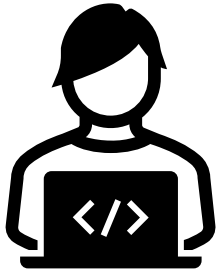
DATA ANALYTICS CASE STUDY

INSTACHART GROCERY BASKET ANALYSIS

Teguh Eka Prahara

TABLE OF CONTENTS

01. PROJECT OVERVIEW



02. DATA PREPARATION

03. DATA EXPLORATION & VISUALIZATION

04. RECOMMENDATIONS

01. PROJECT OVERVIEW

PROJECT NAME

- Instachart Grocery Basket Analysis

PROJECT OBJECTIVE

- To conduct an initial data and exploratory analysis of sales patterns of Instacart, an online grocery store, to derive insights and suggest better segmentation strategies based on provided criteria.

KEY QUESTIONS

- What are the busiest days of the week and hours of the day?
- Are there differences in ordering habits based on a customer's loyalty status?
- Is there a connection between age and family status?

TOOLS

- Jupyter Notebook
- Python (NumPy, Pandas, Matplotlib, Seaborn, Scipy)
- Ms. Excel

SKILLS

- Data Wrangling & Subsetting
- Data Consistency Checks
- Combining Data
- Deriving New Variables
- Grouping Data & Aggregating Variables
- Data Visualization with Python

DATA SET

The Instacart Online Grocery Shopping Dataset 2017", Accessed from https://www.kaggle.com/c/instacart-market-basket-analysis/data on February 23rd, 2023.

02. DATA PREPARATION

DATA WRANGLING

```
#Dropping Columns
df_ords_2 = df_ords.drop(columns = ['eval_set'])
```

```
#Changing a Variable's Data Type
df_ords['order_id'] = df_ords['order_id'].astype('str')
```

```
#Renaming Columns
df_ords.rename(columns = {'order_dow' : 'orders_day_of_week'}, inplace = True)
```

Through data wrangling and consistency checks, I have cleaned, reviewed, and ensured the accuracy and consistency of all the raw data, including orders, products, order_products_prior, and customer datasets.

CONSISTENCY CHECKS

```
In [9]:
#check for mixed-type data in df_ords
for col in df_ords.columns.tolist():
  weird = (df_ords[[col]].applymap(type) != df_ords[[col]].iloc[0].apply(type)).any(axis = 1)
  if len (df_ords[weird]) > 0:
    print (col)
```

There is no mix-type data in orders data set

```
In [17]:
df_dups = df_ords[df_ords.duplicated()]
```

```
In [18]:
#Direction 7 & 8: Check & address the duplicate of orders data set
df_dups
```

Out[18]:

order_id	user_id	order_number	orders_day_of_week	order_hour_of_day	days_since_prior_order

There is no duplicate in this data set

I opted to use the median to fill in the missing values for the "day_since_prior_order" column.

Wrangling steps

Columns dropped	Columns renamed	Columns' type changed	Comment/Reason
eval_set'			This column is irrelevant to data analysis
	order_dow' to 'orders_day_of_week'		Make it clearer to stakeholders
		The 'order_id' data type is changed to 'str'	order_id' is an index that is used to identify a specific order
		The 'user_id' data type is changed to 'str'	user_id' is an index that is used to identify a specific user
		The 'product_id' data type is changed to 'str'	product_id' is an index that is used to identify a specific product
		The 'aisle_id' data type is changed to 'str'	aisle_id' is an index that is used to identify a specific aisle
		The 'department_id' data type is changed to 'str'	aisle_id' is an index that is used to identify a specific department
	First_name' to 'first_name'		Maintain consistency of column naming
	Surnam' to 'surname'		Maintain consistency of column naming
	STATE' to 'state'		Maintain consistency of column naming
	Age' to 'age'		Maintain consistency of column naming
	Gender' to 'gender'		Maintain consistency of column naming
firt_name'			Included in PII Data
surname'			Included in PII Data

Consistency checks

Dataset	Missing values	Missing values treatment	Duplicates
orders	206209 missing values in day_since_prior_order column	Impute the value with the median of the column	No duplicates
products	16 missing values in product_name column	Remove the missing data	5 duplicates
orders_products_prior	No missing values	No missing values treatment	No duplicates
customers	11259 missing values in first_name column	Remove the missing data	No duplicates

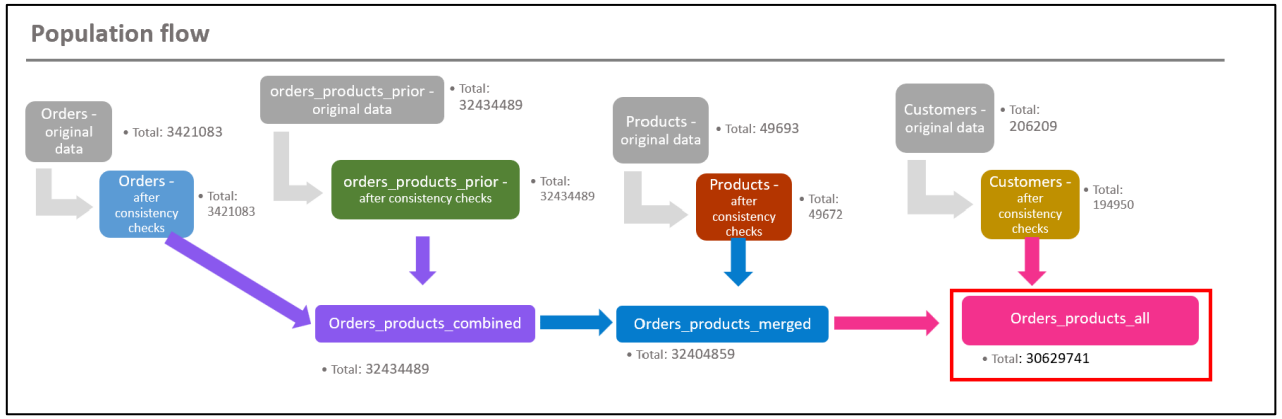
```
In [10]:
#Check for missing values in orders data set
df_ords.isnull().sum()
```

```
Out[10]: order_id                0
         user_id                 0
         order_number            0
         orders_day_of_week      0
         order_hour_of_day       0
         days_since_prior_order  206209
         dtype: int64
```

```
In [14]:
#Direction 6: address the missing values
#median = 2nd Quartile = 7
df_ords['days_since_prior_order'].fillna(7, inplace = True)
```

02. DATA PREPARATION

COMBINING DATA




```
In [15]: # Merge df_ords and df_ords_prior using order_id as a key
         df_merge_large = df_ords.merge(df_ords_prior, on = 'order_id')

In [16]: df_merge_large.head()
Out[16]:
```

	order_id	user_id	order_number	orders_day_of_week	order_hour_
0	2539329	1	1	2	
1	2539329	1	1	2	
2	2539329	1	1	2	
3	2539329	1	1	2	
4	2539329	1	1	2	

```
In [22]: df_merge_large.shape
Out[22]: (32434489, 9)
```

Combining orders and order_products_prior data sets to become the order_products_combined data set

```
In [11]: # Direction 5: Merge df_prods and df_ords_prods_combined using product_id as a key, add a merge flag
         df_combined = df_prods.merge(df_ords_prods_combined, on = 'product_id', indicator = True)

In [12]: df_combined.head()
Out[12]:
```

	product_id	product_name	aisle_id	department_id	prices	order_id	user_id	order_number	orders_day_of_
0	1	Chocolate Sandwich Cookies	61	19	5.8	3139998	138	28	
1	1	Chocolate Sandwich Cookies	61	19	5.8	1977647	138	30	
2	1	Chocolate Sandwich Cookies	61	19	5.8	389851	709	2	
3	1	Chocolate Sandwich Cookies	61	19	5.8	652770	764	1	
4	1	Chocolate Sandwich Cookies	61	19	5.8	1813452	764	3	

```
In [12]: #direction 6: confirm the results of the merge
         df_combined['_merge'].value_counts()
Out[12]: both          32404859
         left_only           0
         right_only          0
         Name: _merge, dtype: int64
```

Combining order_products_combined and products data sets to become the order_products_merged data set

Combining order_products_merged and customers data sets to become the order_products_all data set

```
In [29]: #Merge df_customers_clean and df_ords_prods_merged using user_id as a key
         df_combined = df_customers_clean.merge(df_ords_prods_merged, on = 'user_id')

In [30]: df_combined.head()
Out[30]:
```

	user_id	first_name	surname	gender	state	age	date_joined	n_dependants	fam_status	income	...	busiest_days	busiest_period_of_day	max_ord
0	26711	Deborah	Esquivel	Female	Missouri	48	1/1/2017	3	married	165665	...	Busiest days	Average Orders	
1	26711	Deborah	Esquivel	Female	Missouri	48	1/1/2017	3	married	165665	...	regularly busy	Most Orders	
2	26711	Deborah	Esquivel	Female	Missouri	48	1/1/2017	3	married	165665	...	Busiest days	Most Orders	
3	26711	Deborah	Esquivel	Female	Missouri	48	1/1/2017	3	married	165665	...	regularly busy	Average Orders	
4	26711	Deborah	Esquivel	Female	Missouri	48	1/1/2017	3	married	165665	...	Slowest days	Most Orders	

5 rows × 35 columns

```
In [31]: df_combined.shape
Out[31]: (30629741, 35)
```

03. DATA EXPLORATION & VISUALIZATION

GROUPING DATA & DERIVING VARIABLES

I have grouped and aggregated data to simplify complex data and identify patterns. For example, I have grouped and aggregated data to establish a loyalty flag, categorizing customers into Regular, Loyal, and New. Based on this flag, it is evident that the Regular customer category has the highest number of customers.

Creating a Loyalty Flag for Existing Customers using the Transform() and Loc()

```
In [27]: #agggregate data with transform()
         ords_prods_merge['max_order'] = ords_prods_merge.groupby(['user_id'])['order_number'].transform(np.max)

In [28]: #add loc() functions
         ords_prods_merge.loc[ords_prods_merge['max_order'] > 40, 'loyalty_flag'] = 'Loyal customer'

In [29]: ords_prods_merge.loc[(ords_prods_merge['max_order'] <= 40) & (ords_prods_merge['max_order'] > 10), 'loyalty_flag'] = 'Regular

In [30]: ords_prods_merge.loc[ords_prods_merge['max_order'] <= 10, 'loyalty_flag'] = 'New customer'

In [31]: #count the value of 'loyalty_flag'
         ords_prods_merge['loyalty_flag'].value_counts(dropna = False)

Out[31]: Regular customer    15876776
         Loyal customer      10284093
         New customer         6243990
         Name: loyalty_flag, dtype: int64
```

Checking the basic statistics of the product prices for each loyalty category

```
In [33]: #check the basic statistics of the product prices for each loyalty category
         ords_prods_merge.groupby('loyalty_flag').agg({'prices':['count', 'mean', 'min', 'max']})
```

Out[33]:

	prices			
	count	mean	min	max
loyalty_flag				
Loyal customer	10284093	10.386336	1.0	99999.0
New customer	6243990	13.294670	1.0	99999.0
Regular customer	15876776	12.495717	1.0	99999.0

I used basic statistics such as count, mean, min, and max in this analysis. All categories have the same minimum and maximum prices. However, the loyal customer category has the lowest mean prices. It implies loyal customers might buy products at lower prices than other categories but remains faithful to the brand.

03. DATA EXPLORATION & VISUALIZATION

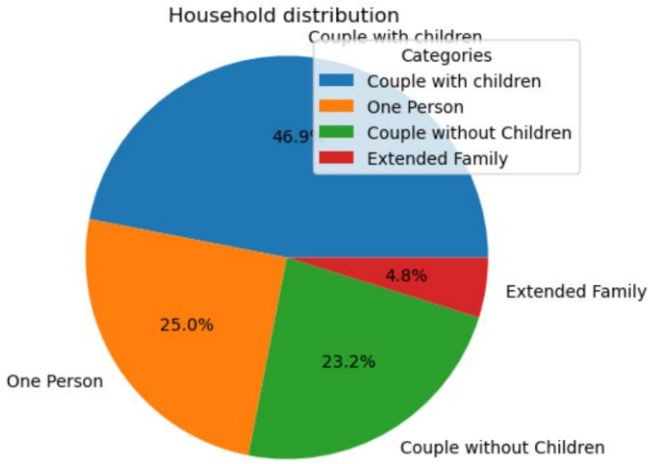
DATA VISUALIZATION

Creating Pie Chart from 'Household' Variable

In [79]:
```
# count values of each category
household_counts = df_customers['household'].value_counts(dropna=False)
```

In [150]:
```
# create pie chart
plt.pie(household_counts, labels=household_counts.index, autopct='%1.1f%%')
plt.axis('equal')
plt.title('Household distribution')
plt.legend(title='Categories')

# save the pie chart as an image
plt.savefig(os.path.join(path, '04. Analysis', 'Visualizations', 'pie_household_distribution.png'))
```



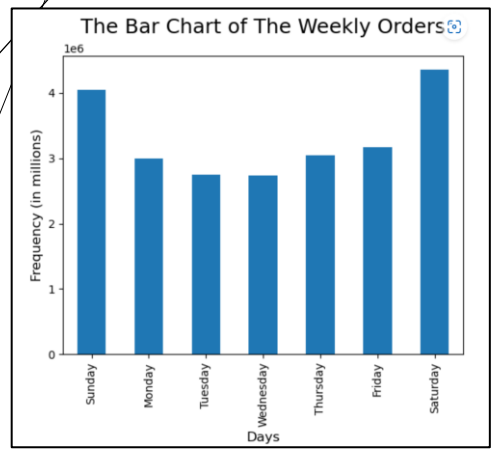
I have used Python to create data visualizations that simplify project results, making them easier for others to understand and interpret. One example is the Household variable, which I visualized in a Pie Chart using the following steps:

- Counting the values of each category
- Creating the Pie Chart using Pyplot (Matplotlib)
- Saving the Pie Chart as an image.

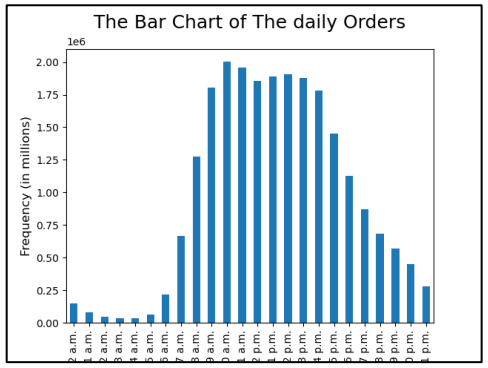
03. DATA EXPLORATION & VISUALIZATION

ANSWERING BUSINESS QUESTIONS

What are the busiest days of the week and hours of the day?

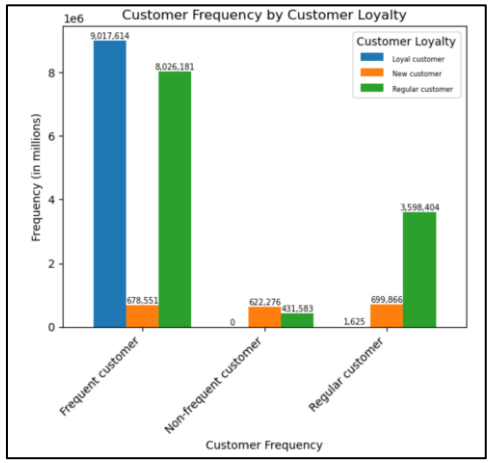


The busiest days of the week are Saturday and Sunday, whereas the slowest days are Tuesday and Wednesday.



Furthermore, most orders of the day are between 9 a.m. and 4 p.m.; the average orders are between 7 and 8 a.m. and between 5 p.m. and 11 p.m.; and the fewest are between 12 a.m. and 8 a.m.

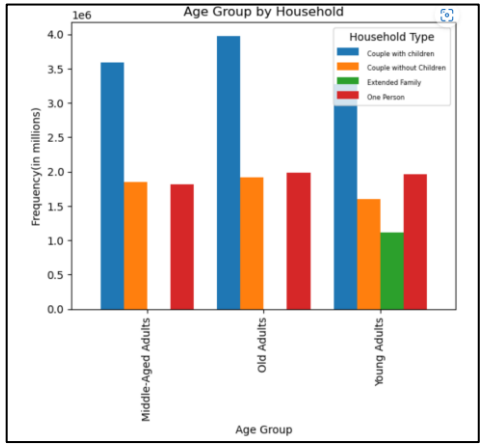
Are there differences in ordering habits based on a customer's loyalty status?



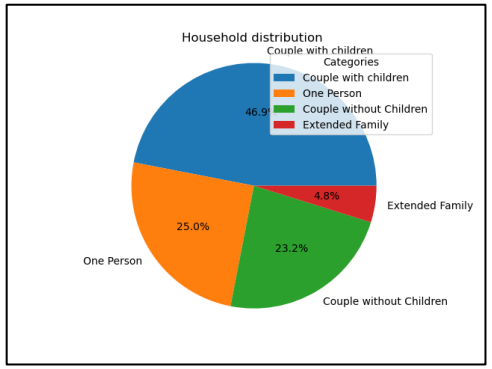
Based on the bar chart of Customer Frequency by Customer loyalty, almost all loyal customers are also frequent.

The mid-range products are most ordered for all customer groups, either for regular, loyal, or new customers.

Is there a connection between age and family status?



All of the extended family statuses came from young adults.



Old couples with children did the highest orders, followed successively by mid-aged couples with children and young couples with children.

04. RECOMENDATIONS

SCHEDULING ADS ON TUESDAY & WEDNESDAY BETWEEN 5 P.M. AND 11 P.M. IS RECOMMENDED

It can increase orders from average to higher on the slowest days. The time was chosen from 7 p.m. to 11 p.m. because that is when people rest from their daily activities.

MAXIMIZING SALES OF MID-RANGE PRODUCTS CAN INCREASE PROFITS

Based on customer loyalty, the highest number of customers is categorized as regular, followed successively by loyal and new customers. Almost all of the loyal customers are frequent customers. Furthermore, the mid-range products are most ordered for all customer groups, either for regular, loyal, or new customers.

YOUNG ADULTS CAN BE FURTHER POTENTIAL CUSTOMERS

All of the extended family statuses came from young adults. Old couples with children did the highest orders, followed successively by mid-aged couples with children and young couples with children. Hence, couples with children can be a good target market. Furthermore, almost all of the products are ordered mainly by young adults.

THANK YOU