1 Implementation

1.1 Data Collection

There is no historical data for bus arrival times available online, so all the data that I will use to train and test my models has to be manually collected.

The following bus routes were chosen to collect data from: 452, 9, 52, 267, 277, 7, 6, 14, 35, 37, 69, 328. Of these routes, the 52, 6, 14, 37 and 69 bus routes are twenty four hour bus routes and the 267, 277, 37 and 69 bus routes do not travel through Zone 1. This allows there to be variety in the travel routes, traffic conditions and times of travel and so the data collected is not heavily biased towards particular areas of London. There is also some overlap in stops between the bus routes, for example the 52 to Victoria and the 452 to Vauxhall have a 23 stop (and therefore, also route) overlap from "Station Terrace" to "Knightsbridge Station / Harrods". This ensures information on journey times between stops doesn't rely on data from just one bus route.

1) Call Countdown API: The Countdown API is called for each stop on the specified route and returns a JSON of the form [[URA array],[Prediction Array]]. An example is shown in Figure **??**, where Countdown was called for bus route 9, stop code 490010357F (commonly known as "North End Road").

Images/Countdown-result.png

Figure 1: Countdown API response

Each item in the *Prediction* array corresponds to each of the response items as stated previously. This data is then converted into a list of dictionaries, where each dictionary represents a single vehicle. The format of each dictionary is as below:

```
vehicle_info = {
                "vehicle_id": vehicle_id,
                "bus_stop_name": bus_stop_name,
                "direction": direction,
                "expected_arrival": eta,
                "time_of_req": time_of_request,
                "arrived": False
            }
```

In the dictionary, the `vehicle_id`, `bus_stop_name` and `direction` are strings. The `expected_arrival` and `time_of_req` are Python `Datetime` objects and `arrived` is a boolean.

2) Update the estimated arrival time of a bus: For each of the items returned in the API call, check if the vehicle corresponds to one already in the `bus_information_route` table. If it already exists, then update its expected arrival time, otherwise, it is a new vehicle to track and add to the table.

3) Check if the bus has arrived or not: If the current time is after the predicted arrival time of the bus, then it is classified as 'due'. If the current time is 5 minutes after the predicted arrival time of the bus, then it is classified as 'arrived'.

4) Write to the relevant database Once a bus is classified as 'arrived', remove it from the `bus_information_route` table and add it to the `bus_arrivals_route` table. When adding to the `bus_arrivals_route` table, check to see if on this day this particular vehicle is already in the table. It is necessary to check this because each vehicle will complete the bus route more than once per day, and therefore, the current trip number has to be kept track of too.

Data stores

- `valid_stop_ids_route` stores the list of valid stop ids and its respective plain English name for that particular route.
- `bus_information_route` stores information on buses that haven't arrived yet for a particular route.
- `bus_arrivals_route` stores information on all buses that have arrived for this route.

Images/database-diagrams.png

Figure 2: Databases

1.2 Data Analytics

Clustering, graphs, exploring the data

Images/spread-of-buses-37.png

Figure 3: Number of buses during the day for route 37

1.3 Historical Models

Images/delay-by-time-of-day-52.png

Figure 4: Delay by time of day

Images/historical-pred-actual-52.png

Figure 5: Bus 52

1.4 Regression Models

1.5 Neural Network Models