Lecture content created by: Sulaiman Ahmed.
Sharing the content without proper credit is prohibited.
email: sulaimanahmed013@gmail.com
LinkedIn: www.linkedin.com/in/sulaimanahmed
Website: www.sulaimanahmed013.wixsite.com/my-site

Normal Distribution, Standard Normal Distribution, and Standardization

Table of Contents

📖 Table of Contents

        - Mean, Median, Mode
- ➤ Empirical Rule (68-95-99.7 Rule)
    - Explanation
    - Breakdown
        - 68%
        - 95%
        - 99.7%
- ➤ Standard Deviation and its Interpretation
    - Definition
    - In the Context of Normal Distribution
- ➤ Z-Score
    - Definition
    - Formula
    - Interpretation
- ➤ Standard Normal Distribution
    - Definition
    - Transformation using Z-Score
    - Importance

III. Data Scaling Techniques

- ➤ Standardization
    - Definition
    - Method
    - Applications in Machine Learning
- ➤ Normalization
    - Definition
    - Methods

        - Min-Max Scaling
        - Other Scaling Methods
    - Applications

IV. Practical Examples

- ➤ Comparing Performances Using Z-Scores
- ➤ The Exam Score Problem
    - Problem
    - Solution
        - Python Implementation
        - R Implementation
- ➤ Detecting Outliers with Z-Scores
- ➤ Outlier Detection in Website Traffic
    - Scenario
    - Python Implementation
    - R Implementation

I. Introduction to Distributions

- **What is a Distribution?** A distribution in statistics describes how data points are spread out or clustered within a data set.
- **Importance of Distributions:** Understanding data distribution provides valuable insights into the nature of the data itself.

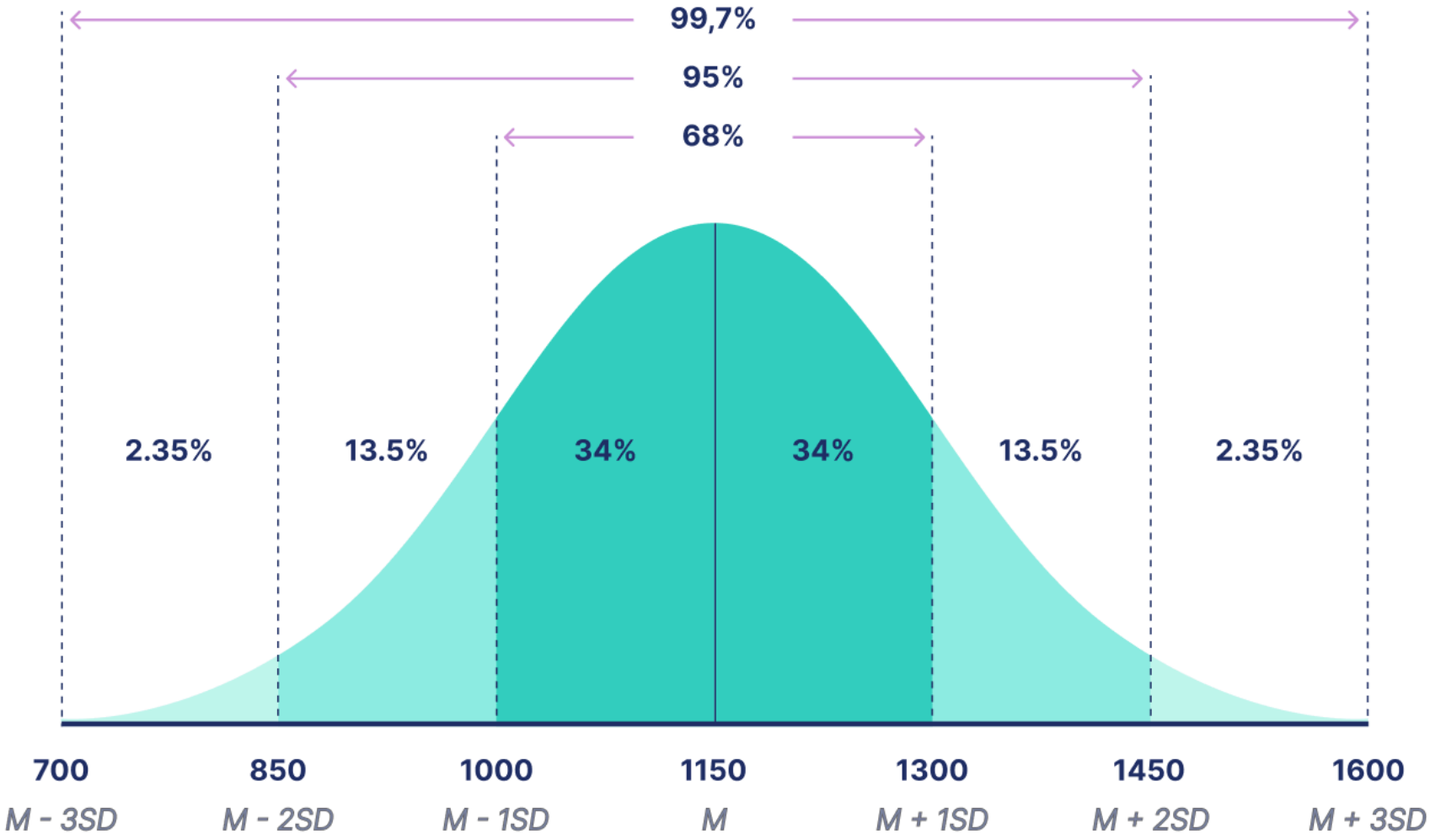
II. Gaussian (Normal) Distribution

- **Definition:** The Gaussian distribution, also known as the normal distribution, is a bell-shaped, symmetrical distribution that is commonly observed in various natural phenomena and statistical analyses.
- **Key Characteristics:**
    - **Bell Curve:** The distribution takes the shape of a symmetrical bell curve.
    - **Symmetry:** The left and right sides of the curve are mirror images.

- **Mean, Median, Mode:** The mean, median, and mode of the distribution are all equal and located at the center of the curve.

III. Empirical Rule (68-95-99.7 Rule)

Using the empirical rule in a normal distribution



Scribbr

- **Explanation:** The empirical rule, also known as the 68-95-99.7 rule, describes the percentage of data that falls within specific standard deviation ranges in a normal distribution.
- **Breakdown:**
  - **68%:** Approximately 68% of the data falls within one standard deviation of the mean (on both sides).
  - **95%:** Approximately 95% of the data falls within two standard deviations of the mean.
  - **99.7%:** Approximately 99.7% of the data falls within three standard deviations of the mean.

IV. Standard Deviation and its Interpretation

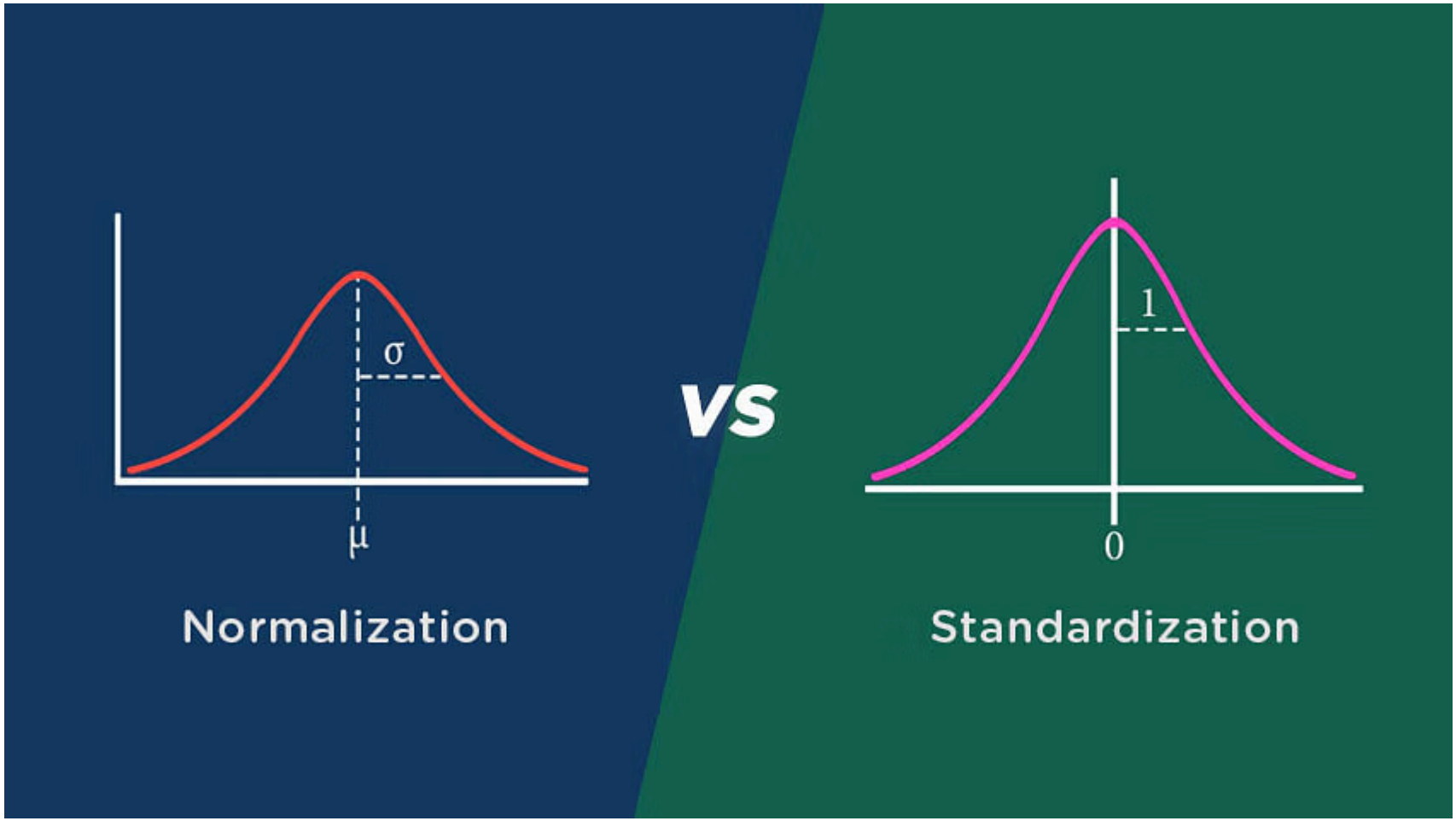
- **Definition:** Standard deviation measures the spread or dispersion of data points around the mean.
- **In the Context of Normal Distribution:**
  - One standard deviation to the right (or left) of the mean represents a specific distance along the x-axis of the bell curve.
  - The empirical rule uses standard deviations to define the ranges within which specific percentages of data fall.

V. Z-Score

- **Definition:** The z-score is a standardized score that measures how many standard deviations a specific data point (x_i) is away from the mean (μ) of the distribution.
- **Formula:** $z = (x_i - \mu) / \sigma$
  - x_i: Individual data point
  - μ: Mean of the distribution
  - σ: Standard deviation of the distribution
- **Interpretation:**
  - A positive z-score indicates the data point is above the mean.
  - A negative z-score indicates the data point is below the mean.
  - The magnitude of the z-score indicates the distance from the mean in terms of standard deviations.

VI. Standard Normal Distribution

- **Definition:** The standard normal distribution is a special case of the normal distribution where the mean (μ) is 0 and the standard deviation (σ) is 1.
- **Transformation using Z-Score:** Any normal distribution can be transformed into a standard normal distribution by applying the z-score formula to each data point.
- **Importance:** Standardizing data using z-scores allows for easier comparison of data sets with different units and scales.



VII. Standardization

- **Definition:** Standardization is the process of transforming data to have a mean of 0 and a standard deviation of 1, essentially converting it to a standard normal distribution.
- **Method:** Primarily achieved through the application of the z-score formula.
- **Applications in Machine Learning:**
  - Ensures features with different units (e.g., age in years, salary in dollars) contribute equally to model training.
  - Improves the performance and stability of many machine learning algorithms.

Data Standardization: Scaling House Prices and Sizes

Scenario:

You are working with a dataset containing house prices and sizes (in square feet). You want to apply a machine learning model to predict house prices. However, the features (price and size) have different scales.

- **Price:** Ranges from 200,000 to 1,000,000
- **Size:** Ranges from 1,200 to 3,500

This difference in scales can negatively impact some machine learning algorithms. Standardization can help address this issue.

Python Implementation (using StandardScaler):

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Sample data (house prices and sizes)
data = {'Price': [250000, 300000, 500000, 800000, 200000],
        'Size': [1200, 1800, 2500, 3200, 1500]}
df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)

# Create a StandardScaler object
scaler = StandardScaler()

# Fit the scaler to the data and transform the features
df[['Price', 'Size']] = scaler.fit_transform(df[['Price', 'Size']])

print("\nStandardized DataFrame:")
print(df)
```

Explanation:

1. **Data Preparation:**
   - Create a Pandas DataFrame with house prices and sizes.
2. **StandardScaler:**
   - Import `StandardScaler` from `sklearn.preprocessing`.
   - Create a `StandardScaler` object. This object will be used to standardize the data.
3. **Fit and Transform:**
   - **`fit_transform()`:** This method does two things:
     - **`fit()`:** Calculates the mean and standard deviation of the features in the training set.

    - `transform()`: Applies the standardization transformation using the calculated mean and standard deviation.
- We apply `fit_transform` to the 'Price' and 'Size' columns to standardize them.

Output:

The code will print the original DataFrame and then the standardized DataFrame. In the standardized DataFrame:

- **Price and Size:** The values will be transformed to have a mean (μ) of 0 and a standard deviation (σ) of 1.

Benefits of Standardization:

- **Improved Model Performance:** Many machine learning algorithms (especially those based on distance calculations) perform better with standardized data.
- **Faster Convergence:** Standardization can help optimization algorithms converge faster during model training.
- **Fair Comparison of Features:** When features have different scales, standardization ensures that they contribute equally to the model's learning process.

VIII. Normalization

- **Definition:** Normalization is a scaling technique that re-scales data into a specific range, typically between 0 and 1 or -1 and 1.
- **Methods:**
    - **Min-Max Scaling:** Scales data to a range of 0 to 1 using the formula: (x - min) / (max - min)
    - **Other Scaling Methods:** Techniques exist to scale data to different ranges, like -1 to 1.
- **Applications:**
    - **Deep Learning (Image Processing):** Pixel values in images are often normalized to the range of 0 to 1 before being fed into Convolutional Neural Networks (CNNs).
    - **Data Preprocessing:** Used to bring features with different scales to a common scale, preventing features with larger magnitudes from dominating the learning process.

Data Normalization: Scaling Exam Scores and Study Hours

Scenario:

You have a dataset of students' exam scores (out of 100) and the number of hours they studied. You want to analyze this data, but the features have different ranges:

- **Exam Scores:** Range from 0 to 100
- **Study Hours:** Range from 1 to 15 (approximately)

Normalization can be used to rescale both features to a common range (0 to 1 in this case), making them more comparable.

Python Implementation (using MinMaxScaler):

```python
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Sample data (exam scores and study hours)
data = {'Exam_Score': [65, 80, 92, 70, 55],
        'Study_Hours': [5, 8, 12, 6, 3]}
df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)

# Create a MinMaxScaler object
scaler = MinMaxScaler()

# Fit the scaler to the data and transform the features
df[['Exam_Score', 'Study_Hours']] = scaler.fit_transform(df[['Exam_Score', 'Study_Hours']])

print("\nNormalized DataFrame:")
print(df)
```

Explanation:

1. **Data Preparation:**
   - Create a Pandas DataFrame containing exam scores and study hours.
2. **MinMaxScaler:**
   - Import `MinMaxScaler` from `sklearn.preprocessing`.
   - Create a `MinMaxScaler` object. This object will be used to normalize the data to a range of 0 to 1.
3. **Fit and Transform:**
   - **`fit_transform()`:**
     - **`fit()`:** Calculates the minimum and maximum values of the features in the training set.

    - `transform()`: Applies the normalization transformation, scaling the features to the specified range (0 to 1 by default).
- We apply `fit_transform` to the 'Exam_Score' and 'Study_Hours' columns to normalize them.

Output:

The code will print the original DataFrame and the normalized DataFrame. In the normalized DataFrame:

- **Exam_Score and Study_Hours:** The values will be scaled to a range of 0 to 1, where 0 represents the minimum value in the original data, and 1 represents the maximum value.

Benefits of Normalization:

- **Improved Comparability:** When features have different ranges, normalization brings them to a common scale, making them more comparable.
- **Algorithm Compatibility:** Some machine learning algorithms, especially those sensitive to feature scaling (like K-Nearest Neighbors or those using gradient descent), benefit from normalized data.
- **Data Interpretation:** Normalizing data to a standard range can sometimes make it easier to interpret, especially when comparing features with different units.

IX. Practical Application: Comparing Performances Using Z-Scores

The lecture provides a practical example of using z-scores to compare the performance of a cricket team in different years, taking into account the average score and standard deviation of each series. While the example data might need adjustments for realism, it illustrates how z-scores can be used for relative performance comparisons in different contexts.

The Exam Score Problem: Understanding the Empirical Rule with Z-Scores

Problem:

Professor Statson just graded the final exam for his Introduction to Statistics course. He tells his students that the exam scores follow a normal distribution with a mean (μ) of 75 and a standard deviation (σ) of 8.

Answer the following questions using the empirical rule and z-scores:

1. **What percentage of students scored between 67 and 83 on the exam?**
2. **What percentage of students scored above 91 on the exam?**
3. **Sarah received her exam score and found out she scored better than 84% of her classmates. What was Sarah's approximate score on the exam?**

Solution:

1. Percentage of students scoring between 67 and 83:

- **Step 1: Calculate the z-scores for both scores.**
  - For 67: z = (67 - 75) / 8 = -1
  - For 83: z = (83 - 75) / 8 = +1
- **Step 2: Interpret the z-scores.**
  - A z-score of -1 means the score of 67 is one standard deviation below the mean.
  - A z-score of +1 means the score of 83 is one standard deviation above the mean.
- **Step 3: Apply the empirical rule.**
  - The empirical rule states that approximately **68%** of data falls within one standard deviation of the mean.

Therefore, approximately 68% of students scored between 67 and 83 on the exam.

2. Percentage of students scoring above 91:

- **Step 1: Calculate the z-score for 91.**
  - z = (91 - 75) / 8 = +2
- **Step 2: Interpret the z-score.**
  - A z-score of +2 means the score of 91 is two standard deviations above the mean.

- **Step 3: Apply the empirical rule.**
    - The empirical rule states that approximately 95% of data falls within two standard deviations of the mean. This means that 2.5% falls above two standard deviations and 2.5% falls below two standard deviations.

Therefore, approximately 2.5% of students scored above 91 on the exam.

3. Sarah's approximate score:

- **Step 1: Determine the z-score corresponding to Sarah's percentile.**
    - Since Sarah scored better than 84% of her classmates, she falls in the 84th percentile. We need to find the z-score that corresponds to the point where 84% of the data falls below it. Looking at a standard normal distribution table or using a calculator, we find that a z-score of approximately +1 corresponds to the 84th percentile.
- **Step 2: Use the z-score formula to solve for Sarah's score (x).**
    - We know $z = +1$, $\mu = 75$, and $\sigma = 8$.
    - $1 = (x - 75) / 8$
    - $8 = x - 75$
    - $x = 83$

Therefore, Sarah's approximate score on the exam was 83.

```python
import numpy as np
import pandas as pd
from scipy.stats import norm

# Set the random seed for reproducibility
np.random.seed(42)

# Generate exam scores for 100 students following a normal distribution
# Mean (μ) = 75, Standard Deviation (σ) = 8
exam_scores = np.random.normal(loc=75, scale=8, size=100)
exam_scores = np.round(exam_scores).astype(int)  # Round to nearest integer

# Create a Pandas DataFrame
df = pd.DataFrame({'Exam_Score': exam_scores})

print("Generated Dataset:")
print(df.head())

# --- Part 1: Percentage between 67 and 83 ---
# Calculate z-scores
df['z_score'] = (df['Exam_Score'] - df['Exam_Score'].mean()) / df['Exam_Score'].std()

# Filter for scores between z-scores of -1 and +1
students_within_one_std = df[(df['z_score'] >= -1) & (df['z_score'] <= 1)]

# Calculate the percentage
percentage_within_one_std = (len(students_within_one_std) / len(df)) * 100

print("\n--- Part 1 ---")
print(f"Percentage of students between scores 67 and 83: {percentage_within_one_std:.2f}%")
```

```python
# --- Part 2: Percentage above 91 ---
# Filter for scores with z-score above +2
students_above_two_std = df[df['z_score'] > 2]

# Calculate the percentage
percentage_above_two_std = (len(students_above_two_std) / len(df)) * 100

print("\n--- Part 2 ---")
print(f"Percentage of students with scores above 91: {percentage_above_two_std:.2f}%")

# --- Part 3: Sarah's Score ---
# Find the z-score for the 84th percentile
sarahs_z_score = norm.ppf(0.84)  # Use the inverse CDF (percent point function)

# Calculate Sarah's score
sarahs_score = (sarahs_z_score * df['Exam_Score'].std()) + df['Exam_Score'].mean()

print("\n--- Part 3 ---")
print(f"Sarah's approximate score: {sarahs_score:.2f}")
```

Explanation:

1. **Dataset Generation:**
    - We use `np.random.normal()` to generate 100 exam scores that follow a normal distribution with the specified mean (75) and standard deviation (8).
    - `np.round()` and `.astype(int)` are used to round the scores to the nearest integer.
    - A Pandas DataFrame is created to store and manage the data.
2. **Part 1: Percentage between 67 and 83:**
    - Z-scores are calculated for each exam score using the formula: `z = (x - mean) / std`.
    - The DataFrame is filtered to select students with z-scores between -1 and +1 (within one standard deviation of the mean).

- The percentage of students within this range is calculated.

3. **Part 2: Percentage above 91:**
   - The DataFrame is filtered to select students with z-scores greater than +2 (above two standard deviations from the mean).
   - The percentage of students above this threshold is calculated.
4. **Part 3: Sarah's Score:**
   - `scipy.stats.norm.ppf(0.84)` is used to find the z-score corresponding to the 84th percentile. The `ppf()` function is the inverse of the cumulative distribution function (CDF), also known as the percent point function.
   - Sarah's score is then calculated using the z-score formula, rearranged to solve for `x` (the exam score).

This code will output the generated dataset, the percentage of students within one standard deviation of the mean, the percentage of students above two standard deviations, and Sarah's approximate score.

Similarly R code implementation,

```r
# Set the random seed for reproducibility
set.seed(42)

# Generate exam scores for 100 students following a normal distribution
# Mean (μ) = 75, Standard Deviation (σ) = 8
exam_scores <- rnorm(n = 100, mean = 75, sd = 8)
exam_scores <- round(exam_scores) # Round to nearest integer

# Create a data frame
df <- data.frame(Exam_Score = exam_scores)

print("Generated Dataset:")
print(head(df))

# --- Part 1: Percentage between 67 and 83 ---
# Calculate z-scores
df$z_score <- (df$Exam_Score - mean(df$Exam_Score)) / sd(df$Exam_Score)

# Filter for scores between z-scores of -1 and +1
students_within_one_std <- df[df$z_score >= -1 & df$z_score <= 1, ]

# Calculate the percentage
percentage_within_one_std <- (nrow(students_within_one_std) / nrow(df)) * 100

cat("\n--- Part 1 ---\n")
cat(sprintf("Percentage of students between scores 67 and 83: %.2f%%\n", percentage_within_one_std))

# --- Part 2: Percentage above 91 ---
# Filter for scores with z-score above +2
students_above_two_std <- df[df$z_score > 2, ]
```

```r
# Calculate the percentage
percentage_above_two_std <- (nrow(students_above_two_std) / nrow(df)) * 100

cat("\n--- Part 2 ---\n")
cat(sprintf("Percentage of students with scores above 91: %.2f%%\n", percentage_above_two_std))

# --- Part 3: Sarah's Score ---
# Find the z-score for the 84th percentile
sarahs_z_score <- qnorm(0.84)  # Use the quantile function (inverse CDF)

# Calculate Sarah's score
sarahs_score <- (sarahs_z_score * sd(df$Exam_Score)) + mean(df$Exam_Score)

cat("\n--- Part 3 ---\n")
cat(sprintf("Sarah's approximate score: %.2f\n", sarahs_score))
```

Explanation:

1. **Dataset Generation:**
   - `rnorm()` generates 100 exam scores following a normal distribution with the given mean (75) and standard deviation (8).
   - `round()` rounds off the scores to the nearest integer.
   - A data frame `df` is created to hold the generated scores.
2. **Part 1: Percentage between 67 and 83:**
   - Z-scores are calculated for each exam score: `z = (x - mean) / sd`.
   - The data frame is subsetted to select students with z-scores between -1 and +1.
   - The percentage of students within this range is calculated.
3. **Part 2: Percentage above 91:**
   - The data frame is subsetted to select students with z-scores greater than +2.
   - The percentage of students above this threshold is calculated.
4. **Part 3: Sarah's Score:**

- `qnorm(0.84)` calculates the z-score corresponding to the 84th percentile using the quantile function (which is the inverse of the CDF).
- Sarah's score is then calculated based on the z-score, mean, and standard deviation.

This R code will print the generated dataset, the percentage of students within one standard deviation of the mean, the percentage of students above two standard deviations, and Sarah's approximate score.

Detecting Outliers with Z-Scores: A Conceptual Explanation

1. Understanding Outliers:

- Outliers are data points that are significantly different from other observations in a dataset. They can be unusually high or unusually low.
- Outliers can occur due to various reasons, such as measurement errors, data entry mistakes, or genuine anomalies in the data.

2. Z-Scores as a Measure of Deviation:

- A z-score tells us how many standard deviations a particular data point is away from the mean of the dataset.
- A positive z-score indicates the data point is above the mean, while a negative z-score indicates it's below the mean.
- The larger the absolute value of the z-score, the farther away the data point is from the mean, making it more likely to be an outlier.

3. Setting a Threshold for Outlier Detection:

- We need to define a threshold for the z-score to determine which data points are considered outliers.
- A common threshold is a z-score of +3 or -3. This means any data point that is three or more standard deviations away from the mean is flagged as a potential outlier.

4. Identifying Outliers:

- Calculate the z-score for each data point in your dataset.
- Compare the absolute value of each z-score to the chosen threshold.
- If the absolute value of the z-score is greater than the threshold, mark that data point as a potential outlier.

Example:

Imagine you have data on the age of students in a class. Most students are between 20-25 years old. However, you have one student who is 50 years old. This data point will likely have a very high z-score compared to the rest of the class, making it a clear outlier.

Important Considerations:

- The choice of the z-score threshold depends on the specific dataset and the context of the analysis.
- Outliers should be investigated further to understand the reason for their presence. They should not be automatically removed from the dataset unless there is a valid justification.

Outlier Detection in Website Traffic Data Using Z-Scores

Scenario:

You are analyzing website traffic data for a month. You have the number of daily visitors for the past 30 days. You want to identify any outlier days where the traffic was unusually high or low compared to the typical traffic pattern.

Dataset (Daily Visitors):

```
200, 220, 235, 215, 198, 205, 240, 230, 225, 210,
195, 218, 232, 228, 208, 170, 185, 202, 216, 224,
238, 245, 212, 209, 221, 236, 500, 227, 219, 233
```

Python Implementation:

```python
import numpy as np
import pandas as pd

# Sample website traffic data (daily visitors)
traffic_data = [200, 220, 235, 215, 198, 205, 240, 230, 225, 210,
                195, 218, 232, 228, 208, 170, 185, 202, 216, 224,
                238, 245, 212, 209, 221, 236, 500, 227, 219, 233]

df = pd.DataFrame({'Visitors': traffic_data})

# Calculate the z-score for each day's traffic
df['z_score'] = (df['Visitors'] - df['Visitors'].mean()) / df['Visitors'].std()

# Define a threshold for outlier detection (e.g., z-score > 3 or z-score < -3)
outlier_threshold = 3

# Identify outliers
df['Outlier'] = np.where(np.abs(df['z_score']) > outlier_threshold, True, False)

print(df)

# Print the outlier days
print("\nOutlier Days:")
print(df[df['Outlier'] == True])
```

Explanation:

1. **Data Preparation:**
    - Create a Pandas DataFrame to store the daily visitor data.
2. **Z-Score Calculation:**

- Calculate the z-score for each day's traffic using the formula: `z = (x - mean) / std`.

3. **Outlier Threshold:**
   - Set a threshold for outlier detection. A common threshold is a z-score greater than 3 or less than -3 (representing data points that are three standard deviations away from the mean). This threshold can be adjusted based on the specific dataset and the desired sensitivity to outliers.
4. **Outlier Identification:**
   - Create a new column in the DataFrame to flag outliers based on the chosen threshold.
5. **Output:**
   - Print the DataFrame with z-scores and outlier flags.
   - Print the rows representing the outlier days.

Output:

The code will print the DataFrame showing the daily visitors, their corresponding z-scores, and a boolean flag indicating whether a day is considered an outlier. The outlier days will be printed separately.

In this example, you'll likely find that the day with 500 visitors has a z-score significantly higher than the threshold, clearly marking it as an outlier. This indicates an unusual spike in traffic that day, which could be worth investigating further (e.g., a marketing campaign, a viral social media post, or a technical issue).

Similarly, R implementation is given below,

```r
# Sample website traffic data (daily visitors)
traffic_data <- c(200, 220, 235, 215, 198, 205, 240, 230, 225, 210,
                  195, 218, 232, 228, 208, 170, 185, 202, 216, 224,
                  238, 245, 212, 209, 221, 236, 500, 227, 219, 233)

df <- data.frame(Visitors = traffic_data)

# Calculate the z-score for each day's traffic
df$z_score <- (df$Visitors - mean(df$Visitors)) / sd(df$Visitors)

# Define a threshold for outlier detection (e.g., z-score > 3 or z-score < -3)
outlier_threshold <- 3

# Identify outliers
df$Outlier <- abs(df$z_score) > outlier_threshold

print(df)

# Print the outlier days
cat("\nOutlier Days:\n")
print(df[df$Outlier, ])
```

Explanation:

1. **Data Preparation:**
   - Create a data frame `df` to store the daily visitor data.
2. **Z-Score Calculation:**
   - Calculate the z-score for each day's traffic.
3. **Outlier Threshold:**
   - Set a threshold for outlier detection (z-score > 3 or z-score < -3 in this case).

4. **Outlier Identification:**
    - Create a new column `Outlier` in the data frame to flag outliers based on the threshold.
5. **Output:**
    - Print the data frame with z-scores and outlier flags.
    - Print the rows representing the outlier days.

This R code will output the data frame with daily visitors, their corresponding z-scores, and a boolean flag indicating outlier days. The outlier days will be printed separately, highlighting the day with unusually high traffic.

Lecture content created by: Sulaiman Ahmed.
Sharing the content without proper credit is prohibited.
email: sulaimanahmed013@gmail.com
LinkedIn: www.linkedin.com/in/sulaimanahmed
Website: www.sulaimanahmed013.wixsite.com/my-site