Statistics

Yair Mau

Invalid Date

Table of contents

Preface

I read Mike X Cohen's excellent book "Modern Statistics", and now it's time to practice.

Part I

data

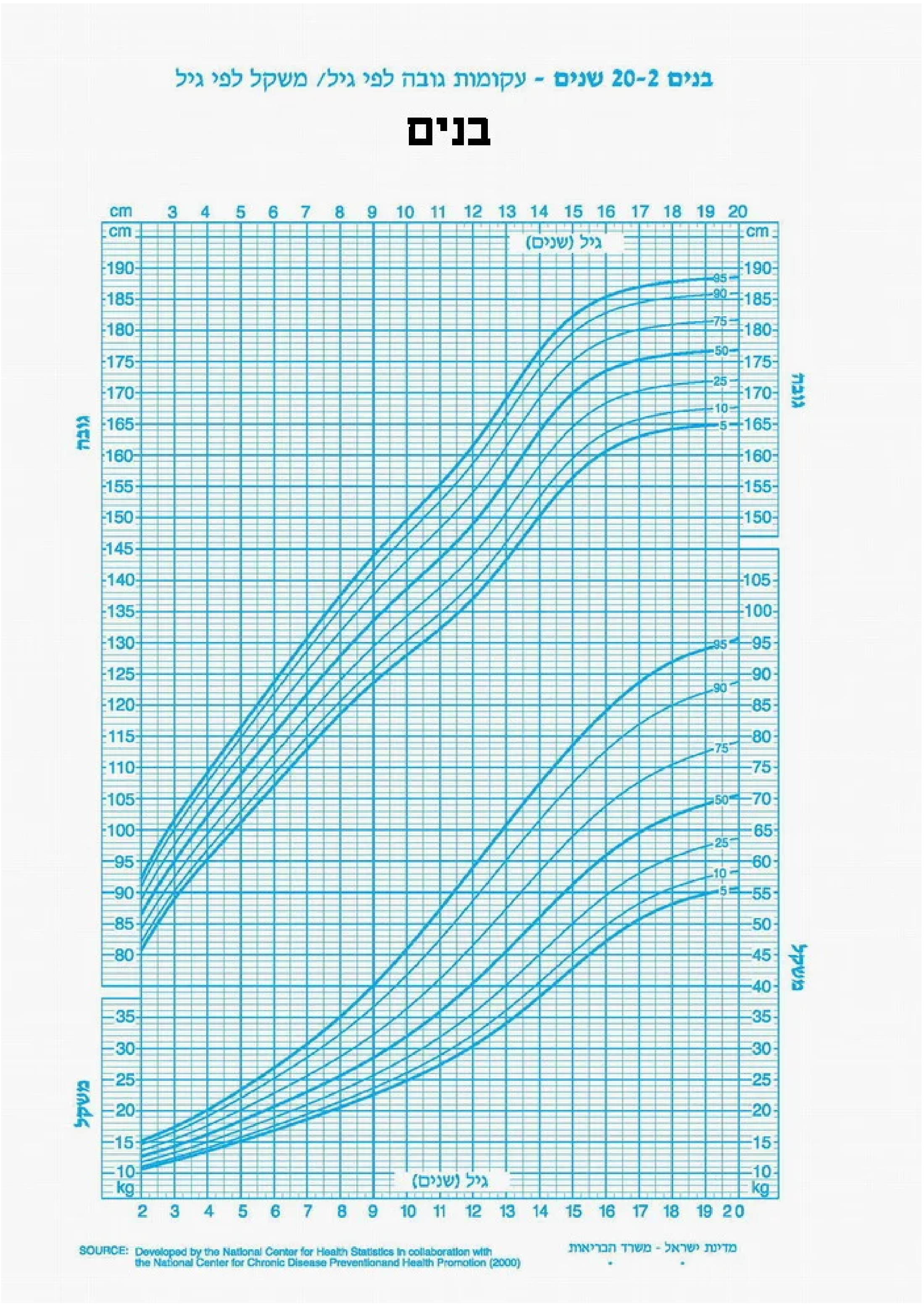
1 height data

I found growth curves for girls and boys in Israel:

- url girls, pdf girls
- url boys, pdf boys
- url both, png boys, png girls.

For example, see this:

בנים


בנים 2-20 שנים - עקומות גובה לפי גיל/ משקל לפי גיל
גיל (שנים)
גובה
משקל
cm
kg
SOURCE: Developed by the National Center for Health Statistics in collaboration with the National Center for Chronic Disease Preventionand Health Promotion (2000)
מדינת ישראל - משרד הבריאות

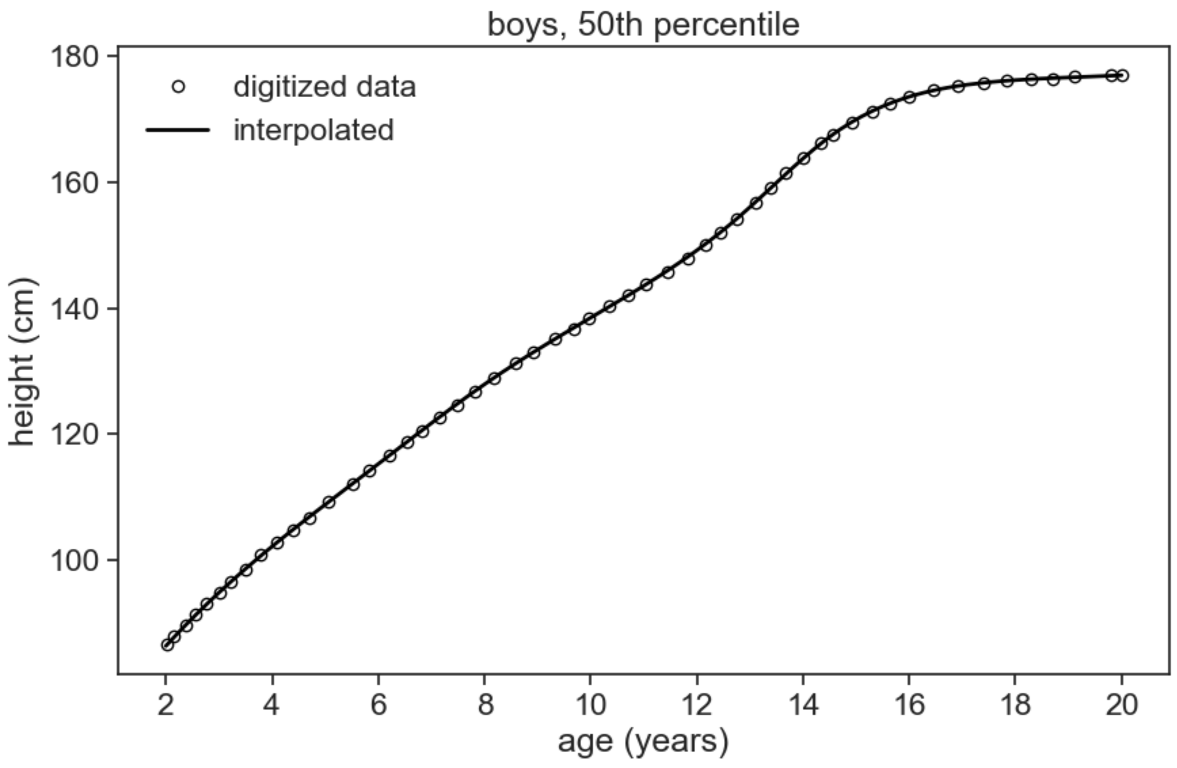
I used the great online resource Web Plot Digitizer v4 to extract the data from the images files. I captured all the growth curves as best as I could. The first step now is to get interpolated versions of the digitized data. For instance, see below the 50th percentile for boys:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style="ticks", font_scale=1.5)
from scipy.optimize import curve_fit
from scipy.special import erf
from scipy.interpolate import UnivariateSpline
import matplotlib.animation as animation
from scipy.stats import norm
import plotly.graph_objects as go
import plotly.io as pio
pio.renderers.default = 'notebook'
# %matplotlib widget
```

```
age_list = np.round(np.arange(2.0, 20.1, 0.1), 1)
height_list = np.round(np.arange(70, 220, 0.1), 1)
```

```
df_temp_boys_50th = pd.read_csv('../archive/data/height/boys-p50.csv', names=['age','height'
spline = UnivariateSpline(df_temp_boys_50th['age'], df_temp_boys_50th['height'], s=0.5)
interpolated = spline(age_list)
```

```
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(df_temp_boys_50th['age'], df_temp_boys_50th['height'], label='digitized data',
        marker='o', markerfacecolor='None', markeredgecolor="black", markersize=6, linestyle
ax.plot(age_list, interpolated, label='interpolated', color="black", linewidth=2)
ax.set(xlabel='age (years)',
       ylabel='height (cm)',
       xticks=np.arange(2, 21, 2),
       title="boys, 50th percentile"
       )
ax.legend(frameon=False);
```



Let's do the same for all the other curves, and then save them to a file.

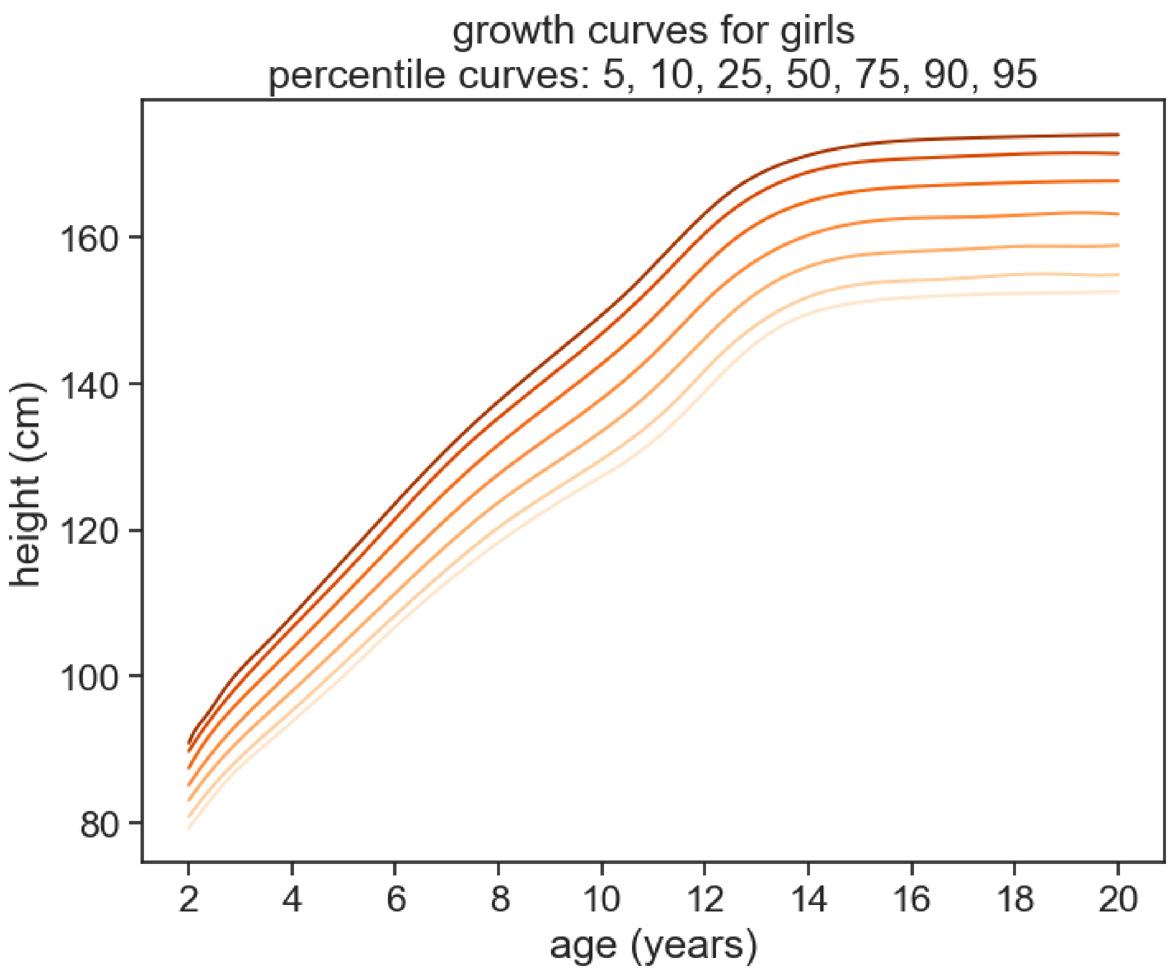
```
col_names = ['p05', 'p10', 'p25', 'p50', 'p75', 'p90', 'p95']
file_names_boys = ['boys-p05.csv', 'boys-p10.csv', 'boys-p25.csv', 'boys-p50.csv',
                   'boys-p75.csv', 'boys-p90.csv', 'boys-p95.csv',]
file_names_girls = ['girls-p05.csv', 'girls-p10.csv', 'girls-p25.csv', 'girls-p50.csv',
                   'girls-p75.csv', 'girls-p90.csv', 'girls-p95.csv',]

# create dataframe with age column
df_boys = pd.DataFrame({'age': age_list})
df_girls = pd.DataFrame({'age': age_list})
# loop over file names and read in data
for i, file_name in enumerate(file_names_boys):
    # read in data
    df_temp = pd.read_csv('../archive/data/height/' + file_name, names=['age','height'])
    spline = UnivariateSpline(df_temp['age'], df_temp['height'], s=0.5)
    df_boys[col_names[i]] = spline(age_list)
for i, file_name in enumerate(file_names_girls):
    # read in data
    df_temp = pd.read_csv('../archive/data/height/' + file_name, names=['age','height'])
    spline = UnivariateSpline(df_temp['age'], df_temp['height'], s=0.5)
    df_girls[col_names[i]] = spline(age_list)

# make age index
df_boys.set_index('age', inplace=True)
df_boys.index = df_boys.index.round(1)
df_boys.to_csv('../archive/data/height/boys_height_vs_age_combined.csv', index=True)
df_girls.set_index('age', inplace=True)
df_girls.index = df_girls.index.round(1)
df_girls.to_csv('../archive/data/height/girls_height_vs_age_combined.csv', index=True)
```

Let's take a look at what we just did.

```
df_girls
```

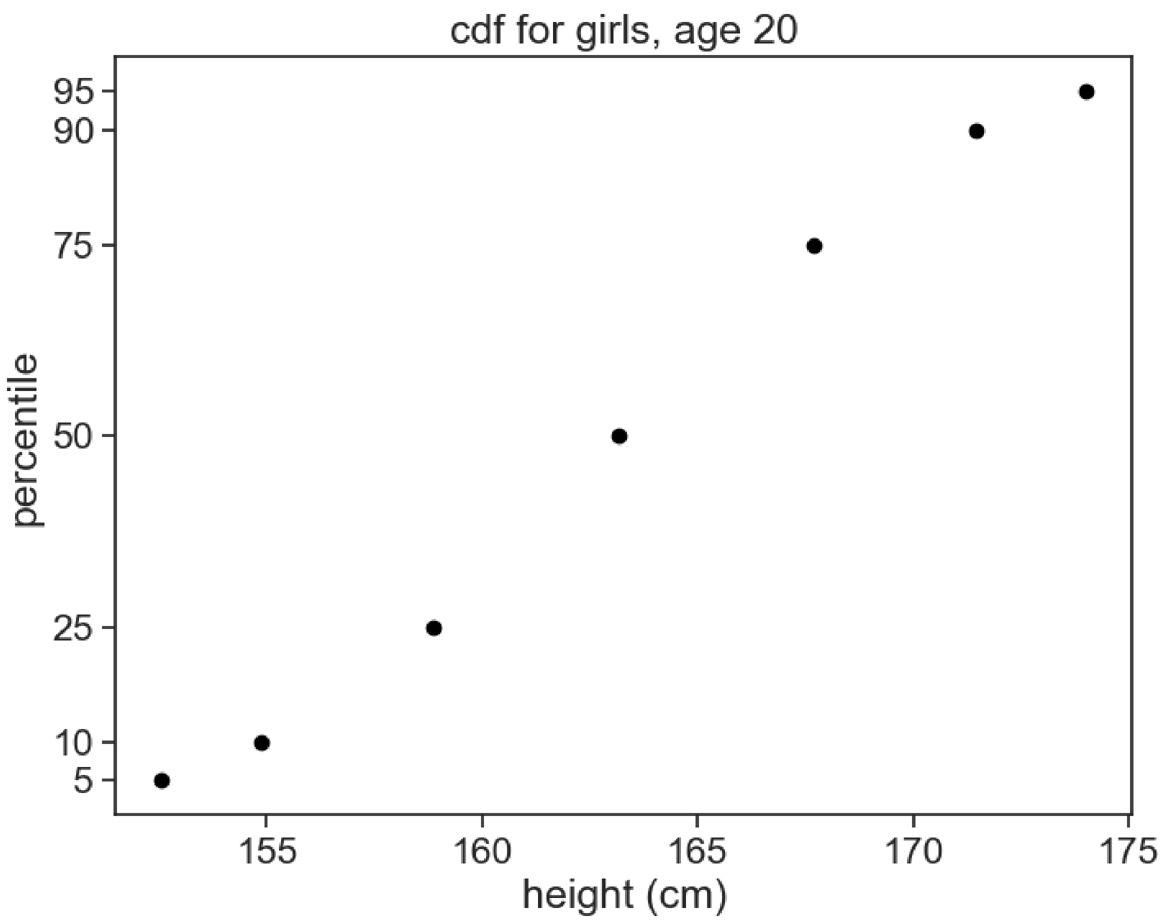
	p05	p10	p25	p50	p75	p90	p95
age							
2.0	79.269087	80.794167	83.049251	85.155597	87.475854	89.779822	90.882059
2.1	80.202106	81.772053	84.052858	86.207778	88.713405	90.883740	92.409913
2.2	81.130687	82.706754	85.011591	87.211543	89.856186	91.940642	93.416959
2.3	82.048325	83.601023	85.928399	88.170313	90.914093	92.953965	94.270653
2.4	82.948516	84.457612	86.806234	89.087509	91.897022	93.927147	95.226089
...	...	...	...	...	...	...	...
19.6	152.520938	154.812286	158.775277	163.337149	167.699533	171.531349	173.969235
19.7	152.534223	154.814440	158.791925	163.310864	167.704618	171.519600	173.980150
19.8	152.548001	154.827666	158.815071	163.275852	167.708562	171.504730	173.990964
19.9	152.562338	154.853760	158.845506	163.231563	167.711342	171.486629	174.001704
20.0	152.577300	154.894521	158.884019	163.177444	167.712936	171.465189	174.012396

```
fig, ax = plt.subplots(figsize=(8, 6))
# loop over col_names and plot each column
colors = sns.color_palette("Oranges", len(col_names))
for col, color in zip(col_names, colors):
    ax.plot(df_girls.index, df_girls[col], label=col, color=color)
ax.set(xlabel='age (years)',
       ylabel='height (cm)',
       xticks=np.arange(2, 21, 2),
       title="growth curves for girls\npercentile curves: 5, 10, 25, 50, 75, 90, 95",
       );
```



Let's now see the percentiles for girls age 20.

```
fig, ax = plt.subplots(figsize=(8, 6))
percentile_list = np.array([5, 10, 25, 50, 75, 90, 95])
data = df_girls.loc[20.0]
ax.plot(data, percentile_list, ls='', marker='o', markersize=6, color="black")
ax.set(xlabel='height (cm)',
         ylabel='percentile',
         yticks=percentile_list,
         title="cdf for girls, age 20"
         );
```



I suspect that the heights in the population are normally distributed. Let's check that. I'll fit the data to the integral of a gaussian, because the percentiles correspond to a cdf. If a pdf is a gaussian, its cumulative is given by

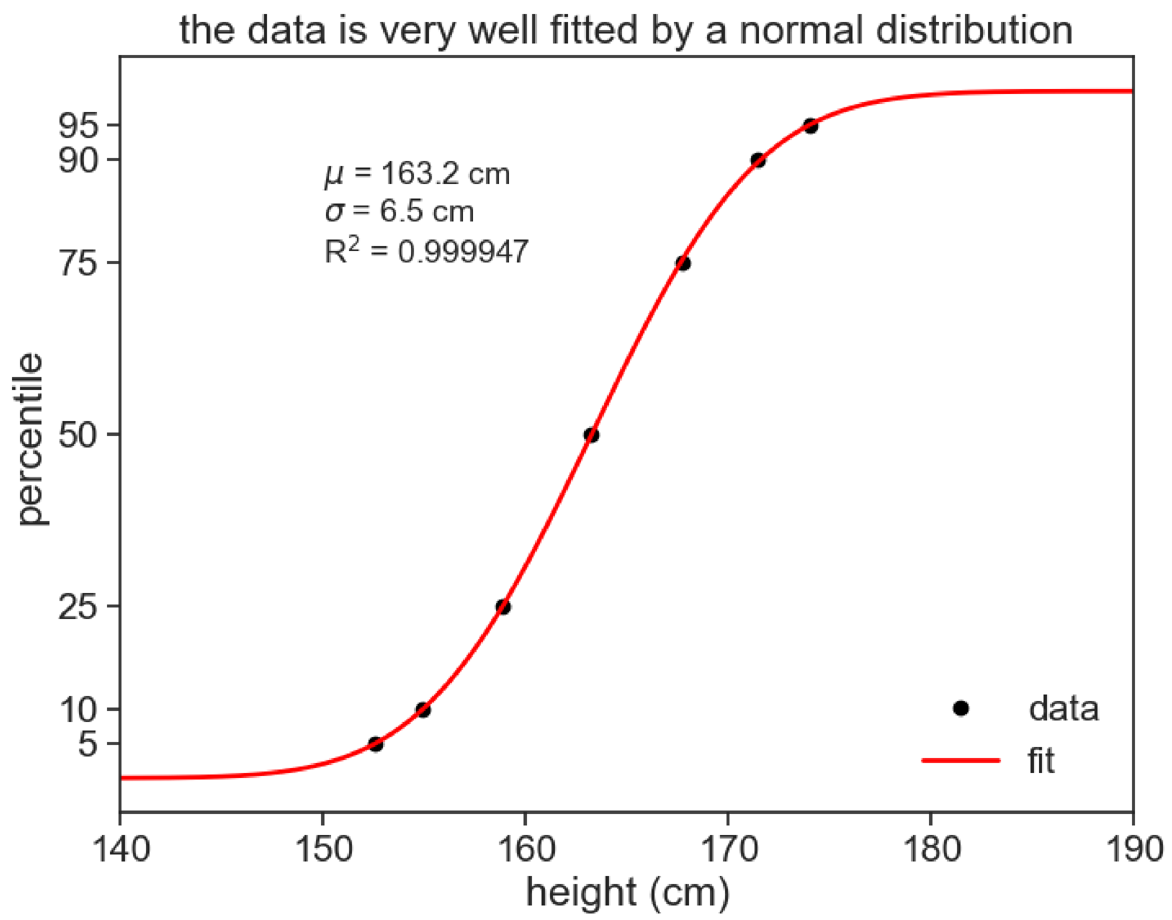
$$\Phi(x) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right)$$

where μ is the mean and σ is the standard deviation of the distribution. The error function erf is a sigmoid function, which is a good approximation for the cdf of the normal distribution.

```
def erf_model(x, mu, sigma):
    return 50 * (1 + erf((x - mu) / (sigma * np.sqrt(2))) )
# initial guess for parameters: [mu, sigma]
p0 = [150, 6]
# Calculate R-squared
def calculate_r2(y_true, y_pred):
    ss_res = np.sum((y_true - y_pred) ** 2)
    ss_tot = np.sum((y_true - np.mean(y_true)) ** 2)
    return 1 - (ss_res / ss_tot)
```

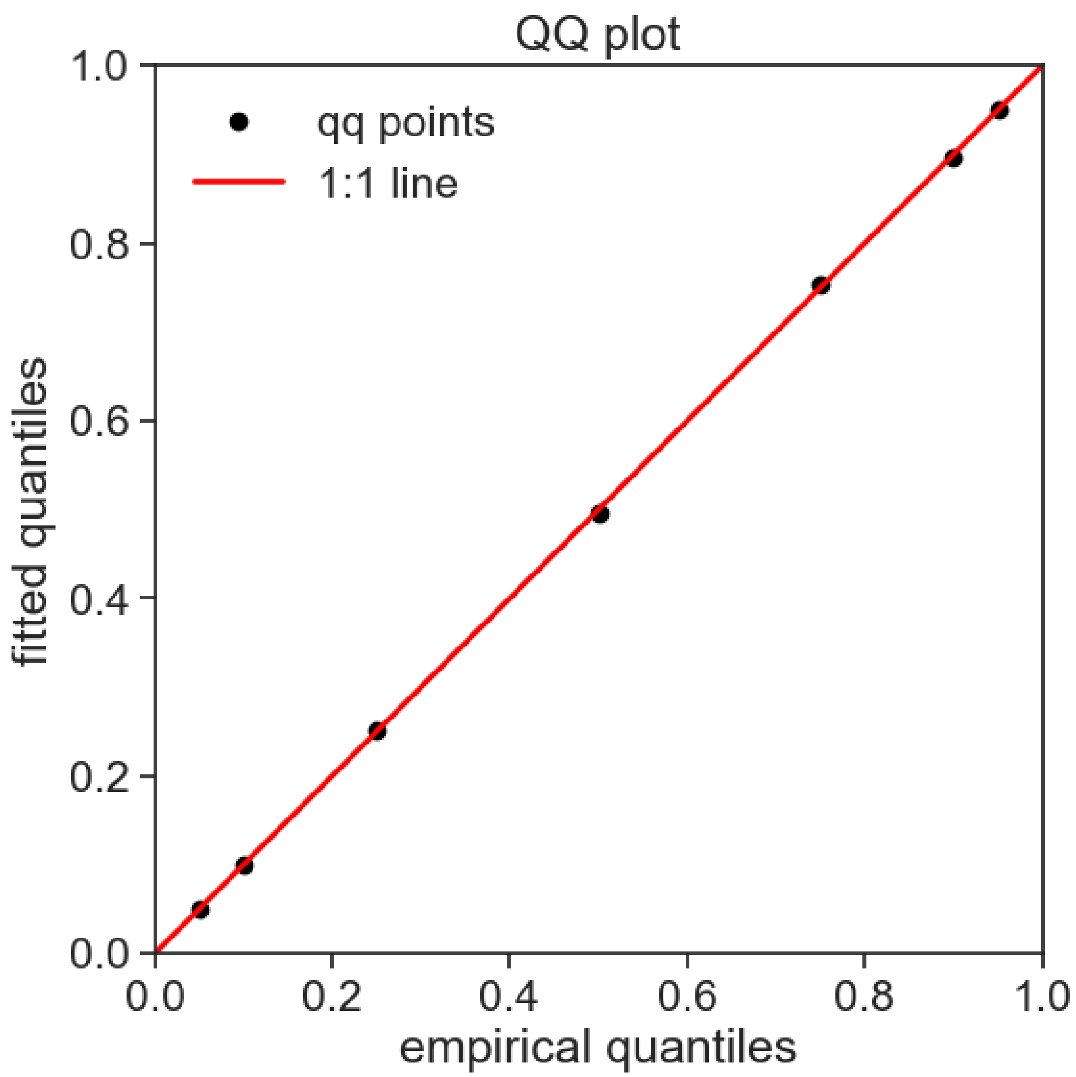
```
data = df_girls.loc[20.0]
params, _ = curve_fit(erf_model, data, percentile_list, p0=p0,
                        bounds=([100, 3],   # lower bounds for mu and sigma
                                [200, 10])  # upper bounds for mu and sigma
                        )
# store the parameters in the dataframe
percentile_predicted = erf_model(data, *params)
# R-squared value
r2 = calculate_r2(percentile_list, percentile_predicted)
```

```
fig, ax = plt.subplots(figsize=(8, 6))
percentile_list = np.array([5, 10, 25, 50, 75, 90, 95])
data = df_girls.loc[20.0]
ax.plot(data, percentile_list, ls='', marker='o', markersize=6, color="black", label='data')
fit = erf_model(height_list, *params)
ax.plot(height_list, fit, label='fit', color="red", linewidth=2)
ax.text(150, 75, f'$\mu$ = {params[0]:.1f} cm\n$\sigma$ = {params[1]:.1f} cm\nR$^2$ = {r2:.6
        fontsize=14, bbox=dict(facecolor='white', alpha=0.5))
ax.legend(frameon=False)
ax.set(xlabel='height (cm)',
       xlim=(140, 190),
         ylabel='percentile',
         yticks=percentile_list,
         title="the data is very well fitted by a normal distribution"
         );
```



Another way of making sure that the model fits the data is to make a QQ plot. In this plot, the quantiles of the data are plotted against the quantiles of the normal distribution. If the data is normally distributed, the points should fall on a straight line.

```
fitted_quantiles = norm.cdf(data, loc=params[0], scale=params[1])
experimental_quantiles = percentile_list / 100
fig, ax = plt.subplots(figsize=(8, 6))
ax.set_aspect('equal', adjustable='box')
ax.plot(experimental_quantiles, fitted_quantiles,
        ls='', marker='o', markersize=6, color="black",
        label='qq points')
ax.plot([0, 1], [0, 1], color='red', linewidth=2, label="1:1 line")
ax.set(xlabel='empirical quantiles',
       ylabel='fitted quantiles',
       xlim=(0, 1),
       ylim=(0, 1),
       title="QQ plot")
ax.legend(frameon=False)
```



Great, now we just need to do exactly the same for both sexes, and all the ages. I chose to divide age from 2 to 20 into 0.1 intervals.

```
df_stats_boys = pd.DataFrame(index=age_list, columns=['mu', 'sigma', 'r2'])
df_stats_boys['mu'] = 0.0
df_stats_boys['sigma'] = 0.0
df_stats_boys['r2'] = 0.0
df_stats_girls = pd.DataFrame(index=age_list, columns=['mu', 'sigma', 'r2'])
```

```
df_stats_girls['mu'] = 0.0
df_stats_girls['sigma'] = 0.0
df_stats_girls['r2'] = 0.0
```

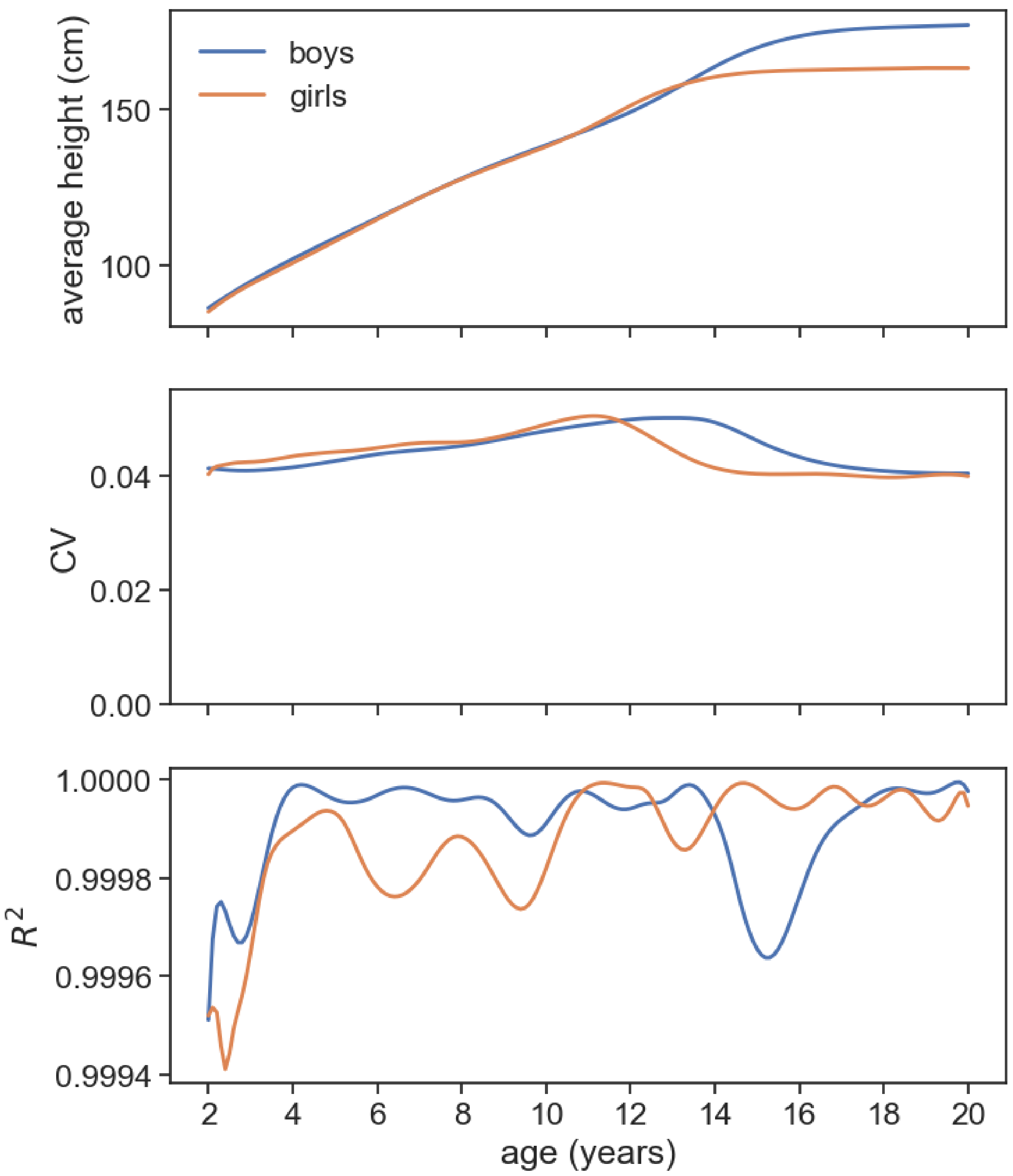
```
p0 = [80, 3]
# loop over ages in the index, calculate mu and sigma
for i in df_boys.index:
    # fit the model to the data
    data = df_boys.loc[i]
    params, _ = curve_fit(erf_model, data, percentile_list, p0=p0,
                          bounds=([70, 2],   # lower bounds for mu and sigma
                                  [200, 10])  # upper bounds for mu and sigma
                         )
    # store the parameters in the dataframe
    df_stats_boys.at[i, 'mu'] = params[0]
    df_stats_boys.at[i, 'sigma'] = params[1]
    percentile_predicted = erf_model(data, *params)
    # R-squared value
    r2 = calculate_r2(percentile_list, percentile_predicted)
    df_stats_boys.at[i, 'r2'] = r2
    p0 = params
# same for girls
p0 = [80, 3]
for i in df_girls.index:
    # fit the model to the data
    data = df_girls.loc[i]
    params, _ = curve_fit(erf_model, data, percentile_list, p0=p0,
                          bounds=([70, 3],   # lower bounds for mu and sigma
                                  [200, 10])  # upper bounds for mu and sigma
                         )
    # store the parameters in the dataframe
    df_stats_girls.at[i, 'mu'] = params[0]
    df_stats_girls.at[i, 'sigma'] = params[1]
    percentile_predicted = erf_model(data, *params)
    # R-squared value
    r2 = calculate_r2(percentile_list, percentile_predicted)
    df_stats_girls.at[i, 'r2'] = r2
    p0 = params

# save the dataframes to csv files
df_stats_boys.to_csv('../archive/data/height/boys_height_stats.csv', index=True)
df_stats_girls.to_csv('../archive/data/height/girls_height_stats.csv', index=True)
```

Let's see what we got. The top panel in the graph shows the average height for boys and girls, the middle panel shows the coefficient of variation (σ/μ), and the bottom panel shows the R2 of the fit (note that the range is very close to 1).

```
df_stats_boys
```

	mu	sigma	r2
2.0	86.463069	3.563785	0.999511
2.1	87.374895	3.596583	0.999676
2.2	88.269676	3.627433	0.999742
2.3	89.148086	3.657263	0.999752
2.4	90.010783	3.686764	0.999733
...	...	...	...
19.6	176.802810	7.134561	0.999991
19.7	176.845789	7.135786	0.999994
19.8	176.892196	7.137430	0.999995
19.9	176.942521	7.139466	0.999990
20.0	176.997255	7.141858	0.999976

```
fig, ax = plt.subplots(3,1, figsize=(8, 10), sharex=True)
fig.subplots_adjust(left=0.15)
ax[0].plot(df_stats_boys['mu'], label='boys', lw=2)
ax[0].plot(df_stats_girls['mu'], label='girls', lw=2)
ax[0].legend(frameon=False)

ax[1].plot(df_stats_boys['sigma'] / df_stats_boys['mu'], lw=2)
ax[1].plot(df_stats_girls['sigma'] / df_stats_girls['mu'], lw=2)

ax[2].plot(df_stats_boys.index, df_stats_boys['r2'], label=r'$r2$ boys', lw=2)
ax[2].plot(df_stats_girls.index, df_stats_girls['r2'], label=r'$r2$ girls', lw=2)

ax[0].set(ylabel='average height (cm)',)
ax[1].set(ylabel='CV',
          ylim=[0,0.055])
ax[2].set(xlabel='age (years)',
            ylabel=r'$R^2$',
            xticks=np.arange(2, 21, 2),
          );
```



Let's see how the pdfs for boys and girls move and morph as age increases.

```
age_list_string = age_list.astype(str).tolist()
df_pdf_boys = pd.DataFrame(index=height_list, columns=age_list_string)
```

```
df_pdf_girls = pd.DataFrame(index=height_list, columns=age_list_string)

for age in df_pdf_boys.columns:
    age_float = round(float(age), 1)
    df_pdf_boys[age] = norm.pdf(height_list,
                                loc=df_stats_boys.loc[age_float]['mu'],
                                scale=df_stats_boys.loc[age_float]['sigma'])
for age in df_pdf_girls.columns:
    age_float = round(float(age), 1)
    df_pdf_girls[age] = norm.pdf(height_list,
                                loc=df_stats_girls.loc[age_float]['mu'],
                                scale=df_stats_girls.loc[age_float]['sigma'])
```

```
df_pdf_girls
```

	2.0	2.1	2.2	2.3	2.4	2.5	2.6
70.0	0.000006	2.962419e-06	1.229580e-06	4.740717e-07	1.893495e-07	7.928033e-08	3.395629e-
70.1	0.000007	3.369929e-06	1.401926e-06	5.423176e-07	2.172465e-07	9.118694e-08	3.914667e-
70.2	0.000008	3.830459e-06	1.597215e-06	6.199308e-07	2.490751e-07	1.048086e-07	4.509972e-
70.3	0.000009	4.350475e-06	1.818328e-06	7.081296e-07	2.853621e-07	1.203810e-07	5.192270e-
70.4	0.000010	4.937172e-06	2.068480e-06	8.082806e-07	3.267014e-07	1.381707e-07	5.973725e-
...	...	...	...	...	...	...	...
219.5	0.000000	5.214425e-307	1.377605e-289	3.568527e-277	6.457994e-266	2.232144e-255	6.340272e-
219.6	0.000000	1.813597e-307	5.050074e-290	1.356408e-277	2.537010e-266	9.046507e-256	2.642444e-
219.7	0.000000	6.302763e-308	1.849870e-290	5.151948e-278	9.959447e-267	3.663840e-256	1.100546e-
219.8	0.000000	2.188653e-308	6.771033e-291	1.955386e-278	3.906942e-267	1.482823e-256	4.580523e-
219.9	0.000000	7.594139e-309	2.476504e-291	7.416066e-279	1.531537e-267	5.997065e-257	1.905138e-

```
import plotly.graph_objects as go
import plotly.io as pio

pio.renderers.default = 'notebook'

# create figure
fig = go.Figure()

# assume both dataframes have the same columns (ages) and index (height)
ages = df_pdf_boys.columns
x_vals = df_pdf_boys.index
```

```
# add traces: 2 per age (boys and girls), all hidden except the first pair
for i, age in enumerate(ages):
    fig.add_trace(go.Scatter(x=x_vals, y=df_pdf_boys[age], name=f'Boys {age}',
                             line=dict(color='#1f77b4'), visible=(i == 0)))
    fig.add_trace(go.Scatter(x=x_vals, y=df_pdf_girls[age], name=f'Girls {age}',
                             line=dict(color='#ff7f0e'), visible=(i == 0)))

# create slider steps
steps = []
for i, age in enumerate(ages):
    vis = [False] * (2 * len(ages))
    vis[2*i] = True      # boys trace
    vis[2*i + 1] = True  # girls trace

    steps.append(dict(
        method='update',
        args=[{'visible': vis},
              {'title': f'Height Distribution - Age: {age}'}],
        label=str(age)
    ))

# define slider
sliders = [dict(
    active=0,
    currentvalue={"prefix": "Age: "},
    pad={"t": 50},
    steps=steps
)]

# update layout
fig.update_layout(
    sliders=sliders,
    title='Height Distribution by Age',
    xaxis_title='Height (cm)',
    yaxis_title='Density',
    yaxis=dict(range=[0, 0.12]),
    showlegend=True,
    height=600,
    width=800
)

fig.show()
```

```
Unable to display output for mime type(s): text/html
```

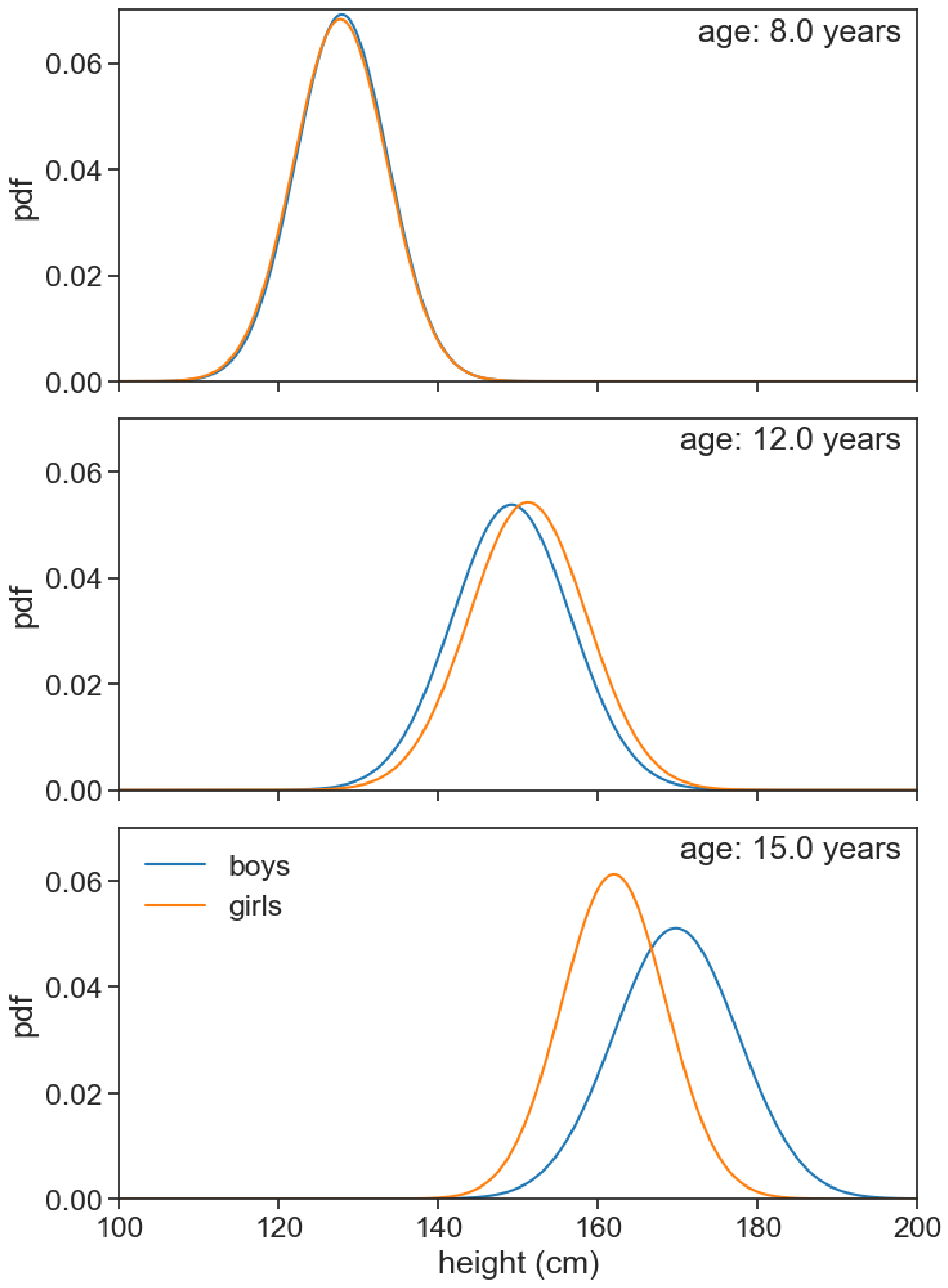
```
Unable to display output for mime type(s): text/html
```

A few notes about what we can learn from the analysis above.

- My impression that 12-year-old girls are taller than boys is indeed true.
- Boys and girls have very similar distributions up to age 11.
- From age 11 to 13 girls are on average taller than boys.
- From age 13 boys become taller than girls, on average.
- The graph showing the coefficient of variation is interesting. CV for girls peaks roughtly at age 12, and for boys it peaks around age 14. These local maxima may be explained by the wide variability in the age ofpuberty onset.
- The height distribution for each sex, across all ages, is indeed extremely well described by the normal distribution. What biological factors may account for such a fact?

I'll plot one last graph from now, let's see what we can learn from it. Let's see the pdf for boys and girls across three age groups: 8, 12, and 15 year olds.

```python
fig, ax = plt.subplots(3, 1, figsize=(8, 12), sharex=True)
fig.subplots_adjust(hspace=0.1)
ages_to_plot = [8.0, 12.0, 15.0]

for i, age in enumerate(ages_to_plot):
    pdf_boys = norm.pdf(height_list, loc=df_stats_boys.loc[age]['mu'], scale=df_stats_boys.lo
    pdf_girls = norm.pdf(height_list, loc=df_stats_girls.loc[age]['mu'], scale=df_stats_girls
    ax[i].plot(height_list, pdf_boys, label='boys', color='tab:blue')
    ax[i].plot(height_list, pdf_girls, label='girls', color='tab:orange')
    ax[i].text(0.98, 0.98, f'age: {age} years', transform=ax[i].transAxes, verticalalignment=
    ax[i].set(ylabel='pdf',
              ylim=(0, 0.07),
             )
ax[2].legend(frameon=False)
ax[2].set(xlabel='height (cm)',
          xlim=(100, 200),);
```



age: 8.0 years
age: 12.0 years
age: 15.0 years
boys
girls
pdf
0.00
0.02
0.04
0.06
100
120
140
160
180
200
height (cm)

- Indeed, boys and girls age 8 have the exact same height distribution.
- 12-year-old girls are indeed taller than boys, on average. This difference is relatiely small, though.
- By age 15 boys have long surpassed girls in height, and the difference is quite large. Boys still have some growing to do, but girls are mostly done growing.

2 weight data

Now that we have height data covered, it's time we deal with weight data.

Yes, I am **VERY WELL AWARE** that weight is a force, and it is not measured in kg. Nevertheless, I will use the word weight in the colloquial sense, and for all purposes it is a synonym for mass.

This analysis will follow the same steps we used for height data, therefore I will skip some of the intermediate steps. Whenever there are differences, I will point them out.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style="ticks", font_scale=1.5)
from scipy.optimize import curve_fit
from scipy.special import erf
from scipy.interpolate import UnivariateSpline
import matplotlib.animation as animation
from scipy.stats import norm
from scipy.stats import lognorm
import plotly.graph_objects as go
import plotly.io as pio
pio.renderers.default = 'notebook'
%matplotlib widget
```

```
age_list = np.round(np.arange(2.0, 20.1, 0.1), 1)
weight_list = np.round(np.arange(10, 100, 0.1), 1)
```

We will load all the digitized data for the weight curves, and will interpolate.

```
col_names = ['p05', 'p10', 'p25', 'p50', 'p75', 'p90', 'p95']
file_names_boys = ['boys-p05.csv', 'boys-p10.csv', 'boys-p25.csv', 'boys-p50.csv',
                   'boys-p75.csv', 'boys-p90.csv', 'boys-p95.csv',]
file_names_girls = ['girls-p05.csv', 'girls-p10.csv', 'girls-p25.csv', 'girls-p50.csv',
                    'girls-p75.csv', 'girls-p90.csv', 'girls-p95.csv',]
```

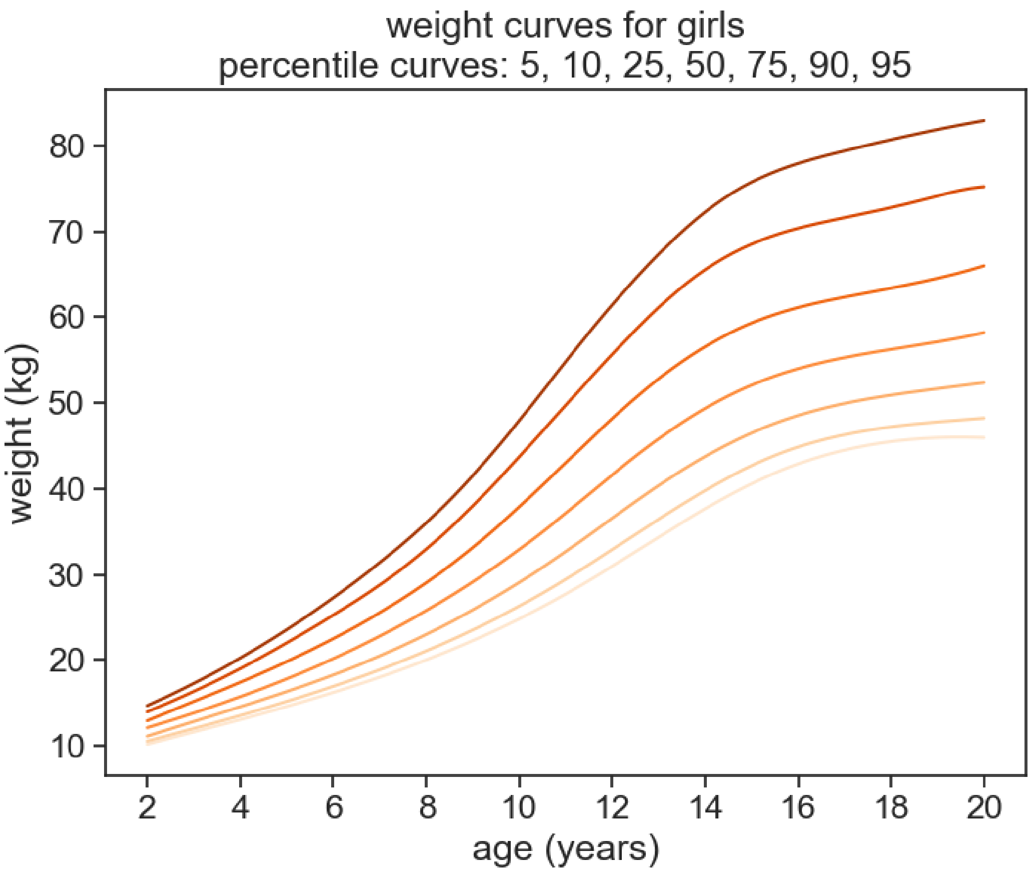
```
# create dataframe with age column
df_boys = pd.DataFrame({'age': age_list})
df_girls = pd.DataFrame({'age': age_list})
# loop over file names and read in data
for i, file_name in enumerate(file_names_boys):
    # read in data
    df_temp = pd.read_csv('../archive/data/weight/' + file_name, names=['age','weight'])
    spline = UnivariateSpline(df_temp['age'], df_temp['weight'], s=0.5)
    df_boys[col_names[i]] = spline(age_list)
for i, file_name in enumerate(file_names_girls):
    # read in data
    df_temp = pd.read_csv('../archive/data/weight/' + file_name, names=['age','weight'])
    spline = UnivariateSpline(df_temp['age'], df_temp['weight'], s=0.5)
    df_girls[col_names[i]] = spline(age_list)

# make age index
df_boys.set_index('age', inplace=True)
df_boys.index = df_boys.index.round(1)
df_boys.to_csv('../archive/data/weight/boys_weight_vs_age_combined.csv', index=True)
df_girls.set_index('age', inplace=True)
df_girls.index = df_girls.index.round(1)
df_girls.to_csv('../archive/data/weight/girls_weight_vs_age_combined.csv', index=True)
```

Let's see now the results:

```
df_girls
```

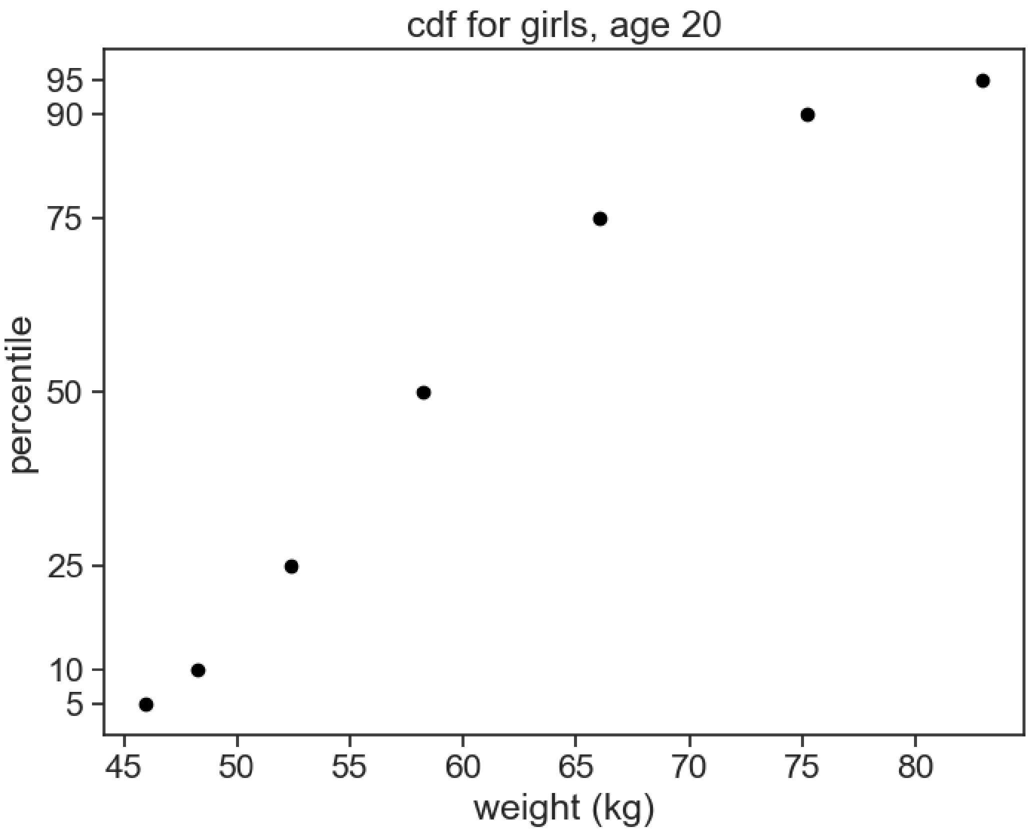
	p05	p10	p25	p50	p75	p90	p95
age							
2.0	10.169139	10.527385	11.117607	12.111512	12.934594	13.970137	14.662625
2.1	10.312243	10.674421	11.291813	12.276811	13.155592	14.191386	14.907424
2.2	10.455176	10.821547	11.465113	12.443392	13.376209	14.415799	15.155615
2.3	10.597984	10.968806	11.637596	12.611299	13.596546	14.643380	15.407214
2.4	10.740711	11.116243	11.809350	12.780573	13.816707	14.874137	15.662236
…	…	…	…	…	…	…	…
19.6	46.015792	48.041956	52.097485	57.744985	65.357860	74.916720	82.545975
19.7	46.008755	48.085732	52.168559	57.855278	65.511084	75.011896	82.648322
19.8	45.997560	48.129771	52.240501	57.969161	65.669232	75.094395	82.748961
19.9	45.982282	48.174284	52.313500	58.086892	65.832528	75.162729	82.847851
20.0	45.962995	48.219480	52.387740	58.208729	66.001197	75.215412	82.944949

```
fig, ax = plt.subplots(figsize=(8, 6))
# loop over col_names and plot each column
colors = sns.color_palette("Oranges", len(col_names))
for col, color in zip(col_names, colors):
    ax.plot(df_girls.index, df_girls[col], label=col, color=color)
ax.set(xlabel='age (years)',
       ylabel='weight (kg)',
       xticks=np.arange(2, 21, 2),
       title="weight curves for girls\npercentile curves: 5, 10, 25, 50, 75, 90, 95",
       );
```



Let's now see the percentiles for girls age 20.

```
fig, ax = plt.subplots(figsize=(8, 6))
percentile_list = np.array([5, 10, 25, 50, 75, 90, 95])
data = df_girls.loc[20.0]
ax.plot(data, percentile_list, ls='', marker='o', markersize=6, color="black")
ax.set(xlabel='weight (kg)',
         ylabel='percentile',
         yticks=percentile_list,
         title="cdf for girls, age 20"
         );
```



This time, I am not so sure that a normal (gaussian) distribution is a good fit for the data. I check if a log-normal distribution does the job.

```
def erf_model(x, mu, sigma):
    return 50 * (1 + erf((x - mu) / (sigma * np.sqrt(2))) )
def erf_lognormal_model(x, mu, sigma):
    return 50 * (1 + erf((np.log(x) - mu) / (sigma * np.sqrt(2))) )
def cdf_lognormal(x, mu, sigma):
    return 100*lognorm.cdf(x, s=sigma, scale=mu)
def cdf_gamma(x, a, b):
    return 100*gamma.cdf(x, a=a, scale=b)
```

```
# initial guess for parameters: [mu, sigma]
p0 = np.array([50, 3])
p0_ln = np.array([50, 0.2])
# Calculate R-squared
def calculate_r2(y_true, y_pred):
    ss_res = np.sum((y_true - y_pred) ** 2)
    ss_tot = np.sum((y_true - np.mean(y_true)) ** 2)
    return 1 - (ss_res / ss_tot)
```

```
data = df_girls.loc[20.0]
params, _ = curve_fit(erf_model, data, percentile_list, p0=p0,
                        bounds=([10, 1],   # lower bounds for mu and sigma
                                [100, 20])  # upper bounds for mu and sigma
                        )
params2, _ = curve_fit(erf_model, np.log(data), percentile_list, p0=np.log(p0),
                        bounds=([np.log(10), 0],   # lower bounds for mu and sigma
                                [np.log(100), np.log(20)])  # upper bounds for mu and sigma
                        )
params3, _ = curve_fit(cdf_lognormal, data, percentile_list, p0=p0_ln,
                        bounds=([10, 0.05],   # lower bounds for mu and sigma
                                [100, 1.00])  # upper bounds for mu and sigma
                        )
p0_gamma = np.array([50, 0.9])
params4, _ = curve_fit(cdf_gamma, data, percentile_list, p0=p0_gamma,
                        bounds=([30, 0.5],   # lower bounds for mu and sigma
                                [80, 1.5])  # upper bounds for mu and sigma
                        )
# store the parameters in the dataframe
percentile_predicted = erf_model(data, *params)
percentile_predicted2 = erf_model(np.log(data), *params2)
percentile_predicted3 = cdf_lognormal(data, *params3)
percentile_predicted4 = cdf_gamma(data, *params4)


# R-squared value
r2 = calculate_r2(percentile_list, percentile_predicted)
r2b = calculate_r2(percentile_list, percentile_predicted2)
r2c = calculate_r2(percentile_list, percentile_predicted3)
r2d = calculate_r2(percentile_list, percentile_predicted4)
```

```
params4, r2d
```

```
(array([38.66428713,  1.46253451]), 0.9924311256135725)
```

```
params3, r2c, percentile_predicted3
```

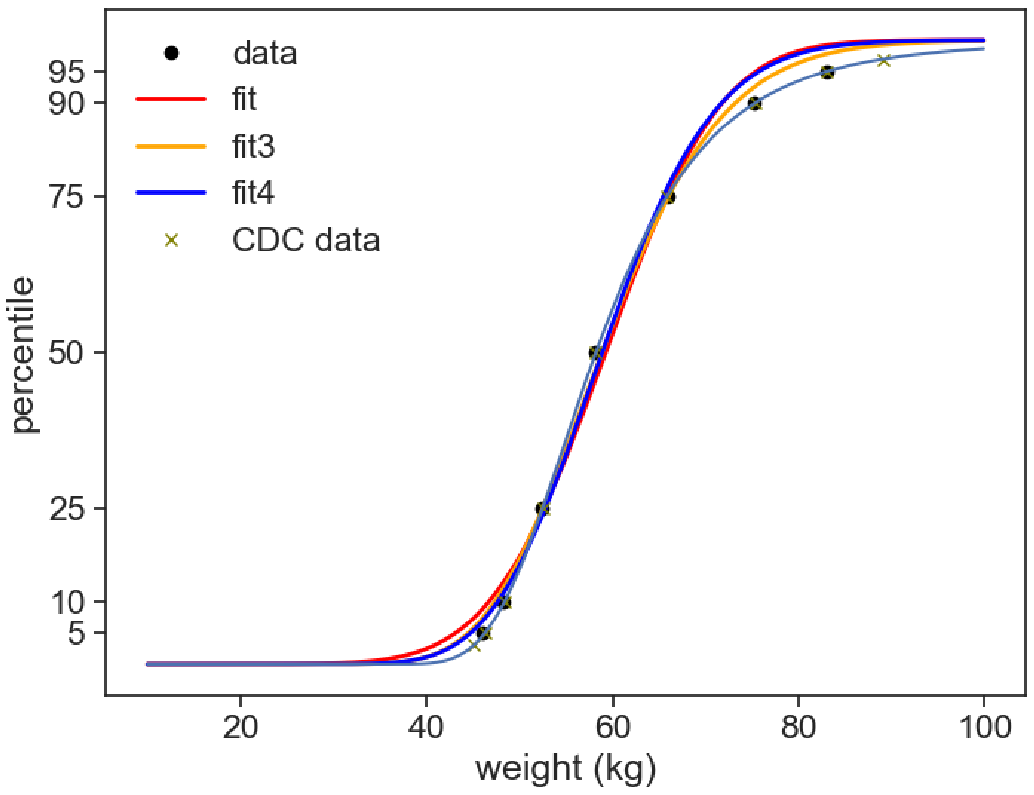
```
(array([55.95300497,  0.16283376]),
 0.9944470433169026,
 array([ 7.97666653, 12.25680131, 24.25581653, 47.087817  , 74.72185354,
        93.52040652, 98.42654193]))
```

```
from scipy.stats import gamma
```

```
percentile_list
```

```
array([ 5, 10, 25, 50, 75, 90, 95])
```

```
fig, ax = plt.subplots(figsize=(8, 6))
percentile_list = np.array([5, 10, 25, 50, 75, 90, 95])
ax.plot(data, percentile_list, ls='', marker='o', markersize=6, color="black", label='data')
fit = erf_model(weight_list, *params)
# fit2 = np.exp(erf_model(np.log(weight_list), *(np.log(params))))
ax.plot(weight_list, fit, label='fit', color="red", linewidth=2)
# ax.plot(weight_list, 100*lognorm.cdf(weight_list, loc=np.exp(params2[0]), s=1*np.exp(params
ax.plot(weight_list, 100*lognorm.cdf(weight_list, scale=params3[0], s=params3[1] ), label='fi
ax.plot(weight_list, 100*gamma.cdf(weight_list, a=params4[0], scale=params4[1] ), label='fit4

# ax.plot(weight_list, 100*lognorm.cdf(weight_list, scale=58, s=0.18 ), label='fit2', color='
# ax.text(150, 75, f'$\mu$ = {params[0]:.1f} cm\n$\sigma$ = {params[1]:.1f} cm\nR$^2$ = {r2:
        # fontsize=14, bbox=dict(facecolor='white', alpha=0.5))




# ax.plot(weight_list, 100*gamma.cdf(weight_list, a=68, scale=0.8), label='gamma', color="blu
cdc_percentile = [3, 5, 10, 25, 50, 75, 90, 95, 97]
ax.plot(cdc_vals[0], cdc_percentile, ls='', marker='x', markersize=6, color="olive", label='C
lms = women.loc[240.0, ['L', 'M', 'S']].values
ax.plot(weight_list, Zscore_to_percentile(cdc_eq(weight_list, *lms)))
ax.legend(frameon=False)
ax.set(xlabel='weight (kg)',
    #    xlim=(140, 190),
         ylabel='percentile',
         yticks=percentile_list,
        #  title="the data is very well fitted by a normal distribution"
         );
```



```
https://www.cdc.gov/growthcharts/cdc-data-files.htm
```

```
SyntaxError: invalid syntax (1102385488.py, line 1)
```

```
def Zscore_to_percentile(z):
    return 100 * norm.cdf(z)
```

```python
cdc_vals[0], percentile_list
```

```
(array([45.04654822, 46.28963394, 48.38346004, 52.47876433, 58.21897289,
        65.85237979, 75.35164989, 82.95375457, 89.04485133]),
 array([ 5, 10, 25, 50, 75, 90, 95]))
```

```python
df_cdc = pd.read_csv('../archive/data/cdc_wt_age.csv')
df_cdc
```

	Sex	Agemos	L	M	S	P3	P5	P10	P25	P50
0	1	24.0	-0.206152	12.670763	0.108126	10.382090	10.640090	11.052656	11.785975	12.6
1	1	24.5	-0.216501	12.741544	0.108166	10.441442	10.700513	11.114904	11.851817	12.7
2	1	25.5	-0.239790	12.881023	0.108275	10.558473	10.819575	11.237473	11.981419	12.8
3	1	26.5	-0.266316	13.018424	0.108421	10.673803	10.936812	11.358059	12.108888	13.0
4	1	27.5	-0.295755	13.154497	0.108605	10.787982	11.052801	11.477280	12.234907	13.1
...	...	...	...	...	...	...	...	...	...	...
431	2	236.5	-1.558179	58.030397	0.165698	45.012603	46.238515	48.304379	52.349208	58.0
432	2	237.5	-1.543846	58.094532	0.165985	45.027517	46.258911	48.333695	52.394603	58.0
433	2	238.5	-1.530642	58.151036	0.166260	45.038521	46.274976	48.357997	52.433762	58.1
434	2	239.5	-1.518754	58.198771	0.166520	45.045100	46.286116	48.376568	52.465761	58.1
435	2	240.0	-1.513362	58.218973	0.166645	45.046548	46.289634	48.383460	52.478764	58.2

```python
def cdc_eq(age, L, M, S):
    if L != 0:
        return ((age/M)**L -1) / (L*S)
    else:
        return np.log(age/M) / S
```

```python
# women.set_index('Agemos', inplace=True)
```

```
array([-1.51336185, 58.21897289,  0.16664475])
```

```python
women = df_cdc[df_cdc['Sex']==2]
women240 = women[women['Agemos']==240.0]
# women240
women240
```

	Sex	Agemos	L	M	S	P3	P5	P10	P25	P50
435	2	240.0	-1.513362	58.218973	0.166645	45.046548	46.289634	48.38346	52.478764	58.21

```
cols = women240.columns[5:]
cdc_vals = women240.loc[:, cols].values
```

```
df_cdc
```

	Sex	Agemos	L	M	S	P3	P5	P10	P25	P50
0	1	24.0	-0.206152	12.670763	0.108126	10.382090	10.640090	11.052656	11.785975	12.6
1	1	24.5	-0.216501	12.741544	0.108166	10.441442	10.700513	11.114904	11.851817	12.7
2	1	25.5	-0.239790	12.881023	0.108275	10.558473	10.819575	11.237473	11.981419	12.8
3	1	26.5	-0.266316	13.018424	0.108421	10.673803	10.936812	11.358059	12.108888	13.0
4	1	27.5	-0.295755	13.154497	0.108605	10.787982	11.052801	11.477280	12.234907	13.1
...	...	...	...	...	...	...	...	...	...	...
431	2	236.5	-1.558179	58.030397	0.165698	45.012603	46.238515	48.304379	52.349208	58.0
432	2	237.5	-1.543846	58.094532	0.165985	45.027517	46.258911	48.333695	52.394603	58.0
433	2	238.5	-1.530642	58.151036	0.166260	45.038521	46.274976	48.357997	52.433762	58.1
434	2	239.5	-1.518754	58.198771	0.166520	45.045100	46.286116	48.376568	52.465761	58.1
435	2	240.0	-1.513362	58.218973	0.166645	45.046548	46.289634	48.383460	52.478764	58.2

Part II

quick questions

3 one-sample t test

3.1 Question

I measured the height of 10 adult men. Were they sampled from the general population of men?

3.2 Hypotheses

- Null hypothesis: The sample mean is equal to the population mean. In this case, the answer would be "yes"
- Alternative hypothesis: The sample mean is not equal to the population mean. Answer would be "no".
- Significance level: 0.05

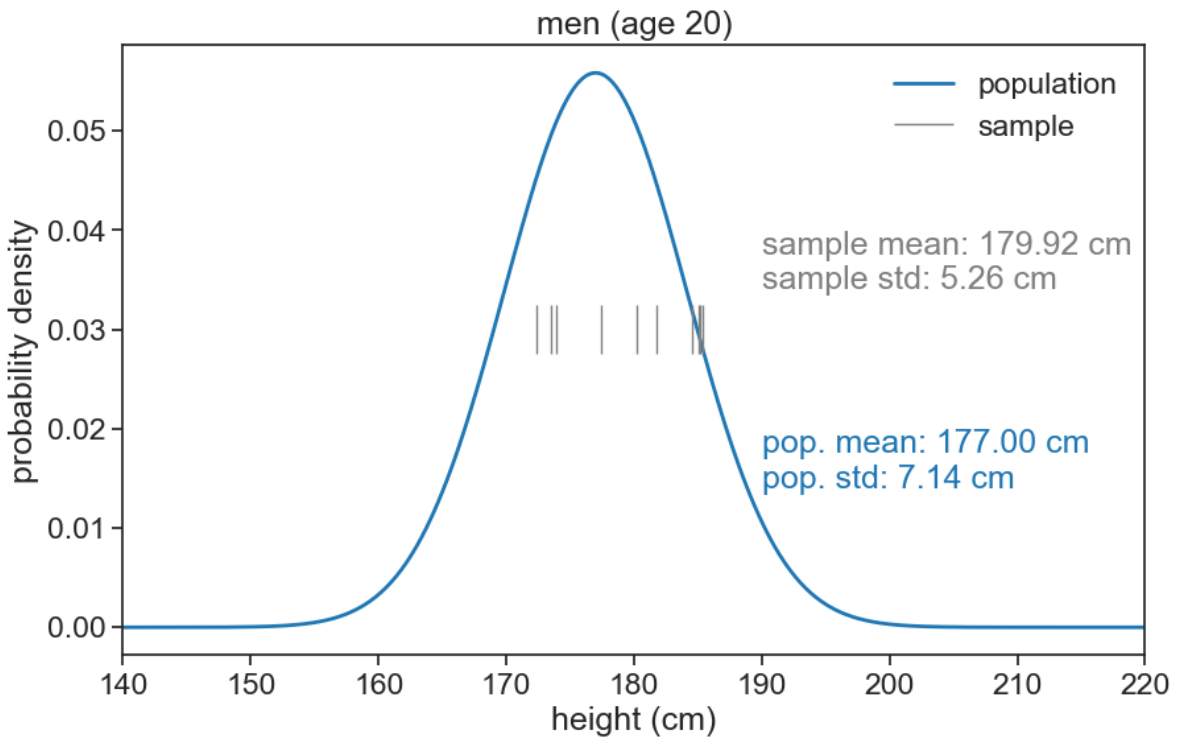
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style="ticks", font_scale=1.5)
from scipy.stats import norm, ttest_1samp, t
# %matplotlib widget
```

```
df_boys = pd.read_csv('../archive/data/height/boys_height_stats.csv', index_col=0)
mu_boys = df_boys.loc[20.0, 'mu']
sigma_boys = df_boys.loc[20.0, 'sigma']
```

Let's start with a sample of 10.

```
N = 10
# set scipy seed for reproducibility
np.random.seed(314)
sample10 = norm.rvs(size=N, loc=mu_boys+2, scale=sigma_boys)
```

```python
height_list = np.arange(140, 220, 0.1)
pdf_boys = norm.pdf(height_list, loc=mu_boys, scale=sigma_boys)

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(height_list, pdf_boys, lw=2, color='tab:blue', label='population')

ax.eventplot(sample10, orientation="horizontal", lineoffsets=0.03,
             linewidth=1, linelengths= 0.005,
             colors='gray', label='sample')

ax.text(190, 0.04,
       f"sample mean: {sample10.mean():.2f} cm\nsample std: {sample10.std(ddof=1):.2f} cm",
       ha='left', va='top', color='gray')

ax.text(190, 0.02,
       f"pop. mean: {mu_boys:.2f} cm\npop. std: {sigma_boys:.2f} cm",
       ha='left', va='top', color='tab:blue')

ax.legend(frameon=False)
ax.set(xlabel='height (cm)',
       ylabel='probability density',
       title="men (age 20)",
       xlim=(140, 220),
       );
```



The t value is calculated as follows:

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}}$$

where

- $\bar{x}$: sample mean
- μ: population mean
- s: sample standard deviation
- n: sample size

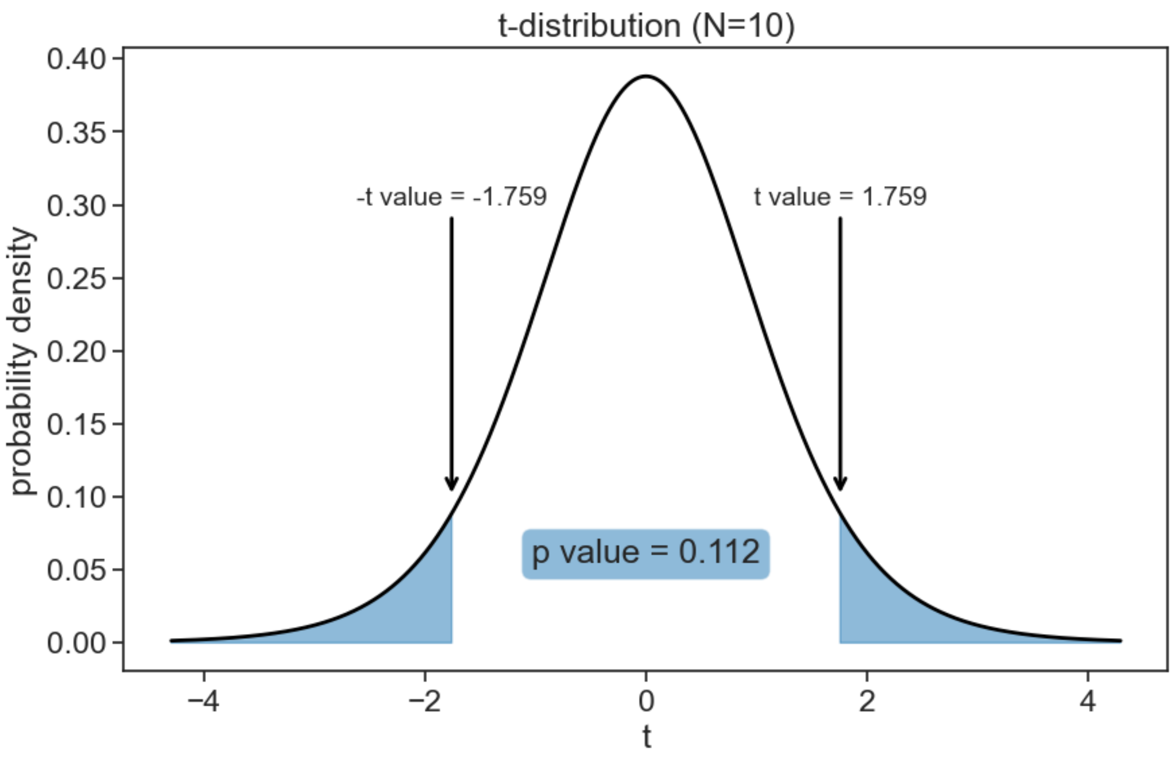
Let's try the formula above and compare it with scipy's ttest_1samp function.

```
t_value_formula = (sample10.mean() - mu_boys) / (sample10.std(ddof=1) / np.sqrt(N))
t_value_scipy = ttest_1samp(sample10, popmean=mu_boys)
print(f"t-value (formula): {t_value_formula:.3f}")
print(f"t-value (scipy): {t_value_scipy.statistic:.3f}")
```

```
t-value (formula): 1.759
t-value (scipy): 1.759
```

Let's convert this t value to a p value. It is easy to visualize the p value by ploting the pdf for the t distribution. The p value is the area under the curve for t greater than the t value and smaller than the negative t value.

```python
# degrees of freedom
dof = N - 1
fig, ax = plt.subplots(figsize=(10, 6))

t_array_min = np.round(t.ppf(0.001, dof),3)
t_array_max = np.round(t.ppf(0.999, dof),3)
t_array = np.arange(t_array_min, t_array_max, 0.001)

# annotate vertical array at t_value_scipy
ax.annotate(f"t value = {t_value_scipy.statistic:.3f}",
                        xy=(t_value_scipy.statistic, 0.10),
                        xytext=(t_value_scipy.statistic, 0.30),
                        fontsize=14,
                        arrowprops=dict(arrowstyle="->", lw=2, color='black'),
                        ha='center')
ax.annotate(f"-t value = -{t_value_scipy.statistic:.3f}",
                        xy=(-t_value_scipy.statistic, 0.10),
                        xytext=(-t_value_scipy.statistic, 0.30),
                        fontsize=14,
                        arrowprops=dict(arrowstyle="->", lw=2, color='black'),
                        ha='center')
# fill between t-distribution and normal distribution
ax.fill_between(t_array, t.pdf(t_array, dof),
                 where=(np.abs(t_array) > t_value_scipy.statistic),
                 color='tab:blue', alpha=0.5,
                 label='rejection region')

# write t_value_scipy.pvalue on the plot
ax.text(0, 0.05,
        f"p value = {t_value_scipy.pvalue:.3f}",
        ha='center', va='bottom',
        bbox=dict(facecolor='tab:blue', alpha=0.5, boxstyle="round"))

ax.plot(t_array, t.pdf(t_array, dof),
       color='black', lw=2)

ax.set(xlabel='t',
       ylabel='probability density',
       title="t-distribution (N=10)",
        );
```



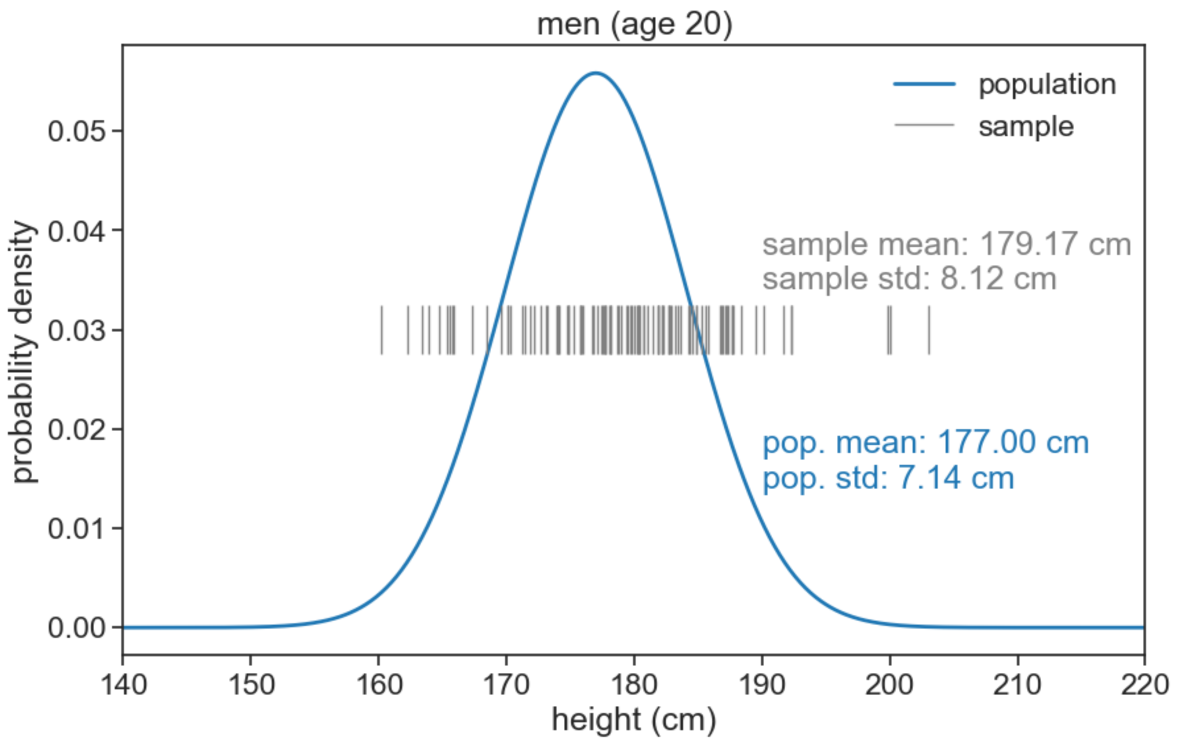
The p value is the fraction of the t distribution that is more extreme than the observed t value. If the p value is less than the significance level, we reject the null hypothesis. In this case, the p value is larger than the significance level, so we fail to reject the null hypothesis. This means that we do not have enough evidence to say that the sample mean is different from the population mean. In other words, we cannot conclude that the 10 men samples were drawn from a distribution different than the general population.

3.3 increase the sample size

Let's see what happens when we increase the sample size to 100.

```
N = 100
# set scipy seed for reproducibility
np.random.seed(628)
sample100 = norm.rvs(size=N, loc=mu_boys+2, scale=sigma_boys)
```

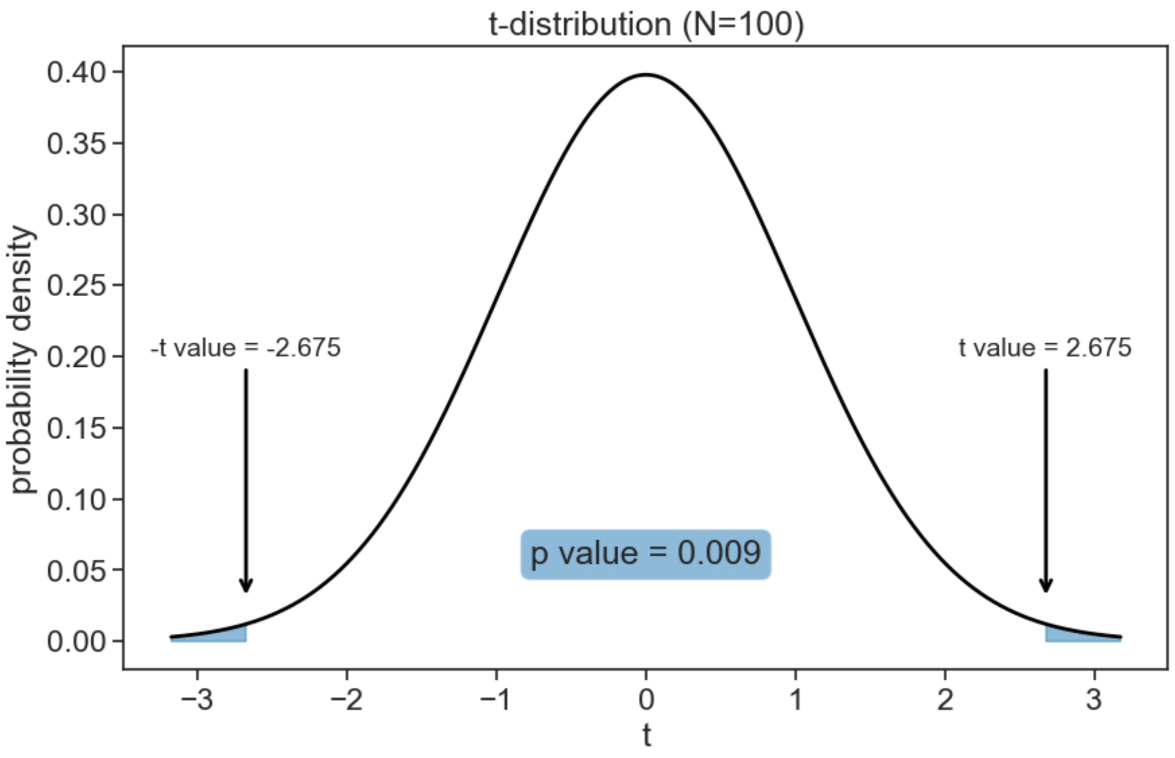
```
height_list = np.arange(140, 220, 0.1)
pdf_boys = norm.pdf(height_list, loc=mu_boys, scale=sigma_boys)

fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(height_list, pdf_boys, lw=2, color='tab:blue', label='population')

ax.eventplot(sample100, orientation="horizontal", lineoffsets=0.03,
             linewidth=1, linelengths= 0.005,
             colors='gray', label='sample')

ax.text(190, 0.04,
       f"sample mean: {sample100.mean():.2f} cm\nsample std: {sample100.std(ddof=1):.2f} cm"
       ha='left', va='top', color='gray')

ax.text(190, 0.02,
       f"pop. mean: {mu_boys:.2f} cm\npop. std: {sigma_boys:.2f} cm",
       ha='left', va='top', color='tab:blue')

ax.legend(frameon=False)
ax.set(xlabel='height (cm)',
       ylabel='probability density',
       title="men (age 20)",
       xlim=(140, 220),
       );
```



```
t_value_scipy = ttest_1samp(sample100, popmean=mu_boys)
print(f"t-value: {t_value_scipy.statistic:.3f}")
print(f"p-value: {t_value_scipy.pvalue:.3f}")
```

```
t-value: 2.675
p-value: 0.009
```

```
# degrees of freedom
dof = N - 1
fig, ax = plt.subplots(figsize=(10, 6))

t_array_min = np.round(t.ppf(0.001, dof),3)
t_array_max = np.round(t.ppf(0.999, dof),3)
t_array = np.arange(t_array_min, t_array_max, 0.001)

# annotate vertical array at t_value_scipy
ax.annotate(f"t value = {t_value_scipy.statistic:.3f}",
                        xy=(t_value_scipy.statistic, 0.03),
                        xytext=(t_value_scipy.statistic, 0.20),
                        fontsize=14,
                        arrowprops=dict(arrowstyle="->", lw=2, color='black'),
                        ha='center')
ax.annotate(f"-t value = -{t_value_scipy.statistic:.3f}",
                        xy=(-t_value_scipy.statistic, 0.03),
                        xytext=(-t_value_scipy.statistic, 0.20),
                        fontsize=14,
                        arrowprops=dict(arrowstyle="->", lw=2, color='black'),
                        ha='center')
# fill between t-distribution and normal distribution
ax.fill_between(t_array, t.pdf(t_array, dof),
                 where=(np.abs(t_array) > t_value_scipy.statistic),
                 color='tab:blue', alpha=0.5,
                 label='rejection region')

# write t_value_scipy.pvalue on the plot
ax.text(0, 0.05,
        f"p value = {t_value_scipy.pvalue:.3f}",
        ha='center', va='bottom',
        bbox=dict(facecolor='tab:blue', alpha=0.5, boxstyle="round"))

ax.plot(t_array, t.pdf(t_array, dof),
       color='black', lw=2)

ax.set(xlabel='t',
       ylabel='probability density',
       title="t-distribution (N=100)",
       );
```



3.4 Question 2

Can we say that the sampled men are taller than the general population?

3.5 Hypotheses

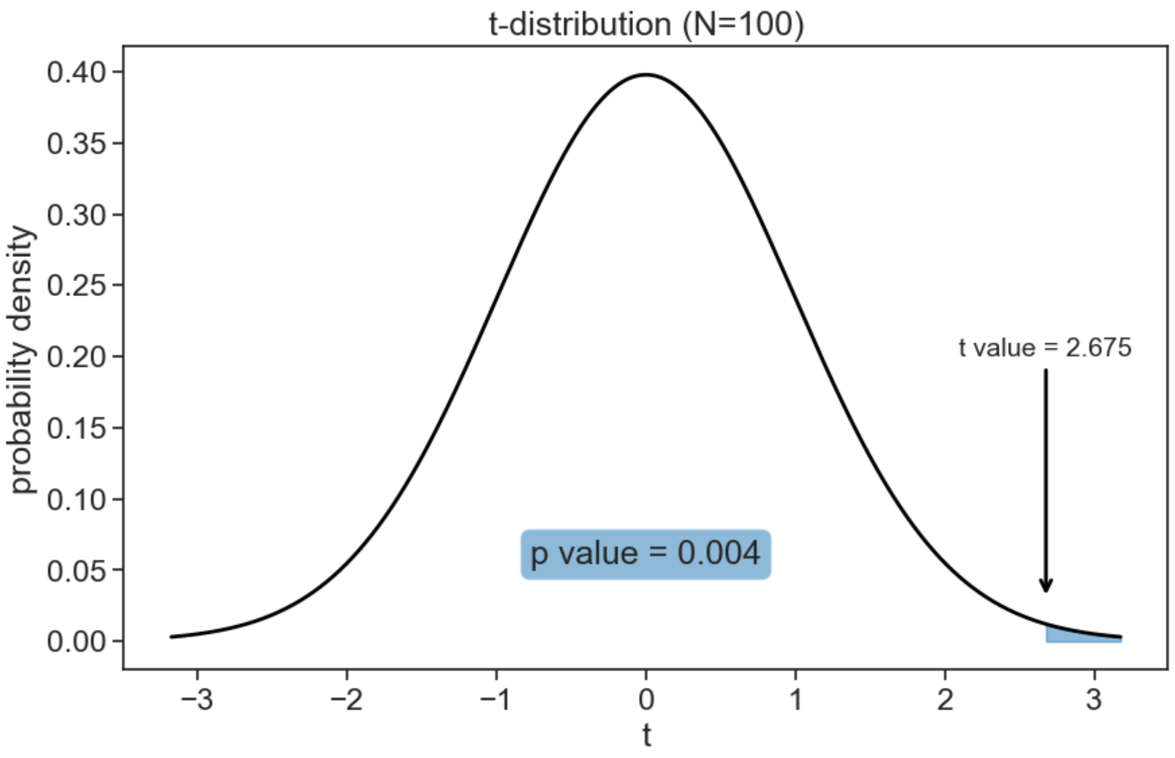
- Null hypothesis: The sample mean is equal to the population mean.
- Alternative hypothesis: The sample mean is higher the population mean.
- Significance level: 0.05

The analysis is the same as before, but we will use a one-tailed test. The t statistic is the same, but the p value is smaller, since we account for a smaller portion of the total area of the pdf.

```
t_value_scipy = ttest_1samp(sample100, popmean=mu_boys, alternative='greater')
print(f"t-value: {t_value_scipy.statistic:.3f}")
print(f"p-value: {t_value_scipy.pvalue:.3f}")
```

```
t-value: 2.675
p-value: 0.004
```

```python
# degrees of freedom
dof = N - 1
fig, ax = plt.subplots(figsize=(10, 6))

t_array_min = np.round(t.ppf(0.001, dof),3)
t_array_max = np.round(t.ppf(0.999, dof),3)
t_array = np.arange(t_array_min, t_array_max, 0.001)

# annotate vertical array at t_value_scipy
ax.annotate(f"t value = {t_value_scipy.statistic:.3f}",
                        xy=(t_value_scipy.statistic, 0.03),
                        xytext=(t_value_scipy.statistic, 0.20),
                        fontsize=14,
                        arrowprops=dict(arrowstyle="->", lw=2, color='black'),
                        ha='center')
# fill between t-distribution and normal distribution
ax.fill_between(t_array, t.pdf(t_array, dof),
                 where=(t_array > t_value_scipy.statistic),
                 color='tab:blue', alpha=0.5,
                 label='rejection region')

# write t_value_scipy.pvalue on the plot
ax.text(0, 0.05,
        f"p value = {t_value_scipy.pvalue:.3f}",
        ha='center', va='bottom',
        bbox=dict(facecolor='tab:blue', alpha=0.5, boxstyle="round"))

ax.plot(t_array, t.pdf(t_array, dof),
       color='black', lw=2)

ax.set(xlabel='t',
       ylabel='probability density',
       title="t-distribution (N=100)",
       );
```



The answer is yes: the sampled men are significantly taller than the general population, since the p value is smaller than the significance level.