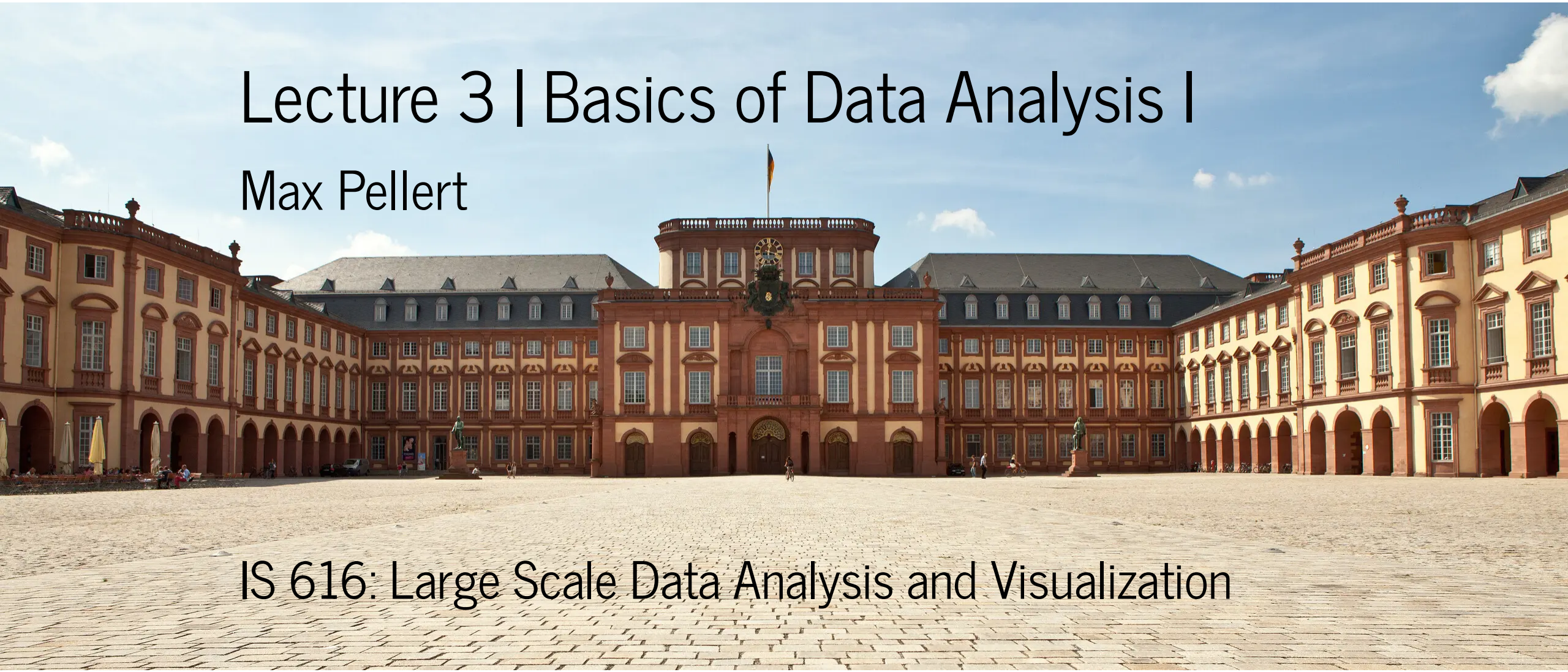
Lecture 3 | Basics of Data Analysis I

Max Pellert

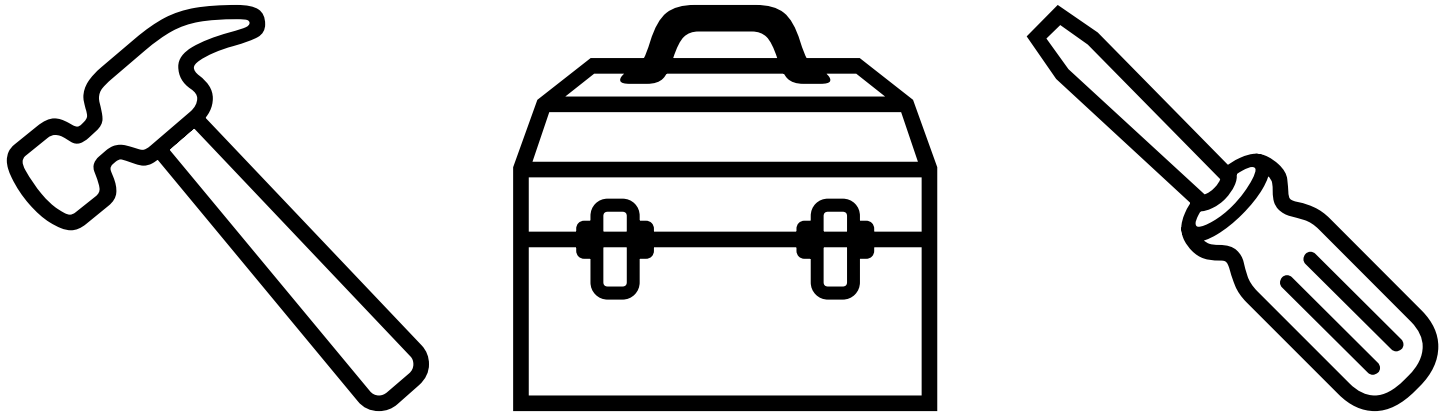


IS 616: Large Scale Data Analysis and Visualization

Aim

These course units are intended as a supplement to your actual work with data

It wants to teach you some tricks that are often not taught



Some Caveats

Don’t expect a full-fledged course that answers it all for you

That also doesn’t fit the subject matter

Data science is more like dentistry than particle physics

But, the aim is to bring everybody to the same level to be able to actually do visualizations (while at the same time also providing content that very likely also the more advanced student also haven’t heard yet)

It should convey some of the (softer) skills that you actually need often

“It is often said that 80% of data analysis is spent on the process of cleaning and preparing the data (Dasu and Johnson 2003).”

Wickham, 2014

Journal of Statistical Software

August 2014, Volume 59, Issue 10. *http://www.jstatsoft.org/*

Tidy Data

Hadley Wickham
RStudio

Abstract

A huge amount of effort is spent cleaning data to get it ready for analysis, but there has been little research on how to make data cleaning as easy and effective as possible. This paper tackles a small, but important, component of data cleaning: data tidying. Tidy datasets are easy to manipulate, model and visualize, and have a specific structure: each variable is a column, each observation is a row, and each type of observational unit is a table. This framework makes it easy to tidy messy datasets because only a small set of tools are needed to deal with a wide range of un-tidy datasets. This structure also makes it easier to develop tidy tools for data analysis, tools that both input and output tidy datasets. The advantages of a consistent data structure and matching tools are demonstrated with a case study free from mundane data manipulation chores.

Keywords: data cleaning, data tidying, relational databases, R.

Tidy Data

Wickham, H. (2014). Tidy Data. Journal of Statistical Software, 59(10). https://doi.org/10.18637/jss.v059.i10

Hadley Alexander Wickham (born 14 October 1979) is a New Zealand statistician known for his work on open-source software for the R statistical programming environment. He is the chief scientist at Posit, PBC and an adjunct professor of statistics at the University of Auckland, Stanford University, and Rice University. His work includes the data visualisation system ggplot2 and the tidyverse, a collection of R packages for data science based on the concept of tidy data.

The RStudio IDE is developed by Posit, PBC, a public-benefit corporation[18] founded by J. J. Allaire,[19] creator of the programming language ColdFusion. Posit has no formal connection to the R Foundation, a not-for-profit organization located in Vienna, Austria,[20] which is responsible for overseeing development of the R environment for statistical computing. Posit was formerly known as RStudio Inc. In July 2022, it announced that it changed its name to Posit, to signify its broadening exploration towards other programming languages such as Python.[21]

What makes a data set tidy?

“each variable is a column”

“each observation is a row”

“each type of observational unit is a table” (also called data frame or data table)

“data tidying: structuring datasets to facilitate analysis”

It provides a “philosophy of data”

What makes a data set untidy?

Generally, data sets can be constructed in all bizarre ways imaginable

- Column headers are values, not variable names.
- Multiple variables are stored in one column.
- Variables are stored in both rows and columns.
- Multiple types of observational units are stored in the same table.
- A single observational unit is stored in multiple tables.

Wide vs. long formats

row	a	b	c
A	1	4	7
B	2	5	8
C	3	6	9

(a) Raw data

row	column	value
A	a	1
B	a	2
C	a	3
A	b	4
B	b	5
C	b	6
A	c	7
B	c	8
C	c	9

(b) Molten data

4.1. Manipulation

Data manipulation includes variable-by-variable transformation (e.g., `log` or `sqrt`), as well as aggregation, filtering and reordering. In my experience, these are the four fundamental verbs of data manipulation:

- Filter: subsetting or removing observations based on some condition.
- Transform: adding or modifying variables. These modifications can involve either a single variable (e.g., log-transformation), or multiple variables (e.g., computing density from weight and volume).
- Aggregate: collapsing multiple values into a single value (e.g., by summing or taking means).
- Sort: changing the order of observations.

Create and use tidy data also in the interest of reproducibility and open science (think of git too!)

Tidyverse

Packages Blog Learn Help Contribute

R packages for data science

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

dplyr

Overview

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.

Introduction to data.table

2023-02-16

This vignette introduces the `data.table` syntax, its general form, how to *subset* rows, *select and compute* on columns, and perform aggregations *by group*. Familiarity with `data.frame` data structure from base R is useful, but not essential to follow this vignette.

Data analysis using `data.table`

Data manipulation operations such as *subset*, *group*, *update*, *join* etc., are all inherently related. Keeping these *related operations together* allows for:

- *concise* and *consistent* syntax irrespective of the set of operations you would like to perform to achieve your end goal.

https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html

Why `data.table`?

- concise syntax: fast to type, fast to read
- fast speed
- memory efficient
- careful API lifecycle management
- community
- feature rich

Features

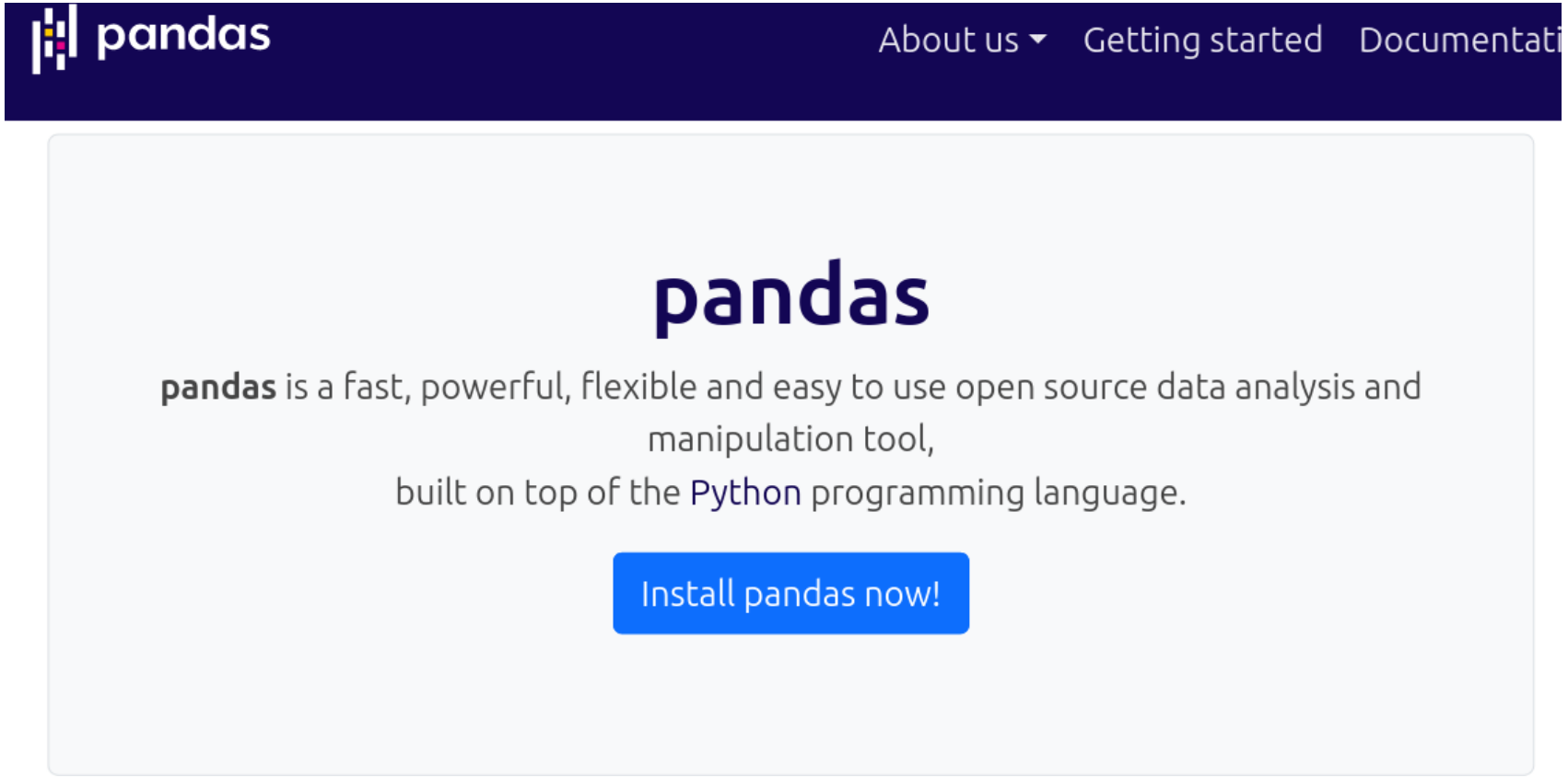
- fast and friendly delimited **file reader**: `?fread`, see also convenience features for *small* data
- fast and feature rich delimited **file writer**: `?fwrite`
- low-level **parallelism**: many common operations are internally parallelized to use multiple CPU threads
- fast and scalable aggregations; e.g. 100GB in RAM (see benchmarks on up to **two billion rows**)
- fast and feature rich joins: **ordered joins** (e.g. rolling forwards, backwards, nearest and limited staleness), **overlapping range joins** (similar to `IRanges::findOverlaps`), **non-equi joins** (i.e. joins using operators `>, >=, <, <=`), **aggregate on join** (`by=.EACHI`), **update on join**
- fast add/update/delete columns **by reference** by group using no copies at all
- fast and feature rich **reshaping** data: `?dcast` (*pivot/wider/spread*) and `?melt` (*unpivot/longer/gather*)
- **any R function from any R package** can be used in queries not just the subset of functions made available by a database backend, also columns of type `list` are supported
- has **no dependencies** at all other than base R itself, for simpler production/maintenance
- the R dependency is **as old as possible for as long as possible**, dated April 2014, and we continuously test against that version; e.g. v1.11.0 released on 5 May 2018 bumped the dependency up from 5 year old R 3.0.0 to 4 year old R 3.1.0

```
library(data.table)
DT = as.data.table(iris)

head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
# FROM[WHERE, SELECT, GROUP BY]
# DT  [i,     j,      by]

DT[Petal.Width > 1.0, mean(Petal.Length), by = Species]
```

```
##       Species       V1
## 1: versicolor 4.362791
## 2:  virginica 5.552000
```



https://pandas.pydata.org/

Main Features

Here are just a few of the things that pandas does well:

- Easy handling of **missing data** (represented as `NaN`, `NA`, or `NaT`) in floating point as well as non-floating point data
- Size mutability: columns can be **inserted and deleted** from DataFrame and higher dimensional objects
- Automatic and explicit **data alignment**: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let `Series`, `DataFrame`, etc. automatically align the data for you in computations
- Powerful, flexible **group by** functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data
- Make it **easy to convert** ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects
- Intelligent label-based **slicing**, **fancy indexing**, and **subsetting** of large data sets
- Intuitive **merging** and **joining** data sets
- Flexible **reshaping** and **pivoting** of data sets
- **Hierarchical** labeling of axes (possible to have multiple labels per tick)
- Robust IO tools for loading data from **flat files** (CSV and delimited), **Excel files**, **databases**, and saving/loading data from the ultrafast **HDF5 format**
- **Time series**-specific functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging

This tutorial uses the Titanic data set, stored as CSV. The data consists of the following data columns:

- PassengerId: Id of every passenger.
- Survived: Indication whether passenger survived. `0` for yes and `1` for no.
- Pclass: One out of the 3 ticket classes: Class `1`, Class `2` and Class `3`.
- Name: Name of passenger.
- Sex: Gender of passenger.
- Age: Age of passenger in years.
- SibSp: Number of siblings or spouses aboard.
- Parch: Number of parents or children aboard.
- Ticket: Ticket number of passenger.
- Fare: Indicating the fare.
- Cabin: Cabin number of passenger.
- Embarked: Port of embarkation.

https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/03_subset_data.html#min-tut-03-subset

```
import pandas as pd

titanic = pd.read_csv("data/titanic.csv")

titanic.head()
```

```
##    PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
## 0            1         0       3  ...   7.2500   NaN         S
## 1            2         1       1  ...  71.2833   C85         C
## 2            3         1       3  ...   7.9250   NaN         S
## 3            4         1       1  ...  53.1000  C123         S
## 4            5         0       3  ...   8.0500   NaN         S
##
## [5 rows x 12 columns]
```

```
ages = titanic["Age"]
ages.head()
```

```
## 0    22.0
## 1    38.0
## 2    26.0
## 3    35.0
## 4    35.0
## Name: Age, dtype: float64
```

```
above_35 = titanic[titanic["Age"] > 35]
above_35.head()
```

```
##     PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
## 1             2         1       1  ...  71.2833   C85         C
## 6             7         0       1  ...  51.8625   E46         S
## 11           12         1       1  ...  26.5500  C103         S
## 13           14         0       3  ...  31.2750   NaN         S
## 15           16         1       2  ...  16.0000   NaN         S
##
## [5 rows x 12 columns]
```

```
titanic["Age"] > 35
```

```
## 0      False
## 1       True
## 2      False
## 3      False
## 4      False
##        ...
## 886    False
## 887    False
## 888    False
## 889    False
## 890    False
## Name: Age, Length: 891, dtype: bool
```

“Two sides to data analysis”

Specialized programming languages like R (or the right packages in Python) are often well suited for your tasks

As we already learned: the bottleneck is usually RAM (because whole objects are kept in memory)

Small command line tools, on the other hand, work differently, usually line by line

This is often due to those tools being ancient and from times of severe hardware limitations

–> very efficient ways to do specific, simple operations

GNU toolchain

Can come in extremely handy

Caveat: Best to use them exactly for the task that they were designed for, even small deviations for other tasks can cause a lot of headache

Because these programs are often missing very basic concepts that are very common today

Usually, those tools work on lines of “humanly readable files” that you could open with any text editor (for example lines of text)

A line has a start and an end (usually the newline character)

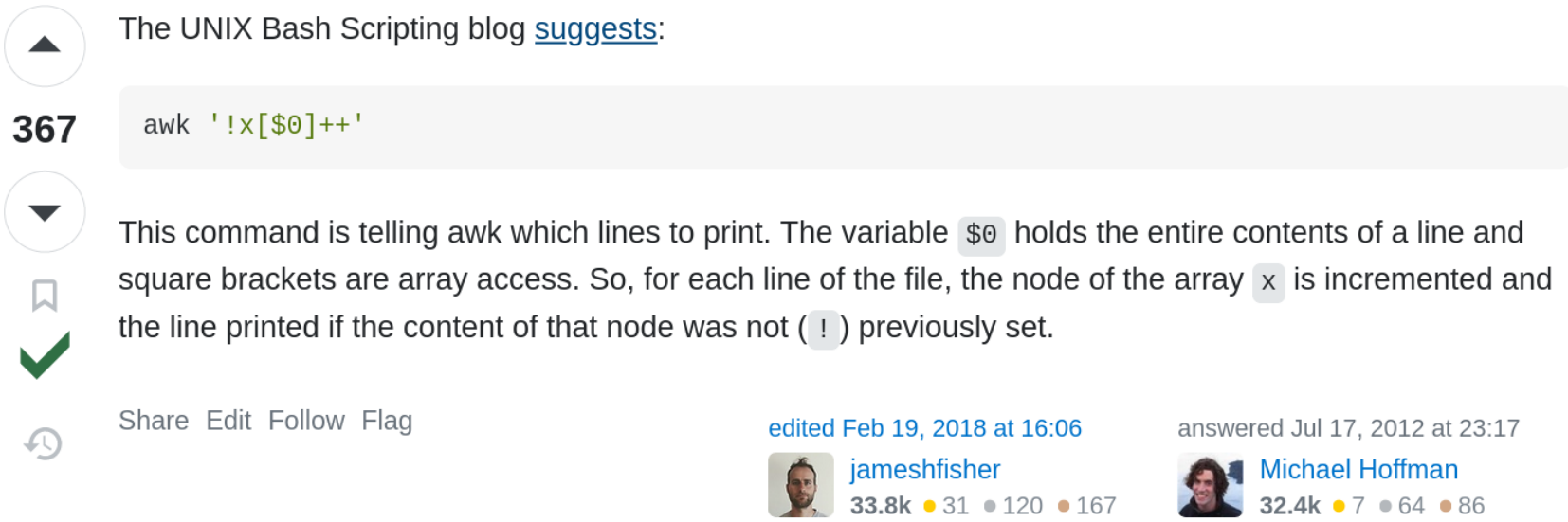
The small programs that we will discuss now have been pioneers by tackling specific tasks that come up often

That's why their functionalities have been modeled by practically all later developments (sometimes even with the same name)

It gives you an idea how to think "algorithmically" about a task, which often helps massively finding a solution

Also helps to ask the question right:

AWK



The UNIX Bash Scripting blog suggests:

367

```
awk '!x[$0]++'
```

This command is telling awk which lines to print. The variable `$0` holds the entire contents of a line and square brackets are array access. So, for each line of the file, the node of the array `x` is incremented and the line printed if the content of that node was not (`!`) previously set.

Share Edit Follow Flag

edited Feb 19, 2018 at 16:06
jameshfisher
33.8k ● 31 ● 120 ● 167

answered Jul 17, 2012 at 23:17
Michael Hoffman
32.4k ● 7 ● 64 ● 86

https://stackoverflow.com/questions/11532157/remove-duplicate-lines-without-sorting

grep

grep searches for *PATTERNS* in each *FILE*. *PATTERNS* is one or more patterns separated by newline characters, and **grep** prints each line that matches a pattern. Typically *PATTERNS* should be quoted when **grep** is used in a shell command.

```
grep 'Smith' data/titanic.csv
```

```
## 175,0,1,"Smith, Mr. James Clinch",male,56,0,0,17764,30.6958,A7,C
## 261,0,3,"Smith, Mr. Thomas",male,,0,0,384461,7.75,,Q
## 285,0,1,"Smith, Mr. Richard William",male,,0,0,113056,26,A19,S
## 347,1,2,"Smith, Miss. Marion Elsie",female,40,0,0,31418,13,,S
```

WC

```
wc - print newline, word, and byte counts for each file
```

"**w**ord **c**ount", but also counts lines with the right option:

```
wc -l data/titanic.csv
```

```
## 892 data/titanic.csv
```

Extremely handy for quick sanity checks, e.g. was all of the data transferred?

paste

paste - merge lines of files

```
cat data/file1.txt
```

```
## Suse
## Fedora
## CentOS
## OEL
## Ubuntu
```

```
cat data/file2.txt
```

```
## Linux
## Unix
## Solaris
## HPUX
## AIX
```

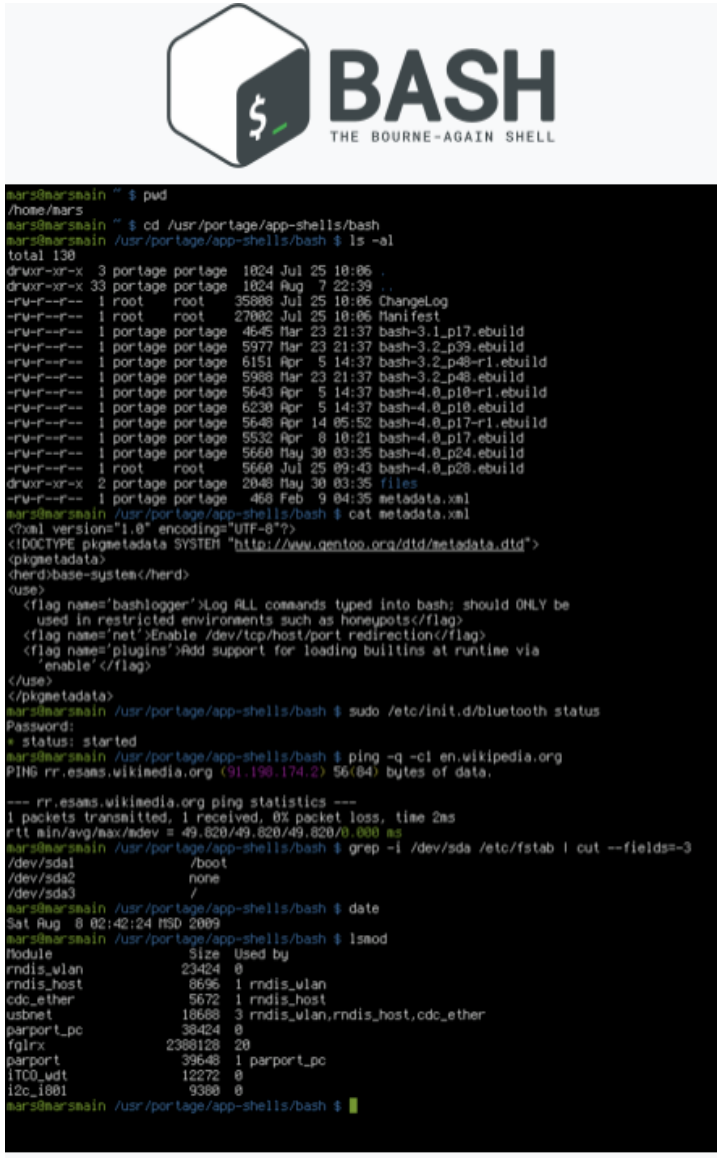
```
paste data/file1.txt data/file2.txt
```

```
## Suse Linux
## Fedora   Unix
## CentOS   Solaris
## OEL  HPUX
## Ubuntu   AIX
```

```
paste -d"," data/file1.txt data/file2.txt
```

```
## Suse,Linux
## Fedora,Unix
## CentOS,Solaris
## OEL,HPUX
## Ubuntu,AIX
```

Learn how to use the terminal!



Screenshot of a Bash session

Looping over files

Allows you to directly script in any directory of your file system

Is often much faster (and sometimes also safer) than to use a Python or R script for that

But still, many unintented things can happen, so be careful!

Basic wildcard matching is usually also possible and can come in very handy, for example to select all files with a specific naming scheme (e.g. date) or file ending

```
ls
```

```
## 03_basics_of_data_analysis_I_lecture.html
## 03_basics_of_data_analysis_I_lecture.Rmd
## awk_dedup_cropped.png
## bash_cropped.png
## data
## data_manipulation_cropped.png
## data.table_cropped.png
## dplyr_cropped.png
## features_data.table.png
## features_pandas.png
## grep_cropped.png
## job_control_cropped.png
## logo-stackoverflow.png
## missing_semester_cropped.png
## missing_semester_why_cropped.png
## molten_data_cropped.png
## pandas_cropped.png
```

```
for i in *.png; do echo $i; done
```

```
## awk_dedup_cropped.png
## bash_cropped.png
## data_manipulation_cropped.png
## data.table_cropped.png
## dplyr_cropped.png
## features_data.table.png
## features_pandas.png
## grep_cropped.png
## job_control_cropped.png
## logo-stackoverflow.png
## missing_semester_cropped.png
## missing_semester_why_cropped.png
## molten_data_cropped.png
## pandas_cropped.png
## pandas.png
## paste_cropped.png
## pipe_abstract.png
```

Chaining (or piping)

Allows you to chain simple tools together

Those tools often only have very limited applications (but usually work on them very efficiently)

Chaining them is extremely powerful as you can build up very complex pipelines from those simple tools

Pipe characters: | (or %>% or %|% or many others)

```
command1 | command2 | command3
```

```
ls -l | grep key | less
```

The command `ls -l` is executed as a process, the output (stdout) of which is piped to the input (stdin) of the process for `grep key`; and likewise for the process for `less`. Each process takes input from the previous process and produces output for the next process via *standard streams*. Each `|` tells the shell to connect the standard output of the command on the left to the standard input of the command on the right by an inter-process communication mechanism called an (anonymous) pipe, implemented in the operating system. Pipes are unidirectional; data flows through the pipeline from left to right.

```
ls | grep png | head -10
```

```
## awk_dedup_cropped.png
## bash_cropped.png
## data_manipulation_cropped.png
## data.table_cropped.png
## dplyr_cropped.png
## features_data.table.png
## features_pandas.png
## grep_cropped.png
## job_control_cropped.png
## logo-stackoverflow.png
```

```
ls | grep png | grep features
```

```
## features_data.table.png
## features_pandas.png
```

The Missing Semester of Your CS Education

Classes teach you all about advanced topics within CS, from operating systems to machine learning, but there's one critical subject that's rarely covered, and is instead left to students to figure out on their own: proficiency with their tools. We'll teach you how to master the command-line, use a powerful text editor, use fancy features of version control systems, and much more!

Students spend hundreds of hours using these tools over the course of their education (and thousands over their career), so it makes sense to make the experience as fluid and frictionless as possible. Mastering these tools not only enables you to spend less time on figuring out how to bend your tools to your will, but it also lets you solve problems that would previously seem impossibly complex.

Read about the motivation behind this class.

Why we are teaching this class

During a traditional Computer Science education, chances are you will take plenty of classes that teach you advanced topics within CS, everything from Operating Systems to Programming Languages to Machine Learning. But at many institutions there is one essential topic that is rarely covered and is instead left for students to pick up on their own: computing ecosystem literacy.

Over the years, we have helped teach several classes at MIT, and over and over we have seen that many students have limited knowledge of the tools available to them. Computers were built to automate manual tasks, yet students often perform repetitive tasks by hand or fail to take full advantage of powerful tools such as version control and text editors. In the best case, this results in inefficiencies and wasted time; in the worst case, it results in issues like data loss or inability to complete certain tasks.

These topics are not taught as part of the university curriculum: students are never shown how to use these tools, or at least not how to use them efficiently, and thus waste time and effort on tasks that *should* be simple. The standard CS curriculum is missing critical topics about the computing ecosystem that could make students' lives significantly easier.

Take a look at ./missing-semester

https://missing.csail.mit.edu/

You learn about small tools and tricks that can be enormous time savers

Especially important, learning about command line interfaces and job control: