Heart Disease Prediction Using Python

Author: Zahid Ali

Meta-Data (About Dataset)

Context

This is a multivariate type of dataset which means providing or involving a variety of separate mathematical or statistical variables, multivariate numerical data analysis. It is composed of 14 attributes which are age, sex, chest pain type, resting blood pressure, serum cholesterol, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercise-induced angina, oldpeak — ST depression induced by exercise relative to rest, the slope of the peak exercise ST segment, number of major vessels and Thalassemia. This database includes 76 attributes, but all published studies relate to the use of a subset of 14 of them. The Cleveland database is the only one used by ML researchers to date. One of the major tasks on this dataset is to predict based on the given attributes of a patient that whether that particular person has heart disease or not and other is the experimental task to diagnose and find out various insights from this dataset which could help in understanding the problem more.

Content

Column Descriptions:

- `id` (Unique id for each patient)
- `age` (Age of the patient in years)
- `dataset` (place of study)
- `sex` (Male/Female)
- `cp` Chest pain type:
  1. typical angina
  2. atypical angina
  3. non-anginal
  4. asymptomatic
- `trestbps` resting blood pressure (resting blood pressure (in mm Hg on admission to the hospital))
- `chol` (serum cholesterol in mg/dl)
- `fbs` (if fasting blood sugar > 120 mg/dl)
- `restecg` (resting electrocardiographic results)
- `Values`: [normal, stt abnormality, lv hypertrophy]
- `thalach`: maximum heart rate achieved
- `exang`: exercise-induced angina (True/ False)
- `oldpeak`: ST depression induced by exercise relative to rest
- `slope`: the slope of the peak exercise ST segment
- `ca`: number of major vessels (0-3) colored by fluoroscopy
- `thal`: [normal; fixed defect; reversible defect]

- `num` : the predicted attribute

Acknowledgements

Creators:

- Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
- University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
- University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
- V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

Relevant Papers:

- Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J., Sandhu, S., Guppy, K., Lee, S., & Froelicher, V. (1989). International application of a new probability algorithm for the diagnosis of coronary artery disease. American Journal of Cardiology, 64,304--310.
- David W. Aha & Dennis Kibler. "Instance-based prediction of heart-disease presence with the Cleveland database."
- Gennari, J.H., Langley, P, & Fisher, D. (1989). Models of incremental concept formation. Artificial Intelligence, 40, 11--61.

Citation Request:

The authors of the databases have requested that any publications resulting from the use of the data include the names of the principal investigator responsible for the data collection at each institution.

They would be:

- Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
- University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
- University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
- V.A. Medical Center, Long Beach and Cleveland Clinic Foundation:Robert Detrano, M.D., Ph.D.

Aims and Objective:

Aim:
To perform exploratory data analysis (EDA) and build a machine learning model that accurately predicts heart disease, helping in early detection and improving healthcare decisions.

Objectives:

- **Explore the Data:** Analyze the dataset to understand key patterns and relationships through EDA.
- **Feature Engineering:** Create or modify features to improve model accuracy.
- **Model Building:** Develop and compare various machine learning models.
- **Model Evaluation:** Measure model performance using accuracy, precision, recall, and F1-score.
- **Insights:** Identify the key factors contributing to heart disease and provide actionable insights.

Import Libraries

Let's start the project by impoprting all the libraries that we will need in this project.

In [185…

```
#To handle the data
import pandas as pd
import numpy as np

# to visualize the dataset
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# To preprocess the data
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.impute import SimpleImputer, KNNImputer
# import iterative imputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# machine learning
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
#for classification tasks
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBo
from xgboost import XGBClassifier
from sklearn.naive_bayes import GaussianNB
#pipeline
from sklearn.pipeline import Pipeline
#metrics
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

Load the Dataset

In [186…

```python
#load the dataset placed in our local pc
df = pd.read_csv('heart_disease_uci.csv')

#display the first 5 rows of the dataset
df.head()
```

Out[186…

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	ex
0	1	63	Male	Cleveland	typical angina	145.0	233.0	True	lv hypertrophy	150.0	F
1	2	67	Male	Cleveland	asymptomatic	160.0	286.0	False	lv hypertrophy	108.0	
2	3	67	Male	Cleveland	asymptomatic	120.0	229.0	False	lv hypertrophy	129.0	
3	4	37	Male	Cleveland	non-anginal	130.0	250.0	False	normal	187.0	F
4	5	41	Female	Cleveland	atypical angina	130.0	204.0	False	lv hypertrophy	172.0	F

Exploratory Data Analysis (EDA)

Explore the Dataset

In [187…

```python
#exploring data types of the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 920 entries, 0 to 919
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   id        920 non-null    int64  
 1   age       920 non-null    int64  
 2   sex       920 non-null    object 
 3   dataset   920 non-null    object 
 4   cp        920 non-null    object 
 5   trestbps  861 non-null    float64
 6   chol      890 non-null    float64
 7   fbs       830 non-null    object 
 8   restecg   918 non-null    object 
 9   thalch    865 non-null    float64
 10  exang     865 non-null    object 
 11  oldpeak   858 non-null    float64
 12  slope     611 non-null    object 
 13  ca        309 non-null    float64
 14  thal      434 non-null    object 
 15  num       920 non-null    int64  
dtypes: float64(5), int64(3), object(8)
memory usage: 115.1+ KB
```

In [188…
```
#data shape
df.shape
```
Out[188…
```
(920, 16)
```

ID Column

In [189…
```
#id column
df['id'].min(), df['id'].max()
```
Out[189…
```
(1, 920)
```

`id column is a unique identifier for each patient, it is not useful for our analysis.`

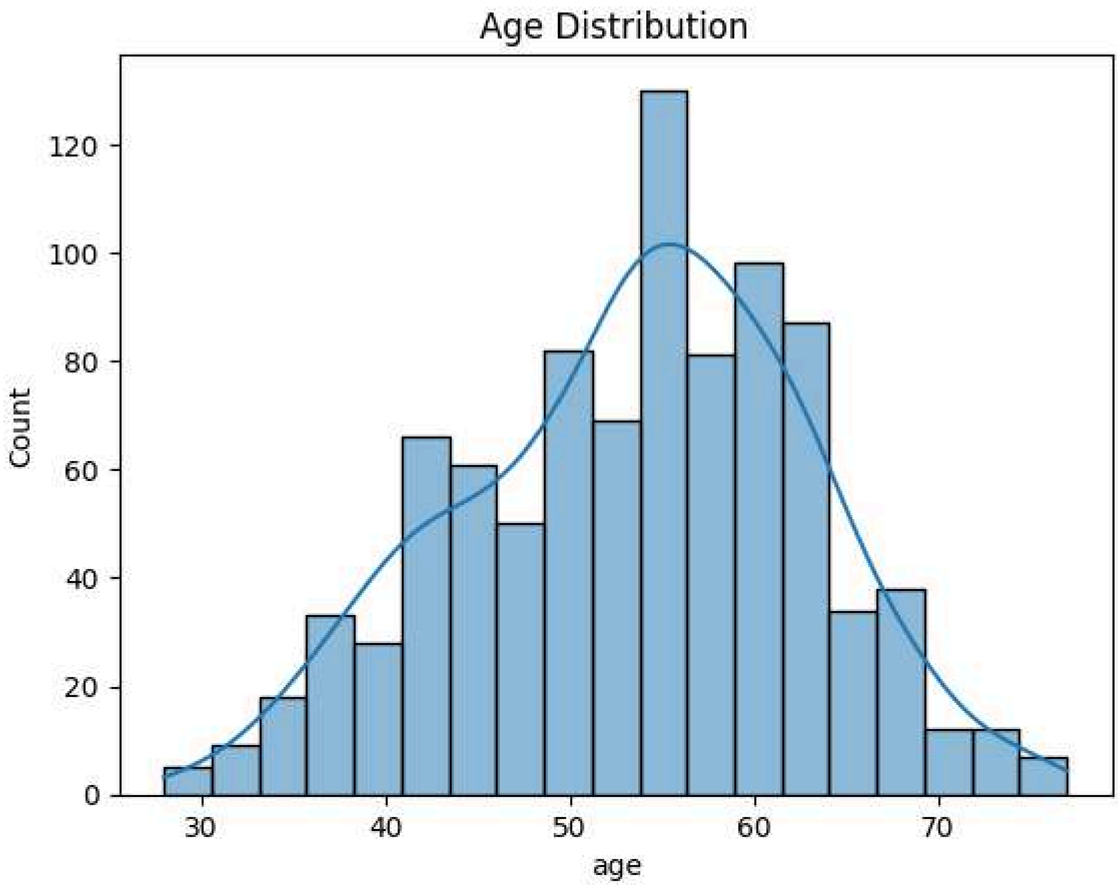
Age Column

In [190…
```
#age column
df['age'].min(), df['age'].max()
```
Out[190…
```
(28, 77)
```

`Observation: The minimum age of the patient is 28 years`

In [191…
```
# summarize age column
df['age'].describe()
```
Out[191…
```
count    920.000000
mean      53.510870
std        9.424685
min       28.000000
25%       47.000000
50%       54.000000
75%       60.000000
max       77.000000
Name: age, dtype: float64
```

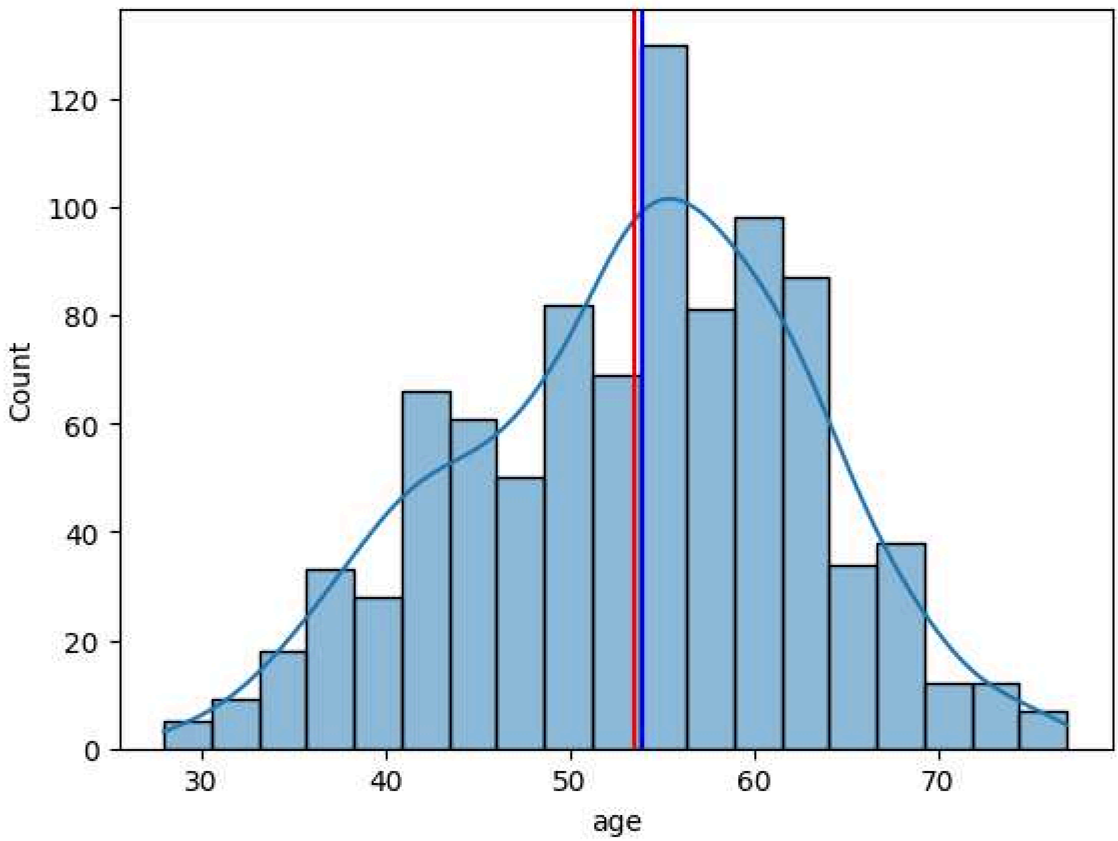
In [192…
```
#histogram to see the distribution of age
sns.histplot(df['age'], kde=True)
plt.title('Age Distribution')
```
Out[192…
```
Text(0.5, 1.0, 'Age Distribution')
```



In [193…

```
#mean, median, & mode of age column
sns.histplot(df['age'], kde=True)
plt.axvline(df['age'].mean(), color='red')
plt.axvline(df['age'].median(), color='green')
plt.axvline(df['age'].mode()[0], color='blue')

# print the value of mean, median and mode of age column
print('Mean:', df['age'].mean())
print('Median:', df['age'].median())
print('Mode:', df['age'].mode()[0])
```

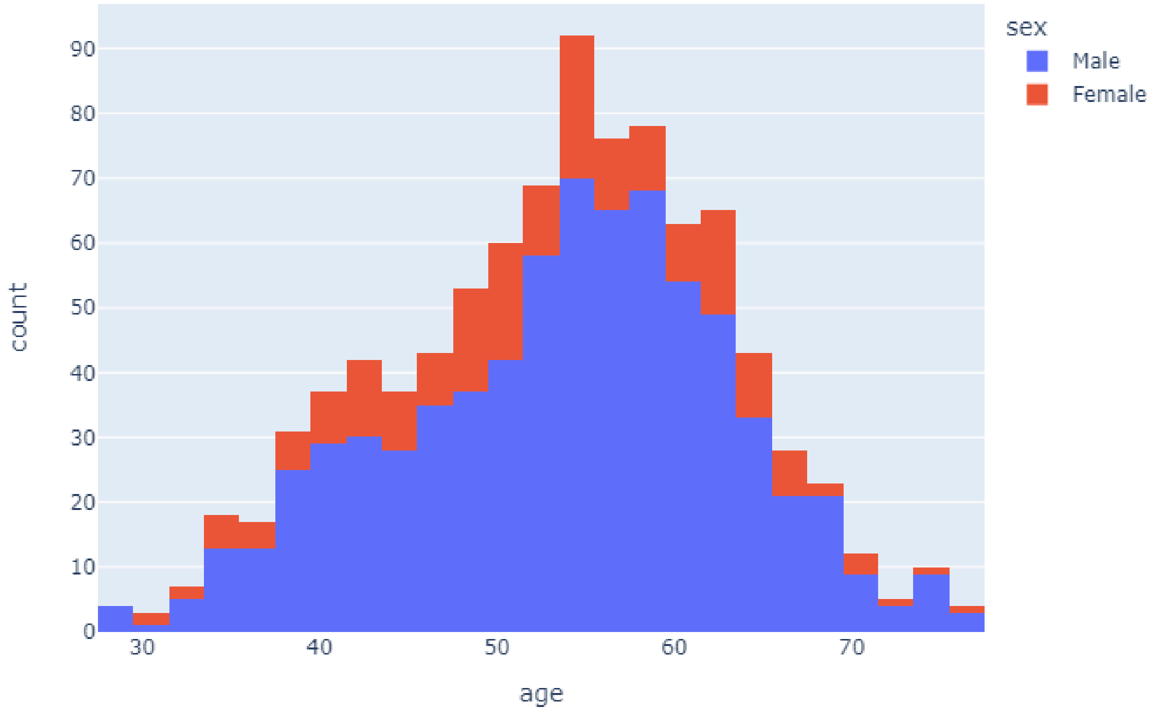
```
Mean: 53.51086956521739
Median: 54.0
Mode: 54
```



Explore Gender Distribution based on age

In [194…

```
#histogram to see the distribution of gender on age using plotly
fig = px.histogram(df, x='age', color='sex')
fig.show()
```



Sex Column

In [195…

```
#value counts of gender
df['sex'].value_counts()
```

Out[195…

```
sex
Male      726
Female    194
Name: count, dtype: int64
```

In [196…

```
#Male & female percenatge in our dataset
male_count = 726
female_count = 194
total_count = male_count + female_count

# calculate percentages
male_percentage = (male_count / total_count) * 100
female_percentage = (female_count / total_count) * 100

# display the results
print(f"Male percentage in the data: {male_percentage:.2f}%")
print(f"Female Percentage in the data: {female_percentage:.2f}%")

# difference
difference_percentage = ((male_count - female_count) / female_count) * 100
print(f"Males are {difference_percentage:.2f}% more than females in the data.")
```

```
Male percentage in the data: 78.91%
Female Percentage in the data: 21.09%
Males are 274.23% more than females in the data.
```

Dataset Column

In [197…

```
#dataset column
df['dataset'].unique()
```

Out[197…

```
array(['Cleveland', 'Hungary', 'Switzerland', 'VA Long Beach'],
      dtype=object)
```

In [198…

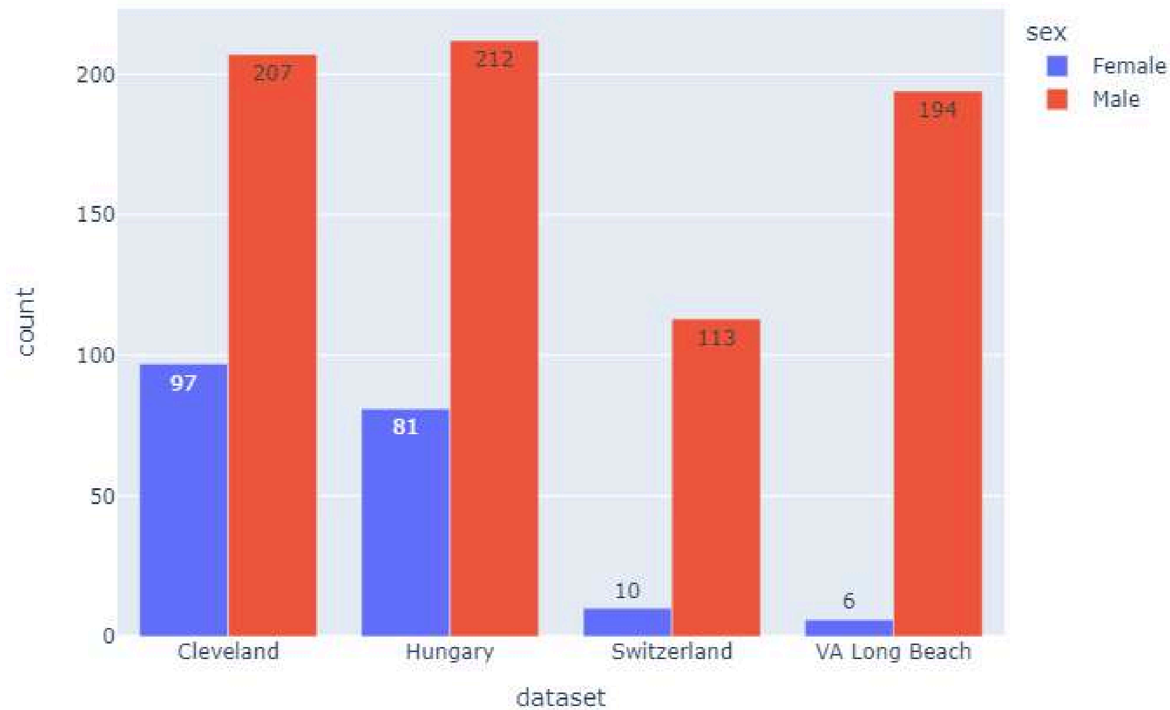
```
#value counts in dataset column
df['dataset'].value_counts()
```

Out[198…

```
dataset
Cleveland        304
Hungary          293
VA Long Beach    200
Switzerland      123
Name: count, dtype: int64
```

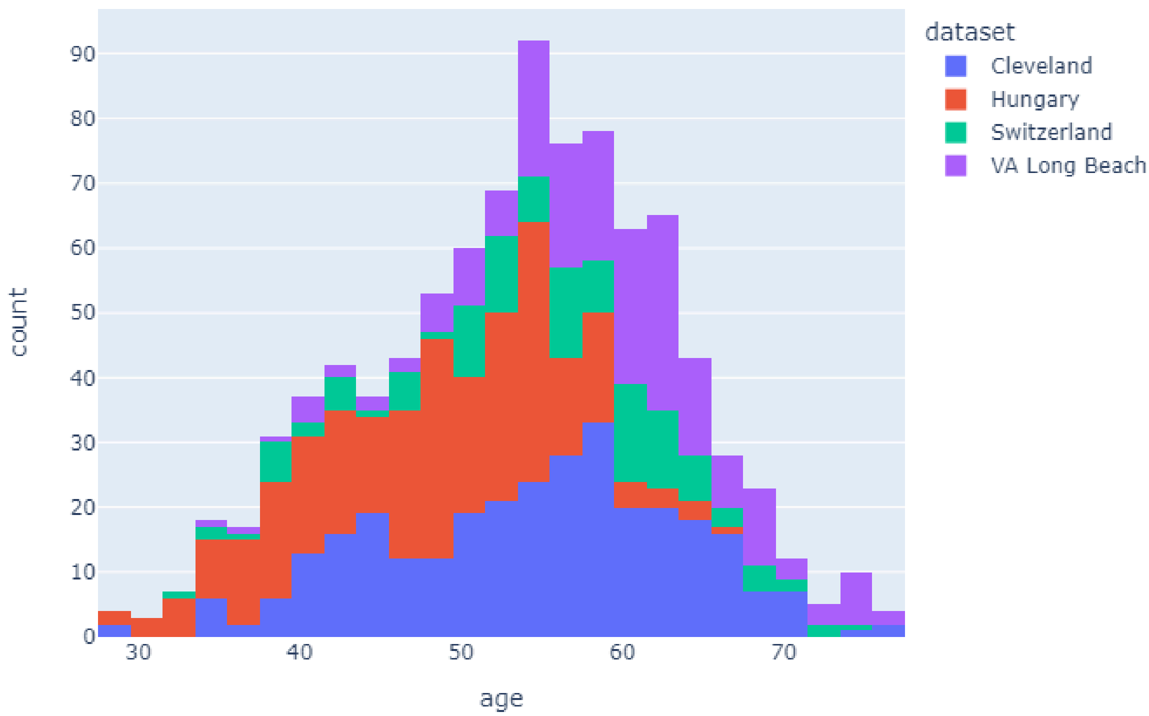
In [199…

```
# Create a bar chart with counts
fig = px.bar(df, x='dataset', color='sex', barmode='group')

# Add counts as text labels
df_counts = df.groupby(['dataset', 'sex']).size().reset_index(name='count')
fig = px.bar(df_counts, x='dataset', y='count', color='sex', barmode='group', text=


fig.show()
```



In [200…

```
#plot the distribution of age on dataset
fig = px.histogram(df, x='age', color='dataset')
fig.show()

#mean, median, & mode of age column on dataset column
print(f"Mean of age based on dataset: {df.groupby('dataset')['age'].mean()}")
print("--------------------------------------")
print(f"Median of age based on dataset: {df.groupby('dataset')['age'].median()}")
print("--------------------------------------")
print(f"Mode of age based on dataset: {df.groupby('dataset')['age'].agg(pd.Series.m
print("--------------------------------------")
```




```
Mean of age based on dataset: dataset
Cleveland        54.351974
Hungary          47.894198
Switzerland      55.317073
VA Long Beach    59.350000
Name: age, dtype: float64
-------------------------------------
Median of age based on dataset: dataset
Cleveland        55.5
Hungary          49.0
Switzerland      56.0
VA Long Beach    60.0
Name: age, dtype: float64
-------------------------------------
Mode of age based on dataset: dataset
Cleveland              58
Hungary                54
Switzerland            61
VA Long Beach    [62, 63]
Name: age, dtype: object
-------------------------------------
```

CP (Chest Pain) Column

In [201…

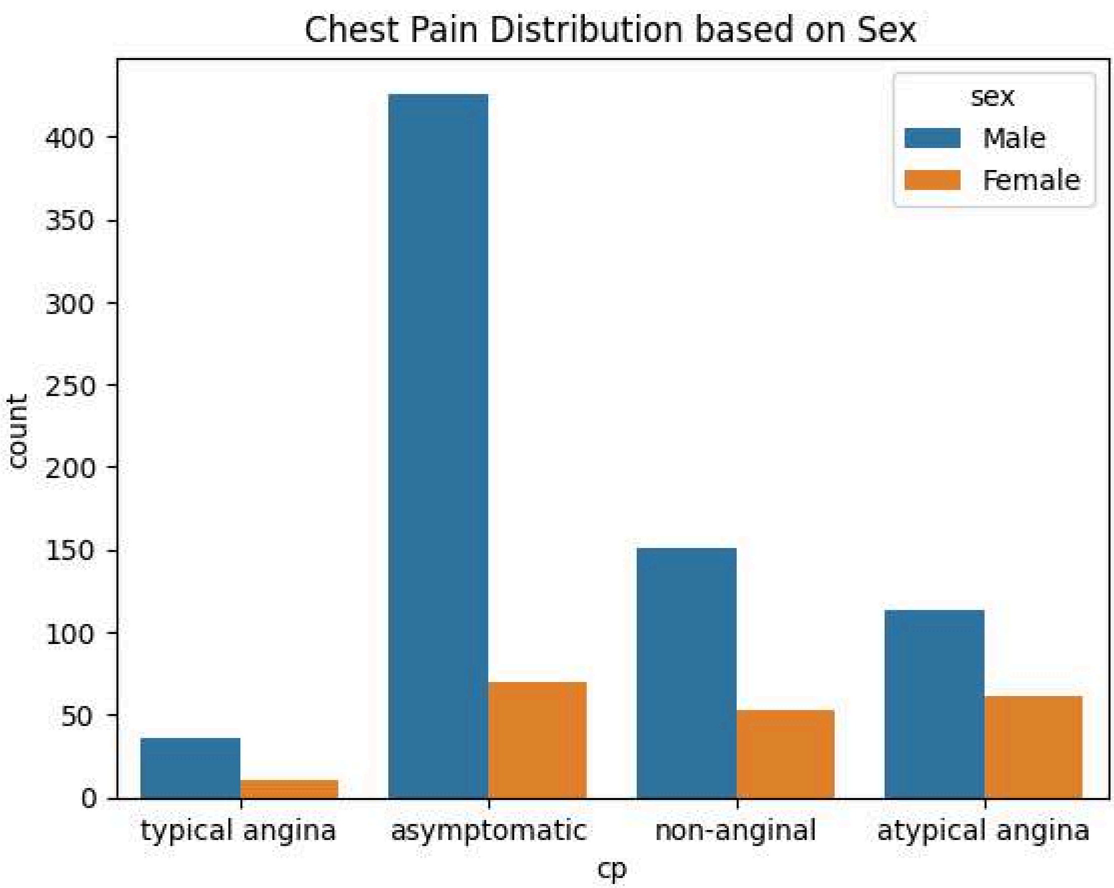
```
#value counts of chest pain column
df['cp'].value_counts()
```

Out[201…

```
cp
asymptomatic       496
non-anginal        204
atypical angina    174
typical angina      46
Name: count, dtype: int64
```

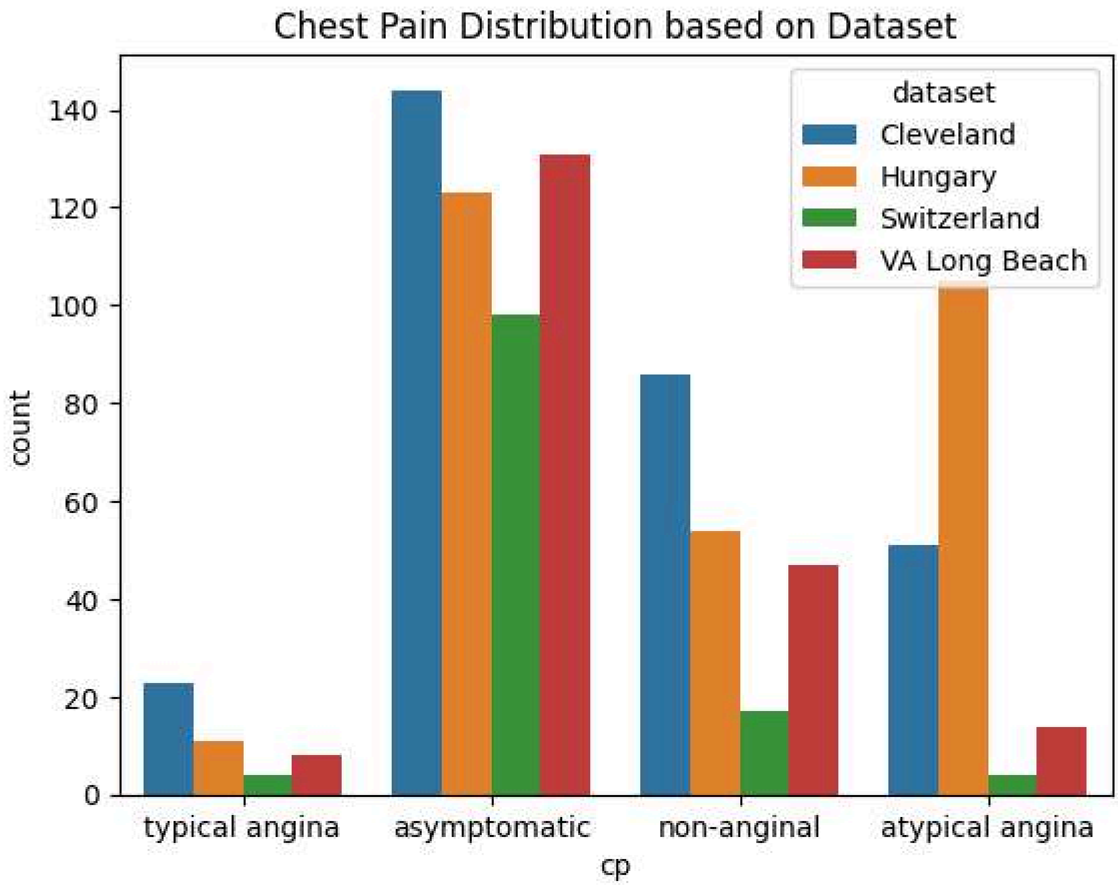
In [202…

```
#plot the cp column using sns
sns.countplot(data=df, x='cp', hue='sex')
plt.title('Chest Pain Distribution based on Sex')
plt.show()
```



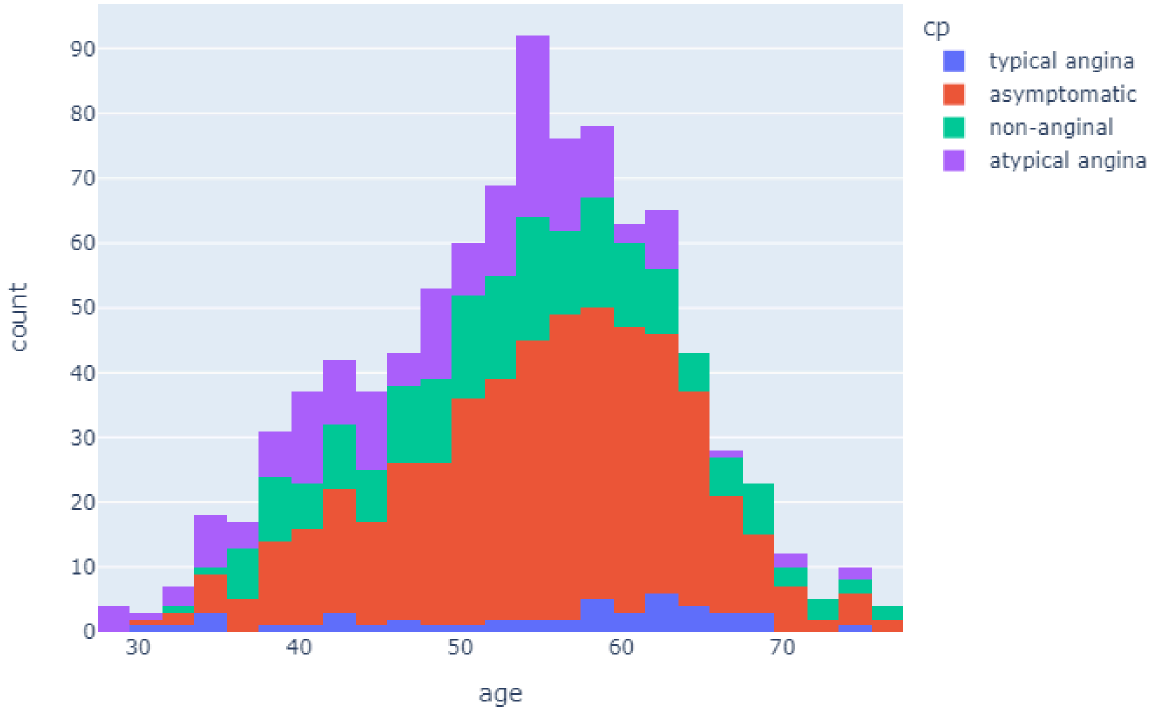
In [203…

```
#plot the cp based on dataset column
sns.countplot(data=df, x='cp', hue='dataset')
plt.title('Chest Pain Distribution based on Dataset')
plt.show()
```



In [204…

```
#plot the cp based on age column using plotly
fig = px.histogram(df, x='age', color='cp')
fig.show()
```



The remaining columns have missing values, we will fill them in the next step.

Dealing with missing values

In [205…

```
df.isnull().sum()[df.isnull().sum() > 0].sort_values(ascending=False)
```

Out[205…

```
ca          611
thal        486
slope       309
fbs          90
oldpeak      62
trestbps     59
thalch       55
exang        55
chol         30
restecg       2
dtype: int64
```

In [206…

```
missing_data_cols = df.isnull().sum()[df.isnull().sum() > 0].index.tolist()
missing_data_cols
```

Out[206…
```
['trestbps',
 'chol',
 'fbs',
 'restecg',
 'thalch',
 'exang',
 'oldpeak',
 'slope',
 'ca',
 'thal']
```

In [207…
```
categorical_cols = ['thal', 'ca', 'slope', 'exang', 'restecg','fbs', 'cp', 'sex', '
bool_cols = ['fbs', 'exang']
numeric_cols = ['oldpeak', 'thalch', 'chol', 'trestbps', 'age']
```

In [208…
```
# define the function to impute the missing values

def impute_categorical_missing_data(passed_col):

    df_null = df[df[passed_col].isnull()]
    df_not_null = df[df[passed_col].notnull()]

    X = df_not_null.drop(passed_col, axis=1)
    y = df_not_null[passed_col]

    other_missing_cols = [col for col in missing_data_cols if col != passed_col]

    label_encoder = LabelEncoder()

    for col in X.columns:
        if X[col].dtype == 'object' or X[col].dtype == 'category':
            X[col] = label_encoder.fit_transform(X[col])

    if passed_col in bool_cols:
        y = label_encoder.fit_transform(y)

    iterative_imputer = IterativeImputer(estimator=RandomForestRegressor(random_sta

    for col in other_missing_cols:
        if X[col].isnull().sum() > 0:
            col_with_missing_values = X[col].values.reshape(-1, 1)
            imputed_values = iterative_imputer.fit_transform(col_with_missing_value
            X[col] = imputed_values[:, 0]
        else:
            pass

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random

    rf_classifier = RandomForestClassifier()

    rf_classifier.fit(X_train, y_train)

    y_pred = rf_classifier.predict(X_test)

    acc_score = accuracy_score(y_test, y_pred)
```

```
    print("The feature '"+ passed_col+ "' has been imputed with", round((acc_score

    X = df_null.drop(passed_col, axis=1)

    for col in X.columns:
        if X[col].dtype == 'object' or X[col].dtype == 'category':
            X[col] = label_encoder.fit_transform(X[col])

    for col in other_missing_cols:
        if X[col].isnull().sum() > 0:
            col_with_missing_values = X[col].values.reshape(-1, 1)
            imputed_values = iterative_imputer.fit_transform(col_with_missing_value
            X[col] = imputed_values[:, 0]
        else:
            pass

    if len(df_null) > 0:
        df_null[passed_col] = rf_classifier.predict(X)
        if passed_col in bool_cols:
            df_null[passed_col] = df_null[passed_col].map({0: False, 1: True})
        else:
            pass
    else:
        pass

    df_combined = pd.concat([df_not_null, df_null])

    return df_combined[passed_col]

def impute_continuous_missing_data(passed_col):

    df_null = df[df[passed_col].isnull()]
    df_not_null = df[df[passed_col].notnull()]

    X = df_not_null.drop(passed_col, axis=1)
    y = df_not_null[passed_col]

    other_missing_cols = [col for col in missing_data_cols if col != passed_col]

    label_encoder = LabelEncoder()

    for col in X.columns:
        if X[col].dtype == 'object' or X[col].dtype == 'category':
            X[col] = label_encoder.fit_transform(X[col])

    iterative_imputer = IterativeImputer(estimator=RandomForestRegressor(random_sta

    for col in other_missing_cols:
        if X[col].isnull().sum() > 0:
            col_with_missing_values = X[col].values.reshape(-1, 1)
            imputed_values = iterative_imputer.fit_transform(col_with_missing_value
            X[col] = imputed_values[:, 0]
        else:
            pass
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random

    rf_regressor = RandomForestRegressor()

    rf_regressor.fit(X_train, y_train)

    y_pred = rf_regressor.predict(X_test)

    print("MAE =", mean_absolute_error(y_test, y_pred), "\n")
    print("RMSE =", mean_squared_error(y_test, y_pred, squared=False), "\n")
    print("R2 =", r2_score(y_test, y_pred), "\n")

    X = df_null.drop(passed_col, axis=1)

    for col in X.columns:
        if X[col].dtype == 'object' or X[col].dtype == 'category':
            X[col] = label_encoder.fit_transform(X[col])

    for col in other_missing_cols:
        if X[col].isnull().sum() > 0:
            col_with_missing_values = X[col].values.reshape(-1, 1)
            imputed_values = iterative_imputer.fit_transform(col_with_missing_value
            X[col] = imputed_values[:, 0]
        else:
            pass

    if len(df_null) > 0:
        df_null[passed_col] = rf_regressor.predict(X)
    else:
        pass

    df_combined = pd.concat([df_not_null, df_null])

    return df_combined[passed_col]
```

In [209…

```
df.isnull().sum()[df.isnull().sum() > 0].sort_values(ascending=False)
```

Out[209…

```
ca          611
thal        486
slope       309
fbs          90
oldpeak      62
trestbps     59
thalch       55
exang        55
chol         30
restecg       2
dtype: int64
```

In [210…

```
#using our function to impute the missing values using for loop
for col in missing_data_cols:
    print("Missing Values", col, ":", str(round((df[col].isnull().sum() / len(df))
    if col in categorical_cols:
        df[col] = impute_categorical_missing_data(col)
    elif col in numeric_cols:
        df[col] = impute_continuous_missing_data(col)
```

```
    else:
        pass
```

```
Missing Values trestbps : 6.41%
MAE = 13.973088803088805

RMSE = 18.897657767881427

R2 = 0.07485079057252497

Missing Values chol : 3.26%
MAE = 48.210749063670406

RMSE = 66.64813668228823

R2 = 0.6489533549015398

Missing Values fbs : 9.78%
The feature 'fbs' has been imputed with 79.52 accuracy

Missing Values restecg : 0.22%
The feature 'restecg' has been imputed with 63.04 accuracy

Missing Values thalch : 5.98%
MAE = 16.792

RMSE = 21.45059027339162

R2 = 0.31794426708159296

Missing Values exang : 5.98%
The feature 'exang' has been imputed with 79.62 accuracy

Missing Values oldpeak : 6.74%
MAE = 0.5583527131782946

RMSE = 0.8045275733491486

R2 = 0.4413503132479568

Missing Values slope : 33.59%
The feature 'slope' has been imputed with 67.39 accuracy

Missing Values ca : 66.41%
The feature 'ca' has been imputed with 65.59 accuracy

Missing Values thal : 52.83%
The feature 'thal' has been imputed with 72.52 accuracy
```

In [211…

```
#check if there are any missing values
df.isnull().sum()
```

Out[211…

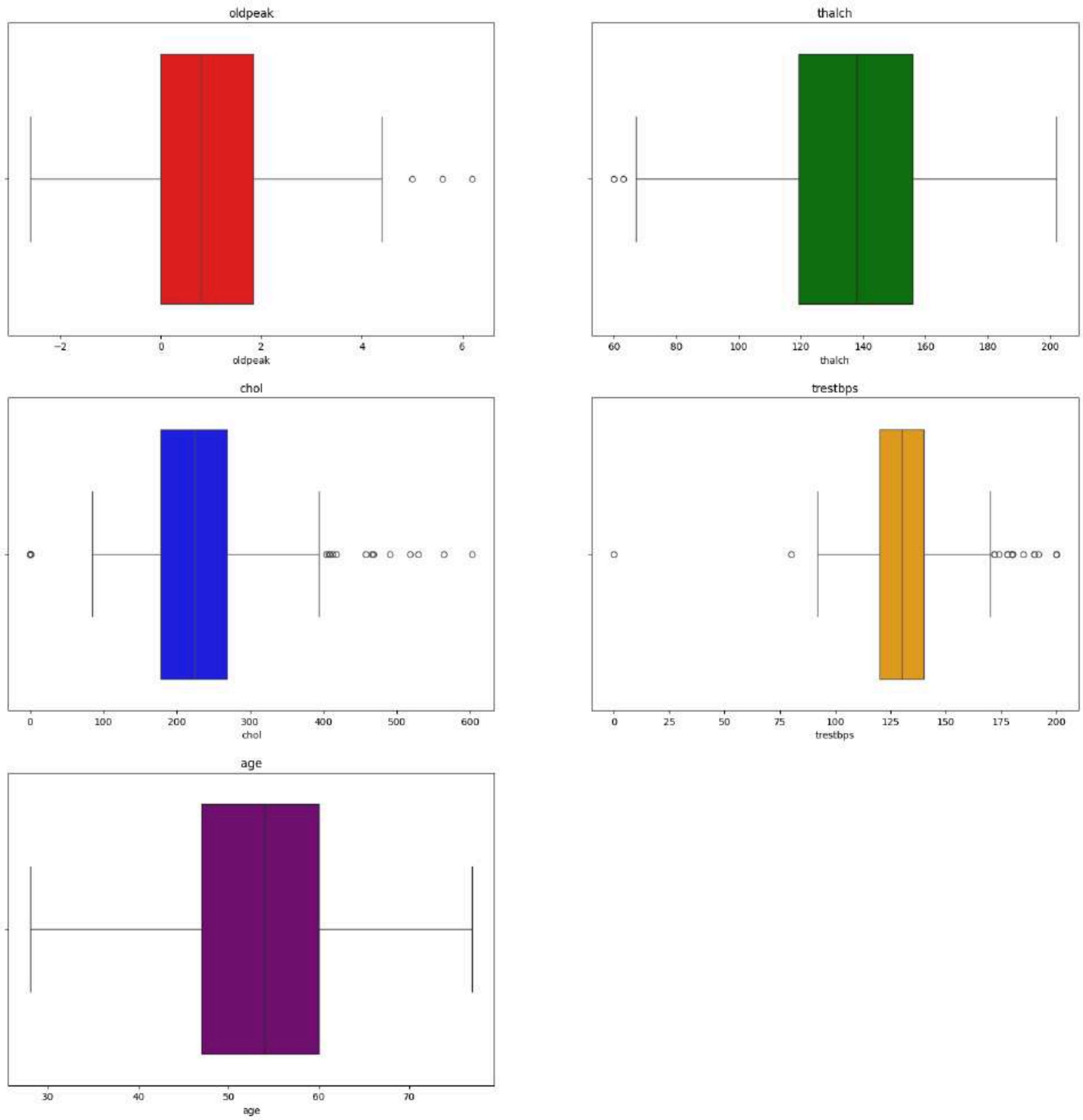
```
id          0
age         0
sex         0
dataset     0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalch      0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
num         0
dtype: int64
```

`Missing values are imputed.`

Dealing with Outliers

In [212…

```
#box plot of all numeric columns using for loop
plt.figure(figsize=(20, 20))

colors = ['red', 'green', 'blue', 'orange', 'purple']

for i, col in enumerate(numeric_cols):
    plt.subplot(3, 2, i+1)
    sns.boxplot(x=df[col], color=colors[i])
    plt.title(col)
plt.show()
```



In [213…

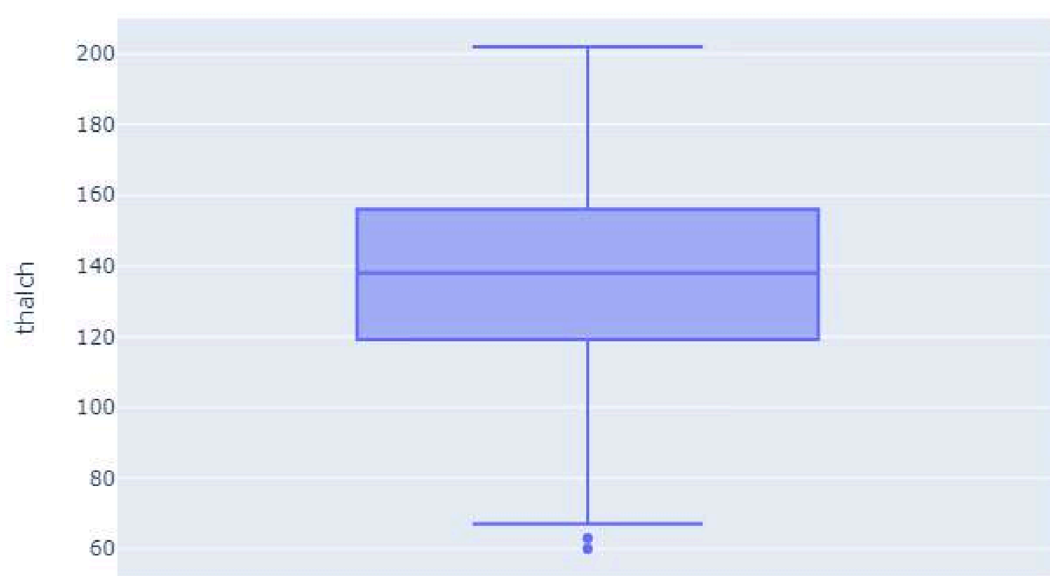
```
#plot box plot for all numeric columns using for loop in plotly
fig = px.box(df, y='age', title='Age Box Plot')
fig.show()

fig = px.box(df, y='trestbps', title='Trestbps Box Plot')
fig.show()

fig = px.box(df, y='chol', title='Chol Box Plot')
fig.show()

fig = px.box(df, y='thalch', title='Thalach Box Plot')
fig.show()

fig = px.box(df, y='oldpeak', title='Oldpeak Box Plot')
fig.show()
```

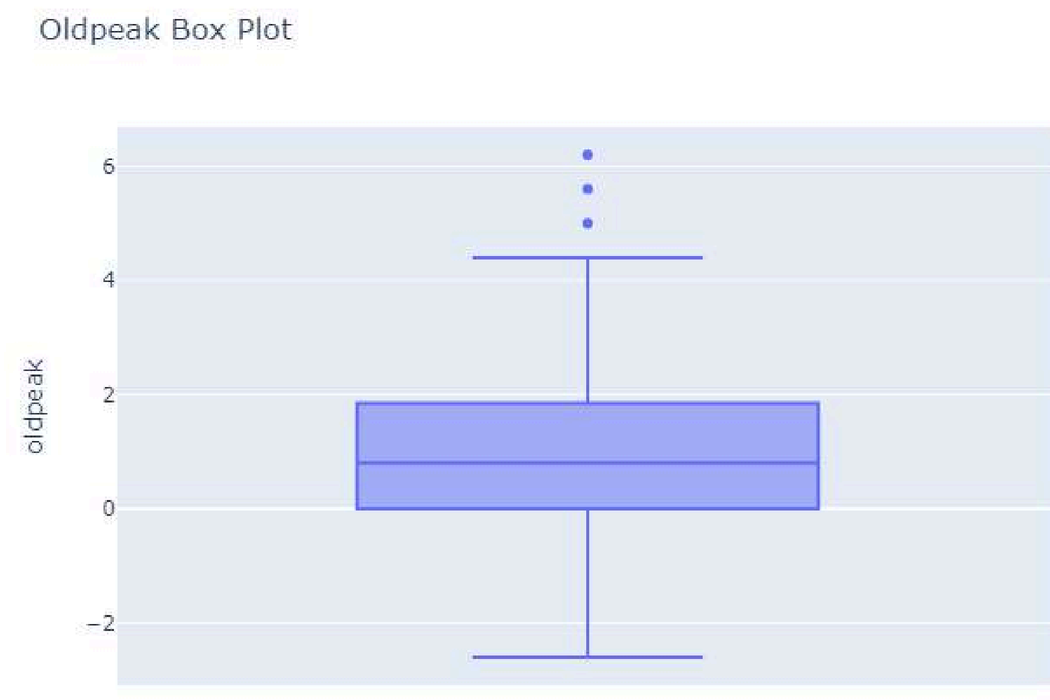
Age Box Plot

Trestbps Box Plot

Chol Box Plot

Thalach Box Plot






```
In [214…
# defining a function for outlier treatment using z-score
def outlier_treatment(df , col):

    # Calculate the Z-scores for each value in the column
    z_scores = np.abs((df[col] - df[col].mean()) / df[col].std())

    # Define the threshold for identifying outliers
    threshold = 3

    # identify rows where any column has a Z-score above the threshold
    outliers = (z_scores > threshold)

    # the number of rows identified as outliers
    print(f'Number of rows identified as outliers in {col}: {outliers.sum()}')

    # Remove the outliers
    df = df[~outliers]

    # print statement
    print('Z score has been successfully applied on {}.'.format(col))

    # returning the dataframe
    return df
```

```
In [215…
# aaplying outlier_treatment function on trestbps
df = outlier_treatment(df , 'trestbps')
```

```
Number of rows identified as outliers in trestbps: 8
Z score has been successfully applied on trestbps.
```

In [216…

```
# aaplying outlier_treatment function on chol
df = outlier_treatment(df , 'chol')
```

```
Number of rows identified as outliers in chol: 3
Z score has been successfully applied on chol.
```

In [217…

```
# # Dropping rows where 'trestbps' or 'chol' are 0, as these values are not medical
df = df[df['chol'] != 0]
```

In [218…

```
# check the row where trestbps is 0
df[df['trestbps']==0]
```

Out[218…

id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope

In [219…

```
# Remove the row where trestbps is not equal to zero
df=df[df['trestbps']!=0]
df.head()
```

Out[219…

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	ex
0	1	63	Male	Cleveland	typical angina	145.0	233.0	True	lv hypertrophy	150.0	F
1	2	67	Male	Cleveland	asymptomatic	160.0	286.0	False	lv hypertrophy	108.0	
2	3	67	Male	Cleveland	asymptomatic	120.0	229.0	False	lv hypertrophy	129.0	
3	4	37	Male	Cleveland	non-anginal	130.0	250.0	False	normal	187.0	F
4	5	41	Female	Cleveland	atypical angina	130.0	204.0	False	lv hypertrophy	172.0	F

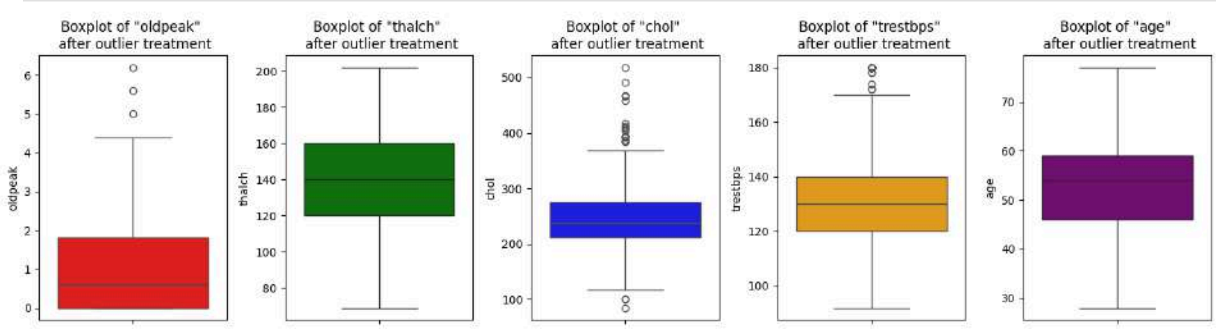
In [220…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 740 entries, 0 to 919
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   id        740 non-null    int64  
 1   age       740 non-null    int64  
 2   sex       740 non-null    object 
 3   dataset   740 non-null    object 
 4   cp        740 non-null    object 
 5   trestbps  740 non-null    float64
 6   chol      740 non-null    float64
 7   fbs       740 non-null    object 
 8   restecg   740 non-null    object 
 9   thalch    740 non-null    float64
 10  exang     740 non-null    object 
 11  oldpeak   740 non-null    float64
 12  slope     740 non-null    object 
 13  ca        740 non-null    float64
 14  thal      740 non-null    object 
 15  num       740 non-null    int64  
dtypes: float64(5), int64(3), object(8)
memory usage: 98.3+ KB
```

In [221…

```python
# setting up the figure size
plt.figure(figsize=(15, 4))
colors = ['red', 'green', 'blue', 'orange', 'purple']

# loop through each column
for i in range(len(numeric_cols)):
    # create a subplot
    plt.subplot(1, len(numeric_cols), i + 1)
    # plotting the boxplot
    sns.boxplot(y=df[numeric_cols[i]], color=colors[i])
    # adding title
    plt.title(f'Boxplot of "{numeric_cols[i]}" \n after outlier treatment')

plt.tight_layout()

plt.show()
```



In [222…

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 740 entries, 0 to 919
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   id        740 non-null    int64  
 1   age       740 non-null    int64  
 2   sex       740 non-null    object 
 3   dataset   740 non-null    object 
 4   cp        740 non-null    object 
 5   trestbps  740 non-null    float64
 6   chol      740 non-null    float64
 7   fbs       740 non-null    object 
 8   restecg   740 non-null    object 
 9   thalch    740 non-null    float64
 10  exang     740 non-null    object 
 11  oldpeak   740 non-null    float64
 12  slope     740 non-null    object 
 13  ca        740 non-null    float64
 14  thal      740 non-null    object 
 15  num       740 non-null    int64  
dtypes: float64(5), int64(3), object(8)
memory usage: 98.3+ KB
```

Let's Continue our EDA

Resting Blood Pressure (trestbps) Column

`The normal resting blood pressure is 120/80 mm Hg.`

1. High BP (Hypertension): Can lead to heart disease, stroke.
2. Low BP (Hypotension): May cause dizziness, fainting.

In [223…

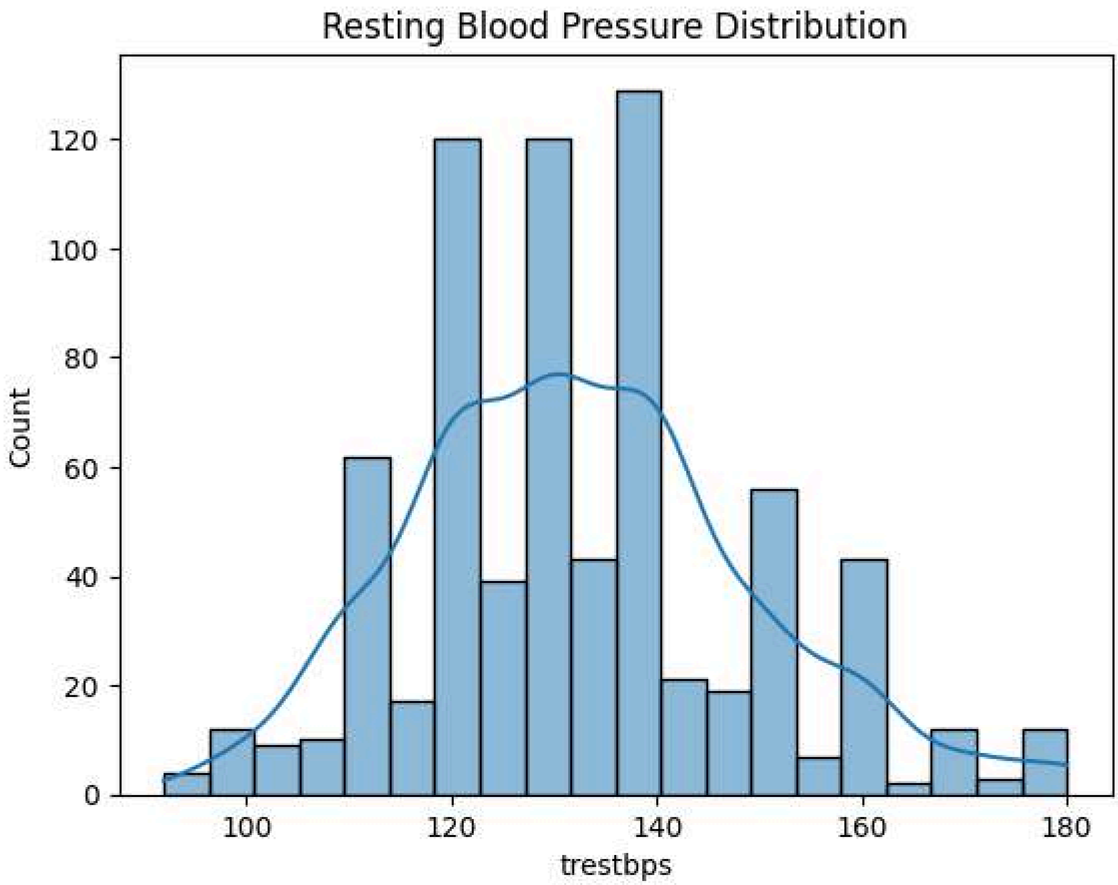
```
#summary statistics of trestbps column
df['trestbps'].describe()
```

Out[223…

```
count    740.000000
mean     132.686770
std       16.629545
min       92.000000
25%      120.000000
50%      130.000000
75%      140.000000
max      180.000000
Name: trestbps, dtype: float64
```

In [224…

```
#histogram of trestbps column
sns.histplot(df['trestbps'], kde=True)
plt.title('Resting Blood Pressure Distribution')
plt.show()
```



In [225…

```
df['trestbps'].value_counts().nlargest(5)
```

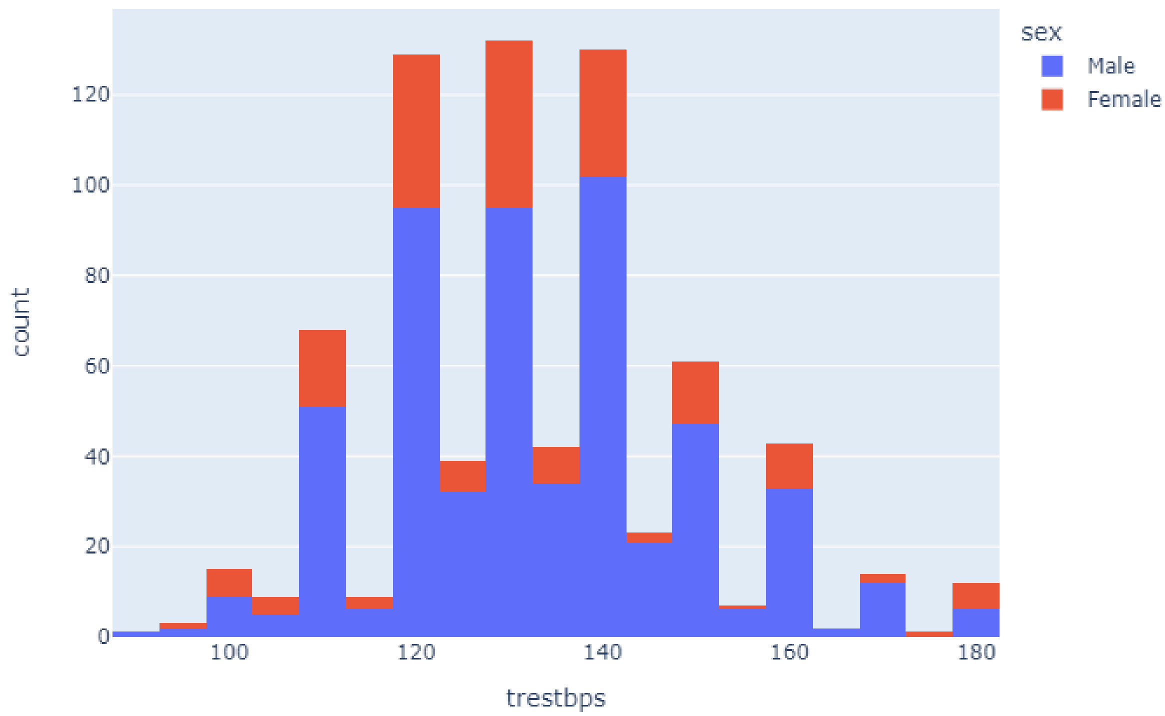
Out[225…

```
trestbps
120.0    110
130.0    100
140.0     90
150.0     49
110.0     47
Name: count, dtype: int64
```

Observation: Majority of the Patients have Resting Blood pressure ranges from 110-150 mm Hg.

In [226…

```
#Plot the distribution of trestbps based on gender
fig = px.histogram(df, x='trestbps', color='sex')
fig.show()
```



According to our dataset, Females have higher resting blood pressure as compared to males.

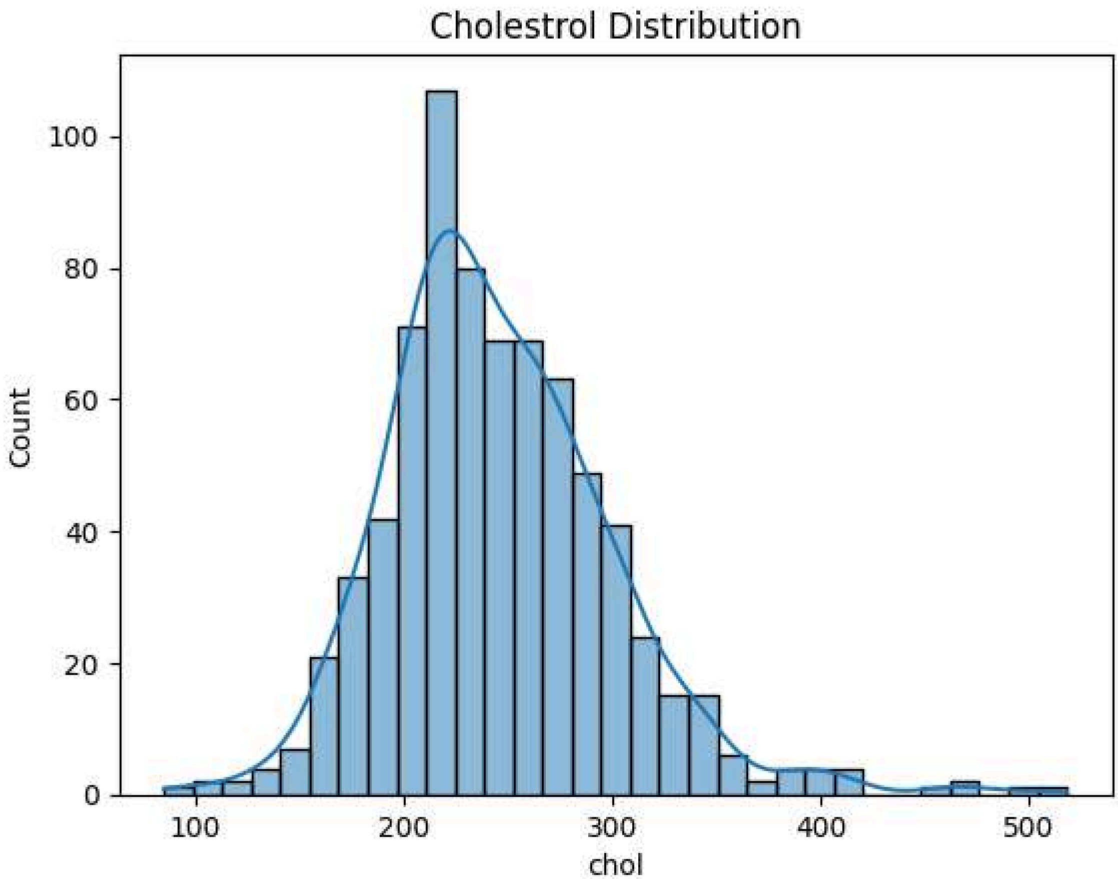
Chol Column

The normal cholesterol level is less than 200 mg/dL.

In [227…
```
df['chol'].describe()
```

Out[227…
```
count    740.000000
mean     245.442216
std       54.257979
min       85.000000
25%      211.000000
50%      238.500000
75%      275.000000
max      518.000000
Name: chol, dtype: float64
```

In [228…
```
#plot the chol column
sns.histplot(df['chol'], kde=True)
plt.title('Cholestrol Distribution')
plt.show()
```



In [229…
```
df['chol'].value_counts().nlargest(5)
```

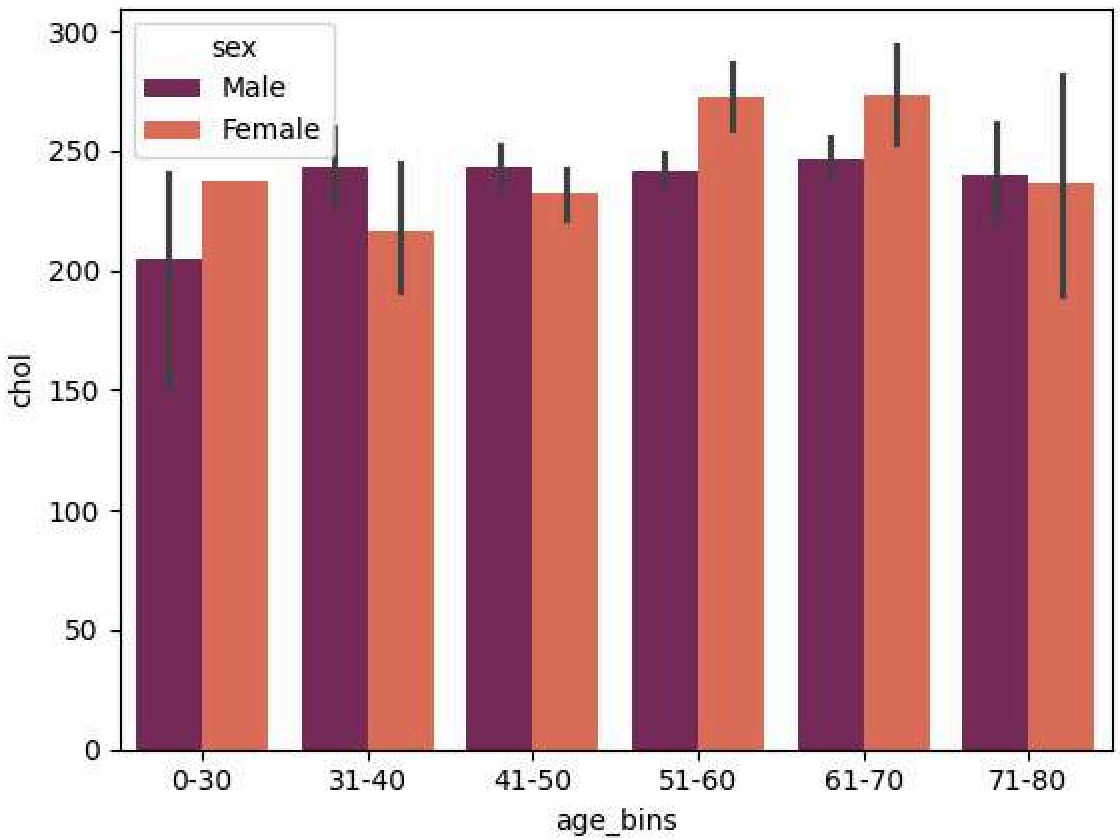
Out[229…
```
chol
254.0    10
220.0    10
223.0     9
230.0     9
204.0     9
Name: count, dtype: int64
```

observation: The majority of the patients have cholesterol levels between 200-300 mg/dl. Which is slightly higher than the normal range.

In [230…
```
#Age Column binning
df['age_bins'] = pd.cut(df['age'], bins=[0, 30, 40, 50, 60, 70, 80], labels=['0-30'
```

In [231…
```
df['age_bins'].value_counts()
```

Out[231…
```
age_bins
51-60    301
41-50    202
61-70    138
31-40     74
71-80     20
0-30       5
Name: count, dtype: int64
```

In [232…
```
df.columns
```
Out[232…
```
Index(['id', 'age', 'sex', 'dataset', 'cp', 'trestbps', 'chol', 'fbs',
       'restecg', 'thalch', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num',
       'age_bins'],
      dtype='object')
```
In [233…
```
sns.barplot(data=df, x='age_bins',y='chol', hue='sex', palette='rocket')
```
Out[233…
```
<Axes: xlabel='age_bins', ylabel='chol'>
```


In [234…
```
#which category has the highest cholestrol
df.groupby('age_bins')['chol'].median().sort_values(ascending=False)
```
Out[234…
```
age_bins
61-70    253.000
51-60    239.000
0-30     237.000
41-50    235.500
71-80    227.215
31-40    223.880
Name: chol, dtype: float64
```
```
The cholesterol level is highest among the age group of 61-70 years.
```

FBS Column

fbs column tells us about the fasting blood sugar levels of the patients.

In [235…

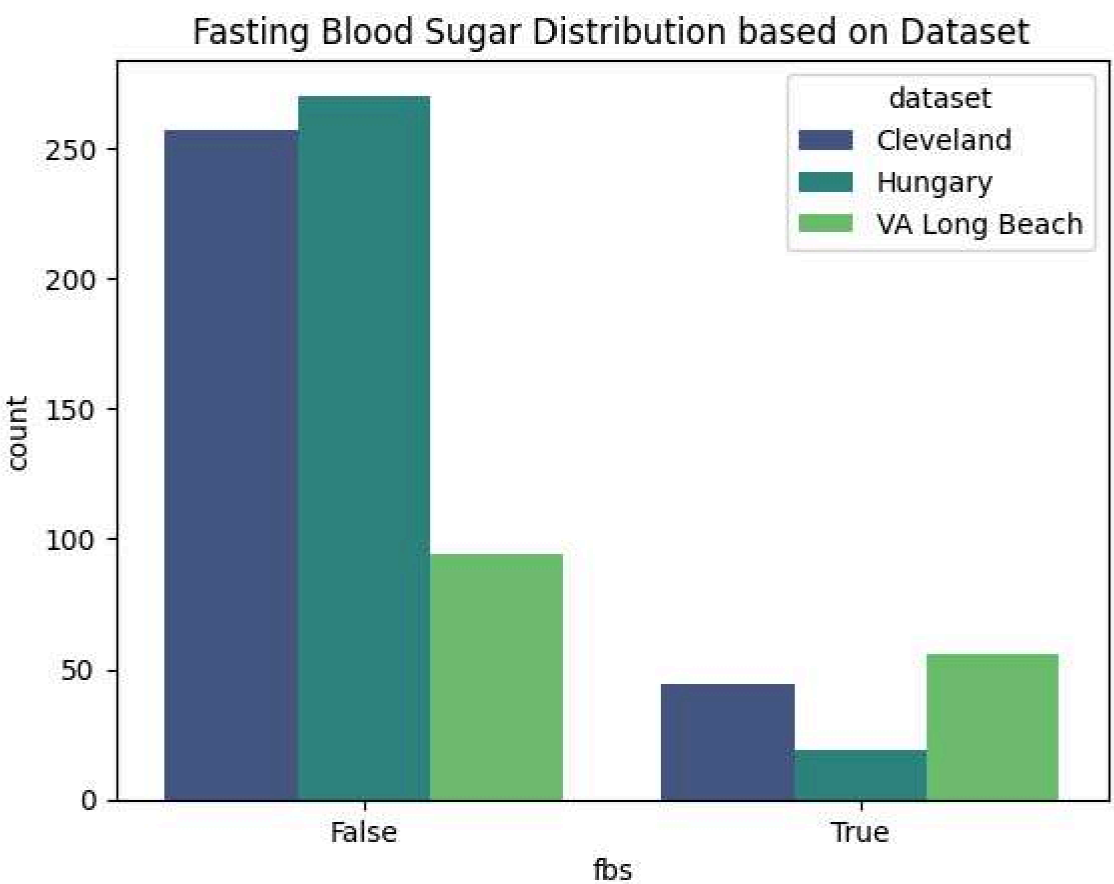
```
df['fbs'].value_counts()
```

Out[235…

```
fbs
False    621
True     119
Name: count, dtype: int64
```

In [236…

```
#make a good plot of fbs column using sns
sns.countplot(data=df, x='fbs', hue='dataset', palette='viridis')
plt.title('Fasting Blood Sugar Distribution based on Dataset')
plt.show()
```



`Observation: The majority of the patients have fasting blood sugar levels less than 120 mg/dl.`

Restecg Column

In [237…

```
df['restecg'].value_counts()
```

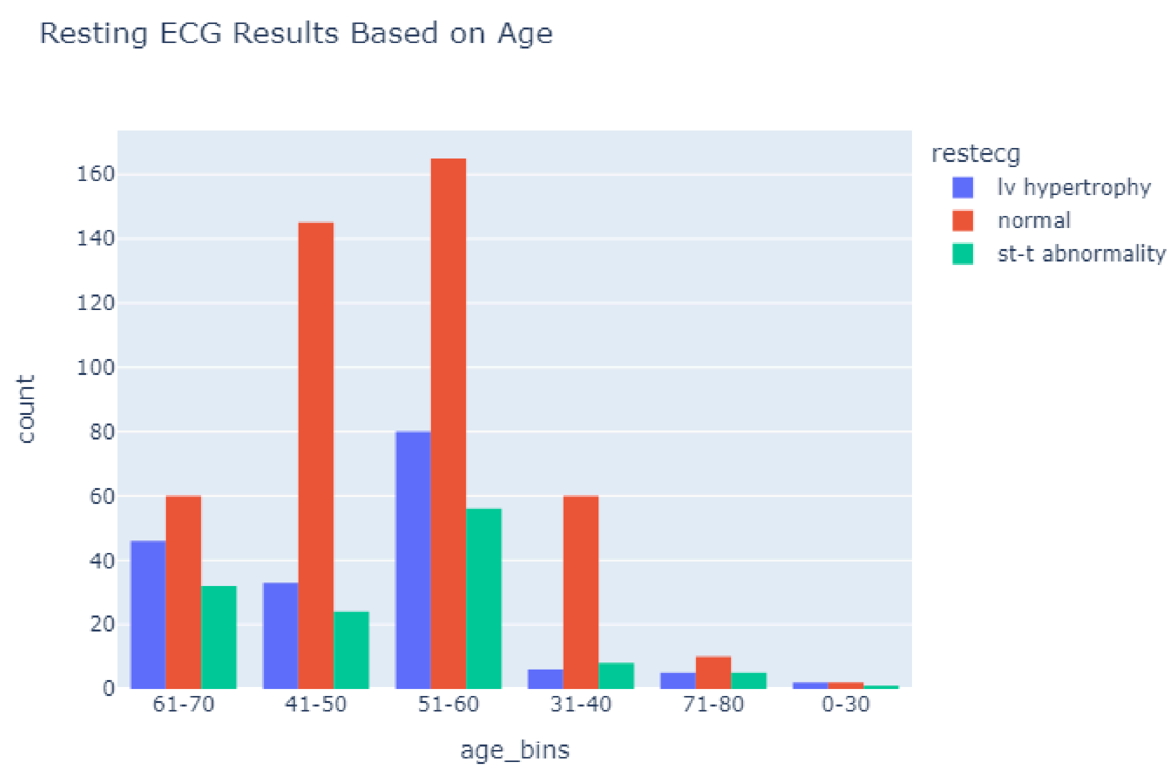
Out[237…

```
restecg
normal              442
lv hypertrophy      172
st-t abnormality    126
Name: count, dtype: int64
```

1. `Normal:` A healthy ECG reading with no signs of heart problems.
2. `LV Hypertrophy:` Thickening of the heart's left side, which can happen when the heart works too hard.
3. `ST-T Abnormality:` Unusual patterns in part of the ECG that may point to heart issues like reduced blood flow or heart attack.

In [238…

```
#plot restecg using plotly count plot
fig = px.histogram(df, x='age_bins', color='restecg', barmode='group', title='Resti
fig.show()
```



`Observation: According to our dataset, majority of the patients have normal Resting ECG but some patients have ST-T wave abnormality. which may indicate heart issues.`

Thalch Column

In [239…

```
df['thalch'].value_counts().nlargest(5)
```

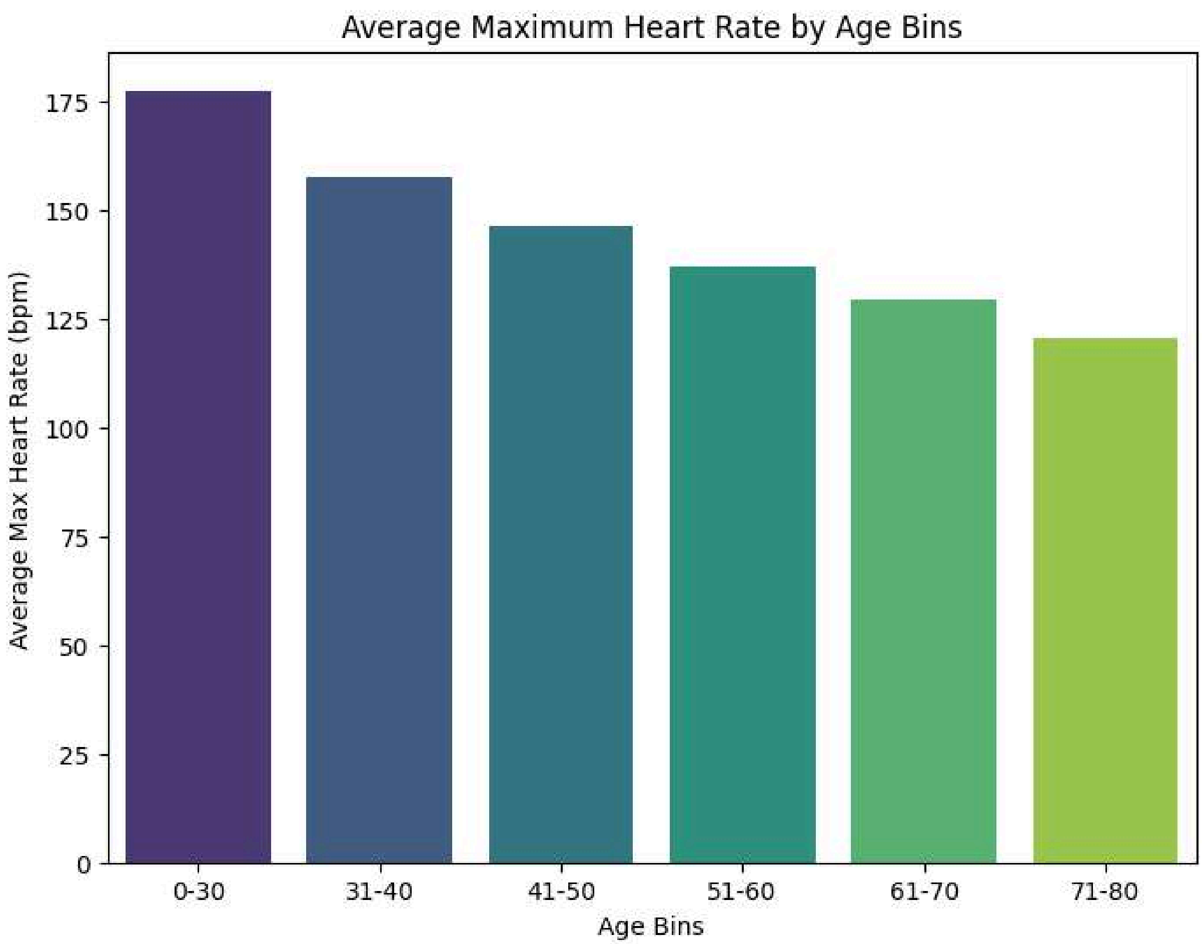
Out[239…

```
thalch
150.0    38
140.0    37
130.0    25
160.0    24
120.0    21
Name: count, dtype: int64
```

In [240…

```python
#groupby thalch based on age_bins
average_thalch = df.groupby('age_bins')['thalch'].mean().sort_values(ascending=Fals
print(average_thalch)
#plotting it
plt.figure(figsize=(8, 6))
sns.barplot(x=average_thalch.index, y=average_thalch.values, palette='viridis')
plt.title('Average Maximum Heart Rate by Age Bins')
plt.xlabel('Age Bins')
plt.ylabel('Average Max Heart Rate (bpm)')
plt.show()
```

```
age_bins
0-30     177.400000
31-40    157.485270
41-50    146.279257
51-60    136.791163
61-70    129.196739
71-80    120.405500
Name: thalch, dtype: float64
```



The plot illustrates that average maximum heart rates decline with age.

`Observation: The young age group has a higher heart rate as compared to the older age group.`

Exang Column

In [241…

```
df['exang'].value_counts()
```

Out[241…

```
exang
False    451
True     289
Name: count, dtype: int64
```

This Column indicates whether a person experiences angina (chest pain) during physical exertion.

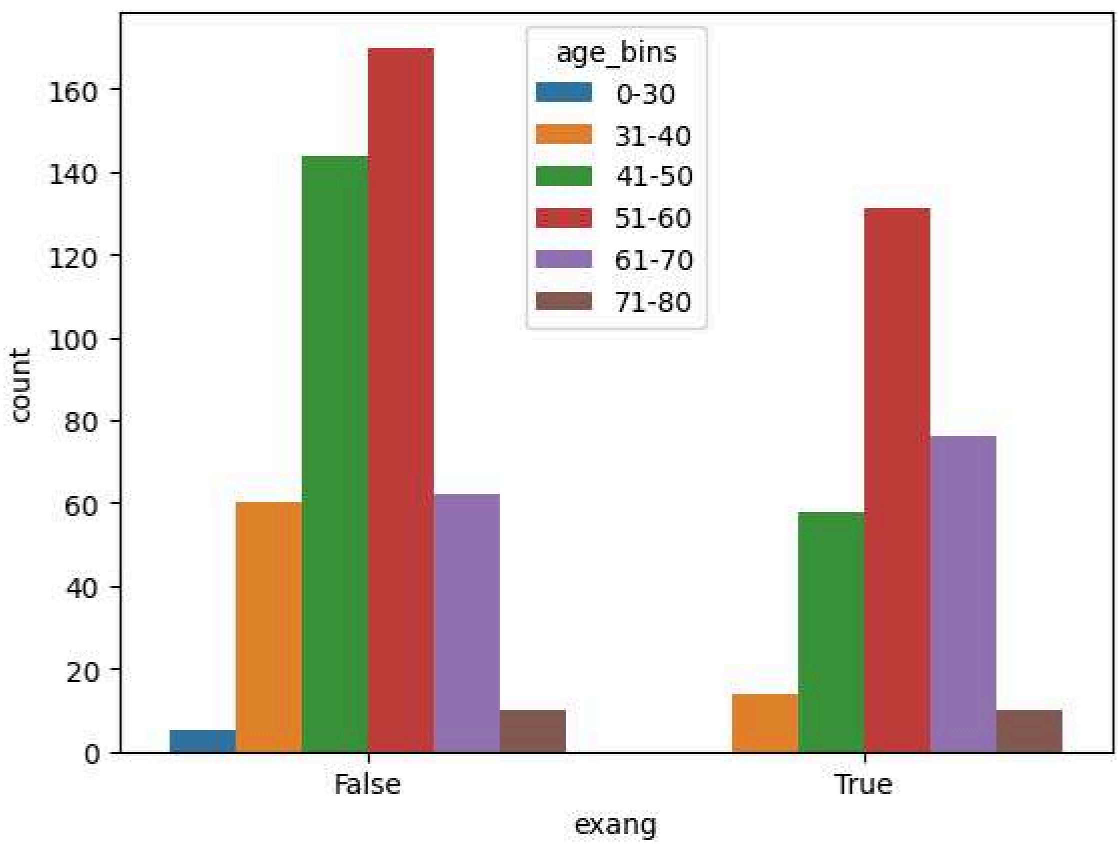
True: The individual experiences angina when exercising.
False: The individual does not experience angina when exercising.

In [242…

```
sns.countplot(data=df,x='exang',hue='age_bins')
```

Out[242…

```
<Axes: xlabel='exang', ylabel='count'>
```



`Observation: According to our dataset, the majority of the patients does not experience angina during physical exertion but age group 51-60 has the`

```
highest number of patients who experience angina during physical
exertion.
```

Oldpeak Column

1. It indicates how much the ST segment falls below the baseline during exercise.
2. A higher oldpeak value suggests more significant ST depression, which can indicate myocardial ischemia (reduced blood flow to the heart).

In [243…
```
df['oldpeak'].value_counts().nlargest(5)
```

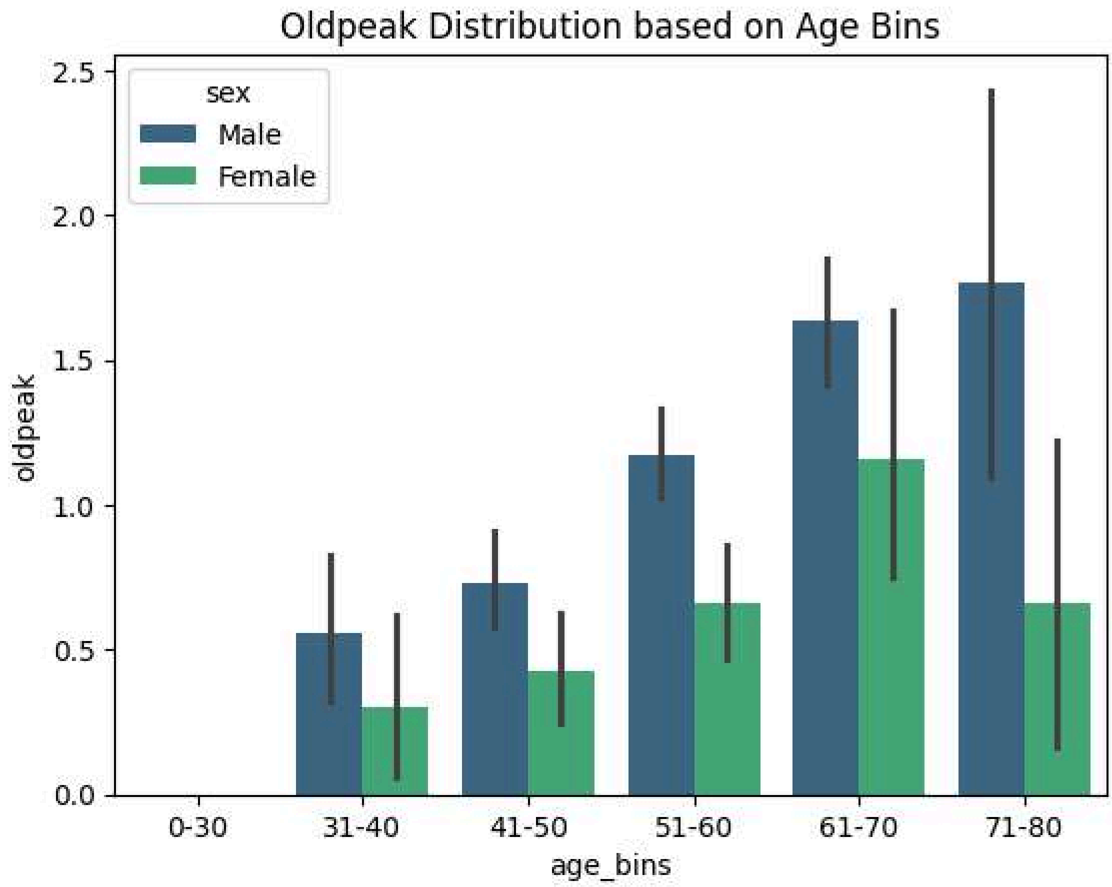
Out[243…
```
oldpeak
0.0    316
1.0     66
2.0     56
1.5     34
3.0     23
Name: count, dtype: int64
```

- 0: No ST depression (normal, healthy heart response).
- 0 to 1 mm: Mild ST depression, usually not concerning but can be observed in some cases.
- Greater than 1 mm: Clinically significant ST depression, which may indicate myocardial ischemia (reduced blood flow to the heart) and is often associated with coronary artery disease.

In [244…
```
#groupby oldpeak based on age_bins
df.groupby('age_bins')['oldpeak'].mean().sort_values(ascending=False)
```

Out[244…
```
age_bins
61-70    1.526609
71-80    1.492450
51-60    1.055847
41-50    0.643119
31-40    0.499041
0-30     0.000000
Name: oldpeak, dtype: float64
```

In [245…
```
#plot oldpeak column based on age_bins using sns
sns.barplot(data=df, x='age_bins', y='oldpeak', palette='viridis',hue='sex')
plt.title('Oldpeak Distribution based on Age Bins')
plt.show()
```



Ages 0-30: 0.00 (no ST depression, normal heart response).
Ages 31-40: 0.50 (mild ST depression).
Ages 41-50: 0.64 (moderate ST depression).
Ages 51-60: 1.05 (significant ST depression).
Ages 61-70: 1.52 (higher level of ST depression).
Ages 71-80: 1.46 (still high, slightly lower than the 61-70 group).

`Observations:`

1. ST depression (oldpeak) rises with age, showing a higher risk of heart issues in older age groups.
2. Age groups 51-80 have average oldpeak values over 1 mm, indicating clinically significant heart stress.
3. The 61-70 group has the highest average oldpeak (1.52 mm), suggesting increased heart disease risk in this age bracket.
4. Males have higher oldpeak values as compared to Femlaes.

Slope Column

In [246…

```
df['slope'].value_counts()
```

Out[246...

```
slope
flat           412
upsloping      270
downsloping     58
Name: count, dtype: int64
```

- Flat (427 cases): Most common slope, indicating a higher likelihood of ischemia.
- Upsloping (254 cases): Suggests healthier heart function; less concerning.
- Downsloping (59 cases): Least common but most alarming, indicating severe heart disease.

In [247...

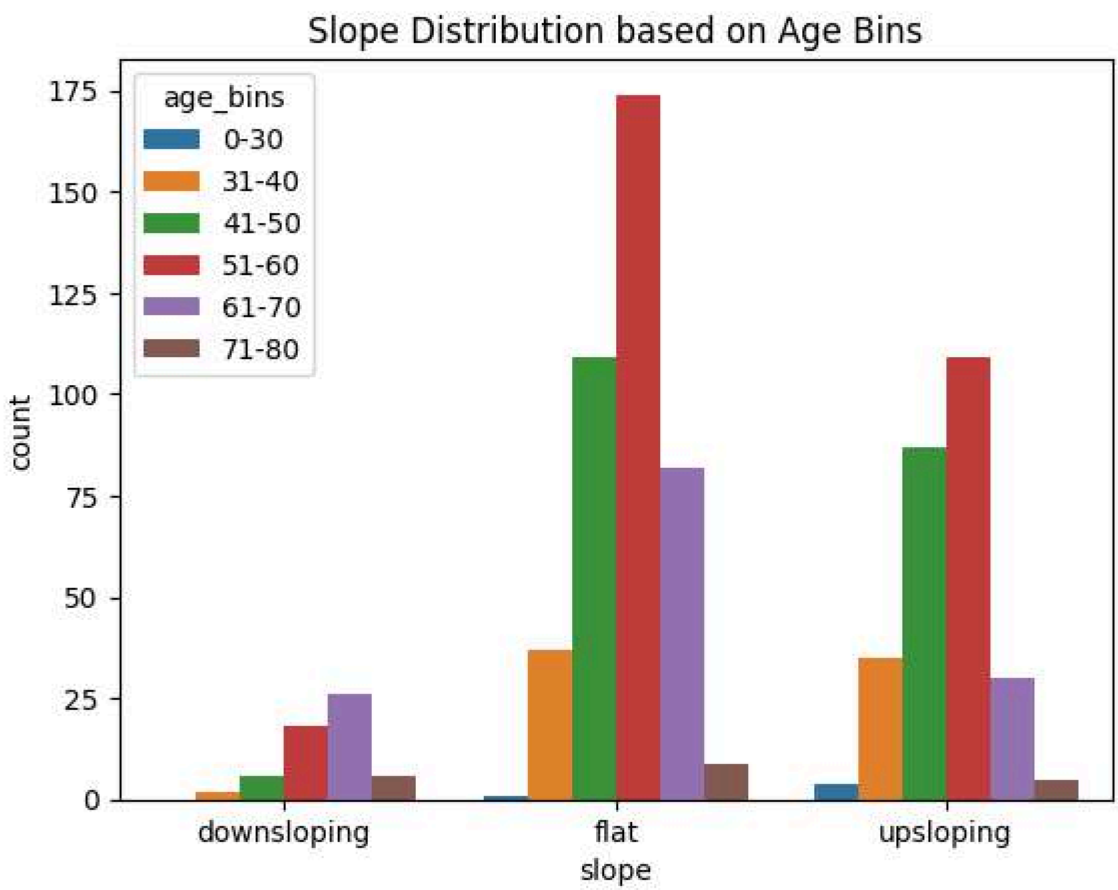
```python
#groupby slope based on age_bins
df.groupby('age_bins')['slope'].value_counts()
```

Out[247...

```
age_bins  slope
0-30      upsloping        4
          flat             1
          downsloping      0
31-40     flat            37
          upsloping       35
          downsloping      2
41-50     flat           109
          upsloping       87
          downsloping      6
51-60     flat           174
          upsloping      109
          downsloping     18
61-70     flat            82
          upsloping       30
          downsloping     26
71-80     flat             9
          downsloping      6
          upsloping        5
Name: count, dtype: int64
```

In [248...

```python
#plot the slope column based on age_bins using sns
sns.countplot(data=df, x='slope', hue='age_bins')
plt.title('Slope Distribution based on Age Bins')
plt.show()
```



`Observations:`

- Younger Age Groups (0-40): Primarily exhibit upsloping and flat slopes, suggesting relatively healthier heart responses.
- Middle Age Groups (41-60): Higher counts of flat slopes (up to 177) indicate an increase in potential ischemia risk.
- Older Age Groups (61-80): A mix of slopes, with a notable presence of downsloping (8 cases in 71-80), indicating a concerning trend toward severe heart conditions.

CA Column

In [249…

```
df['ca'].value_counts()
```

Out[249…

```
ca
0.0    533
1.0    115
2.0     71
3.0     21
Name: count, dtype: int64
```

- 0: No major vessels colored (indicating no visible blockages).
- 1: One major vessel colored.
- 2: Two major vessels colored.

- 3: Three major vessels colored (indicating significant blockage or severe coronary artery disease).

In [250…

```
#groupby ca based on age_bins & sex
df.groupby(['age_bins', 'sex'])['ca'].count().reset_index()
```

Out[250…

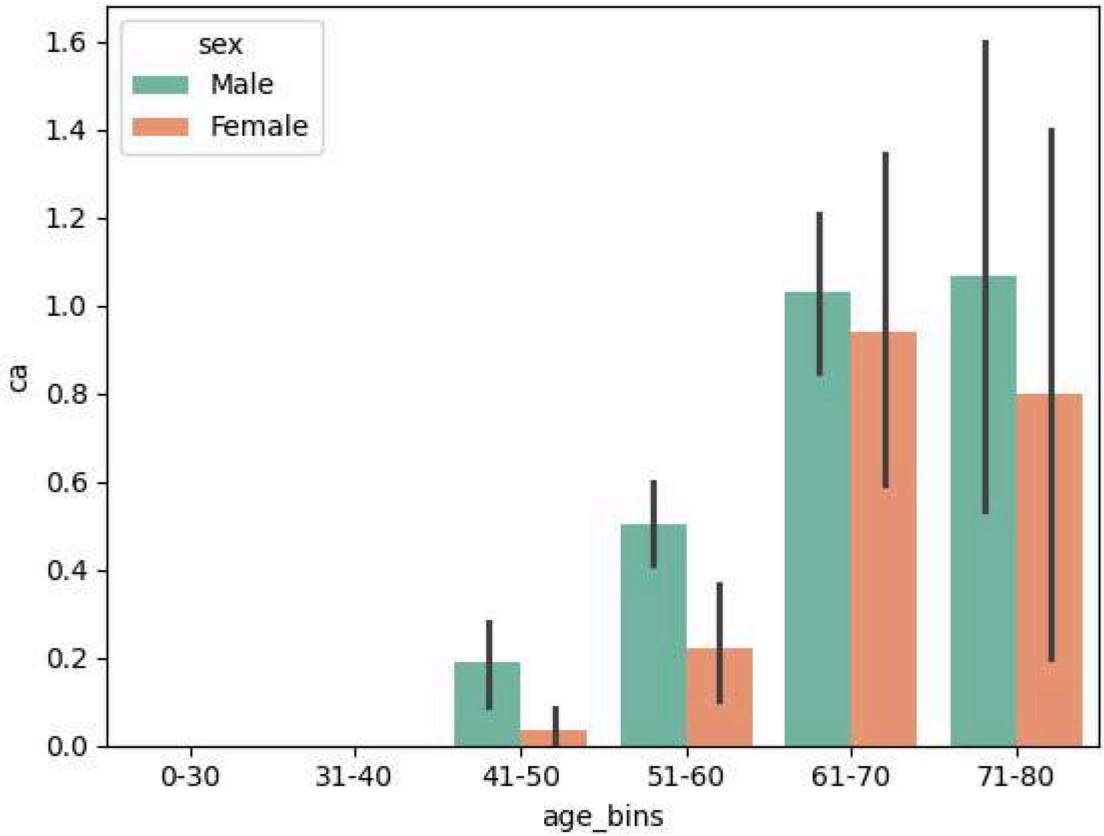
	age_bins	sex	ca
0	0-30	Female	1
1	0-30	Male	4
2	31-40	Female	17
3	31-40	Male	57
4	41-50	Female	58
5	41-50	Male	144
6	51-60	Female	68
7	51-60	Male	233
8	61-70	Female	32
9	61-70	Male	106
10	71-80	Female	5
11	71-80	Male	15

In [251…

```
#plot ca based on age_bins
sns.barplot(data=df, x='age_bins',y='ca',hue='sex', palette='Set2')
```

Out[251…

```
<Axes: xlabel='age_bins', ylabel='ca'>
```



`Observations:`

- Males show more affected vessels across all age bins.
- Significant rise in affected vessels with age, especially in males aged 51-60 (233 cases).
- Few cases in the 0-30 age group (1 female, 4 male).
- Notable increase in the 41-50 age group (144 males).

Thal Column

Helps diagnose coronary artery disease and guides treatment decisions based on blood flow patterns.

In [252…

```
df['thal'].value_counts()
```

Out[252…

```
thal
reversable defect    352
normal               325
fixed defect          63
Name: count, dtype: int64
```

- Reversible Defect (353 cases): Indicates temporary reduced blood flow during stress, suggesting ischemia.
- Normal (325 cases): Shows normal blood flow, indicating no significant heart disease.

- Fixed Defect (62 cases): Indicates permanent reduced blood flow, suggesting previous heart damage.

In [253…

```
df.groupby(['age_bins', 'sex'])['thal'].size().reset_index()
```

Out[253…

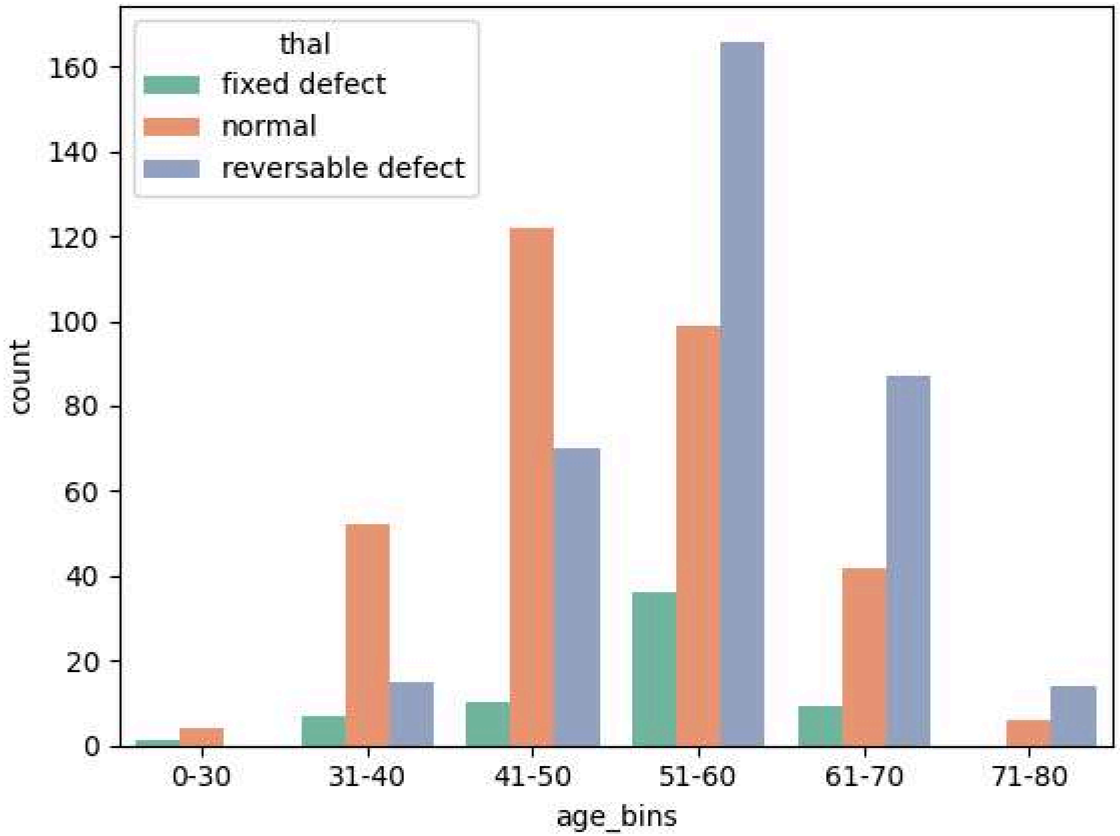
	age_bins	sex	thal
0	0-30	Female	1
1	0-30	Male	4
2	31-40	Female	17
3	31-40	Male	57
4	41-50	Female	58
5	41-50	Male	144
6	51-60	Female	68
7	51-60	Male	233
8	61-70	Female	32
9	61-70	Male	106
10	71-80	Female	5
11	71-80	Male	15

In [254…

```
sns.countplot(data=df, x='age_bins', hue='thal', palette='Set2')
```

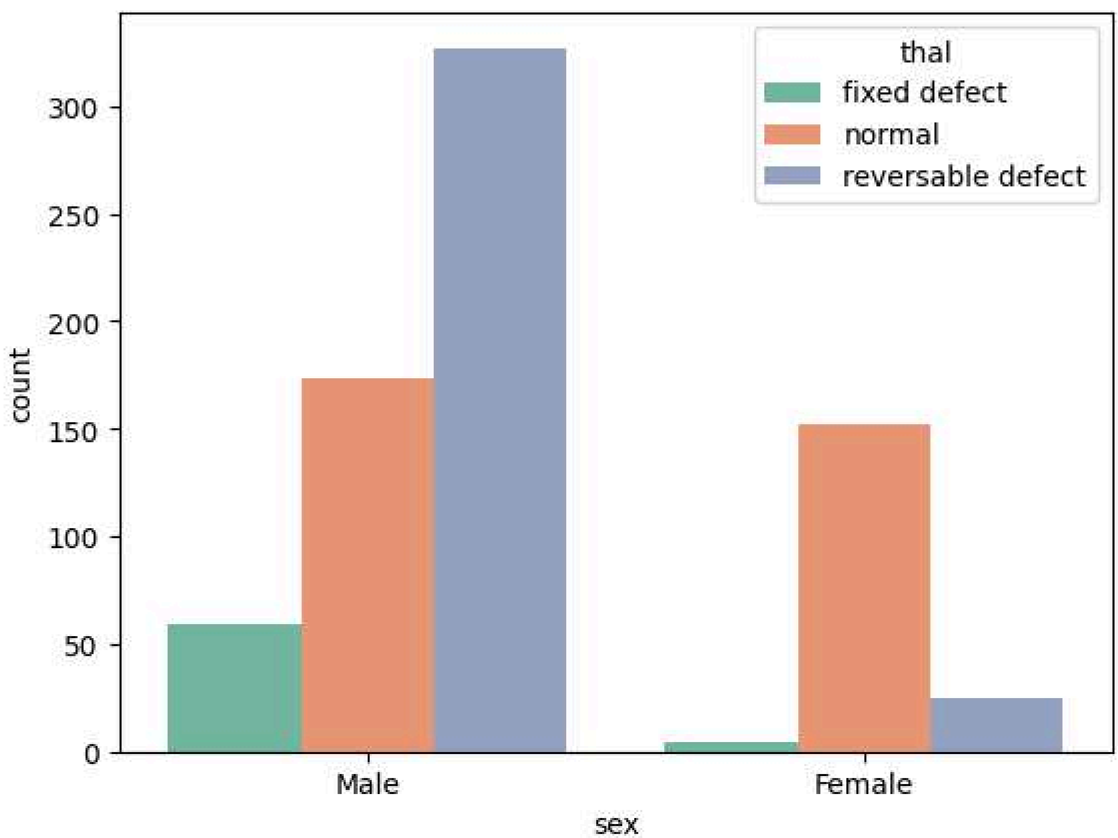
Out[254…

```
<Axes: xlabel='age_bins', ylabel='count'>
```



In [255…
```
sns.countplot(data=df, x='sex', hue='thal', palette='Set2')
```

Out[255…
```
<Axes: xlabel='sex', ylabel='count'>
```



`Observations:`

- Very few cases (1 female, 4 males) in 0-30 age group, indicating low heart disease risk.
- Significant increase in 41-50 age group, especially in males (144 cases), indicating higher risk.
- Highest count in 51-60 age group (233 males), suggesting urgent monitoring.
- Females consistently show lower counts across all age bins.
- Notable cases in 61-70 age group (32 females, 106 males), highlighting increased risk.

Num Column

In [256…

```
df['num'].value_counts()
```

Out[256…

```
num
0    389
1    199
3     68
2     63
4     21
Name: count, dtype: int64
```

- `0 = no heart disease`
- `1 = mild heart disease`
- `2 = moderate heart disease`
- `3 = severe heart disease`
- `4 = critical heart disease`

In [257…

```
df.groupby(['age_bins', 'sex', 'num']).size().reset_index(name='count')
```

Out[257…

	age_bins	sex	num	count
0	0-30	Female	0	1
1	0-30	Female	1	0
2	0-30	Female	2	0
3	0-30	Female	3	0
4	0-30	Female	4	0
5	0-30	Male	0	4
6	0-30	Male	1	0
7	0-30	Male	2	0
8	0-30	Male	3	0
9	0-30	Male	4	0
10	31-40	Female	0	15
11	31-40	Female	1	2
12	31-40	Female	2	0
13	31-40	Female	3	0
14	31-40	Female	4	0
15	31-40	Male	0	39
16	31-40	Male	1	15
17	31-40	Male	2	0
18	31-40	Male	3	2
19	31-40	Male	4	1
20	41-50	Female	0	52
21	41-50	Female	1	5
22	41-50	Female	2	1
23	41-50	Female	3	0
24	41-50	Female	4	0
25	41-50	Male	0	77
26	41-50	Male	1	53
27	41-50	Male	2	5
28	41-50	Male	3	8
29	41-50	Male	4	1

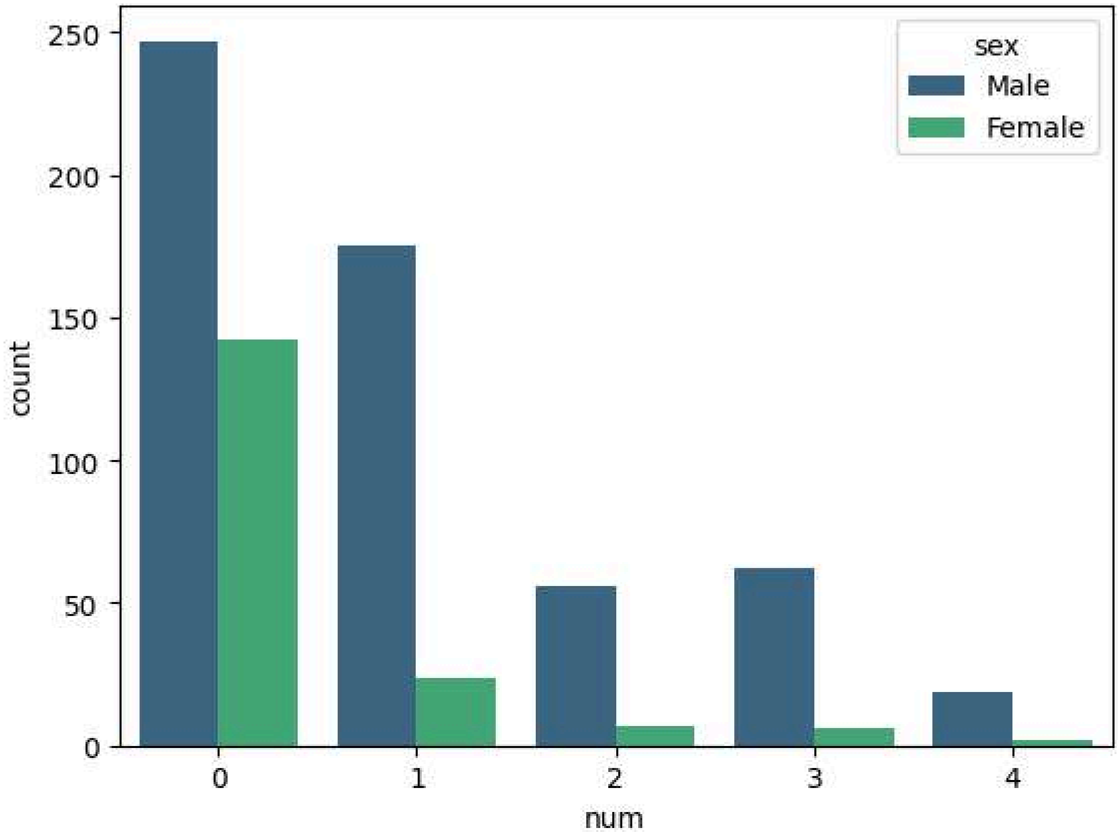
	age_bins	sex	num	count
30	51-60	Female	0	50
31	51-60	Female	1	11
32	51-60	Female	2	4
33	51-60	Female	3	3
34	51-60	Female	4	0
35	51-60	Male	0	100
36	51-60	Male	1	80
37	51-60	Male	2	23
38	51-60	Male	3	23
39	51-60	Male	4	7
40	61-70	Female	0	19
41	61-70	Female	1	6
42	61-70	Female	2	2
43	61-70	Female	3	3
44	61-70	Female	4	2
45	61-70	Male	0	26
46	61-70	Male	1	24
47	61-70	Male	2	26
48	61-70	Male	3	22
49	61-70	Male	4	8
50	71-80	Female	0	5
51	71-80	Female	1	0
52	71-80	Female	2	0
53	71-80	Female	3	0
54	71-80	Female	4	0
55	71-80	Male	0	1
56	71-80	Male	1	3
57	71-80	Male	2	2
58	71-80	Male	3	7
59	71-80	Male	4	2

In [258…

```
#plot num column on age_bins
sns.countplot(data=df, x='num', hue='sex', palette='viridis')
```

Out[258…

```
<Axes: xlabel='num', ylabel='count'>
```

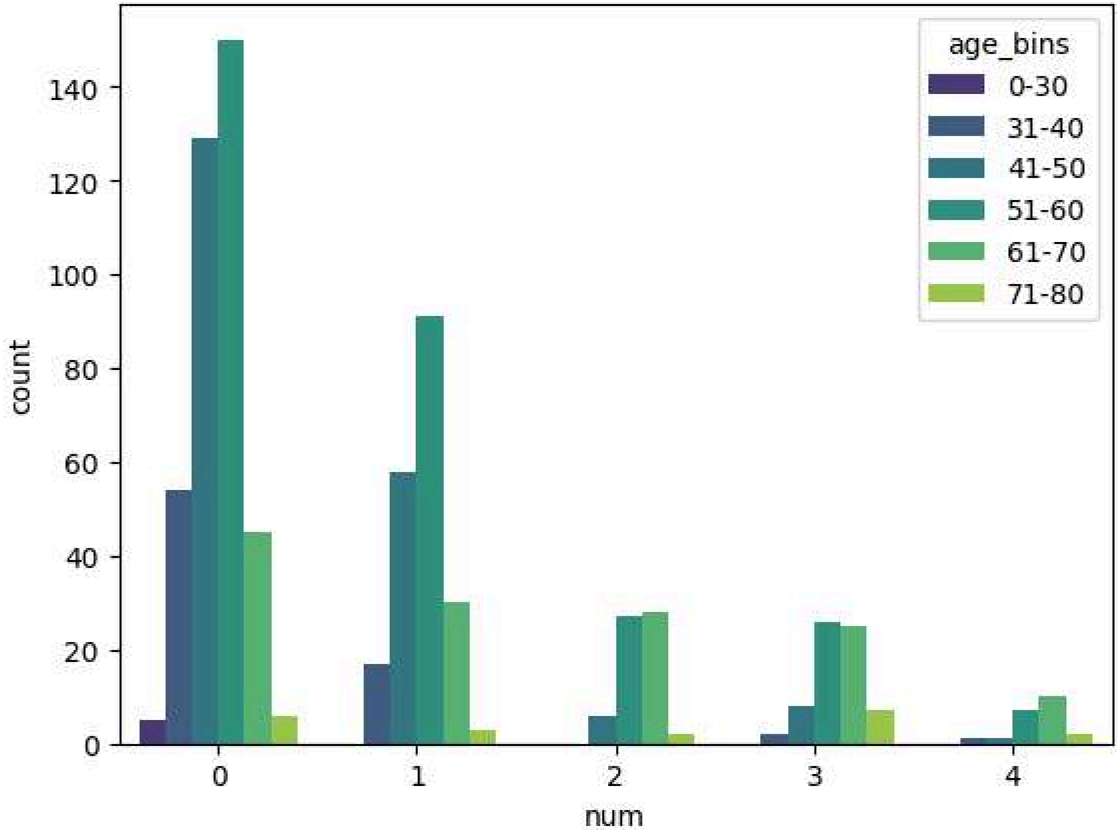


In [259…

```
sns.countplot(data=df, x='num', hue='age_bins', palette='viridis')
```

Out[259…

```
<Axes: xlabel='num', ylabel='count'>
```



`Observations:`

- Minimal heart disease predictions in the 0-30 age group.
- Significant cases (especially num values 1 and 2) in the 41-50 age group.
- Severe predictions rise in 51-60 and 61-70 age groups for males.
- Females show fewer predicted heart disease cases overall.

Machine Learning

In [260…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 740 entries, 0 to 919
Data columns (total 17 columns):
 #   Column    Non-Null Count  Dtype   
---  ------    --------------  -----   
 0   id        740 non-null    int64   
 1   age       740 non-null    int64   
 2   sex       740 non-null    object  
 3   dataset   740 non-null    object  
 4   cp        740 non-null    object  
 5   trestbps  740 non-null    float64 
 6   chol      740 non-null    float64 
 7   fbs       740 non-null    object  
 8   restecg   740 non-null    object  
 9   thalch    740 non-null    float64 
 10  exang     740 non-null    object  
 11  oldpeak   740 non-null    float64 
 12  slope     740 non-null    object  
 13  ca        740 non-null    float64 
 14  thal      740 non-null    object  
 15  num       740 non-null    int64   
 16  age_bins  740 non-null    category
dtypes: category(1), float64(5), int64(3), object(8)
memory usage: 99.2+ KB
```

In [261…

```python
df['num'].value_counts()
```

Out[261…

```
num
0    389
1    199
3     68
2     63
4     21
Name: count, dtype: int64
```

The Target Column is `num` which is the predicted attribute. We will use this column to predict the heart disease. The unique values in this column are: [0, 1, 2, 3, 4], which states that there are 5 types of heart diseases.

- `0 = no heart disease`
- `1 = mild heart disease`
- `2 = moderate heart disease`
- `3 = severe heart disease`
- `4 = critical heart disease`

For this project, we will convert the `num` column into a binary classification problem. We will consider the following values:

- `0 = no heart disease`
- `1 = heart disease`

It will make easy a model to predict the heart disease.

In [262…

```
#split the data into X and Y
X = df.drop(['num','id'], axis=1)
y = df['num']
#target engineering on num
y = np.where((y == 1) | (y == 2) | (y == 3) | (y == 4), 1,0)
```

In [263…

```
label_encoder = LabelEncoder()

for col in X.columns:
    if X[col].dtype == 'object' or X[col].dtype == 'category':
        X[col] = label_encoder.fit_transform(X[col])
    else:
        pass
```

In [264…

```
print(numeric_cols)
```

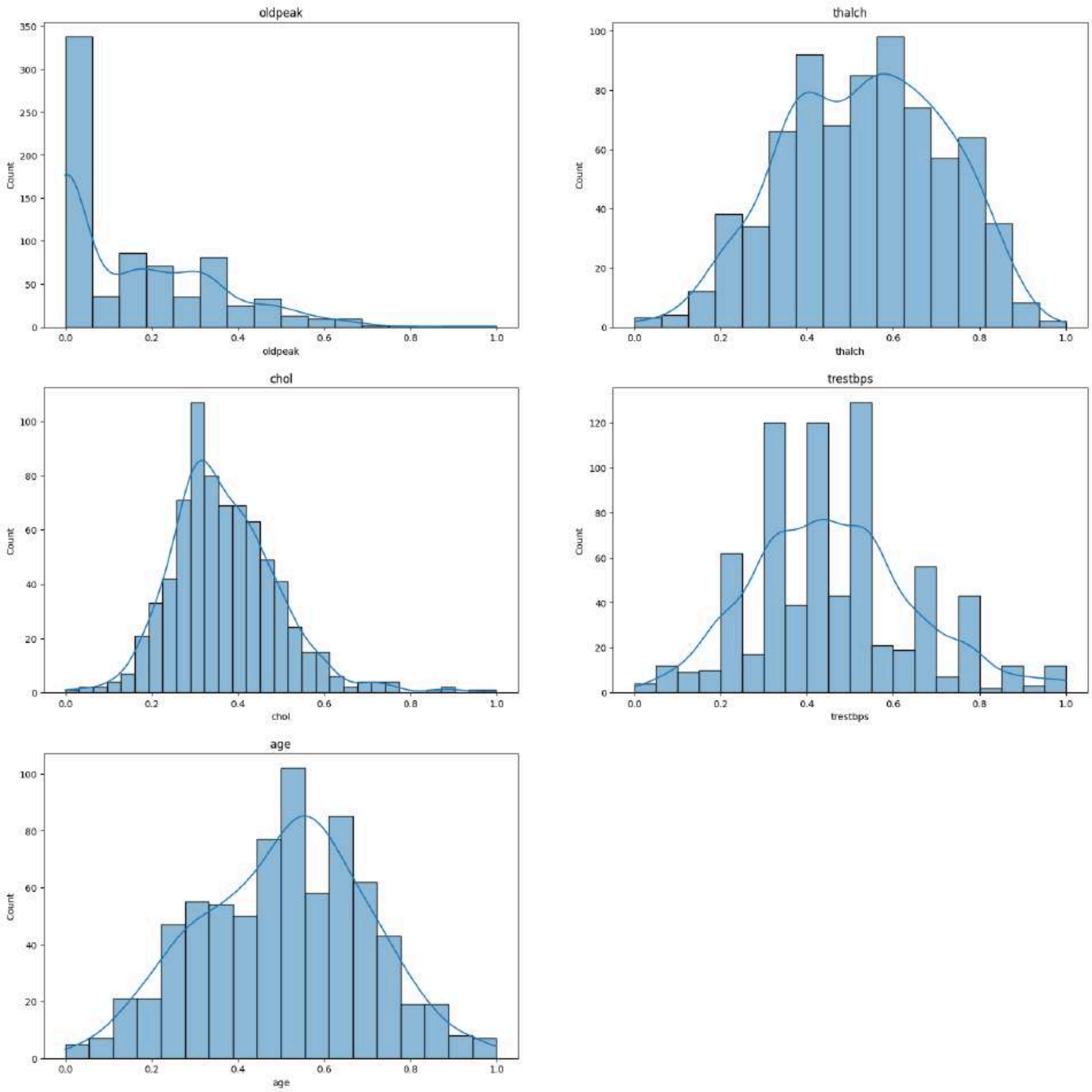
```
['oldpeak', 'thalch', 'chol', 'trestbps', 'age']
```

In [265…

```
#Scaling numeric columns
min_max_scaler = MinMaxScaler()
X[numeric_cols] = min_max_scaler.fit_transform(X[numeric_cols])
```

In [266…

```
#plot all numeric columns
plt.figure(figsize=(20,20))
for i, col in enumerate(numeric_cols):
    plt.subplot(3,2, i+1)
    sns.histplot(X[col], kde=True)
    plt.title(col)
```



In [267…

```
# split the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_st
```

The following models will be used to predict the heart disease:

1. Random Forest
2. Gradient Boosting
3. Support Vector Machine (SVM)
4. Logistic Regression
5. K-Nearest Neighbors (KNN)
6. Decision Tree
7. AdaBoost
8. XGBoost
9. Naive Bayes

In [268…

```
# Define models and hyperparameters
models = {
    'Random Forest': RandomForestClassifier(random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42),
    'Support Vector Machine': SVC(random_state=42),
    'Logistic Regression': LogisticRegression(random_state=42),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Ada Boost': AdaBoostClassifier(random_state=42),
    'XG Boost': XGBClassifier(random_state=42),
    'Naive Bayes': GaussianNB()
}

params = {
    'Random Forest': {
        'model__n_estimators': [100, 200, 300],
        'model__max_depth': [ 10,20],
        'model__min_samples_split': [2, 5]
    },
    'Gradient Boosting': {
        'model__n_estimators': [100, 200],
        'model__learning_rate': [0.1, 0.01],
        'model__max_depth': [3, 5]
    },
    'Support Vector Machine': {
        'model__C': [1, 10],
        'model__gamma': [0.1, 0.01]
    },
    'Logistic Regression': {
        'model__C': [1, 10],
        'model__solver': ['lbfgs', 'liblinear']
    },
    'K-Nearest Neighbors': {
        'model__n_neighbors': [3, 5],
        'model__weights': ['uniform', 'distance']
    },
    'Decision Tree': {
        'model__criterion': ['gini', 'entropy'],
        'model__max_depth': [None, 10],
        'model__min_samples_split': [2, 5]
    },
    'Ada Boost': {
        'model__n_estimators': [50, 100],
        'model__learning_rate': [0.1, 0.01]
    },
    'XG Boost': {
        'model__n_estimators': [100, 200],
        'model__learning_rate': [0.1, 0.01],
        'model__max_depth': [3, 5]
    },
    'Naive Bayes': {
        'model__var_smoothing': [1e-9, 1e-10]
    }
}
```

```python
# Initialize best model tracking
best_model = None
best_accuracy = 0.0

# Train and evaluate each model
for name, model in models.items():
    print(f"Training {name}...")

    # Create a pipeline with the model
    pipeline = Pipeline([
        ('model', model)
    ])

    # Get hyperparameters for the current model
    model_params = params.get(name, {})

    # Create GridSearchCV with the pipeline and parameters
    grid_search = GridSearchCV(pipeline, model_params, cv=5, n_jobs=-1, verbose=0)

    # Fit the pipeline
    grid_search.fit(X_train, y_train)

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

    # Print evaluation metrics
    print(f"{name} - Best Parameters: {grid_search.best_params_}")
    print(f"{name} - Best Score: {grid_search.best_score_}")
    print(f"{name} - Test Accuracy: {accuracy_score(y_test, y_pred)}")
    print(f"{name} - Confusion Matrix:\n{confusion_matrix(y_test, y_pred)}")
    print(f"{name} - Classification Report:\n{classification_report(y_test, y_pred)
    print('\n')

    if accuracy_score(y_test, y_pred) > best_accuracy:
        best_accuracy = accuracy_score(y_test, y_pred)
        best_model = grid_search.best_estimator_

# print the best model & accuracy
print(f"The Best model is {best_model.named_steps['model']} with an accuracy of {be


# Save the best model (optional)
# import pickle
# pickle.dump(best_model, open('best_model.pkl', 'wb'))
```

```
Training Random Forest...
Random Forest - Best Parameters: {'model__max_depth': 20, 'model__min_samples_spli
t': 5, 'model__n_estimators': 300}
Random Forest - Best Score: 0.8468468468468469
Random Forest - Test Accuracy: 0.8918918918918919
Random Forest - Confusion Matrix:
[[89 10]
 [10 76]]
Random Forest - Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.90      0.90        99
           1       0.88      0.88      0.88        86

    accuracy                           0.89       185
   macro avg       0.89      0.89      0.89       185
weighted avg       0.89      0.89      0.89       185



Training Gradient Boosting...
Gradient Boosting - Best Parameters: {'model__learning_rate': 0.01, 'model__max_dept
h': 3, 'model__n_estimators': 200}
Gradient Boosting - Best Score: 0.8414414414414415
Gradient Boosting - Test Accuracy: 0.9081081081081082
Gradient Boosting - Confusion Matrix:
[[92  7]
 [10 76]]
Gradient Boosting - Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.93      0.92        99
           1       0.92      0.88      0.90        86

    accuracy                           0.91       185
   macro avg       0.91      0.91      0.91       185
weighted avg       0.91      0.91      0.91       185



Training Support Vector Machine...
Support Vector Machine - Best Parameters: {'model__C': 10, 'model__gamma': 0.01}
Support Vector Machine - Best Score: 0.8450450450450451
Support Vector Machine - Test Accuracy: 0.8540540540540541
Support Vector Machine - Confusion Matrix:
[[89 10]
 [17 69]]
Support Vector Machine - Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.90      0.87        99
           1       0.87      0.80      0.84        86

    accuracy                           0.85       185
   macro avg       0.86      0.85      0.85       185
weighted avg       0.86      0.85      0.85       185
```

```
Training Logistic Regression...
Logistic Regression - Best Parameters: {'model__C': 1, 'model__solver': 'lbfgs'}
Logistic Regression - Best Score: 0.8504504504504504
Logistic Regression - Test Accuracy: 0.8594594594594595
Logistic Regression - Confusion Matrix:
[[89 10]
 [16 70]]
Logistic Regression - Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.90      0.87        99
           1       0.88      0.81      0.84        86

    accuracy                           0.86       185
   macro avg       0.86      0.86      0.86       185
weighted avg       0.86      0.86      0.86       185



Training K-Nearest Neighbors...
K-Nearest Neighbors - Best Parameters: {'model__n_neighbors': 5, 'model__weights':
'uniform'}
K-Nearest Neighbors - Best Score: 0.8306306306306308
K-Nearest Neighbors - Test Accuracy: 0.8432432432432433
K-Nearest Neighbors - Confusion Matrix:
[[86 13]
 [16 70]]
K-Nearest Neighbors - Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.87      0.86        99
           1       0.84      0.81      0.83        86

    accuracy                           0.84       185
   macro avg       0.84      0.84      0.84       185
weighted avg       0.84      0.84      0.84       185



Training Decision Tree...
Decision Tree - Best Parameters: {'model__criterion': 'gini', 'model__max_depth': No
ne, 'model__min_samples_split': 2}
Decision Tree - Best Score: 0.8072072072072073
Decision Tree - Test Accuracy: 0.8324324324324325
Decision Tree - Confusion Matrix:
[[84 15]
 [16 70]]
Decision Tree - Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.85      0.84        99
           1       0.82      0.81      0.82        86
```

```
    accuracy                           0.83       185
   macro avg       0.83      0.83      0.83       185
weighted avg       0.83      0.83      0.83       185



Training Ada Boost...
Ada Boost - Best Parameters: {'model__learning_rate': 0.1, 'model__n_estimators': 5
0}
Ada Boost - Best Score: 0.8612612612612613
Ada Boost - Test Accuracy: 0.8756756756756757
Ada Boost - Confusion Matrix:
[[90  9]
 [14 72]]
Ada Boost - Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.91      0.89        99
           1       0.89      0.84      0.86        86

    accuracy                           0.88       185
   macro avg       0.88      0.87      0.87       185
weighted avg       0.88      0.88      0.88       185



Training XG Boost...
XG Boost - Best Parameters: {'model__learning_rate': 0.01, 'model__max_depth': 3, 'm
odel__n_estimators': 200}
XG Boost - Best Score: 0.8522522522522523
XG Boost - Test Accuracy: 0.8972972972972973
XG Boost - Confusion Matrix:
[[92  7]
 [12 74]]
XG Boost - Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.93      0.91        99
           1       0.91      0.86      0.89        86

    accuracy                           0.90       185
   macro avg       0.90      0.89      0.90       185
weighted avg       0.90      0.90      0.90       185



Training Naive Bayes...
Naive Bayes - Best Parameters: {'model__var_smoothing': 1e-09}
Naive Bayes - Best Score: 0.818018018018018
Naive Bayes - Test Accuracy: 0.8378378378378378
Naive Bayes - Confusion Matrix:
[[86 13]
 [17 69]]
Naive Bayes - Classification Report:
              precision    recall  f1-score   support
```

```
           0       0.83      0.87      0.85        99
           1       0.84      0.80      0.82        86

    accuracy                           0.84       185
   macro avg       0.84      0.84      0.84       185
weighted avg       0.84      0.84      0.84       185



The Best model is GradientBoostingClassifier(learning_rate=0.01, n_estimators=200,
                           random_state=42) with an accuracy of 90.81081081081082%
```

In [269…

```python
#Best Model and accuracy
print(f"The Best model is {best_model.named_steps['model']} with an accuracy of {be
```

```
The Best model is GradientBoostingClassifier(learning_rate=0.01, n_estimators=200,
                           random_state=42) with an accuracy of 90.81%
```

Summary

1. The minimum age to have a heart disease starts from 28 years old.
2. Most of the people get heart disease at the age of 53-54 years.
3. Most of the males and females get heart disease at the age of 54-55 years.
4. Male percentage in the data: 78.91%
5. Female Percentage in the data: 21.09%
6. Males are 274.23% more than females in the data.
7. > We have highest number of people from Cleveland (304) and lowest from Switzerland (123).
   > The highest number of females in this dataset are from Cleveland (97) and lowest from VA Long Beach (6).
   > The highest number of males in this dataset are from Hungary (212) and lowest from Switzerland (113).
8. Majority of the Patients have Resting Blood pressure ranges from 110-150 mm Hg.
9. The majority of the patients have cholesterol levels between 200-300 mg/dl. Which is slightly higher than the normal range.
10. The majority of the patients have fasting blood sugar levels less than 120 mg/dl.
11. The majority of the patients have normal Resting ECG but some patients have ST-T wave abnormality. which may indicate heart issues.
12. The young age group has a higher heart rate as compared to the older age group.

13. The majority of the patients does not experience angina during physical exertion but age group 51-60 has the highest number of patients who experience angina during physical exertion.

14. > ST depression (oldpeak) rises with age, showing a higher risk of heart issues in older age groups.
    > Age groups 51-80 have average oldpeak values over 1 mm, indicating clinically significant heart stress.
    > The 61-70 group has the highest average oldpeak (1.52 mm), suggesting increased heart disease risk in this age bracket.
    > Males have higher oldpeak values as compared to Femlaes.

15. > Younger Age Groups (0-40): Primarily exhibit upsloping and flat slopes, suggesting relatively healthier heart responses.
    > Middle Age Groups (41-60): Higher counts of flat slopes (up to 177) indicate an increase in potential ischemia risk.
    > Older Age Groups (61-80): A mix of slopes, with a notable presence of downsloping (8 cases in 71-80), indicating a concerning trend toward severe heart conditions.

16. > Males show more affected vessels across all age bins.
    > Significant rise in affected vessels with age, especially in males aged 51-60 (233 cases).
    > Few cases in the 0-30 age group (1 female, 4 male).
    > Notable increase in the 41-50 age group (144 males).

17. > Very few cases (1 female, 4 males) in 0-30 age group, indicating low heart disease risk.
    > Significant increase in 41-50 age group, especially in males (144 cases), indicating higher risk.
    > Highest count in 51-60 age group (233 males), suggesting urgent monitoring.
    > Females consistently show lower counts across all age bins.
    > Notable cases in 61-70 age group (32 females, 106 males), highlighting increased risk.

18. > Minimal heart disease predictions in the 0-30 age group.
    > Significant cases (especially num values 1 and 2) in the 41-50 age group.
    > Severe predictions rise in 51-60 and 61-70 age groups for males.
    > Females show fewer predicted heart disease cases overall.

19. > The model achieved an accuracy of over 90%, indicating strong predictive performance.
    > The confusion matrix showed high true positives and true negatives, effectively distinguishing between patients with and without heart

> disease.
> Precision and recall were both high, minimizing false positives and ensuring most cases of heart disease were correctly identified.
> The F1 score was robust, balancing precision and recall.
> Feature importance analysis highlighted key factors impacting predictions, aiding in targeted healthcare strategies.

20. **Imputing Missing Values:**
    I imputed missing values using the Random Forest algorithm by training the model on features with complete data. The model predicted missing values based on the relationships learned from other features, providing more accurate imputation than simple methods.

21. **Dealing with Outliers:**
    I handled outliers using the Z-score method. I calculated the Z-scores for numeric features and identified outliers as those with Z-scores greater than 3 or less than -3. I then removed these outliers to ensure the model's accuracy and robustness.