INSTACART

About Instacart

Instacart is a same-day delivery and pick up service that allows consumers to shop through the company's mobile app or website from selected grocery stores, such as Star Market, Costco and Target. The company has dedicated employees who shop groceries from the selected stores and deliver to the consumer's doorsteps. The way the company generates revenue is by adding markup on prices for specific stores, delivery and membership fee. Hence, the main business objective for the organization is to, not only increase the number of customer memberships but also improve the repeat visit and orders. Predicting consumer engagement and behavior with the products and grocery stores have a huge impact on Instacart's success.

Objective

Who- Marketa analytics who has hired us as an Algorithmic marketing analysts. Marketa is a consulting organization specializing in Marketing analytical solutions.

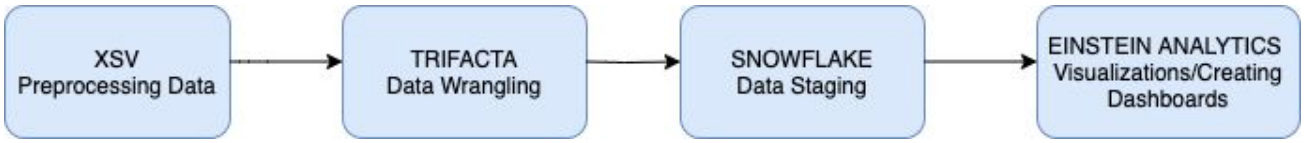
What- To analyze data and build analytical dashboards

Why- To illustrate the value of data driven analytics and to derive meaningful insights to help their business

When -Over a 2 weeks period timeline .

Where-This project will be delivered to our client Instacart. Business Analyst, Pricing Specialist and Business Strategist can use these dashboards to make business decisions which will help Instacart maximize its sales.

How-The company has a challenge using large scale datasets.
To use tools like XSV,Trifacta,Snowflake,Salesforce Einstein Analytics,Python.



Reading Datasets

The dataset is anonymized and contains a sample of over 3 million grocery orders from more than 200,000 Instacart users.
A total of 6 datasets are provided which give information on customer transactional and purchasing order details.
The information about this publicly released dataset can be found here:
https://www.kaggle.com/c/instacart-market-basket-analysis/data

Dataset 1: Aisles

Count: 134

This dataset provides information on the aisles such as aisle ID and aisle names, through which the products were organized.

aisle_id: Labels the ID of the aisles

aisle: Mentions the aisle name in the retail stores

Dataset 2: Department

Count: 21

This dataset provides information on the departments such as department names and department Id.

department_id: Labels the ID of the departments

department: Mentions the department name in the retail stores

Dataset 3: orders_products_prior

Count: 32434489

This dataset gives information on the orders, products, and reordered products

Order_id: Labels the ID of the order made by customer

Product_id: Labels the ID of the products purchased by customers

Add_to_cart_order: Sequence of the order placed in the cart

Reordered: Denotes whether the products are reordered or not

Dataset 4: orders_products_train

Count: 1384617

This dataset gives information on the orders, products, and reordered products

Order_id: Labels the ID of the order made by customer

Product_id: Labels the ID of the products purchased by customers

Add_to_cart_order: Sequence of the order placed in the cart

Reordered: Denotes whether the products are reordered or not

Dataset 5: Orders

Count: 3421083

order_id: Labels the ID of the order made by customers

user_id: Labels the ID of the users who made the purchase
order_number: Denotes the order number made by the customer
order_dow: Denotes the day of the week, the order made by the customer
order_hour_of_day: Denotes the hour of the day, the order made by the customer
days_since_prior_order: Denotes the number of days since last order

Dataset 6: Products

Count: 49688
product_id: Labels the ID of the products purchased by customers
Product_name: Denotes the product name purchased by the customer
Aisle_id: Labels the ID of the aisles
Departments_id: Labels the ID of the departments

Data Profiling

Pandas Profiling Report

Overview | Variables | Interactions | Correlations | Missing values | Sam

Overview | Reproduction | Warnings 4

Dataset statistics

Number of variables	7
Number of observations	3421083
Missing cells	206209
Missing cells (%)	0.9%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	182.7 MiB
Average record size in memory	56.0 B

Variable types

NUM	6
CAT	1

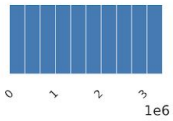
order_id

Real number ($\mathbb{R}_{\geq 0}$)

UNIQUE

Distinct count	3421083
Unique (%)	100.0%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	1710542.0
Minimum	1
Maximum	3421083
Zeros	0
Zeros (%)	0.0%
Memory size	26.1 MiB



Toggle details

First rows

	product_id	product_name	aisle_id	department_id
0	1	Chocolate Sandwich Cookies	61	19
1	2	All-Seasons Salt	104	13
2	3	Robust Golden Unsweetened Oolong Tea	94	7
3	4	Smart Ones Classic Favorites Mini Rigatoni With Vodka Cream Sauce	38	1
4	5	Green Chile Anytime Sauce	5	13
5	6	Dry Nose Oil	11	11
6	7	Pure Coconut Water With Orange	98	7
7	8	Cut Russet Potatoes Steam N' Mash	116	1
8	9	Light Strawberry Blueberry Yogurt	120	16
9	10	Sparkling Orange Juice & Prickly Pear Beverage	115	7

XSV Commands

Index-Indexing on the datafile using xsv

```
C:\Users\Priyanka Malpekar\Downloads\Dataset>
C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv index aisles.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv index departments.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv index orders.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv index order_products__prior.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv index order_products__train.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv index products.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv index sample_submission.csv
```

Headers- to find the the common columns for joins

```
C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv headers aisles.csv
1   aisle_id
2   aisle

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv headers departments.csv
1   department_id
2   department

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv headers orders.csv
1   order_id
2   user_id
3   eval_set
4   order_number
5   order_dow
6   order_hour_of_day
7   days_since_prior_order

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv headers order_products__prior.csv
1   order_id
2   product_id
3   add_to_cart_order
4   reordered

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv headers order_products__train.csv
1   order_id
2   product_id
3   add_to_cart_order
4   reordered

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv headers products.csv
1   product_id
2   product_name
3   aisle_id
4   department_id

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv headers sample_submission.csv
1   order_id
2   products
```

Flatten- A flattened view of departments csv for viewing one record at a time.

```
(base) JUIs-MacBook-Pro:Instacart juiashinkar$ xsv flatten departments.csv
department_id  1
department     frozen
#
department_id  2
department     other
#
department_id  3
department     bakery
#
department_id  4
department     produce
#
department_id  5
department     alcohol
#
department_id  6
department     international
#
department_id  7
department     beverages
#
department_id  8
department     pets
#
department_id  9
department     dry goods pasta
#
department_id  10
department     bulk
#
department_id  11
department     personal care
#
department_id  12
department     meat seafood
#
department_id  13
department     pantry
#
department_id  14
department     breakfast
#
department_id  15
department     canned goods
#
department_id  16
department     dairy eggs
#
department_id  17
department     household
#
department_id  18
department     babies
#
```

Count- to get number of rows in each table

```
(base) JUIs-MacBook-Pro:Instacart juiashinkar$ xsv count aisles.csv
134
(base) JUIs-MacBook-Pro:Instacart juiashinkar$ xsv count departments.csv
21
(base) JUIs-MacBook-Pro:Instacart juiashinkar$ xsv count order_products__prior.csv
32434489
(base) JUIs-MacBook-Pro:Instacart juiashinkar$
(base) JUIs-MacBook-Pro:Instacart juiashinkar$ xsv count order_products__train.csv
1384617
(base) JUIs-MacBook-Pro:Instacart juiashinkar$ xsv count orders.csv
3421083
(base) JUIs-MacBook-Pro:Instacart juiashinkar$ xsv count products.csv
49688
(base) JUIs-MacBook-Pro:Instacart juiashinkar$ xsv count sample_submission.csv
75000
(base) JUIs-MacBook-Pro:Instacart juiashinkar$
```

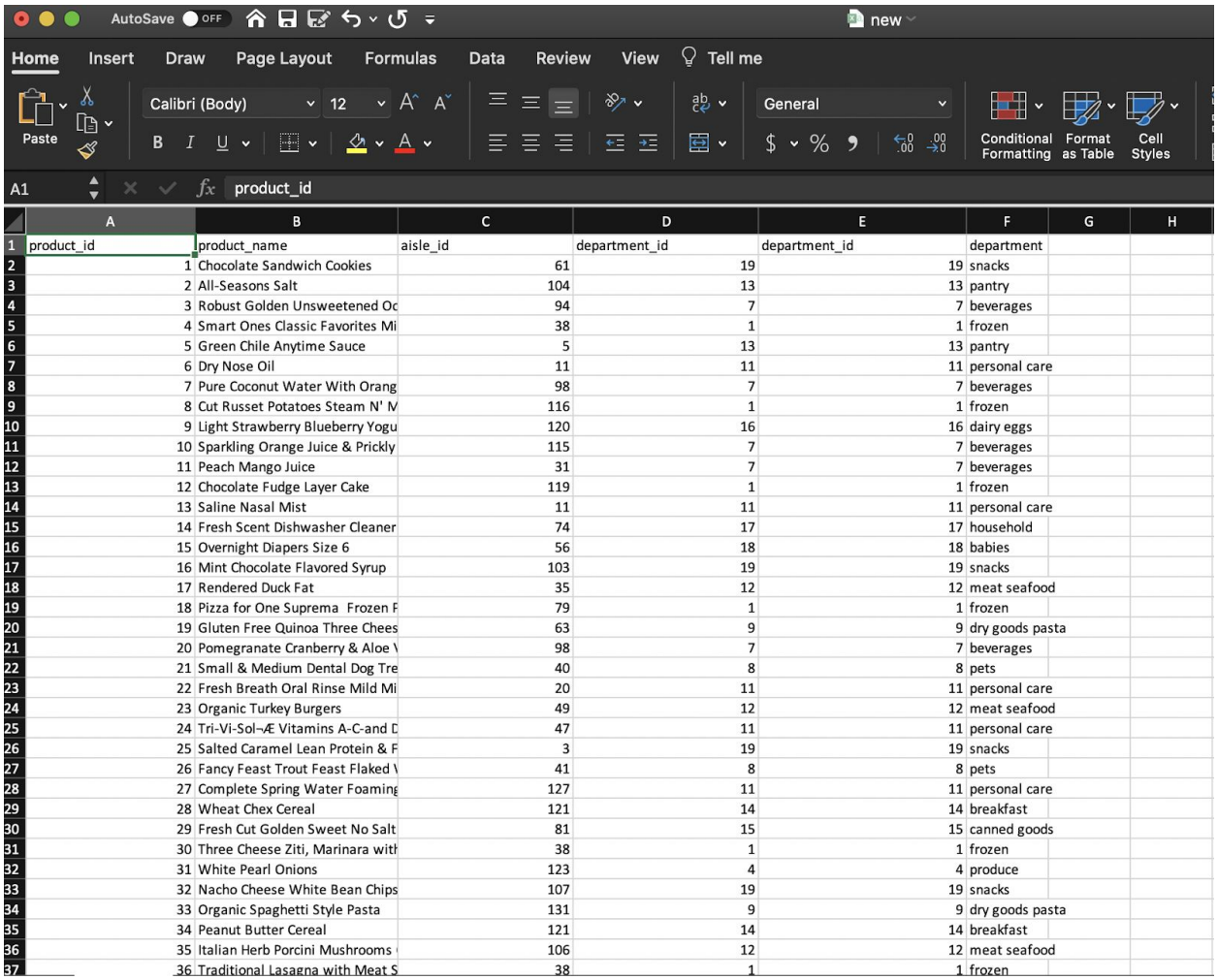
Frequency- to Build frequency tables of each column in CSV data.

```
(base) JUIs-MacBook-Pro:Instacart juiashinkar$ xsv frequency orders.csv
field,value,count
order_id,1652661,1
order_id,1125671,1
order_id,189682,1
order_id,3046061,1
order_id,1794803,1
order_id,2978465,1
order_id,2697778,1
order_id,2291184,1
order_id,904558,1
order_id,2240195,1
user_id,168637,100
user_id,78929,100
user_id,99352,100
user_id,66500,100
user_id,199880,100
user_id,4563,100
user_id,188947,100
user_id,175540,100
user_id,122848,100
user_id,201248,100
eval_set,prior,3214874
eval_set,train,131209
eval_set,test,75000
order_number,1,206209
order_number,4,206209
order_number,3,206209
order_number,2,206209
order_number,5,182223
order_number,6,162633
order_number,7,146468
order_number,8,132618
order_number,9,120918
order_number,10,110728
order_dow,0,600905
order_dow,1,587478
order_dow,2,467260
order_dow,5,453368
order_dow,6,448761
order_dow,3,436972
order_dow,4,426339
order_hour_of_day,10,288418
order_hour_of_day,11,284728
order_hour_of_day,15,283639
order_hour_of_day,14,283042
order_hour_of_day,13,277999
order_hour_of_day,12,272841
order_hour_of_day,16,272553
order_hour_of_day,09,257812
order_hour_of_day,17,228795
order_hour_of_day,18,182912
days_since_prior_order,30.0,369323
days_since_prior_order,7.0,320608
days_since_prior_order,6.0,240013
days_since_prior_order,4.0,221696
days_since_prior_order,3.0,217005
days_since_prior_order,5.0,214503
days_since_prior_order,(NULL),206209
days_since_prior_order,2.0,193206
days_since_prior_order,8.0,181717
days_since_prior_order,1.0,145247
(base) JUIs-MacBook-Pro:Instacart juiashinkar$
```

Join- Joining departments and products on department_id and exporting the new dataset in a csv

```
:\Users\Priyanka Malpekar\Downloads\Dataset>xsv join order_id ordersFinal.csv order_id order_products__train.csv > join_orders.csv

:\Users\Priyanka Malpekar\Downloads\Dataset>
```

```
(base) JUIs-MacBook-Pro:Instacart juiashinkar$ xsv join --left department_id products.csv
 department_id departments.csv> new.csv
```



	A	B	C	D	E	F
1	product_id	product_name	aisle_id	department_id	department_id	department
2	1	Chocolate Sandwich Cookies	61	19	19	snacks
3	2	All-Seasons Salt	104	13	13	pantry
4	3	Robust Golden Unsweetened Oc	94	7	7	beverages
5	4	Smart Ones Classic Favorites Mi	38	1	1	frozen
6	5	Green Chile Anytime Sauce	5	13	13	pantry
7	6	Dry Nose Oil	11	11	11	personal care
8	7	Pure Coconut Water With Orang	98	7	7	beverages
9	8	Cut Russet Potatoes Steam N' M	116	1	1	frozen
10	9	Light Strawberry Blueberry Yogu	120	16	16	dairy eggs
11	10	Sparkling Orange Juice & Prickly	115	7	7	beverages
12	11	Peach Mango Juice	31	7	7	beverages
13	12	Chocolate Fudge Layer Cake	119	1	1	frozen
14	13	Saline Nasal Mist	11	11	11	personal care
15	14	Fresh Scent Dishwasher Cleaner	74	17	17	household
16	15	Overnight Diapers Size 6	56	18	18	babies
17	16	Mint Chocolate Flavored Syrup	103	19	19	snacks
18	17	Rendered Duck Fat	35	12	12	meat seafood
19	18	Pizza for One Suprema Frozen F	79	1	1	frozen
20	19	Gluten Free Quinoa Three Chees	63	9	9	dry goods pasta
21	20	Pomegranate Cranberry & Aloe V	98	7	7	beverages
22	21	Small & Medium Dental Dog Tre	40	8	8	pets
23	22	Fresh Breath Oral Rinse Mild Mi	20	11	11	personal care
24	23	Organic Turkey Burgers	49	12	12	meat seafood
25	24	Tri-Vi-Sol¬Æ Vitamins A-C-and D	47	11	11	personal care
26	25	Salted Caramel Lean Protein & F	3	19	19	snacks
27	26	Fancy Feast Trout Feast Flaked V	41	8	8	pets
28	27	Complete Spring Water Foaming	127	11	11	personal care
29	28	Wheat Chex Cereal	121	14	14	breakfast
30	29	Fresh Cut Golden Sweet No Salt	81	15	15	canned goods
31	30	Three Cheese Ziti, Marinara with	38	1	1	frozen
32	31	White Pearl Onions	123	4	4	produce
33	32	Nacho Cheese White Bean Chips	107	19	19	snacks
34	33	Organic Spaghetti Style Pasta	131	9	9	dry goods pasta
35	34	Peanut Butter Cereal	121	14	14	breakfast
36	35	Italian Herb Porcini Mushrooms	106	12	12	meat seafood
37	36	Traditional Lasagna with Meat S	38	1	1	frozen

Stats- to show the basic types and statistics of each column in the csv file

```
(base) JUIs-MacBook-Pro:Instacart juiashinkar$ xsv stats products.csv
field,type,sum,min,max,min_length,max_length,mean,stddev
product_id,Integer,1234473516,1,49688,1,5,24844.5,14343.690084842185
product_name,Unicode,,#2 Coffee Filters,with a Splash of Pineapple Coconut Water
,3,159,,
aisle_id,Integer,3367335,1,134,1,3,67.76958219288447,38.315776844046276
department_id,Integer,582775,1,21,1,2,11.728687006923241,5.850350638292878
(base) JUIs-MacBook-Pro:Instacart juiashinkar$
```

Slice- to return the rows in the range specified

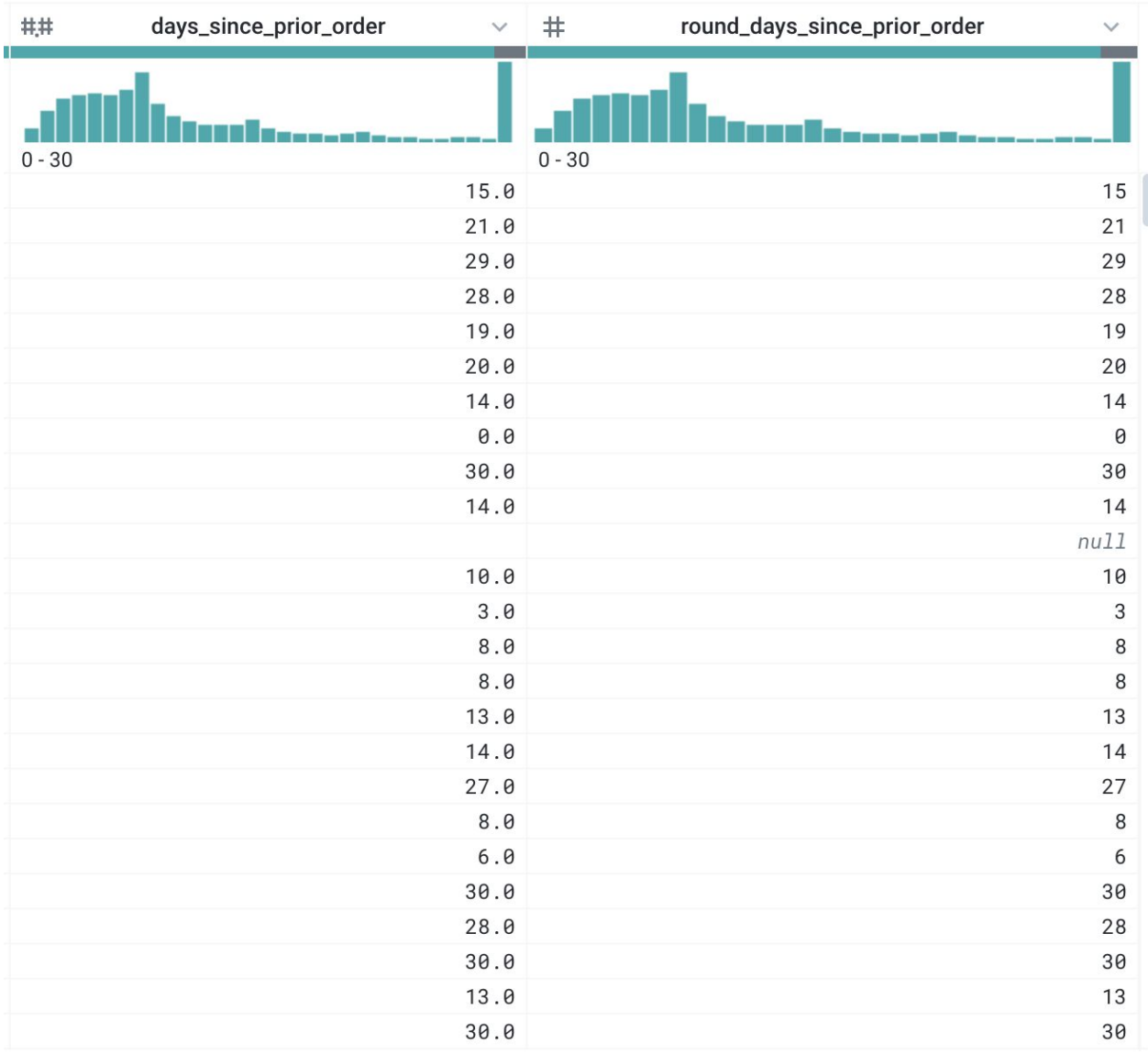
```
C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv slice orders.csv -s 1 -e 3000000 > OrdersNew.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>
C:\Users\Priyanka Malpekar\Downloads\Dataset>
```

```
C:\Users\Priyanka Malpekar\Downloads\Dataset>
C:\Users\Priyanka Malpekar\Downloads\Dataset>
C:\Users\Priyanka Malpekar\Downloads\Dataset>
C:\Users\Priyanka Malpekar\Downloads\Dataset>
C:\Users\Priyanka Malpekar\Downloads\Dataset>
C:\Users\Priyanka Malpekar\Downloads\Dataset>
C:\Users\Priyanka Malpekar\Downloads\Dataset>
C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv slice Orders_Slice.csv -s 1 -e 2500000 > Orders_Final.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv slice Orders_Slice.csv -s 1 -e 2200000 > Orders_Final.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>slice order_products__prior.csv -s 1 -e 3300000 > order_products_prior_Final.csv
'slice' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv slice order_products__prior.csv -s 1 -e 3300000 > order_products_prior_Final.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv slice order_products__prior.csv -s 1 -e 5000000 > order_products_prior_Final.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv slice order_products__prior.csv -s 1 -e 6000000 > order_products_prior_Final.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv slice Orders_V1.csv -s 1 -e 2500000 > Orders_Final.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv slice Orders_V1.csv -s 1 -e 1500000 > Orders_FinalV2.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv slice Orders_V1.csv -s 1 -e 1300000 > Orders_FinalV2.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv slice Orders_V1.csv -s 1 -e 2500000 > order_products_prior_FinalV2.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv slice Orders_V1.csv -s 1 -e 1500000 > order_products_prior_FinalV2.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>xsv slice Orders_V1.csv -s 1 -e 1300000 > order_products_prior_FinalV2.csv

C:\Users\Priyanka Malpekar\Downloads\Dataset>
```

Trifacta-Data wrangling

(A)Orders Dataset

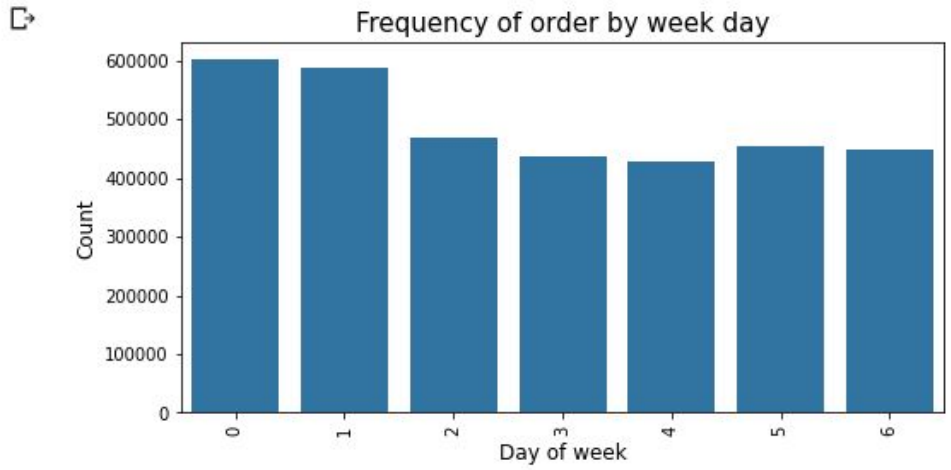
Changing data type of days from decimal to integer

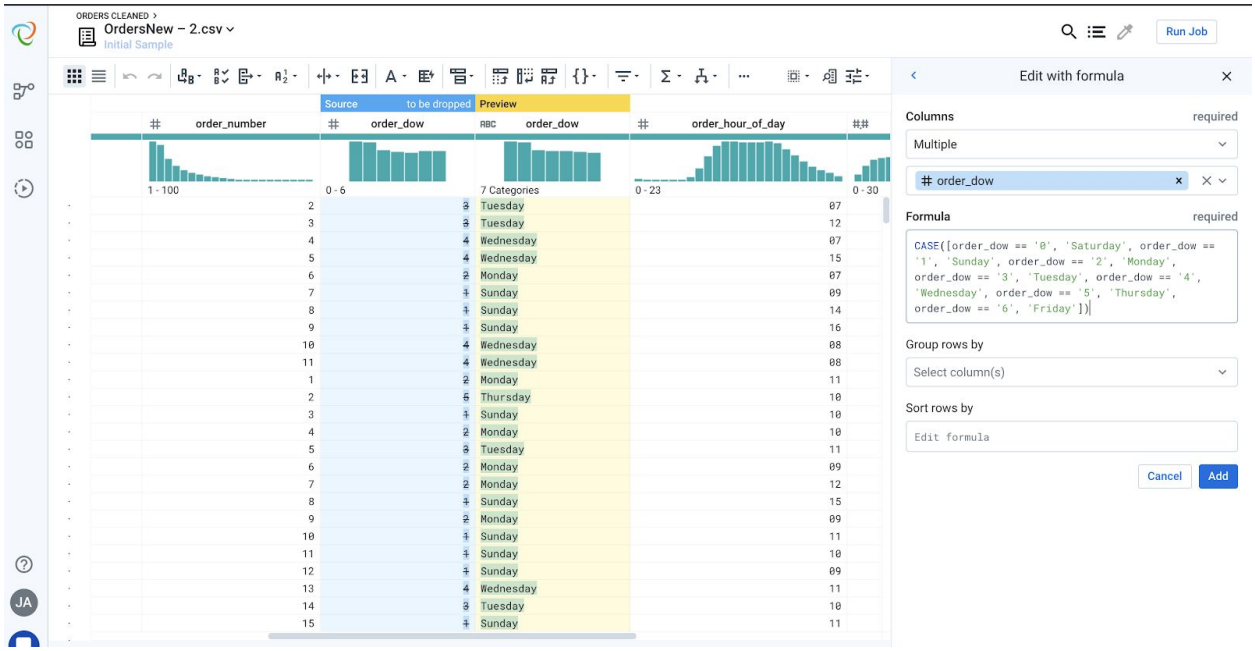


days_since_prior_order	round_days_since_prior_order
0 - 30	0 - 30
15.0	15
21.0	21
29.0	29
28.0	28
19.0	19
20.0	20
14.0	14
0.0	0
30.0	30
14.0	14
	null
10.0	10
3.0	3
8.0	8
8.0	8
13.0	13
14.0	14
27.0	27
8.0	8
6.0	6
30.0	30
28.0	28
30.0	30
13.0	13
30.0	30

Creating a column for Day of Week

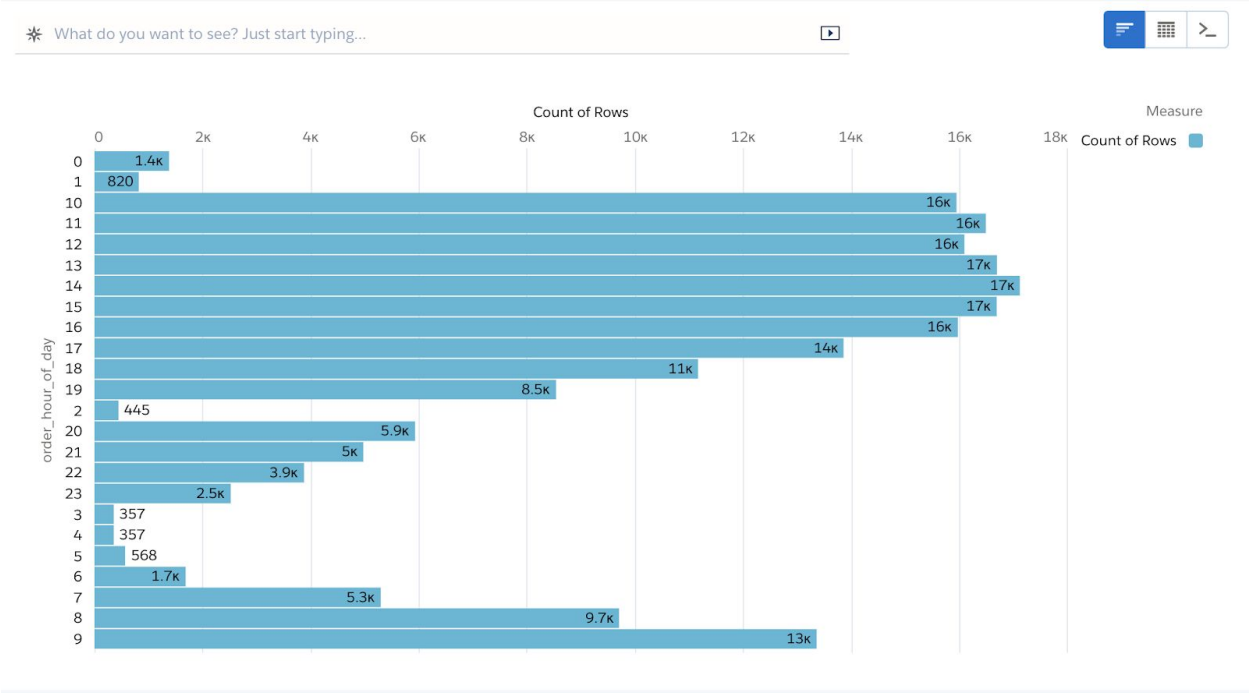
Setting order_dow to respective days of week based on the insights deduced from this following plot



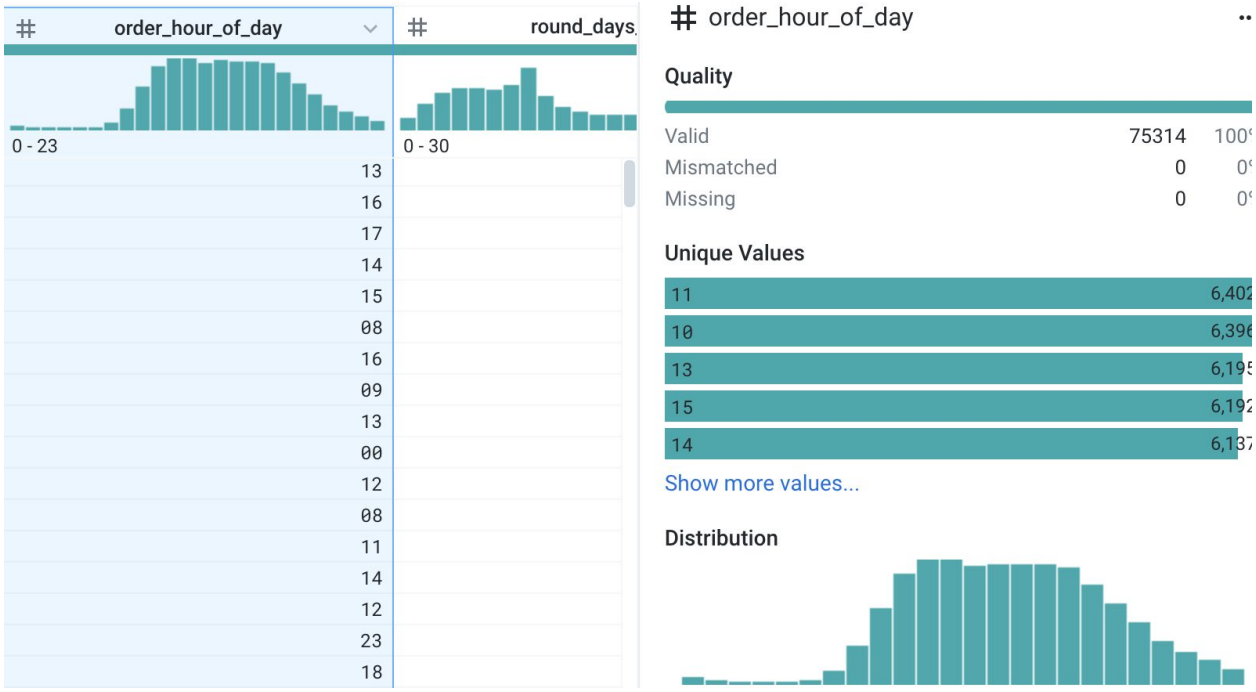


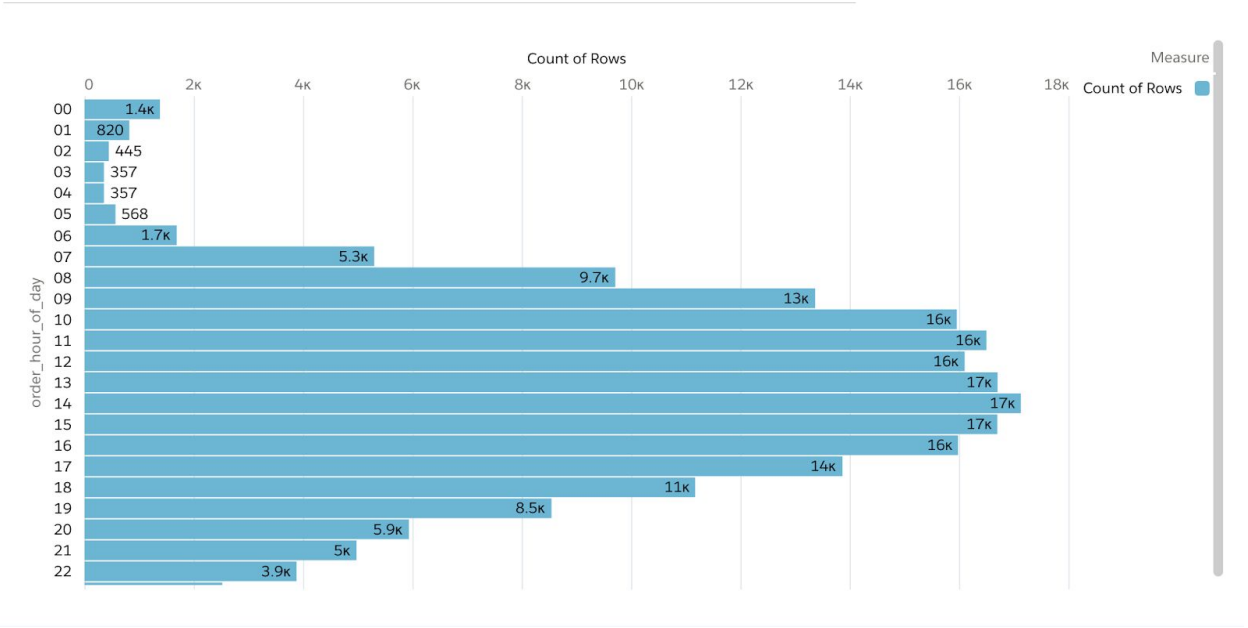
Adding zeros to order_hour_of_day

Order_hour_of_day- while using in salesforce it had to be converted to dimension and when arranged in ascending order it would plot the graph in the following manner giving error in the sequence

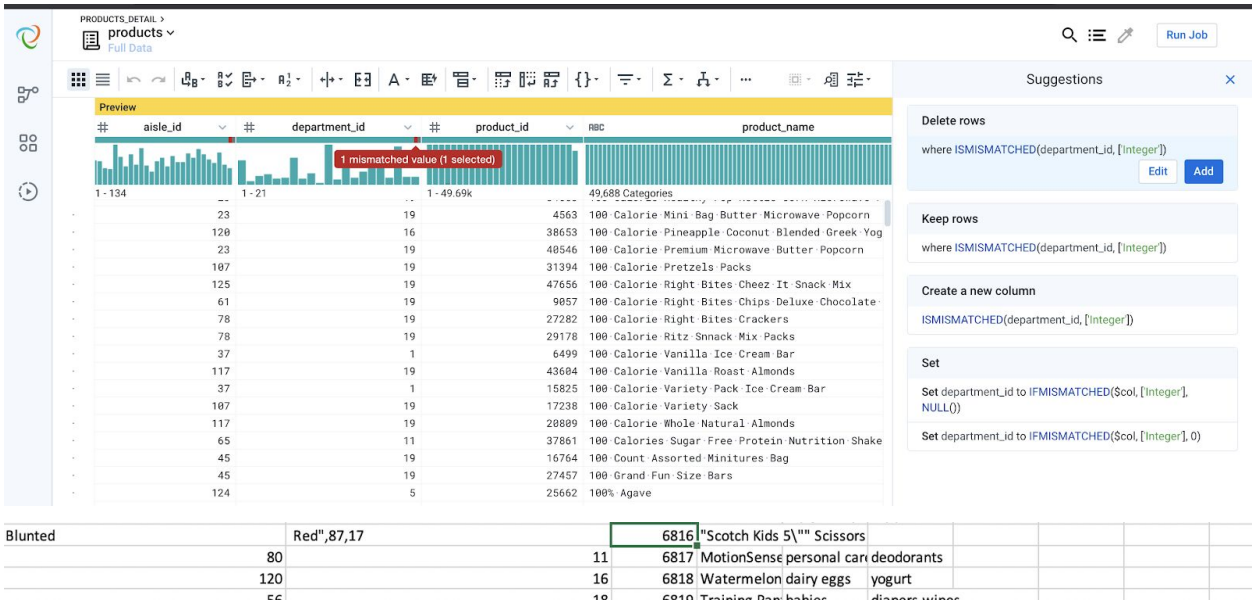


So Adding a 0 to the hour of day column to get the records in ascending order as expected.

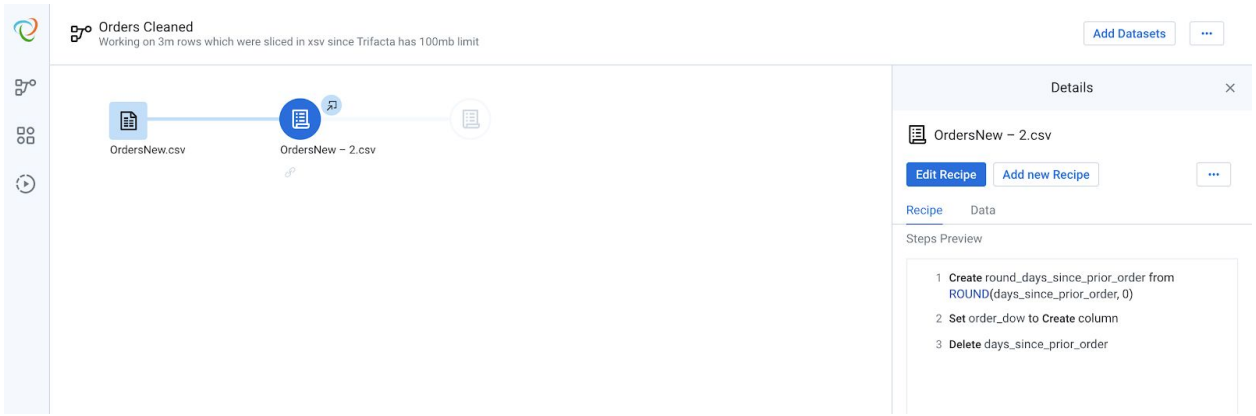




Eliminating mismatch values



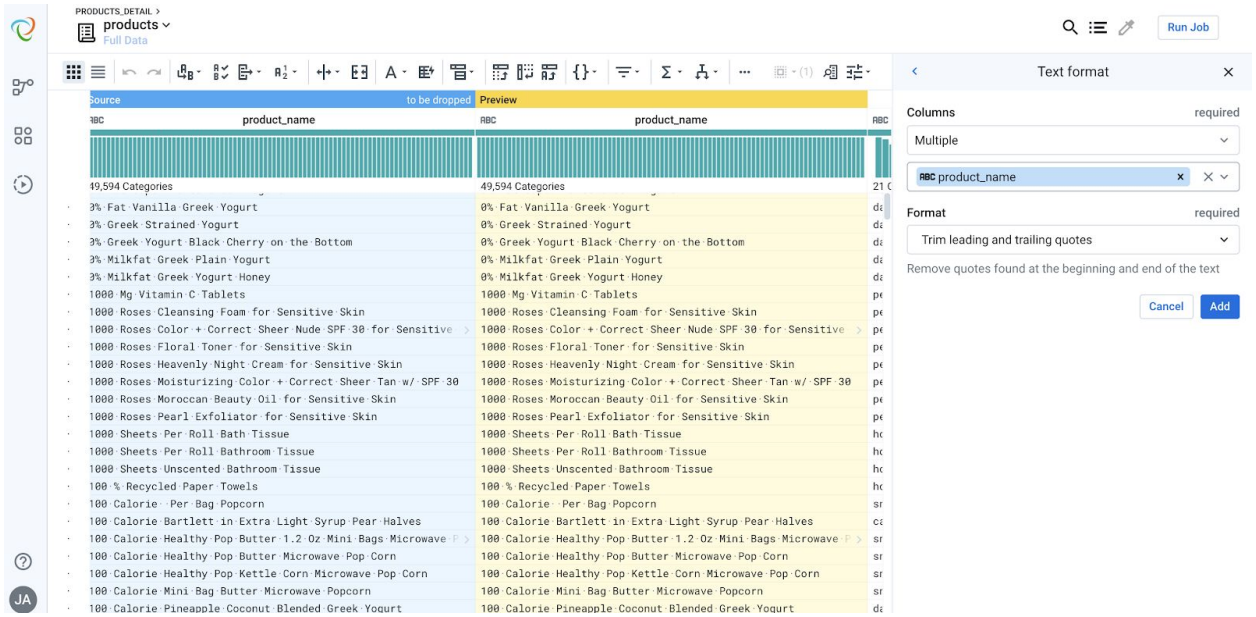
Creating a recipe to do the above changes in the Orders table



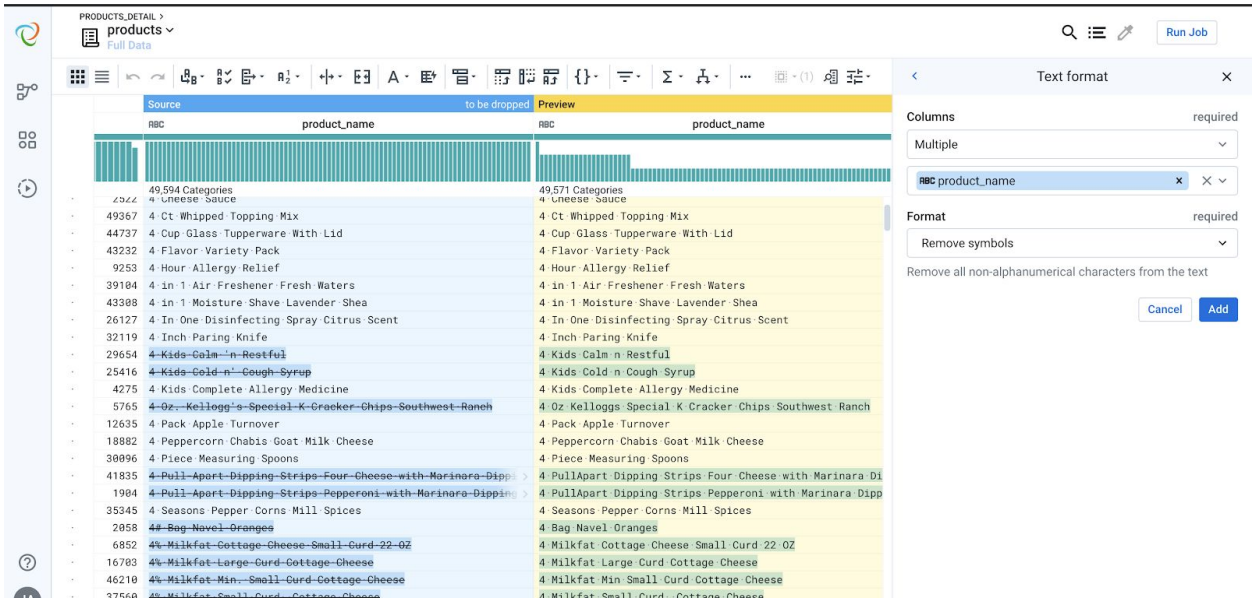
(B)Products Dataset

Trimming quotes in Product name

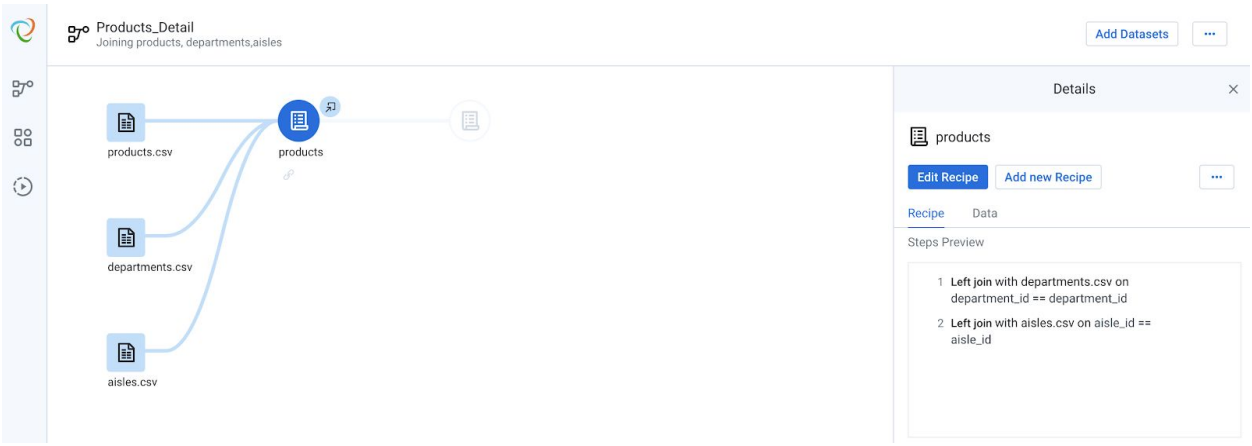
Snowflake does not accept records having certain symbols so eliminating those to make the data load-ready.



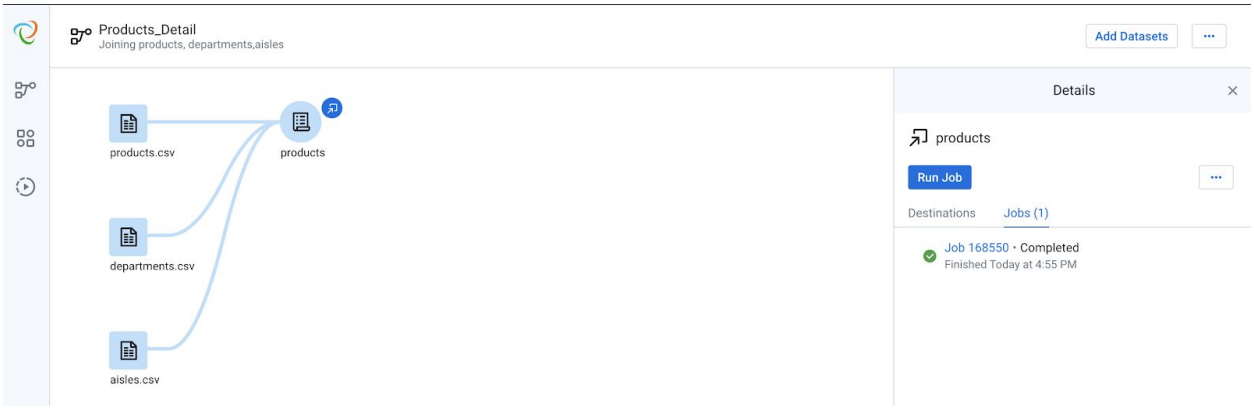
Removing Symbols in Product name



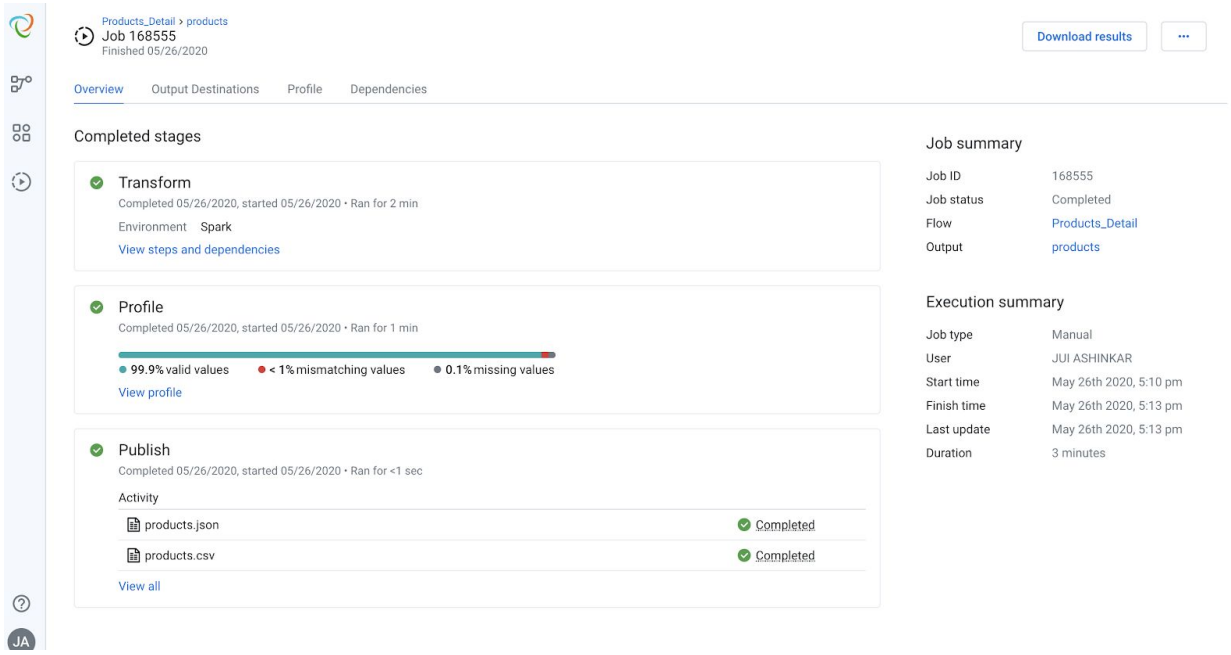
Joining Products with departments and aisles tables



Running JOB



Published Dataset to be used in Snowflake



Snowflake

Installing snowsql CLI

```
[(base) JUIs-MacBook-Pro:~ juiashinkar$ snowsql --bootstrap-version
-bash: snowsql: command not found
[(base) JUIs-MacBook-Pro:~ juiashinkar$ curl -O https://sfc-repo.snowflakecomputi
ng.com/snowsql/bootstrap/1.2/darwin_x86_64/snowsql-1.2.5-darwin_x86_64.pkg
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 26.7M  100 26.7M    0     0  12.9M      0  0:00:02  0:00:02 --:--:-- 12.9M
(base) JUIs-MacBook-Pro:~ juiashinkar$ curl -O https://sfc-repo.azure.snowflakec
omputing.com/snowsql/bootstrap/1.2/darwin_x86_64/snowsql-1.2.5-darwin_x86_64.pkg
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 26.7M  100 26.7M    0     0  11.2M      0  0:00:02  0:00:02 --:--:-- 11.2M
(base) JUIs-MacBook-Pro:~ juiashinkar$
```

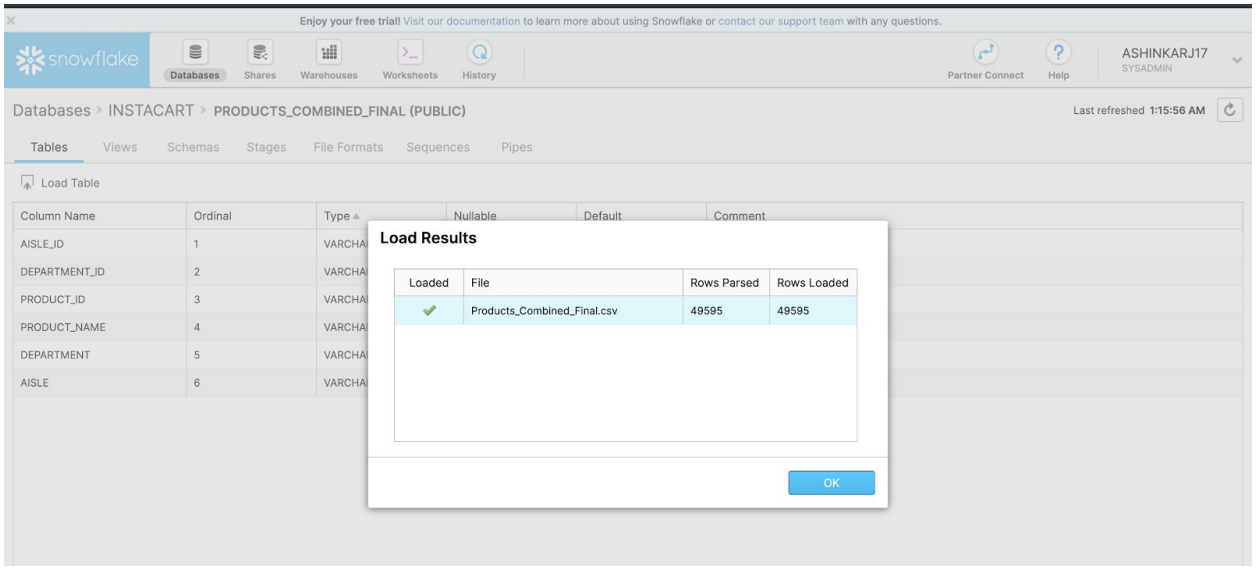
Configuring Snowsql CLI

```
  -f, --filename PATH             File to execute.
  -q, --query TEXT                Query to execute.
  --config PATH                   Path and name of the SnowSQL configuration
                                  file. By default, ~/.snowsql/config.
  -P, --prompt                    Forces a password prompt. By default,
                                  $SNOWSQL_PWD is used to set the password.
  -M, --mfa-prompt                Forces a prompt for the second token for
                                  MFA.
  -c, --connection TEXT           Named set of connection parameters to use.
  --single-transaction            Connects with autocommit disabled. Wraps
                                  BEGIN/COMMIT around statements to execute
                                  them as a single transaction, ensuring all
                                  commands complete successfully or no change
                                  is applied.
  --private-key-path PATH         Path to private key file in PEM format used
                                  for key pair authentication. Private key
                                  file is required to be encrypted and
                                  passphrase is required to be specified in
                                  environment variable
                                  $SNOWSQL_PRIVATE_KEY_PASSPHRASE
  -U, --upgrade                   Force upgrade of SnowSQL to the latest
                                  version.
  -K, --client-session-keep-alive
                                  Keep the session active indefinitely, even
                                  if there is no activity from the user..
  --disable-request-pooling       Disable request pooling. This can help speed
                                  up connection failover
  --token TEXT                    The token to be used with oauth
                                  authentication method
  -?, --help                      Show this message and exit.
(base) JUIs-MacBook-Pro:~ juiashinkar$ use demo_db
-bash: use: command not found
(base) JUIs-MacBook-Pro:~ juiashinkar$ snowsql -a uz77976.us-central1.gcp
User: ASHINKARJ17
Password:
* SnowSQL * v1.2.5
Type SQL statements or !help
ASHINKARJ17#COMPUTE_WH@(no database).(no schema)>
```

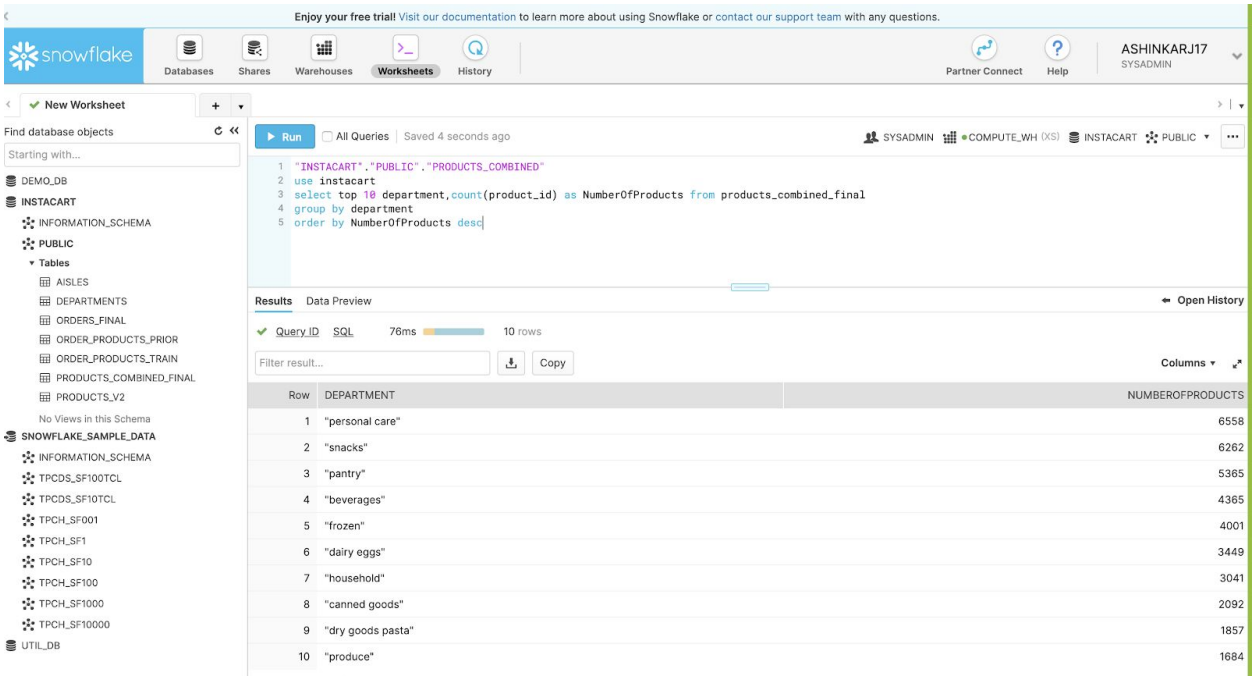
Creating Table using CLI

```
  --config PATH                   Path and name of the SnowSQL configuration
                                  file. By default, ~/.snowsql/config.
  -P, --prompt                    Forces a password prompt. By default,
                                  $SNOWSQL_PWD is used to set the password.
  -M, --mfa-prompt                Forces a prompt for the second token for
                                  MFA.
  -c, --connection TEXT           Named set of connection parameters to use.
  --single-transaction            Connects with autocommit disabled. Wraps
                                  BEGIN/COMMIT around statements to execute
                                  them as a single transaction, ensuring all
                                  commands complete successfully or no change
                                  is applied.
  --private-key-path PATH         Path to private key file in PEM format used
                                  for key pair authentication. Private key
                                  file is required to be encrypted and
                                  passphrase is required to be specified in
                                  environment variable
                                  $SNOWSQL_PRIVATE_KEY_PASSPHRASE
  -U, --upgrade                   Force upgrade of SnowSQL to the latest
                                  version.
  -K, --client-session-keep-alive
                                  Keep the session active indefinitely, even
                                  if there is no activity from the user..
  --disable-request-pooling       Disable request pooling. This can help speed
                                  up connection failover
  --token TEXT                    The token to be used with oauth
                                  authentication method
  -?, --help                      Show this message and exit.
(base) JUIs-MacBook-Pro:~ juiashinkar$ snowsql -a uz77976.us-central1.gcp
User: ASHINKARJ17
Password:
* SnowSQL * v1.2.5
Type SQL statements or !help
ASHINKARJ17#COMPUTE_WH@(no database).(no schema)>use INSTACART;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
1 Row(s) produced. Time Elapsed: 0.129s
ASHINKARJ17#COMPUTE_WH@INSTACART.PUBLIC>CREATE TABLE "INSTACART"."PUBLIC"."AISLE" ("AISLE_ID" STRING, "AISLE" STRING);
+-----------------------------------+
| status                            |
|-----------------------------------|
| Table AISLE successfully created. |
+-----------------------------------+
1 Row(s) produced. Time Elapsed: 0.338s
ASHINKARJ17#COMPUTE_WH@INSTACART.PUBLIC>
```

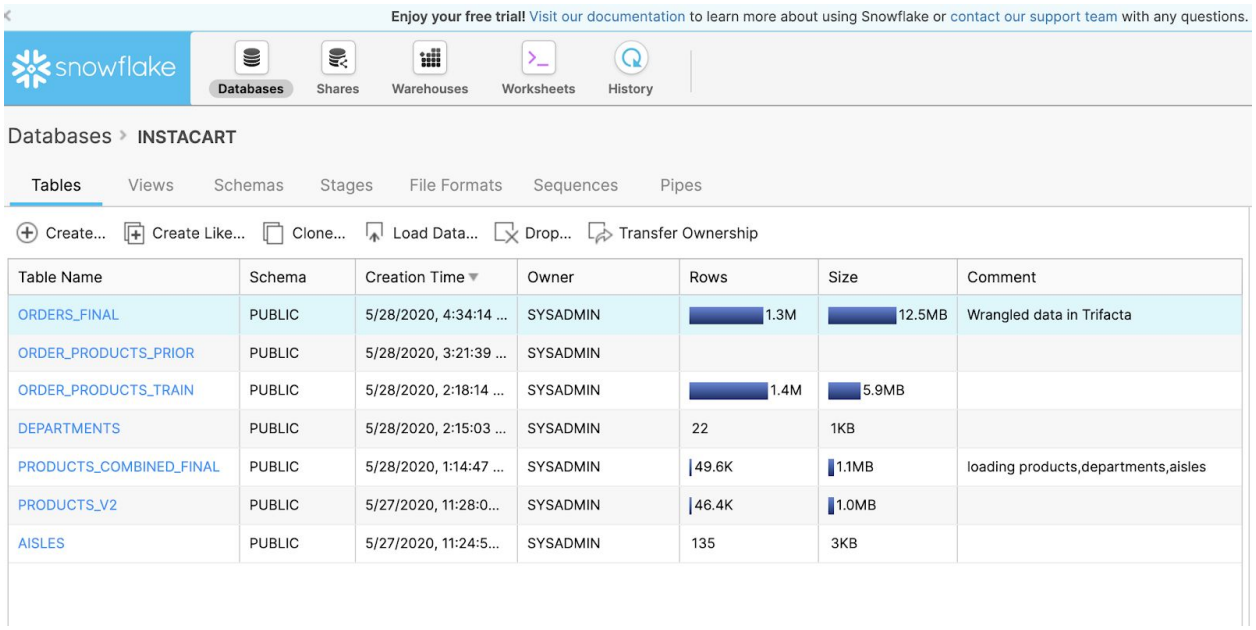
Staging tables that were joined in Trifacta into Snowflake



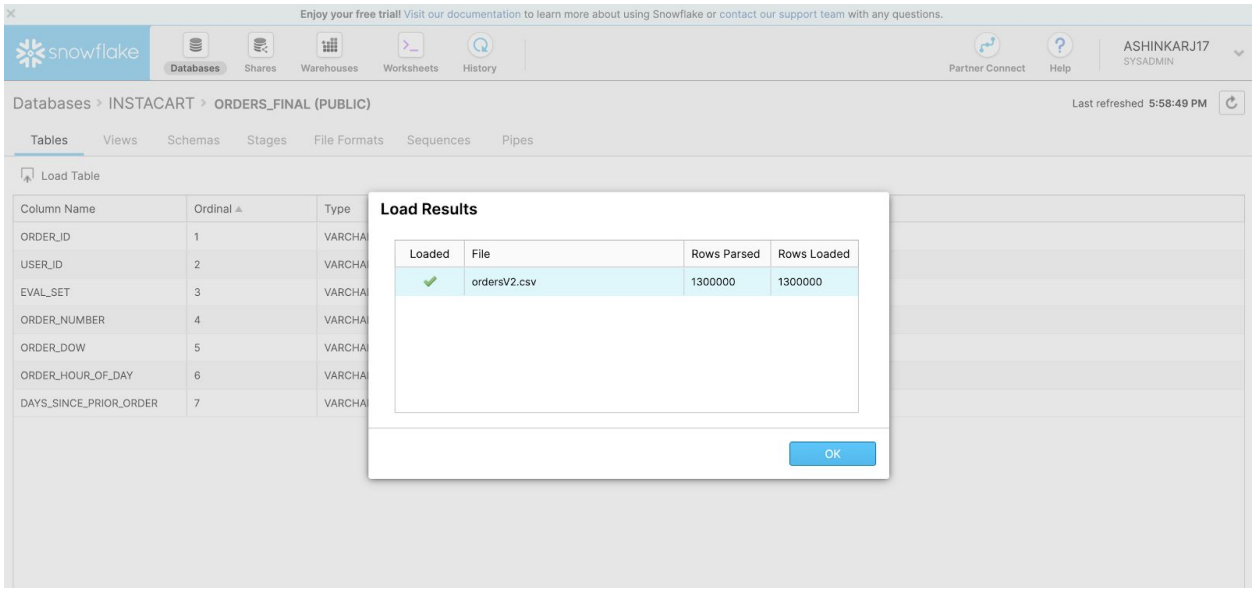
Querying tables



Staging all datasets in Snowflake

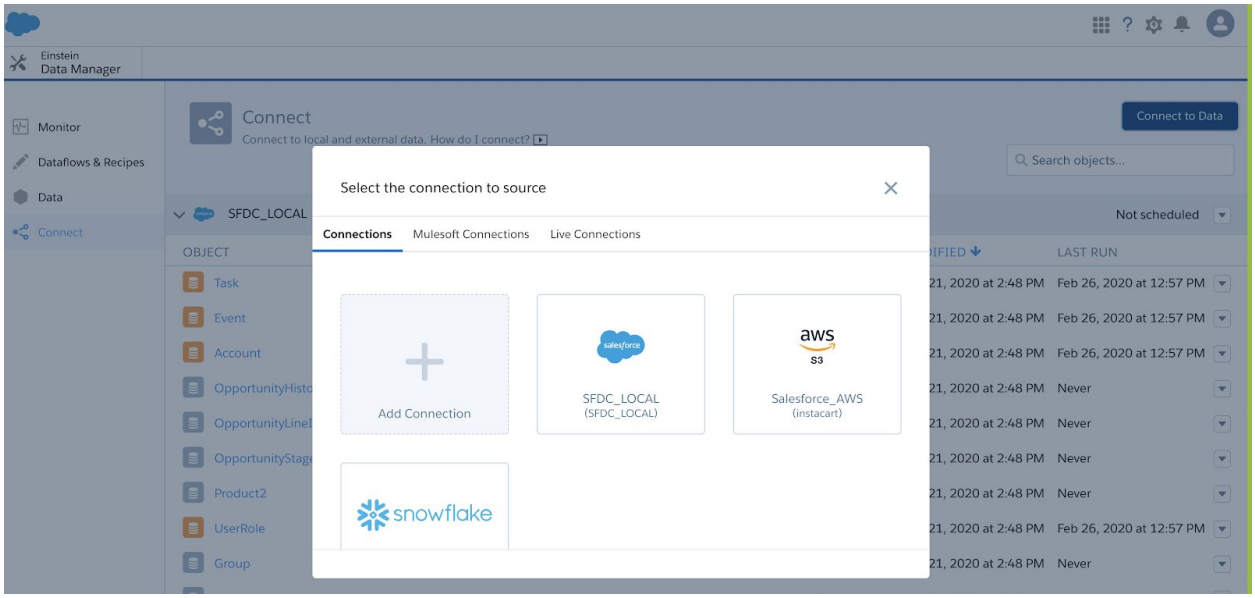


Limit of 50mb in snowflake so loading sampled data of 13m rows

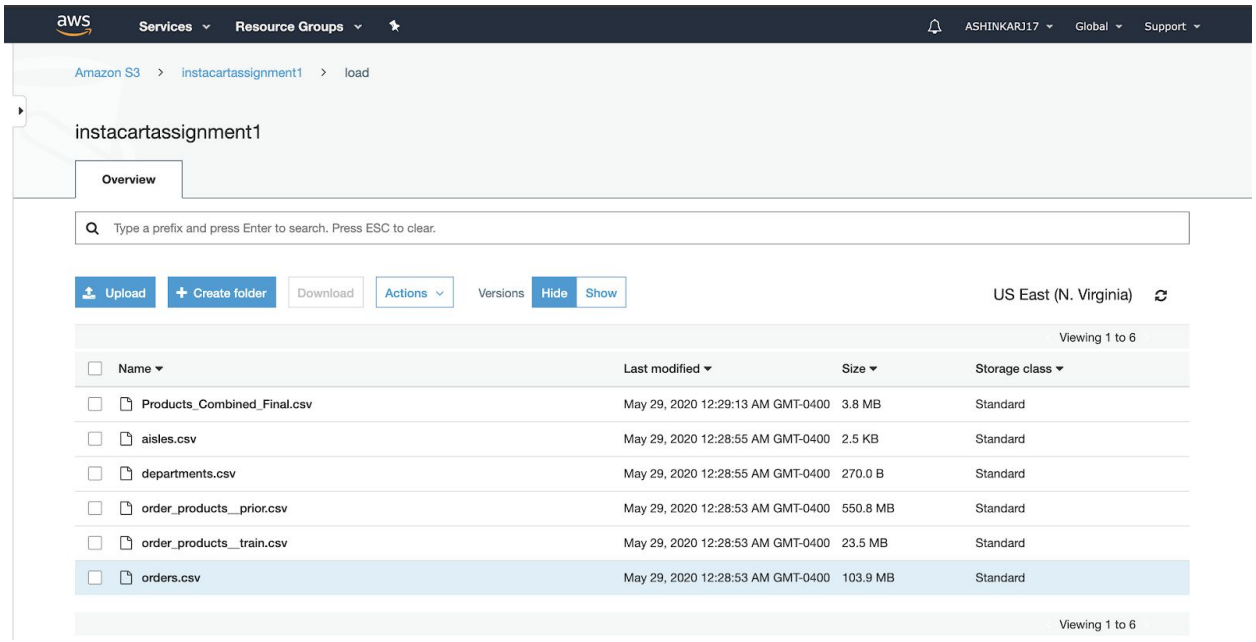


Einstein Analytics

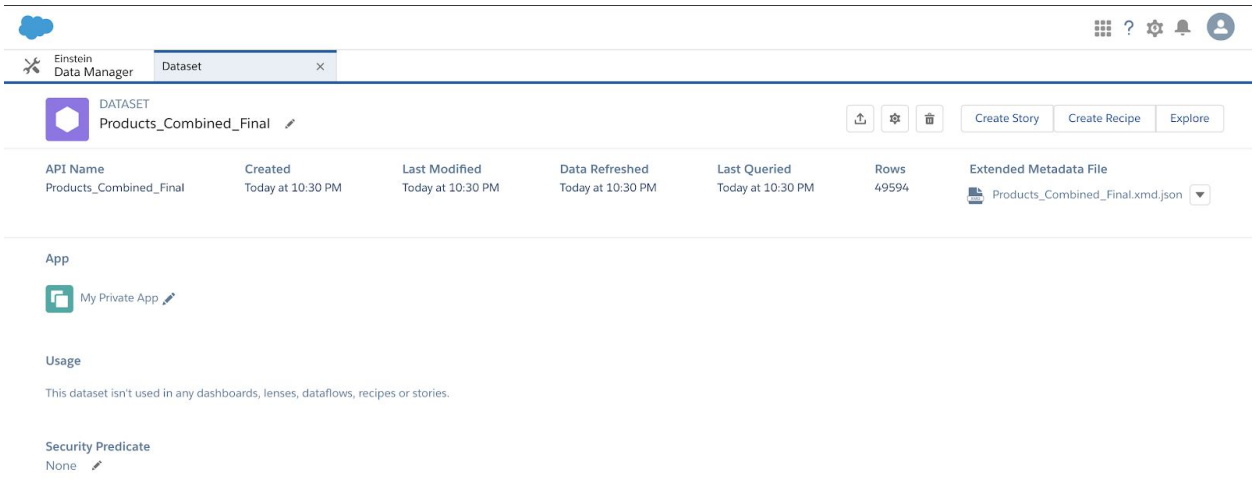
Created Connections to AWS S3 and Snowflake to import Datasets into Einstein



AWS S3 Connector - loaded all datasets in aws bucket



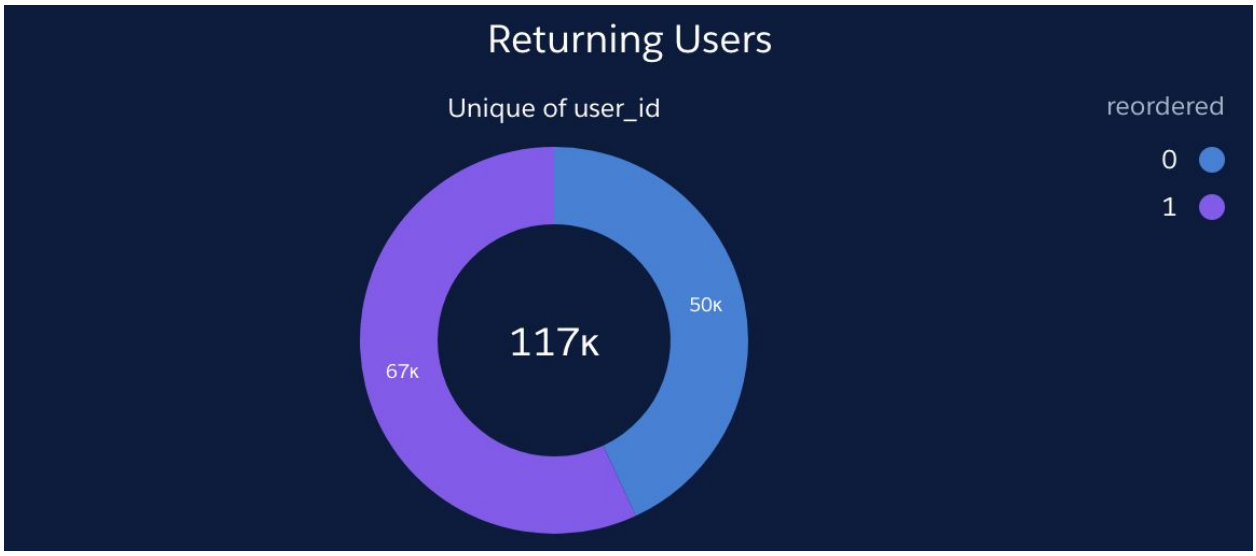
Loaded Data from snowflake



Returning Users

Gives the total number of users who have ordered on Instacart more than once.

Gives an overview of how many users like Instacart and have reordered.

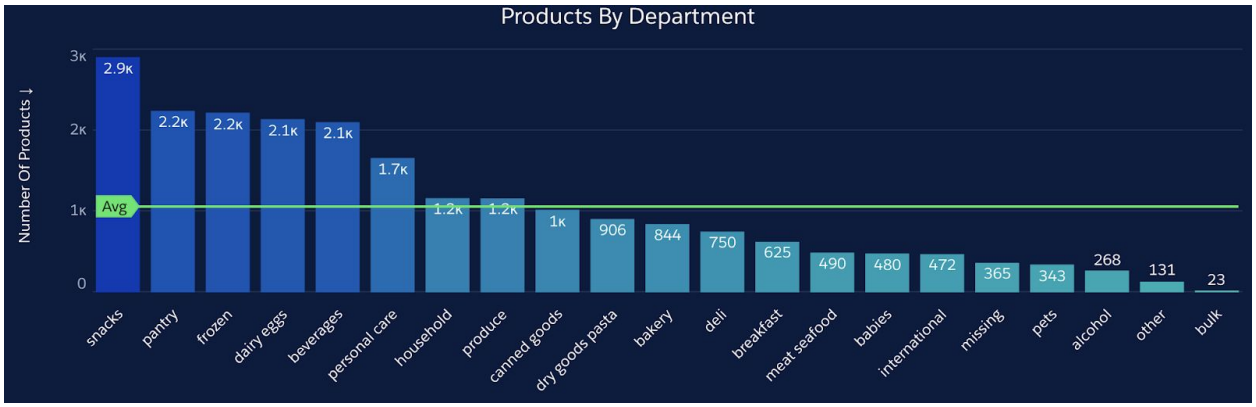


Products By Department

Gives the number of Products Instacart has in each Department.

On an Average there are 1062 Products in each department.

Example: Snacks has 2907 variety of products while Bulk department only has 23 products



Products Ordered By Department

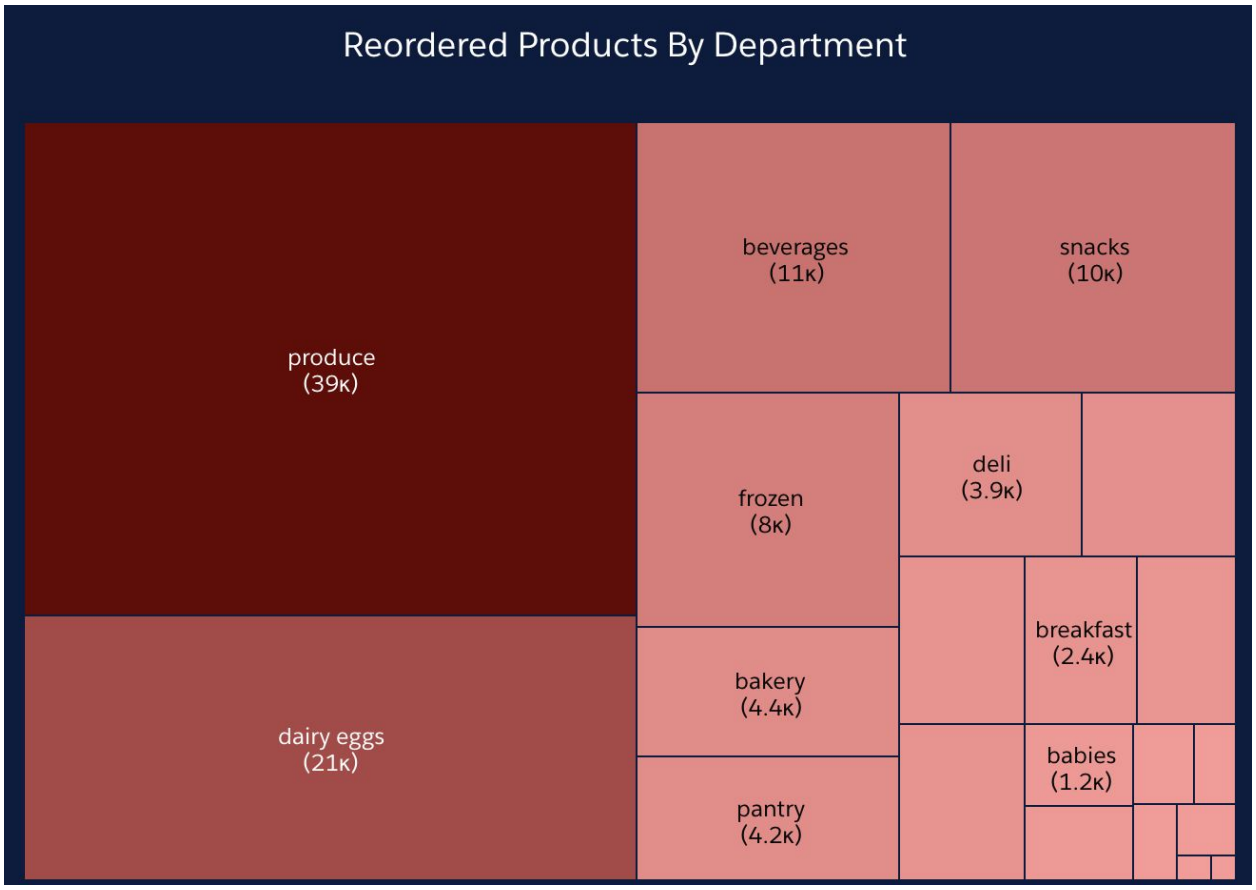
Represents total number of products ordered from each department and how much percent each department contributes towards total ordered products
Example: Produce department has the most number of ordered products(59276) and it is 29.65% of total products ordered.

Products Ordered By Department

department	Products Ordered ↓	Percentage Of Products Ordered
produce	59,276	29.65%
dairy eggs	31,455	15.74%
snacks	17,191	8.6%
beverages	16,676	8.34%
frozen	14,333	7.17%
pantry	11,664	5.84%
bakery	6,925	3.46%
canned goods	6,788	3.4%
deli	6,421	3.21%
dry goods pasta	5,542	2.77%

Reordered Products By Department

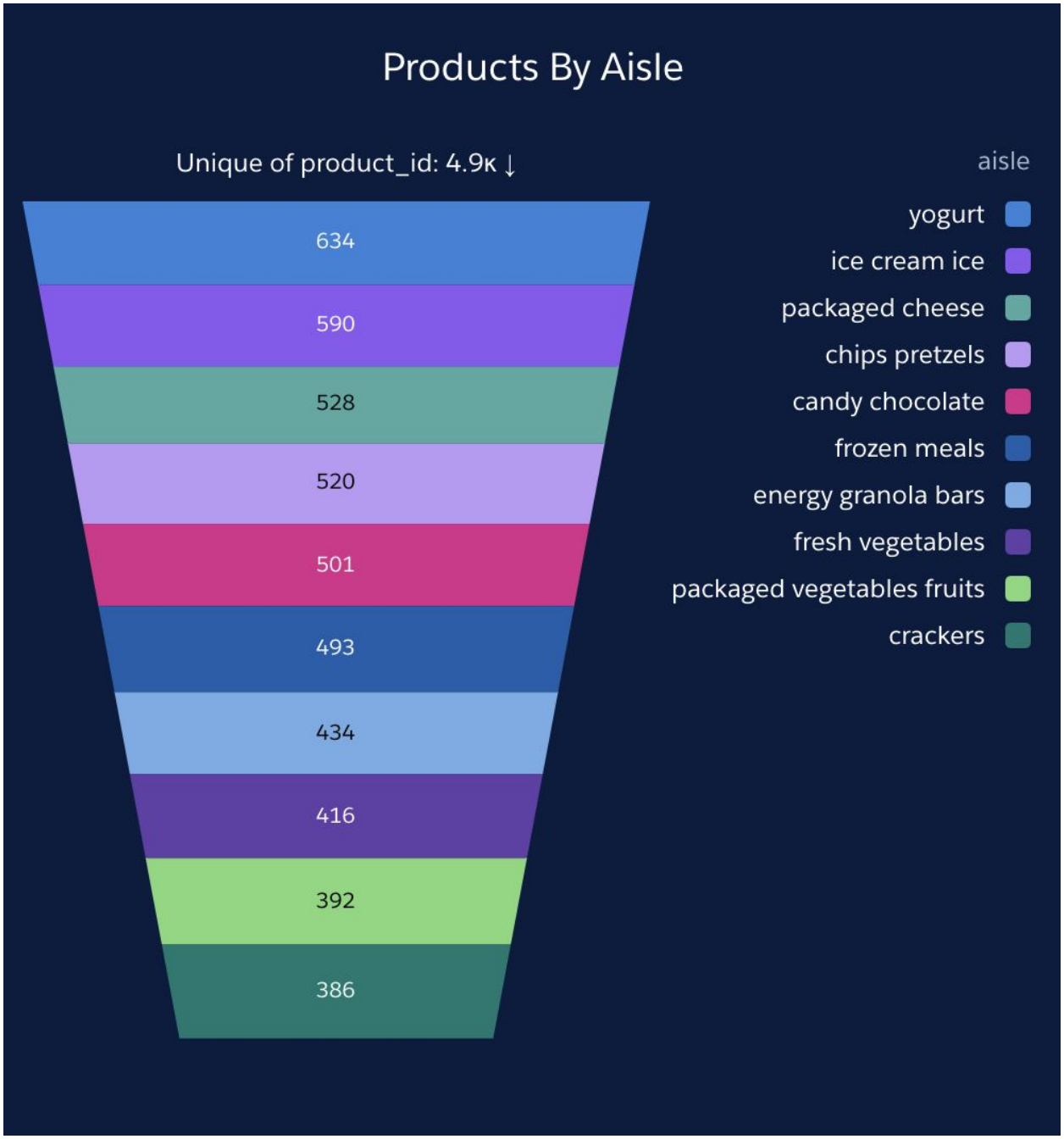
Treemap shows the number of products that were reordered from each department. Produce being the most popular department.



Products By Aisle

Top 10 Aisle having the most number of products.

Example:Instacart has 590 different types of Ice creams.



Products Ordered By Aisle

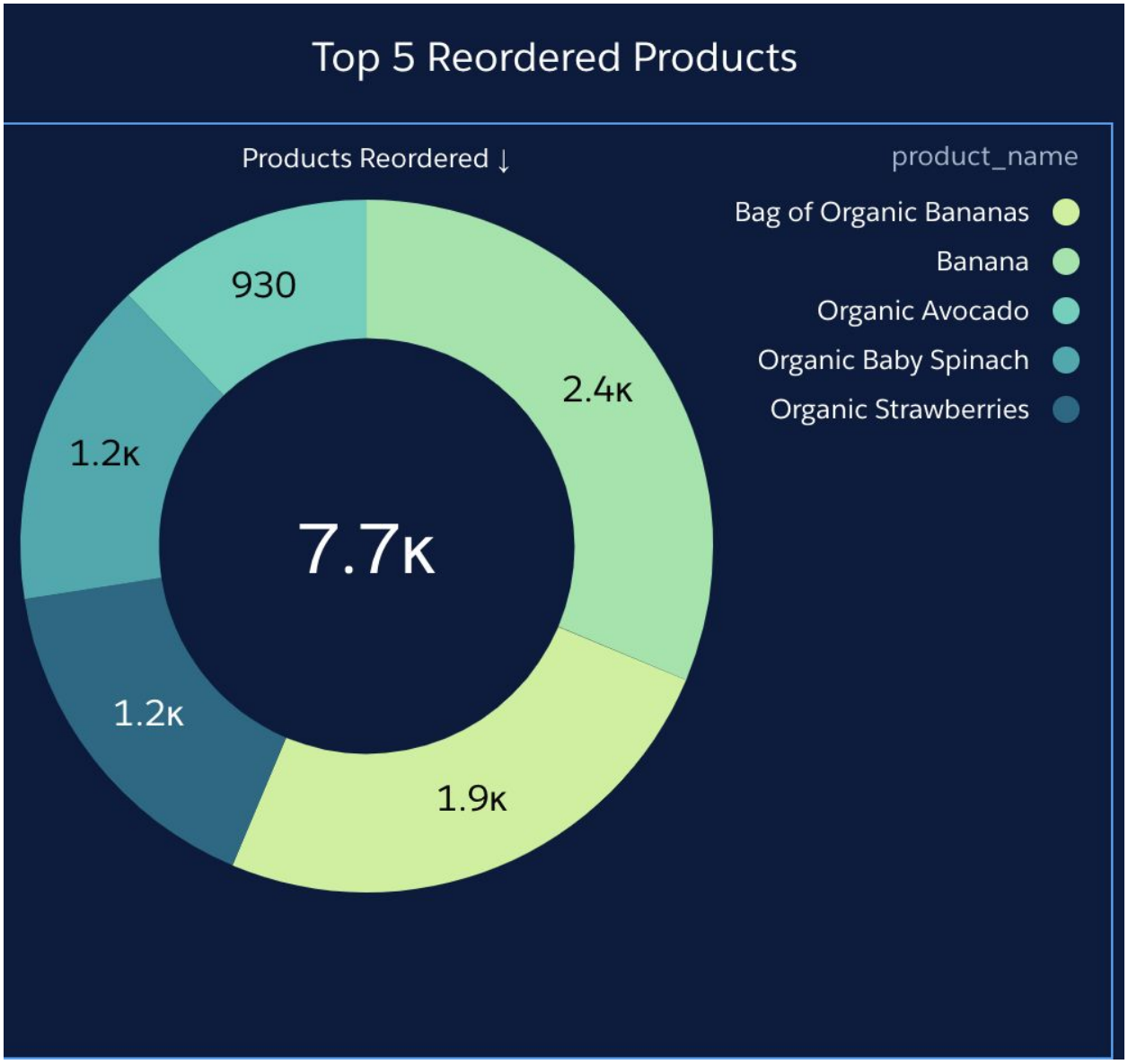
Represents total number of products ordered from each aisle and how much percent each aisle contributes towards total ordered products
Example: 21,922 products were ordered from the fresh fruits aisle which is 10.97% of the total products ordered.

Products Ordered By Aisle

aisle	Products Ordered...	Percentage Of Products Orde...
fresh fruits	21,922	10.97%
fresh vegetables	21,747	10.88%
packaged vegetables fruits	11,410	5.71%
yogurt	7,877	3.94%
packaged cheese	6,158	3.08%
water seltzer sparkling w...	5,265	2.63%
milk	4,747	2.37%
chips pretzels	4,430	2.22%
soy lactosefree	3,799	1.9%
refrigerated	3,435	1.72%

Top 5 Reordered Products

By knowing the products reordered, Instacart can make use of the reordered data to analyze the inventory stocks by ensuring the replenishments and proper scheduling of the products to increase internal productivity



First Product to add to cart

Represents order in which each product was added to the cart.

Bananas are added first to the cart 639 times.

From the table the top products that were added to the cart first seem to be from the produce department. The website can be designed in a way that all the produce products are aligned at the top.

First Product to Add to Cart

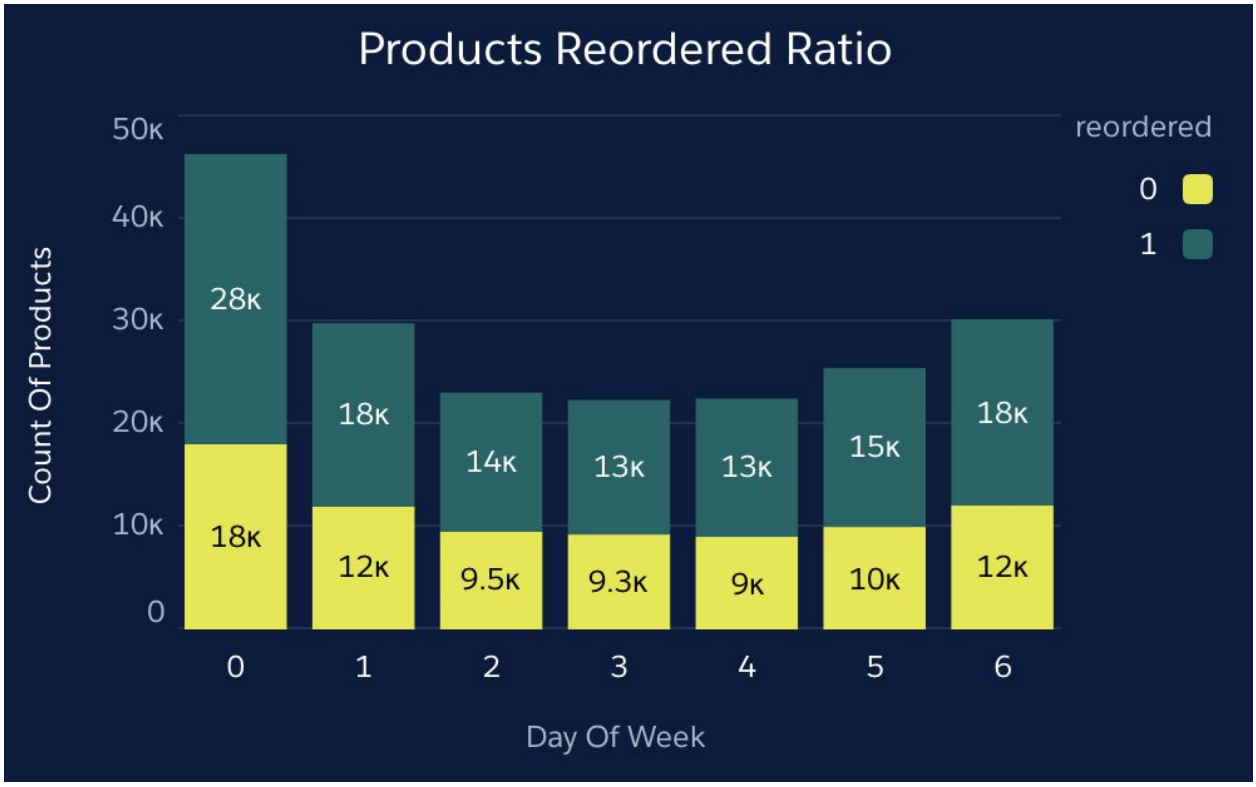
product_name	Count ↓
Banana	639
Bag of Organic Bananas	562
Organic Whole Milk	175
Organic Baby Spinach	137
Organic Avocado	121
Organic Strawberries	117
Organic Hass Avocado	114
Spring Water	104
Strawberries	104
Large Lemon	90
Sparkling Water Grapefruit	89
Organic Raspberries	79

Products reordered Ratio By Days Of Week

Represents Total number of products ordered on each day.

The Stacked Bar plot represents out of the ordered products how many of them were reordered previously.(1 represents reordered)

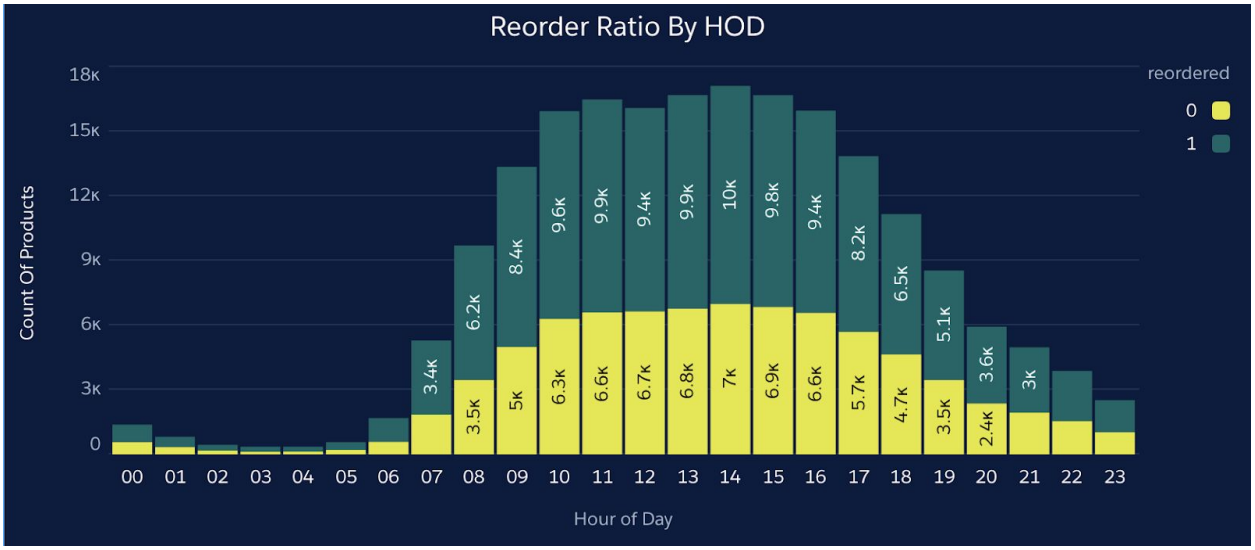
Looking at the plot 0, 1 and 2 has the most number of orders so assuming these days are Saturday, Sunday and Friday respectively.



Products reordered Ratio by Hour of Day

The Plot shows the number of products ordered every hour.

Most of the users prefer buying groceries between 10am and 4pm.



<u>Products Ordered DOW v/s HOD</u>

From the plot we can see the most number of orders placed were on Saturday noon at 3pm.

So we can give limited time offers during peak hours to maximize sales.For example on Saturday 12pm, Organic strawberries "buy one-get-one", offer expires in 2 hours.

Everyday the least number of orders were placed from 2am to 5am.

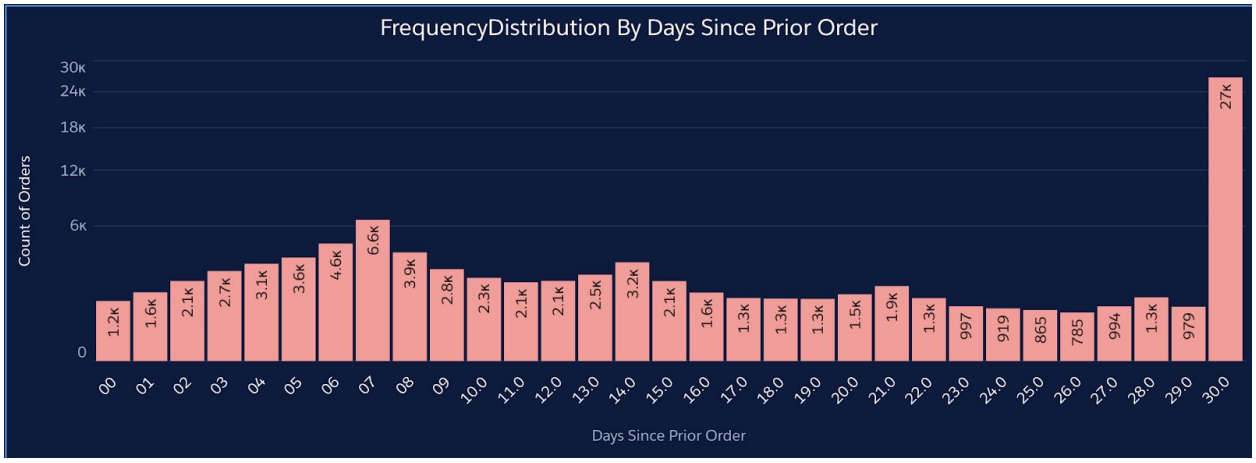
So the website can go under maintenance at this time to avoid any business loss.

Frequency Distribution By Day since Prior Order

Represents how frequently a user places order.It shows days since last order was placed.

From the plot it can be concluded that around 27k people order on a monthly basis while 6.6k people order on a weekly basis.

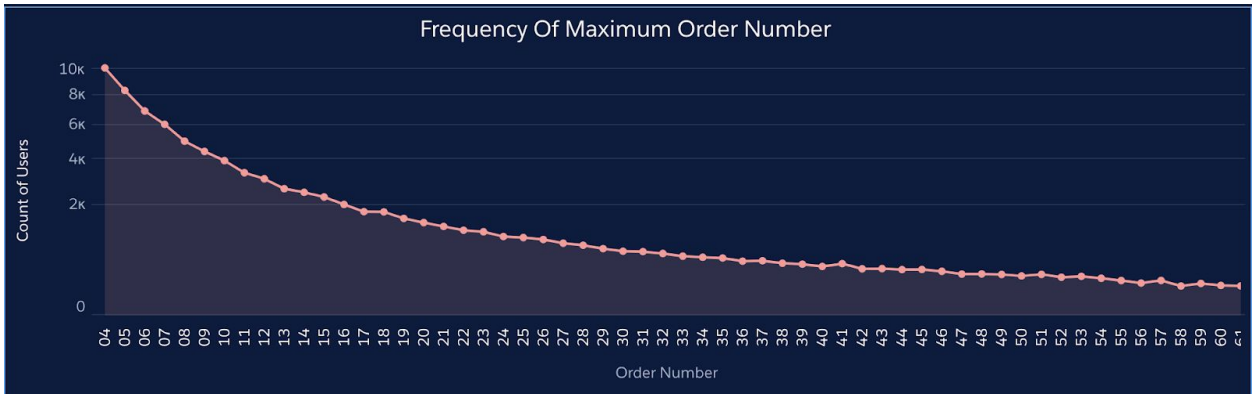
So Instacart can have monthly or weekly subscription plans where they get groceries delivered either monthly or weekly based on their selection for which there will be a fixed delivery fee or no fee at all depending on the customer lifetime value



Frequency Of Number of Orders Placed

Around 10k Users have ordered 4 times and the graph exponentially decreases where only 18 users have ordered 100 times

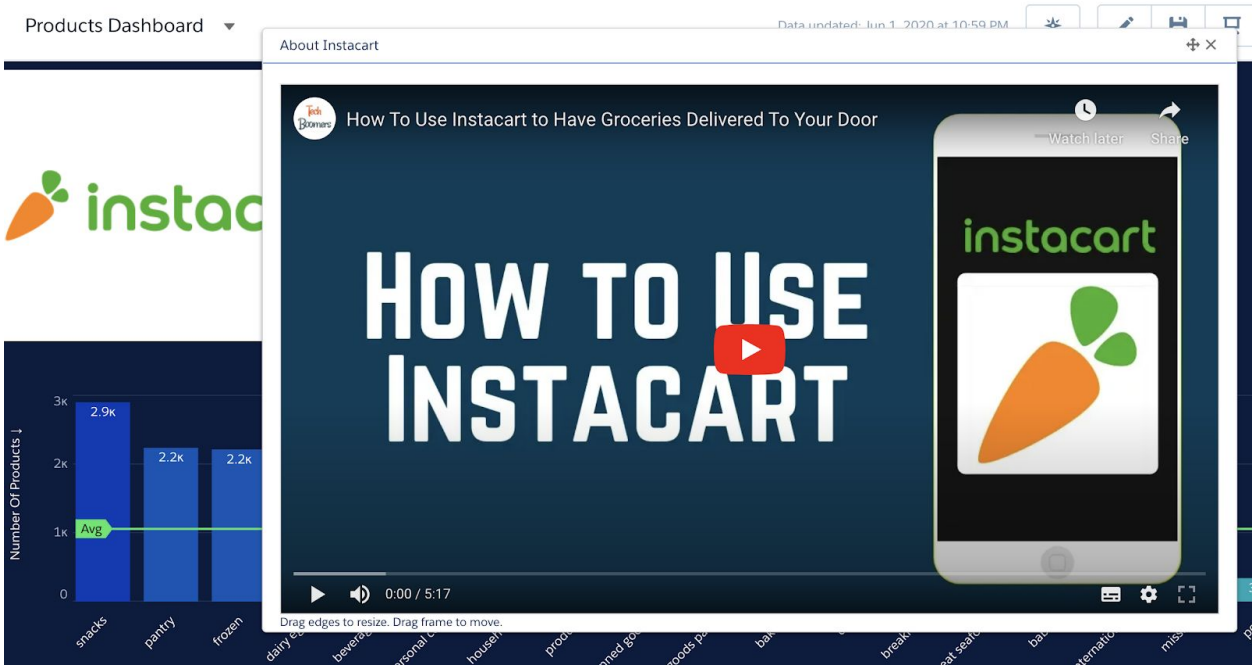
This helps to track down loyal customers



Onboarding

One of the features implemented to show more about INSTACART is when you click learn it opens a popup video of What is INSTACART and how to use it.

Learn

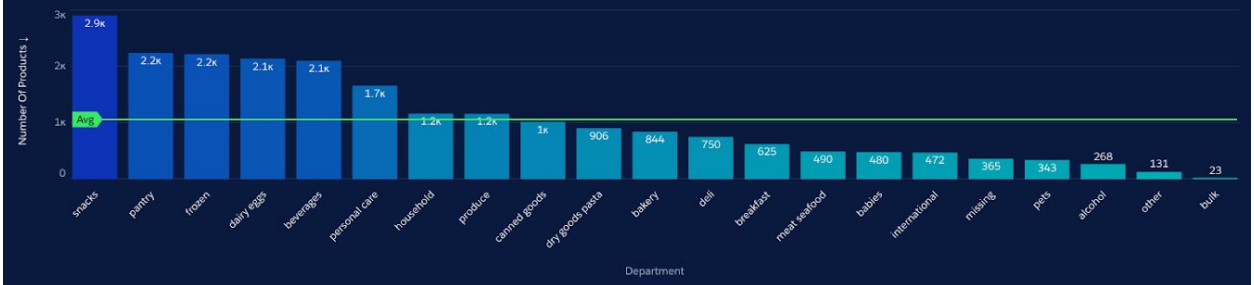

Products Dashboard
About Instacart
How To Use Instacart to Have Groceries Delivered To Your Door
Watch later
Share
HOW TO USE INSTACART
instacart
0:00 / 5:17
Drag edges to resize. Drag frame to move.
Number Of Products
3K
2K
1K
0
2.9K
2.2K
2.2K
Avg
snacks
pantry
frozen

Dashboards Links :
https://na172.lightning.force.com/wave/wave.app?tsid=02u5w000001vPEJ#dashboard/0FK5w000000WFXmGAO

https://na172.lightning.force.com/wave/wave.app?tsid=02u5w000001vPEJ#dashboard/0FK5w000000WFXNGA4/edit

Returning Users

Products By Department



Products By Aisle

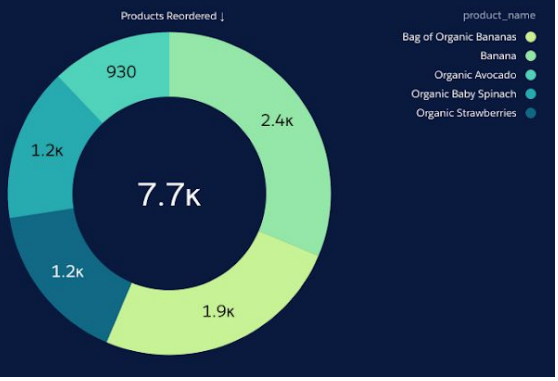
Products Ordered By Aisle

aisle	Products Ordered	Percentage Of Products Ordered
fresh fruits	21,922	10.97%
fresh vegetables	21,747	10.88%
packaged vegetables fruits	11,410	5.71%
yogurt	7,877	3.94%
packaged cheese	6,158	3.08%
water seltzer sparkling water	5,265	2.63%
milk	4,747	2.37%
chips pretzels	4,430	2.22%
soy lactosefree	3,799	1.9%
refrigerated	3,435	1.72%

Products Ordered By Department

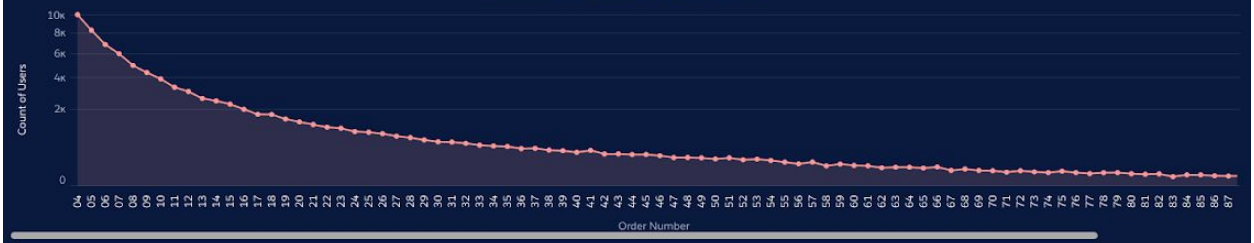
department	Products Ordered	Percentage Of Products Ordered
produce	59,276	29.65%
dairy eggs	31,455	15.74%
snacks	17,191	8.6%
beverages	16,676	8.34%
frozen	14,333	7.17%
pantry	11,664	5.84%
bakery	6,925	3.46%
canned goods	6,788	3.4%
deli	6,421	3.21%
dry goods pasta	5,542	2.77%

Top 5 Reordered Products



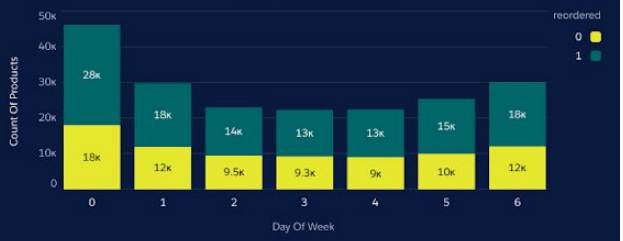
Insights

Frequency Of Maximum Order Number

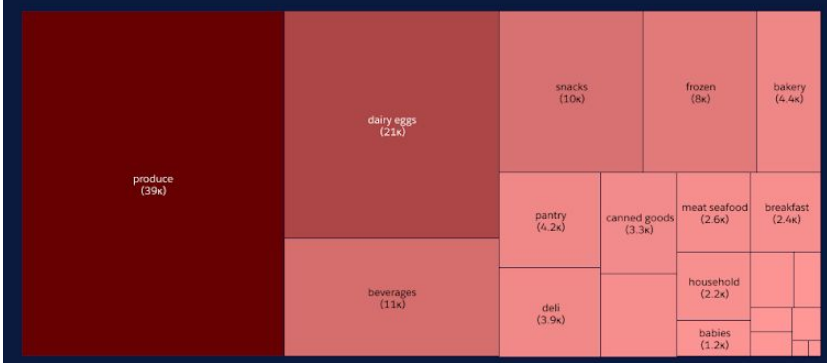


Products Ordered DOW v/s HOD

Products Reordered Ratio



Reordered Products By Department

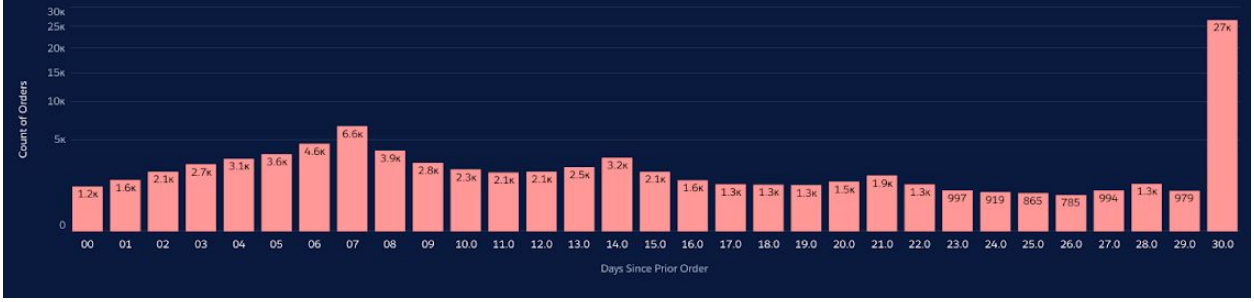


First Product to Add to Cart

product_name	Count
Banana	639
Bag of Organic Bananas	562
Organic Whole Milk	175
Organic Baby Spinach	137
Organic Avocado	121
Organic Strawberries	117
Organic Hass Avocado	114
Spring Water	104
Strawberries	104
Large Lemon	90
Sparkling Water Grapefruit	89
Organic Raspberries	79

Reorder Ratio By HOD

FrequencyDistribution By Days Since Prior Order



Pandas

Data Sampling

Performed random data sampling to select, manipulate and analyze a representative subset of **data**

Sampling Data

In [9]:
```
Sample = orders_df.sample(n=2400000)
```

In [10]:
```
Sample
```

Out[10]:

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order
987605	3295493	59366	prior	9	6	15	9.0
780817	1907936	46944	prior	1	4	11	NaN
2344348	2890915	141117	prior	1	0	21	NaN
700645	1732911	42208	prior	3	1	11	6.0
1064462	968898	64105	train	25	3	16	1.0
...	...	...	...	...	...	...	...
512810	1915835	30936	train	14	0	10	30.0
1738717	1035513	104381	prior	6	2	23	6.0
2785484	311754	167967	prior	21	3	10	7.0
1335696	3285642	80232	prior	79	2	12	5.0
3293400	2968000	198491	prior	10	3	21	13.0

2400000 rows × 7 columns

In [11]:
```
orders_sample = Sample.to_csv ("/Users/Priyanka Malpekar/Downloads/Dataset/Order_Sample.csv", index=False)
```

In [15]:
```
Sample_prior = orders_prior_sample_df.sample(n=2400000)
```

In [16]:
```
Sample_prior
```

Out[16]:

	order_id	product_id	add_to_cart_order	reordered
24770702	2612414	21158	12	1
22105956	2331606	17872	3	1
23318290	2459416	3007	13	0
2251989	237542	21616	4	0
4629388	488547	4799	16	1
...	...	...	...	...
11248555	1187620	41757	1	1
2954642	311985	47766	1	1
27669174	2917946	39427	10	1
23614118	2490512	19816	6	1
21707658	2289753	329	10	1

2400000 rows × 4 columns

In [17]:
```
orders_prior_sample = Sample_prior.to_csv ("/Users/Priyanka Malpekar/Downloads/Dataset/Order_prior_Sample.csv", index=False)
```

Slicing Data

```
In [4]: orders_train = orders_df.loc[orders_df['eval_set'] == 'train']

In [5]: orders_train
```

Out[5]:

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order
10	1187899	1	train	11	4	8	14.0
25	1492625	2	train	15	1	11	30.0
49	2196797	5	train	5	0	11	6.0
74	525192	7	train	21	2	11	6.0
78	880375	8	train	4	1	14	10.0
...	...	...	...	...	...	...	...
3420838	2585586	206199	train	20	2	16	30.0
3420862	943915	206200	train	24	6	19	6.0
3420924	2371631	206203	train	6	4	19	30.0
3420933	1716008	206205	train	4	1	16	10.0
3421082	272231	206209	train	14	6	14	30.0

131209 rows × 7 columns

Exploring Data Analysis

Exploring the data and gaining data-driven insights

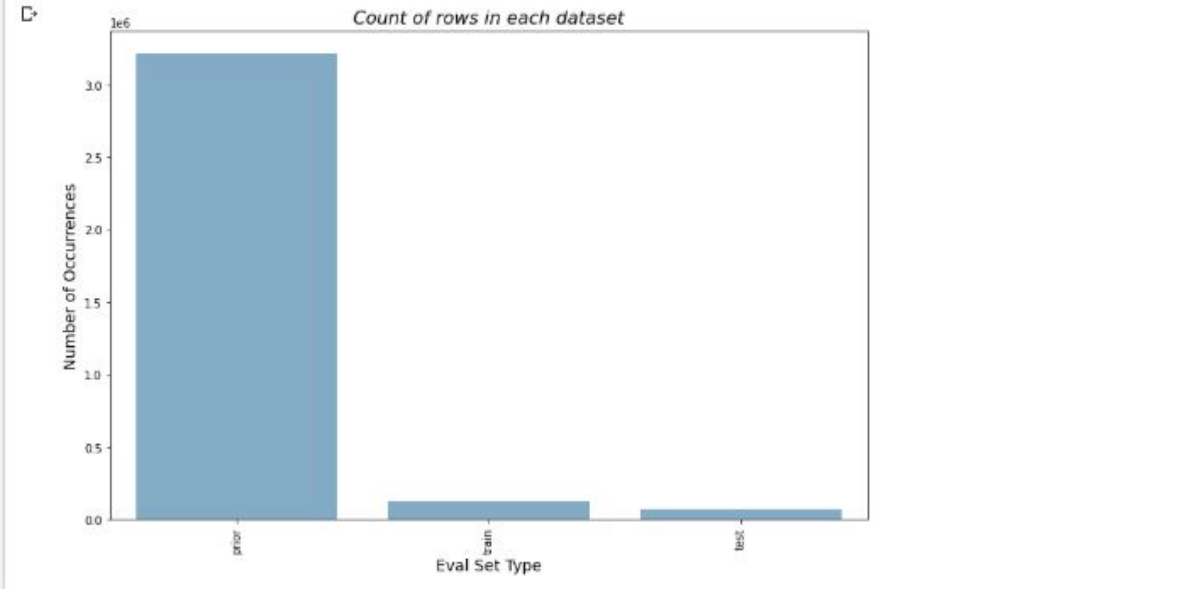
1) **Eval set Count**

```
def get_unique_count_eval_set(x):
    return len(np.unique(x))

Count_eval_set= orders_df.groupby("eval_set")["user_id"].aggregate(get_unique_count_eval_set)
Count_eval_set
```

```
eval_set
prior    206209
test      75000
train    131209
Name: user_id, dtype: int64
```

So there are 206,209 customers in total. Out of which, the purchases of 131,209 customers are given as a train set and we need to predict for the rest 75,000 customers.

```
count_rows = orders_df.eval_set.value_counts()

plt.figure(figsize=(12,8))
sns.barplot(count_rows.index, count_rows.values, alpha=0.6, color=color[0])
plt.ylabel('Number of Occurrences', fontsize=14)
plt.xlabel('Eval Set Type', fontsize=14)
plt.title('Count of rows in each dataset', fontsize=16, fontstyle='italic')
plt.xticks(rotation='vertical')
plt.show()
```



2) Frequency of Products

```
[ ] Count_Products_df = order_products_prior_df['product_name'].value_counts().reset_index().head(20)
    Count_Products_df.columns = ['product_name', 'Count_of_Products']
    Count_Products_df
```

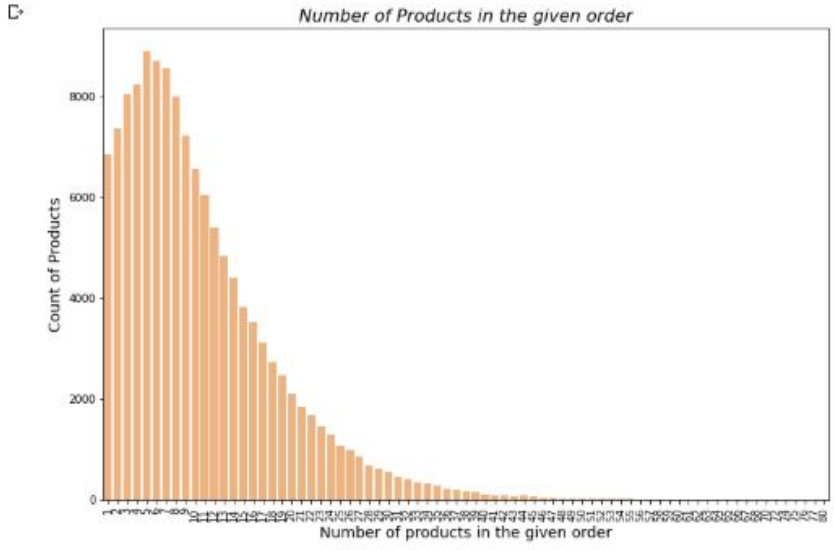
	product_name	Count_of_Products
0	Banana	472565
1	Bag of Organic Bananas	379450
2	Organic Strawberries	264683
3	Organic Baby Spinach	241921
4	Organic Hass Avocado	213584
5	Organic Avocado	176815
6	Large Lemon	152657
7	Strawberries	142951
8	Limes	140627
9	Organic Whole Milk	137905
10	Organic Raspberries	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

Most of them are organic products, the majority of them being fruits.
Most sold product is **Banana**.

3) Number of Products in the given order

Number of Products in the given order

```
[ ] No_of_Products_df = order_products_train_df.groupby("order_id")["add_to_cart_order"].aggregate("max").reset_index()
    cnt_srs = No_of_Products_df.add_to_cart_order.value_counts()

    plt.figure(figsize=(12,8))
    sns.barplot(cnt_srs.index, cnt_srs.values, alpha=0.6, color=color[1] )
    plt.ylabel('Count of Products', fontsize=14)
    plt.xlabel('Number of products in the given order', fontsize=14)
    plt.title('Number of Products in the given order', fontstyle='italic', fontsize=16)
    plt.xticks(rotation='vertical')
    plt.show()
```

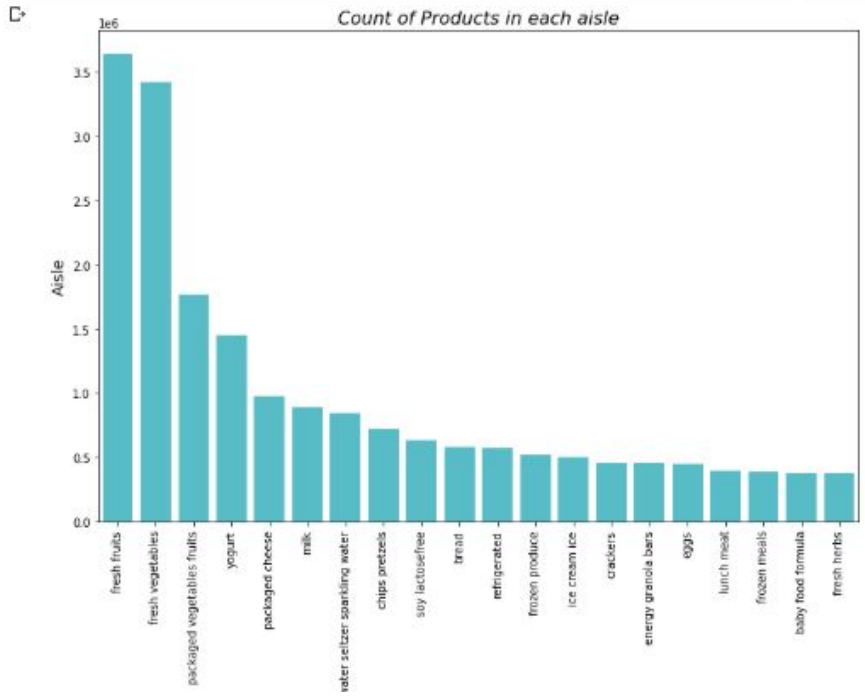


Represents number of products a user purchases in one order. On an Average people buy 11 products at a time.

4) Count of Products in each Aisle

Count of Products in each Aisle

```
[ ] Count_Products = order_products_prior_df['aisle'].value_counts().head(20)
    plt.figure(figsize=(12,8))
    sns.barplot(Count_Products.index, Count_Products.values, alpha=0.8, color=color[9])
    plt.ylabel('Aisle', fontsize=14)
    plt.xlabel('Number of Occurrences', fontsize=14)
    plt.title('Count of Products in each aisle', fontsize=16, fontstyle='italic')
    plt.xticks(rotation='vertical')
    plt.show()
```

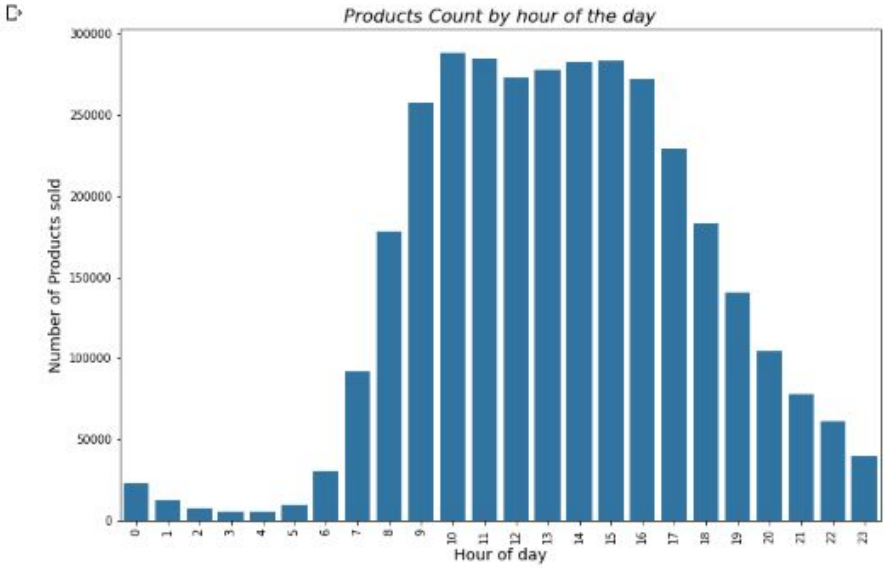


The top aisle is **fresh fruits**.

5) Order Count by Hour of Day

Order Count by hour of the day

```
[ ] plt.figure(figsize=(12,8))
    sns.countplot(x="order_hour_of_day", data=orders_df, color=color[0])
    plt.ylabel('Number of Products sold', fontsize=14)
    plt.xlabel('Hour of day', fontsize=14)
    plt.title('Products Count by hour of the day', fontsize=16, fontstyle='italic')
    plt.xticks(rotation='vertical')
    plt.show()
```

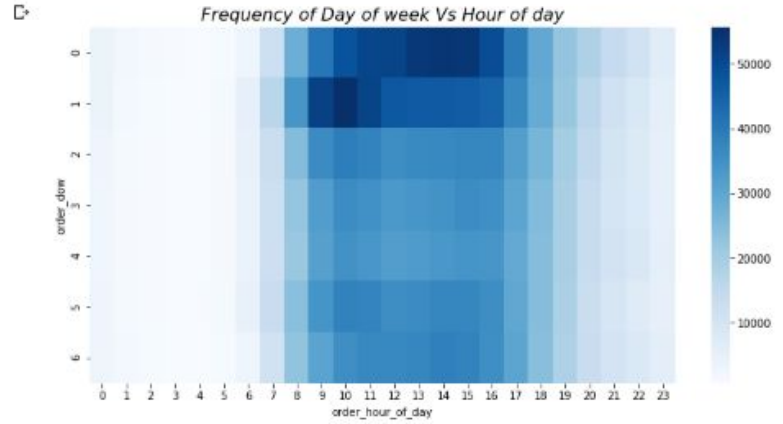


Orders are placed by customers during **mid-Mornings** and late **afternoons**.

6) Frequency of ‘Day of Week’ vs ‘Hour of Day’

Frequency of Day of week Vs Hour of day

```
[ ] grouped_df = orders_df.groupby(["order_dow", "order_hour_of_day"])["order_number"].aggregate("count").reset_index()
    grouped_df = grouped_df.pivot('order_dow', 'order_hour_of_day', 'order_number')

    plt.figure(figsize=(12,6))
    sns.heatmap(grouped_df, cmap="Blues")
    plt.title("Frequency of Day of week Vs Hour of day", fontsize=16, fontstyle='italic')
    plt.show()
```



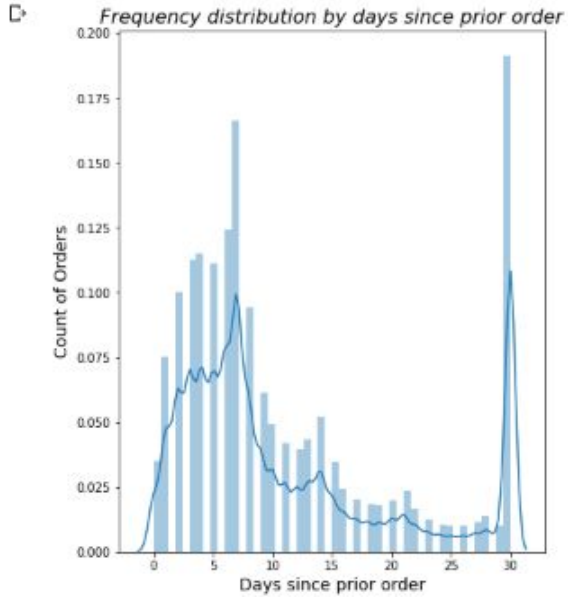
Saturday evenings and **Sunday mornings** are the prime time for orders.

7) Frequency distribution by days since prior order

Frequency distribution by days since prior order

+ Code +

```
[ ]  plt.figure(figsize=(14,8))
     plt.subplot(121)
     dist = sns.distplot(orders_df["days_since_prior_order"], color = color[0])
     dist.set_xlabel("Days since prior order", fontsize=14)
     dist.set_ylabel("Count of Orders", fontsize=14)
     dist.set_title("Frequency distribution by days since prior order", fontsize=16, fontstyle='italic')

     plt.show()
```



Customers order once in every **week** or once in every **month**. This can be seen by looking at the peak for 7 days or at the peak for 30 days.
Few smaller peaks can be seen at weekly intervals.

8) Products reordered in prior and train set

Products reordered in prior and train set

```
[ ] # percentage of re-orders in prior set #
    prior = order_products_prior_df.reordered.sum() / order_products_prior_df.shape[0]
    print("Percentage of reorders in prior set")
    print(prior)

    # percentage of re-orders in train set #
    train = order_products_train_df.reordered.sum() / order_products_train_df.shape[0]
    print("Percentage of reorders in train set")
    print(train)
```

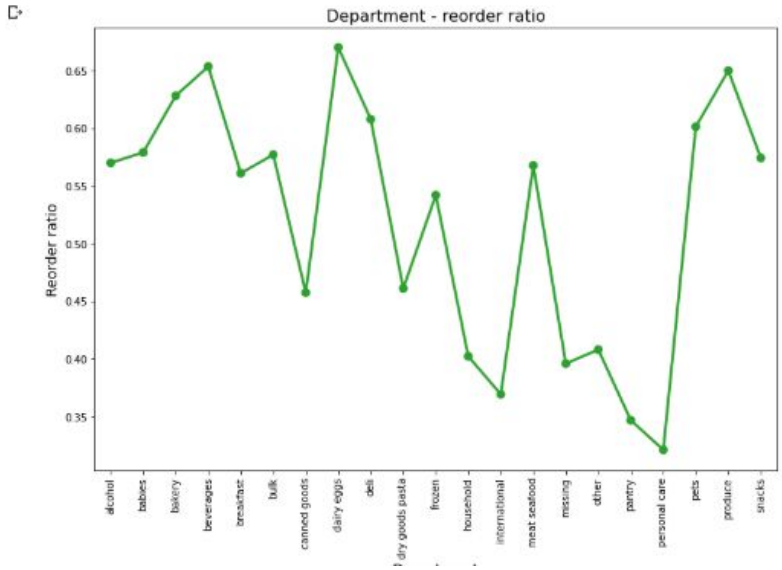
```
Percentage of reorders in prior set
0.5896974667922161
Percentage of reorders in train set
0.5985944127509629
```

9) Department reorder ratio

Department having most reordered products

Department Reorder ratio

```
[ ] dept_df = order_products_prior_df.groupby(["department"])["reordered"].aggregate("mean").reset_index()

    plt.figure(figsize=(12,8))
    sns.pointplot(dept_df['department'].values, dept_df['reordered'].values, alpha=0.8, color=color[2])
    plt.ylabel('Reorder ratio', fontsize=14)
    plt.xlabel('Department', fontsize=14)
    plt.title("Department - reorder ratio", fontsize=16)
    plt.xticks(rotation='vertical')
    plt.show()
```

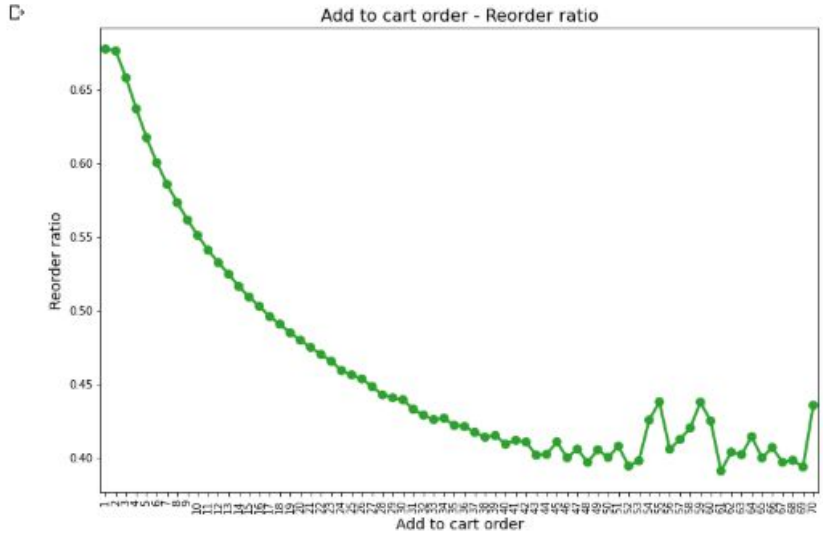


10) Add to cart reorder ratio

The products that are added to the cart first are more likely to be reordered again.

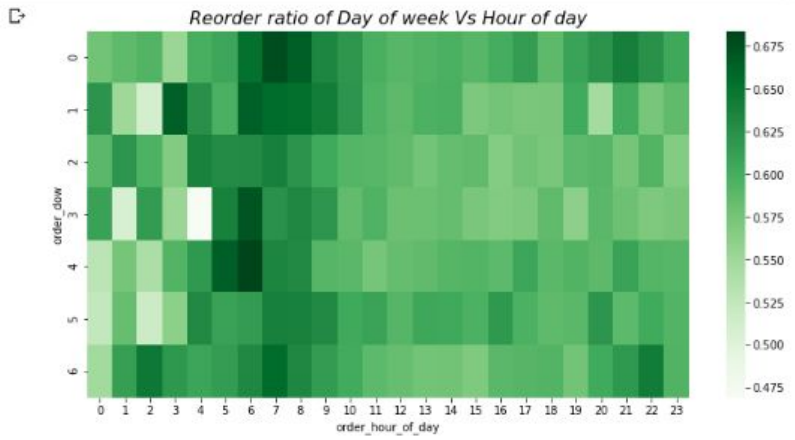
Add to cart - Reorder ratio

```
[ ] order_products_prior_df["add_to_cart_order_mod"] = order_products_prior_df["add_to_cart_order"].copy()
    order_products_prior_df["add_to_cart_order_mod"].loc[order_products_prior_df["add_to_cart_order_mod"]>70] = 70
    AddToCart_df = order_products_prior_df.groupby(["add_to_cart_order_mod"])["reordered"].aggregate("mean").reset_index()

    plt.figure(figsize=(12,8))
    sns.pointplot(AddToCart_df['add_to_cart_order_mod'].values, AddToCart_df['reordered'].values, alpha=0.8, color=color[2])
    plt.ylabel('Reorder ratio', fontsize=14)
    plt.xlabel('Add to cart order', fontsize=14)
    plt.title("Add to cart order - Reorder ratio", fontsize=16)
    plt.xticks(rotation='vertical')
    plt.show()
```



11) Reorder ratio by day of week vs hour of day

```
[ ] grouped_df = order_products_train_df.groupby(["order_dow", "order_hour_of_day"])["reordered"].aggregate("mean").reset_index()
    grouped_df = grouped_df.pivot('order_dow', 'order_hour_of_day', 'reordered')

    plt.figure(figsize=(12,6))
    sns.heatmap(grouped_df, cmap = "Greens")
    plt.title("Reorder ratio of Day of week Vs Hour of day", fontsize=16, fontstyle='italic')
    plt.show()
```



*Reorder ratios are quite high during the **early mornings**.*

Customer Lifetime Value

Customer Lifetime Value (CLV) helps you figure out your ideal customers that will bring the most potential at generating your revenue. By having detailed information about what your best customers desire, you can target that specific market.

Generating random Price and Profit for each user

```
[ ] import random
    clv_df['Total Price'] = np.random.randint(100,1200,size=90269)/10
```

```
[ ] clv_df['Total Profit'] = np.random.randint(2,9,size=90269)/10
```

Calculating CLV by using the following Formula

Customer Value = average purchase value*average purchase frequency rate
Considering average customer lifespan is 1 year and the data set is also for 1 year
Customer lifetime value = average customer lifespan*customer value

```
[ ] clv_df['Customer Lifetime value']=(clv_df['Total Price']*clv_df['Total Profit']*clv_df['Total Visits']).astype(int)
```

```
[ ] clv_df
```

	user_id	Total Visits	Total Price	Total Profit	Customer Lifetime value
0	1	11	97.0	0.8	853
1	2	15	13.9	0.5	104
2	5	5	28.5	0.8	114
3	7	21	40.6	0.6	511
4	8	4	100.9	0.7	282
...	...	...	...	...	...
90264	206199	20	28.1	0.6	337
90265	206200	24	102.3	0.3	736
90266	206203	6	104.9	0.5	314
90267	206205	4	21.8	0.4	34
90268	206209	14	14.9	0.5	104

90269 rows × 5 columns

Segmenting the Customer based on CLV value as Regular, Silver, Gold, Platinum

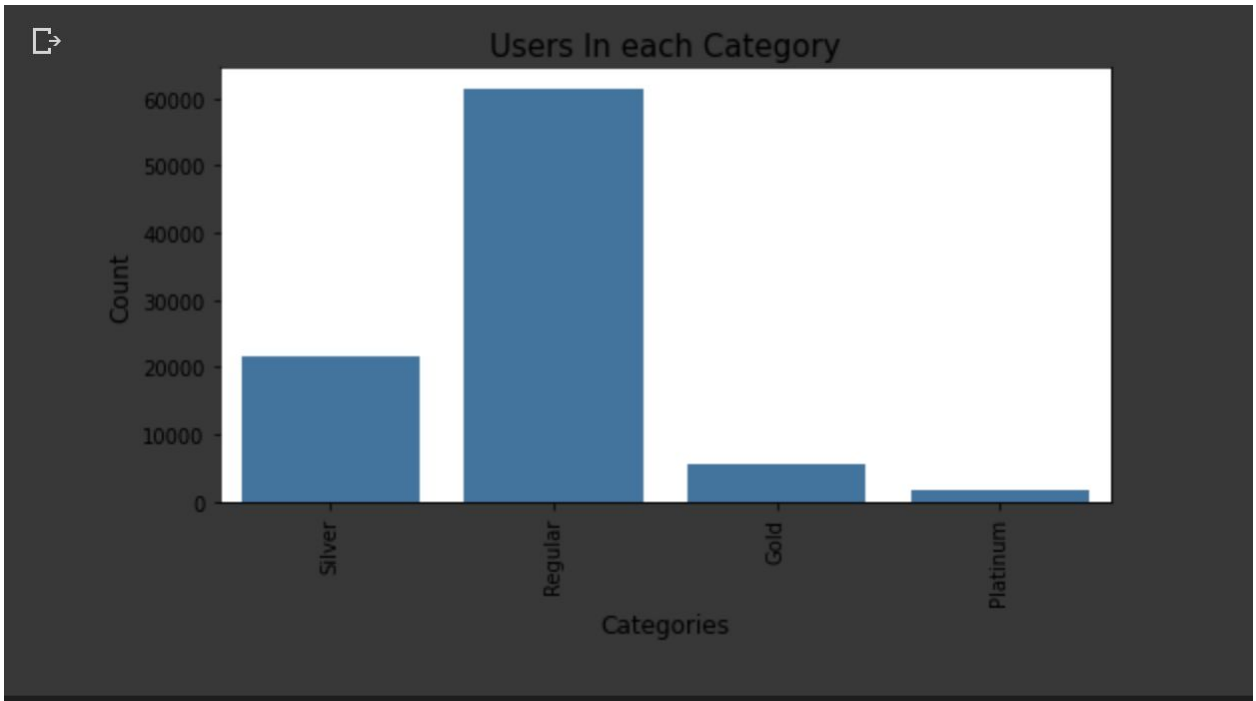
```
] Membership = [clv_df['Customer Lifetime value'].between(0, 500),
                clv_df['Customer Lifetime value'].between(500, 1500),
                clv_df['Customer Lifetime value'].between(1500, 3000),
                clv_df['Customer Lifetime value'].between(3000, 10000)]
  values = ['Regular', 'Silver', 'Gold', 'Platinum']

  clv_df['CLV_Category'] = np.select(Membership, values, 'Platinum' )
```

```
[ ] clv_df.head(100)
```

	user_id	Total Visits	Total Price	Total Profit	Customer Lifetime value	CLV_Category
0	1	11	97.0	0.8	853	Silver
1	2	15	13.9	0.5	104	Regular
2	5	5	28.5	0.8	114	Regular
3	7	21	40.6	0.6	511	Silver
4	8	4	100.9	0.7	282	Regular
...	...	...	...	...	...	...
95	215	6	22.2	0.7	93	Regular
96	217	12	53.3	0.8	511	Silver
97	219	34	23.0	0.8	625	Silver
98	222	71	116.9	0.7	5809	Platinum
99	223	50	24.4	0.8	976	Silver

100 rows × 6 columns

Plotting number of Users in each Category.




```
[ ]  clv_df['CLV_Category'].value_counts()
```

```
Regular     61534
Silver      21561
Gold         5460
Platinum     1714
Name: CLV_Category, dtype: int64
```

RFM Modeling

RFM (Recency, Frequency, Monetary) analysis is a customer segmentation technique that uses past purchase behavior to divide customers into groups. RFM helps divide customers into various categories or clusters to identify customers who are more likely to respond to promotions and also for future personalization services.

- RECENCY (R): Days since last purchase

- FREQUENCY (F): Total number of purchases
- MONETARY VALUE (M): Total money this customer spent.

Getting Recency,Frequency from the dataset and random Monetary with random function.

```
[ ]  Merge_df = recency_df.merge(frequency_df,on='user_id')
     Merge_df.drop('Total Price',axis=1,inplace=True)
     Merge_df
```

	user_id	Recency	Frequency	Monetary
0	1	19	11	12.8
1	2	16	15	29.6
2	3	12	13	82.3
3	4	17	6	89.6
4	5	11	5	41.8
...	...	...	...	...
206204	206205	16	4	82.1
206205	206206	3	68	55.8
206206	206207	14	17	62.8
206207	206208	7	50	118.7
206208	206209	18	14	104.4

206209 rows × 4 columns

Creating Customer Segments with RFM Model

Calculating RFM using Quartiles.

Assigning a score from 1 to 4 to Recency, Frequency and Monetary. Four is the highest value, and one is the lowest value. A final RFM score is calculated by combining individual RFM score numbers.

```
[ ]  def RScore(x,p,d):
         if x <= d[p][0.25]:
             return 4
         elif x <= d[p][0.50]:
             return 3
         elif x <= d[p][0.75]:
             return 2
         else:
             return 1
     def FMScore(x,p,d):
         if x <= d[p][0.25]:
             return 1
         elif x <= d[p][0.50]:
             return 2
         elif x <= d[p][0.75]:
             return 3
         else:
             return 4
```

```
[ ]  rfm_segmentation = Merge_df
     rfm_segmentation['R_Quartile'] = rfm_segmentation['Recency'].apply(RScore, args=('Recency',quantiles,))
     rfm_segmentation['F_Quartile'] = rfm_segmentation['Frequency'].apply(FMScore, args=('Frequency',quantiles,))
     rfm_segmentation['M_Quartile'] = rfm_segmentation['Monetary'].apply(FMScore, args=('Monetary',quantiles,))
```

```
[ ] rfm_segmentation
```

	user_id	Recency	Frequency	Monetary	R_Quartile	F_Quartile	M_Quartile	RFMScore	Total Score
0	1	19	11	12.8	2	3	1	231	6
1	2	16	15	29.6	2	3	1	231	6
2	3	12	13	82.3	3	3	3	333	9
3	4	17	6	89.6	2	1	3	213	6
4	5	11	5	41.8	3	1	2	312	6
...	...	...	...	...	...	...	...	...	...
206204	206205	16	4	82.1	2	1	3	213	6
206205	206206	3	68	55.8	4	4	2	442	10
206206	206207	14	17	62.8	3	3	2	332	8
206207	206208	7	50	118.7	4	4	4	444	12
206208	206209	18	14	104.4	2	3	4	234	9

206209 rows × 9 columns

Best Recency score = 4: most recent purchase.
Best Frequency score = 4: most quantity purchase.
Best Monetary score = 4: spent the most.

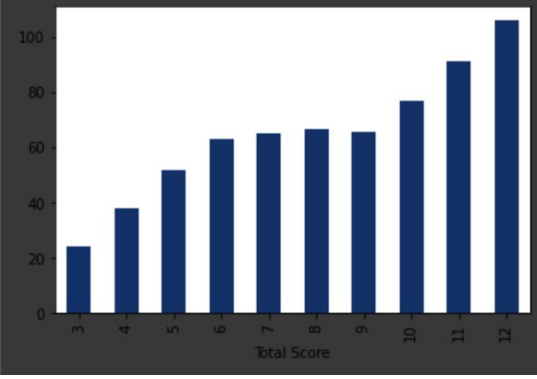
Getting Some Customer Insights

```
[ ] print("Best Customers: ",len(rfm_segmentation[rfm_segmentation['RFMScore']=='444']))
    print('Loyal Customers: ',len(rfm_segmentation[rfm_segmentation['F_Quartile']==4]))
    print("Big Spenders: ",len(rfm_segmentation[rfm_segmentation['M_Quartile']==4]))
    print('Almost Lost: ', len(rfm_segmentation[rfm_segmentation['RFMScore']=='244']))
    print('Lost Customers: ',len(rfm_segmentation[rfm_segmentation['RFMScore']=='111']))
```

```
Best Customers:  7904
Loyal Customers:  50731
Big Spenders:  51529
Almost Lost:  190
Lost Customers:  7235
```

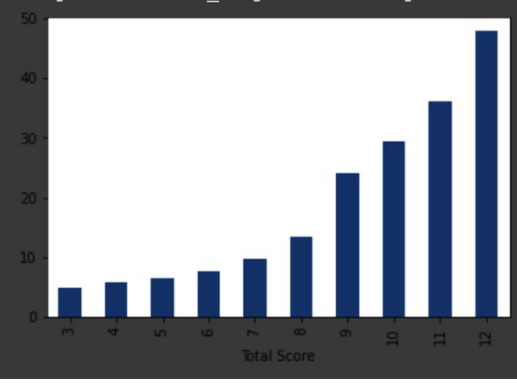
```
rfm_segmentation.groupby('Total Score').agg('Monetary').mean().plot(kind='bar', colormap='Blues_r')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa927c25828>
```



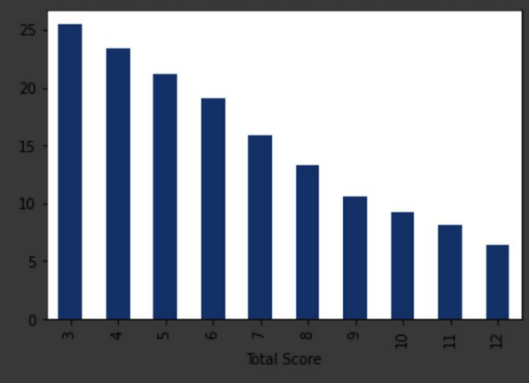

```
[ ] rfm_segmentation.groupby('Total Score').agg('Frequency').mean().plot(kind='bar', colormap='Blues_r')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa927601dd8>
```




```
[ ] rfm_segmentation.groupby('Total Score').agg('Recency').mean().plot(kind='bar', colormap='Blues_r')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa927580588>
```



Customers Insights:

Users with low RFM Score rarely contribute to the sales which indicates those are target customers for promotions and advertisements

Some of the promotions can be :

- Free delivery on first order
- Give discount coupon for next purchase
- Send promotional emails
- Giving out samples on orders above a particular amount

Users belonging to the Platinum group or having high RFM score are most Loyal customers and to retain them continually special offers exclusively for them should be given like

- Loyalty Program
- Offering Free Delivery on all orders
- Offering rewards to new Customers referrals
- Having point system to redeem

Prediction Model - Logistic Regression

What is the next product in Customer's Shopping cart?

Logistic Regression is used when the dependent variable(target) is categorical.
We are using this model to determine the probability of products reordered by the customers such as 0/1.
eg: To predict whether the product is reordered (1) or not(0) ?
This regression has only 2 possible outcomes

Preliminary analysis and modeling

```
train_carts = (order_products_train_df.groupby('user_id',as_index=False)
                                      .agg({'product_id':(lambda x: set(x))})
                                      .rename(columns={'product_id':'latest_cart'}))

df_X = df_X.merge(train_carts, on='user_id')
df_X['in_cart'] = (df_X.apply(lambda row: row['product_id'] in row['latest_cart'], axis=1).astype(int))

df_X.head()
```

	product_id	user_id	user_product_total_orders	latest_cart	in_cart
0	1	1494	1	{44560}	0
1	130	1494	1	{44560}	0
2	1648	1494	1	{44560}	0
3	2745	1494	2	{44560}	0
4	4377	1494	1	{44560}	0

```
#Here I see class imbalance
df_X['in_cart'].value_counts()
```

```
0    1274115
1     257327
Name: in_cart, dtype: int64
```

User Product Features

```
user_prod_features = ['user_product_avg_days_since_prior_order',
                      'user_product_avg_order_dow',
                      'user_product_avg_order_hour_of_day']

user_prod_features_df = (order_products_prior_df.groupby(['product_id','user_id'],as_index=False) \
                                                .agg(OrderedDict(
                                                     [('days_since_prior_order','mean'),
                                                     ('order_dow','mean'),
                                                     ('order_hour_of_day','mean')])))

user_prod_features_df.columns = ['product_id','user_id'] + user_prod_features
user_prod_features_df.head()
```

	product_id	user_id	user_product_avg_days_since_prior_order	user_product_avg_order_dow	user_product_avg_order_hour_of_day
0	1	1494	6.0	2.0	8.0
1	1	1540	7.0	2.0	0.0
2	1	3029	NaN	2.0	20.0
3	1	3904	NaN	1.0	14.0
4	1	4122	8.0	2.0	7.0

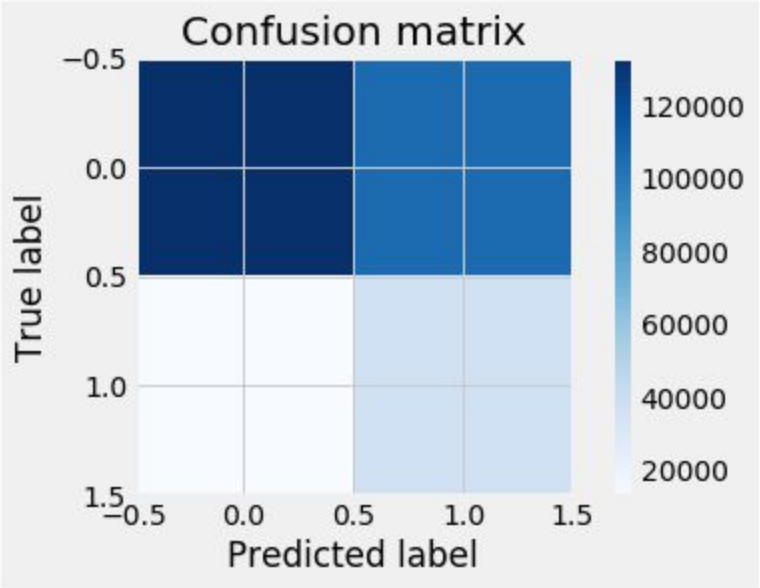
Adding coefficients

```
coefficients = pd.DataFrame(lr_10x.coef_, columns = X_tr.columns)
coefficients = np.exp(coefficients)
coefficients.T
```

	0
user_product_total_orders	1.657417
product_total_orders	1.287423
product_avg_add_to_cart_order	0.913645
user_total_orders	0.905199
user_avg_cartsize	1.036869
user_total_products	0.993214
user_avg_days_since_prior_order	0.994877
user_product_avg_add_to_cart_order	0.943689
user_product_order_freq	1.230198
product_avg_order_dow	0.957915
product_avg_order_hour_of_day	1.029555
product_avg_days_since_prior_order	0.988776
user_avg_order_dow	0.990917
user_avg_order_hour_of_day	1.001688
user_product_avg_days_since_prior_order	0.998982
user_product_avg_order_dow	0.977710
user_product_avg_order_hour_of_day	1.013412
product_total_orders_delta_per_user	0.776765
product_avg_add_to_cart_order_delta_per_user	0.968163
product_avg_order_dow_per_user	0.979754
product_avg_order_hour_of_day_per_user	1.015929
product_avg_days_since_prior_order_per_user	0.989783

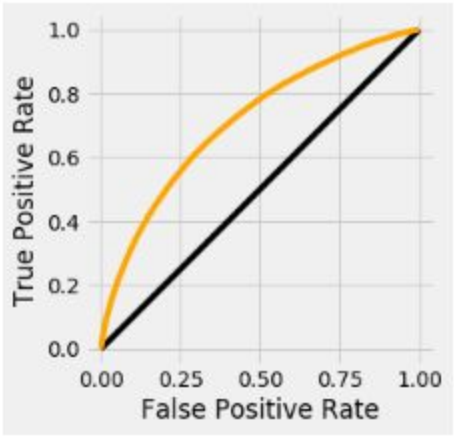
Calculating Precision and Recall

```
Training Data Accuracy: 0.59
Test Data Accuracy:     0.59
[[133228 105900]
 [ 12926  36910]]

Precision:              0.26
Recall:                 0.74
```



With Precision value of 0.26 , our product shall predict 26% of the reordered products and with the recall value of 0.74 determined how many true and false positives our model could predict.

```
from sklearn.metrics import roc_curve, auc

y_score = lr_10x.predict_proba(X_te)[:,1]

fpr, tpr,_ = roc_curve(y_te, y_score)
roc_auc = auc(fpr, tpr)

plt.figure()
# Plotting our Baseline..
plt.plot([0,1],[0,1], linestyle='-', color = 'black')
plt.plot(fpr, tpr, color = 'orange')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.gca().set_aspect('equal', adjustable='box')
```



Strength and weakness of each Tool

PLATFORM	STRENGTH	WEAKNESS
XSV	● No limit on dataset loading ● Effective to perform joins,slicing and storing the result ● Creating Indexes on large datasets to make them easily accessible ● Much faster as compared to other tools	● Cannot handle data cleaning processes like null value handling, date formatting
TRIFACTA	● Data profiling ● Data cleaning ● Good UI ● Data transformation ● Creation of recipes ● Job scheduling	● File limit of 100 MB

SNOWFLAKE	• Allows you to query semi-structure JSON/XML data • Recover Snowflake Object using undrop	• File limit of 50 MB
EINSTEIN ANALYTICS	• Smart dashboard template • Conversational Exploration • Can be connected to AWS, Snowflake and other repositories	• The free version has a limit of working with 2.5m records
PANDAS	• No limit on dataset loading • Wide range of packages available for visualization • Can Import and Export data in various file formats • Reshaping and Reordering can be done • Easy handling of missing data, adding/dropping columns, slicing, indexing, merging and joining of data sets possible	• Slower as compared to XSV

Additional datasets to be used

- Pricing dataset- Quantity, Product price
- Promotional Dataset-Discounts per customer, Coupons value
- Demographic Data- User gender, age, address
- Branding Data- There is no specific data in regards with brands. Everything with respect to milk is classified as almond milk regardless of the brand.
- There's is no explicit time or date data

Key Points

- Proactive actions can be taken by the business persons in order to improve the frequency of customer orders
- Top sold products on particular days can help to analyse inventory issues from the retailers
- Insights generated can be useful for offering promotions, analysing pricing, bundle products for discounts, and by providing recommendations for a particular user based upon his searches
- By knowing the rate of products reordered, Instacart can make use of the reordered data to analyze the inventory stocks by ensuring the replenishments and proper scheduling of the products to increase internal productivity