1 Lecture 7: More statistics

Data Visualization · 1-DAV-105

Lecture by Broňa Brejová

1.1 Importing libraries and data

As usual, we start by importing libraries. We add `scipy.stats` library for working with probability distributions. One more library will be added at the end of the lecture.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from IPython.display import Markdown

import scipy.stats
```

In this lecture, we will use the World bank dataset from Lecture 03b (table `countries`) and the FSEV young people survey from homework I05 (table `fsev`).

Country data set was downloaded from WorldBank https://databank.worldbank.org/home under CC BY 4.0 license.

Survey data set was downloaded from https://www.kaggle.com/miroslavsabo/young-people-survey . The list of questions and meaning of responses is given in this document.

[2]:
```
url = 'https://bbrejova.github.io/viz/data/fsev-responses.csv'
fsev = pd.read_csv(url)
display(Markdown("**Size of fsev table:**"), fsev.shape)
```

Size of fsev table:

```
(1010, 150)
```

[3]:
```
url = 'https://bbrejova.github.io/viz/data/World_bank.csv'
countries = pd.read_csv(url).set_index('Country')
display(countries.describe())
```

```
       Population2000  Population2010  Population2018          Area  \
count    2.170000e+02    2.170000e+02    2.160000e+02  2.100000e+02
mean     2.807447e+07    3.179140e+07    3.502277e+07  6.285202e+05
std      1.154879e+08    1.278245e+08    1.369735e+08  1.853588e+06
min      9.394000e+03    1.000500e+04    1.150800e+04  1.000000e+01
25%      5.924680e+05    6.896920e+05    7.728515e+05  1.175500e+04
50%      5.069302e+06    5.824065e+06    6.572040e+06  1.016695e+05
75%      1.641085e+07    2.053295e+07    2.500432e+07  4.589875e+05
max      1.262645e+09    1.337705e+09    1.392730e+09  1.709825e+07
```

```
             GDP2000        GDP2010        GDP2018  Expectancy2000  \
count    199.000000     206.000000     203.000000      201.000000
mean    8250.267678   15830.982630   18646.763463       67.012010
std    13035.094974   23627.273261   28014.764171       10.092609
min      124.460800     234.235647     271.752044       39.441000
25%      644.496708    1605.885350    2521.451059       60.063000
50%     2007.735175    5642.872248    6941.235848       70.176000
75%    10651.666888   19785.689495   23298.819657       74.403000
max    82367.991277  150725.194124  185829.017960       81.076098

       Expectancy2010  Expectancy2018  Fertility2000  Fertility2010  \
count      200.000000      198.000000     200.000000     201.000000
mean        70.407538       72.719232       3.236187       2.897654
std          8.848635        7.561327       1.734722       1.474274
min         45.100000       52.805000       0.950000       1.093000
25%         65.024500       67.623750       1.773750       1.796000
50%         72.720500       74.097000       2.730000       2.340000
75%         76.609000       78.200610       4.486000       3.913000
max         82.978049       84.934146       7.679000       7.473000

       Fertility2018
count     200.000000
mean        2.647406
std         1.261432
min         0.977000
25%         1.697500
50%         2.198000
75%         3.558000
max         6.913000
```

[4]:
```
display(Markdown("**Values of life expectancy in 2018 in individual countries:
 ↪**"))
display(countries['Expectancy2018'].dropna())
display(Markdown("**Summary statistics over all countries:**"))
display(countries['Expectancy2018'].describe())
```

Values of life expectancy in 2018 in individual countries:

```
Country
Afghanistan              64.486000
Albania                  78.458000
Algeria                  76.693000
Angola                   60.782000
Antigua and Barbuda      76.885000
                           …
Virgin Islands (U.S.)    79.568293
West Bank and Gaza       73.895000
Yemen, Rep.              66.096000
```

```
Zambia                     63.510000
Zimbabwe                   61.195000
Name: Expectancy2018, Length: 198, dtype: float64
```

Summary statistics over all countries:

```
count    198.000000
mean      72.719232
std        7.561327
min       52.805000
25%       67.623750
50%       74.097000
75%       78.200610
max       84.934146
Name: Expectancy2018, dtype: float64
```

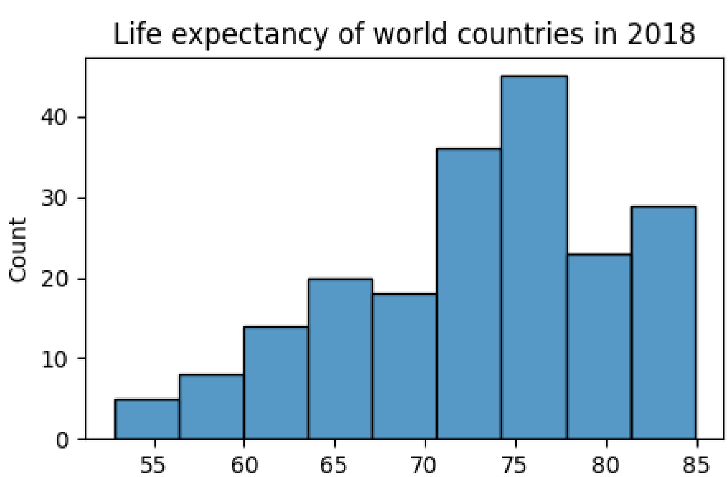
1.2 Histograms

Histograms are well known, and we have seen them in Lecture 03. We split the range of a variable into equally sized bins, count the number of data points in each bin and plot the counts as a bar graph.

Histograms allow us to observe many aspects of distribution of values of a variable:

- range of values, outliers
- central tendency
- unimodality / multimodality
- variance
- symmetry / skewness (šikmosť)

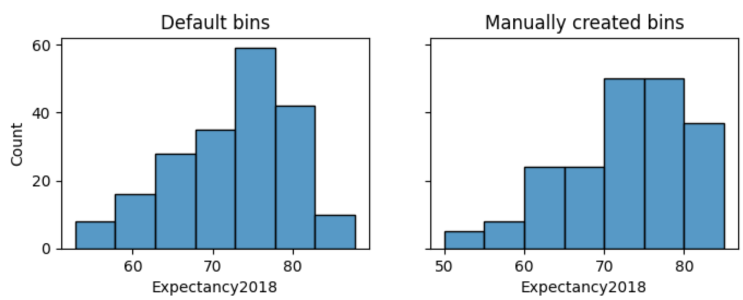
[5]:
```python
axes = sns.histplot(data=countries, x='Expectancy2018')
axes.set_title('Life expectancy of world countries in 2018')
axes.set_xlabel(None)
axes.figure.set_size_inches(5, 3)
pass
```



1.2.1 Custom bins

- Seaborn library makes bins by splitting the range into equally sized intervals, but perhaps a more meaningfull plot uses round values at bin boundaries, e.g. intervals of 5 years 50-55, 55-60, 60-65,...
- We can use manually created bin boundaries in Seaborn.
- Plotly library tries to create more meaningful bins by default.

[6]:
```
# create a figure with two plots
figure, axes = plt.subplots(1, 2, figsize=(8,2.5), sharey=True)

# the first plot has histogram with default bins of width 5
sns.histplot(data=countries, x='Expectancy2018', binwidth=5, ax=axes[0])
axes[0].set_title('Default bins')

# the second plot has manually set bin boundaries 50,55,60,...,85
sns.histplot(data=countries, x='Expectancy2018', bins=range(50,90,5),␣
↪ax=axes[1])
axes[1].set_title('Manually created bins')
pass
```



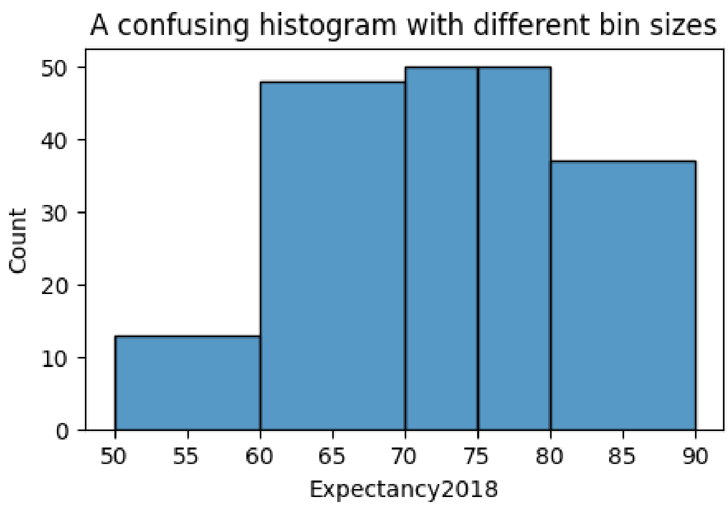
[7]:
```
# in Plotly, we specify the maximum number of bins, library may choose a lower␣
↪number
# to get "nice" bin boundaries
fig = px.histogram(countries, x="Expectancy2018", nbins=8, width=500,␣
↪height=350)
fig.show()
```

1.2.2 Use equally-sized bins

- Manually created bin boundaries can be arbitrary, but if bin width is unequal, the resulting plot is confusing.

- If you really need special bins (e.g. age <18 years, 18-65 years, >65 years), make a categorical variable, then plot it as a bar graph (typically displayed as bars with equal width, spaces between bars), clearly mark the meaning of each bar.

[8]:
```
axes = sns.histplot(data=countries, x='Expectancy2018', bins=[50,60,70,75,80,90])
axes.set_title('A confusing histogram with different bin sizes')
axes.figure.set_size_inches(5, 3)
pass
```



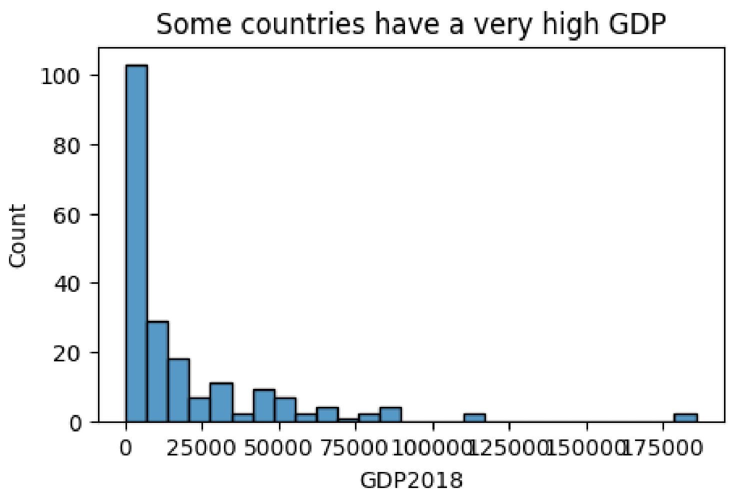
1.2.3 Removing outliers

- Histograms are great for spotting outliers.
- But extreme values reduce the space given to more regular values, so perhaps we want to remove them in subsequent analysis.

We have several options:

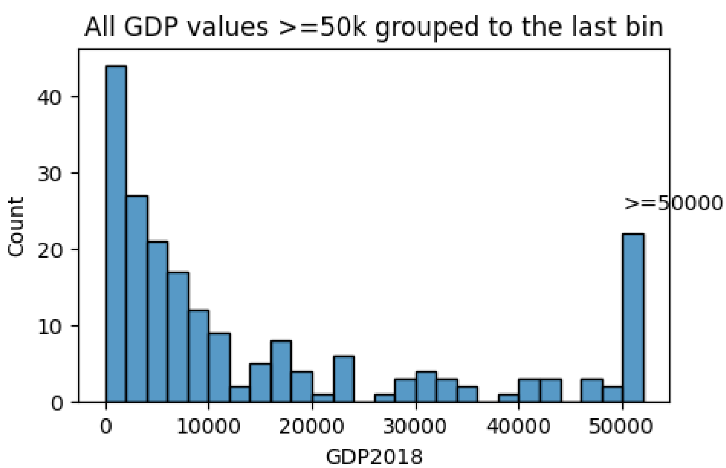
- Remove them from the dataset if we believe them to be errors.
- Or remove them from the plot only, e.g. by `set_xlim` or by using custom bins with a smaller range.
- Or clip values: replace values above some threshold with the threshold value (function `clip` in Pandas). Thus they will be present in the last bin. This bin should be then clearly marked.

[9]:
```
axes = sns.histplot(data=countries, x='GDP2018')
axes.set_title('Some countries have a very high GDP')
axes.figure.set_size_inches(5, 3)
pass
```



[10]:

```
# replace values larger than 51k with 51k
gdp_clipped = countries['GDP2018'].clip(0, 51000)
# make histogram with manual bins, with last bin 50k-52k
axes = sns.histplot(x=gdp_clipped, bins=np.arange(0,53000,2000))
axes.figure.set_size_inches(5, 3)
# mention clipping in plot title
axes.set_title('All GDP values >=50k grouped to the last bin')
# also add a text label to the bin with clipped values
axes.text(x=50000, y=25, s='>=50000')
pass
```



1.2.4 Problems with precision

When working with integers or even real numbers given with a small number of decimal points, we can get artifacts related to different number of possible values falling to different bins.

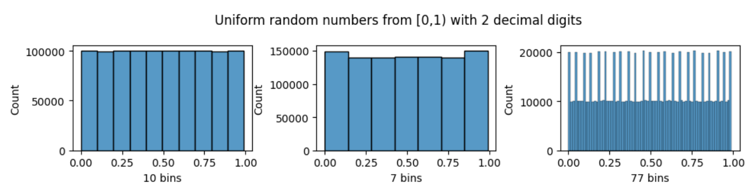
To illustrate this, we uniformly sample million points from the set $\{0, 0.01, 0.02, \ldots, 0.99\}$. * There will be a similar number of samples for each possible value from this set. * We show histogram with 7 or 77 equally sized bins. * For 10 bins, each bin summarizes 10 of the possible values and the sizes are approximately the same. * For 7 bins, the first and the last bin summarize 15 possible values each and remaining bins summarize 14 possible values each. The first and last bin are thus slightly higher. * For 77 bins, some bins summarize 2 different values, others only 1. We see clear differences in bar height.

If we are unaware of this, we may draw incorrect conclusions from the second and third plot.

[11]:
```
sample_uniform = np.random.randint(0, 100, 1000000) / 100
display(Markdown('**Example of data:**'), sample_uniform[0:5])
figure, axes = plt.subplots(1, 3, figsize=(10,2.5))
figure.tight_layout(pad=3)
for (i, bin) in enumerate([10, 7, 77]):
  sns.histplot(x=sample_uniform, bins=bin, ax=axes[i])
  axes[i].set_xlabel(f"{bin} bins")
figure.suptitle("Uniform random numbers from [0,1) with 2 decimal digits")
pass
```

Example of data:

```
array([0.21, 0.97, 0.12, 0.17, 0.2 ])
```

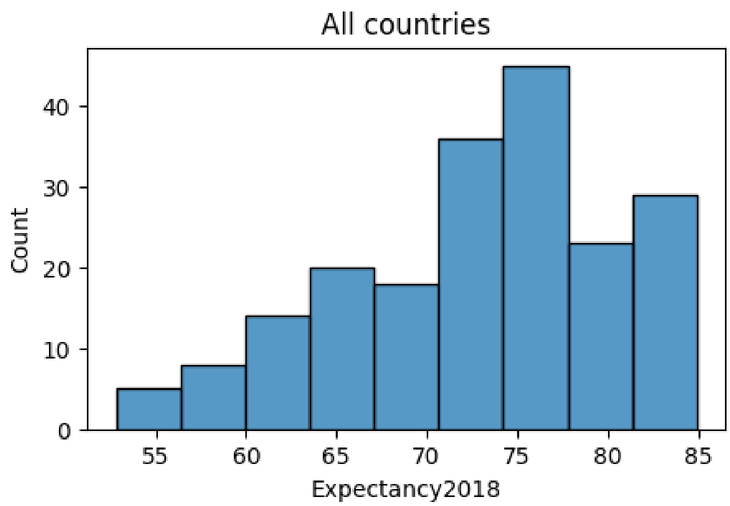


1.2.5 Small samples

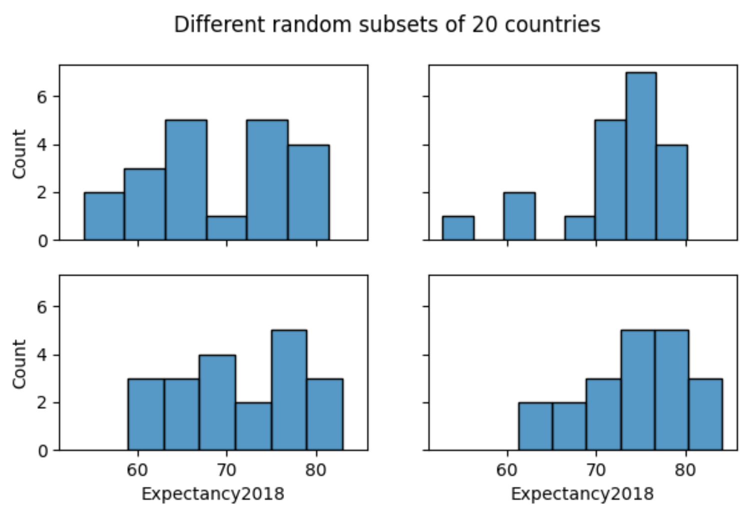
Beware drawing strong conclusions from small samples.

- Below we again first show the histogram for life expectancy over all countries.
- Then we show histograms for four random subsets of 20 countries.
- Any estimates (including histograms) from small samples are subject to random noise.

[12]:
```
axes = sns.histplot(data=countries, x='Expectancy2018')
axes.set_title('All countries')
axes.figure.set_size_inches(5, 3)
pass
```



[13]:
```
figure, axes = plt.subplots(2, 2, sharex=True, sharey=True, figsize=(7,4))
for row in axes:
  for subplot in row:
    expectancy_sample = countries['Expectancy2018'].dropna().sample(20)
    sns.histplot(x=expectancy_sample, ax=subplot)
figure.suptitle("Different random subsets of 20 countries")
pass
```



1.2.6 Summary: Histogram bin size

To summarize, the tricky part of using histograms is to choose the bin size or the number of bins.

Smaller bins mean more details are visible, but some of those details may be artefacts:

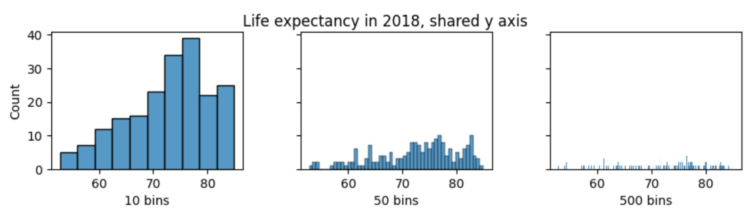
- random fluctuations due to small number of points in the bin, or
- effects related to insufficient resolution of the data.

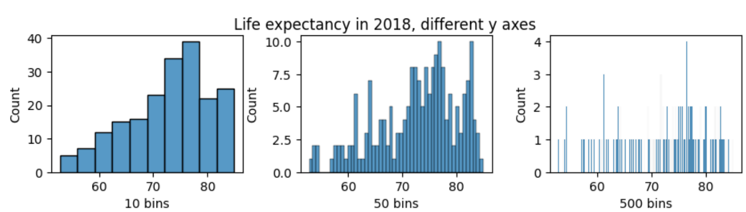
Thus choose bin size based on:

- the amount of data,
- the precision of input values,
- the meaningful resulution of the results.

In the example below we show the life expectancy data with different number of bins. Do 50 or 500 bins show more meaningful information than 10 bins?

[14]:
```
# one set of plots has share y axis, one has not
figure1, axes1 = plt.subplots(1, 3, figsize=(10,2), sharey=True)
figure2, axes2 = plt.subplots(1, 3, figsize=(10,2))
# plot the same plots for each set
for (figure, axes, title) in [(figure1, axes1, 'shared y axis'),
                              (figure2, axes2, 'different y axes')]:
  # iterate over different numbers of bins
  for (i, bin) in enumerate([10, 50, 500]):
    sns.histplot(data=countries, x='Expectancy2018', bins=bin, ax=axes[i])
    axes[i].set_xlabel(f"{bin} bins")
  # title of the whole figure
  figure.suptitle(f"Life expectancy in 2018, {title}")
  figure.subplots_adjust(wspace=0.3)
pass
```





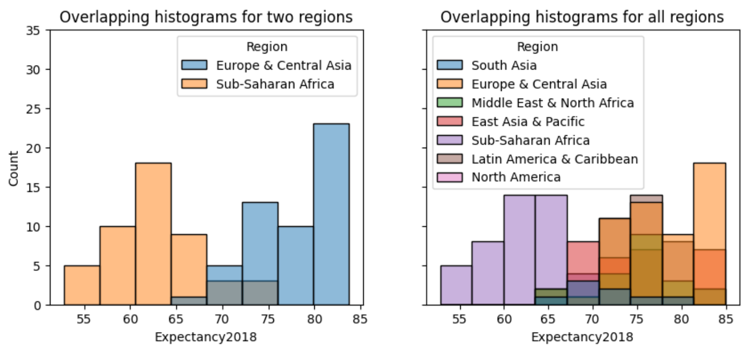
1.2.7 Comparing distributions with histograms

- We can compare distributions of a numerical variable split into groups by a categorical variable.

- For example in our `countries` table, we can compare life expectancy in different regions of the world.
- Seaborn provides several options for doing so.

The first two plots use semi-transparent overlapping histograms, which work well for two regions (left), but are a mess for many regions (right).

[15]:
```
# select two regions
countries_subset = countries.query('Region == "Europe & Central Asia" '
                                   + 'or Region == "Sub-Saharan Africa"')
# create figure with 2 plots
figures, axes = plt.subplots(1, 2, figsize=(10,4), sharey=True)
# plot both histograms
sns.histplot(data=countries_subset, x='Expectancy2018', hue='Region', ax =␣
 ↪axes[0])
sns.histplot(data=countries, x='Expectancy2018', hue='Region', ax=axes[1])
# make bigger y-axis to accomodate legend
axes[0].set_ylim(0,35)
# titles
axes[0].set_title('Overlapping histograms for two regions')
axes[1].set_title('Overlapping histograms for all regions')
pass
```

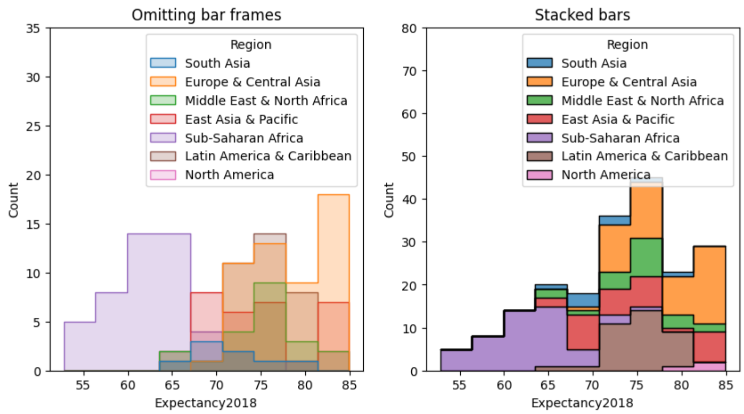


The next two plots attempt to improve the situation.

- On the left we omit bar outlines to simplify the plot.
- On the right we use stacked bars, showing contribution of each region to the whole.

[16]:
```
# create figure with 2 plots
figures, axes = plt.subplots(1, 2, figsize=(10, 5))
# plot both histograms
sns.histplot(data=countries, x='Expectancy2018', hue='Region',
             element='step', ax = axes[0])
sns.histplot(data=countries, x='Expectancy2018', hue='Region', element='step',
             multiple="stack", ax=axes[1])
# make bigger y-axis to accomodate legend
```

```
axes[0].set_ylim(0,35)
axes[1].set_ylim(0,80)
# titles
axes[0].set_title('Omitting bar frames')
axes[1].set_title('Stacked bars')
pass
```



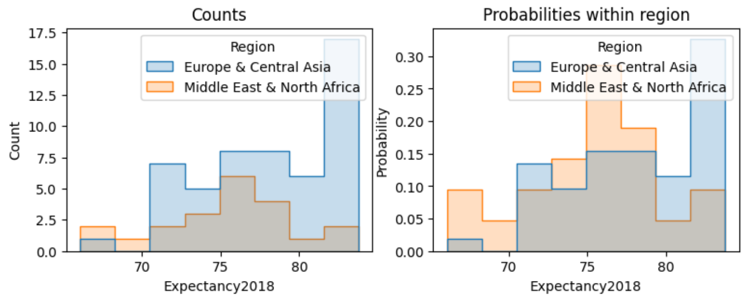
- Regions contain different number of countries.
- To better compare distribution of the expectancy within region, we should normalize the count to probabilities.
- Use `common_norm=False` to normalize each region separately.

[17]:
```
display(Markdown("**Countries in regions:**"))
display(countries.groupby('Region').size().sort_values())

# select two regions of very different sizes
countries_subset2 = countries.query('Region == "Europe & Central Asia" '
                                    + 'or Region == "Middle East & North␣
↪Africa"')
# plot counts and probabilities
figure, axes = plt.subplots(1, 2, figsize=(9,3))
sns.histplot(data=countries_subset2, x='Expectancy2018', hue='Region',␣
↪element='step', ax=axes[0])
sns.histplot(data=countries_subset2, x='Expectancy2018', hue='Region',␣
↪element='step',
             stat="probability", common_norm=False, ax=axes[1])
axes[0].set_title('Counts')
axes[1].set_title('Probabilities within region')
pass
```

Countries in regions:

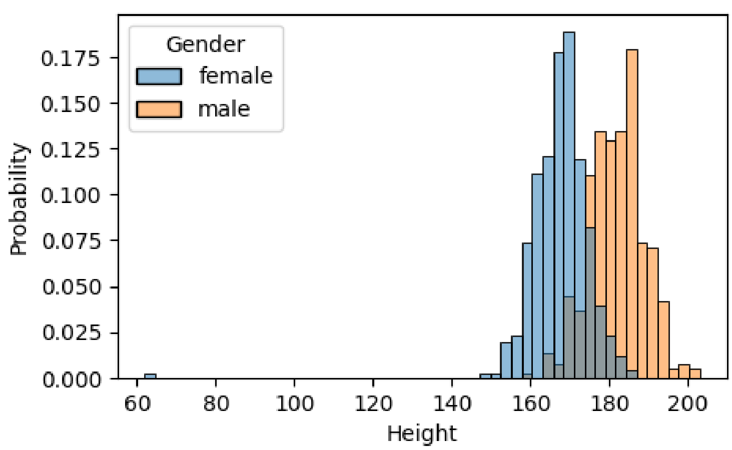
```
Region
```

```
North America                  3
South Asia                     8
Middle East & North Africa    21
East Asia & Pacific           37
Latin America & Caribbean     42
Sub-Saharan Africa            48
Europe & Central Asia         58
dtype: int64
```



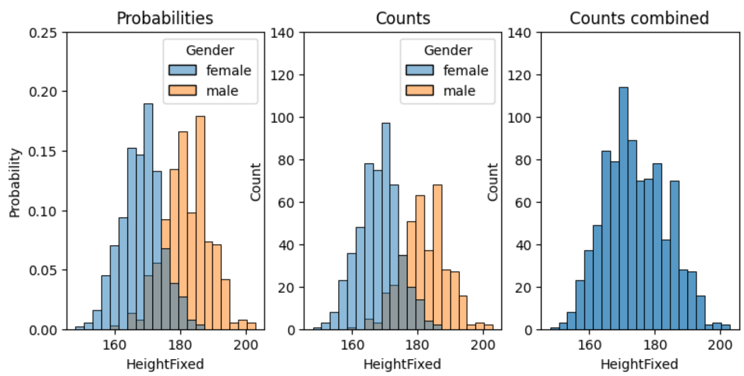
- Using FSEV survey, we can compare self-reported heights of women and men.
- We will use only values of adults above 18 years of age.
- Besides the expected trend, we also see a clear outlier, perhaps an error (although people of such heights exist, the cases are extremely rare).
- We will replace it with NaN.

[18]:
```
fsev.iloc[:,144].value_counts()
adults = fsev.query("Age >= 18").copy(deep=True)
axes = sns.histplot(data=adults, x='Height', hue='Gender',
                    stat="probability", common_norm=False)
axes.figure.set_size_inches(5, 3)
pass
```



Below we see histograms after removal of the extreme value.

```
[19]: # replace all values lower than 100cm by NaN, in a new column HeighFixed
adults['HeightFixed']=adults['Height'].mask(adults['Height']<100, np.nan)
figure, axes = plt.subplots(1, 3, figsize=(9,4))
sns.histplot(data = adults, x='HeightFixed', hue='Gender',
             stat="probability", common_norm=False, ax=axes[0])
sns.histplot(data = adults, x='HeightFixed', hue='Gender', ax=axes[1])
sns.histplot(data = adults, x='HeightFixed', ax=axes[2])
axes[0].set_ylim(0,0.25)
axes[1].set_ylim(0,140)
axes[2].set_ylim(0,140)
axes[0].set_title('Probabilities')
axes[1].set_title('Counts')
axes[2].set_title('Counts combined')
pass
```



1.3 Probability distributions

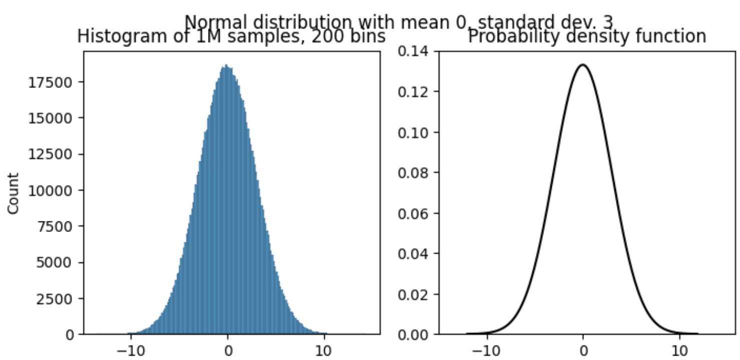
- For a continuous variable, we can imagine having infinitely many data points and making histogram with infinitely small bins, keeping the area under the histogram equal to one.
- Thus we obtain **probability density function** (PDF) (hustota rozdelenia pravdepodobnosti).
- We often assume that our data are a small sample from one of the well-characterized probability distributions (rozdelenie pravdepodnosti).

1.3.1 Normal (Gaussian) distribution

- The normal (or Gaussian) distirbution has two parameters: mean μ and standard deviation σ.
- Its density $f(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$.
- This is the well-known bell shape.

- Below we plot both histogram of a million samples from this distribution and the density given by the function above.
- These two plots are very similar.

[20]:
```
figure, axes = plt.subplots(1, 2, sharex=True, figsize=(8, 3.5))

# sample million points from the normal distrib. with mean 0 and std. dev. 3
sample_normal = np.random.normal(0, 3, 1000000)
# create histogram of the sampled points
sns.histplot(x=sample_normal, bins=200, ax=axes[0])
axes[0].set_title('Histogram of 1M samples, 200 bins')

# create an object representing normal distrib. with mean 0 and std. dev. 3
normal = scipy.stats.norm(0, 3)
# create equally-spaced points
x = np.arange(-12, 12, 0.1)
# compute values of pdf in these points
y = normal.pdf(x)
# plot the function
axes[1].plot(x, y, 'k-')
axes[1].set_title('Probability density function')
axes[1].set_ylim(0, 0.14)

figure.suptitle("Normal distribution with mean 0, standard dev. 3")
pass
```



- Normal distribution often arises in situations where a variable is a result of many small influences.
- One example is the height of a person within one gender and population.
- Below we fit the normal distribution to the histogram of the adult male heights from the FSEV survey.

[21]:
```
# select male height, drop missing values
male_heights = adults.query("Gender=='male'")['Height'].dropna()
# compute the characteristics (means, stdev)
```

```python
display(Markdown("**Mean male height:**"),
        male_heights.mean(),
        Markdown("**Std. dev. male height:**"),
        male_heights.std())

# compute the best fit
parameters = scipy.stats.norm.fit(male_heights)
display(Markdown("**Best fit:**"), parameters)

# get function values for regularly distributed x values
x = np.arange(150, 200, 1)
pdf_fitted = scipy.stats.norm.pdf(x, loc=parameters[0], scale=parameters[1])

# plot histogram, normalized as density (area=1)
figure, axes = plt.subplots(figsize=(5,3))
sns.histplot(x=male_heights, stat='density', ax=axes)
# add a line for fitted density
axes.plot(x, pdf_fitted, 'k-')
axes.set_title('Male heights with normal distribution fit')
pass
```

Mean male height:

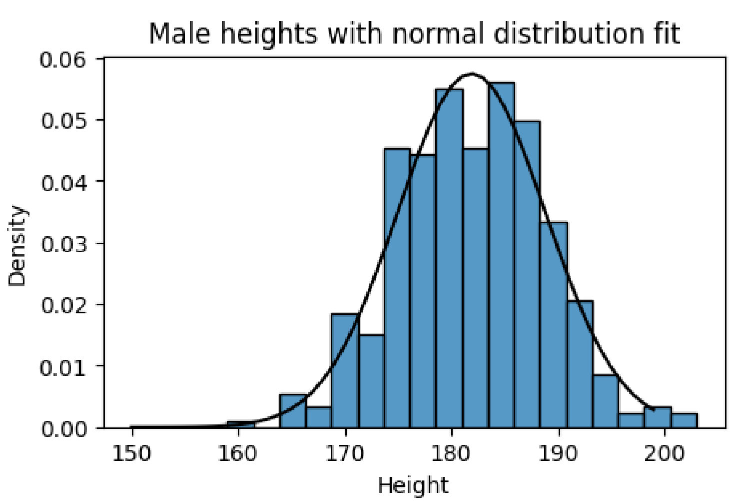
181.91820580474933

Std. dev. male height:

6.957251247475206

Best fit:

(181.91820580474933, 6.948066753375318)



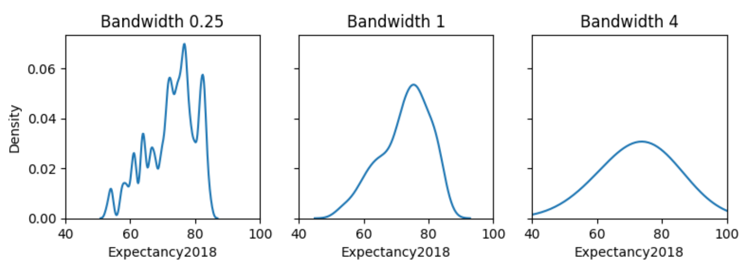
1.4 Kernel Density Estimation (KDE)

- KDE creates a smoothed version of a histogram.
- We choose a kernel function. e.g. the normal distribution.
- For each point in the dataset, the method creates a “kernel” centered at that point.
- It then adds up the heights of all kernel copies.
- The amount of smoothing is controled by the bandwidth parameter (standard deviation for the normal distribution).
- More information is in the scikit-learn documentation.

https://commons.wikimedia.org/wiki/File:Comparison_of_1D_histogram_and_KDE.png Drleft at English Wikipedia, CC BY-SA 3.0

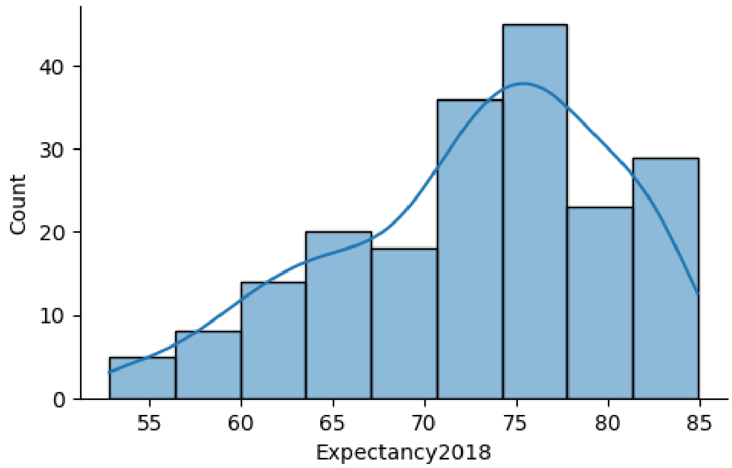
- KDE can be conveniently computed directly in Seaborn’s displot/kdeplot function.
- The bandwidth is adjusted by `bw_adjust`, with default 1.
- A small bandwidth leads to a bumpy plot not representing real trends.
- A large badwidth can obscure real trends.

[22]:
```
figure, axes = plt.subplots(1, 3, sharex=True, sharey=True, figsize=(9,2.5))
for axes, bandwidth in [(axes[0], 0.25), (axes[1], 1), (axes[2], 4)]:
  sns.kdeplot(x=countries["Expectancy2018"], ax=axes, bw_adjust=bandwidth)
  axes.set_title(f'Bandwidth {bandwidth}')
  axes.set_xlim(40,100)
pass
```



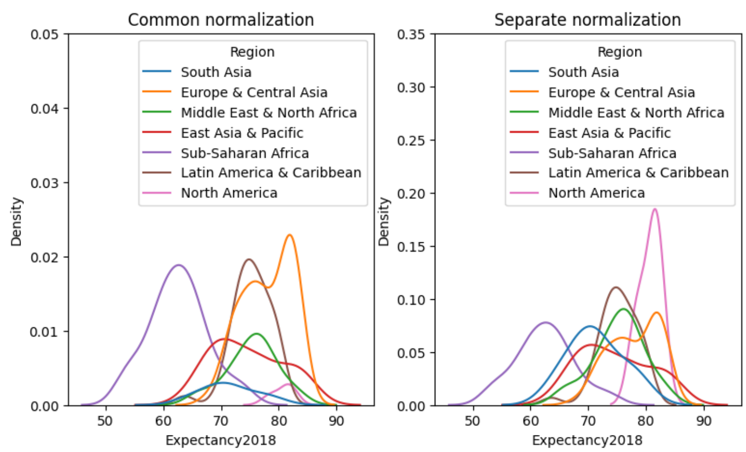
We can display combined histogram and KDE.

[23]:
```
axes = sns.displot(countries, x="Expectancy2018", kde=True)
axes.figure.set_size_inches(5, 3)
pass
```



KDE plots can be also better for comparing multiple distributions, as their smooth curves are easier to follow than histograms.

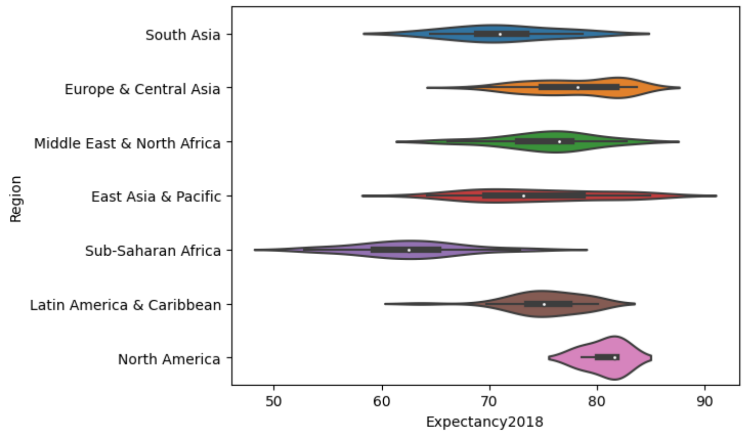
[24]:
```
figure, axes = plt.subplots(1, 2, sharex=True, figsize=(9,5))
sns.kdeplot(x=countries["Expectancy2018"], hue=countries["Region"], ax=axes[0])
axes[0].set_ylim(0, 0.05)
axes[0].set_title('Common normalization')
sns.kdeplot(x=countries["Expectancy2018"], hue=countries["Region"],
            common_norm=False, ax=axes[1])
axes[1].set_title('Separate normalization')
axes[1].set_ylim(0, 0.35)
pass
```



1.5 Violin plots

- Violin plots are often used instead of boxplots to compare distributions for different values of a categorical variable.
- Each violin consist of two symmetric KDE plots.
- They can be accompanied by a boxplot or strip plot.
- More variants can be found in the Seaborn tutorial.

[25]:
```
sns.violinplot(data=countries, y="Region", x="Expectancy2018")
pass
```



1.6 Cumulative distribution function

For a probability density function $f(x)$:

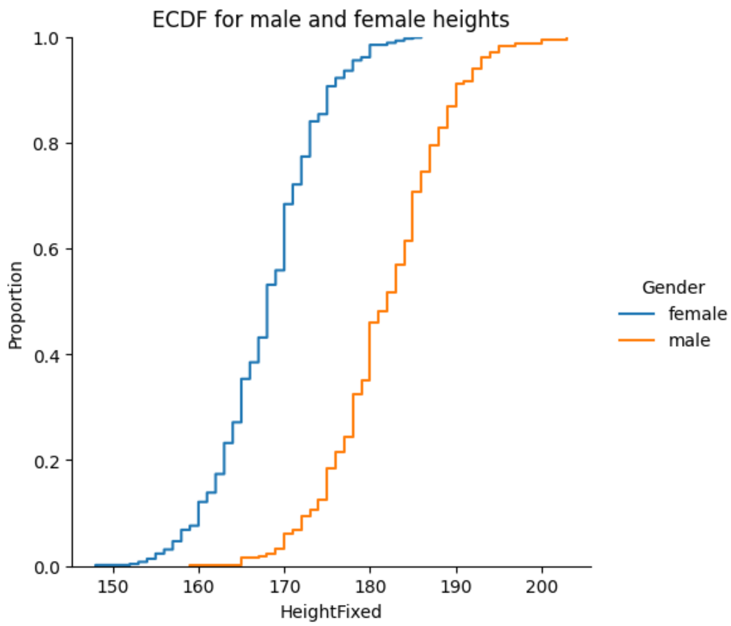
- Its cumulative distribution function (CDF) (distribučná funkcia) is the area under the curve from left up to point x.
- $F(x) = \int_{-\infty}^{x} f(t)\,dt$.
- CDF is non-decreasing.
- $\lim_{x\to-\infty} F(x) = 0$ and $\lim_{x\to\infty} F(x) = 1$.
- $F(x)$ is the probability that the random point from the distribution is $\leq x$.

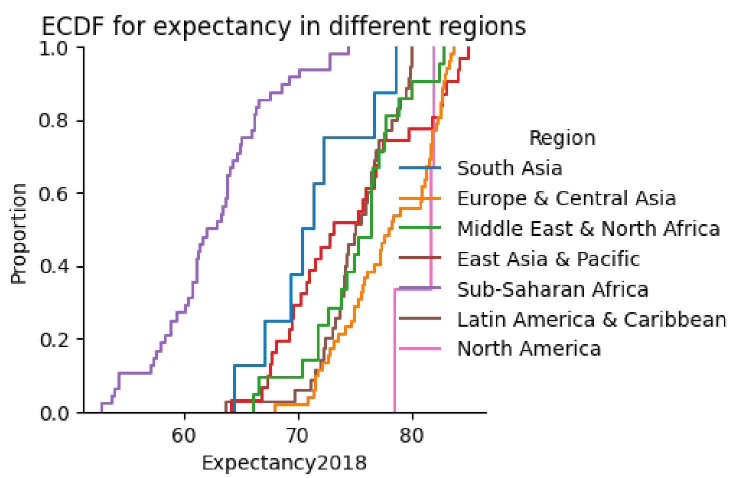
https://commons.wikimedia.org/wiki/File:Normal_Distribution_CDF.svg Inductiveload, Public domain

1.6.1 Empirical cumulative distribution function (ECDF)

- This is a similar concept for a finite sample.
- For each x, $F(x)$ is the fraction of the sample which is $\leq x$.
- This gives us a step-wise function which can be visualized.
- Unlike histograms and KDE, no parameters need to be set (bins, bandwidth).
- Allows comparison of multiple distributions and their quantiles (how?).
- But harder to interpret than histogram in terms of shape.

[26]:
```
grid = sns.displot(adults, x="HeightFixed", hue="Gender", kind="ecdf")
grid.axes[0,0].set_title('ECDF for male and female heights')
grid = sns.displot(countries, x="Expectancy2018", hue="Region", kind="ecdf")
grid.axes[0,0].set_title('ECDF for expectancy in different regions')
grid.figure.set_size_inches(5, 3)
pass
```



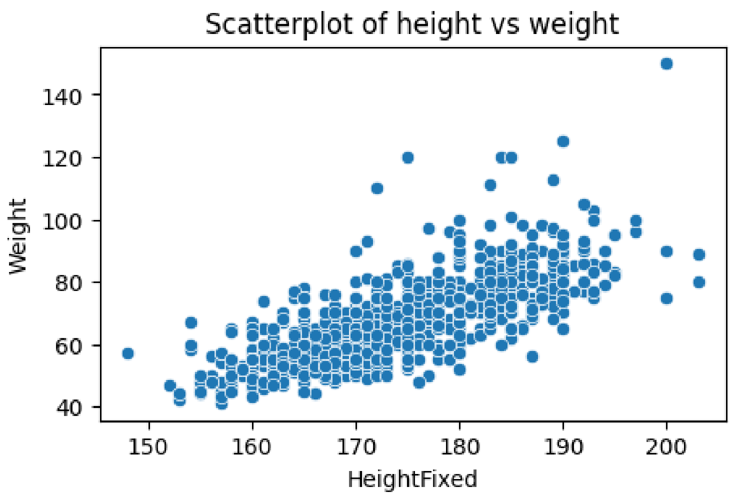


1.7 Two-dimensional histograms / KDE

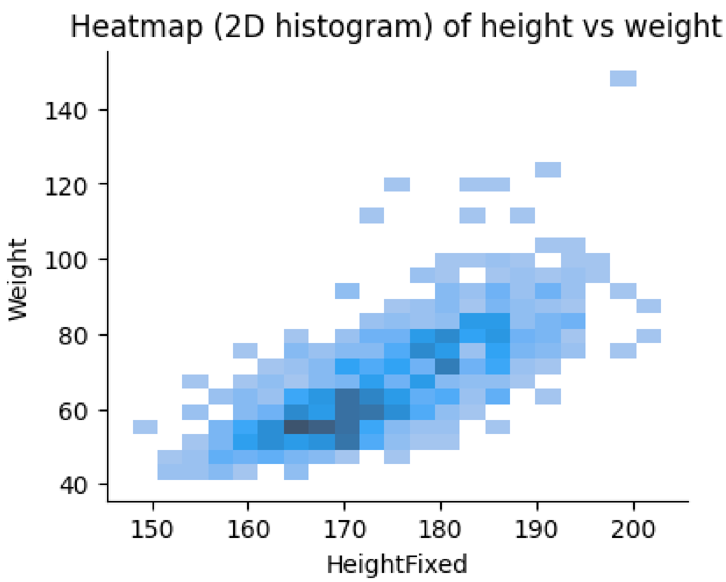
- Instead of scatterplots we can compute 2D histograms or smooth them by KDE.

- This works well if we have many overlapping points.
- Below we show several variants for the plot height vs. weight of adults in the FSEV survey (with the height outlier removed).
- In scatterplot there is a cloud of overlapping points and it is not clear which parts of it are denser (this can be somewhat improved with smaller points, but some values may repeat).

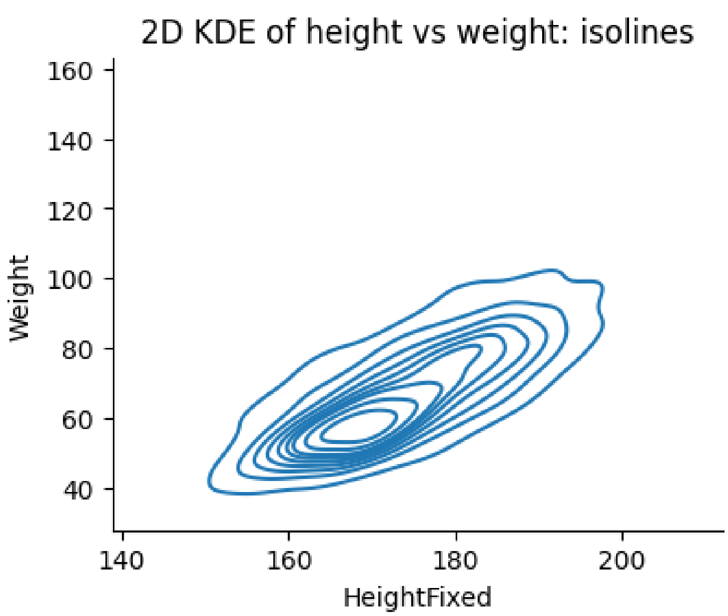
[27]:
```
axes = sns.scatterplot(data=adults, x='HeightFixed', y='Weight')
axes.figure.set_size_inches(5, 3)
axes.set_title('Scatterplot of height vs weight')
pass
```



[28]:
```
grid = sns.displot(data=adults, x='HeightFixed', y='Weight')
grid.figure.set_size_inches(4, 3)
grid.ax.set_title('Heatmap (2D histogram) of height vs weight')
pass
```

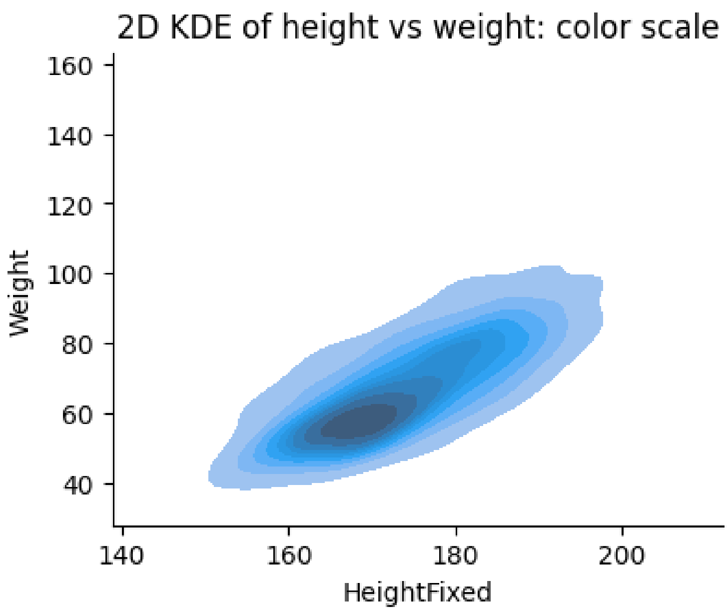


[29]:
```
grid = sns.displot(data=adults, x='HeightFixed', y='Weight', kind="kde")
grid.figure.set_size_inches(4, 3)
grid.ax.set_title('2D KDE of height vs weight: isolines')
pass
```

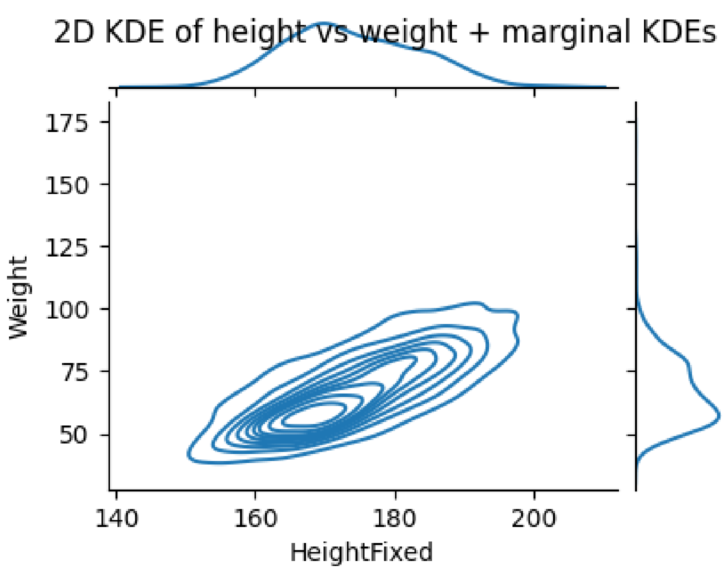


[30]:
```
grid = sns.displot(data=adults, x='HeightFixed', y='Weight', kind="kde",␣
  ↪fill=True)
```

```python
grid.figure.set_size_inches(4, 3)
grid.ax.set_title('2D KDE of height vs weight: color scale')
pass
```



[31]:
```python
grid = sns.jointplot(data=adults, x='HeightFixed', y='Weight', kind="kde")
grid.figure.set_size_inches(4, 3)
grid.figure.suptitle('2D KDE of height vs weight + marginal KDEs')
pass
```



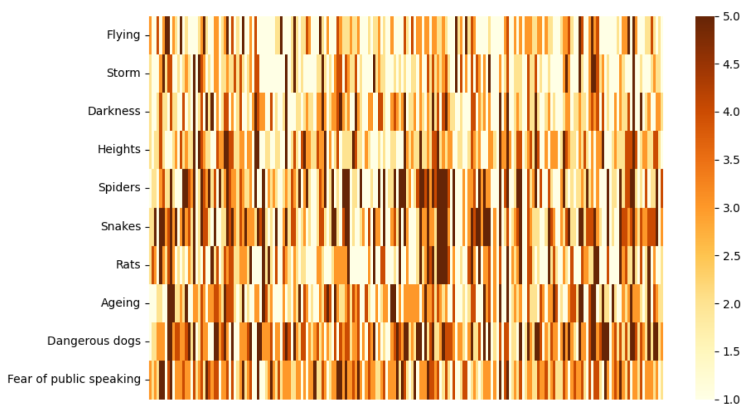
1.8 Clustering multi-dimensional data

- The FSEV survey contains questions about phobias and fears, each with answers 1-5 (5 means highest fear).
- We will first randomly select 200 participants without missing values.
- We will display them as a heatmap.

[32]:
```
# columns 63-72 are fears, drop rows with missing values, sample 200 people
fsev_sample = fsev.iloc[:, 63:73].dropna().sample(200)
# show sample of the data
display(fsev_sample.head())
```

```
     Flying  Storm  Darkness  Heights  Spiders  Snakes  Rats  Ageing  \
5       3.0    2.0       2.0      2.0      1.0       2   2.0     1.0
784     1.0    1.0       1.0      1.0      2.0       2   4.0     1.0
373     1.0    1.0       1.0      2.0      2.0       5   3.0     1.0
606     4.0    1.0       2.0      2.0      1.0       1   1.0     2.0
474     1.0    3.0       4.0      4.0      3.0       5   5.0     2.0

     Dangerous dogs  Fear of public speaking
5               1.0                      3.0
784             2.0                      2.0
373             2.0                      3.0
606             3.0                      2.0
474             3.0                      5.0
```

[33]:
```
figure, axes = plt.subplots(figsize=(10,6))
sns.heatmap(fsev_sample.transpose(), xticklabels=False, ax=axes, cmap="YlOrBr")
pass
```



- Heatmap does not show clear trends, but we see that some phobias have higher values than others.

- We display this more explicitly by sorted means.
- Then we "standardize" values for individual phobias by subtracting the mean and dividing by the standard deviation.
- The resulting table have mean 0 and standard deviation 1 for each phobia.
- People with positive scores fear that subject more than average, with negative scores less than average.

[34]:
```
display(Markdown("**Phobias sorted by mean score in the survey:**"))
display(fsev_sample.mean().sort_values())
```

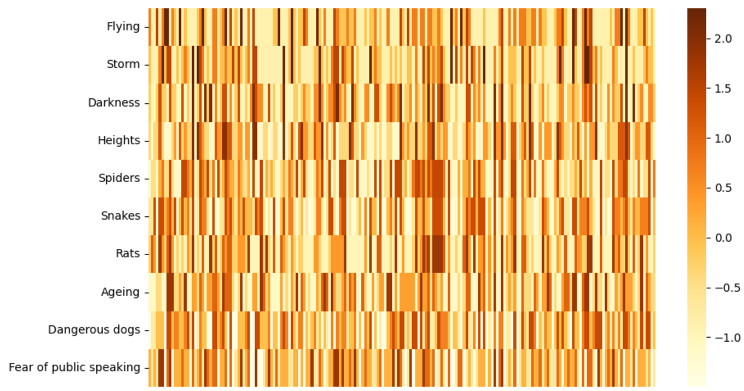
Phobias sorted by mean score in the survey:

```
Storm                     2.080
Flying                    2.095
Darkness                  2.290
Rats                      2.405
Heights                   2.485
Ageing                    2.565
Spiders                   2.740
Fear of public speaking   2.825
Snakes                    2.945
Dangerous dogs            3.055
dtype: float64
```

[42]:
```
fsev_sample_standardized = (fsev_sample - fsev_sample.mean()) / fsev_sample.
 ↪std()
display(fsev_sample_standardized.head())
```

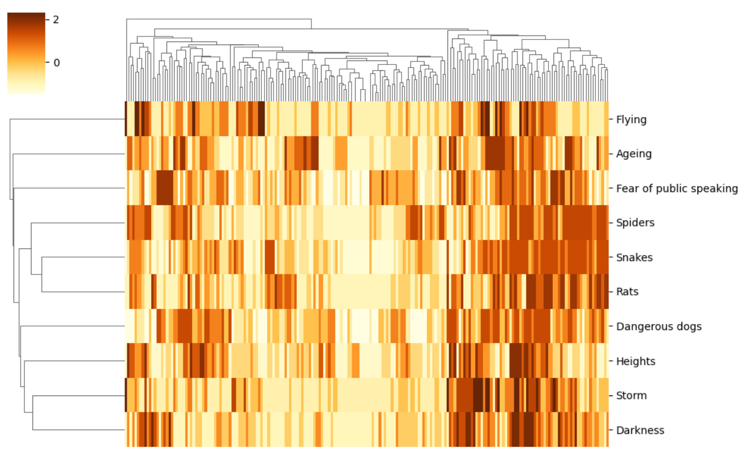
```
       Flying     Storm  Darkness   Heights   Spiders    Snakes      Rats  \
5    0.716823 -0.062821 -0.226168 -0.381303 -1.112867 -0.622642 -0.279378
784 -0.867316 -0.848081 -1.006058 -1.167496 -0.473288 -0.622642  1.100266
373 -0.867316 -0.848081 -1.006058 -0.381303 -0.473288  1.353999  0.410444
606  1.508893 -0.848081 -0.226168 -0.381303 -1.112867 -1.281522 -0.969200
474 -0.867316  0.722439  1.333611  1.191082  0.166290  1.353999  1.790088

       Ageing  Dangerous dogs  Fear of public speaking
5   -1.136792       -1.463446                 0.143762
784 -1.136792       -0.751307                -0.677737
373 -1.136792       -0.751307                 0.143762
606 -0.410407       -0.039168                -0.677737
474 -0.410407       -0.039168                 1.786762
```

[36]:
```
figure,axes = plt.subplots(figsize=(10,6))
sns.heatmap(fsev_sample_standardized.transpose(), xticklabels=False,␣
 ↪cmap="YlOrBr")
pass
```



- Heatmap now does not show much.
- Bellow we reorder its rows and columns by clustering (zhlukovanie), which puts similar rows and similar columns together.
- The degree of similarity is reflected also in the trees (recall last lecture about hierarchies).
- Some areas of dark and light colors now appear.

[37]:
```
sns.clustermap(fsev_sample_standardized.transpose(), xticklabels=False,␣
  ↪figsize=(10,6), cmap="YlOrBr")
pass
```



1.9 Dimensionality reduction

Methods for dimensionality reduction project high-dimensional data into lower-dimensional (usually 2D) space, while trying to preserve some structure in the original data.

- **Principal component analysis** (PCA) uses a linear projection: each new dimension is a linear combination (weighted sum) of the original dimensions. Weights are chosen to maximize variance.

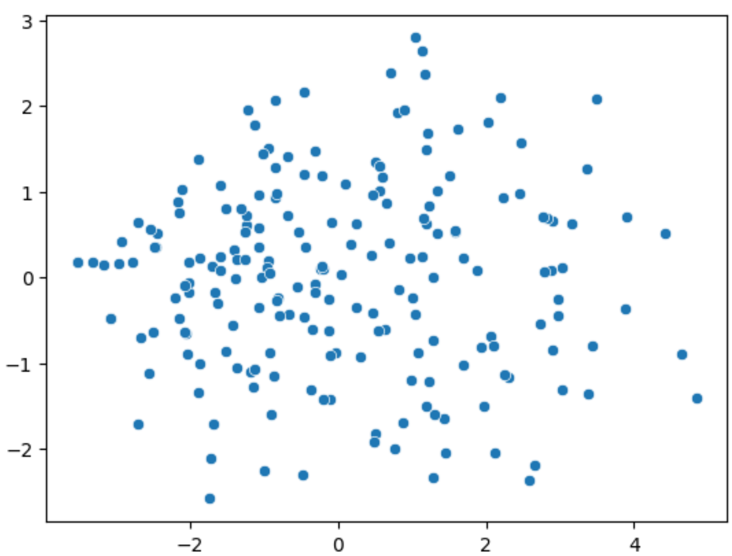
Some methods do not use linear projections, but try to preserve distances between points, for example: * **Multidimensional scaling** (MDS), * T-distributed Stochastic Neighbor Embedding (t-SNE).

We will use methods from the scikit-learn library for machine learning in Python.

[38]:
```python
from sklearn.decomposition import PCA
# compute PCA of our standardized data with 2 dimensions
fsev_pca = PCA(n_components=2).fit_transform(fsev_sample_standardized)
display(Markdown("**PCA transformed values** (first five lines):"))
display(fsev_pca[0:5, :])
axes = sns.scatterplot(x=fsev_pca[:, 0], y=fsev_pca[:, 1])
```

PCA transformed values (first five lines):

```
array([[-1.50521272, -0.85926961],
       [-1.88517477,  1.38373228],
       [-0.93622009,  1.50521849],
       [-1.68507734, -1.7044743 ],
       [ 2.45769085,  0.9727384 ]])
```

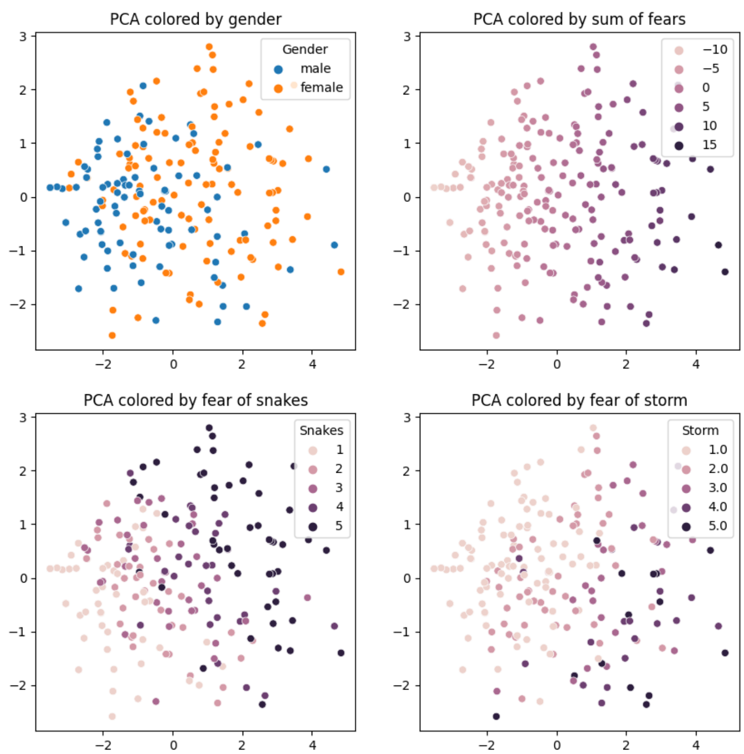


Scatterplot starts to be interesting if we can color points by some known factors.

- Below we see that men and women are quite mixed but women are shifted to the left.
- It seems that the first dimension strongly correlates with the overall fearfuless of a person.
- Fears of snakes and storms are strongly related to the overall fearfulness, but they also have trends along the y-axis.

[39]:
```python
# all columns for our sample to select gender and all data
fsev_sample_all = fsev.loc[fsev_sample.index,]
figure, axes = plt.subplots(2, 2, figsize=(10,10))
```

```
sns.scatterplot(x=fsev_pca[:, 0], y=fsev_pca[:, 1],
                hue=fsev_sample_all['Gender'], ax=axes[0,0])
axes[0,0].set_title('PCA colored by gender')
sns.scatterplot(x=fsev_pca[:, 0], y=fsev_pca[:, 1],
                hue=fsev_sample_standardized.sum(axis=1), ax=axes[0,1])
axes[0,1].set_title('PCA colored by sum of fears')
sns.scatterplot(x=fsev_pca[:, 0], y=fsev_pca[:, 1],
                hue=fsev_sample_all['Snakes'], ax=axes[1,0])
axes[1,0].set_title('PCA colored by fear of snakes')
sns.scatterplot(x=fsev_pca[:, 0], y=fsev_pca[:, 1],
                hue=fsev_sample_all['Storm'], ax=axes[1,1])
axes[1,1].set_title('PCA colored by fear of storm')
pass
```

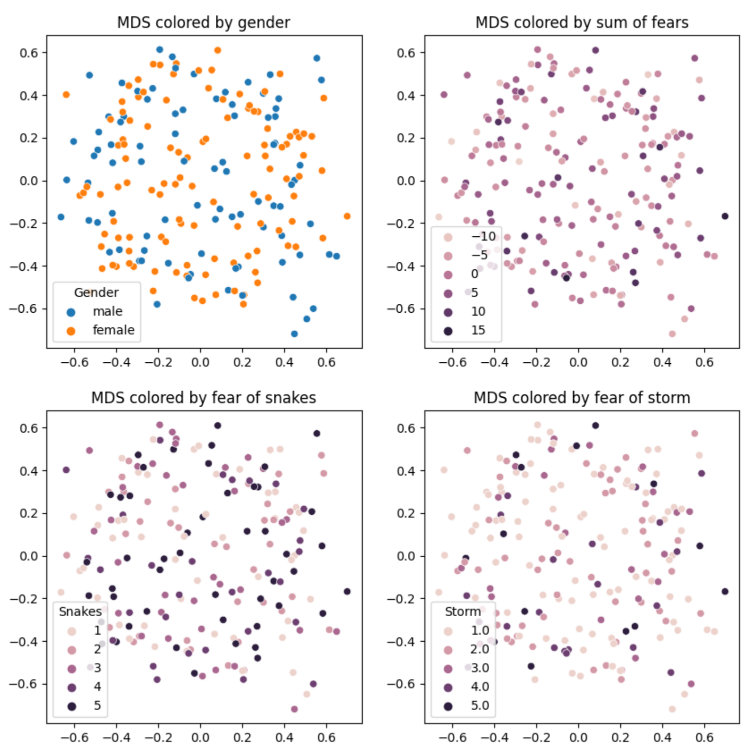


Unfortunately, MDS does not yield useful visualizations in for this input; perhaps these distances cannot be well preserved in 2D.

[40]:
```
from sklearn.manifold import MDS
fsev_mds = MDS(n_components=2, metric=False).
 ↪fit_transform(fsev_sample_standardized)
figure, axes = plt.subplots(2, 2, figsize=(10,10))
sns.scatterplot(x=fsev_mds[:, 0], y=fsev_mds[:, 1],
                hue=fsev_sample_all['Gender'], ax=axes[0,0])
```

```
axes[0,0].set_title('MDS colored by gender')
sns.scatterplot(x=fsev_mds[:, 0], y=fsev_mds[:, 1],
                hue=fsev_sample_standardized.sum(axis=1), ax=axes[0,1])
axes[0,1].set_title('MDS colored by sum of fears')
sns.scatterplot(x=fsev_mds[:, 0], y=fsev_mds[:, 1],
                hue=fsev_sample_all['Snakes'], ax=axes[1,0])
axes[1,0].set_title('MDS colored by fear of snakes')
sns.scatterplot(x=fsev_mds[:, 0], y=fsev_mds[:, 1],
                hue=fsev_sample_all['Storm'], ax=axes[1,1])
axes[1,1].set_title('MDS colored by fear of storm')
pass
```

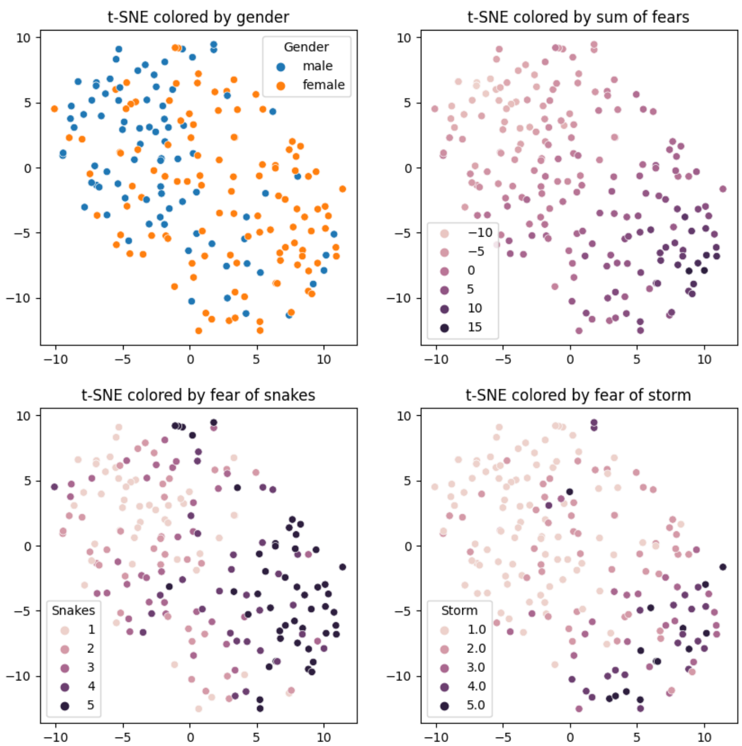
```
/home/bbrejova/viz/notebooks/venv/lib/python3.8/site-
packages/sklearn/manifold/_mds.py:299: FutureWarning:

The default value of `normalized_stress` will change to `'auto'` in version 1.4.
To suppress this warning, manually set the value of `normalized_stress`.
```



In contrast, t-SNE seems to work fine.

[41]:
```
from sklearn.manifold import TSNE
fsev_tsne = TSNE(n_components=2).fit_transform(fsev_sample_standardized)
figure, axes = plt.subplots(2,2,figsize=(10,10))
```

```
sns.scatterplot(x=fsev_tsne[:, 0], y=fsev_tsne[:, 1],
                hue=fsev_sample_all['Gender'], ax=axes[0,0])
axes[0,0].set_title('t-SNE colored by gender')
sns.scatterplot(x=fsev_tsne[:, 0], y=fsev_tsne[:, 1],
                hue=fsev_sample_standardized.sum(axis=1), ax=axes[0,1])
axes[0,1].set_title('t-SNE colored by sum of fears')
sns.scatterplot(x=fsev_tsne[:, 0], y=fsev_tsne[:, 1],
                hue=fsev_sample_all['Snakes'], ax=axes[1,0])
axes[1,0].set_title('t-SNE colored by fear of snakes')
sns.scatterplot(x=fsev_tsne[:, 0], y=fsev_tsne[:, 1],
                hue=fsev_sample_all['Storm'], ax=axes[1,1])
axes[1,1].set_title('t-SNE colored by fear of storm')
pass
```



1.10 Conclusion and other courses

We have briefly covered several statistical concepts often used in visualization:

- histogram,
- kernel density estimation,
- empirical cumulative distribution function,
- clustering,
- dimensionality reduction.

You will learn more in the next years of your study: * Fundamentals of Probability and Statistics, 2W * Principles of Data Science 3W * Linear Algebra this semester