**Practical AI Phase Two**

**Project Report**

**Module Code:** ST1508

**Class:** DAAA/FT/2B/05

**Group No:** 03

**Members:**

Ryan Ueda Teo Shao Ming (2122924)

Chia Wee Leong (2122883)

Nimmagadda Vineesh (2123165)

Table Of Contents

**Introduction:** What Our Project Is About 1

**Section 1:** Data Wrangling & Preprocessing

**Section 1.1:** Dataframe Aggregation 1

**Section 1.2:** Feature Importance 5

**Section 1.3:** Principal Component Analysis 6

**Section 2:** Outlier Detection 7

**Section 3:** Imbalanced Data Handling 9

**Section 4:** Machine Learning Modeling

**Section 4.1:** Model Selection 12

**Section 4.2:** Experimentation With Transformations 13

**Section 4.3:** Hyperparameter Tuning 14

**Section 5:** MLFlow Experiment Monitoring 15

**Section 6:** Graphical User Interface Model Deployment 16

**Section 7:** References 19

**Introduction**

As a continuation to Phase One of this project, in this phase, the objective is to be able to deploy prediction models in the form of a graphical user interface. This is done through a series of MLOps processes. Firstly, application of advanced data processing techniques, such as feature engineering, outlier detection and imbalance data handling are applied on the dataset, in order to perform tasks that allow us to shape the data into a meaningful and useful state that can be fitted into a machine learning prediction model. Apart from the selection, building, optimization and scaling of these models, the learning cycles and experiments performed have to be trackable and monitored as well.

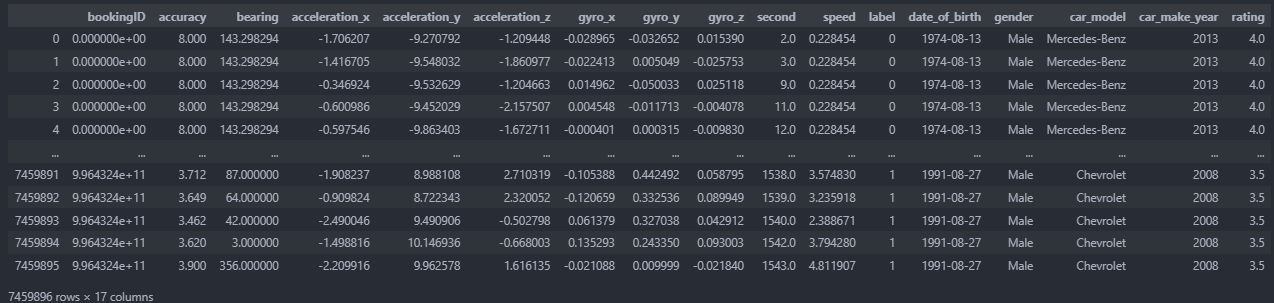
**Section 1: Data Wrangling & Preprocessing**

**Section 1.1: Dataframe Aggregation**

Similar to Phase One, we first extract the data from the SQL Server Database using a Data Pipeline established using sqlalchemy and pyodbc.

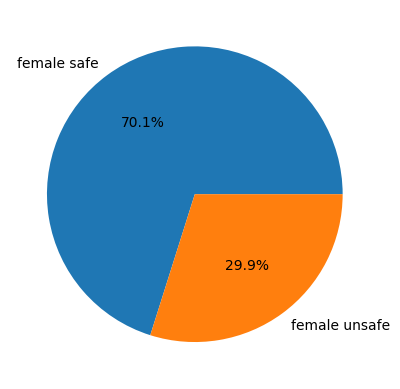
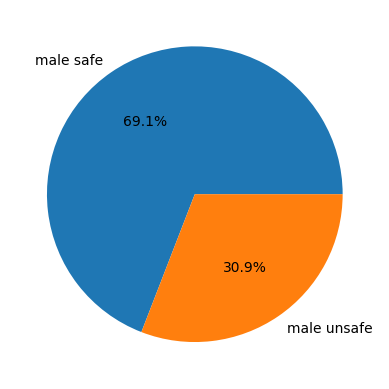
In order to prepare the data such that it is presented in a usable state for machine learning, we must first aggregate the dataframes. This is due to the fact that the X features and y target variables are in separate dataframes. Furthermore, there consists of multiple bookingIDs per target variable. As a result, the full dataset before aggregation consists of roughly 7.4 million rows, which is too large for efficient machine learning modeling.

Firstly, we perform a table join on the three dataframes, sensor, safety and driver. This is so that we are able to obtain a single dataframe consisting of both sensor and driver data (our features) and labels (target variable).

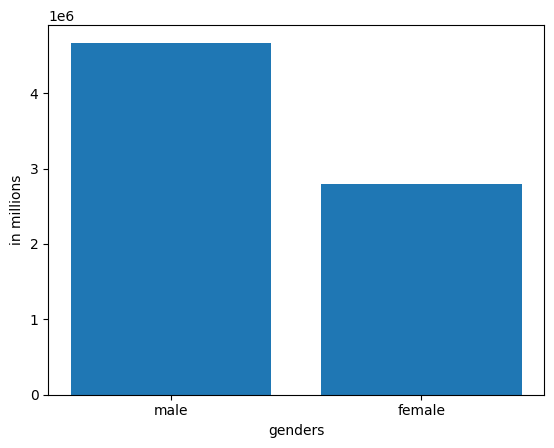


*Figure 1: Merged Dataframe*

Upon joining the three tables, we are able to observe that there are two categorical features, gender and car\_make. In this case, car\_make refers to the brand of the cars.

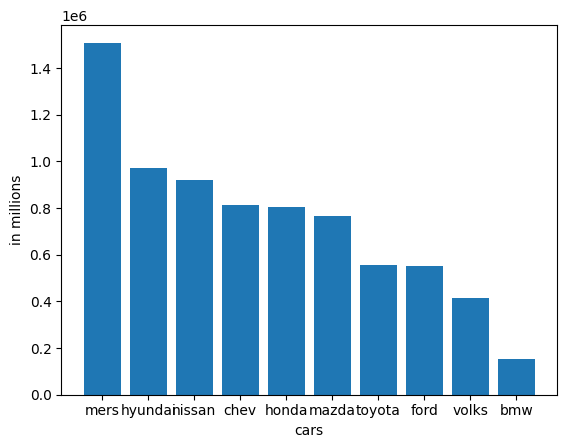
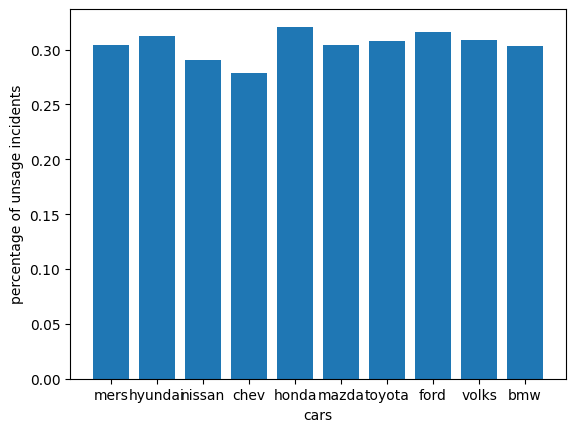


*Figure 2: Safe vs Unsafe Incidents (Male) Figure 3: Safe vs Unsafe Incidents (Female)*



*Figure 4: Total Number Of Male vs Female Drivers (in millions)*

Although there exists a large disparity in the number of male and female drivers, a quick check shows that the distribution of safe and unsafe incidents with drivers across both genders are actually very similar, differing with a margin of only one percent. This means that despite the huge difference in numbers in both genders of almost 2 million, the proportion of unsafe driving incidents across both genders are the same. This shows that there is no benefit to including this variable in the dataset, as it is unable to help the model better in its classification.



*Figure 5: Car Brands vs Percentage Of Unsafe Incidents Figure 6: Car Brands vs Counts (in millions)*

Similarly for car\_make, we can see that despite there being a large disparity in the number of cars from each brand, the percentage distribution of unsafe incidents by car are relatively similar. Once again, this shows that there is no statistical advantage to including this column in the training data.

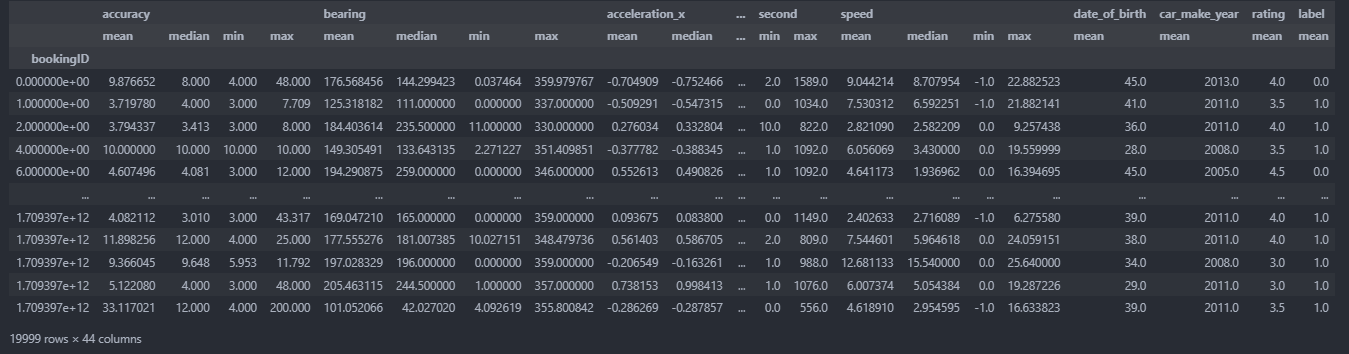
Next, we move on to aggregate the dataframe by using a groupby clause to group the rows by the column bookingID. On top of grouping the rows by bookingID, we can make use of multiple modes of aggregation in order to obtain new features, which can be used in machine learning model building and optimization. This is done by using pandas built-in aggregation function. For quantitative variables, we use the mean, median, minimum and maximum to obtain these values for each feature in the dataset. As for the categorical variables, we will use their mode for the sake of aggregation.

There is, however, one exception to this, which is the *date\_of\_birth* column. This is because although it is ordinal and quantitative in nature, its formatting (YYYY-MM-DD) results in it not being able to be recognized as an integer nor float value, but rather as a string. To resolve this issue, we subtract their year of birth with respect to the year the data was obtained, 2019, in order to calculate the driver’s age. This column will then result in being a column of integers, and can be aggregated along with the rest of the quantitative variables.

This provides us with up to more than four times the original amount of data present in the dataset. Taking into consideration that the original grouped dataset only contains 20000 by 10 samples, this significantly increases our sample size, which is likely to help the performance of the prediction model.

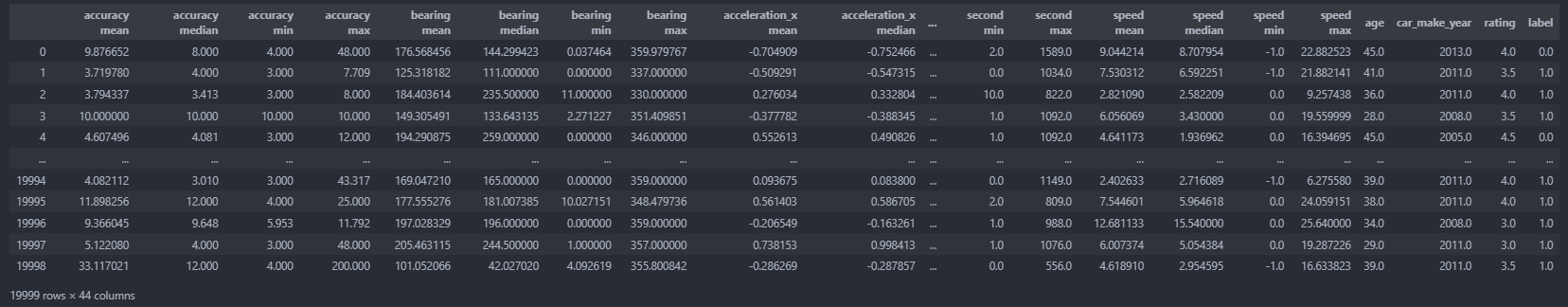


*Figure 7: Code Snippet For Dataframe Aggregation*



*Figure 8: Aggregated Dataframe*

To allow for better data cognizance and visibility, we rename columns by concatenating the method of aggregation to the original header (eg. accuracy mean).



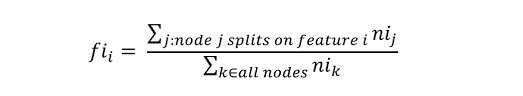
*Figure 9: Dataframe With Renamed Columns*

**Section 1.2: Feature Importance**

After aggregation, we are now left with a dataframe consisting of 43 features excluding the target variable ‘label’. This is an extremely high number of features considering the small row count of 20000, otherwise known as the curse of dimensionality. Thus, before the data is ready to be fitted into the machine learning prediction model, we must first reduce the number of features in the dataset via feature engineering. There are two methods we will utilize to do so.

Firstly, feature importance. Feature importance refers to the measurement of how much each feature contributes to the accurate prediction of the target variable. This is done using a Random Forest Classifier, and is computed by aggregating the results across all decision trees.

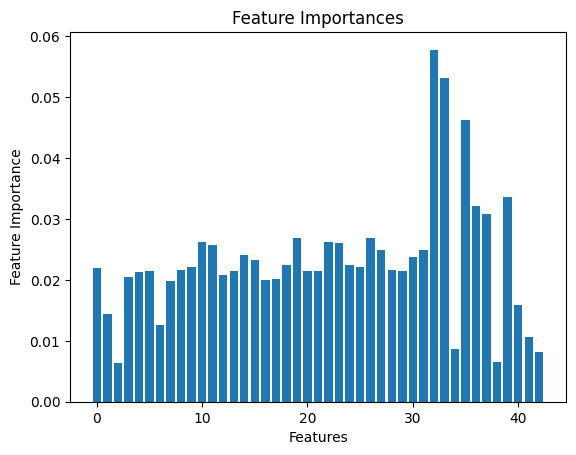
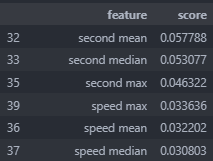
There are two main methods to accomplish this. The first method, Mean Decrease Impurity (MDI), calculates the average decrease in impurity across decision trees caused by each feature. Features that possess a greater average decrease in impurity are considered more important.



*Figure 10: Equation Representing Feature Importance*

The second method, Mean Decrease Accuracy (MDA), computes the average decrease in accuracy across decision trees, caused by randomly permuting each feature. Features that possess a higher average decrease in accuracy are considered more important. These scores give us greater insight into which features are driving the results produced, and can then be used to identify the most important features for the prediction task.

After obtaining the scores, a bar chart of each feature and their scores are plotted. This enables us to easily visualize how important each feature is. Each feature-score pair is then appended into a dataframe, where they are sorted in descending order. Through this, we can quickly identify which features are the most important, or choose which features to keep according to how many features we want.

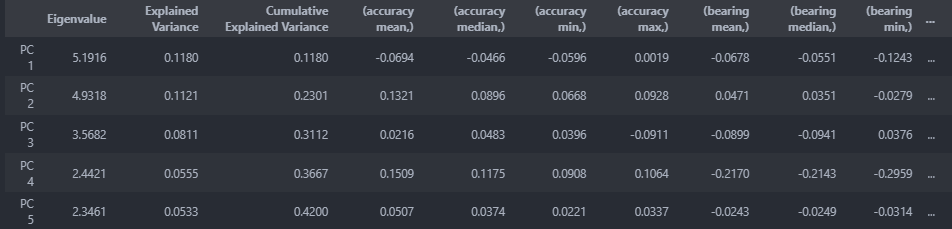
*Figure 11: Feature Importances Plot Figure 12: Sorted Feature Importances Dataframe*

**Section 1.3: Principal Component Analysis**

Secondly, Principal Component Analysis. Originally, the intention was to make use of Factor Analysis Of Mixed Data, or FAMD. This was because the merged dataset contained both quantitative and categorical data. PCA is only able to take in numerical values, and is unable to take in binary or OneHot encoded data either. However, since all the categorical variables have been dropped, we are able to use PCA instead.

PCA is a very popular and commonly used dimensionality reduction technique used in data analysis and machine learning. It works by transforming the data, such that the first principal component explains the most variance, while the second component explains the second most variance, and so on. This allows us to identify the more ‘important’ features in the dataset, and drop the arbitrary components without losing as little information as possible. Through PCA, we gain access to better visualization and performance in machine learning.

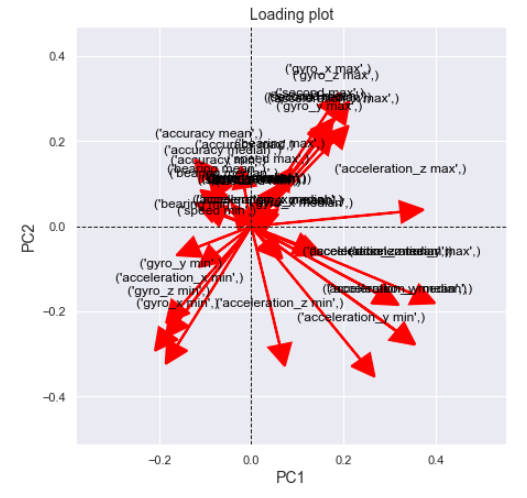
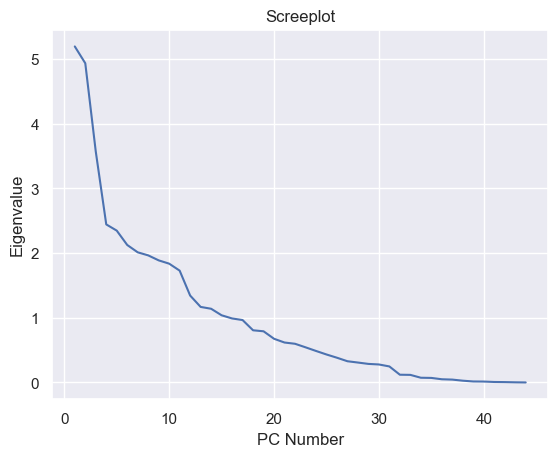
To implement PCA, we used scikit-learn’s built in PCA method. After fitting the dataset into the PCA method, we then compute eigenvalues along with variance explained per principal component. These results are then stored in a dataframe, containing information about each of the 40 principal components, namely eigenvalue, explained variance and cumulative explained variance.



*Figure 13: Principal Component Analysis Results*

Additionally, the scree plot and loading plot are plotted as well for better visualization. Upon obtaining results from the analysis, we need to interpret these results. This can be done through a variety of methods. Typically, an average approximation using all three methods are used to identify the principal components.

The first method is known as Kaiser’s Rule. By Kaiser’s Rule, principal components that possess an eigenvalue of 1 or greater should be extracted. The second method states that minimally, a cumulative variance of 80% should be kept. The last method is the scree plot elbow method. This method states that we should take the values on the left of the elbow in the scree plot. By taking the top two principal components, the loading plot is obtained as well.



*Figure 14: Screeplot Derived From PCA Figure 15: Loading Plot*

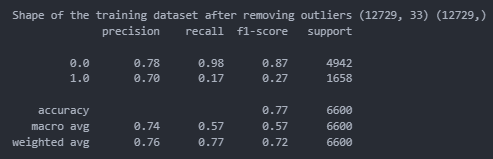
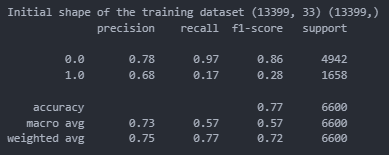
According to the scree plot above, we should take 13 principal components. However it is important to note that these rules are just guidelines and further experimentation is required to achieve good empirical results.

**Section 2: Outlier Detection**

In this section, we will be handling outliers in our data by performing outlier detection. We will experiment with a variety of different models in order to assess which will be the best choice for our data. Outlier detection, otherwise known as anomaly detection, is the process of identifying data points in a dataset that deviates significantly from the rest of the data. There are multiple benefits that can come from performing outlier detection. This includes and is not limited to: improved model accuracy, better data representation and reduction of overfitting. Outliers are known to contribute to overfitting, and removing them can lead to better generalization and robustness of the model.

**Isolation Forest**

A series of outlier detection models will be attempted. At the end of each experiment, the before and after classification reports will be displayed. The first model tested is the Isolation Forest model. Isolation Forest is an unsupervised algorithm that is based off the idea of isolating data points by randomly selecting features and splitting the data along these features. The objective is to isolate each instance as quickly as possible, by creating many short paths in the tree structure. Instances that are isolated very quickly, are considered to be outliers or anomalies. Isolation Forest is popular as it is simple to implement, and requires little to no preprocessing. It is able to scale well to large datasets and handle high-dimensional and multi-variate data.

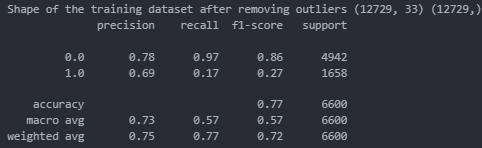
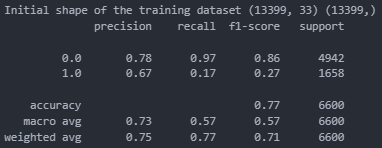


*Figure 16: Original Classification Report Figure 17: Isolation Forest Classification Report*

As shown above, we can see that the classification reports before and after basically do not differ at all, with only the recall and f1-score improving by 0.01. On the contrary, the model deemed over 1000 of the data points as outliers and removed them. This is considered quite a detrimental amount of data to be removed, as there is only an original training size of 13399.

**Local Outlier Factor**

Local Outlier Factor (LOF) measures the degree of abnormality of a data point with respect to its local neighborhood. This is done by calculating the mean distance between an instance and its k nearest neighbors. This information is then used to determine how isolated it is compared to its neighbors. The higher the degree of isolation, the more likely it is that said instance is an outlier. As LOF is a density-based algorithm, it is sensitive to the distribution of data and can handle complex and non-uniform distributed datasets with high-dimensional multivariate data.

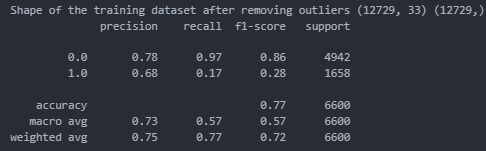
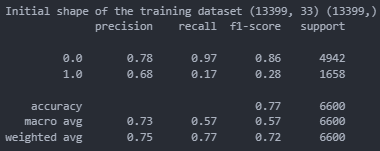


*Figure 18: Original Classification Report Figure 19: Local Outlier Factor Classification Report*

As shown above, it is clear that there is no noticeable difference in the classification reports.

**One-Class Support Vector Machine**

One-Class Support Vector Machine (SVM) aims to locate a decision boundary that separates the normal class from the anomalous classes. This is achieved by finding the hyperplane that maximizes the margin between the normal class and origin, while still being able to classify normal class data correctly. Instances that stray far from the hyperplane are considered to be outliers or anomalies. One-Class SVM is known to work well with complex distributions, or where only a small amount of data exists for anomalous classes. However it is sensitive to the choice of kernel function and vast selection of parameters.



*Figure 20: Original Classification Report Figure 21: One-Class SVM Classification Report*

Similar to before, there is no noticeable difference in the classification reports.

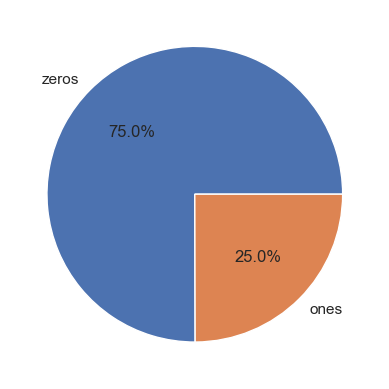
**Conclusion**

To conclude, after trying the three different outlier detection models, we have identified that none are able to produce a significant enough result for selection. Hence, we have decided to try fitting the model with data obtained from all three methods.

However, after performing these experiments, not only does there seem to be no improvement in the performance of the model, but the classification accuracy actually worsens. This result will be documented and shown in detail later on in the machine learning modeling section. This might be due to the fact that not only does the outlier detection model not perform very well, it removes a significant chunk of our data from our already miniscule set of training samples. As a result, we will not be using any outlier detection models to preprocess our data.

It is important to note that although there exists data that could be outliers, they are also likely to be real data that is useful for the model to obtain statistical inferences during training.

**Section 3: Imbalanced Data Handling**

****

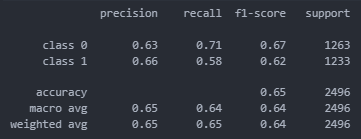
*Figure 22: Pie Chart Depicting Class Imbalance In Target Variable*

For the last part of our data preprocessing, we will be handling imbalanced data in the dataset. Imbalanced data refers to a scenario where the number of instances belonging to the minority class is significantly smaller than the majority class. This can cause machine learning algorithms to be biased towards the majority class, leading to increased difficulty in classifying instances from the minority class.

By plotting a pie chart showing the distribution of our target variable values, we are able to see that there is quite a significant class imbalance, with the zeros taking 75% and the ones taking up only 25% percent, representing a 3:1 ratio. In this section, we will cover various methods that are commonly used to deal with such imbalanced data.

**Random Under Sampling**

The first technique we will attempt is Random Under Sampling. Random Under Sampling. Random Under Sampling randomly selects and removes instances from the majority class until an even class distribution is achieved. Despite being a simple and effective way to handle imbalanced datasets, it can also result in the loss of key information and patterns within the majority class, impacting the machine learning algorithm’s capability of learning.

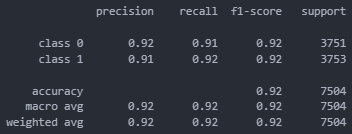


*Figure 23: Random Under Sampling Classification Report*

As shown above, we can see that this does not work particularly well. Although the f1-score is relatively close to one another, the recall has quite a wide variance and the general scores achieved from this model is not ideal.

**Random Over Sampling**

The second model we will experiment with is Random Over Sampling. Contrary to the Random Under Sampling, this technique randomly duplicates instances from the minority class until an even class distribution is achieved. However, this technique can lead to overfitting and poor generalization performance, as the algorithm may over-emphasize the minority class.

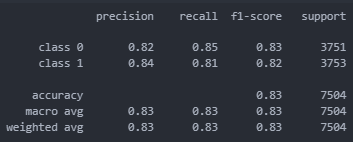


*Figure 24: Random Over Sampling Classification Report*

Here, we can see that this model performs significantly better than Random Under Sampling. The f1-score is almost 0.3 higher than previously, with the precision and recall achieving similarly impressive results. This is likely due to the fact that there exists a small amount of data. As compared to the previous technique which reduced the size of our data, Random Over Sampling increases the amount of data we have.

**Synthetic Minority Over-Sampling Technique (SMOTE)**

Synthetic Minority Over-Sampling Technique (SMOTE) is a variation of over-sampling. It selects two or more minority class instances and calculates the vector between them. This is then multiplied by a random number between 0 and 1, and added to the feature values of one of the selected instances to generate a synthetic instance. This process is then constantly repeated until an even class distribution is achieved. Often, SMOTE is more effective than over-sampling, as the synthetic samples generated are able to capture the underlying minority class distribution. However, SMOTE is also known to overfit and result in poor generalization performance. Furthermore, if the synthetic samples generated lack representation of the overall population, the machine learning algorithm can be overly influenced by these instances and harm its performance.



*Figure 25: SMOTE Classification Report*

In this case, we can see that SMOTE actually performed worse than Random Over-Sampling, but better than Random Under-Sampling. Once again, this is likely due to the size of our data. Since the synthetic samples generated are based off real data, the model overfits on the data created. This is because the amount of data SMOTE is pulling from is already small and unable to generalize well.

**Conclusion**

After experimenting with these three techniques for handling imbalanced data, it is clear that Random Over-Sampling is a far superior technique for handling data imbalance in our dataset. Not only does it resolve our issue of imbalanced data, but also provides us with more training samples for the model to be trained on. Thus, we will be using Random Over-Sampling.

**Section 4: Machine Learning Modeling**

**Section 4.1: Model Selection**

Based on the experiments and tests we have run in the previous section we have found Random over sampling to be the best form of oversampling and from the scree plot an elbow can be seen at 13 components. Hence in our final data transformation we have picked these to be our preliminary steps for data transformation. These steps are preliminary as we have only selected these through some simple baselines and not actual model comparisons.

Now the task is to select a model to use. There are multiple ways of testing machine learning models like AB testing or the multi armed bandit method. Although the multi-armed bandit method is useful to progressively reduce the number of models tested, statistical certainty takes a backseat in this method in favor of efficiency. However in our project implementation we do not have a large number of models to choose from therefore we have opted to place higher importance on statistical certainty and use the AB testing method instead where each model is trained fully on the data and their results are compared to find the best model. A summary of our experiments and their results can be seen below.

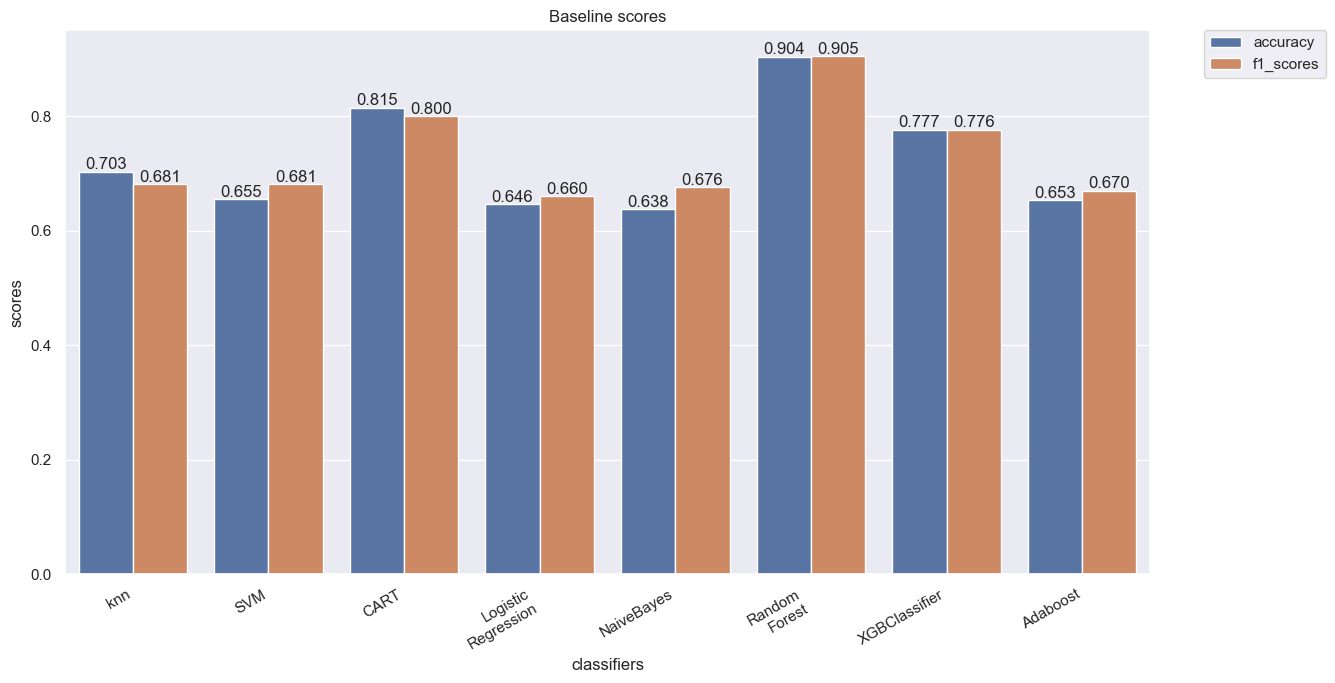
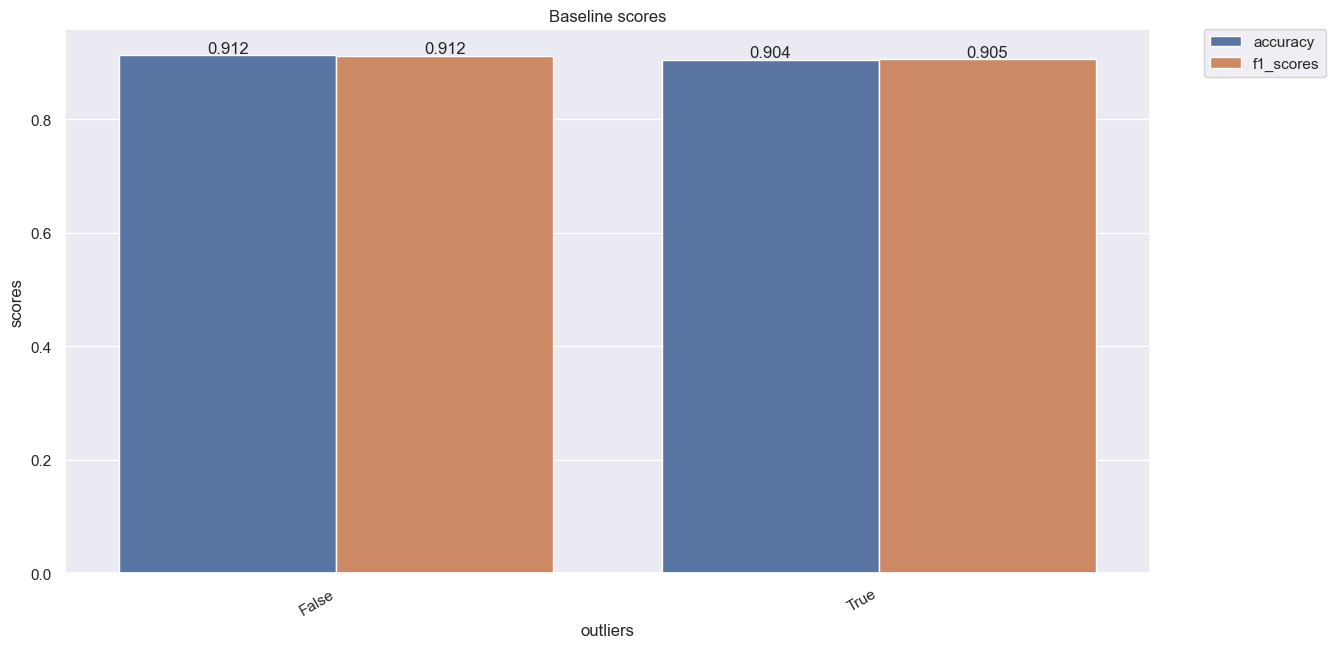
****

Figure 26: Summary of model performances

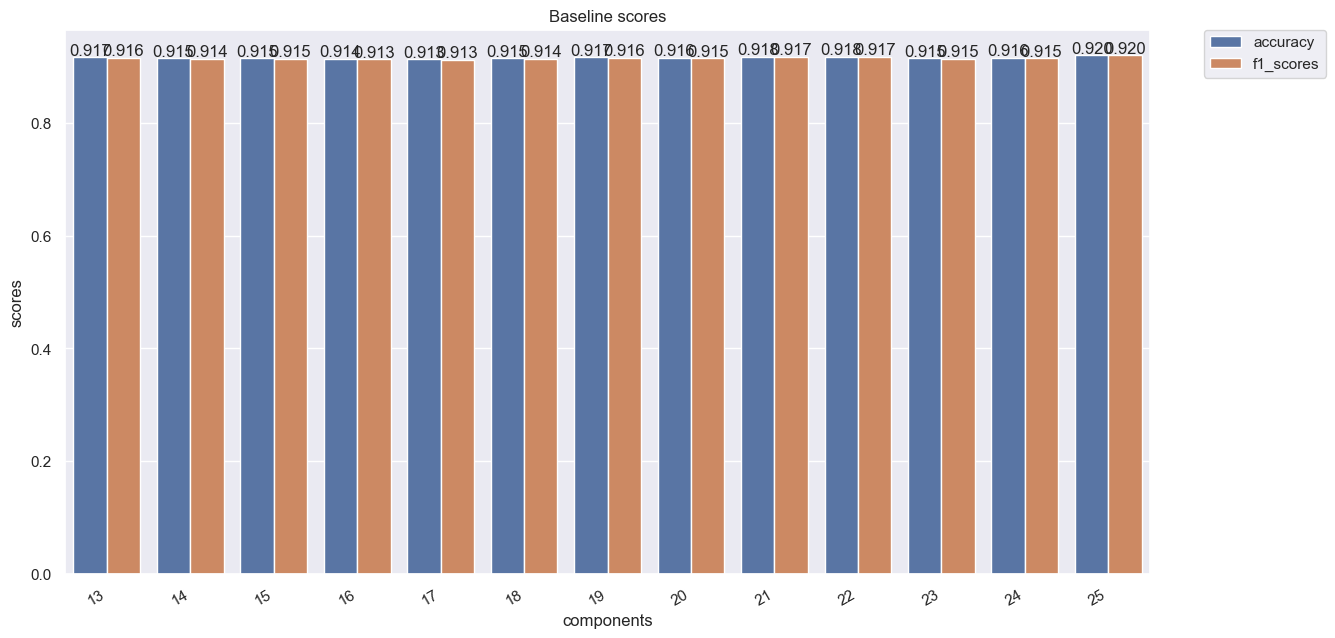
As seen from the bar chart above the random forest model outperforms the rest of the models by a large margin. Logistic regression and naive bayes seem to be the least suited for this task whereas Decision trees offer a comparable performance which is sensible considering that random forest is essentially an ensemble training method to train multiple decision trees together.

**Section 4.2: Experimentation With Transformations**

We have used our preliminary experiments and tests to define parameters for PCA and outlier detection. However these serve only as preliminary tests and are not parameters that have been tuned for the model selected. Hence it is imperative that we once again experiment with different parameters and steps for the model selected. We have decided to experiment with outlier detection and the components of PCA as in the first section we have seen that outlier detection does not seem to improve performance. Here are the summaries of our findings:

****

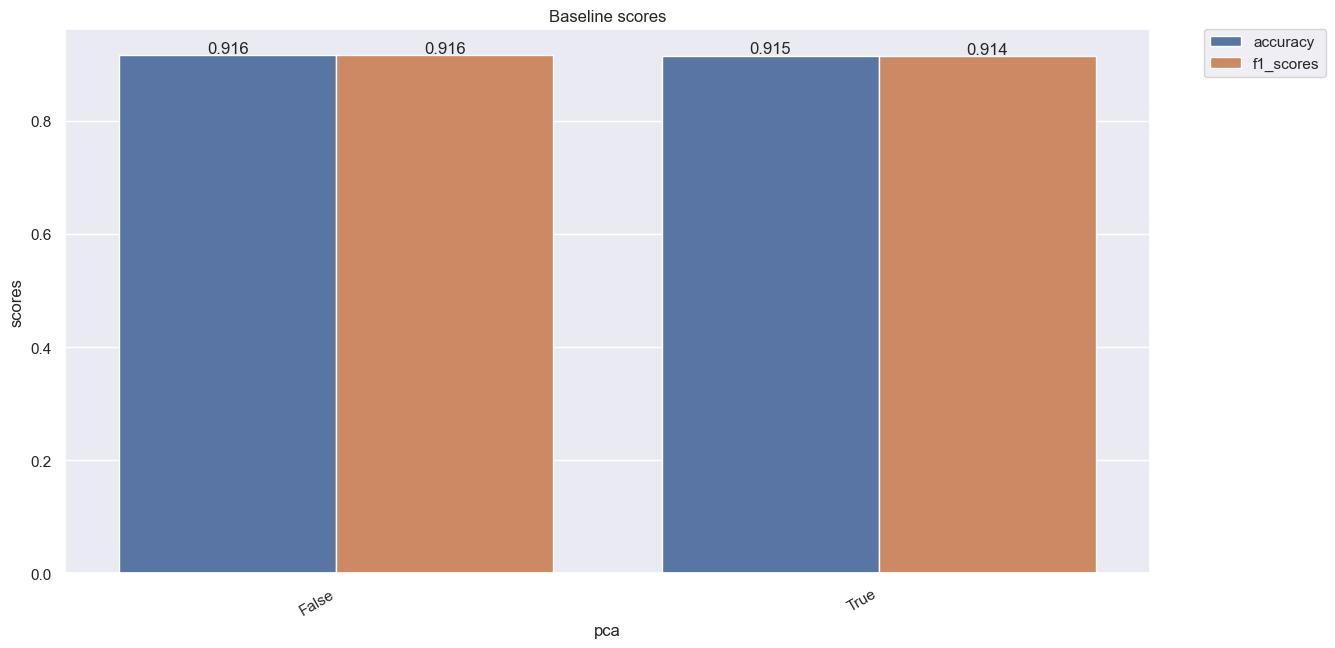
*Figure 27: Outlier Detection Results*

******

*Figure 28: PCA Components Results*

As seen from the bar charts above removing outliers actually worsens the performance of our model. This suggests that the outliers detected are actually statistically important data that is required to make good predictions. This also explains the lack of improvement in performance when experimenting with various types of outlier removal in the earlier sections. Furthermore, looking at the components of PCA we see that performance of the model gradually increases and peaks at 25 components. Hence through experimentation we have found the ideal transformations of data to be with PCA extracting 25 components and no outlier removal.

Lastly we decided to experiment on whether PCA is actually beneficial for our data. In some cases PCA does not summarize the features of the dataset well and we end up losing important information that will be useful for model prediction. In such cases it is recommended to fit the model on the raw data and select features based on the feature importances of the model used to improve performance. However, it must be noted that this will lead to higher computational costs as a large number of features are being fit into the model. Here is the summary of our results:

****

*Figure 29: Summary of presence of PCA experiment*

Looking at the bar graph we can see that not using PCA does improve performance slightly. However the performance only improves by ~0.01 which is not a significant improvement in exchange for the higher computational cost. Furthermore PCA is essentially a summarization of features and we can see that despite extracting only 13 components out of a potential 30 features the model still performs comparably to fitting on the raw data. This shows that using PCA is beneficial in terms of computational cost without sacrificing too much performance.

**Section 4.3 Hyperparameter Tuning**

Now that the ideal transformations have been found it is time to tune the hyperparameters of the model. Multiple methods of hyperparameter search exist in the scikit-learn library such as GridSearchCV, RandomizedSearchCV and so on. We have elected to use GridSearchCV as GridSearchCV is guaranteed to find the best parameters whereas RandomizedSearchCV just runs a series of random trials and picks the best parameters.. We have found the parameters n\_estimators=500, criterion='gini',max\_depth=50,max\_features=’log2’,min\_samples\_leaf=1 to be the best.

**Conclusion**

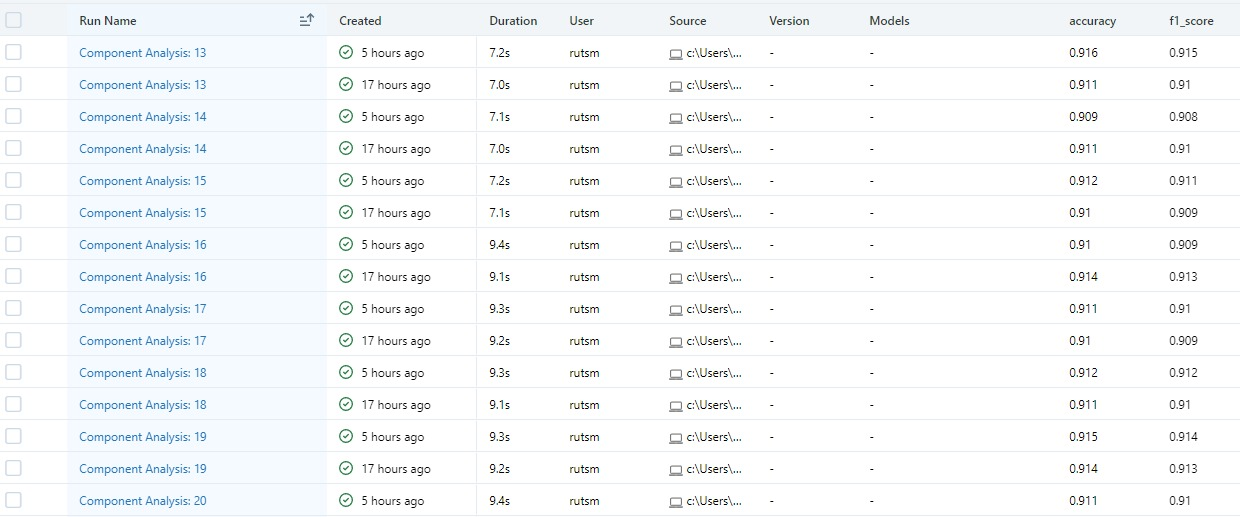
In conclusion, through all our experiments and tests we have found a random forest model with 23 components extracted using PCA and no outlier removal along with model parameters of n\_estimators=500, criterion='gini',max\_depth=50,max\_features='log2',min\_samples\_leaf=1

to provide the best performance to computational cost ratio where results are comparable to fitting the raw data into our model.

**Section 5: MLFlow Experiment Monitoring**

As seen above we have done many experiments and tests and through all these tests it is necessary to use a machine learning platform to log parameters and keep track of models. Although multiple platforms exist such as FBLearner Flow by Facebook or TFX by Google these are typically limited and only support a small set of built-in algorithms and are tied to each company’s infrastructure.

Hence we have picked MLFlow as our choice of machine learning platform. MLFlow is an open source machine learning platform which is compatible with many libraries and is most importantly compatible with sklearn models which is the library we are using in this project for our machine learning algorithms. MLflow has an open interface and is built around REST APIs and simple data formats that can be used from a variety of tools instead of only providing a small set of built-in functionality. We have logged parameters at every experiment and saved the final model as seen below.



*Figure 30: MLFlow logs*

**Section 6: Graphical User Interface Model Deployment**

Model deployment is an important step in the machine learning process, as it allows the trained machine learning model to be used in the real-world to make predictions and enable decision making based on the predictions generated by the model.

Deployment is part of the four main stages in building an ML model:

1. Prepare to deploy

* As done earlier, the machine learning model needs to be developed, trained and hypertuned for it to achieve its best state.

1. Validate the model

* The model is tested on the validation/testing set to ensure that the model performs well to unseen data, and the model is not overfitted

1. Deploy the model

* Model is deployed either online or via GUIs such as Tkinter

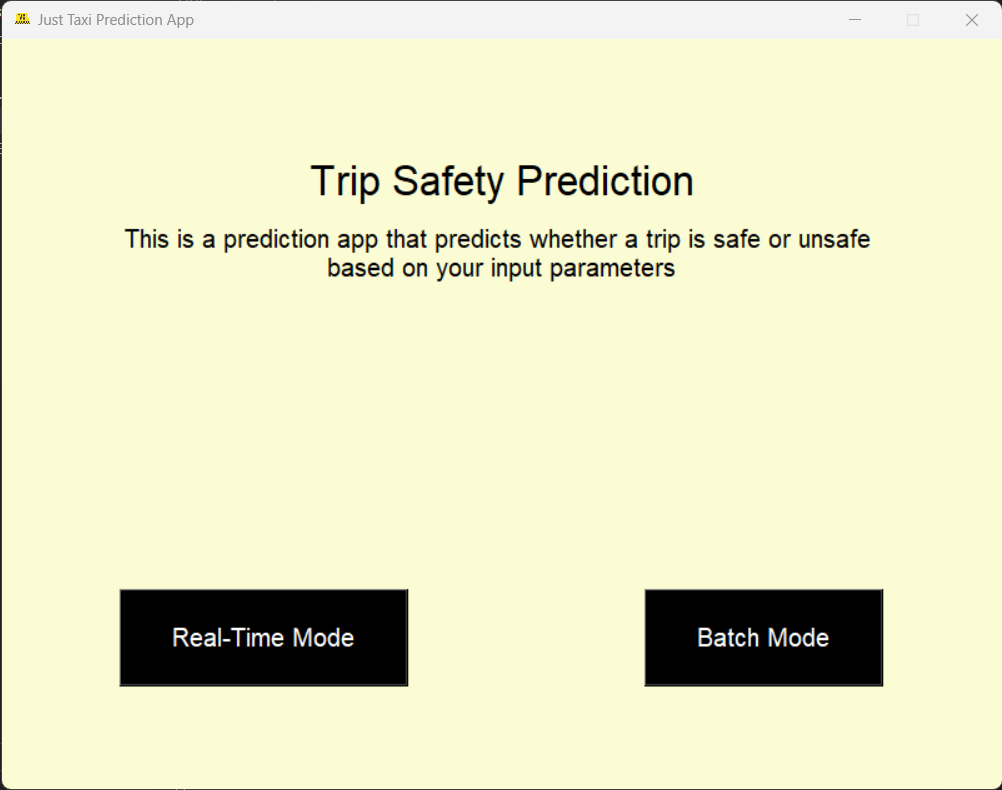
1. Perform Model Monitoring

* Ensure that the deployed ML model is running smoothing and accurately

For this project, we shall deploy the prediction model as a Graphical User Interface (GUI), using a software known as Tkinter.

Tkinter is the de facto way in Python to create a GUI and is included in all standard Python Distributions. It is easy to learn and use as it has a simple and intuitive interface, making it easy for developers to get started and build GUI applications with little experience. It has a large community for developers, making it easier to find solutions to problems faced while developing the project, and is suitable for a wide range of applications, from simple desktop utilities to complex business applications.

*Our group created the GUI as shown below:*

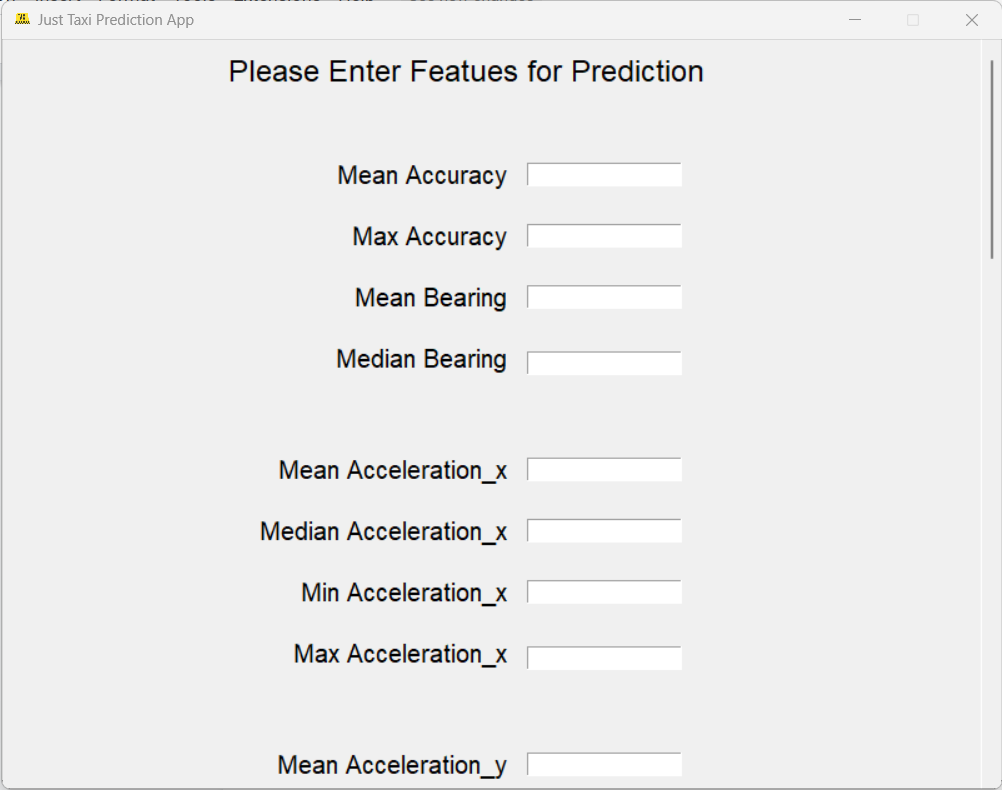
**Home Page**

*Figure 31: Home Page*

When the user starts running the GUI file, they will be greeted with this screen.

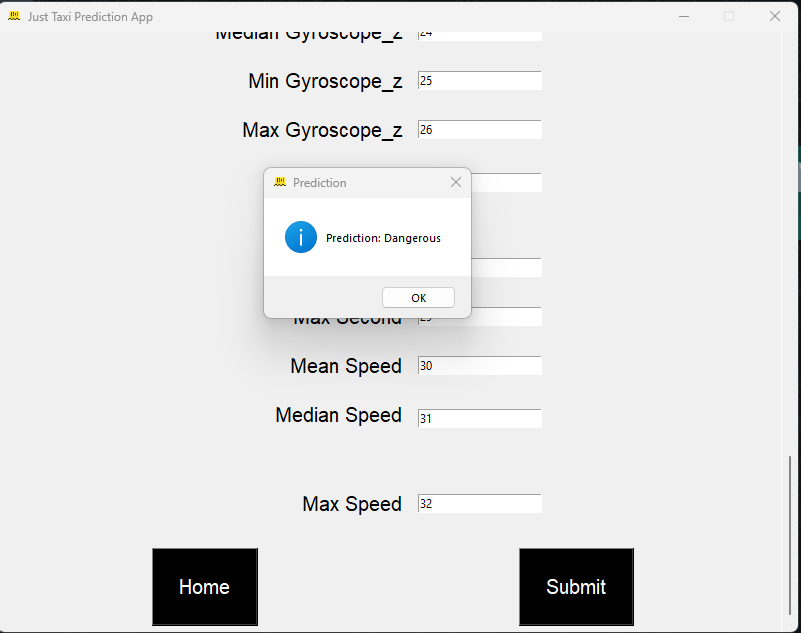
The top half of the screen describes what the app does, while the bottom half of the screen is used to navigate between Real-Time Mode and Batch Mode.

**Real-Time Mode**



*Figure 32: Real-Time Prediction Part 1*

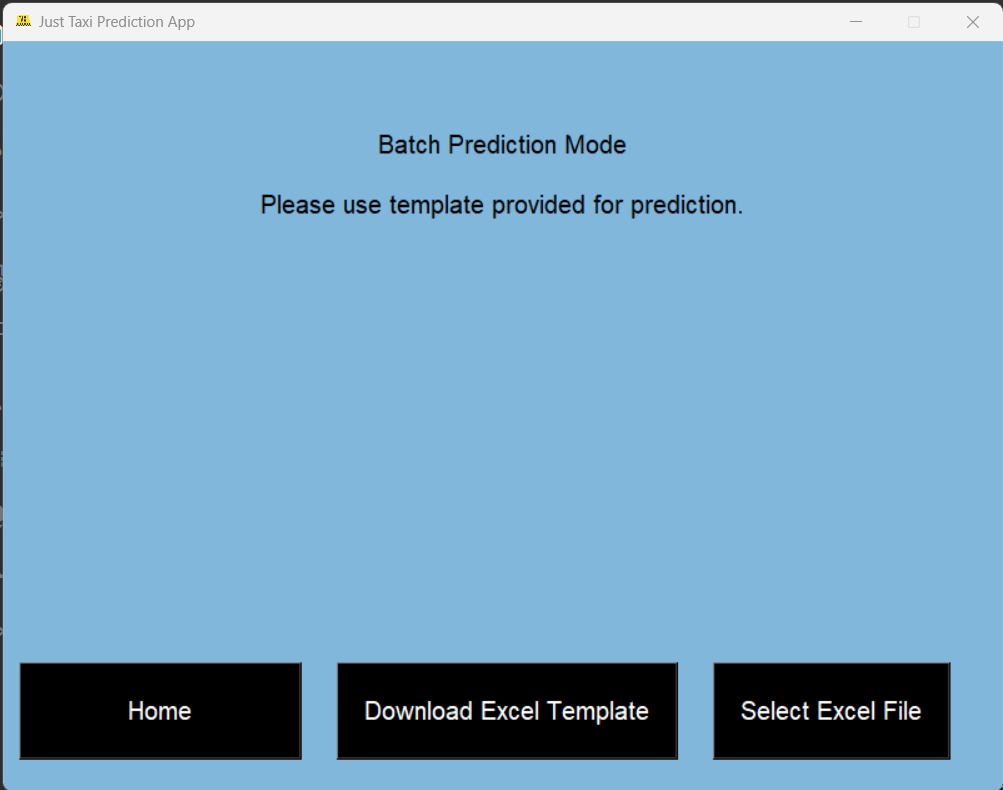
In this mode, users can enter the values for one booking ID manually and make a prediction by clicking on the submit button at the bottom of the page.



*Figure 33: Real-Time Prediction Mode Part 2*

To prevent the application from crashing, error handling is done to guide users on how to adjust their inputs for the model to be able to make predictions successfully.

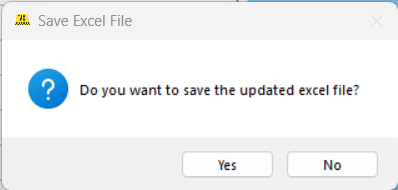
**Batch Mode**



*Figure 34: Batch Mode Prediction Part 1*

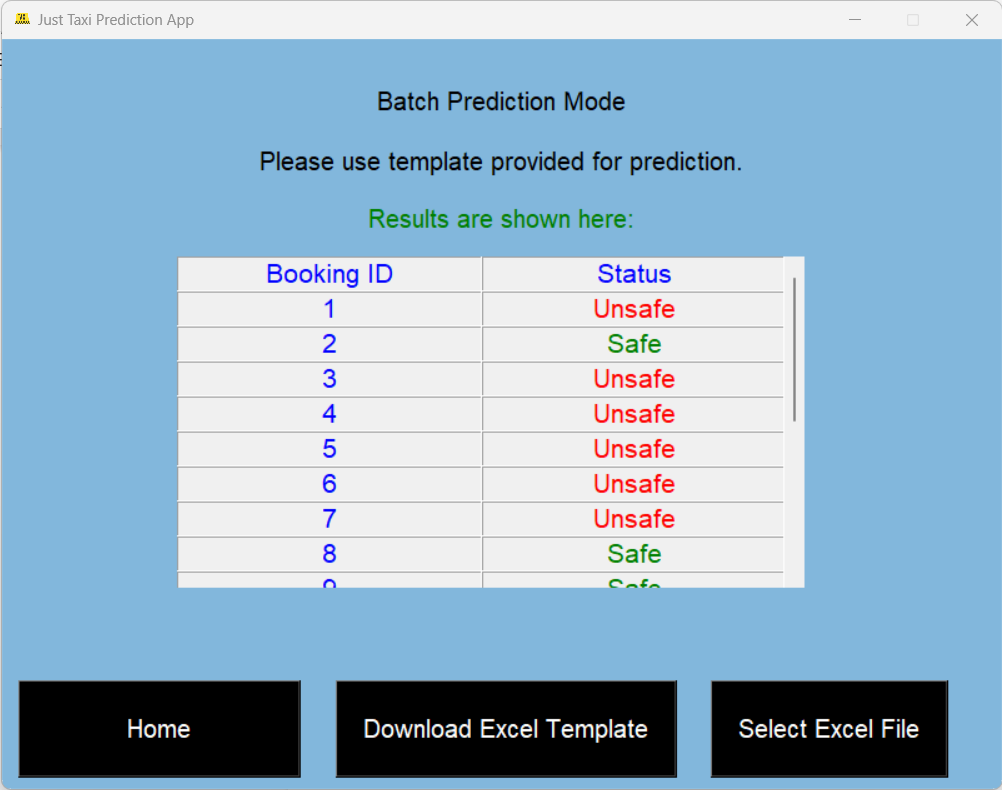
In this mode, users should first click on “Download Excel Template” to obtain the Excel file with the format required for the machine learning model to generate an output successfully.

After users fill in the template file with the required values, they can then click on “Select Excel File” and submit their completed Excel files for the model to predict and generate an output.



*Figure 35: Batch Mode Prediction Part 2*

Once the model has predicted on the data from the excel file, this popup will appear. When ‘Yes’ is clicked, the user will have a new excel file downloaded, with the predicted labels in the last column.



*Figure 35: Batch Mode Prediction Part 3*

This screen shows the generated output for all booking IDs.

**REFERENCES**

1. <https://www.sciencedirect.com/topics/engineering/outlier-detection>
2. <https://towardsdatascience.com/5-outlier-detection-methods-that-every-data-enthusiast-must-know-f917bf439210>
3. [http://scikit-learn.org/stable/auto\_examples/neighbors/plot\_lof\_outlier\_detection.html#:~:text=The%20 Local%20 Outlier%20 Factor%20(LOF,lower%20density%20than%20their%20neighbors.](http://scikit-learn.org/stable/auto_examples/neighbors/plot_lof_outlier_detection.html#:~:text=The%20Local%20Outlier%20Factor%20(LOF,lower%20density%20than%20their%20neighbors.)
4. <https://towardsdatascience.com/local-outlier-factor-for-anomaly-detection-cc0c770d2ebe>
5. <https://medium.com/datasciencearth/local-outlier-factor-7821b5651bc5>
6. <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html>
7. <https://www.analyticsvidhya.com/blog/2021/07/anomaly-detection-using-isolation-forest-a-complete-guide/>
8. <https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e>
9. <https://vwo.com/ab-testing/>
10. <https://hbr.org/2017/06/a-refresher-on-ab-testing>
11. <https://www.optimizely.com/optimization-glossary/ab-testing/>
12. <https://vwo.com/blog/multi-armed-bandit-algorithm/>
13. <https://cxl.com/blog/bandit-tests/>
14. <https://towardsdatascience.com/beyond-a-b-testing-multi-armed-bandit-experiments-1493f709f804>
15. <https://www.mastersindatascience.org/learning/statistics-data-science/undersampling/>
16. <https://machinelearningmastery.com/undersampling-algorithms-for-imbalanced-classification/>
17. [https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification](https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/)
18. <https://towardsdatascience.com/oversampling-and-undersampling-5e2bbaf56dcf>
19. <https://towardsdatascience.com/smote-fdce2f605729#:~:text=SMOTE%20stands%20for%20Synthetic%20Minority,imbalanced%20data%20in%20classification%20problems.>
20. <https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/>
21. <https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html>
22. [https://www.dominodatalab.com/blog/machine-learning-model-deployment#:~:text=Machine%20learning%20model%20deployment%20is,be%20accessed%20by%20end%20 users.](https://www.dominodatalab.com/blog/machine-learning-model-deployment#:~:text=Machine%20learning%20model%20deployment%20is,be%20accessed%20by%20end%20users.)
23. <https://towardsdatascience.com/3-ways-to-deploy-machine-learning-models-in-production-cdba15b00e>
24. <https://www.datarobot.com/wiki/machine-learning-model-deployment/>
25. <https://www.analyticsvidhya.com/blog/2022/06/one-class-classification-using-support-vector-machines/>
26. <http://scikit-learn.org/stable/auto_examples/svm/plot_oneclass.html#:~:text=One%2Dclass%20SVM%20is%20an,different%20to%20the%20training%20set.>
27. <https://medium.com/grabngoinfo/one-class-svm-for-anomaly-detection-6c97fdd6d8af>
28. <https://medium.com/mlearning-ai/machine-learning-models-monitoring-made-easy-with-mlfow-a-concrete-use-case-with-python-api-334691936a7a>
29. <https://whylabs.ai/blog/posts/on-model-lifecycle-and-monitoring>
30. <https://towardsdatascience.com/experiment-tracking-with-mlflow-in-10-minutes-f7c2128b8f2c>
31. [https://www.datarobot.com/wiki/machine-learning-model-deployment/#:~:text=Why%20is%20 Model%20 Deployment%20Important,the%20model%20is%20severely%20limited.](https://www.datarobot.com/wiki/machine-learning-model-deployment/#:~:text=Why%20is%20Model%20Deployment%20Important,the%20model%20is%20severely%20limited.)
32. <https://www.activestate.com/resources/quick-reads/what-is-tkinter-used-for-and-how-to-install-it/>
33. <https://www.jeremyjordan.me/hyperparameter-tuning/>
34. <https://www.analyticsvidhya.com/blog/2021/04/evaluating-machine-learning-models-hyperparameter-tuning/>
35. <https://neptune.ai/blog/hyperparameter-tuning-in-python-complete-guide>
36. <https://towardsdatascience.com/a-practical-guide-to-implementing-a-random-forest-classifier-in-python-979988d8a263>
37. <https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/>