Outlier Analysis-1

February 8, 2025

1 Outlier Detection in EDA

This notebook covers the fundamentals of outlier detection as part of Exploratory Data Analysis (EDA). It details various methods, mathematical concepts, and Python examples to illustrate how to detect outliers using statistical and distance-based techniques. We will explore topics such as the z-score, IQR method, proximity-based approaches, different distance metrics, Mahalanobis distance, and outlier detection in high-dimensional data.

1.1 Outliers and Outlier Analysis

An **outlier** is a data point that significantly deviates from other observations. Outliers can affect statistical analyses and model performance. Two common statistical methods for identifying outliers are:

- **z-Score Method:** For an observation x, the z-score is calculated as

  $$z = \frac{x-\mu}{\sigma}$$

  where μ is the mean and σ is the standard deviation. Points with $|z| > \tau$ (commonly, $\tau = 3$) are considered outliers.

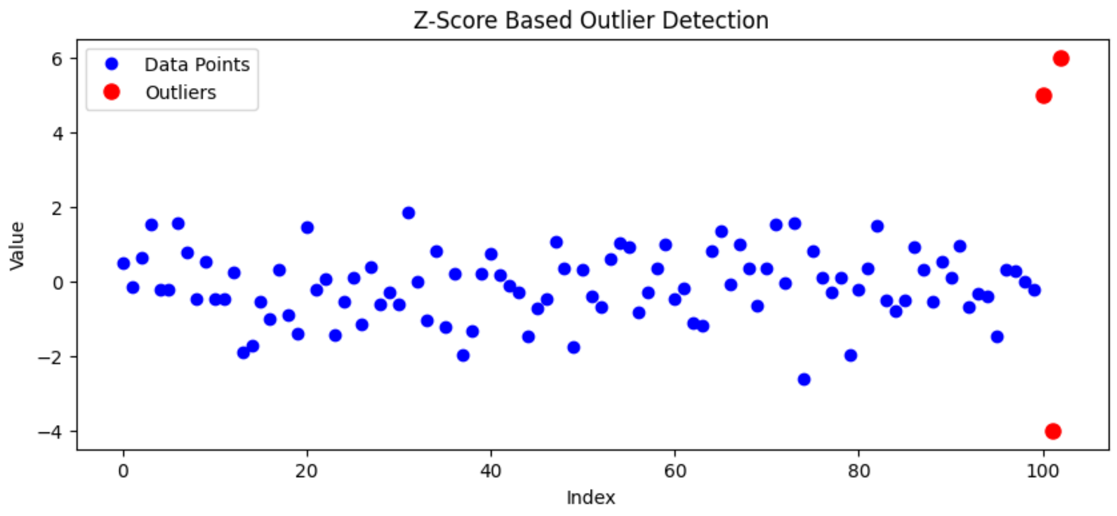
- **IQR Method:** Outliers are defined as points lying below

  $$Q_1 - 1.5 \times IQR \quad \text{or} \quad Q_3 + 1.5 \times IQR,$$

  where $IQR = Q_3 - Q_1$, and Q_1 and Q_3 are the first and third quartiles, respectively.

[5]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Generate synthetic data with a few outliers
np.random.seed(42)
data = np.random.normal(loc=0, scale=1, size=100)
data = np.concatenate([data, np.array([5, -4, 6])])  # adding outliers

# Calculate z-scores
z_scores = np.abs(stats.zscore(data))
threshold = 3
outliers = np.where(z_scores > threshold)[0]
```

```python

# Visualization
plt.figure(figsize=(10, 4))
plt.plot(data, 'bo', label='Data Points')
plt.plot(outliers, data[outliers], 'ro', markersize=8, label='Outliers')
plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Z-Score Based Outlier Detection')
plt.legend()
plt.show()
```



1.2 Outlier Detection Methods

Several approaches can be used for outlier detection:

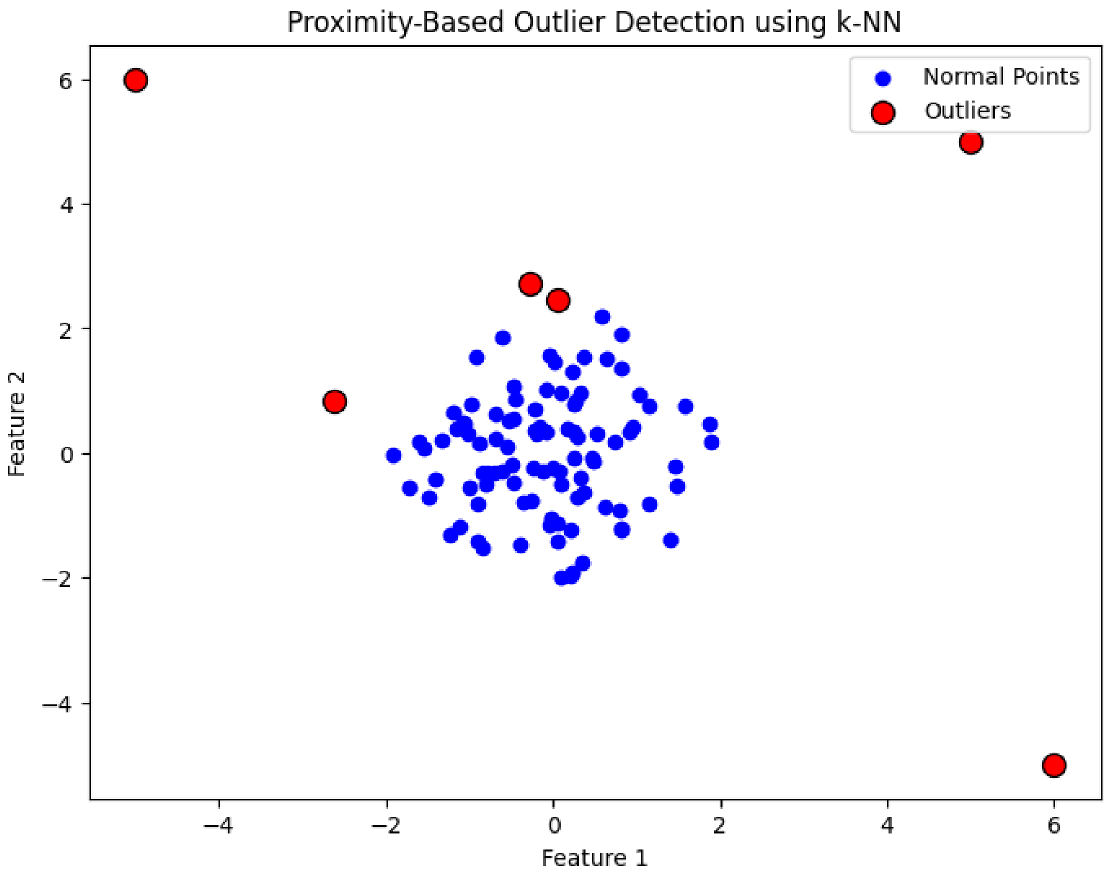
1. **Statistical Methods:** Such as the z-score and IQR methods.
2. **Distance-Based Methods:** These assume that normal observations occur in dense regions, while outliers are isolated.
3. **Proximity-Based Approaches:** These methods analyze the closeness of a data point to its neighbors to determine if it is an outlier.

1.3 Proximity-Based Outlier Detection using k-NN

In proximity-based methods, the average distance of a point to its k nearest neighbors is computed. Points with unusually high average distances are considered outliers.

[2]:
```python
from sklearn.neighbors import NearestNeighbors

# Generate synthetic 2D data with outliers
np.random.seed(42)
```

```python
X = np.random.randn(100, 2)
outlier_points = np.array([[5, 5], [6, -5], [-5, 6]])
X = np.vstack([X, outlier_points])

# Fit k-NN (using k=5)
nbrs = NearestNeighbors(n_neighbors=5)
nbrs.fit(X)
distances, indices = nbrs.kneighbors(X)

# Compute the mean distance to the 5 nearest neighbors
mean_distances = distances.mean(axis=1)

# Determine a threshold (e.g., 95th percentile) for outlier detection
threshold = np.percentile(mean_distances, 95)
detected_outliers = np.where(mean_distances > threshold)[0]

# Visualization
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c='blue', label='Normal Points')
plt.scatter(X[detected_outliers, 0], X[detected_outliers, 1], c='red',␣
 ↪label='Outliers', edgecolors='k', s=100)
plt.title('Proximity-Based Outlier Detection using k-NN')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```



1.4 Distance Metrics

Distance metrics are used to quantify the similarity (or dissimilarity) between data points. Common metrics include:

- **Euclidean Distance:**

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- **Manhattan Distance:**

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

- **Minkowski Distance:**

$$d(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

where setting $p = 2$ gives the Euclidean distance and $p = 1$ gives the Manhattan distance.

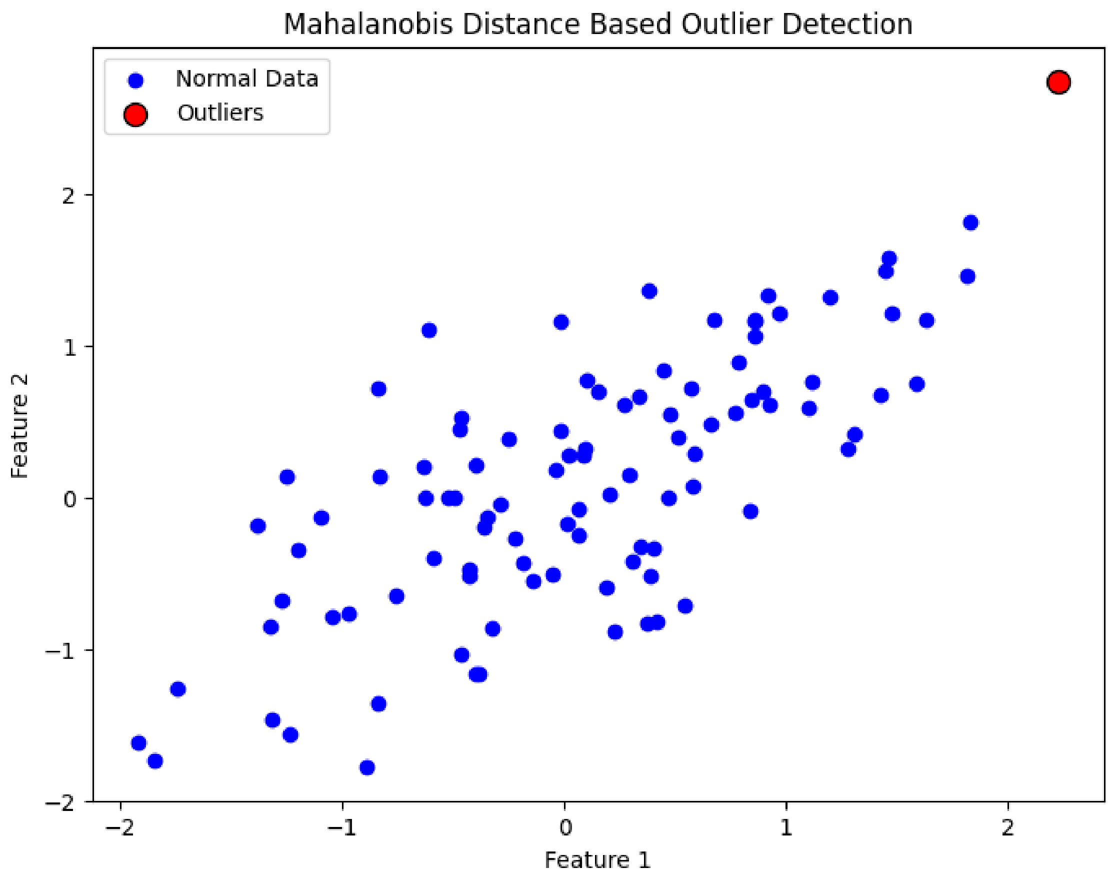
1.5 Mahalanobis Distance

The **Mahalanobis distance** is a multivariate metric that accounts for the covariance among variables. For a data point x, the distance from the mean μ is given by:

$$D_M(x) = \sqrt{(x-\mu)^T S^{-1} (x-\mu)}$$

where S is the covariance matrix of the data. This metric is particularly useful in multivariate analysis, as it normalizes the scale and accounts for variable correlations.

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chi2

# Generate synthetic bivariate data
np.random.seed(42)
mean = np.array([0, 0])
cov = np.array([[1, 0.8],
                [0.8, 1]])
data = np.random.multivariate_normal(mean, cov, size=100)

# Calculate the mean and covariance matrix
data_mean = np.mean(data, axis=0)
cov_matrix = np.cov(data, rowvar=False)
inv_cov_matrix = np.linalg.inv(cov_matrix)

# Compute Mahalanobis distance for each observation
diff = data - data_mean
md = np.sqrt(np.sum(diff @ inv_cov_matrix * diff, axis=1))

# Determine a threshold based on the Chi-Square distribution (99% quantile)
p = data.shape[1]  # number of dimensions
threshold = np.sqrt(chi2.ppf(0.99, df=p))

# Identify outliers
outliers = np.where(md > threshold)[0]

# Visualization
plt.figure(figsize=(8, 6))
plt.scatter(data[:, 0], data[:, 1], c='blue', label='Normal Data')
plt.scatter(data[outliers, 0], data[outliers, 1], c='red', label='Outliers',␣
 ↪edgecolors='k', s=100)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Mahalanobis Distance Based Outlier Detection')
plt.legend()
plt.show()
```

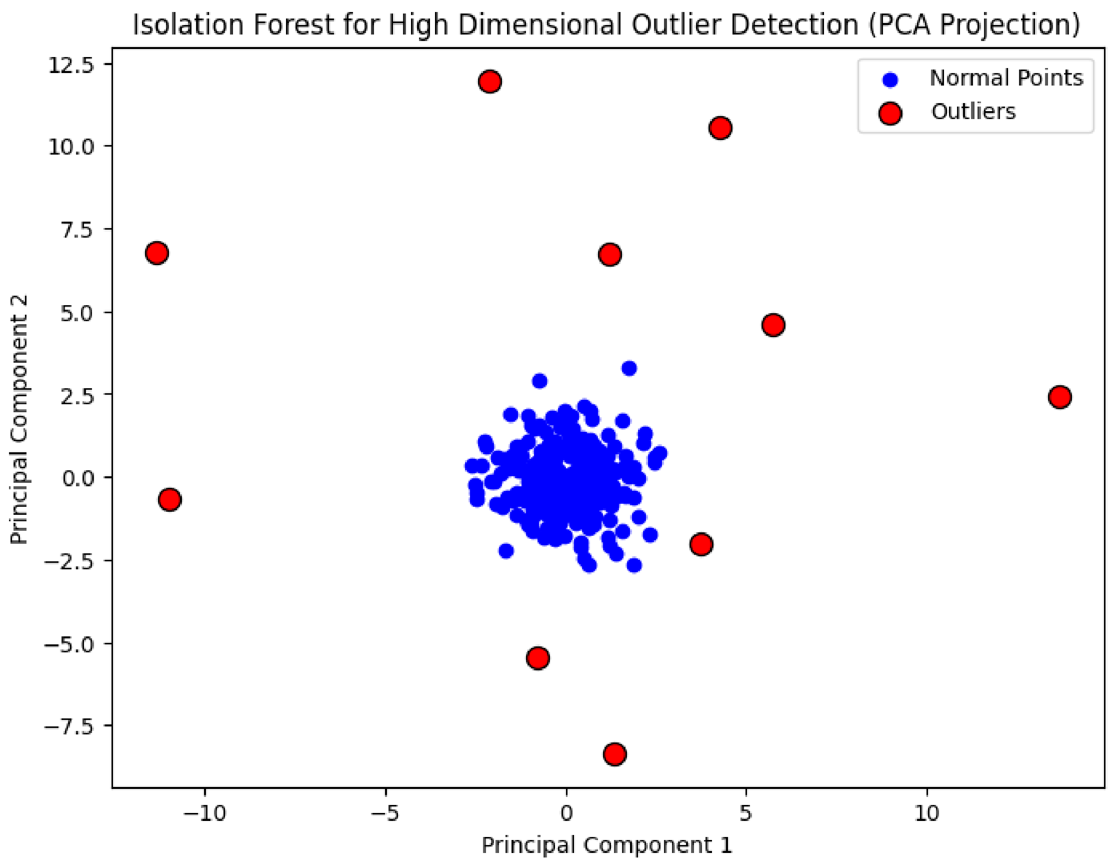


1.6 Outlier Detection in High Dimensional Data

High-dimensional data presents unique challenges for outlier detection due to the **curse of dimensionality**. In high dimensions, distances between points become less distinguishable. To address this, techniques such as dimensionality reduction (e.g., PCA) and advanced algorithms like Isolation Forest are used to detect outliers more effectively.

[4]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest
from sklearn.decomposition import PCA

# Generate synthetic high-dimensional data (e.g., 10 dimensions)
np.random.seed(42)
X = np.random.randn(300, 10)
# Inject some outliers
outliers = np.random.uniform(low=-8, high=8, size=(10, 10))
X = np.vstack([X, outliers])
```

```
# Apply Isolation Forest
iso_forest = IsolationForest(contamination=0.03, random_state=42)
pred = iso_forest.fit_predict(X)  # -1 indicates outliers, 1 indicates normal␣
 ↪points

# For visualization, reduce dimensionality to 2D using PCA
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

plt.figure(figsize=(8, 6))
plt.scatter(X_reduced[pred == 1, 0], X_reduced[pred == 1, 1], c='blue',␣
 ↪label='Normal Points')
plt.scatter(X_reduced[pred == -1, 0], X_reduced[pred == -1, 1], c='red',␣
 ↪label='Outliers', edgecolors='k', s=100)
plt.title('Isolation Forest for High Dimensional Outlier Detection (PCA␣
 ↪Projection)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```



```
[ ]:
```