AIT511-MT Course Project-1 Report

Saatvik Sinha (MT2025722) and Affan Shaikh (MT2025016)

October 2025

0.1 Introduction

The experiments were performed in google colab.

The project code ipynb notebooks repository link: `https://github.com/Saatvik-Sinha/AIT-511-MT-Project1`

4 notebooks have been created for experiments conducted for the project.

The "OG" notebook is the file where the entire python library installations, dataset fetching, exploratory data analysis, preprocessing the training and testing datasets without making any assumptions or modifications on them, multiple models setup and grid-search-based training and submission file generation was done for the first time. It found that xgboost for specific hyperparameter values returned the best performance.

In the "Concat" notebook, we merged the training dataset with the original dataset it was sampled from and then dropped the duplicates, and performed grid-search-based training on xgboost again to check if performance improves- and it did, implying increase of generalization.

In the "RobustPCA" notebook, we applied principal component analysis which is robust to outliers on the continuous features of the concatenated dataset and selected the highest variance containing principal components to replace the continuous features with. We execute multiple models again on this modified dataset, and discover that

- Performance significantly dropped for tree-based classifiers
- Performance negligibly improved for naive-bayes classifier
- Performance significantly improved for knn & balltree classifiers

However none of them could achieve a performance comparable to xgboost of "OG" and "Concat" notebooks.

In the "Oversampling" notebook, we applied Synthetic Minority Oversampling Technique on concatenated training dataset to generate more samples for minority classes so that they have equal number of points with the majority class. We train xgboost again to compare the performance change. While it improved on training set, it dropped back on the test set, implying increase of overfitting.

0.2 Setup

1. Install necessary libraries using pip.
2. Upload kaggle.json file generated from your kaggle account.
3. Import and unzip dataset files from kaggle directly using the json file and kaggle library commands.
4. Import all necessary libraries for experimentation.

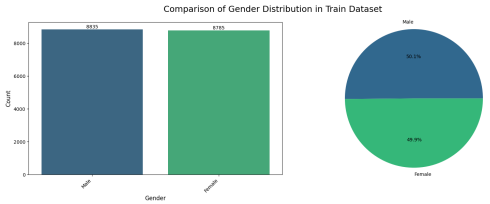
5. Read the original, train and test csv files as dataframes using pandas

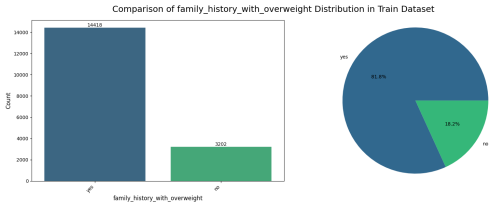
0.3 Exploratory Data Analysis

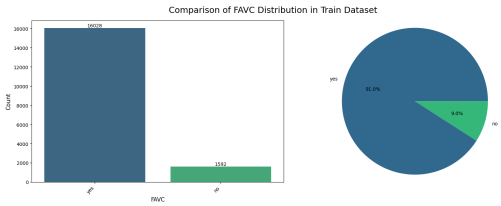
The graphs are plotted for the concatenated training dataset.

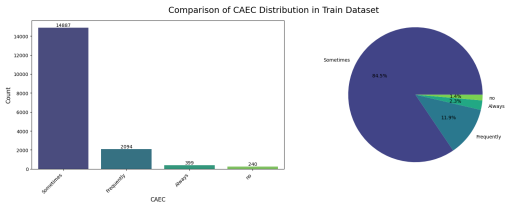
0.3.1 Univariate Analysis

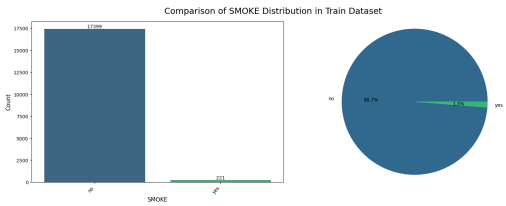
1. There were no null/missing values in the csv files, whether original, train, or test.
2. There were 8 object/categorical features and 8 continuous features.
3. All categorical columns and frequencies of their categories were observed as bar graph and pie chart

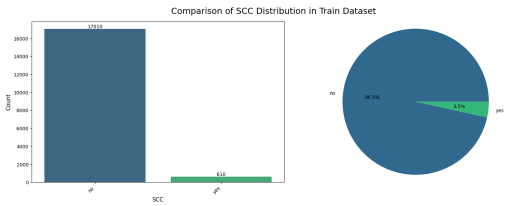


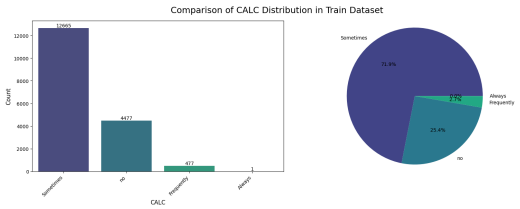


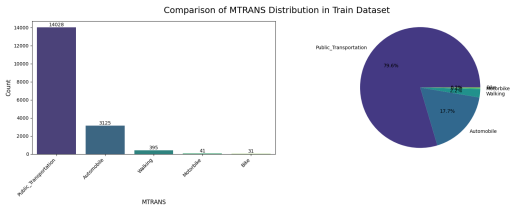












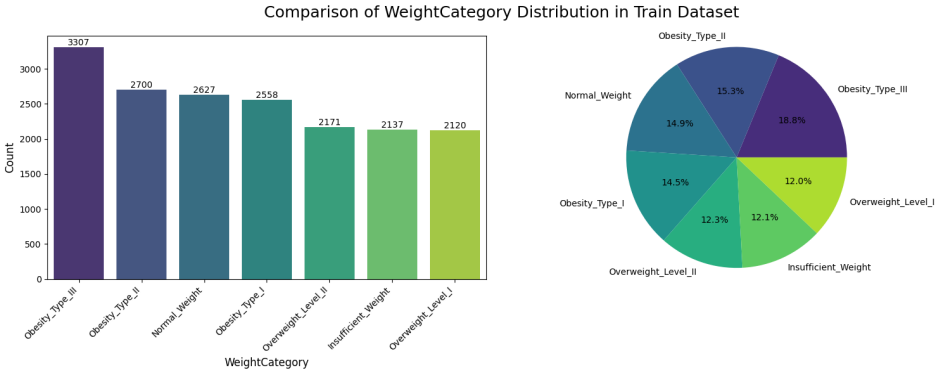
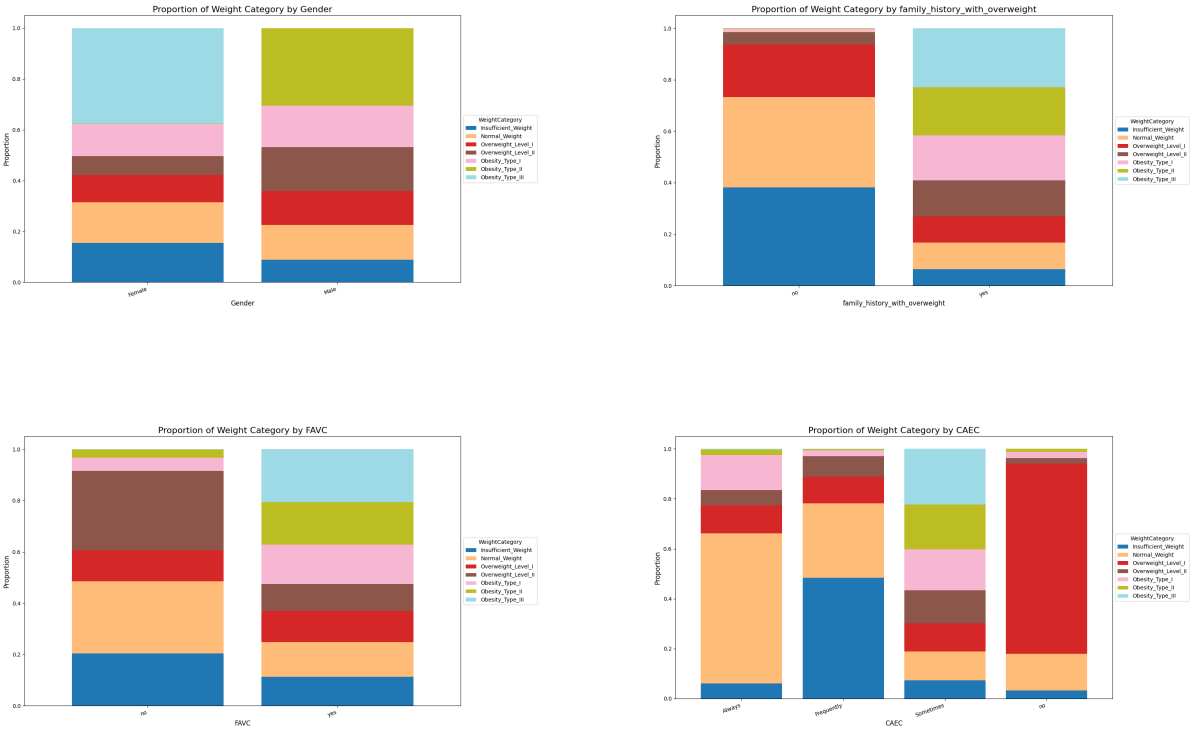


Figure 1: Distribution of categories of categorical columns

- Except Gender and WeightCategory, all columns quite clearly have massive class imbalance

4. Distribution of target categories among categorical features were observed as stacked bar charts to assess their proportion.



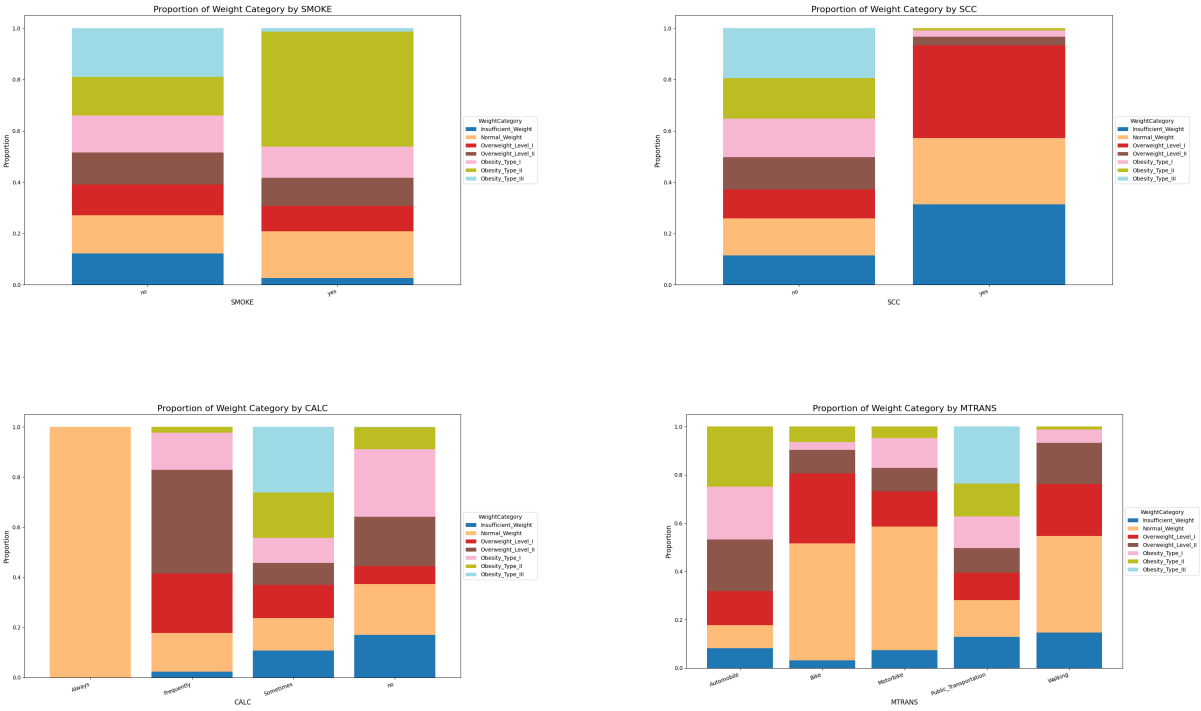
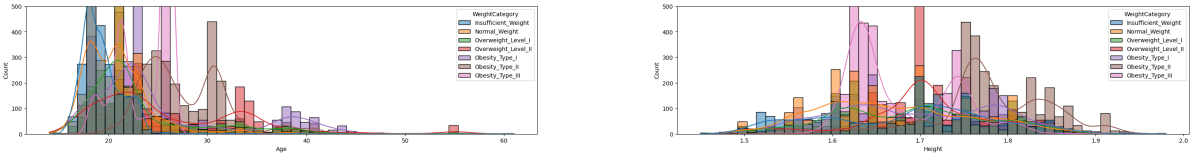


Figure 2: Distribution of target categories among categorical features

- People with Obesity Type II are all male while people with Obesity Type III are all female.
- People who are Overweight Level II and above are very likely to
  - have family history of overweight problems.
  - have no calories consumption monitoring (SCC).
  - use automobile or public transportation.
- People who indulge in Frequent Consumption of High Caloric Food (FAVC) are much more likely to be under the Obese group.

5. Distribution of target categories among continuous features were plotted as histograms and curves to assess likelihood of target value in a given range of continuous values.



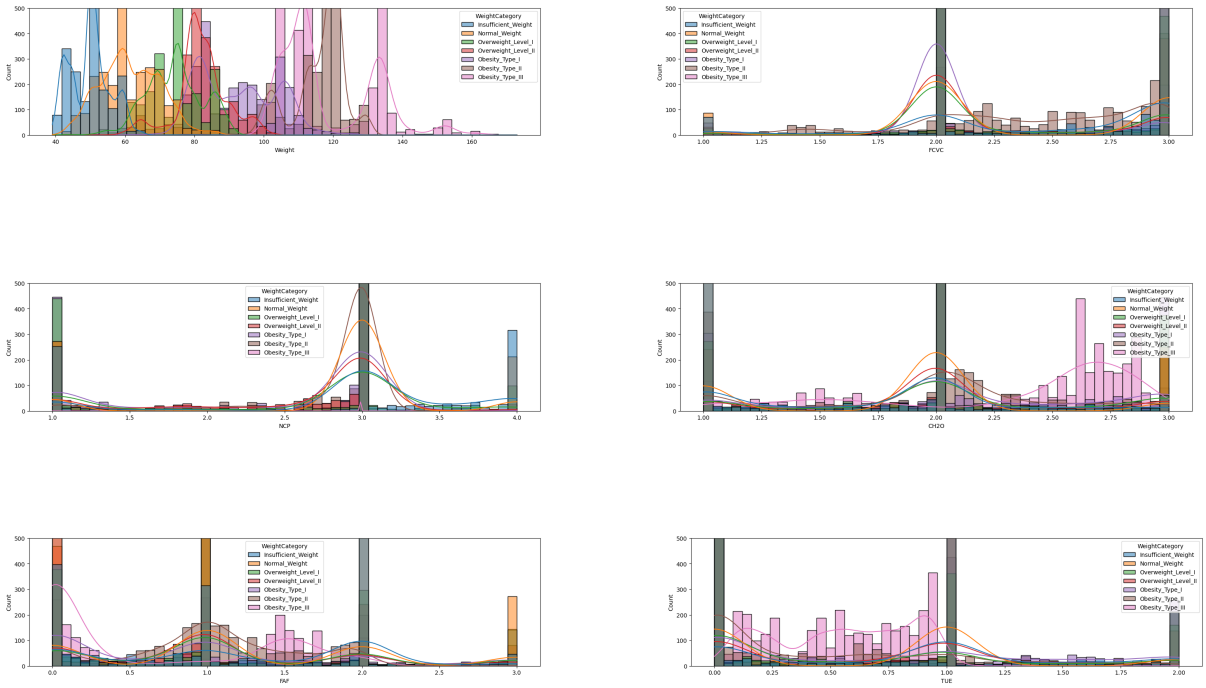


Figure 3: Distribution of target categories among continuous features

- For the Weight feature, as it increases, so does the prevalence of higher levels of WeightCategory, which is a sensible trend. However in many places, multiple WeightCategory types exists for the same range of Weight, which implies that there might exist lots of mislabeled/outlier/noisy points.
- For "TUE" which measures time spent using technological devices, people for Obesity-Type-III which is the highest WeightCategory are heavily concentrated in the left half of the x-axis.
- On the other hand, for "CH2O" which measures the consumption of water daily, the Obesity-Type-III people are mostly present in the right half of the x-axis.
- The other features do not show any particular pattern.

0.3.2 Bivariate Analysis

1. Correlation Analysis

   All the below insights match with the Univariate analysis done above.

- Gender is highly correlated with Weight, Height and WeightCategory
- Weight is highly correlated to family-history-of-overweight.
- family-history-of-overweight is highly correlated to WeightCategory.

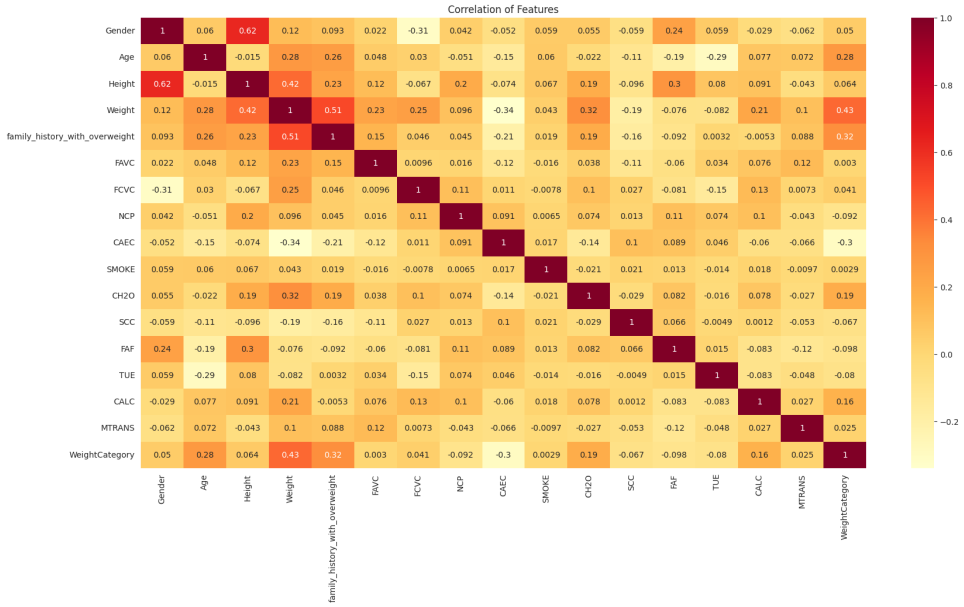
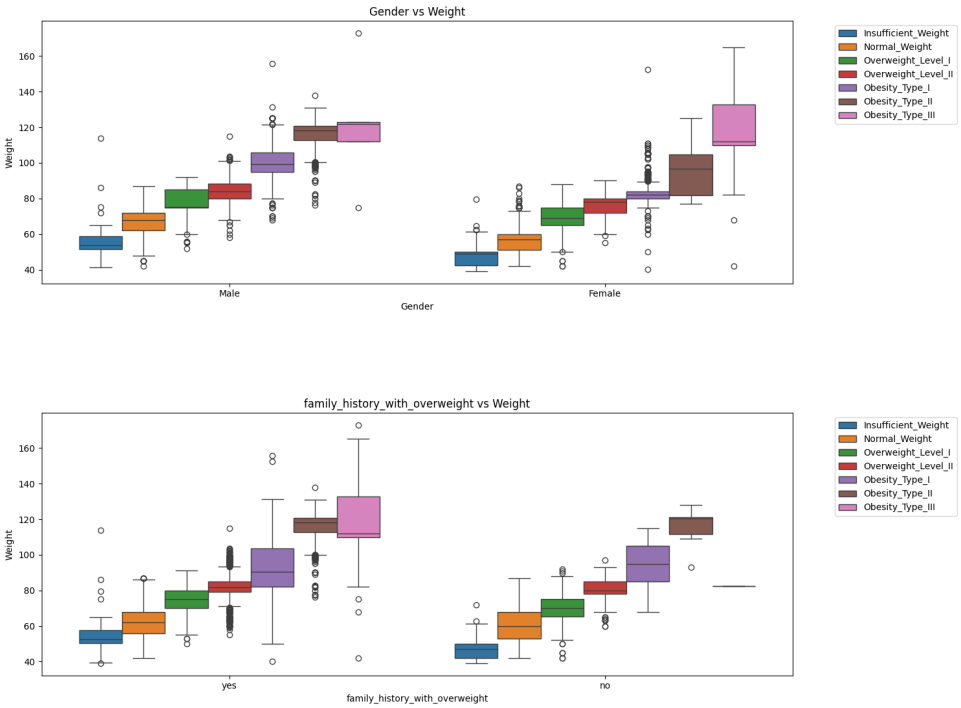


Figure 4: Correlation Matrix

2. Categorical feature vs Categorical feature Analysis
   The proportion of categories of one categorical feature were checked compared to all other categorical features, separated by target categories, which generated 28 stacked bar plots. The plots only showed the clear class imbalance within features and no new insights.

3. Categorical feature vs Continuous feature Analysis



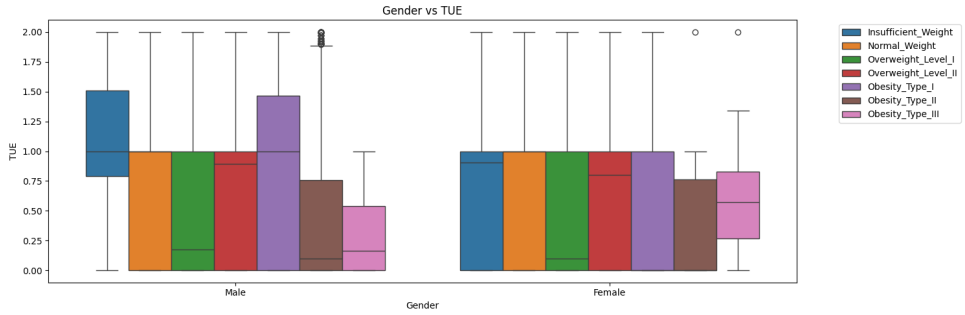


Figure 5: Selected boxplots that display potential outliers

The distribution of continuous values with respect to categorical features, for each target category, was observed using boxplots. 28 box plots were generated out of which only few distributions were helpful.

- Matching with the univariate analysis done above, the features Weight and TUE seem to be the primary cause for potential outliers.

4. Continuous feature vs Continuous feature Analysis

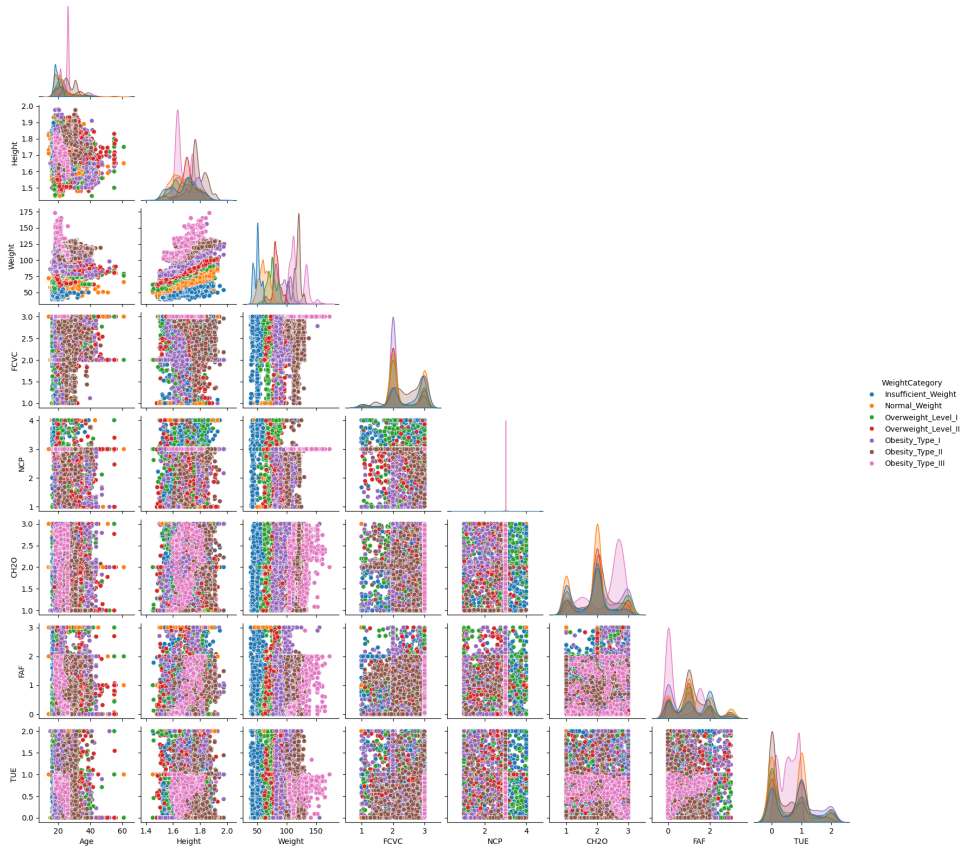


Figure 6: Pairplots according to original features

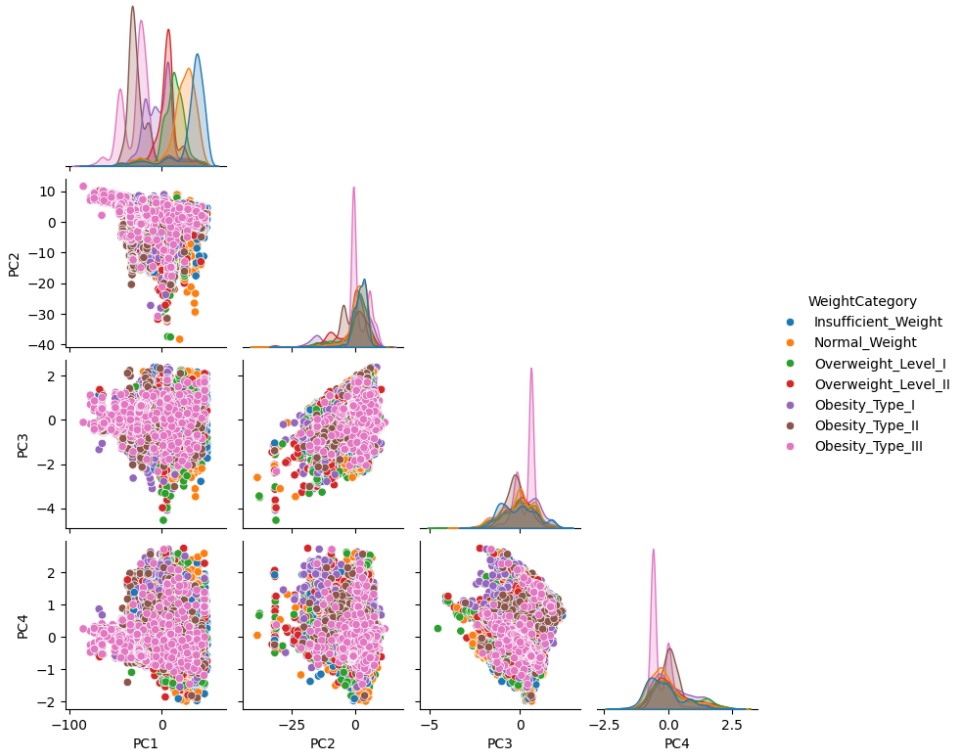


Figure 7: Pairplots according to top principal components

- In the scatter plots based on original features, the same insights from Figure 3 are applicable.
- The principal component analysis seems to not have helped with making the data more separable, rather it reduced it further compared to the original features. It also made the distributions dense by concentrating the points in a lesser area, indicating the effect of the robust component calculation.

0.4 Data processing steps

(a) In notebooks other than "OG", training dataset was supplemented with original dataset and then the duplicates were dropped.

(b) Applied label encoding on categorical variables with 2 classes which converted their values to 0s and 1s.

(c) Categorical variables with 4 classes had the classes [no, sometimes, frequently, always]. They clearly represent increasing order of frequency of the classes, therefore ordinal encoding was applied, with the mapping {no:0, sometimes:1, frequently:2, always:3}.

(d) The MTRANS column consisted of 5 modes of transportation. According to EDA, [Walking, Bike, Motorbike] and [Automobile, Public_Transportation]

respectively had similar distribution among the output categories. Therefore we bin these 2 categories together where the values from former list were labeled as 0 and later were labeled as 1. Using encoded MTRANS column caused better accuracy in XGboost than dropping the column or using one-hot encoding.

(e) Ordinal encoding and binary label encoding applied above are much superior to the alternative of one hot encoding because One-hot encoding is generally not suitable for decision trees and tree-based ensembles like random forests due to several key issues. First, it creates sparsity in the dataset, as each categorical level is represented by a binary column, leading to mostly zero values for most observations. This sparsity makes it difficult for the tree to find meaningful splits, as the gain in purity from splitting on any single dummy variable is typically marginal, causing the algorithm to often ignore the categorical feature in favor of continuous variables. Second, one-hot encoding forces decision trees to use multiple splits to handle a single categorical feature. For a categorical variable with n levels, the tree may require up to n splits to separate the data, which consumes significant tree depth and can lead to shallow trees that fail to capture the full predictive power of the feature. This is particularly problematic because tree depth grows exponentially with the number of splits, and many trees in ensembles are relatively shallow. Third, the resulting feature importance becomes fragmented and difficult to interpret. Instead of a single importance score for the original categorical feature, the model produces separate importance scores for each dummy variable (e.g., "country=USA", "country=UK"), making it challenging to assess the overall contribution of the categorical variable. This fragmentation can also lead to incorrect feature selection, where individual dummy variables are dropped based on low importance, even if the original feature is informative. Theoretically decision trees are capable of handling categorical variables as it is, but the scikit-learn implementations of decision trees cannot. Therefore ordinal and binary label encoding are superior.

(f) MinMaxScaler transforms each feature to a fixed range, typically [0, 1], by subtracting the minimum value and dividing by the range (max - min) of the feature. This results in a normalized dataset where the smallest value becomes 0 and the largest becomes 1, preserving the original shape of the distribution. It is particularly useful when the data has a bounded range, such as image pixel intensities (0–255), or when the algorithm expects inputs within a specific interval, like neural networks. However, MinMaxScaler can be sensitive to outliers, as they compress the inlier values into a narrow range. StandardScaler, on the other hand, standardizes features by removing the mean and scaling to unit variance, resulting in a distribution with a mean of 0 and a

standard deviation of 1. This is achieved by subtracting the mean and dividing by the standard deviation of each feature. It is ideal when the data follows a Gaussian (normal) distribution or when the algorithm assumes normally distributed inputs, such as logistic regression, support vector machines, or principal component analysis (PCA). StandardScaler is less sensitive to outliers compared to MinMaxScaler in some contexts, although it can still be influenced by extreme values. In practice, the choice between MinMaxScaler and StandardScaler often depends on the data distribution and the machine learning algorithm used. If the data is normally distributed, StandardScaler is typically preferred. If the data has a bounded range or is non-Gaussian, MinMaxScaler may be more appropriate. For many cases, especially with algorithms sensitive to feature scale, StandardScaler is a safe default choice. Therefore Standard scaling was performed on continuous features. In the "RobustPCA" section, only mean-centering was done because standard deviation calculation is not robust to outliers.

(g) RobustPCA was applied only on continuous features as PCA requires calculation of terms like mean & covariances which are meant to be obtained from continuous values.

(h) RobustPCA essentially works by calculating a robust to outliers estimator of covariance matrix called minimum covariance determinant. The remaining steps are same as normal PCA.

(i) In the "RobustPCA" notebook, the singular vectors for the continuous columns were calculated and then used to convert the continuous columns into principal components. The top 4 principal components were selected out of 8, and added to the feature set while the original continuous features were removed.

(j) Label encoding was applied on WeightCategory consisting of 7 classes.

(k) In the "Oversampling" notebook, SMOTE was used to generate synthetic samples to solve class imbalance and those were added to concatenated dataset. The k_neighbors hyperparameter was experimented with and best result was obtained from the default value of 5.

0.5 Models used

- KNN (K Neighbors Classifier)
- Balltree (Radius Neighbors Classifier)
- Naive Bayes (Gaussian model can only process continuous features and Categorical model can only process categorical features therefore Mixed Naive

Bayes model used which combines both types of models to process dataset with both continuous and categorical features)

- Decision Tree
- Balanced Random Forest (It differs from a classical random forest by the fact that it will draw a bootstrap sample from the minority class and sample with replacement the same number of samples from the majority class, making it robust to class imbalance)
- AdaBoost
- GradientBoost (Only in "OG" because too slow and did not show better performance than XGBoost)
- XGBoost

0.6 Hyperparameter tuning

Grid Search with Stratified KFold Cross-validation where K=5 was used to perform hyperparameter tuning for the models.

As grid search is a very slow algorithm because of it tests for every single hyperparameter combination, the initial hyperparameter value lists were created based on Rules of Thumb for different parameters in different models to create a smaller but optimal & reliable search space. The different hyperparameter lists for grid search were manipulated after every complete cell run to further narrow down the optimal values. Mean accuracy score on validation set during cross validation and overall accuracy score on total training set was used mainly to assess the performance. Confusion matrix was also plotted and precision-recall-f1score-support values for every output category printed.

Only XGBoost was trained and tuned in "Concat" and "Oversampling" notebooks because it had the best performance in "OG" notebook.

(a) No hyperparameter tuning for Naive Bayes model was done because it is not an iterative algorithm and works simply by calculating conditional probabilities from the entire training data at once.

(b) For all tree-based models, random seed state was fixed at 42 for reproducibility and minimum number of samples that should be present in a leaf was fixed at 2 to prevent overfitting. The extremely small number of 2 was chosen because many categorical features had extreme class imbalance within themselves and trees might need to branch to classify based on the rare feature values.

(c) In Random Forest, variance reduces as number of trees increase but bias saturates early. It has been observed that for large sample size, less than 20 trees

gives unstable results while more than 500 trees gives diminishing returns. The rule of thumb is that 50-200 trees is the sweet spot for training random forest. Therefore we checked n_estimator parameter only for the values 100 and 200. As Adaboost uses trees with depth of 1 only, it will certainly need much higher number of trees for good performance, hence we test n_estimator upto the value of 500.

(d) For singular decision tree algorithm, we tested max_depth among the values [5, 8, 10, 12, 15, 17, 20, 22, 25]. They were selected as every consecutive pair is equidistant from each other and cover both shallow and very deep variants to find the model that achieves the best balance between underfitting and overfitting.

(e) Random forest is a bagging algorithm hence it makes use of deep decision trees for inference. As there are 16 features in the dataset, we check max_depth parameter for 8, 10 and 12 as too less depth will result in underfitting while too deep trees cause overfitting. Adaboost uses decision stumps- that is trees with fixed max_depth of 1. Gradient Boosting and its extreme variant theoretically rely on weak learners to build a very accurate model while avoiding overfitting. Therefore we only check the low max_depth values of 4, 6 and 8 for them.

(f) Boosting models utilize a learning rate in their iterations. Hyperparameters such as learning rate and regularization term tend to be very small positive numbers. When we sample them, we would like to sample values from all the orders of magnitude in a given interval. Let's say we decide to sample uniformly from 0 to 1, then only about 10% of the values would come from 0 to 0.1, and 90% of the values would come from 0.1 to 1. Instead, if we used a logarithmic scale to sample the values such as from -5 to 0, then values from 1e4, 1e3, 1e2, 1e1, 1e0 all have equal chance of being selected. A general rule of thumb derived from this is to sample values that are derived by multiplying 3 constantly to the initial smallest value which is in the form of 1e-x where x is a natural number, such as [0.01, 0.03, 0.1, 0.3] where x=2.

(g) Balltree and KNN classifiers rely on distance metric for classification. They can either choose to treat every 'nearest' neighbor with same importance ("uniform") or give higher weight to points closer to the target point (weight is inversely proportional to "distance"). This was checked using the hyperparameter "weight".

(h) Balltree classifier had its lower limit of radius increased till it stopped return 'NaN' for validation accuracy (radius=3 for 'OG' and radius=8 for 'Robust-PCA'). It was doing so because the radius was too small to include any other points for the classification. The performance then starts saturating as we increase radius by one point constantly.

(i) In KNN classifiers, K should be less than sample size of target class of smallest count to avoid misclassification due to class imbalance. Aside from this, a common rule of thumb is K to be square root of total sample size. Also, K should be odd because in the case K is an even number, there is a risk that the nearest neighbors could be evenly split between two or more classes, leading to an ambiguous classification decision. The square root of sample size was 124+, and it is also less than the sample size of target class of smallest count. Therefore the grid search for K was performed on the list of [31, 63, 125, 189, 251], all of which are either approximate factors or multiples of $125/2 = 63$.

(j) SMOTE does not follow the square root of sample size thumb rule for K despite using KNN because a large k would lose local region information, while a small k would be overly influenced by outliers. A small value (e.g., 5-10) is typical. Therefore values [3, 5, 7, 9, 11] were observed but all of them achieved nearly the same metric numbers.

0.7 Comparative Analysis

Table 1: Model Performance (OG Notebook)

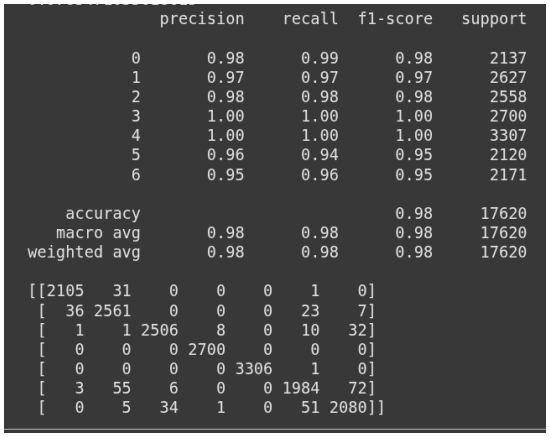
Model	Mean CV Score	Optimal Hyperparameters	Training Accuracy
XGBoost	0.9032	{learning_rate: 0.3, max_depth: 4, n_estimators: 100}	0.9513
AdaBoost	0.7218	{learning_rate: 0.3, n_estimators: 400}	0.7731
Random Forest	0.8942	{max_depth: 12, n_estimators: 100}	0.9495
Decision Tree	0.8696	{max_depth: 10}	0.9106
Naive Bayes	–	–	0.7234
Ball Tree	0.5410	{radius: 3.5, weights: distance}	1.0000
KNN	0.7531	{n_neighbors: 31, weights: distance}	1.0000

Table 2: Model Performance (RobustPCA Notebook)

Model	Mean CV Score	Optimal Hyperparameters	Training Accuracy
XGBoost	0.7541	{learning_rate: 0.07, max_depth: 6, n_estimators: 200}	0.8297
AdaBoost	0.6524	{learning_rate: 0.7, n_estimators: 100}	0.6520
Random Forest	0.8486	{max_depth: 12, n_estimators: 100}	0.9157
Decision Tree	0.8315	{max_depth: 10}	0.8871
Naive Bayes	–	–	0.7310
Ball Tree	0.7150	{radius: 9, weights: distance}	1.0000
KNN	0.8293	{n_neighbors: 31, weights: distance}	1.0000

Table 3: Comparison of XGBoost Model Performance: Concat vs. Oversampling

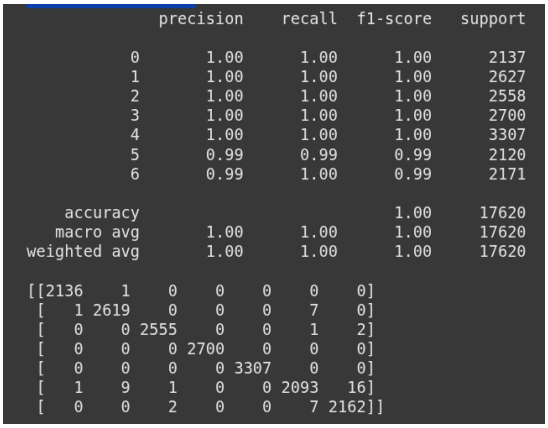
Notebook	Mean CV Score	Optimal Hyperparameters	Training Accuracy	Test Accuracy (%)
Concat	0.9110	{learning_rate: 0.1157, max_depth: 6, n_estimators: 200}	0.9785	91.129
Oversampling	0.9234	{learning_rate: 0.3, max_depth: 6, n_estimators: 200}	0.9973	90.661



	precision	recall	f1-score	support
0	0.98	0.99	0.98	2137
1	0.97	0.97	0.97	2627
2	0.98	0.98	0.98	2558
3	1.00	1.00	1.00	2700
4	1.00	1.00	1.00	3307
5	0.96	0.94	0.95	2120
6	0.95	0.96	0.95	2171
accuracy			0.98	17620
macro avg	0.98	0.98	0.98	17620
weighted avg	0.98	0.98	0.98	17620

```
[[2105   31    0    0    0    1    0]
 [  36 2561    0    0    0   23    7]
 [   1    1 2506    8    0   10   32]
 [   0    0    0 2700    0    0    0]
 [   0    0    0    0 3306    1    0]
 [   3   55    6    0    0 1984   72]
 [   0    5   34    1    0   51 2080]]
```

(a) Concat



	precision	recall	f1-score	support
0	1.00	1.00	1.00	2137
1	1.00	1.00	1.00	2627
2	1.00	1.00	1.00	2558
3	1.00	1.00	1.00	2700
4	1.00	1.00	1.00	3307
5	0.99	0.99	0.99	2120
6	0.99	1.00	0.99	2171
accuracy			1.00	17620
macro avg	1.00	1.00	1.00	17620
weighted avg	1.00	1.00	1.00	17620

```
[[2136    1    0    0    0    0    0]
 [   1 2619    0    0    0    7    0]
 [   0    0 2555    0    0    1    2]
 [   0    0    0 2700    0    0    0]
 [   0    0    0    0 3307    0    0]
 [   1    9    1    0    0 2093   16]
 [   0    0    2    0    0    7 2162]]
```

(b) Oversampling

Figure 8: Comparison of Concat & Oversampling XGBoost metric values

XGBoost with optimal hyperparameters best performing model in OG notebook. XGBoost with optimal hyperparameters in Concat is the best performing overall model with learning_rate=0.1157, max_depth=6, n_estimators=200 achieving 91.1% cross validation score, 97.8547% overall accuracy on training set, with 91.661% accuracy on test set in kaggle leaderboard.

XGBoost improved performance on both train & test csv files after appending original data to training set. However while training accuracy was even higher, on test set the model performed worse when samples generated from SMOTE were added. This could imply that when amount of real-world/good-quality data is increased, or as training data is supplemented with the data it was sampled from, machine learning models tend to improve while synthetic data can cause changes in either direction.

KNN & Balltree classifiers that were 'weighted by distance' had better cross validation score than their 'uniform' counterparts and also had a 100% accuracy with perfect confusion matrices on training set overall. But the cross-validation score was still below or comparable to tree-based classifiers. This quite clearly indicates that they were easily overfitting.

The application of Robust Principal Component Analysis (Robust PCA) had distinct effects on different model families. Robust PCA, implemented through robust covariance estimation, identifies and suppresses the influence of outliers while projecting the data into a lower-dimensional subspace that preserves the principal variance directions. Consequently, it yields a feature space that is denoised, outlier-resistant, and linearly decorrelated. The following subsections discuss the impact

of this transformation on the model performances observed.

0.7.1 Improvement in KNN and Radius Neighbors Classifiers

The performance of distance-based models such as *K-Nearest Neighbors (KNN)* and *Radius Neighbors* improved significantly after applying Robust PCA. These algorithms rely heavily on Euclidean distances to determine neighborhood relationships. Prior to transformation, outliers and correlated features distorted distance computations, leading to unstable neighborhood structures. Robust PCA mitigated these effects by removing outlier influence, reducing feature correlation, and alleviating the curse of dimensionality. As a result, pairwise distances became more representative of true similarity among samples, allowing the classifiers to form more coherent and discriminative decision boundaries. Therefore, the improved performance can be attributed to the creation of a cleaner, low-dimensional, and geometrically meaningful feature space. However they could not achieve a cross-validation score comparable to xgboost of "OG" and "Concat" notebooks.

0.7.2 Performance Drop in Tree-Based Models

In contrast, tree-based models such as *Decision Trees* and *Random Forests* exhibited a significant drop in performance after applying Robust PCA. These models construct axis-aligned splits on original feature dimensions and naturally handle outliers and nonlinearity. The PCA transformation, however, produces linear combinations of the original variables, which removes axis alignment and reduces feature interpretability. Additionally, the smoothing effect of Robust PCA compresses nonlinear feature interactions that tree-based models typically exploit. As a result, the transformed feature space becomes less compatible with the decision tree mechanism, leading to decreased classification accuracy. Hence, while Robust PCA benefits distance-based models, it disrupts the feature structure that tree-based algorithms rely upon.

0.7.3 Minimal Change in Naive Bayes Performance

The *Naive Bayes* classifier showed negligible change in performance following Robust PCA. This algorithm models class-conditional probabilities under the assumption of feature independence. Since PCA yields uncorrelated (orthogonal) components, it is inherently compatible with this assumption. Moreover, Naive Bayes is relatively insensitive to feature scaling and outlier effects, as it depends on global

distribution estimates rather than local geometric relationships. Therefore, while Robust PCA slightly decorrelates features and removes outlier influence, these modifications do not substantially alter the probabilistic structure used by Naive Bayes, resulting in minimal performance variation.

0.8 Future Work

(a) Target classes did not have heavy difference in their frequencies and despite that they could be handled using Stratified Kfold training and oversampling. However, categorical features had class imbalance among their categories. Oversampling with respect to categories of categorical features rather than target categories can be explored as a way to improve performance.

(b) Identifying and removing points that are highly likely to be outliers/noise and then training and testing the models can be explored.

(c) Advanced hyperparameter tuning techniques like those involving evolutionary algorithms or bayesian optimization can be utilized for better selection.

(d) Performance of advanced machine learning models like deep neural networks and support vector machines using kernels can be checked.