# Simulation

Roger D. Peng, Associate Professor of Biostatistics
Johns Hopkins Bloomberg School of Public Health

# Generating Random Numbers

Functions for probability distributions in R

- `rnorm`: generate random Normal variates with a given mean and standard deviation
- `dnorm`: evaluate the Normal probability density (with a given mean/SD) at a point (or vector of points)
- `pnorm`: evaluate the cumulative distribution function for a Normal distribution
- `rpois`: generate random Poisson variates with a given rate

# Generating Random Numbers

Probability distribution functions usually have four functions associated with them. The functions are prefixed with a

- `d` for density
- `r` for random number generation
- `p` for cumulative distribution
- `q` for quantile function

# Generating Random Numbers

Working with the Normal distributions requires using these four functions

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

If $\Phi$ is the cumulative distribution function for a standard Normal distribution, then `pnorm(q)` = $\Phi(q)$ and `qnorm(p)` = $\Phi^{-1}(p)$.

# Generating Random Numbers

```
> x <- rnorm(10)
> x
 [1] 1.38380206 0.48772671 0.53403109 0.66721944
 [5] 0.01585029 0.37945986 1.31096736 0.55330472
 [9] 1.22090852 0.45236742
> x <- rnorm(10, 20, 2)
> x
 [1] 23.38812 20.16846 21.87999 20.73813 19.59020
 [6] 18.73439 18.31721 22.51748 20.36966 21.04371
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  18.32   19.73   20.55   20.67   21.67   23.39
```

# Generating Random Numbers

Setting the random number seed with `set.seed` ensures reproducibility

```
> set.seed(1)
> rnorm(5)
[1] -0.6264538  0.1836433 -0.8356286  1.5952808
[5]  0.3295078
> rnorm(5)
[1] -0.8204684  0.4874291  0.7383247  0.5757814
[5] -0.3053884
> set.seed(1)
> rnorm(5)
[1] -0.6264538  0.1836433 -0.8356286  1.5952808
[5]  0.3295078
```

Always set the random number seed when conducting a simulation!

# Generating Random Numbers

Generating Poisson data

```
> rpois(10, 1)
 [1] 3 1 0 1 0 0 1 0 1 1
> rpois(10, 2)
 [1] 6 2 2 1 3 2 2 1 1 2
> rpois(10, 20)
 [1] 20 11 21 20 20 21 17 15 24 20

> ppois(2, 2)  ## Cumulative distribution
[1] 0.6766764  ## Pr(x <= 2)
> ppois(4, 2)
[1] 0.947347   ## Pr(x <= 4)
> ppois(6, 2)
[1] 0.9954662  ## Pr(x <= 6)
```

# Generating Random Numbers From a Linear Model

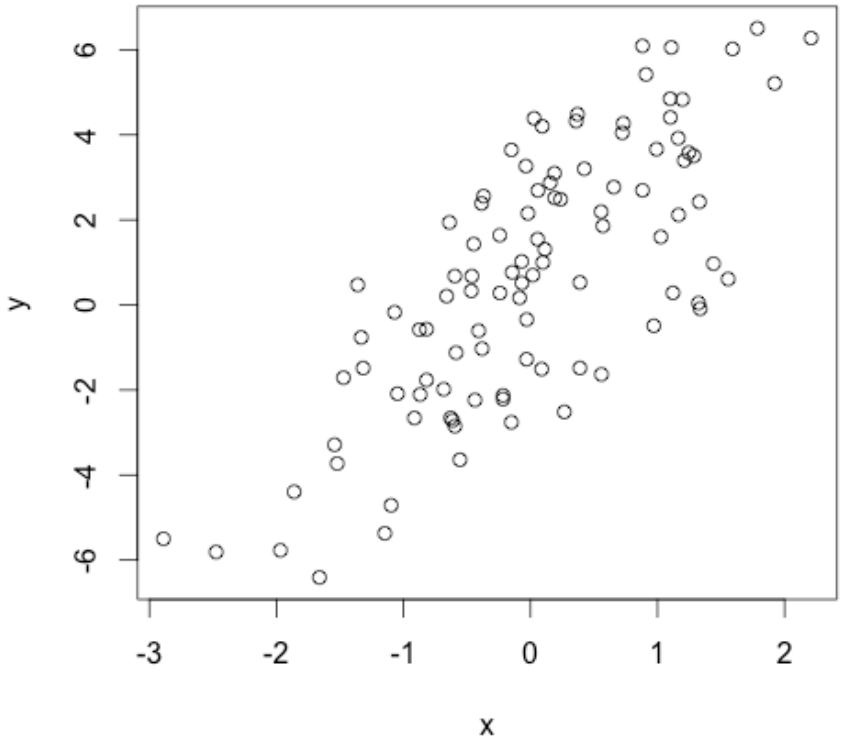
Suppose we want to simulate from the following linear model

$$y = \beta_0 + \beta_1 x + \varepsilon$$

where $\varepsilon \sim \mathcal{N}(0, 2^2)$. Assume $x \sim \mathcal{N}(0, 1^2)$, $\beta_0 = 0.5$ and $\beta_1 = 2$.

```
> set.seed(20)
> x <- rnorm(100)
> e <- rnorm(100, 0, 2)
> y <- 0.5 + 2 * x + e
> summary(y)
    Min.  1st Qu.   Median
-6.4080 -1.5400  0.6789  0.6893  2.9300  6.5050
> plot(x, y)
```

# Generating Random Numbers From a Linear Model

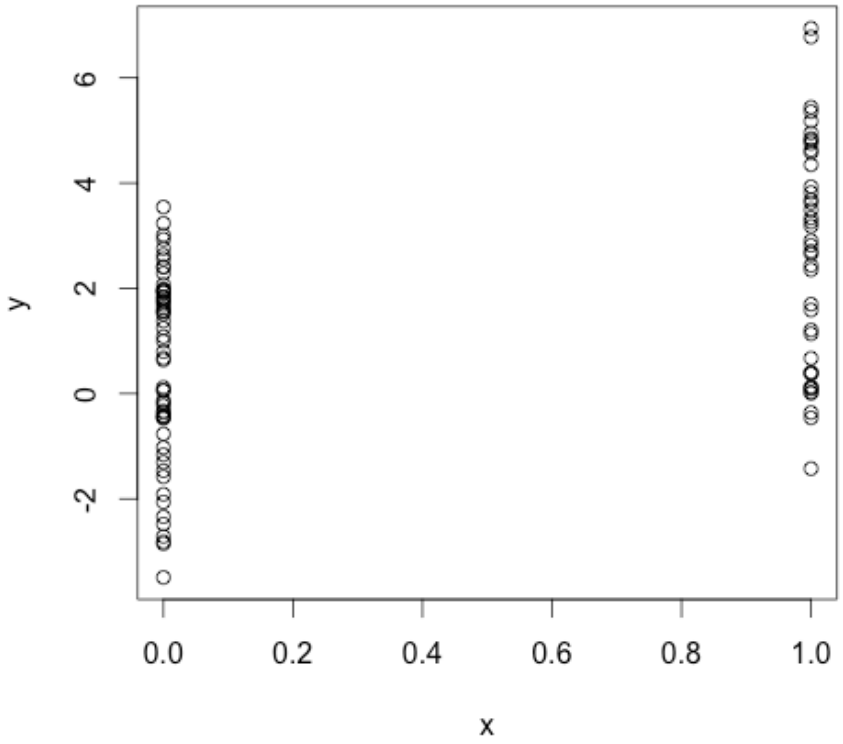


# Generating Random Numbers From a Linear Model

What if `x` is binary?

```
> set.seed(10)
> x <- rbinom(100, 1, 0.5)
> e <- rnorm(100, 0, 2)
> y <- 0.5 + 2 * x + e
> summary(y)
   Min. 1st Qu.  Median
-3.4940 -0.1409  1.5770  1.4320  2.8400  6.9410
> plot(x, y)
```

# Generating Random Numbers From a Linear Model



# Generating Random Numbers From a Generalized Linear Model

Suppose we want to simulate from a Poisson model where

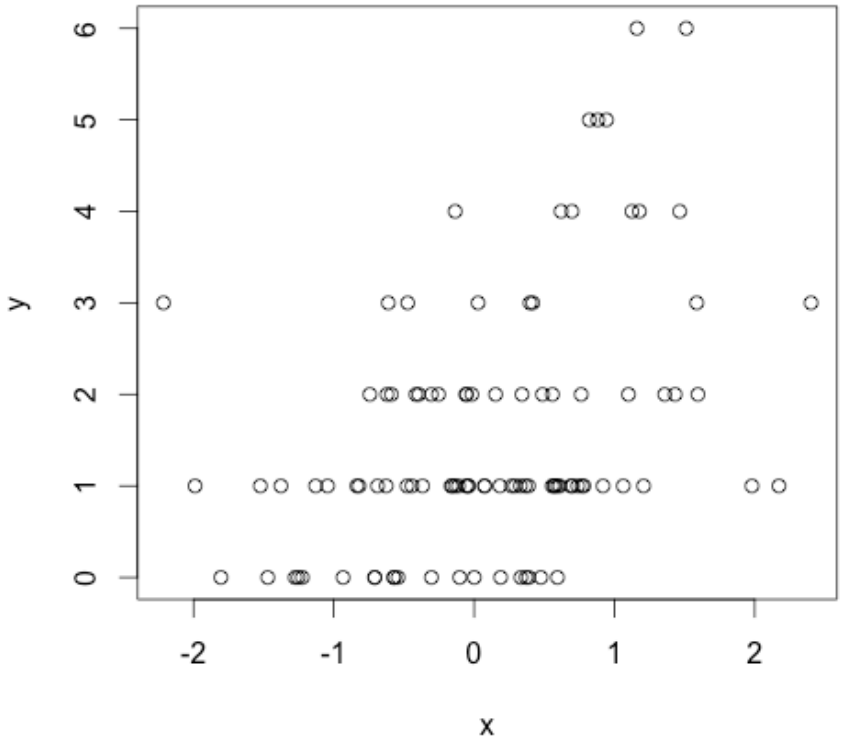
$Y \sim \text{Poisson}(\mu)$

$\log \mu = \beta_0 + \beta_1 x$

and $\beta_0 = 0.5$ and $\beta_1 = 0.3$. We need to use the `rpois` function for this

```
> set.seed(1)
> x <- rnorm(100)
> log.mu <- 0.5 + 0.3 * x
> y <- rpois(100, exp(log.mu))
> summary(y)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    1.00    1.00    1.55    2.00    6.00
> plot(x, y)
```

# Generating Random Numbers From a Generalized Linear Model



# Random Sampling

The `sample` function draws randomly from a specified set of (scalar) objects allowing you to sample from arbitrary distributions.

```
> set.seed(1)
> sample(1:10, 4)
[1] 3 4 5 7
> sample(1:10, 4)
[1] 3 9 8 5
> sample(letters, 5)
[1] "q" "b" "e" "x" "p"
> sample(1:10)  ## permutation
 [1] 4 710 6 9 2 8 3 1 5
> sample(1:10)
 [1]  2  3  4  1  9  5 10  8  6  7
> sample(1:10, replace = TRUE)  ## Sample w/replacement
 [1] 2 9 7 8 2 8 5 9 7 8
```

# Simulation

Summary

- Drawing samples from specific probability distributions can be done with `r*` functions
- Standard distributions are built in: Normal, Poisson, Binomial, Exponential, Gamma, etc.
- The `sample` function can be used to draw random samples from arbitrary vectors
- Setting the random number generator seed via set.seed is critical for reproducibility

# Profiling R Code

Roger D. Peng, Associate Professor of Biostatistics
Johns Hopkins Bloomberg School of Public Health

# Why is My Code So Slow?

- Profiling is a systematic way to examine how much time is spend in different parts of a program
- Useful when trying to optimize your code
- Often code runs fine once, but what if you have to put it in a loop for 1,000 iterations? Is it still fast enough?
- Profiling is better than guessing

# On Optimizing Your Code

- Getting biggest impact on speeding up code depends on knowing where the code spends most of its time

- This cannot be done without performance analysis or profiling

We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil
--Donald Knuth

# General Principles of Optimization

- Design first, then optimize
- Remember: Premature optimization is the root of all evil
- Measure (collect data), don’t guess.
- If you're going to be scientist, you need to apply the same principles here!

# Using `system.time()`

- Takes an arbitrary R expression as input (can be wrapped in curly braces) and returns the amount of time taken to evaluate the expression
- Computes the time (in seconds) needed to execute an expression
  - If there's an error, gives time until the error occurred
- Returns an object of class `proc_time`
  - **user time**: time charged to the CPU(s) for this expression
  - **elapsed time**: "wall clock" time

# Using `system.time()`

- Usually, the user time and elapsed time are relatively close, for straight computing tasks
- Elapsed time may be *greater than* user time if the CPU spends a lot of time waiting around
- Elapsted time may be *smaller than* the user time if your machine has multiple cores/processors (and is capable of using them)
    - Multi-threaded BLAS libraries (vecLib/Accelerate, ATLAS, ACML, MKL)
    - Parallel processing via the **parallel** package

# Using `system.time()`

```
## Elapsed time > user time
system.time(readLines("http://www.jhsph.edu"))
   user  system elapsed
  0.004   0.002   0.431

## Elapsed time < user time
hilbert <- function(n) {
        i <- 1:n
        1 / outer(i - 1, i, "+")
}
x <- hilbert(1000)
system.time(svd(x))
   user  system elapsed
  1.605   0.094   0.742
```

# Timing Longer Expressions

```
system.time({
    n <- 1000
    r <- numeric(n)
    for (i in 1:n) {
        x <- rnorm(n)
        r[i] <- mean(x)
    }
})
```

```
##    user  system elapsed
##   0.097   0.002   0.099
```

# Beyond `system.time()`

- Using `system.time()` allows you to test certain functions or code blocks to see if they are taking excessive amounts of time
- Assumes you already know where the problem is and can call `system.time()` on it
- What if you don’t know where to start?

# The R Profiler

- The `Rprof()` function starts the profiler in R
  - R must be compiled with profiler support (but this is usually the case)
- The `summaryRprof()` function summarizes the output from `Rprof()` (otherwise it's not readable)
- DO NOT use `system.time()` and `Rprof()` together or you will be sad

# The R Profiler

- Rprof() keeps track of the function call stack at regularly sampled intervals and tabulates how much time is spend in each function
- Default sampling interval is 0.02 seconds
- NOTE: If your code runs very quickly, the profiler is not useful, but then you probably don't need it in that case

# R Profiler Raw Output

```
## lm(y ~ x)

sample.interval=10000
"list" "eval" "eval" "model.frame.default" "model.frame" "eval" "eval" "lm"
"list" "eval" "eval" "model.frame.default" "model.frame" "eval" "eval" "lm"
"list" "eval" "eval" "model.frame.default" "model.frame" "eval" "eval" "lm"
"list" "eval" "eval" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"na.omit" "model.frame.default" "model.frame" "eval" "eval" "lm"
"lm.fit" "lm"
"lm.fit" "lm"
"lm.fit" "lm"
```

# Using `summaryRprof()`

- The `summaryRprof()` function tabulates the R profiler output and calculates how much time is spend in which function
- There are two methods for normalizing the data
- "by.total" divides the time spend in each function by the total run time
- "by.self" does the same but first subtracts out time spent in functions above in the call stack

# By Total

```
$by.total
                        total.time total.pct self.time self.pct
"lm"                          7.41    100.00      0.30     4.05
"lm.fit"                      3.50     47.23      2.99    40.35
"model.frame.default"         2.24     30.23      0.12     1.62
"eval"                        2.24     30.23      0.00     0.00
"model.frame"                 2.24     30.23      0.00     0.00
"na.omit"                     1.54     20.78      0.24     3.24
"na.omit.data.frame"          1.30     17.54      0.49     6.61
"lapply"                      1.04     14.04      0.00     0.00
"[.data.frame"                1.03     13.90      0.79    10.66
"["                           1.03     13.90      0.00     0.00
"as.list.data.frame"          0.82     11.07      0.82    11.07
"as.list"                     0.82     11.07      0.00     0.00
```

# By Self

```
$by.self
                        self.time self.pct total.time total.pct
"lm.fit"                     2.99    40.35       3.50     47.23
"as.list.data.frame"         0.82    11.07       0.82     11.07
"[.data.frame"               0.79    10.66       1.03     13.90
"structure"                  0.73     9.85       0.73      9.85
"na.omit.data.frame"         0.49     6.61       1.30     17.54
"list"                       0.46     6.21       0.46      6.21
"lm"                         0.30     4.05       7.41    100.00
"model.matrix.default"       0.27     3.64       0.79     10.66
"na.omit"                    0.24     3.24       1.54     20.78
"as.character"               0.18     2.43       0.18      2.43
"model.frame.default"        0.12     1.62       2.24     30.23
"anyDuplicated.default"      0.02     0.27       0.02      0.27
```

# `summaryRprof()` Output

```
$sample.interval
[1] 0.02

$sampling.time
[1] 7.41
```

# Summary

- `Rprof()` runs the profiler for performance of analysis of R code
- `summaryRprof()` summarizes the output of `Rprof()` and gives percent of time spent in each function (with two types of normalization)
- Good to break your code into functions so that the profiler can give useful information about where time is being spent
- C or Fortran code is not profiled

# Programming Assignment 3

R Programming

## Introduction

Download the file `ProgAssignment3-data.zip` file containing the data for Programming Assignment 3 from the Coursera web site. Unzip the file in a directory that will serve as your working directory. When you start up R make sure to change your working directory to the directory where you unzipped the data.

The data for this assignment come from the Hospital Compare web site (`http://hospitalcompare.hhs.gov`) run by the U.S. Department of Health and Human Services. The purpose of the web site is to provide data and information about the quality of care at over 4,000 Medicare-certified hospitals in the U.S. This dataset essentially covers all major U.S. hospitals. This dataset is used for a variety of purposes, including determining whether hospitals should be fined for not providing high quality care to patients (see `http://goo.gl/jAXFX` for some background on this particular topic).

The Hospital Compare web site contains a lot of data and we will only look at a small subset for this assignment. The zip file for this assignment contains three files

- `outcome-of-care-measures.csv`: Contains information about 30-day mortality and readmission rates for heart attacks, heart failure, and pneumonia for over 4,000 hospitals.
- `hospital-data.csv`: Contains information about each hospital.
- `Hospital_Revised_Flatfiles.pdf`: Descriptions of the variables in each file (i.e the code book).

A description of the variables in each of the files is in the included PDF file named `Hospital_Revised_Flatfiles.pdf`. This document contains information about many other files that are not included with this programming assignment. You will want to focus on the variables for Number 19 ("Outcome of Care Measures.csv") and Number 11 ("Hospital_Data.csv"). You may find it useful to print out this document (at least the pages for Tables 19 and 11) to have next to you while you work on this assignment. In particular, the numbers of the variables for each table indicate column indices in each table (i.e. "Hospital Name" is column 2 in the `outcome-of-care-measures`.csv file).

## 1 Plot the 30-day mortality rates for heart attack

Read the outcome data into R via the `read.csv` function and look at the first few rows.

```
> outcome <- read.csv("outcome-of-care-measures.csv", colClasses = "character")
> head(outcome)
```

There are many columns in this dataset. You can see how many by typing `ncol(outcome)` (you can see the number of rows with the `nrow` function). In addition, you can see the names of each column by typing `names(outcome)` (the names are also in the PDF document.

To make a simple histogram of the 30-day death rates from heart attack (column 11 in the outcome dataset), run

```
> outcome[, 11] <- as.numeric(outcome[, 11])
> ## You may get a warning about NAs being introduced; that is okay
> hist(outcome[, 11])
```

Because we originally read the data in as character (by specifying `colClasses = "character"` we need to coerce the column to be numeric. You may get a warning about NAs being introduced but that is okay.

## 2 Finding the best hospital in a state

Write a function called `best` that take two arguments: the 2-character abbreviated name of a state and an outcome name. The function reads the `outcome-of-care-measures.csv` file and returns a character vector with the name of the hospital that has the best (i.e. lowest) 30-day mortality for the specified outcome in that state. The hospital name is the name provided in the `Hospital.Name` variable. The outcomes can be one of "heart attack", "heart failure", or "pneumonia". Hospitals that do not have data on a particular outcome should be excluded from the set of hospitals when deciding the rankings.

**Handling ties**. If there is a tie for the best hospital for a given outcome, then the hospital names should be sorted in alphabetical order and the first hospital in that set should be chosen (i.e. if hospitals "b", "c", and "f" are tied for best, then hospital "b" should be returned).

The function should use the following template.

```
best <- function(state, outcome) {
        ## Read outcome data

        ## Check that state and outcome are valid

        ## Return hospital name in that state with lowest 30-day death
        ## rate
}
```

The function should check the validity of its arguments. If an invalid `state` value is passed to `best`, the function should throw an error via the `stop` function with the exact message "invalid state". If an invalid `outcome` value is passed to `best`, the function should throw an error via the `stop` function with the exact message "invalid outcome".

Here is some sample output from the function.

```
> source("best.R")
> best("TX", "heart attack")
[1] "CYPRESS FAIRBANKS MEDICAL CENTER"
> best("TX", "heart failure")
[1] "FORT DUNCAN MEDICAL CENTER"
> best("MD", "heart attack")
[1] "JOHNS HOPKINS HOSPITAL, THE"
> best("MD", "pneumonia")
[1] "GREATER BALTIMORE MEDICAL CENTER"
> best("BB", "heart attack")
Error in best("BB", "heart attack") : invalid state
> best("NY", "hert attack")
Error in best("NY", "hert attack") : invalid outcome
>
```

Save your code for this function to a file named `best.R`.

# 3 Ranking hospitals by outcome in a state

Write a function called `rankhospital` that takes three arguments: the 2-character abbreviated name of a state (`state`), an outcome (`outcome`), and the ranking of a hospital in that state for that outcome (`num`). The function reads the `outcome-of-care-measures.csv` file and returns a character vector with the name of the hospital that has the ranking specified by the `num` argument. For example, the call

```
rankhospital("MD", "heart failure", 5)
```

would return a character vector containing the name of the hospital with the 5th lowest 30-day death rate for heart failure. The `num` argument can take values "best", "worst", or an integer indicating the ranking (smaller numbers are better). If the number given by `num` is larger than the number of hospitals in that state, then the function should return `NA`. Hospitals that do not have data on a particular outcome should be excluded from the set of hospitals when deciding the rankings.

**Handling ties**. It may occur that multiple hospitals have the same 30-day mortality rate for a given cause of death. In those cases ties should be broken by using the hospital name. For example, in Texas ("TX"), the hospitals with lowest 30-day mortality rate for heart failure are shown here.

```
> head(texas)
```

```
                      Hospital.Name Rate Rank
3935       FORT DUNCAN MEDICAL CENTER  8.1    1
4085  TOMBALL REGIONAL MEDICAL CENTER  8.5    2
4103 CYPRESS FAIRBANKS MEDICAL CENTER  8.7    3
3954           DETAR HOSPITAL NAVARRO  8.7    4
4010           METHODIST HOSPITAL,THE  8.8    5
3962  MISSION REGIONAL MEDICAL CENTER  8.8    6
```

Note that Cypress Fairbanks Medical Center and Detar Hospital Navarro both have the same 30-day rate (8.7). However, because Cypress comes before Detar alphabetically, Cypress is ranked number 3 in this scheme and Detar is ranked number 4. One can use the `order` function to sort multiple vectors in this manner (i.e. where one vector is used to break ties in another vector).

The function should use the following template.

```
rankhospital <- function(state, outcome, num = "best") {
        ## Read outcome data

        ## Check that state and outcome are valid

        ## Return hospital name in that state with the given rank
        ## 30-day death rate
}
```

The function should check the validity of its arguments. If an invalid `state` value is passed to `rankhospital`, the function should throw an error via the `stop` function with the exact message "invalid state". If an invalid `outcome` value is passed to `rankhospital`, the function should throw an error via the `stop` function with the exact message "invalid outcome".

Here is some sample output from the function.

```
> source("rankhospital.R")
> rankhospital("TX", "heart failure", 4)
```

```
[1] "DETAR HOSPITAL NAVARRO"
```

```
> rankhospital("MD", "heart attack", "worst")
```

```
[1] "HARFORD MEMORIAL HOSPITAL"
```

```
> rankhospital("MN", "heart attack", 5000)
```

```
[1] NA
```

Save your code for this function to a file named `rankhospital.R`.

# 4 Ranking hospitals in all states

Write a function called `rankall` that takes two arguments: an outcome name (`outcome`) and a hospital ranking (`num`). The function reads the `outcome-of-care-measures.csv` file and returns a 2-column data frame containing the hospital in each state that has the ranking specified in `num`. For example the function call `rankall("heart attack", "best")` would return a data frame containing the names of the hospitals that are the best in their respective states for 30-day heart attack death rates. The function should return a value for every state (some may be `NA`). The first column in the data frame is named `hospital`, which contains the hospital name, and the second column is named `state`, which contains the 2-character abbreviation for the state name. Hospitals that do not have data on a particular outcome should be excluded from the set of hospitals when deciding the rankings.

**Handling ties**. The `rankall` function should handle ties in the 30-day mortality rates in the same way that the `rankhospital` function handles ties.

The function should use the following template.

```
rankall <- function(outcome, num = "best") {
        ## Read outcome data

        ## Check that state and outcome are valid

        ## For each state, find the hospital of the given rank

        ## Return a data frame with the hospital names and the
        ## (abbreviated) state name
}
```

**NOTE**: For the purpose of this part of the assignment (and for efficiency), your function should NOT call the `rankhospital` function from the previous section.

The function should check the validity of its arguments. If an invalid `outcome` value is passed to `rankall`, the function should throw an error via the `stop` function with the exact message "invalid outcome". The `num` variable can take values "best", "worst", or an integer indicating the ranking (smaller numbers are better). If the number given by `num` is larger than the number of hospitals in that state, then the function should return `NA`.

Here is some sample output from the function.

```
> source("rankall.R")
> head(rankall("heart attack", 20), 10)
```

```
                              hospital state
AK                                <NA>    AK
AL      D W MCMILLAN MEMORIAL HOSPITAL    AL
AR   ARKANSAS METHODIST MEDICAL CENTER    AR
```

```
AZ JOHN C LINCOLN DEER VALLEY HOSPITAL    AZ
CA               SHERMAN OAKS HOSPITAL    CA
CO            SKY RIDGE MEDICAL CENTER    CO
CT             MIDSTATE MEDICAL CENTER    CT
DC                                <NA>    DC
DE                                <NA>    DE
FL      SOUTH FLORIDA BAPTIST HOSPITAL    FL
```

```
> tail(rankall("pneumonia", "worst"), 3)
```

```
                                     hospital state
WI MAYO CLINIC HEALTH SYSTEM - NORTHLAND, INC    WI
WV                     PLATEAU MEDICAL CENTER    WV
WY           NORTH BIG HORN HOSPITAL DISTRICT    WY
```

```
> tail(rankall("heart failure"), 10)
```

```
                                                          hospital state
TN                          WELLMONT HAWKINS COUNTY MEMORIAL HOSPITAL    TN
TX                                       FORT DUNCAN MEDICAL CENTER    TX
UT VA SALT LAKE CITY HEALTHCARE - GEORGE E. WAHLEN VA MEDICAL CENTER    UT
VA                                         SENTARA POTOMAC HOSPITAL    VA
VI                            GOV JUAN F LUIS HOSPITAL & MEDICAL CTR    VI
VT                                           SPRINGFIELD HOSPITAL    VT
WA                                        HARBORVIEW MEDICAL CENTER    WA
WI                                      AURORA ST LUKES MEDICAL CENTER    WI
WV                                           FAIRMONT GENERAL HOSPITAL    WV
WY                                         CHEYENNE VA MEDICAL CENTER    WY
```

Save your code for this function to a file named `rankall.R`.