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.

Technology Help: Data (Ch. 1)

To import a text file with data in Python, you first have find the path to the file. The data file students.csv used un 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 practive 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 [1]:
```
import pandas as pd
df = pd.read_csv('https://luc-hens.github.io/students.csv', sep=',', skipir
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 [2]:
```
df.head()
```

Out[2]:

	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 [3]:
```
df.tail()
```

Out[3]:

	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 [4]:
```
df.head(10)
```

Out[4]:

	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 [5]:
```
list(df)
```

Out[5]:
```
['case', 'sex', 'height', 'weight', 'major']
```

Inspect the data types in the dataframe called df:

In [6]:
```
df.dtypes
```

Out[6]:
```
case       int64
sex       object
height     int64
weight     int64
major     object
dtype: object
```

To display just one of the variables:

In [7]:
```
print(df.height)      # How can I call variables just by their names:  print
```

```
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
```

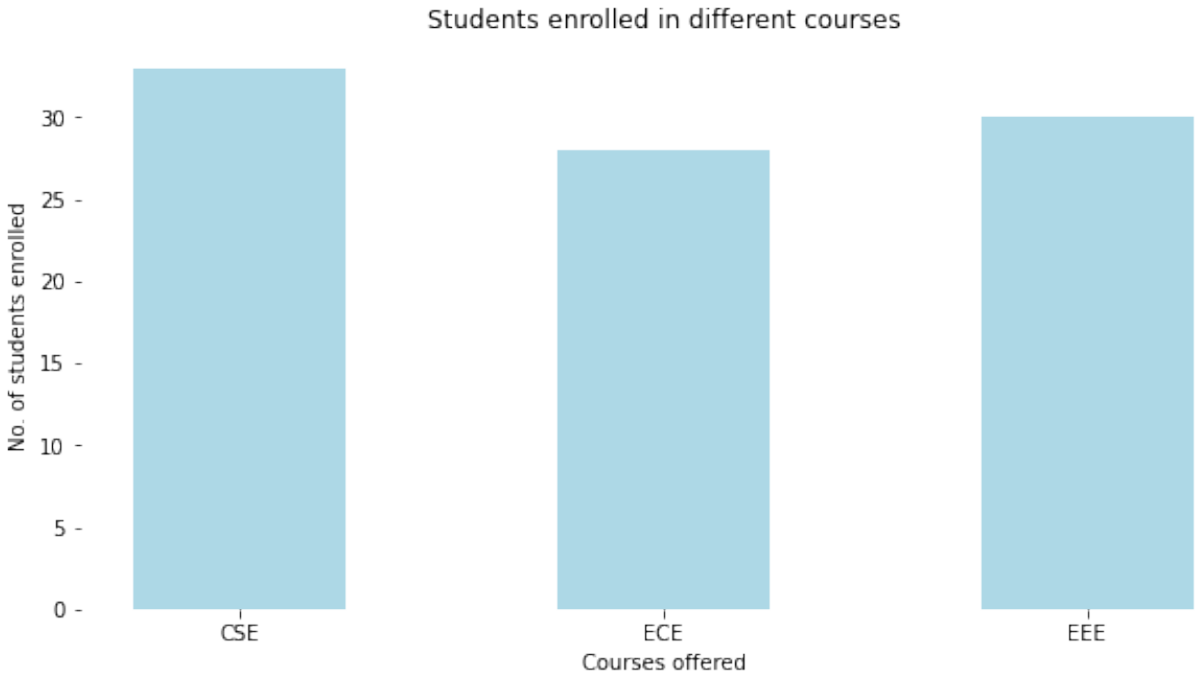
Displaying categorical variables: bar charts, pie charts

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/)

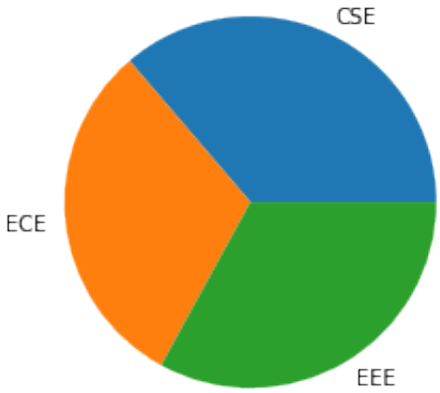
In [8]:
```
import numpy as np
import matplotlib.pyplot as plt
# Dataset generation
data_dict = {'CSE':33, 'ECE':28, 'EEE':30}
courses   = list(data_dict.keys())
values    = list(data_dict.values())
fig = plt.figure(figsize = (10, 5))
#  Bar plot
plt.box(False)              # get rid of the box
plt.bar(courses, values, color ='lightblue', width = 0.5)
plt.xlabel("Courses offered")
plt.ylabel("No. of students enrolled")
plt.title("Students enrolled in different courses")
plt.show()
```



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 [9]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.pie(values, labels=courses)
plt.show()
```



Still to do: generate bar chart and pie chart from imported data set (number of students in the majors from the students.csv data set). seaborn package has more advanced tools to display data.

Still to do: Contigency table (sex, major)

Descriptive statistics

Summary statistics (qualitative variables only):

In [10]:
```
df.describe(include=['object'])
```

Out[10]:

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

Summary statistics (all variables):

In [11]:
```
df.describe(include='all')
```

Out[11]:

	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)

In [12]:
```
df['height'].describe()
```

Out[12]:
```
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
```

In [13]:
```
# summary statistics (quantitative variables only) using the describe() fu
df.describe()
```

Out[13]:

	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

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

In [14]:
```
df['sex'].describe()
```

Out[14]:
```
count          30
unique          2
top        Female
freq           18
Name: sex, dtype: object
```

Mean and standard deviation of one of the variables

Summary statistics for one of the variables ('height'):

In [15]:
```
import pandas as pd
df.height.mean()                          # height.mean() does not work: Name
```

Out[15]:
```
174.03333333333333
```

In [16]:
```
df.height.std()
```

Out[16]:
```
9.625695974240289
```

Other descriptive statistics:

In [17]:
```
df.height.median()    # simularly: min() ; max() ; sum() ; count() ; quantil
```

Out[17]:
```
172.0
```

In [18]:
```
df.height.quantile(q=0.50)    # the 50th percentile is the same as the media
```

Out[18]:
```
172.0
```

In [19]:
```
df.height.quantile(q=0.25)  # the 25th percentile (the first quartile)
```

Out[19]:
```
170.0
```

In [20]:
```
df.height.quantile(q=0.75)  # the 75th percentile (the third quartile)
```

Out[20]:
```
178.0
```

In [21]:
```
df.height.quantile(q=0.75)-df.height.quantile(q=0.25) # the interquartile r
```

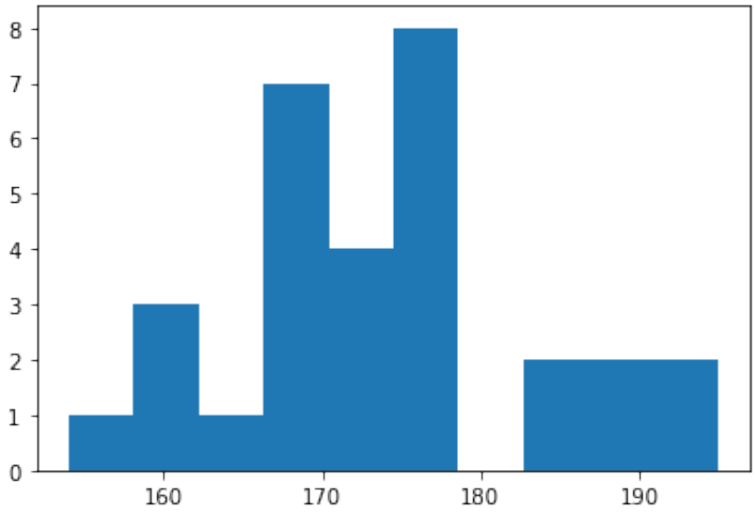
Out[21]:
```
8.0
```

Drawing a histogram:

Frequency histogram:

In [22]:
```
import matplotlib.pyplot as plt
plt.hist(df.height)     # frequency histogram: vertical axis shows counts (a
```

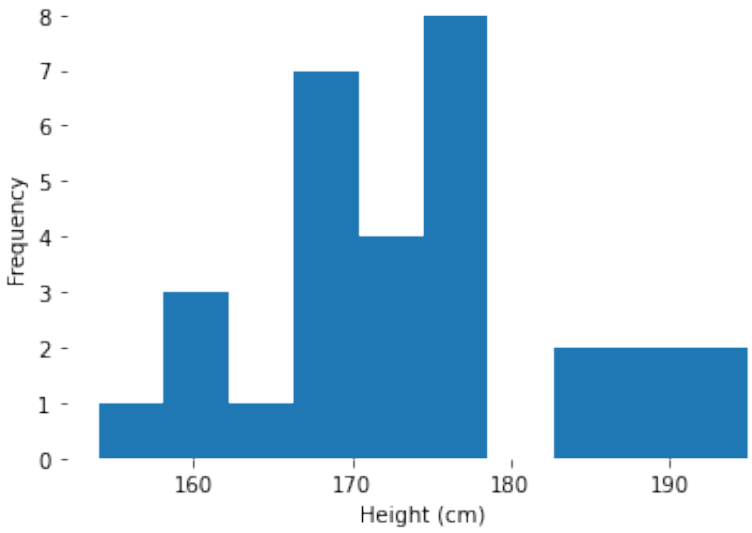
Out[22]:
```
(array([1., 3., 1., 7., 4., 8., 0., 2., 2., 2.]),
 array([154. , 158.1, 162.2, 166.3, 170.4, 174.5, 178.6, 182.7, 186.8,
        190.9, 195. ]),
 <BarContainer object of 10 artists>)
```



Get rid of the box and add labels to the axes:

In [23]:
```
plt.box(False)              # get rid of the box
plt.xlabel('Height (cm)')   # add label on x-axis
plt.ylabel('Frequency')     # add label on x-axis
plt.hist(df.height)         # frequency histogram: vertical axis shows cou
```

Out[23]:
```
(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>)
```



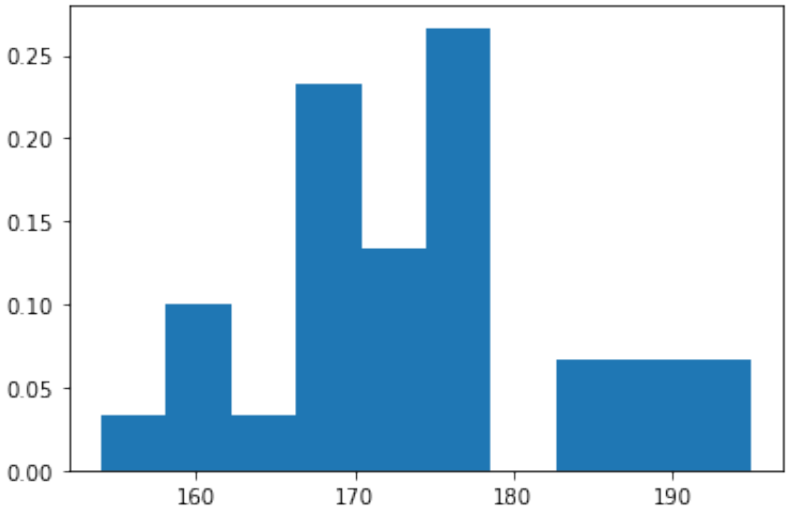
Relative frequency histogram:

In [24]:
```
df.height.size  # size gives the number of observations of the variable hei
```

Out[24]:
```
30
```

In [25]:
```
import numpy as np
plt.hist(df.height, weights=np.zeros_like(df.height) + 1. / df.height.size)
```

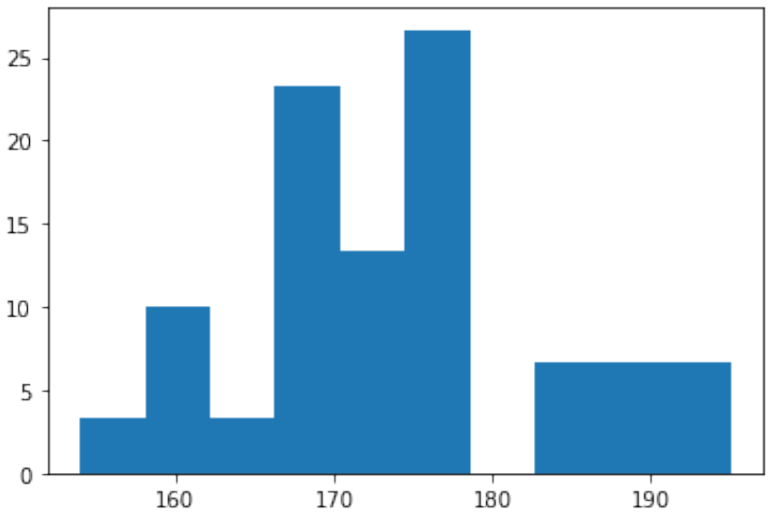
Out[25]:
```
(array([0.03333333, 0.1       , 0.03333333, 0.23333333, 0.13333333,
        0.26666667, 0.        , 0.06666667, 0.06666667, 0.06666667]),
 array([154. , 158.1, 162.2, 166.3, 170.4, 174.5, 178.6, 182.7, 186.8,
        190.9, 195. ]),
 <BarContainer object of 10 artists>)
```



Relative frequency histogram (percentages):

In [26]:
```
import numpy as np
df.height.size  # size gives the number of observations of the variable hei
plt.hist(df.height, weights=100*(np.zeros_like(df.height) + 1. / df.height.
```

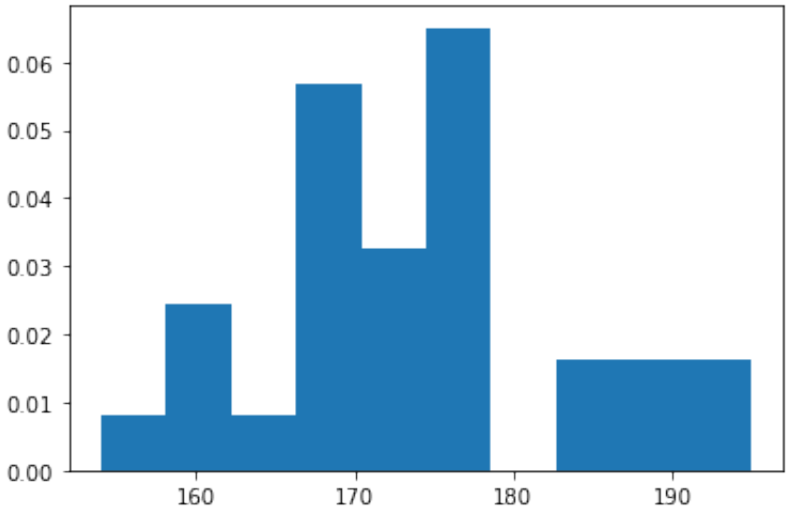
Out[26]:
```
(array([ 3.33333333, 10.        ,  3.33333333, 23.33333333, 13.33333333,
        26.66666667,  0.        ,  6.66666667,  6.66666667,  6.66666667]),
 array([154. , 158.1, 162.2, 166.3, 170.4, 174.5, 178.6, 182.7, 186.8,
        190.9, 195. ]),
 <BarContainer object of 10 artists>)
```



Density histogram:

In [27]:
```
plt.hist(df.height, density=True)    # density histogram: vertical axis sho
```

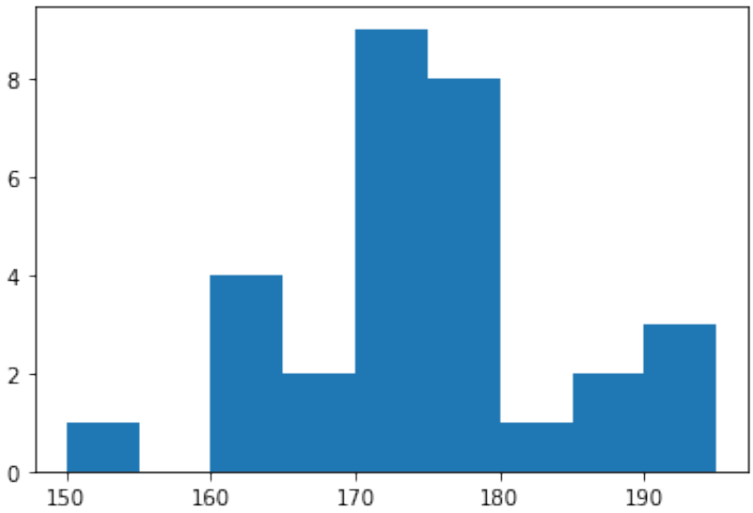
Out[27]:
```
(array([0.00813008, 0.02439024, 0.00813008, 0.05691057, 0.03252033,
        0.06504065, 0.        , 0.01626016, 0.01626016, 0.01626016]),
 array([154. , 158.1, 162.2, 166.3, 170.4, 174.5, 178.6, 182.7, 186.8,
        190.9, 195. ]),
 <BarContainer object of 10 artists>)
```



Histogram with bins starting at 150, 155, 160, ...:

In [28]:
```
plt.hist(df.height, bins=[150,155,160,165,170,175,180,185,190,195])
```

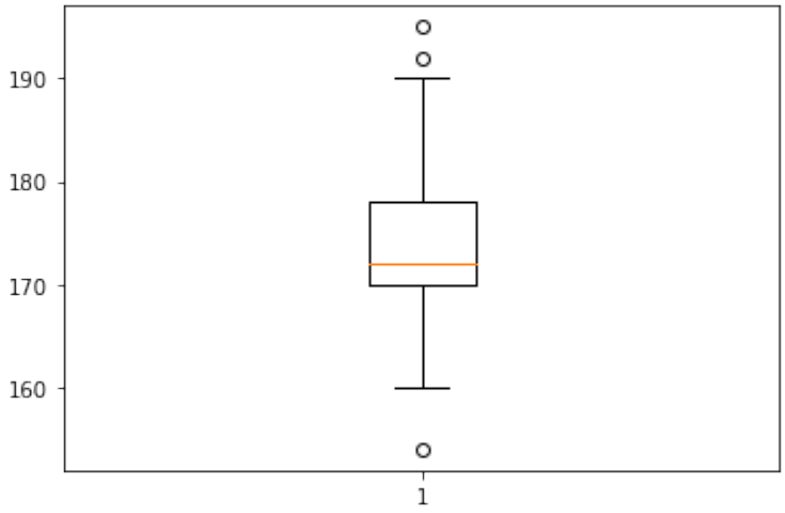
Out[28]:
```
(array([1., 0., 4., 2., 9., 8., 1., 2., 3.]),
 array([150, 155, 160, 165, 170, 175, 180, 185, 190, 195]),
 <BarContainer object of 9 artists>)
```



Box plot (from matplotlib):

In [29]:
```
import matplotlib.pyplot as plt
plt.boxplot(df.height)
```

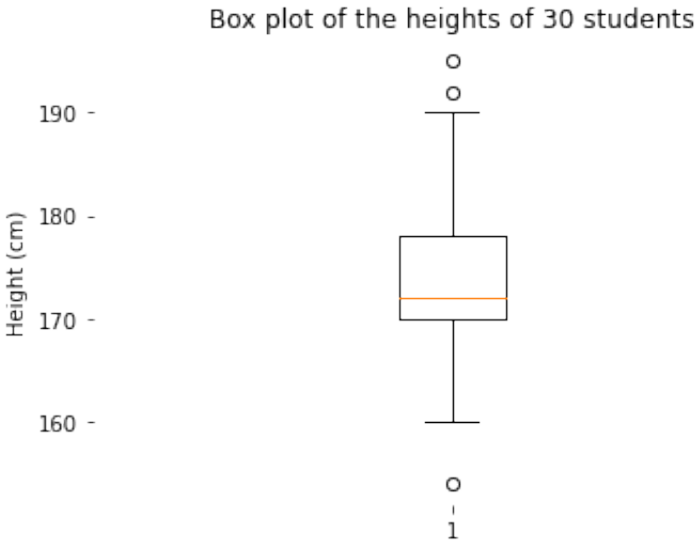
Out[29]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7fe158ba0a30>,
  <matplotlib.lines.Line2D at 0x7fe158ba0d90>],
 'caps': [<matplotlib.lines.Line2D at 0x7fe158baf130>,
  <matplotlib.lines.Line2D at 0x7fe158baf490>],
 'boxes': [<matplotlib.lines.Line2D at 0x7fe158ba06d0>],
 'medians': [<matplotlib.lines.Line2D at 0x7fe158baf7f0>],
 'fliers': [<matplotlib.lines.Line2D at 0x7fe158b938b0>],
 'means': []}
```



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

In [30]:
```
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)
```

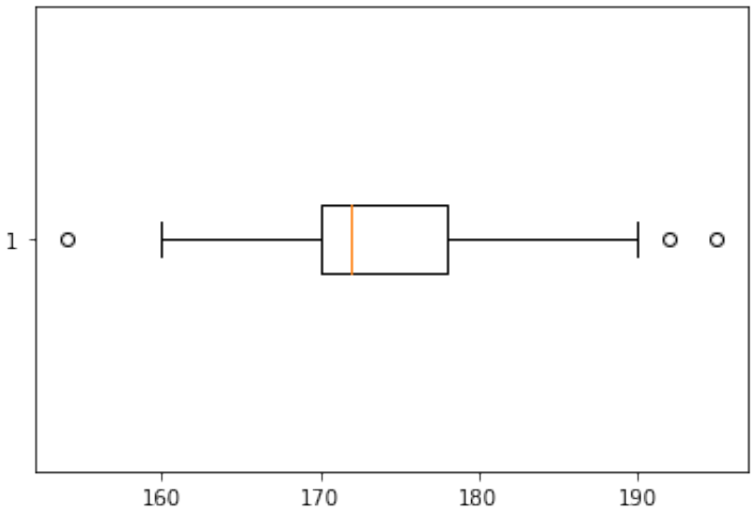
Out[30]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7fe17845d970>,
  <matplotlib.lines.Line2D at 0x7fe17845dcd0>],
 'caps': [<matplotlib.lines.Line2D at 0x7fe17846b070>,
  <matplotlib.lines.Line2D at 0x7fe17846b370>],
 'boxes': [<matplotlib.lines.Line2D at 0x7fe17845d6d0>],
 'medians': [<matplotlib.lines.Line2D at 0x7fe17846b6d0>],
 'fliers': [<matplotlib.lines.Line2D at 0x7fe17846ba30>],
 'means': []}
```



Rotate boxplot to get horizontal orientation:

In [31]:
```
import matplotlib.pyplot as plt
plt.boxplot(df.height,vert=False)
```

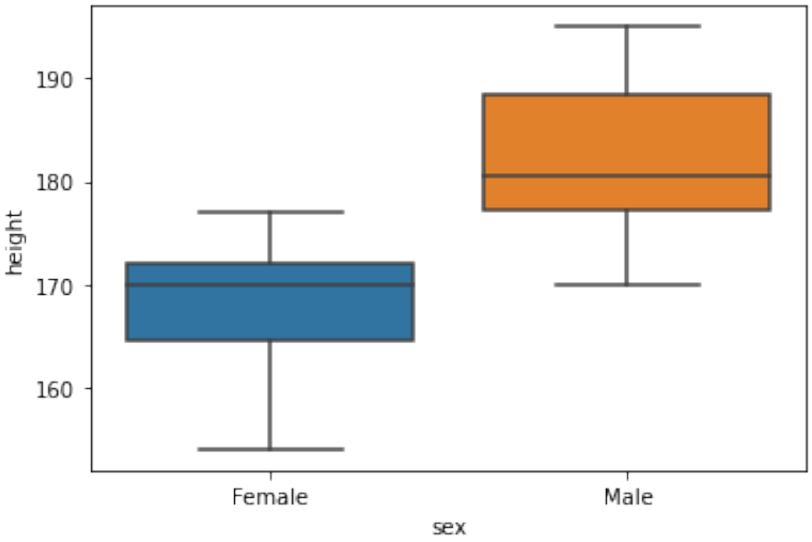
Out[31]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7fe16a030d90>,
  <matplotlib.lines.Line2D at 0x7fe16a03f130>],
 'caps': [<matplotlib.lines.Line2D at 0x7fe16a03f490>,
  <matplotlib.lines.Line2D at 0x7fe16a03f7f0>],
 'boxes': [<matplotlib.lines.Line2D at 0x7fe16a030a00>],
 'medians': [<matplotlib.lines.Line2D at 0x7fe16a03fb50>],
 'fliers': [<matplotlib.lines.Line2D at 0x7fe16a03feb0>],
 'means': []}
```



Side-by-side boxplots to compare the heights of men and women, using the seaborn package:

In [32]:
```
import seaborn as sns
sns.boxplot(data=df,x="sex",y='height')
```

Out[32]:
```
<AxesSubplot:xlabel='sex', ylabel='height'>
```



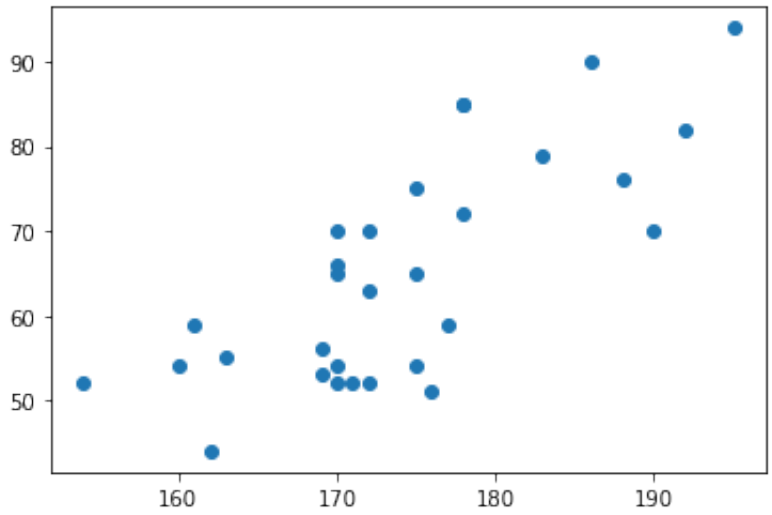
In [33]:
```
df.sex.count()
```

Out[33]:
```
30
```

Scatter plot and correlation

In [34]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.scatter(df.height,df.weight)
```

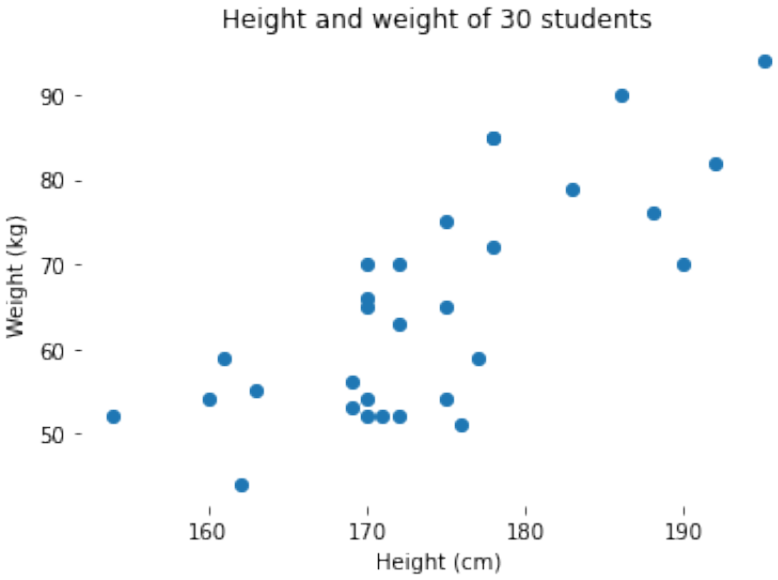
Out[34]:
```
<matplotlib.collections.PathCollection at 0x7fe16acb3340>
```



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

In [35]:
```
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)
```

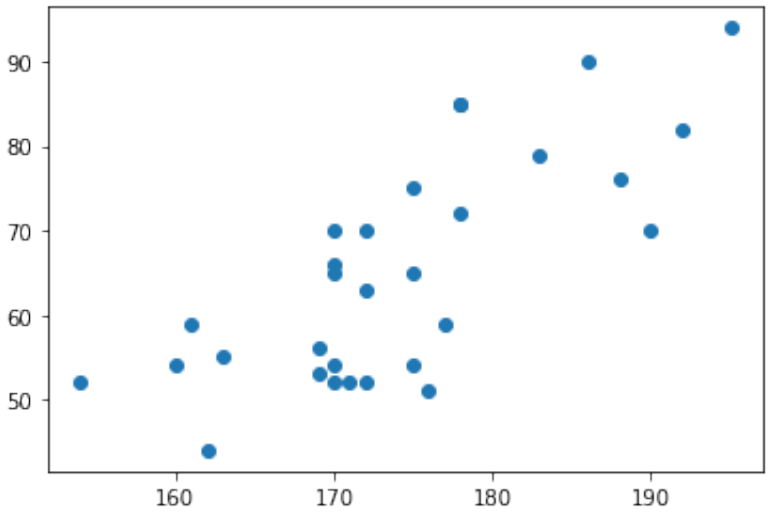
Out[35]:
```
<matplotlib.collections.PathCollection at 0x7fe178589e80>
```



Save plot to .png:

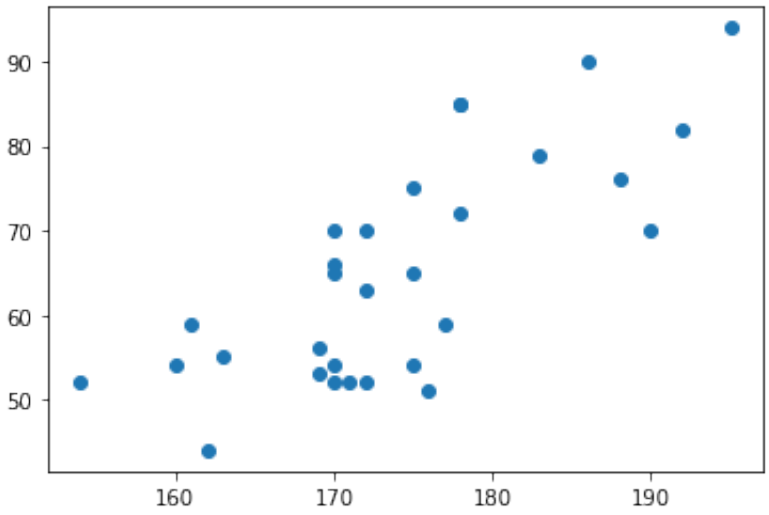
In [36]:
```
fig = plt.figure()
plt.scatter(df.height,df.weight)
fig.savefig('saved_figure-1000dpi.png', dpi = 1000, transparent=True)
# the plot is saved to the current working directory (cwd)
# to find out what the current working directory (cwd) is:
import os
print('The file is saved to the current working directory: ',os.getcwd())
```

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



Save plot to .pdf:

In [37]:
```
fig = plt.figure()
plt.scatter(df.height,df.weight)
fig.savefig('saved_figure-1000dpi.pdf', dpi = 1000, transparent=True)
```



Correlation coefficient:

In [38]:
```
# correlation matrix between all quantitative variables of a data frame:
df.corr()
```

Out[38]:

	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 [39]:
```
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 [40]:
```
# extract the correlation between height and weight (caution: rows and colu
corr_matrix[1][2]
```

Out[40]:
```
0.7525668130284301
```

In [41]:
```
# correlation between height and weight (directly, without computing the co
df.height.corr(df.weight)
```

Out[41]:
```
0.7525668130284301
```

Find the **line of best fit** using statsmodels:

In [42]:
```
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:                Thu, 16 Dec 2021   Prob (F-statistic):           1.61e
-06
Time:                        20:06:38   Log-Likelihood:                -107
.29
No. Observations:                  30   AIC:                             21
8.6
Df Residuals:                      28   BIC:                             22
1.4
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
===
                 coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
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 corr
ectly 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 [43]:
```
results.params
```

Out[43]:
```
Intercept   -116.601087
df.height      1.044251
dtype: float64
```

To get the first coefficient (intercept):

In [44]:
```
results.params[0]
```

Out[44]:
```
-116.6010867272883
```

To get the second coefficient (slope coefficient):

In [45]:
```
results.params[1]
```

Out[45]:
```
1.044250641987867
```

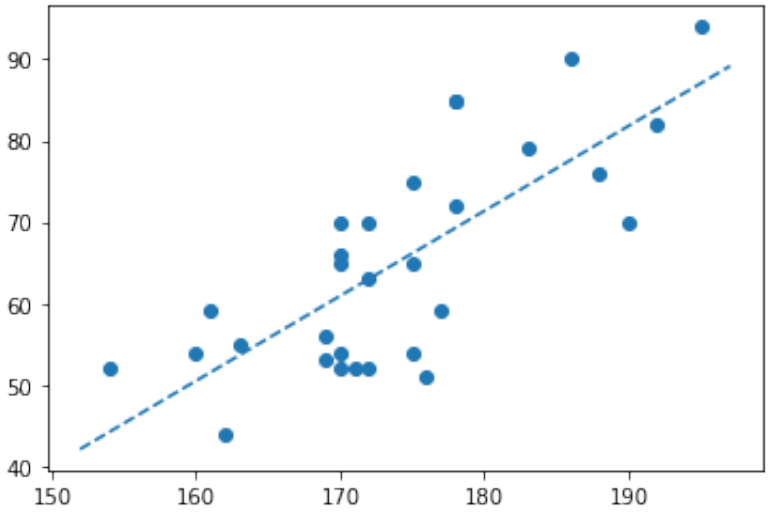
To get the t-values of the coefficients:

In [46]:
```
results.tvalues
```

Out[46]:
```
Intercept   -3.874200
df.height    6.047248
dtype: float64
```

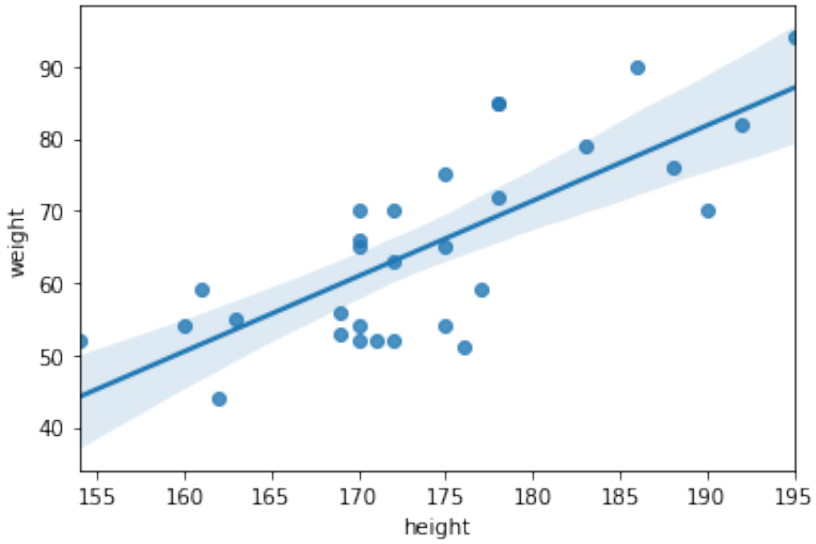
Add line of best fit to the scatter plot from matplotlib:

In [47]:
```
import matplotlib.pyplot as plt
import numpy as np
# From: https://stackoverflow.com/questions/7941226/how-to-add-line-based-
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.scatter(df.height,df.weight)
abline(results.params[1],results.params[0])
```



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

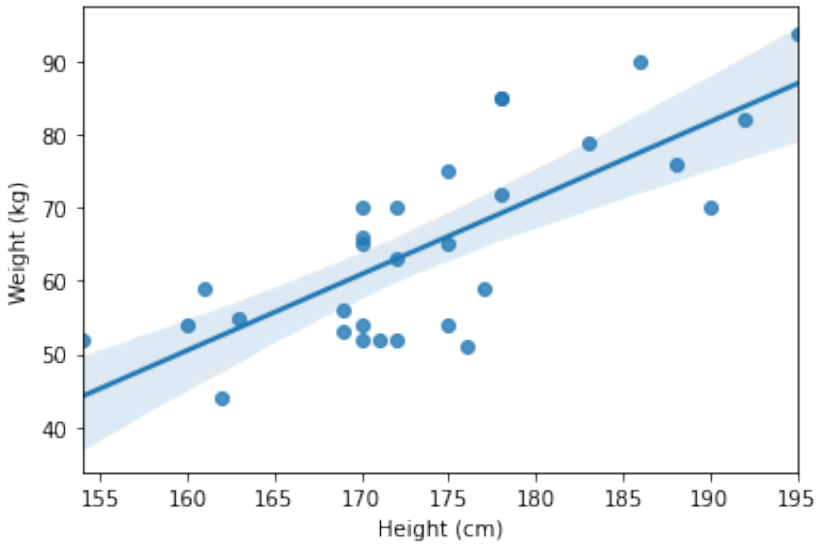
In [48]:
```
import seaborn as sns   # regplot: regression plot
sns.regplot(x=df.height,y=df.weight, data=df)
plt.show()
```



Label the axes:

In [49]:
```
x, y = pd.Series(df.height, name="Height (cm)"), pd.Series(df.weight, name=
sns.regplot(x=x,y=y, data=df)
```

Out[49]:
```
<AxesSubplot:xlabel='Height (cm)', ylabel='Weight (kg)'>
```



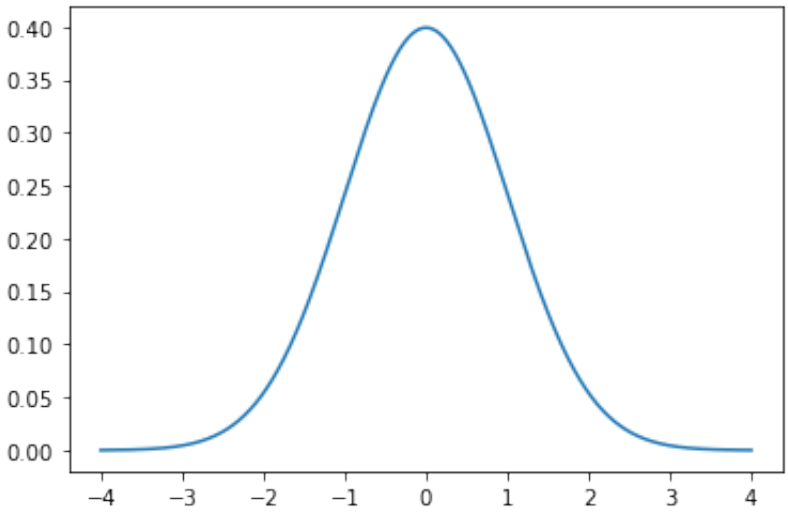
Plot Normal curve:

To plot the **probability density function** (pdf) of the normal curve (https://www.statology.org/plot-normal-distribution-python/):

In [50]:
```
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))
```

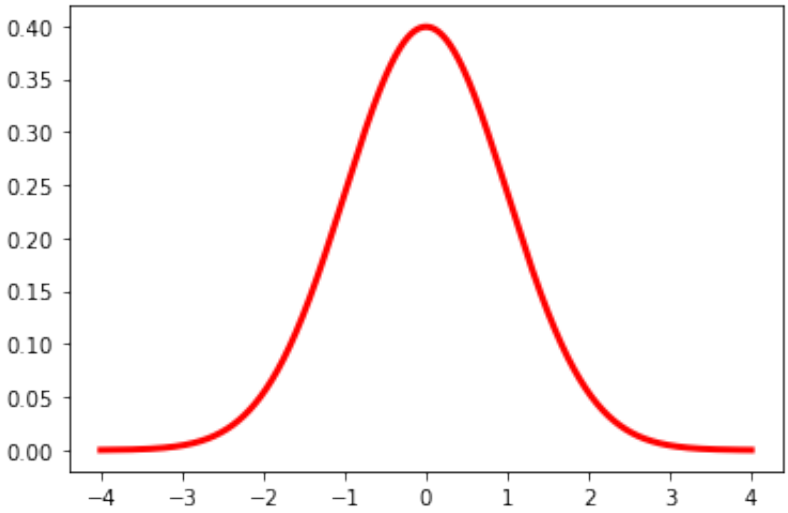
Out[50]: [<matplotlib.lines.Line2D at 0x7fe158e51310>]



Change color, linewidth:

In [51]:
```
plt.plot(x, norm.pdf(x, 0, 1), color='red', linewidth=3)
```

Out[51]: [<matplotlib.lines.Line2D at 0x7fe178861940>]



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)

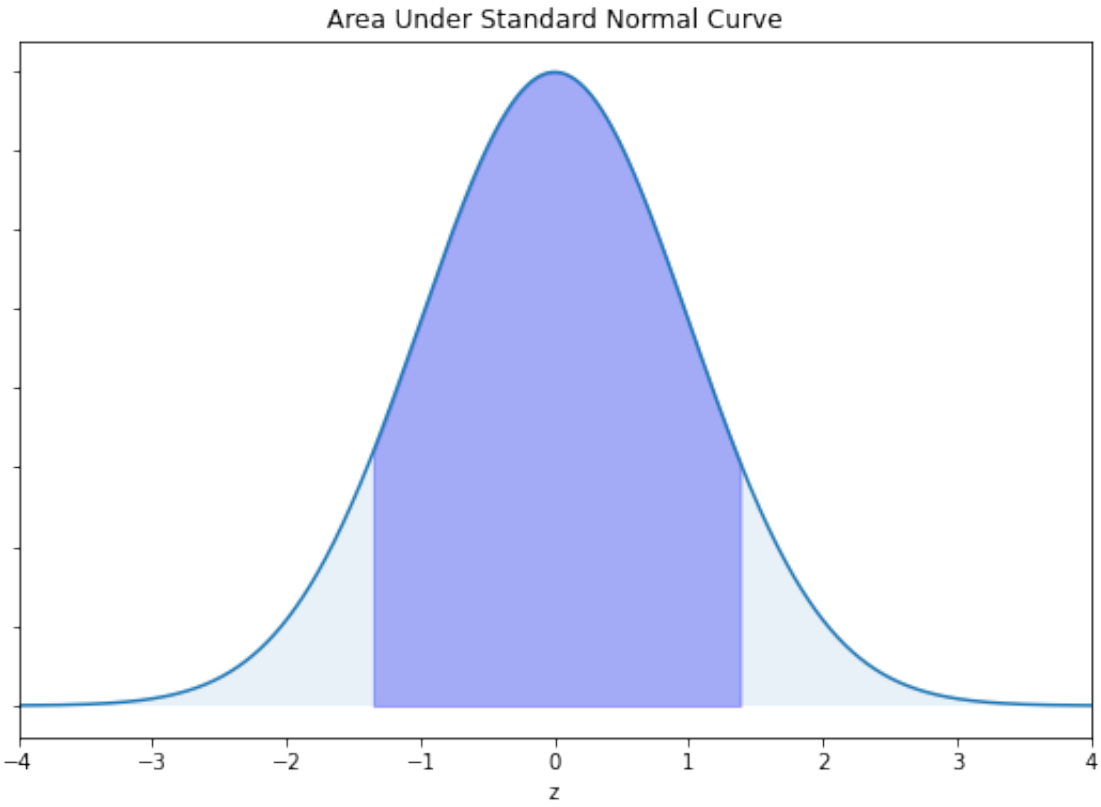
In [52]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
mu    = 0     # mean                (for the standard normal distribution, t
sigma = 1     # standard deviation (for the standard normal distribution, t
x1 = -1.96    # lower boundary
x2 = 1.96   # lower boundary
# area under normal curve between x1 and x2:
norm.cdf(x2, loc=mu, scale=sigma)-norm.cdf(x1, loc=mu, scale=sigma)
```

Out[52]: 0.950004209703559

To **plot** the area under the normal curve (see: https://pythonforundergradengineers.com/plotting-normal-curve-with-python.html)

In [53]:
```
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 s
# 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:
norm.cdf(x2, loc=mu, scale=sigma)-norm.cdf(x1, loc=mu, scale=sigma)
```



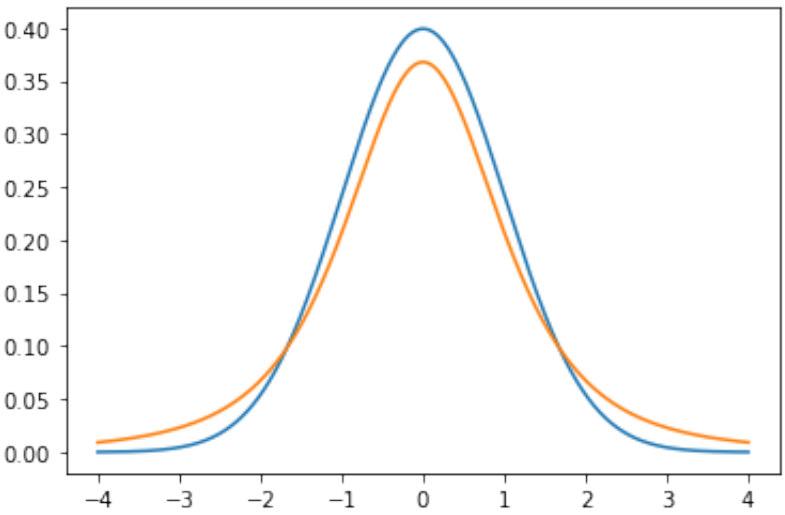
Out[53]: 0.8286268028320297

Other distributions

Still to add: pdf and cdf of other distributions (uniform, binomial)

Student *t* distribution

In [54]:
```
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 benc
plt.plot(x, t.pdf(x, 3))          # the second argument is the degrees of fre
plt.show()
```



Still to add: plotting Student t distributions with different degrees of freedom and compare with standard normal distribution. If time permits, make an animation in which degrees of freedom increase; make a interactive chart in which user can change degrees of freedom. Areas under t distribution.

Confidence intervals for proportion and mean

Still to add: confidence intervals.

Hypothesis tests

Still to add: Hypothesis tests (covered in Statistics II).

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

In [ ]:
```
```

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

The current working directory (cwd) of Python is where the program will look for data files. To find out what the current working directory of Python is:

In [55]:
```
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 [56]:
```
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()`