Predicting New York Taxi Trip Durations

with a RandomForest regression model

March 21, 2025

Uppsala University
1DL034, Introduction to Machine Learning, VT2025
Group 7

Authors:

Tuva Björnberg	tuva.bjornberg.5452@student.uu.se
William Mårback	william.marback.7699@student.uu.se
Nora Reneland	nora.reneland.1770@student.uu.se
Matilda Stenbaek	matilda.stenbaek.1554@student.uu.se

Contents

1 Introduction

The objective of this project is to explore the effectiveness of different machine learning algorithms, preprocessing methods and hyperparameter choices to create a regression model for predicting the duration of a taxi trip in New York City. The data used is a sample from a publicly available dataset from the NYC Taxi and Limousine Commission [1]. The dataset consists of several different features describing the taxi trip, such as trip distance, passenger count and fare amount. All data used is collected from the period of December 2023 to November 2024 and is to be evaluated on an evaluation dataset with similar entries [2].

2 Feature extraction and preprocessing

Before training the model, the data needs to be preprocessed to ensure that "bad" data is removed or modified so as to not have a negative effect on the model by misrepresenting the average data. One part of this is to extract the feature that we want to predict against, in this case the distance, and to drop those that have little to no importance to the predictor. Another important aspect to have in mind is that the regressor can only handle numerical values, so all non-numerical values need to be converted beforehand.

There are also methods that can be applied to "good" data that might improve the regressor's predictability. Among those, bucketing, loading in data differently, and changing the test/train data split could improve the model's predictions.

2.1 NULL values

The first step of preprocessing the data is managing all the rows that have at least one cell with a null value, since the regressor cannot handle null values. In this case that was done by completely removing any rows with null values with the *dropna* function of Python's Pandas library. Since the available training dataset is very large, 32 million rows, dropping rows with missing data is the most logical approach. By dropping the rows completely, instead of trying to fill in the missing values, it is ensured that all the training data has valid values and are as representative of the dataset as possible. If there had not been as much training data available, using methods like imputation or interpolation to instead "fill in" the missing values, would have been good in order to keep as much of the training data as possible.

Interpolation was also attempted as a solution to the missing values, but resulted in a worse prediction score. This shows that the "inaccurate" predicted values do more harm to the predictions, than the benefit of including the extra rows that would otherwise have been dropped.

2.2 Categorical values

Thereafter, relevant categorical values were replaced by numerical values. Initially, the *store_and_fwd_flag* feature only had two string values (N/Y). These are replaced using the Pandas method *replace* with the numerical values [1, 2] to follow the standard of other numerical features in the dataset. Usually, using one-hot encoding is a better practice in order to avoid implying there is an order or numerical relationship between categorical values, e.g. that "Y" is "double" that of "N". But to stay consistent with the rest of the numerical data in the dataset, the categorical data was still chosen to be set to ordered numericals, i.e [1, 2], since that was already the case for several of the features in the dataset.

2.3 Dropping columns

The dataset was also cleared of features deemed not to have a significant impact on the training and testing data. The columns of features *ID, t_pep_pickup_datetime* and *t_pep_dropoff_datetime* were dropped using the *drop* method. The feature *ID* is a unique identifier (for each row) which means it shows no correlation to any other feature or to the target. *t_pep_pickup_datetime* and *t_pep_dropoff_datetime* were removed to scale down the preprocessing complexity. This is because these features consist of a datetime type and would require splitting the data into separate columns, for example, month and minute, to see if any of them have a correlation with the other data. This approach was ultimately not tested, but could be implemented instead of dropping the columns completely.

2.4 Bucketing: Taxi zones

The taxi zone column contains numerical values representing different taxi zones within New York. In the *taxi_zone_lookup.csv* file, these are all mapped to their corresponding borough. This means that the values in the taxi zone column can be replaced by their corresponding borough to get fewer unique values in this column and create a clearer connection between datapoints. The boroughs were then also replaced by numerical values representing the boroughs as described in 2.2. Without implementing bucketing, the RMSLE was 0.26923 and after bucketing, the score was 0.30625, meaning the score got worse with bucketing. This could be the case for a few different reasons. First of all, inspecting the data after bucketing tells you that the vast majority of the taxi trips were carried out in Manhattan. This means that the dataset becomes unbalanced, and predictions might be worse for less frequently appearing boroughs. Furthermore, implementing bucketing ultimately means a loss of more exact information. In this case we can see that the more precise information about the taxi zones was helpful in making the predictions, and bucketing therefore led to a worse score. For this reason, bucketing was not implemented in the final model.

How the bucketing was done could also have an effect on the outcome. For example, choosing to use one-hot encoding for the different boroughs could have led to a different result. The numerical values that replaced the boroughs have no connection to the actual borough, or their relations to each other, since it was done arbitrarily. An alternative way of doing it is to have the numerical values of the boroughs in some way reflect their distances to each other, or the size of the borough, since both of these pieces of information could be useful when making the prediction.

2.5 Labels and splitting the dataset

After pruning null values and managing categorical data, *duration* was extracted as the label, and the rest of the dataset was set as the features.

The labels and features were initially split into test/training data by 20/80 to have enough data to train the model as well as for testing.

Increasing the amount of testing data benefited the predictions the model made. While a test/train split of 1/99 worsened the prediction against the testing data of the split, it did improve the prediction against the evaluation dataset. A reason for this is that more data can be used for training, with a tighter split, and then using the evaluation dataset as the test data for the model. This was especially useful when not loading the complete dataset from *training_dataset.csv*, because of RAM- or time-issues, to ensure there was as much data as possible to train the model on.

Increasing the training data in the split was more useful when a model was established and a sense of what would improve or worsen the predictions. Training the model on more data was one of the main factors that led to better predictions.

2.6 Feature importance - for outliers

To determine which features were of the highest importance and had the strongest correlation to the target value, the feature importance of all remaining features was checked with the help of a feature importance bar plot, as seen in Figure 1. Features with bad prediction scores, such as *store_and_fwd_flag, improvement_surcharge, mta_tax* and *airport_fee*, could be removed to hopefully let the more important features weigh heavier in the dataset. The features mentioned above were at this stage not dropped immediately, but instead used as a reference for further preprocessing.

2.7 Outliers

The dataset contained values that would be considered outliers, such as traveling long distances in impossibly short times, and therefore the dataset had to be pruned to remove those samples. An example can be seen in Figure 2 where an outlier might be traveling

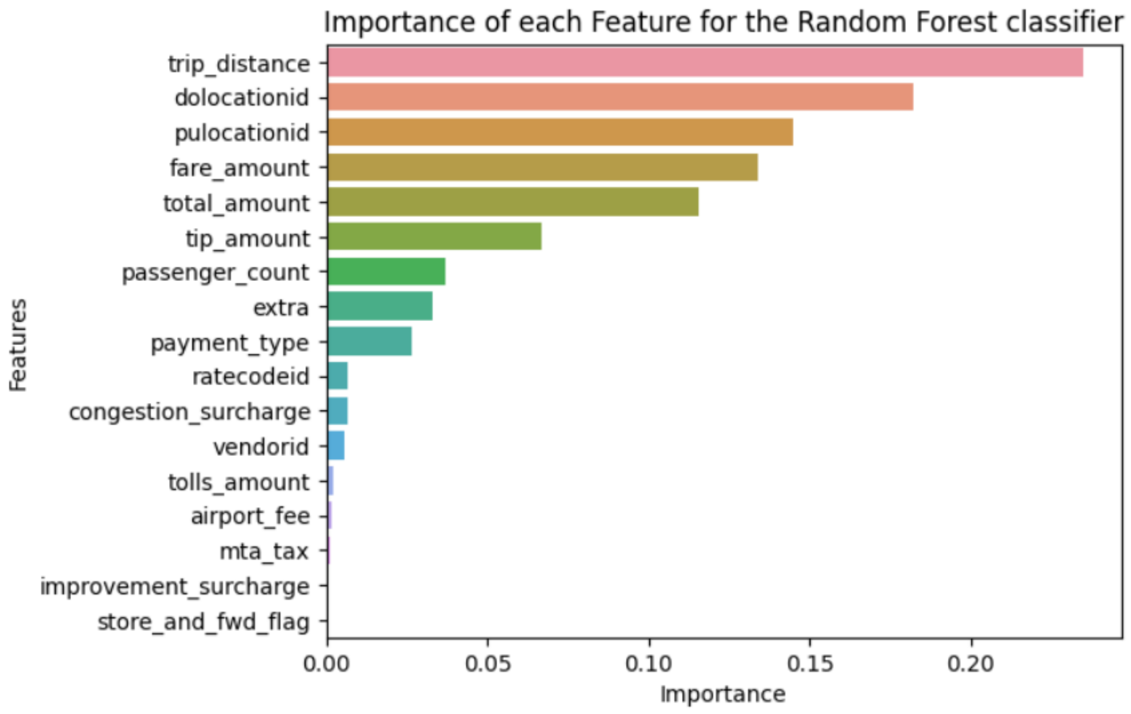

Figure 1: Feature importance plot of model with 500 000 rows loaded

0 miles in 23 hours or in Figure 3 where an outlier seems to be reaching 65 000 miles during 0 seconds. Both of these examples would skew the predictions for this model by misrepresenting the average data.

The scatter plots were made between *duration* and *trip_distance*, the label and the feature with the highest correlation. To prune the data of outliers, 50 miles/hour was decided to be the top speed for a taxi cab, which was deemed a reasonable speed for a vehicle in New York City. Therefore, any data points that had a higher average speed were removed, which is shown as the leftmost red line in Figure 4. Additionally, all data points that had a duration of more than 3000 seconds (50 minutes) and an average speed of under 50 miles/hour were also removed since it was deemed unlikely that a person wouldn't simply leave the cab if this happened. This is represented by the rightmost red line in Figure 4. The result of the cut-offs can be seen in Figure 5 and lets the testing and training data be more accurate to that of the most common data. Keeping the outliers would mean a few data points would bias the entire prediction because of their extreme values, skewing any predictions made.

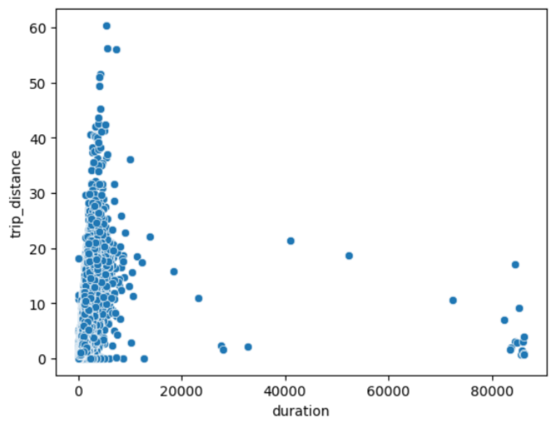

Figure 2: Scatter plot of *duration* and *trip_distance* for 50 000 loaded rows.

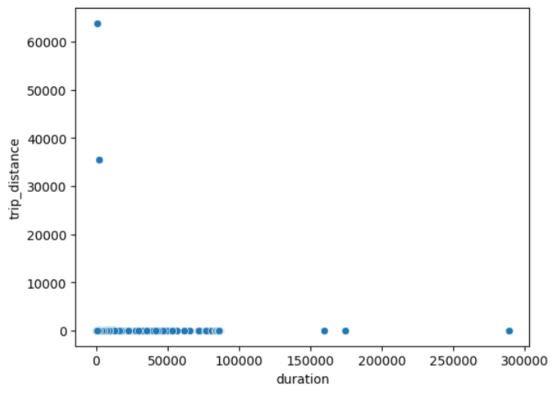

Figure 3: Scatter plot of *duration* and *trip_distance* for 500 000 loaded rows.

2.8 Feature importance - for pruning

After removing outliers, it is relevant to re-evaluate the feature importance since extreme values of the outliers might have been misrepresented for the predictions.

The dataset was re-split, trained again and a prediction was made on the updated DataFrame in order to see how the outliers affected the feature importance.

As seen in Figure 6, most of the previous feature importance in Figure 1 looks like they aren't important, or rather *fare_amount* becomes much more important. The *fare_amount* feature shows higher importance than was initially presented, which was initially obscured by the outliers.

At this stage, it is more relevant to consider dropping or removing features which has a low importance to let more important features weigh the predictions more.

Even though the feature importance should help illustrate which features could be removed in order to let the more important ones weigh heavier, this did not seem to be the case for this project. Removing features for this dataset worsened the predictions. Even removing features with very low importance, such as *improvement_surcharge*, only marginally impacted the R^2-score and root mean squared logarithmic error (RMSLE), usually for the worse.

A marginal improvement of the RMSLE score, from 0.18228 to 0.18160, was made when removing the features *store_and_fwd_flag* as well as *airport_fee*, which were two of the least important features. These two columns were dropped from the dataset.

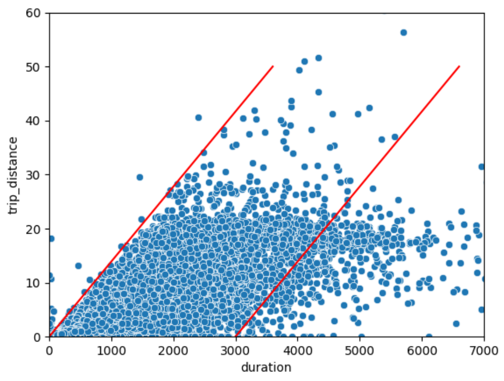

Figure 4: Closeup of scatter plot of *duration* and *trip_distance* for 50 000 loaded rows, with red lines describing the cutoff.

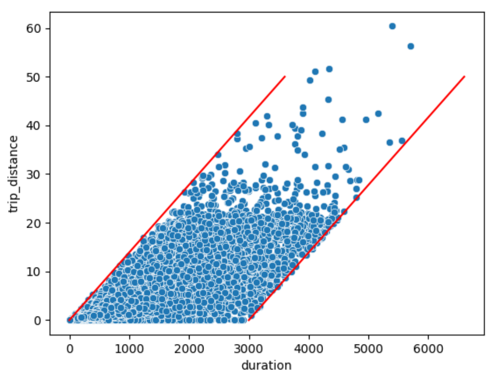

Figure 5: Scatter plot of *duration* and *trip_distance* where outliers are cut off.

2.9 Loading values

Situations may occur where the data in the dataset, on which the model is trained on, is sorted in any way, for example, in descending order of *trip_distance* or having more outliers in the first rows. If this occurs and only part of the dataset is loaded for training, then the model may misrepresent the complete dataset and train on "bad data". For this project, the data used for training, *training_data.csv*, is too large to completely load to systematically test different preprocessing methods and tuning parameters efficiently because it takes too long to train and predict the model. Instead, only part of the dataset is loaded, which was initially the first rows of the CSV file, for example, the first 50,000 rows or first 1,000,000 rows.

Trials of letting *read_csv* read random rows from the dataset were conducted to see if the ordering of the dataset was actually affecting the predictions. However, randomly loading for example 3 million rows from *training_data.csv*, compared to loading the first 3 million rows, did not have a large difference when predicting the model against the testing data, but significantly worsened when using the model on the evaluation dataset. A reason for this could be that the first 3 million rows better represented the data in *evaluation_dataset.csv*.

2.10 Normalization

Normalization of the training data was tested but did not improve the RMSLE. Instead, it worsened the RMSLE a little bit. Usually, normalizing will improve the result since it then handles outliers in the data much better as all data are roughly the same size of $0 < x < 1$,

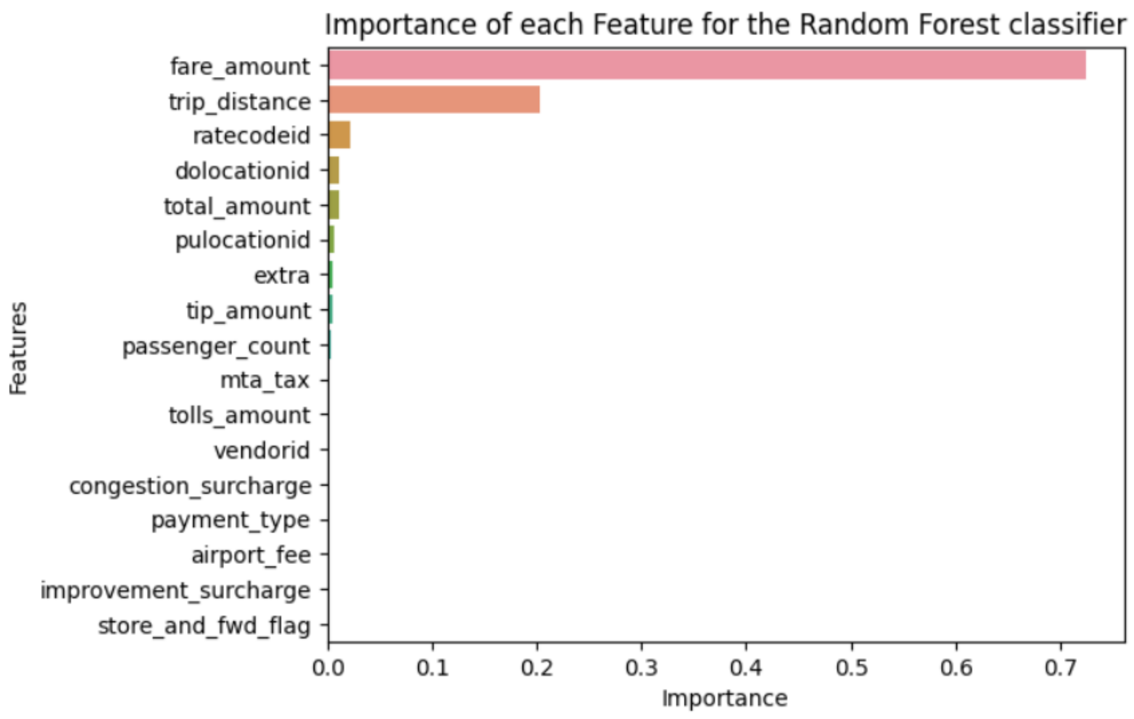

Figure 6: Feature importance plot of model with 500 000 rows loaded, after outliers were cutoff.

where x is the size of the data. Why the result became worse in this case is unknown. However, if the pruning of the data was not done, the effect of normalizing the data played a larger role since outliers were present. It is therefore possible that the pruning of the data which was done earlier made it so that the normalization was not needed, since outliers had already been removed.

2.11 Duplicates

Having duplicates in the dataset can skew the results by making the regressor train twice on certain values and thus reinforcing those occurrences' values more than necessary. This also results in less variation in the dataset and the regressor will have a harder time learning differences. However, this is easily solved by dropping all duplicates of the training data before training the model.

When it was time to check for duplicates, it was found that no duplicates were present, and thus, this step was not needed.

3 Imbalanced data consideration

When one class of data is considerably more frequent than other data in the same dataset, the data is imbalanced. This may cause the algorithm to favor the majority and have difficulty recognizing minorities when encountered.

The New York Taxi dataset has some imbalances in the data. One of these imbalances is concerning the feature *vendorid*. When loading 500k rows of data from the training dataset, it is seen that there is roughly a 3.5:1 split between $vendorid = 1$ and $vendorid = 3$. This imbalance is however not an issue as this split is roughly the same in the evaluation dataset, causing the predictor to operate on the same terms as what the model was trained on.

4 Algorithm exploration

While deciding beforehand on which algorithm to use for regression can save time, it can result in a better algorithm being left out. To ensure that the most suited algorithm was selected for the task, multiple regression algorithms were tested and analyzed. They were all trained and tested on the same amount of data and on the same random seed for comparison. The performance analysis of each model was done using the R^2 score and RMSLE.

An important aspect to reflect upon is that while a model might perform well in testing, its performance may not be fully representative as it might have been given either a favorable or unfavorable seed specific to that model. Thus, it can be fully possible for a model that performed worse than another model, to perform better at another seed.

Table 1: Testing various algorithms

Regression algorithm	**R^2-score**	**RMSLE**
Linear regression	0.69388	0.58211
Random forest	0.92444	0.22762
K-Nearest neighbor	0.84771	0.39751
Decision tree	0.86948	0.24633

Table 1 displays the R^2-score and RMSLE for four different regression algorithms, trained and tested with 50 000 rows from the dataset. A linear and a general Support Vector Regressor were tested as well, but they never converged and thus did not produce any result. The algorithm with the best results was Random forest and it was therefore chosen as the algorithm to move forward with.

5 Hyperparameter tuning

For Random forest regression, there are a few different hyperparameters that are possible to tune. One of them is the number of estimators, i.e. the numbers of trees. The model will typically improve with a larger number of trees, but this comes at a computational and hardware resource cost, and therefore it can be desirable to have a lower value. Other hyperparameters are the maximum number of features to be considered when splitting a node and the maximum depth of the trees. Both of these can be tuned to control under- and overfitting, and it is therefore important to pick good values.

Grid search was used to find the best possible value for the number of estimators, as well as the max depth and number of features. As expected, the best value for the number of estimators was quite high, 600 when the default is 100. However, this significantly increased the training time which made it impossible to use as much training data as with 100 estimators. After testing different combinations of the number of estimators and the amount of training data, it became clear that using more data had a more beneficial impact on the result than using more estimators, and was therefore deemed to be a better use of time and resources.

The cross validation also revealed that the best max depth was "None". Not having an upper bound on the depth of the trees can in some cases lead to overfitting the data, but in this case that didn't appear to be a problem. The same observation was made for the number of features. From the grid search it became apparent that the best option was to not limit the number of features used for splitting. Both of these hyperparameters are considered the default, and no changes had to be made.

6 Evaluation metrics

R^2 score and Root Mean Squared Logarithmic Error was chosen as performance metrics for the models as mentioned in section 4. Both were selected as a means of comparison. R^2 score and RMSLE score differently. The former will score based on correlation, with 1 indicating full correlation and 0 no correlation between predicted and true values. In contrast, the RMSLE, as the name suggests, computes the error between the predicted and true values logarithmically. The closer the error is to 0, the more accurate it is as a result.

Computing logarithmically is to be preferred since multiplications can be avoided and addition is used instead Multiplying many small numbers can cause rounding errors upon each multiplication, resulting in a large rounding error in the end. In contrast, doing multiple additions of the same values makes for a much smaller (if any) rounding error.

In addition, the Mean Absolute Error (MAE) was also analyzed to find by how many seconds the predictions were off by on average. This metric was not used to, for example compare any of the models or calculate hyperparameters, but it can give intuitive insight into how well the final model performs in a way that is easy to understand. The final model had a MAE score of 79, which means that the predictions were wrong by 79 seconds on average. Considering the range of the duration values in the dataset, this is quite low and means the model performs well.

7 Result

Training the model on $n = 500$k pruned rows of the dataset takes around 5 minutes running on 4 threads parallelized on Kaggle, using a maximum of about 5GB RAM. This is done using the joblib threading library for sklearn. Upon evaluation, it received an RMSLE score of ≈ 0.18541, when running on the testing data from the split. When evaluating against the evaluation dataset a RMSLE score of ≈ 0.27421 was gained, when submitting on the Kaggle leaderboard. This score is higher than the internal one and was expected. The internal data was all pruned to ensure it had no bad data, but the evaluation data for the leaderboard might not be. In this case, there might be outliers that are not handled very well by our model since it was not trained on such data.

The model was also run briefly with more hardware than Kaggle allows, allowing it to train and run with all 32 million rows of data while also keeping the default number of estimators $n = 100$. Parallelized on a 8 thread CPU, training the data took ≈ 1.5 hours and predicting took ≈ 20 min. The system had access to 32GB RAM, but needed to use an additional 200GB of swapped data on the disk to store and utilize the tree. This resulted in the best RMSLE score from the evaluation dataset of 0.26077, which is the score that can be seen on the Kaggle leaderboard.

The resulting trained model has a fairly high accuracy with a RMSLE score of less than 0.2, R^2-score larger than 0.93 and an average mean error of about 79 seconds, all when evaluating on the data from the test/training split.

8 Conclusion

A predictor for predicting the duration of a taxi ride was made using a Random Forest regression model. The selection of the type of model was done by testing some of the most "well known" models, those covered in this course, and selecting the one that performed best.

Before testing, the data was pruned to where most, if not all of the "bad data" had been removed. This ensured that the algorithm can converge towards a realistic value without all the outliers worsening the predictions. This includes handling both missing and non-numerical data, as well as removing datapoints with unrealistic values.

The trained model completed predictions of the duration of a taxi trip with a RMSLE score of 0.26077 and with ≈ 79 seconds error from the actual trip.

The greatest impact of the predictions were the amount of data that was used when training the model. Despite using preprocessing methods, like normalization, bucketing or tuning hyperparameters, increasing the amount of data had the greatest impact.

References

[1] *Tlc trip record data*, Accessed: 2025-03-03. [Online]. Available: `https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page`.

[2] *1dl034 project vt25*, Accessed: 2025-03-03. [Online]. Available: `https://www.kaggle.com/competitions/1dl034-project-vt25/overview`.