Table of Contents

INSTACART DATA EXPLORATORY DATA ANALYIS

This project analyzes data collected by Instacart, the grocery delivery platform. The purpose of this project is to clean up the data, and use the cleaned data to report insights on shopping habits of Instacart customers. The data was cleaned by removing duplicate vales, and filling in missing values, all while maintaining the integrity of the dataset.Analyses indicated the number of orders placed, dependent on variables such as time of the day, day of the week, and time since the customer last placed an order. The results demonstrate the distribution of the number of orders customers place, the top 20 products, and the top 20 reordered products.

Loading the data

In [ ]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

Loading the data from the csv files

In [ ]:
```
# read data
df_inst_orders = pd.read_csv('datasets\orders.csv')
df_prod = pd.read_csv('datasets/products.csv')
df_aisles = pd.read_csv('datasets/aisles.csv')
df_dept = pd.read_csv('datasets/departments.csv')
df_order_prod = pd.read_csv('datasets/order_products__prior.csv')
```

In [ ]:
```
# idea of what the datasets look like
display(df_inst_orders)
print()
display(df_prod)
print()
display(df_aisles)
print()
display(df_dept)
print()
display(df_order_prod)
```

	order_id	user_id	order_number	order_dow	order_hour_of_day	days_since_prior_order
0	1515936	183418	11	6	13	30.0
1	1690866	163593	5	5	12	9.0
2	1454967	39980	4	5	19	2.0
3	1768857	82516	56	0	20	10.0
4	3007858	196724	2	4	12	17.0
...	...	...	...	...	...	...
478962	3210681	5617	5	1	14	7.0
478963	3270802	112087	2	3	13	6.0
478964	885349	82944	16	2	11	6.0
478965	216274	4391	3	3	8	8.0
478966	2071924	1730	18	1	14	15.0

478967 rows × 6 columns

	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 Wit...	38	1
4	5	Green Chile Anytime Sauce	5	13
...	...	...	...	...
49689	49690	HIGH PERFORMANCE ENERGY DRINK	64	7
49690	49691	ORIGINAL PANCAKE & WAFFLE MIX	130	14
49691	49692	ORGANIC INSTANT OATMEAL LIGHT MAPLE BROWN SUGAR	130	14
49692	49693	SPRING WATER BODY WASH	127	11
49693	49694	BURRITO- STEAK & CHEESE	38	1

49694 rows × 4 columns

	aisle_id	aisle
0	1	prepared soups salads
1	2	specialty cheeses
2	3	energy granola bars
3	4	instant foods
4	5	marinades meat preparation
...	...	...
129	130	hot cereal pancake mixes
130	131	dry pasta
131	132	beauty
132	133	muscles joints pain relief
133	134	specialty wines champagnes

134 rows × 2 columns

	department_id	department
0	1	frozen
1	2	other
2	3	bakery
3	4	produce
4	5	alcohol
5	6	international
6	7	beverages
7	8	pets
8	9	dry goods pasta
9	10	bulk
10	11	personal care
11	12	meat seafood
12	13	pantry
13	14	breakfast
14	15	canned goods
15	16	dairy eggs
16	17	household
17	18	babies
18	19	snacks
19	20	deli
20	21	missing

	order_id	product_id	add_to_cart_order	reordered
0	2141543	11440	17.0	0
1	567889	1560	1.0	1
2	2261212	26683	1.0	1
3	491251	8670	35.0	1
4	2571142	1940	5.0	1
...	...	...	...	...
4545002	577211	15290	12.0	1
4545003	1219554	21914	9.0	0
4545004	692640	47766	4.0	1
4545005	319435	691	8.0	1
4545006	1398151	28733	9.0	0

4545007 rows × 4 columns

In [ ]:
```
# info on columns
print('inst orders')
df_inst_orders.info()
print()
print('prod')
df_prod.info()
print()
print('aisles')
df_aisles.info()
print()
print('dept')
df_dept.info()
print()
print('order prod')
df_order_prod.info(show_counts=True)
```

```
inst orders
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 478967 entries, 0 to 478966
Data columns (total 6 columns):
 #   Column                  Non-Null Count   Dtype  
---  ------                  --------------   -----  
 0   order_id                478967 non-null  int64  
 1   user_id                 478967 non-null  int64  
 2   order_number            478967 non-null  int64  
 3   order_dow               478967 non-null  int64  
 4   order_hour_of_day       478967 non-null  int64  
 5   days_since_prior_order  450148 non-null  float64
dtypes: float64(1), int64(5)
memory usage: 21.9 MB

prod
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49694 entries, 0 to 49693
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   product_id     49694 non-null  int64 
 1   product_name   48436 non-null  object
 2   aisle_id       49694 non-null  int64 
 3   department_id  49694 non-null  int64 
dtypes: int64(3), object(1)
memory usage: 1.5+ MB

aisles
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 134 entries, 0 to 133
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   aisle_id  134 non-null    int64 
 1   aisle     134 non-null    object
dtypes: int64(1), object(1)
memory usage: 2.2+ KB

dept
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21 entries, 0 to 20
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
```

```
 0   department_id  21 non-null     int64
 1   department     21 non-null     object
dtypes: int64(1), object(1)
memory usage: 464.0+ bytes

order prod
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4545007 entries, 0 to 4545006
Data columns (total 4 columns):
 #   Column             Non-Null Count    Dtype
---  ------             --------------    -----
 0   order_id           4545007 non-null  int64
 1   product_id         4545007 non-null  int64
 2   add_to_cart_order  4544171 non-null  float64
 3   reordered          4545007 non-null  int64
dtypes: float64(1), int64(3)
memory usage: 138.7 MB
```

Conclusions on data

We have a total of 5 datasets from Instacart. The Instacart orders dataset is large and fairly clean, with missing values only in the days since prior order column. This data has close to half a million entries. The second dataset includes the products on the platform. with approximately 50,000 entries. This dataset is fairly clean with some missing product names. The aisles dataset is small with 134 entries, and no missing values. The department dataset contains 21 values, with no missing values. The last dataset is order products. Some values are missing from the add to cart order column. Missing values from these datasets should be assessed, to determine whether the missing data should be removed, or kept in place.

Find and remove duplicate values

`orders` data frame

In [ ]:
```
# Check for duplicated orders
df_inst_orders[df_inst_orders.duplicated(subset='order_id')]
```

Out[ ]:

	order_id	user_id	order_number	order_dow	order_hour_of_day	days_since_prior_order
145574	794638	50898	24	3	2	2.0
223105	2160484	107525	16	3	2	30.0
230807	1918001	188546	14	3	2	16.0
266232	1782114	106752	1	3	2	NaN
273805	1112182	202304	84	3	2	6.0
284038	2845099	31189	11	3	2	7.0
311713	1021560	53767	3	3	2	9.0
321100	408114	68324	4	3	2	18.0
323900	1919531	191501	32	3	2	7.0
345917	2232988	82565	1	3	2	NaN
371905	391768	57671	19	3	2	10.0
394347	467134	63189	21	3	2	2.0
411408	1286742	183220	48	3	2	4.0
415163	2282673	86751	49	3	2	2.0
441599	2125197	14050	48	3	2	3.0

In [ ]:
```
# check the number of duplicates
df_inst_orders['order_id'].duplicated().value_counts()
```

Out[ ]:
```
False    478952
True         15
Name: order_id, dtype: int64
```

We confirm that there are a total of 15 duplicated orders in our dataset that we need to address

In [ ]:
```
# checking orders with highest frequency, to 16 values to check wheter 15 orders are duplicated
df_inst_orders['order_id'].value_counts().head(16)
```

Out[ ]:
```
2125197    2
1782114    2
1286742    2
391768     2
1021560    2
2232988    2
408114     2
2282673    2
1919531    2
2160484    2
1918001    2
794638     2
1112182    2
467134     2
2845099    2
2357032    1
Name: order_id, dtype: int64
```

In [ ]:
```
# number of days in days of week column
print(df_inst_orders['order_dow'].nunique())
```

```
7
```

In [ ]:
```
# Check for all orders placed Wednesday at 2:00 AM
print(df_inst_orders.query("order_dow==4 and order_hour_of_day==2")[['order_id']].count())
```

```
order_id    114
dtype: int64
```

114 orders placed on Wednesday @ 2:00 AM

In [ ]:
```
# orders per jour of day
df_grou = df_inst_orders.groupby('order_hour_of_day')['order_dow'].count()
print(df_grou)
```

```
order_hour_of_day
0      3180
1      1763
2      1004
3       770
4       765
5      1371
6      4215
7     13043
8     25024
9     35896
10    40578
11    40032
12    38034
13    39007
14    39631
15    39789
16    38112
17    31930
18    25510
19    19547
20    14624
21    11019
22     8512
23     5611
Name: order_dow, dtype: int64
```

In [ ]:
```
# Remove duplicate orders
df_inst_orders['order_id'] = df_inst_orders['order_id'].drop_duplicates()
print(df_inst_orders['order_id'])
```

```
0         1515936.0
1         1690866.0
2         1454967.0
3         1768857.0
4         3007858.0
             ...
478962    3210681.0
478963    3270802.0
478964     885349.0
478965     216274.0
478966    2071924.0
Name: order_id, Length: 478967, dtype: float64
```

In [ ]:
```
# Double check for duplicate rows
df_inst_orders.duplicated().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
# Double check for duplicate order IDs only
df_inst_orders['order_id'].value_counts().head(16)
```

Out[ ]:
```
220638.0     1
827753.0     1
516243.0     1
2728009.0    1
391704.0     1
211077.0     1
424589.0     1
21984.0      1
1509114.0    1
467182.0     1
3201275.0    1
569125.0     1
2032571.0    1
652972.0     1
1443182.0    1
3202147.0    1
Name: order_id, dtype: int64
```

In [ ]:
```
# number of unique order id's
df_inst_orders['order_id'].nunique()
```

Out[ ]:
```
478952
```

Conclusions on orders dataframe

The orders dataframe had a few duplicates that we needed to delete, as doing so would improve the results of our findings, without negatively affecting the data. We confirmed 15 duplicated orders, and we deleted the duplicates. After, we confirmed the data no longer contained duplicates. It was noted that the days of the week column had 7 values, and the order hour of the day had 24 values. We were also able to confirm orders, based on the hour of the day, and the day of the week.

`products` data frame

In [ ]:
```
# visual of the data
display(df_prod)
```

	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 Wit...	38	1
4	5	Green Chile Anytime Sauce	5	13
...	...	...	...	...
49689	49690	HIGH PERFORMANCE ENERGY DRINK	64	7
49690	49691	ORIGINAL PANCAKE & WAFFLE MIX	130	14
49691	49692	ORGANIC INSTANT OATMEAL LIGHT MAPLE BROWN SUGAR	130	14
49692	49693	SPRING WATER BODY WASH	127	11
49693	49694	BURRITO- STEAK & CHEESE	38	1

49694 rows × 4 columns

In [ ]:
```
# Check for fully duplicate rows
df_prod.duplicated().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
# Check for just duplicate product IDs
df_prod['product_id'].duplicated().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
# Check for just duplicate product names (convert names to lowercase to compare better)
df_prod['product_name'] = df_prod['product_name'].str.lower()
df_prod['product_name'].value_counts()
```

Out[ ]:
```
green tea with ginseng and honey              3
vitamin d3 1000 iu                            2
original potato chips                         2
cream of celery condensed soup                2
balsamic vinegar of modena                    2
                                             ..
classic crust pepperoni pizza                 1
temptations cat treats tempting tuna          1
soft baked sugar cookies                      1
frosted granny's apple pie toaster pastries   1
stage 1 just bartlett pears baby food         1
Name: product_name, Length: 48332, dtype: int64
```

In [ ]:
```
# Check for duplicate product names
df_prod['product_name'].duplicated().sum()
```

Out[ ]:
```
1361
```

Products Dataframe Conclusions

At first, we could not see duplicated values in the product data. However, product names were converted to lower case words. At this point, we were able to uncover duplicated product names, and we saw 1,361. Since the product names are all lowercase, we can keep the duplicates, as deleting them will alter some of our insights.

`departments` data frame

In [ ]:
```
# visual of the data
display(df_dept)
```

	department_id	department
0	1	frozen
1	2	other
2	3	bakery
3	4	produce
4	5	alcohol
5	6	international
6	7	beverages
7	8	pets
8	9	dry goods pasta
9	10	bulk
10	11	personal care
11	12	meat seafood
12	13	pantry
13	14	breakfast
14	15	canned goods
15	16	dairy eggs
16	17	household
17	18	babies
18	19	snacks
19	20	deli
20	21	missing

In [ ]:
```
# number of duplicates
df_dept.duplicated().sum()
```

Out[ ]:
```
0
```

Departments Dataframe Conclusions

The departments data contains 21 values without missing or duplicated values.

`aisles` data frame

In [ ]:
```
# visual of the data
display(df_aisles)
```

	aisle_id	aisle
0	1	prepared soups salads
1	2	specialty cheeses
2	3	energy granola bars
3	4	instant foods
4	5	marinades meat preparation
...	...	...
129	130	hot cereal pancake mixes
130	131	dry pasta
131	132	beauty
132	133	muscles joints pain relief
133	134	specialty wines champagnes

134 rows × 2 columns

There are 134 aisles with no missing values.

In [ ]:
```
df_aisles.duplicated().sum()
```

Out[ ]:
```
0
```

Aisles Dataframe Conclusions

The aisles data contains 134 values without missing or duplicated values.

`order_products` data frame

In [ ]:
```
# visual of the data
display(df_order_prod)
```

	order_id	product_id	add_to_cart_order	reordered
0	2141543	11440	17.0	0
1	567889	1560	1.0	1
2	2261212	26683	1.0	1
3	491251	8670	35.0	1
4	2571142	1940	5.0	1
...	...	...	...	...
4545002	577211	15290	12.0	1
4545003	1219554	21914	9.0	0
4545004	692640	47766	4.0	1
4545005	319435	691	8.0	1
4545006	1398151	28733	9.0	0

4545007 rows × 4 columns

In [ ]:
```
# Check for fullly duplicate rows
df_order_prod.duplicated().sum()
```

Out[ ]:
```
0
```

In [ ]:
```
# duplicated order id
df_order_prod['order_id'].duplicated().sum()
```

Out[ ]:
```
4094961
```

In [ ]:
```
# duplicated product id
df_order_prod['product_id'].duplicated().sum()
```

Out[ ]:
```
4499434
```

Keep these duplicates because we can have order id's and product id's used multiple times among orders.

In [ ]:
```
# checking for tricky duplicates with order id and product id
df_order_prod[df_order_prod.duplicated(subset=['order_id', 'product_id'])]
```

Out[ ]:

	order_id	product_id	add_to_cart_order	reordered

Order Products Dataframe Conclusions

We kept the duplicate product ids and duplicated order ids, as it is reasonable to conclude that these could be used multiple times. We also checked for hidden duplicates in the pair of columns, order id and product id, for products duplicated within the same order. One of the aisles is labeled as missing.

Find and remove missing values

`products` data frame

In [ ]:
```
# visual of the data
display(df_prod)
```

	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 wit...	38	1
4	5	green chile anytime sauce	5	13
...	...	...	...	...
49689	49690	high performance energy drink	64	7
49690	49691	original pancake & waffle mix	130	14
49691	49692	organic instant oatmeal light maple brown sugar	130	14
49692	49693	spring water body wash	127	11
49693	49694	burrito- steak & cheese	38	1

49694 rows × 4 columns

In [ ]:
```
# Missing product names
df_prod['product_name'].isna().sum()
```

Out[ ]:
```
1258
```

Total of 1258 missing product names.

In [ ]:
```
# Are all of the missing product names associated with aisle ID 100?
print(df_prod.query("aisle_id==100")[['aisle_id', 'product_name']])
```

```
       aisle_id product_name
37          100          NaN
71          100          NaN
109         100          NaN
296         100          NaN
416         100          NaN
...         ...          ...
49552       100          NaN
49574       100          NaN
49640       100          NaN
49663       100          NaN
49668       100          NaN

[1258 rows x 2 columns]
```

In [ ]:
```
# aisle id 100 product names
print(df_prod.query("aisle_id==100")[['aisle_id', 'product_name']].count())
```

```
aisle_id        1258
product_name       0
dtype: int64
```

All of missing product names are associated with aisle ID 100.

In [ ]:
```
# Are all of the missing product names associated with department ID 21?
print(df_prod.query("department_id==21")[['department_id', 'product_name']])
```

```
       department_id product_name
37                21          NaN
71                21          NaN
109               21          NaN
296               21          NaN
416               21          NaN
...              ...          ...
49552             21          NaN
49574             21          NaN
49640             21          NaN
49663             21          NaN
49668             21          NaN

[1258 rows x 2 columns]
```

In [ ]:
```
# department id 21 product names
print(df_prod.query("department_id==21")[['department_id', 'product_name']].count())
```

```
department_id    1258
product_name        0
dtype: int64
```

All of missing product names are also associated with department ID 21.

In [ ]:
```
# What is this aisle and department?
print(df_dept.query("department_id==21"))
print()
print(df_aisles.query("aisle_id==100"))
```

```
    department_id department
20             21    missing

    aisle_id    aisle
99       100  missing
```

Department and aisle both labeled as missing.

In [ ]:
```
# Fill missing product names with 'Unknown'
df_prod['product_name'] = df_prod['product_name'].fillna('unknown')
```

In [ ]:
```
# QC check on aisle id 100 changing product name to unknown
print(df_prod.query("aisle_id==100")[['aisle_id', 'product_name']])
```

```
       aisle_id product_name
37          100      unknown
71          100      unknown
109         100      unknown
296         100      unknown
416         100      unknown
...         ...          ...
49552       100      unknown
49574       100      unknown
49640       100      unknown
49663       100      unknown
49668       100      unknown

[1258 rows x 2 columns]
```

In [ ]:
```
# filling missing department id
df_prod['department_id'] = df_prod['department_id'].fillna('unknown')
```

In [ ]:
```
# QC check on department 21 changing product name to unknown
print(df_prod.query("department_id==21")[['department_id', 'product_name']])
```

```
       department_id product_name
37                21      unknown
71                21      unknown
109               21      unknown
296               21      unknown
416               21      unknown
...              ...          ...
49552             21      unknown
49574             21      unknown
49640             21      unknown
49663             21      unknown
49668             21      unknown

[1258 rows x 2 columns]
```

Products Dataframe Conclusions

We saw that there were 1258 missing product names. These missing values were associated with aisle id 100 and department id 21. Then, we saw that the aisle and department in question were labeled as missing. We cleaned the data by changing these missing values to 'unknown'. We then confirmed proper implementation by once again looking at aisle id 100 and department id 21. Missing values were indeed changed to 'unknown'. This allowed us to keep the data, as aisle and department were not crucial in our analysis.

`orders` data frame

In [ ]:
```
# display dataframe
display(df_inst_orders)
```

	order_id	user_id	order_number	order_dow	order_hour_of_day	days_since_prior_order
0	1515936.0	183418	11	6	13	30.0
1	1690866.0	163593	5	5	12	9.0
2	1454967.0	39980	4	5	19	2.0
3	1768857.0	82516	56	0	20	10.0
4	3007858.0	196724	2	4	12	17.0
...	...	...	...	...	...	...
478962	3210681.0	5617	5	1	14	7.0
478963	3270802.0	112087	2	3	13	6.0
478964	885349.0	82944	16	2	11	6.0
478965	216274.0	4391	3	3	8	8.0
478966	2071924.0	1730	18	1	14	15.0

478967 rows × 6 columns

In [ ]:
```
# Are there any missing values where it's not a customer's first order?
df_inst_orders.query("order_number > 1").isna().sum()
```

Out[ ]:
```
order_id                13
user_id                  0
order_number             0
order_dow                0
order_hour_of_day        0
days_since_prior_order   0
dtype: int64
```

In [ ]:
```
# 13 missing values where it is not a customer's first order
print('Number of missing order id values, where its not a customer\'s first order:')
df_inst_orders.query("order_number > 1")['order_id'].isna().sum()
```

Out[ ]:
```
Number of missing order id values, where its not a customer's first order:
13
```

In [ ]:
```
# ensure proper query of data
print(df_inst_orders.query("order_number > 1").sort_values(by='order_number', ascending=True))
```

```
         order_id  user_id  order_number  order_dow  order_hour_of_day  \
426491  1769679.0   195377             2          4                 11
435359  3349347.0   115073             2          5                 16
201530  2114762.0   125572             2          0                 22
454267  2283616.0   113965             2          5                  9
201532  1289383.0   203762             2          1                 20
...           ...      ...           ...        ...                ...
142916  3406102.0    71049           100          6                 12
60091   1055636.0   142304           100          3                 10
60207   1947672.0   113588           100          0                 15
28574   2378889.0    95171           100          1                 15
2482      94667.0     8220           100          1                 13

        days_since_prior_order
426491                     8.0
435359                    23.0
201530                    30.0
454267                     2.0
201532                     5.0
...                        ...
142916                     5.0
60091                      2.0
60207                      1.0
28574                      0.0
2482                       3.0

[450148 rows x 6 columns]
```

Orders Dataframe conclusions

We see a total of 13 missing values, where it is not a customer's first order. We keep the data, because it still hase usefull information.

`order_products` data frame

In [ ]:
```
# display dataframe
display(df_order_prod)
```

	order_id	product_id	add_to_cart_order	reordered
0	2141543	11440	17.0	0
1	567889	1560	1.0	1
2	2261212	26683	1.0	1
3	491251	8670	35.0	1
4	2571142	1940	5.0	1
...	...	...	...	...
4545002	577211	15290	12.0	1
4545003	1219554	21914	9.0	0
4545004	692640	47766	4.0	1
4545005	319435	691	8.0	1
4545006	1398151	28733	9.0	0

4545007 rows × 4 columns

In [ ]:
```
# What are the min and max values in this column
print('Min, Max')
df_order_prod['product_id'].min(), df_order_prod['product_id'].max()
```

Out[ ]:
```
Min, Max
(1, 49694)
```

In [ ]:
```
# Save all order IDs with at least one missing value in 'add_to_cart_order'
df_miss = df_order_prod[['add_to_cart_order','order_id']].isna().sort_values(by='order_id', ascending=False)
```

In [ ]:
```
# number of missing values
df_order_prod['add_to_cart_order'].isna().sum()
```

Out[ ]:
```
836
```

In [ ]:
```
# create missing cart dataframe
df_miss_cart = df_order_prod[['add_to_cart_order','order_id','product_id']].sort_values(by='add_to_cart_order').tail(83
print(df_miss_cart)
```

```
         add_to_cart_order  order_id  product_id
737                    NaN   2449164        5068
9926                   NaN   1968313       43867
14394                  NaN   2926893       11688
16418                  NaN   1717990        4142
30114                  NaN   1959075       42828
...                    ...       ...         ...
4505662                NaN   1800005        7411
4511400                NaN   1633337         260
4517562                NaN    404157        9517
4534112                NaN   1673227       17835
4535739                NaN   1832957       17949

[836 rows x 3 columns]
```

In [ ]:
```
# merge two order dataframes
df_ord_prod_merge = df_inst_orders.merge(df_order_prod, on='order_id')
print(df_ord_prod_merge)
```

```
          order_id  user_id  order_number  order_dow  order_hour_of_day  \
0        1515936.0   183418            11          6                 13
1        1515936.0   183418            11          6                 13
2        1515936.0   183418            11          6                 13
3        1515936.0   183418            11          6                 13
4        1515936.0   183418            11          6                 13
...            ...      ...           ...        ...                ...
4545002  2071924.0     1730            18          1                 14
4545003  2071924.0     1730            18          1                 14
4545004  2071924.0     1730            18          1                 14
4545005  2071924.0     1730            18          1                 14
4545006  2071924.0     1730            18          1                 14

         days_since_prior_order  product_id  add_to_cart_order  reordered
0                          30.0       19048                1.0          1
1                          30.0       47766                3.0          1
2                          30.0       45066                9.0          0
3                          30.0       24082               11.0          0
4                          30.0       24363                4.0          0
...                         ...         ...                ...        ...
4545002                    15.0        1503               17.0          0
4545003                    15.0        4778               12.0          1
4545004                    15.0       11512                1.0          1
4545005                    15.0        4920                6.0          1
4545006                    15.0       17678                4.0          1

[4545007 rows x 9 columns]
```

In [ ]:
```
# Do all orders with missing values have more than 64 products?
df_miss_merge = df_miss_cart.merge(df_order_prod, on='order_id', how='left')
print(df_miss_merge)
```

```
       add_to_cart_order_x  order_id  product_id_x  product_id_y  \
0                      NaN   2449164          5068          5068
1                      NaN   2449164          5068         14386
2                      NaN   2449164          5068         32864
3                      NaN   2449164          5068          8518
4                      NaN   2449164          5068         24497
...                    ...       ...           ...           ...
74963                  NaN   1832957         17949         38844
74964                  NaN   1832957         17949         19348
74965                  NaN   1832957         17949         16793
74966                  NaN   1832957         17949          5438
74967                  NaN   1832957         17949         17949

       add_to_cart_order_y  reordered
0                      NaN          0
1                      NaN          0
2                     10.0          1
3                      NaN          0
4                     32.0          1
...                    ...        ...
74963                 24.0          1
74964                  NaN          1
74965                 19.0          1
74966                  NaN          1
74967                  NaN          1

[74968 rows x 6 columns]
```

In [ ]:
```
# all orders with missing values have a minimum of 65 products
df_miss_merge.groupby('order_id')['product_id_x'].value_counts().min()
```

Out[ ]:
```
65
```

In [ ]:
```
# Replace missing values with 999 and convert column to integer type
df_order_prod['add_to_cart_order'] = df_order_prod['add_to_cart_order'].fillna('999')
df_order_prod['add_to_cart_order'] = pd.to_numeric(df_order_prod['add_to_cart_order'], errors='coerce')
df_order_prod['add_to_cart_order'] = df_order_prod['add_to_cart_order'].astype('int', errors='ignore')
```

In [ ]:
```
# checking dtype
df_order_prod.dtypes
```

Out[ ]:
```
order_id              int64
product_id            int64
add_to_cart_order     int64
reordered             int64
dtype: object
```

Order Produts Dataframe

This data shows us we have close to 50,000 different products for purchase on the platform. We see missing values in the column that determines the order a product is placed into the cart. We also observe that all orders with missing values contain more than 64 products in the cart. Analyses need to be made with this column, so we change the missing values to '999' in the data. We can do this based on the assumption that the number of items customers usually put into their cart, is less that 64. We will confirm this later in the project. The add to cart order column was then changed to an integer type, as decimals were not needed. This was done because we only add integer values for items to the cart.

[A1] Verify that the `'order_hour_of_day'` and `'order_dow'` values in the `orders` tables are sensible (i.e. `'order_hour_of_day'` ranges from 0 to 23 and `'order_dow'` ranges from 0 to 6)

In [ ]:
```
# min and max order hours of the day
print('Min, Max')
df_inst_orders['order_hour_of_day'].min() , df_inst_orders['order_hour_of_day'].max()
```

Out[ ]:
```
Min, Max
(0, 23)
```

In [ ]:
```
# numnber of hours
sorted(df_inst_orders['order_hour_of_day'].unique())
```

Out[ ]:
```
[0,
 1,
 2,
 3,
 4,
 5,
 6,
 7,
 8,
 9,
 10,
 11,
 12,
 13,
 14,
 15,
 16,
 17,
 18,
 19,
 20,
 21,
 22,
 23]
```

In [ ]:
```
# min and max order days
print('Min, Max')
df_inst_orders['order_dow'].min(), df_inst_orders['order_dow'].max()
```

Out[ ]:
```
Min, Max
(0, 6)
```

In [ ]:
```
# list of order days
sorted(df_inst_orders['order_dow'].unique())
```

Out[ ]:
```
[0, 1, 2, 3, 4, 5, 6]
```

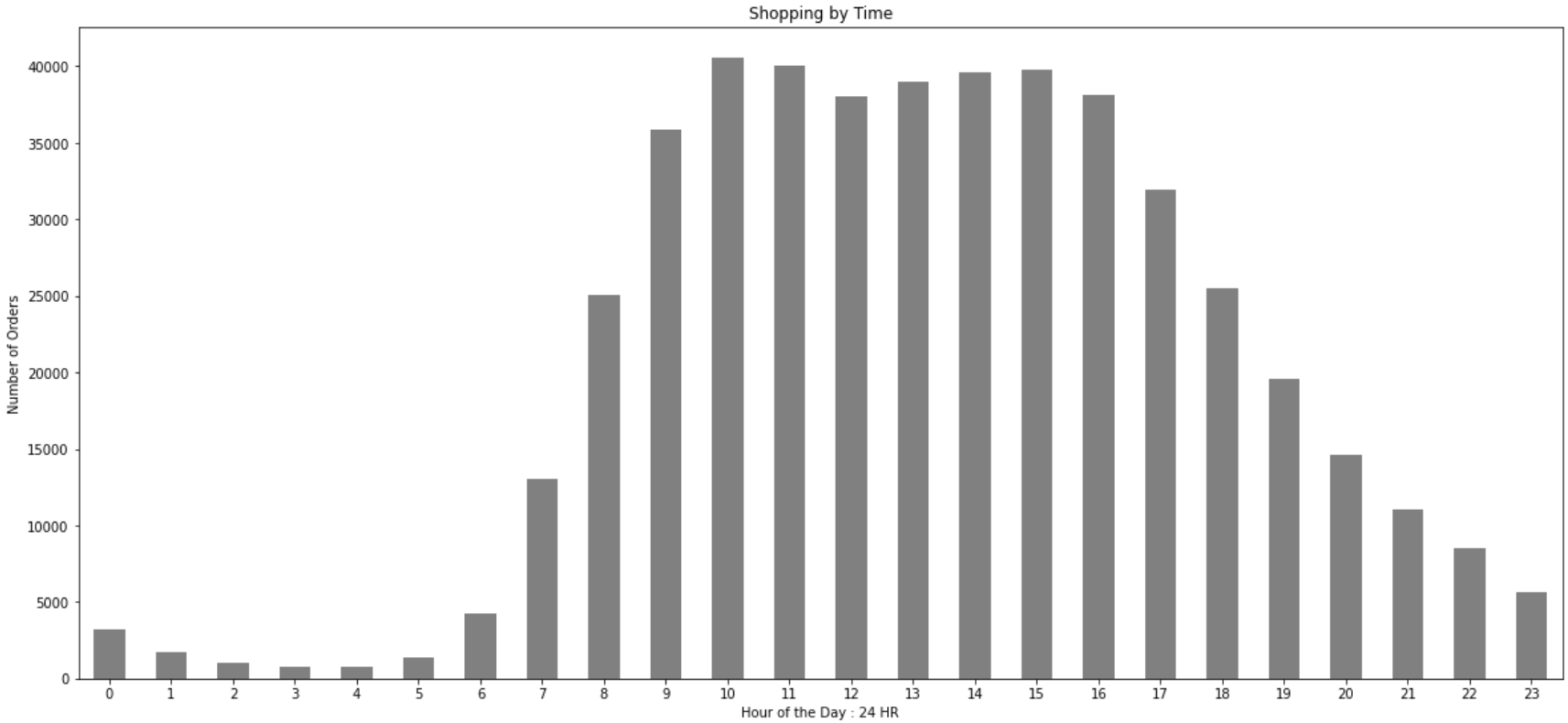
Results

First used describe to show min and max values of the columns, but using min/max for simplicity. Order hour of the day ranges from 0 to 23, which covers 24 hrs in a day. Order day of the week ranges from 0 to 6, which covers 7 days of the week. Assumption is made that the week starts on Sunday, [0]. We also confirm all seven days are accounted for, as well as every hour of the day.

[A2] What time of day do people shop for groceries?

In [ ]:
```
# create shop time dataframe
df_shop_time = df_inst_orders['order_hour_of_day'].value_counts().sort_index()
```

In [ ]:
```
# plotting shop time
df_shop_time.plot(kind='bar',
                  title='Shopping by Time',
                  x='order_hour_of_day',
                  xlabel='Hour of the Day : 24 HR',
                  rot=0,
                  y='count',
                  ylabel='Number of Orders',
                  color='grey',
                 figsize=(20,9))
plt.show()
```



Results

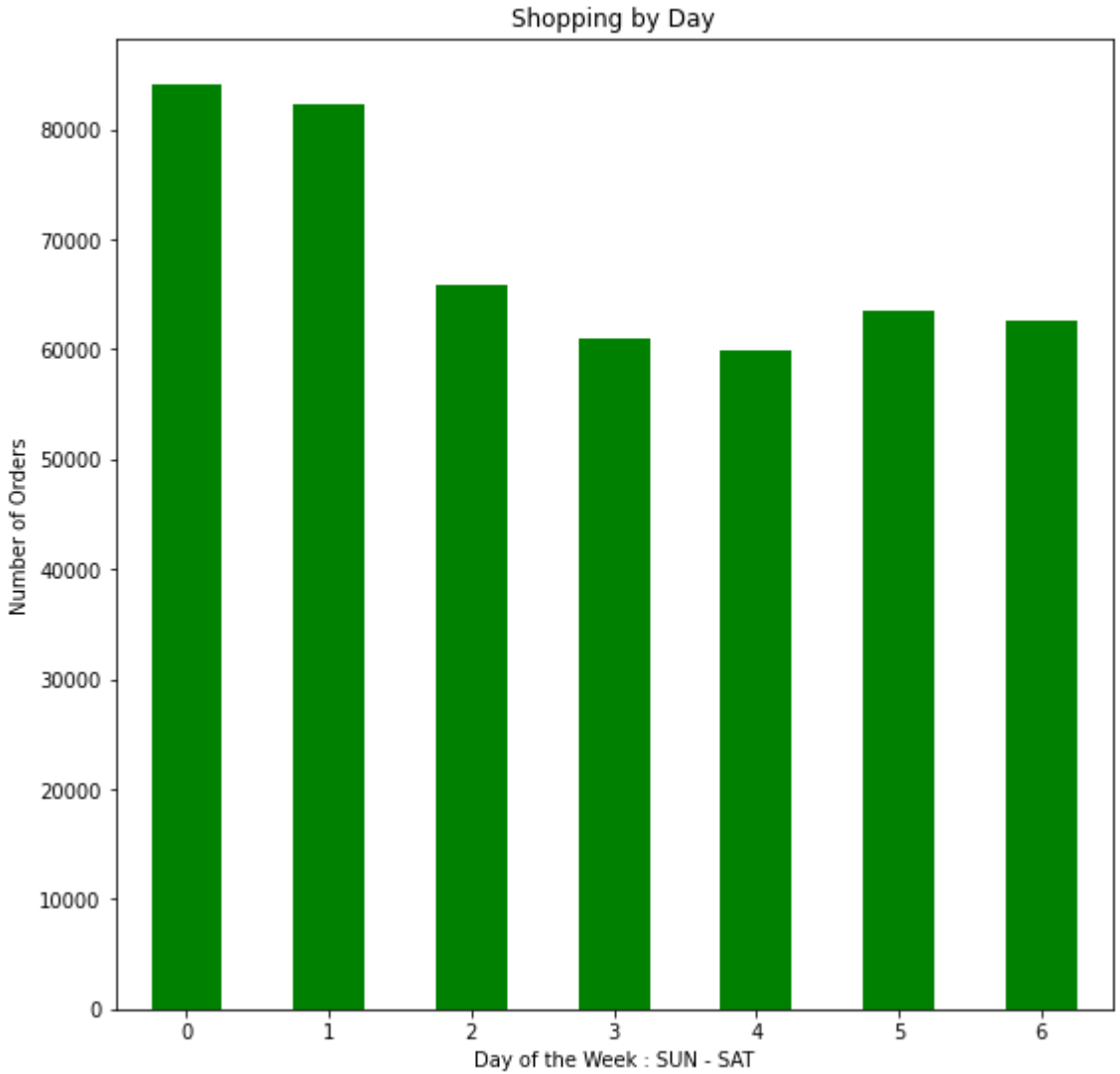
People shop for groceries throughout the day. Early morning has the lowest frequency. Most orders come in at 10 am. Late morning and early afternoons receive the most orders, so values are distributed towards the middle of the day.

[A3] What day of the week do people shop for groceries?

In [ ]:
```
# create day of week dataframe
df_ord_dow = df_inst_orders['order_dow'].value_counts()
df_ord_dow = df_ord_dow.sort_index()
print(df_ord_dow)
```

```
0    84090
1    82185
2    65833
3    60912
4    59810
5    63488
6    62649
Name: order_dow, dtype: int64
```

In [ ]:
```
# plot day of week
df_ord_dow.plot(kind='bar',
                title='Shopping by Day',
                x='order_dow',
                xlabel='Day of the Week : SUN - SAT',
                rot=0,
                y='count',
                ylabel='Number of Orders',
                color='green',
               figsize=(9,9))
plt.show()
```



Results

People shop for groceries thorughout the week, with Sunday and Monday having the highest, and second highest amount of orders, respectively. The middle of the week sees the least amount of orders.

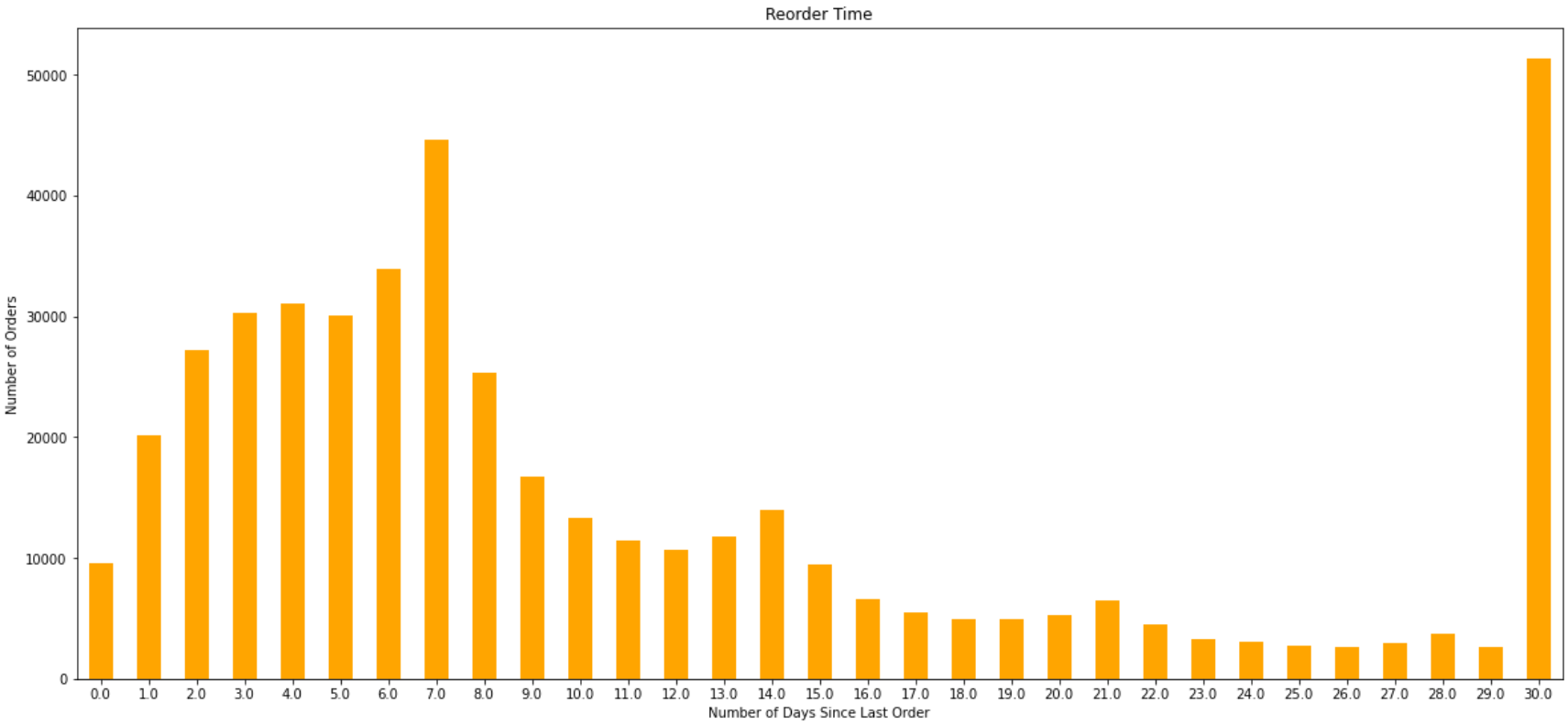
[A4] How long do people wait until placing another order?

In [ ]:
```
# create dataframe for wait for order
df_wait_ord = df_inst_orders['days_since_prior_order'].value_counts().sort_index()
print(df_wait_ord)
```

```
0.0      9589
1.0     20179
2.0     27141
3.0     30225
4.0     31007
5.0     30096
6.0     33931
7.0     44579
8.0     25361
9.0     16754
10.0    13310
11.0    11467
12.0    10658
13.0    11737
14.0    13992
15.0     9416
16.0     6588
17.0     5498
18.0     4972
19.0     4939
20.0     5302
21.0     6448
22.0     4514
23.0     3337
24.0     3015
25.0     2711
26.0     2640
27.0     2986
28.0     3745
29.0     2673
30.0    51338
Name: days_since_prior_order, dtype: int64
```

In [ ]:
```
# plot wait for order
df_wait_ord.plot(kind='bar',
                 title='Reorder Time',
                 x='days_since_prior_order',
                 xlabel='Number of Days Since Last Order',
                 rot=0,
                 y='count',
                 ylabel='Number of Orders',
```

```
                color='orange',
               figsize=(20,9))
plt.show()
```



Results

People usually wait more than a month before placing another order, followed by waiting a week to place another order. We also notice local peaks in the data every seven days. This is in line with what we should expect, with perishable items needing to be replaced quite often, while other items can be frozen or stored for longer periods of time. With most of the orders distributed towards a one week timeframe, this suggests a bulk of the orders on the platform could be perishables.

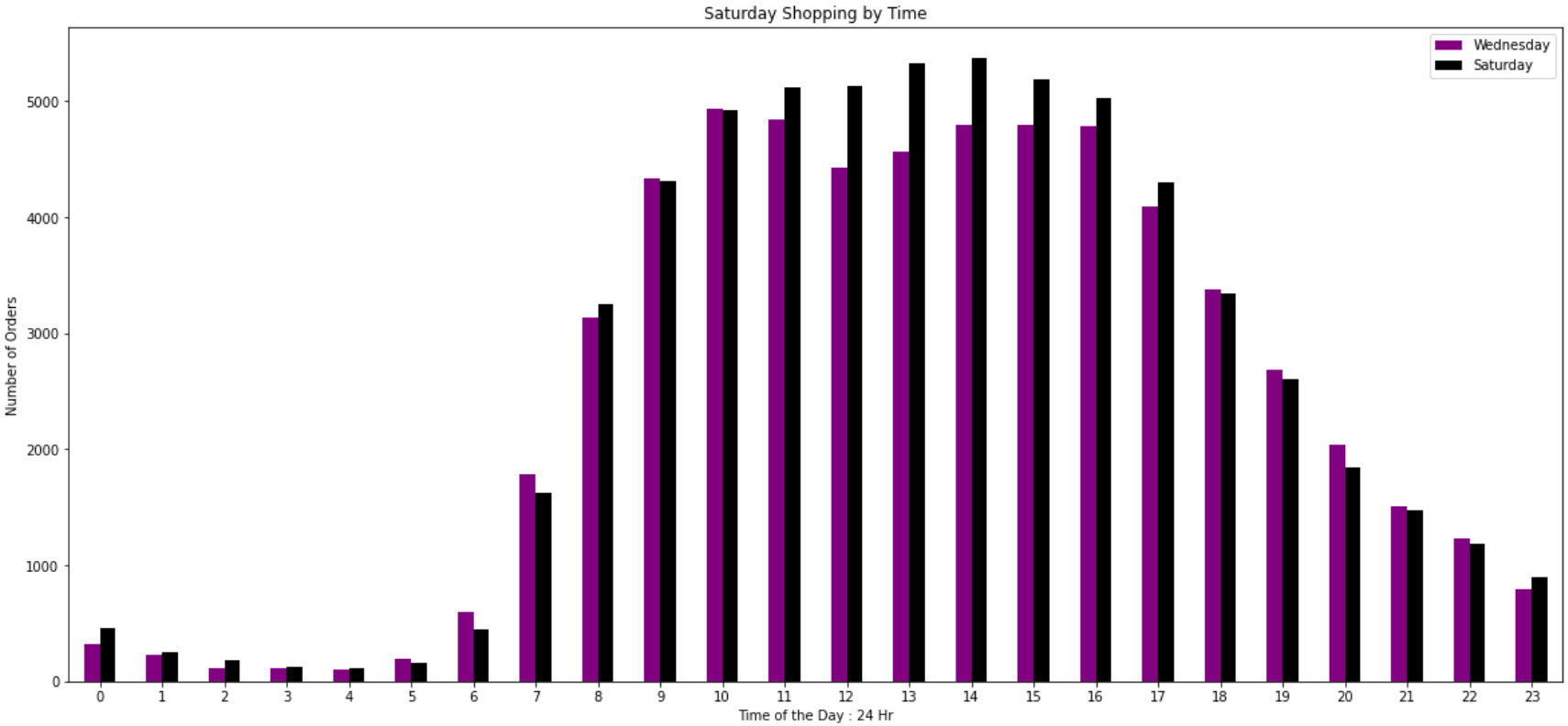
[B1] Is there a difference in `'order_hour_of_day'` distributions on Wednesdays and Saturdays? Plot the histograms for both days and describe the differences that you see.

In [ ]:
```
# order hour by day of the week
display(df_inst_orders.query('order_hour_of_day')[['order_dow']].sort_values(by='order_dow'))
```

	order_dow
23	0
5	0
23	0
7	0
23	0
...	...
14	6
11	6
11	6
20	6
14	6

478967 rows × 1 columns

In [ ]:
```
# wednesday and saturday orders
tmp = df_inst_orders.query('order_dow == 4 or order_dow == 6').pivot_table(index = 'order_hour_of_day',
                                columns = 'order_dow',
                                values = 'order_id',
                                aggfunc = 'count')
tmp.columns = ['Wednesday', 'Saturday']
tmp.plot(kind='bar',
                 title='Saturday Shopping by Time',
                 xlabel='Time of the Day : 24 Hr',
                 rot=0,
                 ylabel='Number of Orders',
                 color=('purple', 'black'),
                figsize=(20,9))
plt.show()
```



Results

Orders from both days have a similar distribution, with more orders during the middle of the day. Early morning sees the least amount of orders. Orders tail off in the evening. Initially, one would assume this data suggests most customers follow a 9-5 work shcedule. Yet, the data includes Saturday, when most people are off from work. A stronger realtionship may be seen with typical sleep schedules, as people go to sleep from the late evening, to the early morning.

[B2] What's the distribution for the number of orders per customer?

In [ ]:
```
# creating order per customer dataframe
df_ord_per_cus = df_inst_orders.query('user_id')['order_number']
print(df_ord_per_cus)
```

```
183418    38
163593    11
39980      3
82516     11
196724     5
          ..
5617      24
112087     1
82944     21
4391      58
1730      27
Name: order_number, Length: 478967, dtype: int64
```

In [ ]:
```
# adding count
df_ord_per_cus = df_inst_orders.groupby('user_id')['order_number'].count()
display(df_ord_per_cus)
```

```
user_id
2         2
4         2
5         1
6         2
7         2
         ..
206203    1
206206    7
206207    5
206208    9
206209    2
Name: order_number, Length: 157437, dtype: int64
```

In [ ]:
```
# plotting orders per customer
df_ord_per_cus.hist(figsize=(10,5), bins=25)
plt.title('Orders Per Customer')
plt.xlabel('Number of orders')
plt.ylabel('Number of users')
```

Out[ ]:
```
Text(0, 0.5, 'Number of users')
```

Results

The histogram is skewed right. The majority of people order a few times, once or twice, but generally under 5 times. The number of customers that order more than 5 times is proportionally small. This is vital information on customer retention. More research needs to be conducted to determine what factors lead to low retention, and methods of increasing retention.

[B3] What are the top 20 popular products (display their id and name)?

In [ ]:
```
# merge product tables, get 20 largest products
(df_order_prod
 .groupby('product_id')['order_id'].count().reset_index()
 .merge(df_prod,how='inner', on='product_id')
 .nlargest(20, 'order_id')
)[['product_id','product_name','order_id']]
```

Out[ ]:

	product_id	product_name	order_id
22808	24852	banana	66050
12025	13176	bag of organic bananas	53297
19370	21137	organic strawberries	37039
20077	21903	organic baby spinach	33971
43271	47209	organic hass avocado	29773
43788	47766	organic avocado	24689
43663	47626	large lemon	21495
15364	16797	strawberries	20018
24047	26209	limes	19690
25556	27845	organic whole milk	19600
25666	27966	organic raspberries	19197
21025	22935	organic yellow onion	15898
22908	24964	organic garlic	15292
41244	45007	organic zucchini	14584
35996	39275	organic blueberries	13879
45561	49683	cucumber kirby	13675
25889	28204	organic fuji apple	12544
5375	5876	organic lemon	12232
7543	8277	apple honeycrisp organic	11993
37301	40706	organic grape tomatoes	11781

Results

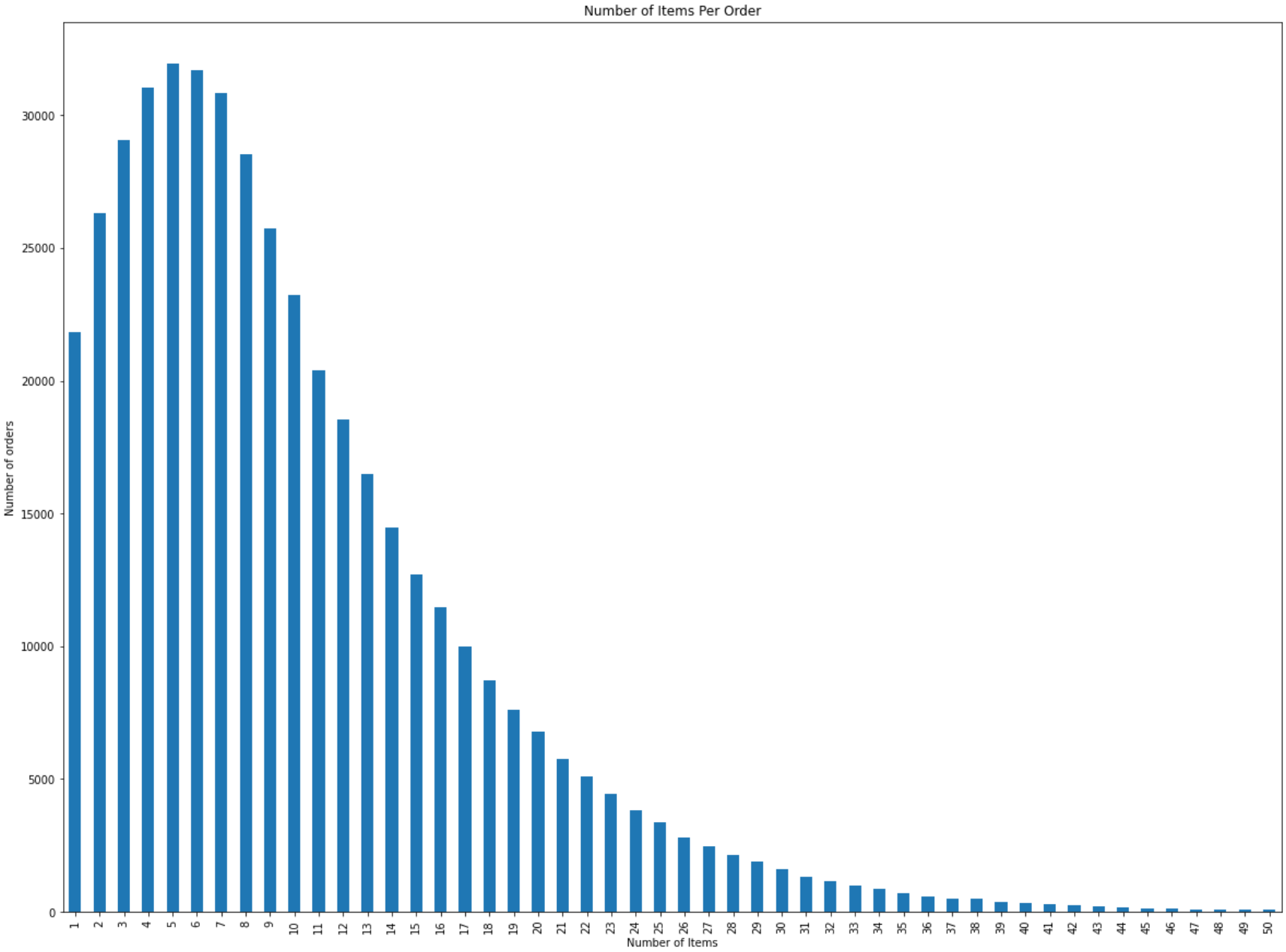
This data shows the most popular products on the platform. We see the list is comprised of perishables. These items likely have a high count of reorders, as they do not last long, and would need to be replenished often.

[C1] How many items do people typically buy in one order? What does the distribution look like?

In [ ]:
```
# creating one order dataframe
df_one_order = df_order_prod.groupby('order_id')['product_id'].count().value_counts().sort_index()
print(df_one_order)
```

```
1      21847
2      26292
3      29046
4      31054
5      31923
       ...
98         1
104        1
108        1
115        1
127        1
Name: product_id, Length: 90, dtype: int64
```

In [ ]:
```
# Cut off at 50 for ease of readability, beyond 50 is extraneous, does not give crucial information
df_one_order.head(50).plot(kind='bar',
                    title='Number of Items Per Order',
                    y='count',
                    ylabel='Number of orders',
                    x='add_to_cart_order',
                    xlabel='Number of Items',
                    figsize= (20,15),
                    xlim=[0,50]
                   )
plt.show()
```


Number of Items Per Order
Number of orders
0
5000
10000
15000
20000
25000
30000
Number of Items
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

Results

The distribution of the data is skewed right. Most people buy only a handfull of items, less than 10. Most customers buy a total of 5 items. The chart was cut off at 50 items, as the values beyond that point apper to be insignificant.

[C2] What are the top 20 items that are reordered most frequently (display their names and product IDs)?

In [ ]:
```
# group by reordered, merge with product id,
df_merged = df_order_prod.merge(df_prod, on='product_id')
```

In [ ]:
```
# Top 20 items reordered most frequently, closely resembles top 20 products. It would make sense that these two
# results are similar.
print(df_merged.query('reordered==1')[['product_id', 'product_name']].value_counts().head(20))
```

```
product_id  product_name
24852       banana                      55763
13176       bag of organic bananas      44450
21137       organic strawberries        28639
21903       organic baby spinach        26233
47209       organic hass avocado        23629
47766       organic avocado             18743
27845       organic whole milk          16251
47626       large lemon                 15044
27966       organic raspberries         14748
16797       strawberries                13945
26209       limes                       13327
22935       organic yellow onion        11145
24964       organic garlic              10411
45007       organic zucchini            10076
49683       cucumber kirby               9538
28204       organic fuji apple           8989
8277        apple honeycrisp organic     8836
39275       organic blueberries          8799
5876        organic lemon                8412
49235       organic half & half          8389
dtype: int64
```

Results

Results show the top 20 items reorderd the most. As anticipated, many of these items are perishables that are also on the top 20 items ordered list.

[C3] What are the top 20 items that people put in their carts first?

In [ ]:
```
# merging product names and order products
df_order_merge = df_order_prod.merge(df_prod, on='product_id')
print(df_order_merge)
```

```
         order_id  product_id  add_to_cart_order  reordered  \
0         2141543       11440                 17          0
1          147172       11440                  7          1
2         3341719       11440                  4          1
3         1938779       11440                  8          1
4         1728338       11440                 23          0
...           ...         ...                ...        ...
4545002    267402       45226                 14          0
4545003   2621676       25201                 10          0
4545004    937623       49153                  2          0
4545005    532895        8182                  1          0
4545006   3132243       40024                  1          0

                             product_name  aisle_id  department_id
0           chicken breast tenders breaded       129              1
1           chicken breast tenders breaded       129              1
2           chicken breast tenders breaded       129              1
3           chicken breast tenders breaded       129              1
4           chicken breast tenders breaded       129              1
...                                    ...       ...            ...
4545002   sweet teriyaki chicken oven sauce         5             13
4545003          crisp waters candle scents       101             17
4545004  shine collection brillance shampoo        22             11
4545005    total mint stripe gel toothpaste        20             11
4545006                 egg replacer powder        17             13

[4545007 rows x 7 columns]
```

In [ ]:
```
# Top 20 products put in the cart first, reordered with names instead of product ID, with unknown in data
df_order_merge.query("add_to_cart_order==1")[['product_name','add_to_cart_order']].value_counts().head(20)
```

Out[ ]:

product_name	add_to_cart_order	
banana	1	15562
bag of organic bananas	1	11026
organic whole milk	1	4363
organic strawberries	1	3946
organic hass avocado	1	3390
organic baby spinach	1	3336
organic avocado	1	3044
spring water	1	2336
strawberries	1	2308
organic raspberries	1	2024
sparkling water grapefruit	1	1914
organic half & half	1	1797
large lemon	1	1737
soda	1	1733
organic reduced fat milk	1	1397
limes	1	1370
hass avocados	1	1340
organic reduced fat 2% milk	1	1310
half & half	1	1309
raspberries	1	1246

dtype: int64

In [ ]:
```
# Top 20 products  put in the cart first, reordered with names instead of product ID, unknown product removed
df_merged.query("add_to_cart_order==1")[['product_name','product_id']].value_counts().head(20)
```

Out[ ]:

product_name	product_id	
banana	24852	15562
bag of organic bananas	13176	11026
organic whole milk	27845	4363
organic strawberries	21137	3946
organic hass avocado	47209	3390
organic baby spinach	21903	3336
organic avocado	47766	3044
spring water	19660	2336
strawberries	16797	2308
organic raspberries	27966	2024
sparkling water grapefruit	44632	1914
organic half & half	49235	1797
large lemon	47626	1737
soda	196	1733
organic reduced fat milk	38689	1397
limes	26209	1370
hass avocados	12341	1340
organic reduced fat 2% milk	5785	1310
half & half	27086	1309
organic yellow onion	22935	1246

dtype: int64

In [ ]:
```
# QC check on correct name and product ID
df_merged.query("product_id==24852")[['product_name']]
```

Out[ ]:

	product_name
356246	banana
356247	banana
356248	banana
356249	banana
356250	banana
...	...
422291	banana
422292	banana
422293	banana
422294	banana
422295	banana

66050 rows × 1 columns

In [ ]:
```
# creating merged dataframe
df_prod_merged = df_order_prod.merge(df_prod, on='product_id')
df_prod_merged = df_prod_merged.query('add_to_cart_order==1')[['product_id']].value_counts().head(20)
print(df_prod_merged)
```

```
product_id
24852         15562
13176         11026
27845          4363
21137          3946
47209          3390
21903          3336
47766          3044
19660          2336
16797          2308
27966          2024
44632          1914
49235          1797
47626          1737
196            1733
38689          1397
26209          1370
12341          1340
5785           1310
27086          1309
43352          1246
dtype: int64
```

Results

Again, we see a similar trend from the tow previous results. The top 20 products put in the cart first are similar to the most popular products list, and the most popular products reordered list.

Conclusions

After cleaning the data from duplicates and missing values, we were able to develop key insights on the Instacart platform. Using the several datasets provided by the company, we were able to illustrate the distribution of orders by the day of the week, and by the time of day. We saw that most orders were made on Sunday, and the least, on Wednesday. We also determined, when considering Wednesday and Saturday data, most orders were made during the middle of the day, with the amount tailing off in the late evening. Hardly any orders were made in the early morning, until around 7 am, on Wednesday or Saturday. These results suggest a probable correlation with typical sleep times. As such, resources should be implemented in the appropriate time frames, to maximize the amount of drivers available for deliveries. We determined the most popular products, as well as the most popular items reordered and placed in the cart first. These products were perishables: fruits, vegetables, and dairy products. Overall, we saw that most reorders were

made after 30 days, and also within 7 days. Local peaks in reorders appeared in 7 day increments, yet not many orders were placed in between those weekly time frames.