NC STATE UNIVERSITY

Streamlining R Code

Justin Post

What do we want to be able to do?

The course provides a brief overview of R data structures followed by the following topics:

- Loops in R
- Vectorized functions (apply family of functions)
- How R functions work
- Function writing

First up, recap and streamline repeated sections of code!

Review of Data Structures in R

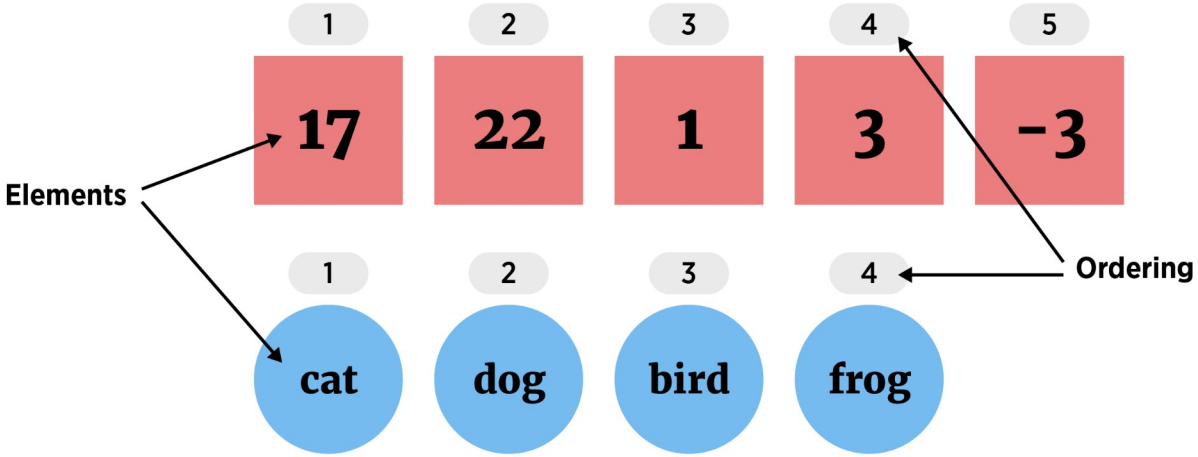
- Five major types

    1. Atomic Vector (1d)
    2. Matrix (2d)
    3. Array (nd)
    4. Data Frame (2d)
    5. List (1d)

Dimension	Homogeneous	Heterogeneous
1d	Atomic Vector	List
2d	Matrix	Data Frame

Vector

Atomic Vector (1D group of elements with an ordering)



- Elements must be same ‘type’
    - numeric (integer or double), character, or logical

Accessing Parts of an Atomic Vector (1D)

- Return elements using square brackets `[]`
- Can 'feed' in a vector of indices to `[]`

```
letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
## [20] "t" "u" "v" "w" "x" "y" "z"
```

```
letters[1:4]
```

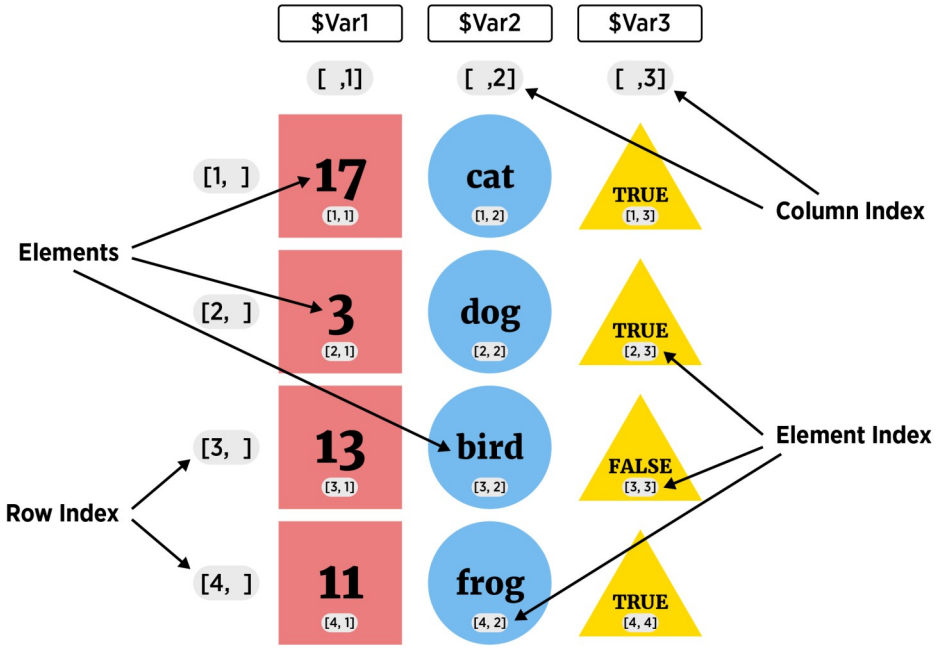
```
## [1] "a" "b" "c" "d"
```

```
x <- c(1, 2, 5); letters[x]
```

```
## [1] "a" "b" "e"
```

Data Frames

- Best R object for data sets
- Collection (list) of vectors of the same **length**



Data Frames

- Consider the built in `iris` data set
- Can see info about object with `str()`

```
myIris <- as_tibble(iris)
str(myIris)
```

```
## tibble [150 x 5] (S3: tbl_df/tbl/data.frame)
##  $ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Accessing Parts of Data Frame (2D)

- Returning rectangular portions of a data frame

```
myIris[1:4, 2:4]
```

```
## # A tibble: 4 x 3
##   Sepal.Width Petal.Length Petal.Width
##         <dbl>        <dbl>       <dbl>
## 1         3.5          1.4         0.2
## 2         3            1.4         0.2
## 3         3.2          1.3         0.2
## 4         3.1          1.5         0.2
```

```
myIris[1, ]
```

```
## # A tibble: 1 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <fct>
## 1          5.1         3.5          1.4         0.2 setosa
```

Accessing Parts of Data Frame (2D)

- Grabbing a column (same result for both)

```
myIris$Sepal.Length
```

```
dplyr::pull(myIris, Sepal.Length)
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

Data Frames

- Grabbing multiple columns

```
dplyr::select(myIris, starts_with("Sepal"))
```

```
## # A tibble: 150 x 2
##   Sepal.Length Sepal.Width
##          <dbl>       <dbl>
## 1          5.1         3.5
## 2          4.9         3
## 3          4.7         3.2
## 4          4.6         3.1
## 5          5           3.6
## # ... with 145 more rows
```

Packages & `tidyverse`

- "TidyVerse" - collection of R packages that share common philosophies and are designed to work together!
- If not installed (downloaded) on computer

```
install.packages("tidyverse")
```

- Once installed, `library()` or `require()` to load

```
library(tidyverse)
```

- `dplry` package made for most standard data manipulation tasks
- `tidyr` handles most of the rest
- `%>%` operator allows coding from left to right

Pipe

- Generically, pipe does the following

`x %>% f(y)` turns into `f(x,y)`

`x %>% f(y) %>% g(z)` turns into `g(f(x, y), z)`

```
library(Lahman) #Install pacakage if needed
Batting %>%
  as_tibble() %>%
  select(starts_with("X"), ends_with("ID"), G) %>%
  rename("Doubles" = X2B, "Triples" = X3B)
```

```
## # A tibble: 105,861 x 7
##   Doubles Triples playerID  yearID teamID lgID      G
##     <int>   <int> <chr>      <int> <fct>  <fct> <int>
## 1       0       0 abercda01   1871 TRO    NA        1
## 2       6       0 addybo01    1871 RC1    NA       25
## 3       4       5 allisar01   1871 CL1    NA       29
## 4      10       2 allisdo01   1871 WS3    NA       27
## 5      11       3 ansonca01   1871 RC1    NA       25
## # ... with 105,856 more rows
```

Streamlining Code

On to the main attraction! Improving R code!

- Often a repetitive task must be done
- Task requires a small change each time it is done

Example:

- Summarize each column of a dataset

Summarizing columns

Consider wine data from UCI machine learning repository

```
wineData <- read_csv("../datasets/winequality-full.csv")
wineData
```

```
## # A tibble: 6,497 x 13
##    `fixed acidity` `volatile acidity` `citric acid` `residual sugar` chlorides
##              <dbl>              <dbl>         <dbl>            <dbl>     <dbl>
## 1            7.4                0.7           0                1.9     0.076
## 2            7.8                0.88          0                2.6     0.098
## 3            7.8                0.76          0.04             2.3     0.092
## 4           11.2                0.28          0.56             1.9     0.075
## 5            7.4                0.7           0                1.9     0.076
## # ... with 6,492 more rows, and 8 more variables: free sulfur dioxide <dbl>,
## #   total sulfur dioxide <dbl>, density <dbl>, pH <dbl>, sulphates <dbl>,
## #   alcohol <dbl>, quality <dbl>, type <chr>
```

Summarizing Columns

- Of course R has some nice functions to help us out…

```
summary(wineData)
```

```
##  fixed acidity    volatile acidity  citric acid     residual sugar
##  Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600
##  1st Qu.: 6.400   1st Qu.:0.2300   1st Qu.:0.2500   1st Qu.: 1.800
##  Median : 7.000   Median :0.2900   Median :0.3100   Median : 3.000
##  Mean   : 7.215   Mean   :0.3397   Mean   :0.3186   Mean   : 5.443
##  3rd Qu.: 7.700   3rd Qu.:0.4000   3rd Qu.:0.3900   3rd Qu.: 8.100
##  Max.   :15.900   Max.   :1.5800   Max.   :1.6600   Max.   :65.800
##    chlorides       free sulfur dioxide total sulfur dioxide    density
##  Min.   :0.00900   Min.   :  1.00      Min.   :  6.0        Min.   :0.9871
##  1st Qu.:0.03800   1st Qu.: 17.00      1st Qu.: 77.0        1st Qu.:0.9923
##  Median :0.04700   Median : 29.00      Median :118.0        Median :0.9949
##  Mean   :0.05603   Mean   : 30.53      Mean   :115.7        Mean   :0.9947
##  3rd Qu.:0.06500   3rd Qu.: 41.00      3rd Qu.:156.0        3rd Qu.:0.9970
##  Max.   :0.61100   Max.   :289.00      Max.   :440.0        Max.   :1.0390
##        pH          sulphates         alcohol         quality
##  Min.   :2.720   Min.   :0.2200   Min.   : 8.00   Min.   :3.000
##  1st Qu.:3.110   1st Qu.:0.4300   1st Qu.: 9.50   1st Qu.:5.000
##  Median :3.210   Median :0.5100   Median :10.30   Median :6.000
##  Mean   :3.219   Mean   :0.5313   Mean   :10.49   Mean   :5.818
##  3rd Qu.:3.320   3rd Qu.:0.6000   3rd Qu.:11.30   3rd Qu.:6.000
```

Custom Column Summary - Hard Way

- Print out mean, median, and trimmed mean for each (numeric) column

```
#fixed acidity
c(Mean = mean(wineData$`fixed acidity`),
  Median = median(wineData$`fixed acidity`),
  TrimmedMean = mean(wineData$`fixed acidity`, 0.05))
```

```
##        Mean      Median TrimmedMean
##    7.215307    7.000000    7.104796
```

```
#volatile acidity
c(Mean = mean(wineData$`volatile acidity`),
  Median = median(wineData$`volatile acidity`),
  TrimmedMean = mean(wineData$`volatile acidity`, 0.05))
```

```
##        Mean      Median TrimmedMean
##   0.3396660   0.2900000   0.3255864
```

```
#...
```

Custom Column Summary

Instead use a **Loop**!

- `for loops` or `while loops` commonly used in R
- `for loop` syntax

```
for(index in values){
  code to be run
}
```

For Loops - Index

- index defines ‘counter’ or variable that varies

```
for (index in 1:10){
  print(index)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

```
for (i in c("cat", "dog", "wolf")){
  print(i)
}
```

```
## [1] "cat"
## [1] "dog"
## [1] "wolf"
```

For Loops - Values

- 'values' define which values index takes on

```
values <- 1:10
for (index in values){
  print(index)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

```
for (i in seq_along(iris)){
  print(names(iris)[i])
}
```

```
## [1] "Sepal.Length"
## [1] "Sepal.Width"
## [1] "Petal.Length"
## [1] "Petal.Width"
## [1] "Species"
```

Custom Column Summary

- Print out mean, median, and trimmed mean for each (numeric) column

```
for(i in 1:12){ #first 12 columns are numeric
  colData <- pull(wineData, i)
  print(names(wineData)[i])
  print(c(Mean = mean(colData),
          Median = median(colData),
          TrimmedMean = mean(colData, 0.05))
  )
}
```

```
## [1] "fixed acidity"
##        Mean      Median TrimmedMean
##    7.215307    7.000000    7.104796
## [1] "volatile acidity"
##        Mean      Median TrimmedMean
##   0.3396660   0.2900000   0.3255864
## [1] "citric acid"
##        Mean      Median TrimmedMean
##   0.3186332   0.3100000   0.3160780
## [1] "residual sugar"
##        Mean      Median TrimmedMean
##    5.443235    3.000000    5.027039
## [1] "chlorides"
```

Custom Column Summary

- Might store the results for later use

```
ncols <- ncol(wineData)
sumDF <- data.frame(varName = names(wineData)[-ncols],
                    mean = numeric(ncols-1),
                    median = numeric(ncols-1),
                    trimmedMean = numeric(ncols-1)
                    )
sumDF
```

```
##                varName mean median trimmedMean
## 1        fixed acidity    0      0           0
## 2     volatile acidity    0      0           0
## 3          citric acid    0      0           0
## 4       residual sugar    0      0           0
## 5            chlorides    0      0           0
## 6  free sulfur dioxide    0      0           0
## 7 total sulfur dioxide    0      0           0
## 8              density    0      0           0
## 9                   pH    0      0           0
## 10           sulphates    0      0           0
## 11             alcohol    0      0           0
## 12             quality    0      0           0
```

Custom Column Summary

- Fill in each row as you go

```
for(i in seq_along(wineData)[-ncols]){
  colData <- pull(wineData, i)
  sumDF[i, 2:4] <- c(mean(colData), median(colData), mean(colData, 0.05))
}
sumDF
```

```
##                varName         mean    median  trimmedMean
## 1        fixed acidity   7.21530706   7.00000   7.10479569
## 2     volatile acidity   0.33966600   0.29000   0.32558643
## 3          citric acid   0.31863322   0.31000   0.31607796
## 4       residual sugar   5.44323534   3.00000   5.02703881
## 5            chlorides   0.05603386   0.04700   0.05197538
## 6  free sulfur dioxide  30.52531938  29.00000  29.64472559
## 7 total sulfur dioxide 115.74457442 118.00000 115.26927680
## 8              density   0.99469663   0.99489   0.99468259
## 9                   pH   3.21850085   3.21000   3.21485040
## 10           sulphates   0.53126828   0.51000   0.52081894
## 11             alcohol  10.49180083  10.30000  10.44339944
## 12             quality   5.81837771   6.00000   5.81073688
```

`while` Loops

- `while` loop similar to `for` loops

```
while(condition) {
    expression to evaluate
  modify condition to FALSE?
}
```

`break` Out of a Loop

- `break` exits a loop

```
for (i in 1:5){
    if (i == 4){
      break
      }
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
```

`next` to Skip

- `next` jumps to the next iteration of the loop

```
for (i in 1:5){
    if (i == 3){
      next
    }
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 4
## [1] 5
```

Quick Examples

- Go to the course files page and try Exercise 1 - Loops

Looping in R

For loops inefficient in R

- R interpreted language
- Must figure out how to evaluate code at each iteration of loop
- Slows it down

Looping in R

For loops inefficient in R

- R interpreted language
- Must figure out how to evaluate code at each iteration of loop
- Slows it down

Vectorized functions much faster!

- Vectorized function: works on entire vector at once
- Avoids costly computation time

Efficient Code

Some ‘built-in’ vectorized functions

- `colMeans()`, `rowMeans()`
- `colSums()`, `rowSums()`
- `colSds()`, `colVars()`, `colMedians()` (`matrixStats` package)
- `ifelse()`, `dplyr::if_else()`
- `apply()` family
- Create your own with `Vectorize()`

`colMeans` - Find Column Means

- `colMeans()` just requires a numeric data frame (array)

```
wineData %>%
  select(-type) %>%
  colMeans()
```

```
##        fixed acidity     volatile acidity          citric acid
##           7.21530706           0.33966600           0.31863322
##       residual sugar            chlorides  free sulfur dioxide
##           5.44323534           0.05603386          30.52531938
## total sulfur dioxide              density                   pH
##         115.74457442           0.99469663           3.21850085
##             sulphates              alcohol              quality
##           0.53126828          10.49180083           5.81837771
```

Compare computational time

- `microbenchmark` package allows for easy recording of computing time

```
install.packages("microbenchmarK")
```

```
library(microbenchmark)
```

Compare computational time

```
wineData2 <- wineData %>% select(-type)
microbenchmark(colMeans(wineData2), unit = "ms")
```

```
## Unit: milliseconds
##                  expr    min      lq     mean median      uq    max neval
##  colMeans(wineData2) 0.2975 0.30905 0.416344 0.3237 0.34855 5.8771   100
```

```
microbenchmark(for(i in 1:12){mean(wineData[[i]])}, unit = "ms")
```

```
## Unit: milliseconds
##                                         expr    min      lq     mean median
##  for (i in 1:12) {     mean(wineData[[i]]) } 1.5886 1.67655 1.961633 1.7918
##      uq    max neval
##  2.0541 4.5707   100
```

`colMedians` - column medians

- `matrixStats::colMedians()` just requires a numeric data frame (array)

```
library(matrixStats)
wineData %>%
  select(-type) %>%
  as.matrix() %>%
  colMedians()
```

```
##  [1]   7.00000   0.29000   0.31000   3.00000   0.04700  29.00000 118.00000
##  [8]   0.99489   3.21000   0.51000  10.30000   6.00000
```

Creating a New Variable

Want to code a new categorical quality variable

- Poor (quality ≤ 3)
- Ok ($3 <$ quality ≤ 5)
- Good ($5 <$ quality ≤ 7)
- Great ($7 <$ quality ≤ 10)
- Initial plan: loop through each observation and use `if` `then` `else`

Inefficient Code

```
#initialize vector to save results
qualityCat <- character()

for (i in 1:(dim(wineData)[1])){
  if(wineData$quality[i] <= 3){
    qualityCat[i] <- "Poor"
  } else if(wineData$quality[i] <= 5){
    qualityCat[i] <- "Ok"
  } else if(wineData$quality[i] <= 7){
    qualityCat[i] <- "Good"
  } else if(wineData$quality[i] <= 10){
    qualityCat[i] <- "Great"
  } else {
    qualityCat[i] <- "Error"
  }
}
```

Inefficient Code

```
wineData$qualityCat <- qualityCat
wineData %>% select(qualityCat, quality, everything())
```

```
## # A tibble: 6,497 x 14
##   qualityCat quality `fixed acidity` `volatile acidity` `citric acid`
##   <chr>        <dbl>           <dbl>              <dbl>         <dbl>
## 1 Ok               5             7.4               0.7           0
## 2 Ok               5             7.8               0.88          0
## 3 Ok               5             7.8               0.76          0.04
## 4 Good             6            11.2               0.28          0.56
## 5 Ok               5             7.4               0.7           0
## # ... with 6,492 more rows, and 9 more variables: residual sugar <dbl>,
## #   chlorides <dbl>, free sulfur dioxide <dbl>, total sulfur dioxide <dbl>,
## #   density <dbl>, pH <dbl>, sulphates <dbl>, alcohol <dbl>, type <chr>
```

Efficiency

- Know for loops not great
- `if_else()` (or `ifelse()`) is vectorized version of `if then else`
- Syntax

```
if_else(vector_condition, if_true_do_this, if_false_do_this)
```

Back to the Code

- Know for loops not great
- `if_else()` (or `ifelse()`) is vectorized version of `if then else`
- Syntax

```
if_else(vector_condition, if_true_do_this, if_false_do_this)
```

```
qualityCat <- if_else(wineData$quality <= 3, "Poor",
                  if_else(wineData$quality <= 5, "Ok",
                        if_else(wineData$quality <= 7, "Good",
                            if_else(wineData$quality <= 10, "Great", "Error"))))
```

Compare speed

```
loopTime<-microbenchmark(
  for (i in 1:(dim(wineData)[1])){
    if(wineData$quality[i] <= 3){
      qualityCat[i] <- "Poor"
    } else if(wineData$quality[i] <= 5){
      qualityCat[i] <- "Ok"
    } else if(wineData$quality[i] <= 7){
      qualityCat[i] <- "Good"
    } else if(wineData$quality[i] <= 10){
      qualityCat[i] <- "Great"
    } else {
      qualityCat[i] <- "Error"
    }
  }
, unit = "us")
```

Compare speed

```
vectorTime <- microbenchmark(
    if_else(wineData$quality <= 3, "Poor",
        if_else(wineData$quality <= 5, "Ok",
            if_else(wineData$quality <= 7, "Good",
                if_else(wineData$quality <= 10, "Great", "Error"))))
, unit = "us")
```

Efficient Code

```
loopTime
```

```
## Unit: microseconds
##
##  for (i in 1:(dim(wineData)[1])) {     if (wineData$quality[i] <= 3) {         qualityCat[i]
##      min      lq    mean  median    uq     max neval
##  31811.9 35022.7 36533.4 36424.6 38195 42296.4   100
```

```
vectorTime
```

```
## Unit: microseconds
##
##  if_else(wineData$quality <= 3, "Poor", if_else(wineData$quality <=      5, "Ok", if_else(wi
##    min     lq     mean median     uq    max neval
##  762.3 802.15 2110.173 841.05 951.15 110538   100
```

`dplyr` to Summarize Data

- `group_by()` and `summarize()` great for quick summaries
- Find mean alcohol for each quality category

```
wineData %>%
  group_by(qualityCat, type) %>%
  summarize(meanAlcohol = mean(alcohol))
```

```
## `summarise()` has grouped output by 'qualityCat'. You can override using the `.groups` argum
```

```
## # A tibble: 8 x 3
## # Groups:   qualityCat [4]
##   qualityCat type  meanAlcohol
##   <chr>      <chr>       <dbl>
## 1 Good       Red          10.8
## 2 Good       White        10.8
## 3 Great      Red          12.1
## 4 Great      White        11.7
## 5 Ok         Red           9.93
## 6 Ok         White         9.84
## 7 Poor       Red           9.96
## 8 Poor       White        10.3
```

`dplyr` to Summarize Data

- `group_by()` and `mutate()` provide a nice way to add to a dataframe

```
wineData %>%
  group_by(qualityCat, type) %>%
  mutate(meanAlcoholCat = mean(alcohol)) %>%
  select(meanAlcoholCat, qualityCat, type, alcohol, everything())
```

```
## # A tibble: 6,497 x 15
## # Groups:   qualityCat, type [8]
##   meanAlcoholCat qualityCat type  alcohol `fixed acidity` `volatile acidity`
##            <dbl> <chr>      <chr>   <dbl>           <dbl>              <dbl>
## 1           9.93 Ok         Red       9.4             7.4               0.7
## 2           9.93 Ok         Red       9.8             7.8               0.88
## 3           9.93 Ok         Red       9.8             7.8               0.76
## 4          10.8  Good       Red       9.8            11.2               0.28
## 5           9.93 Ok         Red       9.4             7.4               0.7
## # ... with 6,492 more rows, and 9 more variables: citric acid <dbl>,
## #   residual sugar <dbl>, chlorides <dbl>, free sulfur dioxide <dbl>,
## #   total sulfur dioxide <dbl>, density <dbl>, pH <dbl>, sulphates <dbl>,
## #   quality <dbl>
```

Quick Examples

- Go to the course files page and try Exercise 2 - if else

Efficient Code

Some ‘built-in’ vectorized functions

- `colMeans()`, `rowMeans()`
- `colSums()`, `rowSums()`
- `colSds()`, `colVars()`, `colMedians()` (`matrixStats` package)
- `ifelse()`, `dplyr::if_else()`
- `apply()` family
- Create your own with `Vectorize()`

`apply()` family

- `apply()` family of functions *pretty* fast
- We'll look at `apply()`, `lapply()`, `sapply()`, and `replicate()`

`apply()` family

- Use `apply()` to find summary for columns of wine data

```
apply(X = wineData %>% select(-type, -qualityCat),
      MARGIN = 2,
      FUN = summary,
      na.rm = TRUE)
```

```
##         fixed acidity volatile acidity citric acid residual sugar  chlorides
## Min.         3.800000         0.080000   0.0000000       0.600000 0.00900000
## 1st Qu.      6.400000         0.230000   0.2500000       1.800000 0.03800000
## Median       7.000000         0.290000   0.3100000       3.000000 0.04700000
## Mean         7.215307         0.339666   0.3186332       5.443235 0.05603386
## 3rd Qu.      7.700000         0.400000   0.3900000       8.100000 0.06500000
## Max.        15.900000         1.580000   1.6600000      65.800000 0.61100000
##         free sulfur dioxide total sulfur dioxide   density       pH sulphates
## Min.                1.00000              6.0000 0.9871100 2.720000 0.2200000
## 1st Qu.            17.00000             77.0000 0.9923400 3.110000 0.4300000
## Median             29.00000            118.0000 0.9948900 3.210000 0.5100000
## Mean               30.52532            115.7446 0.9946966 3.218501 0.5312683
## 3rd Qu.            41.00000            156.0000 0.9969900 3.320000 0.6000000
## Max.              289.00000            440.0000 1.0389800 4.010000 2.0000000
##         alcohol  quality
## Min.     8.0000 3.000000
## 1st Qu.  9.5000 5.000000
```

lapply

- `lapply()` to apply a function to a list
- Create a list object

```
myList <- list(
  norm = rnorm(100),
  unif = runif(25),
  gamma = rgamma(500, rate = 1, shape = 1)
)
```

lapply

- Apply `mean()` function to each list element

```
lapply(X = myList, FUN = mean)
```

```
## $norm
## [1] 0.007024213
##
## $unif
## [1] 0.5315503
##
## $gamma
## [1] 1.026491
```

sapply

- Use `sapply()` similar but returns a vector if possible

```
sapply(X = myList, FUN = mean)
```

```
##        norm        unif       gamma
## 0.007024213 0.531550253 1.026491492
```

replicate

- `replicate()` function great for repeatedly running code
- Estimate a probability using repeated simulations
- Suppose you select five letters at random. What is the probability none are repeated?

```
sample(size = 5, letters, replace = TRUE)
```

```
## [1] "s" "x" "o" "e" "z"
```

```
sample(size = 5, letters, replace = TRUE)
```

```
## [1] "p" "e" "k" "c" "w"
```

replicate

- Suppose you select five letters at random. What is the probability none are repeated?

```
set.seed(1)
sample(size = 5, letters, replace = TRUE) %>%
  unique()
```

```
## [1] "y" "d" "g" "a" "b"
```

```
set.seed(1)
sample(size = 5, letters, replace = TRUE) %>%
  unique() %>%
  length()
```

```
## [1] 5
```

replicate

- Suppose you select five letters at random. What is the probability none are repeated?

```
set.seed(1)
sample(size = 5, letters, replace = TRUE) %>%
  unique() %>%
  length() == 5
```

```
## [1] TRUE
```

replicate

- Suppose you select five letters at random. What is the probability none are repeated?

```
replicate(5,
  sample(size = 5, letters, replace = TRUE) %>%
    unique() %>%
    length() == 5
)
```

```
## [1]  TRUE FALSE  TRUE  TRUE FALSE
```

replicate

- Suppose you select five letters at random. What is the probability none are repeated?

```
replicate(50000,
  sample(size = 5, letters, replace = TRUE) %>%
    unique() %>%
    length() == 5
) %>%
  mean()
```

```
## [1] 0.66264
```

Recap!

- Vectorized functions fast!
- 'Built-in' vectorized functions
  - `colMeans()`, `rowMeans()`
  - `colSums()`, `rowSums()`
  - `colSds()`, `colVars()`, `colMedians()` (`matrixStats` package)
  - `ifelse()`
  - `apply()` family

Quick Examples

- Go to the course files page and try Exercise 3 - apply