```
import pandas as pd

# Load the dataset
df = pd.read_csv('/content/heart_disease_dataset.csv')

# Display the first 5 rows
print("First 5 rows of the dataset:")
display(df.head())

# Display column information (data types and non-null counts)
print("\nColumn information:")
display(df.info())
```

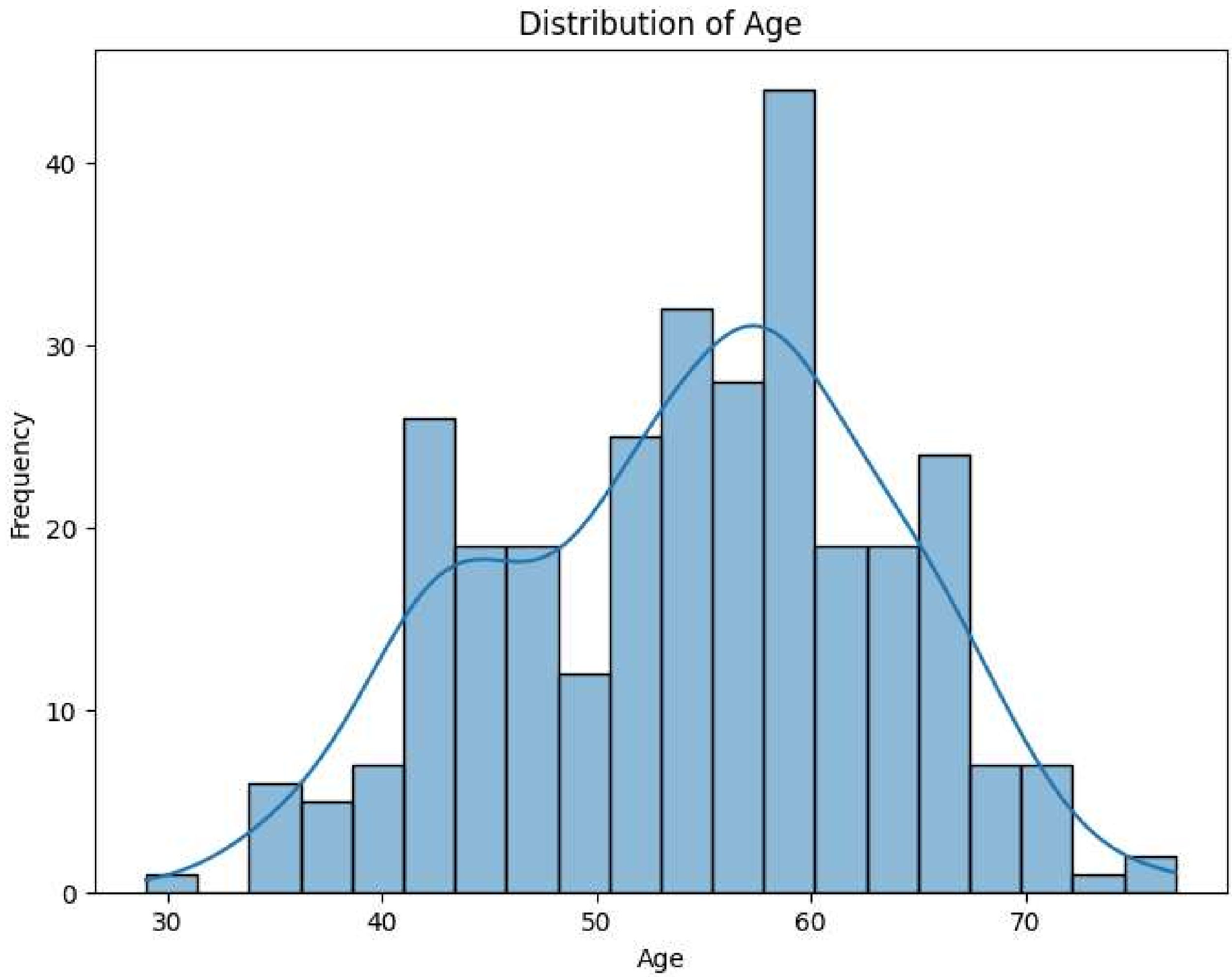
First 5 rows of the dataset:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	
4	57	0	0	120	354	0	1	163	1	0.6	2	0	

```
Column information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trtbps    303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalachh  303 non-null    int64  
 8   exng      303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64  
 11  caa       303 non-null    int64  
 12  thall     303 non-null    int64  
 13  output    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
None
```

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

# Plot a histogram of the 'Age' column
plt.figure(figsize=(8, 6))
sns.histplot(df['age'], bins=20, kde=True)
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```




```
import pandas as pd

# Load the dataset
df = pd.read_csv('/content/heart_disease_dataset.csv')

# Display the first 5 rows
print("First 5 rows of the dataset:")
display(df.head())

# Display column information (data types and non-null counts)
print("\nColumn information:")
display(df.info())
```

First 5 rows of the dataset:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	
4	57	0	0	120	354	0	1	163	1	0.6	2	0	

```
Column information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trtbps    303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalachh  303 non-null    int64
 8   exng      303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64
 11  caa       303 non-null    int64
 12  thall     303 non-null    int64
 13  output    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
None
```

```
# Check for missing values
print("Missing values before handling:")
display(df.isnull().sum())

# Verify that missing values have been handled
print("\nMissing values after handling:")
display(df.isnull().sum())
```

```
Missing values before handling:
```

	0
age	0

```
sns.scatterplot
```

```
seaborn.relational.scatterplot
def scatterplot(data=None, *, x=None, y=None, hue=None, size=None, style=None,
palette=None, hue_order=None, hue_norm=None, sizes=None, size_order=None,
size_norm=None, markers=True, style_order=None, legend='auto', ax=None,
**kwargs)

Draw a scatter plot with possibility of several semantic groupings.

The relationship between `x` and `y` can be shown for different subsets
of the data using the `hue`, `size`, and `style` parameters. These
parameters control what visual semantics are used to identify the different
```

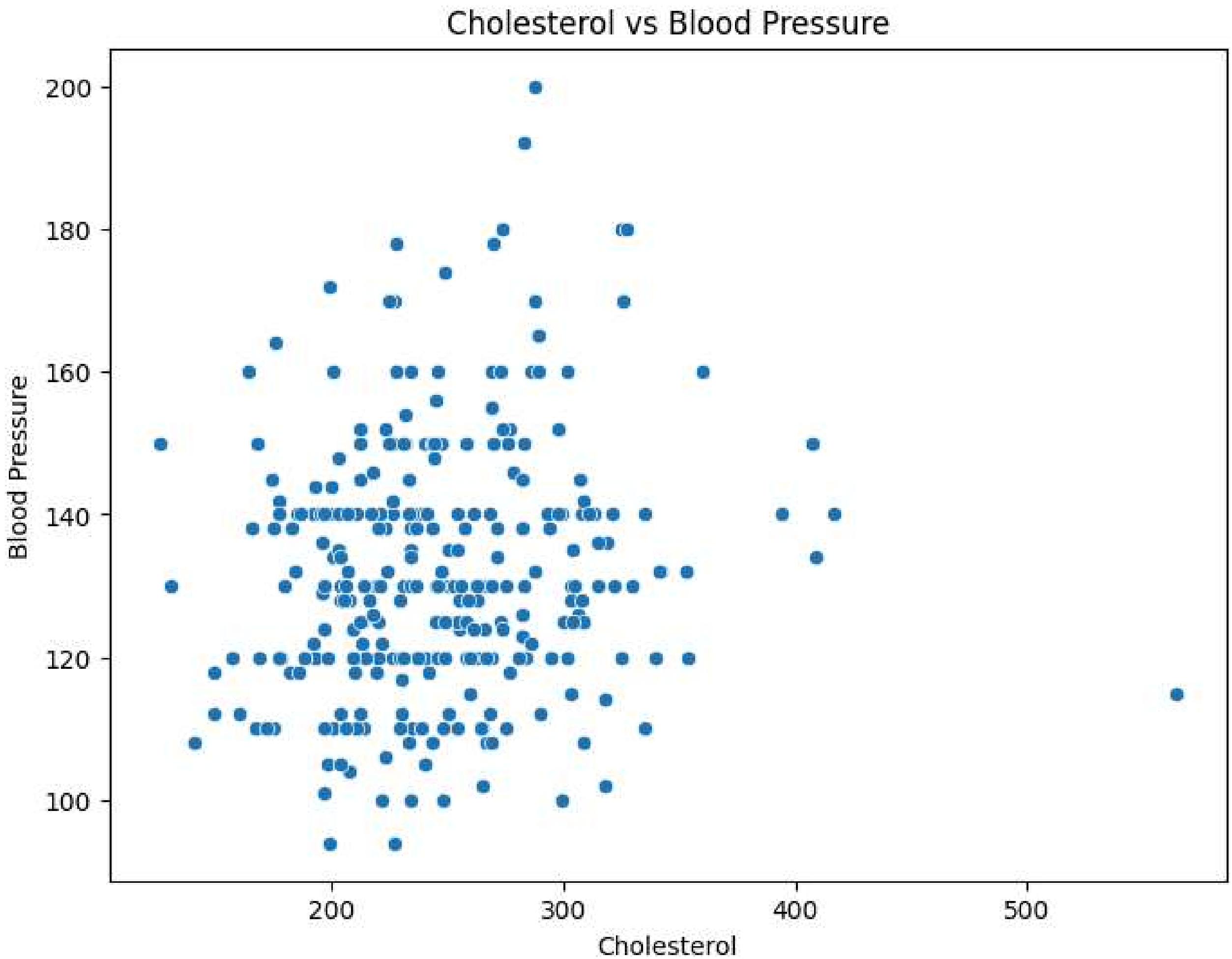

restecg	0
thalachh	0
exng	0
oldpeak	0


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

# Create a scatterplot of 'Cholesterol' vs 'Blood Pressure'
plt.figure(figsize=(8, 6))
sns.scatterplot(x='chol', y='trtbps', data=df)
plt.title('Cholesterol vs Blood Pressure')
plt.xlabel('Cholesterol')
plt.ylabel('Blood Pressure')
plt.show()
```

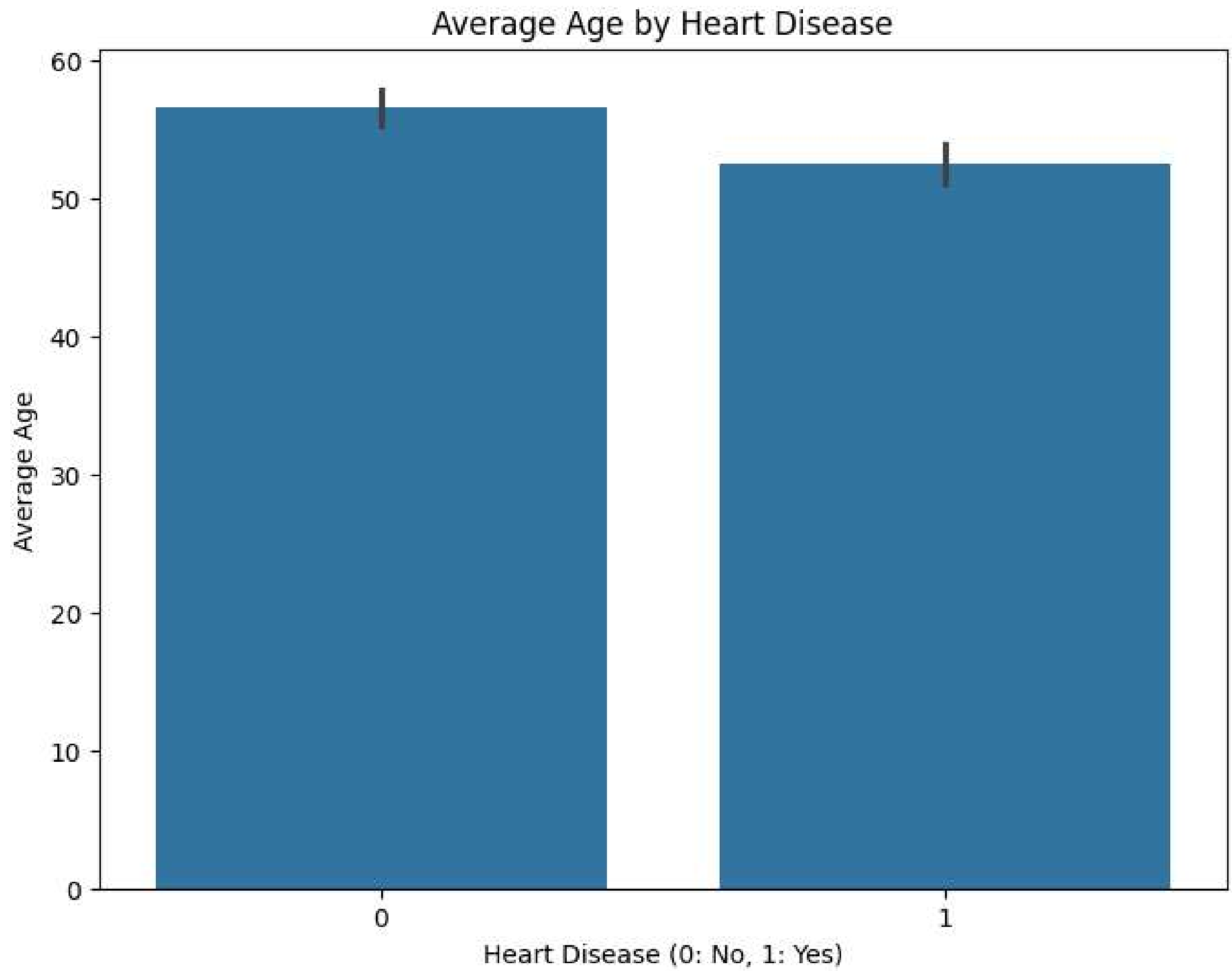
age	0
sex	0
cp	0
trtbps	0
chol	0
fbs	0
restecg	0
thalachh	0
exng	0
oldpeak	0
slp	0
caa	0

thall 0




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

# Create a barplot of average age by heart disease
plt.figure(figsize=(8, 6))
sns.barplot(x='output', y='age', data=df)
plt.title('Average Age by Heart Disease')
plt.xlabel('Heart Disease (0: No, 1: Yes)')
plt.ylabel('Average Age')
plt.show()
```




```
df.columns
```

```
Index(['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh',
       'exng', 'oldpeak', 'slp', 'caa', 'thall', 'output'],
      dtype='object')
```

```
# Identify object type columns
print("Object type columns:")
for col in df.columns:
    if df[col].dtype == 'object':
        print(col)
```

```
Object type columns:
```

```
# Identify object type columns
# In this dataset, all columns are numerical as seen from df.info()
# There are no categorical columns to encode based on the current dataframe struct
# If there were object type columns, we would identify them here and decide on enc
```

```
# For this dataset, we can skip the encoding step as there are no object columns.

print("No object type columns to encode in this dataset based on df.info().")

# If you had object columns and wanted to define encoding methods, you would do it
# encoding_methods = {col: 'One-Hot Encoding' for col in categorical_cols}
# print("Chosen encoding methods for each categorical column:")
# for col, method in encoding_methods.items():
#     print(f"{col}: {method}")
```

```
No object type columns to encode in this dataset based on df.info().
```

```
# import from sklearn.preprocessing import OneHotEncoder

# Based on the analysis, there are no object type columns to encode in this datase
# Therefore, the one-hot encoding step is not necessary for this dataset.
# The dataframe 'df' is already in a suitable format for modeling.

# If you had categorical columns to encode, the code would be as follows:
# categorical_cols = list(encoding_methods.keys())
# encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
# encoded_data = encoder.fit_transform(df[categorical_cols])
# encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(ca
# df = pd.concat([df.drop(columns=categorical_cols), encoded_df], axis=1)

# print("DataFrame after checking for one-hot encoding:")
# display(df.head())
# print("\nColumn information after checking for encoding:")
# display(df.info())

print("Skipping one-hot encoding as there are no object columns in the dataset.")
```

```
Skipping one-hot encoding as there are no object columns in the dataset.
```

```
import pandas as pd
# from sklearn.preprocessing import OneHotEncoder

# Load the dataset again (This cell seems redundant as df is already loaded)
# try:
#     df = pd.read_csv('/content/heart_disease_dataset.csv')
# except FileNotFoundError:
#     print("Error: heart_disease_dataset.csv not found. Please make sure the file
#     df = None

# Based on the analysis, there are no object type columns to encode in this datase
# Therefore, the one-hot encoding step is not necessary for this dataset.
# The dataframe 'df' is already in a suitable format for modeling.

# if df is not None:
#     categorical_cols = list(encoding_methods.keys()) # Need to ensure encoding_m
```

```
#     encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
#     encoded_data = encoder.fit_transform(df[categorical_cols])
#     encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_ou
#     df = pd.concat([df.drop(columns=categorical_cols), encoded_df], axis=1)

#     print("DataFrame after one-hot encoding:")
#     display(df.head())
#     print("\nColumn information after encoding:")
#     display(df.info())
# else:
#     print("DataFrame not loaded, cannot proceed with encoding.")

print("Skipping one-hot encoding as there are no object columns in the dataset.")
```

```
Skipping one-hot encoding as there are no object columns in the dataset.
```

```
import pandas as pd
# from sklearn.preprocessing import OneHotEncoder

# Load the dataset again (This cell seems redundant as df is already loaded)
# try:
#     df = pd.read_csv('/content/heart_disease_dataset.csv')
# except FileNotFoundError:
#     print("Error: heart_disease_dataset.csv not found. Please make sure the file
#     # Indicate failure if the file is not found
#     df = None

# Based on the analysis, there are no object type columns to encode in this datase
# Therefore, the one-hot encoding step is not necessary for this dataset.
# The dataframe 'df' is already in a suitable format for modeling.

# if df is not None:
#     categorical_cols = list(encoding_methods.keys()) # Need to ensure encoding_m

#     # Create a OneHotEncoder instance
#     encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

#     # Fit and transform the selected categorical columns
#     encoded_data = encoder.fit_transform(df[categorical_cols])

#     # Create a new DataFrame from the encoded data
#     encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_ou

#     # Concatenate the new encoded DataFrame with the original DataFrame, droppir
#     df = pd.concat([df.drop(columns=categorical_cols), encoded_df], axis=1)

#     # Display the first few rows of the updated DataFrame
#     print("DataFrame after one-hot encoding:")
#     display(df.head())
```

```
#     # Display column information to verify the new columns and data types
#     print("\nColumn information after encoding:")
#     display(df.info())
# else:
#     print("DataFrame not loaded, cannot proceed with encoding.")

print("Skipping one-hot encoding as there are no object columns in the dataset.")
```

```
Skipping one-hot encoding as there are no object columns in the dataset.
```

```
# from sklearn.preprocessing import OneHotEncoder

# Based on the analysis, there are no object type columns to encode in this datase
# Therefore, the one-hot encoding step is not necessary for this dataset.
# The dataframe 'df' is already in a suitable format for modeling.

# if 'encoding_methods' in locals():
#     categorical_cols = list(encoding_methods.keys())
# else:
#     categorical_cols = [] # Or identify based on df.columns if needed

# if categorical_cols:
#     # Create a OneHotEncoder instance
#     encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

#     # Fit and transform the selected categorical columns
#     encoded_data = encoder.fit_transform(df[categorical_cols])

#     # Create a new DataFrame from the encoded data
#     encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_ou

#     # Concatenate the new encoded DataFrame with the original DataFrame, droppir
#     df = pd.concat([df.drop(columns=categorical_cols), encoded_df], axis=1)

#     # Display the first few rows of the updated DataFrame
#     print("DataFrame after one-hot encoding:")
#     display(df.head())

#     # Display column information to verify the new columns and data types
#     print("\nColumn information after encoding:")
#     display(df.info())
# else:
#     print("No categorical columns to encode.")

print("Skipping one-hot encoding as there are no object columns in the dataset.")
```

```
Skipping one-hot encoding as there are no object columns in the dataset.
```

```
import pandas as pd
# from sklearn.preprocessing import OneHotEncoder
```

```
# Load the dataset
try:
    df = pd.read_csv('/content/heart_disease_dataset.csv')
except FileNotFoundError:
    print("Error: heart_disease_dataset.csv not found. Please make sure the file i
    df = None

if df is not None:
    # Check for missing values
    print("Missing values before handling:")
    display(df.isnull().sum())

    # Based on the dataset info, there are no missing values and no object columns
    # Removing the missing value handling for 'Alcohol Intake' as it's not in the
    # Removing the encoding part as there are no object columns.

    # Identify object type columns - this will be empty based on df.info()
    categorical_cols = [col for col in df.columns if df[col].dtype == 'object']

    if categorical_cols:
        # Ensure encoding_methods dictionary is defined, or define it based on ide
        if 'encoding_methods' not in locals():
            encoding_methods = {col: 'One-Hot Encoding' for col in categorical_col

        # Create a OneHotEncoder instance
        encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

        # Fit and transform the selected categorical columns
        encoded_data = encoder.fit_transform(df[categorical_cols])

        # Create a new DataFrame from the encoded data
        encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_

        # Concatenate the new encoded DataFrame with the original DataFrame, dropp
        df = pd.concat([df.drop(columns=categorical_cols), encoded_df], axis=1)

        print("DataFrame after one-hot encoding:")
        display(df.head())
        print("\nColumn information after encoding:")
        display(df.info())
    else:
        print("No object type columns to encode.")
        print("DataFrame is ready for further analysis/modeling.")
        # Display the first few rows of the DataFrame
        print("DataFrame head:")
        display(df.head())
        print("\nColumn information:")
        display(df.info())
```

```
    else:
        print("DataFrame not loaded.")
```

```
Missing values before handling:
```

	0
age	0
sex	0
cp	0
trtbps	0
chol	0

```python
# Display column information to verify the new columns and data types
print("\nColumn information after encoding:")
display(df.info())
```

thalachh	0
exng	0
oldpeak	0
slp	0
caa	0
thall	0
output	0

dtype: int64

```
No object type columns to encode.
DataFrame is ready for further analysis/modeling.
DataFrame head:
```

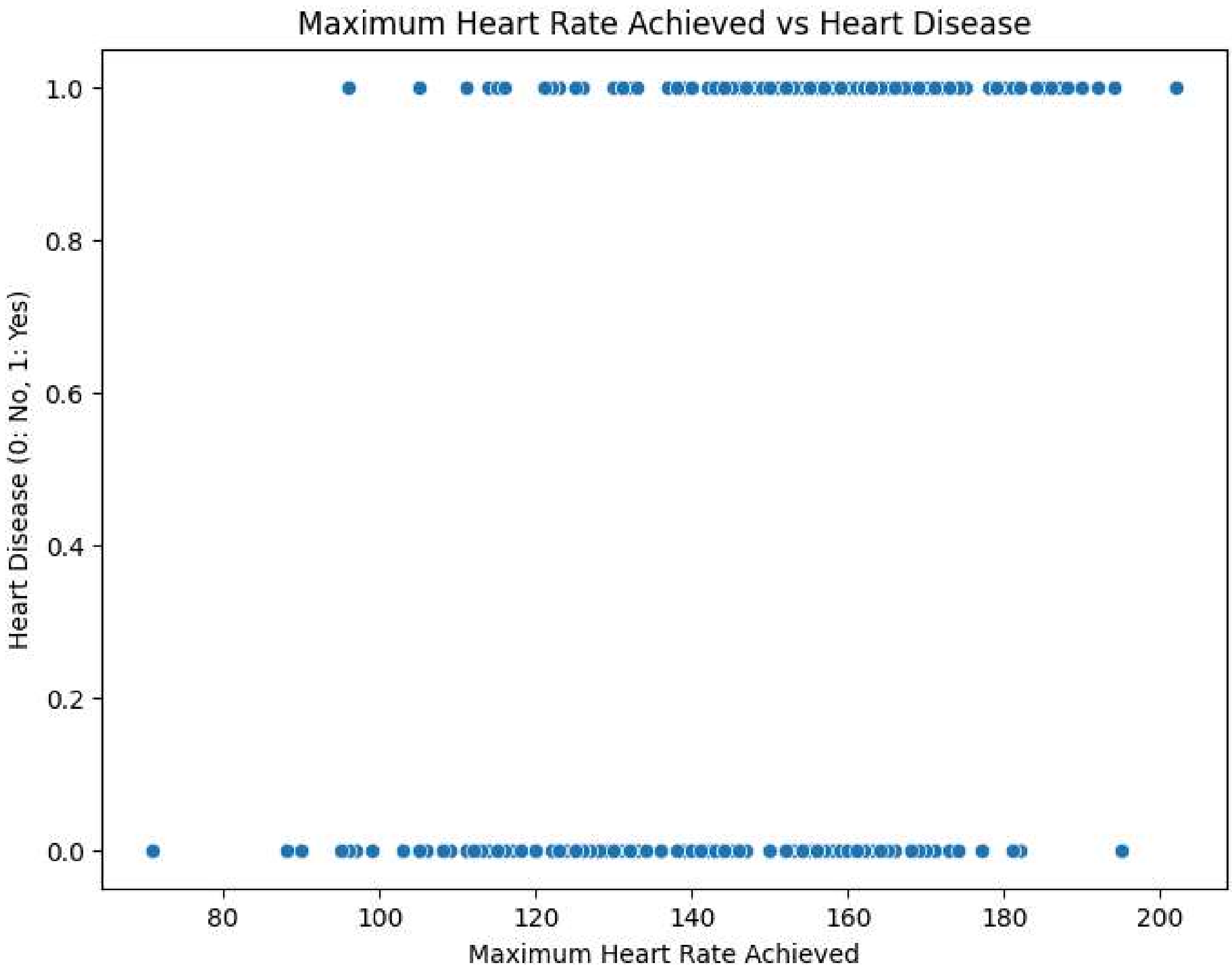
	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	tha
0	63	1	3	145	233	1	0	150	0	2.3	0	0	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	

```
Column information after encoding:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trtbps    303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalachh  303 non-null    int64
 8   exng      303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64
 11  caa       303 non-null    int64
 12  thall     303 non-null    int64
 13  output    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
None
```

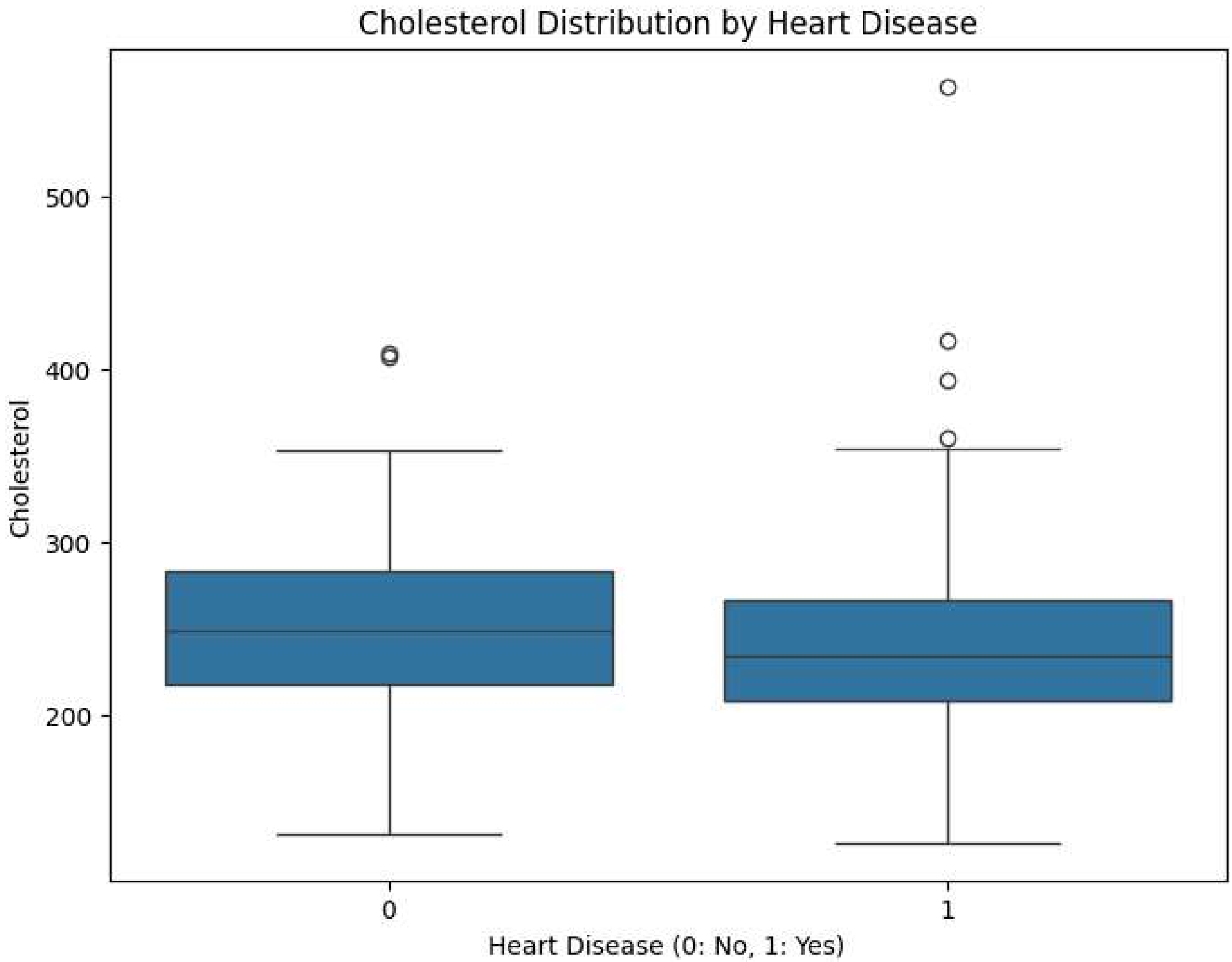
```python
plt.figure(figsize=(8, 6))
sns.scatterplot(x='thalachh', y='output', data=df)
plt.title('Maximum Heart Rate Achieved vs Heart Disease')
plt.xlabel('Maximum Heart Rate Achieved')
plt.ylabel('Heart Disease (0: No, 1: Yes)')
plt.show()
```

```
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trtbps    303 non-null    int64
 4   chol      303 non-null    int64
```

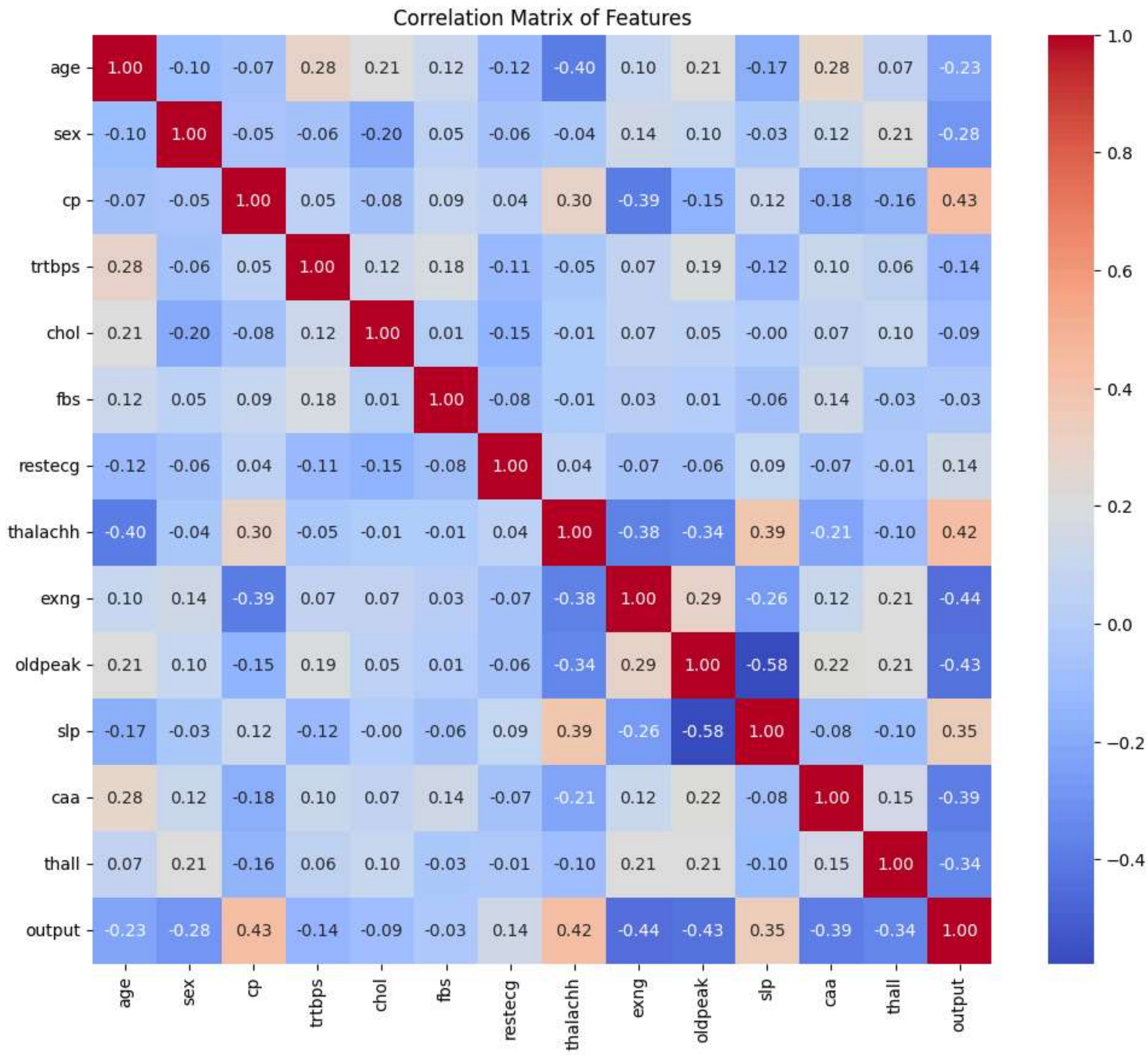
```
5   fbs        303 non-null    int64
```




```
plt.figure(figsize=(8, 6))
sns.boxplot(x='output', y='chol', data=df)
plt.title('Cholesterol Distribution by Heart Disease')
plt.xlabel('Heart Disease (0: No, 1: Yes)')
plt.ylabel('Cholesterol')
plt.show()
```




```
plt.figure(figsize=(12, 10))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Features')
plt.show()
```




```
from sklearn.model_selection import train_test_split

X = df.drop('output', axis=1)
```

```
y = df['output']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st

print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (242, 13)
Shape of X_test: (61, 13)
Shape of y_train: (242,)
Shape of y_test: (61,)
```

```
from sklearn.linear_model import LogisticRegression

# Instantiate the model
model = LogisticRegression(max_iter=1000)

# Train the model
model.fit(X_train, y_train)

print("Model training completed.")
```

```
Model training completed.
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_scor

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

```
# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the evaluation metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-score: {f1:.4f}")
```

```
Accuracy: 0.8033
Precision: 0.7692
Recall: 0.9091
F1-score: 0.8333
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_scor

# Instantiate the RandomForestClassifier model
rf_model = RandomForestClassifier(random_state=42)

# Train the new model
rf_model.fit(X_train, y_train)

# Make predictions with the new model
y_pred_rf = rf_model.predict(X_test)

# Evaluate the new model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)

# Print the evaluation metrics for the new model
print("Random Forest Classifier Performance:")
print(f"Accuracy: {accuracy_rf:.4f}")
print(f"Precision: {precision_rf:.4f}")
print(f"Recall: {recall_rf:.4f}")
print(f"F1-score: {f1_rf:.4f}")

# Compare with Logistic Regression
print("\nComparison with Logistic Regression:")
print(f"Logistic Regression Accuracy: {accuracy:.4f}")
print(f"Random Forest Accuracy: {accuracy_rf:.4f}")
print(f"Logistic Regression Precision: {precision:.4f}")
print(f"Random Forest Precision: {precision_rf:.4f}")
print(f"Logistic Regression Recall: {recall:.4f}")
print(f"Random Forest Recall: {recall_rf:.4f}")
print(f"Logistic Regression F1-score: {f1:.4f}")
print(f"Random Forest F1-score: {f1_rf:.4f}")
```

```
Random Forest Classifier Performance:
Accuracy: 0.8361
Precision: 0.7805
Recall: 0.9697
F1-score: 0.8649

Comparison with Logistic Regression:
Logistic Regression Accuracy: 0.8033
Random Forest Accuracy: 0.8361
Logistic Regression Precision: 0.7692
Random Forest Precision: 0.7805
Logistic Regression Recall: 0.9091
Random Forest Recall: 0.9697
Logistic Regression F1-score: 0.8333
Random Forest F1-score: 0.8649
```

```
from sklearn.preprocessing import StandardScaler

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform both training and testing data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("Data after standard scaling:")
print("Shape of X_train_scaled:", X_train_scaled.shape)
print("Shape of X_test_scaled:", X_test_scaled.shape)
```

```
Data after standard scaling:
Shape of X_train_scaled: (242, 13)
Shape of X_test_scaled: (61, 13)
```

```
# Instantiate the Logistic Regression model
scaled_lr_model = LogisticRegression(max_iter=1000)

# Train the model on the scaled training data
scaled_lr_model.fit(X_train_scaled, y_train)

print("Logistic Regression model trained on scaled data.")
```

```
Logistic Regression model trained on scaled data.
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_scor

# Make predictions on the scaled test set
y_pred_scaled_lr = scaled_lr_model.predict(X_test_scaled)

# Calculate evaluation metrics for the scaled Logistic Regression model
accuracy_scaled_lr = accuracy_score(y_test, y_pred_scaled_lr)
precision_scaled_lr = precision_score(y_test, y_pred_scaled_lr)
recall_scaled_lr = recall_score(y_test, y_pred_scaled_lr)
f1_scaled_lr = f1_score(y_test, y_pred_scaled_lr)

# Print the evaluation metrics
print("Logistic Regression Model Performance on Scaled Data:")
print(f"Accuracy: {accuracy_scaled_lr:.4f}")
print(f"Precision: {precision_scaled_lr:.4f}")
print(f"Recall: {recall_scaled_lr:.4f}")
print(f"F1-score: {f1_scaled_lr:.4f}")
```

```
Logistic Regression Model Performance on Scaled Data:
Accuracy: 0.8033
Precision: 0.7692
Recall: 0.9091
F1-score: 0.8333
```