# Python Lab Exercise #2

## Objectives:

- Load .csv files into `pandas` DataFrames
- Describe and manipulate data in Series and DataFrames
- Visualize data using DataFrame methods and `matplotlib`

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

## What is Pandas?

Pandas, as the Anaconda docs (https://docs.anaconda.com/anaconda/packages/py3.7_osx-64/) tell us, offers us "High-performance, easy-to-use data structures and data analysis tools." It's something like "Excel for Python", but it's quite a bit more powerful.

Let's read in the heart dataset.

Pandas has many methods for reading different types of files. Note that here we have a .csv file.

Read about this dataset [here (https://www.kaggle.com/ronitf/heart-disease-uci)](https://www.kaggle.com/ronitf/heart-disease-uci).

In [5]:
```
heart_df = pd.read_csv('heart-Copy1.csv')
```

The output of the `.read_csv()` function is a pandas *DataFrame*, which has a familiar tabaular structure of rows and columns.

In [6]:
```
type(heart_df)
```

Out[6]:
```
pandas.core.frame.DataFrame
```

In [7]:
```
heart_df
```

Out[7]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | tar |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | 0 | 1 | |
| **1** | 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | 0 | 0 | 2 | |
| **2** | 41 | 0 | 1 | 130 | 204 | 0 | 0 | 172 | 0 | 1.4 | 2 | 0 | 2 | |
| **3** | 56 | 1 | 1 | 120 | 236 | 0 | 1 | 178 | 0 | 0.8 | 2 | 0 | 2 | |
| **4** | 57 | 0 | 0 | 120 | 354 | 0 | 1 | 163 | 1 | 0.6 | 2 | 0 | 2 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **298** | 57 | 0 | 0 | 140 | 241 | 0 | 1 | 123 | 1 | 0.2 | 1 | 0 | 3 | |
| **299** | 45 | 1 | 3 | 110 | 264 | 0 | 1 | 132 | 0 | 1.2 | 1 | 0 | 3 | |
| **300** | 68 | 1 | 0 | 144 | 193 | 1 | 1 | 141 | 0 | 3.4 | 1 | 2 | 3 | |
| **301** | 57 | 1 | 0 | 130 | 131 | 0 | 1 | 115 | 1 | 1.2 | 1 | 1 | 3 | |
| **302** | 57 | 0 | 1 | 130 | 236 | 0 | 0 | 174 | 0 | 0.0 | 1 | 1 | 2 | |

303 rows × 14 columns

# DataFrames and Series

Two main types of pandas objects are the DataFrame and the Series, the latter being in effect a single column of the former:

```
In [8]: age_series = heart_df['age']
        type(age_series)
```

```
Out[8]: pandas.core.series.Series
```

Notice how we can isolate a column of our DataFrame simply by using square brackets together with the name of the column.

Both Series and DataFrames have an *index* as well:

```
In [8]: heart_df.index
```

```
Out[8]: RangeIndex(start=0, stop=303, step=1)
```

```
In [10]: age_series.index
```

```
Out[10]: RangeIndex(start=0, stop=303, step=1)
```

Pandas is built on top of NumPy, and we can always access the NumPy array underlying a DataFrame using `.values`.

```
In [11]: heart_df.values
```

```
Out[11]: array([[63.,  1.,  3., ...,  0.,  1.,  1.],
                [37.,  1.,  2., ...,  0.,  2.,  1.],
                [41.,  0.,  1., ...,  0.,  2.,  1.],
                ...,
                [68.,  1.,  0., ...,  2.,  3.,  0.],
                [57.,  1.,  0., ...,  1.,  3.,  0.],
                [57.,  0.,  1., ...,  1.,  2.,  0.]])
```

## Basic DataFrame Attributes and Methods

### `.head()`

In [12]:
```
heart_df.head()
```

Out[12]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | targe |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | 0 | 1 | |
| **1** | 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | 0 | 0 | 2 | |
| **2** | 41 | 0 | 1 | 130 | 204 | 0 | 0 | 172 | 0 | 1.4 | 2 | 0 | 2 | |
| **3** | 56 | 1 | 1 | 120 | 236 | 0 | 1 | 178 | 0 | 0.8 | 2 | 0 | 2 | |
| **4** | 57 | 0 | 0 | 120 | 354 | 0 | 1 | 163 | 1 | 0.6 | 2 | 0 | 2 | |

## .tail()

In [13]:
```
heart_df.tail()
```

Out[13]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | tar |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **298** | 57 | 0 | 0 | 140 | 241 | 0 | 1 | 123 | 1 | 0.2 | 1 | 0 | 3 | |
| **299** | 45 | 1 | 3 | 110 | 264 | 0 | 1 | 132 | 0 | 1.2 | 1 | 0 | 3 | |
| **300** | 68 | 1 | 0 | 144 | 193 | 1 | 1 | 141 | 0 | 3.4 | 1 | 2 | 3 | |
| **301** | 57 | 1 | 0 | 130 | 131 | 0 | 1 | 115 | 1 | 1.2 | 1 | 1 | 3 | |
| **302** | 57 | 0 | 1 | 130 | 236 | 0 | 0 | 174 | 0 | 0.0 | 1 | 1 | 2 | |

## .info()

In [14]:
```
heart_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

## .describe()

In [15]:
```
heart_df.describe()
```

Out[15]:

| | age | sex | cp | trestbps | chol | fbs | restecg |
|---|---|---|---|---|---|---|---|
| **count** | 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 |
| **std** | 9.082101 | 0.466011 | 1.032052 | 17.538143 | 51.830751 | 0.356198 | 0.525860 |
| **min** | 29.000000 | 0.000000 | 0.000000 | 94.000000 | 126.000000 | 0.000000 | 0.000000 |
| **25%** | 47.500000 | 0.000000 | 0.000000 | 120.000000 | 211.000000 | 0.000000 | 0.000000 |
| **50%** | 55.000000 | 1.000000 | 1.000000 | 130.000000 | 240.000000 | 0.000000 | 1.000000 |
| **75%** | 61.000000 | 1.000000 | 2.000000 | 140.000000 | 274.500000 | 0.000000 | 1.000000 |
| **max** | 77.000000 | 1.000000 | 3.000000 | 200.000000 | 564.000000 | 1.000000 | 2.000000 |

## .dtypes

In [16]:
```
heart_df.dtypes
```

Out[16]:
```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

## `.shape`

In [17]:
```
heart_df.shape
```

Out[17]:
```
(303, 14)
```

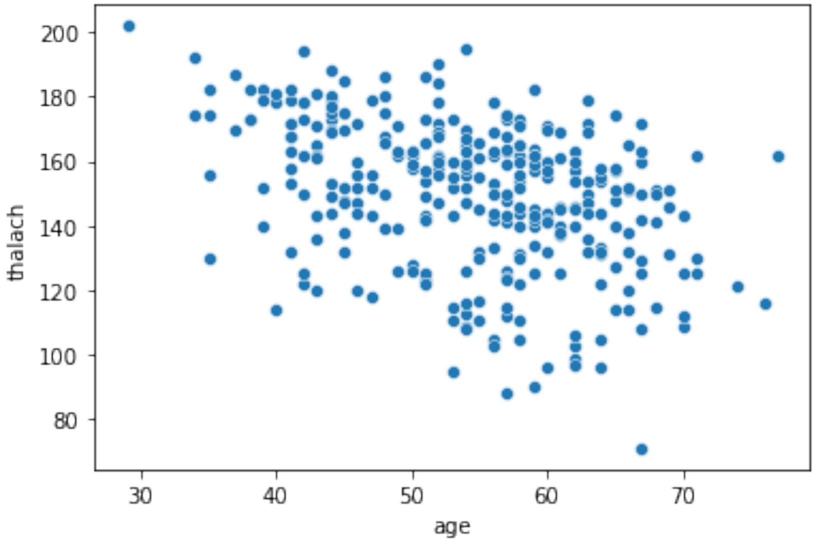
## Exploratory Plots

Let's make ourselves a histogram of ages:

In [ ]:
```
sns.set_style('darkgrid')
sns.distplot(a=heart_df['age']);
# For more recent versions of seaborn:
# sns.histplot(data=heart_df['age'], kde=True);
```

And while we're at it let's do a scatter plot of maximum heart rate vs. age:

```
In [18]: sns.scatterplot(x=heart_df['age'], y=heart_df['thalach']);
```



# Adding to a DataFrame

## Adding Rows

Here are two rows that our engineer accidentally left out of the .csv file, expressed as a Python dictionary:

In [19]:

```
extra_rows = {'age': [40, 30], 'sex': [1, 0], 'cp': [0, 0], 'trestbps'
              'chol': [240, 200],
             'fbs': [0, 0], 'restecg': [1, 0], 'thalach': [120, 122],
              'oldpeak': [0.1, 1.0], 'slope': [1, 1], 'ca': [0, 1], 't
              'target': [0, 0]}
extra_rows
```

Out[19]:

```
{'age': [40, 30],
 'sex': [1, 0],
 'cp': [0, 0],
 'trestbps': [120, 130],
 'chol': [240, 200],
 'fbs': [0, 0],
 'restecg': [1, 0],
 'thalach': [120, 122],
 'exang': [0, 1],
 'oldpeak': [0.1, 1.0],
 'slope': [1, 1],
 'ca': [0, 1],
 'thal': [2, 3],
 'target': [0, 0]}
```

How can we add this to the bottom of our dataset?

In [23]:

```
# Let's first turn this into a DataFrame.
# We can use the .from_dict() method.

missing = pd.DataFrame(extra_rows)
missing
```

Out[23]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | targe |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 40 | 1 | 0 | 120 | 240 | 0 | 1 | 120 | 0 | 0.1 | 1 | 0 | 2 | ( |
| **1** | 30 | 0 | 0 | 130 | 200 | 0 | 0 | 122 | 1 | 1.0 | 1 | 1 | 3 | ( |

In [21]:

```
# Now we just need to concatenate the two DataFrames together.
# Note the `ignore_index` parameter! We'll set that to True.

heart_augmented = pd.concat([heart_df, missing],
                           ignore_index=True)
```

In [22]:
```
# Let's check the end to make sure we were successful!

heart_augmented.tail()
```

Out[22]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | tar |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **300** | 68 | 1 | 0 | 144 | 193 | 1 | 1 | 141 | 0 | 3.4 | 1 | 2 | 3 | |
| **301** | 57 | 1 | 0 | 130 | 131 | 0 | 1 | 115 | 1 | 1.2 | 1 | 1 | 3 | |
| **302** | 57 | 0 | 1 | 130 | 236 | 0 | 0 | 174 | 0 | 0.0 | 1 | 1 | 2 | |
| **303** | 40 | 1 | 0 | 120 | 240 | 0 | 1 | 120 | 0 | 0.1 | 1 | 0 | 2 | |
| **304** | 30 | 0 | 0 | 130 | 200 | 0 | 0 | 122 | 1 | 1.0 | 1 | 1 | 3 | |

## Adding Columns

Adding a column is very easy in `pandas`. Let's add a new column to our dataset called "test", and set all of its values to 0.

In [24]:
```
heart_augmented['test'] = 0
```

In [25]:
```
heart_augmented.head()
```

Out[25]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | targe |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | 0 | 1 | |
| **1** | 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | 0 | 0 | 2 | |
| **2** | 41 | 0 | 1 | 130 | 204 | 0 | 0 | 172 | 0 | 1.4 | 2 | 0 | 2 | |
| **3** | 56 | 1 | 1 | 120 | 236 | 0 | 1 | 178 | 0 | 0.8 | 2 | 0 | 2 | |
| **4** | 57 | 0 | 0 | 120 | 354 | 0 | 1 | 163 | 1 | 0.6 | 2 | 0 | 2 | |

I can also add columns whose values are functions of existing columns.

Suppose I want to add the cholesterol column ("chol") to the resting systolic blood pressure column ("trestbps"):

In [ ]:
```
heart_augmented['chol+trestbps'] = heart_augmented['chol'] + heart_aug
```

In [26]:
```
heart_augmented.head()
```

Out[26]:

|   | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | targe |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | 0 | 1 | |
| **1** | 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | 0 | 0 | 2 | |
| **2** | 41 | 0 | 1 | 130 | 204 | 0 | 0 | 172 | 0 | 1.4 | 2 | 0 | 2 | |
| **3** | 56 | 1 | 1 | 120 | 236 | 0 | 1 | 178 | 0 | 0.8 | 2 | 0 | 2 | |
| **4** | 57 | 0 | 0 | 120 | 354 | 0 | 1 | 163 | 1 | 0.6 | 2 | 0 | 2 | |

## Filtering

We can use filtering techniques to see only certain rows of our data. If we wanted to see only the rows for patients 70 years of age or older, we can simply type:

In [27]:
```
heart_augmented['age'] >= 70
```

Out[27]:
```
0      False
1      False
2      False
3      False
4      False
       ...
300    False
301    False
302    False
303    False
304    False
Name: age, Length: 305, dtype: bool
```

In [28]:
```
heart_augmented[heart_augmented['age'] >= 70]
```

Out[28]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | tar |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **25** | 71 | 0 | 1 | 160 | 302 | 0 | 1 | 162 | 0 | 0.4 | 2 | 2 | 2 | |
| **60** | 71 | 0 | 2 | 110 | 265 | 1 | 0 | 130 | 0 | 0.0 | 2 | 1 | 2 | |
| **129** | 74 | 0 | 1 | 120 | 269 | 0 | 0 | 121 | 1 | 0.2 | 2 | 1 | 2 | |
| **144** | 76 | 0 | 2 | 140 | 197 | 0 | 2 | 116 | 0 | 1.1 | 1 | 0 | 2 | |
| **145** | 70 | 1 | 1 | 156 | 245 | 0 | 0 | 143 | 0 | 0.0 | 2 | 0 | 2 | |
| **151** | 71 | 0 | 0 | 112 | 149 | 0 | 1 | 125 | 0 | 1.6 | 1 | 0 | 2 | |
| **225** | 70 | 1 | 0 | 145 | 174 | 0 | 1 | 125 | 1 | 2.6 | 0 | 0 | 3 | |
| **234** | 70 | 1 | 0 | 130 | 322 | 0 | 0 | 109 | 0 | 2.4 | 1 | 3 | 2 | |
| **238** | 77 | 1 | 0 | 125 | 304 | 0 | 0 | 162 | 1 | 0.0 | 2 | 3 | 2 | |
| **240** | 70 | 1 | 2 | 160 | 269 | 0 | 1 | 112 | 1 | 2.9 | 1 | 1 | 3 | |

Use '&' for "and" and '|' for "or".

## Exercise

Display the patients who are 70 or over as well as the patients whose trestbps score is greater than 170.

In [32]:

```
# Enter your code here
heart_augmented[(heart_augmented['age'] > 70)| (heart_augmented['trest
```

Out[32]:

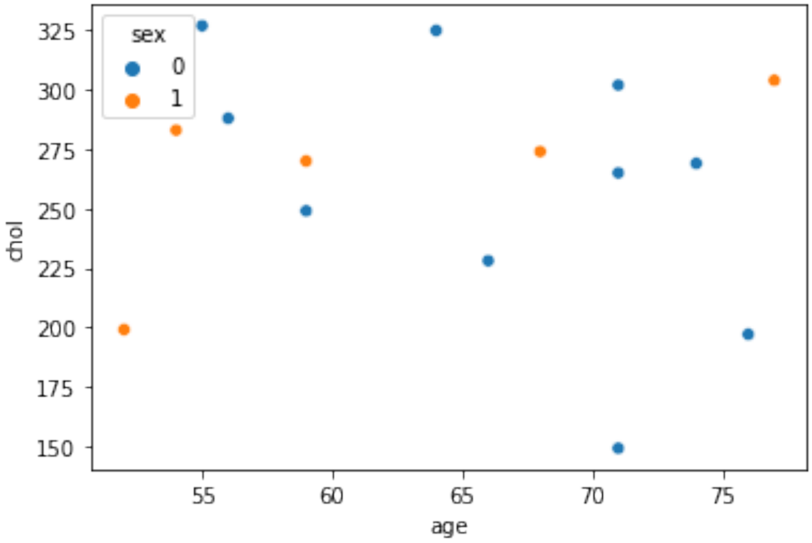
| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | tar |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **8** | 52 | 1 | 2 | 172 | 199 | 1 | 1 | 162 | 0 | 0.5 | 2 | 0 | 3 | |
| **25** | 71 | 0 | 1 | 160 | 302 | 0 | 1 | 162 | 0 | 0.4 | 2 | 2 | 2 | |
| **60** | 71 | 0 | 2 | 110 | 265 | 1 | 0 | 130 | 0 | 0.0 | 2 | 1 | 2 | |
| **101** | 59 | 1 | 3 | 178 | 270 | 0 | 0 | 145 | 0 | 4.2 | 0 | 0 | 3 | |
| **110** | 64 | 0 | 0 | 180 | 325 | 0 | 1 | 154 | 1 | 0.0 | 2 | 0 | 2 | |
| **129** | 74 | 0 | 1 | 120 | 269 | 0 | 0 | 121 | 1 | 0.2 | 2 | 1 | 2 | |
| **144** | 76 | 0 | 2 | 140 | 197 | 0 | 2 | 116 | 0 | 1.1 | 1 | 0 | 2 | |
| **151** | 71 | 0 | 0 | 112 | 149 | 0 | 1 | 125 | 0 | 1.6 | 1 | 0 | 2 | |
| **203** | 68 | 1 | 2 | 180 | 274 | 1 | 0 | 150 | 1 | 1.6 | 1 | 0 | 3 | |
| **223** | 56 | 0 | 0 | 200 | 288 | 1 | 0 | 133 | 1 | 4.0 | 0 | 2 | 3 | |
| **238** | 77 | 1 | 0 | 125 | 304 | 0 | 0 | 162 | 1 | 0.0 | 2 | 3 | 2 | |
| **241** | 59 | 0 | 0 | 174 | 249 | 0 | 1 | 143 | 1 | 0.0 | 1 | 0 | 2 | |
| **248** | 54 | 1 | 1 | 192 | 283 | 0 | 0 | 195 | 0 | 0.0 | 2 | 1 | 3 | |
| **260** | 66 | 0 | 0 | 178 | 228 | 1 | 1 | 165 | 1 | 1.0 | 1 | 2 | 3 | |
| **266** | 55 | 0 | 0 | 180 | 327 | 0 | 2 | 117 | 1 | 3.4 | 1 | 0 | 2 | |

## Exploratory Plot

Using the subframe we just made, let's make a scatter plot of their cholesterol levels vs. age and color by sex:

```
In [34]: at_risk = heart_augmented[(heart_augmented['age'] > 70)| (heart_augmer

         sns.scatterplot(data=at_risk, x='age', y='chol', hue='sex');
```



## .loc and .iloc

We can use `.loc` to get, say, the first ten values of the age and resting blood pressure ("trestbps") columns:

```
In [35]: heart_augmented.loc
```

```
Out[35]: <pandas.core.indexing._LocIndexer at 0x7f9627e0e630>
```

In [36]:
```
heart_augmented.loc[:9, ['age', 'trestbps']]
```

Out[36]:

|   | age | trestbps |
|---|---|---|
| **0** | 63 | 145 |
| **1** | 37 | 130 |
| **2** | 41 | 130 |
| **3** | 56 | 120 |
| **4** | 57 | 120 |
| **5** | 57 | 140 |
| **6** | 56 | 140 |
| **7** | 44 | 120 |
| **8** | 52 | 172 |
| **9** | 57 | 150 |

`.iloc` is used for selecting locations in the DataFrame **by number**:

In [37]:
```
heart_augmented.iloc
```

Out[37]:
```
<pandas.core.indexing._iLocIndexer at 0x7f9626f99d10>
```

In [38]:
```
heart_augmented.iloc[3, 0]
```

Out[38]:
```
56
```

In [39]:
```
heart_augmented.head()
```

Out[39]:

|   | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | targe |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 63 | 1 | 3 | 145 | 233 | 1 | 0 | 150 | 0 | 2.3 | 0 | 0 | 1 |  |
| **1** | 37 | 1 | 2 | 130 | 250 | 0 | 1 | 187 | 0 | 3.5 | 0 | 0 | 2 |  |
| **2** | 41 | 0 | 1 | 130 | 204 | 0 | 0 | 172 | 0 | 1.4 | 2 | 0 | 2 |  |
| **3** | 56 | 1 | 1 | 120 | 236 | 0 | 1 | 178 | 0 | 0.8 | 2 | 0 | 2 |  |
| **4** | 57 | 0 | 0 | 120 | 354 | 0 | 1 | 163 | 1 | 0.6 | 2 | 0 | 2 |  |

### Exercise

How would we get the same slice as just above by using .iloc() instead of .loc()?

In [43]:
```
# Enter your code here

heart_augmented.iloc[0:10, [0,3]]
```

Out[43]:
```
   age  trestbps
0   63       145
1   37       130
2   41       130
3   56       120
4   57       120
5   57       140
6   56       140
7   44       120
8   52       172
9   57       150
```

## Statistics

### `.mean()`

In [44]:

```
heart_augmented.mean()
```

Out[44]:

```
age          54.239344
sex           0.681967
cp            0.960656
trestbps    131.580328
chol        246.091803
fbs           0.147541
restecg       0.527869
thalach     149.459016
exang         0.327869
oldpeak       1.036393
slope         1.396721
ca            0.727869
thal          2.314754
target        0.540984
test          0.000000
dtype: float64
```

Be careful! Some of these will are not straightforwardly interpretable. What does an average "sex" of 0.682 mean?

## `.min()`

In [45]:

```
heart_augmented.min()
```

Out[45]:

```
age          29.0
sex           0.0
cp            0.0
trestbps     94.0
chol        126.0
fbs           0.0
restecg       0.0
thalach      71.0
exang         0.0
oldpeak       0.0
slope         0.0
ca            0.0
thal          0.0
target        0.0
test          0.0
dtype: float64
```

## `.max()`

In [46]:

```
heart_augmented.max()
```

Out[46]:

```
age          77.0
sex           1.0
cp            3.0
trestbps    200.0
chol        564.0
fbs           1.0
restecg       2.0
thalach     202.0
exang         1.0
oldpeak       6.2
slope         2.0
ca            4.0
thal          3.0
target        1.0
test          0.0
dtype: float64
```

# Series Methods

## `.value_counts()`

How many different values does slope have? What about sex? And target?

In [47]:

```
heart_augmented['slope'].value_counts()
```

Out[47]:

```
2    142
1    142
0     21
Name: slope, dtype: int64
```

In [50]:

```
heart_augmented['sex'].value_counts()
```

Out[50]:

```
1    208
0     97
Name: sex, dtype: int64
```

## `.sort_values()`

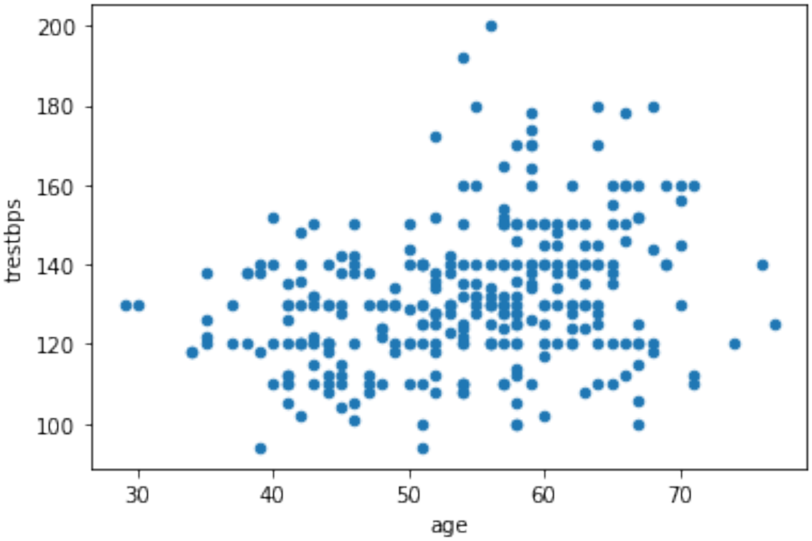
```
In [51]: heart_augmented['age'].sort_values()
```

```
Out[51]: 72     29
         304    30
         58     34
         125    34
         65     35
                ..
         25     71
         60     71
         129    74
         144    76
         238    77
         Name: age, Length: 305, dtype: int64
```

## pandas -Native Plotting

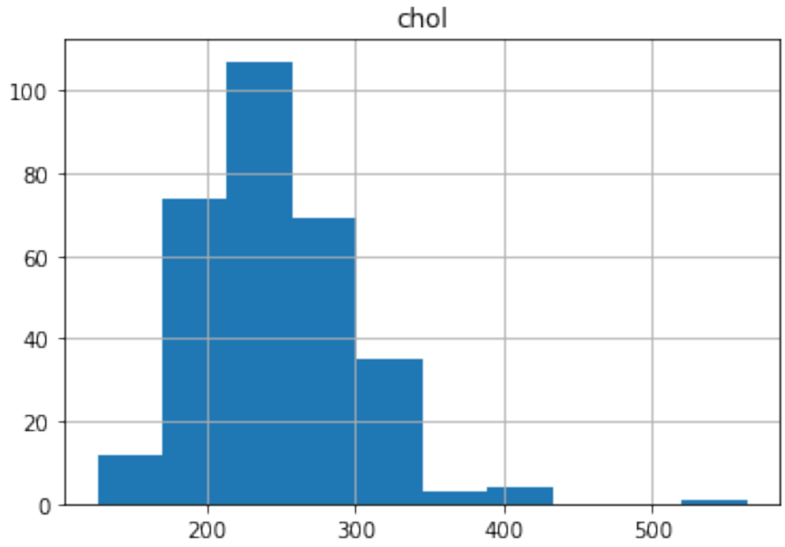
The `.plot()` and `.hist()` methods available for DataFrames use a wrapper around `matplotlib` :

```
In [52]: heart_augmented.plot(x='age', y='trestbps', kind='scatter');
```



In [53]:

```
heart_augmented.hist(column='chol');
```



## Exercises

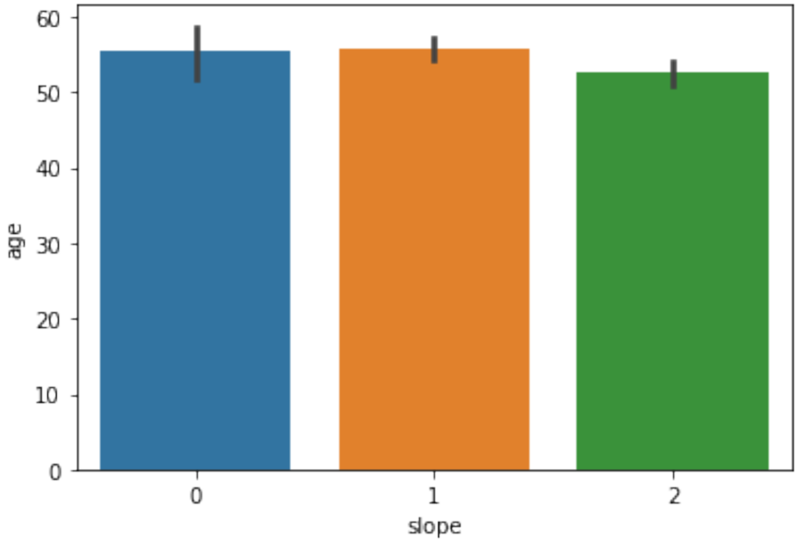
1. Make a bar plot of "age" vs. "slope" for the `heart_augmented` DataFrame.

In [54]:

```
# Enter your code here
sns.barplot(data = heart_augmented, x = 'slope', y ='age')
```

Out[54]:

```
<AxesSubplot:xlabel='slope', ylabel='age'>
```



2. Make a histogram of ages for **just the men** in `heart_augmented` (heart_augmented['sex']=1).

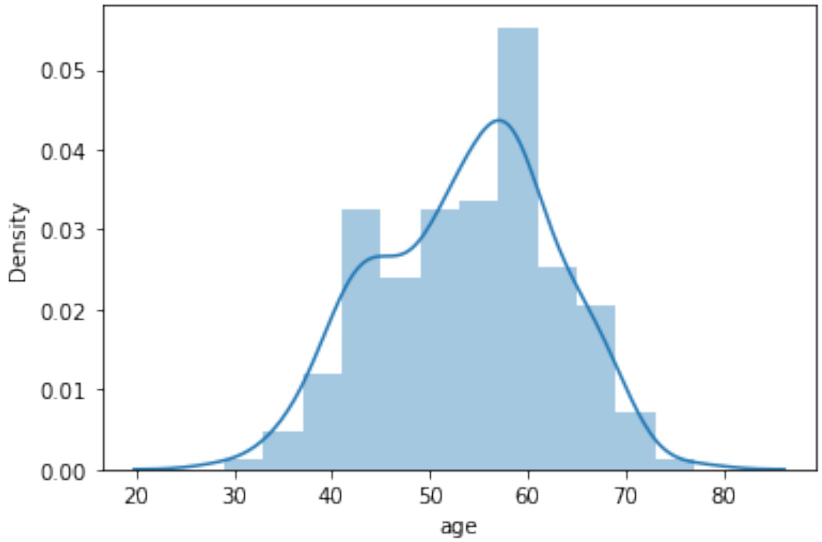
In [58]:

```
# Enter your code here
men = heart_augmented[heart_augmented['sex']== 1]
sns.distplot(a = men['age'])
```

```
/Users/chrisalbert/opt/anaconda3/lib/python3.9/site-packages/seaborn/
distributions.py:2619: FutureWarning: `distplot` is a deprecated func
tion and will be removed in a future version. Please adapt your code
to use either `displot` (a figure-level function with similar flexibi
lity) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[58]:

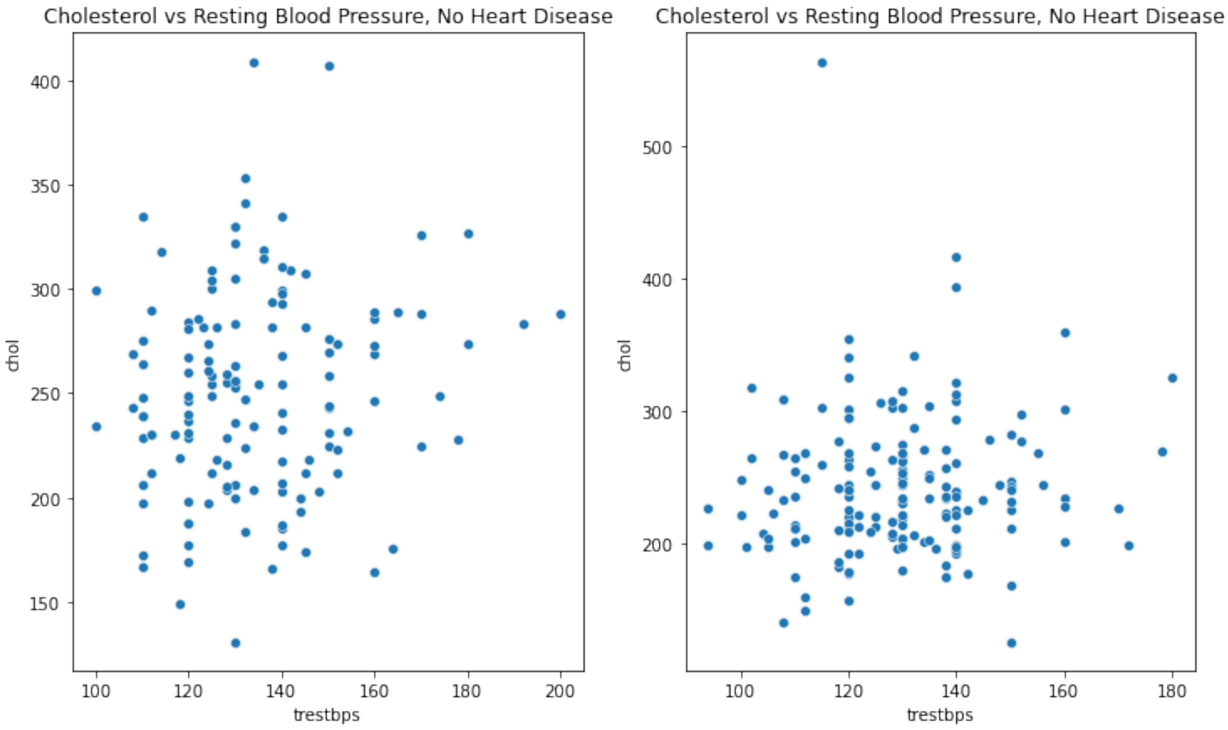
```
<AxesSubplot:xlabel='age', ylabel='Density'>
```



3. Make separate scatter plots of cholesterol vs. resting systolic blood pressure for the target=0 and the target=1 groups. Put both plots on the same figure and give each an appropriate title.

In [67]:
```
# Enter your code here
target0 = heart_augmented[heart_augmented['target']== 0]
target1 = heart_augmented[heart_augmented['target']== 1]
fig, ax = plt.subplots(1,2, figsize =(12,7))
sns.scatterplot(data = target0, x = 'trestbps', y = 'chol', ax=ax[0])
sns.scatterplot(data = target1, x = 'trestbps', y = 'chol', ax=ax[1])
ax[0].set_title('Cholesterol vs Resting Blood Pressure, No Heart Disea
ax[1].set_title('Cholesterol vs Resting Blood Pressure, No Heart Disea
```

Out[67]:
```
Text(0.5, 1.0, 'Cholesterol vs Resting Blood Pressure, No Heart Disea
se')
```



In [ ]: