```
import pandas as pd
import numpy as np
from tabulate import tabulate
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Load the dataset from the CSV file
dataset = pd.read_csv('/content/heart_attack_prediction_dataset.csv')
continentcsv = pd.read_csv('continent.csv')
df = pd.read_csv('heart_attack_prediction_dataset.csv')

names= pd.read_csv('/content/names.csv')
```

New Section

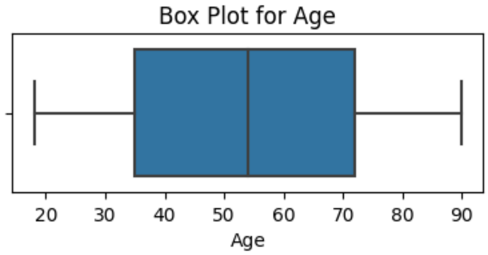
Phase 1

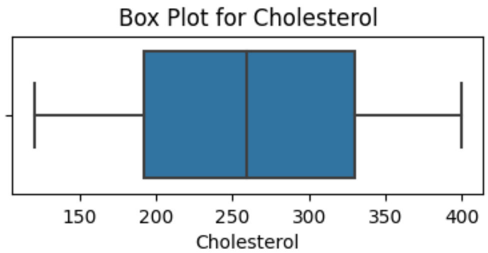
```
# List of columns to analyze

columns_to_analyze = [
    'Age', 'Sex', 'Cholesterol', 'Blood Pressure', 'Heart Rate', 'Diabetes',
    'Family History', 'Smoking', 'Obesity', 'Alcohol Consumption',
    'Exercise Hours Per Week', 'Diet', 'Previous Heart Problems', 'Medication Use',
    'Stress Level', 'Sedentary Hours Per Day', 'Income', 'BMI', 'Triglycerides',
    'Physical Activity Days Per Week', 'Sleep Hours Per Day', 'Country',
    'Continent', 'Hemisphere', 'Heart Attack Risk'
]

# Create a dictionary to store information for each column
column_info = {'Name of data': [], 'Type of data': [], 'Range of data': [],
               'Min': [], 'Max': [], 'Mean': [], 'Mode': [], 'Median': [], 'Outliers': []}

# Analyze each column
for column_name in columns_to_analyze:
    column_data = dataset[column_name]
    data_type = column_data.dtype

    if pd.api.types.is_numeric_dtype(data_type):
        data_summary = column_data.describe()
        mode_value = column_data.mode().values[0]
        median_value = column_data.median()

        Q1 = column_data.quantile(0.25)
        Q3 = column_data.quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        outliers = column_data[(column_data < lower_bound) | (column_data > upper_bound)]


        # Add information to the dictionary
        column_info['Name of data'].append(column_name)
        column_info['Type of data'].append(data_type)
        column_info['Range of data'].append(f"{column_data.min()} - {column_data.max()}")
        column_info['Min'].append(column_data.min())
        column_info['Max'].append(column_data.max())
        column_info['Mean'].append(data_summary['mean'])
        column_info['Mode'].append(mode_value)
        column_info['Median'].append(median_value)
        column_info['Outliers'].append(', '.join(map(str, outliers.tolist())))
    else:
        mode_value = column_data.mode().values[0]
        range_value = column_data.unique()
        # For non-numeric columns, provide some basic information
        column_info['Name of data'].append(column_name)
        column_info['Type of data'].append(data_type)
        column_info['Range of data'].append(range_value)
        column_info['Min'].append('Not applicable')
        column_info['Max'].append('Not applicable')
        column_info['Mean'].append('Not applicable')
        column_info['Mode'].append(mode_value)
        column_info['Median'].append('Not applicable')
        column_info['Outliers'].append('Not applicable')

# Create a table
table = {key: column_info[key] for key in column_info.keys()}

# Display the results in a table
print(tabulate(table, headers='keys', tablefmt='pretty'))
```

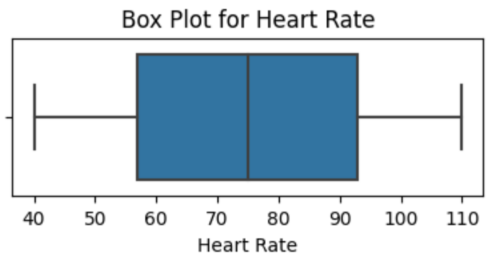
Name of data	Type of data	Range of data	Min
Age	float64	18.0 - 230.0	18.0
Sex	object	['Male' 'x' 'Female' nan]	Not applicabl
Cholesterol	float64	120.0 - 400.0	120.0
Blood Pressure	object	['158/88' '165/93' '174/99' ... '137/94' '94/76' '119/67']	Not applicabl
Heart Rate	float64	40.0 - 110.0	40.0
Diabetes	int64	0 - 1	0
Family History	int64	0 - 1	0
Smoking	int64	0 - 1	0
Obesity	int64	0 - 1	0
Alcohol Consumption	int64	0 - 1	0
Exercise Hours Per Week	float64	0.002442348 - 19.99870905	0.002442348

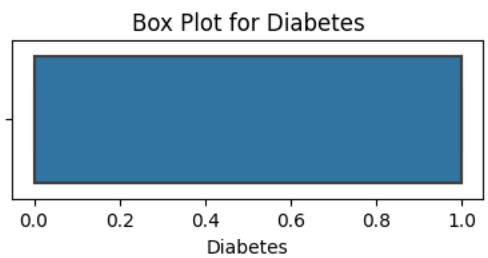
```
            Diet              |    object    |                  ['Average' 'Unhealthy' 'Healthy']                   | Not applicabl
   Previous Heart Problems    |    int64     |                                0 - 1                                 |       0
       Medication Use         |    int64     |                                0 - 1                                 |       0
        Stress Level          |    int64     |                                1 - 10                                |       1
   Sedentary Hours Per Day    |   float64    |                      0.001263206 - 11.99931341                       |  0.001263206
           Income             |    int64     |                           20062 - 299954                             |     20062
             BMI              |   float64    |                       18.00233658 - 39.99721082                      |  18.00233658
        Triglycerides         |    int64     |                               30 - 800                               |      30
ysical Activity Days Per Week |    int64     |                                0 - 7                                 |       0
     Sleep Hours Per Day      |    int64     |                                4 - 10                                |       4
           Country            |    object    |  ['Argentina' 'Canada' 'France' 'Thailand' 'Germany' 'Japan' 'Brazil'  | Not applicabl
                              |              | 'South Africa' 'United States' 'Vietnam' 'China' 'Italy' 'Spain' 'India' |
                              |              |       'Nigeria' 'New Zealand' 'South Korea' 'Australia' 'Colombia'     |
                              |              |                           'United Kingdom']                           |
          Continent           |    object    | ['South America' 'North America' 'Europe' 'Asia' 'Africa' 'Australia'] | Not applicabl
         Hemisphere           |    object    |            ['Southern Hemisphere' 'Northern Hemisphere']              | Not applicabl
      Heart Attack Risk       |    int64     |                                0 - 1                                 |       0
------------------------------+--------------+------------------------------------------------------------------------+-------------
```

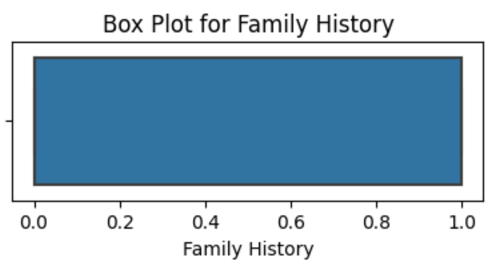
```
# Identify numeric columns
numeric_columns = dataset.select_dtypes(include=['number']).columns

# Set up subplots
fig, axes = plt.subplots(nrows=len(numeric_columns), figsize=(4, 2 * len(numeric_columns)))

# Create box plots for each numeric column
for i, column_name in enumerate(numeric_columns):
    sns.boxplot(x=dataset[column_name], ax=axes[i])
    axes[i].set_title(f'Box Plot for {column_name}')

# Adjust layout
plt.tight_layout()
plt.show()
```

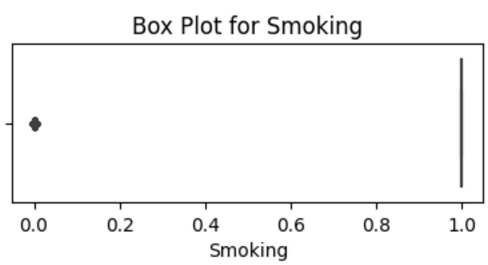

Box Plot for Age
20 30 40 50 60 70 80 90
Age

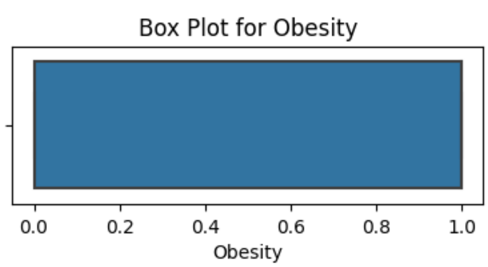

Box Plot for Cholesterol
150 200 250 300 350 400
Cholesterol


Box Plot for Heart Rate
40 50 60 70 80 90 100 110
Heart Rate

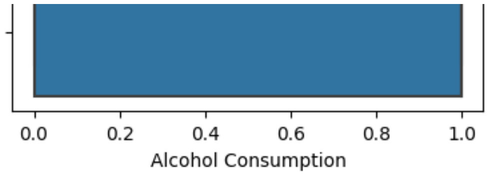

Box Plot for Diabetes
0.0 0.2 0.4 0.6 0.8 1.0
Diabetes


Box Plot for Family History
0.0 0.2 0.4 0.6 0.8 1.0
Family History


Box Plot for Smoking
0.0 0.2 0.4 0.6 0.8 1.0
Smoking

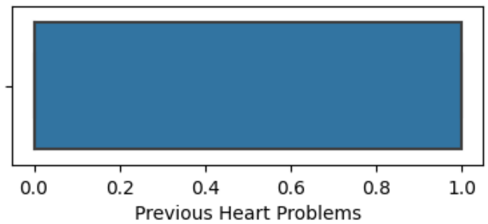

Box Plot for Obesity
0.0 0.2 0.4 0.6 0.8 1.0
Obesity


Box Plot for Alcohol Consumption

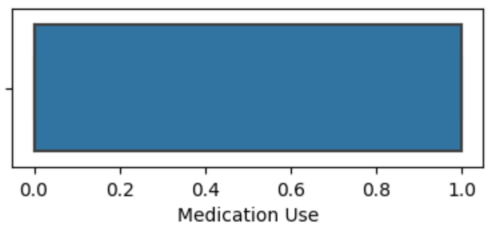


Box Plot for Exercise Hours Per Week

Box Plot for Previous Heart Problems

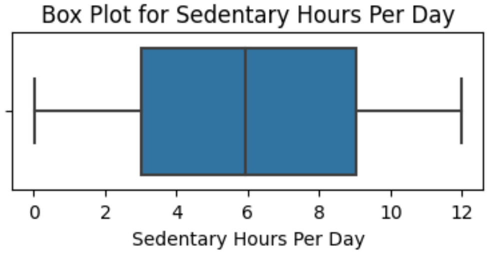


Box Plot for Medication Use

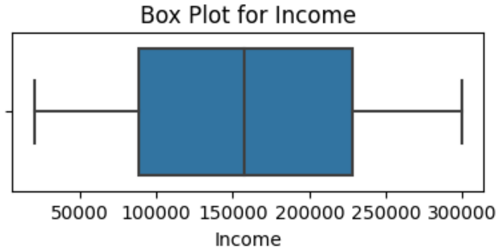


Box Plot for Stress Level

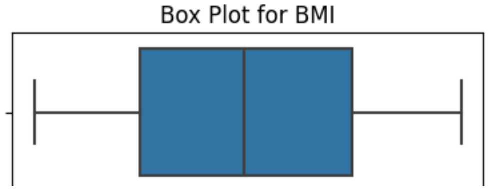
Box Plot for Sedentary Hours Per Day

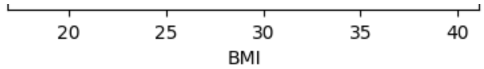


Box Plot for Income



Box Plot for BMI





Box Plot for Triglycerides

Phase 1 part 3

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load your dataset
# Replace '/path/to/your/dataset.csv' with the actual path to your dataset
dataset_path = '/content/heart_attack_prediction_dataset.csv'
df = pd.read_csv(dataset_path)

# Select relevant features
selected_features = ['Age', 'Exercise Hours Per Week', 'Physical Activity Days Per Week', 'Stress Level']

# Drop rows with missing values in selected features
df_selected = df[selected_features + ['Heart Attack Risk']].dropna()

# Define criteria for a healthy lifestyle (you can adjust these criteria)
healthy_criteria = ((df_selected['Age'] < 40) &
                    (df_selected['Exercise Hours Per Week'] >= 3) &
                    (df_selected['Physical Activity Days Per Week'] >= 4) &
                    (df_selected['Stress Level'] <= 3))

# Add a column indicating whether the person is living a healthy life
df_selected['Healthy Lifestyle'] = healthy_criteria.astype(int)

# Split the dataset into features and target
X = df_selected[selected_features]
y = df_selected['Healthy Lifestyle']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a RandomForestClassifier
classifier = RandomForestClassifier(random_state=42)
classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = classifier.predict(X_test)

# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Add the 'Healthy Lifestyle' column to the original dataset based on 'df_selected' Patient IDs
# df['Healthy Lifestyle'] = df.set_index('Patient ID').loc[df_selected.set_index('Patient ID').index]['Healthy Lifestyle'].values
df['healthy_criteria']=healthy_criteria
print(df)
# Display the updated dataset
# print(df['healthy_criteria'].sum())
```

```
     8759    QSV6764   28.0  Female          120        157/102          73
     8760    XKA5925   47.0    Male          250         161/75         105
     8761    EPE6801   36.0    Male          178         119/67          60
     8762    ZWN9666   25.0  Female          356         138/67          75

           Diabetes  Family History  Smoking  Obesity  ...  Income        BMI  \
     0            0               0        1        0  ...  261404  31.251233
     1            1               1        1        1  ...  285768  27.194973
     2            1               0        0        0  ...  235282  28.176571
     3            1               1        1        0  ...  125640  36.464704
     4            1               1        1        1  ...  160555  21.809144
```

```
1           235                             1                 7
2           587                             4                 4
3           378                             3                 4
4           231                             1                 5
...         ...                           ...               ...
8758         67                             7                 7
8759        617                             4                 9
8760        527                             4                 4
8761        114                             2                 8
8762        180                             7                 4

             Country      Continent           Hemisphere  Heart Attack Risk  \
0          Argentina  South America  Southern Hemisphere                  0
1             Canada  North America  Northern Hemisphere                  0
2             France         Europe  Northern Hemisphere                  0
3             Canada  North America  Northern Hemisphere                  0
4           Thailand           Asia  Northern Hemisphere                  0
...              ...            ...                  ...                ...
8758        Thailand           Asia  Northern Hemisphere                  0
8759          Canada  North America  Northern Hemisphere                  0
8760          Brazil  South America  Southern Hemisphere                  1
8761          Brazil  South America  Southern Hemisphere                  0
8762  United Kingdom         Europe  Northern Hemisphere                  1

      healthy_criteria
0                False
1                False
2                False
3                  NaN
4                False
...                ...
8758             False
8759             False
8760             False
8761             False
8762             False

[8763 rows x 27 columns]
```

Phase 2 Part 1

```
import re

# Create a table to store the results
table_data = []


# Validation function for 'Patient ID'
def validate_patient_id(patient_id):
    pattern = re.compile(r'^[A-Za-z]{3}\d{4}$')
    return bool(pattern.match(str(patient_id)))
# Check the uniqueness of 'Patient ID'
is_unique = df['Patient ID'].nunique() / len(df)
# Find non-unique 'Patient IDs'
non_unique_patient_ids = df[df.duplicated(subset='Patient ID', keep=False)]['Patient ID']
# Attribute: 'Patient ID'
column_name = 'Patient ID'
# Number of Data
num_data = len(df)
# Null
num_null = df[column_name].isnull().sum()
# Validity based on the specified pattern
validity = (df[column_name].apply(validate_patient_id).sum() / num_data) * 100 if num_data > 0 else None
# Completeness
completeness = (1 - (num_null / num_data)) * 100
# Consistency (Uniqueness)
consistency = is_unique * 100
# Append the results to the table_data list
table_data.append([column_name, num_data, num_null, f"{validity:.2f}%","-", f"{completeness:.2f}%", f"{consistency:.2f}%","-"])
# Create a DataFrame from the table_data list
result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy", "Completeness",
# Display the result table using tabulate for a nice format
#print(tabulate(result_table, headers='keys', tablefmt='pretty'))
# Print non-unique 'Patient IDs'
#print("\nNon-unique Patient IDs:")
#print(non_unique_patient_ids)



# Validation function for 'Age'
def validate_age(age):
    return 0 <= age <= 150 if pd.notna(age) else None
# Attribute: 'Age'
column_name = 'Age'
# Number of Data
num_data = df[column_name].count()
# Null
num_null = df[column_name].isnull().sum()
# Completeness
completeness = (1 - (num_null / num_data)) * 100
# Validity based on the specified pattern
validity = (df[column_name].apply(validate_age).sum() / num_data) * 100 if num_data > 0 else None
# Accuracy
accuracy_series = df['Age'].apply(lambda x: 0 <= x <= 150 if pd.notna(x) else None)
accuracy = accuracy_series.all()
# Accuracy Percentage
accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
# Append the results to the table_data list
table_data.append([column_name, num_data, num_null, f"{validity:.2f}%", f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
# Create a DataFrame from the table_data list
result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness",
# Print rows where 'Age' does not match the criteria
#invalid_age_rows = df[~df['Age'].apply(validate_age)]
#print("\nRows where 'Age' does not match the criteria:")
#print(tabulate(invalid_age_rows, headers='keys', tablefmt='pretty'))



# Validation function for 'Sex'
def validate_sex(sex):
    return sex.lower() in ['male', 'female'] if pd.notna(sex) else None
# Attribute: 'Sex'
column_name = 'Sex'
# Number of Data
num_data = df[column_name].count()
# Null
num_null = df[column_name].isnull().sum()
# Completeness
```

```
completeness = (1 - (num_null / num_data)) * 100
# Validity based on the specified pattern
validity = (df[column_name].apply(validate_sex).sum() / num_data) * 100 if num_data > 0 else None
# Accuracy
accuracy_series = df['Sex'].apply(lambda x: x.lower() in ['male', 'female'] if pd.notna(x) else None)
# Accuracy Percentage
accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
# Append the results to the table_data list
table_data.append([column_name, num_data, num_null, f"{validity:.2f}%" , f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
# Create a DataFrame from the table_data list
result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness",
# Print rows where 'Sex' does not match the criteria
#invalid_sex_rows = df[~df['Sex'].apply(validate_sex)]
#print("\nRows where 'Sex' does not match the criteria:")
#print(tabulate(invalid_sex_rows, headers='keys', tablefmt='pretty'))


# Accurate function for 'Blood Pressure'
def accurate_blood_pressure(blood_pressure):
    if pd.notna(blood_pressure):
        parts = blood_pressure.split('/')
        if len(parts) == 2:
            try:
                x, y = map(int, parts)
                return x > y
            except ValueError:
                return False
    return False
# Validation function for 'Blood Pressure'
def validate_blood_pressure(blood_pressure):
    if pd.notna(blood_pressure):
        parts = blood_pressure.split('/')
        if len(parts) == 2:
           return True

    return False
# Attribute: 'Blood Pressure'
column_name = 'Blood Pressure'
# Number of Data
num_data = df[column_name].count()
# Null
num_null = df[column_name].isnull().sum()
# Completeness
completeness = (1 - (num_null / num_data)) * 100
# Validity based on the specified pattern
validity = (df[column_name].apply(validate_blood_pressure).sum() / num_data) * 100 if num_data > 0 else None
# Accuracy
accuracy_series = df['Blood Pressure'].apply(accurate_blood_pressure)
# Accuracy Percentage
accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
# Append the results to the table_data list
table_data.append([column_name, num_data, num_null, f"{validity:.2f}%", f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
# Create a DataFrame from the table_data list
result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness",
```

```
column_names = ['Diabetes','Previous Heart Problems','Family History','Smoking','Obesity','Alcohol Consumption','Medication Use','Heart Att

for column_name in column_names:
  def validate_pre(pre):
      return 0 <= pre <= 1 if pd.notna(pre) else None
  def accurate_pre(pre):
      return 0 <= pre <= 1 if pd.notna(pre) else None
  # Number of Data
```

```
    num_data = df[column_name].count()
    # Null
    num_null = df[column_name].isnull().sum()
    # Completeness
    completeness = (1 - (num_null / num_data)) * 100
    # Validity based on the specified pattern
    validity = (df[column_name].apply(validate_pre).sum() / num_data) * 100 if num_data > 0 else None
    # Accuracy
    accuracy_series =  df[column_name].apply(accurate_pre)
    # Accuracy Percentage
    accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
    # Append the results to the table_data list
    table_data.append([column_name, num_data, num_null, f"{validity:.2f}%" , f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
    # Create a DataFrame from the table_data list
    result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness"


  def validate_str(pre):
      return 1 <= pre <= 10 if pd.notna(pre) else None
  def accurate_str(pre):
      return 1 <= pre <= 10 if pd.notna(pre) else None
  column_name = "Stress Level"
  # Number of Data
  num_data = df[column_name].count()
  # Null
  num_null = df[column_name].isnull().sum()
  # Completeness
  completeness = (1 - (num_null / num_data)) * 100
  # Validity based on the specified pattern
  validity = (df[column_name].apply(validate_str).sum() / num_data) * 100 if num_data > 0 else None
  # Accuracy
  accuracy_series =  df[column_name].apply(accurate_str)
  # Accuracy Percentage
  accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
  # Append the results to the table_data list
  table_data.append([column_name, num_data, num_null, f"{validity:.2f}%" , f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
  # Create a DataFrame from the table_data list
  result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness",



  def validate_sed(pre):
      return 0 <= pre <= 24 if pd.notna(pre) else None
  def accurate_sed(pre):
      return 0 <= pre <= 24 if pd.notna(pre) else None
  column_name = "Sedentary Hours Per Day"
  # Number of Data
  num_data = df[column_name].count()
  # Null
  num_null = df[column_name].isnull().sum()
  # Completeness
  completeness = (1 - (num_null / num_data)) * 100
  # Validity based on the specified pattern
  validity = (df[column_name].apply(validate_sed).sum() / num_data) * 100 if num_data > 0 else None
  # Accuracy
  accuracy_series =  df[column_name].apply(accurate_sed)
  # Accuracy Percentage
  accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
  # Append the results to the table_data list
  table_data.append([column_name, num_data, num_null, f"{validity:.2f}%" , f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
  # Create a DataFrame from the table_data list
  result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness",




  def validate_in(inc):
      return 0 <= inc if pd.notna(inc) else None
  def accurate_in(inc):
      return 0 <= inc  if pd.notna(inc) else None
  column_name = "Income"
  # Number of Data
  num_data = df[column_name].count()
  # Null
  num_null = df[column_name].isnull().sum()
  # Completeness
  completeness = (1 - (num_null / num_data)) * 100
```

```
# Validity based on the specified pattern
validity = (df[column_name].apply(validate_in).sum() / num_data) * 100 if num_data > 0 else None
# Accuracy
accuracy_series =  df[column_name].apply(accurate_in)
# Accuracy Percentage
accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
# Append the results to the table_data list
table_data.append([column_name, num_data, num_null, f"{validity:.2f}%" , f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
# Create a DataFrame from the table_data list
result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness",



def validate_bmi(bmi):
    return 0 <= bmi if pd.notna(bmi) else None
def accurate_bmi(bmi):
    return 9 <= bmi <= 105  if pd.notna(bmi) else None
column_name = "BMI"
# Number of Data
num_data = df[column_name].count()
# Null
num_null = df[column_name].isnull().sum()
# Completeness
completeness = (1 - (num_null / num_data)) * 100
# Validity based on the specified pattern
validity = (df[column_name].apply(validate_bmi).sum() / num_data) * 100 if num_data > 0 else None
# Accuracy
accuracy_series =  df[column_name].apply(accurate_bmi)
# Accuracy Percentage
accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
# Append the results to the table_data list
table_data.append([column_name, num_data, num_null, f"{validity:.2f}%" , f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
# Create a DataFrame from the table_data list
result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness",


def validate_T(bmi):
    return 0 <= bmi if pd.notna(bmi) else None
def accurate_T(bmi):
    return 0 <= bmi   if pd.notna(bmi) else None
column_name = "Triglycerides"
# Number of Data
num_data = df[column_name].count()
# Null
num_null = df[column_name].isnull().sum()
# Completeness
completeness = (1 - (num_null / num_data)) * 100
# Validity based on the specified pattern
validity = (df[column_name].apply(validate_T).sum() / num_data) * 100 if num_data > 0 else None
# Accuracy
accuracy_series =  df[column_name].apply(accurate_T)
# Accuracy Percentage
accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
# Append the results to the table_data list
table_data.append([column_name, num_data, num_null, f"{validity:.2f}%" , f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
# Create a DataFrame from the table_data list
result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness",


def validate_P(bmi):
    return 0 <= bmi <= 7 if pd.notna(bmi) else None
def accurate_P(bmi):
    return 0 <= bmi <= 7   if pd.notna(bmi) else None
column_name = "Physical Activity Days Per Week"
# Number of Data
num_data = df[column_name].count()
# Null
num_null = df[column_name].isnull().sum()
# Completeness
completeness = (1 - (num_null / num_data)) * 100
# Validity based on the specified pattern
validity = (df[column_name].apply(validate_P).sum() / num_data) * 100 if num_data > 0 else None
# Accuracy
accuracy_series =  df[column_name].apply(accurate_P)
# Accuracy Percentage
accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
# Append the results to the table_data list
```

```
table_data.append([column_name, num_data, num_null, f"{validity:.2f}%" , f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
# Create a DataFrame from the table_data list
result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness",


def validate_S(bmi):
    return 0 <= bmi <= 24 if pd.notna(bmi) else None
def accurate_S(bmi):
    return 0 <= bmi <= 24 if pd.notna(bmi) else None
column_name = "Sleep Hours Per Day"
# Number of Data
num_data = df[column_name].count()
# Null
num_null = df[column_name].isnull().sum()
# Completeness
completeness = (1 - (num_null / num_data)) * 100
# Validity based on the specified pattern
validity = (df[column_name].apply(validate_S).sum() / num_data) * 100 if num_data > 0 else None
# Accuracy
accuracy_series =  df[column_name].apply(accurate_S)
# Accuracy Percentage
accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
# Append the results to the table_data list
table_data.append([column_name, num_data, num_null, f"{validity:.2f}%" , f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
# Create a DataFrame from the table_data list
result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness",



def validate_C(bmi):
    return 0 <= len(bmi) <= 90   if pd.notna(bmi) else None
def accurate_C(bmi):
    return 3 <= len(bmi) <= 56 if pd.notna(bmi) else None
column_name = "Country"
# Number of Data
num_data = df[column_name].count()
# Null
num_null = df[column_name].isnull().sum()
# Completeness
completeness = (1 - (num_null / num_data)) * 100
# Validity based on the specified pattern
validity = (df[column_name].apply(validate_C).sum() / num_data) * 100 if num_data > 0 else None
# Accuracy
accuracy_series =  df[column_name].apply(accurate_C)
# Accuracy Percentage
accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
# Append the results to the table_data list
table_data.append([column_name, num_data, num_null, f"{validity:.2f}%" , f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
# Create a DataFrame from the table_data list
result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness",


def validate_con(bmi):
    return bmi in ['Europe', 'Australia' , 'Asia' , 'Africa' , 'North America' , 'South America'] if pd.notna(bmi) else None
def accurate_con(bmi):
    return bmi in ['Europe', 'Australia' , 'Asia' , 'Africa' , 'North America' , 'South America'] if pd.notna(bmi) else None
column_name = "Continent"
# Number of Data
num_data = df[column_name].count()
# Null
num_null = df[column_name].isnull().sum()
# Completeness
completeness = (1 - (num_null / num_data)) * 100
# Validity based on the specified pattern
validity = (df[column_name].apply(validate_con).sum() / num_data) * 100 if num_data > 0 else None
# Accuracy
accuracy_series =  df[column_name].apply(accurate_con)
# Accuracy Percentage
accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
# Append the results to the table_data list
table_data.append([column_name, num_data, num_null, f"{validity:.2f}%" , f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
# Create a DataFrame from the table_data list
result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness",
```

```
def validate_hem(bmi):
    return bmi in ['Northern Hemisphere', 'Southern Hemisphere'] if pd.notna(bmi) else None
def accurate_hem(bmi):
    return bmi in ['Northern Hemisphere', 'Southern Hemisphere'] if pd.notna(bmi) else None
column_name = "Hemisphere"
# Number of Data
num_data = df[column_name].count()
# Null
num_null = df[column_name].isnull().sum()
# Completeness
completeness = (1 - (num_null / num_data)) * 100
# Validity based on the specified pattern
validity = (df[column_name].apply(validate_hem).sum() / num_data) * 100 if num_data > 0 else None
# Accuracy
accuracy_series =  df[column_name].apply(accurate_hem)
# Accuracy Percentage
accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
# Append the results to the table_data list
table_data.append([column_name, num_data, num_null, f"{validity:.2f}%" , f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
# Create a DataFrame from the table_data list
result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness",




# Display the result table using tabulate for a nice format
print(tabulate(result_table, headers='keys', tablefmt='pretty'))

# Print rows where 'Blood Pressure' does not match the criteria
#invalid_blood_pressure_rows = df[~df['Blood Pressure'].apply(validate_blood_pressure)]
#print("\nRows where 'Blood Pressure' does not match the criteria:")
#print(tabulate(invalid_blood_pressure_rows, headers='keys', tablefmt='pretty'))
```

```
+----+-------------------------------+----------------+------+----------+----------+--------------+-------------+-------------+
|    |     Name of the Attribute     | Number of Data | Null | Validity | Accuracy | Completeness | Consistency | Currentness |
+----+-------------------------------+----------------+------+----------+----------+--------------+-------------+-------------+
| 0  |          Patient ID           |      8763      |  0   |  99.93%  |    -     |   100.00%    |   99.98%    |      -      |
| 1  |              Age              |      8761      |  2   |  99.99%  |  99.99%  |    99.98%    |      -      |      -      |
| 2  |              Sex              |      8760      |  3   |  99.99%  |  99.99%  |    99.97%    |      -      |      -      |
| 3  |        Blood Pressure         |      8763      |  0   | 100.00%  |  95.08%  |   100.00%    |      -      |      -      |
| 4  |           Diabetes            |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 5  |    Previous Heart Problems    |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 6  |        Family History         |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 7  |            Smoking            |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 8  |            Obesity            |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 9  |      Alcohol Consumption      |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 10 |        Medication Use         |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 11 |       Heart Attack Risk       |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 12 |         Stress Level          |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 13 |    Sedentary Hours Per Day    |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 14 |            Income             |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 15 |              BMI              |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 16 |         Triglycerides         |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 17 | Physical Activity Days Per Week |    8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 18 |      Sleep Hours Per Day      |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 19 |            Country            |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 20 |           Continent           |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
| 21 |          Hemisphere           |      8763      |  0   | 100.00%  | 100.00%  |   100.00%    |      -      |      -      |
+----+-------------------------------+----------------+------+----------+----------+--------------+-------------+-------------+
```

Patient Id validity 3 three letters and 4 digits Consistency unique **Age** validity 0 < Age accuracy 0 < x < 180 **Sex** validity Male and Female **Cholesterol Blood Pressure** validity format x/y x>y**Diabetes Family-History Smoking Obesity Alcohol Consumption** validity 0-1 **Exercise Hours Per Week** validity 0-168 Accuracy 0-100**Diet** validity Unhealthy - Average - Healthy **Previous Heart Problems Medication Use** validity 0-1 **Stress Level** validity 1-10 **Sedentary** Hours Per Day validity 0-24 **income** validity digits only **BMI** validity 0 < x Accuracy 9 < x < 105 **Triglycerides** validity 0 < x **Physical Activity Days Per Week** validity 0 <= x <= 7 **Sleep Hours Per Day** validity 0 <= x <= 24 **Heart Attack Risk** validity 0-1 x

```
import pandas as pd
from tabulate import tabulate

# Load your dataset
# Replace '/path/to/your/dataset.csv' with the actual path to your dataset
```

```
dataset_path = '/content/heart_attack_prediction_dataset.csv'
df = pd.read_csv(dataset_path)

# Select the numeric column for outlier removal
numeric_column = 'Age'

# Calculate quartiles and IQR
Q1 = df[numeric_column].quantile(0.25)
Q3 = df[numeric_column].quantile(0.75)
IQR = Q3 - Q1

# Calculate lower and upper bounds
lower_bound = Q1 - 0.5 * IQR
upper_bound = Q3 + 0.5 * IQR

# Identify outliers
outliers = df[(df[numeric_column] < lower_bound) | (df[numeric_column] > upper_bound)]

# Display outliers in a table
outliers_table = tabulate(outliers, headers='keys', tablefmt='pretty')

print("Identified outliers:")
print(outliers_table)

# Remove outliers from the dataset
df_no_outliers = df[(df[numeric_column] >= lower_bound) & (df[numeric_column] <= upper_bound)]

# Display the updated dataset without outliers
print("\nDataset without outliers:")
print(df_no_outliers.head())
```

```
Identified outliers:
+---+------------+-------+------+-------------+----------------+------------+----------+----------------+---------+---------+----------
|   | Patient ID |  Age  | Sex  | Cholesterol | Blood Pressure | Heart Rate | Diabetes | Family History | Smoking | Obesity | Alcohol C
+---+------------+-------+------+-------------+----------------+------------+----------+----------------+---------+---------+----------
| 1 |  CZE1114   | 230.0 | Male |    389.0    |     165/93     |    98.0    |    1     |       1        |    1    |    1    |
+---+------------+-------+------+-------------+----------------+------------+----------+----------------+---------+---------+----------

Dataset without outliers:
  Patient ID   Age     Sex  Cholesterol Blood Pressure  Heart Rate  Diabetes  \
0     kjreer  67.0    Male        208.0         158/88        72.0         0
2    BNI9906  21.0       x        324.0         174/99        72.0         1
4    ZOO7941  66.0    Male        318.0          91/88        93.0         1
5    ZOO7941  54.0  Female        297.0         172/86        48.0         1
6    WYV0966  90.0    Male        358.0         102/73        84.0         0

   Family History  Smoking  Obesity  ...  Sedentary Hours Per Day  Income  \
0               0        1        0  ...                 6.615001  261404
2               0        0        0  ...                 9.463426  235282
4               1        1        1  ...                 1.514821  160555
5               1        1        0  ...                 7.798752  241339
6               0        1        0  ...                 0.627356  190450

         BMI  Triglycerides  Physical Activity Days Per Week  \
0  31.251233            286                                0
2  28.176571            587                                4
4  21.809144            231                                1
5  20.146839            795                                5
6  28.885811            284                                4

   Sleep Hours Per Day    Country      Continent           Hemisphere  \
0                    6  Argentina  South America  Southern Hemisphere
2                    4     France         Europe  Northern Hemisphere
4                    5   Thailand           Asia  Northern Hemisphere
5                   10    Germany         Europe  Northern Hemisphere
6                   10     Canada  North America  Northern Hemisphere

   Heart Attack Risk
0                  0
2                  0
4                  0
5                  1
6                  1

[5 rows x 26 columns]
```

```
import pandas as pd

# Load your dataset
```

```
# Load your dataset
# Replace '/path/to/your/dataset.csv' with the actual path to your dataset
dataset_path = '/content/heart_attack_prediction_dataset.csv'
df = pd.read_csv(dataset_path)

# List of columns to check for missing values
columns_to_check = [
    'Patient ID', 'Age', 'Sex', 'Cholesterol', 'Blood Pressure', 'Heart Rate',
    'Diabetes', 'Family History', 'Smoking', 'Obesity', 'Alcohol Consumption',
    'Exercise Hours Per Week', 'Diet', 'Previous Heart Problems', 'Medication Use',
    'Stress Level', 'Sedentary Hours Per Day', 'Income', 'BMI', 'Triglycerides',
    'Physical Activity Days Per Week', 'Sleep Hours Per Day', 'Country',
    'Continent', 'Hemisphere', 'Heart Attack Risk'
]

# Iterate over columns and check for missing values
missing_values_info = []
for column in columns_to_check:
    num_missing_values = df[column].isnull().sum()
    missing_values_info.append({'Column': column, 'Missing Values': num_missing_values})

# Create a DataFrame from the information
missing_values_df = pd.DataFrame(missing_values_info)

# Display the information about missing values
print(missing_values_df)
```

```
                             Column  Missing Values
0                        Patient ID               0
1                               Age               2
2                               Sex               3
3                       Cholesterol               1
4                    Blood Pressure               0
5                        Heart Rate               1
6                          Diabetes               0
7                    Family History               0
8                           Smoking               0
9                           Obesity               0
10              Alcohol Consumption               0
11          Exercise Hours Per Week               0
12                             Diet               0
13          Previous Heart Problems               0
14                   Medication Use               0
15                     Stress Level               0
16          Sedentary Hours Per Day               0
17                           Income               0
18                              BMI               0
19                    Triglycerides               0
20  Physical Activity Days Per Week               0
21              Sleep Hours Per Day               0
22                          Country               0
23                        Continent               0
24                       Hemisphere               0
25                Heart Attack Risk               0
```

```
import pandas as pd

# Load your dataset
# Replace '/path/to/your/dataset.csv' with the actual path to your dataset
dataset_path = '/content/heart_attack_prediction_dataset.csv'
df = pd.read_csv(dataset_path)

# List of columns to check for missing values
columns_to_check = [
    'Patient ID', 'Age', 'Sex', 'Cholesterol', 'Blood Pressure', 'Heart Rate',
    'Diabetes', 'Family History', 'Smoking', 'Obesity', 'Alcohol Consumption',
    'Exercise Hours Per Week', 'Diet', 'Previous Heart Problems', 'Medication Use',
    'Stress Level', 'Sedentary Hours Per Day', 'Income', 'BMI', 'Triglycerides',
    'Physical Activity Days Per Week', 'Sleep Hours Per Day', 'Country',
    'Continent', 'Hemisphere', 'Heart Attack Risk'
]

# Iterate over columns and check for missing values
missing_values_info = []
for column in columns_to_check:
    missing_rows = df[df[column].isnull()]
    if not missing_rows.empty:
        missing_values_info.append({'Column': column, 'Missing Rows': missing_rows})
    else:
        missing_values_info.append({'Column': column, 'Missing Rows': None})

# Display the information about missing values
for info in missing_values_info:
    print(f"\nColumn '{info['Column']}':")
    if info['Missing Rows'] is not None:
        print(f"Missing values located at:\n{info['Missing Rows']}")
    else:
        print("No missing values.")
```

```
Column 'Country':
No missing values.

Column 'Continent':
No missing values.

Column 'Hemisphere':
No missing values.

Column 'Heart Attack Risk':
No missing values.
```

checking the single schema problem

single shema problems

```
def validate_bmi(bmi):
    return 0 <= bmi if pd.notna(bmi) else None
def accurate_bmi(bmi):
    return 9 <= bmi <= 105  if pd.notna(bmi) else None
column_name = "BMI"
# Number of Data
num_data = df[column_name].count()
# Null
num_null = df[column_name].isnull().sum()
# Completeness
completeness = (1 - (num_null / num_data)) * 100
# Validity based on the specified pattern
validity = (df[column_name].apply(validate_bmi).sum() / num_data) * 100 if num_data > 0 else None
# Accuracy
accuracy_series =  df[column_name].apply(accurate_bmi)
# Accuracy Percentage
accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
# Append the results to the table_data list
table_data.append([column_name, num_data, num_null, f"{validity:.2f}%" , f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
# Create a DataFrame from the table_data list
result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness",


def validate_age(age):
    return 0 <= age <= 150 if pd.notna(age) else None
# Attribute: 'Age'
column_name = 'Age'
# Number of Data
num_data = df[column_name].count()
# Null
num_null = df[column_name].isnull().sum()
# Completeness
completeness = (1 - (num_null / num_data)) * 100
# Validity based on the specified pattern
validity = (df[column_name].apply(validate_age).sum() / num_data) * 100 if num_data > 0 else None
# Accuracy
accuracy_series = df['Age'].apply(lambda x: 0 <= x <= 150 if pd.notna(x) else None)
accuracy = accuracy_series.all()
# Accuracy Percentage
accuracy_percentage = (accuracy_series.sum() / num_data) * 100 if num_data > 0 else None
# Append the results to the table_data list
table_data.append([column_name, num_data, num_null, f"{validity:.2f}%", f"{accuracy_percentage:.2f}%",f"{completeness:.2f}%","-","-"])
# Create a DataFrame from the table_data list
result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy","Completeness",
# Print rows where 'Age' does not match the criteria

# invalid_age_rows = df[~df['Age'].apply(validate_age)]
# print("\nRows where 'Age' does not match the criteria:")
# print(tabulate(invalid_age_rows, headers='keys', tablefmt='pretty'))




def validate_patient_id(patient_id):
    pattern = re.compile(r'^[A-Za-z]{3}\d{4}$')
    return bool(pattern.match(str(patient_id)))
# Check the uniqueness of 'Patient ID'
is_unique = df['Patient ID'].nunique() / len(df)
# Find non-unique 'Patient IDs'
non_unique_patient_ids = df[df.duplicated(subset='Patient ID', keep=False)]['Patient ID']
# Attribute: 'Patient ID'
column_name = 'Patient ID'
# Number of Data
num_data = len(df)
# Null
num_null = df[column_name].isnull().sum()
# Validity based on the specified pattern
validity = (df[column_name].apply(validate_patient_id).sum() / num_data) * 100 if num_data > 0 else None
# Completeness
completeness = (1 - (num_null / num_data)) * 100
# Consistency (Uniqueness)
consistency = is_unique * 100
# Append the results to the table_data list
table_data.append([column_name, num_data, num_null, f"{validity:.2f}%","-", f"{completeness:.2f}%", f"{consistency:.2f}%","-"])
# Create a DataFrame from the table_data list
result_table = pd.DataFrame(table_data, columns=["Name of the Attribute", "Number of Data", "Null", "Validity", "Accuracy", "Completeness",
# Display the result table using tabulate for a nice format
```

```
#print(tabulate(result_table, headers='keys', tablefmt='pretty'))
# Print non-unique 'Patient IDs'
print("\nNon-unique Patient IDs:")
print(non_unique_patient_ids)
```

```
Non-unique Patient IDs:
4      ZOO7941
5      ZOO7941
539    DFO7465
540    DFO7465
Name: Patient ID, dtype: object
```

```
import pandas as pd
from tabulate import tabulate

# Load your dataset
# Replace '/path/to/your/dataset.csv' with the actual path to your dataset
dataset_path = '/content/heart_attack_prediction_dataset.csv'
df = pd.read_csv(dataset_path)

# List of columns to check for missing values, uniqueness, and validation
columns_to_check = [
    'Patient ID', 'Age', 'Sex', 'Cholesterol', 'Blood Pressure', 'Heart Rate',
    'Diabetes', 'Family History', 'Smoking', 'Obesity', 'Alcohol Consumption',
    'Exercise Hours Per Week', 'Diet', 'Previous Heart Problems', 'Medication Use',
    'Stress Level', 'Sedentary Hours Per Day', 'Income', 'BMI', 'Triglycerides',
    'Physical Activity Days Per Week', 'Sleep Hours Per Day', 'Country',
    'Continent', 'Hemisphere', 'Heart Attack Risk'
]

# Create a list to store the information for each column
column_info = []

# Iterate over columns and check for missing values, uniqueness, and validation
for column in columns_to_check:
    missing_rows = df[df[column].isnull()]
    duplicate_rows = df[df.duplicated(subset=[column], keep=False)]

    # Validation checks for specific columns
    validation_check = None
    if column == 'Age':
        validation_check = df['Age'].apply(lambda x: 0 <= x  if pd.notna(x) else None)
    elif column == 'BMI':
        validation_check = df['BMI'].apply(lambda x: 0 <= x if pd.notna(x) else None)

    info = {
        'Column': column,
        'Missing Rows': missing_rows.head(),
        'Duplicate Rows': duplicate_rows.head(),
        'Validation Check': validation_check
    }
    column_info.append(info)

# Display the information about missing values, uniqueness, and validation in a table
for info in column_info:
    print(f"\nColumn '{info['Column']}':")

    # Format missing values
    missing_values_table = tabulate(info['Missing Rows'], headers='keys', tablefmt='pretty')
    print(f"Missing values located at:\n{missing_values_table if info['Missing Rows'].shape[0] else 'No missing values.'}")

    # Format duplicate values
    # duplicate_values_table = tabulate(info['Duplicate Rows'], headers='keys', tablefmt='pretty')
    # print(f"Duplicated values located at:\n{duplicate_values_table if info['Duplicate Rows'].shape[0] else 'No duplicated values.'}")

    # Format validation check
    if info['Validation Check'] is not None:
        invalid_rows = df[~info['Validation Check'].fillna(False)]
        if not invalid_rows.empty:
            invalid_rows_table = tabulate(invalid_rows, headers='keys', tablefmt='pretty')
            print(f"Invalid rows based on validation check:\n{invalid_rows_table}")
        else:
            print("All rows are valid.")
    else:
        print("No validation check performed for this column.")
```

```
No missing values.
No validation check performed for this column.

Column 'Stress Level':
Missing values located at:
No missing values.
No validation check performed for this column.

Column 'Sedentary Hours Per Day':
Missing values located at:
No missing values.
No validation check performed for this column.

Column 'Income':
Missing values located at:
No missing values.
No validation check performed for this column.

Column 'BMI':
Missing values located at:
No missing values.
All rows are valid.

Column 'Triglycerides':
Missing values located at:
No missing values.
No validation check performed for this column.

Column 'Physical Activity Days Per Week':
Missing values located at:
No missing values.
No validation check performed for this column.

Column 'Sleep Hours Per Day':
Missing values located at:
No missing values.
No validation check performed for this column.

Column 'Country':
Missing values located at:
No missing values.
No validation check performed for this column.

Column 'Continent':
Missing values located at:
No missing values.
No validation check performed for this column.

Column 'Hemisphere':
Missing values located at:
No missing values.
No validation check performed for this column.

Column 'Heart Attack Risk':
Missing values located at:
No missing values.
No validation check performed for this column.
```

instant level single source

```
import pandas as pd

# Load your dataset
# Replace '/path/to/your/dataset.csv' with the actual path to your dataset
dataset_path = '/content/heart_attack_prediction_dataset.csv'
df = pd.read_csv(dataset_path)

# Check for duplicated rows
duplicated_rows = df[df.duplicated()]

# Check for duplicated columns
duplicated_columns = df.columns[df.columns.duplicated()]

# Calculate the correlation matrix
correlation_matrix = df.corr()

# Display the results
print("Duplicated Rows:")
print(duplicated_rows)
```

```
print("\nDuplicated Columns:")
print(duplicated_columns)

print("\nCorrelation Matrix:")
print(correlation_matrix)
```

```
Obesity                        -0.003870 -0.006058       0.001467
Alcohol Consumption            -0.022396  0.010562       0.006169
Exercise Hours Per Week        -0.023414  0.003777       0.001717
Previous Heart Problems        -0.003281  0.015718      -0.019029
Medication Use                 -0.003464  0.009514      -0.011095
Stress Level                   -0.002760 -0.003250      -0.003921
Sedentary Hours Per Day         0.003511 -0.000024      -0.005785
Income                          1.000000  0.008836       0.010739
BMI                             0.008836  1.000000      -0.005964
Triglycerides                   0.010739 -0.005964       1.000000
Physical Activity Days Per Week  0.000130  0.008110      -0.007556
Sleep Hours Per Day            -0.006598 -0.010030      -0.029216
Heart Attack Risk               0.009628  0.000020       0.010471

                                Physical Activity Days Per Week  \
Age                                                    0.000376
Cholesterol                                            0.015874
Heart Rate                                             0.000820
Diabetes                                              -0.002411
Family History                                         0.009561
Smoking                                               -0.006465
Obesity                                                0.005337
Alcohol Consumption                                    0.001593
Exercise Hours Per Week                                0.007725
Previous Heart Problems                                0.008537
Medication Use                                        -0.011139
Stress Level                                           0.007405
Sedentary Hours Per Day                               -0.006178
Income                                                 0.000130
BMI                                                    0.008110
Triglycerides                                         -0.007556
Physical Activity Days Per Week                        1.000000
Sleep Hours Per Day                                    0.014033
Heart Attack Risk                                     -0.005014

                                Sleep Hours Per Day  Heart Attack Risk
Age                                       -0.001774           0.005791
Cholesterol                                0.004540           0.019571
Heart Rate                                 0.001745          -0.004202
Diabetes                                  -0.012457           0.017225
Family History                            -0.011199          -0.001652
Smoking                                   -0.005424          -0.004051
Obesity                                   -0.005314          -0.013318
Alcohol Consumption                       -0.000843          -0.013778
Exercise Hours Per Week                   -0.001245           0.011133
Previous Heart Problems                    0.004460           0.000274
Medication Use                            -0.020393           0.002234
Stress Level                              -0.014205          -0.004111
Sedentary Hours Per Day                    0.004792          -0.005613
Income                                    -0.006598           0.009628
BMI                                       -0.010030           0.000020
Triglycerides                             -0.029216           0.010471
Physical Activity Days Per Week            0.014033          -0.005014
Sleep Hours Per Day                        1.000000          -0.018528
Heart Attack Risk                         -0.018528           1.000000
<ipython-input-15-6ba120e0b2a0>:15: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
  correlation_matrix = df.corr()
```

single source instant level

```
import pandas as pd

# Load your dataset
# Replace '/path/to/your/dataset.csv' with the actual path to your dataset
dataset_path = '/content/heart_attack_prediction_dataset.csv'
df = pd.read_csv(dataset_path)

# Check for duplicated rows
duplicated_rows = df[df.duplicated()]

# Check for duplicated columns
duplicated_columns = df.columns[df.columns.duplicated()]

# Calculate the correlation matrix
# correlation_matrix = df.corr()

# Display the redundant values
print("Duplicated Rows:")
print(duplicated_rows)

print("\nDuplicated Columns:")
print(duplicated_columns)

# print("\nCorrelation Matrix:")
# print(correlation_matrix)
```

```
Duplicated Rows:
Empty DataFrame
Columns: [Patient ID, Age, Sex, Cholesterol, Blood Pressure, Heart Rate, Diabetes, Family History, Smoking, Obesity, Alcohol Consumptio
Index: []

[0 rows x 26 columns]

Duplicated Columns:
Index([], dtype='object')
```

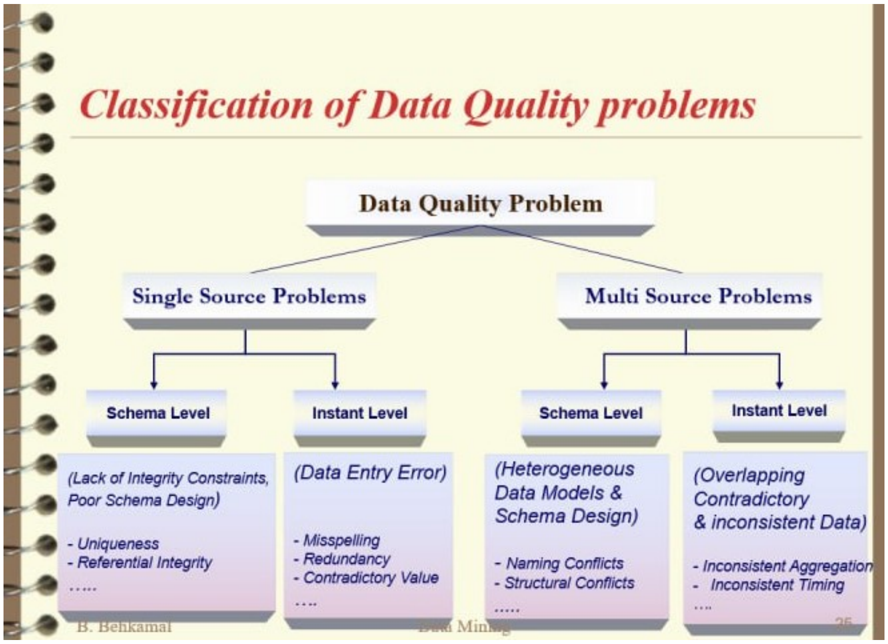
```
Single Schema:
          uniquness -> Patient Id should be unique
          Missing value -> some columns should be not null
          invalid value -> for example age should be between 0 until
          180

single instant level :
Misspelling value for continent
Misspelling value for Hemisphere
BMI > 105 is misspell

Multi source Schema level:
          Naming conflict between main dataset and continent dataset
          in Country column that we have in main dataset and nation i
          continent dataset and between Continent column and continen
          column
          naming conflict between Gender in names.csv and Sex in main
          csv
          structure conflict between Male and Female and 0 and 1

Multisource instant level:
Contradictory Data that have the same Country name but different
Continent
Country (main csv) is not in nation (continent csv)
the same country have two different name like "US" and "United States"
```

Single Schema: uniquness -> Patient Id should be unique Missing value -> some columns should be not null invalid value -> for example age should be between 0 until 180

single instant level : Misspelling value for continent Misspelling value for Hemisphere BMI > 105 is misspell

Multi source Schema level: Naming conflict between main dataset and continent dataset in Country column that we have in main dataset and nation in continent dataset and between Continent column and continent column naming conflict between Gender in names.csv and Sex in main csv structure conflict between Male and Female and 0 and 1

Multisource instant level: Contradictory Data that have the same Country name but different Continent Country (main csv) is not in nation (continent csv) the same country have two different name like "US" and "United States"



single instant level : Misspelling value for continent Misspelling value for Hemisphere BMI > 105 is misspell

Multi source Schema level: Naming conflict between main dataset and continent dataset in Country column that we have in main dataset and nation in continent dataset and between Continent column and continent column

```
for index, row in df.iterrows():

    Continent = row['Continent']
    Country = row['Country']
    Hemisphere = row['Hemisphere']
    if Country == "United States" :
        # Multi source Schema level: Naming conflict
        Country = "US"
    if not (Hemisphere == "Southern Hemisphere" or Hemisphere == "Northern Hemisphere"):
        # Misspelling value for Hemisphere
        print(f"Misspelling value for Hemisphere with index {index}")
    if not (Continent == "South America" or Continent == "North America" or Continent == "Europe" or Continent == "Asia" or Continent == "A
        # Misspelling value for Continent
        print(f"Misspelling value for Hemisphere with index {index}")
    result = continentcsv[continentcsv['nation'] == Country]['continent'].values
    if len(result) > 0:
        if result != Continent:
            if result[0] == 'Oceania' and Continent=='Australia':
                # Multi source Schema level: Naming conflict
                 continue

            # Contradictory Data that have the same Country name but different Continent
            print(f"The continent for {Country} is: {result[0]} but in main csv writed as: {Continent}")
    else:
        # Country (main csv) is not in nation (continent csv)
        print(f"The Country {Country} does not exist in the continentcsv.")
```

```
Misspelling value for Hemisphere with index 10
The continent for South Africa is: Africa but in main csv writed as: a
The Country B does not exist in the continentcsv.
```

```
for index, row in df.iterrows():
    Sex = row['Sex']
    patient = row['Patient ID']

    if Sex == "Male":
        Sex = "0"
    if Sex == "Female":
        Sex = "1"

    result = names[names['Patient ID'] == patient]['Gender'].values

    if len(result) > 0:
        if result[0] != Sex:
            print(f"The dataset names for sex is: {result[0]} but in the main csv written as: {Sex}")
    else:
        print(f"The patient ID {patient} does not exist in the names.csv")
```

```
The dataset names for sex is: 0 but in the main csv written as: 1
The dataset names for sex is: nan but in the main csv written as: nan
The dataset names for sex is: nan but in the main csv written as: nan
The dataset names for sex is: nan but in the main csv written as: nan
The patient ID OFE0541 does not exist in the names.csv
The patient ID YFE0882 does not exist in the names.csv
```

```
import pandas as pd
from tabulate import tabulate

# Load your dataset
# Replace '/path/to/your/dataset.csv' with the actual path to your dataset
dataset_path = '/content/heart_attack_prediction_dataset.csv'
df = pd.read_csv(dataset_path)

# List of columns to check for missing values, uniqueness, and validation
columns_to_check = [
     'Age', 'BMI'
]

# Create a list to store the information for each column
column_info = []

# Iterate over columns and check for missing values, uniqueness, and validation
for column in columns_to_check:
    missing_rows = df[df[column].isnull()]
    duplicate_rows = df[df.duplicated(subset=[column], keep=False)]

    # accuracy checks for specific columns
    accuarcy_check = None
    if column == 'Age':
        accuarcy_check = df['Age'].apply(lambda x: 0 <= x <150  if pd.notna(x) else None)
    elif column == 'BMI':
        accuarcy_check = df['BMI'].apply(lambda x: 9 <= x <105  if pd.notna(x) else None)

    info = {
        'Column': column,
        'Missing Rows': missing_rows.head(),
        'Duplicate Rows': duplicate_rows.head(),
        'accuracy Check': accuarcy_check
    }
    column_info.append(info)

# Display the information about missing values, uniqueness, and validation in a table
for info in column_info:
    print(f"\nColumn '{info['Column']}':")

    # Format missing values
    # missing_values_table = tabulate(info['Missing Rows'], headers='keys', tablefmt='pretty')
    # print(f"Missing values located at:\n{missing_values_table if info['Missing Rows'].shape[0] else 'No missing values.'}")

    # Format duplicate values
    # duplicate_values_table = tabulate(info['Duplicate Rows'], headers='keys', tablefmt='pretty')
    # print(f"Duplicated values located at:\n{duplicate_values_table if info['Duplicate Rows'].shape[0] else 'No duplicated values.'}")

    # Format unaccurate check
    if info['accuracy Check'] is not None:
        unaccurate_rows = df[~info['accuracy Check'].fillna(False)]
        if not unaccurate_rows.empty:
            unaccurate_rows_table = tabulate(unaccurate_rows, headers='keys', tablefmt='pretty')
            print(f"unaccurate rows based on accuracy check:\n{unaccurate_rows_table}")
        else:
            print("All rows are vaccurate.")
    else:
        print("No accuracy check performed for this column.")
```

```
Column 'Age':
unaccurate rows based on accuracy check:
+-----+------------+-------+--------+-------------+----------------+------------+----------+----------------+---------+---------+------
|     | Patient ID |  Age  |  Sex   | Cholesterol | Blood Pressure | Heart Rate | Diabetes | Family History | Smoking | Obesity | Alcoh
+-----+------------+-------+--------+-------------+----------------+------------+----------+----------------+---------+---------+------
|  1  |  CZE1114   | 230.0 |  Male  |    389.0    |     165/93     |    98.0    |    1     |       1        |    1    |    1    |
|  3  |  JLN3497   |  nan  |  Male  |    383.0    |    163/100     |    73.0    |    1     |       1        |    1    |    0    |
| 602 |  RON2853   |  nan  | Female |    224.0    |     136/71     |    97.0    |    0     |       1        |    1    |    0    |
+-----+------------+-------+--------+-------------+----------------+------------+----------+----------------+---------+---------+------

Column 'BMI':
All rows are vaccurate.
```

Double-click (or enter) to edit

```
import pandas as pd

# Load your dataset
# Replace '/path/to/your/dataset.csv' with the actual path to your dataset
dataset_path = '/content/heart_attack_prediction_dataset.csv'
df = pd.read_csv(dataset_path)

# Check the number of unique values in each column
unique_value_counts = df.nunique()

# Filter columns with non-unique values
non_unique_columns = unique_value_counts[unique_value_counts < len(df)].index

# Display the columns with non-unique values
print("Columns with Non-Unique Values:")
print(non_unique_columns)
```

```
Columns with Non-Unique Values:
Index(['Patient ID', 'Age', 'Sex', 'Cholesterol', 'Blood Pressure',
       'Heart Rate', 'Diabetes', 'Family History', 'Smoking', 'Obesity',
       'Alcohol Consumption', 'Diet', 'Previous Heart Problems',
       'Medication Use', 'Stress Level', 'Income', 'Triglycerides',
       'Physical Activity Days Per Week', 'Sleep Hours Per Day', 'Country',
       'Continent', 'Hemisphere', 'Heart Attack Risk'],
      dtype='object')
```

.برای بهبود کیفیت داده ستون قاره چون از کشور قابل استخراج است اضافه است. و همچنین ستون نیم کره اضافه است پس حذف میکنیم

به جای اینکه به صورت باینری خروجی را چاپ کنیم به صورت درصدی عدد را ثبت کنیم با این کار دقت مدل بالا میرود heart rist attackرای

اگر تعدا سطر های دارای مقادیر خالی برای یک ستون خاص کم بود باید با مقادیر نزدیک به اون جایگزتری کنیم

مقادیر تکرار شده رو یک بار استفاده کنیم

داده ها ختما یک مدل باشند

دیتا معتبر و درست باشند سطر های دارای مقادیر اشتباه را حذف میکنیم

replacing the missing values(null) with mean or median

```
import pandas as pd
from sklearn.impute import SimpleImputer

# Load your dataset
# Replace '/path/to/your/dataset.csv' with the actual path to your dataset
dataset_path = '/content/heart_attack_prediction_dataset.csv'
df = pd.read_csv(dataset_path)

# Set the threshold for dropping columns with more than X null values
null_threshold = 100

# Identify columns with more than the threshold null values
columns_to_drop = df.columns[df.isnull().sum() > null_threshold]

# Drop columns with more than the threshold null values
df.drop(columns=columns_to_drop, inplace=True)

# Create a DataFrame to store information about the replaced values
replaced_values_info = pd.DataFrame(columns=['Column', 'Replaced Nulls', 'Replacement Value', 'Reason'])

# Iterate over columns and fill missing values with mean, mode, or median
for column in df.columns:
    if df[column].isnull().any():
        # Determine fill value based on data type
        fill_value = df[column].mean() if pd.api.types.is_numeric_dtype(df[column].dtype) else df[column].mode().iloc[0]

        # Fill missing values
        replaced_rows = df[df[column].isnull()]
        df[column].fillna(fill_value, inplace=True)

        # Store information about replaced values
        replaced_values_info = replaced_values_info.append({
            'Column': column,
            'Replaced Nulls': len(replaced_rows),
            'Replacement Value': fill_value,
            'Reason': 'Mean' if pd.api.types.is_numeric_dtype(df[column].dtype) else 'Mode'
        }, ignore_index=True)

# Display the updated dataset
print(df)

# Display information about replaced values
print("\nInformation about replaced values:")
print(replaced_values_info)
```

```
     Patient ID         Age     Sex  Cholesterol Blood Pressure  Heart Rate  \
0        kjreer   67.000000    Male        208.0         158/88        72.0
1       CZE1114  230.000000    Male        389.0         165/93        98.0
2       BNI9906   21.000000       x        324.0         174/99        72.0
3       JLN3497   53.724803    Male        383.0        163/100        73.0
4       ZOO7941   66.000000    Male        318.0          91/88        93.0
...         ...         ...     ...          ...            ...         ...
8758    MSV9918   60.000000    Male        121.0          94/76        61.0
8759    QSV6764   28.000000  Female        120.0        157/102        73.0
8760    XKA5925   47.000000    Male        250.0         161/75       105.0
8761    EPE6801   36.000000    Male        178.0         119/67        60.0
8762    ZWN9666   25.000000  Female        356.0         138/67        75.0

      Diabetes  Family History  Smoking  Obesity  ...  \
0            0               0        1        0  ...
1            1               1        1        1  ...
2            1               0        0        0  ...
3            1               1        1        0  ...
4            1               1        1        1  ...
...        ...             ...      ...      ...  ...
8758         1               1        1        0  ...
8759         1               0        0        1  ...
8760         0               1        1        1  ...
8761         1               0        1        0  ...
8762         1               1        0        0  ...

      Sedentary Hours Per Day  Income        BMI  Triglycerides  \
0                    6.615001  261404  31.251233            286
1                    4.963459  285768  27.194973            235
2                    9.463426  235282  28.176571            587
3                    7.648981  125640  36.464704            378
4                    1.514821  160555  21.809144            231
...                       ...     ...        ...            ...
8758                10.806373  235420  19.655895             67
```

```
8759                3.833038  217881  23.993866           617
8760                2.375214   36998  35.406146           527
8761                0.029104  209943  27.294020           114
8762                9.005234  247338  32.914151           180

      Physical Activity Days Per Week  Sleep Hours Per Day         Country  \
0                                   0                    6       Argentina
1                                   1                    7          Canada
2                                   4                    4          France
3                                   3                    4          Canada
4                                   1                    5        Thailand
...                               ...                  ...             ...
8758                                7                    7        Thailand
8759                                4                    9          Canada
8760                                4                    4          Brazil
8761                                2                    8          Brazil
8762                                7                    4  United Kingdom

          Continent           Hemisphere  Heart Attack Risk
0     South America  Southern Hemisphere                  0
1     North America  Northern Hemisphere                  0
2            Europe  Northern Hemisphere                  0
3     North America  Northern Hemisphere                  0
```

```
import pandas as pd
from sklearn.impute import SimpleImputer

# Load your dataset
# Replace '/path/to/your/dataset.csv' with the actual path to your dataset
dataset_path = '/content/heart_attack_prediction_dataset.csv'
df = pd.read_csv(dataset_path)

# Set the threshold for dropping columns with more than X percent null values
null_threshold_percent = 10

# Calculate the threshold in terms of the percentage of missing values
null_threshold = len(df) * (null_threshold_percent / 100)

# Identify columns with more than the threshold null values
columns_to_drop = df.columns[df.isnull().sum() > null_threshold]

# Drop columns with more than the threshold null values
df.drop(columns=columns_to_drop, inplace=True)

# Create a DataFrame to store information about the replaced values
replaced_values_info = pd.DataFrame(columns=['Column', 'Replaced Nulls', 'Replacement Value', 'Reason'])

# Iterate over columns and fill missing values with mean, mode, or median
for column in df.columns:
    if df[column].isnull().any():
        # Determine fill value based on data type
        fill_value = df[column].mean() if pd.api.types.is_numeric_dtype(df[column].dtype) else df[column].mode().iloc[0]

        # Fill missing values
        replaced_rows = df[df[column].isnull()]
        df[column].fillna(fill_value, inplace=True)

        # Store information about replaced values
        replaced_values_info = pd.concat([
            replaced_values_info,
            pd.DataFrame({
                'Column': [column],
                'Replaced Nulls': [len(replaced_rows)],
                'Replacement Value': [fill_value],
                'Reason': ['Mean' if pd.api.types.is_numeric_dtype(df[column].dtype) else 'Mode']
            })
        ], ignore_index=True)

# Display the updated dataset
print(df)

# Display information about replaced values
print("\nInformation about replaced values:")
print(replaced_values_info)
```

```
     Patient ID        Age     Sex  Cholesterol Blood Pressure  Heart Rate  \
0        kjreer  67.000000    Male        208.0         158/88        72.0
```