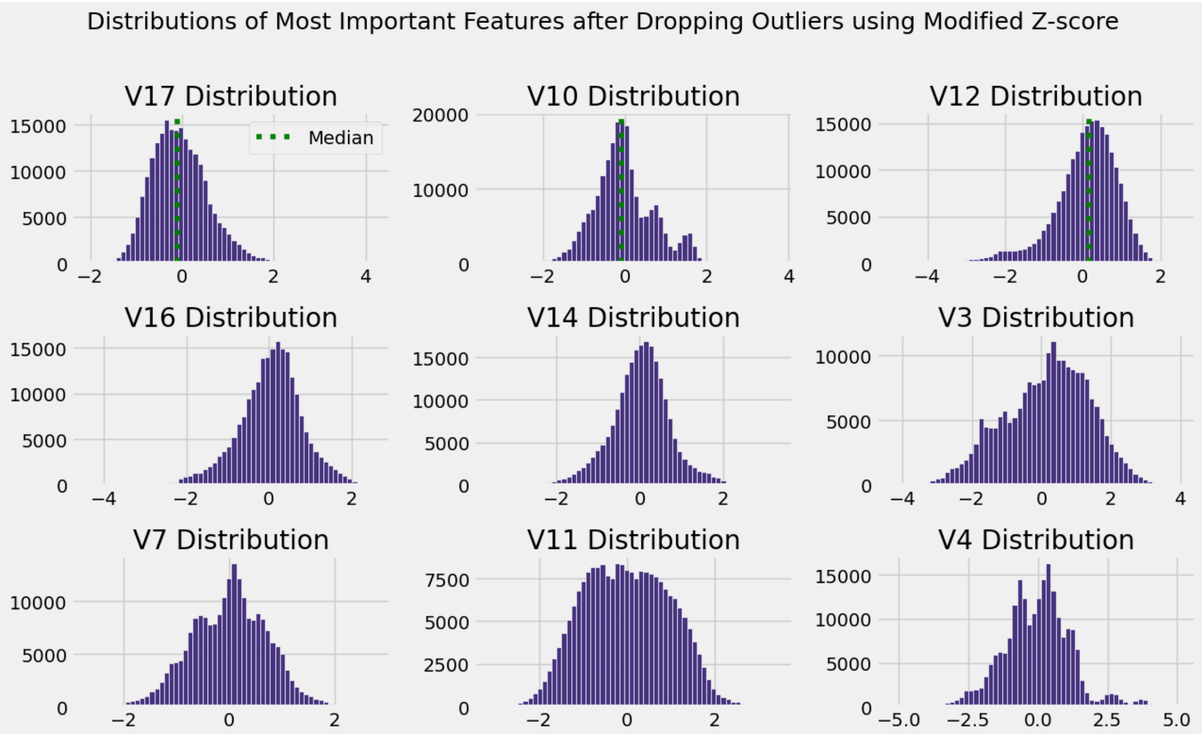
```
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(13, 8))
fig.suptitle('Distributions of Most Important Features after Dropping
Outliers using Modified Z-score\n', size=18)

# Plot histograms for each feature
axes[0, 0].hist(df_out4['V17'], bins=60, linewidth=0.5,
edgecolor="white")
axes[0, 0].axvline(np.median(df_out4['V17']), ls=':', c='g',
label="Median")
axes[0, 0].set_title("V17 Distribution")

axes[0, 1].hist(df_out4['V10'], bins=60, linewidth=0.5,
edgecolor="white")
axes[0, 1].axvline(np.median(df_out4['V10']), ls=':', c='g',
label="Median")
axes[0, 1].set_title("V10 Distribution")

axes[0, 2].hist(df_out4['V12'], bins=60, linewidth=0.5,
edgecolor="white")
axes[0, 2].axvline(np.median(df_out4['V12']), ls=':', c='g',
label="Median")
axes[0, 2].set_title("V12 Distribution")

axes[1, 0].hist(df_out4['V16'], bins=60, linewidth=0.5,
edgecolor="white")
axes[1, 0].set_title("V16 Distribution")

axes[1, 1].hist(df_out4['V14'], bins=60, linewidth=0.5,
edgecolor="white")
axes[1, 1].set_title("V14 Distribution")

axes[1, 2].hist(df_out4['V3'], bins=60, linewidth=0.5,
edgecolor="white")
axes[1, 2].set_title("V3 Distribution")

axes[2, 0].hist(df_out4['V7'], bins=60, linewidth=0.5,
edgecolor="white")
axes[2, 0].set_title("V7 Distribution")

axes[2, 1].hist(df_out4['V11'], bins=60, linewidth=0.5,
edgecolor="white")
axes[2, 1].set_title("V11 Distribution")
```

```
axes[2, 2].hist(df_out4['V4'], bins=60, linewidth=0.5,
edgecolor="white")
axes[2, 2].set_title("V4 Distribution")

# Add legend for the median lines
axes[0, 0].legend()

# Adjust the layout to avoid overlap
plt.tight_layout()

# Show the plot
plt.show()
```



5. Isolation Forest: An Unsupervised Anomaly Detection Algorithm

In [31]:

```
from sklearn.ensemble import IsolationForest

df5 = df.copy()
df5 = df5.drop(['Class'], axis=1)
```

Key Parameters of Isolation Forest

In [32]:

```
# Import the Isolation Forest model from the scikit-learn library
from sklearn.ensemble import IsolationForest

# Create an Isolation Forest model with specified hyperparameters
# - n_estimators: Number of base estimators in the ensemble (150 in this
case)
# - max_samples: Number of samples to draw from the DataFrame ('auto'
means all samples)
# - contamination: The expected proportion of outliers in the dataset
(0.1 or 10% in this case)
# - max_features: Maximum number of features to consider for each split
(1.0 means all features)
model = IsolationForest(n_estimators=150, max_samples='auto',
contamination=float(0.1), max_features=1.0)

# Fit the Isolation Forest model to the DataFrame 'df5'
model.fit(df5)
```

Out[32]:

IsolationForest

```
IsolationForest(contamination=0.1, n_estimators=150)
```

Adding Scores and Anomaly Column

In [33]:

```
# Calculate anomaly scores for each data point in 'df5' using the fitted
Isolation Forest model
scores = model.decision_function(df5)

# Predict whether each data point is an anomaly (outlier) or not
anomaly = model.predict(df5)
```

```python
# Add the calculated anomaly scores as a new column 'scores' in the 'df5' DataFrame
df5['scores'] = scores

# Add the binary anomaly predictions as a new column 'anomaly' in the 'df5' DataFrame
df5['anomaly'] = anomaly

# Display the first 10 rows of the updated 'df5' DataFrame, including the 'scores' and 'anomaly' columns
df5.head(10)
```

Out[33]:

	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	Amount	scores	anomaly
0	-1.360	-0.073	2.536	1.378	-0.338	0.462	0.240	0.099	0.364	0.091	-0.552	-0.618	-0.991	-0.311	1.468	-0.470	0.208	0.026	0.404	0.251	-0.018	0.278	-0.110	0.067	0.129	-0.189	0.134	-0.021	149.620	0.078	1
1	1.192	0.266	0.166	0.448	0.060	-0.082	-0.079	0.085	-0.255	-0.167	1.613	1.065	0.489	-0.144	0.636	0.464	-0.115	-0.183	-0.146	-0.069	-0.226	-0.639	0.101	-0.340	0.167	0.126	-0.009	0.015	2.690	0.088	1

2	-1.358	-1.340	1.773	0.380	-0.503	1.800	0.791	0.248	-1.515	0.208	0.625	0.066	0.717	-0.166	2.346	-2.890	1.110	-0.121	-2.262	0.525	0.248	0.772	0.909	-0.689	-0.328	-0.139	-0.055	-0.060	378.660	-0.011	-1
3	-0.966	-0.185	1.793	-0.863	-0.010	1.247	0.238	0.377	-1.387	-0.055	-0.226	0.178	0.508	-0.288	-0.631	1.060	-0.684	1.966	-1.233	-0.208	-0.108	0.005	-0.190	-1.176	0.647	-0.222	0.063	0.061	123.500	0.053	1
4	-1.158	0.878	1.549	0.403	-0.407	0.096	0.593	-0.271	0.818	0.753	-0.823	0.538	1.346	-1.120	0.175	-0.451	-0.237	-0.038	0.803	0.409	-0.009	0.798	-0.137	0.141	-0.206	0.502	0.219	0.215	69.990	0.062	1
5	-0.426	0.961	1.141	-0.168	0.421	-0.030	0.476	0.260	-0.569	-0.371	1.341	0.360	-0.358	-0.137	0.518	0.402	-0.058	0.069	-0.033	0.085	-0.208	-0.560	-0.026	-0.371	-0.233	0.106	0.254	0.081	3.670	0.092	1
6	1.230	0.141	0.045	1.203	0.192	0.273	-0.005	0.081	0.465	-0.099	-1.417	-0.154	-0.751	0.167	0.050	-0.444	0.003	-0.612	-0.046	-0.220	-0.168	-0.271	-0.154	-0.780	0.750	-0.257	0.035	0.005	4.990	0.084	1

7	-0.644	1.418	1.074	-0.492	0.949	0.428	1.121	-3.808	0.615	1.249	-0.619	0.291	1.758	-1.324	0.686	-0.076	-1.222	-0.358	0.325	-0.157	1.943	-1.015	0.058	-0.650	-0.415	-0.052	-1.207	-1.085	40.800	-0.011	-1
8	-0.894	0.286	-0.113	-0.272	2.670	3.722	0.370	0.851	-0.392	-0.410	-0.705	-0.110	-0.286	0.074	-0.329	-0.210	-0.500	0.119	0.570	0.053	-0.073	-0.268	-0.204	1.012	0.373	-0.384	0.012	0.142	93.200	0.058	1
9	-0.338	1.120	1.044	-0.222	0.499	-0.247	0.652	0.070	-0.737	-0.367	1.018	0.836	1.007	-0.444	0.150	0.739	-0.541	0.477	0.452	0.204	-0.247	-0.634	-0.121	-0.385	-0.070	0.094	0.246	0.083	3.680	0.083	1

In [34]:

```
# Create a DataFrame 'anomaly' by selecting rows where the 'anomaly'
column is equal to -1 (indicating outliers)
anomaly = df5.loc[df5['anomaly'] == -1]

# Extract the indices of the outlier data points as a list
anomaly_index = list(anomaly.index)

# Print the total number of detected outliers and display it
print('Total number of outliers is:', len(anomaly))
```

```
Total number of outliers is: 28373
```

In [35]:

```python
# Select rows from DataFrame 'df5' where the 'anomaly' column is equal to
-1 (indicating outliers)
outliers_df = df5[df5['anomaly'] == -1]

# Display the first 10 rows of the DataFrame containing detected outliers
outliers_df.head(10)
```

Out[35]:

	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	Amount	scores	anomaly
2	-1.358	-1.340	1.773	0.380	-0.503	1.800	0.791	0.248	-1.515	0.208	0.625	0.066	0.717	-0.166	2.346	-2.890	1.110	-0.121	-2.262	0.525	0.248	0.772	0.909	-0.689	-0.328	-0.139	-0.055	-0.060	378.660	-0.011	-1
7	-0.644	1.418	1.074	-0.492	0.949	0.428	1.121	-3.808	0.615	1.249	-0.619	0.291	1.758	-1.324	0.686	-0.076	-1.222	-0.358	0.325	-0.157	1.943	-1.015	0.058	-0.650	-0.415	-0.052	-1.207	-1.085	40.800	-0.011	-1
18	-5.401	-5.450	1.186	1.736	3.049	-1.763	-1.560	0.161	1.233	0.345	0.917	0.970	-0.267	-0.479	-0.527	0.472	-0.725	0.075	-0.407	-2.197	-0.504	0.984	2.459	0.042	-0.482	-0.621	0.392	0.950	46.800	-0.034	-1

51	-1.005	-0.986	-0.038	3.710	-6.632	5.122	4.372	-2.007	-0.279	-0.231	0.145	-0.063	-0.800	-0.342	-0.931	0.511	0.092	0.824	1.190	-0.002	1.393	-0.382	0.970	0.019	0.571	0.333	0.857	-0.076	1402.950	-0.074	-1
69	-1.923	-0.870	2.320	1.989	0.417	-0.380	0.472	-0.557	-0.649	1.411	-0.518	-0.985	-0.401	-0.831	0.338	0.030	0.371	-1.054	1.890	-0.369	-0.686	-0.779	1.086	0.519	-0.364	3.066	-0.589	-0.396	35.000	-0.012	-1
82	-3.005	2.600	1.484	-2.418	0.306	-0.825	2.065	-1.829	4.009	6.052	2.573	0.067	-0.354	-2.837	0.292	-0.304	-1.942	-0.435	-0.934	2.457	-0.852	-0.181	-0.164	0.516	0.136	0.460	-0.251	-1.106	1.460	-0.081	-1
83	-1.199	-1.474	1.840	-4.516	0.328	-0.174	0.960	-1.026	1.700	-0.079	1.663	0.486	-0.933	-1.119	0.141	-2.812	-0.505	0.891	-1.512	-0.770	-0.453	0.335	-0.365	-0.310	-0.303	-1.244	-1.123	-0.734	89.170	-0.035	-1
85	-4.575	-4.429	3.403	0.904	3.002	-0.491	-2.705	0.666	1.922	-0.614	0.385	1.194	-1.021	-1.247	-2.349	-0.213	-0.100	-0.406	1.638	-0.961	-0.047	0.853	-0.972	-0.115	0.408	-0.305	0.548	-0.456	200.010	-0.063	-1

89	-0.773	-4.146	-0.932	0.027	-1.698	0.460	0.737	-0.314	-0.843	0.017	-0.618	-0.198	1.595	-0.260	1.221	1.325	0.121	-1.249	-0.005	2.443	0.891	0.026	-1.135	-0.655	0.098	-0.209	-0.172	0.208	1142.020	-0.033	-1
104	-1.767	2.353	-0.010	-0.364	1.461	-0.205	0.906	-3.384	0.389	0.791	-0.270	0.047	1.643	-2.678	0.728	0.289	0.384	0.455	0.114	0.073	1.964	-0.883	-0.248	-0.759	0.086	0.203	-0.899	-0.944	0.760	-0.014	-1

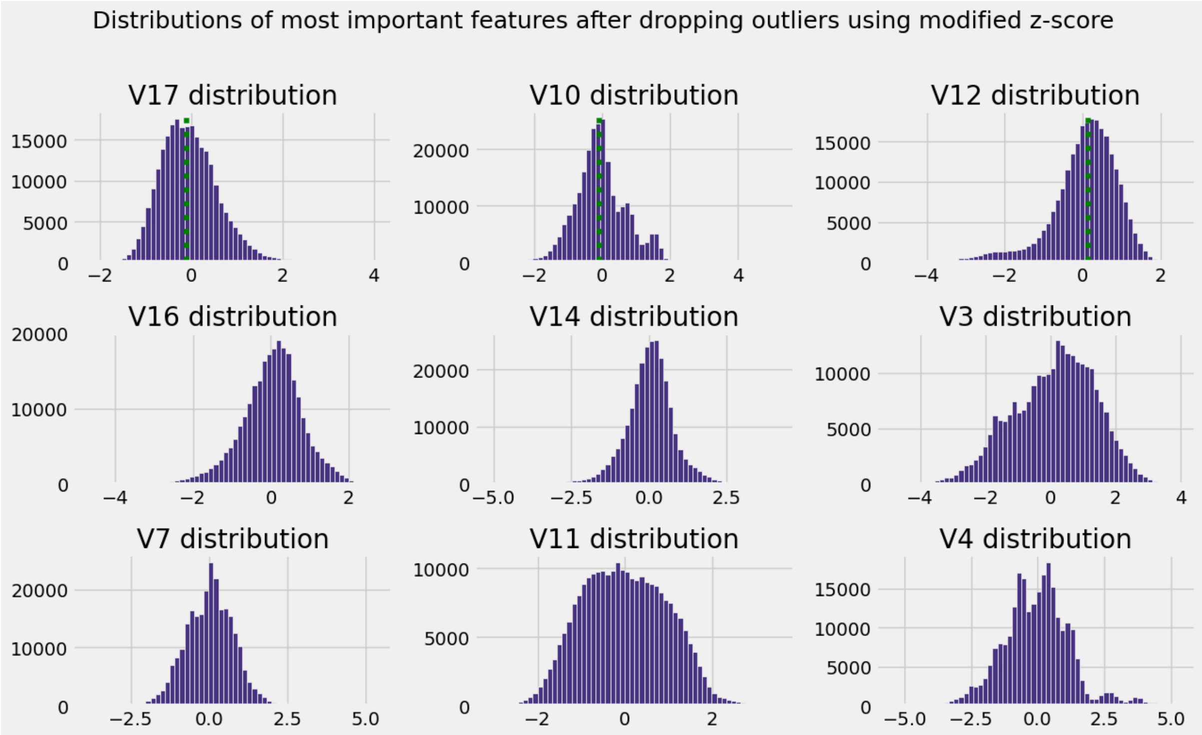
In [36]:

```
# Create a new DataFrame 'df_out5' by dropping rows with outlier indices
# The 'anomaly_index' list contains the indices of detected outliers
df_out5 = df5.drop(anomaly_index, axis=0).reset_index(drop=True)
```

In [37]:

```
# Checking distributions of most important features after dropping
outliers

fig, axes = plt.subplots(nrows=3, ncols=3,figsize=(13,8))
fig.suptitle('Distributions of most important features after dropping
outliers using modified z-score\n', size = 18)

axes[0,0].hist(df_out5['V17'], bins=60, linewidth=0.5,
edgecolor="white")
axes[0,0].axvline(np.median(df_out5['V17']), ls=':', c='g',
label="Median")
axes[0,0].set_title("V17 distribution");

axes[0,1].hist(df_out5['V10'], bins=60, linewidth=0.5,
edgecolor="white")
axes[0,1].axvline(np.median(df_out5['V10']), ls=':', c='g',
label="Median")
```

```
axes[0,1].set_title("V10 distribution");

axes[0,2].hist(df_out5['V12'], bins=60, linewidth=0.5,
edgecolor="white")
axes[0,2].axvline(np.median(df_out5['V12']), ls=':', c='g',
label="Median")
axes[0,2].set_title("V12 distribution");

axes[1,0].hist(df_out5['V16'], bins=60, linewidth=0.5,
edgecolor="white")
axes[1,0].set_title("V16 distribution");

axes[1,1].hist(df_out5['V14'], bins=60, linewidth=0.5,
edgecolor="white")
axes[1,1].set_title("V14 distribution");

axes[1,2].hist(df_out5['V3'], bins=60, linewidth=0.5,
edgecolor="white")
axes[1,2].set_title("V3 distribution");

axes[2,0].hist(df_out5['V7'], bins=60, linewidth=0.5,
edgecolor="white")
axes[2,0].set_title("V7 distribution");

axes[2,1].hist(df_out5['V11'], bins=60, linewidth=0.5,
edgecolor="white")
axes[2,1].set_title("V11 distribution");

axes[2,2].hist(df_out5['V4'], bins=60, linewidth=0.5,
edgecolor="white")
axes[2,2].set_title("V4 distribution");

plt.tight_layout()
```



6. DBSCAN - Density-Based Spatial Clustering of Applications with Noise

In [38]:

```
# Create a copy of the original DataFrame 'df' as 'df6'
df6 = df.copy()

# Drop the 'Class' column from 'df6'
df6 = df6.drop(['Class'], axis=1)
```

In [39]:

```
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Scale the data using StandardScaler
X = StandardScaler().fit_transform(df6.values)

# Create a DBSCAN clustering model with specified hyperparameters
# 'eps' controls the maximum distance between two samples for one to be
considered as in the neighborhood of the other.
```

```
# 'min_samples' sets the minimum number of samples in a neighborhood for
a data point to be considered as a core point.
db = DBSCAN(eps=3.0, min_samples=10).fit(X)

# Extract the cluster labels assigned to each data point
labels = db.labels_
```

In [40]:

```
# Calculate the number of clusters in the dataset
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

# Print the number of clusters
print('The number of clusters in the dataset is:', n_clusters_)
```

```
The number of clusters in the dataset is: 39
```

In [41]:

```
# Convert the cluster labels to a Pandas Series and count occurrences of
each label
label_counts = pd.Series(labels).value_counts()

# Print the counts of each cluster label
print(label_counts)
```

```
0     196273
 1     32446
-1     20325
 9     14841
 2     12254
 5      2046
 7      1405
 12     1168
 10      823
 3       329
 26      287
```

```
13      214
17      212
14      206
11      170
18      166
19       81
4        80
31       48
22       38
21       32
24       32
32       24
15       21
30       20
16       19
27       18
29       17
28       16
23       15
33       14
35       12
37       10
38       10
25       10
6        10
36       10
8         9
20        8
34        7
Name: count, dtype: int64
```

```
nstall datacleaner
```

unfold_moreShow hidden output

In [2]:

```
!pip install fasteda
```

unfold_moreShow hidden output

Importing Libraries

In [3]:

```
#for eda
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
pd.set_option('display.float_format', lambda x: '%.3f' % x)

import numpy as np

from fasteda import fast_eda
from datacleaner import autoclean

import scipy
import scipy.stats as stats

from collections import Counter
```

Data Loading

In [4]:

```
df = pd.read_csv("/kaggle/input/creditcardfraud/creditcard.csv")
```

Data Inspection

In [5]:

```
df.head()
```

Out[5]:

	Time	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	Amount	Class
0	0.000	-1.360	-0.073	2.536	1.378	-0.338	0.462	0.240	0.099	0.364	0.091	-0.552	-0.618	-0.991	-0.311	1.468	-0.470	0.208	0.026	0.404	0.251	-0.018	0.278	-0.110	0.067	0.129	-0.189	0.134	-0.021	149.620	0
1	0.000	1.192	0.266	0.166	0.448	0.060	-0.082	-0.079	0.085	-0.255	-0.167	1.613	1.065	0.489	-0.144	0.636	0.464	-0.115	-0.183	-0.146	-0.069	-0.226	-0.639	0.101	-0.340	0.167	0.126	-0.009	0.015	2.690	0
2	1.000	-1.358	-1.340	1.773	0.380	-0.503	1.800	0.791	0.248	-1.515	0.208	0.625	0.066	0.717	-0.166	2.346	-2.890	1.110	-0.121	-2.262	0.525	0.248	0.772	0.909	-0.689	-0.328	-0.139	-0.055	-0.060	378.660	0
3	1.000	-0.966	-0.185	1.793	-0.863	-0.010	1.247	0.238	0.377	-1.387	-0.055	-0.226	0.178	0.508	-0.288	-0.631	-1.060	-0.684	1.966	-1.233	-0.208	-0.108	0.005	-0.190	-1.176	0.647	-0.222	0.063	0.061	123.500	0

4	2.000	-1.158	0.878	1.549	0.403	-0.407	0.096	0.593	-0.271	0.818	0.753	-0.823	0.538	1.346	-1.120	0.175	-0.451	-0.237	-0.038	0.803	0.409	-0.009	0.798	-0.137	0.141	-0.206	0.502	0.219	0.215	69.990	0

In [6]:

```
df.tail()
```

Out[6]:

	Time	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	Amount	Class
284802	172786.000	-11.881	10.072	-9.835	-2.067	-5.364	-2.607	-4.918	7.305	1.914	4.356	-1.593	2.712	-0.689	4.627	-0.924	1.108	1.992	0.511	-0.683	1.476	0.213	0.112	1.014	-0.509	1.437	0.250	0.944	0.824	0.770	0
284803	172787.000	-0.733	-0.055	2.035	-0.739	0.868	1.058	0.024	0.295	0.585	-0.976	-0.150	0.916	1.215	-0.675	1.165	-0.712	-0.026	-1.221	-1.546	0.060	0.214	0.924	0.012	-1.016	-0.607	-0.395	0.068	-0.054	24.790	0

284804	172788.000	1.920	-0.301	-3.250	-0.558	2.631	3.031	-0.297	0.708	0.432	-0.485	0.412	0.063	-0.184	-0.511	1.329	0.141	0.314	0.396	-0.577	0.001	0.232	0.578	-0.038	0.640	0.266	-0.087	0.004	-0.027	67.880	0
284805	172788.000	-0.240	0.530	0.703	0.690	-0.378	0.624	-0.686	0.679	0.392	-0.399	-1.934	-0.963	-1.042	0.450	1.963	-0.609	0.510	1.114	2.898	0.127	0.265	0.800	-0.163	0.123	-0.569	0.547	0.109	0.105	10.000	0
284806	172792.000	-0.533	-0.190	0.703	-0.506	-0.013	-0.650	1.577	-0.415	0.486	-0.915	-1.040	-0.032	-0.188	-0.084	0.041	-0.303	-0.660	0.167	-0.256	0.383	0.261	0.643	0.377	0.009	-0.474	-0.818	-0.002	0.014	217.000	0

In [7]:

```
df.shape
```

Out[7]:

(284807, 31)

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [9]:

```
df.describe()
```

Out[9]:

	Time	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	Amount	Class
count	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000	284807.000
mean	94813.860	0.000	0.000	-0.000	0.000	0.000	0.000	-0.000	0.000	-0.000	0.000	0.000	-0.000	0.000	0.000	0.000	0.000	-0.000	0.000	0.000	0.000	0.000	-0.000	0.000	0.000	0.000	0.000	-0.000	-0.000	88.350	0.002
std	47488.1	1.959	1.651	1.516	1.416	1.380	1.332	1.237	1.194	1.099	1.089	1.021	0.999	0.995	0.959	0.915	0.876	0.849	0.838	0.814	0.771	0.735	0.726	0.624	0.606	0.521	0.482	0.404	0.330	250.120	0.042

	46																														
min	0.000	-56.408	-72.716	-48.326	-5.683	-113.743	-26.161	-43.557	-73.217	-13.434	-24.588	-4.797	-18.684	-5.792	-19.214	-4.499	-14.130	-25.163	-9.499	-7.214	-54.498	-34.830	-10.933	-44.808	-2.837	-10.295	-2.605	-22.566	-15.430	0.000	0.000
25%	542001.500	-0.920	-0.599	-0.890	-0.849	-0.692	-0.768	-0.554	-0.209	-0.643	-0.535	-0.762	-0.406	-0.649	-0.426	-0.583	-0.468	-0.484	-0.499	-0.456	-0.212	-0.228	-0.542	-0.162	-0.355	-0.317	-0.327	-0.071	-0.053	5.600	0.000
50%	846922.000	0.018	0.065	0.180	-0.020	-0.054	-0.274	0.040	0.022	-0.051	-0.093	-0.033	0.140	-0.014	0.051	0.048	0.066	-0.066	-0.004	0.004	-0.062	-0.029	0.007	-0.011	0.041	0.017	-0.052	0.001	0.011	22.000	0.000
75%	139320.500	1.316	0.804	1.027	0.743	0.612	0.399	0.570	0.327	0.597	0.454	0.740	0.618	0.663	0.493	0.649	0.523	0.400	0.501	0.459	0.133	0.186	0.529	0.148	0.440	0.351	0.241	0.091	0.078	77.165	0.000

max	172792.000	2.455	22.058	9.383	16.875	34.802	73.302	120.589	20.007	15.595	23.745	12.019	7.848	7.127	10.527	8.878	17.315	9.254	5.041	5.592	39.421	27.203	10.503	22.528	4.585	7.520	3.517	31.612	33.848	25691.160	1.000

In [10]:

```
df.skew()
```

Out[10]:

```
Time    -0.036
V1      -3.281
V2      -4.625
V3      -2.240
V4       0.676
V5      -2.426
V6       1.827
V7       2.554
V8      -8.522
V9       0.555
V10      1.187
V11      0.357
V12     -2.278
V13      0.065
V14     -1.995
V15     -0.308
V16     -1.101
V17     -3.845
V18     -0.260
V19      0.109
V20     -2.037
V21      3.593
V22     -0.213
V23     -5.875
V24     -0.552
```

```
V25      -0.416
V26       0.577
V27      -1.170
V28      11.192
Amount   16.978
Class    23.998

dtype: float64
```

In [11]:

```
df.kurtosis()
```

Out[11]:

```
Time      -1.294
V1        32.487
V2        95.773
V3        26.620
V4         2.635
V5       206.905
V6        42.642
V7       405.607
V8       220.587
V9         3.731
V10       31.988
V11        1.634
V12       20.242
V13        0.195
V14       23.879
V15        0.285
V16       10.419
V17       94.800
V18        2.578
V19        1.725
V20      271.016
V21      207.287
V22        2.833
V23      440.089
V24        0.619
V25        4.290
V26        0.919
V27      244.989
```

```
V28       933.398
Amount    845.093
Class     573.888

dtype: float64
```

In [12]:

```
df.isnull().sum()
```

Out[12]:

```
Time      0
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
Amount    0
Class     0
```

```
dtype: int64
```

In [13]:

```python
df.duplicated().sum()
```

Out[13]:

```
1081
```

In [14]:

```python
df = df.drop_duplicates()
df.duplicated().sum()
```

Out[14]:

```
0
```

In [15]:

```python
df.drop("Time", axis = 1, inplace = True)
df.head()
```

Out[15]:

	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	Amount	Class
0	-1.3	-0.0	2.5	1.3	-0.3	0.4	0.2	0.0	0.3	0.0	-0.5	-0.6	-0.9	-0.3	1.4	-0.4	0.2	0.0	0.4	0.2	-0.0	0.2	-0.1	0.0	0.1	-0.1	0.1	-0.0	149.	0

	60	73	36	78	38	62	40	99	64	91	52	18	91	11	68	70	08	26	04	51	18	78	10	67	29	89	34	21	620	
1	1.192	0.266	0.166	0.448	0.060	-0.082	-0.079	0.085	-0.255	-0.167	1.613	1.065	0.489	-0.144	0.636	0.464	-0.115	-0.183	-0.146	-0.069	-0.226	-0.639	0.101	-0.340	0.167	0.126	-0.009	0.015	2.690	0
2	-1.358	-1.340	1.773	0.380	-0.503	1.800	0.791	0.248	-1.515	0.208	0.625	0.066	0.717	-0.166	2.346	-2.890	1.110	-0.121	-2.262	0.525	0.248	0.772	0.909	-0.689	-0.328	-0.139	-0.055	-0.060	378.660	0
3	-0.966	-0.185	1.793	-0.863	-0.010	1.247	0.238	0.377	-1.387	-0.055	-0.226	0.178	0.508	-0.288	-0.631	-1.060	-0.684	1.966	-1.233	-0.208	-0.108	0.005	-0.190	-1.176	0.647	-0.222	0.063	0.061	123.500	0
4	-1.158	0.878	1.549	0.403	-0.407	0.096	0.593	-0.271	0.818	0.753	-0.823	0.538	1.346	-1.120	0.175	-0.451	-0.237	-0.038	0.803	0.409	-0.009	0.798	-0.137	0.141	-0.206	0.502	0.219	0.215	69.990	0

Outlier Analysis

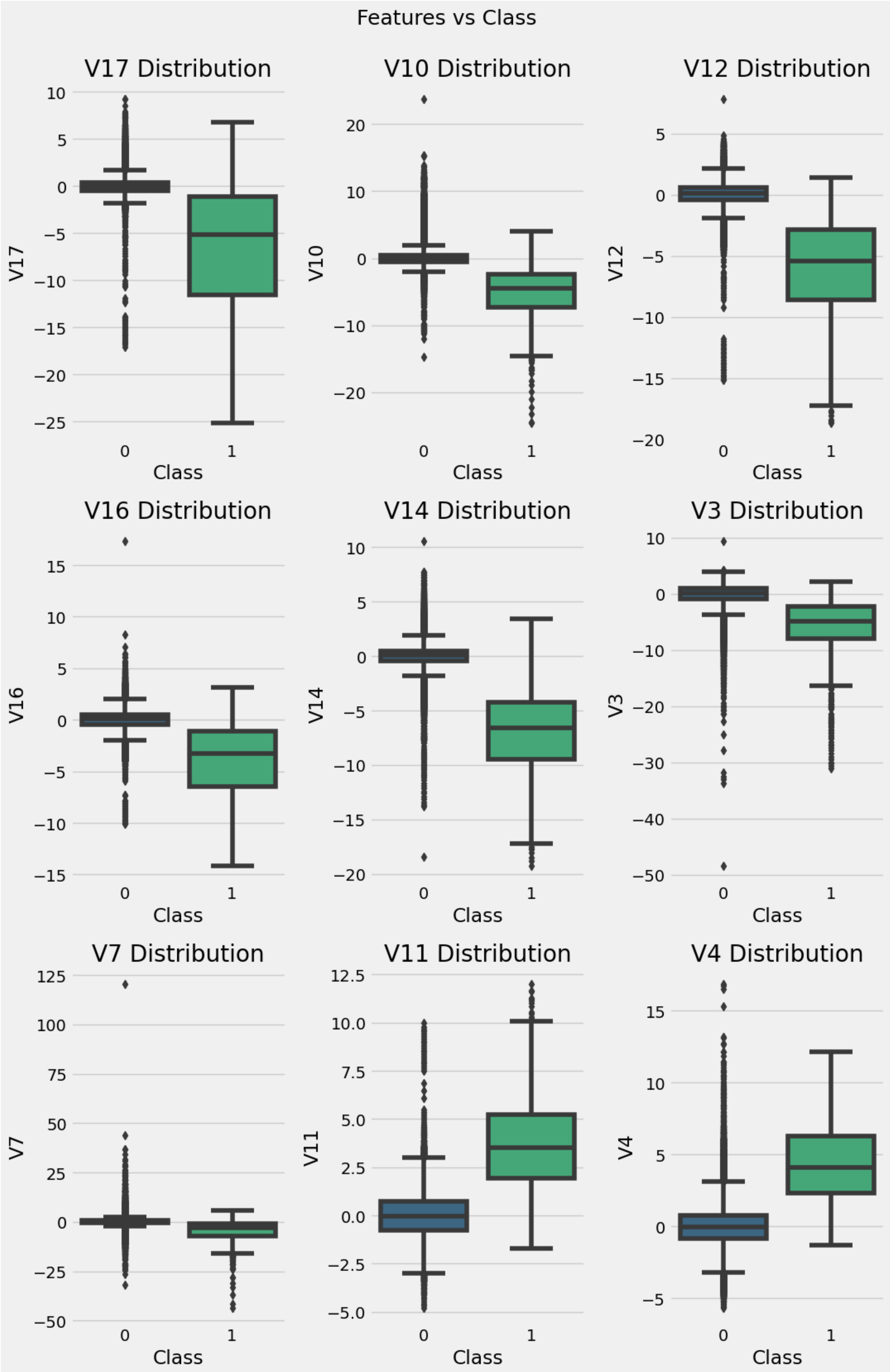
In [16]:

```
# Define the list of features to use
feature_list = ['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',
'Amount']
```

In [17]:

```
# Create subplots for visualizing features vs. Class
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(11, 17))
fig.suptitle('Features vs Class\n', size=18)

# Define the features you want to visualize
features_to_visualize = ['V17', 'V10', 'V12', 'V16', 'V14', 'V3', 'V7',
'V11', 'V4']

# Create boxplots for each feature
for i, feature in enumerate(features_to_visualize):
    row, col = i // 3, i % 3  # Calculate the row and column for the
subplot

    # Create a boxplot for the feature grouped by 'Class' using the
viridis palette
    sns.boxplot(ax=axes[row, col], data=df, x='Class', y=feature,
palette='viridis')
    axes[row, col].set_title(f"{feature} Distribution")

# Adjust the layout to avoid overlap
plt.tight_layout()

# Show the plot
plt.show()
```


Features vs Class
V17 Distribution
V10 Distribution
V12 Distribution
V16 Distribution
V14 Distribution
V3 Distribution
V7 Distribution
V11 Distribution
V4 Distribution
V17
V10
V12
V16
V14
V3
V7
V11
V4
Class
0
1

1. Tukey's IQR Method

In [18]:

```
def IQR_method(df, n, features):
    """
    Identify outliers in a DataFrame using the Tukey IQR method.

    Parameters:
    df (DataFrame): The input DataFrame.
    n (int): The minimum number of outliers in an observation to be
considered.
    features (list): List of feature column names to analyze for
outliers.

    Returns:
    list: A list of indices corresponding to observations with more than
'n' outliers.
    """
    outlier_list = []

    for column in features:
        # 1st quartile (25%)
        Q1 = np.percentile(df[column], 25)
        # 3rd quartile (75%)
        Q3 = np.percentile(df[column], 75)

        # Interquartile range (IQR)
        IQR = Q3 - Q1

        # Outlier step
        outlier_step = 1.5 * IQR

        # Determine a list of indices of outliers
        outlier_list_column = df[(df[column] < Q1 - outlier_step) |
(df[column] > Q3 + outlier_step)].index

        # Append the list of outliers
        outlier_list.extend(outlier_list_column)

    # Count occurrences of each outlier index
    outlier_count = Counter(outlier_list)

    # Select observations containing more than 'n' outliers
```

```
    multiple_outliers = [k for k, v in outlier_count.items() if v > n]

    # Calculate the total number of outliers
    total_outliers = len(multiple_outliers)

    print('Total number of outliers is:', total_outliers)

    return multiple_outliers
```

In [19]:

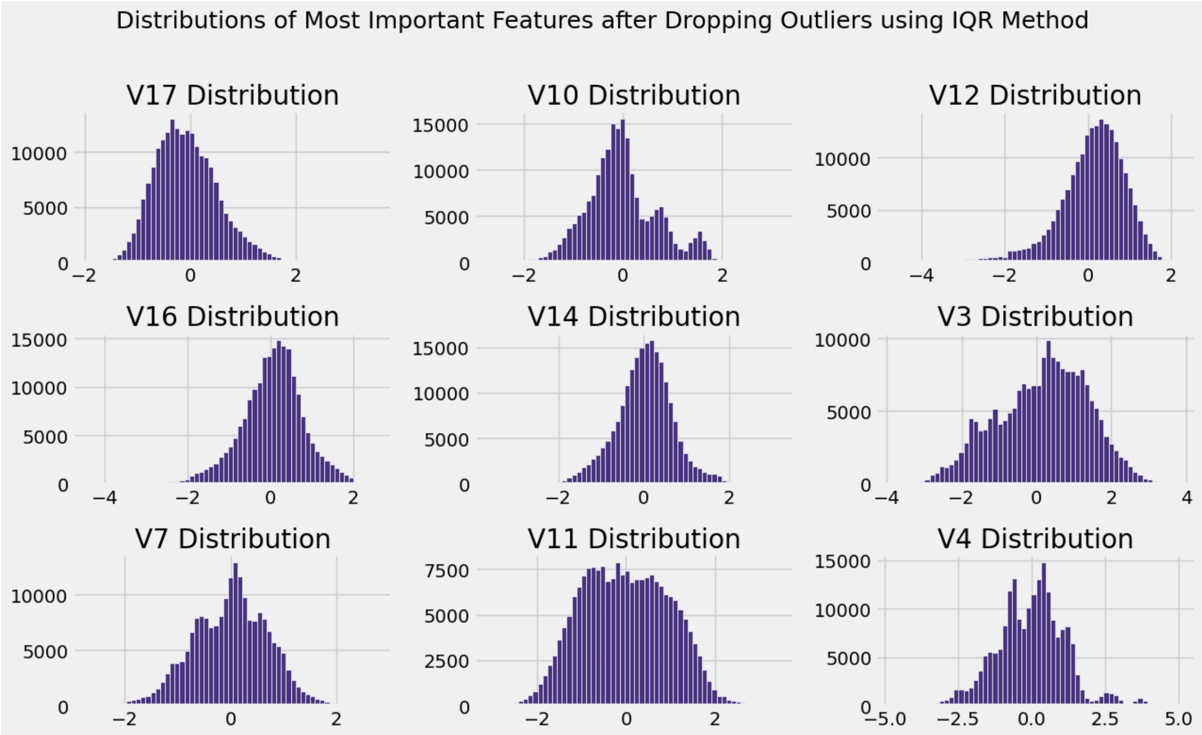
```
# Detecting outliers using the IQR_method function with a threshold of 1
outlier per observation
Outliers_IQR = IQR_method(df, 1, feature_list)

# Dropping outliers from the DataFrame
df_out = df.drop(Outliers_IQR, axis=0).reset_index(drop=True)
```

```
Total number of outliers is: 81014
```

In [20]:

```
# Set the color palette to 'viridis'
sns.set_palette('viridis')

# Create subplots for visualizing the distributions of important features
after outlier removal
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(13, 8))
fig.suptitle('Distributions of Most Important Features after Dropping
Outliers using IQR Method\n', size=18)

# Plot histograms for each feature
axes[0, 0].hist(df_out['V17'], bins=60, linewidth=0.5,
edgecolor="white")
axes[0, 0].set_title("V17 Distribution")

axes[0, 1].hist(df_out['V10'], bins=60, linewidth=0.5,
edgecolor="white")
axes[0, 1].set_title("V10 Distribution")
```

```
axes[0, 2].hist(df_out['V12'], bins=60, linewidth=0.5,
edgecolor="white")
axes[0, 2].set_title("V12 Distribution")

axes[1, 0].hist(df_out['V16'], bins=60, linewidth=0.5,
edgecolor="white")
axes[1, 0].set_title("V16 Distribution")

axes[1, 1].hist(df_out['V14'], bins=60, linewidth=0.5,
edgecolor="white")
axes[1, 1].set_title("V14 Distribution")

axes[1, 2].hist(df_out['V3'], bins=60, linewidth=0.5,
edgecolor="white")
axes[1, 2].set_title("V3 Distribution")

axes[2, 0].hist(df_out['V7'], bins=60, linewidth=0.5,
edgecolor="white")
axes[2, 0].set_title("V7 Distribution")

axes[2, 1].hist(df_out['V11'], bins=60, linewidth=0.5,
edgecolor="white")
axes[2, 1].set_title("V11 Distribution")

axes[2, 2].hist(df_out['V4'], bins=60, linewidth=0.5,
edgecolor="white")
axes[2, 2].set_title("V4 Distribution")

# Adjust the layout to avoid overlap
plt.tight_layout()

# Show the plot
plt.show()
```



In [21]:

```
# Set the color palette to 'viridis'
sns.set_palette('viridis')

# Create subplots for visualizing the distributions of important features
after outlier removal
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(13, 8))
fig.suptitle('Distributions of Most Important Features after Dropping
Outliers using IQR Method\n', size=18)

# Define a hue variable (e.g., 'Class') to add color differentiation
hue_variable = 'Class'

# Plot histograms for each feature with hue
for i, feature in enumerate(features_to_visualize):
    row, col = i // 3, i % 3  # Calculate the row and column for the
subplot

    # Create a histogram for the feature with hue based on 'Class'
    sns.histplot(data=df_out, x=feature, bins=60, linewidth=0.5,
edgecolor="white", hue=hue_variable, ax=axes[row, col])
    axes[row, col].set_title(f"{feature} Distribution")

# Adjust the layout to avoid overlap
```