@関西学院大学神学部
2020.12.28

講演会／ワークショップ
「テキストを対象とした認知人類学・宗教学研究」

テキスト分析のための R入門

明治大学　研究・知財戦略機構
佐藤浩輔

位置づけ

- 2020/12/28
  「テキスト分析のためのR入門」←この講義
- 2021/1/18
  「テキストを対象とした認知人類学・宗教研究の実例」*
  - 研究発表：民話の分析
  - 認知人類学・宗教研究におけるテキスト分析の動向

  「Rを用いたテキスト分析実習」
  - 計量テキスト分析/テキストマイニング入門
  - RとMeCabを用いた日本語テキスト解析実習

*中分遥先生（高知工科大学）担当

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
- サポートサイトについて

講義について

- この講義の目標：
  - Rを用いてテキスト分析を行うために**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
      - ファイル操作に関する基本的事項
    - ②Rの基本的な使い方
      - Rプログラミング初歩
      - データ操作入門
        - テキスト操作
        - データハンドリング
  - 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
  - 統計手法
  - 計算機科学的/数学的詳細

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<u>しません</u>
  - 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
      - 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：10:00~12:00
  - 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
    - 画面を凝視しすぎない
    - 休憩中に体を動かす

サポートサイトについて

- GitHub上の講義ページ
  - https://github.com/satocos135/seminar2020kwangaku
  - 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)
  - 受講者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

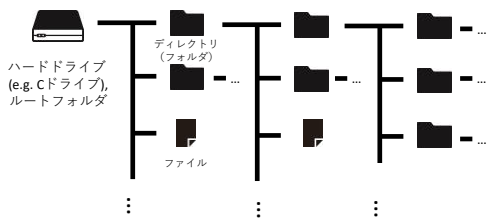
①コンピュータの使い方の初歩

ファイル操作に関する基本的事項

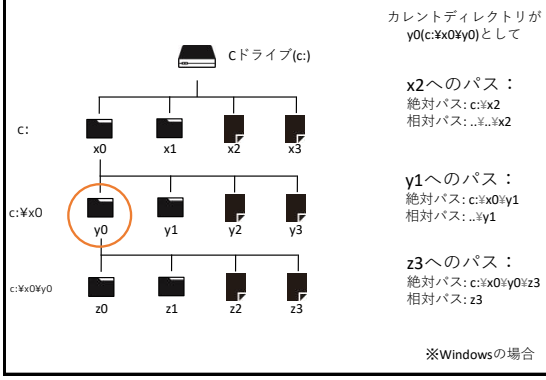
基本的な事柄

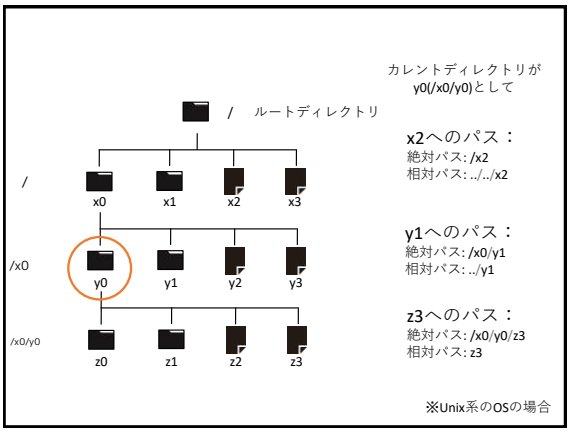
- ファイルとフォルダ
  - ファイル：データやプログラムを管理する単位
    - データファイル：データを格納するファイル
      - e.g., docx, txt
    - 実行ファイル：プログラムを実行するファイル
      - e.g. exe
  - フォルダ（ディレクトリ）：ファイルを整理する入れ物
    - 階層的に配置できる
- ファイル/フォルダの操作
  - コピー：複製する
  - 移動：違う場所へ移す
  - 削除：その場所から消す
  - リネーム：ファイルの名前を変える

ファイルシステム



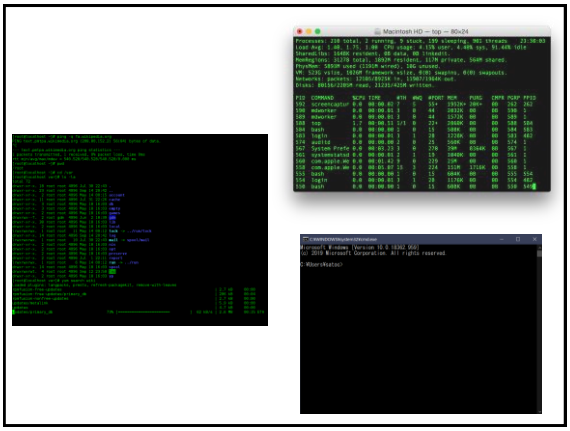
- パス（アドレス）：ファイルの場所を示す住所
  - ディレクトリ名を区切り文字でつなげる
    - 円記号(¥)またはスラッシュ(/), バックスラッシュ
  - 絶対パス：フルの住所
  - 相対パス：今いるディレクトリを基準にした住所
    - 今いるディレクトリ内のディレクトリ/ファイル：
      そのまま名前でアクセスできる
    - 今いるディレクトリ：　.
    - ひとつ上のディレクトリ：　..
    - ホームディレクトリ：~
    - ルートディレクトリ：　/





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**o**p**y)
    - exit: CLIを終了する



過つは人の業(errare humanum est)

- 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
- 初期はとにかく手を動かして慣れる
  - 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. 文字列操作の基礎
4. データハンドリングと作図
5. 一歩進んだデータハンドリング

Rとは

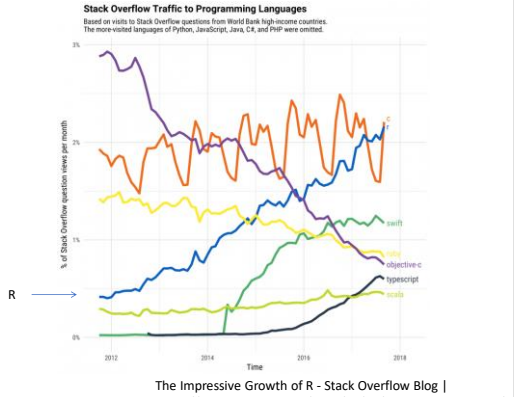
" R is a language and environment for statistical computing and graphics. "

「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」

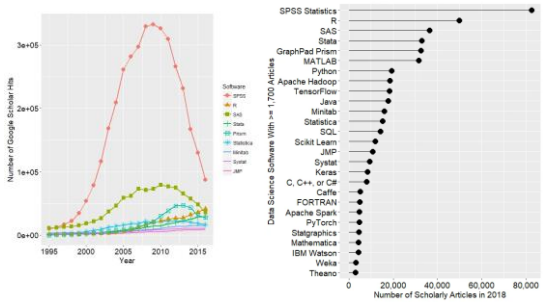
https://www.r-project.org/about.html

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



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



The Popularity of Data Science Software | r4stats.com | http://r4stats.com/articles/popularity/

Rの得手不得手

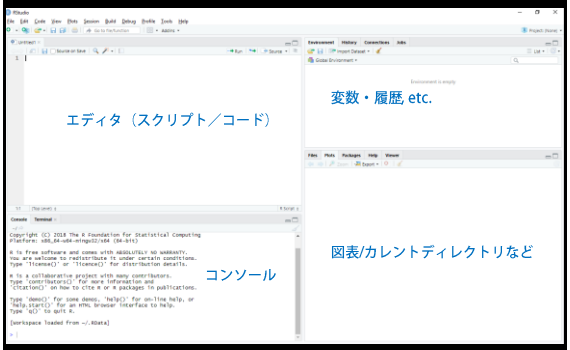
- 得意なこと
  - 統計計算
  - こまごまとしたデータ処理
- 苦手なこと(心理学で扱う程度のデータならあまり気にならないはず)
  - 速さが要求される計算
  - 大量のデータ(1GB~)の処理
    - 今は高速に扱えるパッケージもあるかも
- 科学技術計算の他の選択肢
  - Python
  - Julia

RStudio

- Rの統合開発環境 (**I**ntegrated **D**evelopment **E**nvironment: 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）



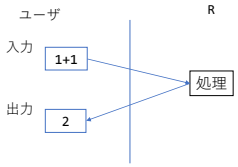
操作についてのアドバイス

- 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

Rの基本操作　はじめに覚えておくこと

基礎の基礎：コマンドの実行

- 基本的な操作の流れ
  - ユーザが命令を入力・実行
  - Rが命令を処理する
  - Rが結果を返す

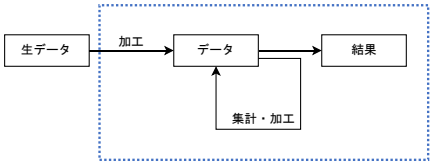


基礎の基礎：関数

- 関数：ひとまとりの命令を呼びだす手軽に呼びだす仕組み
  - 計算をする
  - ファイルやデータの読み書き
  - 図表や文字を表示する
  - etc.

基礎の基礎：分析のフロー

- データを加工して、分析のできる形に持っていく
- 生データから最終的な結果を再現できるよう、すべての処理は分析コード上で行う



このプロセスは分析コード上（今回はR）で行う

基礎の基礎：区別が必要なもの

- 関数や変数など（の名前）
  - 引用符なしの英数字で表現される
  - 関数を実行したり変数の中身を呼びだす
- 値
  - 数値
    - 数を表現するデータ(e.g. 整数・小数)
    - 引用符なしの数字で表現される
  - 文字列
    - 文字情報が格納されているデータ
    - 入力には引用符(')または二重引用符(")で囲む必要がある
    - 半角文字と全角文字は区別される
      - "text" ≠ "テキスト"
      - 全角スペースが紛れ込んでいるとエラーのもと
    - ひらがなとカタカナも区別される
      - "テキスト" ≠ "てきすと"
    - 英大文字と小文字も区別される
      - "TEXT" ≠ "text"
- 予約語
  - プログラムを実行するために用意されている単語群
  - if, forなど

```
# ヘルプを表示する
help()
# 円周率
pi
# 数字の1
1
# テキストとしての1
'1'
```

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
```

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　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()

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(**N**ot **A**vailable)
  - 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(≠): x != y
  - 大なり(>): x > y
  - 小なり(<): x < y
  - 以上: x >= y
  - 以下: x <= 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

```
# mからnまでの連番
m:n
```

- seq()
  - 等差数列を作る

```
seq(開始, 終了, 公差)
```

- rep()
  - ベクトルの繰り返しを作る

```
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

- プログラム：たくさんの要素を扱う
  - ある処理をそれぞれの要素に行いたい
  - 特定の場合に処理を場合分けしたい
    ⇒制御構造と呼ばれるものを使う
- 基本的な制御構造
  - 繰り返し：同じ処理を繰り返し適用する
    - 要素ひとつずつに処理を行うことができる
  - 条件分岐：特定の場合に特定の処理を行う
    - 場合分けができる

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

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
}
```

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
  - 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.

パッケージのインストール

- パッケージのインストール
  - install.packages('パッケージ名')
- パッケージの読み込み
  - library('パッケージ名')

パイプ演算子の導入

- パイプ演算子: %>%
  - magrittrというパッケージに含まれている
    - dplyrを読み込むと同時に読み込まれる
  - 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
    - 入れ子状にならないので、データ分析のコードを可読性高く書ける

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

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
  - tidyverseに含まれている
  - 正規表現が扱える
  - 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/

- 検索
  - str_subset() : 指定されたキーワードを含む要素を返す
  - 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
- これを使えるようになるのがテキスト分析の勘所なので、練習推奨

正規表現があると何が嬉しいか

- 検索の自由度が上がる
  - 異なるつづりに対応できる
    - colou?r →"color" と "colour" 両方にマッチ
  - 複数のパターンを同時に検索できる
    - (暑|寒)い →「暑い」「寒い」両方にマッチ
  - 特定の性質を持つ文字（文字クラス）をまとめて扱える
    - ¥¥d人 →「0人」,「1人」,…「9人」にマッチ
    - ¥¥s →スペース・タブなどの空白文字にマッチ

→文字列を自在に扱えるようになる

- 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, gooogle, 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

データハンドリングの基礎

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？

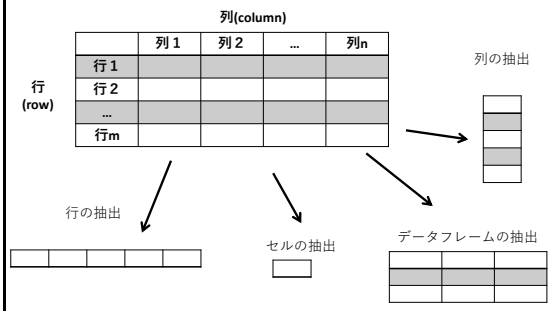
データフレームの要素へのアクセス

- 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.

「居住地が北海道である回答者の年齢」

行を指定する条件　　列を指定

```
# 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
```



変数（列）の追加

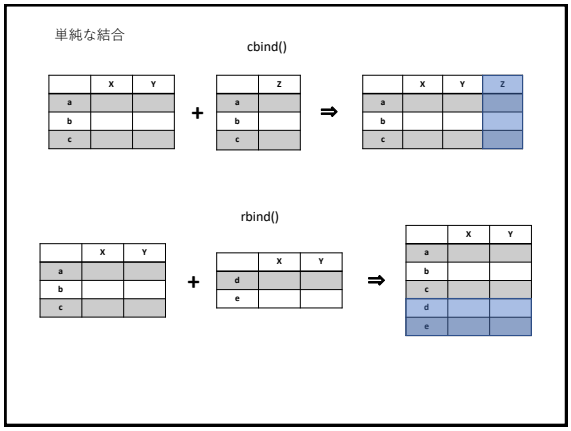
- 新しい変数名を用意して値を格納する

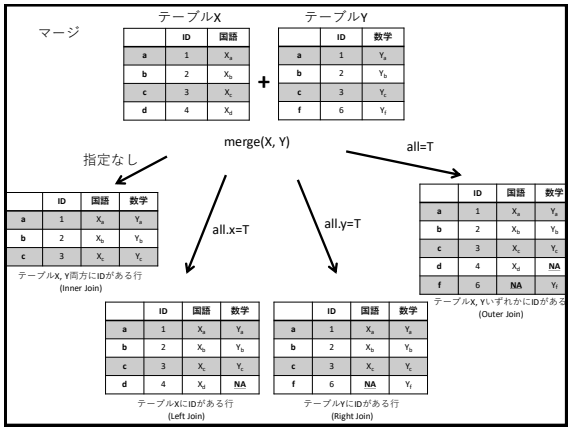
```
df['新しい変数名'] = 値またはベクトル
df$新しい変数名 = 値またはベクトル
```

  - 値を渡すとすべての行に同じ値が入る
  - ベクトルを渡すと位置に応じた値が格納される
  - 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





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
```

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

パイプ演算子の導入（再掲）

- パイプ演算子: %>%
  - magrittrというパッケージに含まれている
    - dplyrを読み込むと同時に読み込まれる
  - 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
    - 入れ子状にならないので、データ分析のコードを可読性高く書ける

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

- 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))
```

本日のまとめ

- 学んだこと
  - ファイル操作の基本
  - Rの基本操作
    - 文字列操作の基本
    - データハンドリングの基本

  →Rの操作のエッセンスを学んだ

  ⇒テキスト分析に必要なピースはおおむね揃った

- 1/18にやること
  - 今日学んだ様々な要素を用いて、実例を通して具体的な分析をどうやっていくかを学ぶ

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.
- 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.