Data Pre-processing Techniques for Fraud Detection

```
In this video, we will walk through a comprehensive process of preparing a
dataset for analysis or machine learning tasks. We will cover the following
steps:

1. Data Cleaning
2. Data Transformation
3. Data Integration
4. Feature Engineering
5. Outlier Detection and Removal

We will illustrate these techniques using a sample dataset containing
transaction data, such as transaction amounts and fraud labels.
```

Import necessary libraries

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import t, zscore
import seaborn as sns
from scipy import stats
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

Create a sample dataset

In [2]:
```
# Create a sample dataset
data = {'TransactionID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
        'Amount': [100, 200, 150, 250, np.nan, 300, 350, 400, np.nan, 450],
        'IsFraud': [0, 1, 0, 0, 1, 0, 1, 0, 1, 0]}

df = pd.DataFrame(data)

df
```

Out[2]:

	TransactionID	Amount	IsFraud
0	1	100.0	0
1	2	200.0	1
2	3	150.0	0
3	4	250.0	0
4	5	NaN	1
5	6	300.0	0
6	7	350.0	1
7	8	400.0	0
8	9	NaN	1
9	10	450.0	0

Data Cleaning Techniques

The first step in the data preparation process is data cleaning. This involves handling missing values, removing duplicates, and correcting data entry errors. In our example, we focus on handling missing values in the 'Amount' column. We fill the missing values using the mean of the non-missing amounts in the column. This approach helps maintain the overall distribution of the data while reducing the impact of missing values on our analysis.

1. Handling missing values

In [3]:
```
df['Amount'].fillna(df['Amount'].mean(), inplace=True)
df
```

Out[3]:

	TransactionID	Amount	IsFraud
0	1	100.0	0
1	2	200.0	1
2	3	150.0	0
3	4	250.0	0
4	5	275.0	1
5	6	300.0	0
6	7	350.0	1
7	8	400.0	0
8	9	275.0	1
9	10	450.0	0

Data Transformation Techniques

Data transformation involves modifying the data to make it more suitable for analysis or machine learning algorithms. In this step, we perform feature scaling on the 'Amount' column using Min-Max normalization. This technique scales the data to a range of 0 to 1, which can be useful for some machine learning algorithms that are sensitive to feature magnitudes. Moreover, it helps ensure that all features have equal weight in the analysis, preventing any single feature from dominating the results.

2. Feature scaling - Min-Max normalization

In [4]:
```
min_amount = df['Amount'].min()
max_amount = df['Amount'].max()
df['NormalizedAmount'] = (df['Amount'] - min_amount) / (max_amount - min_amour
df.head()
```

Out[4]:
```
   TransactionID  Amount  IsFraud  NormalizedAmount
0              1   100.0        0          0.000000
1              2   200.0        1          0.285714
2              3   150.0        0          0.142857
3              4   250.0        0          0.428571
4              5   275.0        1          0.500000
```

Data Integration Techniques

Data integration is the process of combining data from different sources to create a unified dataset. In our example, we merge two datasets based on a common key, the 'TransactionID'. This process allows us to create a comprehensive dataset containing all the necessary information for our analysis. Merging datasets from different sources can help reveal hidden relationships and trends that may not be apparent when analyzing the data separately.

Create sample datasets from different sources

In [5]:
```
data_source1 = {'TransactionID': [1, 2, 3, 4, 5],
                'CustomerID': [101, 102, 103, 104, 105]}

data_source2 = {'TransactionID': [6, 7, 8, 9, 10],
                'CustomerID': [106, 107, 108, 109, 110]}

df_source1 = pd.DataFrame(data_source1)
df_source2 = pd.DataFrame(data_source2)
df_source2.head()
```

Out[5]:

	TransactionID	CustomerID
0	6	106
1	7	107
2	8	108
3	9	109
4	10	110

Merge datasets from different sources

In [6]:
```
df_merged = pd.concat([df_source1, df_source2], ignore_index=True)
df_merged
```

Out[6]:

	TransactionID	CustomerID
0	1	101
1	2	102
2	3	103
3	4	104
4	5	105
5	6	106
6	7	107
7	8	108
8	9	109
9	10	110

Integrate merged dataset with the main dataset

In [7]:
```
df = pd.merge(df, df_merged, on='TransactionID')
df
```

Out[7]:

	TransactionID	Amount	IsFraud	NormalizedAmount	CustomerID
0	1	100.0	0	0.000000	101
1	2	200.0	1	0.285714	102
2	3	150.0	0	0.142857	103
3	4	250.0	0	0.428571	104
4	5	275.0	1	0.500000	105
5	6	300.0	0	0.571429	106
6	7	350.0	1	0.714286	107
7	8	400.0	0	0.857143	108
8	9	275.0	1	0.500000	109
9	10	450.0	0	1.000000	110

Feature Engineering Section

Feature engineering is the process of creating new features from the existing data to improve the performance of machine learning models or enhance the insights gained during analysis. In the provided code, we create a function to generate interaction terms between different columns. Interaction terms can help reveal hidden relationships between features that may not be apparent when considering each feature independently.

To apply feature engineering techniques, you can create custom functions tailored to your specific needs. For example, you can calculate ratios, differences, or combinations of existing features to create new, meaningful variables that capture the underlying structure of the data.

In [8]:
```
def feature_engineering(df):
    # Add your custom feature engineering functions here, e.g.:
    def create_interaction_terms(df, columns):
        for col1 in columns:
            for col2 in columns:
                if col1 != col2:
                    df[f"{col1}_{col2}"] = df[col1] * df[col2]
        return df

    # Apply the feature engineering functions to the dataset
    # E.g., create interaction terms for specific columns
    # df = create_interaction_terms(df, ['col1', 'col2', 'col3'])

    return df

df = feature_engineering(df)
df
```

Out[8]:

	TransactionID	Amount	IsFraud	NormalizedAmount	CustomerID
0	1	100.0	0	0.000000	101
1	2	200.0	1	0.285714	102
2	3	150.0	0	0.142857	103
3	4	250.0	0	0.428571	104
4	5	275.0	1	0.500000	105
5	6	300.0	0	0.571429	106
6	7	350.0	1	0.714286	107
7	8	400.0	0	0.857143	108
8	9	275.0	1	0.500000	109
9	10	450.0	0	1.000000	110

--

Outlier Detection and Removal Section

--

Outliers can have a significant impact on the analysis and may lead to incorrect conclusions. Therefore, it's essential to detect and remove outliers before proceeding with the analysis. In this step, we implement three methods for outlier detection and removal: Z-score, Interquartile Range (IQR), and Grubbs' test (GESD).

Z-score: This method calculates the standard deviations away from the mean for each data point. Data points with an absolute Z-score greater than a specified threshold are considered outliers.

Interquartile Range (IQR): This method identifies outliers as data points falling below the first quartile minus 1.5 times the IQR or above the third quartile plus 1.5 times the IQR.

Grubbs' test (GESD): This method tests for outliers in univariate datasets by comparing the maximum absolute Z-score to a critical value derived from the t-distribution.

In [9]:
```
# GESD test for outlier detection
def grubbs_test(data, alpha=0.05):
    N = len(data)
    Z = zscore(data)
    Z_max = np.abs(Z).max()
    t_alpha = t.ppf(1 - alpha / (2 * N), N - 2)
    G_calculated = Z_max / np.sqrt(1 + (t_alpha ** 2) / (N - 2 + t_alpha ** 2)
    G_critical = ((N - 1) * np.sqrt(np.square(t_alpha))) / (np.sqrt(N) * np.sq

    if G_calculated > G_critical:
        outlier_index = np.argmax(np.abs(Z))
        return outlier_index, True
    else:
        return None, False

def detect_and_remove_outliers(df, method='zscore', threshold=3.0):
    if method == 'zscore':
        z_scores = np.abs(stats.zscore(df))
        outliers = np.where(z_scores > threshold)
        df = df[(z_scores < threshold).all(axis=1)]
    elif method == 'iqr':
        Q1 = df.quantile(0.25)
        Q3 = df.quantile(0.75)
        IQR = Q3 - Q1
        outliers = df[((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(
        df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=
    elif method == 'gesd':
        for column in df.columns:
            has_outliers = True
            while has_outliers:
                index, has_outliers = grubbs_test(df[column])
                if has_outliers:
                    df = df.drop(index)
                    df = df.reset_index(drop=True)
    else:
        raise ValueError("Invalid outlier detection method. Use 'zscore', 'iqr

    return df
```

Remove outliers using one of the methods: 'zscore', 'iqr', or 'gesd'

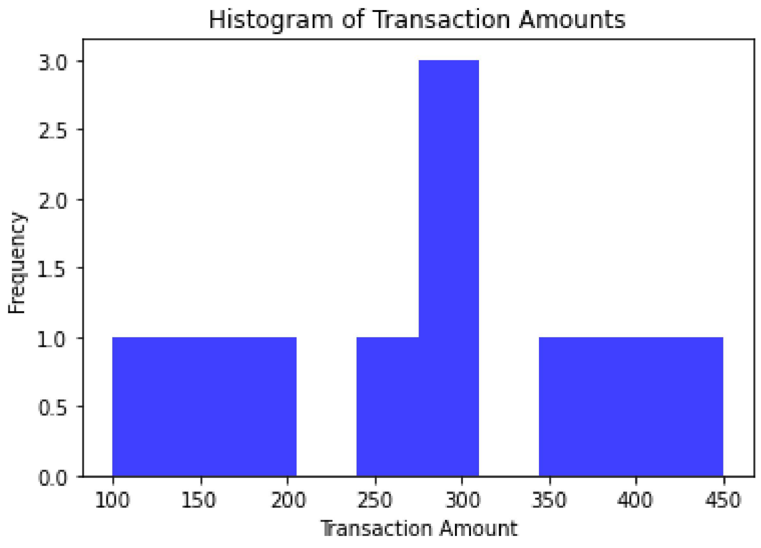
In [10]:
```
df = detect_and_remove_outliers(df, method='gesd')

# Display the cleaned dataset
print(df.head())
```

```
   TransactionID  Amount  IsFraud  NormalizedAmount  CustomerID
0              1   100.0        0          0.000000         101
1              2   200.0        1          0.285714         102
2              3   150.0        0          0.142857         103
3              4   250.0        0          0.428571         104
4              5   275.0        1          0.500000         105
```

Visualizations

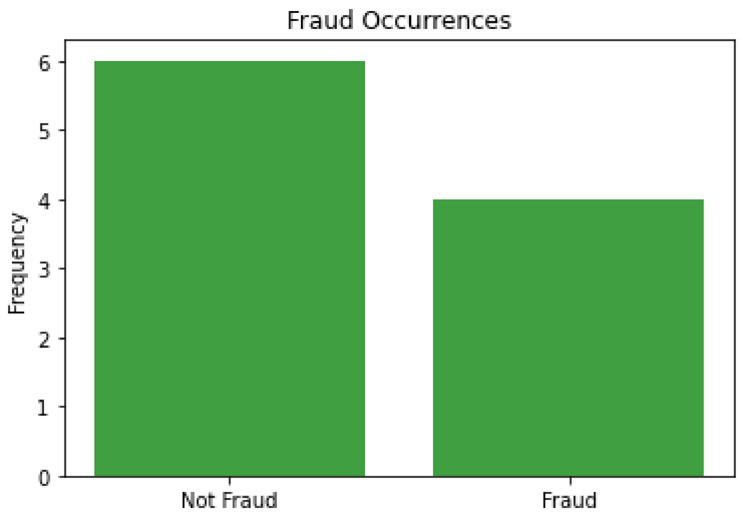
Histogram of transaction amounts

In [11]:
```
plt.hist(df['Amount'], bins=10, alpha=0.75, color='blue')
plt.xlabel('Transaction Amount')
plt.ylabel('Frequency')
plt.title('Histogram of Transaction Amounts')
plt.show()
```



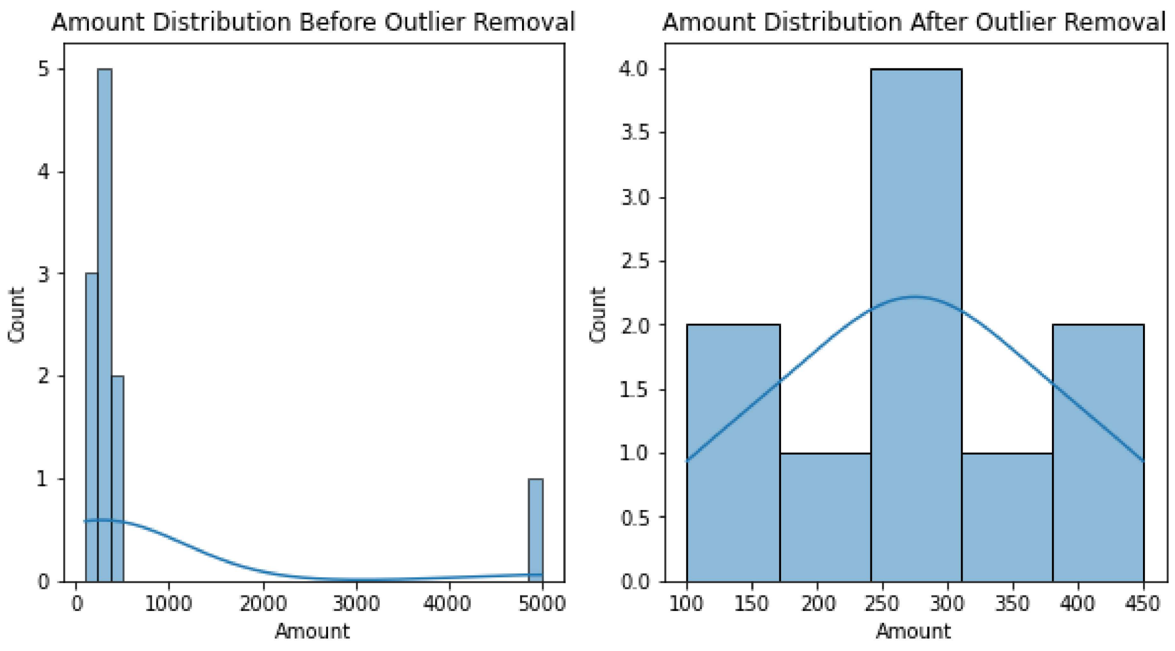
Bar plot of fraud occurrences

In [12]:
```
fraud_counts = df['IsFraud'].value_counts()
plt.bar(fraud_counts.index, fraud_counts.values, alpha=0.75, color='green')
plt.xticks([0, 1], ['Not Fraud', 'Fraud'])
plt.ylabel('Frequency')
plt.title('Fraud Occurrences')
plt.show()
```



Plot the dataset before and after Outlier Detection

In [13]:
```
# Add an outlier to the 'Amount' column
df.loc[10] = [11, 5000, 0, (5000 - min_amount) / (max_amount - min_amount), 11

# Plot the 'Amount' column before outlier removal
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
sns.histplot(df['Amount'], kde=True)
plt.title('Amount Distribution Before Outlier Removal')

# Remove outliers using one of the methods: 'zscore', 'iqr', or 'gesd'
df_cleaned = detect_and_remove_outliers(df, method='gesd')

# Plot the 'Amount' column after outlier removal
plt.subplot(1, 2, 2)
sns.histplot(df_cleaned['Amount'], kde=True)
plt.title('Amount Distribution After Outlier Removal')
plt.show()

# Display the cleaned dataset
print(df_cleaned.head())
```



```
   TransactionID  Amount  IsFraud  NormalizedAmount  CustomerID
0            1.0   100.0      0.0          0.000000       101.0
1            2.0   200.0      1.0          0.285714       102.0
2            3.0   150.0      0.0          0.142857       103.0
3            4.0   250.0      0.0          0.428571       104.0
4            5.0   275.0      1.0          0.500000       105.0
```