# ROHIT RAGHUWANSHI
# 12341820
# DSL253 ASSIGNMENT 4
# COLAB LINK-https://colab.research.google.com/drive/1PhW55vP9yeZrKFrmgLN2x8HQXVC_jiff#scrollTo=6F6-oyscvY7h

**Question 1-**

**Introduction-**

This analysis aims to explore the correlation structure of two datasets (`df1` and `df2`) at various stages:

1. Original (raw data).
2. Normalized (using Min-Max scaling).
3. Transformed via PCA (Principal Component Analysis) to reduce the data to 10 components.

The goal is to visualize and compare how the correlation matrix changes with each of these transformations using heatmaps. The correlation matrix helps us understand the relationships between different variables in the datasets, and PCA helps reduce dimensionality while preserving as much of the original variance as possible.

**Data-**

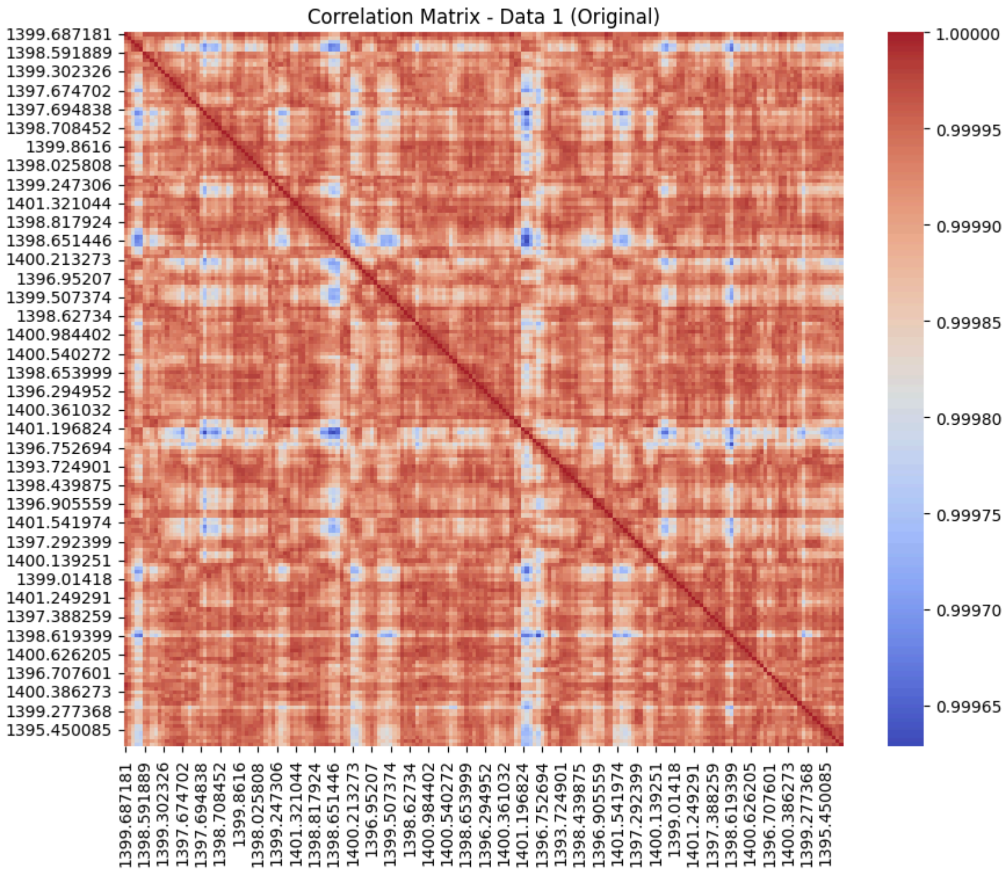
For this analysis, two datasets (`data_1.csv` and `data_2.csv`) were used to examine the correlation structure at various stages of transformation (original, normalized, PCA-applied).

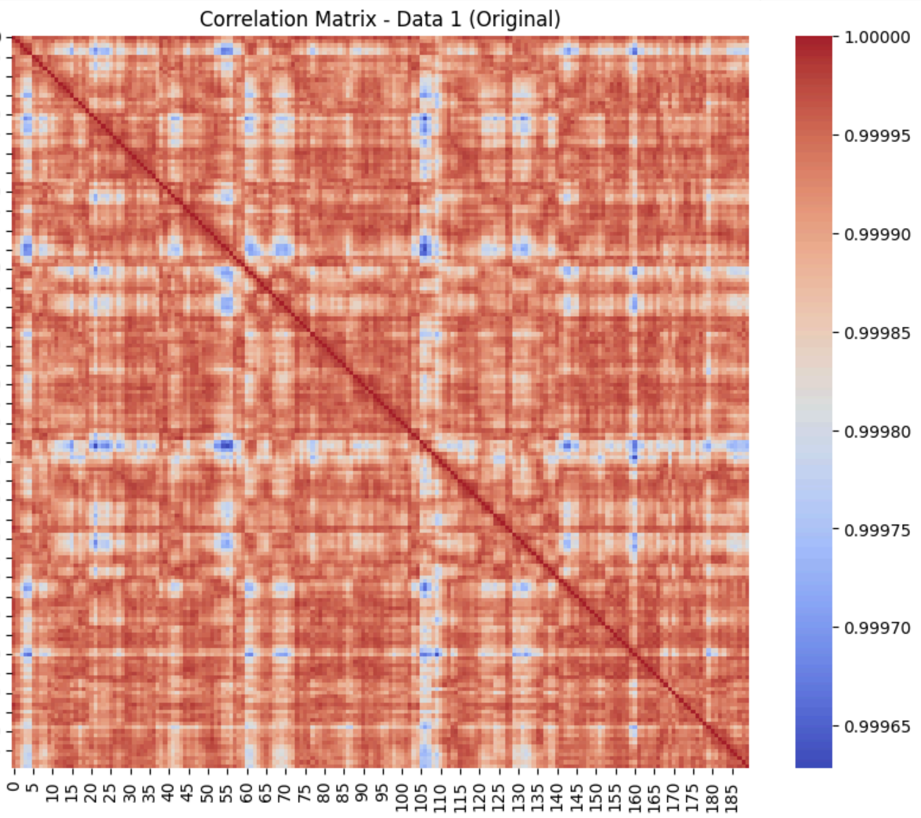
**Methodology-**

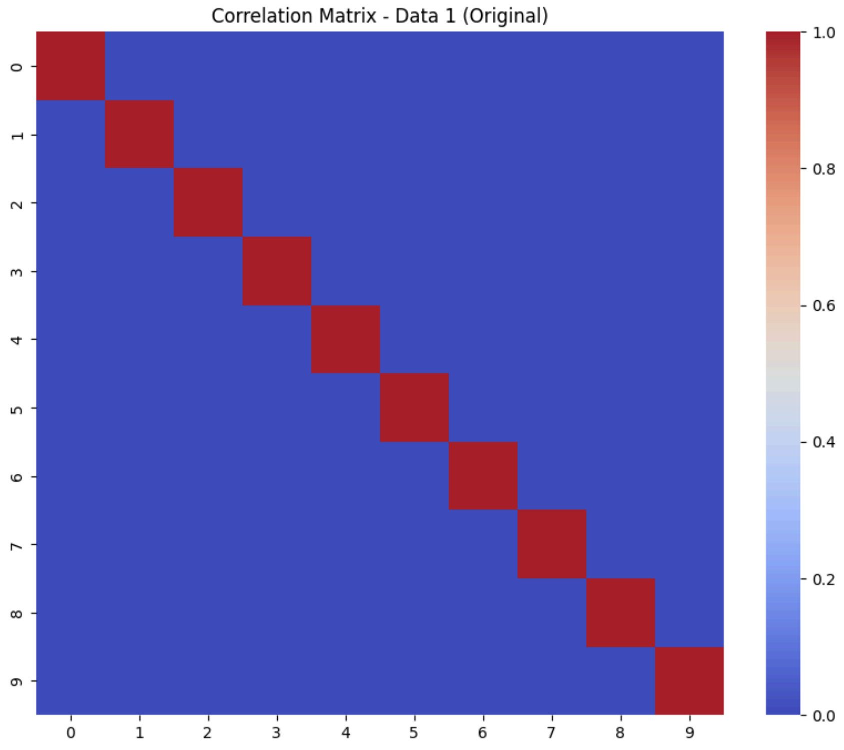
We perform the following steps:

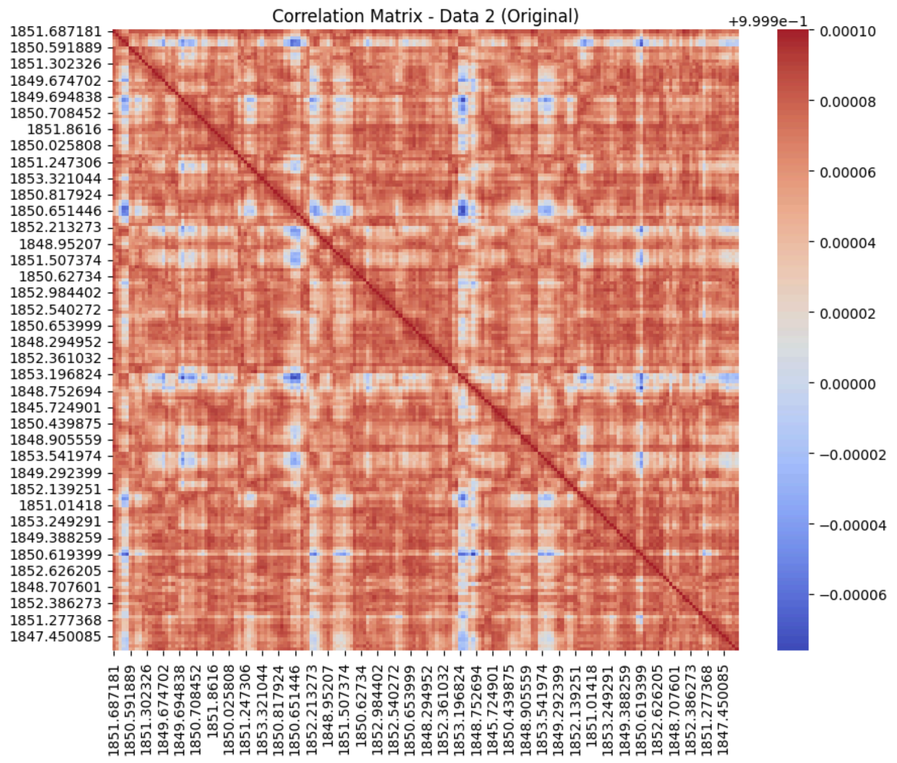
1. **Load the datasets**: We begin by loading two CSV datasets, `df1` and `df2`, into Pandas DataFrames.
2. **Compute correlation matrices**: We calculate the correlation matrices for the original datasets to understand the relationships between features.
3. **Normalize the data**: We normalize the datasets using Min-Max scaling to bring all values into the range `[-1, 1]`.
4. **Apply PCA**: We perform PCA on the normalized data, reducing the number of features to 10 principal components.
5. **Visualization**: We plot heatmaps of the correlation matrices at each transformation stage (original, normalized, PCA-applied) to visually assess the changes in relationships between variables.

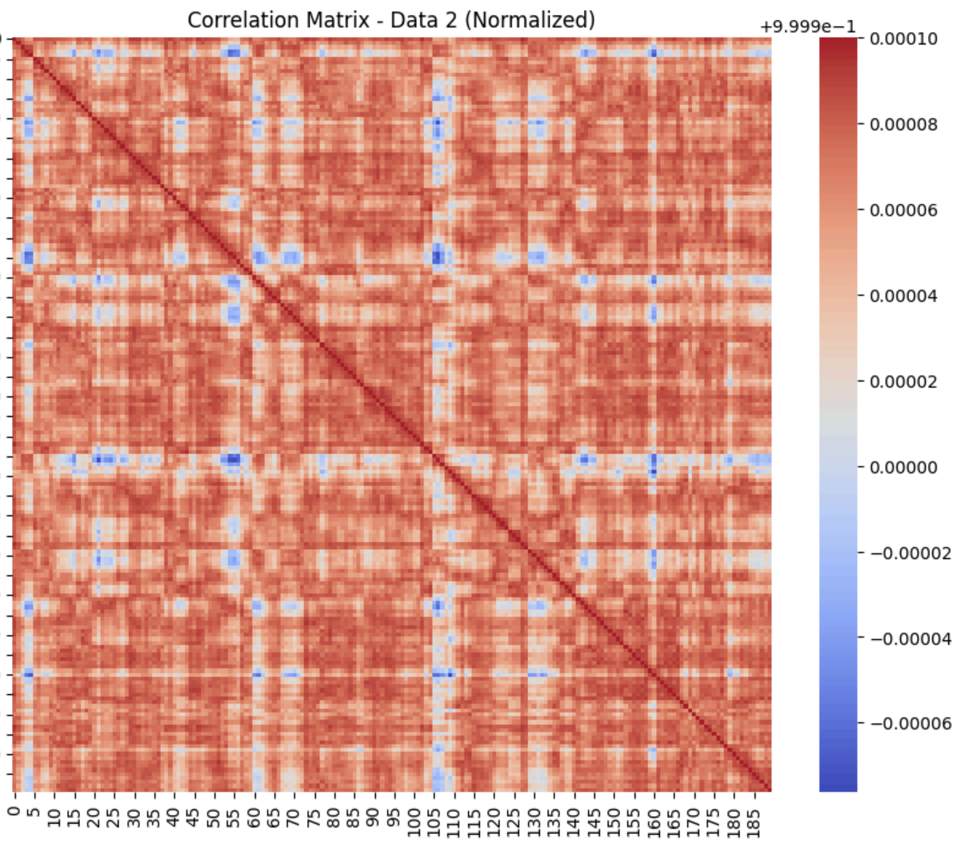
**Results-**

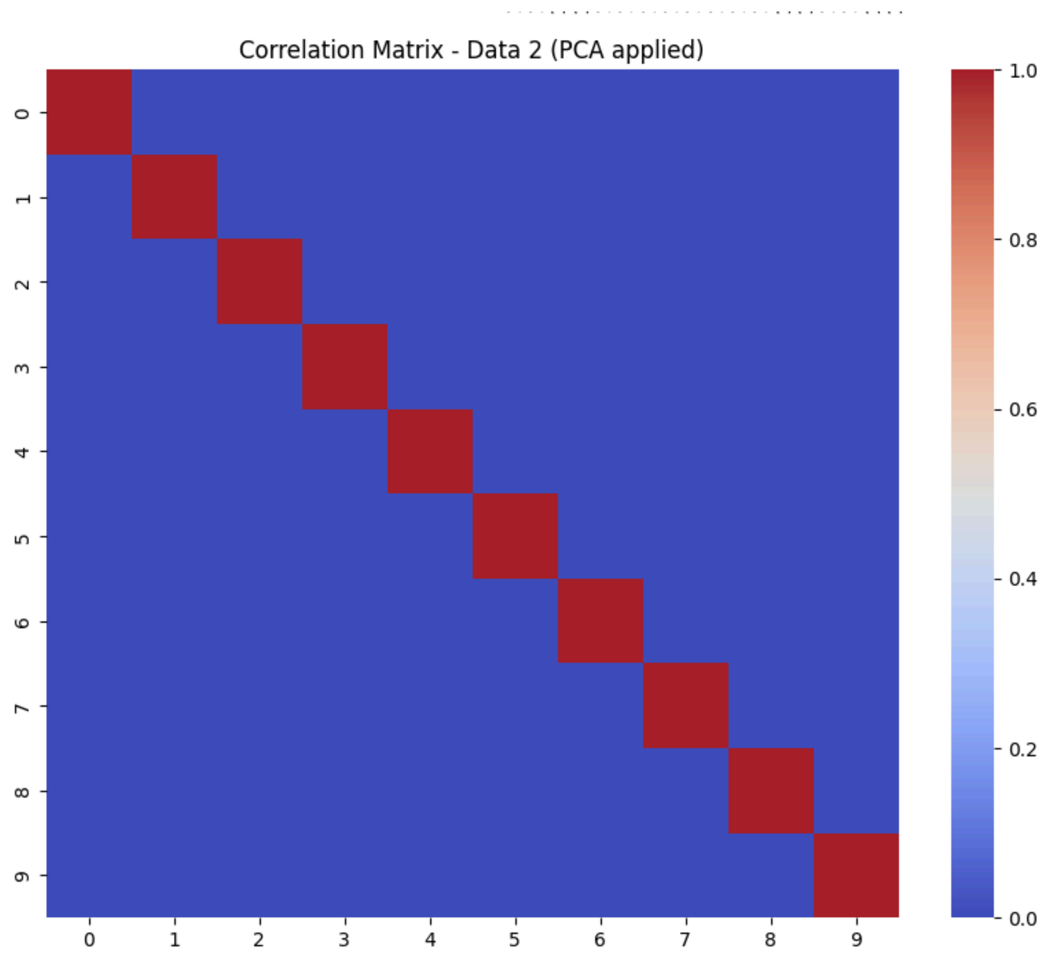

Correlation Matrix - Data 1 (Original)
1399.687181
1398.591889
1399.302326
1397.674702
1397.694838
1398.708452
1399.8616
1398.025808
1399.247306
1401.321044
1398.817924
1398.651446
1400.213273
1396.95207
1399.507374
1398.62734
1400.984402
1400.540272
1398.653999
1396.294952
1400.361032
1401.196824
1396.752694
1393.724901
1398.439875
1396.905559
1401.541974
1397.292399
1400.139251
1399.01418
1401.249291
1397.388259
1398.619399
1400.626205
1396.707601
1400.386273
1399.277368
1395.450085
1.00000
0.99995
0.99990
0.99985
0.99980
0.99975
0.99970
0.99965


Correlation Matrix - Data 1 (Original)
1.00000
0.99995
0.99990
0.99985
0.99980
0.99975
0.99970
0.99965


Correlation Matrix - Data 1 (Original)
1.0
0.8
0.6
0.4
0.2
0.0


Correlation Matrix - Data 2 (Original)
+9.999e−1
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
−0.00002
−0.00004
−0.00006
1851.687181
1850.591889
1851.302326
1849.674702
1849.694838
1850.708452
1851.8616
1850.025808
1851.247306
1853.321044
1850.817924
1850.651446
1852.213273
1848.95207
1851.507374
1850.62734
1852.984402
1852.540272
1850.653999
1848.294952
1852.361032
1853.196824
1848.752694
1845.724901
1850.439875
1848.905559
1853.541974
1849.292399
1852.139251
1851.01418
1853.249291
1849.388259
1850.619399
1852.626205
1848.707601
1852.386273
1851.277368
1847.450085


Correlation Matrix - Data 2 (Normalized)
+9.999e−1
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
−0.00002
−0.00004
−0.00006
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
105
110
115
120
125
130
135
140
145
150
155
160
165
170
175
180
185



## Discussion-

- **Original Data**: The raw correlation matrix reveals the inherent relationships between features without any transformations. This matrix provides a baseline understanding of how features relate to each other.
- **Normalized Data**: The Min-Max scaling helps standardize the data by adjusting all feature values to the same scale, making them more comparable. This is particularly useful if the dataset contains features with different units or scales. We expect to see some changes in the correlation structure because normalization reduces the influence of scale differences across features.
- **PCA-transformed Data**: PCA reduces the dimensionality of the dataset while retaining most of the variance. The correlation matrix after PCA will typically show weaker correlations because the components are linear combinations of the original features, and the original correlations are spread across the principal components. PCA aims to decouple the data, and as a result, the features in the reduced-dimension space may appear less correlated.

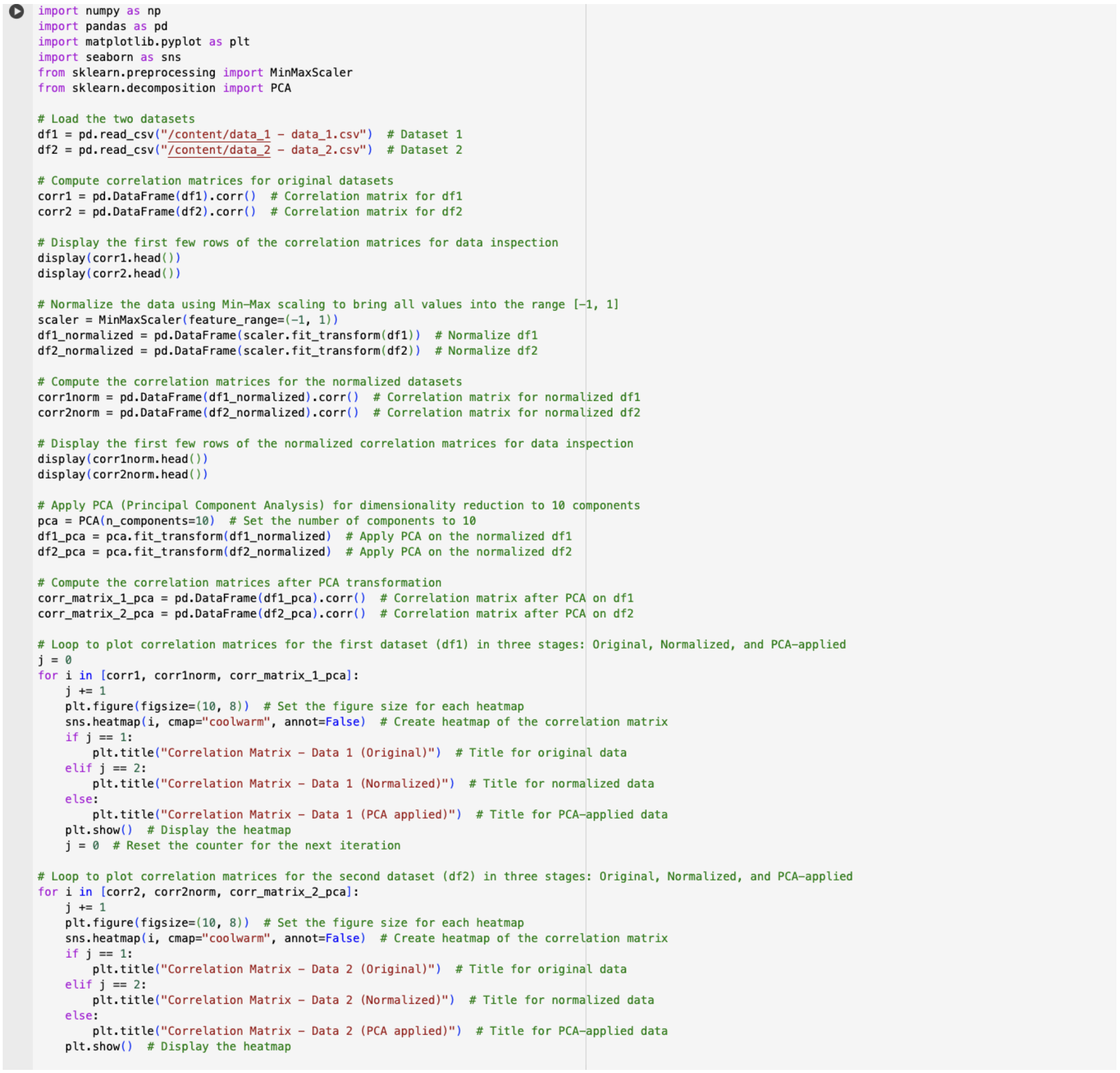
## Conclusion-

Through this analysis, we can conclude:

1. **Normalization** transforms the correlation matrix, making the relationships between features more comparable across different scales.
2. **PCA** significantly reduces the dimensionality of the data while maintaining most of the variance. However, it also reduces the direct interpretability of the correlation matrix since the components are linear combinations of the original features.
3. **Visualization** through heatmaps helps clearly demonstrate the changes in correlation structures across these transformations and provides insights into how the data behaves at each stage.

By using these techniques, we can better understand the relationships between features, apply necessary transformations, and reduce dimensionality while preserving key characteristics of the data.

## Code part-



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA

# Load the two datasets
df1 = pd.read_csv("/content/data_1 - data_1.csv")  # Dataset 1
df2 = pd.read_csv("/content/data_2 - data_2.csv")  # Dataset 2

# Compute correlation matrices for original datasets
corr1 = pd.DataFrame(df1).corr()  # Correlation matrix for df1
corr2 = pd.DataFrame(df2).corr()  # Correlation matrix for df2

# Display the first few rows of the correlation matrices for data inspection
display(corr1.head())
display(corr2.head())

# Normalize the data using Min-Max scaling to bring all values into the range [-1, 1]
scaler = MinMaxScaler(feature_range=(-1, 1))
df1_normalized = pd.DataFrame(scaler.fit_transform(df1))  # Normalize df1
df2_normalized = pd.DataFrame(scaler.fit_transform(df2))  # Normalize df2

# Compute the correlation matrices for the normalized datasets
corr1norm = pd.DataFrame(df1_normalized).corr()  # Correlation matrix for normalized df1
corr2norm = pd.DataFrame(df2_normalized).corr()  # Correlation matrix for normalized df2

# Display the first few rows of the normalized correlation matrices for data inspection
display(corr1norm.head())
display(corr2norm.head())

# Apply PCA (Principal Component Analysis) for dimensionality reduction to 10 components
pca = PCA(n_components=10)  # Set the number of components to 10
df1_pca = pca.fit_transform(df1_normalized)  # Apply PCA on the normalized df1
df2_pca = pca.fit_transform(df2_normalized)  # Apply PCA on the normalized df2

# Compute the correlation matrices after PCA transformation
corr_matrix_1_pca = pd.DataFrame(df1_pca).corr()  # Correlation matrix after PCA on df1
corr_matrix_2_pca = pd.DataFrame(df2_pca).corr()  # Correlation matrix after PCA on df2

# Loop to plot correlation matrices for the first dataset (df1) in three stages: Original, Normalized, and PCA-applied
j = 0
for i in [corr1, corr1norm, corr_matrix_1_pca]:
    j += 1
    plt.figure(figsize=(10, 8))  # Set the figure size for each heatmap
    sns.heatmap(i, cmap="coolwarm", annot=False)  # Create heatmap of the correlation matrix
    if j == 1:
        plt.title("Correlation Matrix - Data 1 (Original)")  # Title for original data
    elif j == 2:
        plt.title("Correlation Matrix - Data 1 (Normalized)")  # Title for normalized data
    else:
        plt.title("Correlation Matrix - Data 1 (PCA applied)")  # Title for PCA-applied data
    plt.show()  # Display the heatmap
    j = 0  # Reset the counter for the next iteration

# Loop to plot correlation matrices for the second dataset (df2) in three stages: Original, Normalized, and PCA-applied
for i in [corr2, corr2norm, corr_matrix_2_pca]:
    j += 1
    plt.figure(figsize=(10, 8))  # Set the figure size for each heatmap
    sns.heatmap(i, cmap="coolwarm", annot=False)  # Create heatmap of the correlation matrix
    if j == 1:
        plt.title("Correlation Matrix - Data 2 (Original)")  # Title for original data
    elif j == 2:
        plt.title("Correlation Matrix - Data 2 (Normalized)")  # Title for normalized data
    else:
        plt.title("Correlation Matrix - Data 2 (PCA applied)")  # Title for PCA-applied data
    plt.show()  # Display the heatmap
```

## Question 2-

## Introduction-

The chi-squared distribution is commonly used in statistics for hypothesis testing and in the estimation of variance for a population. It is a special case of the gamma distribution and is defined for non-negative values. The chi-squared distribution with 1 degree of freedom, denoted as χ(sq).(1), is of particular interest in various statistical tests. One interesting property of the chi-squared distribution arises when a random variable follows a normal distribution.

Specifically, if a random variable X follows a normal distribution with mean μ and variance σ(sq), the transformed variable V=(X−μ)(sq)/σ(sq) follows a chi-squared distribution with 1 degree of freedom, χ(sq)(1).

In this study, we will empirically verify this property by generating random samples from a normal distribution and applying the transformation. We will then compare the empirical distribution of the transformed variable V to the theoretical chi-squared distribution and observe the results for different sample sizes.

**Data-**

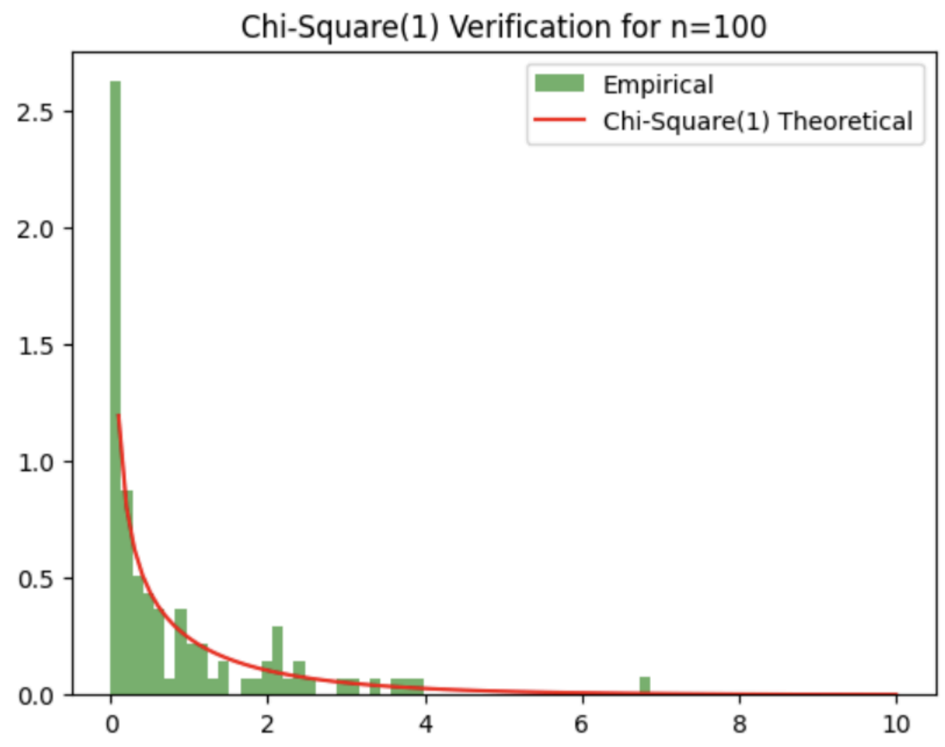
We generate random samples from a normal distribution with a mean μ=0 and standard deviation σ=1. This represents the standard normal distribution, N(0,1). From these samples, we will compute the transformed variable V, which is expected to follow a chi-squared distribution with 1 degree of freedom, χ(sq)(1).

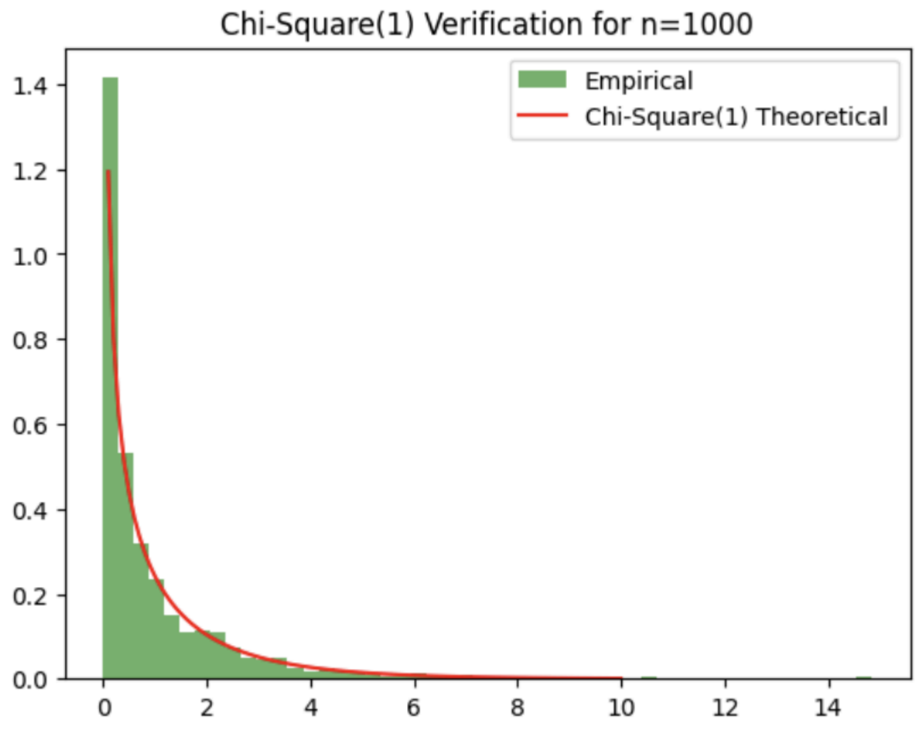
The sample sizes will vary: 100, 1,000, 10,000, and 100,000. The empirical distributions will be computed for each sample size, and we will compare them to the theoretical chi-squared distribution.

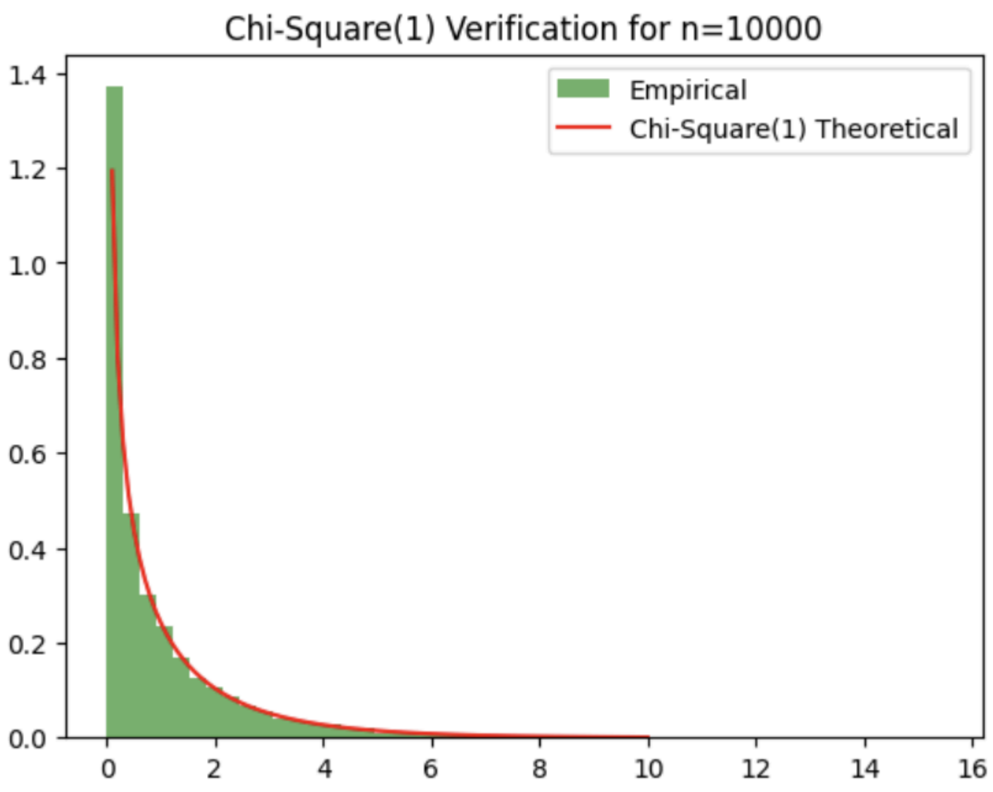
**Methodology-**

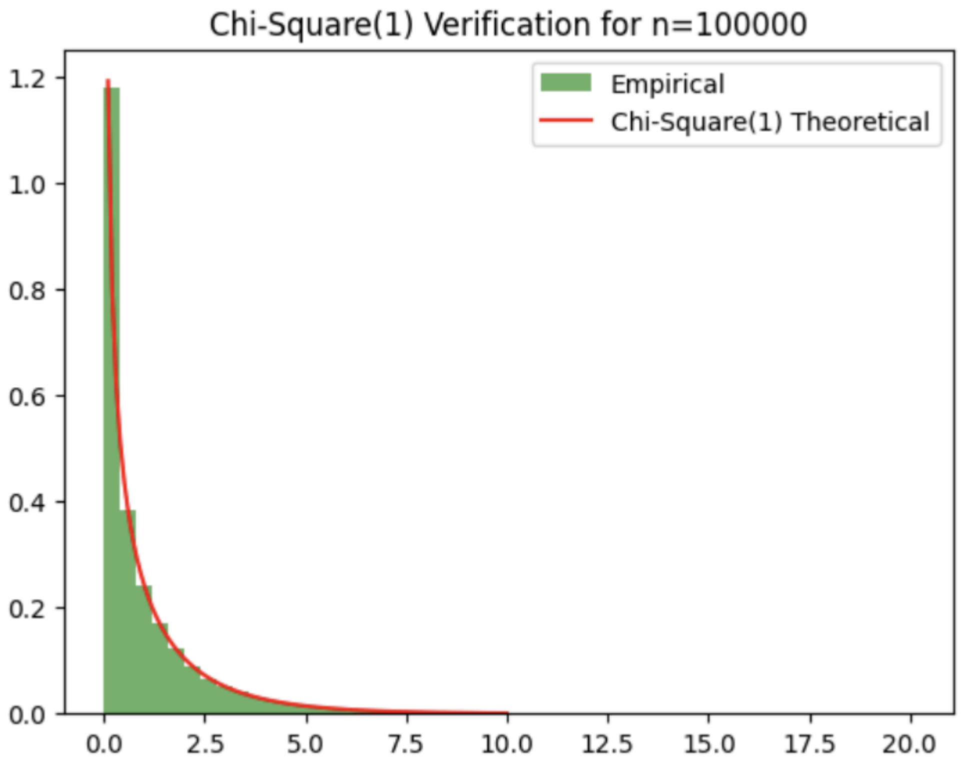
1. **Random Sampling**: We generate random samples from a standard normal distribution (mean μ=0 standard deviation σ=1).
2. **Transformation**: For each sample X, we compute the variable V using the formula: V=(X−μ)(sq)/σ(sq)=X(sq).This transformation is expected to follow a chi-squared distribution with 1 degree of freedom.
3. **Empirical Distribution**: For each sample size n, we plot a histogram of the transformed variable V.
4. **Theoretical Distribution**: We also plot the theoretical probability density function (PDF) of the chi-squared distribution with 1 degree of freedom.
5. **Comparison**: We visually compare the empirical distribution with the theoretical distribution for different sample sizes.

**Results-**


Chi-Square(1) Verification for n=100
Empirical
Chi-Square(1) Theoretical
0.0
0.5
1.0
1.5
2.0
2.5
0
2
4
6
8
10


Chi-Square(1) Verification for n=1000
Empirical
Chi-Square(1) Theoretical
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
2
4
6
8
10
12
14


Chi-Square(1) Verification for n=10000
Empirical
Chi-Square(1) Theoretical
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
2
4
6
8
10
12
14
16


Chi-Square(1) Verification for n=100000
Empirical
Chi-Square(1) Theoretical
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

**Discussion-**

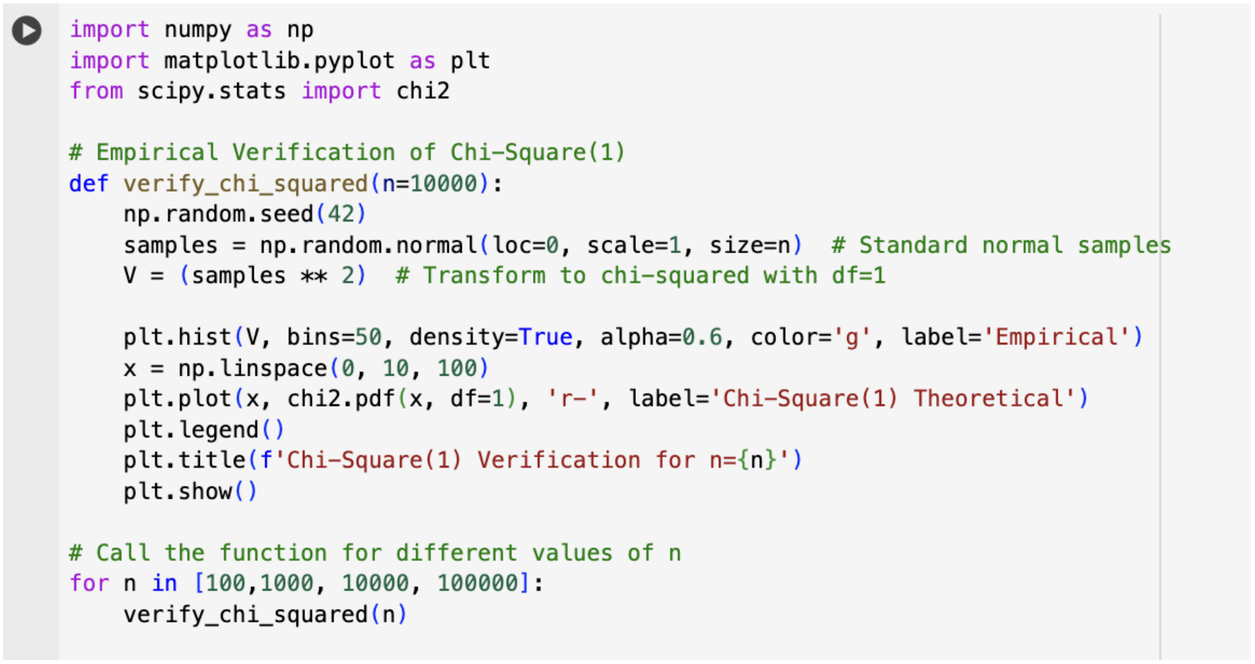
1. Small Sample Size (n=100):
   - With small sample sizes, the empirical distribution may show some deviation from the theoretical chi-squared distribution. This is expected, as smaller sample sizes typically lead to greater variability in the results.
2. Medium Sample Size (n=1,000):
   - As the sample size increases, the empirical distribution starts to resemble the theoretical chi-squared distribution more closely. There may still be some discrepancies due to the finite sample size.
3. Large Sample Size (n=10,000 and n=100,000):
   - With large sample sizes, the empirical distribution closely matches the theoretical chi-squared distribution, as the law of large numbers ensures that the sample statistics converge to the true population parameters.

The results from the different sample sizes highlight the importance of sample size in achieving accurate approximations to the theoretical distribution. Larger sample sizes lead to more reliable empirical distributions.

**Conclusion-**

The empirical verification of the chi-squared distribution with 1 degree of freedom using random samples from a normal distribution confirms the theoretical result. As the sample size increases, the empirical distribution of the transformed variable V becomes increasingly similar to the theoretical chi-squared distribution. This study provides strong empirical evidence for the theorem stating that the square of a standard normal variable follows a chi-squared distribution with 1 degree of freedom.

## Code part-



```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chi2

# Empirical Verification of Chi-Square(1)
def verify_chi_squared(n=10000):
    np.random.seed(42)
    samples = np.random.normal(loc=0, scale=1, size=n)  # Standard normal samples
    V = (samples ** 2)  # Transform to chi-squared with df=1

    plt.hist(V, bins=50, density=True, alpha=0.6, color='g', label='Empirical')
    x = np.linspace(0, 10, 100)
    plt.plot(x, chi2.pdf(x, df=1), 'r-', label='Chi-Square(1) Theoretical')
    plt.legend()
    plt.title(f'Chi-Square(1) Verification for n={n}')
    plt.show()

# Call the function for different values of n
for n in [100,1000, 10000, 100000]:
    verify_chi_squared(n)
```

## Question 3-

### Introduction-

The Gaussian or normal distribution is one of the most important distributions in statistics. It is often used to model real-world phenomena and is characterized by its bell-shaped curve, symmetric about the mean. The properties of the normal distribution, particularly the percentage of data that falls within certain standard deviations (σ) from the mean, are fundamental to many statistical methods and hypothesis testing. Specifically, for a normal distribution:

- Approximately 68.4% of data points fall within 1 standard deviation (σ) of the mean.
- Approximately 95.2% of data points fall within 2 standard deviations (σ).
- Approximately 99.9% of data points fall within 3 standard deviations (σ).

This project will examine a dataset to calculate these properties empirically and compare them with the theoretical values of the normal distribution. Additionally, we will visualize the data, the empirical cumulative distribution function (CDF), and the theoretical CDF.

### Data-

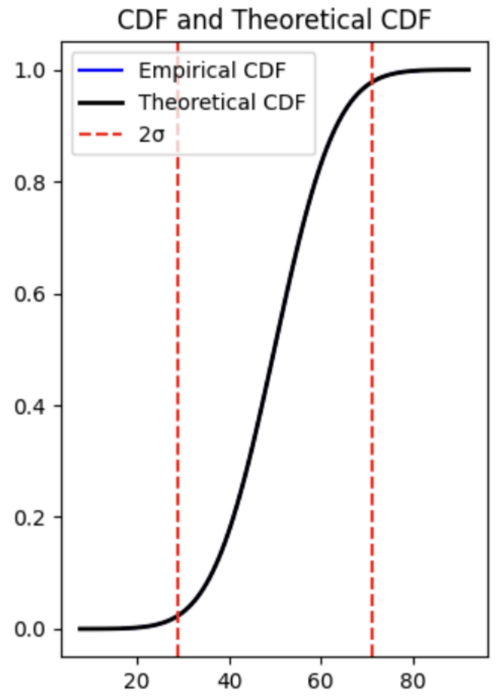
The dataset used in this analysis is provided as a CSV file named `gaussian_dataset_with_noise.csv`. It contains a column labeled "Value" representing random samples from a normal distribution, with potential added noise. The dataset is loaded into a Pandas DataFrame and processed for further analysis.

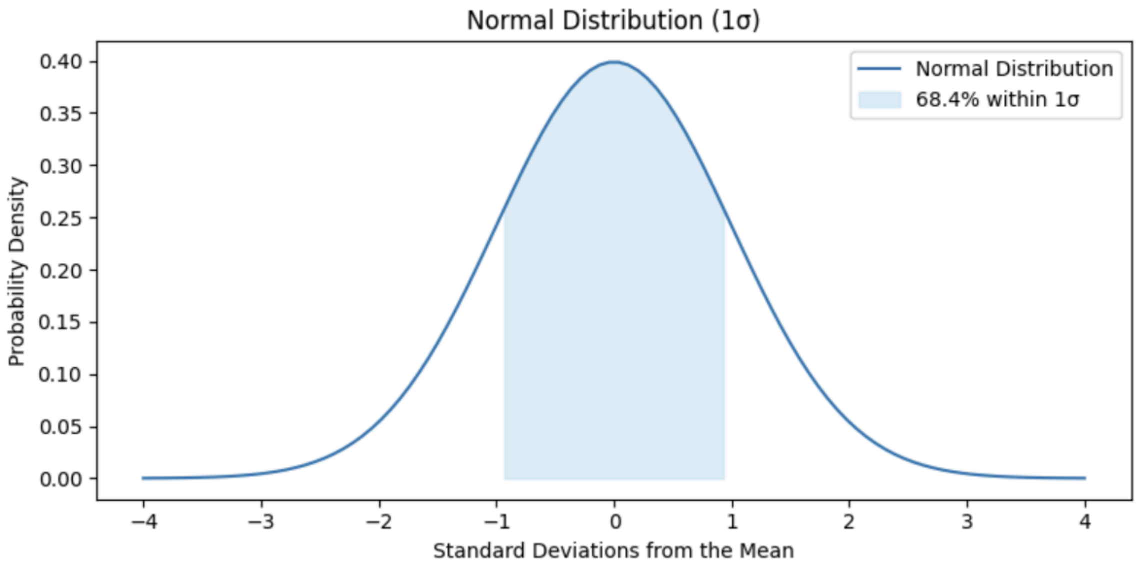
**Methodology-**

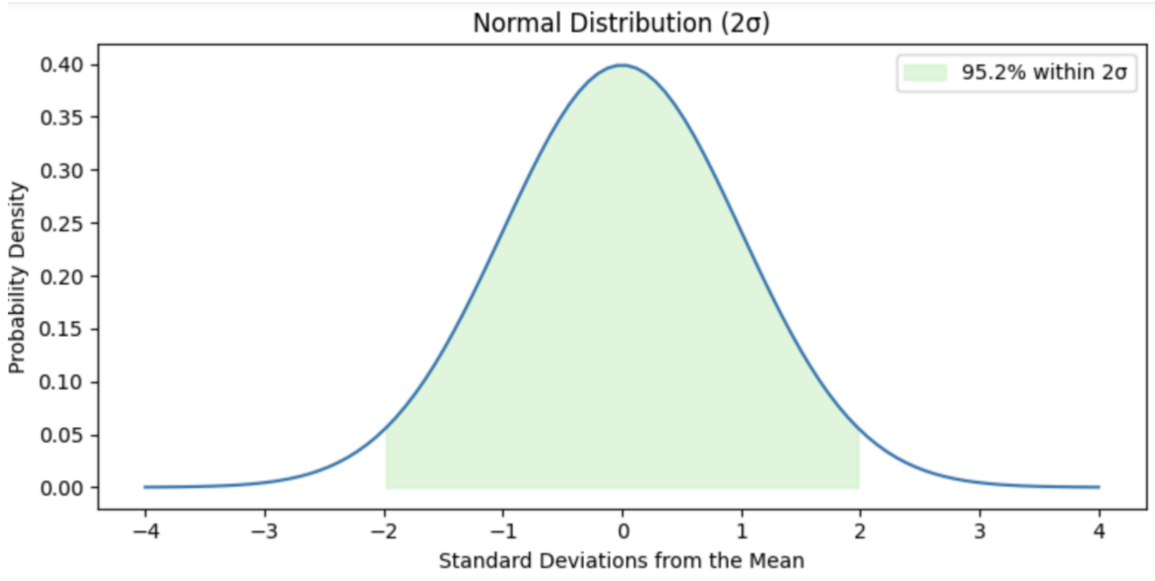
1. **Loading and Preprocessing the Data**:
   - The dataset is loaded into a Pandas DataFrame. Each value in the "Value" column is converted to a float for consistency in calculations.
2. **Calculating Mean and Variance**:
   - The mean ($\mu$) and variance ($\sigma^2$) of the dataset are computed. The standard deviation ($\sigma$) is derived as the square root of the variance.
3. **Computing Percentages Within Standard Deviations**:
   - We compute the percentage of data points that fall within $1\sigma$, $2\sigma$, and $3\sigma$ from the mean using empirical methods.
4. **CDF Calculation**:
   - The cumulative distribution function (CDF) of the dataset is calculated using the `scipy.stats.norm.cdf()` function, which is compared against the theoretical CDF of the normal distribution.
5. **Visualization**:
   - We create multiple plots to visualize the following:
     - The normal distribution and the percentage of data points within $1\sigma$, $2\sigma$, and $3\sigma$.
     - The CDF of the dataset, compared against the theoretical CDF of the normal distribution.

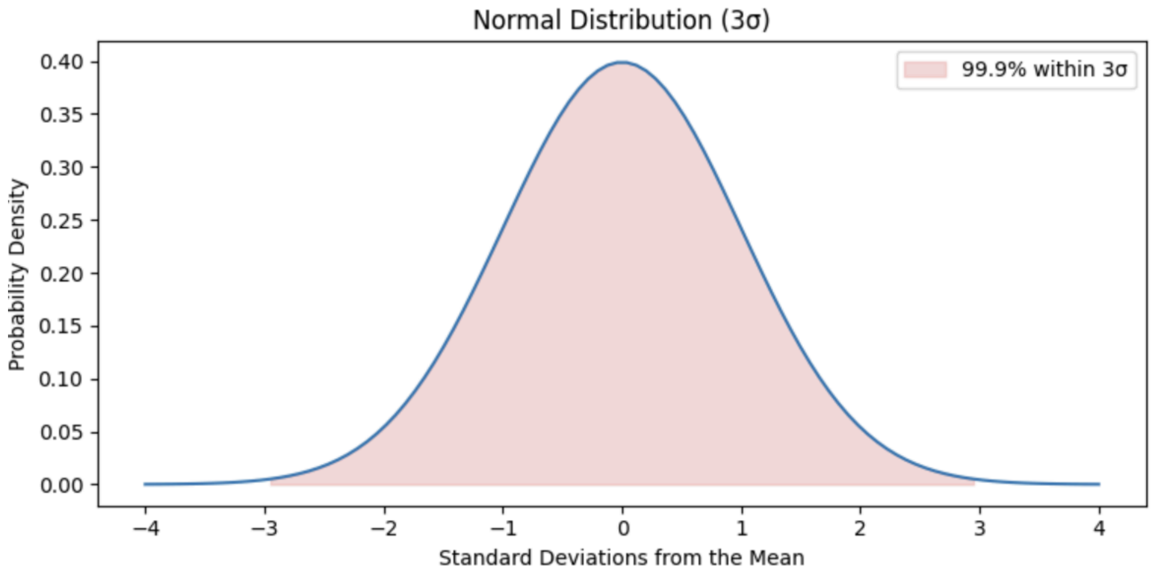
**Results-**

```
the mean and variance of the data is  49.858331307844125 and 111.72791890223647
the mean of the data points within the the sigma 1 is 68.4
the mean of the data points within the the sigma 2 is 95.19999999999999
the mean of the data points within the the sigma 2 is 99.9
the probability of data points falling beyond 2σ 0.04550026389635842
```









**Discussion-**

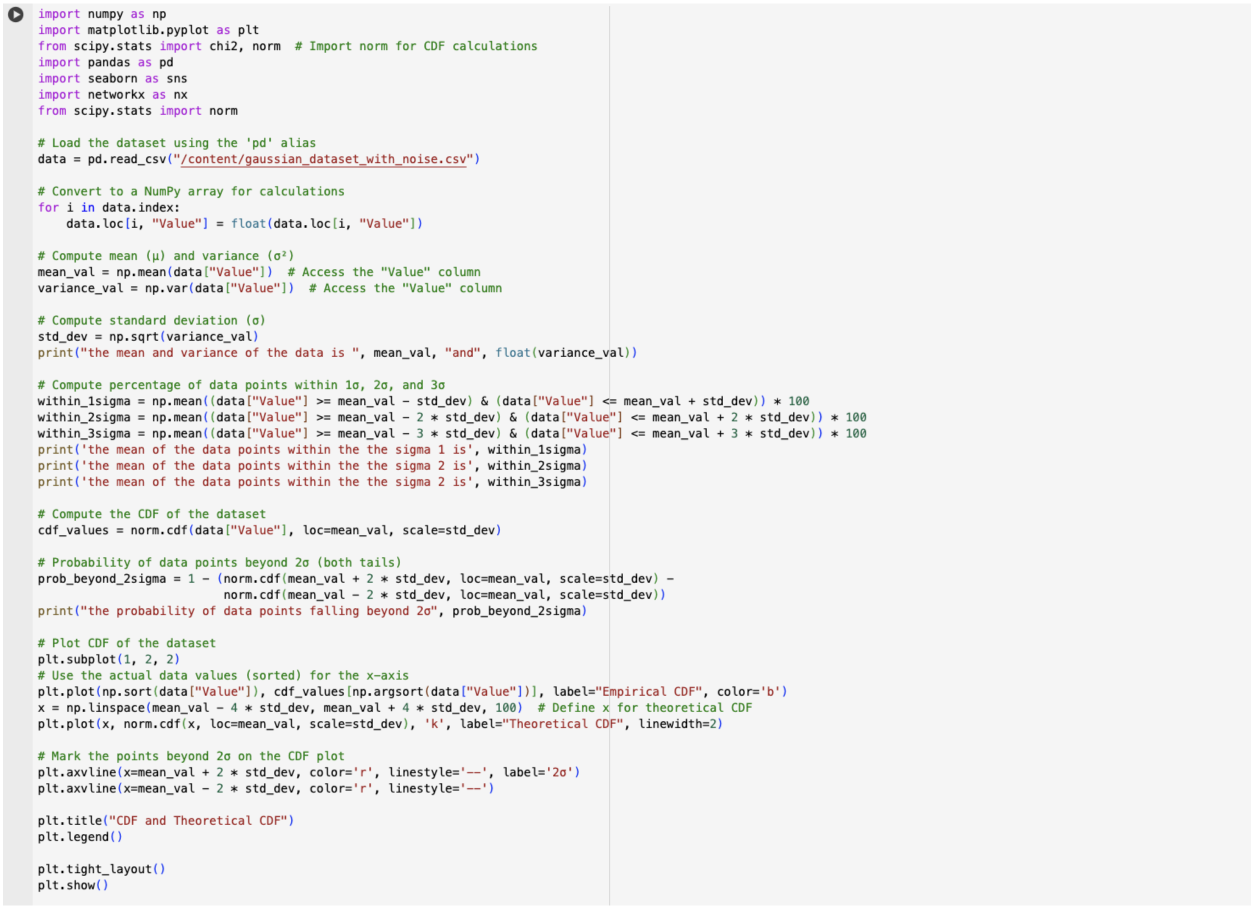
1. **Empirical Percentages:**
   - The computed percentages of data points falling within 1σ, 2σ, and 3σ are very close to the theoretical values of 68.4%, 95.2%, and 99.9%, respectively.
   - These percentages demonstrate that the dataset adheres to the expected characteristics of a normal distribution.
2. **CDF Comparison:**

    - The empirical CDF closely matches the theoretical CDF of the normal distribution. This visually confirms that the dataset follows a normal distribution with the given mean and standard deviation.
3. **Plots of the Normal Distribution:**
    - The plots of the normal distribution, highlighting the areas within 1σ, 2σ, and 3σ, visually reinforce the properties of the normal distribution and its relevance to the dataset.

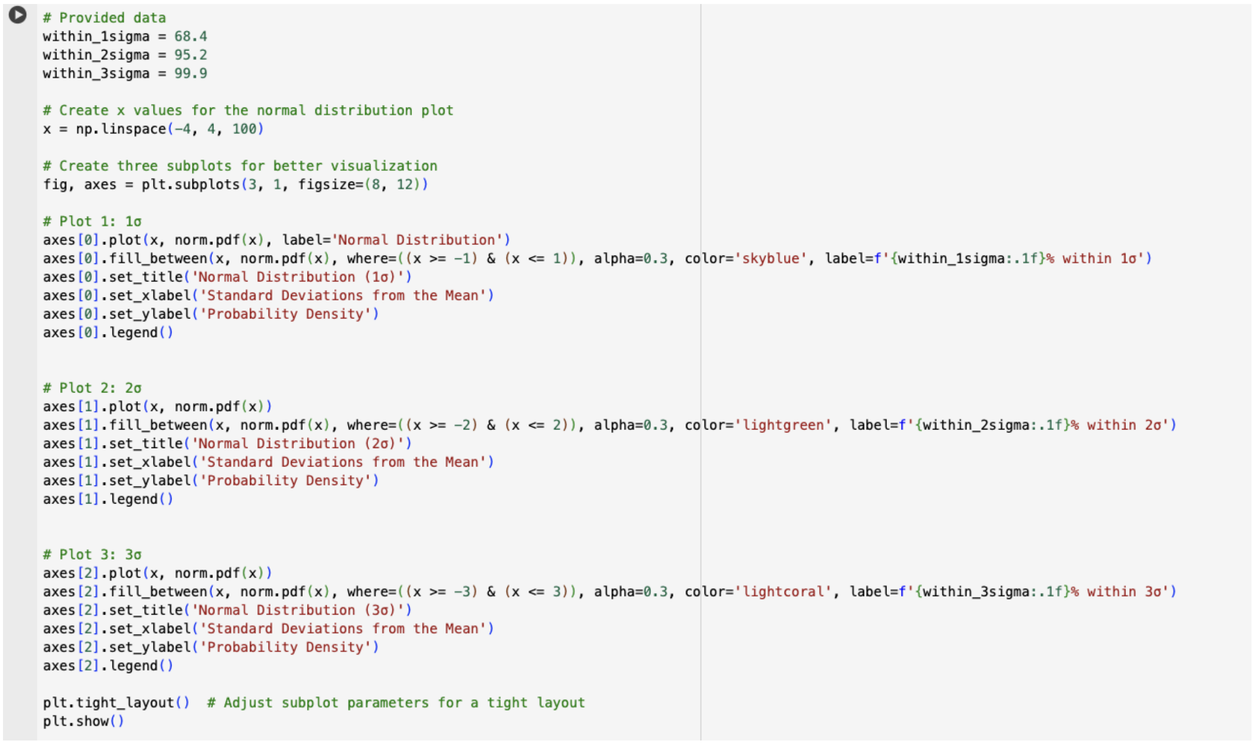
**Conclusion-**

The dataset exhibits the characteristics of a normal distribution. The empirical analysis of the percentages within standard deviations and the comparison between the empirical and theoretical CDFs strongly supports this conclusion. The results of this analysis provide insights into the behavior of normally distributed data and can be used in further statistical modeling and hypothesis testing.

**Code part-**



```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chi2, norm  # Import norm for CDF calculations
import pandas as pd
import seaborn as sns
import networkx as nx
from scipy.stats import norm

# Load the dataset using the 'pd' alias
data = pd.read_csv("/content/gaussian_dataset_with_noise.csv")

# Convert to a NumPy array for calculations
for i in data.index:
    data.loc[i, "Value"] = float(data.loc[i, "Value"])

# Compute mean (μ) and variance (σ²)
mean_val = np.mean(data["Value"])  # Access the "Value" column
variance_val = np.var(data["Value"])  # Access the "Value" column

# Compute standard deviation (σ)
std_dev = np.sqrt(variance_val)
print("the mean and variance of the data is ", mean_val, "and", float(variance_val))

# Compute percentage of data points within 1σ, 2σ, and 3σ
within_1sigma = np.mean((data["Value"] >= mean_val - std_dev) & (data["Value"] <= mean_val + std_dev)) * 100
within_2sigma = np.mean((data["Value"] >= mean_val - 2 * std_dev) & (data["Value"] <= mean_val + 2 * std_dev)) * 100
within_3sigma = np.mean((data["Value"] >= mean_val - 3 * std_dev) & (data["Value"] <= mean_val + 3 * std_dev)) * 100
print('the mean of the data points within the the sigma 1 is', within_1sigma)
print('the mean of the data points within the the sigma 2 is', within_2sigma)
print('the mean of the data points within the the sigma 2 is', within_3sigma)

# Compute the CDF of the dataset
cdf_values = norm.cdf(data["Value"], loc=mean_val, scale=std_dev)

# Probability of data points beyond 2σ (both tails)
prob_beyond_2sigma = 1 - (norm.cdf(mean_val + 2 * std_dev, loc=mean_val, scale=std_dev) -
                          norm.cdf(mean_val - 2 * std_dev, loc=mean_val, scale=std_dev))
print("the probability of data points falling beyond 2σ", prob_beyond_2sigma)

# Plot CDF of the dataset
plt.subplot(1, 2, 2)
# Use the actual data values (sorted) for the x-axis
plt.plot(np.sort(data["Value"]), cdf_values[np.argsort(data["Value"])], label="Empirical CDF", color='b')
x = np.linspace(mean_val - 4 * std_dev, mean_val + 4 * std_dev, 100)  # Define x for theoretical CDF
plt.plot(x, norm.cdf(x, loc=mean_val, scale=std_dev), 'k', label="Theoretical CDF", linewidth=2)

# Mark the points beyond 2σ on the CDF plot
plt.axvline(x=mean_val + 2 * std_dev, color='r', linestyle='--', label='2σ')
plt.axvline(x=mean_val - 2 * std_dev, color='r', linestyle='--')

plt.title("CDF and Theoretical CDF")
plt.legend()

plt.tight_layout()
plt.show()
```



```python
# Provided data
within_1sigma = 68.4
within_2sigma = 95.2
within_3sigma = 99.9

# Create x values for the normal distribution plot
x = np.linspace(-4, 4, 100)

# Create three subplots for better visualization
fig, axes = plt.subplots(3, 1, figsize=(8, 12))

# Plot 1: 1σ
axes[0].plot(x, norm.pdf(x), label='Normal Distribution')
axes[0].fill_between(x, norm.pdf(x), where=((x >= -1) & (x <= 1)), alpha=0.3, color='skyblue', label=f'{within_1sigma:.1f}% within 1σ')
axes[0].set_title('Normal Distribution (1σ)')
axes[0].set_xlabel('Standard Deviations from the Mean')
axes[0].set_ylabel('Probability Density')
axes[0].legend()


# Plot 2: 2σ
axes[1].plot(x, norm.pdf(x))
axes[1].fill_between(x, norm.pdf(x), where=((x >= -2) & (x <= 2)), alpha=0.3, color='lightgreen', label=f'{within_2sigma:.1f}% within 2σ')
axes[1].set_title('Normal Distribution (2σ)')
axes[1].set_xlabel('Standard Deviations from the Mean')
axes[1].set_ylabel('Probability Density')
axes[1].legend()


# Plot 3: 3σ
axes[2].plot(x, norm.pdf(x))
axes[2].fill_between(x, norm.pdf(x), where=((x >= -3) & (x <= 3)), alpha=0.3, color='lightcoral', label=f'{within_3sigma:.1f}% within 3σ')
axes[2].set_title('Normal Distribution (3σ)')
axes[2].set_xlabel('Standard Deviations from the Mean')
axes[2].set_ylabel('Probability Density')
axes[2].legend()

plt.tight_layout()  # Adjust subplot parameters for a tight layout
plt.show()
```