Assignment No. 6

Problem Statement: DATA PREPROCESSING: Perform the following operations using Python on the heart diseases data sets a. Data cleaning b. Error-correcting

Objective: To preprocess the heart disease dataset by performing data cleaning and error correction to enhance data quality and reliability. This includes handling missing values and correcting inconsistencies, preparing the dataset for accurate analysis and predictive modeling in diagnosing heart disease.

Prerequisite :

1. Basic understanding of Python programming.
2. Understanding of Data cleaning, Data Preprocessing and error detecting.
3. Understanding of libraries like Pandas, NumPy, Matplotlib, and Seaborn.
4. Knowledge of libraries such as NumPy and Matplotlib for data generation and visualization

Theory :

Data Cleaning:

Data cleaning is vital to ensure data quality by managing missing values, reducing noise, and correcting inconsistencies. Raw data often leads to inaccurate analyses, making cleaning necessary for reliable results

a) Handling Missing Data

- **Removing Rows/Columns:** Rows or columns with more than 65% missing data, or those completely null, can be removed to maintain data integrity.
- **Duplicate Removal:** Duplicates should be removed to prevent distorted analysis results.
- **Filling Missing Values:** When only a few values are missing, they can be filled in using statistical measures like the mean, median, or mode.

b) Addressing Noisy Data

- **Binning:** Values are organized into categories to reduce fluctuations within each bin.
- **Clustering:** Grouping similar data points can help detect and manage outliers.
- **Regression:** A regression model can be used to smooth data by fitting values to a trend line.

Error-Correction:

Error correction in data science involves identifying, addressing, and acknowledging inaccuracies or omissions in data. These errors can arise from various sources, including manual entry mistakes, sensor malfunctions, software bugs, and issues related to data integration. If left unaddressed, such errors pose risks to various analytical processes, machine learning models, and the decisions derived from the data. Correcting these inaccuracies is crucial for maintaining the integrity and reliability of data-driven insights.

Types of Errors in Data

1. **Duplicate Data**: This refers to identical entries or datasets that can skew analysis results in undesirable ways.
2. **Outliers**: Outliers are unusually high or low values that can result from data entry mistakes or from rare events.
3. **Measurement Errors**: These occur when there are flaws in data collection, often due to the use of faulty instruments or incorrect measurement recordings.
4. **Missing Values**: These are values that should have been included in a dataset but are absent, potentially compromising the integrity of analysis and other processes significantly.
5. **Inconsistent Databases**: These are instances where entries in a database convey the same meaning but are formatted or named differently.
6. **Data Entry Errors**: These mistakes arise during data compilation, such as typographical errors or missing information.

Error Correction Techniques

1. Data Validation

This refers to the process of enforcing constraints on data to ensure it consistently meets specified conditions, thereby minimizing the risk of inputting or recording inaccurate information. By implementing these constraints, organizations can maintain data integrity and reliability, which is crucial for effective analysis and decision-making.

Techniques:

a) **Range Checks:** This procedure ensures that the provided value falls within specified limits. For instance, an age value should be a natural number within a defined range.
b) **Consistency Checks**: These checks verify whether related data items are logically consistent with each other. For example, a start date should not be later than an end date.
c) **Format Checks:** These checks ensure that certain data types, such as letters, email addresses, phone numbers, and dates, adhere to acceptable formats.

2. Data Imputation for Missing Values:

The issue of missing values is addressed by replacing the absent data with suitable values, ensuring that the integrity of the dataset remains intact. This substitution helps maintain the reliability of analyses and prevents biases that could arise from incomplete data.

Techniques:

a) **Mean/Median Imputation:** This method involves replacing missing values with the mean or median of the column, making it particularly useful for numerical data.
b) **Mode Imputation:** For categorical data, missing values are substituted with the most frequently occurring category, ensuring that the data distribution is maintained.
c) **Predictive Imputation:** This approach uses statistical learning models to estimate and fill in missing values based on the relationships with other available variables.
d) **K-Nearest Neighbors (KNN) Imputation:** This technique replaces missing values by averaging the values of the ‘k’ nearest neighbours of a data point, effectively leveraging the information from similar data

3. Outlier Detection and Correction

Outliers are data points that, if not addressed properly, can skew analyses and models. The process of discrepancy detection and correction entails identifying these anomalies and determining the appropriate actions to take in response. This could involve removing the outliers, transforming them, or applying methods to minimize their impact, ensuring that the overall data quality and analysis accuracy are preserved

Methods:

a) **Z-score Method:** This method calculates how many standard deviations a data point is from the mean. Data points that exceed a specified threshold (commonly ±3) are considered outliers.
b) **Interquartile Range (IQR):** Outliers are defined as values that fall outside the range of [Q1−1.5×IQR,Q3+1.5×IQR][Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR][Q1−1.5×IQR,Q3+1.5×IQR], where Q1Q1Q1 and Q3Q3Q3 represent the first and third quartiles, respectively.
c) **Trimming/Capping:** This technique involves removing extreme cases from the dataset or setting a maximum cutoff value for extreme observations to mitigate their influence.
d) **Winsorizing:** In this statistical practice, extreme values are replaced with the nearest acceptable value within a defined range, reducing the impact of outliers while preserving the overall structure of the data.

4. Duplicate Data Detection and removal

The addition of more datasets to the active dataset can lead to the emergence of duplicate entries. These duplicates can skew counts, introduce biases into statistical analyses, and result in overfitting when training models. To maintain the integrity and accuracy of data analyses, it is essential to identify and remove duplicate datasets to ensure that the insights drawn from the data are reliable and valid.

Methods:

7. **Exact Duplicate Removal:** This process involves identifying and deleting entries that are identical to others in the dataset, ensuring that only unique information is retained.
8. **Fuzzy Matching:** This technique assesses similarities between entries using a metric, such as the Levenshtein distance, to identify near-duplicates based on the similarity of their string values. It is particularly useful for handling text data where minor variations may exist.

Benefits of Error Correction

Effective error correction enhances the accuracy, reliability, and consistency of data, making it more usable for analytics, modeling, and decision-making processes. By addressing errors, it helps reduce biases, prevents misleading conclusions, and increases the interpretability and performance of predictive models. Ultimately, this leads to more informed and reliable insights derived from the data.

Challenges in Error Correction

a) **Cost and Time**: Error detection and correction can be resource-intensive, especially for large datasets.
b) **Scalability**: As data grows, error correction methods must scale, often necessitating automation.
c) **Subjectivity**: Some corrections depend on assumptions or domain knowledge, potentially introducing bias.
d) **Balancing Correction with Data Integrity**: Excessive error correction, especially through removal, can lead to data loss and reduce representativeness.

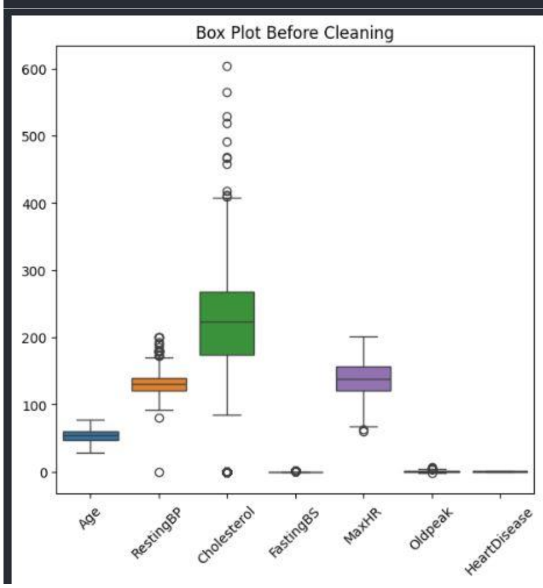
Code & Output:

```
import pandas as pd  # Import the pandas library for data manipulation
# Load the dataset
df = pd.read_csv('C:/Users/vishal_2/Assignments/Datasets/heart.csv')  # Read the CSV file into a DataFrame
# Select the data from the fifth column (index 4)
data = df.iloc[:, 4]  # iloc is used to select rows and columns by index positions
# Display the first five rows of the selected data
print(data.head())  # Print the first 5 values of the selected column
```

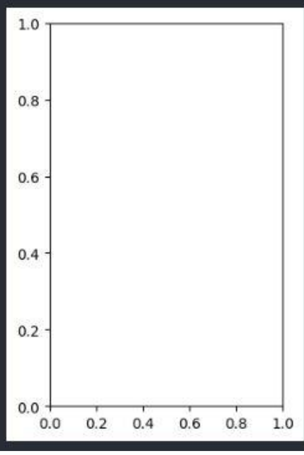
```
0    289
1    180
2    283
3    214
4    195
Name: Cholesterol, dtype: int64
```

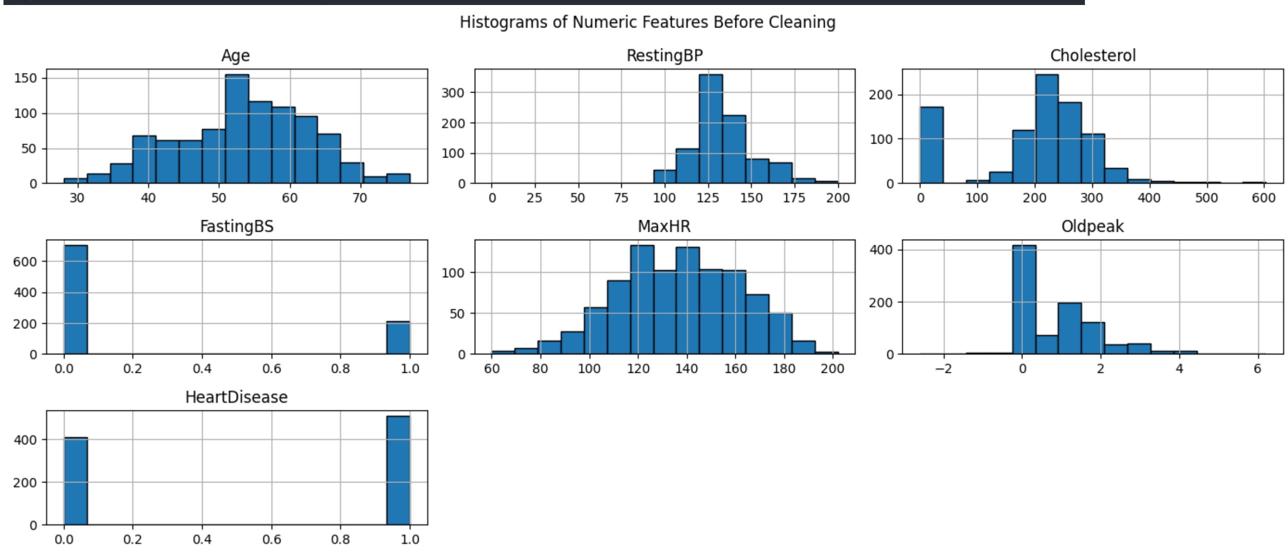
```
# Before Cleaning: Visualize Outliers
plt.figure(figsize=(14, 6))  # Set the figure size for better readability
plt.subplot(1, 2, 1)  # Create a subplot with 1 row and 2 columns, activate the first plot
sns.boxplot(data=df[numeric_columns])  # Generate a box plot for the numeric columns to visualize outliers
plt.title('Box Plot Before Cleaning')  # Add a title to the box plot
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if there are many numeric columns
```

```
([0, 1, 2, 3, 4, 5, 6],
 [Text(0, 0, 'Age'),
  Text(1, 0, 'RestingBP'),
  Text(2, 0, 'Cholesterol'),
  Text(3, 0, 'FastingBS'),
  Text(4, 0, 'MaxHR'),
  Text(5, 0, 'Oldpeak'),
  Text(6, 0, 'HeartDisease')])
```



```python
# Plot Histograms
plt.subplot(1, 2, 2)  # Activate the second plot in a 1x2 subplot layout
df[numeric_columns].hist(bins=15, edgecolor='black', figsize=(14, 6))  # Plot histograms for each numeric column with 15 bins
plt.suptitle('Histograms of Numeric Features Before Cleaning')  # Add a super title for the entire figure
plt.tight_layout()  # Adjust subplot parameters to give padding and prevent overlap
plt.show()  # Display the plot
```





```python
# Step 1: Data Cleaning
# Check for missing values
print("Missing values in each column:")  # Inform that missing values are being checked
print(df.isnull().sum())  # Display the count of missing values for each column in the DataFrame
```

```
Missing values in each column:
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

```python
#NO missing values Dound in the heart disease dataset
```

```python
# Create a boolean DataFrame indicating missing values
dataset_null = df.isnull()  # Each cell will show True if the value is missing, otherwise Fals
print(dataset_null)  # Print the DataFrame to see the location of missing values
```

```
       Age    Sex  ChestPainType  RestingBP  Cholesterol  FastingBS  \
0    False  False          False      False        False      False   
1    False  False          False      False        False      False   
2    False  False          False      False        False      False   
3    False  False          False      False        False      False   
4    False  False          False      False        False      False   
..     ...    ...            ...        ...          ...        ...   
913  False  False          False      False        False      False   
914  False  False          False      False        False      False   
915  False  False          False      False        False      False   
916  False  False          False      False        False      False   
917  False  False          False      False        False      False   

     RestingECG  MaxHR  ExerciseAngina  Oldpeak  ST_Slope  HeartDisease  
0         False  False           False    False     False         False  
1         False  False           False    False     False         False  
2         False  False           False    False     False         False  
3         False  False           False    False     False         False  
4         False  False           False    False     False         False  
..          ...    ...             ...      ...       ...           ...  
913       False  False           False    False     False         False  
914       False  False           False    False     False         False  
915       False  False           False    False     False         False  
916       False  False           False    False     False         False  
917       False  False           False    False     False         False  

[918 rows x 12 columns]
```

```python
# Display DataFrame summary
df.info()  # Provides a concise summary of the DataFrame, including column names, non-null counts, and data types
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

```python
# 1. Check for inconsistent values in categorical columns and display unique values
for col in ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']:  # Loop through specified categorical columns
    print(f"Unique values in {col} column: {df[col].unique()}")  # Display unique values in each categorical column
```

```
Unique values in Sex column: ['M' 'F']
Unique values in ChestPainType column: ['ATA' 'NAP' 'ASY' 'TA']
Unique values in RestingECG column: ['Normal' 'ST' 'LVH']
Unique values in ExerciseAngina column: ['N' 'Y']
Unique values in ST_Slope column: ['Up' 'Flat' 'Down']
```

```python
# 2. Encode categorical columns
df['Sex'] = df['Sex'].map({'M': 1, 'F': 0})  # Encode 'Sex' column: 'M' as 1, 'F' as 0
df['ChestPainType'] = df['ChestPainType'].map({'TA': 0, 'ATA': 1, 'NAP': 2, 'ASY': 3})  # Encode 'ChestPainType' with numerical values
df['RestingECG'] = df['RestingECG'].map({'Normal': 0, 'ST': 1, 'LVH': 2})  # Encode 'RestingECG' with numerical values
df['ExerciseAngina'] = df['ExerciseAngina'].map({'N': 0, 'Y': 1})  # Encode 'ExerciseAngina': 'N' as 0, 'Y' as 1
df['ST_Slope'] = df['ST_Slope'].map({'Up': 0, 'Flat': 1, 'Down': 2})  # Encode 'ST_Slope' with numerical values
```

```python
import numpy as np

# 3. Handle outliers in numeric columns using Interquartile Range (IQR) method
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[column] = np.where(df[column] < lower_bound, lower_bound, df[column])
    df[column] = np.where(df[column] > upper_bound, upper_bound, df[column])

for col in ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']:
    remove_outliers(df, col)
```

```python
# 4. Verify the data types after cleaning and encoding
print(df.info())

# Display the cleaned data
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    float64
 1   Sex             918 non-null    int64
 2   ChestPainType   918 non-null    int64
 3   RestingBP       918 non-null    float64
 4   Cholesterol     918 non-null    float64
 5   FastingBS       918 non-null    int64
 6   RestingECG      918 non-null    int64
 7   MaxHR           918 non-null    float64
 8   ExerciseAngina  918 non-null    int64
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    int64
 11  HeartDisease    918 non-null    int64
dtypes: float64(5), int64(7)
memory usage: 86.2 KB
None
    Age  Sex  ChestPainType  RestingBP  Cholesterol  FastingBS  RestingECG  \
0  40.0    1              1      140.0        289.0          0           0
1  49.0    0              2      160.0        180.0          0           0
2  37.0    1              1      130.0        283.0          0           1
3  48.0    0              3      138.0        214.0          0           0
...
1  156.0               0      1.0         1           1
2   98.0               0      0.0         0           0
3  108.0               1      1.5         1           1
4  122.0               0      0.0         0           0
```

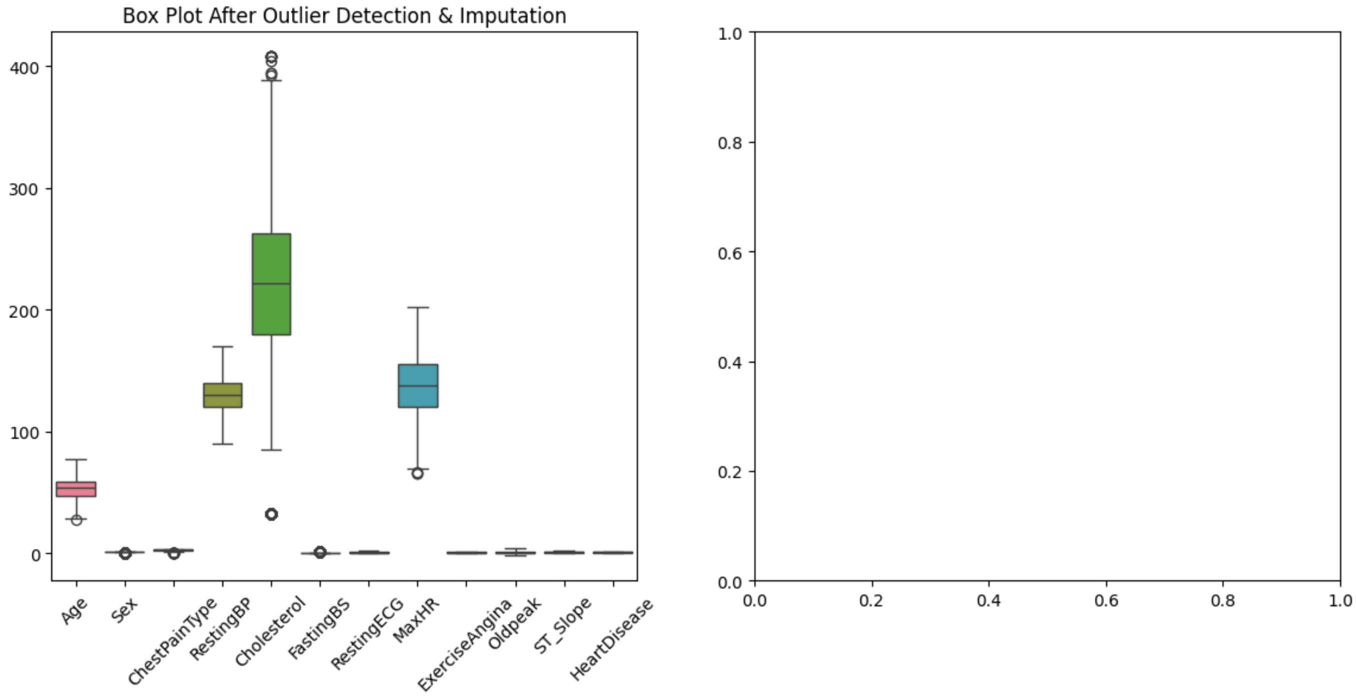
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

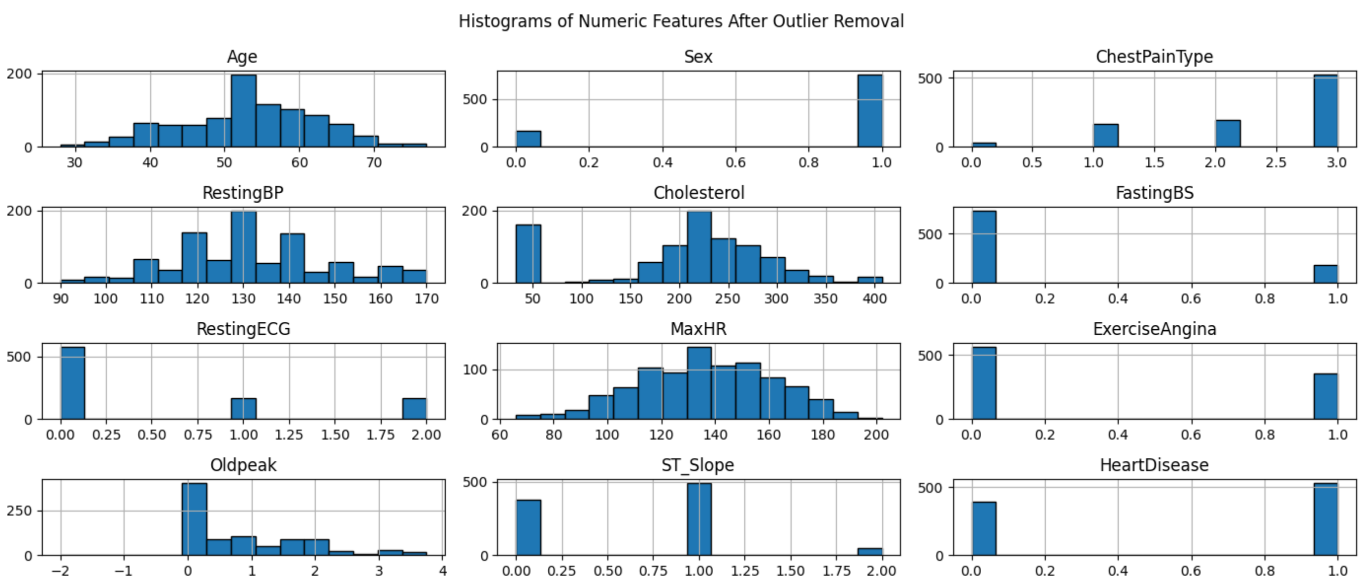
```python
# 1. Data Validation

# Set valid ranges for numerical columns (age, cholesterol, etc.)
valid_age_range = (0, 120)
valid_resting_bp_range = (0, 200)
valid_cholesterol_range = (0, 600)
valid_max_hr_range = (0, 220)
```

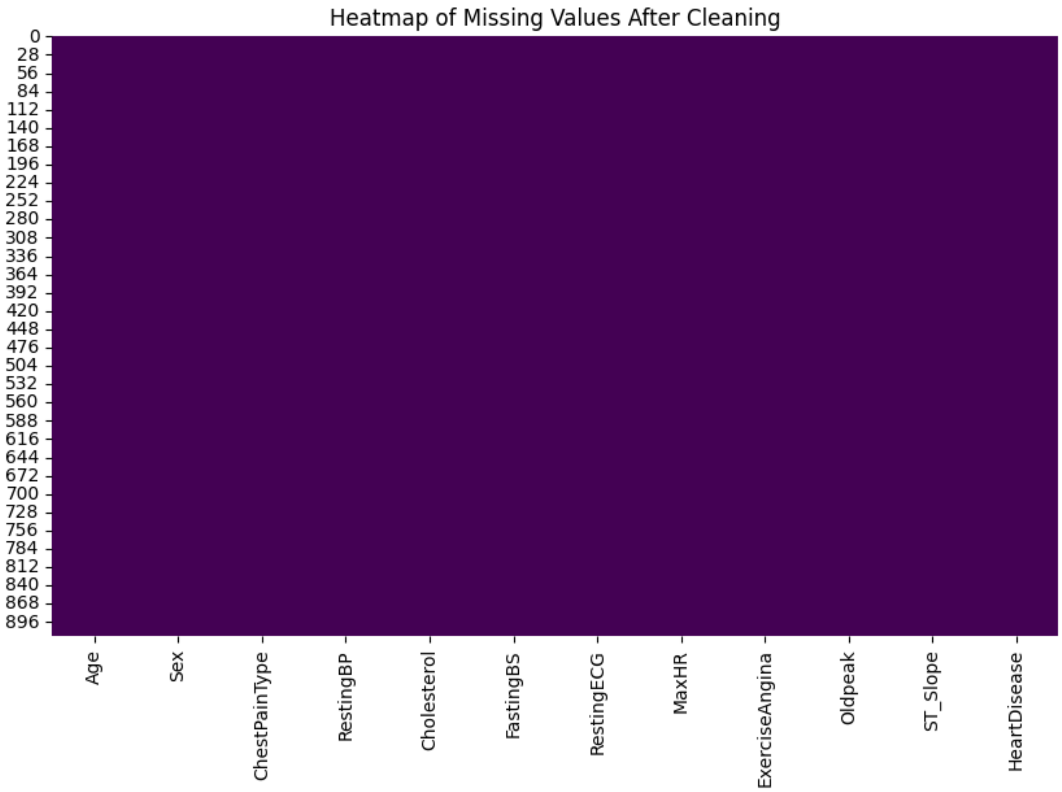
```python
# Apply range checks and replace out-of-range values with NaN
df.loc[~df['Age'].between(*valid_age_range), 'Age'] = np.nan
df.loc[~df['RestingBP'].between(*valid_resting_bp_range), 'RestingBP'] = np.nan
df.loc[~df['Cholesterol'].between(*valid_cholesterol_range), 'Cholesterol'] = np.nan
df.loc[~df['MaxHR'].between(*valid_max_hr_range), 'MaxHR'] = np.nan
```

```python
from scipy import stats
# 2. Outlier Detection and Correction

# Using Z-score method for outlier detection in numerical columns
numeric_columns = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']
z_scores = np.abs(stats.zscore(df[numeric_columns]))
outliers = (z_scores > 3)  # Threshold set to 3 standard deviations
df[numeric_columns] = df[numeric_columns].mask(outliers, np.nan)
```

```python
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.ensemble import IsolationForest
import matplotlib.pyplot as plt
import seaborn as sns

# Load your DataFrame (assuming you've already done this)
# df = pd.read_csv('your_dataset.csv') # Uncomment and modify this line to load your dataset

# Define numeric columns
numeric_columns = df.select_dtypes(include=[np.number]).columns.tolist()

# 1. Handle Missing Values - Impute using Median
imputer = SimpleImputer(strategy='median')
df[numeric_columns] = imputer.fit_transform(df[numeric_columns])

# 2. Detect outliers using Isolation Forest
iso_forest = IsolationForest(contamination=0.05)
outliers = iso_forest.fit_predict(df[numeric_columns])

# 3. Mark outliers as NaN for imputation
df.loc[outliers == -1, numeric_columns] = np.nan

# 4. Impute outliers again using median for numeric columns
df[numeric_columns] = imputer.fit_transform(df[numeric_columns])

# Visualization
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
sns.boxplot(data=df[numeric_columns])
plt.title('Box Plot After Outlier Detection & Imputation')
plt.xticks(rotation=45)

# Plot Histograms After Cleaning
plt.subplot(1, 2, 2)
df[numeric_columns].hist(bins=15, edgecolor='black', figsize=(14, 6))
plt.suptitle('Histograms of Numeric Features After Outlier Removal')

plt.tight_layout()
plt.show()

# Heatmap for Missing Values
plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title('Heatmap of Missing Values After Cleaning')
plt.show()
```


Box Plot After Outlier Detection & Imputation
Age
Sex
ChestPainType
RestingBP
Cholesterol
FastingBS
RestingECG
MaxHR
ExerciseAngina
Oldpeak
ST_Slope
HeartDisease


Histograms of Numeric Features After Outlier Removal
Age
Sex
ChestPainType
RestingBP
Cholesterol
FastingBS
RestingECG
MaxHR
ExerciseAngina
Oldpeak
ST_Slope
HeartDisease

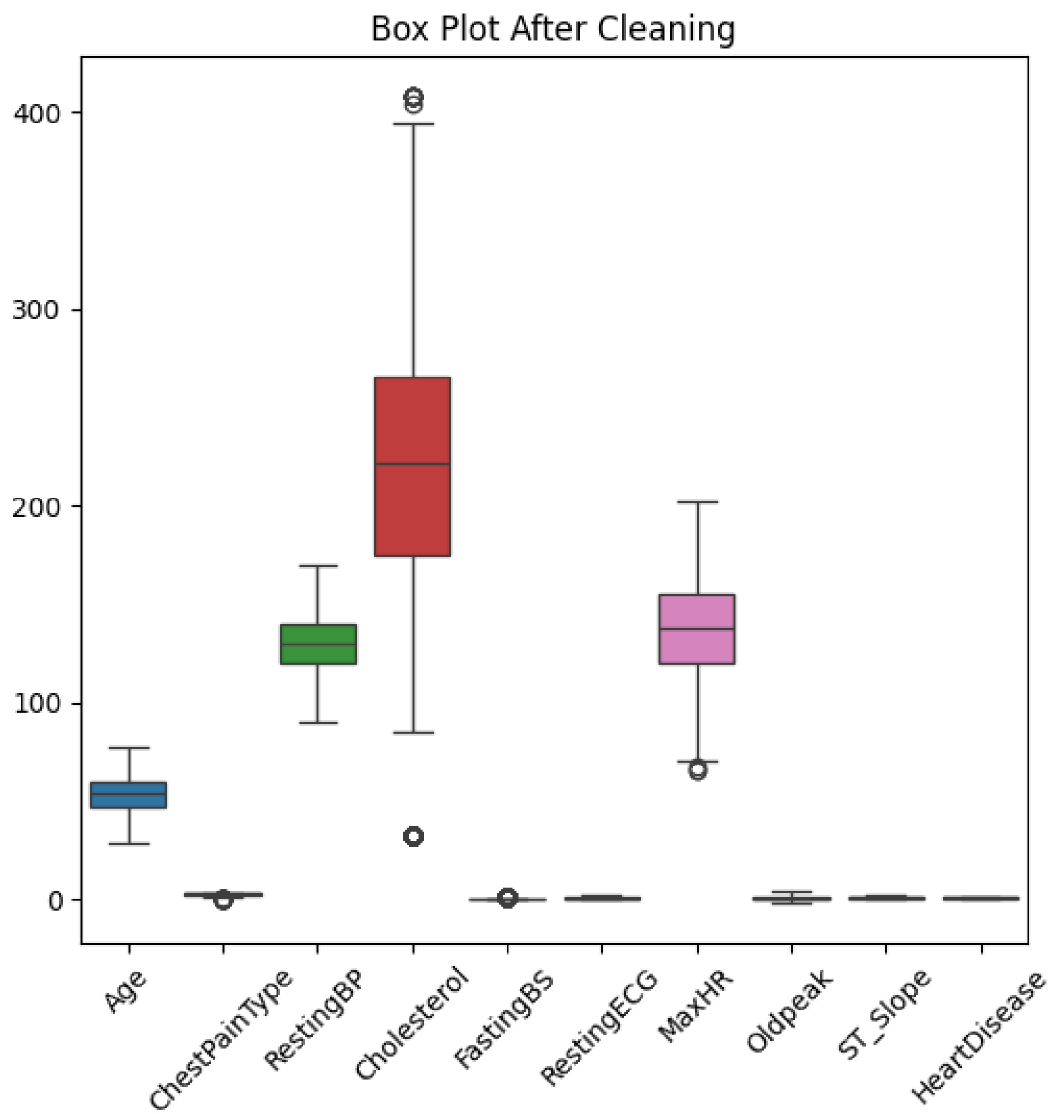


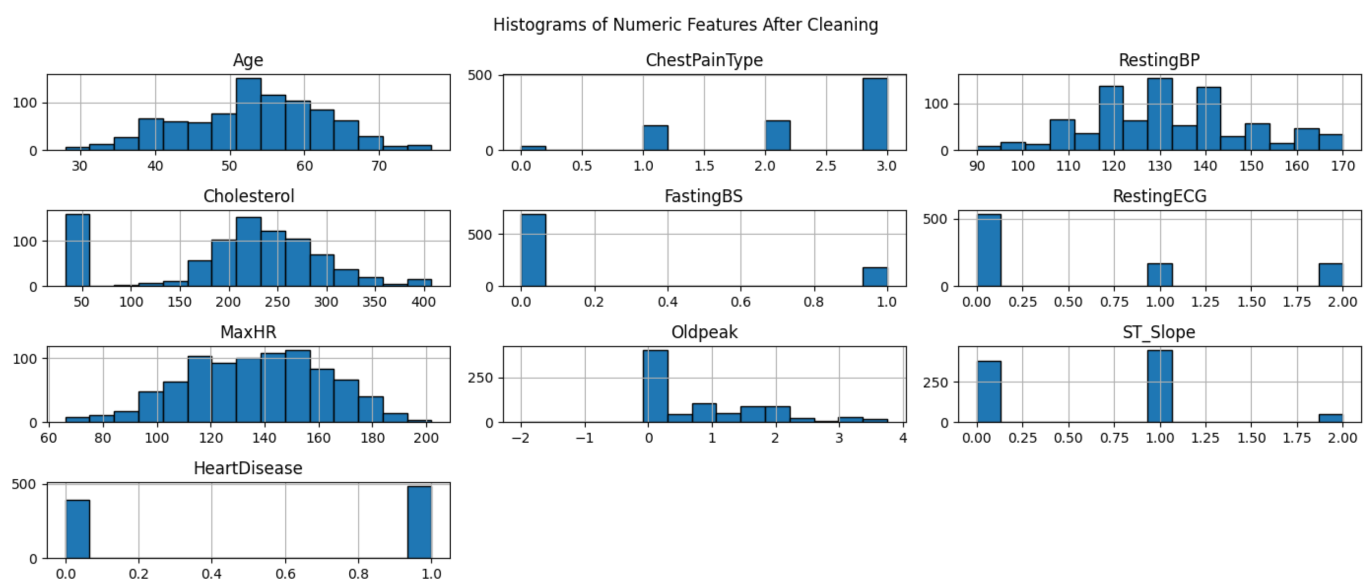

```
#Standardization for Consistency

# Convert Sex and ExerciseAngina to categorical
df['Sex'] = df['Sex'].replace({0: 'Female', 1: 'Male'})
df['ExerciseAngina'] = df['ExerciseAngina'].replace({0: 'No', 1: 'Yes'})
```

```
# Handling Duplicates

# Check and remove exact duplicates
df.drop_duplicates(inplace=True)
```

```
# After Cleaning: Visualize Outliers Again
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
sns.boxplot(data=df[numeric_columns])
plt.title('Box Plot After Cleaning')
plt.xticks(rotation=45)
```

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
 [Text(0, 0, 'Age'),
  Text(1, 0, 'ChestPainType'),
  Text(2, 0, 'RestingBP'),
  Text(3, 0, 'Cholesterol'),
  Text(4, 0, 'FastingBS'),
  Text(5, 0, 'RestingECG'),
  Text(6, 0, 'MaxHR'),
  Text(7, 0, 'Oldpeak'),
  Text(8, 0, 'ST_Slope'),
  Text(9, 0, 'HeartDisease')])
```


Box Plot After Cleaning
400
300
200
100
0
Age
ChestPainType
RestingBP
Cholesterol
FastingBS
RestingECG
MaxHR
Oldpeak
ST_Slope
HeartDisease


Histograms of Numeric Features After Cleaning
Age
ChestPainType
RestingBP
Cholesterol
FastingBS
RestingECG
MaxHR
Oldpeak
ST_Slope
HeartDisease

References :

https://medium.easyread.co/basics-of-data-preprocessing-71c314bc7188
https://medium.com/womenintechnology/data-preprocessing-steps-for-machine-learning- in-phyton-part-1-18009c6f1153

Github- https://github.com/Vishalgodalkar

Conclusion :

In this assignment, we conducted data cleaning and error correction on a heart disease dataset comprising 918 entries. We handled missing values using median imputation and identified outliers with the Isolation Forest algorithm, marking them as NaN for subsequent imputation. Visualization techniques were employed to validate the effectiveness of our cleaning process, ultimately improving the dataset's quality. This enhancement ensures the dataset is more suitable for reliable analysis and predictive modeling concerning heart disease risk.