**Lesson 8, Pandas dataframes**

# Make sure everyone has

# Gapminder in ./data folder

# https://swcarpentry.github.io/python-novice-gapminder/ (python-novice-gapminder-data.zip)

# Kohela in ./data folder

# https://www.biodsc.nl/workshop-materials/

# Pandas dataframe

# Collection of "series"

# one series is one column

# on which you can perform operations

# Built on "numpy",

# a library for working with arrays and mathematical operations

# series very similar to numpy.array

# Advantages

# Access to individual records

# Convenient ways combining information multiple dataframes

# how to get dataframe

import pandas as pd

# data = pd.read\_csv('../data/gapminder\_gdp\_europe.csv', index\_col='country')

data = pd.read\_csv('/Users/m.wehrens/Data\_UVA/2024\_teaching/2025-03-gapminder/data/gapminder\_gdp\_europe.csv', \

index\_col='country')

print(data.head())

# TWO METHODS TO ACCESS DATA

# iloc

# loc

# "iloc" method

# numerical indexing of rows and columns

# remember, it's a 2D table

# indexing similar as accessing chars in string

print(data.iloc[0, 0])

# "loc" method

# acess entries by their label

# Other synthax

# ":" means all columns, or all rows

# print(data.loc["Albania", :])

# print(data.loc["Albania"])

# in loc, first entry is row

# other way around

# print(data.loc[:, "gdpPercap\_1952"])

# More ways to access data

# print(data["gdpPercap\_1952"])

# no "loc"!!!

# <--> series

# also works:

# data.gdpPercap\_1992

# not recommended, confusion with methods

# slicing works on labels as well

print(data.loc['Italy':'Poland', 'gdpPercap\_1962':'gdpPercap\_1972'])

# can be used to perform operations on subsets

print(data.loc['Italy':'Poland', 'gdpPercap\_1962':'gdpPercap\_1972'].max())

# note: applied per column

# Selecting data

# Use a subset of data to keep output readable.

subset = data.loc['Italy':'Poland', 'gdpPercap\_1962':'gdpPercap\_1972']

print('Subset of data:\n', subset)

# notice the escape "\"; \n is signifier for nextline

# Which values were greater than 10000 ?

print('\nWhere are values large?\n', subset > 10000)

# comparisons are done element-wise

# a frame full of "booleans" (true/false)

# sometimes called mask

# (MW: ±filter)

mask = subset > 10000

print(mask)

print(subset[mask])

# returns value where True

# returns NaN where False

# NaN = Not a number

# NaNs are ignored in operations

# e.g. min, max, etc

# e.g.

print(subset[subset > 10000].describe())

# Group By: split-apply-combine

# Group by: split-apply-combine

# note to self: see related/202503\_RNAseq-data.py

# Goal

# table in which rows match condition

# calculate something per condition

# use group\_by

# Using data from Kohela et al.

# RNA-seq data (see powerpoint)

import pandas as pd

# Import data

df\_cells\_kohela = pd.read\_csv('/Users/m.wehrens/Data\_UVA/example-datasets/kohela-et-al/kohela-et-al.csv', index\_col=0)

df\_cells\_kohela2 = df\_cells\_kohela.T

df\_cells\_kohela2.head()

# create new 'masks'

epicardial\_cells = df\_cells\_kohela2['WT1']>3

fibroblast\_cells = df\_cells\_kohela2['COL2A1']>30

fat\_cells = df\_cells\_kohela2['PPARG']>2

# Create a new column

df\_cells\_kohela2['Celltype'] = 'unknown'

# Set values for the column

df\_cells\_kohela2.loc[epicardial\_cells,'Celltype'] = 'epicardial'

df\_cells\_kohela2.loc[fibroblast\_cells, 'Celltype'] = 'fibroblast'

df\_cells\_kohela2.loc[fat\_cells, 'Celltype'] = 'fat'

# OPTIONAL: give an overview of the frequencies of 'Celltype' values

df\_cells\_kohela2['Celltype'].value\_counts()

# now use group\_by to calculate gene expression median values per group

# explain:

# creates grouped object (groupby object)

# now any aggregate function

# e.g. aggregate('XX'), max(), min(), etc

# applied per group

# will paste together the dataframe

df\_cells\_kohela2\_groupedType = df\_cells\_kohela2.groupby('Celltype')

# Now calculate mean expression

df\_results = df\_cells\_kohela2\_groupedType.mean()

print(df\_results.head())

print(df\_cells\_kohela2.columns[0])

print(df\_cells\_kohela2.columns[-2])

df\_means = df\_cells\_kohela2.loc[:,'A1BG':'ZZZ3'].mean()

# And normalize

df\_results\_normalized = df\_results/df\_means

# "Check" their TFAP2A claim

print(df\_results\_normalized.loc[:,'TFAP2A'])

# Check some other highly differentially expressed genes

print(df\_results\_normalized.loc[:,(df\_results\_normalized>10).any()])