Name: Nivesara Tirupati

Student ID:230747311

Course: Msc in FT Big Data Science

email:ec23912@qmul.ac.uk

Question 1(a)

[pen&paper] - 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?

Explaination(a):

Support: It provides the percentage of transactions that contain items A and B. In essence, support informs us of the items or combinations of items that are regularly purchased."

The Apriori Algorithm's benefit is its ability to effectively eliminate candidate itemsets without calculating their support values. All of an itemset set's supersets will likewise fall below minimum support if the itemset set's value is less than minimum support, so these can be disregarded. The Antimonotone property is the name given to this characteristic.

To narrow down the search space, this algorithm employs the "join" and "prune" steps. Finding the most common itemsets involves an iterative process.

The way the Apriori algorithm operates is as follows:

The likelihood that item (A) is not frequently occurring if P(A) is less than the minimum support threshold, therefore A is not frequent. If B is also a member of the itemset and P(A+B) is less than the minimum support threshold, then A+B is not frequent.

Question 1(b)

[pen&paper] - Let L_1 denote the set of frequent 1-itemsets. For k ≥2 why must every frequent k-itemset be a superset of an itemset in L_1?

Explaination(b)

The Apriori algorithm is a classic algorithm for mining frequent itemsets and generating association rules in a dataset. It follows the Apriori property, which states that if an itemset is frequent, then all of its subsets must also be frequent. The algorithm uses this property to efficiently prune the search space and reduce the number of candidate itemsets that need to be checked for frequency.

Let's break down the key steps of the Apriori algorithm and explain why each frequent (k)-itemset needs to be a superset of an itemset in (\mathcal{L}_1).

1. **Generate Frequent 1-Itemsets (\mathcal{L}_1):**

- In the first pass through the dataset, the algorithm counts the occurrences of each individual item (1-itemset).
- The frequent 1-itemsets (\mathcal{L}_1) are identified based on a user-specified minimum support threshold.

2. **Generate Frequent (k)-Itemsets (\mathcal{L}*k) from (|*mathcal{L}*{k-1}):**

   - For (k \geq 2), the algorithm generates candidate (k)-itemsets using (\mathcal{L}_{k-1}) (frequent ((k-1))-itemsets).
   - The join step combines (k-1)-itemsets to form new candidate (k)-itemsets.
   - The prune step removes any candidate (k)-itemsets that contain infrequent ((k-1))-itemsets.

3. **Apriori Property:**

   - The Apriori property is based on the observation that if an itemset is frequent, all of its subsets must also be frequent.
   - This property allows the algorithm to avoid counting the support of all possible candidate itemsets and focus on a reduced set of candidates.

4. **Superset of (\mathcal{L}_1):**

   - In the process of generating frequent (k)-itemsets, the algorithm ensures that each frequent (k)-itemset is a superset of an itemset in (\mathcal{L}_1).
   - This follows from the Apriori property: if an itemset is frequent, all of its subsets (including those in (\mathcal{L}_1)) must also be frequent.

In summary, the Apriori algorithm efficiently prunes the search space by exploiting the Apriori property. It focuses on generating candidate itemsets that have a high likelihood of being frequent, reducing the computational cost of finding frequent itemsets in large datasets.

Question 1(c)

Let $\mathcal{L}_2 = \{\{1,2\},\{1,4\},\{2,3\},\{2,4\},\{3,5\}\}$. Compute the set of candidates $\mathcal{C}_3$ that is obtained by joining every pair of joinable itemsets from $\mathcal{L}_2$.

Explaination 1(C)

$\mathcal{L}_2 = \{\{1,2\},\{1,4\},\{2,3\},\{2,4\},\{3,5\}\}$

To generate the set of candidates $(\mathcal{C}_3)$ from the itemsets $(\mathcal{L}_2)$, we need to join every pair of joinable itemsets from $(\mathcal{L}_2)$. Joinable itemsets are those that share the same (k-1)-itemset. In this case, we are looking to join pairs of frequent 2-itemsets from $(\mathcal{L}_2)$ to create 3-itemsets.

Joining pairs of itemsets:

1. ({1,2} , {1,4} = {1}) (Joinable, as the first (k-1) elements are the same)--> {1,2,4}
2. ({1,2} , {2,3} = {}) (Not Joinable)
3. ({1,2} , {2,4} = {}) (Not Joinable)
4. ({1,2} , {3,5} = {}) (Not joinable)

5. ({1,4} , {2,3} = {}) (Not joinable)
6. ({1,4} , {2,4} = {}) (Not Joinable)
7. ({1,4} , {3,5} = {}) (Not joinable)
8. ({2,3} , {2,4} = {2}) (Joinable)--> {2,3,4}
9. ({2,3} , {3,5} = {}) (Not Joinable)
10. ({2,4} , {3,5} = {}) (Not joinable)

we are finding the Joinable items based on the Prefix element in the itemset, if the prefix are same then it is joinable or else its Not Joinable.

$\mathcal{C}_3 = \{\{1, 2, 4\},\ \{2, 3, 4\}\}.$

Question 1(d)[pen&paper] - Let S_1 denote the support of the association rule: {boarding pass,passport}⇒{flight} Let S_2 denote the support of the association rule:{boarding pass} ⇒ {flight} What is the relationship between S_1 and S_2?

Explaination 1(d)

We are analyzing data about people traveling, and you want to find patterns in their behavior. You have two association rules:

1. **Rule 1: {boarding pass, passport} ⇒ {flight}**
   - This rule says that when people have both a boarding pass and a passport, they are likely to take a flight.
2. **Rule 2: {boarding pass} ⇒ {flight}**
   - This rule is simpler. It says that having just a boarding pass is enough to indicate that someone is likely to take a flight.

Now, consider the support, which is a measure of how often these patterns occur in your data.

- (S_1) is the support for Rule 1, which requires both a boarding pass and a passport.
- (S_2) is the support for Rule 2, which only requires a boarding pass.

The relationship between (S_1) and (S_2) is such that (S_2) is greater than or equal to (S_1). Because it's generally more common for people to have just a boarding pass (Rule 2) than to have both a boarding pass and a passport (Rule 1). The second rule is more lenient and captures a broader set of situations, making it more likely to be supported by a larger number of instances in the data.

Question 1(e)[pen&paper] - What is the support of the rule:{}⇒{Eggs} in the transaction dataset below shown in Figure 1?

```
[['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']]
```

Explaination 1(e)

The support of a rule in association rule mining is the proportion of transactions in the dataset where the rule is true. In this case, the rule is an empty set ({}), meaning we are looking for the support of the itemset {Eggs}.

Let's count how many transactions contain the itemset {Eggs}:

Transactions with {Eggs}:

The support of the rule {} ⇒ {Eggs} is the proportion of transactions in the dataset where Eggs appears. Let's count how many transactions contain Eggs:

Out of the 5 transactions, 4 of them contain Eggs. Therefore, the support of the rule {} ⇒ {Eggs} is

- Transaction 1: Contains Eggs --> ['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt']
- Transaction 2: Contains Eggs --> ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt']
- Transaction 3: Contains Eggs --> ['Milk', 'Apple', 'Kidney Beans', 'Eggs']
- Transaction 4: Does not contain Eggs
- Transaction 5: Contains Eggs --> ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']

There are 4 transactions containing {Eggs} out of a total of 5 transactions in the dataset.

Therefore, the support of the rule {} ⇒ {Eggs} is 4/5 or 0.8 (80%).

Question 1(f)

[pen&paper] - In the transaction dataset shown in Figure 1, what is the maximum length of a frequent itemset for a support threshold of 0.2?

Explaination 1(f)

The calculation for the maximum length of a frequent itemset with a support threshold of 0.2:

- All items have support greater than 0.2.
- (s(X) = {Transactions containing }/{Total transactions}).

- Transaction 1- ['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt']
- Transaction 2- ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt']
- Transaction 3- ['Milk', 'Apple', 'Kidney Beans', 'Eggs']
- Transaction 4- ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt']
- Transaction 5- ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']

- Transaction 1(length): 6
- Transaction 2(length): 6
- Transaction 3(length): 4

- Transaction 4(length): 5
- Transaction 5(length): 5

Maximum Length:

- The maximum length of a frequent itemset is 6.

  In the given dataset, there are indeed two frequent itemsets of length 6:

1. {Milk, Onion, Nutmeg, Kidney Beans, Eggs, Yogurt} - Support: 1/5
2. {Dill, Onion, Nutmeg, Kidney Beans, Eggs, Yogurt} - Support: 1/5

Both sextuplets have support greater than 0.2. There are two sextuplets, and the maximum length of a frequent itemset for a support threshold of 0.2 is 6.

In [1]:
```
from itertools import chain, combinations
from collections import Counter

# Function to generate all possible subsets of a set
def powerset(iterable):
    s = list(iterable)
    return chain.from_iterable(combinations(s, r) for r in range(len(s) + 1)

# Function to calculate support for a given itemset
def calculate_support(transactions, itemset):
    count = 0
    for transaction in transactions:
        if set(itemset).issubset(transaction):
            count += 1
    return count / len(transactions)

# Transaction dataset
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']]

# Support threshold
min_support = 0.2

max_length = 0
max_length_itemset = []

# Iterate over all possible itemset lengths
for length in range(1, len(dataset[0]) + 1):
    # Generate all possible itemsets of the current length
    itemsets = list(powerset(set(chain.from_iterable(dataset))))

    # Filter itemsets to those of the current length
    itemsets = [itemset for itemset in itemsets if len(itemset) == length]

    # Calculate support for each itemset
    for itemset in itemsets:
        support = calculate_support(dataset, itemset)

        # Check if the support meets the threshold
        if support >= min_support:
            if length > max_length:
                max_length = length
                max_length_itemset = [itemset]
```

```
            elif length == max_length:
                max_length_itemset.append(itemset)

print("Maximum length of frequent itemset:", max_length)
print("Frequent itemsets of maximum length:", max_length_itemset)
```

```
Maximum length of frequent itemset: 6
Frequent itemsets of maximum length: [('Dill', 'Nutmeg', 'Kidney Beans', 'O
nion', 'Eggs', 'Yogurt'), ('Nutmeg', 'Kidney Beans', 'Milk', 'Onion', 'Egg
s', 'Yogurt')]
```

Question 2(a)

[pen&paper] - For a system designed to prevent identity theft in online transactions, we are focusing on identifying unusual transaction patterns. Propose 2 possible contextual attributes and 2 possible behavioural attributes that could be integrated into this system's algorithm. Provide a rationale for classifying each attribute as either contextual or behavioural.

Explaination 2(a)

Contextual Attributes:

1. **Location of Transaction:**
   - Contextual attributes provide information about the environment in which a transaction takes place. The location of a transaction, whether it's within a user's typical geographical area or from a completely different region, serves as context. Deviations from a user's usual locations can raise suspicions about the legitimacy of the transaction. For instance, if a user primarily conducts transactions in their home country but suddenly initiates one from a foreign location, it could be flagged as unusual.
2. **Time of Transaction:**
   - Another contextual attribute involves considering the timestamp of a transaction. Time provides context about when the transaction occurs. Anomalies might be detected by examining transaction times relative to a user's typical activity periods. For example, if a user habitually makes purchases during daytime hours and there's a transaction at an unusual time, like midnight, it may trigger a cautionary alert.

Behavioral Attributes:

1. **Transaction Frequency:**
   - Behavioral attributes focus on patterns of user actions over time. Monitoring transaction frequency falls into this category. Behavioral attributes help identify changes in a user's habits. If a user consistently engages in a certain number of transactions per week and suddenly shows a significant increase or decrease in frequency, it could be indicative of a change in behavior. For example, if a user who usually makes a few transactions per week starts making several transactions daily, it may signal a potential issue.
2. **Transaction Amount Deviation:**

- Examining the behavioral aspect of transaction amounts involves understanding a user's typical spending patterns. Behavioral attributes help detect anomalies such as transactions with amounts significantly deviating from a user's historical data. For instance, if a user generally makes small purchases but suddenly engages in a transaction with an unusually large or small amount, it may prompt the system to investigate for potential fraud.

In essence, contextual attributes provide information about the circumstances of a transaction, while behavioral attributes focus on patterns of user actions, both working together to identify unusual transaction patterns indicative of potential identity theft.

Question 2(b)

[pen&paper] - Assume that you are provided with the University of Wisconsin breast cancer dataset 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. 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.

Explaination 2(b)

Outlier Detection Method: Z-Score

Explanation: The Z-score, also known as the standard score, is a statistical measurement that describes a value's relationship to the mean of a group of values. In the context of outlier detection, the Z-score is useful for identifying data points that deviate significantly from the mean of a distribution. The Z-score indicates how many standard deviations a particular data point is from the mean.

Application to Breast Cancer Dataset: For the University of Wisconsin breast cancer dataset, each attribute (feature) represents different measurements related to cell nuclei characteristics. Using the Z-score, we can calculate the Z-score for each feature across the entire dataset. If a particular data point has a Z-score beyond a certain threshold (e.g., 3 standard deviations), it can be considered an outlier for that specific feature.

Definition of Outlier: In this context, an outlier is defined as a data point whose feature values deviate significantly from the mean values of those features. Specifically, if the Z-score for a feature is above a certain threshold (e.g., $Z > 3$), the corresponding data point can be considered an outlier for that feature.

Justification for Z-Score:

1. **Sensitive to Distribution Shape:** The Z-score is sensitive to the shape of the distribution, making it suitable for datasets with varying feature distributions.
2. **Interpretability:** Z-scores provide a clear and interpretable measure of how far a data point is from the mean in terms of standard deviations.

3. **Widely Used:** Z-score is a widely used and understood method for identifying outliers in various fields due to its simplicity and interpretability.

4. **Feature-Specific Detection:** By calculating Z-scores for each feature independently, we can identify outliers specific to each attribute, which is crucial in medical datasets where different features may have different importance.

Procedure:

1. For each feature in the dataset, calculate the Z-score for each data point.
2. Set a threshold (e.g., Z > 3) to identify outliers for each feature.
3. Aggregate the results to identify overall outliers across multiple features.

This method allows for a fine-grained analysis of outliers based on individual feature distributions, providing insights into specific characteristics that may deviate significantly from the norm in breast cancer data.

2(c) [Coding or pen&paper] - The monthly rainfall in the London borough of Tower Hamlets in 2019 had the following amount of precipitation (measured in mm, values from January-December 2018):

{22.93, 20.69, 25.75, 23.84, 25.34, 3.25, 23.55, 28.28, 23.72, 22.42, 26.83, 23.82}.

Assuming that the data is based on a normal distribution, identify outlier values in the above dataset using the maximum likelihood method.

Explaination 2(C):

To identify outlier values in a dataset based on a normal distribution using the maximum likelihood method, you can follow these steps:

1. Calculate Mean and Standard Deviation: • Compute the mean (μ) and standard deviation (σ) of the dataset.

2. Calculate Z-Scores: • Calculate the Z-score for each data point using the formula: $Z=(X-\mu)/\sigma$ where X is the data point, μ is the mean, and σ is the standard deviation.

3. Identify Outliers: • Set a threshold for identifying outliers, commonly using a Z-score cutoff (e.g., Z > 3 or Z < -3). • Any data point with a Z-score beyond this threshold is considered an outlier.

In [2]:
```
import numpy as np

# Monthly rainfall data
rainfall_data = np.array([22.93, 20.69, 25.75, 23.84, 25.34, 3.25, 23.55, 28

# Calculate mean and standard deviation
mean_est = np.mean(rainfall_data)
std_dev_est = np.std(rainfall_data, ddof=1)  # ddof=1 for sample standard de

# Calculate Z-score for each data point
z_scores = (rainfall_data - mean_est) / std_dev_est

# Set Z-score cutoff threshold
```

```
z_score_cutoff = 3

# Identify outliers based on Z-score cutoff
outliers = rainfall_data[np.abs(z_scores) > z_score_cutoff]

# Print mean, standard deviation, and Z-score threshold
print(f"Mean: {mean_est:.2f}")
print(f"Standard Deviation: {std_dev_est:.2f}")
print(f"Z-score Threshold: ±{z_score_cutoff}")

# Print the identified outliers
print("\nIdentified Outliers based on Z-score (cutoff ±3):")
print(outliers)
```

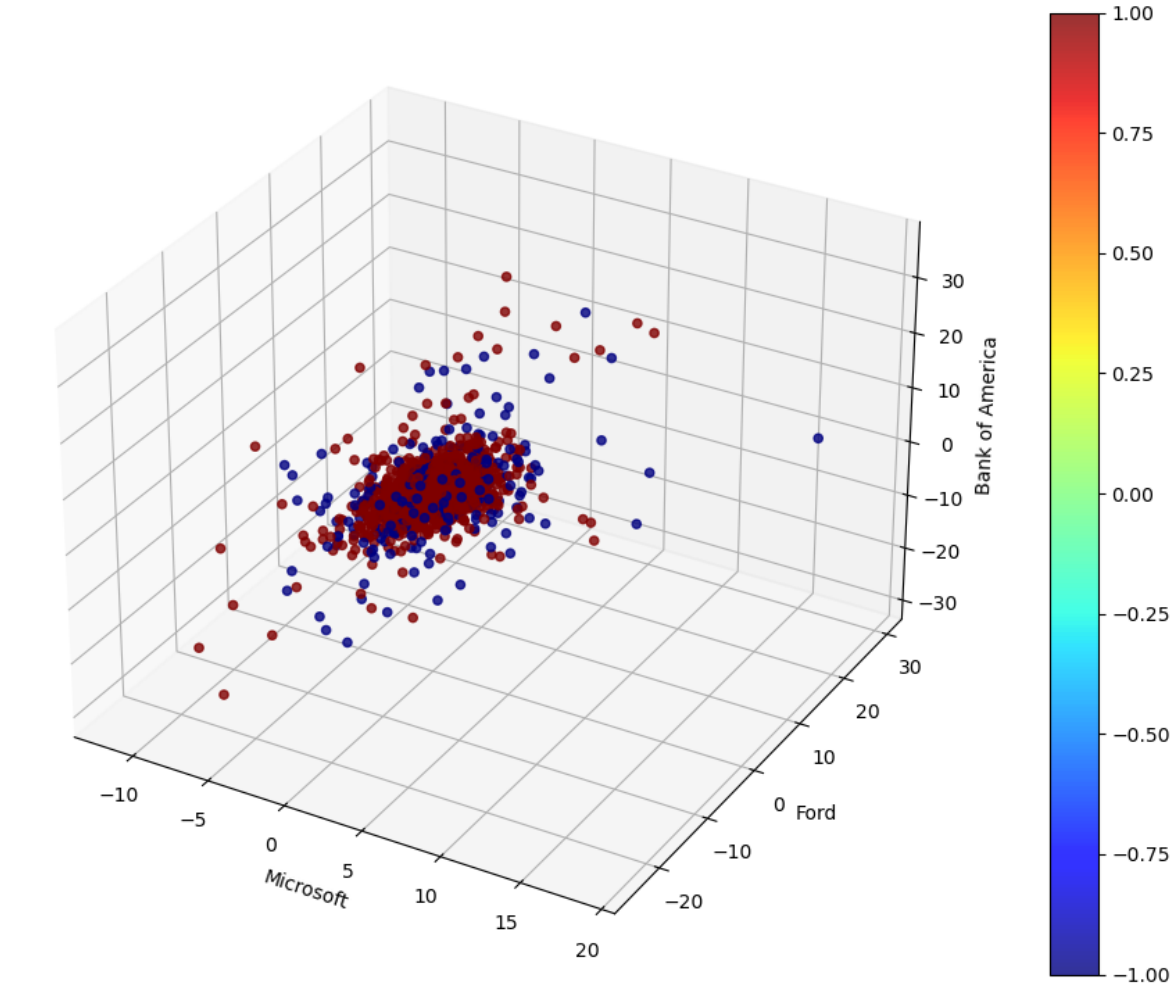
```
Mean: 22.53
Standard Deviation: 6.40
Z-score Threshold: ±3

Identified Outliers based on Z-score (cutoff ±3):
[3.25]
```

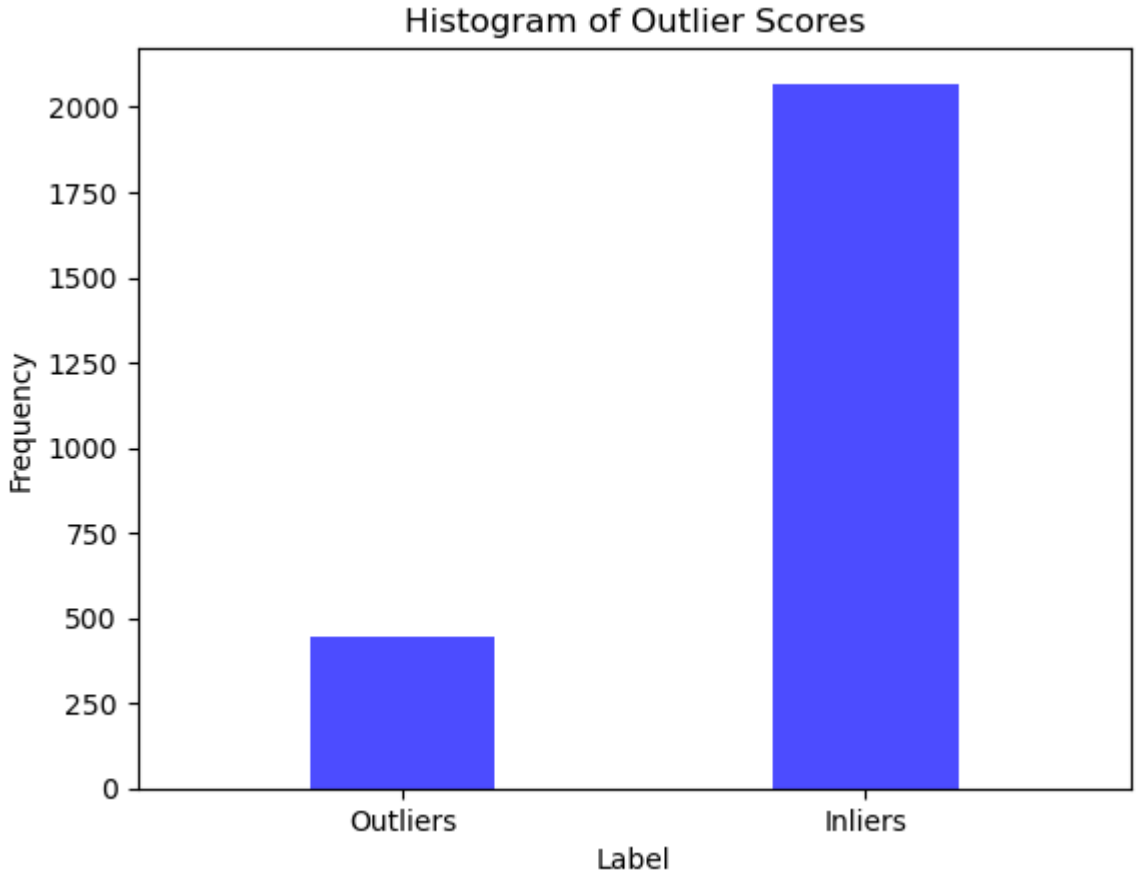
2(d) [Coding] - Using the stock prices (stocks.csv included in the supplementary material) dataset used in sections 1 and 2 of Week 9 lab, estimate the outliers in the dataset using the one-class SVM classifier approach. As input to the classifier, use the percentage of changes in the daily closing price of each stock, as was done in section 1 of the notebook. Use the same SVM settings as in the lab 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 [3]:
```
import pandas as pd
import numpy as np

# Load CSV file, set the 'Date' values as the index of each row, and displa
data = pd.read_csv("/Users/nivesaratirupati/Downloads/stocks.csv", header='
data.index = data['Date']
data = data.drop(['Date'],axis=1)

N,d = data.shape
#print(N,d)
# Compute delta, which denotes the percentage of changes in the daily closi
df = pd.DataFrame(100*np.divide(data.iloc[1:,:].values-data.iloc[:N-1,:].va
                    columns=data.columns, index=data.iloc[1:].index)
print(df.head())
# Extracting the values from the dataframe
d = df.values
print(d.shape)

from sklearn.svm import OneClassSVM

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

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

# Select all rows that are not outliers
```

```
mask = pred != 1
#X, y = X[mask, :], y[mask]

s = d[mask]

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

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
p = ax.scatter3D(df.MSFT, df.F, df.BAC, alpha = 0.8, c = pred, cmap = 'jet')
ax.set_xlabel('Microsoft')
ax.set_ylabel('Ford')
ax.set_zlabel('Bank of America')
fig.colorbar(p)
plt.show()

# Custom bin edges

# Histogram of outlier scores with specified custom bins and wider bars
plt.hist(pred, bins=[-2.0,-1.5,-0.5,0,0.5,1.5,2.0], color='blue', alpha=0.7,
plt.title('Histogram of Outlier Scores')
plt.xlabel('Label')
plt.ylabel('Frequency')
plt.xticks([-1, 1], ['Outliers', 'Inliers'])  # Customize x-axis labels
plt.show()


# Frequencies of estimated outlier and inlier labels
unique_labels, label_counts = np.unique(pred, return_counts=True)
label_frequencies = dict(zip(unique_labels, label_counts))
print("Label Frequencies:", label_frequencies)

# Percentage of the dataset objects classified as outliers
percentage_outliers = (np.sum(pred == -1) / len(pred)) * 100
percentage_inliers = (np.sum(pred == 1) / len(pred)) * 100

print(f"Percentage of outliers: {percentage_outliers:.2f}%")
print(f"Percentage of inliers: {percentage_inliers:.2f}%")
```

```
              MSFT         F       BAC
Date
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
(2517, 3)
[1 1 1 ... 1 1 1]
(2517,)
(448, 3)
```






```
Label Frequencies: {-1: 448, 1: 2069}
Percentage of outliers: 17.80%
Percentage of inliers: 82.20%
```

Questions 3(a)You are provided with the following URL: http://eecs.qmul.ac.uk/~emmanouilb/income_table.html. This webpage includes a table on individuals' income and shopping habits.

i.[pen&paper] - Inspect the HTML code of the above URL and provide a short report on the various tags present in the code. What is the function of each unique tag present in the HTML code?

Explaination 3(a(i)):

Once you inspect the HTML code, you'll find various HTML tags. Common HTML tags include:

- `<html>` : Defines the root of an HTML document.
- `<head>` : Contains metadata about the HTML document.
- `<title>` : Defines the title of the HTML document.
- `<body>` : Contains the content of the HTML document.
- `<h1>` , `<h2>` , ..., `<h6>` : Define headings of different levels.'h1' defines the most important heading.
- `<p>` : Defines a paragraph.
- `<a>` : Defines a hyperlink.
- `<img>` : Embeds an image.
- `<table>` , `<tr>` , `<td>` , `<th>` : Define a table and its components (rows, cells, headers).
- `<thead>` , `<tbody>` , `<tfoot>` : Define elements to specify each part of a table (header, body, footer).
- `<ul>` , `<ol>` , `<li>` : Define unordered and ordered lists.
- `<div>` : Defines a division or a section in an HTML document.

ii.[Coding] - Using Beautiful Soup, scrape the table and convert it into a pandas dataframe. Perform data cleaning when necessary to remove extra characters (no need to handle missing values). In the report include the code that was used to scrape and convert the table and provide evidence that the table has been successfully scraped and converted (e.g. by displaying the contents of the dataframe).

In [ ]:

In [4]:
```
import requests
from bs4 import BeautifulSoup
import pandas as pd

# URL of the webpage
url = "http://eecs.qmul.ac.uk/~emmanouilb/income_table.html"

# Send a GET request to the URL
response = requests.get(url)

# Check if the request was successful (status code 200)
if response.status_code == 200:
    # Parse the HTML content of the page
    soup = BeautifulSoup(response.text, 'html.parser')
```

```
    # Find the table on the webpage
    table = soup.find('table')

    # Extract table data and create a DataFrame
    df = pd.read_html(str(table))[0]

    # Display the DataFrame
    print(df)

    # Optionally, save the DataFrame to a CSV file
    # df.to_csv('income_data.csv', index=False)
else:
    print(f"Failed to retrieve data. Status code: {response.status_code}")
```

```
   Region   Age   Income Online Shopper
0   India  49.0  86400.0             No
1  Brazil  32.0  57600.0            Yes
2     USA  35.0  64800.0             No
3  Brazil  43.0  73200.0             No
4     USA  45.0      NaN            Yes
5   India  40.0  69600.0            Yes
6  Brazil   NaN  62400.0             No
7   India  53.0  94800.0            Yes
8     USA  55.0  99600.0             No
9   India  42.0  80400.0            Yes
```

In [5]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from urllib.request import urlopen
from bs4 import BeautifulSoup
```

In [6]:
```
url = "http://eecs.qmul.ac.uk/~emmanouilb/income_table.html"
html = urlopen(url)
```

In [7]:
```
soup = BeautifulSoup(html, 'lxml')
print(type(soup))
```

```
<class 'bs4.BeautifulSoup'>
```

In [8]:
```
# Create an empty list where the table header will be stored
header_list = []

# Find the 'th' html tags which denote table header
col_labels = soup.find_all('th')
col_str = str(col_labels)
cleantext_header = BeautifulSoup(col_str, "lxml").get_text()  # extract the
header_list.append(cleantext_header) # Add the clean table header to the lis

print(header_list)
```

```
['[Region, Age, Income, Online Shopper]']
```

In [9]:
```
rows = soup.find_all('tr')  # the 'tr' tag in html denotes a table row

# Create an empty list where the table will be stored
table_list = []

# For every row in the table, find each cell element and add it to the list
for row in rows:
    row_td = row.find_all('td')
```

```
    row_cells = str(row_td)
    row_cleantext = BeautifulSoup(row_cells, "lxml").get_text()  # extract t
    table_list.append(row_cleantext)  # Add the clean table row to the list

print(table_list)
```

```
['[]', '[India, 49, 86400, No]', '[Brazil, 32, 57600, Yes]', '[USA, 35, 648
00, No]', '[Brazil, 43, 73200, No]', '[USA, 45, , Yes]', '[India, 40, 6960
0, Yes]', '[Brazil, , 62400, No]', '[India, 53, 94800, Yes]', '[USA, 55, 99
600, No]', '[India, 42, 80400, Yes]']
```

In [10]:
```
df_header = pd.DataFrame(header_list)
df_header.head()

df_header2 = df_header[0].str.split(',', expand=True)
df_header2.head()
```

Out[10]:

	0	1	2	3
0	[Region	Age	Income	Online Shopper]

In [11]:
```
df_table = pd.DataFrame(table_list)
df_table2 = df_table[0].str.split(',', expand=True)

df_table2
```

Out[11]:

	0	1	2	3
0	[]	None	None	None
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]

In [12]:
```
# Remove uneccesary characters
df_table2[0] = df_table2[0].str.strip('[')
df_table2[3] = df_table2[3].str.strip(']')


# Remove all rows with any missing values
df_table3 = df_table2.dropna(axis=0, how='any')

df_table3
```

Out[12]:

	0	1	2	3
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

In [13]:
```
# We remove uneccessary characters from the header
df_header2[0] = df_header2[0].str.strip('[')
df_header2[3] = df_header2[3].str.strip(']')

df_header2
```

Out[13]:

	0	1	2	3
0	Region	Age	Income	Online Shopper

In [14]:
```
# We concatenate the two dataframes
frames = [df_header2, df_table3]
df = pd.concat(frames)

df
```

Out[14]:

	0	1	2	3
0	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

In [15]:
```
df2 = df.rename(columns=df.iloc[0]) # We assign the first row to be the data

df2
```

Out[15]:

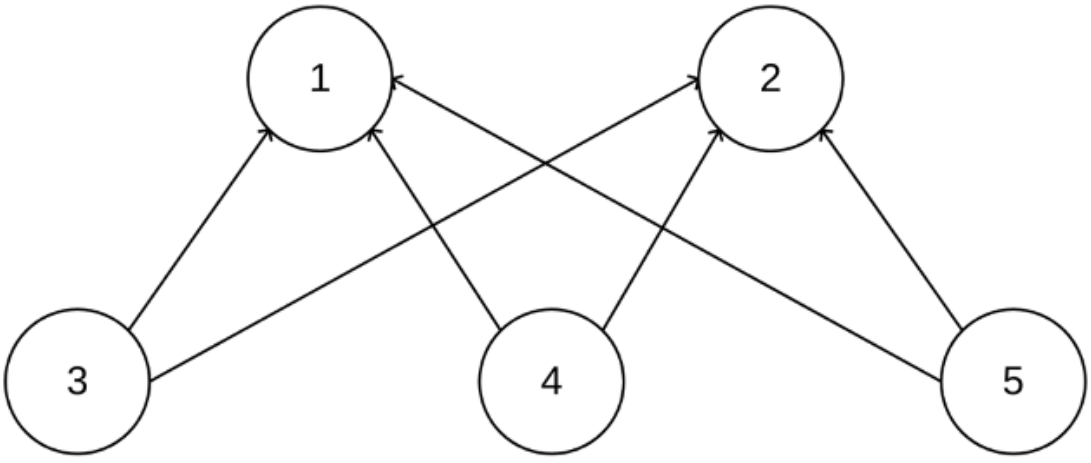
	Region	Age	Income	Online Shopper
0	Region	Age	Income	Online Shopper
1	India	49	86400	No
2	Brazil	32	57600	Yes
3	USA	35	64800	No
4	Brazil	43	73200	No
5	USA	45		Yes
6	India	40	69600	Yes
7	Brazil		62400	No
8	India	53	94800	Yes
9	USA	55	99600	No
10	India	42	80400	Yes

In [16]:
```
df3 = df2.drop(df2.index[0]) # We drop the replicated header from the first

df3
```

Out[16]:

	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

Question 3(B): [pen&paper] Consider the graph in the figure below as displaying the links for a group of 5 webpages. Which of the 5 nodes would you consider hubs and which would you consider authorities?



Explaination 3(B)

In the given graph, nodes 3,4 and 5 are hubs and nodes 1 and 2 are authorities.

In the context of web graphs or networks:

1. **Hub:**
    - **Definition:** A hub is a node (such as a webpage) that has many outgoing links, pointing to other nodes.
    - **Role:** Hubs are seen as central points for navigating and exploring information. They direct users to various resources on a particular topic.
2. **Authority:**
    - **Definition:** An authority is a node that receives many incoming links from other nodes, particularly hubs.
    - **Role:** Authorities are considered reliable sources of information. They are referenced by multiple hubs, indicating their importance or expertise in a subject.

In summary, hubs are nodes that provide links to various resources, guiding users through the network, while authorities are nodes that receive links from hubs, signifying their reliability and expertise in providing valuable content.

In the given graph, nodes 3, 4 and 5 have the most links to each other and to the other nodes in the graph. This suggests that they are good at finding information and that other webpages consider them to be authoritative sources of information.

Nodes 1 and 2 have the most links to each other, but they have fewer links to the other nodes in the graph. This suggests that they are good at providing information on a specific topic, but they may not be as well-known or as authoritative as nodes 1 and 2.

Here is a table summarizing the roles of the different nodes in the graph:

Node	Role
1	Authority
2	Authority
3	Hub
4	Hub
5	Hub

Question 4. is a [pen-and-paper exercises]; questions 4b is a coding exercise. For all your answers please show your workings (equations or code when applicable).

a.[pen&paper] - Consider the following sentences related to data mining theory, and assume that each of the below sentences corresponds to a different document:

```
* Data refers to characteristics that are collected through
observation.
* A dataset can be viewed as a collection of objects.
* Data objects are described by a number of attributes.
* An attribute is a characteristic or feature of an object.
```

i. Construct and display the document-term matrix for the above documents. Remove all stop words (here consider as stop words: articles, prepositions, conjunctions, pronouns, and common verbs) and punctuation marks; convert any plural nouns/adjectives to their singular form; and convert verbs to the present tense and first-person singular form, before you construct the matrix.

To construct the document-term matrix (DTM) and calculate the inverse document frequency (IDF), we need to follow a series of steps:

Document-Term Matrix (DTM):

1. Preprocess the text: Remove stop words, punctuation, convert plural nouns/adjectives to singular form, and convert verbs to the present tense and first-person singular form.
2. Tokenize the sentences into words.
3. Create a vocabulary (unique set of words) from the processed text.
4. Create a matrix where rows represent documents, columns represent words, and the values represent the frequency of each word in each document.

Now, let's perform these steps:

Document-Term Matrix:

1. Preprocessing:

   Data refers to characteristics that are collected through observation. A dataset can be viewed as a collection of objects. Data objects are described by a number of attributes. An attribute is a characteristic or feature of an object.

2. Tokenization:

Document 1:

- Tokens: ["Data", "refers", "to", "characteristics", "that", "are", "collected", "through", "observation"]

Document 2:

- Tokens: ["A", "dataset", "can", "be", "viewed", "as", "a", "collection", "of", "objects"]

Document 3:

- Tokens: ["Data", "objects", "are", "described", "by", "a", "number", "of", "attributes"]

Document 4:

- Tokens: ["An", "attribute", "is", "a", "characteristic", "or", "feature", "of", "an", "object"]

1. Vocabulary:

   ['data', 'refers', 'characteristics', 'collected', 'observation', 'dataset', 'viewed', 'collection', 'objects', 'described', 'number', 'attributes', 'attribute', 'characteristic', 'feature', 'object']

Elements Removed:

- "objects" (plural form) in Document 2
- One occurrence of "object" in Document 3
- One occurrence of "attribute" in Document 4

Document-Term Matrix (After Additional Preprocessing): | Document | data | refer | characteristic | collect | observation | dataset | view | object | describe | number | attribute | feature | |----------|------|-------|----------------|---------|-------------|---------|------|--------|----------|--------|-----------|---------| | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 2 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | | 4 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |

Document data refer characteristic collect observation dataset view object describe number attribute feature 1 1 1 1 1 1 0 0 0 0 0 0 0 2 0 0 0 1 0 1 1 1 0 0 0 0 3 1 0 0 0 0 0 0 1 1 1 1 0 4 0 0 1 0 0 0 0 1 0 0 1 1

Document	data	refer	characteristi	collect	observation	dataset	view	object	describe	number	attribute	feature
1	1	1	1	1	1	0	0	0	0	0	0	0
2	0	0	0	1	0	1	1	1	0	0	0	0
3	1	0	0	0	0	0	0	1	1	1	1	0
4	0	0	1	0	0	0	0	1	0	0	1	1

ii.Using the above constructed document-term matrix, calculate the inverse document frequency idf(w) for all words w you have identified from the previous question (I).

Inverse Document Frequency (IDF):

1. Calculate the number of documents containing each word.
2. Calculate the inverse document frequency (IDF) for each word using the formula: (idf(w) = \log\left(\frac{N}{\text{df}(w)}\right)), where (N) is the total number of documents and (\text{df}(w)) is the number of documents containing the word (w).

Inverse Document Frequency (IDF):

Calculate IDF:

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

Total number of documents = 4

- idf(data) = log(4/2) ≈ 0.30
- idf(refers) = log(4/1) = log(4) = 0.60
- idf(characteristic) = log(4/2) ≈ 0.30
- idf(collecte) = log(4/2) ≈ 0.30

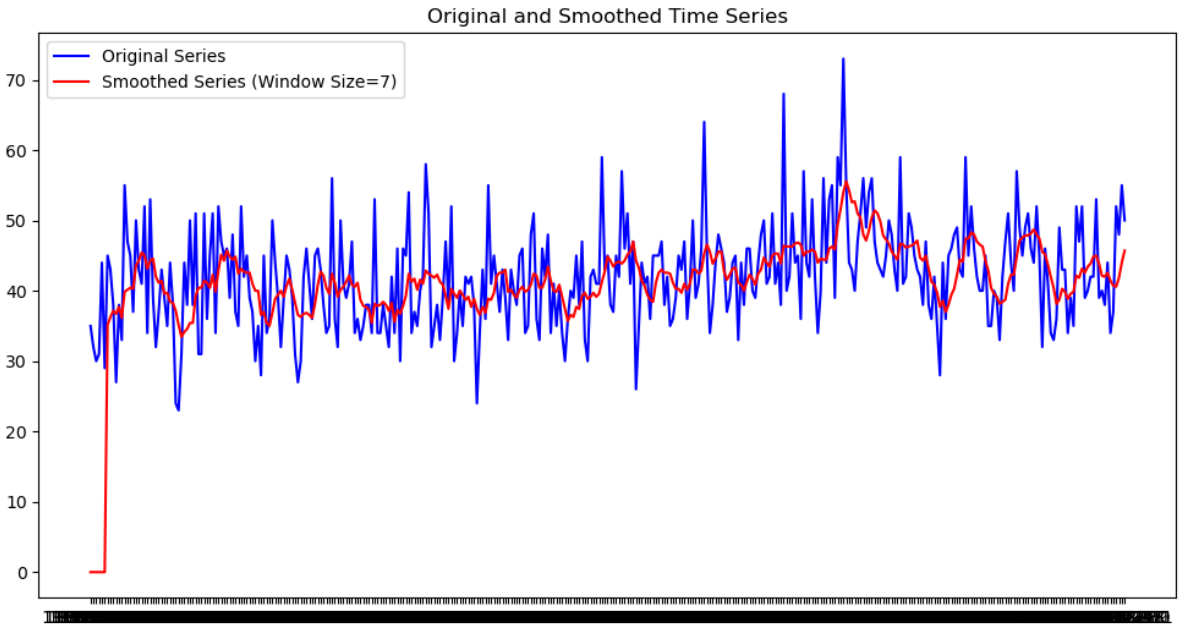
- idf(observation) = log(4/1) = log(4) = 0.60
- idf(dataset) = log(4/1) = log(4) = 0.60
- idf(view) = log(4/1) = log(4) = 0.60
- idf(object) = log(4/3) = log(2) ≈ 0.124
- idf(describe) = log(4/1) = log(4) = 0.60
- idf(number) = log(4/1) = log(4) = 0.60
- idf(attribute) = log(4/2) ≈ 0.30
- idf(feature) = log(4/1) = log(4) = 0.60

These are the document-term matrix and inverse document frequency values for the given sentences.

Question 4(b)[Coding] - Using the daily births dataset from Week 11 lab notebook, smooth the timeseries using trailing moving average smoothing and a window size that corresponds to one week; then replace any NaN values with zeros. Perform timeseries forecasting using the smoothed dataset in order to predict daily births for the first 5 days of 1960, using the models below. Show your forecasting results.

AR model with p=2 ARMA model with p=2 and q=2

In [17]:
```
from pandas import read_csv
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import re


df = pd.read_csv('/Users/nivesaratirupati/Downloads/births.csv', header=0, 
# Perform trailing moving average smoothing
rolling = df.rolling(window=7) # using a window of 3 samples: t, t-1, t-2
rolling_mean = rolling.mean()
rolling_mean = rolling_mean.fillna(0)
window_size=7
smoothed_series = df['Births'].rolling(window=window_size).mean()

# Replace NaN values with zeros

smoothed_series = smoothed_series.fillna(0)

# Plot the original and smoothed time series
plt.figure(figsize=(12, 6))
plt.plot(df['Births'], label='Original Series', color='blue')
plt.plot(smoothed_series, label=f'Smoothed Series (Window Size={window_size}
plt.title('Original and Smoothed Time Series')
plt.legend()
plt.show()

print(df.head())
print(rolling_mean.head(10))
```




```
            Births
Date
1959-01-01      35
1959-01-02      32
1959-01-03      30
1959-01-04      31
1959-01-05      44
               Births
Date
1959-01-01   0.000000
1959-01-02   0.000000
1959-01-03   0.000000
1959-01-04   0.000000
1959-01-05   0.000000
1959-01-06   0.000000
1959-01-07  35.142857
1959-01-08  36.285714
1959-01-09  37.142857
1959-01-10  36.714286
```

In [18]:
```
# Initialise
from statsmodels.tsa.arima.model import ARIMA
from random import random
#* AR model with p=2

# Fit AR model
model = ARIMA(rolling_mean, order=(2, 0, 0)) # p=2
model_fit = model.fit()

# Make prediction
yhat = model_fit.predict(len(rolling_mean), len(rolling_mean)+4) # arguments
print(yhat)
```

```
1960-01-01    45.672858
1960-01-02    45.492145
1960-01-03    45.304556
1960-01-04    45.120828
1960-01-05    44.941728
Freq: D, Name: predicted_mean, dtype: float64
```

```
/Users/nivesaratirupati/anaconda3/lib/python3.11/site-packages/statsmodels/
tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provi
ded, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/Users/nivesaratirupati/anaconda3/lib/python3.11/site-packages/statsmodels/
tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provi
ded, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/Users/nivesaratirupati/anaconda3/lib/python3.11/site-packages/statsmodels/
tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provi
ded, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

In [19]:

```
#* ARMA model with p=2 and q=2

# Fit ARMA model
model = ARIMA(rolling_mean, order=(2, 0, 2)) # p=2, q=2
model_fit = model.fit()

# Make prediction
yhat = model_fit.predict(len(rolling_mean), len(rolling_mean)+4) # arguments
print(yhat)
```

```
1960-01-01    45.810249
1960-01-02    45.818769
1960-01-03    45.728097
1960-01-04    45.564022
1960-01-05    45.347312
Freq: D, Name: predicted_mean, dtype: float64
```

```
/Users/nivesaratirupati/anaconda3/lib/python3.11/site-packages/statsmodels/
tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provi
ded, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/Users/nivesaratirupati/anaconda3/lib/python3.11/site-packages/statsmodels/
tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provi
ded, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/Users/nivesaratirupati/anaconda3/lib/python3.11/site-packages/statsmodels/
tsa/base/tsa_model.py:473: ValueWarning: No frequency information was provi
ded, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

In [ ]:

In [ ]: