```
# importing libraries

# for data analysis
import pandas as pd
import numpy as np

# for data visualization
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# for data preprocessing
from sklearn.preprocessing import
LabelEncoder,StandardScaler,MinMaxScaler,OneHotEncoder

# for imputing missing values
from sklearn.impute import SimpleImputer,KNNImputer

# import iterative imputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# for machine learning
from sklearn.model_selection import
train_test_split,GridSearchCV,cross_val_score,RandomizedSearchCV

# Given that we anticipate solving our dependent feature through
classification, we will proceed to import libraries tailored for
classification tasks.
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import
RandomForestClassifier,RandomForestRegressor ,AdaBoostClassifier ,Grad
ientBoostingClassifier


# for classification evaluation metrices
from sklearn.metrics import
accuracy_score,classification_report,confusion_matrix,mean_absolute_er
ror,precision_score,mean_squared_error,r2_score

# to perform statistical test
from sklearn.feature_selection import chi2 # for categorical fetures
from sklearn.feature_selection import f_classif # for numerical
features (Anova f-test)

# for ignoring warnings
```

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

# load data from csv file placed locally in our pc
df = pd.read_csv("heart_disease_uci.csv")

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

```
   id  age     sex    dataset               cp  trestbps   chol    fbs  \
0   1   63    Male  Cleveland   typical angina     145.0  233.0   True

1   2   67    Male  Cleveland     asymptomatic     160.0  286.0  False

2   3   67    Male  Cleveland     asymptomatic     120.0  229.0  False

3   4   37    Male  Cleveland      non-anginal     130.0  250.0  False

4   5   41  Female  Cleveland  atypical angina     130.0  204.0  False


          restecg  thalch  exang  oldpeak        slope   ca  \
0  lv hypertrophy   150.0  False      2.3  downsloping  0.0
1  lv hypertrophy   108.0   True      1.5         flat  3.0
2  lv hypertrophy   129.0   True      2.6         flat  2.0
3          normal   187.0  False      3.5  downsloping  0.0
4  lv hypertrophy   172.0  False      1.4    upsloping  0.0

                thal  num
0      fixed defect    0
1            normal    2
2  reversable defect   1
3            normal    0
4            normal    0
```

```python
# expploring datatype of each column
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
```

```
# let' view the shape of the data, number of rows & columns
print(f"This dataframe has {df.shape[0]} rows and {df.shape[1]}
columns.")
```

```
This dataframe has 920 rows and 16 columns.
```

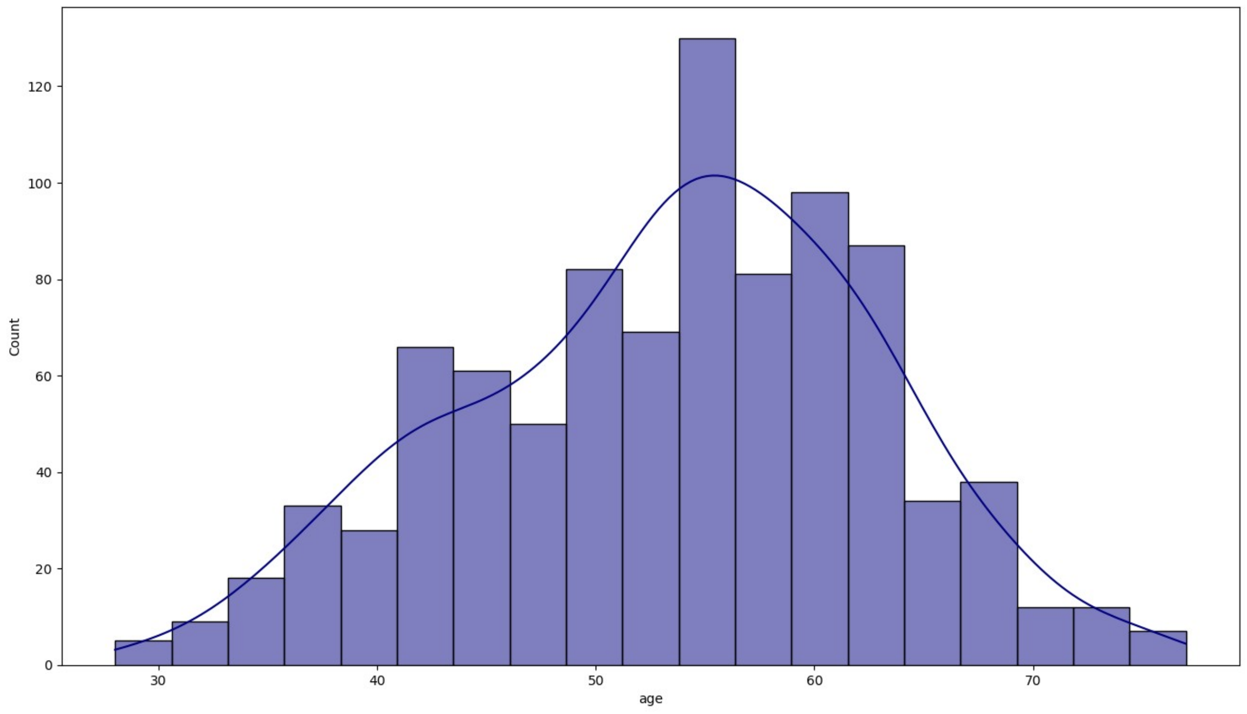
```
# take a look at id column
print(f"The minimum id in id column is {df['id'].min()} and maximum id
is {df['id'].max()}.")
```

```
The minimum id in id column is 1 and maximum id is 920.
```

```
# take a look at age column according to this dataset
print(f"The minimum age in age column is {df['age'].min()} and maximum
age is {df['age'].max()}.")
```

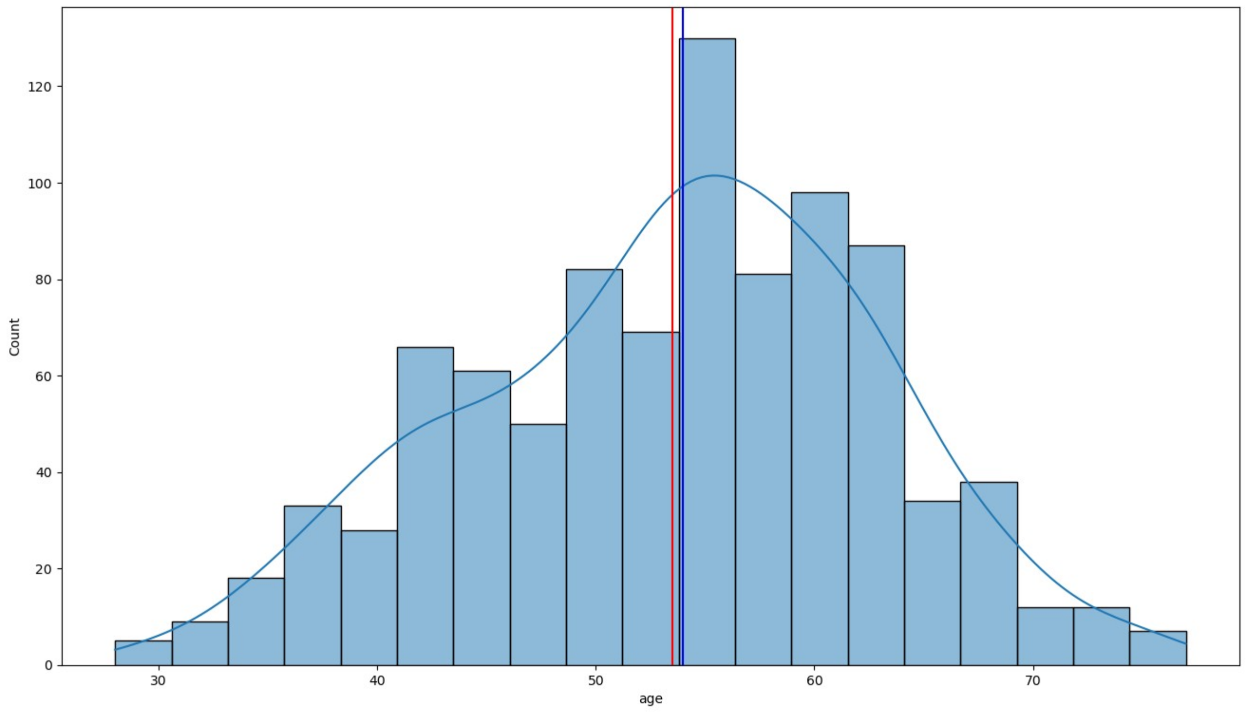
```
The minimum age in age column is 28 and maximum age is 77.
```

```
# Draw a histogram to visualize the distribution of the age column
plt.figure(figsize=(16, 9))
sns.histplot(df["age"], kde=True, color="Navy")
plt.show()
```



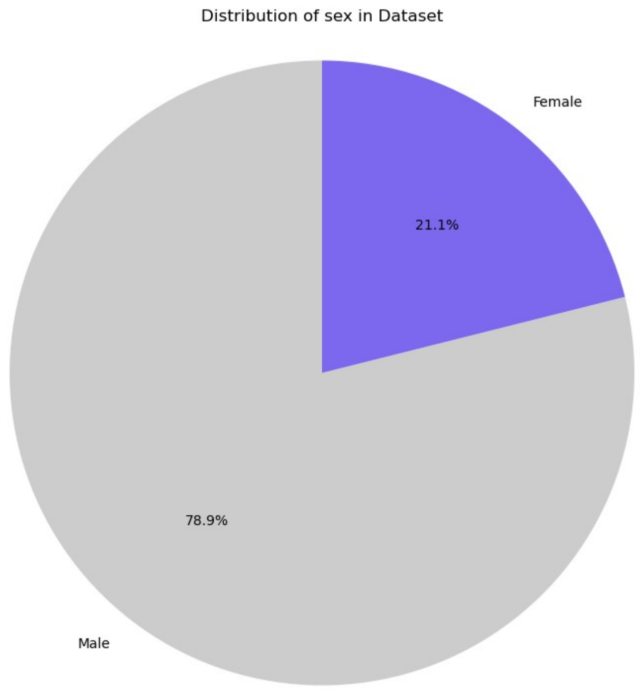
```
# Set the figure size to 16:9
plt.figure(figsize=(16, 9))

# Plot the mean, median and mode of each column using sns
sns.histplot(df["age"], kde=True)
plt.axvline(df["age"].mean(), color="r")
plt.axvline(df["age"].median(), color="g")
plt.axvline(df["age"].mode()[0], color="b")


# Print the mean, median and mode of each 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
```



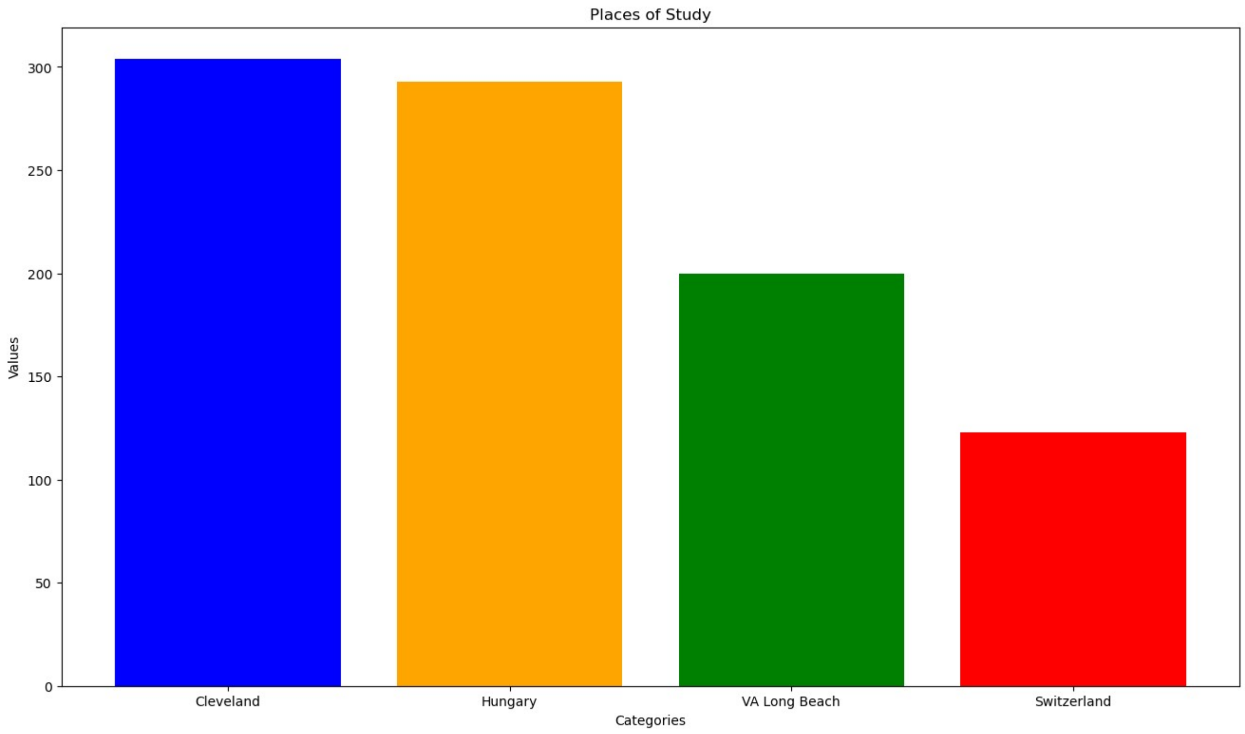
```
# find the count of male and female in sex column
df["sex"].value_counts()

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

# calculate percentage of male and female in sex column
male = 726
female = 194
total = male + female

# calculate percentage
male_percentage = (male/total) * 100
female_percentage = (female/total) * 100

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

# difference
difference_peercentage = ((male - female)/ female) * 100
print(f"Males are {difference_peercentage:.2f}% more than females in
the data.")

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

```
# Pie chart
labels = ['Male', 'Female']
sizes = [male_percentage, female_percentage]
colors = ['#CCCCCC','MediumSlateBlue']

fig1, ax1 = plt.subplots(figsize=(16, 9))
ax1.pie(sizes, colors=colors, labels=labels, autopct='%1.1f%%',
startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a
circle.

plt.title('Distribution of sex in Dataset')
plt.show()
```



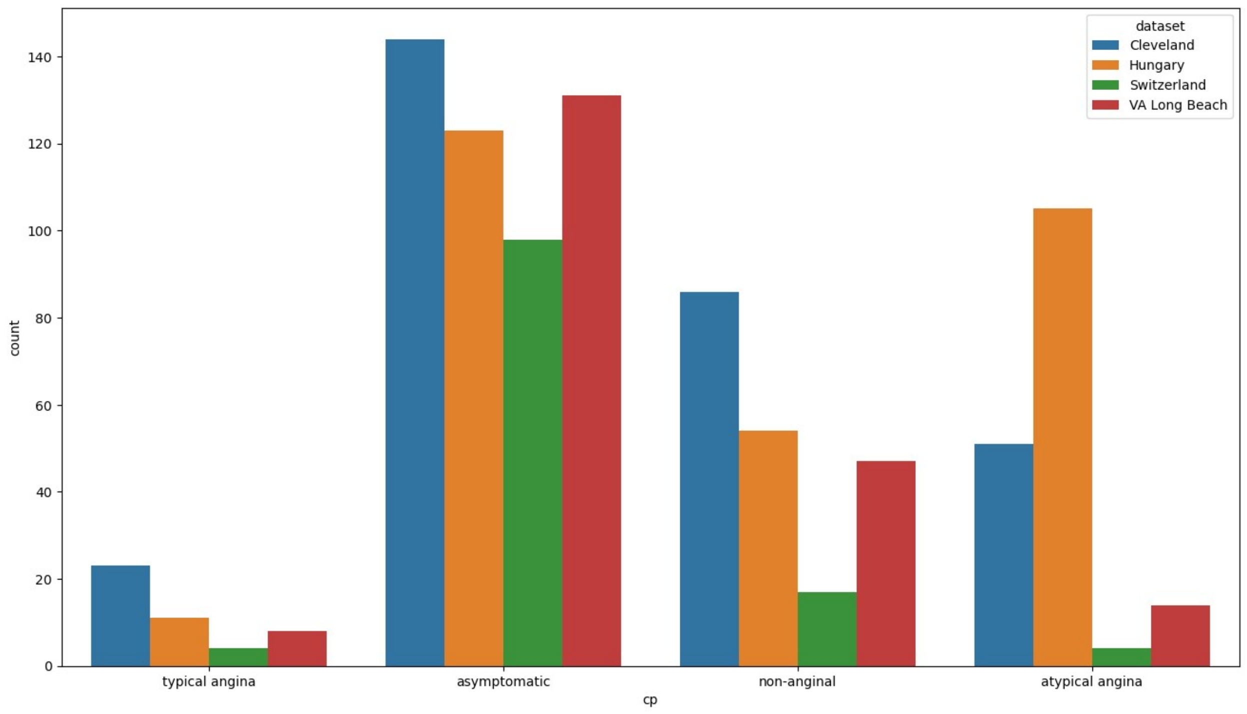
```
# let's find unique values count in dataset column
print("Unique values count in dataset
column:",df["dataset"].value_counts())

Unique values count in dataset column: dataset
Cleveland        304
Hungary          293
VA Long Beach    200
Switzerland      123
Name: count, dtype: int64

import matplotlib.pyplot as plt
```

```python
# Define the categories and their corresponding values
categories = ['Cleveland', 'Hungary', 'VA Long Beach', 'Switzerland']
values = [304, 293, 200, 123]
colors = ['blue', 'orange', 'green', 'red']

# Create a bar plot
plt.figure(figsize=(16, 9))
plt.bar(categories, values, color=colors)

# Add labels and title
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Places of Study')

# Show plot
plt.show()
```

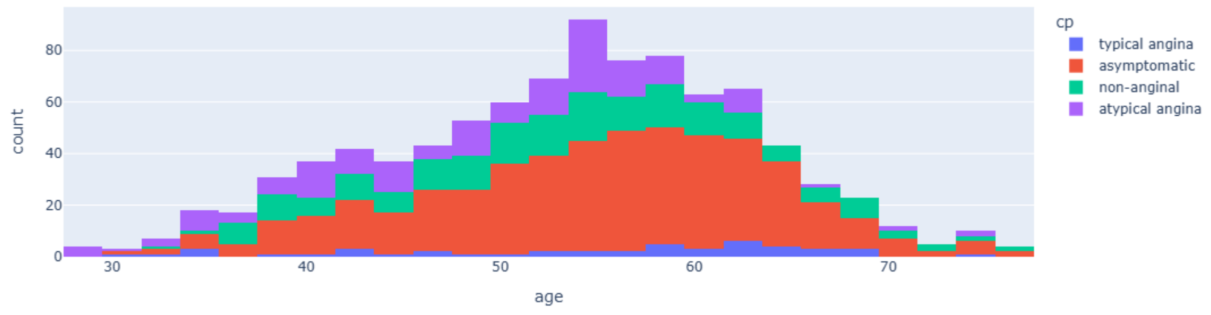


```python
print(df.groupby("sex")["dataset"].value_counts())
```

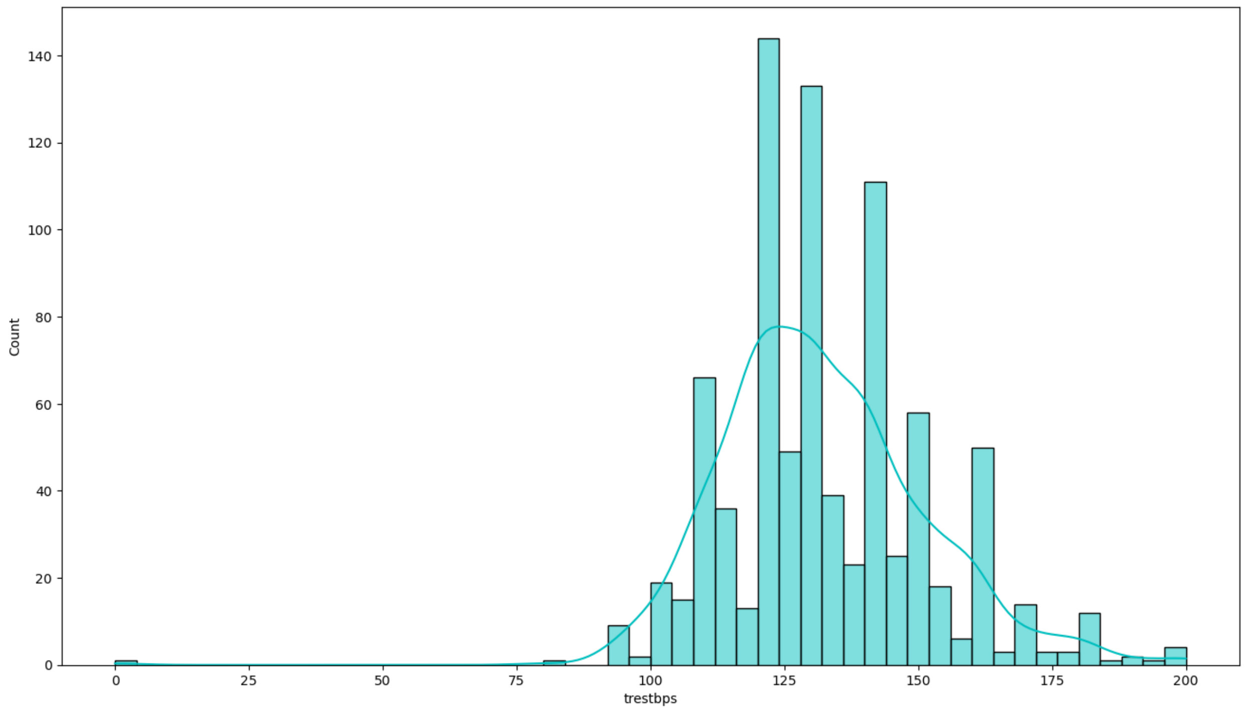
```
sex     dataset
Female  Cleveland         97
        Hungary           81
        Switzerland       10
        VA Long Beach      6
Male    Hungary          212
        Cleveland        207
        VA Long Beach    194
        Switzerland      113
Name: count, dtype: int64

# let's check value count of cp column
print("Value count of cp column:",df["cp"].value_counts())

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

# Set the figure size
plt.figure(figsize=(16, 9))

# Create the count plot
sns.countplot(data=df, x="cp", hue="dataset")

# Display the plot
plt.show()
```



```
# draw the plot of age column grouped by cp column using plotly
fig_3 = px.histogram(data_frame=df,x="age",color="cp")
fig_3.show()
```



```
# let's check summary of trestbps
df["trestbps"].describe()

count    861.000000
mean     132.132404
std       19.066070
min        0.000000
25%      120.000000
50%      130.000000
75%      140.000000
max      200.000000
Name: trestbps, dtype: float64

# Set the figure size to 16x9
plt.figure(figsize=(16, 9))

# Create a histplot of the trestbps column with a KDE overlay
sns.histplot(data=df, x="trestbps", kde=True, bins=50, color='c')

# Display the plot
plt.show()
```



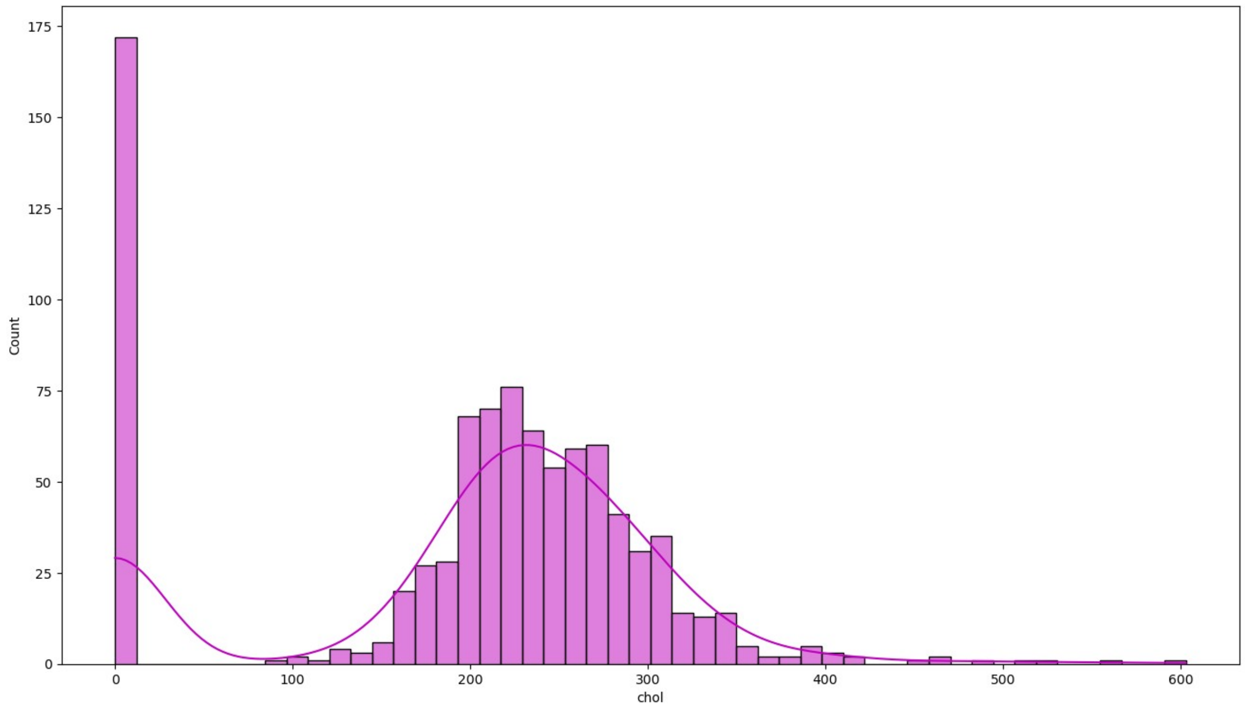
```
df["chol"].value_counts()
```

```
chol
0.0      172
220.0     10
254.0     10
223.0      9
230.0      9
        ...
360.0      1
412.0      1
358.0      1
321.0      1
385.0      1
Name: count, Length: 217, dtype: int64
```

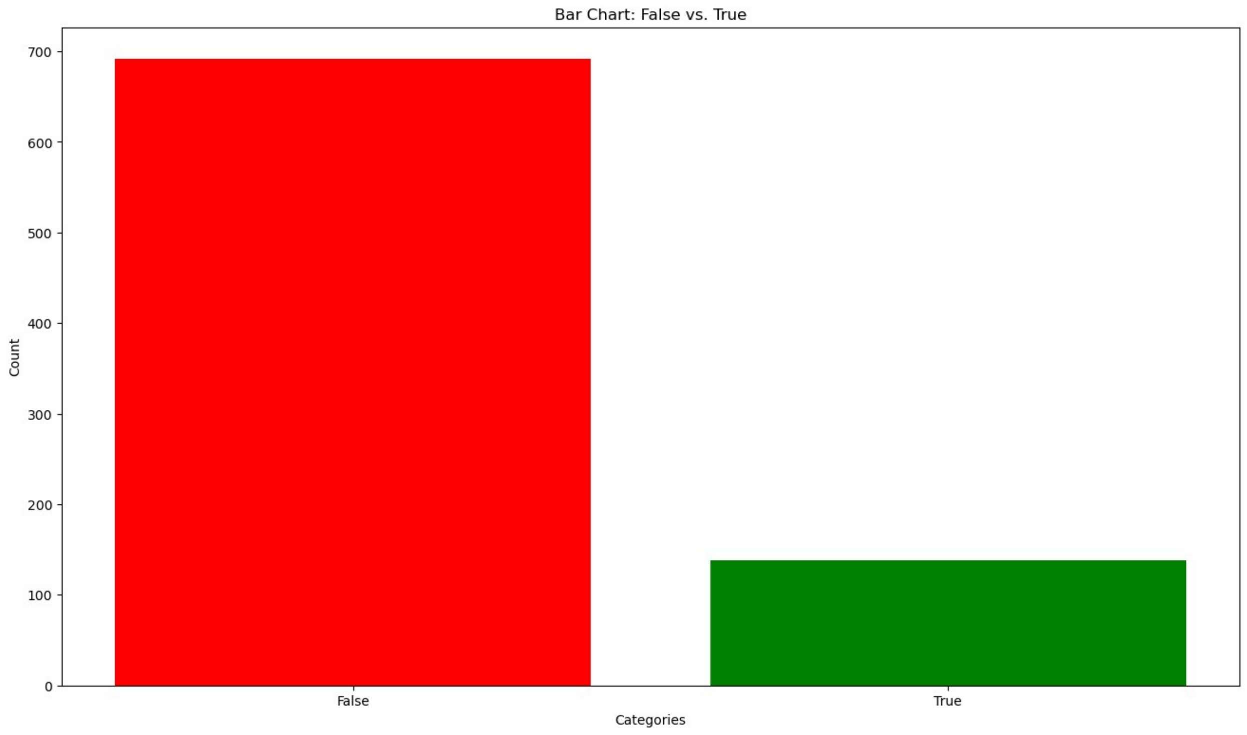
```
# let's check summary of chol
df["chol"].describe()
```

```
count    890.000000
mean     199.130337
std      110.780810
min        0.000000
25%      175.000000
50%      223.000000
75%      268.000000
max      603.000000
Name: chol, dtype: float64
```

```
# Set the figure size to 16x9
plt.figure(figsize=(16, 9))

# Create a histplot of the chol column with a KDE overlay
sns.histplot(data=df, x="chol", kde=True, bins=50, color='m')

# Display the plot
plt.show()
```



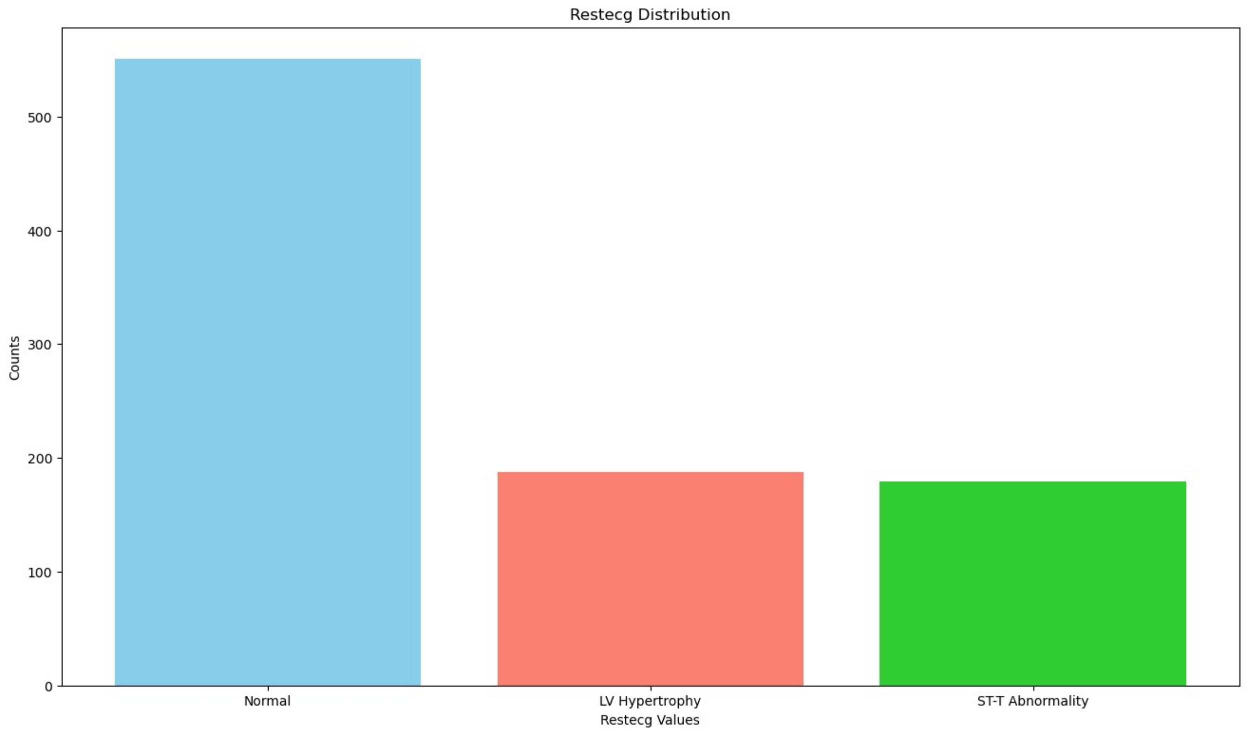
```
df["fbs"].value_counts()

fbs
False    692
True     138
Name: count, dtype: int64

categories = ['False', 'True']
counts = [692, 138]

# Create a bar chart
plt.figure(figsize=(16, 9))  # Set the plot size
plt.bar(categories, counts, color=['red', 'green'])  # Different
colors for True and False
plt.xlabel('Categories')
plt.ylabel('Count')
plt.title('Bar Chart: False vs. True')
plt.show()
```



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

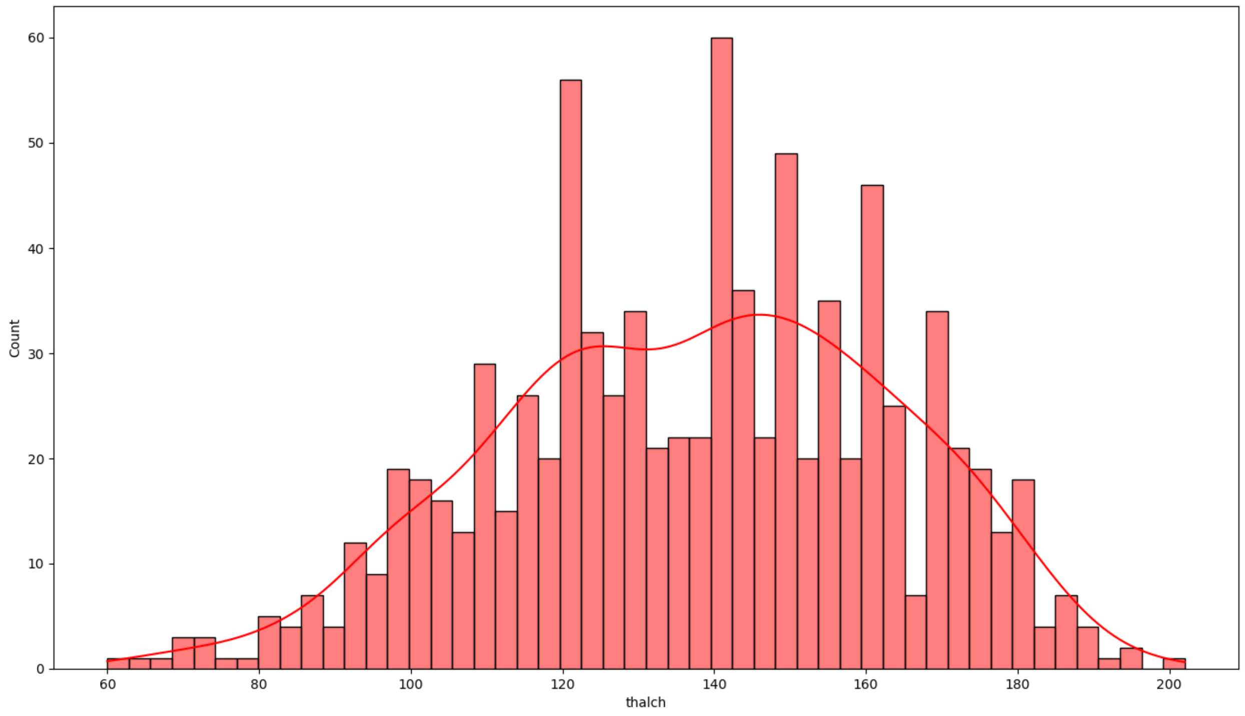
```
restecg
normal              551
lv hypertrophy      188
st-t abnormality    179
Name: count, dtype: int64
```

```
import matplotlib.pyplot as plt

values = ["Normal", "LV Hypertrophy", "ST-T Abnormality"]
counts = [551, 188, 179]

# Custom colors for bars
colors = ["skyblue", "salmon", "limegreen"]

# Create the bar chart
plt.figure(figsize=(16, 9))
plt.bar(values, counts, color=colors)
plt.xlabel("Restecg Values")
plt.ylabel("Counts")
plt.title("Restecg Distribution")
plt.show()
```



```
df["thalch"].value_counts()

thalch
150.0    43
140.0    41
120.0    35
130.0    30
160.0    26
         ..
195.0     1
91.0      1
87.0      1
192.0     1
73.0      1
Name: count, Length: 119, dtype: int64

# let's check summary of thalch
df["thalch"].describe()

count    865.000000
mean     137.545665
std       25.926276
min       60.000000
25%      120.000000
50%      140.000000
75%      157.000000
max      202.000000
Name: thalch, dtype: float64
```

```python
# Set the figure size to 16x9
plt.figure(figsize=(16, 9))

# Create a histplot of the thalch column with a KDE overlay
sns.histplot(data=df, x="thalch", kde=True, bins=50, color='r')

# Display the plot
plt.show()
```

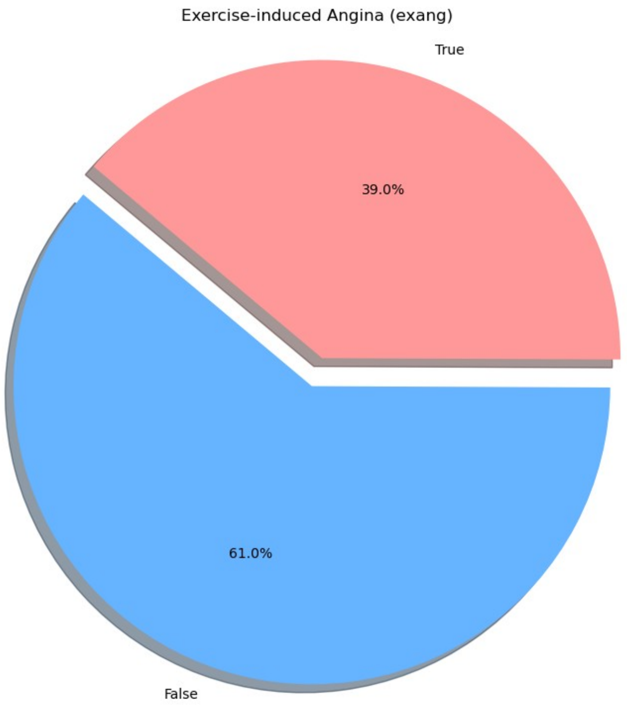


```python
df["exang"].value_counts()
```

```
exang
False    528
True     337
Name: count, dtype: int64
```

```python
import matplotlib.pyplot as plt

# Data
labels = ['False', 'True']
sizes = [528, 337]
colors = ['#66b3ff', '#ff9999']  # Awesome colors
explode = (0, 0.1)  # Slightly explode the 'True' slice for emphasis

# Create a pie chart
plt.figure(figsize=(16, 9))
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=140)
```

```
# Equal aspect ratio ensures that pie is drawn as a circle.
plt.axis('equal')

# Title
plt.title('Exercise-induced Angina (exang)')

# Show the plot
plt.show()
```

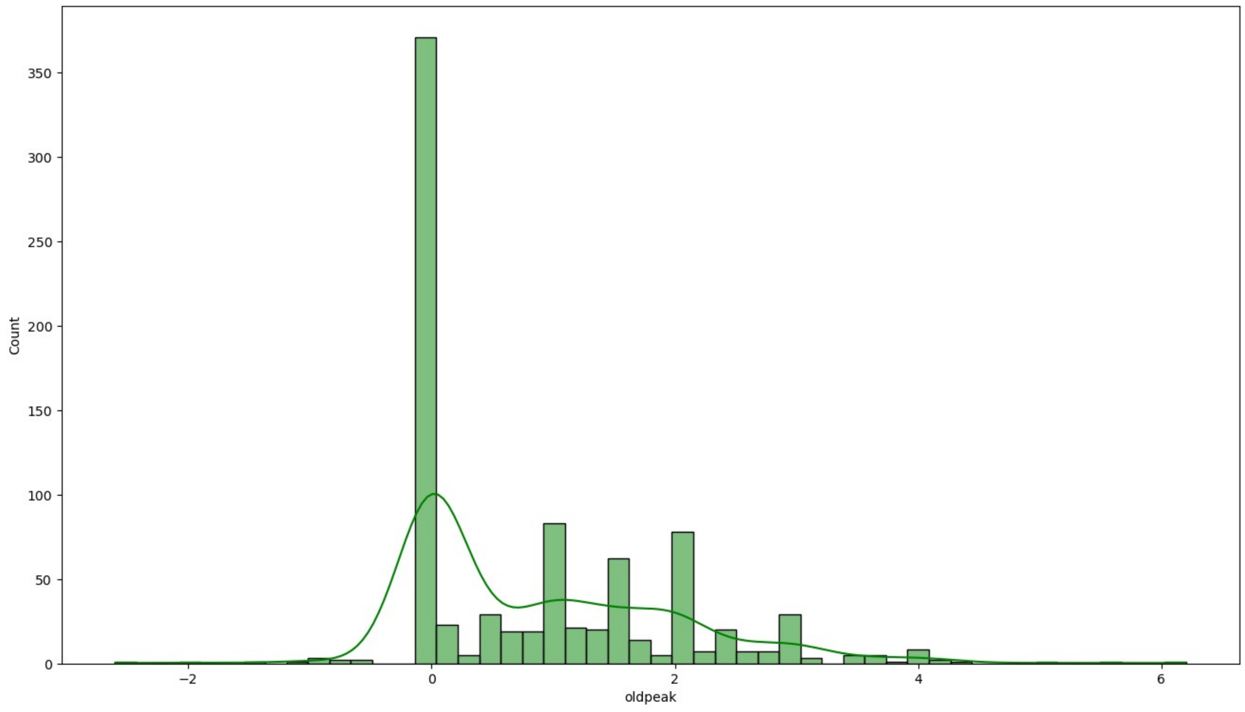


```
df["oldpeak"].value_counts()

oldpeak
 0.0    370
 1.0     83
 2.0     76
 1.5     48
 3.0     28
 0.5     19
 1.2     17
 2.5     16
 1.4     15
 0.8     15
 1.6     14
 0.2     14
 0.6     14
 1.8     12
 0.4     10
 0.1      9
 4.0      8
 2.6      7
 2.8      7
 1.3      5
 2.2      5
 0.7      5
 1.9      5
 0.3      5
 3.6      4
 2.4      4
 1.1      4
 0.9      4
 3.4      3
 1.7      2
-1.0      2
 4.2      2
 2.3      2
 2.1      2
-0.5      2
 3.2      2
 3.5      2
-0.8      1
-0.1      1
-0.9      1
-2.0      1
-0.7      1
-2.6      1
 6.2      1
-1.5      1
-1.1      1
 5.0      1
 4.4      1
 3.8      1
 2.9      1
 5.6      1
 3.1      1
 3.7      1
Name: count, dtype: int64
```

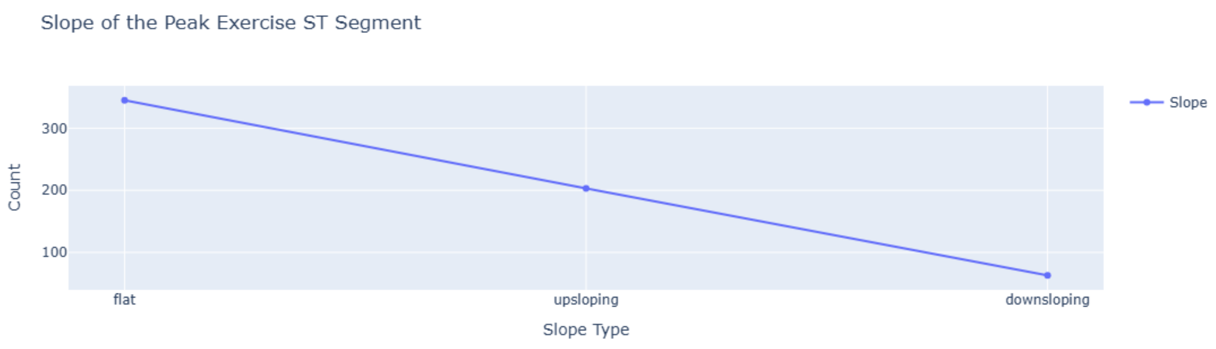
```
# let's check summary of oldpeak
df["oldpeak"].describe()
```

```
count    858.000000
mean       0.878788
std        1.091226
min       -2.600000
25%        0.000000
50%        0.500000
```

```
75%          1.500000
max          6.200000
Name: oldpeak, dtype: float64

# Set the figure size to 16x9
plt.figure(figsize=(16, 9))

# Create a histplot of the thalch column with a KDE overlay
sns.histplot(data=df, x="oldpeak", kde=True, bins=50, color='g')

# Display the plot
plt.show()
```

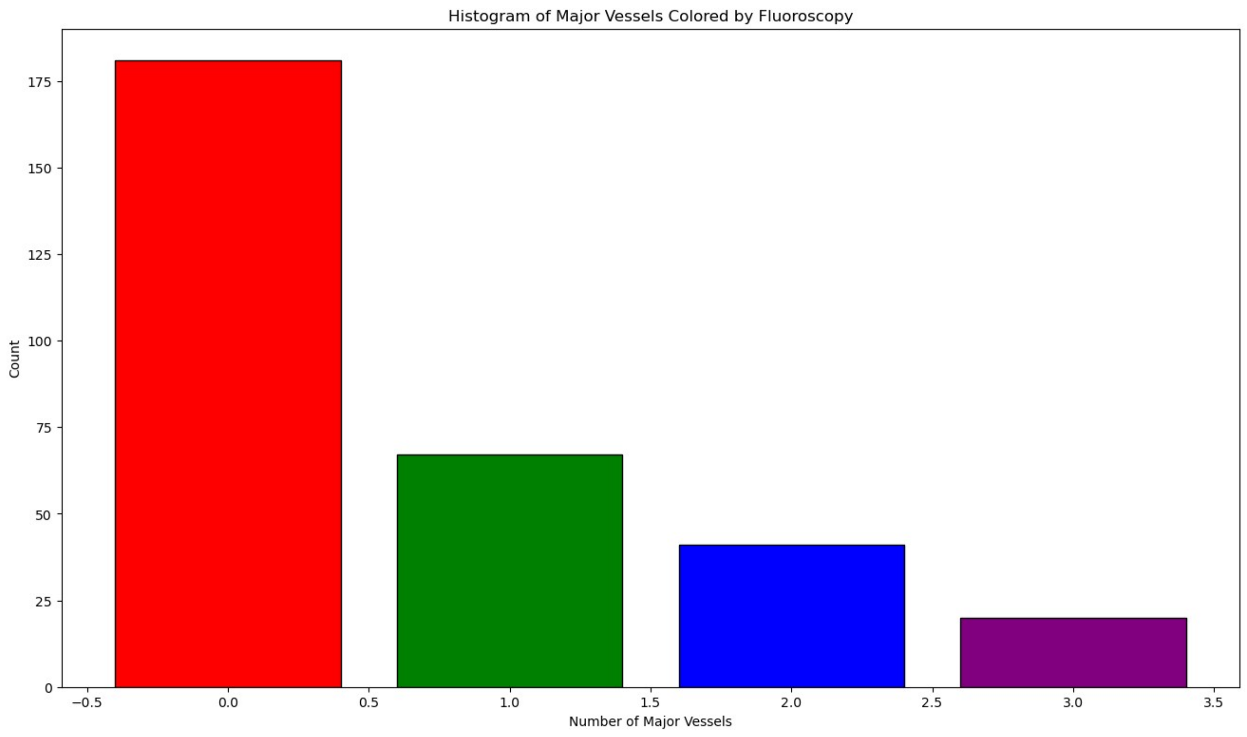


```
df["slope"].value_counts()

slope
flat           345
upsloping      203
downsloping     63
Name: count, dtype: int64

import plotly.graph_objects as go

# Data
slope_data = {
    "flat": 345,
    "upsloping": 203,
    "downsloping": 63
}

# Create the figure
fig = go.Figure()

# Add a scatter plot (line plot)
fig.add_trace(go.Scatter(x=list(slope_data.keys()),
y=list(slope_data.values()), mode='lines+markers', name='Slope'))

# Customize the plot
fig.update_layout(
    title="Slope of the Peak Exercise ST Segment",
    xaxis_title="Slope Type",
    yaxis_title="Count",
    showlegend=True
)

# Show the plot
fig.show()
```



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

ca
0.0    181
1.0     67
2.0     41
3.0     20
Name: count, dtype: int64

import matplotlib.pyplot as plt

# Data
categories = [0.0, 1.0, 2.0, 3.0]
counts = [181, 67, 41, 20]
colors = ['red', 'green', 'blue', 'purple']

# Create the figure with the specified size
```

```python
plt.figure(figsize=(16, 9))

# Create the histogram with specified colors
plt.bar(categories, counts, color=colors, edgecolor="black")

# Labels and title
plt.xlabel("Number of Major Vessels")
plt.ylabel("Count")
plt.title("Histogram of Major Vessels Colored by Fluoroscopy")

# Show the plot
plt.show()
```

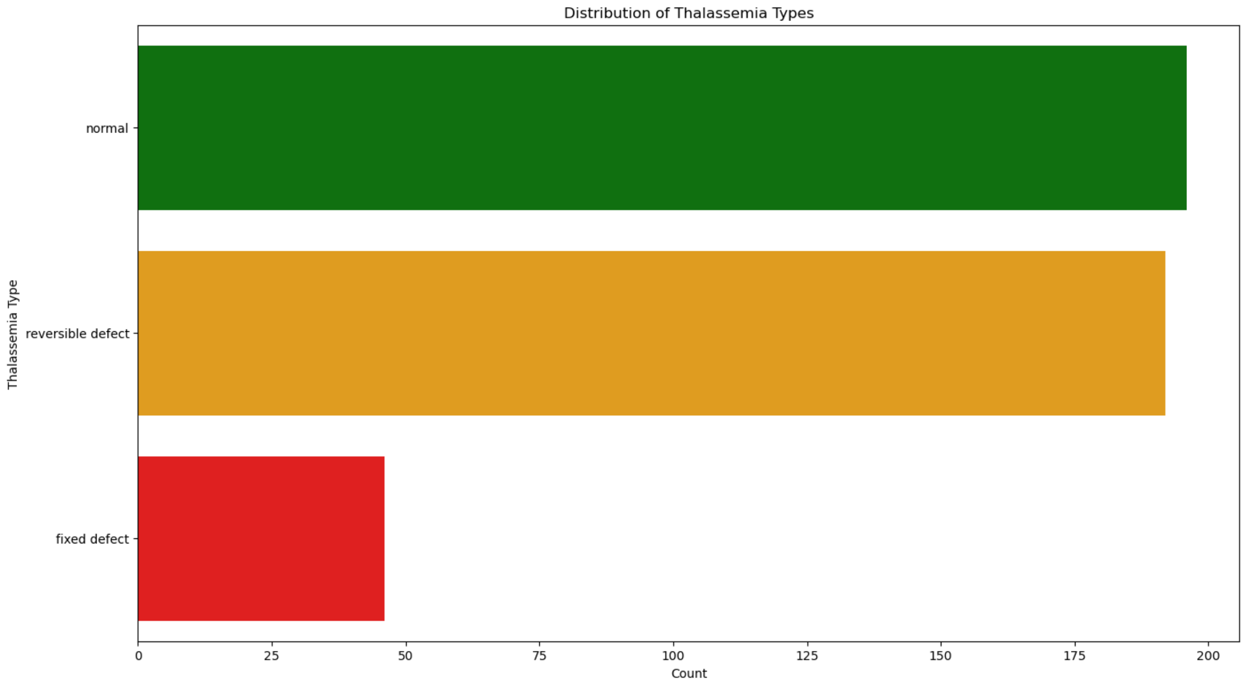


```python
df["thal"].value_counts()
```

```
thal
normal               196
reversable defect    192
fixed defect          46
Name: count, dtype: int64
```

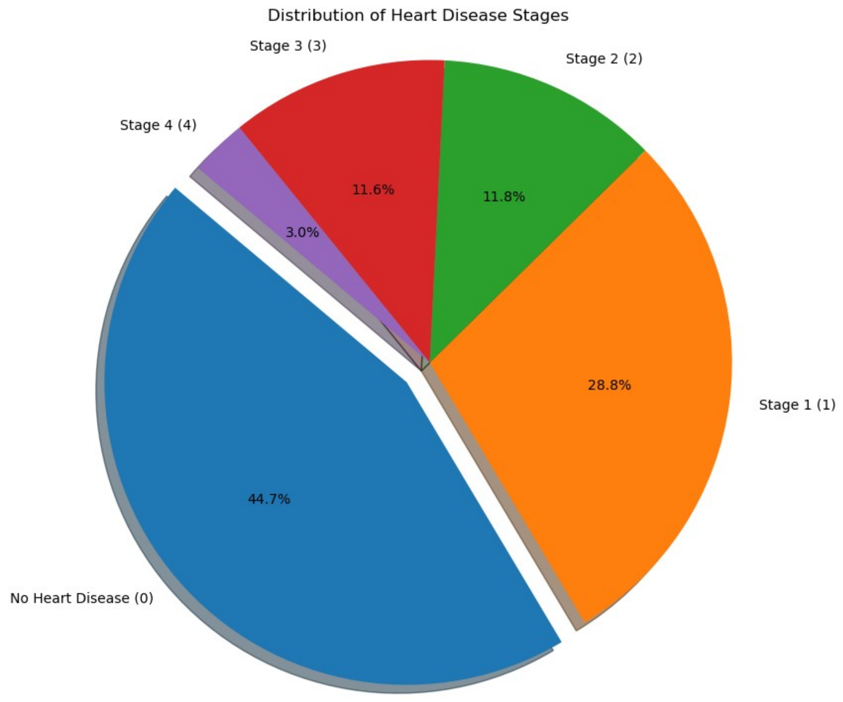
```python
import seaborn as sns
import matplotlib.pyplot as plt



# Data
```

```
thal_values = ['normal'] * 196 + ['reversible defect'] * 192 + ['fixed
defect'] * 46

# Plot
plt.figure(figsize=(16, 9))
sns.countplot(y=thal_values, palette=['green', 'orange', 'red'])
plt.xlabel('Count')
plt.ylabel('Thalassemia Type')
plt.title('Distribution of Thalassemia Types')
plt.show()
```

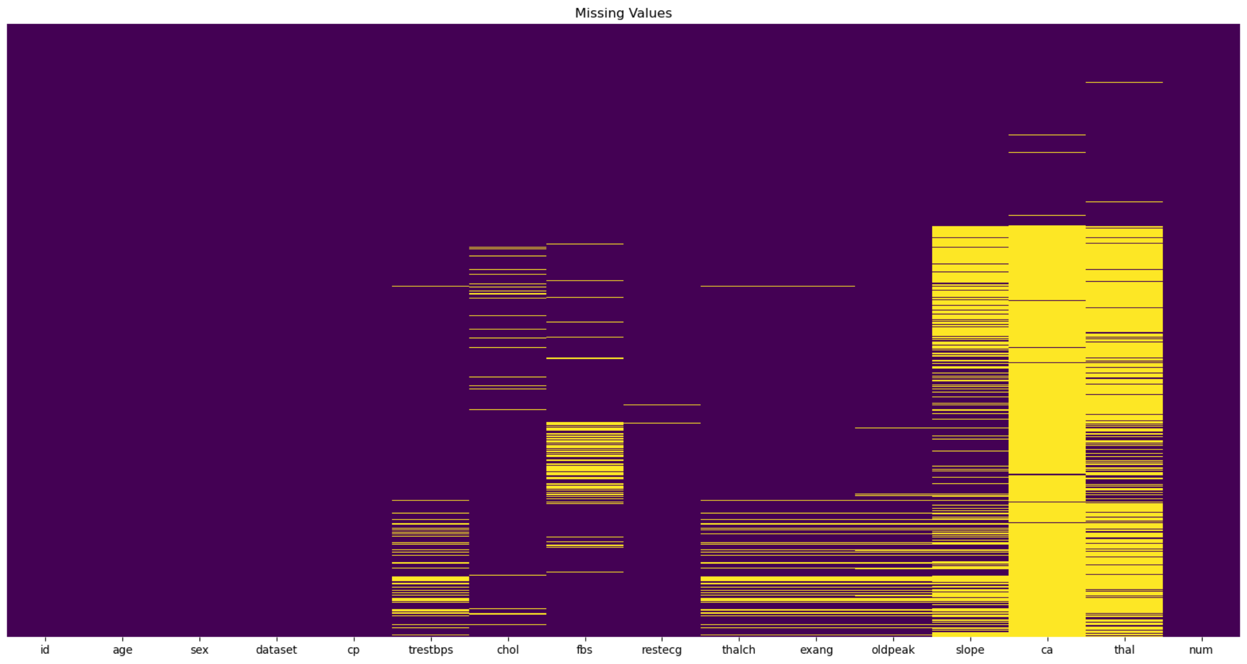


```
df["num"].value_counts()

num
0    411
1    265
2    109
3    107
4     28
Name: count, dtype: int64

import matplotlib.pyplot as plt

# Data
labels = ['No Heart Disease (0)', 'Stage 1 (1)', 'Stage 2 (2)', 'Stage
3 (3)', 'Stage 4 (4)']
sizes = [411, 265, 109, 107, 28]
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd']
```

```
explode = (0.1, 0, 0, 0, 0)  # explode the 1st slice (No Heart
Disease)

# Plot
plt.figure(figsize=(16, 9))
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=140)
plt.title('Distribution of Heart Disease Stages')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a
circle.
plt.show()
```



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

plt.figure(figsize=(20,10))
sns.heatmap(df.isnull(), yticklabels=False, cbar=False,
cmap='viridis')
plt.title('Missing Values')
plt.show()
```



```python
# Let check colmns with missing values
df.isnull().sum()
```

```
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
```

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

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

categorical_cols =
["thal","ca","exang","slope","restecg","fbs","cp","sex","num"]
bool_cols = ["fbs","exang"]
numeric_cols = ["oldpeak","thalch","chol","trestbps","age"]

from sklearn.ensemble import RandomForestClassifier,
RandomForestRegressor
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.metrics import accuracy_score, mean_absolute_error,
mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import pandas as pd

# Function to impute categorical missing data
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.name ==
'category':
            X[col] = label_encoder.fit_transform(X[col].astype(str))

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

    iterative_imputer =
IterativeImputer(estimator=RandomForestRegressor(random_state=42),
add_indicator=True)

    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_values)
            X[col] = imputed_values[:, 0]

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    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 * 100), 2), "accuracy\n")

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

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

    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_values)
            X[col] = imputed_values[:, 0]

    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})

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

    return df_combined[passed_col]


# Function to impute continuous missing data
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.name ==
'category':
            X[col] = label_encoder.fit_transform(X[col].astype(str))

    iterative_imputer =
IterativeImputer(estimator=RandomForestRegressor(random_state=42),
add_indicator=True)

    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_values)
            X[col] = imputed_values[:, 0]

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    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))
    print("RMSE =", mean_squared_error(y_test, y_pred, squared=False))
    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.name ==
'category':
            X[col] = label_encoder.fit_transform(X[col].astype(str))

    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_values)
            X[col] = imputed_values[:, 0]

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

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

    return df_combined[passed_col]
```

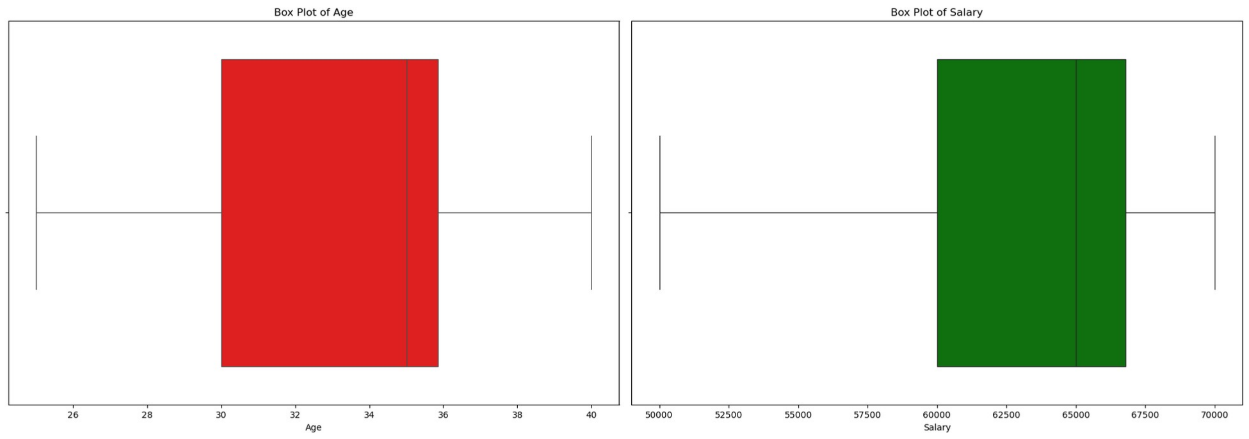
```
df = pd.read_csv("heart_disease_uci.csv")
df.isnull().sum().sort_values(ascending=False)
```

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

```
import pandas as pd
import numpy as np
import warnings
from sklearn.ensemble import RandomForestClassifier,
RandomForestRegressor
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, mean_absolute_error,
mean_squared_error, r2_score

warnings.filterwarnings('ignore')

# Sample DataFrame with missing values
df = pd.DataFrame({
    'Gender': ['Male', 'Female', np.nan, 'Female', 'Male'],
    'Age': [25, 30, 35, np.nan, 40],
    'Purchased': ['Yes', 'No', 'Yes', np.nan, 'No'],
    'Salary': [50000, 60000, np.nan, 65000, 70000],
    'Is_Employed': [True, False, np.nan, True, False]
})
```

```python
# Define column types
categorical_cols = ['Gender', 'Purchased']
numeric_cols = ['Age', 'Salary']
bool_cols = ['Is_Employed']

# Detect columns with missing values
missing_data_cols = [col for col in df.columns if
df[col].isnull().sum() > 0]

# Imputation functions
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.name ==
'category':
            X[col] = label_encoder.fit_transform(X[col].astype(str))

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

    iterative_imputer =
IterativeImputer(estimator=RandomForestRegressor(random_state=42),
add_indicator=True)
    for col in other_missing_cols:
        if X[col].isnull().sum() > 0:
            X[col] = iterative_imputer.fit_transform(X[[col]])[:, 0]

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
    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 * 100), 2), "accuracy\n")

    X = df_null.drop(passed_col, axis=1)
    for col in X.columns:
        if X[col].dtype == 'object' or X[col].dtype.name ==
'category':
            X[col] = label_encoder.fit_transform(X[col].astype(str))
    for col in other_missing_cols:
        if X[col].isnull().sum() > 0:
            X[col] = iterative_imputer.fit_transform(X[[col]])[:, 0]
    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})

    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.name ==
'category':
            X[col] = label_encoder.fit_transform(X[col].astype(str))

    iterative_imputer =
IterativeImputer(estimator=RandomForestRegressor(random_state=42),
add_indicator=True)
    for col in other_missing_cols:
        if X[col].isnull().sum() > 0:
            X[col] = iterative_imputer.fit_transform(X[[col]])[:, 0]

    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
    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))
    print("RMSE =", mean_squared_error(y_test, y_pred, squared=False))
    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.name ==
'category':
            X[col] = label_encoder.fit_transform(X[col].astype(str))
    for col in other_missing_cols:
        if X[col].isnull().sum() > 0:
            X[col] = iterative_imputer.fit_transform(X[[col]])[:, 0]
    if len(df_null) > 0:
        df_null[passed_col] = rf_regressor.predict(X)

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

# 🔄 Impute missing values
for col in missing_data_cols:
    print("Missing Values", col, ":",
str(round((df[col].isnull().sum() / len(df)) * 100, 2)) + "%")
    if col in categorical_cols:
        df[col] = impute_categorical_missing_data(col)
    elif col in numeric_cols:
        df[col] = impute_continuous_missing_data(col)

Missing Values Gender : 20.0%
The feature 'Gender' has been imputed with 0.0 accuracy

Missing Values Age : 20.0%
MAE = 4.5
RMSE = 4.5
R2 = nan

Missing Values Purchased : 20.0%
The feature 'Purchased' has been imputed with 0.0 accuracy

Missing Values Salary : 20.0%
MAE = 1250.0
RMSE = 1250.0
R2 = nan

Missing Values Is_Employed : 20.0%

# remove warning
import warnings
warnings.filterwarnings('ignore')

# impute missing values using our functions
for col in missing_data_cols:
    print("Missing Values", col, ":",
str(round((df[col].isnull().sum() / len(df)) * 100, 2))+"%")
    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 Gender : 0.0%
The feature 'Gender' has been imputed with 0.0 accuracy
```

```
Missing Values Age : 0.0%
MAE = 7.140999999999997
RMSE = 7.140999999999997
R2 = nan

Missing Values Purchased : 0.0%
The feature 'Purchased' has been imputed with 0.0 accuracy

Missing Values Salary : 0.0%
MAE = 3996.0
RMSE = 3996.0
R2 = nan

Missing Values Is_Employed : 20.0%

df.isnull().sum().sort_values(ascending=False)

Is_Employed    1
Gender         0
Age            0
Purchased      0
Salary         0
dtype: int64

import matplotlib.pyplot as plt
import seaborn as sns

# Make sure df and numeric_cols are already defined in your
environment

plt.figure(figsize=(20, 20))

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

for i, col in enumerate(numeric_cols):
    plt.subplot(3, 2, i + 1)  # Adjust layout based on number of plots
    sns.boxplot(x=df[col], color=colors[i % len(colors)])
    plt.title(f'Box Plot of {col}')

plt.tight_layout()
plt.show()
```



```python
# Show all column names to verify correct spelling
print("Available columns:", df.columns.tolist())

# Check if 'trestbps' column exists before proceeding
if 'trestbps' in df.columns:
    # Show rows where trestbps is 0
    print("Rows with trestbps = 0:")
    print(df[df['trestbps'] == 0])

    # Remove rows where trestbps is 0
    df = df[df['trestbps'] != 0]
    print("Updated DataFrame shape:", df.shape)
else:
    print("Column 'trestbps' not found in the DataFrame.")
    df.columns = df.columns.str.strip()
```

```
Available columns: ['Gender', 'Age', 'Purchased', 'Salary',
'Is_Employed']
Column 'trestbps' not found in the DataFrame.
```

```python
# Make a copy of our data

dataset = df.copy()

# List of columns to encode
categorical_columns_for_chi_squared_test = ['sex', 'cp', 'restecg',
'exang', 'slope', 'thal']

from sklearn.preprocessing import LabelEncoder

# Step 1: View all column names to debug
print("Available columns in dataset:", dataset.columns.tolist())

# Step 2: Safe label encoding
label_enc = LabelEncoder()
```

```python
for col in categorical_columns_for_chi_squared_test:
    if col in dataset.columns:
        dataset[col] =
label_enc.fit_transform(dataset[col].astype(str))
    else:
        print(f"Warning: Column '{col}' not found in dataset.
Skipping...")
```

```
Available columns in dataset: ['Gender', 'Age', 'Purchased', 'Salary',
'Is_Employed']
Warning: Column 'sex' not found in dataset. Skipping...
Warning: Column 'cp' not found in dataset. Skipping...
Warning: Column 'restecg' not found in dataset. Skipping...
Warning: Column 'exang' not found in dataset. Skipping...
Warning: Column 'slope' not found in dataset. Skipping...
Warning: Column 'thal' not found in dataset. Skipping...
```

```python
# Step 1: Check actual columns
print("Available columns in dataset:", dataset.columns.tolist())

# Step 2: Desired columns
desired_features = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope',
'thal', 'ca']
target_column = 'num'

# Step 3: Check for missing columns
missing_features = [col for col in desired_features + [target_column]
if col not in dataset.columns]

if missing_features:
    print(" Missing columns in dataset:", missing_features)
else:
    # Step 4: Define independent and target variables
    X1 = dataset[desired_features]
    y1 = dataset[target_column]
    print(" Feature matrix X1 and target y1 created successfully.")
```

```
Available columns in dataset: ['Gender', 'Age', 'Purchased', 'Salary',
'Is_Employed']
 Missing columns in dataset: ['sex', 'cp', 'fbs', 'restecg', 'exang',
'slope', 'thal', 'ca', 'num']
```

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.feature_selection import chi2

# Sample dataset with categorical variables
dataset = pd.DataFrame({
    'sex': ['male', 'female', 'female', 'male', 'male'],
    'cp': ['typical', 'asymptomatic', 'non-anginal', 'atypical',
'typical'],
    'fbs': ['yes', 'no', 'no', 'yes', 'no'],
    'restecg': ['normal', 'abnormal', 'normal', 'abnormal', 'normal'],
    'exang': ['no', 'yes', 'no', 'yes', 'no'],
    'slope': ['up', 'flat', 'down', 'flat', 'up'],
    'thal': ['normal', 'fixed', 'reversible', 'fixed', 'normal'],
    'ca': ['0', '2', '1', '3', '0'],
    'num': [0, 1, 1, 0, 0]  # target variable
})

# Define the features and target
feature_cols = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope',
'thal', 'ca']
target_col = 'num'

# Label encode categorical features
label_enc = LabelEncoder()
for col in feature_cols:
    dataset[col] = label_enc.fit_transform(dataset[col])

# Define X1 and y1
X1 = dataset[feature_cols]
y1 = dataset[target_col]

# Scale features (Chi-square requires non-negative values)
scaler = MinMaxScaler()
X1_scaled = scaler.fit_transform(X1)

# Perform Chi-Square test
chi_scores = chi2(X1_scaled, y1)

# Create and display score DataFrame
chi2_scores = pd.DataFrame({
    "Feature": feature_cols,
    "Score": chi_scores[0],
    "P-Value": chi_scores[1]
})

# Sort and print
chi2_scores = chi2_scores.sort_values(by="Score", ascending=False)
print(chi2_scores)
```

```
   Feature     Score   P-Value
0      sex  2.000000  0.157299
2      fbs  1.333333  0.248213
5    slope  0.680556  0.409395
1       cp  0.395062  0.529651
4    exang  0.083333  0.772830
6     thal  0.083333  0.772830
```

```
7       ca  0.083333  0.772830
3  restecg  0.055556  0.813664
```

```
# Step 1: Check actual columns in your dataset
print("Available columns:", dataset.columns.tolist())

# Step 2: Desired numeric columns for F-test
expected_numeric_cols = ['age', 'trestbps', 'chol', 'thalch',
'oldpeak']

# Step 3: Filter only available ones
available_numeric_cols = [col for col in expected_numeric_cols if col
in dataset.columns]
missing_cols = [col for col in expected_numeric_cols if col not in
dataset.columns]

# Step 4: Handle missing case
if missing_cols:
    print(" Missing columns:", missing_cols)

# Step 5: Select the columns that exist
X_num_for_f_test = dataset[available_numeric_cols]

# Optional: Show the selected DataFrame
print(" Selected columns for F-test:\n", X_num_for_f_test.head())
```

```
Available columns: ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope',
'thal', 'ca', 'num']
 Missing columns: ['age', 'trestbps', 'chol', 'thalch', 'oldpeak']
 Selected columns for F-test:
 Empty DataFrame
Columns: []
Index: [0, 1, 2, 3, 4]
```

```
# Defining the target column
y_for_f_test = dataset['num']

import pandas as pd
import numpy as np
from sklearn.feature_selection import f_classif

# Step 1: Define or normalize your dataset column names if needed
# dataset.columns = dataset.columns.str.strip().str.lower()

# Step 2: List of expected numeric features
expected_numeric_cols = ['age', 'trestbps', 'chol', 'thalch',
'oldpeak']
target_col = 'num'

# Step 3: Check which columns are available
available_numeric_cols = [col for col in expected_numeric_cols if col
in dataset.columns]
missing_cols = [col for col in expected_numeric_cols if col not in
dataset.columns]
if missing_cols:
    print(" Missing numeric columns:", missing_cols)

# Step 4: Extract X and y
X_num_for_f_test = dataset[available_numeric_cols]
y_for_f_test = dataset[target_col] if target_col in dataset.columns
else None

# Step 5: Check and perform F-test
if y_for_f_test is not None and not X_num_for_f_test.empty:
    f_scores, p_values = f_classif(X_num_for_f_test, y_for_f_test)

    # Step 6: Display results
    f_test_df = pd.DataFrame({
        "Feature": available_numeric_cols,
        "F-Score": f_scores,
        "P-Value": p_values
    }).sort_values(by="F-Score", ascending=False)

    print(f_test_df)
else:
    print(" Target column 'num' not found or no valid numeric
features.")
```

```
 Missing numeric columns: ['age', 'trestbps', 'chol', 'thalch',
'oldpeak']
 Target column 'num' not found or no valid numeric features.
```

```
import pandas as pd
import numpy as np
from sklearn.feature_selection import f_classif

# Sample dataset (replace this with your actual dataset if needed)
# This is just for demonstration:
dataset = pd.DataFrame({
    'age': [63, 67, 67, 37, 41],
    'trestbps': [145, 160, 120, 130, 130],
    'chol': [233, 286, 229, 250, 204],
    'thalch': [150, 108, 129, 187, 172],
    'oldpeak': [2.3, 1.5, 2.6, 3.5, 1.4],
    'num': [1, 1, 1, 0, 0]  # Target column
})

# Step 1: Define the numeric feature columns and target column
numeric_cols = ['age', 'trestbps', 'chol', 'thalch', 'oldpeak']
target_col = 'num'
```

```
# Step 2: Check if target and features exist in dataset
existing_cols = [col for col in numeric_cols if col in
dataset.columns]
missing_cols = [col for col in numeric_cols if col not in
dataset.columns]

if not existing_cols:
    print(" None of the numeric columns found in dataset.")
elif target_col not in dataset.columns:
    print(f" Target column '{target_col}' not found.")
else:
    # Step 3: Prepare input X and output y
    X_num_for_f_test = dataset[existing_cols]
    y_for_f_test = dataset[target_col]

    # Step 4: Ensure no NaNs (F-test does not allow missing values)
    if X_num_for_f_test.isnull().any().any() or
y_for_f_test.isnull().any():
        print(" Dataset contains missing values. Please handle them
before F-test.")
    else:
        # Step 5: Perform F-test
        f_scores, p_values = f_classif(X_num_for_f_test, y_for_f_test)

        # Step 6: Display F-scores
        f_scores_df = pd.DataFrame({
            'Feature': existing_cols,
            'F-Score': f_scores,
            'P-Value': p_values
        }).sort_values(by='F-Score', ascending=False)

        print(f_scores_df)
```

```
    Feature     F-Score   P-Value
0       age  137.142857  0.001338
3    thalch    9.231674  0.055939
1  trestbps    0.600000  0.495025
2      chol    0.582483  0.500869
4   oldpeak    0.126593  0.745540
```

```
df.columns
```

```
Index(['Gender', 'Age', 'Purchased', 'Salary', 'Is_Employed'],
dtype='object')
```

```
final_features = ['ca', 'cp', 'exang', 'slope', 'thal', 'oldpeak',
'thalch', 'age', 'chol', 'trestbps']


    # Step 4: Create final dataset using only available columns
    final_dataset = dataset[existing_features + ['num']]
    print(" Final dataset created with shape:", final_dataset.shape)
    print(final_dataset.head())
```

```
 Final dataset created with shape: (5, 1)
   num
0    1
1    1
2    1
3    0
4    0
```

```
final_dataset.head()
```

```
   num
0    1
1    1
2    1
3    0
4    0
```

```
final_dataset = df
```

```
# Check if 'num' column exists in the DataFrame
if 'num' in df.columns:
    print(" 'num' column value counts:")
    print(df["num"].value_counts())
else:
    print(" 'num' column not found in DataFrame.")
    print("Available columns are:", df.columns.tolist())
```

```
 'num' column not found in DataFrame.
Available columns are: ['Gender', 'Age', 'Purchased', 'Salary',
'Is_Employed']
```

```
# Check if 'num' column exists in df
if 'num' in df.columns:
    # Split data into X and y
    X = df.drop(['num'], axis=1)
    y = df['num']
    print(" Data split into X and y.")
else:
    print(" Column 'num' not found in df.")
    print("Available columns are:", df.columns.tolist())
```

```
 Column 'num' not found in df.
Available columns are: ['Gender', 'Age', 'Purchased', 'Salary',
'Is_Employed']
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Step 1: Print column names to help debug
print("□ Columns in df:", df.columns.tolist())

# Step 2: Define the correct target column name
target_col = 'num'  # change this if your actual column is named
differently

# Step 3: Check if target column exists
if target_col in df.columns:
    # Split into features and target
    X = df.drop([target_col], axis=1)
    y = df[target_col]

# Step 4: Label Encode all categorical columns with separate encoders
label_encoders = {}  # Store each encoder

for col in X.columns:
    if X[col].dtype == 'object' or X[col].dtype.name == 'category':
        le = LabelEncoder()
        X[col] = le.fit_transform(X[col].astype(str))  # Ensure all
are strings
        label_encoders[col] = le

# Step 5: Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

#  Summary
print(" Data encoded and split into train/test successfully.")
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
```

```
□ Columns in df: ['Gender', 'Age', 'Purchased', 'Salary',
'Is_Employed']
 Data encoded and split into train/test successfully.
X_train shape: (455, 30)
X_test shape: (114, 30)
```

```
# import all models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.naive_bayes import GaussianNB
```

```
# from lightgbm import LGBMClassifier

# impot pipeline
from sklearn.pipeline import Pipeline

# import metrics
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

# □ Required Imports
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier, AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_breast_cancer  # Example dataset

import pandas as pd

# □ Load a sample dataset (you can replace this with your own
DataFrame)
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)

# □ Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# □ List of models to evaluate
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,
max_iter=1000)),
    ('K-Nearest Neighbors', KNeighborsClassifier()),
    ('Decision Tree', DecisionTreeClassifier(random_state=42)),
    ('Ada Boost', AdaBoostClassifier(random_state=42)),
    ('XG Boost', XGBClassifier(random_state=42,
use_label_encoder=False, eval_metric='logloss')),
    ('Naive Bayes', GaussianNB())
]

best_model = None
best_accuracy = 0.0

# □ Iterate over the models and evaluate their performance
for name, model in models:
    pipeline = Pipeline([
        ('model', model)
    ])

    # Perform cross-validation
    scores = cross_val_score(pipeline, X_train, y_train, cv=5)

    # Calculate mean accuracy
    mean_accuracy = scores.mean()

    # Fit the pipeline on the training data
    pipeline.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = pipeline.predict(X_test)

    # Calculate accuracy score
    accuracy = accuracy_score(y_test, y_pred)

    # Print the performance metrics
    print("Model:", name)
    print("Cross-validation Accuracy:", round(mean_accuracy, 4))
    print("Test Accuracy:", round(accuracy, 4))
    print()

    # Check if the current model has the best accuracy
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_model = (name, pipeline)

# □ Retrieve the best model
print("□ Best Model:", best_model[0], "with Test Accuracy:",
round(best_accuracy, 4))
```

```
Model: Random Forest
Cross-validation Accuracy: 0.9582
Test Accuracy: 0.9649

Model: Gradient Boosting
Cross-validation Accuracy: 0.9516
Test Accuracy: 0.9561

Model: Support Vector Machine
```

```
Cross-validation Accuracy: 0.9033
Test Accuracy: 0.9474

Model: Logistic Regression
Cross-validation Accuracy: 0.9495
Test Accuracy: 0.9561

Model: K-Nearest Neighbors
Cross-validation Accuracy: 0.9231
Test Accuracy: 0.9561

Model: Decision Tree
Cross-validation Accuracy: 0.9165
Test Accuracy: 0.9474

Model: Ada Boost
Cross-validation Accuracy: 0.9648
Test Accuracy: 0.9737

Model: XG Boost
Cross-validation Accuracy: 0.9648
Test Accuracy: 0.9561

Model: Naive Bayes
Cross-validation Accuracy: 0.9341
Test Accuracy: 0.9737

□ Best Model: Ada Boost with Test Accuracy: 0.9737
```