Assignment2_PavanKumar_GondabalRamakrishna_230049961

December 15, 2023

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?

Advantages of Apriori Algorithm Over Subset Support Calculation:

The Apriori algorithm is a widely-used method for discovering frequent itemsets in datasets and generating association rules based on these itemsets. Its advantages over the approach of computing the support of every subset of an itemset are notable:

The algorithm's efficiency is rooted in its iterative level-wise search strategy. By leveraging prior knowledge of frequent itemset properties, the Apriori algorithm explores (k + 1)-itemsets based on existing k-itemsets. This approach significantly reduces computational complexity, especially when dealing with large datasets.

To further enhance efficiency, the Apriori algorithm utilizes the Apriori property, a crucial criterion that dictates that every non-empty subset of a frequent itemset must also be frequent. This property allows the algorithm to discard the generation of non-frequent subsets, effectively minimizing the search space.

The Apriori algorithm operates through two key steps: join and prune. The join step involves obtaining candidate sets (Ck) by pairing every joinable pair of itemsets from the previous level (Lk-1). Subsequently, the prune step eliminates candidates from Ck to derive the final frequent itemset (Lk). Items failing to meet the minimum support threshold are considered infrequent and removed during pruning. This step is crucial for reducing the size of candidate itemsets and avoiding unnecessary calculations for unlikely frequent combinations.

By prioritizing smaller itemsets before larger ones, the Apriori algorithm exhibits memory efficiency. Its approach ensures that the search for frequent itemsets is conducted in a systematic and practical manner, making it a favorable choice over the exhaustive calculation of support for every subset of an itemset.

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?

Certainly! Let's consider a different frequent 3-itemset, such as {Coffee Maker, Blender, Toaster}, in the context of the Apriori property.

For any k (k 2), the Apriori property asserts that every frequent k-itemset must be a superset of at least one itemset in the set of frequent 1-itemsets (L_1). This property is grounded in the idea that if an itemset is frequent, all its subsets must also be frequent.

Now, imagine we are analyzing a dataset of kitchen appliance purchases, with each transaction representing items in a customer's order. Frequent 1-itemsets could include items like Coffee Maker, Blender, and Toaster, which are frequently purchased individually.

Moving on to frequent 2-itemsets, we may find pairs like Coffee Maker and Blender, Coffee Maker and Toaster, and Blender and Toaster, indicating items often bought together based on the frequent 1-itemsets.

Now, let's explore a frequent 3-itemset, {Coffee Maker, Blender, Toaster}. According to the Apriori property, this 3-itemset must include at least one of the frequent 1-itemsets. In this case, it encompasses all three frequent 1-itemsets: Coffee Maker, Blender, and Toaster.

In essence, the Apriori property ensures that if a combination of kitchen appliances is frequently purchased together, it is guaranteed to include at least one of the items recognized as frequently purchased individually. This principle remains valuable for extracting meaningful associations and patterns from transactional data.

c. [pen&paper] - Given the set (L_2 = {{1,2},{1,4},{2,3},{2,4},{3,5}}), calculate the set of candidates (C_3) by combining every pair of joinable itemsets from (L_2).

Two sets are considered joinable if their first elements are identical, and the last element in the first set is smaller than the last element in the second set.

Joinable Itemsets: - ({1,2}) and ({1,4}) share a common element, 1, making them joinable. - ({2,3}) and ({2,4}) share a common element, 2, making them joinable. - ({3,5}) has no joinable partner in (L_2).

Joining Itemsets: - ({1,2}) join ({1,4}) results in ({1,2,4}). - ({2,3}) join ({2,4}) results in ({2,3,4}).

Pruning and Checking Subsets: Subsets of ({1,2,4}): ({1,2}, {1,4}, {2,4}). All are frequent in (L_2). Subsets of ({2,3,4}): ({2,3}, {2,4}, {3,4}). All are frequent in (L_2).

Hence, (C_3 = {{1,2,4}, {2,3,4}}).

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?

S_1 represents the support of the association rule {boarding pass, passport} {flight}, while S_2 represents the support of the association rule {boarding pass} {flight}.

The introduction of additional items like passport in the given context serves to decrease the frequency of transactions where all three items (boarding pass, passport, and flight) are present simultaneously. Additionally, S_2 can be regarded as a subset of S_1. Consequently, the support of the rule {boarding pass, passport} {flight} is consistently less than or equal to the support of the rule {boarding pass} {flight}.

Since S_1 encompasses an extra item (passport) compared to S_2, it can be concluded that S_1 is invariably less than or equal to S_2.

In summary, the relationship between S_1 and S_2 can be succinctly expressed as: S_1>=S_2

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

To calculate the support of a rule, we need to count the number of transactions that contain both the antecedent and the consequent.

Here the antecedent is the empty set and consquent is the item {Eggs}.

In the transaction dataset shown in the image above, there are 5 transactions. Out of these, 5 of the transactions contain eggs:

[‘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 the rule {} => {Eggs} is 4 / 5 = 0.8 Therefore, The support of the rule {} => {Eggs} in the transaction dataset is 80%. This means that all of the transactions in the dataset contain eggs.

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?

[ ]:
```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori

# The transaction dataset
transactions = [
    ['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(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)

# frequent itemsets with a support threshold= 0.2
frequent_itemsets = apriori(df, min_support=0.2, use_colnames=True)

# the maximum length of frequent itemsets
max_length = max(frequent_itemsets['itemsets'].apply(len))

print("Maximum Length of Frequent Itemset:", max_length)
```

```
# Display each step of pruning and elimination
for length in range(1, max_length + 1):
    print(f"\nFrequent {length}-itemsets:")

    # Extract itemsets of the current length
    current_length_itemsets = frequent_itemsets[frequent_itemsets['itemsets'].
 ↪apply(len) == length]

    # Display the current length itemsets
    display(current_length_itemsets)

    # Prune infrequent itemsets
    current_length_itemsets =␣
 ↪current_length_itemsets[current_length_itemsets['support'] >= 0.2]

    # Display the pruned itemsets
    print("\nAfter pruning infrequent itemsets:")
    display(current_length_itemsets)

# Display the final frequent itemsets
print("\nFinal Frequent Itemsets:")
display(frequent_itemsets)
```

```
Maximum Length of Frequent Itemset: 6

Frequent 1-itemsets:

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

	support	itemsets
0	0.2	(apple)
1	0.4	(corn)
2	0.2	(dill)
3	0.8	(eggs)
4	0.2	(ice cream)
5	1.0	(kidney beans)
6	0.6	(milk)
7	0.4	(nutmeg)
8	0.6	(onion)
9	0.2	(unicorn)
10	0.6	(yogurt)

```
After pruning infrequent itemsets:
```

	support	itemsets
0	0.2	(apple)
1	0.4	(corn)
2	0.2	(dill)
3	0.8	(eggs)
4	0.2	(ice cream)
5	1.0	(kidney beans)
6	0.6	(milk)
7	0.4	(nutmeg)
8	0.6	(onion)
9	0.2	(unicorn)
10	0.6	(yogurt)

Frequent 2-itemsets:

	support	itemsets
11	0.2	(eggs, apple)
12	0.2	(kidney beans, apple)
13	0.2	(milk, apple)
14	0.2	(eggs, corn)
15	0.2	(ice cream, corn)
16	0.4	(kidney beans, corn)
17	0.2	(milk, corn)
18	0.2	(onion, corn)
19	0.2	(corn, unicorn)
20	0.2	(yogurt, corn)
21	0.2	(eggs, dill)
22	0.2	(kidney beans, dill)
23	0.2	(nutmeg, dill)
24	0.2	(onion, dill)
25	0.2	(yogurt, dill)
26	0.2	(eggs, ice cream)
27	0.8	(eggs, kidney beans)
28	0.4	(eggs, milk)
29	0.4	(nutmeg, eggs)
30	0.6	(eggs, onion)
31	0.4	(eggs, yogurt)
32	0.2	(kidney beans, ice cream)
33	0.2	(onion, ice cream)
34	0.6	(milk, kidney beans)
35	0.4	(nutmeg, kidney beans)
36	0.6	(kidney beans, onion)
37	0.2	(kidney beans, unicorn)
38	0.6	(yogurt, kidney beans)
39	0.2	(nutmeg, milk)
40	0.2	(milk, onion)
41	0.2	(milk, unicorn)
42	0.4	(milk, yogurt)

```
43      0.4            (nutmeg, onion)
44      0.4           (nutmeg, yogurt)
45      0.4            (yogurt, onion)
46      0.2          (yogurt, unicorn)
```

After pruning infrequent itemsets:

```
    support                   itemsets
11      0.2              (eggs, apple)
12      0.2      (kidney beans, apple)
13      0.2              (milk, apple)
14      0.2               (eggs, corn)
15      0.2          (ice cream, corn)
16      0.4       (kidney beans, corn)
17      0.2               (milk, corn)
18      0.2              (onion, corn)
19      0.2            (corn, unicorn)
20      0.2             (yogurt, corn)
21      0.2               (eggs, dill)
22      0.2       (kidney beans, dill)
23      0.2             (nutmeg, dill)
24      0.2              (onion, dill)
25      0.2             (yogurt, dill)
26      0.2          (eggs, ice cream)
27      0.8       (eggs, kidney beans)
28      0.4               (eggs, milk)
29      0.4             (nutmeg, eggs)
30      0.6              (eggs, onion)
31      0.4             (eggs, yogurt)
32      0.2  (kidney beans, ice cream)
33      0.2         (onion, ice cream)
34      0.6       (milk, kidney beans)
35      0.4     (nutmeg, kidney beans)
36      0.6      (kidney beans, onion)
37      0.2    (kidney beans, unicorn)
38      0.6     (yogurt, kidney beans)
39      0.2             (nutmeg, milk)
40      0.2              (milk, onion)
41      0.2            (milk, unicorn)
42      0.4             (milk, yogurt)
43      0.4            (nutmeg, onion)
44      0.4           (nutmeg, yogurt)
45      0.4            (yogurt, onion)
46      0.2          (yogurt, unicorn)
```

Frequent 3-itemsets:

```
    support                          itemsets
```

```
47      0.2       (eggs, kidney beans, apple)
48      0.2              (eggs, milk, apple)
49      0.2       (milk, kidney beans, apple)
50      0.2           (eggs, ice cream, corn)
51      0.2        (eggs, kidney beans, corn)
52      0.2              (eggs, onion, corn)
53      0.2   (kidney beans, ice cream, corn)
54      0.2         (onion, ice cream, corn)
55      0.2        (milk, kidney beans, corn)
56      0.2       (kidney beans, onion, corn)
57      0.2     (kidney beans, corn, unicorn)
58      0.2      (yogurt, kidney beans, corn)
59      0.2             (milk, corn, unicorn)
60      0.2              (milk, yogurt, corn)
61      0.2          (yogurt, corn, unicorn)
62      0.2        (eggs, kidney beans, dill)
63      0.2              (nutmeg, eggs, dill)
64      0.2               (eggs, onion, dill)
65      0.2              (eggs, yogurt, dill)
66      0.2      (nutmeg, kidney beans, dill)
67      0.2       (kidney beans, onion, dill)
68      0.2      (yogurt, kidney beans, dill)
69      0.2             (nutmeg, onion, dill)
70      0.2           (nutmeg, yogurt, dill)
71      0.2            (yogurt, onion, dill)
72      0.2   (eggs, kidney beans, ice cream)
73      0.2         (eggs, onion, ice cream)
74      0.4        (eggs, milk, kidney beans)
75      0.4      (nutmeg, eggs, kidney beans)
76      0.6       (eggs, onion, kidney beans)
77      0.4      (eggs, yogurt, kidney beans)
78      0.2              (nutmeg, eggs, milk)
79      0.2               (eggs, milk, onion)
80      0.2              (eggs, yogurt, milk)
81      0.4            (nutmeg, eggs, onion)
82      0.4           (nutmeg, eggs, yogurt)
83      0.4            (eggs, yogurt, onion)
84      0.2  (kidney beans, onion, ice cream)
85      0.2      (nutmeg, milk, kidney beans)
86      0.2       (kidney beans, milk, onion)
87      0.2     (milk, kidney beans, unicorn)
88      0.4      (milk, yogurt, kidney beans)
89      0.4     (nutmeg, kidney beans, onion)
90      0.4    (nutmeg, yogurt, kidney beans)
91      0.4     (kidney beans, yogurt, onion)
92      0.2   (yogurt, kidney beans, unicorn)
93      0.2             (nutmeg, milk, onion)
94      0.2           (nutmeg, milk, yogurt)
```

```
95      0.2             (milk, yogurt, onion)
96      0.2           (milk, yogurt, unicorn)
97      0.4           (nutmeg, yogurt, onion)
```

After pruning infrequent itemsets:

```
    support                          itemsets
47      0.2       (eggs, kidney beans, apple)
48      0.2               (eggs, milk, apple)
49      0.2       (milk, kidney beans, apple)
50      0.2           (eggs, ice cream, corn)
51      0.2        (eggs, kidney beans, corn)
52      0.2               (eggs, onion, corn)
53      0.2   (kidney beans, ice cream, corn)
54      0.2         (onion, ice cream, corn)
55      0.2        (milk, kidney beans, corn)
56      0.2       (kidney beans, onion, corn)
57      0.2     (kidney beans, corn, unicorn)
58      0.2      (yogurt, kidney beans, corn)
59      0.2             (milk, corn, unicorn)
60      0.2              (milk, yogurt, corn)
61      0.2           (yogurt, corn, unicorn)
62      0.2        (eggs, kidney beans, dill)
63      0.2              (nutmeg, eggs, dill)
64      0.2               (eggs, onion, dill)
65      0.2              (eggs, yogurt, dill)
66      0.2      (nutmeg, kidney beans, dill)
67      0.2       (kidney beans, onion, dill)
68      0.2      (yogurt, kidney beans, dill)
69      0.2             (nutmeg, onion, dill)
70      0.2            (nutmeg, yogurt, dill)
71      0.2             (yogurt, onion, dill)
72      0.2   (eggs, kidney beans, ice cream)
73      0.2         (eggs, onion, ice cream)
74      0.4        (eggs, milk, kidney beans)
75      0.4      (nutmeg, eggs, kidney beans)
76      0.6       (eggs, onion, kidney beans)
77      0.4      (eggs, yogurt, kidney beans)
78      0.2              (nutmeg, eggs, milk)
79      0.2               (eggs, milk, onion)
80      0.2              (eggs, yogurt, milk)
81      0.4             (nutmeg, eggs, onion)
82      0.4            (nutmeg, eggs, yogurt)
83      0.4             (eggs, yogurt, onion)
84      0.2  (kidney beans, onion, ice cream)
85      0.2      (nutmeg, milk, kidney beans)
86      0.2       (kidney beans, milk, onion)
87      0.2     (milk, kidney beans, unicorn)
```

```
88      0.4      (milk, yogurt, kidney beans)
89      0.4     (nutmeg, kidney beans, onion)
90      0.4    (nutmeg, yogurt, kidney beans)
91      0.4     (kidney beans, yogurt, onion)
92      0.2   (yogurt, kidney beans, unicorn)
93      0.2             (nutmeg, milk, onion)
94      0.2            (nutmeg, milk, yogurt)
95      0.2             (milk, yogurt, onion)
96      0.2           (milk, yogurt, unicorn)
97      0.4           (nutmeg, yogurt, onion)
```

Frequent 4-itemsets:

```
     support                                itemsets
98       0.2       (eggs, milk, kidney beans, apple)
99       0.2   (eggs, kidney beans, ice cream, corn)
100      0.2          (eggs, onion, ice cream, corn)
101      0.2       (eggs, onion, kidney beans, corn)
102      0.2  (kidney beans, onion, ice cream, corn)
103      0.2     (milk, kidney beans, corn, unicorn)
104      0.2      (milk, yogurt, kidney beans, corn)
105      0.2   (yogurt, kidney beans, corn, unicorn)
106      0.2           (milk, yogurt, corn, unicorn)
107      0.2      (nutmeg, eggs, kidney beans, dill)
108      0.2       (kidney beans, eggs, onion, dill)
109      0.2      (eggs, yogurt, kidney beans, dill)
110      0.2             (nutmeg, eggs, onion, dill)
111      0.2            (nutmeg, eggs, yogurt, dill)
112      0.2             (eggs, yogurt, onion, dill)
113      0.2     (nutmeg, kidney beans, onion, dill)
114      0.2    (nutmeg, yogurt, kidney beans, dill)
115      0.2     (kidney beans, yogurt, onion, dill)
116      0.2           (nutmeg, yogurt, onion, dill)
117      0.2  (kidney beans, eggs, onion, ice cream)
118      0.2      (nutmeg, eggs, milk, kidney beans)
119      0.2       (eggs, milk, onion, kidney beans)
120      0.2      (eggs, yogurt, kidney beans, milk)
121      0.4     (nutmeg, eggs, onion, kidney beans)
122      0.4    (nutmeg, eggs, yogurt, kidney beans)
123      0.4     (eggs, yogurt, onion, kidney beans)
124      0.2             (nutmeg, eggs, milk, onion)
125      0.2            (nutmeg, eggs, yogurt, milk)
126      0.2             (eggs, yogurt, onion, milk)
127      0.4           (nutmeg, eggs, yogurt, onion)
128      0.2     (nutmeg, kidney beans, milk, onion)
129      0.2    (nutmeg, milk, yogurt, kidney beans)
130      0.2     (kidney beans, yogurt, onion, milk)
131      0.2   (milk, yogurt, kidney beans, unicorn)
```

```
132      0.4   (nutmeg, kidney beans, yogurt, onion)
133      0.2           (nutmeg, milk, yogurt, onion)
```

After pruning infrequent itemsets:

```
     support                                  itemsets
98       0.2       (eggs, milk, kidney beans, apple)
99       0.2   (eggs, kidney beans, ice cream, corn)
100      0.2          (eggs, onion, ice cream, corn)
101      0.2       (eggs, onion, kidney beans, corn)
102      0.2  (kidney beans, onion, ice cream, corn)
103      0.2     (milk, kidney beans, corn, unicorn)
104      0.2      (milk, yogurt, kidney beans, corn)
105      0.2   (yogurt, kidney beans, corn, unicorn)
106      0.2           (milk, yogurt, corn, unicorn)
107      0.2      (nutmeg, eggs, kidney beans, dill)
108      0.2       (kidney beans, eggs, onion, dill)
109      0.2      (eggs, yogurt, kidney beans, dill)
110      0.2             (nutmeg, eggs, onion, dill)
111      0.2            (nutmeg, eggs, yogurt, dill)
112      0.2             (eggs, yogurt, onion, dill)
113      0.2     (nutmeg, kidney beans, onion, dill)
114      0.2    (nutmeg, yogurt, kidney beans, dill)
115      0.2     (kidney beans, yogurt, onion, dill)
116      0.2           (nutmeg, yogurt, onion, dill)
117      0.2  (kidney beans, eggs, onion, ice cream)
118      0.2      (nutmeg, eggs, milk, kidney beans)
119      0.2       (eggs, milk, onion, kidney beans)
120      0.2      (eggs, yogurt, kidney beans, milk)
121      0.4     (nutmeg, eggs, onion, kidney beans)
122      0.4    (nutmeg, eggs, yogurt, kidney beans)
123      0.4     (eggs, yogurt, onion, kidney beans)
124      0.2             (nutmeg, eggs, milk, onion)
125      0.2            (nutmeg, eggs, yogurt, milk)
126      0.2             (eggs, yogurt, onion, milk)
127      0.4           (nutmeg, eggs, yogurt, onion)
128      0.2     (nutmeg, kidney beans, milk, onion)
129      0.2    (nutmeg, milk, yogurt, kidney beans)
130      0.2     (kidney beans, yogurt, onion, milk)
131      0.2   (milk, yogurt, kidney beans, unicorn)
132      0.4   (nutmeg, kidney beans, yogurt, onion)
133      0.2           (nutmeg, milk, yogurt, onion)
```

Frequent 5-itemsets:

```
     support                                      itemsets
134      0.2  (onion, ice cream, kidney beans, corn, eggs)
135      0.2   (unicorn, yogurt, kidney beans, corn, milk)
```

```
136      0.2     (onion, kidney beans, nutmeg, dill, eggs)
137      0.2    (yogurt, kidney beans, nutmeg, dill, eggs)
138      0.2     (onion, yogurt, kidney beans, dill, eggs)
139      0.2           (onion, yogurt, nutmeg, dill, eggs)
140      0.2   (onion, yogurt, kidney beans, nutmeg, dill)
141      0.2     (onion, kidney beans, nutmeg, eggs, milk)
142      0.2    (yogurt, kidney beans, nutmeg, eggs, milk)
143      0.2     (onion, yogurt, kidney beans, eggs, milk)
144      0.4   (onion, yogurt, kidney beans, nutmeg, eggs)
145      0.2           (onion, yogurt, nutmeg, eggs, milk)
146      0.2   (onion, yogurt, kidney beans, nutmeg, milk)
```

After pruning infrequent itemsets:

```
     support                                      itemsets
134      0.2  (onion, ice cream, kidney beans, corn, eggs)
135      0.2   (unicorn, yogurt, kidney beans, corn, milk)
136      0.2     (onion, kidney beans, nutmeg, dill, eggs)
137      0.2    (yogurt, kidney beans, nutmeg, dill, eggs)
138      0.2     (onion, yogurt, kidney beans, dill, eggs)
139      0.2           (onion, yogurt, nutmeg, dill, eggs)
140      0.2   (onion, yogurt, kidney beans, nutmeg, dill)
141      0.2     (onion, kidney beans, nutmeg, eggs, milk)
142      0.2    (yogurt, kidney beans, nutmeg, eggs, milk)
143      0.2     (onion, yogurt, kidney beans, eggs, milk)
144      0.4   (onion, yogurt, kidney beans, nutmeg, eggs)
145      0.2           (onion, yogurt, nutmeg, eggs, milk)
146      0.2   (onion, yogurt, kidney beans, nutmeg, milk)
```

Frequent 6-itemsets:

```
     support                                              itemsets
147      0.2  (onion, yogurt, kidney beans, nutmeg, dill, eggs)
148      0.2  (onion, yogurt, kidney beans, nutmeg, eggs, milk)
```

After pruning infrequent itemsets:

```
     support                                              itemsets
147      0.2  (onion, yogurt, kidney beans, nutmeg, dill, eggs)
148      0.2  (onion, yogurt, kidney beans, nutmeg, eggs, milk)
```

Final Frequent Itemsets:

```
     support                                              itemsets
0        0.2                                             (apple)
1        0.4                                              (corn)
2        0.2                                              (dill)
3        0.8                                              (eggs)
```

```
4        0.2                                          (ice cream)
..       …                                                     …
144      0.4        (onion, yogurt, kidney beans, nutmeg, eggs)
145      0.2                (onion, yogurt, nutmeg, eggs, milk)
146      0.2        (onion, yogurt, kidney beans, nutmeg, milk)
147      0.2  (onion, yogurt, kidney beans, nutmeg, dill, eggs)
148      0.2  (onion, yogurt, kidney beans, nutmeg, eggs, milk)

[149 rows x 2 columns]
```

Final itemssets with 0.2 support

[ ]:
```
current_length_itemsets
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[ ]:
```
     support                                          itemsets
147      0.2  (onion, yogurt, kidney beans, nutmeg, dill, eggs)
148      0.2  (onion, yogurt, kidney beans, nutmeg, eggs, milk)
```

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.

Here are two potential contextual attributes and two potential behavioral attributes that could be incorporated into the algorithm of an identity theft prevention system.

1. **Contextual Attributes:**

   a. **Device Type and Location:** This attribute defines the type of device (e.g., mobile phone, laptop, computer) and the location from which a transaction is initiated. This information serves to identify possible anomalies, such as a purchase made from a device or location not typically associated with the account holder. For instance, if a user regularly conducts transactions from one city and suddenly a transaction originates from a different city, it may signal a potential risk. The rationale behind this attribute is its contextual nature, as it offers insights into the transaction environment rather than the account holder's behavior.

   b. **Transaction Time and Frequency:** This attribute specifies the time of day and the frequency of transactions. An abrupt increase in transaction frequency, particularly at unusual times, may indicate suspicious activity. For example, if a user typically makes online purchases during the daytime, unexpected transactions occurring late at night could raise concerns. The rationale is contextual, as it provides information about the timing and frequency of transactions, which are not directly linked to the account holder's behavior.

2. **Behavioral Attributes:**

    a. **Spending Habits:** This attribute encompasses the account holder's usual spending patterns, including average transaction amounts, preferred merchants, and product categories. Deviations from these patterns might signal potential fraud. The rationale behind this attribute is behavioral, as it pertains to the account holder's typical spending behavior, reflecting individual preferences.

    b. **Purchase History:** This attribute offers a detailed overview of the user's past transactions, including product types, merchants, and transaction amounts. Analyzing this history helps identify unusual purchases that do not align with the account holder's regular spending habits. The rationale is behavioral, as it provides insights into the account holder's historical purchasing decisions, reflecting individual behavior.

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.

One viable approach for detecting outliers in the University of Wisconsin breast cancer dataset involves the application of the Interquartile Range (IQR) method.

The IQR method is commonly utilized to pinpoint outliers by establishing a range within which the majority of data points are anticipated to reside. This method is particularly adept at handling skewed or non-normally distributed data, aligning well with the characteristics of the University of Wisconsin breast cancer dataset.

In the context of this dataset, an outlier denotes a data point that falls outside the expected range of values for a specific feature. These outliers may stem from various factors, including mislabeled data due to errors in collection or labeling, instances of rare or unconventional breast cancer cases, and inaccuracies in data entry.

The IQR method boasts several advantages that render it a suitable choice for outlier detection in this dataset:

1. **Robustness to Outliers:** The IQR method exhibits resilience to outliers, making it more robust than alternatives like the z-score. Extreme values in the data have a limited impact on quartile calculations when employing the IQR method.

2. **Ease of Implementation:** The simplicity and minimal computational requirements of the IQR method make it well-suited for large datasets such as the breast cancer dataset.

3. **Interpretability:** The IQR method facilitates easy interpretation of outliers by identifying them as data points lying outside the customary range.

4. **Non-parametric Nature:** Being non-parametric, the IQR method refrains from making assumptions about the distribution of data. This characteristic proves beneficial for the breast cancer dataset, where some features might not conform to typical patterns.

In summary, the IQR method proves effective in identifying outliers within various features of the University of Wisconsin breast cancer dataset, such as clump thickness, bare nuclei, and mitoses. By

pinpointing these anomalous values, researchers can gain insights into rare cases, detect potential data errors, and comprehend the overall distribution of breast cancer characteristics.

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 a normal distribution for the dataset, we employ the maximum likelihood method to detect outlier values.

By applying the maximum likelihood method, we derive estimates of ˆµ = 22.53 and ˆ = 6.13 from the given data.

The most deviant value in the dataset is 3.25, exhibiting a substantial deviation of 19.28mm from the estimated mean.

In the context of the monthly rainfall data, the application of the maximum likelihood method pinpoints 3.25 as an outlier due to its significant departure from both the mean and standard deviation of the rainfall dataset.

Considering that the region µ ± 3 encompasses 99.7% of data assuming a normal distribution, and noting that 3.25 surpasses the threshold of 3 standard deviations from the mean (19.28/6.13 = 3.14, greater than 3), the data point 3.25 is identified as an outlier.

[ ]:
```
import pandas as pd
import numpy as np

# list of monthly rainfall
monthly_rainfall = [22.93, 20.69, 25.75, 23.84, 25.34, 3.25, 23.55, 28.28, 23.
 ↪72, 22.42, 26.83, 23.82]

# convert list into array
rainfall_arr = np.array(monthly_rainfall)

# Calculating the mean and standard deviation of the rainfall data
mean = np.mean(rainfall_arr)
std_dev = np.std(rainfall_arr)

# threshold for identifying outliers
threshold = 3 * std_dev

# Identifing the outliers outliers
outliers = rainfall_arr[(rainfall_arr < mean - threshold) | (rainfall_arr >␣
 ↪mean + threshold)]
print("The outliers are as follows:", outliers)
```

```
The outliers are as follows: [3.25]
```

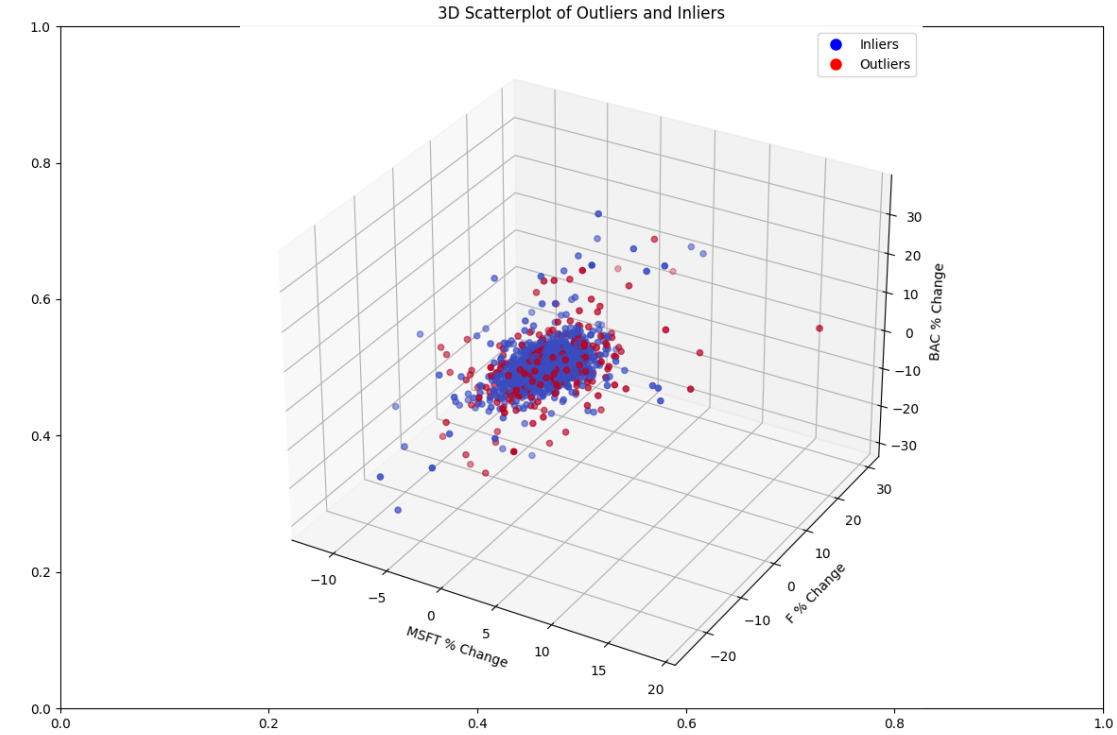
d. [Coding] - Using the stock prices (stocks.csv included in the supplementary material) dataset used in sections 1 and 2 of Week 9 lab, estimate the outliers in the dataset using the one-class SVM classifier approach. As input to the classifier, use the percentage of changes in the daily

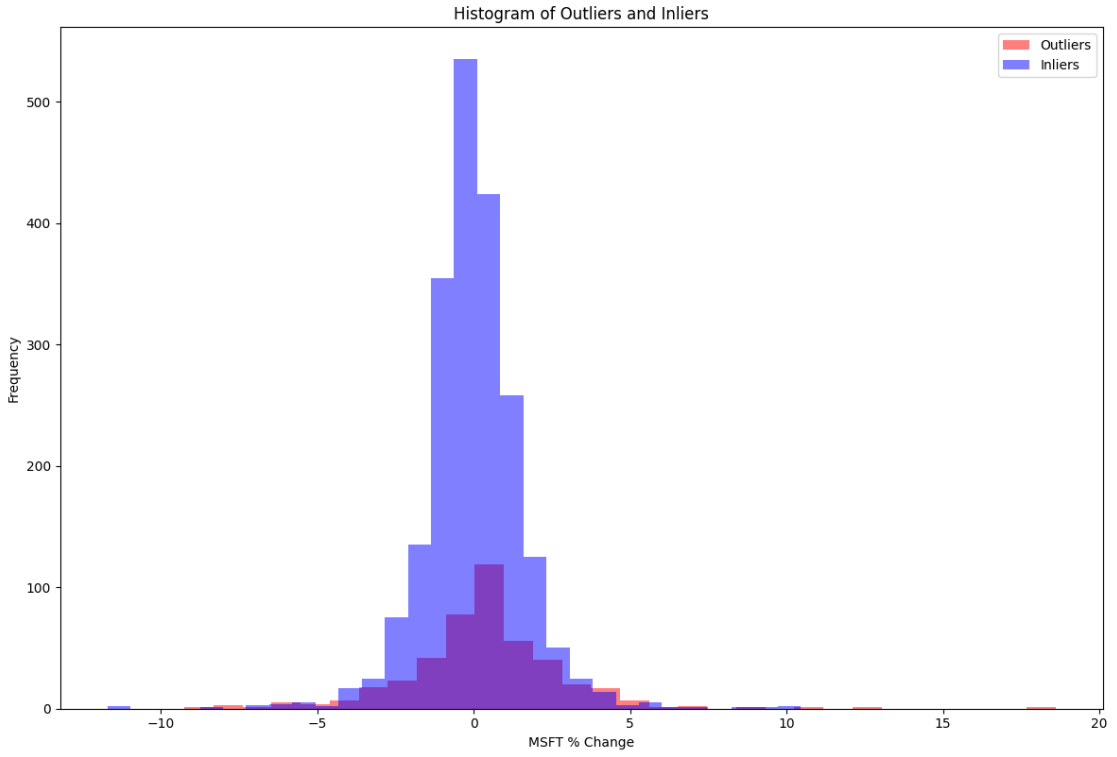
closing price of each stock, as was done in section 1 of the notebook. Use the same SVM settings as in the lab notebook. Plot a 3D scatterplot of the dataset, where each object is color-coded according to whether it is an outlier or an inlier. Also compute a histogram and the frequencies of the estimated 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?

Ans: the one-class SVM approach is more flexible, makes fewer assumptions about data distribution, and is suitable for situations where only normal data is available during training. Parametric methods assume a specific distribution, and proximity-based methods rely on the distance metric and might struggle in high-dimensional spaces. The choice between these methods depends on the characteristics of the data and the assumptions deemed appropriate for the problem at hand.

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import OneClassSVM
from mpl_toolkits.mplot3d import Axes3D

# Assuming 'stocks.csv' contains the stock prices data
# Update the file path accordingly
stocks = pd.read_csv('stocks.csv', parse_dates=['Date'], index_col='Date')

# 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[:
 ↪len(stocks) - 1, :].values,
                                     stocks.iloc[:len(stocks) - 1, :].values),
                     columns=stocks.columns, index=stocks.iloc[1:].index)

# One-Class SVM settings
ee = OneClassSVM(nu=0.01, gamma='auto')
yhat = ee.fit_predict(delta)

# Plot a 3D scatterplot with color-coded outliers and inliers
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(12, 16))

ax1 = fig.add_subplot(211, projection='3d')
ax2 = axes[1]

ax1.scatter(delta['MSFT'], delta['F'], delta['BAC'], c=(yhat == -1),␣
 ↪marker='o', cmap='coolwarm')
ax1.set_xlabel('MSFT % Change')
ax1.set_ylabel('F % Change')
ax1.set_zlabel('BAC % Change')
ax1.set_title('3D Scatterplot of Outliers and Inliers')
```

```
# Display legends
scatter_legend = plt.Line2D([0], [0], marker='o', color='w',␣
 ↪markerfacecolor='blue', markersize=10, label='Inliers')
outlier_legend = plt.Line2D([0], [0], marker='o', color='w',␣
 ↪markerfacecolor='red', markersize=10, label='Outliers')
ax1.legend(handles=[scatter_legend, outlier_legend])

# Plot histograms for outliers and inliers
ax2.hist(delta[yhat == -1]['MSFT'], bins=30, alpha=0.5, color='red',␣
 ↪label='Outliers')
ax2.hist(delta[yhat == 1]['MSFT'], bins=30, alpha=0.5, color='blue',␣
 ↪label='Inliers')
ax2.set_xlabel('MSFT % Change')
ax2.set_ylabel('Frequency')
ax2.set_title('Histogram of Outliers and Inliers')
ax2.legend()

# Show the plots
plt.tight_layout()
plt.show()

# Compute the frequencies of the estimated outlier and inlier labels
unique, counts = np.unique(yhat, return_counts=True)
outlier_freq = counts[0] / len(yhat) * 100
inlier_freq = counts[1] / len(yhat) * 100

# Display frequencies
print(f'Percentage of Outliers: {outlier_freq:.2f}%')
print(f'Percentage of Inliers: {inlier_freq:.2f}%')
```






```
Percentage of Outliers: 17.80%
Percentage of Inliers: 82.20%
```

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.

b. [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?

< !DOCTYPE html >: This initial tag in an HTML document, placed before the < html > tag, informs the browser that the document adheres to the HTML5 standard.

< html >: Serving as the root element of an HTML document, this tag envelops the entire content and encompasses all other HTML elements.

< head >: Containing essential information about the document, such as title and meta tags, this element is pivotal. The title tag specifies the webpage's title, visible in the browser's title bar, while meta tags offer supplementary details like keywords and descriptions.

< body >: Constituting the primary content area of the document, this tag encompasses the visible elements, including text, images, and other components.

< h1 >: Functioning to create top-level headings, the < h1 > tag is the largest heading, followed by < h2 >, < h3 >, and so forth. Heading tags organize the content structure, facilitating user readability.

< p >: Utilized for generating paragraphs, the < p > tag separates blocks of text on the page, providing a coherent structure.

< table >: Serving as the outermost tag for an HTML table, this element encloses all other components, including the table head, table body, and table footer.

< tr >: Dedicated to constructing a table row, the < tr > tag encapsulates all table cells within a row.

< th >: Employed for generating table header cells, the < th > tag typically presents column identifiers in bold text, aiding in the identification of table columns.

< td >: Utilized for creating table data cells, the < td > tag contains the actual content of the table.

< tbody >: Enveloping the table body, the < tbody > tag contains the primary data of the table.

< thead >: Designed to enclose the table header, the < thead > tag houses column labels, providing crucial organization to the table.

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).

[ ]:
```
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

url = "http://eecs.qmul.ac.uk/~emmanouilb/income_table.html"
html = urlopen(url)

soup = BeautifulSoup(html, 'lxml')
print(type(soup))

soup.find_all('td')
```

[ ]:
```
soup.find_all('tr')
```

[46]:
```
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␣
 ↪text without HTML tags
header_list.append(cleantext_header) # Add the clean table header to the list

print(header_list)
```

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

[47]:
```
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 the␣
 ↪text without HTML tags
    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, 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]']
```

[48]:
```
df_header = pd.DataFrame(header_list)
df_header.head()

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

[48]:

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

[49]:
```
df_table = pd.DataFrame(table_list)
df_table2 = df_table[0].str.split(',', expand=True)

df_table2
```

[49]:

	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]

[50]:
```
# 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
```

[50]:

	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
```

[51]:
```
# 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
```

[51]:

	0	1	2	3
0	Region	Age	Income	Online Shopper

[52]:
```
# We concatenate the two dataframes
frames = [df_header2, df_table3]
df = pd.concat(frames)

df
```

[52]:

	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

[53]:
```
df2 = df.rename(columns=df.iloc[0]) # We assign the first row to be the␣
 ↪dataframe header

df2
```

[53]:

	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

[54]:
```
df3 = df2.drop(df2.index[0]) # We drop the replicated header from the first row
↪of the dataframe

df3
```

[54]:

	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

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?

Hub: A webpage containing numerous outbound links to authoritative sources. Hubs are nodes characterized by a high in-degree, signifying that they receive a substantial number of links from other nodes. This suggests frequent citations by other websites, establishing them as significant wellsprings of information. Effective hubs link to multiple reputable authorities.

Authority: A webpage with a substantial number of inbound links. Authorities are nodes distinguished by a high out-degree, indicating they link to numerous other nodes. This implies a depth of knowledge on a specific topic, offering valuable information to their audience. Reputable authority pages are linked to by numerous hubs.

Nodes 1 and 2 boast an in-degree (authority) of 3.

Nodes 3, 4, and 5 exhibit an out-degree (hub) of 2.

Consequently, among the five nodes, nodes 1 and 2 are identified as authorities, while nodes 3, 4, and 5 are classified as hubs.

4A)

1 Original Sentences:

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.

After pre-processing, the documents become:

1. "Data refers characteristic collect observation."
2. "Dataset view collection object."
3. "Data object describe number attribute."
4. "Attribute characteristic feature object."

Now, let's create the Document-Term Matrix (DTM):

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

IDF(Attribute) = log(4/2) log(2) 0.3010

IDF(Characteristic) = log(4/2) log(2) 0.3010

IDF(Collect) = log(4/1) log(4) 0.6021

IDF(Collection) = log(4/1) log(4) 0.6021

IDF(Data) = log(4/2) log(2) 0.3010

IDF(Dataset) = log(4/1) log(4) 0.6021

IDF(Describe) = log(4/1) log(4) 0.6021

IDF(Feature) = log(4/1) log(4) 0.6021

IDF(Number) = log(4/1) log(4) 0.6021

IDF(Object) = log(4/3) log(1.3333) 0.1239

IDF(Observation) = log(4/1) log(4) 0.6021

IDF(refer) = log(4/1) log(4) 0.6021

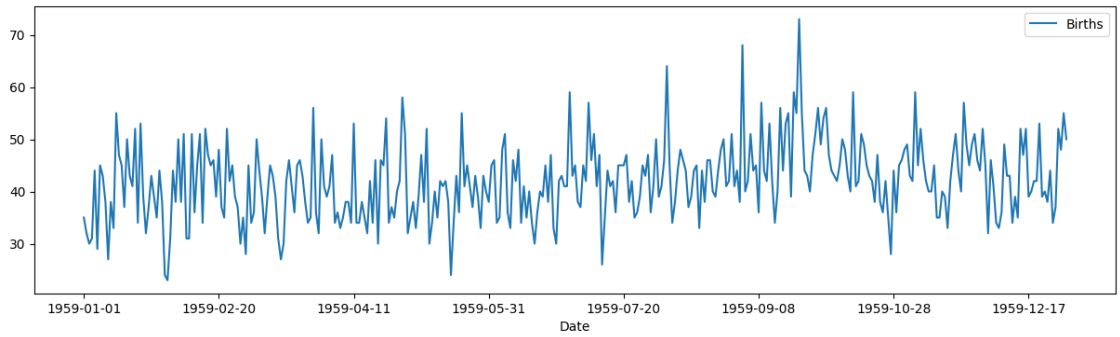
IDF(View) = log(4/1) log(4) 0.6021

4B)

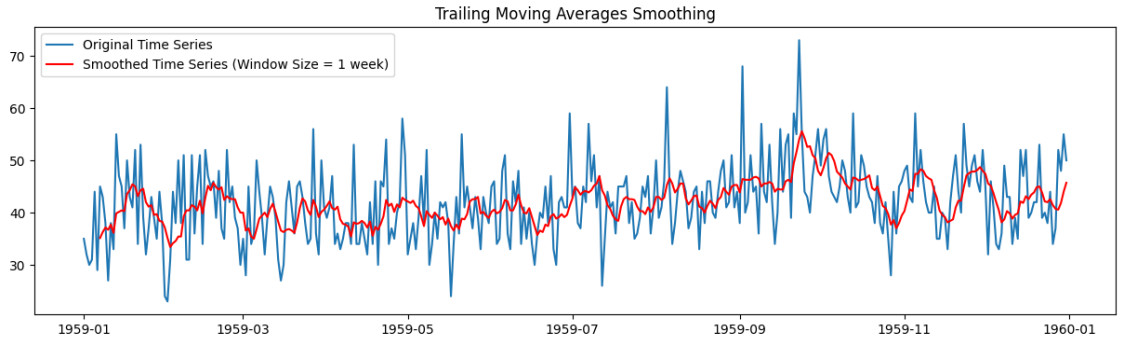
[63]:
```
from pandas import read_csv
import matplotlib.pyplot as plt

series = read_csv('births.csv', header=0, index_col=0)

print(series.head())
series.plot(figsize=(15,4))
```

```
            Births
Date
1959-01-01      35
1959-01-02      32
1959-01-03      30
1959-01-04      31
1959-01-05      44
```

[63]: <Axes: xlabel='Date'>



[64]:
```python
import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'series' is your original time series

# Convert the index to datetime if not already
series.index = pd.to_datetime(series.index)

# Perform trailing moving average smoothing with a 1-week window

rolling_mean = series.rolling(window=7).mean()

# Plot original and transformed dataset
plt.figure(figsize=(15, 4))
plt.plot(series, label='Original Time Series')
plt.plot(rolling_mean, color='red', label=f'Smoothed Time Series (Window Size =␣
 ↪1 week)')
plt.title('Trailing Moving Averages Smoothing')
plt.legend()
plt.show()
```



[65]:
```
rolling_mean.fillna(0,inplace=True)
smoothed_data = rolling_mean
```

[66]:
```
import statsmodels.api as sm
from statsmodels.tsa.ar_model import AutoReg

# AR Model with p=2
# Fit Autoregressive model
model = AutoReg(smoothed_data, lags=2) # "lags" indicates the model order
model_fit = model.fit()

# Make prediction
#yhat = model_fit.predict(len(data), len(data)+3) # arguments denote which␣
 ↪dataset indices to predict
#print(yhat)
ar_forecast = model_fit.predict(start='1960-01-01', end='1960-01-05')
print('AR Model with p=2')
print(ar_forecast)
```

```
AR Model with p=2
1960-01-01    45.380177
1960-01-02    44.960852
1960-01-03    44.590676
1960-01-04    44.271699
1960-01-05    43.997395
Freq: D, dtype: float64
```

[70]:
```
# ARMA Model with p=2, q=2

import warnings

# Suppress all warnings
warnings.filterwarnings("ignore")
```

```
import statsmodels.api as sm
from statsmodels.tsa.arima.model import ARIMA


# ARMA Model with p=2 and q=2
model_arma = ARIMA(smoothed_data, order=(2, 0, 2))  # ARIMA model with p=2,␣
 ↪d=0, q=2
model_fit_arma = model_arma.fit()

# Make prediction
arma_forecast = model_fit_arma.predict(start='1960-01-01', end='1960-01-05')
print('ARMA Model with p=2, q=2')
print(arma_forecast)
```

```
ARMA Model with p=2, q=2
1960-01-01    45.810250
1960-01-02    45.818771
1960-01-03    45.728098
1960-01-04    45.564024
1960-01-05    45.347314
Freq: D, Name: predicted_mean, dtype: float64
```

[71]:
```
plt.figure(figsize=(15, 4))
smoothed_data_100 = smoothed_data.tail(100)
plt.plot(smoothed_data_100, label='Historical Data')
plt.plot(ar_forecast, label='AR Model Forecast', linestyle='--')
plt.plot(arma_forecast, label='ARMA Model Forecast', linestyle='--')
plt.title('Time Series Forecasting')
plt.legend()
plt.show()
```