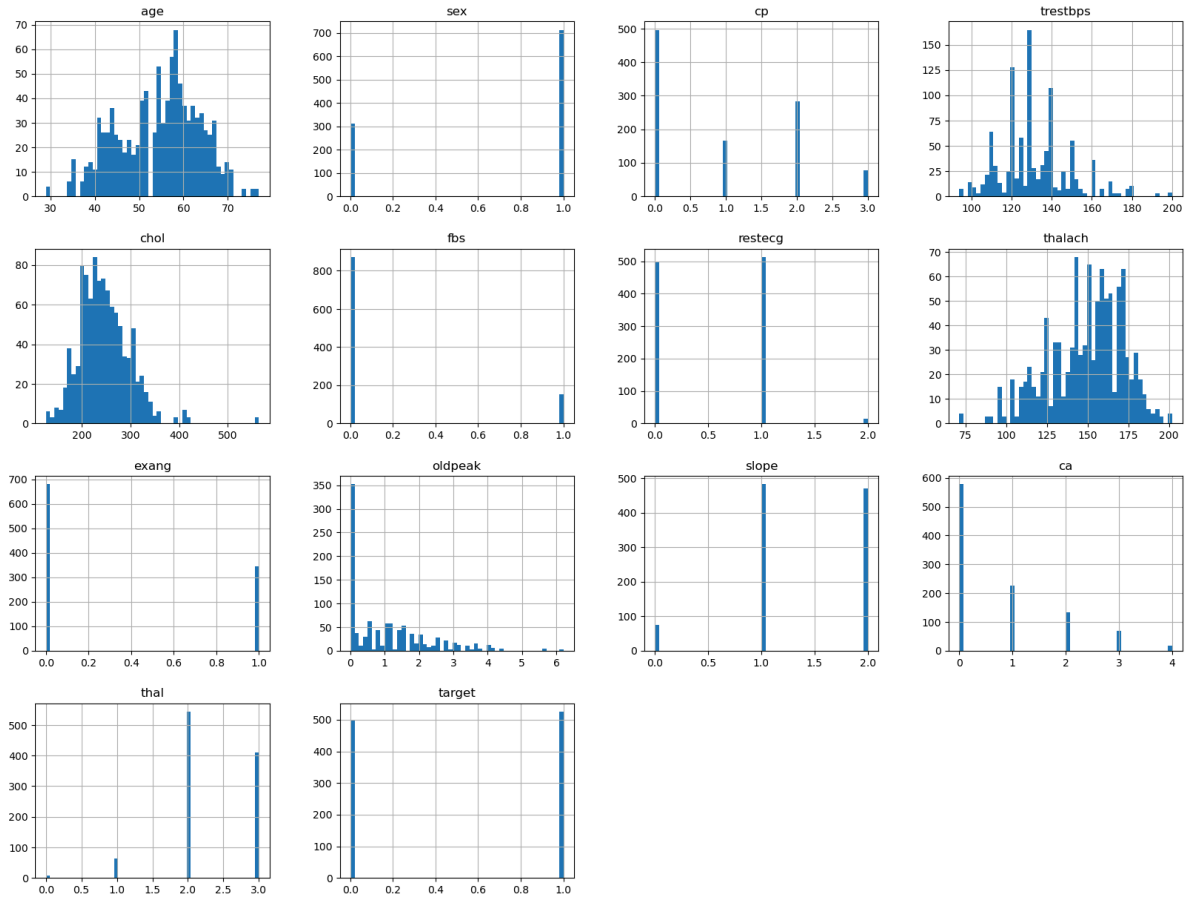
In [24]:

```python
import pandas as pd

# Load the dataset
file_path = "C:\\Users\\91778\Downloads\\heart.csv"
data = pd.read_csv(file_path)

# Display the first few rows and the info of the dataset
print(data.head())
print(data.info())
```

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

   ca  thal  target
0   2     3       0
1   0     3       0
2   0     3       0
3   1     3       0
4   3     2       0
<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
None
```

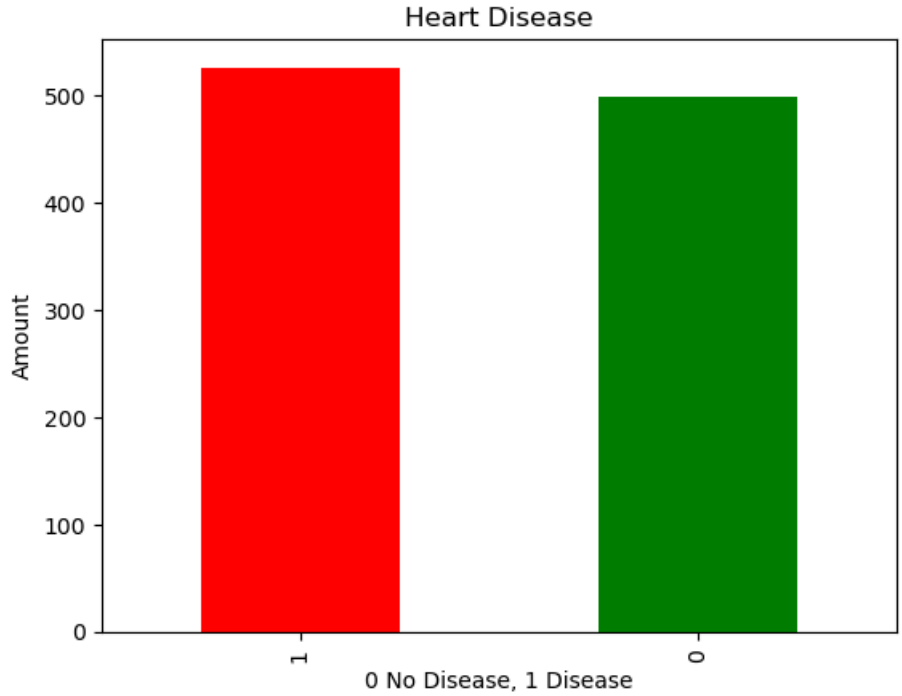
In [25]:
```
import matplotlib.pyplot as plt
data.hist(bins=50,grid=True,figsize=(20,15))
plt.show();
```



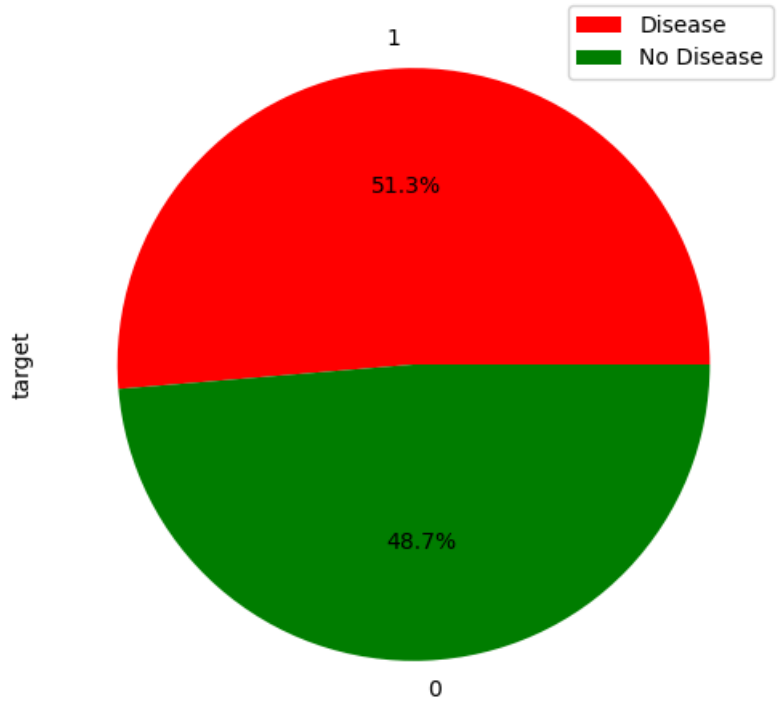
In [26]:
```
# How many people have heart disease and how many people don't have heart disease?
data.target.value_counts
```

Out[26]:
```
<bound method IndexOpsMixin.value_counts of 0       0
1       0
2       0
3       0
4       0
       ..
1020    1
1021    0
1022    0
1023    1
1024    0
Name: target, Length: 1025, dtype: int64>
```

In [27]:
```
data.target.value_counts().plot(kind='bar', color=['red', 'green'])
plt.title('Heart Disease')
plt.xlabel('0 No Disease, 1 Disease')
plt.ylabel('Amount')
plt.show()
```



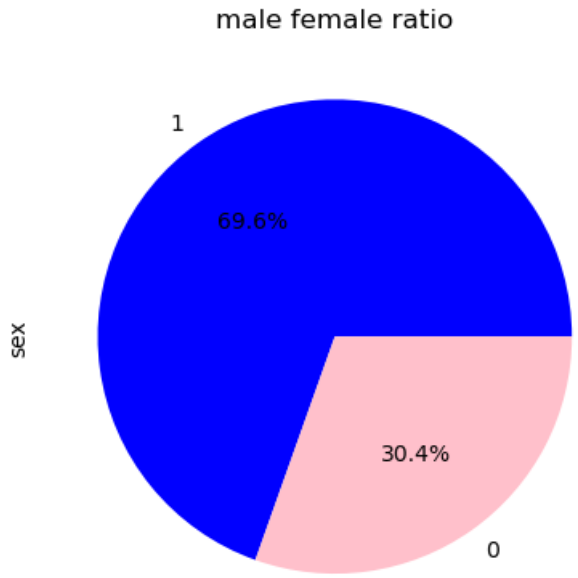
In [28]:
```
data.target.value_counts().plot(kind='pie', figsize=(8,6), colors=['red', 'green'], autopct='%1.1
plt.legend(["Disease", "No Disease"])
plt.show();
```



In [29]:
```
#2) People of which sex has most Heart Disease?
data.sex.value_counts()
```

Out[29]:
```
1    713
0    312
Name: sex, dtype: int64
```

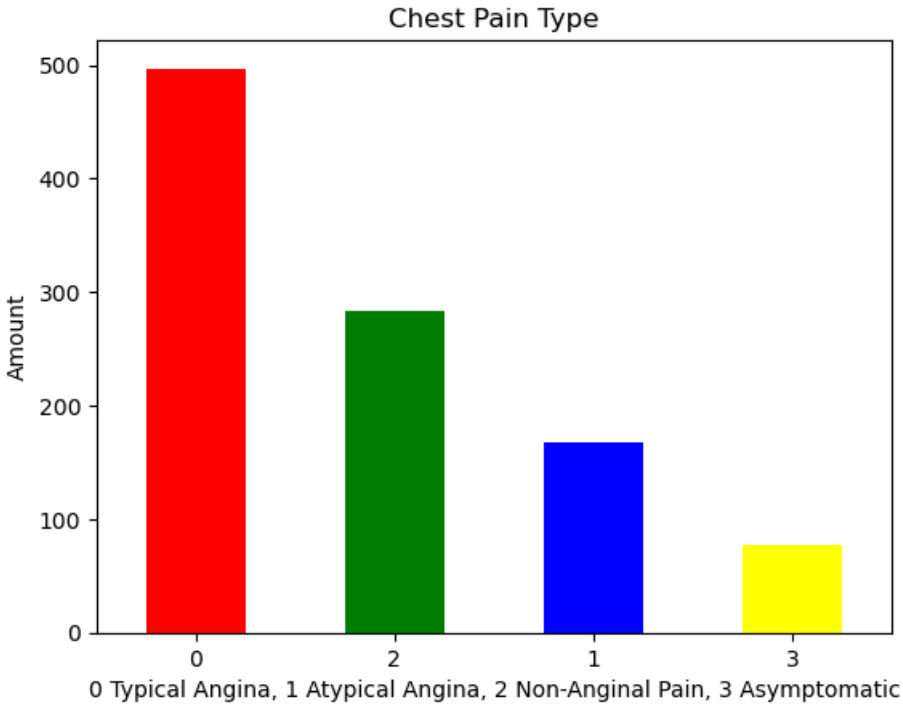
In [30]:
```
data.sex.value_counts().plot(kind='pie', colors=['blue', 'pink'], autopct='%1.1f%%')
plt.title('male female ratio')
plt.show()
```



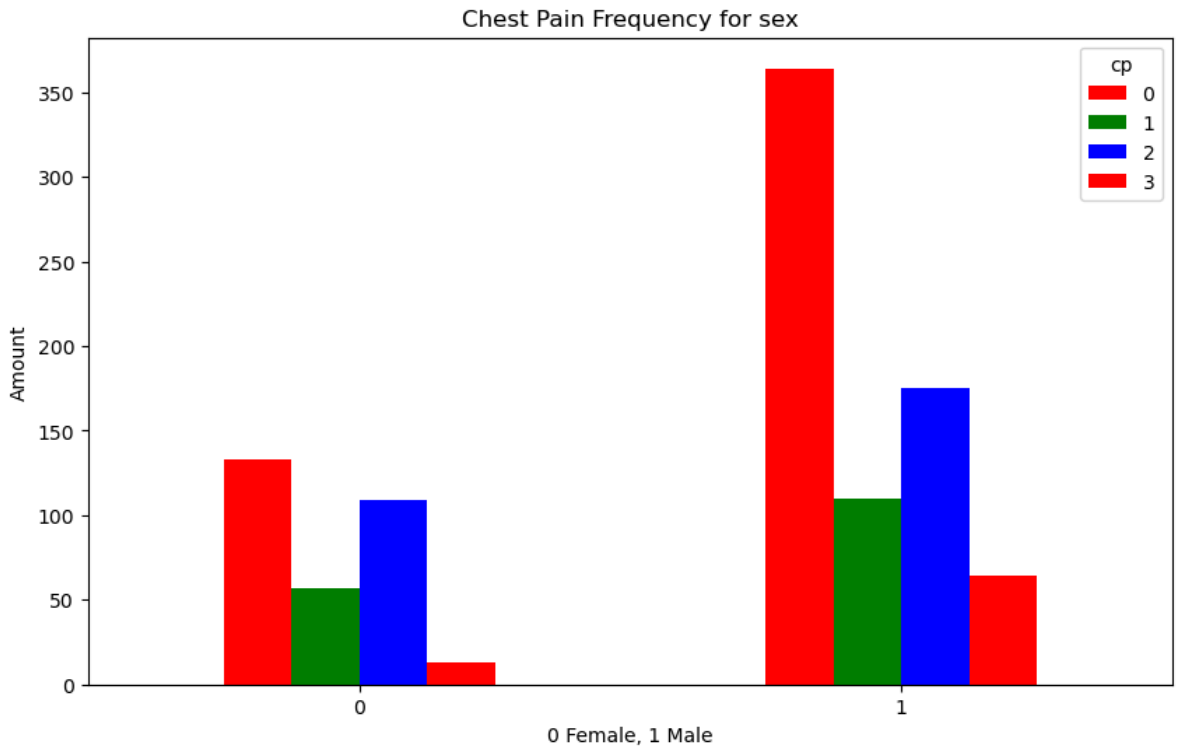
In [31]:
```
#3) People of which sex has which type of chest pain most?
data.cp.value_counts()
```

Out[31]:
```
0    497
2    284
1    167
3     77
Name: cp, dtype: int64
```

In [32]:
```
data.cp.value_counts().plot(kind='bar', color=['red', 'green', 'blue', 'yellow'])
plt.title('Chest Pain Type')
plt.xlabel('0 Typical Angina, 1 Atypical Angina, 2 Non-Anginal Pain, 3 Asymptomatic' )
plt.xticks(rotation=0)
plt.ylabel('Amount')
plt.show()
```



In [33]:
```
pd.crosstab(data.sex, data.cp).plot(kind='bar', figsize=(10,6), color=['red', 'green', 'blue'])
plt.title('Chest Pain Frequency for sex')
plt.xlabel('0 Female, 1 Male')
plt.xticks(rotation=0)
plt.ylabel('Amount')
plt.show()
```

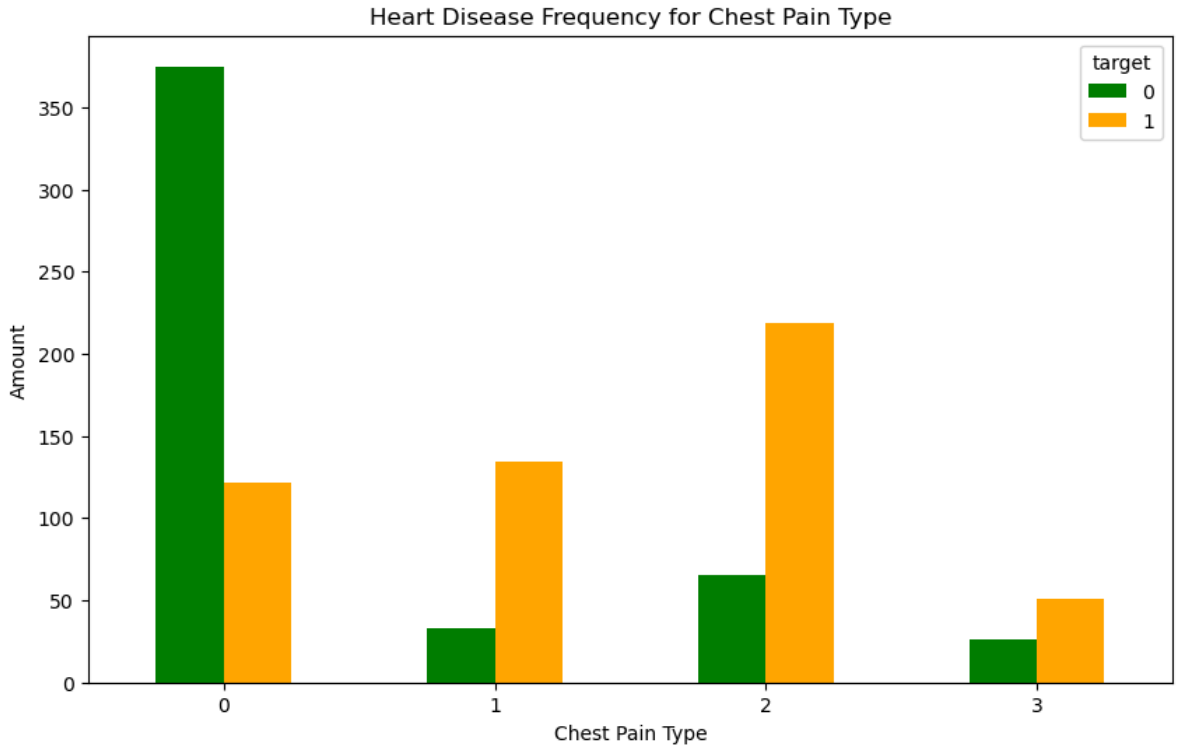


In [34]:
```
#4) People with which chest pain are most prone to have Heart disease?
pd.crosstab(data.cp, data.target)
```

Out[34]:

target	0	1
cp		
0	375	122
1	33	134
2	65	219
3	26	51

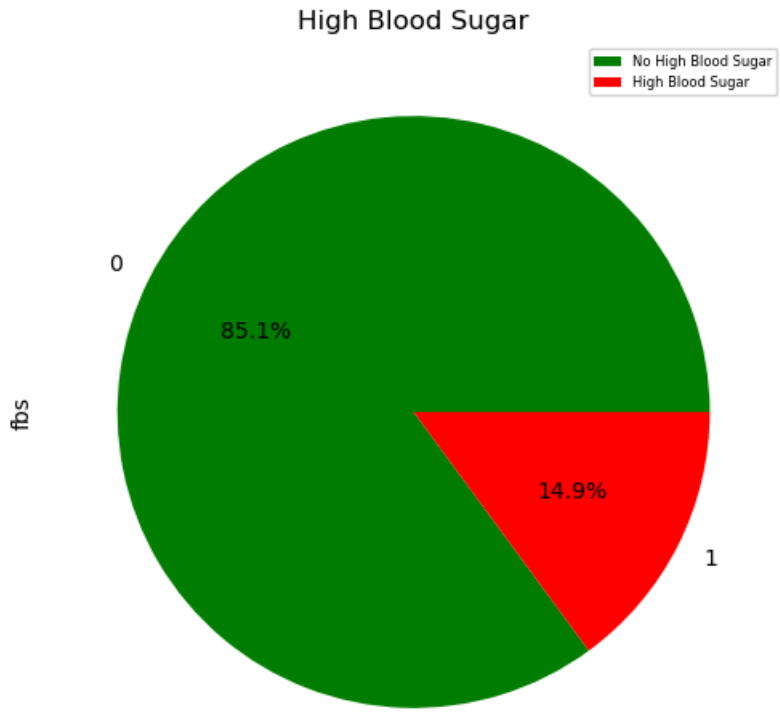
In [35]:
```
pd.crosstab(data.cp, data.target).plot(kind='bar', figsize=(10,6), color=['green', 'orange'])
plt.title('Heart Disease Frequency for Chest Pain Type')
plt.xlabel('Chest Pain Type')
plt.xticks(rotation=0)
plt.ylabel('Amount')
plt.show()
```



In [36]:
```
#5) How many people have high blood sugar how many people don't have high blood sugar?
data.fbs.value_counts()
```

Out[36]:
```
0    872
1    153
Name: fbs, dtype: int64
```

In [37]:
```
data.fbs.value_counts().plot(kind='pie', figsize=(8,6), colors=['green', 'red'],autopct='%1.1f%%'
plt.title('High Blood Sugar')
plt.legend(["No High Blood Sugar", "High Blood Sugar"], fontsize= 6)
plt.show()
```



In [38]:
```
# Binning Age
data['age_binned'] = pd.cut(data['age'], bins=[0, 30, 40, 50, 60, 70, 80, 90], labels=['0-30', '3

# Interaction Features
data['chol_trestbps'] = data['chol'] * data['trestbps']

# Polynomial Features
data['thalach_squared'] = data['thalach'] ** 2

# Display the first few rows to verify new features
print(data.head())
```

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

   ca  thal  target age_binned  chol_trestbps  thalach_squared
0   2     3       0      51-60          26500            28224
1   0     3       0      51-60          28420            24025
2   0     3       0      61-70          25230            15625
3   1     3       0      61-70          30044            25921
4   3     2       0      61-70          40572            11236
```

In [39]:

```
import numpy as np

# Log Transformation
data['log_chol'] = np.log1p(data['chol'])

# One-Hot Encoding for Categorical Variables
data = pd.get_dummies(data, columns=['cp', 'thal', 'age_binned'], drop_first=True)

# Display the first few rows to verify new features
print(data.head())
```

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

   ...  cp_3  thal_1  thal_2  thal_3  age_binned_31-40  age_binned_41-50  \
0  ...     0       0       0       1                 0                 0
1  ...     0       0       0       1                 0                 0
2  ...     0       0       0       1                 0                 0
3  ...     0       0       0       1                 0                 0
4  ...     0       0       1       0                 0                 0

   age_binned_51-60  age_binned_61-70  age_binned_71-80  age_binned_81-90
0                 1                 0                 0                 0
1                 1                 0                 0                 0
2                 0                 1                 0                 0
3                 0                 1                 0                 0
4                 0                 1                 0                 0

[5 rows x 27 columns]
```

In [40]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Drop the target column and standardize the features
X = data.drop(['target'], axis=1)
y = data['target']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA to retain 95% of variance
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)

# Print the explained variance ratio
print(pca.explained_variance_ratio_)
```

```
[0.18132376 0.12221852 0.08197057 0.06532111 0.06155777 0.05399166
 0.05222914 0.04882772 0.04083599 0.03962484 0.03793874 0.03512828
 0.03350385 0.03135557 0.02961297 0.02754249 0.02422306]
```

In [41]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel

# Train a Random Forest model to get feature importances
rf = RandomForestClassifier()
rf.fit(X, y)

# Select features based on importance
sfm = SelectFromModel(rf, threshold='mean', prefit=True)
X_important = sfm.transform(X)

# Important features DataFrame
important_features = pd.DataFrame(X_important, columns=X.columns[sfm.get_support()])

print(important_features.head())
print(rf.feature_importances_)
```

```
    age  trestbps   chol  thalach  exang  oldpeak   ca  chol_trestbps  \
0  52.0     125.0  212.0    168.0    0.0      1.0  2.0        26500.0
1  53.0     140.0  203.0    155.0    1.0      3.1  0.0        28420.0
2  70.0     145.0  174.0    125.0    1.0      2.6  0.0        25230.0
3  61.0     148.0  203.0    161.0    0.0      0.0  1.0        30044.0
4  62.0     138.0  294.0    106.0    0.0      1.9  3.0        40572.0

   thalach_squared  log_chol  thal_2  thal_3
0          28224.0  5.361292     0.0     1.0
1          24025.0  5.318120     0.0     1.0
2          15625.0  5.164786     0.0     1.0
3          25921.0  5.318120     0.0     1.0
4          11236.0  5.686975     1.0     0.0
[0.06291028 0.02339953 0.05161979 0.0499572  0.00637327 0.01082195
 0.08969755 0.05450207 0.09729296 0.03226371 0.10545347 0.05593978
 0.07213867 0.04956511 0.01212232 0.03609642 0.01481656 0.00436268
 0.08146559 0.05938065 0.00275995 0.0062614  0.00888739 0.01037981
 0.0015319  0.        ]

C:\Users\91778\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature name
s, but SelectFromModel was fitted without feature names
  warnings.warn(
```

In [42]:

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

# Split data for PCA features
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y, test_size=0.3, rand

# Split data for important features
X_train_imp, X_test_imp, y_train_imp, y_test_imp = train_test_split(important_features, y, test_s

# Model training with PCA features
model_pca = RandomForestClassifier()
model_pca.fit(X_train_pca, y_train_pca)
y_pred_pca = model_pca.predict(X_test_pca)
accuracy_pca = accuracy_score(y_test_pca, y_pred_pca)
print(f"Model Accuracy with PCA Features: {accuracy_pca}")

# Model training with important features
model_imp = RandomForestClassifier()
model_imp.fit(X_train_imp, y_train_imp)
y_pred_imp = model_imp.predict(X_test_imp)
accuracy_imp = accuracy_score(y_test_imp, y_pred_imp)
print(f"Model Accuracy with Important Features: {accuracy_imp}")
```

```
Model Accuracy with PCA Features: 0.9902597402597403
Model Accuracy with Important Features: 0.9902597402597403
```

In [43]:

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

# Split data for important features
X_train_imp, X_test_imp, y_train_imp, y_test_imp = train_test_split(important_features, y, test_s

# Define the parameter grid for Random Forest
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize the Random Forest model
rf_model = RandomForestClassifier()

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)

# Fit GridSearchCV
grid_search.fit(X_train_imp, y_train_imp)

# Get the best parameters and model
best_params = grid_search.best_params_
best_rf = grid_search.best_estimator_

# Evaluate the model on the test set
y_pred_imp = best_rf.predict(X_test_imp)
accuracy_imp = accuracy_score(y_test_imp, y_pred_imp)

print(f"Best Parameters: {best_params}")
print(f"Model Accuracy with Important Features: {accuracy_imp}")
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
Best Parameters: {'bootstrap': False, 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_spl
it': 2, 'n_estimators': 50}
Model Accuracy with Important Features: 0.9805194805194806
```

In [44]:

```
from sklearn.model_selection import GridSearchCV

# Hyperparameter tuning for the model with important features
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(estimator=RandomForestClassifier(), param_grid=param_grid, cv=3, n_jobs
grid_search.fit(X_train_imp, y_train_imp)

best_model = grid_search.best_estimator_
y_pred_optimized = best_model.predict(X_test_imp)
optimized_accuracy = accuracy_score(y_test_imp, y_pred_optimized)
print(f"Optimized Model Accuracy: {optimized_accuracy}")
```

```
Fitting 3 folds for each of 81 candidates, totalling 243 fits
Optimized Model Accuracy: 1.0
```