Exploratory Data Analysis of Heart Attack Dataset

Introduction

Cardiovascular diseases, including heart attacks, are a leading cause of mortality worldwide. Understanding the risk factors and patterns associated with heart attacks is crucial for effective prevention and treatment. In this Jupyter Notebook, we will perform an exploratory data analysis (EDA) on a dataset containing information related to heart attacks.

Dataset Overview

The dataset used in this analysis contains a comprehensive set of features related to patients and their medical history.

Data Source

The dataset used in this analysis was obtained from Heart Attack Analysis & Prediction Dataset, and it contains 303 records and 14 columns.

Columns :

- Age : Age of the patient
- Sex : Sex of the patient
- exang: exercise induced angina (1 = yes; 0 = no)
- ca: number of major vessels (0-3)
- cp : Chest Pain type chest pain type
- trtbps : resting blood pressure (in mm Hg)
- chol : cholestoral in mg/dl fetched via BMI sensor
- fbs : (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- rest_ecg : resting electrocardiographic results
- thalach : maximum heart rate achieved
- output : 0= less chance of heart attack 1= more chance of heart attack

Import Library

In [58]:

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
import warnings
warnings.filterwarnings('ignore')
```

Uploading Dataset

In [59]:

```
df = pd.read_csv("./heart.csv")
```

Exploratory Data Analysis

Dataset Shape

In [60]:
```
df.shape
```

Out[60]:
```
(303, 14)
```

First Few Rows of the Dataset

In [61]:
```
df.head()
```

Out[61]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Last Few Rows of the Dataset

In [62]:
```
df.tail()
```

Out[62]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

Column Names

In [63]:
```
df.columns
```

Out[63]:
```
Index(['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh',
       'exng', 'oldpeak', 'slp', 'caa', 'thall', 'output'],
      dtype='object')
```

Renaming the 'output' Column to 'hachance'

In [64]:
```
df.rename(columns={'output':'hachance'},inplace=True)
```

Data Types of Columns

In [65]:
```
df.dtypes
```

Out[65]:
```
age          int64
sex          int64
```

```
cp            int64
trtbps        int64
chol          int64
fbs           int64
restecg       int64
thalachh      int64
exng          int64
oldpeak     float64
slp           int64
caa           int64
thall         int64
hachance      int64
dtype: object
```

Summary Statistics

In [66]:
```
df.describe()
```

Out[66]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalach
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.6468
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.9051
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.0000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.5000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.0000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.0000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.0000

Checking for Duplicate Rows

In [67]:
```
df.duplicated()
```

Out[67]:
```
0      False
1      False
2      False
3      False
4      False
       ...
298    False
299    False
300    False
301    False
302    False
Length: 303, dtype: bool
```

Checking for Missing Values

In [68]:
```
df.isnull().sum()
```

Out[68]:
```
age         0
sex         0
cp          0
trtbps      0
chol        0
fbs         0
restecg     0
```

```
thalachh    0
exng        0
oldpeak     0
slp         0
caa         0
thall       0
hachance    0
dtype: int64
```

Exploring Features

In [69]:

```
for d in df.columns :
   print(d," ",df[d].unique())
```

```
age   [63 37 41 56 57 44 52 54 48 49 64 58 50 66 43 69 59 42 61 40 71 51 65 53
 46 45 39 47 62 34 35 29 55 60 67 68 74 76 70 38 77]
sex   [1 0]
cp   [3 2 1 0]
trtbps   [145 130 120 140 172 150 110 135 160 105 125 142 155 104 138 128 108 134
 122 115 118 100 124  94 112 102 152 101 132 148 178 129 180 136 126 106
 156 170 146 117 200 165 174 192 144 123 154 114 164]
chol   [233 250 204 236 354 192 294 263 199 168 239 275 266 211 283 219 340 226
 247 234 243 302 212 175 417 197 198 177 273 213 304 232 269 360 308 245
 208 264 321 325 235 257 216 256 231 141 252 201 222 260 182 303 265 309
 186 203 183 220 209 258 227 261 221 205 240 318 298 564 277 214 248 255
 207 223 288 160 394 315 246 244 270 195 196 254 126 313 262 215 193 271
 268 267 210 295 306 178 242 180 228 149 278 253 342 157 286 229 284 224
 206 167 230 335 276 353 225 330 290 172 305 188 282 185 326 274 164 307
 249 341 407 217 174 281 289 322 299 300 293 184 409 259 200 327 237 218
 319 166 311 169 187 176 241 131]
fbs   [1 0]
restecg   [0 1 2]
thalachh   [150 187 172 178 163 148 153 173 162 174 160 139 171 144 158 114 151 161
 179 137 157 123 152 168 140 188 125 170 165 142 180 143 182 156 115 149
 146 175 186 185 159 130 190 132 147 154 202 166 164 184 122 169 138 111
 145 194 131 133 155 167 192 121  96 126 105 181 116 108 129 120 112 128
 109 113  99 177 141 136  97 127 103 124  88 195 106  95 117  71 118 134
  90]
exng   [0 1]
oldpeak   [2.3 3.5 1.4 0.8 0.6 0.4 1.3 0.  0.5 1.6 1.2 0.2 1.8 1.  2.6 1.5 3.  2.4
 0.1 1.9 4.2 1.1 2.  0.7 0.3 0.9 3.6 3.1 3.2 2.5 2.2 2.8 3.4 6.2 4.  5.6
 2.9 2.1 3.8 4.4]
slp   [0 2 1]
caa   [0 2 1 3 4]
thall   [1 2 3 0]
hachance   [1 0]
```

We can conclude from that : the categorical features and the numerical ones

In [70]:

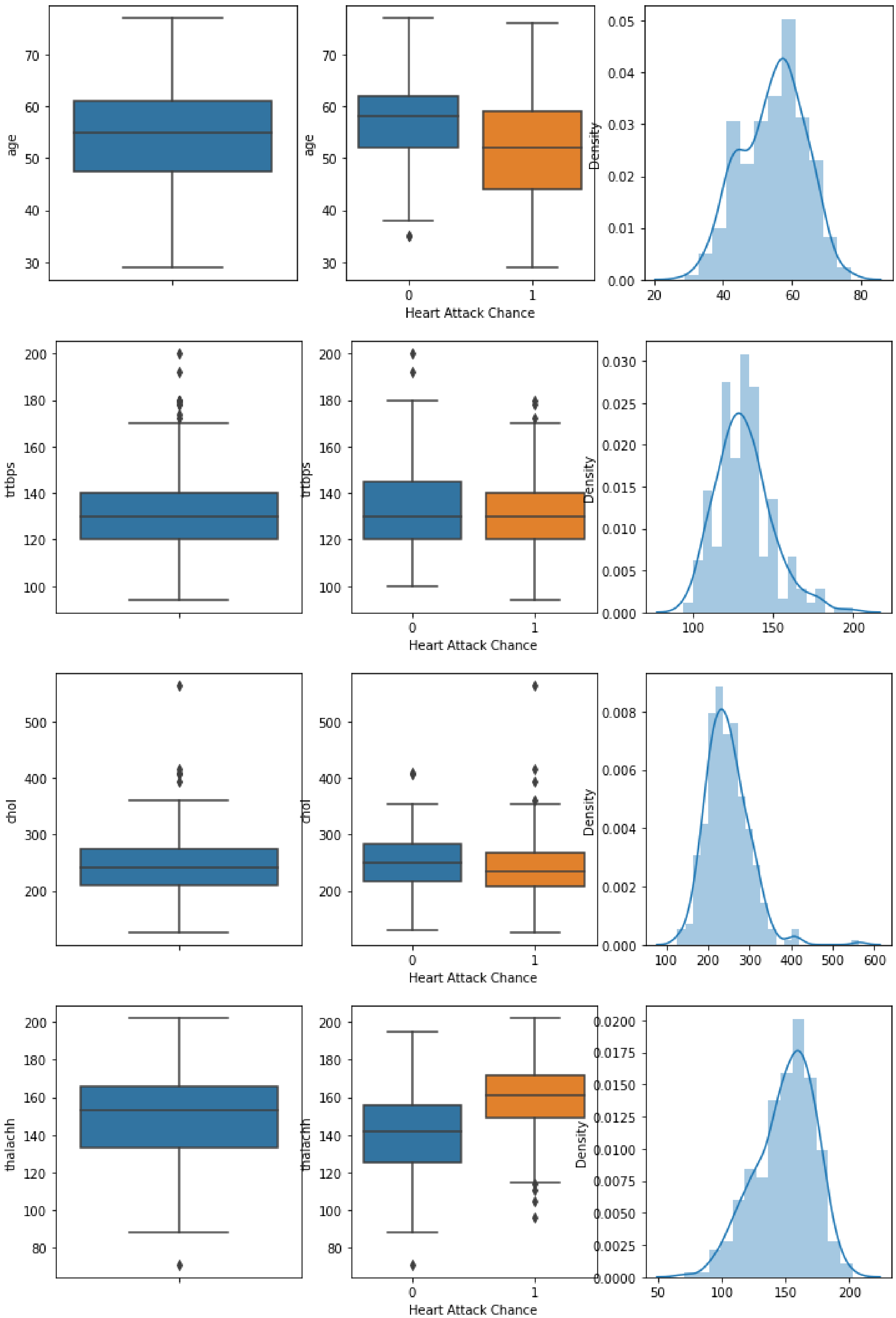
```
categorical_features = df[["thall","caa","slp","exng","restecg","fbs","cp","sex","ha
numerical_features = df.drop(categorical_features,axis=1)
```

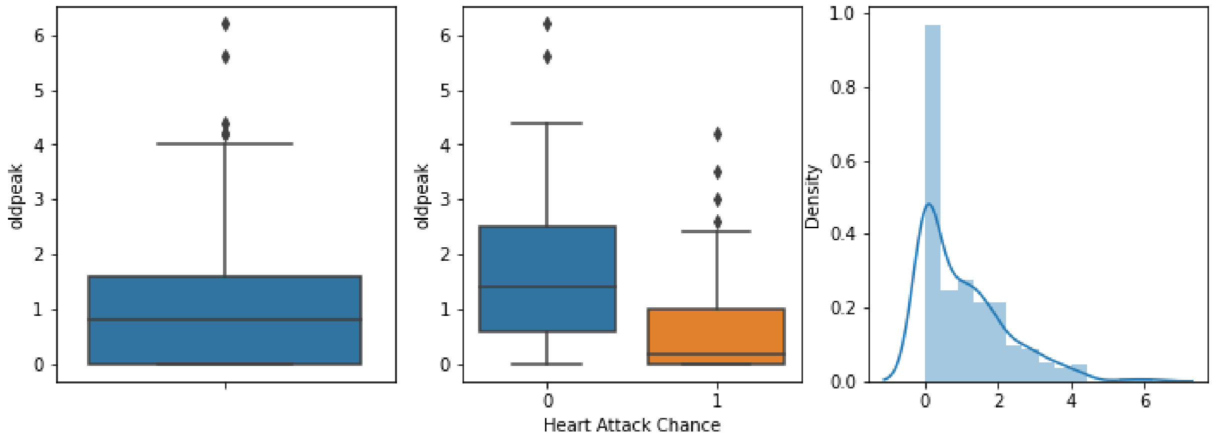
Exploring Numerical Features with Respect to Heart Attack Chance:

In [71]:

```
for i in numerical_features.columns:
    plt.figure(figsize=(12,4))

    plt.subplot(1,3,1)
    sns.boxplot(y=i,data=df)

    plt.subplot(1,3,2)
    sns.boxplot(x=df['hachance'],y=i,data=df)
    plt.xlabel('Heart Attack Chance')

    plt.subplot(1,3,3)
```

```
    sns.distplot(x=df[i])
    plt.show()
```





Age : Distribution closely resembles to a normal Distribution. Most of the patient are aged between 47 and 61

Resting blood presure : There are outlieres where trtbps is above 175 due to this outliers the distribution is slightly right-skewed

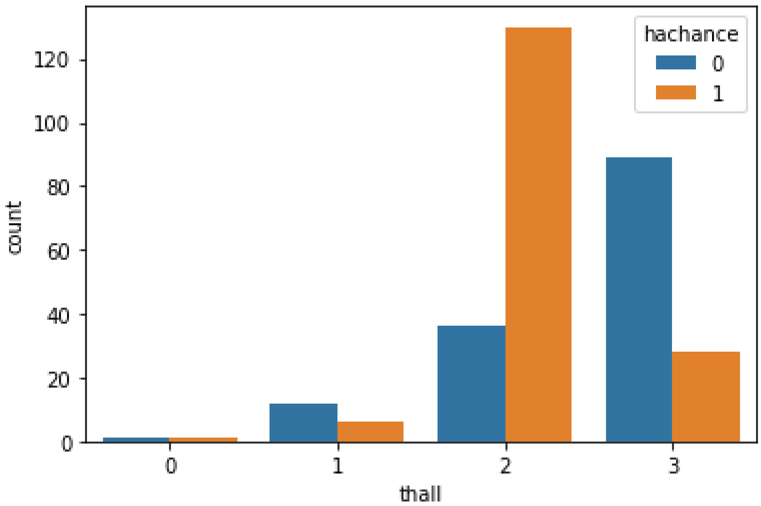
Cholestoral : There are outlieres due to this outliers the distribution is slightly right-skewed

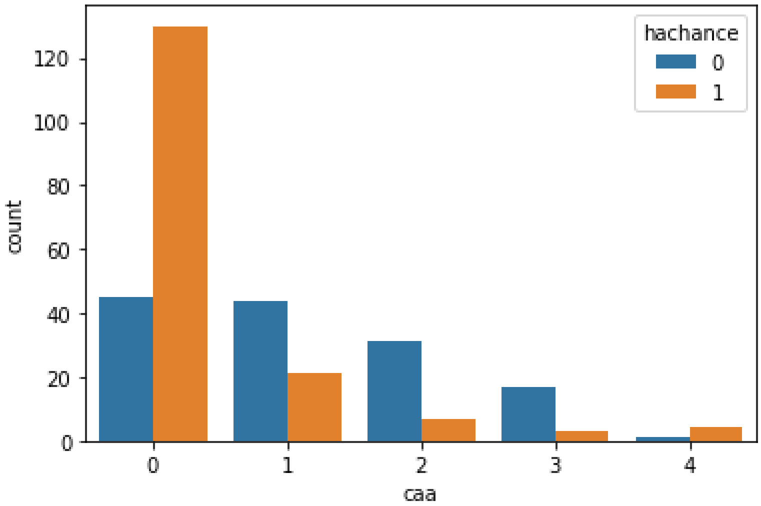
Maximum heart rate achieved : The distribution is left-sekewed due to the outliers under 90

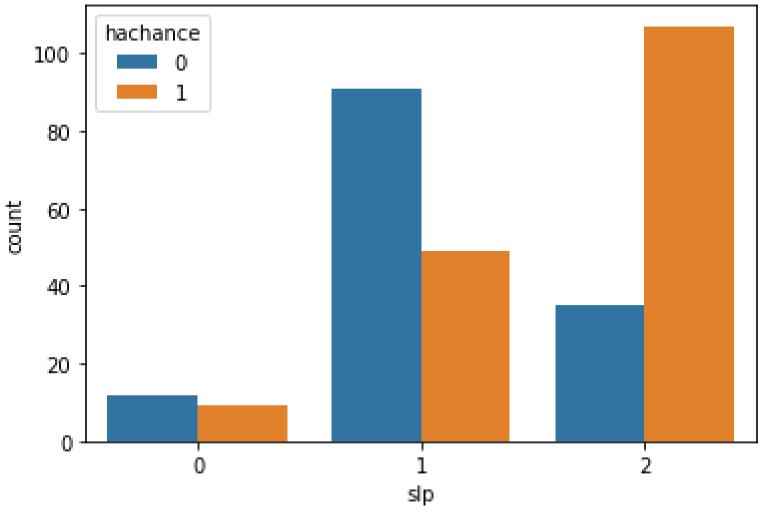
Old peak : There are outlieres where oldpeak is above 4 due to this outliers the distribution is right-skewed

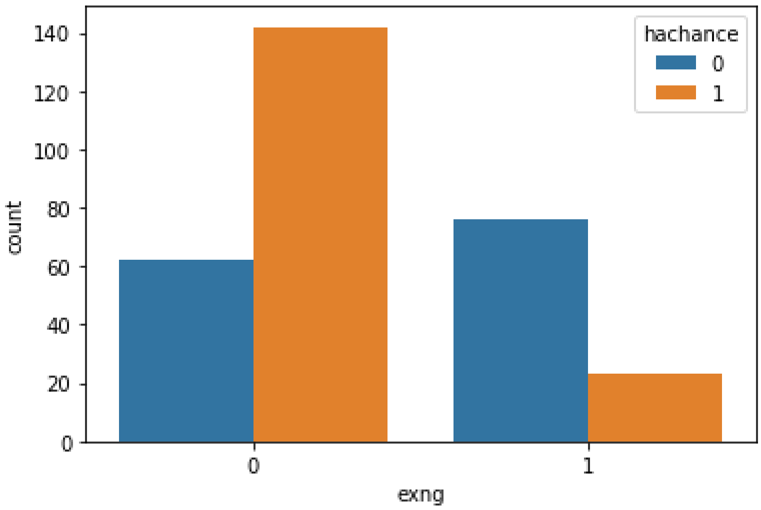
Exploring Categorical Features with Respect to Heart Attack Chance

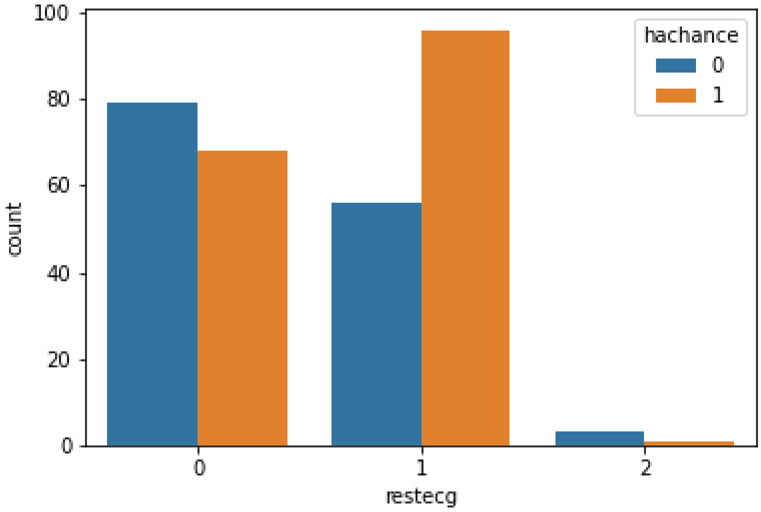
In [72]:
```
for i in categorical_features.columns:
    sns.countplot(x=i,data=df,hue="hachance")
    plt.show()
```

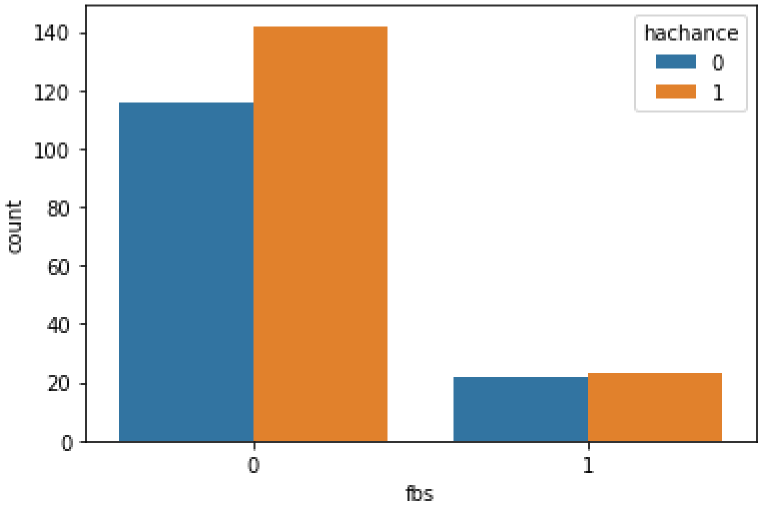


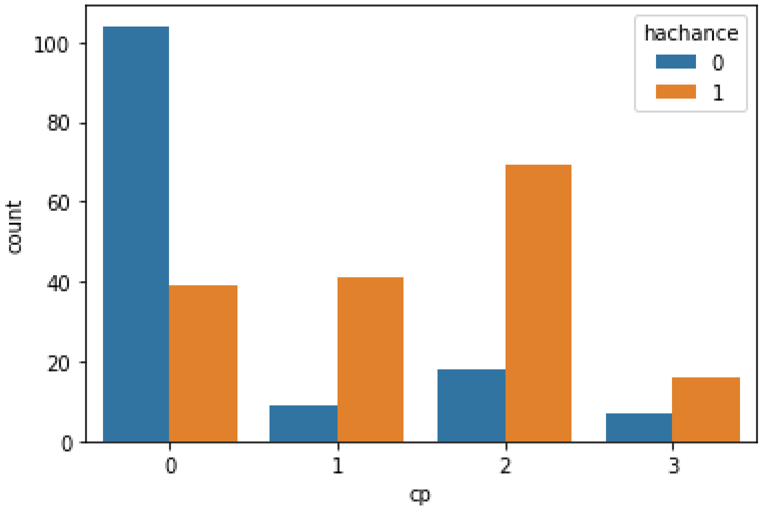

hachance
0
1
count
0
20
40
60
80
100
120
0
1
2
3
4
caa

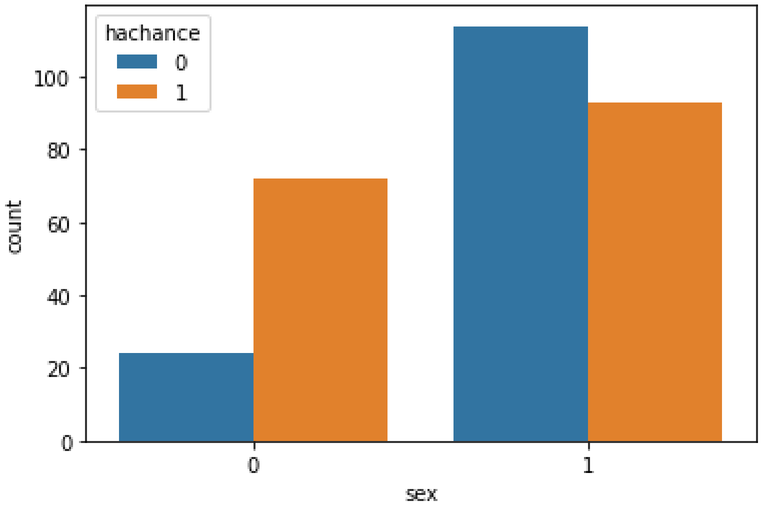

hachance
0
1
count
0
20
40
60
80
100
0
1
2
slp

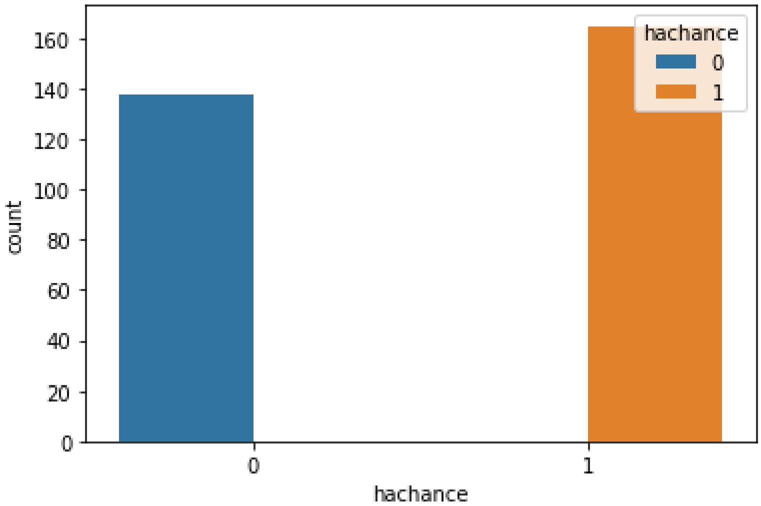

hachance
0
1
count
0
20
40
60
80
100
120
140
0
1
exng


hachance
0
1
count
100
80
60
40
20
0
0
1
2
restecg


hachance
0
1
count
140
120
100
80
60
40
20
0
0
1
fbs


hachance
0
1
count
100
80
60
40
20
0
0
1
2
3
cp





Data preprocessing

In [73]:

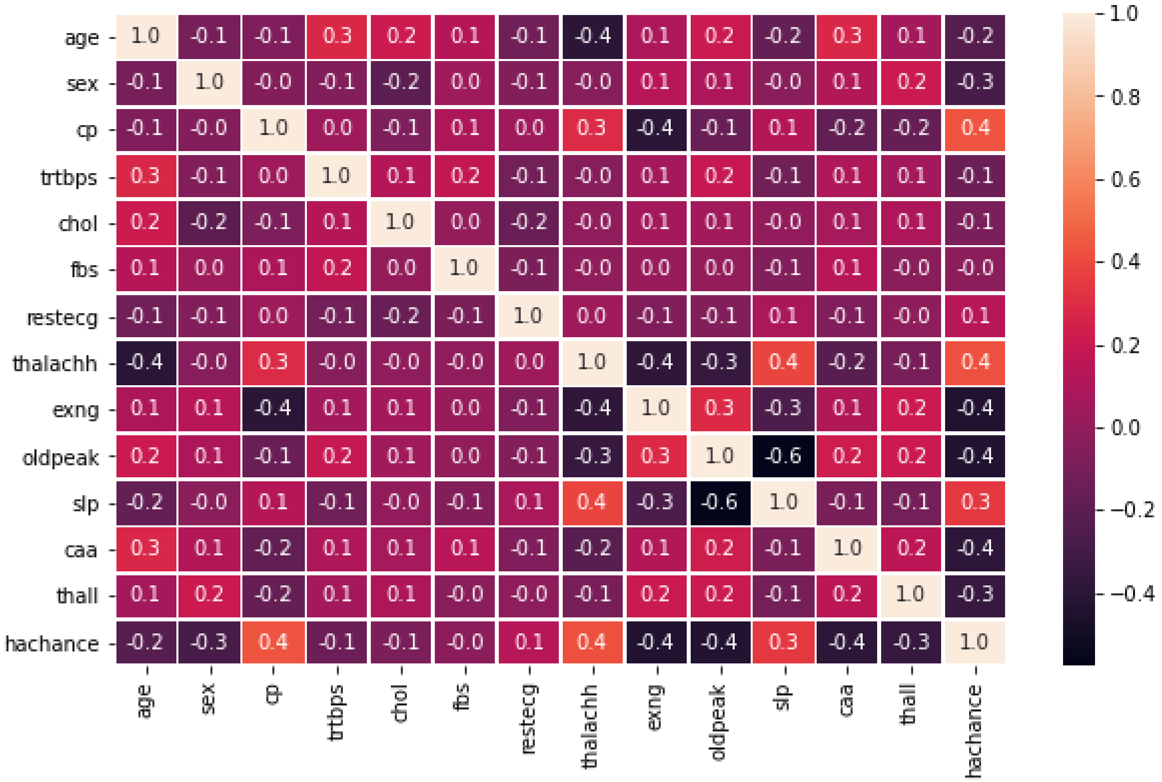
```
scaler = preprocessing.StandardScaler()

# Standardize the numerical columns
df[['age', 'trtbps', 'chol', 'thalachh', 'oldpeak']] = scaler.fit_transform(df[['age
```

Correlation Matrix Heatmap

In [74]:

```
plt.figure(figsize = (10,6))
dff = df.corr()
sns.heatmap(dff, annot = True, fmt = ".1f", linewidth= 0.7)
```

Out[74]: <AxesSubplot:>



Train - Test Split

In [75]:
```
X = df.drop('hachance',axis=1)
Y = df['hachance']
```

In [76]:
```
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.4,random_state=0)
```

Modeling

In [77]:
```
classifiers = [svm.SVC(),DecisionTreeClassifier(),LogisticRegression()]
results = []
means=[]
std =[]
for classifier in classifiers :
    results.append(cross_val_score(classifier,x_train,y_train,cv=10,n_jobs=4))
for result in results:
    means.append(result.mean())
    std.append(result.std())
res = pd.DataFrame({"CrossValMeans":means,"CrossValerrors": std,"Algorithm":["SVC",
res
```

Out[77]:

	CrossValMeans	CrossValerrors	Algorithm
0	0.812865	0.103708	SVC
1	0.740058	0.117108	DecisionTree
2	0.834503	0.085380	LogisticRegression

Best Model Selection

In [78]:
```
regression_model = LogisticRegression(solver='liblinear')
regression_model.fit(x_train,y_train)
```

```
yhat = regression_model.predict(x_test)
```

Confusion Matrix and Visualization

In [53]:

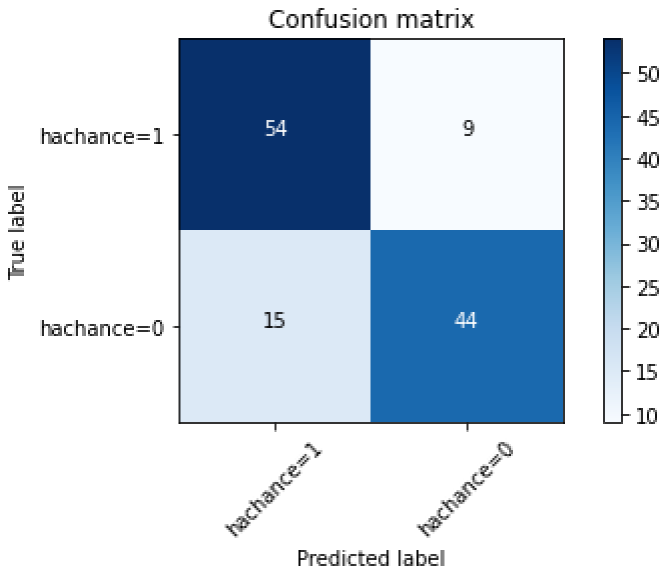
```
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
print(confusion_matrix(y_test, yhat, labels=[1,0]))
```

```
[[54  9]
 [15 44]]
```

In [54]:

```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, yhat, labels=[1,0])
np.set_printoptions(precision=2)


# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['hachance=1','hachance=0'],normalize= Fal
```

```
Confusion matrix, without normalization
[[54  9]
 [15 44]]
```


Confusion matrix
54
9
15
44
hachance=1
hachance=0
True label
hachance=1
hachance=0
Predicted label
50
45
40
35
30
25
20
15
10