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

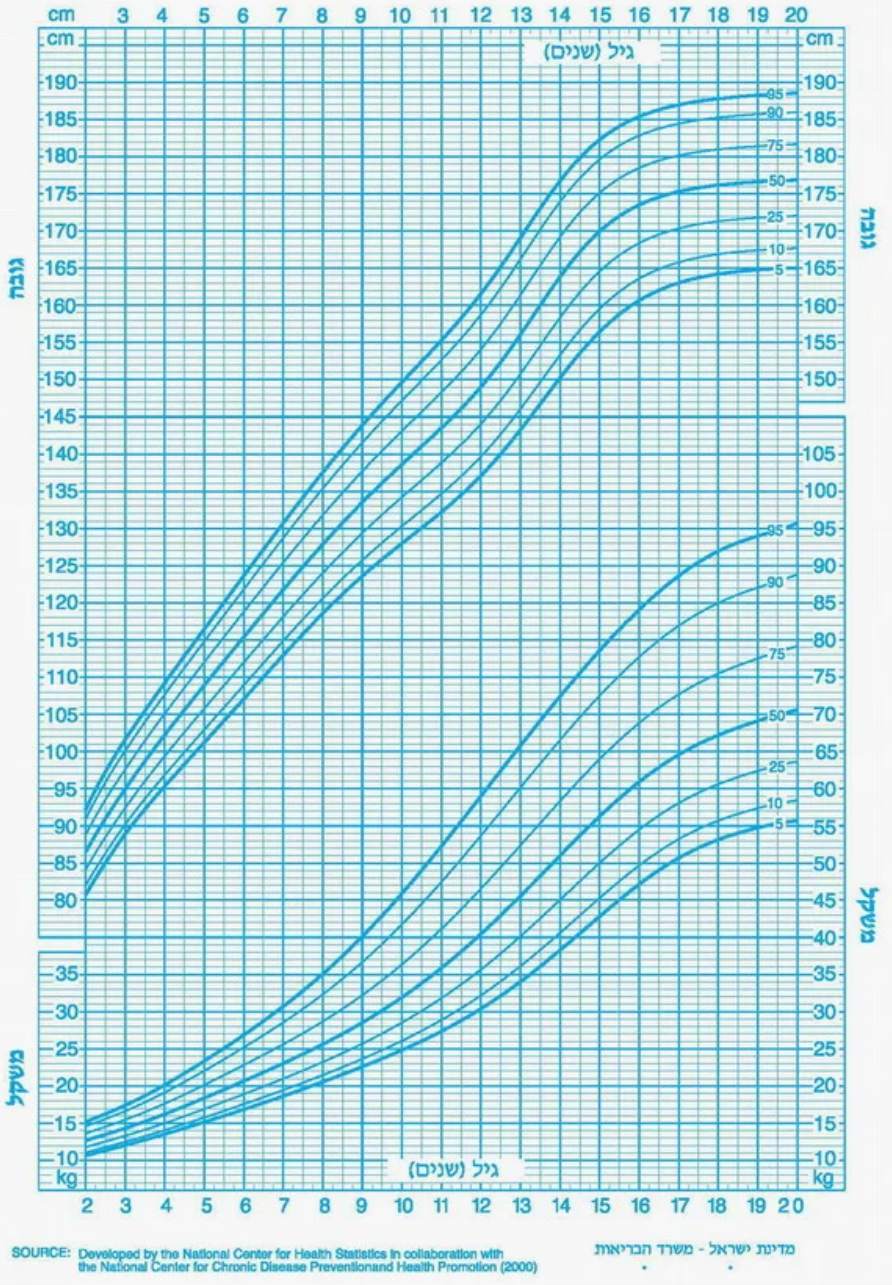
I found growth curves for girls and boys in Israel:

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

For example, see this:

בנים 2-20 שנים - עקומות גובה לפי גיל/ משקל לפי גיל

בנים



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

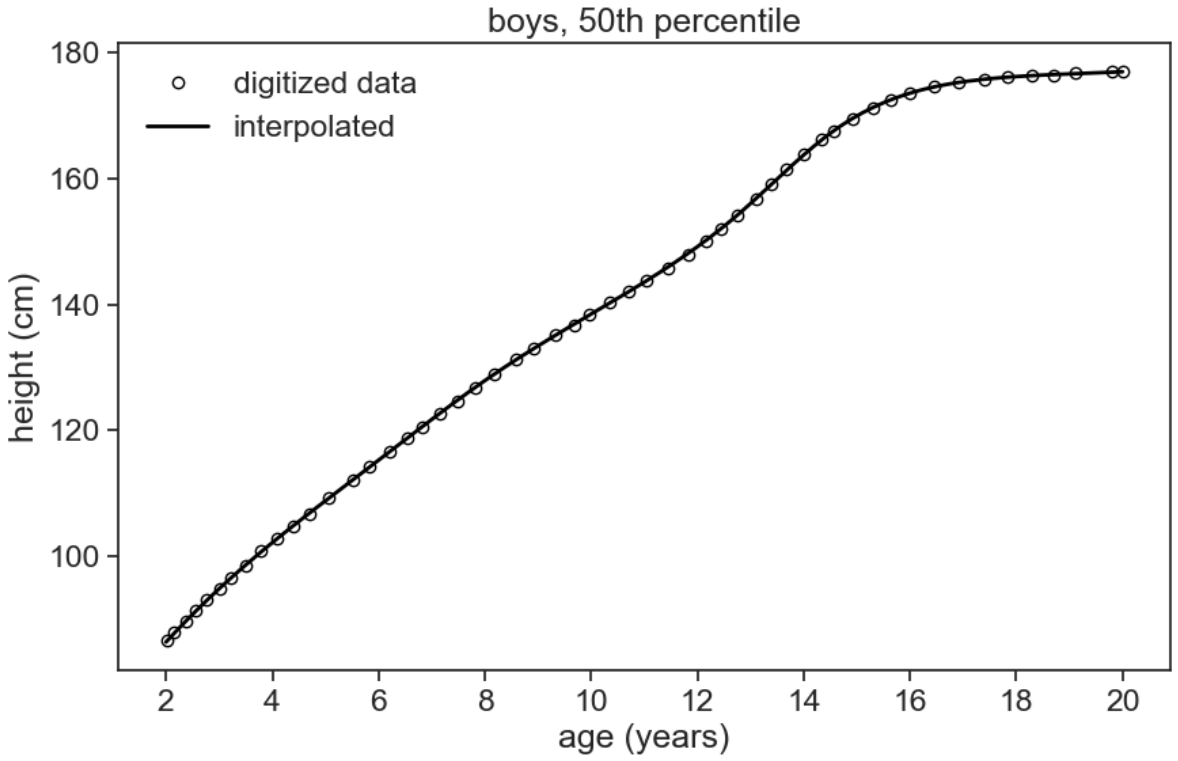
I used the great online resource Web Plot Digitizer v4 to extract the data from the images files. I captured all the growth curves as best as I could. The first step now is to get interpolated versions of the digitized data. For instance, see below the 50th percentile for boys:

```
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'])
```

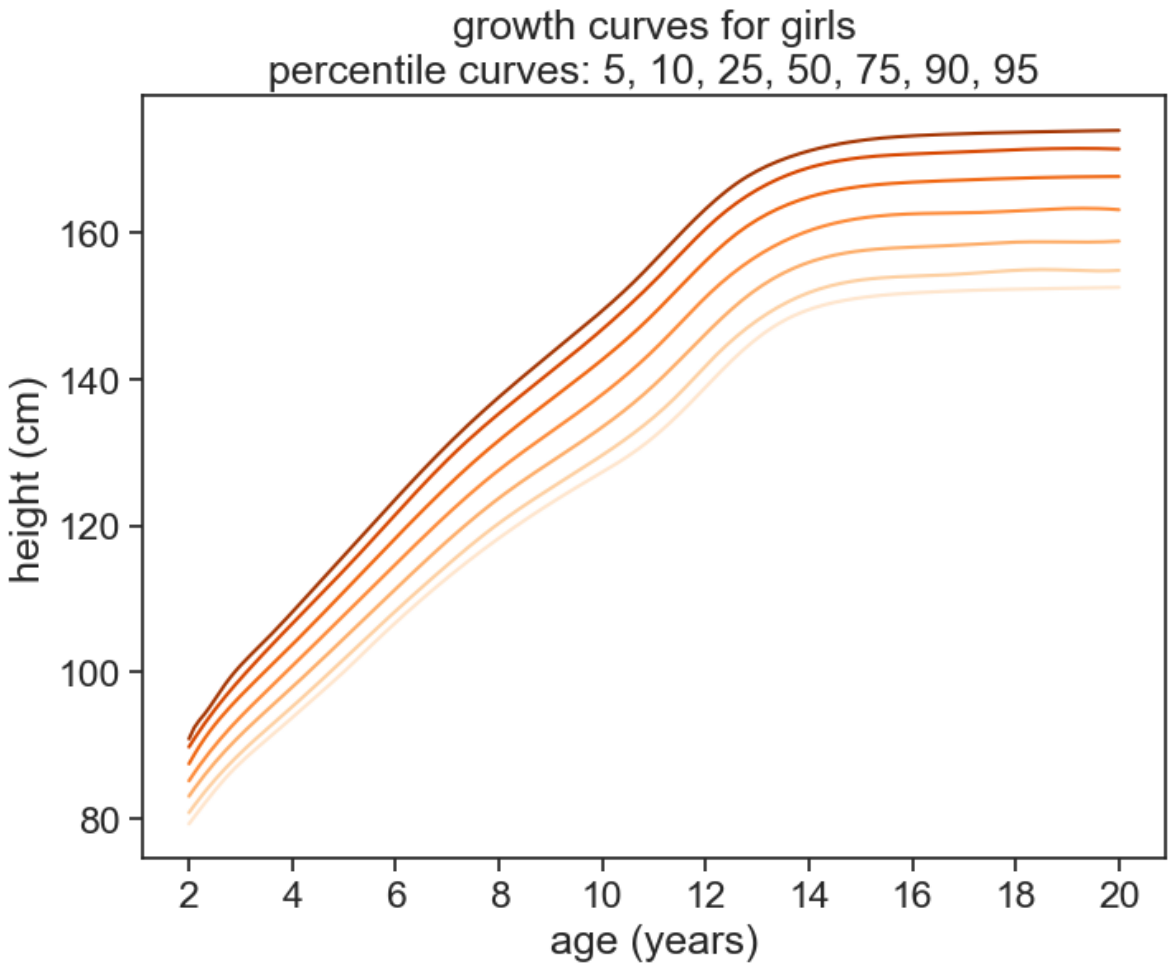
```
    spline = UnivariateSpline(df_temp['age'], df_temp['height'], s=0.5)
    df_girls[col_names[i]] = spline(age_list)

# 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
```

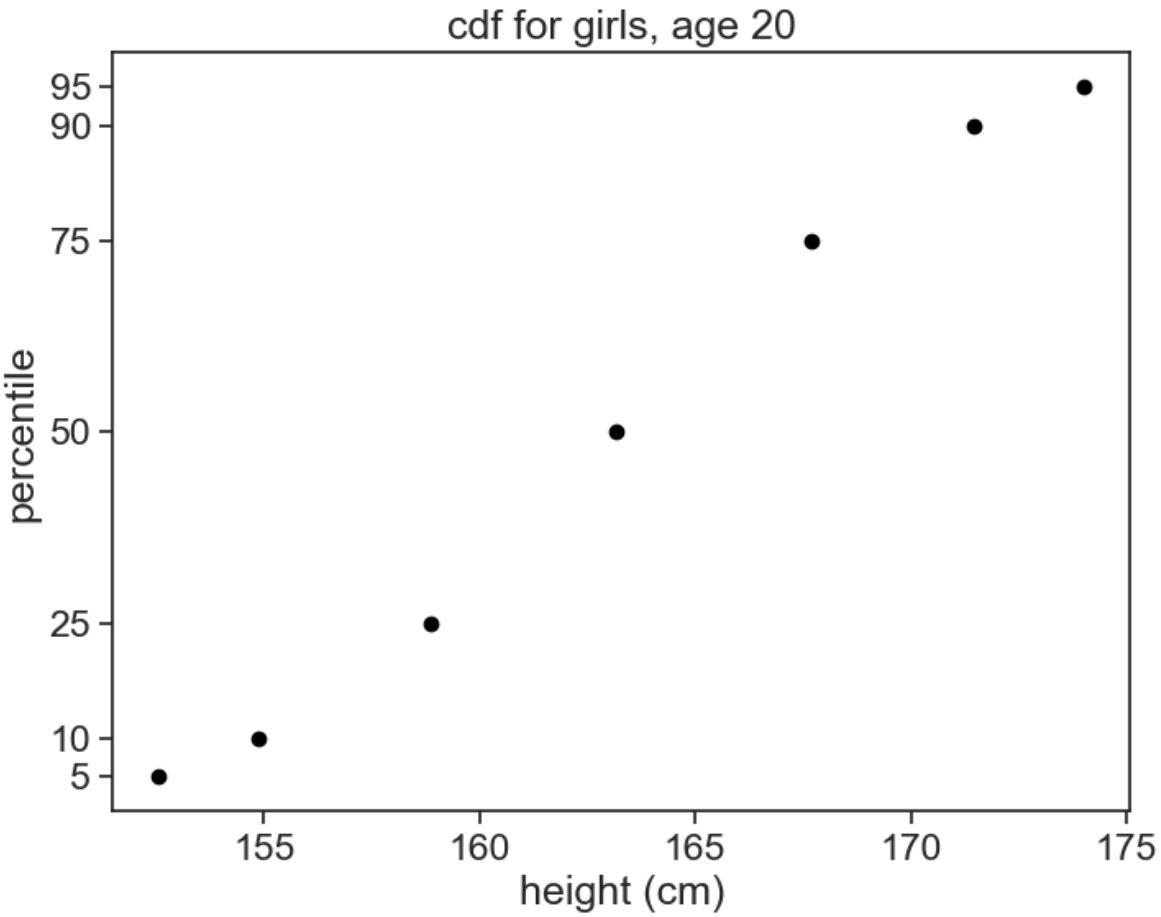
age	p05	p10	p25	p50	p75	p90	p95
2.0	79.269087	80.794167	83.049251	85.155597	87.475854	89.779822	90.882059
2.1	80.202106	81.772053	84.052858	86.207778	88.713405	90.883740	92.409913
2.2	81.130687	82.706754	85.011591	87.211543	89.856186	91.940642	93.416959
2.3	82.048325	83.601023	85.928399	88.170313	90.914093	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

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

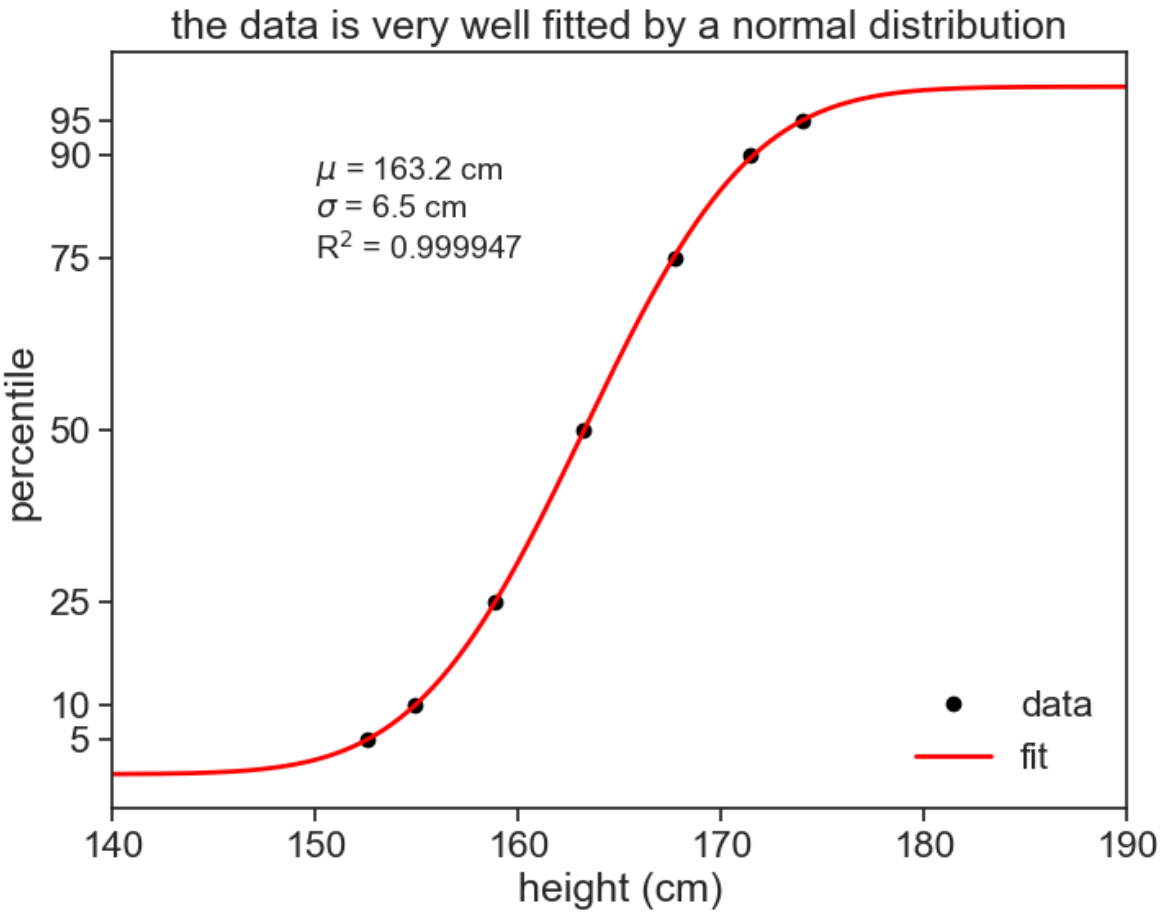
where μ is the mean and σ is the standard deviation of the distribution. The error function erf is a sigmoid function, which is a good approximation for the cdf of the normal distribution.

```
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)
```

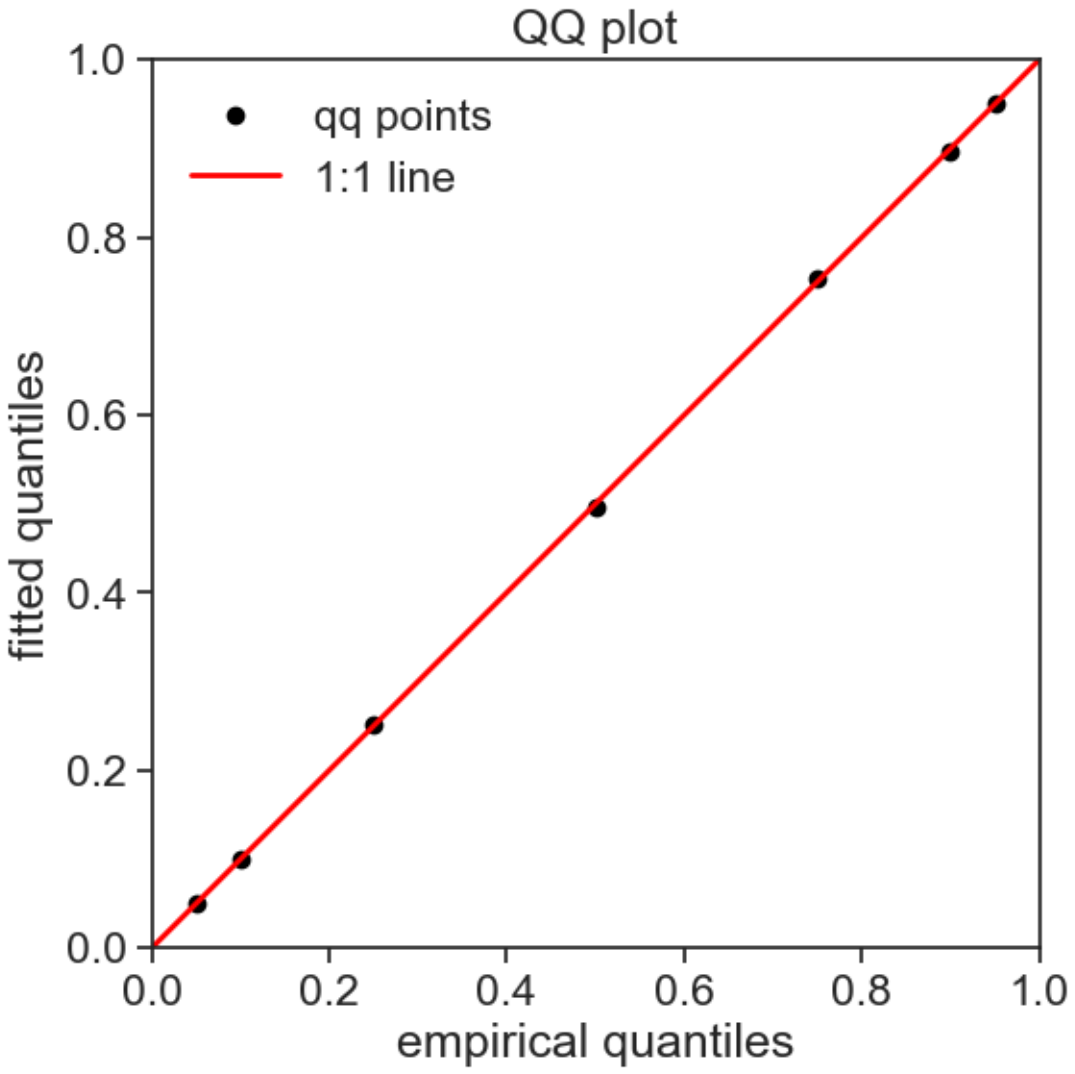
```
fig, ax = plt.subplots(figsize=(8, 6))
percentile_list = np.array([5, 10, 25, 50, 75, 90, 95])
data = df_girls.loc[20.0]
ax.plot(data, percentile_list, ls='', marker='o', markersize=6, color="black", label='data')
fit = erf_model(height_list, *params)
ax.plot(height_list, fit, label='fit', color="red", 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
```

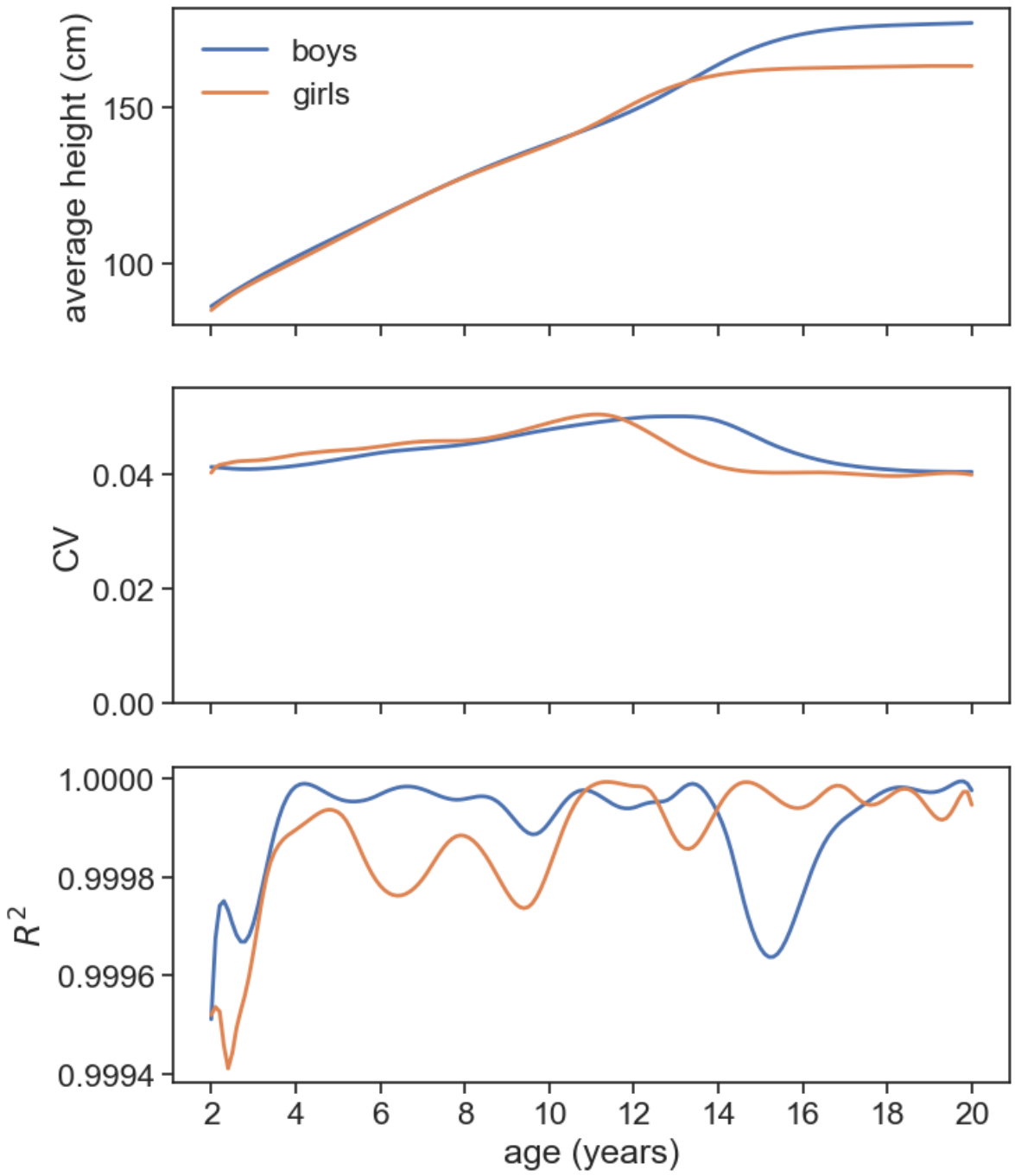
```
p0 = [80, 3]
# loop over ages in the index, calculate mu and sigma
for i in df_boys.index:
    # fit the model to the data
    data = df_boys.loc[i]
    params, _ = curve_fit(erf_model, data, percentile_list, p0=p0,
                          bounds=([70, 2],   # lower bounds for mu and sigma
                                  [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
```

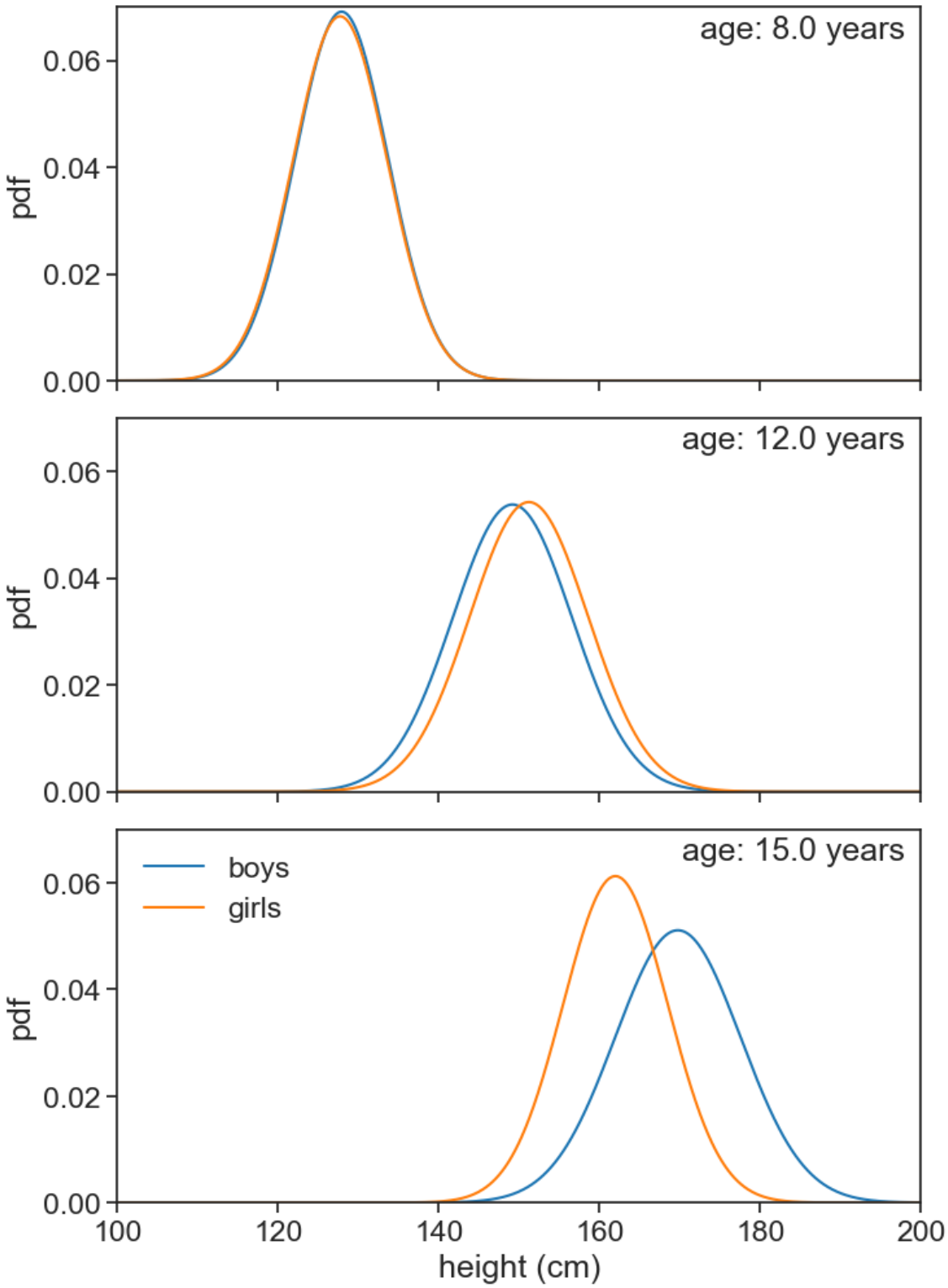
```
Unable to display output for mime type(s): text/html
```

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.

```
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.

Part II

hypothesis testing

2 one-sample t-test

2.1 Question

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

2.2 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

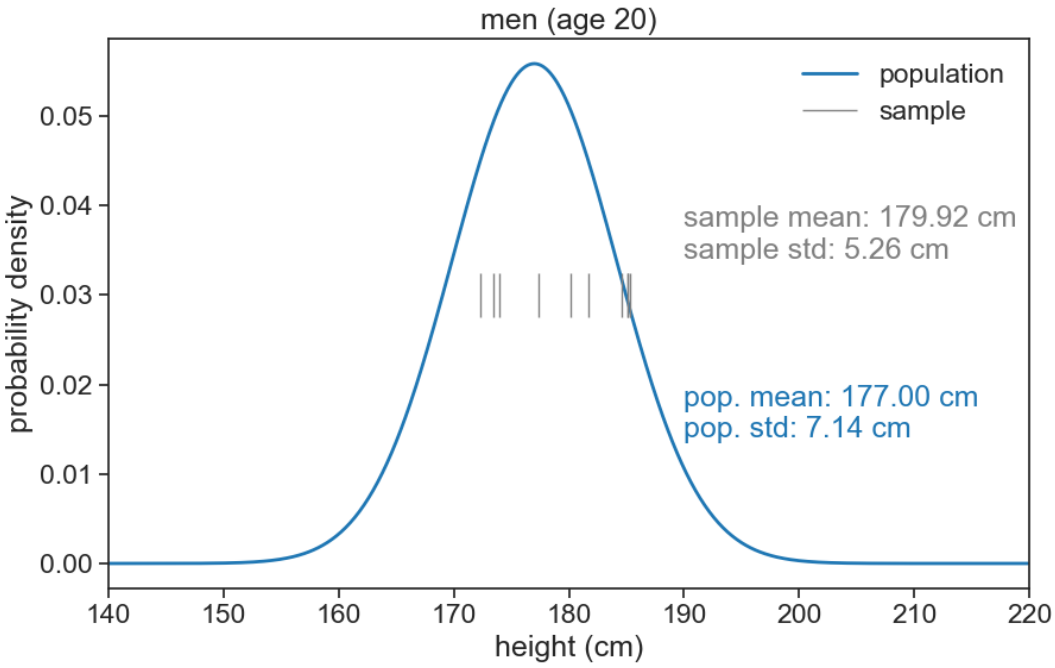
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style="ticks", font_scale=1.5)
from scipy.stats import norm, ttest_1samp, t
%matplotlib widget
```

```
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)
```

```
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.

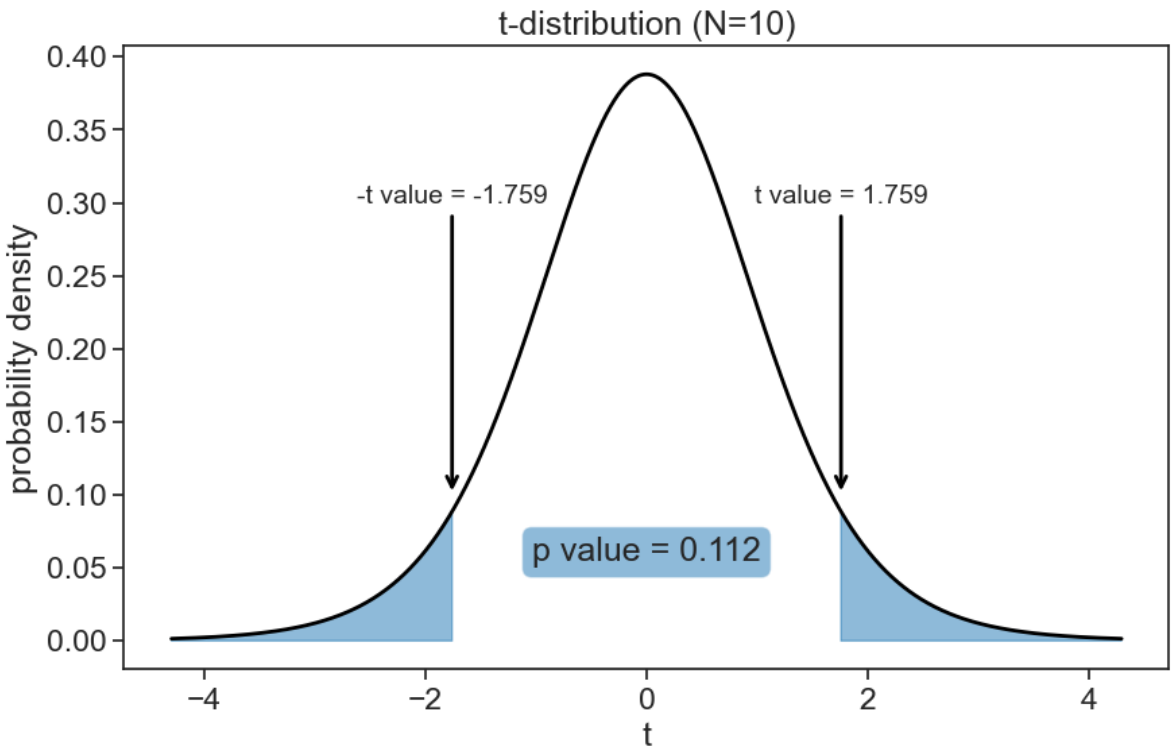
```
t_value_formula = (sample10.mean() - mu_boys) / (sample10.std(ddof=1) / np.sqrt(N))
t_value_scipy = ttest_1samp(sample10, popmean=mu_boys)
print(f"t-value (formula): {t_value_formula:.3f}")
print(f"t-value (scipy): {t_value_scipy.statistic:.3f}")
```

```
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.

```
# 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)",
```

```
    );
```



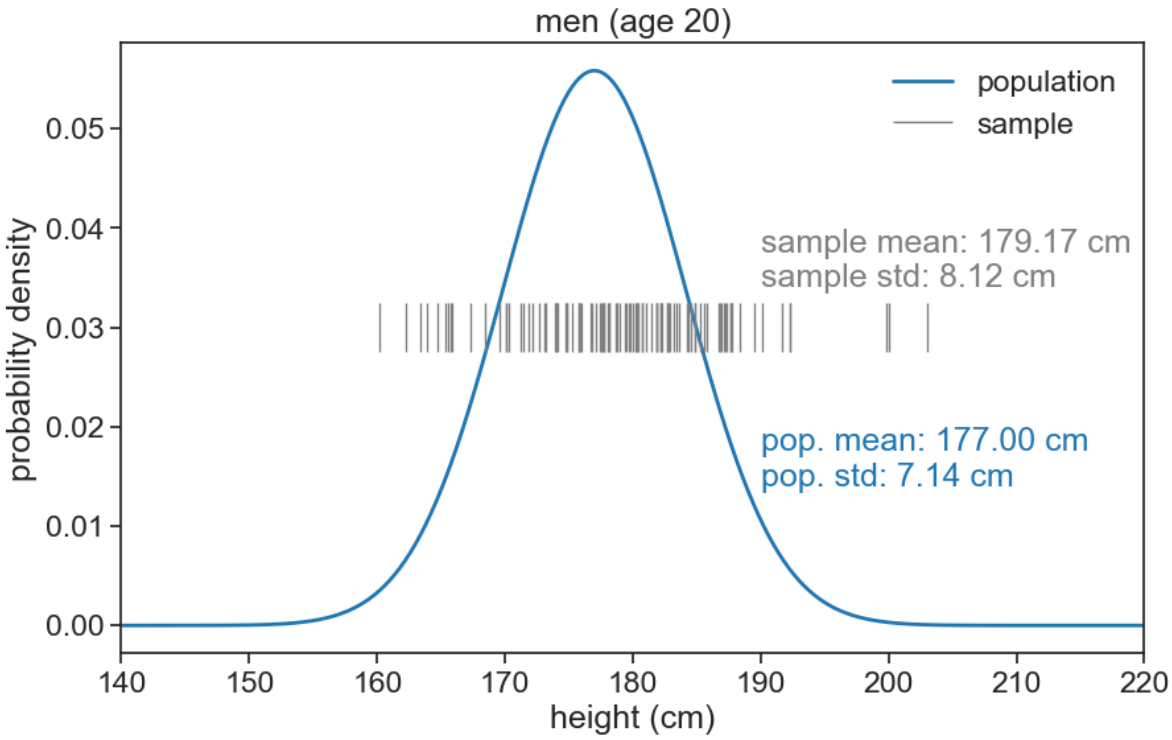
The p value is the fraction of the t distribution that is more extreme than the observed t value. If the p value is less than the significance level, we reject the null hypothesis. In this case, the p value is larger than the significance level, so we fail to reject the null hypothesis. This means that 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.

2.3 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),
       );
```

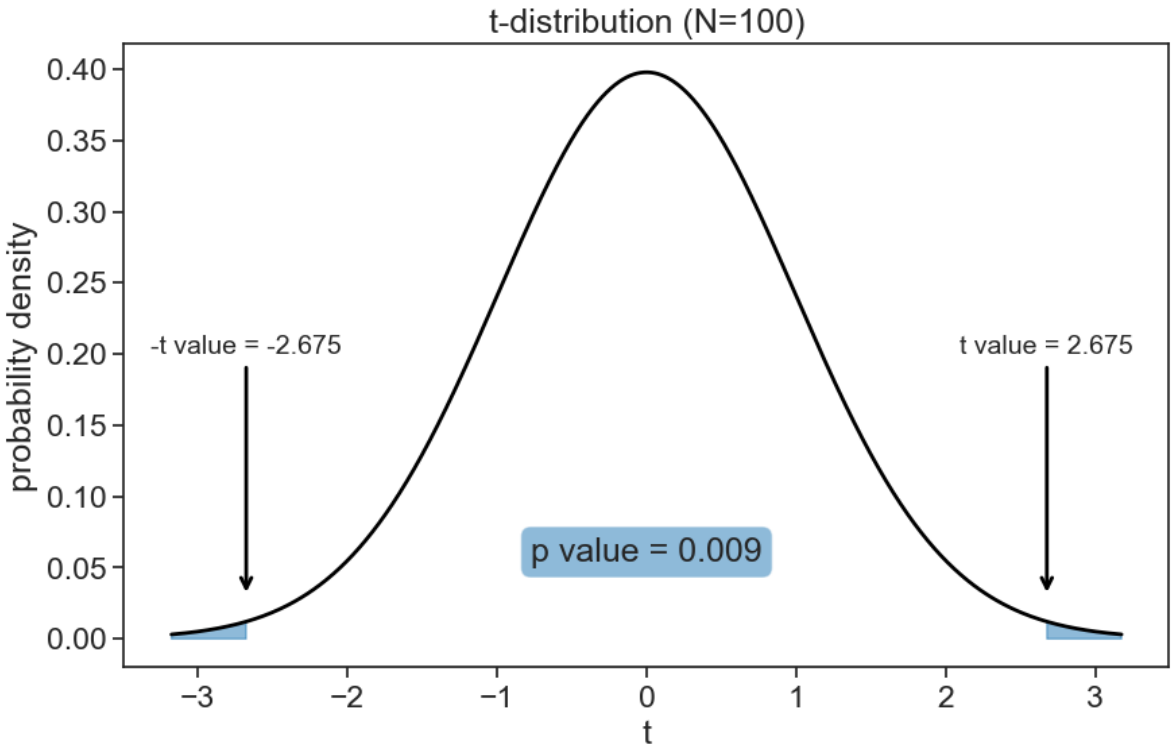



```
t_value_scipy = ttest_1samp(sample100, popmean=mu_boys)
print(f"t-value: {t_value_scipy.statistic:.3f}")
print(f"p-value: {t_value_scipy.pvalue:.3f}")
```

```
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)",
       );
```



2.4 Question 2

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

2.5 Hypotheses

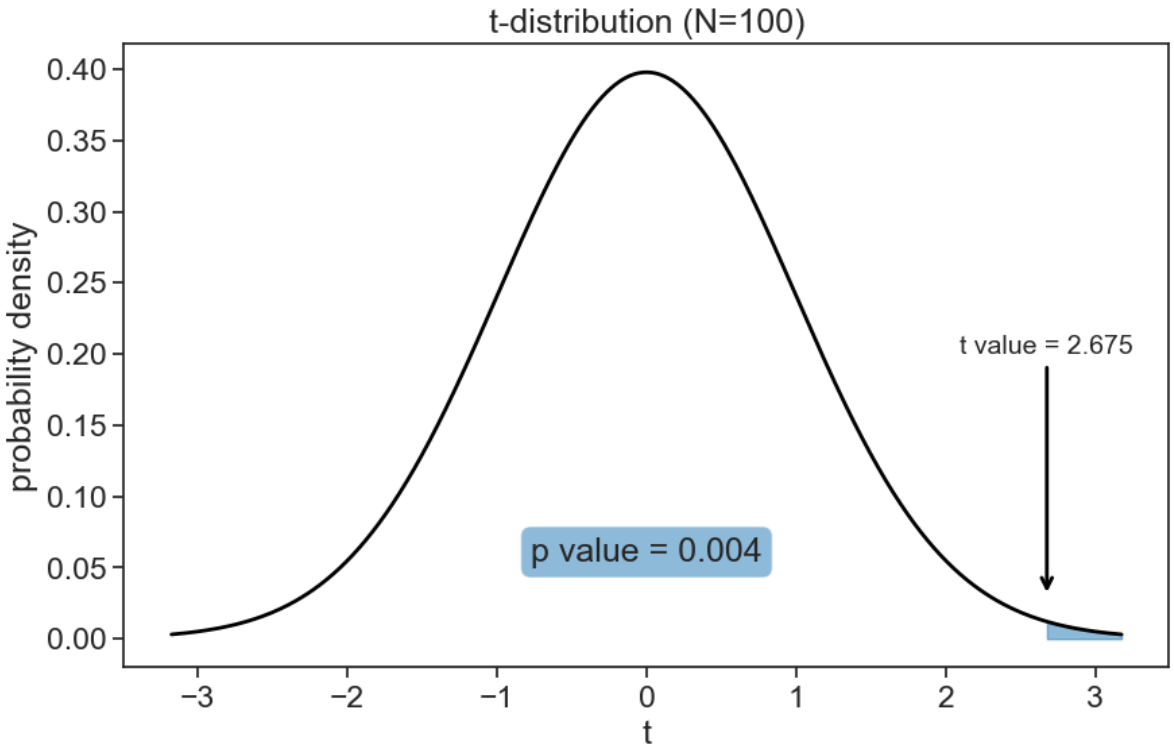
- Null hypothesis: The sample mean is equal to the population mean.
- Alternative hypothesis: The sample mean is higher the population mean.
- Significance level: 0.05

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
```

```
# 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.

3 independent samples t-test

3.1 Question

Are 12-year old girls significantly taller than 12-year old boys?

3.2 Hypotheses

- Null hypothesis: Girls and boys have the same mean height.
- Alternative hypothesis: Girls are *significantly* taller.
- Significance level: 0.05

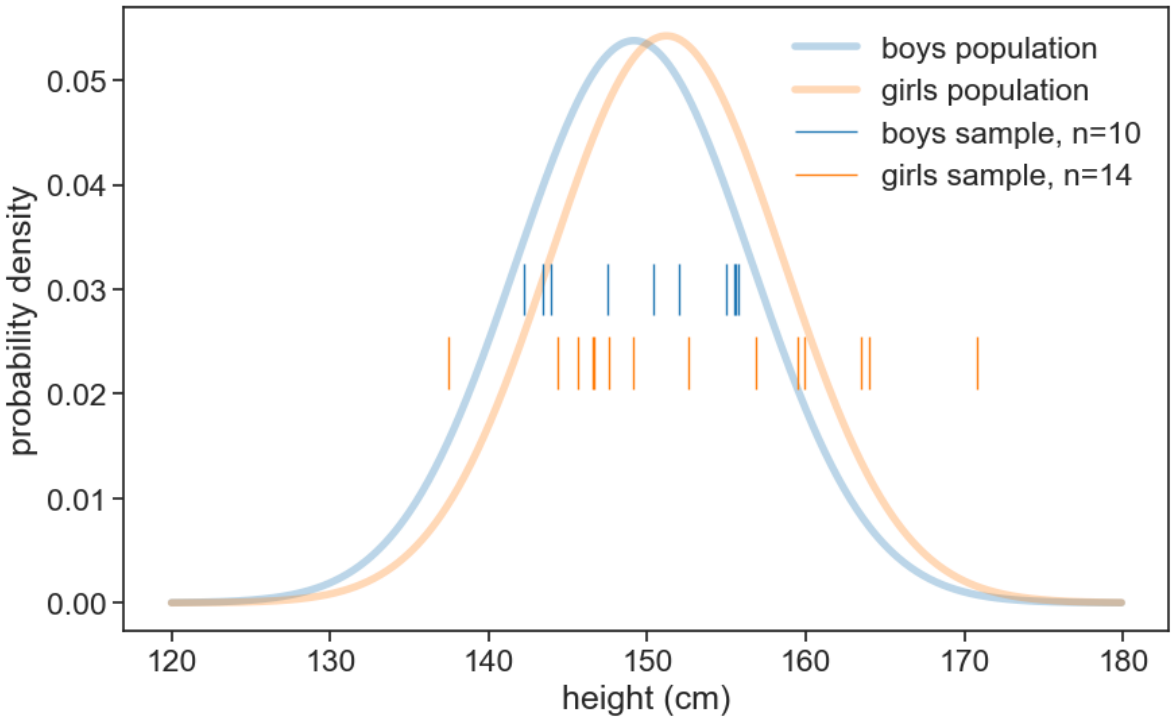
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(style="ticks", font_scale=1.5)
from scipy.stats import norm, ttest_ind, t
# %matplotlib widget
```

```
df_boys = pd.read_csv('../archive/data/height/boys_height_stats.csv', index_col=0)
df_girls = pd.read_csv('../archive/data/height/girls_height_stats.csv', index_col=0)
age = 12.0
mu_boys = df_boys.loc[age, 'mu']
mu_girls = df_girls.loc[age, 'mu']
sigma_boys = df_boys.loc[age, 'sigma']
sigma_girls = df_girls.loc[age, 'sigma']
```

In this example, we sampled 10 boys and 14 girls. See below the samples data and their underlying distributions.

```
N_boys = 10
N_girls = 14
# set scipy seed for reproducibility
np.random.seed(314)
sample_boys = norm.rvs(size=N_boys, loc=mu_boys, scale=sigma_boys)
sample_girls = norm.rvs(size=N_girls, loc=mu_girls, scale=sigma_girls)
```

```
height_list = np.arange(120, 180, 0.1)
pdf_boys = norm.pdf(height_list, loc=mu_boys, scale=sigma_boys)
pdf_girls = norm.pdf(height_list, loc=mu_girls, scale=sigma_girls)
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(height_list, pdf_boys, lw=4, alpha=0.3, color='tab:blue', label='boys population')
ax.plot(height_list, pdf_girls, lw=4, alpha=0.3, color='tab:orange', label='girls population

ax.eventplot(sample_boys, orientation="horizontal", lineoffsets=0.03,
             linewidth=1, linelengths= 0.005,
             colors='tab:blue', label=f'boys sample, n={N_boys}')
ax.eventplot(sample_girls, orientation="horizontal", lineoffsets=0.023,
             linewidth=1, linelengths= 0.005,
             colors='tab:orange', label=f'girls sample, n={N_girls}')
ax.legend(frameon=False)
ax.set(xlabel='height (cm)',
       ylabel='probability density',
    )
```



To answer the question, we will use an independent samples t-test.

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\Theta} \tag{3.1}$$

$$\Theta = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}} \tag{3.2}$$

This is a generalization of the one-sample t-test. If we take one of the samples to be infinite, we get the one-sample t-test.

We can compute the t-statistic by ourselves, and compare the results with those of `scipy.stats.ttest_ind`. Because we are interested in the difference between the means, we will use the `equal_var=False` option to compute Welch's t-test. Also, because we are testing the alternative hypothesis that girls are taller, we will use the one sided test.

```
Theta = np.sqrt(sample_boys.std(ddof=1)**2/sample_boys.size + \
                sample_girls.std(ddof=1)**2/sample_girls.size)
t_stat = (sample_boys.mean() - sample_girls.mean()) / Theta
dof = N_boys + N_girls - 2
p_val = t.cdf(t_stat, dof)
```

```
# the option alternative="less" is used because we are testing whether the first sample (boys
t_value_scipy = ttest_ind(sample_boys, sample_girls, equal_var=False, alternative="less")

print(f"t-statistic: {t_stat:.3f}, p-value: {p_val:.3f}")
print(f"t-statistic (scipy): {t_value_scipy.statistic:.3f}, p-value (scipy): {t_value_scipy.p
```

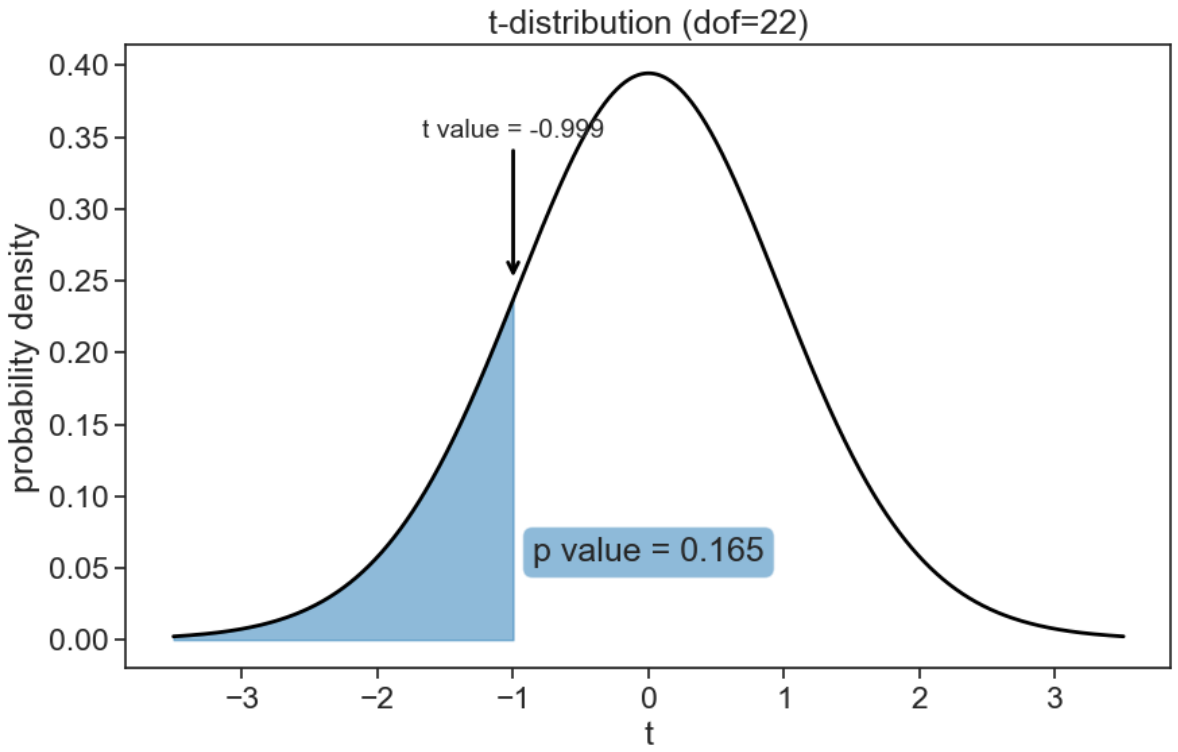
```
t-statistic: -0.999, p-value: 0.164
t-statistic (scipy): -0.999, p-value (scipy): 0.165
```

We got the exact same results :)

Now let's visualize what the p-value means.

```
# degrees of freedom
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, 0.25),
                        xytext=(t_value_scipy.statistic, 0.35),
                        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 (dof=22)",
       );
```



Because the p-value is higher than the significance level, we fail to reject the null hypothesis. This means that, based on the data, we cannot conclude that girls are significantly taller than boys.

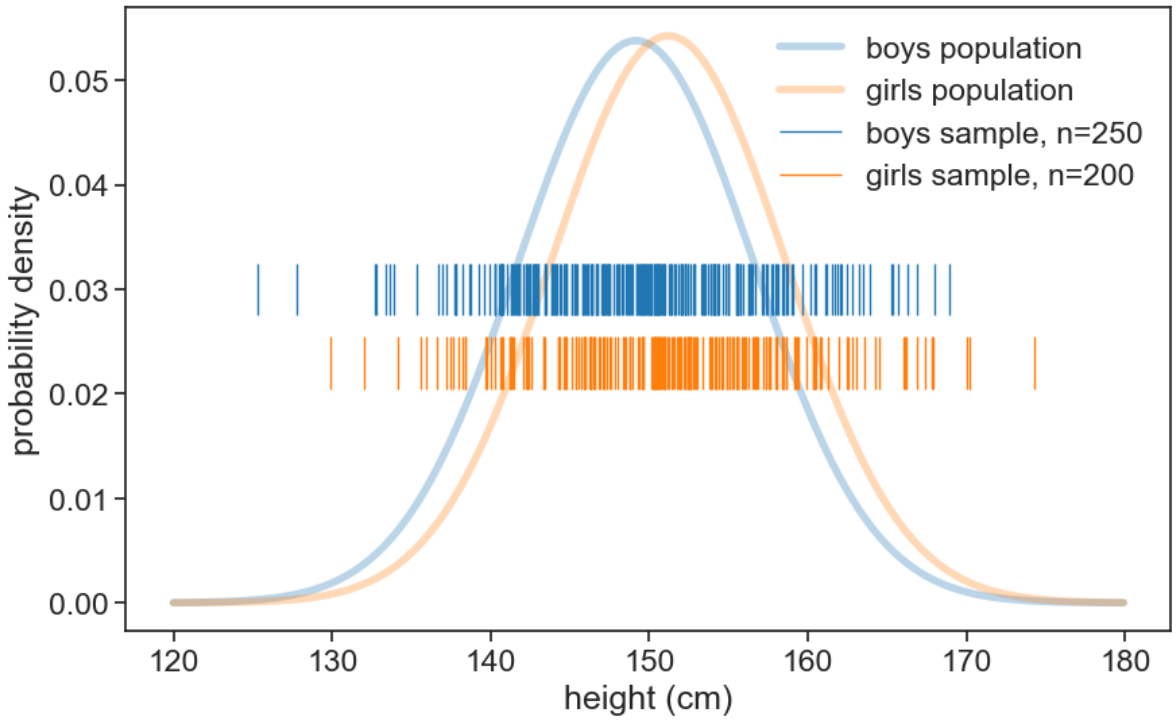
3.3 increasing sample size

Let's increase the sample size to see how it affects the p-value. We'll sample 250 boys and 200 girls now.

```
N_boys = 250
N_girls = 200
# set scipy seed for reproducibility
np.random.seed(314)
```

```
sample_boys = norm.rvs(size=N_boys, loc=mu_boys, scale=sigma_boys)
sample_girls = norm.rvs(size=N_girls, loc=mu_girls, scale=sigma_girls)
```

```
height_list = np.arange(120, 180, 0.1)
pdf_boys = norm.pdf(height_list, loc=mu_boys, scale=sigma_boys)
pdf_girls = norm.pdf(height_list, loc=mu_girls, scale=sigma_girls)
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(height_list, pdf_boys, lw=4, alpha=0.3, color='tab:blue', label='boys population')
ax.plot(height_list, pdf_girls, lw=4, alpha=0.3, color='tab:orange', label='girls population

ax.eventplot(sample_boys, orientation="horizontal", lineoffsets=0.03,
             linewidth=1, linelengths= 0.005,
             colors='tab:blue', label=f'boys sample, n={N_boys}')
ax.eventplot(sample_girls, orientation="horizontal", lineoffsets=0.023,
             linewidth=1, linelengths= 0.005,
             colors='tab:orange', label=f'girls sample, n={N_girls}')
ax.legend(frameon=False)
ax.set(xlabel='height (cm)',
       ylabel='probability density',
    )
```



```
Theta = np.sqrt(sample_boys.std(ddof=1)**2/sample_boys.size + \
                sample_girls.std(ddof=1)**2/sample_girls.size)
t_stat = (sample_boys.mean() - sample_girls.mean()) / Theta
dof = N_boys + N_girls - 2
p_val = t.cdf(t_stat, dof)

# the option alternative="less" is used because we are testing whether the first sample (boys
t_value_scipy = ttest_ind(sample_boys, sample_girls, equal_var=False, alternative="less")

print(f"t-statistic: {t_stat:.3f}, p-value: {p_val:.3f}")
print(f"t-statistic (scipy): {t_value_scipy.statistic:.3f}, p-value (scipy): {t_value_scipy.p
```

```
t-statistic: -2.639, p-value: 0.004
t-statistic (scipy): -2.639, p-value (scipy): 0.004
```

We found now a p-value lower than the significance level, so we reject the null hypothesis. This means that, based on the data, we can conclude that girls are significantly taller than boys.

Part III

confidence interval

4 basic concepts

Suppose we randomly select 30 seven-year-old boys from schools around the country and measure their heights (this is our sample). We'd like to use their average height to estimate the true average height of all seven-year-old boys nationwide (the population). Because different samples of 30 boys would yield slightly different averages, we need a way to quantify that uncertainty. A confidence interval gives us a range—based on our sample data—within which we can be reasonably sure the true population mean lies.

```
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_ind, t
import scipy
from matplotlib.lines import Line2D
import matplotlib.gridspec as gridspec
# %matplotlib widget
```

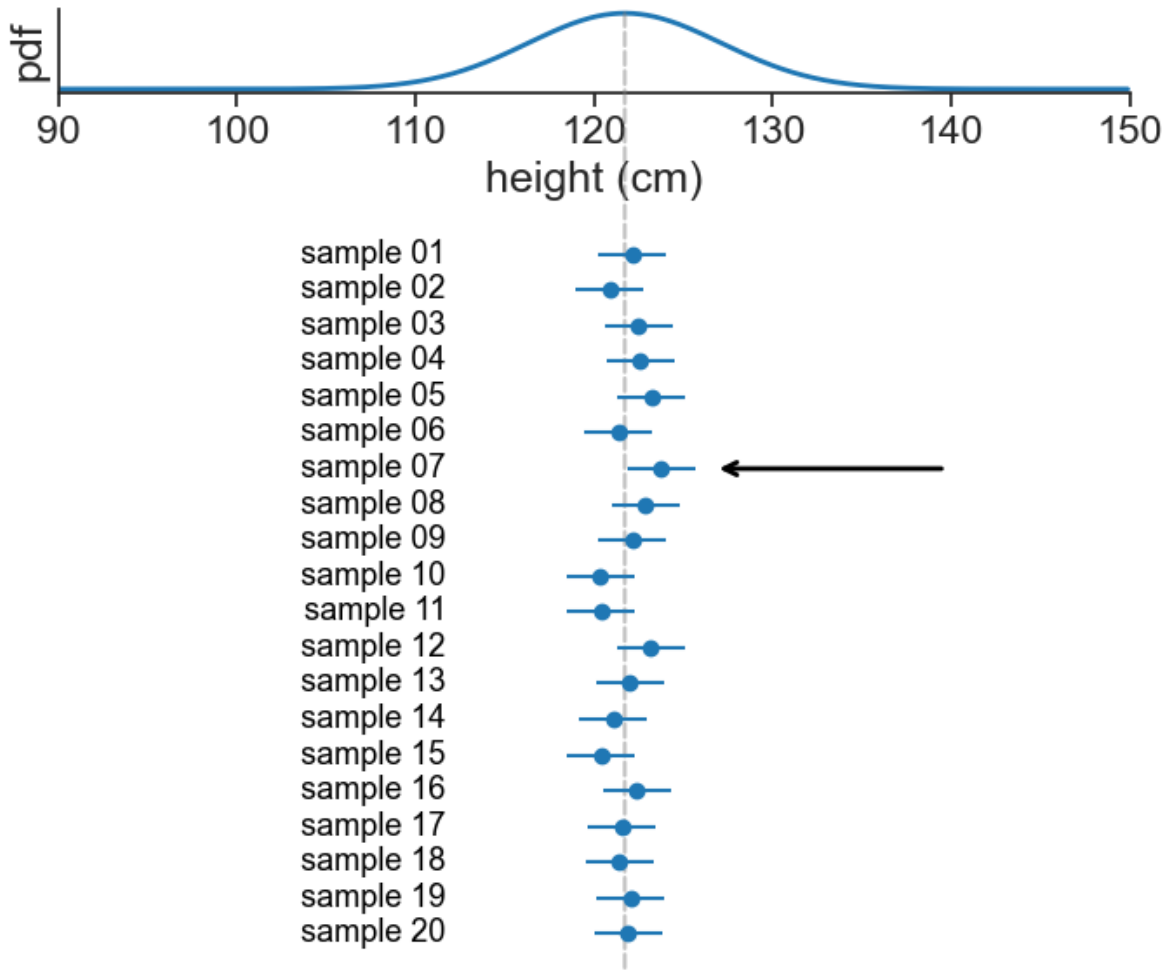
```
df_boys = pd.read_csv('../archive/data/height/boys_height_stats.csv', index_col=0)
age = 7.0
mu_boys = df_boys.loc[age, 'mu']
sigma_boys = df_boys.loc[age, 'sigma']
```

See the distribution of height for seven-year-old boys. Below it we see 20 samples of groups of 30. The 95% confidence interval is the range of values that, on average, 95% of the samples CI fall within the true population mean. In this case, this amounts to one out of the 20 samples.

```
np.random.seed(628)
height_list = np.arange(90, 150, 0.1)
pdf_boys = norm.pdf(height_list, loc=mu_boys, scale=sigma_boys)

fig = plt.figure(figsize=(8, 6))
gs = gridspec.GridSpec(2, 1, height_ratios=[0.1, 0.9])
gs.update(left=0.09, right=0.86,top=0.98, bottom=0.06, hspace=0.30, wspace=0.05)
```

```
ax0 = plt.subplot(gs[0, 0])
ax1 = plt.subplot(gs[1, 0])

ax0.plot(height_list, pdf_boys, lw=2, color='tab:blue', label='population')

N_samples = 20
N = 30

for i in range(N_samples):
    sample = norm.rvs(loc=mu_boys, scale=sigma_boys, size=N)
    sample_mean = sample.mean()
    # confidence interval
    alpha = 0.05
    z_crit = scipy.stats.t.isf(alpha/2, N-1)
    # CI = z_crit * sample.std(ddof=1) / np.sqrt(N)
    CI = 1.9
    ax1.errorbar(sample_mean, i, xerr=CI, fmt='o', color='tab:blue',
                 label=f'sample {i+1}' if i == 0 else "", capsize=0)


from matplotlib.patches import ConnectionPatch
line = ConnectionPatch(xyA=(mu_boys, pdf_boys.max()), xyB=(mu_boys, -1), coordsA="data", coo
                       axesA=ax0, axesB=ax1, color="gray", linestyle='--', linewidth=1.5, alp
ax1.add_artist(line)

ax1.annotate(
        '',
        xy=(mu_boys + 5, 13),  # tip of the arrow (first error bar, y=0)
        xytext=(mu_boys + 5 + 13, 13),  # text location
        arrowprops=dict(arrowstyle='->', lw=2, color='black'),
        fontsize=13,
        color='tab:blue',
        ha='left',
        va='center'
)

# write "sample i" for each error bar
for i in range(N_samples):
    ax1.text(mu_boys -10, i, f'sample {N_samples-i:02d}',
             fontsize=13, color='black',
             ha='right', va='center')
```

```
# ax.legend(frameon=False)
ax0.spines['top'].set_visible(False)
ax0.spines['right'].set_visible(False)
ax1.spines['top'].set_visible(False)
ax1.spines['right'].set_visible(False)
ax1.spines['left'].set_visible(False)
ax1.spines['bottom'].set_visible(False)

ax0.set(xticks=np.arange(90, 151, 10),
        xlim=(90, 150),
        xlabel='height (cm)',
        # xticklabels=[],
        yticks=[],
        ylabel='pdf',
        )
ax1.set(xticks=[],
        xlim=(90, 150),
        ylim=(-1, N_samples),
        yticks=[],
       );
```



The image is of course more complex than this. In the graph above, I chose the same interval for all samples. If you look at the code, we could instead calculate the confidence interval for each sample, and that would yield different intervals because they are dependent on the sample standard deviation. Let's not let that distract us from the main point: the confidence interval is a range of values that we can be reasonably sure contains the true population mean.

5 analytical confidence interval

```
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_ind, t
import scipy
# %matplotlib widget
```

We wish to compute the confidence interval for the mean height of 7-year-old boys, for a sample of size N.

We will start our journey with a refresher of the Central Limit Theorem (CLT).

5.1 CLT

The Central Limit Theorem states that the sampling distribution of the sample mean

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$$

approaches a normal distribution as the sample size N increases, regardless of the shape of the population distribution. This normal distribution can be expressed as:

$$\bar{X} \sim N\left(\mu, \frac{\sigma^2}{N}\right),$$

where μ and σ^2 are the population mean and variance, respectively. When talking about samples, we use $\bar{x}$ and s^2 to denote the sample mean and variance.

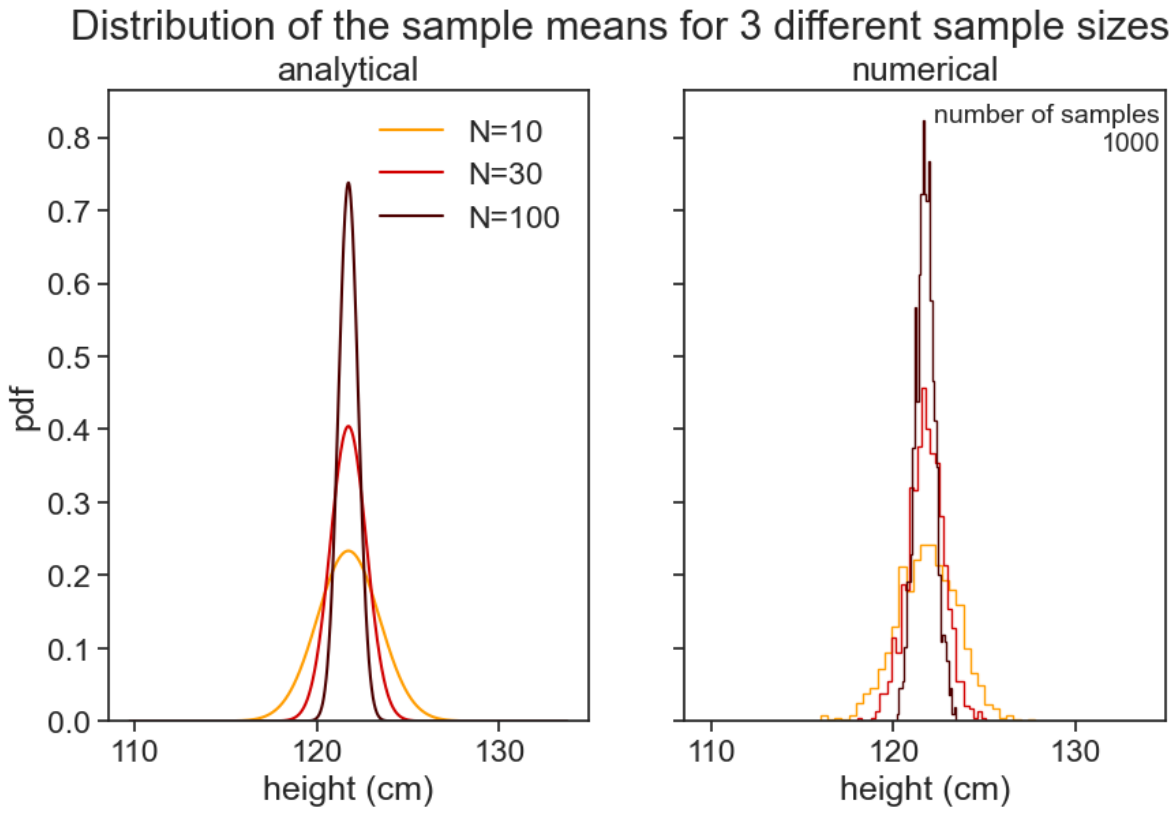
Let's visualize this. The graph below shows how the sample size N affects the sampling distribution of the sample mean $\bar{X}$. The higher the sample size, the more concentrated the

distribution becomes around the population mean μ. If we take N to be infinity, the sampling distribution of the sample mean becomes a delta function at μ, and we will know the exact value of the population mean.

```
df_boys = pd.read_csv('../archive/data/height/boys_height_stats.csv', index_col=0)
mu_boys = df_boys.loc[7.0, 'mu']
sigma_boys = df_boys.loc[7.0, 'sigma']
```

```
fig, ax = plt.subplots(1,2, figsize=(10, 6), sharex=True, sharey=True)

height_list = np.arange(mu_boys-12, mu_boys+12, 0.01)
N_list = [10, 30, 100]
alpha_list = [0.4, 0.6, 1.0]

colors = plt.cm.hot([0.6, 0.3, 0.1])

N_samples = 1000
np.random.seed(628)
mean_list_10 = []
mean_list_30 = []
mean_list_100 = []
for i in range(N_samples):
    mean_list_10.append(np.mean(norm.rvs(size=10, loc=mu_boys, scale=sigma_boys)))
    mean_list_30.append(np.mean(norm.rvs(size=30, loc=mu_boys, scale=sigma_boys)))
    mean_list_100.append(np.mean(norm.rvs(size=100, loc=mu_boys, scale=sigma_boys)))

alpha = 0.05
z_alpha_over_two = norm(loc=mu_boys, scale=SE).ppf(1 - alpha / 2)
z_alpha_over_two = np.round(z_alpha_over_two, 2)

for i,N in enumerate(N_list):
    SE = sigma_boys / np.sqrt(N)
    ax[0].plot(height_list, norm(loc=mu_boys, scale=SE).pdf(height_list),
            color=colors[i], label=f"N={N}")

ax[1].hist(mean_list_10, bins=30, density=True, color=colors[0], label="N=10", align='mid', l
ax[1].hist(mean_list_30, bins=30, density=True, color=colors[1], label="N=10", align='mid', l
ax[1].hist(mean_list_100, bins=30, density=True, color=colors[2], label="N=10", align='mid',

ax[1].text(0.99, 0.98, "number of samples\n1000", ha='right', va='top', transform=ax[1].trans

ax[0].legend(frameon=False)
```

```
ax[0].set(xlabel="height (cm)",
       ylabel="pdf",
       title="analytical"
       )
ax[1].set(xlabel="height (cm)",
          title="numerical"
          )
# title that hovers over both subplots
fig.suptitle(f"Distribution of the sample means for 3 different sample sizes");
```

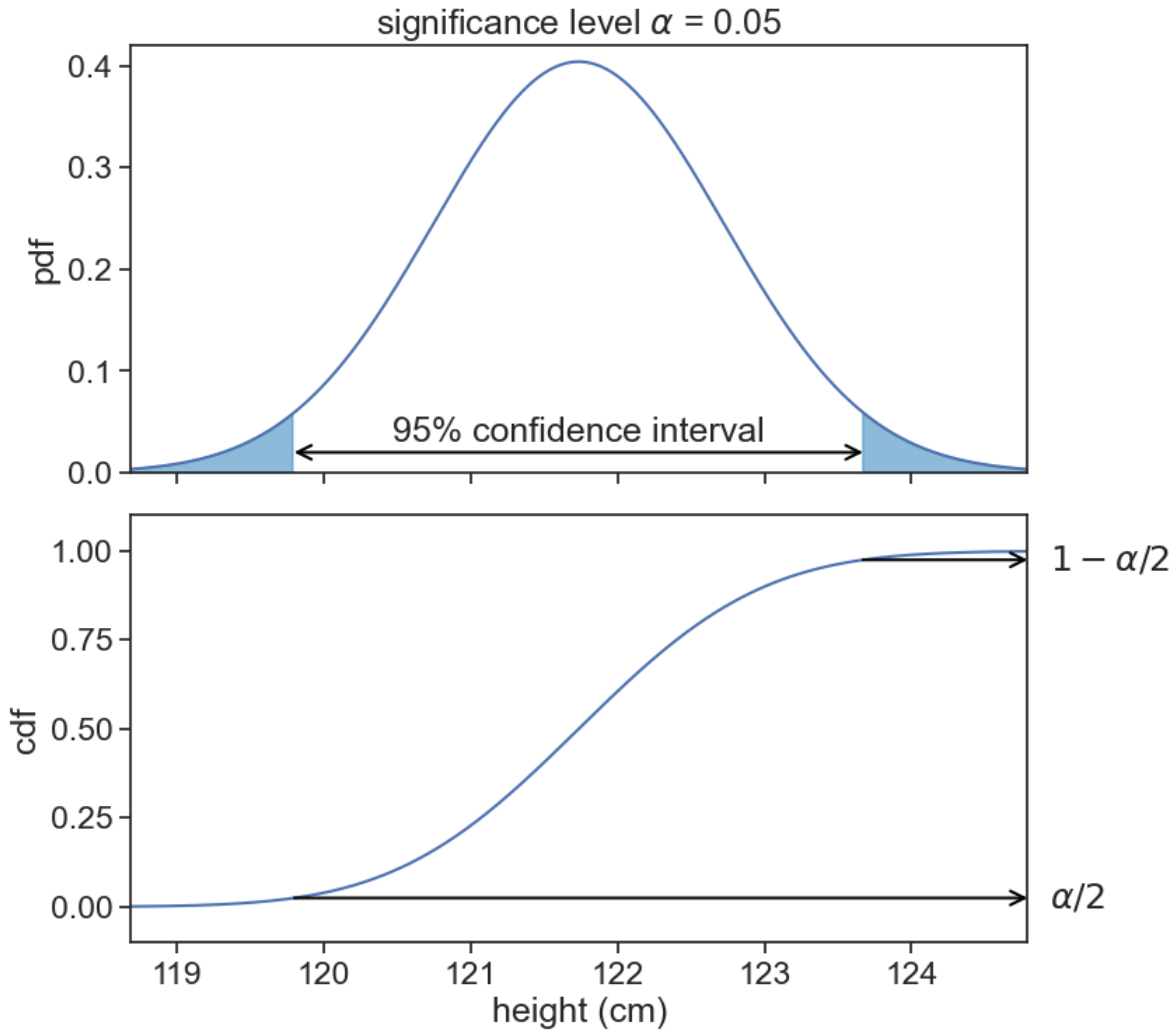


5.2 confidence interval 1

Let's use now the sample size $N = 30$. The confidence interval for a significance level $\alpha = 0.05$ the the interval that leaves $\alpha/2$ of the pdf area in each tail of the distribution.

```
fig, ax = plt.subplots(2, 1, figsize=(8, 8), sharex=True)
plt.subplots_adjust(left=0.1, bottom=0.1, right=0.9, top=0.9, wspace=0.0, hspace=0.1)
N = 30
SE = sigma_boys / np.sqrt(N)

h_min = np.round(norm(loc=mu_boys, scale=SE).ppf(0.001), 2)
h_max = np.round(norm(loc=mu_boys, scale=SE).ppf(0.999), 2)
height_list = np.arange(h_min, h_max, 0.01)

alpha = 0.05
z_alpha_over_two_hi = np.round(norm(loc=mu_boys, scale=SE).ppf(1 - alpha / 2), 2)
z_alpha_over_two_lo = np.round(norm(loc=mu_boys, scale=SE).ppf(alpha / 2), 2)


ax[0].plot(height_list, norm(loc=mu_boys, scale=SE).pdf(height_list))
ax[1].plot(height_list, norm(loc=mu_boys, scale=SE).cdf(height_list))

ax[0].fill_between(height_list, norm(loc=mu_boys, scale=SE).pdf(height_list),
                   where=((height_list > z_alpha_over_two_hi) | (height_list < z_alpha_over_t
                   color='tab:blue', alpha=0.5,
                   label='rejection region')

ax[0].annotate(f"",
               xy=(z_alpha_over_two_hi, 0.02),
               xytext=(z_alpha_over_two_lo, 0.02),
               arrowprops=dict(arrowstyle="<->", lw=1.5, color='black', shrinkA=0.0, shrinkB=
               )

ax[1].annotate(f"",
               xy=(z_alpha_over_two_hi, norm(loc=mu_boys, scale=SE).cdf(z_alpha_over_two_hi))
               xytext=(h_max, norm(loc=mu_boys, scale=SE).cdf(z_alpha_over_two_hi)),
               arrowprops=dict(arrowstyle="<-", lw=1.5, color='black', shrinkA=0.0, shrinkB=0
               )
ax[1].annotate(f"",
               xy=(z_alpha_over_two_lo, norm(loc=mu_boys, scale=SE).cdf(z_alpha_over_two_lo))
               xytext=(h_max, norm(loc=mu_boys, scale=SE).cdf(z_alpha_over_two_lo)),
               arrowprops=dict(arrowstyle="<-", lw=1.5, color='black', shrinkA=0.0, shrinkB=0
               )
ax[1].text(h_max+0.15, norm(loc=mu_boys, scale=SE).cdf(z_alpha_over_two_lo), r"$\alpha/2$",
           ha="left", va="center")
ax[1].text(h_max+0.15, norm(loc=mu_boys, scale=SE).cdf(z_alpha_over_two_hi), r"$1-\alpha/2$"
           ha="left", va="center")
```

```
ax[0].text(mu_boys, 0.03, "95% confidence interval", ha="center")
ax[0].set(ylim=(0, 0.42),
          ylabel="pdf",
          title=r"significance level $\alpha$ = 0.05",
          )
ax[1].set(ylim=(-0.1, 1.1),
          xlim=(h_min, h_max),
          ylabel="cdf",
          xlabel="height (cm)",
          );
```



That's it. That's the whole story.

5.3 confidence interval 2

The rest is repackaging the above in a slightly different way. Instead of finding the top and bottom of the confidence interval according to the cdf of a normal distribution of mean μ and variance σ^2/N, we first standardize this distribution to a standard normal distribution $Z \sim N(0,1)$, compute the confidence interval for Z, and then transform it back to the original distribution.

If the distribution of the sample mean $\bar{X}$

$$\bar{X} \sim N\left(\mu, \frac{\sigma^2}{N}\right),$$

then the standardized variable Z is defined as:

$$Z = \frac{\bar{x} - \mu}{\sigma/\sqrt{N}} \sim N(0,1).$$

Why is this useful? Because we usually use the same significance level α for all confidence intervals, and we can compute the confidence interval for Z once and use it for all confidence intervals. For $Z \sim N(0,1)$ and $\alpha = 0.05$, the top and bottom of the confidence interval are $Z\_\alpha/2 = \pm 1.96$. Now we only have to invert the expression above to get the confidence interval for $\bar{X}$:

$$X_{1,2} = \mu \pm Z_{\alpha/2} \cdot \frac{\sigma}{\sqrt{N}}.$$

The very last thing we have to account for is the fact that we don't know the population statistics μ and σ^2. Instead, we have to use the sample statistics $\bar{x}$ and s^2. Furthermore, we have to use the t-distribution instead of the normal distribution, because we are estimating the population variance from the sample variance. The t-distribution has a shape similar to the normal distribution, but it has heavier tails, which accounts for the additional uncertainty introduced by estimating the population variance. Thus, we replace μ with $\bar{x}$ and σ^2 with s^2, and we use the t-distribution with $N-1$ degrees of freedom. This gives us the final expression for the confidence interval:

$$X_{1,2} = \bar{x} \pm t^*_{N-1} \cdot \frac{s}{\sqrt{N}},$$

where t^*_{N-1} is the critical value from the t-distribution with $N-1$ degrees of freedom.