203 % of the data is consisted of outliers
```

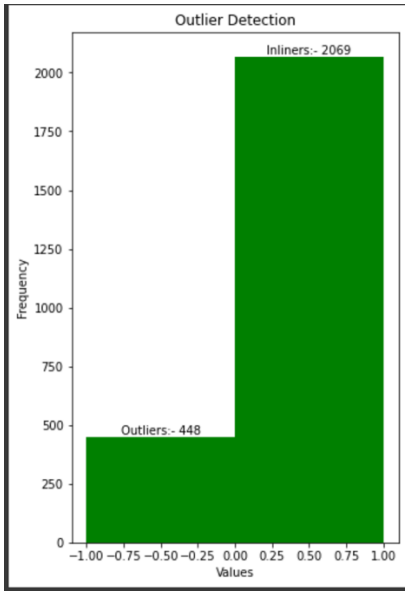
```
import re, seaborn as sns
import numpy as np

from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import ListedColormap

# setting the data for our 3 axis scatter plot that is the three companies
x = delta.MSFT
y = delta.F
z = delta.BAC

# creating axes instance
fig = plt.figure(figsize=(12,12))
ax = fig.add_subplot(111, projection='3d')
fig.add_axes(ax)

# using seaborn to get the colormap
cmap = ListedColormap(sns.color_palette("coolwarm", 256).as_hex())

# plotting the scatterplot with color map equal to our predicted data labels
sc = ax.scatter(x, y, z, s=40, c=outlier_score, marker='o', cmap=cmap, alpha=1,label='Outlier')
ax.set_xlabel('Microsoft')
ax.set_ylabel('Ford')
ax.set_zlabel('Bank of America')

#setting the legend for the scatter plot which helps us differentiate between the outliers and inliners
legend = ax.legend(*sc.legend_elements(), loc="upper right", title="Outliers    Inliners", bbox_to_anchor=(1.05, 1), borderaxespad=2, ncol=4)
ax.add_artist(legend)
```

The 3-D scatter plot of our data with colour separated data points for outliers and inliners.



*Using the SVM approach we calculated the number of outliers and inliners from our data. The number of outliers found out were:- **448** compared to **2069** inliners. The outliers constituted of 17.798 percent of our data. Thereafter we went on to plot a 3d scatter plot for outliers and inliners of the data using a binary colormap (blue and purple) where blue depicts the outliers(-1 value) and purple depicts the inliners(1 value). A histogram was plotted at the end for a better visualisation of the outliers.*

```
# Creating histogram
fig, ax = plt.subplots(1, 1,figsize=(5,8))
ax.hist(yhat, bins=2,color='green')

# Set title
ax.set_title("Outlier Detection")
# adding labels
ax.set_xlabel('Values')
ax.set_ylabel('Frequency')

# Labels
rects = ax.patches
labels = [f'Outliers:- {len(outliers_svm)}',f'Inliners:- {len(yhat)-len(outliers_svm)}']


for rect, label in zip(rects, labels):
    height = rect.get_height()
    ax.text(rect.get_x() + rect.get_width() / 2, height+0.01, label,
            ha='center', va='bottom')

# Show plot
plt.show()
```



Histogram for Outliers and Inliners

Conclusion:-

In terms of visualization when we compare the proximity, parametric based and the SVM approach, the SVM approach has a better readability, i.e instead of calculating the outlier distance(outlier score) it gives us a binary output (1,-1). Here the labels -1 are the outliers and 1s are the Inliners. We can actually differentiate between the number of outliers and inliners in our data if we use the svm algorithm. When it comes to the parametric approach, it works on the condition that the data is normally distributed unlike the one-class SVM based approach which can work on any data type. Hene the one class SVM approach is better than the parametric and proximity based approaches.

Question 5:-

Loading the housing dataset and performing z-score normalization before performing pca

```
[32] import pandas as pd
     import numpy as np
     from sklearn.decomposition import PCA
     import matplotlib.pyplot as plt
     from sklearn.preprocessing import StandardScaler
     import seaborn as sns


     from sklearn.preprocessing import StandardScaler
     data_house = pd.read_csv('https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv', header=None)

     from sklearn import preprocessing
     from scipy import stats

     #normalizing the data using z-score normalization
     scalar = StandardScaler()
     data_norm = scalar.fit_transform(data_house)
```

Dropping the last column from our dataset in order to apply PCA only to our input variables

```
data_norm = data_norm[:,:-1]
# peforming 2components pca analysis
pca = PCA(n_components=2)
finaldata = pca.fit_transform(data_norm)

# variance ratio for the two components
print(" variance ratio ", pca.explained_variance_ratio_)

pca_df = pd.DataFrame(finaldata,columns=['PC1','PC2'])
print(pca_df)

x = pca_df['PC1']
y = pca_df['PC2']
```

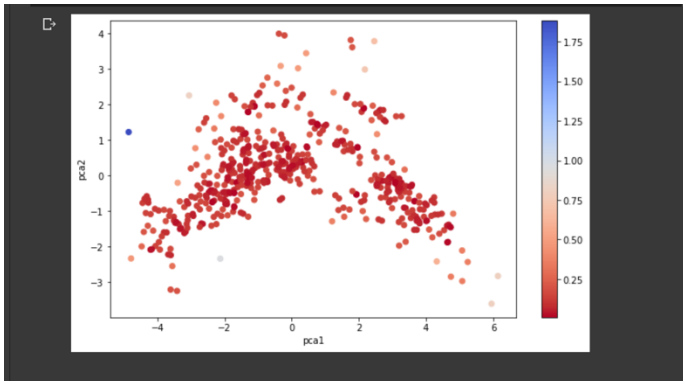
2 component PCA values

```
variance ratio  [0.47129606 0.11025193]
          PC1       PC2
0   -2.098297  0.773111
1   -1.457252  0.591986
2   -2.074598  0.599639
3   -2.611504 -0.006871
4   -2.458185  0.097712
..        ...       ...
501 -0.314968  0.724284
502 -0.110513  0.759307
503 -0.312360  1.155246
504 -0.270519  1.041361
505 -0.125803  0.761978

[506 rows x 2 columns]
```

Using the k-nearest neighbours (k=2) detecting outliers from our dataset. Thereafter plotting the pca values on a scatter plot and using the outlier score as the colormap.

```
from sklearn.neighbors import NearestNeighbors
import numpy as np
from scipy.spatial import distance

# Implement a k-nearest neighbour approach using k=4 neighbours
knn = 2
nbrs = NearestNeighbors(n_neighbors=knn, metric=distance.euclidean).fit(pca_df.values)
distances, indices = nbrs.kneighbors(pca_df.values)

# The outlier score is set as the distance between the point and its k-th nearest neighbour
outlier_score = distances[:,knn-1]

# plotting a scatter plot between both the pca columns using the colormap of outlier score.
fig = plt.figure(figsize=(10,6))
p = plt.scatter(x,y,c=outlier_score,cmap='coolwarm_r')
plt.xlabel('pca1')
plt.ylabel('pca2')
fig.colorbar(p)
plt.show()
```

The scatter plot shows the values of both the pca columns. The colour of the datapoints differ on the basis of the outlier score of datapoints.



Conclusion:-

We used the housing dataset and performed PCA for 2 components after z-score normalization of the data (only on the inputs). A scatter plot was used to visualize both the pca columns according to the colormapping of the outlier score. The greater the score (color turning from red to blue) the greater the distance of the datapoint from the k-nearest neighbour.