Introduction to Programming in Python

Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Pandas Dataframes

Introduction to Pandas

- `pandas` (from *p*anel *d*ata) is another fundamental package (user quide).
- It provides a number of datastructures (*series*, *dataframes*, and *panels*) designed for storing observational data, and powerful methods for manipulating (*munging*, or *wrangling*) these data.
- It is usually imported as `pd`:

In [ ]:
```
import pandas as pd
```

- Pandas is incredibly powerful; we will only scratch the surface here. Working with it will require a lot of googling.

Series

- A pandas `Series` is essentially a NumPy array, but not necessarily indexed with integers.

In [ ]:
```
pop = pd.Series([5.7, 82.7, 17.0], name='Population'); pop  # the descriptive name is optional
```

- The difference is that the index can be anything, not just a list of integers:

In [ ]:
```
pop.index=['DK', 'DE', 'NL']
```

- The index can be used for indexing (duh...):

In [ ]:
```
pop['NL']
```

- The index is preserved when operating on a `series`:

In [ ]:
```
gdp = pd.Series([3494.898, 769.930], name='Nominal GDP in Billion USD', index=['DE', 'NL']); gdp
```

In [ ]:
```
gdp / pop
```

- One advantage of a `Series` compared to NumPy arrays is that they can handle missing data, represented as `NaN` (not a number).

Dataframes

- A `DataFrame` is a collection of `Series` with a common index (which labels the rows).

In [ ]:
```
data = pd.concat([gdp, pop], axis=1, sort=False); data  # concatenate two Series to a DataFrame.
```

- Columns are indexed by column name:

In [ ]:
```
data.columns
```

In [ ]:
```
data['Population']  # data.Population works too
```

- Rows are indexed with the `loc` method:

In [ ]:
```
data.loc['NL']
```

- Unlike arrays, dataframes can have columns with different datatypes.
- There are different ways to add columns. One is to just assign to a new column:

In [ ]:
```
data['Language'] = ['German', 'Danish', 'Dutch']  # add a new column from a list
```

- To add rows, use `loc` or `append`:

In [ ]:

```
print(data.loc["DE"])
data.loc['AT'] = [386.4, 8.7, 'German']  # add a row with index 'AT'.
s = pd.DataFrame([[511.0, 9.9, 'Swedish']], index=['SE'], columns=data.columns)
data = data.append(s)  # add a row by appending another dataframe. May create duplicates.
data
```

- The `dropna` method can be used to delete rows with missing values:

In [ ]:
```
data = data.dropna(); data
```

- Useful methods for obtaining summary information about a dataframe are `mean`, `std`, `info`, `describe`, `head`, and `tail`.

In [ ]:
```
data.describe()
```

In [ ]:
```
data.head() # show the first few rows; data.tail shows the last few
```

- To save a dataframe to disk as a csv file, use

In [ ]:
```
data.to_csv('myfile.csv') # to_excel exists as well.
```

- To load data into a dataframe, use `pd.read_csv`:

In [ ]:
```
pd.read_csv('myfile.csv', index_col=0)
```

In [ ]:
```
import os
os.remove('myfile.csv') # clean up
```

Usually, you won't be creating dataframes from scratch; rather, they result from obtaining data from somewhere. E.g., Pandas can open CSV files directly from a URL, resulting in a dataframe:

In [ ]:

```
import os.path
fname = "coviddata.csv"
URL = "https://covid.ourworldindata.org/data/owid-covid-data.csv"
if os.path.isfile(fname): # only download once
    df = pd.read_csv(fname)
else:
    df = pd.read_csv(URL)
df.head()
```

Split-Apply-Combine

- An important concept in working with data is the "split-apply-combine" paradigm: split the data according to some criterion, apply an operation to it, and then combine the results into a new dataframe.
- Suppose the CoViD data above only contained the daily new cases, not the total number of cases. How could we compute the total number of cases?
- Answer: split the data by country, sum the daily new cases, and then combine the results into a new dataframe.

- In Pandas, the paradigm corresponds to `groupby`:

In [ ]:
```
df.groupby("location").new_cases.sum()
```

Plotting

Pandas can directly be used for plotting. More advanced funtionality requires `matplotlib` (more on that below).

In [ ]:
```
%matplotlib inline
sw = df.loc[df['location'] == "Switzerland"]
sw.plot(x="date", y="new_cases");
```

The first line is an ipython magic. It makes plots appear inline in the notebook.

- Advanced plotting requires the `matplotlib` library (user guide), which is inspired by the plotting facilities of Matlab®.
- Its main plotting facilities reside in its `pyplot` module. It is usually imported as

In [ ]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

`matplotlib` enables us to make the above plot prettier:

In [ ]:
```
sw = df.loc[df['location'] == "Switzerland"]
sw.plot(x="date", y="new_cases")
plt.gcf().autofmt_xdate()
```

- The `seaborn` library (user guide) provides higher-level statistical visualizations:

In [ ]:
```
import seaborn as sns
```

- I will only give a brief introduction to matplotlib here. The fundamental object in matplotlib is a `figure`, inside of which reside `subplots` (or `axes`).
- To create a new figure, add an axis, and plot to it:

In [ ]:

```
# with the inline backend, these need to be in the same cell.
fig = plt.figure(figsize=(6,3)) # create a new empty figure object. size is optional.
ax1 = fig.add_subplot(121) # layout: (1x2). ax1 is the top left one
ax2 = fig.add_subplot(122)
ax1.plot(range(10))
ax2.plot(range(10, 0, -1));
```

- By default, matplotlib plots into the current axis, creating one (and a figure) if needed. Using the convenience method `subplot`, this allows us to achieve the same without explicit reference to figures and axes:

In [ ]:

```
plt.subplot(121)
plt.plot(range(10))
plt.subplot(122)
plt.plot(range(10, 0, -1));
```

- To plot two vectors x and y against each other:

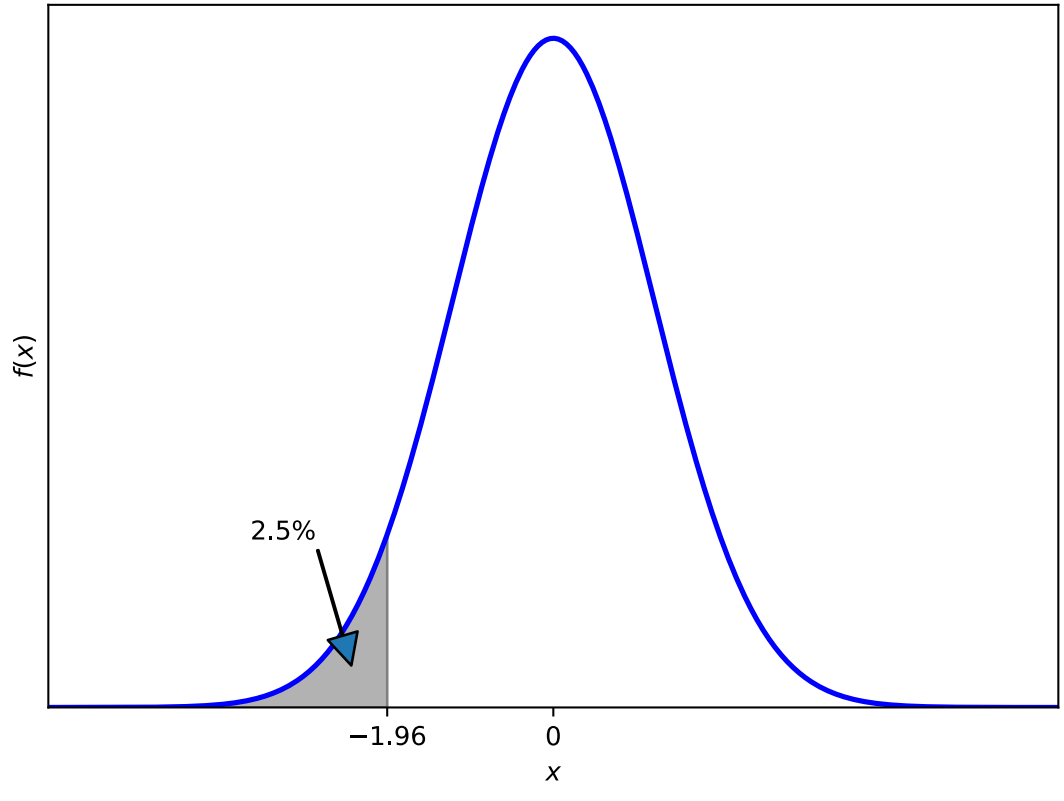
In [ ]:

```
import numpy as np
x = np.linspace(-10, 10, 100)
y = x**2
plt.plot(x,y,'r--') # dashed red line
plt.xlabel('$x$') # math (LaTeX) equations can be included by enclosing in $$.
plt.ylabel('$y$')
plt.title('A Parabola')
plt.legend(['$f(x)$']) # expects a list of strings
plt.xlim(xmin=-8, xmax=8); # axis limits
# plt.savefig('filename.svg') # save the plot to disk
```

A more advanced example: quantiles of the normal distribution

In [ ]:

```
from matplotlib.patches import Polygon
import scipy.stats as stats
a, b, c = -5, 5, stats.norm.ppf(0.05)
x = np.linspace(a, b, 500)
y = stats.norm.pdf(x)
fig = plt.figure(figsize=(7, 5))
ax = fig.add_subplot(111)
plt.plot(x, y, 'b', linewidth=2)
plt.ylim(ymin=0)
plt.xlim(xmin=a, xmax=b)
Ix = np.linspace(a, c)
Iy = stats.norm.pdf(Ix)
verts = [(a, 0)] + list(zip(Ix, Iy)) + [(c, 0)]
poly = Polygon(verts, facecolor='0.7', edgecolor='0.5')
ax.add_patch(poly)
ax.annotate(
            '$2.5\%$', xy=(-2, 0.025), xytext=(-3, 0.1),
            arrowprops=dict(width=.5),
            )
plt.xlabel('$x$')
plt.ylabel('$f(x)$')
ax.set_xticks([c, 0])
ax.set_xticklabels(['$-1.96$', '0'])
ax.set_yticks([])
plt.savefig('img/var.svg')
plt.close()
```


f(x)
2.5%
−1.96
0
x

Working with Time Series

Data Types

- Different data types for representing times and dates exist in Python.
- The most basic one is `datetime` from the eponymous package:

In [ ]:
```
from datetime import datetime
datetime.today()
```

- `datetime` objects can be created from strings using `strptime` and a format specifier:

In [ ]:
```
datetime.strptime('2017-03-31', '%Y-%m-%d')
```

- Pandas uses `Timestamps` instead of `datetime` objects. Unlike timestamps, they store frequency and time zone information. The two can mostly be used interchangeably.

In [ ]:
```
pd.Timestamp('2017-03-31')
```

- A time series is a `Series` with a special index, called a `DatetimeIndex`; essentially an array of `Timestamp`s.
- It can be created using the `date_range` function.

In [ ]:
```
import numpy as np
myindex = pd.date_range(end=pd.Timestamp.today(), normalize=True, periods=100, freq='B')
P = 20 + np.random.randn(100).cumsum()  # make up some share prices.
aapl = pd.Series(P, name="AAPL", index=myindex)
aapl.tail()
```

- As a convenience, Pandas allows indexing timeseries with date strings:

In [ ]:
```
aapl['4/11/2022']
```

Recommended reading

- https://python-course.eu/numerical-programming/ 23-28, 32-34

Homework

https://github.com/guipsamora/pandas_exercises/tree/master/01_Getting_%26_Knowing_Your_