# Software Project Group N

<Assima Amangeldina, Ceyla Kaya, Ziyi Dong, Sabina Nurseitova>

## Task 1

{

`calculate_stats.py`

- Number of passengers, fare amount, total amount, tips amount expressed in number + $

`count_trips.py`

- Count the number of trips for each some Pick up location and dropoff zone, the zones should be expressed rather than their codified version

`calculate_speed.py`

- Create a function that calculate the speed of a trip in Kmh def read_file(file_path: str) -> any:

First, I accessed "dataset_small.txt" with the ‘open()’ function. I then read the first line to obtain the column names which would serve as keys for our dictionaries. Following this, I iterated over each line of the file, splitting the data at commas and applying the appropriate type conversion—‘float’ for numerical fields and ‘datetime’ for date fields. Finally, I compiled the data from each line into a dictionary and collected these dictionaries into a list to complete the structured dataset.

1. def calculate_stats(*data*: List[Tuple[int, float, float, float]]) -> Tuple[float, float, float]:

At very first, I inserted the def read_file into the code, so the program could process all the information from the txt file. Then, as my second step, I instructed the code to read all of the lines of information and store them as a “lines” variable and if not lines, then indicate that this document does not store any piece of information. Then, I assigned the target columns, so the code would only extract data that is needed, such as  number of passengers, fare amount, total amount and tip amount because otherwise, the code would process all the data from the txt file. After that, I initialized the dictionary for storing the data about maximum, minimum and average values for target columns and then assigned to retrieve them. At the end, I showed a file path and assigned a format, according to which the results should be showed.

2. def calculate_speed(*data*: List[Tuple[str, str, float]]) -> Tuple[float, float, float]:

I imported datetime, timedelta, defaultdict and list,tuple. After opening the file in read mode, first I determined which values of the dataset were the pickup times, dropoff times, zone ids, and distances. Then I replaced zone ID’s with their respective names. After that, I made sure pickup times and dropoff times were aligned in 24 hour time (hh:mm:ss or hh:mm) and assigned temporary min and max speed values. Then I converted the distance and durations into kmh format. After that I got the averages of these values and printed them.

3. def count_trips(*data*: List[int], *zones*: Dict[int, str]) -> Dict[str, int]:

First, I wrote a function that reads the original data structure through “`csv.DictReader`” and extracts the pick up zone IDs while creating a list of those IDs. Then I wrote a function that reads through the CSV file and creates a dictionary with zone IDs as key and zone names as value. Those two created data structures need to be the parameters of the count_trips function that was asked by the task. Then I wrote a function that counts the trips from each zone mentioned and returns a dictionary with zone names as key and count of trips in that zone as value. I did this by using the .get() function and counting the frequency.

# Task 2

quick_sort.py

**A Python function that** use two of the sorting algorithms seen in class on Number of passengers, fare amount, total amount, tips amount, and the speed and time for each trip you found on the speed calculation function, choose at least one **non quadratic**

heap_sort.py

**Return time of execution of each execution, compare and comment**

You'll have to apply the right sorting algorithm to your data, so you'll have to check how your data are made, you can apply more than two and just submit what you want

## quick_sort.py

After reading the file, I defined a function *aligndata* to align the keys which represent the passenger counts, fare amounts etc with their respective data. After that I assigned a random pivot to do quicksort on the dataset. I collected values smaller, equal and greater than the pivot in separate lists and after the sorting, calculated the execution time. Finally I printed and called quicksort on the datasets.

## heap_sort.py

After reading the file, we defined a function align_data and dictionary key_mappings to connect each type of data to the corresponding key in the dataset. Then, we check, if the key exists inside of the database, if so, it should return values as floats.Then, we defined heapsort and heapify functions, indexes of the root node in the array and algorithm for the left and right child of the node. Then, we instructed on how to find a largest value in the dataset.Additionally, we defined a max_heap function and assigned the code to put the largest(maximum) values at the end of an array. Then, we we calculated the execution time. Finally, we printed and called heapsort on the datasets.

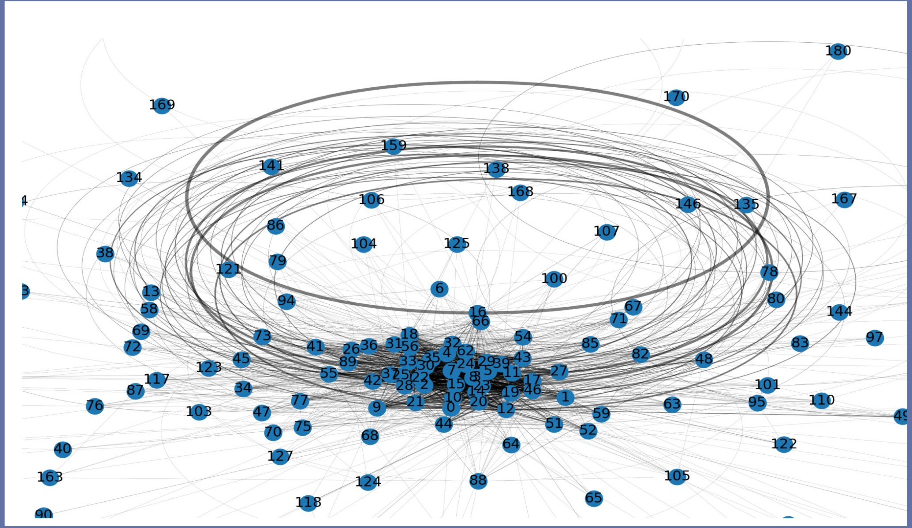
# Task 3

build_graph.py

**A Python function** that build the graph between each pickup and dropoff location, the size of your edge should be proportional to the number of trips between two points. You can use Networkx to build your graph.

Find the connected components in your complete network, treat the graph as undirected. You can use Networkx also in this case (basic level) or use the visiting algorithm DFS and BFS and compare the execution time

}



In this code, we built a graph by activating a virtual environment and using Networkx and matplotlib to visualize our code.

We used the dataset to gather our nodes and the trip counts to gather our edges and their weights in this graph. To avoid having more than one of the same node (location), we adjusted our code to eliminate duplicates.

The amount of trips made in between two nodes (pickup and dropoff locations) determine the weight of our edges. To do this we defined an *edge_width* value that we later used to determine the style of the edge in the networkx function, nx.draw_networkx_edges(graph, pos, width=edge_width, alpha=0.5)