ReneWind Project

Model Tuning

Aug 16, 2023

Contents / Agenda

- Executive Summary
- Business Problem Overview and Solution Approach
- EDA Results
- Data Preprocessing
- Model performance summary for hyperparameter tuning.
- Model building with pipeline
- Appendix

Executive Summary

Summary of observations and conclusions:

- Adaboost model is a suitable model to deploy for ReneWind project. This model has built to minimize the total maintenance cost on this project machinery/processes.
- The main (top 5) attributes of importance for predicting failure and no failure is V30, V18, V12, V22 and V26
- All predicting data from V1-V40 has a normal distribution and a good bell shape curve with few outliers.
- All features are significant in importance feature.

Business Problem Overview and Solution Approach

- Business problem overview:
  - All importance features (V1-V40) is significant. On business standpoint, this is consider as expensive and time-consuming if we want to determine the root cause of breakdown due to a lot of factors need to be considered.
  - About 6% of failures on test data, which is considered a lot for energy sector. Therefore, the company should assess the project risk and establish a back up plan whenever possible failure happen.
  - Extra manpower is needed due to 6% of failures, thus a manpower budget of high skilled worker need to be considered.

Business Problem Overview and Solution Approach

- Solution approach/business improvement/recommendation
  - Further investigations is needed to reduce the significant importance features.
  - Working manpower needs to be properly planned to make energy downtime/disruption shorter.
  - Process improvement is necessary (revise design for next project, searching for a better materials, usage arrangements etc.) as one of the items for future preventive disruption.
  - The company must have a yearly budget to keep train worker/manpower with high skilled technicians as one of the future preventive action to reduce downtime.

EDA Results

- Data shape on the train data: 20,000 rows, 41 columns
- Data shape on the test data: 5,000 rows, 41 columns
- First 5 data head on the train data as below:

	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

EDA Results

Great Learning
POWER AHEAD

- First 5 data head on the test data as below:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	V29	V30	V31	V32	V33	V34	V35	V36	V37	V38	V39	V40	Target
0	-0.613	-3.820	2.202	1.300	-1.185	-4.496	-1.836	4.723	1.206	-0.342	-5.123	1.017	4.819	3.269	-2.984	1.387	2.032	-0.512	-1.023	7.339	-2.242	0.155	2.054	-2.772	1.851	-1.789	-0.277	-1.255	-3.833	-1.505	1.587	2.291	-5.411	0.870	0.574	4.157	1.428	-10.511	0.455	-1.448	0
1	0.390	-0.512	0.527	-2.577	-1.017	2.235	-0.441	-4.406	-0.333	1.967	1.797	0.410	0.638	-1.390	-1.883	5.018	-3.827	2.418	1.762	-3.242	-3.193	1.857	-1.708	0.633	-0.588	0.084	3.014	-0.182	0.224	0.865	-1.782	-2.475	2.494	0.315	2.059	0.684	-0.485	5.128	1.721	-1.488	0
2	-0.875	-0.641	4.084	-1.590	0.526	-1.958	-0.695	1.347	-1.732	0.466	-4.928	3.565	-0.449	-0.656	-0.167	-1.630	2.292	2.396	0.601	1.794	-2.120	0.482	-0.841	1.790	1.874	0.364	-0.169	-0.484	-2.119	-2.157	2.907	-1.319	-2.997	0.460	0.620	5.632	1.324	-1.752	1.808	1.676	0
3	0.238	1.459	4.015	2.534	1.197	-3.117	-0.924	0.269	1.322	0.702	-5.578	-0.851	2.591	0.767	-2.391	-2.342	0.572	-0.934	0.509	1.211	-3.260	0.105	-0.659	1.498	1.100	4.143	-0.248	-1.137	-5.356	-4.546	3.809	3.518	-3.074	-0.284	0.955	3.029	-1.367	-3.412	0.906	-2.451	0
4	5.828	2.768	-1.235	2.809	-1.642	-1.407	0.569	0.965	1.918	-2.775	-0.530	1.375	-0.651	-1.679	-0.379	-4.443	3.894	-0.608	2.945	0.367	-5.789	4.598	4.450	3.225	0.397	0.248	-2.362	1.079	-0.473	2.243	-3.591	1.774	-1.502	-2.227	4.777	-6.560	-0.806	-0.276	-3.858	-0.538	0

EDA Results

- Statistical analysis on the train data

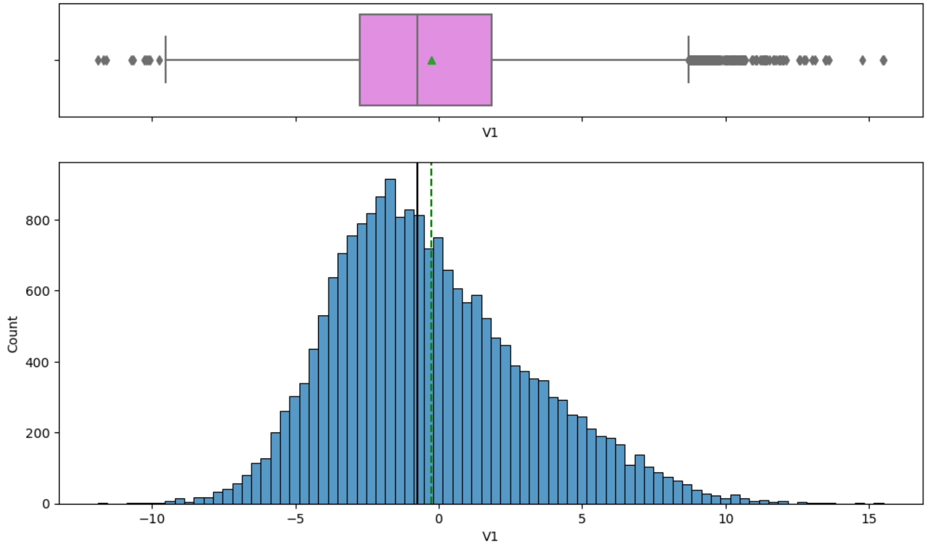
	count	mean	std	min	25%	50%	75%	max
V1	19982.000	-0.272	3.442	-11.876	-2.737	-0.748	1.840	15.493
V2	19982.000	0.440	3.151	-12.320	-1.641	0.472	2.544	13.089
V3	20000.000	2.485	3.389	-10.708	0.207	2.256	4.566	17.091
V4	20000.000	-0.083	3.432	-15.082	-2.348	-0.135	2.131	13.236
V5	20000.000	-0.054	2.105	-8.603	-1.536	-0.102	1.340	8.134
V6	20000.000	-0.995	2.041	-10.227	-2.347	-1.001	0.380	6.976
V7	20000.000	-0.879	1.762	-7.950	-2.031	-0.917	0.224	8.006
V8	20000.000	-0.548	3.296	-15.658	-2.643	-0.389	1.723	11.679
V9	20000.000	-0.017	2.161	-8.596	-1.495	-0.068	1.409	8.138
V10	20000.000	-0.013	2.193	-9.854	-1.411	0.101	1.477	8.108
V11	20000.000	-1.895	3.124	-14.832	-3.922	-1.921	0.119	11.826
V12	20000.000	1.605	2.930	-12.948	-0.397	1.508	3.571	15.081
V13	20000.000	1.580	2.875	-13.228	-0.224	1.637	3.460	15.420
V14	20000.000	-0.951	1.790	-7.739	-2.171	-0.957	0.271	5.671
V15	20000.000	-2.415	3.355	-16.417	-4.415	-2.383	-0.359	12.246
V16	20000.000	-2.925	4.222	-20.374	-5.634	-2.683	-0.095	13.583
V17	20000.000	-0.134	3.345	-14.091	-2.216	-0.015	2.069	16.756
V18	20000.000	1.189	2.592	-11.644	-0.404	0.883	2.572	13.180
V19	20000.000	1.182	3.397	-13.492	-1.050	1.279	3.493	13.238
V20	20000.000	0.024	3.669	-13.923	-2.433	0.033	2.512	16.052
V21	20000.000	-3.611	3.568	-17.956	-5.930	-3.533	-1.266	13.840
V22	20000.000	0.952	1.652	-10.122	-0.118	0.975	2.026	7.410
V23	20000.000	-0.366	4.032	-14.866	-3.099	-0.262	2.452	14.459
V24	20000.000	1.134	3.912	-16.387	-1.468	0.969	3.546	17.163
V25	20000.000	-0.002	2.017	-8.228	-1.365	0.025	1.397	8.223
V26	20000.000	1.874	3.435	-11.834	-0.338	1.951	4.130	16.836
V27	20000.000	-0.612	4.369	-14.905	-3.652	-0.885	2.189	17.560
V28	20000.000	-0.883	1.918	-9.269	-2.171	-0.891	0.376	6.528
V29	20000.000	-0.986	2.684	-12.579	-2.787	-1.176	0.630	10.722
V30	20000.000	-0.016	3.005	-14.796	-1.867	0.184	2.036	12.506
V31	20000.000	0.487	3.461	-13.723	-1.818	0.490	2.731	17.255
V32	20000.000	0.304	5.500	-19.877	-3.420	0.052	3.762	23.633
V33	20000.000	0.050	3.575	-16.898	-2.243	-0.066	2.255	16.692
V34	20000.000	-0.463	3.184	-17.985	-2.137	-0.255	1.437	14.358
V35	20000.000	2.230	2.937	-15.350	0.336	2.099	4.064	15.291
V36	20000.000	1.515	3.801	-14.833	-0.944	1.567	3.984	19.330
V37	20000.000	0.011	1.788	-5.478	-1.256	-0.128	1.176	7.467
V38	20000.000	-0.344	3.948	-17.375	-2.988	-0.317	2.279	15.290
V39	20000.000	0.891	1.753	-6.439	-0.272	0.919	2.058	7.760
V40	20000.000	-0.876	3.012	-11.024	-2.940	-0.921	1.120	10.654
Target	20000.000	0.056	0.229	0.000	0.000	0.000	0.000	1.000

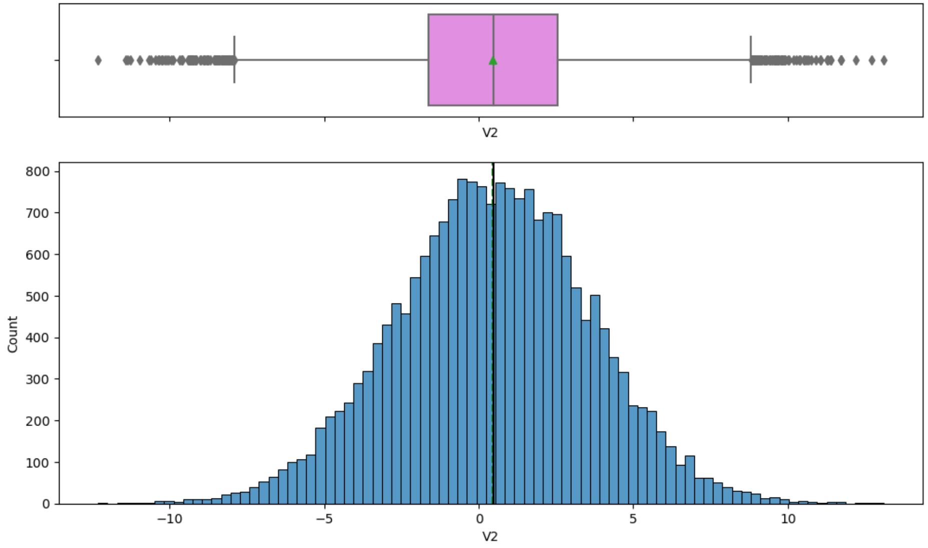
EDA Results

- Statistics summary shows:
  - The target value is either 0 or 1
  - The mean for each parameter is vary. The highest mean found is V3 (2.485) and the lowest mean found is V21 (-3.611)
- Data types:
  - V1 until V40 – float64
  - Target – int64
- No data duplication found for training and test data.
- Missing value for training and test data as below:

Missing value (Training data)	Missing value (Test data)
V1 – 18 V2 – 18	V1 – 5 V2 – 6

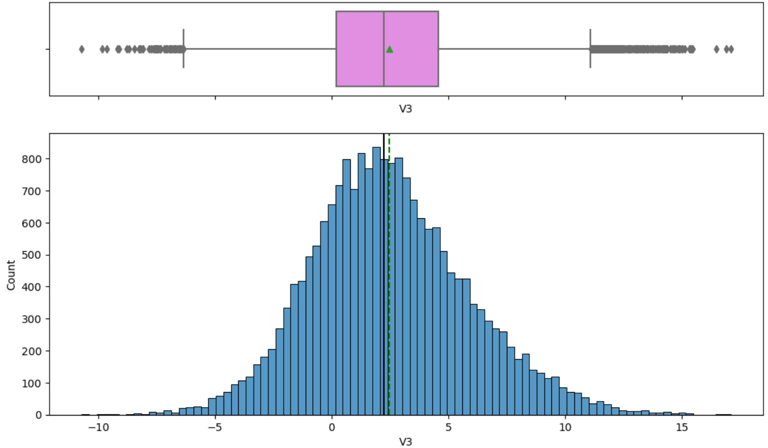
EDA Results

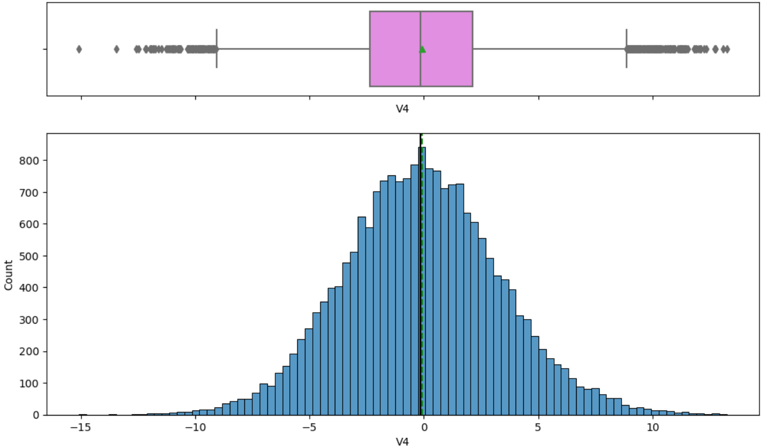


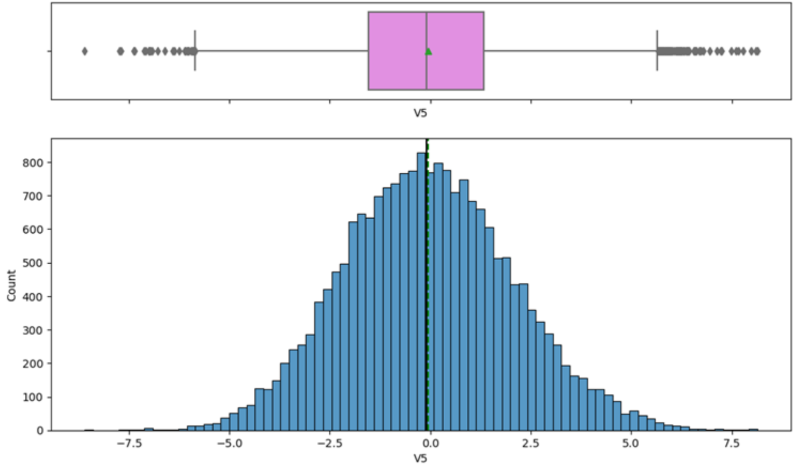


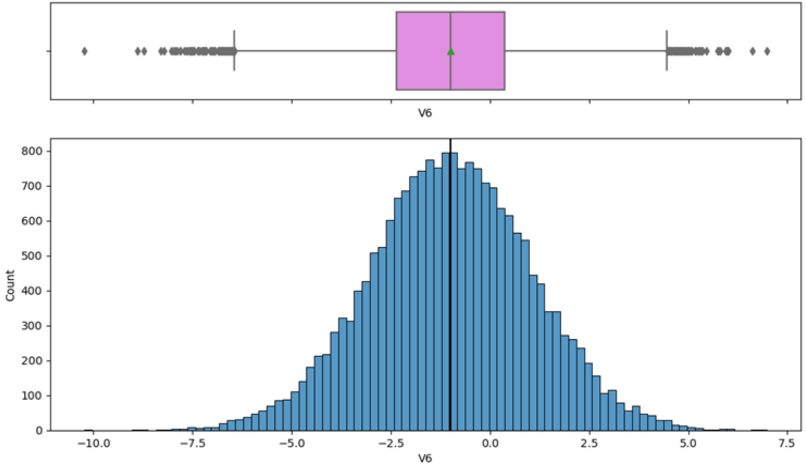
- Both data shows good bell shape curve.
- However, slight outlier was detected on V1 where it skewed to the right.

EDA Results



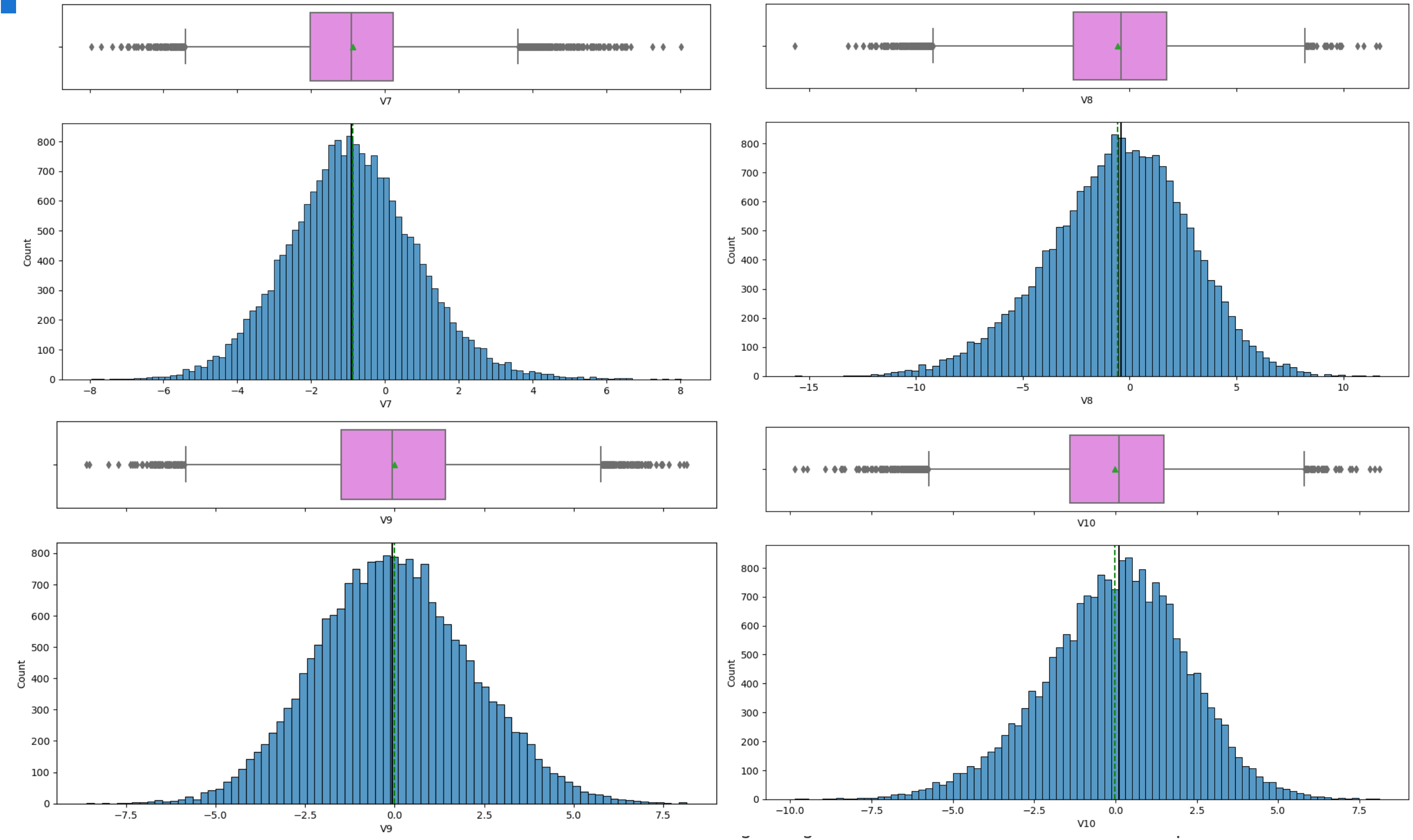






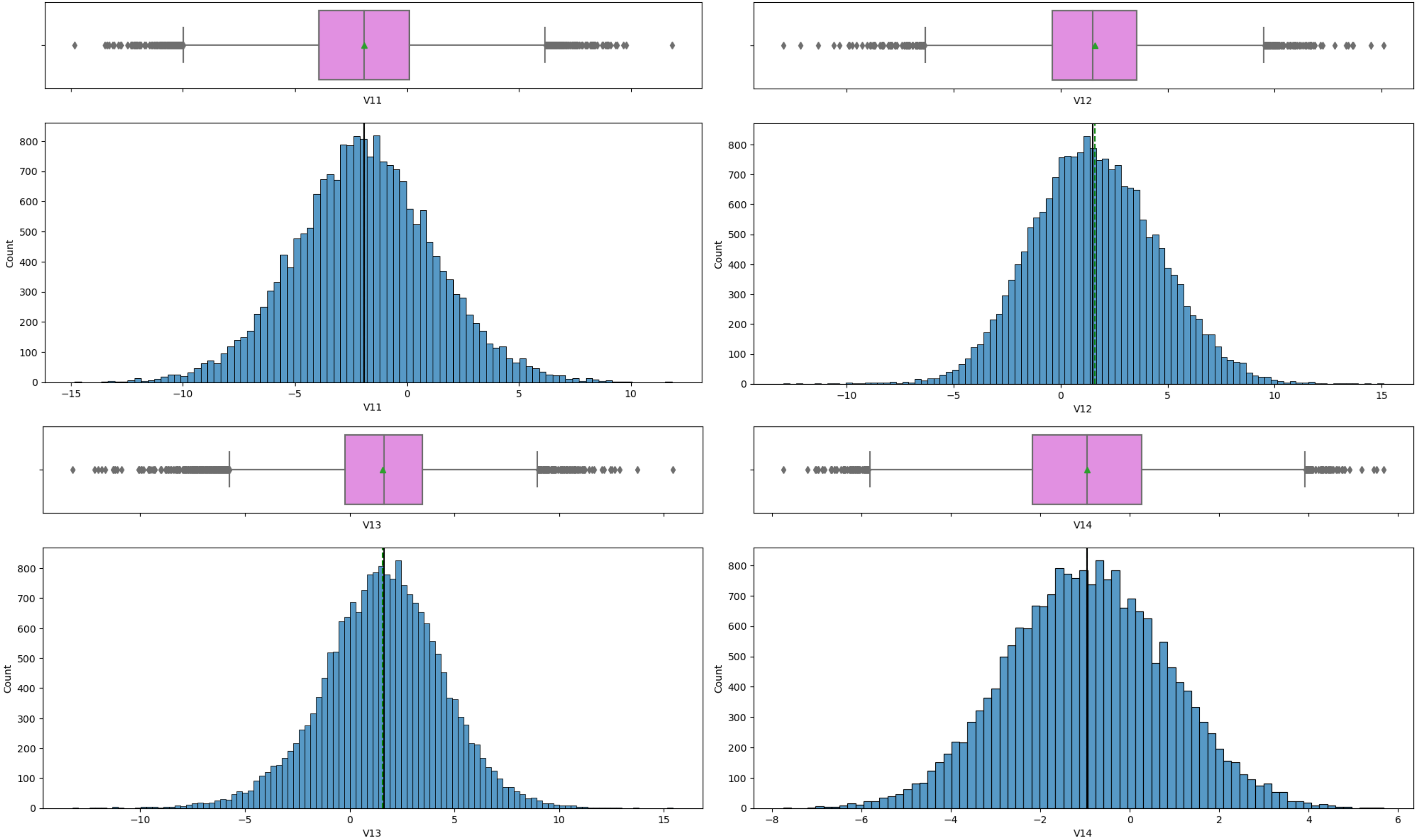
- All data shows good bell shape curve and a good data distribution.

EDA Results



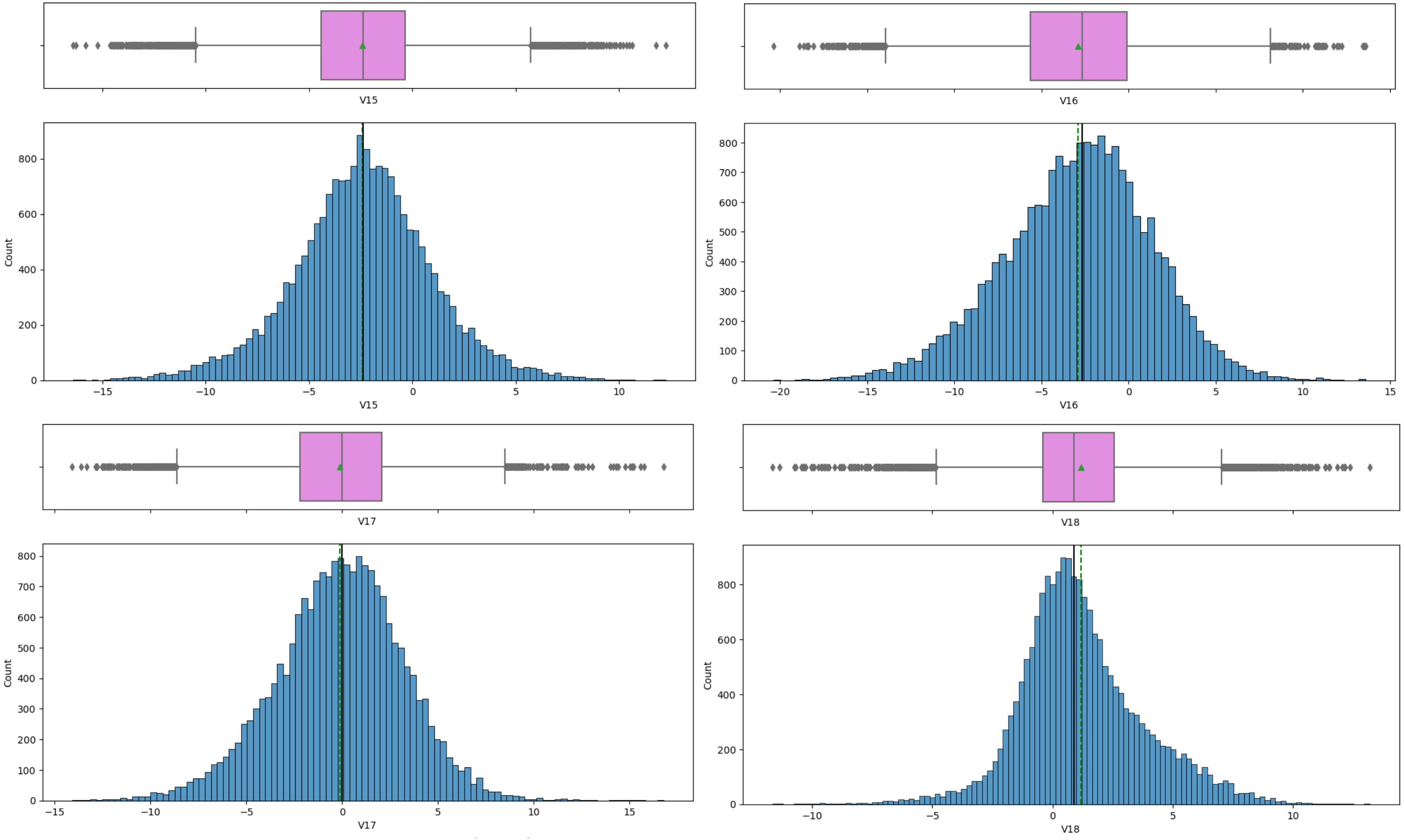
- All data shows good bell shape curve and a good data distribution.
- However, V8 observed slightly skewed to the left.

EDA Results



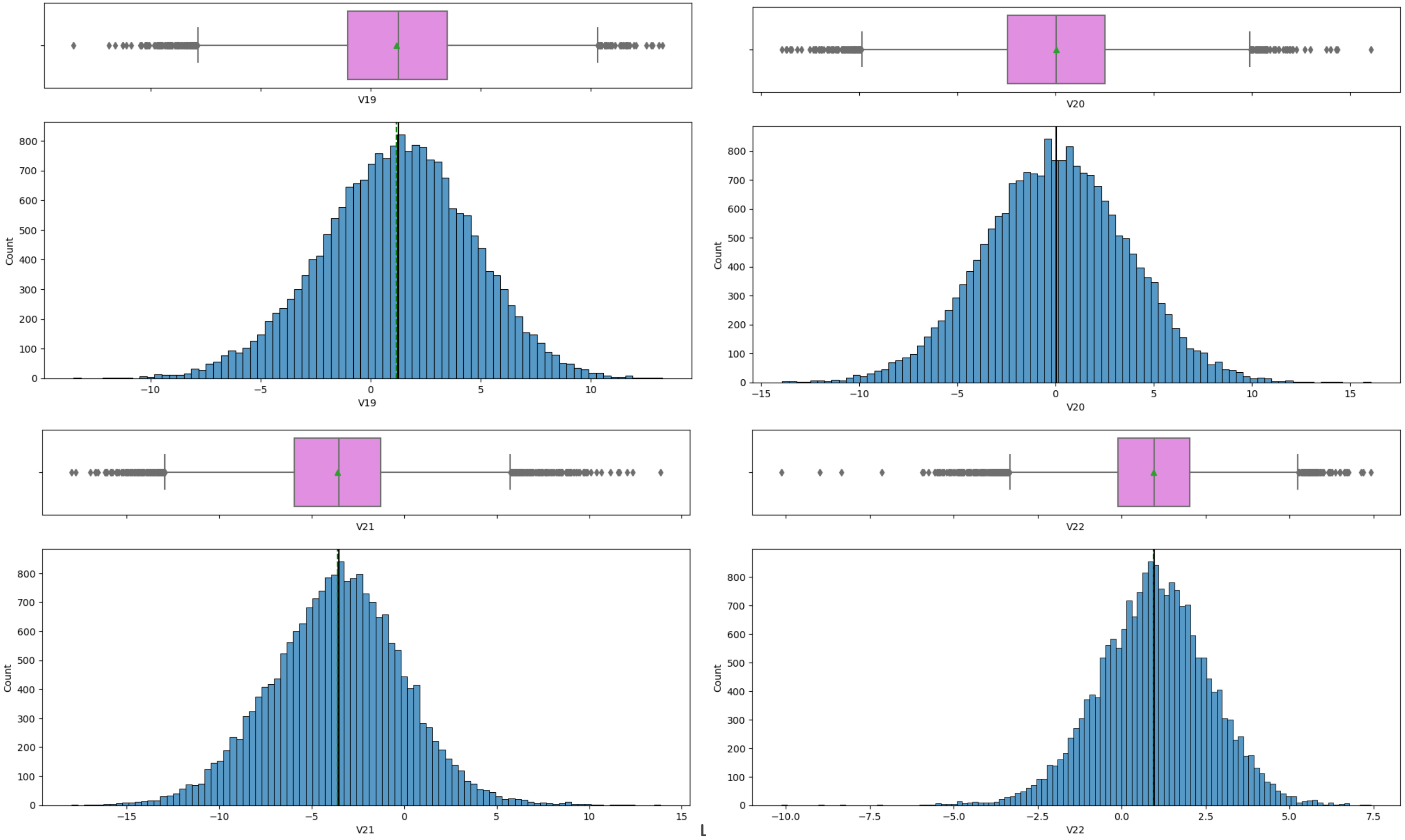
- All data shows good bell shape curve and a good data distribution.

EDA Results



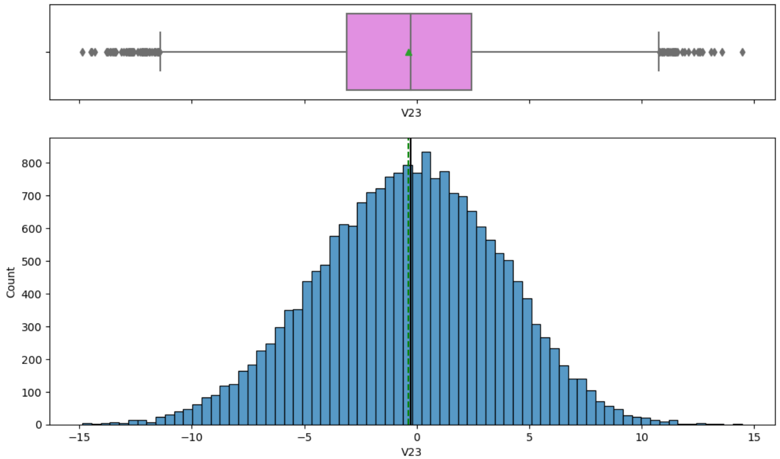
- All data shows good bell shape curve and a good data distribution.

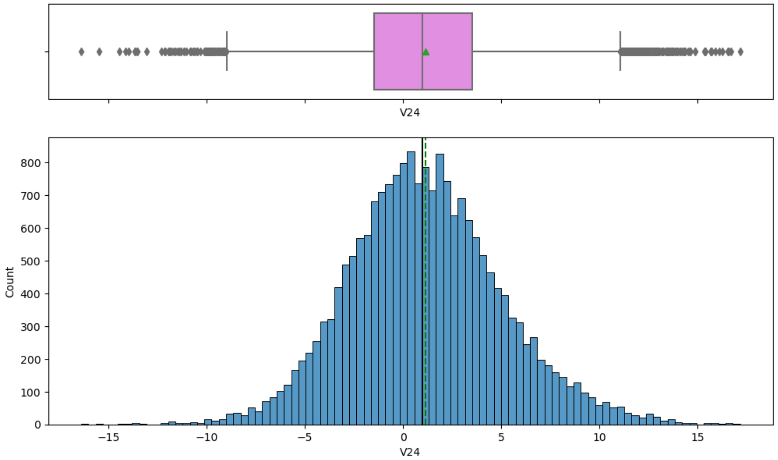
EDA Results

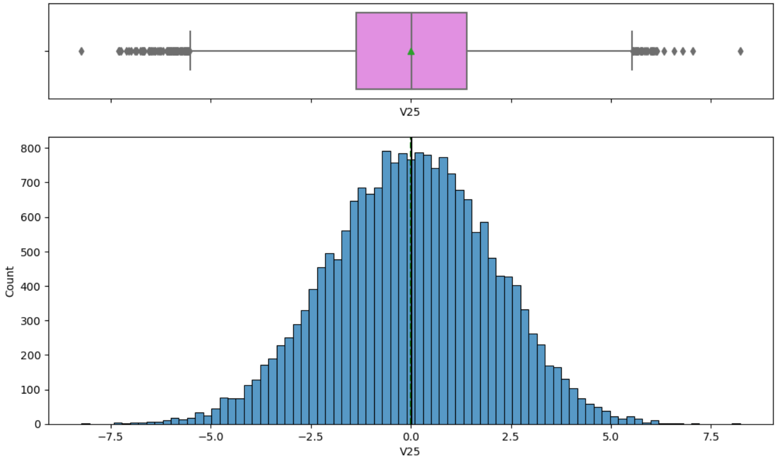


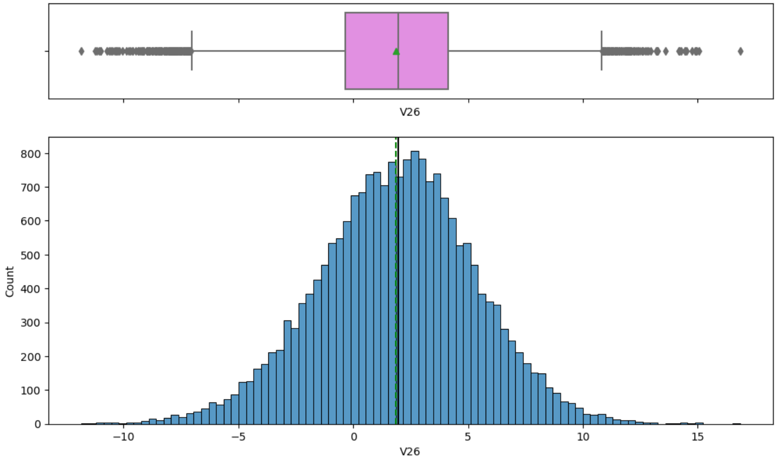
- All data shows good bell shape curve and a good data distribution.

EDA Results



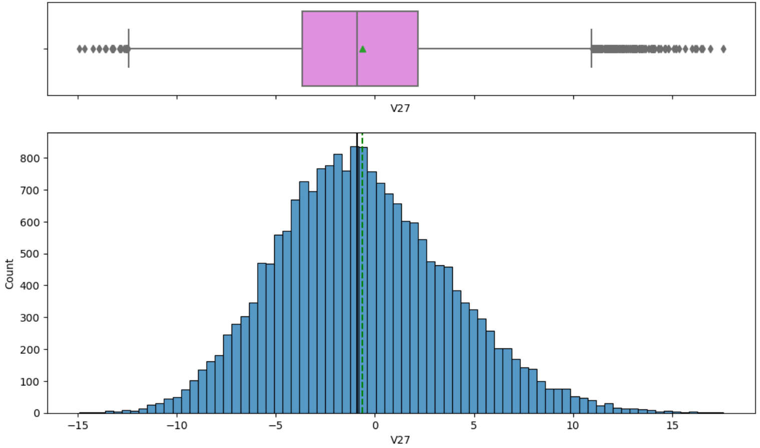


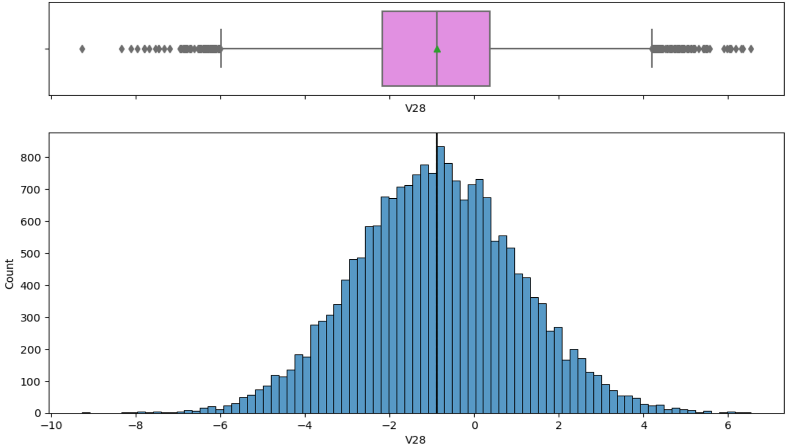


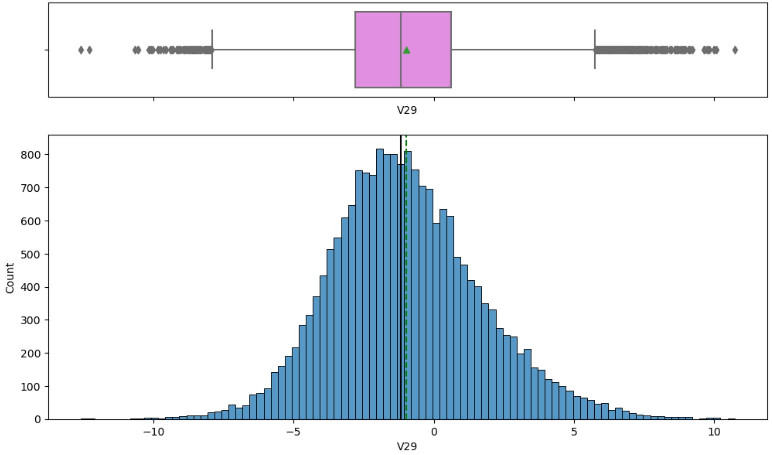


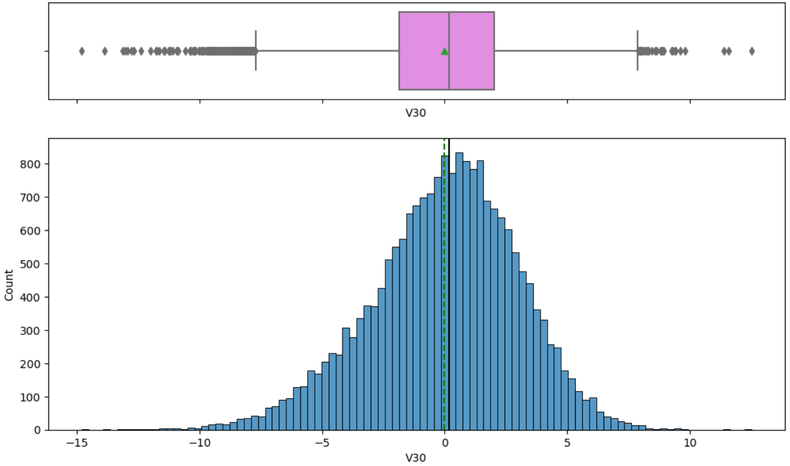
- All data shows good bell shape curve and a good data distribution.

EDA Results



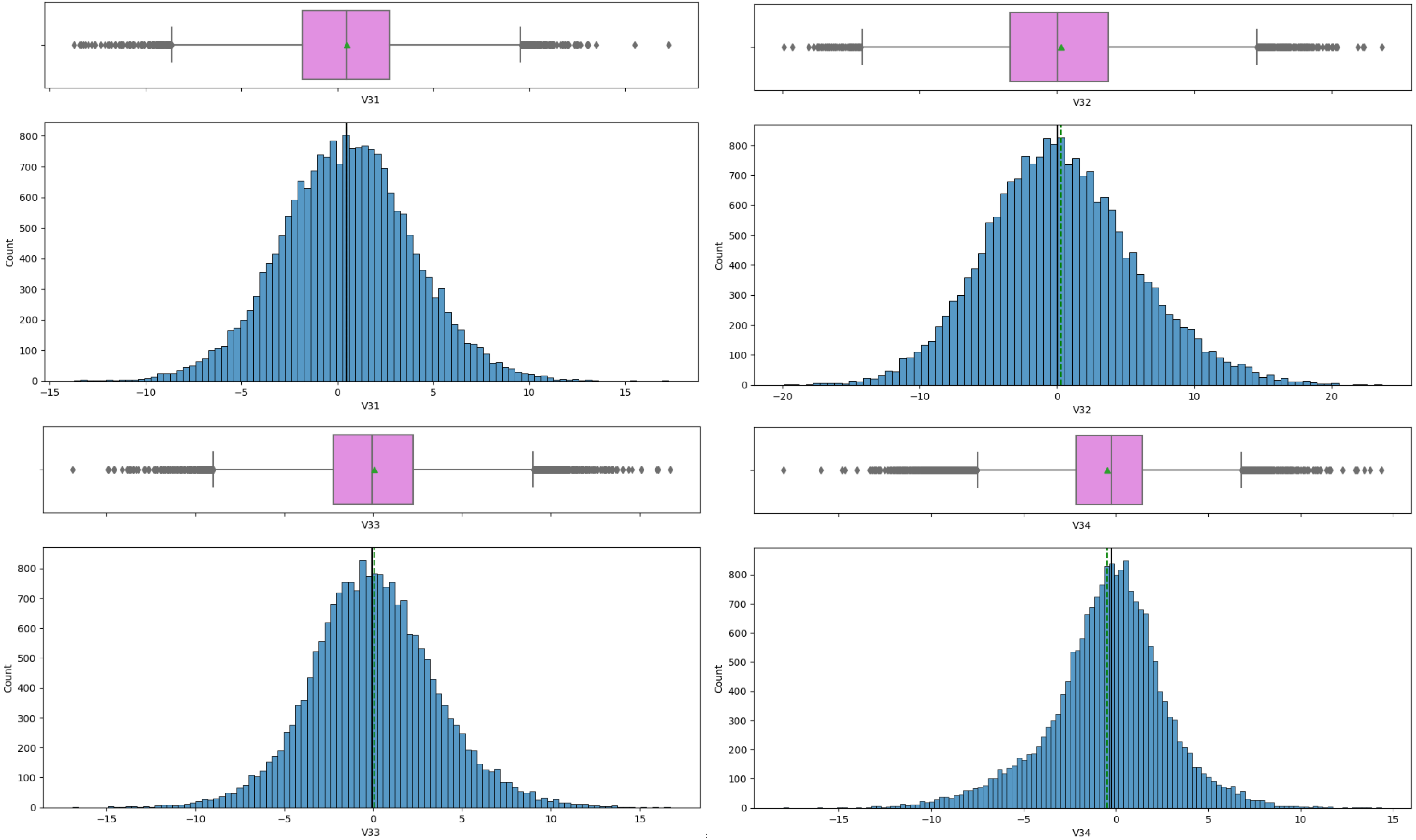






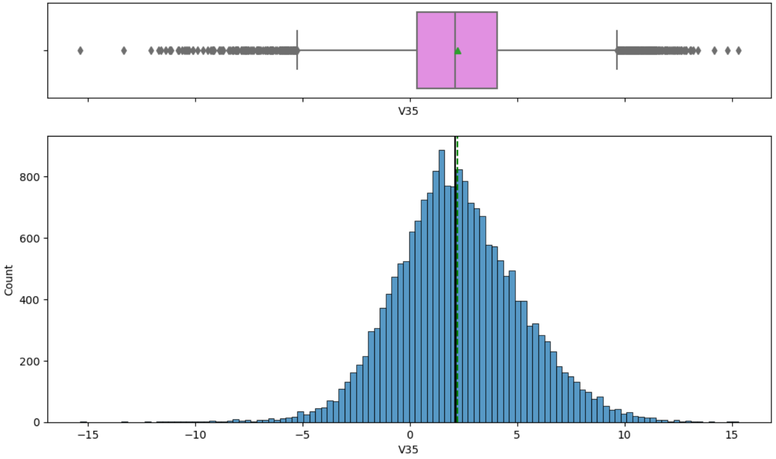
- All data shows good bell shape curve and a good data distribution.

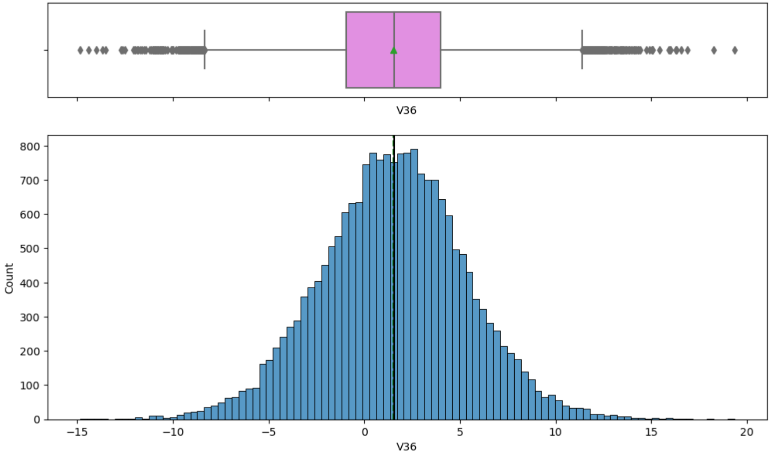
EDA Results

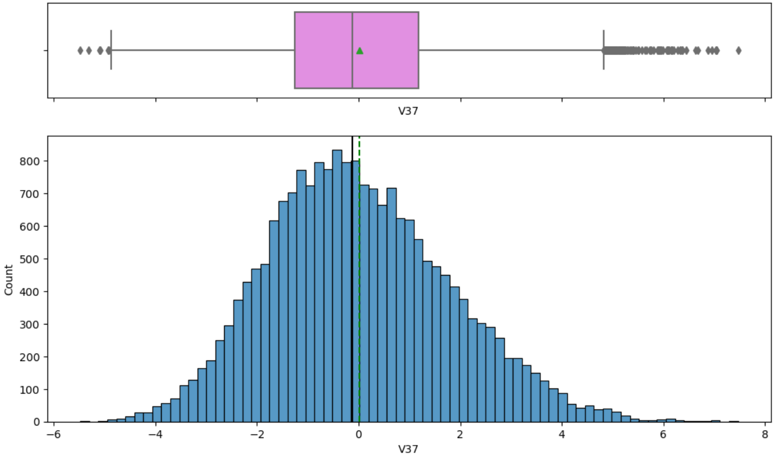


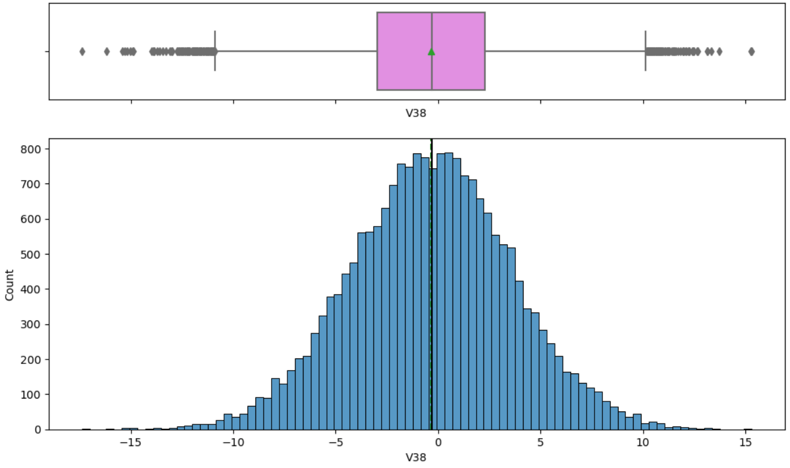
- All data shows good bell shape curve and a good data distribution.

EDA Results



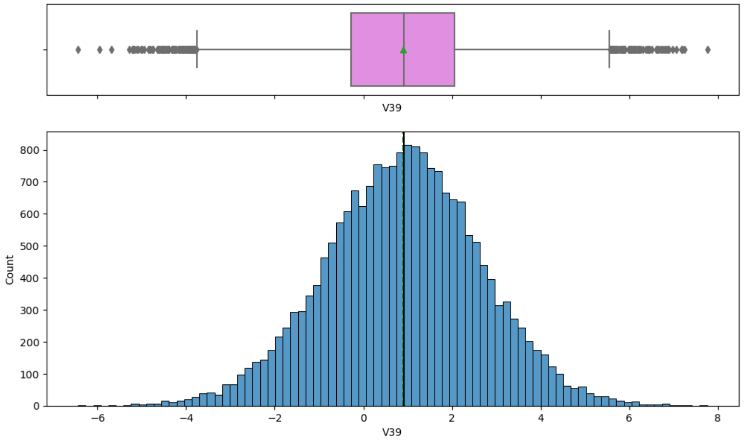


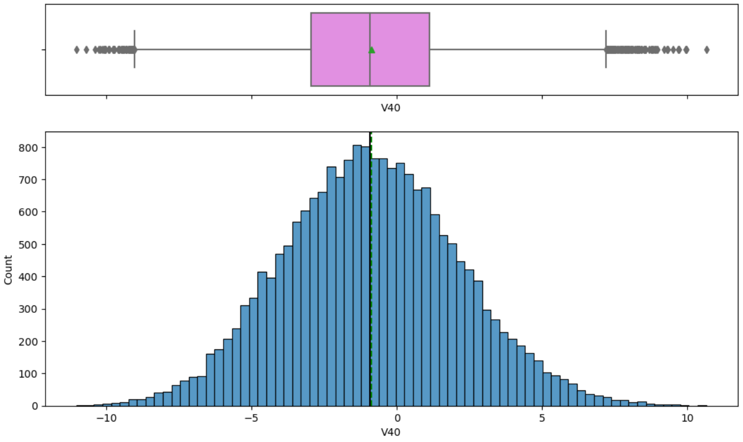


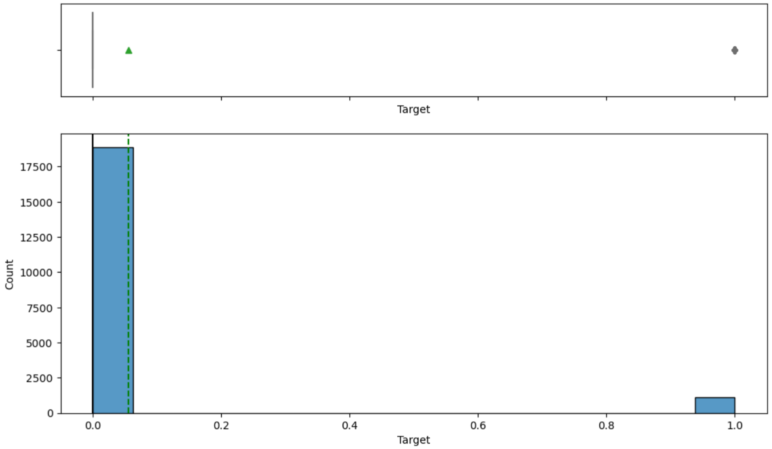


- All data shows good bell shape curve and a good data distribution.

EDA Results



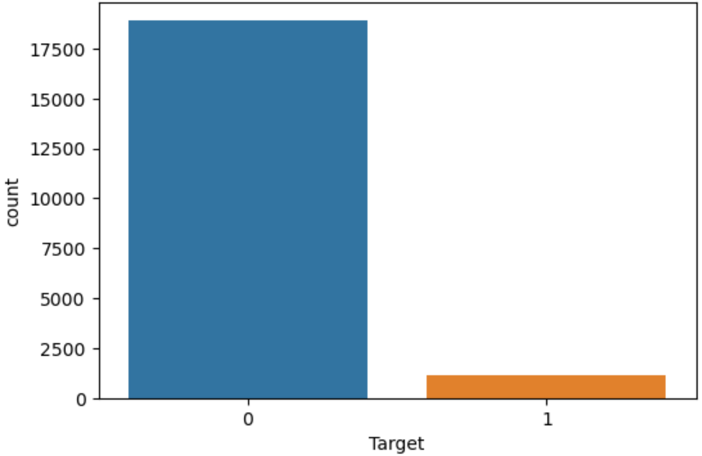




- All data shows good bell shape curve and a good data distribution.
- Target value is either 0 or 1.

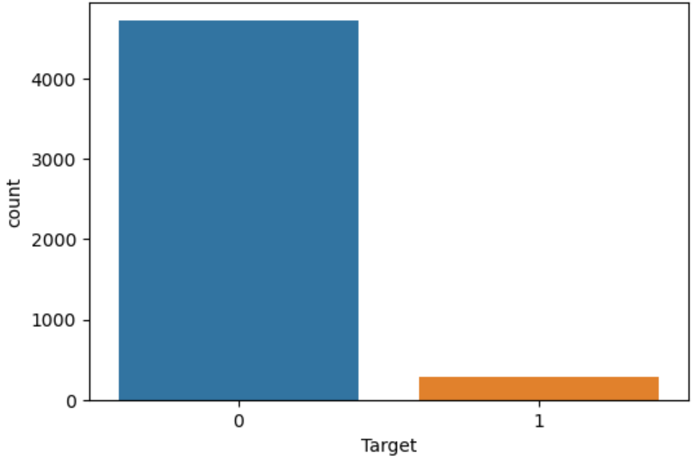
EDA Results

- Train data



0	18890
1	1110

- Test data



0	4718
1	282

- The value from the train and test data is comparable, where 0 represents no failure and 1 is represent a failure.
- Majority data on both train and test set where each is 18,890 and 4718 resulting of no failure while 1110 and 282 results of failure was detected.

Data Preprocessing

- Total data for model training:
  - 15,000 columns, 40 rows
- Total data for validation training:
  - 5,000 columns, 40 rows
- Total data for testing:
  - 5,000 columns, 40 rows
- Total data for testing:
  - 5,000 columns, 40 rows
- All data on train, validation and test set has no missing value
- Six models were chosen for building a model. All these models were started with building on the original data.
  - Logistic Regression
  - Bagging
  - Random Forest
  - GBM
  - Adaboost
  - Decision Tree

Data Preprocessing – Model building on the original data

- Cross validation on the training set:

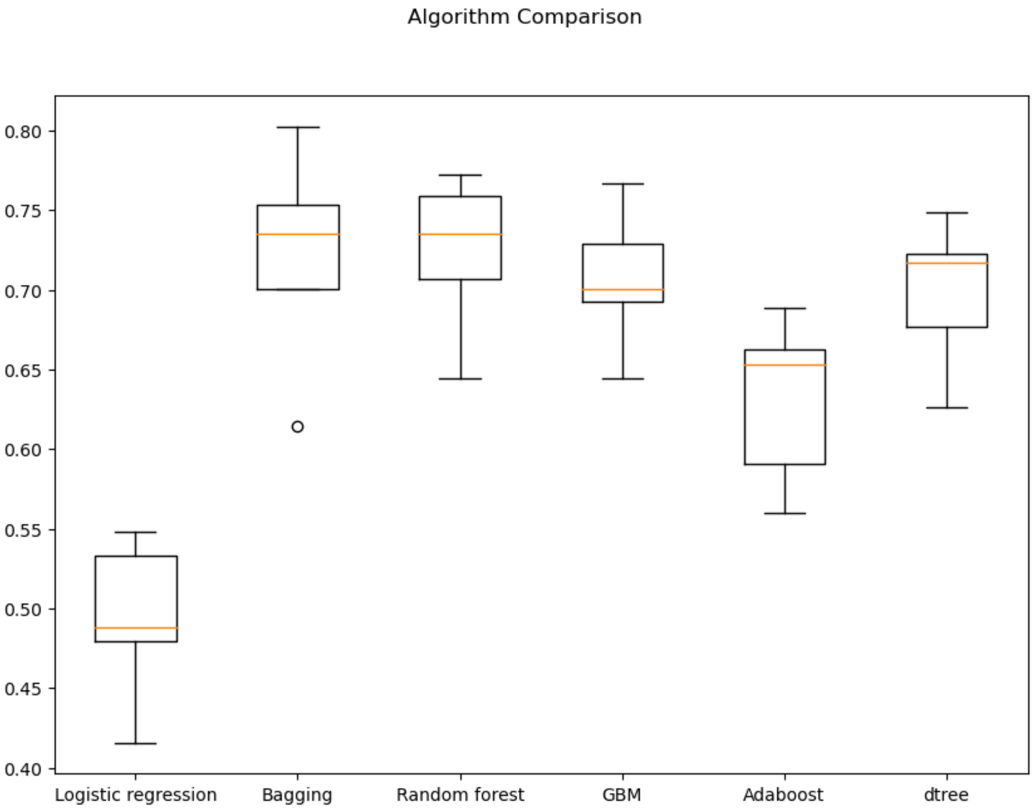
```
Logistic regression: 0.4927566553639709
Bagging: 0.7210807301060529
Random forest: 0.7235192266070268
GBM: 0.7066661857008874
Adaboost: 0.6309140754635308
dtree: 0.6982829521679532
```

- Cross validation on the validation set:

```
Logistic regression: 0.48201438848920863
Bagging: 0.7302158273381295
Random forest: 0.7266187050359713
GBM: 0.7230215827338129
Adaboost: 0.6762589928057554
dtree: 0.7050359712230215
```

- Both result on training and validation data is comparable on all 6 chosen models. However, we are going to try to perform hyperparameter tuning to see whether we can improve the percentage for each model.

Data Preprocessing – Algorithm Comparison on the original data



- All 6 chosen models were compared.
- Logistic regression was found the most inferior compared to the rest on the original data.
- Bagging was found skewed to the right, while random forest was skewed to the left.
- Gradient Boost and Decision tree almost comparable, while Adaboost is the second inferior after logistic regression.
- We will test again all the 6 models on the oversampled data.

Data Preprocessing – Model building on oversampled data

- SMOTE
  - Prior of performing SMOTE, we need to check the shape and value count of 0 (no failure) and 1 (failure).
  - Therefore, it is important to keep/monitor our Recall percentage on our chose model.
  - It is found before performing SMOTE, the value count is not balance between 0 and 1.
  - It is found after SMOTE, the value count now is balance and we found the shape for the train data stated below.

```
Before OverSampling, counts of label '1': 832
Before OverSampling, counts of label '0': 14168

After OverSampling, counts of label '1': 14168
After OverSampling, counts of label '0': 14168

After OverSampling, the shape of train_X: (28336, 40)
After OverSampling, the shape of train_y: (28336,)
```

Data Preprocessing – Model building on oversampled data

- Cross validation on the training set:

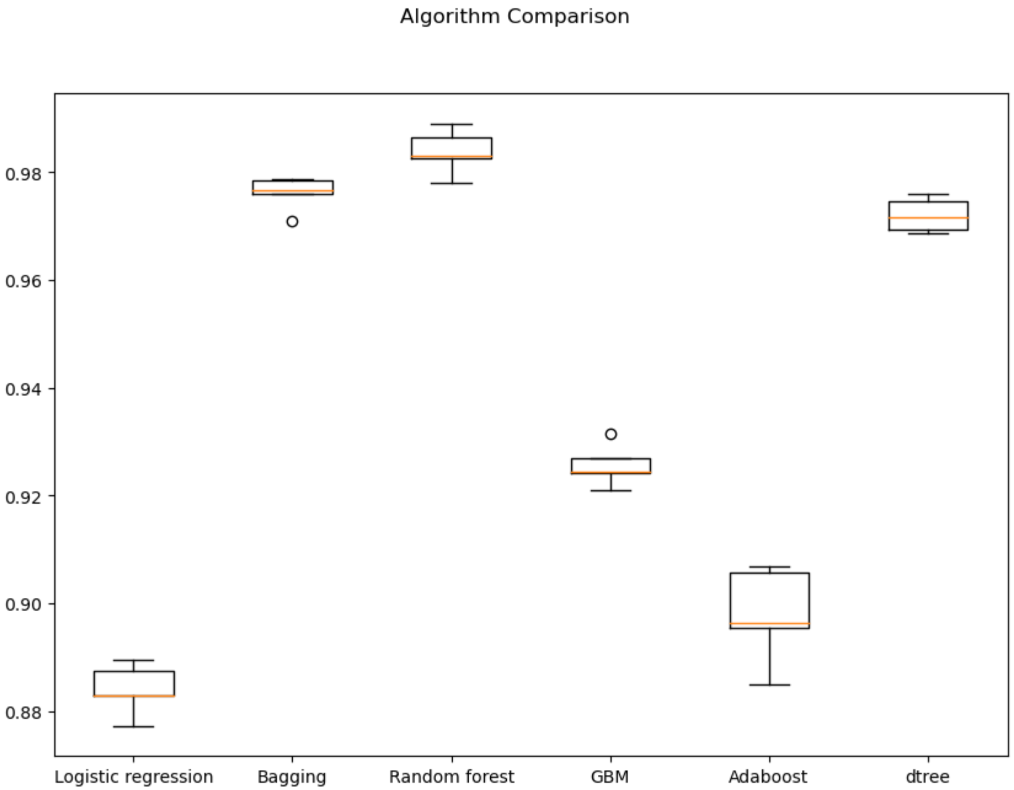
```
Logistic regression: 0.6982829521679532
Bagging: 0.6982829521679532
Random forest: 0.6982829521679532
GBM: 0.6982829521679532
Adaboost: 0.6982829521679532
dtree: 0.6982829521679532
```

- Cross validation on the validation set:

```
Logistic regression: 0.8489208633093526
Bagging: 0.8345323741007195
Random forest: 0.8489208633093526
GBM: 0.8776978417266187
Adaboost: 0.8561151079136691
dtree: 0.7769784172661871
```

- Most data on training and validation on oversampled data set is far from each other, therefore we need to try to fine tune on undersampled data.
- We try to pick 2 model to further improve during RandomizedSearchCV, in this case, we are going to try on Gradient Boost and AdaBoost, since their percentage on validation data is convincing and may improve after tuning.

Data Preprocessing – Algorithm Comparison on oversampled data



- Logistic regression remains the most inferior compared with other 5 models on oversampled data.
- AdaBoost and Gradient Boost became worst compared to its performance on original data. However, these 2 models are start to having a small range compared on the original data, although Adaboost is slightly skewed to the left.
- Bagging, Random Forest and Decision tree is slightly improved on oversampled data and the 3 models are having smaller range.
- Outlier on Bagging is remains both on original and oversampled data.

Data Preprocessing – Model building on undersampled data

- RandomUnderSampler
  - Prior of performin RandomUnderSampler, we need to check the shape and value count of 0 (no failure) and 1 (failure).
  - Therefore, it is important to keep/monitor our Recall percentage on our chose model.
  - It is found before performing RandomUnderSampler, the value count is not balance between 0 and 1. Label 1 is inferior/less/imbalance compared with label 0.
  - It is found after perform RandomUnderSampler, the value count now is balance and we found the shape for the train data stated below.

```
Before UnderSampling, counts of label '1': 832
Before UnderSampling, counts of label '0': 14168

After UnderSampling, counts of label '1': 832
After UnderSampling, counts of label '0': 832

After UnderSampling, the shape of train_X: (1664, 40)
After UnderSampling, the shape of train_y: (1664,)
```

Data Preprocessing – Model building on undersampled data

- Cross validation on the training set:

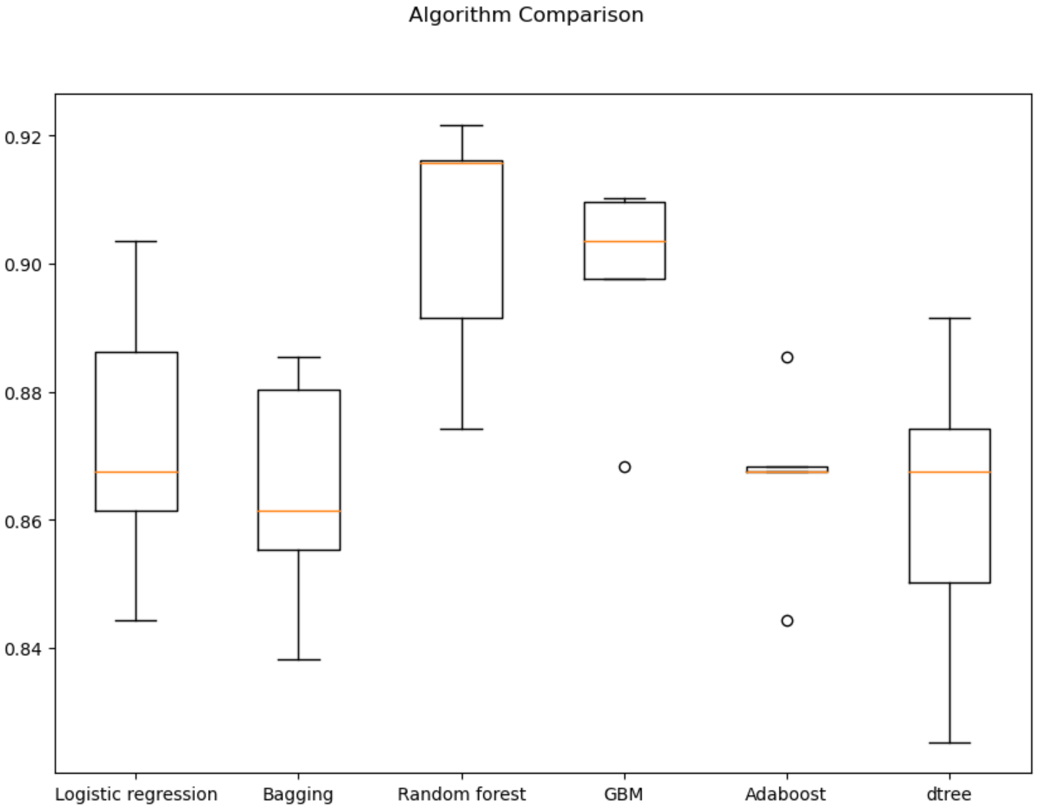
```
Logistic regression: 0.6982829521679532
Bagging: 0.6982829521679532
Random forest: 0.6982829521679532
GBM: 0.6982829521679532
Adaboost: 0.6982829521679532
dtree: 0.6982829521679532
```

- Cross validation on the validation set:

```
Logistic regression: 0.8525179856115108
Bagging: 0.8705035971223022
Random forest: 0.8920863309352518
GBM: 0.8884892086330936
Adaboost: 0.8489208633093526
dtree: 0.841726618705036
```

- Most data on training and validation on undersampled data set is far from each other, even after hyperparameter tuning.
- However, result on Random Forest can been observed having the highest percentage on validation set compared with other 5 models. We are going to choose this model for next step on RandomizedSearchCV to further improve our training and validation data.

Data Preprocessing – Algorithm Comparison on undersampled data



- Most model is performing worst than oversampled data, where 4 out of 6 models are having wider range.
- Logistic regression became better compared with original and oversampled data, but the range is too wide.
- Bagging is no longer observed having outlier, but the new outlier is observed on GBM and Adaboost.
- Adaboost range is improved but it is having far outlier from their population data.

Hyperparameter Tuning – AdaBoost with oversampled data

- Best parameter after tuning with RandomizedSearchCV:

```
Best parameters are {'n_estimators': 70, 'learning_rate': 1, 'base_estimator': DecisionTreeClassifier(max_depth=3, random_state
=1)} with CV score=0.976355414971399:
Wall time: 19min 38s
```

- AdaBoost training performance:

	Accuracy	Recall	Precision	F1
0	0.994	0.994	0.995	0.994

- AdaBoost validation performance:

	Accuracy	Recall	Precision	F1
0	0.975	0.860	0.733	0.791

- We can observed the percentage is improving after RandomizedSearchCV. Precision and F1 score is quite low, some signal of overfitting.
- However, since our concern is Recall, we are going to compare with other model to observe whether Adaboost is a better model.

Hyperparameter Tuning – Random Forest with undersampled data

- Best parameter after tuning with RandomizedSearchCV:

```
Best parameters are {'n_estimators': 300, 'min_samples_leaf': 2, 'max_samples': 0.5, 'max_features': 'sqrt'} with CV score=0.89
90116153235697:
Wall time: 37.2 s
```

- Random Forest training performance:

	Accuracy	Recall	Precision	F1
0	0.961	0.933	0.989	0.960

- Random Forest validation performance:

	Accuracy	Recall	Precision	F1
0	0.938	0.885	0.468	0.612

- Random forest model on Recall can be observed not overfitting/underfitting the data.
- We are going to compare with other model to observe whether this model is better.

Hyperparameter Tuning – Gradient Boosting with oversampled data

- Best parameter after tuning with RandomizedSearchCV:

```
Best parameters are {'subsample': 0.7, 'n_estimators': 125, 'max_features': 0.5, 'learning_rate': 1} with CV score=0.9724734023
671514:
Wall time: 8min 54s
```

- Gradient Boosting training performance:

	Accuracy	Recall	Precision	F1
0	0.993	0.993	0.994	0.993

- Gradient Boosting validation performance:

	Accuracy	Recall	Precision	F1
0	0.971	0.845	0.693	0.762

- Gradient Boost model on Recall can be observed not overfitting/underfitting the data.
- We are going to compare with other model to observe whether this model is better.

Model Performance Comparison

- Model comparison for training performance:

	Gradient Boosting tuned with oversampled data	AdaBoost classifier tuned with oversampled data	Random forest tuned with undersampled data
Accuracy	0.993	0.994	0.961
Recall	0.993	0.994	0.933
Precision	0.994	0.995	0.989
F1	0.993	0.994	0.960

- Model comparison for validation performance:

	Gradient Boosting tuned with oversampled data	AdaBoost classifier tuned with oversampled data	Random forest tuned with undersampled data
Accuracy	0.971	0.975	0.938
Recall	0.845	0.860	0.885
Precision	0.693	0.733	0.468
F1	0.762	0.791	0.612

Model Performance Summary

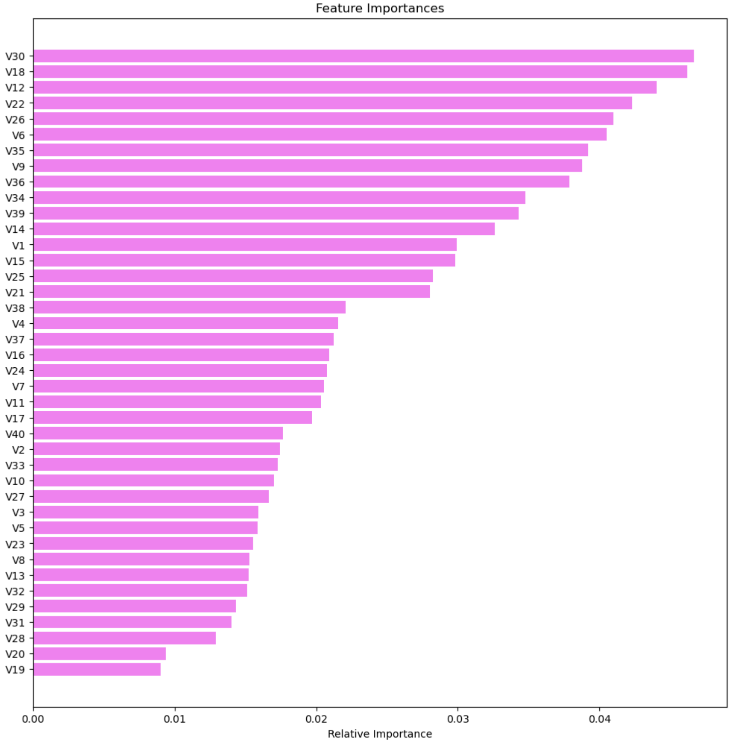
- 3 models were chosen for final comparison, which is Gradient Boost and Adaboost with oversampled data, and Random Forest with undersampled data.
- All samples are improved after hyperparameter tuning.
- Among 3 models, it is observed Adaboost is having good percentage of Recall compared with Gradient Boost.
- Although Random Forest Recall is seen also having higher percentage of Recall by ~ 2% compared with Adaboost, it is observed the percentage of Precision is far too low on its validation performance compared with training performance. Its F1 score also the lowest compared with other 2 models. This may increase some risk during data testing.
- Therefore, among the 3 models, Adaboost is chosen as the final model that may fit on our test data.
- Adaboost will be proceed for our test data and build the pipeline.

Model Performance on Test Data

- Model test performance:

	Accuracy	Recall	Precision	F1
0	0.972	0.844	0.708	0.770

- Feature importance:
  - All features observed having a strong impact on our model.
  - The top 5 feature is V30, V18, V12, V22 and V26.



Productionize and test the final model using pipelines

- We need to the column transformer for building a pipeline. However, this dataset has only one type of data (float64), therefore we do not need to use column transformer.
- We need to declare the pipeline and due to we decided to use AdaBoost on oversampled data, we need to use SMOTE before productionize our model.

```
Pipeline(steps=[('imputer', SimpleImputer(strategy='median')),
                ('AdaBoost',
                 AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3,
                                                                          random_state=1),
                                    learning_rate=1, n_estimators=70,
                                    random_state=1))])
```

- Our final model on the test performance as below:

	Accuracy	Recall	Precision	F1
0	0.971	0.848	0.699	0.766

Model Pipeline Summary

- Comparison between training, validation and test data on Ada Boost is summarized as below:

	AdaBoost training data	AdaBoost validation data	AdaBoost test data	AdaBoost test data using pipeline
Accuracy	0.994	0.975	0.972	0.971
Recall	0.994	0.860	0.844	0.848
Precision	0.995	0.733	0.708	0.699
F1	0.994	0.791	0.770	0.766

- The model is observed performing well on the test data and test data using pipeline. No overfitting or underfitting was detected between training, validation and test data.
- Ada Boost model is a suitable model to deploy for ReneWind project. This model has built to minimize the total maintenance cost on this project machinery/processes.
- The main (top 5) attributes of importance for predicting failure and no failure is V30, V18, V12, V22 and V26.

APPENDIX

Data Info

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20000 entries, 0 to 19999
Data columns (total 41 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   V1      19982 non-null  float64
 1   V2      19982 non-null  float64
 2   V3      20000 non-null  float64
 3   V4      20000 non-null  float64
 4   V5      20000 non-null  float64
 5   V6      20000 non-null  float64
 6   V7      20000 non-null  float64
 7   V8      20000 non-null  float64
 8   V9      20000 non-null  float64
 9   V10     20000 non-null  float64
 10  V11     20000 non-null  float64
 11  V12     20000 non-null  float64
 12  V13     20000 non-null  float64
 13  V14     20000 non-null  float64
 14  V15     20000 non-null  float64
 15  V16     20000 non-null  float64
 16  V17     20000 non-null  float64
 17  V18     20000 non-null  float64
 18  V19     20000 non-null  float64
 19  V20     20000 non-null  float64
 20  V21     20000 non-null  float64
 21  V22     20000 non-null  float64
 22  V23     20000 non-null  float64
 23  V24     20000 non-null  float64
 24  V25     20000 non-null  float64
 25  V26     20000 non-null  float64
 26  V27     20000 non-null  float64
 27  V28     20000 non-null  float64
 28  V29     20000 non-null  float64
 29  V30     20000 non-null  float64
 30  V31     20000 non-null  float64
 31  V32     20000 non-null  float64
 32  V33     20000 non-null  float64
 33  V34     20000 non-null  float64
 34  V35     20000 non-null  float64
 35  V36     20000 non-null  float64
 36  V37     20000 non-null  float64
 37  V38     20000 non-null  float64
 38  V39     20000 non-null  float64
 39  V40     20000 non-null  float64
 40  Target  20000 non-null  int64  
dtypes: float64(40), int64(1)
memory usage: 6.3 MB
```

Missing Value before Pre-Treatment

- Train data

```
V1        18    V26       0
V2        18    V27       0
V3         0    V28       0
V4         0    V29       0
V5         0    V30       0
V6         0    V31       0
V7         0    V32       0
V8         0    V33       0
V9         0    V34       0
V10        0    V35       0
V11        0    V36       0
V12        0    V37       0
V13        0    V38       0
V14        0    V39       0
V15        0    V40       0
V16        0    Target    0
V17        0    dtype: int64
V18        0
V19        0
V20        0
V21        0
V22        0
V23        0
V24        0
V25        0
```

- Test data

```
V1         5    V26       0
V2         6    V27       0
V3         0    V28       0
V4         0    V29       0
V5         0    V30       0
V6         0    V31       0
V7         0    V32       0
V8         0    V33       0
V9         0    V34       0
V10        0    V35       0
V11        0    V36       0
V12        0    V37       0
V13        0    V38       0
V14        0    V39       0
V15        0    V40       0
V16        0    Target    0
V17        0    dtype: int64
V18        0
V19        0
V20        0
V21        0
V22        0
V23        0
V24        0
V25        0
```

Missing Value after Imputation

- Train data

```
V1     0
V2     0
V3     0
V4     0
V5     0
V6     0
V7     0
V8     0
V9     0
V10    0
V11    0
V12    0
V13    0
V14    0
V15    0
V16    0
V17    0
V18    0
V19    0
V20    0
V21    0
V22    0
V23    0
V24    0
V25    0
V26    0
V27    0
V28    0
V29    0
V30    0
V31    0
V32    0
V33    0
V34    0
V35    0
V36    0
V37    0
V38    0
V39    0
V40    0
dtype: int64
-------------
```

- Validation data

```
V1     0
V2     0
V3     0
V4     0
V5     0
V6     0
V7     0
V8     0
V9     0
V10    0
V11    0
V12    0
V13    0
V14    0
V15    0
V16    0
V17    0
V18    0
V19    0
V20    0
V21    0
V22    0
V23    0
V24    0
V25    0
V26    0
V27    0
V28    0
V29    0
V30    0
V31    0
V32    0
V33    0
V34    0
V35    0
V36    0
V37    0
V38    0
V39    0
V40    0
dtype: int64
--------------
```

- Test data

```
V1     0
V2     0
V3     0
V4     0
V5     0
V6     0
V7     0
V8     0
V9     0
V10    0
V11    0
V12    0
V13    0
V14    0
V15    0
V16    0
V17    0
V18    0
V19    0
V20    0
V21    0
V22    0
V23    0
V24    0
V25    0
V26    0
V27    0
V28    0
V29    0
V30    0
V31    0
V32    0
V33    0
V34    0
V35    0
V36    0
V37    0
V38    0
V39    0
V40    0
dtype: int64
```

Great Learning

Happy Learning !