In [1]:
```
!pip3 install scikit-learn==1.3.2
!pip install imbalanced-learn==0.11.0
import pandas as pd
import numpy as np

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.impute import SimpleImputer

from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import (
    AdaBoostClassifier,
    GradientBoostingClassifier,
    RandomForestClassifier,
    BaggingClassifier,
)
from xgboost import XGBClassifier

from sklearn import metrics
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.metrics import (
    f1_score,
    accuracy_score,
    recall_score,
    precision_score,
    confusion_matrix,
    roc_auc_score,
    ConfusionMatrixDisplay,
)

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder


from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

pd.set_option("display.max_columns", None)

pd.set_option("display.float_format", lambda x: "%.3f" % x)


import warnings

warnings.filterwarnings("ignore")
```

```
Requirement already satisfied: scikit-learn==1.3.2 in c:\users\conne\anaconda3\lib\si
te-packages (1.3.2)
Requirement already satisfied: numpy<2.0,>=1.17.3 in c:\users\conne\anaconda3\lib\sit
e-packages (from scikit-learn==1.3.2) (1.24.3)
Requirement already satisfied: scipy>=1.5.0 in c:\users\conne\anaconda3\lib\site-pack
ages (from scikit-learn==1.3.2) (1.10.1)
Requirement already satisfied: joblib>=1.1.1 in c:\users\conne\anaconda3\lib\site-pac
kages (from scikit-learn==1.3.2) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\conne\anaconda3\lib\s
ite-packages (from scikit-learn==1.3.2) (2.2.0)
Requirement already satisfied: imbalanced-learn==0.11.0 in c:\users\conne\anaconda3\l
ib\site-packages (0.11.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\conne\anaconda3\lib\site-pac
kages (from imbalanced-learn==0.11.0) (1.24.3)
Requirement already satisfied: scipy>=1.5.0 in c:\users\conne\anaconda3\lib\site-pack
ages (from imbalanced-learn==0.11.0) (1.10.1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\conne\anaconda3\lib\si
te-packages (from imbalanced-learn==0.11.0) (1.3.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\conne\anaconda3\lib\site-pac
kages (from imbalanced-learn==0.11.0) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\conne\anaconda3\lib\s
ite-packages (from imbalanced-learn==0.11.0) (2.2.0)
```

Here im importing all packages

In [2]:
```python
df = pd.read_csv('/Users/conne/Downloads/Train.csv.csv')
df_test = pd.read_csv('/Users/conne/Downloads/Test.csv.csv')
```

Here im loading both sets of data

In [3]:
```python
data = df.copy()
```

In [4]:
```python
data_test = df_test.copy()
```

In [5]:
```python
data.shape
```

Out[5]:
```
(20000, 41)
```

We can see the first set has 20000 rows and 41 columns

In [6]:
```python
data_test.shape
```

Out[6]:
```
(5000, 41)
```

We can see the first set has 5000 rows and 41 columns

In [7]:
```python
data.head()
```

Out[7]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	
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	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	-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	
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	-
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	-

In [8]:

```
data.tail()
```

Out[8]:

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V1
19995	-2.071	-1.088	-0.796	-3.012	-2.288	2.807	0.481	0.105	-0.587	-2.899	8.868	1.717	1.3
19996	2.890	2.483	5.644	0.937	-1.381	0.412	-1.593	-5.762	2.150	0.272	-2.095	-1.526	0.0
19997	-3.897	-3.942	-0.351	-2.417	1.108	-1.528	-3.520	2.055	-0.234	-0.358	-3.782	2.180	6.1
19998	-3.187	-10.052	5.696	-4.370	-5.355	-1.873	-3.947	0.679	-2.389	5.457	1.583	3.571	9.2
19999	-2.687	1.961	6.137	2.600	2.657	-4.291	-2.344	0.974	-1.027	0.497	-9.589	3.177	1.0

In [9]:

```
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
```

In [10]:
```
data.duplicated().sum()
```
Out[10]:
```
0
```

In [11]:
```
data.isnull().sum()
```

Out[11]:
```
V1        18
V2        18
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
Target     0
dtype: int64
```

we have a couple null values that must be fixed

In [12]:
```
data_test.isnull().sum()
```

Out[12]:
```
V1        5
V2        6
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
Target    0
dtype: int64
```

we have a couple null values that must be fixed

In [13]:
```
data.describe().T
```

Out[13]:

	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

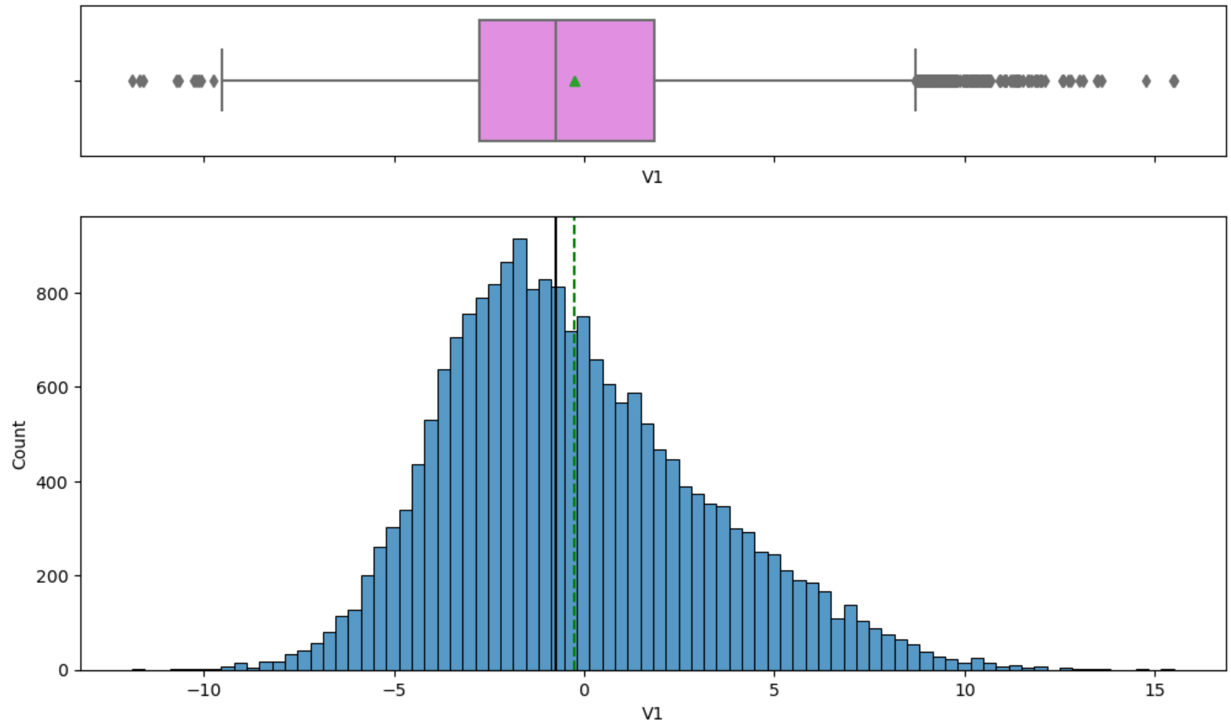
	count	mean	std	min	25%	50%	75%	max
V35	20000.000	2.230	2.937	-15.350	0.336	2.099	4.064	15.291
V36	20000.000	1.515	3.801	-14.833	-0.944	1.567	3.984	19.330
V37	20000.000	0.011	1.788	-5.478	-1.256	-0.128	1.176	7.467
V38	20000.000	-0.344	3.948	-17.375	-2.988	-0.317	2.279	15.290
V39	20000.000	0.891	1.753	-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

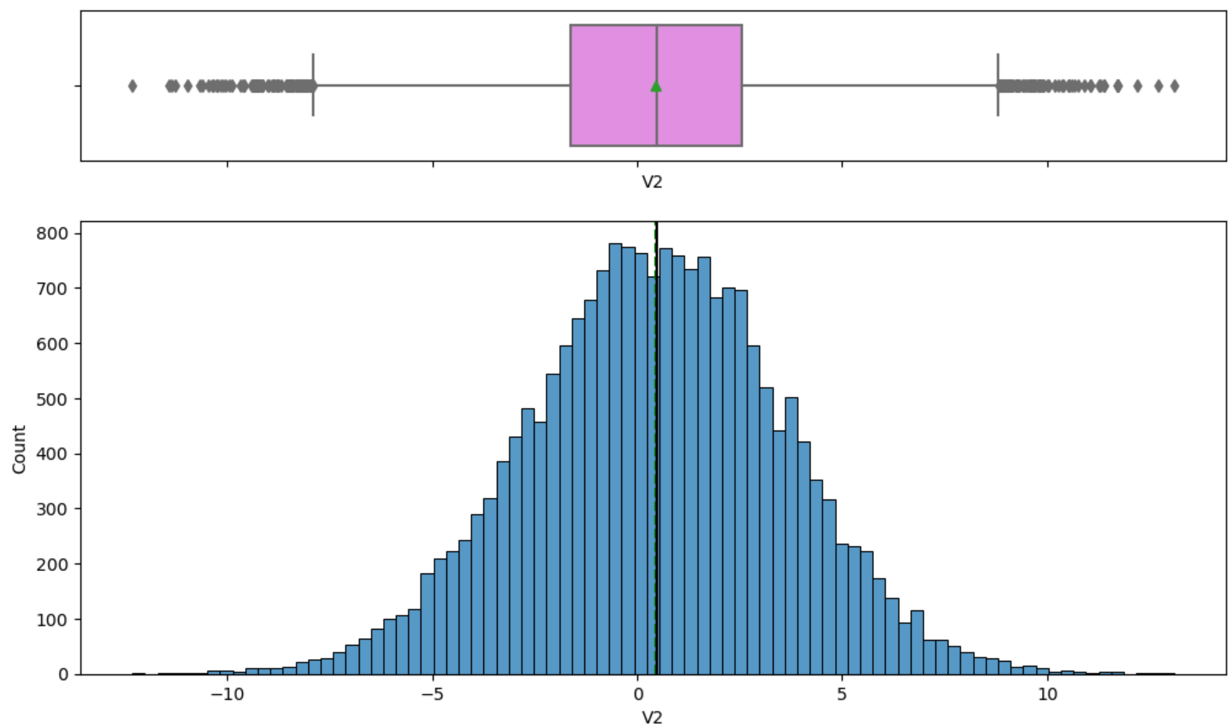
we can see the satistical summary

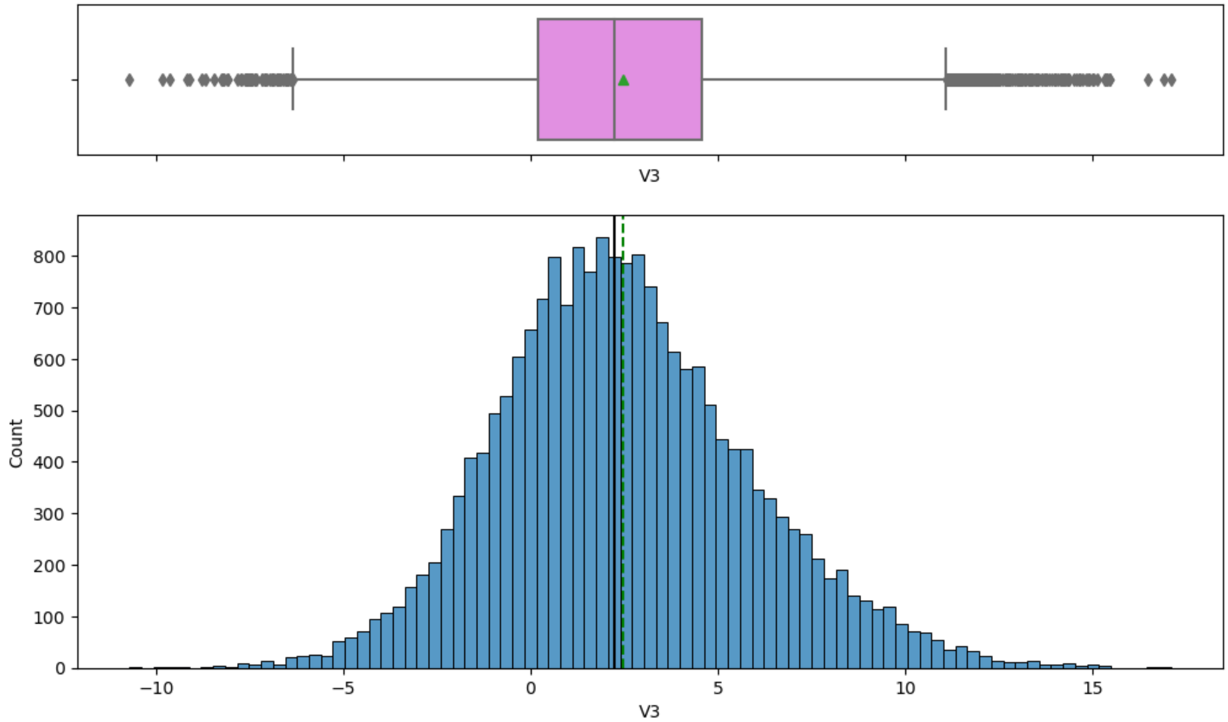
In [14]:
```
def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the column
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

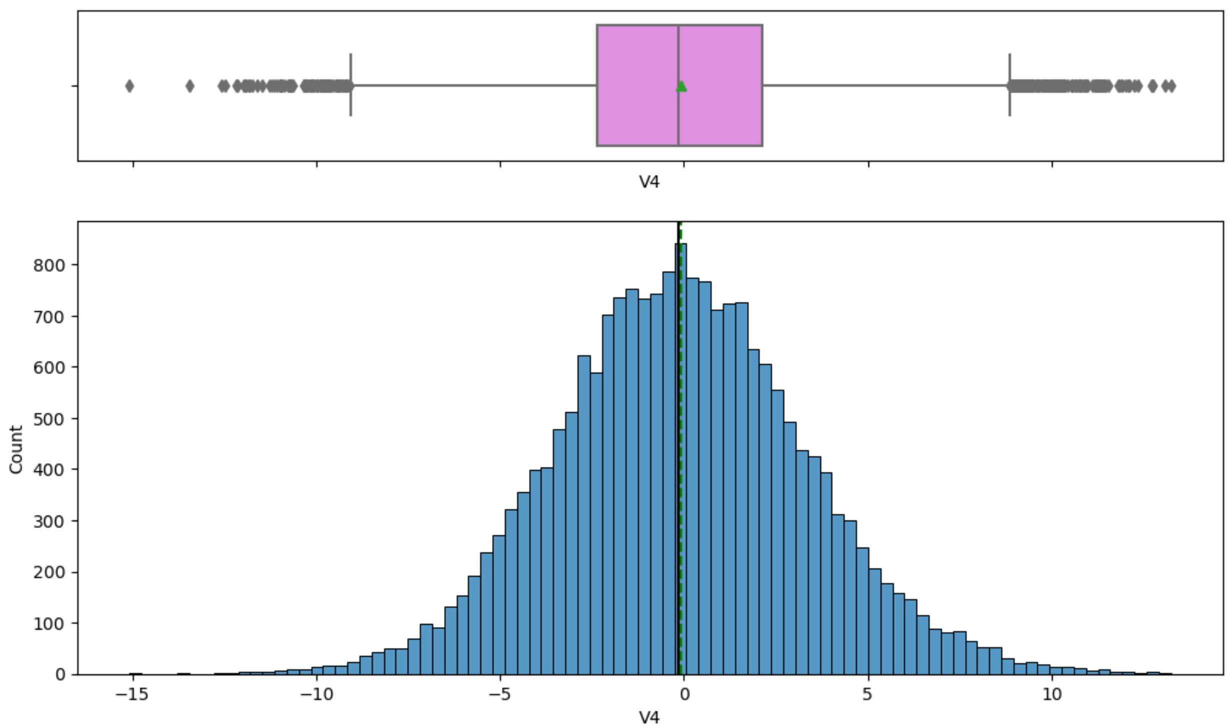
Here i will preform univariant analysis

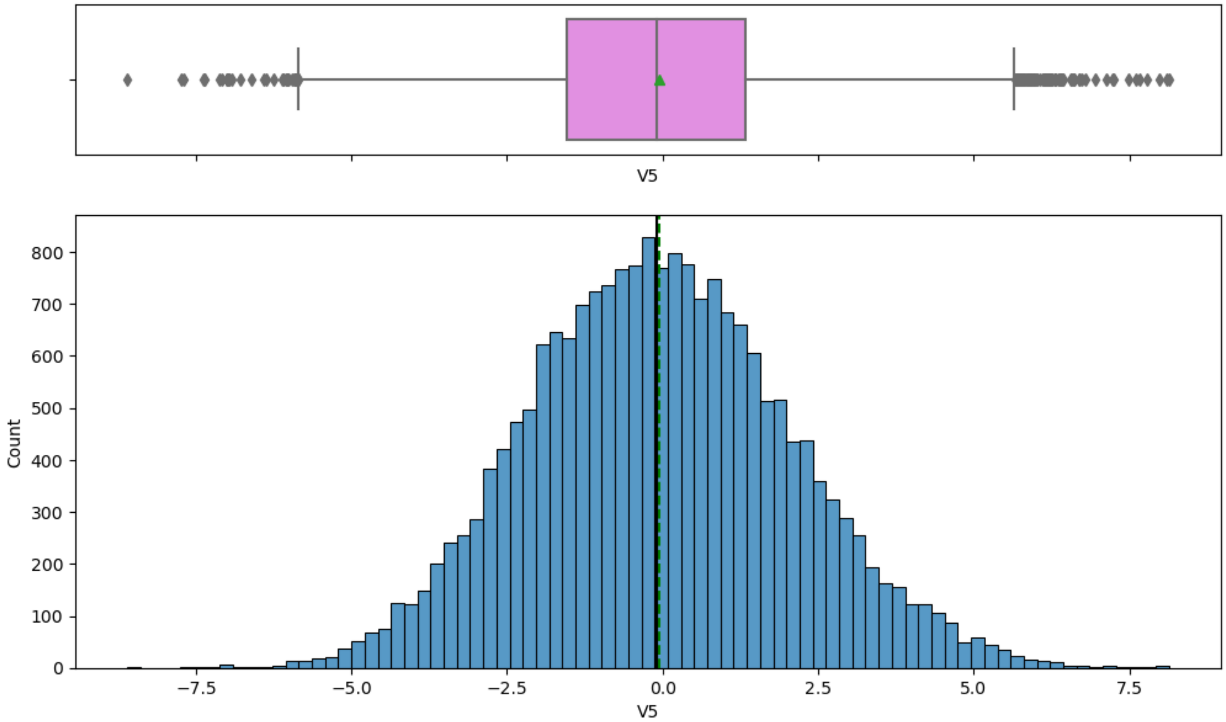
In [15]:
```
for feature in df.columns :
    histogram_boxplot(df, feature, figsize=(12,7), kde=False, bins=None)
```

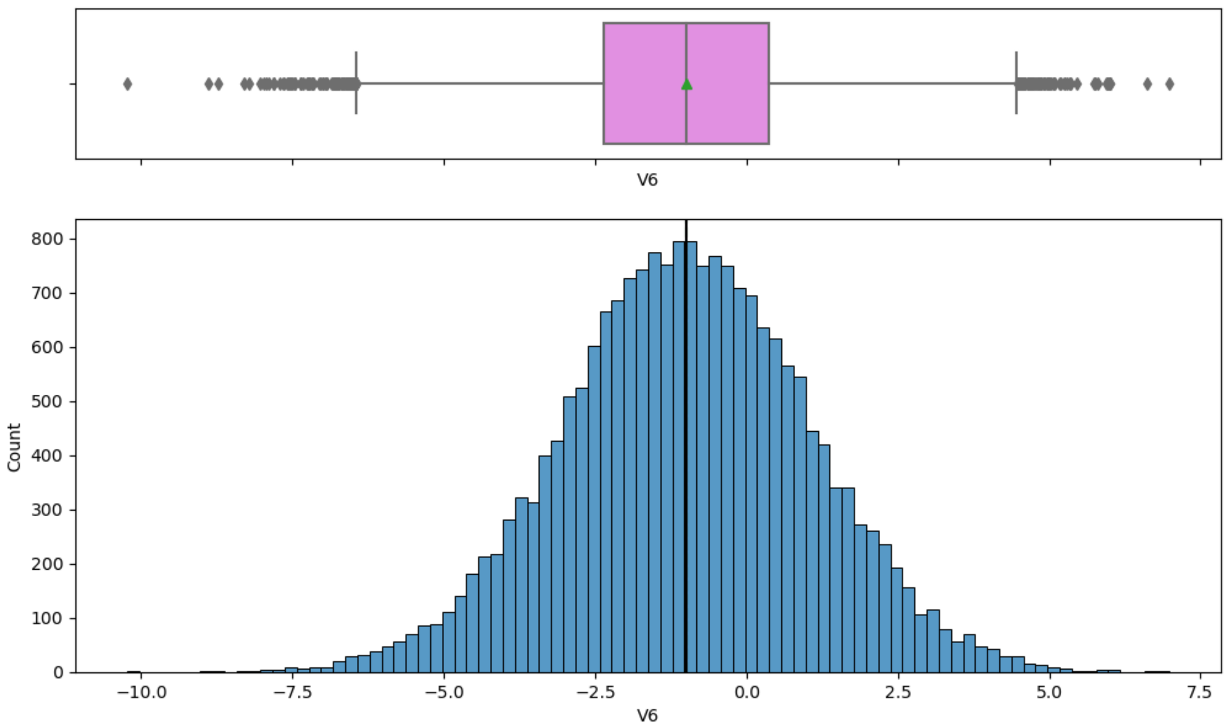

V1
Count
800
600
400
200
0
−10
−5
0
5
10
15
V1

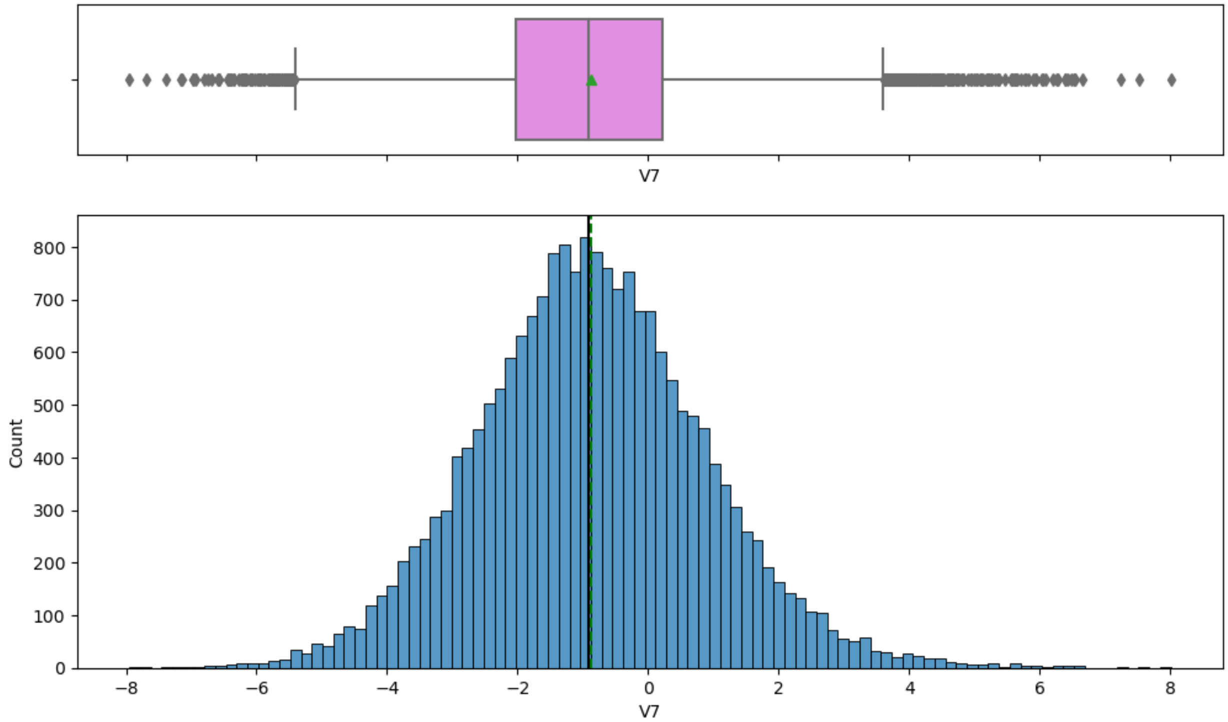

V2
Count
800
700
600
500
400
300
200
100
0
−10
−5
0
5
10
V2

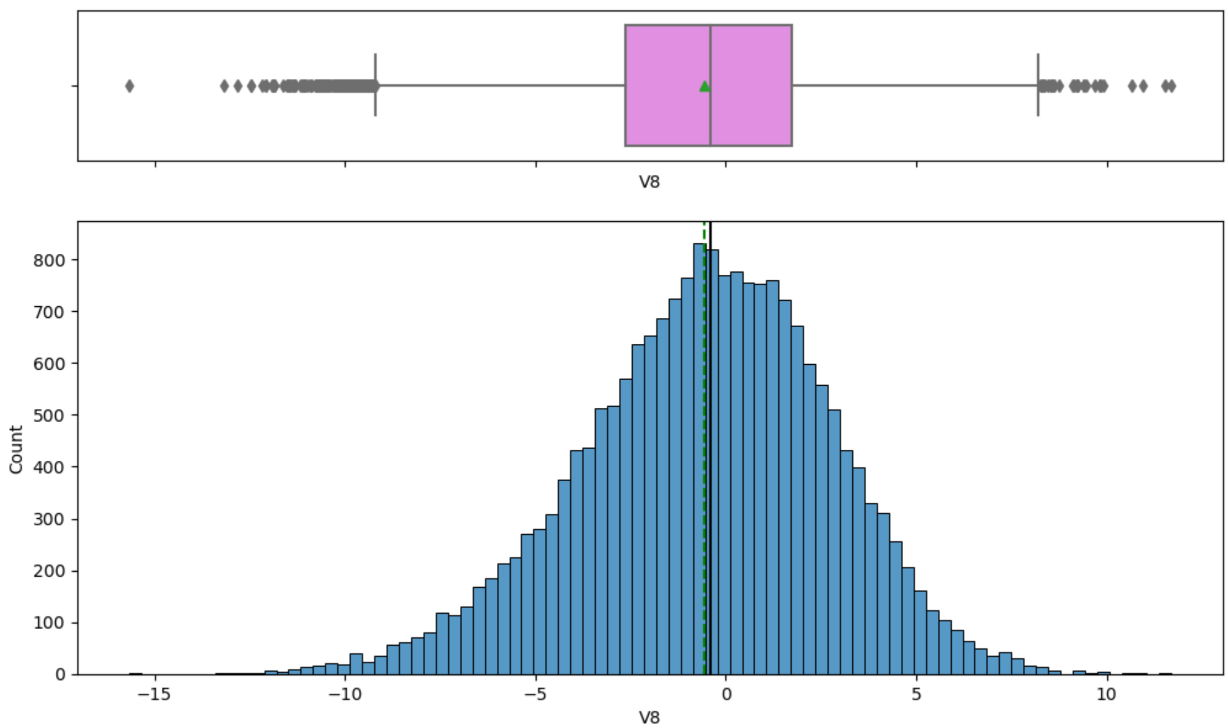

V3
Count
800
700
600
500
400
300
200
100
0
−10
−5
0
5
10
15
V3

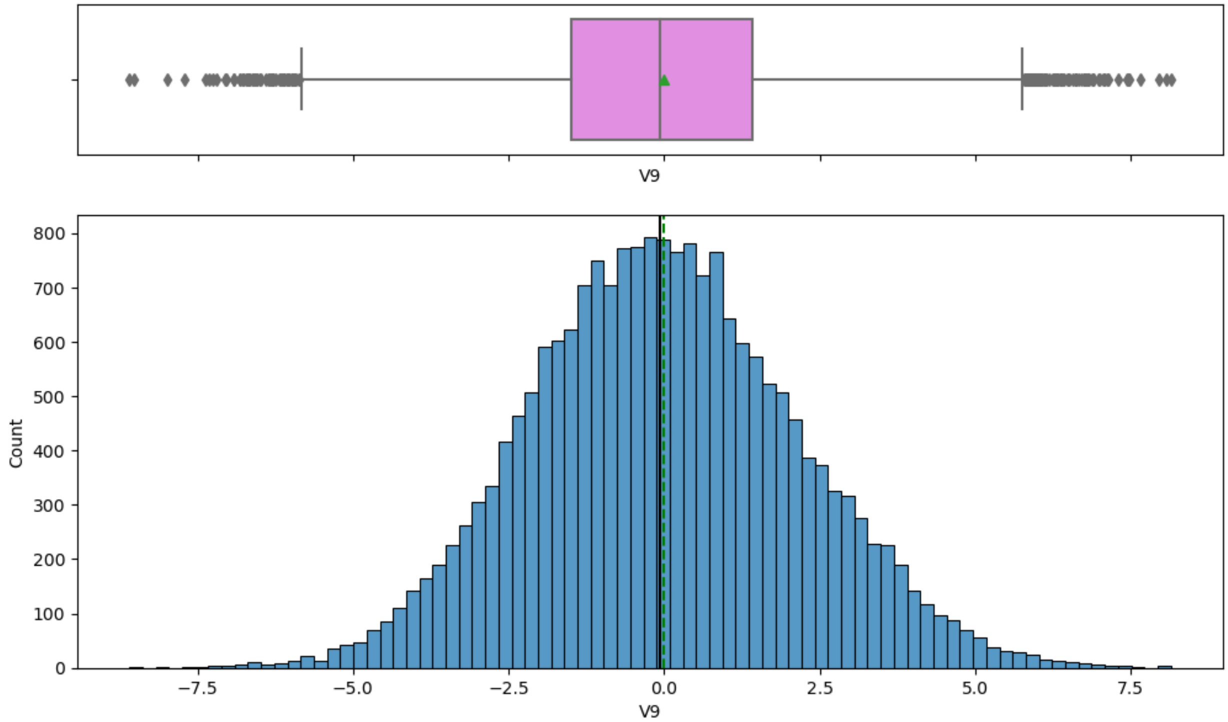

V4
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
V4

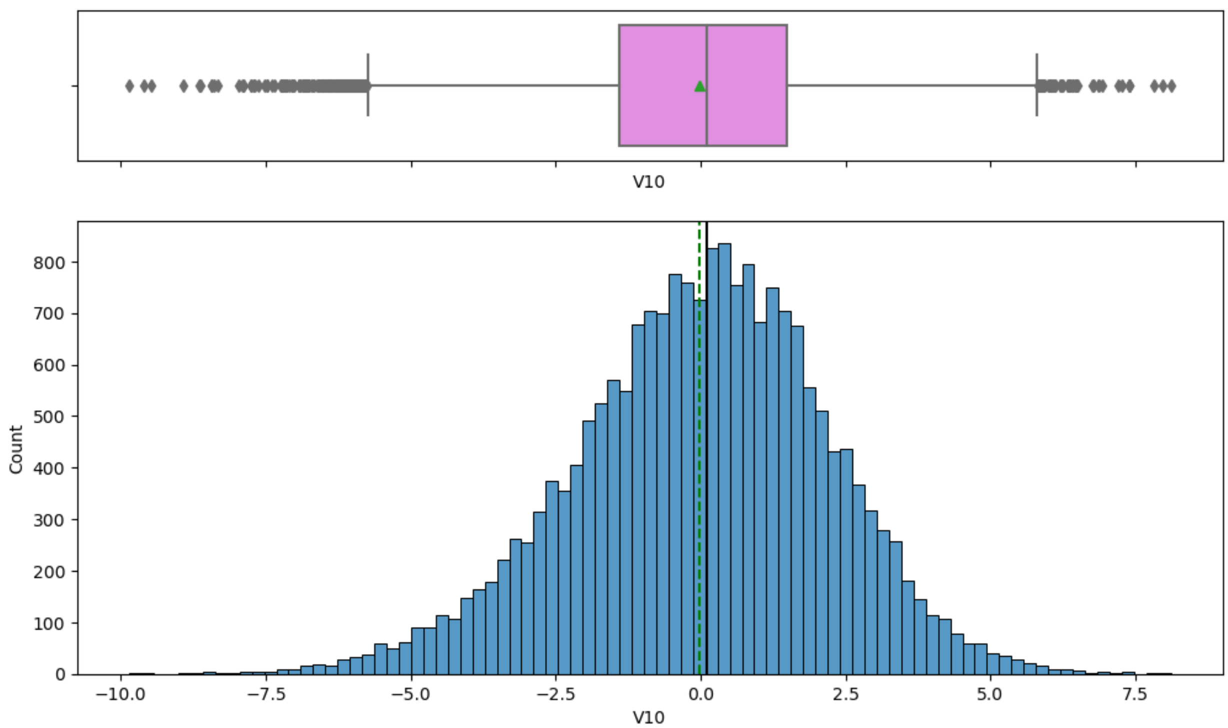

V5
800
700
600
500
400
300
200
100
0
Count
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V5

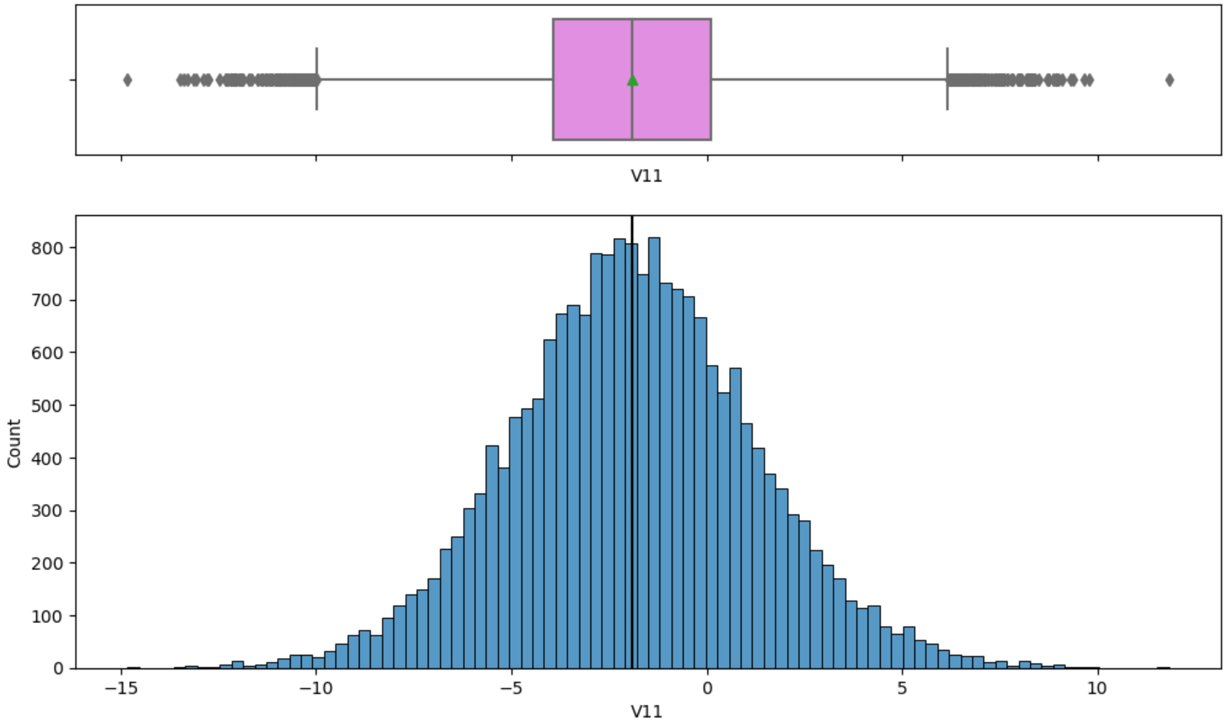

V6
800
700
600
500
400
300
200
100
0
Count
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V6

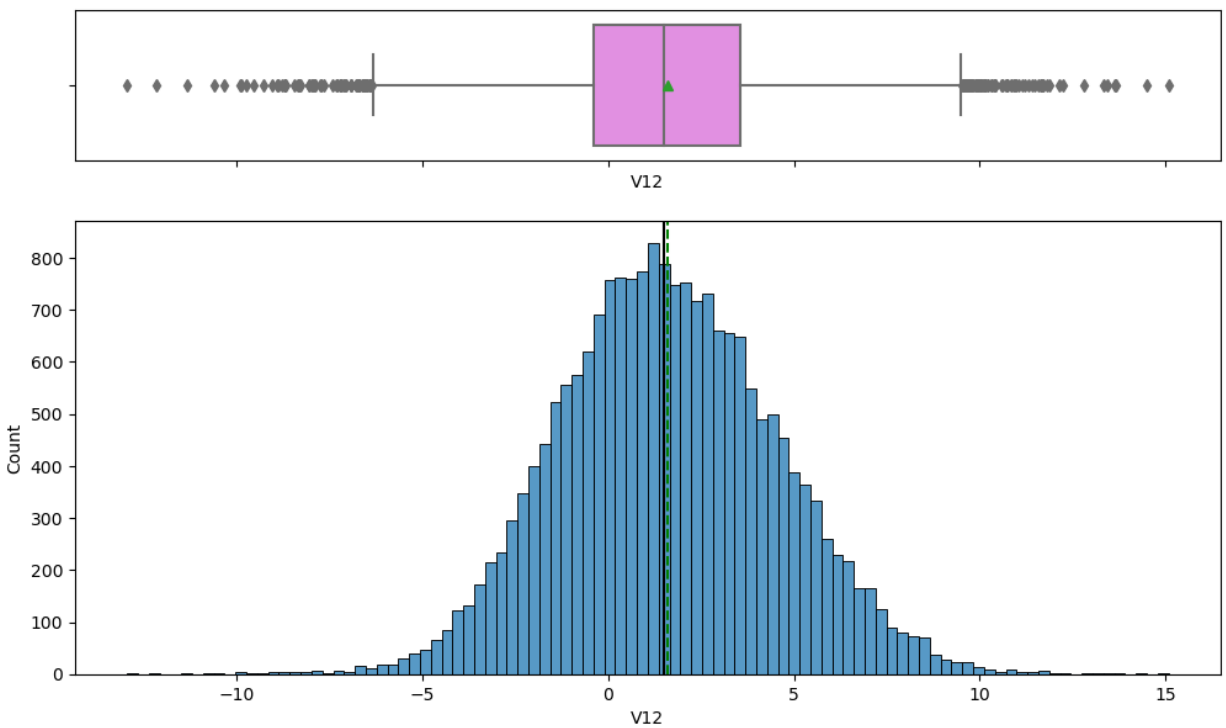

V7
Count
800
700
600
500
400
300
200
100
0
−8
−6
−4
−2
0
2
4
6
8
V7

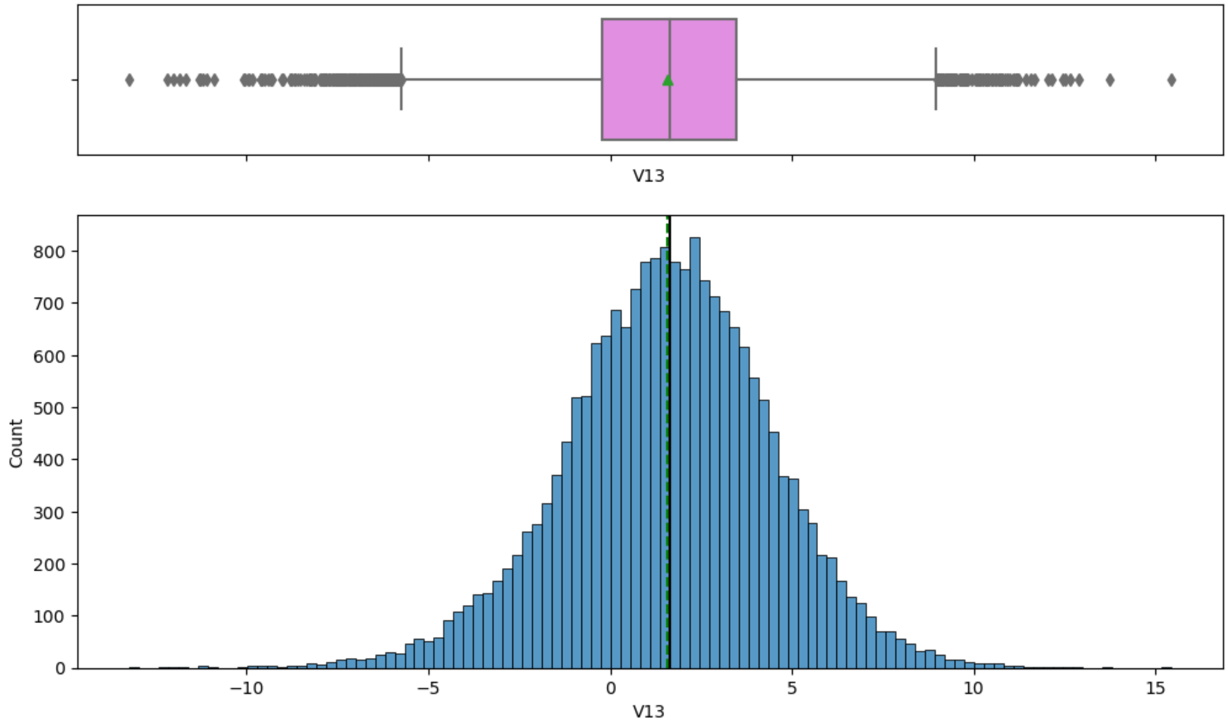

V8
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
V8

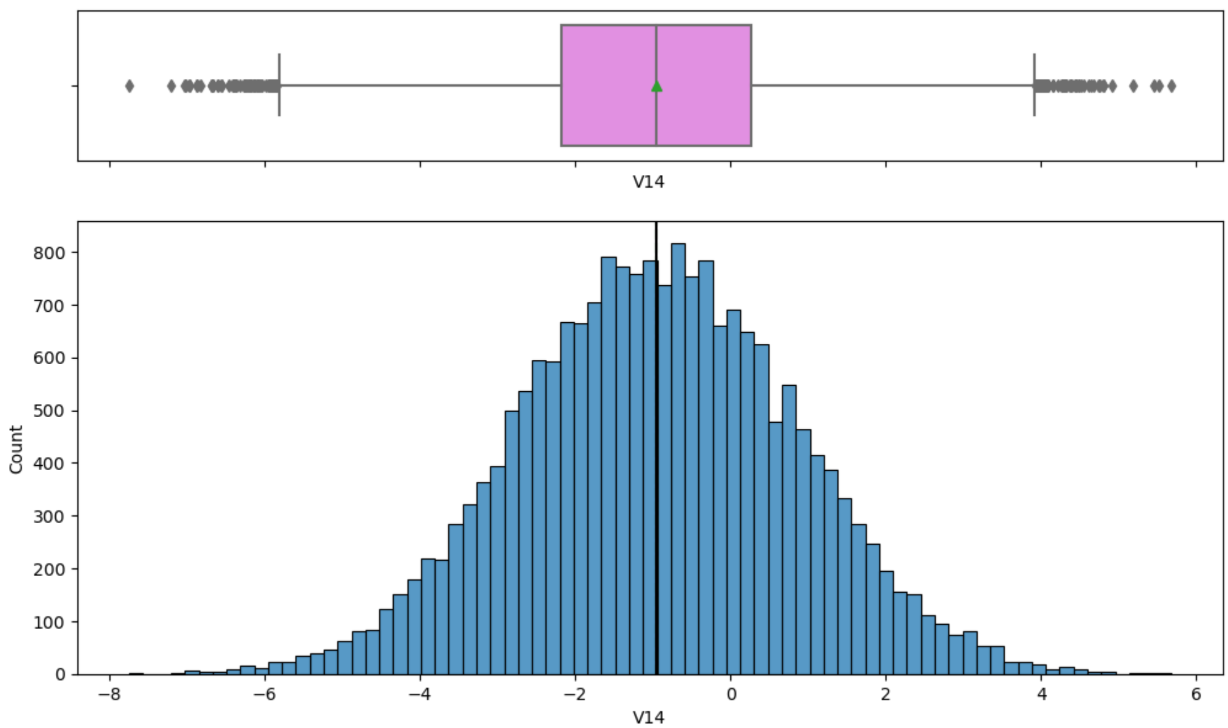

V9
Count
V9

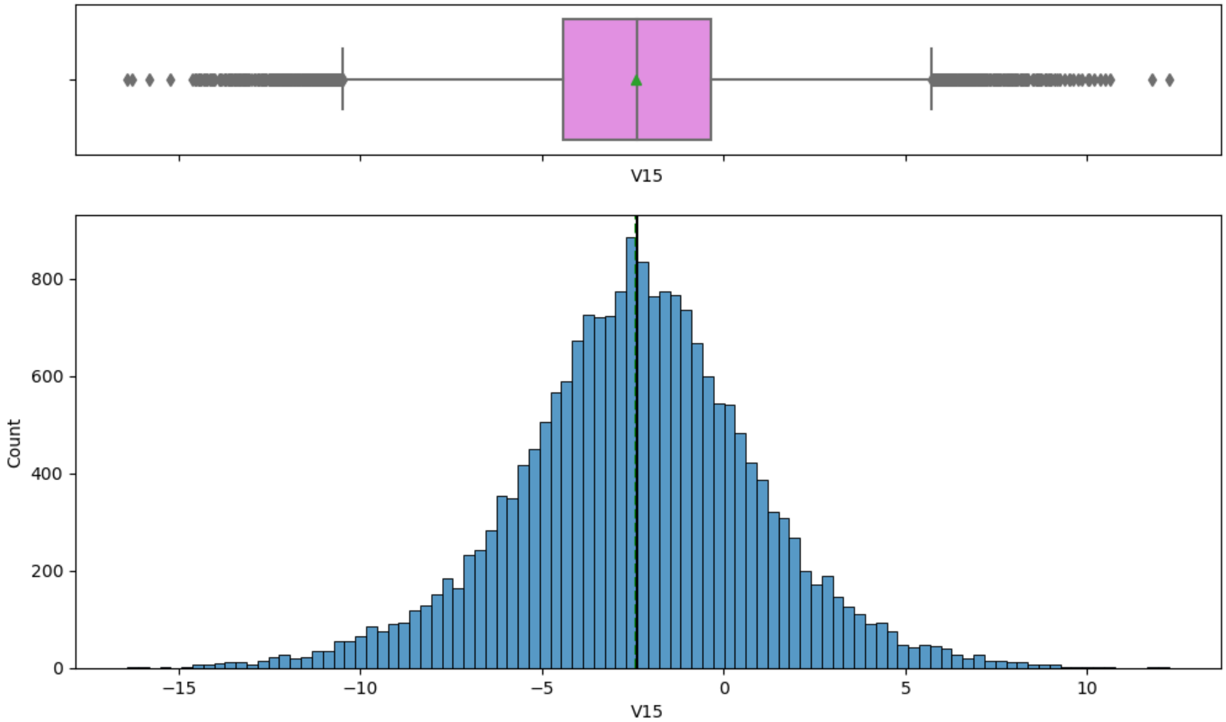

V10
Count
V10

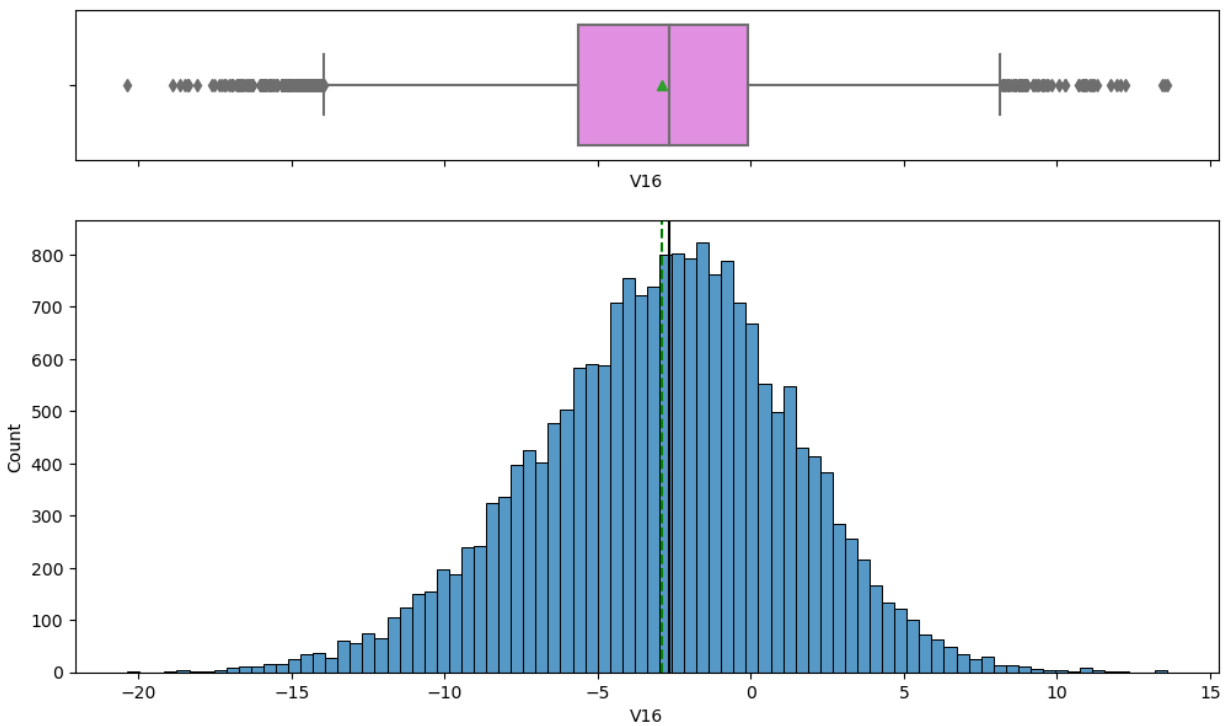

V11
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
V11

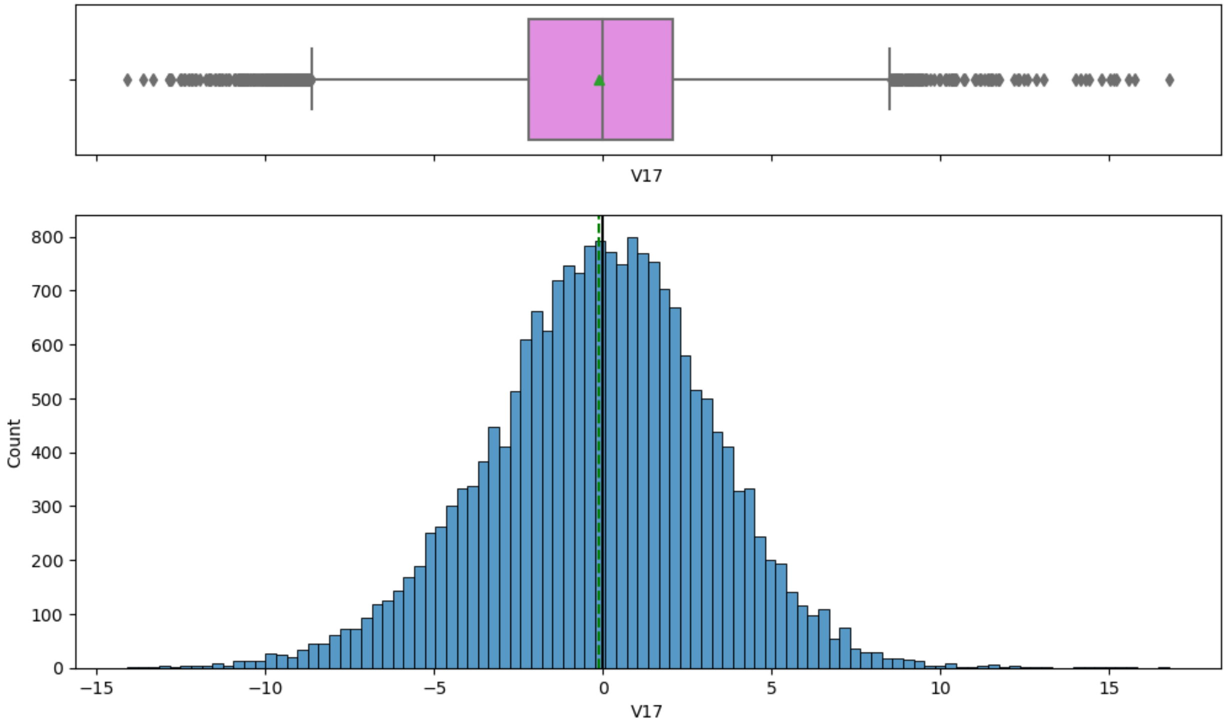

V12
Count
800
700
600
500
400
300
200
100
0
−10
−5
0
5
10
15
V12

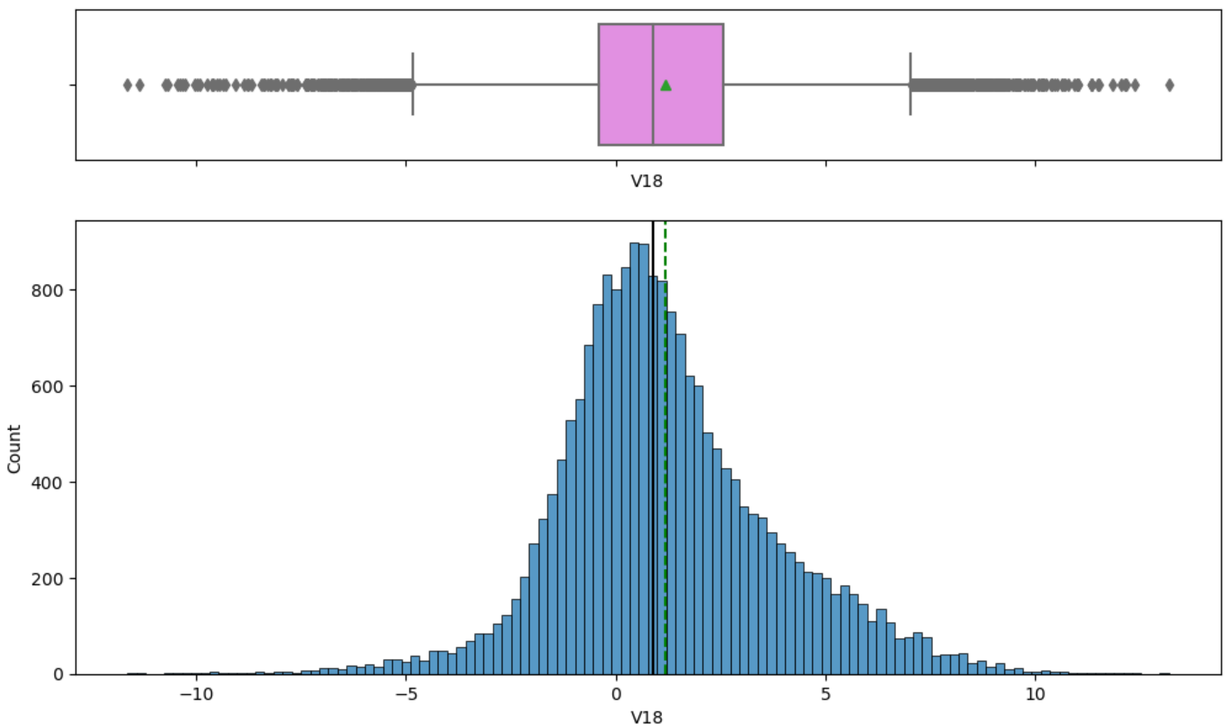

V13
Count
800
700
600
500
400
300
200
100
0
−10
−5
0
5
10
15
V13

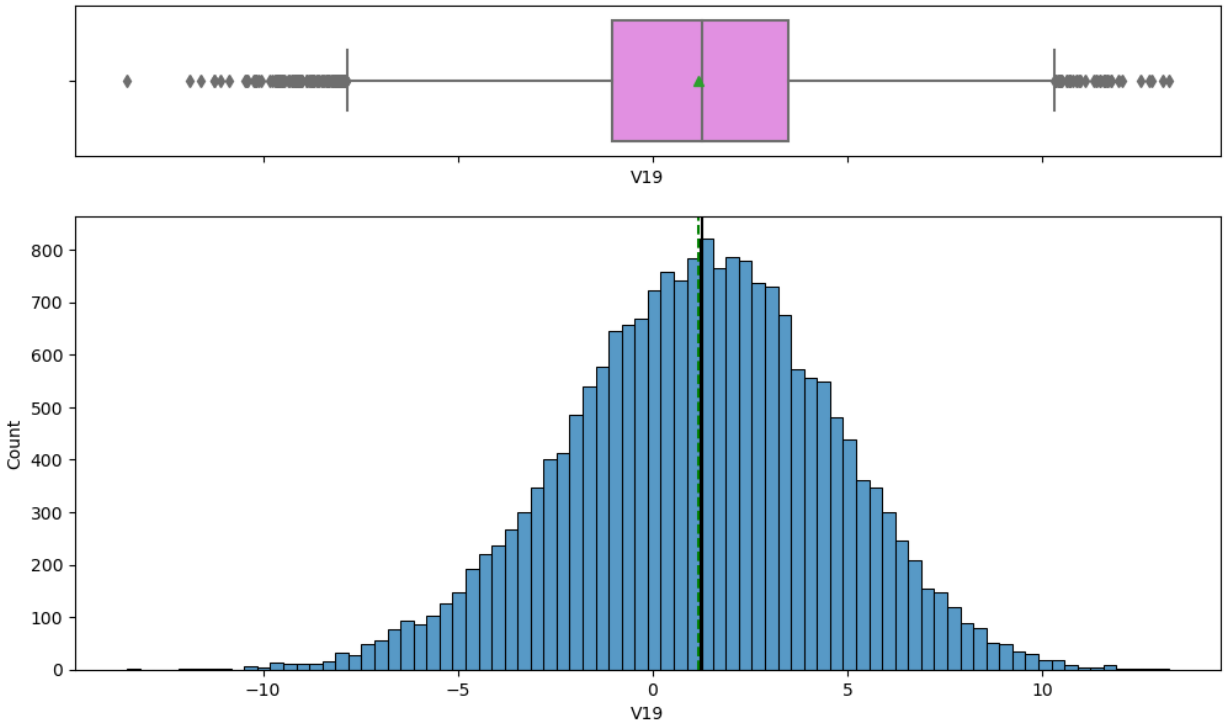

V14
Count
800
700
600
500
400
300
200
100
0
−8
−6
−4
−2
0
2
4
6
V14

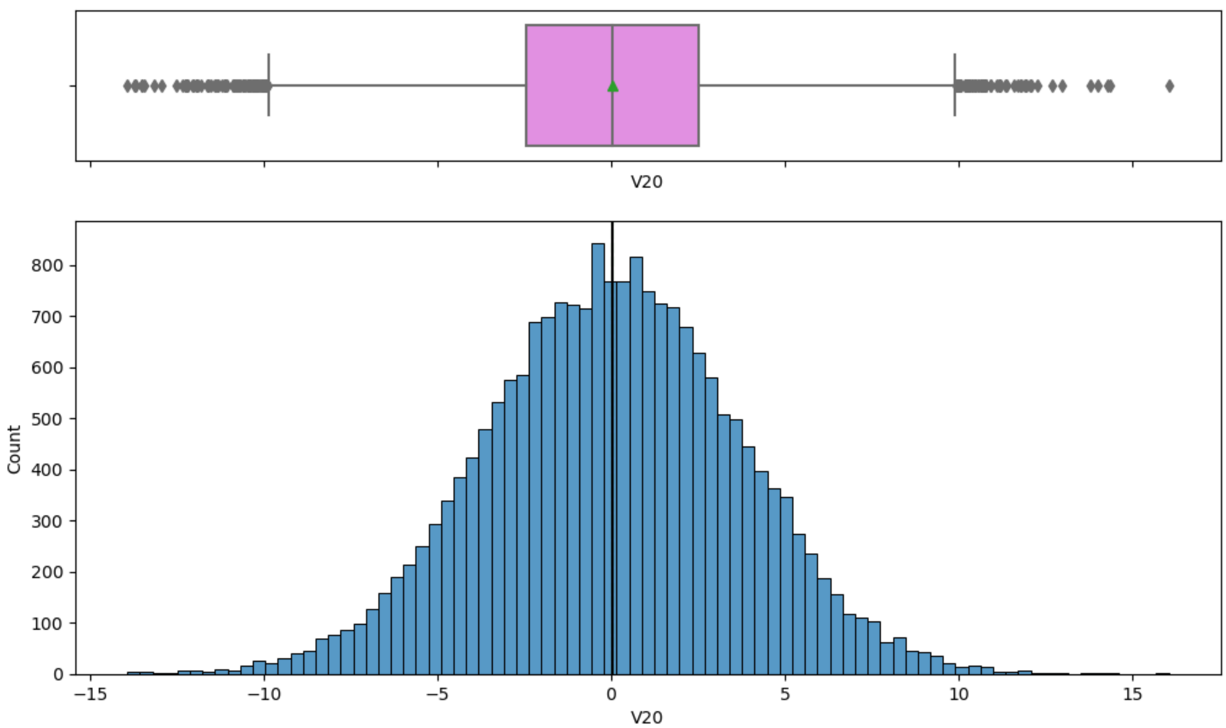

V15
800
600
400
200
0
Count
−15
−10
−5
0
5
10
V15

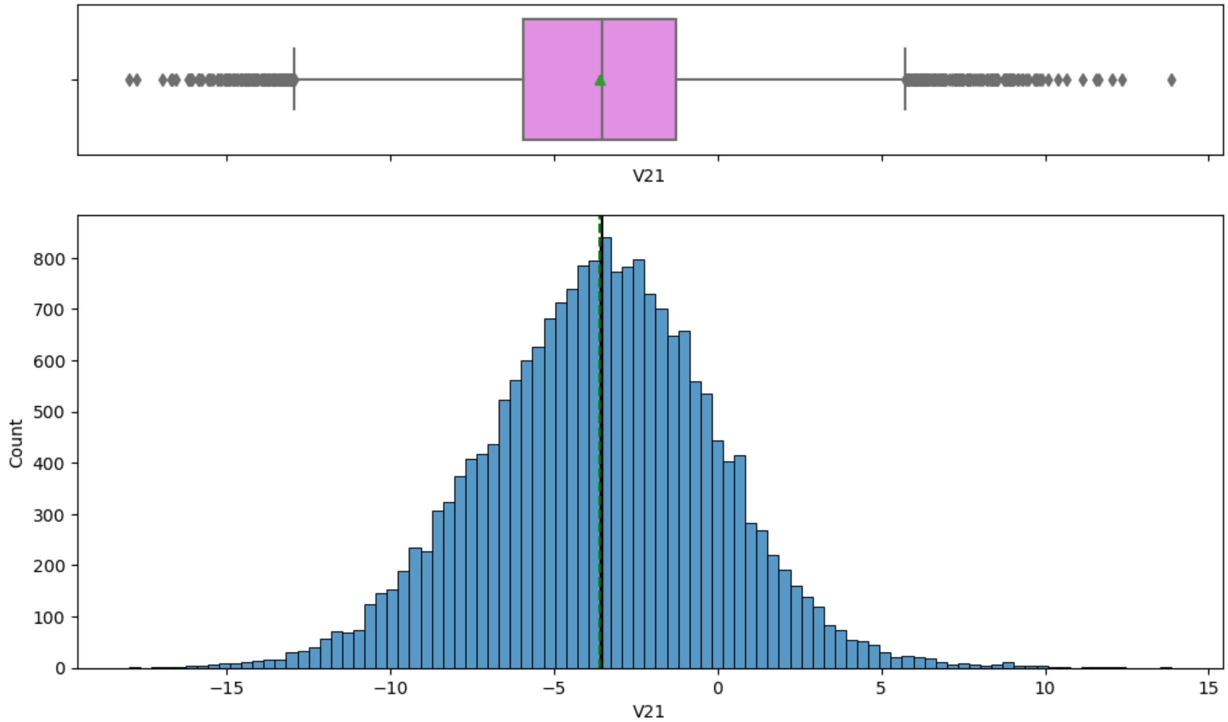

V16
800
700
600
500
400
300
200
100
0
Count
−20
−15
−10
−5
0
5
10
15
V16

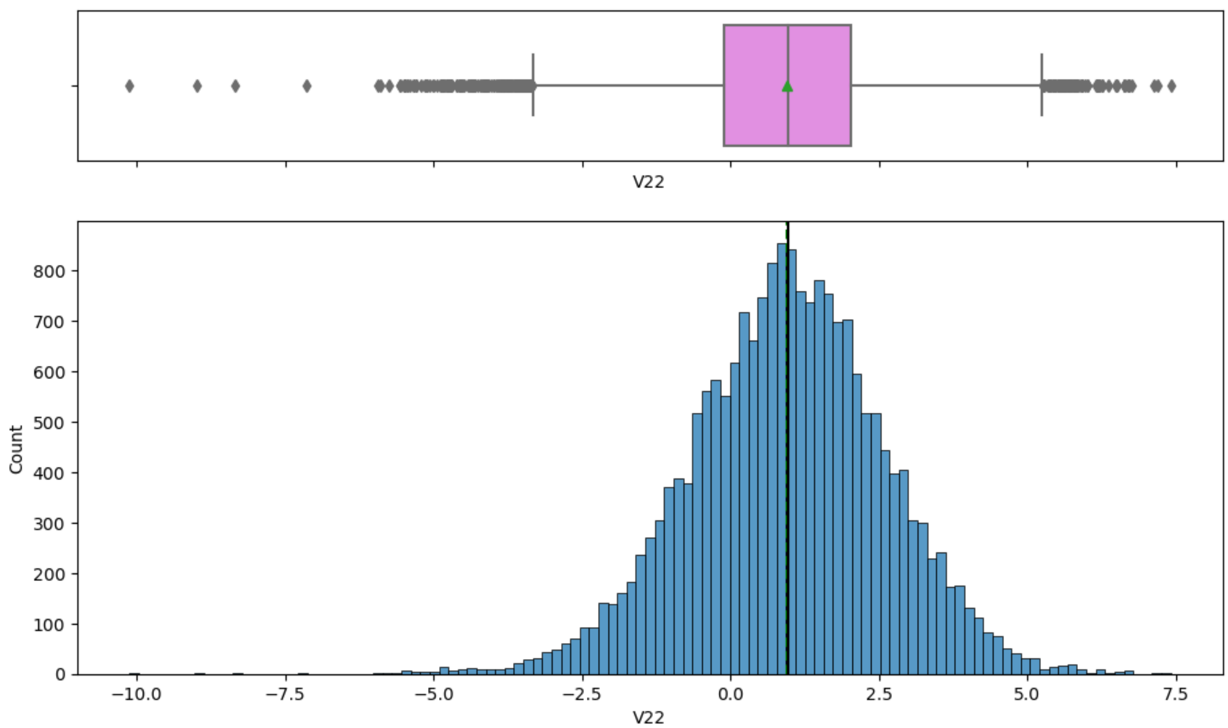

V17
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
15
V17

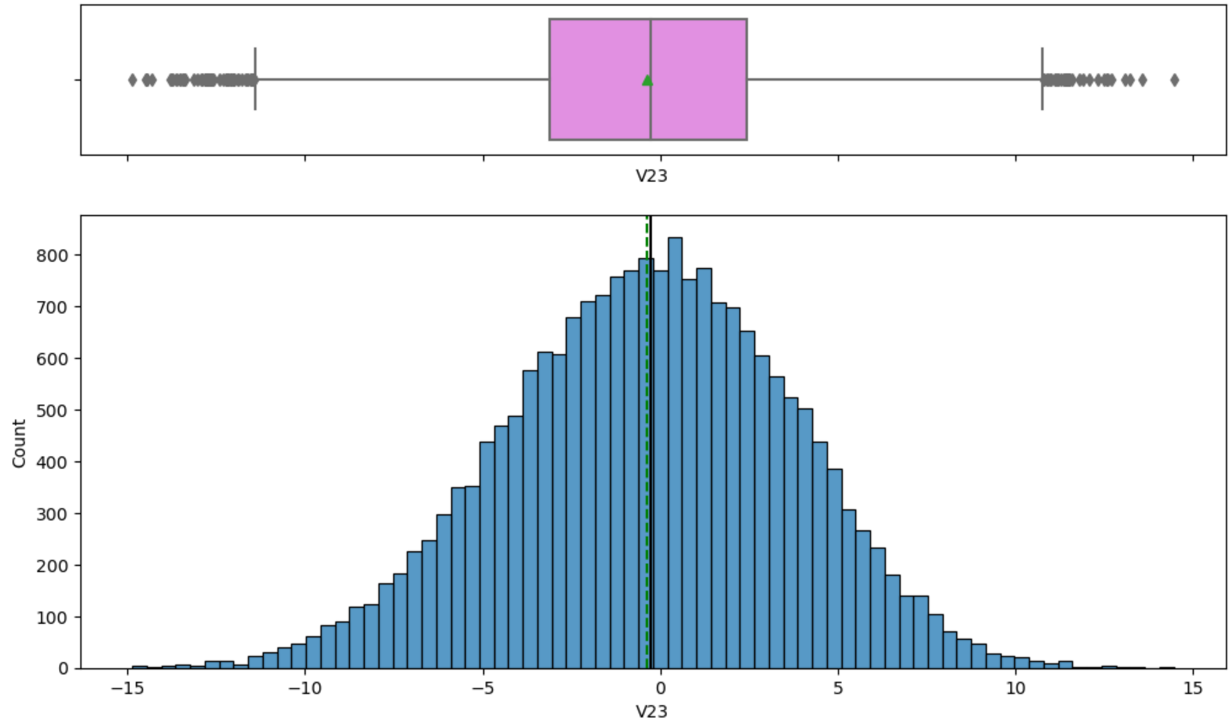

V18
Count
800
600
400
200
0
−10
−5
0
5
10
V18

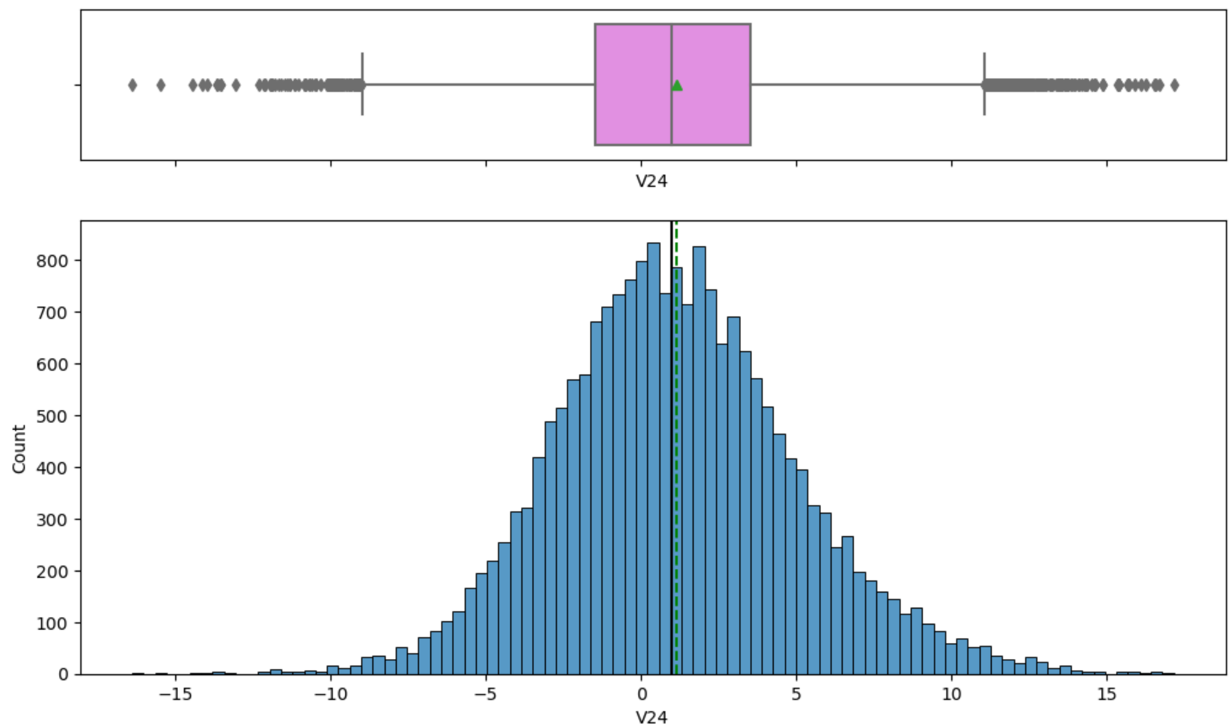

V19
800
700
600
500
400
300
200
100
0
Count
−10
−5
0
5
10
V19

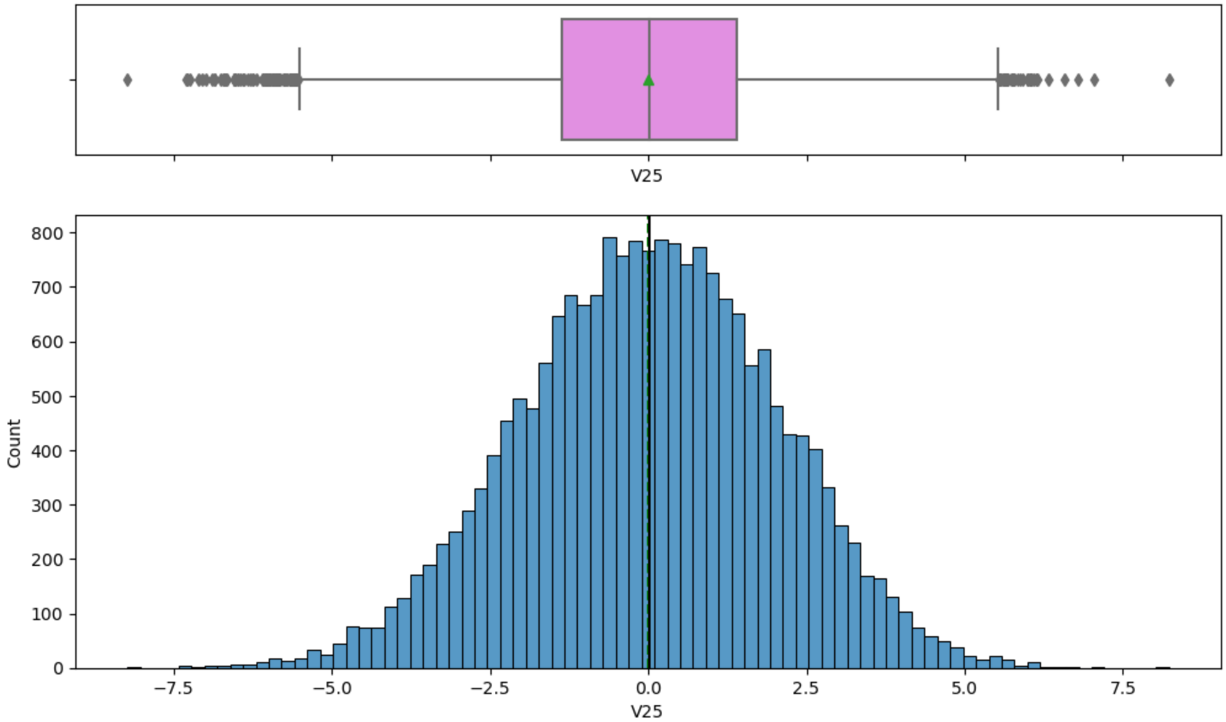

V20
800
700
600
500
400
300
200
100
0
Count
−15
−10
−5
0
5
10
15
V20

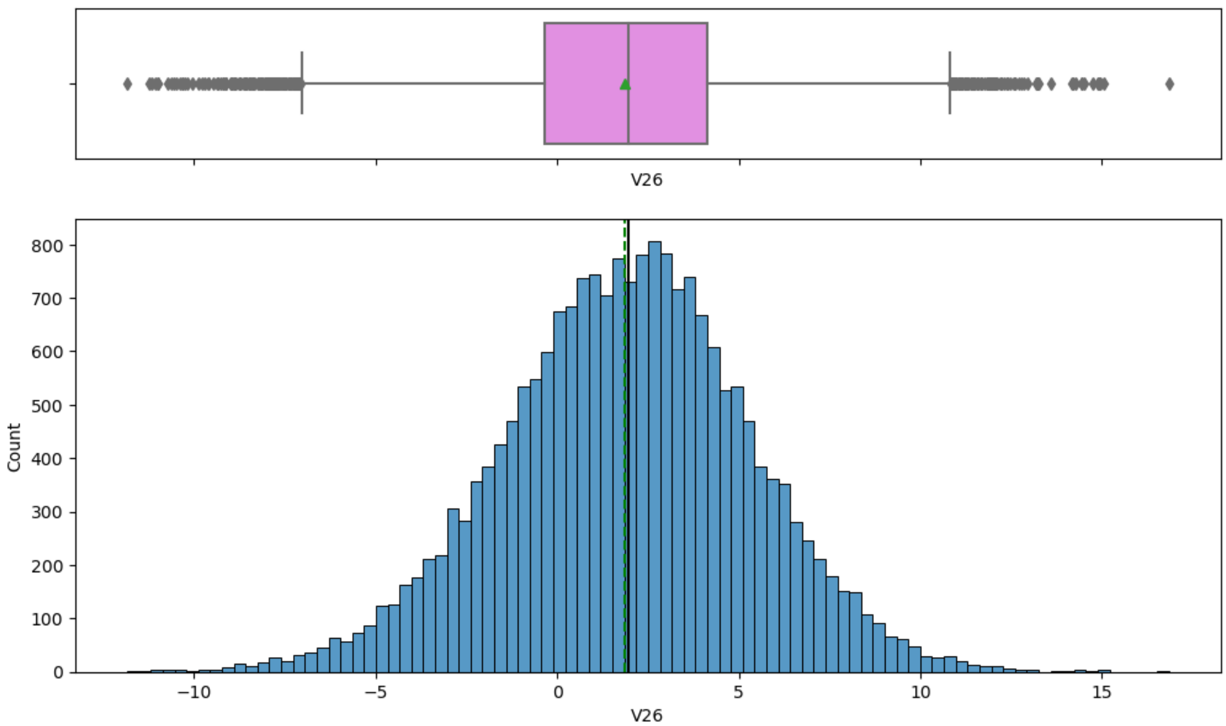

V21
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
15
V21

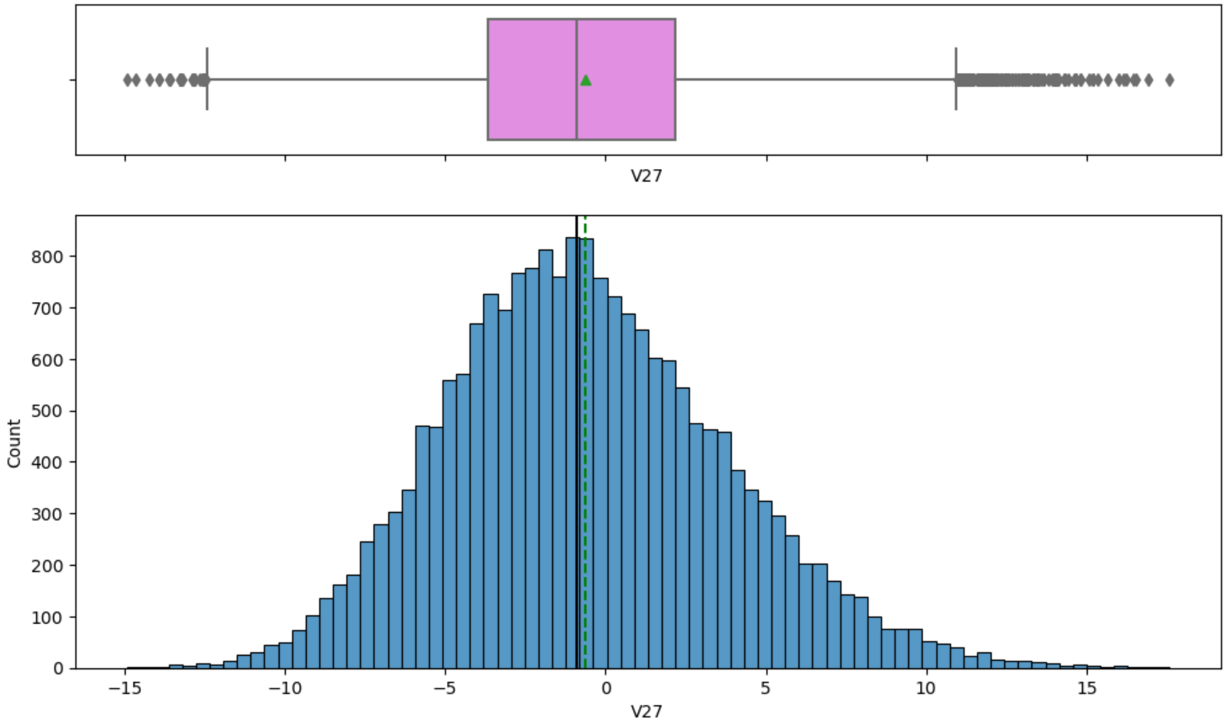

V22
Count
800
700
600
500
400
300
200
100
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V22

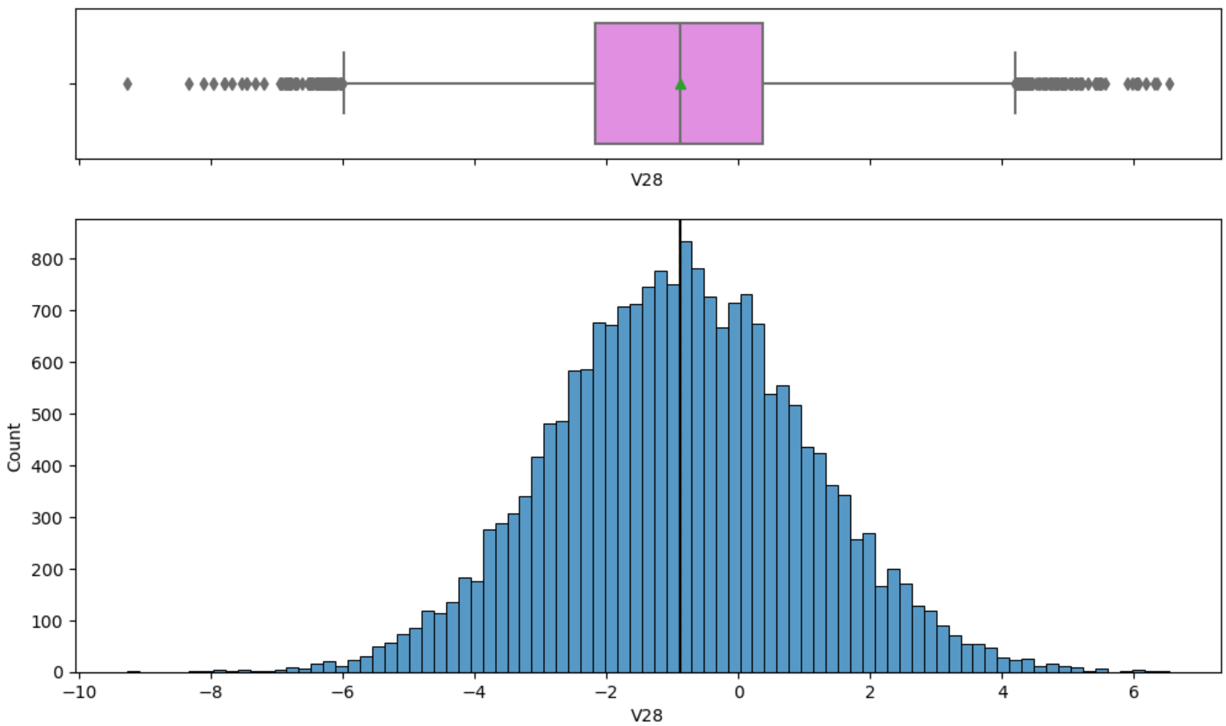

V23
800
700
600
500
400
300
200
100
0
Count
−15
−10
−5
0
5
10
15
V23

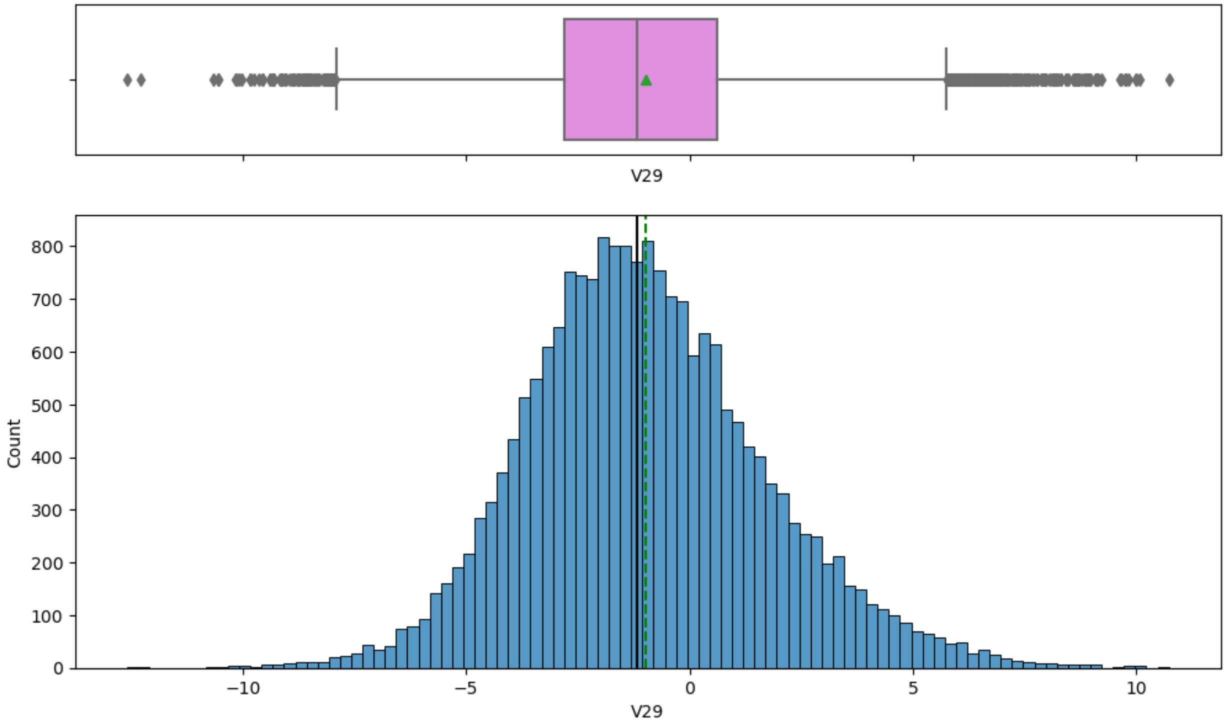

V24
800
700
600
500
400
300
200
100
0
Count
−15
−10
−5
0
5
10
15
V24

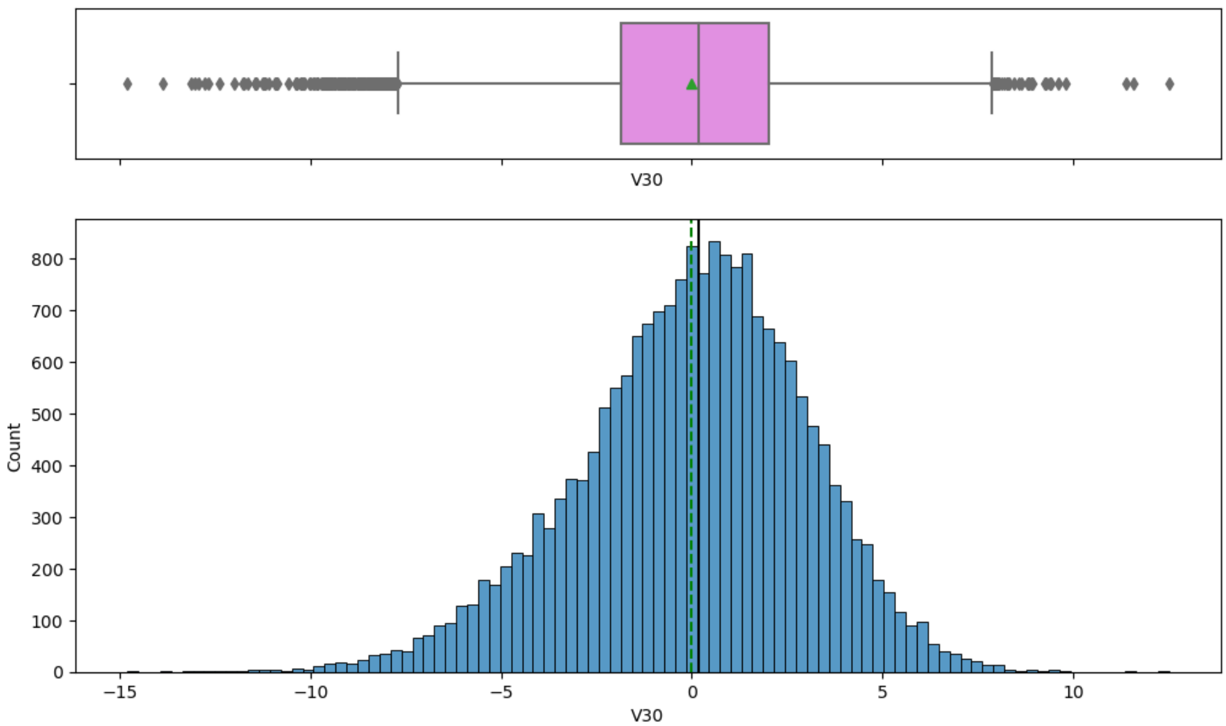

V25
800
700
600
500
400
300
200
100
0
Count
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V25

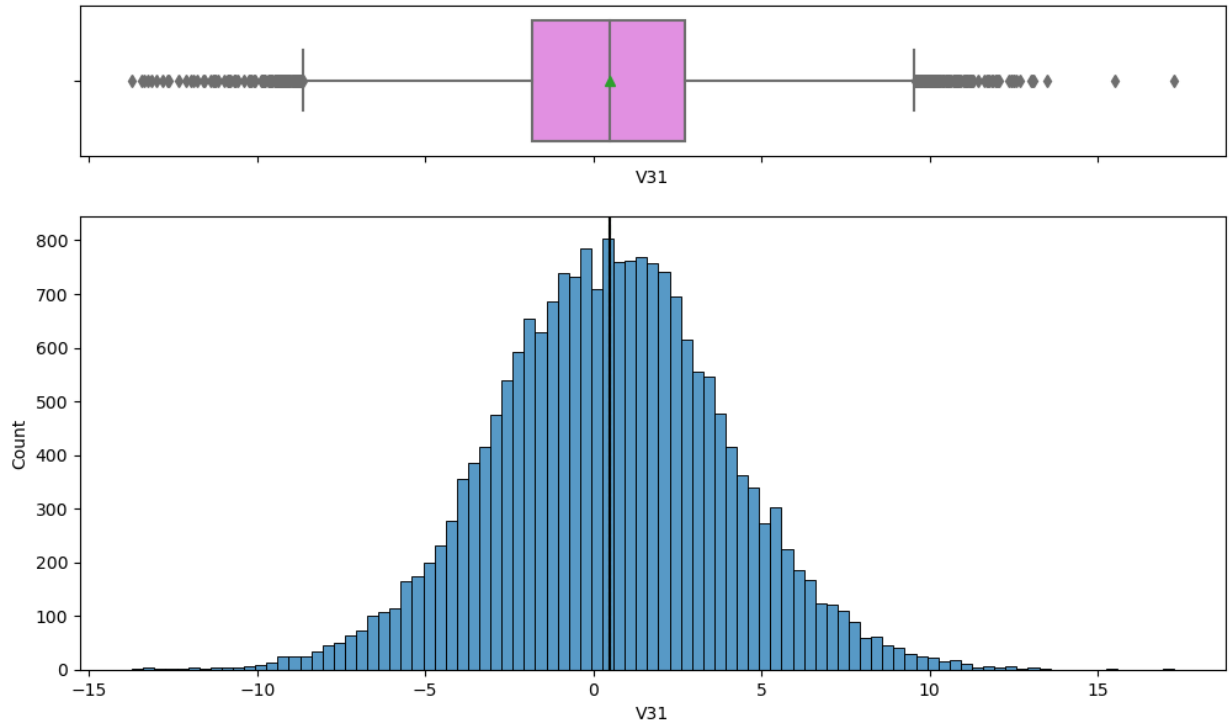

V26
800
700
600
500
400
300
200
100
0
Count
−10
−5
0
5
10
15
V26

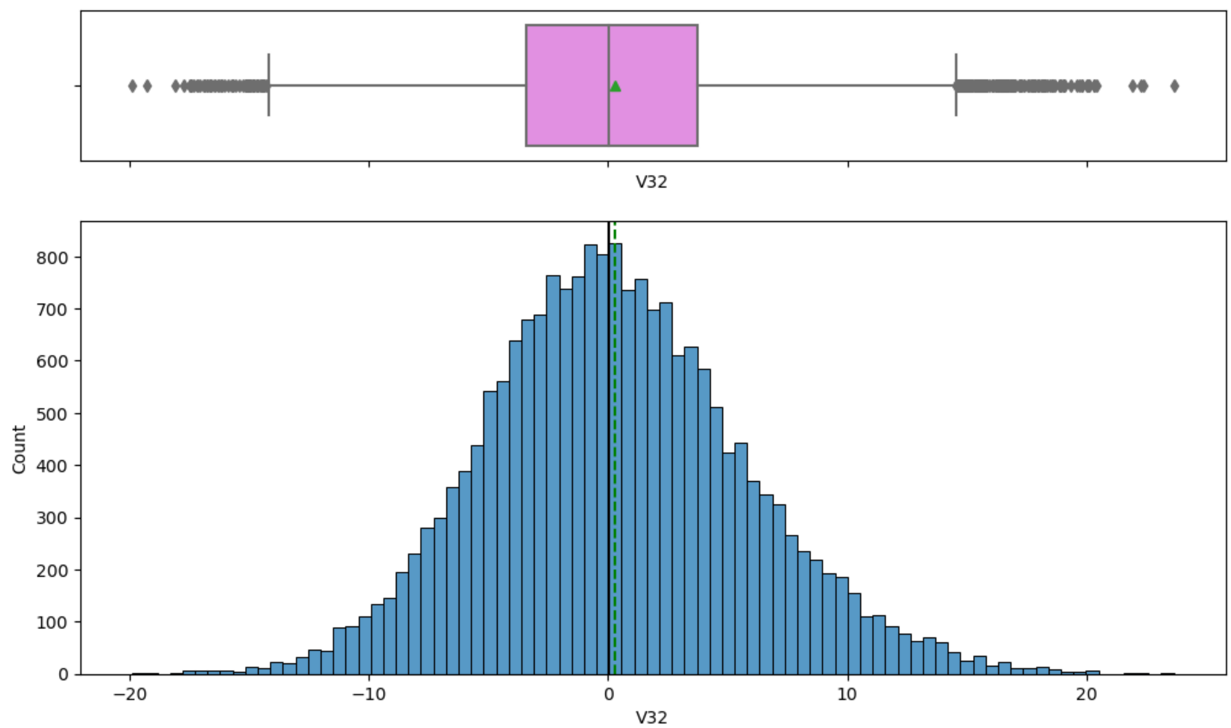

V27
Count
V27

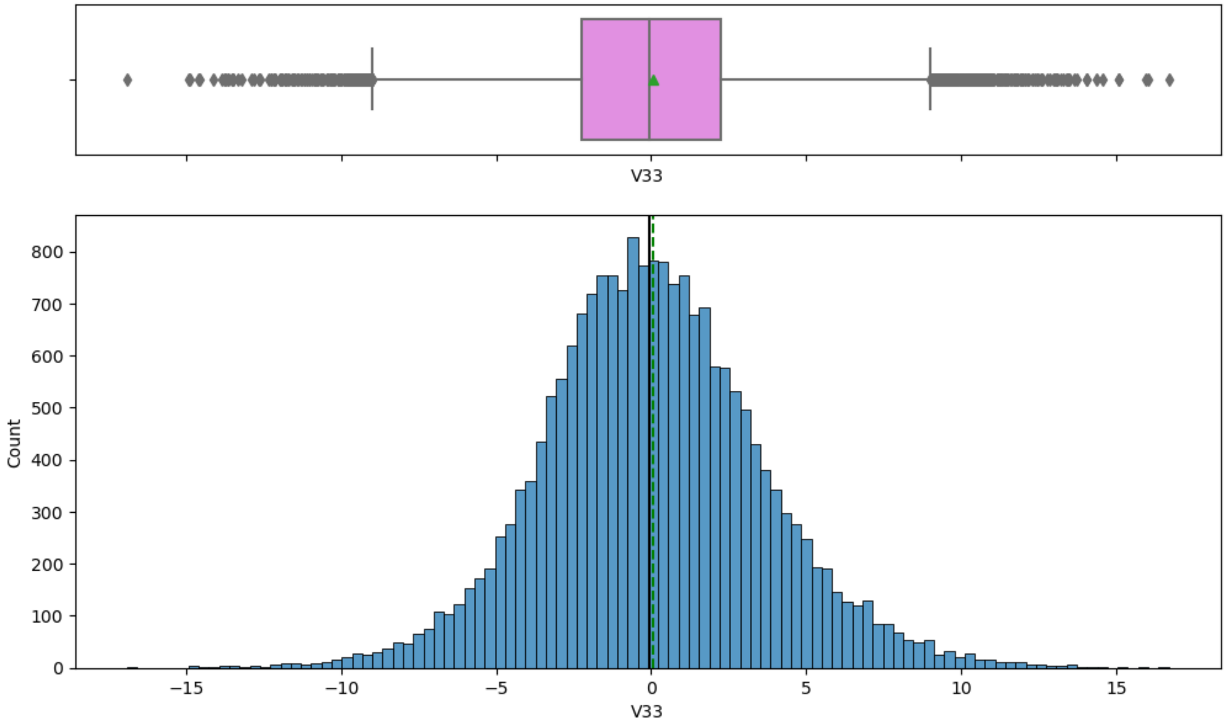

V28
Count
V28

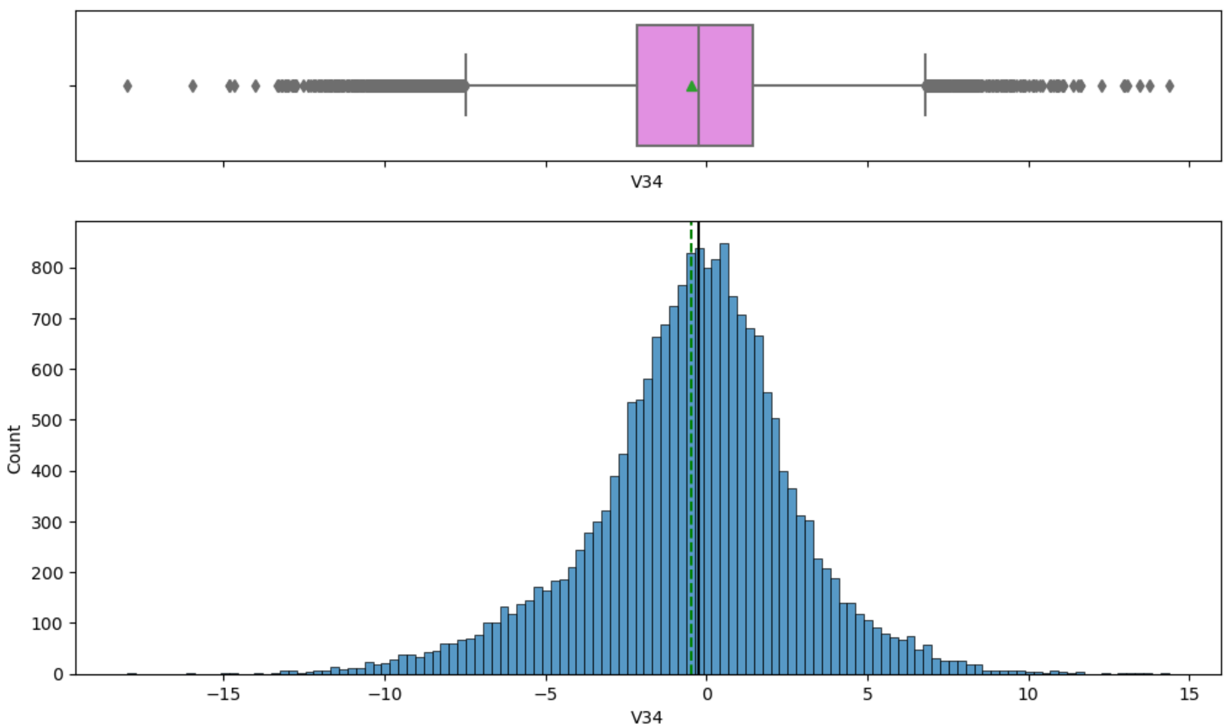

V29
Count
800
700
600
500
400
300
200
100
0
−10
−5
0
5
10
V29

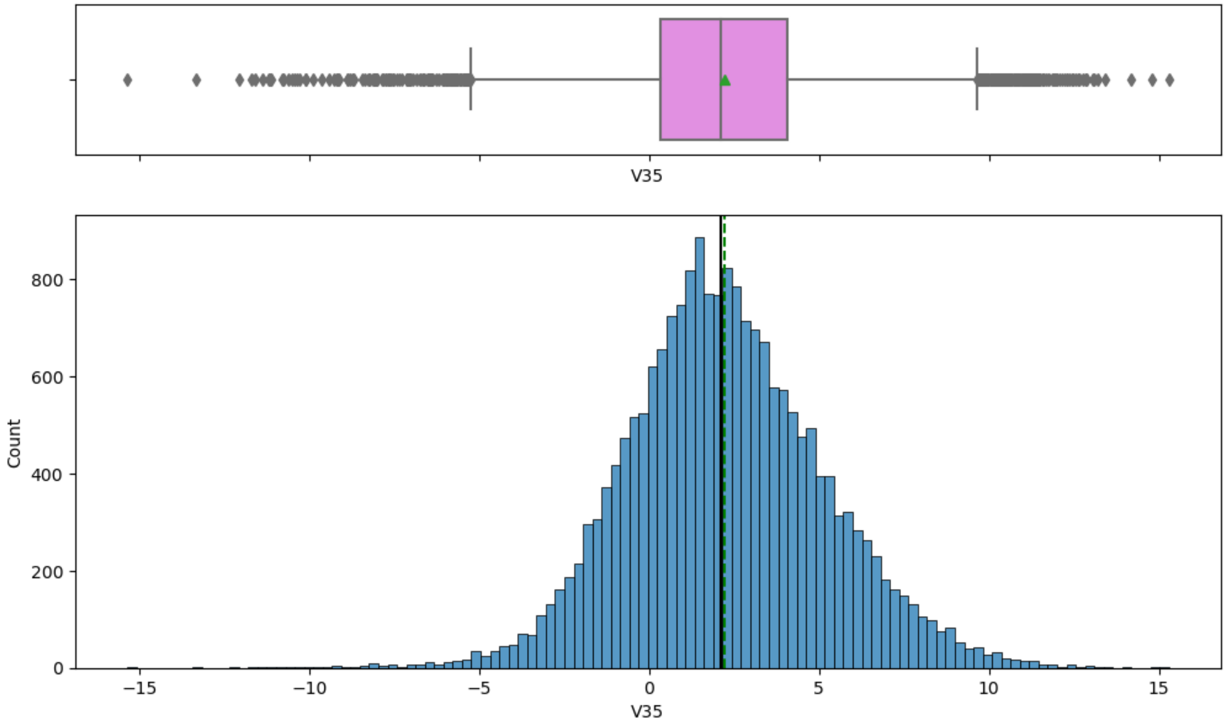

V30
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
V30

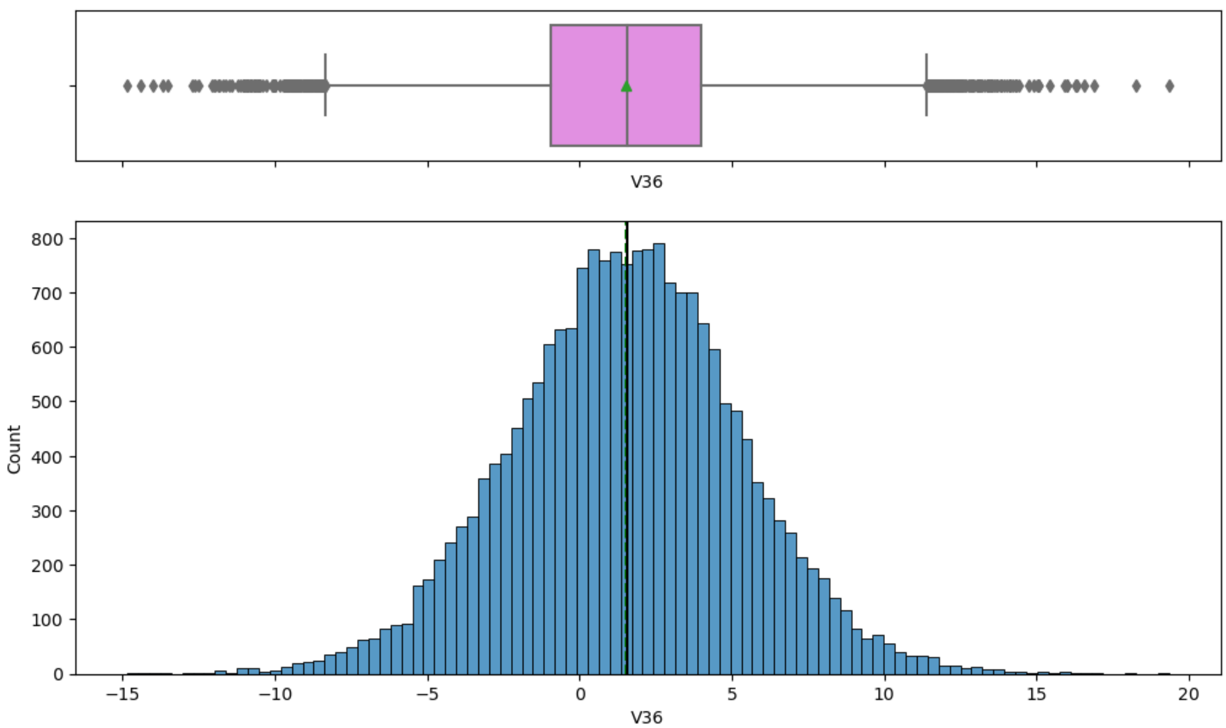

V31
Count
V31

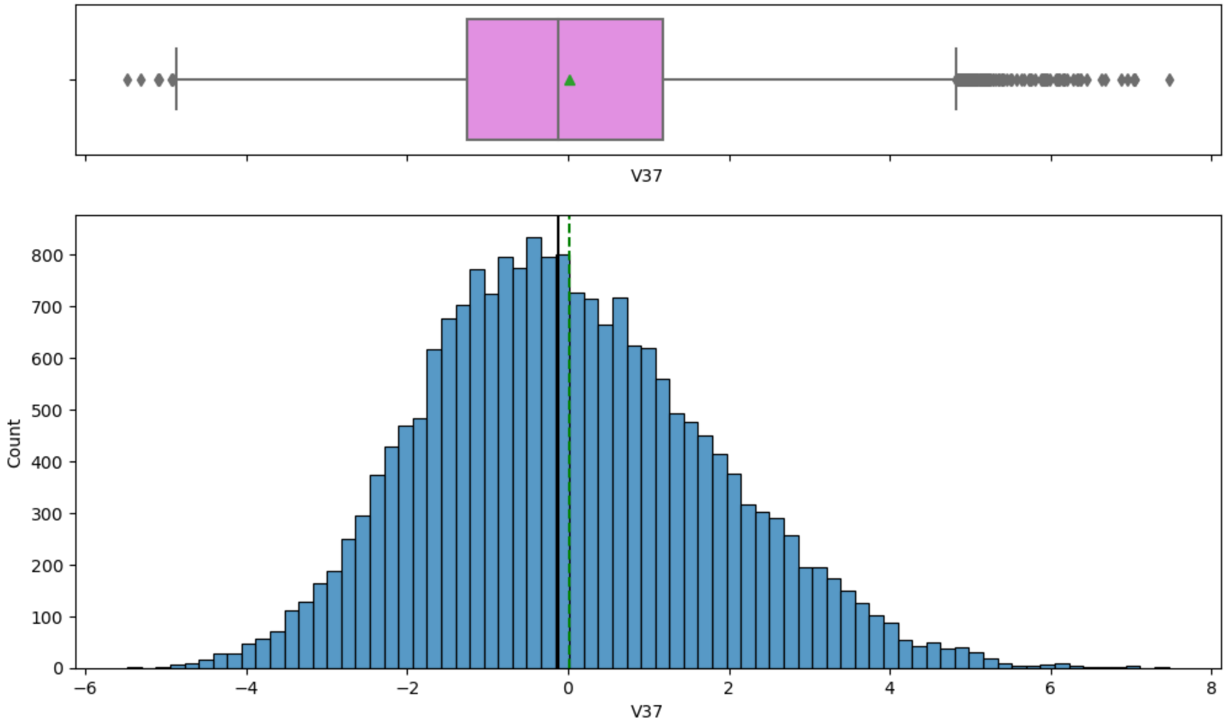

V32
Count
V32

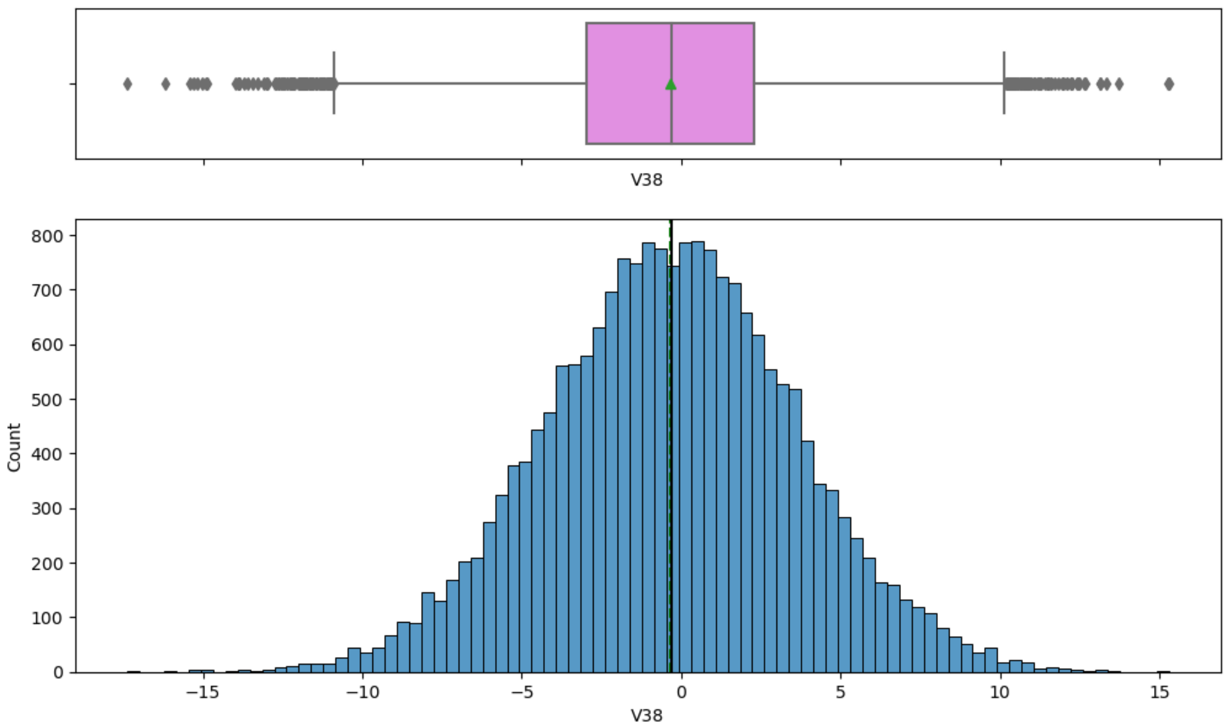

V33
800
700
600
500
400
300
200
100
0
Count
−15
−10
−5
0
5
10
15
V33

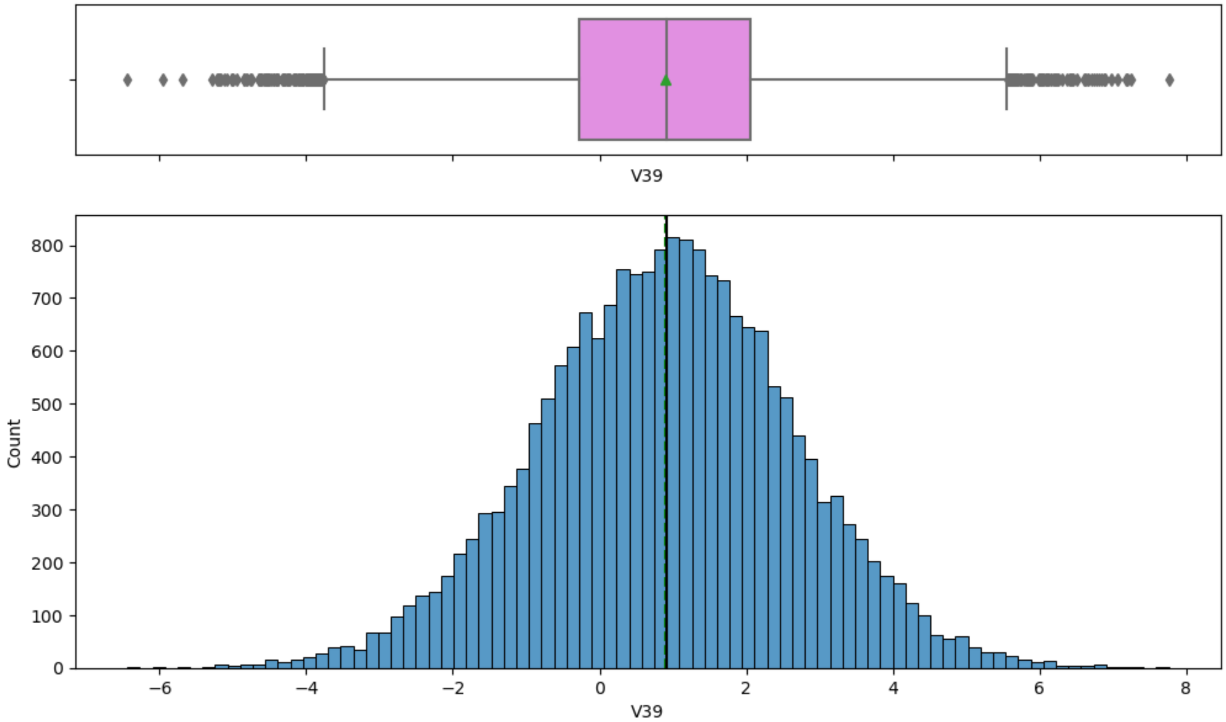

V34
800
700
600
500
400
300
200
100
0
Count
−15
−10
−5
0
5
10
15
V34

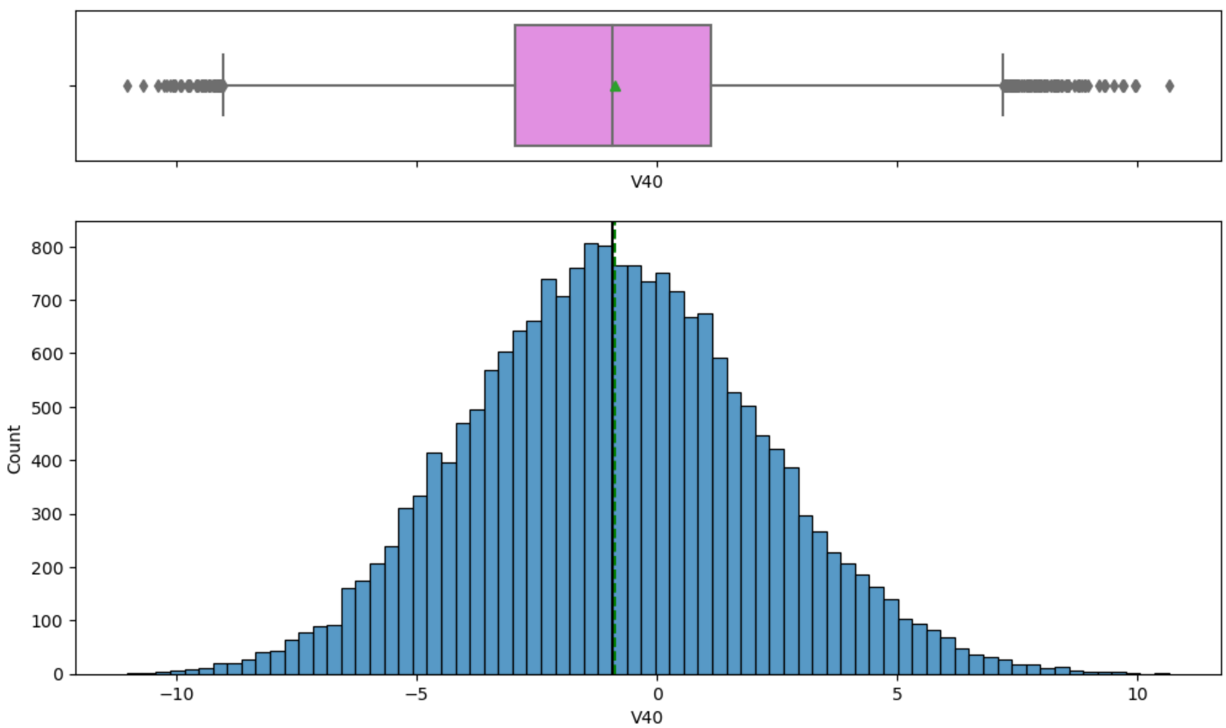

V35
Count
800
600
400
200
0
−15
−10
−5
0
5
10
15
V35

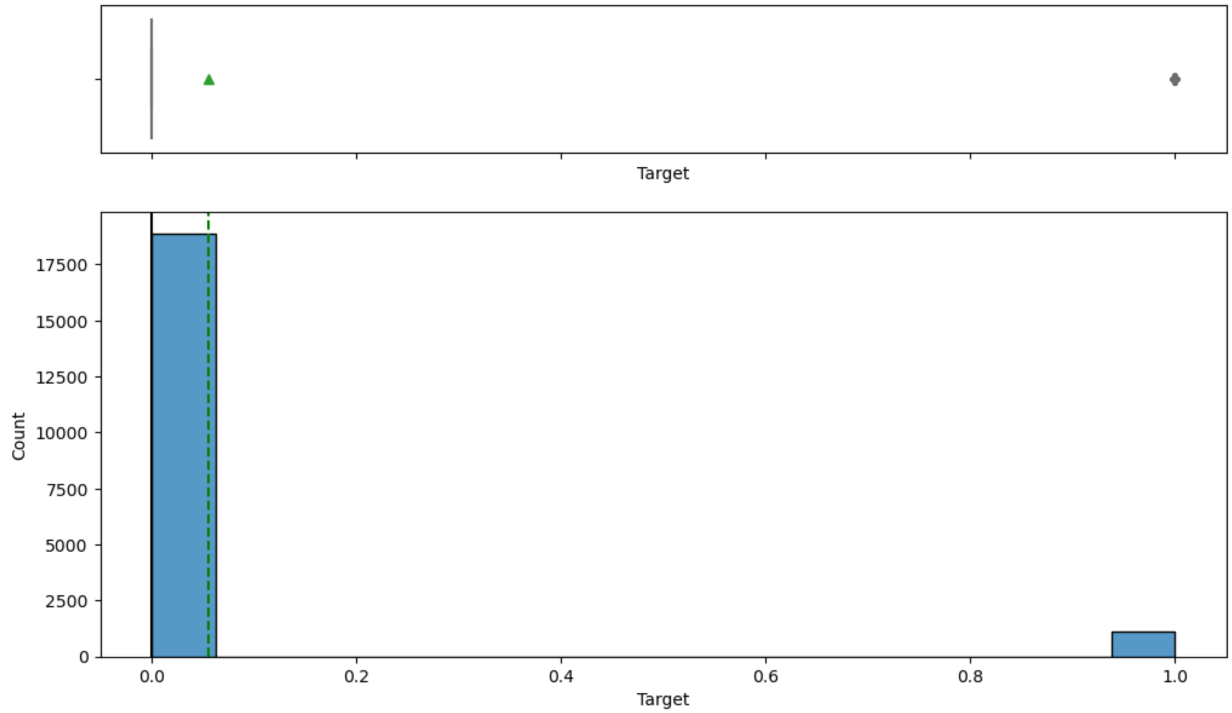

V36
Count
800
700
600
500
400
300
200
100
0
−15
−10
−5
0
5
10
15
20
V36


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


V38
Count
0
100
200
300
400
500
600
700
800
−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
800
700
600
500
400
300
200
100
0
Count
−10
−5
0
5
10
V40



We can see are data is fairly evenlly distributed

In [16]:
```
df["Target"].value_counts()
```

Out[16]:
```
0    18890
1     1110
Name: Target, dtype: int64
```

In [17]:
```
df_test["Target"].value_counts()
```

Out[17]:
```
0    4718
1     282
Name: Target, dtype: int64
```

Next I will preform prepare the data for model building

In [18]:
```
X = df.drop(["Target"], axis=1)
y = df["Target"]
```

In [19]:
```
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.25, random_state=1, stratify=y)
```

In [20]:
```
X_train.shape
```

Out[20]:
```
(15000, 40)
```

In [21]:
```
X_val.shape
```

Out[21]:
```
(5000, 40)
```

In [22]:
```
X_test = df_test.drop(["Target"], axis=1)
y_test = df_test["Target"]
```

In [23]:
```
X_test.shape
```

Out[23]:
```
(5000, 40)
```

In [24]:
```
imp_mode = SimpleImputer(strategy="median")

# fit and transform the imputer on train data
X_train = pd.DataFrame(imp_mode.fit_transform(X_train), columns=X_train.columns)

# Transform on validation and test data
X_val = pd.DataFrame(imp_mode.fit_transform(X_val), columns=X_train.columns)

# fit and transform the imputer on test data
X_test = pd.DataFrame(imp_mode.fit_transform(X_test), columns=X_train.columns)
```

In [25]:
```
print(X_train.isna().sum())
print("-" * 30)
print(X_val.isna().sum())
print("-" * 30)
print(X_test.isna().sum())
```

```
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
------------------------------
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
------------------------------
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
```

In [26]:
```python
def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    TP= confusion_matrix(target, model.predict(predictors))[1,1]
    FP= confusion_matrix(target, model.predict(predictors))[0,1]
    FN= confusion_matrix(target, model.predict(predictors))[1,0]

    pred = model.predict(predictors)

    acc = accuracy_score(target, pred)
    recall = recall_score(target, pred)
    precision = precision_score(target, pred)
    f1 = f1_score(target, pred)

    df_perf = pd.DataFrame(
        {
            "Accuracy": acc,
            "Recall": recall,
            "Precision": precision,
            "F1": f1,
        },
        index=[0],
    )

    return df_perf
```

In [27]:
```python
def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    y_pred = model.predict(predictors)
    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum())]
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

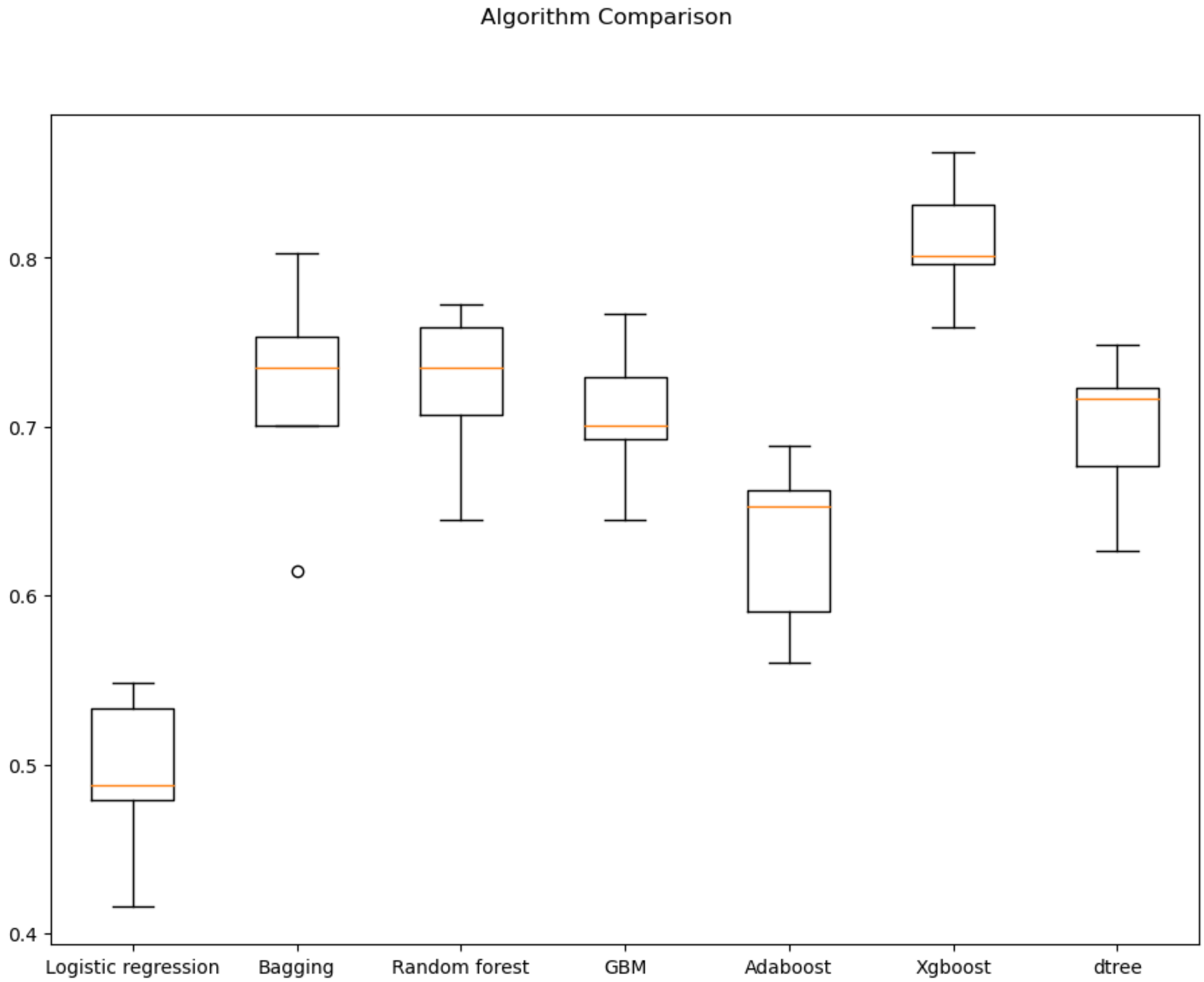
In [28]:
```
scorer = metrics.make_scorer(metrics.recall_score)
```

In [29]:
```
models = []

models.append(("Logistic regression", LogisticRegression(random_state=1)))
models.append(("Bagging", BaggingClassifier(random_state=1)))
models.append(("Random forest", RandomForestClassifier(random_state=1)))
models.append(("GBM", GradientBoostingClassifier(random_state=1)))
models.append(("Adaboost", AdaBoostClassifier(random_state=1)))
models.append(("Xgboost", XGBClassifier(random_state=1, eval_metric="logloss")))
models.append(("dtree", DecisionTreeClassifier(random_state=1)))

results = []
names = []

print("\n" "Cross-Validation Performance:" "\n")
for name, model in models:
    scoring = "recall"
    kfold = StratifiedKFold(
        n_splits=5, shuffle=True, random_state=1
    )
    cv_result = cross_val_score(
        estimator=model, X=X_train, y=y_train, scoring=scoring, cv=kfold
    )
    results.append(cv_result)
    names.append(name)
    print("{}: {}".format(name, cv_result.mean() * 100))

print("\n" "Validation Performance:" "\n")

for name, model in models:
    model.fit(X_train, y_train)
    scores = recall_score(y_val, model.predict(X_val))
    print("{}: {}".format(name, scores))
```

```
Cross-Validation Performance:

Logistic regression: 49.27566553639709
Bagging: 72.1080730106053
Random forest: 72.35192266070268
GBM: 70.66661857008873
Adaboost: 63.09140754635308
Xgboost: 81.00497799581561
dtree: 69.82829521679533

Validation Performance:

Logistic regression: 0.48201438848920863
Bagging: 0.7302158273381295
Random forest: 0.7266187050359713
GBM: 0.7230215827338129
Adaboost: 0.6762589928057554
Xgboost: 0.8309352517985612
dtree: 0.7050359712230215
```

In [30]:
```
fig = plt.figure(figsize=(11,8))

fig.suptitle("Algorithm Comparison")
ax = fig.add_subplot(111)
```

```
plt.boxplot(results)
ax.set_xticklabels(names)

plt.show()
```



In [ ]:

```
The best preforming model now was xgboost
```

In [31]:

```
print("Before UpSampling, counts of label 'Yes': {}".format(sum(y_train == 1)))
print("Before UpSampling, counts of label 'No': {} \n".format(sum(y_train == 0)))

sm = SMOTE(
    sampling_strategy=1, k_neighbors=5, random_state=1
)  # Synthetic Minority Over Sampling Technique
X_train_over, y_train_over = sm.fit_resample(X_train, y_train)


print("After UpSampling, counts of label 'Yes': {}".format(sum(y_train_over == 1)))
print("After UpSampling, counts of label 'No': {} \n".format(sum(y_train_over == 0)))


print("After UpSampling, the shape of train_X: {}".format(X_train_over.shape))
print("After UpSampling, the shape of train_y: {} \n".format(y_train_over.shape))
```

```
Before UpSampling, counts of label 'Yes': 832
Before UpSampling, counts of label 'No': 14168

After UpSampling, counts of label 'Yes': 14168
After UpSampling, counts of label 'No': 14168

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

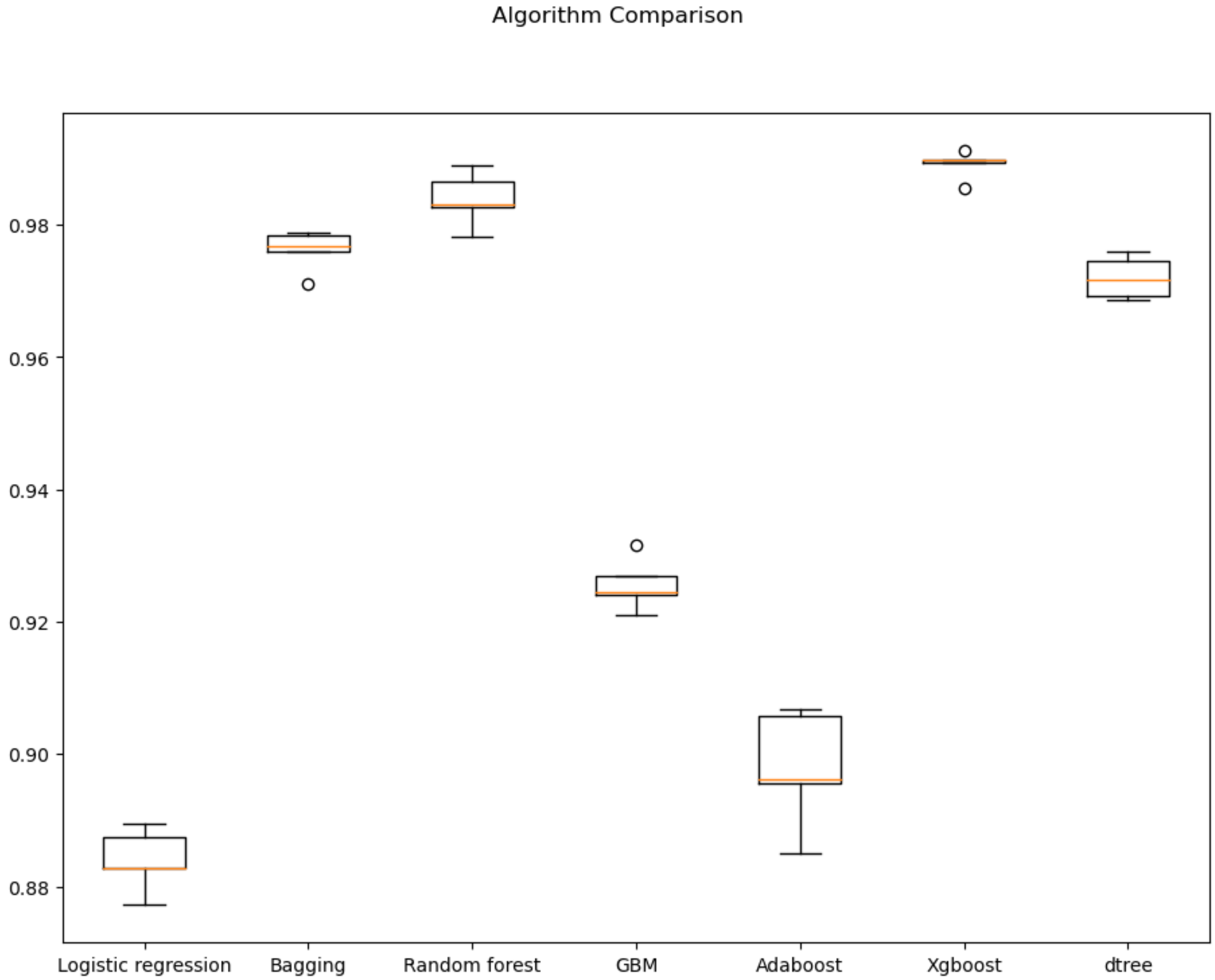
Next I will build my overfitting models

In [32]:
```python
models = []

models.append(("Logistic regression", LogisticRegression(random_state=1)))
models.append(("Bagging", BaggingClassifier(random_state=1)))
models.append(("Random forest", RandomForestClassifier(random_state=1)))
models.append(("GBM", GradientBoostingClassifier(random_state=1)))
models.append(("Adaboost", AdaBoostClassifier(random_state=1)))
models.append(("Xgboost", XGBClassifier(random_state=1, eval_metric="logloss")))
models.append(("dtree", DecisionTreeClassifier(random_state=1)))

results = []
names = []

print("\n" "Cross-Validation Performance:" "\n")
for name, model in models:
    scoring = "recall"
    kfold = StratifiedKFold(
        n_splits=5, shuffle=True, random_state=1
    )
    cv_result = cross_val_score(
        estimator=model, X=X_train_over, y=y_train_over, scoring=scoring, cv=kfold
    )
    results.append(cv_result)
    names.append(name)
    print("{}: {}".format(name, cv_result.mean() * 100))

print("\n" "Validation Performance:" "\n")

for name, model in models:
    model.fit(X_train_over, y_train_over)
    scores = recall_score(y_val, model.predict(X_val))
    print("{}: {}".format(name, scores))
```

```
Cross-Validation Performance:

Logistic regression: 88.3963699328486
Bagging: 97.62141471581656
Random forest: 98.39075260047615
GBM: 92.56068151319724
Adaboost: 89.78689011775472
Xgboost: 98.91305241357217
dtree: 97.20494245534968

Validation Performance:

Logistic regression: 0.8489208633093526
Bagging: 0.8345323741007195
Random forest: 0.8489208633093526
GBM: 0.8776978417266187
Adaboost: 0.8561151079136691
Xgboost: 0.8669064748201439
dtree: 0.7769784172661871
```

In [33]:

```python
fig = plt.figure(figsize=(11,8))

fig.suptitle("Algorithm Comparison")
ax = fig.add_subplot(111)

plt.boxplot(results)
ax.set_xticklabels(names)

plt.show()
```



My best preforming models now were adaboost and GBM

In [34]:
```
rus = RandomUnderSampler(random_state=1)
X_train_un, y_train_un = rus.fit_resample(X_train, y_train)
```

In [35]:
```
print("Before Under Sampling, counts of label 'Yes': {}".format(sum(y_train == 1)))
print("Before Under Sampling, counts of label 'No': {} \n".format(sum(y_train == 0)))

print("After Under Sampling, counts of label 'Yes': {}".format(sum(y_train_un == 1)))
print("After Under Sampling, counts of label 'No': {} \n".format(sum(y_train_un == 0)))

print("After Under Sampling, the shape of train_X: {}".format(X_train_un.shape))
print("After Under Sampling, the shape of train_y: {} \n".format(y_train_un.shape))
```

```
Before Under Sampling, counts of label 'Yes': 832
Before Under Sampling, counts of label 'No': 14168

After Under Sampling, counts of label 'Yes': 832
After Under Sampling, counts of label 'No': 832

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

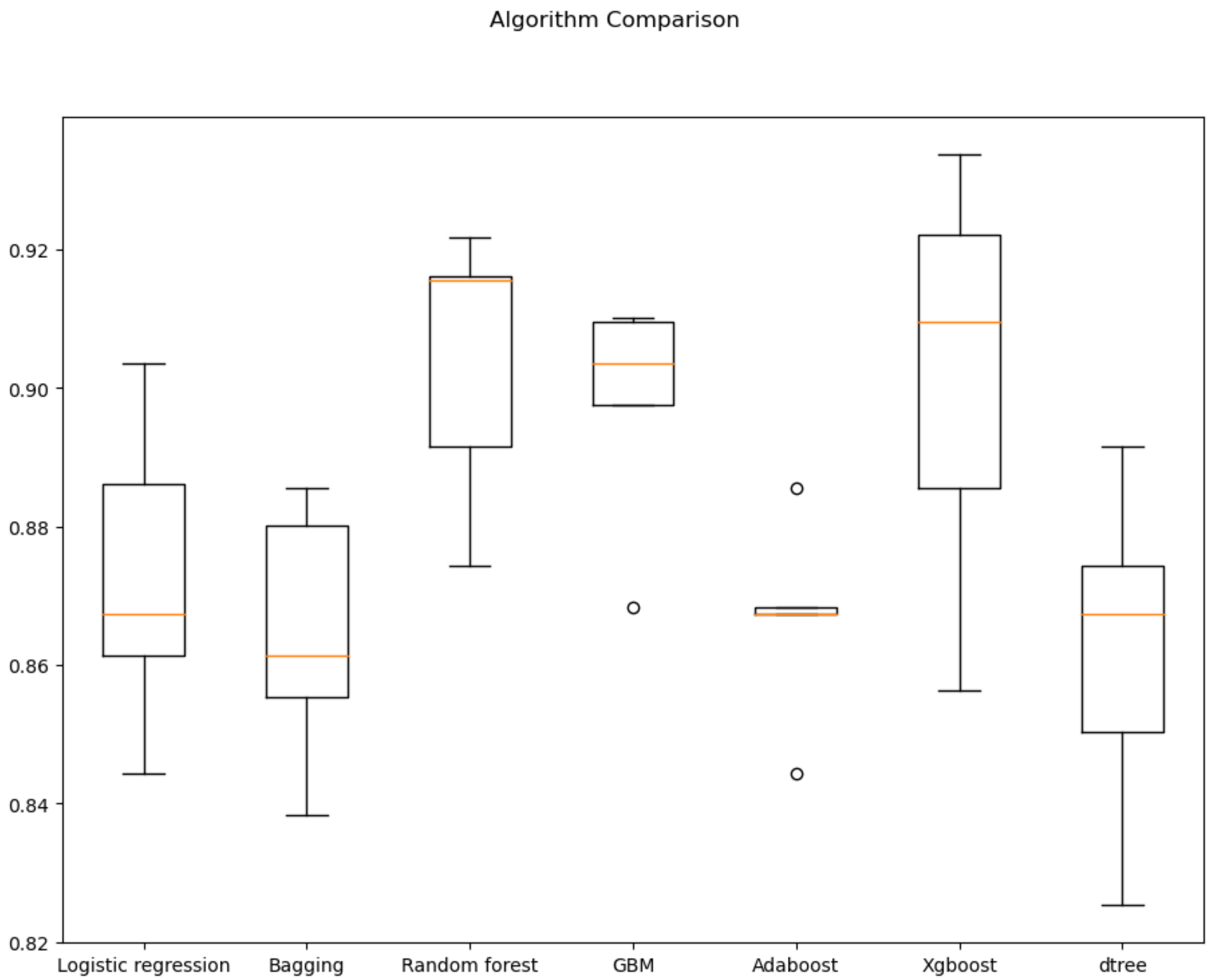
Next I will build my underfitting models

In [36]:
```
models = []

models.append(("Logistic regression", LogisticRegression(random_state=1)))
models.append(("Bagging", BaggingClassifier(random_state=1)))
models.append(("Random forest", RandomForestClassifier(random_state=1)))
models.append(("GBM", GradientBoostingClassifier(random_state=1)))
models.append(("Adaboost", AdaBoostClassifier(random_state=1)))
models.append(("Xgboost", XGBClassifier(random_state=1, eval_metric="logloss")))
models.append(("dtree", DecisionTreeClassifier(random_state=1)))

results = []
names = []

print("\n" "Cross-Validation Performance:" "\n")
for name, model in models:
    scoring = "recall"
    kfold = StratifiedKFold(
        n_splits=5, shuffle=True, random_state=1
    )
    cv_result = cross_val_score(
        estimator=model, X=X_train_un, y=y_train_un, scoring=scoring, cv=kfold
    )
    results.append(cv_result)
    names.append(name)
    print("{}: {}".format(name, cv_result.mean() * 100))

print("\n" "Validation Performance:" "\n")

for name, model in models:
    model.fit(X_train_un, y_train_un)
    scores = recall_score(y_val, model.predict(X_val))
    print("{}: {}".format(name, scores))
```

```
Cross-Validation Performance:

Logistic regression: 87.26138085275232
Bagging: 86.41945025611427
Random forest: 90.38669648654498
GBM: 89.78572974532861
Adaboost: 86.6611355602049
Xgboost: 90.14717552846115
dtree: 86.17776495202367

Validation Performance:

Logistic regression: 0.8525179856115108
Bagging: 0.8705035971223022
Random forest: 0.8920863309352518
GBM: 0.8884892086330936
Adaboost: 0.8489208633093526
Xgboost: 0.89568345323741
dtree: 0.841726618705036
```

In [37]:
```
fig = plt.figure(figsize=(11,8))

fig.suptitle("Algorithm Comparison")
ax = fig.add_subplot(111)

plt.boxplot(results)
```

```
ax.set_xticklabels(names)

plt.show()
```



My best preforming model was random forest

I will now tune my models

In [ ]:
```
%%time

#defining model
model = AdaBoostClassifier(random_state=1)

#Parameter grid to pass in RandomizedSearchCV
param_grid = {'n_estimators':[100,150,200],
             'learning_rate':[0.2,0.05],
             'base_estimator':[DecisionTreeClassifier(max_depth=1, random_state=1), Dec
            }


#Calling RandomizedSearchCV
Randomized_cv = RandomizedSearchCV(estimator=model, param_distributions=param_grid, sc

#Fitting parameters in RandomizedSearchCV
Randomized_cv.fit(X_train_over,y_train_over)
```

```
print("Best parameters are {} with CV score={}:" .format(Randomized_cv.best_params_,Ra
```

In [39]:
```
tune_ada = AdaBoostClassifier(n_estimators= 200, learning_rate= 0.2, base_estimator= D
tune_ada.fit(X_train_over,y_train_over)
```

Out[39]:
```
AdaBoostClassifier
base_estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

In [40]:
```
ada_train_perf = model_performance_classification_sklearn(tune_ada,X_train_over,y_trai
ada_train_perf
```

Out[40]:

	Accuracy	Recall	Precision	F1
0	0.992	0.988	0.995	0.992

In [41]:
```
ada_val_perf = model_performance_classification_sklearn(tune_ada,X_val,y_val)
ada_val_perf
```

Out[41]:

	Accuracy	Recall	Precision	F1
0	0.979	0.849	0.789	0.818

In [42]:
```
model = RandomForestClassifier(random_state=1)

# Parameter grid to pass in RandomizedSearchCV
param_grid =  {'n_estimators':[200,250,300],
            'min_samples_leaf': np.arange(1,4),
            'max_features': [np.arange(0.3,0.6,0.1),'sqrt'],
               'max_samples':np.arange(0.4,0.7,0.1)
           }

# Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_grid,
    n_iter=50,
    n_jobs=-1,
    scoring=scorer,
    cv=5,
    random_state=1,
)

# Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_un, y_train_un)

print(
    "Best parameters are {} with CV score={}:".format(
        randomized_cv.best_params_, randomized_cv.best_score_
    )
)
```

```
Best parameters are {'n_estimators': 300, 'min_samples_leaf': 2, 'max_samples': 0.5,
'max_features': 'sqrt'} with CV score=0.8990116153235697:
```

In [43]:
```
tune_rf = RandomForestClassifier(n_estimators= 300, min_samples_leaf= 2, max_samples=
tune_rf.fit(X_train_un,y_train_un)
```

Out[43]:
```
RandomForestClassifier
RandomForestClassifier(max_samples=0.5, min_samples_leaf=2, n_estimators=30
0)
```

In [44]:
```
rf_train_perf = model_performance_classification_sklearn(tune_rf,X_train_un,y_train_ur
rf_train_perf
```

Out[44]:

	Accuracy	Recall	Precision	F1
0	0.962	0.934	0.990	0.961

In [45]:
```
rf_val_perf = model_performance_classification_sklearn(tune_rf,X_val,y_val)
rf_val_perf
```

Out[45]:

	Accuracy	Recall	Precision	F1
0	0.935	0.881	0.457	0.602

In [46]:
```
model = GradientBoostingClassifier(random_state=1)

# Parameter grid to pass in RandomizedSearchCV
param_grid =  {'n_estimators':[200,250,300],
             'learning_rate': np.arange(.05,.2,1),
             'max_features': np.arange(0.5,0.7),
                'subsample':np.arange(0.5,0.7)
            }

# Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_grid,
    n_iter=50,
    n_jobs=-1,
    scoring=scorer,
    cv=5,
    random_state=1,
)

# Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_over, y_train_over)

print(
    "Best parameters are {} with CV score={}:".format(
        randomized_cv.best_params_, randomized_cv.best_score_
    )
)
```

```
Best parameters are {'subsample': 0.5, 'n_estimators': 300, 'max_features': 0.5, 'lea
rning_rate': 0.05} with CV score=0.9335123572593496:
```

In [47]:
```
tune_gb = GradientBoostingClassifier(n_estimators= 300, subsample= 0.5, learning_rate=
tune_gb.fit(X_train_over, y_train_over)
```

Out[47]:
```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.05, max_features=0.5,
                           n_estimators=300, subsample=0.5)
```

In [48]:
```
gb_train_perf = model_performance_classification_sklearn(tune_gb,X_train_over,y_train_
gb_train_perf
```

Out[48]:

	Accuracy	Recall	Precision	F1
0	0.960	0.937	0.981	0.959

In [49]:
```
gb_val_perf = model_performance_classification_sklearn(tune_gb,X_val,y_val)
gb_val_perf
```

Out[49]:

	Accuracy	Recall	Precision	F1
0	0.971	0.878	0.683	0.769

In [50]:
```
models_train_comp_df = pd.concat(
    [
       gb_val_perf.T,
        rf_val_perf.T,
        ada_val_perf.T,
        ],
    axis=1,
)
models_train_comp_df.columns = [
    "Gradient Boosting",
    "Random Forest",
    "Ada",
]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

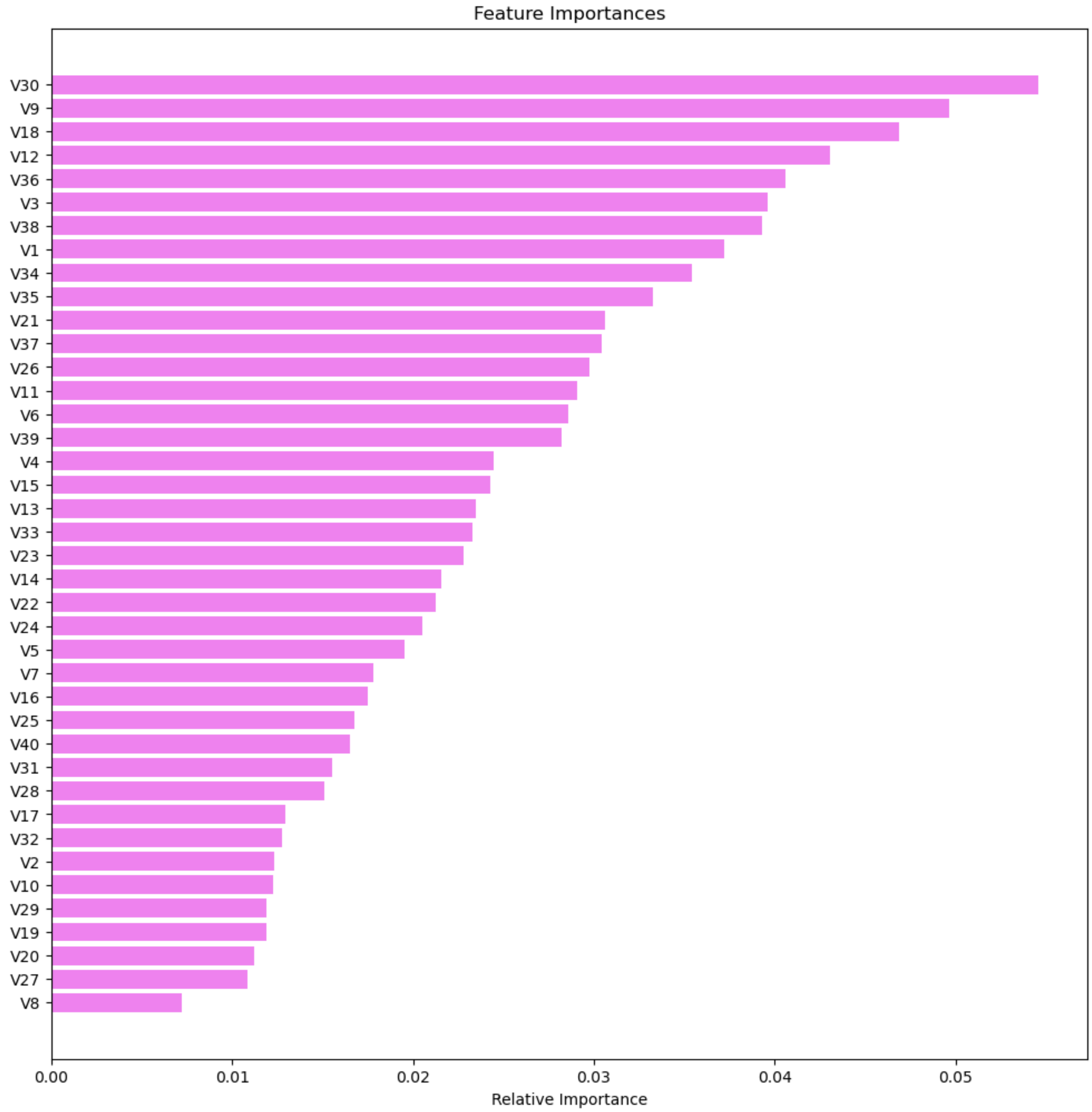
Out[50]:

	Gradient Boosting	Random Forest	Ada
Accuracy	0.971	0.935	0.979
Recall	0.878	0.881	0.849
Precision	0.683	0.457	0.789
F1	0.769	0.602	0.818

My best preforming model was Ada so thats what I'll use to build my final model

In [51]:
```
feature_names = X_train.columns
importances = tune_ada.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12, 12))
```

```
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="violet", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



The 3 most important variables are v30, v9, and v18.

In [54]:
```
# Creating new pipeline with best parameters
Final_pipe = Pipeline(
    steps=[
        ("imputer", SimpleImputer(strategy='median')),
        (
            "Ada",
            AdaBoostClassifier(
               n_estimators= 200, learning_rate= 0.2, base_estimator= DecisionTreeClas
        )
)
    ])
```

```
# Fit the model on training data
Final_pipe.fit(X_train, y_train)
```

Out[54]:

```
Pipeline
  SimpleImputer
  Ada: AdaBoostClassifier
  base_estimator: DecisionTreeClassifier
    DecisionTreeClassifier
```

Next I will prepare my final model

In [60]:
```
X1 = data.drop(["Target"], axis=1)
y1 = data["Target"]
Xt1 = data_test.drop(["Target"], axis=1)
yt1 = data_test["Target"]
```

In [56]:
```
imputer = SimpleImputer(strategy='median')
X1 = imputer.fit_transform(X1)
```

In [57]:
```
SM = SMOTE(sampling_strategy=1, k_neighbors=5, random_state=1)
X_overf, y_overf = sm.fit_resample(X1, y1)
```

In [58]:
```
Final_pipe.fit(X_overf, y_overf)
```

Out[58]:

```
Pipeline
  SimpleImputer
  Ada: AdaBoostClassifier
  base_estimator: DecisionTreeClassifier
    DecisionTreeClassifier
```

In [59]:
```
Final_pipe_perf = model_performance_classification_sklearn(Final_pipe,X_test,y_test)
Final_pipe_perf
```

Out[59]:

	Accuracy	Recall	Precision	F1
0	0.978	0.851	0.774	0.811

Ada using oversampling will yeild the best model

This model will better help predict wind turbine failures

v30, v9, and v18 are the most important variables and play a key role in this model