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

```
# read the csv file
data = pd.read_csv('/content/Heart Disease data.csv')
```

```
data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2

```
data.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
```

```
# DATA CLEANING
```

```
# 1.b.  Based on these findings, remove duplicates (if any) and treat missing values using an appropriate strategy
```

```
data.target.value_counts()
```

```
1    526
0    499
Name: target, dtype: int64
```

```
data.isna().sum()
```

```
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
```

```
data.duplicated()
```

```
0       False
1       False
2       False
3       False
4       False
        ...
1020     True
1021     True
1022     True
1023     True
1024     True
Length: 1025, dtype: bool
```

```
duplicate = data[data.duplicated()]
print("Duplicate Rows :")
duplicate
```

Duplicate Rows :

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
15	34	0	1	118	210	0	1	192	0	0.7	2	0	2	1
31	50	0	1	120	244	0	1	162	0	1.1	2	0	2	1
43	46	1	0	120	249	0	0	144	0	0.8	2	0	3	0
55	55	1	0	140	217	0	1	111	1	5.6	0	0	3	0
61	66	0	2	146	278	0	0	152	0	0.0	1	1	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
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

723 rows × 14 columns

```
data.drop_duplicates(inplace=True)
```

```
data.duplicated()
```

```
0      False
1      False
2      False
3      False
4      False
       ...
723    False
733    False
739    False
843    False
878    False
Length: 302, dtype: bool
```

```
data.nunique()
```

```
age          41
sex           2
cp            4
trestbps     49
chol        152
fbs           2
restecg       3
thalach      91
exang         2
oldpeak      40
slope         3
ca            5
thal          4
target        2
dtype: int64
```

```
data.cp.value_counts()
```

```
0    143
2     86
1     50
3     23
Name: cp, dtype: int64
```

```
data.ca.value_counts()
```

```
0    175
1     65
2     38
3     20
4      4
Name: ca, dtype: int64
```

```
# data dictionary says that ca (Number of major vessels colored by fluoroscopy) is (0-3), but in the given dataset
# its given as 0-4, the four records showing category-4 should be removed
```

```
data[data['ca']==4]
```

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target

```
# those values are marked as NAN , so that it can be removed during removal of missing values phase
```

```
data.loc[data['ca']==4, 'ca']=np.NaN    # marking those values as NAN values
```

```
data['ca'].unique()
```

```
array([2., 0., 1., 3.])
```

```
data.thal.value_counts()
```

```
2.0    165
3.0    117
1.0     18
Name: thal, dtype: int64
```

```
# as per data dict, feature -thal has 3 unique values, but dataset has two 0 values , which needs to be removed
```

```
data.loc[data['thal']==0]
```

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target

```
data.loc[data['thal']==0, 'thal']=np.NaN
```

```
data.thal.unique()
```

```
array([ 3.,  2.,  1., nan])
```

```
data.isna().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          4
thal        2
target      0
dtype: int64
```

```
data.ca.mode()
```

```
0    0.0
Name: ca, dtype: float64
```

```
data['ca'].fillna(data['ca'].mode().iloc[0], inplace=True)
```

```
data.thal.mode()
```

```
0    2.0
Name: thal, dtype: float64
```

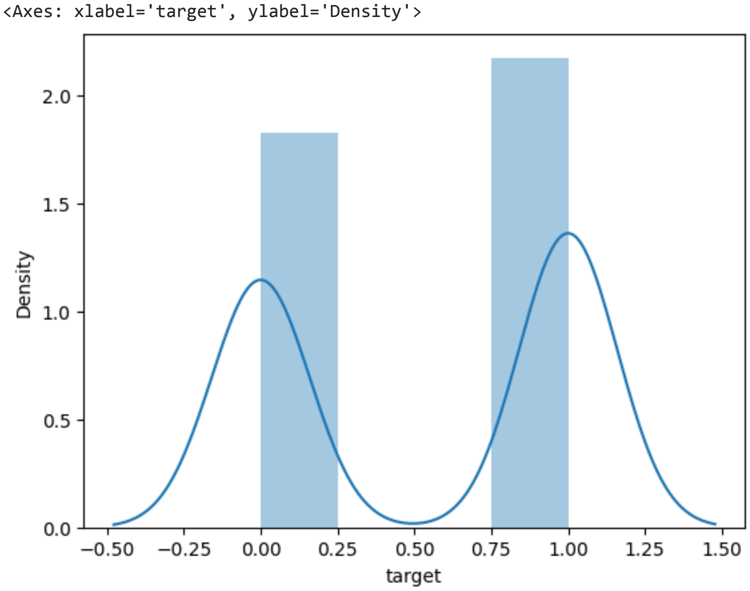
```
data['thal'].fillna(data['thal'].mode().iloc[0], inplace=True)
```

```
data.isnull().sum()
```

```
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
```

```
sns.distplot(data.target)
```

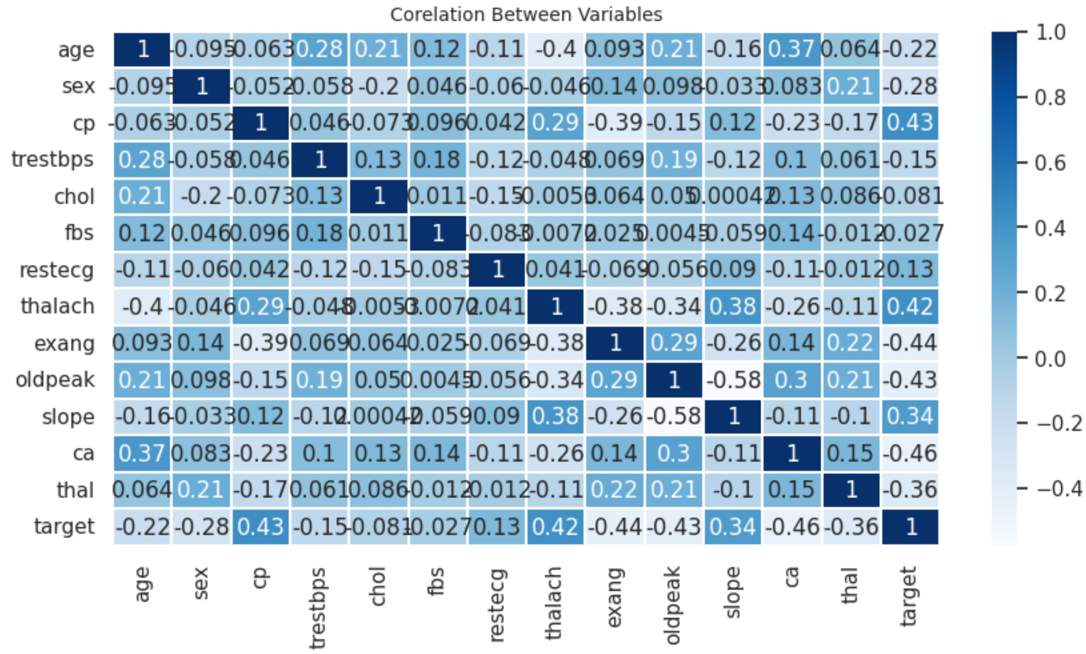
```
<ipython-input-48-b9023a40a00b>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data.target)
<Axes: xlabel='target', ylabel='Density'>
```




```
data.shape
```

```
(302, 14)
```

```
sns.set(style="white")
plt.rcParams['figure.figsize'] = (10, 5)
corrmat=sns.heatmap(data.corr(), annot = True, linewidths=.1, cmap="Blues")
plt.title('Corelation Between Variables', fontsize = 10)
plt.show()
```




```
# Data Exploration
```

```
data.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	
count	302.00000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302
mean	54.42053	0.682119	0.963576	131.602649	246.500000	0.149007	0.526490	149.569536	0.327815	1.043046	1.397351	0
std	9.04797	0.466426	1.032044	17.563394	51.753489	0.356686	0.526027	22.903527	0.470196	1.161452	0.616274	0
min	29.00000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0
25%	48.00000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.250000	0.000000	0.000000	1.000000	0
50%	55.50000	1.000000	1.000000	130.000000	240.500000	0.000000	1.000000	152.500000	0.000000	0.800000	1.000000	0
75%	61.00000	1.000000	2.000000	140.000000	274.750000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1
max	77.00000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	3

```
data['target'] = data.target.replace({1: "Possible CVD", 0: "No CVD"})
```

```
data['sex'] = data.sex.replace({1: "Male", 0: "Female"})
```

```
data['cp'] = data.cp.replace({0: "typical_angina",
                              1: "atypical_angina",
                              2:"non-anginal pain",
                              3: "asymptomatic"})
```

```
data['exang'] = data.exang.replace({1: "Yes", 0: "No"})
```

```
data['fbs'] = data.fbs.replace({1: "True", 0: "False"})
```

```
data['slope'] = data.slope.replace({0: "upslope", 1: "flat",2:"downslope"})
```

```
data['thal'] = data.thal.replace({1: "fixed_defect", 2: "reversable_defect", 3:"normal"})
```

```
data['restecg'] = data.restecg.replace({1: "Normal", 0: "Abnormal"})
```

```
data.head(1)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	s
0	52	Male	typical angina	125	212	False	Normal	168	No	1.0	down

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

def outlier_treatment(data, drop=False):
    for eachFeature in continuousFeatures:
        featureData = data[eachFeature]
        Q1, Q3 = np.percentile(featureData, [25, 75])
        IQR = Q3 - Q1
        outlierCalc = 1.5 * IQR
        outliers = featureData[~((featureData >= Q1 - outlierCalc) & (featureData <= Q3 + outlierCalc))].index.tolist()
        if not drop:
            print('For the feature {}, No of Outliers is {}'.format(eachFeature, len(outliers)))
        if drop:
            data.drop(outliers, inplace=True)
            print('Outliers from {} feature removed'.format(eachFeature))

outlier_treatment(data[continuousFeatures], drop=True)
```

```
Outliers from age feature removed
Outliers from trestbps feature removed
Outliers from chol feature removed
Outliers from thalach feature removed
Outliers from oldpeak feature removed
<ipython-input-62-bd5de829edf8>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  data.drop(outliers, inplace=True)
<ipython-input-62-bd5de829edf8>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  data.drop(outliers, inplace=True)
<ipython-input-62-bd5de829edf8>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  data.drop(outliers, inplace=True)
<ipython-input-62-bd5de829edf8>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  data.drop(outliers, inplace=True)
<ipython-input-62-bd5de829edf8>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  data.drop(outliers, inplace=True)
```

```
# Skewness: If the index is between -1 and 1, then the distribution is symmetric.
# If the index is no more than -1 then it is skewed to the left and if it is at least 1, then it is skewed to the right
```

```
data.skew()
```

```
<ipython-input-49-b3b431164adb>:1: FutureWarning: The default value of numeric_only in DataFrame.skew is deprecated. In a future versior
  data.skew()
age        -0.203743
trestbps    0.716541
chol        1.147332
thalach    -0.532671
oldpeak     1.266173
ca          1.203952
dtype: float64
```

```
# b.    Identify the data variables which are categorical
```

```
# Finding the Numerical and Categorical variables in the dataset
# All the features in the dataset are in numerical (int, float) format
```

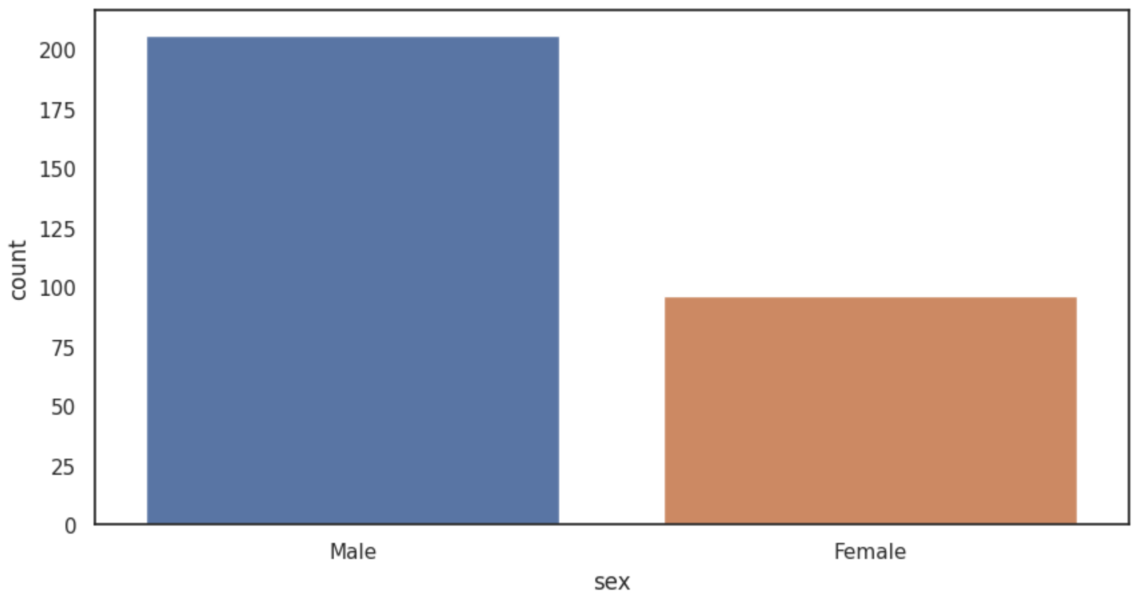
```
categorical_features = data.select_dtypes(include=['object']).columns.tolist()
```

```
categorical_features
```

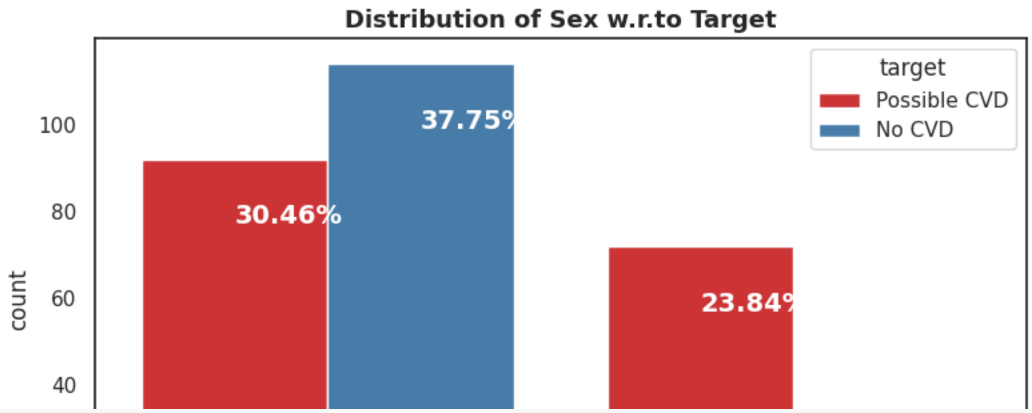
```
['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'thal', 'target']
```

```
#b. Identify the data variables which are categorical
# and describe and explore these variables using the appropriate tools, such as count plot
```

```
sns.countplot(data=data, x='sex')
plt.show()
```



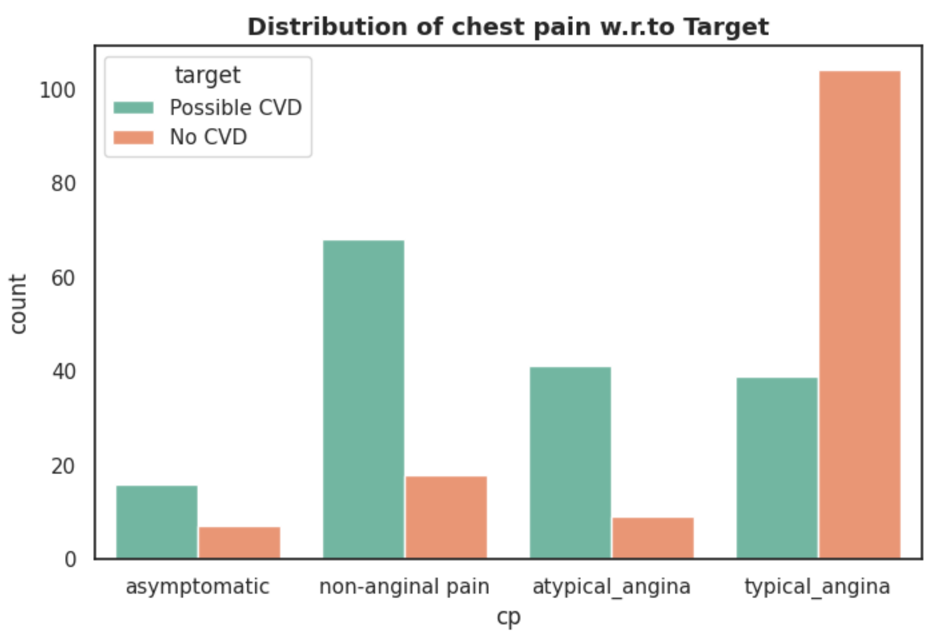

```
fig, ax = plt.subplots(figsize=(8, 5))

sns.countplot(x='sex', hue='target', data=data, palette='Set1', ax=ax)
ax.set_title("Distribution of Sex w.r.to Target", fontsize=13, weight='bold')
ax.set_xticklabels(['Female', 'Male'], rotation=0)

totals = []
for i in ax.patches:
    totals.append(i.get_height())
total = sum(totals)
for i in ax.patches:
    ax.text(i.get_x() + i.get_width() / 2, i.get_height() - 15,
            f"{round((i.get_height() / total) * 100, 2)}%", fontsize=14,
            color='white', weight='bold')

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




```
fig, ax = plt.subplots(figsize=(8, 5))

sns.countplot(x='cp', hue='target', data=data, palette='Set2', ax=ax)
ax.set_title("Distribution of chest pain w.r.to Target", fontsize=13, weight='bold')
ax.set_xticklabels(name, rotation=0)

totals = []
for i in ax.patches:
    totals.append(i.get_height())
total = sum(totals)
for i in ax.patches:
    ax.text(i.get_x() + i.get_width() / 2, i.get_height() - 15,
            f"{round((i.get_height() / total) * 100, 2)}%", fontsize=14,
            color='white', weight='bold')

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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-57-8991ea1c57c9> in <cell line: 5>()
      3 sns.countplot(x='cp', hue='target', data=data, palette='Set2', ax=ax)
      4 ax.set_title("Distribution of chest pain w.r.to Target", fontsize=13, weight='bold')
----> 5 ax.set_xticklabels(name, rotation=0)
      6
      7 totals = []

NameError: name 'name' is not defined
```

SEARCH STACK OVERFLOW




```
fig, ax = plt.subplots(figsize=(8, 5))
```

```
sns.countplot(x='fbs', hue='target', data=data, palette='Set3', ax=ax)
ax.set_title("Distribution of Fasting Blood Sugar w.r.to Target", fontsize=13, weight='bold')
ax.set_xticklabels(['True','False'], rotation=0)

totals = []
for i in ax.patches:
    totals.append(i.get_height())
total = sum(totals)
for i in ax.patches:
    ax.text(i.get_x() + i.get_width() / 2, i.get_height() - 15,
            f"{round((i.get_height() / total) * 100, 2)}%", fontsize=14,
            color='white', weight='bold')

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

```
# Many patients with No CVD are found to have High Fasting Blood Sugar,
# so we can say that this is not a strong feature with respect to our target
```

```
# c.    Study the occurrence of CVD across the Age category
```

```
fig, ax = plt.subplots(figsize=(10, 6))

sns.countplot(x='age', hue='target', data=data, palette='Set2', ax=ax)
ax.set_title("Distribution of CVD across age categories", fontsize=13, weight='bold')
ax.set_xlabel("Age", fontsize=12)
ax.set_ylabel("Count", fontsize=12)
ax.legend(title="CVD", labels=["No", "Yes"])

totals = []
for i in ax.patches:
    totals.append(i.get_height())
total = sum(totals)
for i in ax.patches:
    ax.text(i.get_x() + i.get_width() / 2, i.get_height() - 15,
            f"{round((i.get_height() / total) * 100, 2)}%", fontsize=14,
            color='white', weight='bold')

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

```
#d. Study the composition of all patients with respect to the Sex category
```

```
sex_counts = data['sex'].value_counts()
labels = ['Male', 'Female']

fig, ax = plt.subplots(figsize=(6, 6))
ax.pie(sex_counts, labels=labels, autopct='%1.1f%%', startangle=90, colors=['skyblue', 'lightgreen'])
ax.set_title("Composition of Patients by Sex", fontsize=14, weight='bold')

plt.show()
```

```
# e.    Study if one can detect heart attacks based on anomalies in the resting blood pressure (trestbps) of a patient
```

```
fig, ax = plt.subplots(figsize=(8, 6))
sns.boxplot(x='target', y='trestbps', data=data, ax=ax)
ax.set_title("Resting Blood Pressure (trestbps) vs. Heart Attack", fontsize=14, weight='bold')
ax.set_xlabel("Heart Attack", fontsize=12)
ax.set_ylabel("Resting Blood Pressure (mm Hg)", fontsize=12)
ax.set_xticklabels(["No", "Yes"])

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

```
# f.    Describe the relationship between cholesterol levels and a target variable
```

```
plt.figure(figsize=(8, 6))
sns.histplot(data=data, x='chol', hue='target', kde=True, palette='Set2')
```

```
sns.histplot(data=data, x='chol', hue='target', kde=True, palette='Set2')
plt.title("Distribution of Cholesterol Levels by Target", fontsize=14, weight='bold')
plt.xlabel("Cholesterol Levels", fontsize=12)
plt.ylabel("CVD", fontsize=12)
plt.legend(title="Target", labels=["No", "Yes"])

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

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='target', y='chol', data=data, palette='Set2')
plt.title("Cholesterol Levels by Target Variable", fontsize=14, weight='bold')
plt.xlabel("CVD", fontsize=12)
plt.ylabel("Cholesterol Levels", fontsize=12)
plt.xticks(ticks=[0, 1], labels=["No", "Yes"])

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

```
# We observe that The Cholesterol feature('chol') has less effect on the target
```

```
# g.    State what relationship exists between peak exercising and the occurrence of a heart attack
```

```
fig, ax = plt.subplots(figsize=(8, 5))

sns.countplot(x='slope', hue='target', data=data, palette='Set1', ax=ax)
ax.set_title("Slope Distribution w.r.to Target", fontsize=13, weight='bold')
ax.set_xticklabels(['Upsloping','Flat','Downsloping'], rotation=0)

totals = []
for i in ax.patches:
    totals.append(i.get_height())
total = sum(totals)
for i in ax.patches:
    ax.text(i.get_x() + i.get_width() / 2, i.get_height() - 15,
            f"{round((i.get_height() / total) * 100, 2)}%", fontsize=14,
            color='white', weight='bold')

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

```
#i. List how the other factors determine the occurrence of CVD
```

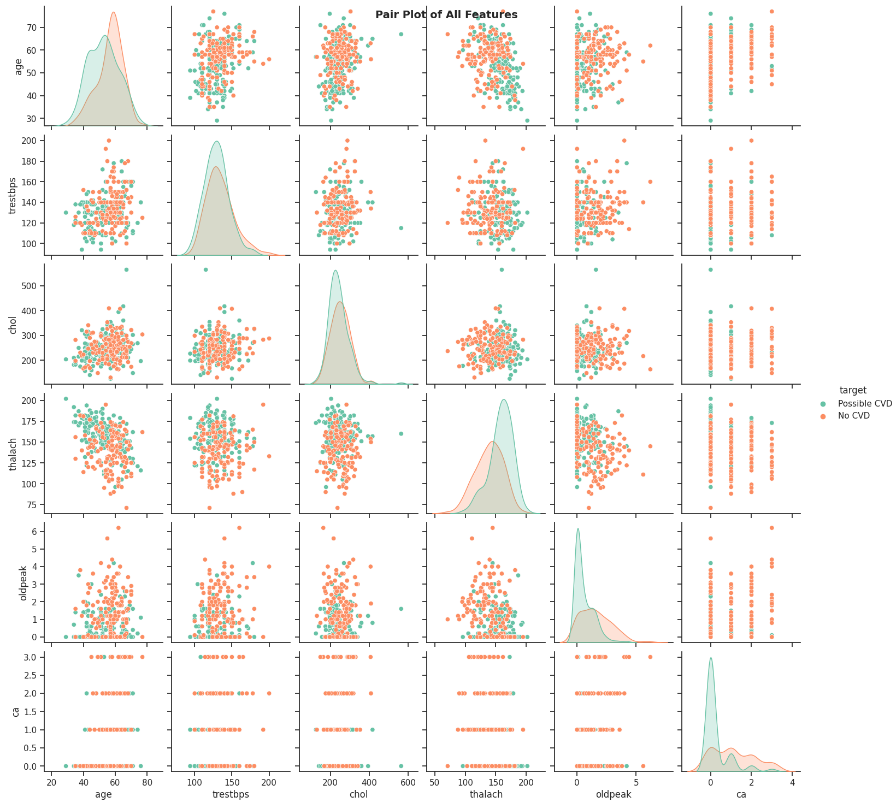
```
# h.    Check if thalassemia is a major cause of CVD
```

```
data.thal.value_counts()
```

```
fig, ax = plt.subplots(figsize=(8, 5))

sns.countplot(x='thal', hue='target', data=data, palette='Set1', ax=ax)
ax.set_title(" Effect of Thalassemia on CVD", fontsize=13, weight='bold')
ax.set_xticklabels(['reversable_defect','normal','fixed_defect'], rotation=0)

totals = []
for i in ax.patches:
    totals.append(i.get_height())
total = sum(totals)
for i in ax.patches:
    ax.text(i.get_x() + i.get_width() / 2, i.get_height() - 15,
            f"{round((i.get_height() / total) * 100, 2)}%", fontsize=14,
            color='white', weight='bold')

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

```
#j. Use a pair plot to understand the relationship between all the given variables
```

```
sns.set(style="ticks")
sns.pairplot(data, hue="target", palette="Set2")
plt.suptitle("Pair Plot of All Features", fontsize=14, fontweight='bold')
plt.show()
```




```
#‘cp’, ‘thalach’, ‘slope’ shows good positive correlation with target
#‘oldpeak’, ‘exang’, ‘ca’, ‘thal’, ‘sex’, ‘age’ shows a good negative correlation with target
#‘fbs’ ‘chol’, ‘trestbps’, ‘restecg’ has low correlation with our target
```

```
categorical_features
```

```
['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'thal', 'target']
```

```
from sklearn.preprocessing import OneHotEncoder

data[categorical_features] = data[categorical_features].astype(str)

# Extract the categorical variables and numerical features into separate DataFrames
categorical_data = data[categorical_features]
numerical_data = data.drop(categorical_features, axis=1)

# Create an instance of OneHotEncoder
encoder = OneHotEncoder(sparse_output=False,drop='first')

# Fit and transform the categorical variables
categorical_encoded = encoder.fit_transform(categorical_data)

# Create a DataFrame from the encoded categorical variables
categorical_encoded_df = pd.DataFrame(categorical_encoded, columns=encoder.get_feature_names_out(categorical_features))

# Concatenate the encoded categorical variables with the numerical features
encoded_data = pd.concat([numerical_data, categorical_encoded_df], axis=1)
```

```
categorical_data
```

	sex	cp	fbs	restecg	exang	slope	thal	target
0	Male	asymptomatic	True	Abnormal	No	upslope	fixed_defect	Possible CVD
1	Male	non-anginal pain	False	Normal	No	upslope	reversable_defect	Possible CVD
2	Female	atypical_angina	False	Abnormal	No	downslope	reversable_defect	Possible CVD
3	Male	atypical_angina	False	Normal	No	downslope	reversable_defect	Possible CVD
4	Female	typical_angina	False	Normal	Yes	downslope	reversable_defect	Possible CVD
...	...	...	...	...	...	...	...	...
298	Female	typical_angina	False	Normal	Yes	flat	normal	No CVD
299	Male	asymptomatic	False	Normal	No	flat	normal	No CVD
300	Male	typical_angina	True	Normal	No	flat	normal	No CVD
301	Male	typical_angina	False	Normal	Yes	flat	normal	No CVD

```
encoded_data.isna().sum()
```

```
age                        1
trestbps                   1
chol                       1
thalach                    1
oldpeak                    1
ca                         1
sex_Male                   1
cp_atypical_angina         1
cp_non-anginal pain        1
cp_typical_angina          1
fbs_True                   1
restecg_Abnormal           1
restecg_Normal             1
exang_Yes                  1
slope_flat                 1
slope_upslope              1
thal_normal                1
thal_reversable_defect     1
target_Possible CVD        1
dtype: int64
```

```
encoded_data.shape
```

```
(303, 19)
```

```
rows_with_nan = encoded_data[encoded_data.isnull().any(axis=1)]
rows_with_nan
```

	age	trestbps	chol	thalach	oldpeak	ca	sex_Male	cp_atypical_angina	cp_non-anginal pain	cp_typical_angina	fbs_True	restecg_Abnormal
302	57.0	130.0	236.0	174.0	0.0	1.0	NaN	NaN	NaN	NaN	NaN	NaN
164	NaN	NaN	NaN	NaN	NaN	NaN	1.0	0.0	0.0	1.0	0.0	1.0

```
encoded_data.dropna(inplace = True)
```

```
#DATA PRE PROCESSING
```

```
#Seperate data as features and label
features = encoded_data.iloc[:,:-1].values
label = encoded_data.iloc[:,-1].values
```

```
features.ndim
```

```
2
```

```
label.ndim
```

```
1
```

```
from sklearn.preprocessing import StandardScaler
X_std=StandardScaler().fit_transform(encoded_data)
```

```
#Create train test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(features,label,test_size=0.2,random_state=4)
```

```
#Apply LogisticRegression
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Converger
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
▾ LogisticRegression
LogisticRegression()
```

```
print(model.score(X_train,y_train))
print(model.score(X_test,y_test))
```

```
0.8708333333333333
0.9180327868852459
```

```
from sklearn.metrics import f1_score,precision_score,recall_score

predictTrain = model.predict(X_train)
predictTest = model.predict(X_test)

print("F1 Score of Training Set : ",f1_score(y_train,predictTrain,average=None))
print("F1 Score of Testing Set : ",f1_score(y_test,predictTest,average=None))
print("F1 Score of Training Set : ",precision_score(y_train,predictTrain,average=None))
print("F1 Score of Testing Set : ",precision_score(y_test,predictTest,average=None))
print("F1 Score of Training Set : ",recall_score(y_train,predictTrain,average=None))
print("F1 Score of Testing Set : ",recall_score(y_test,predictTest,average=None))
```

```
F1 Score of Training Set :  [0.85581395 0.88301887]
F1 Score of Testing Set :  [0.90566038 0.92753623]
F1 Score of Training Set :  [0.86792453 0.87313433]
F1 Score of Testing Set :  [0.96       0.88888889]
F1 Score of Training Set :  [0.8440367  0.89312977]
F1 Score of Testing Set :  [0.85714286 0.96969697]
```

```
from sklearn.metrics import classification_report
print(classification_report(y_train,predictTrain))
```

```
              precision    recall  f1-score   support

         0.0       0.87      0.84      0.86       109
         1.0       0.87      0.89      0.88       131

    accuracy                           0.87       240
   macro avg       0.87      0.87      0.87       240
weighted avg       0.87      0.87      0.87       240
```

```
# Testing for Generalization
```

```
from sklearn.model_selection import train_test_split
for i in range(1,300):
  X_train,X_test,y_train,y_test = train_test_split(features,label,test_size=0.2,random_state=i)
  model = LogisticRegression()
  model.fit(X_train,y_train)
  trainScore = model.score(X_train,y_train)
  testScore = model.score(X_test,y_test)

  if testScore > trainScore:
    print("Test {} Train {} RS {}".format(testScore,trainScore,i))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Test 0.9016393442622951 Train 0.875 RS 2
Test 0.9180327868852459 Train 0.8875 RS 3
Test 0.9180327868852459 Train 0.8708333333333333 RS 4
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
finalmodel = LogisticRegression()
X_train,X_test,y_train,y_test = train_test_split(features,label,test_size=0.2,random_state=144)
finalmodel.fit(X_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: Converger
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
predictEntireDataset = finalmodel.predict(features)

print("F1 Score of Enitre Set : ",np.mean(f1_score(label,predictEntireDataset,average=None)))
```

```
F1 Score of Enitre Set :  0.8889485158477277
```

```
# Create Confusion Matrix

from sklearn.metrics import confusion_matrix
confusion_matrix(label,predictEntireDataset)
```

```
array([[117,  20],
       [ 13, 151]])
```

```
from sklearn.metrics import classification_report
print(classification_report(label,predictEntireDataset))
```

```
              precision    recall  f1-score   support

         0.0       0.90      0.85      0.88       137
         1.0       0.88      0.92      0.90       164

    accuracy                           0.89       301
   macro avg       0.89      0.89      0.89       301
weighted avg       0.89      0.89      0.89       301
```

```
# Random Forest Algorithm
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score

modelrf = RandomForestClassifier(n_estimators=100, random_state=42)
modelrf.set_params(min_samples_leaf=15)
modelrf.set_params(max_depth=6)
cv_scores = cross_val_score(modelrf, X_train, y_train, cv=10)
modelrf.fit(X_train,y_train)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=6, min_samples_leaf=15, random_state=42)
```

```
modelrf.score(X_train,y_train)
```

```
0.8791666666666667
```

```
modelrf.score(X_test,y_test)
```

```
0.9016393442622951
```

```
from sklearn.metrics import accuracy_score

y_pred = modelrf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9016393442622951
```

```
# Accuracy, Precision, f1 score for RFC
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

print(accuracy_score(y_test,y_pred ))
print(classification_report(y_test,y_pred))
```

```
0.9016393442622951
              precision    recall  f1-score   support

         0.0       1.00      0.79      0.88        29
         1.0       0.84      1.00      0.91        32

    accuracy                           0.90        61
   macro avg       0.92      0.90      0.90        61
weighted avg       0.92      0.90      0.90        61
```

```
print("Cross-validation scores:", cv_scores)
```

```
Cross-validation scores: [0.79166667 0.83333333 0.875      0.75       0.875      0.875
 0.83333333 0.79166667 0.83333333 0.875     ]
```

```
# Create Confusion Matrix

from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
```

```
array([[23,  6],
       [ 0, 32]])
```

```
rf_predictions =modelrf.predict(X_test)
```

```
# Evaluate random forest model
rf_accuracy = accuracy_score(y_test, rf_predictions)
rf_precision = precision_score(y_test, rf_predictions)
rf_recall = recall_score(y_test, rf_predictions)
rf_auc = roc_auc_score(y_test, rf_predictions)
```

```
print("\nRandom Forest:")
print("Accuracy:", rf_accuracy)
print("Precision:", rf_precision)
print("Recall:", rf_recall)
print("AUC-ROC:", rf_auc)
```

```
Random Forest:
Accuracy: 0.9016393442622951
Precision: 0.8421052631578947
Recall: 1.0
AUC-ROC: 0.896551724137931
```

```
# Logistic Regression
```

```
modellr= LogisticRegression()
modellr.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
▾ LogisticRegression
LogisticRegression()
```

```
print(modellr.score(X_train,y_train))
print(modellr.score(X_test,y_test))
```

```
0.8708333333333333
0.9672131147540983
```

```
#Evaluate model

from sklearn.metrics import f1_score,precision_score,recall_score
```

```
predictTrain = modellr.predict(X_train)
predictTest = modellr.predict(X_test)

print("F1 Score of Training Set : ",f1_score(y_train,predictTrain,average=None))
print("F1 Score of Testing Set : ",f1_score(y_test,predictTest,average=None))
print("F1 Score of Training Set : ",precision_score(y_train,predictTrain,average=None))
print("F1 Score of Testing Set : ",precision_score(y_test,predictTest,average=None))
print("F1 Score of Training Set : ",recall_score(y_train,predictTrain,average=None))
print("F1 Score of Testing Set : ",recall_score(y_test,predictTest,average=None))
```

```
F1 Score of Training Set :  [0.85308057 0.88475836]
F1 Score of Testing Set :  [0.96428571 0.96969697]
F1 Score of Training Set :  [0.87378641 0.86861314]
F1 Score of Testing Set :  [1.         0.94117647]
F1 Score of Training Set :  [0.83333333 0.90151515]
F1 Score of Testing Set :  [0.93103448 1.        ]
```

```
from sklearn.metrics import classification_report
print(classification_report(y_train,predictTrain))
```

```
              precision    recall  f1-score   support

         0.0       0.87      0.83      0.85       108
         1.0       0.87      0.90      0.88       132

    accuracy                           0.87       240
   macro avg       0.87      0.87      0.87       240
weighted avg       0.87      0.87      0.87       240
```

```
# Testing for Generalization

from sklearn.model_selection import train_test_split
for i in range(1,101):
  X_train,X_test,y_train,y_test = train_test_split(features,label,test_size=0.2,random_state=i)
  modellr1 = LogisticRegression()
  modellr1.fit(X_train,y_train)
  trainScore = modellr1.score(X_train,y_train)
  testScore = modellr1.score(X_test,y_test)

  if testScore > trainScore:
    print("Test {} Train {} RS {}".format(testScore,trainScore,i))
```

```
Test 0.9016393442622951 Train 0.875 RS 2
Test 0.9180327868852459 Train 0.8875 RS 3
Test 0.9180327868852459 Train 0.8708333333333333 RS 4
Test 0.9344262295081968 Train 0.9 RS 6
Test 0.9016393442622951 Train 0.9 RS 7
Test 0.9016393442622951 Train 0.8875 RS 9
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

```
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
encoded_data.head()
```

	age	trestbps	chol	thalach	oldpeak	ca	sex_Male	cp_atypical_angina	cp_non-anginal pain	cp
0	63.0	145.0	233.0	150.0	2.3	0.0	1.0	0.0	0.0	
1	37.0	130.0	250.0	187.0	3.5	0.0	1.0	0.0	1.0	
2	41.0	130.0	204.0	172.0	1.4	0.0	0.0	1.0	0.0	
3	56.0	120.0	236.0	178.0	0.8	0.0	1.0	1.0	0.0	
4	57.0	120.0	354.0	163.0	0.6	0.0	0.0	0.0	0.0	

```
import statsmodels.api as sm

# Separate the predictor variables (X) and the target variable (y)
X = encoded_data[['age', 'thalach', 'restecg_Abnormal', 'thal_reversable_defect']]
y = encoded_data['target_Possible CVD']

# Add constant column to the predictor variables matrix (required for statsmodels)
X = sm.add_constant(X)

# Fit the logistic regression model
logistic_model = sm.Logit(y, X)
result = logistic_model.fit()

# Get the summary of the model
summary = result.summary()

# Extract p-values from the summary table
p_values = result.pvalues

# Print the summary and p-values
print(summary)
print(p_values)
```

```
Optimization terminated successfully.
         Current function value: 0.459648
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:     target_Possible CVD   No. Observations:                  301
Model:                           Logit   Df Residuals:                      296
Method:                            MLE   Df Model:                            4
Date:                 Mon, 12 Jun 2023   Pseudo R-squ.:                  0.3330
Time:                         05:27:02   Log-Likelihood:                -138.35
converged:                        True   LL-Null:                       -207.42
Covariance Type:             nonrobust   LLR p-value:                 7.056e-29
==========================================================================================
                             coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------
const                     -6.8763      1.811     -3.796      0.000     -10.426      -3.326
age                       -0.0046      0.018     -0.254      0.800      -0.040       0.031
thalach                    0.0431      0.008      5.417      0.000       0.028       0.059
restecg_Abnormal          -0.7759      0.306     -2.539      0.011      -1.375      -0.177
```

```
thal_reversable_defect     2.4023      0.309      7.767      0.000       1.796       3.009
==================================================================================
const                     1.468588e-04
age                       7.997720e-01
thalach                   6.066739e-08
restecg_Abnormal          1.111740e-02
thal_reversable_defect    8.031421e-15
dtype: float64
```

```
#The Random Forest model gave 90% accuracy
# The Logistic regression model gave 87% accuracy
```