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`

```
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 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.

```
heart_df = pd.read_csv('heart.csv')
```

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

```
type(heart_df)
```

```
pandas.core.frame.DataFrame

heart_df

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

     slope  ca  thal  target
0        0   0     1       1
1        0   0     2       1
2        2   0     2       1
3        2   0     2       1
4        2   0     2       1
..     ...  ..   ...     ...
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0

[303 rows x 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:

```
age_series = heart_df['age']
type(age_series)
```

```
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:

```
heart_df.index
```

```
RangeIndex(start=0, stop=303, step=1)
```

```
age_series.index
```

```
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`.

```
heart_df.values
```

```
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()`

```
heart_df.head()
```

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

```
   ca  thal  target
0   0     1       1
1   0     2       1
2   0     2       1
3   0     2       1
4   0     2       1
```

.tail()

```
heart_df.tail()
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang
oldpeak  \
298   57    0   0       140   241    0        1      123      1
0.2
299   45    1   3       110   264    0        1      132      0
1.2
300   68    1   0       144   193    1        1      141      0
3.4
301   57    1   0       130   131    0        1      115      1
1.2
302   57    0   1       130   236    0        0      174      0
0.0

     slope  ca  thal  target
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0
```

.info()

```
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()

```
heart_df.describe()
```

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

          restecg     thalach       exang     oldpeak       slope
ca  \
count  303.000000  303.000000  303.000000  303.000000  303.000000
303.000000
mean     0.528053  149.646865    0.326733    1.039604    1.399340
0.729373
std      0.525860   22.905161    0.469794    1.161075    0.616226
1.022606
min      0.000000   71.000000    0.000000    0.000000    0.000000
0.000000
25%      0.000000  133.500000    0.000000    0.000000    1.000000
0.000000
50%      1.000000  153.000000    0.000000    0.800000    1.000000
0.000000
75%      1.000000  166.000000    1.000000    1.600000    2.000000
1.000000
max      2.000000  202.000000    1.000000    6.200000    2.000000
4.000000

             thal      target
count  303.000000  303.000000
```

```
mean     2.313531    0.544554
std      0.612277    0.498835
min      0.000000    0.000000
25%      2.000000    0.000000
50%      2.000000    1.000000
75%      3.000000    1.000000
max      3.000000    1.000000
```

`.dtypes`

```
heart_df.dtypes
```

```
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`

```
heart_df.shape
```

```
(303, 14)
```

Exploratory Plots

Let's make ourselves a histogram of ages:

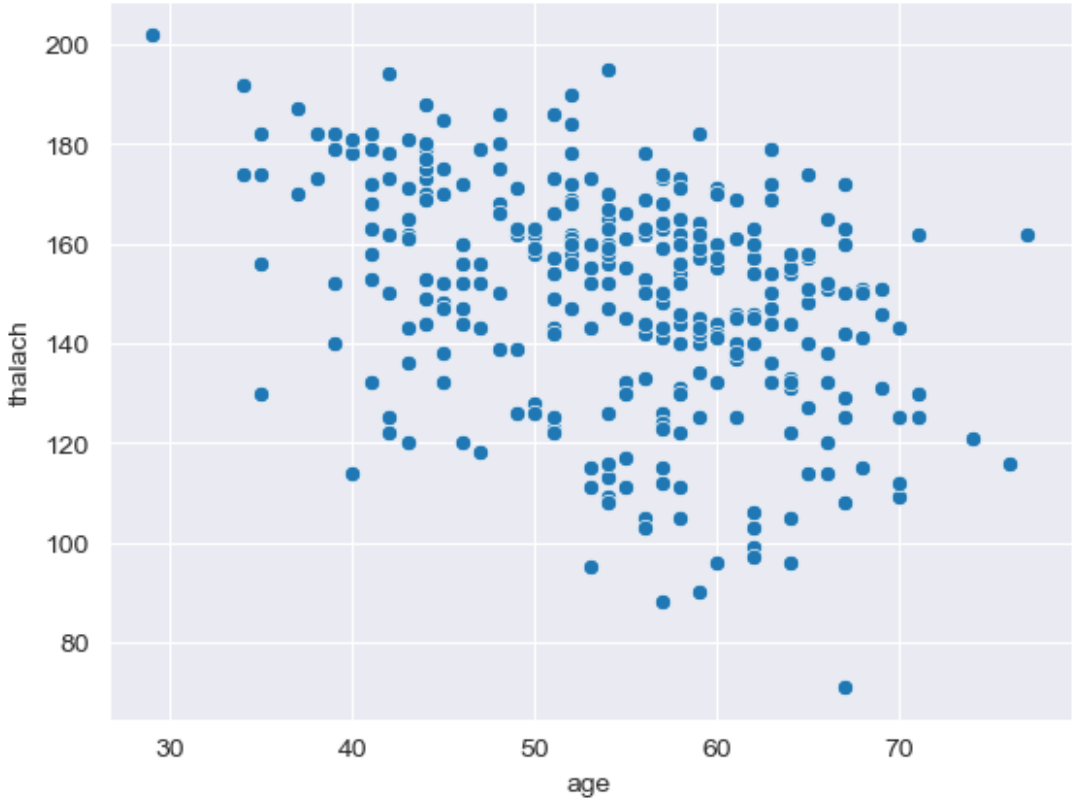
```
sns.set_style('darkgrid')
sns.displot(a=heart_df['age']);
# For more recent versions of seaborn:
# sns.histplot(data=heart_df['age'], kde=True);
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 1
----> 1 sns.set_style('darkgrid')
      2 sns.displot(a=heart_df['age'])
```

```
NameError: name 'sns' is not defined
```

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

```
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:

```
extra_rows = {'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]}
extra_rows
```

```
{'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?

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

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

# 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)

# Let's check the end to make sure we were successful!

heart_augmented.tail()
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang
oldpeak  \
300   68    1   0       144   193    1        1      141      0
3.4
301   57    1   0       130   131    0        1      115      1
1.2
302   57    0   1       130   236    0        0      174      0
0.0
303   40    1   0       120   240    0        1      120      0
0.1
304   30    0   0       130   200    0        0      122      1
1.0

     slope  ca  thal  target
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0
```

```
303      1   0     2       0
304      1   1     3       0
```

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.

```
heart_augmented['test'] = 0

heart_augmented.head()
```

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

   ca  thal  target  test
0   0     1       1     0
1   0     2       1     0
2   0     2       1     0
3   0     2       1     0
4   0     2       1     0
```

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"):

```
heart_augmented['chol+trestbps'] = heart_augmented['chol'] +
heart_augmented['trestbps']

heart_augmented.head()
```

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak
slope  \
0   63    1   3       145   233    1        0      150      0      2.3
0
1   37    1   2       130   250    0        1      187      0      3.5
0
2   41    0   1       130   204    0        0      172      0      1.4
2
```

```
3   56    1   1       120   236    0        1      178      0      0.8
2
4   57    0   0       120   354    0        1      163      1      0.6
2

   ca  thal  target  test  chol+trestbps
0   0     1       1     0            378
1   0     2       1     0            380
2   0     2       1     0            334
3   0     2       1     0            356
4   0     2       1     0            474
```

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:

```
heart_augmented['age'] >= 70

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

heart_augmented[heart_augmented['age'] >= 70]

     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang
oldpeak  \
25    71    0   1       160   302    0        1      162      0
0.4
60    71    0   2       110   265    1        0      130      0
0.0
129   74    0   1       120   269    0        0      121      1
0.2
144   76    0   2       140   197    0        2      116      0
1.1
145   70    1   1       156   245    0        0      143      0
0.0
151   71    0   0       112   149    0        1      125      0
1.6
225   70    1   0       145   174    0        1      125      1
```

```
2.6
234   70    1   0       130   322    0        0       109      0
2.4
238   77    1   0       125   304    0        0       162      1
0.0
240   70    1   2       160   269    0        1       112      1
2.9

     slope  ca  thal  target  test  chol+trestbps
25       2   2     2       1     0            462
60       2   1     2       1     0            375
129      2   1     2       1     0            389
144      1   0     2       1     0            337
145      2   0     2       1     0            401
151      1   0     2       1     0            261
225      0   0     3       0     0            319
234      1   3     2       0     0            452
238      2   3     2       0     0            429
240      1   1     3       0     0            429
```

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.

```
# Enter your code here
heart_augmented[(heart_augmented['age'] >= 70) &
(heart_augmented['trestbps'] > 170)]
```

```
Empty DataFrame
Columns: [age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang,
oldpeak, slope, ca, thal, target, test, chol+trestbps]
Index: []
```

Exploratory Plot

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

```
at_risk = heart_augmented[(heart_augmented['age'] >= 70) &
(heart_augmented['trestbps'] > 170)]
sns.scatterplot(data=at_risk, x='age', y='chol', hue='sex');
#Why are there no data points on the scatter plot?
max_trestbps_70_or_older = heart_augmented[heart_augmented['age'] >=
70]['trestbps'].max()
print(max_trestbps_70_or_older)
```

```
#I see. It's because the max trestbps for individuals aged 70 or older
in the DataFrame is 160.
```

```
---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
Cell In[2], line 1
----> 1 at_risk = heart_augmented[(heart_augmented['age'] >= 70) &
(heart_augmented['trestbps'] > 170)]
      2 sns.scatterplot(data=at_risk, x='age', y='chol', hue='sex');
      3 #Why are there no data points on the scatter plot?

NameError: name 'heart_augmented' is not defined
```

`.loc` and `.iloc`

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

```
heart_augmented.loc
```

```
<pandas.core.indexing._LocIndexer at 0x1bfad122620>
```

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

```
   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**:

```
heart_augmented.iloc
```

```
<pandas.core.indexing._iLocIndexer at 0x1bfac40e440>
```

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

```
56
```

```
heart_augmented.head()
```

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

   ca  thal  target  test  chol+trestbps
0   0     1       1     0            378
1   0     2       1     0            380
2   0     2       1     0            334
3   0     2       1     0            356
4   0     2       1     0            474
```

Exercise

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

```
# Enter your code here
print(heart_augmented.iloc[0:5])
```

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

   ca  thal  target  test  chol+trestbps
0   0     1       1     0            378
1   0     2       1     0            380
2   0     2       1     0            334
3   0     2       1     0            356
4   0     2       1     0            474
```

Statistics

`.mean()`

```
heart_augmented.mean()
# Assuming 0 = female & 1 = male, then an average sex of ~0.682 would
mean that there are slightly more males than females;
# However, the meaning of these binary values would need to be
verified.
```

```
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
chol+trestbps    377.672131
dtype: float64
```

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

`.min()`

```
heart_augmented.min()
```

```
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
```

```
chol+trestbps    249.0
dtype: float64
```

`.max()`

```
heart_augmented.max()
```

```
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
chol+trestbps    679.0
dtype: float64
```

Series Methods

`.value_counts()`

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

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

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

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

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

`.sort_values()`

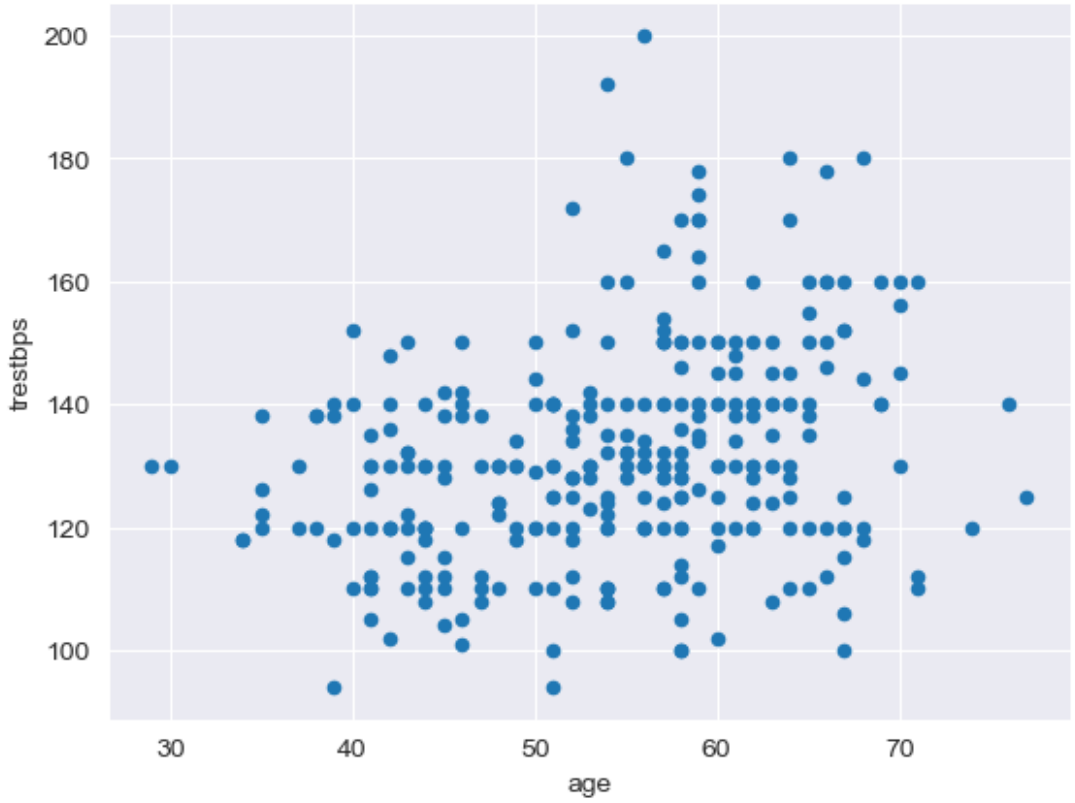
```
heart_augmented['age'].sort_values()
```

```
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
```

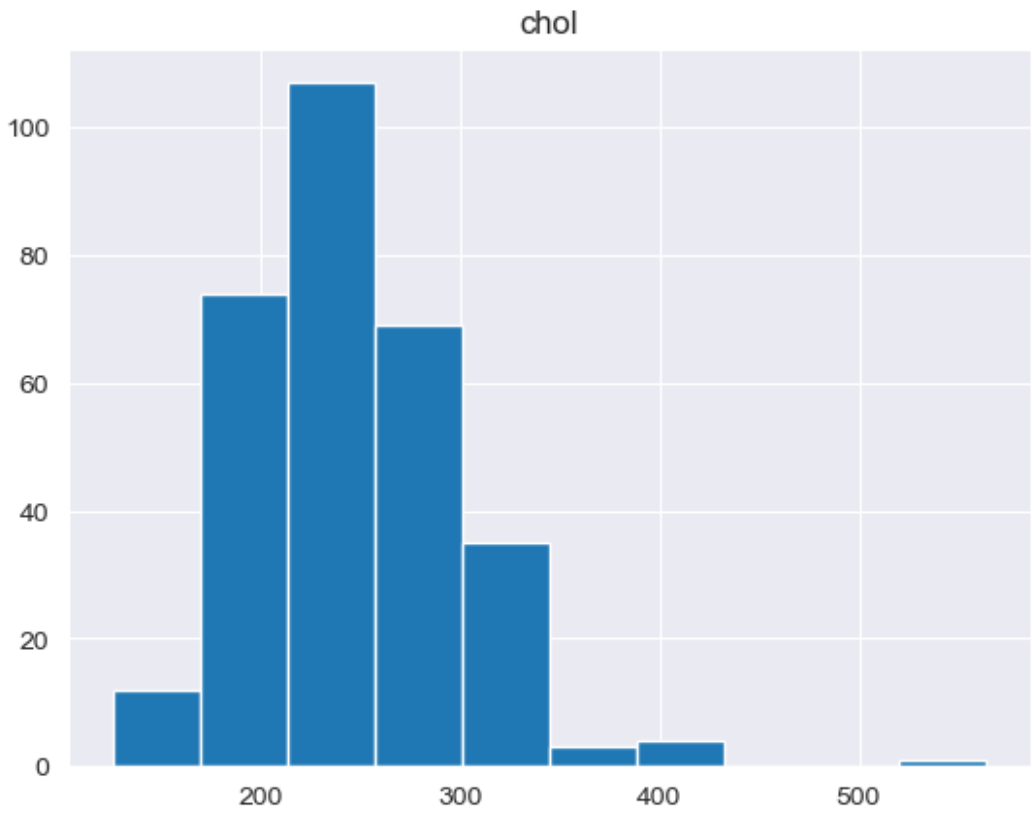
`pandas`-Native Plotting

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

```
heart_augmented.plot(x='age', y='trestbps', kind='scatter');
```




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



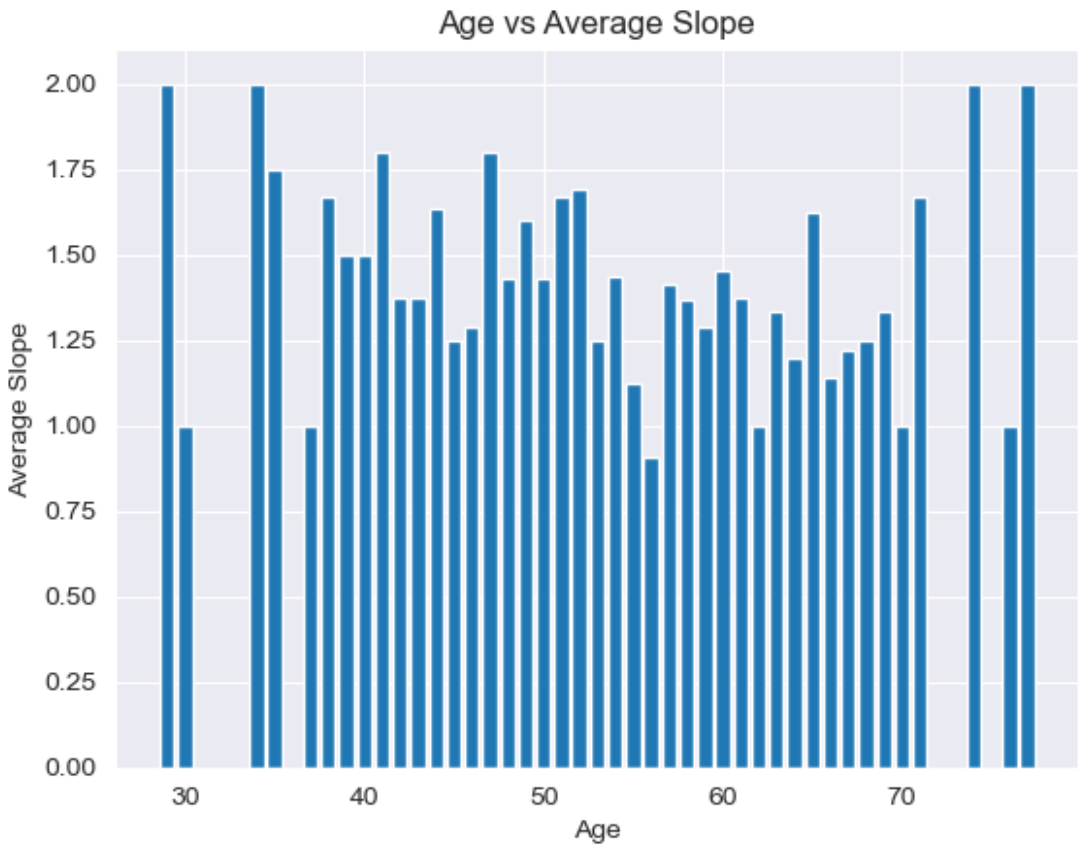
Exercises

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

```
# Enter your code here

# Group by age and calculate the avg slope for each age group
grouped = heart_augmented.groupby('age')['slope'].mean().reset_index()

# Plot a bar chart of age vs avg slope
plt.bar(grouped['age'], grouped['slope'])
plt.xlabel('Age')
plt.ylabel('Average Slope')
plt.title('Age vs Average Slope')
plt.show()
```

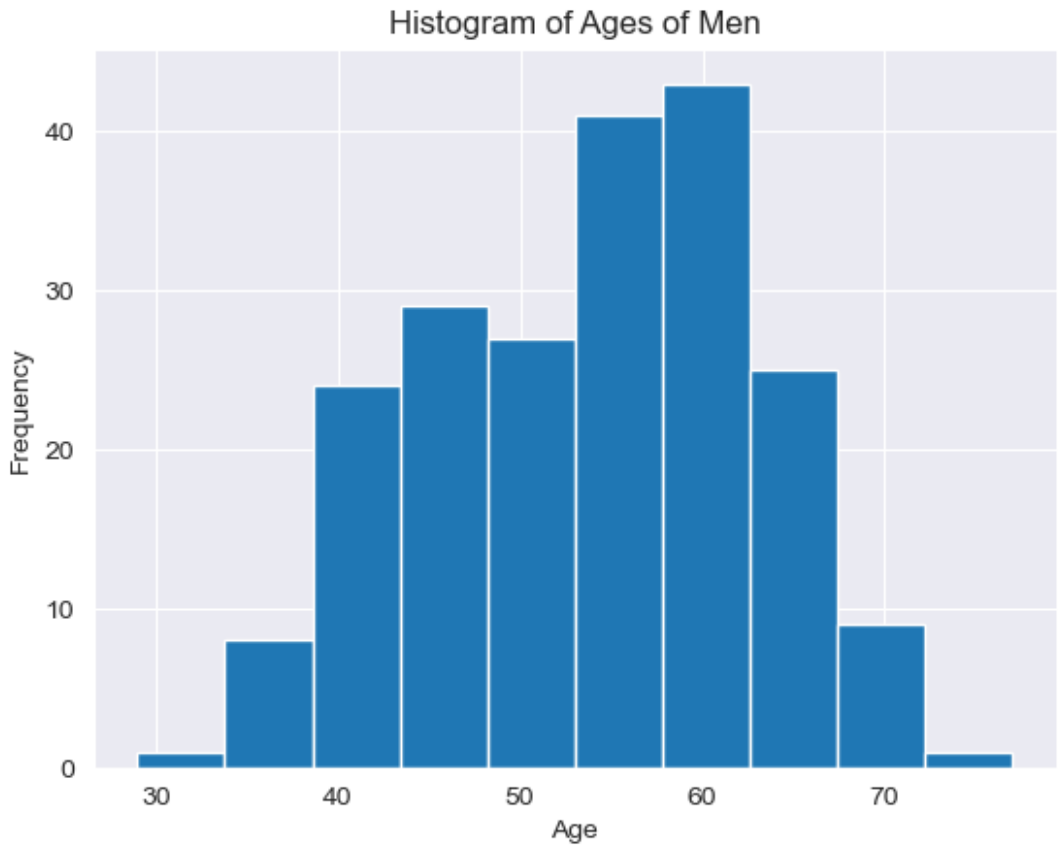


1. Make a histogram of ages for **just the men** in `heart_augmented` (heart_augmented['sex']=1).

```
# Enter your code here

# Filter the DataFrame to include only men
men_df = heart_augmented[heart_augmented['sex'] == 1]

# Plot a histogram of the ages of men
plt.hist(men_df['age'])
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Histogram of Ages of Men')
plt.show()
```



1. 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.

```
# Enter your code here
import matplotlib.pyplot as plt

# Filter the DataFrame based on target values
target_0 = heart_augmented[heart_augmented['target'] == 0]
target_1 = heart_augmented[heart_augmented['target'] == 1]

# Create a figure and a set of subplots
fig, axs = plt.subplots(2, 1, figsize=(10, 8))  # 2 rows, 1 column

# Scatter plot for target = 0
axs[0].scatter(target_0['chol'], target_0['trestbps'], color = 'b')
axs[0].set_xlabel('Cholesterol')
axs[0].set_ylabel('Resting Systolic Blood Pressure')
axs[0].set_title('Cholesterol vs Resting Systolic Blood Pressure
(Target=0)')

# Scatter plot for target = 1
axs[1].scatter(target_1['chol'], target_1['trestbps'], color = 'r')
axs[1].set_title('Cholesterol vs Resting Systolic Blood Pressure
```

```
(Target=1)')
axs[1].set_xlabel('Cholesterol')
axs[1].set_ylabel('Resting Systolic Blood Pressure')

# Adjust layout and show the plots
plt.tight_layout()
plt.show()
```