Lowering barriers to using computational notebooks

Tanaka Chitete

Bachelor of Science (Honours), Computer Science

Dr. Rahul Gopinath

Lecturer, School of Computer Science

THE UNIVERSITY OF SYDNEY

Problems

Setting up

- Problems are faced before computation has even begun
- Loading and cleaning data is menial

Exploring and analysing data

- Exploration feels incomplete on many platforms
- Users find themselves stuck copying and pasting code

Managing code

- Dependency management is difficult for users without software engineering expertise
- Such users are plunged into “dependency hell”

Ensuring reliability

- Execution is unreliable and scaling to big data is unsupported
- Kernel crashes often result, causing data inconsistencies
- The lack of processing power is the main culprit

Archiving

- Some notebooks are made with long-term use in mind
- Most platforms don’t offer sufficient versioning or searching

Securing

- Mitigating data leaks and controlling access are key concerns
- A lack of security measures forces custom implementations
- Such implementations are error-prone

Sharing and collaborating

- Sharing happens *interactively*, *statically*, and as *read-only* files
- Collaboration via real-time editing is desired
- Existing sharing functionalities are insufficient
- Synced notebooks present coordination challenges

Reproducing and reusing

- Marking work and furthering research hinge on reproducibility
- Users want to adapt and reuse code to save time
- Neither reproducing computation or code is feasible
- Notebook and machine dependencies are contributing factors

Deploying to production

- Notebooks are better suited to “quick-and-dirty” work
- Cell-based programming doesn’t coincide with best practices
- Deployment requires significant work

Background

Computational notebook

- Document that encapsulates code, text, and visualisations
- Organised as a sequence of cells

Literate programming

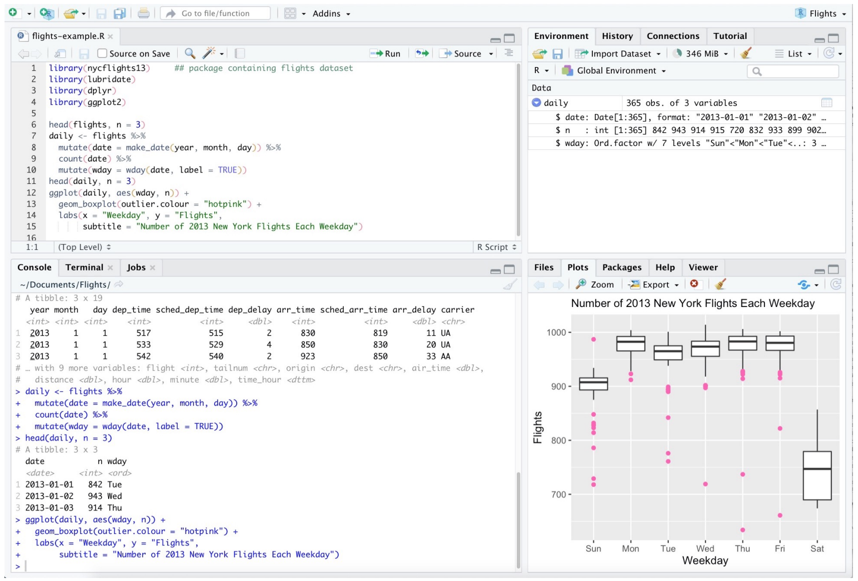
- Paradigm at the foundation of the computational notebook
- Improves readability by considering programs as “works of literature”

Computational platform

- Underlying system that runs the notebook
- Provides the necessary compute to execute code

Chronology

RStudio



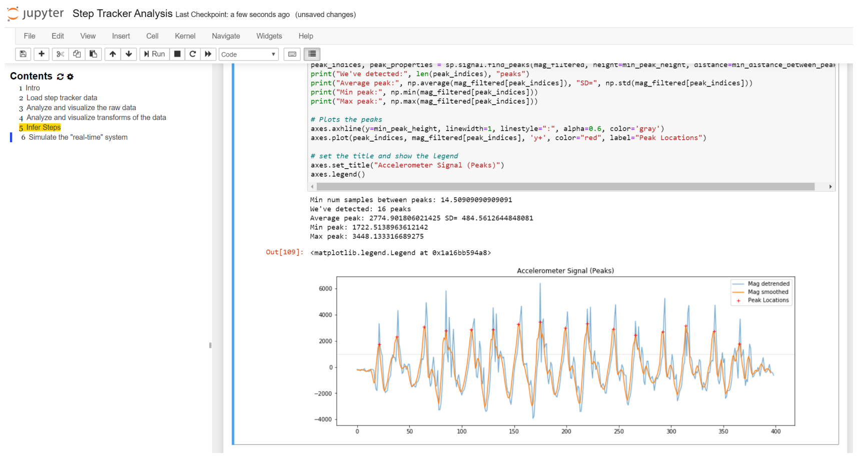
```r
library(nycflights13)     ## package containing flights dataset
library(lubridate)
library(dplyr)
library(ggplot2)

head(flights, n = 3)
daily <- flights %>%
  mutate(date = make_date(year, month, day)) %>%
  count(date) %>%
  mutate(wday = wday(date, label = TRUE))
head(daily, n = 3)
ggplot(daily, aes(wday, n)) +
  geom_boxplot(outlier.colour = "hotpink") +
  labs(x = "Weekday", y = "Flights",
       subtitle = "Number of 2013 New York Flights Each Weekday")
```

```
Data
daily                365 obs. of 3 variables
   $ date: Date[1:365], format: "2013-01-01" "2013-01-02" …
   $ n   : int [1:365] 842 943 914 915 720 832 933 899 902…
   $ wday: Ord.factor w/ 7 levels "Sun"<"Mon"<"Tue"<..: 3 …
```

```
~/Documents/Flights/
# A tibble: 3 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>
1  2013     1     1      517            515         2      830            819        11 UA
2  2013     1     1      533            529         4      850            830        20 UA
3  2013     1     1      542            540         2      923            850        33 AA
# … with 9 more variables: flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
> daily <- flights %>%
+   mutate(date = make_date(year, month, day)) %>%
+   count(date) %>%
+   mutate(wday = wday(date, label = TRUE))
> head(daily, n = 3)
# A tibble: 3 x 3
  date           n wday
  <date>     <int> <ord>
1 2013-01-01   842 Tue
2 2013-01-02   943 Wed
3 2013-01-03   914 Thu
> ggplot(daily, aes(wday, n)) +
+   geom_boxplot(outlier.colour = "hotpink") +
+   labs(x = "Weekday", y = "Flights",
+        subtitle = "Number of 2013 New York Flights Each Weekday")
>
```

- Released in 2011 to offer a native platform for R
- Integrates editor, plotting, console, and variable views
- Features code-completion and syntax highlighting
- Integrates R library-set
- Provides Python interface

Jupyter Notebook



jupyter Step Tracker Analysis Last Checkpoint: a few seconds ago (unsaved changes)

Contents

1 Intro
2 Load step tracker data
3 Analyze and visualize the raw data
4 Analyze and visualize transforms of the data
5 Infer Steps
6 Simulate the "real-time" system

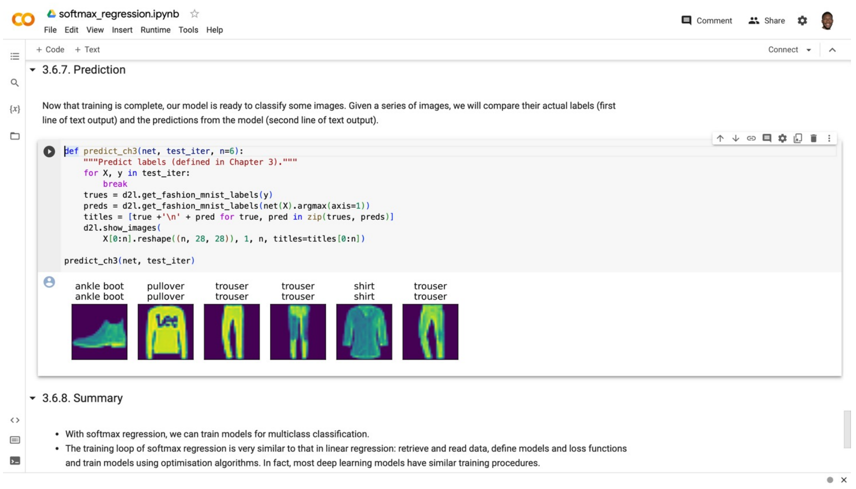
```
peak_indices, peak_properties = sp.signal.find_peaks(mag_filtered, height=min_peak_height, distance=min_distance_between_peak
print("We've detected:", len(peak_indices), "peaks")
print("Average peak:", np.average(mag_filtered[peak_indices]), "SD=", np.std(mag_filtered[peak_indices]))
print("Min peak:", np.min(mag_filtered[peak_indices]))
print("Max peak:", np.max(mag_filtered[peak_indices]))

# Plots the peaks
axes.axhline(y=min_peak_height, linewidth=1, linestyle=":", alpha=0.6, color='gray')
axes.plot(peak_indices, mag_filtered[peak_indices], 'y+', color="red", label="Peak Locations")

# set the title and show the legend
axes.set_title("Accelerometer Signal (Peaks)")
axes.legend()
```

```
Min num samples between peaks: 14.50909090909091
We've detected: 16 peaks
Average peak: 2774.901806021425 SD= 484.5612644848081
Min peak: 1722.5138963612142
Max peak: 3448.133316689275
```

```
Out[109]: <matplotlib.legend.Legend at 0x1a16bb594a8>
```

- Unveiled in 2015 to offer a more versatile platform
- Distinguished by its kernel-based architecture
- Supports over 40 programming languages
- Considering Python alone, supports all major packages

Colab



3.6.7. Prediction

Now that training is complete, our model is ready to classify some images. Given a series of images, we will compare their actual labels (first line of text output) and the predictions from the model (second line of text output).

```
def predict_ch3(net, test_iter, n=6):
    """Predict labels (defined in Chapter 3)."""
    for X, y in test_iter:
        break
    trues = d2l.get_fashion_mnist_labels(y)
    preds = d2l.get_fashion_mnist_labels(net(X).argmax(axis=1))
    titles = [true +'\n' + pred for true, pred in zip(trues, preds)]
    d2l.show_images(
        X[0:n].reshape((n, 28, 28)), 1, n, titles=titles[0:n])

predict_ch3(net, test_iter)
```

```
ankle boot   pullover   trouser   trouser   shirt   trouser
ankle boot   pullover   trouser   trouser   shirt   trouser
```

3.6.8. Summary

- With softmax regression, we can train models for multiclass classification.
- The training loop of softmax regression is very similar to that in linear regression: retrieve and read data, define models and loss functions and train models using optimisation algorithms. In fact, most deep learning models have similar training procedures.

- Launched in 2017 to enhance ML education
- Based on Jupyter platform
- Offers free GPU access
- Supports Python, integrating major ML packages
- Connects to Google Drive

Solution

Demonstrating Yuna

`https://tanaka-chitete.github.io/yuna/`

Implementing the interface

- CodeMirror supplies input cells
- Bootstrap supplies toolbar, buttons, and styling

Executing code

- Pyodide powers code execution
  - Ports Python 3.11 runtime using Wasm
  - Loads runtime into the browser using JavaScript
  - Maintains standard library as a zip archive

Loading packages

- Custom Python script embeds requirements
- micropip installs packages

Restoring sessions

- cloudpickle serialises and deserialises all user-defined objects
  - Variables
  - Functions
  - Imports

Parsing text

- Marked powers rich-text documentation

Evaluation

Managing code – Managing dependencies

- Installing, updating, and handling packages
- Platforms should offer integrated dependency management
- Yuna embeds all dependencies within the file, fully automating dependency management

Ensuring reliability – Executing

- Running code, either select cells or the entire notebook
- Platforms must ensure execution is constantly-available
- Yuna embeds both the kernel and parser within each environment, guaranteeing availability
  - An environment can be as small as 27 MB

Archiving – Versioning

- Tracking changes to source code files
- Notebooks rely on a wide range of artefacts
- Platforms must offer an integrated version control mechanism
- Yuna embeds the notebook, data, packages, and compute within each environment

Sharing and collaborating – Sharing the notebook

- Making the notebook accessible to others
- If sharing interactively is infeasible, platforms should offer cell-collapsing, exporting, and comment-only viewing
- Yuna embeds the notebook and platform within each environment

Sharing and collaborating – Sharing related artefacts

- Making the associated session, data and packages available
- Platforms should provide integrated artefact packaging
- Yuna embeds both data and packages within each environment

Reproducing and reusing – Reproducing

- Reproducibility is the ability to consistently replicate results
- Platforms should be designed to inherently support reproducibility
- Yuna embeds all artefacts within each environment file

Reproducing and reusing – Reusing and adapting

- Leveraging prior code for use in differing contexts
- Platforms should provide integrated dependency management
- By embedding dependencies within each environment, Yuna allows the user to simply copy the file

Discussion

Future work

- Loading data
- Usability testing
- Serialising
- Visualising
- Minimising file sizes
- Maintaining software
- Exporting contents

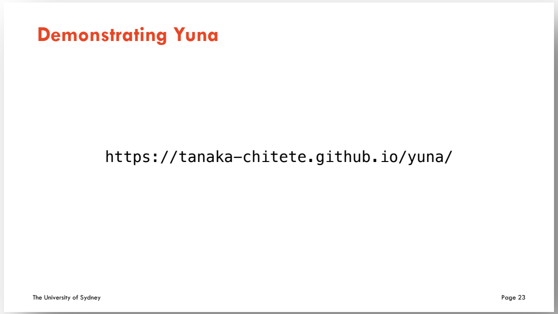
Conclusion

THE UNIVERSITY OF SYDNEY

Contributions

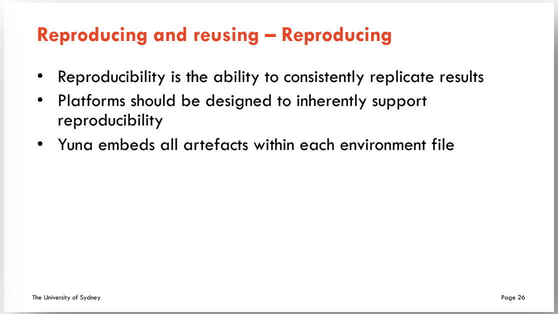
- Compendium of domain knowledge
- 19-point chronology of the most significant platforms
- Yuna, a fully self-contained platform, soothing:
  - Managing code
  - Ensuring reliability
  - Archiving
  - Sharing and collaborating
  - Reproducing and reusing
- Technical evaluation of 13 state-of-the-art platforms

Summary


Demonstrating Yuna

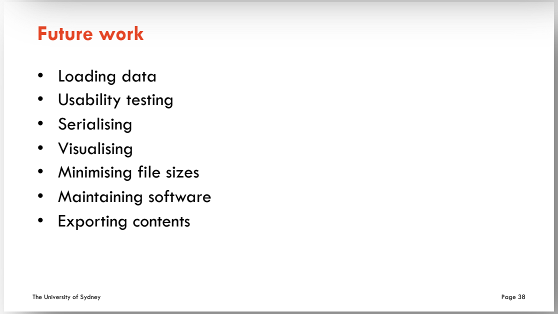
https://tanaka-chitete.github.io/yuna/

The University of Sydney Page 23


Reproducing and reusing – Reproducing

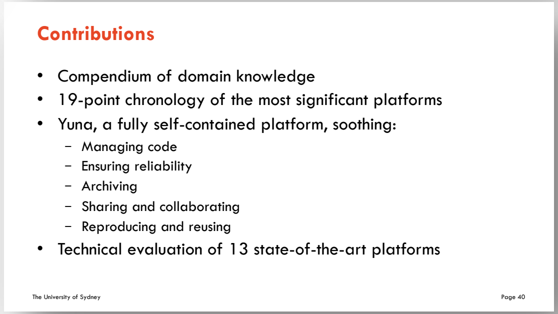
- Reproducibility is the ability to consistently replicate results
- Platforms should be designed to inherently support reproducibility
- Yuna embeds all artefacts within each environment file

The University of Sydney Page 26


Future work

- Loading data
- Usability testing
- Serialising
- Visualising
- Minimising file sizes
- Maintaining software
- Exporting contents

The University of Sydney Page 38


Contributions

- Compendium of domain knowledge
- 19-point chronology of the most significant platforms
- Yuna, a fully self-contained platform, soothing:
  - Managing code
  - Ensuring reliability
  - Archiving
  - Sharing and collaborating
  - Reproducing and reusing
- Technical evaluation of 13 state-of-the-art platforms

The University of Sydney Page 40

References

Aashita. 2023. Beginners Guide to Databricks: Batch Processing and Streaming. *Databricks*.

cdhowe. 2021. The RStudio IDE is a free and open source integrated development environment (IDE) for R. It helps programmers develop software written in R, Python, and several other languages. *Wikimedia Commons*.

Souti Chattopadhyay, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. 2020. What's wrong with computational notebooks. *In Proceedings: 2020 Conference on Human Factors in Computing Systems*.

Databricks. 2022. Databricks | Notebook Development Overview. *YouTube*.

Databricks. 2023a. Collaborate using Databricks notebooks. *Databricks*.

Databricks. 2023b. Databricks for Python developers. *Databricks*.

Databricks. 2023c. Databricks for R developers. *Databricks*.

Databricks. 2023d. Databricks for Scala developers. *Databricks*.

Databricks. 2023e. MLflow guide. *Databricks*.

Databricks. 2023f. SQL language reference. *Databricks*.

Databricks. 2023g. TensorBoard. *Databricks*.

Databricks. 2023h. What is Delta Lake? *Databricks*.

Databricks. n.d.a. Databricks is the data and AI company. *Databricks*.

Databricks. n.d.b. The best data platform is a lakehouse. *Databricks*.

Dave Gershgorn. 2017. Nerds rejoice: Google just released its internal tool to collaborate on AI. *Quartz*.

References

Google. n.d. Welcome to Colab! *Google*.

Donald Knuth. 1984. Literate Programming. *The Computer Journal*.

Makeability Lab. n.d. Jupyter Notebook showing visualization of a 3-axis accelerometer to infer step counts. *Makeability Lab*.

Matplotlib. n.d. Image tutorial. *Matplotlib*.

Microsoft. 2023a. Train TensorFlow models at scale with Azure Machine Learning. *Microsoft*.

Microsoft. 2023b. Train PyTorch models at scale with Azure Machine Learning. Microsoft.

Microsoft. n.d. Azure Machine Learning. *Microsoft*.

Ron Miller. 2015. Microsoft Officially Launches Azure Machine Learning Platform. *TechCrunch*.

NumPy. n.d. NumPy: The Absolute Basics for Beginners. *NumPy*.

pandas. n.d. Intro to Data Structures. *pandas*.

Fernando Perez. 2014. Project Jupyter. *Speaker Deck*.

Project Jupyter. n.d. Jupyter: Free software, open standards, and web services for interactive computing across all programming languages. *Project Jupyter*.

RStudio. n.d.a. Python. *RStudio*.

RStudio. n.d.b. RStudio Open-Source Packages. *RStudio*.

RStudio Team. 2011. RStudio, a new open-source IDE for R. *Posit Software*.

Kevin Ushey. 2023. Code Completion in the RStudio IDE. *Posit Support*.