

# Practical AI CA2 Report

Jeyakumar Sriram  
Shawn Lim Jun Jie  
Yeoh Wei Zheng Benjamin  
Aarron Loke Ruixuan

## Table of Contents

1. Introduction
2. Data Collection
3. Data Cleaning & Preprocessing
4. Exploratory Data Analysis (EDA)
5. Modelling
6. Model Deployment
7. Conclusion

### 1. Business Understanding

The advent of the digital era has significantly transformed the landscape of various industries, including transportation. With the rise in data-driven decision-making, companies in the transportation sector are leveraging the power of data science to enhance their operational efficiency and safety. GoCab, a leading cab service provider, has a vast database of cab trips, which can be used to develop a predictive model that can identify and classify trips as either dangerous or non-dangerous. In this context, our assignment focuses on the application of data science techniques to create this classification model for GoCab's managerial team. This model is deployed as a user-friendly Tkinter application designed to assist managers in predicting the safety of a journey.

### 2. Data Collection (Done in CA1)

Most of the data collection work was done during the first part of the assignment. We will be reusing the database setup and some of the ETL pipeline code while modifying it with recent improvements.

To reiterate our data collection, we are using a SQL Server Database to store the data and using python to perform ETL. All our database scripts are written using SQL utilizing the Bulk Insert functionality. The processed data returned by python is stored in the HDF5 (Hierarchical Data Format version 5) format. This is to ensure speed and space efficiency.

### 3. Data Cleaning & Preprocessing

#### A. Feature Engineering

To create better representations of the sensor data, I will be creating new columns from existing ones. Many existing columns such as Acceleration and Gyroscope are split into x, y, and z dimensions. I will be combining them using physics formulas to summarize them. In addition, I will also be creating 3 new columns, Pitch, Roll and Yaw from the acceleration values.

These are the formulas used.

Net Acceleration: 
$$\sqrt{acceleration_x^2 + acceleration_y^2 + acceleration_z^2}$$

Net Gyroscope 
$$\sqrt{gyro_x^2 + gyro_y^2 + gyro_z^2}$$

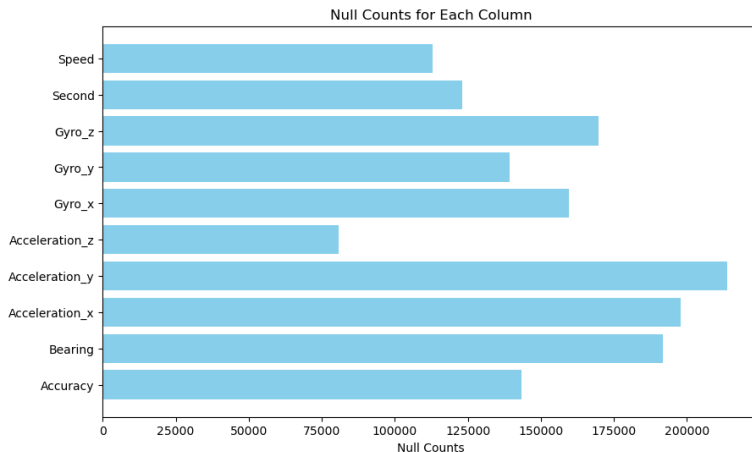
Roll 
$$\frac{180}{\pi} \cdot \arctan\left(\frac{acceleration_y}{\sqrt{acceleration_x^2 + acceleration_z^2}}\right)$$

Pitch 
$$\frac{180}{\pi} \cdot \arctan\left(\frac{acceleration_x}{\sqrt{acceleration_y^2 + acceleration_z^2}}\right)$$

Yaw 
$$\frac{180}{\pi} \cdot \arctan\left(\frac{acceleration_z}{\sqrt{acceleration_x^2 + acceleration_y^2}}\right)$$

For the drivers dataset, I have done data type conversion for the "driver\_name" column, label encoding for "gender" and "car\_brand," and calculated the age of driver and car. Memory optimization was performed with specific data types assigned to columns.

#### B. Missing Values Imputation

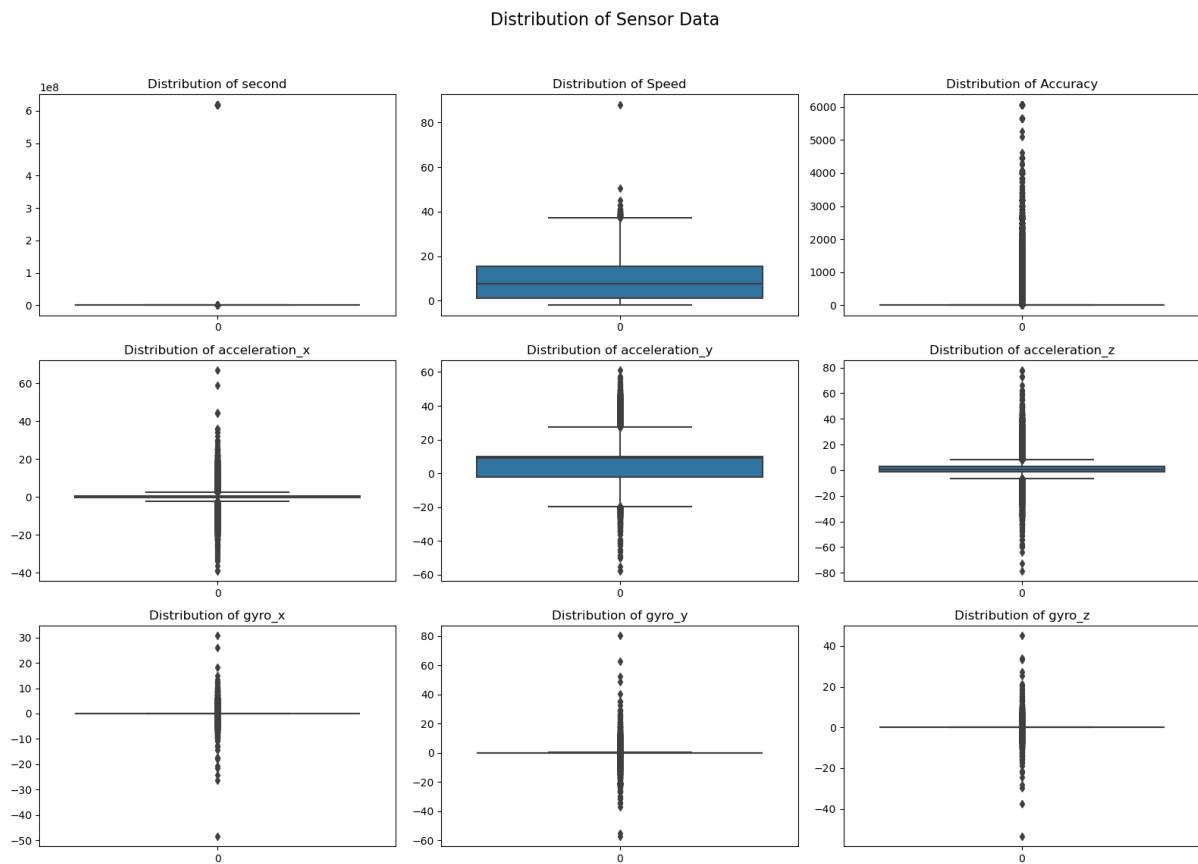


The following columns have null values ranging from 75,000 to over 200,000. For the seconds column, I have no choice but to remove the null values as the seconds define the order of rows within each trip. It is not something that can be imputed. For the remaining columns I will be using a

variety of imputation techniques based on the column. For the Accuracy, Acceleration and Gyro columns, I will be using polynomial imputation with order 5. For the Bearings columns I will be decomposing it into its sine and cosine components and interpolating them separately using ffill and bfill respectively. For the remaining columns, I did linear imputation. For Acceleration and Gyro, we had tried using K-Nearest Neighbors but found it to be unsuitable for the task the change in these values is better model through polynomials.

### C. Outlier Handling (Pre-Aggregation)

To analyze the presence of outliers in the dataset, we plotted the distributions of the numeric columns and visually identified outliers. We also performed manually inspection of certain bookingIDs to find more outliers.



As is evident from the graph above, many of the graphs have very extreme and rare outliers which could be removed without causing significant changes to the dataset. Speed and Seconds has obvious outliers in the top of their graphs. The Gyro values have some outliers mainly above 40 and below -40 which is the range that we will be classifying as outlier. Acceleration does have some extreme values, but we feel that they are still possibly not outliers and might be useful in indicating if a trip is dangerous. For most of the outliers we use imputation rather than removing them. This is to preserve as much of the dataset as possible. Additionally, through manual inspection, we found Booking IDs 163208757335 and 858993459333 to contain high amounts of outliers and thus decided to remove them.

## D. Data Aggregation

To train a classification model on this dataset, it is essential to aggregate it to convert time series data into tabular data. While this conversion takes place, our main aim to ensure the attributes of each trip are represented to the maximum in the features that we aggregate. This is to mitigate the loss of any crucial information.

To capture the presence of extreme acceleration or gyro values, when aggregating we counted such values and included it in our rows. We also added a variety of functions such as maximum and mean to capture the essence of the data. After aggregating, we performed a inner join between the three tables and removed some columns like “name”, “booking\_id” and others which are not very useful for prediction.

After aggregating, we have 19,998 rows and 16 columns of aggregated data.

## E. Outlier Detection

After aggregating the data for our ML model, we will have to perform outlier detection to find extreme data that might degrade the quality of our ML model. We will be looking at 3 main algorithms:

- One-Class SVM: SVM-based method learning a decision function find outliers.
- Isolation Forest: Random forest-based technique isolating anomalies by randomly partitioning features.
- Cluster-based Local Outlier Factor (CBLOF): Identifies outliers based on their deviation from cluster centroids in high-dimensional data.

One important issue is fitting the outlier model on the entire data. Throughout our model fitting process, we found that whenever we fit the outlier model on the entire data, it often detects dataset from the minority class more than the majority class. This results in loss of crucial information in the minority class. To mitigate this, we will be fitting the outlier models on Dangerous and Non-Dangerous data separately and performing outlier detection separately.

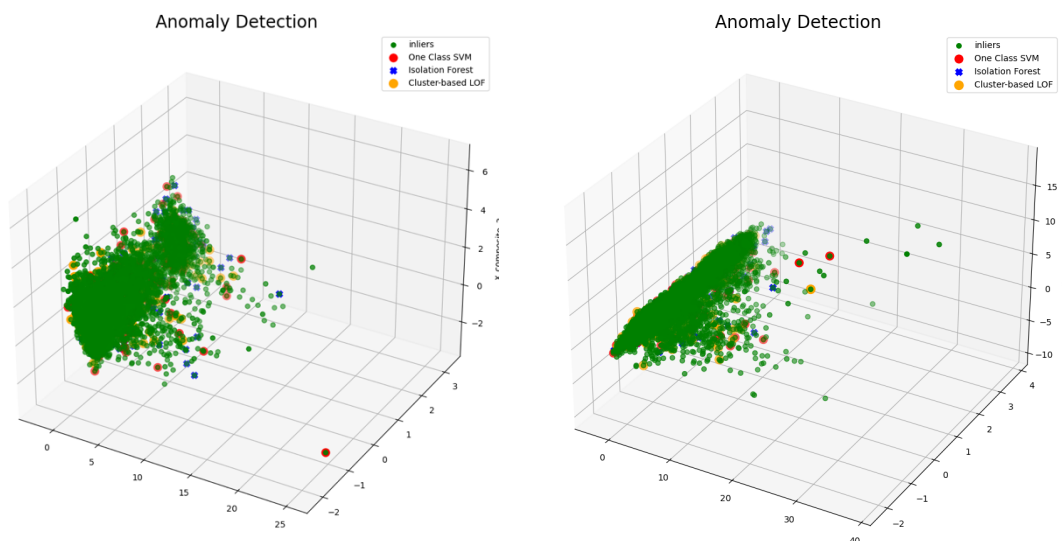


Figure 1 PCA Visualisation of Non-Dangerous Trips Outliers

Figure 2 PCA Visualisation of Dangerous Trips Outliers

To quantitatively compare these models, I will be fitting them on some data and remove the outliers. Use that dataset to fit a RandomForest model. I will be comparing the accuracy of the 3 different outlier detection methods using these scores. This will help me find the optimal combination of the outlier detection methods for both labels. Currently we are using a contamination level of 0.05 which means 5% of the data is considered outliers.

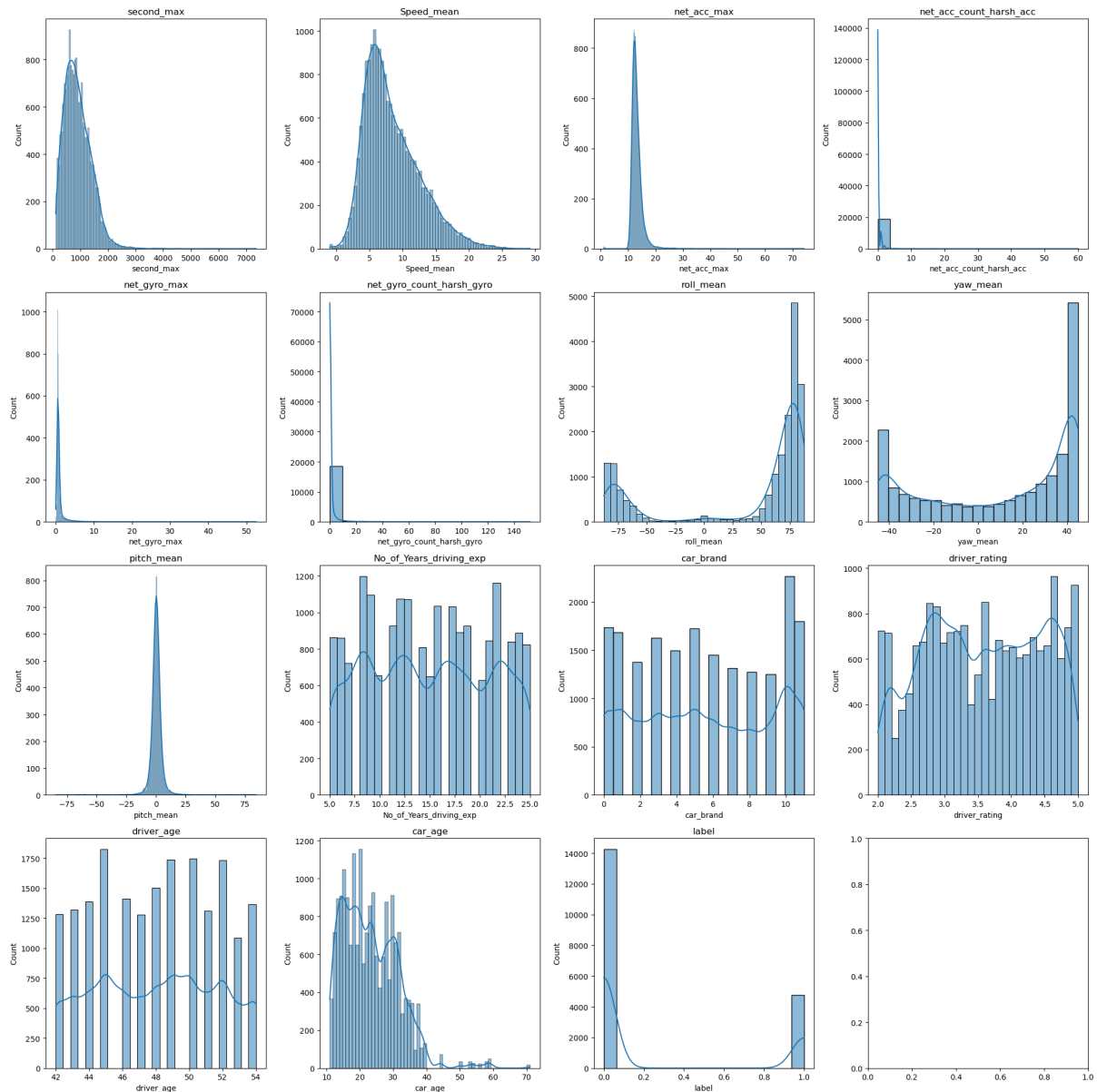
Method Dangerous	Method not dangerous	Outliers Removed	Accuracy	Recall	F2 Score
None	None	0	0.7845	0.2032	0.2373
None	Isolation Forest	0	0.7845	0.2032	0.2373
None	Local Outlier Factor	0	0.7845	0.2032	0.2373
None	One-Class SVM	0	0.7845	0.2032	0.2373
Isolation Forest	None	169	0.7809	0.1786	0.2102
Isolation Forest	Isolation Forest	671	0.7721	0.2549	0.2873
Isolation Forest	Local Outlier Factor	671	0.7724	0.2284	0.2607
Isolation Forest	One-Class SVM	671	0.7833	0.2180	0.2525
Local Outlier Factor	None	169	0.7835	0.1934	0.2266
Local Outlier Factor	Isolation Forest	671	0.7748	0.2297	0.2625
Local Outlier Factor	Local Outlier Factor	671	0.7758	0.2180	0.2507
Local Outlier Factor	One-Class SVM	671	0.7842	0.2094	0.2437
One-Class SVM	None	168	0.7803	0.1817	0.2134
One-Class SVM	Isolation Forest	670	0.7736	0.2445	0.2772
One-Class SVM	Local Outlier Factor	670	0.7742	0.2309	0.2636
One-Class SVM	One-Class SVM	670	0.7838	0.2198	0.2545

Using Isolation Forest for both labels produced the best result. We also tried using contamination level of 0.01 but it produced worse results. After fitting the models on the entire dataset, we removed a total of 1001 rows which were considered outliers.

## 4. Exploratory Data Analysis

### A. Distribution of Data

Before moving on to modelling, we would like to perform basic EDA to decide what kind of transformations would be suitable for this data. I will start with the distribution of numeric values.



There is a strong right skew in Second\_max, Speed\_mean, Car\_age, net\_gyro\_max and net\_acc\_count\_harsh columns. To address this problem, I will be using with box-cox or log transformation. These are the formulas for doing so.

$$y^{(\lambda)} = \begin{cases} \frac{y^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log(y) & \text{if } \lambda = 0 \end{cases} \quad y_{\log} = \log(y)$$

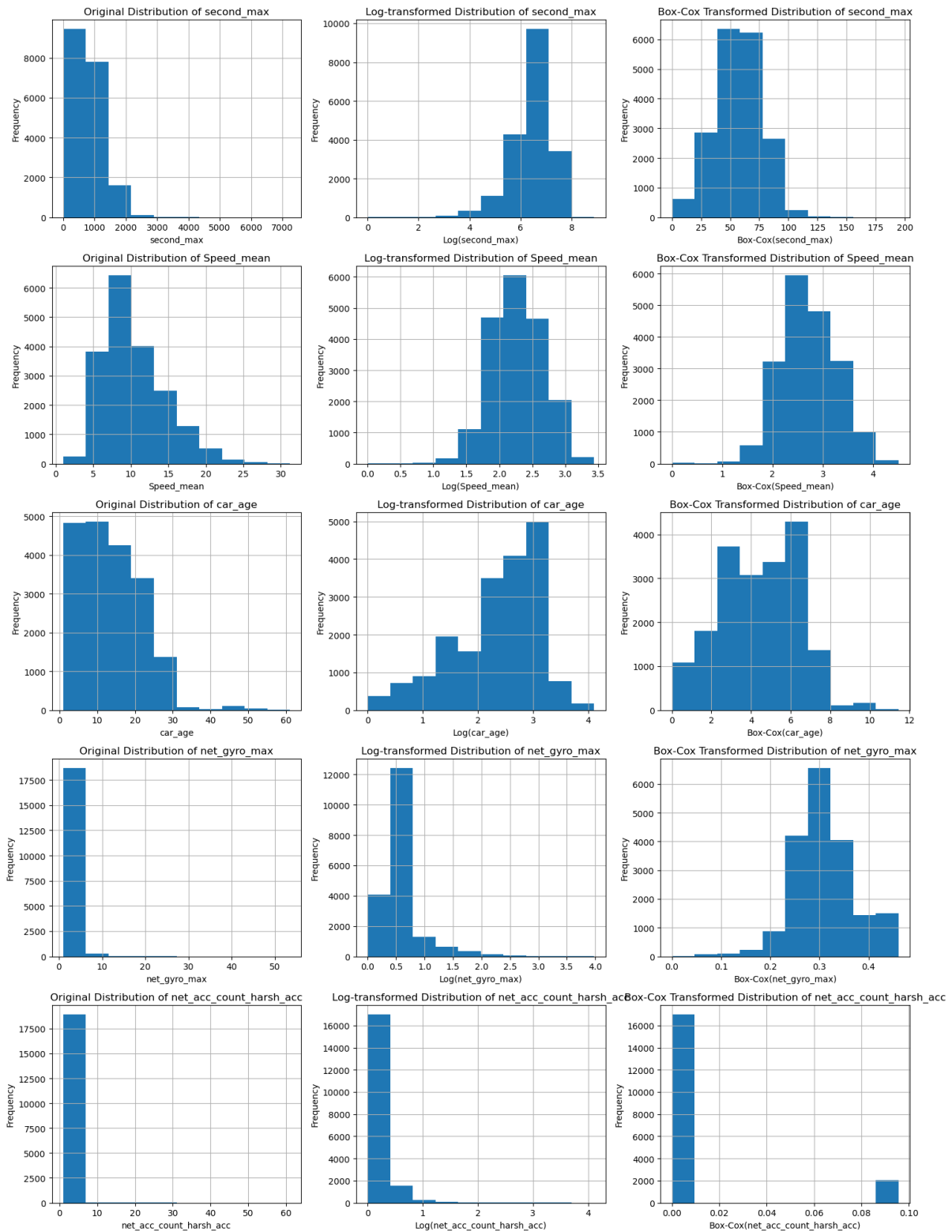
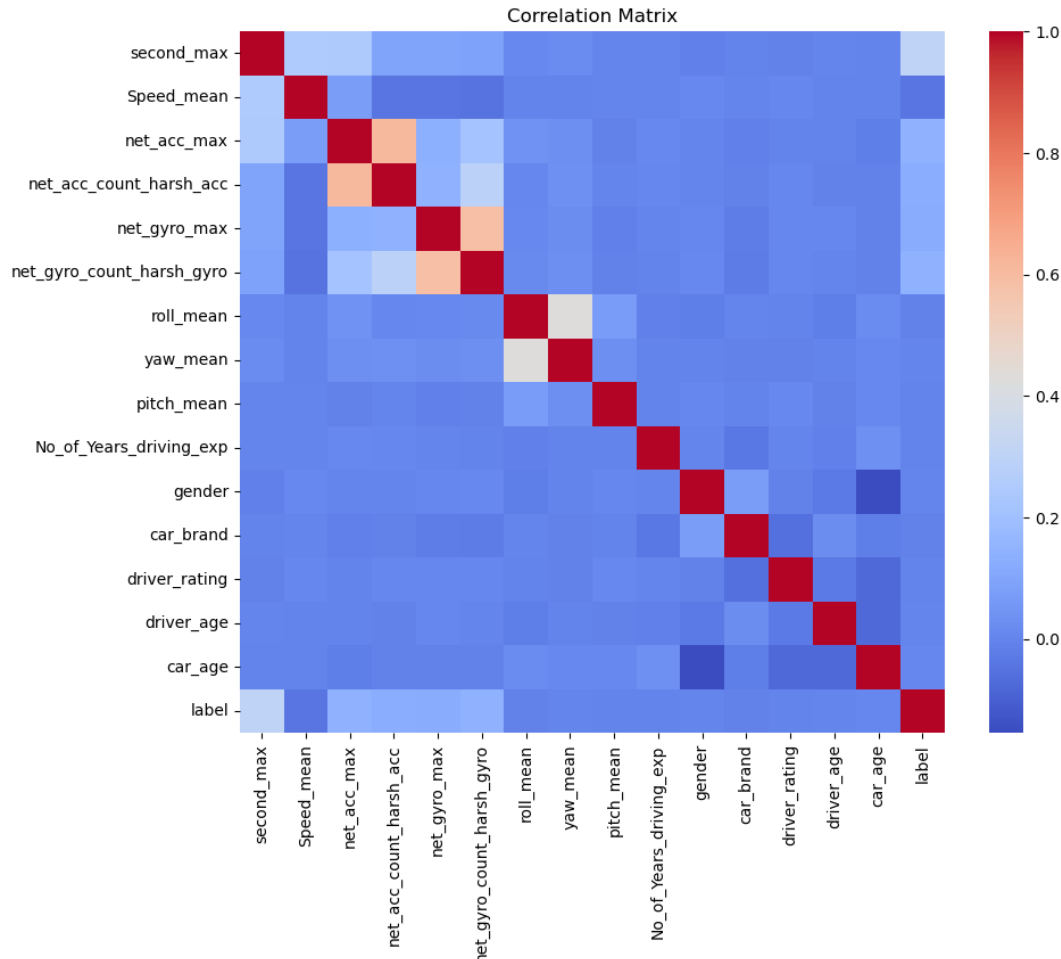


Figure 3 Box-Cox vs Log Transformations

As can be observed from the plots, Box-Cox transformations seem to be more successful in creating a normal distribution than simply logging. Therefore, we will be using this in our preprocessing before applying Standard Scaler.

## B. Tackling Multicollinearity

Multicollinearity occurs when multiple features of a dataset are highly correlated. This can lead to several problems including overfitting and inflated importance of certain correlated features. We will be finding and removing highly correlated columns.

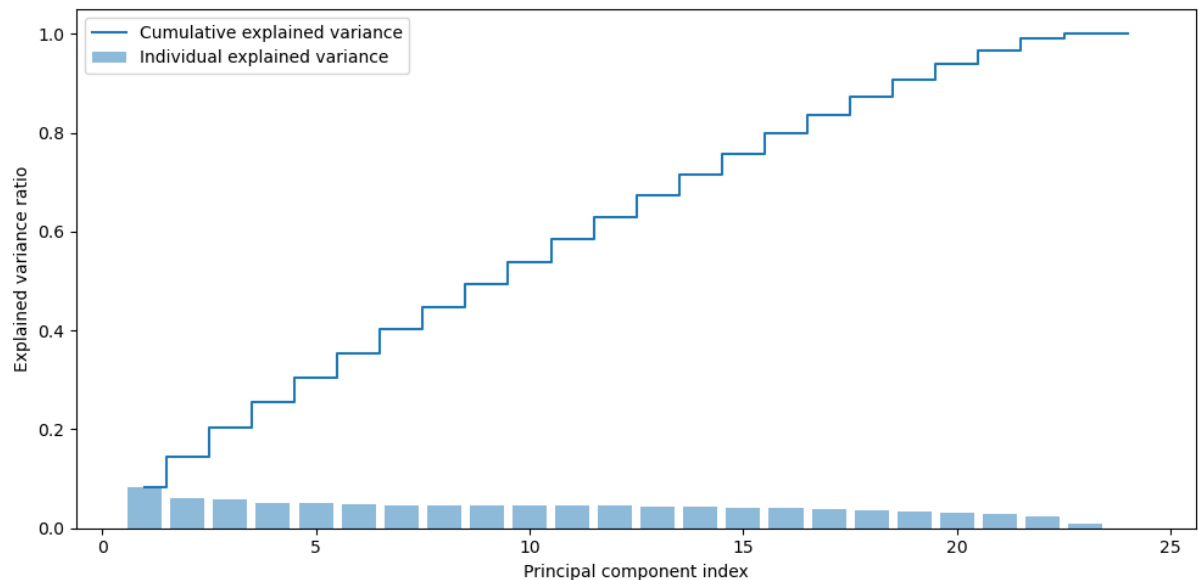


Since the features `net_acc_count_harsh_acc` and `net_gyro_count_harsh_gyro` are highly correlated. These are also columns we created during feature engineering. Since these add unnecessary noise to the dataset, we will remove them. For Yaw and Roll we will be combining them using the Pythagoras theorem to capture the information in a more concise way

## C. Experimenting with PCA



To address the problems with multicollinearity as observed in the previous experiment. PCA is a dimension reduction technique that combines multiple highly correlated columns into Principal Components.



But as can be observed from this, there isn't any elbow in the plot. There is high retained variance even with more than 10 principal components. This suggests minimal reduction in dimensionality. Thus, PCA is not very effective here and we will not be using it.

## 5. Modelling

### A. Pipeline

Based on the EDA above, we have decided that the following transformations will be needed.

One-Hot Encoding for Car Brand:

- Car brand, being categorical, undergoes one-hot encoding.

Ordinal Encoding for Gender:

- Gender, with two values ordinal is same as one hot with first column removed.

Combining Yaw and roll with Pythagorean Theorem:

- Yaw and roll, numerical features, are combined using the Pythagorean theorem.
- This creates a new feature capturing their relationship for improved predictability.

Box-Cox Transformation:

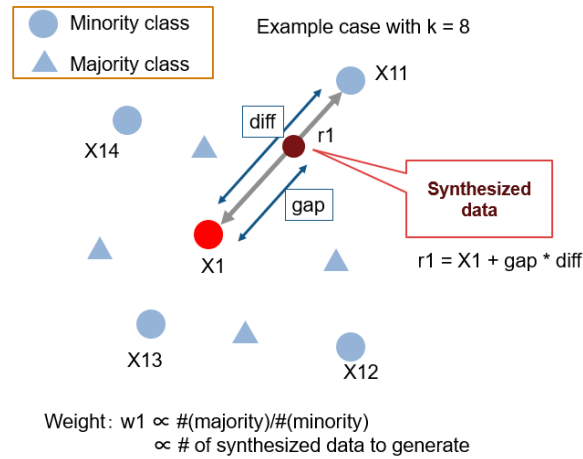
- Box-Cox transformation stabilizes variance and normalizes data.
- It addresses skewness and heteroscedasticity, enhancing model performance.

Standard Scaler:

- Standardization ensures all features have mean 0 and standard deviation 1.
- This preprocessing step facilitates feature comparison and model convergence.

### B. Addressing Imbalanced Classes

To address the imbalance between the Dangerous and Non-Dangerous classes, we will be using SMOTE (Synthetic Minority Over-sampling Technique). SMOTE is a technique used to address class imbalance by oversampling the minority class. This helps improve the model's performance, especially when dealing with imbalanced datasets where one class is significantly underrepresented compared to others.



We will be using SMOTE to increase the number of Dangerous Trips in the dataset. After addressing the imbalance, we will use an 80-10-10 Train-Test-Val split.

### C. Metrics

The metrics that we are looking out for more importantly are recall and F2-scores. Recall score is true positive over all the actual positive cases, which in our case looks out for whether it successfully predicted a dangerous trip correctly, while F2 score weights on recall higher with the following formula:

$$F_{\beta\text{-Score}} = \frac{(1 + \beta^2) \cdot (precision \cdot recall)}{\beta^2 \cdot precision + recall}$$

The reason for prioritising recall score is that we want a model capable of predicting a true dangerous trip correctly as our goal is to help manager makes decision on how to reduce dangerous driving. If we have a model that is poor at doing so, we will missed out instances of dangerous driving data and fail to mitigate the issue.

### D. Models Comparison

- **AdaBoost**: Boosting ensemble method that combines multiple weak classifiers to create a strong classifier.
- **DecisionTreeClassifier**: Non-parametric supervised learning method that makes decisions based on a series of if-else questions.
- **ExtraTreesClassifier**: Ensemble learning method that builds multiple decision trees and averages their predictions.
- **GradientBoostingClassifier**: Boosting ensemble method that builds decision trees sequentially, each correcting the errors of its predecessor.
- **HistGradientBoostingClassifier**: A variant of Gradient Boosting that uses histograms to speed up the training process.
- **KNeighborsClassifier**: Instance-based learning method that classifies instances based on the majority class of their k nearest neighbors.
- **LogisticRegression**: Linear regression model used for binary classification problems.
- **NaiveBayes**: Probabilistic model based on Bayes' theorem with an assumption of independence between features.
- **Perceptron**: Linear binary classifier that makes predictions based on a linear predictor function.
- **RandomForest**: Ensemble learning method that builds multiple decision trees and combines their predictions through averaging.
- **Ridge**: Linear model that uses L2 regularization to prevent overfitting by penalizing large coefficient values.
- **SVC**: Support vector machine algorithm that finds the hyperplane that best separates classes in a high-dimensional space.

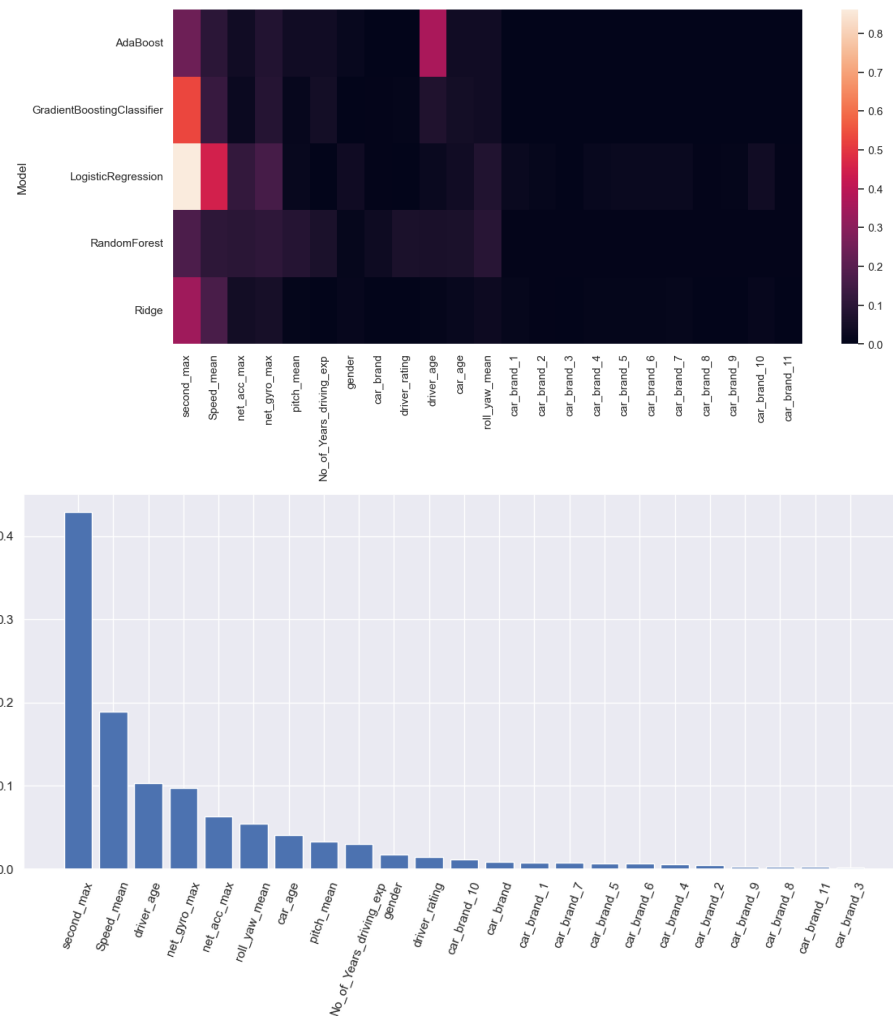
	Model name	Train Accuracy	Train Recall	Train F2 Score	Validation Accuracy	Validation Recall	Validation F2 Score	Fitting time (s)	Predicting time (s)
0	AdaBoost	0.696246	0.696246	0.696129	0.666316	0.573840	0.523077	1.6781	0.0059
1	DecisionTreeClassifier	1.000000	1.000000	1.000000	0.624737	0.396624	0.374353	0.4157	0.0004
2	ExtraTreesClassifier	1.000000	1.000000	1.000000	0.730526	0.276371	0.298270	0.2999	0.0149
3	GradientBoostingClassifier	0.747978	0.747978	0.747553	0.703684	0.516878	0.494949	8.2784	0.0023
4	HistGradientBoostingClassifier	0.845596	0.845596	0.844833	0.732105	0.411392	0.420078	1.0177	0.0049
5	KNeighborsClassifier	0.843354	0.843354	0.839388	0.594211	0.561181	0.488073	0.0018	0.0553
6	LogisticRegression	0.647679	0.647679	0.647535	0.660000	0.632911	0.562219	0.6770	0.0001
7	NaiveBayes	0.610364	0.610364	0.595188	0.737895	0.413502	0.423875	0.0072	0.0005
8	Perceptron	0.609265	0.609265	0.608568	0.630526	0.599156	0.527489	0.0198	0.0035
9	RandomForest	1.000000	1.000000	1.000000	0.731579	0.390295	0.401825	0.9246	0.0136
10	Ridge	0.648163	0.648163	0.648123	0.652105	0.643460	0.566283	0.0121	0.0069
11	SVC	0.719541	0.719541	0.719405	0.651053	0.618143	0.548484	11.3840	1.4822

In terms of accuracy, Decision Trees, ExtraTrees and RandomForest has perfect accuracies on the training data but filter on the validation data. This suggests overfitting and thus they cannot be used.

Models such that Logistic Regression, Gradient Boosting, Ridge and Perceptron maintain their accuracy even in the Validation Set. This suggests these models captured the underlying trends.

## E. Feature Importance

After training the models, we also averaged the features to find which ones are more important to the classification process.



As if observed in this plot, The car brands do not contribute much to the classification process. Therefore, by removing it we can reduce the noise in the model and potentially improve model accuracy.

	Model name	Train Accuracy	Train Recall	Train F2 Score	Validation Accuracy	Validation Recall	Validation F2 Score	Fitting time (s)	Predicting time (s)
0	AdaBoost	0.693477	0.693477	0.693477	0.654211	0.569620	0.515464	1.5994	0.0054
1	DecisionTreeClassifier	1.000000	1.000000	1.000000	0.629474	0.432489	0.404180	0.3954	0.0004
2	ExtraTreesClassifier	1.000000	1.000000	1.000000	0.743158	0.364979	0.383422	0.2530	0.0147
3	GradientBoostingClassifier	0.765295	0.765295	0.764819	0.716842	0.489451	0.478548	8.0136	0.0025
4	HistGradientBoostingClassifier	0.848717	0.848717	0.847173	0.754211	0.327004	0.352433	1.0301	0.0060
5	KNeighborsClassifier	0.847486	0.847486	0.843924	0.586316	0.527426	0.461595	0.0091	0.1454
6	LogisticRegression	0.646537	0.646537	0.646354	0.648421	0.632911	0.557621	0.4048	0.0001
7	NaiveBayes	0.606320	0.606320	0.590143	0.741053	0.417722	0.428571	0.0041	0.0003
8	Perceptron	0.608122	0.608122	0.607975	0.638947	0.630802	0.552476	0.0157	0.0001
9	RandomForest	1.000000	1.000000	1.000000	0.728421	0.383966	0.395308	0.7587	0.0139
10	Ridge	0.646844	0.646844	0.646774	0.641053	0.635021	0.556171	0.0065	0.0112
11	SVC	0.684819	0.684819	0.684789	0.642632	0.645570	0.563951	813.2024	1.2701

As can be observed here, some of the models improved after car\_brand was removed. Among all the models here, Logistic Regression, Ridge and Perceptron seem to perform the best. Therefore we will be tuning them now.

## F. Hyperparameter Tuning

Hyperparameter tuning is a critical step in the machine learning pipeline that involves selecting the optimal set of hyperparameters for a given model. Hyperparameters are initial configuration settings that are not learned during training but are instead specified prior to training. Tuning these hyperparameters can significantly impact the performance of a model.

In this project, hyperparameter tuning was conducted to optimize the performance of various machine learning models. The process involved systematically searching through a predefined set of hyperparameters and evaluating the model's performance using cross-validation.

Model	Tested Parameters	Best Parameters
Logistic Reg	max_iter : [1000, 1500], C : [0.1, 1, 10], penalty : ['l2', 'l1', 'elasticnet']	C : 0.1, max_iter : 1000, penalty : 'l2'
Ridge	alpha : [0, 0.01, 0.1, 0.2, 0.5, 0.75, 0.9, 1, 5, 10]	alpha : 0.001
Perceptron	penalty : ['l2', 'l1', 'elasticnet'], alpha : [0.00001, 0.00005, 0.0001, 0.0005, 0.001]	penalty : 'elasticnet', alpha : 0.00005

G. Final Model

Our final model is a Logistic Regression model with the following hyperparameters: C=0.1, Max\_Iter = 1000 and L2 penalty. These are the final metrics from our model.

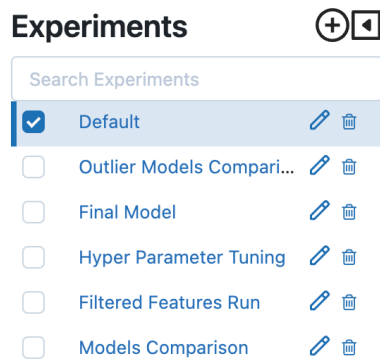
Metric	Value
Train Accuracy	0.6467
Train Recall	0.6227
Train F2 Score	0.6287
Test Accuracy	0.6521
Test Recall	0.6339
Test F2 Score	0.5519
Validation Accuracy	0.6468
Validation Recall	0.6329
Validation F2 Score	0.5570

6. Model Deployment & Monitoring

## A. Tracking Experiments with MLFlow

MLflow was utilized in the project for several reasons. Firstly, MLflow provides a comprehensive platform for managing our machine learning models. By using MLflow, we could easily organize and track experiments, enabling efficient comparison of different models and hyperparameters. Additionally, MLflow's model registry facilitated model versioning and collaboration among team members, ensuring reproducibility and consistency across experiments. Moreover, the built-in integration with sklearn streamlined the process of logging and tracking.

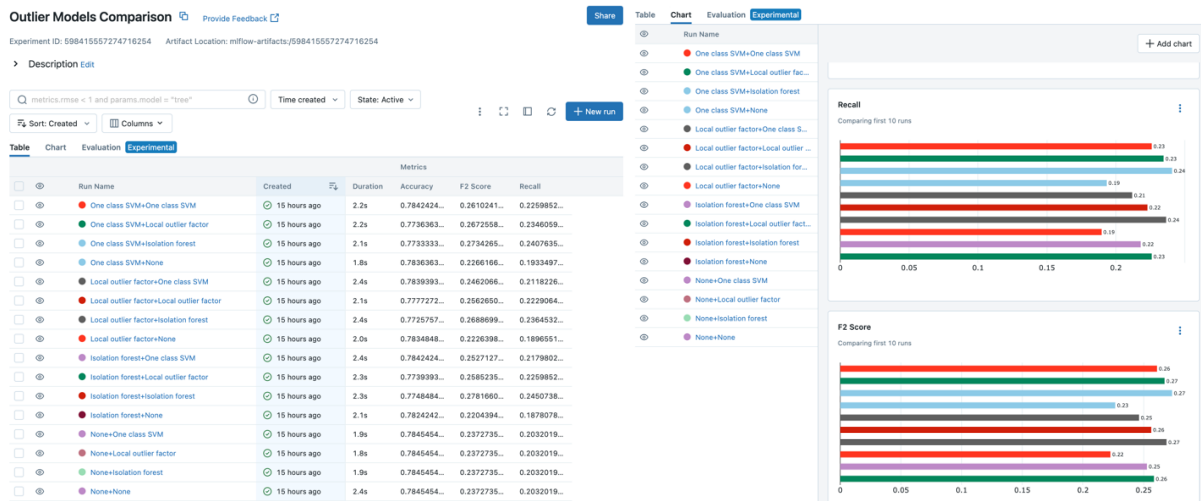
By using the experiments argument, we segmented our project into 5 Machine Learning Experiment phases as shown below.

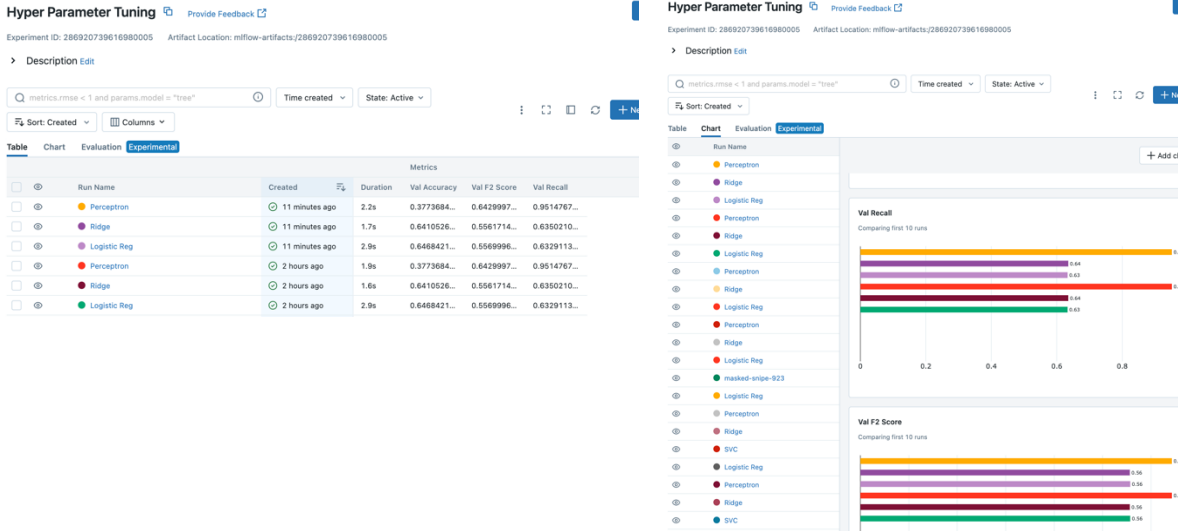
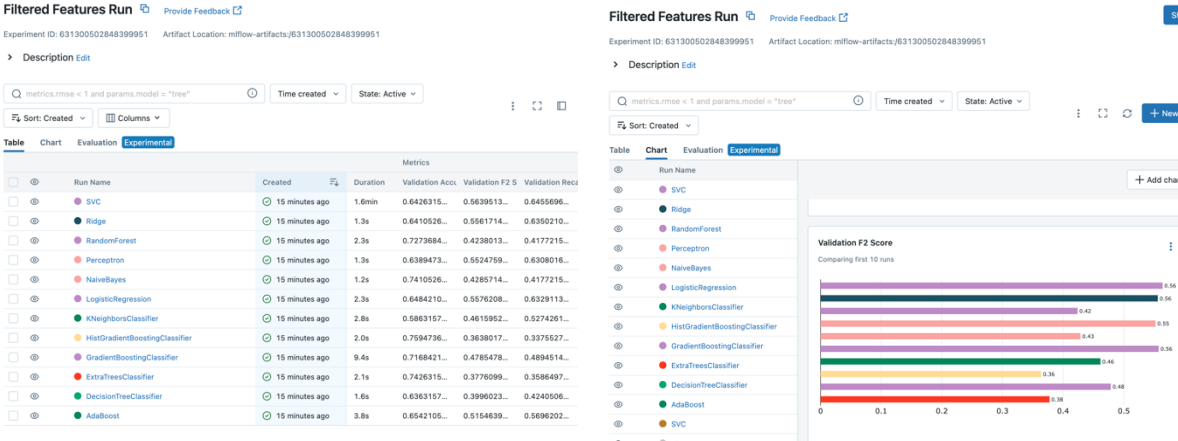
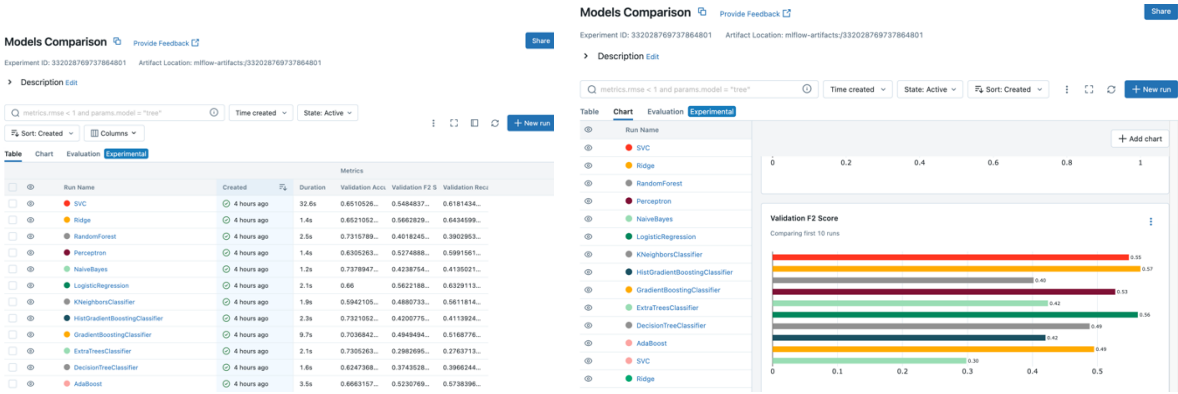


In the Outlier Models phase we have our three outlier detection algorithms and their corresponding RandomForest accuracy. Our Models Comparison is the main loop which compared the validation accuracy of all our models. The Filtered Feature Run was our rerun of the models comparison with car\_brand removed. Hyper Parameter tuning and Final model are self-explanatory.

Along with recording the scores, we also recorded the fitting time and prediction time.

We also used the Models Registry Feature to store our final models. In the future these could be used to deploy models too.





**Registered Models** [Create Model](#)

Filter registered models by ...

Name	Latest version	Staging	Production	Created by	Last modified	Tags
Final Logistic Regression	Version 1	—	—		2024-02-08 15:...	—
Logistic Reg - Tuning	Version 1	—	—		2024-02-08 15:...	—
Ridge - Tuning	Version 1	—	—		2024-02-08 15:...	—

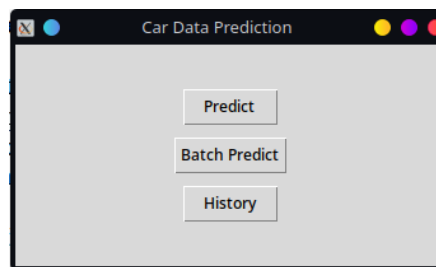
## B. Deployment to Tkinter app

Finally, we deploy the model via the pickle library by exporting it as a pickle file. We also have to export the transformer pipeline so that we can use the fitted pipeline to transform user inputs into the data appropriate for the model.

For the Tkinter app, we build the app such that it has the following main features:

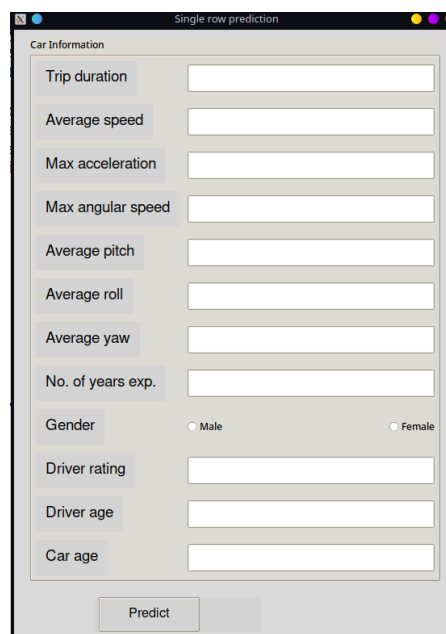
1. Ability to make single row prediction right from the app.
2. Ability to make batch prediction by inserting an excel or CSV file.
3. View past prediction data made through single row or batch prediction.

Firstly, we built a home page that allows user to choose what they want to do as shown:



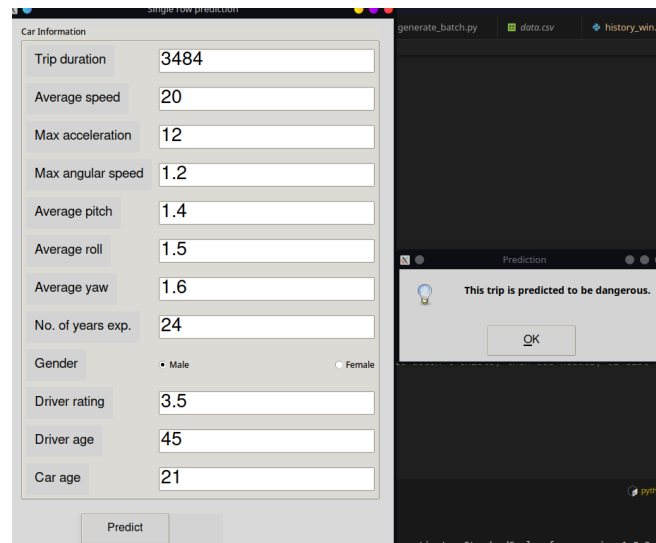
In the home page, users can choose whether they want to do single predict, batch predict, or view the history.

If the user click on “predict”, the app will open up a new window where it lets users input a series of data with regards to a trip and predict whether the trip is dangerous.

A screenshot of a Tkinter window titled "Single row prediction". The window contains a form with various input fields and a "Predict" button at the bottom. The form is titled "Car Information" and includes the following fields: "Trip duration", "Average speed", "Max acceleration", "Max angular speed", "Average pitch", "Average roll", "Average yaw", "No. of years exp.", "Gender" (with radio buttons for "Male" and "Female"), "Driver rating", "Driver age", and "Car age". Each field is a text entry box with a light gray border. The "Predict" button is a simple rectangle with a thin border and a light gray fill.

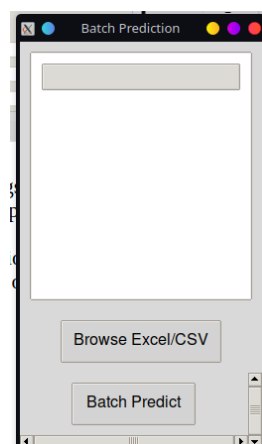
If the user enters some data along those entries and click on predict, the data will then perform the following steps: Firstly, combine the yaw and roll mean the same way as we did when we try to tackle multicollinearity. Secondly, transform the data with the transformer pickle file. Thirdly, pass the data to the model to make a prediction, and output a predicted value. And then translate the value to whether is dangerous or not and display to the user:





Additionally, the data entered alongside the predicted value and timestamp are appended into the history, so users can view their prediction again at a later time.

If user attempts to do batch prediction, they will be presented with an empty tree view and a button to browse and load an excel or csv data, and another button to perform batch predict:



The user must first load either an excel or CSV files, and we assume the column names for those data files are already in the format appropriate for the transformer and the model. Once the user loads a data file, the tree view will be updated with the entire data fully display.

Batch Prediction

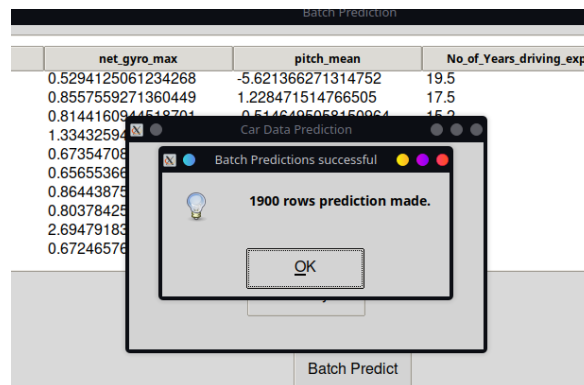
	second_max	Speed_mean	net_acc_max	net_gyro_max	roll_mean	yaw_mean	pitch_mean	No_of_Years_driving_exp	driver_rating	dr
833.1	9.280290423781384	12.70520804707382	0.5294125061234268	34.692004091593176	20.485219555976727	-5.621366271314752	19.5	3.2099998	49.0	
894.1	8.436775832495835	12.554071446422878	0.8557559271360449	32.80929568538757	8.366544357198993	1.228471514766505	17.5	4.1099997	48.0	
795.1	8.59262761145529	13.01479532468134	0.8144160944518701	32.78941487074762	0.5668738337276054	-0.5146495058150964	15.2	4.1400003	48.5	
980.0	7.60901395622288	12.355854922359445	1.3343259478114462	42.426208757955344	18.765979069958213	0.5698419154980259	16.2	3.65	48.1	
1001.3	7.054409748216338	12.839255281670535	0.6735470858310543	33.4862864520284	13.036274352958324	-0.2672055440184987	15.6	3.48	48.7	
904.6	9.912559822431127	12.863415731394458	0.6965536616085205	74.94009961038752	25.03234543195351	-0.819884894751453	18.8	3.55	48.5	
958.4	10.860267856027178	12.802484077745158	0.864438751434661	-1.1103442738472993	3.1067459592622315	-1.469239395141089	18.6	3.5700002	48.2	
1106.2	9.66192756479398	13.138632230824006	0.8037842537179086	27.683896261902724	17.311054282796086	7.035383947829303	18.1	3.6	47.9	
675.3	10.190844763282849	13.387411730897488	2.694791830177067	64.41761841280923	15.593031097986204	-0.2177728265019284	14.2	3.48	48.3	
1069.0	6.9945793033394255	13.225435448702006	0.6724657633418247	28.72869137854583	15.891980435479176	-1.2759630787863571	16.4	3.8400002	48.9	

Browse Excel/CSV

Batch Prediction

The user can now perform a batch predict, which will follow the same steps as the single row prediction. The difference is that instead of displaying the prediction on a message box, the

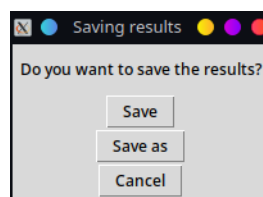
message box will display the number of rows predicted to inform the user on the number of predictions made.



Furthermore, the tree view will also be updated again to display not only the predicted value, but also predicted time stamp.

roll_yaw_mean	label	prediction_time
40.28870025388867	not dangerous	2024-02-08 18:50:47.28261
33.85925202735834	dangerous	2024-02-08 18:50:47.28261
32.79431465222852	dangerous	2024-02-08 18:50:47.28261
46.39121856588503	dangerous	2024-02-08 18:50:47.28261
35.93619037264117	dangerous	2024-02-08 18:50:47.28261
79.01035911473538	not dangerous	2024-02-08 18:50:47.28261
3.2992021553487274	dangerous	2024-02-08 18:50:47.28261
32.650738316640016	dangerous	2024-02-08 18:50:47.28261
66.37323711918421	not dangerous	2024-02-08 18:50:47.28261
32.83127701575719	dangerous	2024-02-08 18:50:47.28261

After the user made a batch predict, they will be prompted whether they want to save the result. The prompt will be given 3 options, save, save as, and cancel. Save will overwrite the data file that was used to load the data into the Tkinter app. Save as will save the data in a separate file, and cancel will do nothing.



Lastly, regardless of whether the user save the result, the result will still be appended into history, so the user can still refer back in case they didn't save it at first and can also compare with the predictions made with the single row predictions.

Finally, when the user wants to view the history, the user will simply click on history on the home window and the app will spawn another window with the tree view of all the past predictions made. The way it works is that we have a "storage" data hidden under the .data folder, where it stores a file called history.csv which stores all the past prediction data. This allows for persistent storage so when the user close the app and rerun it, the data will persist. So when the user made a prediction, the data will be appended into it, and when the app tries to retrieve the data when the user wants to view the history, the program will read that file. This is how the history looks window looks like after making some predictions:

Prediction History													
second_max	Speed_mean	net_acc_max	net_gyro_max	pitch_mean	No_of_Years_driving_s	gender	driver_rating	driver_age	car_age	roll_yaw_mean	label	prediction_time	
4.0	5.0	4.0	3.0	5.0	7.0	1.0	4.0	5.0	6.0	7.81024967590668	not dangerous	2024-02-08 13:33:2	
7443745278.0	5.0	4.0	3.0	5.0	7.0	1.0	4.0	5.0	6.0	7.81024967590668	dangerous	2024-02-08 13:33:2	
833.1	9.2802904237813	12.7052080470738	0.52941250612342	-5.6213662713147	19.5	1.0	3.2099998	49.0	22.8	40.2887002538886	not dangerous	2024-02-08 13:34:1	
894.1	8.4367758324958	12.5540714464226	0.85575592713604	1.2284715147665	17.5	1.0	4.1099997	48.0	21.9	33.859252027358	dangerous	2024-02-08 13:34:1	
795.1	8.5926276114552	13.0147953246813	0.81441609445187	-0.5146495058150	15.2	0.0	4.1400003	48.5	18.1	32.794314652228	dangerous	2024-02-08 13:34:1	
980.0	7.60901395622286	12.3558549223594	1.33432594781144	0.5698419154980	16.2	0.0	3.65	48.1	27.5	46.391218565885	dangerous	2024-02-08 13:34:1	
1001.3	7.0544097482163	12.8392552816705	0.6735470858310	-0.2672055440184	15.6	0.0	3.48	48.7	21.5	35.3361903726411	dangerous	2024-02-08 13:34:1	
904.6	9.9126598224311	12.8634157313944	0.6565536616085	-0.8138848947514	16.8	1.0	3.55	48.5	23.0	79.010359114735	not dangerous	2024-02-08 13:34:1	
958.4	10.8602678560271	12.8024840777451	0.8644387514346	-1.4692393951410	18.6	0.0	3.5700002	48.2	21.8	3.2992021553487	dangerous	2024-02-08 13:34:1	
1106.2	9.6619275647939	13.138632230824	0.8037842537179	7.0353839478293	18.1	1.0	3.6	47.9	22.7	32.650738316640	dangerous	2024-02-08 13:34:1	

## 7. Conclusion

In summary, with our vast data of cab trips and sensor information, we perform bulk insert on the dataset and store in a Microsoft SQL Server Database. Using Python to do ETL and store in HDF format to reduce space and load times. We can then use the data to do EDA and data cleaning and preprocessing, then transform the data for machine learning model and model the data, and then finally deploy the model on Tkinter app to let user make predictions.

In the process, we learnt that the duration of the trip contributes the most on what makes a trip dangerous, while the car brand makes the least. With this knowledge, we can better understand how we can mitigate the issues, such as only accepting short distance trip, and understand what not to bother the most, such as ignoring car brands used for the cabs when analyzing for dangerous trips.

And by deploying the model and serving the managers, we also allow them to enter new information into the model and inform the manager whether a trip is dangerous or not. This allows the managers to understand what makes a trip dangerous and make informed decision to formulate new or better rules for driving.