**Documentation:**

**Z-Score(Standardize Score) Analysis and Normal Distribution Computation**

**Introduction**

This document provides a detailed explanation of the Z-Score computation and normal distribution analysis from an input Excel file, along with an enhanced implementation using PostgreSQL for data storage and processing.

**Input Data Description**

**Input Data Format**

* The input data is stored in an Excel file (MainData.xlsx) with the following columns:
  + PayData: Employee salary information.
  + WeeklyEmailCharacterVolume: Weekly volume of email characters sent.
  + No\_of\_Positive\_emails\_or\_Complaint\_Emails: Count of positive or complaint emails.
* The file is expected to be formatted correctly with numerical data.

**Data Cleaning Requirements**

* Convert textual data to numeric format.
* Handle missing values by coercing them to NaN.
* Ensure valid column names are used for analysis.

**Processing Steps in Python**

**Loading Data from Excel**

* Read the Excel file into a Pandas DataFrame.
* Validate the data structure and confirm that all necessary columns exist.

**Data Processing**

* Convert the required columns to numeric format.
* Identify and handle missing values.
* Compute the mean and standard deviation for each column.
* Calculate Z-Scores using the formula

**Normal Distribution Computation**

* Use the cumulative density function (CDF) to determine the probability of data points falling within a normal distribution

**Output File Generation**

* Save the processed data into an Excel file (Final\_PayData\_with\_Stats.xlsx) with two sheets:
  + **Normal Distribution Settings**: Contains mean and standard deviation values for each column.
  + **Processed Data**: Includes original data along with computed Z-Scores and normal distribution probabilities.

**Code**:

"""

Z-Score Calculation and Normal Distribution Analysis Script

Author: [Your Name]

Date: [Current Date]

Description:

This script reads an employee dataset, processes pay and email-related metrics, and performs the following operations:

1. Converts necessary columns to numeric.

2. Computes mean and standard deviation for selected columns.

3. Calculates standardized scores (Z-scores) for each data point.

4. Computes cumulative probabilities using the normal distribution.

5. Saves results into an Excel file with separate sheets.

Inputs:

- A CSV file containing employee pay data and email volume metrics.

- Expected columns:

  - 'PayData': Employee salary information.

  - 'WeeklyEmailCharacterVolume': Weekly volume of email characters sent.

  - 'No\_of\_Positive\_emails\_or\_Complaint\_Emails': Count of positive or complaint emails.

Outputs:

- An Excel file ('Final\_PayData\_with\_Stats.xlsx') with:

  - 'Normal Distribution Settings' sheet containing mean and standard deviation.

  - 'Processed Data' sheet containing original data with calculated Z-scores and cumulative probabilities.

"""

import pandas as pd

from scipy.stats import zscore, norm

import os

import time

# Start execution timer to measure script performance

start\_time = time.time()

print("Script execution started...")

# Step 1: Load the dataset

file\_path = r"C:\Users\gundl\Downloads\MainData.csv"  # File that contains employee pay data

print("Loading dataset...")

try:

    df = pd.read\_csv(file\_path)  # Read the file into a pandas DataFrame

    print("Dataset loaded successfully. Shape:", df.shape)

except FileNotFoundError:

    print(f"Error: The file {file\_path} was not found.")  # Handle case where file is missing

    exit()

except pd.errors.EmptyDataError:

    print(f"Error: The file {file\_path} is empty.")  # Handle case where file is empty

    exit()

except pd.errors.ParserError:

    print(f"Error: The file {file\_path} could not be parsed.")  # Handle case where file has parsing errors

    exit()

# Step 2: Convert relevant columns to numeric format for calculations

columns\_to\_process = ["PayData", "WeeklyEmailCharacterVolume", "No\_of\_Positive\_emails\_or\_Complaint\_Emails"]

print("Converting columns to numeric format...")

for col in columns\_to\_process:

    df[col] = pd.to\_numeric(df[col], errors='coerce')  # Convert text values to numeric, coerce invalid values to NaN

    num\_missing = df[col].isna().sum()

    print(f"Column {col} - Converted to numeric. Missing values: {num\_missing}")

# Step 3: Compute mean and standard deviation for each numeric column

print("Calculating mean and standard deviation...")

mean\_std\_data = {

    "Metric": ["Mean", "St Dev"],  # Define labels for statistical metrics

}

for col in columns\_to\_process:

    mean\_std\_data[col] = [df[col].mean(), df[col].std()]  # Compute mean and standard deviation

    print(f"{col} -> Mean: {df[col].mean()}, Std Dev: {df[col].std()}")

# Store mean and standard deviation in a new DataFrame

normal\_distribution\_settings = pd.DataFrame(mean\_std\_data)

print("Mean and standard deviation calculations completed.")

# Step 4: Compute standardized scores (Z-scores) to measure deviation from mean

print("Calculating standardized scores (Z-scores)...")

for col in columns\_to\_process:

    std\_dev = df[col].std()

    if std\_dev != 0:

        df[f"{col}\_StandardisedScore"] = (df[col] - df[col].mean()) / std\_dev  # Compute Z-score

    else:

        df[f"{col}\_StandardisedScore"] = 0  # Assign zero if standard deviation is zero to avoid division error

    print(f"Computed Z-scores for {col}.")

# Step 5: Compute cumulative probability using normal distribution (CDF)

print("Calculating normal distribution cumulative probabilities...")

for col in columns\_to\_process:

    df[f"{col}\_Normal\_Dist"] = norm.cdf(df[col], df[col].mean(), df[col].std()) \* 100  # Convert to percentage

    print(f"Computed cumulative probability for {col}.")

# Step 6: Save results into an Excel file with separate sheets

output\_file\_path = "Final\_Data\_with\_Stats.xlsx"

print("Saving results to Excel file...")

# Ensure the output file has the correct extension

if not output\_file\_path.endswith(".xlsx"):

    output\_file\_path = "Final\_Data\_with\_Stats.xlsx"

# Check if the output file exists, remove it first to avoid appending issues

if os.path.exists(output\_file\_path):

    os.remove(output\_file\_path)

    print("Existing file removed to ensure a fresh save.")

# Write results to Excel with separate sheets for statistics and processed data

with pd.ExcelWriter(output\_file\_path, engine='xlsxwriter') as writer:

    normal\_distribution\_settings.to\_excel(writer, sheet\_name="Normal Distribution Settings", index=False)  # Save mean and std dev

    df.to\_excel(writer, sheet\_name="Processed Data", index=False)  # Save data with calculated metrics

# End execution timer and log total execution time

end\_time = time.time()

execution\_time = end\_time - start\_time

print(f"Processing complete. File saved as {output\_file\_path}")

print(f"Total execution time: {execution\_time:.2f} seconds")

**Enhancements Using PostgreSQL**

* The Python-based approach allows for quick data processing with Pandas and SciPy.
* PostgreSQL enhances scalability by allowing computations within the database, reducing data transfer time.
* For large datasets, PostgreSQL provides an efficient alternative by leveraging SQL-based analytics functions.

**Future Enhancements**

* Implement scheduled data processing using PostgreSQL triggers.
* Use PostgreSQL stored procedures for automatic data calculations.
* Develop a web-based dashboard to visualize the data insights in real time.