Basic programming for drug discovery

Python

Margriet Palm, Brandon Bongers, and Gerard van Westen

08/11/2018

Universiteit Leiden
The Netherlands

Recap - Basic Python

- Jupyter notebooks
  - Use notebooks as a python environment
  - Format markdown cells
- Data types and containers
  - strings, floats, integers, etc.
  - lists, tuples, etc.
- Using variables
- Basic Python syntax
  - strings and string formatting
  - printing variables
  - lists
  - control flow

Recap - Numerics

- Creating arrays
  - with fixed values
  - with random values
  - from file
- Indexing arrays to select specific data
- Running mathematical and statistical operations on arrays
- Write array to file

Recap - Pandas

- Curate data
- Select columns by names
- Select rows by conditions
- Compute statistics on full dataframe or selection
- Group data
- Read dataframe from file
- Store dataframe to file

Recap - Plotting

- Plot data from a numpy array with matplotlib
  - Single line plot
  - Multi-line plot
  - Bar chart
  - Error bars
- Annotate plots
  - Axis labels and title
  - Legend
  - Modify ticks
- Exporting plots to file
- Plotting with Seaborn and Pandas dataframes

Recap - data analysis pipeline

1. Import data
   - CSV data: `pd.read_csv`
   - Excel data: `pd.read_excel`
2. Analyze data
   - Select columns by name and rows based on conditions
   - Group data if needed
   - Compute statistics
3. Plot results
   - Use matplotlib for line plots
   - Use seaborn for statistic plots (boxplot, stripplot, etc.)
   - Use matplotlib to annotate

What next

- Continue programming!
- Develop good habits
- Explore and experiment!
- Follow other tutorials
- Learn how to solve problems
- Improve your toolkit
  - useful libraries
  - IDEs
  - version control

Good programming practices

- Why
  - Reliability: does my code what I think it's doing
  - Reproducibility: someone else (including future you) should be able to use the code
  - Maintainability: small changes should require little work
- How
  - Use functions instead of copy and pasting code
  - Use variables instead of hardcoding values
  - Use sensible variable/function names
  - Consistent style (see PEP8)
  - Document with someone else (or future you) in mind
  - Program together; review your colleagues code!
  - Use libraries!!

> *"An hour of searching for software libraries can save you days of programming."*
>
> — Margriet Palm (translated thesis proposition)

GPP - naming variables

Variable name should describe what it holds

- Bad:

```
a = np.random.random(100)

data1 = pd.read_csv('dna_data.csv')
data2 = pd.read_csv('rna_data.csv')
```

- Better

```
rand_nrs = np.random.random(100)

df_dna = pd.read_csv('dna_data.csv')
df_rna = pd.read_csv('rna_data.csv')
```

`snake_case` (all lowercase connected by underscores) is preferred in Python.

GPP - annotation

Annotate complex code

- Bad:

```
a = 10   # variable a
s = 0    # variable s
# for loop
for i in range(a+1):
    s += i   # increase a with 1
```

- (a bit) Better

```
a = 10   # maximum number for range to sum up
s = 0    # placeholder for the sum
# sum up values from 0 to a
for i in range(a+1):
    s += i
```

GPP - Zen of Python

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

GPP - explicit is better than implicit

- Bad:

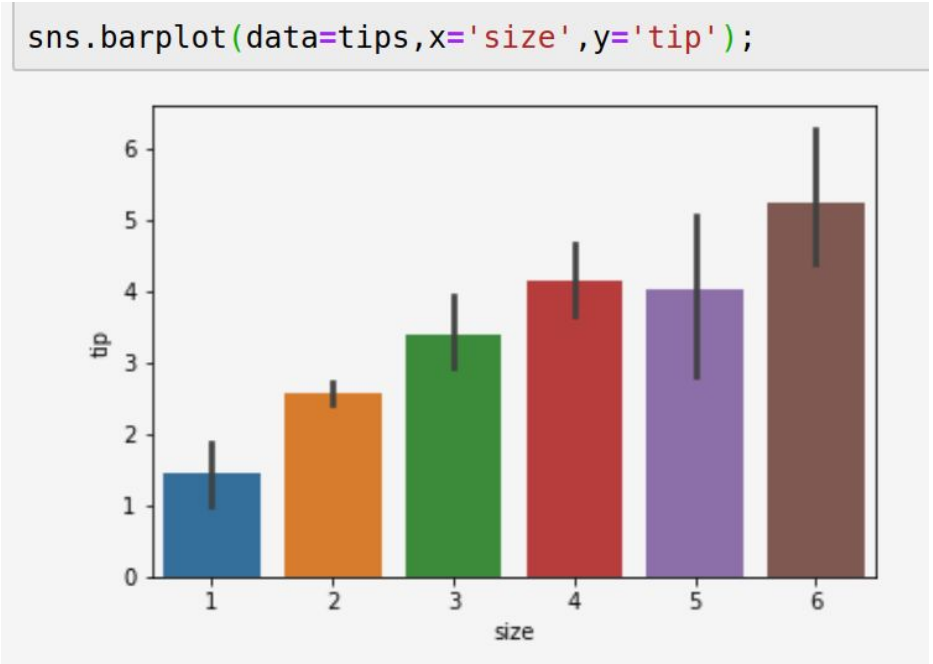
```
plt.plot(df.groupby('time').index,df.groupby('time')['value'])
```

- Good

```
g = df.groupby('time')
plt.plot(g.index,g['value'])
```

GPP - In the face of ambiguity, refuse the temptation to guess

```
sns.barplot(data=tips,x='size',y='tip');
```



Meaning of error bars is not clear

Resources

- The Python Tutorial
- Datacamp Python tutorials
- Python Data Science Handbook
- Effective Pandas
- Matplotlib tutorial
- Seaborn tutorial
- Syntax comparison Python, R and matlab
- GitHub for this course

How to solve problems

1. Is there really a problem
   - When using a notebook, restart kernel and rerun
   - When using interpreter, close and restart
   - When using a script, make sure you saved it
2. Identify what exactly is going wrong
   - Read error message (if any)
   - Print intermediate results
3. Search for similar issues online
   - Usually, this will bring you to stack overflow
   - Read the comments under the original question
   - Read the answers and look at score and comments
   - Try to understand the solution before implementing
4. Before asking for help if you can't find the answer.
   - Boil down your code to the essentials of the problem
   - Create a minimal working example (MWE) and test this
   - Be nice to the people that are trying to help you

Toolkit - libraries

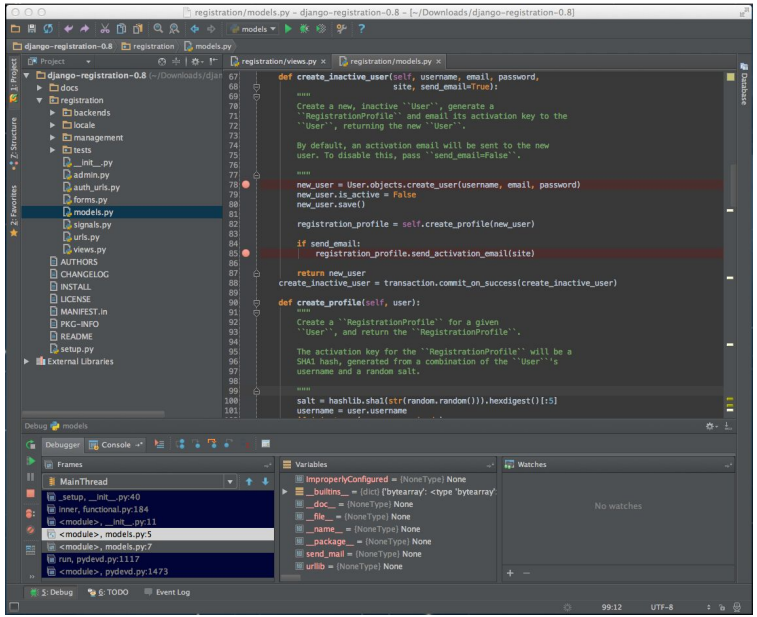
- Plotting
  - ggplot: similar to ggplot package for R
  - interactive plotting:
    - plotly (example) - also works with R
    - bokeh (example)
- Statistics
  - Scipy
  - Statsmodels
- Machine learning
  - scikit-learn
- Image processing:
  - Pillow
- Image analysis:
  - mahotas
  - scikit-image

Toolkit - IDEs

Wikipedia: “An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development.”

Why use an IDE:

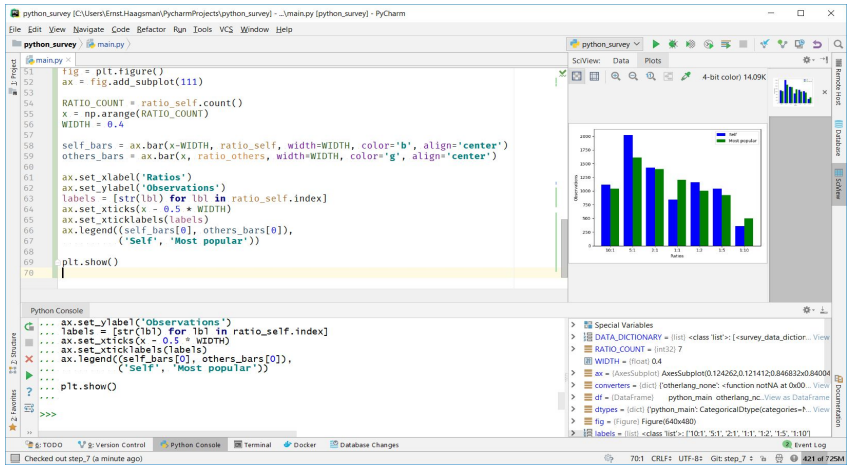
- Project oriented display of code
- Syntax highlighting
- Autocompletion
- Syntax check for style
- Integration with version control tools
- Debugging tools
- You look like a *real* programmer
- and much more



Toolkit - Suggested python IDEs

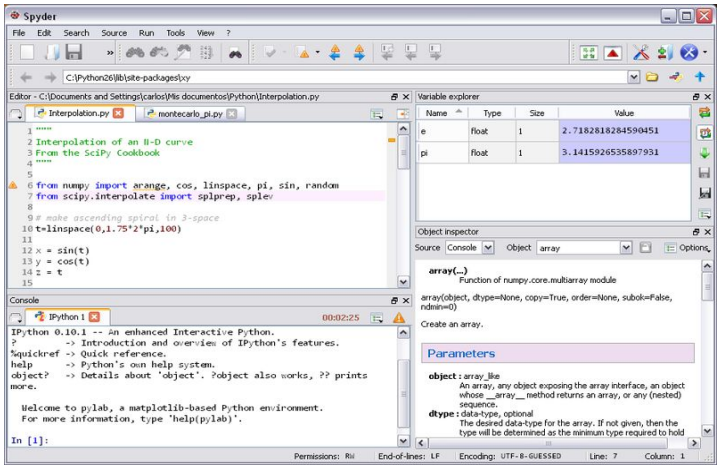
PyCharm

- Developer mode
- Scientific mode (below)
- Free community version + pro version free for academics



Spyder

- Similar to RStudio
- Included in anaconda
- free



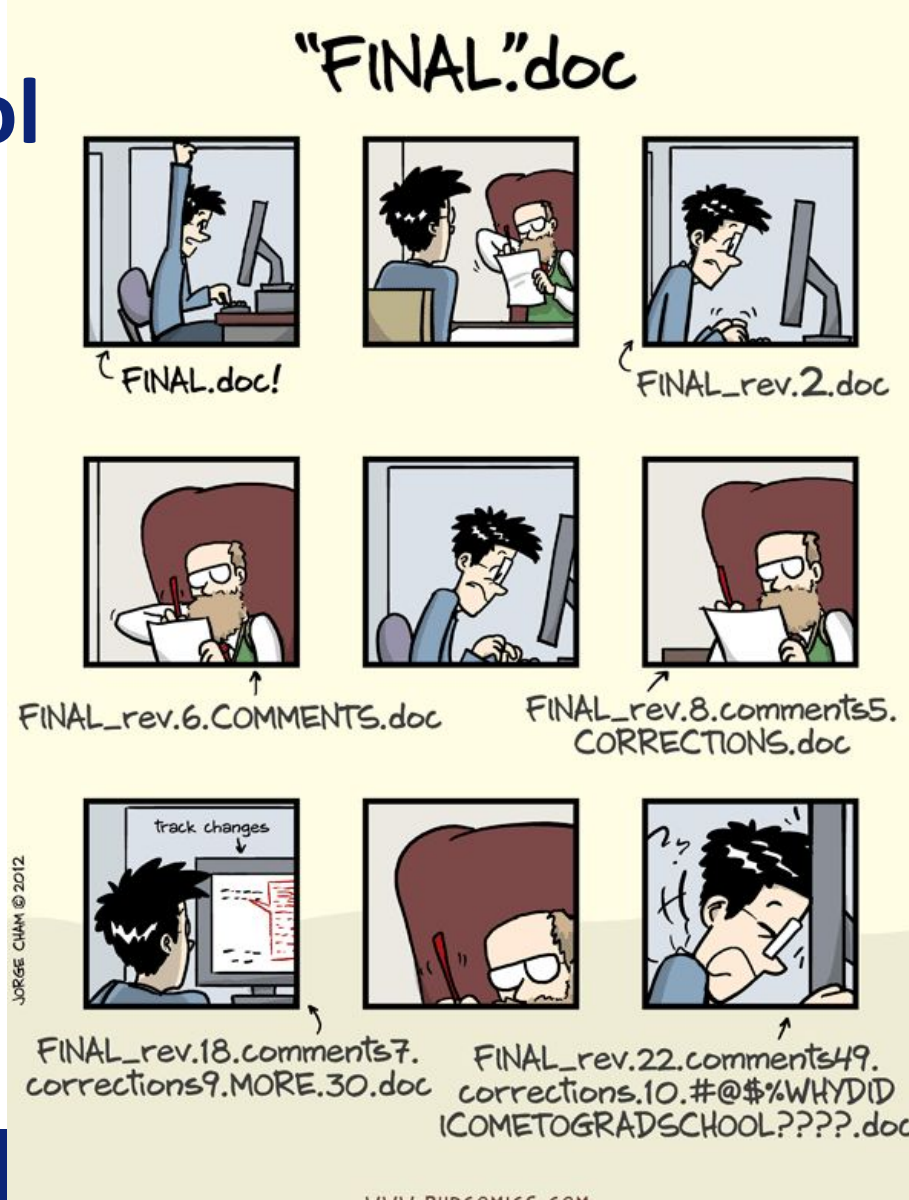
Jupyter notebook and Jupyterlab are also very good tools for data analysis; you don't have to use an IDE.

Toolkit - version control

- Keep track of changes and **why** they were made
- Merge code from multiple developers
- Develop features without breaking the original code

GitHub

GitLab



Case study

- See GitHub
- Description:
  - Data file with data for a series of experiments on cell migration and proliferation
  - Use your Python skills to analyze the dataset:
    - which treatments in which cell line affect migration/proliferation
  - Collect your findings in a nice Jupyter notebook, which should contain
    - different kinds of plots
    - multiple plots in one figure
    - a written discussion of the results
- You need:
  - Anaconda on your own computer
  - Seaborn (install via anaconda)
- Deadline: 30-11-18