

Dataset Name: heart_disease_uci

Dataset Link:

<https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data>

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

```
In [ ]: df = pd.read_csv('heart_disease_uci.csv')
df.head()
```

```
In [3]: df.dtypes
```

```
Out[3]: id          int64
        age          int64
        sex          object
        dataset      object
        cp           object
        trestbps      float64
        chol          float64
        fbs           object
        restecg       object
        thalch        float64
        exang         object
        oldpeak       float64
        slope         object
        ca            float64
        thal          object
        num           int64
        dtype: object
```

```
In [4]: df.columns
```

```
Out[4]: Index(['id', 'age', 'sex', 'dataset', 'cp', 'trestbps', 'chol', 'fbs',
              'restecg', 'thalch', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num'],
              dtype='object')
```

```
In [5]: # Check for missing values
        df.isnull().sum()
```

```
Out[5]: id          0
        age         0
        sex         0
        dataset     0
        cp          0
        trestbps    59
        chol        30
        fbs         90
        restecg     2
        thalch      55
        exang       55
        oldpeak     62
        slope       309
        ca          611
        thal        486
        num         0
        dtype: int64
```

```
In [6]: len(df)
```

```
Out[6]: 920
```

```
In [7]: missing_percentage = df.isnull().mean() * 100
        threshold = 30
        columns_to_drop = missing_percentage[missing_percentage >
        threshold].index
        df = df.drop(columns=columns_to_drop)
        print(f"Columns dropped: {list(columns_to_drop)}")
        print(f"Remaining columns: {df.columns}")
```

```
Columns dropped: ['slope', 'ca', 'thal']
```

```
Remaining columns: Index(['id', 'age', 'sex', 'dataset', 'cp', 'trestbps', 'chol', 'fbs',
        'restecg', 'thalch', 'exang', 'oldpeak', 'num'],
        dtype='object')
```

```
In [8]: # Pre-processing on dataset
        from sklearn.preprocessing import LabelEncoder
        label_encoder = LabelEncoder()
        for column in df.select_dtypes(include=['object']).columns:
```

```
df[column] = label_encoder.fit_transform(df[column])
df.head()
```

Out[8]:

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	num
0	1	63	1	0	3	145.0	233.0	1	0	150.0	0	2.3	0
1	2	67	1	0	0	160.0	286.0	0	0	108.0	1	1.5	2
2	3	67	1	0	0	120.0	229.0	0	0	129.0	1	2.6	1
3	4	37	1	0	2	130.0	250.0	0	1	187.0	0	3.5	0
4	5	41	0	0	1	130.0	204.0	0	0	172.0	0	1.4	0

```
In [9]: for column in df.select_dtypes(include=['float64']).columns:
        mean_value = df[column].mean()
        df[column] = df[column].fillna(mean_value)
```

```
In [10]: for column in df.select_dtypes(include=['object']).columns:
        mode_value = df[column].mode()[0]
        df[column] = df[column].fillna(mode_value)
df.isnull().sum()
```

Out[10]:

```
id          0
age         0
sex         0
dataset     0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalch      0
exang       0
oldpeak     0
num         0
dtype: int64
```

```
In [11]: # Handle Outliers on 'chol' (Cholesterol)
Q1 = df['chol'].quantile(0.25)
Q3 = df['chol'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Capping (Winsorization)
df['chol'] = np.where(df['chol'] > upper_bound, upper_bound, df['chol'])
df['chol'] = np.where(df['chol'] < lower_bound, lower_bound, df['chol'])

print(f"\n--- Outlier Handling ---")
print(f"Cholesterol outliers capped: Lower={lower_bound:.2f}, Upper={upper_bound:.2f}")
```

--- Outlier Handling ---

Cholesterol outliers capped: Lower=43.88, Upper=400.88

```
In [12]: data = df.copy()
def detect_outliers_iqr(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    return (column < lower_bound) | (column > upper_bound)

for col in data.columns:
    data[f'{col}_Outlier'] = detect_outliers_iqr(data[col])

outliers_only = data[data.filter(like='_Outlier').any(axis=1)]

outliers_only
```

Out[12]:

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	...	dataset_Outlier	cp_Outlier	trestbps_Outlier	chol_Outlier
0	1	63	1	0	3	145.000000	233.0	1	0	150.000000	...	False	False	False	False
1	2	67	1	0	0	160.000000	286.0	0	0	108.000000	...	False	False	False	False
2	3	67	1	0	0	120.000000	229.0	0	0	129.000000	...	False	False	False	False
4	5	41	0	0	1	130.000000	204.0	0	0	172.000000	...	False	False	False	False
6	7	62	0	0	0	140.000000	268.0	0	0	160.000000	...	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
915	916	54	0	3	0	127.000000	333.0	1	2	154.000000	...	False	False	False	False
916	917	62	1	3	3	132.132404	139.0	0	2	137.545665	...	False	False	False	False
917	918	55	1	3	0	122.000000	223.0	1	2	100.000000	...	False	False	False	False
918	919	58	1	3	0	132.132404	385.0	1	0	137.545665	...	False	False	False	False
919	920	62	1	3	1	120.000000	254.0	0	0	93.000000	...	False	False	False	False

605 rows × 26 columns



```

In [13]: data=df.copy()
def replace_outliers_with_nan(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    # Replace outliers with NaN
    return column.where((column >= lower_bound) & (column <=
upper_bound), np.nan)
# Replace outliers with NaN for each feature
for col in data.columns:
    data[col] = replace_outliers_with_nan(data[col])
null_counts = data.isnull().sum()

```

```
print("Sum of null values in each column:")  
print(null_counts)
```

Sum of null values in each column:

```
id          0  
age         0  
sex        194  
dataset     0  
cp          0  
trestbps   28  
chol        0  
fbs        228  
restecg    369  
thalch      2  
exang       0  
oldpeak    16  
num         0  
dtype: int64
```

```
In [14]: from sklearn.impute import KNNImputer  
imputer = KNNImputer(n_neighbors=2)  
data_imputed_knn = pd.DataFrame(imputer.fit_transform(data),  
                                columns=data.columns)  
data_imputed_knn.isnull().sum()
```

```
Out[14]: id          0  
age         0  
sex         0  
dataset     0  
cp          0  
trestbps    0  
chol        0  
fbs         0  
restecg     0  
thalch      0  
exang       0  
oldpeak     0  
num         0  
dtype: int64
```

```
In [15]: df=pd.read_csv("heart_disease_uci.csv")
```

```
In [16]: data = df.copy()
```

```
def remove_outliers_iqr_numerical(data_df):  
    # Select only the numerical columns for IQR calculation  
    numerical_data = data_df.select_dtypes(include=np.number)  
  
    Q1 = numerical_data.quantile(0.25)  
    Q3 = numerical_data.quantile(0.75)  
    IQR = Q3 - Q1  
    lower_bound = Q1 - 1.5 * IQR  
    upper_bound = Q3 + 1.5 * IQR  
  
    # Create a boolean mask: True if a value in any numerical column is an outlier  
    outlier_mask = ((numerical_data < lower_bound) | (numerical_data > upper_bound)).any(axis=1)  
  
    # Filter the entire original DataFrame using the mask  
    data_cleaned = data_df[~outlier_mask]  
  
    return data_cleaned  
  
cleaned_data = remove_outliers_iqr_numerical(data)  
  
print("Original Data:")  
print(len(data))  
print("\nCleaned Data (without outliers):")  
print(len(cleaned_data))
```

Original Data:

920

Cleaned Data (without outliers):

688

```
In [17]: data = df.copy()
```

```
def remove_outliers_zscore(data, threshold=3):  
  
    # Boolean DataFrame indicating if a value is an outlier
```

```

outliers = pd.DataFrame(False, index=data.index, columns=data.columns)

# Line 10: for column in data.columns:
for column in data.columns:
    # Line 11: This must be indented!
    if data[column].dtype in ['float64', 'int64']: # Only apply to numeric columns
        mean = data[column].mean()
        std_dev = data[column].std()
        z_scores = (data[column] - mean) / std_dev

        # Mark True in outliers DataFrame where abs(z_score) > threshold
        outliers[column] = abs(z_scores) > threshold

# Filter the rows that have any True in the outliers DataFrame (indicating an outlier)
rows_to_keep = ~outliers.any(axis=1)

# Return the DataFrame with outlier rows removed
return data[rows_to_keep]

# Example usage:
cleaned_data = remove_outliers_zscore(data, threshold=3)
print("Original Data:")
print(len(data))
print("\nCleaned Data (without outliers):")
print(len(cleaned_data))

```

Original Data:

920

Cleaned Data (without outliers):

903

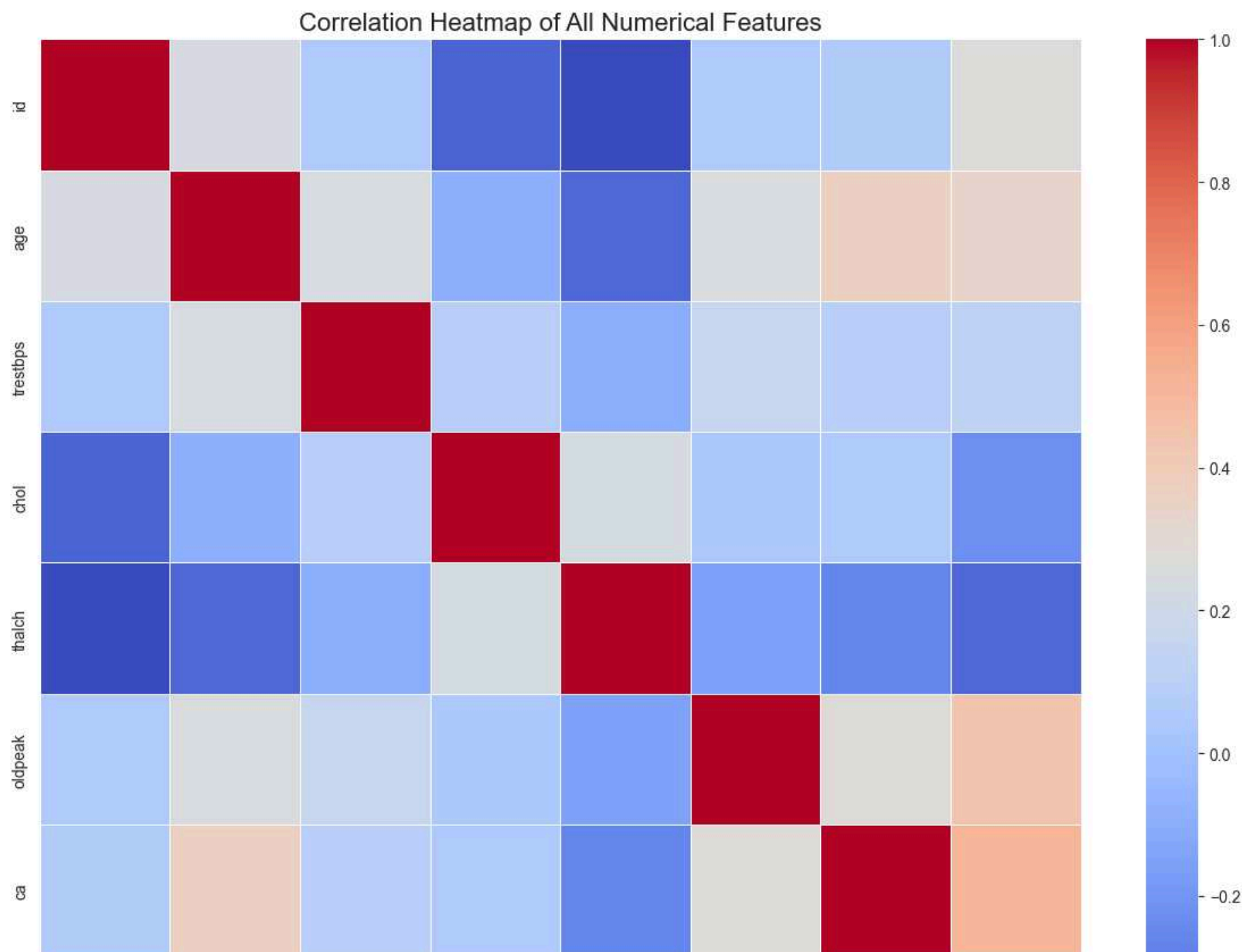
In [18]: # Visualization:

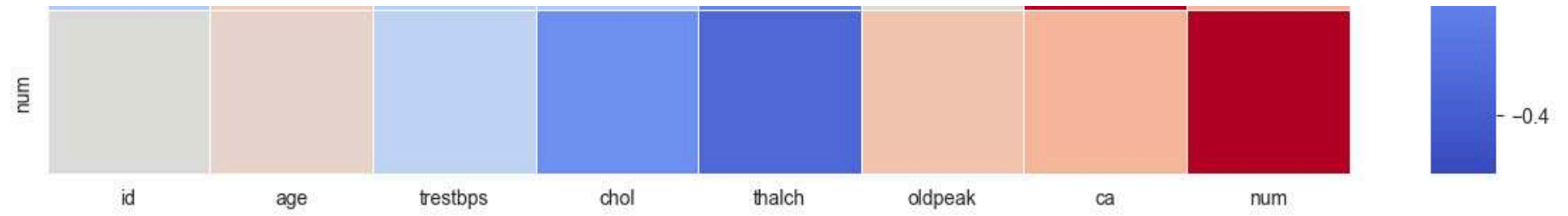
```

sns.set_style('whitegrid')
## 1. Correlation Heatmap
# -----
plt.figure(figsize=(12, 10))
numerical_df = df.select_dtypes(include=np.number)
correlation_matrix = numerical_df.corr()
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm', fmt=".2f", linewidths=.5)

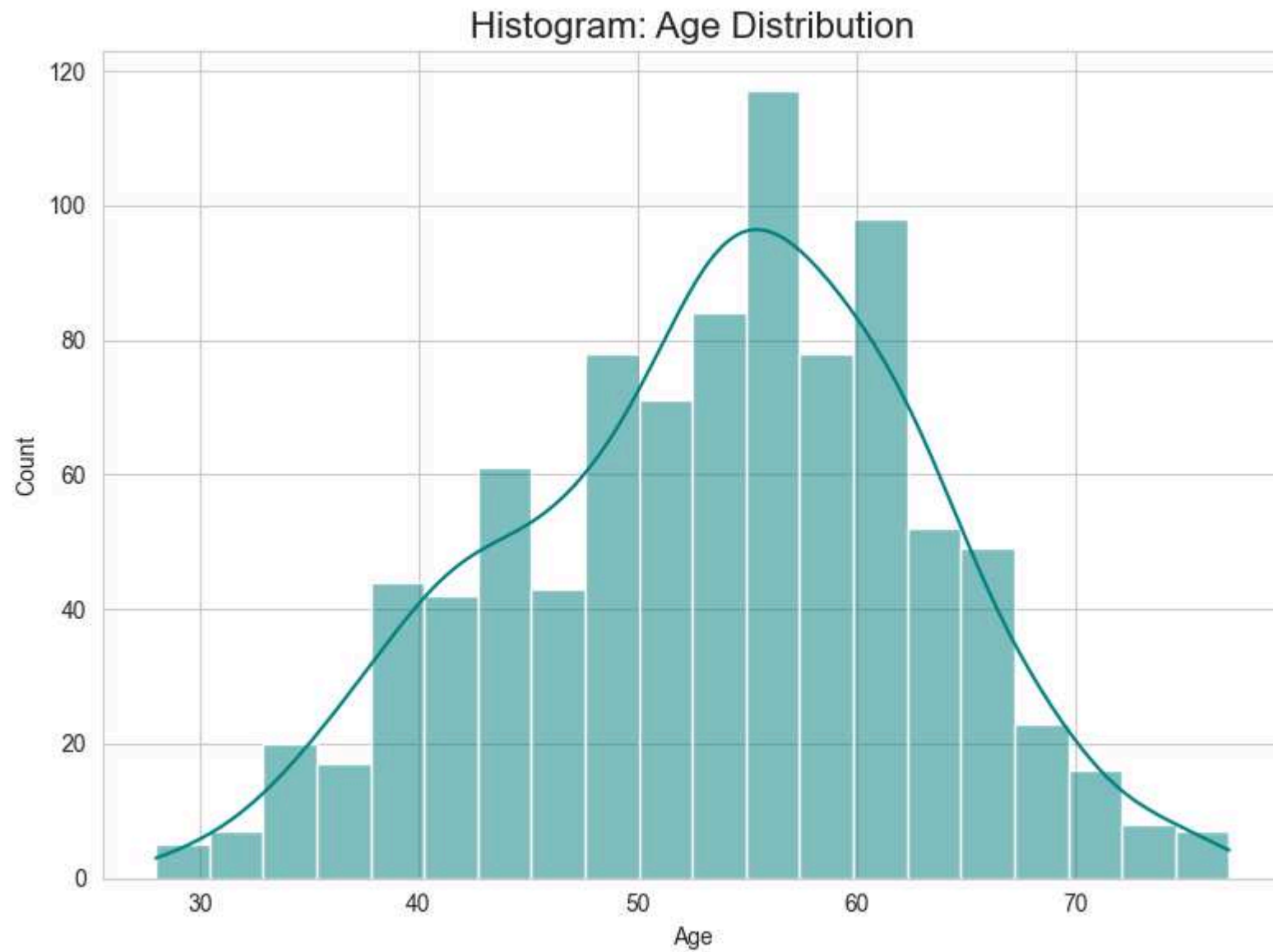
```

```
plt.title('Correlation Heatmap of All Numerical Features', fontsize=16)  
plt.tight_layout()  
plt.savefig('correlation_heatmap.png')  
plt.show()  
plt.close()
```





```
In [19]: plt.figure(figsize=(8, 6))
sns.histplot(df['age'], bins=20, kde=True, color='teal')
plt.title('Histogram: Age Distribution', fontsize=16)
plt.xlabel('Age')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```



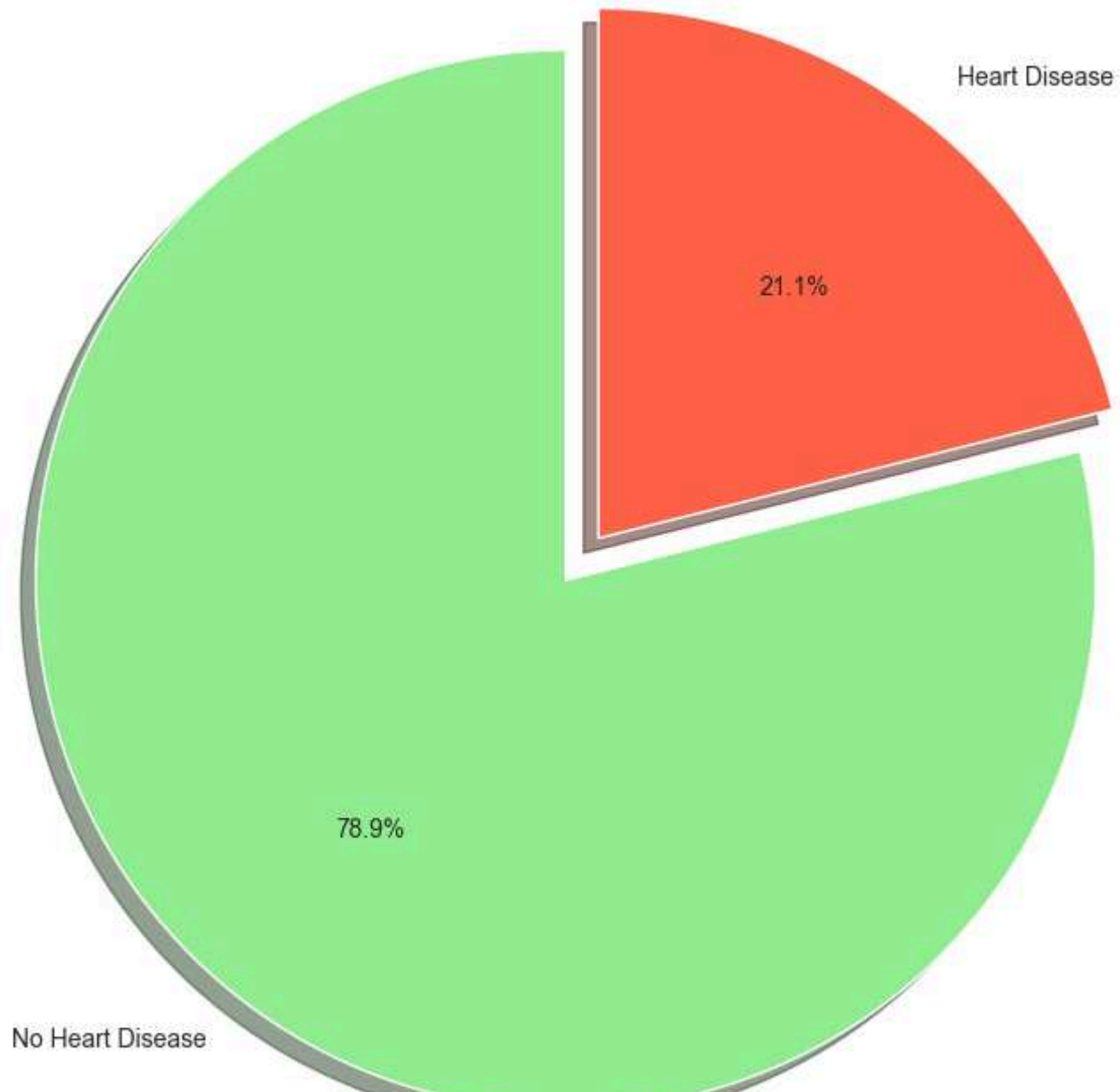
```
In [20]: TARGET_COL = 'sex'
```

```
target_counts = df[TARGET_COL].value_counts()  
labels = ['No Heart Disease', 'Heart Disease']  
explode = [0, 0.1]
```

```
colors = ['lightgreen', 'tomato']

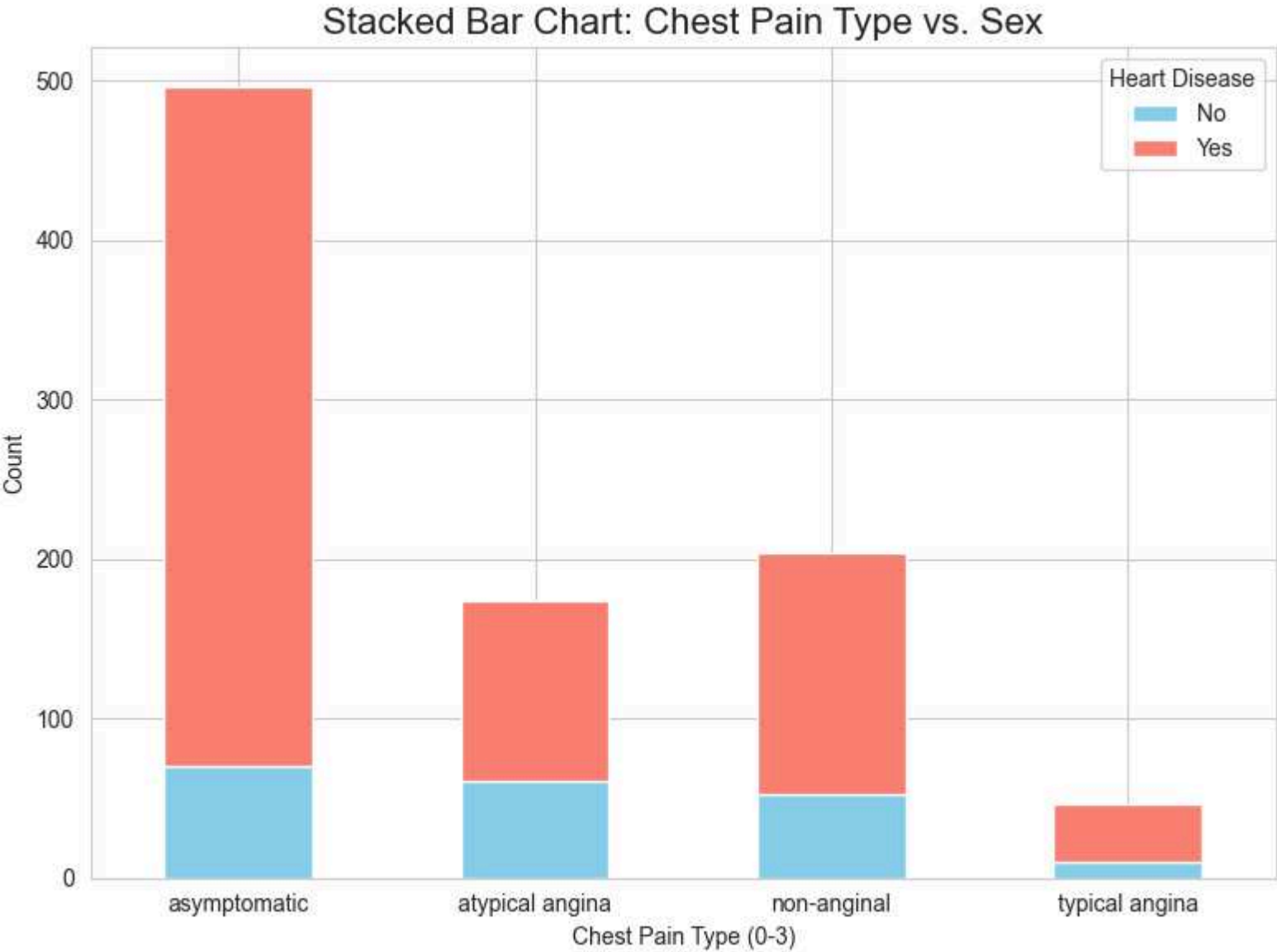
plt.figure(figsize=(8, 8))
plt.pie(target_counts, labels=labels, autopct='%1.1f%%', startangle=90,
        explode=explode, colors=colors, shadow=True)
plt.title(f'Exploded Pie Chart: {TARGET_COL.capitalize()} Distribution', fontsize=16)
plt.tight_layout()
plt.show()
```

Exploded Pie Chart: Sex Distribution





```
In [21]: TARGET_COL = 'sex'
cp_target = pd.crosstab(df['cp'], df[TARGET_COL])
cp_target.plot(kind='bar', stacked=True, figsize=(8, 6), color=['skyblue', 'salmon'])
plt.title(f'Stacked Bar Chart: Chest Pain Type vs. {TARGET_COL.capitalize()}', fontsize=16)
plt.xlabel('Chest Pain Type (0-3)')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.legend(title='Heart Disease', labels=['No', 'Yes'])
plt.tight_layout()
plt.show()
```



```
In [ ]:
```