Rec10

April 19, 2022

1 Recitation 10: Causality

Due by Friday April 22 at 11:59pm

In this recitation you will get some experience understanding the impact of price on requests for Uber trips in New York City, how causality plays a role in estimating this impact, and how one can use an A/B experiment to better understand this impact.

[191]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

1.0.1 Part 1: Exploratory Data Analysis

Begin by reading in `uber_rec10.csv` and printing out the first 5 rows.

This is a file with realistic (but not real) data based on the NYC taxi dataset.

For each zone (location_id) in NYC and each hour of day, for 4 saturdays in July 2017, it contains: * sessions: the number of sessions from riders * drivers: the number of drivers * desire_to_ride: a number indicating riders' price elasticity, i.e., how much an increase in price would discourage them from taking a trip. This would not typically be observed in reality but is provided to allow simulation. Typically, when we have a larger desire to ride, this is because of some event (it's raining, a big concert just got out) and we also see more sessions than usual.

Then number of sessions is the number of people who are looking at the rider app, interested in taking a trip if the price is right. Some sessions "convert", i.e., request a car, while others do not. The price is the biggest factor in determining whether a session converts — lower prices cause more conversion. The focus of this recitation will be understanding the causal impact of price on session conversion.

You can see the map for zones in Manhattan here, https://www1.nyc.gov/assets/tlc/images/content/pages/about/t and the maps for the other borroughs here: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page (scroll down to "Taxi Zone Maps and Lookup Tables")

[ ]:
```
# YOUR CODE HERE
```

To get a sense for the data, for location_id 249 (the west village in Manhattan — selected arbitrarily), plot the number of sessions versus the hour of the day, for each of the 4 saturdays in the data, all on one plot. Since the data isn't sorted by hour, it is easiest to plot using dots.

[ ]:
```
# YOUR CODE HERE
```

Make another plot where each combination of pickup_hour / pickup_date / location_id is a separate dot. Put the number of drivers on the x-axis and the number of sessions on the y-axis.

Notice how they are correlated with each other. Uber drivers know where demand will be high and are also guided by a heatmap in the driver app showing surge pricing. This allows supply and demand to be roughly matched, so that there are usually enough drivers to serve demand, but not too many extra drivers (having extra drivers is undesirable, because it means that people are spending time but no one is benefiting from the time being spent).

[ ]:
```
# YOUR CODE HERE
```

Unfortunately, the number of drivers is not always perfectly matched to the number of sessions. In other words, supply is not always balanced with demand.

This is essentially because life holds surprises — sometimes things are busier than usual for unexpected reasons.

When this happens, Uber raises the price. In 2017, this was done using the surge pricing system. This charges riders a multiple of the ordinary fare based on, roughly speaking, how many more sessions there are than can be served by the available drivers. If the surge multiple is 2, then riders pay twice the normal fare. If the surge multiple is 1, riders pay the normal fare.

This serves two important purposes. First, it compensates drivers more for being smart and hard-working enough to be available when demand was going to be high. That encourages them to continue to be smart and hard-working in the future. Second, since there aren't enough drivers for everyone to take a trip, some people with a session on the rider app aren't going to be able to ride. Raising the price separates the people who have other options (e.g., taking the subway, walking, waiting 10 minutes for more cars to arrive and prices to drop) from those that don't and gives the limited supply of cars to those that are most willing to pay for them.

The desire to ride is correlated with how balanced supply and demand are. To see this, make a plot of desire to ride (on the x-axis) and the ratio of sessions over drivers (on the y-axis). You should see that when the desire to ride is high, the number of sessions relative to the number of drivers is also high. This is because the same events that cause people to look at the uber app more than usual (rain, a big concert or sporting event that just finished, a public transit outage) also cause them to want to ride on Uber more than usual, and makes many of them willing to pay more money.

[ ]:
```
# YOUR CODE HERE
```

1.0.2 Part 2: Simulating the effect of prices on conversion

We are going to simulate prices set by the surge system and the effect that has on session conversion. We begin by defining a function, `conversion_rate`, that accepts a surge multiplier set in the hour and location_id, as well as the desire_to_ride value for that hour and location_id from the data we read in. In our simulation, this will give the conversion rate for our riders. (The conversion rate is the fraction of sessions that convert.)

[251]:
```
def conversion_rate(surge, desire_to_ride):
    return min(1,max(0,0.8 - (surge-1)/desire_to_ride))
```

Make a plot of the conversion rate versus surge multiplier, over the range of surge multipliers from 1 to 2, for three different values of the desire_to_ride: 0.5, 1, and 2.

You should see that when the desire_to_ride is high, price has a smaller impact on conversion.

[ ]:
```
# YOUR CODE HERE
```

Now define a surge pricing function. This searches for the smallest surge multiple (greater than or equal to 1) such that the available drivers in the location_id can serve all of the sessions that convert in that hour, according to a prediction for conversion rate. Because the desire_to_ride is unknown to the surge pricing system, use the average desire_to_ride in the dataset. For the purposes of this recitation, assume that a driver can do 3 trips in an hour.

Your function should start at a surge of 1 and then increase the surge multiple in increments of 0.1. It should stop when it finds the first price such that the number of sessions that would convert (at the average desire_to_ride in the dataset) is below 3 * the number of drivers.

If it reaches 3, stop, making 3 the highest multiple your algorithm will produce. (Since we aren't willing to raise the price any more, we would tell some riders who will willing to pay more than 3x that no cars are available.)

The first line of your solution should be: `def surge_multiple(drivers,sessions):`

Your function should return the surge multiple it finds.

[ ]:
```
# YOUR CODE HERE
```

Then use your function to calculate the surge multiple for each location_id and hour in the dataset and put it into your dataframe. Put this into a field called `surge`.

Also create another field called `conversion` that calculates the conversion rate at this surge multiplier, using the value of `desire_to_ride` in this hour and location_id.

[ ]:
```
# YOUR CODE HERE
```

Make a histogram of the surge multipliers in your dataframe. You should see that the most common surge multiplier is 1, none are below 1, and most are less than 1.5

[ ]:
```
# YOUR CODE HERE
```

Now plot the surge multiplier on the x-axis and the conversion-rate on the y-axis. We know that the surge price causes conversion to fall, based on `conversion_rate`, but your plot should show that conversion is actually *increasing* in the surge price for part of the domain of the plot.

Explain why this is. To support your answer, think about: * the effect that a high desire to ride has on the sessions / drivers ratio (recall the plot you made above) * the effect of the sessions / drivers ratio on surge (plot it, if it's helpful) * the effect that a high desire to ride has on conversion

[ ]:
```
# YOUR CODE AND EXPLANATION HERE
```

1.0.3 A/B experiment

One way to deal with the issue that you just saw above is to run an experiment. We will simulate an experiment where prices are randomly modified in each location_id by a small amount. When considering such experiments, it is important to design them in such a way that market participants are not harmed, by ensuring that the price changes are small and that guardrails are put in place.

With probability 50%, assign each location_id, hour, and date to a treatment group that is either 0 (control) or 1 (treatment). Store this in a field called `df['treatment']`.

Then, create a new surge multiplier, `df['new_surge']`, that is: * the same as the old surge multiplier for location_id / hour / date assigned to control * the old surge multiplier minus 0.2 for location_id / hour / date assigned to treatment (i.e., if you are a lucky rider in a treated location / hour / date, you get a 20% discount)

Note that this experiment offers a discount if the original surge multiplier was 1. Driver payments could be designed to be what they would have been without the rider discount. This reduces the impact on market participants.

[ ]:
```
# YOUR CODE HERE
```

Then calculate a new conversion rate, `df['new_conversion']`, for each location_id / hour / date based on the new surge multiplier.

[ ]:
```
# YOUR CODE HERE
```

Then, using a groupby on surge (before it was modified in the experiment), and using only the location / hour / dates in treatment, calculate the average of the conversion rates that results from the new surge multiplier. Put this a series called `conversion_treatment`.

Then do the same thing for control and put the result in `conversion_control`.

Plot `conversion_treatment` and `conversion_control` versus surge multplier. Then plot their difference versus the surge multiplier.

This difference is the "treatment effect" and quantifies the causal impact on conversion of lowering the surge multiplier by 0.1, as a function of the original value of the surge multplier. You should see that it is positive for all values of the surge multplier.

[ ]:
```
# YOUR CODE HERE
```