**Lesson 9, Plotting**

# Plotting

# matplotlib most widely used

# or sub library "pyplot"

# will render inline by default

import matplotlib.pyplot as plt

# simple plot

time = [0, 1, 2, 3]

position = [0, 100, 200, 300]

plt.plot(time, position)

plt.xlabel('Time (hr)')

plt.ylabel('Position (km)')

# when running python differently

# e.g. terminal

# also need "plt.show()"

# and also plt.close()

# Seaborn

# Can also use "seaborn"

# Based on matplotlib

# Usually used together

# Additional features geared towards dfs

# How to install seaborn in Jupyter

%pip install seaborn

# only needed in online version

# done differently locally

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

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

data.head()

sns.lineplot(x=data.columns, y = data.loc['Australia',:])

# plt.show(); plt.close()

# why do we see ugly axis?

# we'll do something about this later

# more convenient:

# dfs as input directly

# certain expecations from the data

# series to plot should be in columns

# re-arrange our df

data\_transposed = data.T

data\_transposed['Year'] = data.columns

data\_transposed.head()

sns.lineplot(data\_transposed, x='Year', y = 'Australia')

plt.show(); plt.close()

# more re-arrangements needed for full options

# "long format"

# column = variable or condition

# row = measurement

# Current format:

# country Australia New Zealand Year

# gdpPercap\_1952 10039.59564 10556.57566 gdpPercap\_1952

# gdpPercap\_1957 10949.64959 12247.39532 gdpPercap\_1957

# gdpPercap\_1962 12217.22686 13175.67800 gdpPercap\_1962

# gdpPercap\_1967 14526.12465 14463.91893 gdpPercap\_1967

# gdpPercap\_1972 16788.62948 16046.03728 gdpPercap\_1972

# (..)

#

# Goal:

# Year Country GDP

# 0 gdpPercap\_1952 Australia 10039.59564

# 1 gdpPercap\_1957 Australia 10949.64959

# 2 gdpPercap\_1962 Australia 12217.22686

# 3 gdpPercap\_1967 Australia 14526.12465

# 4 gdpPercap\_1972 Australia 16788.62948

# 5 gdpPercap\_1977 Australia 18334.19751

# 6 gdpPercap\_1982 Australia 19477.00928

# 7 gdpPercap\_1987 Australia 21888.88903

# 8 gdpPercap\_1992 Australia 23424.76683

# 9 gdpPercap\_1997 Australia 26997.93657

# 10 gdpPercap\_2002 Australia 30687.75473

# 11 gdpPercap\_2007 Australia 34435.36744

# 12 gdpPercap\_1952 New Zealand 10556.57566

# (..)

data\_long = data\_transposed.melt(id\_vars='Year', var\_name='Country', value\_name='GDP')

# id\_vars: identifier variables

# identifying a specific observation

# keep those

# new id vars will be added, in this case country

# var\_name: name used the new id parameter

# here, column names Australia, New Zealand ---> these are countries

# value\_name: name given to the values, which are taken from multiple columns

#

# - Identifier variables are called this because they uniquely \*\*identify a specific observation\*\* in the dataset.

# These variables are not measured or calculated but instead serve to distinguish one observation from another.

# - In this example, `Year` and `Country` are identifier variables because they uniquely identify each observation

# (e.g., GDP of Australia in 1952).

print(data\_long.head())

# OPTIONAL ##########

data2 = data.copy()

data2['Country']=data2.index

data\_long2 = data2.melt(id\_vars='Country', var\_name='Year', value\_name='GDP')

# Let's not go into technicalities

# sometimes a = b

# a "reference" is made

# instead of new parameter

# list1 = [1,2,3]

# list2 = list1

# list2[1] = 44

# list1

# list2

# END OPTIONAL ##########

sns.lineplot(data\_long, x='Year', y = 'GDP', hue='Country')

# plt.show(); plt.close()

# Finally

# Fix years

# search and replace

# keep year

# remove rest

years = data\_long['Year'].str.replace('gdpPercap\_', '')

# MW:

# "str" is a method which holds more methods

# related to string operations

# str is also part of python standard library

# stand-alone example: str.replace('hoi', 'oi', '')

# convert to int

# put back in dataframe

data\_long['Year'] = years.astype(int)

# Plot again

sns.lineplot(data\_long, x='Year', y = 'GDP', hue='Country')

plt.show(); plt.close()

# Change

# Plot types

# Style

# Seaborn can be manipulated by matplotlib

# More options

# Example

plt.style.use('ggplot')

# Other type, bars

sns.barplot(data\_long, x='Year', y = 'GDP', hue='Country')

# use google to tweak

# for nice examples:

# https://matplotlib.org/stable/gallery/

# https://seaborn.pydata.org/examples/index.html

# Another plot

plt.style.use('default')

sns.violinplot(data\_long, x='Country', y = 'GDP')

# OPTIONAL:

# sns.stripplot(data\_long, x='Country', y = 'GDP', hue='Year')

plt.xticks(rotation=45)

plt.title('GDPs of different countries')

# Calling matplotlib.pyplot directly

# plt.plot(x, y) like above

# also can choose different plot styles and tweak features

# e.g.

selected\_rows = data\_long['Country']=='Australia'

years = data\_long.loc[selected\_rows,'Year']

gdp\_australia = data\_long.loc[selected\_rows,'GDP']

plt.plot(years, gdp\_australia, linestyle='--', color='g')

# we can also use a shorthand argument:

# plt.plot(years, gdp\_australia, fmt='g--')

# more brief

# plt.plot(years, gdp\_australia, 'g--')

# combining data

# Select two countries' worth of data.

gdp\_nz = data\_long.loc[data\_long['Country']=='New Zealand','GDP']

# Plot with differently-colored markers.

plt.plot(years, gdp\_australia, 'b-', label='Australia')

plt.plot(years, gdp\_nz, 'g-', label='New Zealand')

# Create legend.

#plt.legend()

plt.legend(loc='lower right')

plt.xlabel('Year')

plt.ylabel('GDP per capita ($)')

# about legend,

# 2 stages to create:

# "label" argument to label each set of data

# create legend

# plt.legend()

# legend placement

# can use "loc" argument

# per default, tries to find good position

# (show this)

# scatter plot

# other style

# plt.scatter(gdp\_australia, gdp\_nz)

# can be done directly from dataframe also

sns.scatterplot(data\_transposed, x = 'Australia', y = 'New Zealand')

# To save plot

# PARTS OF THIS ARE OPTIONAL

# plt.savefig('my\_figure.png')

# note MW: pdf or svg very convenient

# note MW: also convenient:

# plt.tight\_layout() and

# plt.savefig('blabla.pdf', bbox\_inches='tight')

# plt.savefig will use latest figure that was last displayed

# either make sure you have the right one

# or

# data.plot(kind='bar')

# fig = plt.gcf() # get current figure

# fig.savefig('my\_figure.png')

# good practice

# make text large enough in powerpoints

# use fontsize parameter in xlabel, ylabel, title, and legend, and tick\_params with labelsize

# make symbols readable

# ie large enough; "s" parameter to increase size

# be careful with colors (only) to distinguish lines

# use color-blind friendly palette

# color-blind emulators:

# https://www.color-blindness.com/coblis-color-blindness-simulator/

# https://colororacle.org/

# linestyle to counter black-white printing, or marker for scatter plots

# Good practice example:

import matplotlib.pyplot as plt

# Bang Wong colorblind-friendly color scheme (https://www.nature.com/articles/nmeth.1618)

colors\_bangwong = [

"#E69F00", # Orange

"#56B4E9", # Sky Blue

"#009E73", # Bluish Green

"#F0E442", # Yellow

"#0072B2", # Blue

"#D55E00", # Vermillion

"#CC79A7", # Reddish Purple

"#000000" # Black

]

plt.style.use('default')

fig, ax = plt.subplots(1,1, figsize=(10/2.54,10/2.54))

ax.plot([1,2,3,4], [1,4,9,16], linestyle='--', color=colors\_bangwong[1], label=r'$x^2$')

ax.plot([1,2,3,4], [1,5,11,19], linestyle=':', color=colors\_bangwong[2], label=r'$x^2+(x-1)$')

ax.legend()

ax.set\_xlabel('X-axis', fontsize=12)

ax.set\_ylabel('Y-axis', fontsize=12)

ax.set\_title('Sample Plot', fontsize=12)

ax.legend(fontsize=12)

ax.tick\_params(axis='both', which='major', labelsize=12)

plt.tight\_layout()

# plt.show()

# plt.savefig('/Users/m.wehrens/Desktop/202503\_Python-Gapminder-testingcode.pdf', dpi=300, bbox\_inches='tight')

# plt.close(fig)