Northeastern University

INFO 6105
Data Sci Eng Methods & Tools
Lecture 1

January 7 2019

Part 1

DATA SCIENCE IS SEXY

Secure | https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century

MENU Data Scientist: The Sexiest Job of the 21st Century

DATA

Data Scientist: The Sexiest Job of the 21st Century

by Thomas H. Davenport and D.J. Patil

FROM THE OCTOBER 2012 ISSUE

SUMMARY SAVE SHARE 12 COMMENT TEXT SIZE PRINT $8.95 BUY COPIES

When Jonathan Goldman arrived for work in June 2006 at LinkedIn, the business networking site, the place still felt like a start-up. The company had just under 8 million accounts, and the number was growing quickly as existing members invited their friends and colleagues to join. But users weren't seeking out connections with the people who were already on the site at the rate executives had expected. Something was apparently missing in the social experience. As one LinkedIn manager put it, "It was like arriving at a conference reception and realizing you don't know anyone. So you just stand in the corner sipping your drink—and you probably leave early."

Goldman, a PhD in physics from Stanford, was intrigued by the linking he did see going on and by the richness of the user profiles. It all made for messy data and unwieldy analysis, but as he began exploring people's connections, he started to see possibilities. He began forming theories, testing hunches, and finding patterns that allowed him to predict whose networks a given profile would

https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century

The sexiest job of the 21st Century

- *George Roumeliotis, the head of a data science team at Intuit in Silicon Valley, holds a doctorate in astrophysics*
- *Roumeliotis was clear with us that he (…) begins his search for data scientists by asking candidates if they can develop prototypes in a mainstream programming language. Roumeliotis seeks both a skill set—a solid foundation in, statistics, probability, linear algebra, and computer science—and certain habits of mind. He wants people with a feel for business issues and empathy for customers.*
- *Pay will of course be a factor. A good data scientist will have many doors open to him or her, and salaries will be bid upward.*
- *Survey of the priorities of data scientists revealed something more fundamentally important. They want to be "on the bridge." The reference is to the 1960s television show Star Trek, in which the starship captain James Kirk relies heavily on data supplied by Mr. Spock. Data scientists want to be in the thick of a developing situation*


Karl
Jeri

The sexiest job of the 21st Century

- *But the data scientists we've spoken with say they want to build things, not just give advice to a decision maker. One described being a consultant as "the dead zone — all you get to do is tell someone else what the analyses say they should do." By creating solutions that work, they can have more impact and leave their marks as pioneers of their profession.*
- *Hal Varian, the chief economist at Google, is known to have said, "The sexy job in the next 10 years will be statisticians. People think I'm joking, but who would've guessed that computer engineers would've been the sexy job of the 1990s?"*
- *The advance of big data shows no signs of slowing. The more data we have, the more we need data scientists to analyze it*

The tools..

- *Foundations in **probability & statistics**, **linear algebra**, and **computer science***
  - *Programming*

- *Not **apps** you can download*
  - *Using*

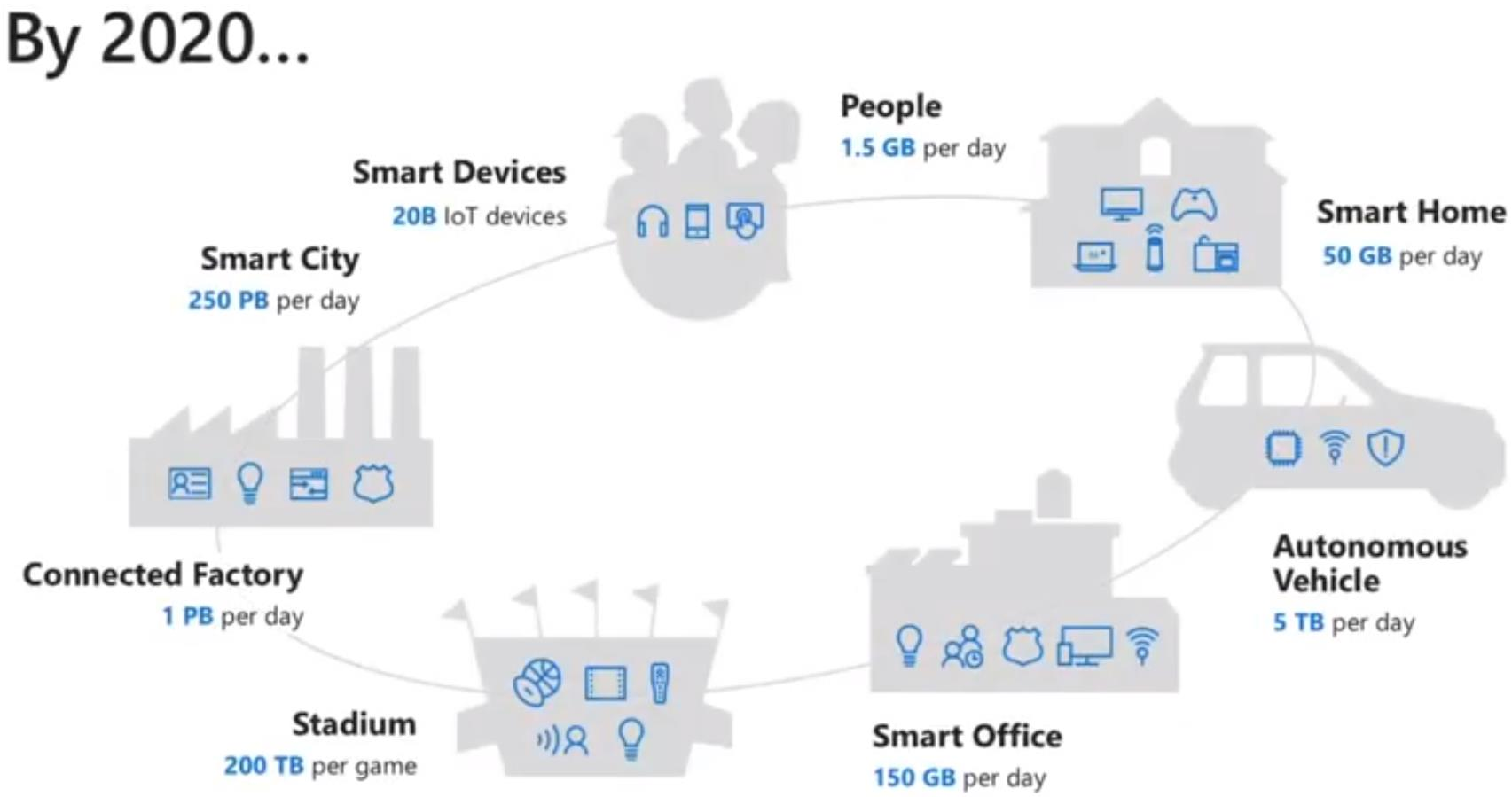
After you take this class..

- ..you will date the partner of your dreams ☺

But first..

- 1. Linear Algebra- Vector Spaces and Norms, Basis, Tensors, Vector and Tesnor Operations, Singular Value Decomposition (SVD), Eigendecomposition of a matrix, LU Decomposition, QR Decomposition/Factorization, Symmetric Matrices, Orthogonalization/Orthonormalization, Matrix Operations, Projections, Eigenvalues & Eigenvectors
- 2. Probability & Statistics - Probability Rules & Axioms, Bayes' Theorem, Random Variables, Variance and Expectation, Conditional and Joint Distributions, Standard Distributions (Bernoulli, Binomial, Multinomial, Uniform and Gaussian), Moment Generating Functions, Maximum Likelihood Estimation (MLE), Prior and Posterior, Maximum a Posteriori Estimation (MAP) and Sampling Methods
- 3. Multivariate Calculus- Differential and Integral Calculus, Partial Derivatives, Vector-Values Functions, Directional Gradient, Hessian, Jacobian, Laplacian and Lagrangian Distribution
- 4. Algorithms and Complex Optimizations- Complexity theory, data structures (Binary Trees, Hashing, Heap, Stack etc.), Dynamic Programming, Randomized & Sublinear Algorithm, Graphs, Gradient/Stochastic Descents

Why Data Science? Because..



Machine Learning is..

- ***Statistics***
  - ***Finding patterns in data***
  - Covariates become ***features***, non-linear regression becomes ***neural networks***, transformations of random variables becomes ***normalising flow***, placing a gaussian prior on parameters becomes ***L2 regularization***
- *For the opposing viewpoint, read here:*
  - *https://towardsdatascience.com/no-machine-learning-is-not-just-glorified-statistics-26d3952234e3*

Data Science with Machine Learning..

- Before *Tensorflow*, *Keras*, and *Torch*..
- You will become familiar with *NumPy*, *Pandas*, *SciPy*, and *SciKit-learn*
- Along with knowledge of *probability & statistics*, *and linear algebra*
  - Not simple, but *necessary*

Why Python

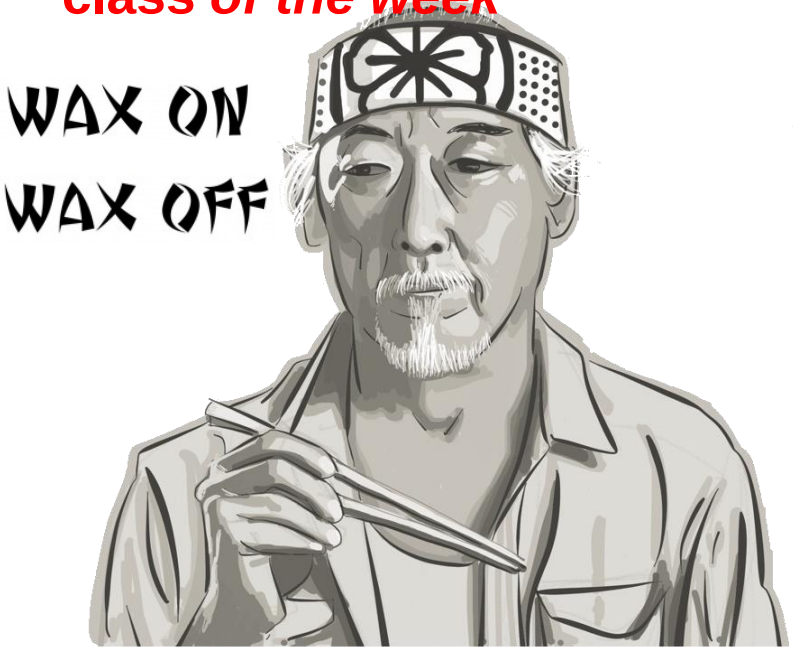
- Part of Python's success is ease of integration with C, C++, and Fortran code, the other part is that it *hides* its OO heritage
- As a result, Python has inherited Fortran legacy
  - Fortran used to solve ODEs and PDEs
  - Terse, functional and not very Object Oriented
  - All legacy Fortran code now in Python
- Python has become the de-facto language of scientists

Why R?

- **Easier than Python**
- **Lots of libraries written for R**
- **An overview of the class**
  - **You can start doing data science right now**
- **Start working with vectors and matrices**
- **R** first appeared in 1995 and served as an implementation of the **S** statistical programming language
- Roger Peng, an 18-year R programming veteran who teaches **R** both at the university and on Coursera notes: *"**R** is the most popular language used in the field of statistics“*
- *"I like **R** because it’s very easy to program”*
- **R**'s advantages include its package ecosystem. *"The vastness of package ecosystem is definitely one of R's strongest qualities -- if a statistical technique exists, odds are there's already an **R** package out there for it”*
- *"There’s a lot of functionality built in that's built for statisticians"*

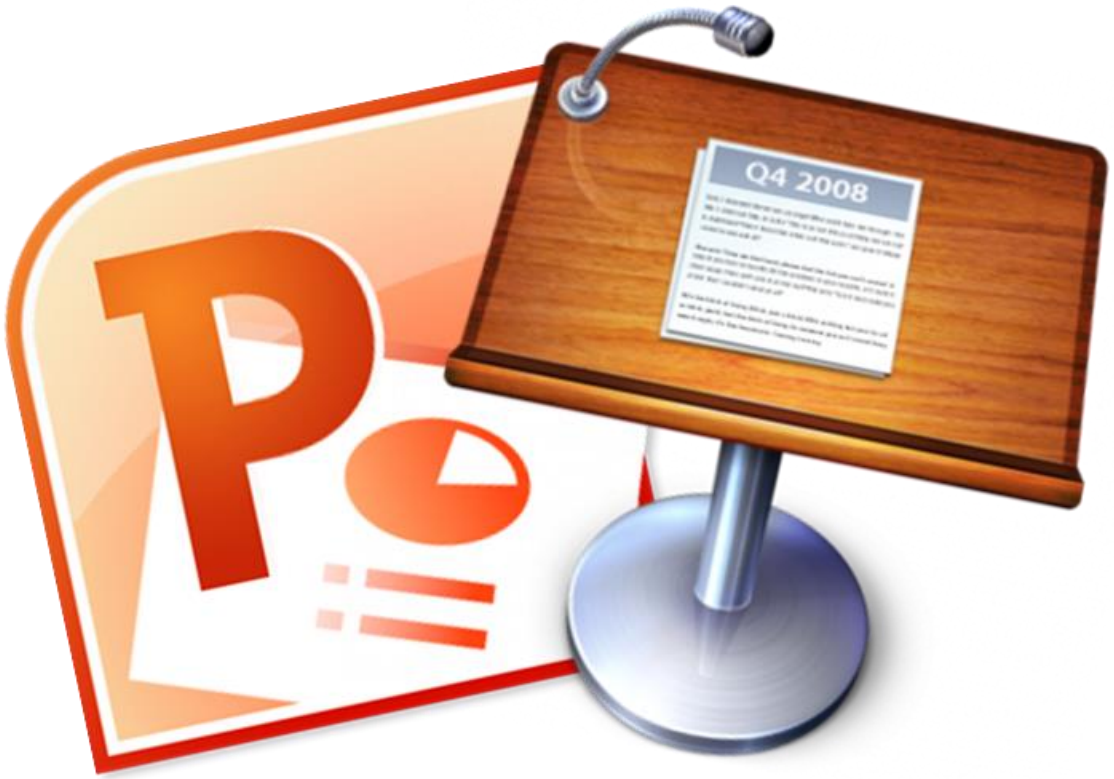
Class Policy

- **Grade: 30% homework, 30% mid-term, 30% final project, 10% final exam**
  - **Why do universities have grades? To separate the good from the bad?**
    - **No, to improve your *skills***
- **Do your homework, ask your TAs how to deliver, due before *first* class *of the week***



Questions, Slides

- **Ask questions: email TAs or dino.k@northeastern.edu**
- **Lecture slide-decks on Blackboard, about an hour before lecture**



Part 2

ABOUT ME

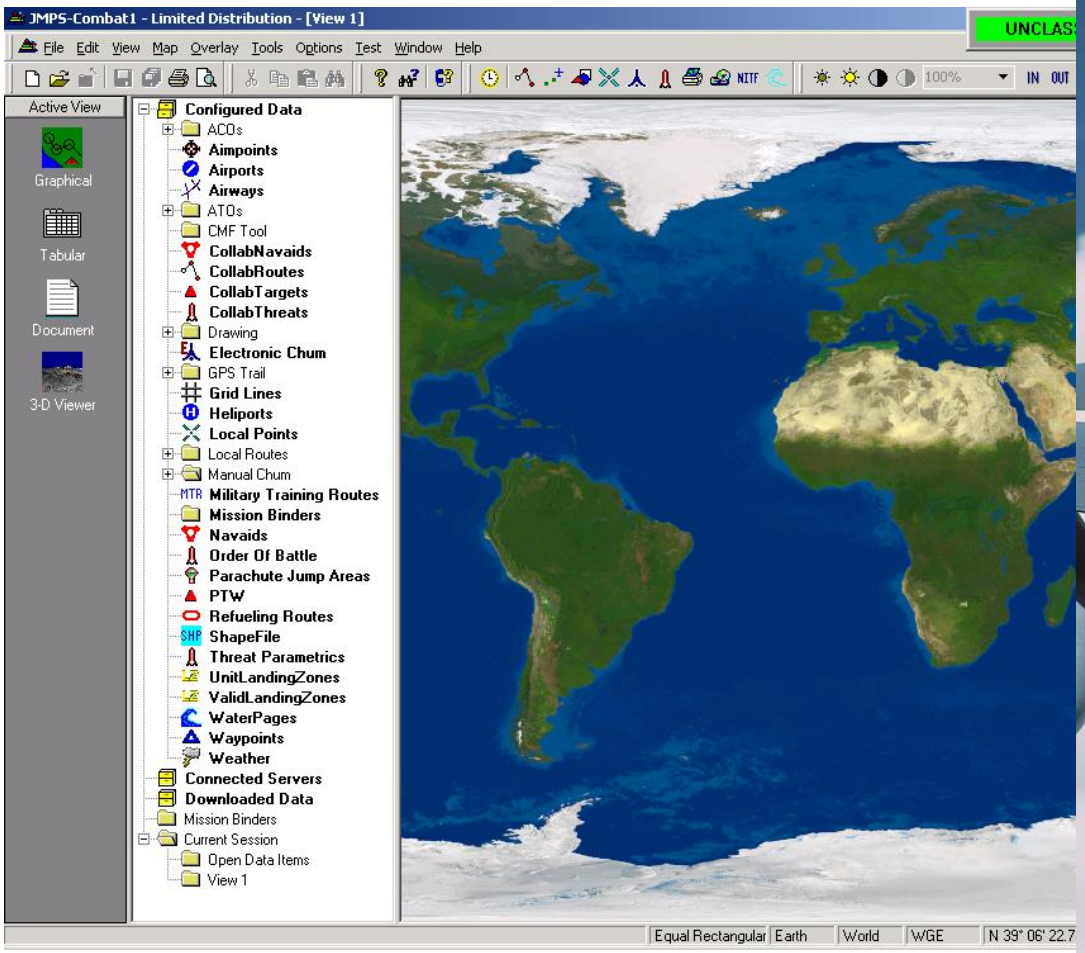
Me

Industry

- Xerox, Operating System research (headless)

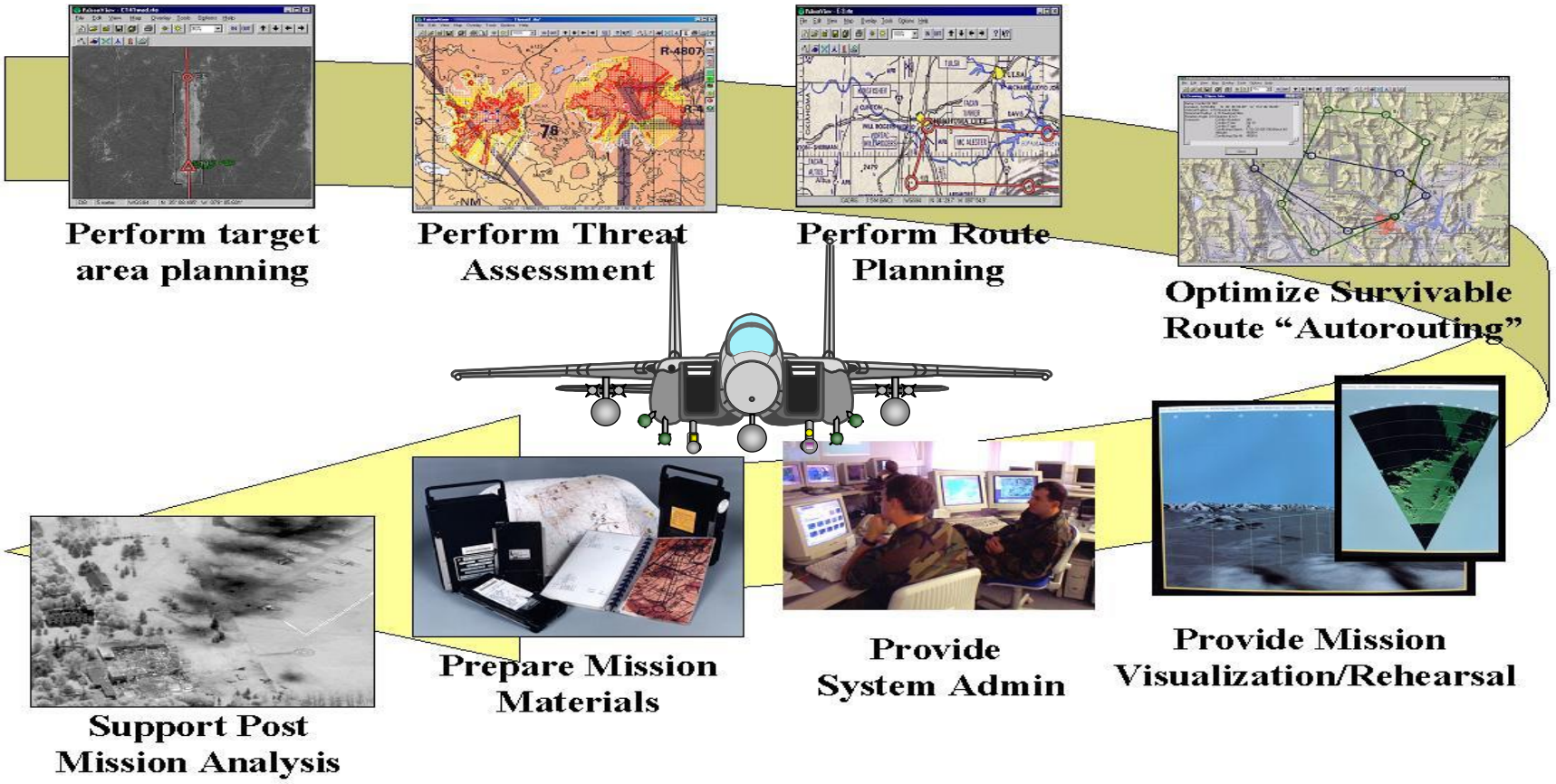
xerox

Government


JMPS-Combat1 - Limited Distribution - [View 1]
File Edit View Map Overlay Tools Options Test Window Help
Active View
Graphical
Tabular
Document
3-D Viewer
Configured Data
ACOs
Aimpoints
Airports
Airways
ATOs
CMF Tool
CollabNavaids
CollabRoutes
CollabTargets
CollabThreats
Drawing
Electronic Chum
GPS Trail
Grid Lines
Heliports
Local Points
Local Routes
Manual Chum
Military Training Routes
Mission Binders
Navaids
Order Of Battle
Parachute Jump Areas
PTW
Refueling Routes
ShapeFile
Threat Parametrics
UnitLandingZones
ValidLandingZones
WaterPages
Waypoints
Weather
Connected Servers
Downloaded Data
Mission Binders
Current Session
Open Data Items
View 1
Equal Rectangular
Earth
World
WGE


© Ted Carlson

Mission Planning

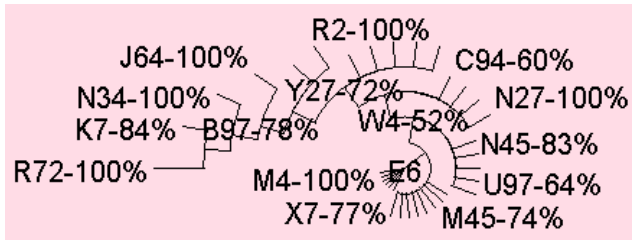


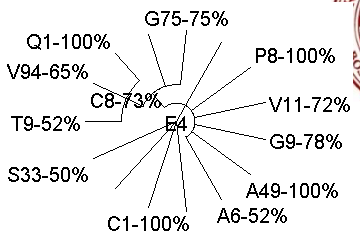
Army

- Use of microwave sensors for imaging continues to grow for applications such as target detection, airport screening, soil moisture estimation, etc.
- Hence, the need for indoor/small theater microwave imaging facilities
- A multistatic system is one where multiple transmit sites or multiple receive sites, or both, are used to construct the radar picture

IRS


INTERNAL
REVENUE
SERVICE

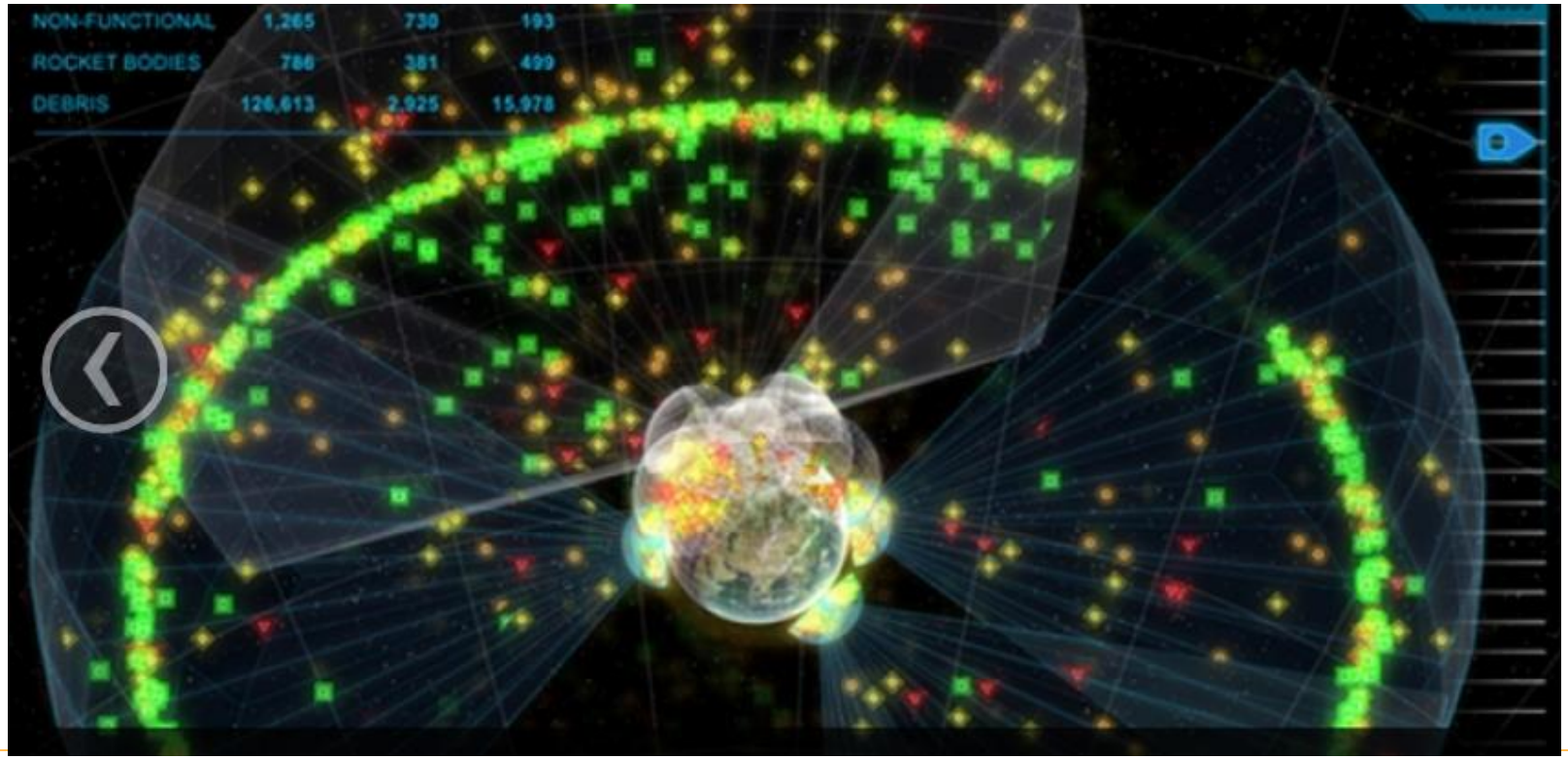

R2-100%
J64-100%
C94-60%
N34-100%
Y27-72%
N27-100%
K7-84%
B97-78%
W4-52%
N45-83%
R72-100%
M4-100%
F6
U97-64%
X7-77%
M45-74%


G75-75%
Q1-100%
P8-100%
V94-65%
C8-73%
V11-72%
T9-52%
F4
G9-78%
S33-50%
A49-100%
C1-100%
A6-52%

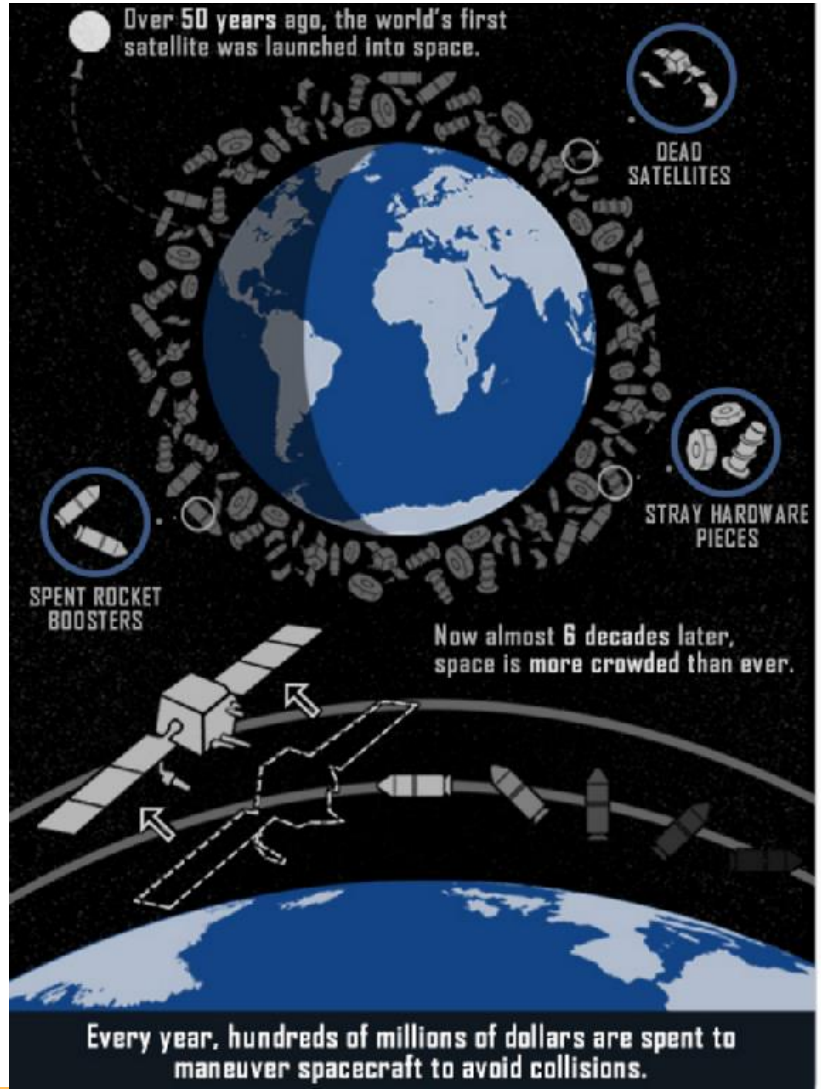
X55-16%.1
J38-17%.1
N96-36%.2
X97-72%
B5-100%
D4-48%.2
F9-18%.4
T1-100%.14
S4-1%.15
F9-5%.11
Z91-89%.15
C51-97%.15
L41-95%.3
N58-30%
F4
T8-8%.90
Y58-79%.6
J4-55%.31
T1-100%.27
S4-44%
Y47-33%.27
Y58-21%.39
N6-24%
H3-17%.16
Y58-79%.3
J4-55%.12
W6-15%.43
T8-8%.44
Z81-14%.28
T9-100%.27
V2-100%.27
C38-20%.7
Y58-21%.40
G55-44%.26
N58-45%
M4-16%
W6-15%.86
F6-32%.12
T1-100%.26
U4-96%
G55-44%.25
C23-86%.25
A53-5%.26
C8-100%
J38-17%.2
D8-49%.37
V9-98%.25
X55-16%.2
T9-100%.26
T1-100%.25
L2-39%.11
M54-93%
V5-100%.2
S55-76%.3
Y4-41%.2
D4-48%.3
C23-86%.28
S5-33%.25
H3-17%.25
Z81-14%.25
W6-15%.82
J4-55%.25
V43-59%.118
H3-40%.25
K8-100%.82
P61-88%.2
D8-49%.36
Y58-21%.34
V43-59%.133
T1-100%.24
R5-82%.2
N96-64%.2
T8-92%.133
W6-15%.80
C51-97%.24
S5-33%.23
A5-54%.2
R5-7%.3
H3-36%.81
K8-100%.76
K8-100%.73
L41-95%.5
W9-73%.6
A5-46%.6
L2-59%.2
Y2-100%-cycle.4
L2-2%.4
D8-49%.38
V43-59%.134
T8-92%.127
N58-25%.24
Y58-79%.5
F9-5%.14
F9-5%.13
H3-40%.24
K8-100%.72
J9-92%.6
C38-20%.6
Y2-100%.4
Y47-33%.19
Y47-33%
T8-92%.134
T8-92%.129
C23-86%.26
S5-33%.22
F6-32%.10
C38-80%.4
Y58-79%.4
Z91-
T8-92%.131
D8-51%.40
W6-15%.71
F9-18%.6
Z91-89%.19
F6-32%.9
G55-44%.18
T8-92%.126
G55-44%.23
S4-55%.10
V2-100%.19
V43-59%.130
T8-92%.121
Y47-33%.22
W6-15%.52
H3-17%.26
K8-100%.70
C51-97%.19
J4-18%.9

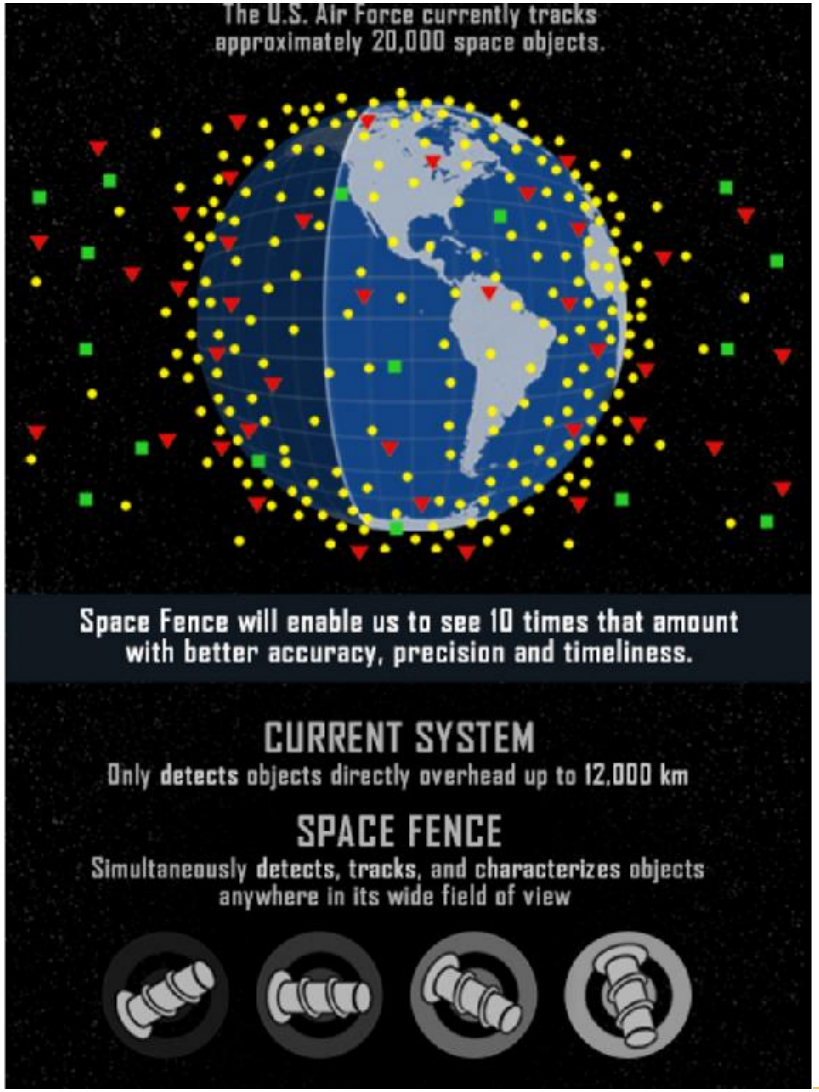
USAF Space Fence

- http://www.lockheedmartin.com/us/products/space-fence.html
- **Kwajalein Atoll in the Marshall Islands: Space Fence, the most powerful radar ever built, to identify and tracks objects in space. Initial operational capability 2018**

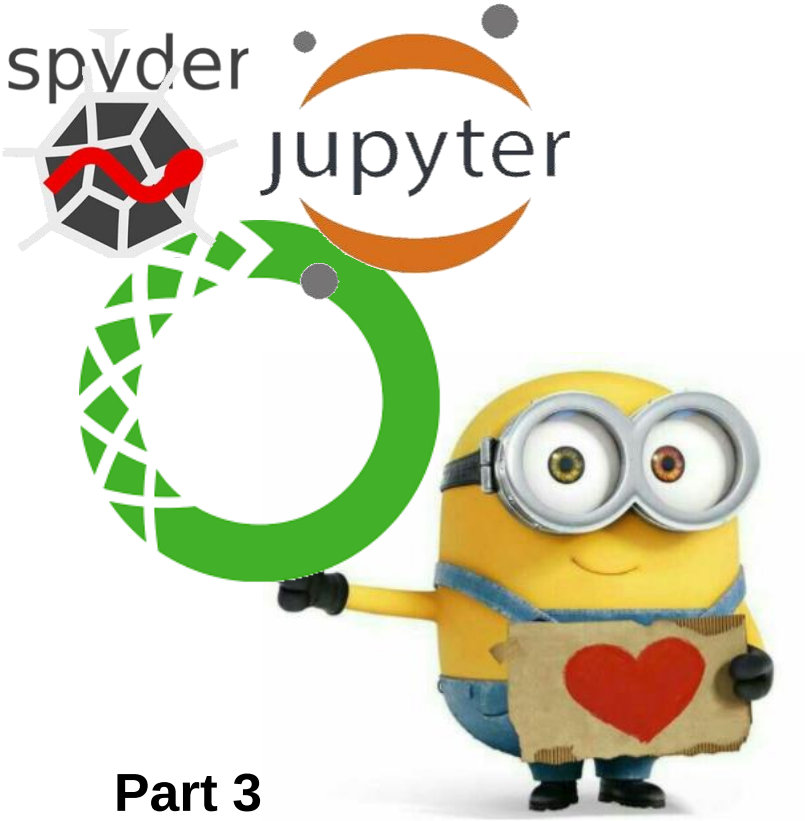


USAF Space Fence





Also



Part 3

INSTALLING ANACONDA PYTHON (NOT NOW, *YOUR HOMEWORK FOR WED*)

(On *Wed*, you'll get more homework for *next* week)

Anaconda

- **Enterprise-ready Python distribution with packages for Big Data processing, predictive analytics, and scientific computing**
- **https://www.continuum.io/anaconda**

ANACONDA
Powered by Continuum Analytics®

Install Anaconda

- Install anaconda, from:
  - https://www.continuum.io/downloads
  - https://repo.continuum.io/archive/index.html
- Option: You may install `miniconda` instead, if you don't have enough disk space
  - Pick Python 3.5 distro
- Test drive anaconda by running all steps in:
  - http://conda.pydata.org/docs/test-drive.html
  - Learn to play with different python environments and manage packages

To create new conda environments

- We are going to create <u>new</u> environments in this class:
  - `conda create --name theano python=3.4`
    - Because theano does not support python 3.5!
- Activate it:
  - Windows: `activate theano`
  - OSX/Linux: `source activate theano`
- List all & default (*) environment:
  - `conda info –envs`
- Python version?
  - `conda version`

Python IDEs

- **Anaconda *spyder***
- **Visual Studio with *Python Tools for Visual Studio* (PVTS)**
- **Visual Studio Code**
- **PyCharm**
- **PyDev (on top of Eclipse)**
- **Wing IDE**
- **Ninja IDE**
- **https://docs.continuum.io/anaconda/ide_integration**

Python notebook

- **Install notebook (for miniconda):**
  - `conda install notebook`
  - **That will install a number of dependencies (incl. ipython and jupyter)**
  - **It will allow us to do python in class like we did R in RStudio..**
- **Full conda should already have notebook installed**

Anaconda IDE

- `conda install spyder`
- **At a command prompt:** `spyder`
- **(like** `code`**)** ☺
- **Full anaconda should already have spyder installed, but probably old version**
  - **Update with:**
    - `conda update spyder`
  - **Windows: Administrator privileges console**
  - **OSX/Linux:** `sudo`

Anaconda add-ons

- **Commercial packages from *Continuum Analytics* and other vendors into Anaconda:**
  - **`IOPro`: fast, memory-efficient Python interface for databases, data files, Amazon S3 and MongoDB**
  - **`Accelerate`: includes `NumbaPro`, a compiler that targets multi-core CPUs and GPUs directly from simple Python syntax**
  - **`MKL Optimizations`: accelerates NumPy, SciPy, scikit-learn and NumExpr using Intel's Math Kernel Library**
- **All commercial packages from *Continuum Analytics* are available for a free 30 day trial**
- **The Anaconda Add-Ons are free for individual Academic use**
  - **Visit the Anaconda Academic page to request an Academic license**

On the Cloud: Wakari

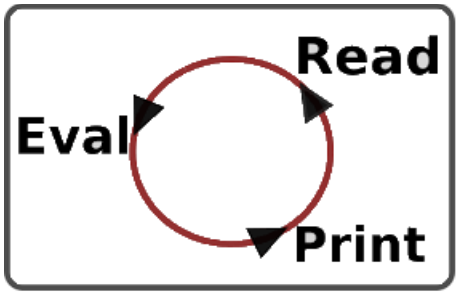
- **Web-based Python environment for collaborative data analysis, exploration and visualization**
  - **You can upload, create, and publish IPython Notebooks easily from your browser, and Wakari has Anaconda already installed**
  - **Create a free account for a full Python environment in the cloud**
- **http://wakari.io**

Google Colaboratory

- **Free Jupyter notebook environment that requires no setup and runs entirely in the cloud**
- **https://colab.research.google.com/notebooks/welcome.ipynb**

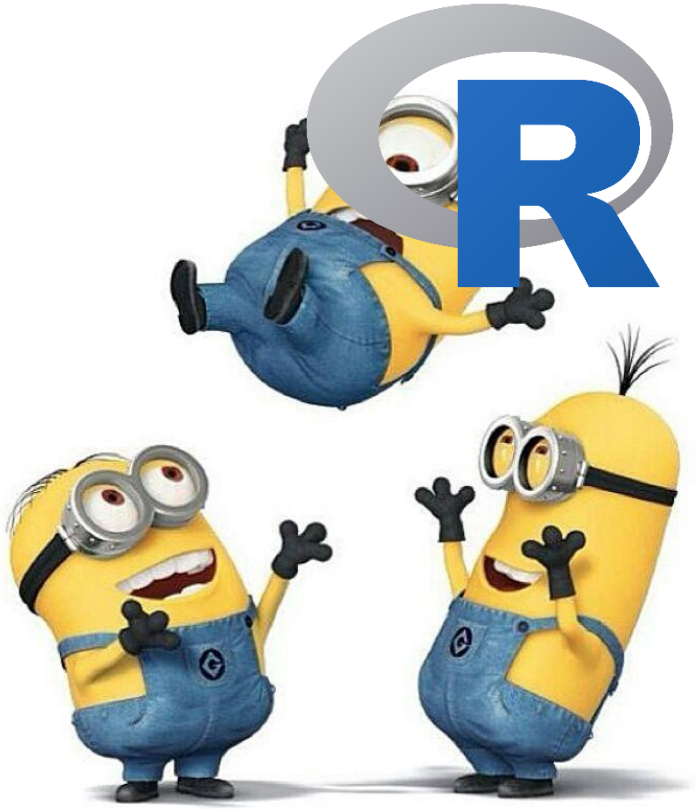
How to run ipython notebook in a browser

- In a command shell: `ipython notebook`
- That will pop up a browser
- You cut and paste code in the `In[]` cells on the iPython page, and then click on the 'Play' button to run each cell
- The great part about the seamless integration of text and code in IPython Notebook is that it's entirely conducive to the process:
  1. Form hypothesis
  2. Evaluate data
  3. Form conclusion
- It's a REPL loop, and we'll use that to do ML



To load a notebook..

- .. That you downloaded from the Web:
  - Open up a command shell
  - Go to the folder where you downloaded your notebook
  - `ipython notebook mynotebook.ipynb`
- You can only run one notebook server at a time..
  - u..Unless you change the port for ipython, because the default port for conda is 8888



Part 4

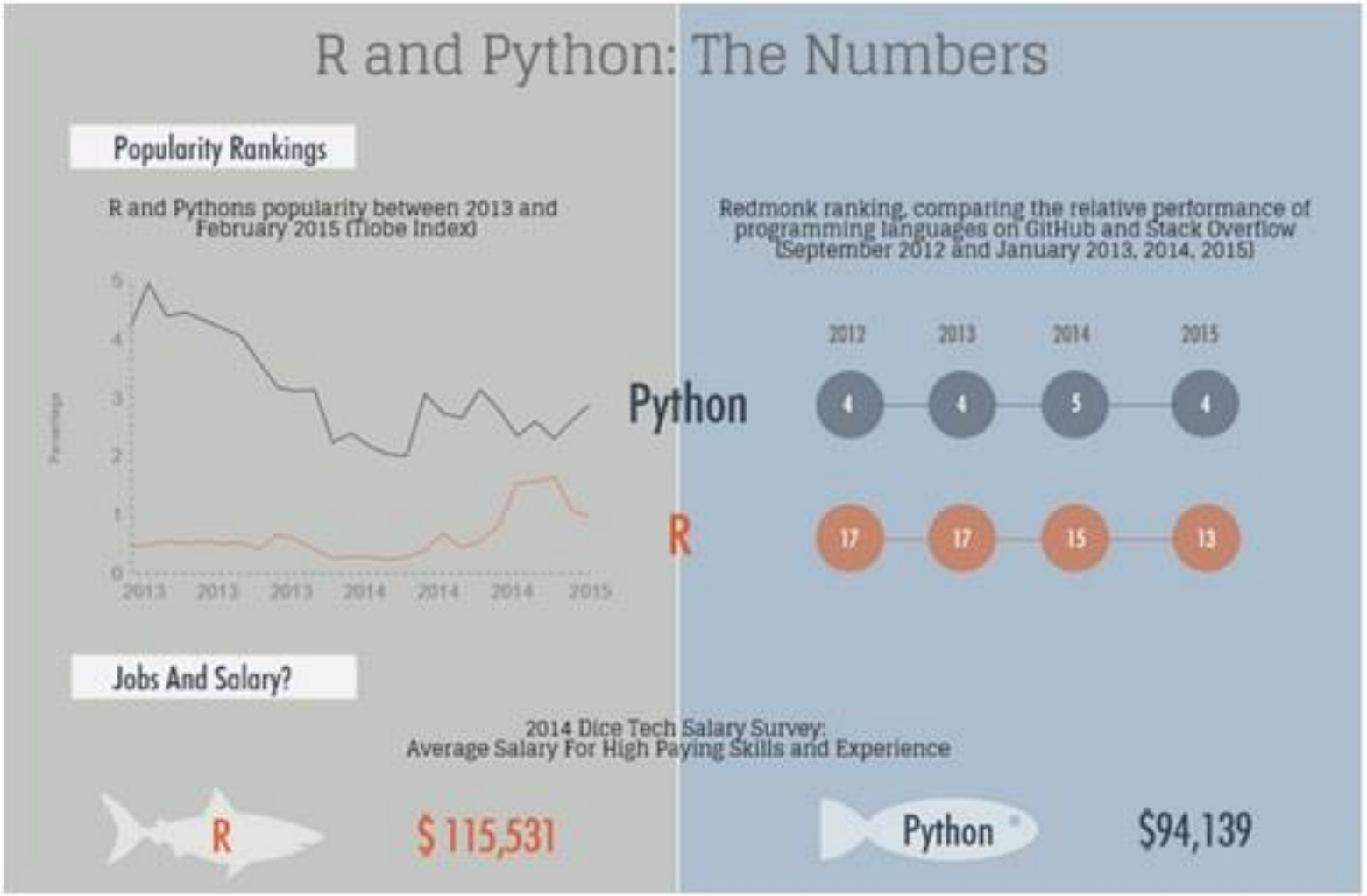
MACHINE LEARNING WITH R

Introduction to Statistical Modeling with R

R Introduction

- **Ross Ihaka and Robert Gentleman created the open-source language R in 1995 as an implementation of the *S programming language***
  - **Purpose was to develop a language that focused on delivering a better and more user-friendly way to do data analysis, statistics and graphical models**
  - **CRAN is a huge repository of curated R packages to which users can easily contribute**
    - **Comprehensive R archival network**
    - **https://cran.r-project.org/**

R vs Python



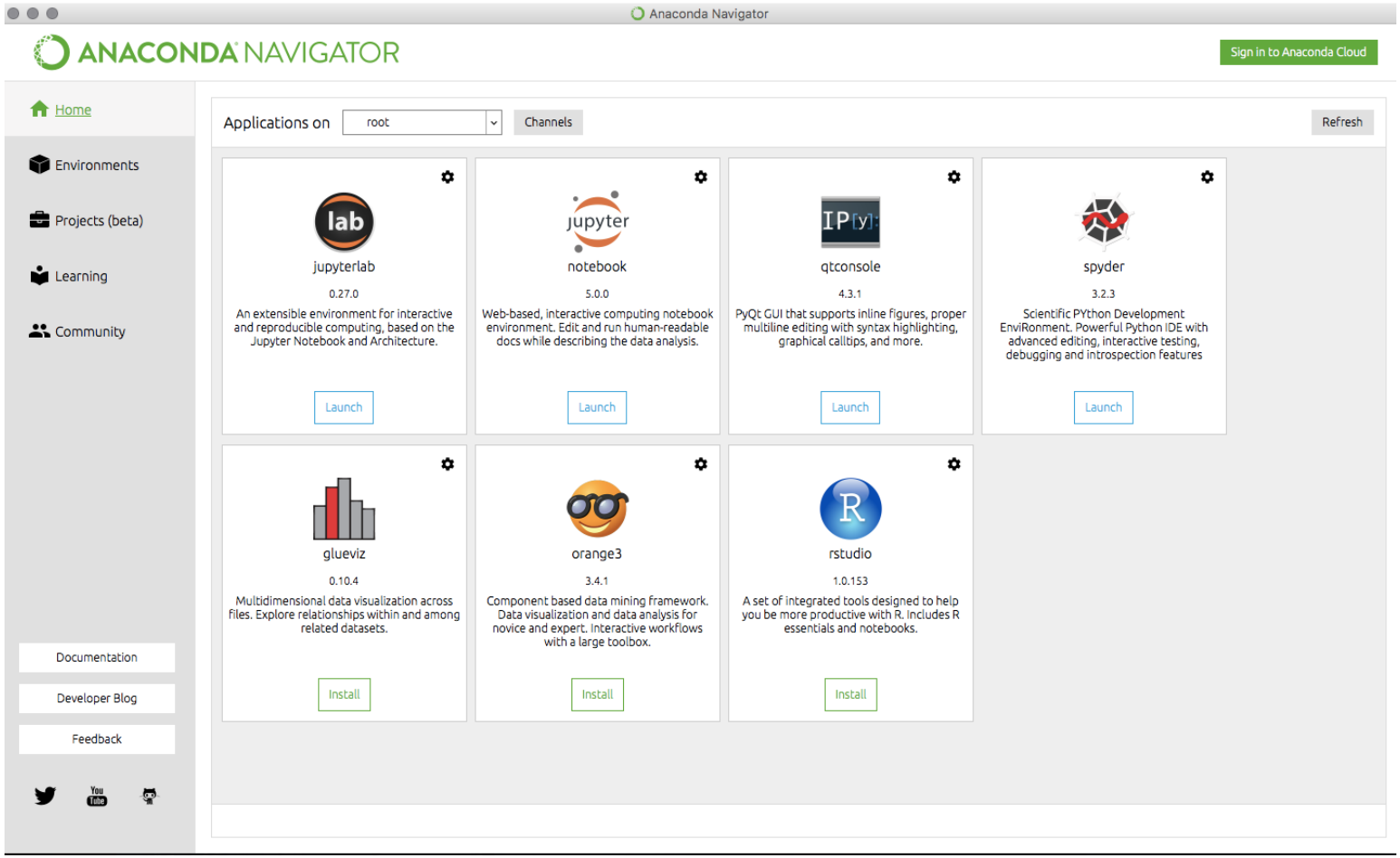
Source www.kdnuggets.com

What is R

- R is a free software environment for statistical computing, data mining, and graphics
  - Hopefully replace `Matlab`, the crack-cocaine of scientists-engineers-turned-programmers
- We use statistical analysis for:
  - inference - making conclusions based on data
  - prediction - what will happen when I observe new data?
  - and we create models to do both of those things

To install R and RStudio, Prerequisites

- **Install it with Anaconda!**



Other option: Standalone R & RStudio

- **Installing the R environment:**

- **Mac OS X**
  - **download and install from:**
  - **http://cran.r-project.org/bin/macosx/ [click R-3.2.0.pkg on left]**

- **Windows**
  - **download and install from:**
  - **http://cran.r-project.org/bin/windows/base/**

- **Linux, etc**
  - **See top of:**
  - **http://cran.us.r-project.org**

RStudio IDE

- **Install *RStudio* (IDE for using R)**
  - **Install for your appropriate system from the list at: http://www.rstudio.com/products/rstudio/download/**

R = or ← ?

In R, both statements **`x = 3`** and **`x <- 3`** have the effect of assigning the value 3 to the variable x. So if they have the same effect, does it matter which you use?
When R (and S before it) was first created, `<-` **was the only choice for the assignment operator.** Old AT&T keyboards had arrow as a key!
But R uses = for yet another purpose: associating function arguments with values (as in `pnorm(1, sd=2)`, **to set sd to 2)**
To make things easier for new users, R added the capability in 2001 to *also* allow = be used as an assignment operator, on the basis that the intent (assignment or association) is usually clear by context. So,

```
x = 3
```

clearly means "assign 3 to `x`", **whereas**

```
f(x = 3)
```

clearly means "call function `f`, setting the argument `x` to 3".
There is one case where ambiguity might occur: if you wanted to assign a variable *during* a function call. The only way to do this in modern versions of R is:

```
f(x <- 3)
```

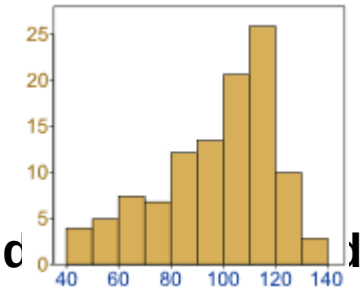
which means "assign 3 to `x`, **and call f with the first argument set to the value 3**

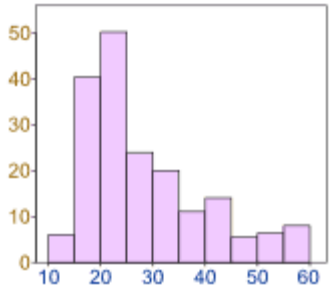
APL Character Set

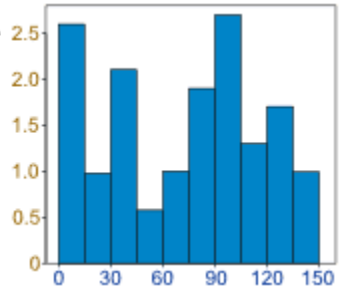
IBM 2741 keyboard layout

Data distributions

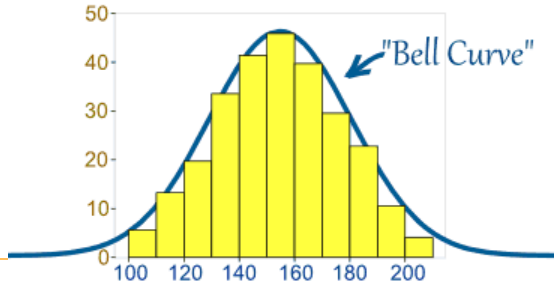
- Data can be "distributed" (spread out) in different ways
  - It can be spread out more on the left
  - Or more on the right
  - Or it can be all jumbled up
  - In many cases, the data tend[illegible] bias left or right
  - It is often called a "Bell Curve" because it looks like a bell







- *How data is distributed* is the foundation of statistics

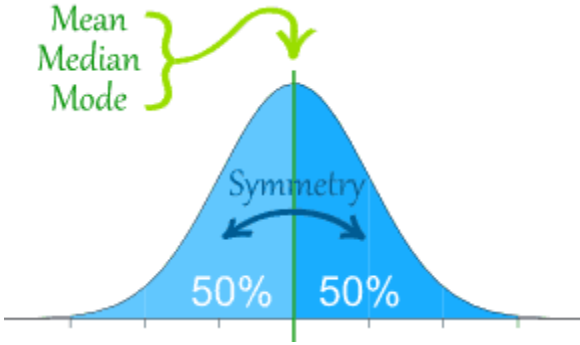


Normal distribution (`rnorm`)

- **We say the data is *normally distributed* when:**
  - **The probability density of the normal distribution is:**

$$f(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

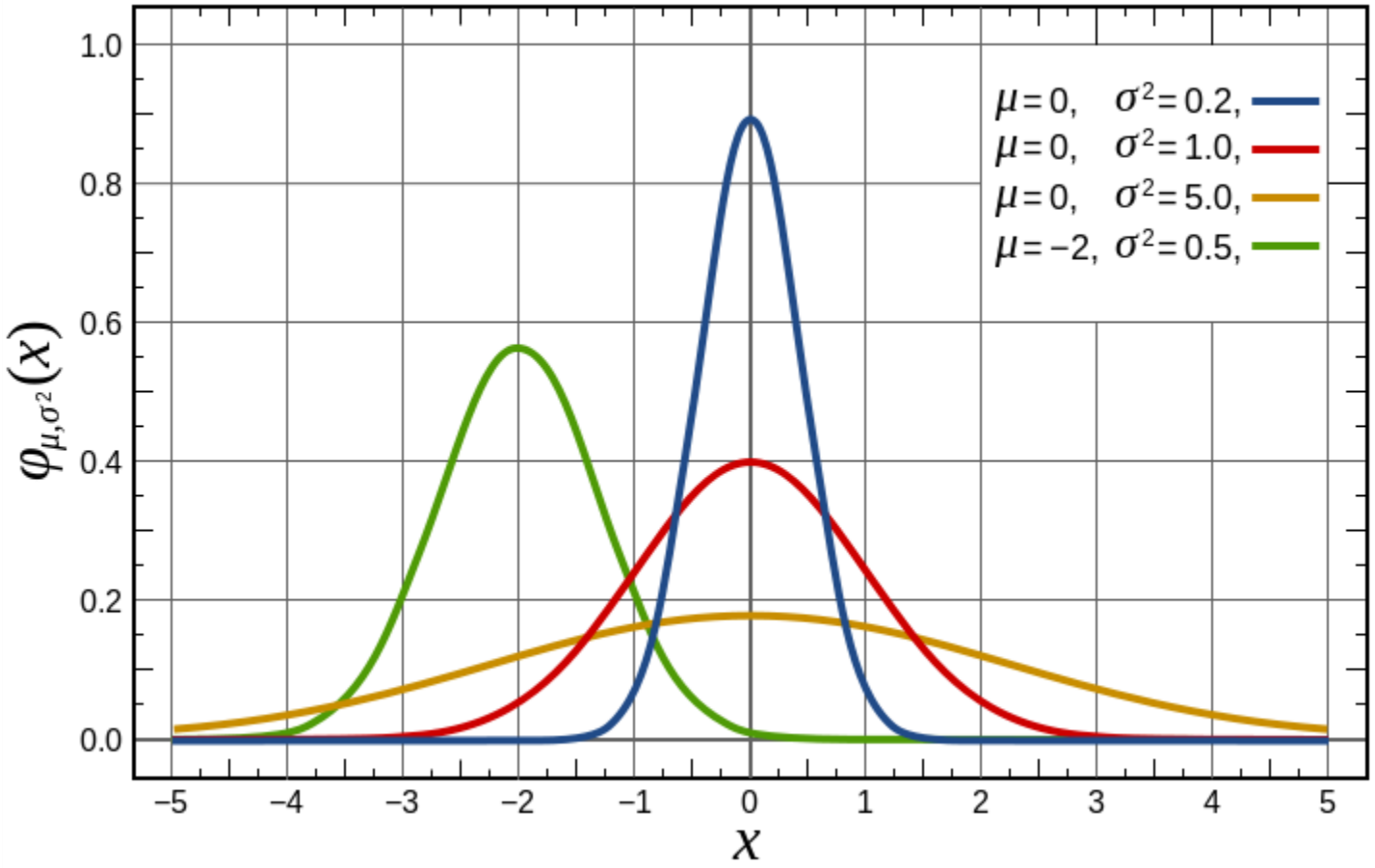
  - **Here, μ is the mean or expectation of the distribution (and also its median and mode). The parameter σ is its standard deviation; its variance is then σ^2**
  - **If $\mu = 0$ and $\sigma = 1$ the distribution is called the *standard normal distribution* or the *unit normal distribution***



symmetry about the center

50% of values less than the mean and 50% greater than the mean

Normal Distributions



Central Limit Theorem

- **In probability theory, the central limit theorem (CLT) states that, given certain conditions, the arithmetic mean of a sufficiently large number of iterates of independent random variables, each with a well-defined expected value and well-defined variance, will be approximately *normally distributed*, regardless of the underlying distribution**
- **https://www.mathsisfun.com/data/quincunx.html**

Correlation

- **In statistics, *dependence* is any statistical relationship between two random variables or two sets of data**
  - **Correlation refers to any of a broad class of statistical relationships involving dependence**
  - **Familiar examples of dependent phenomena include the correlation between the demand for a product and its price**
  - **Correlations are useful because they can indicate a *predictive* relationship that can be exploited**

Linear relationships

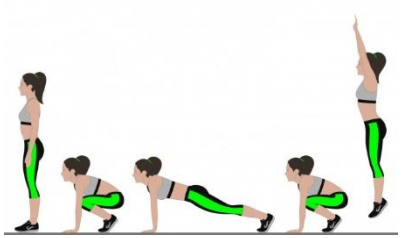
- **The most common of these is the *Pearson correlation coefficient*, which is sensitive *only* to a *linear relationship* between two variables**
- **The Pearson correlation coefficient indicates the strength of a *linear relationship* between two variables**
  - **If we have a series of *n* measurements of *X* and *Y* written as x_i and y_i where *i* = 1, 2, ..., *n***

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}},$$

where x and y are the sample means of *X* and *Y*, and s_x and s_y are the sample standard deviations of *X* and *Y*

Causation

- ***Correlation* does not imply *causation*!**
- **A correlation between `age` and `height` in children is fairly causally transparent (`age` → `height`), but a correlation between `mood` and `health` in people is less so**
  - **Does improved `mood` lead to improved `health` (`mood` → `health`), or does good `health` lead to good `mood` (`health` → `mood`), or both (`mood` ←→ `health`)? Or does some other factor underlie both (`factor X` ←→ `mood + health`)?**
  - **E.g.:**
    - **Exercise**



    - **Culture**
      - **Is local Culture in Crete, which includes a great diet and lots of music, responsible for elevated measures of both health and mood?**

Statistical Analysis

- **We use statistical analysis for:**
  - ***Inference* - making conclusions based on data**
  - ***Prediction* - what will happen when I observe new data?**
  - **And we create *models* to do both of those things**
- ***"All models are wrong - some are useful"* - George E. P. Box**

It’s been said that..

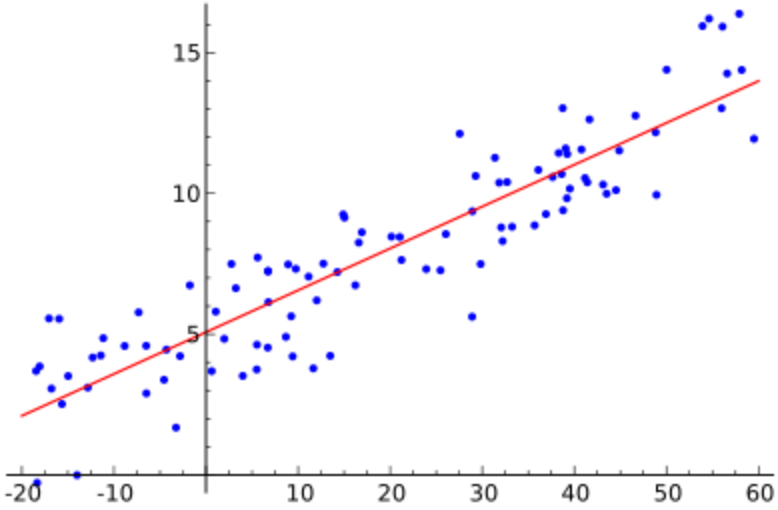
- **Our brain simply consists of a bunch of *predictors*, based on *models* we build for ourselves in our lifetimes..**

Variables

- **The decision as to which variable in a data set is modeled as the *dependent variable* and which are modeled as the *independent variables* may be based on a presumption that the value of one of the variables is caused by, or directly influenced by the other variables**
- **Independent variables are also called *regressors*, *exogenous variables*, *explanatory variables*, *covariates*, *input variables*, and *predictor variables***

Linear Regression

- **Regression is an approach for modeling the relationship between a scalar *dependent* variable y and one or more explanatory variable (*independent* variable) x**
  - **In linear regression, data are modeled using linear predictor functions**
  - **Linear regression models are often fitted using the least squares approach, but they may also be fitted in other ways..**



$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

Logistic Regression

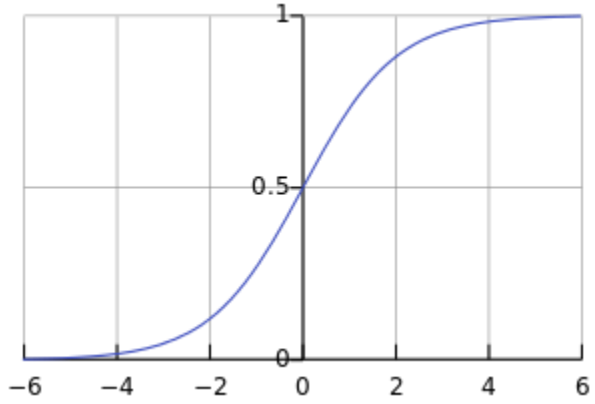
- **Logistic regression describes a kind of classification model in which predictor variables are combined with linear weights and then passed through a soft-limit function that limits the output to the range [0, 1]**
- **Logistic regression is closely related to other models such as:**
  - ***Perceptron*** **(where the soft limit is replaced by a hard limit)**
  - ***Neural networks*** **(where multiple layers of linear combination and soft limiting are used)**
  - ***Naive Bayes*** **(where linear weights are determined strictly by feature frequencies assuming independence)**
- **Logistic regression can't separate all possible classes, but in very high dimensional problems or where you can introduce new variables by combining other predictors, this is less of a problem**
- **Mathematical simplicity of logistic regression allows very efficient and effective learning algorithms to be derived**

Logistic Regression

- **The probabilities describing the possible outcomes of a single trial are modeled, as a function of the explanatory (predictor) variables, using a logistic function**
  - **A logistic function is a common "S" shape (sigmoid curve), with equation:**

$$f(x) = \frac{L}{1 + e^{-k(x - x_0)}}$$

  - **Population growth: The initial stage of growth is approximately exponential, then, as saturation begins, the growth slows, and at maturity, growth stops**



The logistic function is useful because it can take an input with any value from negative to positive infinity, whereas the output always takes values between zero and one and hence is interpretable as a probability

Key distinctions

- **Logistic regression is analogous to linear regression..**
- **However, the model of logistic regression is based on different assumptions about the relationship between dependent and independent variables**
  - **First, the conditional distribution p(y | x) is a *Bernoulli distribution* rather than a *Gaussian distribution*, because the dependent variable is *binary***
  - **Second, the estimated probabilities are restricted to [0,1] through the logistic distribution function**

Example

- **A good grade in this class (the *dependent variable*) is directly influenced by how many hours per week (the *predictor* or *independent variable*) you review the slides and do the homework**
  - **For example:**
    1or less hours per week → F
    2 – 3 hours per week → C
    4 – 5 hours per week → B
    8 – 9 hours per week → A

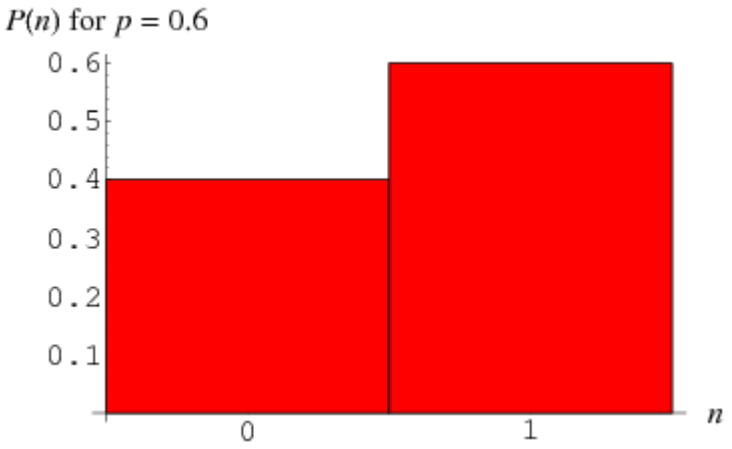
Bernoulli Distribution

- **Named after Swiss scientist *Jacob Bernoulli*, it is the probability distribution of a random variable which takes value 1 with success probability p and value 0 with failure probability q=1-p**
  - **It can be used, for example, to represent the toss of a coin, where "1" is defined to mean "heads" and "0" is defined to mean "tails" (or vice versa)**

$$P(n) = \begin{cases} 1-p & \text{for } n = 0 \\ p & \text{for } n = 1 \end{cases}$$

which can also be written

$$P(n) = p^n (1-p)^{1-n}$$

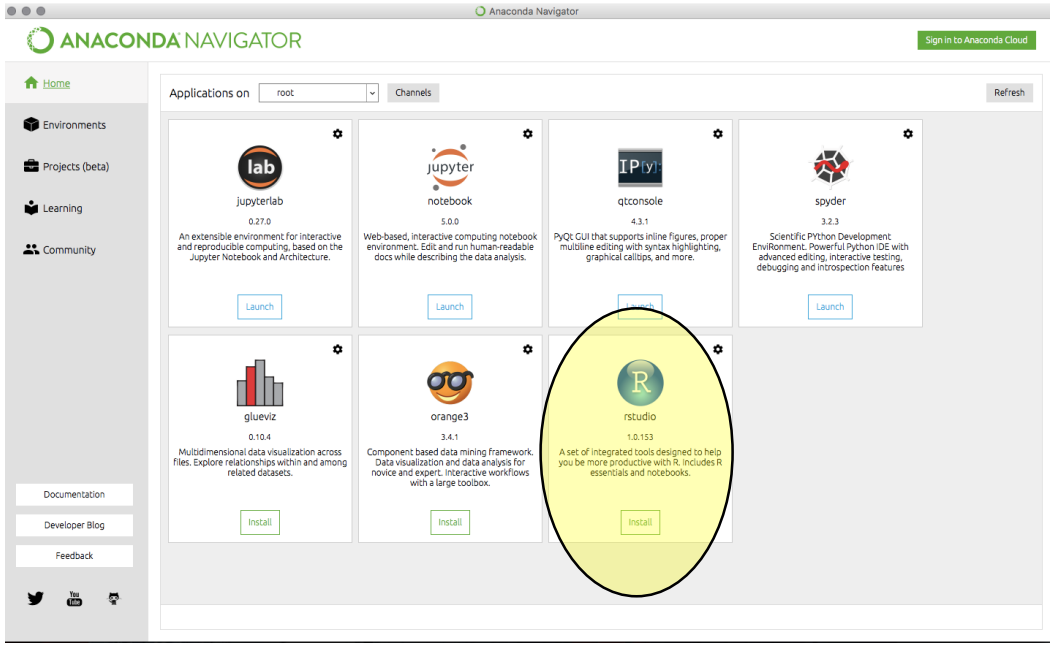


Use cases

- Logistic regression may be used to predict whether a patient has a given disease (e.g. diabetes; coronary heart disease), based on observed characteristics of the patient (age, sex, body mass index, results of various blood tests, etc.)
- Another example might be to predict whether an American voter will vote Democratic or Republican, based on age, income, sex, race, state of residence, votes in previous elections, etc
- The technique can also be used in engineering, especially for predicting the probability of failure of a given process, system or product
- Used in marketing applications such as prediction of a customer's propensity to purchase a product or halt a subscription, etc.
- In economics it can be used to predict the likelihood of a person's choosing to be in the labor force, and a business application would be to predict the likelihood of a homeowner defaulting on a mortgage

Learning R

- Run `Anaconda`
- Click on `RStudio`



Preliminaries

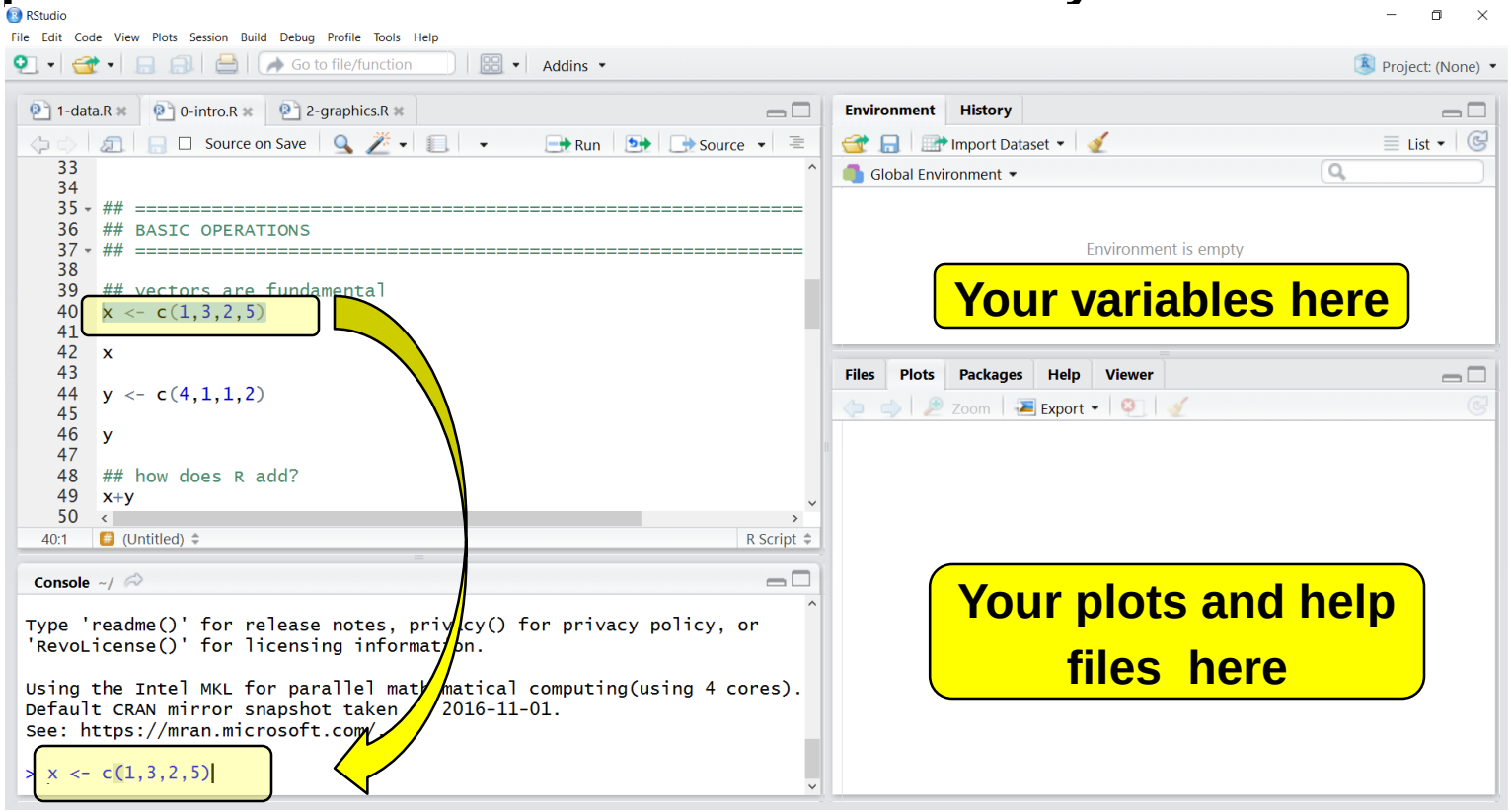
- Unzip the contents of file Rlab.zip to a new folder on your hard drive, let's say "Rlab"
- Set the `RStudio` working directory to that folder
  - `Session` → `Set Working Directory` → `Choose Directory...`
  - Navigate to Rlab, and Open the /programs directory
- *Focus console after executing from source* option: Moves the focus to the console after executing a line or selection of code within the source editor (you *will* thank me)
  - `Tools` → `Global Options` → `Code Editing...`
  - Check "`Focus console after executing from Source`"

Installing R packages:

- Install *dplyr* and *ggplot2* packages
  - `install.packages("dplyr")`
    `install.packages("ggplot2")`
- a) MAC OS X and Linux:
  - Open RStudio
  - In Menu, go to `Packages and Data` → `Package Installer`
  - Search for and install the above two packages (may need to choose a "mirror" - click on something in the USA) as follows:
  - Type in the name of one package, click "`get list`", check "`Install Dependencies`" and then "`Install Selected`"
  - Do the same for the other package
- b) Windows:
  - Open RStudio
  - In Menu, go to "`Packages`" → "`Install package(s)...`" and select each of the packages at top to install

Open R Lab Files

- In `RStudio`, pick `Open File` menu item and open file `/programs/0-Intro.R`
- Read each line, and copy *command lines* from the file and paste them into the RStudio console line by line

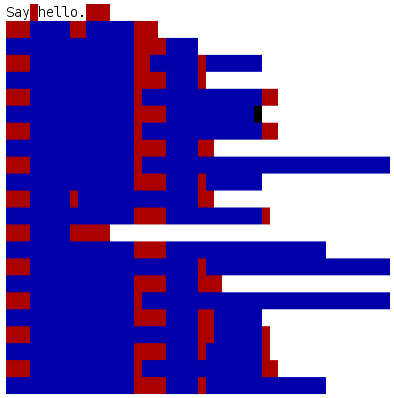


Part 5

INTRODUCTION TO PYTHON

Whitespace

- Esoteric imperative, stack-based programming language supporting integers, developed by Edwin Brady and Chris Morris at the University of Durham, England
  - In Whitespace, only spaces, tabs and linefeeds have meaning, while all other non-whitespace characters are ignored by the interpreter
  - Commands in Whitespace, as well as the entire code, are composed out of sequences of spaces, tab stops and linefeeds
  - **https://www.whoishostingthis.com/resources/whitespace-programming/#history**
  - **http://wiki.c2.com/?HelloWorldInManyProgrammingLanguages**
- **IDE**
  - **http://vii5ard.github.io/whitespace/**
  - **https://github.com/vii5ard/whitespace**
  - **https://github.com/loklaan/whitespace-lang**



WHITESPACE

Python

- Widely used high-level, general-purpose, interpreted, dynamic programming language
- Many different implementations, some *compiled*, some *interpreted*
- Design philosophy emphasizes code readability, and syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java

Side-bar: Interpreted or compiled?

- ***Languages*** **are not interpreted or compiled, but rather language *implementations* either interpret or compile code**
- ***Compilation*** **is translation from a *higher-level language* to a *lower-level one***
- ***Interpretation*** **is execution of a program, possibly one line at a time**
- **Python is *compiled*, since a .py file *usually* gives rise to a .pyc file, equivalent to a .class java file or the code in the `.text` section of a .exe or .dll .NET file**
  - **.pyc is not machine code, but often executes faster than machine code, like .class *bytecode* and .exe/.dll *assembly* files**
  - **.pyc is executed by the Python VM**
- **Some Python implementations are *completely* compiled** they usually include a *just-in-time* compiler that will compile Python byte code into native machine code (CPython transpiles Python into C), **while others interpret from .pyc files**
- **When no .pyc files are created, the python .py program is *interpreted* one line at a time, runs *slower* than native code**

Did you know?

- Virtual Machine Code (.NET IL, Java or python bytecode) is a more compact version of the original source code
  - It is *interpreted* by a virtual machine in order to *execute*, since it is no machine code. That's right, Java `.class` code is *interpreted* by the JVM, .NET `.dll` code by the CLR, and python `.pyc` code by the python runtie
- Some virtual machines generate **machine code** while interpreting the **virtual machine code** for the first time
  - This is called Just in time compilation (JIT)
  - Following invocations of *natively compiled* code will use this machine code directly, which leads to *faster execution*
  - Versions of Java, .NET, and Python VMs can all do this
    - Cython, is a Python language that is transpiled into C
    - PyPy, transpiles from RPython (*restricted subset* of Python that does not support some of the most "dynamic" features of Python) to C or LLVM
    - py2c (http://code.google.com/p/py2c) can transpile python code to C/C++
    - This Python is thus never interpreted, just always compiled
- Some Python VMs (Jython) use a JVM, others (IronPython) the CLR instead

Conclusion

- Although C#, Java, and Python compile down to bytecode..
- *Most* C# and Java compiles down to machine code with a JIT when it's first interpreted by the CLR or JVM
  - The compiled binary is cached and reused
- *Most* Python does ***not*** compile to machine code and its bytecode is interpreted by the Python VM

Bytecode disassemblers

- .NET: `ILDASM`
  - C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\Bin\
- Java: `OPAL` and `Atom`
  - https://atom.io/packages/java-bytecode-disassembler
- Python:

```
>>> def foo():
...     a = 2
...     b = 3
...     return a + b
...

>>> import dis
>>> dis.dis(foo)
```

  - http://akaptur.com/blog/2013/08/14/python-bytecode-fun-with-dis/

ABC ➔ Python

CWI Centrum Wiskunde & Informatica

- **Guido van Rossum, 1991**
- **ABC** is an imperative programming language developed at CWI, Netherlands by *Leo Geurts, Lambert Meertens, & Steven Pemberton*
  - Interactive, structured, high-level, and intended to be used instead of BASIC, Pascal, or AWK. It is not meant to be a systems-programming language but is intended for teaching or prototyping
  - Statement nesting is indicated by indentation, via the *off-side rule*
  - https://en.wikipedia.org/wiki/ABC_(programming_language)
- **ABC** had a major influence on the design of **Python**
  - Guido van Rossum previously worked for several years on the ABC system in the early 1980s

Versions

- First versions of Python 3.x released end of 2008
  - Included changes that made previously written Python 2.x code incompatible
- Python 2.x will reach development end of life in 2020
  - No longer a good idea to start new projects in Python 2.x
- We'll always use Python 3 in this class

Language Semantics

- Python uses whitespace to structure code
- A colon : denotes the start of an indented code block, following which all code must be indented by the same amount until the end of the block

```
for x in array:
    if x < pivot:
        less.append(x)
    else:
        greater.append(x)
```

- Semicolons ; can be used to separate multiple statements in a single line

```
a = 1; b = 2; c = 3
```

Language Semantics (continued)

- In Python, everything is a `Python object`
- Any text preceded by the hash mark (#) is ignored by the Python interpreter
- When assigning a variable in Python, you are creating a reference to the *right hand side*
  - *There is no copying taking place*
  - *To create a copy, instantiate a list of an element*
- When you pass objects as arguments to a function, new local variables are created that are references to the original objects, *without any copying*
- Object references have no types associated with them..
- However, Python is a *strongly typed language*
  - *Implicit conversions occur at runtime*
- Most objects in Python are *mutable*
  - *Strings and tuples are immutable*

Scalar types: `strings`, `ints`, `floats`, etc.

- Strings can be delimited by single `'` or double `"` quotations
- Use backslash to encode special characters (e.g. `\t` for TAB)
  - `len("\t")`
- Use raw strings to represent backslashes (e.g. `r"\t"`)
  - `len(r"\t")`
- Use triple quotes for multiline strings
  - ```
    Myaddress = """2400 Beacon St.
    Chestnut Hill
    MA 02467"""
    ```
    ```

Functions and Lambdas

- A function is a rule for taking zero or more inputs and doing a *transformation* into an output
- ```
  def double(x):
    """docstring: multiply by 2"""
    return x * 2
  ```
- Python functions are *first-class*: assign them to variables and pass them as arguments
  - ```
    def apply_to_one(f):
      return f(1)
    ```
- Short *anonymous* functions are called *lambdas*
  - ```
    y = apply_to_one(lambda x: x + 4)  #5
    ```
- ```
  another_double = lambda x: 2 * x #the same as..
  def another_double(x): return 2 * x
  ```
- ```
  def myprint(message="hello world"):
    print message
  myprint() #prints "hello world"
  myprint("hello dino")  #prints "hello dino"
  ```
  ```

Functions are objects

- `states = [' Alabama', 'Georgia!', 'Georgia', 'georgia', 'massachusetts?"]`
- ```
  import re
  def clean_strings(strings):
    result = []
    for value in strings:
      value = value.strip()
      value = re.sub('[!?', '', value)
      value = value.title()
      result.append(value)
  ```
- `clean_strings(states) #['Alabama', 'Georgia', 'Georgia', 'Georgia', 'Massachusetts']`
  ```

Function Definition

- Python function definitions begin with the **def** keyword
- All class functions and data members have essentially public scope as opposed to languages like Java and C#, which can impose private scope
- The built-in **__init__** function (with two leading and two trailing underscore characters) can be loosely thought of as a *constructor*
- All class function definitions must include the `self` keyword as the first parameter

Function Calling

- def <name>(<parms>): <body>
  - <body> compiled to IL, not executed
  - call name(<args>) executes the body

Class functions

- Even though all class function definitions must include the `self` parameter, when a class function is called the `self` argument isn't used
- But when accessing a class function or variable, the `self` keyword with dot notation must precede all function and variable names

Long statements

- **Python statements can be long..**
- **One way to allow a long statement to span two lines is to *surround the statement with parentheses*:**

```
def makematrix(self, rows, cols):
  result = ([[0 for j in range(cols)]
    for i in range(rows)])
  return result
```

No Main Entry point

- **Unlike many programming languages, there's *no main program entry point* in Python and execution begins with the *first executable statement***
  - **This normally leads to a program structure where the code that C# programmers would consider the Main method is at the *end of the source code file***

No Native array type (!)

- One of the quirks of Python
  - Instead, the basic collection type is a list, roughly analogous to a C# `ArrayList` collection
  - Matrices are stored in list-of-lists-style syntax

No variable declaration

- Variables are not declared as in C#
  - Instead, they come into existence as they're encountered in the script
  - Not true for global variables however..

Duck Typing

- A style of dynamic typing in which an object's *current set of methods and properties* determines the valid semantics, rather than its inheritance from a particular class or implementation of a specific interface
  - "*when I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck*"
  - In duck typing, one is concerned with just those aspects of an object that are used, rather than with the type of the object itself
- In a non-duck-typed language, one creates a function that takes an object of type Duck and calls that object's walk() and quack() methods
  - In a duck-typed language, the equivalent function would take an object of any type and call that object's walk and quack methods
  - If the object does not have the methods that are called then the function signals an error *at run-time*

Exception handling

```
def attempt_float(x):
    try:
      return float(x)
    except:
      return x
```

Container types

- **Tuple**: immutable sequence
  - () (23,) (23, 45) tuple('ciao')
- **List**: mutable sequence (a "vector")
  - [] [23] [23, 45] list('ciao')
- **Set**: mutable:
  - set() set((23,)) set('ciao')
  - immutable variant (frozenset)
- **Dict**: map key→value by hashtable
  - {} {2:3} {4:5, 6:7} dict(ci='ao')
- All containers support:
  - `len(c)`, looping (`for x in c`), membership testing (`if x in c`)

Lists

- **Most fundamental data structure in Python: *Ordered collection***
  - `a = [1,2,3]`
  - `b = [4.5.6]`
  - `List_of_lists = [a, b, []]`
  - `s = sum(a)`
  - `one = a[0]`
  - `three = a[-1]  #last element`
  - `two = a[-2]  #next-to-last eleme`
  - `digits = range(10)`
  - `first_three = digits[:3]`
  - `minus_first_three = digits[3:]`
  - `one_to_four = digits[1:5]`
  - `copy_of_digits = digits[:]`
  - `x.extend([10,11,12])`
  - `x.append(13)`
  - `x, y = [1,2]`

Sorting


```
x = [4,1,2,3]
y = sorted(x)  #x remains unchanged
x.sort()  #x is sorted in place
wc = sorted(word_counts.items(),
            key = lambda (word, count): count
            reverse = True) #instead of comparing
            elements themselves, compare results
            of a function that you specify with
            'key'
```

- Built-in `bisect` module implements binary search and insertion into a sorted list
  - `Bisect.bisect` finds the location where an element should be inserted to keep the list sorted
  - ```
    import bisect
    c = [1,2,2,2,3,4,7]
    bisect.bisect(c,2)  #4
    bisect.bisect(c,5)  #6
    bisect.insort(c, 6) #[1,2,2,2,3,4,6,7]
    ```
    ```

Tuples

- **Lists' immutable cousins**
  - `mylist = [1, 2,3]`
  - `mytuple = (1,2,3)`
- **Tuples are a convenient way to return multiple values from functions:**
  - ```
    def sum_product(x, y):
      return (x + y), (x * y)
    s, p = sum_product(10, 10)
    ```
- **Multiple assignments:**
  - `x, y = 1, 2`
- **Python variable swap:**
  - `x, y = y, x`
    ```

Dictionaries

- **Lists of key/value pairs, or *named arrays***
  - `empty = {}`
  - `also_empty = dict()`
  - `grades = {"dino": 3.9, "elon": 4.0}`
  - `elon_grade = grades["elon"]`
  - `elon_grade = grades.get("elon", 0)`
  - `dinograde_p = "dino" in grades`
  - ```
    json = {
       "title": "my blog",
        "hashtags": ["#bigdata", "#crypto", "#quantum"]
    }
    ```
  - `json.keys()`
  - `json.values()`
  - `json.items()`
- **Dictionary keys are immutable**
  -
    ```

Defaultdict

- Like a regular dictionary, except when you try to look up a key that isn't there, it first adds a value for it using a zero-argument function you provide when you create it
- Useful whem using dictionaries to collect results by some key and don't want to check repeatedly for key existence
  - ```
    From collections import defaultdict
    word_counts = defaultdict(int)
    for word in document:
      word_counts[word] += 1
    ```
  - ```
    dd = defaultdict(dict)
    dd["dino"]["City"] = "Boston"
                        #{"dino": {"City": "Boston"}}
    ```

Counter

-
```
From collections import Counter
word_count = Counter([“to”, “be”, “or”, “not”,\
“to”, “be”])
```

Sets

- **Unordered collection of distinct elements**
  - `s = set()`
  - `s.add(1)`
  - `s.add(2)`
  - `s.add(2)`
  - `p = 2 in s`
- **Performance:**
  - `stopwords_list = [“a”, “the”, …`
    `p = “hello” in stopwords  #slow`
  - `stopwords_set = set(stopwords_list)`
    `p = “hello” in stopw-rds_set  #fast`
- **Distinct:**
  - `word_list = [‘the’, “cat”, “jumps”, …]`
  - `distinct_word_list = set(world_list)`

List comprehensions

- Transformations of lists:
  - `even_numbers = [x for x in range(100) if x % 2 == 0 ]`
  - `even_set = {x for x in range(100) if x % 2 == 0 }`
  - `zeroes = [ 0 for _ in range(100)]`
  -
```
pairs = [ (x,y)
            for x in range(100)
            for y in range(100) ]  #10,000 pairs
```
  -
```
some_tuples = [(1,2,3), (4,5,6), (7,8,9)]
flattened = [x for tup in some_tuples for x in tup]
flattened  #[1,2,3,4,5,6,7,8,9]
```
- List comprehensions is the pythonic equivalent of LINQ in .NET
- We'll use list comprehensions *a lot* in data science because we can use them to express *anamorphisms* (unfolds or maps) and *catamorphisms* (projections) of data structures
  - Get ready for this!

Generators

- ***Generators***, **sometimes called *Coroutines*, are sequences you can iterate over, but which are only produced lazily (as needed)**
- **You can create generators with functions and yield:**
  - ```
    def lazy_range(n):
      """a lazy version of range()"""
      i = 0;
      while i < n:
        yield i
        i += 1
    ```
  - ```
    # to consume yielded values:
    for i in lazy_range(10)
      print(i)
    ```
- **You may also create generators by using list comprehensions wrapped in parenses:**
  - ```
    lazy_ints_under_100 = (i for i in range(100))
    ```
    ```

Iterators

- **Standard `itertools` library has a collection of generators for common data algorithms**
  - ```
    import itertools
    first_letter = lambda x: x[0]
    names = [ 'Alex', 'Aria', 'Wally', 'Will',
    'Ariana', 'Steve']
    for letter, names in itertools.groupby(names,
    first_letter):
      print(letter, list(names))
     # A ['Alex, 'Aria']
       W ['Wally', 'Will']
       A ['Ariana']
       S ['Steve']
    ```
    ```

Control flow

```
if 1 == 2:
  print("uh-oh")
elif 1 == 3:
  print("uh-oh-again")
else:
  print("whew..")
```

```
while x < 100:
  print(x)
```

```
for x in range(100):
  if x < 100:
    continue
  if x > 100:
    break;
  print(x)
```

Enumerations

-
```
#nicely functional
for i, document in enumerate(documents):
  do_something(i, document)
```
-
```
#unpythonic
for i in range(len(documents)):
  document = documents[i]
  do_something(I, document)
```
-
```
#also unpythonic
i = 0
for document in documents:
  do_something(i, document)
  i += 1
```

File IO

- `path = 'myfolder/mybigdata.txt'`
- ```
  f = open(path)
  for line in f:
    print(line)
  #EOL marker intact
  ```
- ```
  Lines = [x.rstrip() for x in open(path)]
  #EOL-free
  ```
- **Using analog:**
  - ```
    with open(path) as f:
      lines = [x.rstrip() for x in f]
    #automatically closes the file when exiting the
    with block
    ```
- ```
  with open(path, 'rb') as f:
    data.decode('utf8')
  ```

Object Oriented Python


```
Class Set:
    def __init__(self, values=None):
      """ctor"""
      self.dict = {} #each instance has its own
                     #dict which is what we use
                     #to track membership
      if values is not None:
        for value in values:
          self.add(value)

    def add(self, value):
      self.dict[value] = True

    def contains(self, value):
      return value in self.dict

    def remove(self, value):
      del self.dict[value]
```

Currying

- **Partially applying functions to create new functions**
  - def exp(base, power):
    return base ** power
  - def two_to_the(power):
    return exp(2, power)
  - From functools import partial
    two_to_the = partial(exp, 2)
  - Print(two_to_the(3))

Function Oriented puzzle

- **Let's say we want to create a higher-order function that takes as input some function `f` and returns a new function that for any input returns *twice* the value of `f`**

```
def dpubler(f):
  def g(x):
    return 2 * f(x)
  return g
```

- **Works in most cases:**

```
def f_plus_1(x)::
  return x + 1;
g = doubler(f_plus_1)
print(g(3))  # 8 = (3 + 1) * 2
```

- **But:**

```
def sum(x, y)::
  return x + y;
g = doubler(sum)
print(g(1,2))
```

`args` and `kwargs`

- **What we need is a way to specify a function that takes arbitrary arguments:**

```
def magic(*args, **kwargs):
  print ("unnamed args: ", args)
  print ("keyword args: ", kwargs)
magic(1,2, key1 = "nu", key2 = 'rocks!");
```

- **`args` is a tuple of its unnamed arguments and `kwargs` is a dictionary of its named arguments. So now we can:**

```
def dpublerr(f):
  """works no matter the inputs"""
  def g(*args, **kwargs):
    """pass all arguments to f"""
    return 2 * f(*args, **kwargs)
  return g
```

- **And now:**

```
g = doublerr(sum)
print g(1, 2)  # 6:
```

Zippers

- `zip` transforms multiple lists into a single list of tuples of corresponding elements
  - `list1 = ['a', 'b', 'c']`
  - `list2 = [1, 2, 3]`
  - `zipper = zip(list1, list2)  #[('a', 1), ('b', 2), ('c', 3)]`
  - `Orig_letters, orig_numbers = zip(*zipper)`
    `# * performs argument unpacking`
  - `def add(a, b): return a + b`
  - `add(1, 2)  #3`
  - `add([1,2])  #TypeError!`
    `add(*[1,2]) #3`

www.rojakpot.com

Good Python Videos

- **Google Developer Python Course**
  - **https://www.youtube.com/playlist?list=PLfZeRfzhgQzTMgwFVezQbnpc1ck0I6CQl**
- **30 minute crash course in Python:**
  - **https://learnxinyminutes.com/docs/python/**
- **Intro to Python for Data Science**
  - **https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-1-python-basics?ex=1&open_outline=true**
- **Introduction to Deep Learning with Python**
  - **https://www.youtube.com/watch?v=S75EdAcXHKk**
- ***Optional*: Lecture 1 | Machine Learning (Stanford)**
  - **https://www.youtube.com/watch?v=UzxYlbK2c7E**
- **Install Anaconda (see next slide for instructions) and test drive here:**
  - **http://conda.pydata.org/docs/test-drive.html**

Reference

- https://docs.python.org
- www.python.org
- www.diveintopython.org

Part 6

BANANAS

Coming up..

Homework (due Wednesday)

- Download and install `Anaconda with Rstudio`, `Visual Studio Code`, install `ggplot2` and `dplyr` R packages
- Optional: Play with `Whitespace`
  - Appreciate and respect whitespace in Python. Your code won't run without!


CLASS
DISMISSED
iStockphoto