eCommerce Data Analysis - Profit Improvement Suggestion

Introduction

Background

- This data analysis project was introduced during the coding bootcamp.
- It was a half-guided module and the problem and the preconditions are presented.
- The insights, the visualizations and the suggestions are my original.

Dataset

- Brazilian E-Commerce Public Dataset by Olist
- kaggle.com/olistbr/brazilian-ecommerce
- Olist offers Logistic and Inventory Management Service to sellers
- Information about ~100k orders made between 2016 and 2018
- 8 csv files (orders, cutomers, reviews, sellers, products...etc)

Problem statement

How should Olist improve it's profit margin, given that the revenue and the cost are calculated as the following condition:

Revenue

1. Olist takes a 10% cut on the product price (excl. freight) of each order delivered.
2. Olist charges 80 BRL by month per seller.

Cost

1. Estimated cost occured by bad review per order

review_score	cost (BRL)
1 star	100
2 stars	50
3 stars	40
4 stars	0
5 stars	0

1. IT costss

Olist's IT costs are estimated to be proportional to *the square-root of the total cumlated number of orders approved*.

The IT department also told you that since the birth of the marketplace, cumulated IT costs have amounted to 500,000 BRL.

Suggestion

After the data analysis on seller data, I would give the following 2 suggestions to improve the profits by at least 10 %.

1. By removing the worst 15 sellers (0.5 % of the sellers) who make negative profits, Olist improves the profits by 10 %

2. By charging 10 % of the review cost directly to the sellers, Olist improves the profits by 13.9 %

EDA of seller dataset

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns
import math
from scipy import stats
import plotly.express as px
```

Original data is imported, cleaned and processed then according to 2.3, revenues and profits for each seller are calculated and the code is saved in seller_updated.py

In [2]:
```
from olist.seller_updated import Seller as SellerUpdated
sellers = SellerUpdated().get_training_data()
```

In [3]:
```
sellers.shape
```

Out[3]:
```
(2967, 18)
```

In [4]:
```
sellers.head()
```

Out[4]:

	seller_id	seller_city	seller_state	delay_to_carrier	wait_time	date_first_sale	date_last_sale	months_
0	3442f8959a84dea7ee197c632cb2df15	campinas	SP	0.000000	13.018588	2017-05-05 16:25:11	2017-08-30 12:50:19	
1	d1b65fc7debc3361ea86b5f14c68d2e2	mogi guacu	SP	0.000000	9.065716	2017-03-29 02:10:34	2018-06-06 20:15:21	
2	ce3ad9de960102d0677a81f5d0bb7b2d	rio de janeiro	RJ	0.000000	4.042292	2018-07-30 12:44:49	2018-07-30 12:44:49	
3	c0f3eea2e14555b6faeea3dd58c1b1c3	sao paulo	SP	0.000000	5.667187	2018-08-03 00:44:08	2018-08-03 00:44:08	
4	51a04a8a6bdcb23deccc82b0b80742cf	braganca paulista	SP	3.353727	35.314861	2017-11-14 12:15:25	2017-11-14 12:15:25	

In [5]:
```
sellers.info()
```

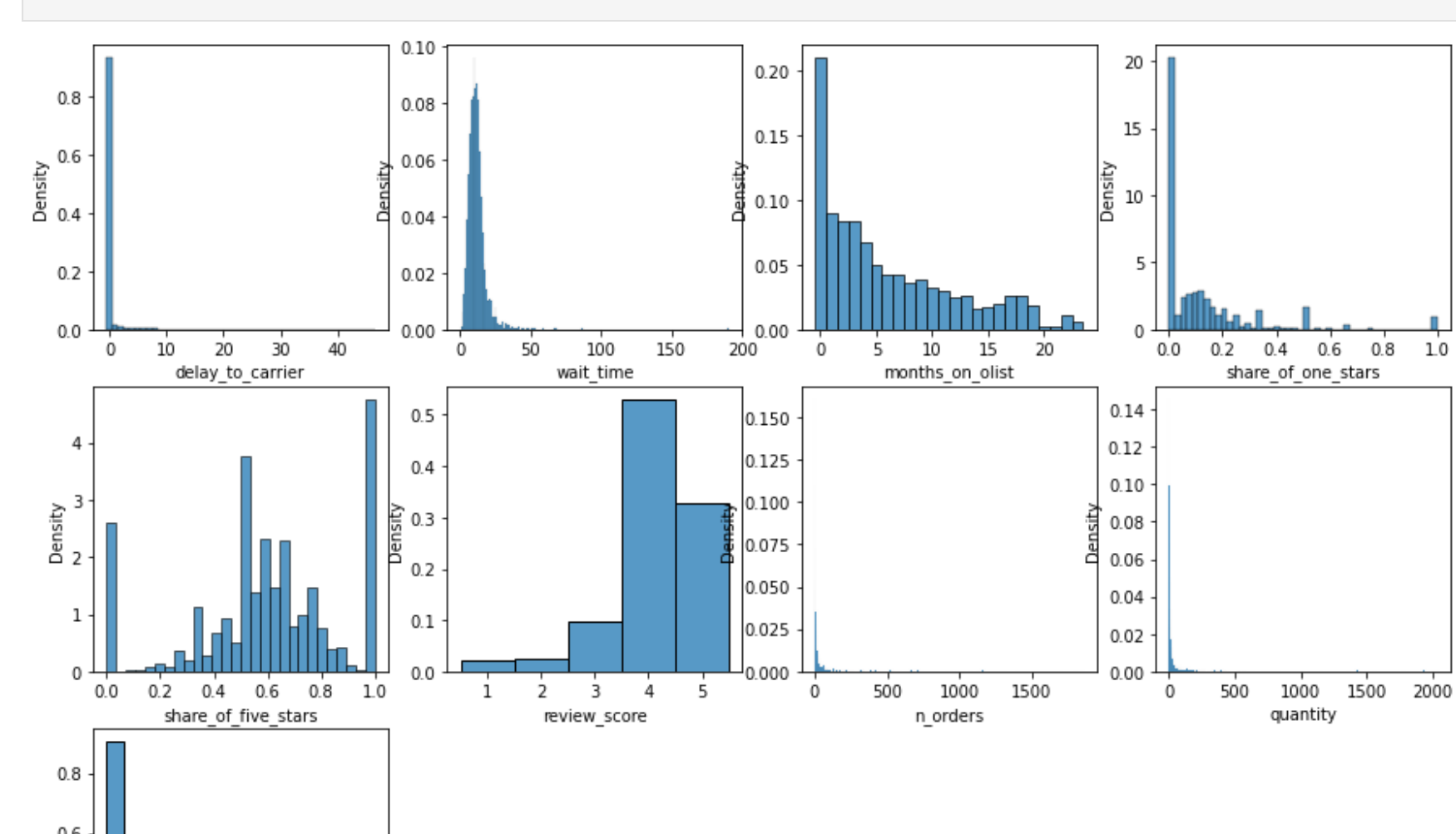
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2967 entries, 0 to 2966
Data columns (total 18 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   seller_id          2967 non-null   object
 1   seller_city        2967 non-null   object
 2   seller_state       2967 non-null   object
 3   delay_to_carrier   2967 non-null   float64
 4   wait_time          2967 non-null   float64
 5   date_first_sale    2967 non-null   datetime64[ns]
 6   date_last_sale     2967 non-null   datetime64[ns]
 7   months_on_olist    2967 non-null   float64
 8   share_of_one_stars 2967 non-null   float64
 9   share_of_five_stars 2967 non-null  float64
 10  review_score       2967 non-null   float64
 11  review_costs       2967 non-null   int64
 12  n_orders           2967 non-null   int64
 13  quantity           2967 non-null   int64
 14  quantity_per_order 2967 non-null   float64
 15  sales              2967 non-null   float64
 16  revenues           2967 non-null   float64
 17  profits            2967 non-null   float64
dtypes: datetime64[ns](2), float64(10), int64(3), object(3)
memory usage: 440.4+ KB
```

In [6]:
```
sellers.describe()
```

Out[6]:

	delay_to_carrier	wait_time	months_on_olist	share_of_one_stars	share_of_five_stars	review_score	review_costs	n_o
count	2967.000000	2967.000000	2967.000000	2967.000000	2967.000000	2967.000000	2967.000000	2967.00
mean	0.385636	12.139417	6.025615	0.120886	0.595077	4.101513	545.402764	33.65
std	2.295812	7.069233	5.994211	0.188857	0.278763	0.801517	1889.147815	107.18
min	0.000000	1.214178	0.000000	0.000000	0.000000	1.000000	0.000000	1.00
25%	0.000000	8.287658	1.000000	0.000000	0.493022	3.846154	0.000000	2.00
50%	0.000000	11.115143	4.000000	0.058824	0.600000	4.210526	100.000000	7.00
75%	0.000000	14.231984	10.000000	0.159043	0.750000	4.626453	365.000000	23.00
max	45.434039	189.863160	23.000000	1.000000	1.000000	5.000000	39400.000000	1854.00

In [7]:
```
sellers_eda = sellers.drop(['review_costs','sales', 'revenues', 'profits'], axis=1)
plt.figure(figsize=(15,11))
for (i, col) in enumerate(sellers_eda.describe().columns):#["wait_time", "delay_to_carrier", "avg_review_score
    plt.subplot(3,4,i+1)
    sns.histplot(sellers[col], kde=False, stat='density', discrete=[True,None][col in ['share_of_one_stars','s
```

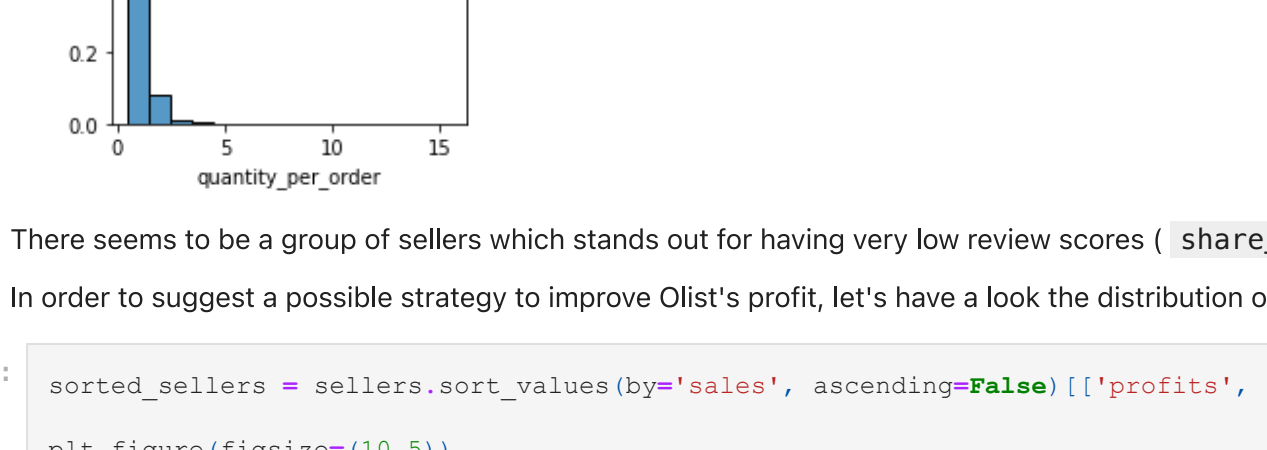


There seems to be a group of sellers which stands out for having very low review scores (`share_of_one_stars` > 0.5).

In order to suggest a possible strategy to improve Olist's profit, let's have a look the distribution of the sales and the profits.

In [8]:
```
sorted_sellers = sellers.sort_values(by='sales', ascending=False)[['profits', 'n_orders', 'sales','revenues']]

plt.figure(figsize=(10,5))
sorted_sellers.sales.plot()
plt.title('Total Sales per Seller', fontsize=13)
plt.xlabel('Seller ID', fontsize=13)
plt.grid(True);
```

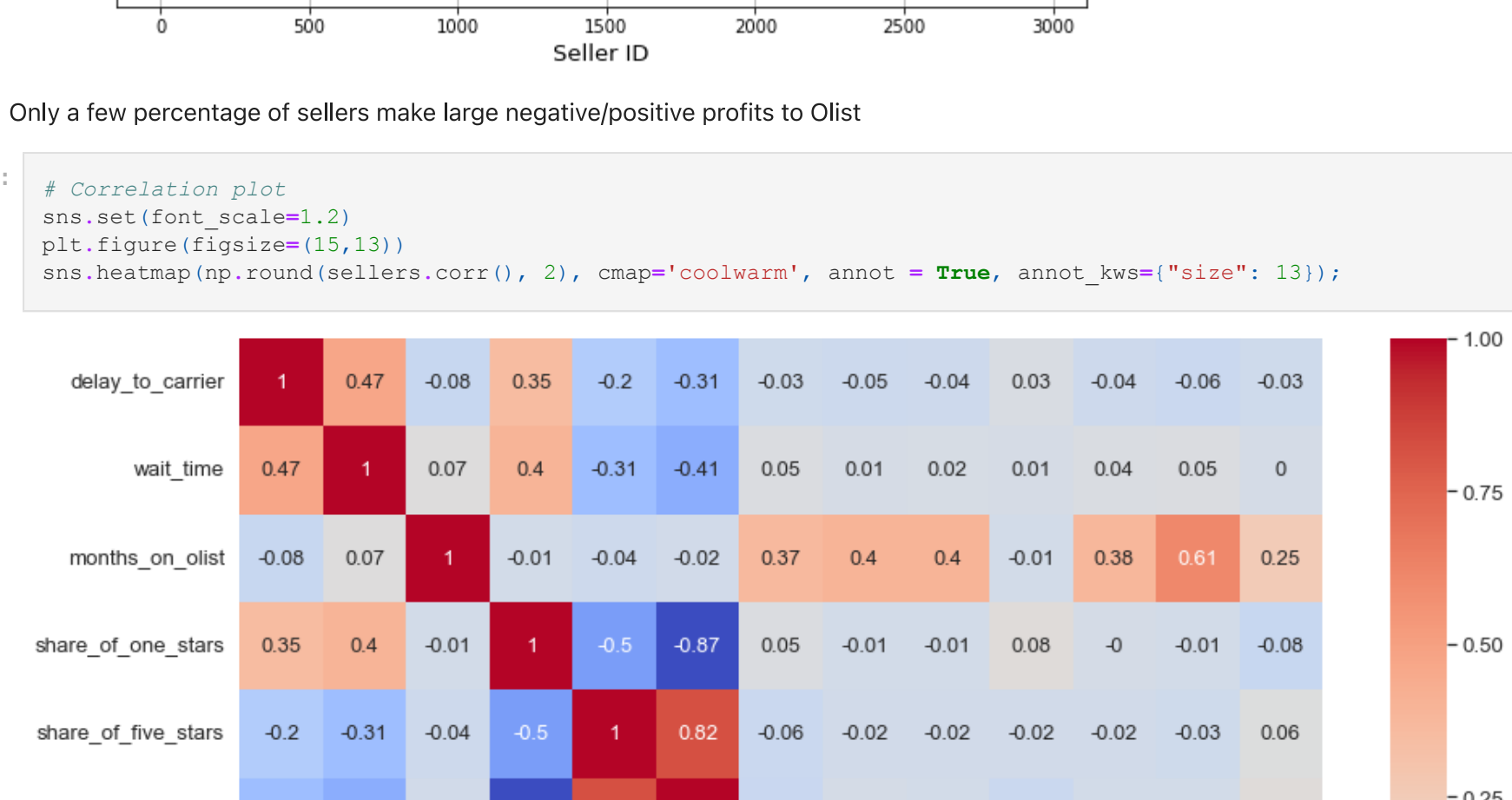


About 100 sellers are making huge sales for them

In [9]:
```
sorted_sellers = sellers.sort_values(by='profits')[['profits', 'n_orders','revenues']].reset_index()
plt.figure(figsize=(10,5))
sorted_sellers.profits.plot()
plt.title('Gross Profit per Seller', fontsize=13)
plt.xlabel('Seller ID', fontsize=13)
plt.grid(True);
```

Only a few percentage of sellers make large negative/positive profits to Olist

In [10]:
```
# Correlation plot
sns.set(font_scale=1.2)
plt.figure(figsize=(15,13))
sns.heatmap(np.round(sellers.corr(), 2), cmap='coolwarm', annot = True, annot_kws={"size": 13});
```



Interesting correlations:

1. Positive correlation between `waiting_time` and `delay_to_carrier`

- `delay_to_carrier` is the number of days when the seller provide the item to the carrier later than expected
- It is logical to think that the main reason of `waiting_time` is `delay_to_carrier`.

1. Negative correlation between `profits` and `n_orders`

- The seller who has large number of orders tend to make less profit
- The large number of order also incur large IT cost (which is not included here yet)

1. Little correlation between `profits` and `review_score`

- The sellers with bad review can still give profits to Olist

Calculate IT cost

In addition, Olist's IT costs (servers, etc...) increase with the amount of orders processed, but less and less rapidly (scale effects).

For the sake of simplicity, Olist's total cumulated IT Costs is considered to be proportional to the square-root of the total cumulated number of orders approved.

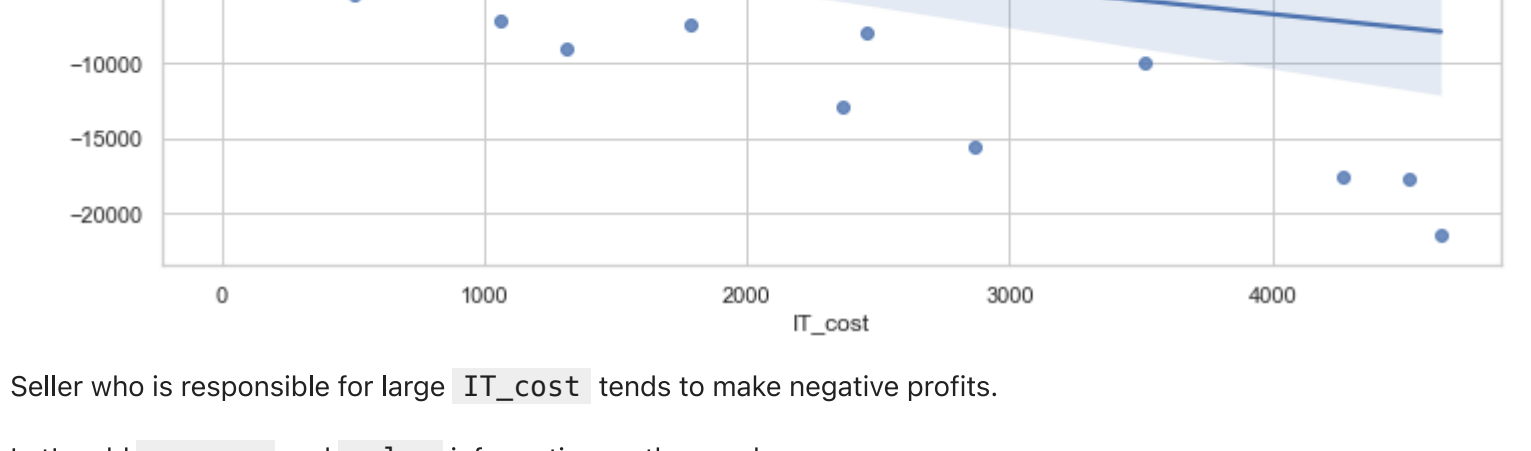
Let's assume that the IT department told me that since the birth of the marketplace, cumulated IT costs have amounted to 500,000 BRL.

In [11]:
```
# y: Cumlated IT cost, a: coefficient
# y = sqrt(cumlated_orders) * a = 500,000
cumlated_orders = sorted_sellers['n_orders'].sum()
a = 500000/math.sqrt(cumlated_orders)
# a = 1582.39
# z: Current IT cost per order  = 2.5 BRL
z = math.sqrt(cumlated_orders + 1) * a - 500000
z
```

Out[11]:
```
2.503975060360972
```

In [12]:
```
# Calculate IT_cost for each seller
sellers['IT_cost'] = sellers['n_orders'] * z
```

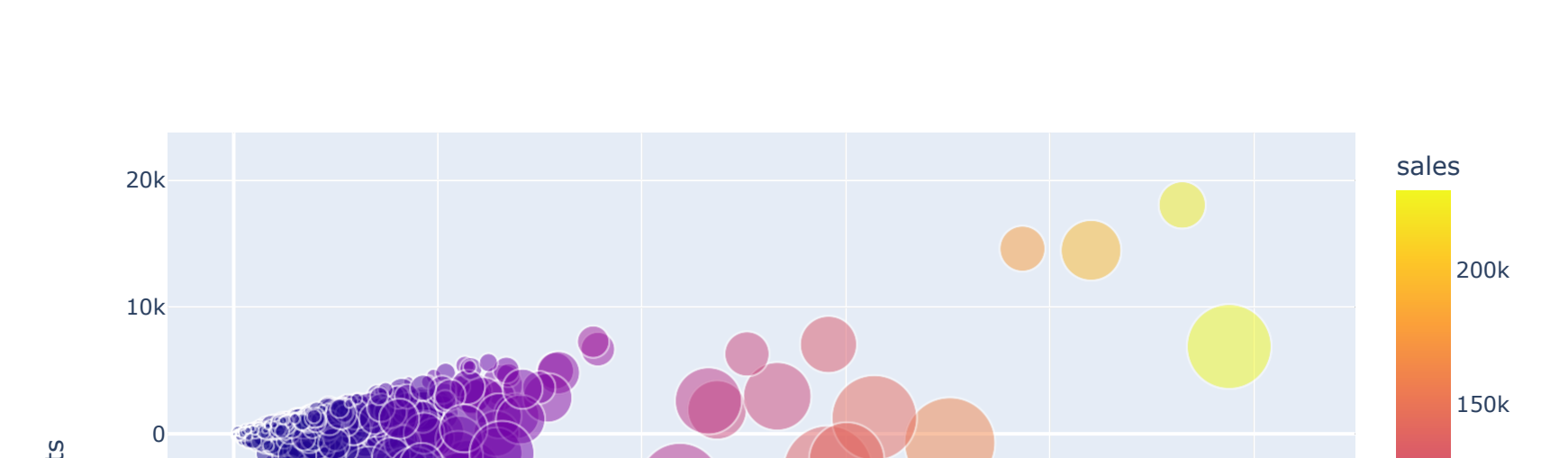
In [13]:
```
# Distribution of IT_cost and profits
sns.set_theme(style='whitegrid')
fig, ax = plt.subplots(figsize=(12,6))
slope, intercept, r_value, pv, se = stats.linregress(sellers.IT_cost, sellers.profits)
sns.regplot(x='IT_cost', y='profits', data=sellers, line_kws={'label':'$y=%3.7s*x+%3.7s$'%(slope, intercept)})
plt.legend();
```



Seller who is responsible for large `IT_cost` tends to make negative profits.

Let's add `revenue` and `sales` information on the graph.

In [14]:
```
fig = px.scatter(data_frame = sellers, x="revenues", y="profits", size="IT_cost", color="sales", size_max = 60
fig.show()
```



High revenue sellers do not always give positive profits to Olist

What makes seller unprofitable?

Does `IT_cost` make seller unprofitable (for Olist)?

In [15]:
```
worst_sellers = np.round(sellers[['profits', 'revenues','review_costs', 'IT_cost', 'review_score']].sort_value
worst_sellers.head(7)
```

Out[15]:

	profits	revenues	review_costs	IT_cost	review_score
0	-21519.52	13770.48	35290	4642.37	3.94
1	-17752.71	21647.29	39400	4522.18	3.83
2	-17581.16	11948.84	29530	4271.78	4.08
3	-15542.25	4677.75	20220	2869.56	4.00
4	-12886.43	2553.57	15440	2361.25	4.07
5	-9926.08	12053.92	21980	3515.58	4.13
6	-9017.61	3562.39	12580	1312.08	3.75

- The 5 worst sellers make more than 10,000 BRL loss to Olist
- `review_costs` is the main reason of making loss to Olist rather than `IT_cost`, although it is interesting that the average `review_score` of the sellers are good (around 4.0)

In [16]:
```
#Create a chart with revenue, costs and profits
sorted_sellers = sellers.sort_values(by='profits')[['profits', 'review_costs','n_orders','revenues', 'IT_cost'
plt.figure(figsize=(12,8))
sorted_sellers.profits[:100].plot(linewidth=6)
(sorted_sellers.review_costs * - 1.0)[:100].plot(linewidth=1, marker='o')
(sorted_sellers.IT_cost * - 1.0)[:100].plot(linewidth=1, marker='o')
(sorted_sellers.revenues )[:100].plot(linewidth=1, marker='o')
plt.title('gross profit and review costs per seller (non-profitable seller only)', fontsize=13)
plt.xlabel('Seller ID', fontsize=13)
plt.ylabel('Profits (BRL)', fontsize=13)
plt.legend();
```

`review_costs` is the strong contributor to the negative `profits`.

Suggestion for profit improvement

Dropping the non-profitable sellers

We can drop the non-profitable sellers who make less than the threshold from the market.

In [17]:
```
def profit_change(thresh):
    """
    Take threshold as parameter to cut sellers depending on their negative profits and reutrn
    the relative profit increase, percentage of removed sellers and percentage of removed orders
    """
    #Calculate the increase of profit by removing the sellers
    good_sellers = sorted_sellers[sorted_sellers['profits'] > thresh]
    abs_increase = good_sellers['profits'].sum() - sorted_sellers['profits'].sum()
    rel_increase = (abs_increase) * 100 / sorted_sellers['profits'].sum()
    #Calculate the percentage of the sellers to be removed
    sellers_count = good_sellers['index'].count()
    rem_sellers_pct = 100 - sellers_count * 100 / sorted_sellers['index'].count()
    #Calculate the percentage of the orders to be removed
    orders_count = good_sellers['n_orders'].sum()
    rem_orders_pct = 100 - orders_count * 100 / sorted_sellers['n_orders'].sum()
    return rel_increase, rem_sellers_pct, rem_orders_pct
```

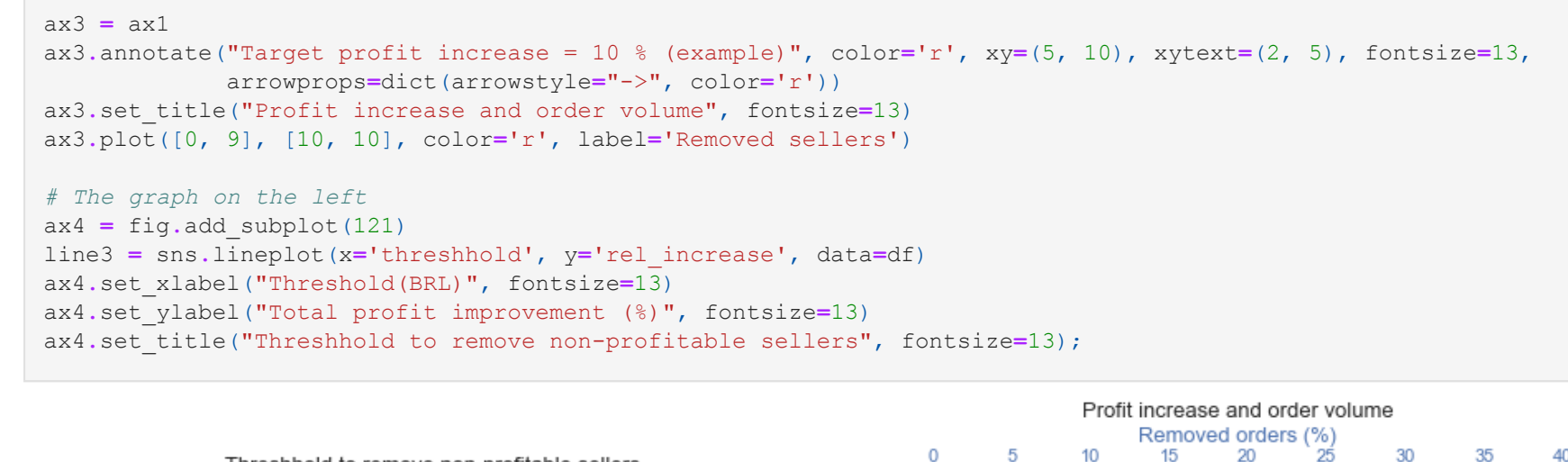
In [18]:
```
#Make array based on the graphs
df = pd.DataFrame([[th, *profit_change(th)] for th in range(-22000, 0, 50)])
df.columns = ['threshhold', 'rel_increase', 'rem_sellers_pct', 'rem_orders_pct']

#The graph on the right
fig = plt.figure(figsize=(16,6))
ax1 = fig.add_subplot(122)
line1 = sns.lineplot(x='rem_sellers_pct', y='rel_increase', data=df, color='darkgreen', label='Removed sellers
ax1.set_ylabel("Total profit improvement (%)", fontsize=13)
ax1.set_xlabel("Removed sellers (%)", color='darkgreen', fontsize=13)
ax1.tick_params(axis="x", colors='darkgreen')
ax1.set_xticks([1, 2, 3, 4, 5, 6, 7, 8, 9])
top_line = mpatches.Patch(color='g', label='Removed sellers')

ax2 = ax1.twiny()
line2 = sns.lineplot(x='rem_orders_pct', y='rel_increase', data=df, color='b',legend=False)
ax2.set_xlabel("Removed orders (%)", color='b', fontsize=13)
ax2.tick_params(axis="x", colors='b')
bottom_line = mpatches.Patch(color='b', label='Removed orders')
plt.legend(handles=[top_line, bottom_line])

ax3 = ax1
ax3.annotate("Target profit increase = 10 % (example)", color='r', xy=(5, 10), xytext=(2, 5), fontsize=13,
             arrowprops=dict(arrowstyle="->", color='r'))
ax3.set_title("Profit increase and order volume", fontsize=13)
ax3.plot([0, 9], [10, 10], color='r', label='Removed sellers')

# The graph on the left
ax4 = fig.add_subplot(121)
line3 = sns.lineplot(x='threshhold', y='rel_increase', data=df)
ax4.set_xlabel("Threshold(BRL)", fontsize=13)
ax4.set_ylabel("Total profit improvement (%)", fontsize=13)
ax4.set_title("Threshhold to remove non-profitable sellers", fontsize=13);
```



>> Removing the worst 0.5 % of the sellers (15 sellers) increases 10% of Olist's profits

As an alternative, it would also make sense to introduce the new fee to sellers when they receive bad reviews, since the large negative `profits` were due to `review_costs`.

Introducing bad review charge

What if we charge a partial cost for bad reviews to the sellers, depending on the cost to Olist by percentage?

An example of **bad review charge** per order with different rates. The numbers of the unit is BRL.

review_score	cost to Olist	5 % charge	10 % charge	15 % charge	20 % charge
1 star	100	5	10	15	20
2 stars	50	2.5	5	7.5	10
3 stars	40	2	4	6	8
4 stars	0	0	0	0	0
5 stars	0	0	0	0	0

What is the improvement on the profits based on the different charging rate (0 - 20 %)?

In [19]:
```
# Current total profits
sellers.profits.sum()
```

Out[19]:
```
1167608.4440000001
```

In [20]:
```
# Create DataFrame for plot
charge_df = pd.DataFrame()
charge_df['charge_rate'] = np.linspace(0, 0.2, num=5)
charge_df['charge_rate_pct'] = (charge_df.charge_rate * 100).astype('int64')
#charge_df['charge_rate_pct'].astype('int64')
charge_df['review_charge'] = [sellers.review_costs.sum() * x for x in charge_df.charge_rate]
charge_df['new_profits'] = sellers.profits.sum() + charge_df.review_charge
charge_df['profits_imp_pct'] = (charge_df.new_profits /sellers.profits.sum() - 1 ) * 100
charge_df['current_profits'] = sellers.profits.sum()
l = ['charge_rate_pct', 'new_profits', 'profits_imp_pct']
np.round(charge_df[l].rename({l[0]:'Bad review charge(%)',l[1]:'Improved profits(BRL)',l[2]:'Total profit impr
```

Out[20]:

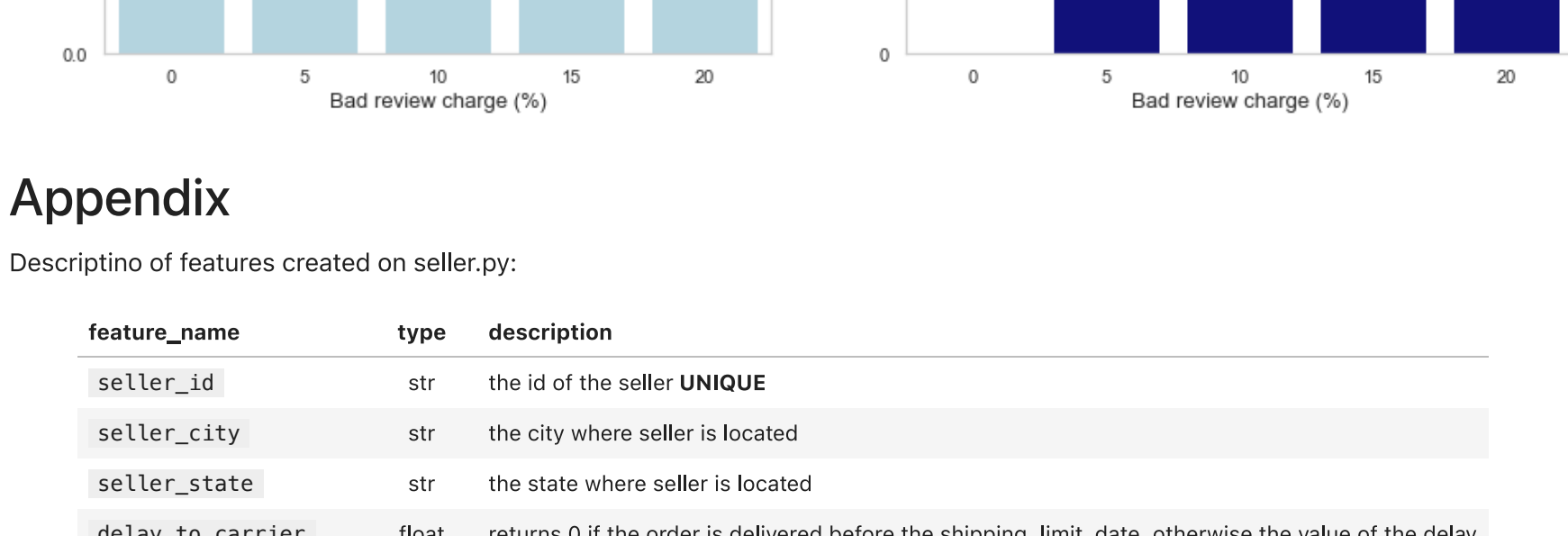
	Bad review charge(%)	Improved profits(BRL)	Total profit improvement(%)
0	0	1167608.4	0.0
1	5	1248518.9	6.9
2	10	1329429.4	13.9
3	15	1410339.9	20.8
4	20	1491250.4	27.7

By charging 10 % of the review cost to the sellers, Olist improves the profits by 13.9 %

In [21]:
```
# Visualize the profit improvements with different charge rate
charge_df['new_profits_mil'] = charge_df['new_profits'] / 1000_000
charge_df['current_profits_mil'] = charge_df['current_profits'] / 1000_000

fig = plt.figure(figsize=(16,6))
ax1 = fig.add_subplot(121)
bar1 = sns.barplot(x='charge_rate_pct', y='new_profits_mil', data=charge_df, color='darkblue', ax = ax1)
bar2 = sns.barplot(x='charge_rate_pct', y='current_profits_mil', data=charge_df, color='lightblue', ax = ax1)
top_bar = mpatches.Patch(color='darkblue', label='Profit Improvement')
bottom_bar = mpatches.Patch(color='lightblue', label='Current Profits')
plt.legend(handles=[top_bar, bottom_bar])
ax1.set_xlabel("Bad review charge (%)", fontsize=13)
ax1.set_ylabel("Total profit (mil. BRL)", fontsize=13)
ax1.set_title("Bad reciew charge and total profit", fontsize=13)

ax2 = fig.add_subplot(122)
sns.barplot(x='charge_rate_pct', y='profits_imp_pct', data=charge_df, color='darkblue', ax = ax2)
ax2.set_xlabel("Bad review charge (%)", fontsize=13)
ax2.set_ylabel("Total profit improvement (%)", fontsize=13)
ax2.set_title("Bad review charge and profit improvement", fontsize=13);
```



Appendix

Descriptino of features created on seller.py:

feature_name	type	description
`seller_id`	str	the id of the seller **UNIQUE**
`seller_city`	str	the city where seller is located
`seller_state`	str	the state where seller is located
`delay_to_carrier`	float	returns 0 if the order is delivered before the shipping_limit_date, otherwise the value of the delay
`wait_time`	float	average wait_time (duration of deliveries) per seller
`date_first_sale`	datetime	date of the first sale on Olist
`date_last_sale`	datetime	date of the last sale on Olist
`months_on_olist`	float	round number of months on Olist
`share_of_five_stars`	float	share of five-star reviews for orders in which the seller was involved
`share_of_one_stars`	float	share of one-star reviews for orders in which the seller was involved
`review_score`	float	average review score for orders in which the seller was involved
`n_orders`	int	number of unique orders the seller was involved with
`quantity`	int	total number of items sold by this seller
`quantity_per_order`	float	average number of items per order for this seller
`sales`	float	total sales associated with this seller (excluding freight value) in BRL