# brazilian_ecommerce

December 6, 2020

# 1 DS4A / COLOMBIA 4.0

# 2 Topic: Consumer behavior on e-Commerce

Data science for the digital future


- Ximena Rios Cotazo
- Ramon Manuel Sandoval
- Luisa Maria Carabali
- Alejandro Camargo Garcia
- Cristian Sarmiento
- Hector Melo


## 2.1 E-Commerce

This is a Brazilian ecommerce public dataset of orders made at Olist Store. The dataset has information of 100k orders from 2016 to 2018 made at multiple marketplaces in Brazil. Its features allows viewing an order from multiple dimensions: from order status, price, payment and freight performance to customer location, product attributes and finally reviews written by customers. We also released a geolocation dataset that relates Brazilian zip codes to lat/lng coordinates.

This is real commercial data, it has been anonymised, and references to the companies and partners in the review text have been replaced with the names of Game of Thrones great houses.

## 2.2 Context

This dataset was generously provided by Olist, the largest department store in Brazilian marketplaces. Olist connects small businesses from all over Brazil to channels without hassle and with a single contract. Those merchants are able to sell their products through the Olist Store and ship them directly to the customers using Olist logistics partners. See more on our website: www.olist.com

After a customer purchases the product from Olist Store a seller gets notified to fulfill that order. Once the customer receives the product, or the estimated delivery date is due, the customer gets a satisfaction survey by email where he can give a note for the purchase experience and write down some comments.

Taken from:Kaggle

### 2.3 Topic: Consumer behavior on E-Commerce

### 2.4 Research question

What aspects of consumers' online behavior is useful for businesses to better understand their customers and predict consumer trends, spending habits, variables related to the shopping?

### 2.5 Objectives:

- Data understanding and cleaning
- Analysis and modeling
- Build prediction models
- Build dashboards to visualize the insight

### 2.6 Description of the relationship between datasets:

### 2.7 Requirements

```
In [1]: import os
```

```
In [2]: import numpy                  as np
        import pandas                 as pd
        import matplotlib.pyplot      as plt
        import seaborn                as sns
        import sklearn.metrics        as Metrics
```

```
In [3]: ruta=os.getcwd()+'/Data/'
```

```
In [4]: customer = pd.read_csv(ruta+'olist_customers_dataset.csv', delimiter=',')
        order_items=pd.read_csv(ruta+'olist_order_items_dataset.csv', delimiter=',')
        orders=pd.read_csv(ruta+'olist_orders_dataset.csv', delimiter=',')
        products=pd.read_csv(ruta+'olist_products_dataset.csv', delimiter=',')
        order_payments=pd.read_csv(ruta+'olist_order_payments_dataset.csv', delimiter=',')
        reviews=pd.read_csv(ruta+'olist_order_reviews_dataset.csv', delimiter=',')
        order_items=pd.read_csv(ruta+'olist_order_items_dataset.csv', delimiter=',')
```

# 3 City and product trends

In this part we try to analyze which is the trend of online shopping by city.

For this analysis we generate a dataset called `df`, which is obtained by crossing the fields: `order_id`,`customer_id`, `customer_unique_id`, `ustomer_city`,`customer_state`, `order_item_id`, `product_id`, `product_category_name` of the dataframes: `olist_customers_dataset`, `olist_order_items_dataset`, `olist_orders_dataset`, `olist_products_dataset`. For this we use pyhton's `merge` function:

```
In [5]: # order_id:Identificador único del pedido. customer_id: Id del cliente.customer_unique_
        df1=orders[['order_id','customer_id']]
        df2=customer[['customer_id','customer_unique_id','customer_city','customer_state']]
        df=pd.merge(df1, df2, how="left", left_on="customer_id", right_on="customer_id")
```

```
In [6]: #product_id: Identificador único del producto.order_item_id: identifica el número de a
        df3=order_items[['order_id','order_item_id','product_id']]
        df=pd.merge(df, df3, how="left", left_on="order_id", right_on="order_id")
```

```
In [7]: #product_category_name:Categoría raíz del producto, en portugués
        df4=products[['product_id','product_category_name']]
        df=pd.merge(df, df4, how="left", left_on="product_id", right_on="product_id")
```

The states are by ISO code, we create an dictionary (`estados`) with these codes and the corresponding name, and another dictionary (`regiones`) with the regions for each state, then we add two columns to the dataframe, one with the name of the state and the other with the region, we do this for a more comfortable viewing.

```
In [8]: estados= {'AC':'Acre', 'AL':'Alagoas', 'AM':'Amazonas', 'AP':'Amapá', 'BA':'Bahía', 'C
                  'DF':'Distrito Federal', 'ES':'Espírito Santo', 'GO':'Goiás', 'MA':'Maranhão
                  'MS':'Mato Grosso del Sur', 'MT':'Mato Grosso', 'PA':'Pará', 'PB':'Paraíba',
                  'PI':'Piauí', 'PR':'Paraná', 'RJ':'Río de Janeiro', 'RN':'Río Grande del Nor
                  'RR':'Roraima', 'RS':'Río Grande del Sur', 'SC':'Santa Catarina', 'SE':'Serg
        df['name_state']=df['customer_state']
        for i in range(len(estados)):
            df["name_state"]=df["name_state"].str.replace(list(estados.keys())[i],list(estados
```

```
In [9]: region = ["Norte","Sur","Sudeste","Nordeste","CentroOeste"]
        regiones= {'AC':'Norte', 'AL':'Nordeste', 'AM':'Norte', 'AP':'Norte', 'BA':'Nordeste',
                   'CE':'Nordeste', 'DF':'CentroOeste', 'ES':'Sudeste', 'GO':'CentroOeste',
                   'MA':'Nordeste', 'MG':'Sudeste', 'MS':'CentroOeste', 'MT':'CentroOeste',
                   'PA':'Norte', 'PB':'Nordeste', 'PE':'Nordeste', 'PI':'Nordeste', 'PR':'Sur'
                   'RJ':'Sudeste', 'RN':'Nordeste', 'RO':'Norte', 'RR':'Norte', 'RS':'Sur',
                   'SC':'Sur', 'SE':'Nordeste', 'SP':'Sudeste', 'TO':'Norte'}
        df['regions']=df['customer_state']
        for i in range(len(estados)):
            df["regions"]=df["regions"].str.replace(list(regiones.keys())[i],list(regiones.valu
```

```
In [10]: df[['order_id','customer_city','customer_state','regions','name_state']].head(2)
```

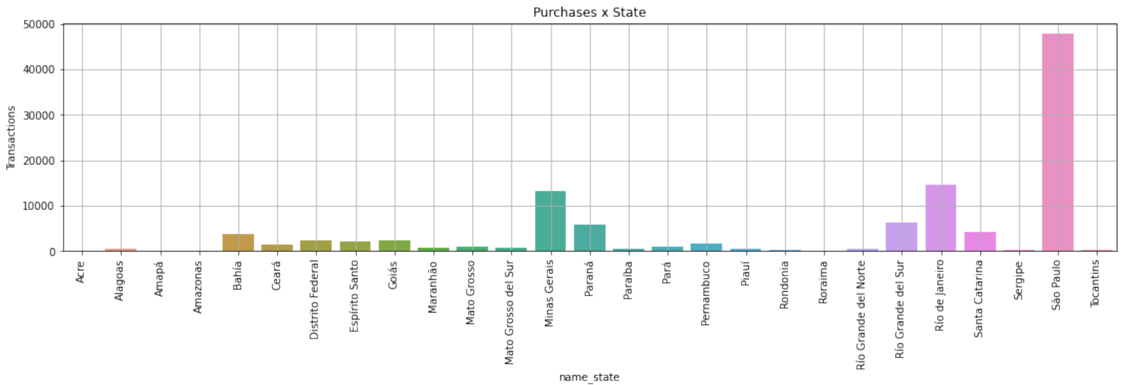
```
Out[10]:                            order_id customer_city customer_state   regions  \
         0  e481f51cbdc54678b7cc49136f2d6af7     sao paulo             SP   Sudeste
         1  53cdb2fc8bc7dce0b6741e2150273451     barreiras             BA  Nordeste

           name_state
         0  São Paulo
         1      Bahía
```

### 3.1 Purchases by state

We generate a graph with the amount of online purchases made in each state

```
In [11]: plt.figure(figsize=(18, 4))
         aux=df.groupby('name_state').size().to_frame().rename(columns={0:'Transactions'}).hea
```

```
ax=sns.barplot(x='name_state',y='Transactions',data=aux);
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
ax.set_title('Purchases x State')
ax.grid()
```



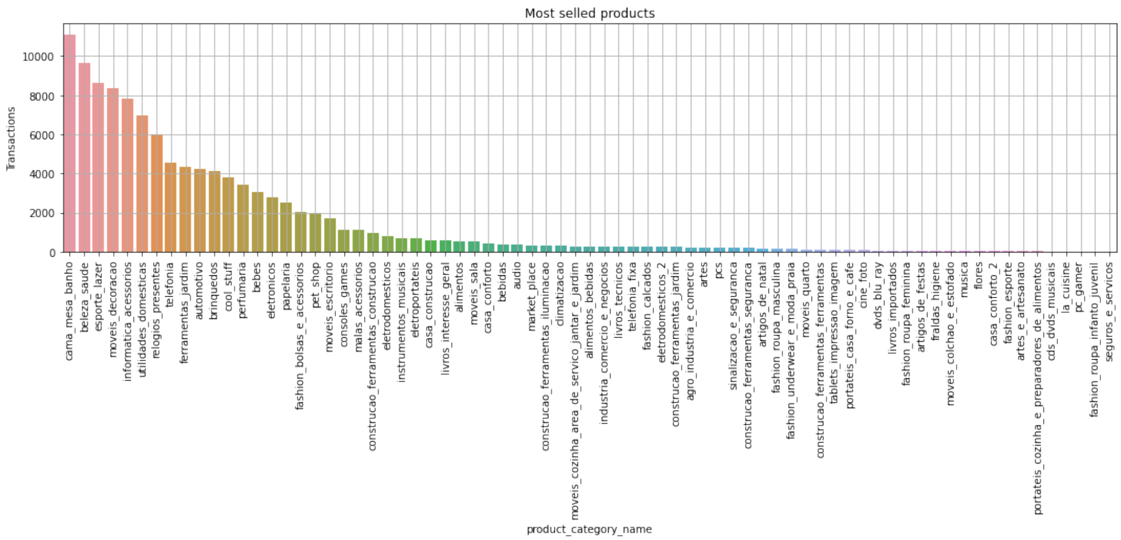

The states that make the most purchases are Sao Pablo and Rio de Janeiro, the two main states in the country.

### 3.2 Most selled products

The dataframe has 74 categories of products, we generate a graph with the amount of products sold by category and organize it descendingly.

```
In [12]: plt.figure(figsize=(18, 4))
         aux=df.groupby('product_category_name').size().to_frame().rename(columns={0:'Transact
         listaP=list(aux["product_category_name"])
         ax=sns.barplot(x='product_category_name',y='Transactions',data=aux);
         ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
         ax.set_title('Most selled products')
         ax.grid()
         print("The dataframe has ",len(df['product_category_name'].unique())," categories of p
```

```
The dataframe has  74  categories of products
```

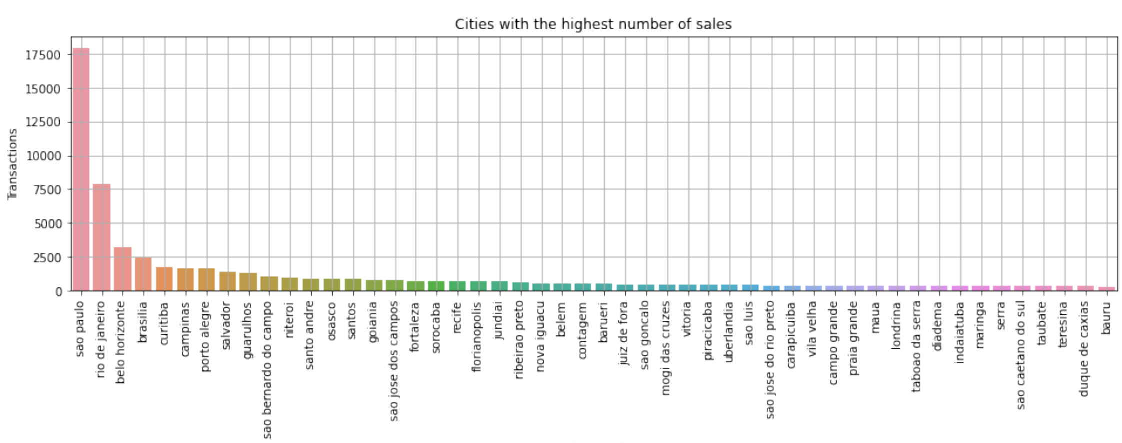



## 3.3 Cities with the highest number of sales

The dataframe has 4119 cities, we take the 50 cities that have the most registered sales, and we generate a graph with the number of sales per city, organized in descending order.

```
In [13]: plt.figure(figsize=(16, 4))
         aux=df.groupby('customer_city').size().to_frame().rename(columns={0:'Transactions'}).s
         listaC=list(aux["customer_city"])
         ax=sns.barplot(x='customer_city',y='Transactions',data=aux);
         ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
         ax.set_title('Cities with the highest number of sales')
         ax.grid()
         print("We have", len(df["customer_city"].unique()),"cities, the 50 that register the h
```

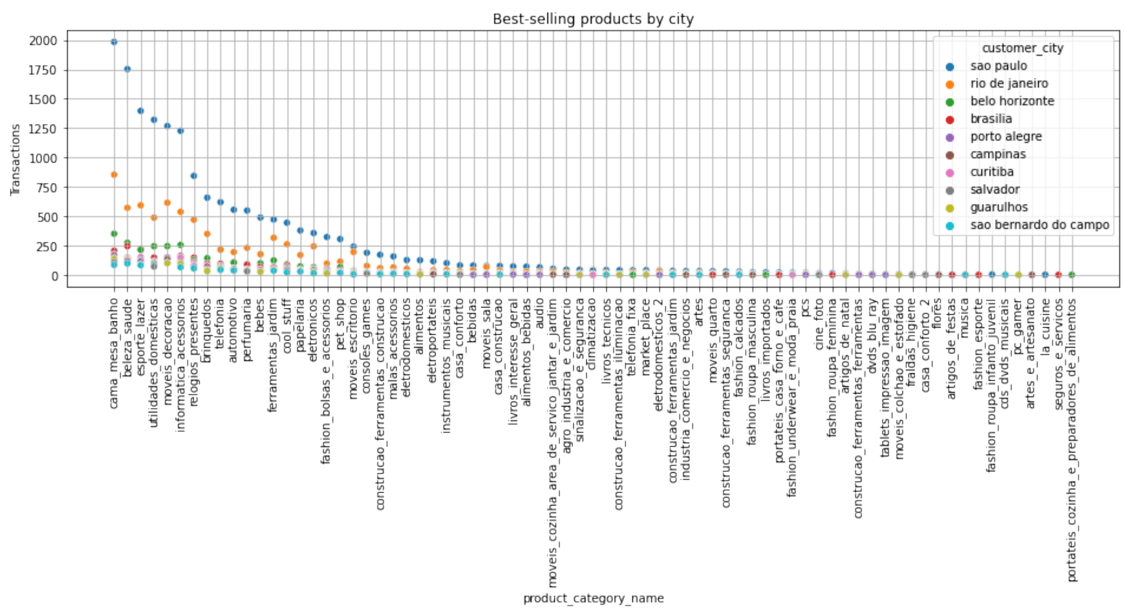
```
We have 4119 cities, the 50 that register the highest number of sales are shown
```



### 3.4 Best-selling products by city

The dataframe has 74 product categories, we generate a graph with the amount of products sold by city and we organize it in descending order, we take the 10 cities that register the highest number of sales.

```
In [14]: listaC2=listaC[:10]
         plt.figure(figsize=(16, 4))
         aux=df[(df["customer_city"].isin(listaC2))][["product_category_name","customer_city"]]
         aux=aux.groupby(["product_category_name","customer_city"]).size().to_frame().rename(co
         aux=aux.sort_values('Transactions', ascending=False)
         #aux=aux[:50]
         #ax=sns.lineplot(x='product_category_name', y='Number de transaciones', hue='customer_
         ax=sns.scatterplot(x='product_category_name', y='Transactions', hue='customer_city', c
         plt.xticks(rotation=90)
         ax.set_title('Best-selling products by city')
         ax.grid()
```

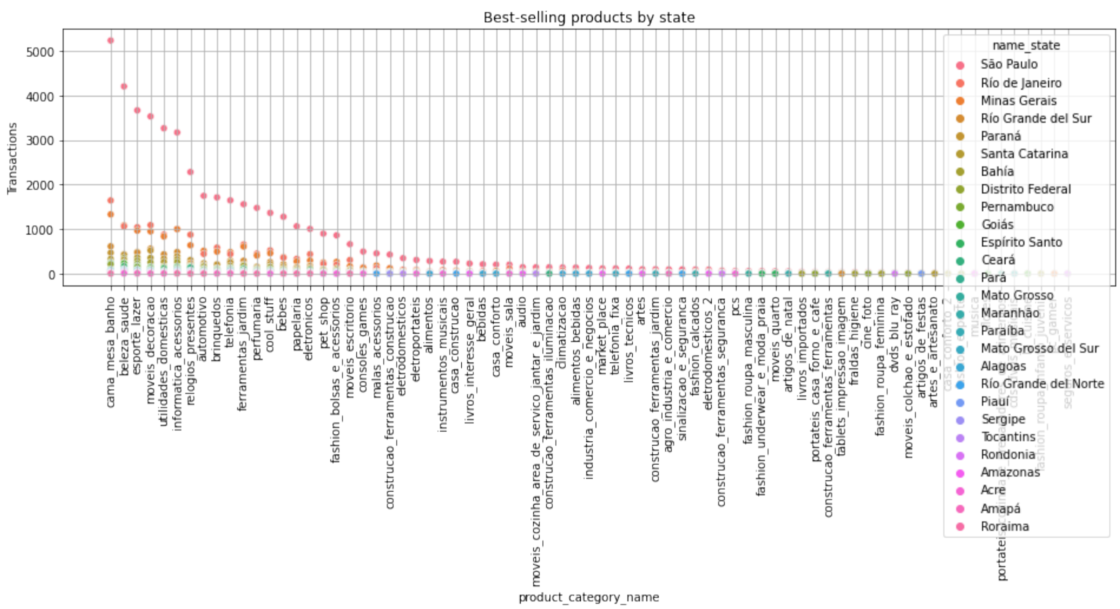



### 3.5 Best-selling products by state

The dataframe has 74 product categories, we generate a graph with the amount of products sold by state and we organize it in descending order.

```
In [15]: plt.figure(figsize=(16, 4))
         aux=df[["product_category_name","name_state"]]
```

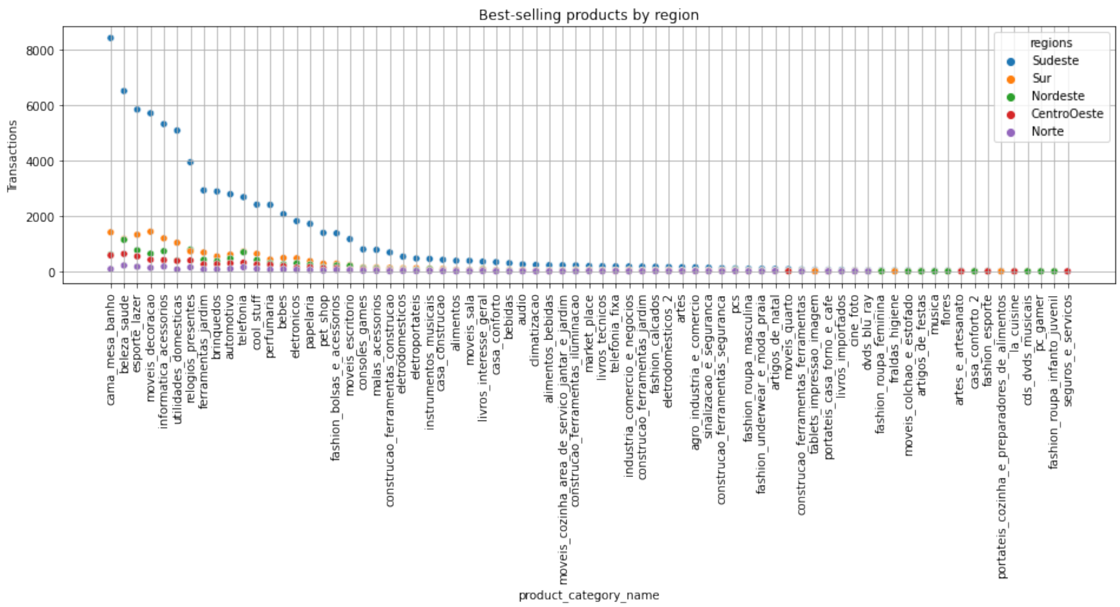
```
aux=aux.groupby(["product_category_name","name_state"]).size().to_frame().rename(colu
aux=aux.sort_values('Transactions', ascending=False)
#aux=aux[:50]
#ax=sns.lineplot(x='product_category_name', y='Number de transaciones', hue='customer
ax=sns.scatterplot(x='product_category_name', y='Transactions', hue='name_state', data
plt.xticks(rotation=90)
ax.set_title('Best-selling products by state')
ax.grid()
```




### 3.6 Best-selling products by region

The dataframe has 74 product categories, we generate a graph with the amount of products sold by region and we organize it in descending order.

```
In [16]: plt.figure(figsize=(16, 4))
         aux=df[["product_category_name","regions"]]
         aux=aux.groupby(["product_category_name","regions"]).size().to_frame().rename(columns
         aux=aux.sort_values('Transactions', ascending=False)
         #aux=aux[:50]
         #ax=sns.lineplot(x='product_category_name', y='Number de transaciones', hue='customer
         ax=sns.scatterplot(x='product_category_name', y='Transactions', hue='regions', data=a
         plt.xticks(rotation=90)
         ax.set_title('Best-selling products by region')
         ax.grid()
```




## 4 Highest Income per Category

```
In [17]: a=pd.merge(order_items, orders, on=['order_id'])
         merged_df=pd.merge(a, products, on=['product_id'])
```

```
In [18]: merged_df.order_status.value_counts()
```

```
Out[18]: delivered      110197
         shipped          1185
         canceled          542
         invoiced          359
         processing        357
         unavailable         7
         approved            3
         Name: order_status, dtype: int64
```

```
In [19]: top50categories_df=merged_df['price'].groupby(merged_df['product_category_name']).sum
         top50categories_df
```

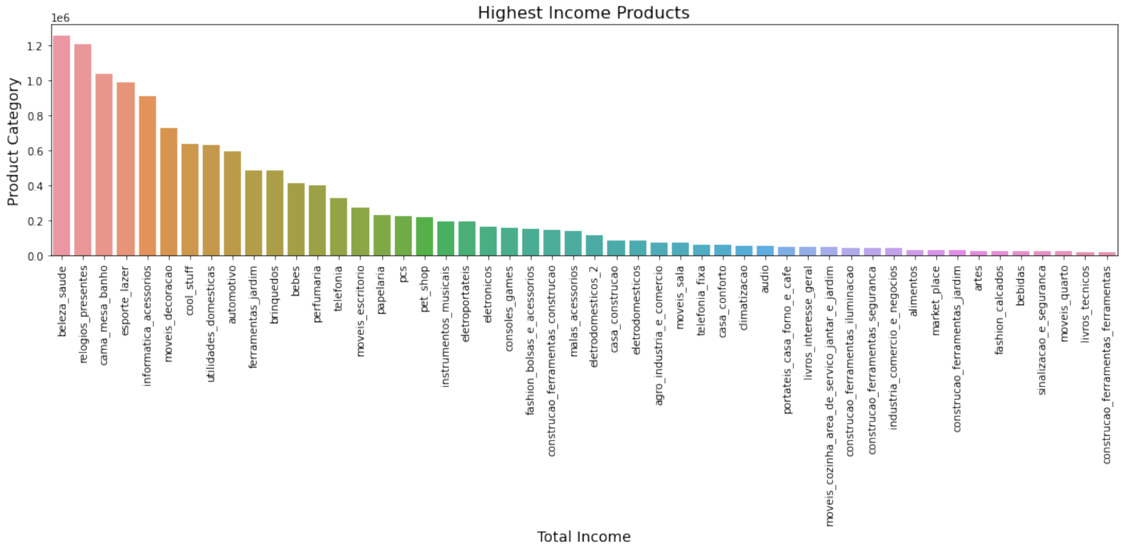
```
Out[19]: product_category_name
         beleza_saude                                     1258681.34
         relogios_presentes                               1205005.68
         cama_mesa_banho                                  1036988.68
         esporte_lazer                                     988048.97
         informatica_acessorios                            911954.32
         moveis_decoracao                                  729762.49
         cool_stuff                                        635290.85
```

```
utilidades_domesticas                                  632248.66
automotivo                                             592720.11
ferramentas_jardim                                     485256.46
brinquedos                                             483946.60
bebes                                                  411764.89
perfumaria                                             399124.87
telefonia                                              323667.53
moveis_escritorio                                      273960.70
papelaria                                              230943.23
pcs                                                    222963.13
pet_shop                                               214315.41
instrumentos_musicais                                  191498.88
eletroportateis                                        190648.58
eletronicos                                            160246.74
consoles_games                                         157465.22
fashion_bolsas_e_acessorios                            152823.54
construcao_ferramentas_construcao                      144677.59
malas_acessorios                                       140429.98
eletrodomesticos_2                                     113317.74
casa_construcao                                         83088.12
eletrodomesticos                                        80171.53
agro_industria_e_comercio                               72530.47
moveis_sala                                             68916.56
telefonia_fixa                                          59583.00
casa_conforto                                           58572.04
climatizacao                                            55024.96
audio                                                   50688.50
portateis_casa_forno_e_cafe                             47445.71
livros_interesse_geral                                  46856.88
moveis_cozinha_area_de_servico_jantar_e_jardim          46328.37
construcao_ferramentas_iluminacao                       41080.00
construcao_ferramentas_seguranca                        40544.52
industria_comercio_e_negocios                           39669.61
alimentos                                               29393.41
market_place                                            28378.47
construcao_ferramentas_jardim                           25715.89
artes                                                   24202.64
fashion_calcados                                        23562.77
bebidas                                                 22428.70
sinalizacao_e_seguranca                                 21509.23
moveis_quarto                                           20028.78
livros_tecnicos                                         19096.06
construcao_ferramentas_ferramentas                      15903.95
Name: price, dtype: float64
```

In [64]:
```
fig=plt.figure(figsize=(18,4))
sns.barplot(y=top50categories_df.values,x=top50categories_df.index)
plt.title('Highest Income Products',fontsize=16)
```

```
plt.xlabel('Total Income',fontsize=14)
plt.xticks(rotation=90)
plt.ylabel('Product Category',fontsize=14)
```

Out[64]: Text(0, 0.5, 'Product Category')




It's interesting looking at this chart that the highest incomes for the platform come from the category health, beauty.

# 5 Late Deliveries

In [21]:
```
delivered_time = pd.to_datetime(merged_df.order_delivered_customer_date)
estimated_time = pd.to_datetime(merged_df.order_estimated_delivery_date)
merged_df["late_delivery"] = delivered_time - estimated_time
merged_df["late_delivery"]=merged_df["late_delivery"]/np.timedelta64(1,"D")
```

In [22]:
```
late_deliveries_df = merged_df[merged_df.late_delivery > 0]
late_deliveries_df[["product_category_name", "order_delivered_customer_date", "order_
```

Out[22]:
```
                product_category_name order_delivered_customer_date  \
5                           cool_stuff           2017-08-31 20:19:52
16                  ferramentas_jardim           2018-04-02 22:32:10
29               utilidades_domesticas           2017-07-10 11:46:40
36                        beleza_saude           2018-03-29 18:17:31
68                        beleza_saude           2018-05-23 17:51:15
69                        beleza_saude           2018-05-23 17:51:15
71                        beleza_saude           2018-02-24 16:26:53
97         fashion_bolsas_e_acessorios           2018-08-07 13:56:52
120                    cama_mesa_banho           2017-09-04 13:34:13
121                    cama_mesa_banho           2018-03-29 23:42:46
```

```
    order_estimated_delivery_date  late_delivery
5             2017-08-24 00:00:00       7.847130
16            2018-03-23 00:00:00      10.939005
29            2017-07-10 00:00:00       0.490741
36            2018-03-29 00:00:00       0.762164
68            2018-05-16 00:00:00       7.743924
69            2018-05-16 00:00:00       7.743924
71            2018-02-21 00:00:00       3.685336
97            2018-08-07 00:00:00       0.581157
120           2017-08-31 00:00:00       4.565428
121           2018-03-16 00:00:00      13.988032
```

# 6 Difference between Delivered Time and Delivered Estimated Time vs Reviews

```
In [23]: delivered_time = pd.to_datetime(merged_df.order_delivered_customer_date)
         approved_time = pd.to_datetime(merged_df.order_approved_at)
         merged_df["delivery_time"] = delivered_time - approved_time
         merged_df["delivery_time"] = merged_df["delivery_time"]/np.timedelta64(1,"D")
```

```
In [24]: delivery_time_df = merged_df[merged_df.delivery_time > 0]
         delivery_time_df[["product_category_name", "order_delivered_customer_date", "order_es
```

```
Out[24]:   product_category_name order_delivered_customer_date  \
         0            cool_stuff           2017-09-20 23:43:48
         1            cool_stuff           2017-07-13 20:39:29
         2            cool_stuff           2018-06-04 18:34:26
         3            cool_stuff           2017-08-09 21:26:33
         4            cool_stuff           2017-08-24 20:04:21
         5            cool_stuff           2017-08-31 20:19:52
         6            cool_stuff           2018-03-28 21:57:44
         7            cool_stuff           2017-08-14 18:13:03
         8            cool_stuff           2017-06-26 13:52:03
         9              pet_shop           2017-05-12 16:04:24

           order_estimated_delivery_date  late_delivery
         0           2017-09-29 00:00:00      -8.011250
         1           2017-07-26 00:00:00     -12.139248
         2           2018-06-07 00:00:00      -2.226088
         3           2017-08-25 00:00:00     -15.106563
         4           2017-09-01 00:00:00      -7.163646
         5           2017-08-24 00:00:00       7.847130
         6           2018-04-12 00:00:00     -14.084907
         7           2017-09-06 00:00:00     -22.240938
         8           2017-07-06 00:00:00      -9.422187
         9           2017-05-15 00:00:00      -2.330278
```

```
In [25]: orders1 = orders.copy()
         orders1 = orders1[['order_id', 'order_delivered_customer_date', 'order_estimated_deli
         orders1['order_delivered_customer_date'] = pd.to_datetime(orders1['order_delivered_cus
         orders1['order_estimated_delivery_date'] = pd.to_datetime(orders1['order_estimated_del
         orders1['Estimated_Delivered'] = orders1['order_delivered_customer_date']-orders1['ord
         orders1['Estimated_Delivered'] = orders1['Estimated_Delivered'].dt.days
         orders1.head()
```

```
Out[25]:                            order_id order_delivered_customer_date  \
         0  e481f51cbdc54678b7cc49136f2d6af7           2017-10-10 21:25:13
         1  53cdb2fc8bc7dce0b6741e2150273451           2018-08-07 15:27:45
         2  47770eb9100c2d0c44946d9cf07ec65d           2018-08-17 18:06:29
         3  949d5b44dbf5de918fe9c16f97b45f8a           2017-12-02 00:28:42
         4  ad21c59c0840e6cb83a9ceb5573f8159           2018-02-16 18:17:02

           order_estimated_delivery_date  Estimated_Delivered
         0                    2017-10-18                 -8.0
         1                    2018-08-13                 -6.0
         2                    2018-09-04                -18.0
         3                    2017-12-15                -13.0
         4                    2018-02-26                -10.0
```

Here we calculate the delta between Estimated Delivery date versus the real time the costumer delivered time

```
In [26]: orders_reviews = pd.merge(orders1, reviews, on="order_id", how="left")
         orders_reviews = orders_reviews[['order_id', 'order_delivered_customer_date', 'order_e
         orders_reviews['Puntuality'] = np.where(orders_reviews['Estimated_Delivered'] <= 0, "I
         orders_reviews.head()
```

```
Out[26]:                            order_id order_delivered_customer_date  \
         0  e481f51cbdc54678b7cc49136f2d6af7           2017-10-10 21:25:13
         1  53cdb2fc8bc7dce0b6741e2150273451           2018-08-07 15:27:45
         2  47770eb9100c2d0c44946d9cf07ec65d           2018-08-17 18:06:29
         3  949d5b44dbf5de918fe9c16f97b45f8a           2017-12-02 00:28:42
         4  ad21c59c0840e6cb83a9ceb5573f8159           2018-02-16 18:17:02

           order_estimated_delivery_date  Estimated_Delivered  review_score Puntuality
         0                    2017-10-18                 -8.0             4   Punctual
         1                    2018-08-13                 -6.0             4   Punctual
         2                    2018-09-04                -18.0             5   Punctual
         3                    2017-12-15                -13.0             5   Punctual
         4                    2018-02-26                -10.0             5   Punctual
```

Now, we make cathegorical variables by the reviews scores as punctual or unpunctual by the difference between Estimated Delivery and Real Delivery. When this delta is negative or zero, we classify it as “Punctual” because the company accomplish his terms of service, else the variable takes the “Unpunctual” value

```
In [27]: x_review_punctual = pd.crosstab(orders_reviews['review_score'], orders_reviews['Puntu
         x_review_punctual
```

```
Out[27]: Puntuality    Punctual  Unpunctual
         review_score
         1                 6205        5653
         2                 2442         793
         3                 7345         942
         4                18382         818
         5                56076        1344
```

Here we can see the count by Puntuality and the review scores. Scannig the table, we realize that there is a inverse correlation between the score of the review and the puntuality of the delivery. This relationship is inverse. When the company is punctual in his delivery, then the costumer tends to make a good review with a good score and viceversa.

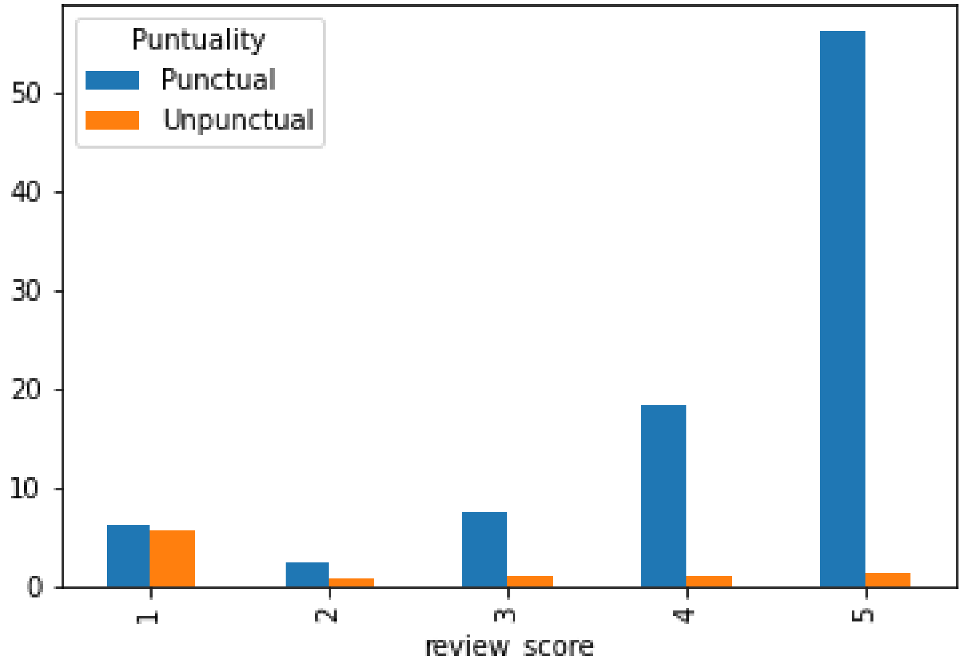
```
In [28]: x_review_punctual_p = pd.crosstab(orders_reviews['review_score'], orders_reviews['Pun
         x_review_punctual_p.reset_index
         x_review_punctual_p
```

```
Out[28]: Puntuality    Punctual  Unpunctual
         review_score
         1                6.205       5.653
         2                2.442       0.793
         3                7.345       0.942
         4               18.382       0.818
         5               56.076       1.344
```

Here we see the table by percentages

```
In [29]: x_review_punctual_p.plot(kind="bar")
         #x_review_punctual_p.plot(kind="line")
```

```
Out[29]: <AxesSubplot:xlabel='review_score'>
```




---

## 7 Purchasing trend by hour or day of the week

In [30]:
```
#unifying the variables in one data frame
orders_by_date = pd.merge(left = orders , right = order_payments , how='left', left_o
orders_by_date.head()
```

Out[30]:
```
                           order_id                       customer_id  \
0  e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d
1  e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d
2  e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d
3  53cdb2fc8bc7dce0b6741e2150273451  b0830fb4747a6c6d20dea0b8c802d7ef
4  47770eb9100c2d0c44946d9cf07ec65d  41ce2a54c0b03bf3443c3d931a367089

  order_status order_purchase_timestamp   order_approved_at  \
0    delivered      2017-10-02 10:56:33 2017-10-02 11:07:15
1    delivered      2017-10-02 10:56:33 2017-10-02 11:07:15
2    delivered      2017-10-02 10:56:33 2017-10-02 11:07:15
3    delivered      2018-07-24 20:41:37 2018-07-26 03:24:27
4    delivered      2018-08-08 08:38:49 2018-08-08 08:55:23

  order_delivered_carrier_date order_delivered_customer_date  \
0          2017-10-04 19:55:00           2017-10-10 21:25:13
```

```
1          2017-10-04 19:55:00          2017-10-10 21:25:13
2          2017-10-04 19:55:00          2017-10-10 21:25:13
3          2018-07-26 14:31:00          2018-08-07 15:27:45
4          2018-08-08 13:50:00          2018-08-17 18:06:29

  order_estimated_delivery_date  payment_sequential payment_type  \
0           2017-10-18 00:00:00                 1.0  credit_card
1           2017-10-18 00:00:00                 3.0      voucher
2           2017-10-18 00:00:00                 2.0      voucher
3           2018-08-13 00:00:00                 1.0       boleto
4           2018-09-04 00:00:00                 1.0  credit_card

   payment_installments  payment_value
0                   1.0          18.12
1                   1.0           2.00
2                   1.0          18.59
3                   1.0         141.46
4                   3.0         179.12
```

In [31]:
```
#Elimitating columns we dont need yet
orders_by_date = orders_by_date.drop(['order_approved_at', 'order_delivered_carrier_da
       'order_delivered_customer_date', 'order_estimated_delivery_date','payment_seque
```

In [32]:
```
#validation of any null values in the order_payment data frame to use
orders_by_date.isnull().any()
```

Out[32]:
```
order_id                      False
customer_id                   False
order_status                  False
order_purchase_timestamp      False
payment_value                  True
dtype: bool
```

In [33]:
```
#Elimitating nulls
orders_by_date = orders_by_date.dropna()
orders_by_date = orders_by_date.reset_index(drop=True)
orders_by_date
```

Out[33]:
```
                                order_id                       customer_id  \
0       e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d
1       e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d
2       e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d
3       53cdb2fc8bc7dce0b6741e2150273451  b0830fb4747a6c6d20dea0b8c802d7ef
4       47770eb9100c2d0c44946d9cf07ec65d  41ce2a54c0b03bf3443c3d931a367089
...                                  ...                               ...
103881  9c5dedf39a927c1b2549525ed64a053c  39bd1228ee8140590ac3aca26f2dfe00
103882  63943bddc261676b46f01ca7ac2f7bd8  1fca14ff2861355f6e5f14306ff977a7
103883  83c1379a015df1e13d02aae0204711ab  1aa71eb042121263aafbe80c1b562c9c
103884  11c177c8e97725db2631073c19f07b62  b331b74b18dc79bcdf6532d51e1637c1
```

```
103885  66dea50a8b16d9b4dee7af250b4be1a5  edb027a75a1449115f6b43211ae02a24

       order_status order_purchase_timestamp  payment_value
0         delivered      2017-10-02 10:56:33          18.12
1         delivered      2017-10-02 10:56:33           2.00
2         delivered      2017-10-02 10:56:33          18.59
3         delivered      2018-07-24 20:41:37         141.46
4         delivered      2018-08-08 08:38:49         179.12
...             ...                      ...            ...
103881    delivered      2017-03-09 09:54:05          85.08
103882    delivered      2018-02-06 12:58:58         195.00
103883    delivered      2017-08-27 14:46:43         271.01
103884    delivered      2018-01-08 21:28:27         441.16
103885    delivered      2018-03-08 20:57:30          86.86

[103886 rows x 5 columns]
```

In [34]:
```python
orders_by_date.isnull().any()
```

Out[34]:
```
order_id                    False
customer_id                 False
order_status                False
order_purchase_timestamp    False
payment_value               False
dtype: bool
```

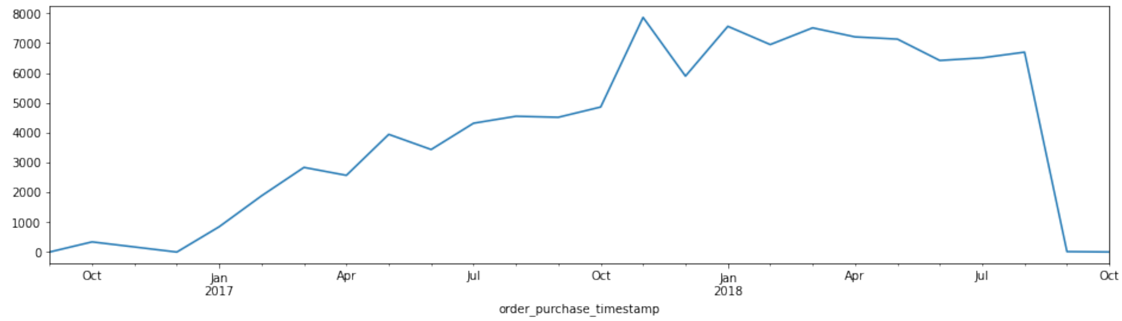
In [35]:
```python
#First we need to change the format of the column DATE to datetime:
orders_by_date['order_purchase_timestamp'] = pd.to_datetime(orders_by_date['order_purc
orders_by_date
```

Out[35]:
```
                                order_id                       customer_id  \
0       e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d
1       e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d
2       e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d
3       53cdb2fc8bc7dce0b6741e2150273451  b0830fb4747a6c6d20dea0b8c802d7ef
4       47770eb9100c2d0c44946d9cf07ec65d  41ce2a54c0b03bf3443c3d931a367089
...                                  ...                               ...
103881  9c5dedf39a927c1b2549525ed64a053c  39bd1228ee8140590ac3aca26f2dfe00
103882  63943bddc261676b46f01ca7ac2f7bd8  1fca14ff2861355f6e5f14306ff977a7
103883  83c1379a015df1e13d02aae0204711ab  1aa71eb042121263aafbe80c1b562c9c
103884  11c177c8e97725db2631073c19f07b62  b331b74b18dc79bcdf6532d51e1637c1
103885  66dea50a8b16d9b4dee7af250b4be1a5  edb027a75a1449115f6b43211ae02a24

       order_status order_purchase_timestamp  payment_value
0         delivered      2017-10-02 10:56:33          18.12
1         delivered      2017-10-02 10:56:33           2.00
2         delivered      2017-10-02 10:56:33          18.59
3         delivered      2018-07-24 20:41:37         141.46
4         delivered      2018-08-08 08:38:49         179.12
```

```
...              ...                           ...               ...
103881    delivered      2017-03-09 09:54:05            85.08
103882    delivered      2018-02-06 12:58:58           195.00
103883    delivered      2017-08-27 14:46:43           271.01
103884    delivered      2018-01-08 21:28:27           441.16
103885    delivered      2018-03-08 20:57:30            86.86

[103886 rows x 5 columns]
```

In [36]:
```python
#Then, check whether the number of orders has increased over time
plt.figure(figsize=(16, 4))
monthly_payments = orders_by_date.groupby(orders_by_date['order_purchase_timestamp'].d
monthly_payments.plot.line()
```

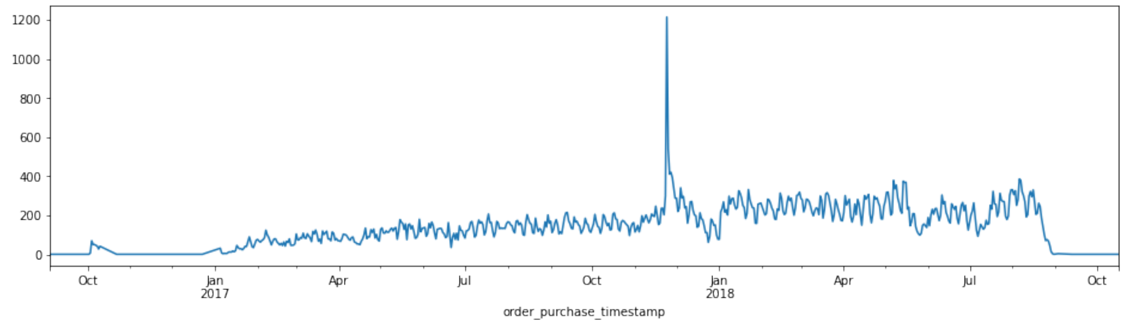
Out[36]: <AxesSubplot:xlabel='order_purchase_timestamp'>




As shown in the graph, the number of orders has been increasing over the 2017, till around November-December 2017 where we can see a decreasing, and around January 2018 continue fluctuating but increasing till October when the data end.

In [37]:
```python
#Then, check whether the number of orders has increased over time
plt.figure(figsize=(16, 4))
Daily_payments = orders_by_date.groupby(orders_by_date['order_purchase_timestamp'].dt
Daily_payments.plot.line()
```

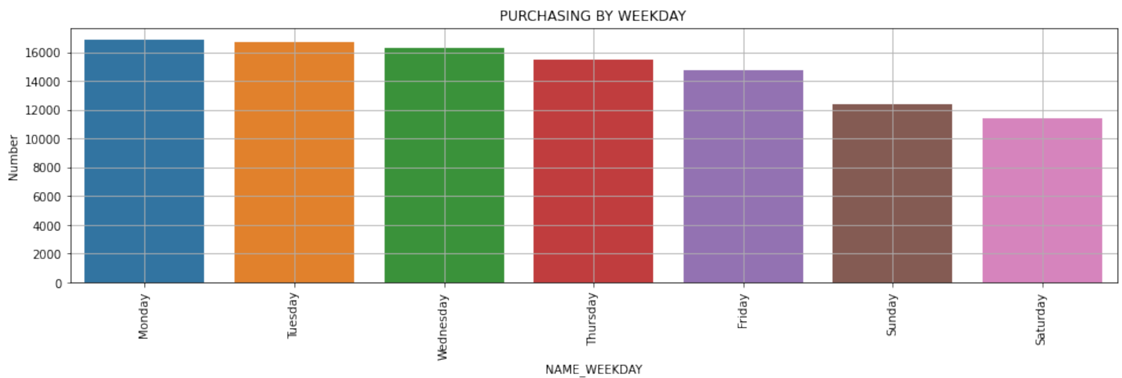
Out[37]: <AxesSubplot:xlabel='order_purchase_timestamp'>



If we see the graph daily we can see how and the end of November 2017 the number of orders increased significative for some days and later decreasing, we are evaluating the event that took place those days to see the variability of the values.

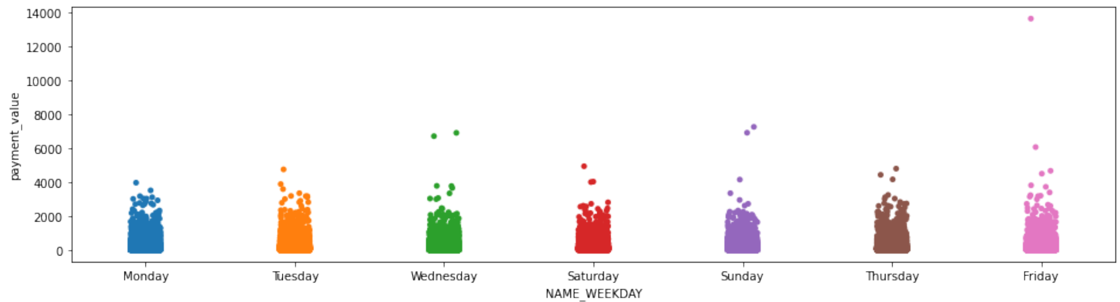
```
In [38]: #Adding the days to the Data frame
         orders_by_date["NAME_WEEKDAY"] = orders_by_date['order_purchase_timestamp'].dt.day_nam
```

```
In [39]: plt.figure(figsize=(16, 4))
         aux= orders_by_date.groupby("NAME_WEEKDAY").size().to_frame().rename(columns={0:'Numbe
         listaC=list(aux["NAME_WEEKDAY"])
         ax=sns.barplot(x="NAME_WEEKDAY",y='Number',data=aux);
         ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
         ax.set_title('PURCHASING BY WEEKDAY')
         ax.grid()
```




When we evaluate the number of purchases per week we can see how, during the week, the largest number of purchases is found on Mondays followed closely with the other days of the week and lastly we find the weekends, with Saturday being the day with the least amount of purchases.

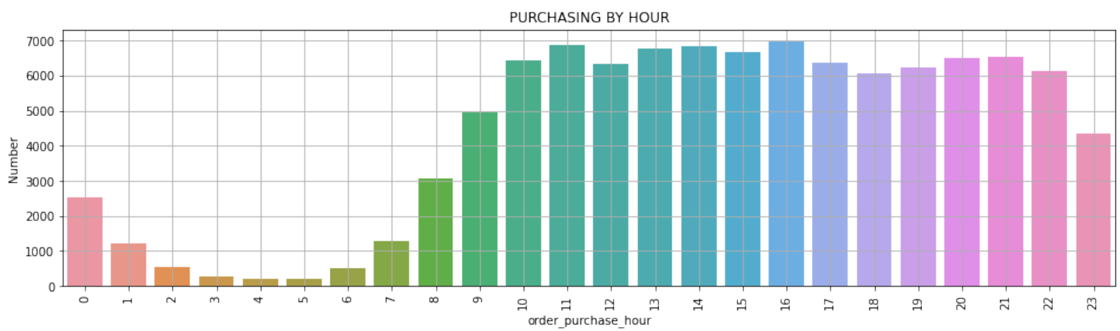
```
In [40]: plt.figure(figsize=(16, 4))
         ax = sns.stripplot(x="NAME_WEEKDAY", y="payment_value", data = orders_by_date)
         plt.ylabel('payment_value')
         plt.show()
```



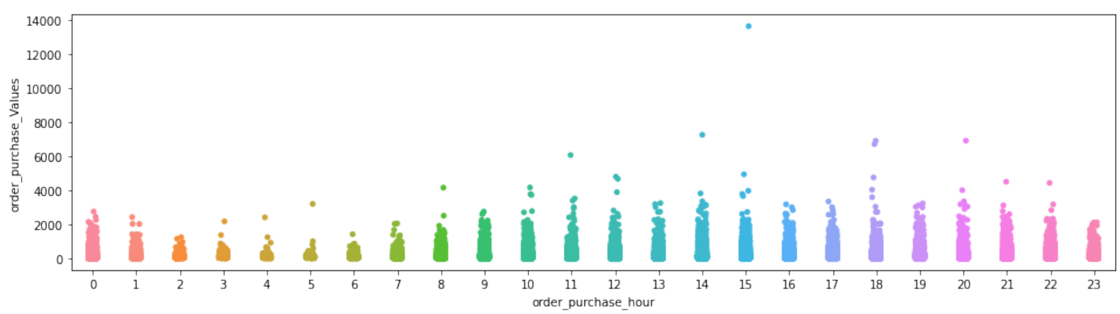
As shown in the graph, the sales values are more or less grouped below 3000, only a few values exceeded 3000 and are located below 6000, another four sales above 8000 and a single one per ma of 13000 reais.

```
In [41]: #Adding the hours to the Data frame
         orders_by_date["order_purchase_hour"] = orders_by_date['order_purchase_timestamp'].dt
```

```
In [42]: plt.figure(figsize=(16, 4))
         aux= orders_by_date.groupby("order_purchase_hour").size().to_frame().rename(columns={
         listaC=list(aux["order_purchase_hour"])
         ax=sns.barplot(x="order_purchase_hour",y='Number',data=aux);
         ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
         ax.set_title('PURCHASING BY HOUR')
         ax.grid()
```




```
In [43]: plt.figure(figsize=(16, 4))
         bx = sns.stripplot(x="order_purchase_hour", y="payment_value", data = orders_by_date)
         plt.ylabel('order_purchase_Values')
         plt.show()
```




Here in the Stripplox we can see a better look of how the purchasings are accumulated over the hours with more of the values around 3000 and less over the 3000 and 9000 and just one value below the 14000.

```
In [44]: plt.figure(figsize=(16, 4))
         aux = orders_by_date[["order_purchase_hour","NAME_WEEKDAY"]]
         aux = aux.groupby(["order_purchase_hour","NAME_WEEKDAY"]).size().to_frame().rename(co
         #aux = aux.sort_values('Number', ascending=False)
         #aux=aux[:50]
         ax=sns.scatterplot(x="order_purchase_hour", y='Number', hue="NAME_WEEKDAY", data=aux)
         #ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
         ax.set_title('PURCHASE BY HOUR AND DAY WEEKDAY ')
         ax.grid()
```


Here we can see the relation between the day of the week and the purchase hour, we can observe how is the behavior.

```
In [45]: orders_by_date.pivot_table("payment_value","order_purchase_hour", aggfunc=np.sum)
```

Out[45]:

| order_purchase_hour | payment_value |
|---|---|
| 0 | 374429.87 |
| 1 | 175968.42 |
| 2 | 66206.97 |
| 3 | 41914.04 |
| 4 | 28583.21 |
| 5 | 26216.23 |
| 6 | 67684.37 |
| 7 | 182607.59 |
| 8 | 463172.92 |
| 9 | 799881.96 |
| 10 | 993466.13 |
| 11 | 1034498.00 |
| 12 | 995854.79 |
| 13 | 1029882.85 |
| 14 | 1109777.98 |
| 15 | 1063015.78 |
| 16 | 1100540.43 |
| 17 | 987826.22 |
| 18 | 961676.61 |

```
19                          969475.12
20                         1006763.35
21                          984404.18
22                          921667.28
23                          623357.82
```

We did a pivot to see the total value of all purchasing by each hour and by date below.

In [46]:
```
orders_by_date.pivot_table("payment_value","NAME_WEEKDAY", aggfunc=np.sum)
```

Out[46]:
```
              payment_value
NAME_WEEKDAY
Friday           2307128.20
Monday           2622457.97
Saturday         1768427.68
Sunday           1872456.36
Thursday         2384544.22
Tuesday          2560743.03
Wednesday        2493114.66
```

Purchasing by date below.

# 8 Payment type trends

In [47]:
```
efective_orders=orders[(orders.order_status!= 'canceled') & (orders.order_status!= 'u
```

In [48]:
```
#payment_orders= pd.concat([order_payments,efective_orders], axis=1)
payment_orders=pd.merge(order_payments, efective_orders)
```

In [49]:
```
import datetime
timesMonth=[]
timesYear=[]
weekday=[]
hour=[]
for i in payment_orders['order_purchase_timestamp']:
    fecha=pd.to_datetime(i)
    timesMonth.append(fecha.month)
    timesYear.append(fecha.year)
    weekday.append(fecha.dayofweek)
    hour.append(fecha.hour)
payment_orders['Month']=timesMonth
payment_orders['Year']=timesYear
payment_orders['WeekDay']=weekday
payment_orders['Hour']=hour
payment_orders.head()
# pd.datetime.now().year*100+pd.datetime.now().month
```

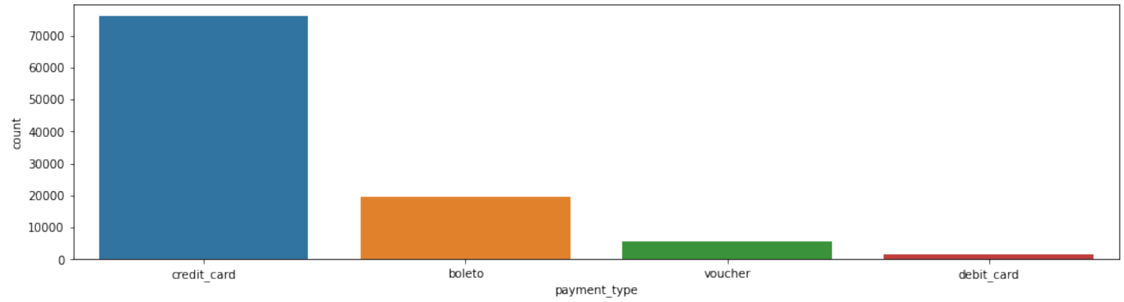
Out[49]:
```
                           order_id  payment_sequential payment_type  \
0  b81ef226f3fe1789b1e8b2acac839d17                   1  credit_card
```

```
1  a9810da82917af2d9aefd1278f1dcfa0                    1  credit_card
2  25e8ea4e93396b6fa0d3dd708e76c1bd                    1  credit_card
3  ba78997921bbcdc1373bb41e913ab953                    1  credit_card
4  42fdf880ba16b47b59251dd489d4441a                    1  credit_card

   payment_installments  payment_value                       customer_id  \
0                     8          99.33  0a8556ac6be836b46b3e89920d59291c
1                     1          24.39  f2c7fc58a9de810828715166c672f10a
2                     1          65.71  25b14b69de0b6e184ae6fe2755e478f9
3                     8         107.78  7a5d8efaaa1081f800628c30d2b0728f
4                     2         128.45  15fd6fb8f8312dbb4674e4518d6fa3b3

  order_status order_purchase_timestamp    order_approved_at  \
0    delivered      2018-04-25 22:01:49  2018-04-25 22:15:09
1    delivered      2018-06-26 11:01:38  2018-06-26 11:18:58
2    delivered      2017-12-12 11:19:55  2017-12-14 09:52:34
3    delivered      2017-12-06 12:04:06  2017-12-06 12:13:20
4    delivered      2018-05-21 13:59:17  2018-05-21 16:14:41

  order_delivered_carrier_date order_delivered_customer_date  \
0          2018-05-02 15:20:00           2018-05-09 17:36:51
1          2018-06-28 14:18:00           2018-06-29 20:32:09
2          2017-12-15 20:13:22           2017-12-18 17:24:41
3          2017-12-07 20:28:28           2017-12-21 01:35:51
4          2018-05-22 11:46:00           2018-06-01 21:44:53

  order_estimated_delivery_date  Month  Year  WeekDay  Hour
0           2018-05-22 00:00:00      4  2018        2    22
1           2018-07-16 00:00:00      6  2018        1    11
2           2018-01-04 00:00:00     12  2017        1    11
3           2018-01-04 00:00:00     12  2017        2    12
4           2018-06-13 00:00:00      5  2018        0    13
```

**Most people in Brazil E Commerce use Credit Card to pay their buys**

In [50]:
```python
plt.figure(figsize=(16, 4))
sns.countplot('payment_type',data=payment_orders)

plt.show()
```

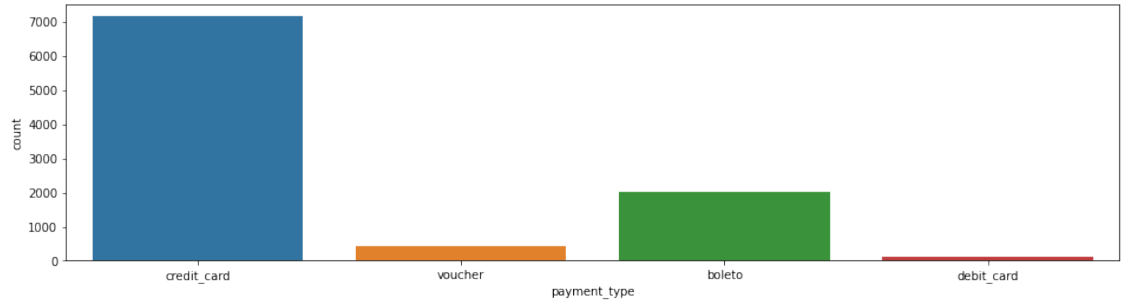
```
/home/jovyan/.local/lib/python3.6/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
```




**Sales of best-selling products, show the same behavior as total sales**

```
In [51]: #Sales with more seller products(mor than 100 units per product)
         dfrs = order_items.groupby('product_id').count()
         dfrs.sort_values(by=['order_id'], ascending=False, inplace=True)
         dfrs1 = dfrs[(dfrs['order_id']>100)]
         ListProducts=dfrs1.index
         ListOrders=order_items[(order_items['product_id'].isin(ListProducts))]['order_id'].uni
         pop=payment_orders[(payment_orders['order_id']).isin(ListOrders)]
         plt.figure(figsize=(16, 4))
         sns.countplot('payment_type',data=pop)
         plt.show()
```

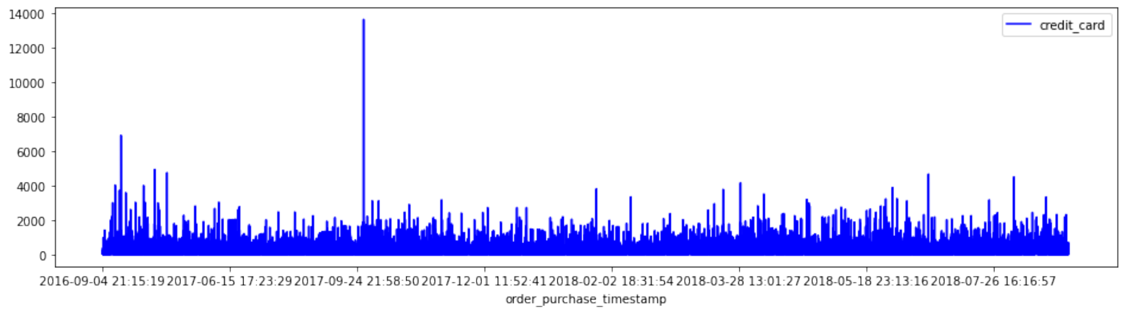
```
/home/jovyan/.local/lib/python3.6/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass
  FutureWarning
```

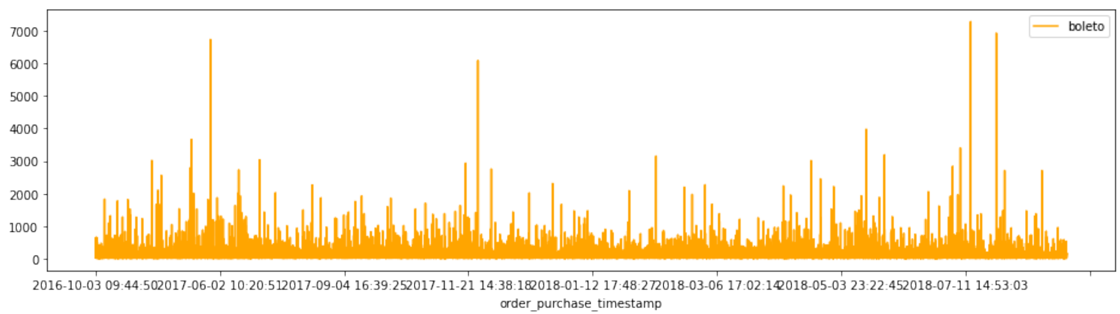


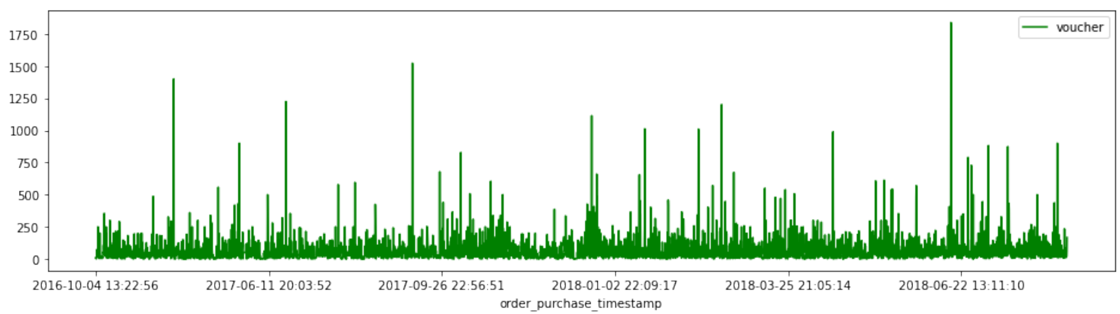

## 8.1 There is no relevant information in a general line of time

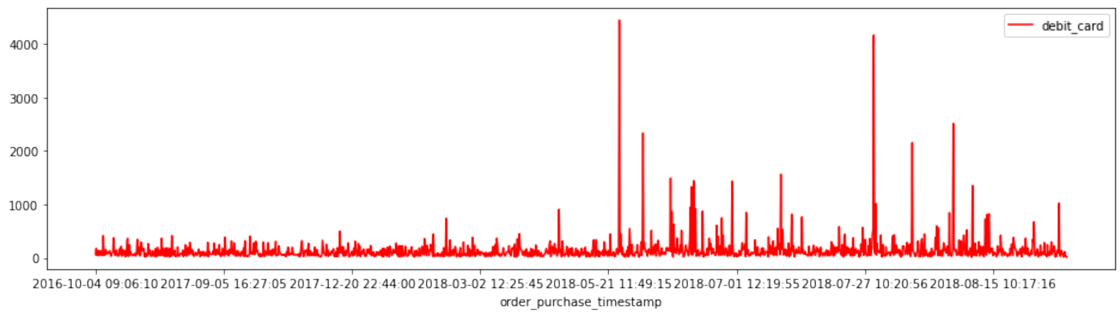
```
In [52]: # Comportamiento en el tiempo por Tipo de Pago
         payment_orders.sort_values(by=['order_purchase_timestamp'], inplace=True)
         listPaymentT = ['credit_card','boleto','voucher','debit_card']
         color = ['Blue','Orange','Green','Red']
```

```
for i,var in enumerate(listPaymentT):
    #print(var)
    temp = payment_orders[(payment_orders['payment_type'] == var)][['order_purchase_t
    temp.plot(figsize=(16, 4), kind='line',x='order_purchase_timestamp',y='payment_val
    plt.tittle = 'Payment Type ' + var
    #plt.legend()
plt.show()
```



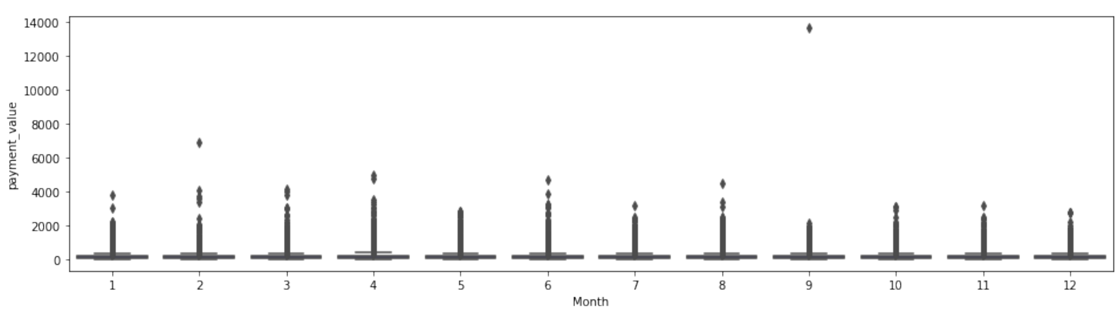



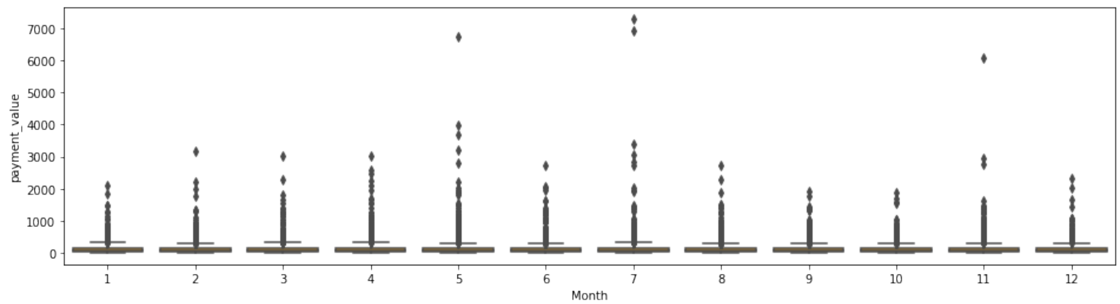



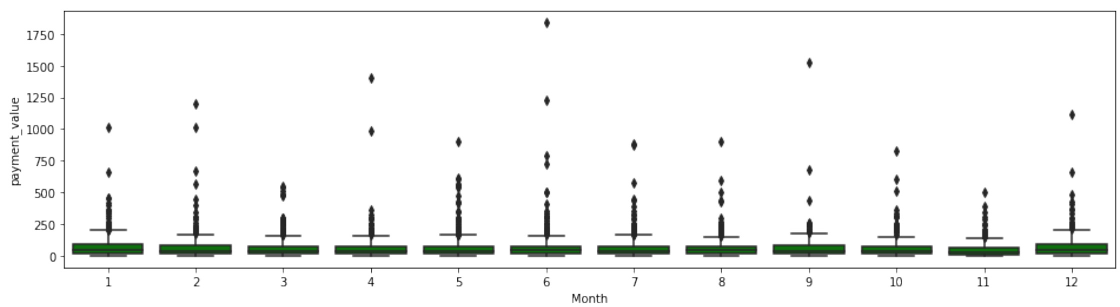


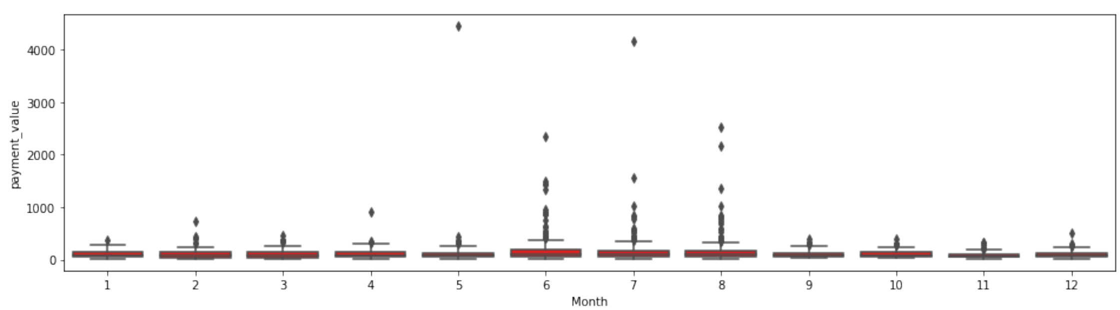

## 8.2 In Monthly bases Credit card payment has shown changes in September, payments with boleto show some peak in may, july and november,debit-card is more used in Jun to august

In [53]:
```
#Behavior Monthly by Payment Type

payment_orders.sort_values(by=['Month'], inplace=True)
listPaymentT = ['credit_card','boleto','voucher','debit_card']
color = ['Blue','Orange','Green','Red']

for i,var in enumerate(listPaymentT):
    plt.figure(figsize=(16, 4))
    #print(var)
    temp = payment_orders[(payment_orders['payment_type'] == var)][['Month','payment_
    #plt.subplot(2,2,i+1)
    sns.boxplot(x='Month',y='payment_value', data=temp ,color=color[i])
    #plt.tittle = 'Payment Type ' + var
    #plt.legend()
    plt.show()
```





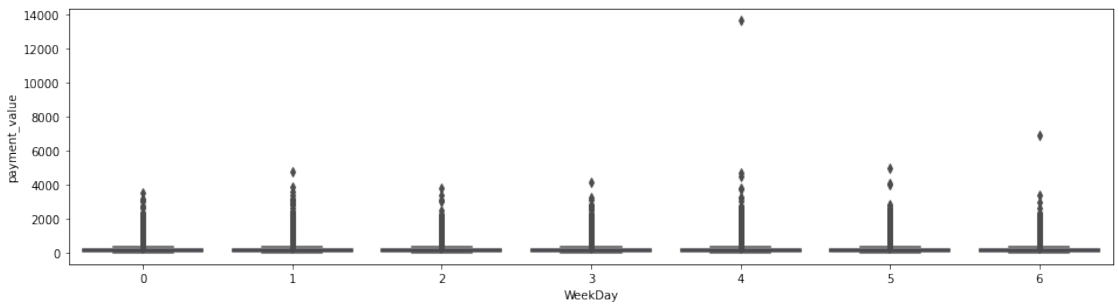




### 8.3 Thursday and Saturday shows peaks of buyers in credit card, boletos and bouchers are some disperse along week.

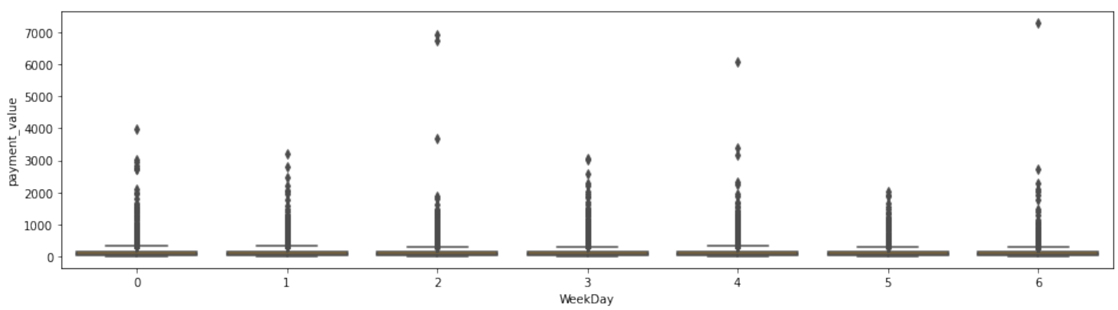
In [54]:
```
payment_orders.sort_values(by=['WeekDay'], inplace=True)
listPaymentT = ['credit_card','boleto','voucher','debit_card']
color = ['Blue','Orange','Green','Red']

for i,var in enumerate(listPaymentT):
    plt.figure(figsize=(16, 4))
    print(var)
    temp = payment_orders[(payment_orders['payment_type'] == var)][['WeekDay','paymen
```

```
    sns.boxplot(x='WeekDay',y='payment_value', data=temp ,color=color[i])
    plt.show()
```

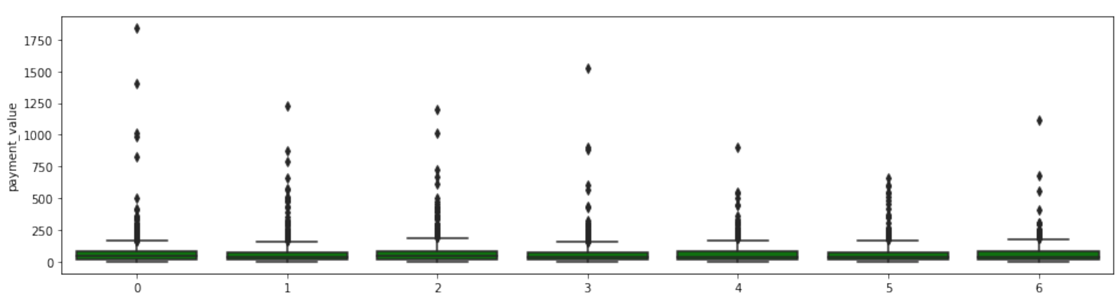
credit_card



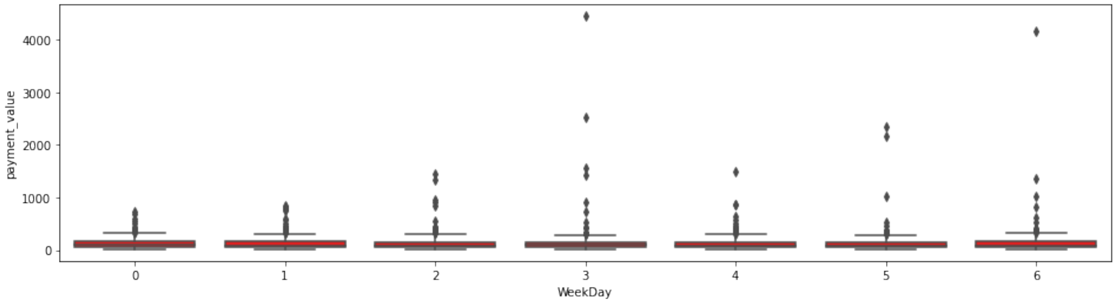

boleto




voucher

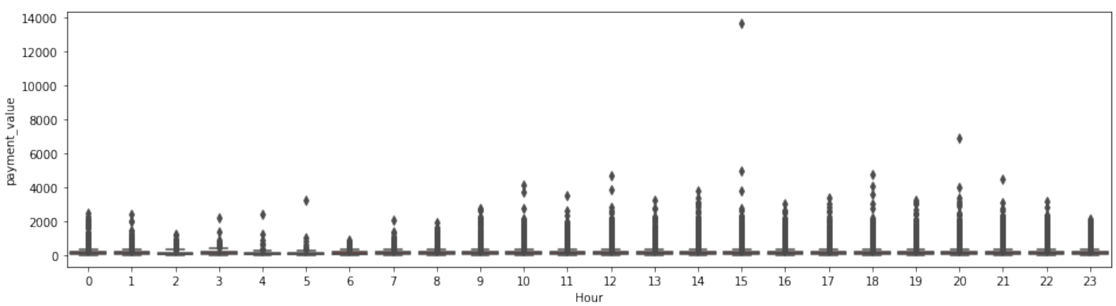


debit_card

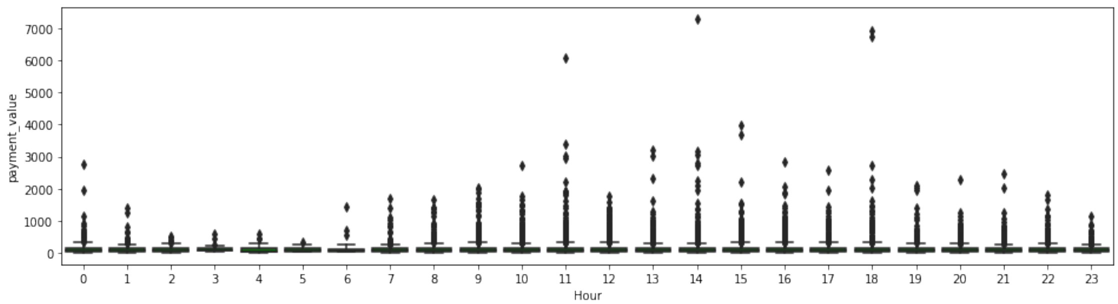



In [55]:
```
payment_orders.sort_values(by=['Hour'], inplace=True)
listPaymentT = ['credit_card','boleto','voucher','debit_card']
color = ['Red','Green','Blue','Orange']

for i,var in enumerate(listPaymentT):
    print(var)
    plt.figure(figsize=(16, 4))
    temp = payment_orders[(payment_orders['payment_type'] == var)][['Hour','payment_va
    sns.boxplot(x='Hour',y='payment_value', data=temp ,color=color[i])
    plt.show()
```

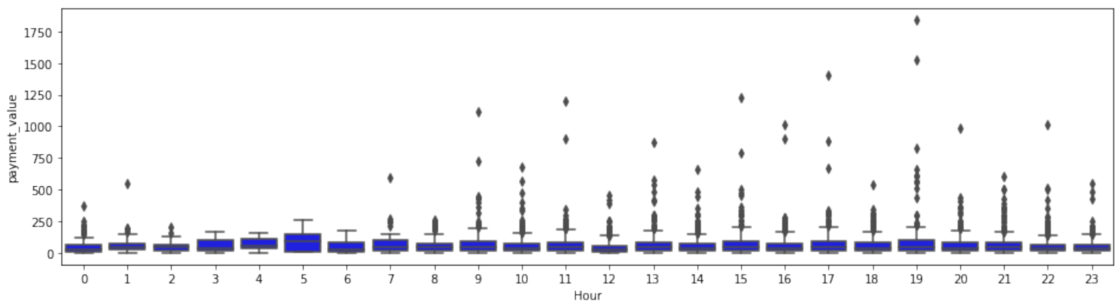
credit_card



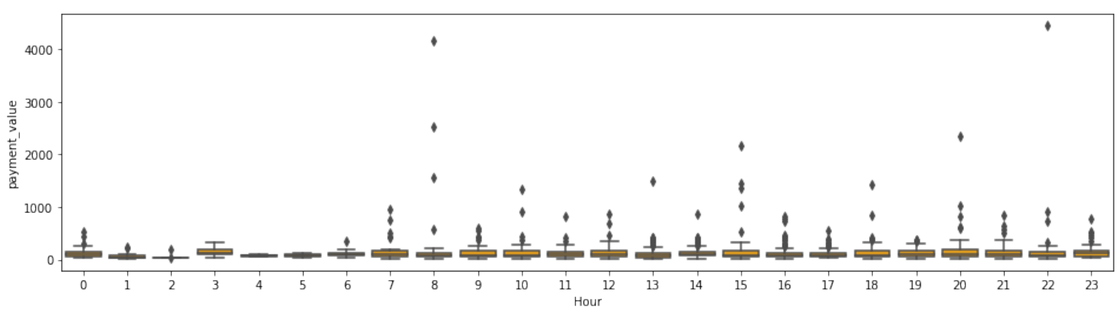

boleto




`voucher`




`debit_card`

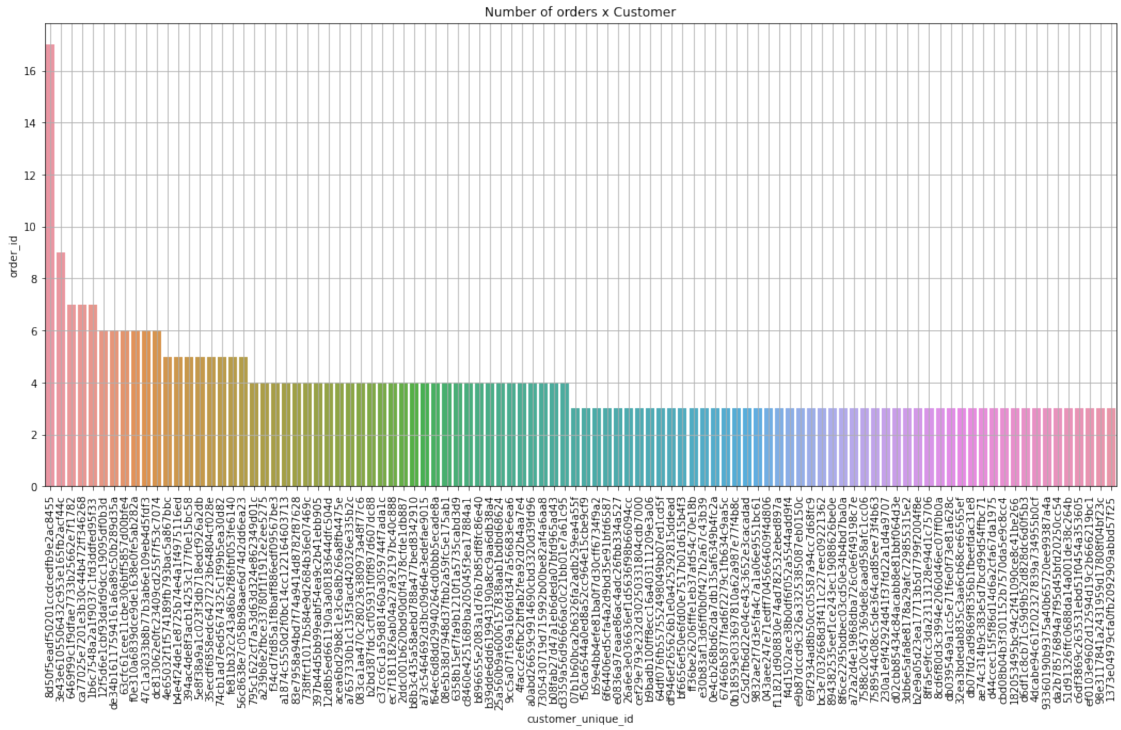



# 9 Customer purchase trends

## 9.1 Number of orders x Customer

The idea of this exploration of the data is to understand the number of orders made per customer, in order to understand the volume of purchases made by customers who buy the most.

```
In [57]: costumer_orders=pd.merge(customer, orders)
         aux=costumer_orders.groupby(['customer_unique_id']).agg({'order_id':'count'}).reset_i
         aux=aux.sort_values(by='order_id',ascending=False).head(100)
         plt.figure(figsize=(18, 8))
         ax=sns.barplot(x='customer_unique_id',y='order_id',data=aux);
         ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
         ax.set_title('Number of orders x Customer')
         ax.grid()
```

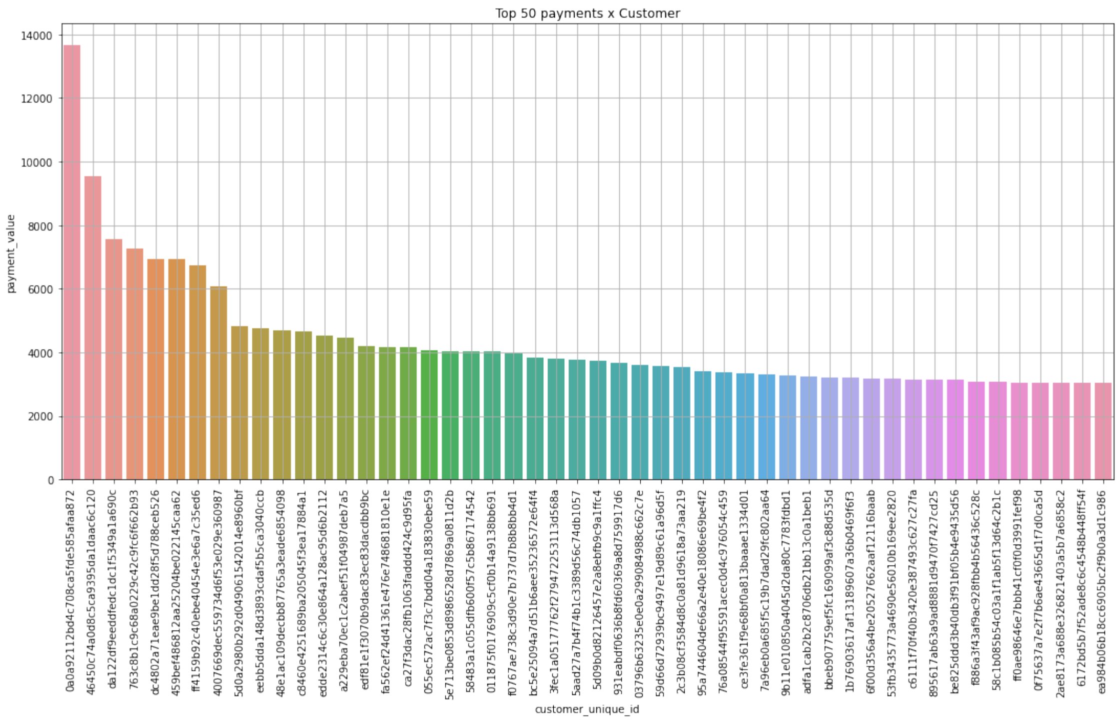



Looking at the top 50 of purchases, they have more than 4 purchases in the period, with a couple of exceptions that they make more.

### 9.2 Top Payments x customer

Regarding the amount of payments per customer, we can see that the top 50 customers have made purchases in the year for amounts close to US 3000, the average value of total purchases is US 154

```
In [58]: costumer_payments=pd.merge(costumer_orders, order_payments)
         aux=costumer_payments.groupby(['customer_unique_id']).agg({'payment_value':'sum'}).res
         aux=aux.sort_values(by='payment_value',ascending=False).head(50)
         plt.figure(figsize=(18, 8))
         ax=sns.barplot(x='customer_unique_id',y='payment_value',data=aux);
         ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
         ax.set_title('Top 50 payments x Customer')
         ax.grid()
         top_customer=aux
```




```
In [59]: costumer_payments.describe()
```

Out[59]:

| | customer_zip_code_prefix | payment_sequential | payment_installments |
|---|---|---|---|
| count | 103886.000000 | 103886.000000 | 103886.000000 |
| mean | 35072.550555 | 1.092679 | 2.853349 |
| std | 29743.491677 | 0.706584 | 2.687051 |
| min | 1003.000000 | 1.000000 | 0.000000 |
| 25% | 11366.250000 | 1.000000 | 1.000000 |
| 50% | 24360.000000 | 1.000000 | 1.000000 |
| 75% | 58418.000000 | 1.000000 | 4.000000 |
| max | 99990.000000 | 29.000000 | 24.000000 |

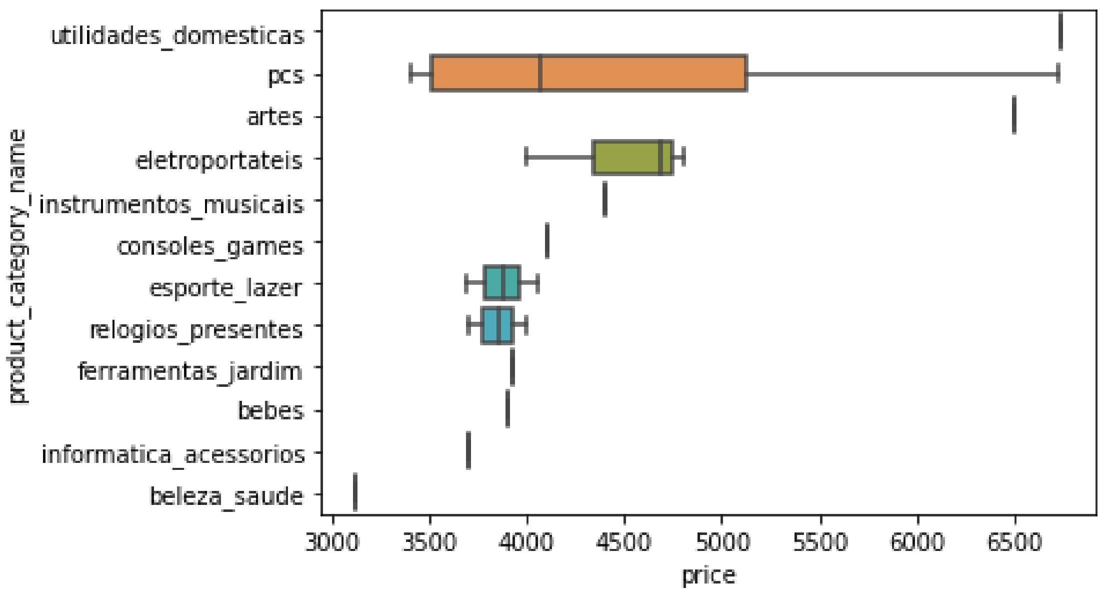
| | payment_value |
|---|---|
| count | 103886.000000 |
| mean | 154.100380 |
| std | 217.494064 |
| min | 0.000000 |
| 25% | 56.790000 |
| 50% | 100.000000 |
| 75% | 171.837500 |
| max | 13664.080000 |

## 9.3 Prices per product

Regarding the prices for the different product categories, we find that the most expensive products are related to the categories of Household utilities, Computers, Arts and electrical appliances. Within these, the highest price range refers to computers

```
In [60]: order_product=pd.merge(products, order_items)
         top_products= order_product.sort_values(['price'],ascending=False).head(20)
         sns.boxplot(y=top_products['product_category_name'], x=top_products["price"])
```

```
Out[60]: <AxesSubplot:xlabel='price', ylabel='product_category_name'>
```



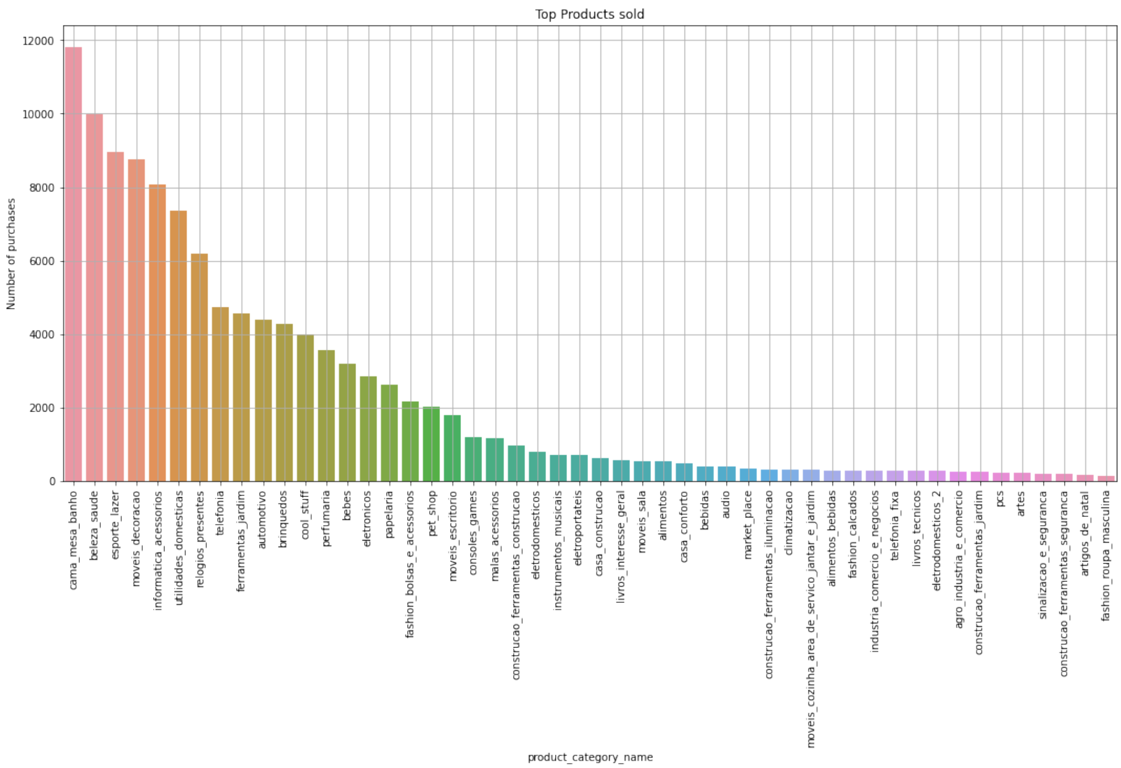

## 9.4 Top Products sold

Regarding the best-selling products, the following categories have the highest amount of sales:

| Product Category | Number of purchases |
|---|---|
| Cama_mesa_banho | 11.823 |
| beleza_saude | 9.972 |
| esporte_lazer | 8.945 |
| moveis_decoracao | 8744 |
| informatica_acessorios | 8082 |
| utilidades_domesticas | 7355 |

```
In [63]: client_product=pd.merge(costumer_payments, order_product)
         aux=client_product.groupby('product_category_name').size().to_frame().rename(columns=
         aux=aux.sort_values(by='Number of purchases',ascending=False).head(50)
```

```
plt.figure(figsize=(18, 8))
ax=sns.barplot(x='product_category_name',y='Number of purchases',data=aux);
ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
ax.set_title('Top Products sold')
ax.grid()
```




## 9.5 HYPOTHESIS FOR NEXT SUBMISSION

- 1 Standing in the graph presented in the secction “Difference between Delivered Time and Delivered Estimated Time vs Reviews”, we could afirm that puntuality in the delivery time is correlated in a inverse relationship for the review score when a costumer purchases a product online, and further insides in his decision of not buy in the same store again

- 2 According with analysis section of payment methods The Percentage of Credit Card payment of General Customer is equal to the Percentage of Credit Card payment for Customer of best seller products

- 3 The number of purchases by product category is associated with the region where the customer is located.

- 4 Would be equivalent in hypothesizing that the amount of purchasing is higher on weekends.

- 5 Would the buyers purchase more at nights