Draft version July 15, 2019
Preprint typeset using LaTeX style AASTeX6 v. 1.0

TAXI PROBLEM

Jeffrey A. Gritton
(Dated: July 15, 2019)

ABSTRACT

In this analysis we aim to determine the best method to maximize income by driving a Yellow Taxi in New York City. Utilizing the New York City Taxi & Limousine Commission data for Yellow Taxis and a series of assumptions detailed in the following sections we have determined the best two five-hour shifts to work are Mondays and Tuesdays starting at 12:00 AM. In this report we shall detail the assumptions and methods that have lead to this result and improvements to the data set that would improve its predictive capabilities.

1. INTRODUCTION

In this data science challenge we are given the scenario, “Imagine that you decide to drive a taxi for 10 hours each week to earn a little extra money. Explain how you would approach maximizing your income as a taxi driver." We are further instructed to use data from the New York City Taxi & Limousine Commission (TLC), specifically Yellow Taxi data from June of 2017[1]. From the wording of the scenario we assume that driving a taxi will be in addition to a full time job. We only assume that the full time job has flexible work hours so that we do not need to constrain our driving shifts to specific days or times. In order to maintain a sensible work-life balance and not become fatigued by working a single 10 hour driving shift in addition to our full time job we split the driving hours between two five-hour shifts. A secondary group of assumptions come from taxi driving being a part-time endeavor. In order to work as a Yellow Taxi driver one must either obtain their own medallion (e.g. own a Taxi and medallion) or lease/rent from a Yellow Taxi company. As a part-time driver we assume we will go with the lease/rent option. According to the article “How Much of Fare Do Taxi Drivers Keep?” by Shala Munroe[2] there are two options for leasing/renting a cab and medallion. The first is a flat daily fee ranging from \$ 75 to \$ 150 per day to rent a Taxi with medallion. The second option is to pay the owners 33% of the earned fares. We will focus on the latter option because the same article suggests that drivers pull long 10-12 hour shifts in a single day. If we are only driving 5 hours shifts twice a week it may become unlikely that we will be able to afford the rental costs resulting in loss of money. Lastly, we shall assume that in any shift we will not have control over which fares are taken. Unlike driving for Uber or Lyft where we can choose which fares we take, it is more likely that the majority of our fares will be gained by being hailed either on the road or at a cab stand. Because of this we will not consider drop off or pick up locations and adopt a method of driving where we take any fare available and wait at cab stands during down times. This will reduce the use of gas which must be considered in our daily costs by maximizing the amount of time driving with a fare.

2. METHODS AND RESULTS

In this challenge we are instructed to use data from the TLC from June of 2017 for Yellow Taxis. This data set contains 9,658,043 entries for taxi fares. For a list of the data columns contained within the dataset see Appendix A.

2.1. *SQL Methods*

Using SQL we select all data from Pick-up Date and Time and Drop-off Date and Time. From this information we determine the day of the week for each fare, the time of day that each fare began, and the length of time for each fare. We preferentially select fares that were paid using either credit or cash. We ignore “Unknown” and “Voided” trips because there is only a single “Unknown” fare and zero “Voided” fares. We ignore the 15,511 “Disputed” fares because it is uncertain if, how, or when these fares are refunded to the passenger. This only represents about 0.2% of the total data and is a reasonable loss. Lastly we ignore “No Charge” fares because it is uncertain how these relate to the data. If these are trips taken but not paid for they represent only a small loss in revenue due to fuel expenditure. Additionally, “No Charge” fares only represent 0.5% of the total data and is an acceptable loss. To expedite prototyping we import the Day of Week, Time of Day, Trip Distance, Payment Type, Fare Amount, and Tip Amount into a new table.

By inspecting the data we can find a number of outliers. These outliers are likely due to user error or internal errors in the hardware and/or software of the taxi-meter. The former error type presents itself in fares that have exceptionally large distances and/or time, no charged fare, exceptionally large fares, and minimum charged fare ($ 2.50). These errors are likely due to the driver activating the meter accidentally, neglecting to end a fare, or inputting a cash fare improperly. The latter error type presents itself with zero recorded distance and/or time or as trips that could not physically happen. The former error is likely due to time or distance not recording at all due to an internal error in the meter. The latter error can be more difficult to detect as the distance and time for the fares are not zero but if we calculate the average speed during the trip we find unlikely speeds. Some of these speeds exceed the speed of sound. For the above reasons we select only trips that have a non-zero trip time, a non-zero trip distance, a fare greater than $ 2.50, and an average trip speed below a reasonable speed limit of 60 MPH. Admittedly, having an average speed of 60 MPH in a city is unlikely, but any further assumptions to the maximum speed other than legal speed limits would be conjecture. Furthermore, to remove odd fares that may be driver input error we remove fare rates (dollars per mile) that are above a certain value. From the TLC website we can determine that the maximum, with minimum distance, would be a one mile trip from Manhattan to the JFK airport due to the $ 52 base fare. Although this fare is impossible it does give us an upper bound on fare rates. We select only rates below this threshold as anything above this is possibly user input error or an internal error. We select only fares lasting less than one hour to maintain resolution in time domain and reduce the effects concentrating high fare and long trips into hourly bins. With the above assumptions and filters we use the Fare Amount (FA), Tip Amount (TA), and Trip Distance (TD) to calculate the Fare Rate and Tip Rate per mile as follows:

$$FR = \frac{1}{N}\sum_i \frac{FA_i}{TD_i}$$

$$TR = \frac{1}{N}\sum_i \frac{TA_i}{TD_i}$$

where FR, TR, and N are the Fare Rate, Tip Rate, and number of entries within an hourly bin respectively. The TR is calculated using only fares paid by credit card as most cash tips are not reported. We also calculate the standard deviation on the Trip Distance, Fare Amount, and Tip Amount (σ_X). To calculate the uncertainties on FR and TR we use propagation of error which yields the following equations for the respective uncertainties:

$$\sigma_{FR} = FR\sqrt{\left(\frac{\sigma_{FA}}{FA}\right)^2 + \left(\frac{\sigma_{TD}}{TD}\right)^2}$$

$$\sigma_{TR} = TR\sqrt{\left(\frac{\sigma_{TA}}{TA}\right)^2 + \left(\frac{\sigma_{TD}}{TD}\right)^2}$$

Our final query returns 168 rows of data including the Day of Week, Time of Day, TD, σ_{TD}, FR, σ_{FR}, TR, σ_{TR} and is output to a comma separated file for further analysis using Python.

2.2. *Python Methods*

Using python we calculate the the Take Home Amount (TH) for a single hourly shift. The Take Home Amount is a function of the Fare Rate, Tip Rate, Trip Distance, vehicle fuel economy, and fuel costs. As we are asked to focus on June of 2017 we will also assume values for vehicle fuel economy and fuel costs from the same time. According to the New York State Energy Research and Development Authority the average cost of fuel in the state for June of 2017 was \$ 2.42 per gallon[3]. In 2017 the average fuel economy for vehicles in the United States was 24.9 miles per gallon according to the Environmental Protections Agency Automotive Trends Report[4]. From these values we calculate the cost of fuel per mile as \$ 0.097 per mile. To calculate the take home amount we implement the following for each hourly bin.

$$TH = (0.66FR + TR - 0.097)\,TD$$

with an uncertainty of

$$\sigma_{TH} = \sqrt{(0.66\sigma_{FR}TD)^2 + (\sigma_{TR}TD)^2 + \sigma_{TD}^2\,(0.66FR + TR - 0.097)^2}$$

We average over five one-hour bins through the week to determine the average earning potential of a single shift. From this we find a series of peaks in average earning potential throughout the week as seen in Figure 1. Each shift carries with it a risk and reward of low shift earnings and high shift earnings represented as the

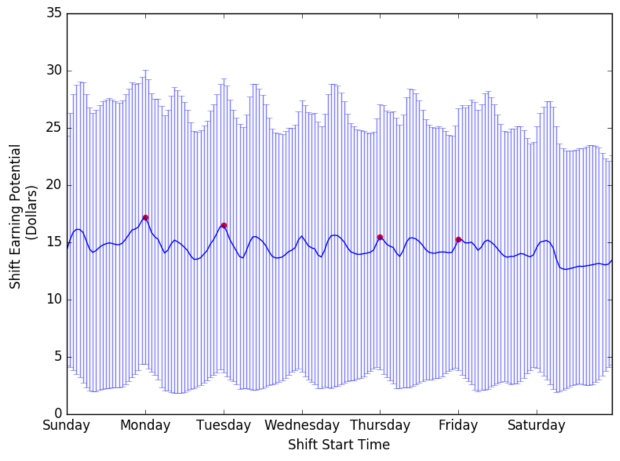

Figure 1. Five-hour shift averages representing the earning potential of a single shift. Blue line represents the shift earning potential while the red dots represent the best shifts to take.

the error bars in Figure: 1. If we want stable income we want shifts that are both high paying (peaks in Figure: 1) and have relatively lower risk. We can weight the Shift Earning Potential values of Figure: 1 by their own uncertainty. We find the peaks in Figure 1 that represent the shifts with the greatest earning potential. We select only peaks greater than the average and separated by at least 12 hours to minimize driver fatigue and allow for our full time job. From Figure 2 we determine the location of both peaks and troughs. Peaks represent the most stable earning potential while troughs represent the least stable. Ideally we'd like to choose shifts closest to the most stable shifts and furthest from least stable. We implement a k-dimensional tree to determine the nearest neighbors and distances between the highest earning potential shifts and the most and least stable shifts. We define each peak to most stable distance (dh) and each peak to least stable distance (dl). The most stable shifts to work will have a minimized value of dh/dl. From the

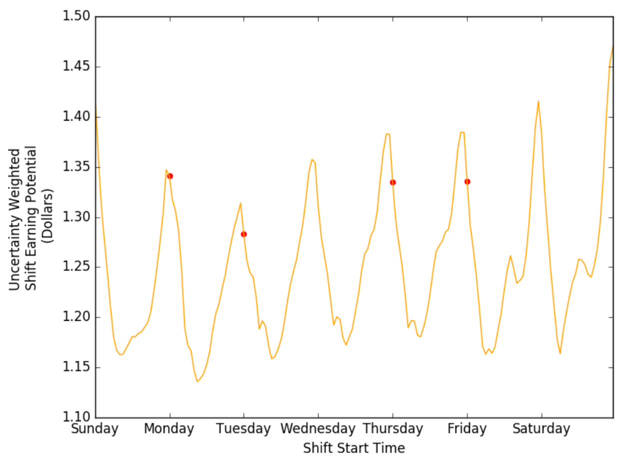

Figure 2. Five-hour shift averages weighted by uncertainty. Red points correspond to the best shifts to take.

resulting peaks we are left with only 4 shifts that have a $dh/dl < 1.0$. The four remaining shifts are Mondays, Tuesdays, Thursdays, and Fridays all beginning at 12:00 AM. Of these four shifts the two best to work due to high earning potential and small dh/dl value are Mondays and Tuesdays. These two shifts yield hourly potential earnings of $\$17.21 \pm 12.83$ and $\$16.49 \pm 12.85$ for Monday and Tuesday respectively. This would result in take home amount, before taxes, of $\$168.80 \pm 90.79$ each week.

3. SUMMARY AND DISCUSSION

Using the New York City Taxi & Limousine Commission data from June of 2017 and a combination of SQL and Python techniques we have concluded that to maximize earnings as a part-time driver for 10 hours per week we should work two five-hour shifts beginning at 12:00 AM on both Mondays and Tuesdays. We have come to this conclusion from the assumptions that we are renting a cab and medallion through a company that takes payment in the form of 33% of our daily fares and require the vehicle to be refueled. We incorporate these assumptions by using values for fuel economy and fuel prices from the same time frame as the data. We split the 10 hours of driving into two 5 hour shifts to reduce the risk of driver fatigue and and to more easily fit into a weekly schedule with a full-time job. In addition to the initial question to determine the best shifts to drive we were asked, “If you could enrich the dataset, what would you add? Is there anything in the dataset that you donâĂŹt find especially useful?” We believe a driver ID and transaction ID would enrich the dataset. This would allow for us to cross-reference fares paid by card or cash with voided and no charge fares. Because these fares represent a net loss due to cost of fuel they can have an impact on the daily take home amount. By cross-referencing transaction ID’s we would be able to keep entries paired fare and void entries in our analysis and reject any entries that are not paired. Additionally, such ID’s in concert with the Pick-up Location and Drop-off Location would allow for a training data set of routes from real shifts to suggest the best starting location for a shift through the implementation of a machine learning algorithm. From this dataset the only column that was not useful in either the final analysis or prototyping was the “Store and Forward Flag”. While we assume that this flag has its uses it was not particularly useful in this analysis.

REFERENCES

[1] NYC Taxi and Limousine Commission. "Trip Record Data June". 2017 https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_ tripdata_ 2017-06.csv [2] Shala Munroe "How Much of Fare Do Taxi Drivers Keep?" 2018. https://work.chron.com/much-fare-taxi-drivers-keep-22871.html [3] New York State Energy Research and Development Authority. "Monthly Average Motor Gasoline Prices". 2019 https://www.nyserda.ny.gov/Researchers-and-Policymakers /Energy-Prices/Motor-Gasoline/Monthly-Average-Motor-Gasoline-Prices

[4] Environmental Protection Agency. "Automotive Trends Report". 2019. https://www.epa.gov/automotive-trends/explore-automotive-trends-data

APPENDIX

A. APPENDIX TLC DATA

For this project we utilized data from the TLC focused on June of 2017 for Yellow Cabs. The data is organized under 17 categories as follows:

1. Vendor ID (VendorID)
   - A code indicating the TPEP provider that provided the record.
2. Pick-up Date and Time (tpep_ pickup_ datetime)
   - The date and time when the meter was engaged.
3. Drop-off Date and Time (tpep_ dropoff_ datetime)
   - The date and time when the meter was disengaged.
4. Passenger Count (Passenger_ count)
   - The number of passengers in the vehicle.
5. Trip Distance (Trip_ distance)
   - The elapsed trip distance in miles reported by the taximeter.
6. Pick-up Location (PULocationID)
   - TLC Taxi Zone in which the taximeter was engaged.
7. Drop-off Location (DOLocationID)
   - TLC Taxi Zone in which the taximeter was disengaged
8. Rate Code (RateCodeID)
   - The final rate code in effect at the end of the trip.
9. Store and Forward Flag (Store_ and_ fwd_ flag)
   - This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka âĂIJstore and forward,âĂİ because the vehicle did not have a connection to the server.
10. Payment Type (Payment_ type)
    - A numeric code signifying how the passenger paid for the trip.

11. Fare Amount (Fare_ amount)
    - The time-and-distance fare calculated by the meter
12. Extra Charges (Extra)
    - Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.
13. MTA Tax (MTA_ tax)
    - $ 0.50 MTA tax that is automatically triggered based on the metered rate in use
14. Improvement Surcharge (Improvement_ surcharge)
    - $ 0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.
15. Tip Amount (Tip_ amount)
    - This field is automatically populated for credit card tips. Cash tips are not included.
16. Tolls Amount (Tolls_ amount)
    - Total amount of all tolls paid in trip.
17. Total Amount Charged (Total_ amount)
    - The total amount charged to passengers. Does not include cash tips.

B. APPENDIX SQL QUERY

```
CREATE TABLE YTD3
select hour(tpep_pickup_datetime) as TOD,
dayofweek(tpep_pickup_datetime) as DOW,
time_to_sec(timediff(tpep_dropoff_datetime, tpep_pickup_datetime)) as Trip_Time,
trip_distance,
payment_type,
fare_amount,
tip_amount
from yellow_tripdata_2
;

select DOW,
TOD,
ROUND(avg(trip_distance),2) as TD,
ROUND(stddev_samp(trip_distance),2) as sig_TD,
ROUND(avg(fare_amount/trip_distance),2) as Fare_Rate,
ROUND(avg(fare_amount/trip_distance)*sqrt(pow(stddev_samp(fare_amount)/avg(fare_amount),2)
+pow(stddev_samp(trip_distance)/avg(trip_distance),2)),2) as sig_Fare_Rate,
ROUND(avg(if(payment_type=1, tip_amount/trip_distance,null)),2) as Tip_Rate,
ROUND(avg(if(payment_type=1, tip_amount/trip_distance,null))*
sqrt(pow(stddev_samp(if(payment_type=1, tip_amount, null))/avg(if(payment_type=1, tip_amount, null)),2)
+pow(stddev_samp(trip_distance)/avg(trip_distance),2)),2) as sig_Tip_Rate
from YTD3
```

```
where Trip_Time_s > 0.0
and Trip_Time/3600 < 1.0
and trip_distance >0.0
and trip_distance/(Trip_Time/3600) < 60
and fare_amount>2.50
and fare_amount/trip_distance <52
and (payment_type = 1 or payment_type = 2)
group by DOW, TOD
order by DOW, TOD
;
```

C. APPENDIX PYTHON SCRIPT

```
#!/usr/bin/env python3

#SQL/Python Taxi Problem

def choose_file():
import tkinter as tk
from tkinter import filedialog

root = tk.Tk()
root.withdraw()
root.filename =  tk.filedialog.askopenfilename(initialdir =
"~/home/zerostrain/Documents/SQL_Practice/",title =
"Select file",filetypes =(("all files","*.*"),("jpeg files","*.jpg")))

return root.filename

def TH(data):
import numpy as np
#Calculate take home amount from taxi matrix
# (0.66*Fare_Rate + Tip_rate - Dollar/mile)*TD
# Fare_Rate = taxi[4]
# Tip_Rate = taxi[6]
# Dollar/mile = DPG/MPG
# TD = taxi[2]

MPG=24.9
DPG=2.42

take_home=(0.66*data.loc[:,"Fare_Rate"].values+data.loc[:,"Tip_Rate"].values
-(DPG/MPG))*data.loc[:,"TD"].values

return take_home

def sig_TH(data):
import numpy as np

A=0.66
MPG=24.9
```

```
DPG=2.42
B=DPG/MPG

x=((data.loc[:,"sig_Fare_Rate"].values)**2)*(A*data.loc[:,"TD"].values)**2
y=((data.loc[:,"sig_Tip_Rate"].values)**2)*(data.loc[:,"TD"].values)**2
z=((data.loc[:,"sig_TD"].values)**2)*((A*data.loc[:,"Fare_Rate"].values+
data.loc[:,"Tip_Rate"].values-B)**2)

sig_Take_Home=np.sqrt(x+y+z)

return sig_Take_Home

def DT(data):
#Create unique day time notation that can be plotted in cronological order as a single array.
import numpy as np

date_time = 24*(data.loc[:,"DOW"].values-1)+data.loc[:,"TOD"].values


return date_time

def SA(TH,sTH):
#Find averages for 5 hour block shifts.
import numpy as np
SA=np.zeros(len(TH))
sSA=np.zeros(len(TH))
for i in range(len(TH)):
SA[i]=np.average(TH.take(range(i,i+6), mode='wrap'))
sSA[i]=np.sqrt(np.sum(sTH.take(range(i,i+6),mode='wrap')**2))/5
return SA, sSA

def nearest_neighbor(date_time, shift_ave, sig_shift_ave, peaks, peak_h, peak_l):
from scipy.spatial import KDTree
#Find nearest neighbors for all peaks compared to shift_ave/sig_shift_ave
#
dh, ih = KDTree(list(zip(date_time[peak_h],shift_ave[peak_h]/
sig_shift_ave[peak_h]))).query(list(zip(date_time[peaks],shift_ave[peaks]/sig_shift_ave[peaks])))
dl, il = KDTree(list(zip(date_time[peak_l],shift_ave[peak_l]/
sig_shift_ave[peak_l]))).query(list(zip(date_time[peaks],shift_ave[peaks]/sig_shift_ave[peaks])))

return (dh/dl).argsort()[:4];

def main():
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.signal as sps
plt.cla()
day_of_week=['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
```

```
fl_nm=choose_file()
data=pd.read_csv(fl_nm)

take_home=TH(data)
date_time=DT(data)

sig_Take_Home=sig_TH(data)
shift_ave, sig_shift_ave=SA(take_home,sig_Take_Home)

#Use signal peak finder to determine local maxima value separated by at least 12 hours
#Select a minimum height of the average to only select highest peaks
peaks=(sps.find_peaks(shift_ave,distance=12+1, height=np.average(shift_ave)))[0]
peak_h=(sps.find_peaks(shift_ave/sig_shift_ave,distance=12+1))[0]
#Find troughs
peak_l=(sps.find_peaks(-1*shift_ave/sig_shift_ave,distance=12+1))[0] #,distance=12+1


good=peaks[nearest_neighbor(date_time, shift_ave, sig_shift_ave, peaks, peak_h, peak_l)]


nearest_neighbor2(date_time, shift_ave, sig_shift_ave, peaks, peak_h, peak_l)

fig, ax1 = plt.subplots(nrows=1, ncols=1, sharex=True)
#Earning potential
markers, caps, bars = ax1.errorbar(date_time, shift_ave,yerr=sig_shift_ave, color='blue')
ax1.scatter(date_time[good], shift_ave[good], color='red')
#Visualization edits
[bar.set_alpha(0.5) for bar in bars]
[cap.set_alpha(0.5) for cap in caps]

ax1.set_xlim(left=np.min(date_time), right=np.max(date_time))

ax1.set_ylabel('Shift Earning Potential \n (Dollars)')
plt.xticks(range(0,len(date_time), 24), day_of_week)
ax1.set_xlabel('Shift Start Time')

plt.tight_layout()
plt.savefig('SA_ave.png')


fig, ax2 = plt.subplots(nrows=1, ncols=1, sharex=True)
#Weighted earning potential
ax2.plot(date_time, shift_ave/sig_shift_ave, color='orange')
ax2.scatter(date_time[good], shift_ave[good]/sig_shift_ave[good], color='red')

ax2.set_xlim(left=np.min(date_time), right=np.max(date_time))

ax2.set_ylabel('Uncertainty Weighted \n Shift Earning Potential \n (Dollars)')
```

```
plt.xticks(range(0,len(date_time), 24), day_of_week)
ax2.set_xlabel('Shift Start Time')

plt.tight_layout()
plt.savefig('W_SA_ave.png')

print('The best 5 hour shifts to work are: ')

for i in good:
dow = day_of_week[data.loc[i,'DOW']-1]
ampm=('AM' if data.loc[i,'TOD'] <12 else 'PM')
time = (data.loc[i,'TOD'] if (data.loc[i,'TOD'] <12) else (data.loc[i,'TOD']-12))

print(dow, time, ampm,shift_ave[i],sig_shift_ave[i])

return;
```