**Indian Institute of Information Technology**

**Dharwad**

# Case Study Analysis REPORT

**Team Members:**

Ashith Shetty(21bcs019)

Jaishana Bindu Priya(21bcs045)

Zaid Ragib(21bcs069)

**Submitted To:**

Dr. Ramesh Athe Sir

# Statistical Perspectives on Bone Health:

(Analyzing the Dynamics of Bone Mineral Index)

## Objective:-

Certainly, we will enhance our statistical analysis by employing various measures and techniques, including mean, median, mode, measures of dispersion, and more. Through these statistical tools, we intend to gain a comprehensive understanding of the dataset's distribution, central tendencies, and variability.

This enriched analysis will complement our regression and classification objectives, enabling us to extract deeper insights into the relationships between individual attributes and bone health indicators.

## Motive:

Our motive is to enhance our understanding of bone health by conducting a detailed statistical analysis. This analysis will not only help us grasp the dataset's characteristics but also facilitate the development of predictive models and classification systems to improve early detection and prevention of bone-related issues, ultimately enhancing overall well-being.

## Background:

Bone health is integral to overall well-being, affecting one's quality of life and susceptibility to conditions like osteoporosis and fractures. The Bone Mineral Index (BMI) is a vital parameter for assessing bone health, reflecting bone mineral density and strength. Our case study centers on a comprehensive statistical analysis of BMI data.

By employing statistical tools such as mean, median, mode, and measures of dispersion, we aim to gain deep insights into the dataset's distribution, central tendencies, and variability. This analysis will not only establish a solid foundation but also bolster our regression and classification objectives. Ultimately, our research seeks to uncover the intricate relationships between various attributes and bone health indicators. This understanding will enable the development of predictive models and classification systems, aiding in the early detection and prevention of bone-related issues, thereby enhancing the overall health and well-being of the studied population.

The results of studies exploring the association between serum uric acid (SUA) and bone mineral density (BMD) have been controversial and inconsistent. We thus sought to explore whether SUA levels were independently associated with BMD in patients with osteoporosis (OP).

## Data Set:

Below are the columns used in our case study:

**Gender:** The gender of the individuals in the study, categorized as male or female.

**Age:** The age of the participants in years, representing the age range of the subjects under investigation.

**Height**: The height of the individuals in centimeters, which can impact bone health.

**Weight**: The weight of the participants in kilograms, another important factor for assessing bone health.

**BMI (Body Mass Index):** A calculated value based on an individual's weight and height, used to assess whether they are underweight, normal weight, overweight, or obese.

**L1-4:** Measurement of bone density in the L1 to L4 vertebrae, often used in bone health assessments.

**L1.4T:** Bone density measurement in the L1 to L4 vertebrae, possibly with a specific measurement technique or treatment.

**FN (Femoral Neck):** Measurement of bone density at the femoral neck, a common site for bone health evaluations.

**FNT (Femoral Neck T-Score):** T-score value indicating the bone density at the femoral neck, typically used for osteoporosis diagnosis.

**TL (Total Lumbar):** Total lumbar bone density measurement, which can be relevant for overall bone health assessment.

**TLT (Total Lumbar T-Score):** T-score value indicating the total lumbar bone density, often used for osteoporosis diagnosis.

**ALT (Alanine Aminotransferase):** Liver enzyme levels that may provide insights into the liver's health.

**AST (Aspartate Aminotransferase):** Another liver enzyme whose levels can indicate liver health.

**BUN (Blood Urea Nitrogen):** A blood test measuring the level of urea nitrogen in the blood, which can reflect kidney function.

**CREA (Creatinine):** Measurement of creatinine in the blood, used to assess kidney function.

**URIC (Uric Acid):** Measurement of uric acid levels in the blood, which can be relevant for assessing conditions like gout.

**FBG (Fasting Blood Glucose):** Measurement of fasting blood glucose levels, an important marker for diabetes and metabolic health.

**HDL-C (High-Density Lipoprotein Cholesterol):** Measurement of "good" cholesterol levels in the blood, important for heart health.

**LDL-C (Low-Density Lipoprotein Cholesterol)**: Measurement of "bad" cholesterol levels in the blood, a key risk factor for heart disease.

**Ca (Calcium):** Measurement of calcium levels in the blood, essential for bone health and various bodily functions.

**P (Phosphorus):** Measurement of phosphorus levels in the blood, important for bone and overall health.

**Mg (Magnesium):** Measurement of magnesium levels in the blood, which plays a role in bone and muscle health.

Calcium: Possibly a duplicate column for measuring calcium levels, requiring clarification.

**Calcitriol**: Measurement of calcitriol, an active form of vitamin D, which is vital for calcium absorption and bone health.

**Bisphosphonate**: An indicator of whether individuals are taking bisphosphonate medications, often prescribed for osteoporosis.

**Calcitonin**: An indicator of whether individuals are taking calcitonin medications, used to treat certain bone conditions.

**HTN (Hypertension):** Presence or absence of hypertension (high blood pressure) in the participants.

**COPD (Chronic Obstructive Pulmonary Disease):** Presence or absence of COPD, a lung disease, in the individuals.

**DM (Diabetes Mellitus):** Presence or absence of diabetes mellitus, a metabolic disorder, in the study population.

**Hyperlipidaemia**: Presence or absence of hyperlipidemia, elevated blood lipid levels, in the participants.

**Hyperuricemia**: Presence or absence of hyperuricemia, elevated uric acid levels in the blood, in the study group.

**AS (Ankylosing Spondylitis):** Presence or absence of ankylosing spondylitis, a type of arthritis, in the individuals.

**VT (Vitamin Therapy**): An indicator of whether individuals are receiving vitamin therapy.

**VD (Vitamin D):** An indicator of vitamin D levels, important for bone health.

**OP (Osteoporosis):** Presence or absence of osteoporosis, a condition characterized by weakened bones, in the participants.

**CAD (Coronary Artery Disease):** Presence or absence of coronary artery disease, a heart condition, in the study population.

**CKD (Chronic Kidney Disease**): Presence or absence of chronic kidney disease in the individuals, which can impact bone health.

**Fracture**: An indicator of whether participants have a history of bone fractures.

**Smoking**: An indicator of whether individuals are current or former smokers.

**Drinking**: An indicator of alcohol consumption habits in the study group.

These columns represent a comprehensive set of variables that can be analyzed to gain valuable insights into bone health and its relationships with various health indicators and lifestyle factors.

## Citation Metadata:

**We used this data from Harvard DetaVerse. Below is the information of this data.**

**Persistent Identifier:** doi:10.7910/DVN/UDZIJS

**Publication Date:** December 17, 2022

**Title**: Bone Mineral Density

**Author**: Linfeng He

**Description**: This dataset constitutes the original data used in the research paper titled "Nonlinear Association between Serum Uric Acid levels and risk of Osteoporosis: A Retrospective Study," published on

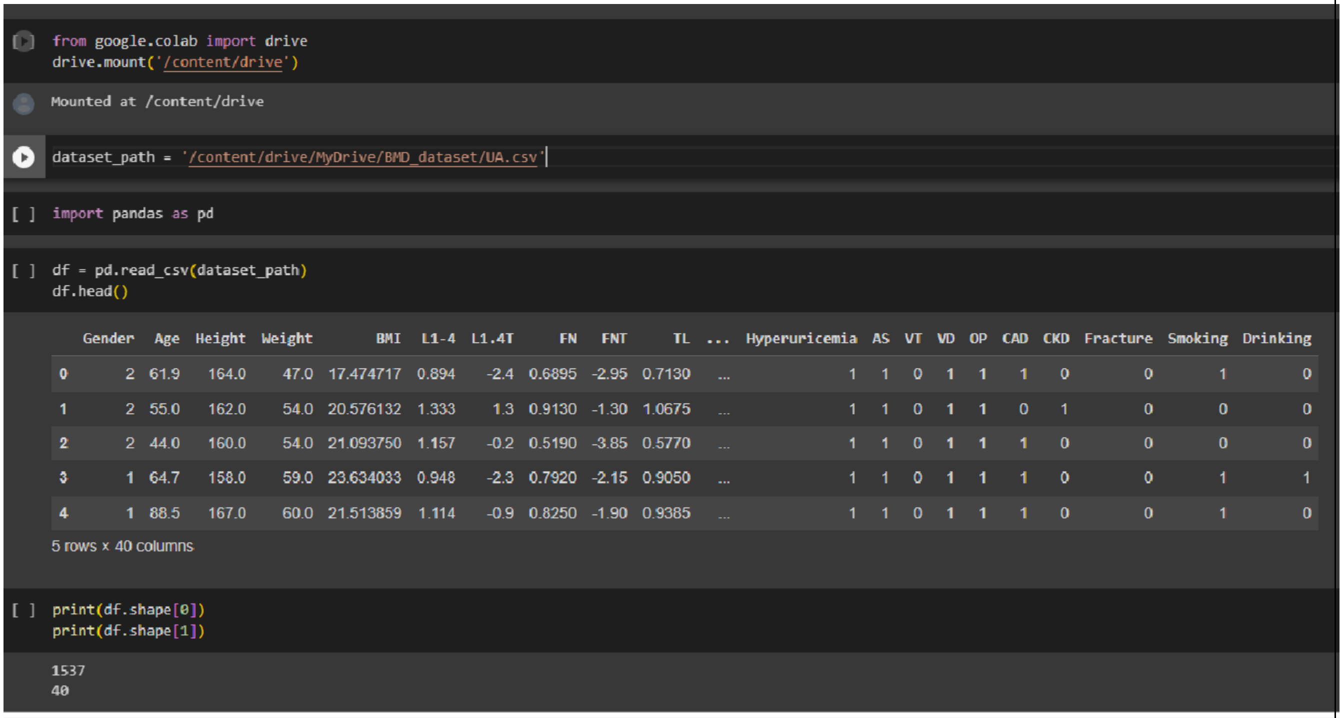
October 11, 2022. It encompasses clinical baseline information and the results of dual-energy X-ray measurements, making it a valuable resource for exploring the relationship between serum uric acid levels and the risk of osteoporosis. This dataset serves as the foundational data source for the mentioned research, facilitating further analysis and investigation in the fields of Medicine, Health, and Life Sciences.

**Subject**: Medicine, Health and Life Sciences

**Depositor**: Linfeng He

**Deposit Date:** October 11, 2022

## Code::



```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
dataset_path = '/content/drive/MyDrive/BMD_dataset/UA.csv'
```

```
[ ] import pandas as pd
```

```
[ ] df = pd.read_csv(dataset_path)
    df.head()
```

| | Gender | Age | Height | Weight | BMI | L1-4 | L1.4T | FN | FNT | TL | ... | Hyperuricemia | AS | VT | VD | OP | CAD | CKD | Fracture | Smoking | Drinking |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 61.9 | 164.0 | 47.0 | 17.474717 | 0.894 | -2.4 | 0.6895 | -2.95 | 0.7130 | ... | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 2 | 55.0 | 162.0 | 54.0 | 20.576132 | 1.333 | 1.3 | 0.9130 | -1.30 | 1.0675 | ... | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 2 | 2 | 44.0 | 160.0 | 54.0 | 21.093750 | 1.157 | -0.2 | 0.5190 | -3.85 | 0.5770 | ... | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 3 | 1 | 64.7 | 158.0 | 59.0 | 23.634033 | 0.948 | -2.3 | 0.7920 | -2.15 | 0.9050 | ... | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 4 | 1 | 88.5 | 167.0 | 60.0 | 21.513859 | 1.114 | -0.9 | 0.8250 | -1.90 | 0.9385 | ... | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |

5 rows × 40 columns

```
[ ] print(df.shape[0])
    print(df.shape[1])
```

```
1537
40
```

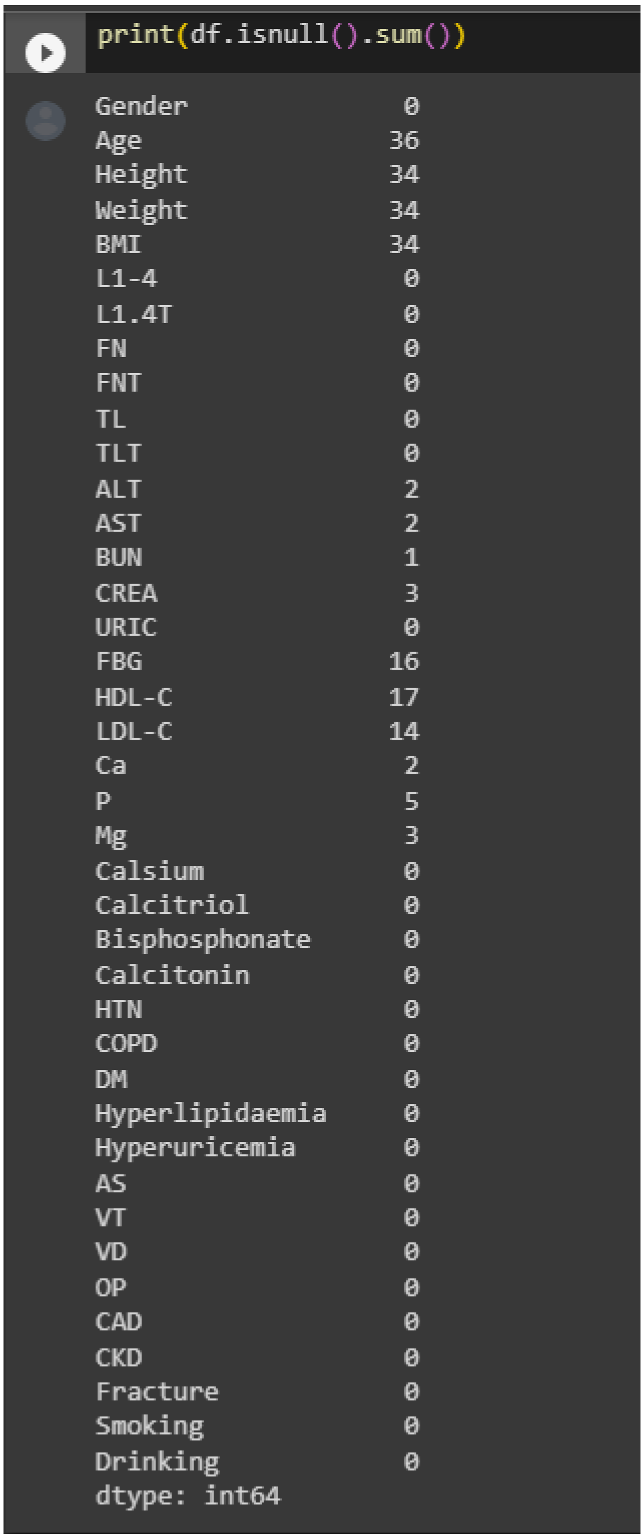
As shown in the above screenshot, we have first mounted the data from our drive that is : (‘/contet/drive’)

After that, we have set our dataset_path :
**'/content/drive/MyDrive/BMD_dataset/UA.csv'**

**Then after importing pandas we have generated outr dataframe, as shown above.**

**And as you can see have total Columns=40**

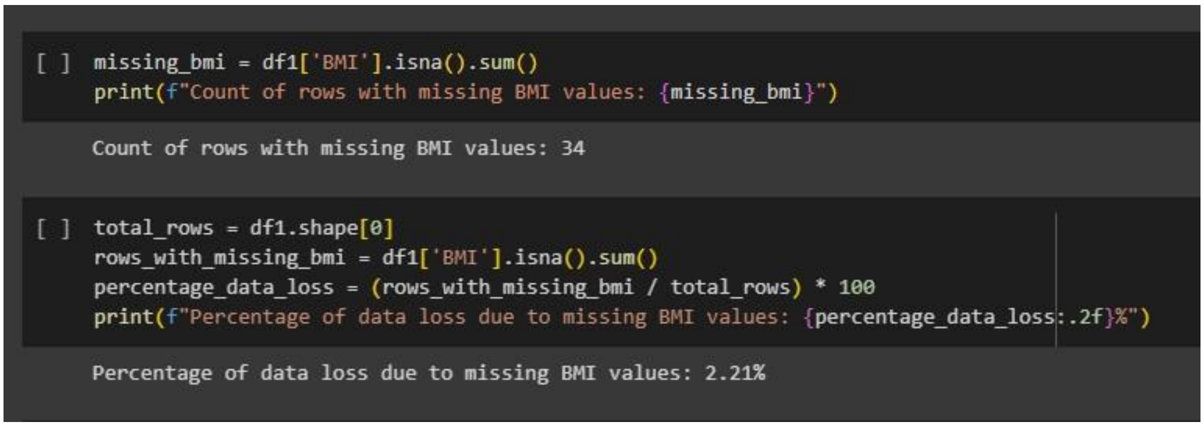
**Rows = 1537**



```
print(df.isnull().sum())
```

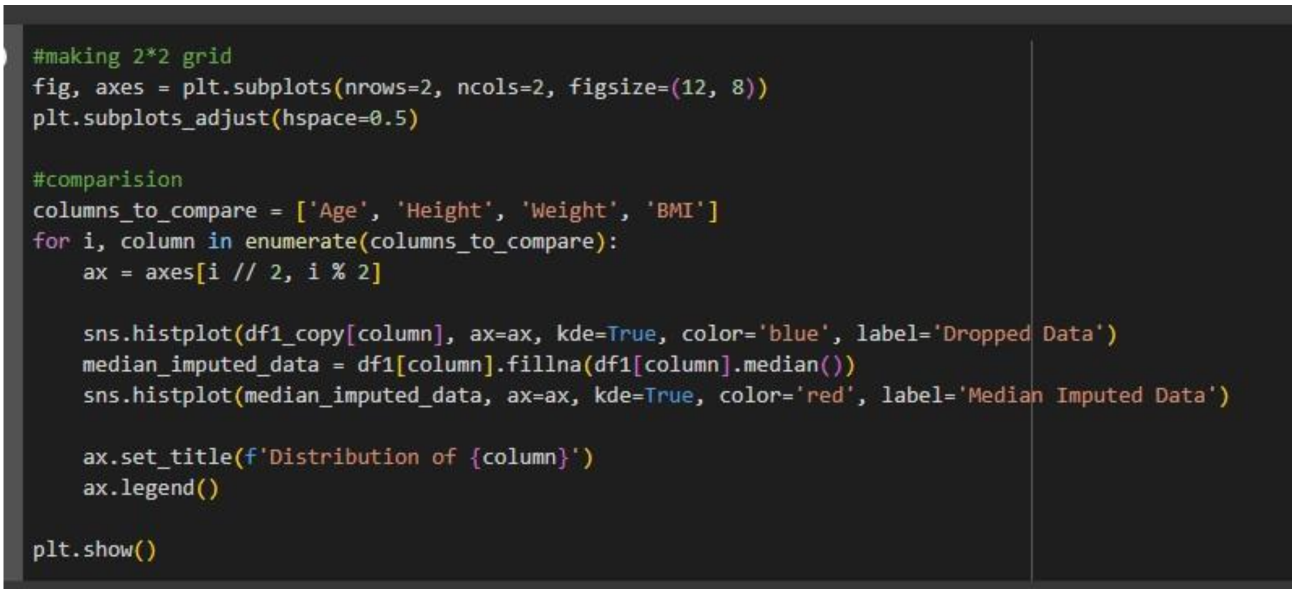
```
Gender              0
Age                36
Height             34
Weight             34
BMI                34
L1-4                0
L1.4T               0
FN                  0
FNT                 0
TL                  0
TLT                 0
ALT                 2
AST                 2
BUN                 1
CREA                3
URIC                0
FBG                16
HDL-C              17
LDL-C              14
Ca                  2
P                   5
Mg                  3
Calsium             0
Calcitriol          0
Bisphosphonate      0
Calcitonin          0
HTN                 0
COPD                0
DM                  0
Hyperlipidaemia     0
Hyperuricemia       0
AS                  0
VT                  0
VD                  0
OP                  0
CAD                 0
CKD                 0
Fracture            0
Smoking             0
Drinking            0
dtype: int64
```

Above are the total number of null values exists in our dataset.



```
[ ]  missing_bmi = df1['BMI'].isna().sum()
     print(f"Count of rows with missing BMI values: {missing_bmi}")

     Count of rows with missing BMI values: 34

[ ]  total_rows = df1.shape[0]
     rows_with_missing_bmi = df1['BMI'].isna().sum()
     percentage_data_loss = (rows_with_missing_bmi / total_rows) * 100
     print(f"Percentage of data loss due to missing BMI values: {percentage_data_loss:.2f}%")

     Percentage of data loss due to missing BMI values: 2.21%
```

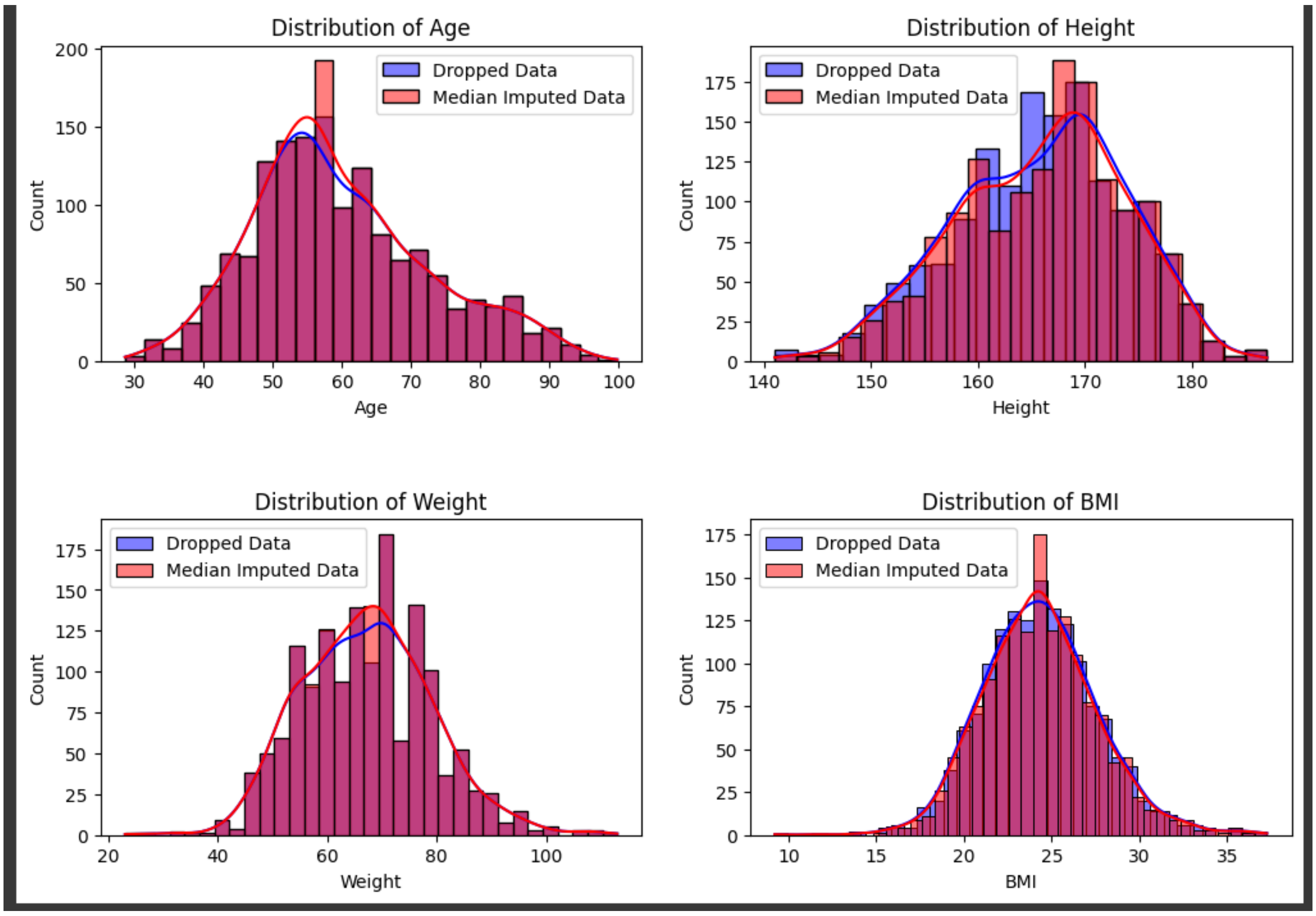
In the above code snippet, we have generated the count of rows with missing BMI values and the percentage of data loss due to missing BMI values which comes out to be 2.21%.



```
#making 2*2 grid
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 8))
plt.subplots_adjust(hspace=0.5)

#comparision
columns_to_compare = ['Age', 'Height', 'Weight', 'BMI']
for i, column in enumerate(columns_to_compare):
    ax = axes[i // 2, i % 2]

    sns.histplot(df1_copy[column], ax=ax, kde=True, color='blue', label='Dropped Data')
    median_imputed_data = df1[column].fillna(df1[column].median())
    sns.histplot(median_imputed_data, ax=ax, kde=True, color='red', label='Median Imputed Data')

    ax.set_title(f'Distribution of {column}')
    ax.legend()

plt.show()
```

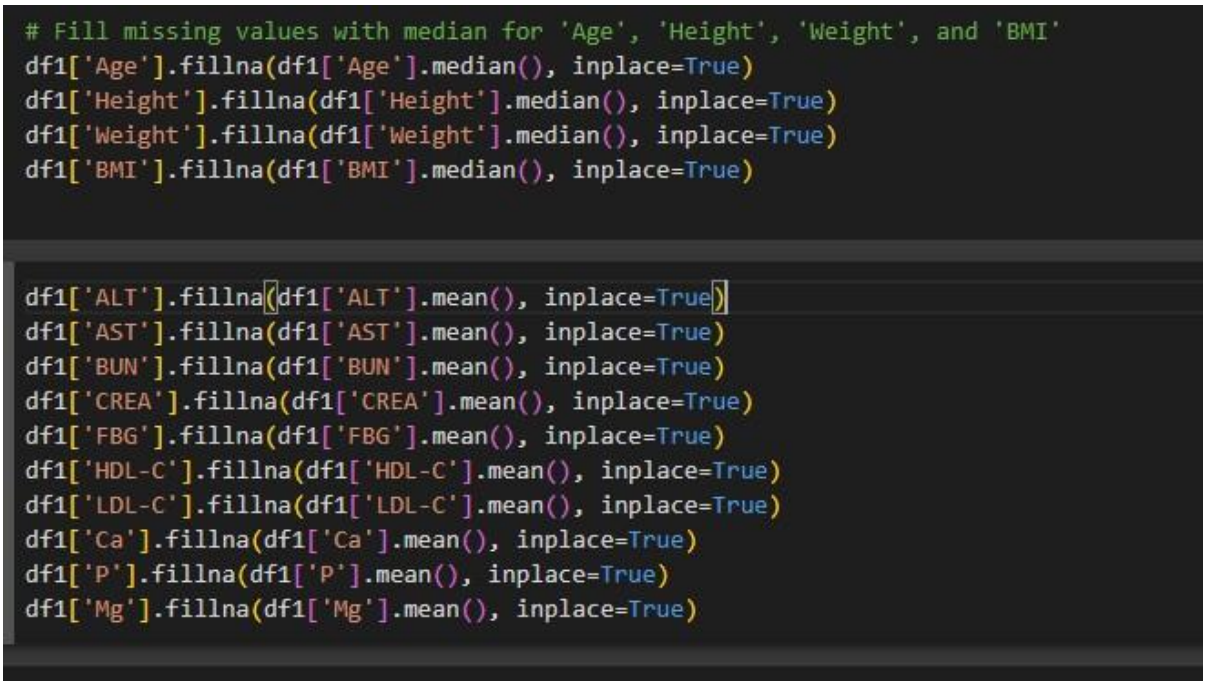
Above code snippet is used for drawing the graph of Distribution of Age, Height, Weight, BMI VS Count.

And below are the graphs we got after the above coding part::



So In the above dataset: there were some missing values, now we will fill all these values with the median of the data.

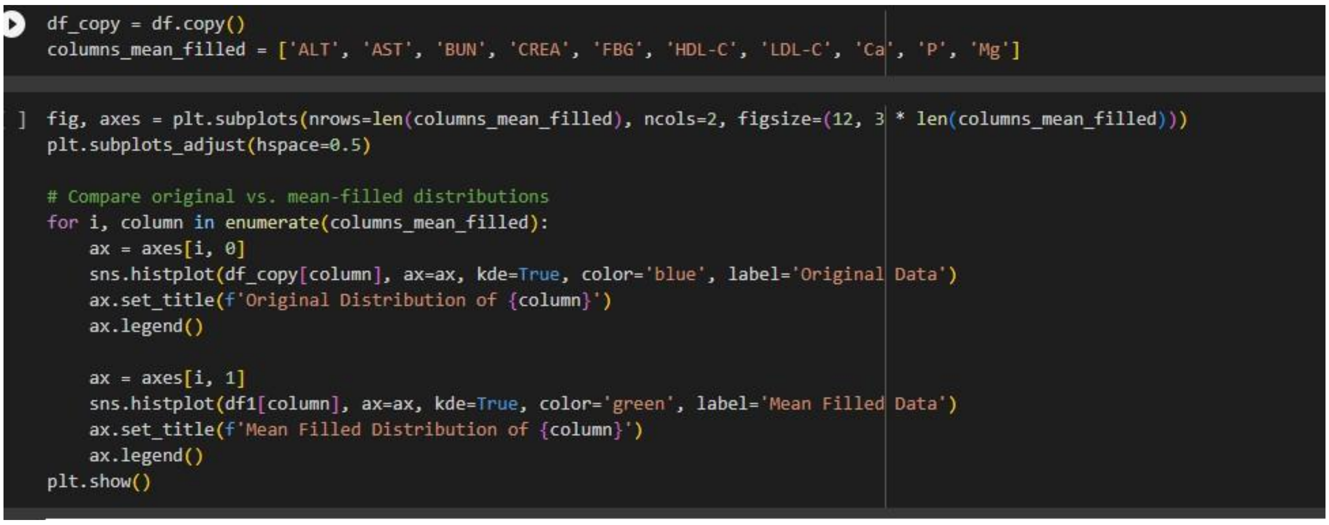
In the below code we are filling the missing data with median.



```
# Fill missing values with median for 'Age', 'Height', 'Weight', and 'BMI'
df1['Age'].fillna(df1['Age'].median(), inplace=True)
df1['Height'].fillna(df1['Height'].median(), inplace=True)
df1['Weight'].fillna(df1['Weight'].median(), inplace=True)
df1['BMI'].fillna(df1['BMI'].median(), inplace=True)
```

```
df1['ALT'].fillna(df1['ALT'].mean(), inplace=True)
df1['AST'].fillna(df1['AST'].mean(), inplace=True)
df1['BUN'].fillna(df1['BUN'].mean(), inplace=True)
df1['CREA'].fillna(df1['CREA'].mean(), inplace=True)
df1['FBG'].fillna(df1['FBG'].mean(), inplace=True)
df1['HDL-C'].fillna(df1['HDL-C'].mean(), inplace=True)
df1['LDL-C'].fillna(df1['LDL-C'].mean(), inplace=True)
df1['Ca'].fillna(df1['Ca'].mean(), inplace=True)
df1['P'].fillna(df1['P'].mean(), inplace=True)
df1['Mg'].fillna(df1['Mg'].mean(), inplace=True)
```

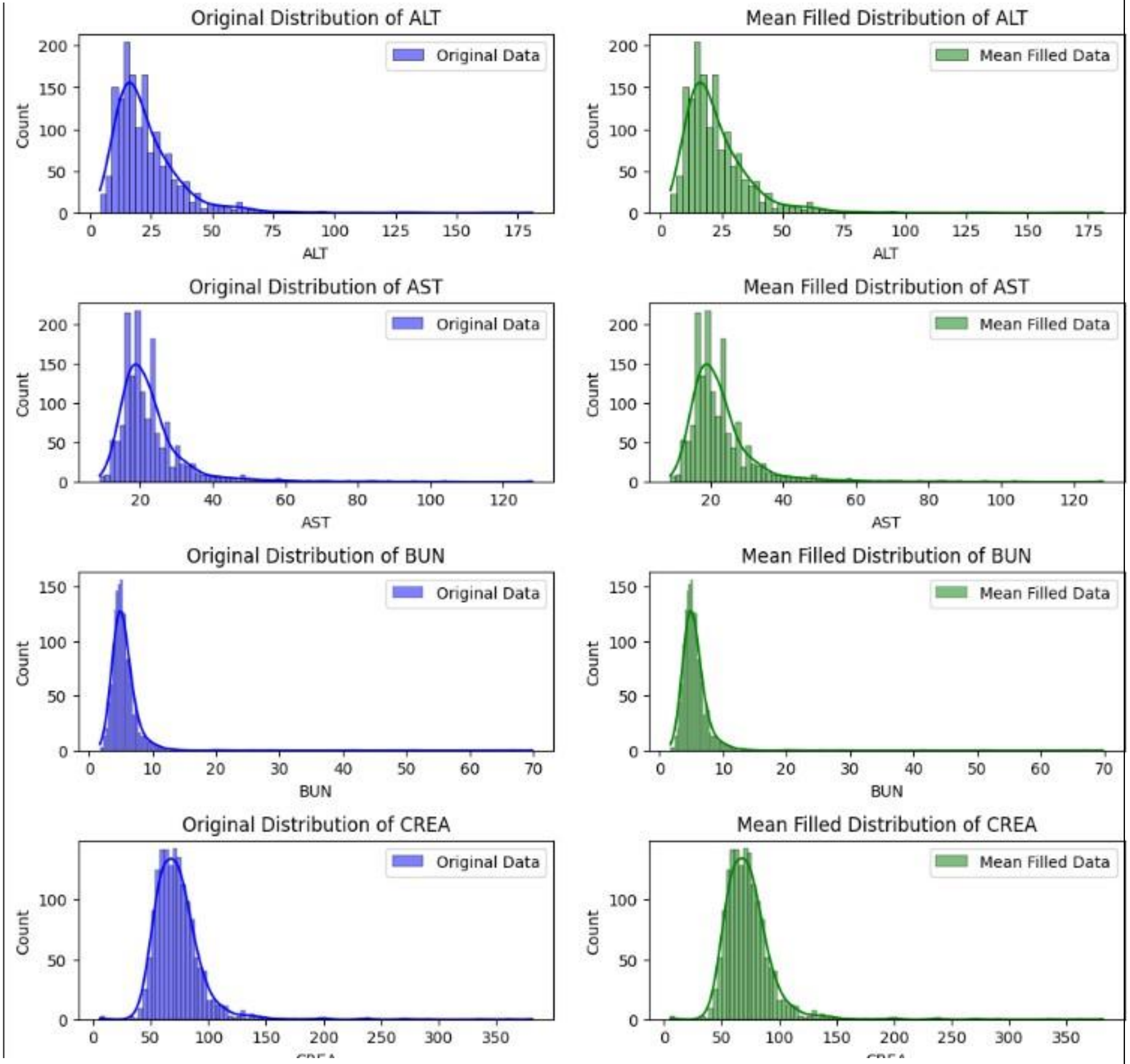
And the below columns missing values are filled by mean of the data. Below code shows the missing data filling with mean.

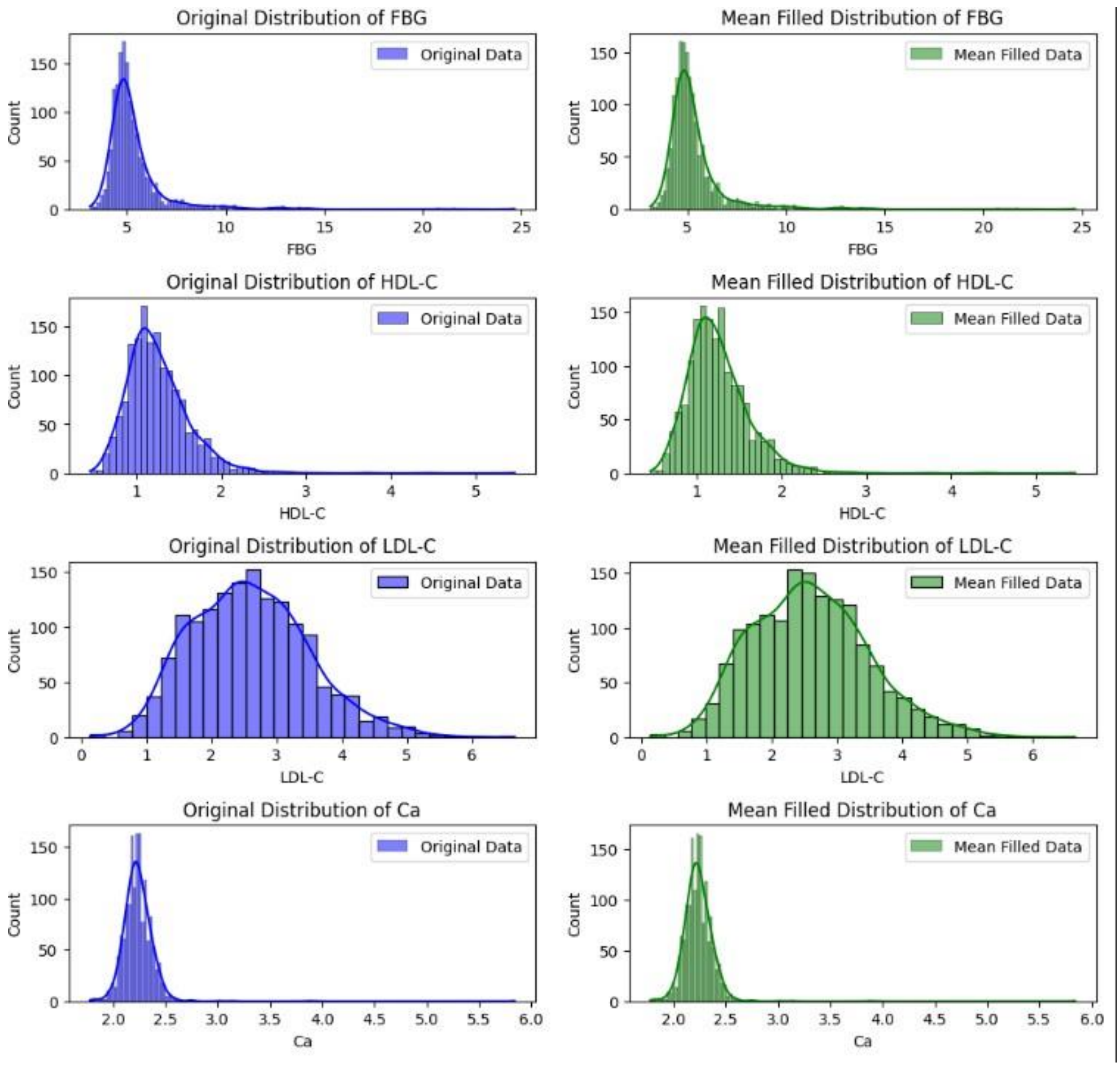


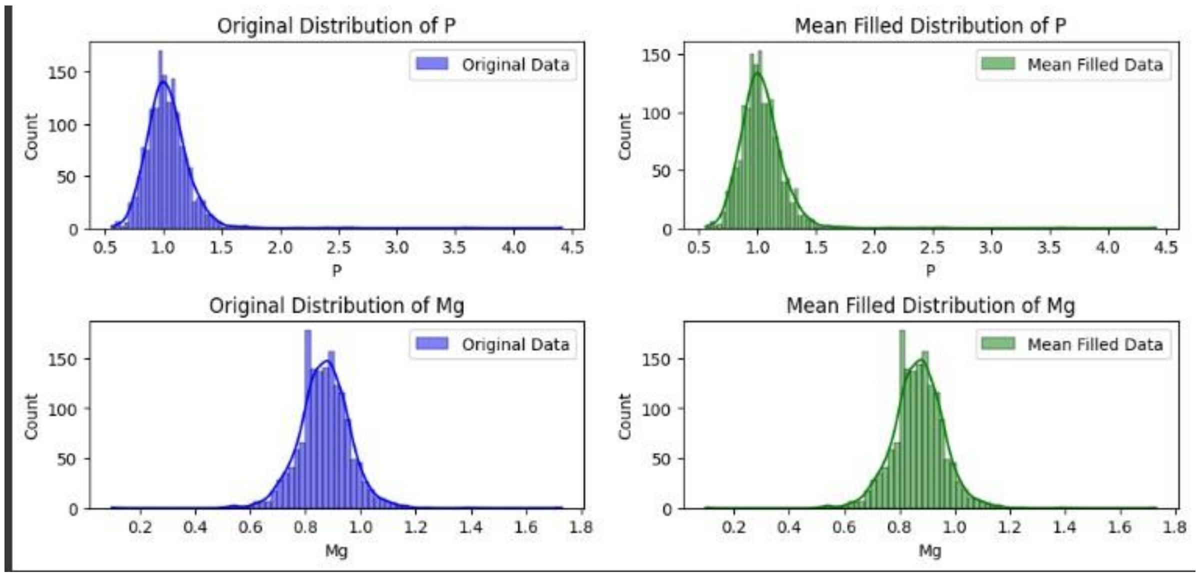
```
df_copy = df.copy()
columns_mean_filled = ['ALT', 'AST', 'BUN', 'CREA', 'FBG', 'HDL-C', 'LDL-C', 'Ca', 'P', 'Mg']
```

```
fig, axes = plt.subplots(nrows=len(columns_mean_filled), ncols=2, figsize=(12, 3 * len(columns_mean_filled)))
plt.subplots_adjust(hspace=0.5)

# Compare original vs. mean-filled distributions
for i, column in enumerate(columns_mean_filled):
    ax = axes[i, 0]
    sns.histplot(df_copy[column], ax=ax, kde=True, color='blue', label='Original Data')
    ax.set_title(f'Original Distribution of {column}')
    ax.legend()

    ax = axes[i, 1]
    sns.histplot(df1[column], ax=ax, kde=True, color='green', label='Mean Filled Data')
    ax.set_title(f'Mean Filled Distribution of {column}')
    ax.legend()
plt.show()
```

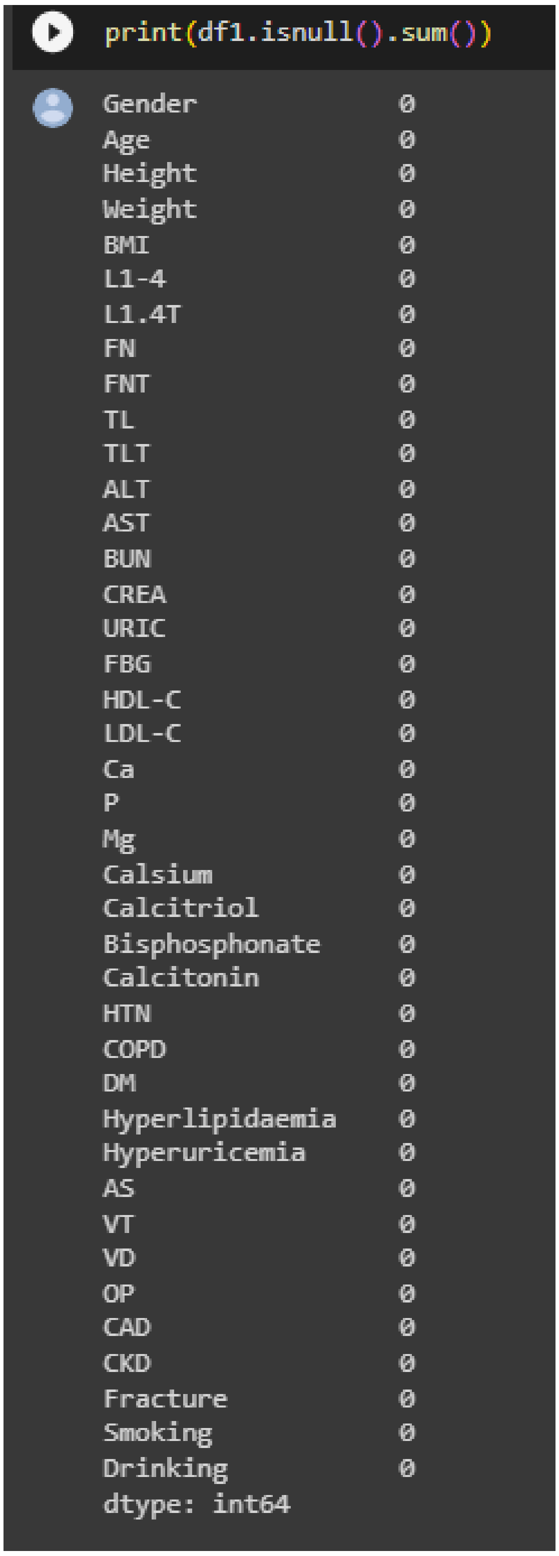
After filling all the missing values , we get the below graph before and after the mean filing data.


Original Distribution of ALT
Mean Filled Distribution of ALT
Original Distribution of AST
Mean Filled Distribution of AST
Original Distribution of BUN
Mean Filled Distribution of BUN
Original Distribution of CREA
Mean Filled Distribution of CREA
Original Data
Mean Filled Data
Count
ALT
AST
BUN


Original Distribution of FBG
Original Data
Count
FBG
Mean Filled Distribution of FBG
Mean Filled Data
Count
FBG
Original Distribution of HDL-C
Original Data
Count
HDL-C
Mean Filled Distribution of HDL-C
Mean Filled Data
Count
HDL-C
Original Distribution of LDL-C
Original Data
Count
LDL-C
Mean Filled Distribution of LDL-C
Mean Filled Data
Count
LDL-C
Original Distribution of Ca
Original Data
Count
Ca
Mean Filled Distribution of Ca
Mean Filled Data
Count
Ca

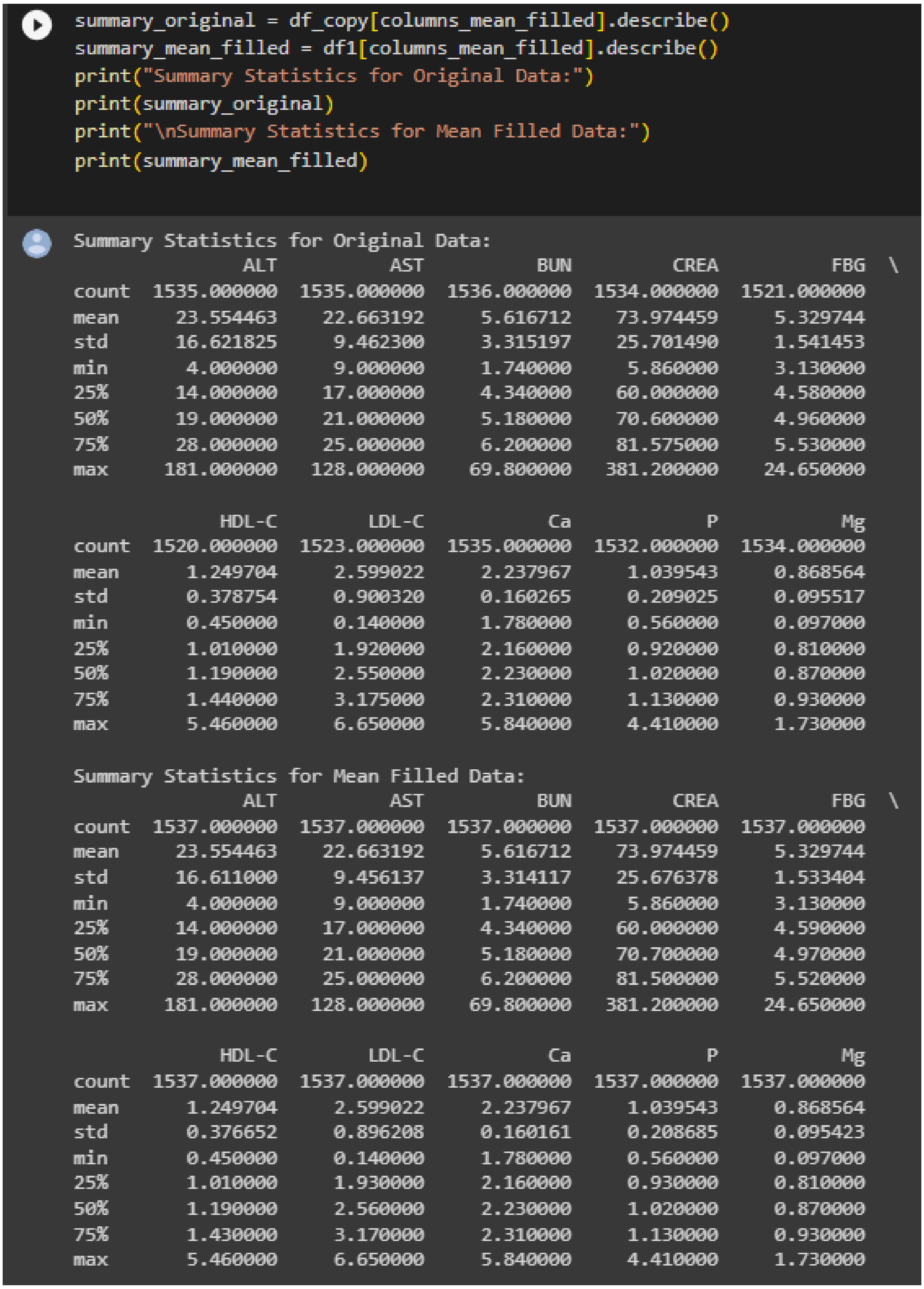

Original Distribution of P
Original Data
Count
P
Mean Filled Distribution of P
Mean Filled Data
Count
P
Original Distribution of Mg
Original Data
Count
Mg
Mean Filled Distribution of Mg
Mean Filled Data
Count
Mg

After all these missing values filled with mean and median of data. Now In the below screen shoot, you can see the all the null values becomes zero.



```
print(df1.isnull().sum())

Gender             0
Age                0
Height             0
Weight             0
BMI                0
L1-4               0
L1.4T              0
FN                 0
FNT                0
TL                 0
TLT                0
ALT                0
AST                0
BUN                0
CREA               0
URIC               0
FBG                0
HDL-C              0
LDL-C              0
Ca                 0
P                  0
Mg                 0
Calsium            0
Calcitriol         0
Bisphosphonate     0
Calcitonin         0
HTN                0
COPD               0
DM                 0
Hyperlipidaemia    0
Hyperuricemia      0
AS                 0
VT                 0
VD                 0
OP                 0
CAD                0
CKD                0
Fracture           0
Smoking            0
Drinking           0
dtype: int64
```

Below screen shoot shows the summary statistics of original data vs the Mean filled Data.

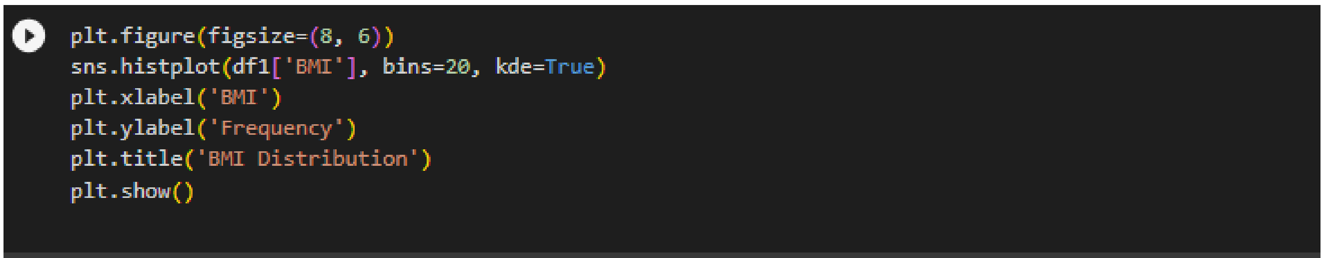


```
summary_original = df_copy[columns_mean_filled].describe()
summary_mean_filled = df1[columns_mean_filled].describe()
print("Summary Statistics for Original Data:")
print(summary_original)
print("\nSummary Statistics for Mean Filled Data:")
print(summary_mean_filled)
```

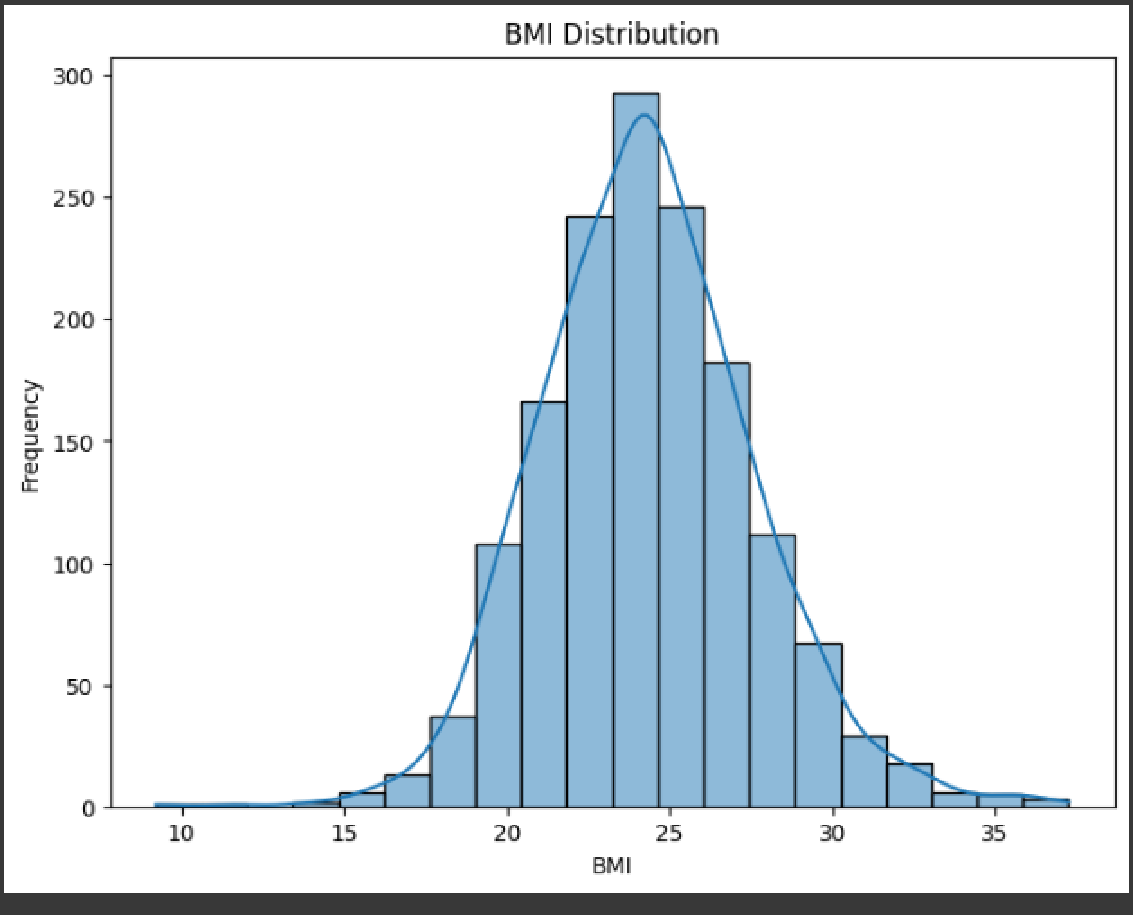
```
Summary Statistics for Original Data:
               ALT          AST          BUN         CREA          FBG  \
count  1535.000000  1535.000000  1536.000000  1534.000000  1521.000000
mean     23.554463    22.663192     5.616712    73.974459     5.329744
std      16.621825     9.462300     3.315197    25.701490     1.541453
min       4.000000     9.000000     1.740000     5.860000     3.130000
25%      14.000000    17.000000     4.340000    60.000000     4.580000
50%      19.000000    21.000000     5.180000    70.600000     4.960000
75%      28.000000    25.000000     6.200000    81.575000     5.530000
max     181.000000   128.000000    69.800000   381.200000    24.650000

             HDL-C        LDL-C           Ca            P           Mg
count  1520.000000  1523.000000  1535.000000  1532.000000  1534.000000
mean      1.249704     2.599022     2.237967     1.039543     0.868564
std       0.378754     0.900320     0.160265     0.209025     0.095517
min       0.450000     0.140000     1.780000     0.560000     0.097000
25%       1.010000     1.920000     2.160000     0.920000     0.810000
50%       1.190000     2.550000     2.230000     1.020000     0.870000
75%       1.440000     3.175000     2.310000     1.130000     0.930000
max       5.460000     6.650000     5.840000     4.410000     1.730000

Summary Statistics for Mean Filled Data:
               ALT          AST          BUN         CREA          FBG  \
count  1537.000000  1537.000000  1537.000000  1537.000000  1537.000000
mean     23.554463    22.663192     5.616712    73.974459     5.329744
std      16.611000     9.456137     3.314117    25.676378     1.533404
min       4.000000     9.000000     1.740000     5.860000     3.130000
25%      14.000000    17.000000     4.340000    60.000000     4.590000
50%      19.000000    21.000000     5.180000    70.700000     4.970000
75%      28.000000    25.000000     6.200000    81.500000     5.520000
max     181.000000   128.000000    69.800000   381.200000    24.650000

             HDL-C        LDL-C           Ca            P           Mg
count  1537.000000  1537.000000  1537.000000  1537.000000  1537.000000
mean      1.249704     2.599022     2.237967     1.039543     0.868564
std       0.376652     0.896208     0.160161     0.208685     0.095423
min       0.450000     0.140000     1.780000     0.560000     0.097000
25%       1.010000     1.930000     2.160000     0.930000     0.810000
50%       1.190000     2.560000     2.230000     1.020000     0.870000
75%       1.430000     3.170000     2.310000     1.130000     0.930000
max       5.460000     6.650000     5.840000     4.410000     1.730000
```

## Data Visualization:

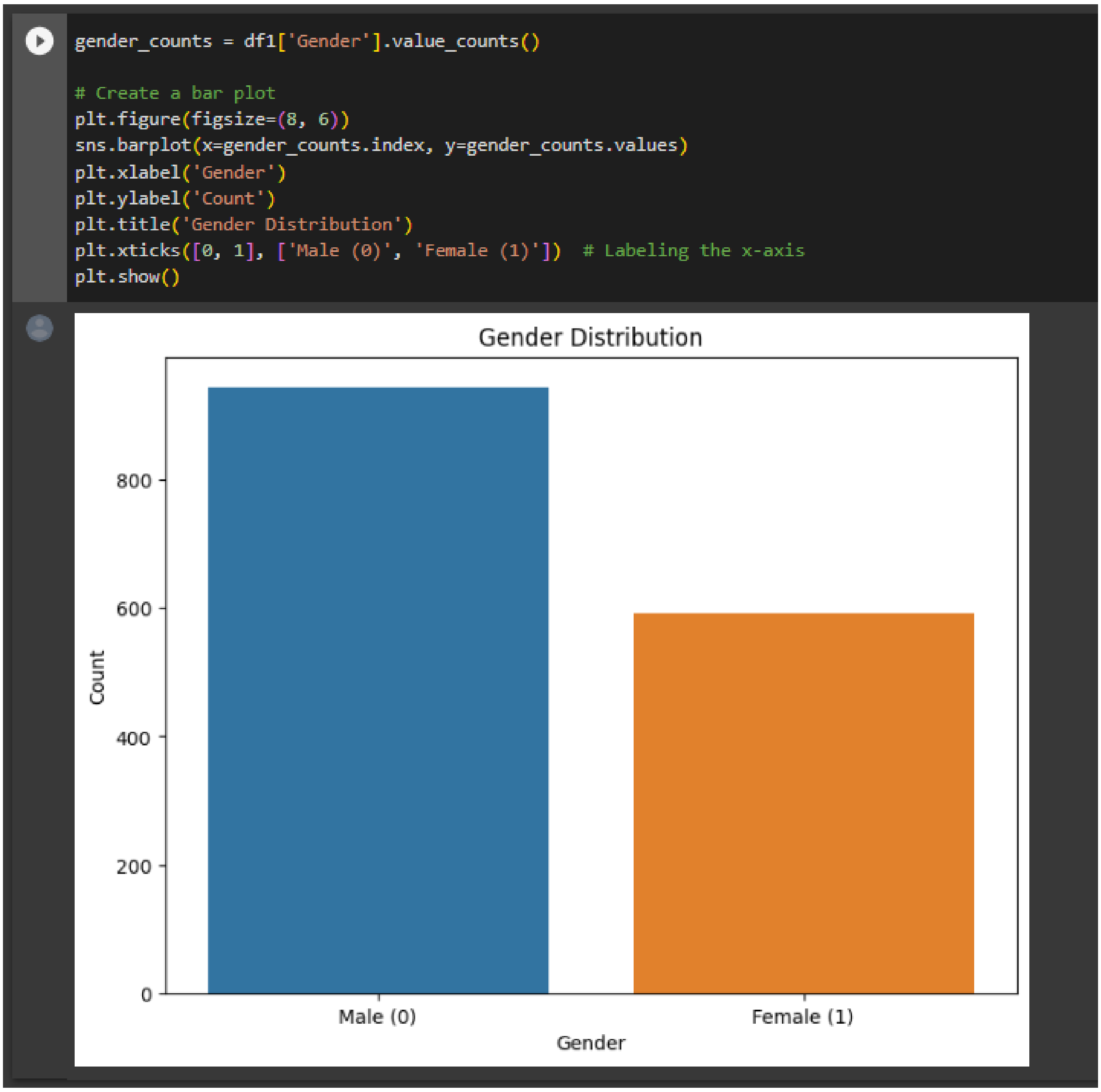
Relation between the BMI distribution Vs Frequency using the histogram.



```
plt.figure(figsize=(8, 6))
sns.histplot(df1['BMI'], bins=20, kde=True)
plt.xlabel('BMI')
plt.ylabel('Frequency')
plt.title('BMI Distribution')
plt.show()
```

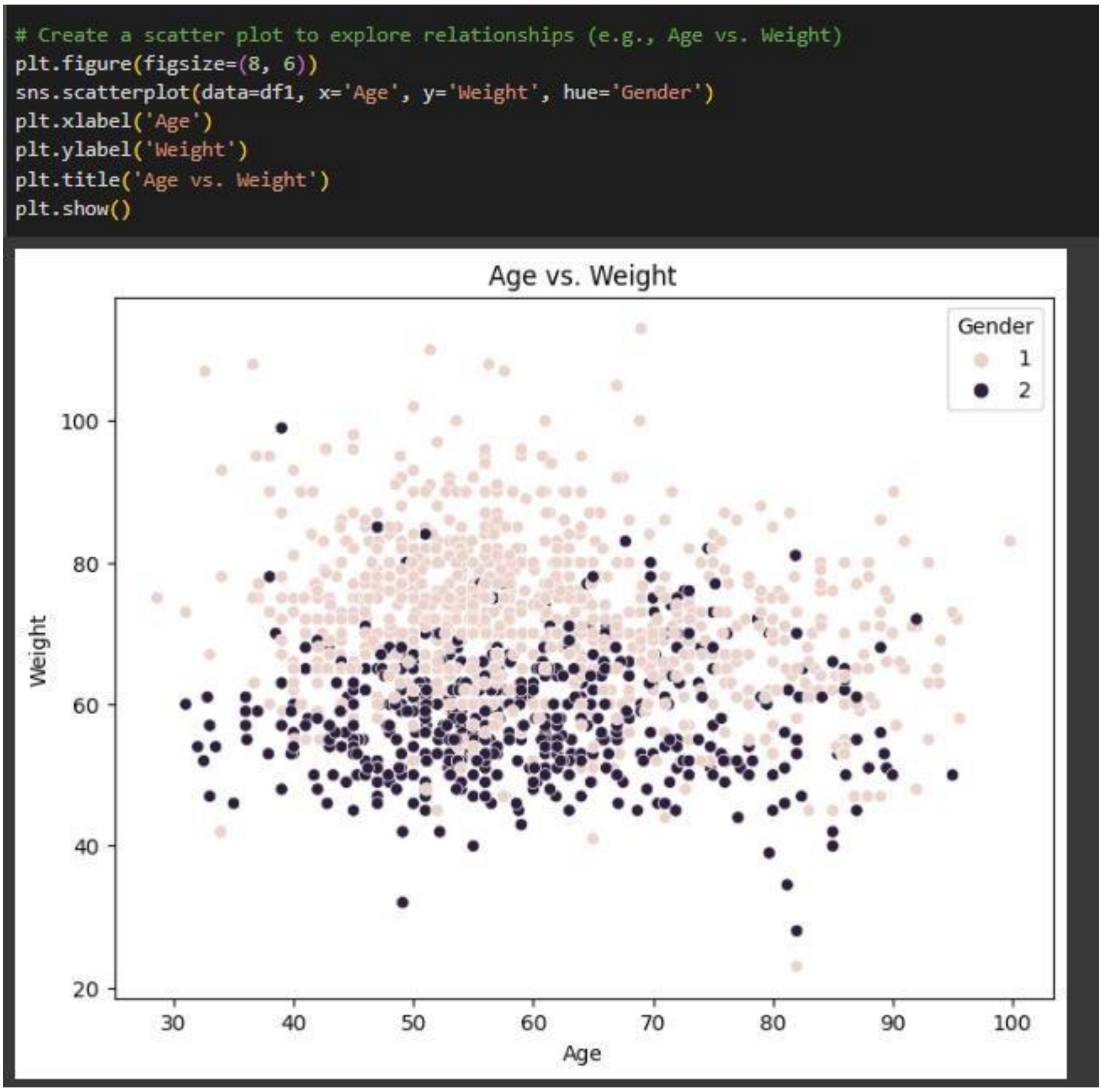


Below is the relation between Gender Distribution Vs Count



```python
gender_counts = df1['Gender'].value_counts()

# Create a bar plot
plt.figure(figsize=(8, 6))
sns.barplot(x=gender_counts.index, y=gender_counts.values)
plt.xlabel('Gender')
plt.ylabel('Count')
plt.title('Gender Distribution')
plt.xticks([0, 1], ['Male (0)', 'Female (1)'])  # Labeling the x-axis
plt.show()
```

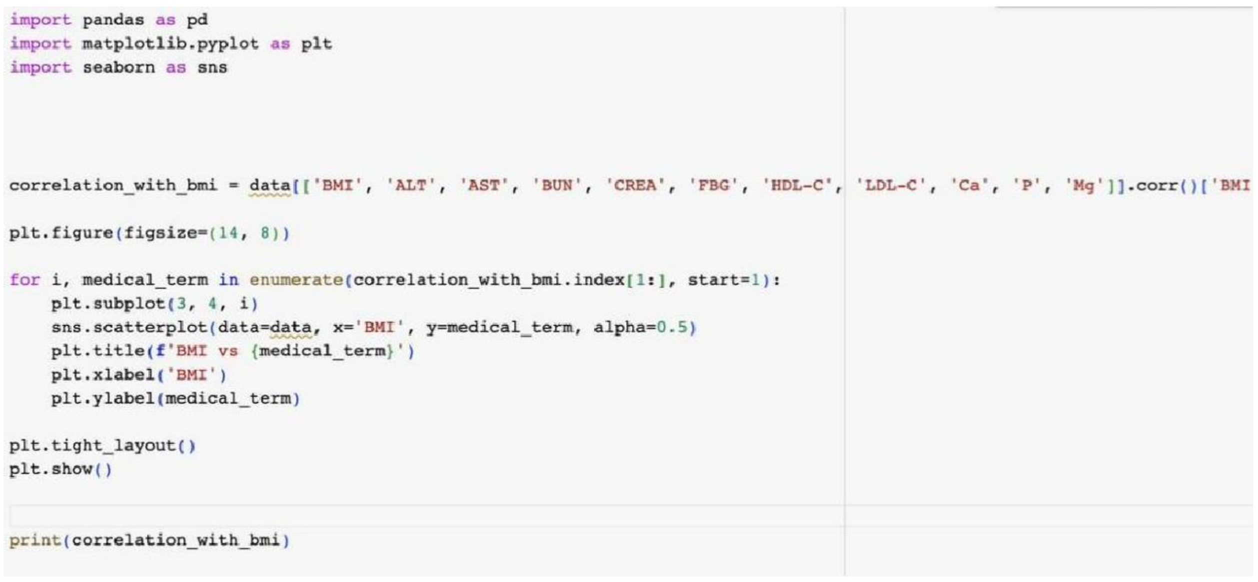
- Below Screen shot show the scatter plot to explore relationships (e.g Age Vs Weight)



```
# Create a scatter plot to explore relationships (e.g., Age vs. Weight)
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df1, x='Age', y='Weight', hue='Gender')
plt.xlabel('Age')
plt.ylabel('Weight')
plt.title('Age vs. Weight')
plt.show()
```

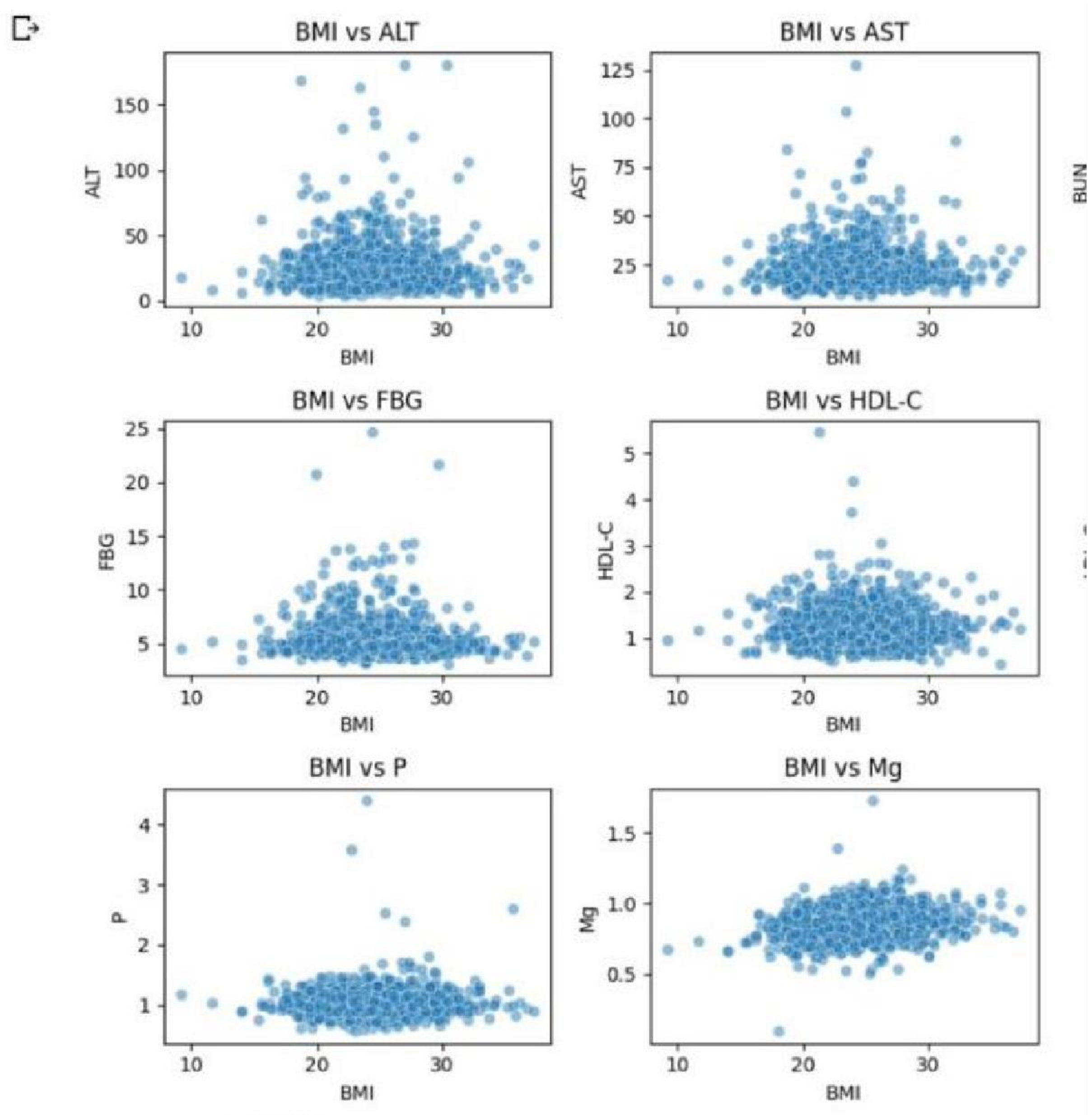
## Scatter Plot of All data Vs BMI

A scatter plot is a visual representation used to display the relationship between two variables. It consists of individual data points plotted on a two-dimensional graph, with one variable on the x-axis and the other on the y-axis. Scatter plots help reveal patterns, trends, and correlations in the data, making them valuable tools in data analysis and visualization.

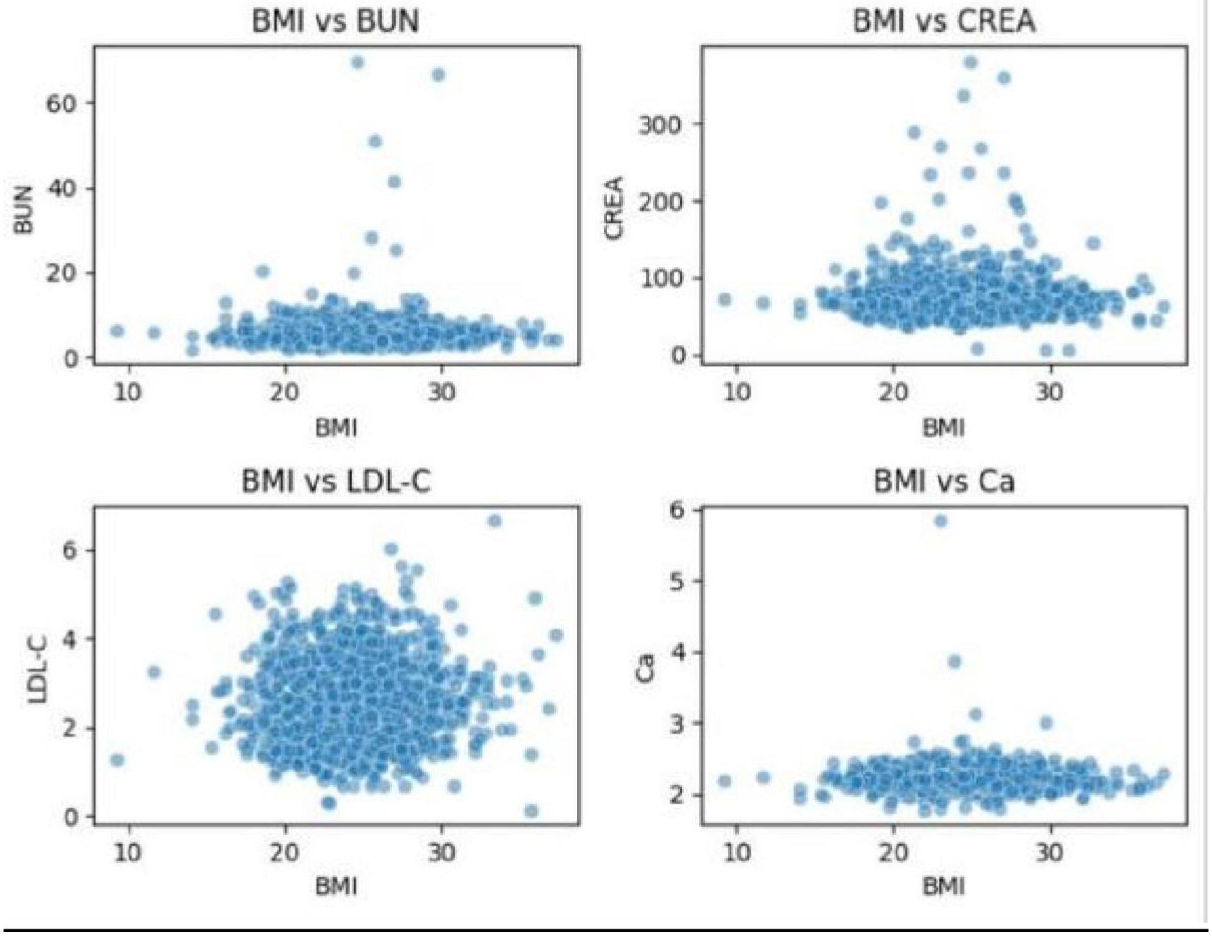


```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns



correlation_with_bmi = data[['BMI', 'ALT', 'AST', 'BUN', 'CREA', 'FBG', 'HDL-C', 'LDL-C', 'Ca', 'P', 'Mg']].corr()['BMI

plt.figure(figsize=(14, 8))

for i, medical_term in enumerate(correlation_with_bmi.index[1:], start=1):
    plt.subplot(3, 4, i)
    sns.scatterplot(data=data, x='BMI', y=medical_term, alpha=0.5)
    plt.title(f'BMI vs {medical_term}')
    plt.xlabel('BMI')
    plt.ylabel(medical_term)

plt.tight_layout()
plt.show()


print(correlation_with_bmi)
```

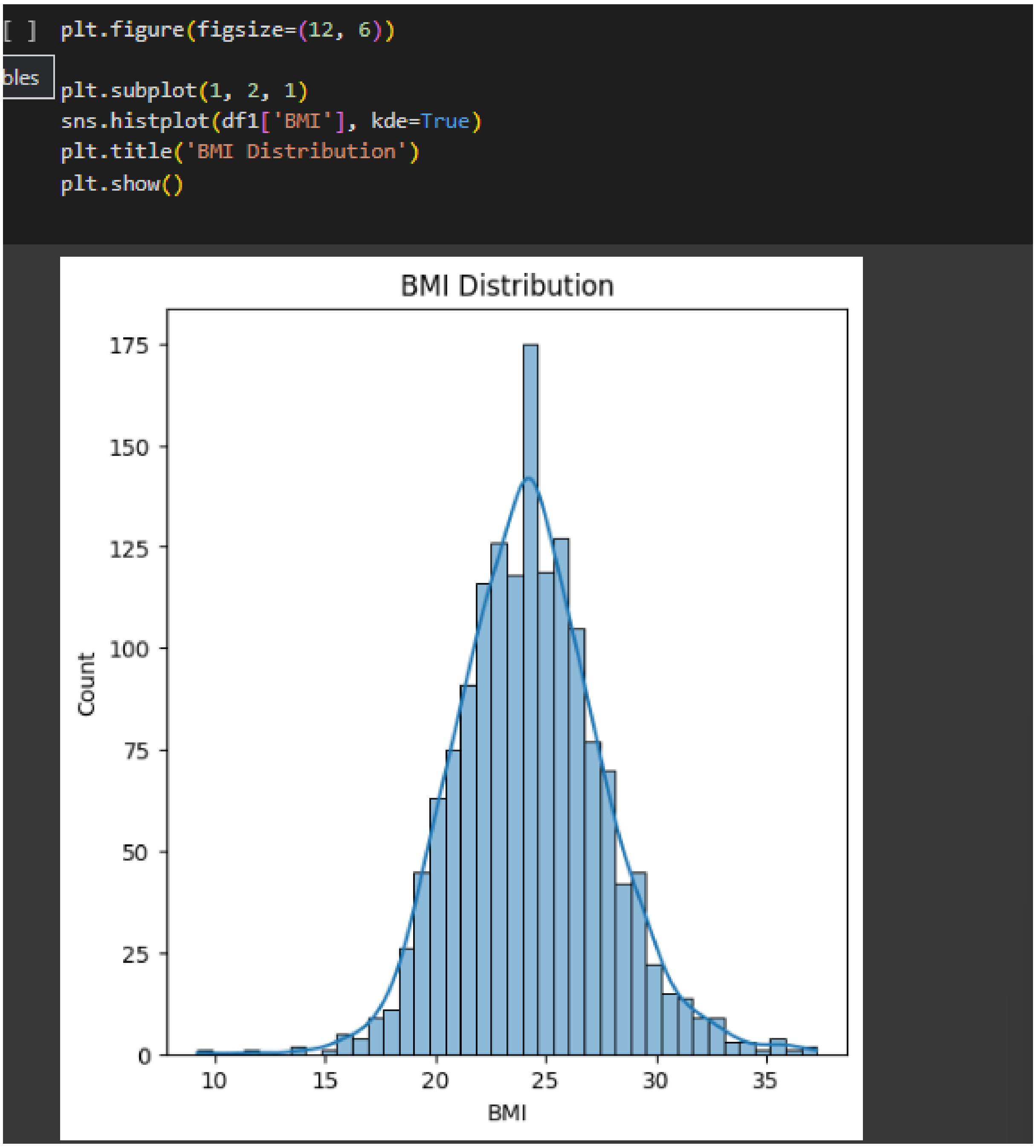
Below graph are scatter plot of other data vs BMI:



```
BMI       1.000000
ALT       0.005828
AST      -0.002530
BUN       0.026686
CREA     -0.008656
```


BMI vs BUN
BUN
BMI
BMI vs CREA
CREA
BMI
BMI vs LDL-C
LDL-C
BMI
BMI vs Ca
Ca
BMI

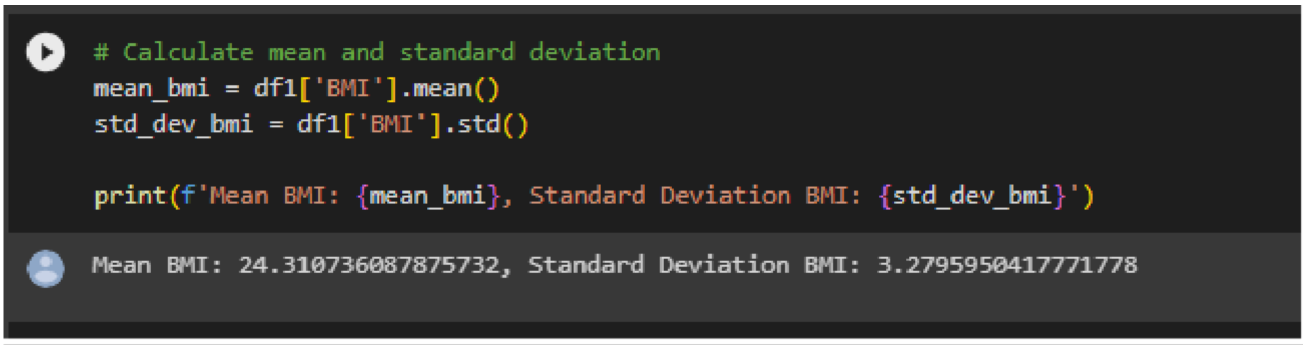
- Below histogram shows the BMI distribution Vs Count graph.



```
[ ]  plt.figure(figsize=(12, 6))
bles
     plt.subplot(1, 2, 1)
     sns.histplot(df1['BMI'], kde=True)
     plt.title('BMI Distribution')
     plt.show()
```

# UNIT -2

## Calculation of Mean and Median of above population.



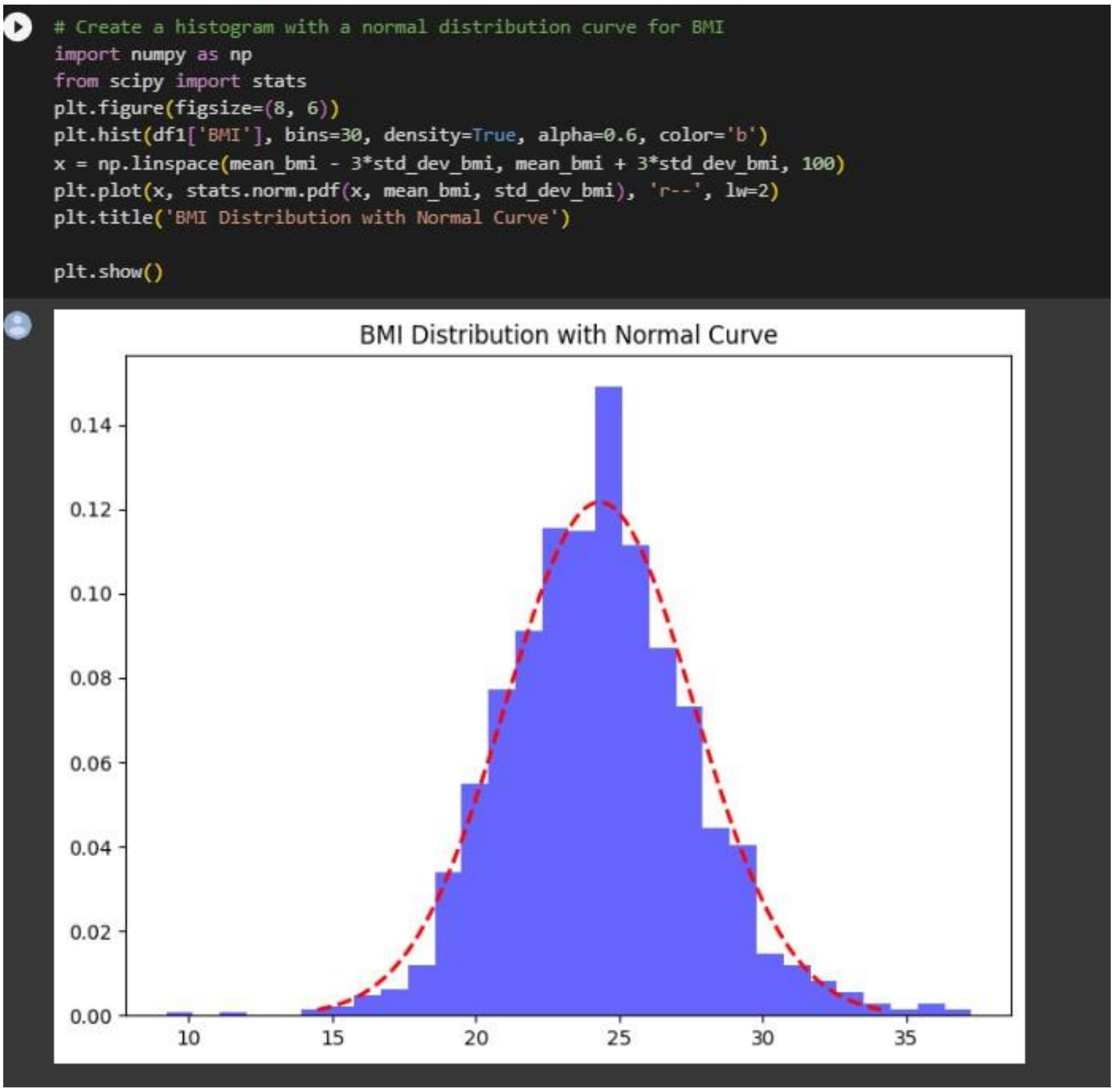
```
# Calculate mean and standard deviation
mean_bmi = df1['BMI'].mean()
std_dev_bmi = df1['BMI'].std()

print(f'Mean BMI: {mean_bmi}, Standard Deviation BMI: {std_dev_bmi}')
```

```
Mean BMI: 24.310736087875732, Standard Deviation BMI: 3.2795950417771778
```

So ,from the above population, we got:

- Mean BMI: 24.310736087875732,
- Standard Deviation BMI: 3.2795950417771778

## Below picuture shows the normal distribution curve of BMI of above population

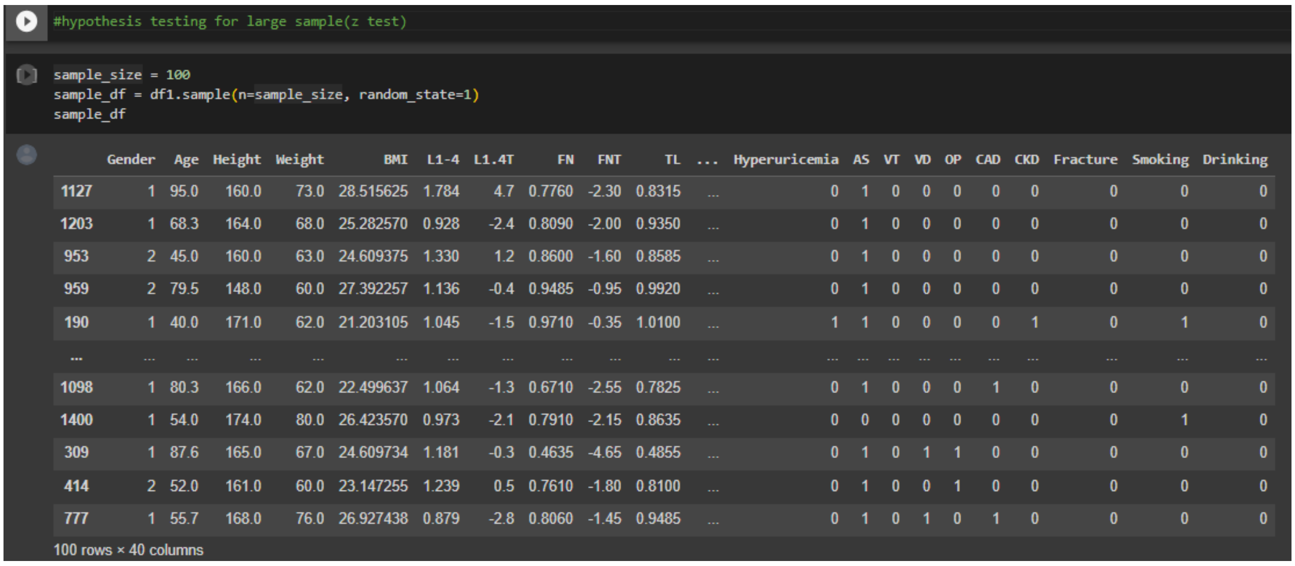


```
# Create a histogram with a normal distribution curve for BMI
import numpy as np
from scipy import stats
plt.figure(figsize=(8, 6))
plt.hist(df1['BMI'], bins=30, density=True, alpha=0.6, color='b')
x = np.linspace(mean_bmi - 3*std_dev_bmi, mean_bmi + 3*std_dev_bmi, 100)
plt.plot(x, stats.norm.pdf(x, mean_bmi, std_dev_bmi), 'r--', lw=2)
plt.title('BMI Distribution with Normal Curve')

plt.show()
```

# TESTING:

## Hypothesis Testing (large values):

Hypothesis testing with a large sample (typically over 30) assesses population parameters using Z-tests or t-tests. Small sample testing (usually under 30) employs t-tests, considering sample size's impact. Both require random, representative samples and follow normal distribution assumptions to determine whether to accept or reject the null hypothesis based on significance levels and test statistics.



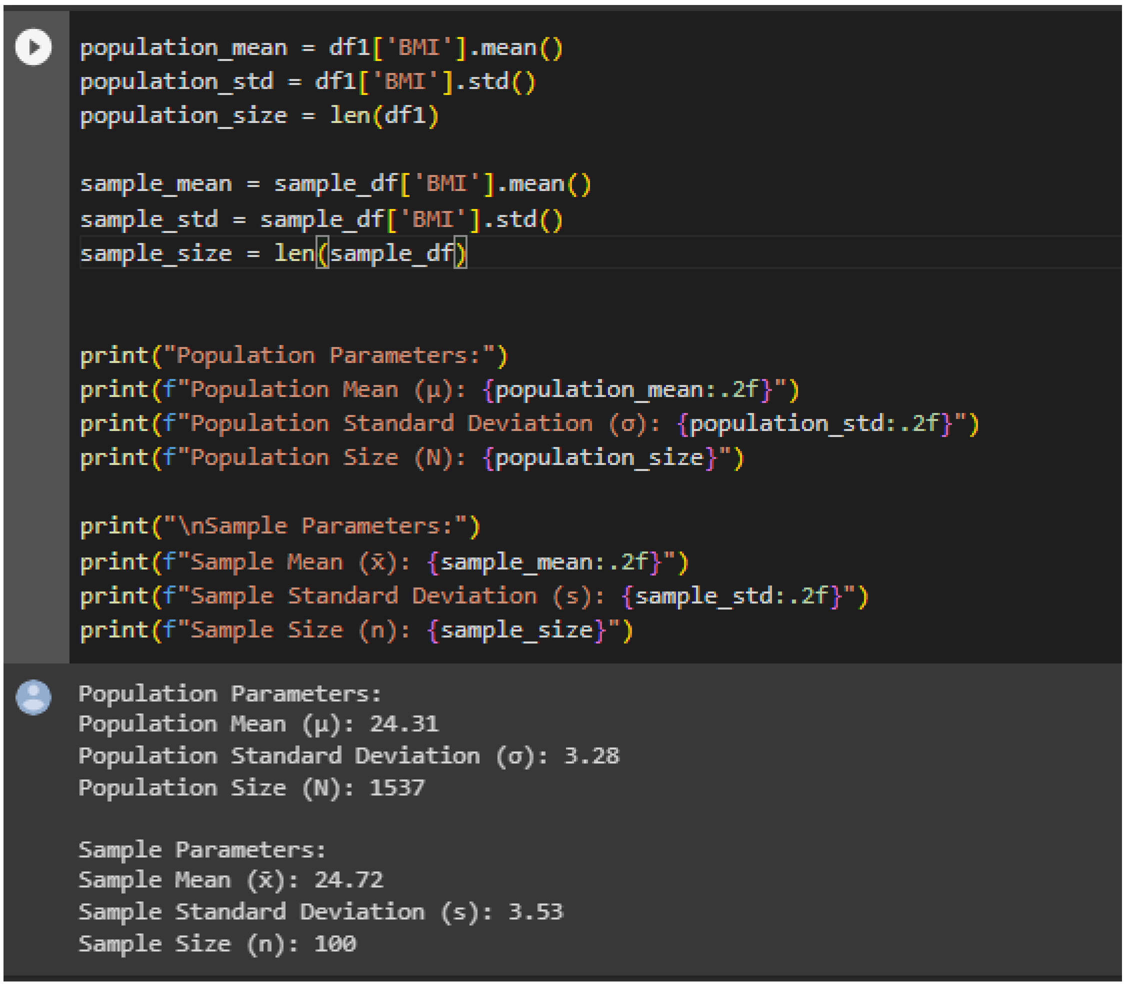
```
#hypothesis testing for large sample(z test)
```

```
sample_size = 100
sample_df = df1.sample(n=sample_size, random_state=1)
sample_df
```

| | Gender | Age | Height | Weight | BMI | L1-4 | L1.4T | FN | FNT | TL | ... | Hyperuricemia | AS | VT | VD | OP | CAD | CKD | Fracture | Smoking | Drinking |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1127** | 1 | 95.0 | 160.0 | 73.0 | 28.515625 | 1.784 | 4.7 | 0.7760 | -2.30 | 0.8315 | ... | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1203** | 1 | 68.3 | 164.0 | 68.0 | 25.282570 | 0.928 | -2.4 | 0.8090 | -2.00 | 0.9350 | ... | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **953** | 2 | 45.0 | 160.0 | 63.0 | 24.609375 | 1.330 | 1.2 | 0.8600 | -1.60 | 0.8585 | ... | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **959** | 2 | 79.5 | 148.0 | 60.0 | 27.392257 | 1.136 | -0.4 | 0.9485 | -0.95 | 0.9920 | ... | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **190** | 1 | 40.0 | 171.0 | 62.0 | 21.203105 | 1.045 | -1.5 | 0.9710 | -0.35 | 1.0100 | ... | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1098** | 1 | 80.3 | 166.0 | 62.0 | 22.499637 | 1.064 | -1.3 | 0.6710 | -2.55 | 0.7825 | ... | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **1400** | 1 | 54.0 | 174.0 | 80.0 | 26.423570 | 0.973 | -2.1 | 0.7910 | -2.15 | 0.8635 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **309** | 1 | 87.6 | 165.0 | 67.0 | 24.609734 | 1.181 | -0.3 | 0.4635 | -4.65 | 0.4855 | ... | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| **414** | 2 | 52.0 | 161.0 | 60.0 | 23.147255 | 1.239 | 0.5 | 0.7610 | -1.80 | 0.8100 | ... | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **777** | 1 | 55.7 | 168.0 | 76.0 | 26.927438 | 0.879 | -2.8 | 0.8060 | -1.45 | 0.9485 | ... | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |

100 rows × 40 columns

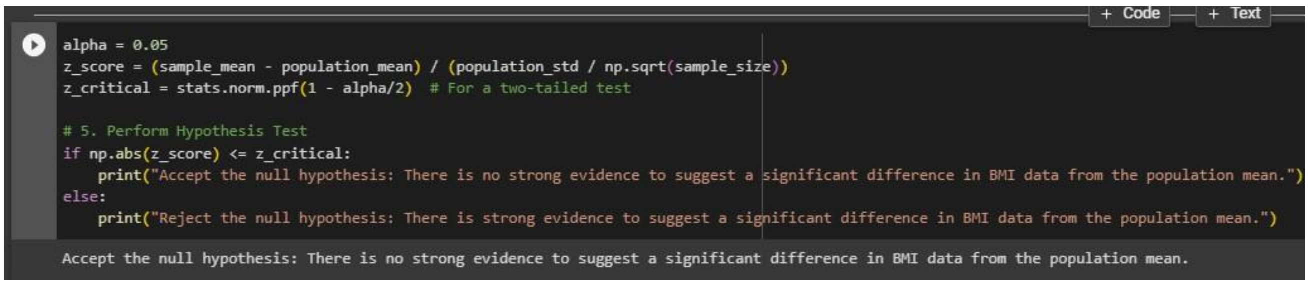
Mean , Std Deviation , Size:



```
population_mean = df1['BMI'].mean()
population_std = df1['BMI'].std()
population_size = len(df1)

sample_mean = sample_df['BMI'].mean()
sample_std = sample_df['BMI'].std()
sample_size = len(sample_df)


print("Population Parameters:")
print(f"Population Mean (μ): {population_mean:.2f}")
print(f"Population Standard Deviation (σ): {population_std:.2f}")
print(f"Population Size (N): {population_size}")

print("\nSample Parameters:")
print(f"Sample Mean (x̄): {sample_mean:.2f}")
print(f"Sample Standard Deviation (s): {sample_std:.2f}")
print(f"Sample Size (n): {sample_size}")
```

```
Population Parameters:
Population Mean (μ): 24.31
Population Standard Deviation (σ): 3.28
Population Size (N): 1537

Sample Parameters:
Sample Mean (x̄): 24.72
Sample Standard Deviation (s): 3.53
Sample Size (n): 100
```

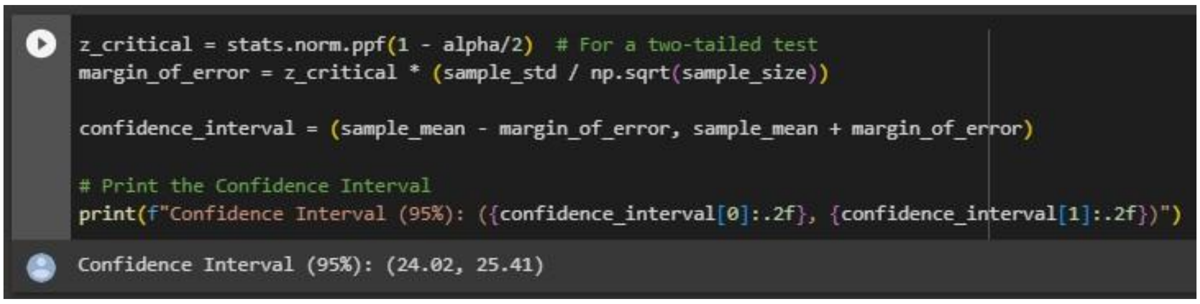
After the hypothesis testing, we get to know that, There is no strong evidence to suggest a significant difference in BMI data from the population mean.



```
alpha = 0.05
z_score = (sample_mean - population_mean) / (population_std / np.sqrt(sample_size))
z_critical = stats.norm.ppf(1 - alpha/2)  # For a two-tailed test

# 5. Perform Hypothesis Test
if np.abs(z_score) <= z_critical:
    print("Accept the null hypothesis: There is no strong evidence to suggest a significant difference in BMI data from the population mean.")
else:
    print("Reject the null hypothesis: There is strong evidence to suggest a significant difference in BMI data from the population mean.")
```

```
Accept the null hypothesis: There is no strong evidence to suggest a significant difference in BMI data from the population mean.
```

And Confidence Interval we got is 95%.

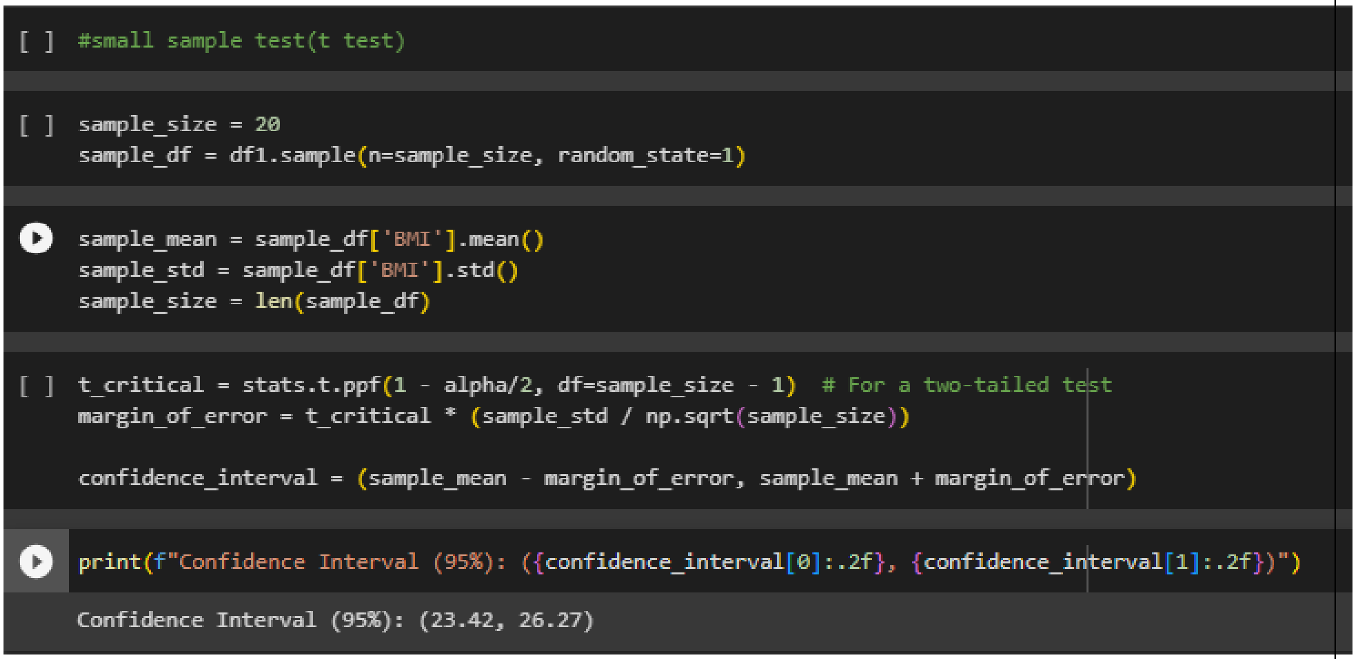


```
z_critical = stats.norm.ppf(1 - alpha/2)  # For a two-tailed test
margin_of_error = z_critical * (sample_std / np.sqrt(sample_size))

confidence_interval = (sample_mean - margin_of_error, sample_mean + margin_of_error)

# Print the Confidence Interval
print(f"Confidence Interval (95%): ({confidence_interval[0]:.2f}, {confidence_interval[1]:.2f})")
```

```
Confidence Interval (95%): (24.02, 25.41)
```

## Small Hypothesis Testing :

Small sample hypothesis testing analyzes limited data (typically fewer than 30 observations) using t-tests. Assumptions include random, representative sampling and adherence to normal distribution assumptions. Researchers compare the calculated t-test statistic with critical values or calculate p-values to decide whether to accept or reject the null hypothesis based on significance levels.



```
#small sample test(t test)
```

```
sample_size = 20
sample_df = df1.sample(n=sample_size, random_state=1)
```

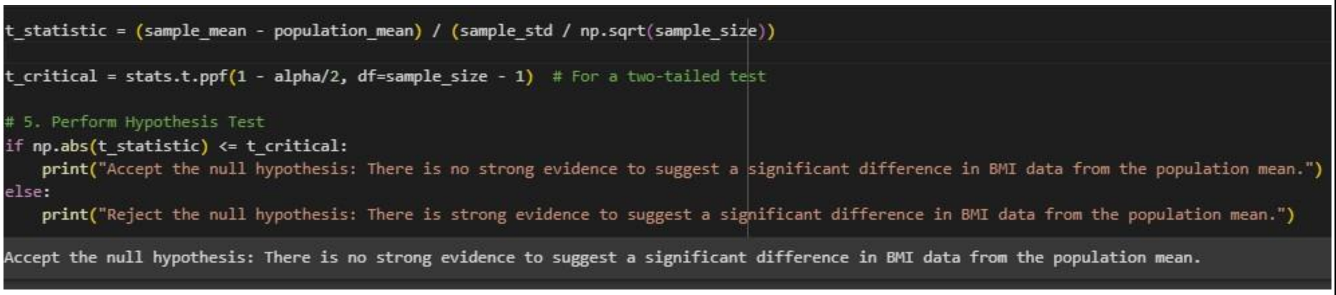
```
sample_mean = sample_df['BMI'].mean()
sample_std = sample_df['BMI'].std()
sample_size = len(sample_df)
```

```
t_critical = stats.t.ppf(1 - alpha/2, df=sample_size - 1)  # For a two-tailed test
margin_of_error = t_critical * (sample_std / np.sqrt(sample_size))

confidence_interval = (sample_mean - margin_of_error, sample_mean + margin_of_error)
```

```
print(f"Confidence Interval (95%): ({confidence_interval[0]:.2f}, {confidence_interval[1]:.2f})")
```

```
Confidence Interval (95%): (23.42, 26.27)
```

And here , we are getting Confidence Interval 95%.



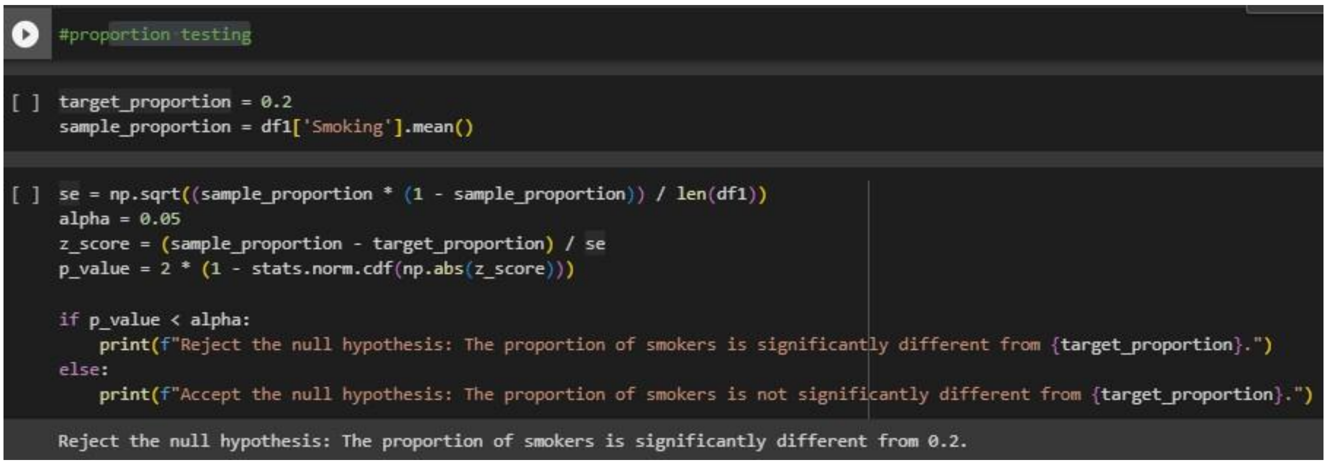
```
t_statistic = (sample_mean - population_mean) / (sample_std / np.sqrt(sample_size))

t_critical = stats.t.ppf(1 - alpha/2, df=sample_size - 1)  # For a two-tailed test

# 5. Perform Hypothesis Test
if np.abs(t_statistic) <= t_critical:
    print("Accept the null hypothesis: There is no strong evidence to suggest a significant difference in BMI data from the population mean.")
else:
    print("Reject the null hypothesis: There is strong evidence to suggest a significant difference in BMI data from the population mean.")
```

```
Accept the null hypothesis: There is no strong evidence to suggest a significant difference in BMI data from the population mean.
```

And this small sample test, we get to know that, it accepts the null hypothesis and There is no strong evidence to suggest a significant difference in BMI data from the population mean.

## Proportion Testing:

Proportion testing assesses population proportions through hypothesis testing. It investigates whether a sample proportion significantly differs from a hypothesized proportion, often using the Z-test or chi-squared test. Researchers calculate test statistics and compare them to critical values or calculate p-values to make decisions regarding the null hypothesis, which represents the expected proportion.



```
#proportion testing

target_proportion = 0.2
sample_proportion = df1['Smoking'].mean()

se = np.sqrt((sample_proportion * (1 - sample_proportion)) / len(df1))
alpha = 0.05
z_score = (sample_proportion - target_proportion) / se
p_value = 2 * (1 - stats.norm.cdf(np.abs(z_score)))

if p_value < alpha:
    print(f"Reject the null hypothesis: The proportion of smokers is significantly different from {target_proportion}.")
else:
    print(f"Accept the null hypothesis: The proportion of smokers is not significantly different from {target_proportion}.")

Reject the null hypothesis: The proportion of smokers is significantly different from 0.2.
```

From the above test, we get to know that, It rejects the null hypothesis: and The proportion of smokers is significantly different from 0.2.

# UNIT -3

## Different types of Test:

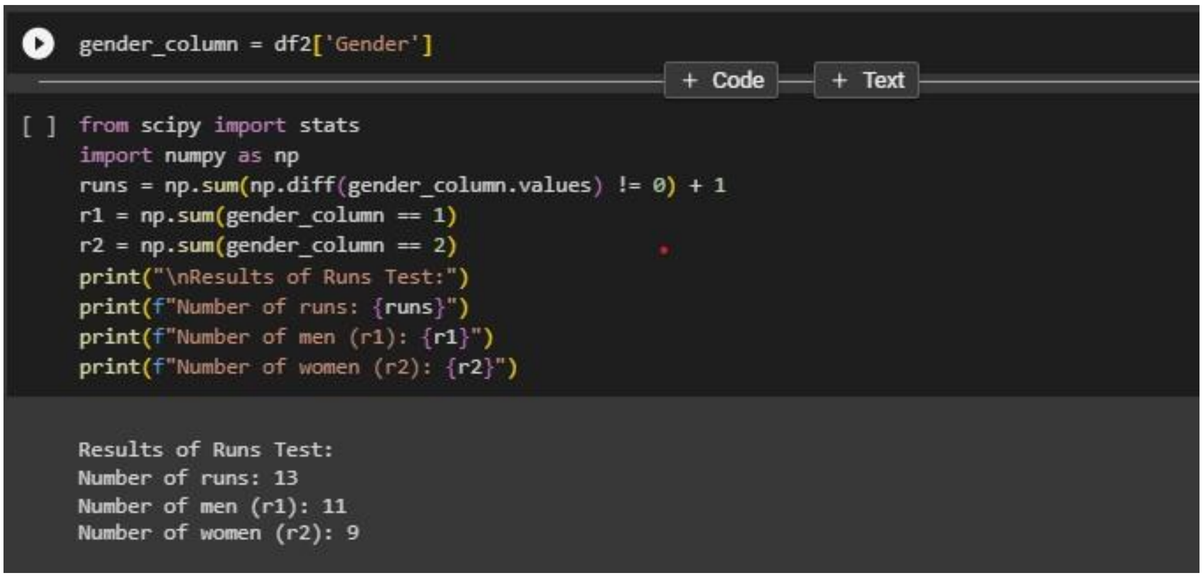
1) ***Runs Test for less than 26:*** A run is a succession of identical letters preceded or followed by a different letter or no letter at all, such as the beginning or end of the succession.

**Ex: A B AAA B B A B B B**

There are six runs, as shown.

A B AAA BB A BBB

1 2 3 4 5 6

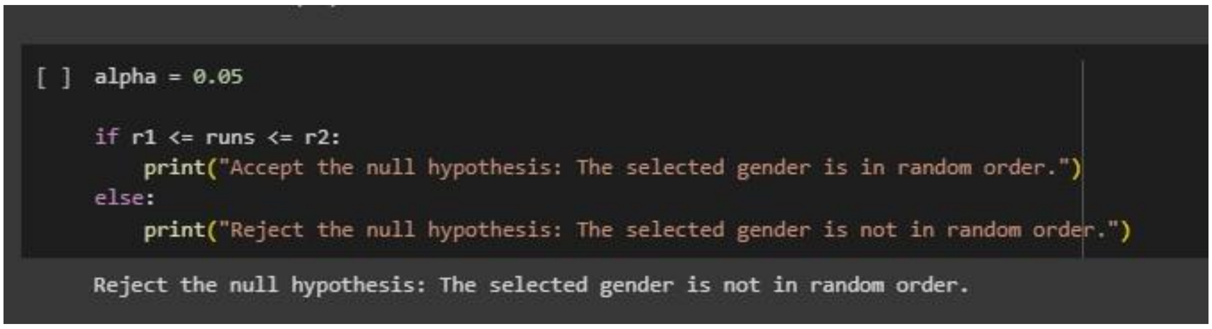


```
gender_column = df2['Gender']
```

```
from scipy import stats
import numpy as np
runs = np.sum(np.diff(gender_column.values) != 0) + 1
r1 = np.sum(gender_column == 1)
r2 = np.sum(gender_column == 2)
print("\nResults of Runs Test:")
print(f"Number of runs: {runs}")
print(f"Number of men (r1): {r1}")
print(f"Number of women (r2): {r2}")
```

```

Results of Runs Test:
Number of runs: 13
Number of men (r1): 11
Number of women (r2): 9
```

Same as above, we are taking the total runs from our dataset in which we have total 11 mens and 9 womens and we get total runs as 13.



```
[ ]  alpha = 0.05

     if r1 <= runs <= r2:
         print("Accept the null hypothesis: The selected gender is in random order.")
     else:
         print("Reject the null hypothesis: The selected gender is not in random order.")

     Reject the null hypothesis: The selected gender is not in random order.
```

Now, we have alpha=0.05

And checking runs whether runs comes between r1 or r2.

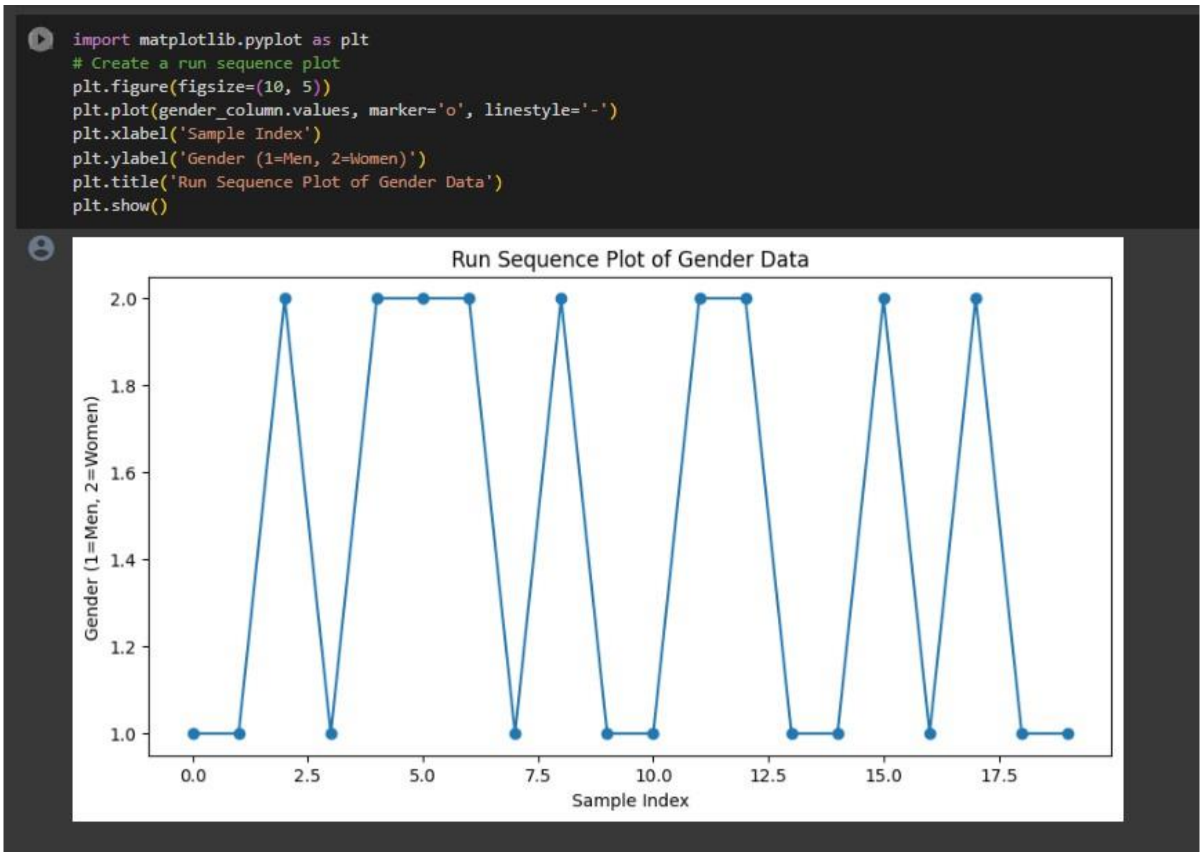
But from the data, its doesn’t comes in between as

Mens:11

Womens:9

Runs:13

So we got output as : *Reject the null hypothesis: The selected gender is not in random order.*



```python
import matplotlib.pyplot as plt
# Create a run sequence plot
plt.figure(figsize=(10, 5))
plt.plot(gender_column.values, marker='o', linestyle='-')
plt.xlabel('Sample Index')
plt.ylabel('Gender (1=Men, 2=Women)')
plt.title('Run Sequence Plot of Gender Data')
plt.show()
```

In the above code snippet, we are plotting the graph of gender Vs runs sequence plot of gender data.

In the first line we are importing matplotlib and then creating the run sequence plot.

(Where 1 = men and 2 = women)

## Advantages:

**Simplicity:** The runs test is relatively simple to understand and apply, making it accessible to individuals without extensive statistical expertise.

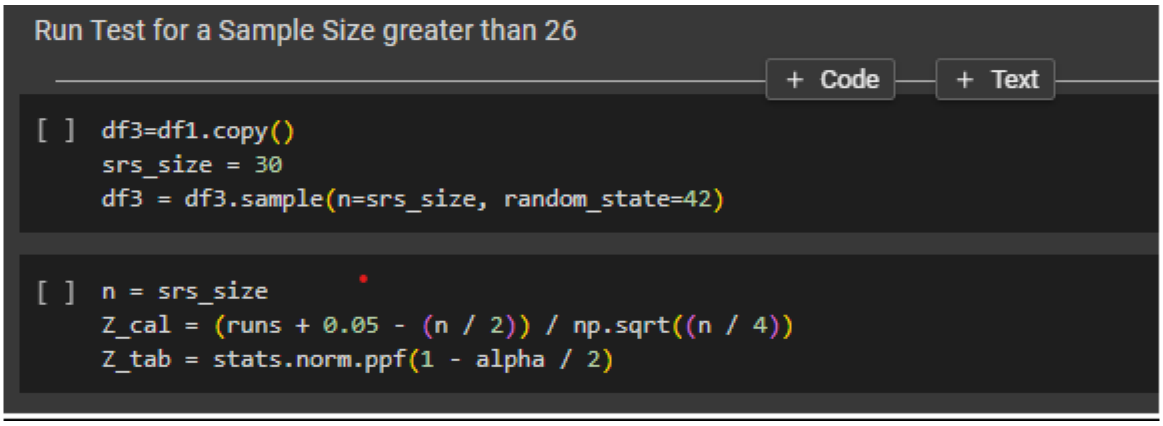
**Quick Analysis: It can provide a quick initial assessment of the randomness or serial correlation in a small dataset.**

## Disadvantages:

**Limited Sensitivity: With a small sample size (less than 26), the runs test may have limited power to detect departures from randomness or serial independence. It might not be able to detect subtle patterns or deviations from randomness effectively.**

**Lower Precision: The results of the runs test on small samples may have wide confidence intervals and lower precision. This makes it challenging to draw strong conclusions about the data's behavior.**

## *(1)Runs test for greater the 26:*



```
[ ] df3=df1.copy()
    srs_size = 30
    df3 = df3.sample(n=srs_size, random_state=42)
```

```
[ ] n = srs_size
    Z_cal = (runs + 0.05 - (n / 2)) / np.sqrt((n / 4))
    Z_tab = stats.norm.ppf(1 - alpha / 2)
```

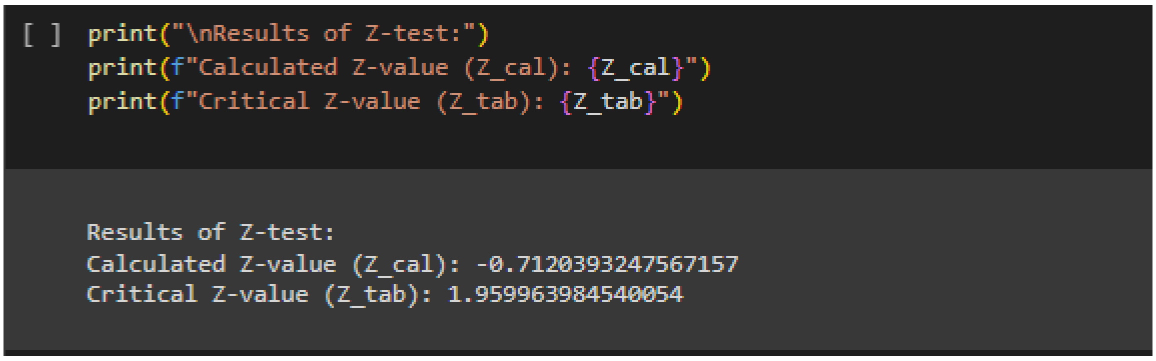
From the above code snippet, we are the taking thedata of more then 26 samples. And putting the formula of runs test more than 26 samples that is ::

$$Z = \frac{(X + 0.05) - (n/2)}{\sqrt{n/2}}$$

**Where X: runs and n: sample size**

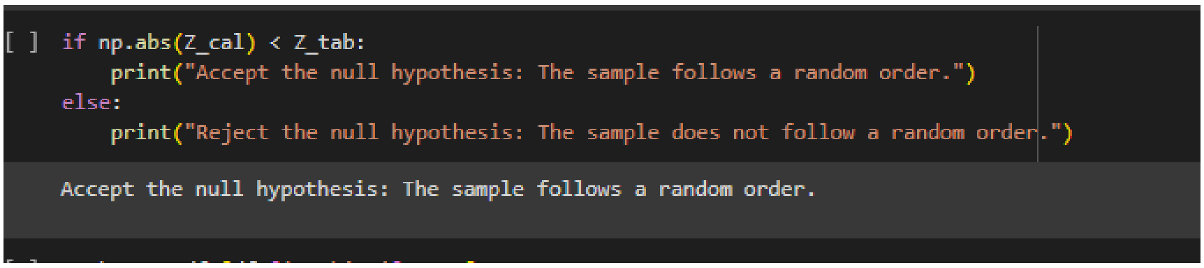
**Here, n=30.**

From the above data calculations we are getting thebelow results:



```
[ ] print("\nResults of Z-test:")
    print(f"Calculated Z-value (Z_cal): {Z_cal}")
    print(f"Critical Z-value (Z_tab): {Z_tab}")


    Results of Z-test:
    Calculated Z-value (Z_cal): -0.7120393247567157
    Critical Z-value (Z_tab): 1.959963984540054
```

**Now, we have to checking whether z_cal < z_tab: if so, then we will accept the result otherwise reject:**



```
[ ] if np.abs(Z_cal) < Z_tab:
        print("Accept the null hypothesis: The sample follows a random order.")
    else:
        print("Reject the null hypothesis: The sample does not follow a random order.")

    Accept the null hypothesis: The sample follows a random order.
```

Now here we are plotting the scatter graph ofFracture Vs smoking status:

**Where, red dot represents smokers and blue dot represents non-smokers.**

## Advantages:

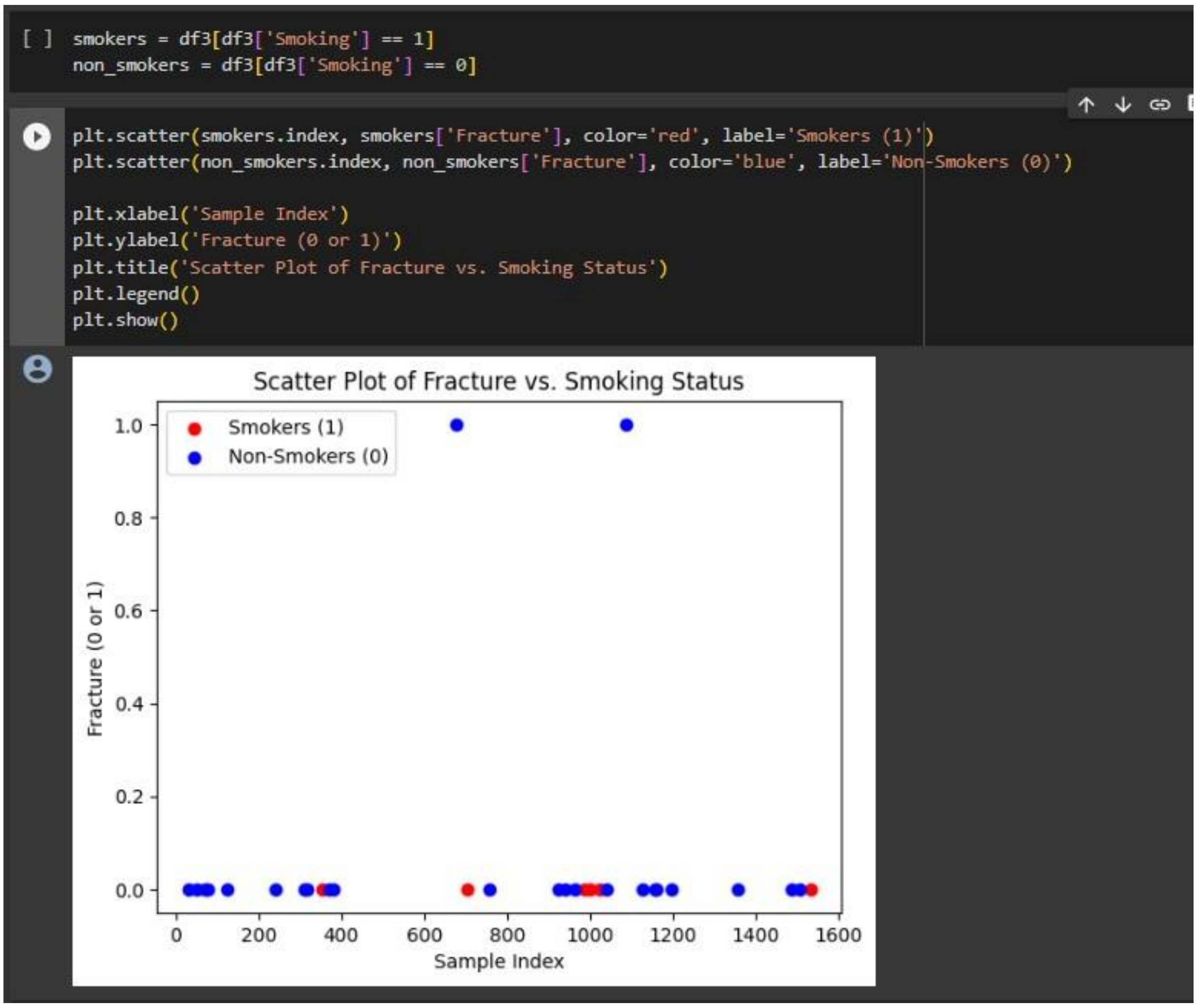
**Increased Sensitivity:** With a larger sample size, the runs test tends to have greater statistical power to detect departures from randomness or serial independence. It can be more effective at identifying subtle patterns or correlations in the data.

**Improved Precision:** Larger sample sizes result in smaller standard errors and more precise estimates of the test statistic, increasing the reliability of the test results.

## Disadvantages:

**Computational Complexity:** The computational burden of the runs test increases with larger sample sizes, especially if performed manually. Statistical software or programming may be necessary for efficient analysis.

**Interpretation Challenges:** While larger sample sizes can increase the sensitivity of the test, it can also lead to statistically significant results that may not have practical significance or meaningful interpretation in real-world scenarios. Careful interpretation is required.



```
[ ] smokers = df3[df3['Smoking'] == 1]
    non_smokers = df3[df3['Smoking'] == 0]
```

```
plt.scatter(smokers.index, smokers['Fracture'], color='red', label='Smokers (1)')
plt.scatter(non_smokers.index, non_smokers['Fracture'], color='blue', label='Non-Smokers (0)')

plt.xlabel('Sample Index')
plt.ylabel('Fracture (0 or 1)')
plt.title('Scatter Plot of Fracture vs. Smoking Status')
plt.legend()
plt.show()
```

***(2) <u>Sign Test-One Sample:</u>*** *The simplest nonparametric test, thesign test for single samples, is used to test the value of a median for a specific sample.*

*When using the sign test, the researcher hypothesizes the specific value for the median of a population; then he or she selects a sample of data and compares each value with the conjectured median.*

*If the data value is above the conjectured median, it is assigned a plus sign.*

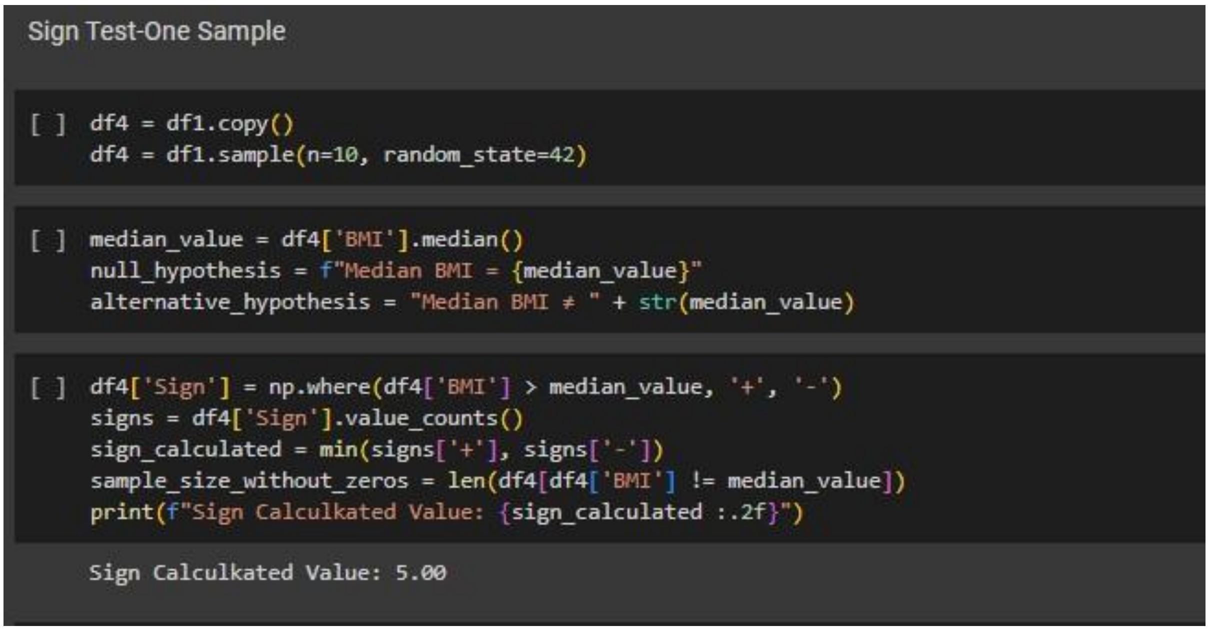
*If it is below the conjectured median, it is assigned a minus sign.*

*And if it is exactly the same as the conjectured median, it is assigned a 0.*

## ***<u>Advantages:</u>***

***<u>Non-parametric:</u>*** *The sign test does not make assumptions about the underlying distribution of the data, making it a useful option when dealing with non-normally distributed data or data with unknown distributions.*

***<u>Robustness</u>:*** *The sign test is robust against outliers or extreme values in the data, as it focuses on the signs of the observations rather than their magnitudes.*

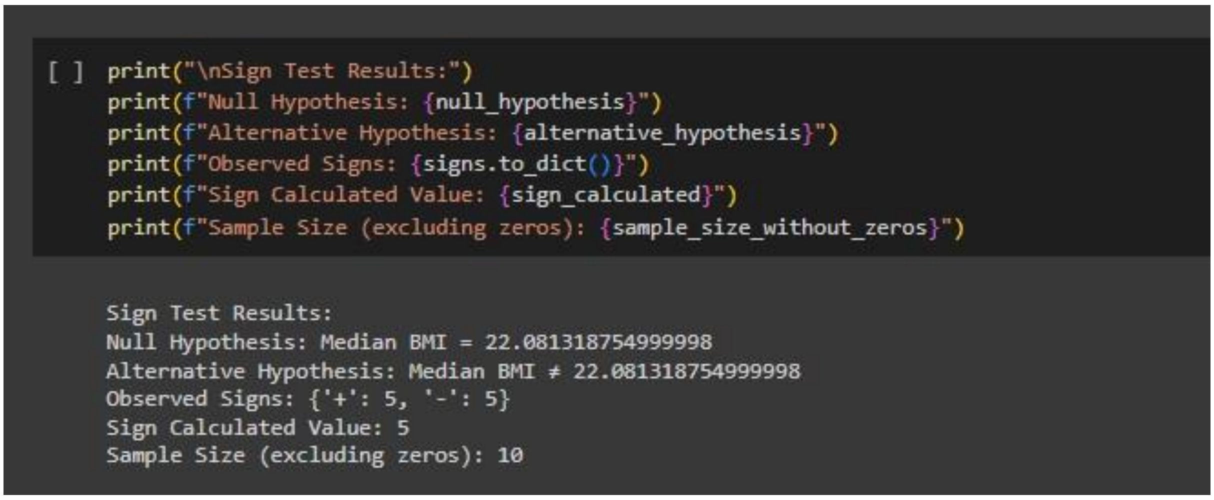

Sign Test-One Sample

```
[ ]  df4 = df1.copy()
     df4 = df1.sample(n=10, random_state=42)
```

```
[ ]  median_value = df4['BMI'].median()
     null_hypothesis = f"Median BMI = {median_value}"
     alternative_hypothesis = "Median BMI ≠ " + str(median_value)
```

```
[ ]  df4['Sign'] = np.where(df4['BMI'] > median_value, '+', '-')
     signs = df4['Sign'].value_counts()
     sign_calculated = min(signs['+'], signs['-'])
     sample_size_without_zeros = len(df4[df4['BMI'] != median_value])
     print(f"Sign Calculkated Value: {sign_calculated :.2f}")

     Sign Calculkated Value: 5.00
```

**Here, in the above snippet of code we are using sign test one sample. And here we are taking sample of n=10. And after calculation, we are getting sign-cal value =5.00**



```
[ ] print("\nSign Test Results:")
    print(f"Null Hypothesis: {null_hypothesis}")
    print(f"Alternative Hypothesis: {alternative_hypothesis}")
    print(f"Observed Signs: {signs.to_dict()}")
    print(f"Sign Calculated Value: {sign_calculated}")
    print(f"Sample Size (excluding zeros): {sample_size_without_zeros}")
```

```
Sign Test Results:
Null Hypothesis: Median BMI = 22.081318754999998
Alternative Hypothesis: Median BMI ≠ 22.081318754999998
Observed Signs: {'+': 5, '-': 5}
Sign Calculated Value: 5
Sample Size (excluding zeros): 10
```

**Now are printing the results of sign-test ,**

**After calculations we are getting ::**

***Null Hypothesis: Median BMI = 22.081318754999998***

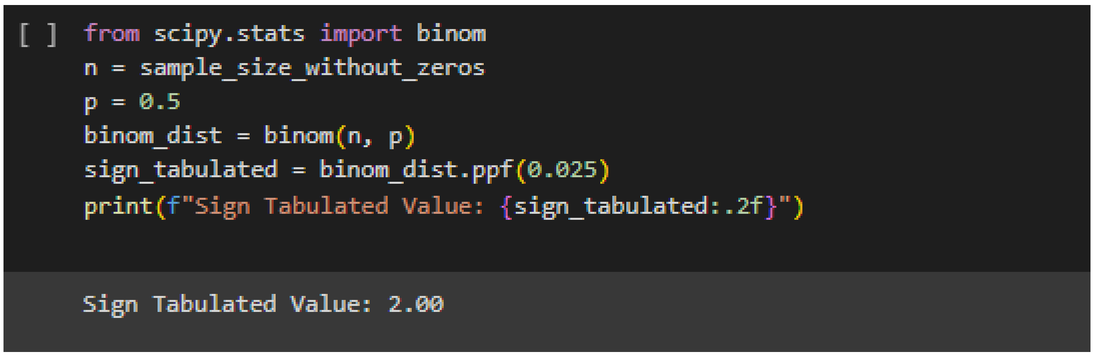
***Alternative Hypothesis: Median BMI ≠ 22.081318754999998***

***Observed Signs: {'+': 5, '-': 5}***

***Sign Calculated Value: 5***

***Sample Size (excluding zeros): 10***

Now, we need to calculate the tabulated value of this test.

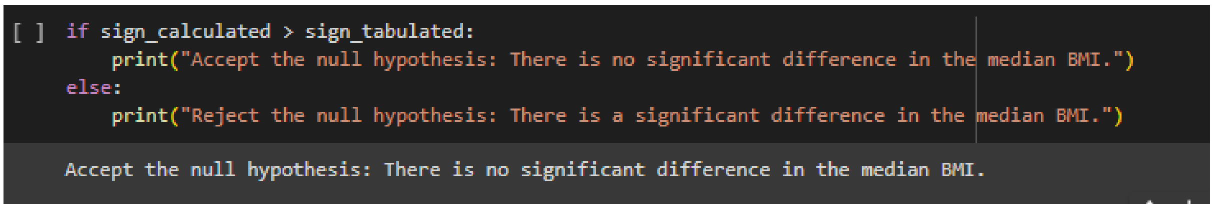


```
[ ] from scipy.stats import binom
    n = sample_size_without_zeros
    p = 0.5
    binom_dist = binom(n, p)
    sign_tabulated = binom_dist.ppf(0.025)
    print(f"Sign Tabulated Value: {sign_tabulated:.2f}")
```

```
Sign Tabulated Value: 2.00
```

So after calculations, as you can see that we are getting value=2

And now we check whether sign_cal > sign_tab if so, we accepts the hypothesis.



```
[ ] if sign_calculated > sign_tabulated:
        print("Accept the null hypothesis: There is no significant difference in the median BMI.")
    else:
        print("Reject the null hypothesis: There is a significant difference in the median BMI.")
```

```
Accept the null hypothesis: There is no significant difference in the median BMI.
```

So, you can see that sign_cal =5 and sign_tab=2 so it accepts the hypothesis.

## *(3)* **Wilcoxon Test:**

The Wilcoxon test, also known as the Wilcoxonsigned-rank test, is a non-parametric statistical test used to determine ifthere is a significant difference between two related groups or paired observations. It is particularly useful when the data does not meet the
assumptions of normality required for parametric tests like the t-test.

## **Advantages:**

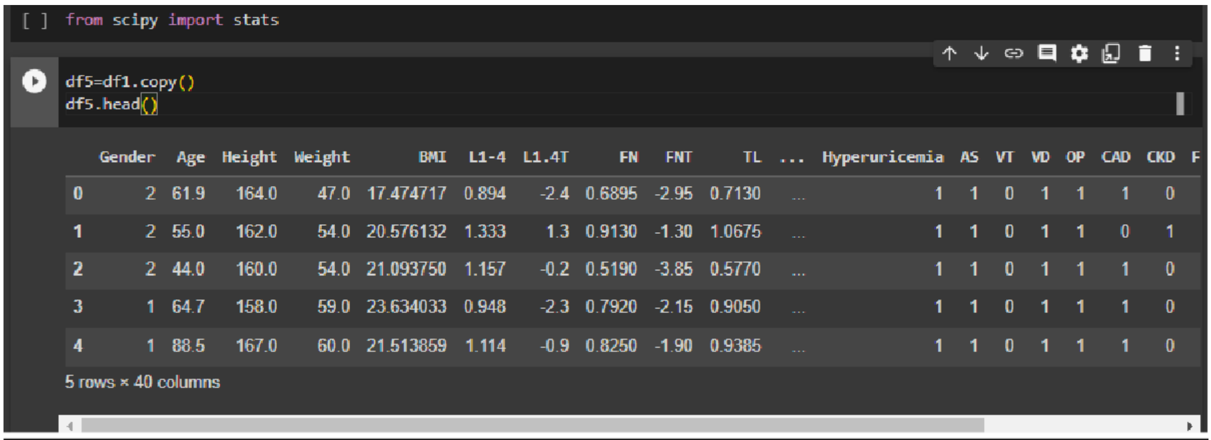
**Non-parametric:** The Wilcoxon test does not rely on assumptions about the underlying data distribution, making it applicable to a wide range of data types, including non-normally distributed data.

**Paired Data:** It is designed for paired data, where each observation in one group corresponds to a related observation in the other group. This makes it suitable for before-and-after studies or matched pairs.

## **Disadvantages:**

**Less Power:** The Wilcoxon test is generally less powerful than parametric tests like the paired t-test when the data is approximately normally distributed. It may require a larger sample size to detect the same effect size.

**Loss of Information:** Like other non-parametric tests, the Wilcoxon test only considers the ranks of the data and ignores the actual magnitudes of the differences between paired observations.



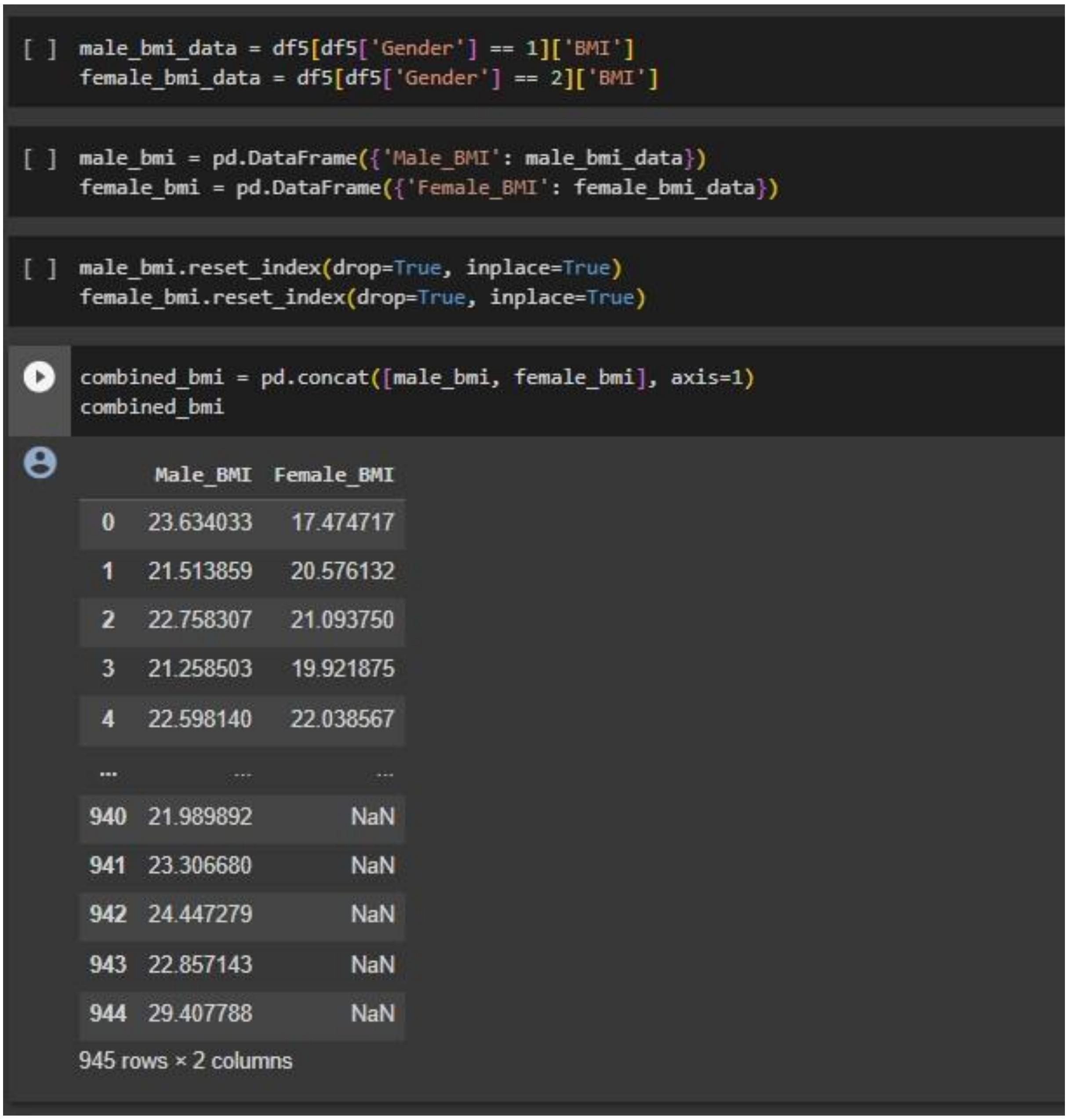
```
[ ] from scipy import stats
```

```
df5=df1.copy()
df5.head()
```

```
   Gender  Age  Height  Weight        BMI   L1-4  L1.4T      FN    FNT      TL  ...  Hyperuricemia  AS  VT  VD  OP  CAD  CKD  F
0       2  61.9  164.0    47.0  17.474717  0.894   -2.4  0.6895  -2.95  0.7130  ...              1   1   0   1   1    1    0
1       2  55.0  162.0    54.0  20.576132  1.333    1.3  0.9130  -1.30  1.0675  ...              1   1   0   1   1    0    1
2       2  44.0  160.0    54.0  21.093750  1.157   -0.2  0.5190  -3.85  0.5770  ...              1   1   0   1   1    1    0
3       1  64.7  158.0    59.0  23.634033  0.948   -2.3  0.7920  -2.15  0.9050  ...              1   1   0   1   1    1    0
4       1  88.5  167.0    60.0  21.513859  1.114   -0.9  0.8250  -1.90  0.9385  ...              1   1   0   1   1    1    0
5 rows × 40 columns
```

Here, we are taking the sample of 5 people.

Now we are trying to fetch the Male_BMI and Female_BMI.



```
[ ] male_bmi_data = df5[df5['Gender'] == 1]['BMI']
    female_bmi_data = df5[df5['Gender'] == 2]['BMI']
```

```
[ ] male_bmi = pd.DataFrame({'Male_BMI': male_bmi_data})
    female_bmi = pd.DataFrame({'Female_BMI': female_bmi_data})
```

```
[ ] male_bmi.reset_index(drop=True, inplace=True)
    female_bmi.reset_index(drop=True, inplace=True)
```

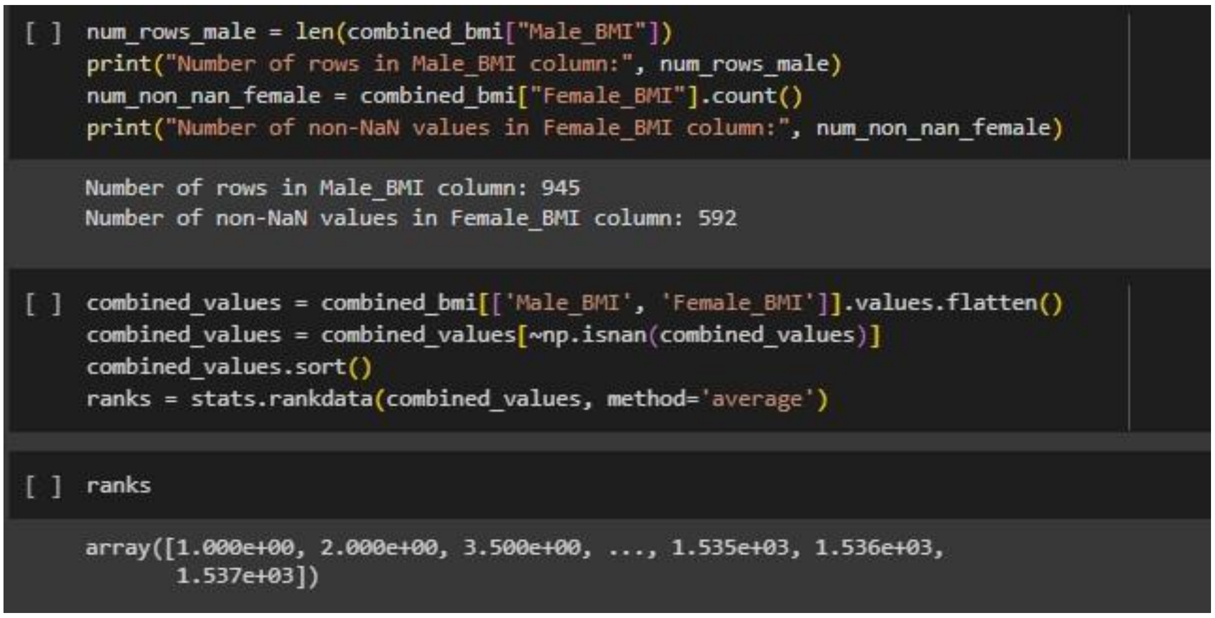
```
combined_bmi = pd.concat([male_bmi, female_bmi], axis=1)
combined_bmi
```

```
      Male_BMI  Female_BMI
0    23.634033   17.474717
1    21.513859   20.576132
2    22.758307   21.093750
3    21.258503   19.921875
4    22.598140   22.038567
...        ...         ...
940  21.989892         NaN
941  23.306680         NaN
942  24.447279         NaN
943  22.857143         NaN
944  29.407788         NaN
945 rows × 2 columns
```

Now, according to Wilcoxon , we are merging both the rows of male and female we have male rows =945 and female rows =592.

So min rows =592.

And we are sorting them in decreasing order. As show in below code snippet:



```
[ ] num_rows_male = len(combined_bmi["Male_BMI"])
    print("Number of rows in Male_BMI column:", num_rows_male)
    num_non_nan_female = combined_bmi["Female_BMI"].count()
    print("Number of non-NaN values in Female_BMI column:", num_non_nan_female)

    Number of rows in Male_BMI column: 945
    Number of non-NaN values in Female_BMI column: 592

[ ] combined_values = combined_bmi[['Male_BMI', 'Female_BMI']].values.flatten()
    combined_values = combined_values[~np.isnan(combined_values)]
    combined_values.sort()
    ranks = stats.rankdata(combined_values, method='average')

[ ] ranks

    array([1.000e+00, 2.000e+00, 3.500e+00, ..., 1.535e+03, 1.536e+03,
           1.537e+03])
```

Now using below formulas we calculate sigma, mu , and z_cal and z_tab

$$Z = \frac{R - \mu_R}{\sigma_R} \quad \text{where} \quad \mu_R = \frac{n_1(n_1 + n_2 + 1)}{2} \quad \sigma_R = \sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}}$$

$R$ sum of ranks for smaller sample size ($n_1$ or $n_2$)

$n1$ First sample size

$n2$ Second sample size

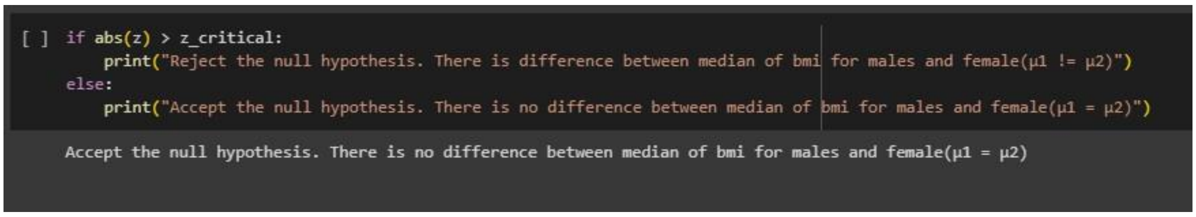
$n1 \geq 10$ and $n2 \geq 10$

So from calculations we are getting ::

*Test Statistic (z): 0.9755922009630267*

*Critical Value (z_critical): 1.959963984540054*

So , now we will check whether abs(z) > z_critical, if so we accepts the hypothesis else reject it.



```
[ ] if abs(z) > z_critical:
        print("Reject the null hypothesis. There is difference between median of bmi for males and female(μ1 != μ2)")
    else:
        print("Accept the null hypothesis. There is no difference between median of bmi for males and female(μ1 = μ2)")

    Accept the null hypothesis. There is no difference between median of bmi for males and female(μ1 = μ2)
```

So from the above, code smippet, we are getting that, abs(z) > z_critical.
Hence we are accepting this is hypothesis.

# UNIT -4
## Goodness of fit

### Chi square test

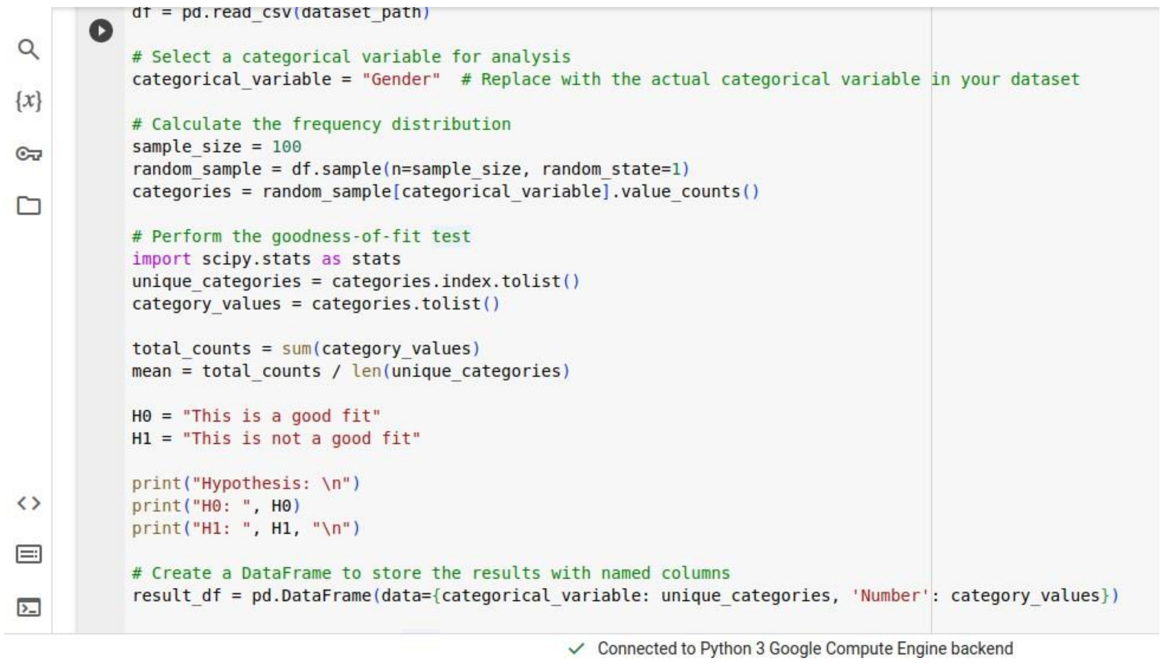
A chi-squared test is a hypothesis test used in the analysis of contingency tables when the sample sizes are large, this test is primarily used to examine whether two categorical variables are independent in influencing the test statistic.

$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Statistical independence:

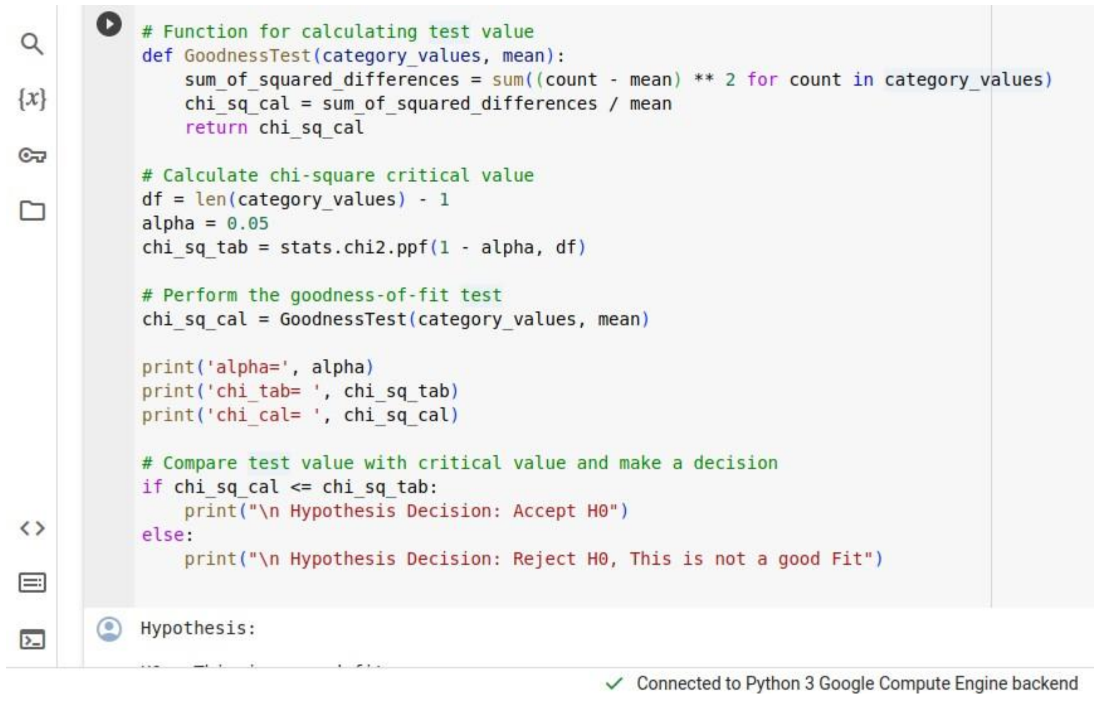
Absence of association between two categories.

Goodness of fit



```
df = pd.read_csv(dataset_path)

# Select a categorical variable for analysis
categorical_variable = "Gender"  # Replace with the actual categorical variable in your dataset

# Calculate the frequency distribution
sample_size = 100
random_sample = df.sample(n=sample_size, random_state=1)
categories = random_sample[categorical_variable].value_counts()

# Perform the goodness-of-fit test
import scipy.stats as stats
unique_categories = categories.index.tolist()
category_values = categories.tolist()

total_counts = sum(category_values)
mean = total_counts / len(unique_categories)

H0 = "This is a good fit"
H1 = "This is not a good fit"

print("Hypothesis: \n")
print("H0: ", H0)
print("H1: ", H1, "\n")

# Create a DataFrame to store the results with named columns
result_df = pd.DataFrame(data={categorical_variable: unique_categories, 'Number': category_values})
```

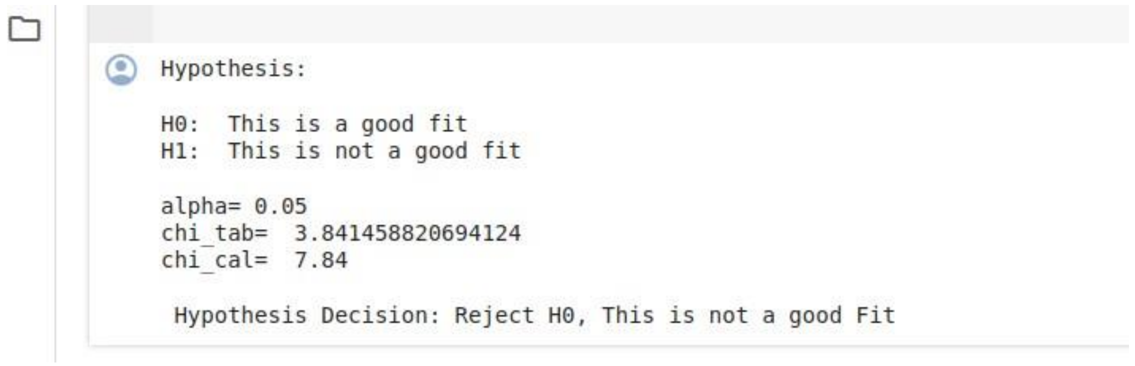
✓ Connected to Python 3 Google Compute Engine backend



```python
# Function for calculating test value
def GoodnessTest(category_values, mean):
    sum_of_squared_differences = sum((count - mean) ** 2 for count in category_values)
    chi_sq_cal = sum_of_squared_differences / mean
    return chi_sq_cal

# Calculate chi-square critical value
df = len(category_values) - 1
alpha = 0.05
chi_sq_tab = stats.chi2.ppf(1 - alpha, df)

# Perform the goodness-of-fit test
chi_sq_cal = GoodnessTest(category_values, mean)

print('alpha=', alpha)
print('chi_tab= ', chi_sq_tab)
print('chi_cal= ', chi_sq_cal)

# Compare test value with critical value and make a decision
if chi_sq_cal <= chi_sq_tab:
    print("\n Hypothesis Decision: Accept H0")
else:
    print("\n Hypothesis Decision: Reject H0, This is not a good Fit")
```

```
Hypothesis:
```

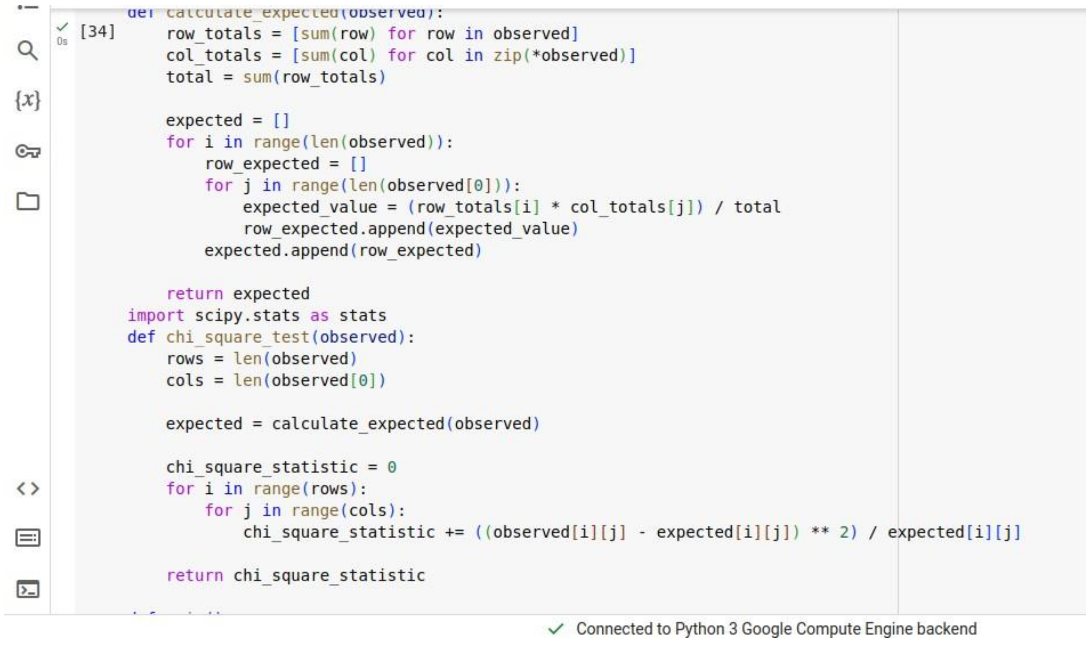
Output



```
Hypothesis:

H0:  This is a good fit
H1:  This is not a good fit

alpha= 0.05
chi_tab=  3.841458820694124
chi_cal=  7.84

 Hypothesis Decision: Reject H0, This is not a good Fit
```

## <u>Independence test</u>

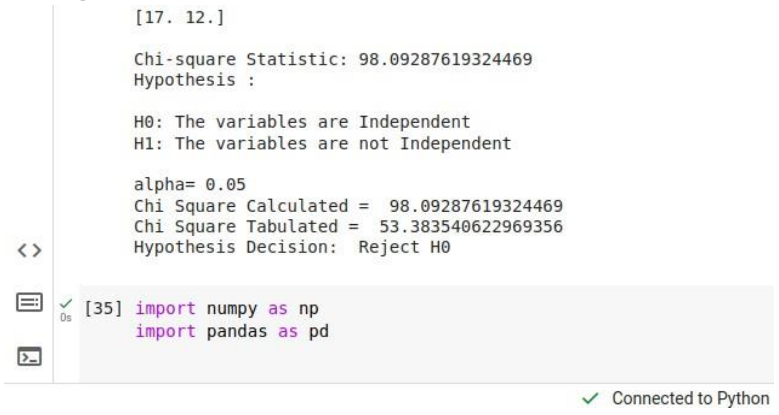
The Chi-square test of independence checks whether two variables are likely to be related or not. We have counts for two categorical or nominal variables. We also have an idea that the two variables are not related. The test gives us a way to decide ifour idea is plausible or not.



```
[34]    def calculate_expected(observed):
            row_totals = [sum(row) for row in observed]
            col_totals = [sum(col) for col in zip(*observed)]
            total = sum(row_totals)

            expected = []
            for i in range(len(observed)):
                row_expected = []
                for j in range(len(observed[0])):
                    expected_value = (row_totals[i] * col_totals[j]) / total
                    row_expected.append(expected_value)
                expected.append(row_expected)

            return expected
        import scipy.stats as stats
        def chi_square_test(observed):
            rows = len(observed)
            cols = len(observed[0])

            expected = calculate_expected(observed)

            chi_square_statistic = 0
            for i in range(rows):
                for j in range(cols):
                    chi_square_statistic += ((observed[i][j] - expected[i][j]) ** 2) / expected[i][j]

            return chi_square_statistic
```

✓ Connected to Python 3 Google Compute Engine backend

## Output:



```
[17. 12.]

Chi-square Statistic: 98.09287619324469
Hypothesis :

H0: The variables are Independent
H1: The variables are not Independent

alpha= 0.05
Chi Square Calculated =  98.09287619324469
Chi Square Tabulated =  53.383540622969356
Hypothesis Decision:  Reject H0
```

```
[35] import numpy as np
     import pandas as pd
```

✓ Connected to Python

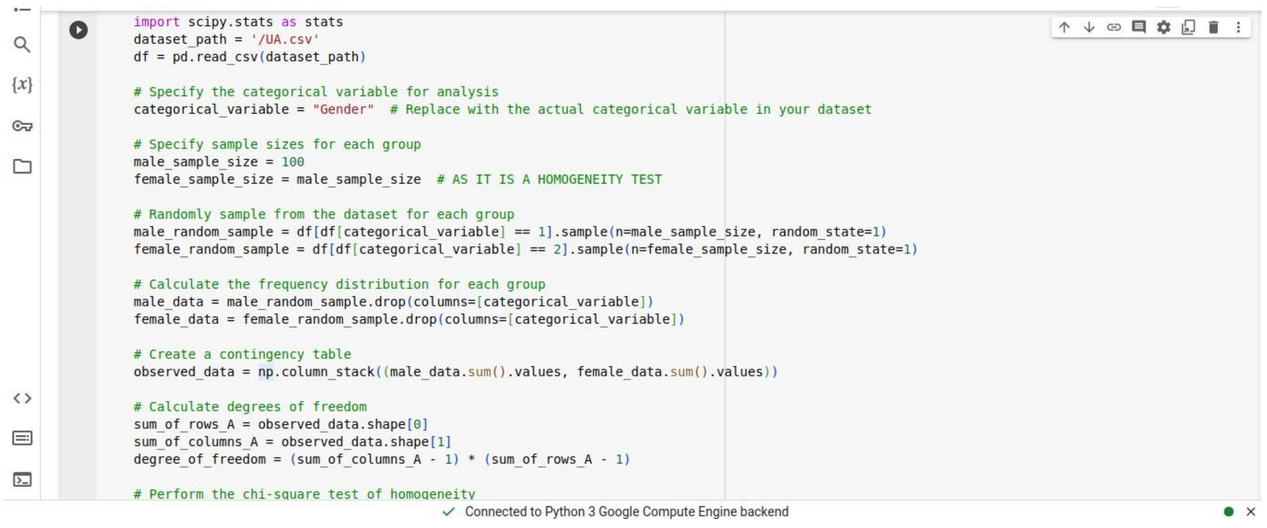
# Homogeneity test using chi square

The chi-square test of homogeneity is the **nonparametric test used in a situation where the dependent variable is categorical**. Data can be presented using a contingency table in which populations and categories of the variable are the row and column labels

$$E_{r,c} = \frac{n_r \cdot n_c}{n}$$

$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

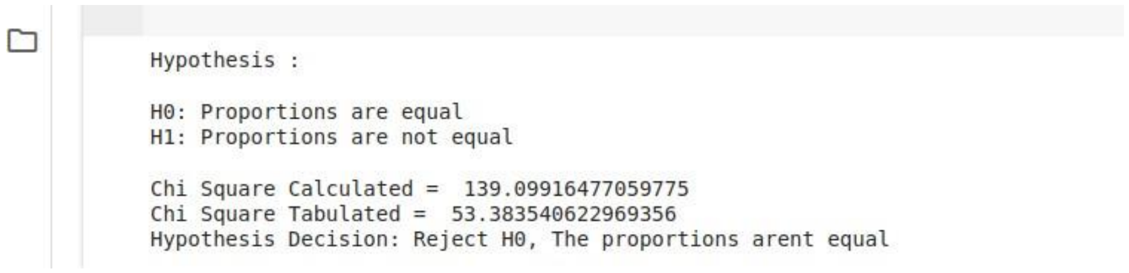
Code:



```
import scipy.stats as stats
dataset_path = '/UA.csv'
df = pd.read_csv(dataset_path)

# Specify the categorical variable for analysis
categorical_variable = "Gender"  # Replace with the actual categorical variable in your dataset

# Specify sample sizes for each group
male_sample_size = 100
female_sample_size = male_sample_size  # AS IT IS A HOMOGENEITY TEST

# Randomly sample from the dataset for each group
male_random_sample = df[df[categorical_variable] == 1].sample(n=male_sample_size, random_state=1)
female_random_sample = df[df[categorical_variable] == 2].sample(n=female_sample_size, random_state=1)

# Calculate the frequency distribution for each group
male_data = male_random_sample.drop(columns=[categorical_variable])
female_data = female_random_sample.drop(columns=[categorical_variable])

# Create a contingency table
observed_data = np.column_stack((male_data.sum().values, female_data.sum().values))

# Calculate degrees of freedom
sum_of_rows_A = observed_data.shape[0]
sum_of_columns_A = observed_data.shape[1]
degree_of_freedom = (sum_of_columns_A - 1) * (sum_of_rows_A - 1)

# Perform the chi-square test of homogeneity
```

Connected to Python 3 Google Compute Engine backend

Output



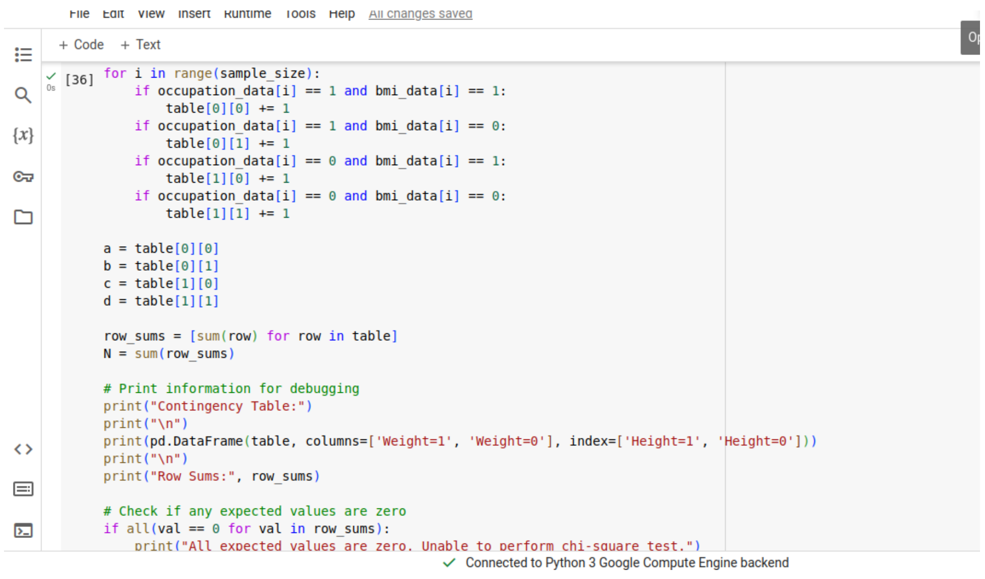
```
Hypothesis :

H0: Proportions are equal
H1: Proportions are not equal

Chi Square Calculated =  139.09916477059775
Chi Square Tabulated =  53.383540622969356
Hypothesis Decision: Reject H0, The proportions arent equal
```

## <u>2x2 contingency table Chi square</u>

The result of the chi-square is compared to the tabled critical value based on df = (R -1)(C -1), where R and C represent thenumber of rows and the number of columns,

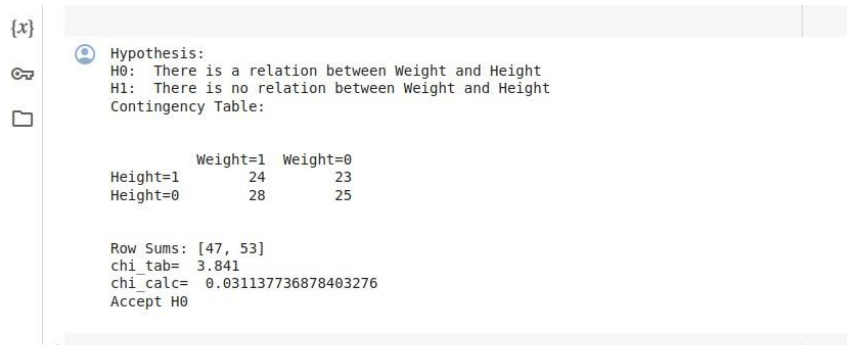
Respectively ntermediate values *are first calculated for each cell based on the observed and expected frequencies in that cell*.

$$\chi^2 = \frac{N(ad-bc)^2}{(a+b)(c+d)(a+c)(b+d)} \sim \chi^2(1)df$$


File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[36]
Connected to Python 3 Google Compute Engine backend

```
for i in range(sample_size):
    if occupation_data[i] == 1 and bmi_data[i] == 1:
        table[0][0] += 1
    if occupation_data[i] == 1 and bmi_data[i] == 0:
        table[0][1] += 1
    if occupation_data[i] == 0 and bmi_data[i] == 1:
        table[1][0] += 1
    if occupation_data[i] == 0 and bmi_data[i] == 0:
        table[1][1] += 1

a = table[0][0]
b = table[0][1]
c = table[1][0]
d = table[1][1]

row_sums = [sum(row) for row in table]
N = sum(row_sums)

# Print information for debugging
print("Contingency Table:")
print("\n")
print(pd.DataFrame(table, columns=['Weight=1', 'Weight=0'], index=['Height=1', 'Height=0']))
print("\n")
print("Row Sums:", row_sums)

# Check if any expected values are zero
if all(val == 0 for val in row_sums):
    print("All expected values are zero. Unable to perform chi-square test.")
```

## Output

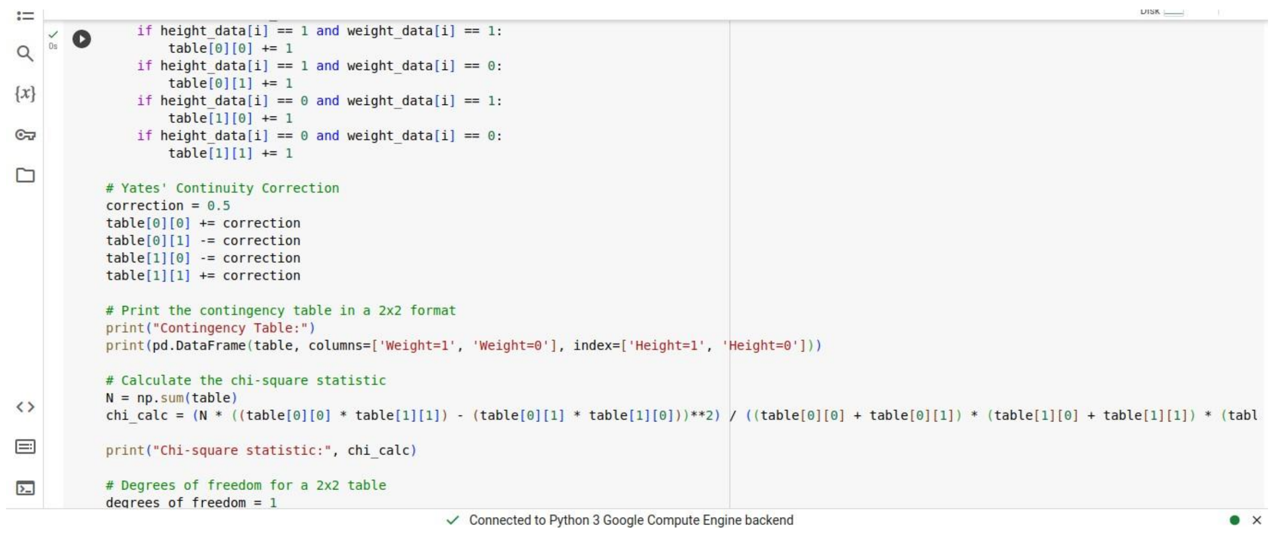


```
Hypothesis:
H0:  There is a relation between Weight and Height
H1:  There is no relation between Weight and Height
Contingency Table:


          Weight=1  Weight=0
Height=1        24        23
Height=0        28        25


Row Sums: [47, 53]
chi_tab=  3.841
chi_calc=  0.031137736878403276
Accept H0
```

## Yates correction test for 2x2

To reduce the error in approximation a correction for continuity that adjusts the formula for Pearson's chi-squared test by **subtracting 0.5 from the difference between each observed value and its expected value in a 2 × 2 contingency table**.

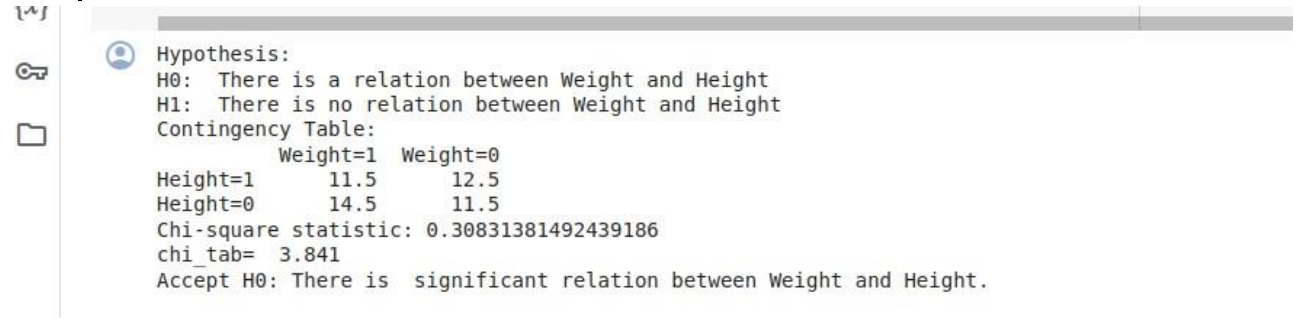
$$\chi^2 = \frac{N\{I(ad-bc)I-(N/2)\}^2}{(a+b)(c+d)(a+c)(b+d)} \sim \chi^2(1)df$$



```
        if height_data[i] == 1 and weight_data[i] == 1:
            table[0][0] += 1
        if height_data[i] == 1 and weight_data[i] == 0:
            table[0][1] += 1
        if height_data[i] == 0 and weight_data[i] == 1:
            table[1][0] += 1
        if height_data[i] == 0 and weight_data[i] == 0:
            table[1][1] += 1

    # Yates' Continuity Correction
    correction = 0.5
    table[0][0] += correction
    table[0][1] -= correction
    table[1][0] -= correction
    table[1][1] += correction

    # Print the contingency table in a 2x2 format
    print("Contingency Table:")
    print(pd.DataFrame(table, columns=['Weight=1', 'Weight=0'], index=['Height=1', 'Height=0']))

    # Calculate the chi-square statistic
    N = np.sum(table)
    chi_calc = (N * ((table[0][0] * table[1][1]) - (table[0][1] * table[1][0]))**2) / ((table[0][0] + table[0][1]) * (table[1][0] + table[1][1]) * (tabl

    print("Chi-square statistic:", chi_calc)

    # Degrees of freedom for a 2x2 table
    degrees of freedom = 1
```

Connected to Python 3 Google Compute Engine backend

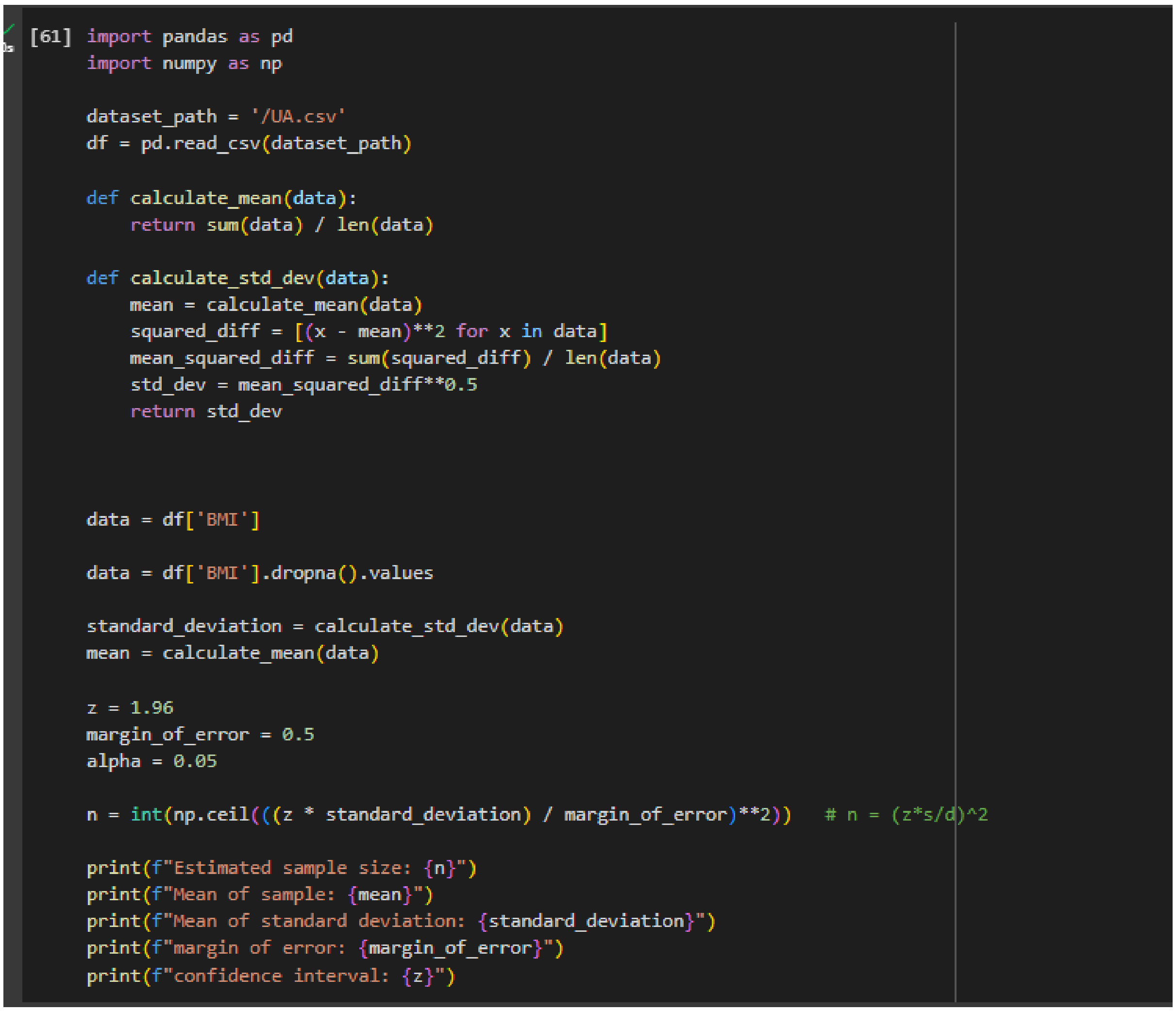
Output



```
Hypothesis:
H0:  There is a relation between Weight and Height
H1:  There is no relation between Weight and Height
Contingency Table:
          Weight=1  Weight=0
Height=1      11.5      12.5
Height=0      14.5      11.5
Chi-square statistic: 0.30831381492439186
chi_tab=  3.841
Accept H0: There is  significant relation between Weight and Height.
```

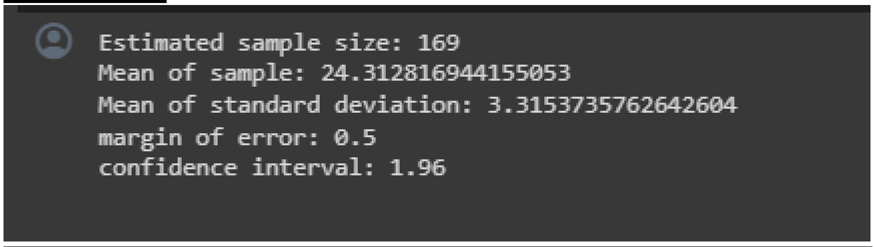
# UNIT-5
## Estimation of Sample Size:

### (1) For One Sample (Single Mean)



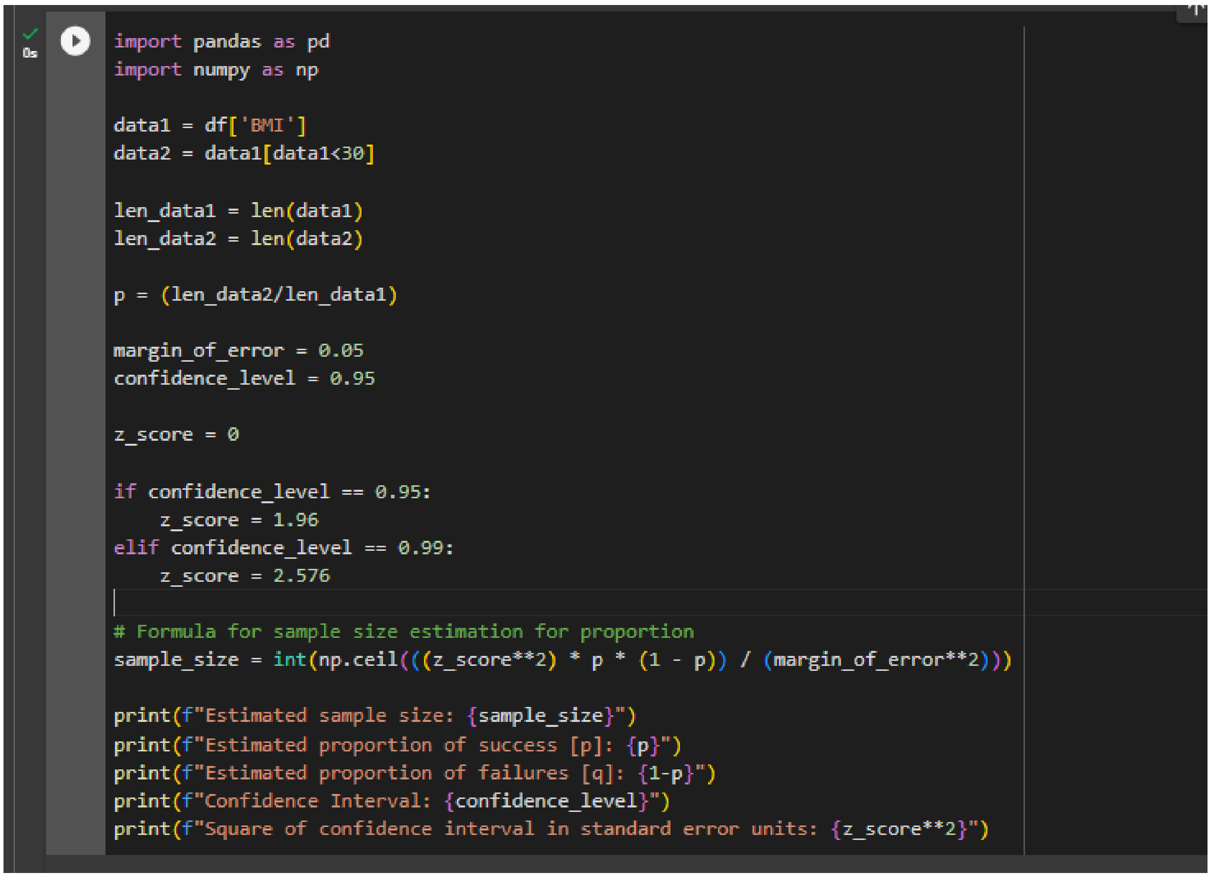
```
[61] import pandas as pd
     import numpy as np

     dataset_path = '/UA.csv'
     df = pd.read_csv(dataset_path)

     def calculate_mean(data):
         return sum(data) / len(data)

     def calculate_std_dev(data):
         mean = calculate_mean(data)
         squared_diff = [(x - mean)**2 for x in data]
         mean_squared_diff = sum(squared_diff) / len(data)
         std_dev = mean_squared_diff**0.5
         return std_dev



     data = df['BMI']

     data = df['BMI'].dropna().values

     standard_deviation = calculate_std_dev(data)
     mean = calculate_mean(data)

     z = 1.96
     margin_of_error = 0.5
     alpha = 0.05

     n = int(np.ceil(((z * standard_deviation) / margin_of_error)**2))   # n = (z*s/d)^2

     print(f"Estimated sample size: {n}")
     print(f"Mean of sample: {mean}")
     print(f"Mean of standard deviation: {standard_deviation}")
     print(f"margin of error: {margin_of_error}")
     print(f"confidence interval: {z}")
```

OutPut:



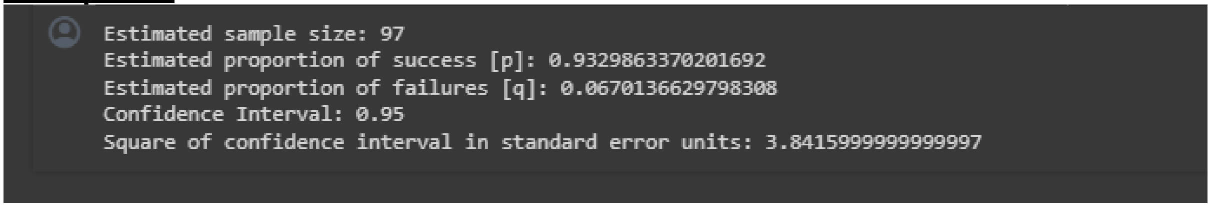
```
Estimated sample size: 169
Mean of sample: 24.312816944155053
Mean of standard deviation: 3.3153735762642604
margin of error: 0.5
confidence interval: 1.96
```

### (2) Sample Size for Proportion:



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

data1 = df['BMI']
data2 = data1[data1<30]

len_data1 = len(data1)
len_data2 = len(data2)

p = (len_data2/len_data1)

margin_of_error = 0.05
confidence_level = 0.95

z_score = 0

if confidence_level == 0.95:
    z_score = 1.96
elif confidence_level == 0.99:
    z_score = 2.576

# Formula for sample size estimation for proportion
sample_size = int(np.ceil(((z_score**2) * p * (1 - p)) / (margin_of_error**2)))

print(f"Estimated sample size: {sample_size}")
print(f"Estimated proportion of success [p]: {p}")
print(f"Estimated proportion of failures [q]: {1-p}")
print(f"Confidence Interval: {confidence_level}")
print(f"Square of confidence interval in standard error units: {z_score**2}")
```

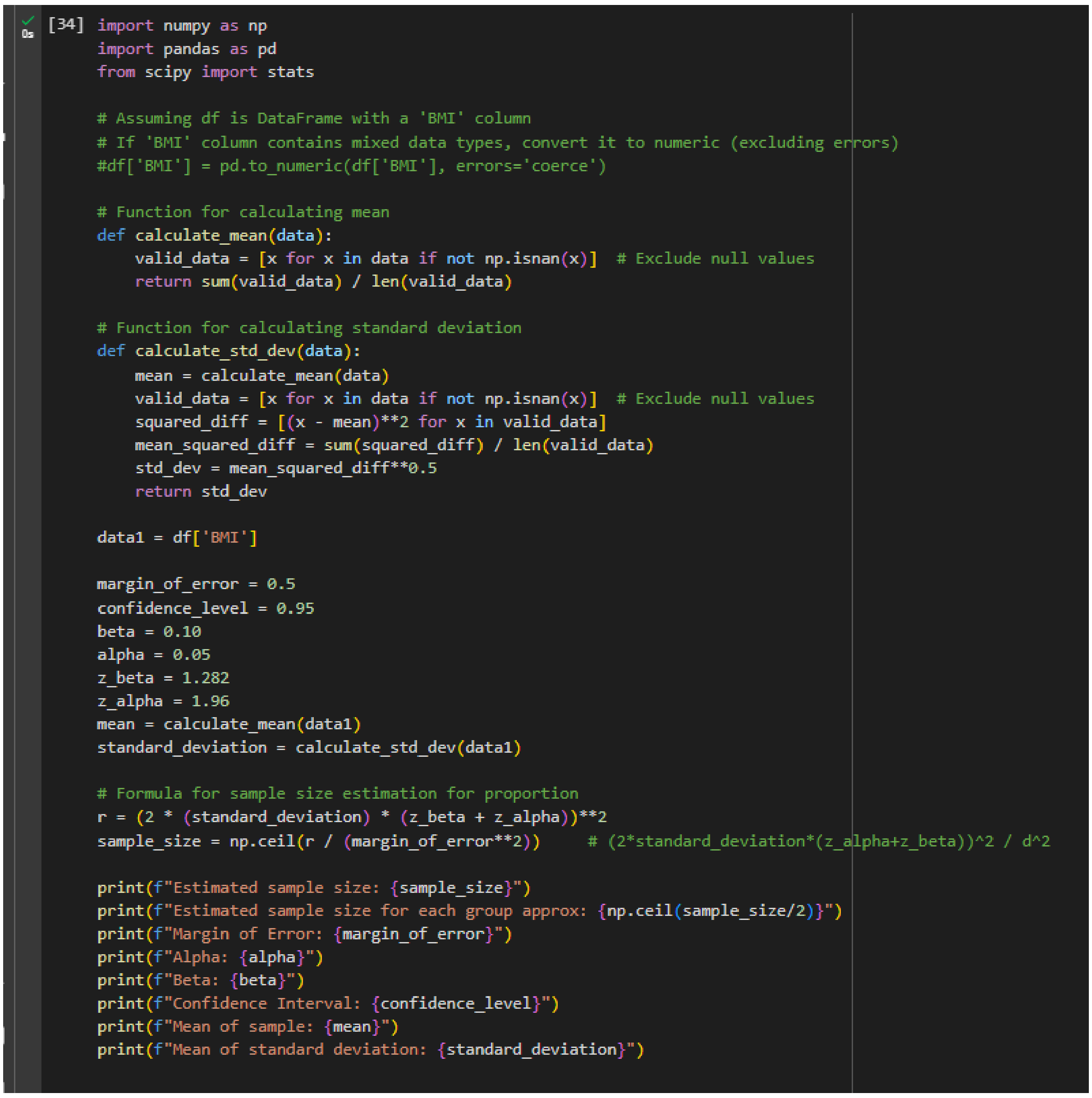
<u>Output::</u>



```
Estimated sample size: 97
Estimated proportion of success [p]: 0.9329863370201692
Estimated proportion of failures [q]: 0.0670136629798308
Confidence Interval: 0.95
Square of confidence interval in standard error units: 3.8415999999999997
```

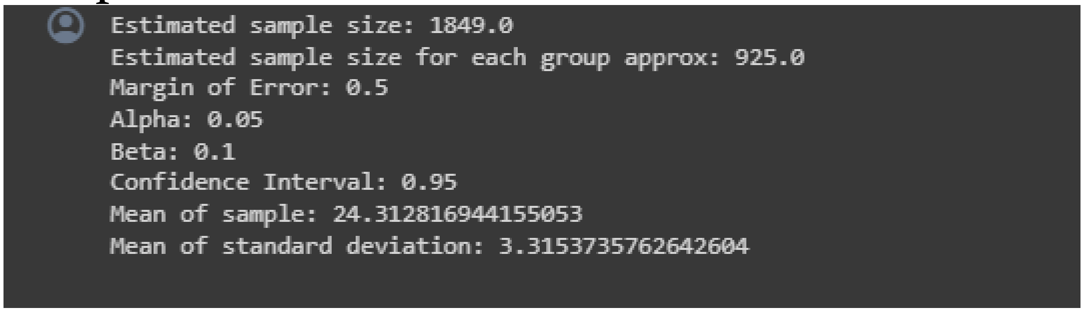
## <u>Sample Size for Two Samples:</u>

### (1) <u>For Two Mean</u>



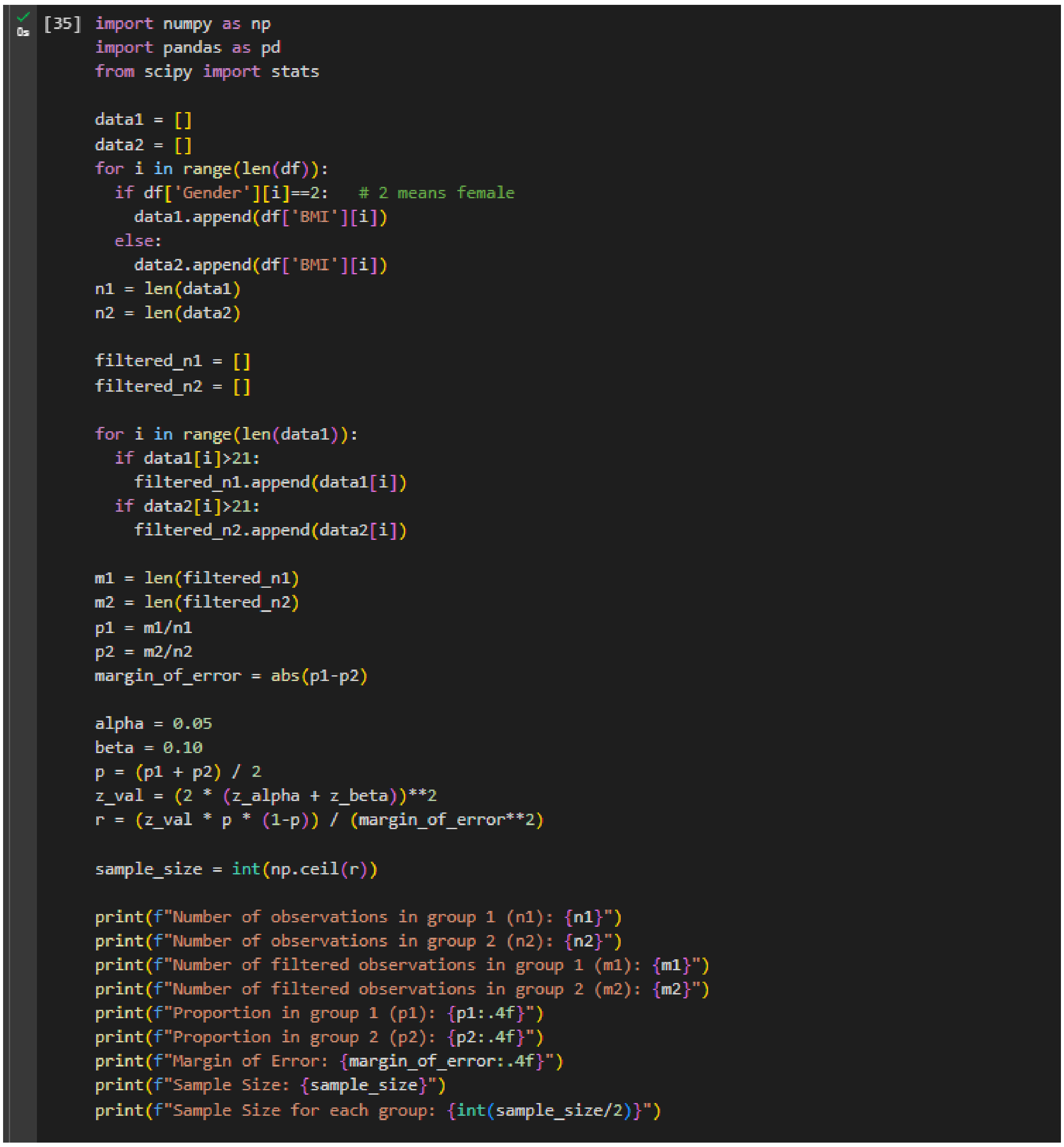
```python
import numpy as np
import pandas as pd
from scipy import stats

# Assuming df is DataFrame with a 'BMI' column
# If 'BMI' column contains mixed data types, convert it to numeric (excluding errors)
#df['BMI'] = pd.to_numeric(df['BMI'], errors='coerce')

# Function for calculating mean
def calculate_mean(data):
    valid_data = [x for x in data if not np.isnan(x)]  # Exclude null values
    return sum(valid_data) / len(valid_data)

# Function for calculating standard deviation
def calculate_std_dev(data):
    mean = calculate_mean(data)
    valid_data = [x for x in data if not np.isnan(x)]  # Exclude null values
    squared_diff = [(x - mean)**2 for x in valid_data]
    mean_squared_diff = sum(squared_diff) / len(valid_data)
    std_dev = mean_squared_diff**0.5
    return std_dev

data1 = df['BMI']

margin_of_error = 0.5
confidence_level = 0.95
beta = 0.10
alpha = 0.05
z_beta = 1.282
z_alpha = 1.96
mean = calculate_mean(data1)
standard_deviation = calculate_std_dev(data1)

# Formula for sample size estimation for proportion
r = (2 * (standard_deviation) * (z_beta + z_alpha))**2
sample_size = np.ceil(r / (margin_of_error**2))     # (2*standard_deviation*(z_alpha+z_beta))^2 / d^2

print(f"Estimated sample size: {sample_size}")
print(f"Estimated sample size for each group approx: {np.ceil(sample_size/2)}")
print(f"Margin of Error: {margin_of_error}")
print(f"Alpha: {alpha}")
print(f"Beta: {beta}")
print(f"Confidence Interval: {confidence_level}")
print(f"Mean of sample: {mean}")
print(f"Mean of standard deviation: {standard_deviation}")
```

## Output:



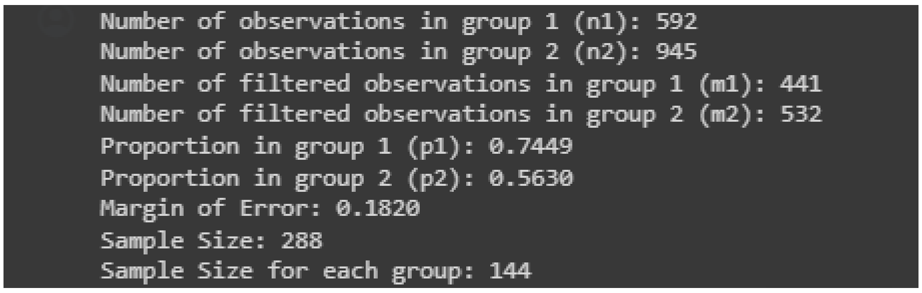
```
Estimated sample size: 1849.0
Estimated sample size for each group approx: 925.0
Margin of Error: 0.5
Alpha: 0.05
Beta: 0.1
Confidence Interval: 0.95
Mean of sample: 24.312816944155053
Mean of standard deviation: 3.3153735762642604
```

## (1) For Two Sample Proportions:



```python
[35] import numpy as np
     import pandas as pd
     from scipy import stats

     data1 = []
     data2 = []
     for i in range(len(df)):
       if df['Gender'][i]==2:   # 2 means female
         data1.append(df['BMI'][i])
       else:
         data2.append(df['BMI'][i])
     n1 = len(data1)
     n2 = len(data2)

     filtered_n1 = []
     filtered_n2 = []

     for i in range(len(data1)):
       if data1[i]>21:
         filtered_n1.append(data1[i])
       if data2[i]>21:
         filtered_n2.append(data2[i])

     m1 = len(filtered_n1)
     m2 = len(filtered_n2)
     p1 = m1/n1
     p2 = m2/n2
     margin_of_error = abs(p1-p2)

     alpha = 0.05
     beta = 0.10
     p = (p1 + p2) / 2
     z_val = (2 * (z_alpha + z_beta))**2
     r = (z_val * p * (1-p)) / (margin_of_error**2)

     sample_size = int(np.ceil(r))

     print(f"Number of observations in group 1 (n1): {n1}")
     print(f"Number of observations in group 2 (n2): {n2}")
     print(f"Number of filtered observations in group 1 (m1): {m1}")
     print(f"Number of filtered observations in group 2 (m2): {m2}")
     print(f"Proportion in group 1 (p1): {p1:.4f}")
     print(f"Proportion in group 2 (p2): {p2:.4f}")
     print(f"Margin of Error: {margin_of_error:.4f}")
     print(f"Sample Size: {sample_size}")
     print(f"Sample Size for each group: {int(sample_size/2)}")
```

OutPut::



```
Number of observations in group 1 (n1): 592
Number of observations in group 2 (n2): 945
Number of filtered observations in group 1 (m1): 441
Number of filtered observations in group 2 (m2): 532
Proportion in group 1 (p1): 0.7449
Proportion in group 2 (p2): 0.5630
Margin of Error: 0.1820
Sample Size: 288
Sample Size for each group: 144
```

THANK YOU!!