Project Report

Heart Attack Analysis & Prediction

GitHub URL

https://github.com/ajaymalik8/UCDPA_AjayMalik.git

Artifact List

Python Code	Project_Report_AjayMalik_UCDPA_BatchC.py
Jupyter Notebook	Project_Report_AjayMalik_UCDPA_BatchC.ipynb
Jupyter Notebook (PDF)	Project_Report_AjayMalik_UCDPA_BatchC - Jupyter Notebook.pdf
Project Report	Project_Report_AjayMalik_UCDPA_BatchC.pdf

Abstract

Cardiovascular diseases (CVDs) are the leading cause of death globally. The number of heart attacks is rising in India as well. I selected this data set to analyse the vital and sound an early warning to lessen its impact.

In this Study, we will explore...

- The heart disease dataset using exploratory data analysis (EDA)
- Exercise with ML algorithms for prediction (modelling)

Introduction

The phrase "heart disease" is a general one that refers to various illnesses and ailments that affect the heart and circulatory system. They are also known as cardiovascular illnesses. It is a major cause of disability all around the world. Since the heart is one of the body's most important organs, ailments that affect it also impact other organs and body parts. Heart disorders come in a variety of shapes and sorts. The most frequent ones result in heart failure and heart attacks by narrowing or blocking the coronary arteries, altering the heart's valves, growing the size of the heart, among other effects."

Dataset

https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset

Implementation Process

1. Importing data: For On-boarding of data, I have downloaded it from Kaggle as CSV and start processing data from csv file with below code.

Importing Data for analysis

```
In [7]: ds_heart = pd.read_csv("Data/heart.csv")
```

```
In [8]: #Printing Dataset Shape
        print("\nDateset Shape is : ",ds_heart.shape)

        Dateset Shape is :  (303, 14)
```

Let's Understand Our Data

```
In [9]: ds_heart.sample()
```

Out[9]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0

2. Understanding Our Data:

```
In [9]: ds_heart.sample()
```

Out[9]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0

```
In [10]: ds_heart.describe()
```

Out[10]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000

Total 303 records with 1 Duplicate. Cleaning duplicate record.

```
In [11]: #Columns List
         ds_heart.columns

Out[11]: Index(['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh',
                'exng', 'oldpeak', 'slp', 'caa', 'thall', 'output'],
               dtype='object')
```

```
In [12]: #List of Numeric Columns
         numeric_columns = [column for column in ds_heart.columns if (ds_heart[column].dtype == 'float64' or ds_heart[column].dtype ==
         print(numeric_columns)

         ['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh', 'exng', 'oldpeak', 'slp', 'caa', 'thall', 'output']
```

```
In [13]: #Duplicate Values
         ds_heart.duplicated().sum()

Out[13]: 1
```

```
In [14]:  #Removing duplicate value
          ds_heart.drop_duplicates(inplace=True)

In [15]:  #Total Records 303 unique records 302
          #Printing Dataset Shape
          print("\nDataset Shape is : ",ds_heart.shape,"(Unqiue Records)\n")

          Dataset Shape is :  (302, 14) (Unqiue Records)
```

3. Data Dictionary:

About this dataset		
1	Age	Age of the patient
2	Sex	Sex of the patient
3	exang	Exercise induced angina (1 = yes; 0 = no)
4	caa	Number of major vessels (0-3)
5	cp	Chest Pain type (1: typical angina, 2: atypical angina 3: non-anginal pain, 4: asymptomatic)
6	trtbps	Resting blood pressure (in mm Hg)
7	chol	Cholestoral in mg/dl fetched via BMI sensor
8	fbs	(Fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
9	rest_ecg	Resting electrocardiographic results (0 : normal, 1 : having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), 2 : showing probable or definite left ventricular hypertrophy by Estes' criteria)
10	thalach	Maximum heart rate achieved
11	oldpeak	Previous peak
12	slp	Slope
13	thall	Thal rate (Stress Test)

4. Data Cleaning and Manipulations
    a. Rename Column for better understanding

```
In [16]:  ds_heart.columns

Out[16]:  Index(['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh',
                 'exng', 'oldpeak', 'slp', 'caa', 'thall', 'output'],
                dtype='object')

In [17]:  ## Rename few columns to understand
          ds_heart.rename(columns={'output': 'attack',
                                   'thall': 'stresstest',
                                   'caa':'numberofmajorvessels',
                                   'cp':'chestpaintype',
                                   'exng':'exerciseinducedangina',
                                   'restecg':'restingecg',
                                   'fbs':'fastingbloodsugar',
                                   'trtbps':'restingbloodpressure',
                                   'thalachh':'maxheartrateachieved',
                                   'slp':'slope',
                                   'chol':'cholestoral'}, inplace=True)
          ds_heart.columns

Out[17]:  Index(['age', 'sex', 'chestpaintype', 'restingbloodpressure', 'cholestoral',
                 'fastingbloodsugar', 'restingecg', 'maxheartrateachieved',
                 'exerciseinducedangina', 'oldpeak', 'slope', 'numberofmajorvessels',
                 'stresstest', 'attack'],
                dtype='object')
```

b. Finding Missing Values: No Missing values in data

```
In [18]:  #finding Missing Values
          pd.options.display.max_rows = 15
          print(ds_heart.isnull().sum())
          pd.options.display.max_rows = 5

          age                       0
          sex                       0
          chestpaintype             0
          restingbloodpressure      0
          cholestoral               0
          fastingbloodsugar         0
          restingecg                0
          maxheartrateachieved      0
          exerciseinducedangina     0
          oldpeak                   0
          slope                     0
          numberofmajorvessels      0
          stresstest                0
          attack                    0
          dtype: int64
```

c. Validating Valid values: Validating using group functions. Some sample pasted below.

```
In [19]:  #chest pain type: chest pain type
          # 0: typical angina
          # 1: atypical angina
          # 2: non-anginal pain
          # 3: asymptomatic
          #Validating Values
          ds_heart.groupby(['chestpaintype'])['chestpaintype'].count()

Out[19]:  chestpaintype
          0    143
          1     50
          2     86
          3     23
          Name: chestpaintype, dtype: int64

In [20]:  #fasting blood sugar > 120 mg/dl
          #1 = true;
          #0 = false
          #Validating Values
          ds_heart.groupby(['fastingbloodsugar'])['fastingbloodsugar'].count()

Out[20]:  fastingbloodsugar
          0    257
          1     45
          Name: fastingbloodsugar, dtype: int64
```

```
In [33]:  #Max Heart Rate Achieved
          #ds_heart.groupby(['maxheartrateachieved'])['maxheartrateachieved'].count()
          np.unique(ds_heart['maxheartrateachieved'])

Out[33]:  array([ 71,  88,  90,  95,  96,  97,  99, 103, 105, 106, 108, 109, 111,
                 112, 113, 114, 115, 116, 117, 118, 120, 121, 122, 123, 124, 125,
                 126, 127, 128, 129, 130, 131, 132, 133, 134, 136, 137, 138, 139,
                 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152,
                 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165,
                 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 177, 178, 179,
                 180, 181, 182, 184, 185, 186, 187, 188, 190, 192, 194, 195, 202],
                dtype=int64)
```

Found issue with number of major vessels. This has 4 invalid values. Handling invalid values by filling max values instead of deleting the record from dataset. (This is done to demonstrate the use of filling missing value using fillna and numpy functions)

```
In [23]: # Number of major vessels
         # Valid Values 0,1,2,3
         ds_heart.groupby(['numberofmajorvessels'])['numberofmajorvessels'].count()

Out[23]: numberofmajorvessels
         0    175
         1     65
         2     38
         3     20
         4      4
         Name: numberofmajorvessels, dtype: int64

In [24]: # Found 4 Invalid record for Number of major vessels
         # Either Defaulting with meanvalue , max values or
         # Removing record with Invalid Values
         # To demostrate the concept of filling Missing Value we will use option 1

In [25]: #Code to remove Invalid values (Not used)
         #ds_heart=ds_heart[ds_heart.numberofmajorvessels!=4]

In [26]: ds_heart['numberofmajorvessels'] = ds_heart['numberofmajorvessels'].replace(4,np.nan)

In [27]: np.unique(ds_heart['numberofmajorvessels'])

Out[27]: array([ 0.,  1.,  2.,  3., nan])

In [28]: ds_heart['numberofmajorvessels'] = ds_heart['numberofmajorvessels'].fillna(ds_heart['numberofmajorvessels'].max())

In [29]: ds_heart = ds_heart.astype({'numberofmajorvessels':'int64'})
         ds_heart.groupby(['numberofmajorvessels'])['numberofmajorvessels'].count()

Out[29]: numberofmajorvessels
         0    175
         1     65
         2     38
         3     24
         Name: numberofmajorvessels, dtype: int64

In [30]: ds_heart.shape

Out[30]: (302, 14)
```

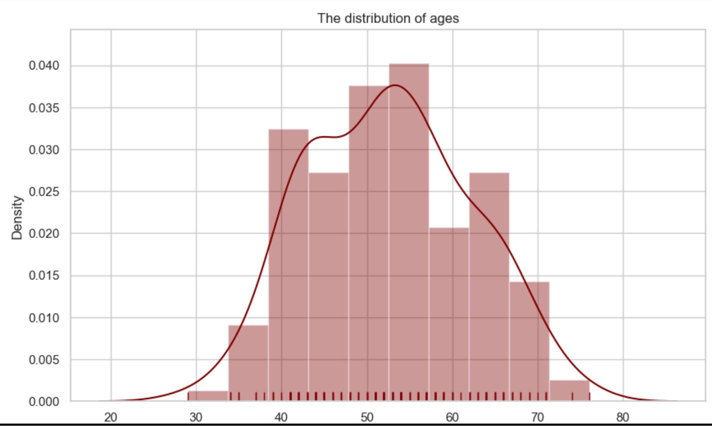
Summary:

- Data shape: 303 rows and 14 columns
- Columns includes 13 independent and 1 target variable
- Found 1 Duplicate record
- Data has no missing values
- Found 4 Invalid record for “Number of major vessels” (Fixed)

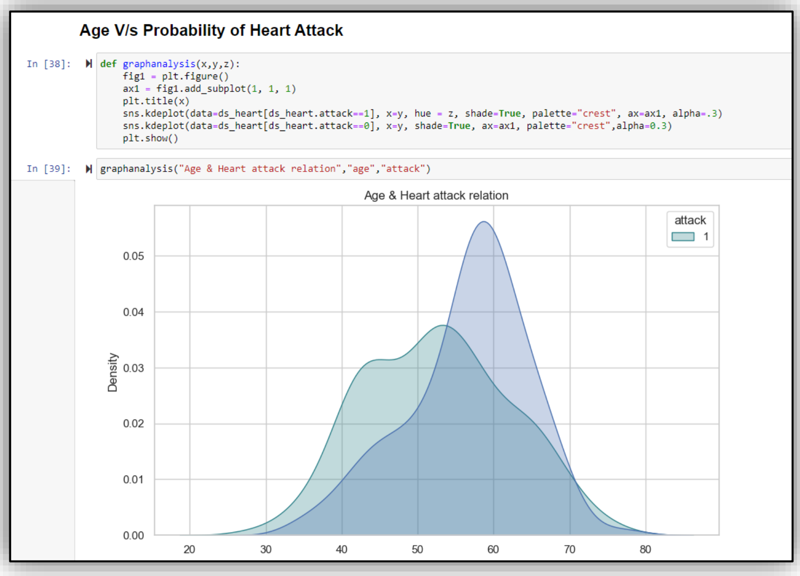
5. Generating insides
   a. People between age 50 to 55 are more likely to have heart attack. Second peak is between 40 to 45

Generating insights

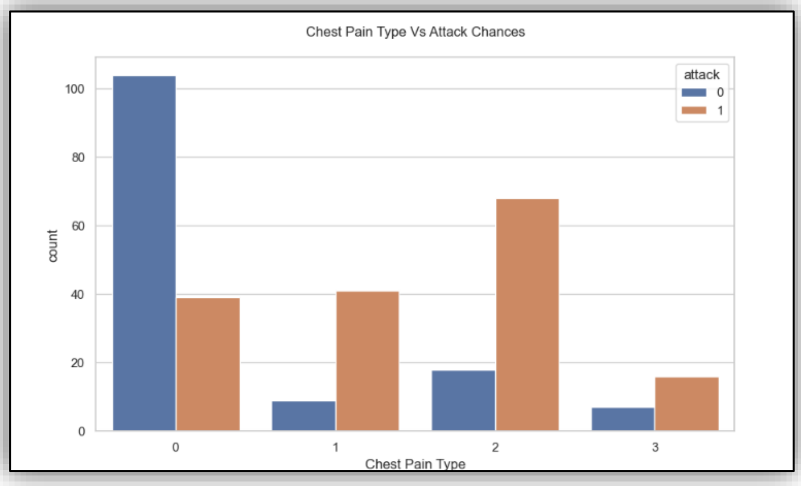
```
ds_attack = ds_heart[ds_heart.attack==1]

ax= sns.distplot(ds_attack.age, rug=True, color="maroon")
plt.title("The distribution of ages")
plt.show()
```



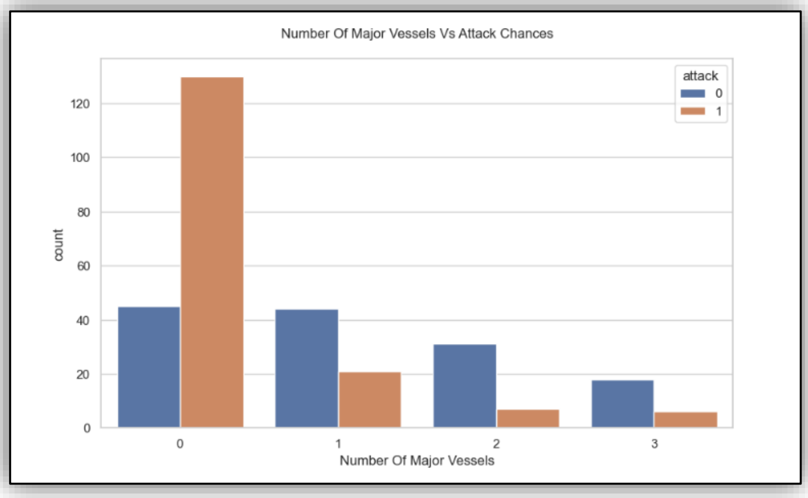
   b.



c. People with Non-Anginal chest pain, that is with cp = 2 have higher chances of heart attack.



d. People with 0 major vessels, that is with caa = 0 have high chance of heart attack.

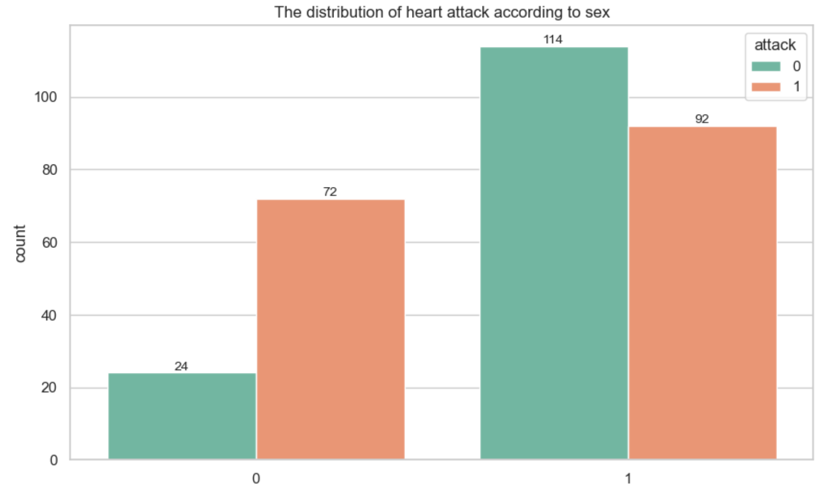


e. People with sex = 0 (Female) have higher chance of heart attack.

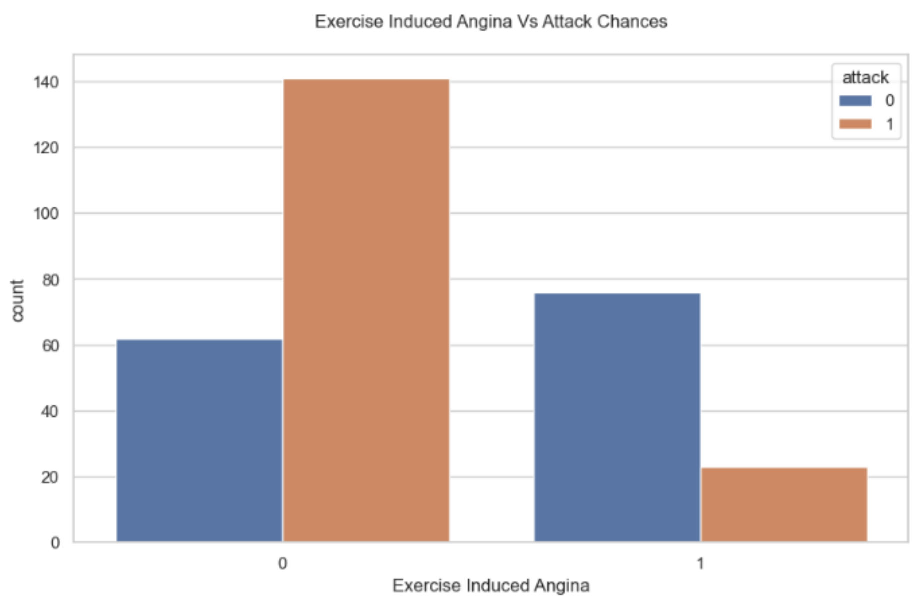
```
In [41]:  # the average heart attack risk percentage according to sex
          # 1 --> male
          # 0 --> female
          ds_heart.groupby('sex').attack.apply(lambda x: x.sum()/x.size * 100)

Out[41]:  sex
          0    75.000000
          1    44.660194
          Name: attack, dtype: float64
```

Gender V/s Probability of Heart Attack

```
In [72]:  ax= sns.countplot(ds_heart.sex, hue=ds_heart.attack, palette="Set2")
          plt.title("The distribution of heart attack according to sex")
          for p in ax.containers:
              ax.bar_label(p)
```



f. People with no exercise induced angina, that is with exng = 0 have higher chance of heart attack.

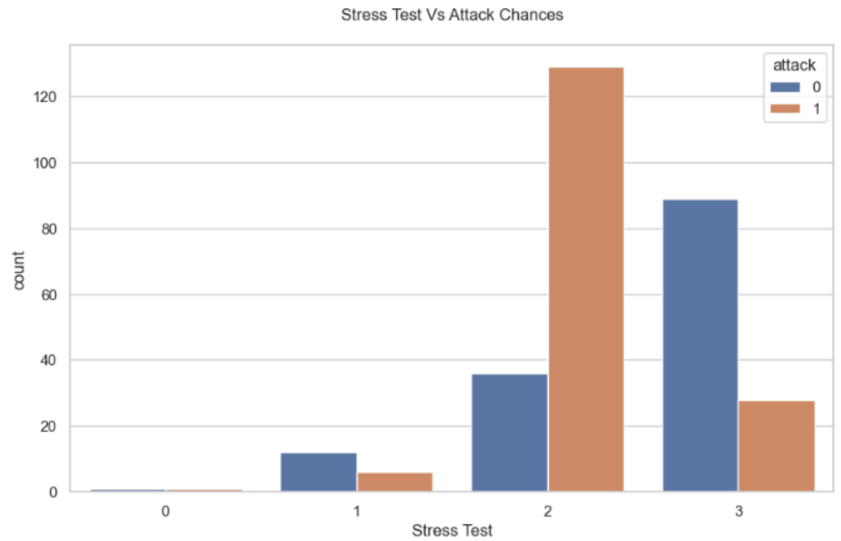


g. People with Stress Test = 2 have much higher chance of heart attack.

Analysis other Varriable Result in Heart Attack

```
7]: def graphshow(x,y,z):
        sns.countplot(data=ds_heart, x=y, hue=z)
        title=('{} Vs Attack Chances\n').format(x)
        plt.title(title)
        plt.xlabel(x)
        plt.show()
```

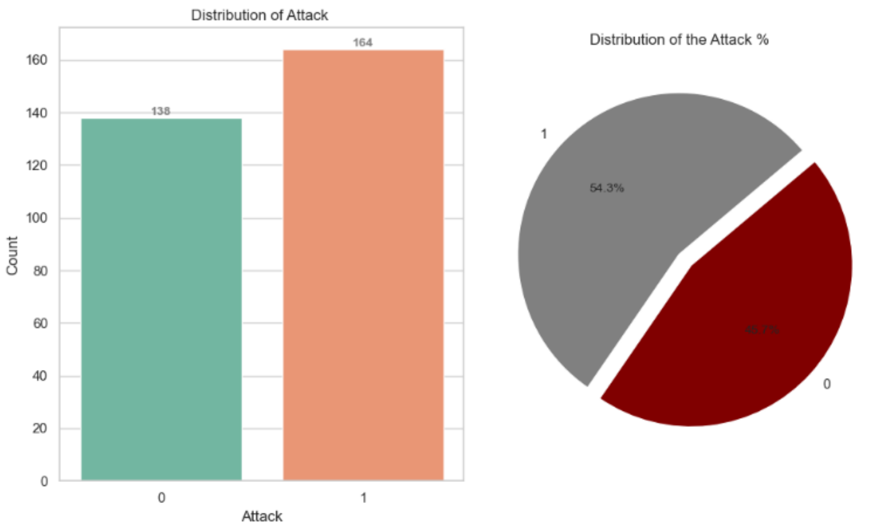
```
8]: graphshow('Stress Test','stresstest','attack')
    graphshow('Number Of Major Vessels','numberofmajorvessels','attack')
    graphshow('Chest Pain Type','chestpaintype','attack')
    graphshow('Resting ECG','restingecg','attack')
    graphshow('Slope','slope','attack')
    graphshow('Exercise Induced Angina','exerciseinducedangina','attack')
    graphshow('Fasting Blood Sugar','fastingbloodsugar','attack')
```



Number Of Major Vessels Vs Attack Chances

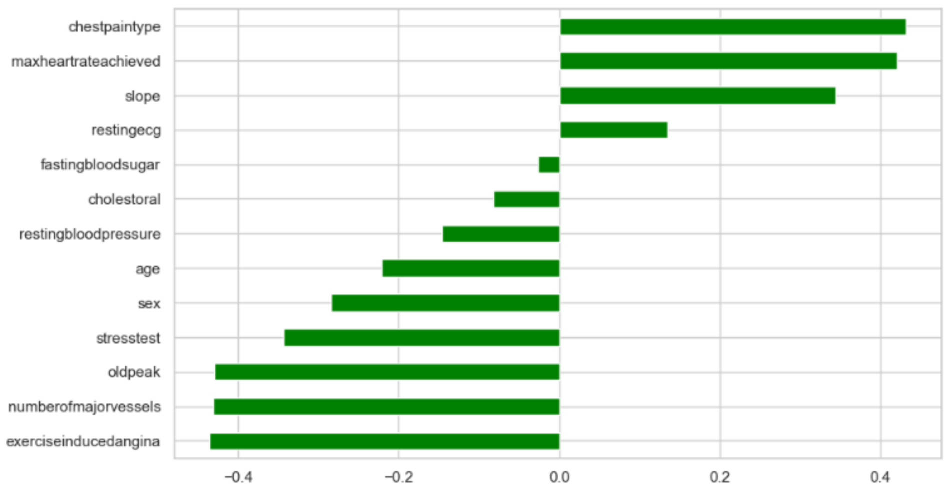
6. Model

In [42]:
```
plt.subplot(1,2,1)
ax = sns.countplot(x='attack', data=ds_heart, palette="Set2")
plt.title('Distribution of Attack ')
plt.xlabel('Attack')
plt.ylabel('Count')
ax.bar_label(ax.containers[0], fontsize=10, color='grey', fontweight='bold')
plt.subplot(1,2,2)
plt.pie(ds_heart.attack.value_counts(), labels = ds_heart.attack.value_counts().index, autopct = '%1.1f%%', startangle = 40,
plt.title('Distribution of the Attack %')
plt.tight_layout()
plt.show()
```

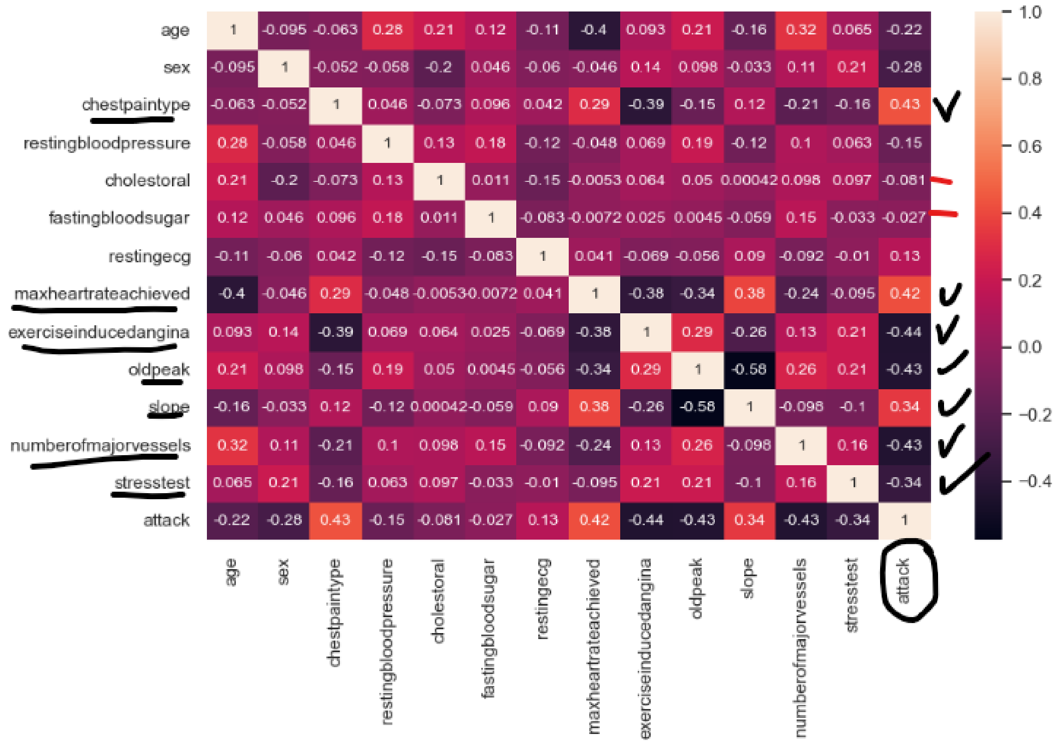


Let's understand the corelation Correlations:

n [49]:
```
ds_heart.corr()['attack'].sort_values().drop('attack').plot(kind = 'barh', color="green");
# Correlation of the target column with other columns
```



```
In [70]:  sns.heatmap(ds_heart.corr(), annot=True)
Out[70]: <AxesSubplot:>
```



There is no apparent linear correlation between continuous variable according to the heatmap.

<u>Features (columns) data type</u>:

- Six features are numerical
- The rest (seven features) are categorical variables
- Target variable is fairly balanced, 54% no-disease to 46% has-disease

<u>Correlations</u>:

- Correlation between features is weak at best
- From the numerical features Number of major vessels, Maximum heart rate achieved and Thal rate (Stress Test) are reasonably fairly correlated with the target variable at -0.43, 0.42 and -0.34 correlation coefficient respectively.
- From the categorical features Exercise induced angina, Chest Pain type and Slope are better correlated with the target variable, Exercise induced angina being the highest at 0.44.
- Surprisingly, Cholesterol has less correlation with heart disease.
- Fasting blood sugar has less correlation with heart disease.

Takeaway: features that have higher predictive power could be, Chest Pain type, Number of major vessels, Maximum heart rate Achieved, Thal rate (Stress Test), Exercise induced angina and Slope.

Now we have balance dataset and feature and categorical feature identified let's proceed with model outcome.

Splitting Date set for training and testing

Modelling

```
In [51]: # Split 75:25
x_train=ds_heart.drop(columns=["attack"])
y_train=ds_heart["attack"]
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.25)

In [52]: print('Train dataset shape:',x_train.shape)
print('Test dataset shape', y_train.shape)

Train dataset shape: (226, 13)
Test dataset shape (226,)

In [53]: numeric_columns = [column for column in x_train.columns if (ds_heart[column].dtype == 'float64' or ds_heart[column].dtype ==
print(numeric_columns)
print('#'*99)
categorical_columns = x_train.select_dtypes(include='object').columns
print(categorical_columns)

['age', 'sex', 'chestpaintype', 'restingbloodpressure', 'cholestoral', 'fastingbloodsugar', 'restingecg', 'maxheartrateachi
eved', 'exerciseinducedangina', 'oldpeak', 'slope', 'numberofmajorvessels', 'stresstest']
###################################################################################################
Index([], dtype='object')
```

<u>Train and Test data split ration is 75:25</u>

```
In [54]: numeric_features = Pipeline([
    ('handlingmissingvalues',SimpleImputer(strategy='median')),
    ('scaling',StandardScaler(with_mean=True))
])
print(numeric_features)

Pipeline(steps=[('handlingmissingvalues', SimpleImputer(strategy='median')),
                ('scaling', StandardScaler())])

In [55]: categorical_features = Pipeline([
    ('handlingmissingvalues',SimpleImputer(strategy='most_frequent')),
    ('encoding', OneHotEncoder()),
    ('scaling', StandardScaler(with_mean=False))
])
print(categorical_features)

Pipeline(steps=[('handlingmissingvalues',
                 SimpleImputer(strategy='most_frequent')),
                ('encoding', OneHotEncoder()),
                ('scaling', StandardScaler(with_mean=False))])

In [56]: processing = ColumnTransformer([
                                ('numeric', numeric_features, numeric_columns),
                                ('categorical', categorical_features, categorical_columns)
                               ])
print(processing)
```

Preparing Basic function to run multiple algorithms together.

1. Random Forest classifier
2. Gradientboot classifier
3. XGBClassifier

Model Preparation & Model Evaluation

In [57]:
```
def prepare_model(algorithm):
    model = Pipeline(steps= [
                            ('processing',processing),
                            ('pca', TruncatedSVD(n_components=3, random_state=12)),
                            ('modeling', algorithm)
                           ])
    model.fit(x_train, y_train)
    return model
```

In [58]:
```
def prepare_confusion_matrix(algo, model):
    print(algo)
    plt.figure(figsize=(6,3))
    pred = model.predict(x_test)
    cm = confusion_matrix(y_test, pred)
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax=ax)
    plt.show()

    # labels, title and ticks
    ax.set_xlabel('Predicted Labels');ax.set_ylabel('True Labels');
    ax.set_title('Confusion Matrix');
```

In [59]:
```
def prepare_classification_report(algo, model):
    print(algo+' Report :')
    pred = model.predict(x_test)
    print(classification_report(y_test, pred))
```

In [60]:
```
def prepare_roc_curve(algo, model):
    print(algo)
    y_pred_proba = model.predict_proba(x_test)[::,1]
    fpr, tpr, thresholds = roc_curve(y_test,  y_pred_proba)
    roc_auc = auc(fpr, tpr)
    curve = RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc)
    curve.plot()
    plt.show()
```

In [61]:
```
algorithms = [('Random Forest calssifier', RandomForestClassifier()),
              ('Gradientboot classifier',GradientBoostingClassifier()),
              ('XGBClassifier', XGBClassifier())
             ]
```

In [62]:
```
trained_models = []
model_and_score = {}

for index, tup in enumerate(algorithms):
    model = prepare_model(tup[1])
    model_and_score[tup[0]] = str(model.score(x_train,y_train)*100)+"%"
    trained_models.append((tup[0],model))
```

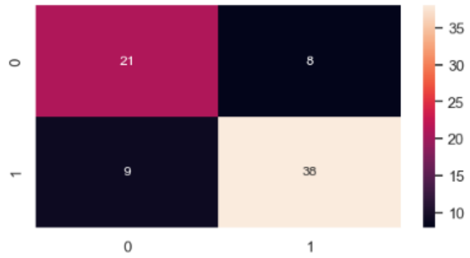
Evaluation Metrics

63]:
```
print(model_and_score)
```

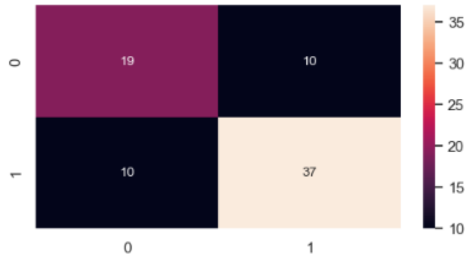
```
{'Random Forest calssifier': '100.0%', 'Gradientboot classifier': '98.67256637168141%', 'XGBClassifier': '100.0%'}
```

```
In [64]:  for index, tup in enumerate(trained_models):
              prepare_confusion_matrix(tup[0], tup[1])
```

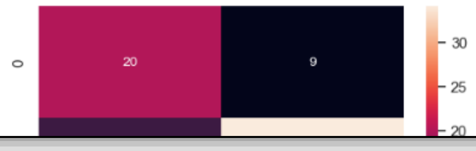
```
Random Forest calssifier
```




```
Gradientboot classifier
```




```
XGBClassifier
```




```
In [65]:  for index, tup in enumerate(trained_models):
              prepare_classification_report(tup[0], tup[1])
              print("\n")
```

```
Random Forest calssifier Report :
              precision    recall  f1-score   support

           0       0.70      0.72      0.71        29
           1       0.83      0.81      0.82        47

    accuracy                           0.78        76
   macro avg       0.76      0.77      0.76        76
weighted avg       0.78      0.78      0.78        76



Gradientboot classifier Report :
              precision    recall  f1-score   support

           0       0.66      0.66      0.66        29
           1       0.79      0.79      0.79        47

    accuracy                           0.74        76
   macro avg       0.72      0.72      0.72        76
weighted avg       0.74      0.74      0.74        76



XGBClassifier Report :
              precision    recall  f1-score   support

           0       0.61      0.69      0.65        29
           1       0.79      0.72      0.76        47

    accuracy                           0.71        76
   macro avg       0.70      0.71      0.70        76
weighted avg       0.72      0.71      0.71        76
```

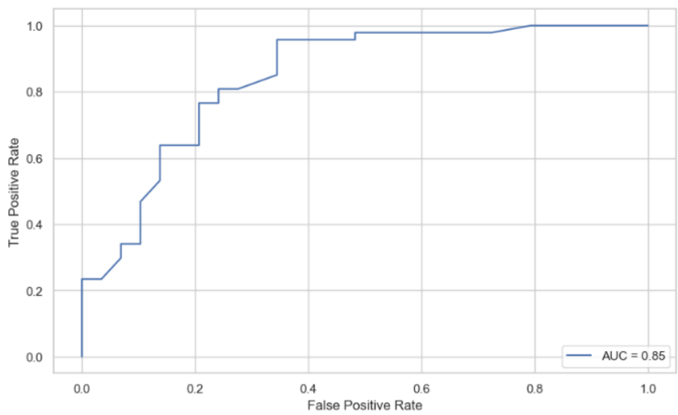
```
In [66]:  print('Test dataset shape:',x_test.shape)
          print('Tes dataset shape', y_test.shape)
```

```
Test dataset shape: (76, 13)
Tes dataset shape (76,)
```

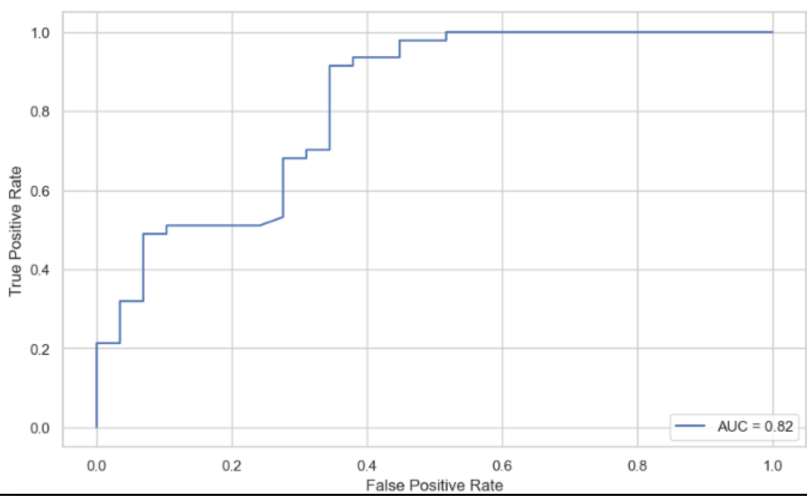
In [67]:

```
encoder = LabelEncoder()
y_test = encoder.fit_transform(y_test)

for index, tup in enumerate(trained_models):
    prepare_roc_curve(tup[0], tup[1])
```

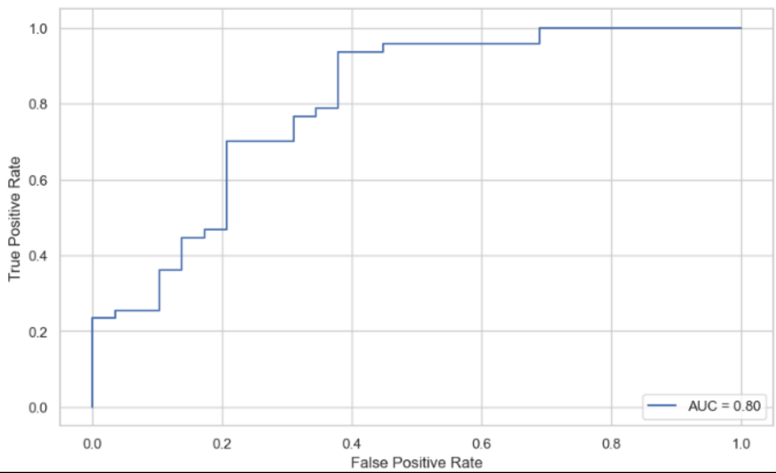
Random Forest calssifier



Gradientboot classifier



XGBClassifier



Results

```
In [75]: x = pd.DataFrame([
             [":","Random Forest calssifier",":","100",":","0.85",":","0.78"],
             [":","Gradientboot classifier",":","98.6",":","0.82",":","0.74"],
             [":","XGB Classifier",":","100",":","0.80",":","0.71"]],
             columns=[":","Model",":","Train Accuracy",":","AUC SCORE",":","f1-Score"]
         )
         print(x)

   :                     Model  : Train Accuracy  : AUC SCORE  : f1-Score
0  :  Random Forest calssifier  :            100  :      0.85  :     0.78
1  :   Gradientboot classifier  :           98.6  :      0.82  :     0.74
2  :            XGB Classifier  :            100  :      0.80  :     0.71
```

- We have achieved 86A Model Can be selected based on the best Training and AUC score
- from the above analysis "Random Forest classifier" and "XGB Classifier" show highest level of accuracy.
- we can take any one of these algorithms for further hyper param tuning and utilizations (Not Covered in this report as it was not the part of curriculum)
- The data set size is also very tiny and need to perform similar analysis in an extended data set.

Insights:

- People between age 50 to 55 are more likely to have heart attack. Second peak is between 40 to 45
- People with Non-Anginal chest pain, that is with cp = 2 have higher chances of heart attack.
- People with 0 major no of vessels, have high chance of heart attack.
- People with sex = 0 (Female) have higher chance of heart attack.
- People with Stress Test = 2 have much higher chance of heart attack.
- People with no exercise induced angina (0) have higher chance of heart attack.
- People with higher maximum heart rate achieved have higher chances of heart attack.
- People with lower previous peak achieved have higher chances of heart attack.
- It is intuitive that elder people might have higher chances of heart attack but according to the distribution plot of age w.r.t output, it is evident that this isn't the case.
- The most important features are thallium stress test, chest pain type, and number of major vessels.
- The less important features are sex, cholesterol, and resting blood pressure.

References

1. An introduction to seaborn — seaborn 0.12.1 documentation (pydata.org)
2. UCD Curriculum
3. Data Camp
4. Stackoverflow
5. GeeksforGeeks
6. Kaggle