Capstone Project - Recommendation System

Group 5 - WONG, Shing Fung Dov

Project Objective

This project aims to study and discover the behavior patterns of e-commerce users and visualize the findings. Ultimately, I will develop recommendation systems to suggest products to these users.

Dataset Description

The dataset was downloaded from Kaggle (Retailrocket recommender system dataset). It includes four CSV files: event, category_tree, item_properties_part,1 and part2. The data covers the period from 2015-05-03 03:00 to 2015-09-18 02:59.

event.csv (2756101 entries)	
Columns	**Value**
timestamp	Timestamp in Unix format
visitorid	User ID in integer format
event	Three different values: “view”, “addtocart”, “transaction”
itemid	Product ID in integer format
transactionid	Transaction ID in integer format, only exist when “event” is “transaction”

category_tree.csv (1667 entries)	
Columns	**Value**
categoryid	Category ID in integer format
parentid	Parent ID of “categoryid”, in integer format; not all “categoryid” have a parent

item_properties.csv (10999999 + 9275903 entries)	
Columns	**Value**
timestamp	Timestamp in Unix format
itemid	Product ID in integer format
property	Three types of data in this column : 1. String value: “categoryid”, then the “value” column shows an integer value of categoryid 2. String value: “available”, then the “value” column shows 1 (available) or 0 (unavailable) 3. Time-based property in integer format (unknown meaning)
value	Two other types of data (besides the two mentioned in “property”): 1. Integer value: hashed value from normalized text 2. Float value started with “n”: hashed value from numeric value

	timestamp	itemid	property	value
0	1435460400000	460429	categoryid	1338
1	1441508400000	206783	888	1116713 960601 n277.200
2	1439089200000	395014	400	n552.000 639502 n720.000 424566

Data Preprocessing

No missing values were found except for “transactionid”. After checking, all rows with “transaction” in the “event” column contain a value in the “transactionid” column. Furthermore, duplicated rows were removed, and Unix timestamps were converted to readable timestamps.

```
df_event.isnull().sum()
```

```
timestamp            0
visitorid            0
event                0
itemid               0
transactionid    2733644
```

```
print("Before dropping duplication: ", df_event.shape[0])
df_event.drop_duplicates(inplace=True)
df_event.reset_index(drop=True, inplace=True)
print("After dropping duplication: ", df_event.shape[0])
```

```
Before dropping duplication:  2756101
After dropping duplication:  2755641
```

```
print(df_event["event"].unique())
print(((df_event["event"] == "transaction") & (df_event["transactionid"]).isna()).sum())
```

```
['view' 'addtocart' 'transaction']
0
```

```
df_event["time"] = pd.to_datetime(df_event["timestamp"], unit = "ms")
df_event["time"].head()
```

```
0   2015-06-02 05:02:12.117
1   2015-06-02 05:50:14.164
2   2015-06-02 05:13:19.827
3   2015-06-02 05:12:35.914
4   2015-06-02 05:02:17.106
Name: time, dtype: datetime64[ns]
```

Identification of abnormal users, window shoppers and outliers

It was observed that some users only view products without making any purchases. As user purchases are the ultimate goal, these users may not provide valuable data (As shown on below picture, user ID 892013 has 2023 “views” but only one “addtocart” and no purchases). Although they could be valuable for some analyses or for understanding how to encourage them to buy, I will not address this in this project. Instead, I will identify and remove these users.

```
for id in top20:
    view_time = df_event[(df_event['visitorid'] == id) & (df_event['event'] == "view")].shape[0]
    add_time = df_event[(df_event['visitorid'] == id) & (df_event['event'] == "addtocart")].shape[0]
    tran_time = df_event[(df_event['visitorid'] == id) & (df_event['event'] == "transaction")].shape[0]
    print("visitor with id", id, "have", view_time, "times of view, and","have", add_time, "times of add to cart, and", tran_time, "times of tr
```

```
visitor with id 1150086 have 6479 times of view, and have 719 times of add to cart, and 559 times of transactions.
visitor with id 530559 have 3623 times of view, and have 419 times of add to cart, and 286 times of transactions.
visitor with id 152963 have 2304 times of view, and have 371 times of add to cart, and 349 times of transactions.
visitor with id 895999 have 2368 times of view, and have 56 times of add to cart, and 50 times of transactions.
visitor with id 163561 have 2194 times of view, and have 124 times of add to cart, and 92 times of transactions.
visitor with id 371606 have 2141 times of view, and have 110 times of add to cart, and 94 times of transactions.
visitor with id 286616 have 2057 times of view, and have 120 times of add to cart, and 75 times of transactions.
visitor with id 684514 have 1826 times of view, and have 231 times of add to cart, and 189 times of transactions.
visitor with id 892013 have 2023 times of view, and have 1 times of add to cart, and 0 times of transactions.
visitor with id 861299 have 1573 times of view, and have 230 times of add to cart, and 188 times of transactions.
visitor with id 836635 have 1720 times of view, and have 124 times of add to cart, and 90 times of transactions.
visitor with id 76757 have 1402 times of view, and have 296 times of add to cart, and 185 times of transactions.
visitor with id 638482 have 1828 times of view, and have 1 times of add to cart, and 0 times of transactions.
```

I filtered out users who have no “addtocart” or those who have a ratio of “view” to “addtocart” and “view” to “transaction” below 100 and 500, respectively.

```python
event_counts = df_event.groupby(['visitorid', 'event']).size().unstack(fill_value=0)
event_counts['total_events'] = event_counts.sum(axis=1)

active_users = event_counts[event_counts['total_events'] > 100]

abnormal_list = []

for id in active_users.index:
    view_time = event_counts.loc[id, 'view']
    add_time = event_counts.loc[id, 'addtocart']
    tran_time = event_counts.loc[id, 'transaction']

    if add_time == 0 or (tran_time != 0 and view_time / add_time >= 100 and view_time / tran_time >= 500):
        print(f"Visitor with ID {id} has {view_time} view(s), {add_time} add(s) to cart, and {tran_time} transaction(s).")
        abnormal_list.append(id)

print(f"Identified {len(abnormal_list)} abnormal users/windows shoppers/outliers.")
```

The identified users are shown in the picture below and have been removed from the dataset.

```
Visitor with ID 54791 has 368 view(s), 0 add(s) to cart, and 0 transaction(s).
Visitor with ID 77177 has 118 view(s), 0 add(s) to cart, and 0 transaction(s).
Visitor with ID 78724 has 198 view(s), 0 add(s) to cart, and 0 transaction(s).
Visitor with ID 93504 has 133 view(s), 0 add(s) to cart, and 0 transaction(s).
Visitor with ID 95128 has 114 view(s), 0 add(s) to cart, and 0 transaction(s).
Visitor with ID 97112 has 108 view(s), 0 add(s) to cart, and 0 transaction(s).
Visitor with ID 104829 has 121 view(s), 0 add(s) to cart, and 0 transaction(s).
Visitor with ID 130999 has 249 view(s), 0 add(s) to cart, and 0 transaction(s).
Visitor with ID 148927 has 142 view(s), 0 add(s) to cart, and 0 transaction(s).
Visitor with ID 157844 has 166 view(s), 0 add(s) to cart, and 0 transaction(s).
Visitor with ID 165052 has 553 view(s), 0 add(s) to cart, and 0 transaction(s).
Visitor with ID 169586 has 189 view(s), 0 add(s) to cart, and 0 transaction(s).
Visitor with ID 170277 has 131 view(s), 0 add(s) to cart, and 0 transaction(s).
```

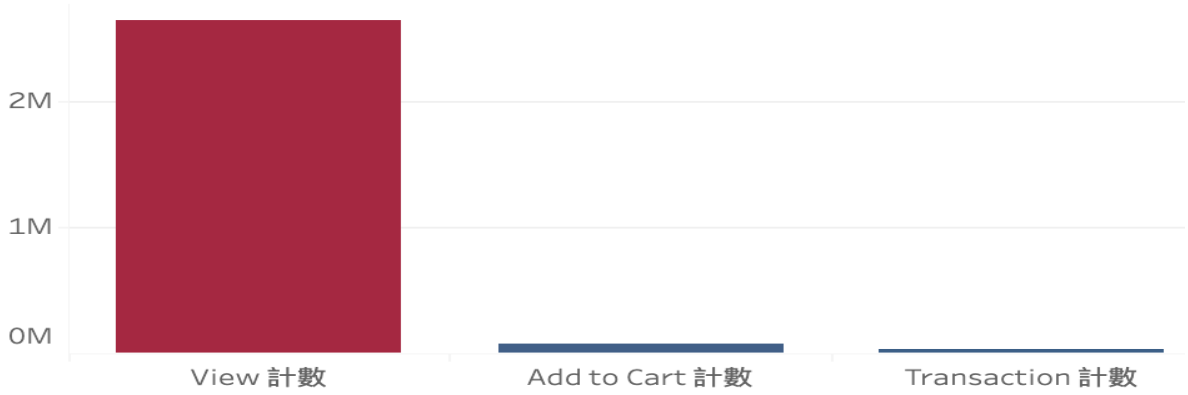
```python
df_event_cleaned = df_event[~df_event['visitorid'].isin(abnormal_list)]
```

Exploratory Data Analysis (EDA)

There are 2,636,027 instances of “view”, 68,962 instances of “addtocart” and 22,454 instances of “transaction”. Only 2.62% of “views” convert to “addtocart,” and 0.85% convert to “transactions,” while 32.56% of “addtocart” convert to “transactions.”

```python
print(df_event_cleaned.groupby("event")["itemid"].count())
print(f"""
    There are {df_event_cleaned['visitorid'].nunique()} distinct users.
    There are {df_event_cleaned['transactionid'].nunique()} distinct transactions.
    There are {df_event_cleaned['itemid'].nunique()} distinct items.
    """)
df_event_cleaned['visitorid'].value_counts().describe().round(2)
```

```
event
addtocart        68962
transaction      22454
view           2636027
Name: itemid, dtype: int64

    There are 1407448 distinct users.
    There are 17669 distinct transactions.
    There are 233262 distinct items.
```



The top five most viewed items, best-selling items, most active users (based on the sum of “view”, “addtocart” and “transaction”), and most active buyers are identified and displayed in the pictures below.

```
Most Viewed Items:
itemid
187946     3411
461686     2971
5411       2330
370653     1854
219512     1798
Name: count, dtype: int64

Most Active Users:
visitorid
1150086     7757
530559      4328
152963      3024
895999      2474
163561      2410
Name: count, dtype: int64
```

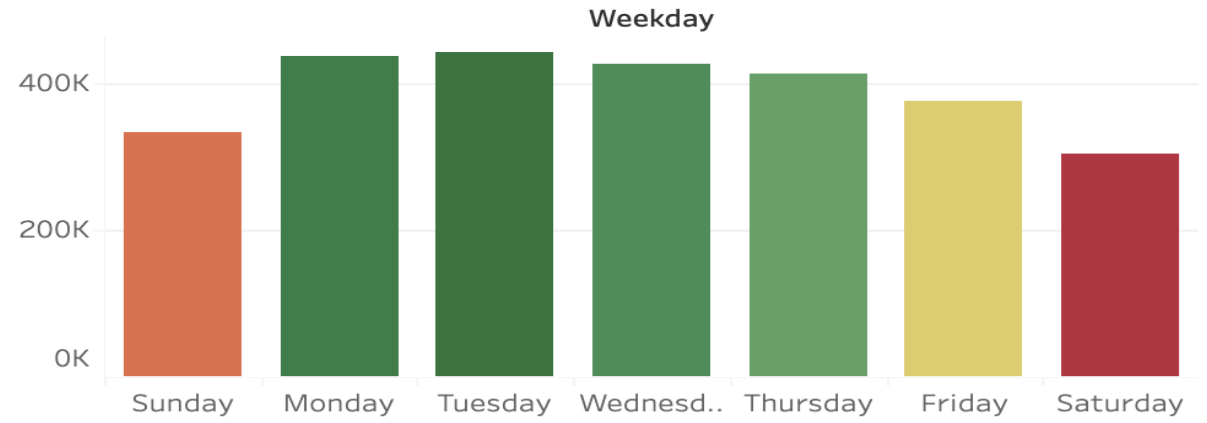
```
Best Sold Items:
itemid
461686     133
119736      97
213834      92
312728      46
7943        46
Name: count, dtype: int64

Most Active Buyers:
visitorid
1150086     559
152963      349
530559      286
684514      189
861299      188
Name: count, dtype: int64
```

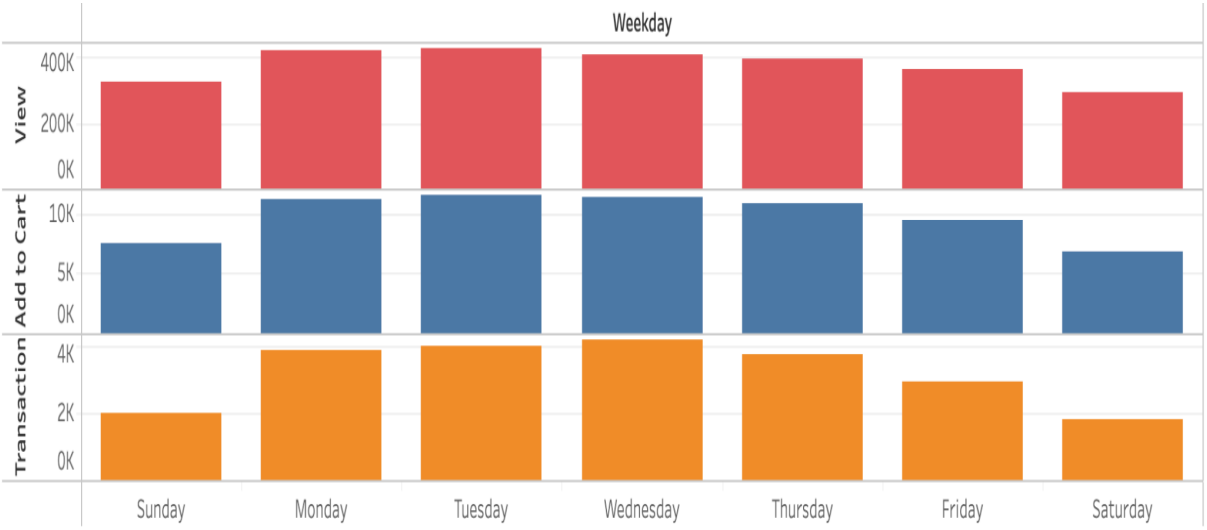
Moreover, the dates and weekdays with the highest activity counts (sum of “view”, “addtocart” and “transaction”) and the count of items sold are also displayed in the pictures below.

```
Date of activities most:
date
2015-07-26    32613
2015-07-27    28336
2015-07-25    27037
2015-05-18    26736
2015-05-19    26566
Name: count, dtype: int64
\Activities of weekday:
weekday
Tuesday      441322
Monday       435451
Wednesday    426172
Thursday     413586
Friday       375076
Sunday       332526
Saturday     303310
Name: count, dtype: int64
```

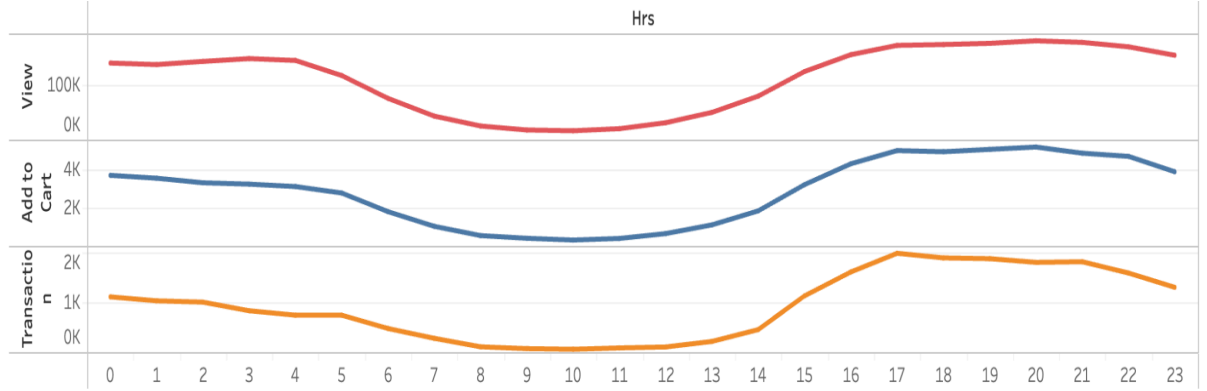
```
Date of sell most:
date
2015-06-16    276
2015-07-28    273
2015-06-17    269
2015-07-27    267
2015-07-22    266
Name: count, dtype: int64

Sale of weekday:
weekday
Wednesday    4150
Tuesday      3973
Monday       3848
Thursday     3749
Friday       2928
Sunday       1995
Saturday     1811
Name: count, dtype: int64
```

From the graphs below, the proportion of overall activity levels and individual activities across weekdays is consistent. Most activities occur on Monday and Tuesday, while weekends show the least activity.





The hourly activity level throughout the day is interesting. Activity levels drop at 21:00 as people begin to sleep, until about 11:00 when people start working. Activity levels increase rapidly after 12:00, suggesting that people stop working after lunch and start viewing and shopping.



Average view count before purchase

Additionally, I attempted to find the average view count before a person buys a product. The methodology is as follows:

1. Broke down the different values in the “event” column into three dataframes: one for “view”, one for “addtocart”, and one for “transaction”, each containing a column for date and time.

```
item_tra = df_event_cleaned[['visitorid', 'itemid', 'time']][df_event_cleaned['event'] == 'transaction']
item_add = df_event_cleaned[['visitorid', 'itemid', 'time']][df_event_cleaned['event'] == 'addtocart']
item_view = df_event_cleaned[['visitorid', 'itemid', 'time']][df_event_cleaned['event'] == 'view']
```

2. Performed inner joins for these three new dataframe using the “transaction” dataframe as a base, creating a new dataframe that include ”visitorid”, “itemid”, time(transaction), time(add to cart), and time(view).

```
df_purchase = item_tra.merge(item_add, how='inner', on=['visitorid', 'itemid'], suffixes=['(transaction)', '(add to cart)'])
df_purchase = df_purchase.merge(item_view, how='inner', on=['visitorid', 'itemid'])
df_purchase = df_purchase.rename(columns={'time':'time(view)'})
df_purchase.head()
```

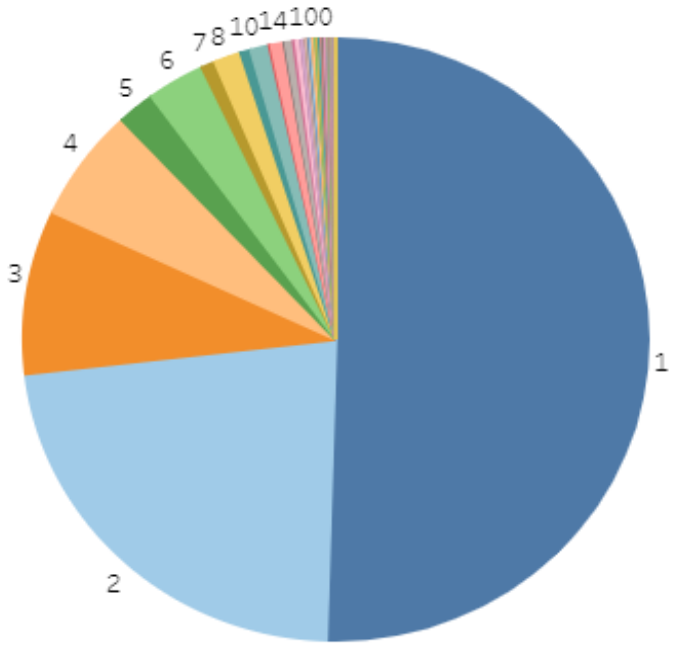
	visitorid	itemid	time(transaction)	time(add to cart)	time(view)
0	599528	356475	2015-06-02 05:17:56.276	2015-06-02 05:12:21.632	2015-06-02 05:11:36.102
1	599528	356475	2015-06-02 05:17:56.276	2015-06-02 05:12:21.632	2015-06-03 02:12:25.235
2	599528	356475	2015-06-02 05:17:56.276	2015-06-02 05:12:21.632	2015-06-06 21:10:10.602
3	599528	356475	2015-06-02 05:17:56.276	2015-06-02 05:12:21.632	2015-06-08 02:44:57.480
4	599528	356475	2015-06-02 05:17:56.276	2015-06-02 05:12:21.632	2015-06-10 00:34:37.794

3. Dropped out the rows where the time of “view” occurs before the time of “addtocart” and the time of “addtocart” occurs before the time of “transaction”. The cleaned dataframe contains all times of “addtocat” and “view” for each transaction.

```
print("Before: ", df_purchase.shape[0])
purchase_record = (df_purchase['time(transaction)'] - df_purchase['time(view)']) > np.timedelta64(0, 's')
tmp = df_purchase[purchase_record]
purchase_record2 = (tmp['time(transaction)'] - tmp['time(add to cart)']) > np.timedelta64(0, 's')
tmp = tmp[purchase_record2]
print("Final: ", tmp.shape[0])
tmp
```

```
Before:  75932
Final:  49424
```

4. Grouped by ‘visitorid’ and ‘itemid’ to gather the information. The result shows that the average view count before purchase is 2.89, meaning that users typically view a product about three times before deciding to buy.



The distribution of average view count before purchase is shown in the pie char on the left. Over half of users will buy the item instantly. About 20% of users will buy after viewing two times.

Average view time before purchase

In addition to the average view count, I also examined the average view time before purchase. I divided the transaction data into two groups, instant purchase (buying after a single view) and non-instant purchase (buying after multiple views).

For instant purchases, the average time from view to transaction is about two hours. For non-instant purchases, the average time from the first view to transaction is about two days and three hours, and the average time from last view to transaction is approximately three and a half hours.

```
def printTime(df, word1, word2):
    time_diff = df['time(transaction)'] - df['time(view)']
    avg_time = time_diff.mean()
    days = avg_time.days
    totalsec = avg_time.seconds
    hrs, tmp = divmod(totalsec, 3600)
    mins, sec = divmod(tmp, 60)
    print('For %s, Average Time from %s to transaction: %s days %s hours %s minutes %s seconds.' % (word1, word2, days, hrs, mins, sec))

# Time for instant purchase
printTime(single_view, "Instant Puchase", "Last View")

# Time for non-instant purchase

# Sort the dataframe
multi_view = multi_view.sort_values(['visitorid', 'itemid', 'time(view)'])
multi_view = multi_view.reset_index(drop=True)
first_view = multi_view[~(multi_view.duplicated(subset=['visitorid', 'itemid'], keep='first'))]

last = multi_view[~(multi_view.duplicated(subset=['visitorid', 'itemid'], keep='last'))]
printTime(first_view, "Non-Instant Puchase", "First View")
printTime(last, "Non-Instant Puchase", "Last View")
```

```
For Instant Puchase, Average Time from Last View to transaction: 0 days 2 hours 0 minutes 3 seconds.
For Non-Instant Puchase, Average Time from First View to transaction: 2 days 3 hours 9 minutes 32 seconds.
For Non-Instant Puchase, Average Time from Last View to transaction: 0 days 3 hours 32 minutes 4 seconds.
```

Recommendation System

Content Based Filtering

In this dataset, certain item features can be utilized for content based filtering. In this project, I decided to use “property”, “value”, “categoryid”, and “parentid” as features.

1. Performed Data Preprocessing, including dropping rows where “property” is “available”, dropping the “timestamp”, and removing duplicates.
2. Initially, I intended to use “categoryid” as a feature. However, after checking, I found that about 20% of items lack a “categoryid” in the dataset so I could not use it as a feature and dropped it as well.

```
categoryid_items = df_prop_CBR[df_prop_CBR['property'] == 'categoryid']['itemid'].unique()
items_event = df_event_cleaned['itemid'].drop_duplicates()
matching_items = items_event[items_event.isin(categoryid_items)]

print("No. of items in distunct event:", len(items_event))
print("No. of items in event that have category id:", len(matching_items))
print("No. of items in event that have not category id:", (len(items_event) - len(matching_items)))
```

✓ 0.0s

```
No. of items in distunct event: 233262
No. of items in event that have category id: 183922
No. of items in event that have not category id: 49340
```

3. Aggregated the value of “property” and “value” for the same items.

itemid	property	value
0	112 283 227 6 1056 225 189 6 776 917 888 364 1...	679677 66094 372274 478989 1152934 1238769 115...
1	296 59 813 33 790 790 790 790 185 764 839 284 ...	866110 769062 814966 1128577 1000087 421694 n5...
2	282 332 159 283 443 790 641 877	n192.000 145688 n72.000 519769 822092 325894 5...
3	159 678 1080 283 250 227 689 562 283 459 888 9...	519769 327918 769062 138228 150169 1182824 327...
4	115 897 28 839 202 698 689 764 678 776 888	n24.000 324209 150169 176547 508476 371058 714...

4. Checked for any null values to ensure that all items have “property” and “value”.
5. Due to a RAM error while processing further, I had to reduce the scale. I sampled 10% of the data and set limitations for vectorization. Additionally, since there were more values in “value” compared to “property”, I dropped the “property” column.

```
MemoryError: Unable to allocate 2.68 TiB for an array with shape (417053, 882157
```

```
sample_size = max(1, len(df_prop_CBR2) // 10)
df_prop_CBR2_sampled = df_prop_CBR2.sample(n = sample_size, random_state = 37)
```

6. Vectorized the values in the “value” column.

```
# Limited input
tfidf = TfidfVectorizer(min_df = 500, max_df = 0.7)
value_tfidf = tfidf.fit_transform(df_prop_CBR2_sampled['value'])

# Since property vary over time and we have no ram...so I decide not to use property
final_df_CBR = pd.concat([df_prop_CBR2_sampled[['itemid']], pd.DataFrame(value_tfidf.toarray(), columns=tfidf.get_feature_names_out())], axis=1)
```

7. Implemented the Cosine Similarity Algorithm to calculate the similarity of those items based on the vectorized values.

```
CBR_result = cosine_similarity(tfidf_df.iloc[:, 1:])
CBR_result
```

```
array([[1.        , 0.26479574, 0.06290316, ..., 0.0108911 , 0.29338723,
        0.33007943],
       [0.26479574, 1.        , 0.18072076, ..., 0.11674878, 0.29691512,
        0.32257883],
       [0.06290316, 0.18072076, 1.        , ..., 0.04084371, 0.06896945,
        0.22644518],
       ...,
       [0.0108911 , 0.11674878, 0.04084371, ..., 1.        , 0.0143297 ,
        0.02665029],
       [0.29338723, 0.29691512, 0.06896945, ..., 0.0143297 , 1.        ,
        0.36980864],
       [0.33007943, 0.32257883, 0.22644518, ..., 0.02665029, 0.36980864,
        1.        ]])
```

The final step was to validate the results. However, all values in this dataset are unexplainable, making it difficult to validate the results intuitively.

One indirect method is to compare the similarity of items with the same parent, as they should share similar features. However, the result showed low similarities, suggesting that my approach may have failed.

```
parent_counts = df_cat.groupby('parentid')['categoryid'].count()
valid_parents = parent_counts[parent_counts >= 8].index
category_lists = [df_cat[df_cat['parentid'] == parentid]['categoryid'].tolist() for parentid in valid_parents]


similar_items_with_scores = []

for category_list in category_lists:
    found_items = df_prop_CBR2_sampled[df_prop_CBR2_sampled['itemid'].isin(category_list)]['itemid'].tolist()

    if len(found_items) >= 2:
        indices = [df_prop_CBR2_sampled[df_prop_CBR2_sampled['itemid'] == item].index[0] for item in found_items]

        extracted_sim = CBR_result[indices, :][:, indices]

        similar_items_with_scores.append((found_items, extracted_sim))

print("Similar Items Found (with 2 or more itemids) and their Cosine Similarity:")
for items, scores in similar_items_with_scores:
    print(f"Items: {items}")
    print(f"Cosine Similarity Matrix:\n{scores}\n")
```

```
Cosine Similarity Matrix:
[[1.         0.08357705]
 [0.08357705 1.        ]]

Items: [563, 1642, 1411]
Cosine Similarity Matrix:
[[1.        0.        0.       ]
 [0.        1.        0.0949806]
 [0.        0.0949806 1.       ]]

Items: [1420, 678]
Cosine Similarity Matrix:
[[1.         0.08145747]
 [0.08145747 1.        ]]
```

The values are lower than 0.1, which mean they are not similar.

Collaborative Filtering

Since the Content Based Filtering approach did not succeed, I attempted another method: Collaborative Filtering. Due to computing resource limitations encountered during Content Based Filtering, I selected a simplefeature for Collaborative Filtering, which is daily activity levels throughout the week.

1. Dropped all unnecessary columns.

```
# Drop useless columns
df_event_Col = df_event_cleaned.drop(columns=['timestamp', 'event', 'itemid', 'transactionid', 'time', 'date', 'hrs'])
df_event_Col
```

2. Calculated the total activity count and the proportion of activity counts for weekdays.

```
total_counts = df_event_Col.groupby('visitorid')['weekday'].count()
weekday_counts = df_event_Col.groupby(['visitorid', 'weekday']).size().unstack(fill_value=0)
weekday_proportions = weekday_counts.div(total_counts, axis=0)
df_event_Col = df_event_Col.merge(weekday_proportions, on='visitorid', how='left', suffixes=('', '_proportion'))
weekday_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
proportion_columns = [day for day in weekday_order if day in weekday_proportions.columns]
df_event_Col2 = df_event_Col[['visitorid'] + proportion_columns].drop_duplicates()
```

	visitorid	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
0	257597	0.500000	0.500000	0.000000	0.0	0.000000	0.000000	0.000000
1	992329	0.266667	0.133333	0.066667	0.3	0.166667	0.033333	0.033333
2	111016	0.000000	0.500000	0.000000	0.0	0.000000	0.000000	0.500000
3	483717	0.000000	1.000000	0.000000	0.0	0.000000	0.000000	0.000000
4	951259	0.000000	1.000000	0.000000	0.0	0.000000	0.000000	0.000000

3. Performed sampling to use 1% of the data (I tested several times and found that using 1% was manageable for my laptop and Google Cola).
4. Implemented MinMAxScaler to normalize the values of "Activity Count" to a range between 0 and 1 for the Cosine Similarity Algorithm.
5. Implemented the Cosine Similarity Algorithm to calculate the similarity of users based on their daily activities patterns.

The result was not satisfactory, as many values were either 0 and 1 (as shown in the first picture). Upon investigation, this was due to many users having only one activity. To address this issue, I removed users with an activity count of less than 30, and the results improved (as shown in the second picture).

visitorid	64865	857996	979037	467967	1033321	1220775	1119204	30129	16074	310227	...	635437	949037	1247105	676663	169835
visitorid																
64865	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.000000	0.0	...	0.000000	0.000000	0.707102	0.0	0.000000
857996	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000019	0.0	...	0.999997	0.999997	0.000009	0.0	0.413710
979037	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.000000	0.000000	0.000000	1.0	0.901993
467967	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.000000	0.0	...	0.000000	0.000000	0.707102	0.0	0.000000
1033321	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.000000	0.0	...	0.000000	0.000000	0.707102	0.0	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1264138	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.999972	0.0	...	0.000000	0.000000	0.707102	0.0	0.000000
587756	1.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.000000	0.0	...	0.000000	0.000000	0.707102	0.0	0.000000
1316221	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.999972	0.0	...	0.000000	0.000000	0.707102	0.0	0.000000
690564	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	1.0	...	0.000000	0.000000	0.000000	0.0	0.037583
643891	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.999972	0.0	...	0.000000	0.000000	0.707102	0.0	0.000000

visitorid	992329	712443	492414	85734	820159	1185234	121688	73449	286616	1141573	...	1080047	321766
visitorid													
992329	1.000000	0.532854	0.714299	0.832385	0.577350	0.804775	0.577350	0.885796	0.753800	0.871018	...	0.218002	0.288675
712443	0.532854	1.000000	0.904081	0.633850	0.499363	0.683658	0.499363	0.457995	0.849922	0.189645	...	0.124758	0.677708
492414	0.714299	0.904081	1.000000	0.733547	0.774597	0.755925	0.774597	0.557257	0.897879	0.327183	...	0.129991	0.516398
85734	0.832385	0.633850	0.733547	1.000000	0.443321	0.970222	0.443321	0.921680	0.763006	0.588785	...	0.581295	0.554527
820159	0.577350	0.499363	0.774597	0.443321	1.000000	0.477827	1.000000	0.337770	0.528020	0.292425	...	0.000000	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...
272883	0.177413	0.071244	0.088157	0.530364	0.000000	0.525991	0.000000	0.549237	0.133747	0.153849	...	0.996428	0.000000
1296849	0.658089	0.033189	0.084087	0.550168	0.000000	0.568824	0.000000	0.826627	0.307071	0.880604	...	0.532561	0.000000
1186695	0.649519	0.000000	0.043033	0.333618	0.000000	0.370316	0.000000	0.650703	0.282079	0.930443	...	0.079493	0.000000
1271070	0.072169	0.000000	0.129099	0.003006	0.000000	0.000000	0.000000	0.000000	0.376440	0.000000	...	0.000000	0.000000
533590	0.360844	0.000000	0.129099	0.347143	0.000000	0.113484	0.000000	0.238426	0.046177	0.186089	...	0.000000	0.000000

To validate the results, I compare a user with "visitorid" of 992329 to users 492414 (similarity 0.714) and 85734 (similarity 0.832), as well as users 1080047 (similarity 0.218) and 321766 (similarity 0.289).

Comparing the results of the high-similarity group (first picture) and low-similarity group (second picture), it appears that users in the high-similarity group share similar activity patterns.

```
# Highly similar
df_event_Col3[df_event_Col3['visitorid'].isin([992329, 492414, 85734])]
```

	visitorid	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Activity Count
1	992329	0.266667	0.133333	0.066667	0.300000	0.166667	0.033333	0.033333	30
106	492414	0.391304	0.260870	0.043478	0.021739	0.065217	0.065217	0.152174	46
174	85734	0.201916	0.252567	0.234086	0.151951	0.158111	0.001369	0.000000	1461

	visitorid	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Activity Count
1	992329	0.266667	0.133333	0.066667	0.300000	0.166667	0.033333	0.033333	30
1367805	1080047	0.000000	0.068966	0.862069	0.068966	0.000000	0.000000	0.000000	58
1367854	321766	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	33

In the end, I implemented a simple recommendation system based on Collaborative Filtering.

This system generates a list of users in descending order of similarity to the target user. It then extracts items purchased by those similar users and checks if the target user has already bought those items. If not, the system recommends the item and extracts another item from the next user until ten items are recommended.

```
# Simple Recommendation System

target_user_id = 992329
similarity_scores = df_similarity3[target_user_id]
similar_users = similarity_scores[similarity_scores > 0.5]
sorted_similar_users = similar_users.sort_values(ascending=False)

# drop first row since it will be user himself
sorted_similar_users = sorted_similar_users.iloc[1:]
sorted_similar_users.head()
```

✓ 0.0s 開啟 Data Wrangler 中的 'sorted_similar_users'

```
visitorid
311205     0.977726
1284779    0.975541
1080579    0.974428
1175225    0.974158
895999     0.966107
```

```
recommendation_list = []

target_user_items = set(tmp[tmp['visitorid'] == target_user_id]['itemid'])

for user_id in sorted_similar_users.index:
    # Check is this user buy something
    if user_id in tmp['visitorid'].values:
        user_items = tmp[tmp['visitorid'] == user_id]['itemid'].tolist()

        for item in user_items:
            # Not recommend this item if target user bought this item
            if item not in target_user_items:
                recommendation_list.append(item)
                break

    # recommend 10 items only
    if len(recommendation_list) >= 10:
        break

print("Recommended items:", recommendation_list)
```

✓ 0.0s

```
Recommended items: [437490, 40484, 227311, 8786, 349318, 420238, 396732, 372188, 352788, 294242]
```

Challenges and Limitations

1. **Unexplainable values**: The values of "itemid," "categoryid," "parentid," "property," and "value" cannot be directly interpreted. This makes it impossible to use and analyze them effectively. If these values were interpretable, I could extract more useful data such as price, color, discount, etc., for further processing. Additionally, it is difficult to use these values to validate my analysis results.
2. **Missing Category ID**: There are too many missing category IDs, preventing their use in analysis. Category ID can serve as a strong feature and should relate directly to the properties and content of the item.
3. **Potentially Inappropriate Aggregation**: Since the value of "property" varies over time and the value of "value" cannot be interpreted, simple aggregation may not be the best approach.
4. **Limitation of Computing Resources and Knowledge**: It is challenging to conduct larger-scale and more complex analyses.
5. **Short Coverage Period**: This dataset covers only 4.5 months, which is insufficient to present user activity trends over the time.

Future Work

1. **Alternative Dataset**: It would be beneficial to find another dataset that contains interpretable values and covers a longer period.
2. **Inclusion of More Features**: Including more features, such as item popularity and user-item interactions, should improve the accuracy of similarity calculations.
3. **Implementation of a Comprehensive Recommendation System**: I plan to develop a comprehensive recommendation system that integrates various algorithms, combining both content-based and collaborative filtering methods.

Conclusion

The analysis found that most users view and purchase products online primarily on Mondays and Tuesdays, with surprisingly little activity during weekends. Furthermore, the activity level remains high from 17:00 until 04:00.

On average, users view items about three times before making a purchase. For instant purchases, it takes about two hours from view to transaction on average. For non-instant purchases, it takes approximately two days and three hours from the first view to transaction, and about three and a half hours from the last view to transaction.

In the recommendation system section, the aggregation of “value” for developing content-based filtering did not succeed. In contrast, collaborative filtering effectively identified similar users based on behavior patterns.

Reference

https://www.kaggle.com/code/kanncaa1/recommendation-systems-tutorial
https://www.kaggle.com/code/python4sp/rs-content-based-collaborative-filtering
https://www.kaggle.com/datasets/retailrocket/ecommerce-dataset

Github

https://www.kaggle.com/datasets/retailrocket/ecommerce-dataset/data

Dataset

https://www.kaggle.com/datasets/retailrocket/ecommerce-dataset/data