Types of data in R

- numeric
- character
- logical
- factor

Types of R objects

- vectors
- matrices
- data frames
- lists

Lists in R

- lists are R objects that contain other named R objects
- often used to return results of functions when different parts of the results are of different types and lengths

Example: Using the `glm` function for generalized linear models

```r
> help(glm)
```

```
glm package:stats R Documentation

Fitting Generalized Linear Models

Description:
'glm' is used to fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution.

Usage:

glm(formula, family = gaussian, data, weights, subset, na.action, start = NULL, etastart, mustart, offset, control = glm.control(...), model = TRUE, method = "glm.fit", x = FALSE, y = TRUE, contrasts = NULL, ...)
```
Arguments:

- **formula**: a symbolic description of the model to be fit. The details of model specification are given below.
- **family**: a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions.)
- **data**: an optional data frame, list or environment (or object coercible by `as.data.frame`) to a data frame containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `glm` is called.
- **weights**: an optional vector of weights to be used in the fitting process. Should be `NULL` or a numeric vector.
- **method**: the method to be used in fitting the model. The default method `"glm.fit"` uses iteratively reweighted least squares (IWLS). The only current alternative is `"model.frame"` which returns the model frame and does no fitting.
- **intercept**: logical. Should an intercept be included in the _null_ model?
- **...**: further arguments passed to or from other methods.

Value:

- `glm` returns an object of class inheriting from `"glm"` which inherits from the class `"lm"`. See later in this section.
- The function `summary` (i.e., `summary.glm`) can be used to obtain or print a summary of the results and the function `anova` (i.e., `anova.glm`) to produce an analysis of variance table.
- The generic accessor functions `coefficients`, `effects`, `fitted.values` and `residuals` can be used to extract various useful features of the value returned by `glm`.
- `weights` extracts a vector of weights, one for each case in the fit (after subsetting and `na.action`).

An object of class `"glm"` is a list containing at least the following components:

- **coefficients**: a named vector of coefficients
- **residuals**: the _working_ residuals, that is the residuals in the final iteration of the IWLS fit. Since cases with zero weights are omitted, their working residuals are `NA`.
- **fitted.values**: the fitted mean values, obtained by transforming the linear predictors by the inverse of the link function.
- **rank**: the numeric rank of the fitted linear model.
- **family**: the `family` object used.
- **linear.predictors**: the linear fit on link scale.
- **deviance**: up to a constant, minus twice the maximized log-likelihood. Where sensible, the constant is chosen so that a saturated model has deviance zero.
- **aic**: Akaike’s _An Information Criterion_, minus twice the maximized log-likelihood plus twice the number of coefficients (so assuming that the dispersion is known).
- **null.deviance**: The deviance for the null model, comparable with `deviance`. The null model will include the offset, and an intercept if there is one in the model.
- **iter**: the number of iterations of IWLS used.
- **weights**: the _working_ weights, that is the weights in the final iteration of the IWLS fit.
- **prior.weights**: the case weights initially supplied.
- **df.residual**: the residual degrees of freedom.
- **df.null**: the residual degrees of freedom for the null model.
- **y**: the `y` vector used. (It is a vector even for a binomial model.)
- **converged**: logical. Was the IWLS algorithm judged to have converged?
- **boundary**: logical. Is the fitted value on the boundary of the attainable values?
- **call**: the matched call.
A worked example

Create the data frame for analysis

```r
> clotting <- data.frame(  
+   u = c(5,10,15,20,30,40,60,80,100),  
+   lot1 = c(118,58,42,35,27,25,21,19,18),  
+   lot2 = c(69,35,26,21,18,16,13,12,12))
```

Assign output of `glm` function to an object

```r
> clot.glm <- glm(lot1 ~ log(u), data=clotting, family=Gamma)
> names(clot.glm)
[1] "coefficients" "residuals" "fitted.values"  
[4] "effects" "qr" "family" "linear.predictors"  
[7] "deviance" "aic" "null.deviance"  
[10] "iter" "weights" "prior.weights"  
[13] "df.residual" "df.null" "y"  
[16] "deviance" "boundary" "model"  
[19] "call" "formula" "terms"  
[22] "data" "offset" "control"  
[25] "method" "contrasts" "levels"  
```

Apply generic R function to the object

```r
> summary(clot.glm)
Call:
glm(formula = lot1 ~ log(u), family = Gamma, data = clotting)
Deviance Residuals:
Min 1Q Median 3Q Max
-0.04008 -0.03756 -0.02637 0.02905 0.08641

Coefficients:
            Estimate Std. Error t value Pr(>|t|)  
(Intercept) -0.0165544  0.0009275   -17.85 4.28e-07 ***  
log(u)       0.0153431  0.0004150    36.98 2.75e-09 ***  

(Dispersion parameter for Gamma family taken to be 0.002446013)

Null deviance: 3.512826 on 8 degrees of freedom
Residual deviance: 0.016730 on 7 degrees of freedom
AIC: 37.99
Number of Fisher Scoring iterations: 3
```

Extract items from the list object

```r
> clot.glm$coefficients
(Intercept)  log(u)
  -0.01655438  0.01534311

> coefficients(clot.glm)
(Intercept)  log(u)
  -0.01655438  0.01534311

> clot.glm[[1]]
(Intercept)  log(u)
  -0.01655438  0.01534311

> clot.glm$residuals
1 2 3 4 5
3.219040e-04 -1.669367e-03 -1.245089e-03 -8.626301e-04 1.353040e-03
6 7 8 9
-4.456895e-05 1.314957e-03 1.879617e-03 1.414312e-03

> clot.glm$call
glm(formula = lot1 ~ log(u), family = Gamma, data = clotting)
```

Writing your own R functions

- **structure of a function**

  ```r
  function( <arguments>)
  {
  <body of function>
  <object to return>
  }
  ```
example: function to solve quadratic equations

```r
quadratic <- function( a, b, c ) {
  discrim <- b * b - 4 * a * c
  if( discrim < 0 ) {
    solution <- c(NA,NA)
    errflag <- 1
  } else {
    if( a ) # if a is not 0
      mult <- c(1,-1)
      solution <- ( -b + mult * sqrt( discrim ) ) / (2 * a)
      errflag <- 0
    else
      solution <- c(NA,NA)
      errflag <- 2
  }
  list( solution = solution, errflag = errflag)
}
```

This program is saved in a file called “quadratic.R.” Using `source` to execute this file in R will put the function in R’s memory.

```r
source("quadratic.R")
```

Looping in R

```r
for (i in 1:10) {
}

for (j in c(1,37, 81) {
  print( j/2 )
}
```

Timing R calculations

- `system.time` function with a line of R code as its argument
- third number returned is the elapsed time
- in example below, execution of the loop took .004 seconds

```r
> ans <- 0
> system.time( for(i in 1:5000) ans <- i + ans )
user  system elapsed
0.003 0.000 0.004
```