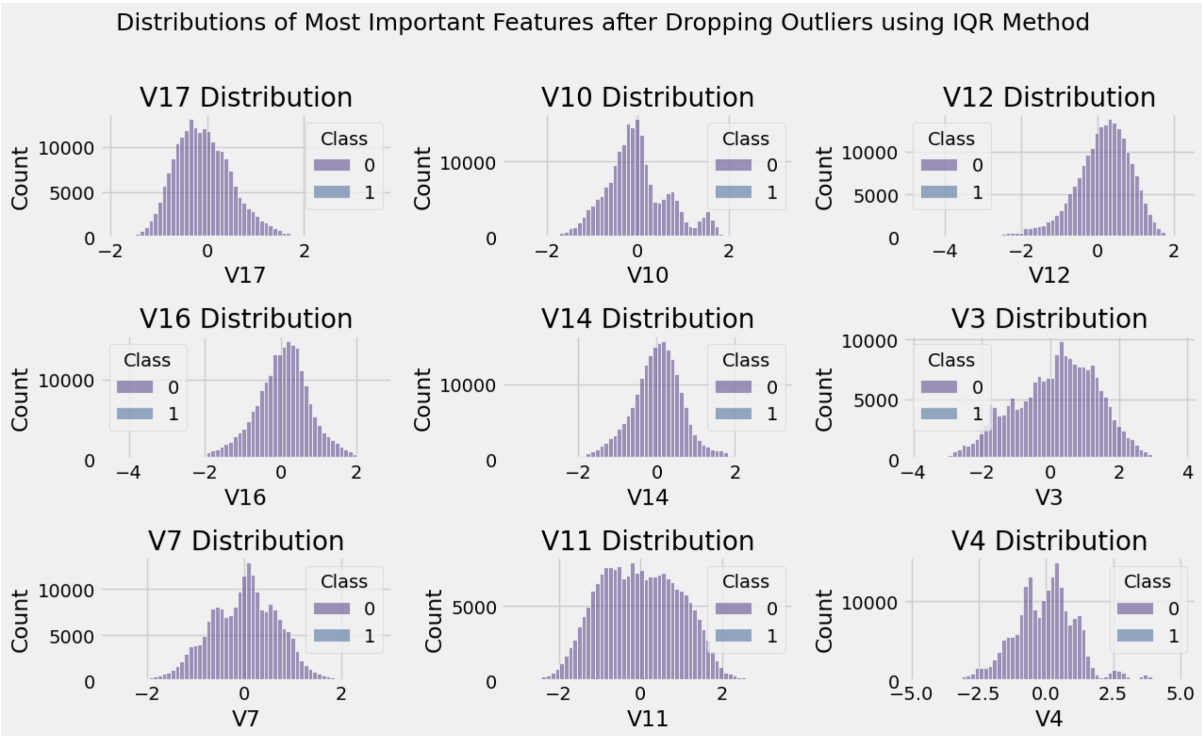
```python
plt.tight_layout()

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



2. Standard Deviation

In [22]:

```python
def StDev_method(df, n, features):
    """
    Identify outliers in a DataFrame using the Standard Deviation 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:
```

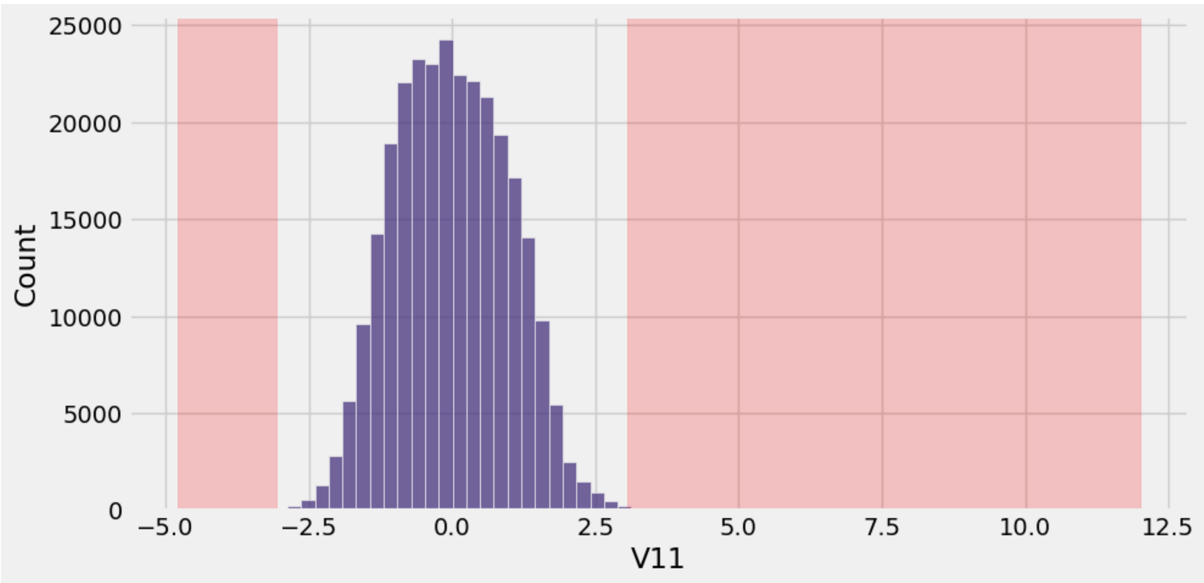
```
    list: A list of indices corresponding to observations with more than
'n' outliers.
    """
    outlier_indices = []

    for column in features:
        # Calculate the mean and standard deviation of the feature column
        data_mean = df[column].mean()
        data_std = df[column].std()

        # Calculate the cutoff value (3 standard deviations from the
mean)
        cut_off = data_std * 3

        # Determine a list of indices of outliers for the feature column
        outlier_list_column = df[(df[column] < data_mean - cut_off) |
(df[column] > data_mean + cut_off)].index

        # Append the found outlier indices for the column to the list of
outlier indices
        outlier_indices.extend(outlier_list_column)

    # Select observations containing more than 'n' outliers
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = [k for k, v in outlier_indices.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 [23]:

```
import seaborn as sns

# Calculate the mean and standard deviation of the 'V11' feature
data_mean, data_std = df['V11'].mean(), df['V11'].std()

# Calculate the cutoff value (3 standard deviations from the mean)
cut_off = data_std * 3
```

```python
# Calculate the lower and upper bounds
lower, upper = data_mean - cut_off, data_mean + cut_off

# Print the lower and upper bound values
print('The lower bound value is:', lower)
print('The upper bound value is:', upper)

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

# Create a histogram to visualize the 'V11' feature
plt.figure(figsize=(10, 5))
sns.histplot(x='V11', data=df, bins=70)

# Highlight the regions outside the bounds in red
plt.axvspan(xmin=lower, xmax=df['V11'].min(), alpha=0.2, color='red')
plt.axvspan(xmin=upper, xmax=df['V11'].max(), alpha=0.2, color='red')

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

```
The lower bound value is: -3.055958700386002
The upper bound value is: 3.0563622156659207
```

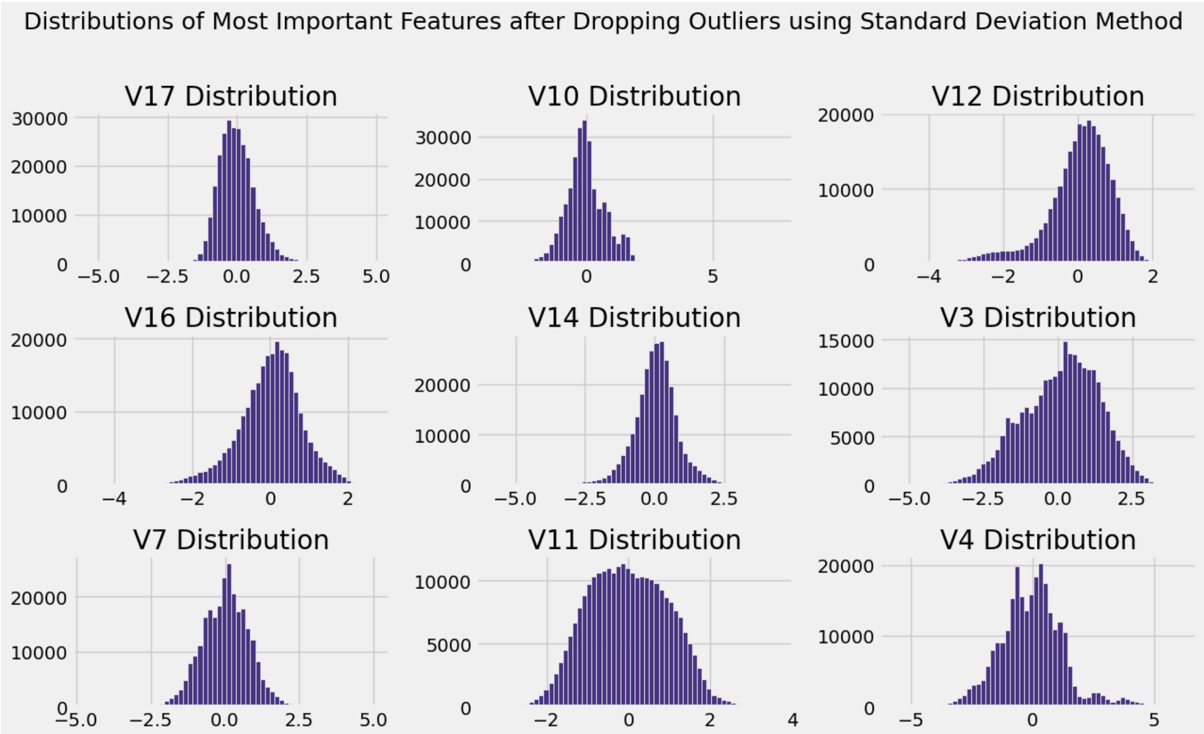


In [24]:

```
# detecting outliers using the StDev_method
Outliers_StDev = StDev_method(df, 1, feature_list)

# dropping outliers
df_out2 = df.drop(Outliers_StDev, axis=0).reset_index(drop=True)

# 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 Standard Deviation Method\n', size=18)

# Plot histograms for each feature
axes[0, 0].hist(df_out2['V17'], bins=60, linewidth=0.5,
edgecolor="white")
axes[0, 0].set_title("V17 Distribution")

axes[0, 1].hist(df_out2['V10'], bins=60, linewidth=0.5,
edgecolor="white")
axes[0, 1].set_title("V10 Distribution")

axes[0, 2].hist(df_out2['V12'], bins=60, linewidth=0.5,
edgecolor="white")
axes[0, 2].set_title("V12 Distribution")

axes[1, 0].hist(df_out2['V16'], bins=60, linewidth=0.5,
edgecolor="white")
axes[1, 0].set_title("V16 Distribution")

axes[1, 1].hist(df_out2['V14'], bins=60, linewidth=0.5,
edgecolor="white")
axes[1, 1].set_title("V14 Distribution")

axes[1, 2].hist(df_out2['V3'], bins=60, linewidth=0.5,
edgecolor="white")
axes[1, 2].set_title("V3 Distribution")

axes[2, 0].hist(df_out2['V7'], bins=60, linewidth=0.5,
edgecolor="white")
```

```
axes[2, 0].set_title("V7 Distribution")

axes[2, 1].hist(df_out2['V11'], bins=60, linewidth=0.5,
edgecolor="white")
axes[2, 1].set_title("V11 Distribution")

axes[2, 2].hist(df_out2['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()
```

```
Total number of outliers is: 14544
```



3. Z-Score

In [25]:

```
def z_score_method(df, n, features):
    """
    Identify outliers in a DataFrame using the Z-score 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 = []
    threshold = 3  # Z-score threshold for identifying outliers

    for column in features:
        # Calculate the mean and standard deviation of the feature column
        data_mean = df[column].mean()
        data_std = df[column].std()

        # Calculate the Z-score for each data point
        z_score = abs((df[column] - data_mean) / data_std)

        # Determine a list of indices of outliers for the feature column
        outlier_list_column = df[z_score > threshold].index

        # Append the found outlier indices for the column to the list of outlier indices
        outlier_list.extend(outlier_list_column)

    # Select observations containing more than 'n' outliers
    outlier_list = Counter(outlier_list)
    multiple_outliers = [k for k, v in outlier_list.items() if v > n]

    # Calculate the total number of outlier records
    df1 = df[df.index.isin(multiple_outliers)]
    total_outliers = df1.shape[0]
    print('Total number of outliers is:', total_outliers)
```

```python
    return multiple_outliers
```

In [26]:

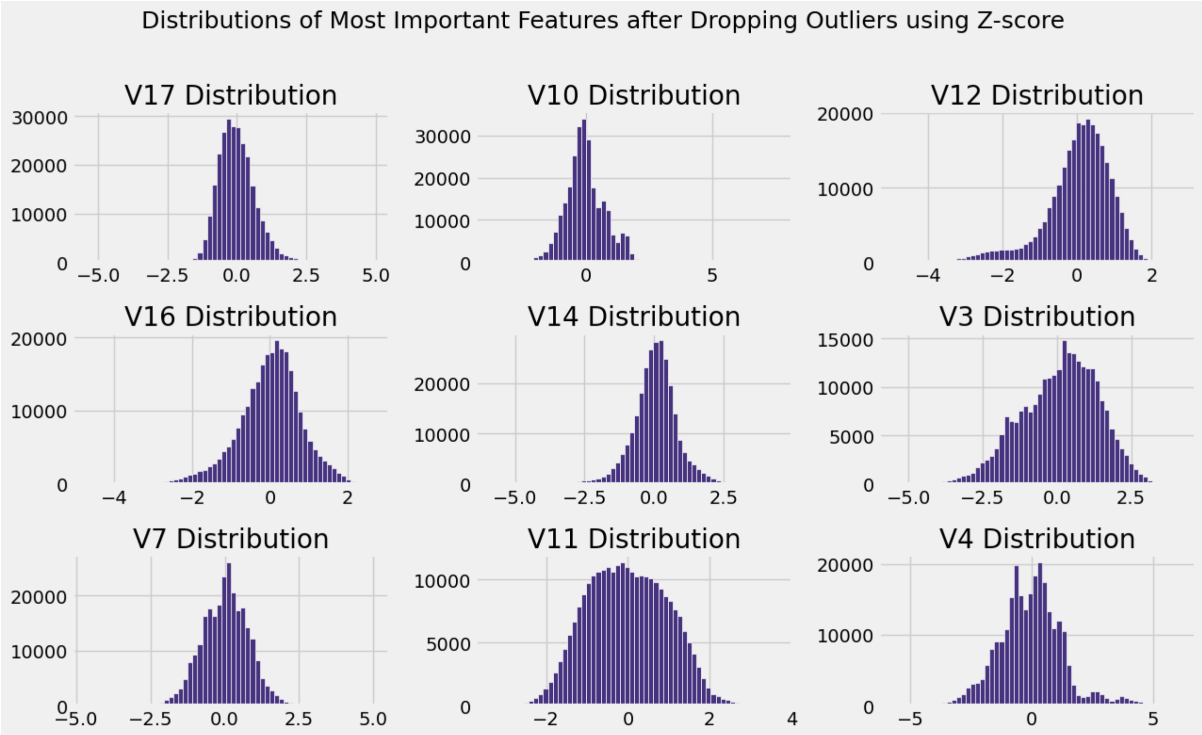
```python
# Detecting outliers using the z_score_method function with a threshold
of 1 outlier per observation
Outliers_z_score = z_score_method(df, 1, feature_list)

# Dropping outliers from the DataFrame
df_out3 = df.drop(Outliers_z_score, axis=0).reset_index(drop=True)
```

```
Total number of outliers is: 14544
```

In [27]:

```python
# 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 Z-score\n', size=18)

# Plot histograms for each feature
axes[0, 0].hist(df_out3['V17'], bins=60, linewidth=0.5,
edgecolor="white")
axes[0, 0].set_title("V17 Distribution")

axes[0, 1].hist(df_out3['V10'], bins=60, linewidth=0.5,
edgecolor="white")
axes[0, 1].set_title("V10 Distribution")

axes[0, 2].hist(df_out3['V12'], bins=60, linewidth=0.5,
edgecolor="white")
axes[0, 2].set_title("V12 Distribution")

axes[1, 0].hist(df_out3['V16'], bins=60, linewidth=0.5,
edgecolor="white")
```

```
axes[1, 0].set_title("V16 Distribution")

axes[1, 1].hist(df_out3['V14'], bins=60, linewidth=0.5,
edgecolor="white")
axes[1, 1].set_title("V14 Distribution")

axes[1, 2].hist(df_out3['V3'], bins=60, linewidth=0.5,
edgecolor="white")
axes[1, 2].set_title("V3 Distribution")

axes[2, 0].hist(df_out3['V7'], bins=60, linewidth=0.5,
edgecolor="white")
axes[2, 0].set_title("V7 Distribution")

axes[2, 1].hist(df_out3['V11'], bins=60, linewidth=0.5,
edgecolor="white")
axes[2, 1].set_title("V11 Distribution")

axes[2, 2].hist(df_out3['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()
```



4. Modified Z-Score

In [28]:

```
from scipy.stats import median_abs_deviation

def z_scoremod_method(df, n, features):
    """
    Identify outliers in a DataFrame using the modified z-score 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 = []
    threshold = 3

    for column in features:
```

```
        # Calculate the mean and modified Z-score for each data point
        data_mean = df[column].mean()
        data_mad = median_abs_deviation(df[column])

        mod_z_score = abs(0.6745 * (df[column] - data_mean) / data_mad)

        # Determine a list of indices of outliers for the feature column
        outlier_list_column = df[mod_z_score > threshold].index

        # Append the found outlier indices for the column to the list of
outlier indices
        outlier_list.extend(outlier_list_column)

    # Select observations containing more than 'n' outliers
    outlier_list = Counter(outlier_list)
    multiple_outliers = [k for k, v in outlier_list.items() if v > n]

    # Calculate the total number of outlier records
    df1 = df[df.index.isin(multiple_outliers)]
    total_outliers = df1.shape[0]
    print('Total number of outliers is:', total_outliers)

    return multiple_outliers
```

In [29]:

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

# Dropping outliers from the DataFrame
df_out4 = df.drop(Outliers_z_score, axis=0).reset_index(drop=True)
```

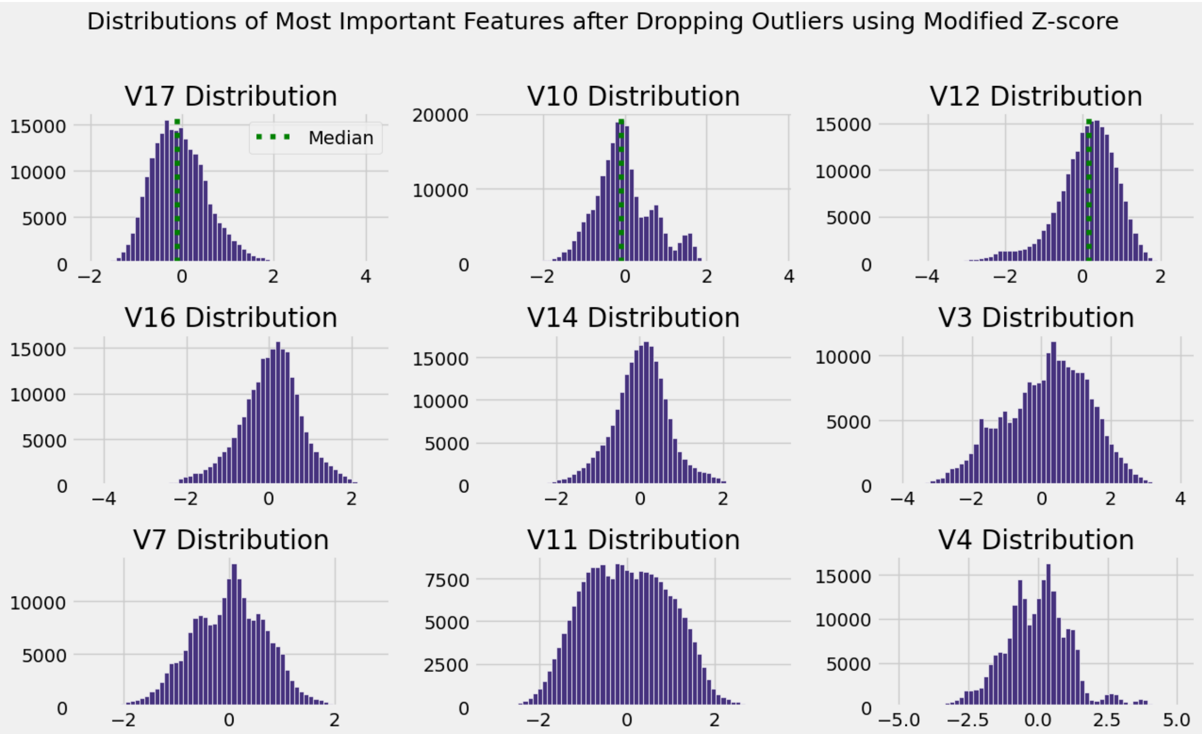
```
Total number of outliers is: 64564
```

In [30]:

```
# 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 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	2000.010	-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

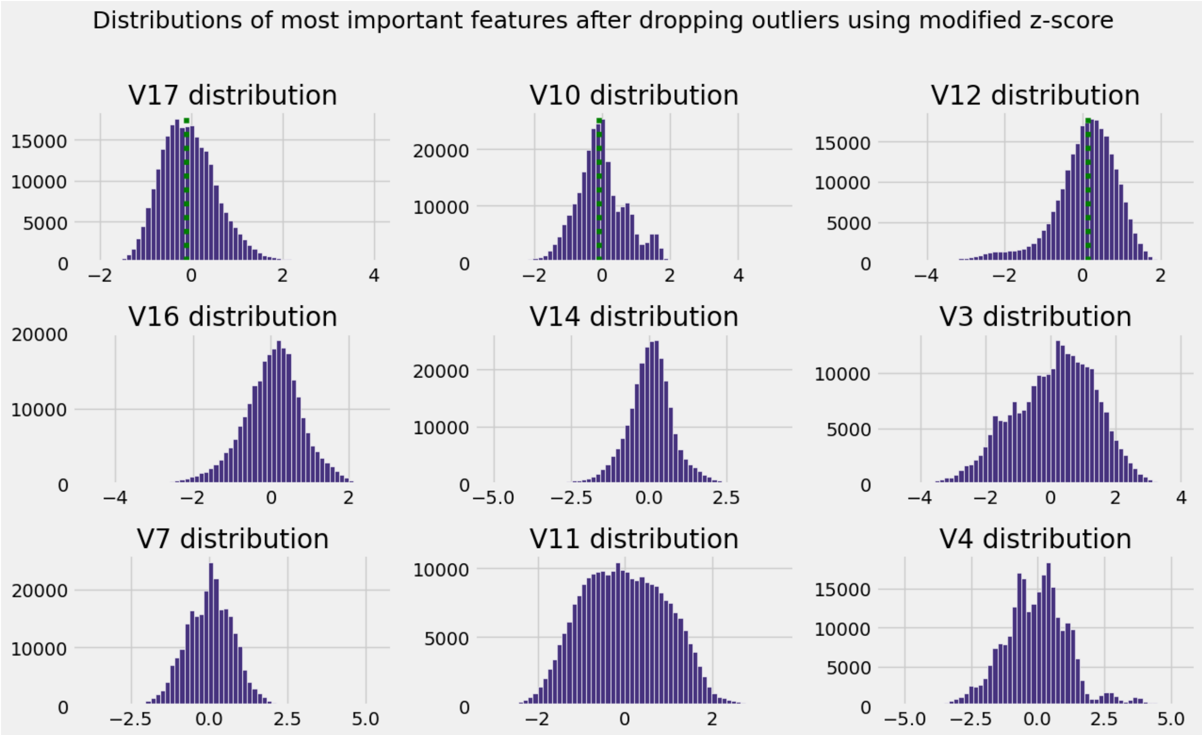
In [36]:

```
# 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.
```