STAT 405 Final Presentation

By Ella Gruen, Anthony Pagas, John Chumlea, Shien Zhu and Samuel Negus

Introduction

Dataset: **NYC Taxis**

- Taxi trip records exclusively in NYC with variables such as pickup + dropoff time, trip details, passenger count, and vendor ID

Research Question: **Which taxi vendor is the most time efficient to take during any day of the week?**

Statistical Computation Method: **Group-wise aggregation** , we grouped trip data by vendor ID and day of the week and calculated the mean of minutes per mile within each group. Ex: “On Tuesday around 12 AM, CMT taxis traveling between 0-1 mile averaged about 5.39 minutes per mile”

Data Example

	vendor_id	distance_bin	hour	dayofweek	minutes_per_mile
1	CMT	0–1mi	0	0	5.617163863057147
2	CMT	0–1mi	0	1	5.384884046247732
3	CMT	0–1mi	0	2	5.566144601759156
4	CMT	0–1mi	0	3	5.583951529747951
5	CMT	0–1mi	0	4	6.491775980278212
6	CMT	0–1mi	0	5	7.035207710896743
7	CMT	0–1mi	0	6	7.658428971013421
8	CMT	0–1mi	1	0	4.981453313955257
9	CMT	0–1mi	1	1	5.268372882074294
10	CMT	0–1mi	1	2	5.579129417837019

Data

Source/Size: Kaggle (https://www.kaggle.com/datasets/chilam/nyctaxis), 28.85 GB

Cleaning:

- Dropped all trip distances that were not a valid number
- Converted “pickup_datetime” variable into a datetime object and dropped any corrupt time formats
- Removed trips with zero or negative distance or time

Computation:

- We ran 12 parallel jobs using 2 GB of request disk/memory and 1 CPU. All jobs lasted about 10-20 minutes

Key Points of Coding

- efficiency.py
  - 2GB/ csv, so all to doing computation for each creates an issue in the loop.
  - Idea: Using chunk library, we can compute 100,000 rows each iteration.
- Parallel job
  - Created a python environment, similar to the way we were able to run Python scripts in HW4.
- Issues:
  - The .csv file is so large that the compute node has a difficult time doing computation -> change python library (panda -> chunk)
  - HTC doesn't have Python environment to run our code -> Create Python environment, run code in the environment

Results

Overall Vendor Efficiency (Lower = Faster)

		vendor_id	minutes_per_mile
1	1	VTS	5.1159957909080 6
2	0	CMT	5.6303473699478 03

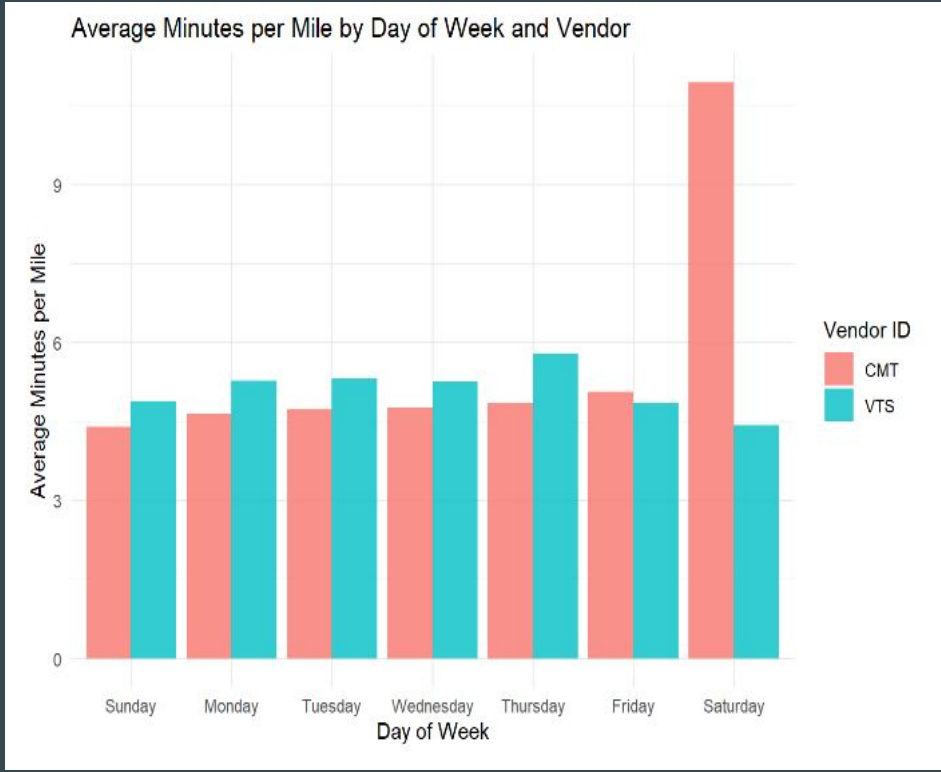
Vendor Efficiency By Day Of Week

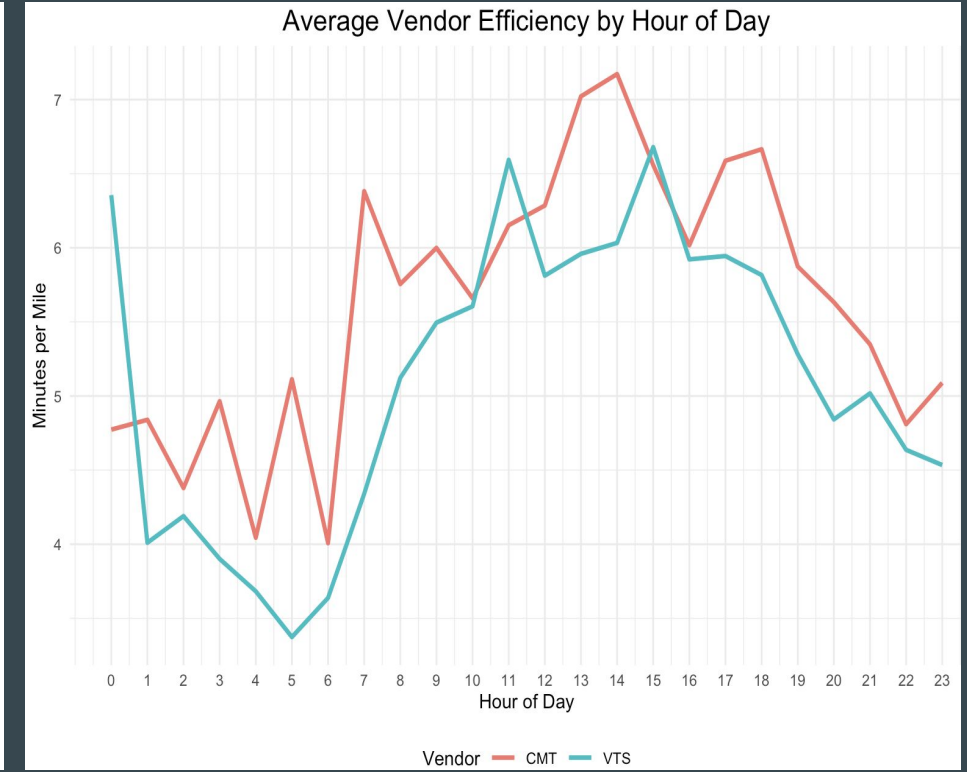
	vendor_id	dayofweek	minutes_per_mile
1	CMT	0	4.399204986834561
2	CMT	1	4.653656065358734
3	CMT	2	4.741001525048589
4	CMT	3	4.772009267280266
5	CMT	4	4.838987150324669
6	CMT	5	5.062342596746403
7	CMT	6	10.94522999804137
8	VTS	0	4.889820306720169 5
9	VTS	1	5.279960543313334
10	VTS	2	5.319016756058505
11	VTS	3	5.263004729268868
12	VTS	4	5.795670357851397
13	VTS	5	4.840227497428352
14	VTS	6	4.424270345715 79

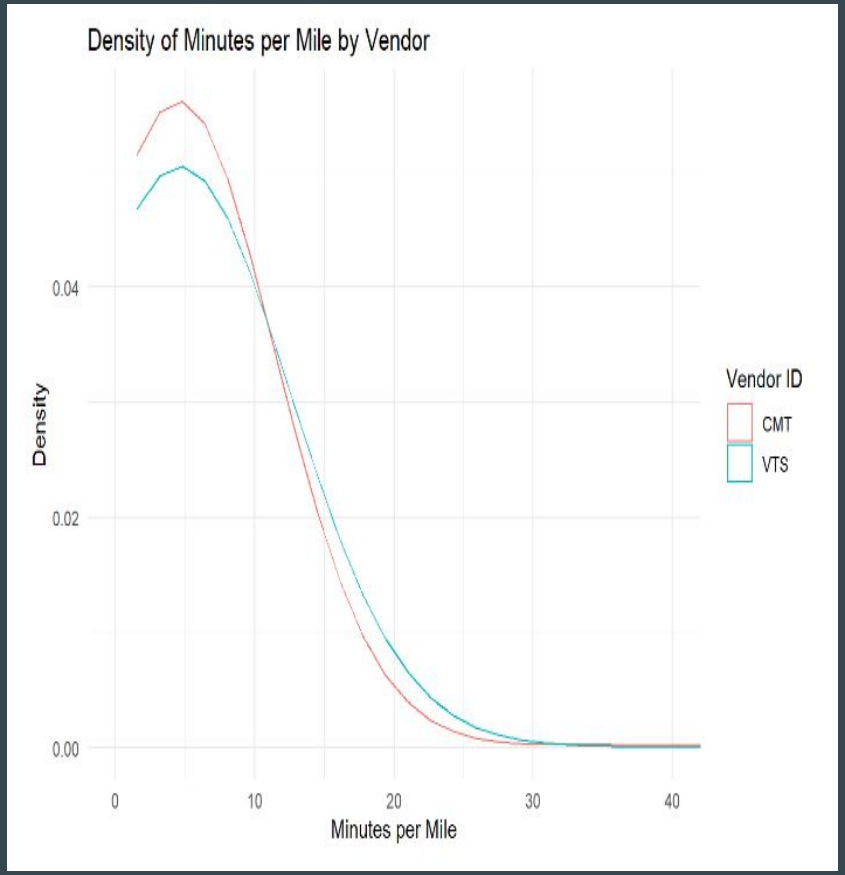
Vendor Efficiency By Distance Bin

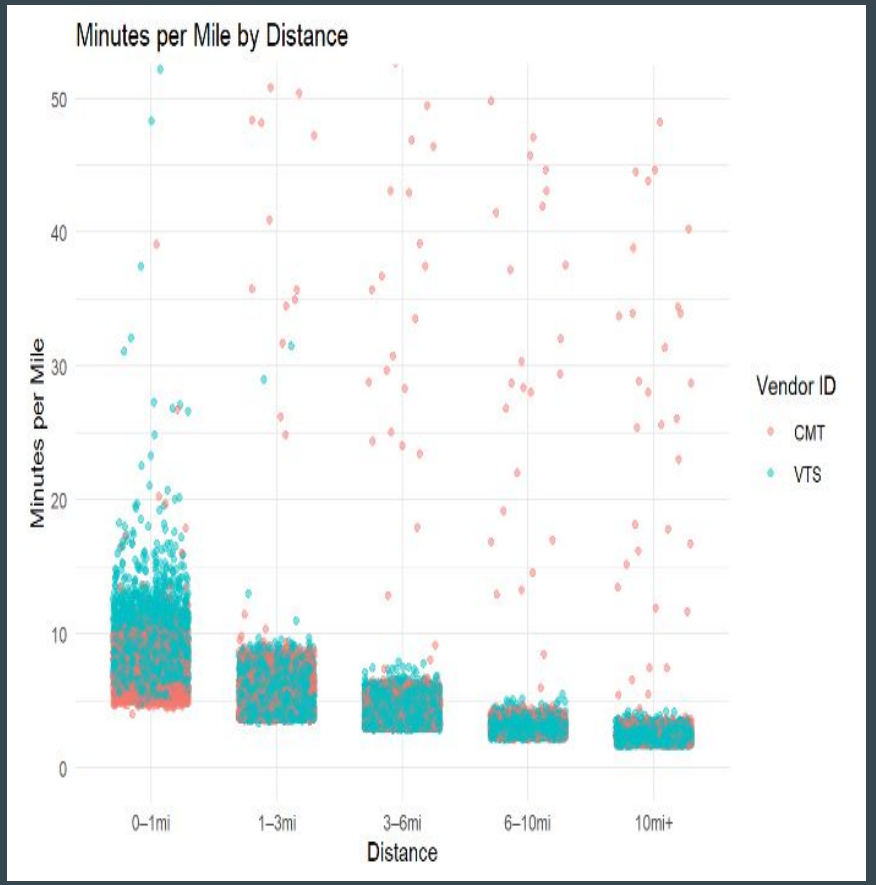
	vendor_id	distance_bin	minutes_per_mile
1	CMT	0–1mi	10.197733785041486
2	CMT	10mi+	2.7193125560845606
3	CMT	1–3mi	6.494375570967718
4	CMT	3–6mi	5.158493985672197
5	CMT	6–10mi	3.581820951973047
6	VTS	0–1mi	10.20690240840536
7	VTS	10mi+	2.3339834471347545
8	VTS	1–3mi	5.67027205467496
9	VTS	3–6mi	4.353812532711422
10	VTS	6–10mi	3.015008511613799

Graphical Representations






Density of Minutes per Mile by Vendor
Density
0.04
0.02
0.00
0
10
20
30
40
Minutes per Mile
Vendor ID
CMT
VTS


Minutes per Mile by Distance
Minutes per Mile
50
40
30
20
10
0
0–1mi
1–3mi
3–6mi
6–10mi
10mi+
Distance
Vendor ID
CMT
VTS

Conclusion

- VTS is the most efficient vendor in our NYC Taxi Data, averaging about 5 minutes and 6 seconds per mile
- Even though CMT vendor beats out VTS in minutes per mile 5 out of the 7 days of the week, VTS beats CMT in every distance bin category except for 0-1 mile
- We struggled a lot with disk space and python processing
- Prevent data inconsistencies by implementing distance bin (ex: if CMT had the second fastest time of day but only had one trip but VTS had the fastest time of the day but had multiple trips, CMT has a bias)