Name: Vinayak V Thayil

Roll No: AM.EN.U4CSE21161

Exploring the Dynamics of Diabetes: A Comprehensive Synthetic Dataset Analysis

Below Code:

- Used for downloading and Installing Faker.

In [1]:
```
!pip install faker
```

```
Collecting faker
  Downloading Faker-22.5.0-py3-none-any.whl (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 8.6 MB/s eta 0:00:00
Requirement already satisfied: python-dateutil>=2.4 in /usr/local/lib/python3.10/d
ist-packages (from faker) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages
(from python-dateutil>=2.4->faker) (1.16.0)
Installing collected packages: faker
Successfully installed faker-22.5.0
```

Below Code:

- Importing all necessary Libraries

In [38]:
```
import pandas as pd
import random
import numpy as np
import missingno as msno
import matplotlib.pyplot as plt
import seaborn as sns
```

In [26]:
```
from faker import Faker
from tabulate import tabulate
```

Below code:

- Initialize Faker
- Set seed for reproducibility
- Generate synthetic data with missing values
- Convert the dictionary to a DataFrame
- Save the synthetic dataset with missing values to a **CSV file**

In [52]:
```
fake = Faker()
```

In [53]:
```
random.seed(42)

num_records = 2000
```

In [54]:
```
data = {
    'Patient_ID': [fake.random_int(min=10000, max=99999999) for _ in range(num_recc
    'Age': [random.randint(18, 85) for _ in range(num_records)],
    'Gender': [fake.random_element(['Male', 'Female']) for _ in range(num_records)]
    'BMI': [random.uniform(18.5, 40.0) for _ in range(num_records)],
    'Blood_Pressure': [f"{random.randint(90, 160)}/{random.randint(60, 100)}" for _
    'Glucose_Level': [random.randint(70, 200) if random.random() > 0.1 else np.nan
    'Cholesterol_Level': [fake.random_element(['Normal', 'High']) if random.random(
    'Physical_Activity': [random.uniform(0, 10) for _ in range(num_records)],
    'Diet_Type': [fake.random_element(['Balanced', 'High Carb', 'High Protein']) fo
    'Insulin_Usage': [fake.random_element(['Yes', 'No']) for _ in range(num_records
    'HbA1c_Level': [random.uniform(4.0, 10.0) if random.random() > 0.1 else np.nan
}
```

In [55]:
```
df = pd.DataFrame(data)

df.to_csv('Diabetes.csv', index=False)
```

Below Code:

- To understand the size of the dataset

In [56]:
```
df.shape
```

Out[56]:
```
(2000, 11)
```

Below code:

- Displaying the data from the excel file.

In [57]:
```
df
```

Out[57]:

	Patient_ID	Age	Gender	BMI	Blood_Pressure	Glucose_Level	Cholesterol_Level	Physi
0	23850408	32	Female	33.127351	135/62	164.0	High	
1	35245233	21	Female	31.310011	118/91	183.0	Normal	
2	67524007	53	Female	35.428470	119/64	NaN	Normal	
3	46392654	49	Female	29.798393	123/83	141.0	High	
4	22603531	46	Female	29.855722	119/62	161.0	Normal	
...	...	...	...	...	...	...	...	
1995	80021442	65	Female	30.676681	93/88	104.0	High	
1996	40045059	71	Male	30.664969	105/82	153.0	Normal	
1997	68276694	78	Male	37.252196	138/98	178.0	High	
1998	13190067	45	Male	21.643280	135/84	88.0	NaN	
1999	46645668	49	Male	34.238096	112/60	76.0	High	

2000 rows × 11 columns

Below code:

- To understand more about the features of the dataset

In [31]:
```
df.dtypes
```

Out[31]:
```
Patient_ID             int64
Age                    int64
Gender                object
BMI                  float64
Blood_Pressure        object
Glucose_Level        float64
Cholesterol_Level     object
Physical_Activity    float64
Diet_Type             object
Insulin_Usage         object
HbA1c_Level          float64
dtype: object
```

Below Code:

- Summary statistics of the DataFrame

In [32]:
```
summary_stats_selected = tabulate(df.describe(), headers='keys', tablefmt='fancy_gr
print(summary_stats_selected)
```

```
╒═══════╤═══════════════╤═══════════╤═══════════╤═══════════════╤═══════════
═══╤═══════════════╕
│       │     Patient_ID │       Age │       BMI │   Glucose_Level │   Physical
_Activity │   HbA1c_Level │
╞═══════╪═══════════════╪═══════════╪═══════════╪═══════════════╪═══════════
═══╪═══════════════╡
│ count │   2000        │ 2000      │ 2000      │      1789     │      2000
│    1770       │
├───────┼───────────────┼───────────┼───────────┼───────────────┼───────────
───┼───────────────┤
│ mean  │     4.99813e+07 │   52.0685 │   29.3303 │       135.224 │
4.9823    │       7.05503 │
├───────┼───────────────┼───────────┼───────────┼───────────────┼───────────
───┼───────────────┤
│ std   │     2.91695e+07 │   19.3418 │    6.15315 │        38.1555 │
2.90219   │       1.73724 │
├───────┼───────────────┼───────────┼───────────┼───────────────┼───────────
───┼───────────────┤
│ min   │ 117915        │   18      │   18.5387 │        70     │
9.4794e-05 │       4.0039  │
├───────┼───────────────┼───────────┼───────────┼───────────────┼───────────
───┼───────────────┤
│ 25%   │     2.43796e+07 │   35      │   23.9095 │       102     │
2.45643   │       5.49241 │
├───────┼───────────────┼───────────┼───────────┼───────────────┼───────────
───┼───────────────┤
│ 50%   │     4.96247e+07 │   53      │   29.6082 │       134     │
5.01912   │       7.1125  │
├───────┼───────────────┼───────────┼───────────┼───────────────┼───────────
───┼───────────────┤
│ 75%   │     7.54952e+07 │   69      │   34.5575 │       169     │
7.5596    │       8.57603 │
├───────┼───────────────┼───────────┼───────────┼───────────────┼───────────
───┼───────────────┤
│ max   │     9.99752e+07 │   85      │   39.9877 │       200     │
9.98145   │       9.99066 │
╘═══════╧═══════════════╧═══════════╧═══════════╧═══════════════╧═══════════
═══╧═══════════════╛
```

Below Code:

- Used to count the number of unique values in a DataFrame.

In [33]:
```
df.nunique()
```

Out[33]:
```
Patient_ID           2000
Age                    68
Gender                  2
BMI                  2000
Blood_Pressure       1449
Glucose_Level         131
Cholesterol_Level       2
Physical_Activity    2000
Diet_Type               3
Insulin_Usage           2
HbA1c_Level          1770
dtype: int64
```

Below code:

- Checking the presence of null values

In [34]:
```
missing_values = df.isnull().sum()
print("Missing Values:")
print(missing_values)
```

```
Missing Values:
Patient_ID             0
Age                    0
Gender                 0
BMI                    0
Blood_Pressure         0
Glucose_Level        211
Cholesterol_Level     83
Physical_Activity      0
Diet_Type              0
Insulin_Usage          0
HbA1c_Level          230
dtype: int64
```

Below Code:

- Display the number of missing values before removal.
- Remove row with missing values.
- Display the number of missing values after removal

In [58]:
```
print("Number of missing values before removal:")
print(df.isnull().sum())

df_cleaned = df.dropna()

print("\nNumber of missing values after removal:")
print(df_cleaned.isnull().sum())
```

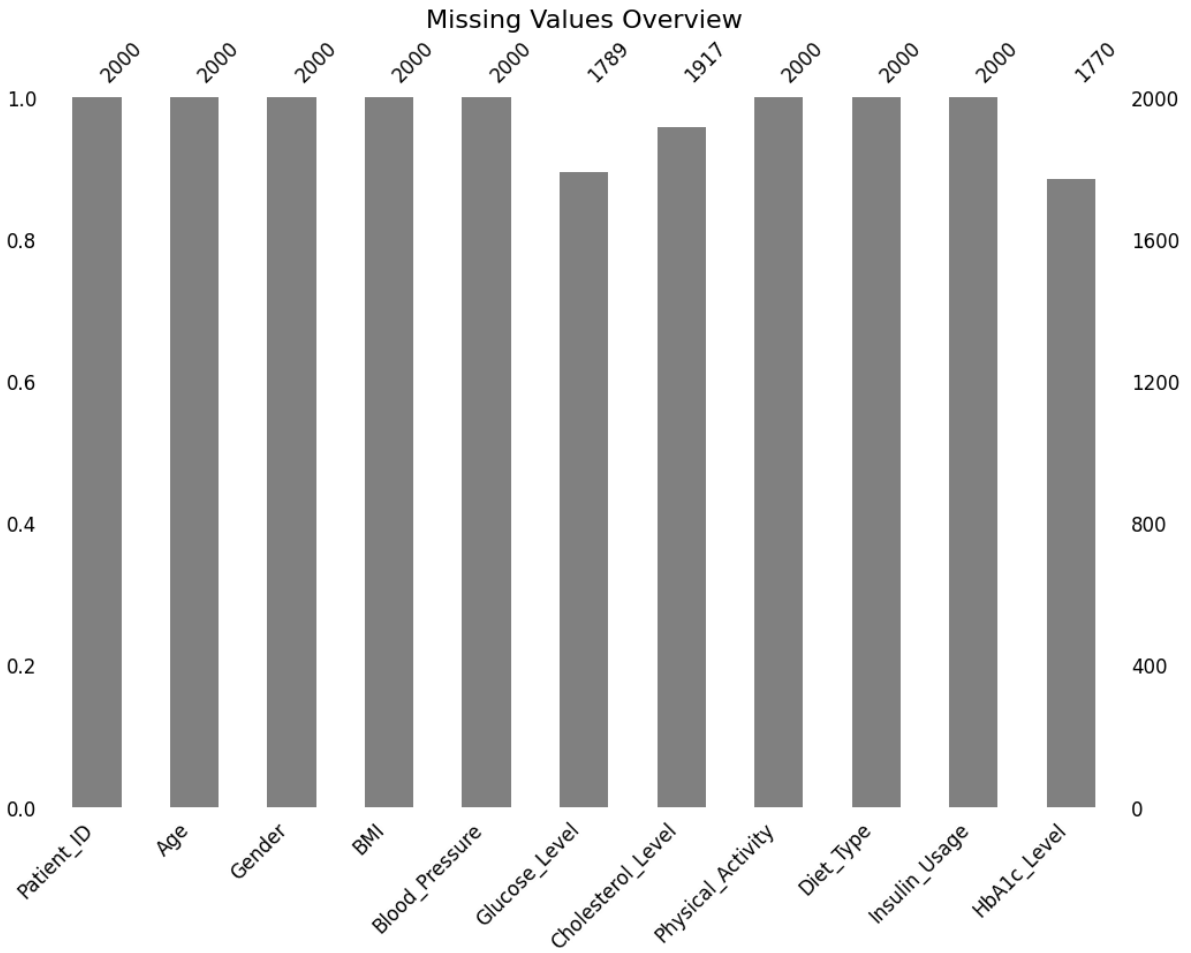
```
Number of missing values before removal:
Patient_ID             0
Age                    0
Gender                 0
BMI                    0
Blood_Pressure         0
Glucose_Level        211
Cholesterol_Level     83
Physical_Activity      0
Diet_Type              0
Insulin_Usage          0
HbA1c_Level          230
dtype: int64

Number of missing values after removal:
Patient_ID           0
Age                  0
Gender               0
BMI                  0
Blood_Pressure       0
Glucose_Level        0
Cholesterol_Level    0
Physical_Activity    0
Diet_Type            0
Insulin_Usage        0
HbA1c_Level          0
dtype: int64
```

Below Code:

- Plot the missing values using **msno.bar** graph

In [36]:
```
# Plot the missing values using missingno.bar
msno.bar(df, figsize=(12, 8), color='grey', fontsize=12, labels=True)
plt.title('Missing Values Overview', fontsize=16)
plt.show()
```



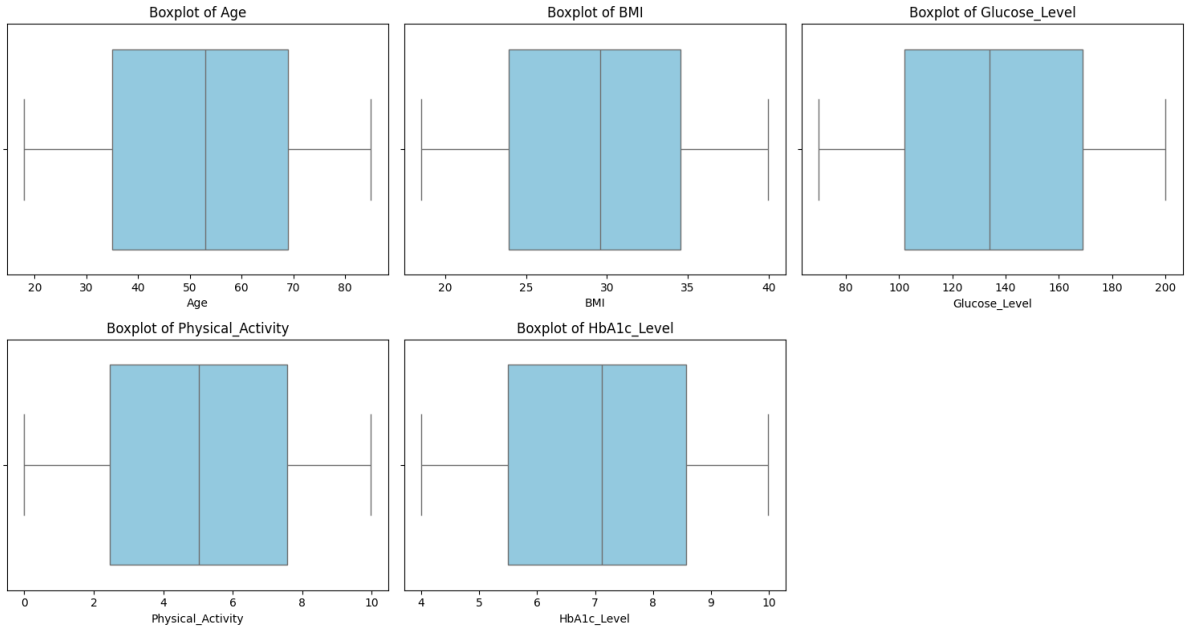
Below Code:

- Create box plots for the numerical columns.

In [59]:
```
numerical_columns = ['Age', 'BMI', 'Glucose_Level', 'Physical_Activity', 'HbA1c_Lev

plt.figure(figsize=(15, 8))
for i, column in enumerate(numerical_columns, 1):
    plt.subplot(2, 3, i)
    sns.boxplot(x=df[column], color='skyblue')
    plt.title(f'Boxplot of {column}')

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



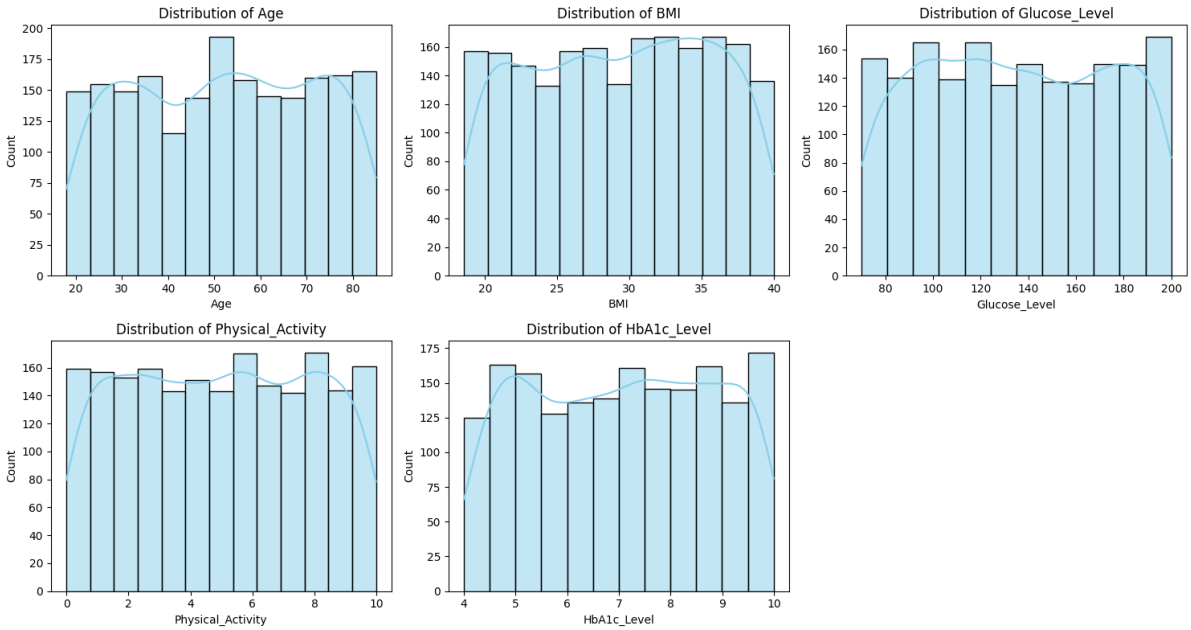
Below code:

- create a set of histograms to visualize the distribution of numerical columns

In [40]:
```
numerical_columns = ['Age', 'BMI', 'Glucose_Level', 'Physical_Activity', 'HbA1c_Lev

plt.figure(figsize=(15, 8))
for i, column in enumerate(numerical_columns, 1):
    plt.subplot(2, 3, i)
    sns.histplot(df[column], kde=True, color='skyblue')
    plt.title(f'Distribution of {column}')

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



Below Code:

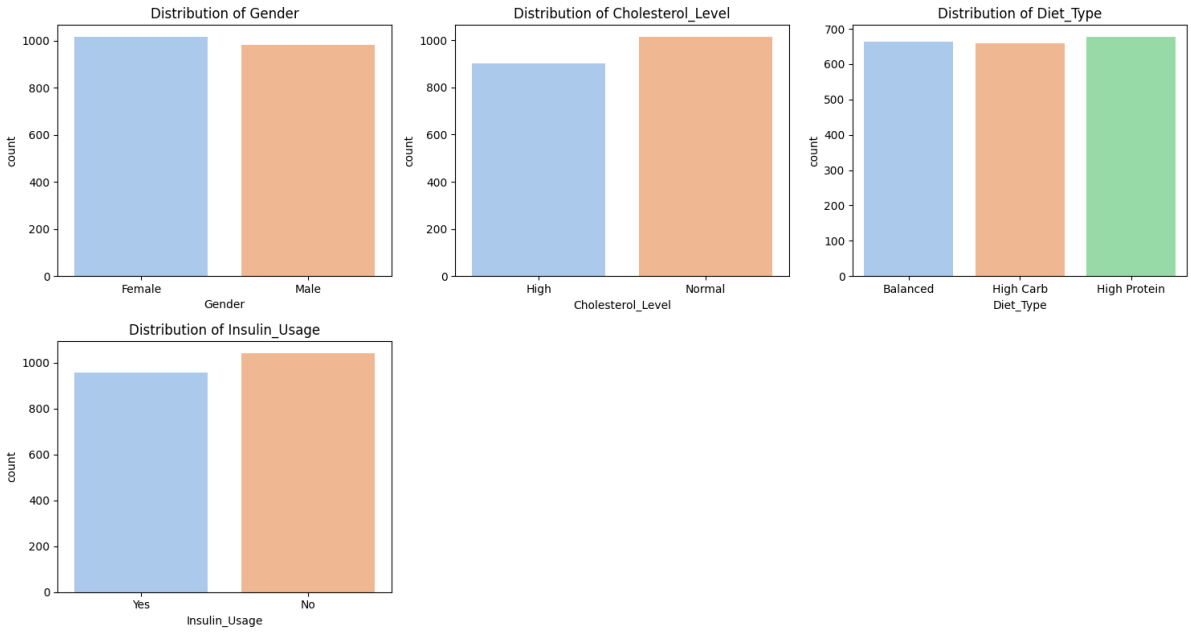
- Used to create bar plots for the categorical columns in a DataFrame.

In [60]:
```
categorical_columns = ['Gender', 'Cholesterol_Level', 'Diet_Type', 'Insulin_Usage']

plt.figure(figsize=(15, 8))
```

```
for i, column in enumerate(categorical_columns, 1):
    plt.subplot(2, 3, i)
    sns.countplot(x=df[column], palette='pastel', hue=df[column], legend=False)
    plt.title(f'Distribution of {column}')

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

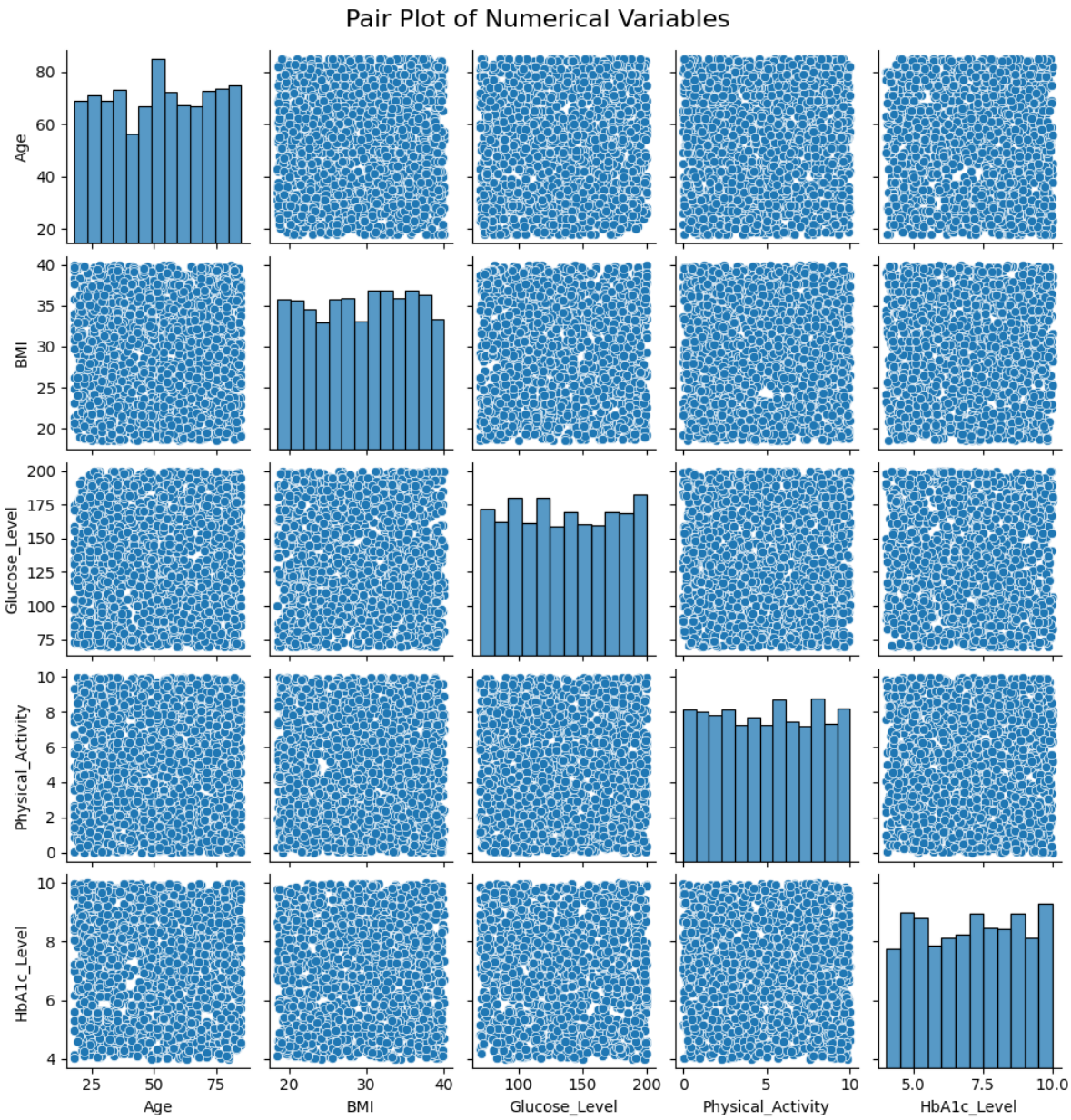


Below Code:

- Used to create a pair plot using Seaborn for a subset of numerical columns in a DataFrame

In [64]:
```
numerical_columns = ['Age', 'BMI', 'Glucose_Level', 'Physical_Activity', 'HbA1c_Lev

sns.pairplot(df[numerical_columns], height=2)
plt.suptitle('Pair Plot of Numerical Variables', y=1.02, fontsize=16)
plt.show()
```



Below Code:

- The code you provided is used to calculate the correlation matrix for numerical columns in a pandas DataFrame (df).
- It then prints the correlation matrix using the tabulate function from an external library (presumably the tabulate library).
- The tabulate function is used to format the correlation matrix as a table for better readability when printing.

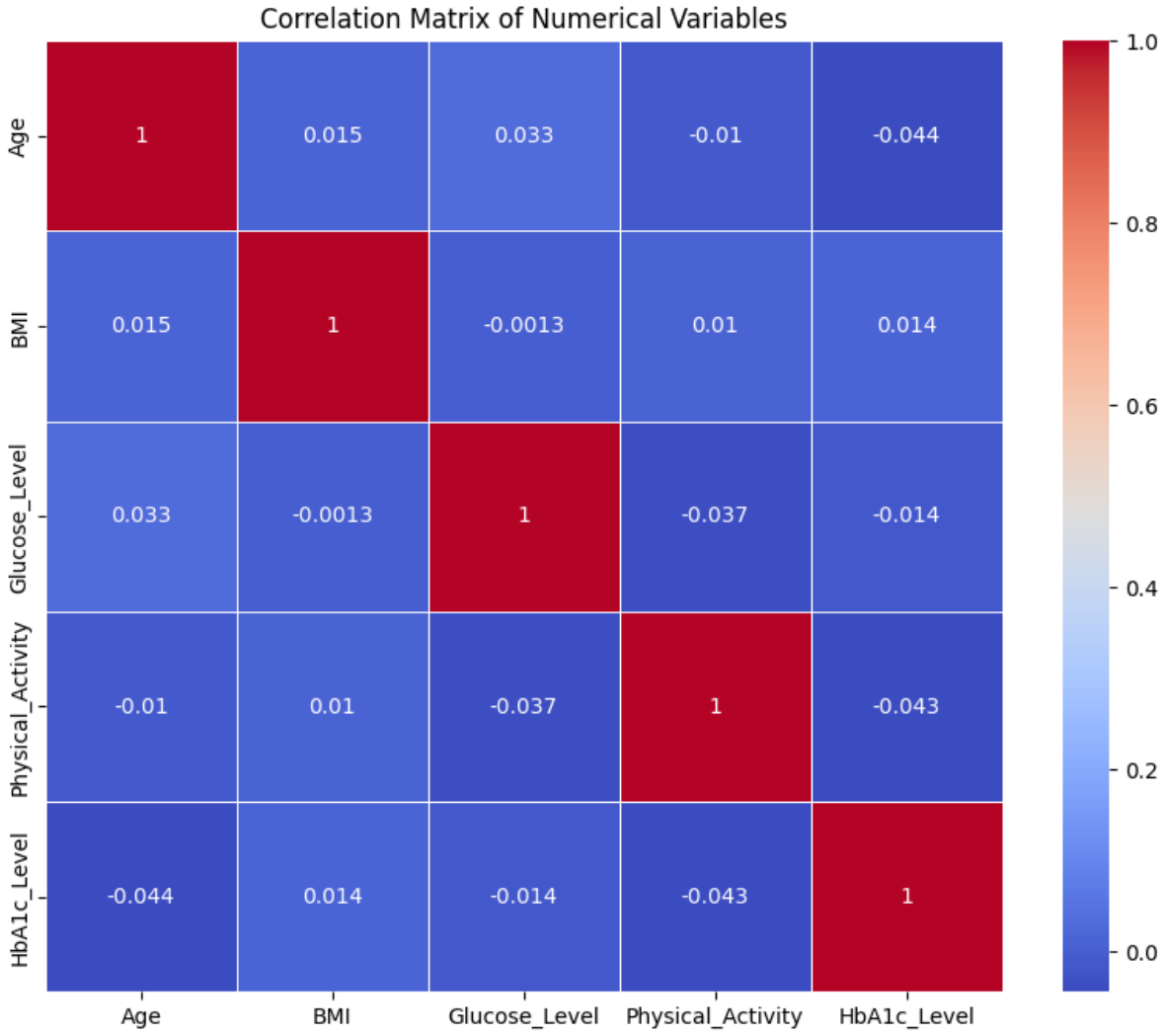
In [61]:
```
correlation_matrix = tabulate(df[numerical_columns].corr(),headers='keys', tablefmt
print(correlation_matrix)
```

	Age	BMI	Glucose_Level	Physical_Activity	HbA1c_Level
Age	1	0.014684	0.0326281	-0.0103207	-0.044441
BMI	0.014684	1	-0.00132464	0.0099931	0.0144518
Glucose_Level	0.0326281	-0.00132464	1	-0.0372564	-0.0137819
Physical_Activity	-0.0103207	0.0099931	-0.0372564	1	-0.0434523
HbA1c_Level	-0.044441	0.0144518	-0.0137819	-0.0434523	1

Below Code:

- Used to create a heatmap of the correlation matrix for numerical variables in a DataFrame using the **seaborn (sns)** and **matplotlib (plt)** libraries.

In [62]:
```
correlation_matrix = df[numerical_columns].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=.5)
plt.title('Correlation Matrix of Numerical Variables')
plt.show()
```



Q1.How does the distribution of ages vary based on insulin usage in the given diabetes dataset?

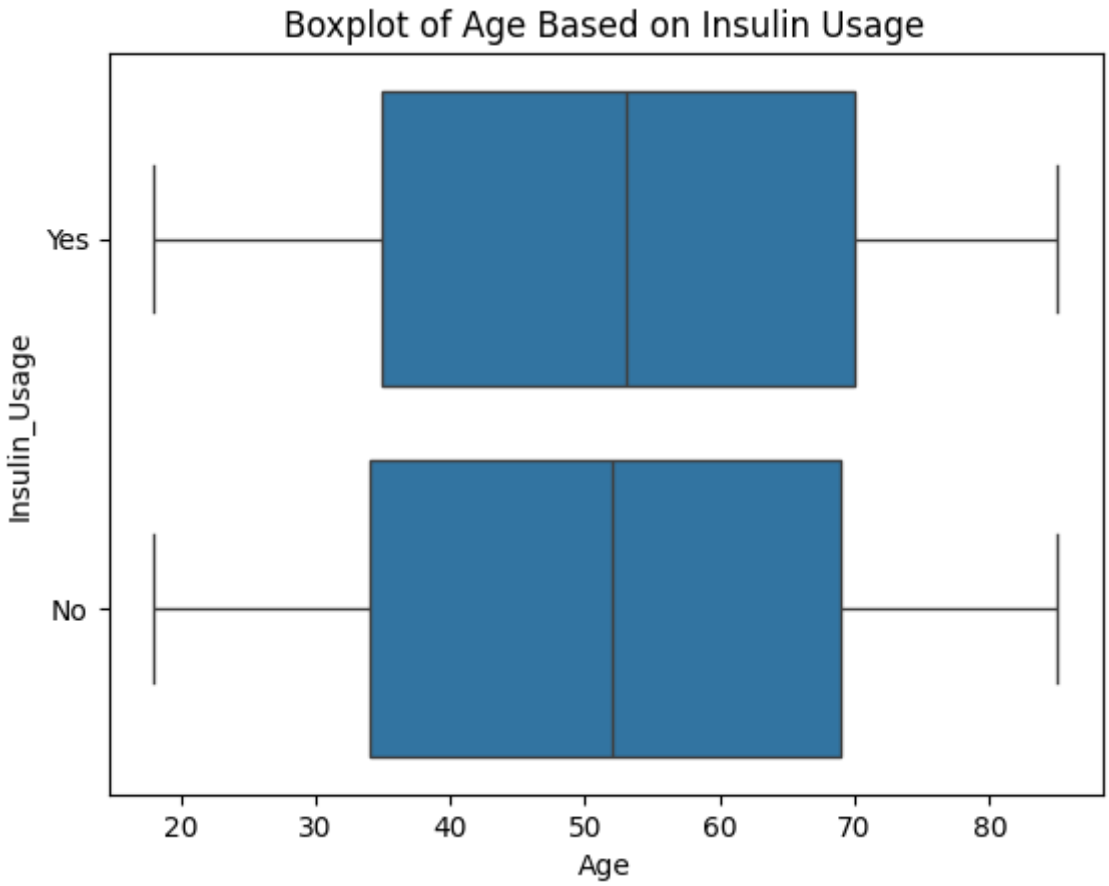
- Observe if the medians of the two groups (Insulin Usage and No Insulin Usage) differ significantly. A substantial difference might indicate an association between age and insulin usage.
- Check for differences in the spread of ages within each group. A wider spread in one group might suggest greater age variability.

Below code:

- Used to create a box plot using Seaborn library

In [18]:
```
sns.boxplot(x='Age',y='Insulin_Usage',data=df)
plt.title('Boxplot of Age Based on Insulin Usage')
```

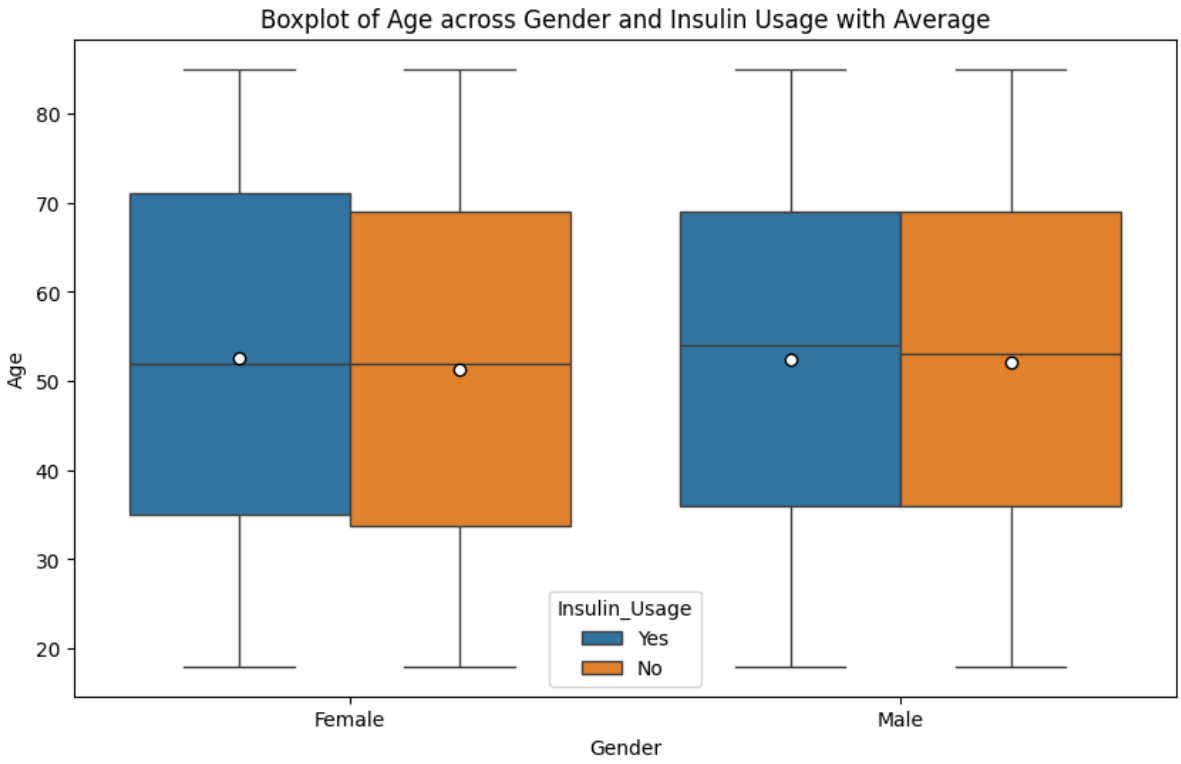
Out[18]:
```
Text(0.5, 1.0, 'Boxplot of Age Based on Insulin Usage')
```



Q2. How does the distribution of ages vary among different genders, considering the use of insulin?

Boxplot enables a comprehensive exploration of age distributions, gender differences, and the impact of insulin usage within the diabetes dataset. Further statistical analysis or subgroup comparisons may provide deeper insights into these observed patterns.

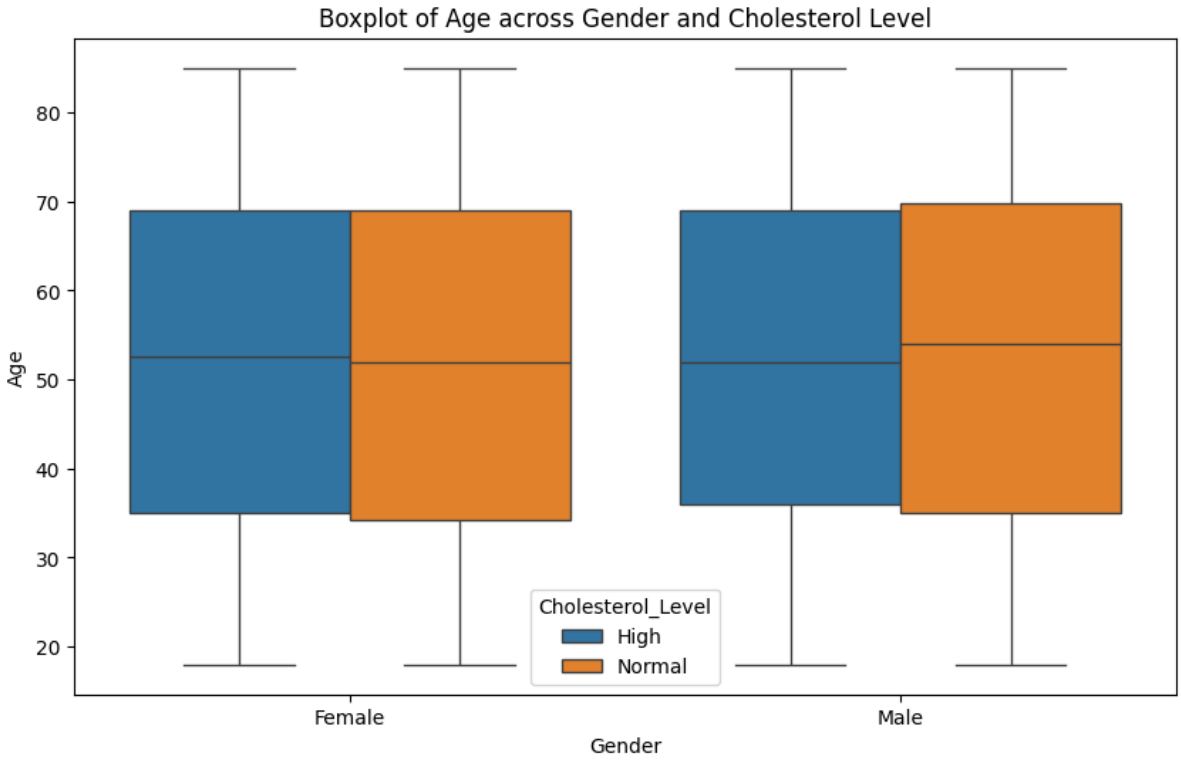
In [19]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(x='Gender', y='Age', hue='Insulin_Usage', data=df, showmeans=True, mear
plt.title('Boxplot of Age across Gender and Insulin Usage with Average')
plt.show()
```



Q3. How does age vary across different genders concerning cholesterol levels?

The boxplot visualization reveals interesting insights into the distribution of age across different genders, considering varying levels of cholesterol within the diabetes dataset. By examining the boxes and whiskers, we can discern the central tendency, spread, and potential outliers in the data.

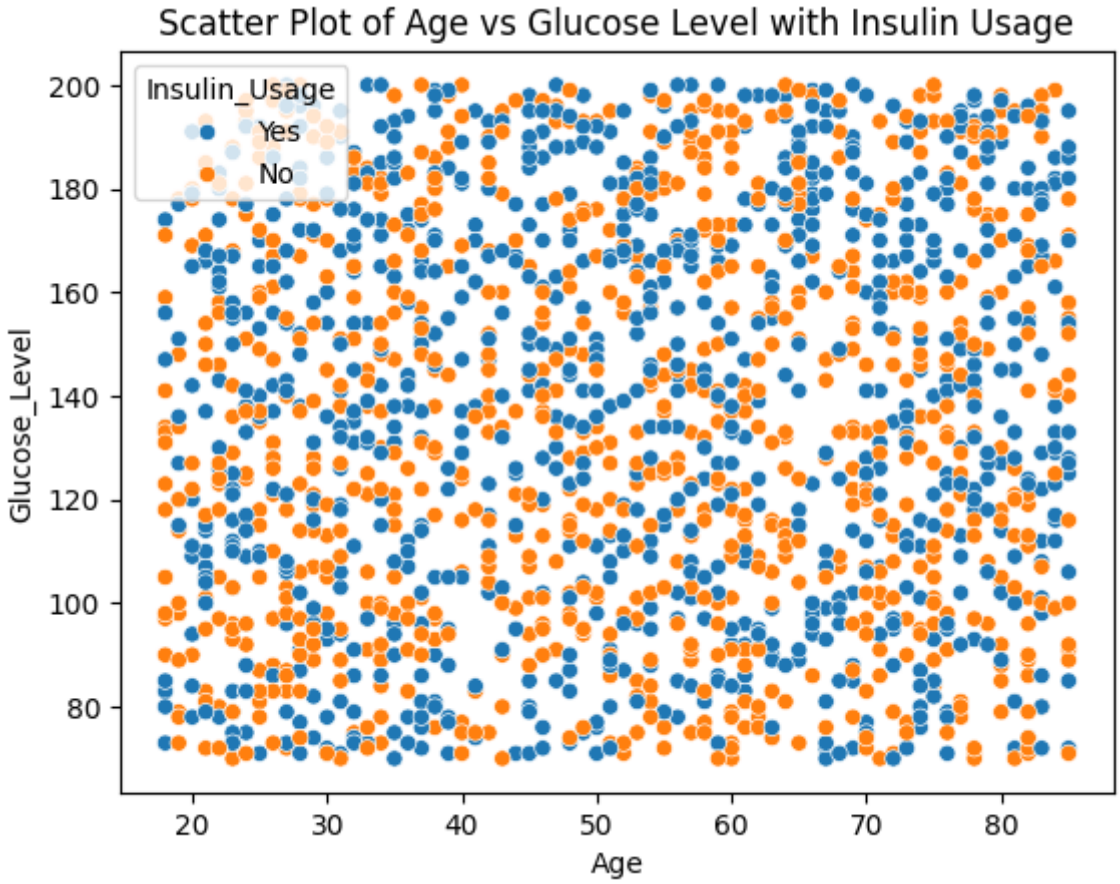
In [20]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(x='Gender', y='Age', hue='Cholesterol_Level', data=df)
plt.title('Boxplot of Age across Gender and Cholesterol Level')
plt.show()
```



Q1.How does the scatter plot of age versus glucose level with insulin usage differentiate patterns in the dataset?

The scatter plot of age versus glucose level with insulin usage provides a visual representation of the dataset's dynamics, allowing us to discern patterns and potential correlations. By incorporating insulin usage as a hue factor, the plot enables a more nuanced exploration of how this variable influences the relationship between age and glucose levels.

In [21]:
```
sns.scatterplot(x='Age', y='Glucose_Level', hue='Insulin_Usage', data=df)
plt.title('Scatter Plot of Age vs Glucose Level with Insulin Usage')
plt.show()
```

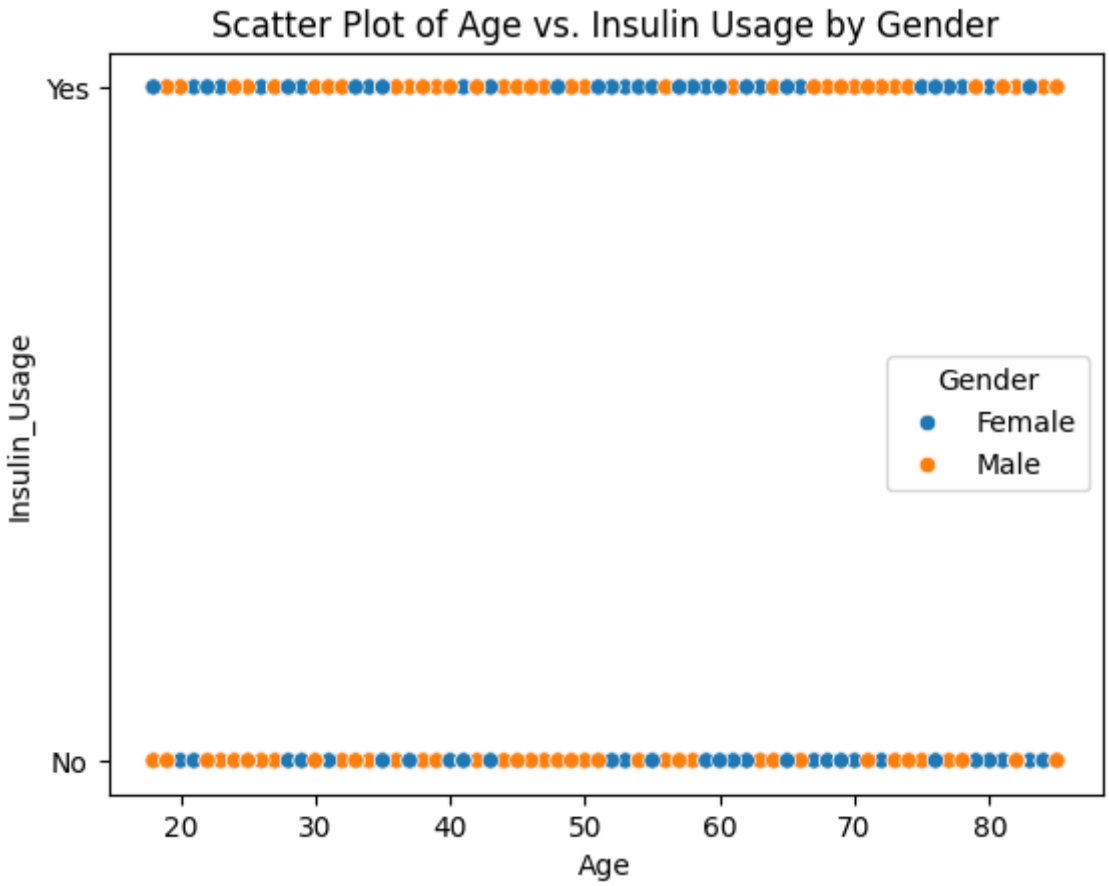


Q2. How does the scatter plot of age versus insulin usage, differentiated by gender, provide insights into the distribution of insulin usage across different age groups?

The scatter plot provides a visual exploration of age-related patterns in insulin usage, and the inclusion of gender as a hue allows for a more comprehensive analysis of how these patterns may vary between different demographic groups.

In [22]:
```
sns.scatterplot(x='Age', y='Insulin_Usage', hue='Gender', data=df)
plt.title('Scatter Plot of Age vs. Insulin Usage by Gender')
```

Out[22]:
```
Text(0.5, 1.0, 'Scatter Plot of Age vs. Insulin Usage by Gender')
```



Below Code:

- Used to calculate the distribution of unique values in the **'Diet_Type'** column of a pandas DataFrame (df).

In [63]:
```
Diet_Type_distribution = df['Diet_Type'].value_counts()
print(Diet_Type_distribution)
```

```
High Protein    677
Balanced        664
High Carb       659
Name: Diet_Type, dtype: int64
```

Below Code:

- The df.info() code is used in pandas to get a concise summary of a DataFrame.
- When you call info() on a DataFrame, it provides information about the DataFrame, including:

1. The total number of entries (rows) in the DataFrame.
2. The range index, which shows the index range (e.g., from 0 to 1999 in your case).
3. The data columns, including the column names, the count of non-null values, and the data type of each column and the memory usage of the DataFrame.

In [24]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Patient_ID         2000 non-null   int64  
 1   Age                2000 non-null   int64  
 2   Gender             2000 non-null   object 
 3   BMI                2000 non-null   float64
 4   Blood_Pressure     2000 non-null   object 
 5   Glucose_Level      1789 non-null   float64
 6   Cholesterol_Level  1917 non-null   object 
 7   Physical_Activity  2000 non-null   float64
 8   Diet_Type          2000 non-null   object 
 9   Insulin_Usage      2000 non-null   object 
 10  HbA1c_Level        1770 non-null   float64
dtypes: float64(4), int64(2), object(5)
memory usage: 172.0+ KB
```