Instacart Recommender

Recommendation for Groceries

Our Problem Statement

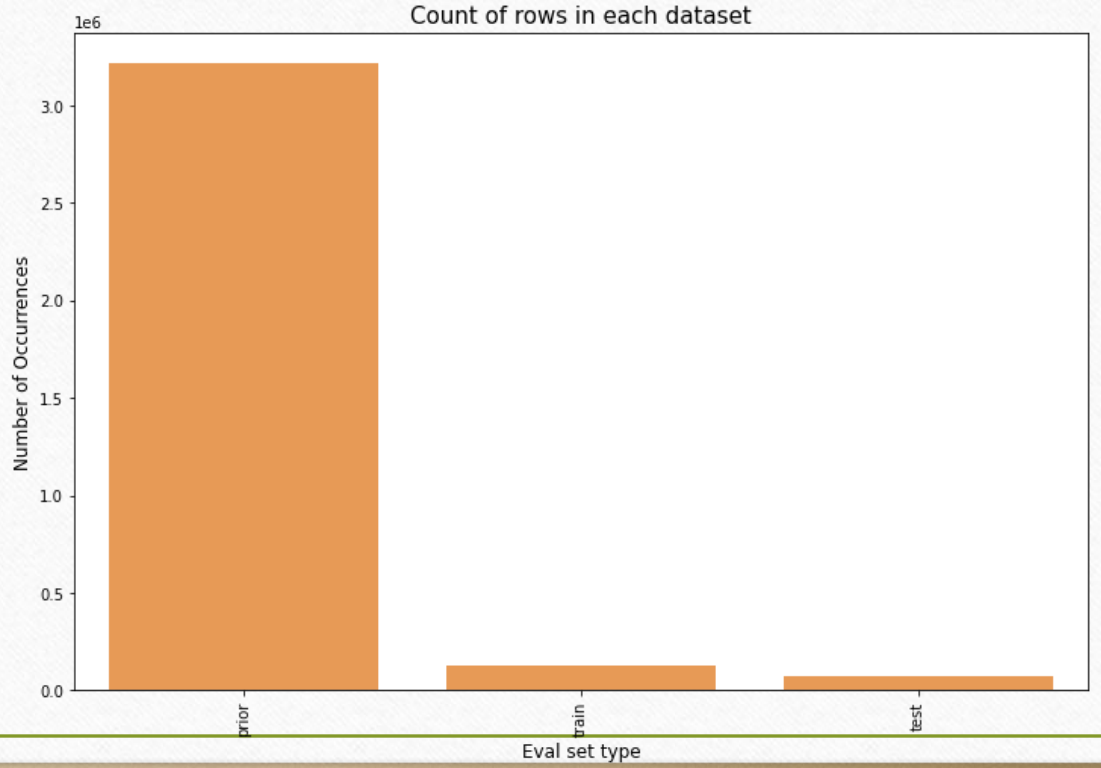
- As part of a team of data analysts at Instacart, a grocery delivery and pick-up service in the US and Canada, we would like to recommend **which products will be in a user's next order,** based on their prior orders.

Exploratory Data Analysis (EDA)

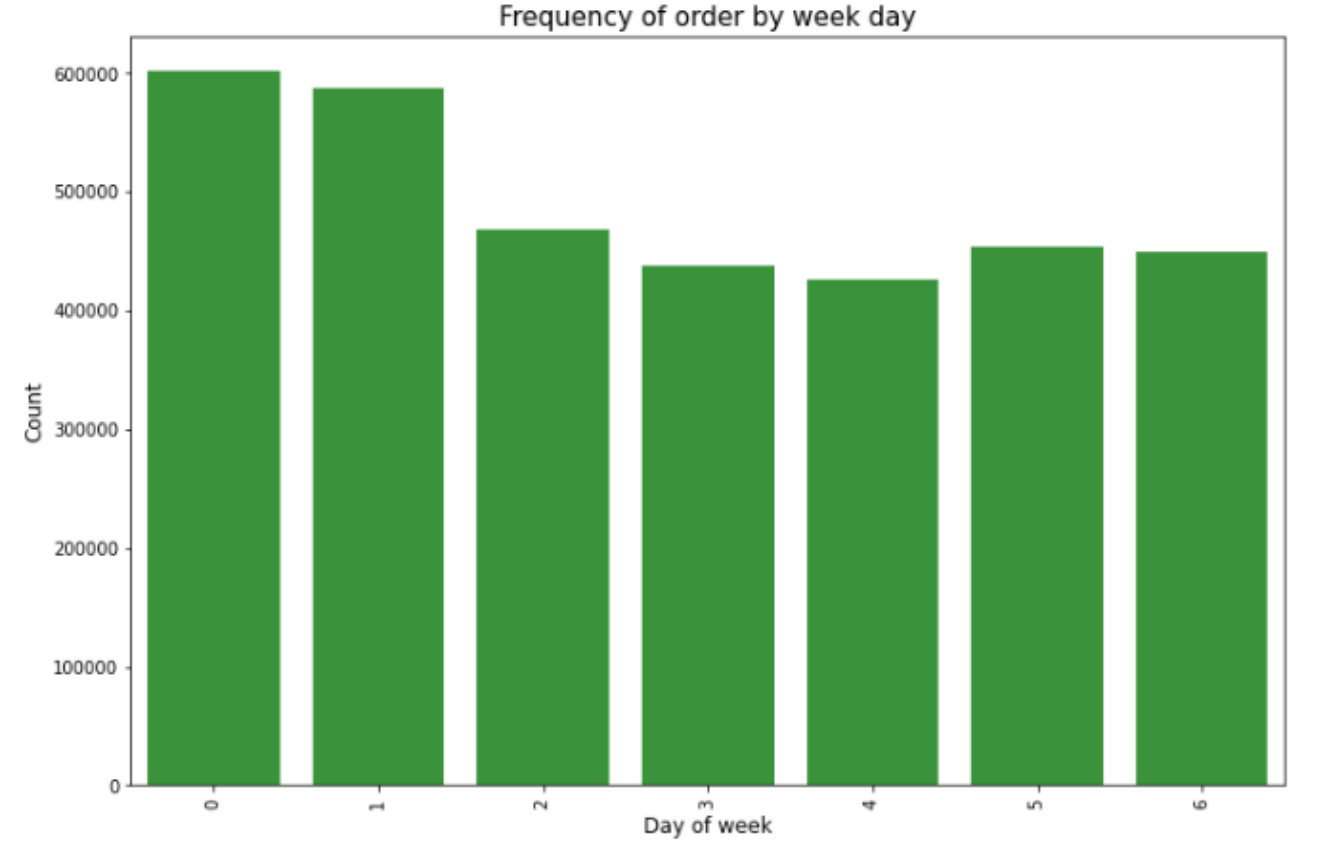
- 6 datasets
  - Aisles.csv: Type of food (fresh pasta, prepared soups etc.)
  - Department.csv: Broad categories they are in (frozen food, bakes etc.)
  - Product.csv: Contains all the products, with names and associated aisle and department
  - Prior.csv: All prior orders, on individual item level
  - Train.csv: Most recent order, on individual item level
  - Orders.csv: All orders, but on order level (have to match to prior or train)

EDA

- There are 206209 unique users in the dataset
- Each user has multiple purchases and orders (minimum of 4, max 100)
- Their last purchase is either a train or test set as intended by Instacart (131209 and **75000** respectively)

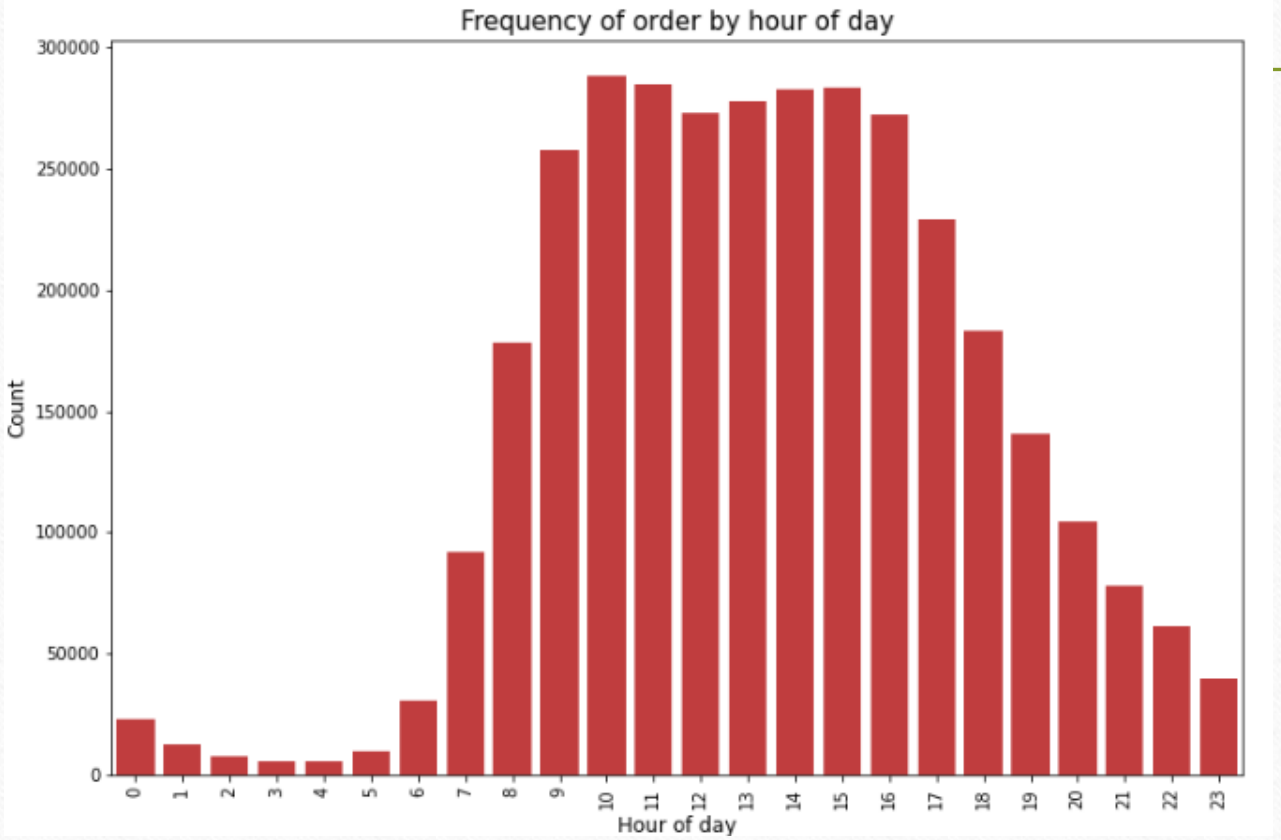


EDA



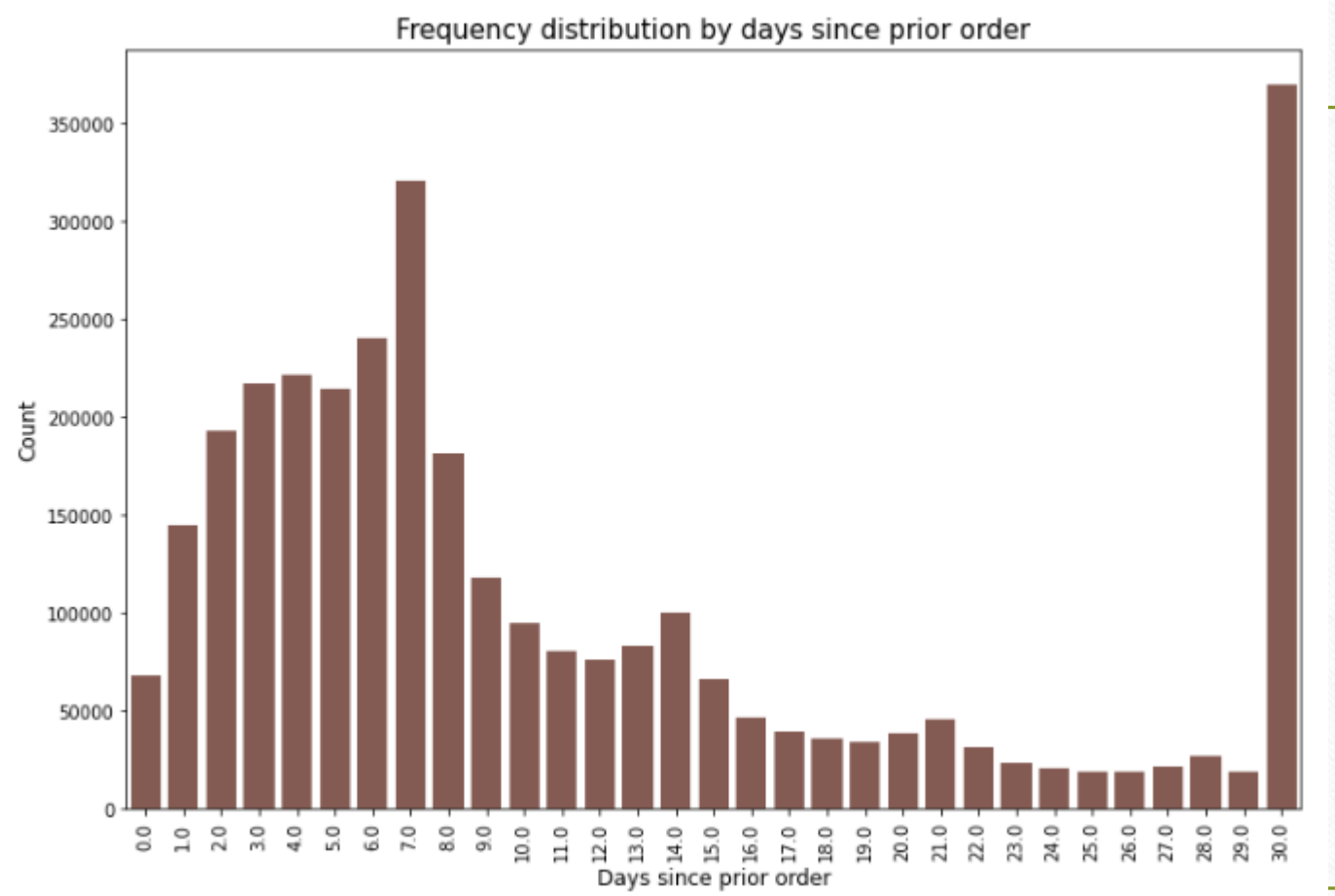
- Seems like 0 and 1 correspond to Saturday and Sunday
  Wednesday (4) seems to be when it performs downwards the most.

EDA

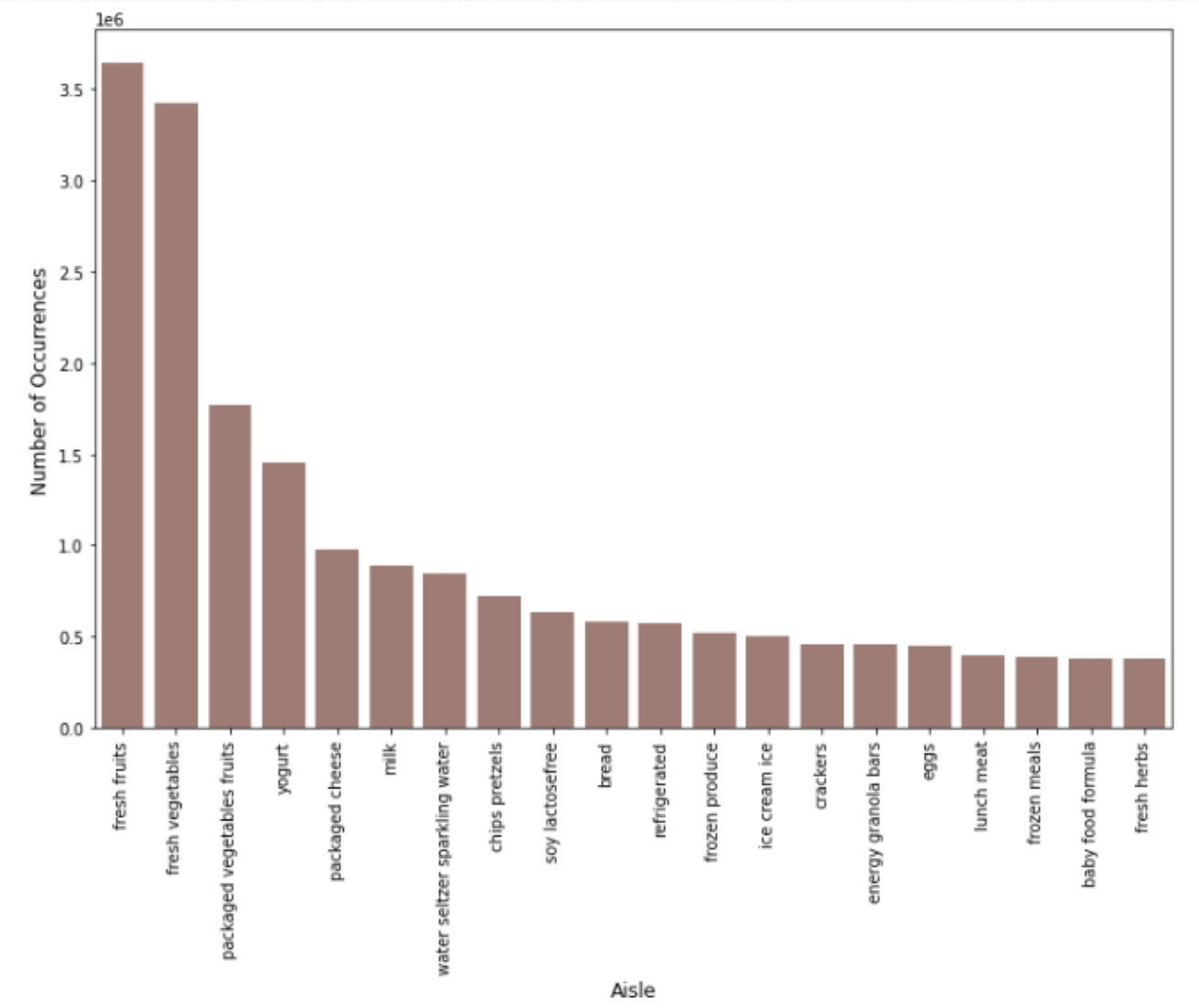


- Most of the orders are done during day time, between 9am and 5pm.

EDA



- Trend where customers order on the 7th day
  - Weekly habit
- There are also peaks at 14 and 21 days too –
  - Customers who bought on weekends might only do so then, and increase the chances of it being in multiples of 7.
- Huge peak for 30 days
  - Could be monthly grocery shopping.
  - Monthly recurrent orders



EDA

- Fresh fruits
- Fresh vegetables
- Packaged vegetables and fruits
- Yoghurt
- Packaged cheese…

	product_name	frequency_count
0	Banana	472565
1	Bag of Organic Bananas	379450
2	Organic Strawberries	264683
3	Organic Baby Spinach	241921
4	Organic Hass Avocado	213584
5	Organic Avocado	176815
6	Large Lemon	152657
7	Strawberries	142951
8	Limes	140627
9	Organic Whole Milk	137905
10	Organic Raspberries	137057
11	Organic Yellow Onion	113426
12	Organic Garlic	109778
13	Organic Zucchini	104823
14	Organic Blueberries	100060
15	Cucumber Kirby	97315
16	Organic Fuji Apple	89632
17	Organic Lemon	87746
18	Apple Honeycrisp Organic	85020
19	Organic Grape Tomatoes	84255

EDA

- Fruits seem to dominate the list
  - Bananas as the first 2 highest counts
- Organic produce are highly favoured by Instacart users
  - Half of the top 20 list

EDA

- On average, more than half the items or about **59%** of them are re-ordered items

```
# percentage of re-orders in prior set
prior.reordered.sum() / prior.shape[0]
```

```
0.5896974667922161
```

```
# percentage of re-orders in train set
train.reordered.sum() / train.shape[0]
```

```
0.5985944127509629
```

Evaluation / Metrics

- Space-delimited list as submission
- F1-score as metric
  - Another way of measuring accuracy
  - True positive is important (numerator)
  - Reducing false positives (false hits) and false negatives (missed hits) are also important

order_id	products
2774568	38596 40604
329954	25146 43704
1528013	38293 40992
1376945	30480 34551
1356845	44422 31506
2161313	37710 27839
1416320	41950 44359
1735923	42913 34690
1980631	22362 13914
139655	27845 24964
1411408	26452 37119
2940603	44632 30592

$$\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}$$

$$\text{Precision} = \frac{tp}{tp + fp}$$

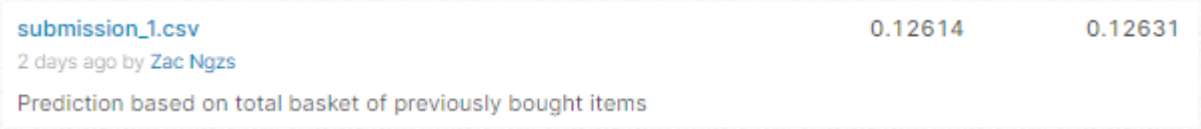
$$\text{Recall} = \frac{tp}{tp + fn}$$

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{\text{tp}}{\text{tp} + \frac{1}{2}(\text{fp} + \text{fn})}.$$

Base model: Giving them everything

- Given that most products were reordered products
  - More than 87% of the data within the prior dataset contains re-ordered data
  - Kaggle score of **0.126**

	order_id	products
33819106	2774568	38596 21903 248 40604 8021 17668 21137 23650 3...
33819107	329954	22199 25146 1200 17769 43704 37646 11865 35469...
33819108	1528013	38293 20323 40992 21903 45007 11068 10644 4940...
33819109	1376945	8230 30480 22950 34551 44632 15261 33037 20383...
33819110	1356845	13176 14992 44422 11520 31506 22959 7120 37687...



Base model +1: Top purchases / customer

- Only the top 50% by value counts of each customer's basket will be considered
- F1 Score dropped from 0.126 to **0.119**
  - Even when we halved the list of recommended items, F1-scored remained relatively similar
  - TP (numerator) dropped, but FP and FN dropped too (denominator)

```
list = []
for i in data_test_set["user_id"].unique():
    df= all_data[all_data["user_id"]==i]
    lowest = df.sort_values('product_id',ascending=False).head(int(df.shape[0]*.5))
    for i in lowest["index"]:
        list.append(i)

bought_50 = all_data.loc[all_data["index"].isin(list)]
```

$$F_1 = \frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} = 2\frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{tp}}{\mathrm{tp} + \frac{1}{2}(\mathrm{fp} + \mathrm{fn})}.$$

Collaborative filtering

Collaborative filtering

"Collaborative filtering is **a technique that can filter out items that a user might like on the basis of reactions by similar users**. It works by searching a large group of people and finding a smaller set of users with tastes similar to a particular user."

- Caveats for grocery recommendations
  - Consumers generally are not ranking these items
  - We are recommending based on the number of times they have bought it before to recommend it subsequently.

User-based: Processing

32_434_489 rows of data → needs to be reduced

Manipulating the **non-test set only:**

- Drop products that were purchased less than 200 times
- Keep users that have made at least 20 orders
- Cut down the dataset by two-thirds (computationally unfeasible)

Add back the test set (we do not want to lose any rows)

User-based: Processing

```
1 df_matrix
```

product_id	1	2	3	4	8	9	10	11	12	18	...	49670	49674	49677	49678	49679	49680	49682
user_id																		
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32541	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
32542	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
32543	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
32544	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN

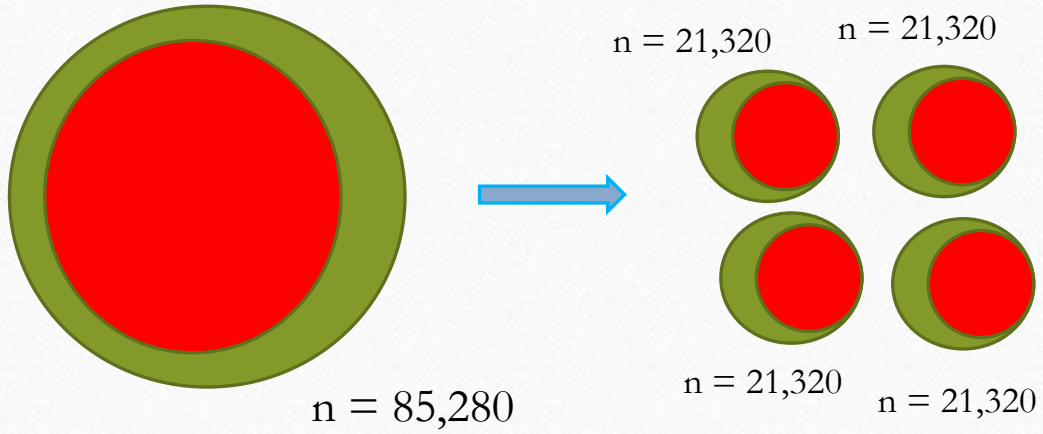
User-based: Processing

- However, these was still too many rows for the collaborative filtering process
- df_matrix.shape = (85280, 11091)
  - There are 85280 users, which will create an X by X matrix which is too huge
- Split the dataset into 4
  - Collaborative filtering with

    about **21,000** other users
  - Less ideal than getting the bigger dataset

User-based: Processing

- However, these was still too many rows for the collaborative filtering process
- df_matrix.shape = (85280, 11091)
  - There are 85280 users, which will create an X by X matrix which is too huge
- Split the dataset into 4
  - Collaborative filtering with about **21,000** other users
  - Less ideal than getting the bigger dataset



Collaborative filtering- User-based

user_id	3	4	6	7	11	12	15	16	17	19	...	40994	40997	40998
user_id														
3	0.000000	1.0	0.81443	0.983898	1.000000	0.949515	1.0	0.828379	0.943500	0.984345	...	1.0	0.987977	1.000000
4	1.000000	0.0	1.00000	1.000000	1.000000	0.975116	1.0	1.000000	0.876229	1.000000	...	1.0	1.000000	1.000000
6	0.814430	1.0	0.00000	1.000000	0.965890	1.000000	1.0	0.808820	1.000000	1.000000	...	1.0	1.000000	1.000000
7	0.983898	1.0	1.00000	0.000000	0.993542	0.984739	1.0	0.910723	1.000000	0.996214	...	1.0	0.998183	0.966148
11	1.000000	1.0	0.96589	0.993542	0.000000	0.962881	1.0	0.988262	1.000000	0.990792	...	1.0	0.986740	0.956085

Collaborative filtering- User-based

	users	closest
0	3	27868
1	4	29199
2	6	6320
3	7	23642
4	11	39034
...	...	...
21315	206202	166901
21316	206204	170857
21317	206206	194177
21318	206207	186182
21319	206208	189412

85280 rows × 2 columns

	user_id_x	closest	order_id	user_id_y	product_id_matched
0	3	27868	34782	27868	41007 47766 41007 41007 47766 41007 47766 4100...
1	28054	27868	34782	27868	41007 47766 41007 41007 47766 41007 47766 4100...
2	30218	27868	34782	27868	41007 47766 41007 41007 47766 41007 47766 4100...
3	32949	27868	34782	27868	41007 47766 41007 41007 47766 41007 47766 4100...
4	4	29199	321304	29199	26576 30752 26576 27356 30752 47141 29529 3075...
...	...	...	...	...	...
74995	206122	175161	784110	175161	39121 37067 46720 45013 42719 27344 41694 3329...
74996	206131	183313	303796	183313	46522 28083 34320 42677 45622 47865 34449 4386...
74997	206147	184769	14541	184769	41787 30305 47209 31717 49383 42445 29447 4276...
74998	206197	164092	950190	164092	30450 33120 39534 30450 39275 39534 33120
74999	206204	170857	8643	170857	24419 29447 33037 30233 26283 29447 33037 4869...

75000 rows × 5 columns

Collaborative filtering- User-based

- Added the two lists together
- F1-score of **0.12834**

Model	F1-Score
Baseline	**0.12631**
Baseline // Top Frequency items	**0.11949**
Collaborative Filtering – User	**0.12834**

Conclusion / Next Steps

Conclusion / Next Steps

- F1 score is a combination of not just hitting the right predictions, but also not recommending excessively (false positives)
  - Narrowing down recommendations, even if there might be fewer true positives, can still produce a good F1 score
- There is evidence that the target might purchase related items – can fine-tune user-based collaborative filtering more.

Conclusion / Next Steps

- Create our own train-test split to do error analysis
  - Splitting of data was difficult
- Figure out how whether any other packages can work well (e.g. Turicreate)
  - Data size issue
- Using orders that were bought > 50% of the time, rather than the top 50% frequency
  - Difference is that this user might have been buying items once for 90% of the time and recurring only for staples such as eggs- using the top half of a value_count list might recommend him items that he has stopped buying already

Conclusion / Next Steps

- More can be done to utilise more EDA features
  - Add organic products or bananas to every list (low-hanging fruits) as initial models
  - Use average of the number of items they ordered previously as a way to decide how many to include
  - Use weightage to diminish value of products for those further in time
  - Items that are placed in the cart first to have higher weightage

Thank you.