# **Outlier Analysis Techniques Using Python**

Outliers, defined as data points that deviate significantly from the majority of observations, can substantially impact data analysis and the performance of machine learning models 1. Identifying and handling these anomalies is a critical step in data preprocessing to ensure accurate and reliable results 7. Python offers a rich ecosystem of libraries, including NumPy, pandas, scikit-learn, and PyOD, that provide various techniques for outlier analysis 1. This report explores common outlier detection techniques implemented in Python, explains their underlying concepts, provides example scripts, discusses visualization methods, and examines strategies for handling outliers.

## **Statistical Outlier Detection Methods**

Statistical methods are a fundamental approach to identifying outliers by leveraging the properties of data distributions 9. Three commonly used statistical techniques are the Z-score method, the Interquartile Range (IQR) method, and the Median Absolute Deviation (MAD) method 2.

### **Z-score Method**

The Z-score method identifies outliers based on how many standard deviations a data point is from the mean of the dataset 2. The Z-score is calculated using the formula: Z = (X - μ) / σ, where X is the data point, μ is the mean, and σ is the standard deviation 2. Typically, data points with a Z-score greater than 3 or less than -3 are considered outliers, as these values lie far from the mean in a normal distribution 10.

Python

import numpy as np  
import pandas as pd  
from scipy import stats  
  
# Sample data  
data = {'values': }  
df = pd.DataFrame(data)  
  
# Calculate Z-scores  
df['z\_score'] = stats.zscore(df['values'])  
  
# Identify outliers (using a threshold of 3)  
threshold = 3  
outliers\_zscore = df[np.abs(df['z\_score']) > threshold]  
print("Outliers (Z-score method):\n", outliers\_zscore)

This code snippet demonstrates the practical application of the Z-score method in Python, utilizing the scipy.stats library for efficient calculation 2.

The Z-score method is straightforward to understand and implement, making it a popular choice for outlier detection, especially when the data is approximately normally distributed 2. It also provides a numerical measure indicating the degree to which a data point deviates from the mean 11. However, this method assumes a normal distribution and can be significantly affected by the presence of extreme outliers, which can skew the mean and standard deviation, potentially masking other outliers 2. The selection of the Z-score threshold (e.g., 3) can also be somewhat arbitrary and might need adjustment based on the specific dataset and context 12. Furthermore, the Z-score method might not be as effective in identifying multiple outliers simultaneously 12. The method's reliance on the mean and standard deviation makes it vulnerable because these statistics are not robust to extreme values. An outlier can inflate the standard deviation, making it harder for other points to exceed the threshold 2.

### **Interquartile Range (IQR) Method**

The Interquartile Range (IQR) method is a robust statistical technique used to identify outliers by measuring the spread of the middle 50% of the data 2. The IQR is calculated as the difference between the third quartile (Q3, 75th percentile) and the first quartile (Q1, 25th percentile) 2. Outliers are typically defined as data points that fall below Q1 - 1.5 \* IQR or above Q3 + 1.5 \* IQR 2. The 1.5 multiplier is a common convention that approximates around 2.7 standard deviations for normally distributed data 13. A multiplier of 3 can be used to identify extreme outliers 19.

Python

import pandas as pd  
import numpy as np  
  
# Sample data  
data =   
df = pd.DataFrame({'values': data})  
  
# Calculate Q1, Q3, and IQR  
Q1 = df['values'].quantile(0.25)  
Q3 = df['values'].quantile(0.75)  
IQR = Q3 - Q1  
  
# Define outlier boundaries  
lower\_bound = Q1 - 1.5 \* IQR  
upper\_bound = Q3 + 1.5 \* IQR  
  
# Identify outliers  
outliers\_iqr = df[(df['values'] < lower\_bound) | (df['values'] > upper\_bound)]  
print("Outliers (IQR method):\n", outliers\_iqr)

This example demonstrates how to implement the IQR method using pandas to calculate quartiles and NumPy for logical operations 7.

The IQR method is robust to extreme outliers because it relies on percentiles, which are less influenced by extreme values than the mean and standard deviation 2. It also works well with skewed distributions as it focuses on the central 50% of the data 2. The method is relatively simple and effective to implement and interpret 16. However, the IQR method might not be as effective for very small datasets where the calculated quartiles might not accurately represent the underlying distribution 2. While some sources suggest it works well with skewed data, others note that it assumes the data is somewhat evenly spread within the IQR 8. Additionally, the IQR method primarily focuses on univariate outliers and might not effectively detect multivariate outliers 14. The effectiveness of the IQR method depends on having a sufficiently large and representative dataset for accurate quartile calculation. In very small datasets, the position of these quartiles can be significantly affected by individual values, potentially leading to unreliable outlier detection 2.

### **Median Absolute Deviation (MAD) Method**

The Median Absolute Deviation (MAD) method is another robust statistical measure of dispersion that is less sensitive to outliers than the standard deviation 2. It is calculated as the median of the absolute deviations from the data's median: MAD = median(|X<sub>i</sub> - median(X)|) 2. Outliers are often identified as data points that are a certain number of MADs away from the median, typically using a threshold of 3 or 3.5 2.

Python

import pandas as pd  
import numpy as np  
  
# Sample data  
data = {'values': }  
df = pd.DataFrame(data)  
  
# Calculate the median and MAD  
median\_val = df['values'].median()  
mad\_val = np.median(np.abs(df['values'] - median\_val))  
  
# Identify outliers (using a threshold of 3)  
threshold = 3  
outliers\_mad = df[np.abs(df['values'] - median\_val) / mad\_val > threshold]  
print("Outliers (MAD method):\n", outliers\_mad)

This code demonstrates the implementation of the MAD method using pandas to calculate the median and NumPy for the absolute deviation and median calculation 2.

The MAD method is highly robust to extreme outliers and is particularly suitable for skewed or non-normal data 2. However, it might be more complex to interpret compared to the Z-score method 2. The choice of the threshold (number of MADs) can also influence the results, and this method is less commonly used than Z-score or IQR, which might make it less familiar to some users. While the MAD method offers superior robustness, its interpretability might be a hurdle for some users accustomed to the more intuitive standard deviation-based Z-score. Understanding the scale of MAD relative to standard deviation for different distributions can aid in interpretation 2.

## **Visualization Techniques for Outlier Detection**

Visualizing data is an essential step in outlier analysis as it allows for a qualitative assessment of potential anomalies 9. Several Python libraries, such as matplotlib, seaborn, and plotly, offer tools for creating insightful visualizations 6.

### **Histograms**

Histograms provide a graphical depiction of a dataset's distribution, showing the frequency of observations within distinct intervals or bins 6. Outliers can be visually identified as solitary bins located far from the main body of the distribution, typically in the tails 6. Histograms can also help in understanding the modality of the data 23.

Python

import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.datasets import fetch\_california\_housing  
  
# Load the California Housing dataset  
data = fetch\_california\_housing(as\_frame=True)  
df = data['data']  
df['MedHouseVal'] = data['target']  
  
# Plot histogram for a single feature  
plt.hist(df['MedInc'], bins="auto")  
plt.title('Histogram of Median Income')  
plt.xlabel('Median Income')  
plt.ylabel('Frequency')  
plt.show()  
  
# Plot histograms for all numerical features  
df.hist(bins='auto', figsize=(10, 10))  
plt.tight\_layout()  
plt.show()

This example uses matplotlib to create histograms for the 'MedInc' feature and all numerical features in the California Housing dataset 6. Histograms offer a quick visual way to spot potential outliers by showing the overall shape of the data distribution and highlighting values that occur with very low frequency at the extremes. However, they might not be as effective for detecting outliers in high-dimensional datasets or subtle outliers within dense regions 6.

### **Box Plots**

Box plots, also known as box-and-whisker plots, offer a standardized way to visualize the distribution of data based on quartiles 9. The box represents the IQR, the line inside the box indicates the median, and the whiskers typically extend to 1.5 times the IQR from the quartiles. Data points outside the whiskers are often considered potential outliers and are plotted as individual points 9. Box plots are particularly useful for identifying univariate outliers and comparing distributions across different groups 6.

Python

import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
import plotly.express as px  
from sklearn.datasets import fetch\_california\_housing  
  
# Load the California Housing dataset  
data = fetch\_california\_housing(as\_frame=True)  
df = data['data']  
df['MedHouseVal'] = data['target']  
  
# Using seaborn  
sns.boxplot(x=df['MedHouseVal'])  
plt.title('Box Plot of Median House Value (Seaborn)')  
plt.show()  
  
# Using plotly  
fig = px.box(df, y="MedHouseVal", title='Box Plot of Median House Value (Plotly)')  
fig.show()

This example demonstrates creating box plots using both seaborn and plotly.express for the 'MedHouseVal' feature 9. Box plots offer a concise visual summary of the data's central tendency, spread, and potential outliers based on the IQR rule. They are especially effective for comparing the distribution of a single variable across different groups or for quickly identifying extreme values in a dataset 9.

### **Scatter Plots**

Scatter plots are used to visualize the relationship between two numerical variables 9. They can be helpful for identifying multivariate outliers, where a data point might not be an outlier in any single variable but is unusual when considering the combination of its values across multiple variables 9. Outliers will appear as points that are far away from the general cluster of data points in the scatter plot.

Python

import pandas as pd  
import matplotlib.pyplot as plt  
import plotly.express as px  
from sklearn.datasets import fetch\_california\_housing  
  
# Load the California Housing dataset  
data = fetch\_california\_housing(as\_frame=True)  
df = data['data']  
  
# Using matplotlib  
plt.scatter(df['MedInc'], df['MedHouseVal'])  
plt.xlabel('Median Income')  
plt.ylabel('Median House Value')  
plt.title('Scatter Plot of Median Income vs. Median House Value (Matplotlib)')  
plt.show()  
  
# Using plotly  
fig = px.scatter(df, x="MedInc", y="MedHouseVal", title='Scatter Plot of Median Income vs. Median House Value (Plotly)')  
fig.show()

This example shows how to create scatter plots using matplotlib and plotly.express to visualize the relationship between 'MedInc' and 'MedHouseVal' 9. Scatter plots are invaluable for detecting outliers in the joint distribution of two variables. Points that deviate significantly from the established relationship or the main cluster of points can indicate unusual combinations of values. However, scatter plots are limited to visualizing relationships between two or at most three variables 9.

## **Proximity-Based Outlier Detection Techniques**

Proximity-based methods identify outliers based on the relationships between data points, considering their distance or density relative to their neighbors 3. Two popular proximity-based techniques are the Local Outlier Factor (LOF) and k-Nearest Neighbors (kNN) 1.

### **Local Outlier Factor (LOF)**

The Local Outlier Factor (LOF) algorithm measures the local deviation of the density of a data point with respect to its neighbors 3. It identifies outliers as points with a substantially lower density than their neighbors. A LOF score significantly greater than 1 indicates an outlier 3. LOF is effective in datasets with varying densities and does not assume a specific data distribution 11.

Python

import numpy as np  
from sklearn.neighbors import LocalOutlierFactor  
  
# Sample data  
X = np.array([[1, 2], [1.5, 1.8], [3, 4], [8, 8], [1, 0.6], [5, 6]])  
  
# Initialize LOF detector  
clf = LocalOutlierFactor(n\_neighbors=2, contamination='auto')  
  
# Fit the detector and get outlier scores  
y\_pred = clf.fit\_predict(X)  
lof\_scores = clf.negative\_outlier\_factor\_  
  
# Identify outliers (where y\_pred == -1)  
outliers\_lof = X[y\_pred == -1]  
print("Outliers (LOF method):\n", outliers\_lof)  
print("LOF Scores:\n", lof\_scores)

This example uses the LocalOutlierFactor class from scikit-learn to detect outliers in a sample dataset 3. LOF is particularly useful for identifying local outliers that might not be detected by global methods. However, its performance is sensitive to the choice of the number of neighbors (n\_neighbors), and it can be computationally expensive for large datasets 11. The need to tune the n\_neighbors parameter based on the data's local structure can be challenging, and an inappropriate choice can lead to suboptimal results 11.

### **k-Nearest Neighbors (kNN)**

The k-Nearest Neighbors (kNN) approach to outlier detection identifies outliers based on the distance to their k-th nearest neighbor 1. Outliers are expected to have larger distances to their nearest neighbors compared to normal data points. The outlier score can be the distance to the k-th neighbor, the average distance, or the median distance 1.

Python

import numpy as np  
from pyod.models.knn import KNN  
  
# Sample data  
X = np.array([[1, 2], [1.5, 1.8], [3, 4], [8, 8], [1, 0.6], [5, 6]])  
  
# Initialize kNN detector  
clf = KNN(n\_neighbors=2)  
  
# Fit the detector and get outlier scores  
clf.fit(X)  
outlier\_scores = clf.decision\_scores\_  
  
# Identify outliers (using a threshold on the scores)  
threshold = np.percentile(outlier\_scores, 90) # Example threshold: top 10%  
outliers\_knn\_indices = np.where(outlier\_scores > threshold)  
outliers\_knn = X[outliers\_knn\_indices]  
print("Outliers (kNN method):\n", outliers\_knn)  
print("kNN Outlier Scores:\n", outlier\_scores)

This example uses the KNN class from the PyOD library to perform outlier detection 1. The kNN method is simple and intuitive, effective for identifying isolated outliers. However, its performance depends on the choice of k, and it can be computationally expensive for large datasets 11. The choice of k is critical; a small k might make the method sensitive to noise, while a large k might smooth out distances and fail to identify local outliers 11.

## **Model-Based Outlier Detection Techniques**

Model-based outlier detection techniques involve fitting a model to the normal data and then identifying data points that deviate significantly from this model 1. Two popular model-based methods are Isolation Forest and One-Class Support Vector Machines (OCSVM) 1.

### **Isolation Forest**

Isolation Forest is an efficient tree-based anomaly detection algorithm that isolates outliers by randomly partitioning the data 1. Outliers, being rare and different, are expected to be isolated in fewer splits (shorter paths in the trees) compared to normal data points.

Python

import numpy as np  
from sklearn.ensemble import IsolationForest  
  
# Sample data  
X = np.array([[1, 2], [1.5, 1.8], [3, 4], [8, 8], [1, 0.6], [5, 6], ])  
  
# Initialize Isolation Forest detector  
clf = IsolationForest(random\_state=42, contamination='auto')  
  
# Fit the detector and get outlier predictions  
y\_pred = clf.fit\_predict(X)  
  
# Get anomaly scores  
anomaly\_scores = clf.decision\_function(X)  
  
# Identify outliers (where y\_pred == -1)  
outliers\_iforest = X[y\_pred == -1]  
print("Outliers (Isolation Forest method):\n", outliers\_iforest)  
print("Anomaly Scores:\n", anomaly\_scores)

This example uses the IsolationForest class from scikit-learn 1. Isolation Forest is effective for high-dimensional data and large datasets and performs well with multimodal data 11. However, it is sensitive to the contamination parameter, which represents the expected proportion of outliers in the dataset 11. An inaccurate estimate of contamination can affect the algorithm's performance 11.

### **One-Class Support Vector Machines (OCSVM)**

One-Class Support Vector Machines (OCSVM) is an unsupervised learning algorithm that learns a decision boundary around the "normal" data 18. Data points falling outside this boundary are flagged as potential outliers. OCSVM is particularly effective for novelty detection.

Python

import numpy as np  
from sklearn.svm import OneClassSVM  
  
# Sample data  
X = np.array([[1, 2], [1.5, 1.8], [3, 4], [8, 8], [1, 0.6], [5, 6], ])  
  
# Initialize One-Class SVM detector  
clf = OneClassSVM(gamma='auto', nu=0.1) # nu parameter controls the sensitivity  
  
# Fit the detector and get outlier predictions  
y\_pred = clf.fit\_predict(X)  
  
# Identify outliers (where y\_pred == -1)  
outliers\_ocsvm = X[y\_pred == -1]  
print("Outliers (One-Class SVM method):\n", outliers\_ocsvm)

This example uses the OneClassSVM class from scikit-learn 18. OCSVM is effective for high-dimensional data and useful for novelty detection 18. However, its performance is sensitive to the choice of kernel and hyperparameters like nu and gamma, and it can be computationally expensive for large datasets 18.

## **Choosing the Right Outlier Detection Technique**

Selecting the appropriate outlier detection technique depends on various factors related to the data and the analysis goals 2. These factors include the data distribution (normal, skewed, multimodal, unknown), dataset size (small, medium, large), dimensionality (number of features), the type of outliers expected (global, contextual, collective), computational cost, interpretability requirements, and the availability of labeled data 2.

| **Technique** | **Underlying Concept** | **Advantages** | **Disadvantages** | **Best For** |
| --- | --- | --- | --- | --- |
| Z-score | Standard deviation from the mean | Simple, effective for normal distributions, provides numerical measure of extremeness. | Assumes normality, sensitive to extreme outliers, choice of threshold can be arbitrary, less effective for multiple outliers. | Approximately normally distributed, univariate data. |
| IQR | Spread of the middle 50% of the data (using quartiles) | Robust to extreme outliers, works well with skewed distributions, simple to implement and interpret. | Less effective for small datasets, primarily for univariate outliers, assumes even spread within IQR. | Skewed or non-normal, univariate data. |
| MAD | Median of absolute deviations from the median | Highly robust, ideal for skewed data. | More complex to interpret than Z-score, choice of threshold can influence results, less common. | Highly skewed or non-normal data. |
| LOF | Local density deviation from neighbors | Effective for varying densities, no distribution assumptions, provides anomaly scores. | Sensitive to parameter choice (n\_neighbors), computationally expensive for large datasets, can struggle in high dimensions. | Datasets with varying densities or clusters. |
| kNN | Distance to k-th nearest neighbor | Simple, intuitive, effective for isolated outliers, adaptable with different distance metrics. | Performance depends on k, computationally expensive for large datasets, might struggle with outliers in low-density regions. | Detecting isolated outliers. |
| Isolation Forest | Isolating outliers through random partitions | Effective for high-dimensional data, handles mixed variable types, efficient for large datasets, good for multimodal data. | Sensitive to contamination parameter, hyperparameter tuning might be needed, interpretation of scores can be challenging. | High-dimensional data, large datasets, multimodal data. |
| One-Class SVM | Learning a boundary around normal data | Effective for high-dimensional data, useful for novelty detection. | Sensitive to kernel and hyperparameters, computationally expensive for large datasets, might struggle with multi-clustered normal data. | High-dimensional data, novelty detection. |

Starting with exploratory data analysis (EDA) using visualizations like histograms and box plots can provide initial insights into potential outliers and the data distribution 9. For approximately normal, univariate data, Z-score is a good starting point 2. For skewed or non-normal univariate data, IQR or MAD are more robust 2. Multivariate outlier detection can benefit from scatter plots for two or three variables, while higher dimensions might require LOF, Isolation Forest, or OCSVM 3. For large datasets where computational efficiency is key, Isolation Forest is a viable option 3. Often, using multiple methods and comparing results can provide a more comprehensive understanding of potential outliers 18. It is always important to interpret detected outliers within the context of domain knowledge 3.

## **Handling Outliers After Detection**

Once outliers are detected, a decision needs to be made on how to handle them. Common approaches include removal (trimming), capping/winsorization, transformation, and imputation 2.

### **Removal (Trimming)**

Removing outliers involves excluding them from the dataset 2. This is appropriate when outliers are likely due to errors or are not representative of the population 2.

Python

import pandas as pd  
import numpy as np  
from sklearn.datasets import fetch\_california\_housing  
  
# Load the California Housing dataset  
data = fetch\_california\_housing(as\_frame=True)  
df = data['data']  
df['MedHouseVal'] = data['target']  
  
# Using IQR to identify bounds and remove  
Q1 = df['MedHouseVal'].quantile(0.25)  
Q3 = df['MedHouseVal'].quantile(0.75)  
IQR = Q3 - Q1  
lower\_bound = Q1 - 1.5 \* IQR  
upper\_bound = Q3 + 1.5 \* IQR  
df\_no\_outliers\_iqr = df[(df['MedHouseVal'] >= lower\_bound) & (df['MedHouseVal'] <= upper\_bound)]  
print("Original DataFrame shape:", df.shape)  
print("DataFrame shape after removing IQR outliers:", df\_no\_outliers\_iqr.shape)  
  
# Using Z-score to identify bounds and remove  
mean\_val = df['MedHouseVal'].mean()  
std\_val = df['MedHouseVal'].std()  
threshold = 3  
df\_no\_outliers\_z = df[((df['MedHouseVal'] - mean\_val) / std\_val).abs() <= threshold]  
print("DataFrame shape after removing Z-score outliers:", df\_no\_outliers\_z.shape)

Removal can lead to loss of information if outliers are genuine data points and should be done with caution and justification 2.

### **Capping/Winsorization**

Capping involves setting upper and lower limits for the data and replacing outlier values with these limits 2. For IQR, values below Q1 - 1.5 \* IQR are capped at the lower bound, and values above Q3 + 1.5 \* IQR are capped at the upper bound 2.

Python

import pandas as pd  
import numpy as np  
from sklearn.datasets import fetch\_california\_housing  
  
# Load the California Housing dataset  
data = fetch\_california\_housing(as\_frame=True)  
df = data['data']  
df['MedHouseVal'] = data['target']  
  
# Using IQR for capping  
Q1 = df['MedHouseVal'].quantile(0.25)  
Q3 = df['MedHouseVal'].quantile(0.75)  
IQR = Q3 - Q1  
lower\_bound = Q1 - 1.5 \* IQR  
upper\_bound = Q3 + 1.5 \* IQR  
df['MedHouseVal\_capped\_iqr'] = np.where(df['MedHouseVal'] < lower\_bound, lower\_bound,  
 np.where(df['MedHouseVal'] > upper\_bound, upper\_bound, df['MedHouseVal']))  
  
# Using Z-score for capping  
mean\_val = df['MedHouseVal'].mean()  
std\_val = df['MedHouseVal'].std()  
upper\_limit\_z = mean\_val + 3 \* std\_val  
lower\_limit\_z = mean\_val - 3 \* std\_val  
df['MedHouseVal\_capped\_z'] = np.where(df['MedHouseVal'] < lower\_limit\_z, lower\_limit\_z,  
 np.where(df['MedHouseVal'] > upper\_limit\_z, upper\_limit\_z, df['MedHouseVal']))  
  
print(df[['MedHouseVal', 'MedHouseVal\_capped\_iqr', 'MedHouseVal\_capped\_z']].head())

Capping retains all data but reduces variability and might distort true values 2.

### **Transformation**

Transformation involves applying mathematical functions (e.g., logarithmic, square root) to reduce the impact of outliers by compressing large values 2.

Python

import pandas as pd  
import numpy as np  
from sklearn.datasets import fetch\_california\_housing  
  
# Load the California Housing dataset  
data = fetch\_california\_housing(as\_frame=True)  
df = data['data']  
  
# Log transformation (example for a positive-valued column)  
df['MedInc\_log'] = np.log(df['MedInc'] + 1e-9) # Adding a small constant to handle potential zeros  
print(df[['MedInc', 'MedInc\_log']].head())

Transformation can make data interpretation more difficult, and the choice of transformation should be appropriate for the data 2.

### **Imputation**

Imputation involves replacing outlier values with estimated values, such as the mean or median of the non-outlier data 7.

Python

import pandas as pd  
import numpy as np  
from sklearn.datasets import fetch\_california\_housing  
  
# Load the California Housing dataset  
data = fetch\_california\_housing(as\_frame=True)  
df = data['data']  
df['MedHouseVal'] = data['target']  
  
# Using IQR to define non-outlier range and impute with median  
Q1 = df['MedHouseVal'].quantile(0.25)  
Q3 = df['MedHouseVal'].quantile(0.75)  
IQR = Q3 - Q1  
lower\_bound = Q1 - 1.5 \* IQR  
upper\_bound = Q3 + 1.5 \* IQR  
non\_outlier\_median = df[(df['MedHouseVal'] >= lower\_bound) & (df['MedHouseVal'] <= upper\_bound)]['MedHouseVal'].median()  
df['MedHouseVal\_imputed'] = np.where((df['MedHouseVal'] < lower\_bound) | (df['MedHouseVal'] > upper\_bound),  
 non\_outlier\_median, df['MedHouseVal'])  
  
print(df[['MedHouseVal', 'MedHouseVal\_imputed']].head())

Imputation can introduce bias if not done carefully, and the choice of imputation method should be considered 7.

## **Conclusion**

Outlier analysis is a crucial step in the data analysis pipeline, and Python provides a comprehensive set of tools and libraries to perform this task effectively. This report has explored several common techniques for outlier detection, including statistical methods like Z-score, IQR, and MAD, visualization techniques such as histograms, box plots, and scatter plots, and proximity-based and model-based algorithms like LOF, kNN, Isolation Forest, and OCSVM. Each technique has its own underlying principles, advantages, and disadvantages, making the choice of method dependent on the specific characteristics of the data and the objectives of the analysis. Furthermore, the report discussed various strategies for handling outliers after detection, including removal, capping, transformation, and imputation, each with its own implications for the subsequent analysis. Ultimately, a thorough understanding of the data and careful consideration of the available techniques are essential for effective outlier analysis and for ensuring the quality and reliability of data-driven insights.

#### Works cited

1. An Awesome Tutorial to Learn Outlier Detection in Python using PyOD Library, accessed March 30, 2025, <https://www.analyticsvidhya.com/blog/2019/02/outlier-detection-python-pyod/>
2. Outlier Detection and Treatment: Z-score, IQR, and Robust Methods | by Aakash R - Medium, accessed March 30, 2025, <https://medium.com/@aakash013/outlier-detection-treatment-z-score-iqr-and-robust-methods-398c99450ff3>
3. Episode #497 - Outlier Detection with Python, accessed March 30, 2025, <https://talkpython.fm/episodes/show/497/outlier-detection-with-python>
4. Z score for Outlier Detection - Python - GeeksforGeeks, accessed March 30, 2025, <https://www.geeksforgeeks.org/z-score-for-outlier-detection-python/>
5. Detecting Outliers Using Z-score — part -2 | by Irshad Alam | Medium, accessed March 30, 2025, <https://medium.com/@irshadalamtech/detecting-outliers-using-z-score-part-2-85297f54ea84>
6. How To Find Outliers in Data Using Python (and How To Handle Them) - CareerFoundry, accessed March 30, 2025, <https://careerfoundry.com/en/blog/data-analytics/how-to-find-outliers/>
7. Guide on Outlier Detection Methods - Analytics Vidhya, accessed March 30, 2025, <https://www.analyticsvidhya.com/blog/2021/05/feature-engineering-how-to-detect-and-remove-outliers-with-python-code/>
8. Outlier Detection: Spotting the Odd One Out - Let's Data Science, accessed March 30, 2025, <https://letsdatascience.com/outlier-detection/>
9. Detect and Remove the Outliers using Python - GeeksforGeeks, accessed March 30, 2025, <https://www.geeksforgeeks.org/detect-and-remove-the-outliers-using-python/>
10. How to detect outliers with z-score - Machine Learning Plus, accessed March 30, 2025, <https://www.machinelearningplus.com/machine-learning/how-to-detect-outliers-with-z-score/>
11. Top 5 Outlier Detection Methods Every Data Enthusiast Must Know - DataHeroes, accessed March 30, 2025, <https://dataheroes.ai/blog/outlier-detection-methods-every-data-enthusiast-must-know/>
12. Outliers: Detecting Outliers Using Z Scores: A Statistical Approach - FasterCapital, accessed March 30, 2025, <https://fastercapital.com/content/Outliers--Detecting-Outliers-Using-Z-Scores--A-Statistical-Approach.html>
13. 1.5 IQR Rule Explained | Built In, accessed March 30, 2025, <https://builtin.com/articles/1-5-iqr-rule>
14. (IQR Formula) The Interquartile Range Method For Outliers - ProCogia, accessed March 30, 2025, <https://procogia.com/interquartile-range-method-for-reliable-data-analysis/>
15. Interquartile Range to Detect Outliers in Data - GeeksforGeeks, accessed March 30, 2025, <https://www.geeksforgeeks.org/interquartile-range-to-detect-outliers-in-data/>
16. Outlier Detection and Removal using the IQR Method | by Paresh Patil | Medium, accessed March 30, 2025, <https://medium.com/@pp1222001/outlier-detection-and-removal-using-the-iqr-method-6fab2954315d>
17. Interquartile Statistics: Understanding the Basics and Applications - Decoding Data Science, accessed March 30, 2025, <https://decodingdatascience.com/interquartile-statistics-understanding-the-basics-and-applications/>
18. Outlier Detection (with examples) - Hex, accessed March 30, 2025, <https://hex.tech/templates/data-science/outlier-detection/>
19. How to Use Interquartile Range to Identify Outliers - YouTube, accessed March 30, 2025, <https://www.youtube.com/watch?v=5ETv8-gwM0A>
20. Z-Scores vs. Box Plots: The Quirky Quest to Unearth Outliers in Statistical Wonderland | by Patwariraghottam | DevOps.dev, accessed March 30, 2025, <https://blog.devops.dev/z-scores-vs-box-plots-the-quirky-quest-to-unearth-outliers-in-statistical-wonderland-eb2f0489984e>
21. Outlier detection and removal using IQR - Kaggle, accessed March 30, 2025, <https://www.kaggle.com/code/dilipkumar2k6/outlier-detection-and-removal-using-iqr>
22. 5 Easy Ways to Detect Outliers in Python | by Marc Bolle - Medium, accessed March 30, 2025, <https://medium.com/@marc.bolle/5-easy-ways-to-detect-outliers-in-python-b07639ba32d5>
23. A Guide to Outlier Detection in Python | Built In, accessed March 30, 2025, <https://builtin.com/data-science/outlier-detection-python>
24. yzhao062/pyod: A Python Library for Outlier and Anomaly Detection, Integrating Classical and Deep Learning Techniques - GitHub, accessed March 30, 2025, <https://github.com/yzhao062/pyod>