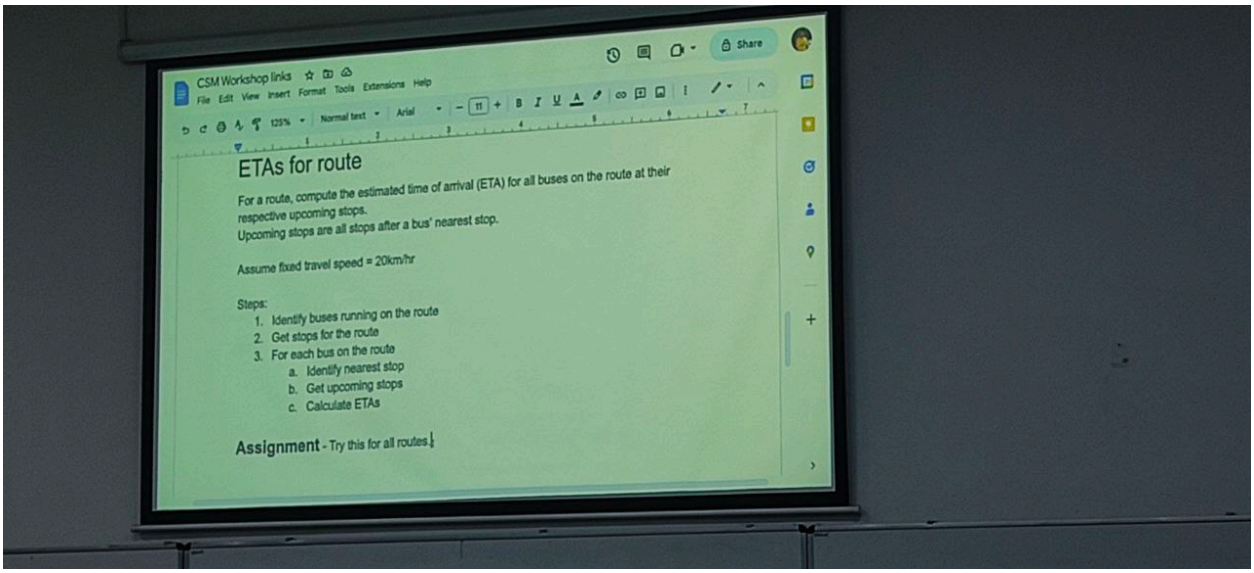
Problem Statement -



Final.ipynb - Step by Step Solving the Problem Statemen using python libraries

To make Python more user interactive we use flask which help to run python in back for a server in a system

ETA-bus.zip - this contain flask project, the flask project having routes.py which is running behind the flask interface

Screenshots from final.ipynb

All four data frames - routes, trips, stops, stop_times

routes:

	agency_id	route_id	route_long_name	route_short_name	route_type
0	DIMTS	0	764MSTLDOWN	NaN	3
1	DIMTS	1	102STLDOWN	NaN	3
2	DIMTS	2	819STLDOWN	NaN	3
3	DIMTS	3	138UP	NaN	3
4	DIMTS	4	978STLDOWN	NaN	3

trips:

	route_id	service_id	trip_id	shape_id
0	0	1	0_06_00	NaN
1	0	1	0_06_10	NaN
2	0	1	0_06_20	NaN
3	0	1	0_06_30	NaN
4	0	1	0_06_40	NaN

stops:

	stop_id	stop_code	stop_name	stop_lat	stop_lon	\
0	0	DC4539	Narela Terminal	28.852004	77.088059	
1	1	DC4540	Police Station Narela	28.853366	77.088535	
2	2	DC4541	Safiyabad Crossing	28.855266	77.089730	
3	3	DC4542	Ramdev Chowk Pithori Jhori	28.857752	77.092856	
4	4	DC4543	Narela A-6 / CPJ College	28.857652	77.095746	

	zone_id
0	0
1	1
2	2
3	3
4	4

stop_times:

	trip_id	arrival_time	departure_time	stop_id	stop_sequence
0	0_06_00	06:00:00	06:00:00	3127	0
1	0_06_00	06:01:13	06:01:13	2620	1
2	0_06_00	06:07:27	06:07:27	2020	2
3	0_06_00	06:11:36	06:11:36	2021	3
4	0_06_00	06:16:38	06:16:38	4120	4

Step 1 -

Step 1 -> Identify Buses running on route {per_route}

```
from datetime import datetime

# Get all buses on a particular route
all_buses = merged_df[merged_df['route_id'] == per_route]

# Sort the buses by arrival time
all_buses = all_buses.sort_values('stop_sequence')

# Get unique trip IDs
all_trips_id = all_buses['trip_id'].unique()

# Initialize variables for counting buses and repeat buses
count = 0
rep_bus = 0

# Initialize empty lists to store arrival and departure times
time1 = []
time2 = []

# Get the first and last arrival time for each trip
for i in range(len(all_trips_id)):
  all_busess = all_buses[all_buses['trip_id']==all_trips_id[i]]
  time1.append(all_busess['arrival_time'].reset_index(drop=True)[0])
  time2.append(all_busess['arrival_time'].reset_index(drop=True)[len(all_busess['arrival_time'])-1])

# Define the bus times
trip_id = all_trips_id
arrival_time = time1
departure_time = time2

# Convert the times to minutes since midnight
arrival_minutes = [int(t.split(':')[0]) * 60 + int(t.split(':')[1]) for t in arrival_time]
departure_minutes = [int(t.split(':')[0]) * 60 + int(t.split(':')[1]) for t in departure_time]

# Sort the points by arrival time
points = sorted(zip(trip_id, arrival_minutes, departure_minutes), key=lambda x: x[1])

# Initialize the bus schedule
bus_schedule = []

# Loop through the points and assign them to buses
for point in points:
    assigned = False
    # Try to assign the point to an existing bus
    for i, bus in enumerate(bus_schedule):
        if bus[-1][2] <= point[1]:
            # If the bus can make it to the point before its arrival time, assign the point to the bus
            bus_schedule[i].append(point)
            assigned = True
            break
    # If the point could not be assigned to an existing bus, add a new bus
    if not assigned:
        bus_schedule.append([point])

# Print the number of buses needed
print(len(bus_schedule))


print(f'in the route {per_route} have atleast {len(bus_schedule)} buses')
print(f'in the route {per_route} have atmax {len(all_trips_id)} buses')
```

```
2
in the route 5555 have atleast 2 buses
in the route 5555 have atmax 4 buses
```

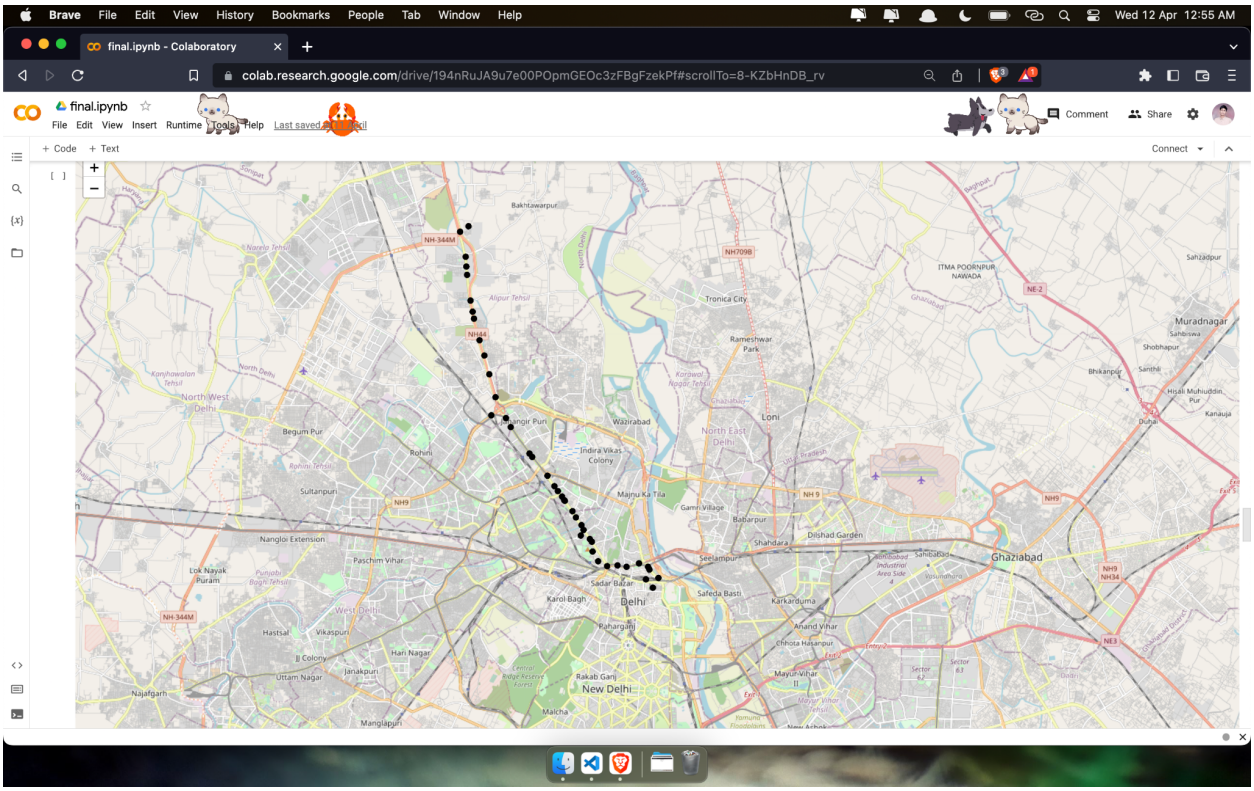
Step 2 ->

Step 2 -> Get stops for the route

```
[ ]   # Get unique stop IDs for the route
      all_stops_route = all_buses['stop_id'].unique()

      # Print all stops and total number of stops for the route
      print(f"""
      All stops for the route {per_route}
      {all_buses['stop_id'].unique()}

      Total Stops on route {per_route} - {len(all_stops_route)}""")
```

```
All stops for the route 5555
[13861 12682 10545 11097 11096 10547 10548 10549 10550 10022 10023 10024
 14739 10026 13707 10228 13925 10032 10033 10034 10035 14705 10036 10037
 10038 10039 13766 13836 14732 10041 10042 10043 10867 10045 10046 10047
 10101 10102 10702 14761 10339]

Total Stops on route 5555 - 41
```

Plot all stop points in a map



Step 3 ->

3.1 -> Identify nearest stop

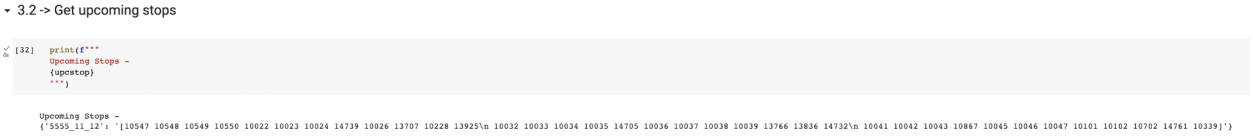
for accessing the each bus we use trip_id
assuming trip_id is unique for the each bus

To find the nearest point to the bus we use the time - current time and the time from which bus leave from previous stop ['departure_time'] and the bus reached to next stop ['arrival_time']
We will check which one is more close to current time and find the stop_id of that stop which is closer to bus

first we are checking
what is the next stop of the bus using current time and the arrival time of next stop of the bus,
from the next stop we use the coordinates to check which stop is closer to the bus

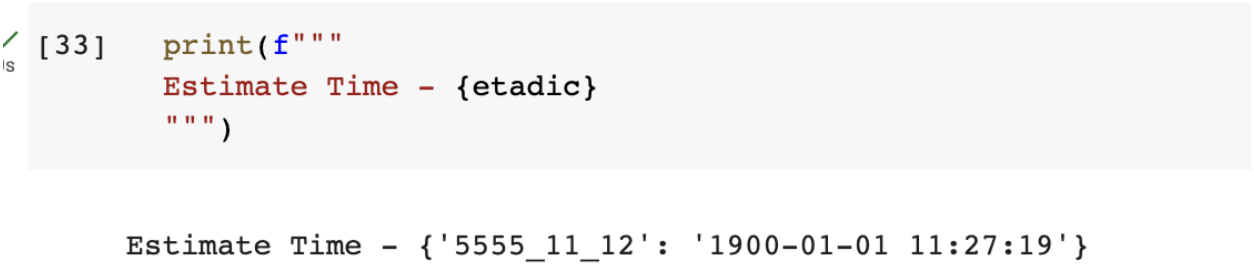
```
# Importing datetime module to work with time values
from datetime import datetime

# Creating empty dictionaries to store bus stop information and ETA values
etadic = {}
upcstop = {}
alldic = {}

# Looping through all the trips in the data
for i in range(len(all_trips_id)):
    # Getting the ID of the current trip
    single_trip_id = all_trips_id[i]

    # Getting all the stops for the current trip
    all_stops_single_bus_route = all_buses[all_buses['trip_id'] == single_trip_id]

    # Checking where the bus currently is and what the previous and next stops are
    for j in range(1, len(all_stops_route)):
        pre_dep_time = all_stops_single_bus_route['departure_time'].reset_index(drop=True)[j-1]
        curr_time = curr_time_str
        nxt_arr_time = all_stops_single_bus_route['arrival_time'].reset_index(drop=True)[j]

        # Checking if the current time is between the previous and next stop times
        if pre_dep_time < curr_time_str and curr_time_str < nxt_arr_time:
            pre_stop = all_stops_route[j-1]
            next_stop = all_stops_route[j]

            # Converting the time values to datetime objects to perform calculations
            pre_dep_time = datetime.strptime(f"{pre_dep_time}", "%H:%M:%S")
            curr_time = datetime.strptime(f"{curr_time}", "%H:%M:%S")
            curr_time_r = curr_time
            nxt_arr_time = datetime.strptime(f"{nxt_arr_time}", "%H:%M:%S")

            # Calculating the distance from the previous stop to the bus's current location
            dis_pre_stop = (curr_time - pre_dep_time).seconds * ((20*1000)/3600)  # d = t * s, where s = 20 m/s
            print(curr_time - pre_dep_time)
            print(type(curr_time))
            print(type(curr_time - pre_dep_time))

            # Calculating the distance from the bus's current location to the next stop
            nxt_arr_stop = (nxt_arr_time - curr_time).seconds * ((20*1000)/3600)  # d = t * s, where s = 20 m/s

            # Checking which stop is closer to the bus's current location and printing its details
            if dis_pre_stop < nxt_arr_stop:
                print(f"""For the Bus trip_id {single_trip_id}
                        {all_stops_route[j-1]} stop is closest point
                        Name of the Stop - {(all_buses['stop_name'].loc[all_buses['stop_id']==all_stops_route[j-1]]).unique()}
                        and coordinates points
                        stop_lat - {(all_buses['stop_lat'].loc[all_buses['stop_id']==all_stops_route[j-1]]).unique()}
                        stop_lon - {(all_buses['stop_lon'].loc[all_buses['stop_id']==all_stops_route[j-1]]).unique()}
                        Time - {pre_dep_time}

                        if
                        """)
```

Step 3.2->



List of all upcoming stops starting from the next upcoming stop

Step 3.3->



Estimate time for the each - the time when the bus reach to the next upcoming stop

Flask - taking two inputs from user

Bus Estimate time

Route: 5555

Current Time: 11:51:30

Submit

Step 1. Identify the buses running on the route 5555

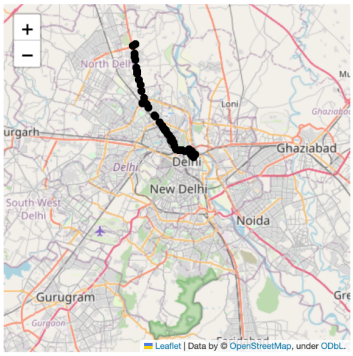
the buses running on the route 5555 is **atleast 2**

Step 2. Get Stops in the route 5555

the total number of stops in the route is 41

All stops list for the route 5555

[13861 12682 10545 11097 11096 10547 10548 10549 10550 10022 10023 10024 14739 10026 13707 10228 13925 10032 10033 10034 10035 14705 10036 10037 10038 10039 13766 13836 14732 10041 10042 10043 10867 10045 10046 10047 10101 10102 10702 14761 10339]



Step 3. For each bus on the route

Step 3. For each bus on the route

1. Identify nearest stop

the nearest stop is **{'5555_11_38': '10547', '5555_11_12': '10026'}**
the starting stop of route is 13861 - ['Bakuli Temple']
the ending stop of route is 10339 - ['Old Delhi Railway Station']
if the nearest stop is empty that means at this time no bus is in the route

2. Upcoming Stops -

{'5555_11_38': '[10547 10548 10549 10550 10022 10023 10024 14739 10026 13707 10228 13925\n 10032 10033 10034 10035 14705 10036 10037 10038 10039 13766 13836 14732\n 10041 10042 10043 10867 10045 10046 10047 10101 10102 10702 14761 10339]', '5555_11_12': '[10026 13707 10228 13925 10032 10033 10034 10035 14705 10036 10037 10038\n 10039 13766 13836 14732 10041 10042 10043 10867 10045 10046 10047 10101\n 10102 10702 14761 10339]'}

3. Expected time to reach

{'5555_11_38': '1900-01-01 11:53:19', '5555_11_12': '1900-01-01 11:52:48'}

Steps to run Flask project on your system

1. Install the required python library files and dataset in your system

pip install -r requirements.txt
python3 install_dataset.py run

run this comment in the terminal of project directory

2. Run flask project

python app.py run

3. Open the localhost - a server will be created on the local host [http://127.0.0.1:5000]

Now you can enter any value in the project and see the desired results

Note – For the first time flask project will take

Note – Make sure dataset is in the right directory

Dataset link – https://github.com/kshitijsriv/csm-gtfs-workshop/raw/master/GTFS.zip?raw=true