Steven Chen – Western University
Shopify Data Science Challenge Submission

NOTE: My queries only work with MySQL v8.0 (default on fiddle is v5.6) because it uses the new lag(..) function.

1. Are there incidences of shops are increasing their prices? Does it occur on a regular basis?

   My queries and analysis use purchase order information as a proxy for product price information. My metrics uses the assumption that: the order's date of purchase = date that the price changed AND that all price changes are reflected in the orders (there doesn't exist a price change that is NOT recorded as an order).

```
select
product,
placed_on,
price,
price - lag(price, 1) over (partition by product order by placed_on) as
price_change,
datediff(placed_on, lag(placed_on, 1) over (partition by product order
by placed_on)) as length_of_time
from order_items
left join product_variations on
order_items.product_variation=product_variations.id
left join orders on order_items.order = orders.id
```

   The query above creates two new columns:
   "price_change" which stores the price change of a product from the previous order's price AND
   "length_of_time" which stores the number of days between the date the current price was recorded and the date the previous price was recorded.
   Below is a portion of the query result.

product	placed_on	price	price_change	length_of_time
1	2014-01-10 09:23:48	91.44	null	null
1	2014-02-16 16:09:51	91.66	0.220001220703125	37
1	2015-01-15 17:52:33	93.69	2.029998779296875	333
1	2015-01-20 23:39:51	93.09	-0.600006103515625	5
1	2015-02-15 11:18:16	94.41	1.32000732421875	26

```
select count(*), avg(length_of_time), avg(price_change)

from (
  select
  product,
  placed_on,
  price,
  price - lag(price, 1) over (partition by product order by placed_on) as
price_change,
  datediff(placed_on, lag(placed_on, 1) over (partition by product order by
placed_on)) as length_of_time
  from order_items
  left join product_variations on
order_items.product_variation=product_variations.id
  left join orders on order_items.order = orders.id
) as tmp

where price_change>0
```

By adding the subquery above, it gives us the number of price increases, the average length of time between price changes, and average price change. There are **709** instances of price increases found in the data set and on average it occurs roughly every **79 days** which means it occurs somewhat frequently. If database was connected to third-party software, it would be more insightful to visualize the distribution of the average length of time (instead of just the average of 79). The average price increase is about **$0.37**.

2. What is the average annual price increase of products in this database, if any.

```
select avg(length_of_time), avg(price_change)

from (
  select
  product,
  placed_on,
  price,
  price - lag(price, 1) over (partition by product order by
placed_on) as price_change,
  datediff(placed_on, lag(placed_on, 1) over (partition by
product order by placed_on)) as length_of_time
  from order_items
  left join product_variations on
order_items.product_variation=product_variations.id
  left join orders on order_items.order = orders.id
) as tmp

where price_change is not NULL
```

My approach to estimating this statistic is by using average price change and average length of time for a price change found the query above. The query is very similar to the

query in question 1 except that is accounts for ALL price changes, including price decreases.

$$\frac{avg\ price\ change}{avg\ length\ of\ time} = \frac{\$0.19}{61.6\ \ days}$$

$$\frac{\$0.19}{61.6\ \ days} \times \frac{5.92}{5.92} = \frac{\$1.108}{365\ days}$$

So, the estimated annual price change of products in this database is $1.11.

3. If it were being redeveloped, what changes would you make to the database schema given to make it more flexible?

   One of the biggest issues with the current database schema is that product price changes are not tracked directly, we only infer price changes from order information. We need a way of tracking price changes directly for more accurate analysis. One possible way of doing this is by adding a table called 'prices' which contains columns 'price_id', 'product_id', and 'effective_date'. This table would hold all pricing change history. A change to a product's price is recorded in this table. I would remove the 'price' column of the products table and if I wanted the current price, I could query the 'prices' table for most recent price. Also, if desired, the 'orders' table can have a 'price_id' column instead of a 'price' column to relate these tables. If wanted to analysis pricing history for a product, I could select all rows with desired 'product_id' from the prices table.

   Another minor change is that I would include a column in the 'orders' table that stores the merchant id. This way, you won't have to do SQL joins between the tables orders, product_variations, and product, and merchants just to see what order belong to which merchant. This will allow much faster merchant-order analysis.

4. Does the query (/queries) you wrote scale? What if there were hundreds of thousands of products, customers, variations and orders? What changes might you make to your technique, or the database itself, to optimize this sort of analysis?

   My query may significantly slow down as database scales. Without changing the database schema, one way to improve performance of my query is by maintaining indexes on the columns used to "join" in my query. However, maintaining indexes takes up space and slows down table modifications. If we were to implement the database schema change I mentioned in the previous question by storing price information in one table, this would significantly improve performance as you can perform time series analysis of product price information without using table joins.

5. Without rewriting it, how would your analysis change if the prices were presented in multiple currencies?

   You would need to convert the prices to the same currency, say Canadians dollars, by converting the price using the exchange rate at the given the point in time. You would also need the exchange rate data which could be stored in a separate table if the conversion was to be done in SQL which would probably slightly slow down the query time. If query time is an issue, you can store both the original price and converted price in 2 columns when rows are inserted. This way, you don't have to do price conversions every time you query.

6. Based on your findings above, would you recommend building the new feature? Why or why not?

   On average merchants increase their prices every 79 days and the average price increase is $0.37 which means merchants tend to make *small incremental* increases in their prices. Therefore, it likely means that merchants do not find it too troublesome to update their prices – otherwise, they would only update their prices when warranted by a *large incremental* increase. The average annual price change of $1.11 indicates that prices are overall keeping pace with positive inflation. Based on the simple metrics that we gathered, it seems that merchants are adequately adjusting prices using existing tools that are provided. Therefore, a new feature may not be necessary.

   There are however several shortfalls to our analysis that we must recognize. Firstly, our assumptions may be weak in that purchase order information may not accurately reflect product's price changes. Moreover, this analysis is only based on a limited set of statistics and we have not considered the variation and distribution of the data. There may be a significant subset of merchants who rarely change prices and find it difficult to do - this might not be captured in the previously mentioned statistics. Furthermore, just because merchants know how to and are changing their prices adequately, it does not necessarily mean that they find it *easy* to do. Adding a simple price changing feature may still greatly improve user experience.