INSTACART GROCERY BASKET ANALYSIS

Instacart's 30 million+ customer base was analyzed utilizing **Python** in **Jupyter Notebook** with libraries - **Pandas**, **Numpy**, **Scipy**, **Matplotlib.pyplot** and **Seaborn** - to assess their purchasing behaviors.

This case study details the analytical process undertaken and the skills developed throughout. For immediate access to the findings and scripts, please click the link below:

REPORT

GITHUB

BACKGROUND

Instacart wants a deeper understanding of their customers and their purchasing behaviors. They are interested in creating customer profiles to target their marketing campaigns and measure how these strategies affect proceeding sales.

NEED

Perform an initial exploratory analysis about their customers and their purchasing behaviors. Then create customer profiles and suggest marketing strategies for each.

QUESTIONS ADDRESSED:

What days and times are the busiest / are customers spending the most?

What price range groups can the products be organized in?

What types of products are more popular than others?

Which departments have the highest frequency of orders?

What are the different types of customers and their ordering behaviors?

DATASETS USED:

This analysis utilized part real ("Instacart Online Grocery Shopping Datasets 2017", Kaggle) and part fabricated datasets created for educational purposes.

The following four datasets were analyzed (key variables in bold):

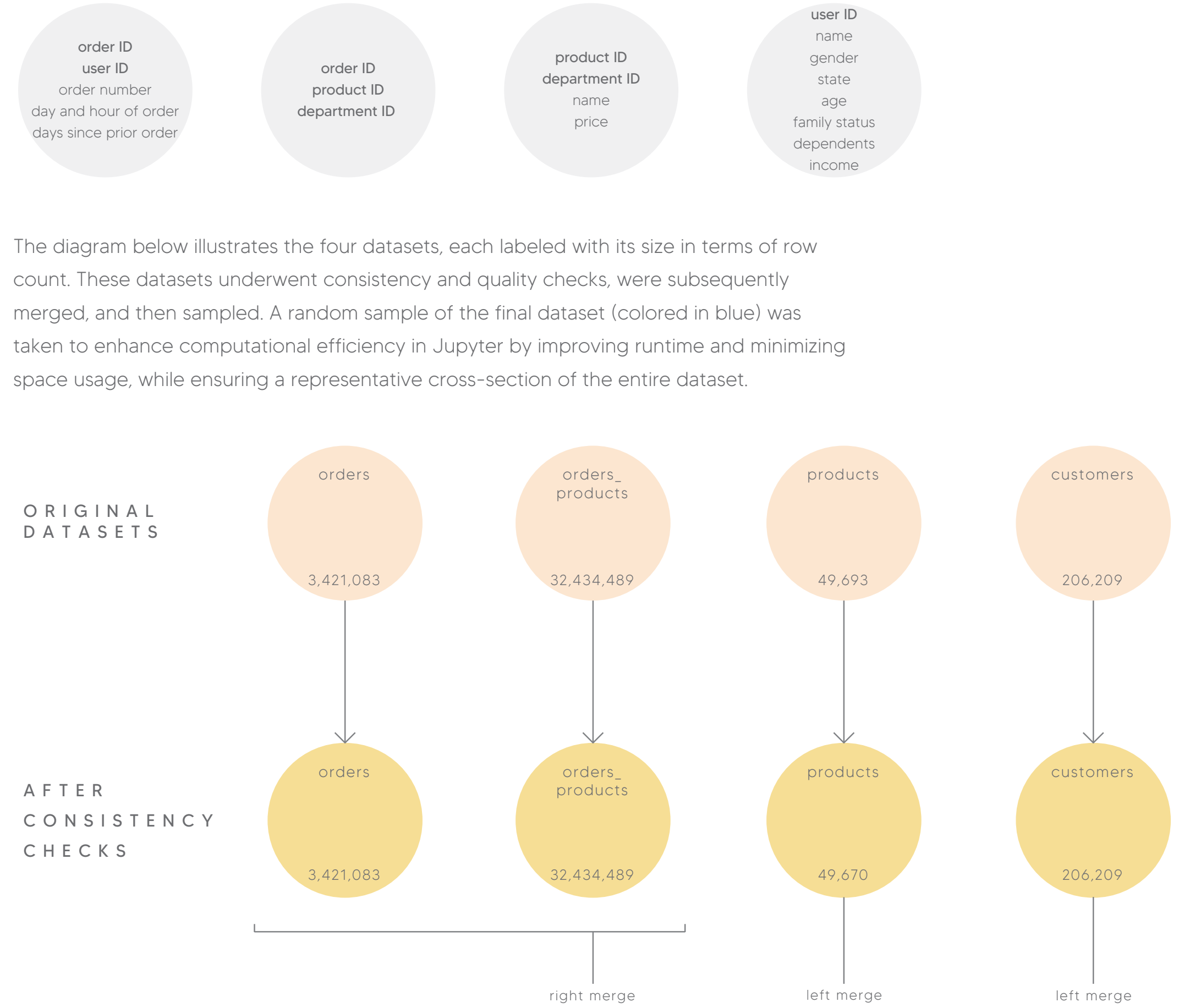
orders
- order ID
- user ID
- order number
- day and hour of order
- days since prior order

orders_products
- order ID
- product ID
- department ID

products
- product ID
- department ID
- name
- price

customers
- user ID
- name
- gender
- state
- age
- family status
- dependents
- income

The diagram below illustrates the four datasets, each labeled with its size in terms of row count. These datasets underwent consistency and quality checks, were subsequently merged, and then sampled. A random sample of the final dataset (colored in blue) was taken to enhance computational efficiency in Jupyter by improving runtime and minimizing space usage, while ensuring a representative cross-section of the entire dataset.


ORIGINAL DATASETS
orders 3,421,083
orders_products 32,434,489
products 49,693
customers 206,209
AFTER CONSISTENCY CHECKS
orders 3,421,083
orders_products 32,434,489
products 49,670
customers 206,209
right merge
left merge
left merge
AFTER MERGING
orders_products_merged 32,434,489
orders_products_merged 32,434,489
orders_products_customers_merged 32,434,489
AFTER RANDOM DATA SAMPLING
orders_products_customers_merged_smallsample 9,058,349

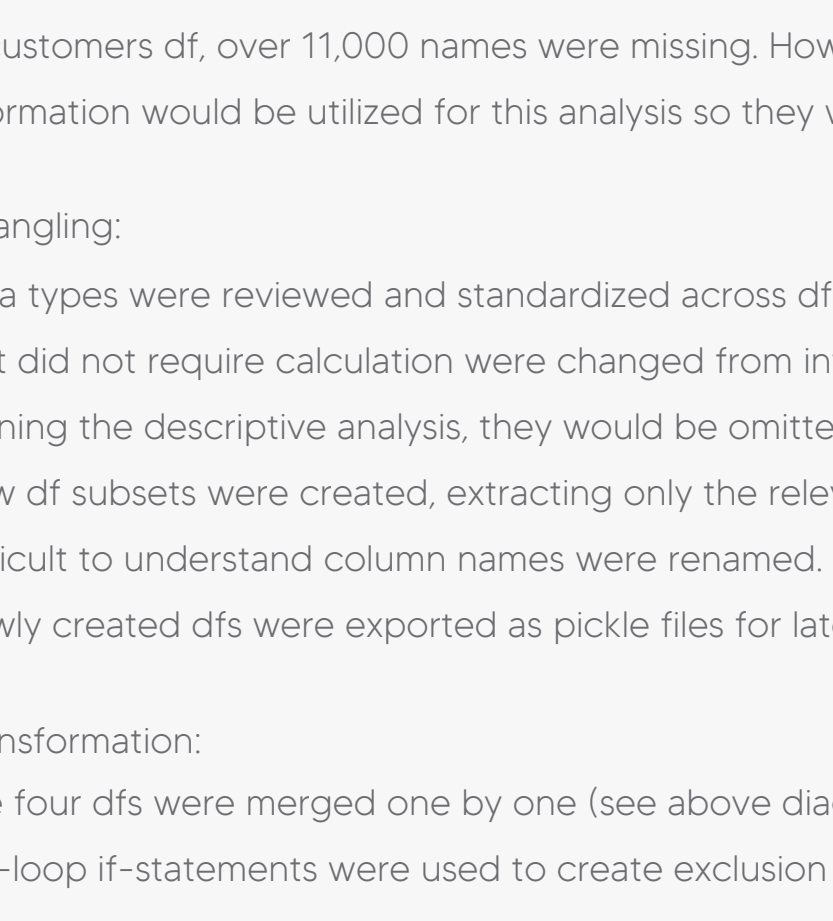
DATA PREPARATION:

Data cleaning:
- Each dataframe (df) was checked for missing and duplicate values. Intentionally missing values were left as-is. Missing product names were removed as there were no information in those rows that would contribute to the analysis.
- Duplicates in products df were checked and removed. Initially, only the entire row duplicates were removed. As products and orders dfs were merged, there were two other individual product_id duplicates causing an outcome of 570 duplicates post-merge. These product_id duplicates were removed from products df, then merged again, resulting in a clean merge.
- In products df, there were two anomaly prices: 99,999 and 14,999. These were either placeholders, outliers, errors or encoding. These were removed pre-merging.
- In customers df, over 11,000 names were missing. However, no Personally Identifiable Information would be utilized for this analysis so they were left as-is.

Data wrangling:
- Data types were reviewed and standardized across dfs. Index numbers or numbers that did not require calculation were changed from integer to string/object so when running the descriptive analysis, they would be omitted from calculation.
- New df subsets were created, extracting only the relevant columns.
- Difficult to understand column names were renamed.
- Newly created dfs were exported as pickle files for later use.

Data transformation:
- The four dfs were merged one by one (see above diagram; green).
- For-loop if-statements were used to create exclusion of certain customers, as well as creating conditional columns, from which customer profiles were built (click examples below to see full scripts).
- A random sample of the df was taken to reduce the size to 30%.

Excluding low-activity customers:



4. Exclude low spending customers from dataset

steps:
- make if-statement and new flag column for low spending customers ('max_order' <=5 orders)
- create new subset df excluding these customers
- export df as pickle

```
# for-loop if-statement: assign 'drop' to customers with <=5 'max_order'

result = []

for max_order in opc_merge['max_order']:
  if max_order <=5:
    result.append('exclude')
  else:
    result.append('keep')
```

```
# input result in new column named 'low_activity_customers'

opc_merge['low_activity_customers'] = result
```

```
# checking if new column was added

opc_merge.head(1)
```

```
me  ...  last_name  gender    state  age  date_joined  num_of_dependants  fam_status  income  _merge  low_activity_customers
da  ...     Nguyen  Female  Alabama   31   2019-02-17                  3     married   40423    both                    keep
```

```
# counting keep vs exclude

opc_merge['low_activity_customers'].value_counts()
```

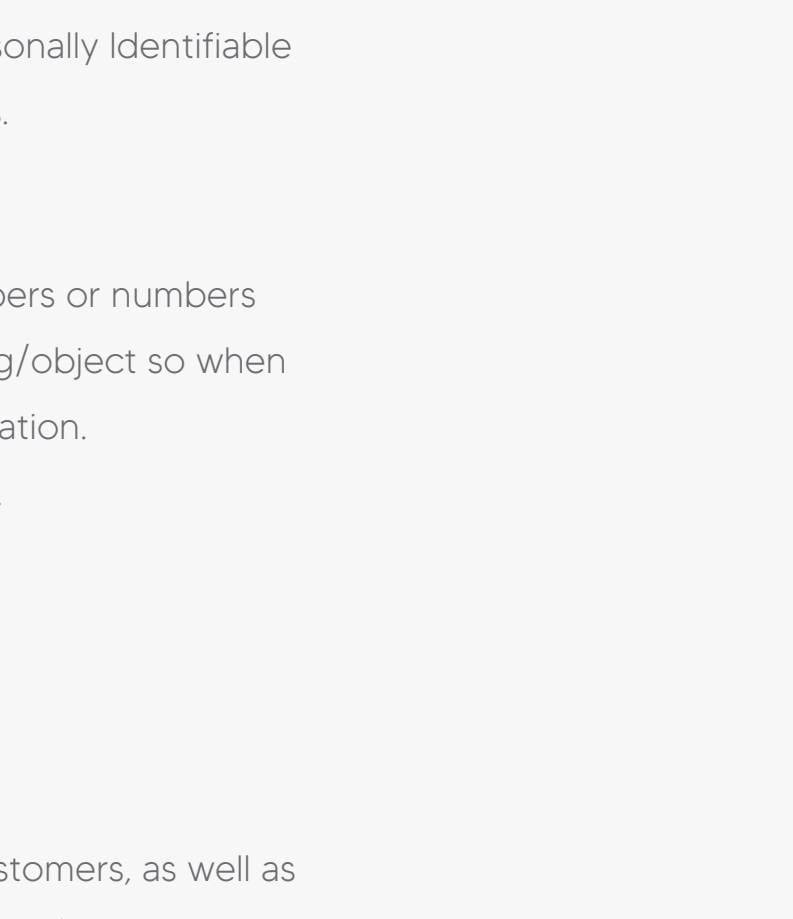
```
keep       30199150
exclude     2235339
Name: low_activity_customers, dtype: int64
```

```
# creating a subset df for 'keep' customers

keep = opc_merge.loc[opc_merge['low_activity_customers']=='keep']
```

```
# checking that row count of new df matches 'keep' count of above value_count()

keep.shape
```

```
(30199150, 36)
```

Random data sample:



b. Create a small df sample

Creating a sample of df opc_merge because keep getting "not enough memory" error messages after running the for-loop statements

```
# assigning random values to create a 70/30 sample

np.random.seed(4)
dev = np.random.rand(len(opc_merge_keep)) <= 0.7
```

```
# splitting into two sample groups, small and big

big = opc_merge_keep[dev]
small = opc_merge_keep[~dev]
```

```
len(dev)
```

```
30199150
```

```
len(small)
```

```
9058349
```

```
len(big)
```

```
21140801
```

```
len(small)+len(big)
```

```
30199150
```

```
len(opc_merge_keep)
```

```
30199150
```

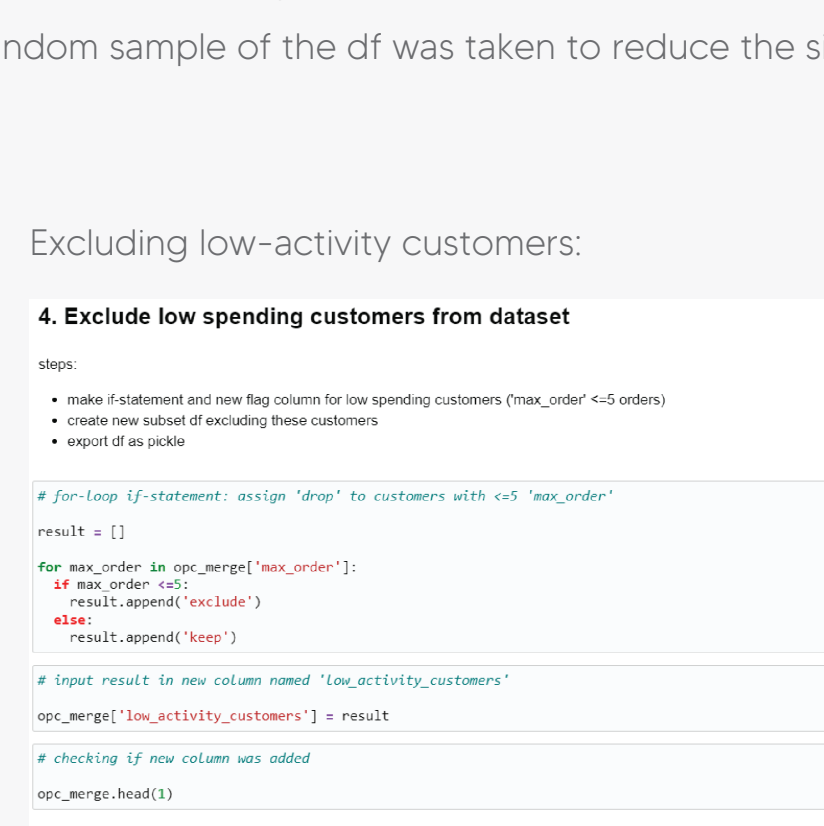
```
# make new df

opc_merge_keep_sample = small
```

```
# checking all columns are still there

opc_merge_keep_sample.head(1)
```

```
   order_id  user_id  order_number  order_day_of_week  order_hour_of_day  days_since_prior_order  product_id  add_to_cart_orc
0   2539329        1             1                  2                  8                     NaN         196

1 rows × 36 columns
```

```
# checking if number of rows match above count

opc_merge_keep_sample.shape
```

Customer Profile: Family Structure



c. Profile 2: family structure

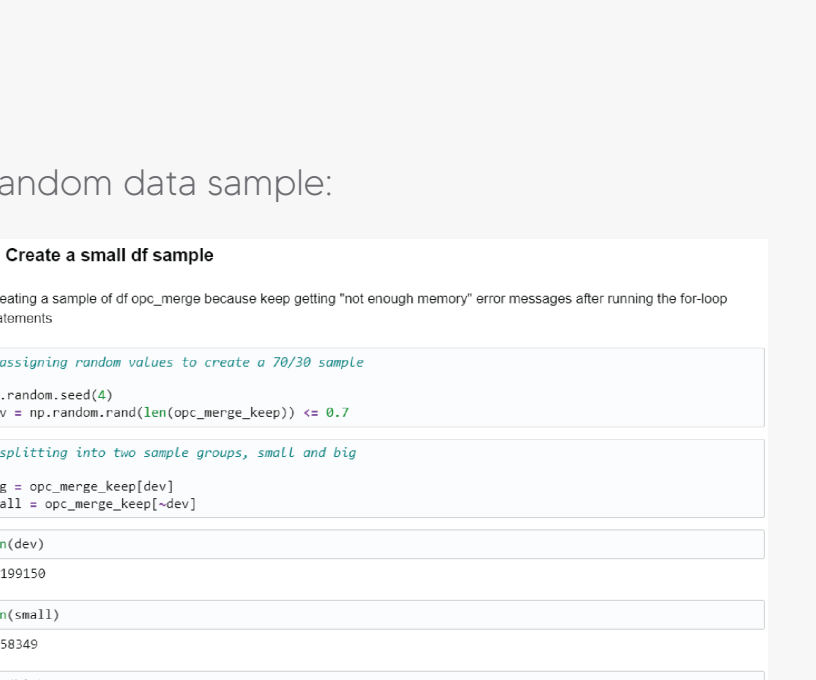
```
# group by combinations of 'fam_status', 'num_of_dependants', 'age_group' and count occurrences

opc_mkss.groupby(['age_group', 'fam_status', 'num_of_dependants']).size()
```

```
age_group  fam_status                     num_of_dependants
18-24      living with parents and siblings  1                  112314
                                             2                  109568
                                             3                  106397
           married                           1                  110305
                                             2                  102880
                                             3                  111517
           single                            0                  206838
25-34      living with parents and siblings  1                   36560
                                             2                   32559
                                             3                   36910
           married                           1                  341187
                                             2                  357300
                                             3                  351544
           single                            0                  397732
35-44      married                           1                  360999
                                             2                  352204
                                             3                  350849
           single                            0                  351146
45-54      married                           1                  360721
                                             2                  351188
                                             3                  365094
           single                            0                  350921
55-64      divorced/widowed                  0                  182318
           married                           1                  343877
                                             2                  341786
                                             3                  352059
           single                            0                  184786
65+        divorced/widowed                  0                  591751
           married                           1                  592705
                                             2                  612243
                                             3                  600091
dtype: int64
```

Profiles to create:
- cohab with dependents (18-34)
- single no dependents (18-65+)
- married with dependents (18-65+)
- divorced/widowed no dependents (55-65+)

```
# for-loop if-statement: assign criterias

result = []

for index, row in opc_mkss.iterrows():
    fam_status = row['fam_status']
```

Customer Profile: Age x Income



a. create a profile for age x income

Based on scatterplot of age and income, there are clear groupings of age and income:
- age: 18-39 (younger)
  - low income: 0-99,999
  - mid income: 100,000-199,999
  - high income: 200,000-400,000
- age: 40-85 (older)
  - low income: 0-179,999
  - mid income: 180,000-299,999
  - high income: 300,000-600,000

```
# for-loop if-statement: assign criterias

result = []

for index, row in opc_mkss.iterrows():
    income = row['income']
    age = row['age']

    if (18 <= age <= 39) and (0 <= income <= 99999):
        result.append('younger low income')
    elif (18 <= age <= 39) and (100000 <= income <= 199999):
        result.append('younger mid income')
    elif (18 <= age <= 39) and (200000 <= income <= 400000):
        result.append('younger high income')
    elif (40 <= age <= 85) and (0 <= income <= 179999):
        result.append('older low income')
    elif (40 <= age <= 85) and (180000 <= income <= 299999):
        result.append('older mid income')
    elif (40 <= age <= 85) and (300000 <= income <= 600000):
        result.append('older high income')
    else:
        result.append('NA')
```

```
# input result in new column named 'age_income'

opc_mkss['age_income_status'] = result
```

```
# count of each group

opc_mkss['age_income_status'].value_counts().sort_index()
```

```
older high income        23125
older low income       5866911
older mid income         46377
younger high income      11589
younger low income     3020502
younger mid income       89845
Name: age_income_status, dtype: int64
```

DATA EXPLORATION:

After the creation of the conditional columns, descriptive analysis and visualizations were generated to gain further insights.

Below are the conditional columns and customer profiles created, as well as an example of a descriptive analysis (click image to see entire script):

CONDITIONAL COLUMNS

product price range
- low
- mid
- high

customer loyalty
- new
- regular
- loyal

customer spenditure
- high spender
- low spender

product departments

reorder frequency
- non-frequent
- reg-frequent
- frequent

low-activity customers (excluded)

age group
- younger
- mid
- older

regions
- Northwest
- Midwest
- South
- West

income group
- lower
- mid
- higher

CUSTOMER PROFILES

family structure

Criteria:
1. cohabitating with dependents (18-34)
2. single, no dependents (18-65+)
3. married, with dependents (18-65+)
4. divorced/widowed, no dependents (55-65+)

* These criteria were created based on inherent groupings that emerged after organizing the demographic data.

age group x income

Criteria:

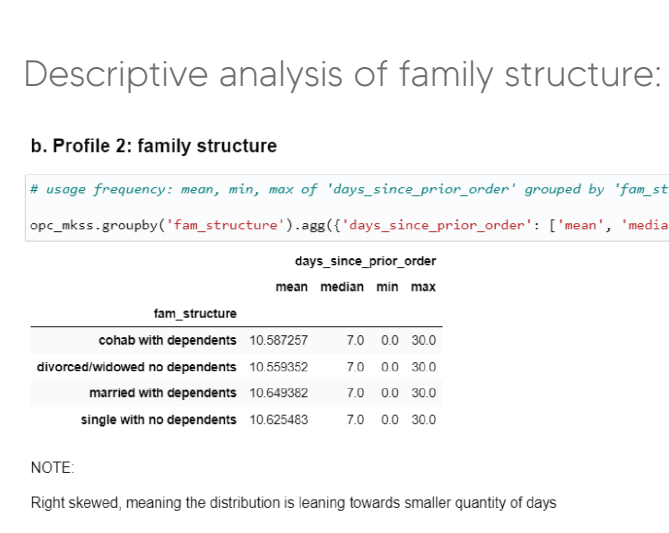
age: 18-39 (younger)
1. lower income: $0-99,999
2. mid income: $100,000-199,999
3. higher income: $200,000-400,000

age: 40-85 (older)

4. lower income: $0-179,999
5. mid income: $180,000-299,999
6. higher income: $300,000-600,000

* These criteria were created based on inherent groupings that emerged after income was distributed on a scatterplot.

Descriptive analysis of family structure:



b. Profile 2: family structure

```
# usage frequency: mean, min, max of 'days_since_prior_order' grouped by 'fam_structure'

opc_mkss.groupby('fam_structure').agg({'days_since_prior_order': ['mean', 'median', 'min', 'max']})
```

```
                               days_since_prior_order
                                   mean  median  min   max
fam_structure
cohab with dependents         10.587257     7.0  0.0  30.0
divorced/widowed no dependents 10.559352    7.0  0.0  30.0
married with dependents       10.649382     7.0  0.0  30.0
single with no dependents     10.625483     7.0  0.0  30.0
```

NOTE:

Right skewed, meaning the distribution is leaning towards smaller quantity of days

```
# expenditure: mean, min, max of 'prices' grouped by 'fam_structure'

opc_mkss.groupby('fam_structure').agg({'prices': ['mean', 'median', 'min', 'max']})
```

```
                                 prices
                                   mean  median  min   max
fam_structure
cohab with dependents          7.806203     7.4  1.0  25.0
divorced/widowed no dependents 7.804206     7.4  1.0  25.0
married with dependents        7.790513     7.4  1.0  25.0
single with no dependents      7.780913     7.4  1.0  25.0
```

```
# order time: mean, min, max of 'order_hour_of_day' grouped by 'fam_structure'

opc_mkss.groupby('fam_structure').agg({'order_hour_of_day': ['mean', 'median', 'min', 'max']})
```

```
                               order_hour_of_day
                                   mean  median  min  max
fam_structure
cohab with dependents         13.390384    13.0    0   23
divorced/widowed no dependents 13.421341   13.0    0   23
married with dependents       13.400801    13.0    0   23
single with no dependents     13.406975    13.0    0   23
```

```
# order day: mean, min, max of 'order_day_of_week' grouped by 'fam_structure'
```

CHALLENGES:

Dataset fabrication:

The customer dataset fabricated for educational purposes included attributes such as age, family structure, income and location that lacked real-world diversity:
- Distribution of the number of dependents across all ages (18-81) were nearly uniform, obscuring realistic family structures.
- Orders were nearly uniformly assigned to each year of age, resulting in a distribution where larger age groups like 65+ (spanning 17 years) accumulated more sales compared to smaller groups like 18-24 (spanning only 7 years).
- This method obscured realistic insights into how different age groups might be engaining with Instacart. In reflection, a scatterplot of order volumes and age would have shown age as not a differenciating factor.

Python visualization:

Some challenges were faced when creating stacked bar charts with Python caused partly due to my limitations with the tool and partly Python's constraints in the ability to customize colors and sort by values and variables.
- A workaround could have been to import the dataset into Tableau, or import .png of the visuals into Adobe Illustrator for customization. While the visuals created with Python conveyed the insights adequately, there was room for improved aesthetics for readability.

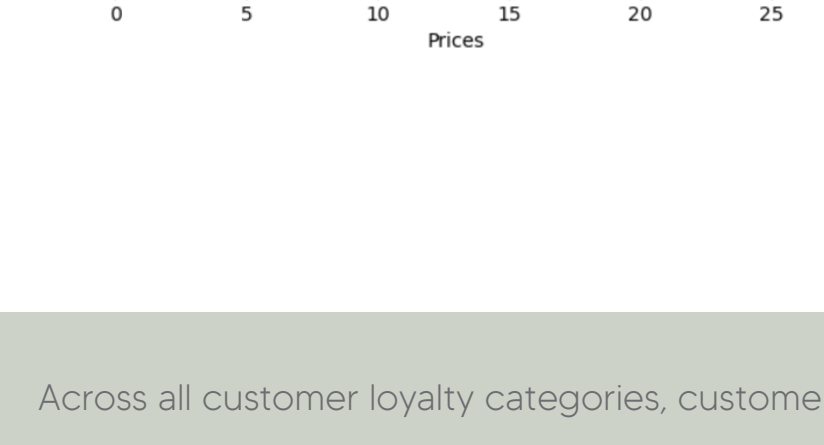
RAM issues:

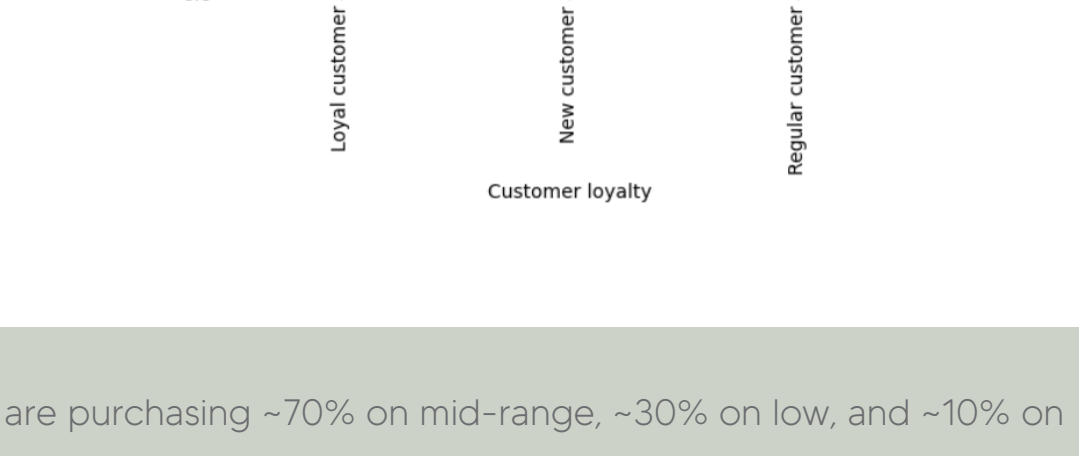
There were moments where RAM usage was reaching maximum capacity, resulting in long run-times, particularly when creating some more complex visuals.
- To reduce the size of the df, a random data sample was created.
- In hindsight, the df size could have been reduced by removing irrelevant columns or those used solely for conditional columns that subsequently became redundant. By shedding irrelevent data, it may have been possible to use the entire df instead of a sample.

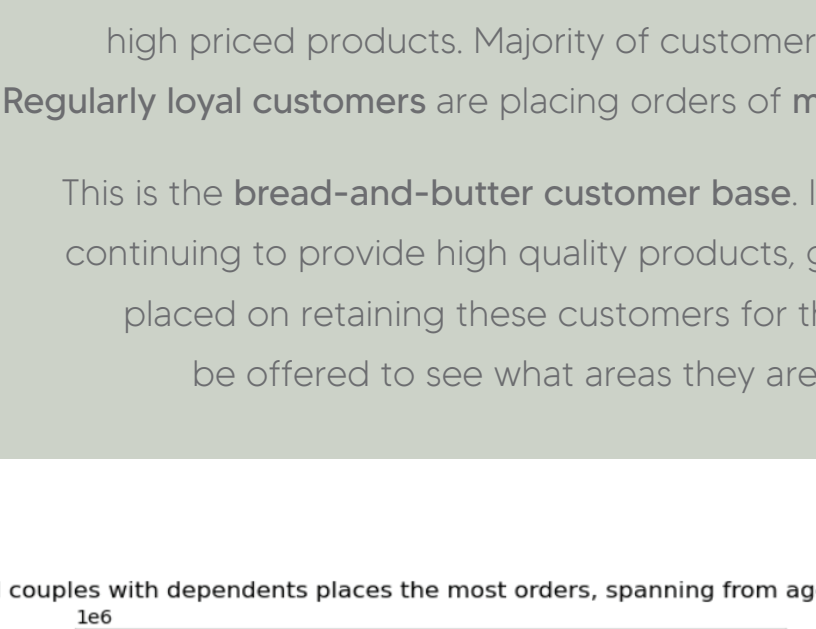
Notebook organization:

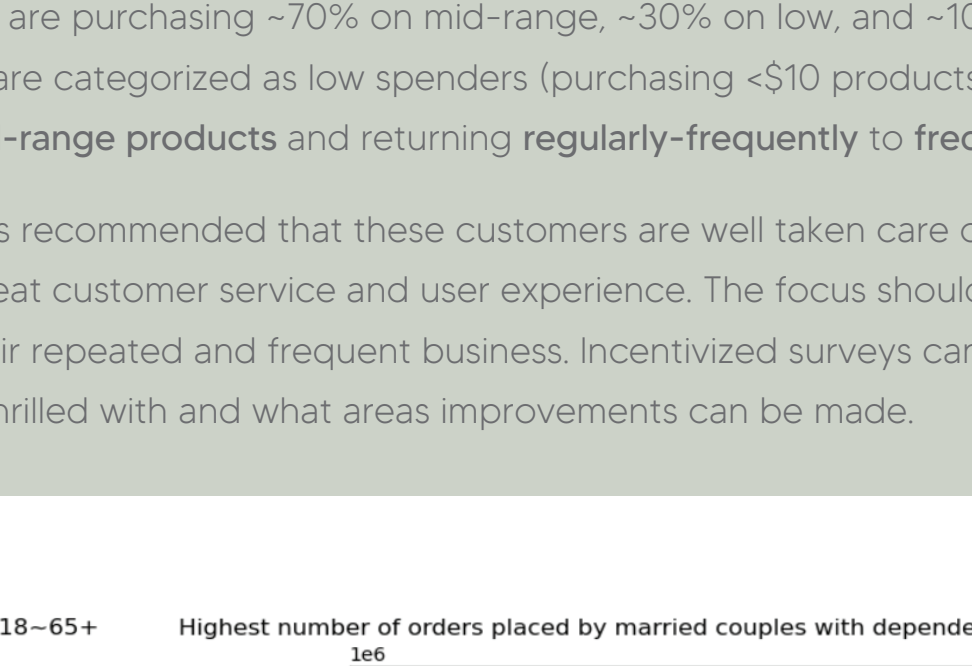
In a single notebook, the data aggregation, integration, and the full analysis was incorporated.
- The notebook was well-organized with markdown titles, subsections, and comments, but it ultimately became lengthy and cumbersome, making it difficult to locate and modify specific scripts.
- This would have benefited from having two notebooks: one for data preparation and another for analysis, where preparation was completed in the initial notebook, then imported into the second for analysis and visual creation, with any adjustments to the df made in the first notebook, and the revised data re-imported into and ran in the second notebook to complete the analysis.

REPORT HIGHLIGHTS:



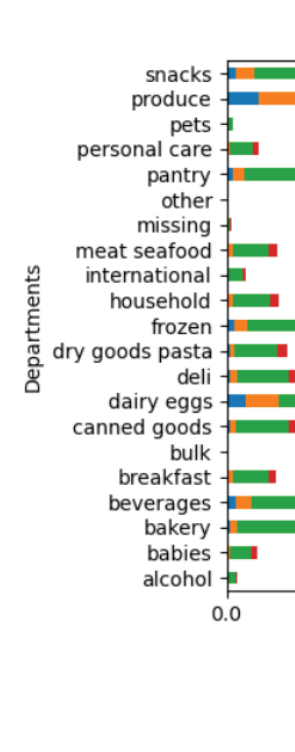


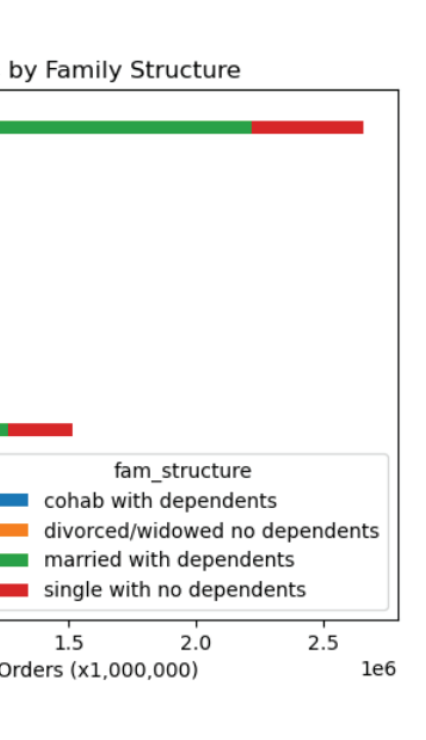


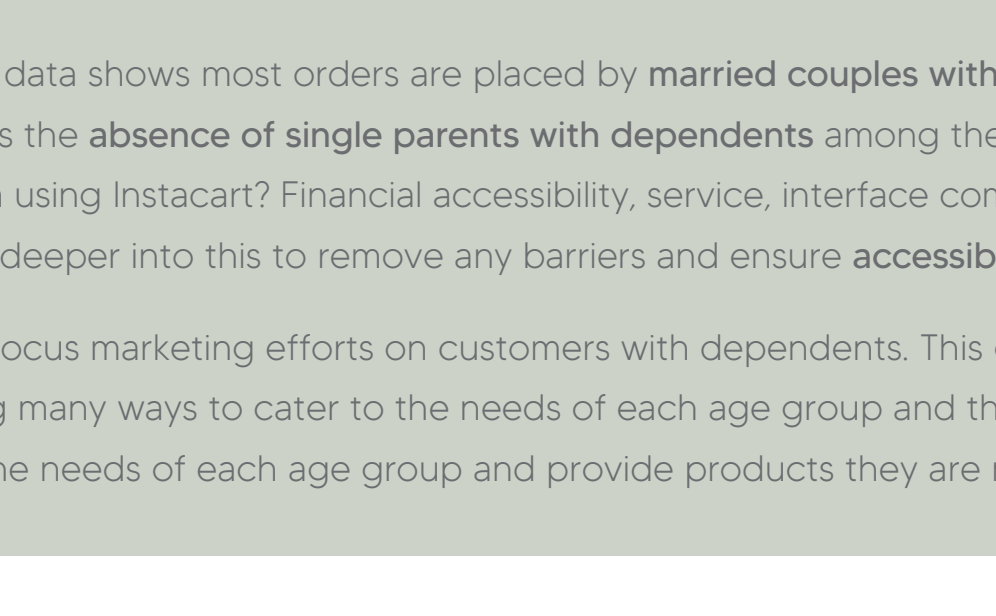


Across all customer loyalty categories, customers are purchasing ~70% on mid-range, ~30% on low, and ~10% on high priced products. Majority of customers are categorized as low spenders (purchasing <$10 products). **Regularly loyal customers** are placing orders of **mid-range products** and returning **regularly-frequently** to **frequently**.

This is the **bread-and-butter customer base**. It is recommended that these customers are well taken care of by continuing to provide high quality products, great customer service and user experience. The focus should be placed on retaining these customers for their repeated and frequent business. Incentivized surveys can be offered to see what areas they are thrilled with and what areas improvements can be made.

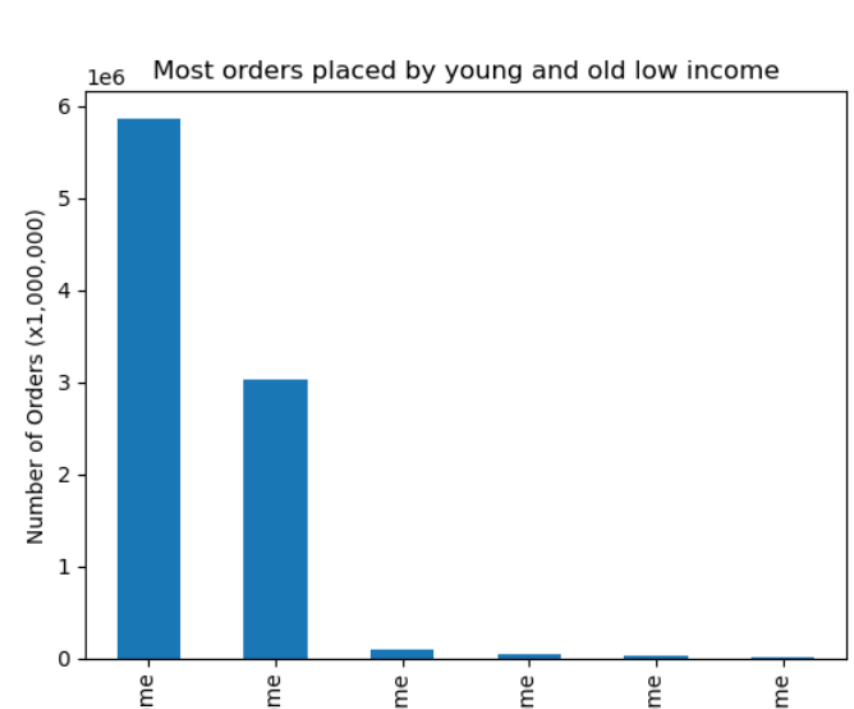


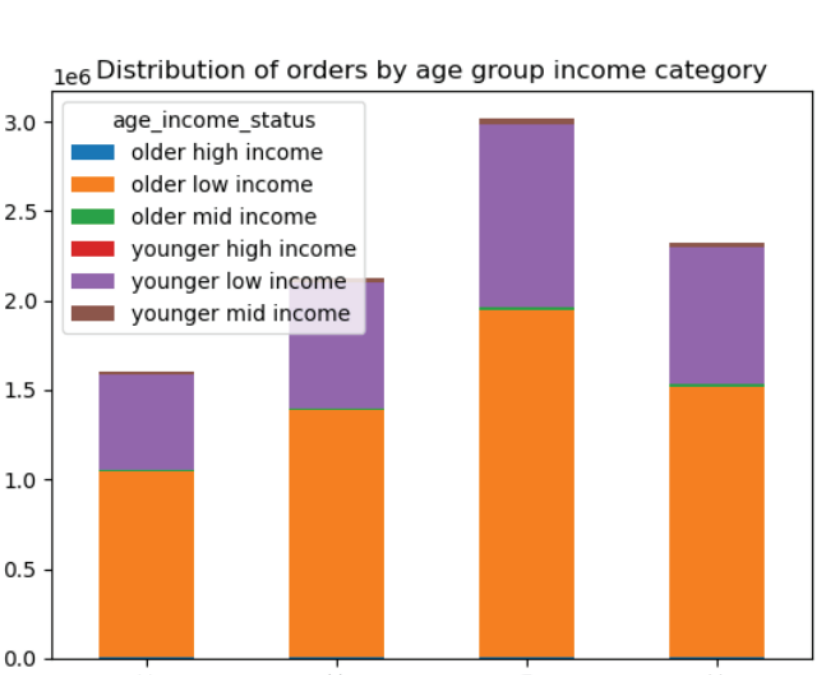




As expected, data shows most orders are placed by **married couples with dependents**.
What is striking is the **absence of single parents with dependents** among the customer base.
What is keeping them from using Instacart? Financial accessibility, service, interface complications, marketing bias?
It is recommended to look deeper into this to remove any barriers and ensure **accessibility to a broader audience**.

It is also recommended to focus marketing efforts on customers with dependents. This group is wide-ranging from ages 18 to 65+, providing many ways to cater to the needs of each age group and their growing dependents. It is key to understand the needs of each age group and provide products they are needing and wishing for.



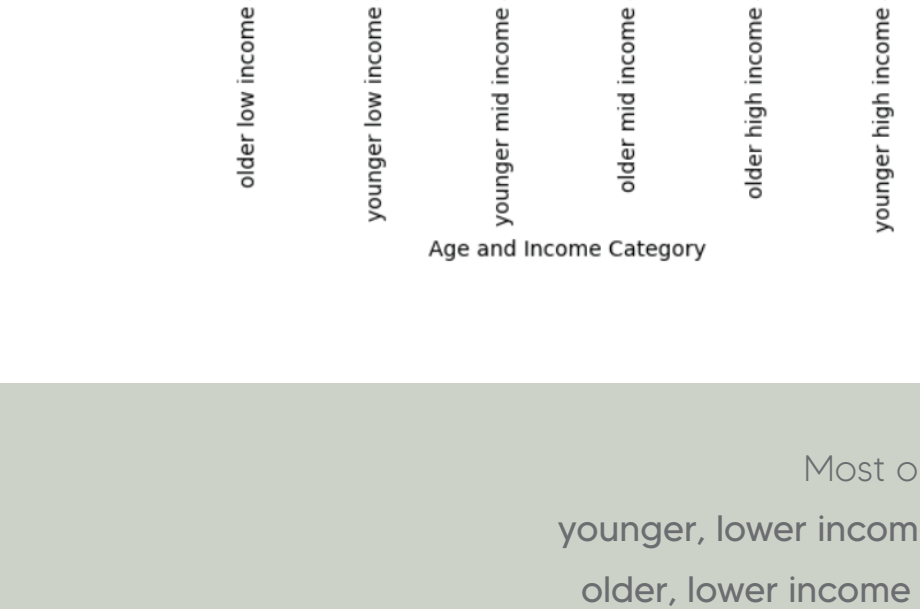


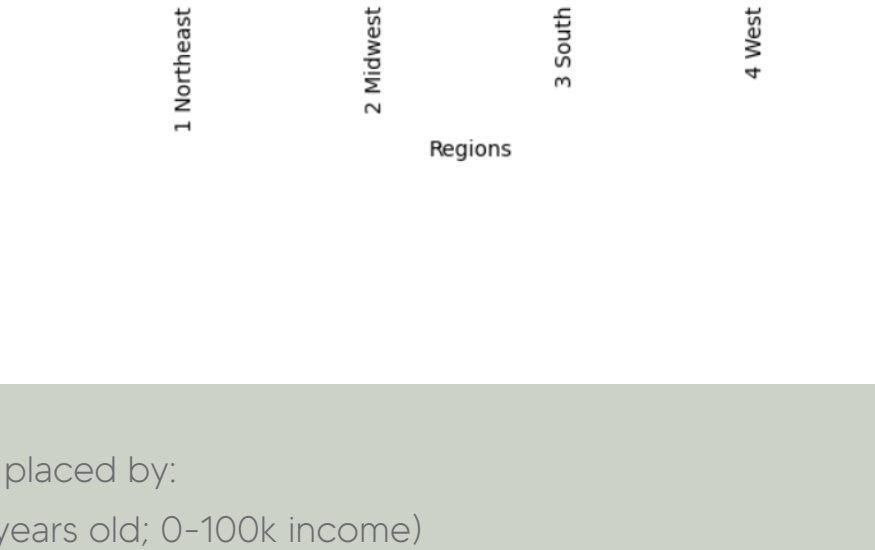
The scatterplot on the left, where age and income were plotted against each other, we can see clear groupings of income:

Age 18-40 ("young"):
- Group 1 ("low"): income between 0-100k
- Group 2 ("mid"): income between 100-200k
- Group 3 ("high"): income between 200-400k

Age 41-81 ("old"):
- Group 1 ("low"): income between 0-200k
- Group 2 ("mid"): income between 200-300k
- Group 3 ("high"): income between 300-600k

These breakdowns were used to create income category used below.





Most orders are placed by:
younger, lower income (18-41 years old; 0-100k income)
older, lower income (41-81 years old; 0-200k income)

CONCLUSION:

The objective of this project was to conduct an exploratory analysis of Instacart's customer base and their ordering behaviors, then create customer profiles to formulate targeted marketing strategies for each.

One notable aspect of this project involved using a fabricated customers dataset, which left a lot to be desired. Through this analysis, I was reminded of the importance of accurate and diverse set of variables when attempting to accurately characterize people. Comprehensive data ensures that behavioral assessments are fair and reflective of true diversity in customer behaviors.

A valuable lesson from this project was the importance of not only interpreting the data presented but also what it omits, and to compare findings against assumptions. Seeing both what exists and what doesn't highlights potential gaps in the data, improving informed decision-making.

SKILLS ACQUIRED:

Python Libraries:
- Pandas for data manipulation
- NumPy for numerical operations
- Matplotlib.pyplot and Seaborn for data visualization
- SciPy for scientific computing

Jupyter Notebook functions:
- importing datasets
- exporting dataframes as .csv and .pkl
- comments, markdowns, notebook grammar, best practices
- removing missing and duplicate values
- renaming column names
- dropping columns and rows
- changing data types
- subsetting dataframes
- aggregations (mean, median, min, max)
- merging dataframes
- if-statements; loc() function, for-loop, user-defined
- randomized sampling
- data visualizations (Matplotlib.pyplot and Seaborn): bar, stacked bar, line, histogram, scatterplot

Akiko Oguchi | Data Analyst | Information Designer | oguchiakiko@gmail.com