

Optimizing Online Sports Retail Revenue



Project Overview

- This is an SQL project is based on the product and revenue analysis for an example online sports clothing company.
- Exploratory Data analysis in SQL skills is used to analyze the data to produce recommendations on how the company can maximize revenue!

Objective:

- The company is specifically interested in how it can improve revenue.

Prerequisites:

- Jupyter Notebook
- PostgreSQL and a database (DB Name: Sports)

Libraries to install:

- `ipython-sql` -- This enables the use of SQL magic functions that contain % and %% , allows to write SQL style code right in Jupyter Notebook.
- `sqlalchemy` -- SQL toolkit and object-relational-mapper for Python
- A python database API (DBAPI) library (For PostgreSQL, I've used `psycopg2`)

Techniques used:

- Exploratory Data Analysis includes:
 - SQL JOINS
 - Aggregation
 - cleaning
 - labeling (Conditional column using CASE..WHEN)
 - Common Table Expressions
 - Correlation

Data types used:

- Numeric (float/int)
- String (varchar/char)
- Timestamp - timestamp

Database tables:

info

column	data type	description
product_name	varchar	Name of the product
product_id	varchar	Unique ID for product
description	varchar	Description of the product

finance

column	data type	description
product_id	varchar	Unique ID for product
listing_price	float	Listing price for product
sale_price	float	Price of the product when on sale
discount	float	Discount, as a decimal, applied to the sale price
revenue	float	Amount of revenue generated by each product, in US dollars

reviews

column	data type	description
product_name	varchar	Name of the product
product_id	varchar	Unique ID for product
rating	float	Product rating, scored from 1.0 to 5.0
reviews	float	Number of reviews for the product

traffic

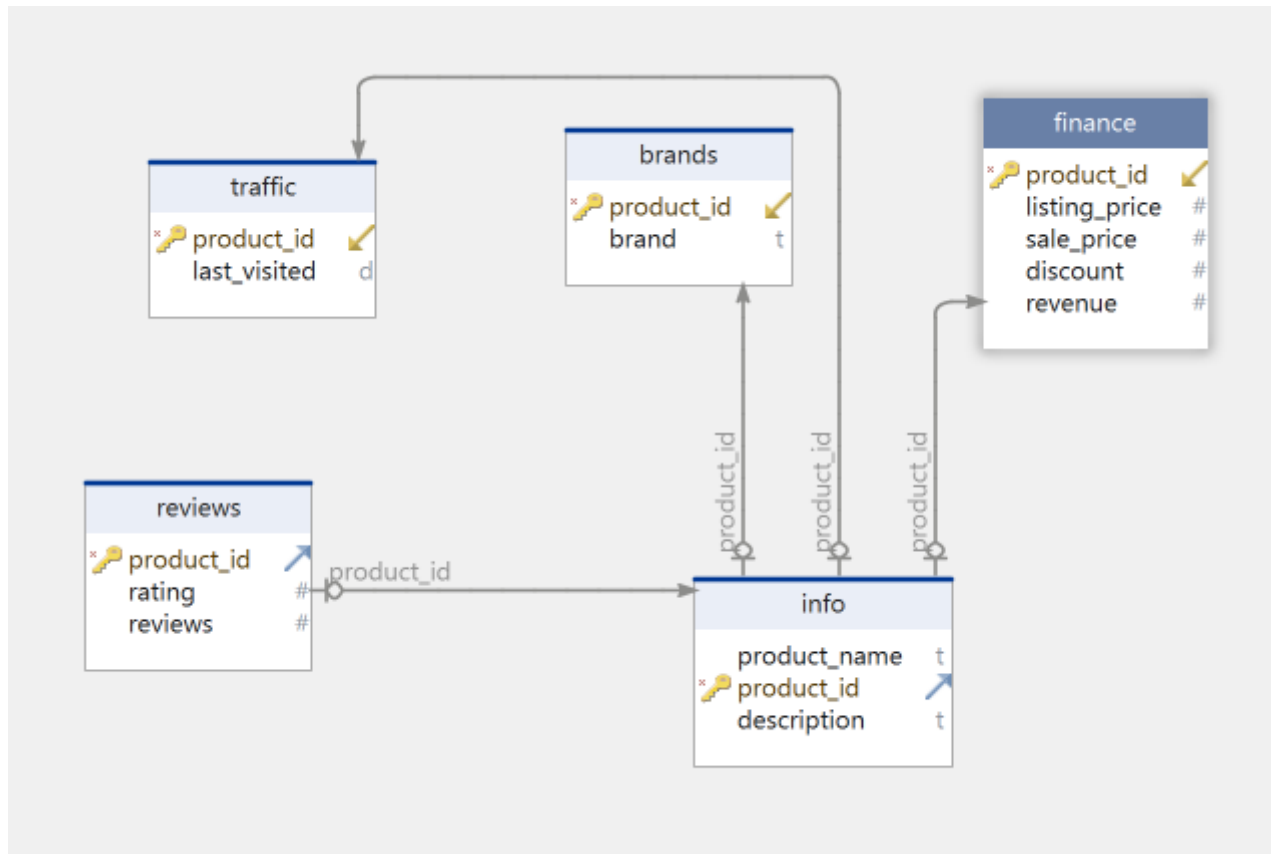
column	data type	description
product_id	varchar	Unique ID for product
last_visited	timestamp	Date and time the product was last viewed on the website

brands

column	data type	description
product_id	varchar	Unique ID for product

brand varchar Brand of the product

Data Model:



Getting Started

Importing Libraries

```
In [1]: import warnings
warnings.simplefilter(action='ignore', category=UserWarning)
```

```
import sqlalchemy as sa
```

Connecting to PostgreSQL DB

```
In [2]: engine = sa.create_engine('postgresql://postgres:password@localhost:5432/Sports')
```

```
In [3]: %load_ext sql
```

```
In [4]: %sql postgresql://postgres:password@localhost/Sports
```

1. Counting missing values

Checking for missing data as well as numeric, string, and timestamp data types to draw insights about the products in the online store. Checking how complete the data is.

```
In [14]: %%sql

-- Counting all columns as total_rows
-- Counting the number of non-missing entries for description, listing_price, and last_visited
-- Joining info, finance, and traffic

SELECT COUNT(*) AS total_rows,
       COUNT(i.description) AS count_description,
       COUNT(f.listing_price) AS count_listing_price,
       COUNT(t.last_visited) AS count_last_visited
FROM info AS i
INNER JOIN finance AS f
     ON i.product_id = f.product_id
INNER JOIN traffic AS t
     ON t.product_id = f.product_id;

* postgresql://postgres:***@localhost/Sports
1 rows affected.
```

```
Out[14]:
```

total_rows	count_description	count_listing_price	count_last_visited
3179	3117	3120	2928

Findings:

- The database contains 3,179 products in total.
- Of the columns previewed, only one — last_visited — is missing more than five percent of its values.

2. Checking Product Pricing :

- How do the price points of different brand products differ?
- Answering this question can help to build a picture of the company's stock range and customer market.
- Below query is used to produce a distribution of the listing_price and the count for each price, grouped by brand .

In [15]: `%%sql`

```
-- Selecting the brand, listing_price as an integer, and a count of all products in finance
-- Joining brands to finance on product_id
-- Filtering for products with a listing_price more than zero
-- Aggregating results by brand and listing_price, and sort the results by listing_price in descending order

SELECT brand,cast(listing_price AS int),COUNT(finance.product_id)
FROM
brands
INNER JOIN finance
ON brands.product_id = finance.product_id
GROUP BY
brand , listing_price
HAVING listing_price > 0
ORDER BY listing_price DESC;
```

* postgresql://postgres:***@localhost/Sports
77 rows affected.

Out[15]:

brand	listing_price	count
-------	---------------	-------

Adidas	300	2
Adidas	280	4
Adidas	240	5
Adidas	230	8
Adidas	220	11
Nike	200	1
Adidas	200	8
Nike	190	2
Adidas	190	7
Nike	180	4
Adidas	180	34
Nike	170	14
Adidas	170	27
Nike	160	31
Adidas	160	28
Adidas	150	41
Nike	150	6
Adidas	140	36
Nike	140	12
Adidas	130	96
Nike	130	12
Adidas	120	115
Nike	120	16
Nike	110	17

brand	listing_price	count
Adidas	110	91
Nike	100	14
Adidas	100	72
Adidas	96	2
Nike	95	1
Nike	90	13
Adidas	90	89
Adidas	86	7
Nike	85	5
Adidas	85	1
Adidas	80	322
Nike	80	16
Nike	79	1
Adidas	76	149
Adidas	75	1
Nike	75	7
Adidas	70	87
Nike	70	4
Adidas	66	102
Nike	65	1
Adidas	63	1
Nike	60	2
Adidas	60	211
Adidas	56	174

brand	listing_price	count
Adidas	55	2
Adidas	53	43
Nike	50	5
Adidas	50	183
Adidas	48	42
Nike	48	1
Adidas	46	163
Nike	45	3
Adidas	45	1
Adidas	43	51
Adidas	40	81
Nike	40	1
Adidas	38	24
Adidas	36	25
Adidas	33	24
Adidas	30	37
Nike	30	2
Adidas	28	38
Adidas	27	18
Adidas	25	28
Adidas	23	1
Adidas	20	8
Adidas	18	4
Adidas	16	4

brand	listing_price	count
Adidas	15	27
Adidas	13	27
Adidas	12	1
Adidas	10	11
Adidas	9	1

Findings:

- It turns out there are 77 unique prices for the products in the database for Nike and Adidas brands, which makes the output of the last query quite difficult to analyze.

3. Labeling price ranges

- Assigning labels to different price ranges, grouping by brand and label .Also included the total revenue for each price range and brand .
- Adding conditional column "price_category" for labeling products using CASE..WHEN statement.

In [16]:

```
%%sql

-- Selecting the brand, a count of all products in the finance table, and total revenue
-- Creating four labels for products based on their price range, aliasing as price_category
-- Joining brands to finance on product_id and filter out products missing a value for brand
-- Grouping results by brand and price_category, sort by total_revenue

SELECT brand,COUNT(finance.product_id),SUM(revenue) AS total_revenue,
CASE
WHEN listing_price < 42 THEN 'Budget'
WHEN listing_price >= 42 AND listing_price < 74 THEN 'Average'
WHEN listing_price >= 74 AND listing_price < 129 THEN 'Expensive'
ELSE 'Elite' END AS price_category
FROM finance
INNER JOIN brands
ON brands.product_id = finance.product_id
GROUP BY brand ,price_category
```

```
HAVING brand is not null
ORDER BY total_revenue DESC;
```

```
* postgresql://postgres:***@localhost/Sports
8 rows affected.
```

Out[16]:

brand	count	total_revenue	price_category
-------	-------	---------------	----------------

Adidas	849	4626980.0699999999	Expensive
Adidas	1060	3233661.0600000001	Average
Adidas	307	3014316.8299999987	Elite
Adidas	359	651661.1200000002	Budget
Nike	357	595341.0199999992	Budget
Nike	82	128475.59000000003	Elite
Nike	90	71843.15000000004	Expensive
Nike	16	6623.5	Average

Findings:

- Interestingly, grouping products by brand and price range shows that Adidas items generate more total revenue regardless of price category!

4. Checking for average discount by brand

- The listing_price may not be the price that the product is ultimately sold for.
- To understand revenue better, now checking the discount , which is the percent reduction in the listing_price when the product is actually sold.
- Let's understand whether there is a difference in the amount of discount offered between brands, as this could be influencing revenue.

In [17]:

```
%%sql
-- Selecting brand and average_discount as a percentage
-- Joining brands to finance on product_id
-- Aggregating by brand
-- Filtering for products without missing values for brand
```

```
SELECT brand,AVG(discount)*100 AS average_discount
FROM brands
INNER JOIN finance
ON brands.product_id = finance.product_id
GROUP BY brand
HAVING brand is not null;
```

```
* postgresql://postgres:***@localhost/Sports
2 rows affected.
```

Out[17]:

brand	average_discount
Nike	0.0
Adidas	33.452427184465606

Findings:

- Strangely, no discount is offered on Nike products! In comparison, not only do Adidas products generate the most revenue, but these products are also heavily discounted.

5. Checking correlation between revenue and reviews

- Exploring whether relationships exist between the columns in our database. Checking the strength and direction of a correlation between revenue and reviews.

In [18]:

```
%%sql

-- Calculating the correlation between reviews and revenue as review_revenue_corr
-- Joining the reviews and finance tables on product_id

SELECT corr("reviews", "revenue") as "review_revenue_corr"
FROM
reviews
INNER JOIN finance
ON reviews.product_id = finance.product_id;
```

```
* postgresql://postgres:***@localhost/Sports
1 rows affected.
```

Out[18]: **review_revenue_corr**

0.6518512283481301

Findings:

- Interestingly, there is a strong positive correlation between revenue and reviews . This means, potentially, getting more reviews on the company's website, may increase sales of those items with a larger number of reviews.

6. Checking ratings and reviews by product description length

- Checking whether the length of a product's description might influence a product's rating and reviews — if so, the company can produce content guidelines for listing products on their website and test if this influences revenue .

In [19]: **%%sql**

```
-- Calculating description_length
-- Converting rating to a numeric data type and calculate average_rating
-- Joining info to reviews on product_id and group the results by description_length
-- Filtering for products without missing values for description, and sort results by description_length
```

```
SELECT TRUNC(LENGTH(info.description), -2) AS description_length,
       ROUND(AVG(reviews.rating::numeric), 2) AS average_rating
FROM info
INNER JOIN reviews
  ON info.product_id = reviews.product_id
WHERE info.description IS NOT NULL
GROUP BY description_length
ORDER BY description_length;
```

```
* postgresql://postgres:***@localhost/Sports
7 rows affected.
```

Out[19]:

description_length	average_rating
0	1.87
100	3.21
200	3.27
300	3.29
400	3.32
500	3.12
600	3.65

description_length	average_rating
0	1.87
100	3.21
200	3.27
300	3.29
400	3.32
500	3.12
600	3.65

Findings:

- There doesn't appear to be a clear pattern between the length of a product's description and its rating .

7. Checking reviews by month and brand

- Taking a look at the volume of reviews by month to see if there are any trends or gaps to exploit.

In [20]:

```
%%sql

-- Selecting brand, month from last_visited, and a count of all products in reviews aliased as num_reviews
-- Joining traffic with reviews and brands on product_id
-- Grouping by brand and month, filtering out missing values for brand and month
-- Ordering the results by brand and month

SELECT b.brand, DATE_PART('month', t.last_visited) AS month , COUNT(r.product_id) as num_reviews
FROM brands b
INNER JOIN reviews r
    ON b.product_id = r.product_id
INNER JOIN traffic t
    ON r.product_id = t.product_id
GROUP BY b.brand , month
HAVING b.brand IS NOT NULL
    AND DATE_PART('month', t.last_visited) IS NOT NULL
ORDER BY b.brand, month;
```

```
* postgresql://postgres:***@localhost/Sports  
24 rows affected.
```

Out[20]:

brand	month	num_reviews
-------	-------	-------------

Adidas	1.0	253
Adidas	2.0	272
Adidas	3.0	269
Adidas	4.0	180
Adidas	5.0	172
Adidas	6.0	159
Adidas	7.0	170
Adidas	8.0	189
Adidas	9.0	181
Adidas	10.0	192
Adidas	11.0	150
Adidas	12.0	190
Nike	1.0	52
Nike	2.0	52
Nike	3.0	55
Nike	4.0	42
Nike	5.0	41
Nike	6.0	43
Nike	7.0	37
Nike	8.0	29
Nike	9.0	28
Nike	10.0	47
Nike	11.0	38
Nike	12.0	35

Findings:

- Seems product reviews are highest in the first quarter of the calendar year.

Analyzing type of products sold

8. Checking for Footwear product performance

- As there are no labels for product type, will create a Common Table Expression (CTE) that filters description for keywords, then use the results to find out how much of the company's stock consists of footwear products and the median revenue generated by these items.

In [21]:

```
%%sql
-- Creating the footwear CTE, containing description and revenue
-- Filtering footwear for products with a description containing %shoe%, %trainer, or %foot%
-- Also filtering for products that are not missing values for description
-- Calculating the number of products and median revenue for footwear products

WITH footwear AS
(SELECT i.description , f.revenue
FROM info i
INNER JOIN finance f
ON i.product_id = f.product_id
WHERE (i.description ILIKE '%shoe%' OR i.description ILIKE '%trainer%' OR i.description ILIKE '%foot%' )
AND i.description IS NOT NULL)

SELECT COUNT(*) AS num_footwear_products,
PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY revenue) AS median_footwear_revenue
FROM
footwear
;
```

```
* postgresql://postgres:***@localhost/Sports
1 rows affected.
```

```
Out[21]: num_footwear_products  median_footwear_revenue
```

```
          2700                3118.36
```

Findings:

- Recalling from the first task that we found there are 3,117 products without missing values for description . Of those, 2,700 are footwear products, which accounts for around 85% of the company's stock. They also generate a median revenue of over \$3000 dollars!

9. Checking for Clothing product performance

- To understand whether footwear's median_revenue is good or bad compared to other products. So,for the final task, let's examine how this differs to clothing products. footwear CTE is re-used , adding a filter afterward to count the number of products and median_revenue of products that are not in footwear .

```
In [22]: %%sql

-- Copying the footwear CTE from the previous task
-- Calculating the number of products in info and median revenue from finance
-- Inner join info with finance on product_id
-- Filtering the selection for products with a description not in footwear

WITH footwear AS
(SELECT i.description AS description , f.revenue
FROM info i
INNER JOIN finance f
ON i.product_id = f.product_id
WHERE (i.description ILIKE '%shoe%' OR i.description ILIKE '%trainer%' OR i.description ILIKE '%foot%' )
AND i.description IS NOT NULL)

SELECT COUNT(*) AS num_clothing_products,
PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY revenue) AS median_clothing_revenue
FROM info i
INNER JOIN finance f
ON i.product_id = f.product_id
```

```
WHERE i.description NOT IN ( select description FROM footwear )  
;
```

```
* postgresql://postgres:***@localhost/Sports  
1 rows affected.
```

Out[22]: **num_clothing_products median_clothing_revenue**

417

503.82

Summarizing

- Specifically, "Elite" Adidas products priced \$129 or more typically generate the highest revenue, so the company can potentially increase revenue by shifting their stock to have a larger proportion of these products!
- To improve revenue further, the company could try to reduce the amount of discount offered on Adidas products, and monitor sales volume to see if it remains stable. Alternatively, it could try offering a small discount on Nike products. This would reduce average revenue for these products, but may increase revenue overall if there is an increase in the volume of Nike products sold.
- As we know a correlation exists between reviews and revenue , one approach the company could take is to run experiments with different sales processes encouraging more reviews from customers about their purchases, such as by offering a small discount on future purchases.
- Footwear's median_revenue is (3118.36) is much higher than clothing products (503.82) so the focus needs to be more on increasing company's stock toward footwear and coming up with better discount offers on each brand.

In []: