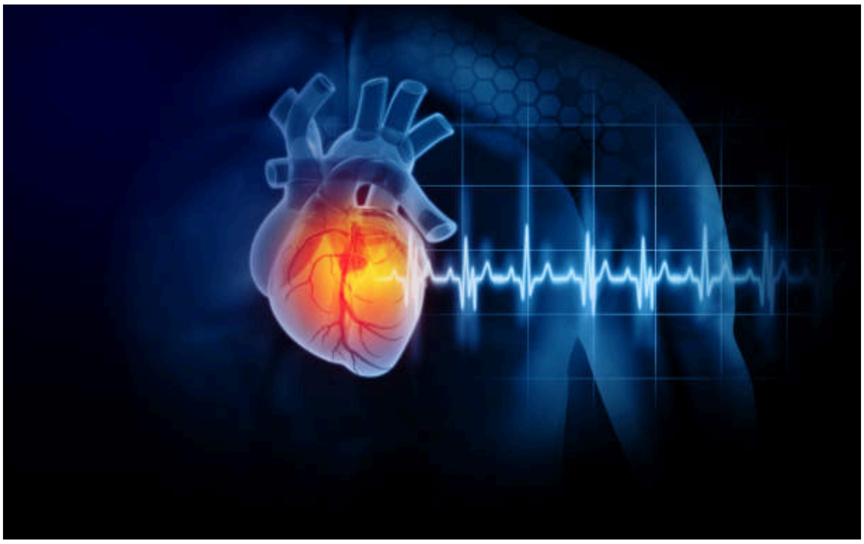
Project Title: Heart Disease Diagnostic Analysis



Import necessary libraries

In [1]:
```
# Import necessary libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Loading Dataset

In [2]:
```
# Loading Dataset

Heart_Disease = pd.read_csv('C:\\Users\\stati\\OneDrive\\Desktop\\Heart Disease data.csv')

Heart_Disease
```

Out[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

1025 rows × 14 columns

EDA (Exploratory Data Analysis)

In [3]:
```
Heart_Disease.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

In [4]:
```
Heart_Disease.isnull().sum()
```

Out[4]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

In [5]:
```
Heart_Disease.duplicated().sum()
```

Out[5]:
```
723
```

In [6]:
```
Heart_Disease.drop_duplicates(inplace=True)
```

In [7]:
```
Heart_Disease.duplicated().sum()
```

Out[7]:
```
0
```

In [8]:
```
Heart_Disease.shape
```

Out[8]:
```
(302, 14)
```

In [9]:
```
Heart_Disease.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 302 entries, 0 to 878
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       302 non-null    int64  
 1   sex       302 non-null    int64  
 2   cp        302 non-null    int64  
 3   trestbps  302 non-null    int64  
 4   chol      302 non-null    int64  
 5   fbs       302 non-null    int64  
 6   restecg   302 non-null    int64  
 7   thalach   302 non-null    int64  
 8   exang     302 non-null    int64  
 9   oldpeak   302 non-null    float64
 10  slope     302 non-null    int64  
 11  ca        302 non-null    int64  
 12  thal      302 non-null    int64  
 13  target    302 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 35.4 KB
```

In [10]:
```
Heart_Disease.describe().T
```

Out[10]:

	count	mean	std	min	25%	50%	75%	max
age	302.0	54.420530	9.047970	29.0	48.00	55.5	61.00	77.0
sex	302.0	0.682119	0.466426	0.0	0.00	1.0	1.00	1.0
cp	302.0	0.963576	1.032044	0.0	0.00	1.0	2.00	3.0
trestbps	302.0	131.602649	17.563394	94.0	120.00	130.0	140.00	200.0
chol	302.0	246.500000	51.753489	126.0	211.00	240.5	274.75	564.0
fbs	302.0	0.149007	0.356686	0.0	0.00	0.0	0.00	1.0
restecg	302.0	0.526490	0.526027	0.0	0.00	1.0	1.00	2.0
thalach	302.0	149.569536	22.903527	71.0	133.25	152.5	166.00	202.0
exang	302.0	0.327815	0.470196	0.0	0.00	0.0	1.00	1.0
oldpeak	302.0	1.043046	1.161452	0.0	0.00	0.8	1.60	6.2
slope	302.0	1.397351	0.616274	0.0	1.00	1.0	2.00	2.0
ca	302.0	0.718543	1.006748	0.0	0.00	0.0	1.00	4.0
thal	302.0	2.314570	0.613026	0.0	2.00	2.0	3.00	3.0
target	302.0	0.543046	0.498970	0.0	0.00	1.0	1.00	1.0

In [11]:

```
Heart_Disease.target.value_counts()
```

Out[11]:

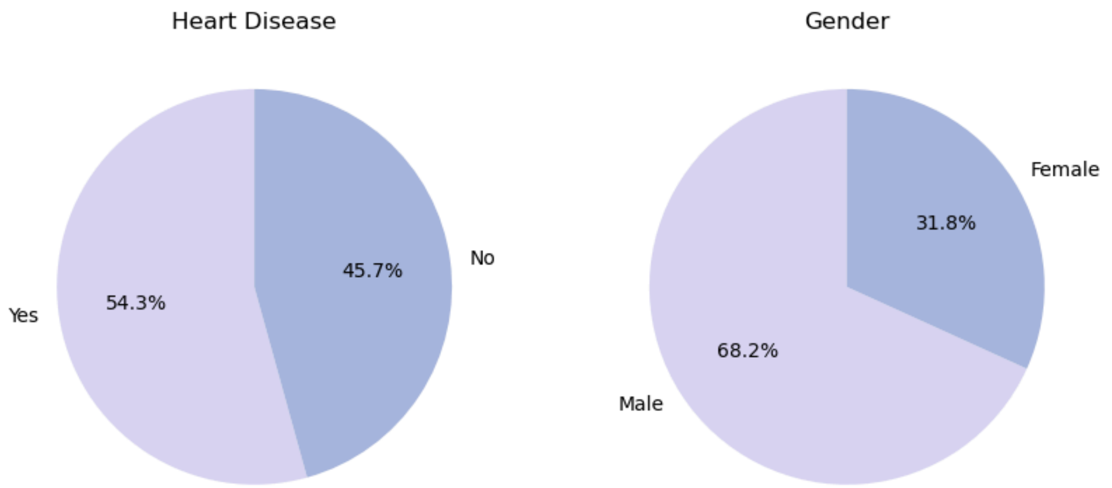
```
target
1    164
0    138
Name: count, dtype: int64
```

Visualization

Pie charts for Heart Disease Distribution and Gender Distribution

In [12]:

```
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

# Heart Disease Distribution
axs[0].pie(Heart_Disease.target.value_counts(),
           labels=['Yes', 'No'], colors=sns.cubehelix_palette(start=2),
           autopct='%1.1f%%', startangle=90)
axs[0].set_title("Heart Disease")

# Gender Distribution
axs[1].pie(Heart_Disease.sex.value_counts(),
           labels=['Male', 'Female'], colors=sns.cubehelix_palette(start=2),
           autopct='%1.1f%%', startangle=90)
axs[1].set_title("Gender")

[ax.axis('off') for ax in axs]
plt.show()
```

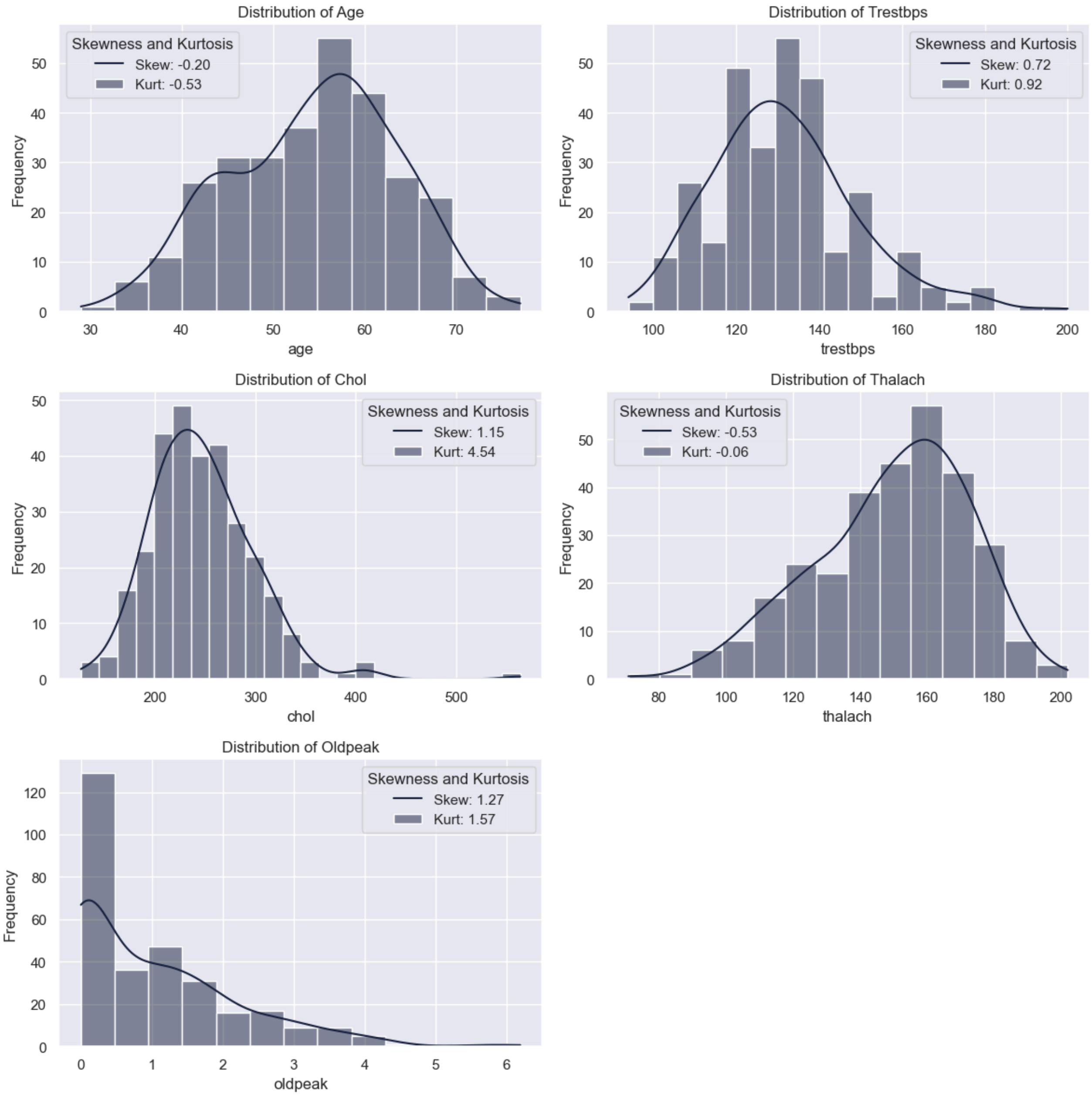


Distribution Analysis of Numerical Data

In [13]:

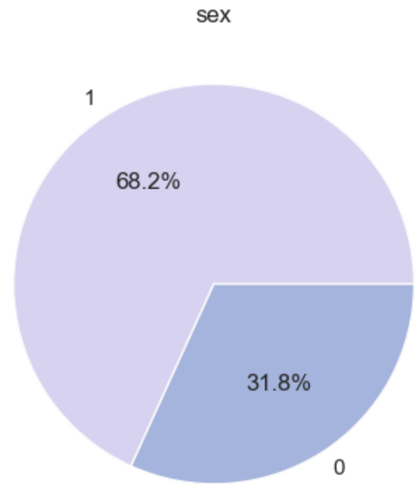
```
sns.set(style="darkgrid")
fig, axes = plt.subplots(3, 2, figsize=(12, 12))

Numerical_feature = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
```

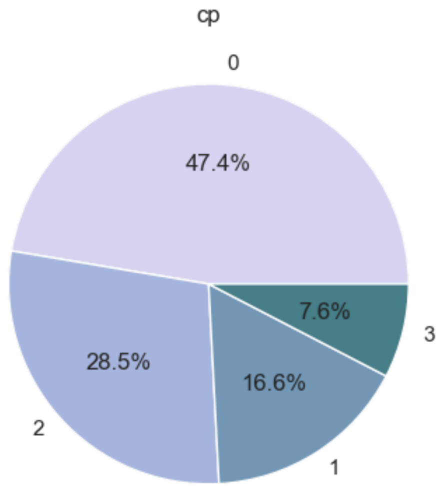
```
for ax, Numerical_feature in zip(axes.flatten(),Numerical_feature ):
    sns.histplot(Heart_Disease[Numerical_feature], kde=True, color=sns.color_palette("cubehelix")[0], ax=ax)
    ax.set_title(f'Distribution of {Numerical_feature.capitalize()}')
    ax.set(xlabel=Numerical_feature, ylabel='Frequency')
    ax.legend([f"Skew: {Heart_Disease[Numerical_feature].skew():.2f}",
               f"Kurt: {Heart_Disease[Numerical_feature].kurt():.2f}"], title="Skewness and Kurtosis")

# Remove the last subplot as it's unused
fig.delaxes(axes.flatten()[-1])

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

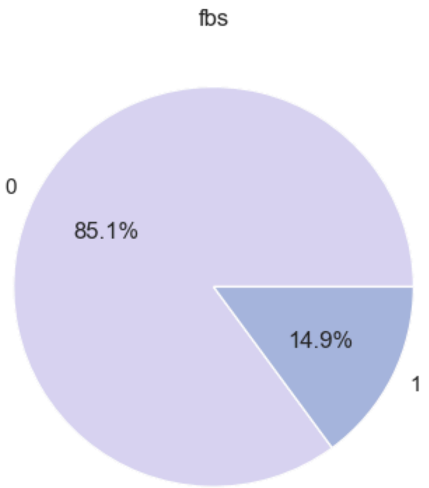


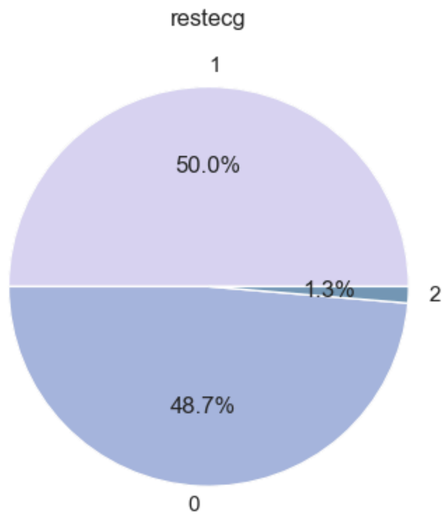
Pie Chart of Categorical Data

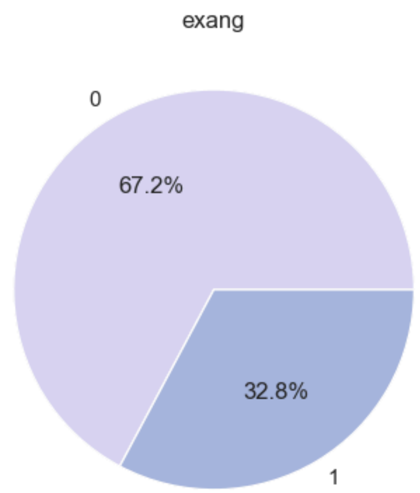
In [14]:
```
Categorical_Features = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal']

Cat_features = len(Categorical_Features)
Cat_rows, Cat_cols = (Cat_features + 1) // 2, min(2, Cat_features)

fig, axes = plt.subplots(Cat_rows, Cat_cols, figsize=(10, 4*Cat_rows))

for i, feature in enumerate(Categorical_Features):
    r, c = i // Cat_cols, i % Cat_cols
    values = Heart_Disease[feature].value_counts()
    axes[r, c].pie(values, labels=values.index, autopct='%1.1f%%', colors=sns.cubehelix_palette(start=2))
    axes[r, c].set_title(feature)

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

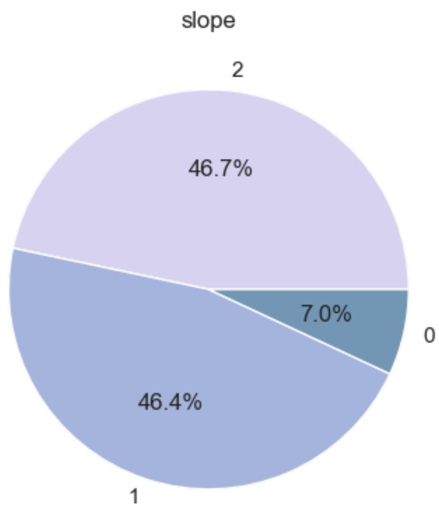

sex
1
68.2%
31.8%
0

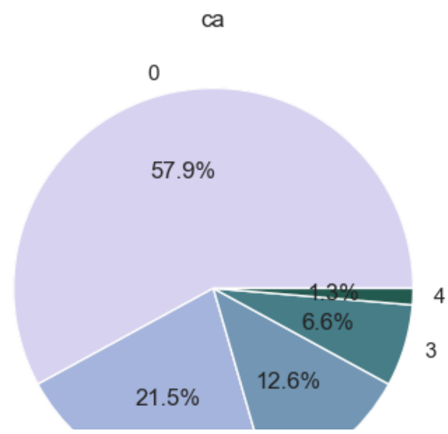

cp
0
47.4%
7.6%
3
28.5%
16.6%
2
1

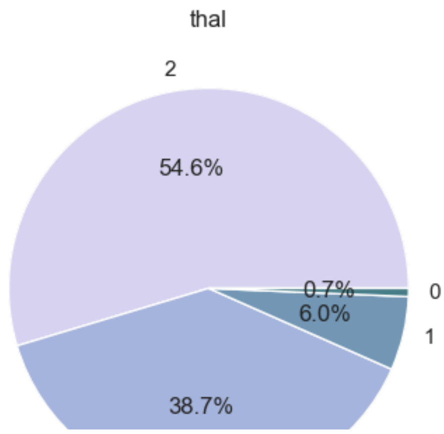

fbs
0
85.1%
14.9%
1


restecg
1
50.0%
1.3%
2
48.7%
0


exang
0
67.2%
32.8%
1

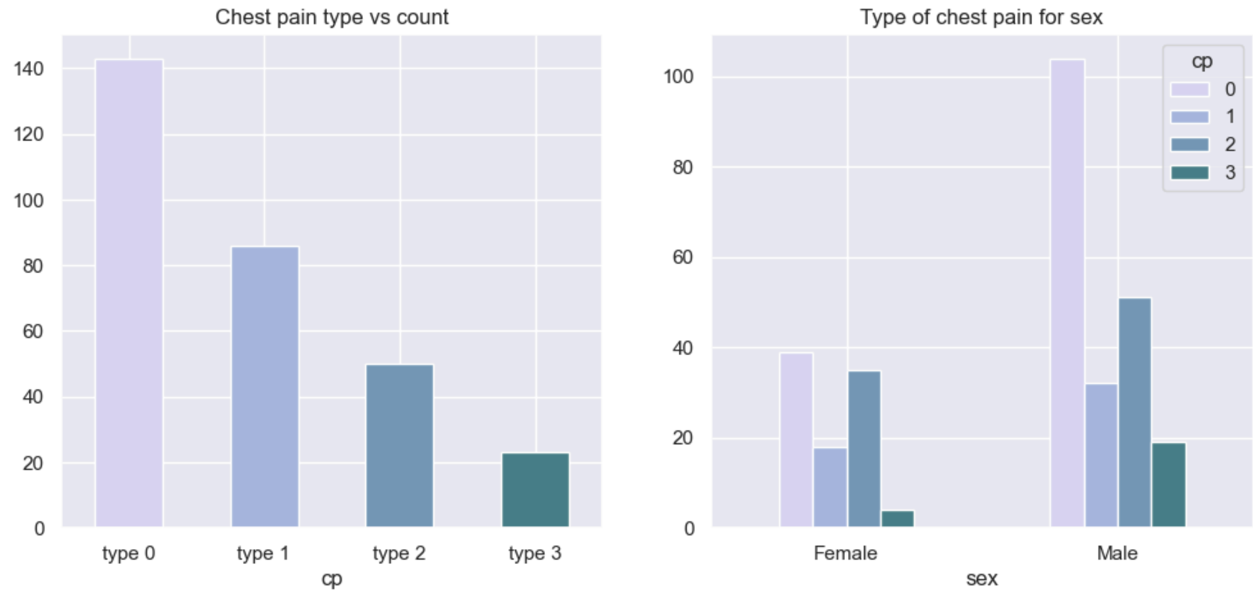

slope
2
46.7%
7.0%
0
46.4%
1


ca
0
57.9%
1.3%
4
6.6%
3
12.6%
21.5%


thal
2
54.6%
0.7%
0
6.0%
1
38.7%

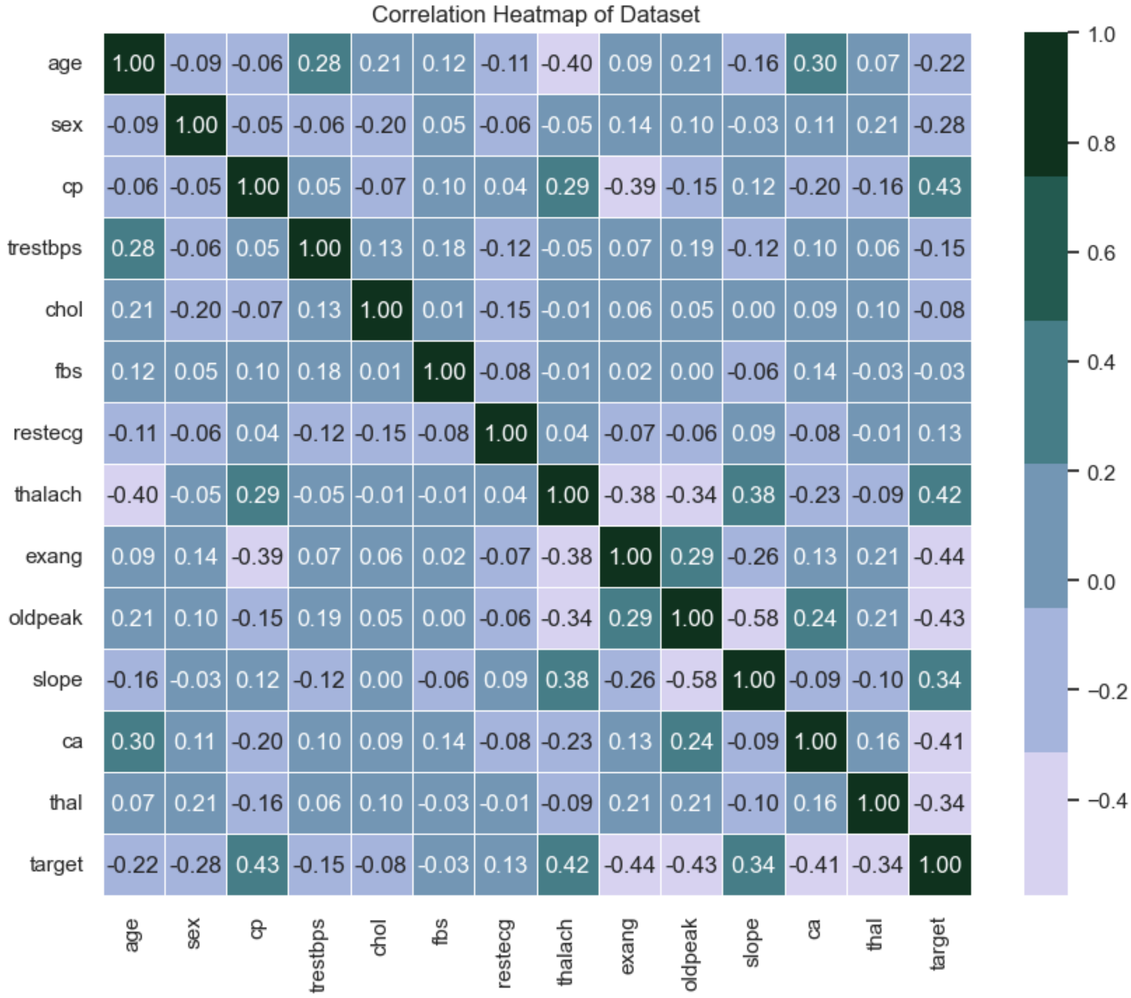
Bar chart for Chest Pain Type counts

In [15]:
```
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Chart 1 - Chest Pain Type vs Count
ax1 = Heart_Disease['cp'].value_counts().plot(kind='bar', color=sns.cubehelix_palette(start=2), ax=axes[0])
ax1.set_xticklabels(labels=['type 0', 'type 1', 'type 2', 'type 3'], rotation=0)
ax1.set_title('Chest pain type vs count')

# Chart 2 - Type of chest pain for sex
ax2 = pd.crosstab(Heart_Disease['sex'], Heart_Disease['cp']).plot(kind='bar', color=sns.cubehelix_palette(start=2), ax=axes[1
ax2.set_xticklabels(labels=['Female', 'Male'], rotation=0)
ax2.set_title('Type of chest pain for sex')

plt.show()
```



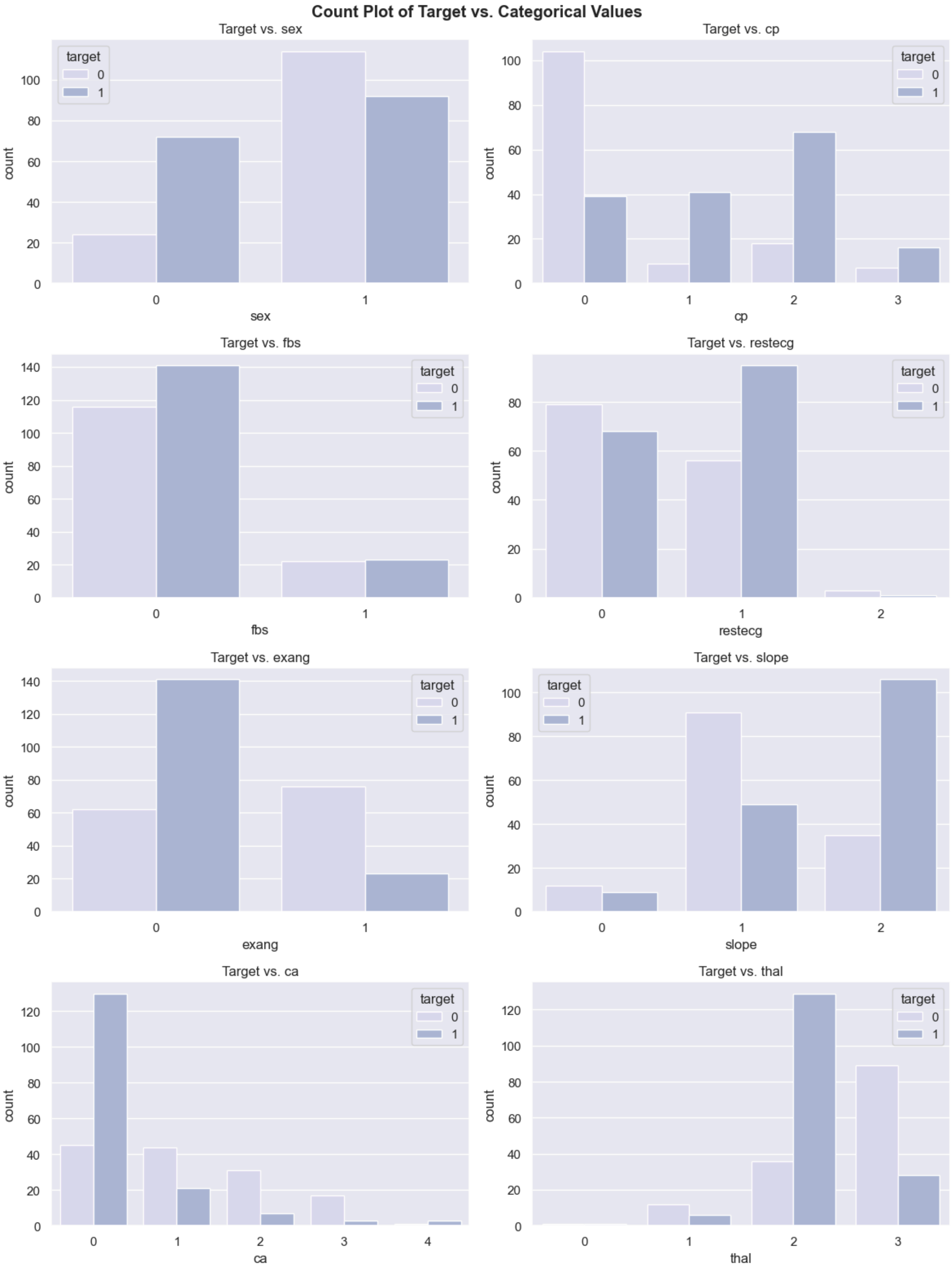
Correlation Heatmap of Numerical Variables

In [16]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(Heart_Disease.corr(), annot=True, cmap=
            sns.cubehelix_palette(start=2), fmt=".2f", linewidths=.5).set(
    title="Correlation Heatmap of Dataset")
plt.show()
```



The Relationship Between Categorical Variables and Heart Disease (Target)

In [17]:
```
sns.set(style="darkgrid")

fig, axes = plt.subplots(nrows=4, ncols=2, figsize=(12, 16))

Categorical_Features = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal']

for i, cat in enumerate(Categorical_Features):
    row = i // 2
    col = i % 2
    sns.countplot(x=cat, hue='target', data=Heart_Disease, ax=axes[row, col], palette=sns.cubehelix_palette(start=2))
    axes[row, col].set_title(f"Target vs. {cat}")

fig.suptitle("Count Plot of Target vs. Categorical Values", fontweight='bold')
plt.tight_layout()
plt.show()
```



Modeling

In [18]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

In [19]:
```
# Splitting data into features (x) and target (y)

x = Heart_Disease.drop("target", axis=1)

y = Heart_Disease["target"]
```

In [20]:
```
# Splitting the data into train and test sets

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.5, random_state=50)
```

In [21]:
```
# Data Normalization using Min-Max Method

x = MinMaxScaler().fit_transform(x)
```

Model Implementation

Logistic Regression

In [22]:
```
LR_classifier = LogisticRegression(max_iter=1000, random_state=1,
                                   solver='liblinear', penalty='l1').fit(
    x_train, y_train)

y_pred_LR = LR_classifier.predict(x_test)
```

In [23]:
```
LR_Accuracy = accuracy_score(y_pred_LR, y_test)

print('Logistic Regression Accuracy:'+'\033[1m {:.2f}%'.format(LR_Accuracy*100))
```

```
Logistic Regression Accuracy: 82.12%
```

K-Nearest Neighbour (KNN)

In [24]:
```
KNN_Classifier = KNeighborsClassifier(n_neighbors=3).fit(
    x_train, y_train)

y_pred_KNN = KNN_Classifier.predict(x_test)
```

In [25]:
```
KNN_Accuracy = accuracy_score(y_pred_KNN, y_test)

print('K-Nearest Neighbour Accuracy:'+'\033[1m {:.2f}%'.format(KNN_Accuracy*100))
```

```
K-Nearest Neighbour Accuracy: 57.62%
```

Support Vector Machine (SVM)

In [26]:
```
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

In [27]:
```
SVM_Classifier = SVC(kernel='linear', max_iter=5000, C=10, probability=True).fit(
    x_train_scaled, y_train)

y_pred_SVM = SVM_Classifier.predict(x_test_scaled)
```

```
C:\Users\stati\AppData\Roaming\Python\Python311\site-packages\sklearn\svm\_base.py:297: ConvergenceWarning: Solver terminated
early (max_iter=5000).  Consider pre-processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
```

In [28]:
```
SVM_Accuracy = accuracy_score(y_pred_SVM, y_test)

print('Support Vector Machine Accuracy:', '\033[1m{:.2f}%'.format(SVM_Accuracy * 100))
```

```
Support Vector Machine Accuracy: 82.12%
```

Decision Tree

In [29]:
```
DT_Classifier = DecisionTreeClassifier(max_depth=3, min_samples_leaf=5, criterion='entropy', min_samples_split=5,
                                       splitter='random', random_state=1).fit(
    x_train, y_train)

y_pred_DT = DT_Classifier.predict(x_test)
```

In [30]:
```
DT_Accuracy = accuracy_score(y_pred_DT, y_test)

print('Decision Tree Accuracy:', '\033[1m{:.2f}%'.format(DT_Accuracy * 100))
```

```
Decision Tree Accuracy: 78.15%
```

Random Forest

In [31]:
```
RF_Classifier = RandomForestClassifier(n_estimators=1000, random_state=1,
                                       max_leaf_nodes=20, min_samples_split=15).fit(
    x_train, y_train)
```

```
y_pred_RF = RF_Classifier.predict(x_test)
```

In [32]:
```
RF_Accuracy = accuracy_score(y_pred_RF, y_test)

print('Random Forest Accuracy:'+'\033[1m {:.2f}%'.format(RF_Accuracy*100))
```

```
Random Forest Accuracy: 81.46%
```

Model Comparison

In [33]:
```
compare = pd.DataFrame({'Model': ['Logistic Regression', 'K-Nearest Neighbors',
                                  'Support Vector Machine', 'Decision Tree', 'Random Forest'],
                        'Accuracy': [
                            LR_Accuracy * 100, KNN_Accuracy * 100, SVM_Accuracy * 100,
                            DT_Accuracy * 100, RF_Accuracy * 100]})
```

In [34]:
```
# Sorting and styling the comparison table

compare_sorted = compare.sort_values(by='Accuracy',
                                     ascending=False).reset_index(drop=True)
compare_styled = compare_sorted.style.background_gradient(
    cmap='Blues', subset=['Accuracy']).set_properties(**{'font-family': 'Segoe UI'})
```

In [35]:
```
compare_styled
```

Out[35]:

	Model	Accuracy
0	Logistic Regression	82.119205
1	Support Vector Machine	82.119205
2	Random Forest	81.456954
3	Decision Tree	78.145695
4	K-Nearest Neighbors	57.615894