Python_Lab_Exercise_2

October 2, 2022

1 Python Lab Exercise #2

1.1 Objectives:

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

[3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

1.2 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.

[4]:
```
heart_df = pd.read_csv('C:/Users/maxil/OneDrive - informatik.hs-fulda.de/
↪Studium Hochschule Fulda/Master/3. Semester (USA)/Technologies in Data␣
↪Analytics/DS311-Technologies-in-Data-Analytic/Week_3_Pandas_and_Matplotlib/
↪Lab_Assignment/data/heart.csv')
```

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

[5]:
```
type(heart_df)
```

[5]:
```
pandas.core.frame.DataFrame
```

[6]:
```
heart_df
```

[6]:
```
      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]
```

1.3 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:

[7]:
```
age_series = heart_df['age']
type(age_series)
```

[7]:
```
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:

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

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

[9]:
```
age_series.index
```

[9]:
```
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`.

[10]:
```
heart_df.values
```

[10]:
```
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.]])
```

1.4 Basic DataFrame Attributes and Methods

1.4.1 .head()

[11]:
```
heart_df.head()
```

[11]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
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

1.4.2 .tail()

[12]:
```
heart_df.tail()
```

[12]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
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

1.4.3 .info()

[13]:
```
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
```

1.4.4 .describe()

[14]:
```
heart_df.describe()
```

[14]:

	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
```

1.4.5 .dtypes

[15]:
```
heart_df.dtypes
```

[15]:
```
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
```

1.4.6 .shape

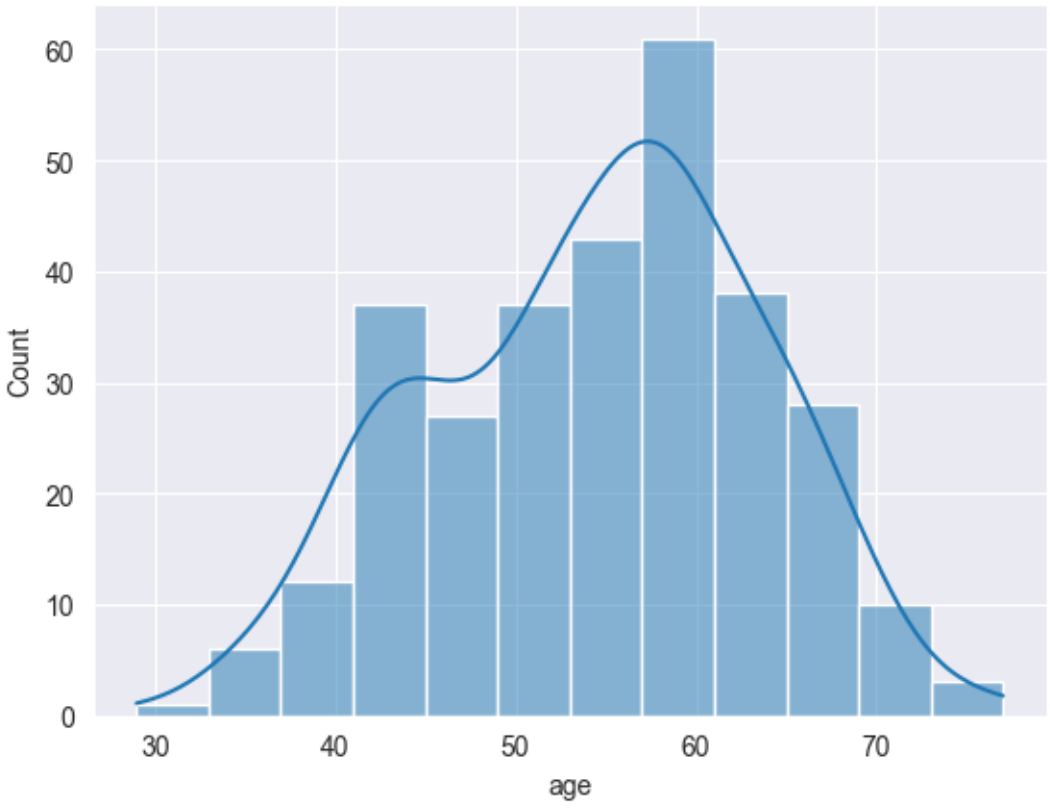
[16]:
```
heart_df.shape
```

[16]:
```
(303, 14)
```

1.4.7 Exploratory Plots

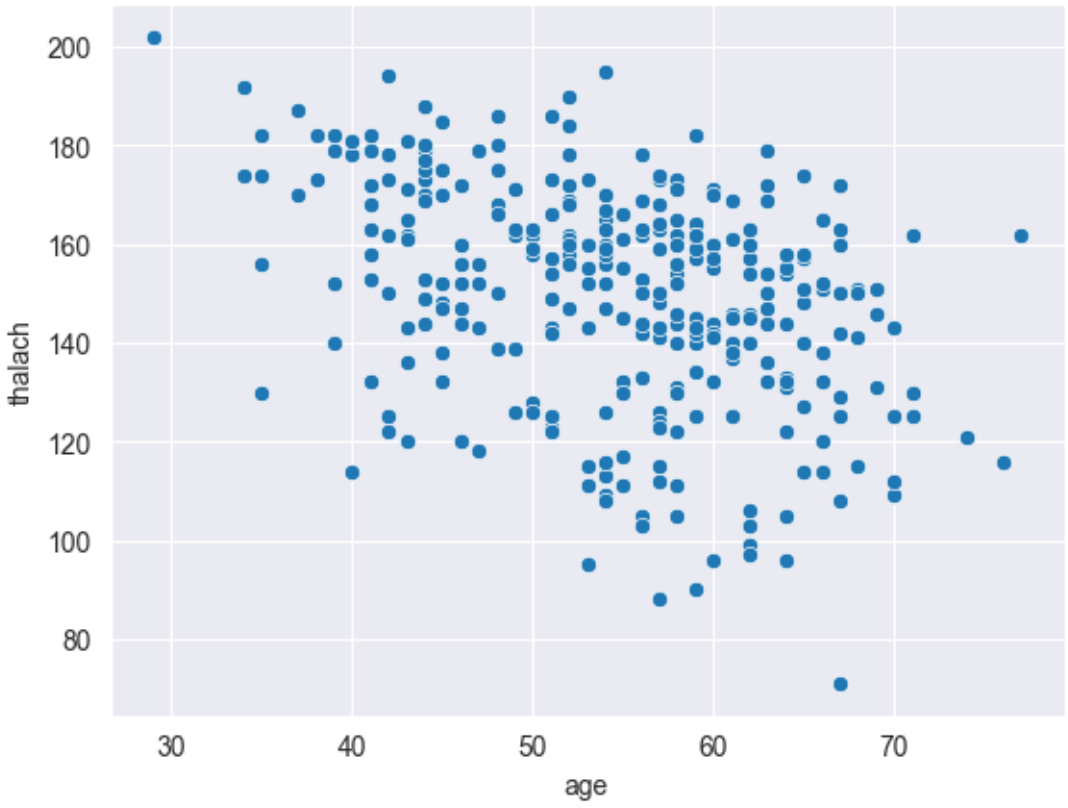
Let's make ourselves a histogram of ages:

[17]:
```
sns.set_style('darkgrid')
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:

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



1.5 Adding to a DataFrame

1.5.1 Adding Rows

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

[19]:
```
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
```

[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?

[20]:
```
# Let's first turn this into a DataFrame.
# We can use the .from_dict() method.

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

[20]:

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

[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)
```

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

heart_augmented.tail()
```

[22]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
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
303	40	1	0	120	240	0	1	120	0	0.1	1	0	2	0
304	30	0	0	130	200	0	0	122	1	1.0	1	1	3	0

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

1.5.2 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.

[23]:
```
heart_augmented['test'] = 0
```

[24]:
```
heart_augmented.head()
```

[24]:

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

[25]:
```
heart_augmented['chol+trestbps'] = heart_augmented['chol'] +␣
↪heart_augmented['trestbps']
```

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

[26]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target	test	chol+trestbps
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1	0	378
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1	0	380
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1	0	334
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1	0	356
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1	0	474

1.6 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:

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

[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
```

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

[28]:

	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".

1.6.1 Exercise

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

[29]:
```
# Enter your code here
heart_augmented[(heart_augmented['age'] >= 70) | (heart_augmented['trestbps'] >␣
↪170)]
```

[29]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	\
8	52	1	2	172	199	1	1	162	0	0.5	
25	71	0	1	160	302	0	1	162	0	0.4	
60	71	0	2	110	265	1	0	130	0	0.0	
101	59	1	3	178	270	0	0	145	0	4.2	
110	64	0	0	180	325	0	1	154	1	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	
203	68	1	2	180	274	1	0	150	1	1.6	
223	56	0	0	200	288	1	0	133	1	4.0	
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	
241	59	0	0	174	249	0	1	143	1	0.0	
248	54	1	1	192	283	0	0	195	0	0.0	
260	66	0	0	178	228	1	1	165	1	1.0	
266	55	0	0	180	327	0	2	117	1	3.4	

	slope	ca	thal	target	test	chol+trestbps
8	2	0	3	1	0	371
25	2	2	2	1	0	462
60	2	1	2	1	0	375
101	0	0	3	1	0	448
110	2	0	2	1	0	505
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
203	1	0	3	0	0	454
223	0	2	3	0	0	488
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
241	1	0	2	0	0	423
248	2	1	3	0	0	475

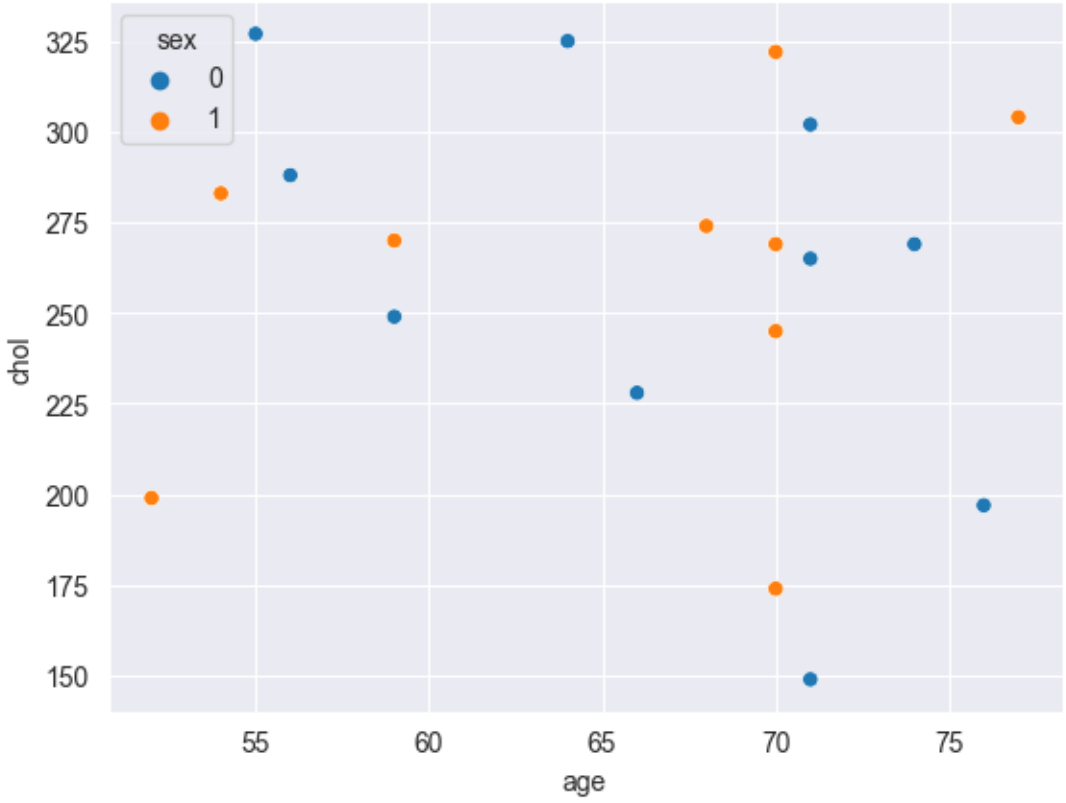
260	1	2	3	0	0	406
266	1	0	2	0	0	507

1.6.2 Exploratory Plot

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

[30]:
```
at_risk = heart_augmented[(heart_augmented['age'] >= 70) |␣
↪(heart_augmented['trestbps'] > 170)]

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



1.6.3 `.loc` and `.iloc`

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

[31]:
```
heart_augmented.loc
```

[31]:
```
<pandas.core.indexing._LocIndexer at 0x2490f2742c0>
```

[32]:
```
heart_augmented.loc[:9, ['age', 'trestbps']]
```

[32]:

	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**:

[33]:
```
heart_augmented.iloc
```

[33]:
```
<pandas.core.indexing._iLocIndexer at 0x24911596c20>
```

[34]:
```
heart_augmented.iloc[3, 0]
```

[34]:
```
56
```

[35]:
```
heart_augmented.head()
```

[35]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target	test	chol+trestbps
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1	0	378
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1	0	380
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1	0	334
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1	0	356
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1	0	474

1.6.4 Exercise

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

[36]:
```
# Enter your code here
heart_augmented.iloc[:10, [0, 3]]
```

[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
```

1.7 Statistics

1.7.1 .mean()

[37]:
```
heart_augmented.mean()
```

[37]:
```
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?

1.7.2 .min()

[38]:
```
heart_augmented.min()
```

[38]:
```
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
```

1.7.3 .max()

[39]:
```
heart_augmented.max()
```

[39]:
```
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
```

1.8 Series Methods

1.8.1 .value_counts()

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

[40]:
```
heart_augmented['slope'].value_counts()
```

[40]:
```
2    142
1    142
0     21
Name: slope, dtype: int64
```

[41]:
```
heart_augmented['sex'].value_counts()
```

[41]:
```
1    208
0     97
Name: sex, dtype: int64
```

1.8.2 .sort_values()

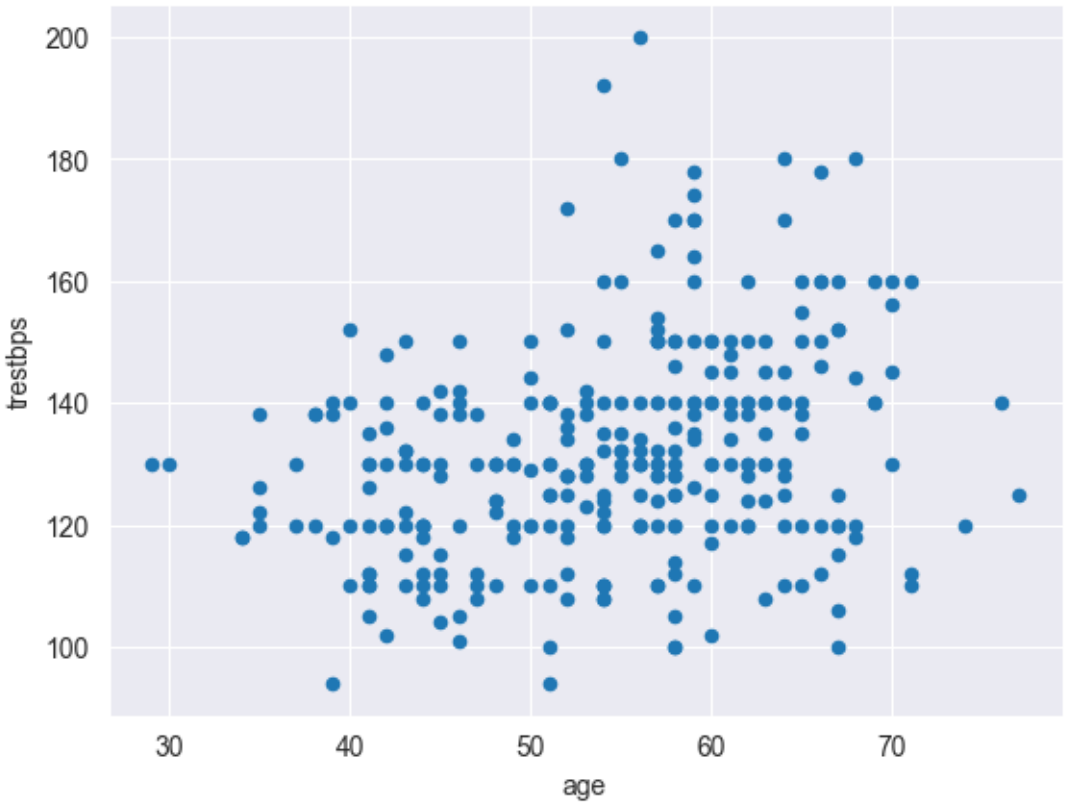
[42]:
```
heart_augmented['age'].sort_values()
```

[42]:
```
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
```

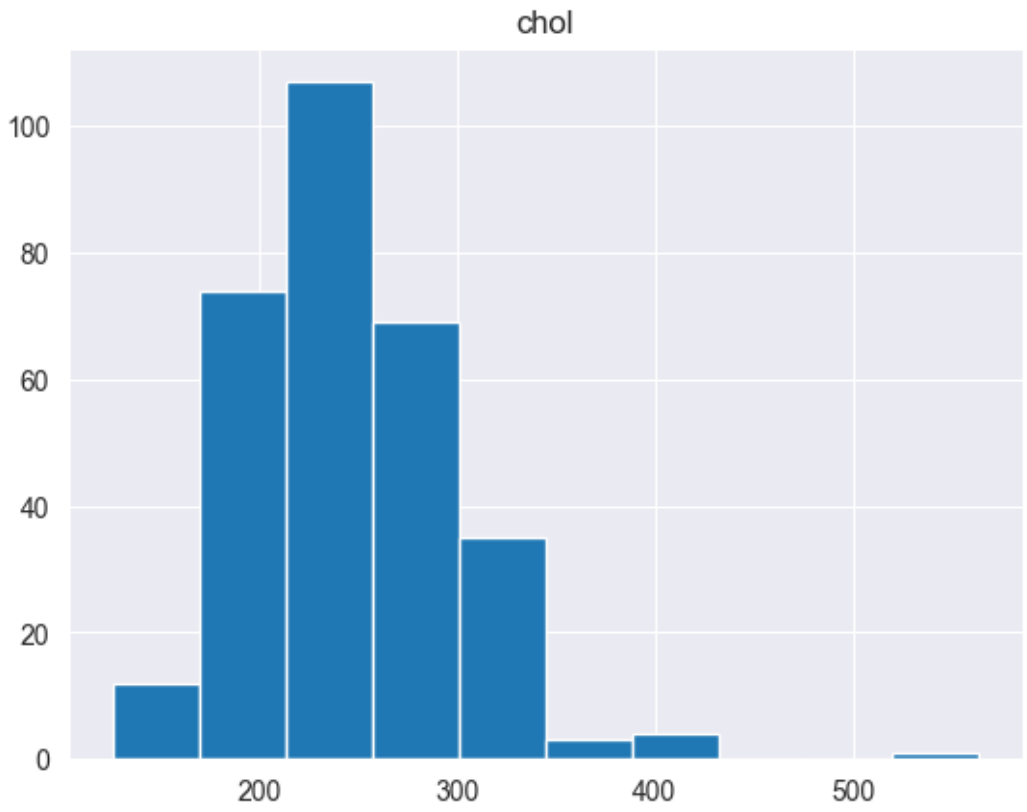
1.9 pandas-Native Plotting

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

[43]:
```
heart_augmented.plot(x='age', y='trestbps', kind='scatter');
```



[44]:
```
heart_augmented.hist(column='chol');
```

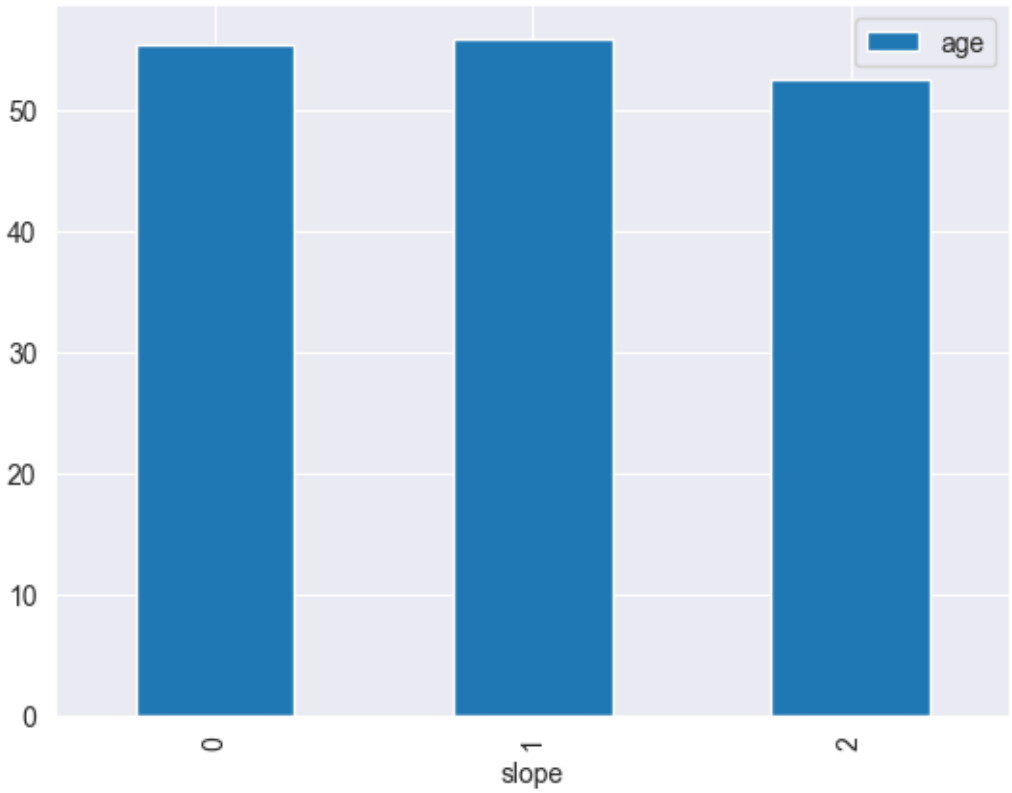


1.9.1 Exercises

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

[45]:
```
# Enter your code here
testtest = heart_augmented.groupby('slope', as_index=False)['age'].mean()
testtest.plot(x='slope', kind='bar')
```

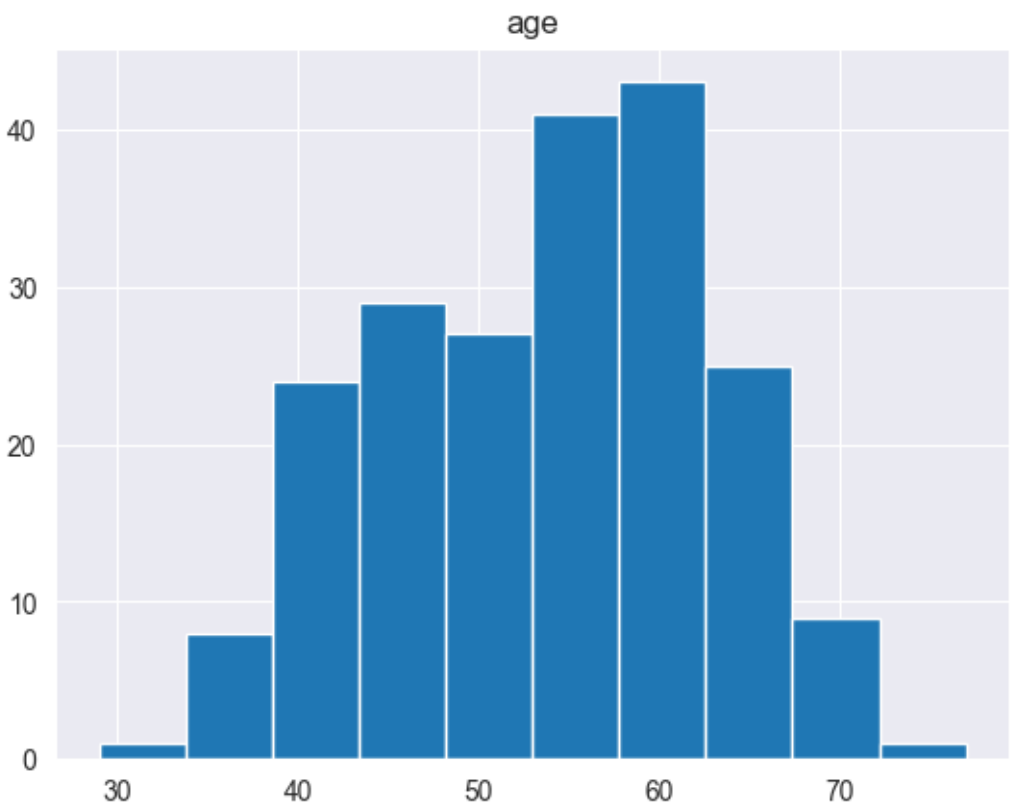
[45]: <AxesSubplot: xlabel='slope'>



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

[46]:
```
# Enter your code here
histoplot = heart_augmented[heart_augmented['sex'] == 1]
histoplot.hist(column='age')
```

[46]:
```
array([[<AxesSubplot: title={'center': 'age'}>]], dtype=object)
```



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.

[47]:
```
# Enter your code here
target0 = heart_augmented[heart_augmented['target'] == 0]
target1 = heart_augmented[heart_augmented['target'] == 1]
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.tight_layout(pad=2.0)
ax1.scatter(target0['chol'], target0['trestbps'])
ax1.set_xlabel('cholesterol')
ax1.set_ylabel('trestbps')
ax1.set_title('chol vs. trestbps with target=0')
ax2.scatter(target1['chol'], target1['trestbps'])
ax2.set_xlabel('cholesterol')
ax2.set_ylabel('trestbps')
ax2.sharex(ax1)
ax2.sharey(ax1)
ax2.set_title('chol vs. trestbps with target=1')
```

[47]:
```
Text(0.5, 1.0, 'chol vs. trestbps with target=1')
```


chol vs. trestbps with target=0
chol vs. trestbps with target=1
trestbps
cholesterol
200
180
160
140
120
100
200
300
400
500