Transforming Excel Analysis into pandas

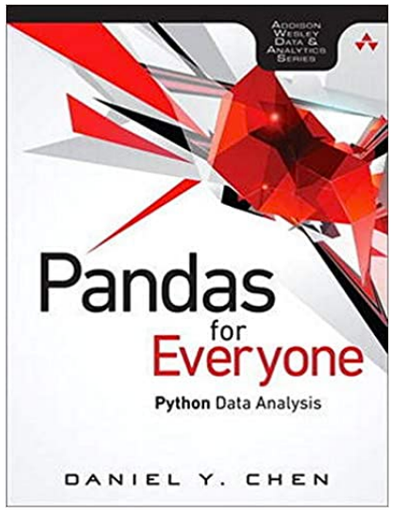
Bruno Gonçalves

www.data4sci.com/newsletter

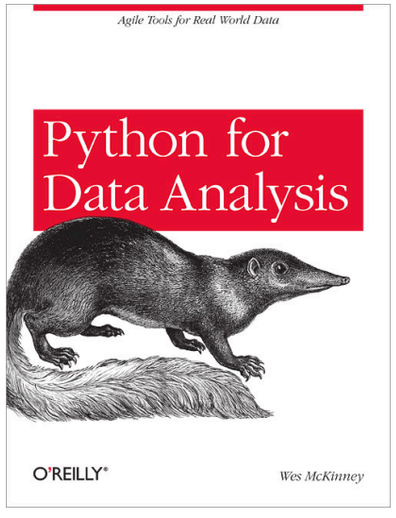
https://github.com/DataForScience/Excel

References

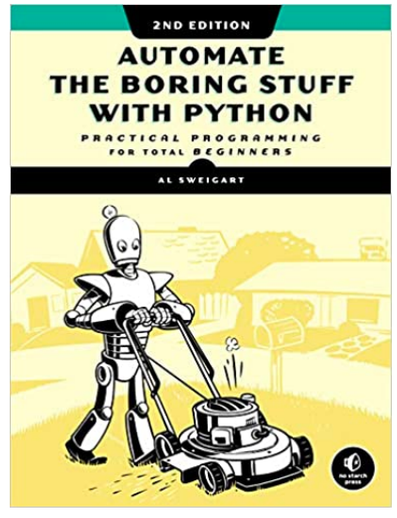
https://github.com/DataForScience/Excel



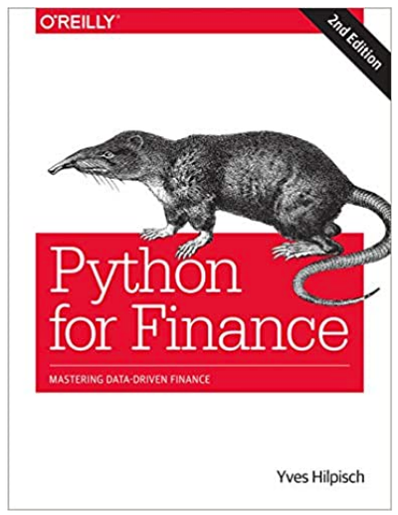
https://amzn.to/3bk8fkV



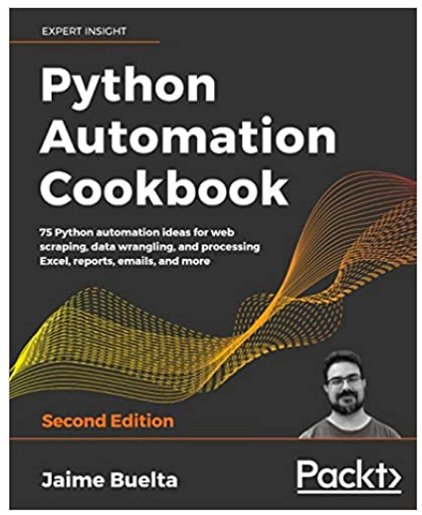
https://amzn.to/2Pmg6UX



https://amzn.to/3hYokiA



https://amzn.to/3bnycQu



https://amzn.to/3gNQzPC

Lesson I:
Excel Pitfalls

Microsoft Excel

- Developed by Microsoft in 1983.
- Part of the Microsoft Office suite of software
- Currently available for Windows, macOS, Android and iOS.
- Allows for calculation, plotting, pivot tables, and a macro programming language called Visual Basic for Applications.
- Essentially a visual WYSIWYG tool
- Cell based computation

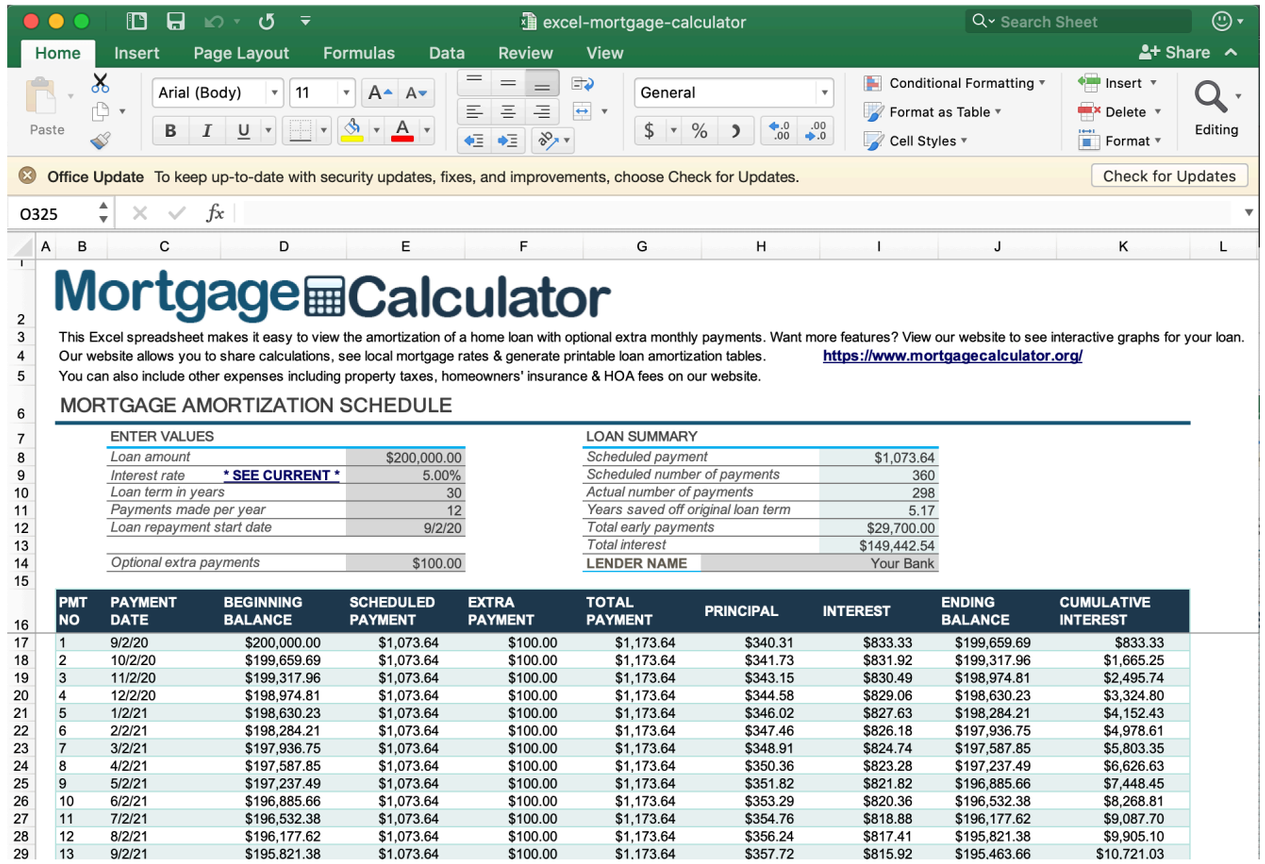
Microsoft Excel

https://support.microsoft.com/en-us/office/excel-specifications-and-limits-1672b34d-7043-467e-8e27-269d656771c3

Total number of rows and columns on a worksheet	1,048,576 rows by 16,384 columns
Column width	255 characters
Total number of characters that a cell can contain	32,767 characters
Noncontiguous cells that can be selected	2,147,483,648 cells

Cell based computations

- Cells show only the current value.



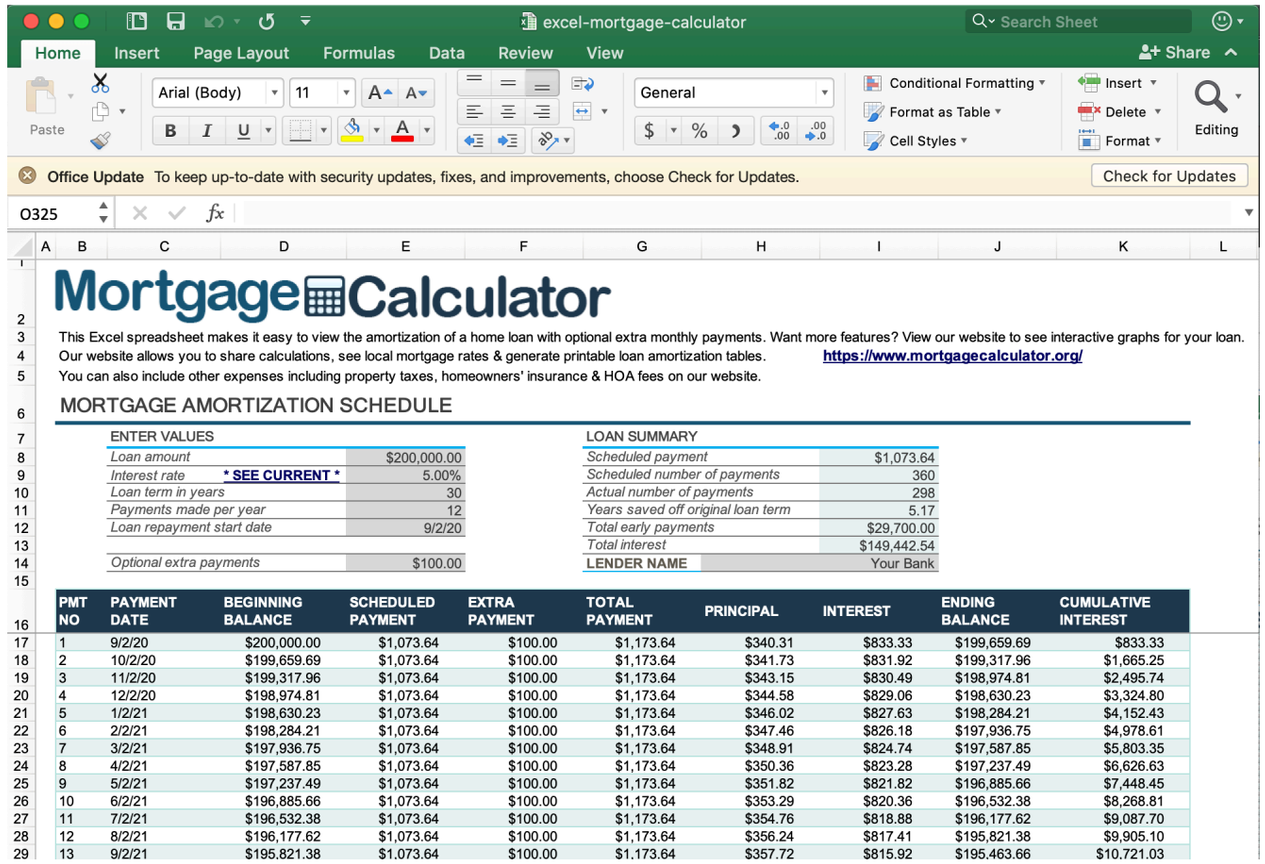
Cell based computations

- Cells show only the current value.
- Underlying logic is not clear

G22

=IF([@[PMT NO]]<>"",IF([@[SCHEDULED PAYMENT]]+[@[EXTRA PAYMENT]]<=[@[BEGINNING BALANCE]],[@[SCHEDULED PAYMENT]]+[@[EXTRA PAYMENT]],[@[BEGINNING BALANCE]]),"")

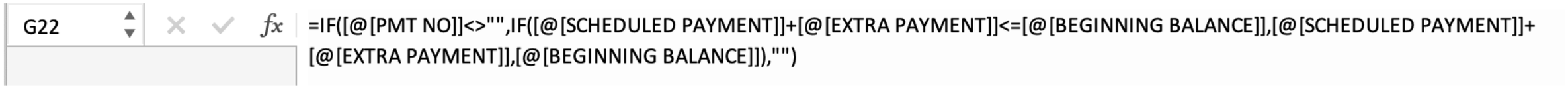
- Logic bugs are hard or impossible to spot



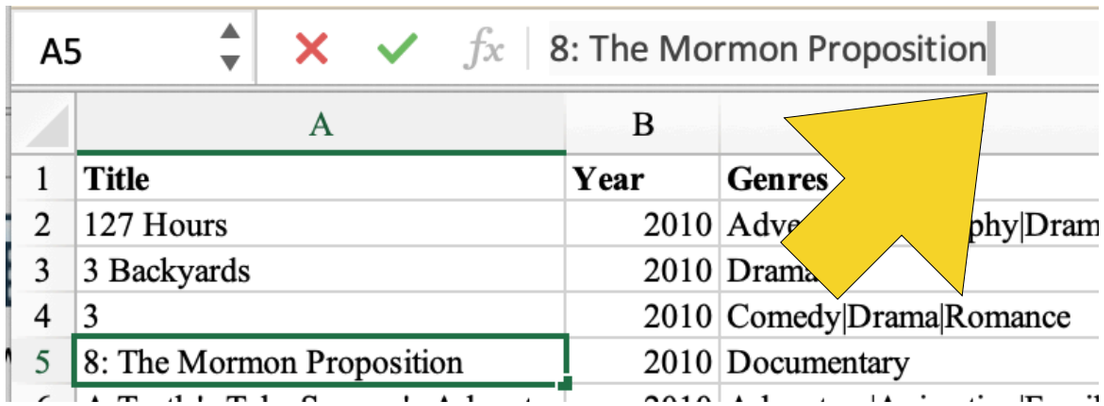
PMT NO	PAYMENT DATE	BEGINNING BALANCE	SCHEDULED PAYMENT	EXTRA PAYMENT	TOTAL PAYMENT	PRINCIPAL	INTEREST	ENDING BALANCE	CUMULATIVE INTEREST
1	9/2/20	$200,000.00	$1,073.64	$100.00	$1,173.64	$340.31	$833.33	$199,659.69	$833.33
2	10/2/20	$199,659.69	$1,073.64	$100.00	$1,173.64	$341.73	$831.92	$199,317.96	$1,665.25
3	11/2/20	$199,317.96	$1,073.64	$100.00	$1,173.64	$343.15	$830.49	$198,974.81	$2,495.74
4	12/2/20	$198,974.81	$1,073.64	$100.00	$1,173.64	$344.58	$829.06	$198,630.23	$3,324.80
5	1/2/21	$198,630.23	$1,073.64	$100.00	$1,173.64	$346.02	$827.63	$198,284.21	$4,152.43
6	2/2/21	$198,284.21	$1,073.64	$100.00	$1,173.64	$347.46	$826.18	$197,936.75	$4,978.61
7	3/2/21	$197,936.75	$1,073.64	$100.00	$1,173.64	$348.91	$824.74	$197,587.85	$5,803.35
8	4/2/21	$197,587.85	$1,073.64	$100.00	$1,173.64	$350.36	$823.28	$197,237.49	$6,626.63
9	5/2/21	$197,237.49	$1,073.64	$100.00	$1,173.64	$351.82	$821.82	$196,885.66	$7,448.45
10	6/2/21	$196,885.66	$1,073.64	$100.00	$1,173.64	$353.29	$820.36	$196,532.38	$8,268.81
11	7/2/21	$196,532.38	$1,073.64	$100.00	$1,173.64	$354.76	$818.88	$196,177.62	$9,087.70
12	8/2/21	$196,177.62	$1,073.64	$100.00	$1,173.64	$356.24	$817.41	$195,821.38	$9,905.10
13	9/2/21	$195,821.38	$1,073.64	$100.00	$1,173.64	$357.72	$815.92	$195,463.66	$10,721.03

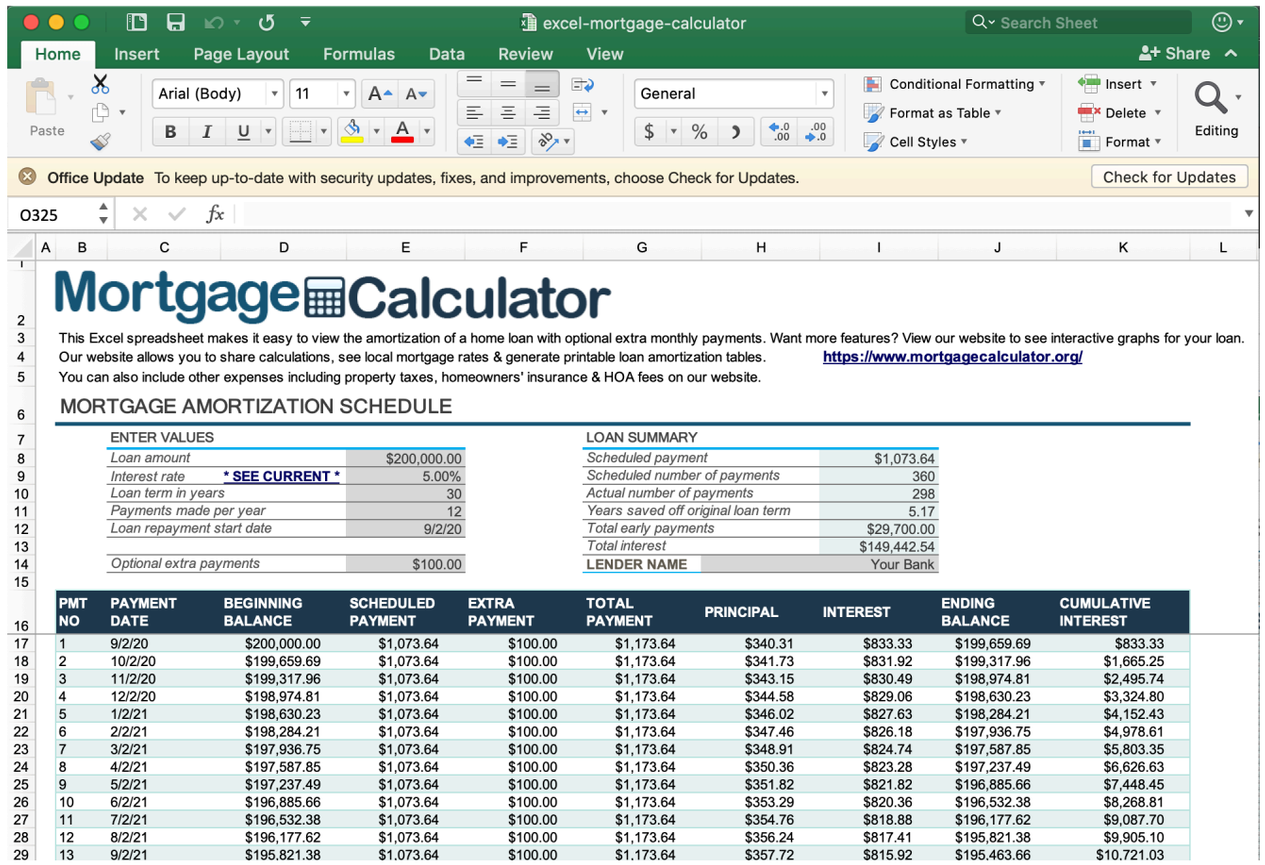
Cell based computations

- Cells show only the current value.
- Underlying logic is not clear



- Logic bugs are hard or impossible to spot
- No way to standardize the data across the entire sheet (issues with white space and non-printing characters)





Hard to Automate

- The Graphical interface makes it easy to explore the functionality, but hard to automate it
- The only way to reproduce a computation is to redo each step, manually, a process that is error-prone
- Data is hard to validate
- Different worksheets in the same workbook can follow different conventions

Code - Pitfalls

https://github.com/DataForScience/Excel

Lesson II:
Pandas DataFrames

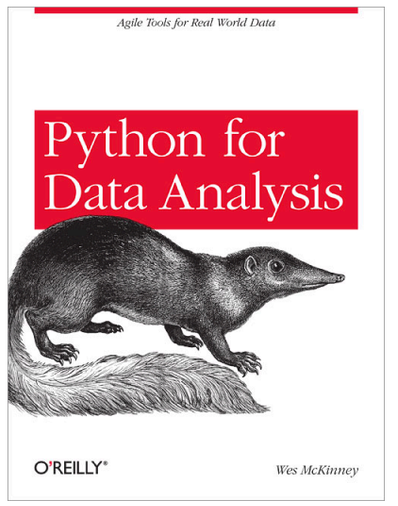
pandas

pandas.pydata.org

- Created by Wes McKinney in 2008 while working for AQR Capital Management, currently working at Two Sigma and Ursa Labs
- Pandas is specifically designed to handle time series and data frames
- High performance, flexible tool for quantitative analysis on financial data
- The functionality is built on top numpy and matplotlib (see next lecture)

- The fundamental data structures are Series (1-dimensional) and DataFrame (2-dimensional).
- Each column of a DataFrame can have a different dtype but all the elements in a column (or Series) must be of the same dtype
- Conventionally imported as

```
import pandas as pd
```



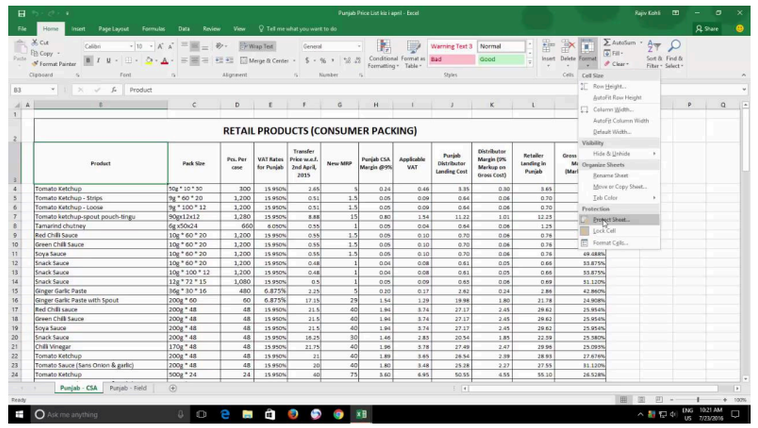
https://amzn.to/2Pmg6UX

Series and Data Frames

- A Series is conceptually equivalent to a **numpy** array with some extra information (column and row names, etc)

- A DataFrame can be thought of as the union of several Series, with names associated.

- Similarly, you can think of a DataFrame as an Excel **sheet** and of a Series as an individual **column**



- A minimal DataFrame implementation would be a dict where each element is a list

Series

	id
0	23
1	42
2	12
3	86

+

Series

	Name
0	“Bob”
1	“Karen”
2	“Kate”
3	“Bill”

=

DataFrame

	id	Name
0	23	“Bob”
1	42	“Karen”
2	12	“Kate”
3	86	“Bill”

DataFrame Fields

- DataFrames contain a great deal of extra information in addition to just the data values. In particular, they include:
    - columns - column names
    - index - row names
    - dtypes - dtype associated with each column
    - shape - number of rows and columns
    - ndim - the number of dimensions
- DataFrames make it easy to add new columns.
- DataFrame elements can be accessed and modified in various ways

DataFrame Exploration

- We get information about the contents of a DataFrame using several simple methods:
    - head() - display the top 5 rows
    - tail() - display the bottom 5 rows
    - info() - Print a concise summary of a DataFrame
    - describe() - Generate descriptive statistics that summarize the data

Indexing and Slicing

- pandas supports various ways of indexing the contents of a DataFrame
- [<column name>] - select a given column by it's name. column names can also be used as field names
- .loc[<row name>] - select a given row by it's (Index) name
- .iloc[<position>] - select a given row by it's position in the Index, starting at 0
- .loc[<row name>, <column name>] - select an individual element by row and column name
- .iloc[<row position>, <row position>] - select an individual element by row and column position (starting at 0)
- .iloc also supports ranges and slices similarly to python lists or numpy arrays

Importing and Exporting Data

- pandas has powerful methods to read and write data from multiple sources:
  - pd.read_clipboard() - Create DataFrame from the clipboard data
  - pd.read_csv() - Read a comma-separated values file
  - pd.read_fwf() - Read fixed width text file
  - pd.read_excel() - Read an Excel file
  - pd.read_html() - Read tabular data from a URL (or local html file)
  - pd.read_pickle() - Read a Pickle file
- Each of these functions accepts a large number of options and parameters controlling its behavior. Use help(<function name>) to explore further.
- Each read_*() function has a complementary to_*() function to write out a DataFrame to disk. The to_*() functions are members of the DataFrame object

Importing and Exporting Data

- pandas has powerful methods to read and write data from multiple sources:
  - pd.read_clipboard() - Create DataFrame from the clipboard data
  - pd.read_csv() - Read a comma-separated values file
  - pd.read_fwf() - Read fixed width text file
  - pd.read_excel() - Read an Excel file
  - pd.read_html() - Read tabular data from a URL (or local html file)
  - pd.read_pickle() - Read a Pickle file
- Each of these functions accepts a large number of options and parameters controlling its behavior. Use help(<function name>) to explore further.
- Each read_*() function has a complementary to_*() function to write out a DataFrame to disk. The to_*() functions are members of the DataFrame object

Text-based formats

- pd.read_csv() - Read a DataFrame from a comma-separated values file. This is perhaps the most common and most robust read method and is able to process practically any well-formed text-based format (including compressed files)
    - sep=' ' - Define the separator to use. ',' is the default (hence the name). ' ', '|', and '\t' are also common
    - header=0 - Row number to use as column names
    - names - List of column names to use.
    - index_col=None - Column(s) to use as the row labels
    - usecols - Subset of the columns to use. By default it uses all columns
    - nrows - Number of rows of file to read. All by default.
    - parse_dates - List of column names to treat as dates.
    - dtype - Dictionary of column names to dtypes to use

Web pages

- pd.read_html() - Read an html file or URL. Whenever possible, parameter compatibility with pd.read_csv() is maintained. The most important difference is that it will automatically return a list of DataFrames if the file or webpage contains multiple tables.
    - attrs=0 - A dictionary of HTML attributes that you can pass to use to identify the table in the HTML. dict keys must be valid HTML attributes
    - flavor=None - Similar to engine for excel files. Use a specific library to process the file. The currently supported ones are "bs4", "html5lib" or “lxml" -The default value of None tries them in order until a valid one is found, starting with ”lxml”.

Excel workbook

- pd.read_excel() - Read an Excel file. Most parameters work similarly to pd.read_csv() but a few other are of particular interest:
    - sheet_name=0 - Which sheet name to load. Strings are used for sheet names and zero-index Integers are used to specify sheet positions (0 is the first, 1 the second, etc). If a list is provided, multiple sheets can be loaded at the same time as a dict keyed by sheet_name.
    - engine=None - Use a specific library to process the file. The currently supported ones are "xlrd", "openpyxl" or “odf" - Each engine will have small differences in the way it works and will support different formats (for example, "openpyxl" does not support old xls files, while “odf" requires an extra package to be installed.
    - na_values - Additional values to recognize as NaN. You can use a dict to specify different values per column.

Excel workbook

- DataFrame.to_excel(filename) - Write the contents of the DataFrame as an Excel file. filename can also be a file or ExcelWriter object:
    - sheet_name='Sheet1' - Which sheet name to write. 'Sheet1' is the default name when none is specified.
    - engine=None - Use a specific library to process the file. Currently supported options are "'openpyxl' or 'xlsxwriter'.
    - index=True - whether or not to write the row names.
    - na_rep="" - String used to represent NaN values.

Excel workbook

- pd.ExcelFile(filename) - Open an Excel file and return an ExcelFile object that represents the entire file:
    - sheet_names - list all the sheet names
    - book - an object representing the entire work book
    - .parse(sheet_name) - parse a specific excel sheet and return it as a DataFrame. Most common read_excel arguments can be used here as well
- pd.ExcelWriter(filename, mode) - open a file for writing (mode="w") or appending (mode="a")
    - return a "file pointer-like" object (similar to the vanilla open()) that can be passed to the .to_excel() DataFrame method.

Time Series

- pandas was originally developed to handle financial data
- Temporal sequences (time series) are a common type of data encountered in financial applications, so, naturally, pandas has good support for the most common time series operations.
- pd.to_datetime - converts a Series or value into a date time timestamp.
    - Format is inferred by looking at the entire Series data
    - Format can also be manually specified using specific arguments:
        - format - detailed string specifying the full format
        - dayfirst=True - parse 10/11/12 as Nov 10, 2012
        - yearfirst=True - parse 10/11/12 as Nov 12, 2010
- pd.to_timedelta - converts a Series into an absolute difference in time
- Dates can also be parse automatically at read time by passing a list of the columns containing dates to the parse_dates argument of pd.read_csv(), etc…

Time Series

- Sequences of dates/times can be created using pd.date_range(), similar to np.arange()
  - start/end - specify the limits
  - freq - specify frequency (step)
    - B - business day frequency
    - D - calendar day frequency
    - W - weekly frequency
    - M - month end frequency
    - Q - quarter end frequency
    - A - year end frequency
    - H - hourly frequency
    - T - minutely frequency
    - S - secondly frequency
- By setting the index to be a date time, we turn the DataFrame into a Time Series.
- Pandas has several methods that are specialized to this case
- index.day/index.month/index.year - return the day, month and year of the time series index value

Transformed values

- We often need to apply some simple transformation to the values in our original Series. Pandas offers several methods to accomplish this goal:
  - map(func) - Map values of Series according to input correspondence.
    - func - function, dict or Series to use for mapping
  - transform(func) - Transform each column/row of the DataFrame using a function. Output must have the same shape as the original
    - axis = 0 - apply function to columns
    - axis = 1 - apply function to rows
- apply(func) - Apply a function along an axis of the DataFrame
  - Similar to transform() but without the limitation of preserving the shape

Lagged values and differences

- While analyzing time series, we often refer to values that our time series took 1, 2, 3, etc time steps in the past
- These are known as lagged values and denoted:

$$x_{t-l}$$

- where l is the value of the lag we are considering.
- pandas supports lagged values with the shift(l) DataFrame method
- Perhaps the most common use case for lagged values is for the calculation of differences of the form:

$$x_t - x_{t-l}$$

- Where $l \geq 1$ is the value of the lag we are interested in.
- Differences are also a particularly simple way to detrend a time series
- pandas supports lagged values with the diff(l) DataFrame method

merge/join

- merge() and join() allows us to perform database-style join operation by columns or indexes (rows)

- Some of the most important arguments are common to both methods:

    - on - Column(s) to use for joining, otherwise join on index. Can pass an array as the join key if not already contained in the calling DataFrame. Like an Excel VLOOKUP operation

    - how - Type of join to perform: {'left', 'right', 'outer', 'inner'}

- merge() is more sophisticated and flexible. It also allows us to specify:

    - left_on/right_on - Field names to join on for each DataFrame

    - left_index/right_index - Whether or not to use the left/right index as join key(s)

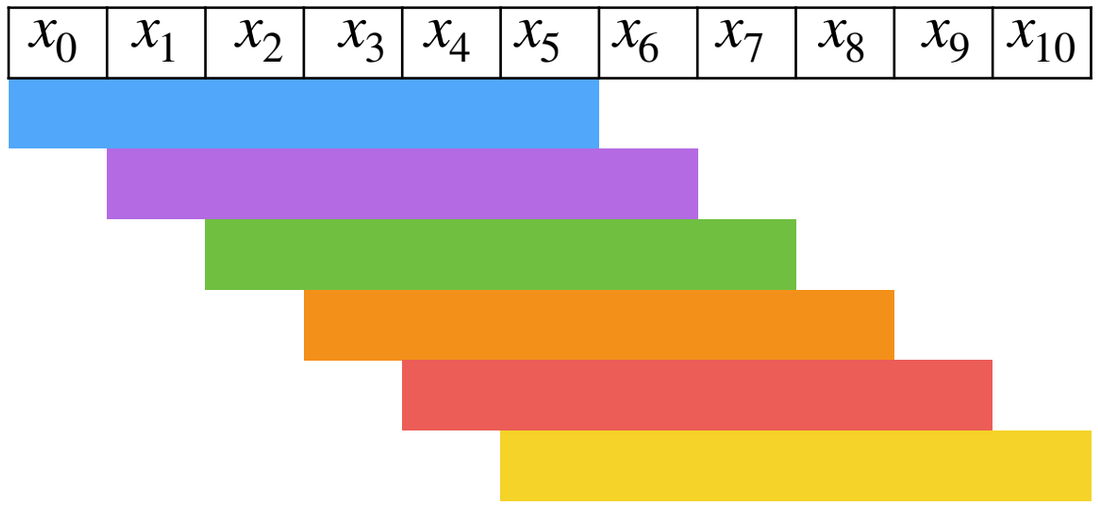
Code - Pandas DataFrames

https://github.com/DataForScience/Excel

Lesson III:
Simple Data Modeling

Windowing

- When analyzing the temporal behavior of of a signal, we often need to evaluate if specific quantities are time varying or not
- A common approach is to use sliding windows of a given length to evaluate the required values
- So a sliding window of width 6 on a series of length 11 would look like:



$$= \begin{matrix} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \\ x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \\ x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} \end{matrix}$$

- and we would calculate the metric of interest within each window.
- In Pandas we use the method rolling(w) that returns dynamic view of the data allowing us to daisy-chain other common numpy operators. rolling(w).mean(), rolling(w).max(), etc…

GroupBy

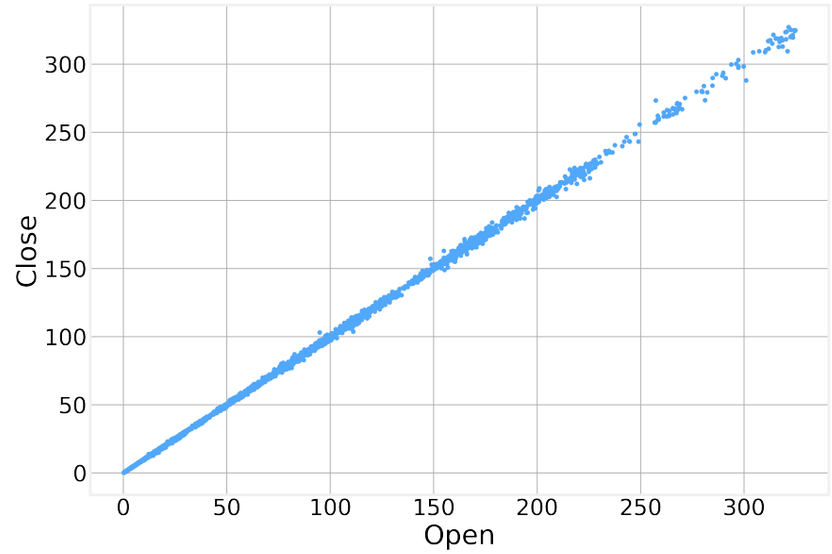
- Sometimes we need to calculate statistics for subsets of our data
- groupby() allows us to group data based on the values of a column
- groupby() returns a GroupBy object that supports several aggregations functions, including:
    - max()/min()/mean()/median()
    - transform()/apply()
    - sum()/cumsum()
    - prod()/cumprod()
    - quantile()
- Each of these functions is applied to the contents of each group

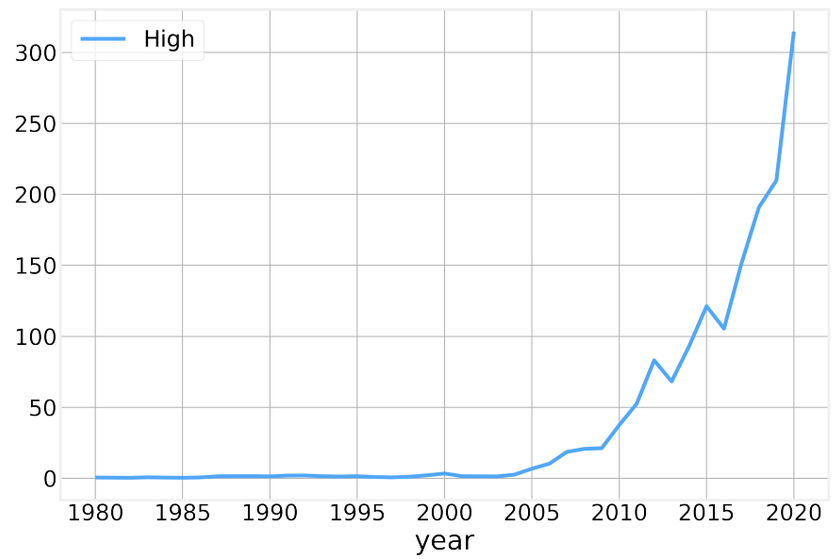
Pivot Tables

- Pandas provides an extremely flexible pivot table implementation
- pd.pivot_table() - Creates a spreadsheet-style pivot table as a DataFrame.
- Three important arguments:
    - values - column to aggregate
    - index - Keys to group by on the pivot table index. If an array is passed, it is being used as the same manner as column values.
    - columns - Keys to group by on the pivot table column.
    - aggfunc - Function to use for aggregation. Defaults to np.mean()
- index, columns and aggfunc can also be lists of keys or functions to use.

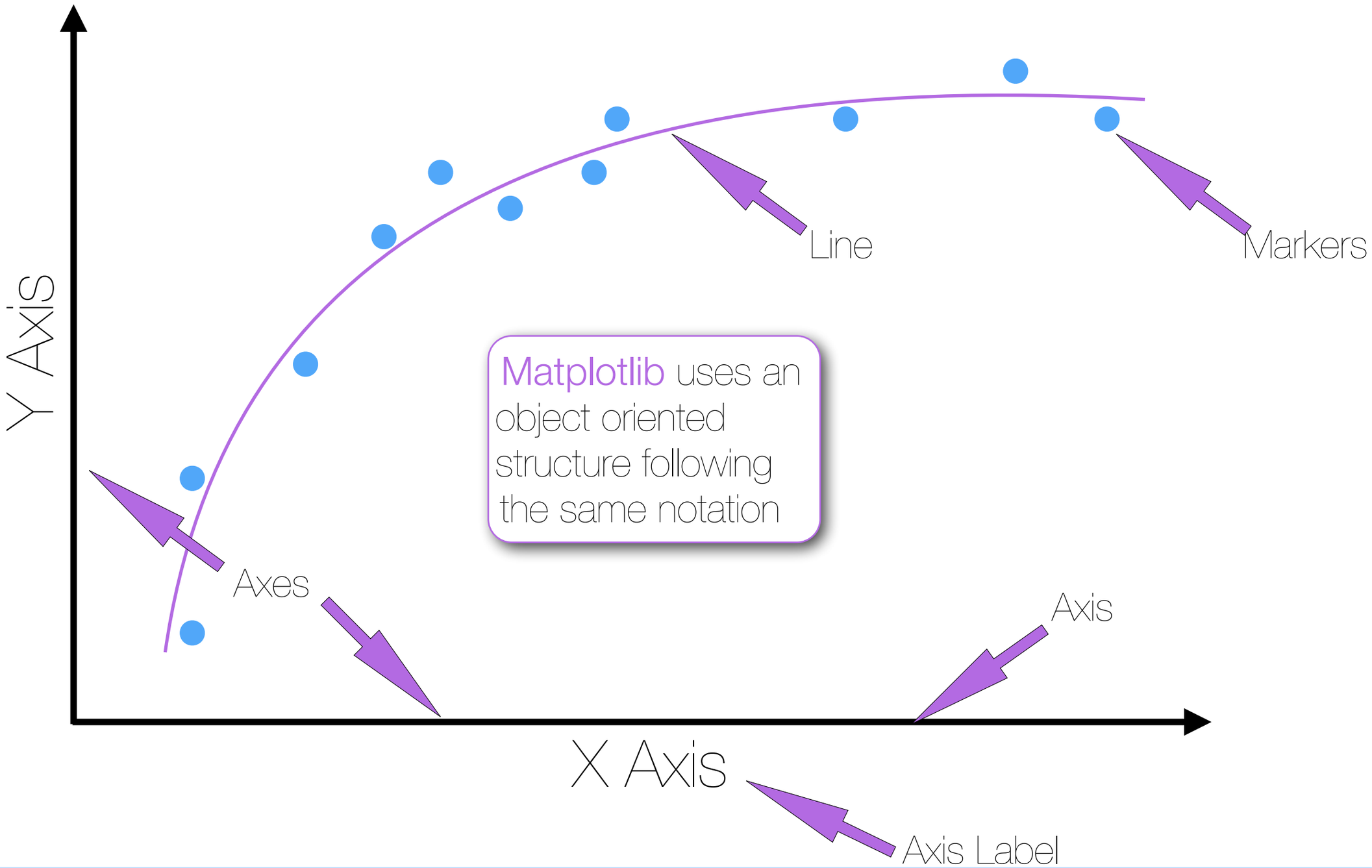
plotting

- Finally, pandas also provides a simple interface for basic plotting through the plot() function
- By specifying some basic parameters the variables plotted and even the kind of plot can be easily modified:
  - **x/y** -column name to use for the **x/y** axis
  - **kind** - type of plot
    - ‘line’ - line plot (default)
    - ‘bar’/‘barh’ - vertical/horizontal bar plot
    - ‘hist’ - histogram
    - ‘box’ - boxplot
    - ‘pie’ - pie plot
    - ‘scatter’ - scatter plot
- The plot() function returns a matplotlib Axes object

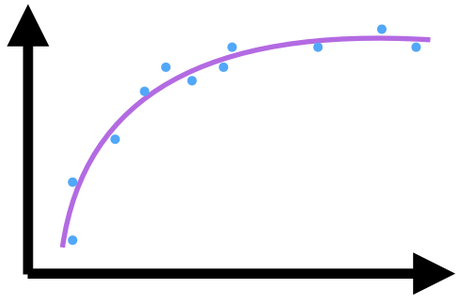




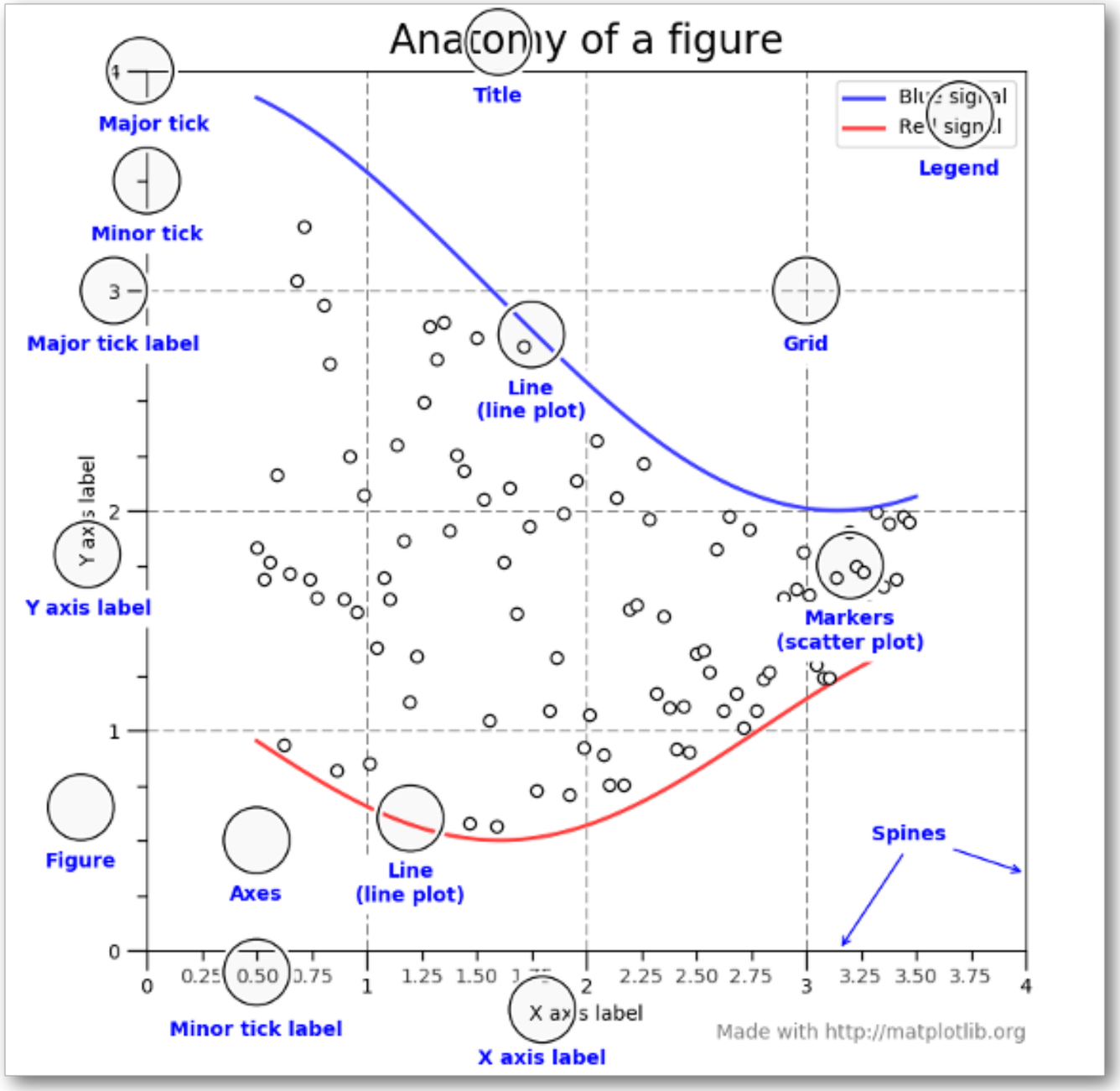
Basic Plotting



Basic Plotting - Programmatically!

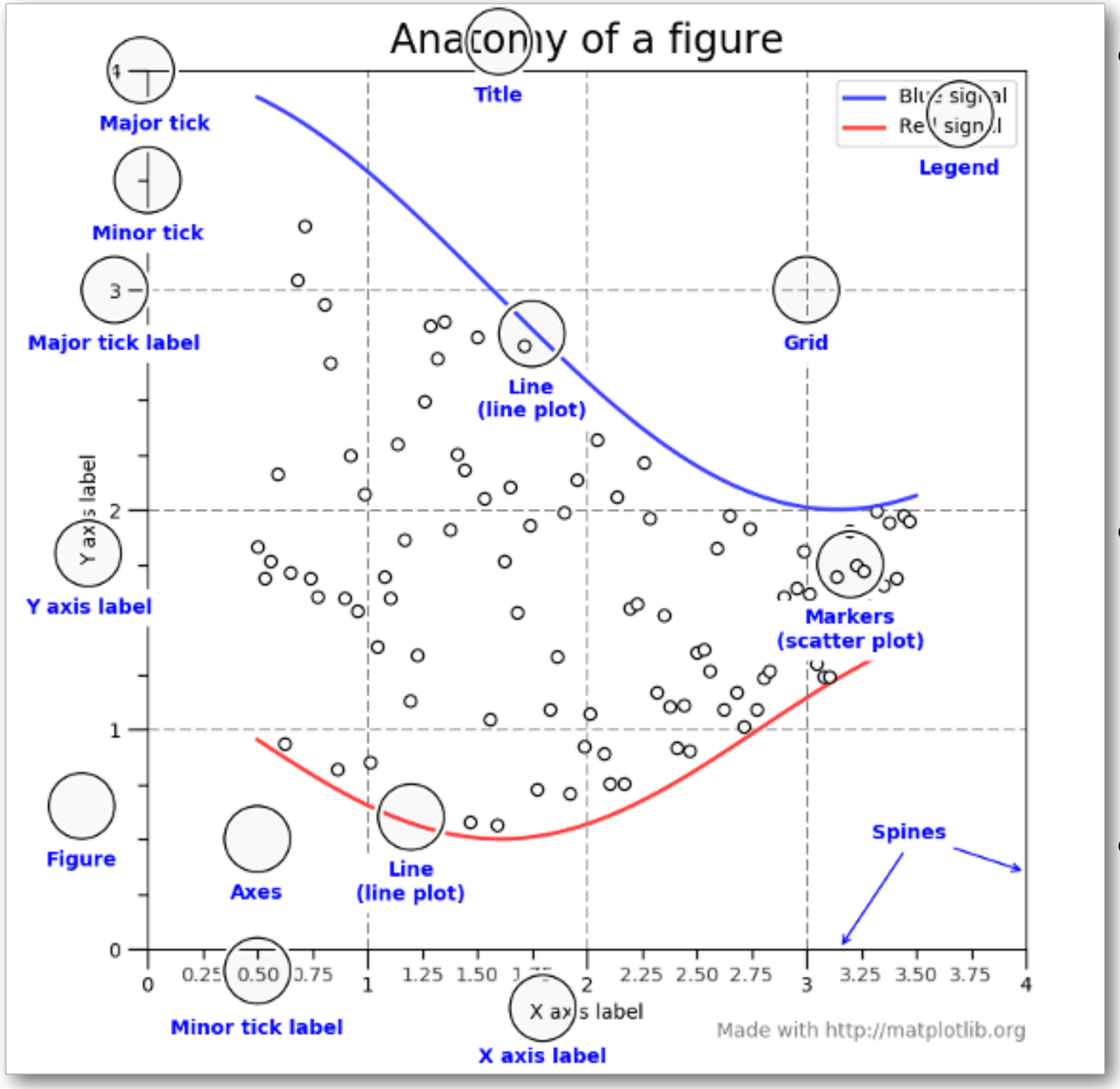


matplotlib.org/3.1.0/



matplotlib - decorations

matplotlib.org/3.1.0/



- The respective functions are named in an intuitive way, Every Axes object has as methods:
  - .set_xlabel(label)
  - .set_ylabel(label)
  - .set_title(title)
- And axis limits can be set using:
  - .set_xlim(xmin, xmax)
  - .set_ylim(ymin, ymax)
- Tick marks and labels are set using:
  - .set_xticks(ticks)/.set_yticks(ticks)
  - .set_xticklabels(labels)/.set_yticklabels(labels)

- Python has a huge number of high quality libraries that allow us to perform almost any operation we might be interested in

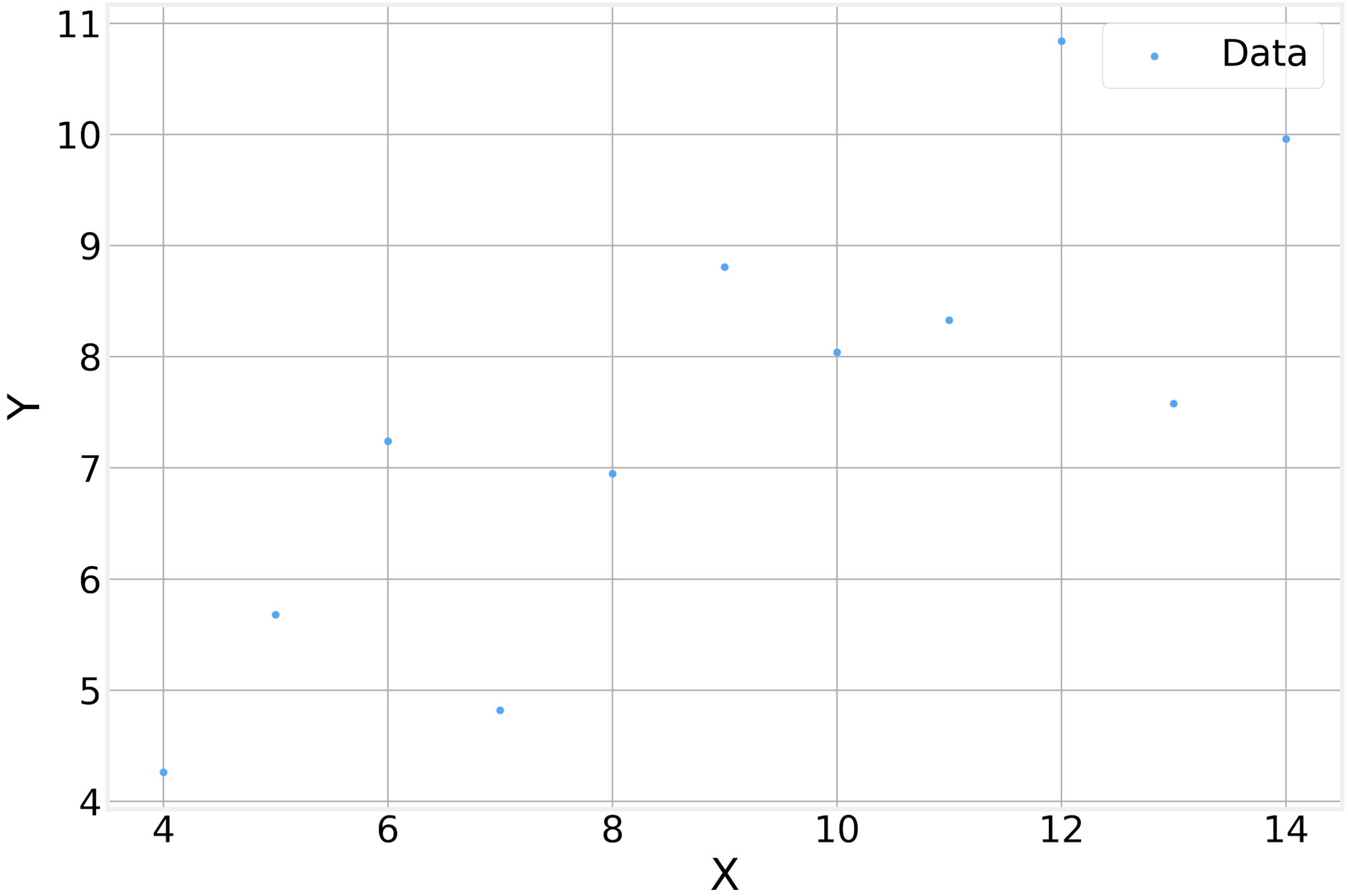
- scikit-learn is a state of the art library for Machine Learning, including

    - Linear Regression
    - Logistic Regression
    - Optimization
    - Support Vector Machines
    - Clustering
    - Neural Networks
    - Decision Tree
    - etc…

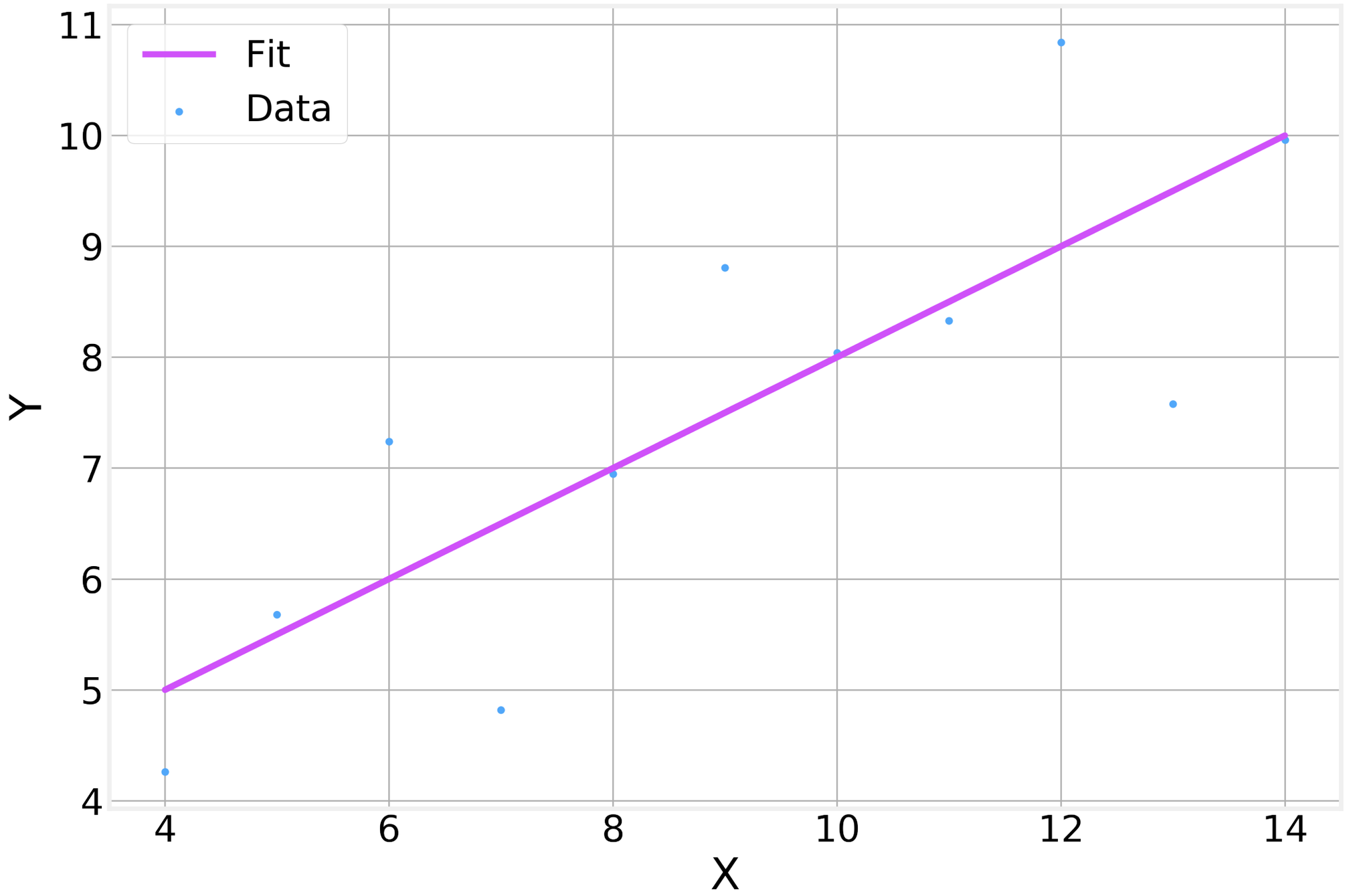
Linear Regression

- sklearn provides a LinearRegression object under the sklearn.linear_model module
- The LinearRegression object provides an important methods:
  - .fit(X, y) - determine the parameter values for the model. The dimensions of X determine the number of parameters computed
- After fitting the model, we can also use:
  - .predict(X) - Calculate the predicted value for any input values X
  - .coef_ - returns the computed parameters of the model
  - .intercept_ - returns the computed intercept value
- The fit()/predict() model is used throughout all of sklearn's models making it easy to explore various ML models.

Linear Regression



scikit learn Linear Regression



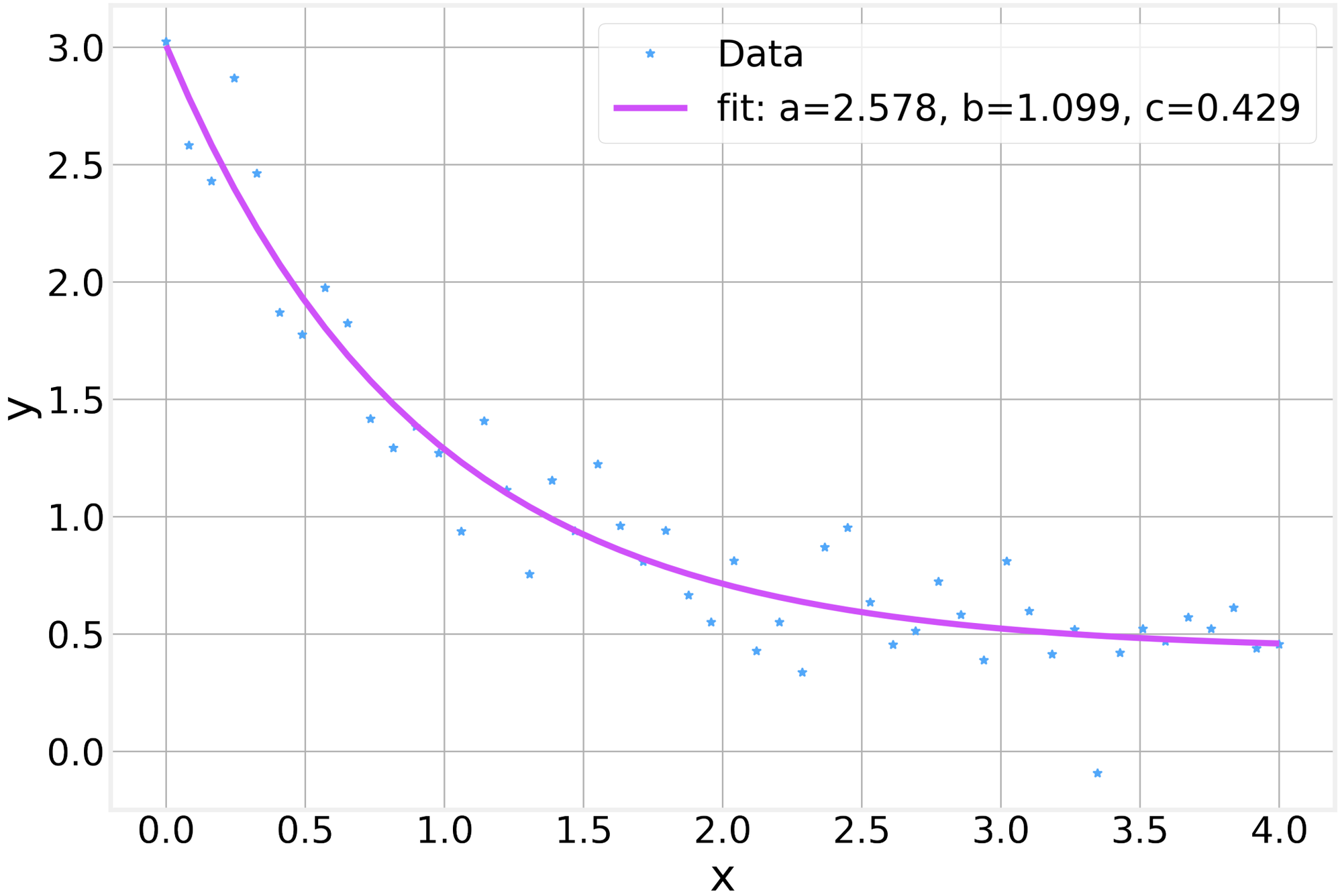
SciPy

- SciPy is a state of the art library for scientific computation.
- Together with numpy, matplotlib, pandas, etc it is part of the Sci-P umbrella project
- Includes functionality for:
    - Special Functions
    - Integration
    - Optimization
    - Signal Processing
    - Linear Algebra
    - Statistics
    - etc

SciPy Curve fitting

- The scipy library provides a curve_fit() function within the scipy.optimize module
- curve_fit() takes several fundamental arguments:
  - func - The function to fit
  - xdata - the x values
  - ydata - the y values
  - p0=None - the initial guess of the parameters.
- curve_fit() figures out the number of parameters to fit from the signature of func. If p0 is set, the number of arguments to func

SciPy Curve fitting



Code - Simple Data Modeling

https://github.com/DataForScience/Excel

Lesson IV:
Manipulating Excel Spreadsheets

Python packages

- There are several Python packages that allow you to read, write and manipulate Excel files, with varying degrees of ease and compatibility:
  - pandas - basic reading and writing
  - xlrd - library for reading data and formatting information from Excel files, whether they are .xls or .xlsx files - https://xlrd.readthedocs.io/en/latest/
  - xlrt - library for writing data and formatting information to older xls Excel files - https://xlwt.readthedocs.io/en/latest/
  - xlutils - a collection of utilities for working with Excel files, built on top of xlrd and xlwt - https://xlutils.readthedocs.io/en/latest/
  - pyexcel - unified application programming interface to read, manipulate and write data in various excel formats - http://docs.pyexcel.org/en/latest/
  - openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files - https://openpyxl.readthedocs.io/en/stable/

Python packages

- There are several Python packages that allow you to read, write and manipulate Excel files, with varying degrees of ease and compatibility:
  - pandas - basic reading and writing
  - xlrd - library for reading data and formatting information from Excel files, whether they are .xls or .xlsx files - https://xlrd.readthedocs.io/en/latest/
  - xlrt - library for writing data and formatting information to older xls Excel files - https://xlwt.readthedocs.io/en/latest/
  - xlutils - a collection of utilities for working with Excel files, built on top of xlrd and xlwt - https://xlutils.readthedocs.io/en/latest/
  - pyexcel - unified application programming interface to read, manipulate and write data in various excel formats - http://docs.pyexcel.org/en/latest/
  - openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files - https://openpyxl.readthedocs.io/en/stable/

Most features and the best documentation

openpyxl

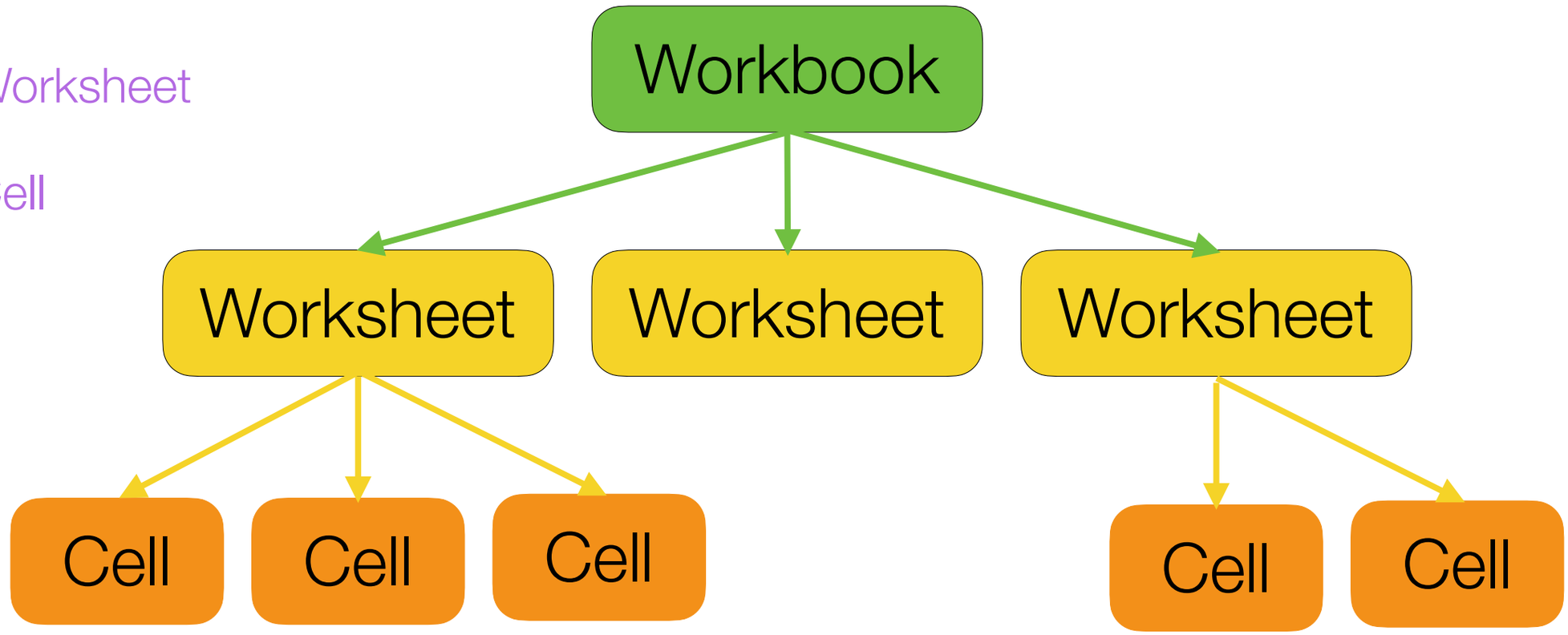
openpyxl.readthedocs.io

- openpyxl provides a complete view of the entire Excel workbook.
- Originally created as a port of the PHPExcel library to Python
- It fully supports Excel 2010 xlsx/xlsm/xltx/xltm
- In particular, it allows us to:
  - Open existing Excel files
  - Create/modify/delete sheets from files
  - Duplicate sheets within a workbook
  - Embed images, charts and formulas in to sheets
  - Stylize and format sheets.

openpyxl

openpyxl.readthedocs.io

- openpyxl relies on three fundamental concepts, represented as Python objects:
  - Workbook
  - Worksheet
  - Cell



- Workbooks can contain any number of sheets allowed by the specific format, and Worksheet objects can contain any number of Cells allowed.

Workbook

- An empty Workbook can be created simply by calling Workbook(). This will be an empty workbook that exists only in memory. To save it to disk, use .save(filename).

- To load a Workbook from a file on disk, we call load_workbook(filename). This function accepts a few optional parameter that are useful to understand:

  - read_only=False - Prevent edits. This mode is optimized for reading.

  - keep_vba=False - Prevent VBA content from being corrupted (it can not be used)

  - data_only=False - Return only the stored values or the formulas. The default is to return the formulas, set to True to return the current values instead.

- The list of all Worksheets that are part of the Workbook is available in the .sheetnames list, while the .worksheets member list contains the respective sheet objects.

- You can get the currently active worksheet (equivalent to the currently selected sheet in Excel) by accessing the .active field of the Workbook object.

Workbook

- The Workbook object provides a dict like interface to access Worksheets by name.
- To save a Workbook we use .save(filename)

Worksheet

- To create a new worksheet we use .create_worksheet(title=None, index=None). Both arguments are optional, the default names will be Sheet, Sheet1, Sheet2, etc… index provides the location where to insert the new sheet, and by default each new sheet will be place at the end of the workbook.

- To duplicate a Worksheet within the Workbook, you use .copy_worksheet(name).

- The name of the Worksheet can be assigning it to the .title member field.

- The Worksheet contains several useful fields:
  - min_column/max_column/min_row/max_row - column and row ranges
  - dimensions - reference of the upper-left and lower-right cells.

Worksheet

- And useful methods:
  - .insert_rows(idx)/.insert_columns(idx) - insert a row/column BEFORE position idx
  - .delete_rows(idx)/.delete_columns(idx) - delete a row/column AT position idx
    - amount=1 - All these functions take an optional parameter amount to specify how many row/column to add/delete
      - row/column indices start at 1
- The Worksheet object provides a dict like interface to access Cells by name (“A1”, “B3", etc).

Worksheet

- We can efficiently iterate over rows/columns using .iter_rows()/iter_columns(). These methods take 5 arguments, all of which are optional
  - min_row=None - Starting row (inclusive)
  - max_row=None - Ending row (inclusive)
  - min_col=None - Starting column (inclusive)
  - max_col=None - Ending column (inclusive)
  - values_only=False - Return the current numerical values (True) or the corresponding formula (False)
- By default, all rows/columns are returned as tuples of Cell objects.
- We can append() rows to an existing Worksheet, by providing tuples of values.
- When a WorkSheet is first created, it contains no Cells. Cells are dynamically created when they are accessed for the first time.

Cell

- We can access a specific Cell directly from the Worksheet object:
  - sheet[cell_ref] - where cell_ref is the Excel cell reference
  - Assigning a value to this object, changes the value of the cell
- Another alternative to set the value of a cell is to use the .cell(row, column, value) method.
- Ranges can be specified using Excel slice notation (similar to Python notation):
  - sheet["A3:B5"] - returns the 3x2 matrix of cells between A3 and B5 (inclusive) as a tuple of tuples of Cell objects.
  - Ranges can't be used for assignment (tuples are immutable)
- Cell objects have several useful fields:
  - .value - returns the value of the cell. Also useful for assignment
  - .row/.column - returns the row/column coordinates of the Cell

Code - Basic Excel Spreadsheets

https://github.com/DataForScience/Excel

Formulas

- Cells can contain formulas that compute values.
- We can simply assign a formula as a string to a specific cell
- openpyxl doesn't evaluate formula values
- openpyxl.utils.FORMULAE - A frozenset containing a list of 352 known Excel function names that you may use as a quick reference.

Images

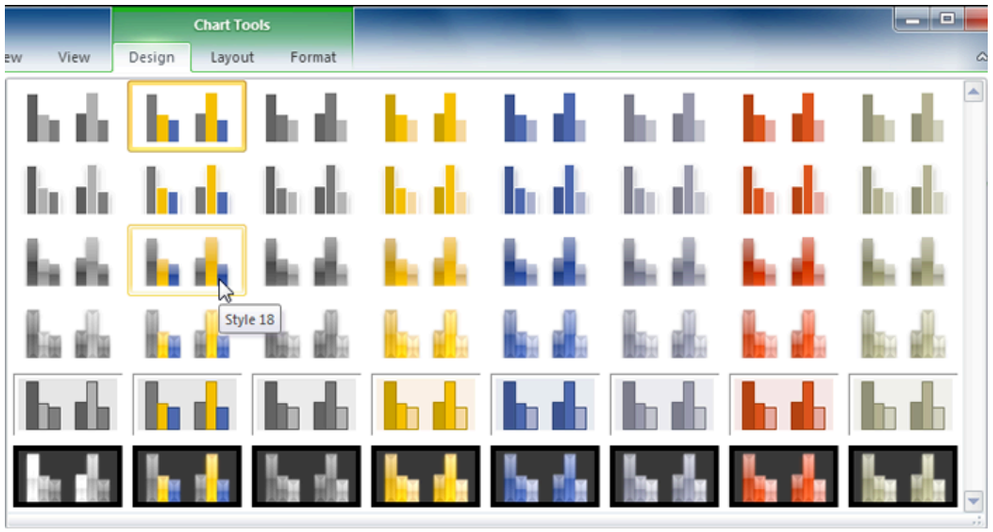
- Adding images to a Worksheet is a two step process:
  - load the image using the Image() object in the openpyxl.drawing.image module
  - place the image into a Worksheet using .add_image(img, anchor=None) where img is the Image object and anchor is the Cell where to place the top left hand corner of the image
- Image natively support 'gif', 'jpeg', and 'png' images. it will attempt to convert all other formats to 'png'.
- openpyxl provides very limited support for images beyond simply placing them in a specific location. It's up to you to make sure the image is the right size to display properly

Charts

- Excel supports many different kinds of charts for visualization
- openpyxl allows us to add charts to a Worksheet in a simple and convenient way.
- You can create AreaChart, AreaChart3D, BarChart, BarChart3D, BubbleChart, DoughnutChart, LineChart, LineChart3D, PieChart, PieChart3D, ProjectedPieChart, RadarChart, ScatterChart, StockChart, SurfaceChart, SurfaceChart3D
- The process to create a Chart requires several steps:
  - Create a Chart object and configuring it with labels, legends, etc
  - Define the data to use, by creating a Reference object
  - Pass the Reference object to to the Chart object using add_data()
  - Add the Chart to a Worksheet (similarly to an Image)
- Chart and Reference Objects live inside the openpyxl.chart module

Charts

- All Chart objects contain fields that allow us to customize them. In particular:
  - .title - The main title for the chart
  - .style - The style to use, represented as an integer.
  - .x_axis.title - x-label
  - .y_axis.title - y-label



  - .x_axis.scaling.min/x_axis.scaling.max - minimum and maximum limits for the x-axis
  - .y_axis.scaling.min/y_axis.scaling.max - minimum and maximum limits for the y-axis

Reference

- A Reference object requires 5 parameters:
  - worksheet - A reference to the Worksheet where the data lives
  - min_col/max_col/min_row/max_row - the coordinates delimiting the cells containing the data
- .add_data(data) adds the Reference object to the Chart we are creating.This method takes two optional parameters:
  - from_rows=False - Is the data in row format (instead of the more common column format)
  - titles_from_data=False - Does the Reference include the column/row names to use for each Series?

Series

- When we add the Reference to our Chart, it automatically creates a Series of data points for each column in the Reference object.

- Series objects represent individual datasets within the Chart. and can reaccessed from the .series list

- Series objects contain a wealth of methods and fields to customize the appearance of the respective dataset, In particular:

  - .smooth - Boolean to specify wether the line should be smooth or not
  - .marker.symbol - The symbol to use for each point, (‘circle’, 'dash', 'diamond', 'dot', 'plus', 'square', 'star', 'triangle', 'x', ‘auto’)
  - .graphicalProperties.line.noFill - Boolean specifying wether to draw a line or not
  - .marker.graphicalProperties.solidFill - Hexcode for the marker color
  - .graphicalProperties.line.width - Line width
  - etc…

Series

https://openpyxl.readthedocs.io/en/stable/styles.html

- By default, the Series contain just the y values and using the vector position for the x-value.
- To use specific values for the x-axis, we can create a Series "by-hand'
- Series(ref_y, ref_x) - where ref_x and ref_y are References to the x-values and the y-values, respectively
- Series can be added to a Chart using .series.append(). Chart behaves as if it were a list of Series

openpyxl has extensive support to styles.
You should explore the documentation!

Pandas support

- openpyxl was designed with pandas in mind, and provides us with some pandas specific functionality to help us integrate both.

- The function dataframe_to_rows(), inside the module openpyxl.utils.dataframe provides a convenient way to convert DataFrame values to Worksheet rows by returning an integrator over the DataFrame rows.

    - index=True - Include the DataFrame index
    - header=True - Include the DataFrame column names

Code - Advanced Excel Spreadsheets

https://github.com/DataForScience/Excel

Events

www.data4sci.com/newsletter

http://paypal.me/data4sci

Natural Language Processing (NLP) for Everyone

Sept 16, 2020 - 5am-9am (PST)

Graphs and Network Algorithms for Everyone

Oct 7, 2020 - 5am-9am (PST)

Why and What If – Causal Analysis for Everyone

Oct 16, 2020 - 5am-9am (PST)

Natural Language Processing (NLP) from Scratch

http://bit.ly/LiveLessonNLP - On Demand