Anomaly Detection Techniques in Python

Objective:

The goal of this assignment is to implement and understand various anomaly detection techniques in Python using different datasets and methods.

In [9]:
```
import warnings
import sys
if not sys.warnoptions:
    warnings.simplefilter("ignore")
```

Task 1: Statistical Anomaly Detection

1. **Objective**: Detect anomalies in a numeric dataset using Z-Score and IQR methods.
2. **Instructions**:
   - Create a dataset with numeric values (e.g., [10, 12, 14, 10, 13, 100, 11]).
   - Use Z-Score to identify anomalies (threshold: Z > 3 or Z < -3).
   - Use the IQR method to identify anomalies.

In [11]:
```
from scipy.stats import zscore
import numpy as np

# Dataset
data = np.array([10, 12, 14, 10, 13, 100, 11])

# Z-Score Method
z_scores = zscore(data)
anomalies_zscore = data[np.abs(z_scores) > 3]
print("Anomalies using Z-Score:", anomalies_zscore)

# IQR Method
q1, q3 = np.percentile(data, [25, 75])
```

```
iqr = q3 - q1
lower_bound, upper_bound = q1 - 1.5 * iqr, q3 + 1.5 * iqr
anomalies_iqr = data[(data < lower_bound) | (data > upper_bound)]
print("Anomalies using IQR:", anomalies_iqr)
```

```
Anomalies using Z-Score: []
Anomalies using IQR: [100]
```

Task 2: Machine Learning-Based Anomaly Detection

1. **Objective**: Detect anomalies using K-Nearest Neighbors (KNN) and Isolation Forest.
2. **Instructions**:
   - Use the dataset: `[[10], [12], [14], [10], [13], [100], [11]]`.
   - Use `sklearn.neighbors.LocalOutlierFactor` for KNN-based anomaly detection.
   - Use `sklearn.ensemble.IsolationForest` for Isolation Forest.

In [13]:
```
from sklearn.neighbors import LocalOutlierFactor
from sklearn.ensemble import IsolationForest

# Dataset
data = np.array([[10], [12], [14], [10], [13], [100], [11]])

# KNN Method
lof = LocalOutlierFactor(n_neighbors=2)
anomalies_knn = lof.fit_predict(data)
print("Anomalies using KNN:", data[anomalies_knn == -1])

# Isolation Forest
iso_forest = IsolationForest(contamination=0.1)
iso_forest.fit(data)
anomalies_iso = iso_forest.predict(data)
print("Anomalies using Isolation Forest:", data[anomalies_iso == -1])
```

```
Anomalies using KNN: [[100]]
Anomalies using Isolation Forest: [[100]]
```

Task 3: Deep Learning-Based Anomaly Detection

1. **Objective**: Use an Autoencoder for anomaly detection.

2. **Instructions**:
   - Generate a synthetic dataset with normal and anomalous data points.
   - Build and train an Autoencoder neural network to reconstruct the input data.
   - Identify anomalies using reconstruction error.

In [15]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import numpy as np

# Synthetic Dataset
np.random.seed(42)
data = np.random.normal(size=(100, 10))
data[95:] = np.random.uniform(10, 15, size=(5, 10))  # Add anomalies

# Autoencoder
autoencoder = Sequential([
    Dense(10, activation='relu', input_shape=(10,)),
    Dense(5, activation='relu'),
    Dense(10, activation='relu')
])
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.fit(data, data, epochs=20, batch_size=10, verbose=0)

# Reconstruction Error
reconstructions = autoencoder.predict(data)
reconstruction_errors = np.mean((reconstructions - data) ** 2, axis=1)
threshold = np.percentile(reconstruction_errors, 95)
anomalies_dl = data[reconstruction_errors > threshold]
print("Anomalies using Autoencoder:", anomalies_dl)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 116ms/step
Anomalies using Autoencoder: [[10.83741291 10.5228392  13.18215125 13.53237863 10.15793072 14.68106123
  10.25985642 12.70648168 13.5453026  14.35484562]
 [13.57043466 14.00864042 11.69725096 14.07412557 10.40057423 14.47408328
  12.73796188 14.08648885 12.26159142 13.21788848]
 [12.6320133  13.65794761 10.40814991 10.30176042 11.23551617 10.7977234
  14.35891783 11.09606994 14.87932628 11.68447896]
 [10.91058958 13.94849254 13.29353888 12.49097858 12.77681775 13.59600889
  11.14227371 14.98166958 14.87396581 13.25162843]
 [10.99771225 13.40114121 10.36099204 10.15326251 11.28841444 12.31311478
  14.34136253 13.63584535 13.71353261 12.12746667]]
```

Task 4: Clustering-Based Anomaly Detection

1. **Objective**: Detect anomalies using DBSCAN.
2. **Instructions**:
   - Use the dataset: `[[10], [12], [14], [10], [13], [100], [11]]`.
   - Apply DBSCAN clustering and identify outliers (labeled as -1).

In [17]:
```python
from sklearn.cluster import DBSCAN

# Dataset
data = np.array([[10], [12], [14], [10], [13], [100], [11]])

# DBSCAN
dbscan = DBSCAN(eps=5, min_samples=2)
labels = dbscan.fit_predict(data)
anomalies_dbscan = data[labels == -1]
print("Anomalies using DBSCAN:", anomalies_dbscan)
```

```
Anomalies using DBSCAN: [[100]]
```

Task 5: Time-Series Anomaly Detection

1. **Objective**: Detect anomalies in a time-series dataset using ARIMA.
2. **Instructions**:
   - Generate a time-series dataset with anomalies.
   - Use ARIMA to model the data and detect anomalies as deviations from the predicted values.

In [19]:
```
from statsmodels.tsa.arima.model import ARIMA
import pandas as pd

# Synthetic Time-Series Data
data = pd.Series([10, 12, 14, 13, 15, 120, 12, 14, 13, 10])

# ARIMA Model
model = ARIMA(data, order=(1, 1, 1))
model_fit = model.fit()
forecast = model_fit.predict(start=0, end=len(data)-1)
anomalies_ts = data[(data - forecast).abs() > 10]  # Threshold
print("Anomalies in Time-Series:", anomalies_ts)
```

```
Anomalies in Time-Series: 5    120
6     12
7     14
8     13
9     10
dtype: int64
```