The Instacart Case Study

Instacart is an American company that operates as a same-day grocery delivery service. Customers select groceries through a web application from various retailers and delivered by a personal shopper. Instacart's service is mainly provided through a smartphone app, available on iOS and Android platforms, apart from its website.

The objective is to predict which previously purchased products will be in a user's next order.

Problem definition

The data that Instacart opened up include orders of 200,000 Instacart users with each user having between 4 and 100 orders. Instacart indicates each order in the data as prior, train or test. Prior orders describe the past behaviour of a user while train and test orders regard the future behaviour that we need to predict.

As a result, we want to predict which previously purchased products (prior orders) will be in a user's next order (train and test orders).

For the train orders Instacart reveals the results (i.e., the ordered products) while for the test orders we do not have this piece of information. Moreover, the future order of each user can be either train or test meaning that each user will be either a train or a test user.

Factors that can lead to reorder:

- Number of times product was ordered (frequency)
- Time/day of order
- Category
- Quantity
- Date since last order

In [1]:

```
# Importing packages

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import gc                           # Garbage Collector to free up memory
gc.enable()                         # Activate
```

In [2]:

```
# Importing all the data sets

orders = pd.read_csv(r'C:\Users\Madhvi\Desktop\IVY - Project\Session 6\orders.csv')
order_products_train = pd.read_csv(r'C:\Users\Madhvi\Desktop\IVY - Project\Session 6\order_products_train.csv')
order_products_prior = pd.read_csv(r'C:\Users\Madhvi\Desktop\IVY - Project\Session 6\order_products_prior.csv')
products = pd.read_csv(r'C:\Users\Madhvi\Desktop\IVY - Project\Session 6\products.csv')
aisles = pd.read_csv(r'C:\Users\Madhvi\Desktop\IVY - Project\Session 6\aisles.csv')
departments = pd.read_csv(r'C:\Users\Madhvi\Desktop\IVY - Project\Session 6\departments.csv')
```

In [3]:

```
### COMMANDS FOR CODING TESTING - Get 10% of users because data is big and might take some time to process
### Comment it if you want to run analysis on all the users

orders = orders.loc[orders.user_id.isin(orders.user_id.drop_duplicates().sample(frac=0.1, random_state=25))]
```

In [4]:

```
# Understanding data

orders.head()
```

Out[4]:

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order
54	2565571	7	prior	1	3	9	NaN
55	2402008	7	prior	2	1	19	30.0
56	121053	7	prior	3	0	18	30.0
57	1695742	7	prior	4	2	10	9.0
58	3321109	7	prior	5	5	18	3.0

In [5]:

```
order_products_train.head()
```

Out[5]:

	order_id	product_id	add_to_cart_order	reordered
0	1	49302	1	1
1	1	11109	2	1
2	1	10246	3	0
3	1	49683	4	0
4	1	43633	5	1

In [6]:

```
order_products_prior.head()
```

Out[6]:

	order_id	product_id	add_to_cart_order	reordered
0	2	33120	1	1
1	2	28985	2	1
2	2	9327	3	0
3	2	45918	4	1
4	2	30035	5	0

In [7]:

```
products.head()
```

Out[7]:

	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

In [8]:

```
aisles.head()
```

Out[8]:

	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

In [9]:

```
departments.head()
```

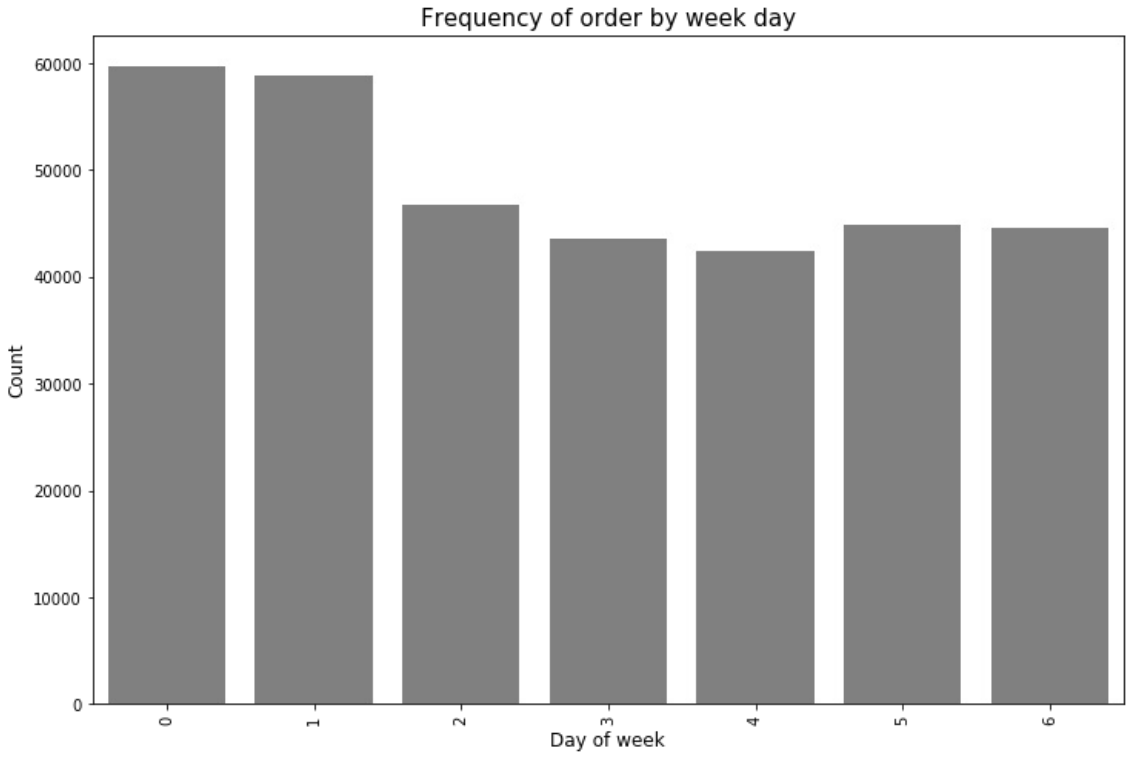
Out[9]:

	department_id	department
0	1	frozen
1	2	other
2	3	bakery
3	4	produce
4	5	alcohol

Let us see how the ordering habit changes with day of week.

In [10]:

```
plt.figure(figsize=(12,8))
sns.countplot(x="order_dow", data=orders, color = 'grey')
plt.ylabel('Count', fontsize=12)
plt.xlabel('Day of week', fontsize=12)
plt.xticks(rotation='vertical')
plt.title("Frequency of order by week day", fontsize=15)
plt.show()
```

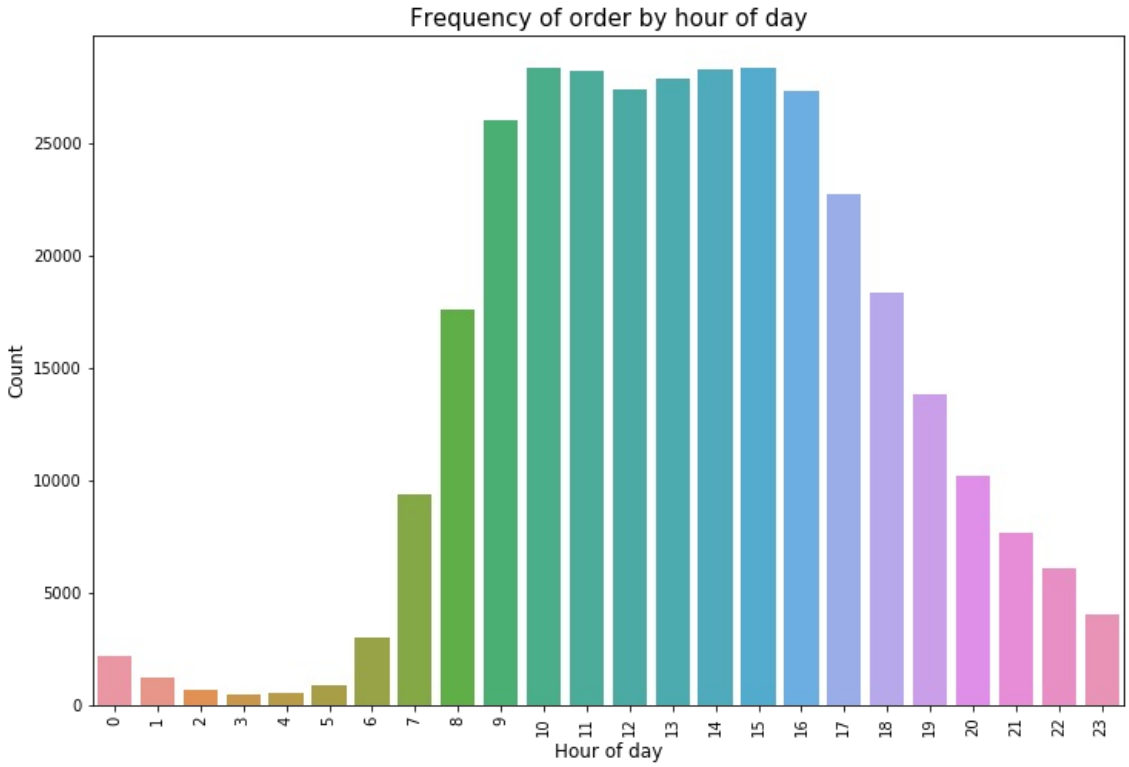


Seems like 0 and 1 is Saturday and Sunday when the orders are high and low during Wednesday.

Now we shall see how the distribution is with respect to time of the day.

In [11]:

```
plt.figure(figsize=(12,8))
sns.countplot(x="order_hour_of_day", data=orders)
plt.ylabel('Count', fontsize=12)
plt.xlabel('Hour of day', fontsize=12)
plt.xticks(rotation='vertical')
plt.title("Frequency of order by hour of day", fontsize=15)
plt.show()
```

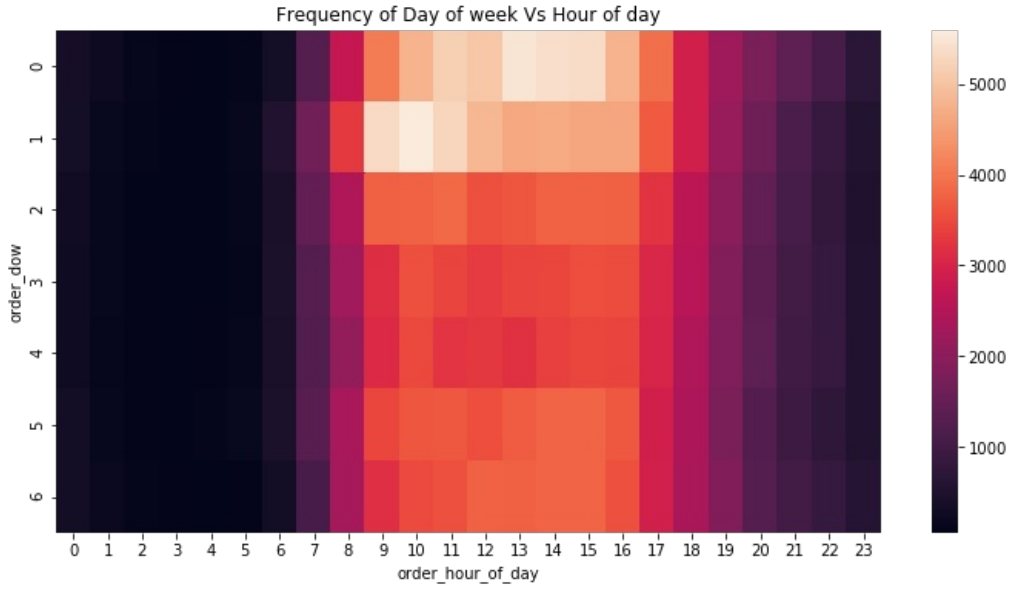


So majority of the orders are made during day time. Now let us combine the day of week and hour of day to see the distribution.

In [12]:

```
grouped_df = orders.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)
plt.title("Frequency of Day of week Vs Hour of day")
plt.show()
```

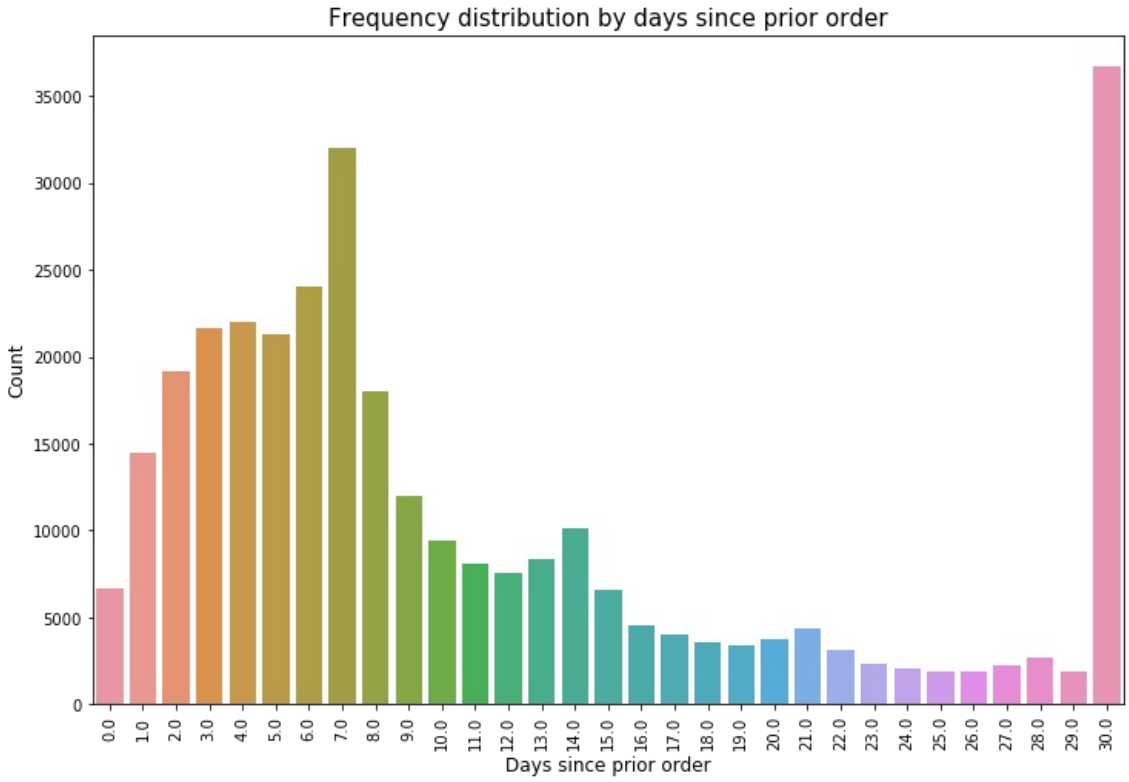


Seems Satuday evenings and Sunday mornings are the prime time for orders.

Now let us check the time interval between the orders.

In [13]:

```
plt.figure(figsize=(12,8))
sns.countplot(x="days_since_prior_order", data=orders)
plt.ylabel('Count', fontsize=12)
plt.xlabel('Days since prior order', fontsize=12)
plt.xticks(rotation='vertical')
plt.title("Frequency distribution by days since prior order", fontsize=15)
plt.show()
```



Looks like customers order once in every week (check the peak at 7 days) or once in a month (peak at 30 days). We could also see smaller peaks at 14, 21 and 28 days (weekly intervals).

Since our objective is to figure out the re-orders, let us check out the re-order percentage in prior set and train set.

In [14]:

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

Out[14]:

```
0.5896974667922161
```

In [15]:

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

Out[15]:

```
0.5985944127509629
```

On an average, about 59% of the products in an order are re-ordered products.

No re-ordered products:

Now that we have seen 59% of the products are re-ordered, there will also be situations when none of the products are re-ordered. Let us check that now.

In [16]:

```
grouped_df = order_products_prior.groupby("order_id")["reordered"].aggregate("sum").reset_index()
grouped_df["reordered"].ix[grouped_df["reordered"]>1] = 1
grouped_df.reordered.value_counts() / grouped_df.shape[0]
```

```
D:\Anaconda\lib\site-packages\ipykernel_launcher.py:2: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated
```

Out[16]:

```
1    0.879151
0    0.120849
Name: reordered, dtype: float64
```

In [17]:

```
grouped_df = order_products_train.groupby("order_id")["reordered"].aggregate("sum").reset_index()
grouped_df["reordered"].ix[grouped_df["reordered"]>1] = 1
grouped_df.reordered.value_counts() / grouped_df.shape[0]
```

```
D:\Anaconda\lib\site-packages\ipykernel_launcher.py:2: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated
```

Out[17]:

```
1    0.93444
0    0.06556
Name: reordered, dtype: float64
```

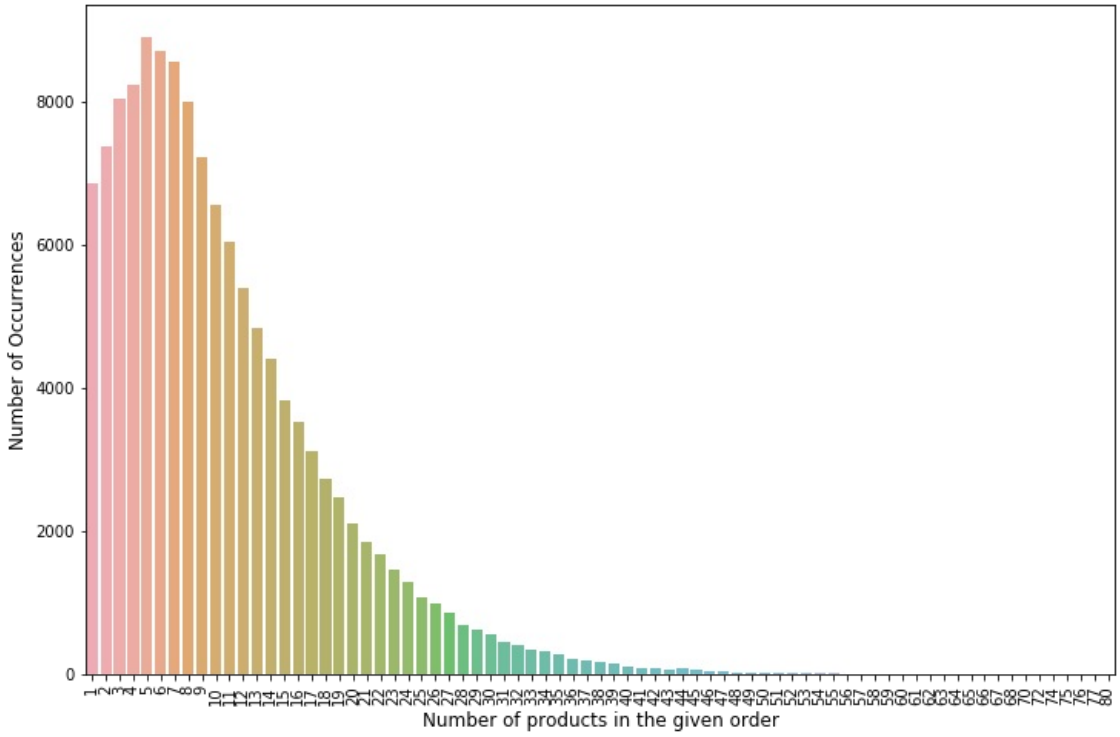
About 12% of the orders in prior set has no re-ordered items while in the train set it is 7%.

Now let us see the number of products bought in each order.

In [18]:

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

plt.figure(figsize=(12,8))
sns.barplot(cnt_srs.index, cnt_srs.values, alpha=0.8)
plt.ylabel('Number of Occurrences', fontsize=12)
plt.xlabel('Number of products in the given order', fontsize=12)
plt.xticks(rotation='vertical')
plt.show()
```



A right tailed distribution with the maximum value at 5.!

In [ ]:

```

```

In [19]:

```
# We convert character variables into category.
# In Python, a categorical variable is called category and has a fixed number of different values

aisles['aisle'] = aisles['aisle'].astype('category')
departments['department'] = departments['department'].astype('category')
orders['eval_set'] = orders['eval_set'].astype('category')
products['product_name'] = products['product_name'].astype('category')
```

Create a DataFrame with the orders and the products that have been purchased on prior orders (op)

In [20]:

```
#Merge the orders DF with order_products_prior by their order_id, keep only these rows with order_id that they ar
e appear on both DFs

op = orders.merge(order_products_prior, on='order_id', how='inner')
op.head()
```

Out[20]:

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order	product_id	add_to_cart_order	r
0	2565571	7	prior	1	3	9	NaN	45628	1	
1	2565571	7	prior	1	3	9	NaN	39275	2	
2	2565571	7	prior	1	3	9	NaN	6361	3	
3	2565571	7	prior	1	3	9	NaN	45066	4	
4	2565571	7	prior	1	3	9	NaN	13249	5	

Create Predictor Variables

We are now ready to identify and calculate predictor variables based on the provided data. We can create various types of predictors such as:

User predictors describing the behavior of a user e.g. total number of orders of a user. Product predictors describing characteristics of a product e.g. total number of times a product has been purchased. User & product predictors describing the behavior of a user towards a specific product e.g. total times a user ordered a specific product.

Create user predictors

Number of orders per customer

We calculate the total number of placed orders per customer. We create a user DataFrame to store the results.

In [21]:

```
## First approach in one step:
# Create distinct groups for each user, identify the highest order number in each group, save the new column to a
DataFrame
user = op.groupby('user_id')['order_number'].max().to_frame('user_t_orders') #
user.head()

## Second approach in two steps:
#1. Save the result as DataFrame with Double brackets --> [[ ]]
#user = op.groupby('user_id')[['order_number']].max()
#2. Rename the label of the column
#user.columns = ['user_t_orders']
#user.head()
```

Out[21]:

user_id	user_t_orders
7	20
14	13
22	15
24	18
29	18

In [22]:

```
# Reset the index of the DF so to bring user_id from index to column (pre-requisite for step 2.4)

user = user.reset_index()
user.head()
```

Out[22]:

	user_id	user_t_orders
0	7	20
1	14	13
2	22	15
3	24	18
4	29	18

Create product predictors

Number of purchases for each product

We calculate the total number of purchases for each product (from all customers). We create a prd DataFrame to store the results.

In [23]:

```
# Create distinct groups for each product, count the orders, save the result for each product to a new DataFrame

prd = op.groupby('product_id')['order_id'].count().to_frame('prd_t_purchases') #
prd.head()
```

Out[23]:

product_id	prd_t_purchases
1	205
2	13
3	15
4	37
6	1

In [24]:

```
# Reset the index of the DF so to bring product_id rom index to column (pre-requisite for step 2.4)

prd = prd.reset_index()
prd.head()
```

Out[24]:

	product_id	prd_t_purchases
0	1	205
1	2	13
2	3	15
3	4	37
4	6	1

Create user-product predictors

How many times a user bought a product

We create different groups that contain all the rows for each combination of user and product. Then with the aggregation function .count( ) we get how many times each user bought a product. We save the results on new uxp DataFrame.

In [25]:

```
# Create distinct groups for each combination of user and product, count orders, save the result for each user X
product to a new DataFrame

uxp = op.groupby(['user_id', 'product_id'])['order_id'].count().to_frame('uxp_t_bought') #
uxp.head()
```

Out[25]:

user_id	product_id	uxp_t_bought
7	**274**	1
	519	2
	4920	7
	4945	3
	6361	5

In [26]:

```
# Reset the index of the DF so to bring user_id & product_id rom indices to columns (pre-requisite for step 2.4)

uxp = uxp.reset_index()
uxp.head()
```

Out[26]:

	user_id	product_id	uxp_t_bought
0	7	274	1
1	7	519	2
2	7	4920	7
3	7	4945	3
4	7	6361	5

Merge all features

We now merge the DataFrames with the three types of predictors that we have created (i.e., for the users, the products and the combinations of users and products).

In [27]:

```
#Merge uxp features with the user features
#Store the results on a new DataFrame

data = uxp.merge(user, on='user_id', how='left')
data.head()
```

Out[27]:

	user_id	product_id	uxp_t_bought	user_t_orders
0	7	274	1	20
1	7	519	2	20
2	7	4920	7	20
3	7	4945	3	20
4	7	6361	5	20

In [28]:

```
#Merge uxp & user features (the new DataFrame) with prd features

data = data.merge(prd, on='product_id', how='left')
data.head()
```

Out[28]:

	user_id	product_id	uxp_t_bought	user_t_orders	prd_t_purchases
0	7	274	1	20	232
1	7	519	2	20	111
2	7	4920	7	20	8126
3	7	4945	3	20	599
4	7	6361	5	20	333

Delete previous DataFrames

The information from the DataFrames that we have created to store our features (op, user, prd, uxp) is now stored on data.

As we won't use them anymore, we now delete them.

In [29]:

```
del op, user, prd, uxp
gc.collect()
```

Out[29]:

```
202
```

Create train and test DataFrames

- We select the orders DataFrame to keep only the future orders (labeled as "train" & "test).
- Keep only the columns of our desire ['eval_set', 'order_id'] **AND** 'user_id' as is the matching key with our data DataFrame
- Merge data DataFrame with the information for the future order of each customer using as matching key the 'user_id'

To filter and select the columns of our desire on orders (the 2 first steps) there are numerous approaches:

In [30]:

```
## First approach:
# In two steps keep only the future orders from all customers: train & test
orders_future = orders[((orders.eval_set=='train') | (orders.eval_set=='test'))]
orders_future = orders_future[ ['user_id', 'eval_set', 'order_id'] ]
orders_future.head(10)

## Second approach (if you want to test it you have to re-run the notebook):
# In one step keep only the future orders from all customers: train & test
#orders_future = orders.loc[((orders.eval_set=='train') | (orders.eval_set=='test')), ['user_id', 'eval_set', 'or
der_id'] ]
#orders_future.head(10)

## Third approach (if you want to test it you have to re-run the notebook):
# In one step exclude all the prior orders so to deal with the future orders from all customers
#orders_future = orders.loc[orders.eval_set!='prior', ['user_id', 'eval_set', 'order_id'] ]
#orders_future.head(10)
```

Out[30]:

	user_id	eval_set	order_id
74	7	train	525192
129	14	train	2316178
272	22	test	139655
296	24	train	965160
439	29	train	3110252
485	34	train	698604
570	38	train	3173750
588	40	test	2431024
669	48	train	2924697
999	64	train	2639013

In [31]:

```
# bring the info of the future orders to data DF

data = data.merge(orders_future, on='user_id', how='left')
data.head(10)
```

Out[31]:

	user_id	product_id	uxp_t_bought	user_t_orders	prd_t_purchases	eval_set	order_id
0	7	274	1	20	232	train	525192
1	7	519	2	20	111	train	525192
2	7	4920	7	20	8126	train	525192
3	7	4945	3	20	599	train	525192
4	7	6361	5	20	333	train	525192
5	7	8277	3	20	8990	train	525192
6	7	8518	3	20	6828	train	525192
7	7	9598	3	20	523	train	525192
8	7	10504	1	20	731	train	525192
9	7	10895	3	20	547	train	525192

Prepare the train DataFrame

- We keep only the customers who are labelled as "train" from the competition
- For these customers we get from order_products_train the products that they have bought, in order to create the response variable (reordered:1 or 0)
- We make all the required manipulations on that dataset and we remove the columns that are not predictors

So now we filter the data DataFrame so to keep only the train users:

In [32]:

```
#Keep only the customers who we know what they bought in their future order

data_train = data[data.eval_set=='train'] #
data_train.head()
```

Out[32]:

	user_id	product_id	uxp_t_bought	user_t_orders	prd_t_purchases	eval_set	order_id
0	7	274	1	20	232	train	525192
1	7	519	2	20	111	train	525192
2	7	4920	7	20	8126	train	525192
3	7	4945	3	20	599	train	525192
4	7	6361	5	20	333	train	525192

In [33]:

```
#Get from order_products_train all the products that the train users bought bought in their future order

data_train = data_train.merge(order_products_train[['product_id','order_id', 'reordered']], on=['product_id','ord
er_id'], how='left' )
data_train.head(15)
```

Out[33]:

	user_id	product_id	uxp_t_bought	user_t_orders	prd_t_purchases	eval_set	order_id	reordered
0	7	274	1	20	232	train	525192	NaN
1	7	519	2	20	111	train	525192	NaN
2	7	4920	7	20	8126	train	525192	NaN
3	7	4945	3	20	599	train	525192	NaN
4	7	6361	5	20	333	train	525192	NaN
5	7	8277	3	20	8990	train	525192	NaN
6	7	8518	3	20	6828	train	525192	NaN
7	7	9598	3	20	523	train	525192	NaN
8	7	10504	1	20	731	train	525192	NaN
9	7	10895	3	20	547	train	525192	NaN
10	7	11520	1	20	4005	train	525192	NaN
11	7	12196	2	20	97	train	525192	NaN
12	7	13176	1	20	38499	train	525192	NaN
13	7	13198	8	20	1240	train	525192	1.0
14	7	13249	1	20	1446	train	525192	NaN

Data Preparation

In [34]:

```
#Where the previous merge, left a NaN value on reordered column means that the customers they haven't bought the
product. We change the value on them to 0.

data_train['reordered'] = data_train['reordered'].fillna(0)
data_train.head(15)
```

Out[34]:

	user_id	product_id	uxp_t_bought	user_t_orders	prd_t_purchases	eval_set	order_id	reordered
0	7	274	1	20	232	train	525192	0.0
1	7	519	2	20	111	train	525192	0.0
2	7	4920	7	20	8126	train	525192	0.0
3	7	4945	3	20	599	train	525192	0.0
4	7	6361	5	20	333	train	525192	0.0
5	7	8277	3	20	8990	train	525192	0.0
6	7	8518	3	20	6828	train	525192	0.0
7	7	9598	3	20	523	train	525192	0.0
8	7	10504	1	20	731	train	525192	0.0
9	7	10895	3	20	547	train	525192	0.0
10	7	11520	1	20	4005	train	525192	0.0
11	7	12196	2	20	97	train	525192	0.0
12	7	13176	1	20	38499	train	525192	0.0
13	7	13198	8	20	1240	train	525192	1.0
14	7	13249	1	20	1446	train	525192	0.0

In [35]:

```
#We set user_id and product_id as the index of the DF

data_train = data_train.set_index(['user_id', 'product_id'])
data_train.head(15)
```

Out[35]:

		uxp_t_bought	user_t_orders	prd_t_purchases	eval_set	order_id	reordered
user_id	**product_id**						
7	**274**	1	20	232	train	525192	0.0
	519	2	20	111	train	525192	0.0
	4920	7	20	8126	train	525192	0.0
	4945	3	20	599	train	525192	0.0
	6361	5	20	333	train	525192	0.0
	8277	3	20	8990	train	525192	0.0
	8518	3	20	6828	train	525192	0.0
	9598	3	20	523	train	525192	0.0
	10504	1	20	731	train	525192	0.0
	10895	3	20	547	train	525192	0.0
	11520	1	20	4005	train	525192	0.0
	12196	2	20	97	train	525192	0.0
	13176	1	20	38499	train	525192	0.0
	13198	8	20	1240	train	525192	1.0
	13249	1	20	1446	train	525192	0.0

In [36]:

```
#We remove all non-predictor variables

data_train = data_train.drop(['eval_set', 'order_id'], axis=1)
data_train.head(15)
```

Out[36]:

		uxp_t_bought	user_t_orders	prd_t_purchases	reordered
user_id	product_id				
7	**274**	1	20	232	0.0
	519	2	20	111	0.0
	4920	7	20	8126	0.0
	4945	3	20	599	0.0
	6361	5	20	333	0.0
	8277	3	20	8990	0.0
	8518	3	20	6828	0.0
	9598	3	20	523	0.0
	10504	1	20	731	0.0
	10895	3	20	547	0.0
	11520	1	20	4005	0.0
	12196	2	20	97	0.0
	13176	1	20	38499	0.0
	13198	8	20	1240	1.0
	13249	1	20	1446	0.0

Prepare the test DataFrame

- Keep only the customers who are labelled as test
- Set as index the column(s) that uniquely describe each row (in our case "user_id" & "product_id")
- Remove the columns that are predictors (in our case:'eval_set', 'order_id')

In [37]:

```
#Keep only the future orders from customers who are labelled as test

data_test = data[data.eval_set=='test'] #
data_test.head()
```

Out[37]:

	user_id	product_id	uxp_t_bought	user_t_orders	prd_t_purchases	eval_set	order_id
210	22	2452	2	15	1096	test	139655
211	22	4217	1	15	68	test	139655
212	22	4421	1	15	1238	test	139655
213	22	5212	1	15	2379	test	139655
214	22	5450	1	15	5120	test	139655

In [38]:

```
#We set user_id and product_id as the index of the DF

data_test = data_test.set_index(['user_id', 'product_id'])
data_test.head()
```

Out[38]:

		uxp_t_bought	user_t_orders	prd_t_purchases	eval_set	order_id
user_id	product_id					
22	**2452**	2	15	1096	test	139655
	4217	1	15	68	test	139655
	4421	1	15	1238	test	139655
	5212	1	15	2379	test	139655
	5450	1	15	5120	test	139655

In [39]:

```
#We remove all non-predictor variables
data_test = data_test.drop(['eval_set','order_id'], axis=1)
#Check if the data_test DF, has the same number of columns as the data_train DF, excluding the response variable
data_test.head()
```

Out[39]:

		uxp_t_bought	user_t_orders	prd_t_purchases
user_id	product_id			
22	**2452**	2	15	1096
	4217	1	15	68
	4421	1	15	1238
	5212	1	15	2379
	5450	1	15	5120

Create predictive model

To create the predictive model we:

- We create a DataFrame with all the predictors, named X_train and a Series with the response, named y_train
- We initiata a Logistic regression model with a specific random_state (so we can reproduce if we want to our model).
- Finally we train our model with the X_train and y_train data.

In [40]:

```
#IMPORT REQUIRED PACKAGES

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split #validate algorithm
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score, precision_score,f1_score, roc_auc_sco
re  #validate algorithm
```

In [41]:

```
#CREATE X_train, y_train

X_train, y_train = data_train.drop('reordered', axis=1), data_train.reordered
X_train, X_val, y_train, y_val = train_test_split(data_train.drop('reordered', axis=1), data_train.reordered, tes
t_size=0.8, random_state=42)
```

In [42]:

```
# INITIATE AND TRAIN MODEL

log = LogisticRegression(random_state=42)
model = log.fit(X_train, y_train)
```

```
D:\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver wi
ll be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

In [43]:

```
# SCORE MODEL

log.score(X_val,y_val ) # validate algorithm
```

Out[43]:

```
0.9034229701156011
```

In [44]:

```
# Predicting values

y_pred = model.predict(X_val)
y_pred
```

Out[44]:

```
array([0., 0., 0., ..., 0., 0., 0.])
```

In [45]:

```
print("Trainig accuracy",log.score(X_train,y_train))
print("Testing accuracy",log.score(X_val, y_val))
```

```
Trainig accuracy 0.9025528689185075
Testing accuracy 0.9034229701156011
```

In [46]:

```
# Predict values for test data with our model the results are saved as a Python array
test_pred = model.predict(data_test).astype(int)
test_pred[0:20] #display the first 20 predictions of the numpy array
```

Out[46]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [47]:

```
#Save the prediction in a new column in the data_test DF
data_test['prediction'] = test_pred
data_test.head()
```

Out[47]:

		uxp_t_bought	user_t_orders	prd_t_purchases	prediction
user_id	product_id				
22	2452	2	15	1096	0
	4217	1	15	68	0
	4421	1	15	1238	0
	5212	1	15	2379	0
	5450	1	15	5120	0

In [48]:

```
#Reset the index
final = data_test.reset_index()
#Keep only the required columns to create our submission file (Chapter 6)
final = final[['product_id', 'user_id', 'prediction']]

gc.collect()
final.head(45)
```

Out[48]:

	product_id	user_id	prediction
0	2452	22	0
1	4217	22	0
2	4421	22	0
3	5212	22	0
4	5450	22	0
5	7088	22	0
6	7948	22	0
7	8518	22	0
8	13176	22	0
9	14678	22	0
10	14966	22	0
11	15392	22	0
12	15984	22	0
13	16987	22	0
14	17794	22	0
15	21903	22	0
16	22115	22	0
17	22935	22	0
18	22963	22	0
19	24506	22	0
20	24964	22	0
21	27171	22	0
22	27845	22	0
23	32096	22	0
24	32655	22	0
25	35221	22	0
26	36311	22	0
27	36724	22	0
28	38312	22	0
29	39040	22	0
30	41950	22	0
31	44359	22	0
32	44968	22	0
33	49533	22	0
34	2238	40	0
35	4193	40	0
36	5322	40	0
37	5450	40	0
38	5699	40	0
39	6975	40	0
40	9290	40	0
41	11777	40	0
42	13176	40	1
43	13740	40	0
44	13870	40	0

Another Method

In [49]:

```
# #Build the logistic regression model
import statsmodels.api as sm
logit = sm.Logit(y_train, sm.add_constant(X_train))
lg = logit.fit()
```

```
D:\Anaconda\lib\site-packages\numpy\core\fromnumeric.py:2389: FutureWarning: Method .ptp is deprecat
ed and will be removed in a future version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)

Optimization terminated successfully.
         Current function value: 0.283800
         Iterations 7
```

In [50]:

```
#Summary of logistic regression
from scipy import stats
stats.chisqprob = lambda chisq, df: stats.chi2.sf(chisq, df)
print(lg.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:              reordered   No. Observations:               167537
Model:                          Logit   Df Residuals:                   167533
Method:                           MLE   Df Model:                            3
Date:                Tue, 30 Jun 2020   Pseudo R-squ.:                  0.1192
Time:                        14:50:00   Log-Likelihood:                -47547.
converged:                       True   LL-Null:                       -53979.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==================================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------
const              -2.0661      0.014   -149.868      0.000      -2.093      -2.039
uxp_t_bought        0.2087      0.002     85.465      0.000       0.204       0.213
user_t_orders      -0.0424      0.001    -61.052      0.000      -0.044      -0.041
prd_t_purchases  3.053e-05   1.12e-06     27.293      0.000    2.83e-05    3.27e-05
==================================================================================
```

Interpretation of Pseudo R^2

A pseudo R^2 of 11.9% indicates that 11.9% of the uncertainty of the intercept only model is explained by the full model

Calculate the odds ratio from the coef using the formula odds ratio=exp(coef)

Calculate the probability from the odds ratio using the formula probability = odds / (1+odds)

In [51]:

```
#Calculate Odds Ratio, probability
#create a data frame to collate Odds ratio, probability and p-value of the coef
import numpy as np
lgcoef = pd.DataFrame(lg.params, columns=['coef'])
lgcoef.loc[:, "Odds_ratio"] = np.exp(lgcoef.coef)
lgcoef['probability'] = lgcoef['Odds_ratio']/(1+lgcoef['Odds_ratio'])
lgcoef['pval']=lg.pvalues
pd.options.display.float_format = '{:.2f}'.format
```

In [52]:

```
# FIlter by significant p-value (pval <0.1) and sort descending by Odds ratio
lgcoef = lgcoef.sort_values(by="Odds_ratio", ascending=False)
pval_filter = lgcoef['pval']<=0.1
lgcoef[pval_filter]
```

Out[52]:

	coef	Odds_ratio	probability	pval
uxp_t_bought	0.21	1.23	0.55	0.00
prd_t_purchases	0.00	1.00	0.50	0.00
user_t_orders	-0.04	0.96	0.49	0.00
const	-2.07	0.13	0.11	0.00

Measures of Accuracy

In [53]:

```
print('AIC:', (2*-47547)+(2*4))
```

AIC: -95086

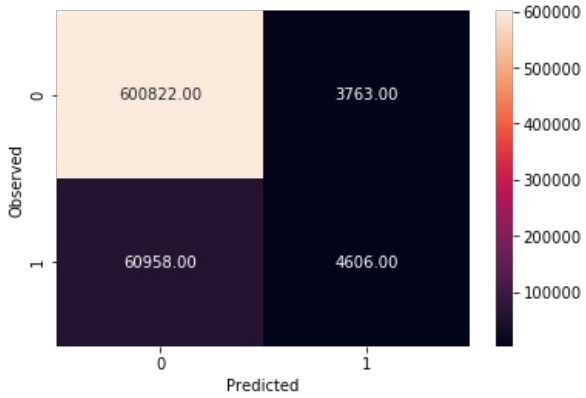
In [54]:

```
## function to get confusion matrix in a proper format
import seaborn as sns
import matplotlib.pyplot as plt
def draw_cm( actual, predicted ):
    cm = confusion_matrix( actual, predicted)
    sns.heatmap(cm, annot=True,  fmt='.2f', xticklabels = [0,1] , yticklabels = [0,1] )
    plt.ylabel('Observed')
    plt.xlabel('Predicted')
    plt.show()
```

In [55]:

```
print('Confusion Matrix')
print(draw_cm(y_val,y_pred))
```

Confusion Matrix



None

In [56]:

```
print("Recall:",recall_score(y_val,y_pred))
```

Recall: 0.07025196754316393

In [57]:

```
print("Precision:",precision_score(y_val,y_pred))
```

Precision: 0.5503644401959613

In [58]:

```
print("F1 Score:",f1_score(y_val,y_pred))
```

F1 Score: 0.12459929936564186

In [59]:

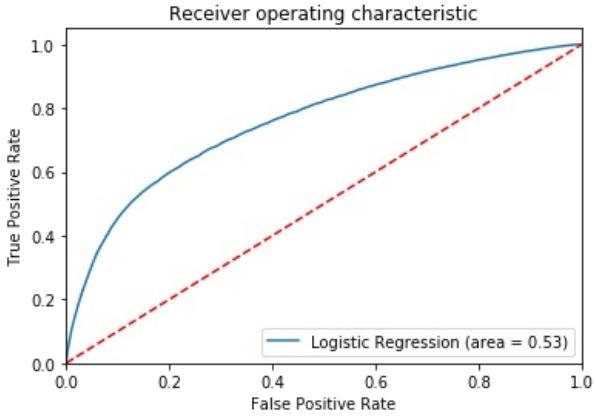
```
print("Roc Auc Score:",roc_auc_score(y_val,y_pred))
```

Roc Auc Score: 0.5320139317028075

In [60]:

```
#AUC ROC curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

logit_roc_auc = roc_auc_score(y_val, log.predict(X_val))
fpr, tpr, thresholds = roc_curve(y_val, log.predict_proba(X_val)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



In [ ]:

```
```