



Current Topics in Media Computing and HCI

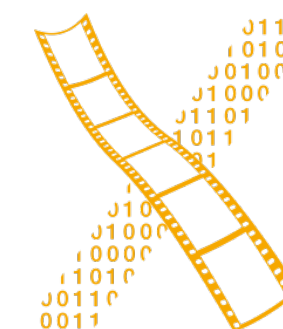
Data Science Programming

Krishna Subramanian, M.Sc.

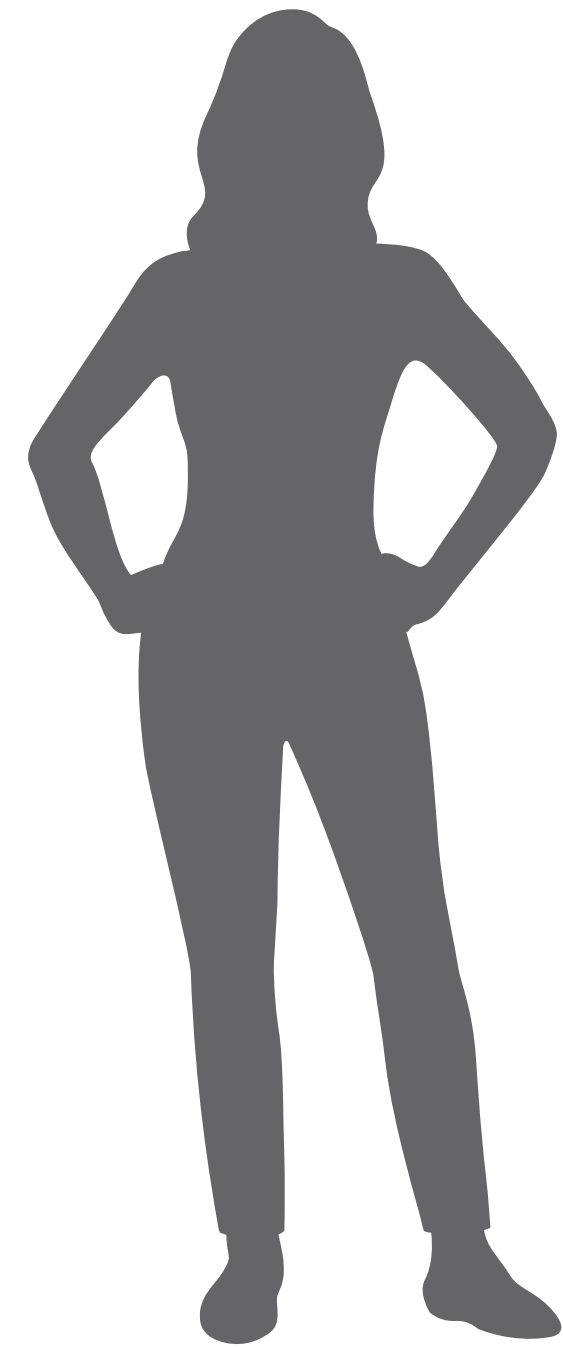
Media Computing Group
RWTH Aachen University

Summer Semester 2020

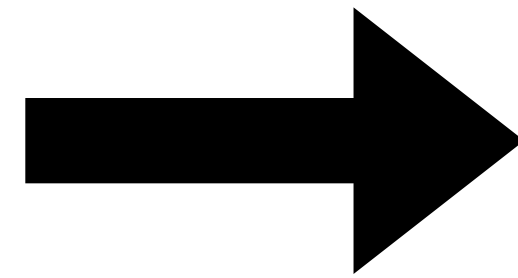
<https://hci.rwth-aachen.de/cthci>



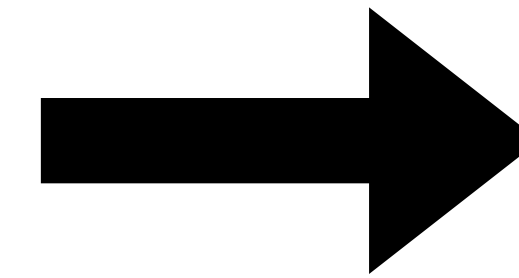
RWTHAACHEN
UNIVERSITY



Data scientist



Programming tools



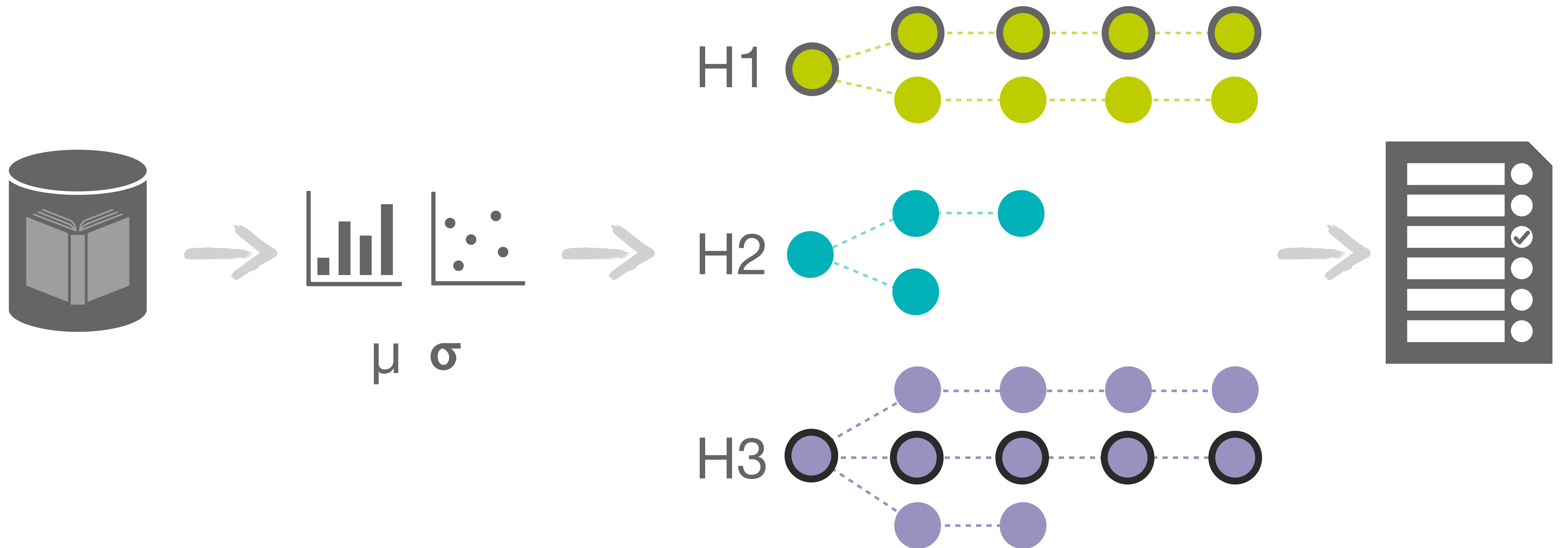
Data science

Data Science



- How we extract knowledge from data
- Example applications: Targeted advertising/recommendations in Amazon/Netflix, validate research findings, and train a robot to detect humans
- What is so special about data science?
 - **Open-ended, iterative workflow** that involves a lot of backtracking
 - Usually involves *deliberate* **bad programming practices** like writing non-modular code, not using version control, and code hoarding

Data Science Workflow



[Subramanian et al., TRACTUS: Understanding and Supporting..., 2020]

Open-ended: Goals were not predefined, but were defined (and modified) during analysis

Iterative and involves backtracking: Previous analysis is revisited during analysis and afterwards when writing the final version of source code

Programming Practice in Data Science

- In data science, the focus is the end goal (results, findings) but not the process
 - Contrast this to software engineering, where the process (source code) also needs to be well documented, run fast, be secure, be memory efficient, etc.

[Kery et al., Exploring Exploratory Programming, 2017]

Messy Code

- *“I know how to write code. And I know that I could write functions to reuse functions and I could try to modularize things better, and sometimes I just don't care because why am I going to put effort in that if I'm not going to use it again?”*
- Leads to source code that is hard to re-use, navigate, and understand

```
# read the CSVs – each DV is a separate file
measure =
read.csv("Measurements_a_bit_clean.csv", sep =
";")

#apply the change to the hours.
measure <- within(measure, Time[(Schema == 1 &
IsHours == 1 & Time >= 6)|(Schema == 1 & IsHours
== 0 & Time >= 30)|(Schema == 2 & Time >= 10)|
(Schema == 3 & IsHours == 1 & (Time != 12&Time !=
9&Time != 6&Time != 3))|(Schema == 3 & IsHours ==
0 & (Time != 0&Time != 45&Time != 30&Time !=
15))] <- 'difficult')
measure <- within(measure, Time[Time !=
"difficult"] <- 'easy')

# change the column type of the experimental
design (expansive vs. constrictive)
measure$User <- as.factor(measure$User)
measure$Time <- as.factor(measure$Time)
measure$Schema <- as.factor(measure$Schema)
measure$IsHours <- as.factor(measure$IsHours)
```

[Kery et al., Variolite: Supporting Exploratory ..., 2017]

Informal Versioning and Code Hoarding

- Remember that data science involves iteration and backtracking
- This requires data scientists to keep all source code from explorations

```
537 print "hc policy distribution entropy: \n" + str
538 #print "2-fold CV Var: \n" + str(CV_variances)
539
540 #print "Stdev of (V_M under h^f, avg over states
541 #print "Stdev of (V_M under h^c, avg over states
542 #print "Average V_M under h^f: \n" + str(V_M_hf)
543 #print "Average V_M under h^c: \n" + str(V_M_hc)
544
545 fig = plt.figure(figsize=(9,4))
546 ax = fig.add_subplot(1,1,1)
547 ax.set_xticks(D_sizes)
548 plt.plot(D_sizes, [V_star_avgState]*len(D_sizes))
549 #plt.plot(D_sizes, AvgState_V_hfs)
550 plt.errorbar(D_sizes,AvgState_V_hfs,yerr=Std_Vs_
551 #plt.plot(D_sizes, AvgState_V_hcs)
552 plt.errorbar(D_sizes,AvgState_V_hcs,yerr=Std_Vs_
553 plt.xlim(0,max(D_sizes))
554 plt.xlabel('|D|')
555 plt.ylabel('mean(V)')
556 plt.grid()
557 plt.show()
558
559 ...
560 fig = plt.figure(figsize=(10,4))
561 ax = fig.add_subplot(1,1,1)
562 ax.set_xticks(D_sizes)
```

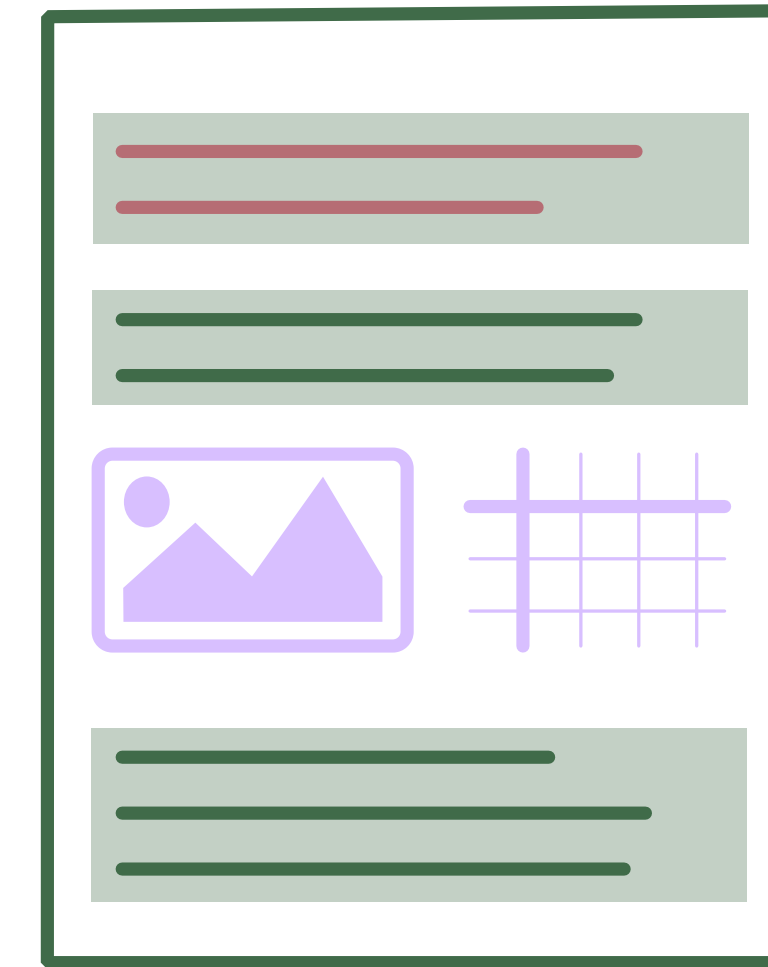
[Kery et al., Variolite: Supporting Exploratory ..., 2017]

Programming Tools

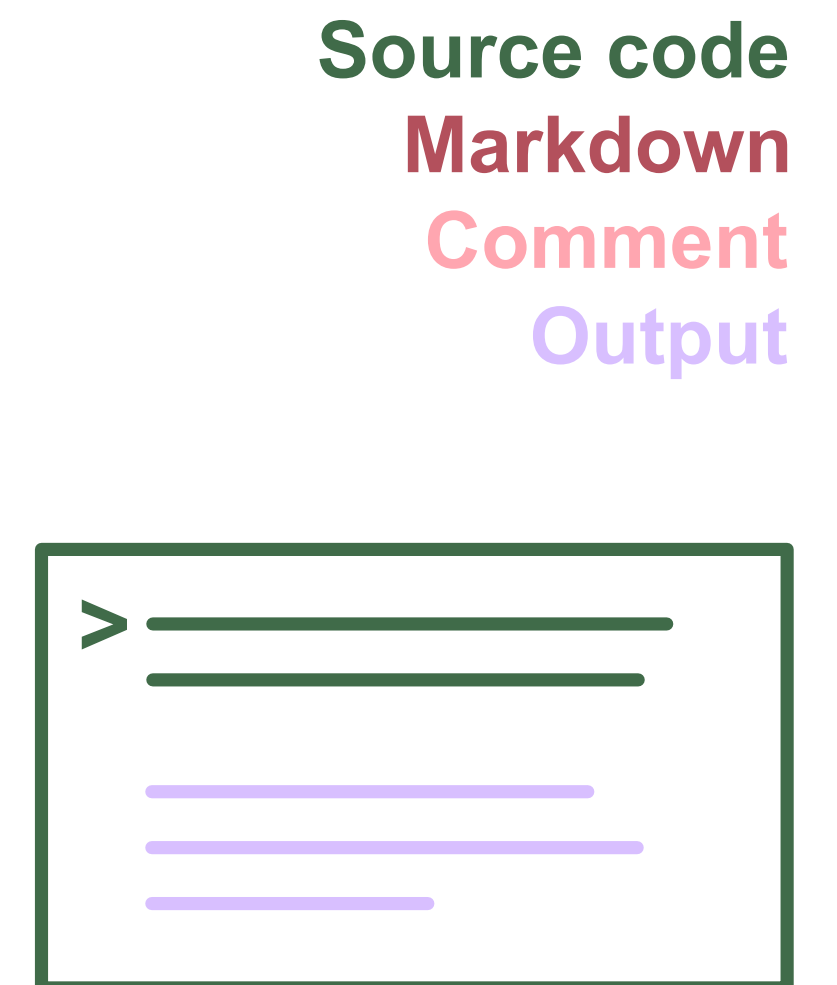
- Scripting languages are very common: R, Python, and MATLAB
- IDEs for these languages offer three interfaces to program in
 - Scripts
 - Computational notebooks
 - Consoles



Script



Computational Notebook

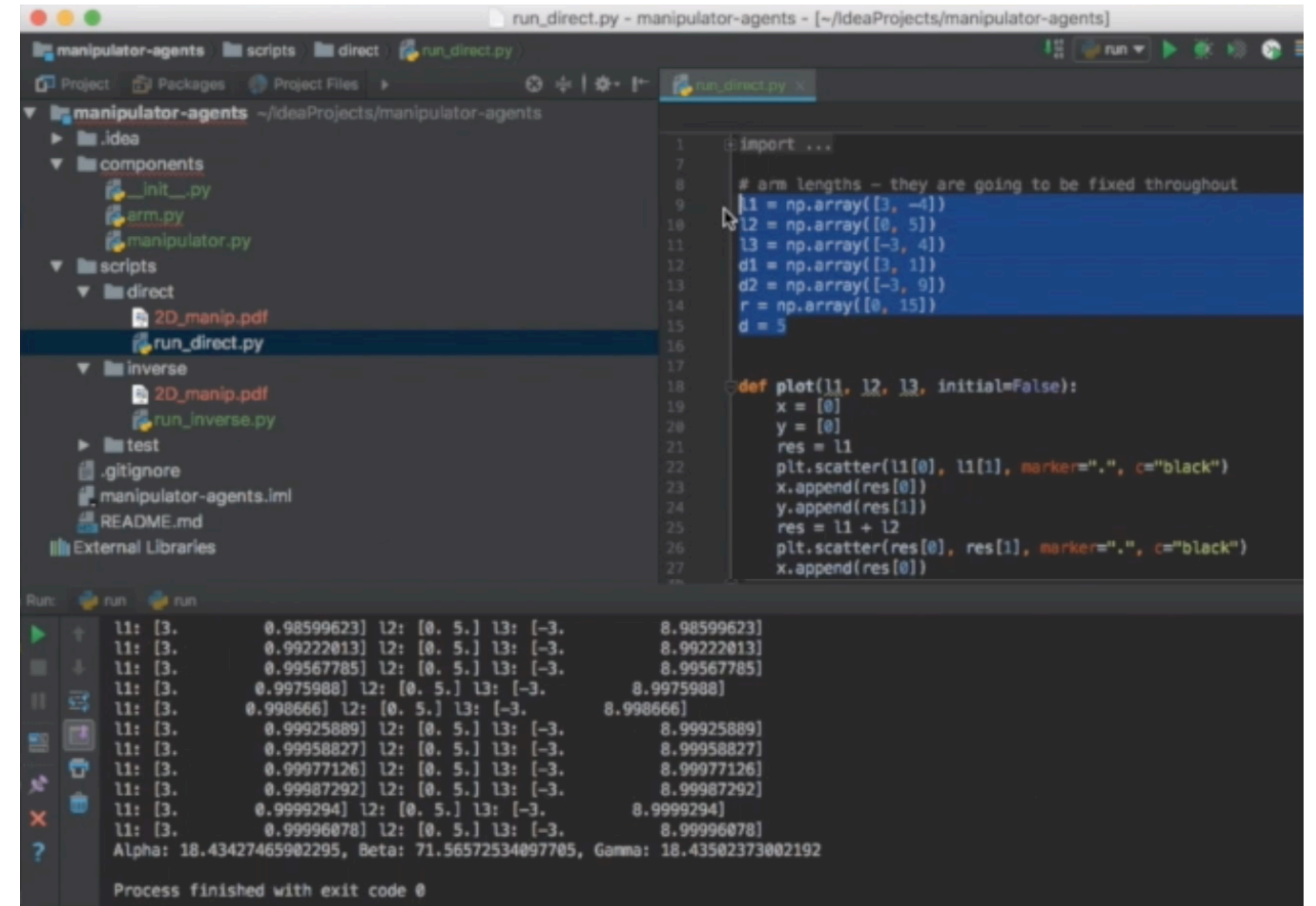


Interactive Console

[Subramanian et al., Casual Notebooks and ..., 2020]

Scripts

- Traditional way to store source code
- Supports complete/partial execution of source code (via selection)
- Output is shown in a separate window or the console window

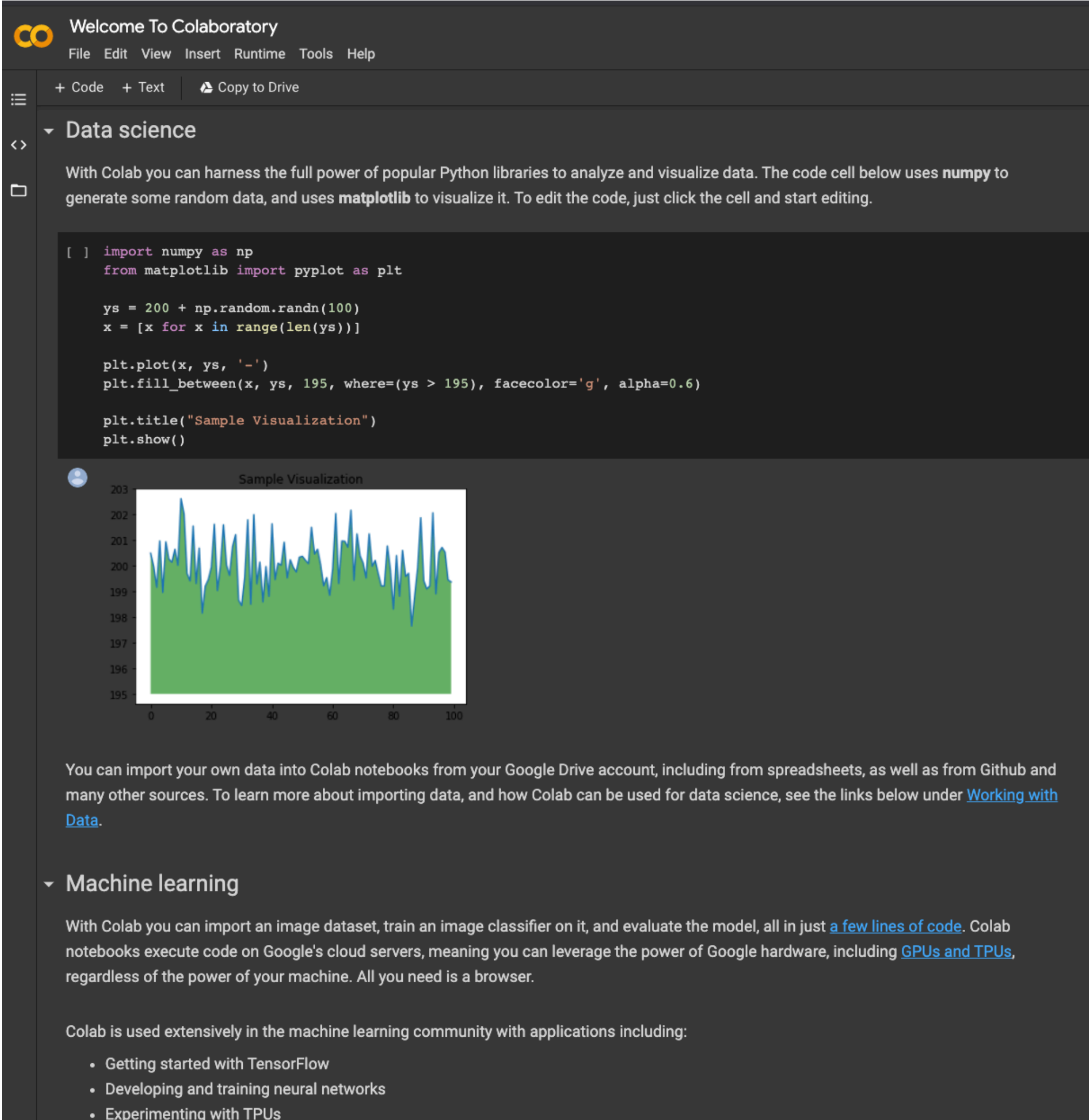


The screenshot shows an IDE window titled 'run_direct.py - manipulator-agents - [~/IdeaProjects/manipulator-agents]'. The left sidebar displays a project tree with folders 'manipulator-agents', 'components', 'scripts', 'inverse', and 'test'. The 'scripts' folder is expanded, showing '2D_manip.pdf' and 'run_direct.py'. The main editor displays the code for 'run_direct.py', which includes imports, array definitions, and a plot function. The bottom panel shows the execution output, displaying a series of numerical values for l1, l2, and l3, followed by the calculated Alpha, Beta, and Gamma values, and a final message 'Process finished with exit code 0'.

```
1 import ...
2
3 # arm lengths - they are going to be fixed throughout
4 l1 = np.array([3, -4])
5 l2 = np.array([0, 5])
6 l3 = np.array([-3, 4])
7 d1 = np.array([3, 1])
8 d2 = np.array([-3, 9])
9 r = np.array([0, 15])
10 d = 5
11
12 def plot(l1, l2, l3, initial=False):
13     x = [0]
14     y = [0]
15     res = l1
16     plt.scatter(l1[0], l1[1], marker=".", c="black")
17     x.append(res[0])
18     y.append(res[1])
19     res = l1 + l2
20     plt.scatter(res[0], res[1], marker=".", c="black")
21     x.append(res[0])
22     y.append(res[1])
23
24 Run: run run
25 l1: [3. 0.98599623] l2: [0. 5.] l3: [-3. 8.98599623]
26 l1: [3. 0.99222013] l2: [0. 5.] l3: [-3. 8.99222013]
27 l1: [3. 0.99567785] l2: [0. 5.] l3: [-3. 8.99567785]
28 l1: [3. 0.9975988] l2: [0. 5.] l3: [-3. 8.9975988]
29 l1: [3. 0.998666] l2: [0. 5.] l3: [-3. 8.998666]
30 l1: [3. 0.99925889] l2: [0. 5.] l3: [-3. 8.99925889]
31 l1: [3. 0.99958827] l2: [0. 5.] l3: [-3. 8.99958827]
32 l1: [3. 0.99977126] l2: [0. 5.] l3: [-3. 8.99977126]
33 l1: [3. 0.99987292] l2: [0. 5.] l3: [-3. 8.99987292]
34 l1: [3. 0.9999294] l2: [0. 5.] l3: [-3. 8.9999294]
35 l1: [3. 0.99996078] l2: [0. 5.] l3: [-3. 8.99996078]
36 Alpha: 18.43427465902295, Beta: 71.56572534097705, Gamma: 18.43502373002192
37 Process finished with exit code 0
```

Computational Notebooks

- Cell-based programming
- Cells can be executed in a non-sequential order
- Output is shown immediately next to the cell that was executed
- Cells can also include Markdown commands (to describe the analysis process)



The screenshot displays the Google Colaboratory web interface. At the top, there's a 'Welcome To Colaboratory' header with a menu (File, Edit, View, Insert, Runtime, Tools, Help) and buttons for '+ Code', '+ Text', and 'Copy to Drive'. The left sidebar shows a file explorer with a 'Data science' folder selected. The main area contains a code cell with the following Python code:

```
[ ] import numpy as np
    from matplotlib import pyplot as plt

    ys = 200 + np.random.randn(100)
    x = [x for x in range(len(ys))]

    plt.plot(x, ys, '-')
    plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)

    plt.title("Sample Visualization")
    plt.show()
```

Below the code cell, the output is a line plot titled 'Sample Visualization'. The x-axis ranges from 0 to 100, and the y-axis ranges from 195 to 203. The plot shows a blue line representing the data points, with a green shaded area underneath it, indicating the region where the values are greater than 195. Below the plot, there is a paragraph of text explaining that data can be imported from Google Drive, spreadsheets, or GitHub, and a link to 'Working with Data'. Further down, there is a section titled 'Machine learning' which describes how Colab can be used for training models on Google's cloud servers, and lists some applications like TensorFlow, neural networks, and TPUs.

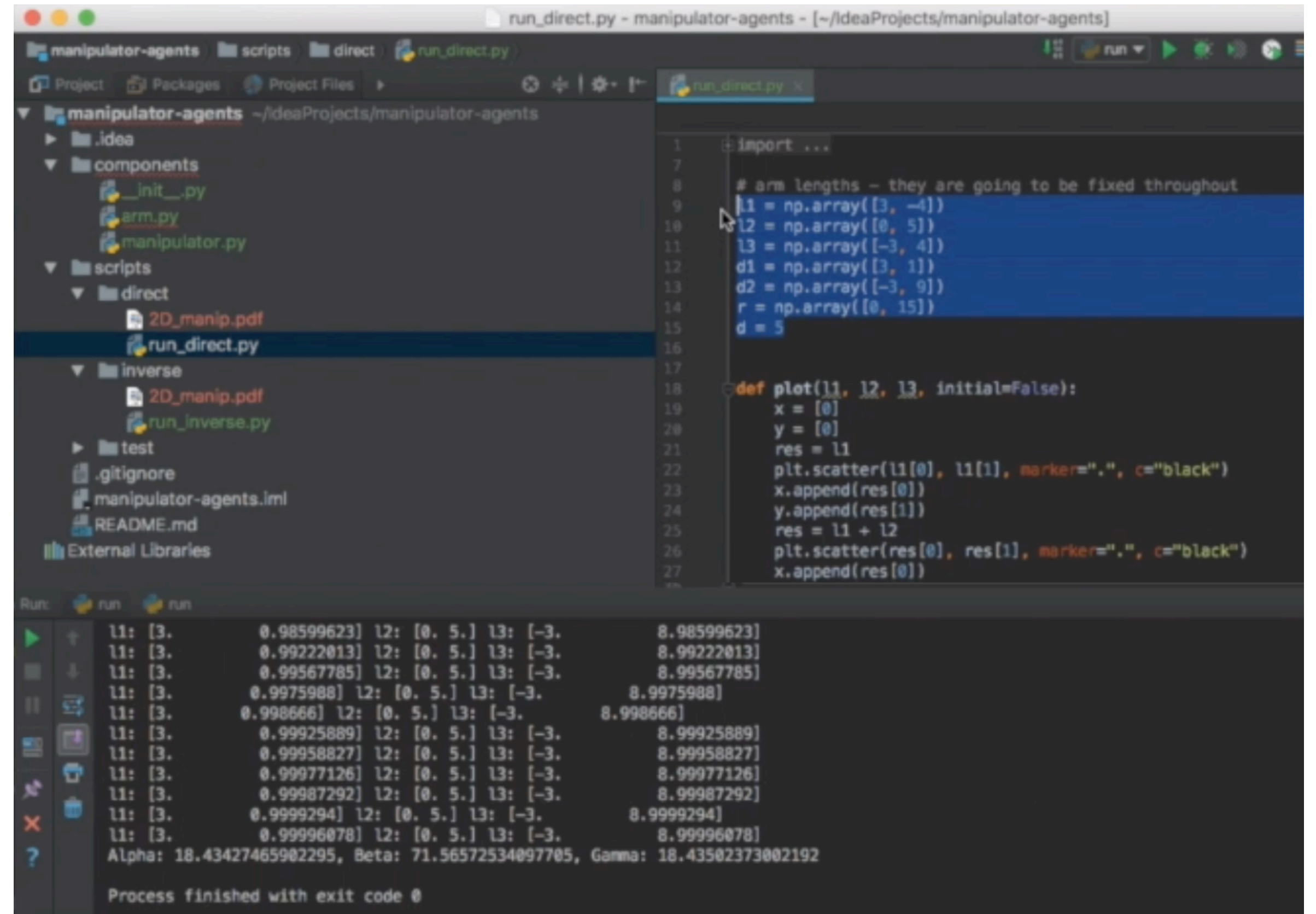
Scripts vs. Notebooks

| Data Science Task | Notebooks | Scripts |
|----------------------------------|-----------|---------|
| Experimentation | ✓ | |
| Refactor code | | ✓ |
| Present code | ✓ | |
| Share code | ✓ | |
| Execute from command line/on GPU | | ✓ |

[Subramanian et al., Casual Notebooks and ..., 2020]

Interactive Consoles

- Used mostly for secondary tasks like testing API, loading libraries, etc.
- However: Novice data workers who do not use notebooks tend to use consoles even for their primary data science work



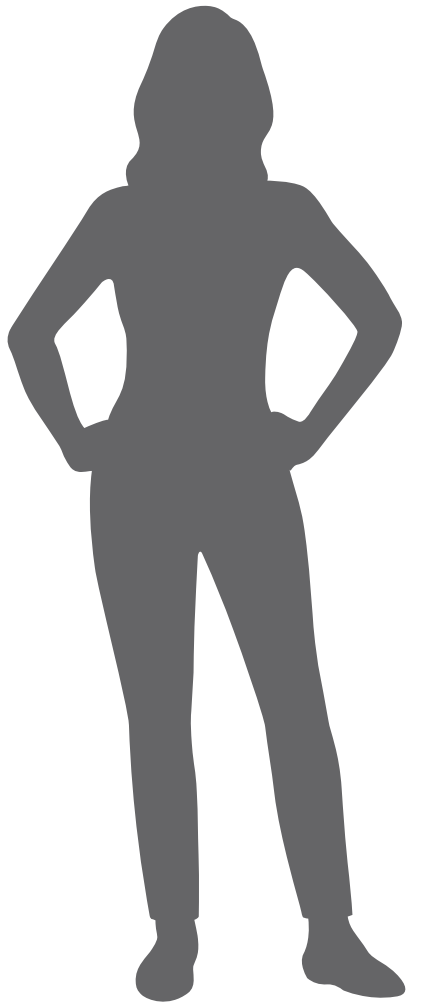
```
run_direct.py - manipulator-agents - [~/IdeaProjects/manipulator-agents]
manipulator-agents scripts direct run_direct.py
Project Packages Project Files run_direct.py
manipulator-agents ~/IdeaProjects/manipulator-agents
  .idea
  components
    __init__.py
    arm.py
    manipulator.py
  scripts
    direct
      2D_manip.pdf
      run_direct.py
    inverse
      2D_manip.pdf
      run_inverse.py
    test
      .gitignore
      manipulator-agents.iml
      README.md
  External Libraries

1 import ...
2
3 # arm lengths - they are going to be fixed throughout
4 l1 = np.array([3, -4])
5 l2 = np.array([0, 5])
6 l3 = np.array([-3, 4])
7 d1 = np.array([3, 1])
8 d2 = np.array([-3, 9])
9 r = np.array([0, 15])
10 d = 5
11
12 def plot(l1, l2, l3, initial=False):
13     x = [0]
14     y = [0]
15     res = l1
16     plt.scatter(l1[0], l1[1], marker=".", c="black")
17     x.append(res[0])
18     y.append(res[1])
19     res = l1 + l2
20     plt.scatter(res[0], res[1], marker=".", c="black")
21     x.append(res[0])
22
23 Run: run run
24 l1: [3. 0.98599623] l2: [0. 5.] l3: [-3. 8.98599623]
25 l1: [3. 0.99222013] l2: [0. 5.] l3: [-3. 8.99222013]
26 l1: [3. 0.99567785] l2: [0. 5.] l3: [-3. 8.99567785]
27 l1: [3. 0.9975988] l2: [0. 5.] l3: [-3. 8.9975988]
28 l1: [3. 0.998666] l2: [0. 5.] l3: [-3. 8.998666]
29 l1: [3. 0.99925889] l2: [0. 5.] l3: [-3. 8.99925889]
30 l1: [3. 0.99958827] l2: [0. 5.] l3: [-3. 8.99958827]
31 l1: [3. 0.99977126] l2: [0. 5.] l3: [-3. 8.99977126]
32 l1: [3. 0.99987292] l2: [0. 5.] l3: [-3. 8.99987292]
33 l1: [3. 0.9999294] l2: [0. 5.] l3: [-3. 8.9999294]
34 l1: [3. 0.99996078] l2: [0. 5.] l3: [-3. 8.99996078]
35 Alpha: 18.43427465902295, Beta: 71.56572534097705, Gamma: 18.43502373002192
36 Process finished with exit code 0
```

[Subramanian et al., Casual Notebooks and ..., 2020]

Data Scientists

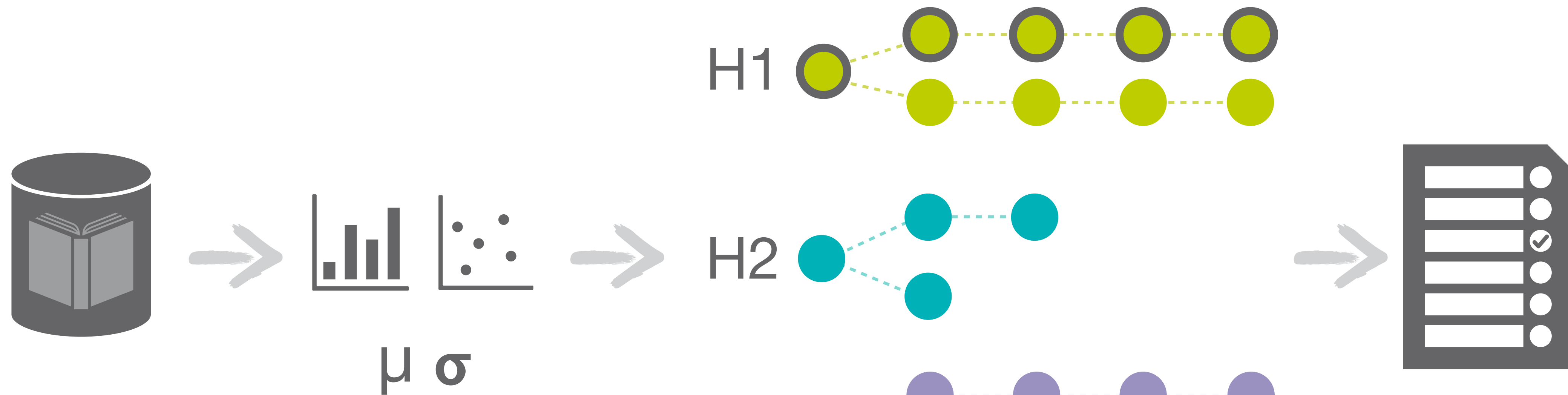
- “... people who understand how to fish out answers to important business questions from today’s tsunami of unstructured information.” [Davenport 2012]
- Data scientists are
 - impactful,
 - help make key decisions
- Several data scientists are not professionals, we call them “data workers”
 - do not have formal training in data science
 - may not have good programming practices



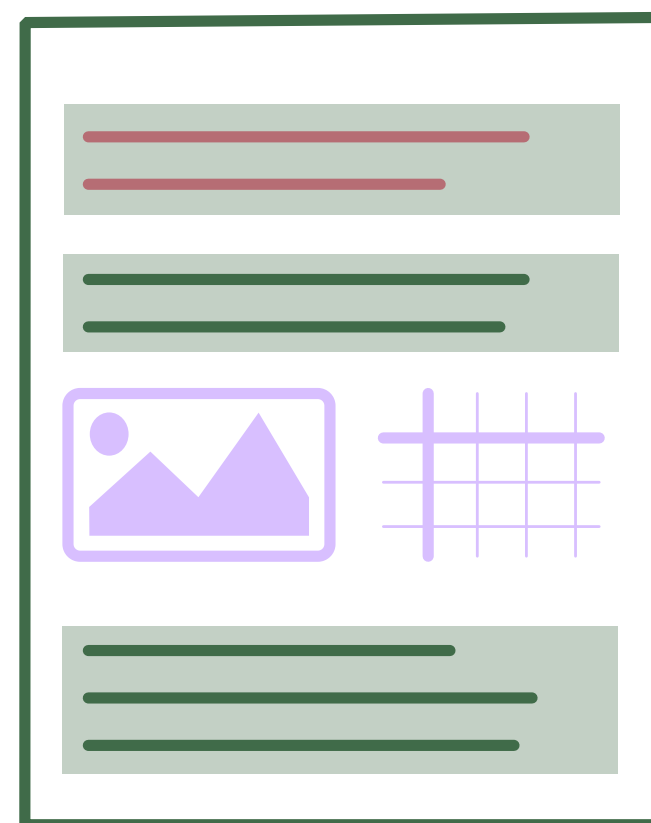
[Boukhelifa et al., How Data Workers..., 2017]

Future of Data Science Research

- Notebooks: Collaboration, better support for use in production, history navigation, etc.
- Understanding data science workflows across several fields like machine learning, significance testing, etc.
- Data science in AR, VR, and tabletops



Script



Computational Notebook



Interactive Console

- Data is everywhere, and data science is a valuable skill to have in the current day and age
- Improving data scientists' workflows and tools can be vastly beneficial!