```
!pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.6-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.6
```

```
#import all essential libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

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

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
heart_disease = fetch_ucirepo(id=45)

# data (as pandas dataframes)
X = heart_disease.data.features
y = heart_disease.data.targets

# metadata
print(heart_disease.metadata)

# variable information
print(heart_disease.variables)
```

```
{'uci_id': 45, 'name': 'Heart Disease', 'repository_url': 'https://archive.ics.uci.edu/dataset/45/heart+disease', 'data_url': 'https://a
        name     role         type demographic  \
0        age  Feature      Integer         Age
1        sex  Feature  Categorical         Sex
2         cp  Feature  Categorical        None
3   trestbps  Feature      Integer        None
4       chol  Feature      Integer        None
5        fbs  Feature  Categorical        None
6    restecg  Feature  Categorical        None
7    thalach  Feature      Integer        None
8      exang  Feature  Categorical        None
9    oldpeak  Feature      Integer        None
10     slope  Feature  Categorical        None
11        ca  Feature      Integer        None
12      thal  Feature  Categorical        None
13       num   Target      Integer        None

                                       description  units missing_values
0                                             None  years             no
1                                             None   None             no
2                                             None   None             no
3   resting blood pressure (on admission to the ho...  mm Hg             no
4                                 serum cholestoral  mg/dl             no
5                   fasting blood sugar > 120 mg/dl   None             no
6                                             None   None             no
7                       maximum heart rate achieved   None             no
8                           exercise induced angina   None             no
9   ST depression induced by exercise relative to ...   None             no
10                                            None   None             no
11  number of major vessels (0-3) colored by flour...   None            yes
12                                            None   None            yes
13                      diagnosis of heart disease   None             no
```

```
dataset = pd.read_csv('https://archive.ics.uci.edu/static/public/45/data.csv')
dataset.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0
1	67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0
2	67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.0
3	37	1	3	130	250	0	0	187	0	3.5	3	0.0	3.0
4	41	0	2	130	204	0	2	172	0	1.4	1	0.0	3.0

Exploratory Data Analysis

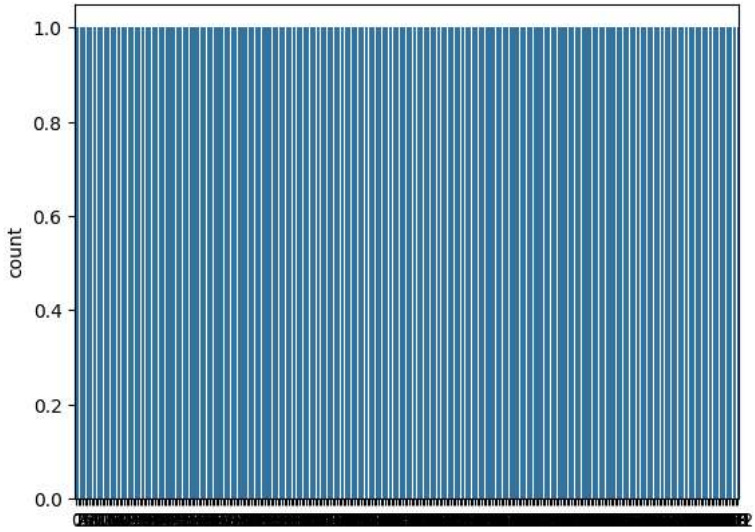
```
y = dataset["num"]

sns.countplot(y)
```

```
target_temp = dataset.num.value_counts()

print(target_temp)
```

```
num
0    164
1     55
2     36
3     35
4     13
Name: count, dtype: int64
```




```
print("Percentage of patience without heart problems: "+str(round(target_temp[0]*100/303,2)))
print("Percentage of patience with heart problems: "+str(round(target_temp[1]*100/303,2)))
```

```
Percentage of patience without heart problems: 54.13
Percentage of patience with heart problems: 18.15
```

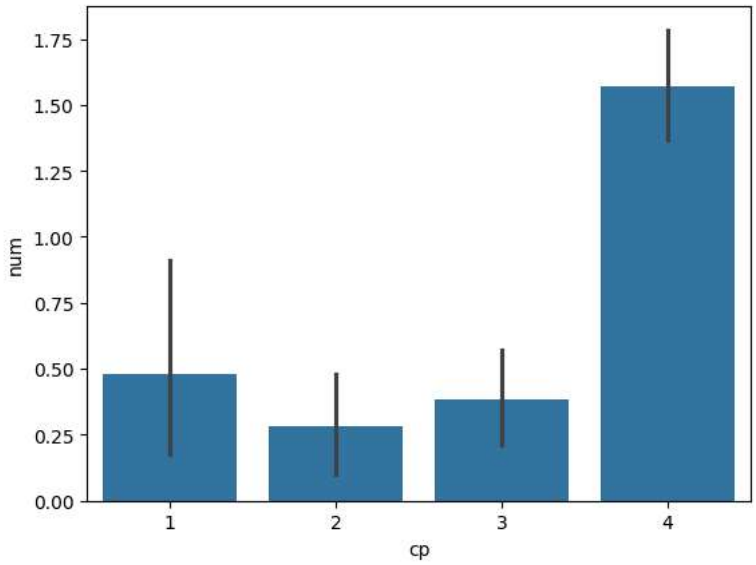
Analysing the 'Chest Pain Type' feature

```
dataset["cp"].unique()
```

```
array([1, 4, 3, 2])
```

```
sns.barplot(x=dataset["cp"],y=y)
```

```
<Axes: xlabel='cp', ylabel='num'>
```



Analysing the FBS feature

```
dataset["fbs"].describe()
```

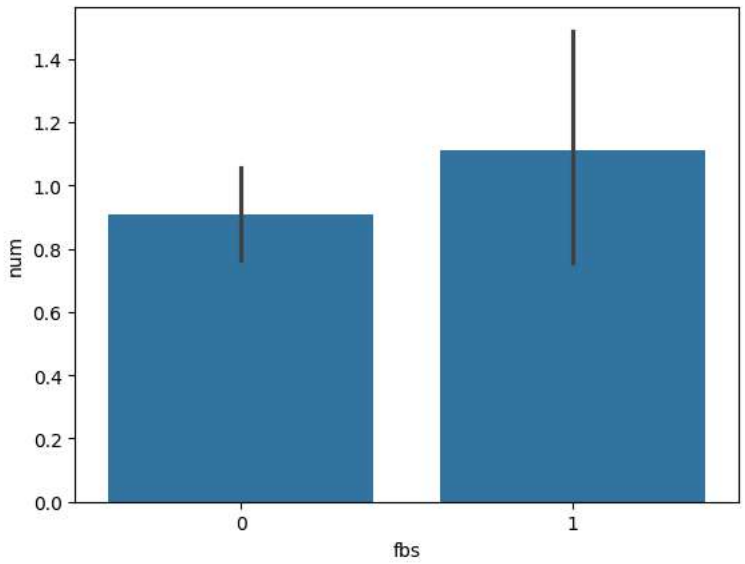
```
count    303.000000
mean       0.148515
std        0.356198
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        1.000000
Name: fbs, dtype: float64
```

```
dataset["fbs"].unique()
```

```
array([1, 0])
```

```
sns.barplot(x=dataset["fbs"],y=y)
```

```
<Axes: xlabel='fbs', ylabel='num'>
```



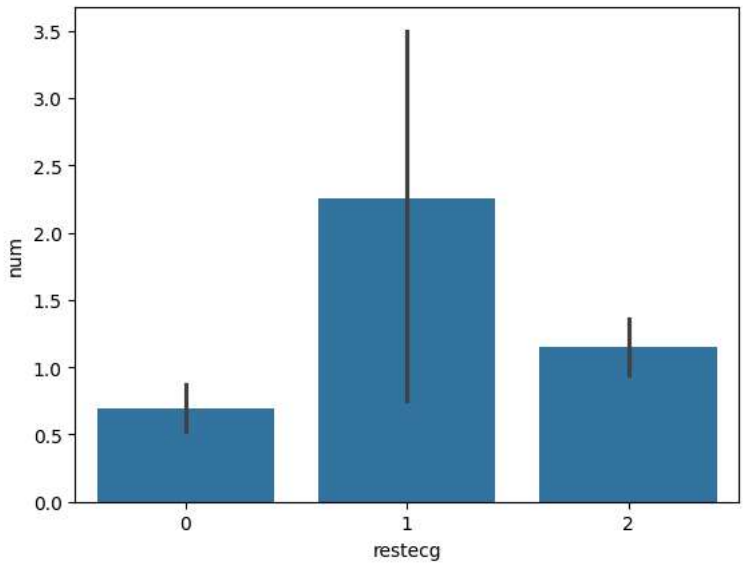
Analysing the restecg feature

```
dataset["restecg"].unique()
```

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

```
sns.barplot(x=dataset["restecg"],y=y)
```

```
<Axes: xlabel='restecg', ylabel='num'>
```



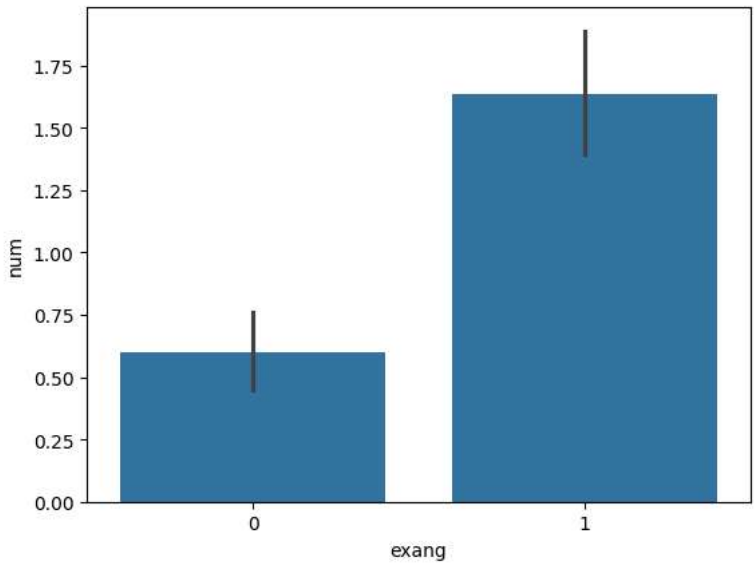
Analysing the 'exang' feature

```
dataset["exang"].unique()
```

```
array([0, 1])
```

```
sns.barplot(x=dataset["exang"],y=y)
```

```
<Axes: xlabel='exang', ylabel='num'>
```



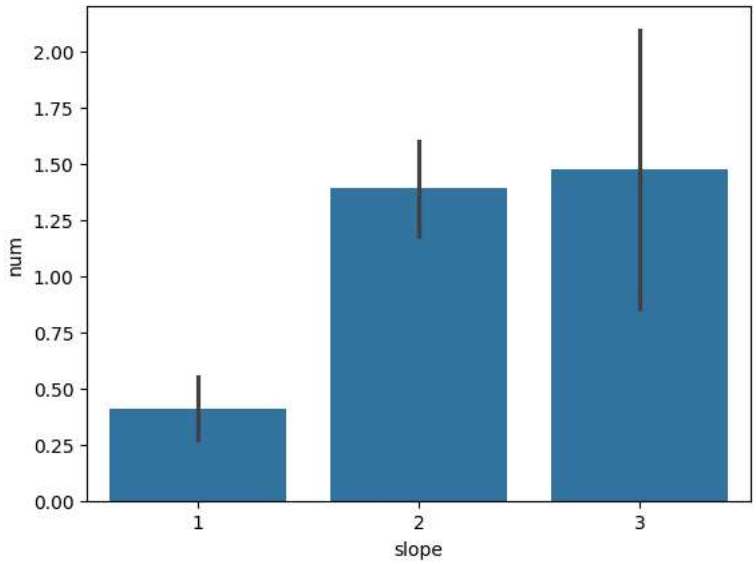
Analysing the Slope feature

```
dataset["slope"].unique()
```

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

```
sns.barplot(x=dataset["slope"],y=y)
```

```
<Axes: xlabel='slope', ylabel='num'>
```



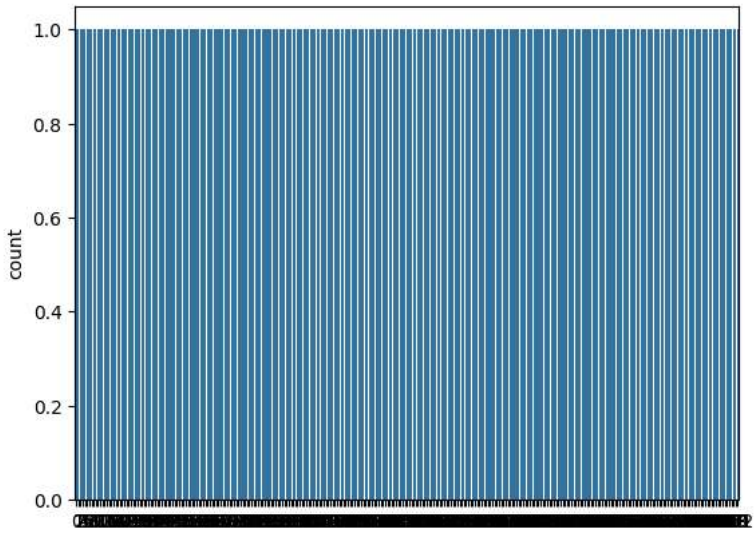
Analysing the 'ca' feature

```
dataset["ca"].unique()
```

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

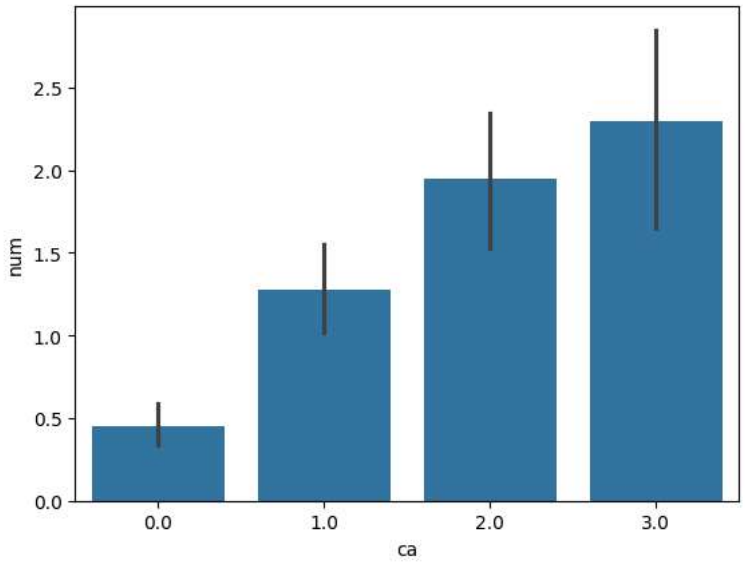
```
sns.countplot(dataset["ca"])
```

```
<Axes: ylabel='count'>
```




```
sns.barplot(x=dataset["ca"],y=y)
```

```
<Axes: xlabel='ca', ylabel='num'>
```



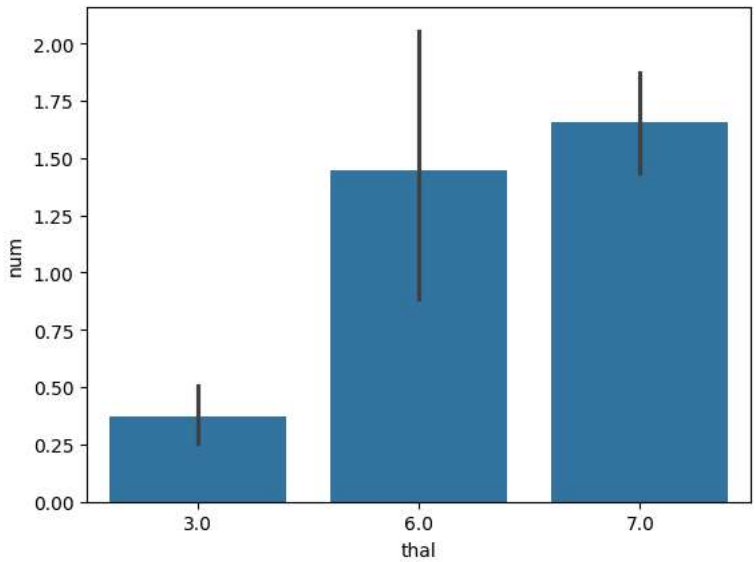
Analysing the 'thal' feature

```
dataset["thal"].unique()
```

```
array([ 6.,  3.,  7., nan])
```

```
sns.barplot(x=dataset["thal"],y=y)
```

```
<Axes: xlabel='thal', ylabel='num'>
```



Train Test split

```
from sklearn.model_selection import train_test_split

predictors = dataset.drop("num",axis=1)
target = dataset["num"]

X_train,X_test,Y_train,Y_test = train_test_split(predictors,target,test_size=0.20,random_state=0)
```

```
X_train.shape
```

```
(242, 13)
```

```
X_test.shape
```

```
(61, 13)
```

```
Y_train.shape
```

```
(242,)
```

```
Y_test.shape
```

```
(61,)
```

Model Fitting

```
from sklearn.metrics import accuracy_score
```

Logistic Regression

```
import pandas as pd
from sklearn.linear_model import LogisticRegression

missing_values = X_train.isnull().sum()
print(missing_values)
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          3
thal        2
dtype: int64
```

```
X_train = X_train.dropna()
Y_train = Y_train[X_train.index]

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="mean")
X_train = imputer.fit_transform(X_train)

lr = LogisticRegression()
lr.fit(X_train, Y_train)
```

```
LogisticRegression()
```

```
X_test.isnull().sum().sum()
```

```
1
```

```
X_test = X_test.dropna()

from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="mean")
X_test = imputer.fit_transform(X_test)

Y_pred_lr = lr.predict(X_test)

Y_pred_lr.shape
```

```
(57,)
```

```
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.4.0)
```

```
from sklearn.metrics import accuracy_score
```

```
print(Y_pred_lr.shape)
print(Y_test.shape)
```

```
(57,)
(61,)
```

```
if Y_pred_lr.shape[0] > Y_test.shape[0]:
    Y_pred_lr = Y_pred_lr[:Y_test.shape[0]]
else:
    Y_test = Y_test[:Y_pred_lr.shape[0]]
```

```
score_lr = round(accuracy_score(Y_pred_lr,Y_test)*100,2)
```

```
print("The accuracy score achieved using Logistic Regression is: "+str(score_lr)+" %")
```

```
The accuracy score achieved using Logistic Regression is: 61.4 %
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()

nb.fit(X_train,Y_train)

Y_pred_nb = nb.predict(X_test)
```

```
Y_pred_nb.shape
```

```
(60,)
```

```
print(len(Y_pred_nb))
print(len(Y_test))
```

```
60
57
```

```
# Truncate the longer array
if len(Y_pred_nb) > len(Y_test):
    Y_pred_nb = Y_pred_nb[:len(Y_test)]
elif len(Y_test) > len(Y_pred_nb):
    Y_test = Y_test[:len(Y_pred_nb)]

# Calculate the accuracy score
score_nb = round(accuracy_score(Y_pred_nb, Y_test) * 100, 2)

# Print the accuracy score
print("The accuracy score achieved using Naive Bayes is: " + str(score_nb) + " %")
```

```
The accuracy score achieved using Naive Bayes is: 45.61 %
```

SVM

```
from sklearn import svm

sv = svm.SVC(kernel='linear')

sv.fit(X_train, Y_train)

Y_pred_svm = sv.predict(X_test)
```

```
Y_pred_svm.shape
```

```
(60,)
```

```
from sklearn.metrics import accuracy_score
```

```
print(Y_pred_svm.shape)
print(Y_test.shape)
```

```
(60,)
(57,)
```

```
if len(Y_pred_svm) > len(Y_test):
    Y_pred_svm = Y_pred_svm[:len(Y_test)]
elif len(Y_test) > len(Y_pred_svm):
    Y_test = Y_test[:len(Y_pred_nb)]

# Calculate the accuracy score
score_svm = round(accuracy_score(Y_pred_svm, Y_test) * 100, 2)

# Print the accuracy score
print("The accuracy score achieved using SVM is: " + str(score_svm) + " %")
```

```
The accuracy score achieved using SVM is: 54.39 %
```

K Nearest Neighbors

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train,Y_train)
Y_pred_knn=knn.predict(X_test)
```

```
Y_pred_knn.shape
```

```
(60,)
```

```
if len(Y_pred_knn) > len(Y_test):
    Y_pred_knn = Y_pred_knn[:len(Y_test)]
elif len(Y_test) > len(Y_pred_knn):
    Y_test = Y_test[:len(Y_pred_knn)]

# Calculate the accuracy score
score_knn = round(accuracy_score(Y_pred_knn, Y_test) * 100, 2)

# Print the accuracy score
print("The accuracy score achieved using KNN is: " + str(score_knn) + " %")
```

```
The accuracy score achieved using KNN is: 54.39 %
```

Decision Tree

```
print(Y_pred_dt.shape)
print(Y_test.shape)
```

```
(60,)
(57,)
```

```
difference = Y_pred_dt.shape[0] - Y_test.shape[0]
print(difference)
```

```
3
```

```
Y_pred_dt = Y_pred_dt[:Y_test.shape[0]]


current_accuracy = round(accuracy_score(Y_pred_dt,Y_test)*100,2)


print(Y_pred_dt.shape)
```

```
(57,)
```

```
score_dt = round(accuracy_score(Y_pred_dt,Y_test)*100,2)

print("The accuracy score achieved using Decision Tree is: "+str(score_dt)+" %")
```

```
The accuracy score achieved using Decision Tree is: 56.14 %
```

XGBoost

```
import xgboost as xgb
from sklearn.metrics import accuracy_score

xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
xgb_model.fit(X_train, Y_train)

Y_pred_xgb = xgb_model.predict(X_test)


print(Y_pred_xgb.shape)
print(Y_test.shape)
```

```
(60,)
(57,)
```

```
Y_pred_xgb.shape
```

```
(60,)
```

```
print(f"Length of Y_pred_xgb: {len(Y_pred_xgb)}")
print(f"Length of Y_test: {len(Y_test)}")
```

```
Length of Y_pred_xgb: 9
Length of Y_test: 10
```

```
Y_pred_xgb = Y_pred_xgb[:-1]


import numpy as np

# Convert the lists to numpy arrays
Y_pred_xgb = np.array(Y_pred_xgb)
Y_test = np.array(Y_test)

# Check the shape of the arrays
print(Y_pred_xgb.shape)
print(Y_test.shape)

# Now the code will execute without errors
```