lab-1

October 26, 2024

0.0.1 1.1 Variable Classification

Based on the dataset, we classify each variable into one of the following categories: 1. **Nominal Categorical**: Categories with no specific order. 2. **Binary Categorical**: Two possible categories, such as yes/no or 0/1. 3. **Discrete**: Whole numbers, like a count of items. 4. **Continuous**: Values that can take any value within a range, such as height or temperature.

Let's load the data and assign each column to a category.

[28]:
```python
# Importing necessary libraries
import pandas as pd

# Load the dataset
df = pd.read_csv('heart.csv')

# Categorize each variable
variable_classification = {
    'sbp': 'Continuous',
    'tobacco': 'Continuous',
    'ldl': 'Continuous',
    'adiposity': 'Continuous',
    'famhist': 'Binary Categorical',
    'typea': 'Continuous',
    'obesity': 'Continuous',
    'alcohol': 'Continuous',
    'age': 'Discrete',
    'chd': 'Binary Categorical'
}

# Display the classification
for column, category in variable_classification.items():
    print(f"{column}: {category}")
```

```
sbp: Continuous
tobacco: Continuous
ldl: Continuous
adiposity: Continuous
famhist: Binary Categorical
typea: Continuous
obesity: Continuous
```

```
alcohol: Continuous
age: Discrete
chd: Binary Categorical
```

0.0.2 1.2 Find the Number of Null Values for Each Column

In this section, we will check for any missing values in each column of the dataset and display the count of null values.

[31]:
```python
# Check for null values in each column
null_counts = df.isnull().sum()

# Display the number of null values for each column
print("Number of null values for each column:")
print(null_counts)
```

```
Number of null values for each column:
sbp          28
tobacco      40
ldl          39
adiposity    40
famhist      45
typea        41
obesity      40
alcohol      40
age          35
chd          39
dtype: int64
```

0.0.3 1.31 General Descriptive Statistics

Let's display the general descriptive statistics of the dataset to understand the basic distribution of each variable.

[34]:
```python
# Display general descriptive statistics
print("General Descriptive Statistics:")
df.describe()
```

```
General Descriptive Statistics:
```

[34]:

	sbp	tobacco	ldl	adiposity	typea	obesity	\
count	384.000000	372.000000	373.000000	372.000000	371.000000	372.000000	
mean	139.216146	3.676425	4.569303	25.210753	52.008086	25.763602	
std	20.307368	4.568564	1.888691	7.760257	9.822888	3.854265	
min	101.000000	0.000000	0.980000	7.120000	20.000000	17.890000	
25%	124.000000	0.057500	3.240000	19.307500	46.000000	22.835000	
50%	136.000000	1.800000	4.220000	26.115000	52.000000	25.675000	
75%	148.500000	5.640000	5.470000	30.790000	58.000000	28.167500	
max	218.000000	27.400000	14.160000	42.490000	73.000000	40.340000	

```
       alcohol         age         chd
count  372.000000  377.000000  373.000000
mean    18.425134   42.453581    0.335121
std     25.971090   15.312649    0.472667
min      0.000000   15.000000    0.000000
25%      0.195000   30.000000    0.000000
50%      7.300000   45.000000    0.000000
75%     25.820000   57.000000    1.000000
max    145.290000   64.000000    1.000000
```

0.0.4 1.32 Oldest Person

Identify the age of the oldest person in the dataset and display all individuals who are of that age.

[37]:
```python
# Find the maximum age and filter the dataset
oldest_age = df['age'].max()
oldest_people = df[df['age'] == oldest_age]

print(f"The age of the oldest person is: {oldest_age}")
print("People with the oldest age:")
oldest_people
```

```
The age of the oldest person is: 64.0
People with the oldest age:
```

[37]:
```
       sbp  tobacco   ldl  adiposity  famhist  typea  obesity  alcohol   age  \
58   158.0     3.60  2.97        NaN   Absent    NaN    26.64   108.00  64.0
70   152.0    12.18  4.04      37.83  Present   63.0    34.57     4.17  64.0
110  126.0     0.00  5.98      29.06  Present   56.0    25.39    11.52  64.0
167  148.0     8.20  7.75      34.46  Present   46.0    26.53     6.04  64.0
170  128.0     5.16  4.90        NaN  Present   57.0    26.42     0.00  64.0
206    NaN     8.60  3.90      32.16  Present   52.0    28.51    11.11  64.0
241  160.0     0.60  6.94      30.53   Absent   36.0    25.68     1.42  64.0
256  138.0     2.00  5.11      31.40  Present   49.0    27.25     2.06  64.0
276  128.0     0.73  3.97      23.52   Absent    NaN    23.81      NaN  64.0
348  140.0     8.60  3.90      32.16  Present   52.0    28.51    11.11  64.0
374  160.0     0.60  6.94      30.53   Absent   36.0    25.68      NaN  64.0
402  174.0     2.02  6.57      31.90  Present   50.0    28.75    11.83  64.0

     chd
58   0.0
70   0.0
110  1.0
167  1.0
170  0.0
206  1.0
241  0.0
```

```
256    1.0
276    0.0
348    1.0
374    0.0
402    1.0
```

0.0.5 1.33 Youngest Person

Identify the age of the youngest person in the dataset and display all individuals who are of that age.

[40]:
```
# Find the minimum age and filter the dataset
youngest_age = df['age'].min()
youngest_people = df[df['age'] == youngest_age]

print(f"The age of the youngest person is: {youngest_age}")
print("People with the youngest age:")
youngest_people
```

```
The age of the youngest person is: 15.0
People with the youngest age:
```

[40]:
```
      sbp  tobacco   ldl  adiposity famhist  typea  obesity  alcohol   age  \
9   132.0      0.0  1.87      17.21  Absent   49.0    23.63     0.97  15.0
38    NaN      0.0  3.67      12.13  Absent    NaN    19.15     0.60  15.0

    chd
9   0.0
38  0.0
```

0.0.6 1.34 Average and Standard Deviation of Age

Calculate the average (mean) and standard deviation of the age column to understand its central tendency and spread.

[43]:
```
# Calculate mean and standard deviation of the age column
age_mean = df['age'].mean()
age_std = df['age'].std()

print(f"Average age: {age_mean}")
print(f"Standard deviation of age: {age_std}")
```

```
Average age: 42.45358090185676
Standard deviation of age: 15.31264927550187
```

0.0.7 1.35 Median Age

Calculate the median of the age column.

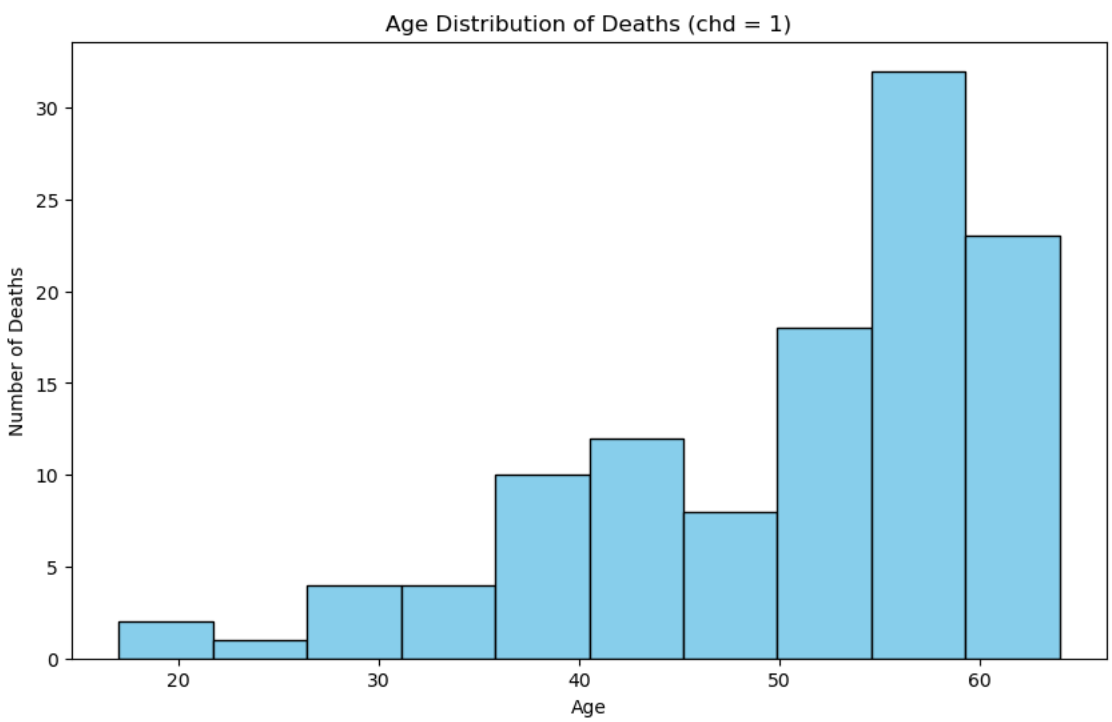
[46]:
```
# Calculate the median age
age_median = df['age'].median()

print(f"Median age: {age_median}")
```

```
Median age: 45.0
```

0.0.8 1.36 Bar Chart of Deaths vs. Age

Filter the data for individuals who have died (chd = 1) and create a histogram to observe the age distribution of deaths.

[49]:
```
import matplotlib.pyplot as plt

# Filter data for deaths (chd = 1)
deaths_data = df[df['chd'] == 1]

# Plot histogram
plt.figure(figsize=(10, 6))
plt.hist(deaths_data['age'], bins=10, color='skyblue', edgecolor='black')
plt.xlabel("Age")
plt.ylabel("Number of Deaths")
plt.title("Age Distribution of Deaths (chd = 1)")
plt.show()

# Insight
# From this chart, we can observe that ...
```

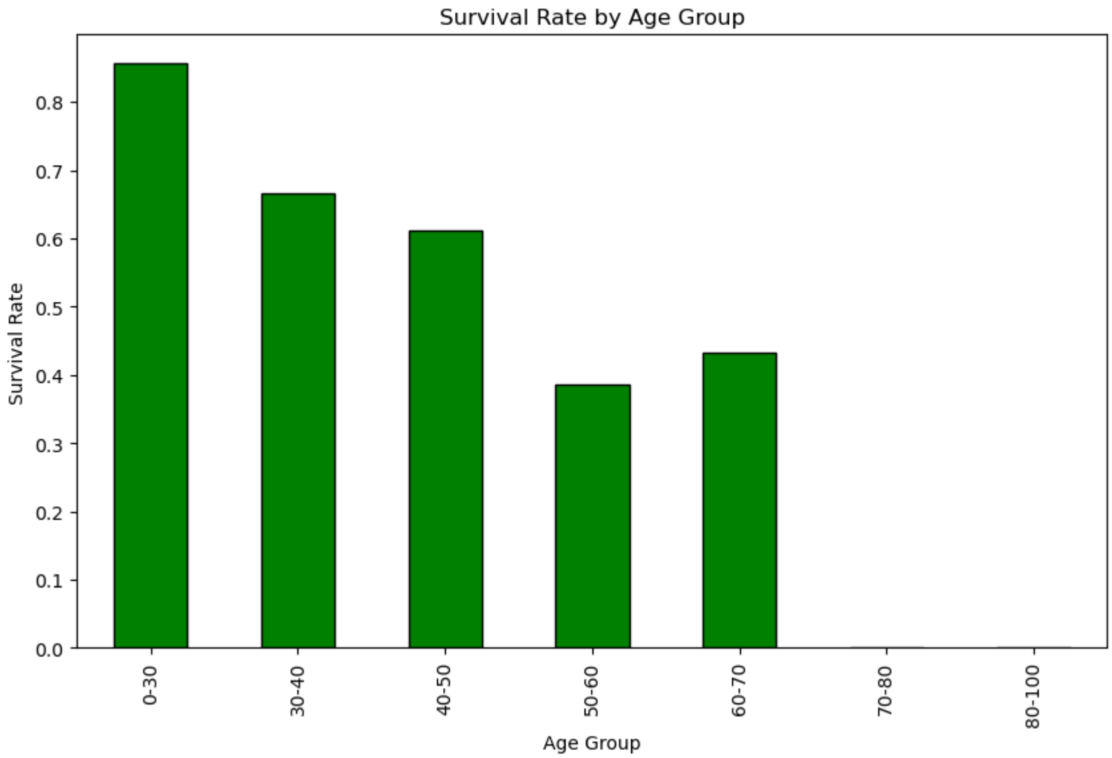


0.0.9 1.37 Age Groups with Largest Survival Rate

Divide the data into age groups and calculate the survival rate for each group, then visualize it.

[52]:
```
# Define age groups and calculate survival rates
df['age_group'] = pd.cut(df['age'], bins=[0, 30, 40, 50, 60, 70, 80, 100],␣
↪labels=['0-30', '30-40', '40-50', '50-60', '60-70', '70-80', '80-100'])
survival_rate = df[df['chd'] == 0].groupby('age_group').size() / df.
↪groupby('age_group').size()

# Plot survival rate
plt.figure(figsize=(10, 6))
survival_rate.plot(kind='bar', color='green', edgecolor='black')
plt.xlabel("Age Group")
plt.ylabel("Survival Rate")
plt.title("Survival Rate by Age Group")
plt.show()
```

```
C:\Users\Niranja Rao\AppData\Local\Temp\ipykernel_13800\1887164646.py:3:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  survival_rate = df[df['chd'] == 0].groupby('age_group').size() /
df.groupby('age_group').size()
C:\Users\Niranja Rao\AppData\Local\Temp\ipykernel_13800\1887164646.py:3:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  survival_rate = df[df['chd'] == 0].groupby('age_group').size() /
df.groupby('age_group').size()
```



0.0.10 1.38 Relationship between Family History (famhist) and Coronary Heart Disease (chd)

Analyze the relationship between `famhist` and `chd`.

[55]:
```python
# Calculate the relationship between famhist and chd
famhist_chd_relationship = df.groupby('famhist')['chd'].mean()

print("Relationship between Family History and CHD:")
print(famhist_chd_relationship)
```

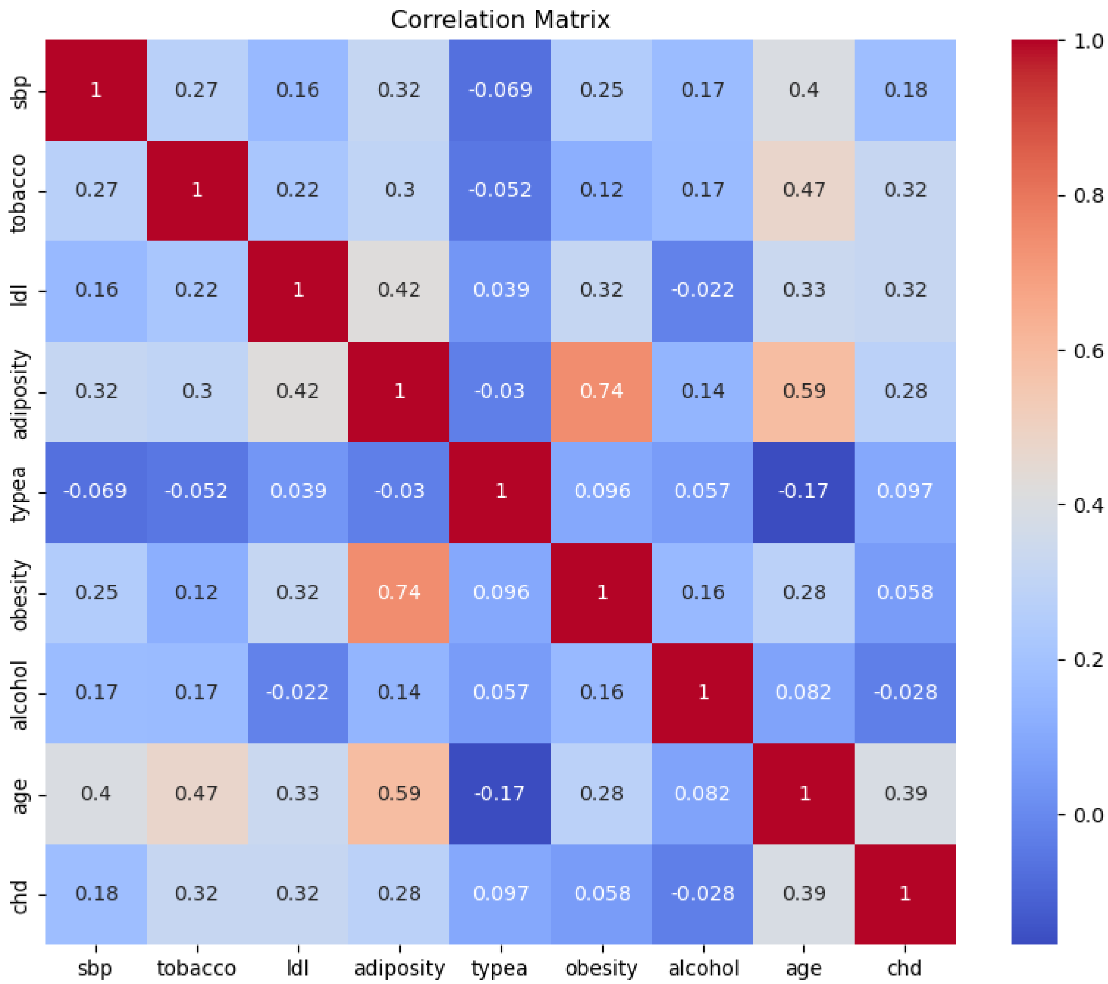
```
Relationship between Family History and CHD:
famhist
Absent     0.243094
Present    0.443709
Name: chd, dtype: float64
```

0.0.11 1.39 Visualizations for Data Distributions

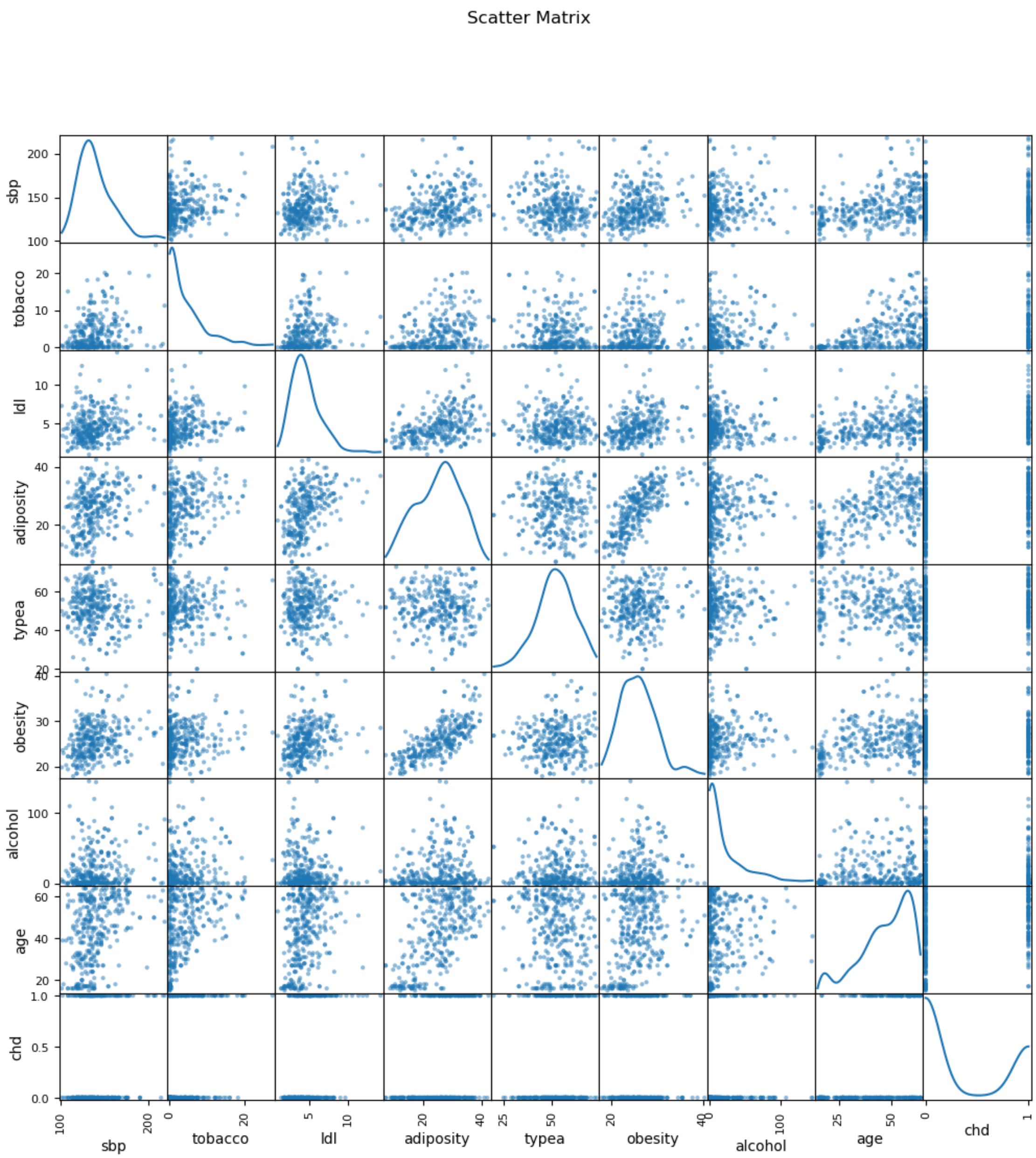
Let's create various visualizations to better understand data distributions.

[70]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
```

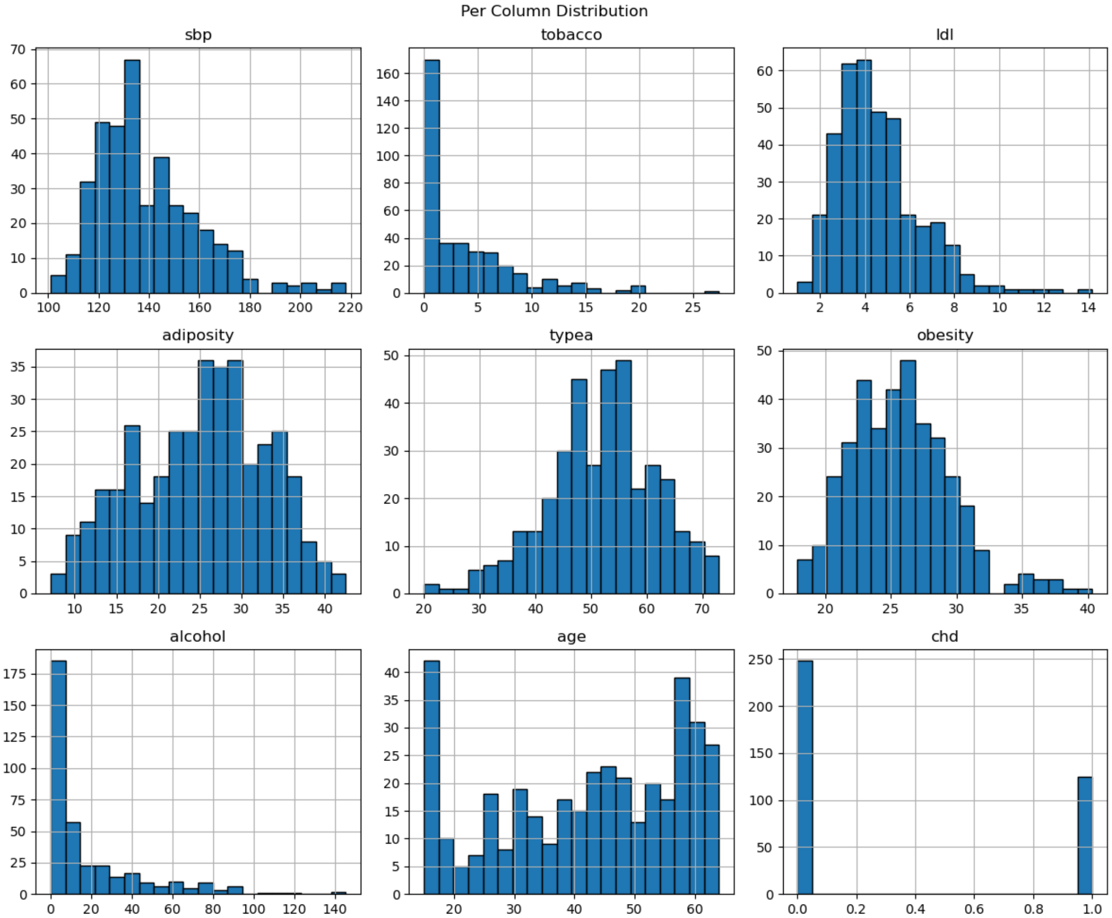
```
# i. Correlation Matrix
plt.figure(figsize=(10, 8))
numeric_df = df.select_dtypes(include=['float64', 'int64'])  # Select only␣
↪numeric columns
sns.heatmap(numeric_df.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Matrix")
plt.show()

# ii. Scatter Matrix
plt.figure(figsize=(12, 12))
scatter_matrix(numeric_df, figsize=(12, 12), diagonal='kde')
plt.suptitle("Scatter Matrix")
plt.show()

# iii. Per Column Distribution
plt.figure(figsize=(12, 10))
numeric_df.hist(bins=20, edgecolor='black', figsize=(12, 10))
plt.suptitle("Per Column Distribution")
plt.tight_layout()
plt.show()

# iiii. Heatmap for Missing Values
plt.figure(figsize=(12, 6))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title("Heatmap of Missing Values")
plt.show()
```

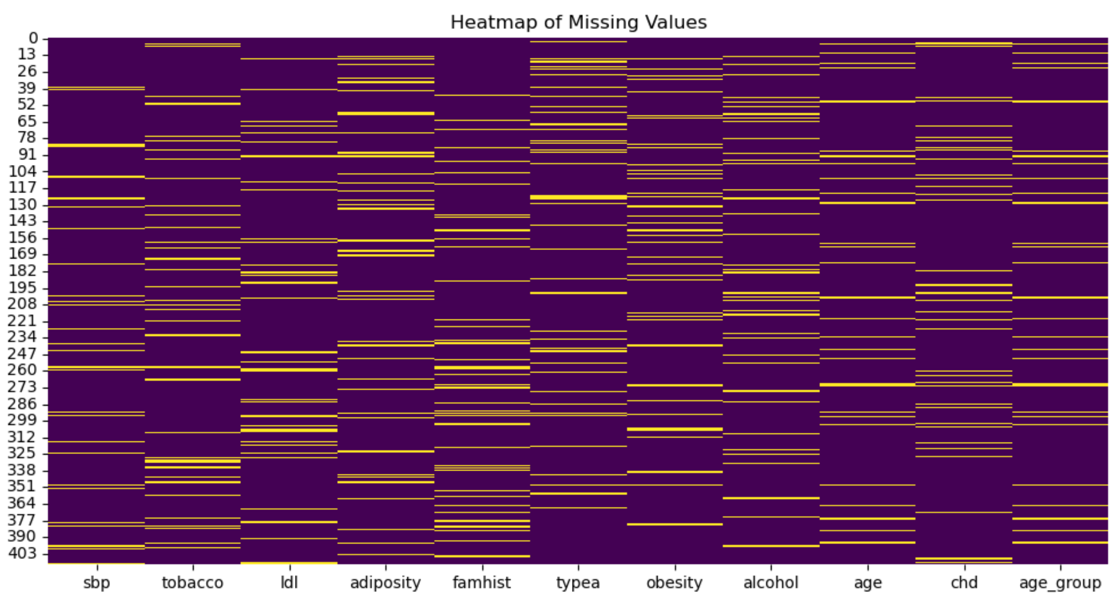



```
<Figure size 1200x1200 with 0 Axes>
```



<Figure size 1200x1000 with 0 Axes>


Per Column Distribution
sbp
tobacco
ldl
adiposity
typea
obesity
alcohol
age
chd


Heatmap of Missing Values
sbp
tobacco
ldl
adiposity
famhist
typea
obesity
alcohol
age
chd
age_group

0.0.12 1.3.10 Handling Null Values

Aside from simply dropping rows with missing data, there are several techniques for handling null values:

1. **Imputation**:
   - Replace missing values with the mean, median, or mode of the column.
   - Useful for continuous variables (e.g., replacing with mean or median) or categorical variables (e.g., replacing with mode).
2. **Interpolation**:
   - For time series data, interpolate missing values based on neighboring values, maintaining data continuity.
3. **Using Machine Learning Models**:
   - Train a predictive model to estimate missing values based on other features in the dataset.
   - Examples include k-Nearest Neighbors (k-NN) and linear regression.
4. **Using Domain-Specific Values**:
   - For some fields, domain knowledge can inform a reasonable substitute for missing values.
5. **Multiple Imputation**:
   - Create multiple imputations for missing values and use the resulting datasets for robust statistical analysis.

Choosing the best approach depends on the type of data, its distribution, and the analysis goals.

0.0.13 2.1 Basic Matrix Multiplication

Define a function that multiplies two matrices, `A` and `B`, without using any built-in matrix multiplication functions.

[85]:
```
import numpy as np

def matrix_multiply(A, B):
    """
    Multiplies two matrices A and B without using built-in matrix multiplication␣
 ↪functions.

    Parameters:
    A (numpy.ndarray): First matrix.
    B (numpy.ndarray): Second matrix.

    Returns:
    numpy.ndarray: The product of matrices A and B.
    """
    # Check if the matrices can be multiplied
    if A.shape[1] != B.shape[0]:
        raise ValueError("Number of columns in A must equal the number of rows␣
 ↪in B.")
```

```
    # Initialize the result matrix with zeros
    result = np.zeros((A.shape[0], B.shape[1]))

    # Perform matrix multiplication
    for i in range(A.shape[0]):
        for j in range(B.shape[1]):
            for k in range(A.shape[1]):
                result[i][j] += A[i][k] * B[k][j]

    return result

# Example usage
A = np.array([[12, 21], [2, 8]])
B = np.array([[13, 7], [7, 8]])
print("Matrix A:")
print(A)
print("Matrix B:")
print(B)
print("Product of A and B:")
print(matrix_multiply(A, B))
```

```
Matrix A:
[[12 21]
 [ 2  8]]
Matrix B:
[[13  7]
 [ 7  8]]
Product of A and B:
[[303. 252.]
 [ 82.  78.]]
```

0.0.14 2.2 Compute the Determinant

Define a function to compute the determinant of a square matrix using the numpy.linalg module.

[88]:
```
import numpy as np

def compute_determinant(A):
    """
    Computes the determinant of a square matrix A.

    Parameters:
    A (numpy.ndarray): Square matrix.

    Returns:
    float: Determinant of the matrix A.
    """
    return np.linalg.det(A)
```

```
# Example usage
A = np.array([[11, 13], [15, 17]])
print("Matrix A:")
print(A)
print("Determinant of A:")
print(compute_determinant(A))
```

```
Matrix A:
[[11 13]
 [15 17]]
Determinant of A:
-8.000000000000012
```

0.0.15 2.3 Solve a System of Linear Equations

Define a function to solve the system of linear equations (Ax = b) using numpy.linalg.solve().

[68]:
```
import numpy as np

def solve_linear_system(A, b):
    """
    Solves the system of linear equations Ax = b.

    Parameters:
    A (numpy.ndarray): Coefficient matrix.
    b (numpy.ndarray): Constant vector.

    Returns:
    numpy.ndarray: Solution vector x.
    """
    return np.linalg.solve(A, b)

# Example usage
A = np.array([[3, 1], [1, 2]])
b = np.array([9, 8])
print("Coefficient matrix A:")
print(A)
print("Constant vector b:")
print(b)
print("Solution vector x:")
print(solve_linear_system(A, b))
```

```
Coefficient matrix A:
[[3 1]
 [1 2]]
Constant vector b:
[9 8]
Solution vector x:
```

```
[2. 3.]
```