

Statistics I: Technology help — Python

Author: Luc Hens

Version: 16 December 2021

Description: Python code for an introductory applied statistics course. This is work in progress.

Module 1: Data and decisions

To import a text file with data in Python, you first have to find the path to the file. The data file `students.csv` used in the example is stored on my web site: <https://luc-hens.github.io/students.csv> Usually, you will have your data file stored on your computer. In that case you have to give the path to that file, which looks something like:

`/Users/luchens/Documents/Data/students.csv`

This is for macOS; I think Windows uses backslashes: `\` (check).

Note: it is good practice to not have spaces in a csv file. The pandas command `read_csv` has an option `skipinitialspace=True` ("Skip spaces after delimiter") but that does not solve all the problems caused by blank spaces in a csv file.

```
In [43]: import pandas as pd
df = pd.read_csv('https://luc-hens.github.io/students.csv', sep=',', skipinitialspace=True)
print(df)
display(df)
```

	case	sex	height	weight	major
0	1	Female	172	63	Business
1	2	Female	170	70	International Affairs
2	3	Female	170	52	Other
3	4	Female	171	52	Communications
4	5	Male	186	90	Business
5	6	Male	183	79	Business
6	7	Male	170	66	Communications
7	8	Female	169	56	Business
8	9	Male	175	75	International Affairs
9	10	Female	175	65	Communications
10	11	Male	195	94	Business
11	12	Female	176	51	International Affairs
12	13	Male	188	76	International Affairs
13	14	Male	192	82	Business
14	15	Male	172	70	International Affairs
15	16	Female	169	53	Business
16	17	Female	172	52	International Affairs
17	18	Male	178	85	Business
18	19	Female	177	59	Communications
19	20	Male	178	72	International Affairs
20	21	Female	160	54	Business
21	22	Female	175	54	International Affairs
22	23	Male	190	70	International Affairs
23	24	Male	178	85	Business
24	25	Female	163	55	Business
25	26	Female	161	59	Business
26	27	Female	162	44	Communications

27	28	Female	170	54	Business
28	29	Female	154	52	Business
29	30	Female	170	65	Business

	case	sex	height	weight	major
0	1	Female	172	63	Business
1	2	Female	170	70	International Affairs
2	3	Female	170	52	Other
3	4	Female	171	52	Communications
4	5	Male	186	90	Business
5	6	Male	183	79	Business
6	7	Male	170	66	Communications
7	8	Female	169	56	Business
8	9	Male	175	75	International Affairs
9	10	Female	175	65	Communications
10	11	Male	195	94	Business
11	12	Female	176	51	International Affairs
12	13	Male	188	76	International Affairs
13	14	Male	192	82	Business
14	15	Male	172	70	International Affairs
15	16	Female	169	53	Business
16	17	Female	172	52	International Affairs
17	18	Male	178	85	Business
18	19	Female	177	59	Communications
19	20	Male	178	72	International Affairs
20	21	Female	160	54	Business
21	22	Female	175	54	International Affairs
22	23	Male	190	70	International Affairs
23	24	Male	178	85	Business
24	25	Female	163	55	Business
25	26	Female	161	59	Business
26	27	Female	162	44	Communications
27	28	Female	170	54	Business
28	29	Female	154	52	Business
29	30	Female	170	65	Business

To display just the first couple of lines of the data frame called df:

```
In [44]: df.head()
```

```
Out[44]:
```

	case	sex	height	weight	major
0	1	Female	172	63	Business
1	2	Female	170	70	International Affairs
2	3	Female	170	52	Other
3	4	Female	171	52	Communications
4	5	Male	186	90	Business

To display the last couple of lines of the data frame called df:

```
In [45]: df.tail()
```

```
Out[45]:
```

	case	sex	height	weight	major
25	26	Female	161	59	Business
26	27	Female	162	44	Communications
27	28	Female	170	54	Business
28	29	Female	154	52	Business
29	30	Female	170	65	Business

To display the 10 first lines of the data frame:

```
In [46]: df.head(10)
```

```
Out[46]:
```

	case	sex	height	weight	major
0	1	Female	172	63	Business
1	2	Female	170	70	International Affairs
2	3	Female	170	52	Other
3	4	Female	171	52	Communications
4	5	Male	186	90	Business
5	6	Male	183	79	Business
6	7	Male	170	66	Communications
7	8	Female	169	56	Business
8	9	Male	175	75	International Affairs
9	10	Female	175	65	Communications

Show the column names (variable names) (this is useful to check whether there are no blank spaces in the variable names):

```
In [47]: list(df)
```

```
Out[47]: ['case', 'sex', 'height', 'weight', 'major']
```

Inspect the data types in the dataframe called df:

```
In [48]: df.dtypes
```

```
Out[48]: case      int64  
sex      object  
height   int64  
weight   int64  
major    object  
dtype: object
```

To display just one of the variables:

```
In [49]: print(df.height)
```

```
0      172  
1      170  
2      170  
3      171  
4      186  
5      183  
6      170  
7      169  
8      175  
9      175  
10     195  
11     176  
12     188  
13     192  
14     172  
15     169  
16     172  
17     178  
18     177  
19     178  
20     160  
21     175  
22     190  
23     178  
24     163  
25     161  
26     162  
27     170  
28     154  
29     170
```

```
Name: height, dtype: int64
```

```
In [ ]:
```

Module 2: Displaying and describing categorical data

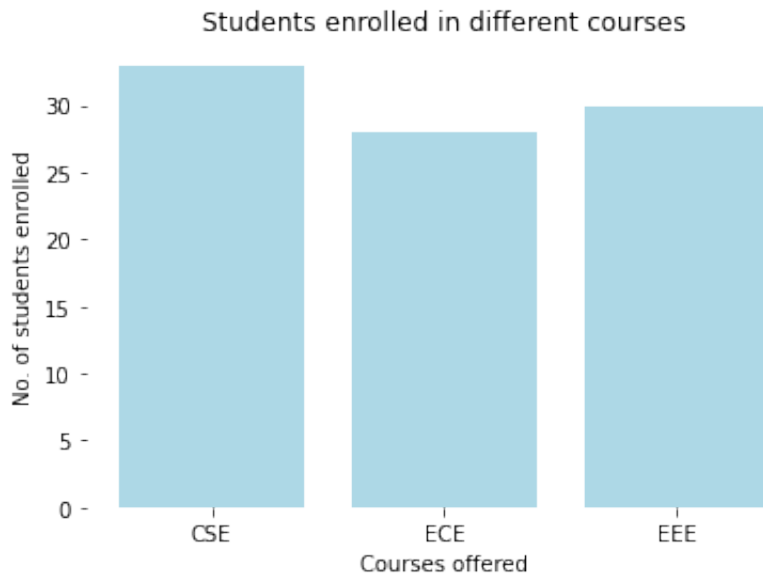
Bar chart and pie chart

Generate a **bar chart** showing enrolment in three classes with course codes CSE (33 students), ECE (28 students), EEE (30 students):

(Documentation: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html)

(Example is taken from <https://www.analyticsvidhya.com/blog/2021/08/understanding-bar-plots-in-python-beginners-guide-to-data-visualization/>)

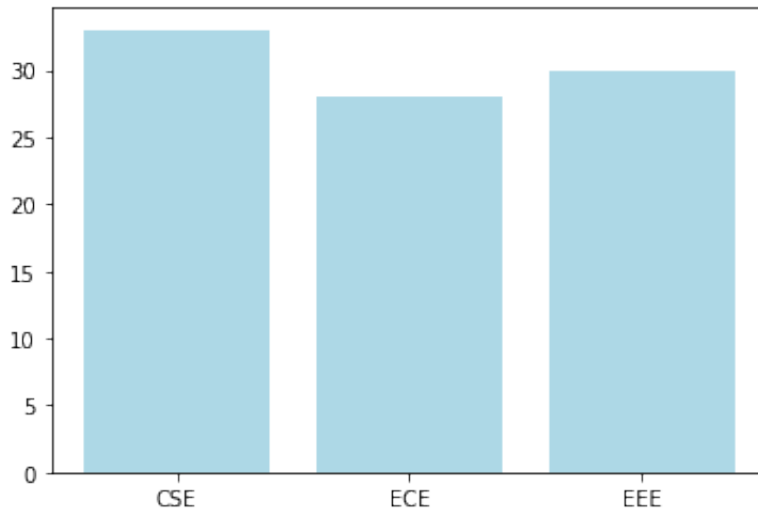
```
In [50]: import numpy as np
import matplotlib.pyplot as plt
# Dataset generation
data_dict = {'CSE':33, 'ECE':28, 'EEE':30}
courses = list(data_dict.keys())
values = list(data_dict.values())
fig = plt.figure() # add option `figsize = (10, 5)` to control size
# Bar plot
plt.box(False) # get rid of the box
plt.bar(courses, values, color='lightblue')
plt.xlabel("Courses offered")
plt.ylabel("No. of students enrolled")
plt.title("Students enrolled in different courses")
plt.show()
```



Save a plot to a file (.png or .pdf):

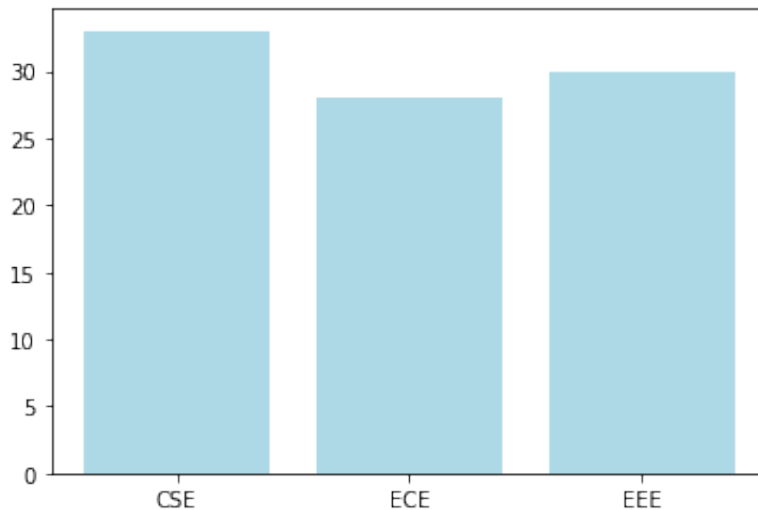
```
In [51]: fig = plt.figure()
plt.bar(courses, values, color='lightblue')
fig.savefig('saved_figure-1000dpi.png', dpi = 1000, transparent=True)
# the plot is saved to the current working directory (cwd)
# to find out what the current working directory (cwd) is:
import os
print('The file is saved to the current working directory: ', os.getcwd())
```

The file is saved to the current working directory: /Users/luchens/Documents/jupyter-notebooks



To save the plot as a .pdf:

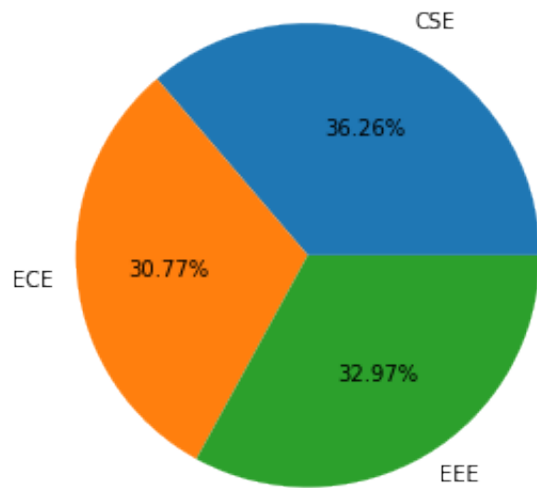
```
In [52]: fig = plt.figure()
plt.bar(courses, values, color='lightblue')
fig.savefig('saved_figure-1000dpi.pdf', dpi = 1000, transparent=True)
```



Generate a **pie chart** for the same data (pie charts are usually a poor way to display data):

(documentation: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.pie.html)

```
In [53]: import numpy as np
import matplotlib.pyplot as plt
# Dataset generation
data_dict = {'CSE':33, 'ECE':28, 'EEE':30}
courses = list(data_dict.keys())
values = list(data_dict.values())
fig = plt.figure(figsize = (10, 5))
# Pie chart:
plt.pie(values, labels=courses, autopct='%1.2f%%') # option: autopct to show
plt.show()
```



Generate a bar chart and a pie chart from an imported data set (students.csv):

Use `value_counts()` to count the how many times each of the values of the variable 'sex' in the dataframe 'df' occurs:

```
In [54]: a = df.sex.value_counts()
         print(a)
```

```
Female    18
Male      12
Name: sex, dtype: int64
```

```
In [55]: a.values
```

```
Out[55]: array([18, 12])
```

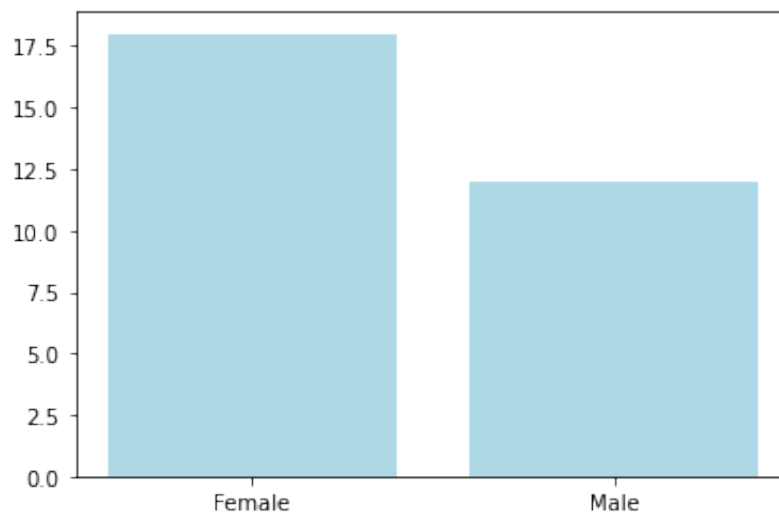
```
In [56]: a.index
```

```
Out[56]: Index(['Female', 'Male'], dtype='object')
```

Then use `a` as input for the `bar()` function:

```
In [57]: plt.bar(a.index, a.values, color='lightblue')
```

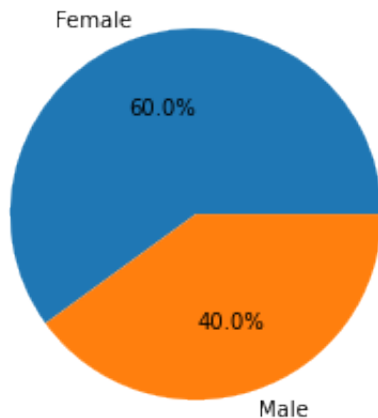
```
Out[57]: <BarContainer object of 2 artists>
```



Or use `a` as input for the `pie()` function:

```
In [58]: plt.pie(a.values, labels=a.index, autopct='%1.1f%%')
```

```
Out[58]: ([<matplotlib.patches.Wedge at 0x7ff30aa5c160>,
<matplotlib.patches.Wedge at 0x7ff30aa5c850>],
[Text(-0.3399187721714582, 1.046162142464278, 'Female'),
Text(0.3399188701202255, -1.0461621106387813, 'Male')],
[Text(-0.18541023936624992, 0.5706338958896061, '60.0%'),
Text(0.18541029279285026, -0.5706338785302443, '40.0%')])
```



Contingency table

To create a contingency table use `pandas.crosstab(index, columns)`. For an example see: <https://www.statology.org/contingency-table-python/> Here is how to create a contingency table for the categorical variables (sex, major) from the students data file stored in the dataframe called `df`:

```
In [59]: import pandas as pd
pd.crosstab(index=df['major'], columns=df['sex'], margins=True)
```

```
Out[59]:
```

	sex	Female	Male	All
major				
Business		9	6	15
Communications		4	1	5
International Affairs		4	5	9
Other		1	0	1
All		18	12	30

To express all frequencies as relative frequencies, divide by number of observations:

```
In [60]: n = len(df)      # number of observations
pd.crosstab(index=df['major'], columns=df['sex'], margins=True)/n
```



```
Out[60]:
```

	sex	Female	Male	All
major				
Business		0.300000	0.200000	0.500000
Communications		0.133333	0.033333	0.166667
International Affairs		0.133333	0.166667	0.300000
Other		0.033333	0.000000	0.033333
All		0.600000	0.400000	1.000000

... and multiply by 100 (percent) to get percentages:

```
In [61]: 100*pd.crosstab(index=df['major'], columns=df['sex'], margins=True)/n
```

```
Out[61]:
```

	sex	Female	Male	All
major				
Business		30.000000	20.000000	50.000000
Communications		13.333333	3.333333	16.666667
International Affairs		13.333333	16.666667	30.000000
Other		3.333333	0.000000	3.333333
All		60.000000	40.000000	100.000000

Summary statistics (qualitative variables only):

```
In [62]: df.describe(include=['object'])
```

```
Out[62]:
```

	sex	major
count	30	30
unique	2	4
top	Female	Business
freq	18	15

Summary statistics of one categorical variable (in this case: sex)

```
In [63]: df['sex'].describe()
```

```
Out[63]: count          30
unique           2
top            Female
freq            18
Name: sex, dtype: object
```

Module 3: Displaying and describing quantitative variables

Descriptive statistics (mean, median, standard deviation,...)

Summary statistics (all variables) using the `describe()` function from pandas:

```
In [64]: df.describe(include='all')
```

```
Out[64]:
```

	case	sex	height	weight	major
count	30.000000	30	30.000000	30.000000	30
unique	NaN	2	NaN	NaN	4
top	NaN	Female	NaN	NaN	Business
freq	NaN	18	NaN	NaN	15
mean	15.500000	NaN	174.033333	65.133333	NaN
std	8.803408	NaN	9.625696	13.356474	NaN
min	1.000000	NaN	154.000000	44.000000	NaN
25%	8.250000	NaN	170.000000	54.000000	NaN
50%	15.500000	NaN	172.000000	64.000000	NaN
75%	22.750000	NaN	178.000000	74.250000	NaN
max	30.000000	NaN	195.000000	94.000000	NaN

Summary statistics of one variable (in this case: height) using the `describe()` function from pandas

```
In [65]: df['height'].describe()
```

```
Out[65]: count      30.000000
mean       174.033333
std         9.625696
min        154.000000
25%        170.000000
50%        172.000000
75%        178.000000
max        195.000000
Name: height, dtype: float64
```

Summary statistics (quantitative variables only) using the `describe()` function from pandas:

```
In [66]: df.describe()
```

```
Out[66]:
```

	case	height	weight
count	30.000000	30.000000	30.000000
mean	15.500000	174.033333	65.133333
std	8.803408	9.625696	13.356474
min	1.000000	154.000000	44.000000
25%	8.250000	170.000000	54.000000
50%	15.500000	172.000000	64.000000
75%	22.750000	178.000000	74.250000
max	30.000000	195.000000	94.000000

Mean and standard deviation of one of the variables ('height') from the 'df' dataframe:

```
In [67]: import pandas as pd
df.height.mean() # height.mean() does not work: NameError
```

```
Out[67]: 174.03333333333333
```

```
In [68]: df.height.std()
```

```
Out[68]: 9.625695974240289
```

Other descriptive statistics:

```
In [69]: df.height.median() # similarly: min() ; max() ; sum() ; count() ; quantile()
```

```
Out[69]: 172.0
```

```
In [70]: df.height.quantile(q=0.50) # the 50th percentile is the same as the median
```

```
Out[70]: 172.0
```

```
In [71]: df.height.quantile(q=0.25) # the 25th percentile (the first quartile)
```

```
Out[71]: 170.0
```

```
In [72]: df.height.quantile(q=0.75) # the 75th percentile (the third quartile)
```

```
Out[72]: 178.0
```

```
In [73]: df.height.quantile(q=0.75)-df.height.quantile(q=0.25) # the interquartile range
```

```
Out[73]: 8.0
```

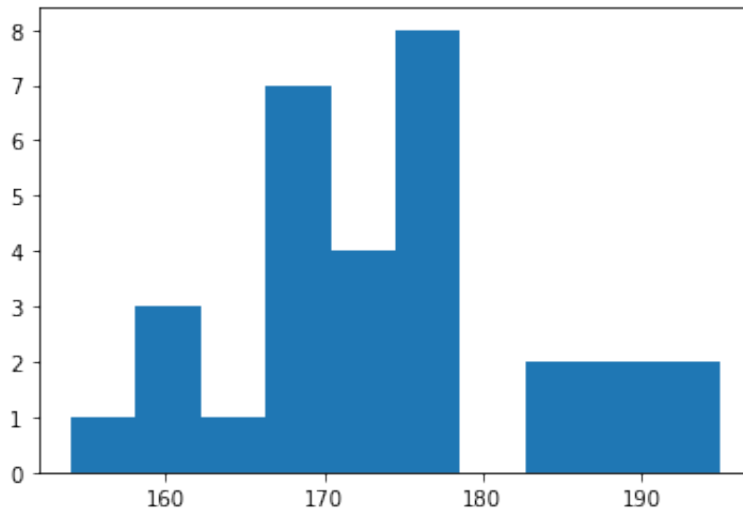
Histogram

To draw a **histogram** use `hist()` from the `matplotlib.pyplot` package.

Frequency histogram (vertical axis shows counts, absolute frequencies):

```
In [74]: import matplotlib.pyplot as plt
plt.hist(df.height) # the default is a frequency histogram: vertical axis
```

```
Out[74]: (array([1., 3., 1., 7., 4., 8., 0., 2., 2., 2.]),
array([154., 158.1, 162.2, 166.3, 170.4, 174.5, 178.6, 182.7, 186.8,
190.9, 195. ]),
<BarContainer object of 10 artists>)
```

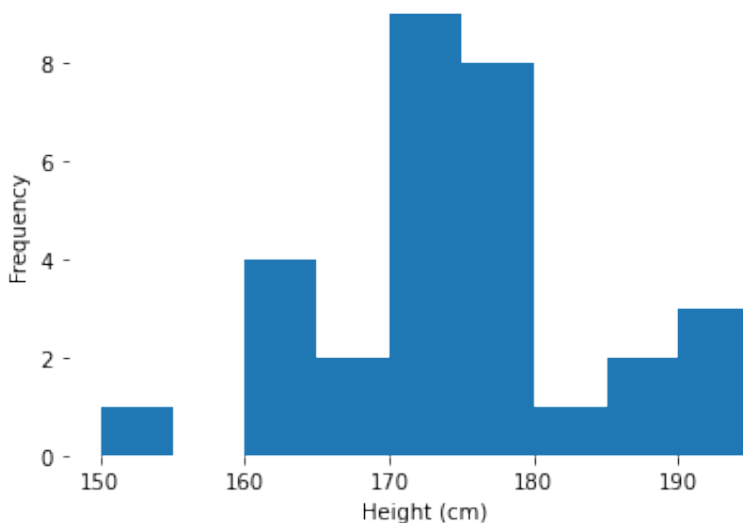


In the output, the first array gives the counts (absolute frequencies) for each of the classes (bins). The second array gives the edges of the bins (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html).

Get rid of the box, add labels to the axes, and let the bins start at 150, 155, 160,...:

```
In [75]: import matplotlib.pyplot as plt
plt.box(False) # get rid of the box
plt.xlabel('Height (cm)') # add label on x-axis
plt.ylabel('Frequency') # add label on y-axis
plt.hist(df.height, bins=[150, 155, 160, 165, 170, 175, 180, 185, 190, 195])
```

```
Out[75]: (array([1., 0., 4., 2., 9., 8., 1., 2., 3.]),
array([150, 155, 160, 165, 170, 175, 180, 185, 190, 195]),
<BarContainer object of 9 artists>)
```



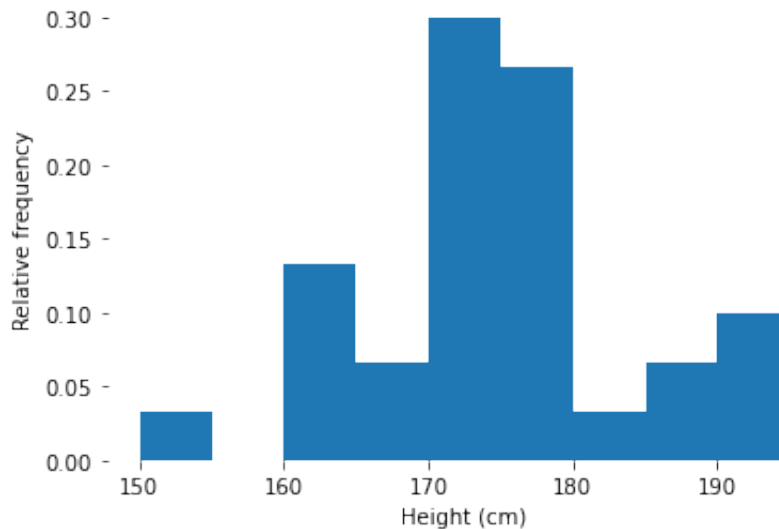
Relative frequency histogram (vertical axis shows relative frequencies):

```
In [76]: df.height.size # size gives the number of observations of the variable height
```

Out[76]: 30

```
In [77]: import matplotlib.pyplot as plt
import numpy as np
plt.box(False) # get rid of the box
plt.xlabel('Height (cm)') # add label on x-axis
plt.ylabel('Relative frequency') # add label on x-axis
plt.hist(df.height, weights=np.zeros_like(df.height) + 1. / df.height.size,
         bins=[150,155,160,165,170,175,180,185,190,195])
```

```
Out[77]: (array([0.03333333, 0.         , 0.13333333, 0.06666667, 0.3         ,
0.26666667, 0.03333333, 0.06666667, 0.1         ]),
array([150, 155, 160, 165, 170, 175, 180, 185, 190, 195]),
<BarContainer object of 9 artists>)
```

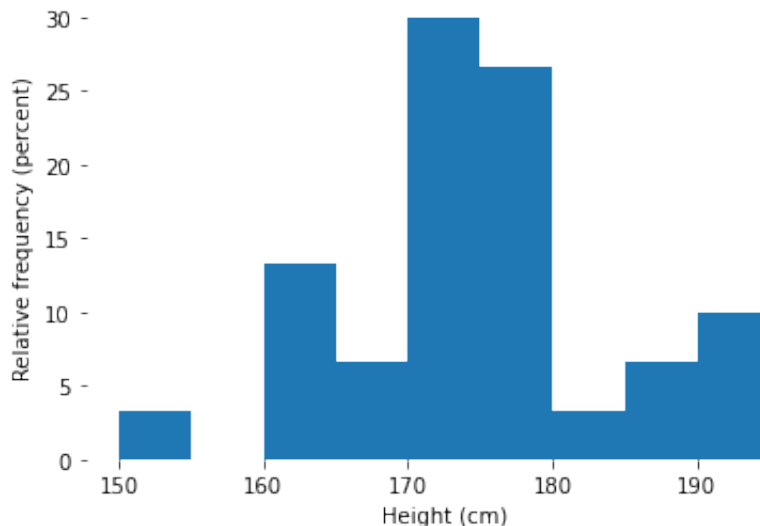


In the output, the first array gives the relative frequencies for each of the classes (bins). The second array gives the edges of the bins (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html).

To get a relative frequency histogram with relative frequencies expressed as percentages, multiply the weights by 100:

```
In [78]: import matplotlib.pyplot as plt
import numpy as np
df.height.size # size gives the number of observations of the variable height
plt.box(False) # get rid of the box
plt.xlabel('Height (cm)') # add label on x-axis
plt.ylabel('Relative frequency (percent)') # add label on x-axis
plt.hist(df.height, weights=100*(np.zeros_like(df.height) + 1. / df.height.size),
         bins=[150,155,160,165,170,175,180,185,190,195])
```

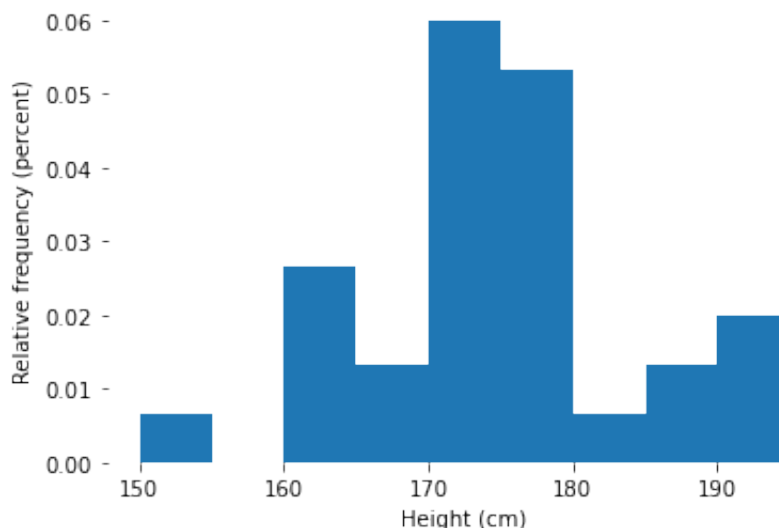
```
Out[78]: (array([ 3.33333333,  0.          , 13.33333333,  6.66666667, 30.          ,
                26.66666667,  3.33333333,  6.66666667, 10.          ]),
         array([150, 155, 160, 165, 170, 175, 180, 185, 190, 195]),
         <BarContainer object of 9 artists>)
```



Density histogram (vertical axis shows densities):

```
In [79]: import matplotlib.pyplot as plt
df.height.size # size gives the number of observations of the variable height
plt.box(False) # get rid of the box
plt.xlabel('Height (cm)') # add label on x-axis
plt.ylabel('Relative frequency (percent)') # add label on y-axis
plt.hist(df.height, density=True, bins=[150,155,160,165,170,175,180,185,190,195])
```

```
Out[79]: (array([0.00666667, 0.          , 0.02666667, 0.01333333, 0.06          ,
                0.05333333, 0.00666667, 0.01333333, 0.02          ]),
         array([150, 155, 160, 165, 170, 175, 180, 185, 190, 195]),
         <BarContainer object of 9 artists>)
```



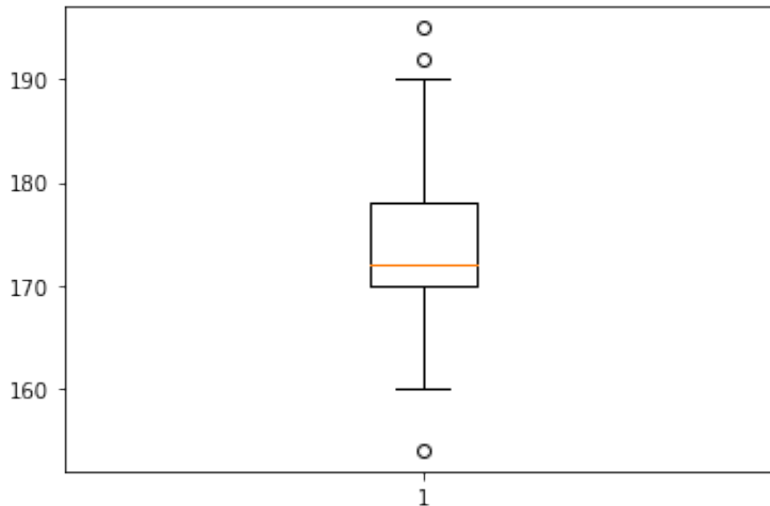
In the output, the first array gives the densities for each of the classes (bins). The second array gives the edges of the bins (https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html).

Box plot

Use `boxplot()` from `matplotlib`:

```
In [80]: import matplotlib.pyplot as plt
plt.boxplot(df.height)
```

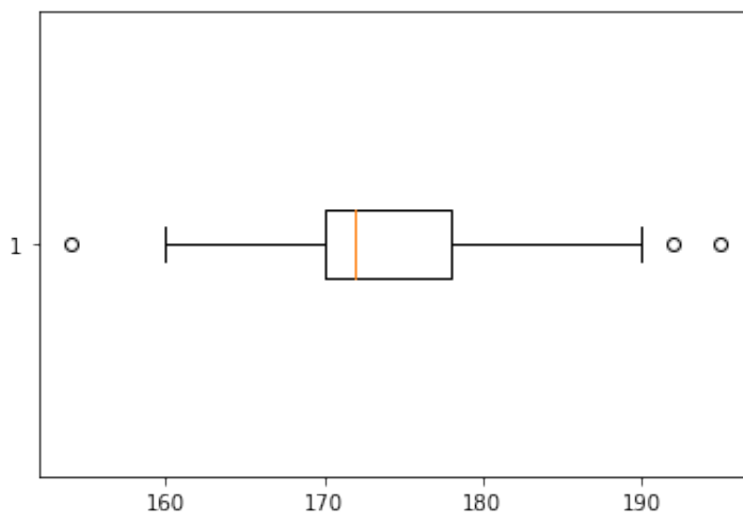
```
Out[80]: {'whiskers': [<matplotlib.lines.Line2D at 0x7ff2f8f4d7c0>,
<matplotlib.lines.Line2D at 0x7ff2f8f4db20>],
'caps': [<matplotlib.lines.Line2D at 0x7ff2f8f4de80>,
<matplotlib.lines.Line2D at 0x7ff2f8f58220>],
'boxes': [<matplotlib.lines.Line2D at 0x7ff2f8f4d460>],
'medians': [<matplotlib.lines.Line2D at 0x7ff2f8f58580>],
'fliers': [<matplotlib.lines.Line2D at 0x7ff2f8f588e0>],
'means': []}
```



Rotate the boxplot to get a horizontal orientation:

```
In [81]: import matplotlib.pyplot as plt
plt.boxplot(df.height,vert=False)
```

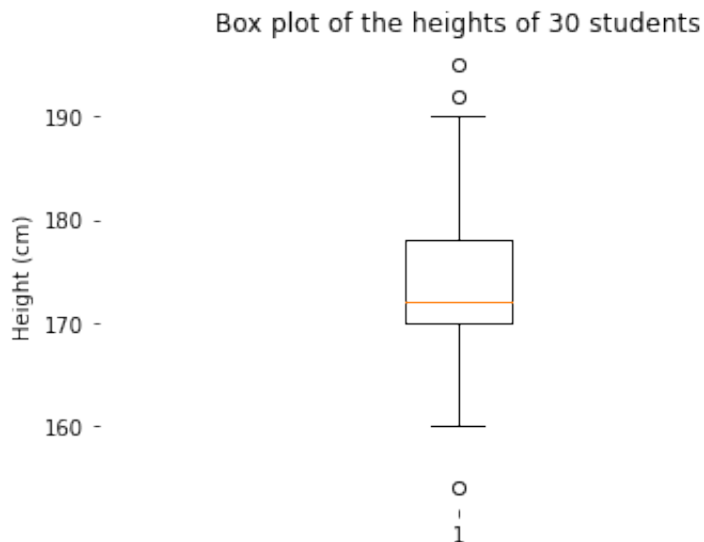
```
Out[81]: {'whiskers': [<matplotlib.lines.Line2D at 0x7ff2f9006370>,
<matplotlib.lines.Line2D at 0x7ff2f90066d0>],
'caps': [<matplotlib.lines.Line2D at 0x7ff2f9006a30>,
<matplotlib.lines.Line2D at 0x7ff2f9006d90>],
'boxes': [<matplotlib.lines.Line2D at 0x7ff2f8f99fd0>],
'medians': [<matplotlib.lines.Line2D at 0x7ff2f9012130>],
'fliers': [<matplotlib.lines.Line2D at 0x7ff2f9012490>],
'means': []}
```



Make the box plot prettier (get rid of the box, label the axes, add title) (still to fix: get rid of the "1" label):

```
In [82]: import matplotlib.pyplot as plt
plt.box(False) # get rid of the box
plt.title("Box plot of the heights of 30 students") # add title
plt.xlabel("") # I want to get rid of the 1 on the x-axis
plt.ylabel("Height (cm)") # add label to y-axis
plt.boxplot(df.height)
```

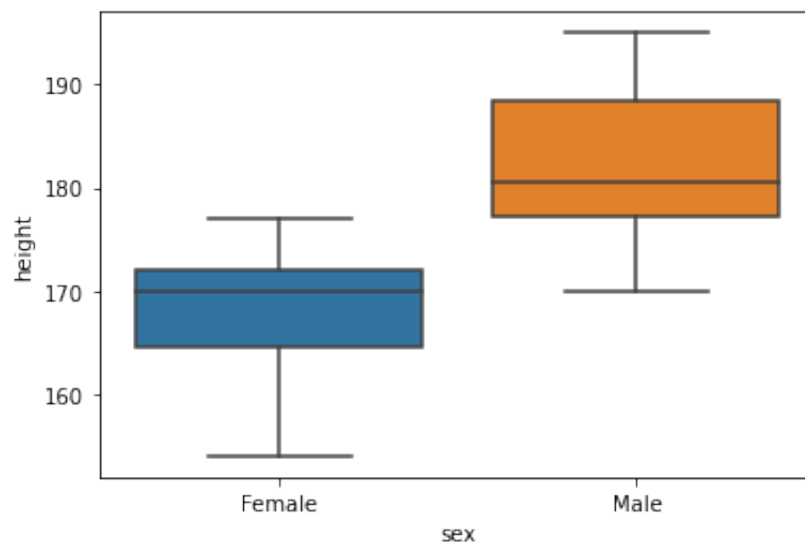
```
Out[82]: {'whiskers': [<matplotlib.lines.Line2D at 0x7ff2e8a184f0>,
<matplotlib.lines.Line2D at 0x7ff2e8a18850>],
'caps': [<matplotlib.lines.Line2D at 0x7ff2e8a18bb0>,
<matplotlib.lines.Line2D at 0x7ff2e8a18eb0>],
'boxes': [<matplotlib.lines.Line2D at 0x7ff2e8a18190>],
'medians': [<matplotlib.lines.Line2D at 0x7ff2e8a221f0>],
'fliers': [<matplotlib.lines.Line2D at 0x7ff2e8a22550>],
'means': []}
```



To compare the heights of men and women, use **side-by-side boxplots** (from the seaborn package):

```
In [83]: import seaborn as sns
sns.boxplot(data=df, x="sex", y='height')
```

```
Out[83]: <AxesSubplot:xlabel='sex', ylabel='height'>
```



```
In [84]: df.sex.count()
```

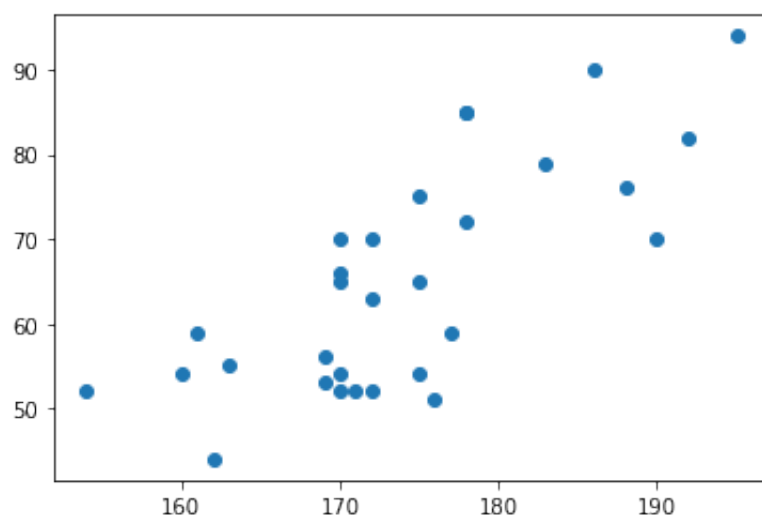

Out[84]: 30

Module 4: Correlation and Regression

Make a **scatter plot** of heights (horizontal axis) and weights (vertical axis) using `scatter()` from `matplotlib.pyplot`:

```
In [85]: import numpy as np
import matplotlib.pyplot as plt
plt.scatter(df.height, df.weight)
```

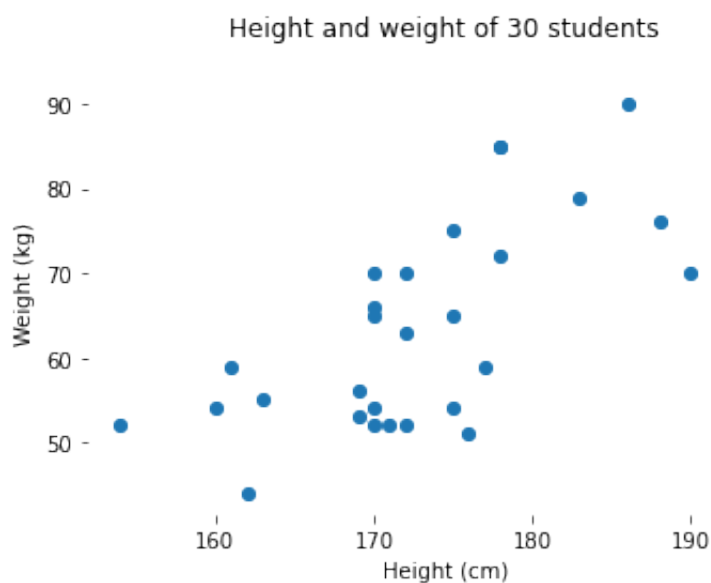
Out[85]: <matplotlib.collections.PathCollection at 0x7ff2f912b7f0>



Make the scatter plot prettier (get rid of the box, add labels to the axes):

```
In [86]: plt.box(False) # get rid of the box
plt.title("Height and weight of 30 students") # add title
plt.xlabel("Height (cm)") # add label to x-axis
plt.ylabel("Weight (kg)") # add label to y-axis
plt.scatter(df.height, df.weight)
```

Out[86]: <matplotlib.collections.PathCollection at 0x7ff2e8a61460>



Correlation coefficient:

```
In [90]: # correlation between height and weight:
df.height.corr(df.weight)
```

Out[90]: 0.7525668130284301

```
In [87]: # correlation matrix between all quantitative variables of a data frame:
df.corr()
```

Out[87]:

	case	height	weight
case	1.000000	-0.320457	-0.254260
height	-0.320457	1.000000	0.752567
weight	-0.254260	0.752567	1.000000

To get *one* of the correlation coefficients, first convert to matrix:

```
In [88]: import numpy as np
corr_matrix = np.array(df.corr())
print(corr_matrix)

[[ 1.          -0.32045686 -0.25426022]
 [-0.32045686  1.          0.75256681]
 [-0.25426022  0.75256681  1.          ]]
```

```
In [89]: # extract the correlation between height and weight (caution: rows and columns)
corr_matrix[1][2]
```

Out[89]: 0.7525668130284301

Find the **line of best fit** (using the ordinary least squares method) using `statsmodels` :

```
In [91]: import statsmodels.api as sms
import statsmodels.formula.api as smf
# Fit regression model:
results = smf.ols('df.weight ~ df.height', data=df).fit()
# Inspect the results:
print(results.summary())
```

```

OLS Regression Results
=====
Dep. Variable:          df.weight      R-squared:          0.566
Model:                  OLS            Adj. R-squared:      0.551
Method:                 Least Squares   F-statistic:         36.57
Date:                  Mon, 20 Dec 2021  Prob (F-statistic):  1.61e-06
Time:                  17:35:56         Log-Likelihood:      -107.29
No. Observations:      30              AIC:                21.86
Df Residuals:          28              BIC:                22.14
Df Model:               1
Covariance Type:       nonrobust
=====

```

	coef	std err	t	P> t	[0.025	0.975]
Intercept	-116.6011	30.097	-3.874	0.001	-178.252	-54.951
df.height	1.0443	0.173	6.047	0.000	0.691	1.398

```

=====
Omnibus:                2.804      Durbin-Watson:         2.486
Prob(Omnibus):          0.246      Jarque-Bera (JB):         1.333
Skew:                   0.007      Prob(JB):                 0.514
Kurtosis:               1.968      Cond. No.                 3.21e+03
=====

```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 3.21e+03. This might indicate that there are strong multicollinearity or other numerical problems.

To get just the coefficients:

```
In [92]: results.params
```

```
Out[92]: Intercept    -116.601087
df.height         1.044251
dtype: float64
```

The intercept is the first coefficient:

```
In [93]: results.params[0]
```

```
Out[93]: -116.6010867272883
```

The slope coefficient is the second coefficient:

```
In [94]: results.params[1]
```

```
Out[94]: 1.044250641987867
```

To get the t -values of the coefficients (Statistics II):

```
In [112... results.tvalues
```

```
Out[112... Intercept    -3.874200  
df.height      6.047248  
dtype: float64
```

To get the p -values of the coefficients (Statistics II):

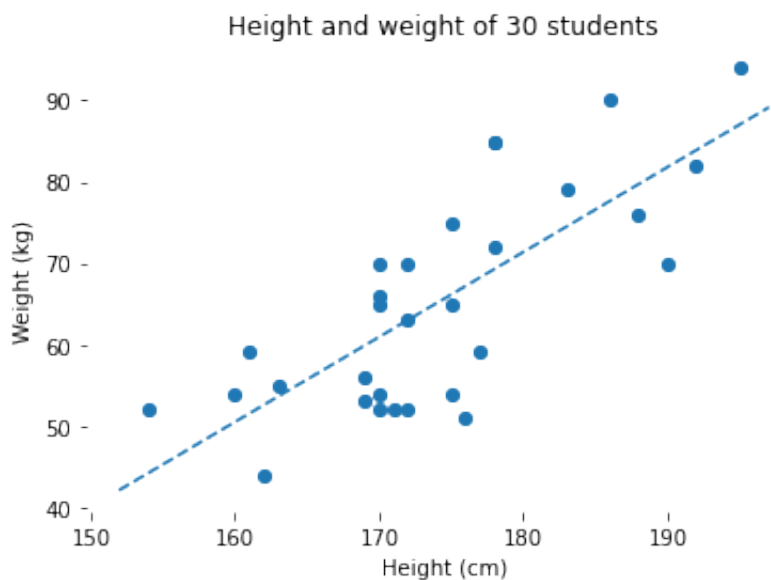
```
In [113... results.pvalues
```

```
Out[113... Intercept      0.000588  
df.height      0.000002  
dtype: float64
```

Add line of best fit to the scatter plot (see:

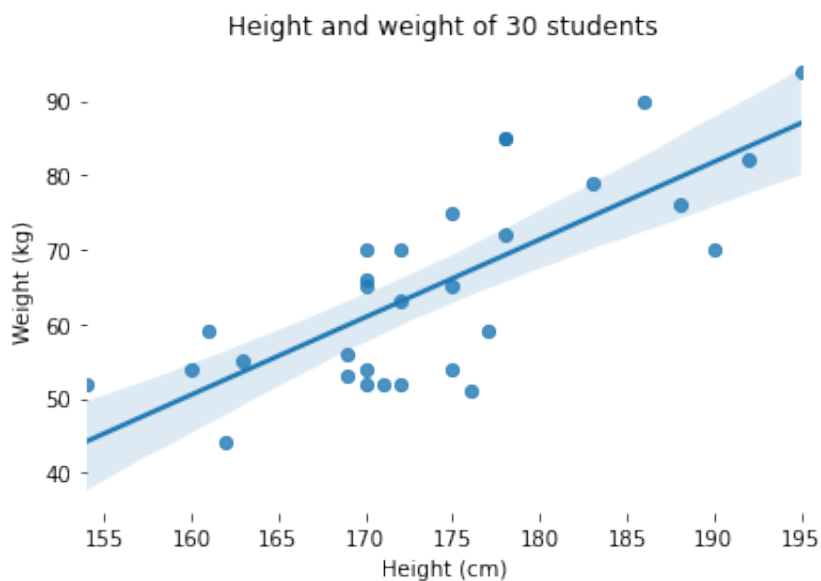
<https://stackoverflow.com/questions/7941226/how-to-add-line-based-on-slope-and-intercept-in-matplotlib>):

```
In [96]: import matplotlib.pyplot as plt  
import numpy as np  
import statsmodels.api as sms  
import statsmodels.formula.api as smf  
  
# Fit regression model:  
results = smf.ols('df.weight ~ df.height', data=df).fit()  
  
def abline(slope, intercept):  
    """Plot a line from slope and intercept"""  
    axes = plt.gca()  
    x_vals = np.array(axes.get_xlim())  
    y_vals = intercept + slope * x_vals  
    plt.plot(x_vals, y_vals, '--')  
  
plt.box(False) # get rid of the box  
plt.title("Height and weight of 30 students") # add title  
plt.xlabel("Height (cm)") # add label to x-axis  
plt.ylabel("Weight (kg)") # add label to y-axis  
plt.scatter(df.height, df.weight)  
abline(results.params[1], results.params[0]) # add line y=a*x+b (a = s
```



The `seaborn` package has more sophisticated ways to visualize data. Add a **line of best fit** to the scatter plot using the `seaborn` package:

```
In [119... import seaborn as sns
sns.regplot(x=df.height, y=df.weight, data=df) # regplot: regression plot
plt.box(False) # get rid of the box
plt.title("Height and weight of 30 students") # add title
plt.xlabel("Height (cm)") # add label to x-axis
plt.ylabel("Weight (kg)") # add label to y-axis
plt.show()
```



Module 5: Randomness and Probability

(No Python code.)

Module 6: Random variables and probability models

Calculate **binomial probability** (k = number of successes, n = number of trials, p = probability of success) (`pmf` stands for probability mass function—like `pdf` but for a discrete random variable):

```
In [99]: # calculate binomial probability (k= number of successes, n = number of trials)
from scipy.stats import binom
binom.pmf(k=10, n=12, p=0.6) # pmf: probability mass function (like pdf for continuous)
```

Out[99]: 0.06385228185599987

Calculate cumulative binomial probability:

```
In [100]: from scipy.stats import binom
binom.cdf(k=10, n=12, p=0.6)
```

Out[100]: 0.980408958976

Module 7: The Normal distribution

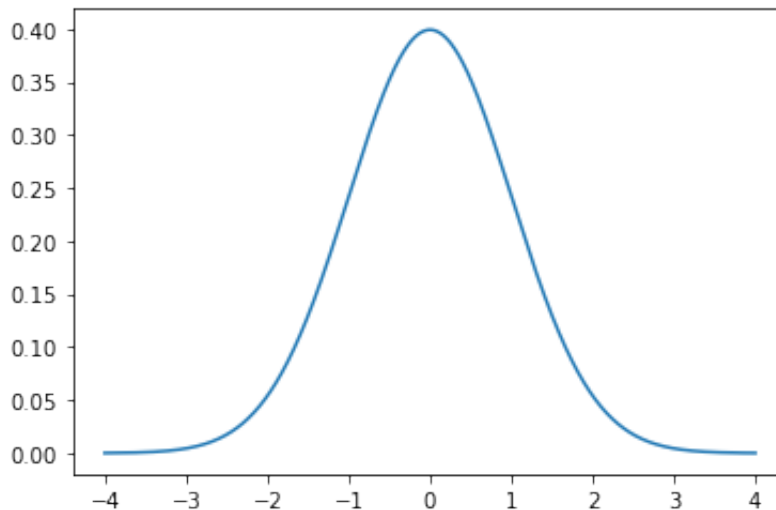
Plot the **probability density function** (pdf) of the normal curve

(<https://www.statology.org/plot-normal-distribution-python/>):

```
In [101]: import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
# x-axis ranges from -4 and 4 with .001 steps:
x = np.arange(-4, 4, 0.001)

# plot normal distribution with mean 0 and standard deviation 1
plt.plot(x, norm.pdf(x, 0, 1))
```

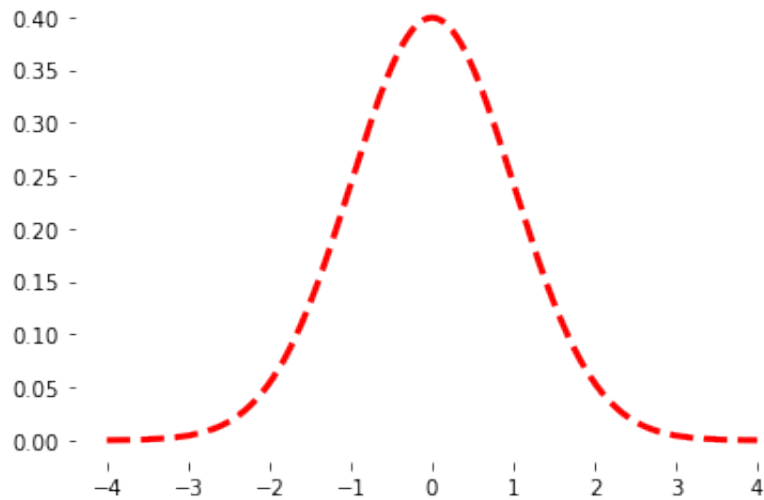
Out[101]: [<matplotlib.lines.Line2D at 0x7ff2e8af6f40>]



Get rid of the box, change color, linewidth, linestyle:

```
In [115]: plt.box(False)
plt.plot(x, norm.pdf(x, 0, 1), color='red', linewidth=3, linestyle='dashed')
```

Out[115... [



Area under normal curve:

To find an area under the normal curve, use the **cumulative density function** (cdf) of the normal distribution. (documentation: see:

<https://docs.scipy.org/doc/scipy/reference/stats.html>)

```
In [103... import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
mu = 0 # mean (for the standard normal distribution, mu = 0)
sigma = 1 # standard deviation (for the standard normal distribution, sigma = 1)
x1 = -1.96 # lower boundary
x2 = 1.96 # lower boundary
# area under normal curve between x1 and x2:
area = norm.cdf(x2, loc=mu, scale=sigma)-norm.cdf(x1, loc=mu, scale=sigma)
print('The area under the normal curve is', area)
```

The area under the normal curve is 0.950004209703559

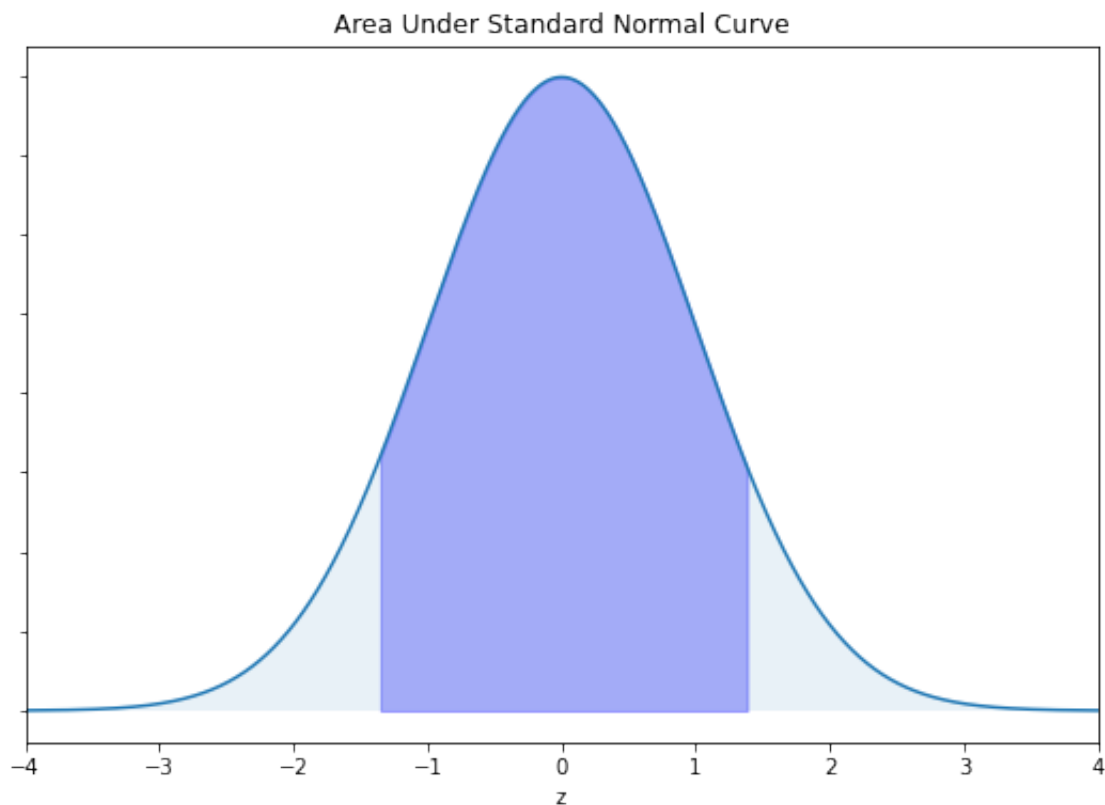
To **plot** the area under the normal curve (see:

<https://pythonforundergradengineers.com/plotting-normal-curve-with-python.html>)

In [104...

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
# define constants
mu = 998.8 # mean
sigma = 73.10 # standard deviation
x1 = 900 # lower boundary
x2 = 1100 # lower boundary
# calculate the standardized values:
z1 = ( x1 - mu ) / sigma
z2 = ( x2 - mu ) / sigma
x = np.arange(z1, z2, 0.001) # range of x in spec
x_all = np.arange(-10, 10, 0.001) # entire range of x, both in and out of spec
# for standard normal distribution, mean = 0, stddev = 1:
y = norm.pdf(x,0,1)
y2 = norm.pdf(x_all,0,1)
###
# build the plot
fig, ax = plt.subplots(figsize=(9,6))
ax.plot(x_all,y2)
ax.fill_between(x,y,0, alpha=0.3, color='b')
ax.fill_between(x_all,y2,0, alpha=0.1)
ax.set_xlim([-4,4])
ax.set_xlabel('z')
ax.set_yticklabels([])
ax.set_title('Area Under Standard Normal Curve')
plt.savefig('normal_curve.png', dpi=72, bbox_inches='tight')
plt.show()

# area under normal curve between x1 and x2:
area = norm.cdf(x2, loc=mu, scale=sigma)-norm.cdf(x1, loc=mu, scale=sigma)
print('The area under the normal curve is', area)
```



The area under the normal curve is 0.8286268028320297

Other continuous distributions

Uniform distribution

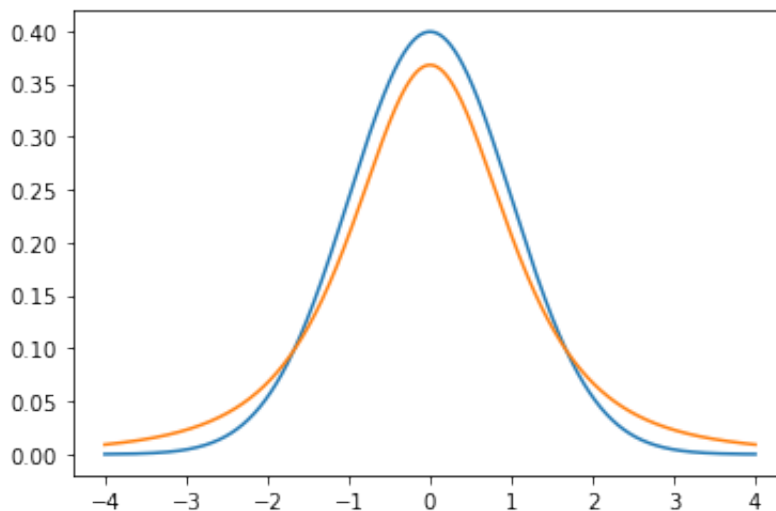
(<https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.uniform.html>):

```
In [105... from scipy.stats import uniform
# In the standard form, the distribution is uniform on [0, 1].
# Using the parameters loc and scale, one obtains the uniform distribution
uniform.cdf(0.9)
```

Out[105... 0.9

Student t distribution (covered in Statistics II)

```
In [106... import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import t
# x-axis ranges from -4 and 4 with .001 steps:
x = np.arange(-4, 4, 0.001)
plt.figure()
plt.plot(x, norm.pdf(x,0,1 )) # plot the standard normal curve as a benchmark
plt.plot(x, t.pdf(x, 3))      # the second argument is the degrees of freedom
plt.show()
```



Still to add: plotting Student t distributions with different degrees of freedom and compare with standard normal distribution (see Haslwanter (2016), p. 110); an animation in which degrees of freedom increase; an interactive diagram in which user can change degrees of freedom; areas under t distribution.

Module 9: Sampling Distributions and Confidence Intervals for Proportions

To find the **confidence interval for a proportion**:

(https://www.statsmodels.org/dev/generated/statsmodels.stats.proportion.proportion_confint)

```
In [107... 310 / 1126      # sample proportion = p=hat = number of successes / number of

import statsmodels.api as sm
from statsmodels.stats.proportion import proportion_confint # Function for

proportion_confint(count=310,      # count= number of successes
                    nobs=1126,      # nobs = number of trials
                    alpha=(1 - 0.95)) # alpha = 1 - confidence level
```

```
Out[107... (0.24922129423231776, 0.30140037539468045)
```

Module 10: Sampling Distributions and Confidence Intervals for Means

To find the **confidence interval for a mean** (in this case: the mean height of all students in the dataframe df):

```
In [108... import numpy as np
import scipy.stats as stats

degrees_of_freedom = len(df)-1      # degrees of freedom = sample size - 1
sample_mean        = np.mean(df.height) # sample mean
sample_standard_error = stats.sem(df.height) # sample standard error of the mean

# create confidence interval for the population mean:
stats.t.interval(alpha=0.05, df=degrees_of_freedom, loc=sample_mean, scale=
```

```
Out[108... (173.9221744669485, 174.14449219971817)
```

Statistics II: Hypothesis tests

For hypothesis tests (covered in Statistics II), see:

<https://www.statsmodels.org/stable/stats.html>

Interacting with the operating system (changing current working directory etc.)

The `os` package allows you to interact with the operating system using Python code.

The **current working directory** (cwd) is where Python will look for input (such as data files) and where it will store output (such as .png or .pdf figures and tables with results).

To find out what the current working directory of Python is:

```
In [109... import os
print(os.getcwd())
```

```
/Users/luchens/Documents/jupyter-notebooks
```

If you want to change the current working directory, use the `chdir` command of the `os` package to do so. Here is how to change the current working directory (the expression in quotes is the path to the new working directory — it will be a different path for you, of course):

In [110...

```
os.chdir('/Users/luchens/Documents/Data/')  
print(os.getcwd())
```

/Users/luchens/Documents/Data

To display the files and directories in the working directory, use `os.listdir()`