ReneWind Project

Model Tuning Problem

Yair Brama – November 2024

Contents / Agenda

Executive Summary - Business Context

"ReneWind" is working on improving the machinery/processes involved in the production of wind energy using machine learning. It collected data of generator failure of wind turbines using sensors. The shared data we have, has 40 predictors, 20000 observations in the training set and 5000 in the test set.

This objective of this project is to build various classification models, tune them, and find the best one that will help identify failures so that the generators could be repaired before failing/breaking to reduce the overall maintenance cost. The nature of predictions made by the classification model will translate as follows:

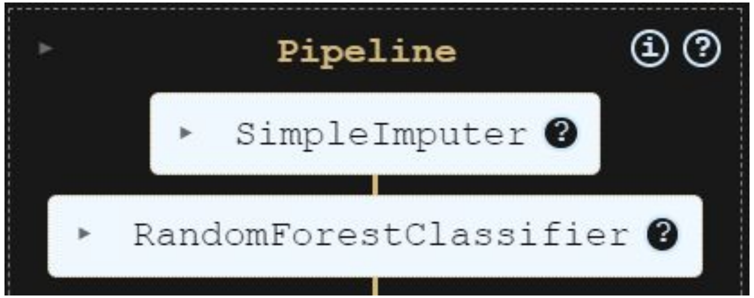
True positives (TP) are failures correctly predicted by the model. These will result in repairing costs. The **accuracy** score will reflect how good the model in predicting those cases.

False negatives (FN) are real failures where there is no detection by the model. These will result in replacement costs. The **recall** score will reflect how good the model is in minimizing those cases.

False positives (FP) are detections where there is no failure. These will result in inspection costs. It is given that the cost of repairing a generator is much less than the cost of replacing it, and the cost of inspection is less than the cost of repair. The **precision** score will evaluate these cases.

Executive Summary – Final Recommendations

Based on the results of our models, we recommend to use the **Tuned Random Forest Classifier** model, trained on **under sampled** data. The pipeline is constructed with the following steps:



Step 1 - Imputer – Makes sure the missing data is filled based on the median of the column

Step 2 - model – Fandom Forest with the following parameters: n_estimators': 300, 'min_samples_leaf': 2, 'max_samples': 0.5, 'max_features': 'sqrt'

The score we got for Recall when running the test set is: **0.872.**
That means that only 12.8% of the actual failure cases were misclassified by the model.

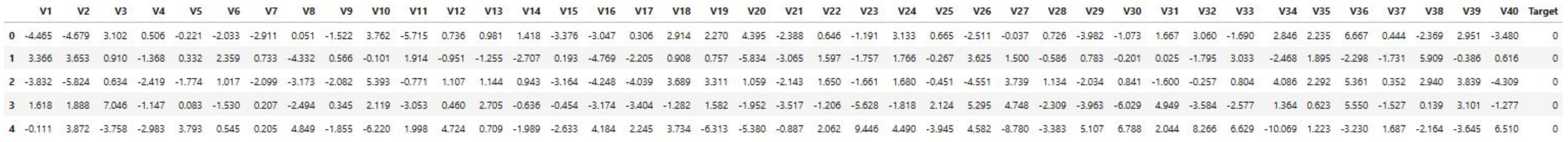
Rationale – This model gave us the highest recall score, comparing to the results of other models (as we ran them on the validation set). We are focusing on Recall, since minimizing the False Negative predictions is our main goal for the ML.

Solution Approach – All models

Data initial review	EDA	Data Preprocessing	Model Building	Run cross validation for 6 model with default params	Run models after tuning	Create Pipeline
Check for null values Check for duplicates Check data types	Univariate Analysis – Box plots, Histograms	Check outliers Impute missing values	Split the dataset between training and validation data	Run the models: Logistic regression, bagging, random forest, GBM, Adaboost, XGBoost with default parameters Run it with oversampling data Run it with under sampled data	Perform randomized search to tune and find the optimal values of the hyperparameters Check scores Compare the scores	Compare the results of all iterations and recommend the best option Create a pipeline Run the pipeline on the test set

Data Description

The data set consists of 40 predictor variables and 1 target variable ("1" in the target variables should be considered as "failure" and "0" represents "No failure".)



	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20
0	-4.465	-4.679	3.102	0.506	-0.221	-2.033	-2.911	0.051	-1.522	3.762	-5.715	0.736	0.981	1.418	-3.376	-3.047	0.306	2.914	2.270	4.395
1	3.366	3.653	0.910	-1.368	0.332	2.359	0.733	-4.332	0.566	-0.101	1.914	-0.951	-1.255	-2.707	0.193	-4.769	-2.205	0.908	0.757	-5.834
2	-3.832	-5.824	0.634	-2.419	-1.774	1.017	-2.099	-3.173	-2.082	5.393	-0.771	1.107	1.144	0.943	-3.164	-4.248	-4.039	3.689	3.311	1.059
3	1.618	1.888	7.046	-1.147	0.083	-1.530	0.207	-2.494	0.345	2.119	-3.053	0.460	2.705	-0.636	-0.454	-3.174	-3.404	-1.282	1.582	-1.952
4	-0.111	3.872	-3.758	-2.983	3.793	0.545	0.205	4.849	-1.855	-6.220	1.998	4.724	0.709	-1.989	-2.633	4.184	2.245	3.734	-6.313	-5.380

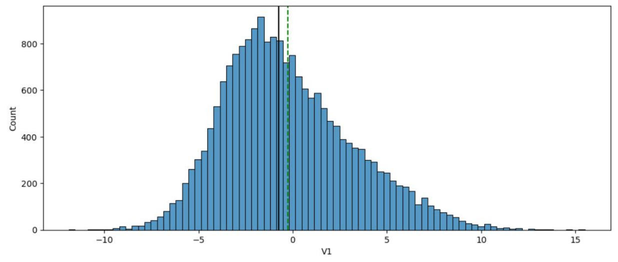
	V21	V22	V23	V24	V25	V26	V27	V28	V29	V30	V31	V32	V33	V34	V35	V36	V37	V38	V39	V40	Target
0	-2.388	0.646	-1.191	3.133	0.665	-2.511	-0.037	0.726	-3.982	-1.073	1.667	3.060	-1.690	2.846	2.235	6.667	0.444	-2.369	2.951	-3.480	0
1	-3.065	1.597	-1.757	1.766	-0.267	3.625	1.500	-0.586	0.783	-0.201	0.025	-1.795	3.033	-2.468	1.895	-2.298	-1.731	5.909	-0.386	0.616	0
2	-2.143	1.650	-1.661	1.680	-0.451	-4.551	3.739	1.134	-2.034	0.841	-1.600	-0.257	0.804	4.086	2.292	5.361	0.352	2.940	3.839	-4.309	0
3	-3.517	-1.206	-5.628	-1.818	2.124	5.295	4.748	-2.309	-3.963	-6.029	4.949	-3.584	-2.577	1.364	0.623	5.550	-1.527	0.139	3.101	-1.277	0
4	-0.887	2.062	9.446	4.490	-3.945	4.582	-8.780	-3.383	5.107	6.788	2.044	8.266	6.629	-10.069	1.223	-3.230	1.687	-2.164	-3.645	6.510	0

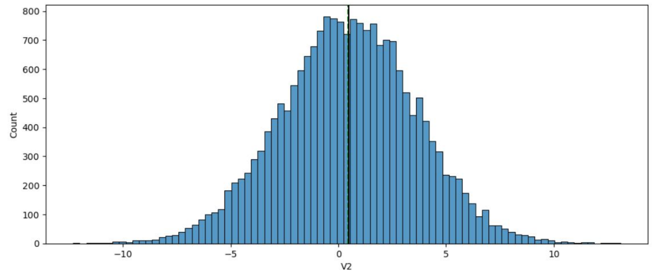
The training data includes 20000 rows, and the test data has 5000 rows. There are few missing values (will be described in the next slides) and no duplicates. 1110 of the records are "failure", and 18890 are "no failure".

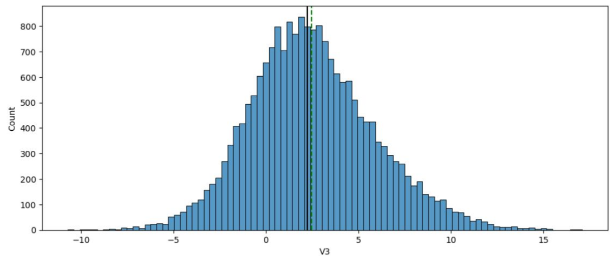
All the fields are 'float64', which means we don't need to transform any of them as part of the model preparation.

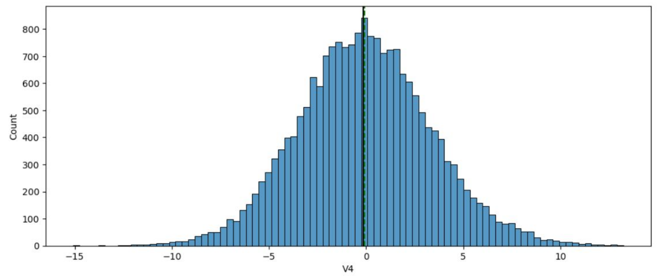
EDA Results

Observation – All the 40 variables have normal distribution, some of them are slightly skewed to the right or to the left, but none has a serious abnormality that requires normalization. All outliers will remain as is at this point.









The full display of all the attributes can be found in the appendix.

Data Pre-processing

- **Missing Values Imputation**– In the training set we found 2 columns with 18 missing values each. We used the ***SimpleImputer***(strategy="**median**") to fill these values with the median values of the columns.

- **Data Set Split** – The training set is split to 15000 records in the training set, and 5000 records as a validation set (75:25 split). This will allow us to create and compare multiple models and run the best model on the ‘clean’ test set, without data leakage.

Models Performance – Cross Validation with Original Set

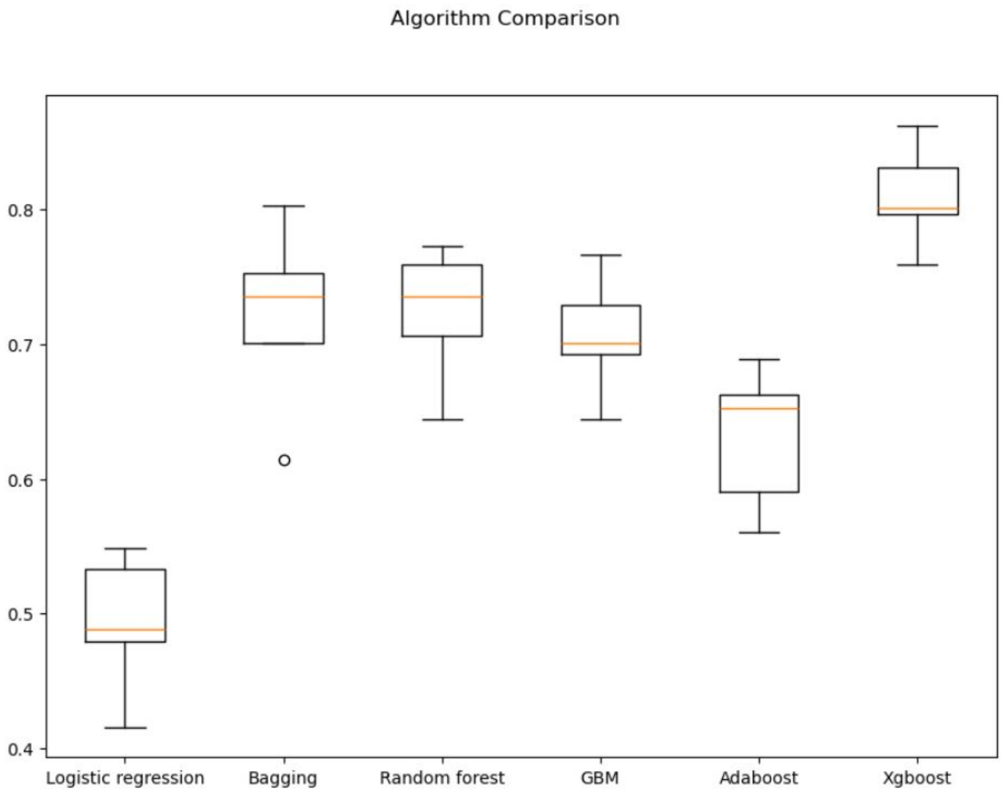
Models Performance Summary – Cross Validation

(n_splits = 5, random_state=1, scorer = recall)

Model Name	Training score	Validation score
Logistic Regression	0.49	0.49
Bagging	0.72	0.73
Random Forest	0.72	0.73
GBM	0.71	0.72
Adaboost	0.63	0.67
XGBoost	0.81	0.83

Results – Without tuning or balancing the target variable, XGBoost model stands out with 83% success in recall (average score of the 4 runs)

Models Performance Summary – Cross Validation



Results – When comparing the results, we can see that logistic regression did poorly compare to the others, and XGBoost is the only one in the range of 0.8

Models Performance – Cross Validation with oversampling

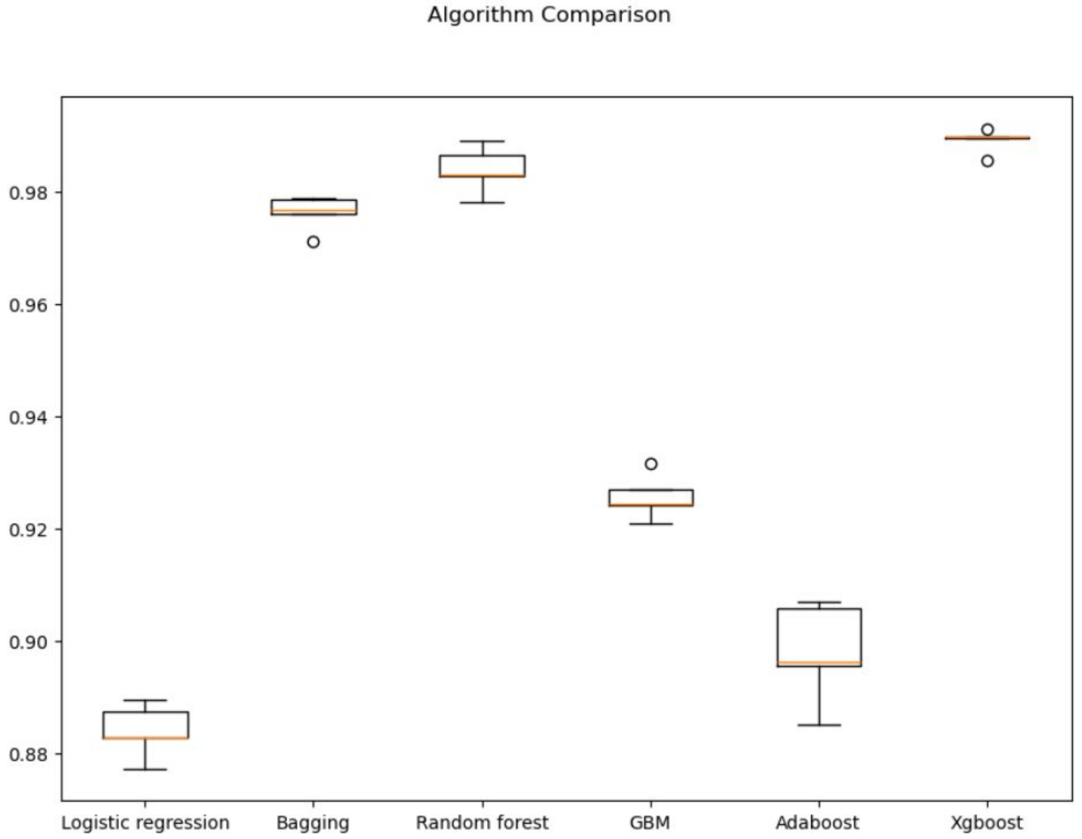
Models Performance Summary – Cross Validation with oversampling

(n_splits = 5, random_state=1, scorer = recall)

Model Name	Training score	Validation score
Logistic Regression	0.88	0.85
Bagging	0.98	0.83
Random Forest	0.98	0.85
GBM	0.93	0.88
Adaboost	0.90	0.86
XGBoost	0.99	0.87

Results – With oversampling, we added synthetic data to balance the target values, and there are 14168 records with each value (1 and 0). When running the same models 4 times, without tuning, we see that GBM model perform better, with a slight overfitting

Models Performance Summary – Cross Validation with oversampling



Results – Although XGBoost performs better in training, it shows a strong overfit (12 points between train and validation), therefore GB Model is the winner of this attempt.

Models Performance – Cross Validation with under sampling

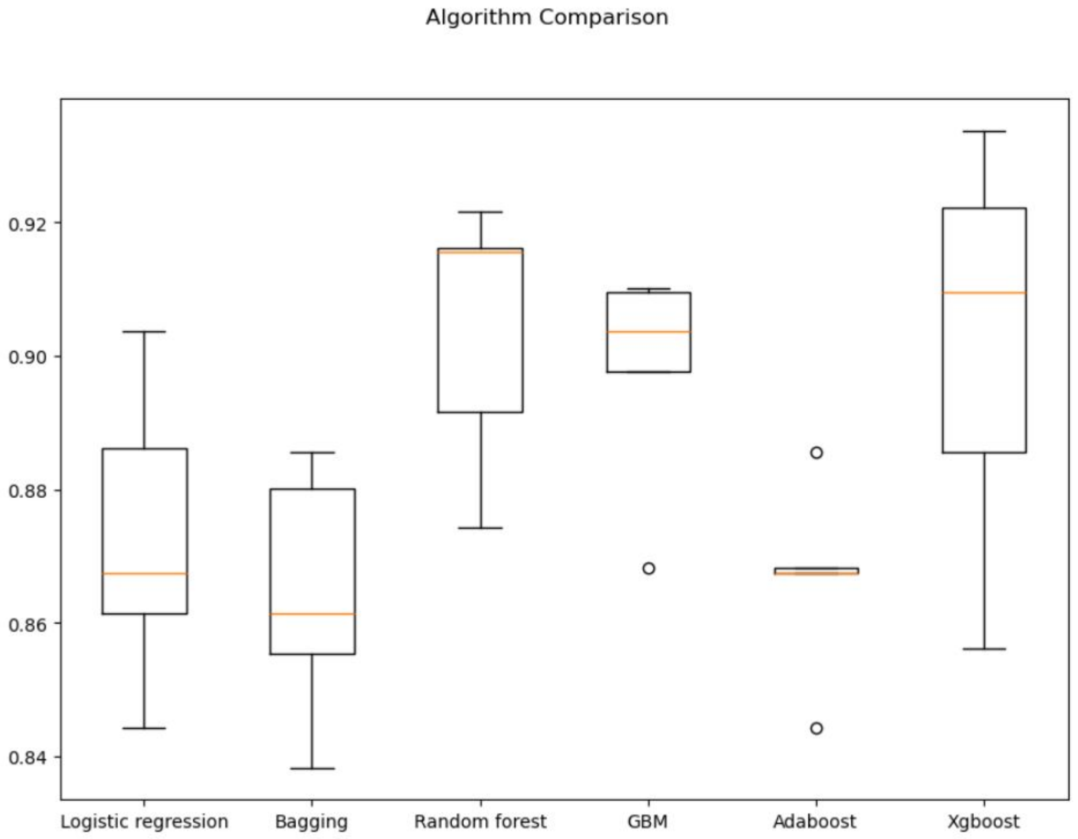
Models Performance Summary – Cross Validation with undersampling

(n_splits = 5, random_state=1, scorer = recall)

Model Name	Training score	Validation score
Logistic Regression	0.87	0.85
Bagging	0.86	0.87
Random Forest	0.90	0.89
GBM	0.90	0.89
Adaboost	0.87	0.85
XGBoost	0.90	0.90

Results – With under sampling, we removed some data to balance the target values, and there are 832 records with each value (1 and 0). When running the same models 4 times, without tuning, we see that XGBoost model perform better, with no overfitting

Models Performance Summary – Cross Validation with undersampling



Results – XGBoost performs better than the rest, with no overfitting.

* We see in the image on the left that the median of the Random Forest model is slightly better, but the mean of the 2 models is practically the same

Models Performance –With Hyperparameter Tuning

Adaboost Performance Summary – Tuned with Oversampled data

(n_estimators= 200, learning_rate= 0.2, estimator= DecisionTreeClassifier(max_depth=3, random_state=1)

Score	Training score	Validation score
Accuracy	0.992	0.979
Recall	0.988	0.853
Precision	0.995	0.79
F1	0.992	0.82

Results – This model shows overfitting, and the recall score in validation is just above 0.85

Random Forest Performance Summary – Tuned with Undersampled data

(n_estimators': 300, 'min_samples_leaf': 2, 'max_samples': 0.5, 'max_features': 'sqrt')

Score	Training score	Validation score
Accuracy	0.961	0.938
Recall	0.933	0.885
Precision	0.989	0.468
F1	0.96	0.612

Results – We see very good results in the training set, with overfitting in precision and F1 scores.

Gradient Boosting Performance Summary – Tuned, Oversampled data

('subsample': 0.7, 'n_estimators': 125, 'max_features': 0.5, 'learning_rate': 1)

Score	Training score	Validation score
Accuracy	0.993	0.971
Recall	0.993	0.845
Precision	0.994	0.693
F1	0.993	0.762

Results – Similar to the other tuned models,we see good results in the training set, but Recall score shows overfitting by 12 points.

XGBoosting Performance Summary – Tuned, Oversampled data

('subsample': 0.8, 'scale_pos_weight': 10, 'n_estimators': 250, 'learning_rate': 0.1, 'gamma': 3)

Score	Training score	Validation score
Accuracy	0.997	0.977
Recall	1.0	0.885
Precision	0.994	0.743
F1	0.997	0.808

Results – This tuned model shows the best results in the training set, but Recall score in validation performs 12 points less than in training. Precision and F1 are slightly better than GBoosting and Random Forest, but these scores are not as important for us.

Performance Summary – Training vs. Validation Set

Training Set:

Scores	GBoosting tuned, oversampled data	AdaBoost tuned, oversampled data	Random forest tuned, undersampled data	XGBoost tuned, oversampled data
Accuracy	0.993	0.992	0.961	0.997
Recall	0.993	0.988	0.933	1.000
Precision	0.994	0.995	0.989	0.994
F1	0.993	0.992	0.960	0.997

Validation Set:

Accuracy	0.971	0.979	0.938	0.977
Recall	0.845	0.853	0.885	0.885
Precision	0.693	0.790	0.468	0.743
F1	0.762	0.820	0.612	0.808

Performance Summary – Pipeline Choice

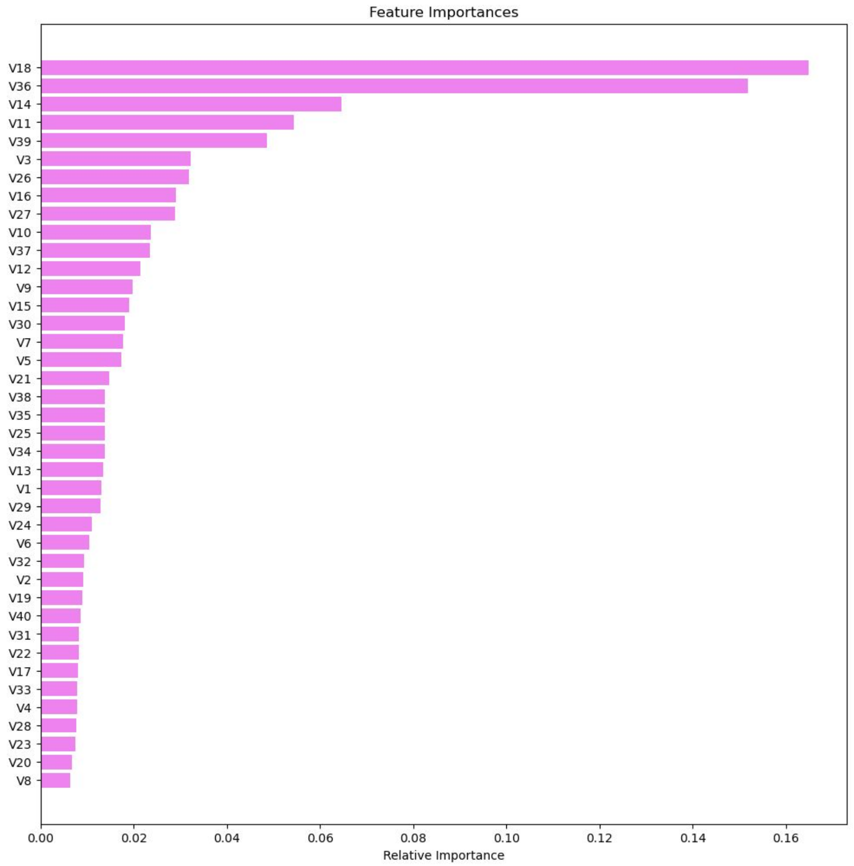
Summary – All tuned models show some level of overfitting between the training set and the validation set.
After analysis, we chose to continue with the **tuned Random Forest model**, trained on **under sampled data** for our pipeline:

```
Pipeline_model_rf = Pipeline(
    steps=[
        ("imputer", SimpleImputer(missing_values=np.nan, strategy="median")),
        (
            "rf",
            RandomForestClassifier(
                max_features='sqrt',
                random_state=1,
                max_samples=0.5,
                n_estimators=300,
                min_samples_leaf=2,
            ),
        ),
    ]
)
```

Performance Summary – Pipeline Results

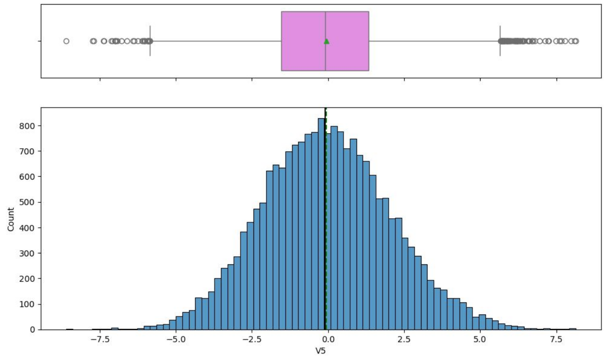
Scores	Full training set, under sampled	Full training set (original)	Test Set
Accuracy	0.962	0.951	0.945
Recall	0.929	0.929	0.872
Precision	0.995	0.534	0.507
F1	0.961	0.678	0.641

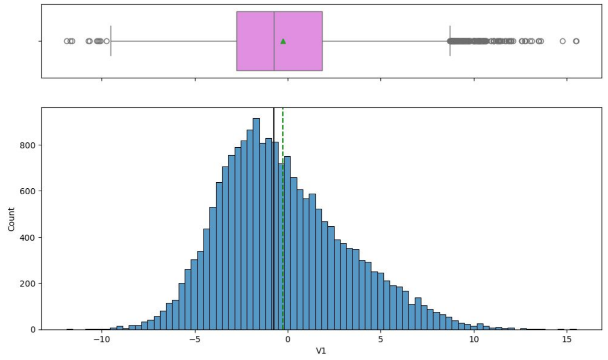
Analysis:
This model shows less overfitting when it comes to Recall score. When running it on the test set, it gives a good result of **0.87**

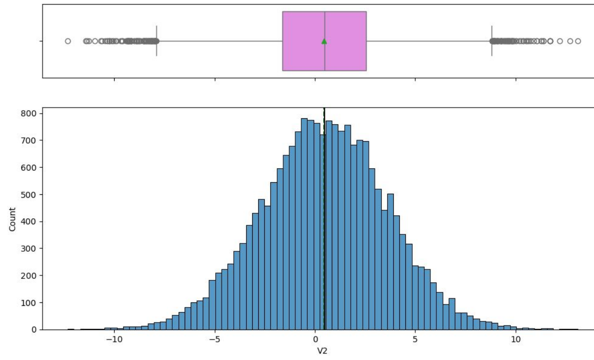


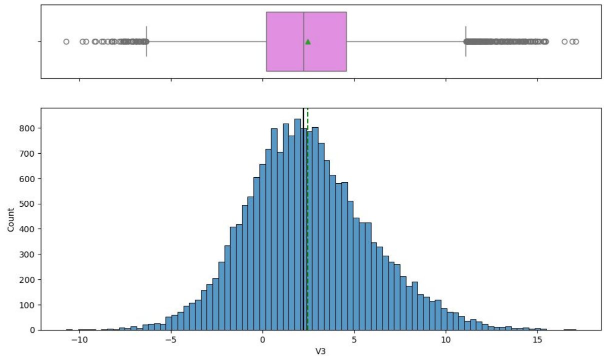
APPENDIX

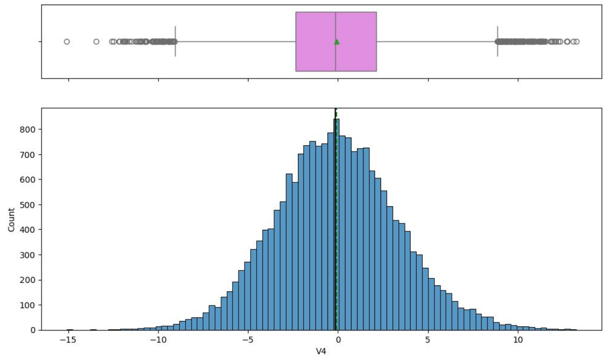
Attributes 1-6

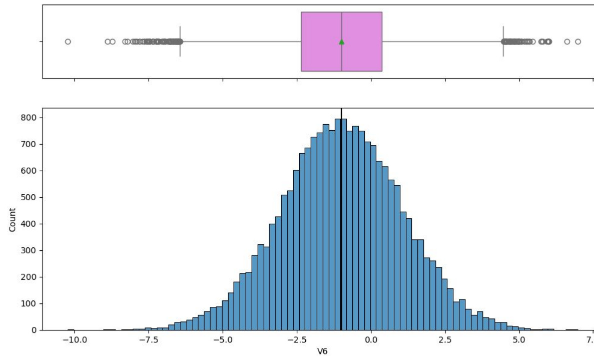




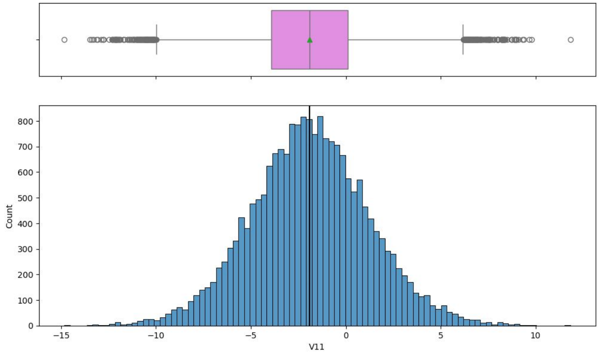


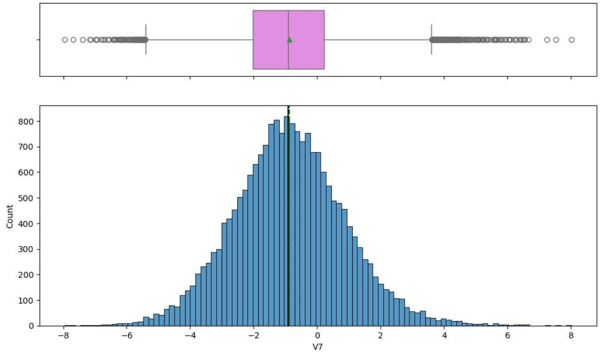


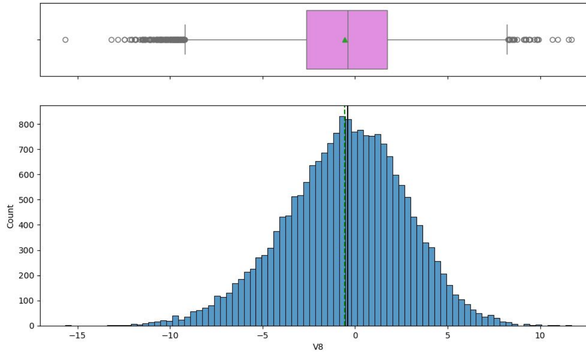


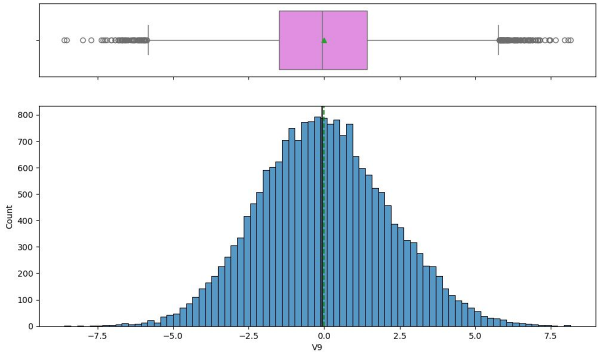


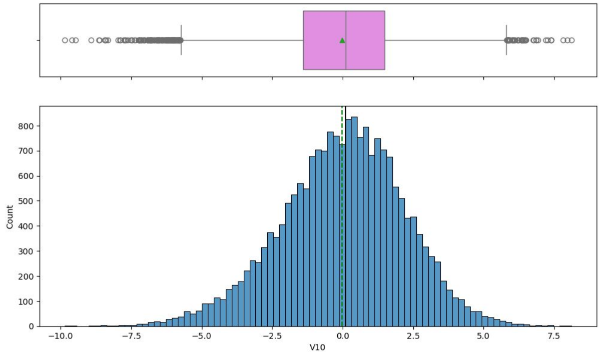
Attributes 7-12

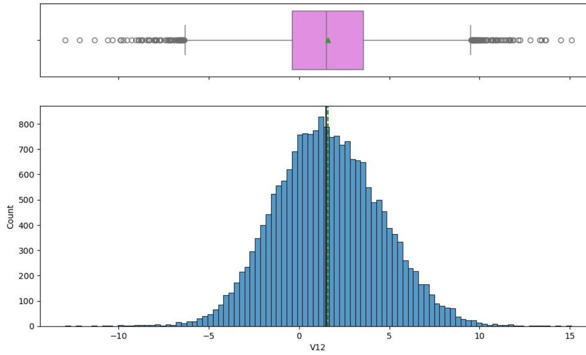




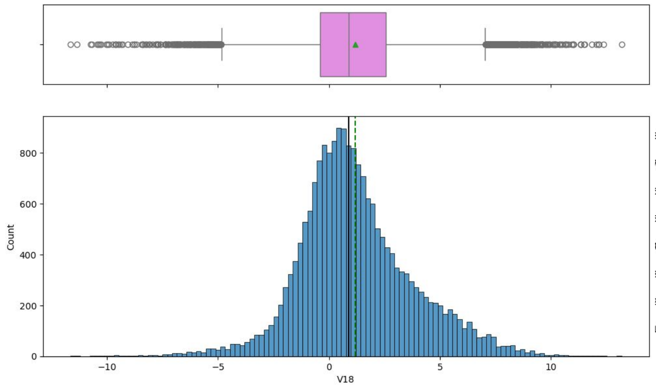


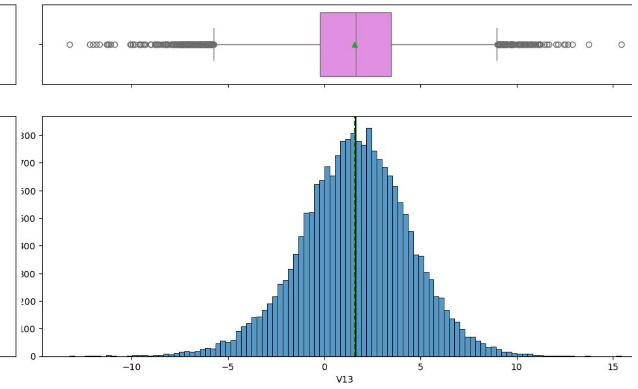


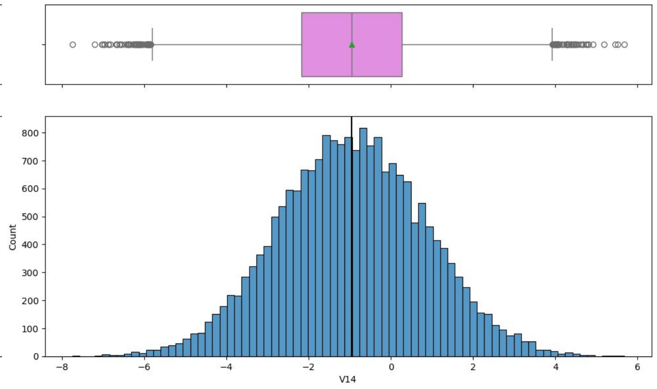


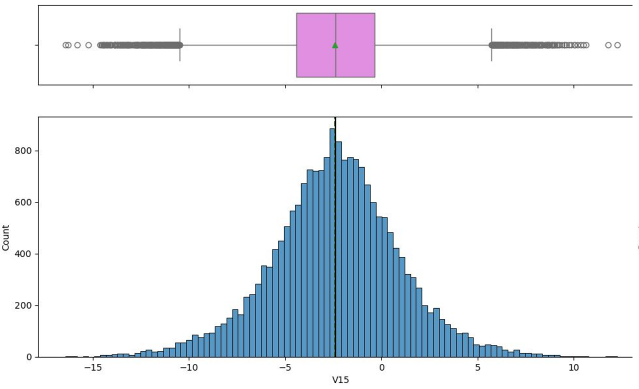


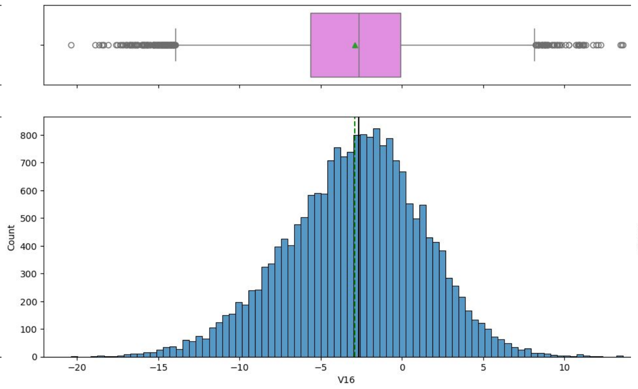
Attributes 13-18

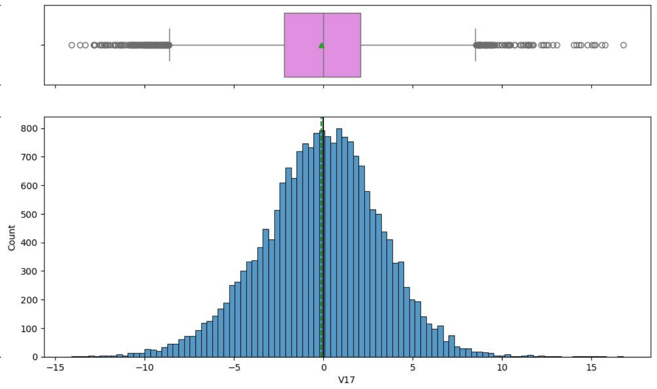




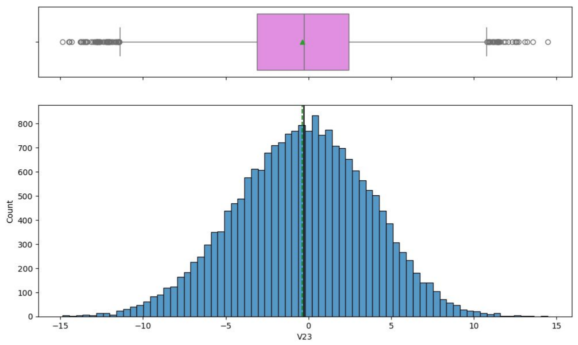


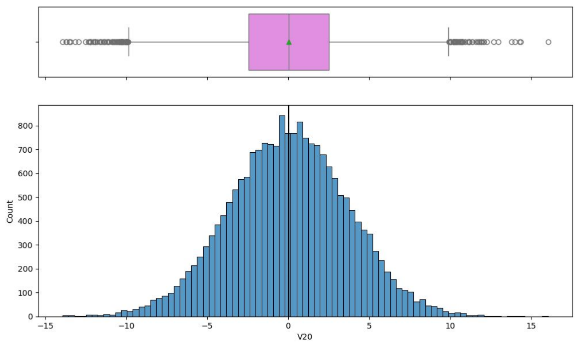


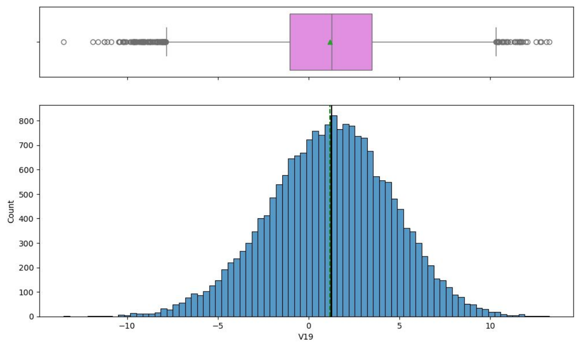


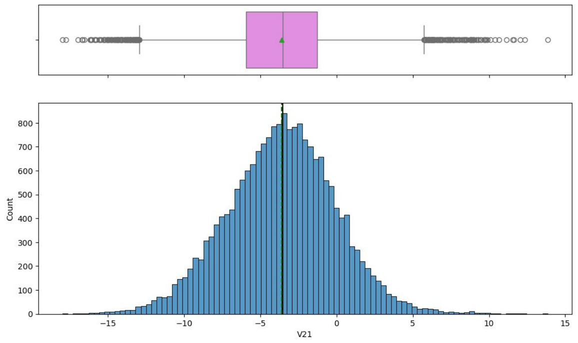


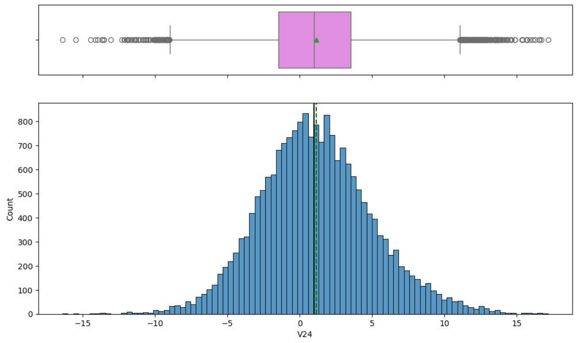
Attributes 19-24

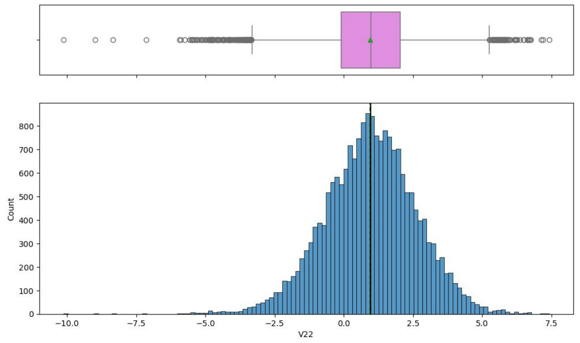




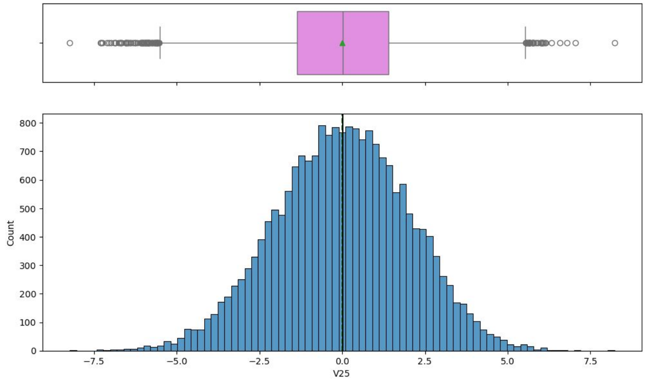


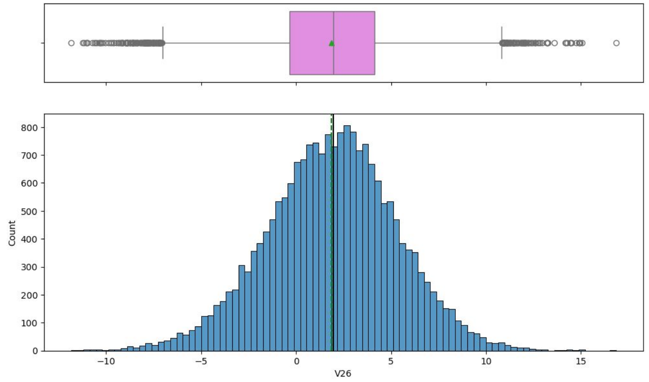


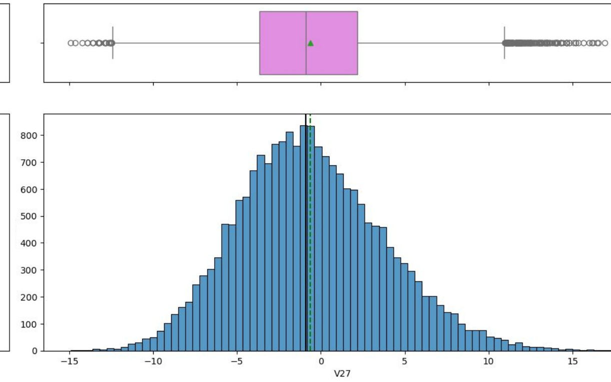


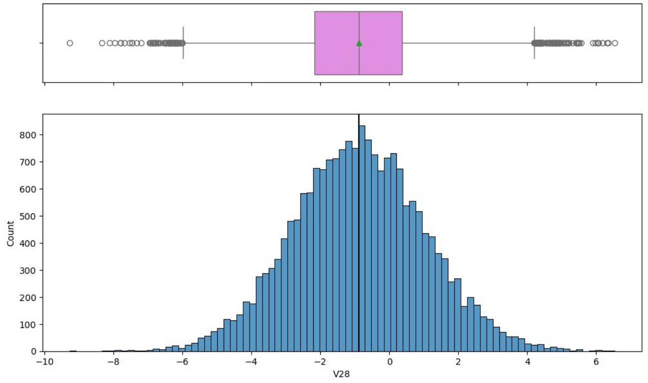


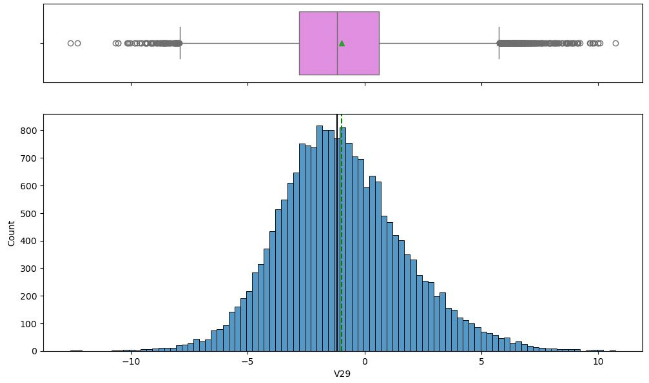
Attributes 25-30

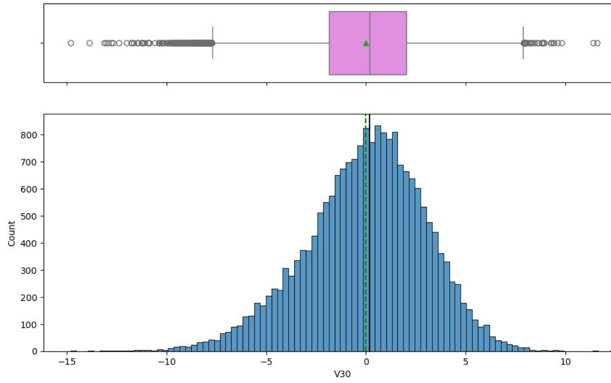




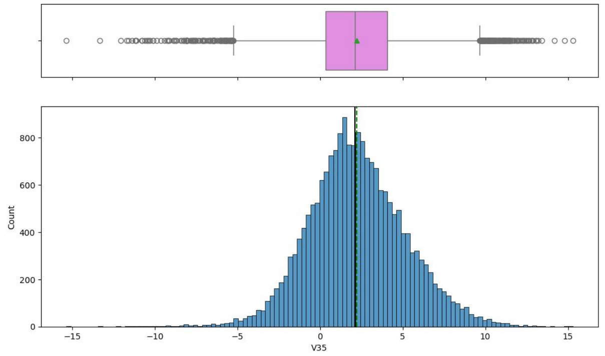


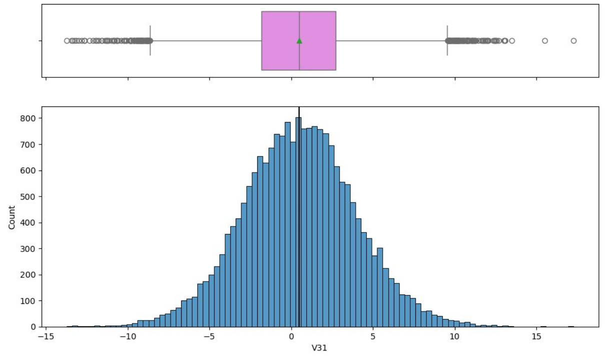


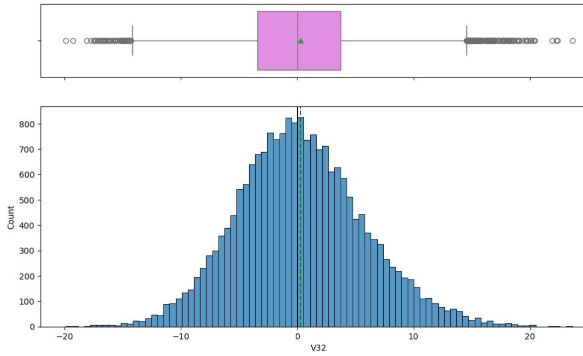


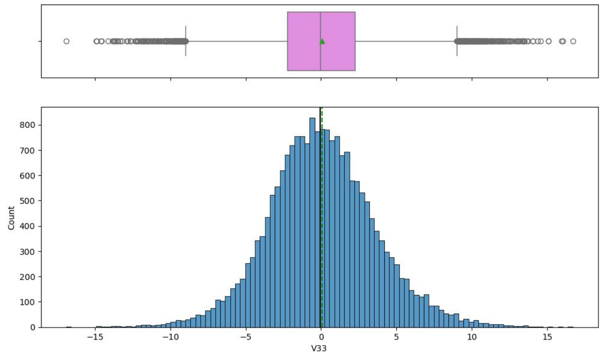


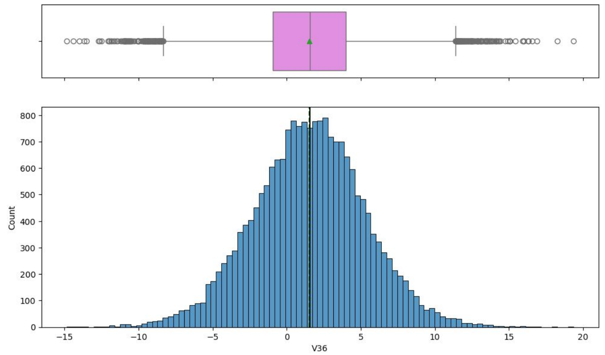
Attributes 31-36

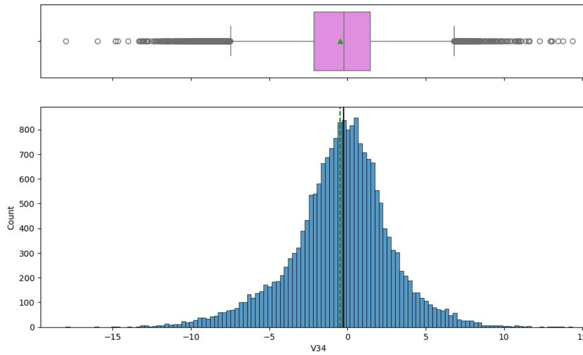




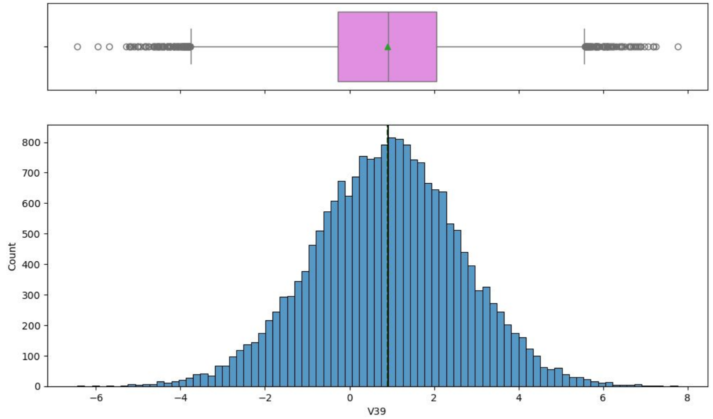


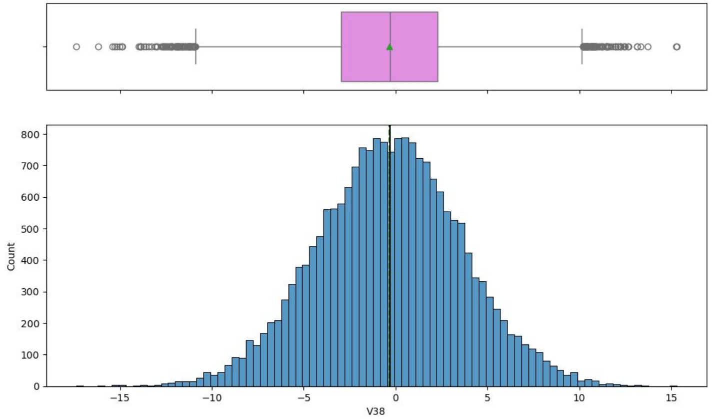


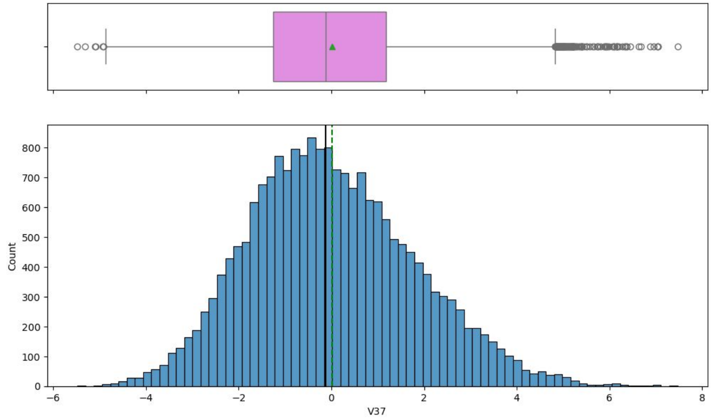


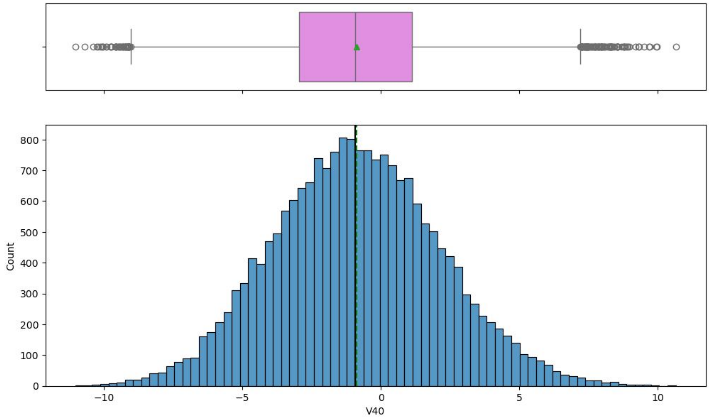


Attributes 37-40









Great Learning

Happy Learning !