sunscrapers

HOW TO INCREASE SALES ON ONLINE MARKETPLACES WITH DATA SCIENCE

build a data-driven sales strategy
to gain a competitive advantage

A DATA SCIENCE CASE STUDY

CONTENTS

INTRODUCTION

Does your business sell products on an online marketplace like Amazon or eBay?

Online marketplaces are full of sellers competing with you at every step. It's not easy to increase sales when other people offer products that are very similar to yours.

What makes buyers choose one seller over another if they all offer pretty much the same thing?

You might be thinking that it's all down to the price. So did I, but then I couldn't resist the temptation to see whether there are other factors that may influence consumers.

In this whitepaper, I step into the shoes of an online seller to show you that a data-driven approach using Python programming tools can help drive sales and take your business to the next level.

Let's begin!

AUTHOR

Filip Ciesielski

Filip is a Python developer and data scientist at Sunscrapers. He's passionate about the full-stack software design process, but mainly focuses on backend technologies. His professional background in biophysics inspires him to ask difficult questions and solve puzzling problems.

PLANNING
OUR RESEARCH

What makes some sellers more successful than others if they all sell pretty much the same products?

The idea for this study came from a simple question:

What makes some sellers more successful than others if they all sell pretty much the same products?

To get some answers, I decided to carry out a research study.

First, I needed to find an online marketplace that is rich in content about sales history. It's critical to pick one that offers some kind of indicator of sales performance to enable us to evaluate it against the offer features. I chose one of the most popular global online marketplaces as it provides ample information about products and their 100 most recent purchases.

Once I picked my data source, I had to come up with a method that would allow me to compare different offers.

The trick was finding a range of very similar products sold by a large number of sellers who used different sales strategies.

A quick Google search revealed that some of the most commonly sold products on that marketplace were **smartphone screen protectors**.

And that's the product I decided to investigate.

TOOLS
OF THE TRADE

Python is my programming language of choice for data science.

Also, I chose it because it comes with many useful libraries for the job. Among them is Scrapy, which offers convenient features such as marshalling and preprocessing of field values, and I can run it via an online scrapinghub platform. You can use other libraries for this task as well (such as `requests` in combination with `beautifulsoup`, etc.).

I used `pandas` to load data into tables, clean it, process it, and analyze it. Finally, I visualized the findings using `seaborn` and `matplotlib`. All the work is completed with the help of Jupyter Notebook.

[1] We cannot use the brand name of the online marketplace in compliance with legal regulations.

GETTING
THE DATA

The website structure of the chosen marketplace includes information of interest spread among multiple levels.

Before moving on to the procedure, note that the web crawler code content is rather extensive and will vary for different marketplace websites. That's why I chose not to include any examples in this article.

Instead, I will summarize the process as concisely as possible. Scrapy, the Python framework for crawling websites I use here offers rich documentation and features some easy-to-follow tutorials, so I encourage you to refer to them if needed.

First, I manually searched for smartphone screen protectors on the online marketplace and started the crawling process from there. Typically, a search pattern starts with some kind of an offer list where each listing points to an item-dedicated page with more information, possibly even a list of past purchases.

Note that you can usually harvest valuable information about the product's presentation already in the search results list (like the seller's status, points, no. of past purchases). After all, this is the customer's first exposure to the offer and it may impact their decision-making process.

After the crawl, I ended up with two Pandas tables. The main table (`df`) had all of the product information, with each row corresponding to an item. The other table (`sale_history`), stored information about the past purchases - with each product including many rows of individual sale events data. I'll show the table examples later on.

PROCESSING
THE DATA

After the data extraction step, it was time to do some cleaning and data preparation.

Besides all the usual steps (removing nulls, casting columns into the right data types, etc.) , there were a couple of interesting steps that I'd like to mention here - again, without going into the details.

As a first step, I tend to go through individual columns with the Pandas `unique()` method. That's how I can see whether the values are consistent and sensible - and catch any potential issues. Then I check for data duplication by grouping rows by the column that serves as the unique identifier for a specific item - in this case I used `product_id`.

One of the first things I noticed is that some product pages were linked to multiple listings in the search results page (coincidence? I don't think so!). I got rid of the duplicates, but decided to keep this information for analysis. So I created a new column with the number of listings per item first, then deleted the copies except for one:

```
df['same_offer_count'] = df.groupby('product_id')['product_id'].transform('count')
df = df.drop_duplicates(subset='product_id', keep='first')
```

Another interesting problem was dealing with multiple currencies used throughout the marketplace. My raw data table contained bare price string values along with the quoted currency symbol (for example, 'US $1.09' or 'C $2.42'), so I needed to extract the numeric values and unify all the price currencies by converting them to USD. Here are a few example rows before the transformation:

	product_id	item_model	current_price	...
8	142887872836	For Samsung Galaxy S8	US $5.69	...
9	152955791986	For Samsung Galaxy S9 Plus	US $1.59	...
10	253143606985	Galaxy S8	C $0.99	...
11	292646641735	For Samsung Galaxy S9	US $1.48	...

Here is the code I used to transform it:

```
import re

from currency_converter import CurrencyConverter

cc = CurrencyConverter()
currency_shortcuts = {'C':'CAD', 'US':'USD', 'AU':'AUD'}  #  first I checked only these occur...
regx_str=r'(\w+\s*)\$[ ]?(\d+[.|,]?\d+)' # note the two 're' groups!

df[['currency', 'quoted_price']] = df['current_price'].str.extract(pat=regx_str)
df['currency'] = df['currency'].str.replace(' ', '')
df['currency'] = df['currency'].map(currency_shortcuts)
```

```
    df['price_USD'] = df['quoted_price'].copy()

    for currency in [ c for c in df['currency'].unique() if c not in ['USD']]:
        fltr = df['currency'].isin([currency])
        df.loc[fltr, 'price_USD'] = df.loc[fltr, 'quoted_price']\
            .apply(lambda x: cc.convert(x, currency, 'USD'))
```

Which resulted in:

	product_id	item_model	current_price	currency	quoted_price	price_USD	...
8	142887872836	For Samsung Galaxy S8	US $5.69	USD	5.69	5.69	...
9	152955791986	For Samsung Galaxy S9 Plus	US $1.59	USD	1.59	1.59	...
10	253143606985	Galaxy S8	C $0.99	CAD	0.99	0.76	...
11	292646641735	For Samsung Galaxy S9	US $1.48	USD	1.48	1.48	...

Next, I processed the sales history table (`sale_history`). I performed some basic type fixes, extracted and converted prices and currencies, and filled in the null values (code not shown). I ended up with this table (again, it's just a snapshot of rows):

	product_variant	purchase_date	quantity_sold	sell_price	brand	product_id	total_price
0	Model: For Samsung Galaxy S8+Style: Soft PET F...	2018-08-01	1	1.26	Unbranded	112410221730	1.26
0	Fit: For Samsung Galaxy S8Color: Black	2018-10-24	1	2.31	ELEGANT CHOISE	112500407359	2.31
1	Fit: For Samsung Galaxy S8 PlusColor: Black	2018-10-24	1	2.85	ELEGANT CHOISE	112500407359	2.85
2	Fit: For Samsung Galaxy S8Color: Clear TPU Case	2018-10-18	1	0.76	ELEGANT CHOISE	112500407359	0.76
3	Fit: For Samsung Galaxy S8Color: Clear	2018-10-17	1	2.31	ELEGANT CHOISE	112500407359	2.31

To make it useful for analysis and plotting let's aggregate the entries by date (and the `product_id`, of course) and calculate the number of sold items and daily selling rates. Wrapping all this into a single function allows applying it row-wise to the data frame:

```
    def calculate_sale_history_stats(df):
        """Calculates statistics on sale history, returns new dataframe"""

        delta = df['purchase_date'].max() - df['purchase_date'].min()
        days = int(delta.days)
        values = list(df['quantity_sold'])
        earnings = list(df['total_price'])
        sold_count = sum(values)

        if len(values) < days:
            values.extend([0]*(len(values) - days))
            earnings.extend([0]*(len(earnings) - days))

        res = pd.Series(
            [sold_count, np.mean(values), np.std(values), np.mean(earnings), np.std(earnings)],
            index=['Sold_count',
                   'Mean_daily_sold_count', 'Sold_count_St.Dev',
                   'Daily_earnings', 'Daily_earnings_St.Dev'])
        return round(res, 2)
```

And by applying it to our `sale_history` dataframe:

```
    sale_history_stats = sale_history.groupby('brand').apply(calculate_sale_history_stats)
```

resulted in:

brand	Sold_count	Mean_daily_sold_count	Mean_Sold_count_St.Dev	Daily_earnings	Daily_earnings_St.Dev
4D GlassBD	26.0	0.32	0.81	0.50	1.26
9H	4.0	0.14	0.34	0.95	2.38
AO	3.0	1.50	0.50	14.92	4.97
Alcase	5.0	0.15	0.50	0.26	0.86
Baseus	2.0	0.04	0.20	0.47	2.28

Finally, I merged the aggregated sales stats (`sale_history_stats`) into the main `df` table:

```
df = pd.merge(
    how='left',
    on='product_id',
    left=aggreg_sale_history,
    right=df[['product_id','shipping_cost', 'shipping_from', 'top_rating_badge',
              'seller_feedback_score', 'seller_feedback_perc',]]
)
```

Here's the resulting `df` table (again, only a selection of columns is shown):

	item_model	product_id	brand	shipping_from	seller_feedback_perc	seller_feedback_score	currency	price_USD	shipping_cost	top_rating_badge	discount_shown
20	Galaxy S7 edge	132747469991	Unbranded/Generic	China	97.8	346707.0	USD	0.99	0.0	False	False
21	2X S8 Tempered Glass	142464440056	Unbranded/Generic	United States	99.0	42618.0	USD	7.94	3.0	True	False
22	Samsung Galaxy S7 edge	401431247278	Unbranded/Generic	China	97.8	346707.0	CAD	0.76	0.0	False	False
23	Clear	362388460603	Unbranded/Generic	China	98.5	78495.0	USD	1.68	0.0	False	True
24	Samsung Galaxy S8 (2 Pack)	401425908602	Spigen	China	97.8	346707.0	CAD	0.76	0.0	False	False

CRAFTING A SALES STRATEGY

Before we delve into the intricacies of marketing our product and stealing customers from our competitors, we need to research the battlefield.

Having a realistic picture of the situation 'out there' helps to build solid foundations for a sales strategy.

Let's say that we want to sell smartphone screen protectors in an online marketplace.

What are we up against? Which brands should we sell? And how much should we charge for our product to turn a high profit?

Let's begin our analysis.

WHAT ARE THE AVERAGE PRICES?

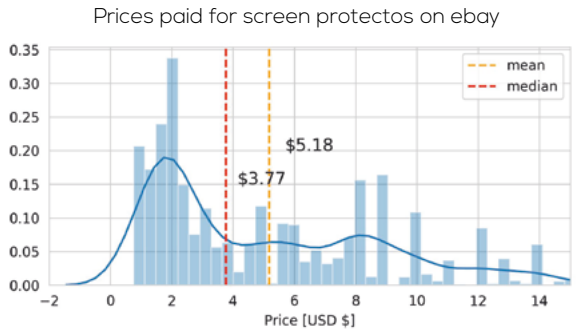
First things first, let's see how much profit we can make from screen protectors in general.

How much do sellers charge for them?

We can analyze the prices in our marketplace and see what people usually pay for smartphone screen protectors:

```
import matplotlib.pyplot as plt
import seaborn as sns

prices = df['price_USD']
mean, std, median = prices.mean(), prices.std(), prices.median()
sns.distplot(df['price_USD'], ax=ax, bins=200)

paid_prices = sale_history['sell_price']
paid_mean, paid_std, paid_median = paid_prices.mean(), paid_prices.std(), paid_prices.median()
sns.distplot(paid_prices, ax=ax, bins=100)
```

And here are the two resulting histograms:



As you can see, the majority of products appears to cost about 1.15$ (average ~3.9$). However, it seems that customers often prefer to chip in a few extra bucks to their purchase (average of ~5%, median ~3.8$). **'The cheaper, the better' rule doesn't work here.**

If we take that insight into account, we may assume that choosing a price that oscillates around 4$ is a good strategy. But let's have a closer look at other factors - surprises may still lurk around the corner!

HOW DOES PRICING AFFECT SELLABILITY?

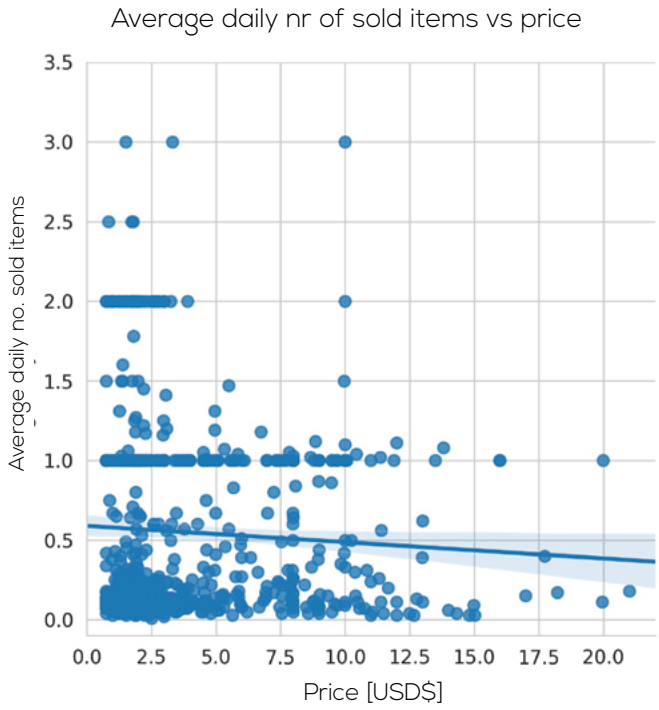
The price is probably the most essential factor in a customer's decision-making process when making a purchase.

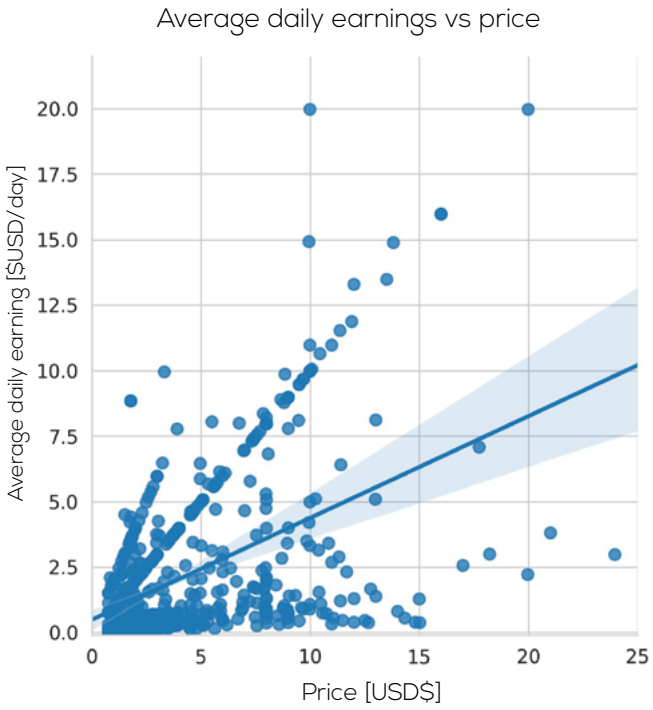
Sellers often assume that a high price may discourage consumers from buying their products.

First things first, let's see how much profit we can make from screen protectors in general. How much do sellers charge for them?
We can analyze the prices in our marketplace and see what people usually pay for smartphone screen protectors:

Let's check how the number of sold items and daily earnings match the unit price (as a daily average):

```
# The daily earnings vs price:
sns.lmplot(x='sell_price', y='Daily_earnings',
           data=df[['sell_price', 'Daily_earnings']])

# Plot the sales frequency vs price:
sns.lmplot(x='sell_price', y='Mean_daily_sold_count',
           data=df[['sell_price', 'Mean_daily_sold_count']],)
```





As expected, higher prices mean fewer sales on average.

But when we look at the daily income, it seems that profits tend to increase with higher prices. At this point, it would be interesting to find out whether pricing reflects the quality and/or reputation of the product, but, unfortunately, that's beyond the scope of this study.

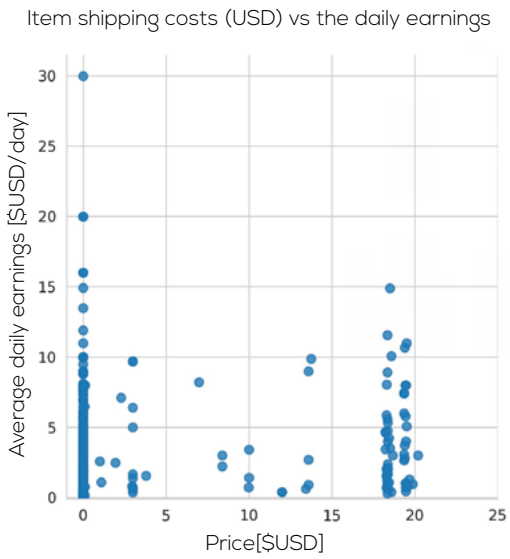
DO SHIPPING COSTS IMPACT CUSTOMER DECISIONS?

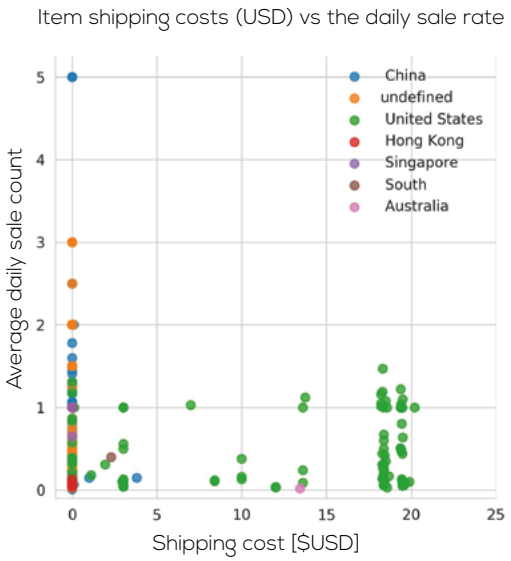
It seems that customers are prepared to pay a little extra for a higher-quality product.

But what about shipping costs? They're an additional cost that doesn't add any value to the purchased goods, especially with this type of product.

To visualize the relationship between sales and shipping costs, we used the following code:

```
sns.lmplot(data=df[['shipping_cost', 'Daily_earnings']],
           x='shipping_cost', y='Daily_earnings', fit_reg=False)

sns.lmplot(data=df[['shipping_cost', 'Mean_daily_sold_count']],
           x='shipping_cost', y='Mean_daily_sold_count', fit_reg=False, hue='shipping_from')
```





It appears that higher shipping costs impact on an item's sellability much. But we can note a small inverse correlation - free shipping options tend to be more popular.

Also, we can see a system of segregation in shipping costs as shipping is mostly either free or costs $18, with relatively few values in between.

Do the high delivery charges come with importing items from China? Adding the seller's location to the mix - as you can see in the third dimension (hue) - reveals that high delivery costs can be attributed mainly to items originating from within the US. Perhaps some customers just don't mind paying a bit more for courier services?

HOW MANY SELLERS DISCOUNT THEIR PRODUCT?

Speaking of pricing...

...let's check how many sellers discount their products - or at least how many of them indicate that in their offer, regardless of whether the promotion is real or not.

Let's start with a bit of code.

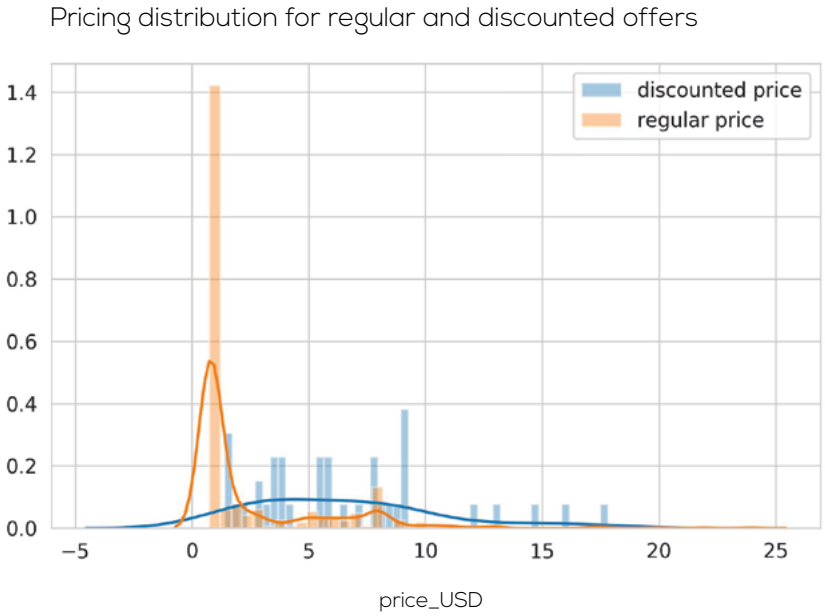
```
disc_prices_fraction = 100 * len(df.loc[df[discount_shown]]) / len(df)

fig, axes = plt.subplots()
fig.set_size_inches(5,5)

values = [disc_prices_fraction, 100 - disc_prices_fraction]
names = ['{} ({}%)'.format(val, round(values[int(n)]*100/sum(values), 1))
        for n, val in enumerate(['discounted', 'no discount'])]
```

```
9
10      my_circle=plt.Circle((0,0), 0.5, color='white')
11      plt.pie(values, labels=names, wedgeprops = {'linewidth': 4, 'edgecolor': 'white'})
12      plt.gca().add_artist(my_circle)
```

And we get this graph:

It seems that only a dozen percent of offers indicate that their price has been discounted. That's a small faction. It probably makes sense for the online marketplace to keep the number of discounted offers low so they appear to be more special.

Let's have a look at the price distribution of offers with and without discount.



This graph tells us that discounts are mostly applied to items that were more expensive in the first place.

The question is: are we seeing a real bargain or is it just a trick to earn more money?

To find that out, we'd need to check how many product variants were sold with and without a discount, and what the two pricing levels actually were. Unfortunately, that question lies beyond the planned scope of this study, so we'll leave it unanswered for now.

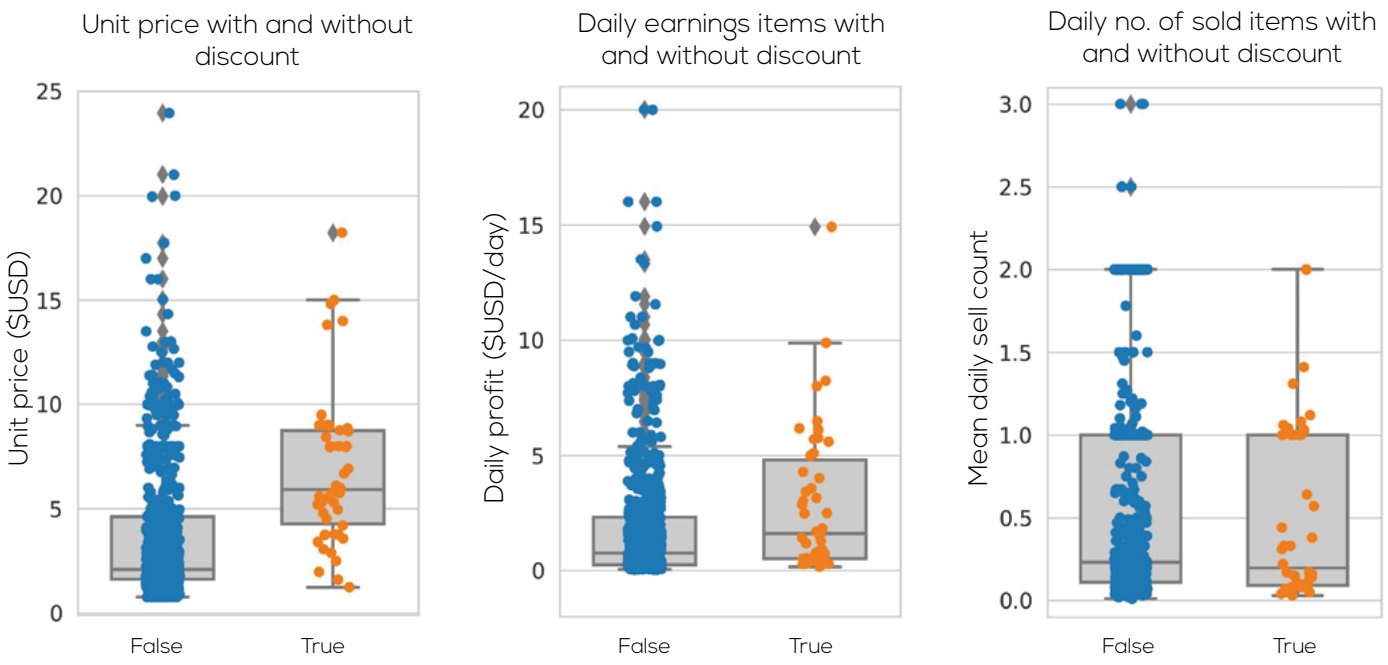
However, the explanation is likely to be far more straightforward - the regularly low prices are already so small that it would be silly to apply a discount to them.

SO, DOES IT MAKE SENSE TO OFFER DISCOUNTS?

The premise is straightforward: to attract more customers to your offer, you need to create a feeling that your price is an opportunity not to be missed. But is it worth the trouble? Plotting the following should shine some light on that:

```
sns.boxplot(data=df, x='discount_shown', y='Daily_earnings')
sns.stripplot(data=df, x='discount_shown', y='Daily_earnings')

sns.boxplot(data=df, x='discount_shown', y='Mean_daily_sold_count')
sns.stripplot(data=df, x='discount_shown', y='Mean_daily_sold_count')

sns.boxplot(data=df, x='discount_shown', y='sell_price')
sns.stripplot(data=df, x='discount_shown', y='sell_price')
```

That yields the following box/strip charts:



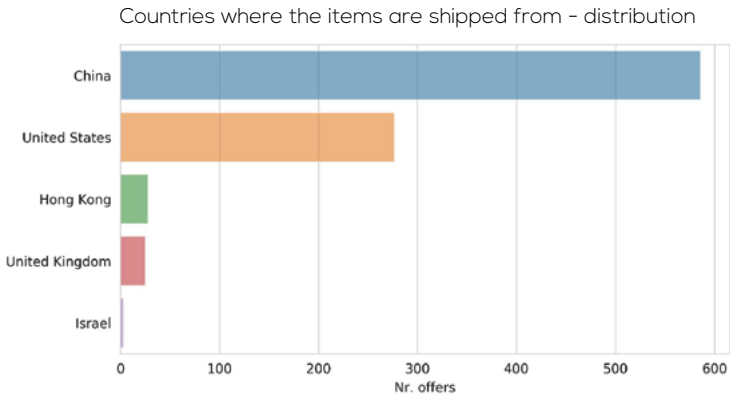
Just as we discovered before, discounts are more common among items with higher prices. It seems that the net effect is only a slight increase in an average daily income and no change to the daily sales rate. Looks like discounting products isn't worth it.

WHERE ARE THE PRODUCTS SHIPPED FROM?

Another thing we may want to learn about our competition is where they ship their products from. Here's how we can find that out:

```
shipped_from = pd.DataFrame(
    df['shipping_from'].value_counts())\
    .rename(columns={'shipping_from':'count'})
shipped_from = shipped_from.fillna('undefined')
plotted_df = shipped_from.head(5)

sns.set_style('whitegrid')
fig, ax = plt.subplots()

sns.barplot(data=plotted_df, y=plotted_df.index, x='count', ax=ax, alpha=0.6)
ax.set_xlabel('Nr. offers')
plt.suptitle('Countries where the items are shipped from - distribution',
             fontsize=14, y=1.04)
```

It's probably not surprising that China and the US are the two most significant sources of smartphone screen protectors (in that order). What raises an eyebrow is that some currencies in which the offers are being listed don't correspond to the declared shipping-from origins:




```
pd.DataFrame(df['currency'].value_counts()).rename(columns={'currency': 'nr. offers'})
```

	nr. offers
USD	542
CAD	389
AUD	3

That results in the following table:

Note, that while ~40 percent of offers listed with Canadian prices (see table), only a few offers indicate that items are in fact shipped from Canada.

Do all those sellers keep their goods for shipping in the US or do they use Canadian accounts (and perhaps now live in US)?

That's an interesting question, but well outside the scope of our study.

DOES IT MAKE ANY DIFFERENCE TO CUSTOMERS WHERE THEY PURCHASE PRODUCTS FROM?

When purchasing a new gadget, we customers usually want to receive it as quickly as possible.

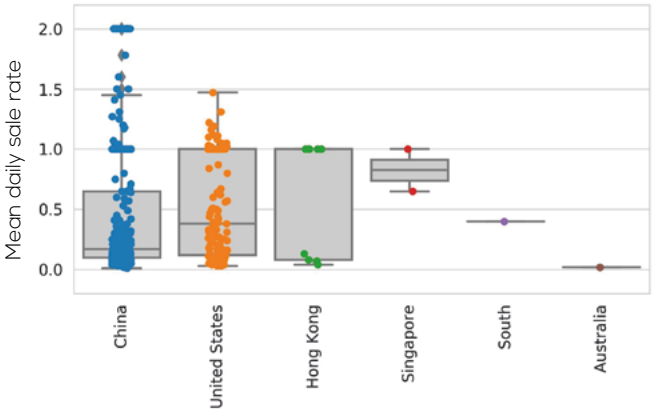
That may manifest in how they choose sellers - for example, by evaluating offers on the basis of their location and time it takes for products to arrive.

Since most customers of the online marketplace we studied are based in the US, it's reasonable to speculate that offers from Asian countries may suffer on the selling rate with respect to local suppliers.

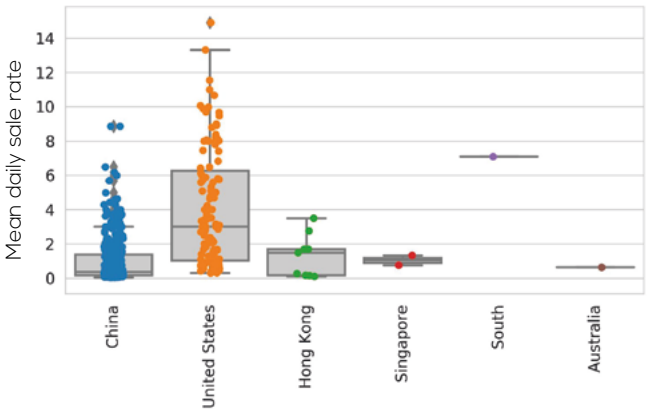
Let's check if there is any correlation:

```
sns.boxplot(data=df, x='shipping_from', y='Mean_daily_sold_count', color=".8")
sns.stripplot(data=df, x='shipping_from', y='Mean_daily_sold_count', jitter=True)

sns.boxplot(data=df, x='shipping_from', y='Daily_earnings', color=".8")
sns.stripplot(data=df, x='shipping_from', y='Daily_earnings',jitter=True)
```

Mean daily sale rates vs country where items are shipped from



Mean daily sale rates vs country where items are shipped from



Apparently, most customers of the marketplace are okay with the fact that their purchase is shipped from the other side of the planet. We can see a high frequency of transactions.

However, products offered from within the US generally yield higher profits. Taking into consideration our findings from the previous section - that higher prices translate into higher gains - it's tempting to conclude that products of Asian origin are being sold at lower prices to maintain a comparable sell rate.

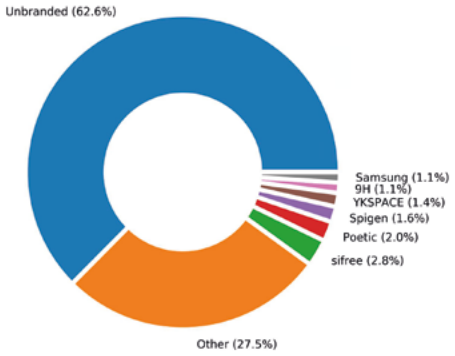
WHAT ARE THE MOST POPULAR SCREEN PROTECTOR BRANDS?

When looking at brand names, we can see a spectrum of values that refer to “no brand.” So let’s clean this mess up first:

```
df[‘brand’] = df[‘brand’].apply(
     lambda s: ‘Unbranded’ if s in
         [‘Does not apply’, ‘Does Not Apply’, ‘Unbranded/Generic’. ‘unbranded’]
     else s)
```

Now we can put data into work. We’ll get a pie chart showing brand names. It indicates that the majority of the products offered on our online marketplace (c. 60 percent) has no branding at all.

Most commonly offered brands.



Now, consumers may want to stick to a recognizable name, so let’s ignore the unnamed products for a moment.

Let’s focus on the top 20 brands offered in the marketplace that sell the highest number of products daily.

For this, we will use our `sale_history` table with all the transactions data.

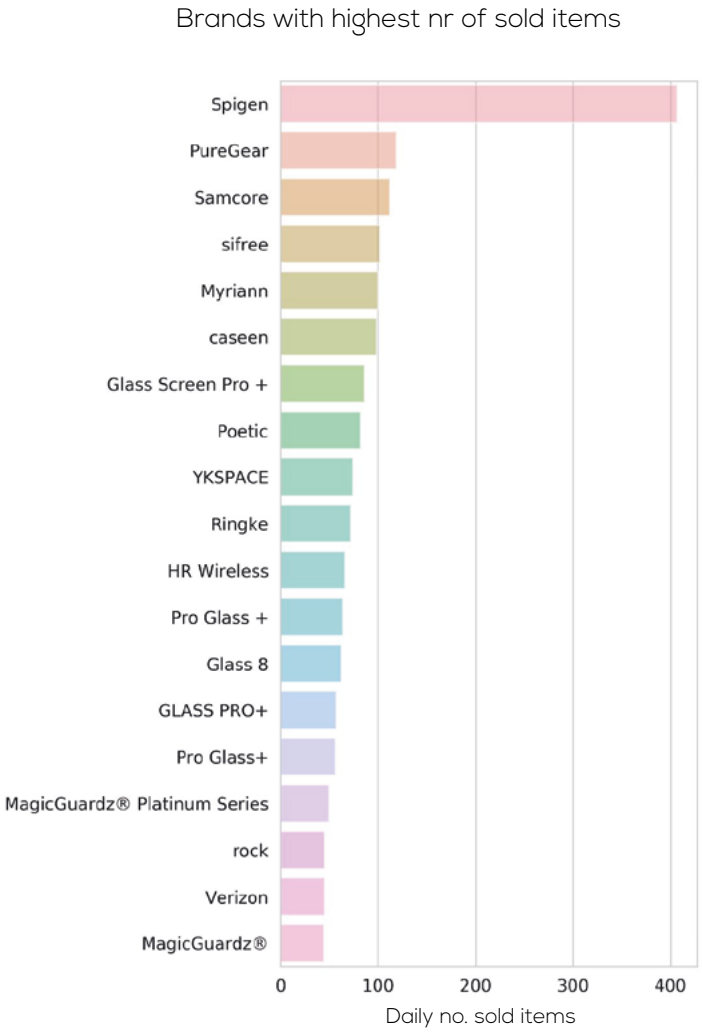
Let’s create a table with information about the offered brands:

```
sold_brands = sale_history.groupby(‘brand’).apply(calculate_sale_history_stats)
```

Next, let’s see the top 10 brands that have recorded the highest number of sold items so far.
We can create a table and plot it as follows:

```
top_sold_brands = sold_brands.sort_values(
    by=[‘Sold_count’, ‘Daily_earnings’, ‘Mean_daily_sold_count’],
    ascending=False).reset_index()

sns.barplot(data=top_sold_brands.iloc[1:21], x=’brand’, y=’Sold_count’)
```

	brand	Sold_count
0	Unbranded	4818.0
1	Spigen	407.0
2	PureGear	119.0
3	Samcore	112.0
4	sifree	102.0
5	Myriann	100.0
6	caseen	98.0
7	Glass Screen Pro +	86.0
8	Poetic	82.0
9	YKSPACE	74.0
10	Ringke	72.0
11	HR Wireless	66.0
12	Pro Glass +	64.0
13	Glass 8	62.0
14	GLASS PRO+	57.0
15	Pro Glass+	56.0
16	MagicGuardz® Platinum Series	50.0
17	rock	45.0
18	Verizon	45.0
19	MagicGuardz®	44.0



One glance and we can see that all the unnamed brands combined have accumulated the highest number of items sold. However, Spigen appears to be the runner-up in this category and dominates the market popularity among named brand products.

As sellers, we're interested in turning a profit.

So which of these brands are the most profitable? Which ones bring the highest revenue in the shortest time?

Let's bring the unbranded products back to the table as the picture might be slightly different when it comes to earnings:

```
most_profitable_brands = sold_brands.sort_values(
    by=['Daily_earnings', 'Mean_daily_sold_count', 'Sold_count'],
    ascending=False).reset_index()
most_profitable_brands = most_profitable_brands[[
    'brand', 'Daily_earnings', 'Daily_earnings_St.Dev','Sold_count',
    'Mean_daily_sold_count', 'Mean_Sold_count_St.Dev']]
```

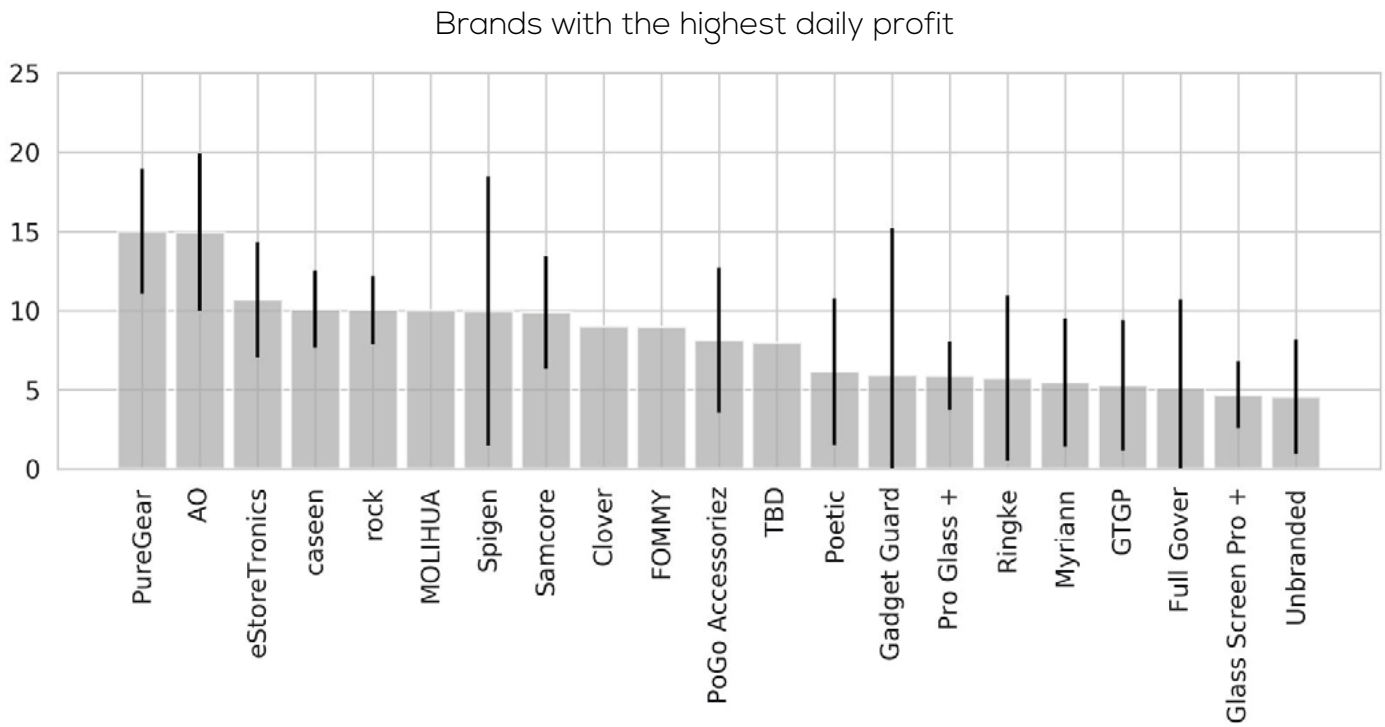
We get this table:

	brand	Daily_earnings	Daily_earnings_St.Dev	Sold_count	Mean_daily_sold_count	Mean_Sold_count_St.Dev
0	PureGear	14.98	3.95	119.0	1.09	0.29
1	AO	14.92	4.97	3.0	1.50	0.50
2	eStoreTronics	10.66	3.64	29.0	1.04	0.19
3	caseen	10.06	2.43	98.0	1.00	0.00
4	rock	10.01	2.15	45.0	1.00	0.00
5	MOLIHUA	9.98	0.00	1.0	1.00	0.00
6	Spigen	9.95	8.50	407.0	1.12	0.58
7	Samcore	9.87	3.55	112.0	1.12	0.43
8	Clover	8.99	0.00	1.0	1.00	0.00
9	FOMMY	8.95	0.00	2.0	1.00	0.00

And let's visualize it on a barchart like so:

```
1    plt.bar(x, y, width=0.85, yerr=y_err, alpha=0.7, color='darkgrey', ecolor='black')
```

That should create the following graph:



It's clear now that unbranded products aren't that profitable. Note that the order of brands changed significantly when we compare the revenue instead of the total number of sold items. It seems that brands with mid-range prices have come up to the top now (like PureGear that costs about $9.5) - despite their infrequent daily rate of sales (c. 1-2/day).

"Quality over quantity," as the old saying goes, is probably the smartest way of approaching a sales strategy for online marketplaces.

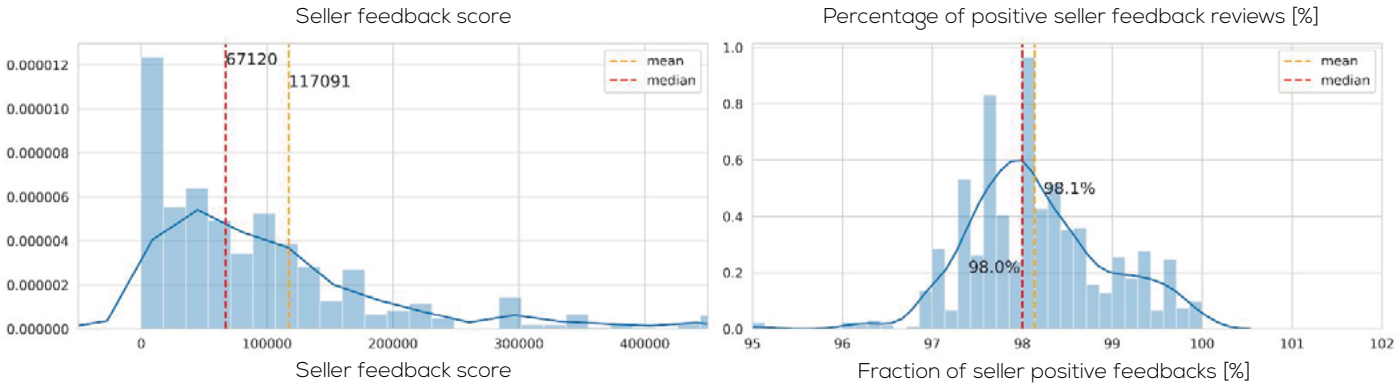
WHAT ARE THE AVERAGE RATINGS OF SELLERS WHO OFFER THIS PRODUCT?

One of the most essential aspects customers take into account when making decisions are reviews and ratings.

Let's check the average ratings of sellers who offer smartphone screen protectors on our marketplace:

```
fig, axes = plt.subplots(nrows=1, ncols=2)

ax = axes[0]
sns.distplot(sfeedback_scores, ax=ax, bins=250)
ax.axvline(mean_score, linestyle='--', color='orange', label='mean')
ax.axvline(median_score, color='red', linestyle='--', label='median')

ax1 = axes[1]
sns.distplot(sfeedback_perc, ax=ax1, bins=100)
ax1.axvline(mean_perc, linestyle='--', color='orange', label='mean')
```

Feedback scores: Median: 67.1 k, Mean: 117 k ± 306 k,
Percentage of positive reviews: Median: 98.0%, Mean: 98.1% ± 1.0%



There isn't much we can do to sway customers to our product when it comes to reviews. That's why it's enough to just make sure that we get our order right and deliver products on time. The data suggests that having at least 98% of positive feedback score gets you a healthy relationship with your customers.

HOW DOES THE SELLER FEEDBACK SCORE AFFECT SALES?

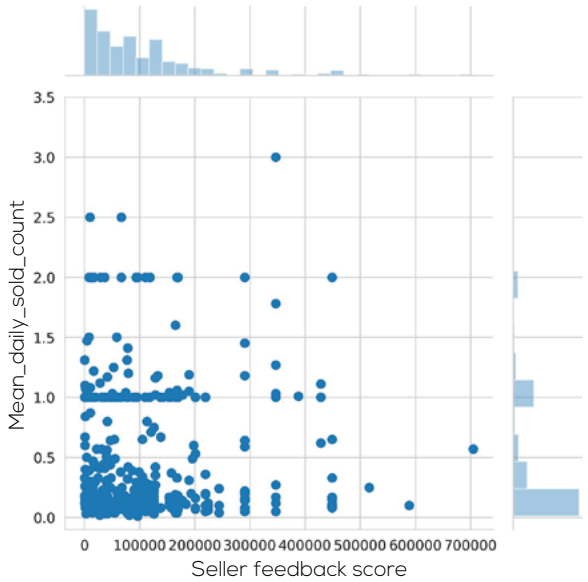
Another critical factor that affects customer purchase decisions is seller feedback.

After all, consumers prefer to choose reliable providers.

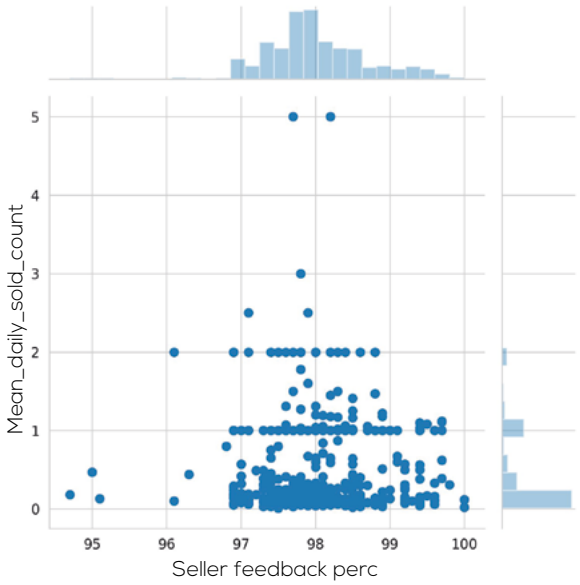
Is that intuitive preference reflected in the numbers? Let's check whether if there is any correlation between sales with seller feedback:

```
sns.jointplot(data=df[['seller_feedback_score', 'Mean_daily_sold_count']],
              x='seller_feedback_score', y='Mean_daily_sold_count', kind='scatter')

sns.jointplot(data=df[['seller_feedback_perc', 'Mean_daily_sold_count']],
              x='seller_feedback_perc', y='Mean_daily_sold_count', kind='scatter')
```

Seller feedback score vs the daily sale rate



Fraction of seller positive feedback reviews vs daily sale rate



At first glance, the data suggests that the higher the score, the weaker the selling rate. Interpreting that can be tricky because of how scores are awarded. The most likely explanation is that many sellers have not managed to gain the points yet and so the trend is simply a manifestation of a normal-like distribution of values.

It's likely that customers don't pay much attention to the feedback score when it comes to buying smartphone screen protectors. But that may be entirely different for premium goods.

HOW MANY OFFERS HAVE A QUALITY BADGE?

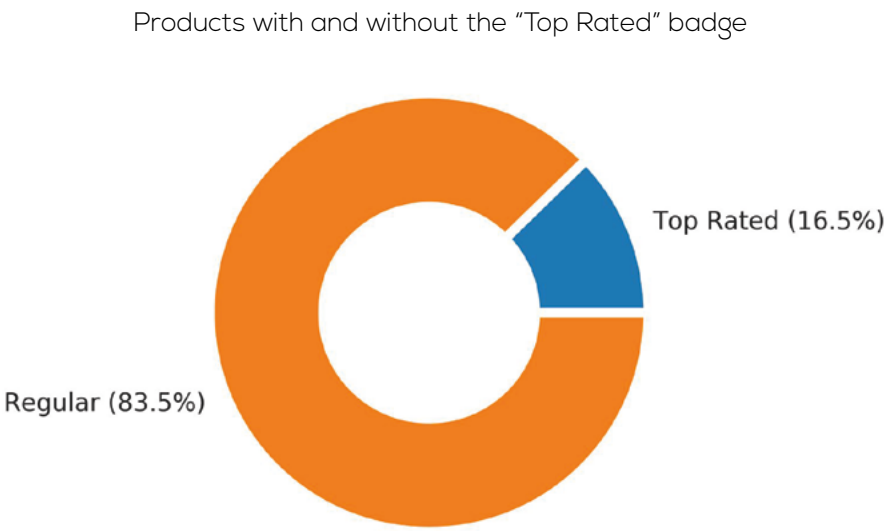
Speaking of seller reliability...

...one of the features customers may be taking into account when choosing products is a quality badge sellers award to the most successful and highest-rated products.
Let's check how often sellers award a quality badge to smartphone screen protectors:

```
top_rated = [len(df.loc[df['top_rating_badge']]), len(df.loc[~df['top_rating_badge']])]
fig, axis = plt.subplots()

names = ['{} ({}%)'.format(val, round(top_rated[int(n)]*100/sum(top_rated), 1))
        for n, val in enumerate(['Top Rated Badge', 'regular'])]
my_circle=plt.Circle((0,0), 0.5, color='white')

plt.pie(size, labels=names, wedgeprops = { 'linewidth' : 4, 'edgecolor' : 'white' })
axis.add_artist(my_circle), plt.title('Fraction of offers with Top Rating badge', fontsize=14)
```

It looks like the ratio of offers that have a quality badge is kept low - only about 17 percent of screen protectors sold on our marketplace has it.

Is it difficult to get this badge? Or does the marketplace cap the number of possible badges awarded to maintain its premium status? The most likely explanation is a combination of both: an algorithm may simply select a fraction of the top products.



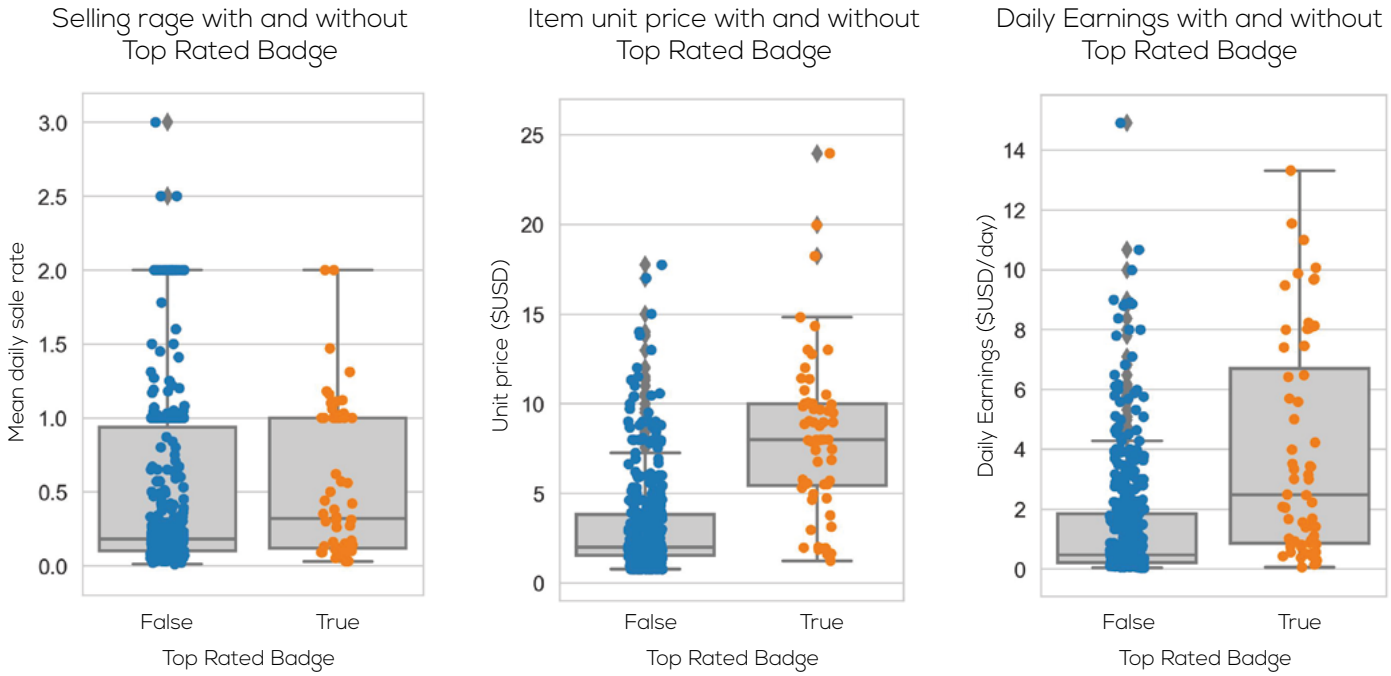
DOES HAVING A QUALITY BADGE BOOST PRODUCT SALES?

Achieving a top rank among sellers takes a lot of time and effort, so it's only natural to ask:

Is it worth to be awarded a quality badge for your product offer? Does it translate into more sales?

Customers probably pay attention to such extra features, but let's find out and pull the numbers again:

```
sns.boxplot(data=df, x='top_rating_badge', y='Mean_daily_sold_count')
sns.stripplot(data=df, x='top_rating_badge', y='Mean_daily_sold_count')

sns.boxplot(data=df, x='top_rating_badge', y='Daily_earnings')
sns.stripplot(data=df, x='top_rating_badge', y='Daily_earnings')

sns.boxplot(data=df, x='top_rating_badge', y='sell_price')
sns.stripplot(data=df, x='top_rating_badge', y='sell_price')
```



Even though it doesn't make a massive impact on the sales rate, a quality badge awarded to products results in a price increase and, as we established already before, an increase in the overall profit for the seller.

CONCLUSIONS

This research study was designed to show how we can use Python programming tools to develop a smarter sales...

...strategy and answer some of the critical questions about what really matters when listing a product on online marketplaces.

We used a smartphone screen protector as an example to check which aspects of an offer translate into higher profits and to what should sellers pay attention when entering this online marketplace.

Key insights:

- Quality is better than quantity - selling mid-range price variants can yield higher profits, despite a slightly lower average selling rate.
- Shipping rates and product origin aren't that important for US-based customers - however, we can earn more when selling locally.
- It's best to choose one or two well-known brands rather than sell unbranded items.
- Ideally, sellers should gain a quality badge.
- Seller feedback score and discount rates brought no significant advantages, probably because the vast majority of sellers had already at least ~98 percent of positive reviews and it was hard to stand out from the crowd.

There's no golden ticket to success if you're competing with sellers who offer many products similar to yours on an online marketplace. But our study shows that the devil is in the details. The right combination of all these elements guarantees success.

WHAT'S NEXT?

This study is by no means exhaustive as we only scratched the surface of this complex topic.

Our inquiry raised many new questions that we can't address with the currently available data - for example, are discounts we observed truly lower prices or just a trick?

It would be interesting to check how the less obvious characteristics of product presentation - such as picture quality, color palette, product description, and others - help sellers attract customers and boost sales.

We will be addressing some of these issues in our next study, so stay tuned for more!

For more quality content about Python programming and data science

Check out our blog

Are you looking for a technology partner?

Hire us!

Become part of our team!

Get in touch

sunscrapers

Unrivaled Python Engineers

Sunscrapers Sp. z o.o.
Pokorna 2/947 (entrance 9)
00-199 Warsaw

New Business
hello@sunscrapers.com
Careers
careers@sunscrapers.com

TOP DEVELOPERS
Clutch
POLAND
2017