Assignment 3

Part 1

1. What is the advantage of using the Apriori algorithm in comparison with computing the support of every subset of an itemset in order to find the frequent itemsets in a transaction dataset?

"Support: It gives the fraction of transactions which contains item A and B. Basically Support tells us about the frequently bought items or the combination of items bought frequently."

The advantage of the Apriori Algorithm is that it is effective at eliminating candidate itemset without computing their support values.
If an itemset set has value less than minimum support then all of its supersets will also fall below min support, and thus can be ignored. This property is called the Antimonotone property.

This algorithm uses two steps “join” and “prune” to reduce the search space. It is an iterative approach to discover the most frequent itemsets.

Apriori algorithm works in this way:

The probability that item (A) is not frequent if:
P(A) < minimum support threshold, then A is not frequent.
P(A+B) < minimum support threshold, then A+B is not frequent, where B also belongs to itemset.

2. Let $\mathcal{L}_1$ denote the set of frequent 1-itemsets. For $k \geq 2$, why must every frequent k-itemset be a superset of an itemset in $\mathcal{L}_1$?

Apriori starts by finding the set of frequent 1-itemsets $\mathcal{L}_1$
Subsequently, it uses $\mathcal{L}_k - 1$ to find $\mathcal{L}_k$ for every k ≥ 2
Because the join step generates a set of candidates $\mathcal{L}_k$ from $\mathcal{L}_k - 1$.
Also Apriori algorithm guarentees that if an itemset is frequent, then all of its subsets must also be frequent.
All of above explain why must every frequent k-itemset be a superset of an itemset in $\mathcal{L}_1$

3. Let $\mathcal{L}_2 = \{\{1,2\},\{1,5\},\{2,3\},\{3,4\},\{3,5\}\}$. Compute the set of candidates C_3 that is obtained by joining every pair of joinable itemsets from $\mathcal{L}_2$.

$C_3 = \{\{1,2,3\},\{1,2,5\},\{1,3,5\},\{2,3,4\},\{2,3,5\},\{3,4,5\}\}.$

4. Let S_1 denote the support of the association rule $\{\text{popcorn, soda}\} \Rightarrow \{\text{movie}\}$. Let S_2 denote the support of the association rule $\{\text{popcorn}\} \Rightarrow \{\text{movie}\}$. What is the relationship between S_1 and S_2?

The support of the association rule S_1: $\{\text{popcorn, soda}\} \Rightarrow \{\text{movie}\}$. suggests that there is a strong relathioship between the sale of (popcorn, soda) and the sale of movies.

And the support of the association rule S_2: $\{\text{popcorn}\} \Rightarrow \{\text{movie}\}$. suggests that there is a strong relathioship between the sale of only (popcorn) and the sale of movies.

The two rules indicate that sale of popcorn will highly likely lead to the sale of a movie.

5. What is the support of the rule $\{\} \Rightarrow \{\text{Kidney Beans}\}$ in the transaction dataset used in the tutorial presented above?

$\{\text{Onion, Eggs}\} \Rightarrow \{\text{Kidney Beans}\}$

6. In the transaction dataset used in the tutorial presented above, what is the maximum length of a frequent itemset for a support threshold of 0.2?

In [159]:
```
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']]

from mlxtend.preprocessing import TransactionEncoder

te = TransactionEncoder()
te_ary = te.fit_transform(dataset)
#print(te_ary)

import pandas as pd

df = pd.DataFrame(te_ary, columns=te.columns_)
#display(df)

from mlxtend.frequent_patterns import apriori

frequent_itemsets = apriori(df, min_support=0.2, use_colnames=True)
#display(frequent_itemsets)


frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x
print("The maximum length of a frequent itemset for a support threshold of 0.2 is
print('\nFrequent 6-itemsets:')
display(frequent_itemsets[frequent_itemsets['length'] == 6])
```

```
The maximum length of a frequent itemset for a support threshold of 0.2 is:  6

Frequent 6-itemsets:
```

	support	itemsets	length
147	0.2	(Onion, Kidney Beans, Dill, Nutmeg, Yogurt, Eggs)	6
148	0.2	(Onion, Kidney Beans, Nutmeg, Yogurt, Eggs, Milk)	6

7. Implement a function that receives a `DataFrame` of frequent itemsets and a strong association rule (represented by a `frozenset` of antecedents and a `frozenset` of consequents). This function should returns the corresponding Kulczynski measure. Include the code in your report.

In [161]:
```
from mlxtend.frequent_patterns import association_rules
import numpy as np
import pandas as pd

strong_rules = association_rules(frequent_itemsets, metric="confidence", min_thre

strong_rules = strong_rules.drop(['lift' , 'leverage' , 'conviction'], axis=1)
```

In [162]:
```
strong_rules.insert(6, 'Kulczynski', np.nan)

display(strong_rules)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	Kulczynski
0	(Apple)	(Eggs)	0.2	0.8	0.2	1.0	NaN
1	(Apple)	(Kidney Beans)	0.2	1.0	0.2	1.0	NaN
2	(Apple)	(Milk)	0.2	0.6	0.2	1.0	NaN
3	(Ice cream)	(Corn)	0.2	0.4	0.2	1.0	NaN
4	(Corn)	(Kidney Beans)	0.4	1.0	0.4	1.0	NaN
...	...	...	...	...	...	...	...
689	(Nutmeg, Yogurt, Milk)	(Eggs, Onion, Kidney Beans)	0.2	0.6	0.2	1.0	NaN
690	(Nutmeg, Eggs, Milk)	(Yogurt, Onion, Kidney Beans)	0.2	0.4	0.2	1.0	NaN
691	(Eggs, Yogurt, Milk)	(Nutmeg, Onion, Kidney Beans)	0.2	0.4	0.2	1.0	NaN
692	(Onion, Milk)	(Eggs, Nutmeg, Yogurt, Kidney Beans)	0.2	0.4	0.2	1.0	NaN
693	(Nutmeg, Milk)	(Eggs, Yogurt, Onion, Kidney Beans)	0.2	0.4	0.2	1.0	NaN

694 rows × 7 columns

In [163]:

```
def cal_Kulczynski(strong_rules) :

    for i in range(len(strong_rules)) :

        antecedent_support = strong_rules.loc[i, "antecedent support"]

        consequent_support = strong_rules.loc[i, "consequent support"]

        support = strong_rules.loc[i, "support"]

        #display(antecedent_support)
        #display(consequent_support)
        #display(support)


        a = support / antecedent_support
        b = support / consequent_support
        c =1/2

        d = c*( a + b )

        strong_rules.at[i, "Kulczynski"] = d
```

In [164]:

```
cal_Kulczynski(strong_rules)
```

In [165]:
```
display(strong_rules)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	Kulczynski
0	(Apple)	(Eggs)	0.2	0.8	0.2	1.0	0.625000
1	(Apple)	(Kidney Beans)	0.2	1.0	0.2	1.0	0.600000
2	(Apple)	(Milk)	0.2	0.6	0.2	1.0	0.666667
3	(Ice cream)	(Corn)	0.2	0.4	0.2	1.0	0.750000
4	(Corn)	(Kidney Beans)	0.4	1.0	0.4	1.0	0.700000
...	...	...	...	...	...	...	...
689	(Nutmeg, Yogurt, Milk)	(Eggs, Onion, Kidney Beans)	0.2	0.6	0.2	1.0	0.666667
690	(Nutmeg, Eggs, Milk)	(Yogurt, Onion, Kidney Beans)	0.2	0.4	0.2	1.0	0.750000
691	(Eggs, Yogurt, Milk)	(Nutmeg, Onion, Kidney Beans)	0.2	0.4	0.2	1.0	0.750000
692	(Onion, Milk)	(Eggs, Nutmeg, Yogurt, Kidney Beans)	0.2	0.4	0.2	1.0	0.750000
693	(Nutmeg, Milk)	(Eggs, Yogurt, Onion, Kidney Beans)	0.2	0.4	0.2	1.0	0.750000

694 rows × 7 columns

8. Implement a function that receives a `DataFrame` of frequent itemsets and a strong association rule (represented by a `frozenset` of antecedents and a `frozenset` of consequents). This function should return the corresponding imbalance ratio. Include the code in your report.

In [166]:
```
strong_rules.insert(7, 'Imbalance Ratio', np.nan)

display(strong_rules)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	Kulczynski	Imbalan Rat
0	(Apple)	(Eggs)	0.2	0.8	0.2	1.0	0.625000	Na
1	(Apple)	(Kidney Beans)	0.2	1.0	0.2	1.0	0.600000	Na
2	(Apple)	(Milk)	0.2	0.6	0.2	1.0	0.666667	Na
3	(Ice cream)	(Corn)	0.2	0.4	0.2	1.0	0.750000	Na
4	(Corn)	(Kidney Beans)	0.4	1.0	0.4	1.0	0.700000	Na
...	...	...	...	...	...	...	...	
689	(Nutmeg, Yogurt, Milk)	(Eggs, Onion, Kidney Beans)	0.2	0.6	0.2	1.0	0.666667	Na
690	(Nutmeg, Eggs, Milk)	(Yogurt, Onion, Kidney Beans)	0.2	0.4	0.2	1.0	0.750000	Na
691	(Eggs, Yogurt, Milk)	(Nutmeg, Onion, Kidney Beans)	0.2	0.4	0.2	1.0	0.750000	Na
692	(Onion, Milk)	(Eggs, Nutmeg, Yogurt, Kidney Beans)	0.2	0.4	0.2	1.0	0.750000	Na
693	(Nutmeg, Milk)	(Eggs, Yogurt, Onion, Kidney Beans)	0.2	0.4	0.2	1.0	0.750000	Na

694 rows × 8 columns

In [167]:
```
def cal_Imbalance_Ratio(strong_rules):

    for i in range(len(strong_rules)) :

        antecedent_support = strong_rules.loc[i, "antecedent support"]

        consequent_support = strong_rules.loc[i, "consequent support"]

        support = strong_rules.loc[i, "support"]

        #display(antecedent_support)
        #display(consequent_support)
        #display(support)


        a = abs(antecedent_support - consequent_support)
        b = antecedent_support + consequent_support - support


        c = ( a / b )

        strong_rules.at[i, "Imbalance Ratio"] = c
```

In [168]:
```
cal_Imbalance_Ratio(strong_rules)
```

In [169]:
```
display(strong_rules)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	Kulczynski	Imbalan Rat
0	(Apple)	(Eggs)	0.2	0.8	0.2	1.0	0.625000	0.7500
1	(Apple)	(Kidney Beans)	0.2	1.0	0.2	1.0	0.600000	0.8000
2	(Apple)	(Milk)	0.2	0.6	0.2	1.0	0.666667	0.6666
3	(Ice cream)	(Corn)	0.2	0.4	0.2	1.0	0.750000	0.5000
4	(Corn)	(Kidney Beans)	0.4	1.0	0.4	1.0	0.700000	0.6000
...	...	...	...	...	...	...	...	
689	(Nutmeg, Yogurt, Milk)	(Eggs, Onion, Kidney Beans)	0.2	0.6	0.2	1.0	0.666667	0.6666
690	(Nutmeg, Eggs, Milk)	(Yogurt, Onion, Kidney Beans)	0.2	0.4	0.2	1.0	0.750000	0.5000
691	(Eggs, Yogurt, Milk)	(Nutmeg, Onion, Kidney Beans)	0.2	0.4	0.2	1.0	0.750000	0.5000
692	(Onion, Milk)	(Eggs, Nutmeg, Yogurt, Kidney Beans)	0.2	0.4	0.2	1.0	0.750000	0.5000
693	(Nutmeg, Milk)	(Eggs, Yogurt, Onion, Kidney Beans)	0.2	0.4	0.2	1.0	0.750000	0.5000

694 rows × 8 columns

Part 2

1. For an application on credit card fraud detection, we are interested in detecting contextual outliers. Suggest 2 possible contextual attributes and 2 possible behavioural attributes that could be used for this application, and explain why each of your suggested attribute should be considered as either contextual or behavioural.

credit card fraud detection:

Here we are detecting fraudulant transactions

Contextual attributes: (transaction location) and (transaction time).
These should be considered as contextual attributes because they can define the context of a transaction on a credit card. For example, in every transaction instance has a time and/or location attributes which define it.

Behavioral attributes: (transaction amount) and (number of transactions)
These should be considered as contextual because the (transaction amount) attribute can describe a transactions' amount in certain context (transaction location) or (transaction time) or both, as with (number of transactions) attribute it can describe the number of transactions in certain context (transaction location) or (transaction time) or both.

The anomalous behavior is determined using the values for the behavioral attributes within a specific context.

2. Assume that you are provided with the [University of Wisconsin breast cancer dataset (https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data)](https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data) from the Week 3 lab, and that you are asked to detect outliers from this dataset. Additional information on the dataset attributes can be found [online (https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.names)](https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.names). Explain one possible outlier detection method that you could apply for detecting outliers for this particular dataset, explain what is defined as an outlier for your suggested approach given this particular dataset, and justify why would you choose this particular method for outlier detection.

Wisconsin breast cancer dataset has two labeled classes (2 for benign, 4 for malignant)
Class distribution:

Benign: 458 (65.5%)
Malignant: 241 (34.5%)

There is class imbalance here.

I will be using a Density-based outlier detection methode.
A datapoint that does not belong to one of the two cluster is considerd as an outlier. Meaning it has a far less density than the datapoints in cluster

In this scenario objects labeled as outlier or normal are not available. Therefore, an unsupervised method can be used. This type of outlier detection method makes an assumption that normal objects are clustered.

We use this Density-based approach because this method investigates the density of a datapoint and that of its neighbors. Here, a datapoint is identified as an outlier if its density is relatively much lower than that of its neighbors in its cluster.

The idea of density-based is that we need to compare the density around an object with the density around its local neighbors. The basic assumption of density-based outlier detection methods is that the density around a nonoutlier object is similar to the density around its neighbors, while the density around an outlier object is significantly different from the density around its neighbors.

3. The monthly rainfall in the London borough of Tower Hamlets in 2018 had the following amount of precipitation (measured in mm, values from January-December 2018): {22.93, 20.59, 25.65, 23.74, 25.24, 4.55, 23.45, 28.18, 23.52, 22.32, 26.73, 23.42}. Assuming that the data is based on a normal distribution, identify outlier values in the above dataset using the maximum likelihood method.

In [109]:
```
from statistics import mean
import numpy as np

dataset = [22.93, 20.59, 25.65, 23.74, 25.24, 4.55, 23.45, 28.18, 23.52, 22.32, 2

m = mean(dataset)
x = np.std(dataset)

m = round(m, 2)
std = round(x, 2)

print("mean: ", m )
print("standard deviation: ", std )
```

```
mean:  22.53
standard deviation:  5.76
```

In [110]:
```
dataset
```

Out[110]:
```
[22.93,
 20.59,
 25.65,
 23.74,
 25.24,
 4.55,
 23.45,
 28.18,
 23.52,
 22.32,
 26.73,
 23.42]
```

99.7% of data will fall within three standard deviations from the mean. This means there is a 99.7% probability of randomly selecting a score between -3 and +3 standard deviations from the mean.

In [111]:
```
length = len(dataset)

outputString = "Outliers: "

for i in range(length):

    a = round(abs( dataset[i] - m ) , 2)
    b = a/std
    b = round(b, 2)

    if b > 3:
        outputString += str(dataset[i])
        print(outputString)
```

```
Outliers: 4.55
```

4. You are provided with the graduation rate dataset used in the Week 4 lab (file graduation_rate.csv in the Week 4 lab supplementary data). For the 'high school gpa' attribute, compute the relative frequency (i.e. frequency normalised by the size of the dataset) of each value. Show these computed relative frequencies in your report. Two new data points are included in the dataset, one with a 'high school gpa' value of 3.6, and one with a 'high school gpa' value of 2.8. Using the above computed relative frequencies, which of the two new data points would you consider as an outlier and why?

In [136]:
```
import pandas as pd

df = pd.read_csv('graduation_rate.csv')

print('Dataset (head and tail):')
display(df)
```

Dataset (head and tail):

	ACT composite score	SAT total score	parental level of education	parental income	high school gpa	college gpa	years to graduate
0	30	2206	master's degree	94873	4.0	3.8	3
1	26	1953	some college	42767	3.6	2.7	9
2	28	2115	some high school	46316	4.0	3.3	5
3	33	2110	some high school	52370	4.0	3.5	4
4	30	2168	bachelor's degree	92665	4.0	3.6	4
...	...	...	...	...	...	...	...
995	30	1967	high school	49002	3.8	3.5	6
996	28	2066	some college	83438	3.9	3.5	4
997	27	1971	high school	68577	3.6	3.7	5
998	30	2057	some college	56876	3.8	3.6	3
999	29	2054	some high school	40068	3.9	3.3	5

1000 rows × 7 columns

In [137]:
```
df = df.sort_values(by='high school gpa', ascending=False)
```

In [138]:
```
abs_frequency = df['high school gpa'].value_counts()
display(abs_frequency)
```

```
4.0    294
3.8    132
3.9    108
3.7    106
3.6    101
3.5     71
3.4     64
3.3     52
3.2     25
3.0     20
3.1     15
2.9      8
2.7      3
2.8      1
Name: high school gpa, dtype: int64
```

In [139]:
```
print("Relative Frequency of high school gpa:")

relative_frequency = df['high school gpa'].value_counts()/len(df)
display(relative_frequency)

rel = relative_frequency.tolist()
rel1 = relative_frequency.tolist()
rel2 = relative_frequency.tolist()
```

```
Relative Frequency of high school gpa:

4.0    0.294
3.8    0.132
3.9    0.108
3.7    0.106
3.6    0.101
3.5    0.071
3.4    0.064
3.3    0.052
3.2    0.025
3.0    0.020
3.1    0.015
2.9    0.008
2.7    0.003
2.8    0.001
Name: high school gpa, dtype: float64
```

In [140]:
```
### Mean and Standard deviation with the two new data points included

m = mean(rel)
x = np.std(rel)

m = round(m, 2)
std = round(x, 2)

print("mean: ", m )
print("standard deviation: ", std )
```

```
mean:  0.07
standard deviation:  0.07
```

In [141]:
```
### Mean and Standard deviation without the two new data point included

rel.pop(13)
rel[4]=0.100

m = mean(rel)
x = np.std(rel)

m = round(m, 2)
std = round(x, 2)

print("mean: ", m )
print("standard deviation: ", std )
```

```
mean:  0.08
standard deviation:  0.08
```

In [142]:
```
### Mean and Standard deviation with only the 3.6 data point included

rel1.pop(13)

m = mean(rel1)
x = np.std(rel1)

m = round(m, 2)
std = round(x, 2)

print("mean: ", m )
print("standard deviation: ", std )
```

```
mean:  0.08
standard deviation:  0.08
```

In [143]:
```
### Mean and Standard deviation with only the 2.8 data point included

rel2[4]=0.100

m = mean(rel2)
x = np.std(rel2)

m = round(m, 2)
std = round(x, 2)

print("mean: ", m )
print("standard deviation: ", std )
```

```
mean:  0.07
standard deviation:  0.07
```

From these results I would consider the new 2.8 data point as an outlier because it affected the mean after it was added

5. Using the stock prices dataset used in sections 1 and 2, estimate the outliers in the dataset using the one-class SVM classifier approach. As input to the classifier, use the percentage of changes in the daily closing price of each stock, as was done in section 1 of

the notebook.

Plot a 3D scatterplot of the dataset, where each object is color-coded according to whether it is an outlier or an inlier. Also compute a histogram and the frequencies of the estimated outlier and inlier labels. In terms of the plotted results, how does the one-class SVM approach for outlier detection differ from the parametric and proximity-based methods used in the lab notebook? What percentage of the dataset objects are classified as outliers?

In [561]:
```
import pandas as pd

# Load CSV file, set the 'Date' values as the index of each row, and display the
stocks = pd.read_csv('stocks.csv', header='infer')
stocks.index = stocks['Date']
stocks = stocks.drop(['Date'],axis=1)
stocks.head()
```

Out[561]:

Date	MSFT	F	BAC
1/3/2007	29.860001	7.51	53.330002
1/4/2007	29.809999	7.70	53.669998
1/5/2007	29.639999	7.62	53.240002
1/8/2007	29.930000	7.73	53.450001
1/9/2007	29.959999	7.79	53.500000

In [562]:
```
import numpy as np

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

Out[562]:

Date	MSFT	F	BAC
1/4/2007	-0.167455	2.529960	0.637532
1/5/2007	-0.570278	-1.038961	-0.801185
1/8/2007	0.978411	1.443570	0.394438
1/9/2007	0.100231	0.776197	0.093543
1/10/2007	-1.001332	-0.770218	0.149536

In [563]:
```
# Extracting the values from the dataframe
data = delta.values

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

# Summarize the shape of the dataset
print(data.shape)
```

```
(2517, 3)
```

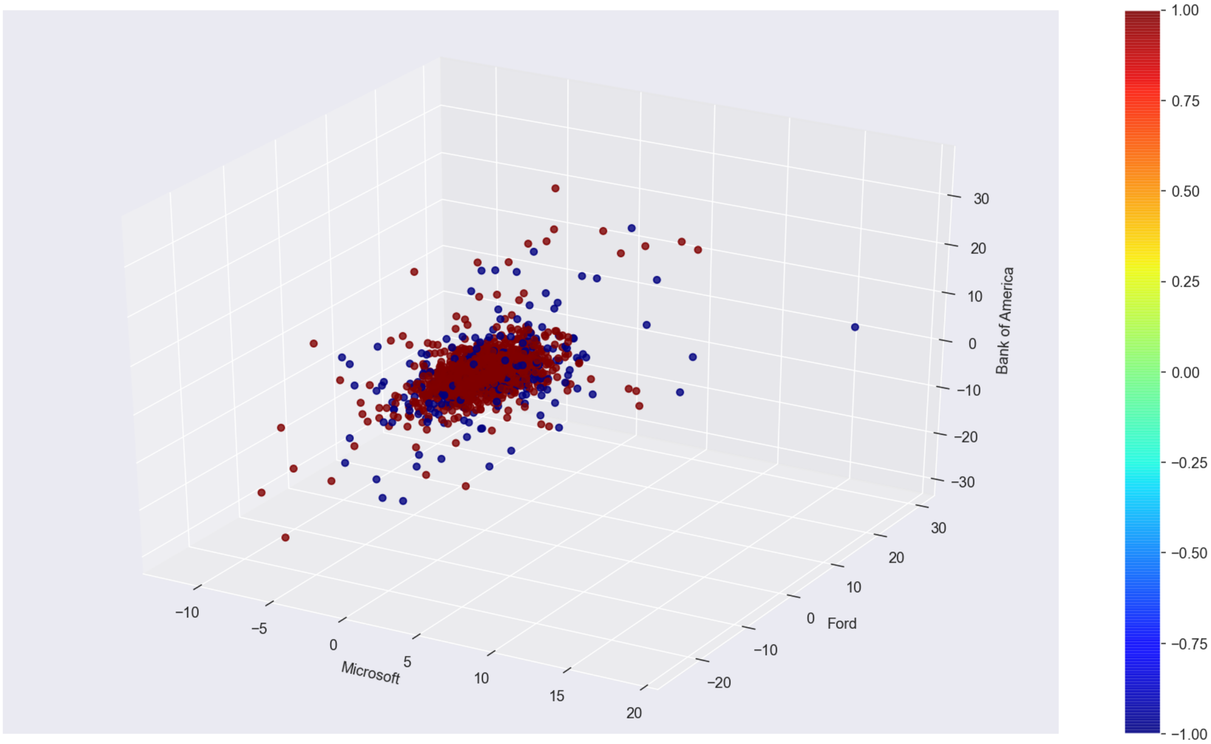
In [564]:
```
from sklearn.svm import OneClassSVM

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

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

```
[1 1 1 ... 1 1 1]
(2517,)
```

In [565]:
```
# Select all rows that are not outliers
mask = pred != 1
#X, y = X[mask, :], y[mask]

s = data[mask]

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

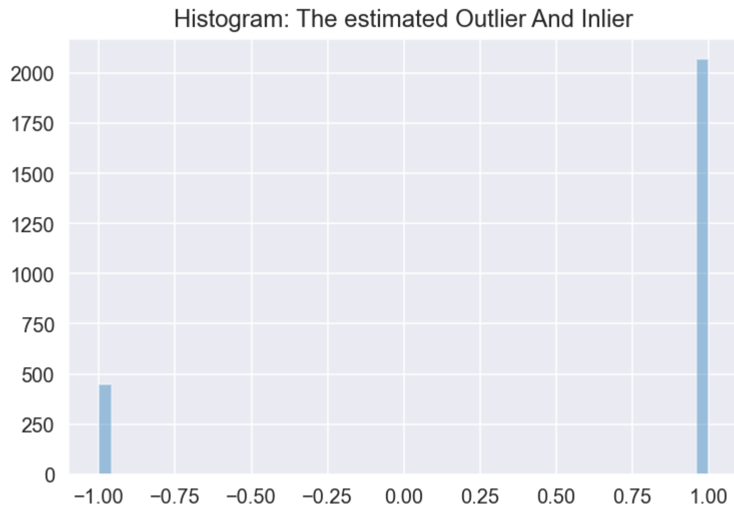
```
(448, 3)
```

In [566]:
```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
%matplotlib inline

# Plot 3D scatterplot of outlier scores
fig = plt.figure(figsize = (16, 9))
ax = fig.add_subplot(111, projection='3d')
ax.grid(b = True, color ='grey', linestyle ='-.', linewidth = 0.3, alpha = 0.2)
p = ax.scatter3D(delta.MSFT, delta.F, delta.BAC, alpha = 0.8, c = pred, cmap = 'j
ax.set_xlabel('Microsoft')
ax.set_ylabel('Ford')
ax.set_zlabel('Bank of America')
fig.colorbar(p)
plt.show()
```



In [567]:
```
%config InlineBackend.figure_formats = set(['retina'])

import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style('darkgrid')
```

In [568]:
```
sns.distplot(pred, bins=None, kde=False)
plt.title('Histogram: The estimated Outlier And Inlier')
plt.show()
```



In [570]:
```
print("Absolute Frequency of Outliers And Inliers: \n")

unique, counts = np.unique(pred, return_counts=True)
dict(zip(unique, counts))
```

```
Absolute Frequency of Outliers And Inliers:
```

Out[570]:
```
{-1: 448, 1: 2069}
```

In [571]:
```
print("Relative Frequency of Outlier And Inlier: \n")

l = len(pred)

length = len(counts)

for i in range(length):

    a = round( counts[i]/l , 2)
    b = unique[i]

    print( b," :", str(a))
```

```
Relative Frequency of Outlier And Inlier:

-1  : 0.18
1  : 0.82
```

As seen from the plots the one-class SVM approach for outlier detection differs from the parametric and proximity-based methods because the one-class SVM approach gives an outright decision on which datapoint is an outlier or inliers.

Whereas the parametric and proximity-based methods usee types of distances between datapoints to detect outliers, however it is still ambigious where the line between outlier or inliers lays and its upto to the user to decide.

percentage of the dataset objects are classified as outliers is: 18%

6. This question will combine concepts from both data preprocessing and outlier detection. Using the house prices dataset from Section 3 of this lab notebook, perform dimensionality reduction on the dataset using PCA with 2 principal components (make sure that the dataset is z-score normalised beforehand, and remember that PCA should only be applied on the input attributes). Then, perform outlier detection on the pre-processed dataset using the k-nearest neighbours approach using k=2. Display a scatterplot of the two principal components, where each object is colour-coded according to the computed outlier score.

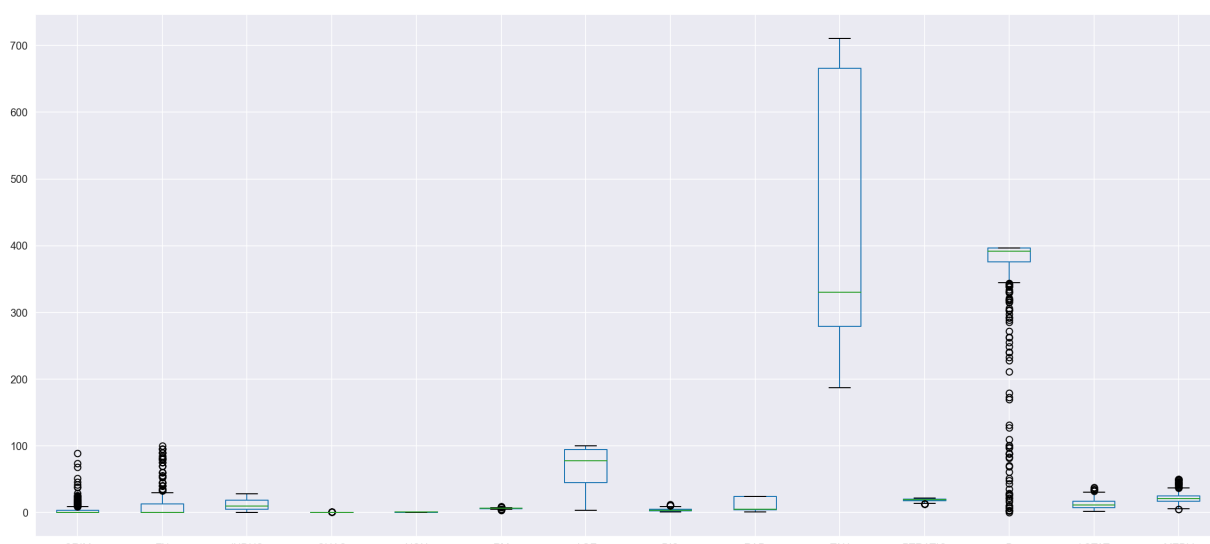
In [572]:
```
from pandas import read_csv

# Loading the dataset

df = pd.read_csv('housing.csv', header=None)

df.columns = [ 'CRIM','ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RAD','TAX','PTF

#df.head()

display(df)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST/
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99	9.
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90	9.
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90	5.
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45	6.
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90	7.

506 rows × 14 columns

In [573]:
```
df.boxplot(figsize=(20,9))
```

Out[573]: <matplotlib.axes._subplots.AxesSubplot at 0x23941d402b0>



In [574]:
```
import pandas as pd
import numpy as np
import scipy.stats as stats
```

In [575]:
```
z_scores = stats.zscore(df, axis=0)
```

In [576]:
```
abs_z_scores = np.abs(z_scores)
```

In [577]:
```
filtered_entries = (abs_z_scores < 3).all(axis=1)
```

In [578]:
```
nomalized_df = df[filtered_entries]
```

In [579]:
```
nomalized_df =  nomalized_df
```

In [580]:
```
print('Number of rows before z-score normalisation = %d' % (df.shape[0]))

print('Number of rows after z-score normalisation = %d' % (nomalized_df.shape[0])
```

```
Number of rows before z-score normalisation = 506
Number of rows after z-score normalisation = 415
```

In [581]:
```
display(nomalized_df)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99	9.
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90	9.
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90	5.
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45	6.
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90	7.

415 rows × 14 columns

In [582]:
```
nomalized_df = nomalized_df.reset_index()
```

In [583]:
```
nomalized_df = nomalized_df.drop(columns=['index'])
```

In [584]:
```
nomalized_df
```

Out[584]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST/
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.
...	...	...	...	...	...	...	...	...	...	...	...	...	
410	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273.0	21.0	391.99	9.
411	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273.0	21.0	396.90	9.
412	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273.0	21.0	396.90	5.
413	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273.0	21.0	393.45	6.
414	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273.0	21.0	396.90	7.

415 rows × 14 columns

In [585]:
```
features = list(nomalized_df.columns)
features.remove('MEDV')
features
```

Out[585]:
```
['CRIM',
 'ZN',
 'INDUS',
 'CHAS',
 'NOX',
 'RM',
 'AGE',
 'DIS',
 'RAD',
 'TAX',
 'PTRATIO',
 'B',
 'LSTAT']
```

In [586]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Separating out the features
x = nomalized_df.loc[:, features].values
# Separating out the target
y = nomalized_df.loc[:,['MEDV']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)
```

In [587]:
```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents , columns = ['principal com
```

In [588]:
```
finalDf = pd.concat([principalDf, nomalized_df[['MEDV']]], axis = 1)
finalDf
```

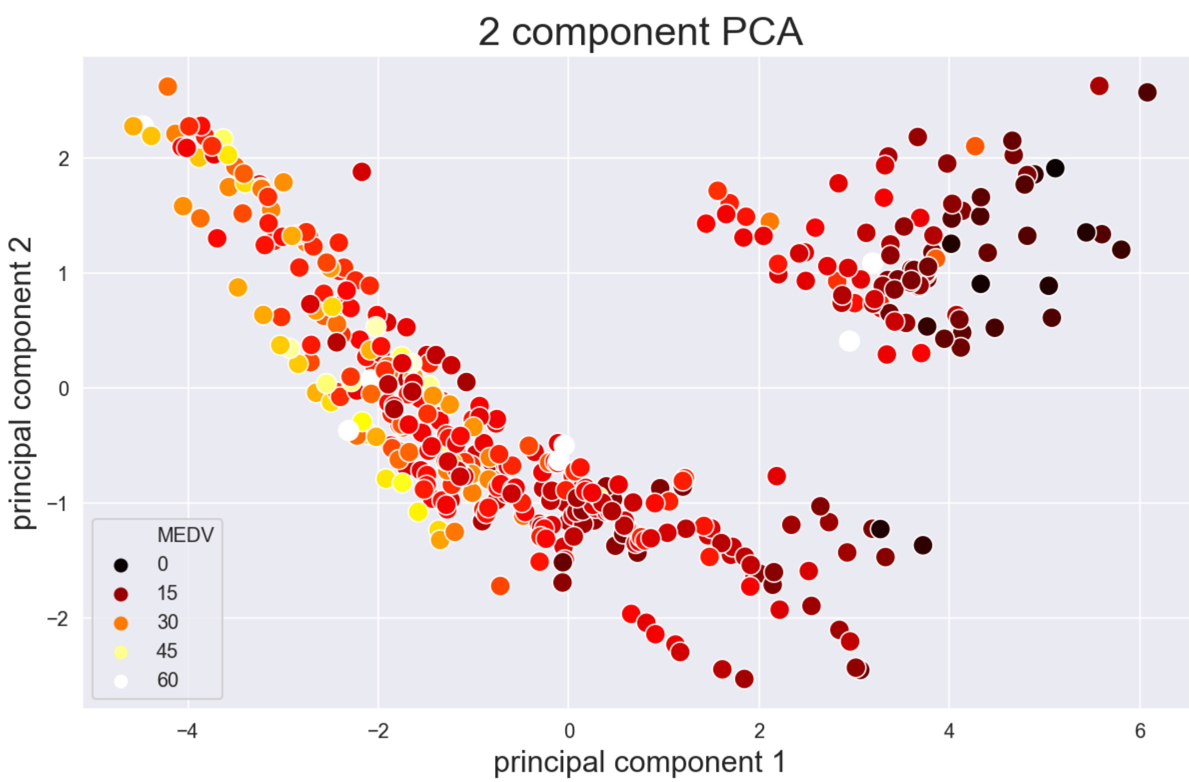
Out[588]:

	principal component 1	principal component 2	MEDV
0	-2.085682	-0.402316	24.0
1	-1.434076	-0.962625	21.6
2	-2.101529	-0.405423	34.7
3	-2.653843	-0.040594	33.4
4	-2.497059	-0.122638	36.2
...	...	...	...
410	-0.257429	-1.223892	22.4
411	-0.041436	-1.496261	20.6
412	-0.296152	-1.292951	23.9
413	-0.239364	-1.311929	22.0
414	-0.061760	-1.516900	11.9

415 rows × 3 columns

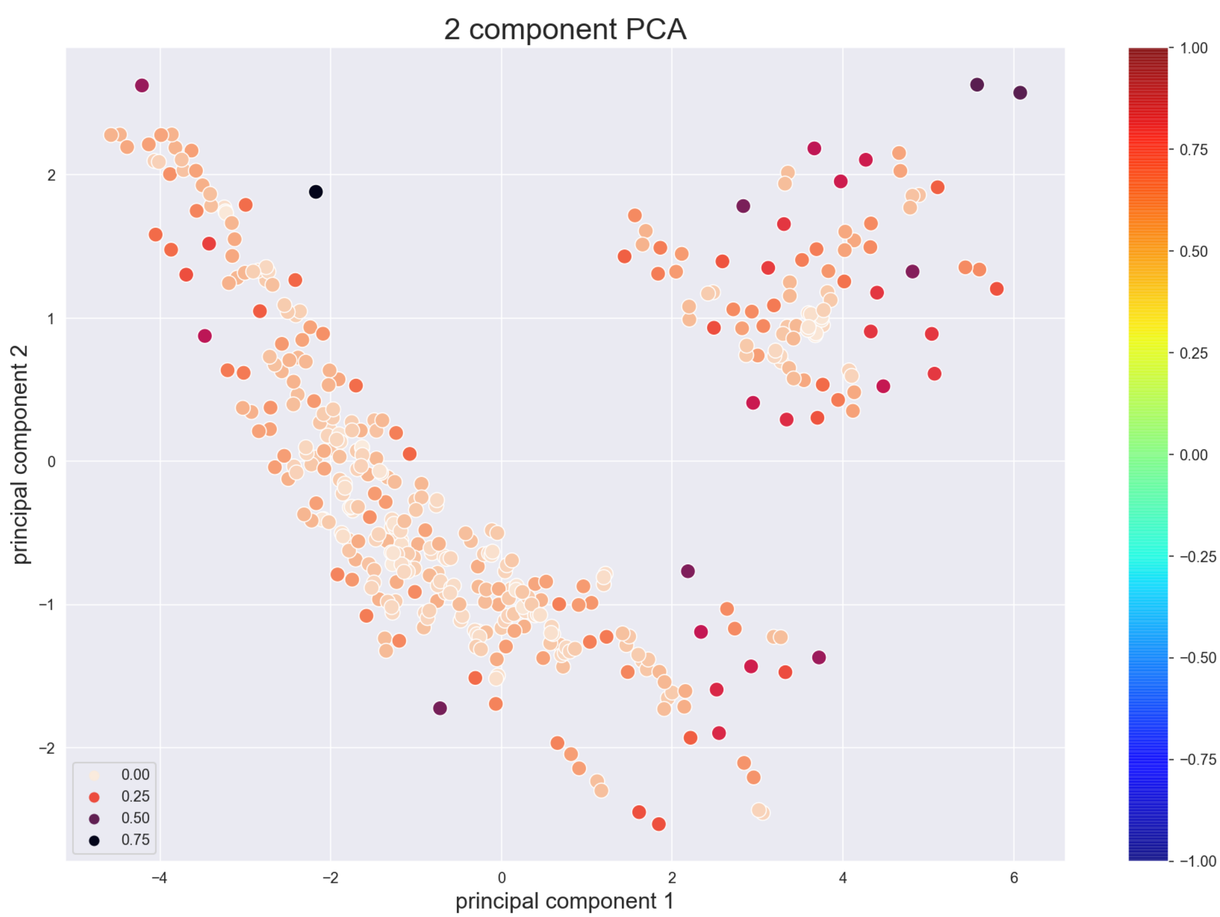
In [590]:
```
fig = plt.figure(figsize = (10,6))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)

class_list = nomalized_df['MEDV'].tolist()
myset = set(class_list)
targets = list(myset)

sns.scatterplot(data = finalDf ,x = 'principal component 1', y = 'principal compo
```

Out[590]: `<matplotlib.axes._subplots.AxesSubplot at 0x2393cdf3280>`



In [591]:
```
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(principal
distances, indices = nbrs.kneighbors(principalDf.values)

# The outlier score is set as the distance between the point and its k-th nearest


outlier_score = distances[:,knn-1]
```

In [594]:
```
fig = plt.figure(figsize = (15,10))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
fig.colorbar(p)

sns.scatterplot(data = principalDf ,x = 'principal component 1', y = 'principal c


plt.show()
```



In [ ]: