Task 1. Data Summarization:

Calculate basic summary statistics (mean, median, standard deviation, etc.) for each numerical variable.

```
from google.colab import files

uploaded = files.upload()
```

Choose Files healthcare-…ke-data.csv

- **healthcare-dataset-stroke-data.csv**(text/csv) - 316971 bytes, last modified: 1/26/2021 - 100% done

```
Saving healthcare-dataset-stroke-data.csv to healthcare-dataset-stroke-data.csv
```

```
import pandas as pd

dataset = pd.read_csv('healthcare-dataset-stroke-data.csv')
dataset.head()
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_st
0	9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly sm
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never sm
2	31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never sm
3	60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	sm
4	1665	Female	79.0	1	0	Yes	Self-	Rural	174.12	24.0	never sm

```
numerical_variable = [feature for feature in dataset.columns if dataset[feature].dtypes != 'O']
print('Number of numerical variables: ', len(numerical_variable))
dataset[numerical_variable].head()
```

```
Number of numerical variables:  7
```

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
0	9046	67.0	0	1	228.69	36.6	1
1	51676	61.0	0	0	202.21	NaN	1
2	31112	80.0	0	1	105.92	32.5	1
3	60182	49.0	0	0	171.23	34.4	1
4	1665	79.0	1	0	174.12	24.0	1

```
means = dataset.mean()
means
```

```
<ipython-input-4-2e51b005e08c>:1: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future version, it will default to False. In addition, specifying 'num
  means = dataset.mean()
```

```
id                  36517.829354
age                    43.226614
hypertension            0.097456
heart_disease           0.054012
avg_glucose_level     106.147677
bmi                    28.893237
stroke                  0.048728
dtype: float64
```

Conclusion:

Based on the calculated means of features from the dataset:

ID: The average ID value is approximately 36,517.83, suggesting that the dataset may have a wide range of unique identifiers.

Age: The average age is approximately 43.23 years, indicating that the dataset contains individuals with a broad age range.

Hypertension: The mean hypertension rate is around 9.75%, implying that a relatively small proportion of individuals in the dataset have hypertension.

Heart Disease: The mean heart disease rate is approximately 5.40%, indicating that a small percentage of individuals in the dataset have a history of heart disease.

Average Glucose Level: The average glucose level is about 106.15 mg/dL, providing insights into the typical blood glucose levels in the dataset.

BMI (Body Mass Index): The mean BMI is roughly 28.89, representing the average body mass index of individuals in the dataset.

Stroke: The mean stroke rate is approximately 4.87%, indicating that a small proportion of individuals in the dataset have experienced a stroke.

These mean values provide a summary of the central tendencies of the dataset's features.

```
medians = dataset.median()
medians
```

```
<ipython-input-5-fbf1ff32e499>:1: FutureWarning: The default value of numeric_only in DataFrame.median is deprecated. In a future version, it will default to False. In addition, specifying 'n
  medians = dataset.median()
id                  36932.000
age                    45.000
hypertension            0.000
heart_disease           0.000
avg_glucose_level      91.885
bmi                    28.100
stroke                  0.000
dtype: float64
```

For id, it represents the median identification number, which doesn't provide meaningful statistical information.

age has a median of 45.000, indicating that approximately half of the individuals in the dataset are below 45 years of age, and the other half are above 45.

hypertension, heart_disease, and stroke are binary variables, and their medians of 0.000 suggest that the majority of individuals do not have hypertension, heart disease, or stroke based on this dataset.

avg_glucose_level has a median of 91.885, which represents the middle value of the glucose levels in the dataset. This can be used to understand the typical glucose level of the population.

bmi has a median of 28.100, indicating that the middle value of BMI (Body Mass Index) in the dataset is 28.100, which can help assess the average weight status of individuals in the dataset.

```
standard_deviation = dataset.std()
standard_deviation
```

```
<ipython-input-6-45780231bd12>:1: FutureWarning: The default value of numeric_only in DataFrame.std is deprecated. In a future version, it will default to False. In addition, specifying 'nume
  standard_deviation = dataset.std()
id                   21161.721625
age                     22.612647
hypertension             0.296607
heart_disease            0.226063
avg_glucose_level       45.283560
bmi                      7.854067
stroke                   0.215320
dtype: float64
```

age: The standard deviation of age is relatively low (22.61), indicating that the ages in the dataset are relatively close to the mean, with limited variation.

hypertension and heart_disease: Both hypertension and heart disease have low standard deviations (0.30 and 0.23, respectively), suggesting that these binary variables have little variability within the dataset.

avg_glucose_level: The standard deviation for average glucose level is relatively high (45.28), indicating a wider range of values, suggesting more significant variability.

bmi: The standard deviation for BMI (body mass index) is moderately high (7.85), indicating a notable spread in BMI values across the dataset.

stroke: The standard deviation for the target variable, 'stroke,' is relatively low (0.22), indicating that it is imbalanced, with a majority of non-stroke cases.

```
dataset.describe()
```

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	5110.000000
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.854067	0.215320
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.500000	0.000000
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.100000	0.000000
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.100000	0.000000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

Count the frequency of unique values for categorical variables

```
categorical_variable = [feature for feature in dataset.columns if dataset[feature].dtypes == 'O']
print('Number of Categorical variables: ', len(categorical_variable))
dataset[categorical_variable].head()
```

```
Number of Categorical variables:  5
```

	gender	ever_married	work_type	Residence_type	smoking_status
0	Male	Yes	Private	Urban	formerly smoked
1	Female	Yes	Self-employed	Rural	never smoked
2	Male	Yes	Private	Rural	never smoked
3	Female	Yes	Private	Urban	smokes
4	Female	Yes	Self-employed	Rural	never smoked

```
for feature in categorical_variable:
    unique_value = dataset[feature].value_counts()
    print(f"Unique Values for {feature}:\n{unique_value}\n")
```

```
Unique Values for gender:
Female    2994
Male      2115
Other        1
Name: gender, dtype: int64

Unique Values for ever_married:
Yes    3353
No     1757
Name: ever_married, dtype: int64

Unique Values for work_type:
Private          2925
Self-employed     819
children          687
Govt_job          657
Never_worked       22
Name: work_type, dtype: int64

Unique Values for Residence_type:
Urban    2596
Rural    2514
Name: Residence_type, dtype: int64

Unique Values for smoking_status:
never smoked       1892
Unknown            1544
formerly smoked     885
smokes              789
Name: smoking_status, dtype: int64
```

Calculate the number of missing values for each variable

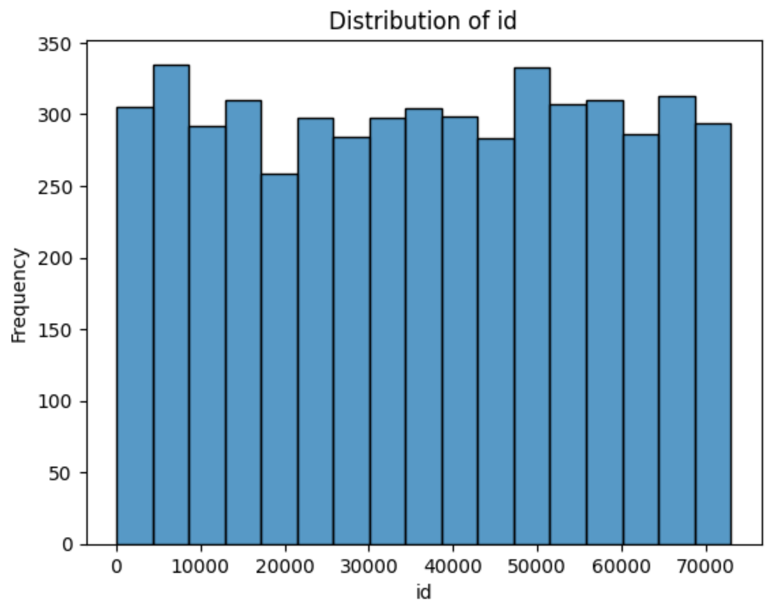
```
missing_values = dataset.isna().sum()
print(missing_values)
```

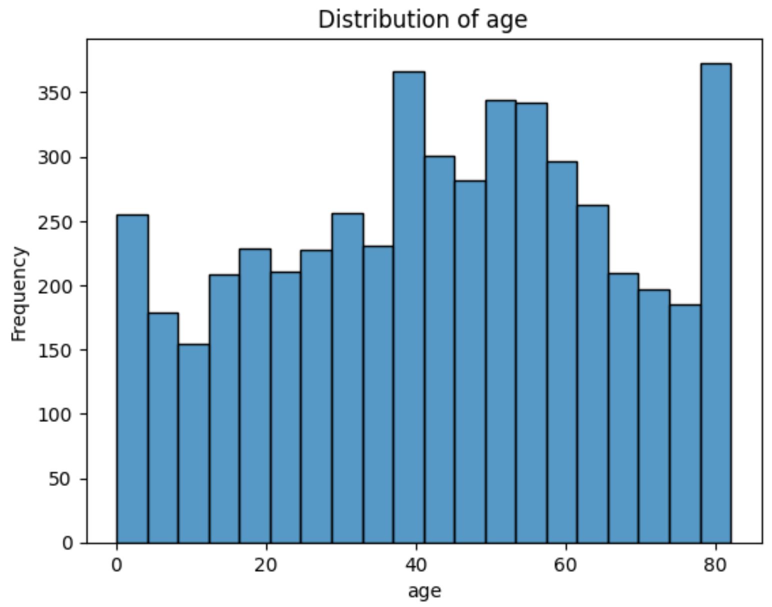
```
id                     0
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

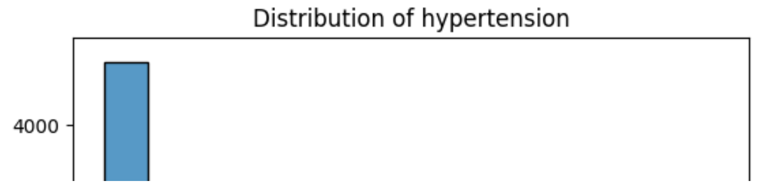
Task 2. Data Visualization:

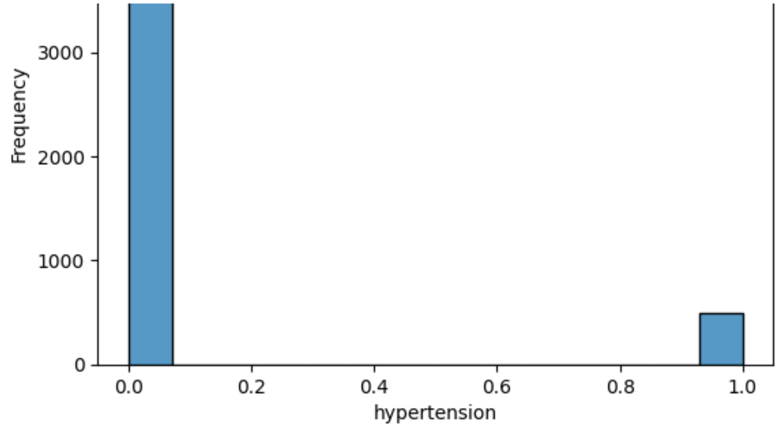
Create histograms or density plots to visualize the distribution of numerical variables

```
import seaborn as sns
import matplotlib.pyplot as plt

for Feature in numerical_variable:
    sns.histplot(dataset[Feature])
    plt.title(f'Distribution of {Feature}')
    plt.xlabel(Feature)
    plt.ylabel('Frequency')
    plt.show()
```

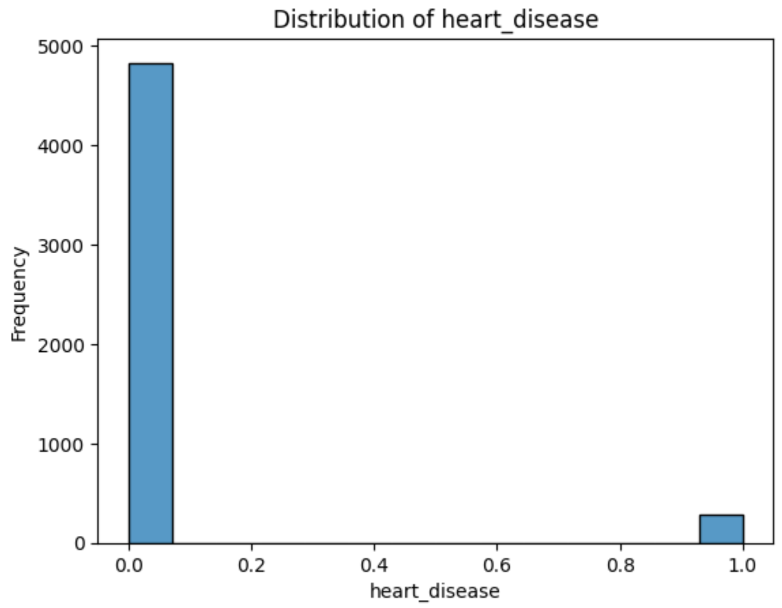

Distribution of id
Frequency
id
0
50
100
150
200
250
300
350
0
10000
20000
30000
40000
50000
60000
70000


Distribution of age
Frequency
age
0
50
100
150
200
250
300
350
0
20
40
60
80

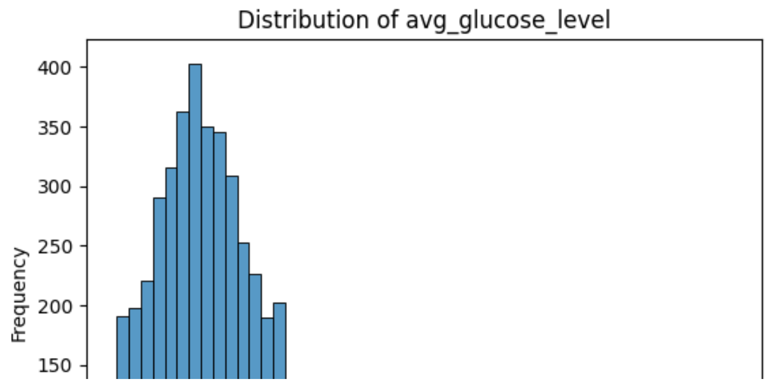

Distribution of hypertension
4000



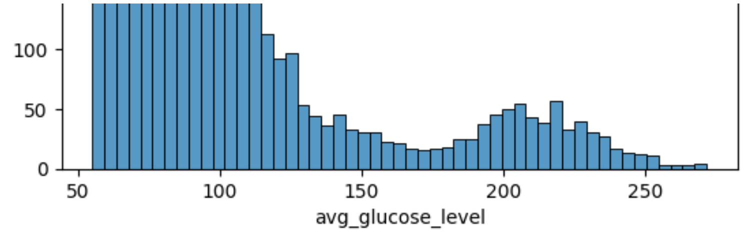
Frequency
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
hypertension

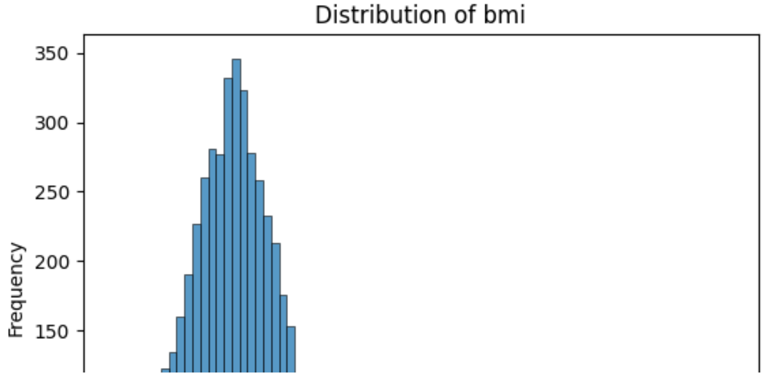


Distribution of heart_disease
Frequency
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
heart_disease



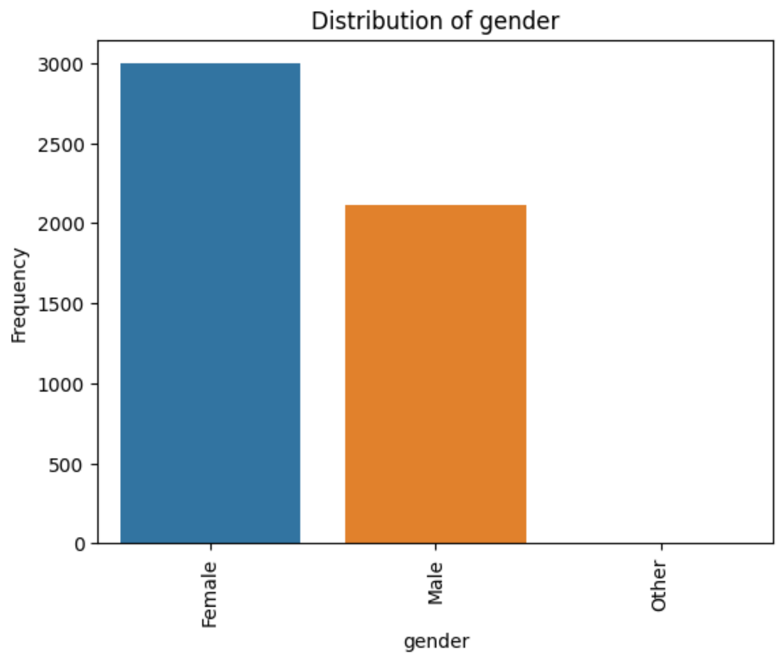
Distribution of avg_glucose_level
Frequency
400
350
300
250
200
150

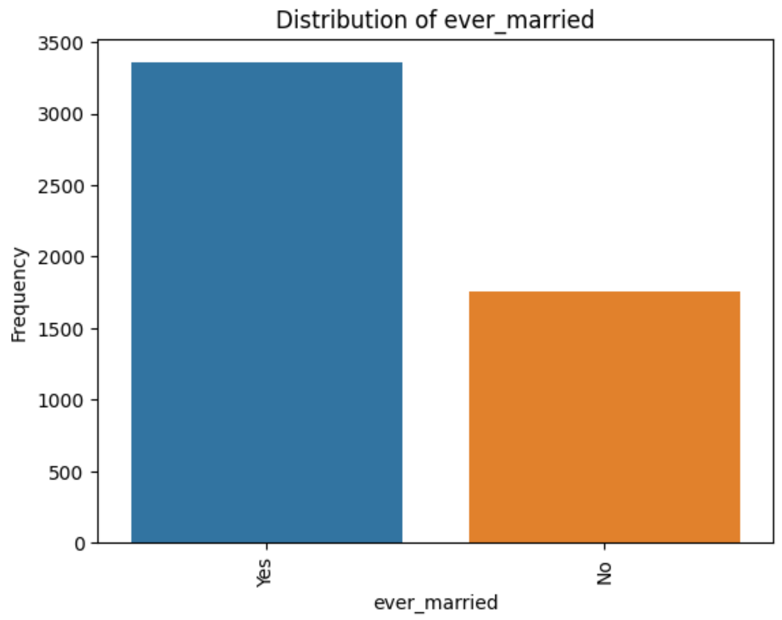


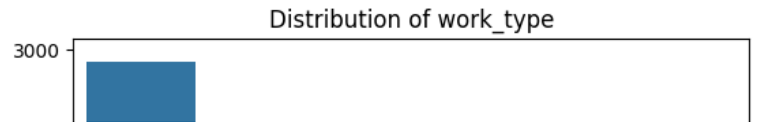


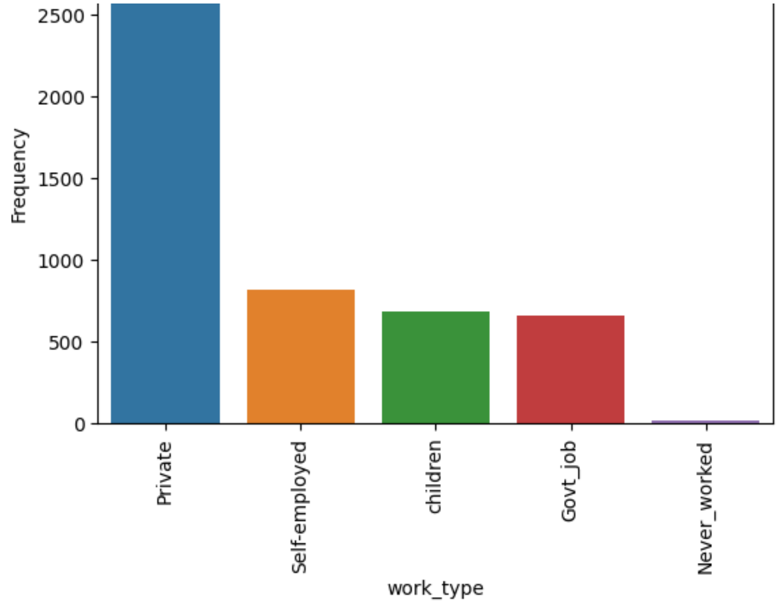
Generate bar plots or pie charts to visualize the distribution of categorical variables

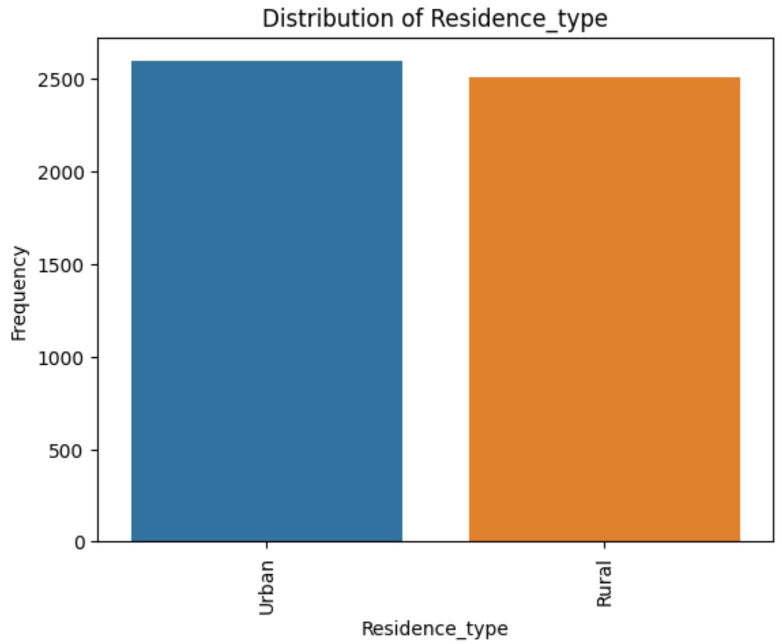
```
for Feature in categorical_variable:
    unique_values = dataset[Feature].value_counts()
    sns.barplot(x=unique_values.index, y=unique_values.values)
    plt.title(f'Distribution of {Feature}')
    plt.xlabel(Feature)
    plt.ylabel('Frequency')
    plt.xticks(rotation=90)
    plt.show()
```

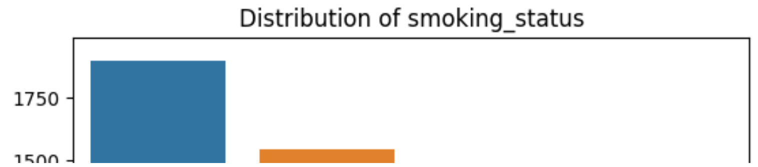

Distribution of gender
Frequency
3000
2500
2000
1500
1000
500
0
Female
Male
Other
gender

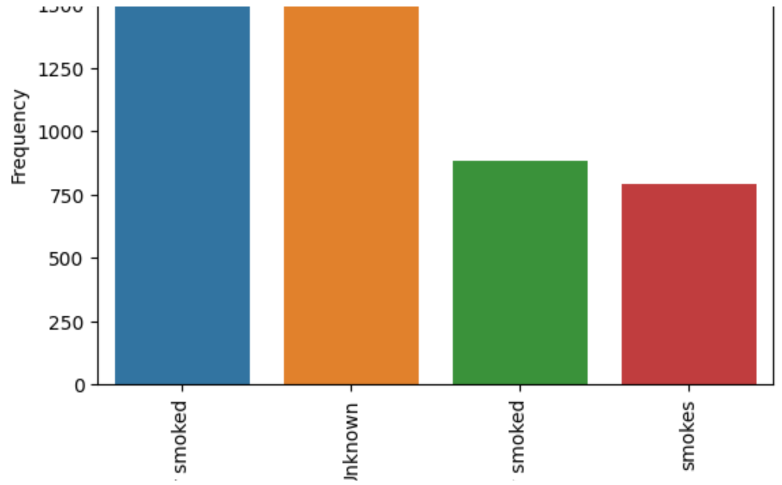

Distribution of ever_married
Frequency
3500
3000
2500
2000
1500
1000
500
0
Yes
No
ever_married


Distribution of work_type
3000


Frequency
2500
2000
1500
1000
500
0
Private
Self-employed
children
Govt_job
Never_worked
work_type

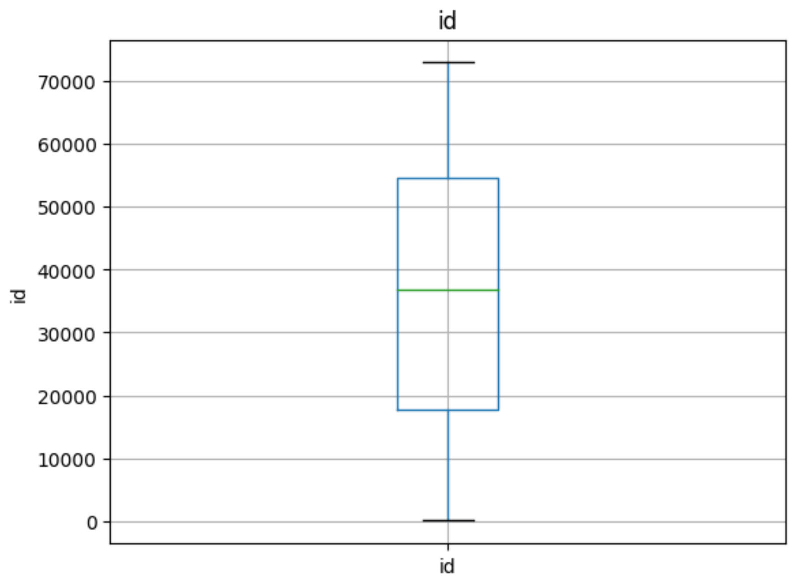

Distribution of Residence_type
Frequency
2500
2000
1500
1000
500
0
Urban
Rural
Residence_type

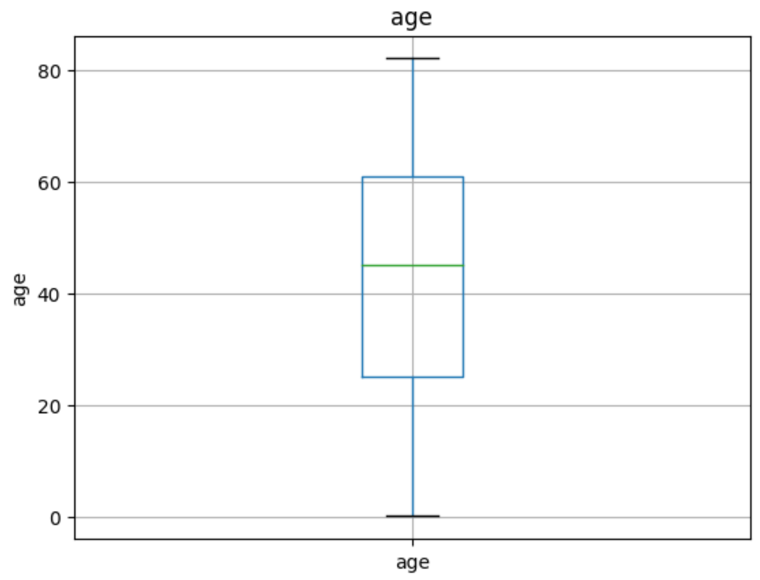

Distribution of smoking_status
1750
1500

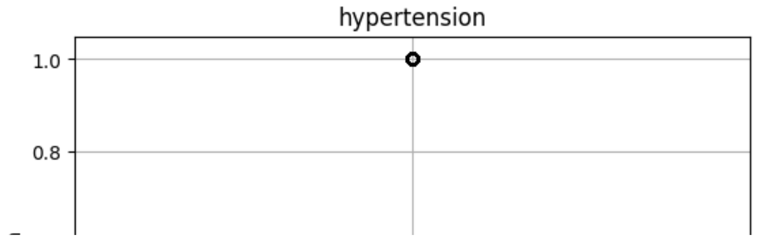


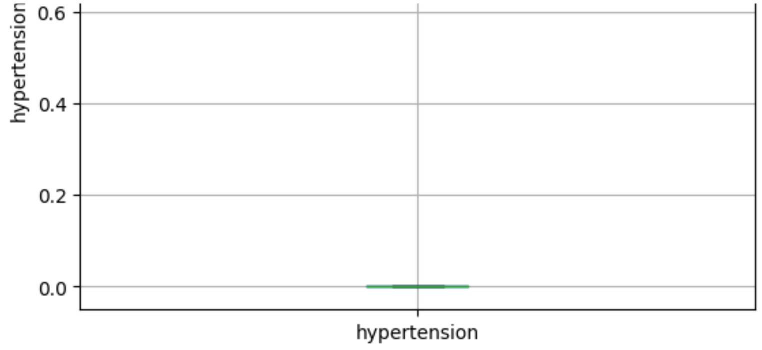
Create box plots to identify outliers and understand the spread of data

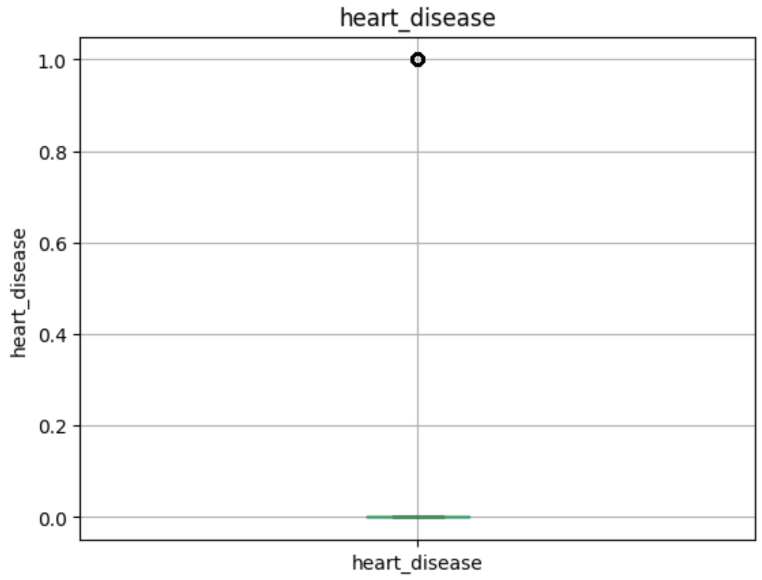
```
import numpy as np
for feature in numerical_variable:
      dataset.boxplot(column=feature)
      plt.ylabel(feature)
      plt.title(feature)
      plt.show()
```

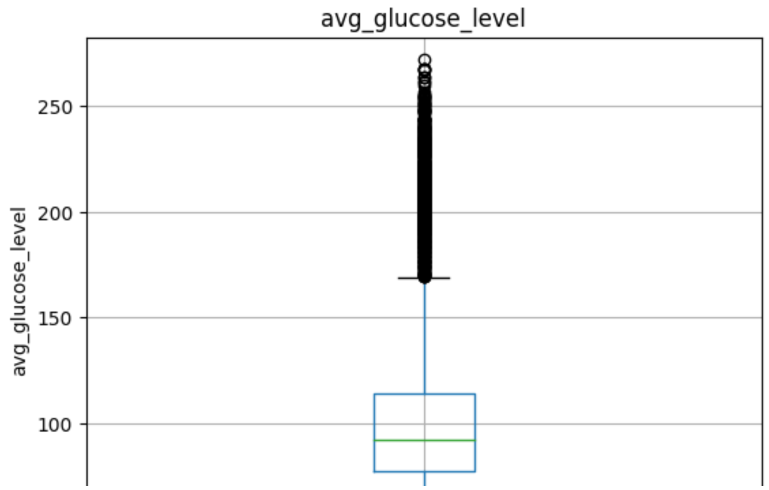

id
70000
60000
50000
40000
30000
20000
10000
0
id
id

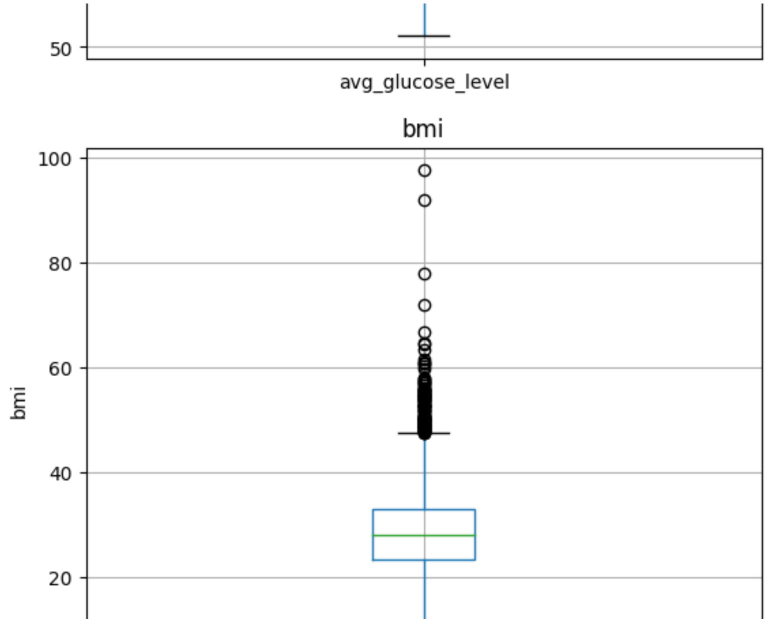

age
80
60
40
20
0
age
age


hypertension
1.0
0.8


hypertension
0.6
0.4
0.2
0.0
hypertension

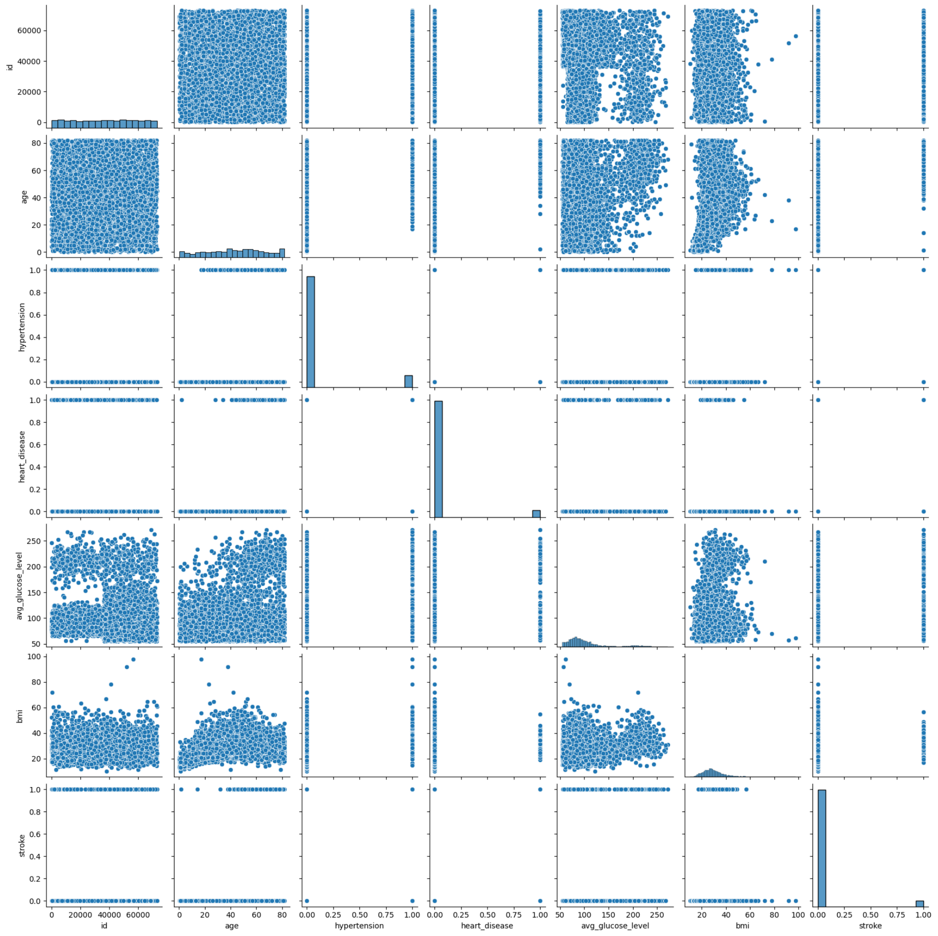

heart_disease
1.0
0.8
0.6
0.4
0.2
0.0
heart_disease


avg_glucose_level
250
200
150
100
avg_glucose_level



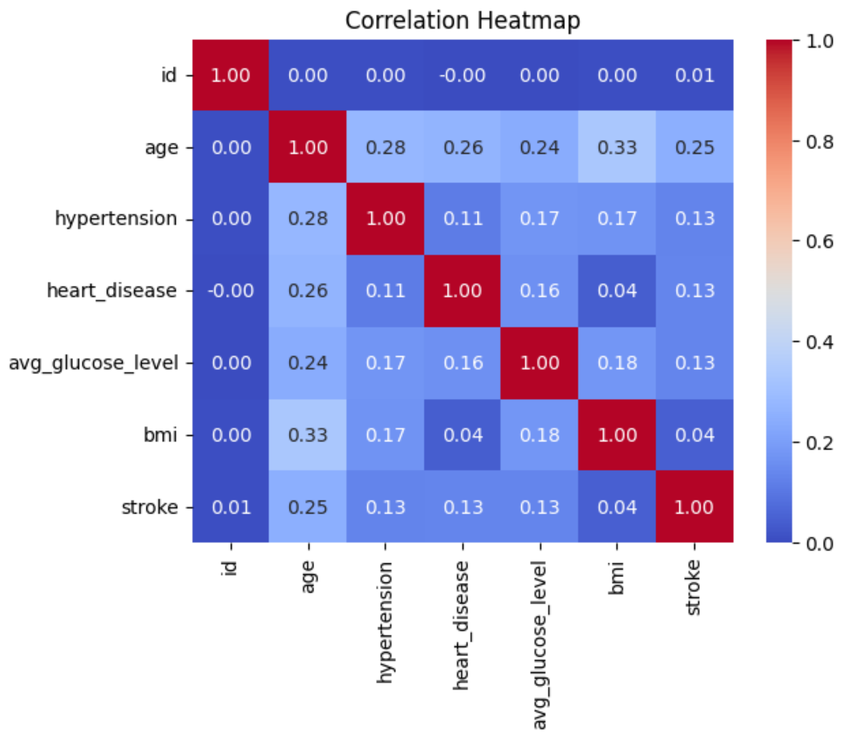
Construct scatter plots to explore relationships between pairs of variables.

```
sns.pairplot(dataset[numerical_variable])
plt.show()
```


id
age
hypertension
heart_disease
avg_glucose_level
bmi
stroke

Use heatmaps to visualize correlations between variables.

```
correlation_matrix = dataset[numerical_variable].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```

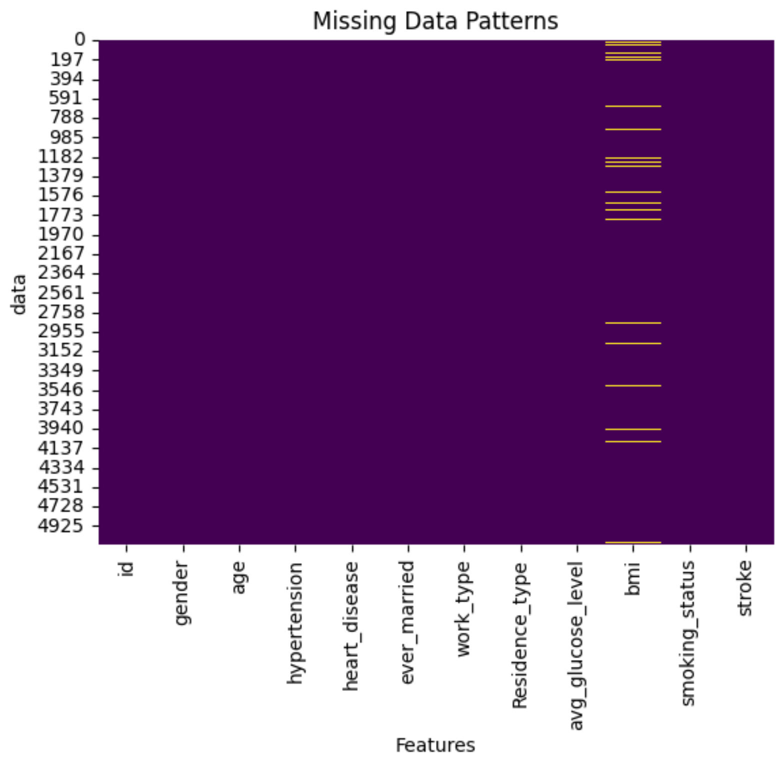


	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
id	1.00	0.00	0.00	-0.00	0.00	0.00	0.01
age	0.00	1.00	0.28	0.26	0.24	0.33	0.25
hypertension	0.00	0.28	1.00	0.11	0.17	0.17	0.13
heart_disease	-0.00	0.26	0.11	1.00	0.16	0.04	0.13
avg_glucose_level	0.00	0.24	0.17	0.16	1.00	0.18	0.13
bmi	0.00	0.33	0.17	0.04	0.18	1.00	0.04
stroke	0.01	0.25	0.13	0.13	0.13	0.04	1.00

Task 3. Handling Missing Data:

Explore the patterns of missing data across variables

```
missing_data_matrix = dataset.isnull()

sns.heatmap(missing_data_matrix, cmap='viridis', cbar=False)
plt.title('Missing Data Patterns')
plt.xlabel('Features')
plt.ylabel('data')
plt.show()
```



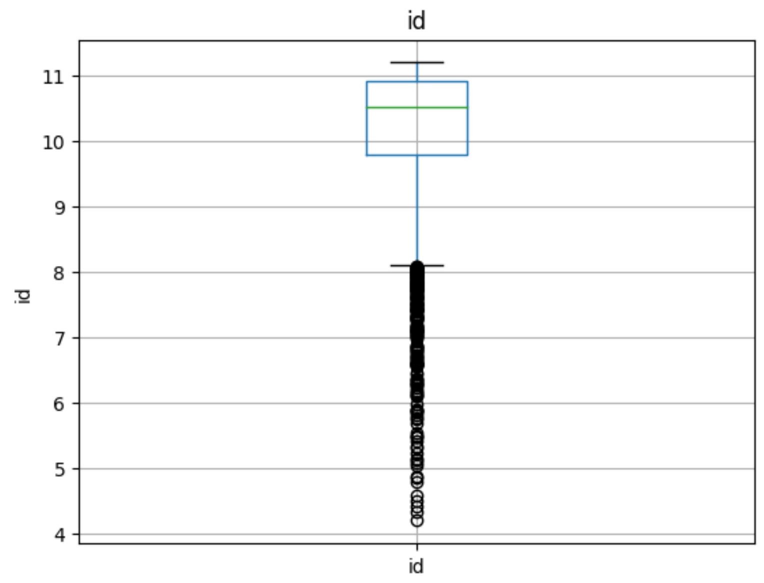
Decide on an appropriate strategy for handling missing values (imputation, removal, etc.)

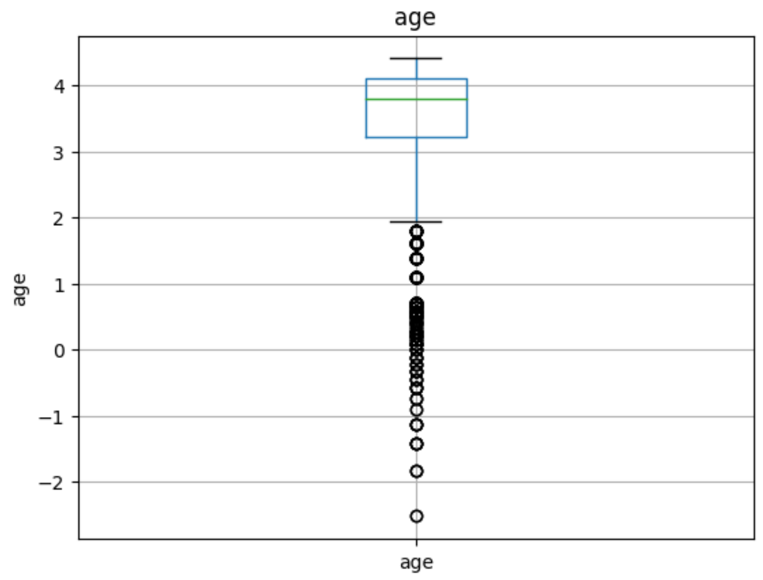
We can fill the missing values be calcuting distances with other samples and fill the missing value with closest neighbour.

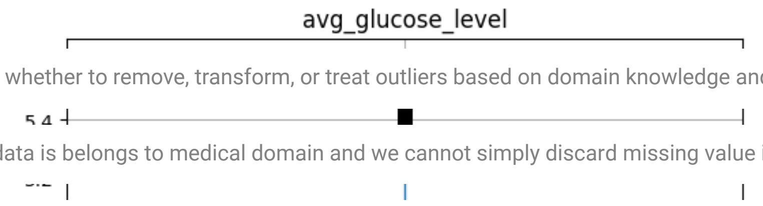
Task 4. Outlier Detection and Treatment:

Identify and visualize outliers in numerical variables.

```
for Feature in numerical_variable:
    data=dataset.copy()
    if 0 in data[Feature].unique():
        pass
    else:
        data[Feature]=np.log(data[Feature])
        data.boxplot(column=Feature)
        plt.ylabel(Feature)
        plt.title(Feature)
        plt.show()
```







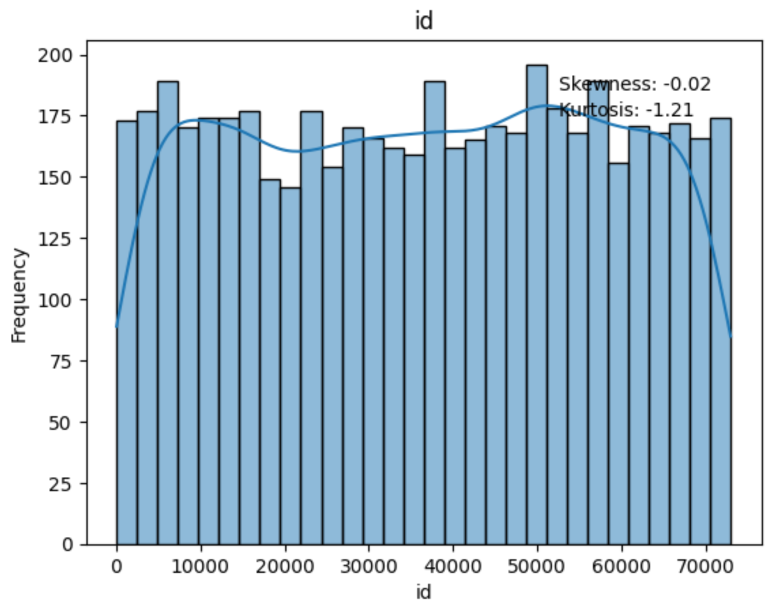
Decide whether to remove, transform, or treat outliers based on domain knowledge and analysis goals.

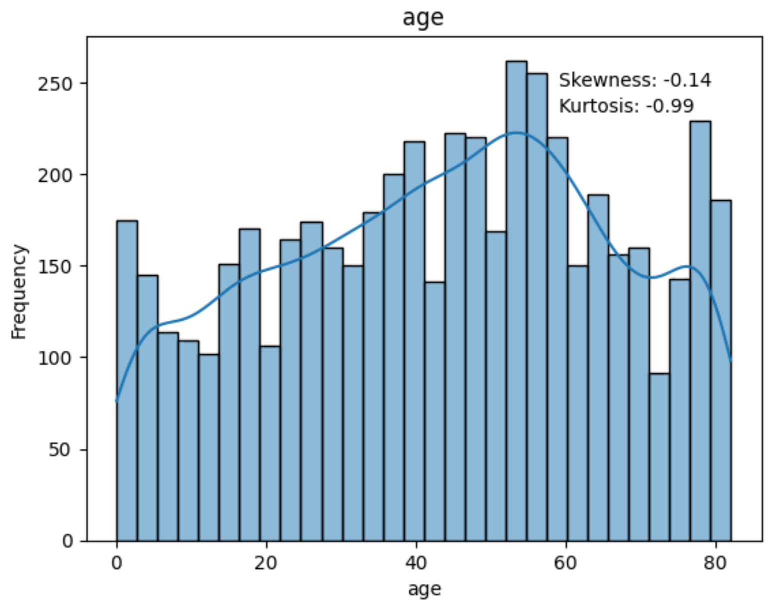
Since data is belongs to medical domain and we cannot simply discard missing value instead we can normalize by taking log function

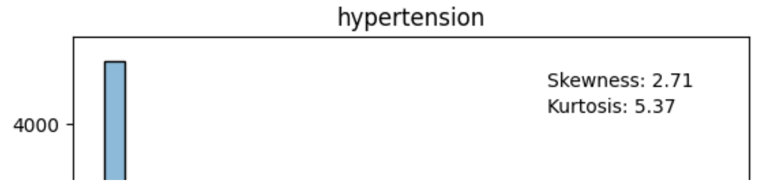
Task 5. Data Distribution Analysis:

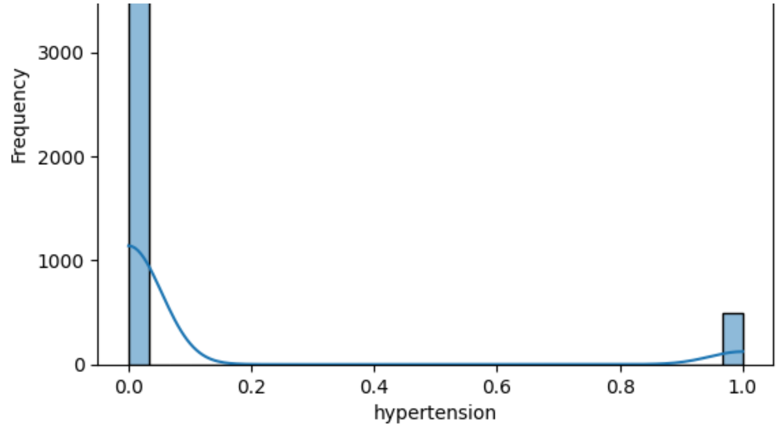
Visualize the data distribution and assess skewness and kurtosis

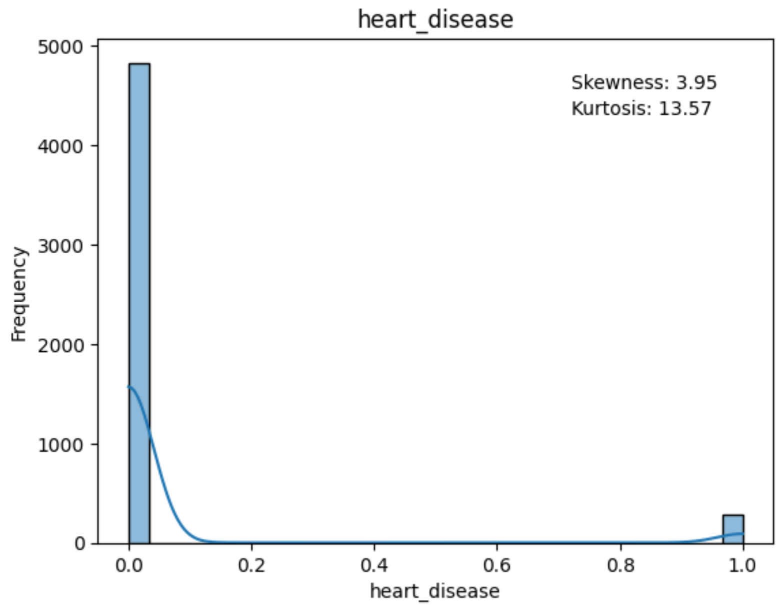
```
from scipy.stats import skew, kurtosis

for Feature in numerical_variable:
    sns.histplot(dataset[Feature], kde=True, bins=30)
    skewness = skew(dataset[Feature])
    kurt = kurtosis(dataset[Feature])
    plt.text(0.7, 0.9, f'Skewness: {skewness:.2f}', transform=plt.gca().transAxes)
    plt.text(0.7, 0.85, f'Kurtosis: {kurt:.2f}', transform=plt.gca().transAxes)
    plt.title(f'{Feature}')
    plt.xlabel(Feature)
    plt.ylabel('Frequency')
    plt.show()
```

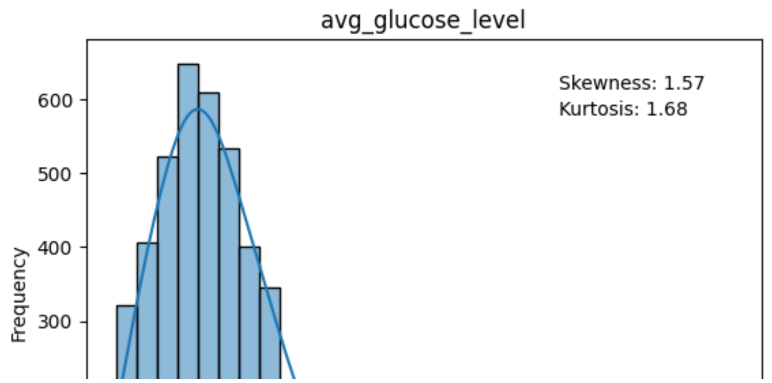

id
Skewness: -0.02
Kurtosis: -1.21
Frequency
id
0
25
50
75
100
125
150
175
200
0
10000
20000
30000
40000
50000
60000
70000

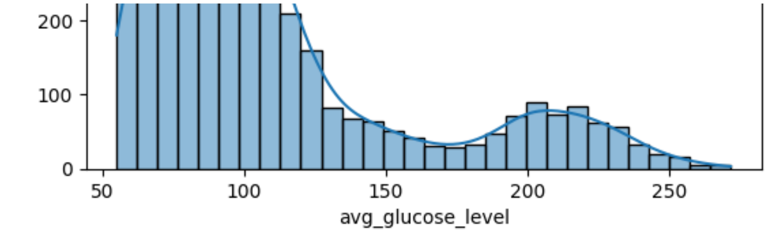

age
Skewness: -0.14
Kurtosis: -0.99
Frequency
age
0
50
100
150
200
250
0
20
40
60
80

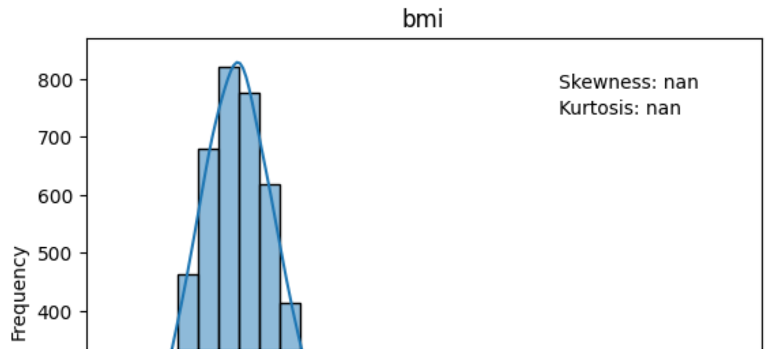

hypertension
Skewness: 2.71
Kurtosis: 5.37
4000


Frequency
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
hypertension


heart_disease
Skewness: 3.95
Kurtosis: 13.57
Frequency
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
heart_disease


avg_glucose_level
Skewness: 1.57
Kurtosis: 1.68
Frequency
600
500
400
300

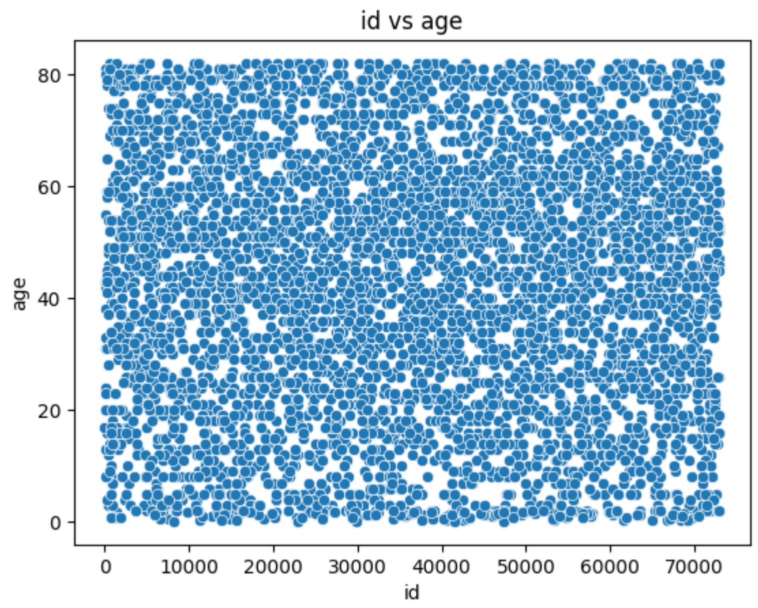


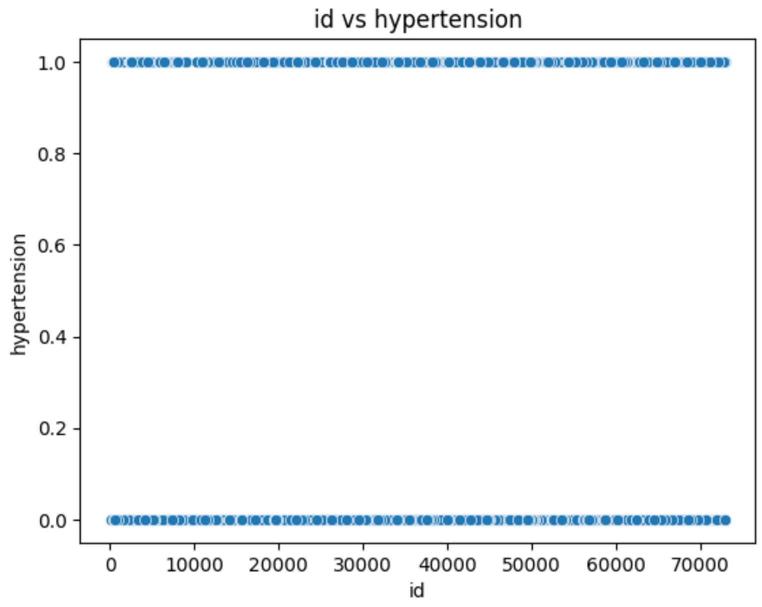


Task 6. Bivariate Analysis:

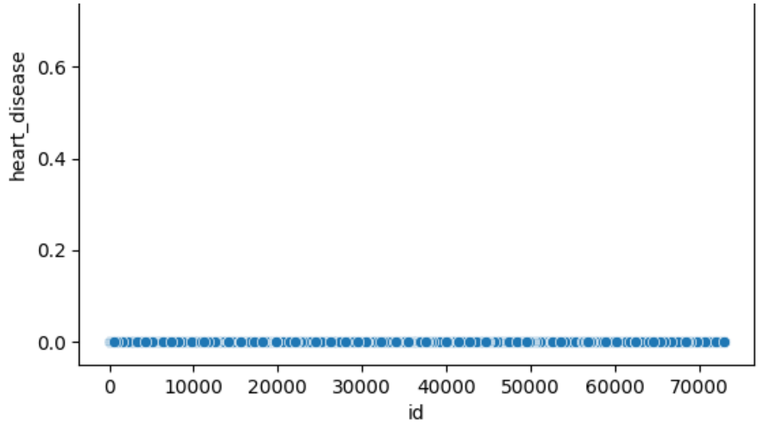
Analyze relationships between pairs of variables through scatter plots

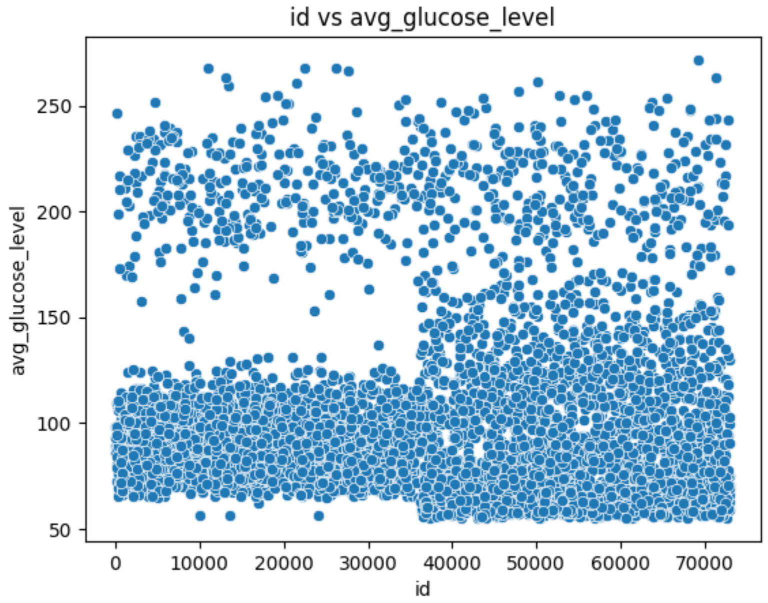
```
for i in range(len(numerical_variable)):
    for j in range(i + 1, len(numerical_variable)):
        sns.scatterplot(data=dataset, x=numerical_variable[i], y=numerical_variable[j])
        plt.xlabel(numerical_variable[i])
        plt.ylabel(numerical_variable[j])
        plt.title(f'{numerical_variable[i]} vs {numerical_variable[j]}')
        plt.show()
```

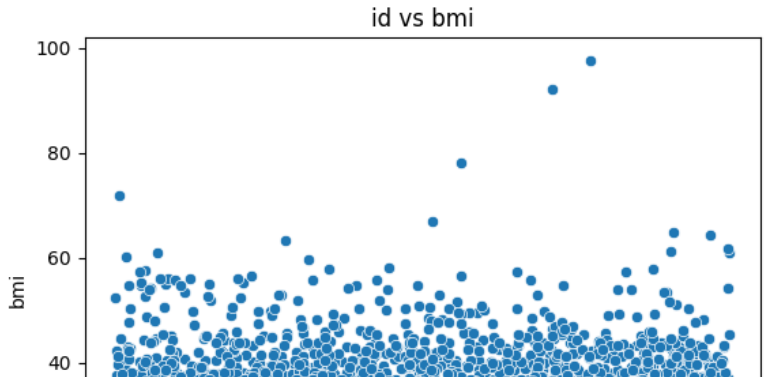

id vs age
age
80
60
40
20
0
0
10000
20000
30000
40000
50000
60000
70000
id

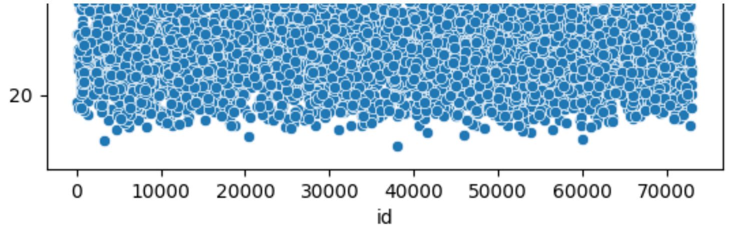

id vs hypertension
hypertension
1.0
0.8
0.6
0.4
0.2
0.0
0
10000
20000
30000
40000
50000
60000
70000
id

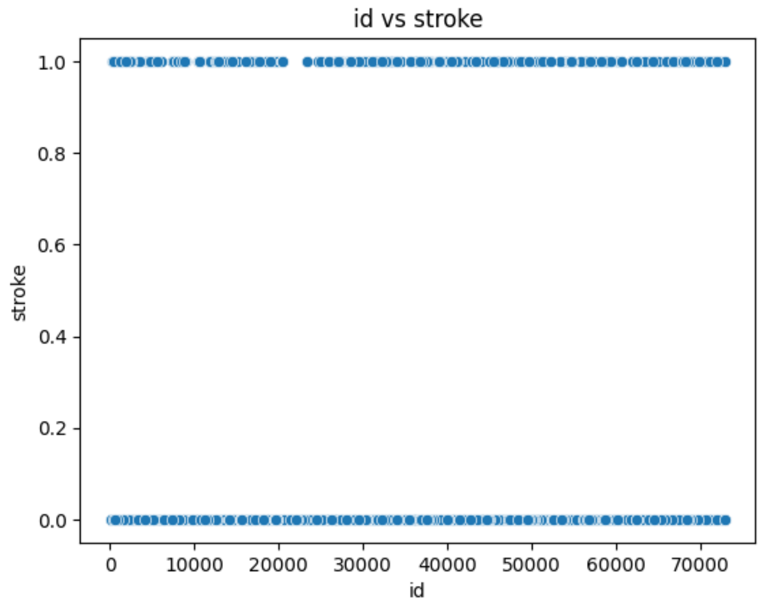
id vs heart_disease
1.0
0.8

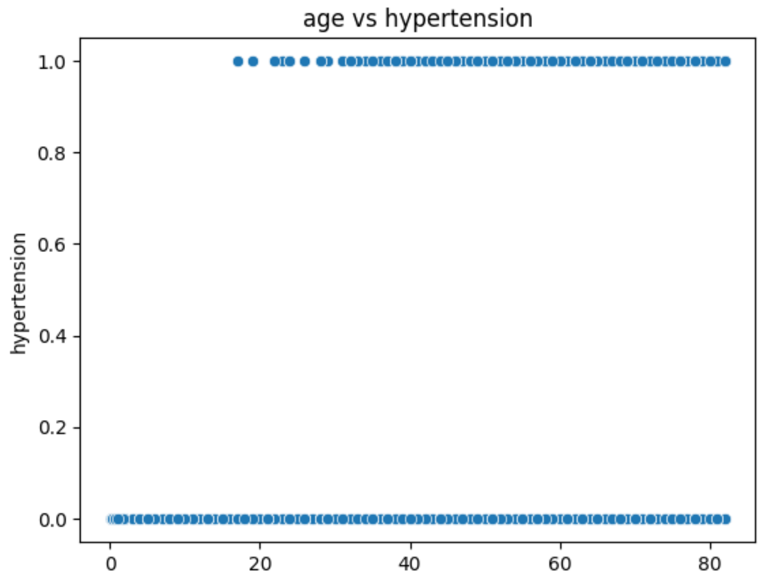

heart_disease
0.6
0.4
0.2
0.0
0
10000
20000
30000
40000
50000
60000
70000
id


id vs avg_glucose_level
avg_glucose_level
250
200
150
100
50
0
10000
20000
30000
40000
50000
60000
70000
id

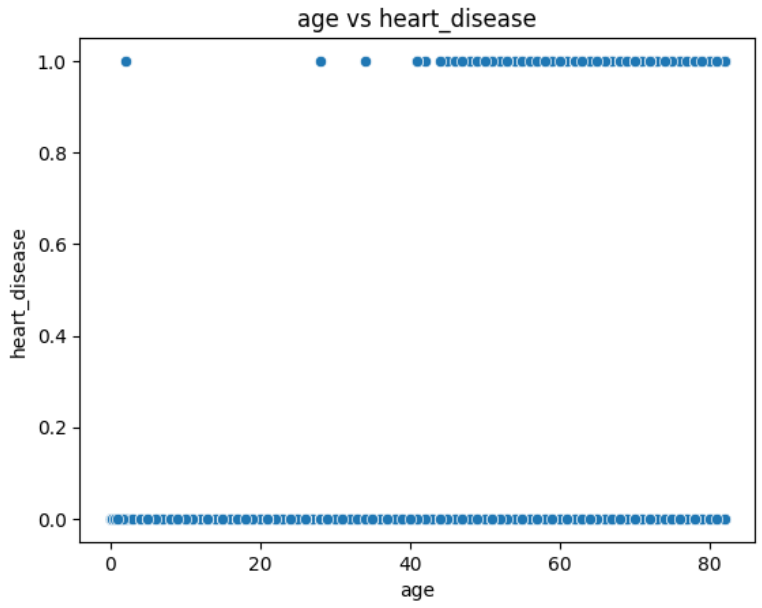

id vs bmi
bmi
100
80
60
40

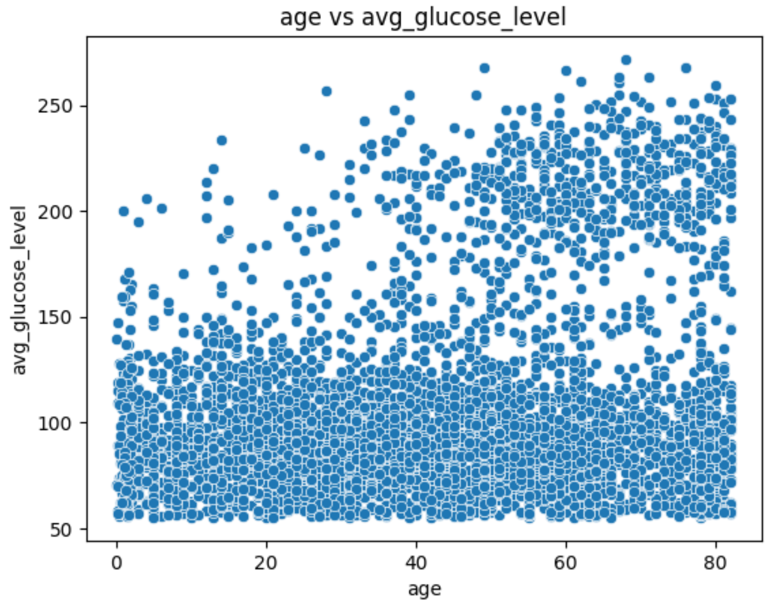

20
0
10000
20000
30000
40000
50000
60000
70000
id

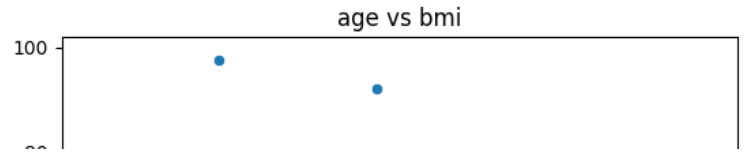

id vs stroke
stroke
1.0
0.8
0.6
0.4
0.2
0.0
0
10000
20000
30000
40000
50000
60000
70000
id

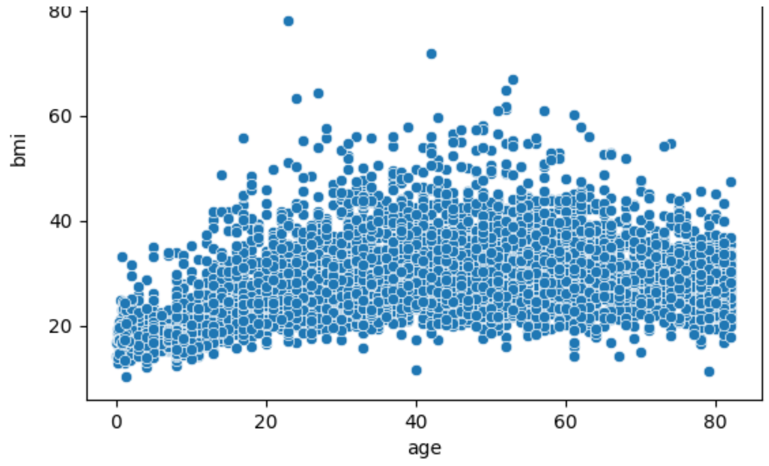

age vs hypertension
hypertension
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80

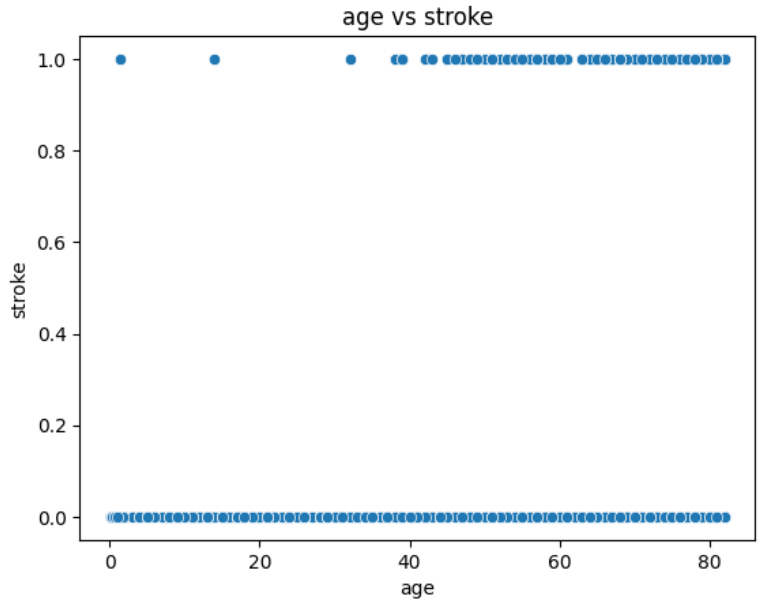
age

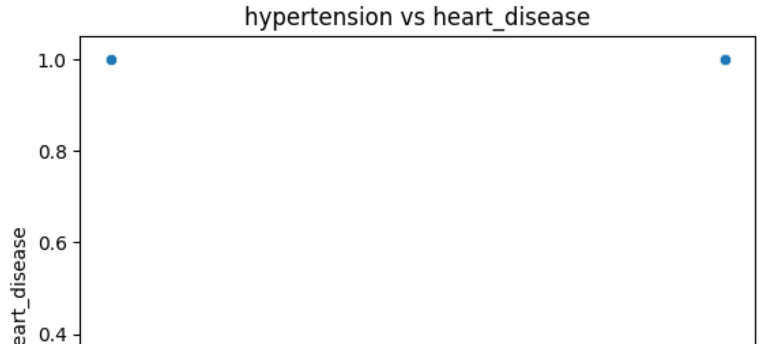

age vs heart_disease
heart_disease
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
age

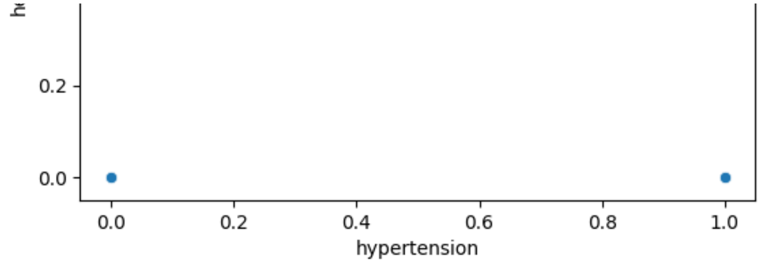

age vs avg_glucose_level
avg_glucose_level
250
200
150
100
50
0
20
40
60
80
age

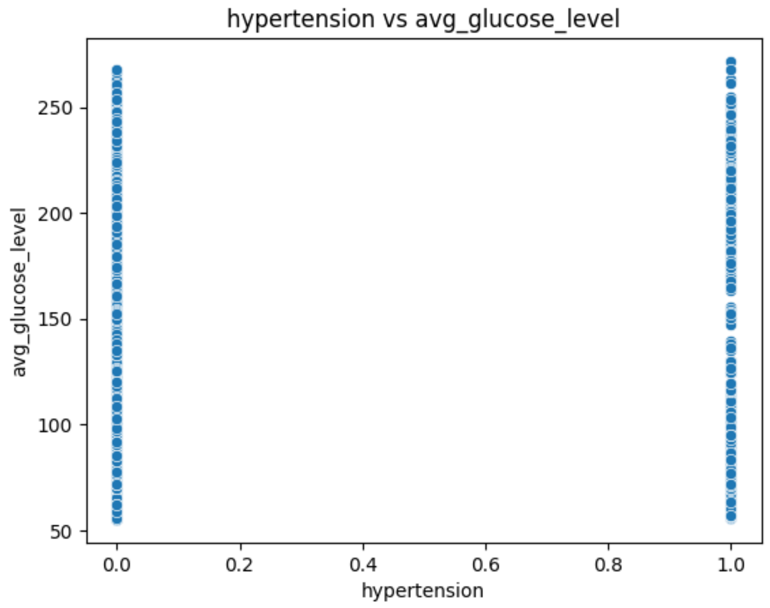

age vs bmi
100

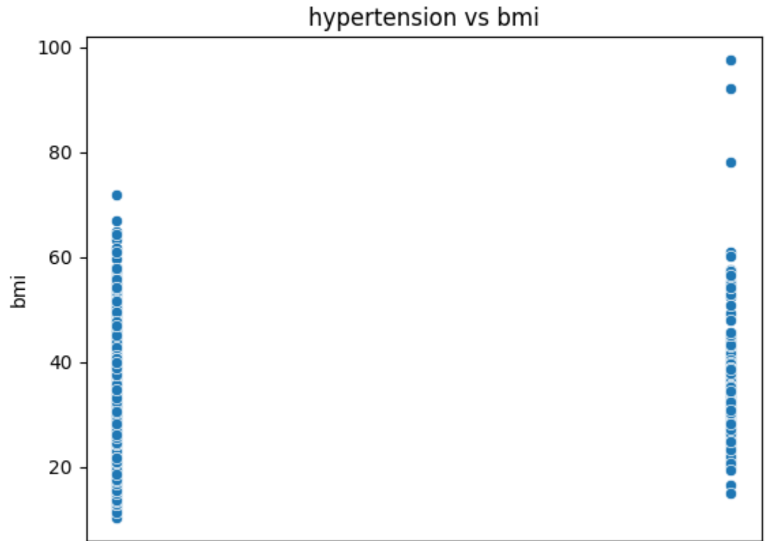

bmi
80
60
40
20
0
20
40
60
80
age

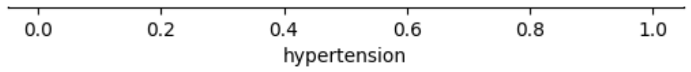

age vs stroke
1.0
0.8
0.6
0.4
0.2
0.0
stroke
0
20
40
60
80
age

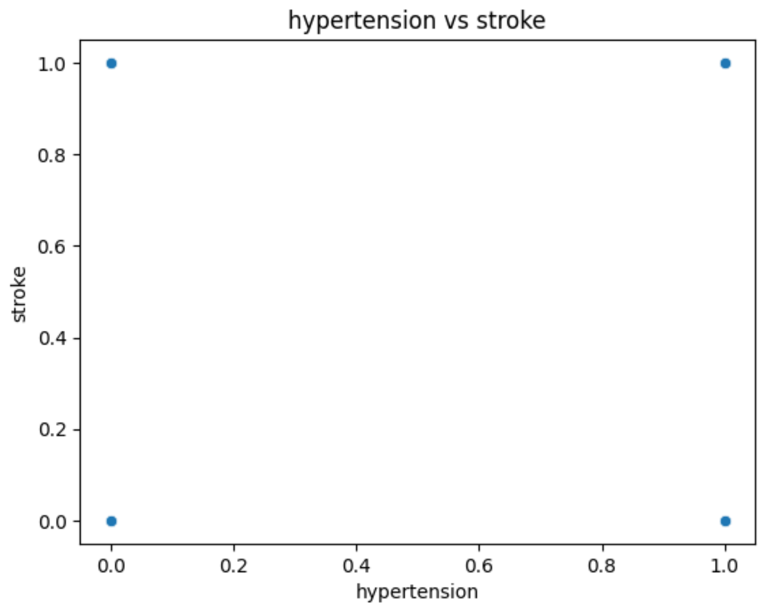

hypertension vs heart_disease
1.0
0.8
0.6
0.4
eart_disease

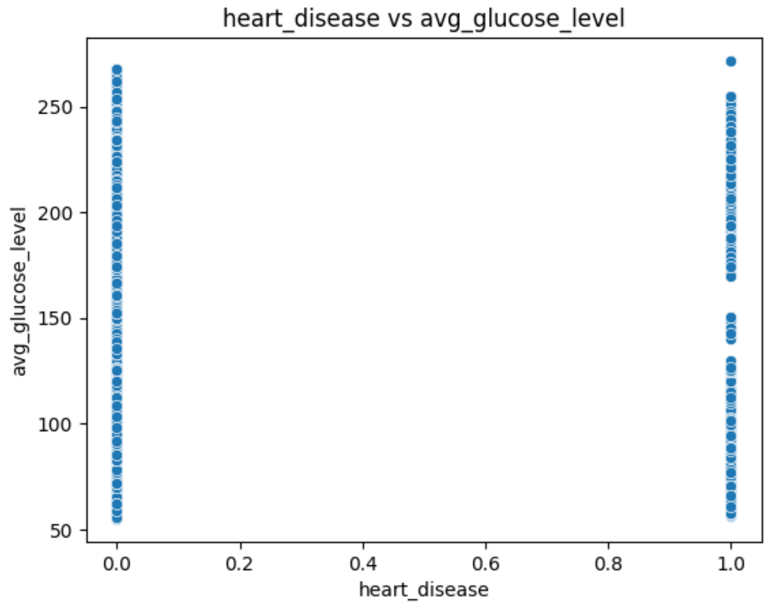

he
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
hypertension

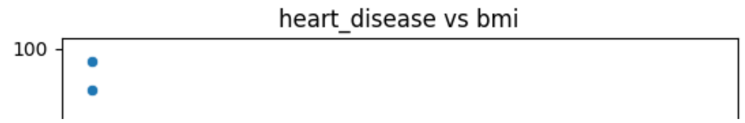

hypertension vs avg_glucose_level
avg_glucose_level
250
200
150
100
50
0.0
0.2
0.4
0.6
0.8
1.0
hypertension


hypertension vs bmi
bmi
100
80
60
40
20


0.0
0.2
0.4
0.6
0.8
1.0
hypertension


hypertension vs stroke
stroke
hypertension


heart_disease vs avg_glucose_level
avg_glucose_level
heart_disease


heart_disease vs bmi
100

bmi
80
60
40
20

Task 7. Grouping and Aggregation:

Group data by categorical variables and calculate summary statistics within each group.

```
for Feature in categorical_variable:
  grouped = dataset.groupby(Feature)
  summary_statistics = grouped[numerical_variable].agg(['mean', 'median', 'std', 'count'])
  summary_statistics.columns = [f'{col}_{agg}' for col, agg in summary_statistics.columns]
  summary_statistics.reset_index(inplace=True)
  print(summary_statistics)
```

```
   2        Private  36951.227009    37492.0  21257.455472       2925  45.503932
   3  Self-employed  35551.288156    35315.0  20828.468932        819  60.201465
   4       children  35769.432314    35106.0  21005.291797        687   6.841339

      age_median    age_std  age_count  hypertension_mean  ...  \
   0        51.0  15.438879        657           0.111111  ...
   1        16.0   2.342899         22           0.000000  ...
   2        45.0  18.444200       2925           0.096068  ...
   3        63.0  16.835961        819           0.175824  ...
   4         6.0   4.533364        687           0.000000  ...

      avg_glucose_level_std  avg_glucose_level_count   bmi_mean  bmi_median  \
   0              47.697200                      657  30.522063       29.40
   1              28.697132                       22  25.545455       23.15
   2              45.886102                     2925  30.304625       28.90
   3              51.719490                      819  30.211871       29.40
```

```
      1            52.256627                      885  30.747192        29.8
```

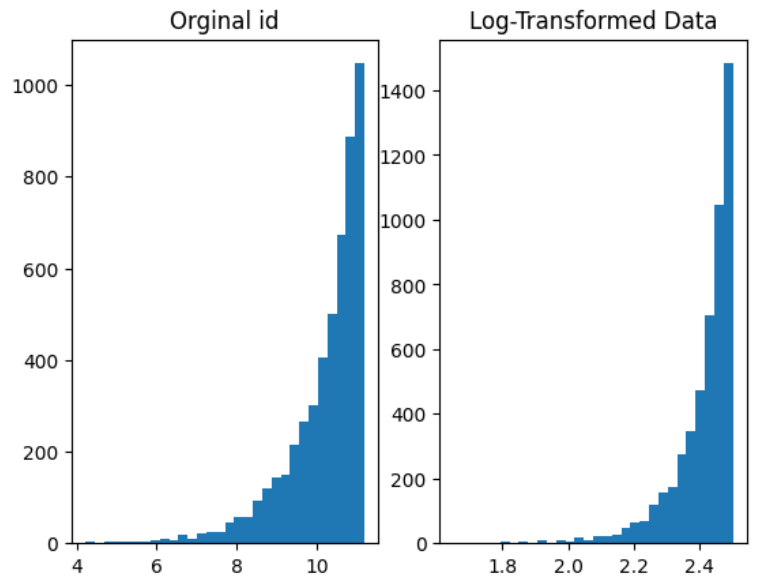
Explore differences or patterns between different groups

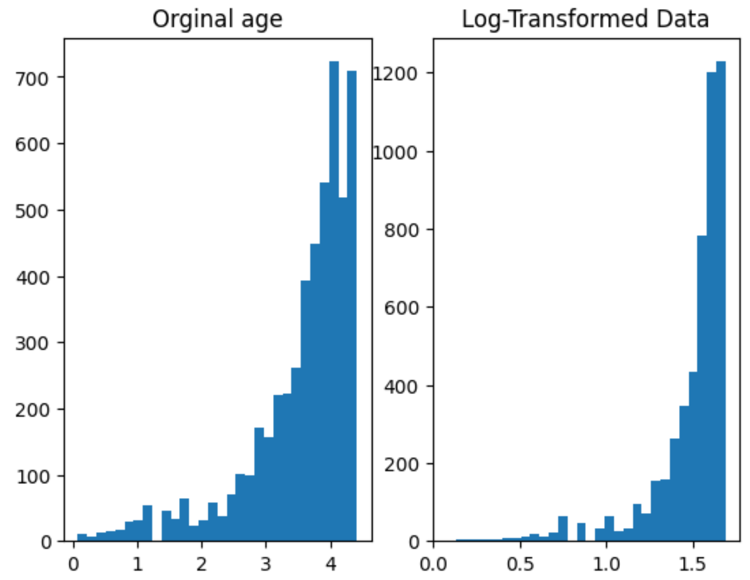
These conclusions provide a detailed overview of the summary statistics for each categorical variable and how they relate to various numerical fields, including age, hypertension, BMI, and stroke. They offer insights into potential patterns and trends within the dataset based on different categories.

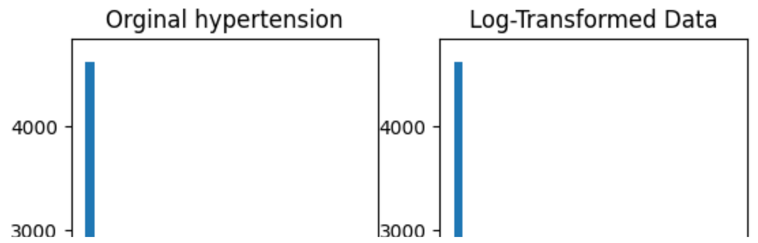
Task 8. Data Transformation:

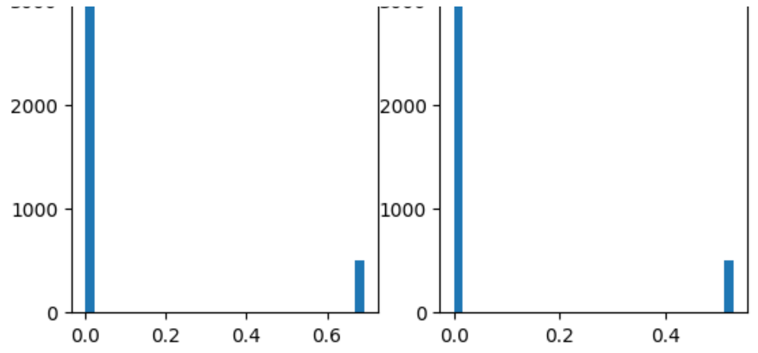
Apply mathematical transformations (e.g., logarithmic or exponential transformations) to normalize data

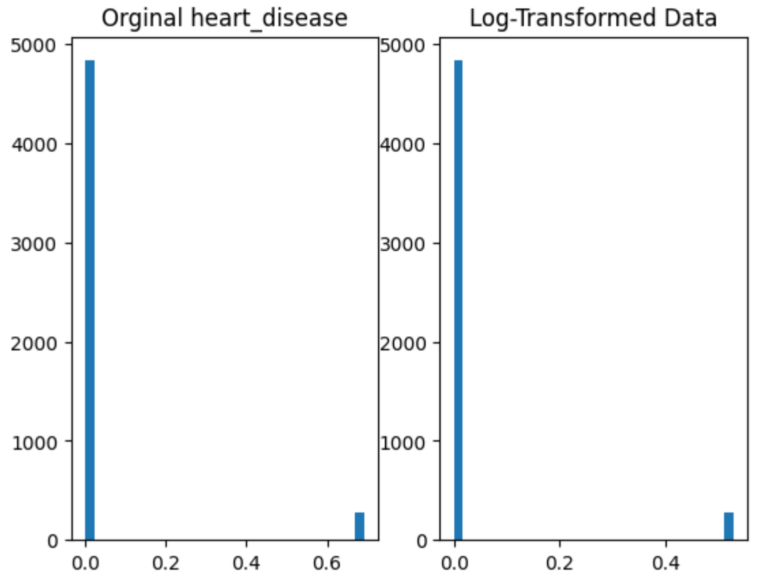
```
for Feature in numerical_variable:
  dataset['log_transformed'] = np.log1p(dataset[Feature])
  plt.subplot(1, 2, 1)
  plt.hist(dataset[Feature], bins=30)
  plt.title(f'Orginal {Feature}')
  plt.subplot(1, 2, 2)
  plt.hist(dataset['log_transformed'], bins=30)
  plt.title('Log-Transformed Data')
  plt.show()
```

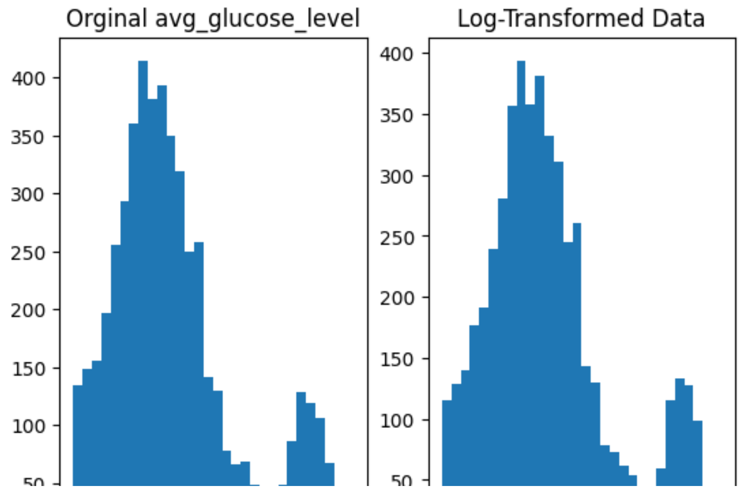

Orginal id
Log-Transformed Data
1000
800
600
400
200
0
4
6
8
10
1400
1200
1000
800
600
400
200
0
1.8
2.0
2.2
2.4

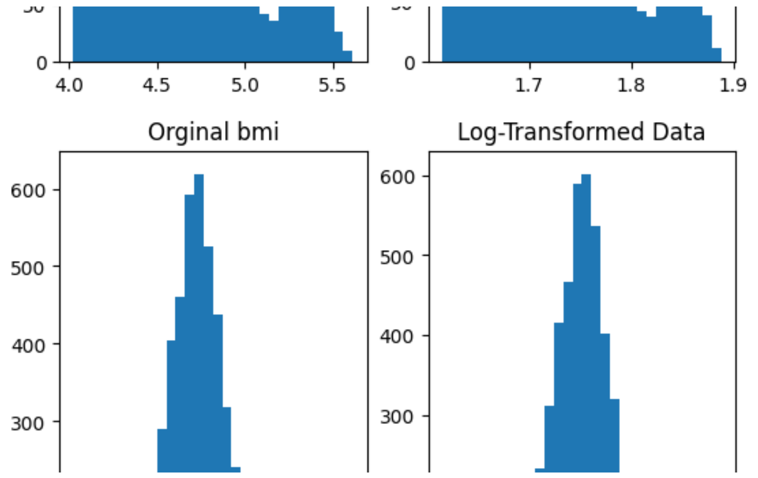

Orginal age
Log-Transformed Data
700
600
500
400
300
200
100
0
0
1
2
3
4
1200
1000
800
600
400
200
0
0.0
0.5
1.0
1.5


Orginal hypertension
Log-Transformed Data
4000
3000
4000
3000


2000
1000
0
0.0
0.2
0.4
0.6
2000
1000
0
0.0
0.2
0.4


Orginal heart_disease
Log-Transformed Data
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.0
0.2
0.4


Orginal avg_glucose_level
Log-Transformed Data
400
350
300
250
200
150
100
50



Convert categorical variables to numerical format using encoding techniques

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

for Feature in categorical_variable:
  data[Feature] = label_encoder.fit_transform(dataset[Feature])

for column in dataset.select_dtypes(include='object').columns:
  label_encoder.fit(dataset[column])
  encoding_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))
  print(f"Encoding Mapping for {column}:")
  print(encoding_mapping)
  print()

data[categorical_variable]
```

```
Encoding Mapping for gender:
{'Female': 0, 'Male': 1, 'Other': 2}

Encoding Mapping for ever_married:
{'No': 0, 'Yes': 1}

Encoding Mapping for work_type:
{'Govt_job': 0, 'Never_worked': 1, 'Private': 2, 'Self-employed': 3, 'children': 4}

Encoding Mapping for Residence_type:
{'Rural': 0, 'Urban': 1}

Encoding Mapping for smoking_status:
{'Unknown': 0, 'formerly smoked': 1, 'never smoked': 2, 'smokes': 3}
```

	gender	ever_married	work_type	Residence_type	smoking_status
0	1	1	2	1	1
1	0	1	3	0	2
2	1	1	2	0	2
3	0	1	2	1	3

These encoding mappings are useful for machine learning and data analysis tasks as they convert categorical data into a format that can be used by algorithms that require numerical inputs. They make it easier to work with categorical data in analytical pipelines and enable the use of these variables in predictive modeling and statistical analyses.

5106	0	1	3	1	2

0s completed at 3:09 PM