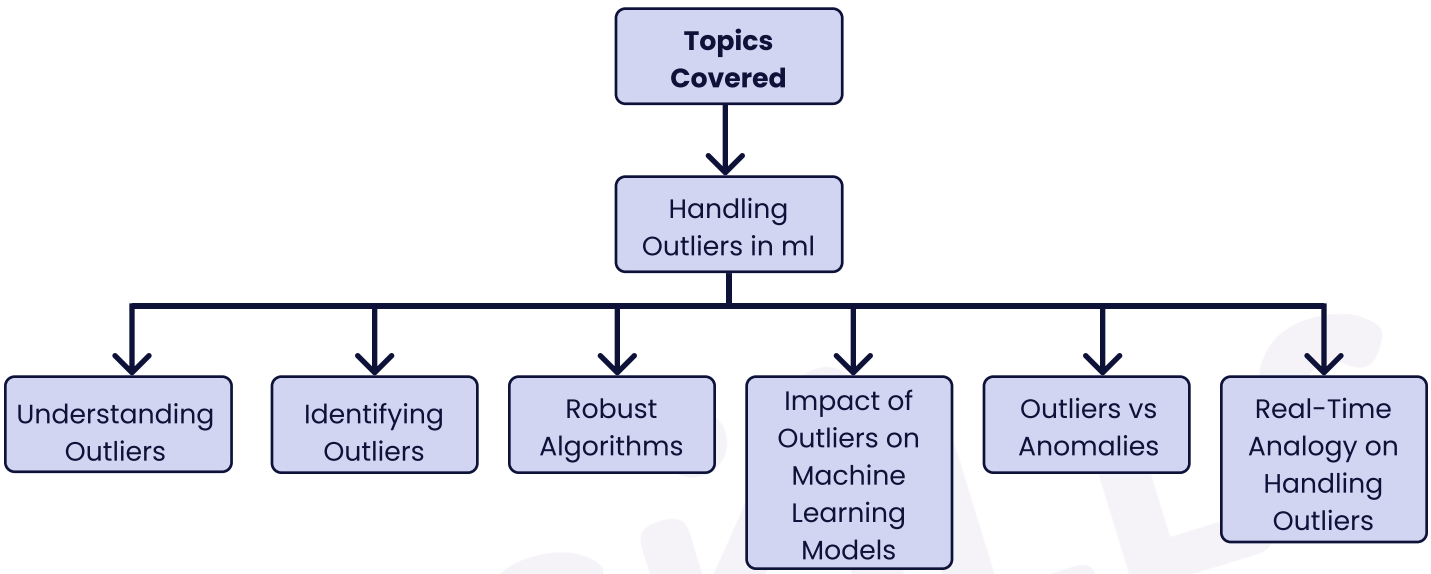
PW SKILLS

Lesson Plan

Handling Outliers

Topics Covered:

- Understanding Outliers
- Identifying Outliers
- Robust Algorithms
- Impact of Outliers on Machine Learning Models
- Outliers vs Anomalies
- Real-Time Analogy on Handling Outliers



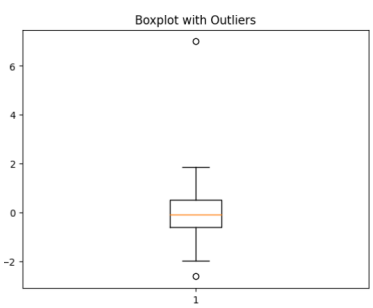
Understanding Outliers

- Outliers are data points that significantly differ from other observations in a dataset.
- They can arise due to measurement errors, variability in the data, or genuine extreme values.
- Outliers can skew statistical analyses, affect model performance, and distort visualizations.

Python Code

```
import matplotlib.pyplot as plt
import numpy as np

# Generating random data with outliers
np.random.seed(42)
data = np.random.normal(0, 1, 100)
data[5] = 7  # Introducing an outlier
# Visualizing data with box plot
plt.boxplot(data)
plt.title('Boxplot with Outliers')
plt.show()
```

Output:



Identifying Outliers

- Z-score: Identifies outliers by calculating the deviation of a data point from the mean in terms of standard deviations.
- One of the most commonly used tools in determining outliers is the Z-score. The Z-score is just the number of standard deviations away from the mean that a certain data point is.
- The statistical formula for a value's z-score is calculated using the following formula:
  - z = (x - μ) / σ
- Where:
  - z = Z-score
  - x = the value being evaluated
  - μ = the mean
  - σ = the standard deviation

Python code:

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

# Generate random data
np.random.seed(42)
data = np.random.normal(0, 1, 100)

# Calculate Z-scores for the data
z_scores = stats.zscore(data)
print("The Z-score of data \n", z_scores)
```

Output:

```
The Z-score of data
 [ 0.66461921 -0.03808894  0.83169721  1.80040643 -0.14420598 -0.14418781
  1.86258212  0.96421608 -0.4046274   0.71535523 -0.39792468 -0.40048335
  0.38269433 -2.00243618 -1.79398218 -0.50734042 -1.00594105  0.46268965
 -0.8899548  -1.4480232   1.73690481 -0.13493537  0.18965433 -1.46179507
 -0.48752581  0.23767736 -1.15884054  0.53069494 -0.54978229 -0.20788388
 -0.55096412  2.16477389  0.09998636 -1.05560794  1.02520457 -1.23614114
  0.3460652  -2.05377422 -1.35493317  0.33278261  0.93215805  0.30457046
 -0.0130606  -0.21829753 -1.52130463 -0.68170281 -0.39484934  1.28480297
  0.49519344 -1.8361708   0.4735755  -0.31123365 -0.63420233  0.79184373
  1.2558939   1.14553797 -0.81380903 -0.22727112  0.48152076  1.19452448
 -0.41536189 -0.09053895 -1.10941845 -1.20887622  1.01411679  1.61582604
  0.03523221  1.22549756  0.51513303 -0.59900791  0.51486697  1.81701382
  0.07527585  1.84645895 -2.78425581  1.02449364  0.21125517 -0.21597758
  0.21647166 -2.08464882 -0.12817983  0.51012709  1.75045623 -0.45862802
 -0.77980796 -0.44035348  1.12796634  0.47874046 -0.47134358  0.68293814
  0.22235554  1.18688834 -0.66201402 -0.24768883 -0.31900894 -1.50469687
  0.44262908  0.40382388  0.12058214 -0.14468601]
```

```
# Plotting the Z-scores
plt.figure(figsize=(8, 5))

plt.scatter(np.arange(len(data)), data, c=z_scores, cmap='viridis')
plt.colorbar(label='Z-score')
plt.title('Z-scores for Random Data')
plt.xlabel('Index')
plt.ylabel('Data Value')
```

```
plt.show()
```

Output:

- **Interquartile Range (IQR):** Determines outliers based on the range between the 25th and 75th percentiles.
- The Interquartile Range (IQR) is a robust statistical measure used to identify outliers in a dataset.
- It is the range between the 75th percentile (Q3) and the 25th percentile (Q1) of the data.
- The formula for finding the interquartile range takes the third quartile value and subtracts the first quartile value.
- IQR = Q3 – Q1
- To find outliers, you'll need to know your data's IQR, Q1, and Q3 values. Take these values and input them into the equations below. Statisticians call the result for each equation an outlier gate.
- Q1 - 1.5 * IQR: Lower outlier gate.
- Q3 + 1.5 * IQR: Upper outlier gate.

Python Code

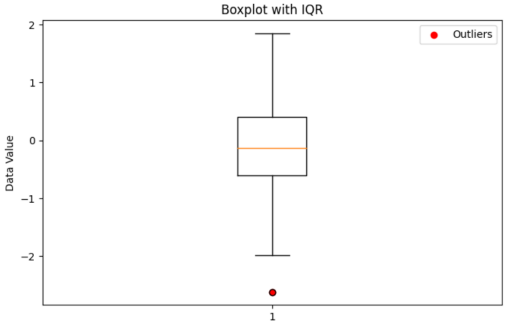
```
import numpy as np
import matplotlib.pyplot as plt

# Generate random data
np.random.seed(42)
data = np.random.normal(0, 1, 100)

# Calculate quartiles (Q1, Q3) and IQR
q1 = np.percentile(data, 25)
q3 = np.percentile(data, 75)
iqr = q3 - q1

# Identify potential outliers using IQR
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr
outliers = [x for x in data if x < lower_bound or x > upper_bound]
print("The Outliers are \n", outliers)
```

Output:

```
The Outliers are
 [-2.6197451040897444]
```

```
# Plotting the data and IQR
plt.figure(figsize=(8, 5))

plt.boxplot(data)
plt.title('Boxplot with IQR')
plt.ylabel('Data Value')

# Highlighting potential outliers
if outliers:
    plt.scatter(np.ones(len(outliers)), outliers, color='red',
label='Outliers')

plt.legend()
plt.show()
```

Output:



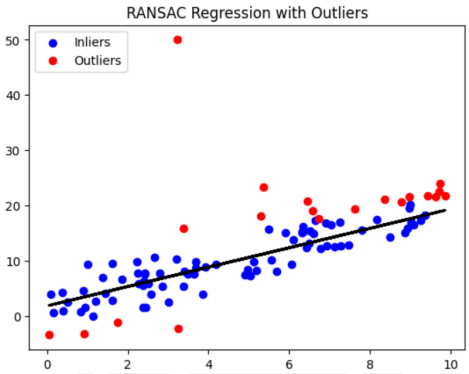
Robust Algorithms

- Robust algorithms are less sensitive to outliers and handle them better than traditional algorithms.
- Robust regression techniques like RANSAC or Huber Regression are resilient to outliers.

Python Code:

```
from sklearn.linear_model import RANSACRegressor

# Creating data for robust regression
x = np.random.rand(100, 1) * 10
y = 2 * x + 1 + np.random.randn(100, 1) * 3
y[5] = 50  # Introducing an outlier

# Applying RANSAC regression
ransac = RANSACRegressor()
ransac.fit(x, y)
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)

# Visualizing RANSAC regression with outliers
plt.scatter(x[inlier_mask], y[inlier_mask], color='blue', label='Inliers')
plt.scatter(x[outlier_mask], y[outlier_mask], color='red',
label='Outliers')
plt.plot(x, ransac.predict(x), color='black', linewidth=2)
plt.legend(loc='upper left')
plt.title('RANSAC Regression with Outliers')
plt.show()
```

Output:



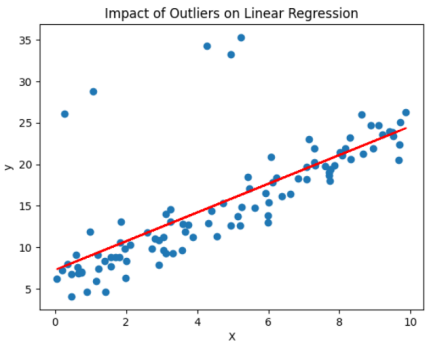
Impact of Outliers on Machine Learning Models

- Outliers can significantly influence the performance of machine learning models by skewing the results.
- For instance, in linear regression, outliers can heavily impact the slope and intercept of the regression line, affecting the model's predictions.

Python Code:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Generating sample data with outliers
np.random.seed(42)
X = np.random.rand(100, 1) * 10
y = 2 * X + 5 + np.random.randn(100, 1) * 2
y[95:] += 20  # Introducing outliers

# Fitting a linear regression model without handling outliers
regression = LinearRegression()
regression.fit(X, y)

# Plotting the data points and regression line
plt.scatter(X, y)
plt.plot(X, regression.predict(X), color='red')
plt.title('Impact of Outliers on Linear Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.show()
```

Output:



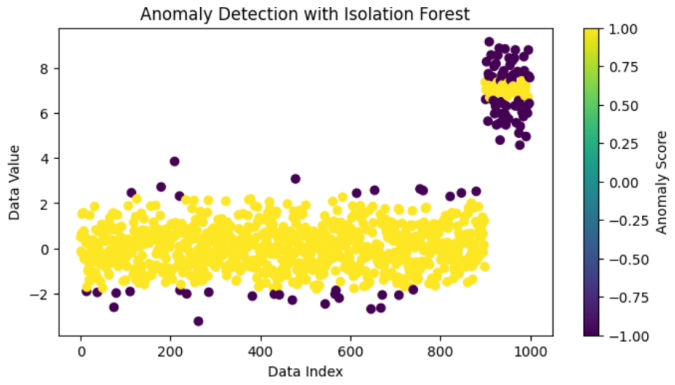
Outliers vs Anomalies

- Outliers are data points that deviate significantly from other observations.
- while anomalies are points that are rare or unexpected. Anomalies might not always be outliers and vice versa.

Python code:

```
from sklearn.ensemble import IsolationForest

# Generating a synthetic dataset
np.random.seed(42)
data = np.concatenate([np.random.normal(0, 1, 900), np.random.normal(7, 1,
100)])

# Using Isolation Forest for anomaly detection
model = IsolationForest(contamination=0.1)  # Assuming 10% anomalies
model.fit(data.reshape(-1, 1))

# Predicting anomalies
anomaly_preds = model.predict(data.reshape(-1, 1))
anomaly_indices = np.where(anomaly_preds == -1)

# Visualizing anomalies
plt.figure(figsize=(8, 4))
plt.scatter(range(len(data)), data, c=anomaly_preds, cmap='viridis')
plt.title('Anomaly Detection with Isolation Forest')
plt.xlabel('Data Index')
plt.ylabel('Data Value')
plt.colorbar(label='Anomaly Score')
plt.show()
```

Output:



Real-Time Analogy

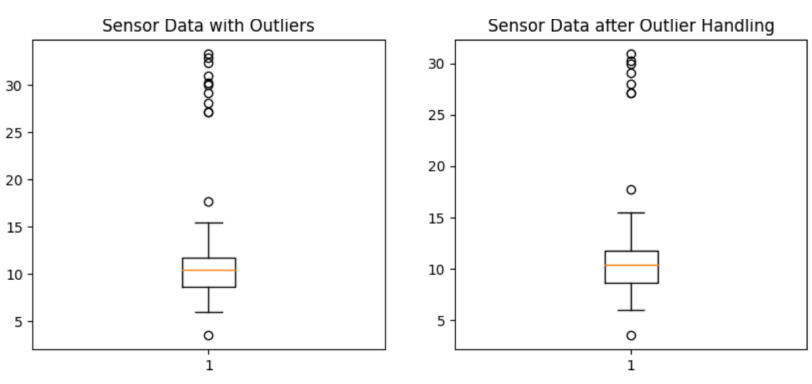
- Different domains might have specific considerations when dealing with outliers. For instance, in sensor data, outliers might occur due to measurement errors or equipment malfunction.

Code:

```
# Generating simulated sensor data with outliers
sensor_data = np.random.normal(loc=10, scale=2, size=100)
sensor_data[90:] += 20  # Introducing outliers

# Identifying outliers using z-score method
z_scores = (sensor_data - np.mean(sensor_data)) / np.std(sensor_data)
threshold = 3
outlier_indices = np.where(np.abs(z_scores) > threshold)
```

```
# Handling outliers by replacing them with the mean value
sensor_data_cleaned = sensor_data.copy()
sensor_data_cleaned[outlier_indices] = np.mean(sensor_data)

# Displaying before and after handling outliers
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.boxplot(sensor_data)
plt.title('Sensor Data with Outliers')

plt.subplot(1, 2, 2)
plt.boxplot(sensor_data_cleaned)
plt.title('Sensor Data after Outlier Handling')
plt.show()
```

Output:



PW SKILLS

THANK YOU !