Q2)

```
import pandas as pd
import numpy as np
from mlxtend.frequent_patterns import apriori, fpmax, fpgrowth
from mlxtend.frequent_patterns import association_rules
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import association_rules

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']]

te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

frequent_itemsets = fpgrowth(df, min_support=0.7, use_colnames=True)

frequent_itemsets

def imbalance(df, rule):
   antecedents = rule
   consequents = rule
   antecedents = {'Eggs', 'Kidney Beans'}
   consequents = {'Milk', 'Onion'}
   x = len(df)
   lengthofa = len(df[df.apply(lambda x: antecedents.issubset(x), axis=1)]) #length of frozenset, number of
items
   lengthofc = len(df[df.apply(lambda x: consequents.issubset(x), axis=1)])
   newrule = len(df[df.apply(lambda x: antecedents.issubset(x) and consequents.issubset(x), axis=1)])
   return abs((lengthofa - lengthofc)) / (lengthofa + lengthofc - newrule)
df = pd.DataFrame(dataset)
rule = (['Onion', 'Eggs'], ['Kidney Beans'])
print(imbalance(df, rule))
```

I first started by importing the libraries and generated the frequent itemsets from using the fpgrowth function. In order to get the imbalance ratio for the itemsets and antecedants and consequents I first started by assigning them to a rule and then defining what the frozen set should be for both of them. In order to calculate the imbalance ratio I started by finding the lengths of the antecedents and consequents and then defined the formula for the imbalance ratio. I then implemented the dataframe and rule within the function.

Q1)

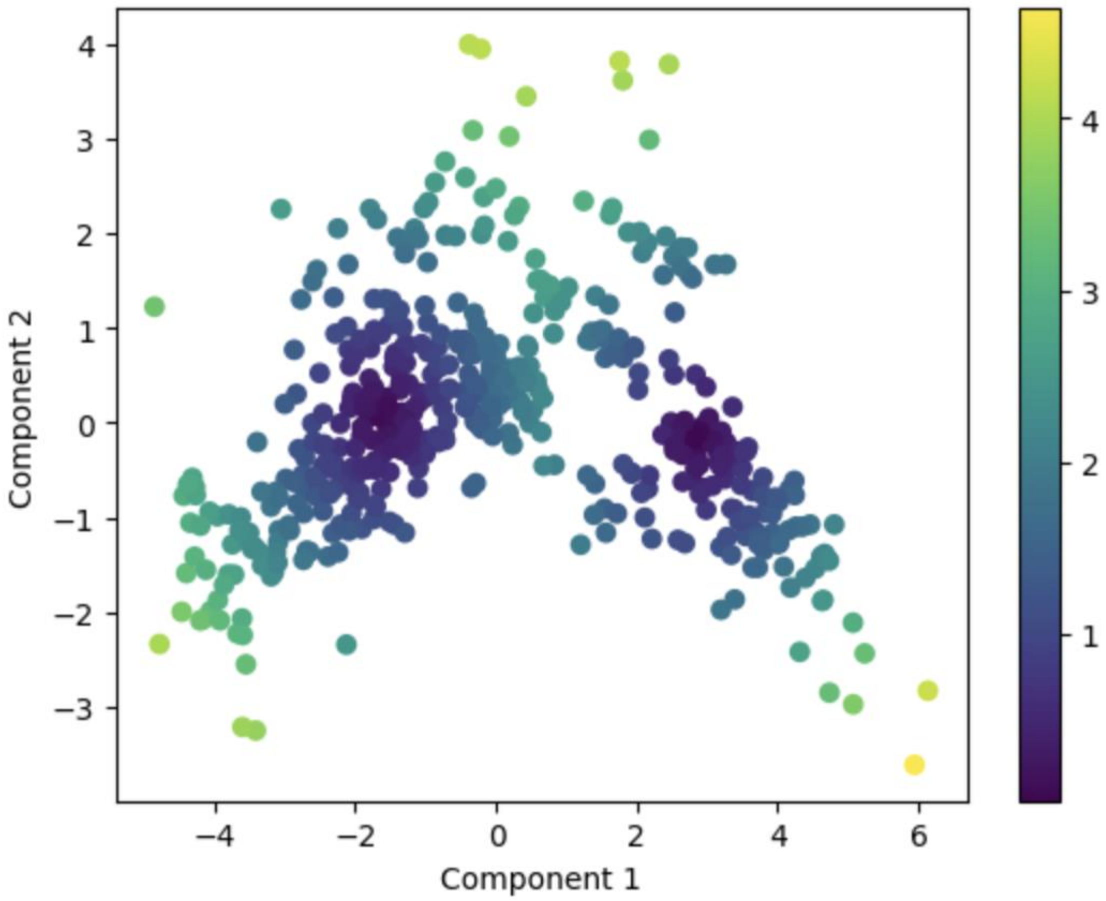
```
import pandas as pd
import numpy as np
from mlxtend.frequent_patterns import apriori, fpmax, fpgrowth
from mlxtend.frequent_patterns import association_rules
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import association_rules

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']]

te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

frequent_itemsets = fpgrowth(df, min_support=0.7, use_colnames=True)

frequent_itemsets

def kmeasure(df, rule):
   antecedents = rule
   consequents = rule
   antecedents = {'Eggs', 'Kidney Beans'}
   consequents = {'Milk', 'Onion'}
   x = len(df)
   lengthofa = len(df[df.apply(lambda x: antecedents.issubset(x), axis=1)]) #length of frozenset, number of
items
   lengthofc = len(df[df.apply(lambda x: consequents.issubset(x), axis=1)])
   newrule = len(df[df.apply(lambda x: antecedents.issubset(x) and consequents.issubset(x), axis=1)])
   return (newrule / lengthofa + newrule / lengthofc) / 2 #calculation

df = pd.DataFrame(dataset)
rule = (['Onion', 'Eggs'], ['Kidney Beans'])
print(kmeasure(df, rule))
```

I first started by importing the libraries and generated the frequent itemsets from using the fpgrowth function. In order to get the Kulczynski measure for the itemsets and antecedants and consequents I first started by assigning them to a rule and then defining what the frozen set should be for both of them. In order to calculate the Kulczynski measure I started by finding the lengths of the antecedents and consequents in order to get the confidence measures and then defined the formula for the Kulczynski measures. I then implemented the dataframe and rule within the function.

Q6)

```
from pandas import read_csv
from scipy.stats import zscore
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from numpy import sqrt, random, array, argsort, vstack
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import pandas as pd
from sklearn.neighbors import NearestNeighbors
import numpy as distance

url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
df = read_csv(url,header = None)

data = df.values

scalar = StandardScaler()
scaled_data = pd.DataFrame(scalar.fit_transform(df)) #scaling the data
scaled_data

pca = PCA(n_components = 2)
pca.fit(scaled_data)
data_pca = pca.transform(scaled_data)
data_pca = pd.DataFrame(data_pca,columns=['PC1','PC2'])
data_pca.head()

nbrs = NearestNeighbors(n_neighbors = 2)
nbrs.fit(data)

kmeans = KMeans(n_clusters = 2).fit(principalComponents)
center = kmeans.cluster_centers_
print(center)

center1 = sqrt(((principalComponents-centers[0])**2).sum(axis=1))
center2 = sqrt(((principalComponents-centers[1])**2).sum(axis=1))
distance = distance.hstack((center1.reshape(-1,1),center2.reshape(-1,1))).min(axis=1)

plt.scatter(principalComponents[:,0],principalComponents[:,1],c=distance)
plt.xlabel("Component 1")
plt.ylabel("Component 2")
plt.colorbar()
plt.show()
```

```
[[ 2.85277723 -0.15226879]
 [-1.61627936  0.08626993]]
```



I first started by importing all the necessary libraries and then loaded the dataset into the dataframe. To prepare the data further I converted the columns to arrays and then proceeded with the PCA. I standardised the data using the Standard Scalar library by creating an object of Standard Scalar and then putting it into the dataframe and then performed PCA. I initialised the components to 2 since we only want 2 principcal components in our final dataset. To perform outlier detection using k-nearest neighbours I first initialised the model so that it could fit the k-nearest algorithm with the relevant parameters where I set k=2 since we will be calculating the the distances later using 2 neighbours. I then used clustering to divide the data in order to find the outliers. By calculating the distance amongst the clusters and finding out which cluster is the farthest from the centroid would be displayed as the outlier. I created the centroids for 2 clusters and then computed the Euclidean distance for each and then used hstack() to stack it in an array. I then built the scatterplot to display the results.

Q7a)

For “ECS766P Data Mining - Week 10”, the <h1> tag is used as the heading

<h1>ECS766P Data Mining - Week 10</h1>

For “The below table contains income data per country; the same table was used for the Week 3 lab.” the paragraph tag <p> is being used

<p>The below table contains income data per country; the same table was used for the Week 3 lab.</p>

For “Region Age Income Online Shopper” the <thead> tag is being used as group headers for the following table, <table> defines a HTML table and <tr> defines a row in the table.

<table>

<thead>

<tr>

<th>Region</th>

<th>Age</th>

<th>Income</th>

<th>Online</th>

<th>Shopper</th>

</tr>

</thead>

India	49	86400	No
Brazil	32	57600	Yes
USA	35	64800	No
Brazil	43	73200	No
USA	45		Yes
India	40	69600	Yes
Brazil		62400	No
India	53	94800	Yes
USA	55	99600	No
India	42	80400	Yes

For the entries in the table the <tbody> tag is used to group the body in the table and it follows the same format as above.

```
<tbody>
   <tr>
     <td>India</td>
     <td>48</td>
     <td>86400</td>
     <td>No</td>
   </tr>
   <tr>
     <td>Brazil</td>
     <td>32</td>
    <td>57600</td>
    <td>Yes</td>
etc.
   </tr>
  </tbody>
```

Q7b)

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
url = 'http://eecs.qmul.ac.uk/~emmanouilb/income_table.html'
page = requests.get(url)
print(page.status_code)

soup = BeautifulSoup(page.text, 'lxml')
soup

table = soup.findAll('table',{"class":"table table-bordered table-hover table-condensed"})[0]
table

headers = []
for i in table.findAll('th'):
 title = i.text
 headers.append(title)
 print(table)

data = pd.DataFrame(columns = headers)

for j in table.find_all('tr')[1:]:
 row_data = j.find_all('td')
 row = [i.text for i in row_data]
 length = len(data)
 data.loc[length] = row
 print(table)

data.to_csv('data.csv', index=False)
data2 = pd.read_csv('data.csv')
```

	Region	Age	Income	Online Shopper
1	India	49	86400	No
2	Brazil	32	57600	Yes
3	USA	35	64800	No
4	Brazil	43	73200	No
5	USA	45		Yes
6	India	40	69600	Yes
7	Brazil		62400	No
8	India	53	94800	Yes
9	USA	55	99600	No
10	India	42	80400	Yes

I first imported all the relevant libraries and then created an URL object to fetch the page to request permission from the hosting site. I used ‘lxml’ to structure the format of the HTML page a bit better. In order to inspect every table element and to find specifically as to what we are looking for I used soup.find to find the table and passed in the location of exactly where it was. In order to get all the headers in the table I used a for loop to fill a new empty list with every column. I then proceeded to creating the dataframe and created a second for loop to fill in the rows and data and then created a new CSV file to make the final output more readable.

Q4)

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

Parisrainfall = np.array([20.95, 22.41, 25.21, 25.78, 28.43, 22.67, 24.55, 5.49, 23.11, 26.42, 23.90, 23.53])

mean = np.mean(Parisrainfall)
std = np.std(Parisrainfall)

zscores = (Parisrainfall - mean) / std
```

After importing the libraries I created an array for the rainfall dataset in order to calculate the mean, standard deviation and standard score. The range to calculate the area holding the data of the normal distribution would be the mean being 3 times more the standard deviation, the range was: 16.92 to 28.48 so the outlier is at 5.49.

Q3)

N^2 itemset in the worst case.

Q5)

```
from sklearn.svm import OneClassSVM
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D
from numpy import quantile, where, random
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import numpy as np
%matplotlib inline

# Loading the dataset
stocks = pd.read_csv('stocks.csv', header='infer')
stocks.index = stocks['Date']
stocks = stocks.drop(['Date'],axis=1)
stocks.head()

N,d = stocks.shape

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)

dataframe = delta.values

X, y = dataframe[:, :-1], data[:, -1]

print(X.shape, y.shape)
delta.head()
classifier = OneClassSVM(nu=0.01,gamma='auto')
label = classifier.fit_predict(X)

mask = label != -1
X, y = X[mask, :], y[mask]

print(X.shape, y.shape)

fig = plt.figure()
ax = plt.axes(projection ="3d")
graph = ax.scatter3D(delta.MSFT,delta.F,delta.BAC, c=yhat, cmap='inferno')
ax.set_xlabel('Microsoft')
ax.set_xlabel('Ford')
ax.set_xlabel('Bank of America')
fig.colorbar(graph)
plt.show()

plt.hist(label)
plt.xlabel('-1: Outliers, 1: Inliers ')
plt.ylabel('Frequency')
```

I started by importing the libraries and loading the dataset and then proceeded to computing the value for delta which signifies the percentage change of the closing price of each stock. I then went on to extracting the values from the dataframe and then splitting the dataset into x and y which are the input and output elements. I then selected the rows that were not the outliers and then configured a scatterplot to display the following results. 35.6% were outliers.

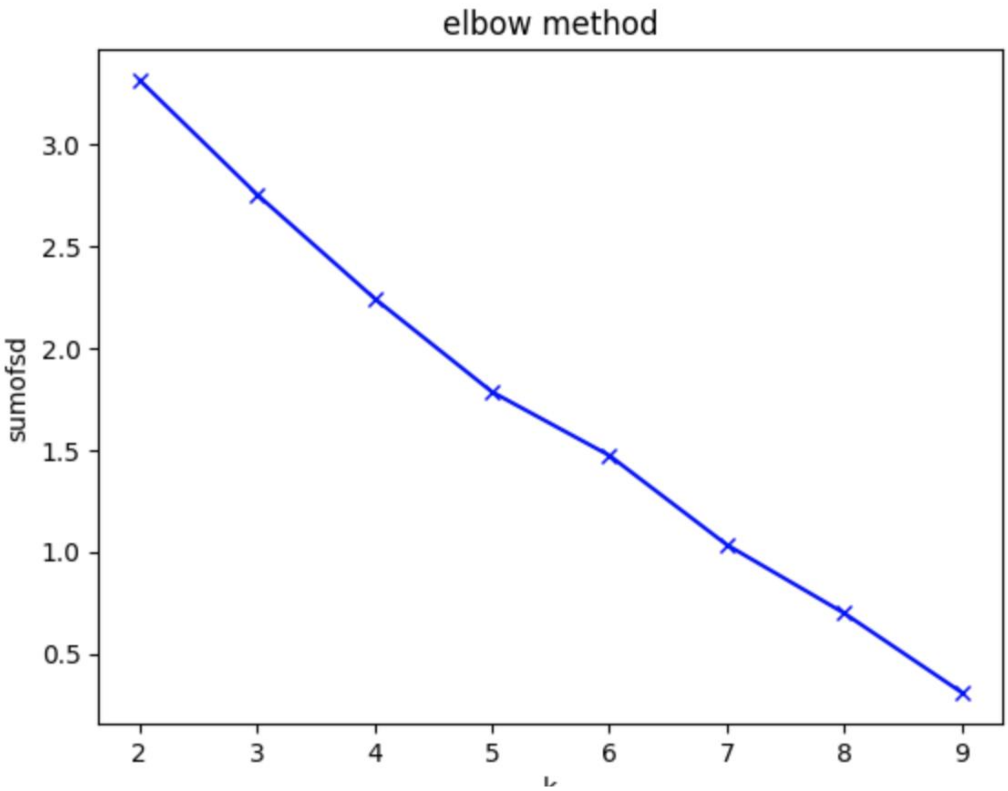
Q8)

```
import pandas as pd
import wikipedia
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

articles = ['Supervised Learning', 'Unsupervised Learning', 'Semi-Supervised Learning', 'Association Rule Learning', 'Anomaly Detection', 'Cluster Analysis', 'Dimensionality Reduction', 'Regression Analysis', 'Statistical Classification', 'Data Warehouse',  'Online Analytical Processing']
wikilist=[]
title=[]
```

```
for article in articles:
  wikilist.append(wikipedia.page(article).content)
  title.append(article)

vectorizer = TfidfVectorizer(stop_words={'english'})
X = vectorizer.fit_transform(wikilist)

sumofsd = []

K = range(2,10)

for k in K:
  km = KMeans(n_clusters=k, max_iter=200, n_init=10)
  km = km.fit(X)
  sumofsd.append(km.inertia_)

plt.plot(K, sumofsd, 'bx-')
plt.xlabel('k')
plt.ylabel('sumofsd')
plt.title('elbow method')
plt.show()

true_k = 6

model = KMeans(n_clusters=true_k, init='k-means++', max_iter=200, n_init=10)
model.fit(X)

labels=model.labels_

wiki_cl=pd.DataFrame(list(zip(title,labels)),columns=['title','cluster'])
print(wiki_cl.sort_values(by=['cluster']))


result={'cluster':labels,'wiki':wikilist}
result=pd.DataFrame(result)

for k in range(0,true_k):
  s=result[result.cluster==k]
  text=s['wiki'].str.cat(sep=' ')
  text=text.lower()
  text=' '.join([word for word in text.split()])
  wordcloud = WordCloud(max_font_size=50, max_words=100, background_color="white").generate(text)

  print('Cluster: {}'.format(k))
  print('Titles')

  titles=wiki_cl[wiki_cl.cluster==k]['title']

  print(titles.to_string(index=False))
```

```
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

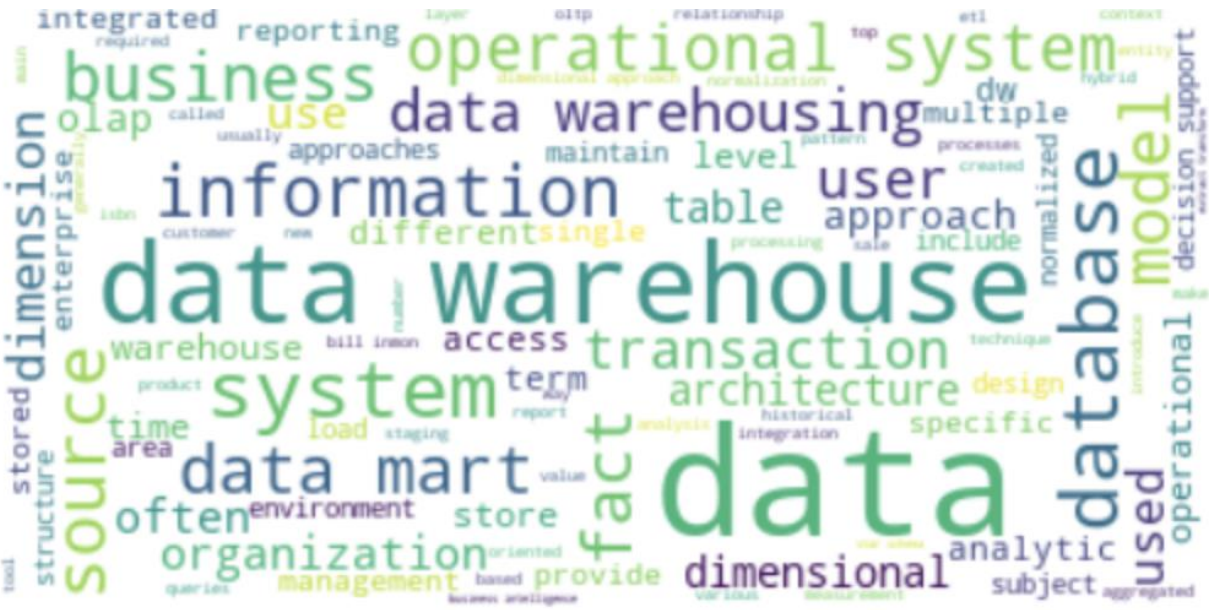
I started by importing the libraries and then fetching the Wikipedia articles. I made an array to store the Wikipedia articles and then used the elbow method to find out the number of clusters. I clustered them into groups of 6.



k

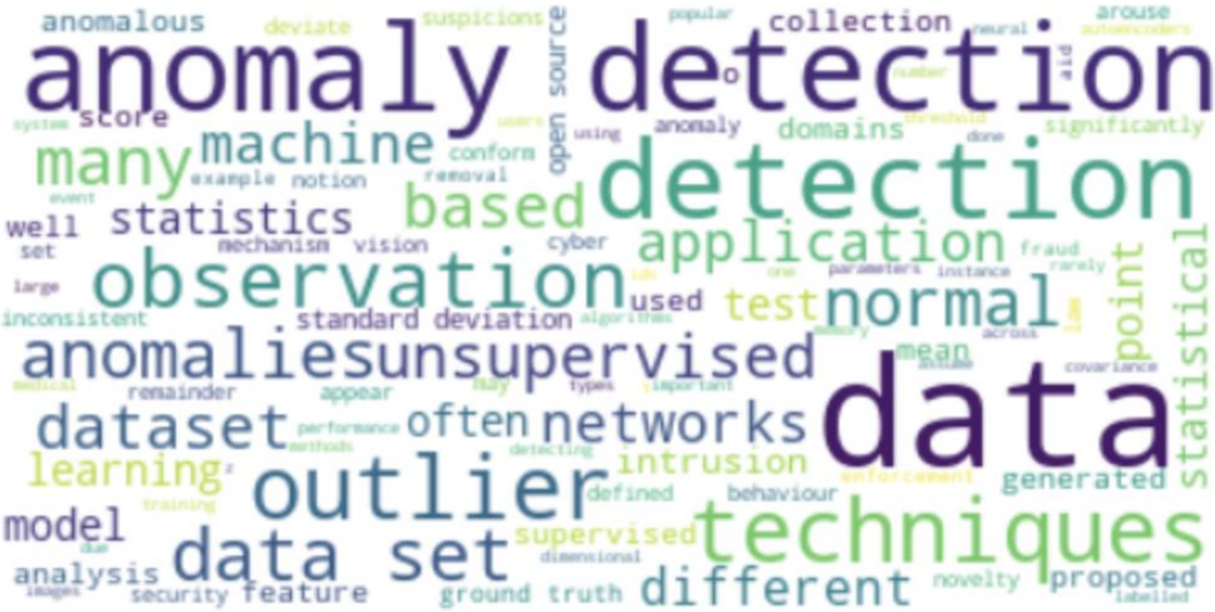
```
                               title  cluster
0            Supervised Learning        0
1          Unsupervised Learning        0
2       Semi-Supervised Learning        0
7            Regression Analysis        0
8      Statistical Classification       0
9                 Data Warehouse        1
3        Association Rule Learning      2
5               Cluster Analysis        2
4              Anomaly Detection        3
10  Online Analytical Processing        4
6       Dimensionality Reduction        5
Cluster: 0
Titles
       Supervised Learning
     Unsupervised Learning
  Semi-Supervised Learning
       Regression Analysis
Statistical Classification
```


```
Cluster: 1
Titles
Data Warehouse
```

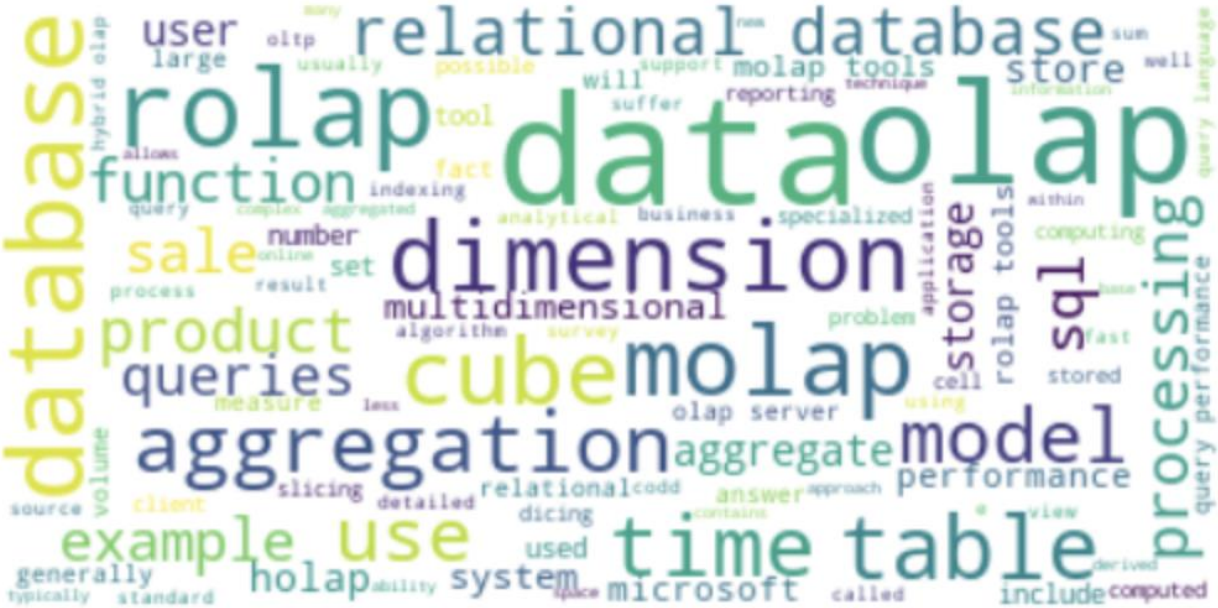


Cluster: 2
Titles
Association Rule Learning
          Cluster Analysis

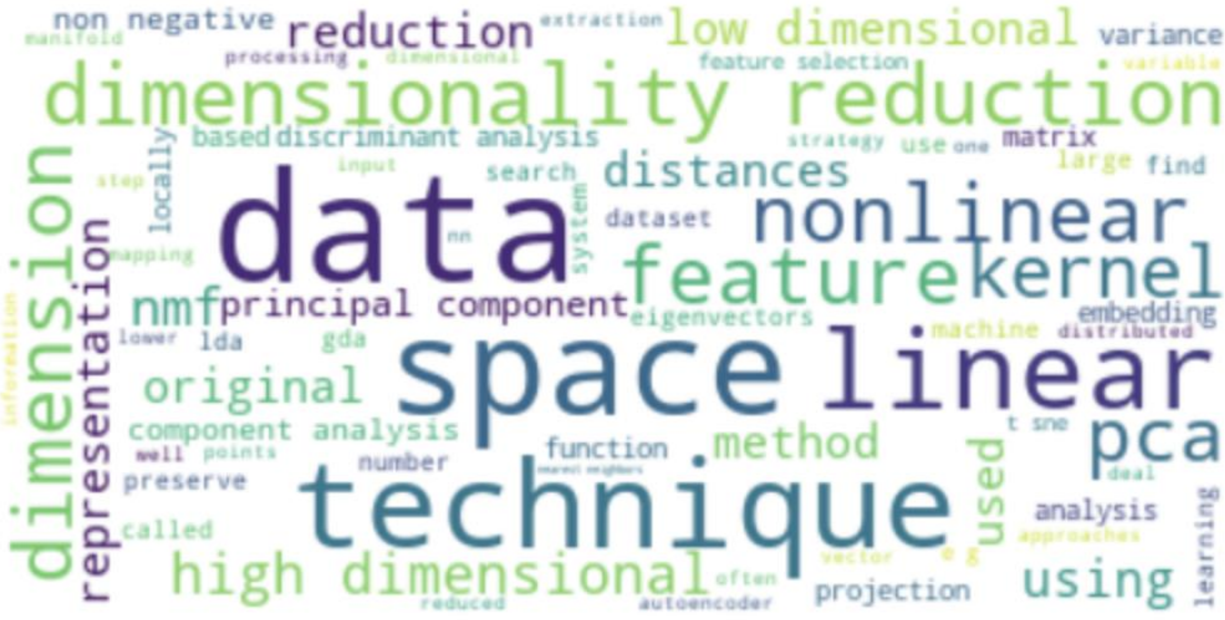
Cluster: 3
Titles
Anomaly Detection



Cluster: 4
Titles
Online Analytical Processing



Cluster: 5
Titles
Dimensionality Reduction


non negative
reduction
low dimensional
variance
dimensionality reduction
discriminant analysis
matrix
distances
data
nonlinear
feature
kernel
nmf
principal component
eigenvectors
embedding
dimension
representation
space
linear
original
component analysis
method
pca
technique
function
used
analysis
preserve
called
high dimensional
using
projection
locally
based
search
dataset
machine
distributed
lower
lda
gda
number
preserve
autoencoder
feature selection
extraction
processing
learning
large
find
mapping
t sne
deal
approaches
vector
often
reduced
well
points
system
strategy
use
one
input
step
manifold
information
variable