Conveyance

Problem Statement:

When a delivery rider arrives at the customer-location that the customer entered for the order, the rider logs in our rider-app that they have arrived at the customer. However, because of any number of reasons (buildings blocking GPS signal, poor data entry by the customer, a rider's newness, etc) the customer's actual location might not match the customer's entered location. As a result, the rider then starts the journey of finding their way from the customer's entered location to the customer's actual location. We track this traversal distance as dropoff_distance. In an ideal world, all dropoff_distance's are zero.

Objective:

- Create a model that predicts the dropoff_distance of an order, given some features about the order. A model like this could then be fed into the logistics rider-scheduling algorithms.
- Propose some front-end features that might help reduce dropoff_distance

Acceptance Criteria

- Jupyter notebooks and any well-known NLP and ML libraries (pandas, numpy, scikit-learn, environment-management, etc)
- Jupyter notebook and code should have the following structure:
    - requirements installations (in separate requirements file)
    - data cleanup, validation, initial explorations, including visualisations, any feature engineering
    - models you tried, and the one you kept
    - explanation of front-end feature proposals
    - any other findings that you would like to share
    - ideas about improvements that can be done if you had more time

Approach Constraint

- The approach to the problem has been limited to showcasing possible features as well what can be done given the acceptance criteria
- The error or accuracy of predictions have a secondary importance hence the solution isn't reflective of the model being a deployable one.
- Few ML algorithms were taking too much time to train and hence they were either ignored, or trained on subset
- Project structure doesn't include unit tests as well as multiple yaml files to segregate various parameters

IDE & Tools

- Pycharm IDE
- Jupyter Notebook

Approach

1. **Data Ingestion**
    - Since the data is a json, we need to convert it into a data exploration format that can be used with ML libraries
    - We convert it to a csv file and read it as a pandas DataFrame to make use of optimised numpy operations underneath
    - In a real life scenario, we'll have a data ingestion module that should query from various data sources and be able to convert to various other formats apart from csv

2. **Data Exploration (Exact Details in Notebook)**
    - The next part was to understand what the data was with numbers and visuals which would make the ML feature selection easier as well as the process of creating new features
    - First thing was to check for duplicate rows and understand what each row meant i.e in this case, each row was a unique order. There were 2.5 million rows approximately
    - Customer_id was used repeatedly and was one of the fields that would help with aggregation
    - Identified missing values which were only present in delivery postal code.
    - Checked for outliers in terms of the target variable and removed them
    - Checked for correlation between variables and the instructions columns were highly correlated due to them being sparse. But checking the mean of distances between their distinct values revealed a difference of 20m between their means hence they were kept
    - Geohash had many values hence a certain amount of characters were considered to make it more generalised. A better would be to use geohash6 which had approx 622 values and cluster them according to similar dropoff distances
    - For demonstration purposes, a geohash level 5 was considered and 2 features were extracted. Though the number of orders were too small for those codes and ideally would have not been used.
    - Timestamp has many possible features that could be used like day of month, day of week, time of day (hour) or even clusters of time like - "morning", "noon", "evening" and so on. Only "hour" was extracted and there was a clear pattern in terms of order density for each hour.
    -

3. **Training a model**
    - The features that were dropped were the order_id, customer_id, timestamp, index, geohashprecision8 and delivery postal code.
    - Since the target variable is a continuous variable, the loss functions used are Mean Absolute Error(MAE) and Root Mean Squared Error (RMSE)
    - To train the model, following steps are followed:
        i. Creating a pipeline to preprocess and train so that multiple classifiers or encoders can be plugged in or out

    ii. Categorical variables are one hot encoded whilst numeric variables are normalised between 0 and 1
    iii. Fit pipeline on Training set
    iv. Predict on validation set. In this exercise, I've only used Train and Test but ideally we extract the Test set and do not use it to evaluate models. Validation set is used for that purpose with test acting as unseen data
- Two Base benchmarks were obtained during Training for each evaluation metric. First, MAE and RMSE by **predicting the mean of the entire training set** and second, **predicting the weighted mean of distances below 200m of the training set.** This helped set a reference point to see if it can be improved or not.
- The various models considered are:
    i. Linear models i.e Linear Regression and its variants - Ridge and Lasso
    ii. Tree models i.e Decision Trees Regression and Random Forests Regression
    iii. Support Vector Regressor (Regression using SVM)
- Model Selected:
    i. Support Vector Regressor (Regression using SVM)

4. **Next Steps (What I'd try if I had more time):**
    - Given the current set of parameters, hyperparameter tuning, cross-validation and kernel transformations would help reduce the error
    - Used PCA to reduce dimensions
    - Having more features - we miss out on rider history and customer history
    - Associating confidence in predictions for each customer id or rider id. First time customers will have a lower associated confidence w.r.t a smaller distance
    - Solving the problem in a different way i.e
        i. Instead of focusing on directly predicting distance, we can create clusters of ranges of distances and try tackling them independently
        ii. For eg, anything less than 50m is acceptable owing to gps sensitivity or nearest vehicle accessible point before the rider has to walk
        iii. Grouping of distances in 30m range and turning it into a classification problem with different treatments for different groups
        iv. Identify the best way to tackle or predict them in advance.
        v. Use an ensemble of models and use confidence values to decide the best estimate for the distance
    - Use geohash level 6 instead. This will introduce variance if certain groups of areas can be identified and features are extracted accordingly
    - Extract many features from timestamp and add them to models
    - Experimenting with evaluation metrics apart from absolute error- like squared errors and logs
    - Experimenting with kernel approximations
    - Adding confidence to predictions

Front End Feature Proposals

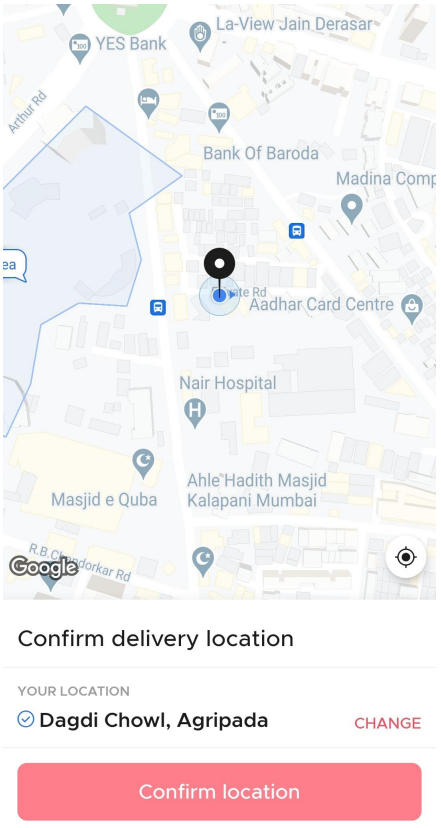
Problems:

The main problem, according to me, are the outliers and the distances exceeding 60m

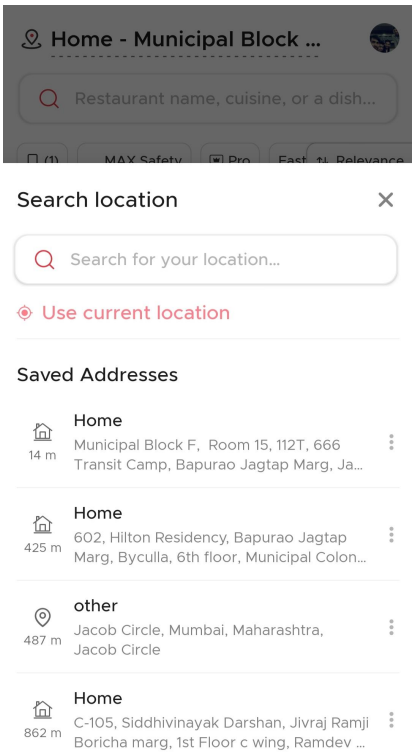
1. Anything greater than 800m or 1km is an outlier, and the possible reasons for this are:
    - Customer enters the address and the suggestions map some place which is a few kms away or even in a different country (which can explain why we have a 50000+ m value too). Customer selects it unaware that it doesn't match his gps location
    - Location permission is perhaps turned off
    - Customer is ordering for someone else
    - Customer is travelling
2. Random Reasons could influence someone wanting the rider to drop off because of certain set of rules within places for eg, a housing colony that doesn't allow outsiders to enter and the security guards pick it up at the gate for the customer
3. Issue with address suggestions which aren't reflective of actual location on map and unknowingly the marked position is 100+ metres away

Solutions:

1. In general, location data should be compared with the dropped pin and in case of a distance mismatch, there should be a pop-up or prompt message to highlight to them that the location is 200+ or n+ metres away from their actual location. This will assist in tackling issues of locations set far away by users unknowingly
   Reference Image: Zomato app



2. Saved addresses or preselected addresses should then display the distance of the dropped pin from location of user to highlight the actual distance as seen below



3. Highlight instruction fields (to encourage user to fill) in case dropped pin is more than a certain distance away from the actual location
4. More granular pins for areas with high order density such that dropoff distances are minimised
5. Suggestion ranking score when an address is typed, with confidence that is inversely proportional to the distance