

応用心理学 I
データ収集後探索的解析（テキストマイニング） その1

人文社会系学生のための Rブートキャンプ

明治大学 研究・知財戦略機構
佐藤浩輔

位置づけ

- 7/25 午後 **Rブートキャンプ←この講義**
- 7/26 午前 質的データの量的コーディング*
- 7/26 午後 (前半) テキストマイニング理論編
(後半) テキストマイニング実践編

講義にあたって

- 自己紹介
- 講義について
 - 概要
 - 扱う内容
- 講義の進め方
 - 講義の形式について
 - Zoomの使い方
 - 講義時間
- 課題について
- サポートサイトについて

講義について：概要

①Rブートキャンプ（この講義）

- コンピュータを用いた基本的なファイル操作・Rを用いたデータ処理の基礎を学ぶ

②テキストマイニング入門：講義編

- テキストマイニング（計量テキスト分析）と自然言語処理の概要を学び、何ができるかを知る
- テキストを用いた研究のデザイン、研究計画の立て方を学ぶ

③テキストマイニング入門：実習編

- 実際にデータを扱いながら学ぶ
 - ①前処理を行い、分析ができるようデータを加工する
 - ②分析を行い、解釈する

講義について：扱う内容

- この講義で学ぶこと：
 - コンピュータを用いて分析を行うために最低限必要な知識を身に着けること
 - ①コンピュータの使い方の初歩
 - 基本的なファイル操作
 - 情報検索の仕方の初歩
 - ②Rの使い方
 - データ操作
 - 発展的な内容も扱います（★マークで表示）
 - 講義内では詳しくは扱いませんが、興味があれば調べてください
- この講義では扱わないこと
 - 各種統計手法の詳細
 - ベイズ

講義の進め方①

- この講義ではR言語の使い方について説明します
 - 実習部分を試すにはRがインストールされている環境*が必要です
 - オンライン講義なので：
 - 講義動画を見返す前提の構成です
 - × 全部記憶する
 - 必要な時に検索・参照する
 - スライド・Rコードは講義終了後公開します
- 今回はハンズオン形式にはしません
 - 皆さんが操作するのを待つ時間はとりません
 - あとで復習しながら試してください
 - 復習用のエクササイズ（演習）を用意しています
 - 講義中に解説・解答していきます
 - 解答ファイルは講義後にアップロードします

*Windows環境を想定しています

講義の進め方②

- Zoomの使い方

- マイク/ビデオについて

- 喋っていないときはマイクを**ミュート**
 - ビデオ表示（カメラ）はオフでも**OK**です
 - 通信量節約のため基本的に**オフ**で

- 講義中にコメント/質問等あるときは

- ①直接発言する
 - ②Zoomのチャット機能を使う のどちらもOK

講義の進め方③

- 講義時間：13:00~16:25（予定）
 - ところどころ休憩を入れます
 - 無理のないように受講してください
 - 飲み物を補給する（熱中症予防）
 - 画面を凝視しすぎない
 - 休憩中に体を動かす

課題について

- 課題があります
 - 7/25-7/26の講義で学んだことを使ってテキストを分析する内容です
 - 課題はR以外の言語を使ってもOKです
 - ※Rの知識を直接問うような課題ではありません
 - 詳細は7/26の講義の終了時にお伝えします

サポートサイトについて

- GitHub上の講義ページ
 - <https://github.com/satocos135/lecture2020shimane>
 - このページに講義資料をアップロードします
 - スライド
 - 分析用データ など
- Discourse
 - 掲示板形式のWebアプリケーション(OSS)
 - 受講生・聴講生限定で閲覧および書き込みができます
 - 事前に送った招待リンクからアクセスするとアカウントを作ってログインできます
 - 技術的サポートは基本こちらで
 - メールだと僕がパンクするので

①コンピュータの使い方の初歩

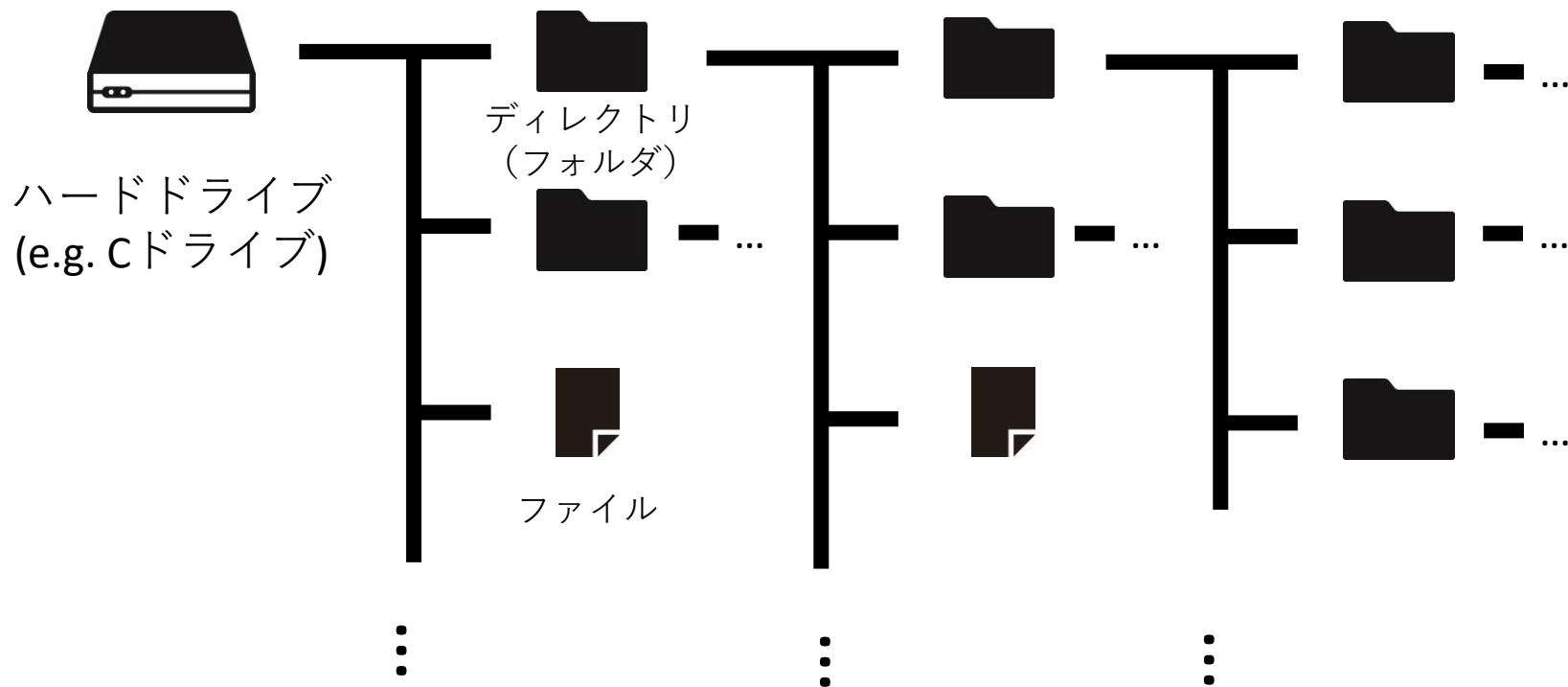
1. ファイル操作の基本
2. 情報の調べ方
3. その他Tips

ファイル操作の基本

基本的な事柄

- ファイルとフォルダ
 - ファイル：データやプログラムを管理する単位
 - データファイル：データを格納するファイル
 - e.g., docx, txt
 - 実行ファイル：プログラムを実行するファイル
 - e.g. exe
 - フォルダ（ディレクトリ）：ファイルを整理する入れ物
 - 階層的に配置できる
- ファイル/フォルダの操作
 - コピー：複製する
 - 移動：違う場所へ移す
 - 削除：その場所から消す
 - リネーム：ファイルの名前を変える

ファイルシステム

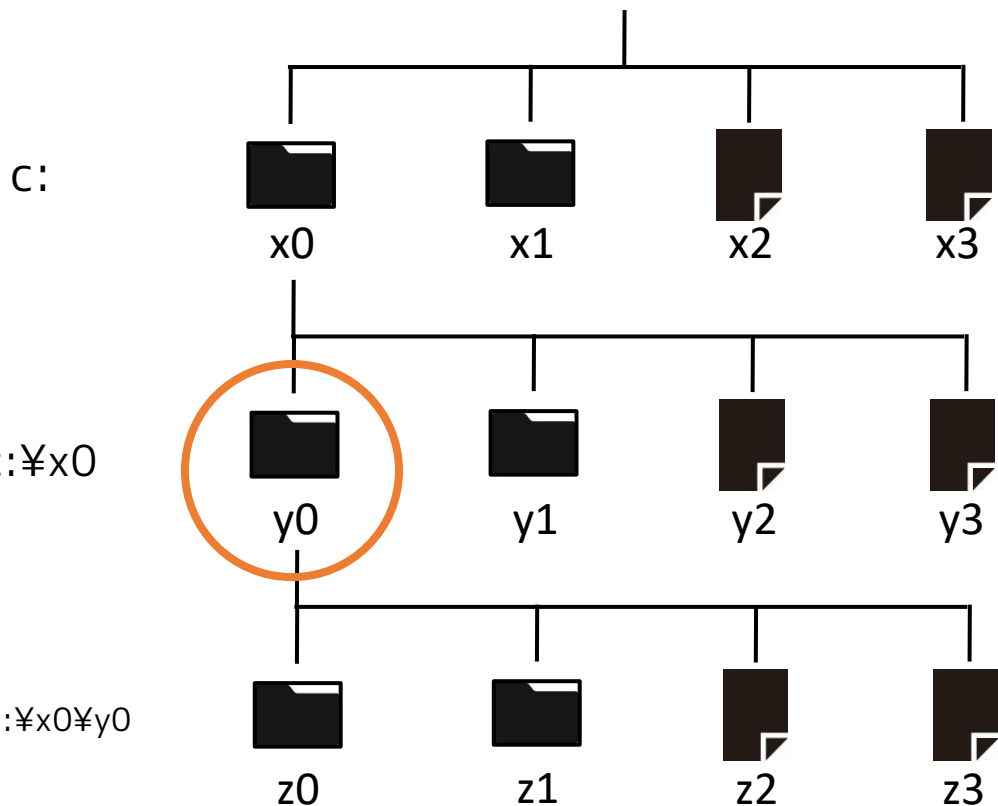


- パス（アドレス）：ファイルの場所を示す住所
 - ディレクトリ名を区切り文字でつなげる
 - 円記号(¥)またはスラッシュ(/), バックスラッシュ
 - 絶対パス：フルの住所
 - 相対パス：今いるディレクトリを基準にした住所
 - 今いるディレクトリ内のディレクトリ/ファイル：
そのまま名前でアクセスできる
 - 今いるディレクトリ： .
 - ひとつ上のディレクトリ： ..
 - ホームディレクトリ： ~

カレントディレクトリが
 $y_0(c:\backslash x_0 \backslash y_0)$ として



Cドライブ(c:)



x_2 へのパス：

絶対パス: $c:\backslash x_2$

相対パス: $..\backslash..\backslash x_2$

y_1 へのパス：

絶対パス: $c:\backslash x_0 \backslash y_1$

相対パス: $..\backslash y_1$

z_3 へのパス：

絶対パス: $c:\backslash x_0 \backslash y_0 \backslash z_3$

相対パス: z_3

コマンドライン

- CLI (**C**ommand **L**ine **I**nterface)
 - 文字によってやりとりを行うインターフェース
 - CUI(**C**haracter **U**ser **I**nterface) cf. GUI(**G**raphical **U**ser **I**nterface)
 - コマンドプロンプト/シェル/ターミナル
 - コマンドを使って操作する
 - 半角スペース区切り
 - 引数やオプションをとるコマンドもある
 - コマンドの例：
 - cd: ディレクトリを移動する(**c**hange **d**irectory)
 - cp: ファイルをコピーする(**c**opy)

```
[root@localhost ~]# ping -q fa.wikipedia.org
PING text.pmtpa.wikimedia.org (208.80.152.2) 56(84) bytes of data.
^C
--- text.pmtpa.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 540.528/540.528/540.528/0.000 ms
[root@localhost ~]# pwd
/root
[root@localhost ~]# cd /var
[root@localhost var]# ls -la
total 72
drwxr-xr-x. 18 root root 4096 Jul 30 22:43 .
drwxr-xr-x. 23 root root 4096 Sep 14 20:42 ..
drwxr-xr-x.  2 root root 4096 May 14 00:15 account
drwxr-xr-x. 11 root root 4096 Jul 31 22:26 cache
drwxr-xr-x.  3 root root 4096 May 18 16:03 db
drwxr-xr-x.  3 root root 4096 May 18 16:03 empty
drwxr-xr-x.  2 root root 4096 May 18 16:03 games
drwxrwx--T.  2 root gdm  4096 Jun  2 18:39 gdm
drwxr-xr-x. 38 root root 4096 May 18 16:03 lib
drwxr-xr-x.  2 root root 4096 May 18 16:03 local
lrwxrwxrwx.  1 root root    11 May 14 00:12 lock -> ../run/lock
drwxr-xr-x. 14 root root 4096 Sep 14 20:42 log
lrwxrwxrwx.  1 root root    10 Jul 30 22:43 mail -> spool/mail
drwxr-xr-x.  2 root root 4096 May 18 16:03 nis
drwxr-xr-x.  2 root root 4096 May 18 16:03 opt
drwxr-xr-x.  2 root root 4096 May 18 16:03 preserve
drwxr-xr-x.  2 root root 4096 Jul  1 22:11 report
lrwxrwxrwx.  1 root root    16 May 14 00:12 run -> ../run
drwxr-xr-x. 14 root root 4096 May 18 16:03 spool
drwxrwxrwt.  4 root root 4096 Sep 12 23:50 tmp
drwxr-xr-x.  2 root root 4096 May 18 16:03 yp
[root@localhost var]# yum search wiki
Loaded plugins: langpacks, presto, refresh-packagekit, remove-with-leaves
rpmfusion-free-updates                               | 2.7 kB      00:00
rpmfusion-free-updates/primary_db                     | 206 kB      00:04
rpmfusion-nonfree-updates                             | 2.7 kB      00:00
updates/metalink                                       | 5.9 kB      00:00
updates                                                 | 4.7 kB      00:00
updates/primary_db                                     73% [=====] | 62 kB/s | 2.6 MB 00:15 ETA
```

```
Macintosh HD — top — 80x24

Processes: 210 total, 2 running, 9 stuck, 199 sleeping, 901 threads   23:30:03
Load Avg: 1.40, 1.75, 1.00  CPU usage: 4.15% user, 4.40% sys, 91.44% idle
SharedLibs: 1648K resident, 0B data, 0B linkedit.
MemRegions: 31278 total, 1892M resident, 117M private, 564M shared.
PhysMem: 5893M used (1191M wired), 10G unused.
VM: 523G vsize, 1026M framework vsize, 0(0) swapins, 0(0) swapouts.
Networks: packets: 12105/8925K in, 11907/1964K out.
Disks: 80156/2205M read, 21235/425M written.

PID  COMMAND      %CPU  TIME    #TH  #WQ  #PORT  MEM    PURG    CMPR  PGRP  PPID
592  screencaptur  0.0   00:00.02  7    5    55+    1952K+ 20K+    0B    262  262
590  mdworker      0.0   00:00.01  3    0    44     2032K  0B      0B    590  1
589  mdworker      0.0   00:00.01  3    0    44     1572K  0B      0B    589  1
588  top           1.7   00:00.51 1/1    0    22+    2860K  0B      0B    588  584
584  bash          0.0   00:00.00  1    0    15     588K   0B      0B    584  583
583  login         0.0   00:00.01  3    1    28     1228K  0B      0B    583  482
574  auditd        0.0   00:00.00  2    0    25     560K   0B      0B    574  1
567  System Prefe  0.0   00:03.23  3    0    270    39M    8364K  0B    567  1
561  systemstatsd  0.0   00:00.01  2    1    19     1040K  0B      0B    561  1
560  com.apple.We  0.0   00:01.42  9    0    229    25M    0B      0B    560  1
558  com.apple.We  0.0   00:05.07 15    3    224    151M    1716K  0B    558  1
555  bash          0.0   00:00.00  1    0    15     604K   0B      0B    555  554
554  login         0.0   00:00.01  3    1    28     1176K  0B      0B    554  482
550  bash          0.0   00:00.00  1    0    15     608K   0B      0B    550  549
```

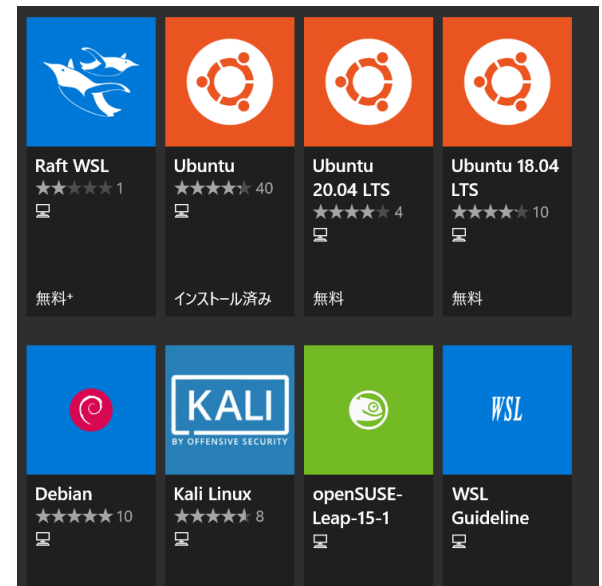
```
C:\WINDOWS\system32\cmd.exe

Microsoft Windows [Version 10.0.18362.959]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ysatoc>
```

★ Windows Subsystems for Linux(WSL)

- Windows内でLinuxのシステムを導入できる
 - Windows 10またはWindows Serverに導入可能
 - 便利なLinuxコマンドを使えたりする



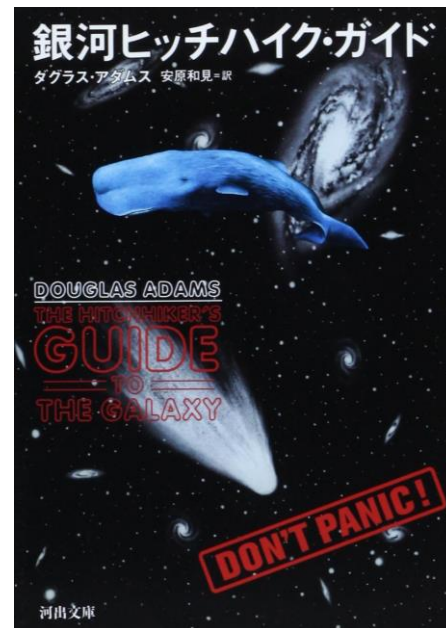
「四十二だと!」 ルーンコールが叫んだ。「七百五十万年かけて、それだけか?」

「何度も徹底的に検算しました」 コンピュータが応じた。

「まちがいなくそれが答えです。率直なところ、みなさんのほうで究極の疑問が何であるかわかっていなかったところに問題があるのです」

——ダグラス・アダムズ『銀河ヒッチハイク・ガイド』

情報の調べ方



検索スキルの重要性

- 研究やプログラミング：自分で調べる作業が必須
 - 「すべてを知っている先生」みたいな人はなかなかいない
 - いたとしても忙しい/時間を割いてくれるとは限らない
 - 自分で情報を集めて問題を解決する必要がある
- 情報検索に伴う問題
 - うまく情報にたどりつけるか
 - 信頼できる情報を集められるかどうか

基本的な考え方

- 適切な情報を得るためには適切な場所を探す必要がある
 - 医療に関する情報を得るには：
 - × 知り合いに相談する
 - 医療従事者に相談する
- 適切な回答を得るためには適切なクエリ（質問文）を投げる必要がある
 - 不適切なクエリ：「R 動かない」
 - 背景情報や様々な詳細が必要

検索におけるノイズ

- 主題と関係ない情報
 - 似たようなキーワードを持つが関係ない情報
- 主題と関係ある情報
 - 誤った情報
 - 誤解・デマ, etc.
 - 信頼性の低い情報
 - 科学的知見に基づかない記述・個人の思い込み・噂, etc.
 - 低質な情報
 - 不正確な記述
 - 自動生成されたコンテンツ(e.g. 英文記事の自動翻訳), etc.

⇒ノイズをよけて主題に関する情報を得たい

一般的な方策

- 得たい情報：
 - 主題に関係があり、信頼性が高く、かつ正確で良質な情報
- 対策
 - 検索テクニックを使う
 - 適切なキーワードを使う
 - 完全一致・フレーズ検索：引用符(','')でくくる
 - エラーメッセージを貼り付けて調べるのはかなり有効
 - 特定のキーワードを除外する：マイナス(-)をつける
 - ノイズになるサイトをブロックする
 - 検索対象に関する知識を増やす
 - 詳しくなればなるほど、情報の良しあしを判断できる
 - 複数の情報源にあたる

その他Tips

タッチタイピングを身に付けてほしい

- 入力：PC操作のボトルネック
- タッチタイピングを身に着けることで：
 - 入力速度が速くなる
 - 画面や資料を見ながら入力できる
 - ミスタイプが減る
- タッチタイピングを覚えなくてもいいけど：
 - 最低でも60cpm(1秒に1文字)くらいの入力スピードは欲しい
(将来キーボードに替わる入力デバイスが現れたら事情は変わるかもしれない)

過つは人の業

- コンピュータは悪くない
 - コンピュータは命令したことを実行するだけ
 - 命令していないことは実行しない
 - ミスするのは人間の責任
- エラーは必ず起きる
 - 「間違いを犯さないのは何も新しいことをしようとしてこなかった人間」 (Albert Einstein)
 - エラー出力を恐れない・ちゃんと中身を読む
 - 間違いが起きても修正することが重要
 - 致命的な間違いは（そんなに）多くない
- とにかく手を動かして慣れる
 - ほとんどは宣言的知識というより手続き的知識（自転車の乗り方を覚えるようなもの）

②Rの使い方

1. RとRStudio
2. Rの基本操作
 - はじめに覚えておくこと
 - 演算
 - 関数と制御構造
3. データハンドリングと作図
4. 一歩進んだデータハンドリング

Rとは

" R is a language and environment for
statistical computing and graphics. "

「Rは統計計算とグラフィックスのためのプログラミング言語環境です」



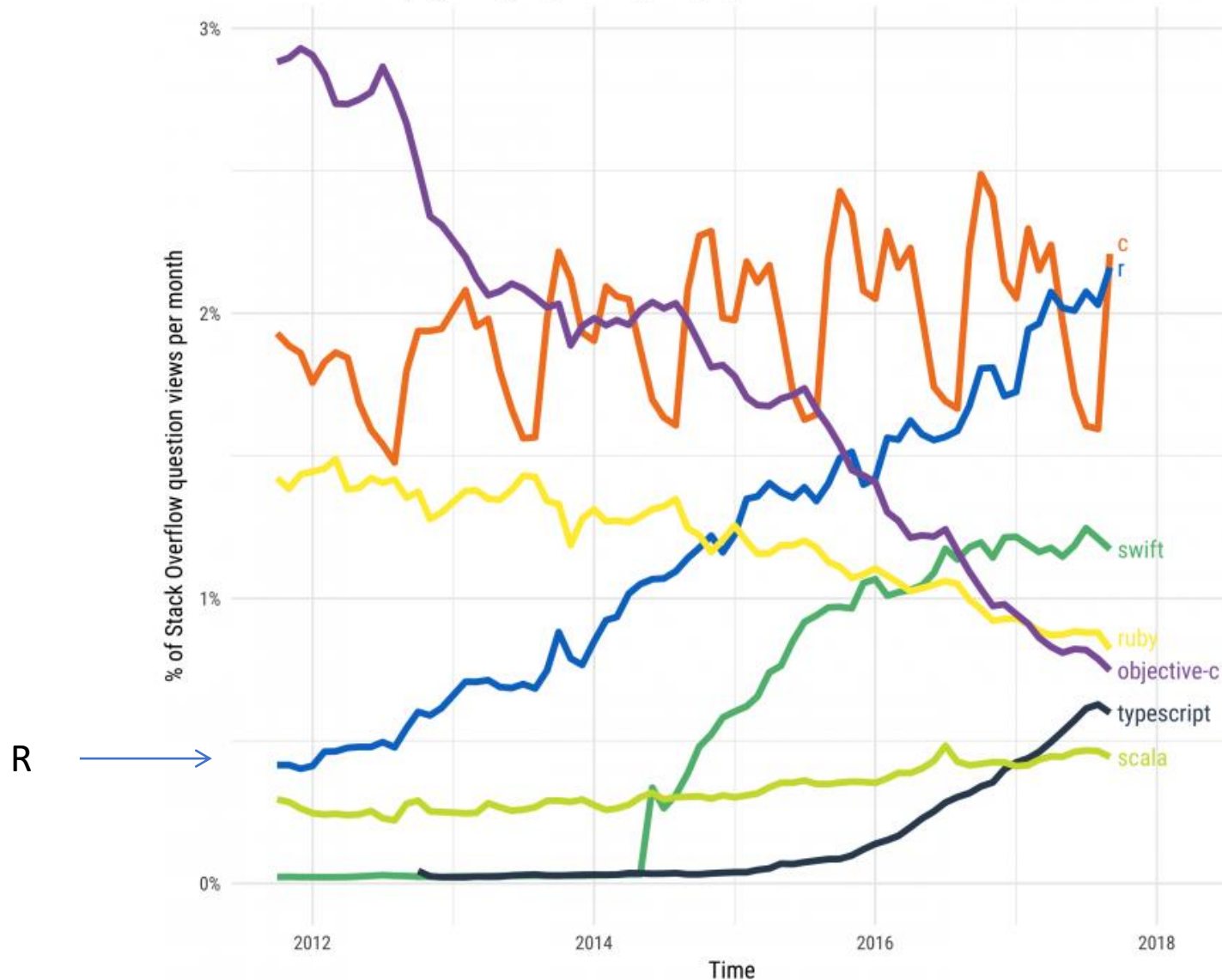
<https://www.r-project.org/about.html>

Why R ?

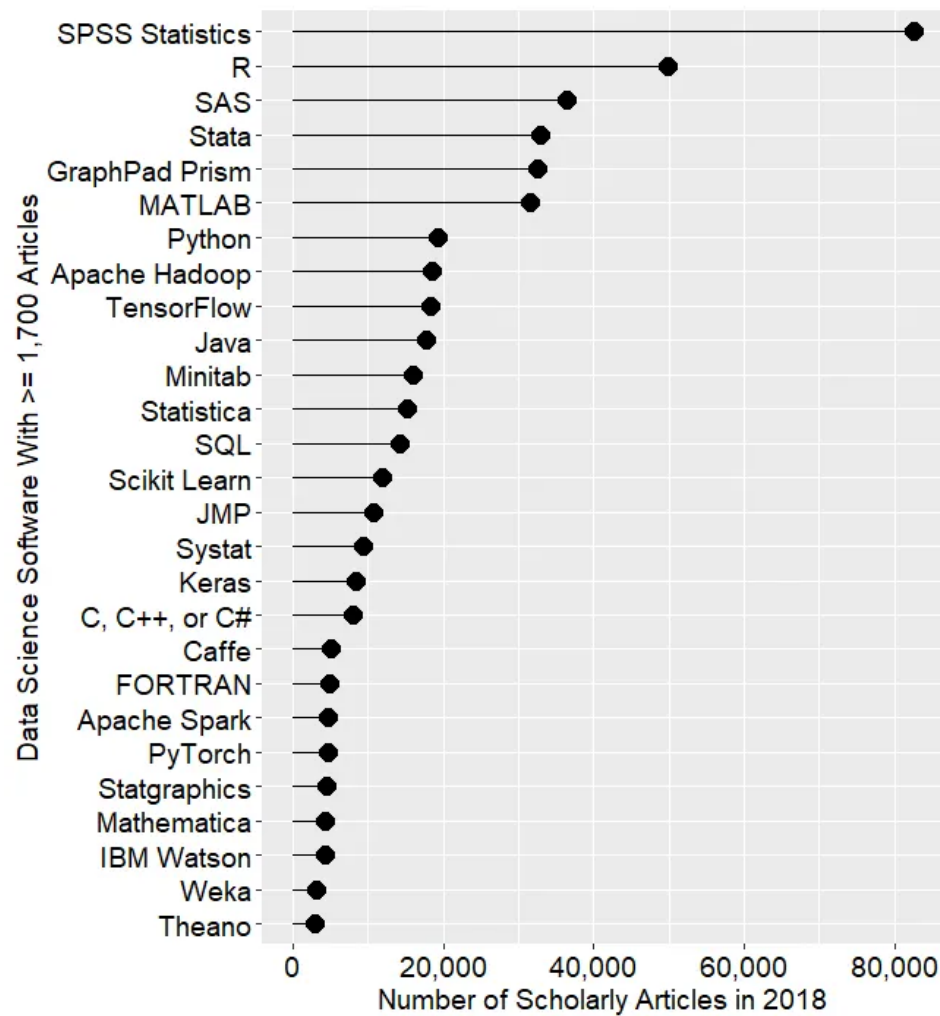
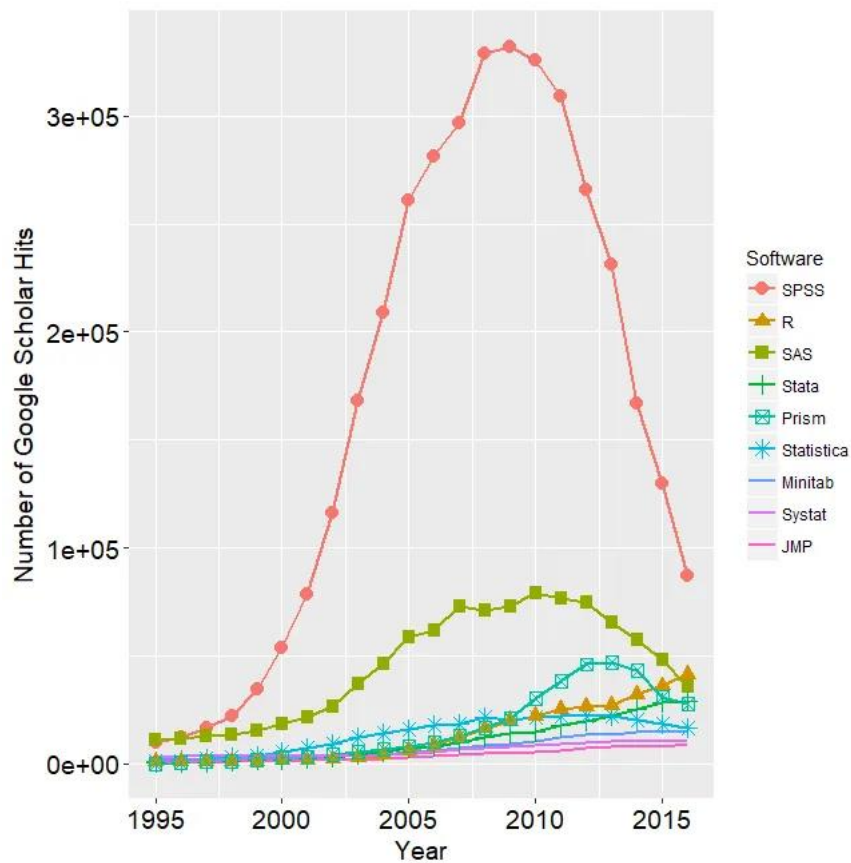
- **無料**である（どこでも分析環境を構築できる）
- 統計・科学技術計算用途で開発された言語である
 - 特に統計に強い
- 科学技術コミュニティで使われている
 - バグや脆弱性があっても早期に発見/対策できる
 - Rを解説する書籍やWebサイトもだいぶ増えてきた
- 研究者が開発した先進的な分析パッケージも使える

Stack Overflow Traffic to Programming Languages

Based on visits to Stack Overflow questions from World Bank high-income countries.
The more-visited languages of Python, JavaScript, Java, C#, and PHP were omitted.



The Impressive Growth of R - Stack Overflow Blog |
<https://stackoverflow.blog/2017/10/10/impressive-growth-r/>



Rの得手不得手

- 得意なこと
 - 統計計算
 - こまごまとしたデータ処理
- 苦手なこと (心理学で扱う程度のデータならあまり気にならないはず)
 - 速さが要求される計算
 - 大量のデータ(1GB~)の処理
 - 今は高速に扱えるパッケージもあるかも
- 科学技術計算の他の選択肢
 - Python
 - Julia



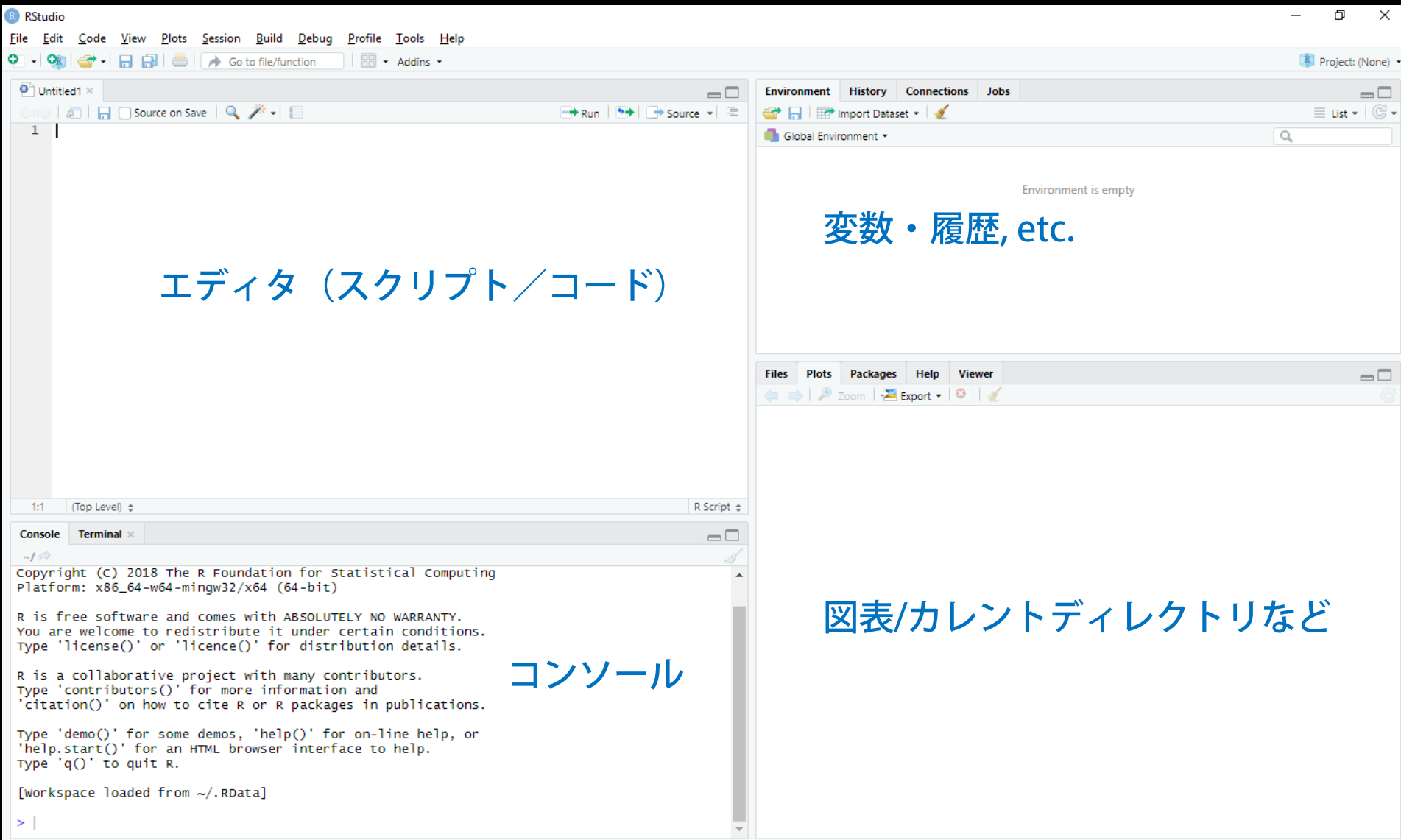
RStudio

- Rの統合開発環境 (Integrated Development Environment: IDE)
 - Rを使って分析したり開発したりが便利
 - e.g. 入力補完
 - 無料版と有償版がある
 - サーバ版もある

※この講義ではRStudio以外の開発環境を使ってもOKです



RStudioの画面（初期設定）



この講義の他のすべてを忘れても これだけは覚えて帰ってほしいこと

- 分析コードを残す！
 - 分析結果 = データ + 処理
 - データと処理（コード）が残されていれば、分析のすべては再現できる
 - 処理が不明瞭なら結果の信頼性が疑われる
 - 人間が間違える生き物であることを忘れない
- コードはエディタ上で実行する
 - コンソールにいちいち貼り付けて実行するのは時間と手間の無駄
 - 人生は短いので時間は有効に使いたい
 - **Ctrl+Enter** で選択した部分のみをコンソールに送って実行できる
- 必ずコードを保存する
 - バックアップを取る
 - Gitなどを使うとなおよい

分析結果の再現性

- 原則：**分析結果の再現性**を担保する
 - **データ**の再現性：データ加工は統計ソフト上で
 - **分析**の再現性：ソースコードを残す
 - **解釈**の再現性：コード表を作る
- 再現性がないと結果の信頼性が損なわれる
- 誤り・改善点があっても発見できない

データの再現性

- データの再現：分析を再現するための最低限の要件
 - 元のデータが違えば（当然）結果は異なる
 - 分析に使ったデータ≠生データ
 - 分析対象にならないデータの除外
 - 逆転項目の処理 etc.
 - 生データからの加工過程がわかるようにする必要がある
- データ加工は統計ソフト上で
 - **絶対に**生データをいじらない！
 - 修正してよいのは誤入力など生データ自体に誤りがあったときのみ
 - 直接除外すると、除外したこと自体がわからない
 - なぜこう加工するのかを後から見た人でもわかるようソースコードに付記する

分析の再現性

- 分析の再現：データから結果を再現するために必要
 - 「この結果はどうやって出したんだろう？」
 - 「なぜこんな分析をしたんだろう？」
 - 他の人にも分析の過程がわかるようにソースコードを残す
- ソースコードを残す
 - 思考過程がたどれるようにする（分析過程だけでは不十分）
 - コードをわかりやすく整理する・説明する
 - マジックナンバーを使わない
ex. 「Q2の値が**2.57**以上か以下かで二群に分け...」
...その数字はどこから来たの？ 何の値か記述
 - わかりやすい名前をつける

解釈の再現性

- 解釈の再現：
 - 「数値は再現できた。でも、この変数って**どういう意味**なんだろう...？」
 - 「卒論ではこの項目分析されていないけど、**何を測っている**んだろう...？」
 - どういう意図で項目を入れたのか、どの論文から引用してきたのかが後から見てわかるようにする。
- コード表を残す
 - どのような項目で聞いているか
 - どのような変数か

★Jupyter Notebook/ Jupyter lab

- Jupyter Notebookとは
 - ノート形式でコードと結果を対話的に実行・保存できる環境
 - 人に説明したりするときによい
 - 実行できる言語：Python / R / Ruby / Lisp / Julia etc.
 - この講義の資料もJupyter Notebook形式で作っている
- Google Colaboratryというのものもある
 - Jupyter Notebookがベースになっている
 - GPUが無料で使えてしまう（時間制限あり）

文字コードについて

- 文字コード: コンピュータ上で文字を表現するためのデータ形式
 - 色々種類がある
 - Shift-JIS
 - UTF-8
 - 異なる文字コードで読み込むと「文字化け」することがある
- この講義では基本的にUTF-8というものを使う
- 文字コードの変換方法
 - テキストエディタ(e.g. メモ帳)で開き、「名前を付けて保存」時に文字コードを指定する

Rの基本操作 はじめに覚えておくこと

作業ディレクトリ (working directory)

- 作業ディレクトリ：ここを基準に相対パスが決まる
 - `getwd()`: 作業ディレクトリを確認する
 - `setwd()`: 作業ディレクトリを変更する

```
setwd(変更したいパス名)
```

コメントとヘルプ

- コメント
 - #より後の部分は実行されない
 - 後で他の人が読んでもわかるようにコードの説明を入れる
- ヘルプ
 - 関数名の前に?をつけて実行すると関数のヘルプを参照できる

変数の定義（代入）

- 変数を定義して値を使うことができる
 - 半角英数字・ピリオド(.)・アンダースコア(_)を使用できる
 - 先頭の文字は数字・記号以外
 - 予約されている語(e.g., if, for, function, TRUEなど)は変数名に使うことができない
 - Rは小文字と大文字は区別される
 - "TEST" と "test" は違う文字列

```
変数名 <- 値  
# または  
変数名 = 値
```

Rの基本操作 演算

算術演算

- 算術演算
 - 加算： $+$
 - 減算： $-$
 - 乗算： $*$
 - 除算： $/$
 - 累乗： $^$
- 整数除算
 - 整数除算の商： $\%/\%$
 - 整数除算の余り： $\%\%$

数学関数

- 総和: `sum()`
- 平均: `mean()`
- 平方根: `sqrt()`

- 分散: `var()`
 - 標本分散ではなく不偏分散が出力される
- 標準偏差 `std()`
 - 不偏分散を用いて計算される

様々なデータ型（値の種類）

- 整数型 **integer**
 - 整数値を扱う
- 実数型 **numeric**
 - 実数を扱う
- 文字列型 **character**
 - 文字列を扱う
 - 文字列は引用符(')または二重引用符(")で囲む
- 因子型 **factor**
 - カテゴリカルな変数を扱うのに便利な型

- 論理型 logical
 - 真偽値を扱う
 - TRUE: 真
 - FALSE: 偽
 - NA: 欠損値(**Not Available**)
 - TRUE, FALSEはそれぞれT, Fと簡略に記法できる
 - 型の変換
 - 論理値を数値に変換すると
 - TRUEは1
 - FALSEは0 として扱われる
 - 数値を論理型に変換すると
 - 0はFALSE
 - それ以外はTRUE として扱われる
 - NAと計算した結果はNAになる
 - NAにならない場合も例外的にある
 - NAを判定するにはis.na()を使う

比較演算子

- 比較演算子：値同士を比較して論理値を返す
 - 等号(=): $x == y$
 - 等しくない(\neq): $x != y$
 - 大なり(>): $x > y$
 - 小なり(<): $x < y$
 - 以上: $x \geq y$
 - 以下: $x \leq y$
- %in%演算子：要素として含まれるかを返す
 - x が y の要素に含まれる: $x \%in\% y$

論理演算：論理値同士の計算

- 論理積(and)
 - $x \& y$ (要素ごとに比較)
 - $x \&\& y$ (先頭だけ比較)
- 論理和(or)
 - $x | y$ (要素ごとに比較)
 - $x || y$ (先頭だけ比較)
- 否定(not)
 - $!x$

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

X	NOT X
1	0
0	1

データ構造：ベクトル

- ベクトル(vector)：複数の値を一次元的に並べたもの
 - 同じデータ型のみが含まれる
 - e.g. 数値型, 文字列型
 - ベクトルを使って様々な演算ができる
- 簡易的な作り方
 - 関数c()を使って要素を列挙する `c(要素1, 要素2, ...)`

簡易的なベクトルの作成

- コロン演算子(:)
 - 連番の数値ベクトルを作成
- seq()
 - 等差数列を作る
- rep()
 - ベクトルの繰り返しを作る

```
# mからnまでの連番  
m:n
```

```
seq(開始, 終了, 公差)
```

```
rep(ベクトル, 回数)
```

ベクトルの演算

- ベクトルと数値の演算
 - ベクトルのそれぞれの要素と数値との計算結果を返す
- ベクトル同士の演算
 - ベクトルの要素同士での計算結果を返す
- ベクトルと関数
 - 要素それぞれを対象にする関数: e.g., `sqrt()`
 - 要素全体を対象とする関数: e.g., `mean()`, `std()`
 - ベクトル自体を対象とする関数
 - `length()`: ベクトルの長さ
 - `rev()`: 要素を逆順にする
 - `sort()`: 要素をソートする

ベクトルの要素へのアクセス

- インデックス(添字番号)によるアクセス
 - 一致する位置にある要素にアクセス
 - 整数ベクトルを与えてもよい
- 論理ベクトルによるアクセス
 - TRUEであるもののみを抽出する
- 除外
 - マイナス(-)をつけてインデックスまたは整数ベクトルを与える
 - 条件を使う
 - 論理ベクトルで除外したいものの以外すべてを抽出する

その他の基本的なデータ構造

- 行列/配列
 - 行列：2次元に拡張されたベクトル
 - 配列：3次元以上に拡張されたベクトル
- リスト
 - 異なるデータ型を格納できる
 - リストにベクトルを格納したり
 - リストにリストを格納できる

Rの基本操作 関数と制御構造

関数

- 関数：様々な処理を行ったり値を返したりするもの
 - 引数(ひきすう, arguments)：関数を実行するときに渡す値
 - 返り値(かえりち, return value)：関数を実行した結果得られる値
 - 例：
 - 関数`sqrt()`は数値を引数として受け取り、平方根を返り値とする
 - 関数`print()`は値を表示する（返り値はない）
 - 関数の定義の仕方

```
関数名 = function(引数){  
    # 呼び出したときに実行される処理  
}
```

制御構造

- 繰り返し：同じ処理を繰り返し適用する
- 条件分岐：特定の場合に特定の処理を行う

if, else, else if

if

```
if(x){  
    # xが真の場合実行  
}
```

else if()

```
if(x){  
    # xが真の場合実行  
}else if(y){  
    # xが偽かつyが真の場合実行  
}
```


else:

```
if(x1){  
    # x1が真の場合実行  
}else if(x2){  
    # x2が真の場合実行  
}else{  
    # x1,x2のいずれも偽の場合実行  
}
```

for

for文の中で使う変数名

ここから一つずつ取り出して変数に代入していく



```
for(変数名 in シーケンス){  
    # 繰り返し処理  
}
```

for文の中でbreak()関数を実行すると処理をそこで中断してループの外に出る

★for vs. ベクトル

- for文は遅いので大量のデータを処理する目的にはそぐわない
 - ベクトル/行列の操作に最適化した関数を使うのがよい
 - e.g. 条件に応じて値を割り当てる：
 - `ifelse()`関数: 与えられた論理ベクトルによって異なる値を割り当てる

`ifelse`(論理ベクトル, TRUEのときに返す値, FALSEのときに返す値)

パッケージ

- 特定の目的のために開発された関数などの集まり
 - **psych** : 心理統計向けパッケージ
 - `describe()`: 要約統計量を出す
 - `corr.test()`: 複数の変数間の相関係数・偏相関係数・有意確率を出す
 - `fa()`: 因子分析
 - `fa.parallel()`: 平行分析
 - **tidyverse**: データ科学に便利なパッケージ集
 - **ggplot2**: 作図・可視化
 - **dplyr**: データ処理
 - **stringr**: 文字列処理
 - ...etc.



データハンドリングの基礎と作図

データハンドリングの基本操作

- 作成・変形
 - データを読み込む / データを生成する
 - 値を変える / 値を与える
 - ソートする
- 抽出
 - 特定のデータ(行・列)を抽出する
- 結合
 - 他のデータと結合する
- 集計
 - 集約した値を得る

データフレーム

- 列/行からなる表形式のデータ構造
 - 列ごとに同じデータ型のデータが入る
 - マージや結合ができる
 - 列名や行名が設定できる
 - `colnames()`: 列名へのアクセス
 - `rownames()`: 行名へのアクセス

		列(column)			
行 (row)		列 1	列 2	...	列n
	行 1				
	行 2				
	...				
	行m				

ひとつひとつの値が
入る箱：**セル**

データフレームの読み込み

- **csv / tsv**を読み込む
 - **csv**: カンマ区切り
 - 一般的なデータ形式
 - `read.csv()`で読み込む
 - **tsv**: タブ区切り
 - 言語データだとカンマがデータ内に含まれることがあるので**tsv**の方が取り回しがよいことがある
 - `read.delim()`で`sep='\\t'`のオプションを指定する
- データ量が多い場合は**data.table**パッケージの**fread()**関数を使うとよい
 - ただしデータフレームとは若干勝手が違う
- 読み込む前に気を付けること
 - 欠損値はどのように入力されているか？
 - ファイルの文字エンコーディング（文字コード）は何か？

データフレームの要素へのアクセス

- 複合的に指定可能
 - インデックス
 - 列名/行名
 - 論理ベクトル
- または
 - ドル記号(\$)を使ってアクセス
- e.g.

```
# 指定した列を返す  
df[指定列]  
# 指定したセルを返す  
df[指定行, 指定列]  
# 指定した行を返す  
df[指定行, ]  
# 指定した列を返す  
df[, 指定列]  
# 指定した列を返す  
df$変数名
```

「条件にあてはまる参加者の, 変数x,y,zの値」

列(column)

行
(row)

	列 1	列 2	...	列n
行 1				
行 2				
...				
行m				

列の抽出

行の抽出

--	--	--	--	--

セルの抽出

--

データフレームの抽出

変数（列）の追加

- 新しい変数名を用意して値を格納する

```
df['新しい変数名'] = 値またはベクトル  
df$新しい変数名 = 値またはベクトル
```

- 値を渡すとすべての行に同じ値が入る(cf. ベクトルの計算)
- データフレーム内に同じ名前の変数があると上書きされる

データフレームの結合

- 単純な結合
 - 列方向(column)の結合: `cbind()`
 - 行方向(row)の結合: `rbind()`
- `merge()`
 - キーとなる同じ列名を持つデータ同士を結合する
 - e.g. 成績データと健康データを「学生番号」を使って結合する

単純な結合

`cbind()`

	X	Y
a		
b		
c		

+

	Z
a	
b	
c	

⇒

	X	Y	Z
a			
b			
c			

`rbind()`

	X	Y
a		
b		
c		

+

	X	Y
d		
e		

⇒

	X	Y
a		
b		
c		
d		
e		

マージ

テーブルX

	ID	国語
a	1	X _a
b	2	X _b
c	3	X _c
d	4	X _d

テーブルY

	ID	数学
a	1	Y _a
b	2	Y _b
c	3	Y _c
f	6	Y _f

+

merge(X, Y)

指定なし

	ID	国語	数学
a	1	X _a	Y _a
b	2	X _b	Y _b
c	3	X _c	Y _c

テーブルX, Y両方にIDがある行
(Inner Join)

all.x=T

	ID	国語	数学
a	1	X _a	Y _a
b	2	X _b	Y _b
c	3	X _c	Y _c
d	4	X _d	<u>NA</u>

テーブルXにIDがある行
(Left Join)

all.y=T

	ID	国語	数学
a	1	X _a	Y _a
b	2	X _b	Y _b
c	3	X _c	Y _c
f	6	<u>NA</u>	Y _f

テーブルYにIDがある行
(Right Join)

all=T

	ID	国語	数学
a	1	X _a	Y _a
b	2	X _b	Y _b
c	3	X _c	Y _c
d	4	X _d	<u>NA</u>
f	6	<u>NA</u>	Y _f

テーブルX, YいずれかにIDがある
(Outer Join)

便利な関数

- `ncol()` / `nrow()`
 - 列/行の数(number)を返す
- `colMeans()` / `rowMeans()`
 - 列/行方向の平均(mean)を計算する
- `colSums()` / `rowSums()`
 - 列/行方向の和(sum)を計算する
- `head()` / `tail()`
 - データの先頭/末尾を抽出(デフォルト5件)
- `t()`
 - 転置(transform): 行と列を入れ替えたものを返す
- データのソート
 - `order()`: 与えたベクトルの順位 (順番)を返す
 - デフォルトで昇順、オプション `decreasing=T` で降順

```
# 列xの値で昇順でソート
df[order(df$x), ]
# 列xの値で降順にソート
df[order(df$x), ]
# 列x, yの値でソート
df[order(df$x, df$y), ]
```

データフレームの出力

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",  
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,  
            col.names = TRUE, qmethod = c("escape", "double"),  
            fileEncoding = "")
```

x: 保存したい表形式のデータ(e.g. データフレーム)

file: 保存するファイルのファイルパス

append: 追加で書き込むか(FALSEで上書き)

quote: 引用符で囲むかどうか

sep: 区切り文字(separator)

eol: 改行文字(end of line)

na: NAである行にいれる文字

dec: 小数点

row.names: 行の名前/行番号を書き込むか

col.names: 列の名前/変数名を書き込むか

fileEncoding: 保存する文字エンコーディング

作表と作図

- 作表関数
 - 分割表: `table()`
- 作図関数
 - 散布図
 - 棒グラフ: `barplot()`
 - 折れ線グラフ
 - 箱ひげ図: `boxplot()`
 - ヒストグラム: `hist()`
- `ggplot2`などの高度な作図ライブラリを使うとよりきれいな作図ができる

グラフの保存

- 出力したいファイル形式ごとに関数がある
 - pdf(), png(), jpeg(), bmp(), etc.
 - 保存用の関数を実行し、一連の作図を終えた後で dev.off() で閉じる

```
png('example.png', width=480, height=480) # 保存用の関数
# ここに作図関数を入れる
dev.off() # 終了して保存する
```

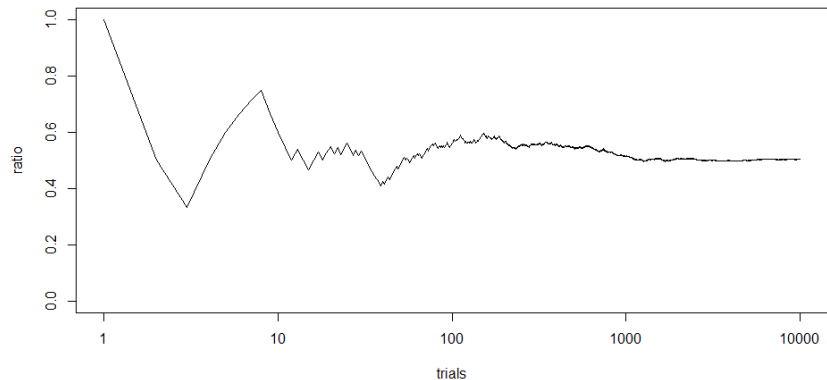
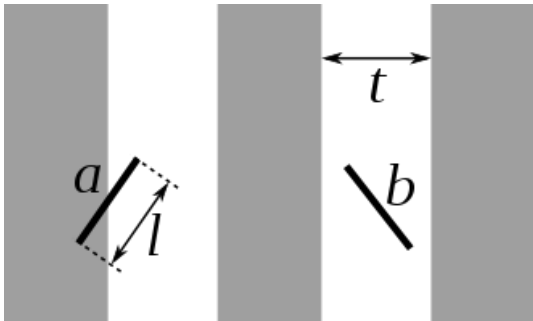
★Rと乱数

- 確率分布・乱数に関する関数
 - 確率密度関数 **d** + 確率分布名
 - **dnorm()**: 正規分布(normal distribution)の確率密度関数
 - 累積分布関数 **p** + 確率分布名
 - **pbinom()**: 二項分布(binomial distribution)の累積分布関数
 - 確率点 **q** + 確率分布名
 - **qpois()**: ポアソン分布(Poisson distribution)の確率点
 - 乱数 **r** + 確率分布名
 - **runif()**: 一様分布(unified distribution)にしたがう乱数
 - **sample()**関数
 - 与えたベクトルからランダムに取り出す
 - 復元抽出・非復元抽出いずれも可能

★モンテカルロ・シミュレーション

• モンテカルロ法

- 乱数を用いて近似解を求めるシミュレーション手法
 - 18C後半：「Buffonの針」...モンテカルロ法の起源
 - 多数の平行線を引いた床に針を落としたとき針と線が交差する確率はどれくらいか？
 - 1947: von NeumannとUlamのシミュレーション
 - 核分裂における中性子の拡散現象をモデル化して実験
 - 「モンテカルロ法 Monte Carlo method」と命名



一歩進んだデータハンドリング

dplyrによるデータハンドリング

- dplyr: データ操作を容易にするパッケージ
 - 整然データ (tidy data) を効率的に処理
 - 整然データでなくても使える
 - RStudioのページで公開されているチートシートが参考になる
 - <https://www.rstudio.com/resources/cheatsheets/>



- ★整然データ tidy data (Wickham, 2014)

- 各変数がそれぞれ列に
- 各観測(observation)がそれぞれ行に
- 各タイプの観測単位が1つの表に

まとまっているデータ

列にvariable

	身長	体重	血圧	測定日
Aさん				
Bさん				
Cさん				

行にobservation

tidy dataの例

- 整然ではないデータの例

列見出しが、値であって変数名でない

religion	<\$10k	\$10–20k	\$20–30k	\$30–40k	\$40–50k	\$50–75k
Agnostic	27	34	60	81	76	137
Atheist	12	27	37	52	35	70
Buddhist	27	21	30	34	33	58
Catholic	418	617	732	670	638	1116
Don't know/refused	15	14	15	11	10	35
Evangelical Prot	575	869	1064	982	881	1486
Hindu	1	9	7	9	11	34
Historically Black Prot	228	244	236	238	197	223
Jehovah's Witness	20	27	24	24	21	30
Jewish	19	19	25	25	30	95

Table 4: The first ten rows of data on income and religion from the Pew Forum. Three columns, \$75–100k, \$100–150k and >150k, have been omitted.

Wickham(2014)

ひとつの変数に複数の値

	友人
Aさん	Bさん, Dさん
Bさん	Aさん, Eさん
Cさん	Fさん

- パイプ演算子: %>%
 - magrittrというパッケージに含まれている
 - パイプ処理風に記述できる
 - 通常の処理：ひとつの処理が完全に終わったら次の処理、
というように逐次的に処理する
 - 処理n(処理n-1(...(処理2(処理1 ()))
 - パイプ処理：終わった部分から徐々に次の処理に流して
いき、並列的に処理する
 - 処理1 | 処理2 | ... | 処理n
 - データ分析のコードを可読性高く書ける

```
funcA(funcB(funcC()))  
# 以下のように書き換えられる  
funcC() %>% funcB() %>% funcA()
```

- filter(): 行の選択

```
# Andでつなげる場合
df %>% filter(条件1, 条件2, ...)
# または
df %>% filter(条件1 & 条件2 & ...)
# こう書いてもよい:
df %>% filter(条件1) %>% filter(条件2) %>% ...

# ORでつなげる場合
df %>% filter(条件1 | 条件2 | ...)
```

- select(): 列の選択

```
df %>% select(列1, 列2, ...)
# 新しい列名を指定できる
df %>% select(列1の新しい名前=列1, 列2の新しい名前=列2)
```

- summarise(): 集計
 - 関数を用いて集計できる
 - sum(), mean(), min(), max(), median(), etc.

```
df %>% summarise(表示名1 = 関数1(列名1), 表示名2 = 関数2(列名2), ...)
```

- group_by(): 変数によるグルーピング
 - グルーピングするとグループごとにsummarise()できる

```
# 変数の値によってグルーピングできる
```

```
df %>% group_by(集約に使う列1, 集約に使う列2, ...) %>% ...
```

- arrange(): ソート

```
df %>% arrange(列名1, 列名2, ...)
```

```
# 降順にするにはdesc()を使う
```

```
df %>% arrange(desc(列名1))
```


stringrを用いた文字列操作

- stringrパッケージ
 - Rで文字列を統一的に使いやすく処理するためのパッケージ
 - 正規表現が扱える



- 検索

- `str_subset()`: 指定されたキーワードが含まれている要素を返す
- `str_detect()`: 指定されたキーワードが含まれていればTRUE, そうでなければFALSEを返す

- 置換・分割

- `str_replace()`: 指定された検索語を置換語で置き換える
- `str_split()`: 指定された文字で文字列を分割する

- パターンマッチ

- `str_extract()`: マッチした最初の文字列を返す
- `str_extract_all()`: マッチしたすべての文字列を返す
- `str_count()`: マッチした個数を返す

正規表現

- 正規表現 regular expression: 文字列の集合を表現するための記法
- 文字列の検索や置換など様々な操作に使うことができる

- 基本

- ^ 文頭
- \$ 文末
- . 任意の1文字
- | 「または」

```
# 文頭のaにマッチ  
'^a'  
# 文末のaにマッチ  
'a$'  
# 文頭の2文字にマッチ  
'^..'  
# aまたはbにマッチ  
'a|b'
```

- 量指定子

- * 0回以上の繰り返し
- + 1回以上の繰り返し
- ? 0または1回
- {n} n回の繰り返し
- {n,} n回以上の繰り返し
- {n, m} n回以上m未満の繰り返し

```
# 任意の長さの文字列にマッチ  
'.*'  
# 1文字以上の連続するzにマッチ  
'z+'  
# linkまたはinkにマッチ  
'l?ink'  
# googleにマッチ  
'go{2}gle'  
# google, goooogle, goo...gleに  
マッチ  
'go{2,}gle'
```

- エスケープシーケンス

- `\d` 数字(digits)
- `\w` 単語に使われる文字・数字・アンダーバー(words)
- `\s` 空白(spaces)
- `\n` 改行(newline)
- `\t` タブ(tab)

1桁以上の数字にマッチ

`'\d+'`

1文字以上の文字・数字・アンダーバーにマッチ

`'\w+'`

1文字以上の長さの空白文字にマッチ

`'\s+'`

文末の改行にマッチ1文字以上の長さのタブにマッチ

`'\n$'`

1文字以上の長さのタブ文字にマッチ

`'\t+'`

- 文字クラスとグループ化
 - `[]` 括弧内に含まれる文字にマッチ
 - ハイフン(-)でつなぐと範囲を指定できる
 - `[a-e]` ...aからeまで
 - `[0-9]` ...0から9まで
 - `[^]` 括弧内に含まれない文字にマッチ
 - ハイフンで範囲指定可能
 - `()` グループ化

```
# 1文字以上の半角英小文字の連続にマッチ
'[a-z]+'
```

```
# 母音にマッチ
'[aiueo]'
```

```
# 数字以外の文字の連続にマッチ
'^[0-9]+'
```

```
# leftまたはrightにマッチ
'(left|right)'
```

- その他Rに用意されている文字クラス
 - [:alpha:] アルファベット
 - [:lower:] 小文字アルファベット
 - [:upper:] 大文字アルファベット
 - [:digit:] 数字
 - [:alnum:] アルファベット + 数字
 - [:punct:] 記号
 - [:graph:] 数字 + 記号 + アルファベット
 - [:blank:] 空白、タブ

本日のまとめ

- この講義で学んだこと
 - コンピュータの基本操作
 - Rの基本操作
 - データハンドリング
- 重要なこと
 - 自分で調べる力を身に着ける
 - コードを残す

References

- 舟尾暢男. (2009). *The R tips : データ解析環境Rの基本技・グラフィックス活用集 (第2版)*. オーム社.
- 川端一光, 岩間徳兼, & 鈴木雅之. (2018). *Rによる多変量解析入門: データ分析の実践と理論*. オーム社.
- 間瀬茂. (2007). *Rプログラミングマニュアル*. 数理工学社.
- Rサポーターズ. (2017). *パーフェクトR*. 技術評論社.
- 伊藤俊秀, & 草薙信照. (2006). *コンピュータシミュレーション*. オーム社.
- Wickham, H. (2014). Tidy Data. *Journal of Statistical Software*, 59(10), 1–23. <https://doi.org/10.18637/jss.v059.i10>
- Wickham, H., & Grolemund, G. (2016). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media.