A report on

HealthCare Data Cleaning

Submitted in partial fulfillment of the requirements for

the Master of Computer Application.

Submitted By:

Pragya Tiwari	202410116100144
Saurabh Kumar Singh	202410116100189
Rishabh Singh Bhadauriya	202410116100166
Nitin Gangwar	202410116100136

MCA 1st year (Sec-C)

Submitted To:

Ms. Komal Salgotra
Department of Computer Application
KIET Group of Institutions
Ghaziabad, Uttar Pradesh

Introduction

The healthcare industry generates massive amounts of data daily, including patient records, medical imaging, electronic health records (EHRs), insurance claims, and data from wearable health devices. This data is essential for clinical decision-making, research, healthcare management, and policy formulation. However, due to the complexity and volume of healthcare data, inconsistencies, missing values, redundancies, and errors frequently occur. Poor-quality data can lead to inaccurate diagnoses, inappropriate treatments, and flawed research conclusions.

Data cleaning, also known as data preprocessing, is a critical step in data preparation that ensures accuracy, completeness, and consistency. The objective of data cleaning is to detect and correct (or remove) corrupt, inaccurate, or irrelevant data. In the healthcare industry, clean data is essential for improving patient care, optimizing hospital operations, enhancing medical research, and enabling effective machine learning applications.

This project focuses on applying data cleaning techniques to a healthcare dataset. It includes handling missing values, standardizing categorical variables, removing duplicate records, treating outliers, and performing exploratory data analysis (EDA) to extract meaningful insights. The ultimate goal is to refine the dataset to ensure its reliability for further analytics and decision-making.

Objectives

This project aims to:

- Identify and handle missing values efficiently.
- Detect and remove duplicate records to prevent redundancy.
- Standardize categorical variables for consistency and uniformity.
- Identify and treat outliers to maintain data integrity.
- Perform exploratory data analysis (EDA) to understand data distribution and relationships.
- Generate visualizations to facilitate better understanding of data patterns.
- Save and export the cleaned dataset for further analytical and machine-learning applications.

Dataset Overview

The dataset used in this project contains anonymized patient records, covering demographic information, medical history, and test results. The key attributes in the dataset include:

- **Patient ID** – A unique identifier assigned to each patient.
- **Age** – Patient's age in numerical form.
- **Gender** – Categorical variable (Male/Female).
- **Blood Pressure** – Systolic and diastolic blood pressure readings.
- **Cholesterol Levels** – Recorded cholesterol levels of patients.
- **Diabetes Status** – Indicator (Yes/No) showing if the patient has diabetes.
- **Medical History** – Records of previous illnesses or treatments.
- **Lab Test Results** – Values from various medical examinations.

Data Cleaning Techniques

1. **Handling Missing Values**

Healthcare data frequently contains missing values due to human error, system failures, or incomplete records. To manage missing values:

- **Categorical features** were imputed using the mode (most frequent value) to maintain consistency.
- **Numerical features** were imputed using the median to prevent extreme values from skewing the dataset.
- **Visualization**: A heatmap was generated to identify missing values across different attributes.

2. **Removing Duplicates**

Duplicate records can distort analytical results and affect predictive modeling. To ensure data integrity:

- Identified duplicate entries using the Patient ID and key attributes.

- Removed duplicate rows while preserving the first occurrence.
- Verified dataset integrity after deduplication.

3. **Standardizing Categorical Data**

Inconsistent categorical values often arise due to manual data entry errors. Examples include:

- **Gender Column:** Standardized variations such as "Male", "male", "M" into "Male" and "Female", "female", "F" into "Female".
- **Diabetes Status Column:** Ensured consistency by converting variations ("Yes", "yes", "Y") into "Yes" and ("No", "no", "N") into "No".

4. **Handling Outliers**

Outliers in healthcare data may result from measurement errors or natural variations in patient health parameters. These were managed using:

- **Interquartile Range (IQR) Method** to identify and cap extreme values.
- **Boxplots** to visualize and detect outliers in blood pressure and cholesterol levels.
- **Winsorization** to cap extreme outliers and prevent distortion of data distributions.

Exploratory Data Analysis (EDA)

EDA was performed to understand data distributions, correlations, and trends using various visualization techniques.

1. **Data Distribution Visualization**

- **Histograms** were used to analyze age distribution among patients.
- **Kernel Density Estimation (KDE) plots** helped visualize the distribution of blood pressure and cholesterol levels.

2. **Correlation Heatmap**

- A **correlation matrix** was generated to analyze relationships between numerical attributes such as blood pressure, cholesterol, and age.
- **High correlation values** between certain variables indicated potential predictive relationships.

3. **Count Plots**

- **Count plots** helped analyze categorical distributions, such as the number of male and female patients and the prevalence of diabetes.

4. **Pair Plots**

- **Pair plots** were generated to visualize relationships between numerical variables (e.g., Age vs. Blood Pressure, Cholesterol vs. Diabetes Status).

5. **Violin Plots**

- **Violin plots** were used to compare distributions of key numerical features with respect to categorical distinctions like gender and diabetes status.

Findings

After conducting data cleaning and exploratory analysis, the following observations were made:

- **Missing values** were successfully handled, reducing data loss while maintaining statistical integrity.
- **Duplicate records** were removed, ensuring data uniqueness and accuracy.
- **Outliers** were treated using IQR-based capping, preventing data distortion.
- **Categorical data** was standardized to maintain uniformity across the dataset.
- **Data visualizations** provided valuable insights into feature relationships and data distributions.

Conclusion & Future Scope

➢ **Conclusion**

Clean data is fundamental for accurate analysis and predictive modeling in healthcare. This project successfully demonstrated:

- The importance of data preprocessing in healthcare analytics.
- Effective data-cleaning techniques for missing values, duplicates, and outliers.
- The role of EDA and visualization in understanding healthcare data patterns.

- The impact of data quality on medical decision-making and research outcomes.

➢ **Future Scope**

To enhance this project further, the following improvements can be considered:

- **Automated anomaly detection** using machine learning techniques.
- **Real-time data cleaning pipelines** for continuously updating healthcare datasets.
- **Feature engineering** to derive new insights from the cleaned dataset.
- **Integration with predictive analytics** for disease detection and risk assessment.

References

- Healthcare Data Analytics by Chandan K. Reddy & Charu C. Aggarwal.
- Principles of Data Cleaning in Healthcare - ResearchGate.
- Data Science in Healthcare: Applications and Techniques - MIT Press.

This project successfully demonstrates the importance of high-quality healthcare data and provides a structured approach to data cleaning, making it suitable for further research and analysis.

Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from google.colab import files

import shutil


# Load the dataset

file_path = '/content/healthcare_data.csv'  # Ensure this matches the uploaded file name
```

```
df = pd.read_csv(file_path)

# Display basic information
print("Dataset Overview:")
print(df.info())
print("\nFirst 5 rows:")
print(df.head())

# Visualize missing values
plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cmap='viridis', cbar=False)
plt.title('Missing Values Heatmap')
plt.show()

# Handling missing values
# Fill numerical columns with median, categorical columns with mode
for col in df.columns:
    if df[col].dtype == 'object':
        df[col].fillna(df[col].mode()[0], inplace=True)
    else:
        df[col].fillna(df[col].median(), inplace=True)

# Removing duplicates
duplicate_count = df.duplicated().sum()
print(f"\nNumber of duplicate rows removed: {duplicate_count}")
```

```
df.drop_duplicates(inplace=True)

# Checking inconsistencies (example: standardizing categorical values)
if 'Gender' in df.columns:
    df.replace({'Male': 'M', 'Female': 'F', 'male': 'M', 'female': 'F'}, inplace=True)

# Identifying outliers using IQR method and capping extreme values
for col in df.select_dtypes(include=np.number).columns:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[col] = np.where(df[col] < lower_bound, lower_bound, df[col])
    df[col] = np.where(df[col] > upper_bound, upper_bound, df[col])

# Summary after cleaning
print("\nData after Cleaning:")
print(df.info())

# Visualization: Distribution of a numerical column (e.g., Age if exists)
if 'Age' in df.columns:
    plt.figure(figsize=(8,5))
    sns.histplot(df['Age'], bins=20, kde=True, color='blue')
    plt.title('Age Distribution After Cleaning')
```

```
    plt.xlabel('Age')
    plt.ylabel('Count')
    plt.show()

# Visualization: Correlation Heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap of Numeric Features')
plt.show()

# Visualization: Boxplot for Outlier Analysis
if 'Age' in df.columns:
    plt.figure(figsize=(8,5))
    sns.boxplot(x=df['Age'], color='red')
    plt.title('Boxplot of Age After Outlier Handling')
    plt.show()

# Additional Visualizations
# Count plot for categorical features
for col in df.select_dtypes(include=['object']).columns:
    plt.figure(figsize=(8, 5))
    sns.countplot(x=df[col], palette='Set2')
    plt.title(f'Count Plot of {col}')
    plt.xticks(rotation=45)
    plt.show()
```

 # Pairplot for numerical data

```
sns.pairplot(df.select_dtypes(include=np.number))
plt.show()
```

 # Violin plot for distribution analysis

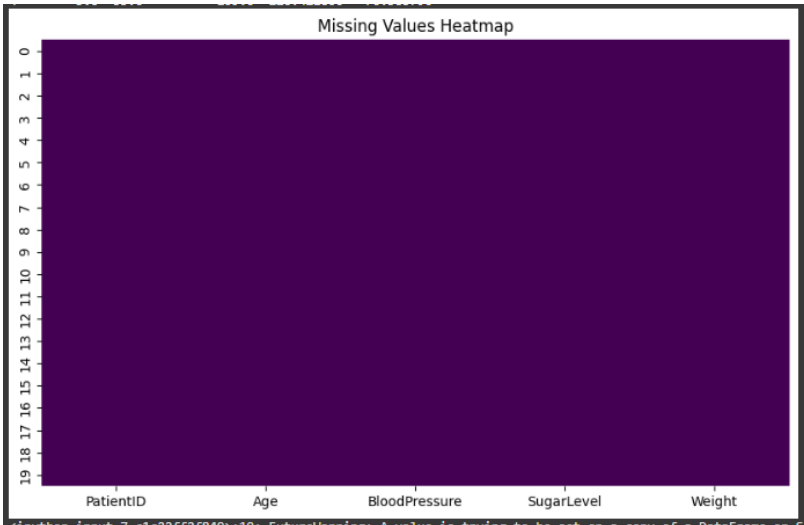
```
for col in df.select_dtypes(include=np.number).columns:
    plt.figure(figsize=(8, 5))
    sns.violinplot(y=df[col], palette='muted')
    plt.title(f'Violin Plot of {col}')
    plt.show()
```

 # Save cleaned dataset

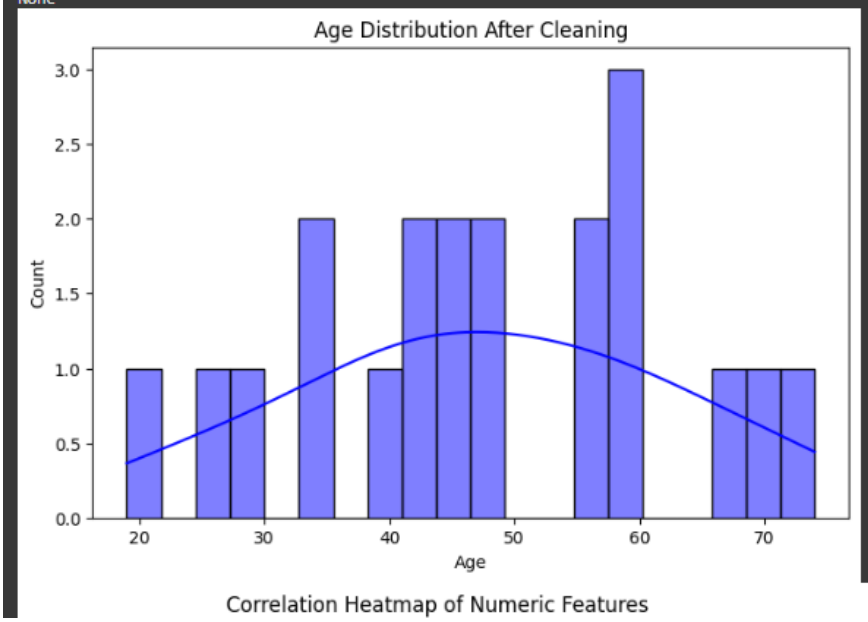
```
df.to_csv('cleaned_healthcare_data.csv', index=False)
shutil.move('cleaned_healthcare_data.csv', '/content/cleaned_healthcare_data.csv')
files.download('/content/cleaned_healthcare_data.csv')
```

Output:

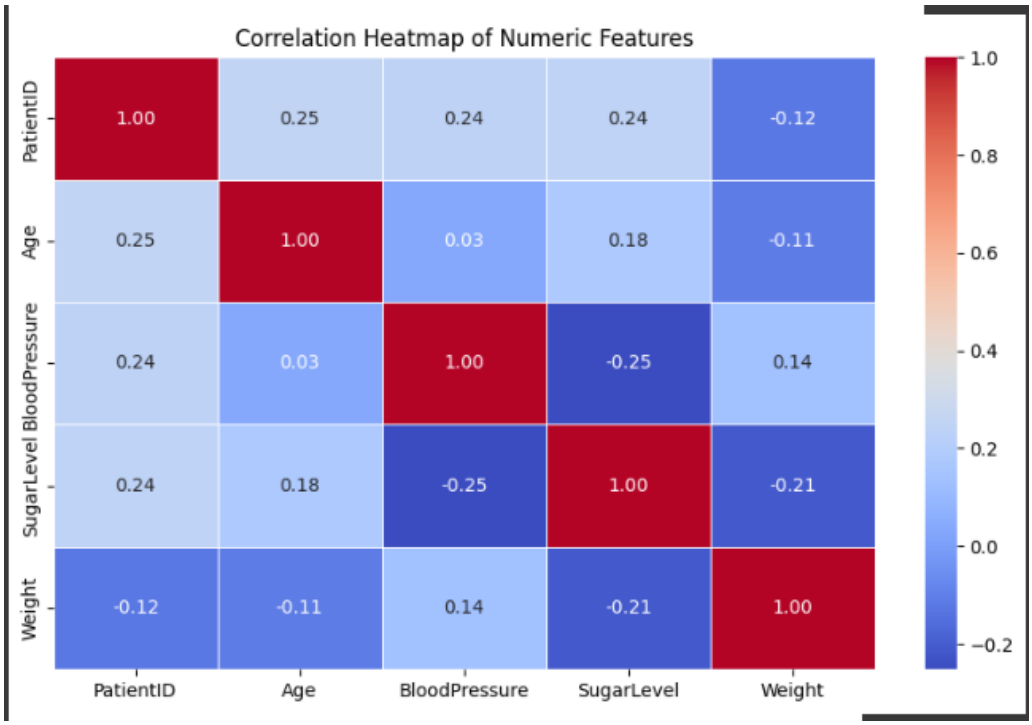
```
Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   PatientID      20 non-null     float64
 1   Age            20 non-null     float64
 2   BloodPressure  20 non-null     float64
 3   SugarLevel     20 non-null     float64
 4   Weight         20 non-null     float64
dtypes: float64(5)
memory usage: 932.0 bytes
None

First 5 rows:
   PatientID   Age  BloodPressure  SugarLevel      Weight
0        1.0  44.0          118.0   87.892495  105.568034
1        2.0  39.0          109.0  177.321803  105.703426
2        3.0  49.0          149.0  144.148273   77.787070
3        4.0  58.0          121.0   90.355404  115.244784
4        5.0  35.0          109.0  126.421800   70.383790
```

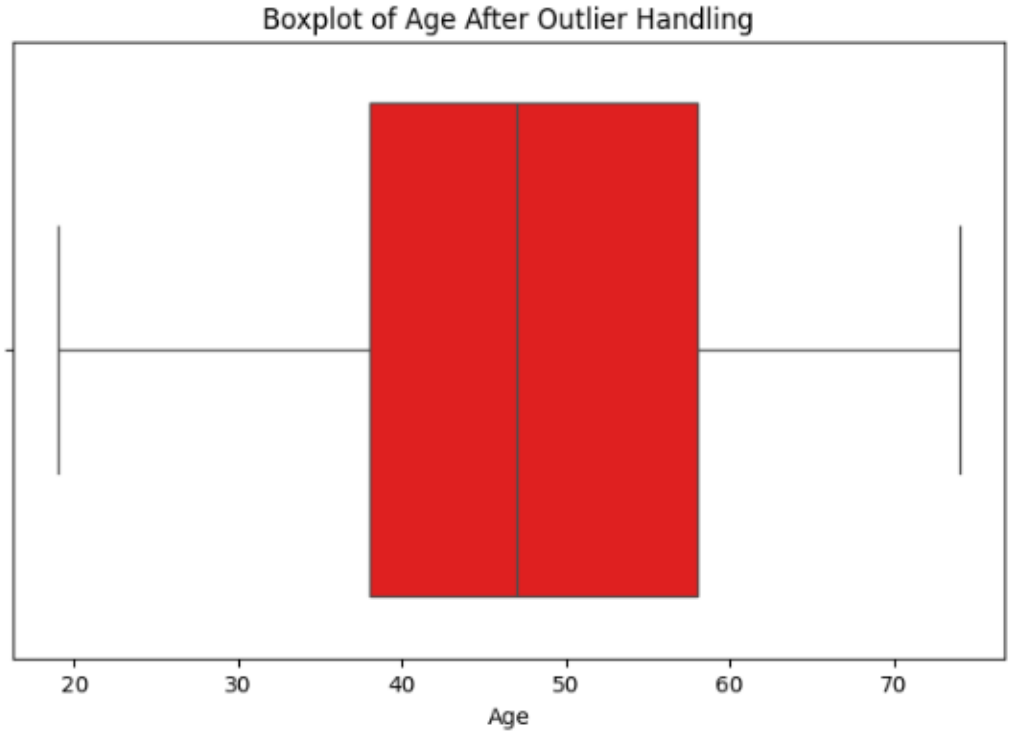


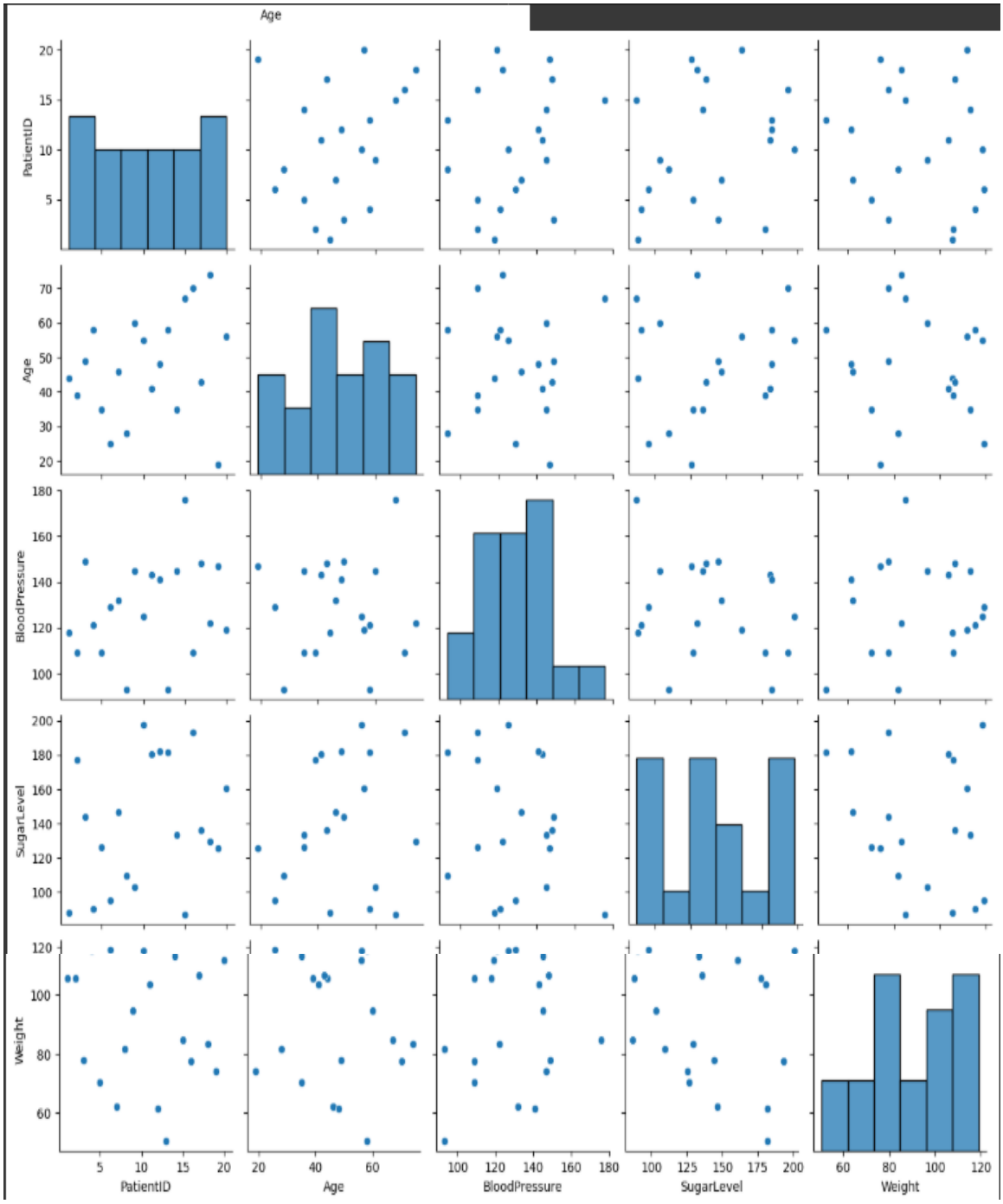

```
<ipython-input-7-c1c22ff2f849>:19: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the


  df[col].fillna(df[col].median(), inplace=True)

Number of duplicate rows removed: 0

Data after Cleaning:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   PatientID      20 non-null     float64
 1   Age            20 non-null     float64
 2   BloodPressure  20 non-null     float64
 3   SugarLevel     20 non-null     float64
 4   Weight         20 non-null     float64
dtypes: float64(5)
memory usage: 932.0 bytes
None
```



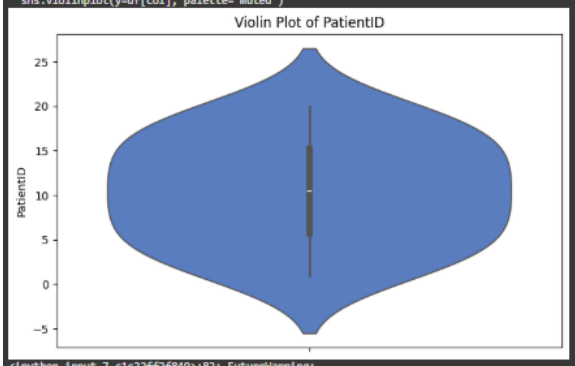
Correlation Heatmap of Numeric Features


Correlation Heatmap of Numeric Features
PatientID
Age
BloodPressure
SugarLevel
Weight
1.00
0.25
0.24
0.24
-0.12
0.25
1.00
0.03
0.18
-0.11
0.24
0.03
1.00
-0.25
0.14
0.24
0.18
-0.25
1.00
-0.21
-0.12
-0.11
0.14
-0.21
1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

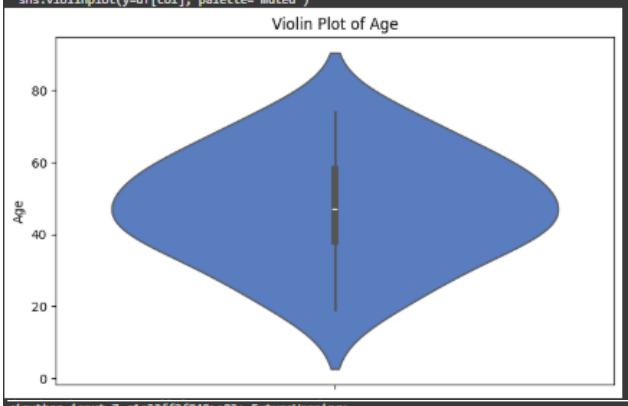

Boxplot of Age After Outlier Handling
20
30
40
50
60
70
Age


Age
PatientID
Age
BloodPressure
SugarLevel
Weight
PatientID
Age
BloodPressure
SugarLevel
Weight

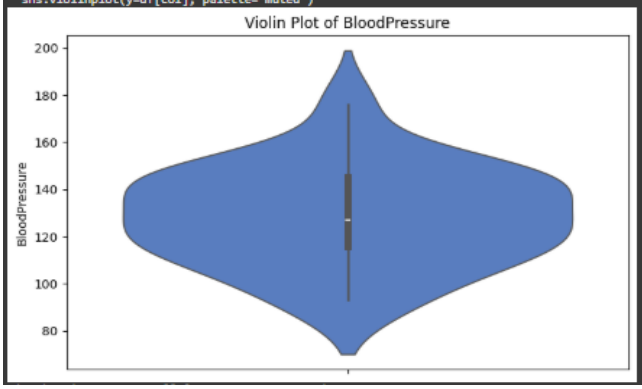
```
<ipython-input-7-c1c22ff2f849>:82: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.violinplot(y=df[col], palette='muted')
```



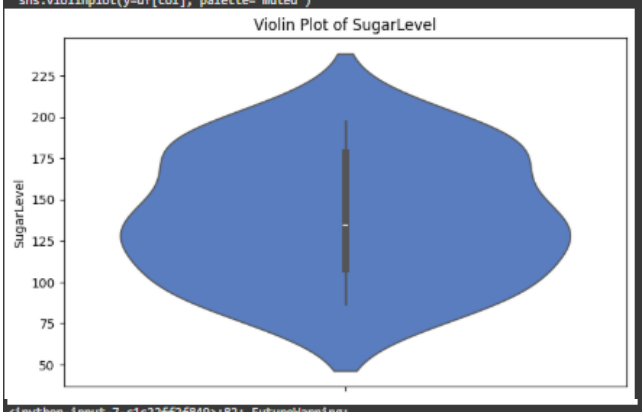

```
<ipython-input-7-c1c22ff2f849>:82: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.violinplot(y=df[col], palette='muted')
<ipython-input-7-c1c22ff2f849>:82: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.violinplot(y=df[col], palette='muted')
```




```
<ipython-input-7-c1c22ff2f849>:82: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.violinplot(y=df[col], palette='muted')
```



```
<ipython-input-7-c1c22ff2f849>:82: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.violinplot(y=df[col], palette='muted')
```




```
<ipython-input-7-c1c22ff2f849>:82: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.violinplot(y=df[col], palette='muted')
```