ReneWind

Model Tuning

02-24-2023

Contents / Agenda

Executive Summary

- We have been able to build a predictive model that can be used by the company to identify failures so that the generators could be repaired before failing/breaking to reduce the overall maintenance cost with recall score of 0.88 and 0.87 on both the validation and test sets and formulate corrective actions accordingly.
- The model built can be used to predict the generator failure and can correctly identify 88% of the machine failures.
- V36, V18, V39, V15 and V3 are the most important variables in predicting whether the generator will be failed or not, so the company should thoroughly and timely inspect these variables for the predictive maintenance.

Business Problem Overview and Solution Approach

- **Business problem overview**
- ReneWind is a company working on improving the machinery/processes involved in the production of wind energy. The company planned to achieve operational efficiency using predictive maintenance practices, this calls for a machine learning based solution 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.
- **Solution Approach/ methodology**
- Overview of the data
- Review the patterns and trends of the data using EDA
- Data Pre processing
- Model performance summary for hyperparameter tuning.
- Model building with pipeline

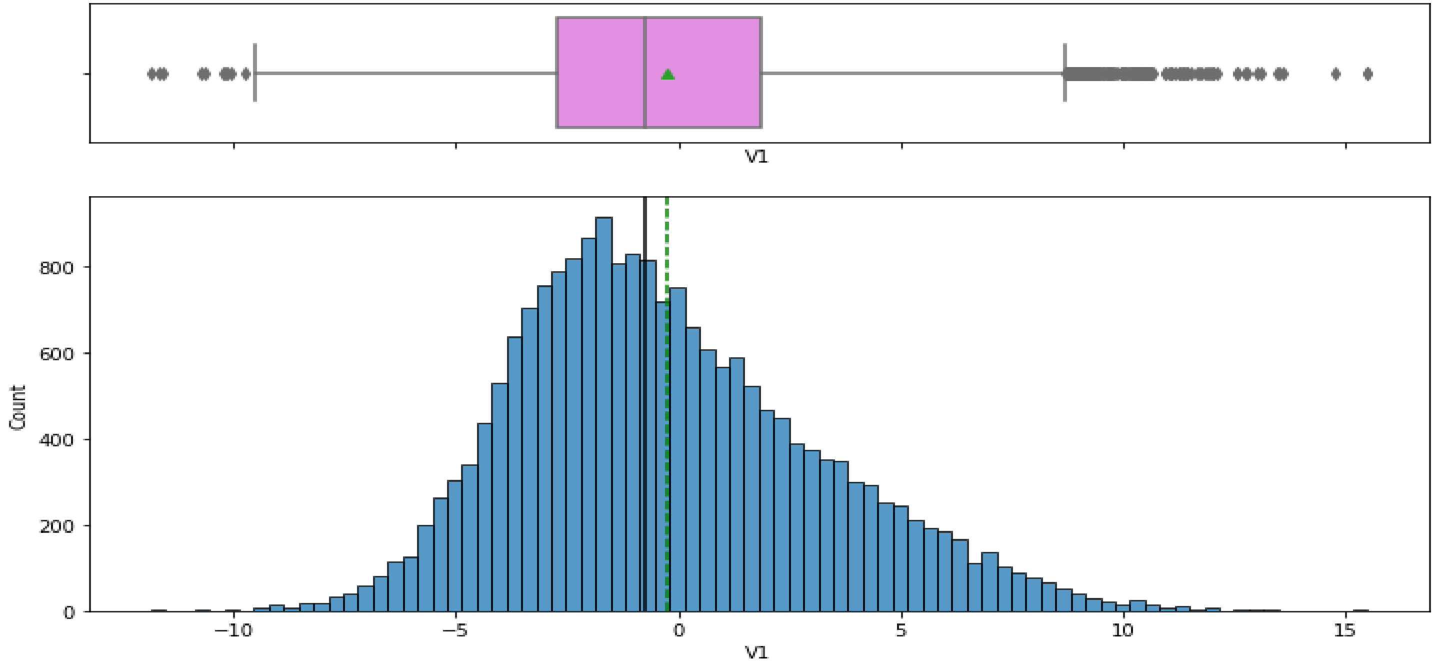
EDA Results

- The data set has 25000 observations and 41 Attributes.
- All of the variables are numeric type variables.
- There are outliers on all attributes.

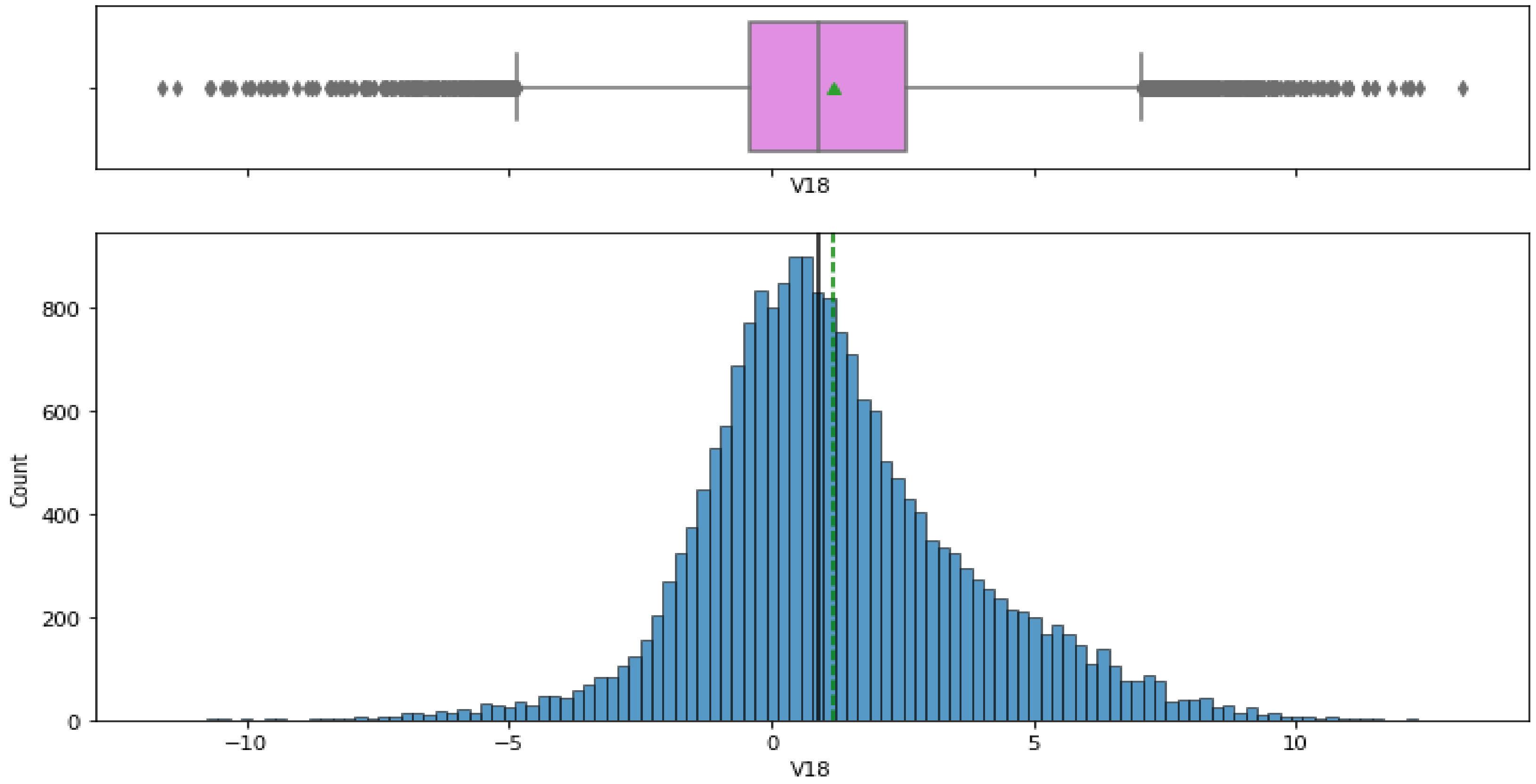
Link to Appendix slide on data background check

EDA Results

- The distribution for V1 and V16 is highly skewed to the right.



EDA Results



Data Preprocessing

- There is no duplicate value in the data.
- There are missing values on V1 and V2 variables. We treat them by their median value.
- There are outliers on all the attributes. However, we will not treat them as they are proper values.
- We Split the data into train and Validation in the ratio 75:25 to be able to evaluate the model that we build on the train data.

***Note**: You can use more than one slide if needed*

Model Performance Summary

Model Performance Summary (original data)

	Cross Validation Performance	
Models	Training	Validation
Logistic regression	0.493	0.482
Decision Tree	0.698	0.705
Bagging	0.721	0.73
Random Forest	0.724	0.727
AdaBoost	0.631	0.676
Gradient Boosting	0.707	0.723

- From the above table we can see that Bagging is giving the highest cross-validated recall followed by random forest.

Link to Appendix slide on model assumptions

Model Performance Summary (oversampled data)

- We have choosen SMOTE(Synthetic Minority oversampling Technique) to oversample the data.

	Cross Validation Performance	
Models	Training	Validation
Logistic regression	0.884	0.849
Decision Tree	0.972	0.777
Bagging	0.976	0.835
Random Forest	0.984	0.849
AdaBoost	0.898	0.856
Gradient Boosting	0.926	0.878

- We can see that Gradient boosting is giving the highest cross-validated recall followed by Ada boosting.

Model Performance Summary (undersampled data)

- We have Using randomundersampler method to undersample the data.

	Cross Validation Performance	
Models	Training	Validation
Logistic regression	0.873	0.853
Decision Tree	0.862	0.842
Bagging	0.864	0.871
Random Forest	0.904	0.892
AdaBoost	0.867	0.849
Gradient Boosting	0.899	0.888

- We can see that Random forest is giving the highest cross-validated recall followed by Gradient boosting.

Model Performance – Hyper parameter Tuning models

- The random forest on undersample data, Adaboost and gradient boosting on over sample data has given a generalized performance with the highest recall on validation data, so we will tune them to improve their performance and to select the final model.
- To save time taken to for running, I'm using Randomized searchCV hyperparameter tuning method.
- The best parameters to tune the Adaboost model on oversample data are : n_estimator=200, learning rate=0.2 and base estimator= DecisionTreeClassifier(max_depth=3, random_state=1)
- The best parameters to tune the Random Forest model on undersample data are : n_estimator=300, minimum sample leaf=2, max samples = 0.5 and max features = 'sqrt'.
- The best parameters to tune the Gradient Boosting model on oversample data are : sub sample=0.7, n_estimators= 125, max feature = 0.5 and learning rate=1

Model Performance Summary – Tuned models

- Training performance comparison

	Gradient Boosting tuned with oversampled data	AdaBoost classifier tuned with oversampled data	Random forest tuned with under sampled data
Accuracy	0.993	0.992	0.961
Recall	0.992	0.988	0.933
Precision	0.994	0.995	0.989
F1	0.993	0.992	0.96

- Validation Performance Comparison

	Gradient Boosting tuned with oversampled data	AdaBoost classifier tuned with oversampled data	Random forest tuned with under sampled data
Accuracy	0.969	0.979	0.938
Recall	0.856	0.849	0.885
Precision	0.678	0.789	0.468
F1	0.757	0.818	0.612

- The Tuned random forest model on undersample data has given a generalized performance with the highest recall on validation data, so we will consider it as our final model.

Model Performances

- The performance on the test data for the tuned random forest model on under sample data is generalized.

Accuracy	Recall	Precision	F1
0.944	0.879	0.5	0.638

- The variables V36, V18, are the most important features, followed by V39, V15 and V3.

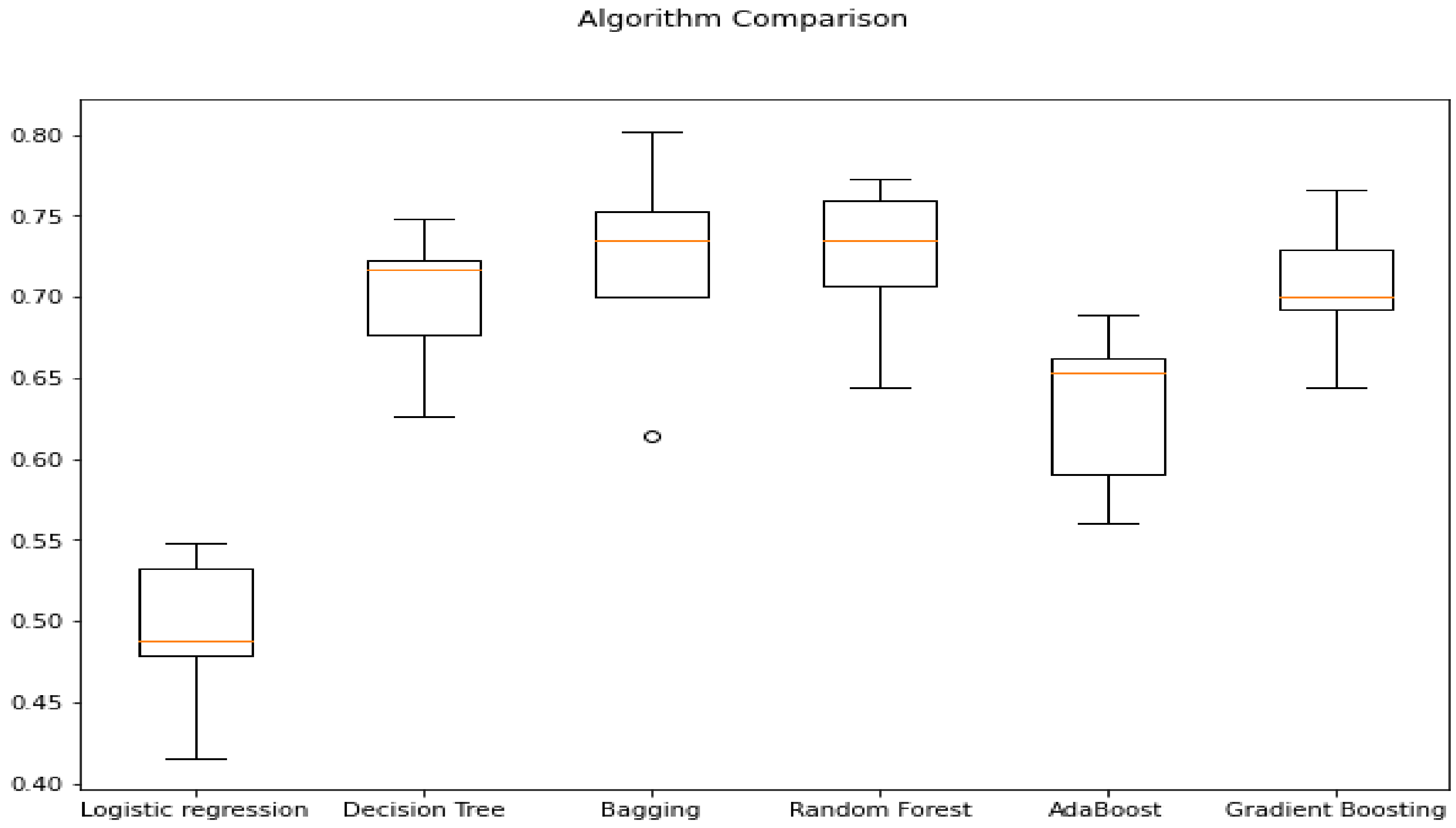
Productionize and test the final model using pipelines

- **Steps taken to create a pipeline for the final model.**
- Since we have only one datatype in the data, we don't need to use column transformer here.
- Create pipeline for the random forest model.
- Separate the target variable and other variables, we don’t need to divide data into train and test as we already have a separate test set.
- Treating the missing value on train set before we undersample the data. We don’t need to impute missing values in the test set as it will be done inside the pipeline.
- Perform undersampling as the best model is built on the undersample data.
- Fit the model obtained from the above steps
- Check performance on the test set.

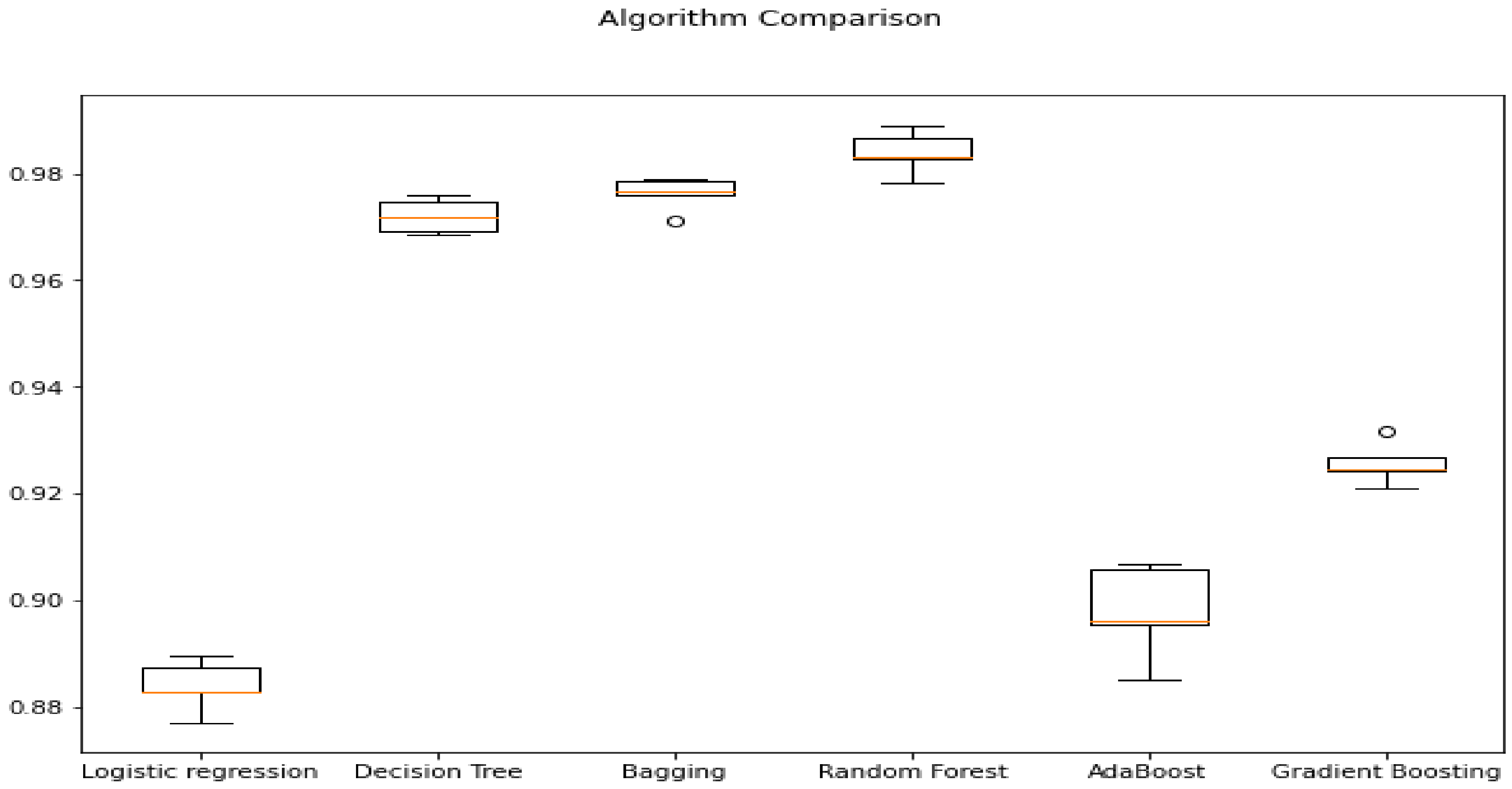
APPENDIX

Data Back ground and contents

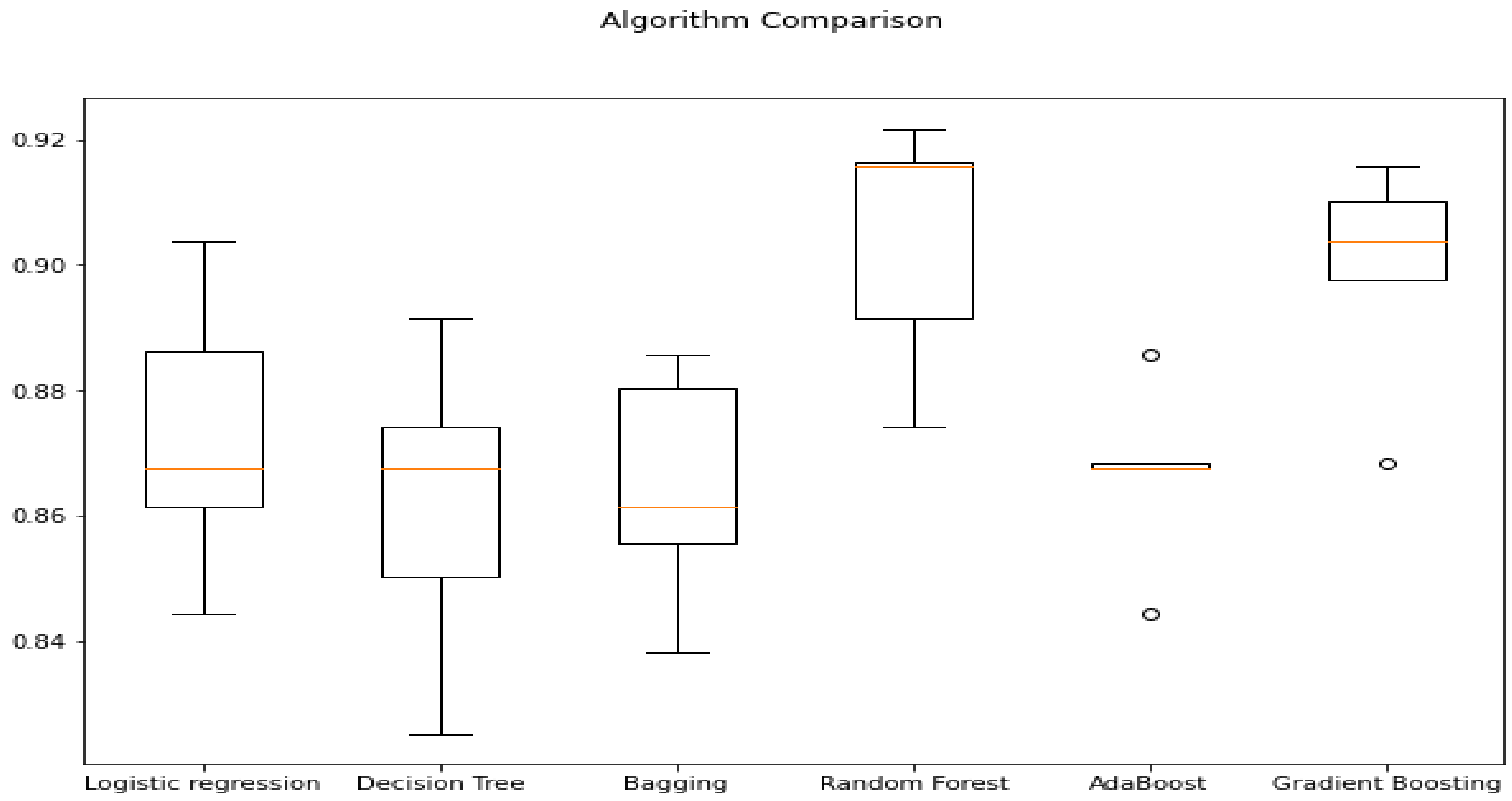
Original data – Algorithm Comparison



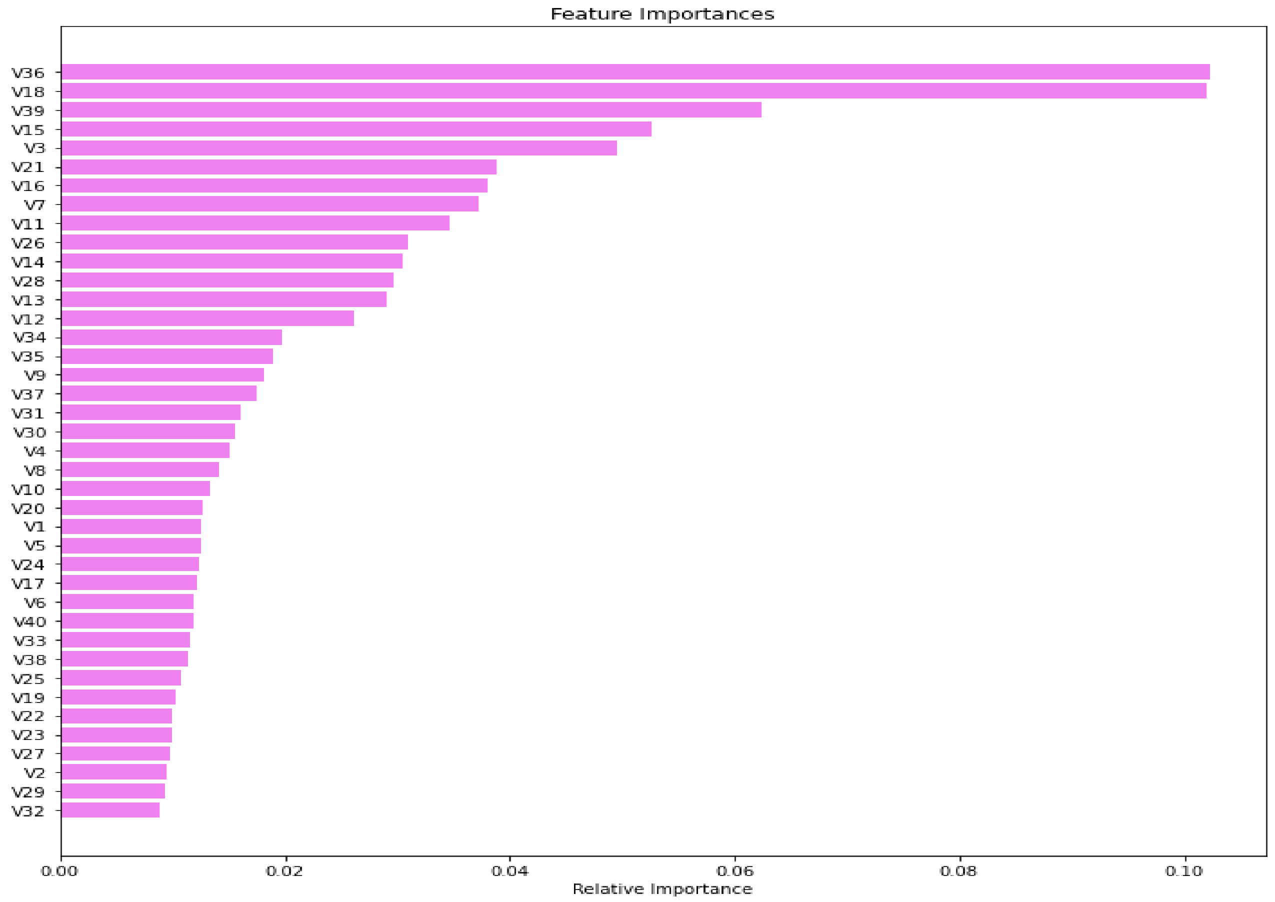
Over sampled data – Algorithm Comparison

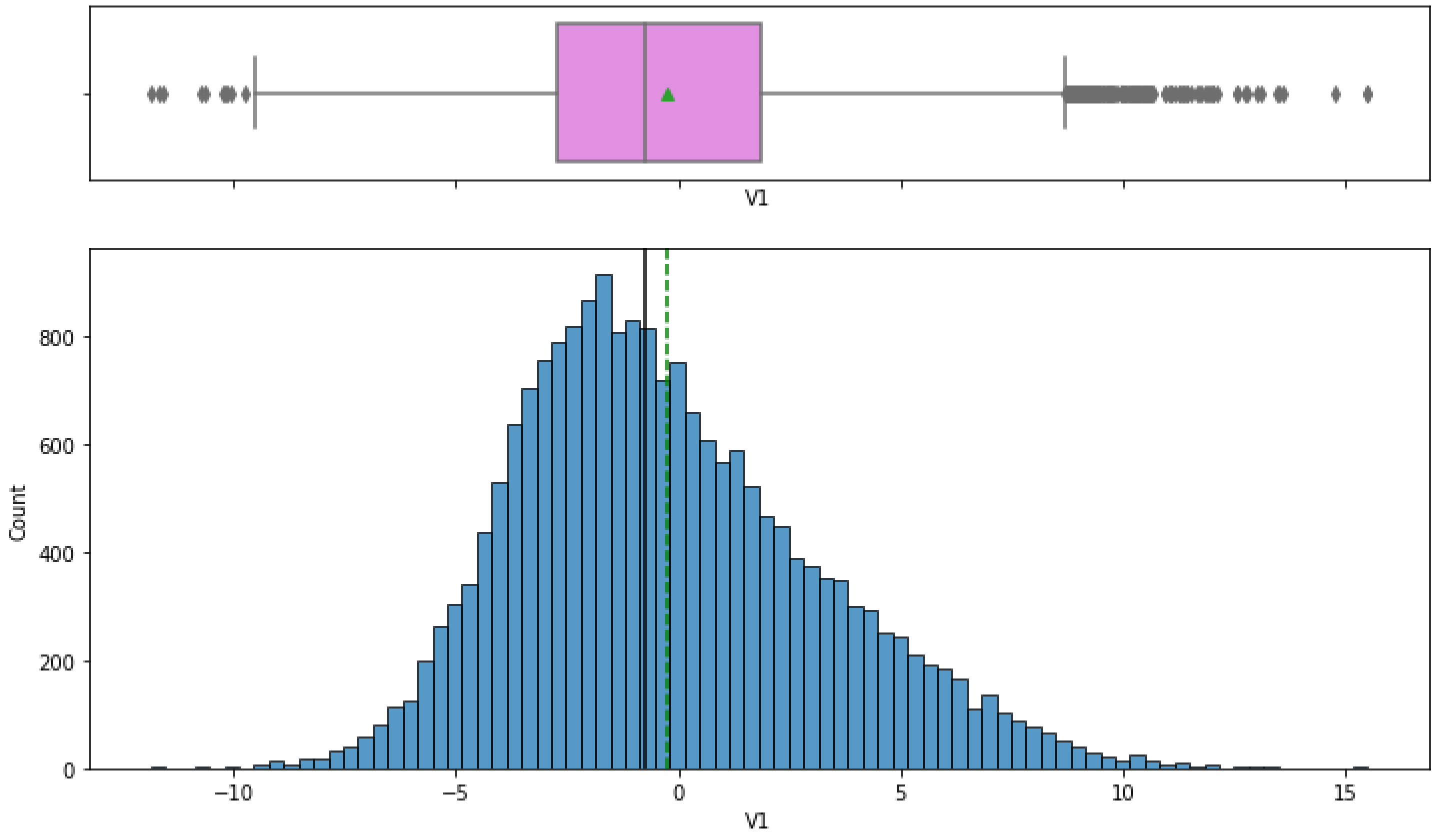


Under sampled data – Algorithm Comparison

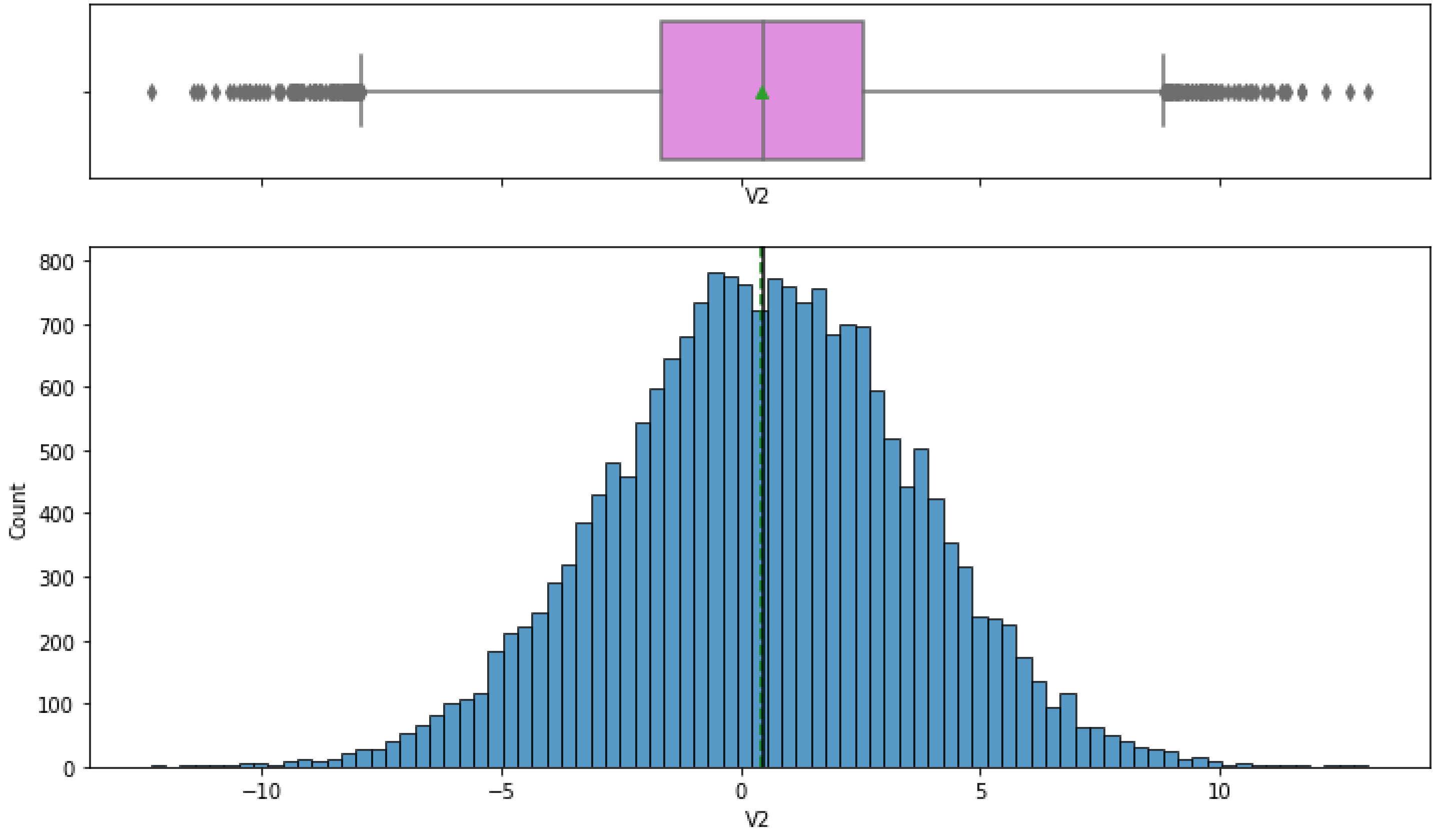


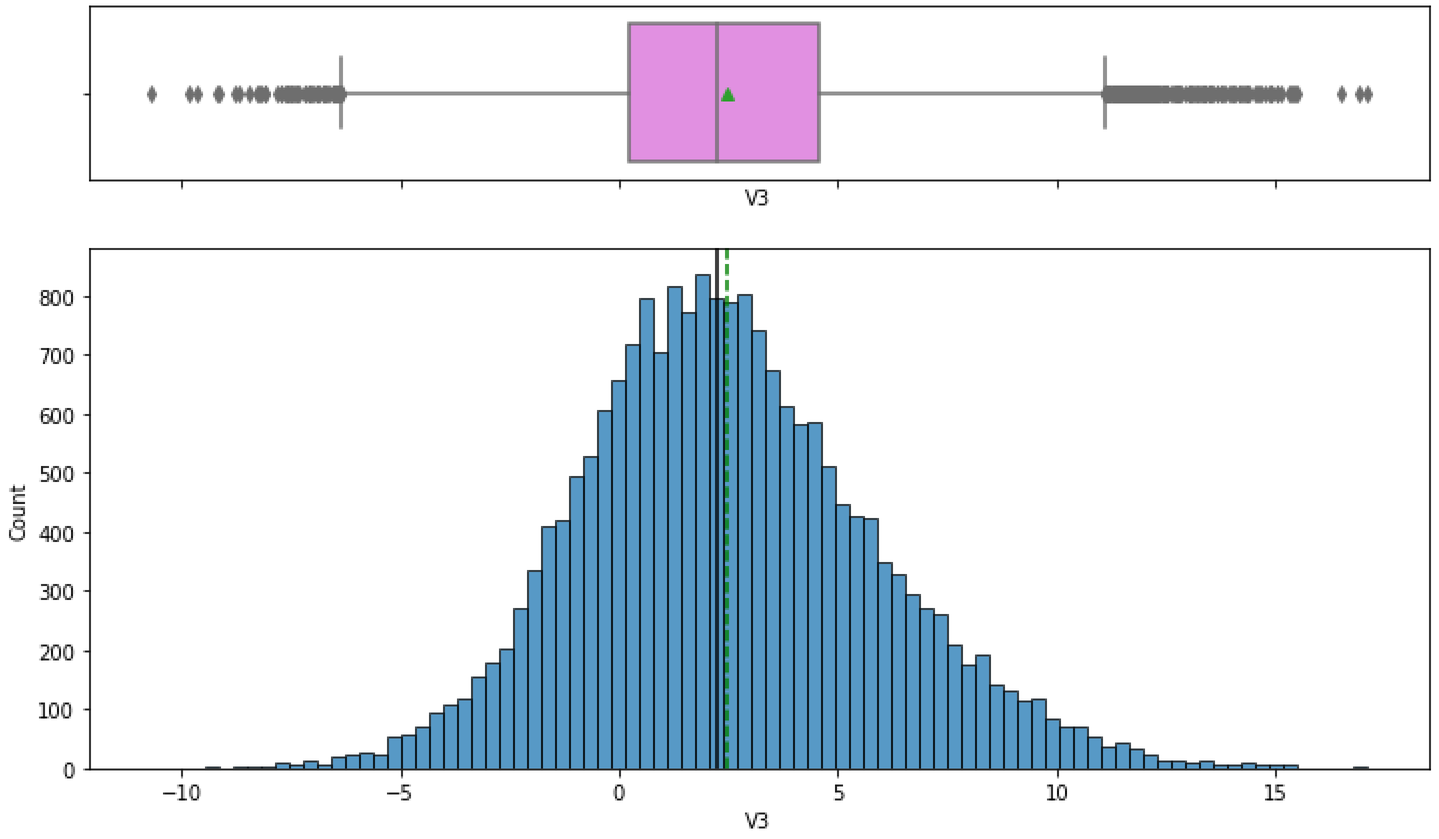
Feature importance's for the final model

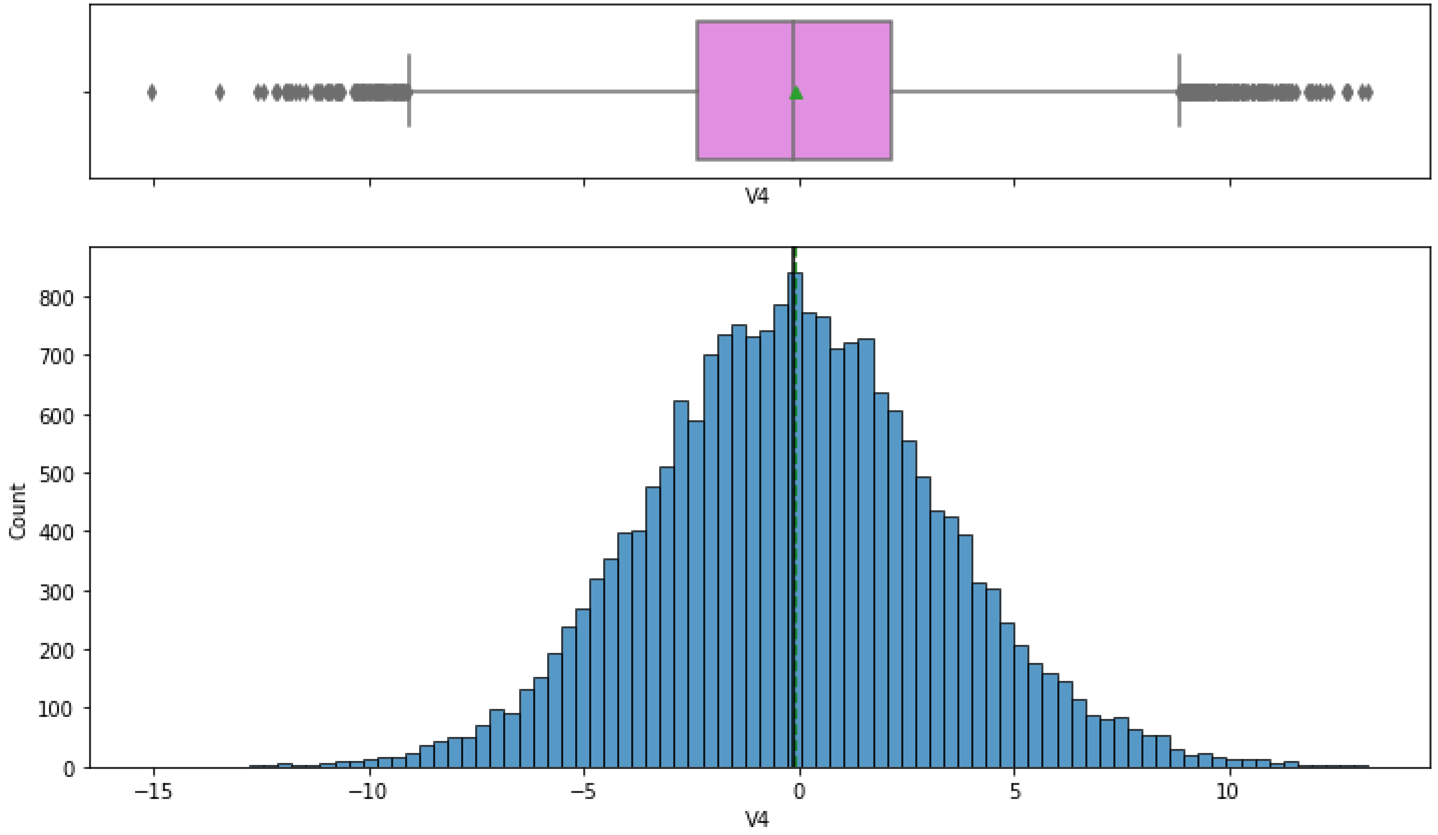


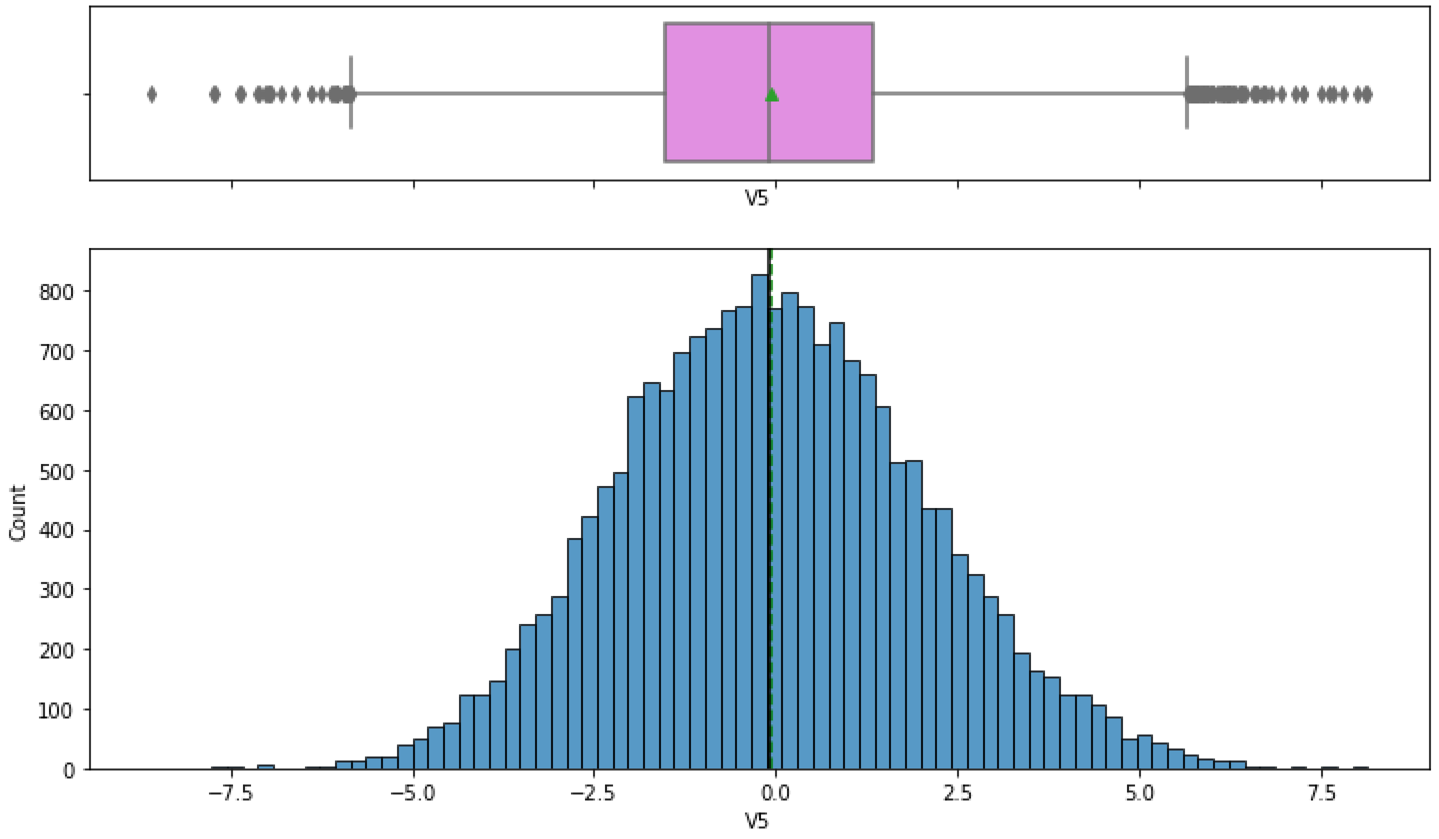


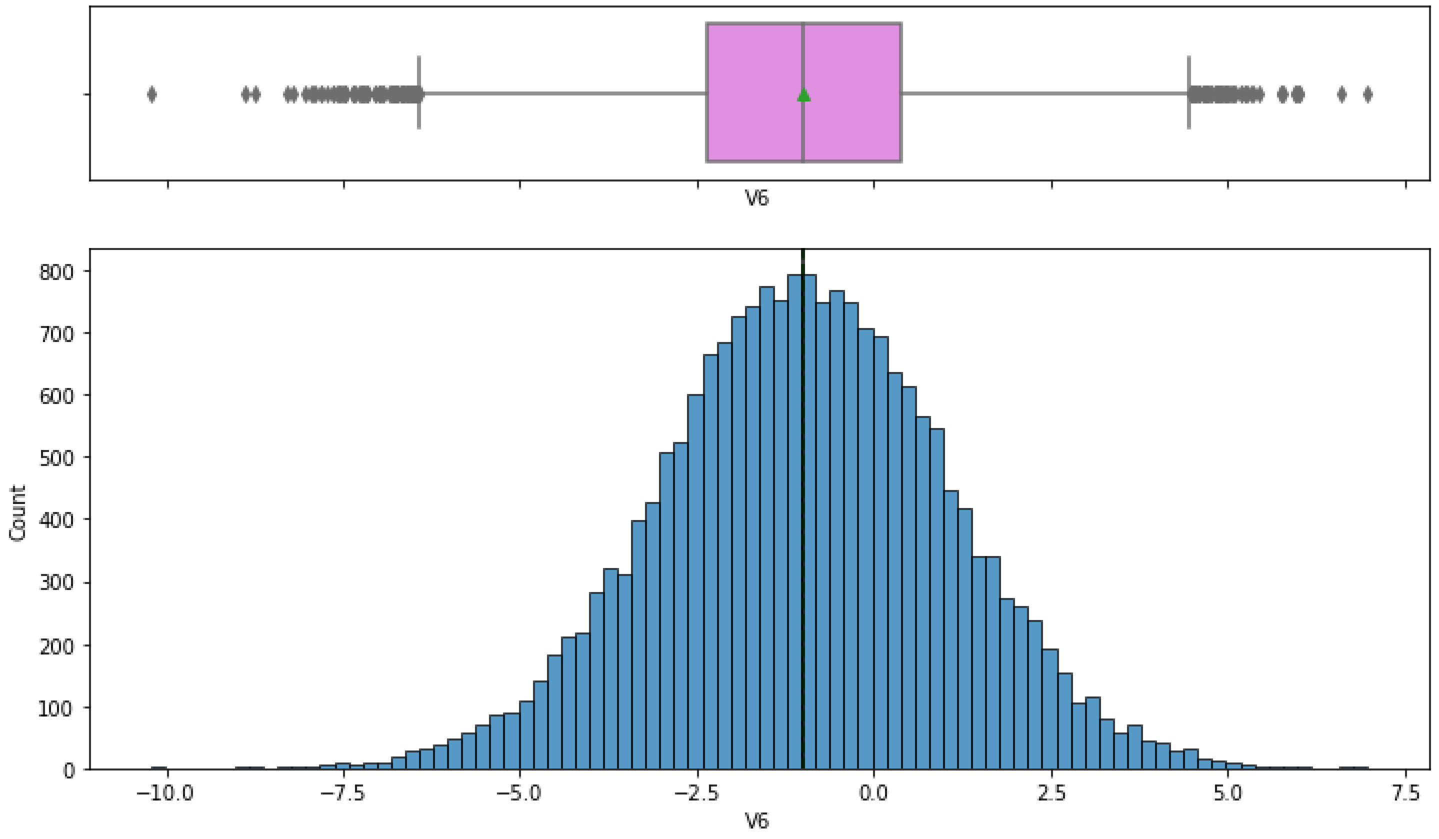
V1
800
600
400
200
0
Count
−10
−5
0
5
10
15
V1

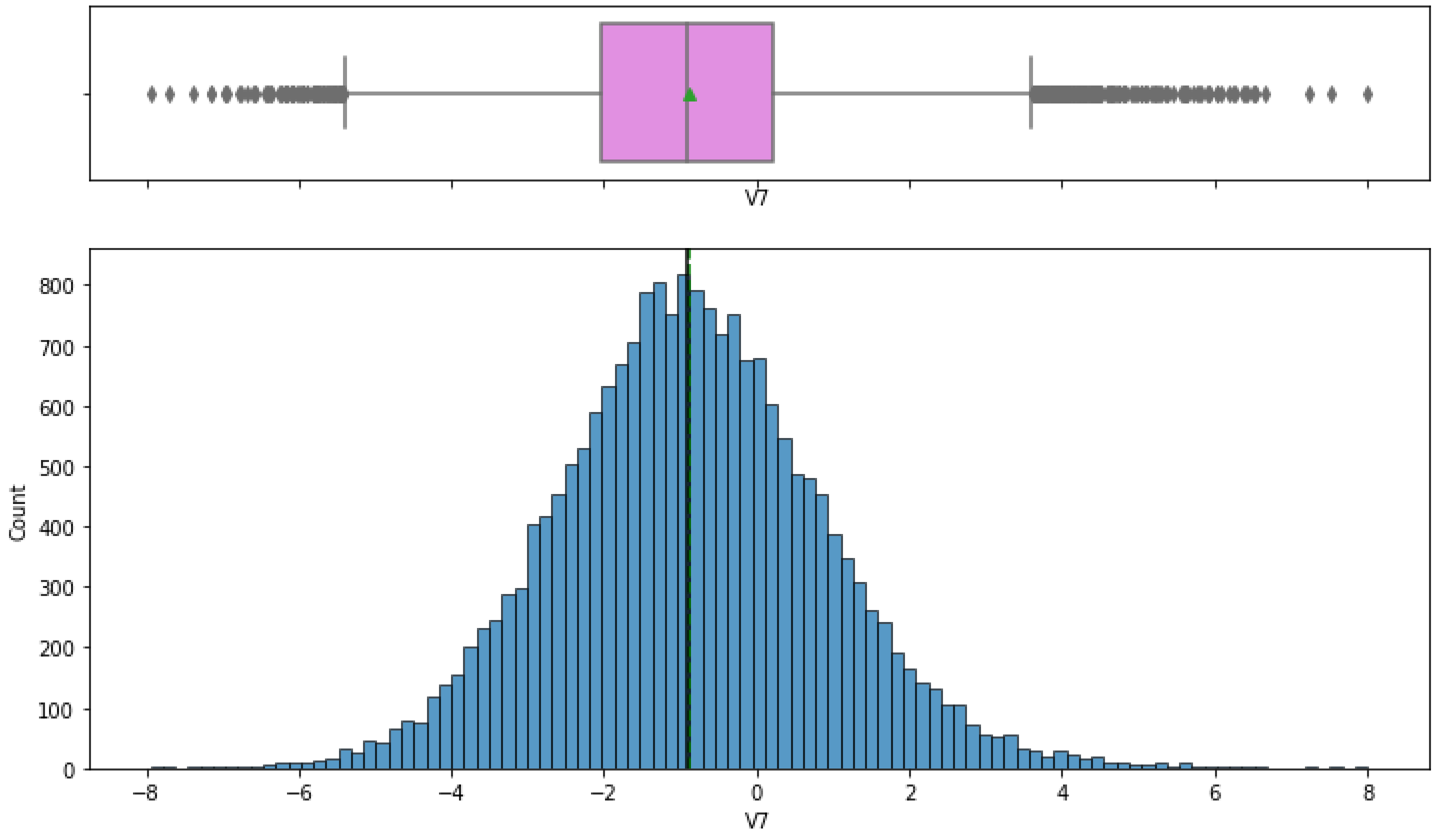

V2
800
700
600
500
400
300
200
100
0
Count
−10
−5
0
5
10
V2

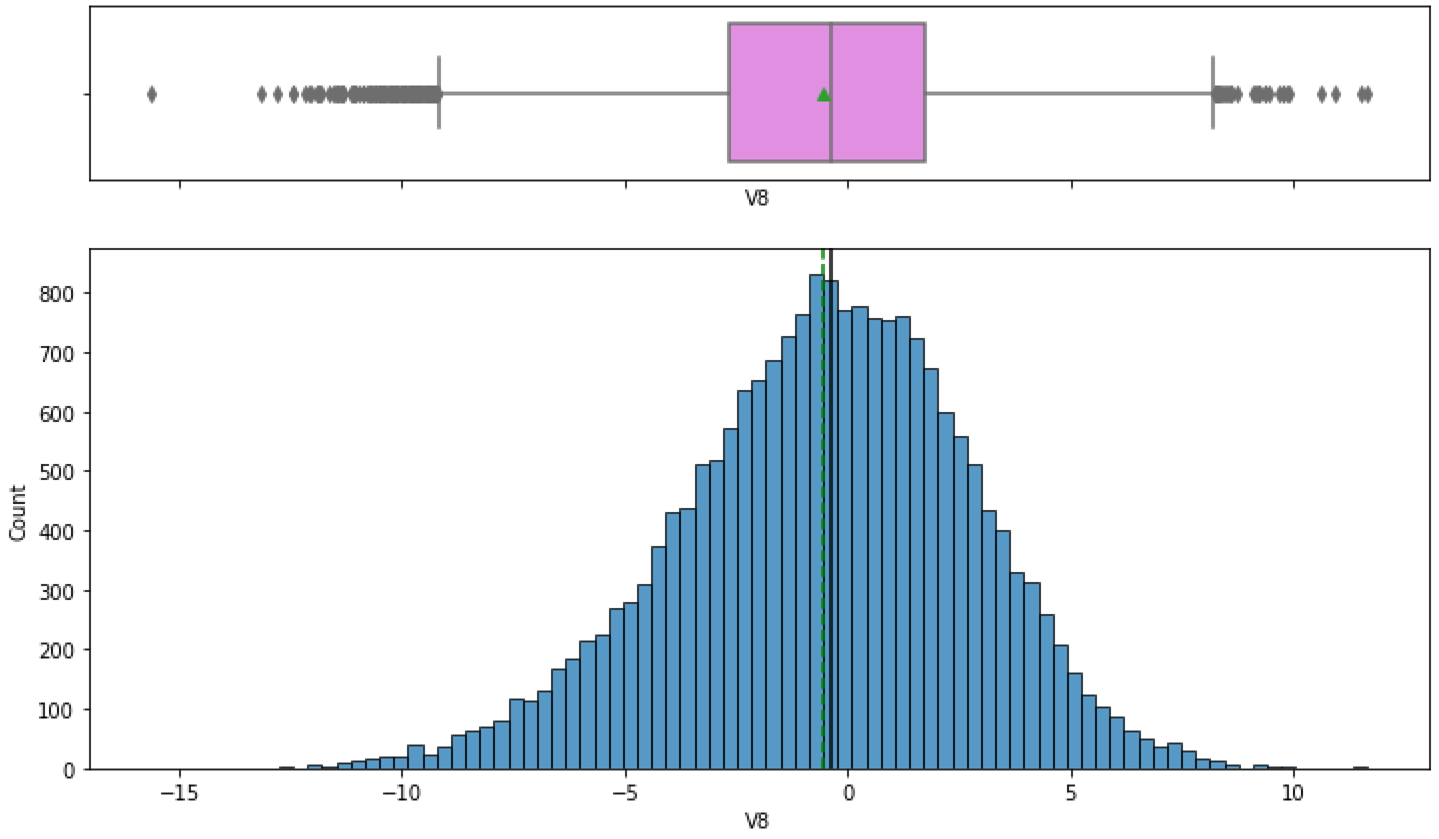

V3
Count
800
700
600
500
400
300
200
100
0
−10
−5
0
5
10
15
V3

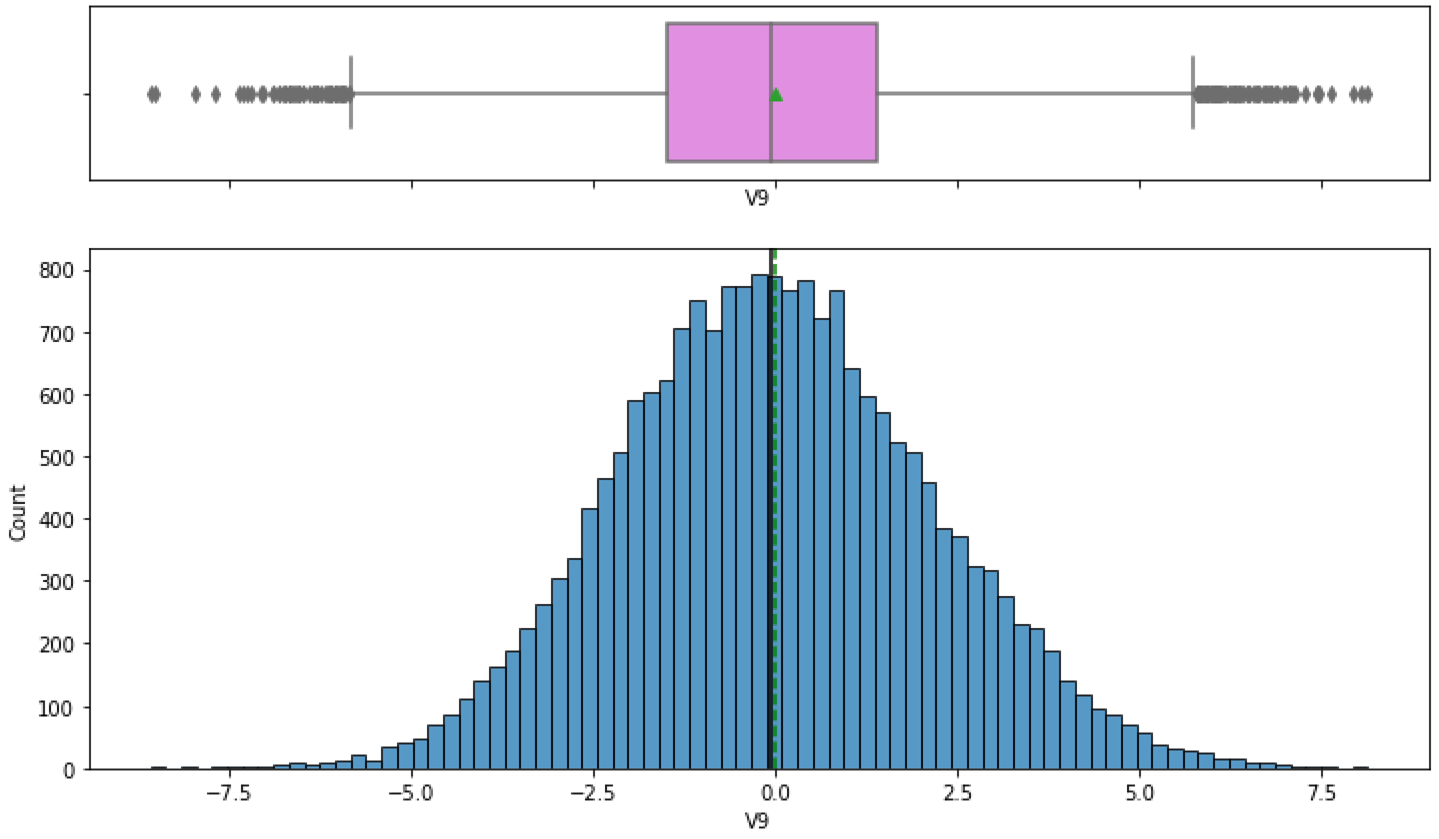

V4
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
V4

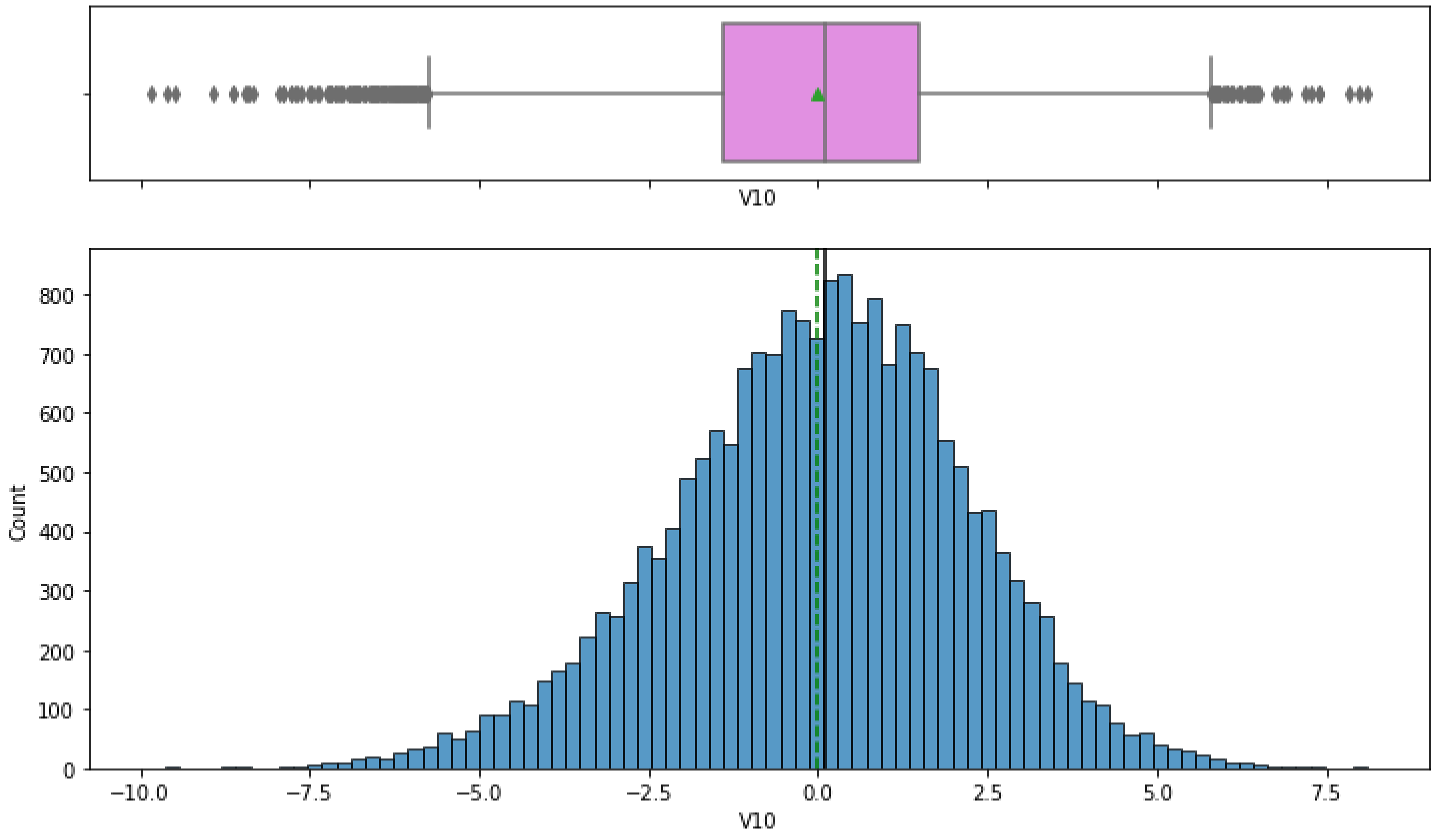

V5
800
700
600
500
400
300
200
100
0
Count
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V5

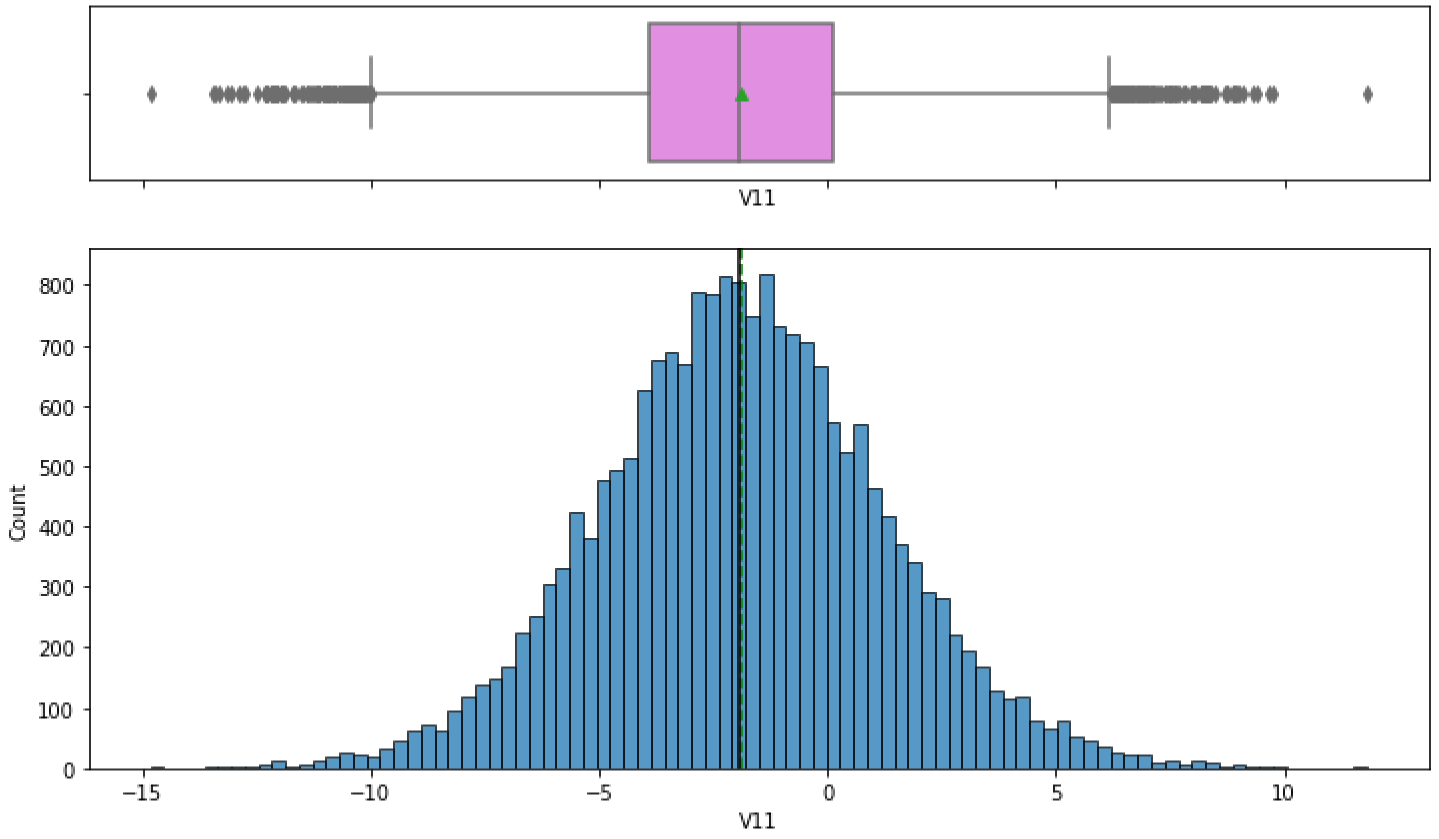

V6
800
700
600
500
400
300
200
100
0
Count
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V6

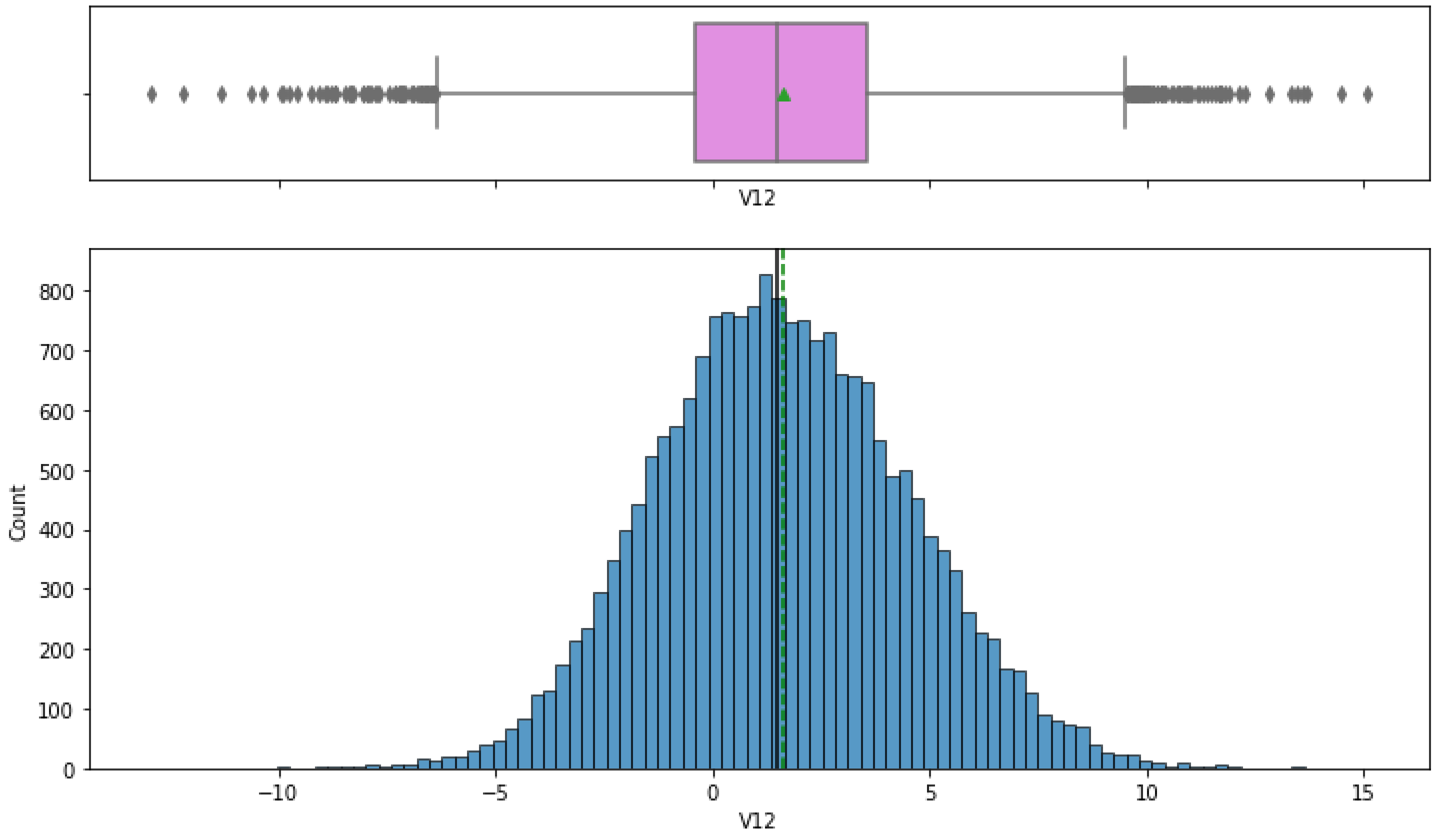

V7
800
700
600
500
400
300
200
100
0
Count
−8
−6
−4
−2
0
2
4
6
8
V7

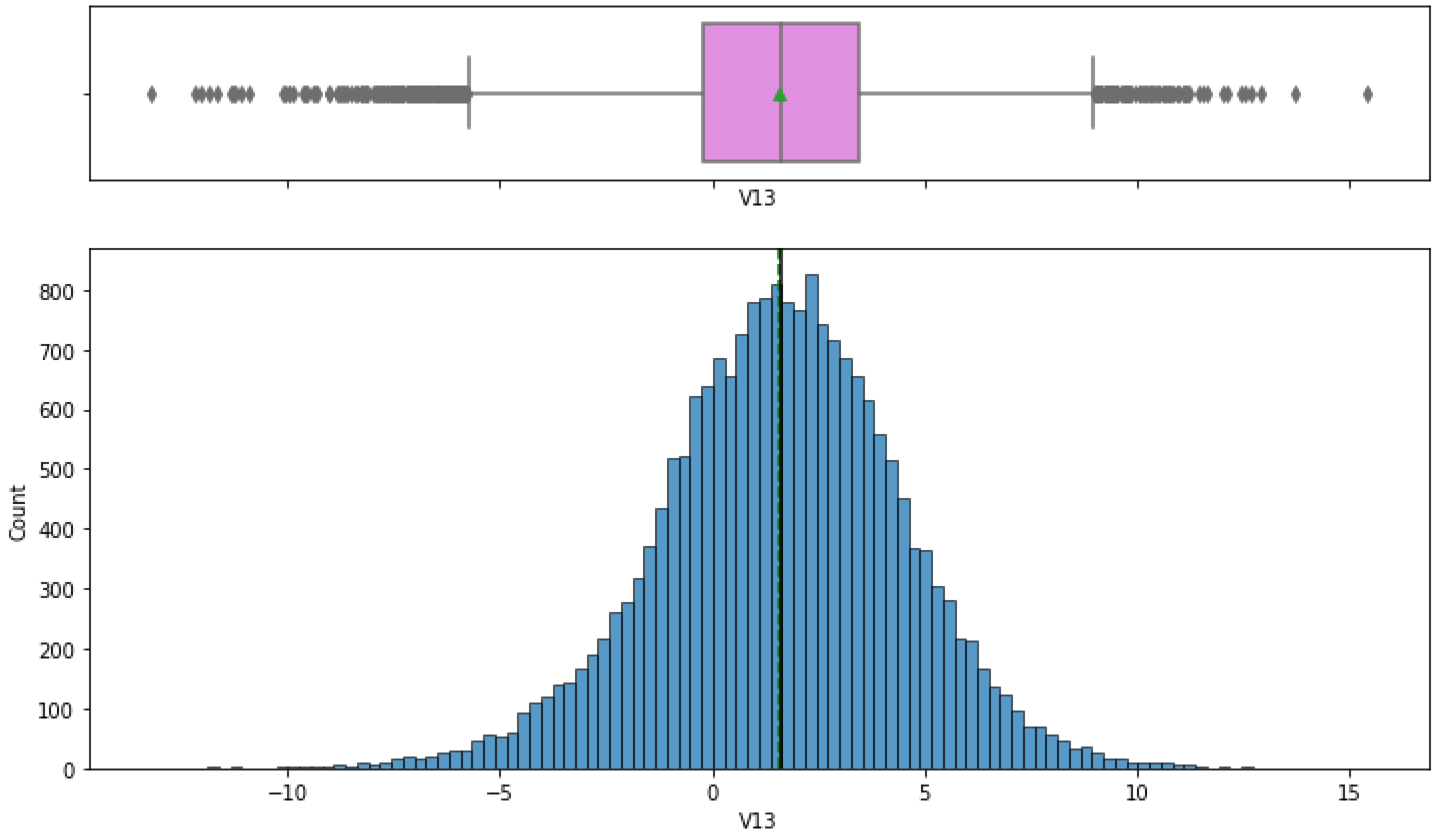

V8
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
V8

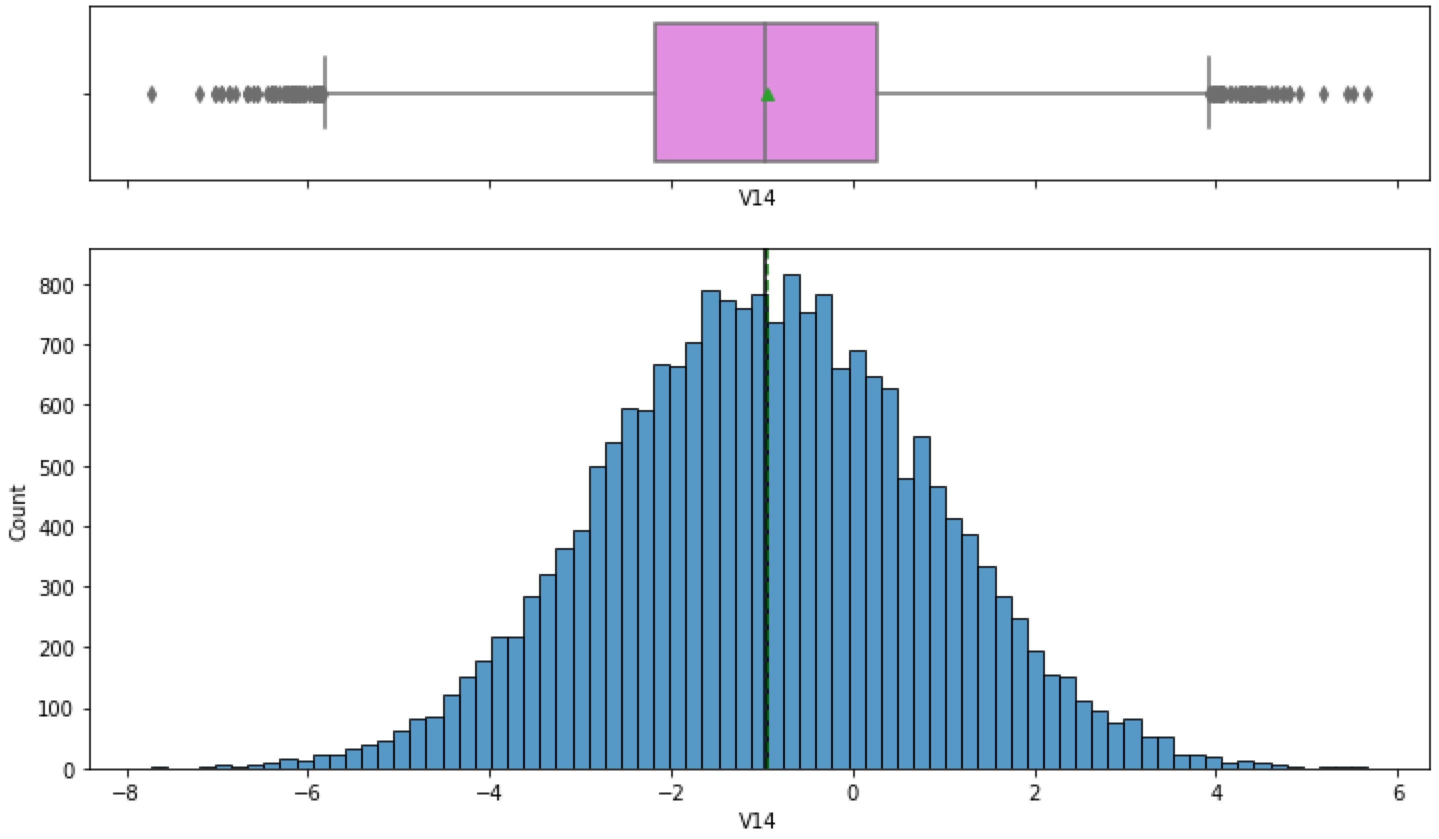

V9
Count
800
700
600
500
400
300
200
100
0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V9

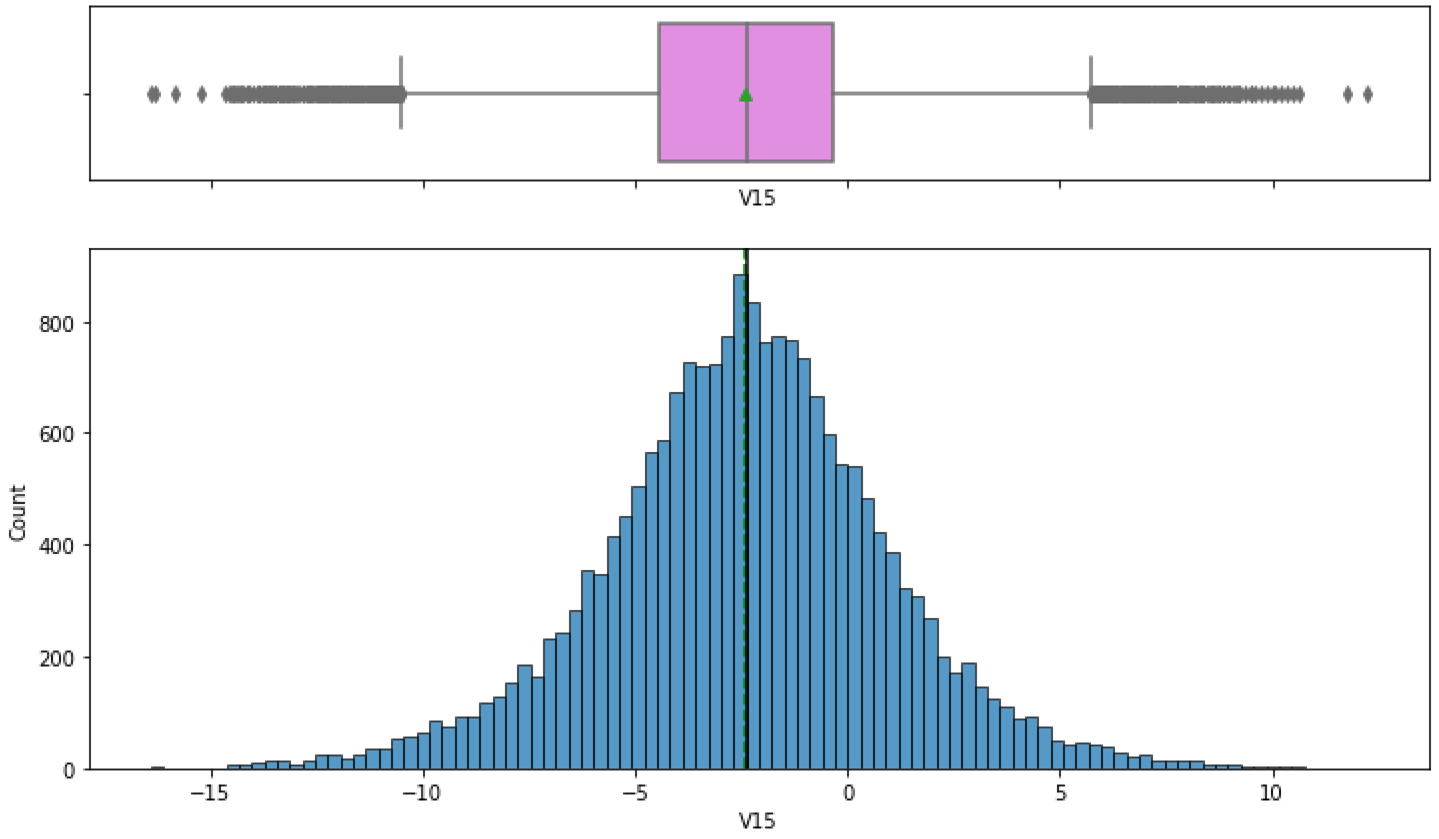

V10
Count
800
700
600
500
400
300
200
100
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V10

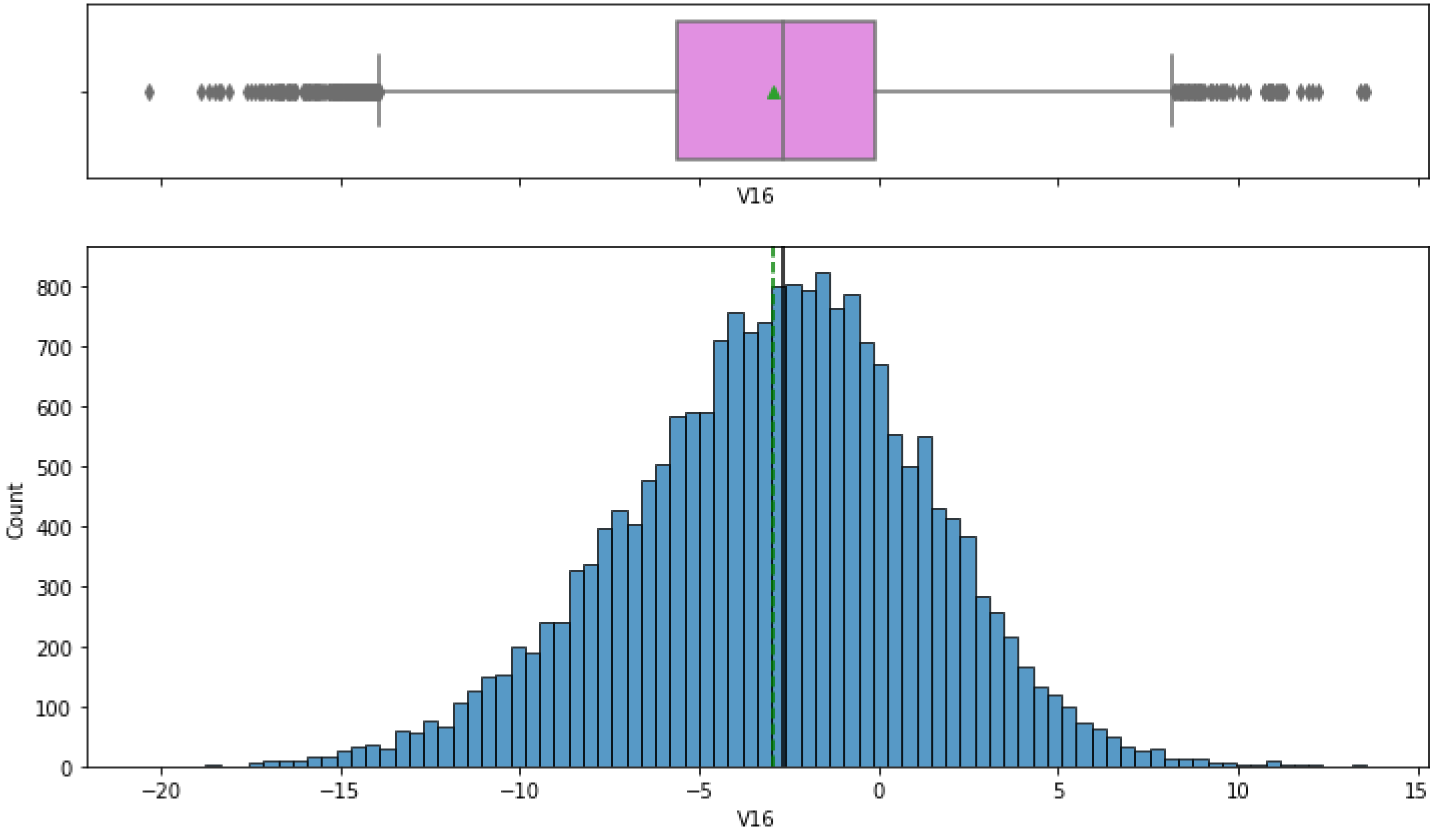

V11
800
700
600
500
400
300
200
100
0
Count
−15
−10
−5
0
5
10
V11

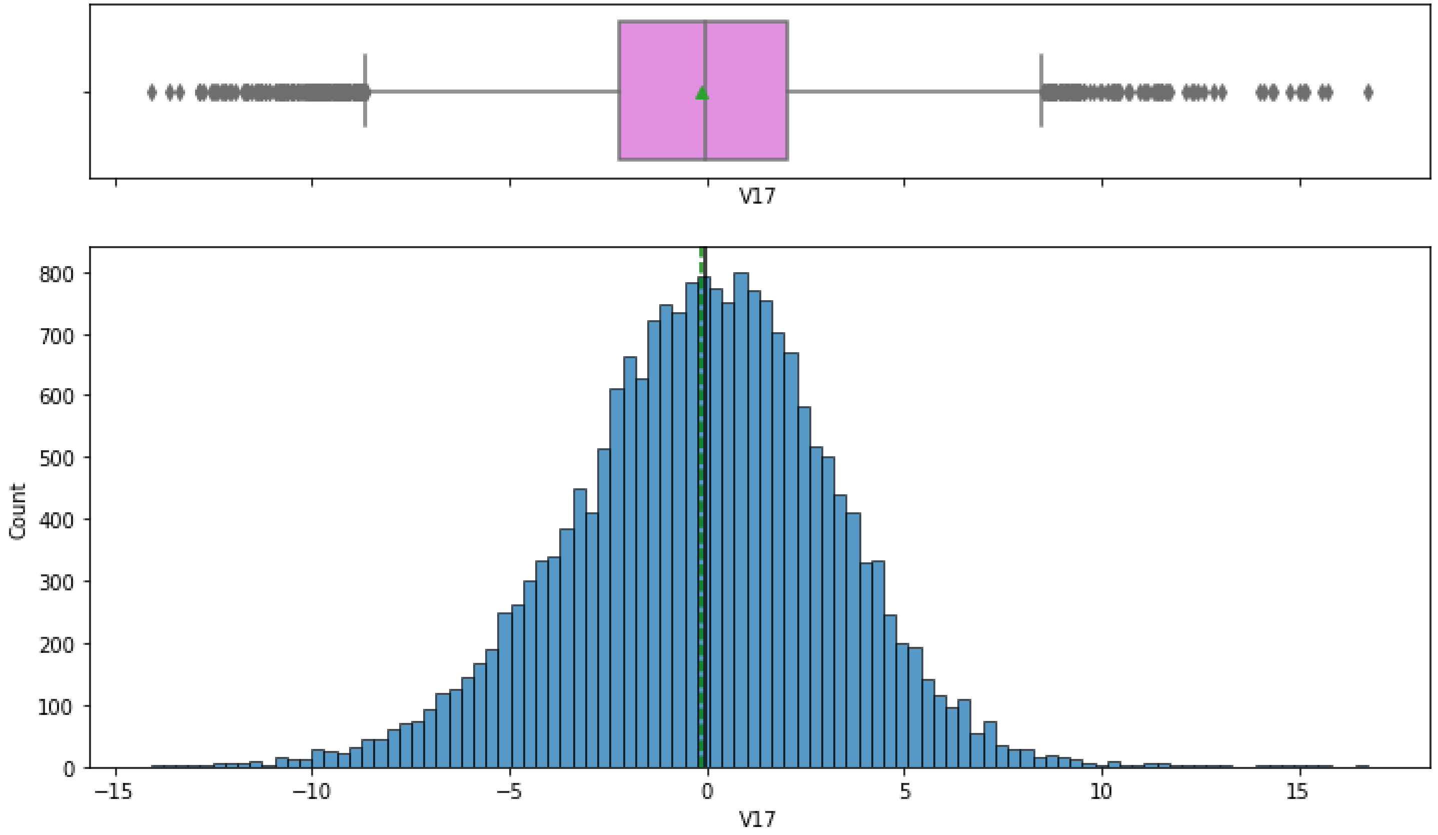

V12
Count
800
700
600
500
400
300
200
100
0
−10
−5
0
5
10
15
V12

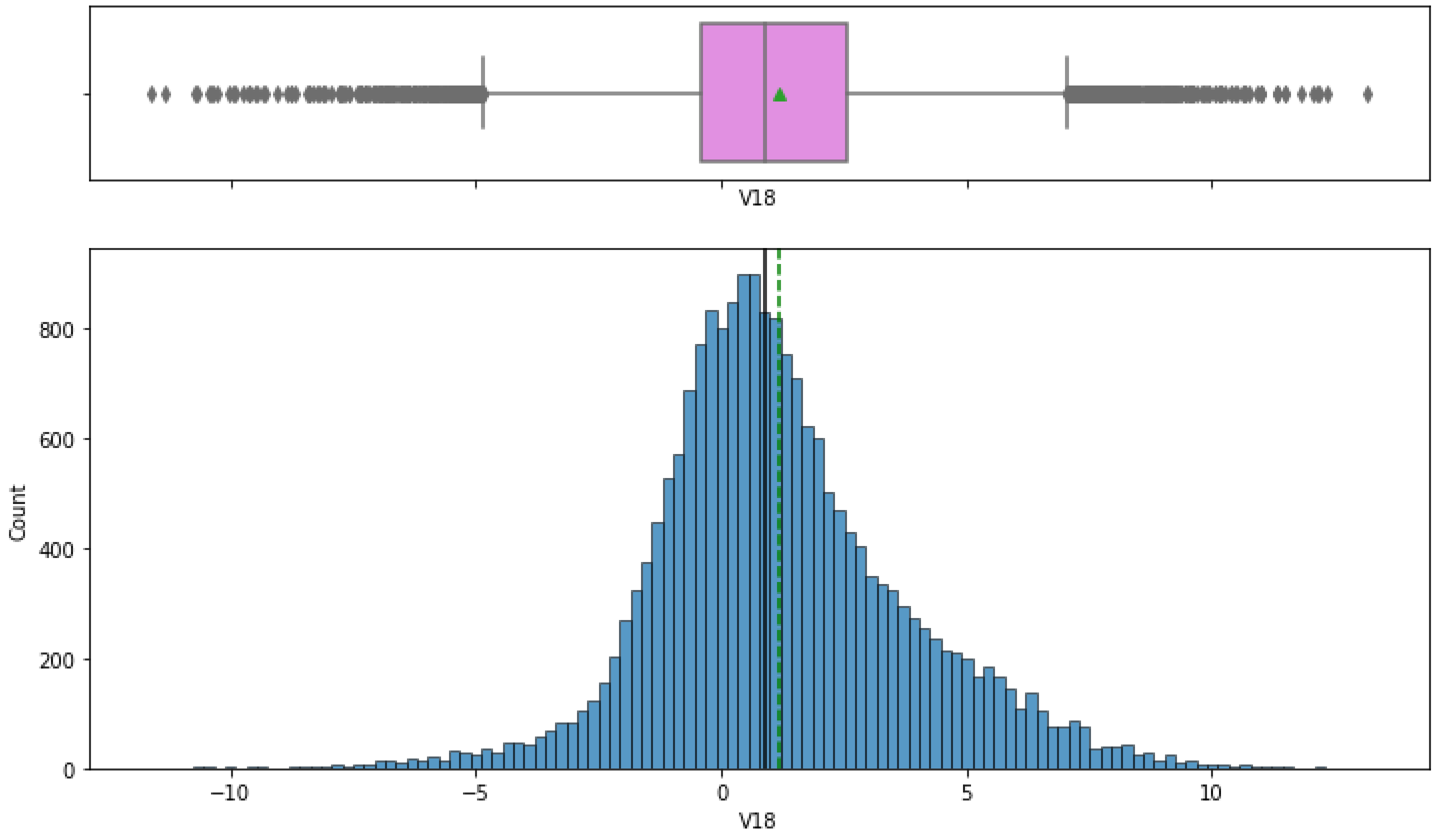

V13
800
700
600
500
400
300
200
100
0
Count
−10
−5
0
5
10
15
V13

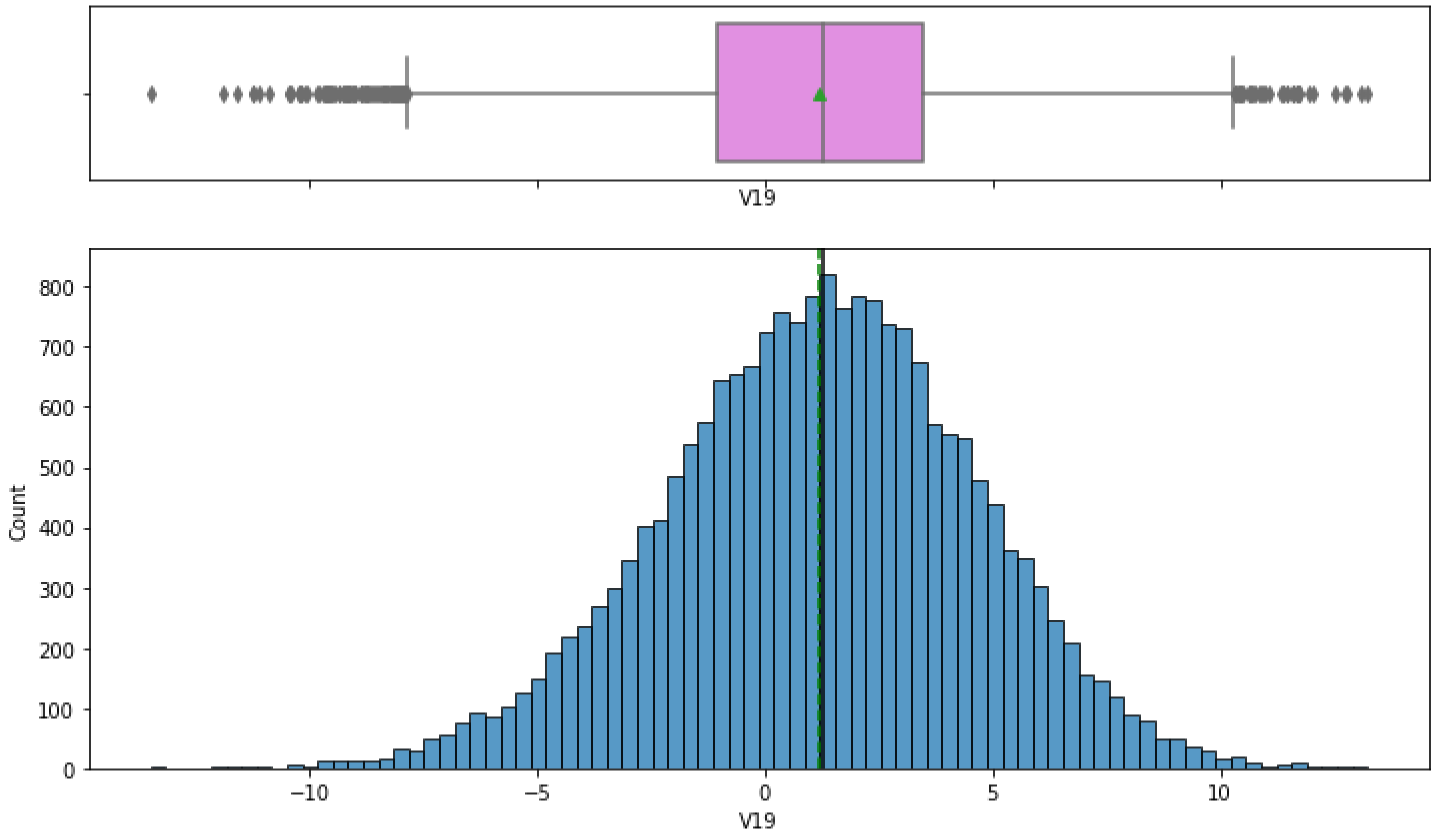

V14
Count
800
700
600
500
400
300
200
100
0
−8
−6
−4
−2
0
2
4
6
V14

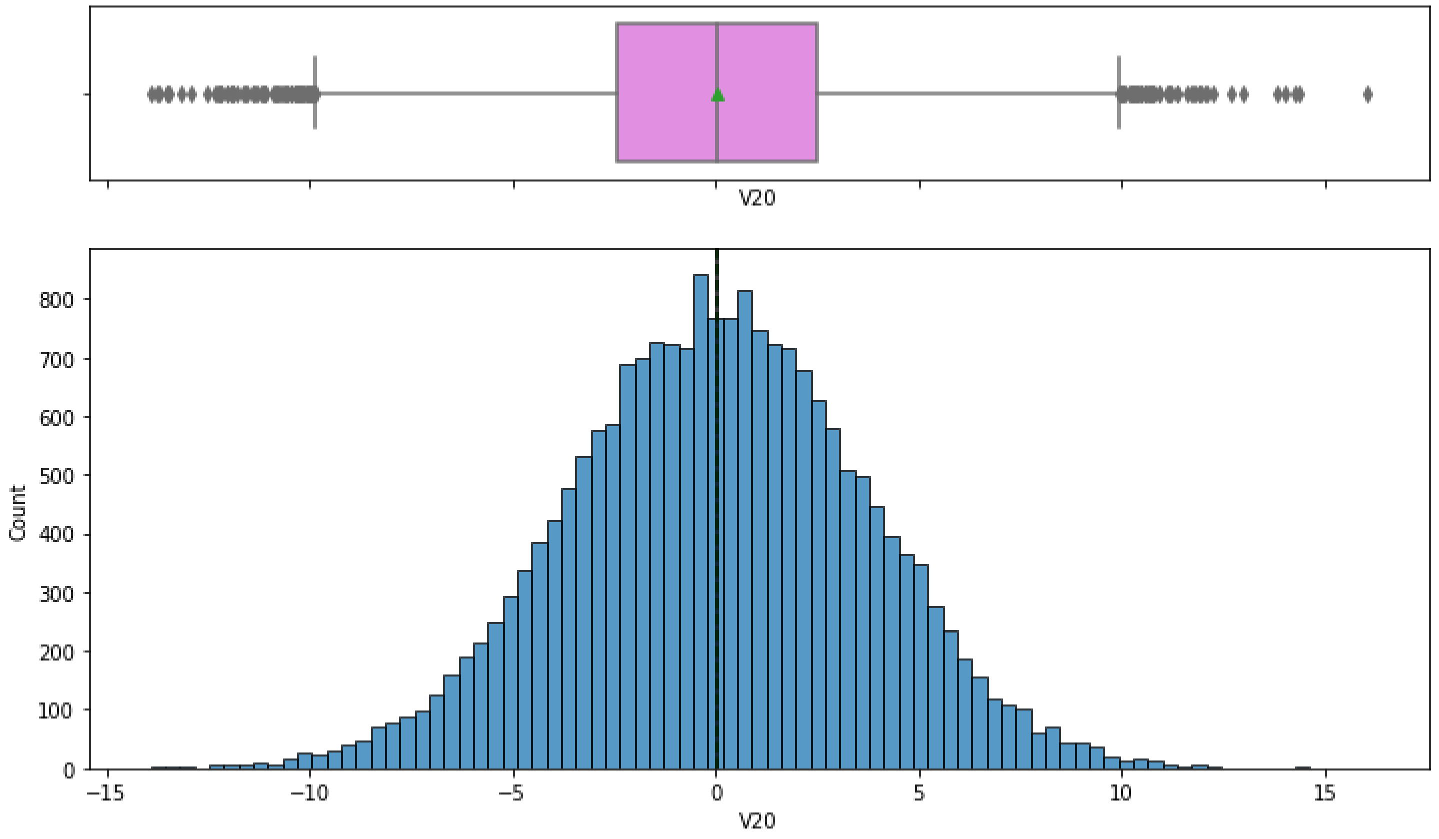

V15
Count
800
600
400
200
0
−15
−10
−5
0
5
10
V15

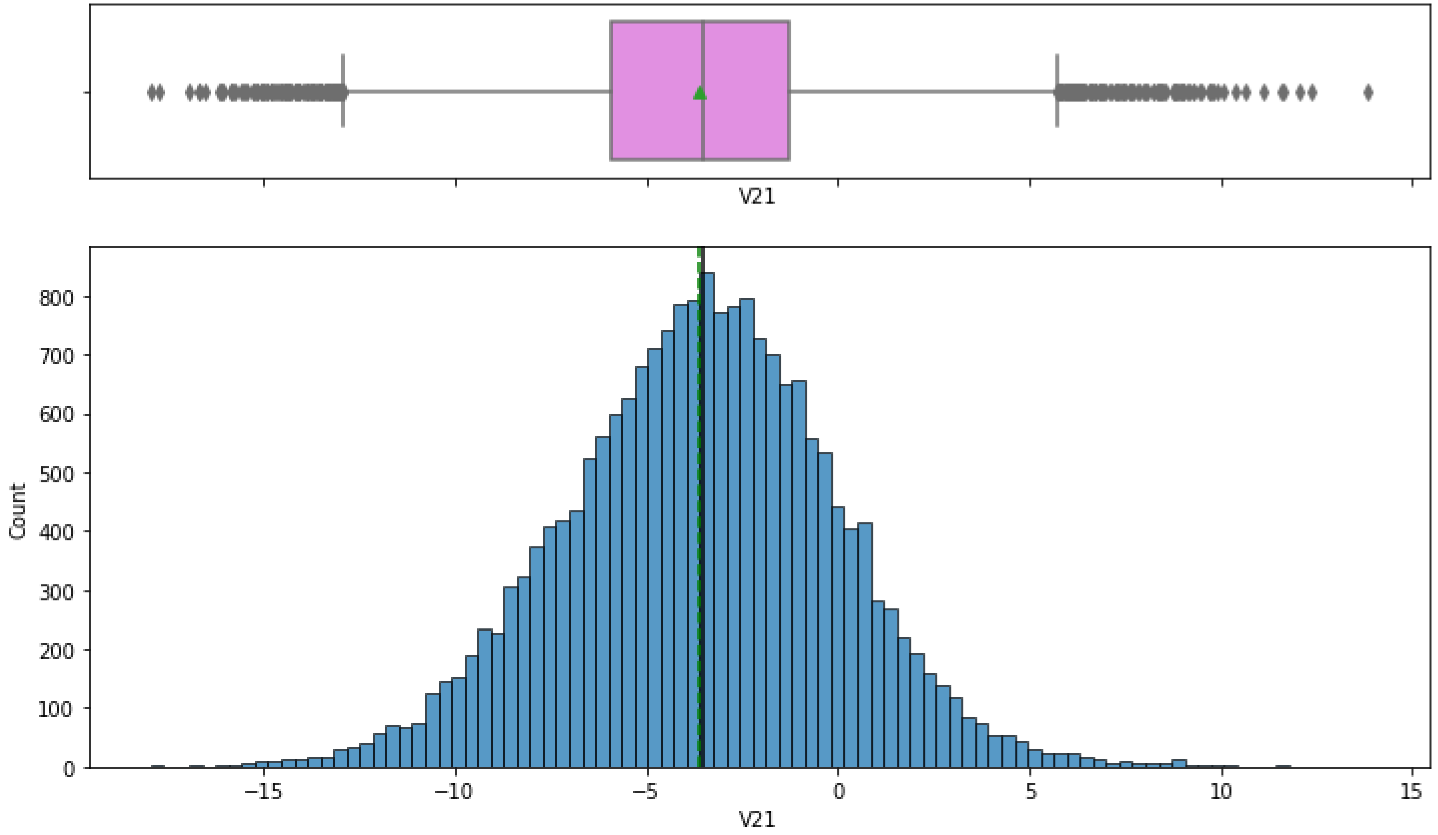

V16
Count
800
700
600
500
400
300
200
100
0
−20
−15
−10
−5
0
5
10
15
V16

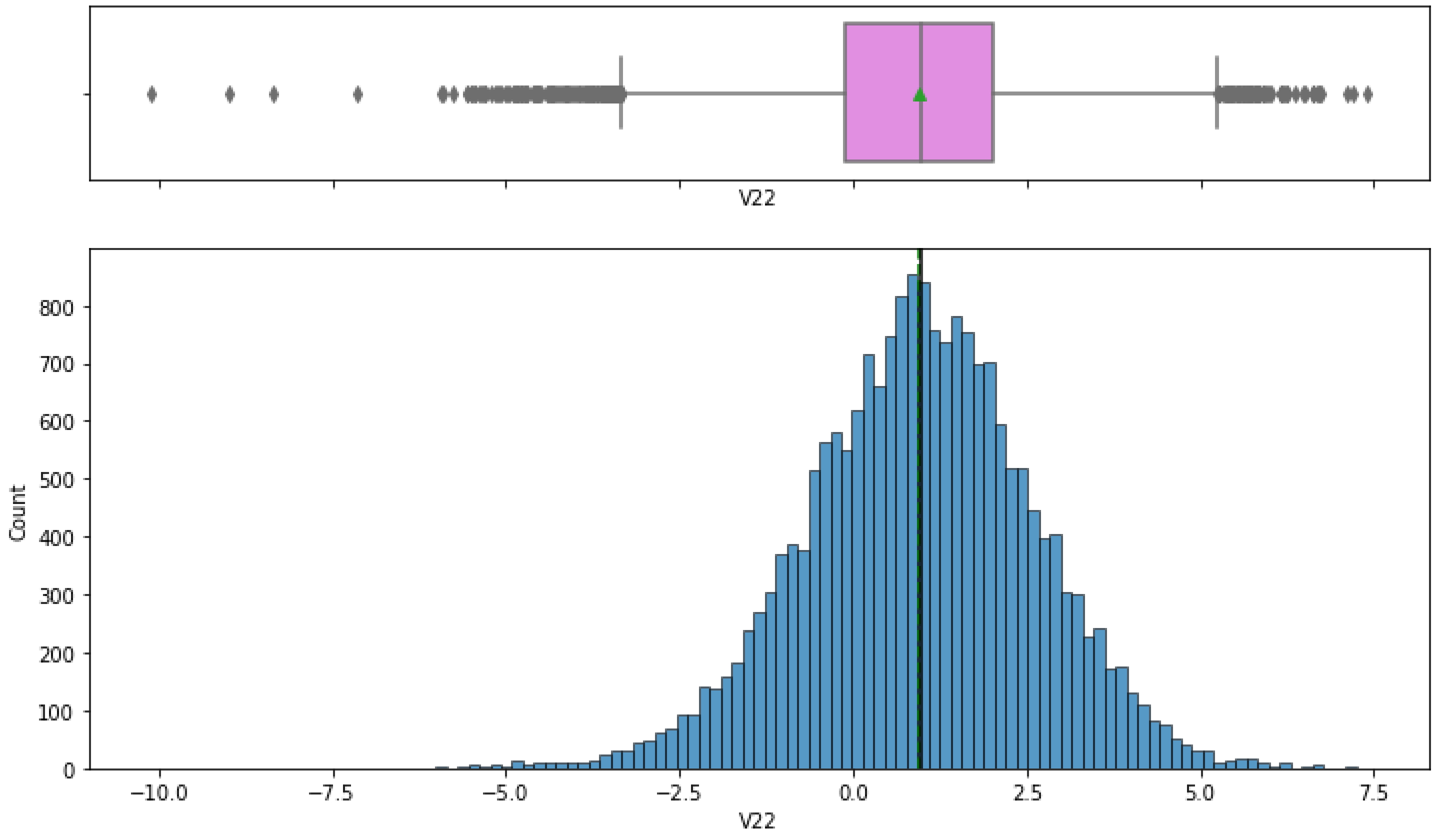

V17
800
700
600
500
400
300
200
100
0
Count
−15
−10
−5
0
5
10
15
V17

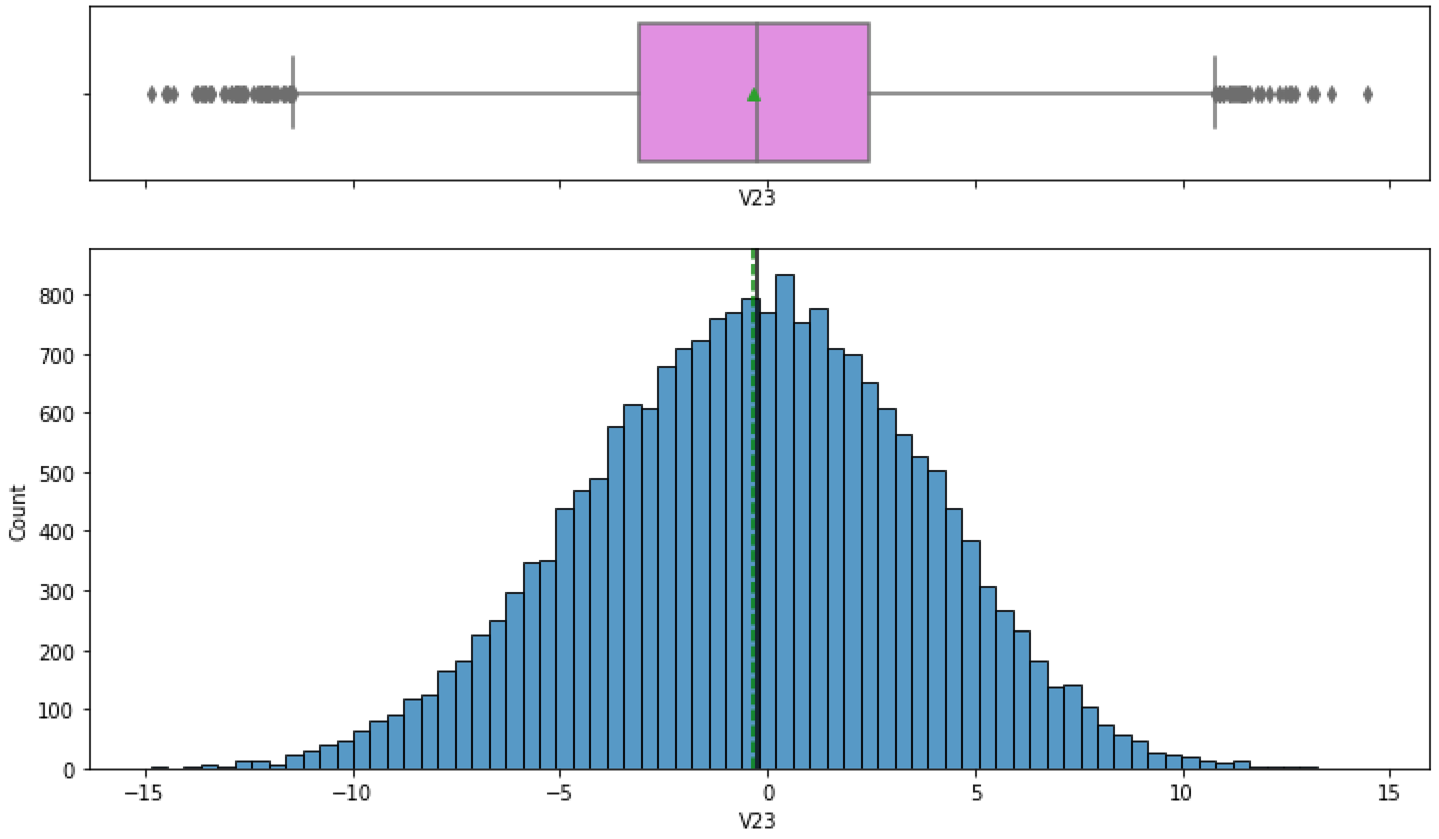

V18
800
600
400
200
0
Count
−10
−5
0
5
10
V18

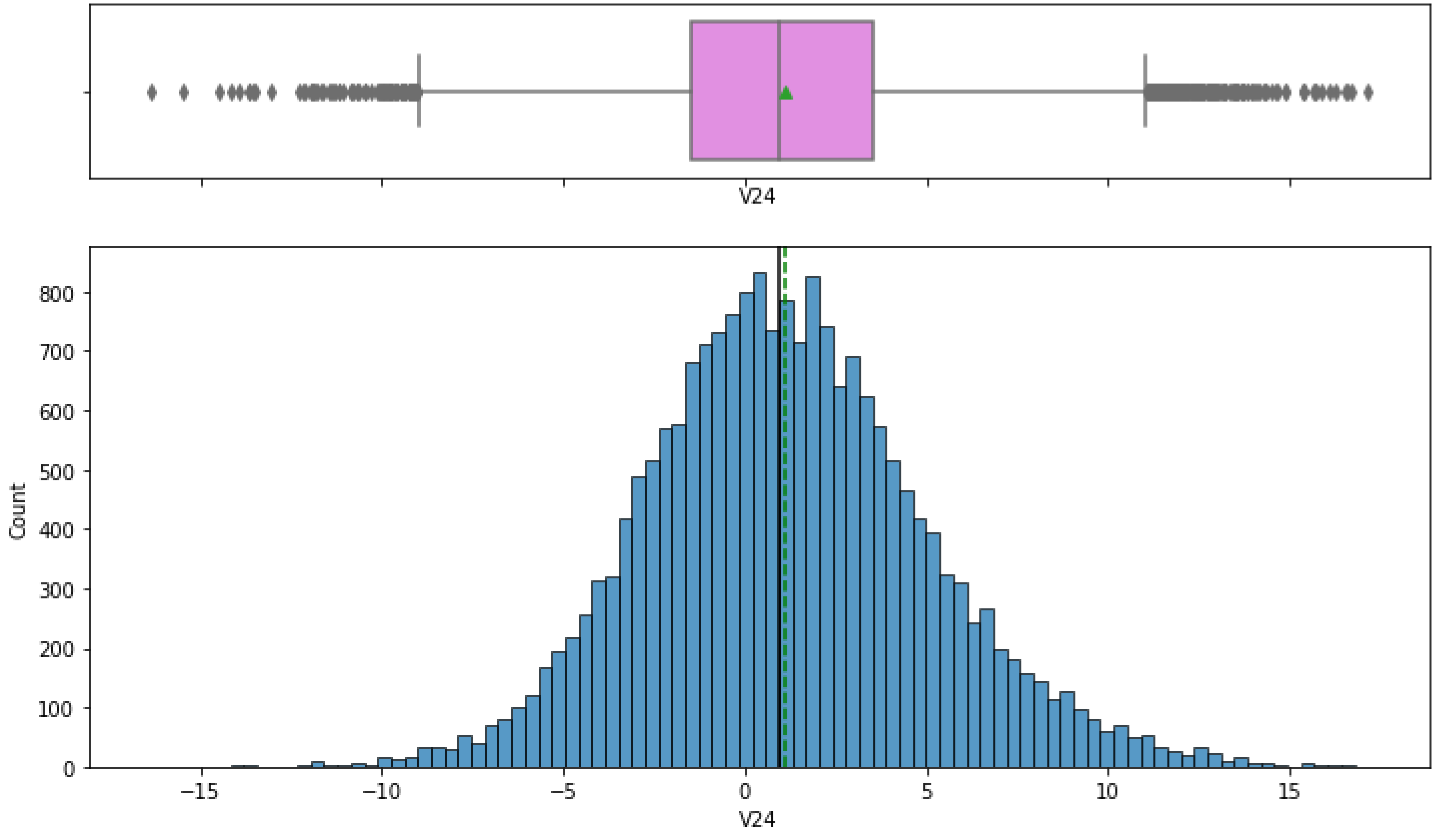

V19
Count
800
700
600
500
400
300
200
100
0
−10
−5
0
5
10
V19

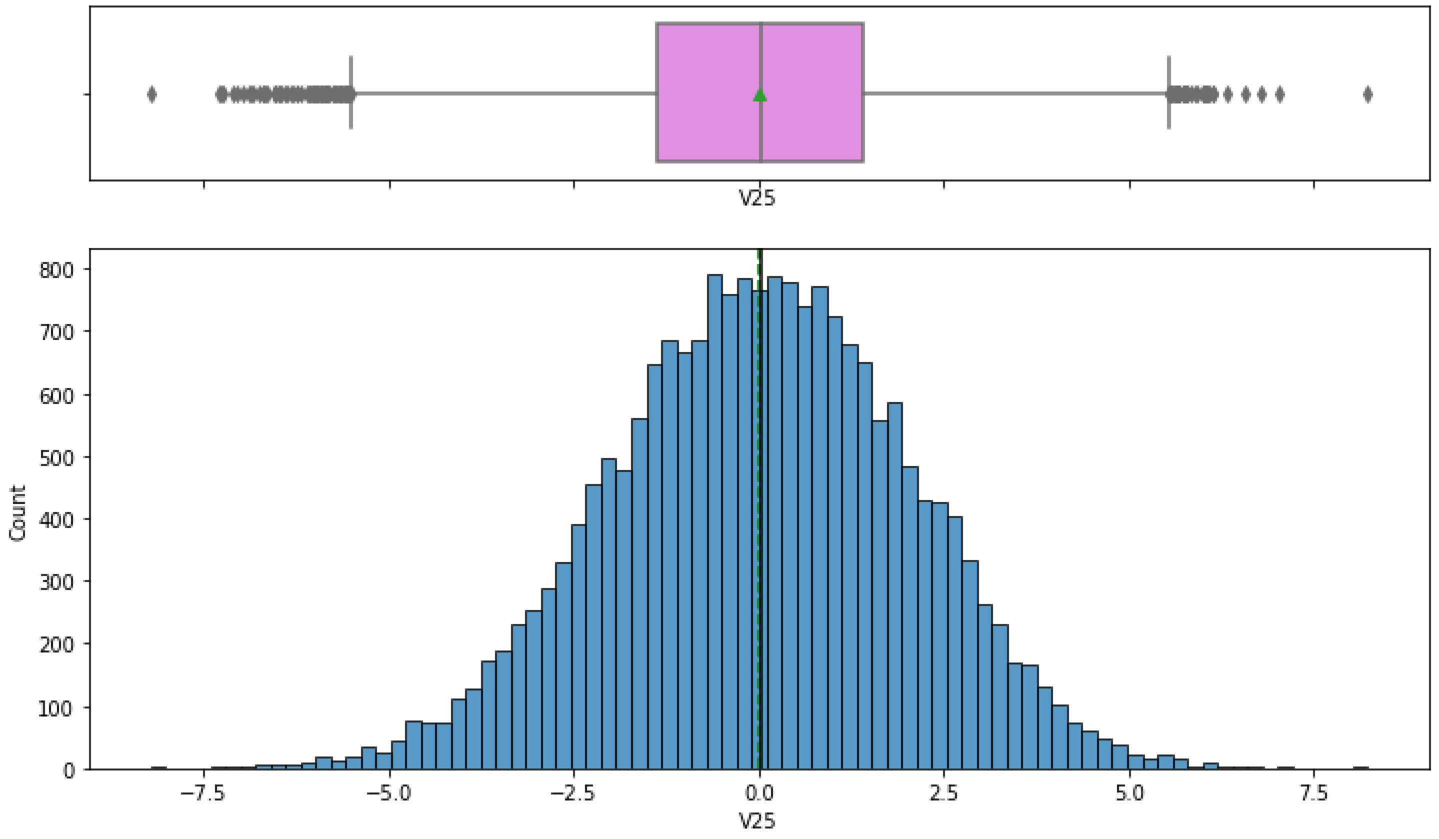

V20
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
15
V20

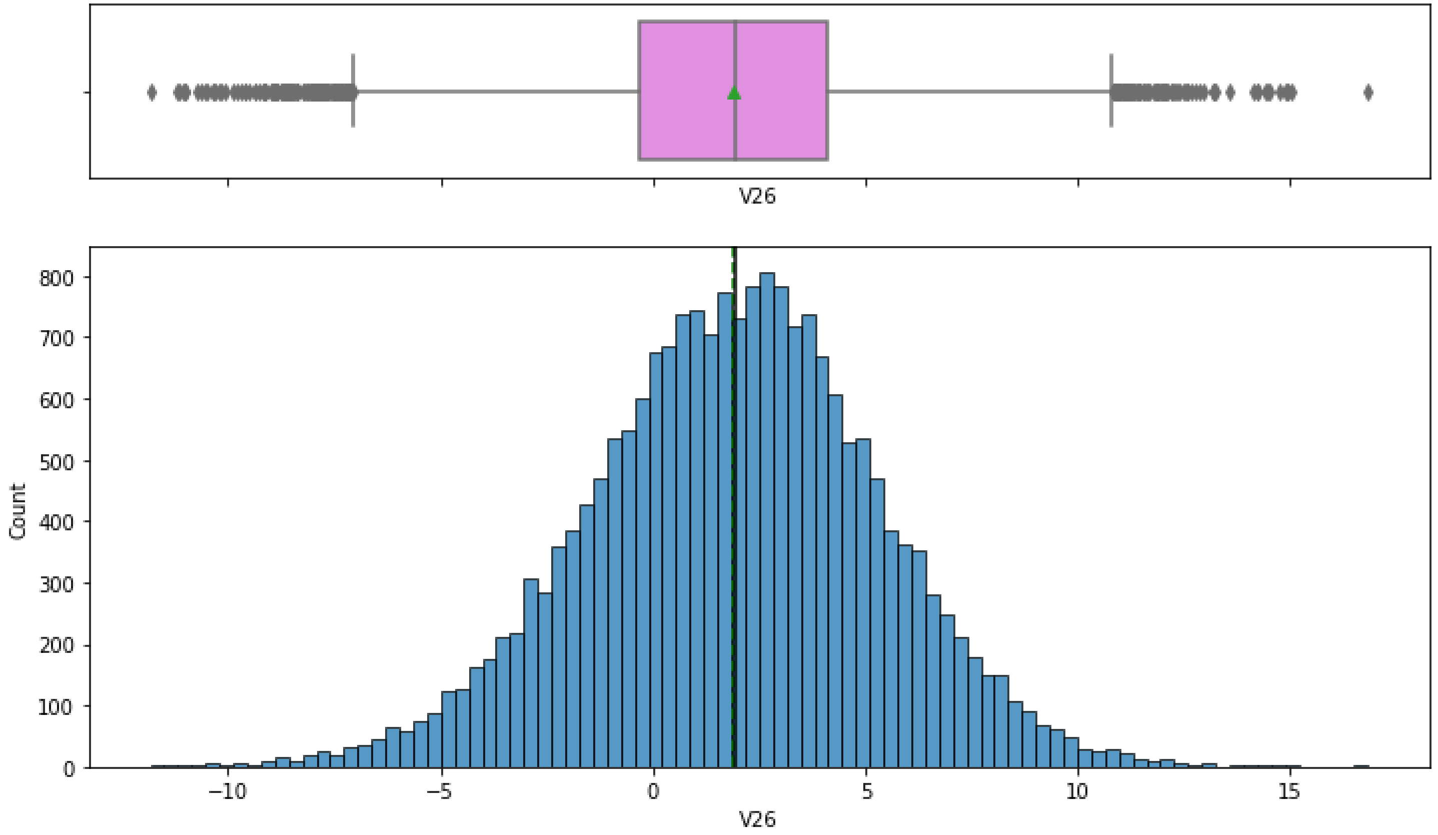

V21
800
700
600
500
400
300
200
100
0
Count
−15
−10
−5
0
5
10
15
V21

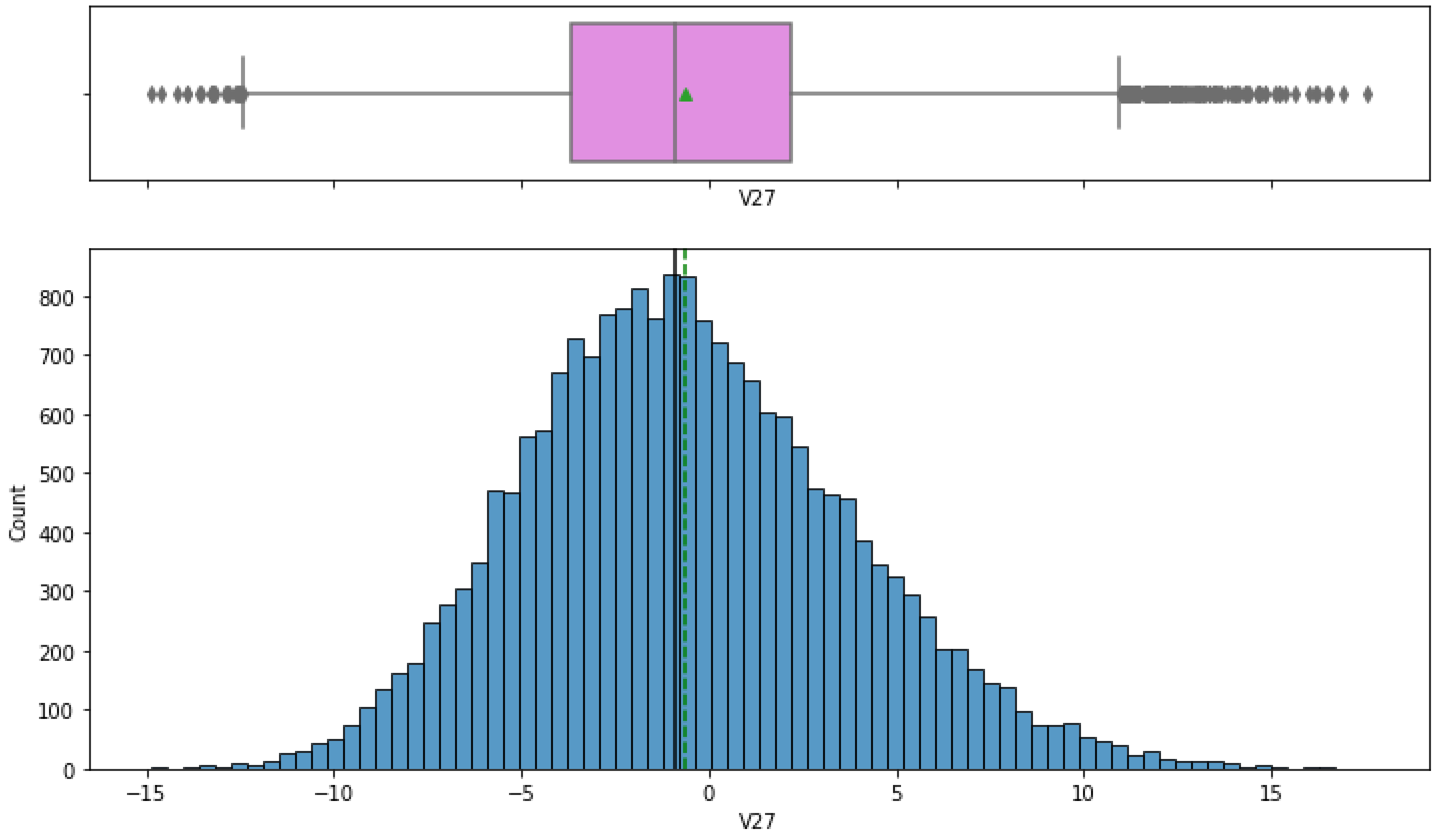

V22
800
700
600
500
400
300
200
100
0
Count
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V22

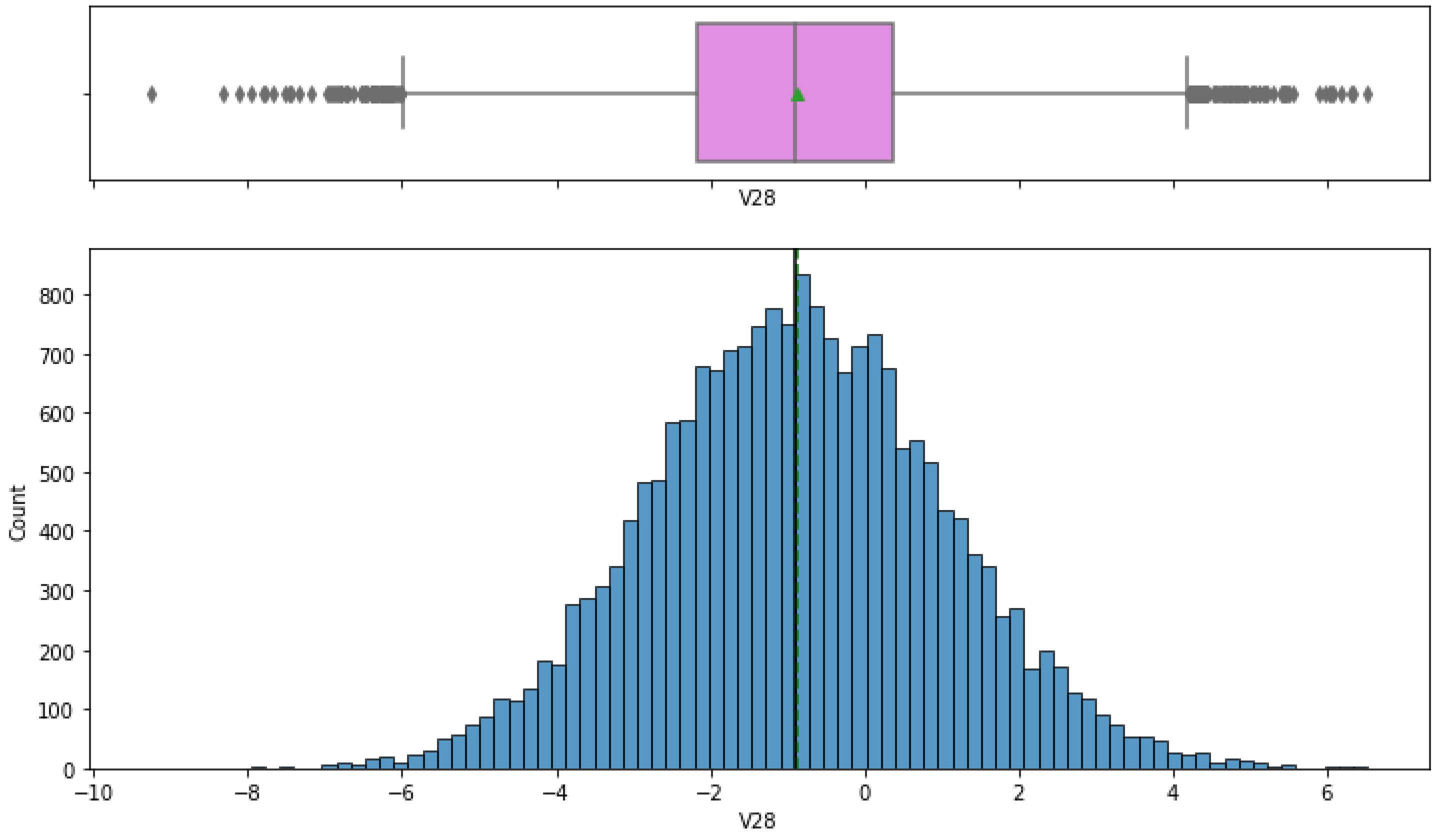

V23
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
15
V23

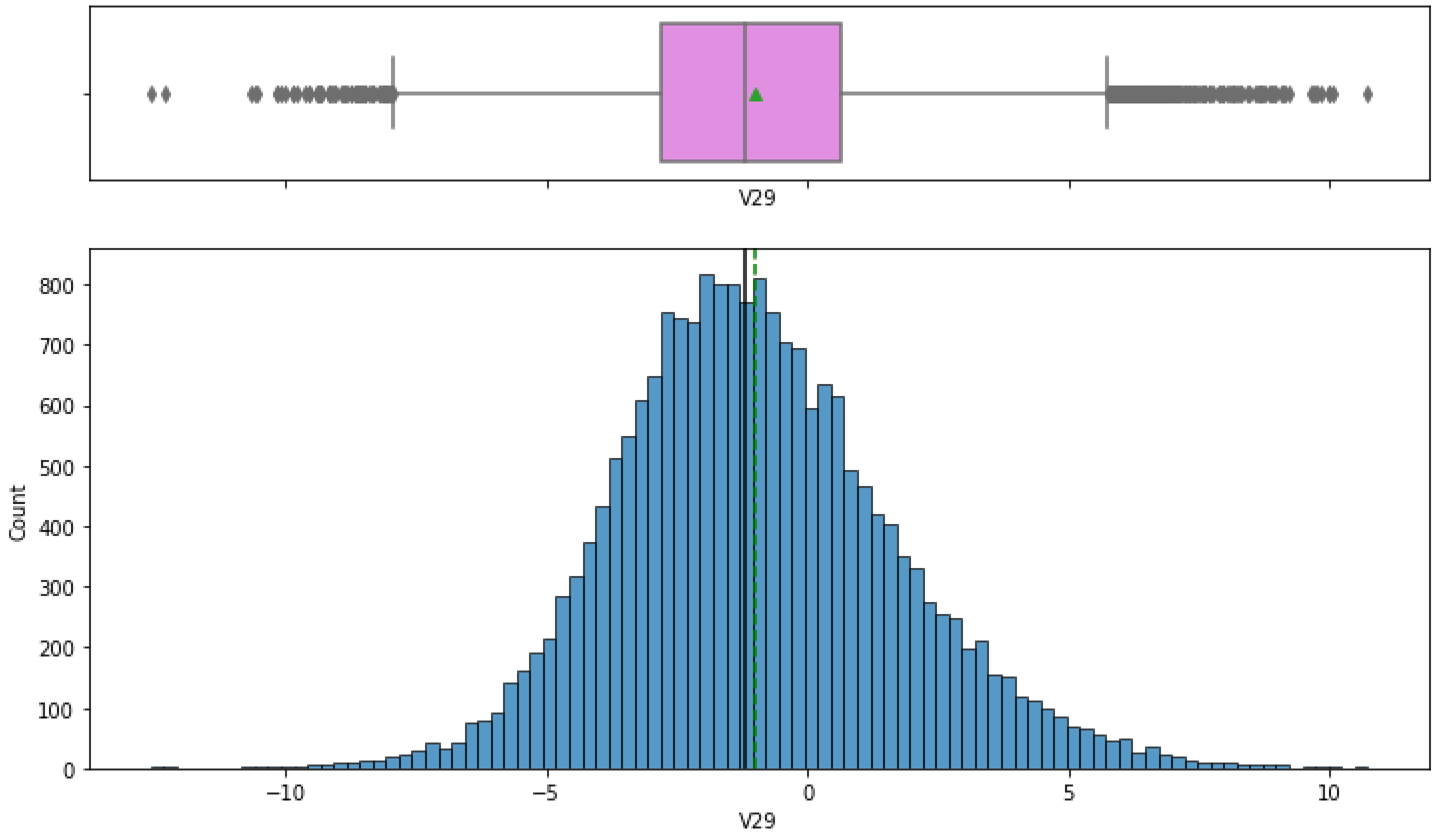

V24
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
15
V24

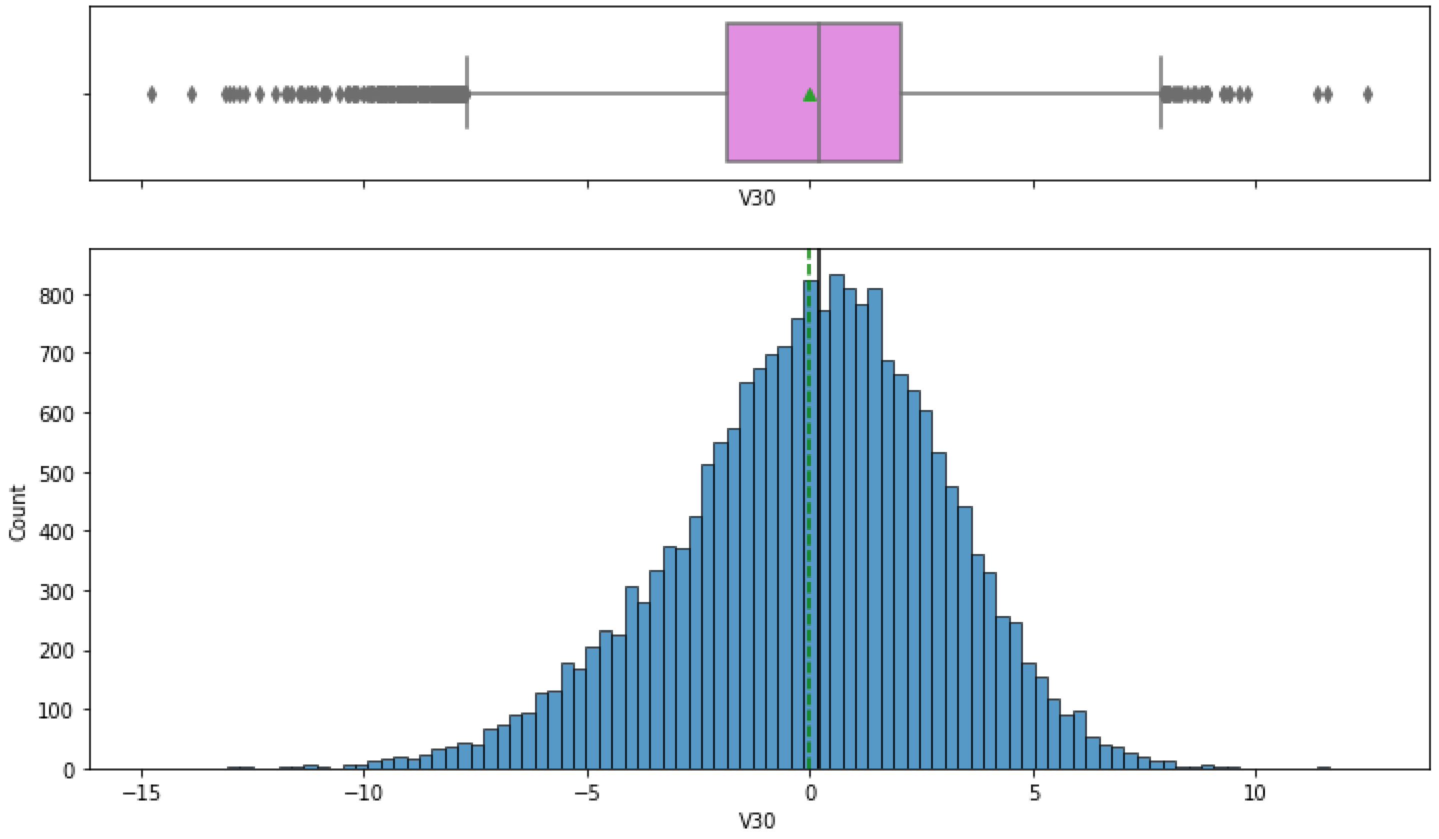

V25
Count
800
700
600
500
400
300
200
100
0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V25

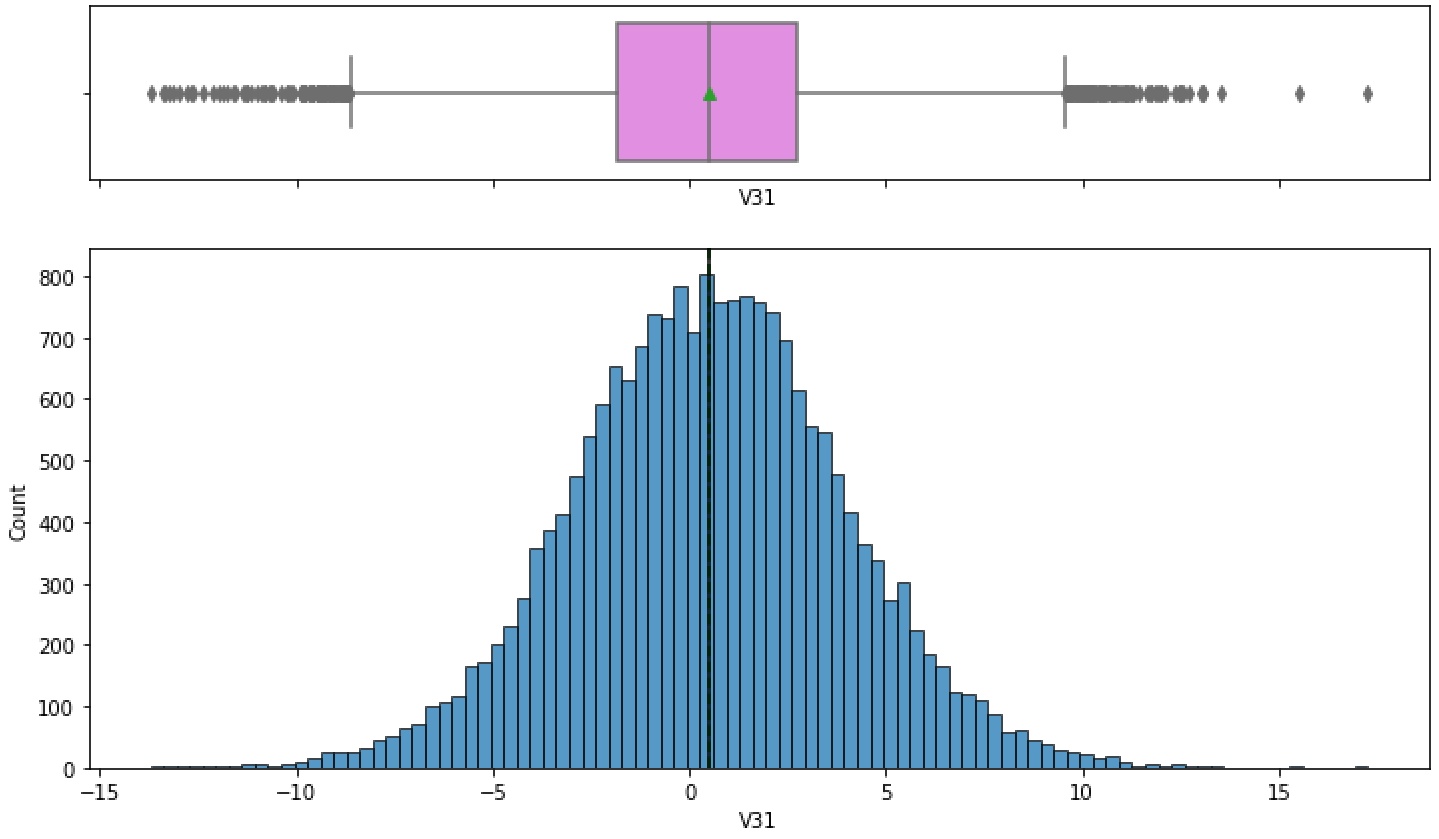

V26
800
700
600
500
400
300
200
100
0
Count
−10
−5
0
5
10
15
V26

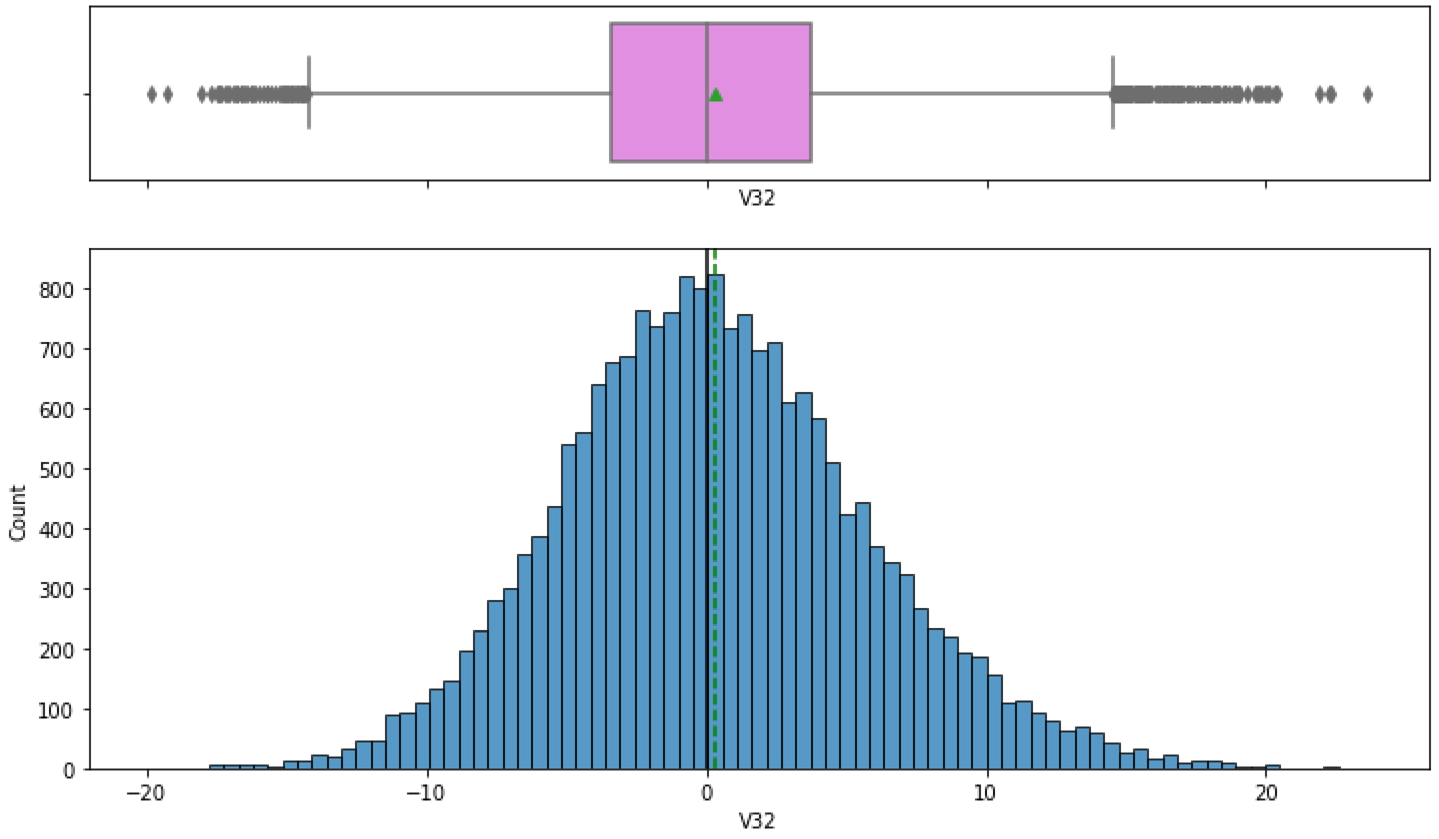

V27
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
15
V27

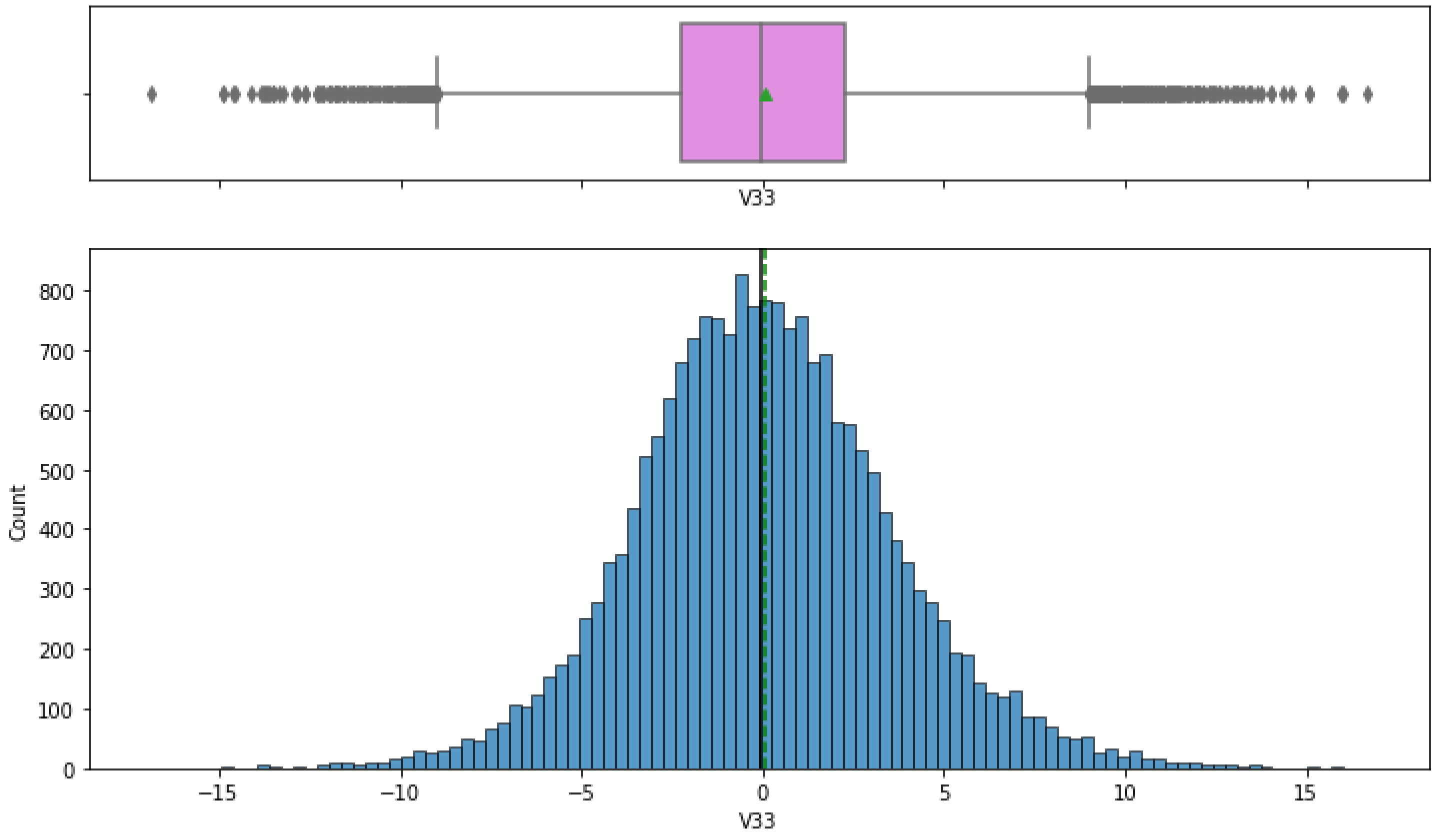

V28
800
700
600
500
400
300
200
100
0
Count
−10
−8
−6
−4
−2
0
2
4
6
V28

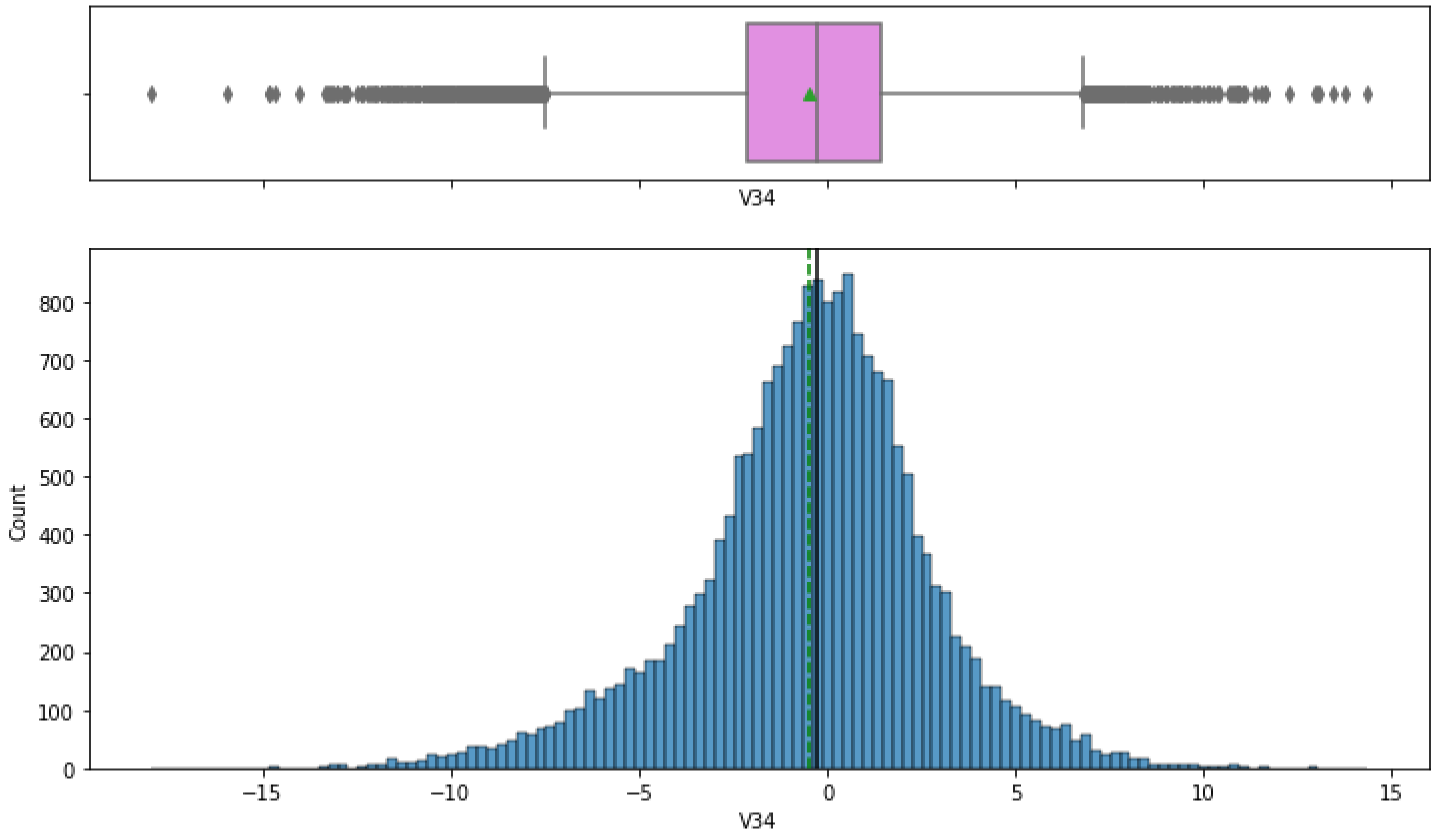

V29
800
700
600
500
400
300
200
100
0
Count
−10
−5
0
5
10
V29

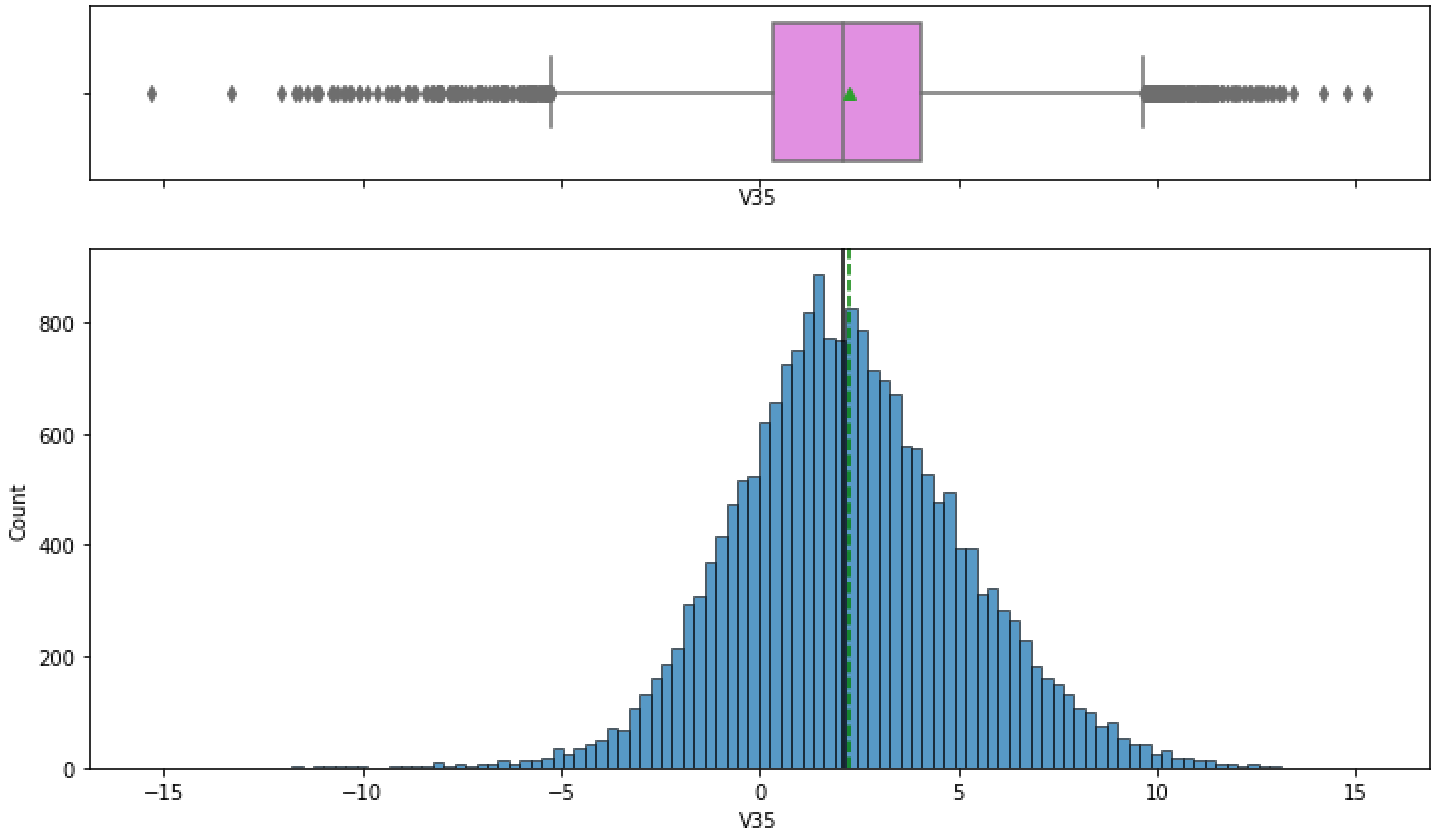

V30
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
V30

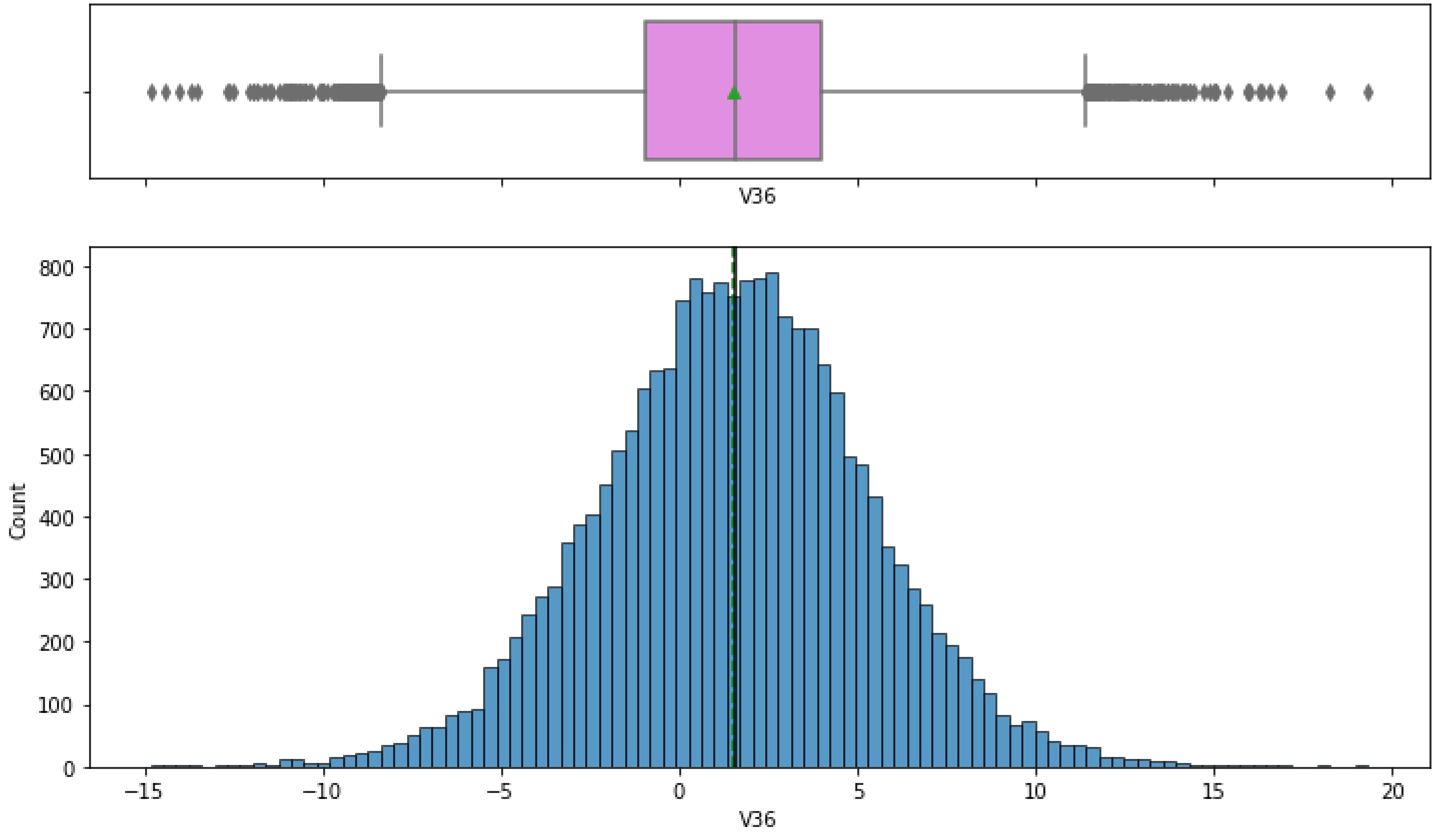

V31
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
15
V31

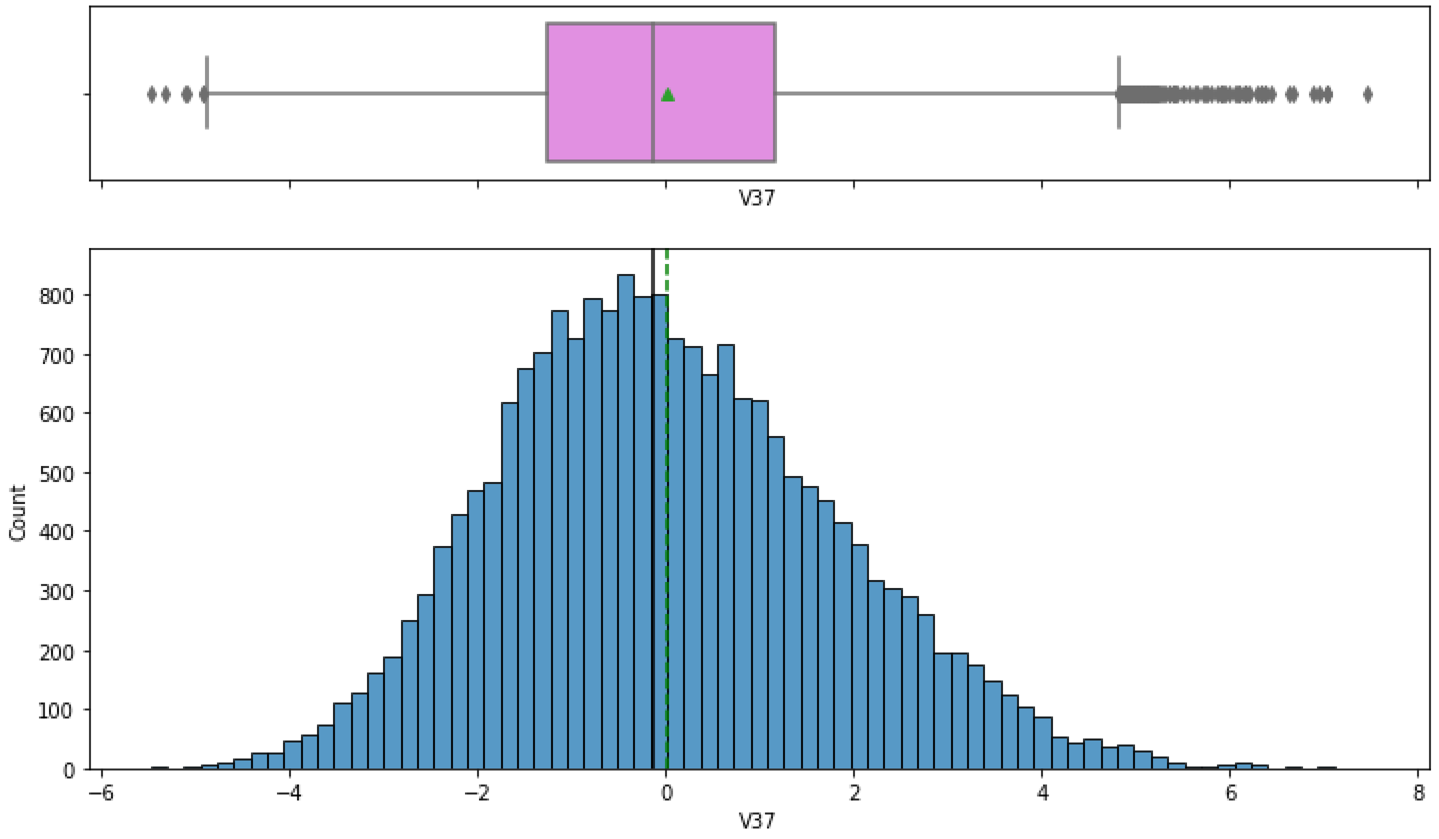

V32
Count
800
700
600
500
400
300
200
100
0
−20
−10
0
10
20
V32

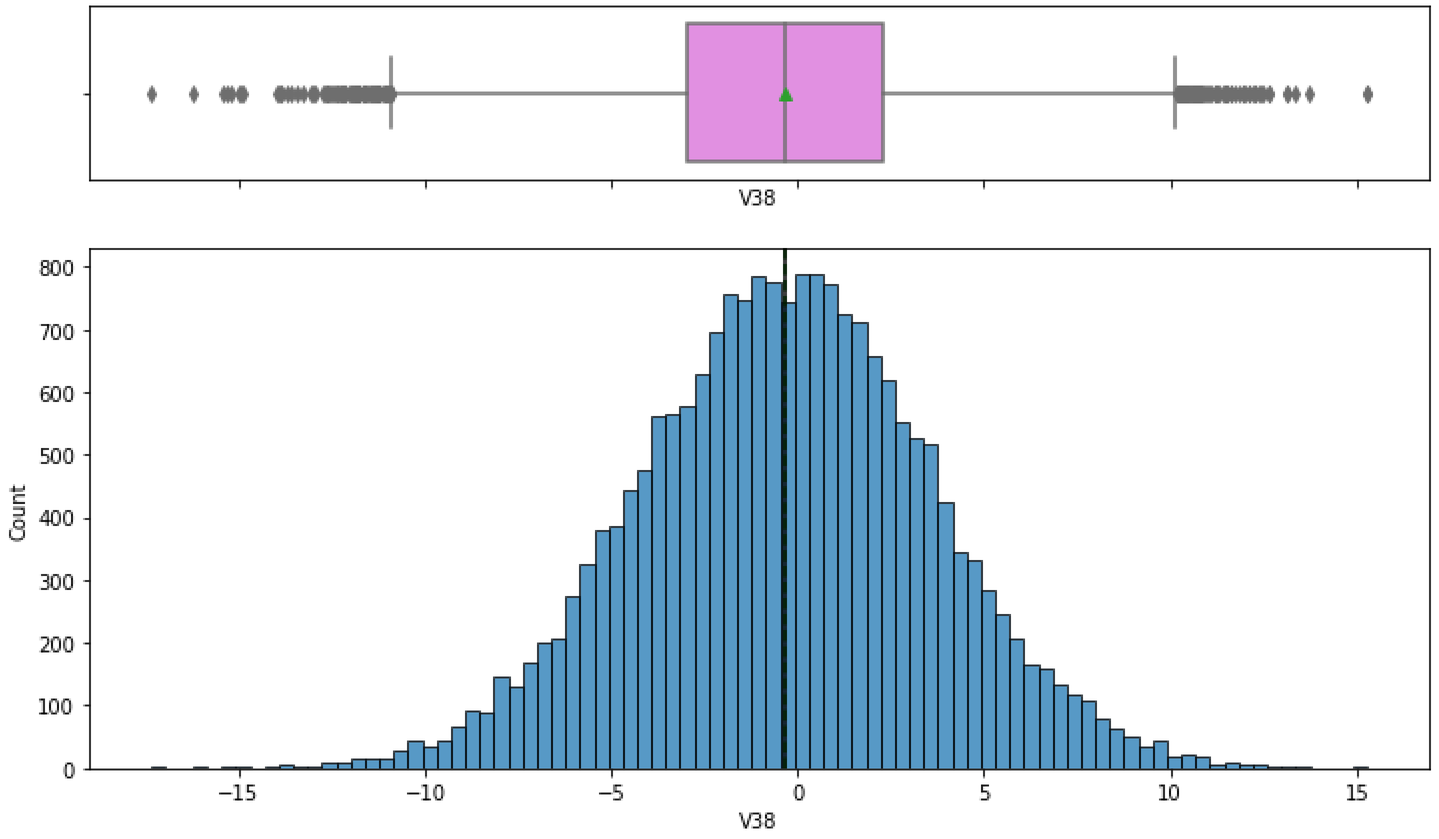

V33
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
15
V33

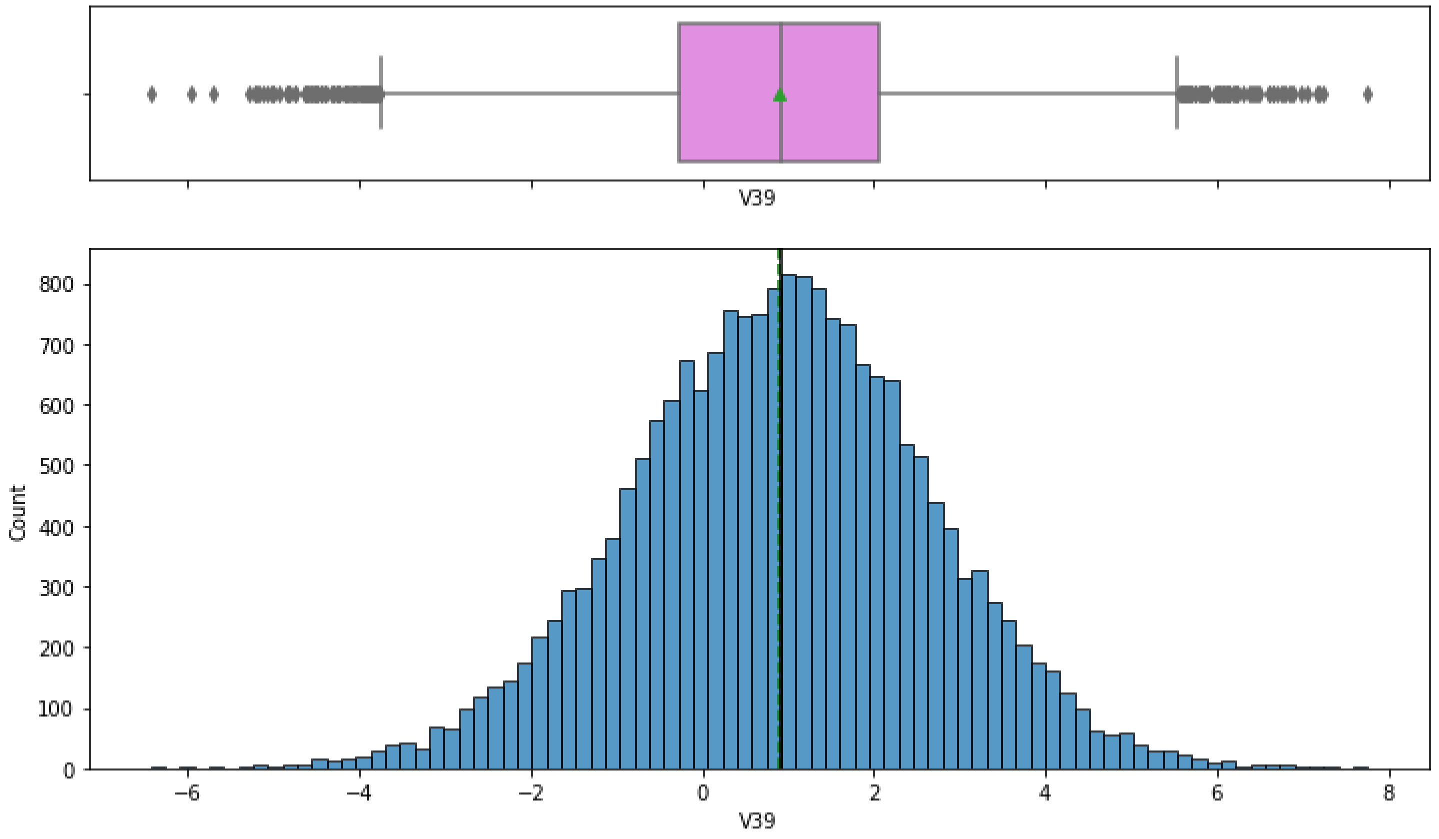

V34
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
15
V34

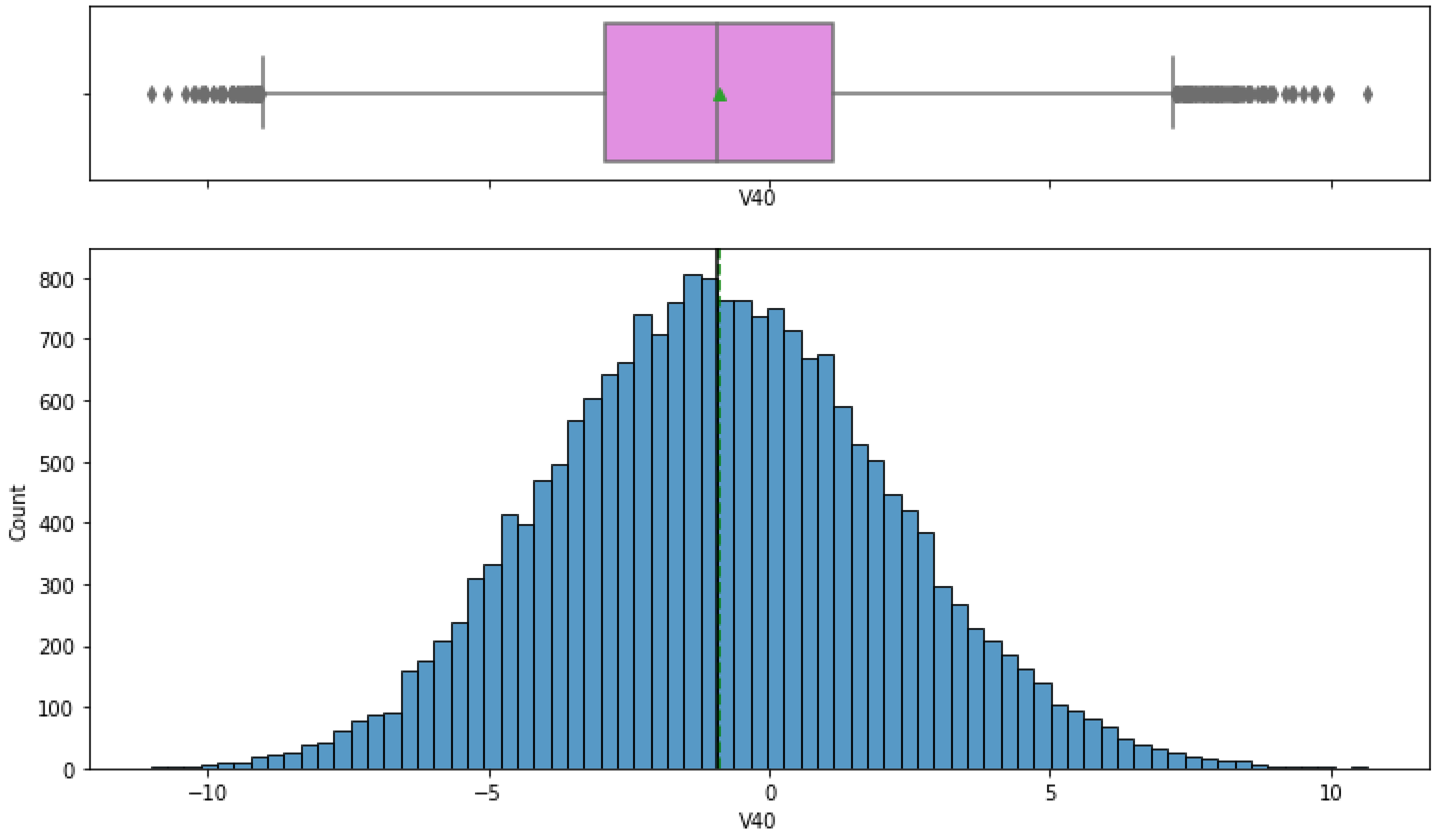

V35
800
600
400
200
0
Count
−15
−10
−5
0
5
10
15
V35

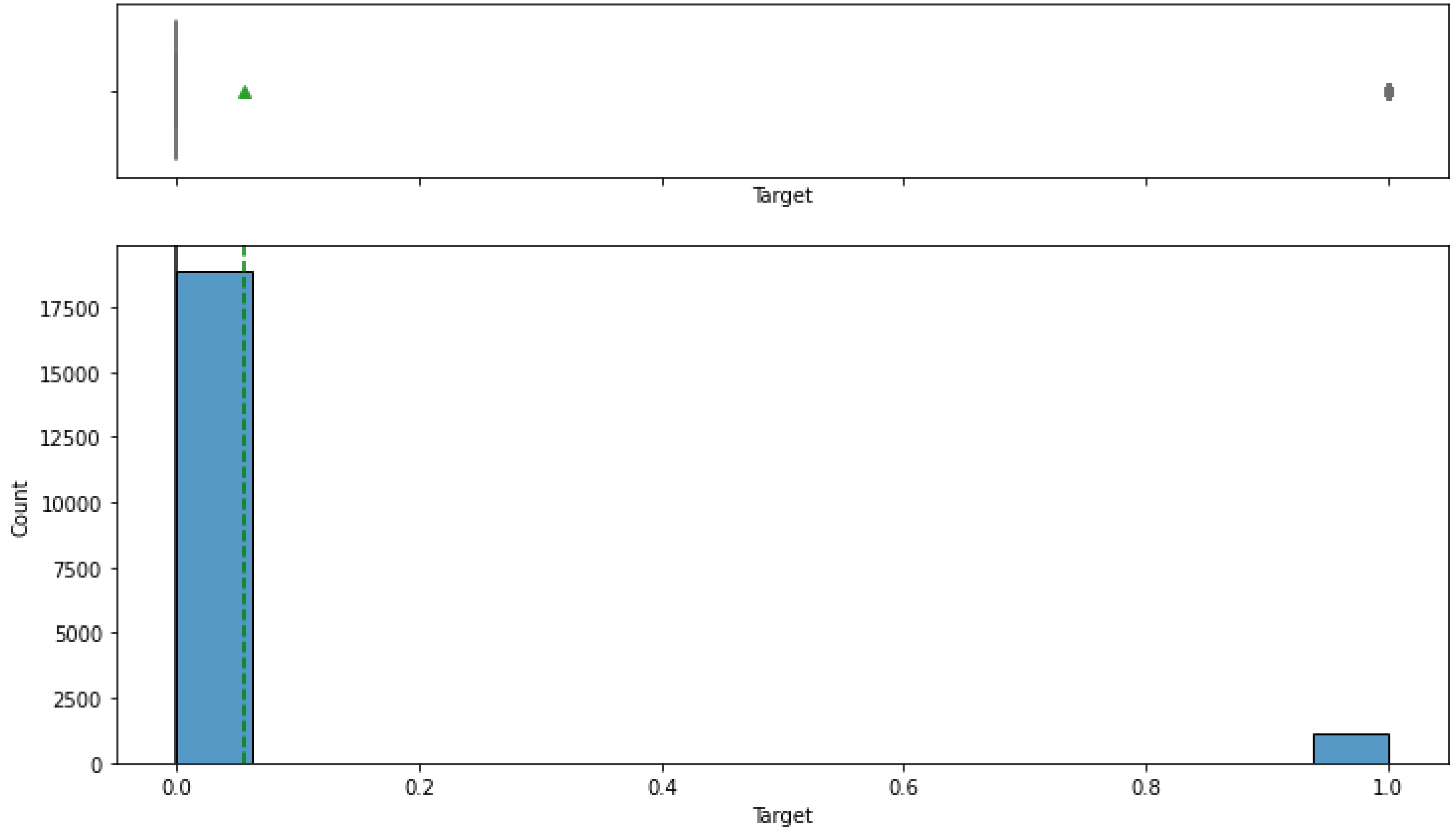

V36
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
15
20
V36


V37
800
700
600
500
400
300
200
100
0
Count
−6
−4
−2
0
2
4
6
8
V37


V38
800
700
600
500
400
300
200
100
0
Count
−15
−10
−5
0
5
10
15
V38


V39
800
700
600
500
400
300
200
100
0
Count
−6
−4
−2
0
2
4
6
8
V39


V40
Count
800
700
600
500
400
300
200
100
0
−10
−5
0
5
10
V40


Target
17500
15000
12500
10000
7500
5000
2500
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
Target

Happy Learning !