# FinalProject 1.8

January 24, 2020

## 1 Ask 1:

### 1.0.1 Identify and describe your dataset

Our dataset is about Brazilian E-commerce orders made at Olist store, which is the largest department store in Brazilian marketplaces. There are seven datasets in our database, including customers dataset, orders dataset, order item dataset, order payments dataset, order reviews dataset, products dataset and sellers dataset, which allow us to analyze data in various perspectives and view each order from different dimensions.

In the customers dataset, information about the customer are included, such as the state and city location of each customer. Moreover, each customer has a unique customer_id based on each order, which can used to identify customers who repurchased at Olist store.

In the orders dataset, information about orders are included, such as the order status, when is the purchase time of the order, the estimated and actual delivery date of the order. Since we have the the estimated and actual delivery date of the order in the dataset, we additionally add a new column for calculating the difference of these two dates, which we can know more about the delivery of orders.

In the order items dataset, information about items purchased in each order are included, such as the price of the item, which seller provide the item, and the freight value of the item.

In the order payments dataset, information about what payment is used for each orders are included, such as the payment type and payment value.

In the order reviews dataset, information about reviews made by customers are included, such as the review score, review comment, when is the date the satisfaction survey was sent to the customer and the date customer answered the satisfaction survey. Since we have the date the satisfaction survey was sent to the customer and the date customer answered the satisfaction survey in the dataset, we additionally add a new column for calculating the difference of these two dates, which we can know more about how customer response to the satisfaction survey.

In the products dataset, information about all products sold by Olist are included, such as the product id, product category name, and product size.

In the sellers dataset, information about sellers that provide products for orders are included, such as the seller id and the state and city location of the seller.

### 1.0.2 Identify dataset source

https://www.kaggle.com/olistbr/brazilian-ecommerce#olist_sellers_dataset.csv

### 1.0.3 Why is important and what appeals to you about it?

This dataset was provided by Olist, the largest department store in Brazilian marketplaces. This database allows us to have a better understanding about the Olist Onlie store's business, sales, market trends and customer base. We think this database is really close to real life business operation and it helps us to get the sense of how to use data to extract information and improve business operation. That is why we think it is important to use this database and learn analytics skills from real life business example.

### 1.0.4 Describe the analytical questions you want to answer with the data:

Question 1: Find the number of orders by day of the week.

Question 2: Find the top 10 product categories based on average price

Question 3: Find the average days of product delays of each state based on the difference between estimatated delivery days and actural delivery days.

## 2 Ask 2

### 2.0.1 wrangle the data into a format suitable for Dimensional modeling analysis. This may involve: – Cleaning, filtering, merging , modeling steps

**Merging and cleaning** We used JMP to wrangle the data. We first joined all datasets into one single dataset using order id. Then we deleted useless columns such including payment_installments, payment_sequential, order_approved_at, order_delivered_carrier_date, product_name_lenght, product_description_lenght, product_photos_qty.

**Creating new columns** We used JMP to create two new columns, difference_delivered_purchase and difference_estimated_delivered. The formula of difference_delivered_purchase is (order_delivered_customer_date - order_purchase_timestamp) / (3600 * 24), which represents the days between customers make a purchase and receive the order. The formula of difference_estimated_delivered is (order_estimated_delivery_date - order_delivered_customer_date) / (3600 * 24), which represents the days between the estimated delivery date and the delivery date.

[3]:
```
!pip freeze | grep -E 'ipython-sql|psycopg2'
```

```
ipython-sql==0.3.9
psycopg2==2.7.5
psycopg2-binary==2.7.5
```

[2]:
```
!dropdb -U student final_project
```

[3]:
```
!createdb -U student final_project
```

[1]:
```
%load_ext sql
```

[2]:
```
%sql postgresql://student@/final_project
```

[2]:
```
'Connected: student@final_project'
```

### 2.1 Data Setup

[7]:
```
## lets examine data. Moving it first to more friendly file name
!mv order_customer_payments_item_product_seller.csv data.csv
```

[7]:
```
!wc -l data.csv
```

```
117602 data.csv
```

[8]:
```
## use csvcut to find the heading (attribute labels) of the Q1 file
!csvcut -n data.csv
```

```
  1: order_purchase_timestamp
  2: order_delivered_customer_date
  3: difference_delivered_purchase
  4: order_estimated_delivery_date
  5: difference_estimated_delivered
  6: customer_unique_id
  7: customer_zip_code_prefix
  8: customer_city
  9: customer_state
 10: seller_id
 11: seller_zip_code_prefix
 12: seller_city
 13: seller_state
 14: product_id
 15: price
 16: freight_value
 17: product_category_name
 18: product_weight_g
 19: product_length_cm
 20: product_height_cm
 21: product_width_cm
```

### 2.2 Create Table and Import

[9]:
```
## let's first have a feeling for the data values in the 21 fields
!head -n 10 data.csv
```

```
order_purchase_timestamp,order_delivered_customer_date,difference_delivered_pur
chase,order_estimated_delivery_date,difference_estimated_delivered,customer_uniq
ue_id,customer_zip_code_prefix,customer_city,customer_state,seller_id,seller_zip
_code_prefix,seller_city,seller_state,product_id,price,freight_value,product_cat
egory_name,product_weight_g,product_length_cm,product_height_cm,product_width_cm
2017/5/5 16:12,2017/6/2 16:57,28.03142361,2017/5/30 0:00,-3.706759259,b452742346
9300ee354458e1b5f961be,32223,contagem,MG,3442f8959a84dea7ee197c632cb2df15,13023,
campinas,SP,f4621f8ad6f54a2e3c408884068be46d,101.7,15.92,esporte_lazer,600,35,15
,28
2017/8/30 11:47,2017/9/1 16:51,2.210810185,2017/9/20 0:00,18.29761574,af0f26435f
ade1ca984d9affda307199,9310,maua,SP,3442f8959a84dea7ee197c632cb2df15,13023,campi
nas,SP,325a06bcce0da45b7f4ecf2797dd40e4,10.8,2.42,esporte_lazer,300,16,5,15
2017/8/21 20:35,2017/8/30 16:07,8.813530093,2017/9/1
0:00,1.328321759,f421a2a66b69dbfe6db0c87845281a90,4661,sao paulo,SP,3442f8959a84
dea7ee197c632cb2df15,13023,campinas,SP,ffb64e34a37740dafb6c88f1abd1fa61,106.2,9.
56,esporte_lazer,700,43,15,35
2017/4/28 14:20,2017/5/9 14:27,11.00483796,2017/6/1 0:00,22.3978588,00ac9cd5c4ad
19e16e7c6f6864711737,37500,itajuba,MG,d1b65fc7debc3361ea86b5f14c68d2e2,13844,mog
i guacu,SP,765c417cdc38443aaa558a0159a98591,209.9,21.55,malas_acessorios,3500,40
,55,25
2017/4/27 9:09,2017/5/4 13:20,7.173877315,2017/6/6
0:00,32.44436343,51dc56123336c573f2977f5da81b17b9,20251,rio de
janeiro,RJ,d1b65fc7debc3361ea86b5f14c68d2e2,13844,mogi guacu,SP,765c417cdc38443a
aa558a0159a98591,209.9,21.55,malas_acessorios,3500,40,55,25
2018/4/9 23:40,2018/4/30 18:41,20.79236111,2018/4/30
0:00,-0.778541667,177e10134f99776d8a2b0c10c3fed38c,37190,tres
pontas,MG,d1b65fc7debc3361ea86b5f14c68d2e2,13844,mogi guacu,SP,cb378611bbb39f171
6b4f0c335201448,399.99,36.34,malas_acessorios,16850,38,58,25
2017/5/7 12:42,2017/5/19 10:07,11.8921875,2017/6/1 0:00,12.57799769,28b9099dc657
7fdeceb2e4468e69f556,88440,imbuia,SC,d1b65fc7debc3361ea86b5f14c68d2e2,13844,mogi
guacu,SP,d46169ff14ef99286a176b5391a7a1c8,1197.9,130.18,malas_acessorios,22600,3
7,80,60
2017/5/24 21:02,2017/6/2 10:48,8.573275463,2017/6/21 0:00,18.54983796,20555f7b23
72553063a07264ff6d3808,71727,brasilia,DF,d1b65fc7debc3361ea86b5f14c68d2e2,13844,
mogi
guacu,SP,aab5e2a4e6a2434fb61e694ed85a9888,129.9,15.66,papelaria,1800,32,40,16
2017/5/23 23:25,2017/5/28 2:48,4.141122685,2017/6/14
0:00,16.88275463,ff2cfbe44d7249b98eed0e860fe3e53c,2310,sao
paulo,SP,d1b65fc7debc3361ea86b5f14c68d2e2,13844,mogi guacu,SP,2020db9c389956e879
dd05e6250413d8,229.9,13.11,malas_acessorios,4000,38,52,22
```

[10]:
```
## we won't be sure of our above conclusions on the data unless we examine a␣
↪good sample of it. Let's do that for first 1% of the records
!head -n 1000 data.csv | csvstat
```

1. "order_purchase_timestamp"

```
Type of data:          DateTime
Contains null values:  False
Unique values:         895
Smallest value:        2016-10-08 10:55:00
Largest value:         2018-08-28 09:18:00
Most common values:    2018-07-12 18:57:00 (8x)
                       2018-08-15 14:02:00 (5x)
                       2018-01-06 23:02:00 (5x)
                       2017-05-27 19:20:00 (4x)
                       2018-05-07 21:54:00 (4x)
```

2. "order_delivered_customer_date"

```
Type of data:          DateTime
Contains null values:  True (excluded from calculations)
Unique values:         868
Smallest value:        2016-10-27 10:58:00
Largest value:         2018-09-19 15:46:00
Most common values:    None (29x)
                       2018-07-24 17:40:00 (8x)
                       2018-01-22 15:12:00 (5x)
                       2017-06-07 08:53:00 (4x)
                       2018-05-17 20:21:00 (4x)
```

3. "difference_delivered_purchase"

```
Type of data:          Number
Contains null values:  True (excluded from calculations)
Unique values:         871
Smallest value:        1.046
Largest value:         209.629
Sum:                   12,599.694
Mean:                  12.989
Median:                10.164
StDev:                 13.411
Most common values:    None (29x)
                       11.947 (8x)
                       15.674 (5x)
                       10.564 (4x)
                       9.935 (4x)
```

4. "order_estimated_delivery_date"

| | |
|---|---|
| Type of data: | DateTime |
| Contains null values: | False |
| Unique values: | 333 |
| Smallest value: | 2017-01-11 00:00:00 |
| Largest value: | 2018-10-10 00:00:00 |
| Most common values: | 2018-03-12 00:00:00 (12x) |
| | 2018-02-20 00:00:00 (11x) |
| | 2018-05-24 00:00:00 (11x) |
| | 2018-08-06 00:00:00 (10x) |
| | 2018-05-10 00:00:00 (10x) |

5. "difference_estimated_delivered"

| | |
|---|---|
| Type of data: | Number |
| Contains null values: | True (excluded from calculations) |
| Unique values: | 871 |
| Smallest value: | -181.609 |
| Largest value: | 75.543 |
| Sum: | 11,759.813 |
| Mean: | 12.124 |
| Median: | 12.428 |
| StDev: | 14.585 |
| Most common values: | None (29x) |
| | 12.263 (8x) |
| | 28.366 (5x) |
| | 21.63 (4x) |
| | 20.152 (4x) |

6. "customer_unique_id"

| | |
|---|---|
| Type of data: | Text |
| Contains null values: | False |
| Unique values: | 897 |
| Longest value: | 32 characters |
| Most common values: | c63a8c4fb13043a3fbe33bd17c69d17d (8x) |
| | 8d20b985a2670e363f0609d87d58a023 (5x) |
| | 9077449283dc1319d5e51fb3159b28e2 (4x) |
| | 93e765999b1ac5d5498d5cbdd316c7ca (4x) |
| | 71532174b89899cc2edc27886af89ec9 (4x) |

7. "customer_zip_code_prefix"

| | |
|---|---|
| Type of data: | Number |
| Contains null values: | False |
| Unique values: | 844 |
| Smallest value: | 1,033 |

```
   Largest value:         99,750
   Sum:                   36,877,584
   Mean:                  36,914.498
   Median:                29,023
   StDev:                 30,069.043
   Most common values:    61,624 (8x)
                          2,971 (5x)
                          11,030 (5x)
                          35,502 (4x)
                          29,230 (4x)
```

8. "customer_city"

```
   Type of data:          Text
   Contains null values:  False
   Unique values:         382
   Longest value:         24 characters
   Most common values:    sao paulo (161x)
                          rio de janeiro (62x)
                          brasilia (24x)
                          belo horizonte (23x)
                          campinas (19x)
```

9. "customer_state"

```
   Type of data:          Text
   Contains null values:  False
   Unique values:         24
   Longest value:         2 characters
   Most common values:    SP (395x)
                          MG (107x)
                          RJ (104x)
                          RS (53x)
                          PR (44x)
```

10. "seller_id"

```
   Type of data:          Text
   Contains null values:  False
   Unique values:         40
   Longest value:         32 characters
   Most common values:    ccc4bbb5f32a6ab2b7066a4130f114e3 (197x)
                          2ff97219cb8622eaf3cd89b7d9c09824 (97x)
                          1f9ab4708f3056ede07124aad39a2554 (82x)
                          116ccb1a1604bc88e4d234a8c23f33de (73x)
                          d9a84e1403de8da0c3aa531d6d108ba6 (67x)
```

11. "seller_zip_code_prefix"

```
   Type of data:          Number
   Contains null values:  False
   Unique values:         40
   Smallest value:        1,222
   Largest value:         98,910
   Sum:                   32,983,768
   Mean:                  33,016.785
   Median:                17,602
   StDev:                 30,802.385
   Most common values:    80,310 (197x)
                          13,320 (97x)
                          17,602 (82x)
                          9,850 (73x)
                          3,562 (67x)
```

12. "seller_city"

```
   Type of data:          Text
   Contains null values:  False
   Unique values:         28
   Longest value:         21 characters
   Most common values:    curitiba (203x)
                          sao paulo (202x)
                          salto (97x)
                          tupa (82x)
                          sao bernardo do campo (73x)
```

13. "seller_state"

```
   Type of data:          Text
   Contains null values:  False
   Unique values:         10
   Longest value:         2 characters
   Most common values:    SP (608x)
                          PR (205x)
                          MG (67x)
                          PE (42x)
                          DF (33x)
```

14. "product_id"

```
   Type of data:          Text
   Contains null values:  False
   Unique values:         384
   Longest value:         32 characters
   Most common values:    6cdd53843498f92890544667809f1595 (159x)
                          8aa6223e400af9c97b07c75993142721 (48x)
```

```
                          f1d4ce8c6dd66c47bbaa8c6781c2a923 (19x)
                          9fc063fd34fed29ccc57b7f8e8d03388 (17x)
                          3ce21e38e6a3060c20f4e74bdab770c8 (15x)
```

15. "price"

```
   Type of data:          Number
   Contains null values:  False
   Unique values:         247
   Smallest value:        5.9
   Largest value:         2,749
   Sum:                   156,850.6
   Mean:                  157.008
   Median:                87
   StDev:                 195.507
   Most common values:    349.9 (122x)
                          64.9 (40x)
                          169.9 (40x)
                          364 (32x)
                          174.9 (28x)
```

16. "freight_value"

```
   Type of data:          Number
   Contains null values:  False
   Unique values:         489
   Smallest value:        2.36
   Largest value:         157.23
   Sum:                   22,900.58
   Mean:                  22.924
   Median:                18.7
   StDev:                 15.903
   Most common values:    7.39 (40x)
                          18.23 (25x)
                          20.1 (15x)
                          15.1 (13x)
                          7.78 (13x)
```

17. "product_category_name"

```
   Type of data:          Text
   Contains null values:  True (excluded from calculations)
   Unique values:         41
   Longest value:         46 characters
   Most common values:    beleza_saude (224x)
                          bebes (130x)
                          brinquedos (112x)
                          automotivo (73x)
```

| | |
|---|---|
| | utilidades_domesticas (69x) |

18. "product_weight_g"

| | |
|---|---|
| Type of data: | Number |
| Contains null values: | False |
| Unique values: | 153 |
| Smallest value: | 50 |
| Largest value: | 26,400 |
| Sum: | 2,236,638 |
| Mean: | 2,238.877 |
| Median: | 900 |
| StDev: | 3,567.438 |
| Most common values: | 900 (171x) |
| | 200 (70x) |
| | 400 (57x) |
| | 5,000 (49x) |
| | 500 (39x) |

19. "product_length_cm"

| | |
|---|---|
| Type of data: | Number |
| Contains null values: | False |
| Unique values: | 46 |
| Smallest value: | 16 |
| Largest value: | 95 |
| Sum: | 29,068 |
| Mean: | 29.097 |
| Median: | 25 |
| StDev: | 12.369 |
| Most common values: | 25 (169x) |
| | 16 (167x) |
| | 40 (116x) |
| | 17 (81x) |
| | 45 (43x) |

20. "product_height_cm"

| | |
|---|---|
| Type of data: | Number |
| Contains null values: | False |
| Unique values: | 61 |
| Smallest value: | 2 |
| Largest value: | 88 |
| Sum: | 17,665 |
| Mean: | 17.683 |
| Median: | 12 |
| StDev: | 15.109 |
| Most common values: | 12 (235x) |

```
                                10 (137x)
                                2 (72x)
                                21 (49x)
                                4 (35x)
```

21. "product_width_cm"

```
        Type of data:           Number
        Contains null values:   False
        Unique values:          43
        Smallest value:         9
        Largest value:          92
        Sum:                    26,064
        Mean:                   26.09
        Median:                 24
        StDev:                  13.051
        Most common values:     38 (160x)
                                11 (124x)
                                40 (94x)
                                16 (72x)
                                15 (65x)
```

```
Row count: 999
```

Based on these values, We expect we can create transactional table as follows:

[11]:
```
%%sql
CREATE TABLE orders_facts (
    order_purchase_timestamp TIMESTAMP,
    order_delivered_customer_date TIMESTAMP,
    difference_delivered_purchase numeric,
    order_estimated_delivery_date TIMESTAMP,
    difference_estimated_delivered numeric,
    customer_unique_id varchar(100) Not Null,
    customer_zip_code_prefix numeric,
    customer_city varchar(100),
    customer_state varchar(100),
    seller_id varchar(100) Not Null,
    seller_zip_code_prefix numeric,
    seller_city varchar(100),
    seller_state varchar(100),
    product_id varchar(100) Not Null,
    price numeric,
    freight_value numeric,
    product_category_name varchar(100),
    product_weight_g numeric,
    product_lenght_cm numeric,
    product_height_cm numeric,
```

```
    product_width_cm numeric
);
```

```
 * postgresql://student@/final_project
Done.
```

[11]:
```
[]
```

Now we'll load the data directly using COPY command. Note that this requires the use of an absolute path, so adjust it to your location:

[12]:
```
!cp data.csv /tmp/data.csv
```

[13]:
```
%%sql
COPY orders_facts FROM '/tmp/data.csv'
CSV
HEADER;
```

```
 * postgresql://student@/final_project
117601 rows affected.
```

[13]:
```
[]
```

~12K records were loaded into the database. We can check the count using SQL.

[14]:
```
%%sql
SELECT COUNT(*) FROM orders_facts;
```

```
 * postgresql://student@/final_project
1 rows affected.
```

[14]:
```
[(117601,)]
```

[15]:
```
!wc -l data.csv
```

```
117602 data.csv
```

let's have a look at few loaded db records to make sure the data was loaded successfully.

[17]:
```
%%sql
SELECT * FROM orders_facts
LIMIT 10
```

```
 * postgresql://student@/final_project
10 rows affected.
```

[17]:
```
[(datetime.datetime(2017, 5, 5, 16, 12), datetime.datetime(2017, 6, 2, 16, 57),
Decimal('28.03142361'), datetime.datetime(2017, 5, 30, 0, 0),
Decimal('-3.706759259'), 'b4527423469300ee354458e1b5f961be', Decimal('32223'),
```

```
'contagem', 'MG', '3442f8959a84dea7ee197c632cb2df15', Decimal('13023'),
'campinas', 'SP', 'f4621f8ad6f54a2e3c408884068be46d', Decimal('101.7'),
Decimal('15.92'), 'esporte_lazer', Decimal('600'), Decimal('35'), Decimal('15'),
Decimal('28')),
 (datetime.datetime(2017, 8, 30, 11, 47), datetime.datetime(2017, 9, 1, 16, 51),
Decimal('2.210810185'), datetime.datetime(2017, 9, 20, 0, 0),
Decimal('18.29761574'), 'af0f26435fade1ca984d9affda307199', Decimal('9310'),
'maua', 'SP', '3442f8959a84dea7ee197c632cb2df15', Decimal('13023'), 'campinas',
'SP', '325a06bcce0da45b7f4ecf2797dd40e4', Decimal('10.8'), Decimal('2.42'),
'esporte_lazer', Decimal('300'), Decimal('16'), Decimal('5'), Decimal('15')),
 (datetime.datetime(2017, 8, 21, 20, 35), datetime.datetime(2017, 8, 30, 16, 7),
Decimal('8.813530093'), datetime.datetime(2017, 9, 1, 0, 0),
Decimal('1.328321759'), 'f421a2a66b69dbfe6db0c87845281a90', Decimal('4661'),
'sao paulo', 'SP', '3442f8959a84dea7ee197c632cb2df15', Decimal('13023'),
'campinas', 'SP', 'ffb64e34a37740dafb6c88f1abd1fa61', Decimal('106.2'),
Decimal('9.56'), 'esporte_lazer', Decimal('700'), Decimal('43'), Decimal('15'),
Decimal('35')),
 (datetime.datetime(2017, 4, 28, 14, 20), datetime.datetime(2017, 5, 9, 14, 27),
Decimal('11.00483796'), datetime.datetime(2017, 6, 1, 0, 0),
Decimal('22.3978588'), '00ac9cd5c4ad19e16e7c6f6864711737', Decimal('37500'),
'itajuba', 'MG', 'd1b65fc7debc3361ea86b5f14c68d2e2', Decimal('13844'), 'mogi
guacu', 'SP', '765c417cdc38443aaa558a0159a98591', Decimal('209.9'),
Decimal('21.55'), 'malas_acessorios', Decimal('3500'), Decimal('40'),
Decimal('55'), Decimal('25')),
 (datetime.datetime(2017, 4, 27, 9, 9), datetime.datetime(2017, 5, 4, 13, 20),
Decimal('7.173877315'), datetime.datetime(2017, 6, 6, 0, 0),
Decimal('32.44436343'), '51dc56123336c573f2977f5da81b17b9', Decimal('20251'),
'rio de janeiro', 'RJ', 'd1b65fc7debc3361ea86b5f14c68d2e2', Decimal('13844'),
'mogi guacu', 'SP', '765c417cdc38443aaa558a0159a98591', Decimal('209.9'),
Decimal('21.55'), 'malas_acessorios', Decimal('3500'), Decimal('40'),
Decimal('55'), Decimal('25')),
 (datetime.datetime(2018, 4, 9, 23, 40), datetime.datetime(2018, 4, 30, 18, 41),
Decimal('20.79236111'), datetime.datetime(2018, 4, 30, 0, 0),
Decimal('-0.778541667'), '177e10134f99776d8a2b0c10c3fed38c', Decimal('37190'),
'tres pontas', 'MG', 'd1b65fc7debc3361ea86b5f14c68d2e2', Decimal('13844'), 'mogi
guacu', 'SP', 'cb378611bbb39f1716b4f0c335201448', Decimal('399.99'),
Decimal('36.34'), 'malas_acessorios', Decimal('16850'), Decimal('38'),
Decimal('58'), Decimal('25')),
 (datetime.datetime(2017, 5, 7, 12, 42), datetime.datetime(2017, 5, 19, 10, 7),
Decimal('11.8921875'), datetime.datetime(2017, 6, 1, 0, 0),
Decimal('12.57799769'), '28b9099dc6577fdeceb2e4468e69f556', Decimal('88440'),
'imbuia', 'SC', 'd1b65fc7debc3361ea86b5f14c68d2e2', Decimal('13844'), 'mogi
guacu', 'SP', 'd46169ff14ef99286a176b5391a7a1c8', Decimal('1197.9'),
Decimal('130.18'), 'malas_acessorios', Decimal('22600'), Decimal('37'),
Decimal('80'), Decimal('60')),
 (datetime.datetime(2017, 5, 24, 21, 2), datetime.datetime(2017, 6, 2, 10, 48),
Decimal('8.573275463'), datetime.datetime(2017, 6, 21, 0, 0),
```

```
Decimal('18.54983796'), '20555f7b2372553063a07264ff6d3808', Decimal('71727'),
'brasilia', 'DF', 'd1b65fc7debc3361ea86b5f14c68d2e2', Decimal('13844'), 'mogi
guacu', 'SP', 'aab5e2a4e6a2434fb61e694ed85a9888', Decimal('129.9'),
Decimal('15.66'), 'papelaria', Decimal('1800'), Decimal('32'), Decimal('40'),
Decimal('16')),
 (datetime.datetime(2017, 5, 23, 23, 25), datetime.datetime(2017, 5, 28, 2, 48),
Decimal('4.141122685'), datetime.datetime(2017, 6, 14, 0, 0),
Decimal('16.88275463'), 'ff2cfbe44d7249b98eed0e860fe3e53c', Decimal('2310'),
'sao paulo', 'SP', 'd1b65fc7debc3361ea86b5f14c68d2e2', Decimal('13844'), 'mogi
guacu', 'SP', '2020db9c389956e879dd05e6250413d8', Decimal('229.9'),
Decimal('13.11'), 'malas_acessorios', Decimal('4000'), Decimal('38'),
Decimal('52'), Decimal('22')),
 (datetime.datetime(2017, 5, 23, 23, 25), datetime.datetime(2017, 5, 28, 2, 48),
Decimal('4.141122685'), datetime.datetime(2017, 6, 14, 0, 0),
Decimal('16.88275463'), 'ff2cfbe44d7249b98eed0e860fe3e53c', Decimal('2310'),
'sao paulo', 'SP', 'd1b65fc7debc3361ea86b5f14c68d2e2', Decimal('13844'), 'mogi
guacu', 'SP', 'd3d5a1d52abe9a7d234908d873fc377b', Decimal('229.9'),
Decimal('13.11'), 'malas_acessorios', Decimal('4900'), Decimal('38'),
Decimal('55'), Decimal('22'))]
```

So far so good. Next we need to look at the current transactional design and build the dimensional equivalent to the design

### 2.3 More ETL with SQL

Today we started with this schema:

[6]:
```
from IPython.display import Image
```

[7]:
```
Image("image1.png")
```

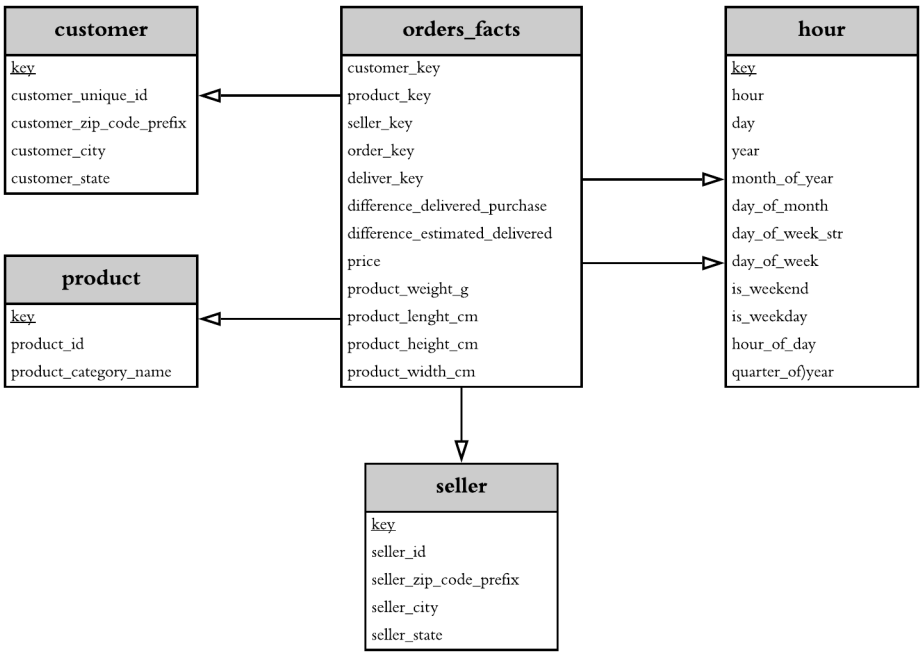
[7]:

| orders_facts |
| --- |
| order_purchase_timestamp |
| order_delivered_customer_date |
| difference_delivered_purchase |
| order_estimated_delivery_date |
| difference_estimated_delivered |
| customer_unique_id |
| customer_zip_code_prefix |
| customer_city |
| customer_state |
| seller_id |
| seller_zip_code_prefix |
| seller_city |
| seller_state |
| product_id |
| price |
| freight_value |
| product_category_name |
| product_weight_g |
| product_lenght_cm |
| product_height_cm |
| product_width_cm |

our aim is to build a star schema that allow us to do analytical analysis and business intellegence on the data. I have created a star schema for discussion. Let's go over the design.

[8]:
```
Image("image2.png")
```

[8]:




Now we can create a new dimension table to house the unique station ids and names. For key we use the data type serial. Check Postgresql documentation @ https://www.postgresql.org/docs/9.5/datatype-numeric.html#DATATYPE-SERIAL serial is the equivalent of creating a sequence that its value is equal to the largest number in the table + 1.

Customer Dimension

[18]:
```
%%sql
DROP TABLE IF EXISTS customer;

CREATE TABLE customer (
    key SERIAL PRIMARY KEY,
    customer_unique_id varchar(100) Not Null,
    customer_zip_code_prefix numeric,
    customer_city varchar(100),
    customer_state varchar(100)
);
```

```
 * postgresql://student@/final_project
Done.
Done.
```

[18]: []

Upload customer information using the union query. serial will automatically insert a serial integer into the table

[19]:
```
%%sql
INSERT INTO customer (customer_unique_id, customer_zip_code_prefix,␣
 ↪customer_city, customer_state)
SELECT DISTINCT customer_unique_id AS customer_unique_id,
    customer_zip_code_prefix AS customer_zip_code_prefix,
    customer_city AS customer_city,
    customer_state AS customer_state
FROM orders_facts
UNION
SELECT DISTINCT customer_unique_id AS customer_unique_id,
    customer_zip_code_prefix AS customer_zip_code_prefix,
    customer_city AS customer_city,
    customer_state AS customer_state
FROM orders_facts;
```

```
 * postgresql://student@/final_project
95670 rows affected.
```

[19]:
```
[]
```

[20]:
```
%%sql
SELECT * FROM customer
LIMIT 10;
```

```
 * postgresql://student@/final_project
10 rows affected.
```

[20]:
```
[(1, '233bced31686284a380870eb2761109f', Decimal('18480'), 'itaporanga', 'SP'),
 (2, 'fe25d225d2494b252321b09821ebdbdf', Decimal('27420'), 'quatis', 'RJ'),
 (3, '92b18241cb0ae7c3be4c5c9e4b9db371', Decimal('22743'), 'rio de janeiro',
'RJ'),
 (4, 'edc699eb8c0fc6ec7a6ec536382840b3', Decimal('6714'), 'cotia', 'SP'),
 (5, '41415f9d98446c12f76c9c4d5af62b03', Decimal('6730'), 'vargem grande
paulista', 'SP'),
 (6, 'bb182f55e4cff6eabfc26d08354b5995', Decimal('95900'), 'lajeado', 'RS'),
 (7, '37b708a378a17996174d1b8f6ed3d3e9', Decimal('20720'), 'rio de janeiro',
'RJ'),
 (8, '9aba54994850ca5befd6f16384e925b2', Decimal('60140'), 'fortaleza', 'CE'),
 (9, 'ead79bd27a251fe8a2070b1304f10044', Decimal('37270'), 'campo belo', 'MG'),
 (10, '59cd47f7d0957b3cf7860f25986cf4b5', Decimal('21760'), 'rio de janeiro',
'RJ')]
```

We now add FK customer_key to the fact table. repeat of step 1-3 as describes below

[21]:
```
%%sql
-- Step 1
ALTER TABLE orders_facts
ADD COLUMN customer_key INTEGER,
-- Step 2
ADD CONSTRAINT fk_customer
    FOREIGN KEY (customer_key)
    REFERENCES customer (key);
```

```
 * postgresql://student@/final_project
Done.
```

[21]:
```
[]
```

Now we update the customer_key in the fact table with the values from customer dimension table based on unique customers.

[22]:
```
%%sql
-- Step 3
UPDATE orders_facts AS f
SET customer_key = c.key
FROM customer AS c
WHERE f.customer_unique_id = c.customer_unique_id;
```

```
 * postgresql://student@/final_project
117601 rows affected.
```

[22]:
```
[]
```

Product Dimension

[23]:
```
%%sql
DROP TABLE IF EXISTS product;

CREATE TABLE product (
    key SERIAL PRIMARY KEY,
    product_id varchar(100) Not Null,
    product_category_name varchar(100)
);
```

```
 * postgresql://student@/final_project
Done.
Done.
```

[23]:
```
[]
```

[24]:
```
%%sql
INSERT INTO product (product_id, product_category_name)
```

```
SELECT DISTINCT product_id AS product_id,
    product_category_name AS product_category_name
FROM orders_facts
UNION
SELECT DISTINCT product_id AS product_id,
    product_category_name AS product_category_name
FROM orders_facts;
```

```
 * postgresql://student@/final_project
32951 rows affected.
```

[24]:
```
[]
```

[25]:
```
%%sql
SELECT * FROM product
LIMIT 5;
```

```
 * postgresql://student@/final_project
5 rows affected.
```

[25]:
```
[(1, '59542ce967e2cbc86b0cfd48baf77d96', 'informatica_acessorios'),
 (2, '0a2fff0d95ef3bbb7dffc618f9542ba9', 'relogios_presentes'),
 (3, '87f87c717c93f801e1a62367ce5ff63f', 'construcao_ferramentas_iluminacao'),
 (4, 'c03538c4936ed498a78ab92db5105f46', 'utilidades_domesticas'),
 (5, '112dad9870ca76046f327b9c08f00b29', 'brinquedos')]
```

Load order_reviews.csv into relational database

[26]:
```
%%sql
-- Step 1
ALTER TABLE orders_facts
ADD COLUMN product_key INTEGER,
-- Step 2
ADD CONSTRAINT fk_product
    FOREIGN KEY (product_key)
    REFERENCES product (key);
```

```
 * postgresql://student@/final_project
Done.
```

[26]:
```
[]
```

[27]:
```
%%sql
-- Step 3
UPDATE orders_facts AS f
SET product_key = p.key
FROM product AS p
WHERE f.product_id = p.product_id;
```

```
 * postgresql://student@/final_project
117601 rows affected.
```

[27]:
```
[]
```

Seller Dimension

[28]:
```
%%sql
DROP TABLE IF EXISTS seller;

CREATE TABLE seller (
    key SERIAL PRIMARY KEY,
    seller_id varchar(100) Not Null,
    seller_zip_code_prefix numeric,
    seller_city varchar(100),
    seller_state varchar(100)
);
```

```
 * postgresql://student@/final_project
Done.
Done.
```

[28]:
```
[]
```

[29]:
```
%%sql
INSERT INTO seller (seller_id, seller_zip_code_prefix, seller_city,␣
 ↪seller_state)
SELECT DISTINCT seller_id AS seller_id,
    seller_zip_code_prefix AS seller_zip_code_prefix,
    seller_city AS seller_city,
    seller_state AS seller_state
FROM orders_facts
UNION
SELECT DISTINCT seller_id AS seller_id,
    seller_zip_code_prefix AS seller_zip_code_prefix,
    seller_city AS seller_city,
    seller_state AS seller_state
FROM orders_facts;
```

```
 * postgresql://student@/final_project
3095 rows affected.
```

[29]:
```
[]
```

[30]:
```
%%sql
select *
from seller
limit 5
```

```
 * postgresql://student@/final_project
5 rows affected.
```

[30]:
```
[(1, '0015a82c2db000af6aaaf3ae2ecb0532', Decimal('9080'), 'santo andre', 'SP'),
 (2, '001cca7ae9ae17fb1caed9dfb1094831', Decimal('29156'), 'cariacica', 'ES'),
 (3, '001e6ad469a905060d959994f1b41e4f', Decimal('24754'), 'sao goncalo', 'RJ'),
 (4, '002100f778ceb8431b7a1020ff7ab48f', Decimal('14405'), 'franca', 'SP'),
 (5, '003554e2dce176b5555353e4f3555ac8', Decimal('74565'), 'goiania', 'GO')]
```

[31]:
```
%%sql
-- Step 1
ALTER TABLE orders_facts
ADD COLUMN seller_key INTEGER,
-- Step 2
ADD CONSTRAINT fk_seller
    FOREIGN KEY (seller_key)
    REFERENCES seller (key);
```

```
 * postgresql://student@/final_project
Done.
```

[31]:
```
[]
```

[32]:
```
%%sql
-- Step 3
UPDATE orders_facts AS f
SET seller_key = s.key
FROM seller AS s
WHERE f.seller_id = s.seller_id;
```

```
 * postgresql://student@/final_project
117601 rows affected.
```

[32]:
```
[]
```

### 2.3.1 Creating the hour dimension table

**Link hour dimension table to the orders_facts table**

[33]:
```
%%sql
SELECT DISTINCT TO_CHAR(order_purchase_timestamp, 'YYYY-MM-DD HH24:00:00') AS␣
 ↪hour,
    TO_CHAR(order_purchase_timestamp, 'YYYY-MM-DD') AS day,
    TO_CHAR(order_purchase_timestamp, 'YYYY') AS year,
    TO_CHAR(order_purchase_timestamp, 'Month') AS month_of_year_str,
    TO_CHAR(order_purchase_timestamp, 'MM') AS month_of_year,
    TO_CHAR(order_purchase_timestamp, 'DD') AS day_of_month,
    TO_CHAR(order_purchase_timestamp, 'Day') AS day_of_week_str,
```

```
    TO_CHAR(order_purchase_timestamp, 'D') AS day_of_week,
    CASE WHEN CAST(TO_CHAR(order_purchase_timestamp, 'D') AS INTEGER) >= 6
        THEN 'true'
        ELSE 'false'
    END AS is_weekend,
    CASE WHEN CAST(TO_CHAR(order_purchase_timestamp, 'D') AS INTEGER) < 6
        THEN 'true'
        ELSE 'false'
    END AS is_weekday,
    TO_CHAR(order_purchase_timestamp, 'HH24') AS hour_of_day,
    TO_CHAR(order_purchase_timestamp, 'Q') AS quarter_of_year
FROM orders_facts
LIMIT 10;
```

```
 * postgresql://student@/final_project
10 rows affected.
```

[33]:
```
[('2016-09-04 21:00:00', '2016-09-04', '2016', 'September', '09', '04', 'Sunday
', '1', 'false', 'true', '21', '3'),
 ('2016-09-05 00:00:00', '2016-09-05', '2016', 'September', '09', '05', 'Monday
', '2', 'false', 'true', '00', '3'),
 ('2016-10-02 22:00:00', '2016-10-02', '2016', 'October  ', '10', '02', 'Sunday
', '1', 'false', 'true', '22', '4'),
 ('2016-10-03 09:00:00', '2016-10-03', '2016', 'October  ', '10', '03', 'Monday
', '2', 'false', 'true', '09', '4'),
 ('2016-10-03 16:00:00', '2016-10-03', '2016', 'October  ', '10', '03', 'Monday
', '2', 'false', 'true', '16', '4'),
 ('2016-10-03 21:00:00', '2016-10-03', '2016', 'October  ', '10', '03', 'Monday
', '2', 'false', 'true', '21', '4'),
 ('2016-10-03 22:00:00', '2016-10-03', '2016', 'October  ', '10', '03', 'Monday
', '2', 'false', 'true', '22', '4'),
 ('2016-10-04 09:00:00', '2016-10-04', '2016', 'October  ', '10', '04', 'Tuesday
', '3', 'false', 'true', '09', '4'),
 ('2016-10-04 10:00:00', '2016-10-04', '2016', 'October  ', '10', '04', 'Tuesday
', '3', 'false', 'true', '10', '4'),
 ('2016-10-04 11:00:00', '2016-10-04', '2016', 'October  ', '10', '04', 'Tuesday
', '3', 'false', 'true', '11', '4')]
```

creating dimension table Hour

[34]:
```
%%sql
DROP TABLE IF EXISTS hour;

CREATE TABLE hour (
    key SERIAL PRIMARY KEY,
    hour CHAR(19),
    day CHAR(10),
```

```
    year INTEGER,
    month_of_year_str VARCHAR(12),
    month_of_year INTEGER,
    day_of_month INTEGER,
    day_of_week_str CHAR(9),
    day_of_week INTEGER,
    is_weekend BOOLEAN,
    is_weekday BOOLEAN,
    hour_of_day INTEGER,
    quarter_of_year INTEGER
);
```

```
 * postgresql://student@/final_project
Done.
Done.
```

[34]:
```
[]
```

Populating dimension table hour with data from orders_facts (union of order_purchase_timestamp and order_delivered_customer_date)

[35]:
```
%%sql
INSERT INTO hour (hour, day, year, month_of_year_str, month_of_year,␣
 ↪day_of_month,
                  day_of_week_str, day_of_week, is_weekend, is_weekday,
                  hour_of_day, quarter_of_year)
SELECT DISTINCT TO_CHAR(order_purchase_timestamp, 'YYYY-MM-DD HH24:00:00') AS␣
 ↪hour,
    TO_CHAR(order_purchase_timestamp, 'YYYY-MM-DD') AS day,
    CAST(TO_CHAR(order_purchase_timestamp, 'YYYY') AS INTEGER) AS year,
    TO_CHAR(order_purchase_timestamp, 'Month') AS month_of_year_str,
    CAST(TO_CHAR(order_purchase_timestamp, 'MM') AS INTEGER) AS month_of_year,
    CAST(TO_CHAR(order_purchase_timestamp, 'DD') AS INTEGER) AS day_of_month,
    TO_CHAR(order_purchase_timestamp, 'Day') AS day_of_week_str,
    CAST(TO_CHAR(order_purchase_timestamp, 'D') AS INTEGER) AS day_of_week,
    CASE WHEN CAST(TO_CHAR(order_purchase_timestamp, 'D') AS INTEGER) IN (1, 7)
        THEN TRUE
        ELSE FALSE
    END AS is_weekend,
    CASE WHEN CAST(TO_CHAR(order_purchase_timestamp, 'D') AS INTEGER) NOT IN␣
 ↪(1, 7)
        THEN TRUE
        ELSE FALSE
    END AS is_weekday,
    CAST(TO_CHAR(order_purchase_timestamp, 'HH24') AS INTEGER) AS hour_of_day,
    CAST(TO_CHAR(order_purchase_timestamp, 'Q') AS INTEGER) AS quarter_of_year
FROM orders_facts
```

```
UNION
SELECT DISTINCT TO_CHAR(order_delivered_customer_date, 'YYYY-MM-DD HH24:00:00')␣
↪AS hour,
   TO_CHAR(order_delivered_customer_date, 'YYYY-MM-DD') AS day,
   CAST(TO_CHAR(order_delivered_customer_date, 'YYYY') AS INTEGER) AS year,
   TO_CHAR(order_delivered_customer_date, 'Month') AS month_of_year_str,
   CAST(TO_CHAR(order_delivered_customer_date, 'MM') AS INTEGER) AS␣
↪month_of_year,
   CAST(TO_CHAR(order_delivered_customer_date, 'DD') AS INTEGER) AS␣
↪day_of_month,
   TO_CHAR(order_delivered_customer_date, 'Day') AS day_of_week_str,
   CAST(TO_CHAR(order_delivered_customer_date, 'D') AS INTEGER) AS day_of_week,
   CASE WHEN CAST(TO_CHAR(order_delivered_customer_date, 'D') AS INTEGER) IN␣
↪(1, 7)
        THEN TRUE
        ELSE FALSE
   END AS is_weekend,
   CASE WHEN CAST(TO_CHAR(order_delivered_customer_date, 'D') AS INTEGER) NOT␣
↪IN (1, 7)
        THEN TRUE
        ELSE FALSE
   END AS is_weekday,
   CAST(TO_CHAR(order_delivered_customer_date, 'HH24') AS INTEGER) AS␣
↪hour_of_day,
   CAST(TO_CHAR(order_delivered_customer_date, 'Q') AS INTEGER) AS␣
↪quarter_of_year
FROM orders_facts;
```

```
 * postgresql://student@/final_project
12923 rows affected.
```

[35]:
```
[]
```

[36]:
```
%%sql
SELECT * FROM hour
LIMIT 10;
```

```
 * postgresql://student@/final_project
10 rows affected.
```

[36]:
```
[(1, '2016-09-04 21:00:00', '2016-09-04', 2016, 'September', 9, 4, 'Sunday   ',
1, True, False, 21, 3),
 (2, '2016-09-05 00:00:00', '2016-09-05', 2016, 'September', 9, 5, 'Monday   ',
2, False, True, 0, 3),
 (3, '2016-10-02 22:00:00', '2016-10-02', 2016, 'October  ', 10, 2, 'Sunday   ',
1, True, False, 22, 4),
 (4, '2016-10-03 09:00:00', '2016-10-03', 2016, 'October  ', 10, 3, 'Monday   ',
```

```
2, False, True, 9, 4),
 (5, '2016-10-03 16:00:00', '2016-10-03', 2016, 'October  ', 10, 3, 'Monday   ',
2, False, True, 16, 4),
 (6, '2016-10-03 21:00:00', '2016-10-03', 2016, 'October  ', 10, 3, 'Monday   ',
2, False, True, 21, 4),
 (7, '2016-10-03 22:00:00', '2016-10-03', 2016, 'October  ', 10, 3, 'Monday   ',
2, False, True, 22, 4),
 (8, '2016-10-04 09:00:00', '2016-10-04', 2016, 'October  ', 10, 4, 'Tuesday  ',
3, False, True, 9, 4),
 (9, '2016-10-04 10:00:00', '2016-10-04', 2016, 'October  ', 10, 4, 'Tuesday  ',
3, False, True, 10, 4),
 (10, '2016-10-04 11:00:00', '2016-10-04', 2016, 'October  ', 10, 4, 'Tuesday
', 3, False, True, 11, 4)]
```

We now add FK start_hour_key to the fact table. repeat of step 1-3 as describes above

[37]:
```
%%sql
-- Step 1
ALTER TABLE orders_facts
ADD COLUMN order_key INTEGER,
-- Step 2
ADD CONSTRAINT fk_order_hour
    FOREIGN KEY (order_key)
    REFERENCES hour (key);
```

```
 * postgresql://student@/final_project
Done.
```

[37]:
```
[]
```

Now we update the start_hour_key in the fact table with the values from hour dimension table based on start time of the trips.

[38]:
```
%%sql
-- Step 3
UPDATE orders_facts
SET order_key = hour.key
FROM hour
WHERE TO_CHAR(orders_facts.order_purchase_timestamp, 'YYYY-MM-DD HH24:00:00') =␣
 ↪hour.hour;
```

```
 * postgresql://student@/final_project
117601 rows affected.
```

[38]:
```
[]
```

Do the same for the deliver_key

[39]:
```
%%sql
-- Step 1
ALTER TABLE orders_facts
ADD COLUMN deliver_key INTEGER,
-- Step 2
ADD CONSTRAINT fk_deliver_hour
    FOREIGN KEY (deliver_key)
    REFERENCES hour (key);
```

```
 * postgresql://student@/final_project
Done.
```

[39]:
```
[]
```

[40]:
```
%%sql
-- step 3
UPDATE orders_facts
SET deliver_key = hour.key
FROM hour
WHERE TO_CHAR(orders_facts.order_delivered_customer_date, 'YYYY-MM-DD HH24:00:
 ↪00') = hour.hour;
```

```
 * postgresql://student@/final_project
115034 rows affected.
```

[40]:
```
[]
```

We can drop columns from orders_facts table

[41]:
```
%%sql
ALTER TABLE orders_facts
DROP COLUMN order_purchase_timestamp,
DROP COLUMN order_delivered_customer_date,
DROP COLUMN order_estimated_delivery_date,
DROP COLUMN customer_unique_id,
DROP COLUMN customer_zip_code_prefix,
DROP COLUMN customer_city,
DROP COLUMN customer_state,
DROP COLUMN seller_id,
DROP COLUMN seller_zip_code_prefix,
DROP COLUMN seller_city,
DROP COLUMN seller_state,
DROP COLUMN product_id,
DROP COLUMN freight_value,
DROP COLUMN product_category_name;
```

```
 * postgresql://student@/final_project
Done.
```

[41]: []

[42]:
```
%%sql
SELECT * FROM orders_facts
LIMIT 10;
```

```
 * postgresql://student@/final_project
10 rows affected.
```

[42]:
```
[(None, None, Decimal('32.9'), Decimal('1800'), Decimal('32'), Decimal('6'),
Decimal('28'), 68786, 31248, 247, 1, None),
 (None, None, Decimal('39.99'), Decimal('1400'), Decimal('32'), Decimal('6'),
Decimal('28'), 68786, 24855, 247, 1, None),
 (None, None, Decimal('59.5'), Decimal('700'), Decimal('25'), Decimal('2'),
Decimal('25'), 27114, 31388, 1999, 2, None),
 (None, None, Decimal('100'), Decimal('500'), Decimal('18'), Decimal('18'),
Decimal('18'), 5492, 27139, 446, 3, None),
 (None, None, Decimal('164'), Decimal('900'), Decimal('28'), Decimal('18'),
Decimal('14'), 16038, 32143, 1681, 12, None),
 (None, None, Decimal('189.9'), Decimal('400'), Decimal('19'), Decimal('7'),
Decimal('17'), 80715, 27727, 890, 12, None),
 (None, None, Decimal('239.9'), Decimal('400'), Decimal('18'), Decimal('16'),
Decimal('16'), 73027, 23180, 2203, 13, None),
 (None, None, Decimal('22.37'), Decimal('900'), Decimal('16'), Decimal('34'),
Decimal('34'), 17411, 22225, 288, 14, None),
 (None, None, Decimal('599'), Decimal('20100'), Decimal('90'), Decimal('60'),
Decimal('20'), 26169, 7464, 808, 15, None),
 (None, None, Decimal('129.9'), Decimal('700'), Decimal('31'), Decimal('5'),
Decimal('25'), 20610, 2259, 1170, 19, None)]
```

Now we are ready to explore the data.

## 3 Ask 3

**Business Question BQ1** Find the number of orders by day of the week.

[42]:
```
%%sql
SELECT day_of_week, day_of_week_str, COUNT(*) count
FROM orders_facts
JOIN hour
  ON orders_facts.order_key = hour.key
GROUP BY day_of_week_str, day_of_week
ORDER BY day_of_week;
```

```
 * postgresql://student@/final_project
7 rows affected.
```

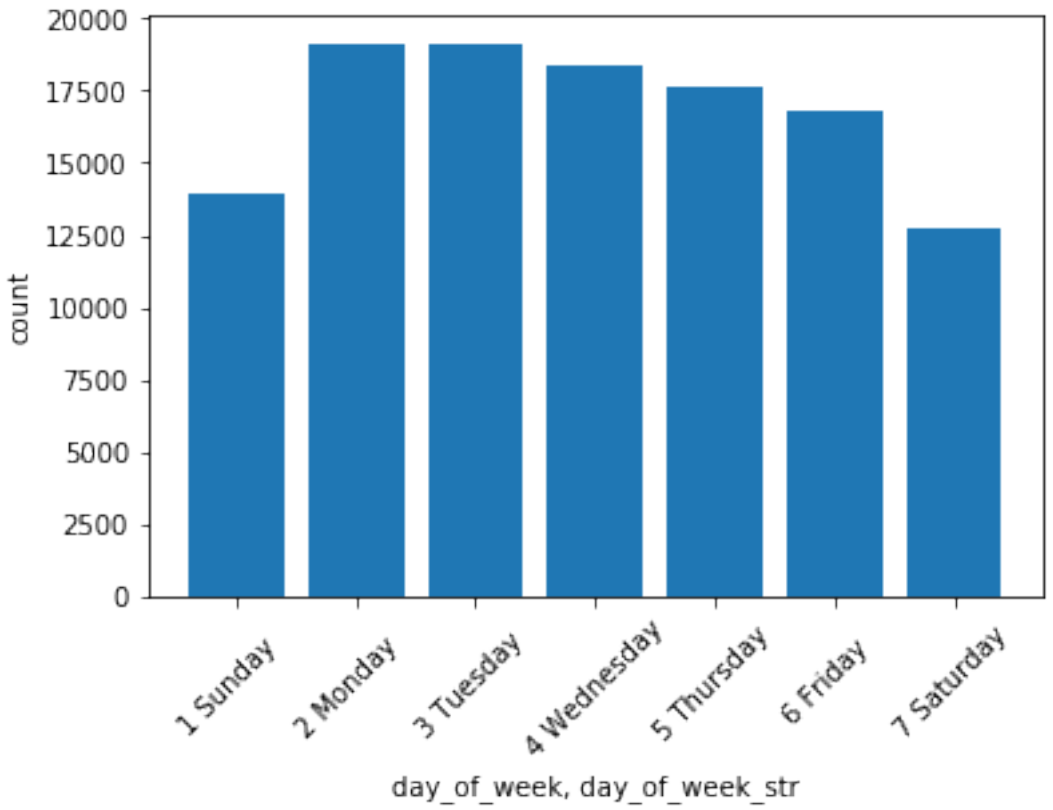
[42]:
```
[(1, 'Sunday   ', 13936),
 (2, 'Monday   ', 19130),
 (3, 'Tuesday  ', 19077),
 (4, 'Wednesday', 18380),
 (5, 'Thursday ', 17590),
 (6, 'Friday   ', 16760),
 (7, 'Saturday ', 12728)]
```

Interpretation: customers tend to place more orders on Monday and Tuesday. Therefore, sellers could invest more on Monday and Tuesday' marketing.

[43]:
```
%matplotlib inline
```

[44]:
```
_.bar()
```

[44]:
```
<BarContainer object of 7 artists>
```




**BQ2:** Find the top 10 product categories based on average price

[48]:
```
%%sql
SELECT round(AVG(price),2) as average_price, p.product_category_name
FROM orders_facts
JOIN product AS p
  ON orders_facts.product_key = p.key
GROUP BY p.product_category_name
ORDER BY average_price DESC
LIMIT 10;
```

```
 * postgresql://student@/final_project
10 rows affected.
```

[48]:
```
[(Decimal('1103.69'), 'pcs'),
 (Decimal('627.51'), 'portateis_casa_forno_e_cafe'),
 (Decimal('459.95'), 'eletrodomesticos_2'),
 (Decimal('332.71'), 'agro_industria_e_comercio'),
 (Decimal('293.77'), 'instrumentos_musicais'),
 (Decimal('286.61'), 'eletroportateis'),
 (Decimal('264.57'), 'portateis_cozinha_e_preparadores_de_alimentos'),
 (Decimal('227.68'), 'telefonia_fixa'),
 (Decimal('210.94'), 'construcao_ferramentas_seguranca'),
 (Decimal('202.09'), 'relogios_presentes')]
```

Interpretation: Olist and sellers could generate more revenue by increasing sales in computers, small appliances, home appliances, and musical instruments.

**BQ3:** Find the average days of product delays of each state based on the difference between estimatated delivery days and actural delivery days.

[6]:
```
%%sql
SELECT round(AVG(f.difference_estimated_delivered),2) as average_delay, c.
 ↪customer_state
FROM orders_facts AS f
JOIN customer AS c
  ON f.customer_key = c.key
GROUP BY c.customer_state
HAVING round(AVG(f.difference_estimated_delivered),2) IS NOT NULL
ORDER BY average_delay DESC;
```

```
 * postgresql://student@/final_project
27 rows affected.
```

[6]:
```
[(Decimal('20.54'), 'AC'),
 (Decimal('19.26'), 'RO'),
 (Decimal('19.15'), 'AM'),
 (Decimal('18.03'), 'AP'),
 (Decimal('17.62'), 'RR'),
```

```
 (Decimal('13.69'), 'MT'),
 (Decimal('13.60'), 'PA'),
 (Decimal('13.49'), 'RS'),
 (Decimal('13.00'), 'RN'),
 (Decimal('12.82'), 'PE'),
 (Decimal('12.78'), 'PR'),
 (Decimal('12.65'), 'MG'),
 (Decimal('12.34'), 'PB'),
 (Decimal('11.86'), 'TO'),
 (Decimal('11.59'), 'GO'),
 (Decimal('11.50'), 'DF'),
 (Decimal('11.33'), 'RJ'),
 (Decimal('10.91'), 'SC'),
 (Decimal('10.80'), 'PI'),
 (Decimal('10.59'), 'CE'),
 (Decimal('10.56'), 'MS'),
 (Decimal('10.54'), 'SP'),
 (Decimal('10.29'), 'BA'),
 (Decimal('9.89'), 'ES'),
 (Decimal('9.38'), 'SE'),
 (Decimal('9.19'), 'MA'),
 (Decimal('7.86'), 'AL')]
```

Interpretation: Customers in AC, RO, and AM have the worst delivery experience. Olist and sellers should come up with better delivery solutions to increase customer experience.