Lesson 1

Lesson 1

File ‘sf_crime-data.csv’ was downloaded and placed in the same folder as this markdown

```
#read.csv("sf_crime-data-2017.csv") -> not in the correct working directory
```

We need to define the directory properly. This can be done by explicitly defining the file path (absolute) as such:

```
head(read.csv("C:/Users/Carissa Ying/Documents/Year 3 Sem 1/DBA3702/Lesson 1/sf_crime-data-20
17.csv"))
```

```
##   IncidntNum               Category                      Descript DayOfWeek
## 1  170533616                  FRAUD  CREDIT CARD, THEFT BY USE OF    Sunday
## 2  170527017 SEX OFFENSES, FORCIBLE                SEXUAL BATTERY    Sunday
## 3  170514133         SUSPICIOUS OCC         SUSPICIOUS OCCURRENCE    Sunday
## 4  170465285           NON-CRIMINAL                FOUND PROPERTY    Sunday
## 5  170451814           NON-CRIMINAL                 LOST PROPERTY    Sunday
## 6  170451109                  FRAUD FRAUDULENT CREDIT APPLICATION    Sunday
##         Date  Time PdDistrict Resolution                    Address         X
## 1 01/01/2017 08:00  INGLESIDE       NONE 3800 Block of SAN BRUNO AV -122.4018
## 2 01/01/2017 00:01    MISSION       NONE   100 Block of GUERRERO ST -122.4246
## 3 01/01/2017 13:00  INGLESIDE       NONE        0 Block of FOOTE AV -122.4451
## 4 01/01/2017 00:01   SOUTHERN       NONE    1200 Block of MARKET ST -122.4154
## 5 01/01/2017 00:01   RICHMOND       NONE     5100 Block of GEARY BL -122.4745
## 6 01/01/2017 15:00    TARAVAL       NONE      1400 Block of 45TH AV -122.5047
##          Y                                  Location         PdId
## 1 37.71452  (37.71452361297482, -122.40176175711419) 1.705336e+13
## 2 37.76854  (37.76854434073097, -122.42458239880908) 1.705270e+13
## 3 37.71295  (37.71294771749951, -122.44505165759051) 1.705141e+13
## 4 37.77829 (37.778293520129026, -122.41544875956173) 1.704653e+13
## 5 37.78043 (37.780434756690276, -122.47453735746846) 1.704518e+13
## 6 37.75952  (37.75952426807323, -122.50472074943889) 1.704511e+13
```

However, this is not recommended since it affects code sharing. We can use relative paths instead.

```
# other users can change this part to suit their own systems
setwd("C:/Users/Carissa Ying/Documents/Year 3 Sem 1/DBA3702/Lesson 1")
head(read.csv("sf_crime-data-2017.csv"))
```

```
##   IncidntNum              Category                      Descript DayOfWeek
## 1  170533616                 FRAUD  CREDIT CARD, THEFT BY USE OF    Sunday
## 2  170527017 SEX OFFENSES, FORCIBLE                SEXUAL BATTERY    Sunday
## 3  170514133         SUSPICIOUS OCC         SUSPICIOUS OCCURRENCE    Sunday
## 4  170465285           NON-CRIMINAL                FOUND PROPERTY    Sunday
## 5  170451814           NON-CRIMINAL                 LOST PROPERTY    Sunday
## 6  170451109                 FRAUD FRAUDULENT CREDIT APPLICATION    Sunday
##         Date  Time PdDistrict Resolution                    Address         X
## 1 01/01/2017 08:00  INGLESIDE       NONE 3800 Block of SAN BRUNO AV -122.4018
## 2 01/01/2017 00:01    MISSION       NONE   100 Block of GUERRERO ST -122.4246
## 3 01/01/2017 13:00  INGLESIDE       NONE        0 Block of FOOTE AV -122.4451
## 4 01/01/2017 00:01   SOUTHERN       NONE    1200 Block of MARKET ST -122.4154
## 5 01/01/2017 00:01   RICHMOND       NONE     5100 Block of GEARY BL -122.4745
## 6 01/01/2017 15:00    TARAVAL       NONE      1400 Block of 45TH AV -122.5047
##          Y                                Location         PdId
## 1 37.71452  (37.71452361297482, -122.40176175711419) 1.705336e+13
## 2 37.76854  (37.76854434073097, -122.42458239880908) 1.705270e+13
## 3 37.71295  (37.71294771749951, -122.44505165759051) 1.705141e+13
## 4 37.77829 (37.778293520129026, -122.41544875956173) 1.704653e+13
## 5 37.78043 (37.780434756690276, -122.47453735746846) 1.704518e+13
## 6 37.75952  (37.75952426807323, -122.50472074943889) 1.704511e+13
```

```
# . refers to the current working directory
head(read.csv("./sf_crime-data-2017.csv"))
```

```
##   IncidntNum              Category                      Descript DayOfWeek
## 1  170533616                 FRAUD  CREDIT CARD, THEFT BY USE OF    Sunday
## 2  170527017 SEX OFFENSES, FORCIBLE                SEXUAL BATTERY    Sunday
## 3  170514133         SUSPICIOUS OCC         SUSPICIOUS OCCURRENCE    Sunday
## 4  170465285           NON-CRIMINAL                FOUND PROPERTY    Sunday
## 5  170451814           NON-CRIMINAL                 LOST PROPERTY    Sunday
## 6  170451109                 FRAUD FRAUDULENT CREDIT APPLICATION    Sunday
##         Date  Time PdDistrict Resolution                    Address         X
## 1 01/01/2017 08:00  INGLESIDE       NONE 3800 Block of SAN BRUNO AV -122.4018
## 2 01/01/2017 00:01    MISSION       NONE   100 Block of GUERRERO ST -122.4246
## 3 01/01/2017 13:00  INGLESIDE       NONE        0 Block of FOOTE AV -122.4451
## 4 01/01/2017 00:01   SOUTHERN       NONE    1200 Block of MARKET ST -122.4154
## 5 01/01/2017 00:01   RICHMOND       NONE     5100 Block of GEARY BL -122.4745
## 6 01/01/2017 15:00    TARAVAL       NONE      1400 Block of 45TH AV -122.5047
##          Y                                Location         PdId
## 1 37.71452  (37.71452361297482, -122.40176175711419) 1.705336e+13
## 2 37.76854  (37.76854434073097, -122.42458239880908) 1.705270e+13
## 3 37.71295  (37.71294771749951, -122.44505165759051) 1.705141e+13
## 4 37.77829 (37.778293520129026, -122.41544875956173) 1.704653e+13
## 5 37.78043 (37.780434756690276, -122.47453735746846) 1.704518e+13
## 6 37.75952  (37.75952426807323, -122.50472074943889) 1.704511e+13
```

```
# .. navigates to the parent directory
```

It is also possible to set the root working directory under Tools > Global Options > General

Packages

You can install packages by using the ‘install.packages(“dplyr”)’ command. You can then load the package using ‘library(dplyr)’

Lecture 2

Data Management

The process of storing, acquiring, validating, processing, storing and protecting data to ensure the accessibility, reliability, and timeliness of the data for users.

Data Workflow

1. Collection and Storage
2. Preparation
3. Analysis and Visualisation
4. Experimentation and Prediction

Data Types

1. Structured Data

- stored in relational databases

2. Unstructured Data

- stored in non-relational (aka non-SQL) databases
- e.g. text, images, audio

Structured Data

- every column has a dedicated meaning and a single, fixed data type
- columns are called data variables
- rows are called data records

```
head(read.csv("./Police Use of Force.csv"), 3)
```

```
##   INCIDENT_DATE INCIDENT_TIME UOF_NUMBER OFFICER_ID OFFICER_GENDER OFFICER_RACE
## 1      9/3/2016    4:14:00 AM      37702      10810           Male        Black
## 2       3/22/16   11:00:00 PM      33413       7706           Male        White
## 3       5/22/16    1:29:00 PM      34567      11014           Male        Black
##   OFFICER_HIRE_DATE OFFICER_YEARS_ON_FORCE OFFICER_INJURY
## 1          5/7/2014                     2             No
## 2          1/8/1999                    17            Yes
## 3           5/20/15                     1             No
##            OFFICER_INJURY_TYPE OFFICER_HOSPITALIZATION SUBJECT_ID SUBJECT_RACE
## 1 No injuries noted or visible                      No      46424        Black
## 2                Sprain/Strain                     Yes      44324     Hispanic
## 3 No injuries noted or visible                      No      45126     Hispanic
##   SUBJECT_GENDER SUBJECT_INJURY          SUBJECT_INJURY_TYPE
## 1         Female            Yes      Non-Visible Injury/Pain
## 2           Male             No No injuries noted or visible
## 3           Male             No No injuries noted or visible
##   SUBJECT_WAS_ARRESTED SUBJECT_DESCRIPTION SUBJECT_OFFENSE REPORTING_AREA BEAT
## 1                  Yes   Mentally unstable           APOWW           2062  134
## 2                  Yes   Mentally unstable           APOWW           1197  237
## 3                  Yes             Unknown           APOWW           4153  432
##   SECTOR  DIVISION LOCATION_DISTRICT STREET_NUMBER  STREET_NAME
## 1    130   CENTRAL               D14           211        Ervay
## 2    230 NORTHEAST                D9          7647     Ferguson
## 3    430 SOUTHWEST                D6           716 bimebella dr
##   STREET_DIRECTION STREET_TYPE LOCATION_FULL_STREET_ADDRESS_OR_INTERSECTION
## 1                N         St.                                211 N ERVAY ST
## 2             NULL         Rd.                               7647 FERGUSON RD
## 3             NULL         Ln.                               716 BIMEBELLA LN
##   LOCATION_CITY LOCATION_STATE LOCATION_LATITUDE LOCATION_LONGITUDE
## 1        Dallas             TX          32.78220          -96.79746
## 2        Dallas             TX          32.79898          -96.71749
## 3        Dallas             TX          32.73971          -96.92519
##   INCIDENT_REASON REASON_FOR_FORCE   TYPE_OF_FORCE_USED1 TYPE_OF_FORCE_USED2
## 1          Arrest           Arrest Hand/Arm/Elbow Strike
## 2          Arrest           Arrest           Joint Locks
## 3          Arrest           Arrest     Take Down - Group
##   TYPE_OF_FORCE_USED3 TYPE_OF_FORCE_USED4 TYPE_OF_FORCE_USED5
## 1
## 2
## 3
##   TYPE_OF_FORCE_USED6 TYPE_OF_FORCE_USED7 TYPE_OF_FORCE_USED8
## 1
## 2
## 3
##   TYPE_OF_FORCE_USED9 TYPE_OF_FORCE_USED10 NUMBER_EC_CYCLES FORCE_EFFECTIVE
## 1                                                      NULL             Yes
## 2                                                      NULL             Yes
## 3                                                      NULL             Yes
```

5 Atomic Data Types

You can use class(x) to find the type of a variable.

1. Logical

- TRUE(T) / FALSE(F)

2. Integer
3. Numeric

- decimals

4. Complex
5. Character

- non-numeric

```
a = T # short for TRUE
class(a)
```

```
## [1] "logical"
```

```
# read
data <- read.csv("./Police Use of Force.csv", stringsAsFactors = F)
# run the structure data command, gives the data type of each variable
str(data)
```

```
## 'data.frame':    2383 obs. of  47 variables:
##  $ INCIDENT_DATE                                : chr  "9/3/2016" "3/22/16" "5/22/16" "1/1
0/2016" ...
##  $ INCIDENT_TIME                                : chr  "4:14:00 AM" "11:00:00 PM" "1:29:00
PM" "8:55:00 PM" ...
##  $ UOF_NUMBER                                   : chr  "37702" "33413" "34567" "31460" ...
##  $ OFFICER_ID                                   : int  10810 7706 11014 6692 9844 9855 9881
9058 10381 9705 ...
##  $ OFFICER_GENDER                               : chr  "Male" "Male" "Male" "Male" ...
##  $ OFFICER_RACE                                 : chr  "Black" "White" "Black" "Black" ...
##  $ OFFICER_HIRE_DATE                            : chr  "5/7/2014" "1/8/1999" "5/20/15" "7/2
9/91" ...
##  $ OFFICER_YEARS_ON_FORCE                       : int  2 17 1 24 7 7 7 9 4 8 ...
##  $ OFFICER_INJURY                               : chr  "No" "Yes" "No" "No" ...
##  $ OFFICER_INJURY_TYPE                          : chr  "No injuries noted or visible" "Spra
in/Strain" "No injuries noted or visible" "No injuries noted or visible" ...
##  $ OFFICER_HOSPITALIZATION                      : chr  "No" "Yes" "No" "No" ...
##  $ SUBJECT_ID                                   : int  46424 44324 45126 43150 47307 46549
47555 44172 43723 47889 ...
##  $ SUBJECT_RACE                                 : chr  "Black" "Hispanic" "Hispanic" "Hispa
nic" ...
##  $ SUBJECT_GENDER                               : chr  "Female" "Male" "Male" "Male" ...
##  $ SUBJECT_INJURY                               : chr  "Yes" "No" "No" "Yes" ...
##  $ SUBJECT_INJURY_TYPE                          : chr  "Non-Visible Injury/Pain" "No injuri
es noted or visible" "No injuries noted or visible" "Laceration/Cut" ...
##  $ SUBJECT_WAS_ARRESTED                         : chr  "Yes" "Yes" "Yes" "Yes" ...
##  $ SUBJECT_DESCRIPTION                          : chr  "Mentally unstable" "Mentally unstab
le" "Unknown" "FD-Unknown if Armed" ...
##  $ SUBJECT_OFFENSE                              : chr  "APOWW" "APOWW" "APOWW" "Evading Arr
est" ...
##  $ REPORTING_AREA                               : int  2062 1197 4153 4523 2167 1134 2049 3
122 2072 4403 ...
##  $ BEAT                                         : int  134 237 432 641 346 235 132 515 133
614 ...
##  $ SECTOR                                       : int  130 230 430 640 340 230 130 510 130
610 ...
##  $ DIVISION                                     : chr  "CENTRAL" "NORTHEAST" "SOUTHWEST" "N
ORTH CENTRAL" ...
##  $ LOCATION_DISTRICT                            : chr  "D14" "D9" "D6" "D11" ...
##  $ STREET_NUMBER                                : int  211 7647 716 5600 4600 1234 511 4709
300 18600 ...
##  $ STREET_NAME                                  : chr  "Ervay" "Ferguson" "bimebella dr" "L
BJ" ...
##  $ STREET_DIRECTION                             : chr  "N" "NULL" "NULL" "NULL" ...
##  $ STREET_TYPE                                  : chr  "St." "Rd." "Ln." "Frwy." ...
##  $ LOCATION_FULL_STREET_ADDRESS_OR_INTERSECTION: chr  "211 N ERVAY ST" "7647 FERGUSON RD"
"716 BIMEBELLA LN" "5600 L B J FWY" ...
##  $ LOCATION_CITY                                : chr  "Dallas" "Dallas" "Dallas" "Dallas"
...
##  $ LOCATION_STATE                               : chr  "TX" "TX" "TX" "TX" ...
##  $ LOCATION_LATITUDE                            : num  32.8 32.8 32.7 NA NA ...
##  $ LOCATION_LONGITUDE                           : num  -96.8 -96.7 -96.9 NA NA ...
##  $ INCIDENT_REASON                              : chr  "Arrest" "Arrest" "Arrest" "Arrest"
...
##  $ REASON_FOR_FORCE                             : chr  "Arrest" "Arrest" "Arrest" "Arrest"
...
##  $ TYPE_OF_FORCE_USED1                          : chr  "Hand/Arm/Elbow Strike" "Joint Lock
```

```
s" "Take Down - Group" "K-9 Deployment" ...
##  $ TYPE_OF_FORCE_USED2                    : chr  "" "" "" "" ...
##  $ TYPE_OF_FORCE_USED3                    : chr  "" "" "" "" ...
##  $ TYPE_OF_FORCE_USED4                    : chr  "" "" "" "" ...
##  $ TYPE_OF_FORCE_USED5                    : chr  "" "" "" "" ...
##  $ TYPE_OF_FORCE_USED6                    : chr  "" "" "" "" ...
##  $ TYPE_OF_FORCE_USED7                    : chr  "" "" "" "" ...
##  $ TYPE_OF_FORCE_USED8                    : chr  "" "" "" "" ...
##  $ TYPE_OF_FORCE_USED9                    : chr  "" "" "" "" ...
##  $ TYPE_OF_FORCE_USED10                   : chr  "" "" "" "" ...
##  $ NUMBER_EC_CYCLES                       : chr  "NULL" "NULL" "NULL" "NULL" ...
##  $ FORCE_EFFECTIVE                        : chr  " Yes" " Yes" " Yes" " Yes" ...
```

Logical

Often the result of a relational operation

```
a = 7
a > 8 # FALSE, 7 is less than 8
```

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

Can also be the result of a logical operation

```
a = 8 > 7 # TRUE
b = 5 == 3 # FALSE
a & b # FALSE
```

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

```
a | b # TRUE
```

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

Integer and Numeric

An integer is a subset of numeric. If a numeric has no decimal, it does not mean that it is treated as an integer. **By default, any number assigned to a variable will be treated as numeric**.

```
a = 3.5
class(a) # numeric
```

```
## [1] "numeric"
```

```
b = 3
class(b) # numeric
```

```
## [1] "numeric"
```

To make a variable an integer there are 2 methods. First is to use the function as.integer(x):

```
class(3)
```

```
## [1] "numeric"
```

```
class(as.integer(3))
```

```
## [1] "integer"
```

The other is to add a capital ‘L’ to the end of the number

```
class(3)
```

```
## [1] "numeric"
```

```
class(3L)
```

```
## [1] "integer"
```

During mathematical calculations, the more inclusive type is used:

```
class(4L) # integer
```

```
## [1] "integer"
```

```
class(1) # numeric
```

```
## [1] "numeric"
```

```
class(4L - 1) # numeric
```

```
## [1] "numeric"
```

Character

Aka text / strings

1. Length

```
nchar("hello test") # returns the number of char, including space
```

```
## [1] 10
```

2. Concatenate

```
paste("I", "am", "taking", "DBA3702") # separated by spaces by default
```

```
## [1] "I am taking DBA3702"
```

```
paste("I", "am", "taking", "DBA3702", sep = "%") # change separator
```

```
## [1] "I%am%taking%DBA3702"
```

3. Split

```
strsplit("I am taking DBA3702", " ") # returns a list
```

```
## [[1]]
## [1] "I"       "am"      "taking"  "DBA3702"
```

4. Substring

```
substr("I am taking DBA3702", 13, 19) # indexes are inclusive
```

```
## [1] "DBA3702"
```

```
substring("I am taking DBA3702", 13) # only specify start index
```

```
## [1] "DBA3702"
```

5. Replace

```
txt = "According to all known laws of aviation, there is no way that a B should be able to fl
y. Its wings are too small to get its fat little body off the ground. The B, of course, flies
anyways. Because Bs don't care what humans think is impossible."
gsub("B", "bee", txt)
```

```
## [1] "According to all known laws of aviation, there is no way that a bee should be able to
fly. Its wings are too small to get its fat little body off the ground. The bee, of course, f
lies anyways. beeecause bees don't care what humans think is impossible."
```

Other replacement functions include: - grep - grepl - sub - regexpr - gregexpr - regexev

Other useful functions include: - toupper - tolower - abbreviate - trimws

The package “stringr” also contains useful functions for handling strings.

Data Structures

A data structure is a data organization, management, and storage format that enables efficient access and modification of data.

It is a collection of data values, the relationships among them and the functions and operations that can be applied to the data.

Basic R Data Structures

1. Vector
2. Matrix
3. List
4. Data Frame

Vector

A vector is a sequence of data elements of the same basic data type. As some may realise, this is a similar condition to that of a column in a structured data set.

```
data = read.csv("./Police Use of Force.csv")
col = data$INCIDENT_DATE
# confirm that the columns is a vector
is.vector(col)
```

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

The most common method to create a vector is the function c(x):

```
v1 = c(1, 2, 3)
is.vector(v1)
```

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

```
v2 = c(First=100, Second=200, Third=300) # vector with labels
v2
```

```
##  First Second  Third
##    100    200    300
```

A common way to create a sequence of consecutive number is to use a ':' or the seq(from=x, to=y, by=z) function:

```
3:10 # start and end inclusive
```

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

```
5:-3
```

```
## [1]  5  4  3  2  1  0 -1 -2 -3
```

```
seq(from=1, to=10, by=3)
```

```
## [1]  1  4  7 10
```

```
seq(2, 10, 2)
```

```
## [1]  2  4  6  8 10
```

```
seq(2, 10, length.out=5) # length.out dictates the number of elements
```

```
## [1]  2  4  6  8 10
```

```
2 * 1:5
```

```
## [1]  2  4  6  8 10
```

```
2 * (1:5)
```

```
## [1]  2  4  6  8 10
```

R also has built in vectors for common uses:

```
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] "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"
```

```
month.abb
```

```
##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

```
month.name
```

```
##  [1] "January"   "February"  "March"     "April"     "May"       "June"
##  [7] "July"      "August"    "September" "October"   "November"  "December"
```

All data have to have the same type. Otherwise, the other data will be converted to a suitable data type.

```
v1 = c(1, F, "a")
```

This process of converting values to the correct data type is called coercion hierarchy. It follows the order, logical < integer < numeric < complex < character.

Mathematical Operations on Vectors

```
a = c(1, 2, 3)
b = c(5, 6, 7)
a + b
```

```
## [1]  6  8 10
```

```
b ^ a
```

```
## [1]   5  36 343
```

When 2 vectors have different lengths, R automaticaly repeats elements in the shorter on until it is long enough to match the long vector. This is called **recycling**.

```
a = 1:10
b = 0:1
a * b
```

```
##  [1]  0  2  0  4  0  6  0  8  0 10
```

Difference between ‘&’ and ‘&&’

‘&’ operates on all elements and returns a vector of logical types. ‘&&’ operates on only the first element in each of the vectors.

```
a = c(T, F, T, F)
b = c(T, T, F, F)
c = c(F, T)

a & b # element-wise
```

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

```
a && b # first pair
```

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

```
a && c
```

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

Vector Manipulation

We can get an element using its position in the vector:

```
salaries = c(John=10000, James=8000, Lily=5600, Jo=7000)
salaries
```

```
##  John James  Lily    Jo
## 10000  8000  5600  7000
```

```
# retrieve
salaries[2]
```

```
## James
##  8000
```

```
salaries[2:3]
```

```
## James  Lily
##  8000  5600
```

```
salaries[c(1,3)]
```

```
##  John  Lily
## 10000  5600
```

We can also edit values in the vector, like so:

```
salaries[2] = 9000
salaries
```

```
##  John James  Lily    Jo
## 10000  9000  5600  7000
```

```
salaries[2:3] = c(8500, 6500)
salaries
```

```
##  John James  Lily    Jo
## 10000  8500  6500  7000
```

```
# can use label if vector is named, must use double quotes
salaries["John"] = 10200
salaries
```

```
##  John James  Lily    Jo
## 10200  8500  6500  7000
```

Negative indexes can be used to exclude elements.

```
salaries[-2]
```

```
##  John  Lily    Jo
## 10200  6500  7000
```

```
salaries[-(1:2)]
```

```
## Lily   Jo
## 6500 7000
```

```
# salaries[-1][1:2] will first process (salaries[-1]) and exclude John. Then, the first 2 ele
ments of the result are taken
salaries[-1][1:2]
```

```
## James  Lily
##  8500  6500
```

We can also find elements that meet a certain logical criteria.

```
salaries > 7500 # returns logical vector
```

```
##  John James  Lily    Jo
##  TRUE  TRUE FALSE FALSE
```

```
salaries[salaries > 7500] # filters out F
```

```
##  John James
## 10200  8500
```

Lastly, we can sort a vector

```
asc = order(salaries) # returns a integer vector of the order
desc = order(-salaries)
salaries[asc]
```

```
##  Lily    Jo James  John
##  6500  7000  8500 10200
```

```
salaries[desc]
```

```
##  John James    Jo  Lily
## 10200  8500  7000  6500
```

Matrices

Matrices store data in a two-dimensional struction. Matrices can be created by arranging a vector by rows and/or columns.

```
# arrange by columns
matrix(1:12, nrow=4)
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
matrix(1:10, ncol=3) # will recycle
```

```
## Warning in matrix(1:10, ncol = 3): data length [10] is not a sub-multiple or
## multiple of the number of rows [4]
```

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

```
#arrange by rows
matrix(1:12, nrow=4, byrow=T)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
```

Another way of creating matrices is to bind multiple rows/columns together.

```
a = 1:5
b = 2:4
c = 3:6

rbind(a, b, c) # by rows
```

```
## Warning in rbind(a, b, c): number of columns of result is not a multiple of
## vector length (arg 2)
```

```
##   [,1] [,2] [,3] [,4] [,5]
## a    1    2    3    4    5
## b    2    3    4    2    3
## c    3    4    5    6    3
```

```
cbind(a, b, c) # by cols
```

```
## Warning in cbind(a, b, c): number of rows of result is not a multiple of vector
## length (arg 2)
```

```
##      a b c
## [1,] 1 2 3
## [2,] 2 3 4
## [3,] 3 4 5
## [4,] 4 2 6
## [5,] 5 3 3
```

The same commands can be used to grow a matrix

```
m = matrix(1:25, nrow=5, byrow=T)
a = c(0,1)
rbind(m, a)
```

```
## Warning in rbind(m, a): number of columns of result is not a multiple of vector
## length (arg 2)
```

```
##   [,1] [,2] [,3] [,4] [,5]
##      1    2    3    4    5
##      6    7    8    9   10
##     11   12   13   14   15
##     16   17   18   19   20
##     21   22   23   24   25
## a    0    1    0    1    0
```

```
cbind(a, m)
```

```
## Warning in cbind(a, m): number of rows of result is not a multiple of vector
## length (arg 1)
```

```
##      a
## [1,] 0  1  2  3  4  5
## [2,] 1  6  7  8  9 10
## [3,] 0 11 12 13 14 15
## [4,] 1 16 17 18 19 20
## [5,] 0 21 22 23 24 25
```

Similar to vectors, we can also name the columns and rows of matrices.

```
colnames(m) = c("Col1", "Col2", "Col3", "Col4", "Col5")
rownames(m) = c("Row1", "Row2", "Row3", "Row4", "Row5")
m
```

```
##      Col1 Col2 Col3 Col4 Col5
## Row1    1    2    3    4    5
## Row2    6    7    8    9   10
## Row3   11   12   13   14   15
## Row4   16   17   18   19   20
## Row5   21   22   23   24   25
```

Mathematical Operations on Matrices

```
m1 = matrix(1:12, nrow=4)
m2 = matrix(c(1,-1), nrow=4, ncol=3)
m1 * m2
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]   -2   -6  -10
## [3,]    3    7   11
## [4,]   -4   -8  -12
```

We can also transpose a matrix using the function t(x)

```
m1
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
t(m1)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

We can also perform matrix multiplication using '%*%'.

```
m1 = matrix(1:6, nrow=2, ncol=3)
m2 = matrix(c(1, -1, 2, -2), nrow=3, ncol=4)
m1 %*% m2 # results in a 2 x 4 matrix
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    8   -4    1   -5
## [2,]   10   -6    2   -6
```

Manipulating Matrices

We can retrieve elements in a way similar to vectors, only that we supply both the row and the column this time.

```
m = matrix(1:12, ncol=4, byrow=T)
colnames(m) = c("Col1", "Col2", "Col3", "Col4")
rownames(m) = c("Row1", "Row2", "Row3")
m
```

```
##      Col1 Col2 Col3 Col4
## Row1    1    2    3    4
## Row2    5    6    7    8
## Row3    9   10   11   12
```

```
# retrieve
m[2,3:4]
```

```
## Col3 Col4
##    7    8
```

```
# leaving a field blank means all elements for that row/col
m[, c("Col1", "Col3")]
```

```
##      Col1 Col3
## Row1    1    3
## Row2    5    7
## Row3    9   11
```

We can also find the maximum, sum rows/columns, and filter.

```
sales = rbind(c(100, 102, 95, 200),
              c(95, 60, 50, 120),
              c(150, 100, 98, 205))
colnames(sales) = c("Stephen", "Alice", "Mary", "John")
rownames(sales) = c("October", "November", "December")
sales
```

```
##          Stephen Alice Mary John
## October      100   102   95  200
## November      95    60   50  120
## December     150   100   98  205
```

```
# find max
sales[which.max(sales)]
```

```
## [1] 205
```

```
# sum
rowSums(sales)
```

```
##  October November December
##      497      325      553
```

```
# filter
sales[ , colSums(sales) > 300]
```

```
##          Stephen John
## October      100  200
## November      95  120
## December     150  205
```

Lists

The requirement of same data type limits the usage of vectors and matrices. Some data might appear in different types and need to be stored differently.

List is a 1-dimensional structure that allows for a variety of data types. Lists can even store other data structures.However, note that it will lose some of it's efficiency.

```
list(1, 2, 3)
```

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

```
steven = list(first.name="Steven",
              age=30, married=T,
              children=c("Mary", "Sue"))
steven
```

```
## $first.name
## [1] "Steven"
##
## $age
## [1] 30
##
## $married
## [1] TRUE
##
## $children
## [1] "Mary" "Sue"
```

Like a vector, we can access elements by their position. Additionally, we can use '$' to access elements by their name.

```
# returns lists
steven["first.name"]
```

```
## $first.name
## [1] "Steven"
```

```
steven[c(1,3)]
```

```
## $first.name
## [1] "Steven"
##
## $married
## [1] TRUE
```

```
# returns element
steven$age
```

```
## [1] 30
```

Difference between '[]' and '[[]]'

When retrieving an element from a list using '[]', the result is still a list. '[[]]' opens the structure and can hence only return one element at a time.

```
steven["children"][2] # cannot retrieve
```

```
## $<NA>
## NULL
```

```
steven[["children"]][2]
```

```
## [1] "Sue"
```

```
steven$children[2]
```

```
## [1] "Sue"
```

Difference between '=' and '<-'

'=' assigns variables in a smaller scope. '<-' assigns variables on a global scope.

```
rm(x)
```

```
## Warning in rm(x): object 'x' not found
```

```
mean(x=1:10)
```

```
## [1] 5.5
```

```
# finding x now will produce an error
mean(x<-1:10)
```

```
## [1] 5.5
```

```
x
```

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

Factors

In the “Police Use of Force.csv” file, we have 3 types of character columns: date-time, free text, and categorical.

Categorical values need to be handled differently because: - statistical analysis (e.g. counting) by category - improve memory usage, reduce storing of repeated strings - need to order and rank categories - needed in machine learning

In R, we use **factors** to represent categories.

```
data = read.csv("./Police Use of Force.csv", stringsAsFactors=F)

head(data$OFFICER_GENDER, 20)
```

```
##  [1] "Male"   "Male"   "Male"   "Male"   "Male"   "Male"   "Male"   "Female"
##  [9] "Male"   "Male"   "Male"   "Male"   "Male"   "Male"   "Male"   "Male"
## [17] "Female" "Male"   "Male"   "Male"
```

```
object.size(data$OFFICER_GENDER)
```

```
## 19224 bytes
```

```
# convert to factor
data$OFFICER_GENDER = factor(data$OFFICER_GENDER)
head(data$OFFICER_GENDER, 20)
```

```
##  [1] Male   Male   Male   Male   Male   Male   Male   Female Male   Male
## [11] Male   Male   Male   Male   Male   Male   Female Male   Male   Male
## Levels: Female Male
```

```
object.size(data$OFFICER_GENDER)
```

```
## 10096 bytes
```

Note that the categories are actually stored as integers. Female is stored as 1, Male is stored as 2. This is ordered alphabetically and can be explicitly changed using the factor(x, levels) function.

```
as.integer(head(data$OFFICER_GENDER, 20))
```

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

```
a = c("M", "F", "M") # ordered by alphabet
b = as.factor(a)
c(b,"4")
```

```
## [1] "2" "1" "2" "4"
```

```
# change order
b = factor(a, levels=c("M", "F"))
c(b, "4")
```

```
## [1] "1" "2" "1" "4"
```

We can count categories usign the table(x) function

```
table(data$OFFICER_GENDER)
```

```
##
## Female   Male
##    240   2143
```

Lecture 3

Data Wrangling

- the combined process of data pre-processing and data transformation
- data pre-processing is the stage of cleaning data (e.g. empty values, repeated values)
- data transformation is the convertion of raw data to a useful data format

---

Data Frames

What if we have data on all the company's customers, including information like their name, birthday, contract duration, etc. What would be the best structure to store all these data?

Is it a matrix? Matrices have a square structure that allows for quick comparisons acros columns and rows. This is good for anylsis. However, it can only store 1 data type. All our data will be stored as characters!

Is it a list? A list is flexible enough to allow for multiple data types. However, it lacks the square structure that a matrix has. It is hence more difficult to make comparisons or search in a list.

The ideal structure in this case is the **data frame**. Similar to the data frame structure in Python, it is a happy mixture of both the list and matrix data structures.

Working with Data Frames

- each column contains values about a specific attribute of the data
- each row contains all information about a specific instance of the data
- within a column, the data type is the same
- between different columns, the data type can be different
- can be thought of as a list of vectors

Creating the data frame manually:

```
# method 1
Name = c("James", "Joe", "Janice", "Jessica")
Position = c("CEO", "VF", "Manager", "Clerk")
Gender = c("M", "M", "F", "F")
Salary = c(30000, 13000, 9800, 3500)
employees = data.frame(Name, Position, Gender, Salary)

# method 2
employees = data.frame(Name=c("James", "Joe", "Janice", "Jessica"),
                       Position=c("CEO", "VF", "Manager", "Clerk"),
                       Gender=c("M", "M", "F", "F"),
                       Salary=c(30000, 13000, 9800, 3500))

employees
```

```
##      Name Position Gender Salary
## 1   James      CEO      M  30000
## 2     Joe       VF      M  13000
## 3  Janice  Manager      F   9800
## 4 Jessica    Clerk      F   3500
```

Read from an existing file:

```
df = read.csv("./Data/SRX_Data.csv")
class(df)
```

```
## [1] "data.frame"
```

Accessing a Data Frame

```
# single square bracket -> returns a data frame
employees[1]
```

```
##      Name
## 1   James
## 2     Joe
## 3  Janice
## 4 Jessica
```

```
class(employees[1])
```

```
## [1] "data.frame"
```

```
# double square brackets -> returns a vector
employees[[1]]
```

```
## [1] "James"   "Joe"     "Janice"  "Jessica"
```

```
is.vector(employees[[1]])
```

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

```
# dollar sign
employees$Position
```

```
## [1] "CEO"     "VF"      "Manager" "Clerk"
```

```
is.vector(employees$Position)
```

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

```
# indexing
employees[2,3]
```

```
## [1] "M"
```

```
employees[2, ]
```

```
##   Name Position Gender Salary
## 2  Joe       VF      M  13000
```

Naming

To facilitate searching, you can name the rows, provided all row names are unique.

```
rownames(employees) = employees$Name
employees
```

```
##            Name Position Gender Salary
## James     James      CEO      M  30000
## Joe         Joe       VF      M  13000
## Janice   Janice  Manager      F   9800
## Jessica Jessica    Clerk      F   3500
```

```
# rows can then be accessed by their names
employees["Joe", "Salary"]
```

```
## [1] 13000
```

Removing Columns

```
employees$Name = NULL
employees
```

```
##         Position Gender Salary
## James        CEO      M  30000
## Joe           VF      M  13000
## Janice   Manager      F   9800
## Jessica    Clerk      F   3500
```

Viewing Data Frames

- type View(df) into the console
- click on the table icon next to the df object in the environment tab

Structure Information

```
# dimensions
dim(employees)
```

```
## [1] 4 3
```

```
# structure
str(employees)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ Position: chr  "CEO" "VF" "Manager" "Clerk"
##  $ Gender  : chr  "M" "M" "F" "F"
##  $ Salary  : num  30000 13000 9800 3500
```

```
# summary
summary(employees) # provides some basic statistics
```

```
##    Position           Gender              Salary
##  Length:4           Length:4           Min.   : 3500
##  Class :character   Class :character   1st Qu.: 8225
##  Mode  :character   Mode  :character   Median :11400
##                                        Mean   :14075
##                                        3rd Qu.:17250
##                                        Max.   :30000
```

Manipulating Data Frames

```
companies = read.csv("./Data/Largest Companies in the World.csv")
summary(companies)
```

```
##  ï..Global.Rank     Company          Sales...billion. Profits...billion.
##  Min.   :   1.0   Length:1924        Min.   :  0.00   Min.   :-24.500
##  1st Qu.: 500.8   Class :character   1st Qu.:  4.10   1st Qu.:  0.300
##  Median : 997.5   Mode  :character   Median :  9.00   Median :  0.600
##  Mean   : 997.2                      Mean   : 19.27   Mean   :  1.226
##  3rd Qu.:1494.2                      3rd Qu.: 18.43   3rd Qu.:  1.200
##  Max.   :1999.0                      Max.   :469.20   Max.   : 44.900
##  Assets...billion.  Market.Value...billion.   Country
##  Min.   :   1.000   Min.   :  0.00          Length:1924
##  1st Qu.:   9.675   1st Qu.:  5.30          Class :character
##  Median :  19.250   Median :  9.60          Mode  :character
##  Mean   :  79.508   Mean   : 19.56
##  3rd Qu.:  45.800   3rd Qu.: 19.20
##  Max.   :3226.200   Max.   :416.60
##   Continent           Location
##  Length:1924        Length:1924
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

Maxima / Minima

```
# finding extreme values
max(companies$Assets...billion.)
```

```
## [1] 3226.2
```

```
min(companies$Assets...billion.)
```

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

```
range(companies$Assets...billion.) # finds both
```

```
## [1]    1.0 3226.2
```

```
# finding position of extreme values in the data frame
idx = which.max(companies$Assets...billion.) # gives the row number
companies[idx, "Company"]
```

```
## [1] "Fannie Mae"
```

Sort

```
# top 10
sorter = order(companies$Assets...billion., decreasing=T)
companies_sorted = companies[sorter, ]
head(companies_sorted, 10)$Company
```

```
##  [1] "Fannie Mae"               "ICBC"
##  [3] "HSBC Holdings"            "Mitsubishi UFJ Financial"
##  [5] "Deutsche Bank"            "BNP Paribas"
##  [7] "CrÃƒÂ©dit Agricole"       "Barclays"
##  [9] "JPMorgan Chase"           "China Construction Bank"
```

Count

```
# returns table object with the counts stored as values
continents = table(companies$Continent)

# convert to df for convenience
continents = as.data.frame((continents))

continents
```

```
##            Var1 Freq
## 1        Africa   25
## 2          Asia  719
## 3        Europe  459
## 4 North America  629
## 5       Oceania   43
## 6 South America   49
```

Filter

```
condition = companies$Continent == "Asia" &
             companies$Profits...billion. >= 30
result = companies[condition, ]

head(result)
```

```
##   ï..Global.Rank                Company Sales...billion. Profits...billion.
## 1              1                   ICBC            134.8               37.8
## 2              2 China Construction Bank            113.1               30.6
##   Assets...billion. Market.Value...billion. Country Continent
## 1            2813.5                   237.3   China      Asia
## 2            2241.0                   202.0   China      Asia
##               Location
## 1 (35.86166,104.195397)
## 2 (35.86166,104.195397)
```

Average

```
mean(companies$Market.Value...billion.)
```

```
## [1] 19.55816
```

Package ‘dplyr’

‘dplyr’ is a powerful package that helps to streamline a lot of data manipulation.

```
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Filter

```
# don't have to repeat companies in the logical operator
set1 = filter(companies, Continent == "Asia"
              & Assets...billion. >= 50)

head(set1)
```

```
##   ï..Global.Rank                    Company Sales...billion. Profits...billion.
## 1              1                       ICBC            134.8               37.8
## 2              2    China Construction Bank            113.1               30.6
## 3              8 Agricultural Bank of China            103.0               23.0
## 4              9                  PetroChina            308.9               18.3
## 5             11              Bank of China             98.1               22.1
## 6             20        Samsung Electronics            187.8               21.7
##   Assets...billion. Market.Value...billion.     Country Continent
## 1            2813.5                   237.3       China      Asia
## 2            2241.0                   202.0       China      Asia
## 3            2124.2                   150.8       China      Asia
## 4             347.8                   261.2       China      Asia
## 5            2033.8                   131.7       China      Asia
## 6             196.3                   174.4 South Korea      Asia
##                 Location
## 1  (35.86166,104.195397)
## 2  (35.86166,104.195397)
## 3  (35.86166,104.195397)
## 4  (35.86166,104.195397)
## 5  (35.86166,104.195397)
## 6 (35.907757,127.766922)
```

Calculations

```
# adds a new column
set2 = mutate(set1, Profit.Margin =
                Profits...billion./Sales...billion.)
head(set2)
```

```
##   ï..Global.Rank                  Company Sales...billion. Profits...billion.
## 1              1                     ICBC            134.8               37.8
## 2              2    China Construction Bank            113.1               30.6
## 3              8 Agricultural Bank of China            103.0               23.0
## 4              9                PetroChina            308.9               18.3
## 5             11              Bank of China             98.1               22.1
## 6             20        Samsung Electronics            187.8               21.7
##   Assets...billion. Market.Value...billion.     Country Continent
## 1            2813.5                   237.3       China      Asia
## 2            2241.0                   202.0       China      Asia
## 3            2124.2                   150.8       China      Asia
## 4             347.8                   261.2       China      Asia
## 5            2033.8                   131.7       China      Asia
## 6             196.3                   174.4 South Korea      Asia
##                 Location Profit.Margin
## 1  (35.86166,104.195397)    0.28041543
## 2  (35.86166,104.195397)    0.27055703
## 3  (35.86166,104.195397)    0.22330097
## 4  (35.86166,104.195397)    0.05924247
## 5  (35.86166,104.195397)    0.22528033
## 6 (35.907757,127.766922)    0.11554846
```

Sort

```
# ascending order by default
# add a negative to sort by descending order
set3 = arrange(set2, -Profit.Margin)
head(set3)
```

```
##   ï..Global.Rank                Company Sales...billion. Profits...billion.
## 1            307 Sun Hung Kai Properties              8.8               5.6
## 2            747   Samba Financial Group              2.0               1.2
## 3            514     Qatar National Bank              4.0               2.3
## 4            539           Al Rajhi Bank              4.0               2.1
## 5            692 National Bank of Kuwait              2.5               1.1
## 6            813              Riyad Bank              2.1               0.9
##   Assets...billion. Market.Value...billion.      Country Continent
## 1              60.3                    36.4    Hong Kong      Asia
## 2              53.1                    10.8 Saudi Arabia      Asia
## 3             100.8                    25.7        Qatar      Asia
## 4              71.3                    26.0 Saudi Arabia      Asia
## 5              58.4                    14.6       Kuwait      Asia
## 6              50.7                     9.1 Saudi Arabia      Asia
##                  Location Profit.Margin
## 1 (22.396428,114.109497)     0.6363636
## 2  (23.885942,45.079162)     0.6000000
## 3  (25.354826,51.183884)     0.5750000
## 4  (23.885942,45.079162)     0.5250000
## 5   (29.31166,47.481766)     0.4400000
## 6  (23.885942,45.079162)     0.4285714
```

Selection

```
set4 = top_n(set3, 10)
```

```
## Selecting by Profit.Margin
```

```
select(set4, Company, Profit.Margin) # selects columns
```

```
##                       Company Profit.Margin
## 1     Sun Hung Kai Properties     0.6363636
## 2       Samba Financial Group     0.6000000
## 3         Qatar National Bank     0.5750000
## 4               Al Rajhi Bank     0.5250000
## 5     National Bank of Kuwait     0.4400000
## 6                   Riyad Bank     0.4285714
## 7     Oversea-Chinese Banking     0.3975904
## 8                 Aozora Bank     0.3750000
## 9  National Bank of Abu Dhabi     0.3548387
## 10                  DBS Group     0.3522727
```

Pipe operator

Currently, we are using 4 intermediate data sets, which take up memory. dplyr has a pipe operator, "%>%", that pushes an argument to a function.

```
set = companies %>%
  filter(Continent == "Asia" & Assets...billion. >= 50) %>%
  mutate(Profit.Margin = Profits...billion./Sales...billion.) %>%
  arrange(-Profit.Margin) %>%
  select(Company, Profit.Margin)

head(set)
```

```
##                   Company Profit.Margin
## 1 Sun Hung Kai Properties     0.6363636
## 2   Samba Financial Group     0.6000000
## 3     Qatar National Bank     0.5750000
## 4           Al Rajhi Bank     0.5250000
## 5 National Bank of Kuwait     0.4400000
## 6              Riyad Bank     0.4285714
```

Data Summary

- statistics summarised by some criteria
- similar to Pivot Table in Excel

Group By and Summarise

First, we group by continents, then specify what we want summarised:

```
companies %>% group_by(Continent) %>%             # group continents
  summarise(Max.Proft = max(Profits...billion.),  # define what to
            No.of.Companies = n(),                # summarise
            Avg.Asset = mean(Assets...billion.))
```

```
## # A tibble: 6 x 4
##   Continent     Max.Proft No.of.Companies Avg.Asset
##   <chr>             <dbl>           <int>     <dbl>
## 1 Africa              2.9              25      25.2
## 2 Asia               37.8             719      66.2
## 3 Europe             40.6             459     122.
## 4 North America      44.9             629      66.6
## 5 Oceania            15.4              43      93.6
## 6 South America      11                49      60.9
```

We can also group by multiple columns, use other dplyr operations or define our own functions to calculate our own summary values:

```
my_func = function(x) {
  half1 = x[seq(1, length(x), 2)] # odd idx
  sq = mean(half1^2)              # square
  half2 = x[seq(2, length(x), 2)] # even idx
  rt = mean(half2^0.5)            # sqrt
  return(sq + rt)
}

companies %>% group_by(Continent) %>%
  summarise(Median.Proft = median(Profits...billion.),
            No.of.Companies = n(),
            My.Value = my_func(Assets...billion.))
```

```
## # A tibble: 6 x 4
##   Continent     Median.Proft No.of.Companies My.Value
##   <chr>                <dbl>           <int>    <dbl>
## 1 Africa                 0.6              25    3084.
## 2 Asia                   0.5             719   54320.
## 3 Europe                 0.6             459  129036.
## 4 North America          0.6             629   91546.
## 5 Oceania                0.6              43   68175.
## 6 South America          0.5              49   14032.
```

Data Format

Right now, the data we have is structured such that each column represents a unique attribute and each row, a unique record. This is called **tidy data** or **data in long format**.

But not all data is structured this way. Here, the columns and rows are flipped:

```
house_w = read.csv("./Data/Home_Ownership_Rate.csv", check.names=F)

head(house_w)
```

```
##                     Property.Type 1980 1990 1995 2000 2001 2002 2003 2004 2005
## 1 Condominiums & Other Apartments 49.2 78.0 83.3 80.8 82.6 82.6 80.9 80.8 77.2
## 2               Landed Properties 75.4 87.6 92.3 89.8 95.0 92.8 92.2 91.1 89.8
## 3                     Other Types 44.5 41.3 50.1 56.1 64.0 74.1 81.7 65.2 57.1
## 4                             HDB 61.5 89.8 90.6 93.1 94.0 94.1 93.4 94.2 93.0
##   2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018
## 1 77.1 77.1 79.6 77.2 76.6 79.0 80.3 82.5 83.3 84.1 84.1 83.6 84.9
## 2 90.1 88.9 89.3 90.2 88.7 88.7 89.5 91.2 90.9 92.0 92.8 92.0 92.8
## 3 62.4 62.4 45.8 51.7 46.8 44.2 65.1 47.7 47.1 54.9 51.5 57.1 58.4
## 4 92.2 91.9 91.7 90.4 88.8 90.1 91.7 91.8 91.6 92.0 92.2 92.1 92.2
```

This is called **data in wide format**.

Each format has their own advantages and disadvantages. You might consider one over the other depending on the type of statistics you want.

Data transformation

To transform one data format to the other, we use the “tidyr” package.

```
library(tidyr)
```

Wide to long:

```
# gather(column_names..., column_range)
house_l = house_w %>% gather(Year, Rate, 2:23)
head(house_l)
```

```
##                     Property.Type Year Rate
## 1 Condominiums & Other Apartments 1980 49.2
## 2               Landed Properties 1980 75.4
## 3                     Other Types 1980 44.5
## 4                             HDB 1980 61.5
## 5 Condominiums & Other Apartments 1990 78.0
## 6               Landed Properties 1990 87.6
```

```
# negative can be used to exclude columns
house_l = house_w %>% gather(Year, Rate, -Property.Type)
head(house_l)
```

```
##                     Property.Type Year Rate
## 1 Condominiums & Other Apartments 1980 49.2
## 2               Landed Properties 1980 75.4
## 3                     Other Types 1980 44.5
## 4                             HDB 1980 61.5
## 5 Condominiums & Other Apartments 1990 78.0
## 6               Landed Properties 1990 87.6
```

Long to wide:

```
house_w = house_l %>% spread(Year, Rate)
head(house_w)
```

```
##                     Property.Type 1980 1990 1995 2000 2001 2002 2003 2004 2005
## 1 Condominiums & Other Apartments 49.2 78.0 83.3 80.8 82.6 82.6 80.9 80.8 77.2
## 2                             HDB 61.5 89.8 90.6 93.1 94.0 94.1 93.4 94.2 93.0
## 3               Landed Properties 75.4 87.6 92.3 89.8 95.0 92.8 92.2 91.1 89.8
## 4                     Other Types 44.5 41.3 50.1 56.1 64.0 74.1 81.7 65.2 57.1
##   2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018
## 1 77.1 77.1 79.6 77.2 76.6 79.0 80.3 82.5 83.3 84.1 84.1 83.6 84.9
## 2 92.2 91.9 91.7 90.4 88.8 90.1 91.7 91.8 91.6 92.0 92.2 92.1 92.2
## 3 90.1 88.9 89.3 90.2 88.7 88.7 89.5 91.2 90.9 92.0 92.8 92.0 92.8
## 4 62.4 62.4 45.8 51.7 46.8 44.2 65.1 47.7 47.1 54.9 51.5 57.1 58.4
```

Alternative Data

Alternative data is usually non-financial data, from non-traditional sources, used by investment professionals in their evaluation processes.

It is often about people or companies that can be used to gauge some statistics before any official reports are published. It can also supplement current data to give a more holistic view.

Growing Data

- adding new attributes
- adding new records
- merging with new data sets

1. Adding New Attributes

```
No.Of.Children = c(1, 2, 0, 2)
employees_1 = cbind(employees, No.Of.Children)
employees_1
```

```
##         Position Gender Salary No.Of.Children
## James        CEO      M  30000              1
## Joe           VF      M  13000              2
## Janice   Manager      F   9800              0
## Jessica    Clerk      F   3500              2
```

2. Adding new records

```
John = data.frame(Position="Clerk", Salary=2900, Gender="M",
                  No.Of.Children=0)
employees_2 = rbind(employees_1, John=John)
employees_2
```

```
##         Position Gender Salary No.Of.Children
## James        CEO      M  30000              1
## Joe           VF      M  13000              2
## Janice   Manager      F   9800              0
## Jessica    Clerk      F   3500              2
## John       Clerk      M   2900              0
```

What if we have an missing column data or extra column data?

```
# rbind does not work here
Jack = data.frame(Position="Engineer", Salary=2900, Gender="M",
                  Education="B.Sc.")

# use bind_rows from dplyr package
employees_3 = bind_rows(employees_2, Jack=Jack)
employees_3 # fills missing data with NA
```

```
##         Position Gender Salary No.Of.Children Education
## James        CEO      M  30000              1      <NA>
## Joe           VF      M  13000              2      <NA>
## Janice   Manager      F   9800              0      <NA>
## Jessica    Clerk      F   3500              2      <NA>
## John       Clerk      M   2900              0      <NA>
## ...6    Engineer      M   2900             NA     B.Sc.
```

3. Merging Data Sets

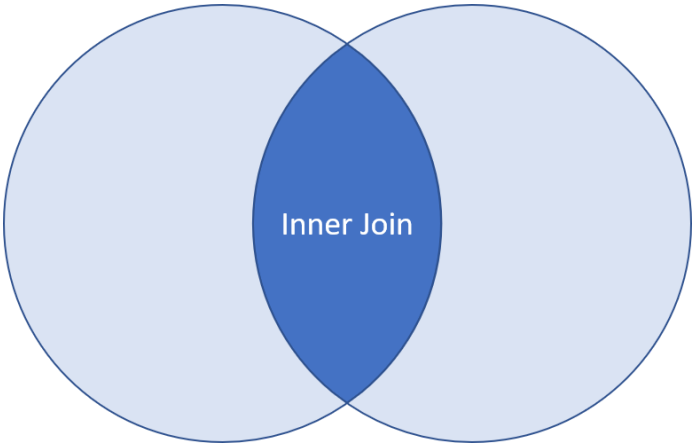
Data relations can be: - 1-to-many (e.g. each country has many companies, each company belongs to 1 country) - 1-to-1 (e.g. each company has 1 CEO, each CEO only in charge of 1 company) - many-to-many (e.g. each company has many customers, each customer associates with many companies)

```
countries = read.csv("./Data/Countries.csv")
head(countries)
```

```
##                Country Population GDP_Per_Capita Life_Expectancy
## 1          Afghanistan   37209007            544            60.5
## 2               Angola   31787566           3669            52.4
## 3              Albania    2938428           5289            77.8
## 4 United Arab Emirates    9682088          40711            77.1
## 5            Argentina   45101781          11627            76.3
## 6              Armenia    2936706           4169            74.8
```

Inner join:

- returns only the rows in companies which have matching country names in countries
- aka only companies both in countries and companies data sets

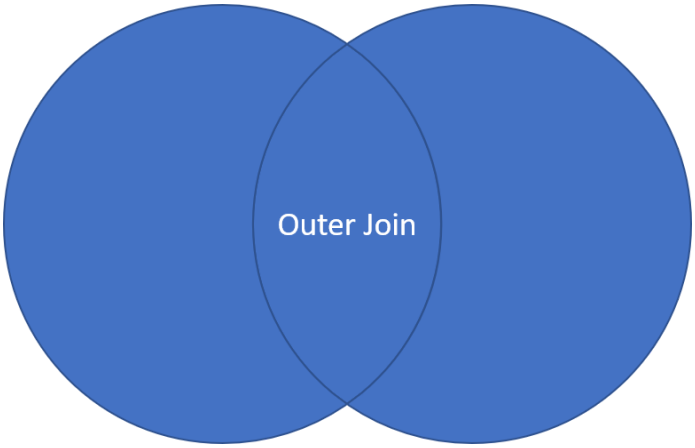



```
set1 = merge(companies, countries, by="Country")
head(set1) # "The Netherlands" dropped, not matching with "Netherlands"
```

```
##     Country ï..Global.Rank                  Company Sales...billion.
## 1 Australia           1833      CFS Retail Property              0.8
## 2 Australia             44             BHP Billiton             72.2
## 3 Australia             58 National Australian Bank             49.2
## 4 Australia           1242               AGL Energy              7.6
## 5 Australia             40        Commonwealth Bank             47.8
## 6 Australia           1487                Crown Ltd              2.8
##   Profits...billion. Assets...billion. Market.Value...billion. Continent
## 1                0.4               8.6                     6.0   Oceania
## 2               15.4             129.3                   184.7   Oceania
## 3                4.2             791.3                    76.3   Oceania
## 4                0.1              14.5                     9.5   Oceania
## 5                7.3             735.2                   117.5   Oceania
## 6                0.5               5.9                     9.4   Oceania
##                  Location Population GDP_Per_Capita Life_Expectancy
## 1 (-25.274398,133.775136)   25088636          56352            82.8
## 2 (-25.274398,133.775136)   25088636          56352            82.8
## 3 (-25.274398,133.775136)   25088636          56352            82.8
## 4 (-25.274398,133.775136)   25088636          56352            82.8
## 5 (-25.274398,133.775136)   25088636          56352            82.8
## 6 (-25.274398,133.775136)   25088636          56352            82.8
```

Outer join:

- returns all rows from both companies and countries, joining records from companies which have matching keys in countries
- aka all countries from companies and contries data sets

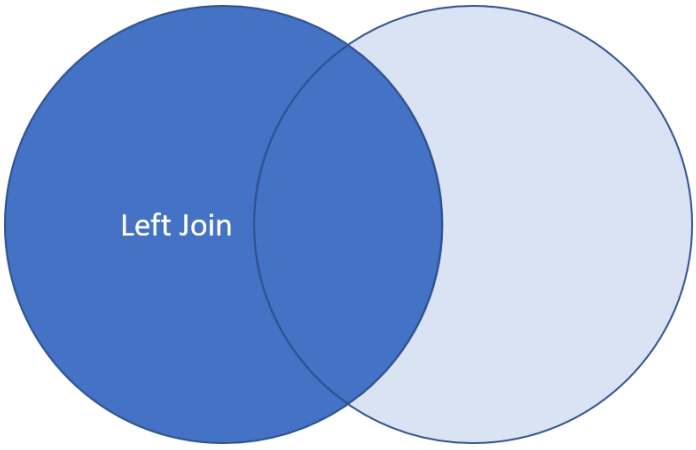


```
set2 = merge(companies, countries, by="Country", all=T)
head(set2) # missing data filled with NA
```

```
##             Country ï..Global.Rank Company Sales...billion. Profits...billion.
## 1       Afghanistan             NA    <NA>               NA                 NA
## 2           Albania             NA    <NA>               NA                 NA
## 3           Algeria             NA    <NA>               NA                 NA
## 4            Angola             NA    <NA>               NA                 NA
## 5 Antigua & Barbuda             NA    <NA>               NA                 NA
## 6         Argentina             NA    <NA>               NA                 NA
##   Assets...billion. Market.Value...billion. Continent Location Population
## 1                NA                     NA      <NA>     <NA>   37209007
## 2                NA                     NA      <NA>     <NA>    2938428
## 3                NA                     NA      <NA>     <NA>   42679018
## 4                NA                     NA      <NA>     <NA>   31787566
## 5                NA                     NA      <NA>     <NA>     104084
## 6                NA                     NA      <NA>     <NA>   45101781
##   GDP_Per_Capita Life_Expectancy
## 1            544            60.5
## 2           5289            77.8
## 3           4238            75.6
## 4           3669            52.4
## 5          17636            76.4
## 6          11627            76.3
```

Left join:

- returns all rows from countries with data from companies wherever there are matching keys
- aka all countries in companies data set

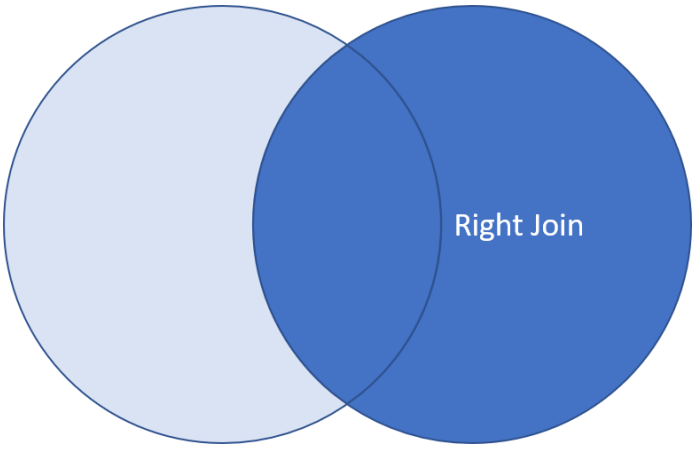



```
set3 = merge(companies, countries, by="Country", all.x=T)
head(set3)
```

```
##     Country ï..Global.Rank                 Company Sales...billion.
## 1 Australia           1833      CFS Retail Property              0.8
## 2 Australia             44             BHP Billiton             72.2
## 3 Australia             58 National Australian Bank             49.2
## 4 Australia           1242               AGL Energy              7.6
## 5 Australia             40        Commonwealth Bank             47.8
## 6 Australia           1487                Crown Ltd              2.8
##   Profits...billion. Assets...billion. Market.Value...billion. Continent
## 1                0.4               8.6                     6.0   Oceania
## 2               15.4             129.3                   184.7   Oceania
## 3                4.2             791.3                    76.3   Oceania
## 4                0.1              14.5                     9.5   Oceania
## 5                7.3             735.2                   117.5   Oceania
## 6                0.5               5.9                     9.4   Oceania
##                  Location Population GDP_Per_Capita Life_Expectancy
## 1 (-25.274398,133.775136)   25088636          56352            82.8
## 2 (-25.274398,133.775136)   25088636          56352            82.8
## 3 (-25.274398,133.775136)   25088636          56352            82.8
## 4 (-25.274398,133.775136)   25088636          56352            82.8
## 5 (-25.274398,133.775136)   25088636          56352            82.8
## 6 (-25.274398,133.775136)   25088636          56352            82.8
```

Right join:

- returns all rows from companies with data from countries wherever there are matching keys
- aka all countries in countries data set




```
set4 = merge(companies, countries, by="Country", all.y=T)
head(set4)
```

```
##             Country ï..Global.Rank Company Sales...billion. Profits...billion.
## 1       Afghanistan             NA    <NA>               NA                 NA
## 2           Albania             NA    <NA>               NA                 NA
## 3           Algeria             NA    <NA>               NA                 NA
## 4            Angola             NA    <NA>               NA                 NA
## 5 Antigua & Barbuda             NA    <NA>               NA                 NA
## 6         Argentina             NA    <NA>               NA                 NA
##   Assets...billion. Market.Value...billion. Continent Location Population
## 1                NA                     NA      <NA>     <NA>   37209007
## 2                NA                     NA      <NA>     <NA>    2938428
## 3                NA                     NA      <NA>     <NA>   42679018
## 4                NA                     NA      <NA>     <NA>   31787566
## 5                NA                     NA      <NA>     <NA>     104084
## 6                NA                     NA      <NA>     <NA>   45101781
##   GDP_Per_Capita Life_Expectancy
## 1            544            60.5
## 2           5289            77.8
## 3           4238            75.6
## 4           3669            52.4
## 5          17636            76.4
## 6          11627            76.3
```

Native R

```
attach(employees) # adds column names to search path
```

```
## The following objects are masked _by_ .GlobalEnv:
##
##     Gender, Position, Salary
```

```
Position
```

```
## [1] "CEO"     "VF"      "Manager" "Clerk"
```

```
detach(employees) # removes column names
# Position        # throws error
```

Lecture 4

```
knitr::opts_knit$set(root.dir = './Data')
```

Importing Local Data

1. CSV

```
head(read.csv("airport.csv"))
```

```
##                          Airport      Country  Latitude Longtitude Type
## 1      Jiuzhai Huanglong Airport        China  32.85333 103.682222    U
## 2                  Navoi Airport   Uzbekistan  40.11720  65.170799    E
## 3           Roitzschjora Airport      Germany  51.57778  12.494444    U
## 4 Bad Neuenahr-Ahrweiler Airport      Germany  50.55778   7.136389    E
## 5                  Parys Airport South Africa -26.88930  27.503401    U
## 6               Iwo Jima Airport        Japan  24.78400 141.322998    U
```

2. TXT

```
read.delim("employees.txt", sep=",") # , is the delimiter
```

```
##    Name Position Date.of.Join   Salary Gender
## 1 James      CEO   2010-01-01 20934.89   Male
## 2   Joe       VP   2011-03-09 13094.54   Male
## 3   May  Manager   2015-08-08  9800.00 Female
## 4 Alice    Clerk   2012-04-05  3500.45 Female
```

```
# assume no header by default
read.table("employees.txt", header=T, sep=",")
```

```
##    Name Position Date.of.Join   Salary Gender
## 1 James      CEO   2010-01-01 20934.89   Male
## 2   Joe       VP   2011-03-09 13094.54   Male
## 3   May  Manager   2015-08-08  9800.00 Female
## 4 Alice    Clerk   2012-04-05  3500.45 Female
```

3. DAT

Other file types that can be viewed by a text editor can also use read.delim and read.table

```
read.delim("employees.txt", sep=",")
```

```
##    Name Position Date.of.Join   Salary Gender
## 1 James      CEO   2010-01-01 20934.89   Male
## 2   Joe       VP   2011-03-09 13094.54   Male
## 3   May  Manager   2015-08-08  9800.00 Female
## 4 Alice    Clerk   2012-04-05  3500.45 Female
```

```
read.table("employees.txt", header=T, sep=",")
```

```
##    Name Position Date.of.Join   Salary Gender
## 1 James      CEO   2010-01-01 20934.89   Male
## 2   Joe       VP   2011-03-09 13094.54   Male
## 3   May  Manager   2015-08-08  9800.00 Female
## 4 Alice    Clerk   2012-04-05  3500.45 Female
```

4. Excel

Excel files can have multiple worksheets. We can use the **readxl** package to read an excel sheet properly

```
library("readxl")
```

```
## Warning: package 'readxl' was built under R version 4.1.1
```

```
# sheet number
read_excel("employees.xlsx", sheet=2)
```

```
## # A tibble: 4 x 5
##   Name  Position `Data of Join`      Salary Gender
##   <chr> <chr>    <dttm>               <dbl> <chr>
## 1 James CEO      2010-01-01 00:00:00 20935. Male
## 2 Joe   VP       2011-03-09 00:00:00 13095. Male
## 3 May   Manager  2015-08-08 00:00:00  9800  Female
## 4 Alice Clerk    2012-04-05 00:00:00  3500. Female
```

```
# sheet name
read_excel("employees.xlsx", sheet="employees")
```

```
## # A tibble: 4 x 5
##   Name  Position `Data of Join`      Salary Gender
##   <chr> <chr>    <dttm>               <dbl> <chr>
## 1 James CEO      2010-01-01 00:00:00 20935. Male
## 2 Joe   VP       2011-03-09 00:00:00 13095. Male
## 3 May   Manager  2015-08-08 00:00:00  9800  Female
## 4 Alice Clerk    2012-04-05 00:00:00  3500. Female
```

Crawling Data from the Internet

Some data on websites are already presented in a table format. To extract these data, we use 2 libraries: **curl** and **XML**.

```
library("curl")
```

```
## Warning: package 'curl' was built under R version 4.1.1
```

```
## Using libcurl 7.64.1 with Schannel
```

```
library("XML")
```

```
## Warning: package 'XML' was built under R version 4.1.1
```

```
# Store the url as a string
theurl = "http://apps.saferoutesinfo.org/legislation_funding/state_apportionment.cfm"

# extract url data
url = curl(theurl)
urldata = readLines(url)

## MUST CLOSE CONNECTION
close(url)

## SLEEP AFTER NUMEROUS CRAWLS
Sys.sleep(1)

# read HTML table
data = readHTMLTable(urldata, stringAsFactors=F)

head(data[[1]])
```

```
##   State\n\t\t\t  \t\t          Actual 2005\n\t\t\t  \t\t
## 1               Alabama                       $1,000,000
## 2                 Alaska                       $1,000,000
## 3                Arizona                       $1,000,000
## 4               Arkansas                       $1,000,000
## 5             California                       $1,000,000
## 6               Colorado                       $1,000,000
##   Actual 2006*\n\t\t\t  \t\t           Actual 2007\n\t\t\t  \t\t
## 1                     $1,313,659                     $1,767,375
## 2                       $990,000                     $1,000,000
## 3                     $1,557,644                     $2,228,590
## 4                       $990,000                     $1,027,338
## 5                    $11,039,310                    $14,832,295
## 6                     $1,254,403                     $1,679,463
##   Actual 2008\n\t\t\t  \t\t          Actual 2009\n\t\t\t  \t\t
## 1                    $2,199,717                     $2,738,816
## 2                    $1,000,000                     $1,000,000
## 3                    $2,896,828                     $3,612,384
## 4                    $1,297,202                     $1,622,447
## 5                   $18,066,131                    $22,580,275
## 6                    $2,119,802                     $2,659,832
##   Actual 2010\n\t\t\t  \t\t          Actual 2011\n\t\t\t  \t\t
## 1                    $2,738,816                     $2,994,316
## 2                    $1,000,000                     $1,554,670
## 3                    $3,612,384                     $3,733,355
## 4                    $1,622,447                     $1,911,273
## 5                   $22,580,275                    $25,976,518
## 6                    $2,659,832                     $3,022,085
##   Actual 2012\n\t\t\t  \t\t          Total\n\t\t  \t\t
## 1                    $2,556,869              $17,309,568
## 2                      $933,567               $8,478,237
## 3                    $3,372,404              $22,013,589
## 4                    $1,514,664              $10,985,371
## 5                   $21,080,209             $137,155,013
## 6                    $2,483,132              $16,878,549
```

What if the page has multiple tables?

```
theurl = "https://en.wikipedia.org/wiki/FIFA_World_Cup"
url = curl(theurl)
urldata = readLines(url)
```

```
## Warning in readLines(url): incomplete final line found on 'https://
## en.wikipedia.org/wiki/FIFA_World_Cup'
```

```
data = readHTMLTable(urldata, stringAsFactors=F)

# data is a list
class(data)
```

```
## [1] "list"
```

```
length(data)
```

```
## [1] 25
```

```
# attendance table
head(data[3][[1]])
```

```
##       V1       V2            V3              V4      V5            V6
## 1   Year    Hosts Venues/Cities Totalattendance Matches Avg.attendance
## 2 Number    Venue       Game(s)            <NA>    <NA>          <NA>
## 3   1930  Uruguay           3/1         590,549      18        32,808
## 4   1934    Italy           8/8         363,000      17        21,353
## 5   1938   France          10/9         375,700      18        20,872
## 6   1950   Brazil           6/6       1,045,246      22        47,511
##                     V7                              V8
## 1 Highest attendances †                           <NA>
## 2                 <NA>                            <NA>
## 3               93,000   Estadio Centenario, Montevideo
## 4               55,000       Stadio Nazionale PNF, Rome
## 5               58,455     Olympique de Colombes, Paris
## 6           173,850[82] Maracanã Stadium, Rio de Janeiro
##                                 V9
## 1                             <NA>
## 2                             <NA>
## 3 Uruguay 6–1 Yugoslavia, Semi-final
## 4    Italy 2–1 Czechoslovakia, Final
## 5    France 1–3 Italy, Quarter-final
## 6 Brazil 1–2 Uruguay, Deciding match
```

HTML

- Hypertext Markup Language
- designed to display data
- tags must be pre-defined

XML

- Extensible Markup Language
- designed to stroe and transport data

```
<?xml version"1.0"?>
# <startingTag>XML text</endingTag>
<person>
  <firstName>Lilian</fistName>
  <lastName>TeO</lastName>
  <age>42</age>
  <spouse></spouse>

  # tree like structure. Each tag is a node.
  <children>

    # gender attribute in the tag itself
    <child gender="male">
      <name>Melvyn</name>
      <age>12</age>
    </child>

    <child gender="female">
      <name>Joan</name>
      <age>8</age>
    </child>

  </children>
</person>
```

Data in XML format can be read and explored using the **XML** library.

```
library("XML")

# read data
data = xmlParse("books.xml")

# get nodes
root = xmlRoot(data)
nodes = xmlChildren(root)
books = xmlChildren(nodes[[2]])

books[1]
```

```
## $book
## <book id="bk101" type="HardCover">
##   <author>Gambardella, Matthew</author>
##   <title>XML Developer's Guide</title>
##   <genre>Computer</genre>
##   <price>44.95</price>
##   <publish_date>2000-10-01</publish_date>
##   <description>An in-depth look at creating applications
##         with XML.</description>
## </book>
```

However, this method of exploring the data is tedious. If we know the structure of the xml data set, we can use the tag/attribute to search for the data we want.

```
books = getNodeSet(data,"/library/catalog/book[@type='HardCover']")

books[1]
```

```
## [[1]]
## <book id="bk101" type="HardCover">
##   <author>Gambardella, Matthew</author>
##   <title>XML Developer's Guide</title>
##   <genre>Computer</genre>
##   <price>44.95</price>
##   <publish_date>2000-10-01</publish_date>
##   <description>An in-depth look at creating applications
##         with XML.</description>
## </book>
```

We can also convert the data into other formats

```
# list
Lib = xmlToList(data)
Lib
```

```
## $location
## [1] "HSSML"
##
## $catalog
## $catalog$book
## $catalog$book$author
## [1] "Gambardella, Matthew"
##
## $catalog$book$title
## [1] "XML Developer's Guide"
##
## $catalog$book$genre
## [1] "Computer"
##
## $catalog$book$price
## [1] "44.95"
##
## $catalog$book$publish_date
## [1] "2000-10-01"
##
## $catalog$book$description
## [1] "An in-depth look at creating applications \n        with XML."
##
## $catalog$book$.attrs
##          id        type
##     "bk101" "HardCover"
##
##
## $catalog$book
## $catalog$book$author
## [1] "Ralls, Kim"
##
## $catalog$book$title
## [1] "Midnight Rain"
##
## $catalog$book$genre
## [1] "Fantasy"
##
## $catalog$book$price
## [1] "5.95"
##
## $catalog$book$publish_date
## [1] "2000-12-16"
##
## $catalog$book$description
## [1] "A former architect battles corporate zombies, \n        an evil sorceress, and her ow
n childhood to become queen \n        of the world."
##
## $catalog$book$.attrs
##          id        type
##     "bk102" "HardCover"
##
##
## $catalog$book
## $catalog$book$author
## [1] "Corets, Eva"
##
```

```
## $catalog$book$title
## [1] "Maeve Ascendant"
##
## $catalog$book$genre
## [1] "Fantasy"
##
## $catalog$book$price
## [1] "5.95"
##
## $catalog$book$publish_date
## [1] "2000-11-17"
##
## $catalog$book$description
## [1] "After the collapse of a nanotechnology \n        society in England, the young surviv
ors lay the \n        foundation for a new society."
##
## $catalog$book$.attrs
##          id        type
##     "bk103" "HardCover"
##
##
## $catalog$book
## $catalog$book$author
## [1] "Corets, Eva"
##
## $catalog$book$title
## [1] "Oberon's Legacy"
##
## $catalog$book$genre
## [1] "Fantasy"
##
## $catalog$book$price
## [1] "5.95"
##
## $catalog$book$publish_date
## [1] "2001-03-10"
##
## $catalog$book$description
## [1] "In post-apocalypse England, the mysterious \n        agent known only as Oberon helps
to create a new life \n        for the inhabitants of London. Sequel to Maeve \n        Ascen
dant."
##
## $catalog$book$.attrs
##          id        type
##     "bk104" "SoftCover"
##
##
## $catalog$book
## $catalog$book$author
## [1] "Corets, Eva"
##
## $catalog$book$title
## [1] "The Sundered Grail"
##
## $catalog$book$genre
## [1] "Fantasy"
##
## $catalog$book$price
```

```
## [1] "5.95"
##
## $catalog$book$publish_date
## [1] "2001-09-10"
##
## $catalog$book$description
## [1] "The two daughters of Maeve, half-sisters, \n        battle one another for control of
England. Sequel to \n        Oberon's Legacy."
##
## $catalog$book$.attrs
##          id        type
##     "bk105" "HardCover"
##
##
## $catalog$book
## $catalog$book$author
## [1] "Randall, Cynthia"
##
## $catalog$book$title
## [1] "Lover Birds"
##
## $catalog$book$genre
## [1] "Romance"
##
## $catalog$book$price
## [1] "4.95"
##
## $catalog$book$publish_date
## [1] "2000-09-02"
##
## $catalog$book$description
## [1] "When Carla meets Paul at an ornithology \n        conference, tempers fly as feathers
get ruffled."
##
## $catalog$book$.attrs
##          id        type
##     "bk106" "SoftCover"
##
##
## $catalog$book
## $catalog$book$author
## [1] "Thurman, Paula"
##
## $catalog$book$title
## [1] "Splish Splash"
##
## $catalog$book$genre
## [1] "Romance"
##
## $catalog$book$price
## [1] "4.95"
##
## $catalog$book$publish_date
## [1] "2000-11-02"
##
## $catalog$book$description
## [1] "A deep sea diver finds true love twenty \n        thousand leagues beneath the sea."
##
```

```
## $catalog$book$.attrs
##          id        type
##     "bk107" "HardCover"
##
##
## $catalog$book
## $catalog$book$author
## [1] "Knorr, Stefan"
##
## $catalog$book$title
## [1] "Creepy Crawlies"
##
## $catalog$book$genre
## [1] "Horror"
##
## $catalog$book$price
## [1] "4.95"
##
## $catalog$book$publish_date
## [1] "2000-12-06"
##
## $catalog$book$description
## [1] "An anthology of horror stories about roaches,\n        centipedes, scorpions  and oth
er insects."
##
## $catalog$book$.attrs
##          id        type
##     "bk108" "SoftCover"
##
##
## $catalog$book
## $catalog$book$author
## [1] "Kress, Peter"
##
## $catalog$book$title
## [1] "Paradox Lost"
##
## $catalog$book$genre
## [1] "Science Fiction"
##
## $catalog$book$price
## [1] "6.95"
##
## $catalog$book$publish_date
## [1] "2000-11-02"
##
## $catalog$book$description
## [1] "After an inadvertant trip through a Heisenberg\n        Uncertainty Device, James Sal
way discovers the problems \n        of being quantum."
##
## $catalog$book$.attrs
##          id        type
##     "bk109" "SoftCover"
##
##
## $catalog$book
## $catalog$book$author
## [1] "O'Brien, Tim"
```

```
##
## $catalog$book$title
## [1] "Microsoft .NET: The Programming Bible"
##
## $catalog$book$genre
## [1] "Computer"
##
## $catalog$book$price
## [1] "36.95"
##
## $catalog$book$publish_date
## [1] "2000-12-09"
##
## $catalog$book$description
## [1] "Microsoft's .NET initiative is explored in \n        detail in this deep programmer's
reference."
##
## $catalog$book$.attrs
##          id        type
##     "bk110" "HardCover"
##
##
## $catalog$book
## $catalog$book$author
## [1] "O'Brien, Tim"
##
## $catalog$book$title
## [1] "MSXML3: A Comprehensive Guide"
##
## $catalog$book$genre
## [1] "Computer"
##
## $catalog$book$price
## [1] "36.95"
##
## $catalog$book$publish_date
## [1] "2000-12-01"
##
## $catalog$book$description
## [1] "The Microsoft MSXML3 parser is covered in \n        detail, with attention to XML DOM
interfaces, XSLT processing, \n        SAX and more."
##
## $catalog$book$.attrs
##          id        type
##     "bk111" "HardCover"
##
##
## $catalog$book
## $catalog$book$author
## [1] "Galos, Mike"
##
## $catalog$book$title
## [1] "Visual Studio 7: A Comprehensive Guide"
##
## $catalog$book$genre
## [1] "Computer"
##
## $catalog$book$price
```

```
## [1] "49.95"
##
## $catalog$book$publish_date
## [1] "2001-04-16"
##
## $catalog$book$description
## [1] "Microsoft Visual Studio 7 is explored in depth,\n        looking at how Visual Basic,
Visual C++, C#, and ASP+ are \n        integrated into a comprehensive development \n
environment."
##
## $catalog$book$.attrs
##          id        type
##     "bk112" "SoftCover"
```

```
# data frame
Books = xmlToDataFrame(books)
head(Books)
```

```
##                 author                                    title    genre price
## 1 Gambardella, Matthew                 XML Developer's Guide Computer 44.95
## 2           Ralls, Kim                         Midnight Rain  Fantasy  5.95
## 3          Corets, Eva                       Maeve Ascendant  Fantasy  5.95
## 4          Corets, Eva                    The Sundered Grail  Fantasy  5.95
## 5       Thurman, Paula                          Splish Splash  Romance  4.95
## 6         O'Brien, Tim Microsoft .NET: The Programming Bible Computer 36.95
##   publish_date
## 1   2000-10-01
## 2   2000-12-16
## 3   2000-11-17
## 4   2001-09-10
## 5   2000-11-02
## 6   2000-12-09
##
description
## 1                                                                          An in-depth l
ook at creating applications \n        with XML.
## 2 A former architect battles corporate zombies, \n        an evil sorceress, and her own c
hildhood to become queen \n        of the world.
## 3 After the collapse of a nanotechnology \n        society in England, the young survivors
lay the \n        foundation for a new society.
## 4      The two daughters of Maeve, half-sisters, \n        battle one another for control
of England. Sequel to \n        Oberon's Legacy.
## 5                                                    A deep sea diver finds true love tw
enty \n        thousand leagues beneath the sea.
## 6                                          Microsoft's .NET initiative is explored in \n
detail in this deep programmer's reference.
```

Efficient Data Importing

Why is CSV preferred over XLSX in the data science community? - More efficient in storing large amounts of data - can be used across platforms without reliance on Microsoft Excel

However, since there is no real size limit for CSV files, they can get very big!

When importing, the class for each column is selected automatically. However, such an assignment could be undesired (e.g. zipcode is better as a character/factor). It is also a lot slower for R to go through every line in order to determine the best class for that column.

```
care.data = read.csv("hospital-data.csv", stringsAsFactors=T)

# overview
str(care.data)
```

```
## 'data.frame':    4826 obs. of  13 variables:
##  $ Provider.Number   : Factor w/ 4826 levels "010001","010005",..: 1 2 3 4 5 6 7 8 9 10
...
##  $ Hospital.Name     : Factor w/ 4624 levels "ABBEVILLE AREA MEDICAL CENTER",..: 3674 2203
1068 2564 857 1482 2202 3918 929 3699 ...
##  $ Address.1         : Factor w/ 4800 levels " 1200 EAST PECAN ST",..: 459 2103 1726 3926
228 1681 4182 3224 1582 2642 ...
##  $ Address.2         : logi  NA NA NA NA NA NA ...
##  $ Address.3         : logi  NA NA NA NA NA NA ...
##  $ City              : Factor w/ 3018 levels "ABBEVILLE","ABERDEEN",..: 723 278 899 1973 1
560 1140 1099 250 929 2679 ...
##  $ State             : Factor w/ 56 levels "AK","AL","AR",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ ZIP.Code          : int  36301 35957 35631 36467 36049 35640 35976 35235 35968 36784
...
##  $ County            : Factor w/ 1533 levels "","ABBEVILLE",..: 652 861 785 347 356 939 86
1 695 388 297 ...
##  $ Phone.Number      : num  3.35e+09 2.57e+09 2.57e+09 3.34e+09 3.34e+09 ...
##  $ Hospital.Type     : Factor w/ 4 levels "ACUTE CARE - VETERANS ADMINISTRATION",..: 2 2 2
2 2 2 2 2 2 2 ...
##  $ Hospital.Ownership: Factor w/ 14 levels "Government-Federal",..: 5 5 5 14 9 9 5 14 9 9
...
##  $ Emergency.Services: Factor w/ 3 levels "No","Not Available",..: 3 3 3 3 3 2 3 3 3 2 ...
```

```
# time taken
system.time(care.data<-read.csv("hospital-data.csv"))
```

```
##    user  system elapsed
##    0.05    0.00    0.04
```

Instead, we can first view the first few rows to get a feel for the layout of the data.

Then, we can pre-set the column classes, **colclasses* of the data

```
# view first few rows
data.sample = read.csv("hospital-data.csv", nrows=5)
data.sample
```

```
##   Provider.Number                    Hospital.Name                  Address.1
## 1           10001 SOUTHEAST ALABAMA MEDICAL CENTER     1108 ROSS CLARK CIRCLE
## 2           10005    MARSHALL MEDICAL CENTER SOUTH 2505 U S HIGHWAY 431 NORTH
## 3           10006   ELIZA COFFEE MEMORIAL HOSPITAL         205 MARENGO STREET
## 4           10007         MIZELL MEMORIAL HOSPITAL              702 N MAIN ST
## 5           10008      CRENSHAW COMMUNITY HOSPITAL        101 HOSPITAL CIRCLE
##   Address.2 Address.3     City State ZIP.Code     County Phone.Number
## 1        NA        NA   DOTHAN    AL    36301    HOUSTON   3347938701
## 2        NA        NA     BOAZ    AL    35957   MARSHALL   2565938310
## 3        NA        NA FLORENCE    AL    35631 LAUDERDALE   2567688400
## 4        NA        NA      OPP    AL    36467  COVINGTON   3344933541
## 5        NA        NA  LUVERNE    AL    36049   CRENSHAW   3343353374
##          Hospital.Type                           Hospital.Ownership
## 1 Acute Care Hospitals Government - Hospital District or Authority
## 2 Acute Care Hospitals Government - Hospital District or Authority
## 3 Acute Care Hospitals Government - Hospital District or Authority
## 4 Acute Care Hospitals              Voluntary non-profit - Private
## 5 Acute Care Hospitals                                 Proprietary
##   Emergency.Services
## 1                Yes
## 2                Yes
## 3                Yes
## 4                Yes
## 5                Yes
```

```
# preset column class
colclass = c("character", "character", "character",
             "character", "character",
             "character", "factor", "factor", "factor",
             "character", "factor", "factor", "factor")

# slight drop in time
system.time(care.data<-read.csv("hospital-data.csv",
                                colClasses=colclass))
```

```
##    user  system elapsed
##    0.11    0.00    0.12
```

Data from API

Application Programming Interface (API) is a software intermediary that allows communication between 2 applications.It is a programming standard agreed and used by 2 parties to exchange data.

CSV files only store static data. However, some data, collected over time and stored in databases, are dynamic.

JSON

- lightweight data interchange format
- language independent
- standard accepted by all programming languages

e.g.

```
{
  # "fieldName": fieldValue,
  "firstName": "Lilian",
  "lastName": "Teo",
  "age": 42,
  "spouse": NULL,
  "children": [ #list of JSON objects
    {
      "name": "Melvyn",
      "age": 12
    },
    {
      "name": "Joan",
      "age": 8
    }
  ]
}
```

In order to read data in the JSON format, we can use **jsonlite**.

```
library("jsonlite")
```

```
## Warning: package 'jsonlite' was built under R version 4.1.1
```

```
# API url
url = "https://api.data.gov.sg/v1/transport/carpark-availability"

# read data
data = fromJSON(url)

# fields contained in the data
head(as.data.frame(data$items$carpark_data))
```

```
##   carpark_info carpark_number     update_datetime
## 1   105, C, 17           HE12 2021-10-02T15:58:27
## 2  583, C, 367            HLM 2021-10-02T15:58:23
## 3  329, C, 148            RHM 2021-10-02T15:58:27
## 4    97, C, 77           BM29 2021-10-02T15:58:02
## 5    96, C, 35            Q81 2021-10-02T15:58:05
## 6    172, C, 1            C20 2021-10-02T15:58:11
```

Missing Data

Supplementing with Other Information

Data can be inter-related between different columns/records.

```
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
properties = read.csv("properties.csv")

# Developer is missing
properties %>%
  filter(Property.Name == "Le Quest") %>%
  .[c("Property.Name", "Developer")]
```

```
##   Property.Name                     Developer
## 1      Le Quest                          <NA>
## 2      Le Quest Qingjian Realty (BBC) Pte Ltd
```

```
# can take the developer from the other property of the same name
developer = properties %>%
  filter(Property.Name == "Le Quest" &
           !is.na(Developer)) %>% .$Developer %>% head(1)

properties[properties$Property.Name == "Le Quest", "Developer"] = developer

# result
properties %>%
  filter(Property.Name == "Le Quest") %>%
  .[c("Property.Name", "Developer")]
```

```
##   Property.Name                     Developer
## 1      Le Quest Qingjian Realty (BBC) Pte Ltd
## 2      Le Quest Qingjian Realty (BBC) Pte Ltd
```

Numeric

1. Convert numeric data into categorical data

- NA values converted to an “Unknown” category

2. Replace the data with value 0

- e.g. assume people with missing children information don’t have children
- assumptions may or may not be valid

3. Replace the data with mean value

- as to not overly bias the data when handling missing values

4. Calibrate the data and estimate the missing value

- e.g. property price is affected by type, location, area. We can try to find the mean price per square feet of similar properties to the one we are calibrating. Use the PSF to estimate the property price

Other Data Problems

1. Data entry problem
2. Logical error
3. Outdated
4. Different standard

```
## MAKE SURE TO CLOSE
# closeAllConnections()
```

Lecture 5

Conditional Statements

```
x = 0

if (x > 0) {
  x = 10
} else if (x < -2) {
  x = 20
} else {
  x = 40
}

x # -2<x<0, returns 40
```

```
## [1] 40
```

```
ifelse(x<30, "Low", ifelse(x>50, "High", "Medium"))
```

```
## [1] "Medium"
```

For Loops

```
sum = 0
a = c()
for (i in seq(0, 100, 5)) {
  sum = sum + i
  a = append(a, sum)
}

a
```

```
##  [1]    0    5   15   30   50   75  105  140  180  225  275  330  390  455  525
## [16]  600  680  765  855  950 1050
```

```
properties_data = read.csv("./Data/properties.csv")

nrows = nrow(properties_data)

for (i in 1:nrows) {
  # get the ith property
  property = properties_data[i, ]

  # if developer is missing
  if (is.na(property$Developer)) {
    # get other properties with the same name
    idx = properties_data$Property.Name == property$Property.Name &
      !is.na(properties_data$Developer)
    # if they have a developer
    if (sum(idx) > 0) {
      # use data to fill in missing
      property$Developer = properties_data[idx, "Developer"][[1]]
    }
  }
}
```

While Loops

```
fibs = c(1, 1)
i = 1; j = 2

while (fibs[i] + fibs[j] < 1000) { # good for uncertain ranges
  fibs[j+1] = fibs[i] + fibs[j]
  i = i + 1
  j = j + 1
}

fibs
```

```
##  [1]   1   1   2   3   5   8  13  21  34  55  89 144 233 377 610 987
```

Using break to escape the while loop:

```
fibs = c(1, 1)
i = 1; j = 2

while (T) { # good for uncertain ranges
  fibs[j+1] = fibs[i] + fibs[j]
  i = i + 1
  j = j + 1
  if (fibs[i] + fibs[j] >= 1000) { # be careful; can lead to infinite looping
    break
  }
}

fibs
```

```
##  [1]   1   1   2   3   5   8  13  21  34  55  89 144 233 377 610 987
```

Functions

Define a function using:

```
function_name = function(inputs) { # can have zero inputs
  actions
  return(output) # optional, returns the last line by default
}
```

```
# will give the tightest fibonacci series from start to stop inclusive
generate_fib = function(stop, start=1) {
  fib = c(1, 1)

  i = 1; j = 2;
  while (fib[i] + fib[j] <= stop) {
    fib[j+1] = fib[i] + fib[j]
    i = i + 1
    j = j + 1
  }

  fib = fib[fib >= start & fib <= stop]
  return(fib)
}

generate_fib(1000, start=10)
```

```
##  [1]  13  21  34  55  89 144 233 377 610 987
```

Apply Functions

- allows application of a function to a margin of a matrix or data frame without the use of explicit looping
- iterates across rows or columns more efficiently
- apply(), sapply(), mapply(), lapply(), vapply(), rapply(), tapply()

Apply

```
ID = c(3000, 1234, 2000)
start = c(5, 50, 0)
end = c(100, 100, 0)

m = cbind(ID, start, end)

# MARGIN = 1 performs operations across rows
# MARGIN = 2 performs operations along columns
# FUNC must be able to accept a variable as its input
apply(m, 2, sum)
```

```
##    ID start   end
##  6234    55   200
```

Can apply user-defined functions too:

```
CountOdd = function(x) {
  return(sum(x%%2))
}

apply(m, 2, CountOdd)
```

```
##    ID start   end
##     0     1     0
```

```
CountOdderEven = function(x, flag) {
  if(flag) {
    return(sum(x%%2)) # flag true, count odd
  } else {
    return(length(x) - sum(x%%2)) # flag false, count even
  }
}

apply(m, 2, CountOdderEven, F) # need to supply the flag variable
```

```
##    ID start   end
##     3     2     3
```

Other apply functions work similarly but on more specific data structures:

2. Lapply

- input is a list of vectors
- returns a list of the same length as the input

```
x = list(A=1:4, B=seq(0.1, 1, by=0.1))

lapply(x, mean)
```

```
## $A
## [1] 2.5
##
## $B
## [1] 0.55
```

3. Sapply

- A wrapper function of lapply
- returns a vector instead of a list

```
x = list(A=1:4, B=seq(0.1, 1, by=0.1))

sapply(x, mean)
```

```
##    A    B
## 2.50 0.55
```

4. Rapply

- recursive apply
- good for lists of lists

```
x = list(A=2, B=list(-1, 3), c=list(-2, list(-5, 6)))

rapply(x, function(x){x^2})
```

```
##  A B1 B2 c1 c2 c3
##  4  1  9  4 25 36
```

5. Mapply

- some functions are multivariate
- takes multiple vectors as inputs
- applies functions to combinations of elements from each vector

```
mapply(rep, 1:5, c(4,4,4,4,4))
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    1    2    3    4    5
## [3,]    1    2    3    4    5
## [4,]    1    2    3    4    5
```

```
## 1 3 1
## 2 4 2
A = matrix(c(1,2,3,4), ncol=2)
## 1 2 3
## 4 5 6
B = matrix(c(1,2,3,4,5,6), ncol=2, byrow=T)
A
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
B
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
```

```
mapply(rep, A, B)
```

```
## Warning in mapply(rep, A, B): longer argument not a multiple of length of
## shorter
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2 2 2
##
## [[3]]
## [1] 3 3 3 3 3
##
## [[4]]
## [1] 4 4
##
## [[5]]
## [1] 1 1 1 1
##
## [[6]]
## [1] 2 2 2 2 2 2
```

6. Tapply

- applies function to each group of an array, grouped by certain factors

```
x = 1:10
y = factor(c("A", "A", "A", "B", "B", "B", "B", "C", "C", "C"))

x
```

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

```
y
```

```
##  [1] A A A B B B B C C C
## Levels: A B C
```

```
tapply(x, y, sum)
```

```
##  A  B  C
##  6 22 27
```

Pivot Table

```
# total murders per region
murders = read.csv("./Data/murders.csv")
tapply(murders$total, murders$region, sum)
```

```
## North Central     Northeast         South          West
##          1828          1469          4195          1911
```

```
# divides population into intervals, gets mean
bins = cut(murders$population, breaks=c(0, 1e+06, 1e+07, 1e+08))
tapply(murders$population, bins, mean)
```

```
##     (0,1e+06] (1e+06,1e+07] (1e+07,1e+08]
##      734430.1     4595944.4    19790683.9
```

Split

- splits data frame into a list of data frames by a factor array

```
# ratio of murder cases to population by region
## total by region
total = tapply(murders$total, murders$region, sum)
## population by region
pop = tapply(murders$population, murders$region, sum)
total/pop
```

```
## North Central     Northeast         South          West
##  2.731334e-05  2.655592e-05  3.626558e-05  2.656175e-05
```

```
# DONE USING SPLIT
region = split(murders, murders$region)
sapply(region, function(x){sum(x$total)/sum(x$population)})
```

```
## North Central     Northeast         South          West
##  2.731334e-05  2.655592e-05  3.626558e-05  2.656175e-05
```

Group by Multiple Factors

Use a list for the aggregating factor:

```
head(mtcars) # built in dataset
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
# Ave. h-p by a-m and gear
# am becomes the rows, gear becomes the columns
tapply(mtcars$mpg, INDEX=list(mtcars$am, mtcars$gear), mean)
```

```
##          3      4     5
## 0 16.10667 21.050    NA
## 1       NA 26.275 21.38
```

Lecture 7

Data Visualisation

Types of Data Visualisation

1. Idea illustration
2. Idea generation
3. Everyday data visualisation
4. Visual discovery

- Trendspotting
- Sense-making
- Deep analysis

Principles of Data Visualisation

1. Simplify

- captures only the essence of the data

2. Compare

- compare visualisations side-by-side
- allow audience to spot trends, patterns and differences by sight

3. Attend

- highlight details that audience needs to attend to

4. Explore

- Allow audience to explore data and discover things by sight

5. View Diversely

- look at the same data from different perspectives at the same time
- learn how the data fits together

6. Ask Why

- allow the audience to dig into the data to find out why things are happening

7. Be Skeptical

- don’t be contented with the first answer we get
- always explore further

8. Respond

- share the data to allow others to make use of it and to check it

ggplot2

1. Layer

- allow combine graphs in different layers to make one complex graph

2. Scale

- maps values in the data space to values in the aesthetic space (e.g. color, size, shape)

3. Coordinate system

- how points are positioned in space

4. Faceting

- split the data into subsets of the entire data set

```
library("ggplot2")
countries = read.csv("./Data/Countries.csv")
head(countries)
```

```
##                Country Population GDP_Per_Capita Life_Expectancy
## 1          Afghanistan   37209007            544            60.5
## 2               Angola   31787566           3669            52.4
## 3              Albania    2938428           5289            77.8
## 4 United Arab Emirates    9682088          40711            77.1
## 5            Argentina   45101781          11627            76.3
## 6              Armenia    2936706           4169            74.8
```

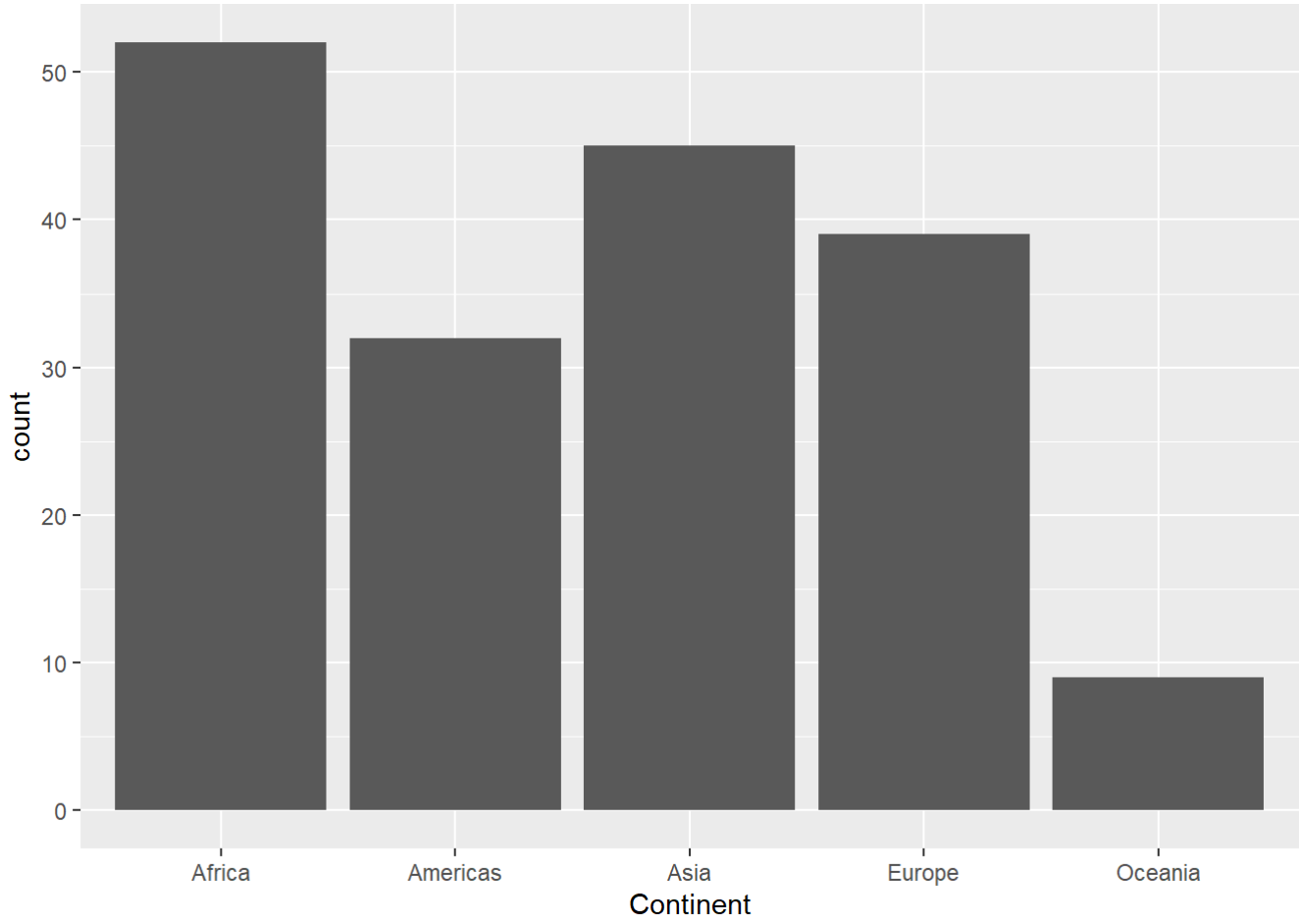
Plotting

Comparing the number of countries among different continents:

```
library("countrycode")
```

```
## Warning: package 'countrycode' was built under R version 4.1.1
```

```
countries$Continent <- countrycode(sourcevar = countries[, "Country"],
                           origin = "country.name",
                           destination = "continent")

ggplot(data=countries,aes(x=Continent)) + geom_bar()
```



aes means aesthetics and determines the look of the graph. Here, we are specifying that the Continent column is mapped to the x-axis of the layer. **geom_bar** function means that we are expecting the plot to be a bar chart.

This is only using a few of the possible components of a layer. Each geom function has its own statistical transformation (in this case, count) and position by default.

We can change this transformation using **stat**.We can also colour the bars using **fill** and specify bar dimensions using **width**.

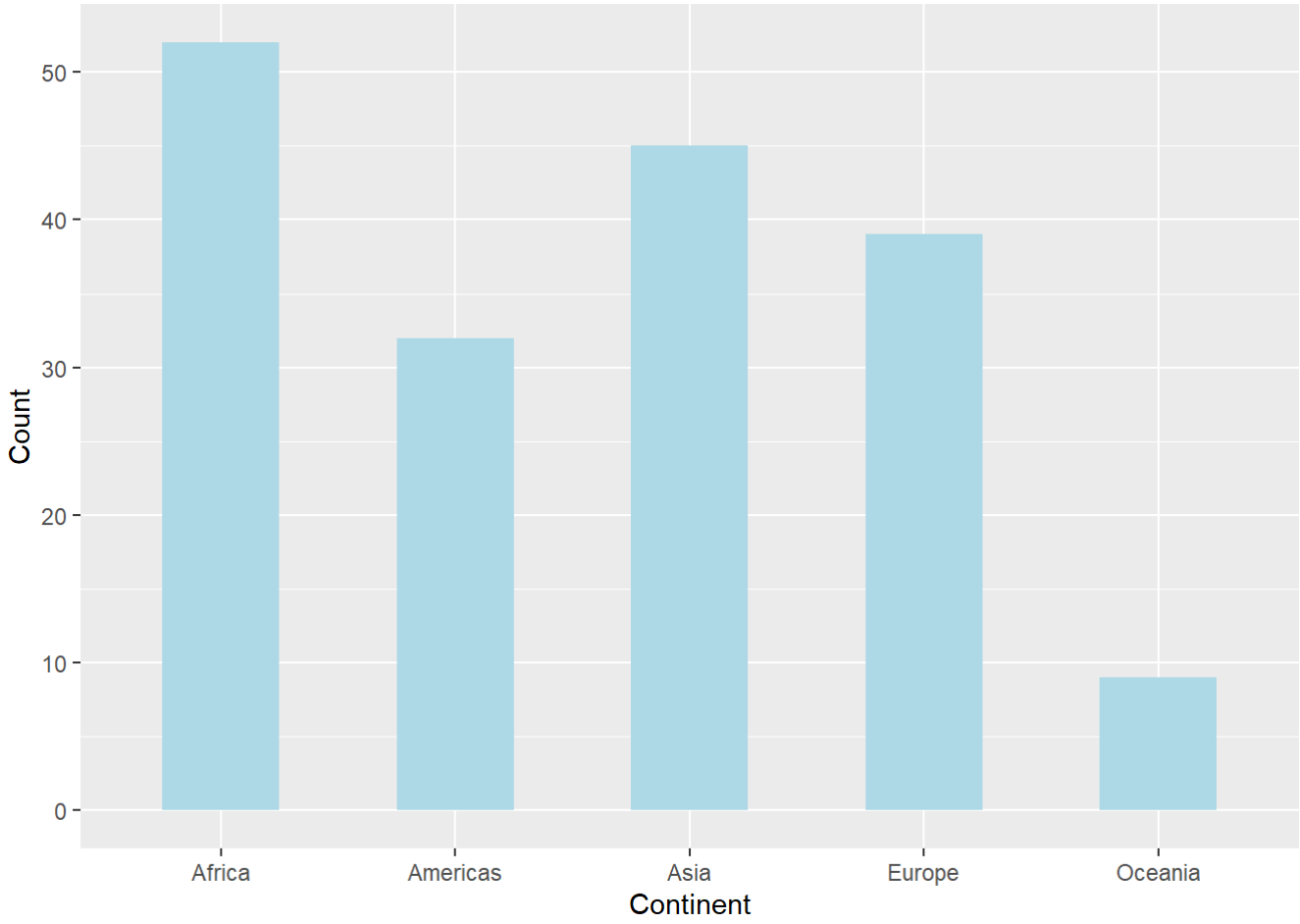
```
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

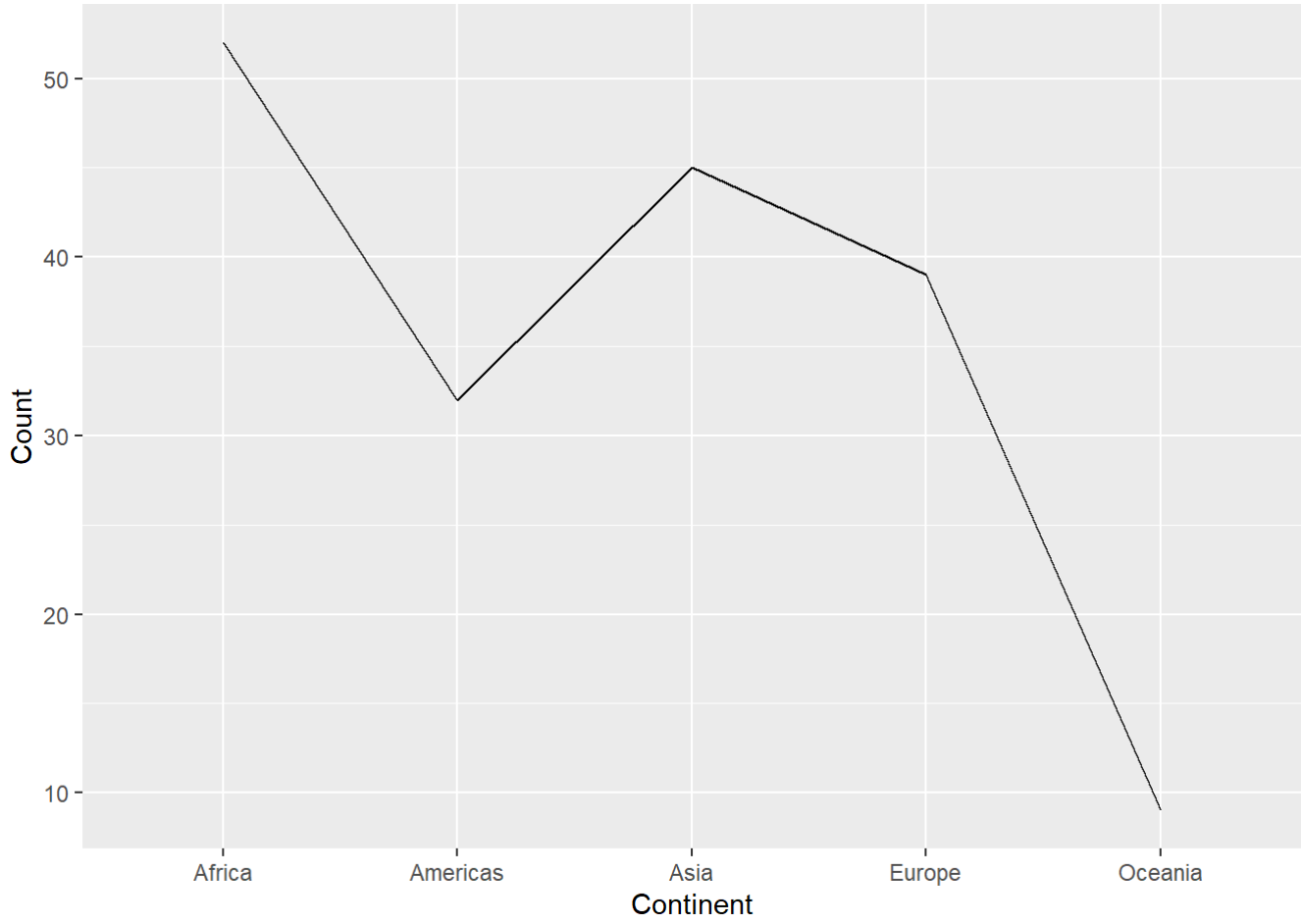
```
df = countries %>% group_by(Continent) %>% summarize(Count=n())

ggplot(data=df,aes(x=Continent, y=Count)) + geom_bar(stat="identity", fill="lightblue", width
=0.5)
```



Overlay Layers

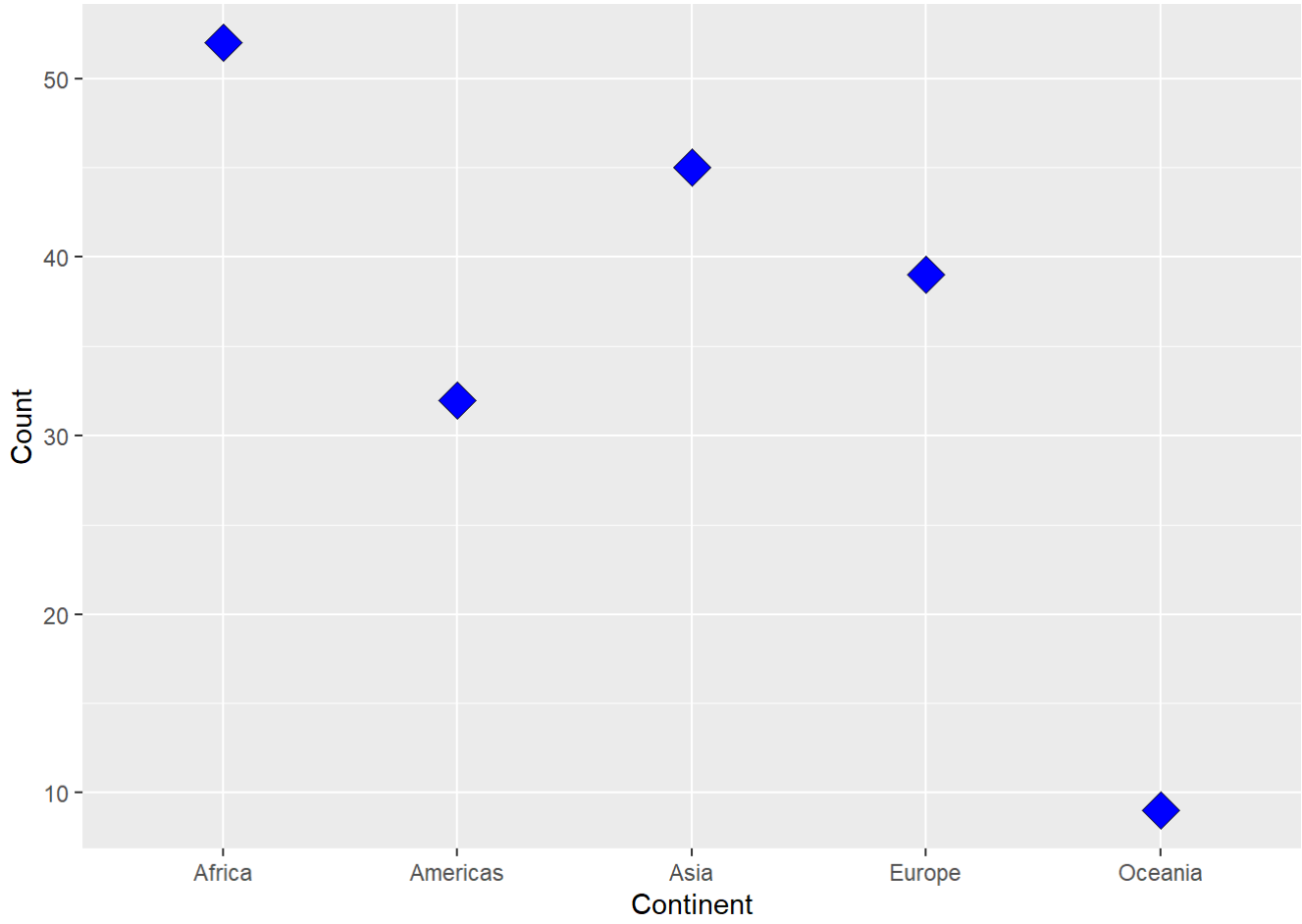
For a line plot, we use **geom_line**. Note that geom_line connects lines in the same group. Hence, we need to specify **group=1**.

```
ggplot(data=df, aes(x=Continent, y=Count)) + geom_line(group=1)
```



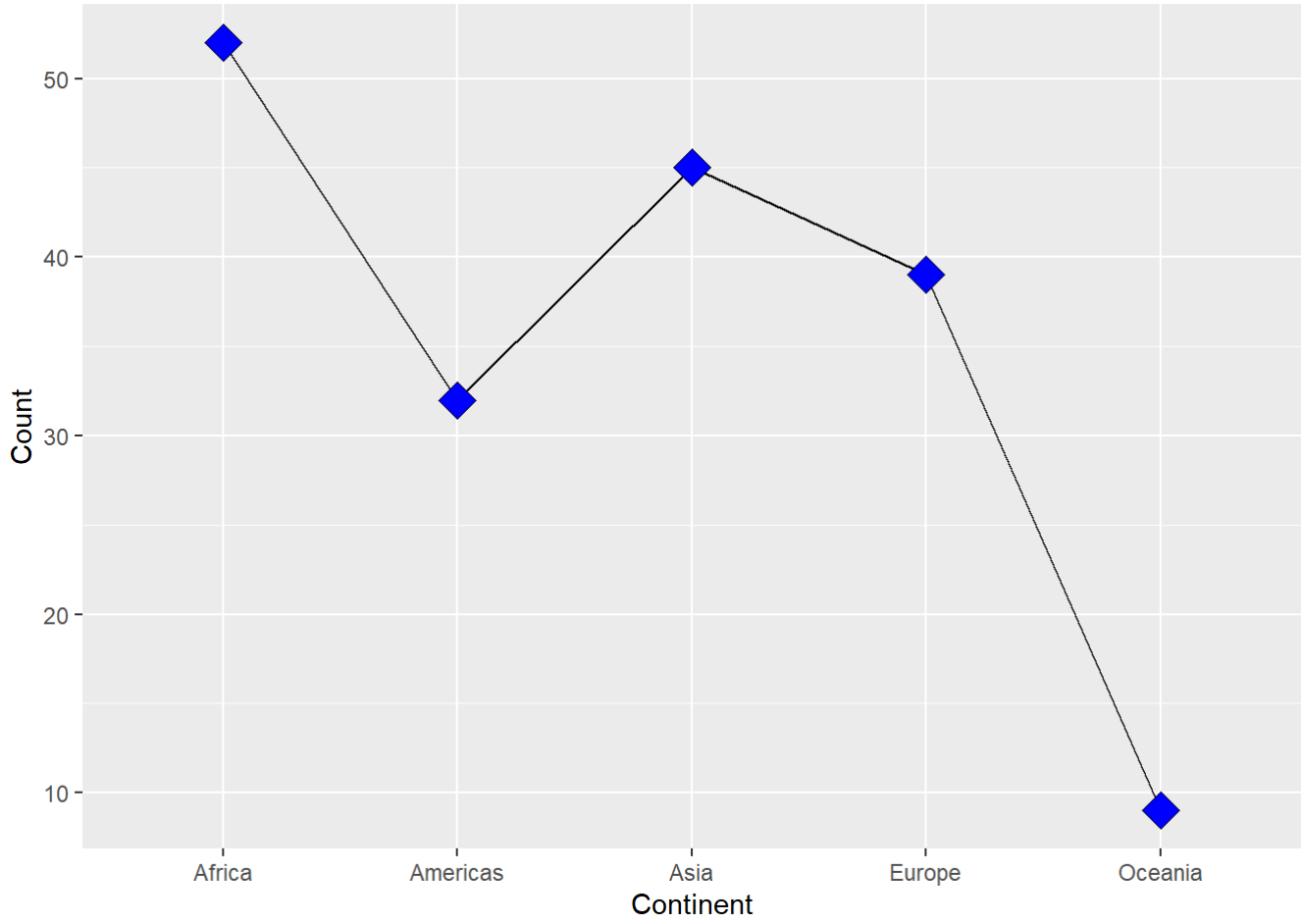
We can use **geom_point** for a scatter plot and use **size**, **shape**, and fill to specify the look of our points.

```
ggplot(data=df, aes(x=Continent, y=Count)) + geom_point(size=5, shape=23, fill="blue")
```



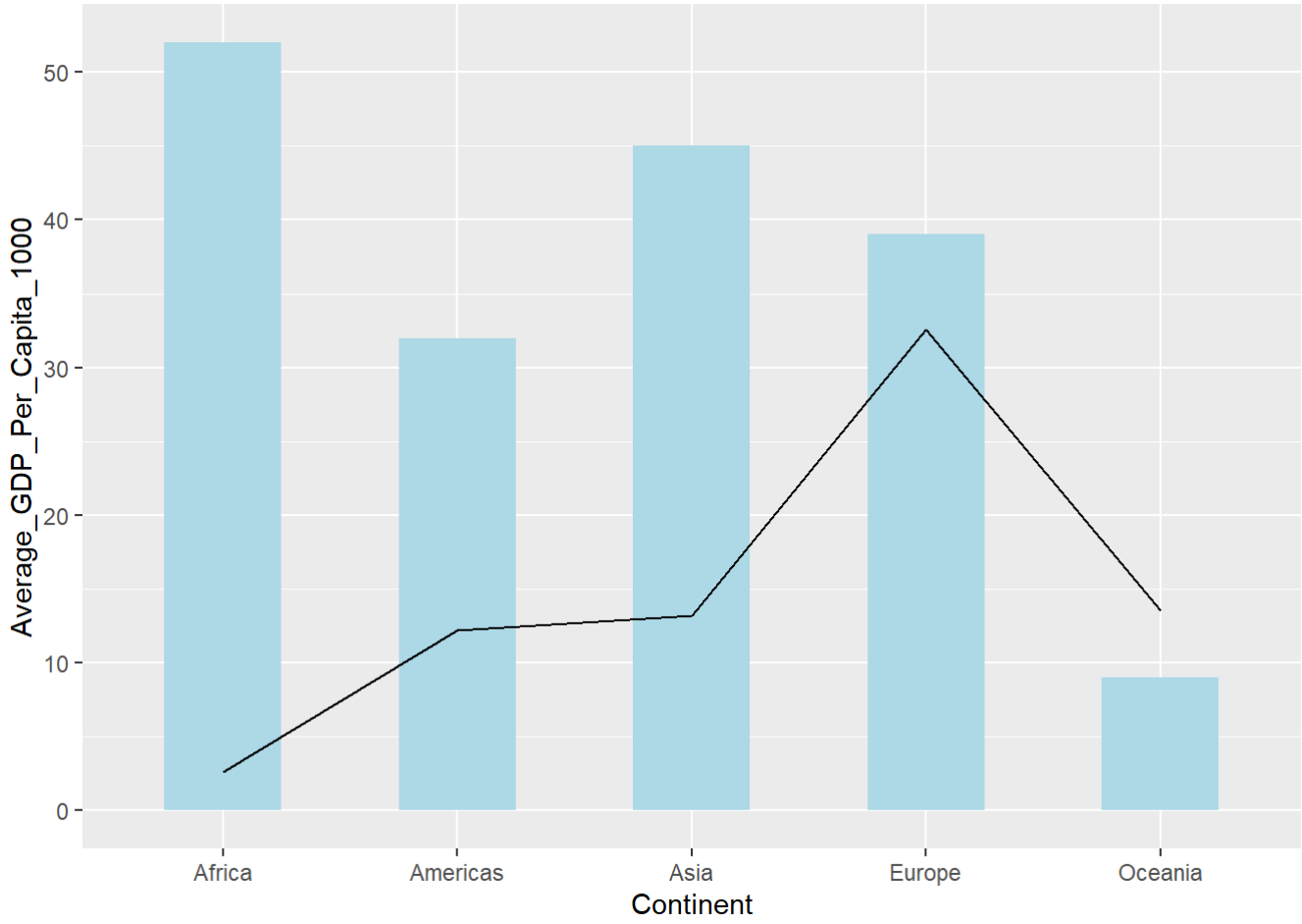
To overlay the two, we can just add the two geom functions:

```
ggplot(data=df, aes(x=Continent, y=Count)) + geom_line(group=1) +
  geom_point(size=5, shape=23, fill="blue")
```



This works in this case because the data here shares the same data source. However, ggplot2 also offers flexibility for plots of different data like so:

```
df = countries %>% group_by(Continent) %>% summarize(Average_GDP_Per_Capita_1000=mean(GDP_Per
_Capita)/1000)

ggplot() + geom_bar(data=countries, aes(x=Continent), fill="lightblue", width=0.5) +
  geom_line(data=df, aes(x=Continent, y=Average_GDP_Per_Capita_1000), group=1)
```

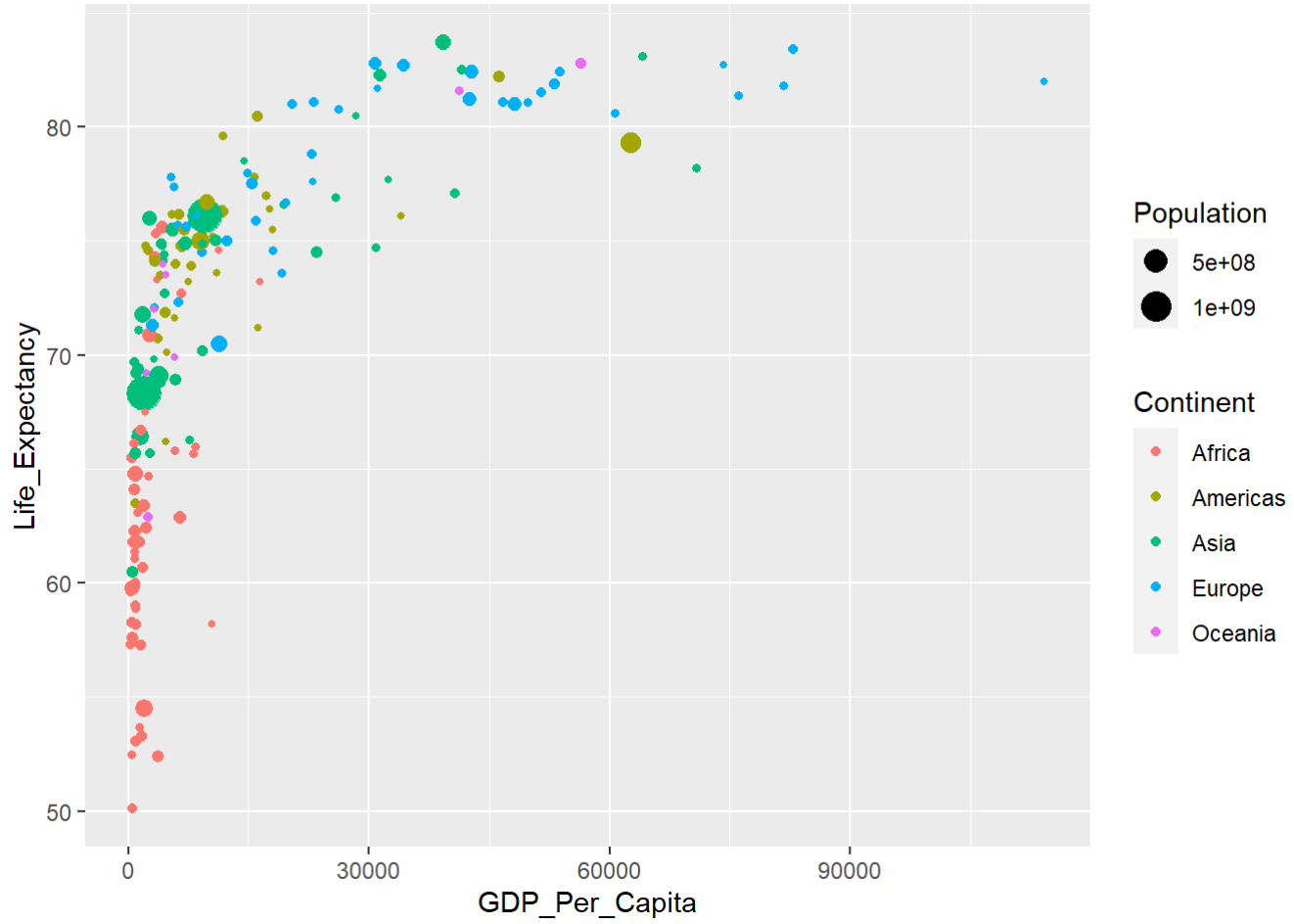


Note that the sequence of geom functions determines the order in which the layers are added. Since the geom_line is added last, it is layered on the top.

Scaling

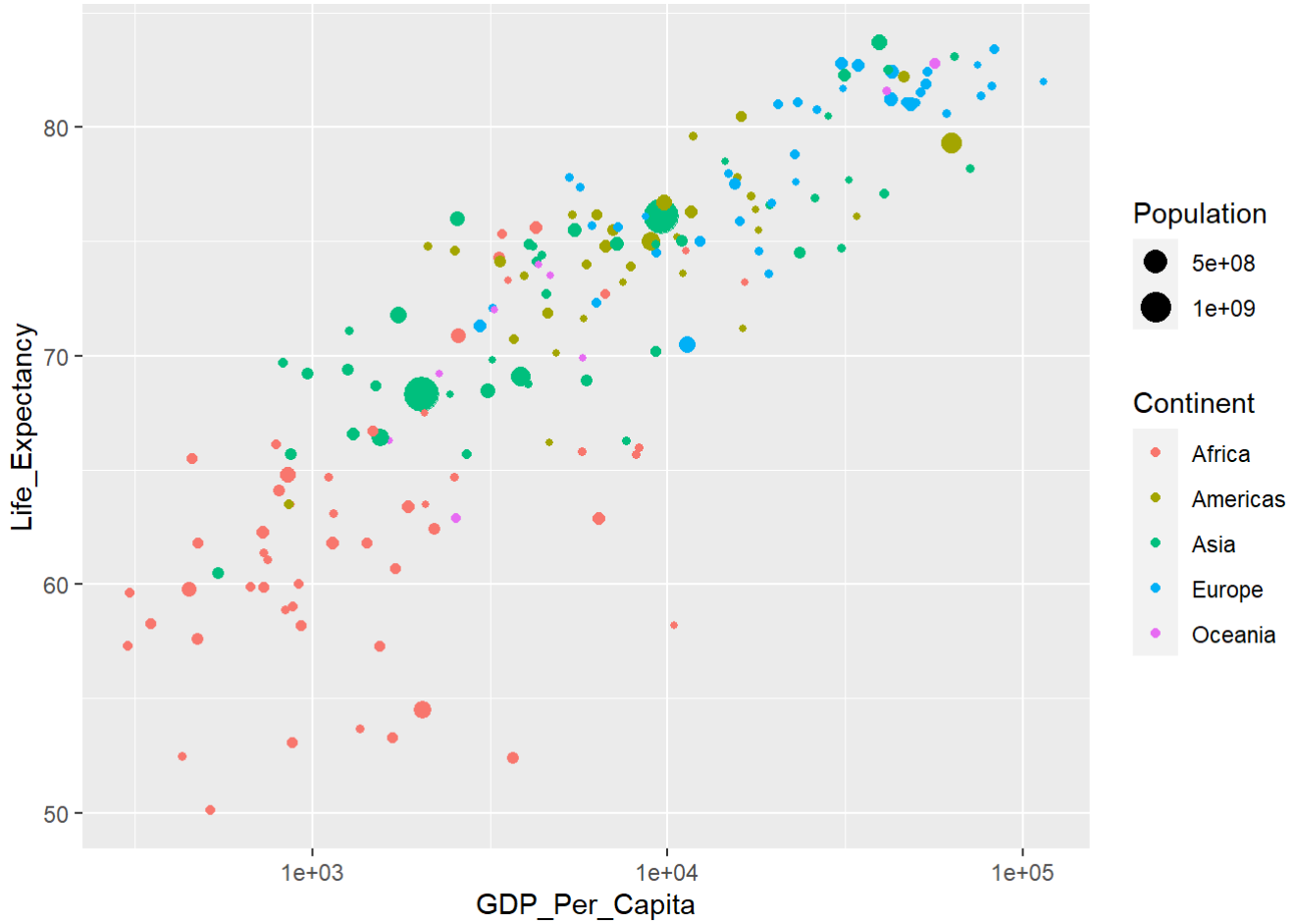
GDP per Capita vs Life Expectancy:

```
ggplot(data=countries, aes(x=GDP_Per_Capita, y=Life_Expectancy,
                           size=Population, color=Continent)) + geom_point()
```



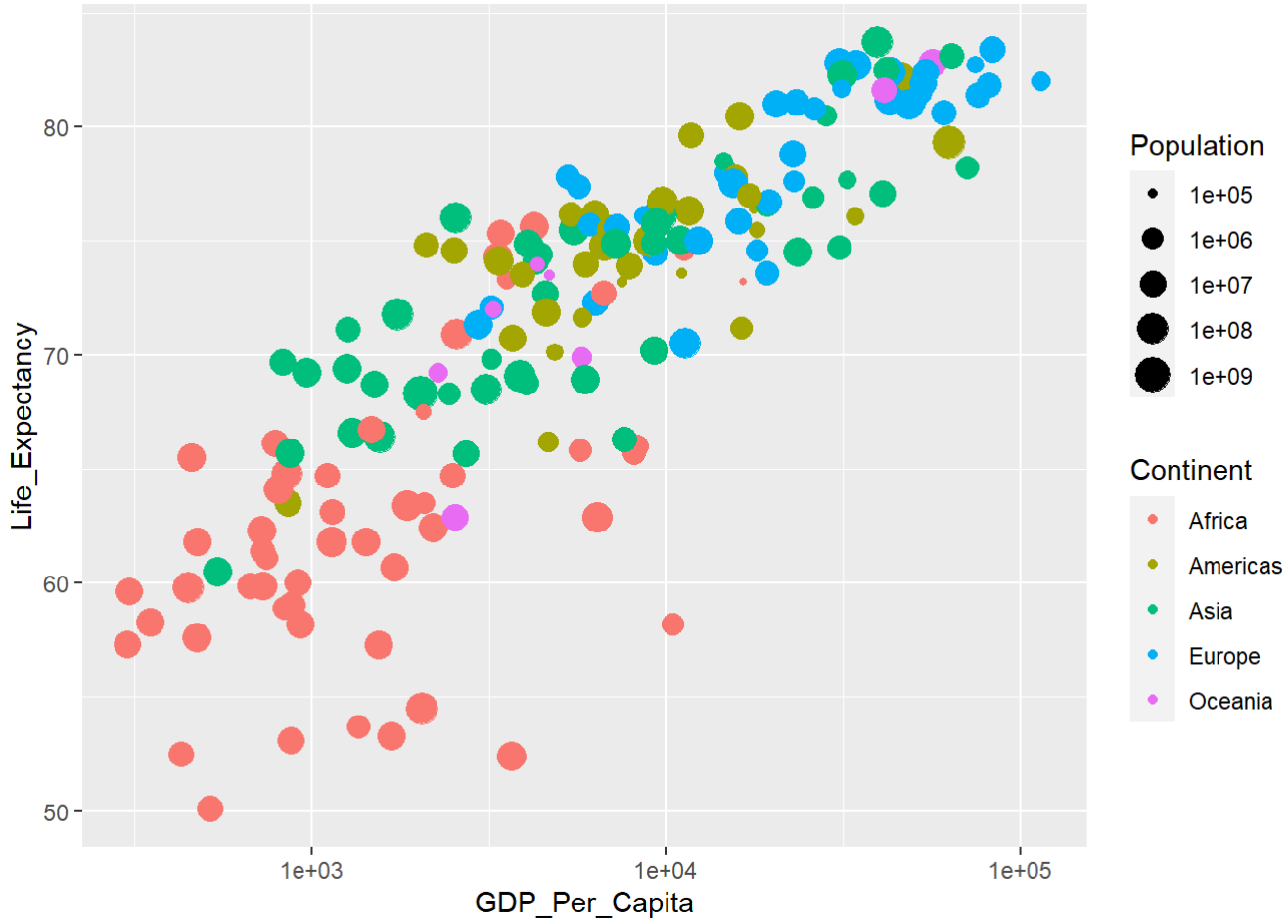
This relationship is non-linear (likely logarithmic). We can rescale the GDP per Capita using **scale_x_log10**:

```
ggplot(data=countries, aes(x=GDP_Per_Capita, y=Life_Expectancy, size=Population,
                           color=Continent)) + geom_point() + scale_x_log10()
```



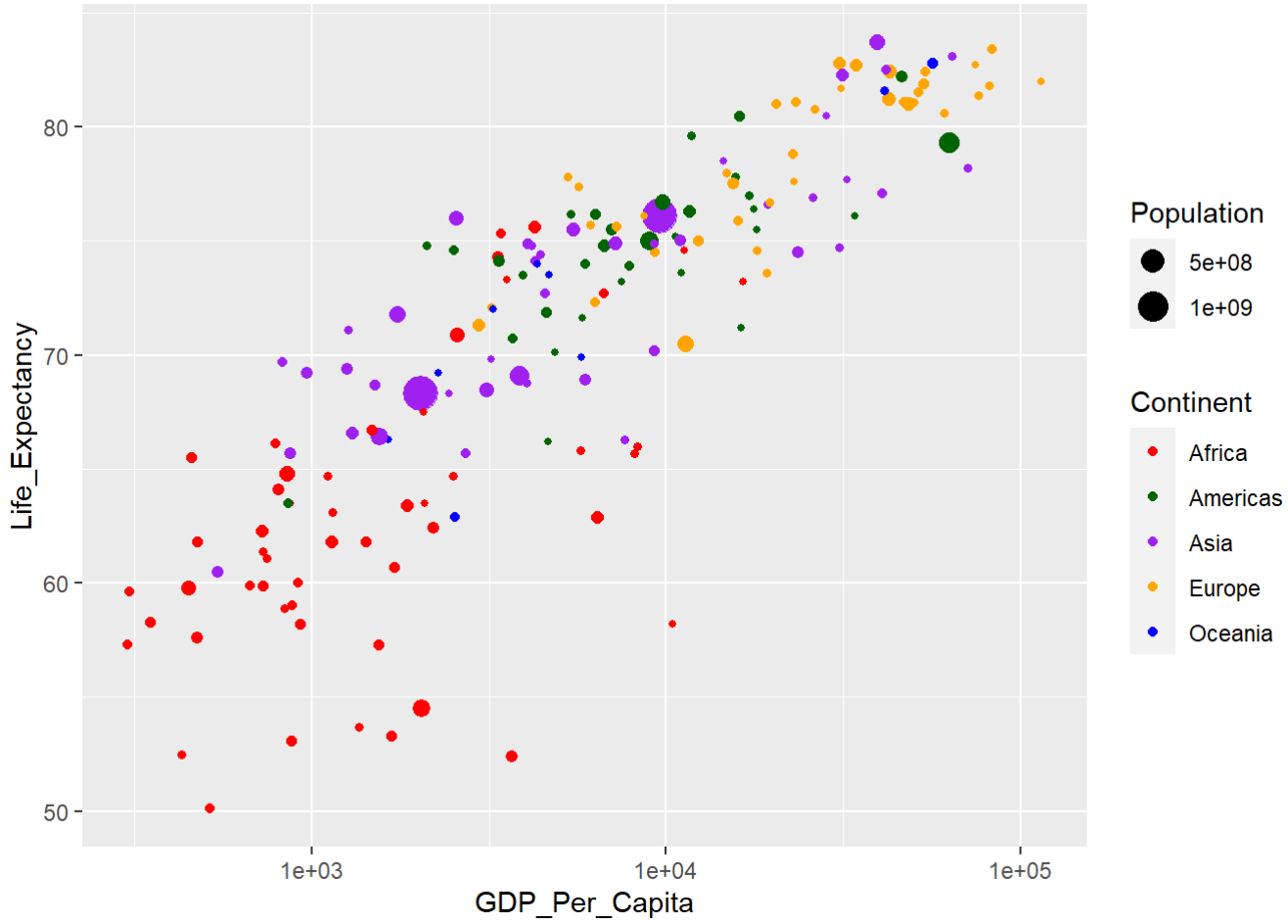
We can also scale other factors (e.g. point size). In this case, we resize the population to a logarithmic scale using **scale_size_continuous**:

```
ggplot(data=countries, aes(x=GDP_Per_Capita, y=Life_Expectancy,
                           size=Population, color=Continent)) +
  geom_point() + scale_x_log10() + scale_size_continuous(trans="log10")
```



We can also set the colours of the points manually using **scale_color_manual**:

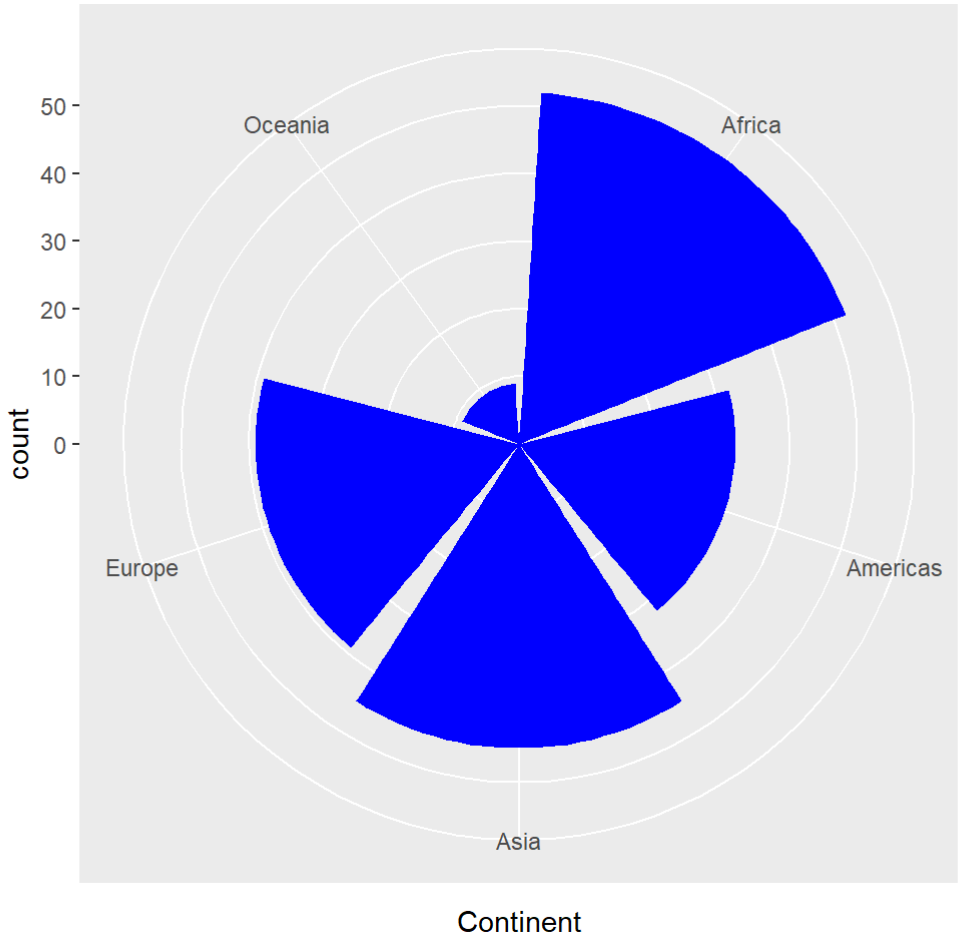
```
ggplot(data=countries, aes(x=GDP_Per_Capita, y=Life_Expectancy,
                           size=Population, color=Continent)) +
  geom_point() + scale_x_log10() + scale_color_manual(values=c("red", "darkgreen", "purple",
"orange", "blue"))
```



Coordinate System

By default, ggplot2 uses the Cartesian coordinate system. We can switch to a Polar system using **coord_polar**:

```
ggplot(data=countries,aes(x=Continent)) + geom_bar(fill="blue") + coord_polar()
```

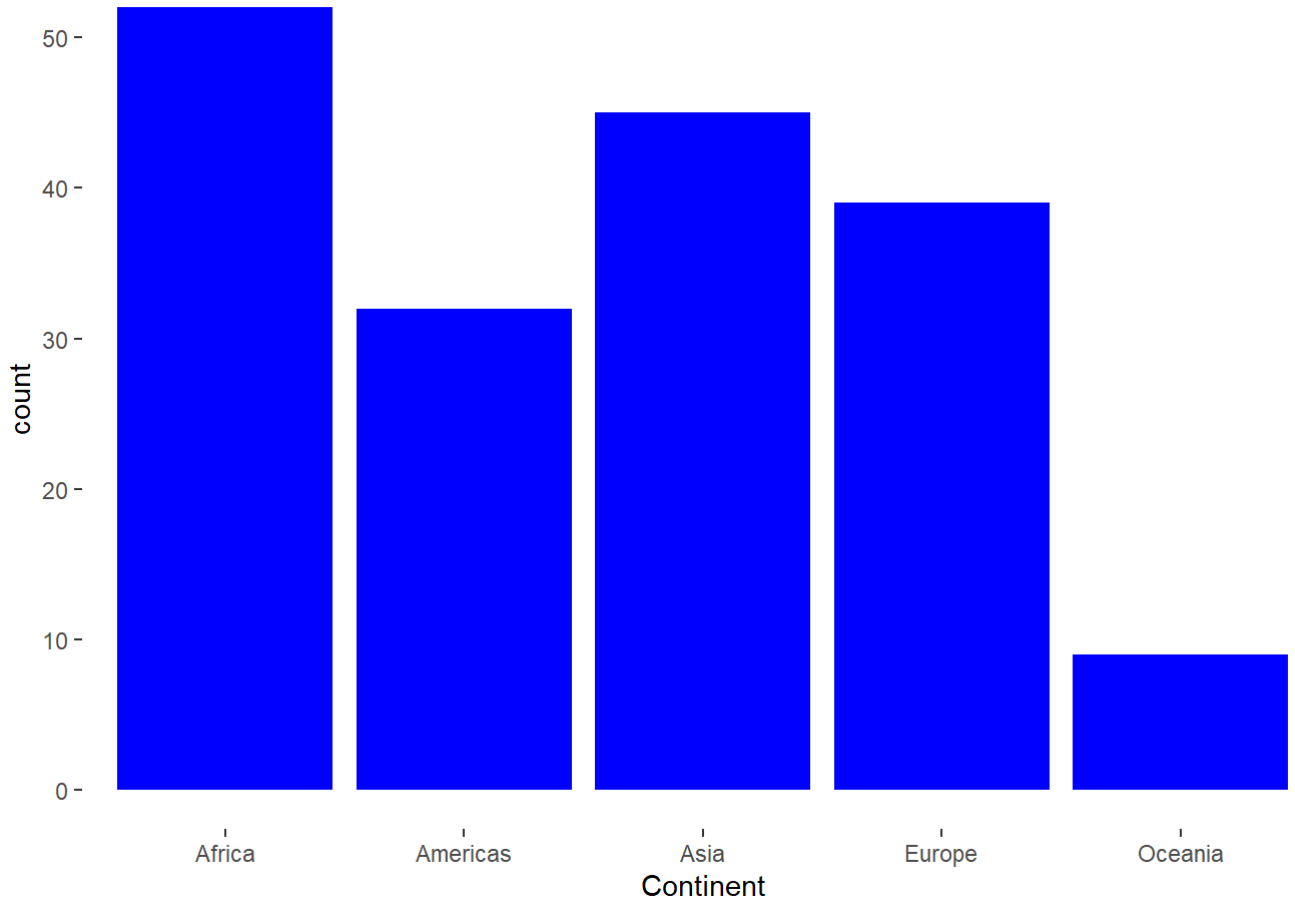


Graph Components

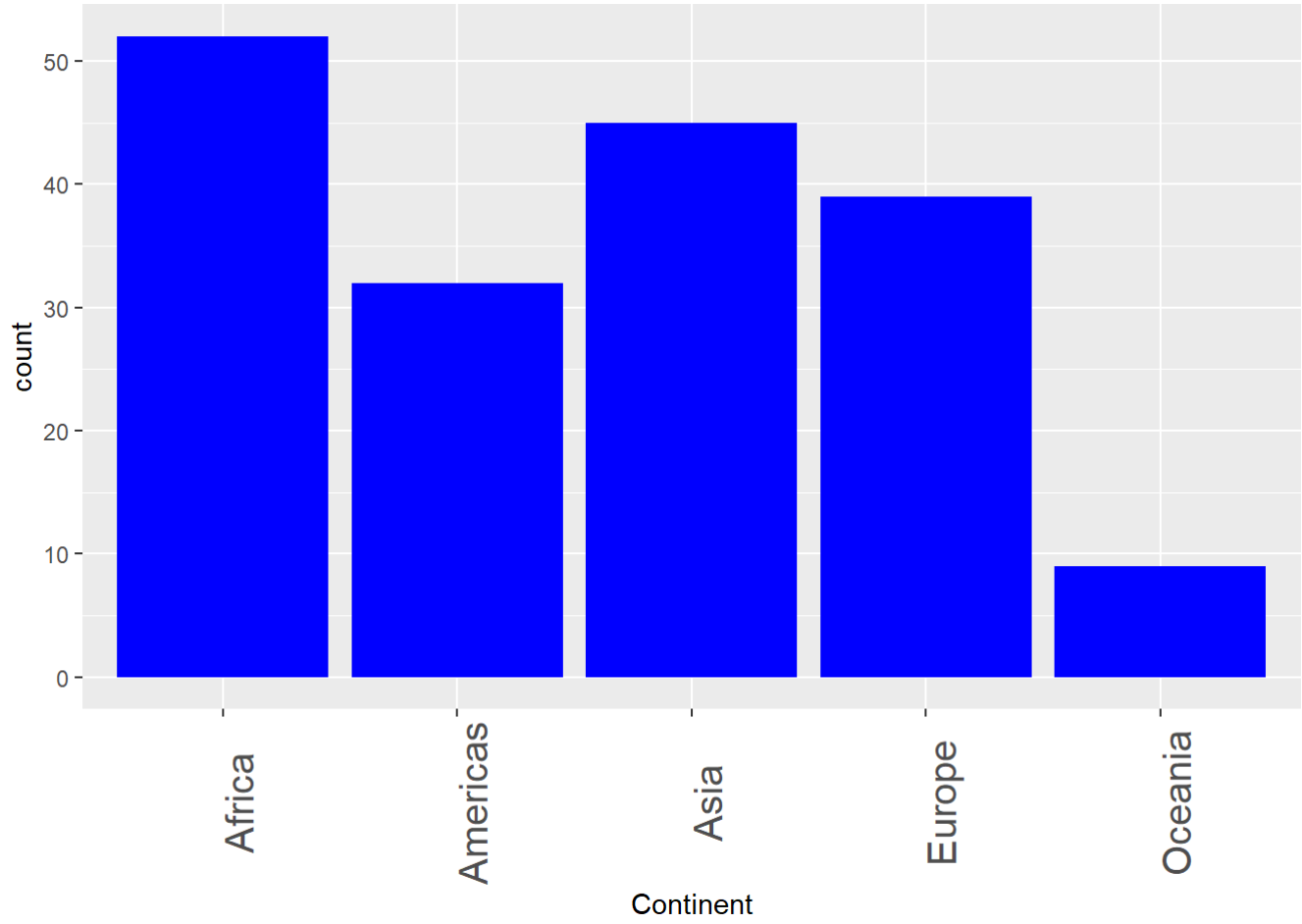
1. Plot title (Text)
2. Axis Title (Text)
3. Tick Label (Text)
4. Background Panel (Rectangle)
5. Grid Lines (Line)
6. Others (found under ?theme)

Each of the 3 types can be manipulated using the **element_text**, **element_rect** and **element_line** functions respectively:

```
## change bg to white
ggplot(data=countries,aes(x=Continent)) + geom_bar(fill="blue") +
  theme(panel.background=element_rect(fill="white"))
```

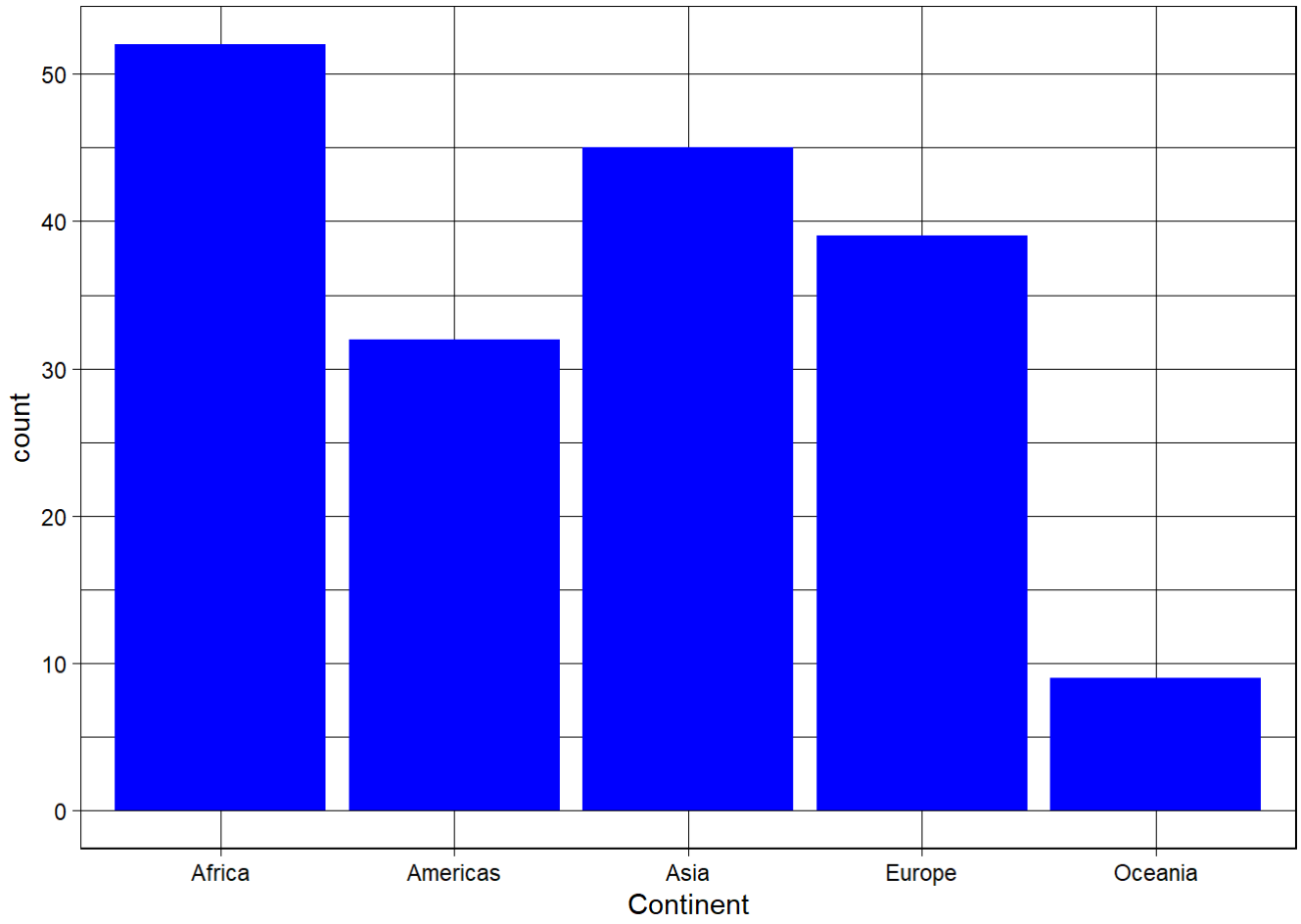



```
## change orientation of tick labels
ggplot(data=countries,aes(x=Continent)) + geom_bar(fill="blue") +
  theme(axis.text.x=element_text(size=15, angle=90))
```



We can also adopt a theme:

```
ggplot(data=countries,aes(x=Continent)) + geom_bar(fill="blue") + theme_linedraw()
```



Faceting

```
library("gapminder")
```

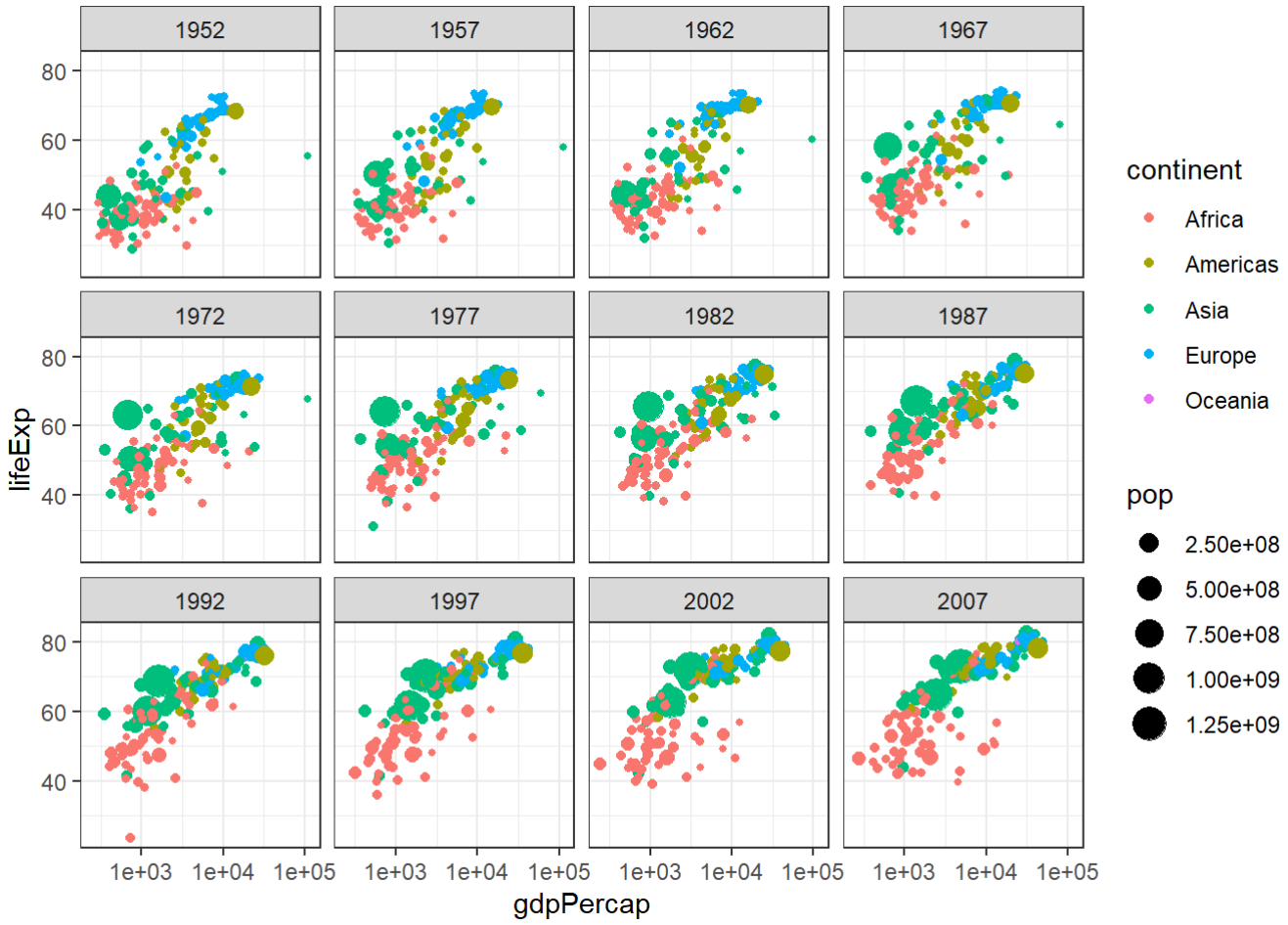
```
## Warning: package 'gapminder' was built under R version 4.1.1
```

```
head(gapminder)
```

```
## # A tibble: 6 x 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

gapminder contains the data for multiple years. If we plot it as such, the data becomes very messy. Instead, we can facet the data by year using **facet_wrap** to produce multiple side-by-side plots:

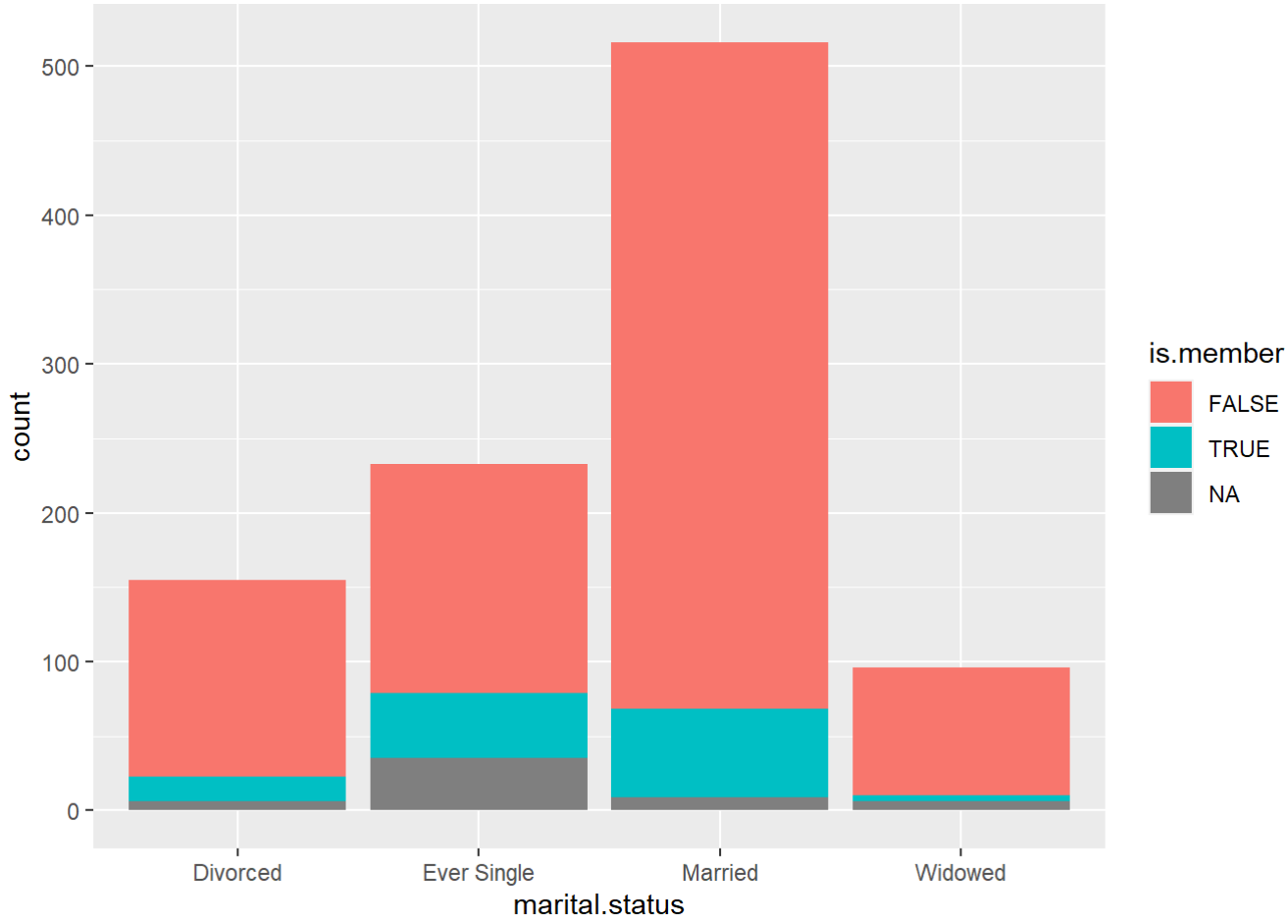
```
ggplot(data=gapminder, aes(x=gdpPercap, y=lifeExp, size=pop, color=continent)) +
  geom_point() + scale_x_log10() + theme_bw() + facet_wrap(~year)
```



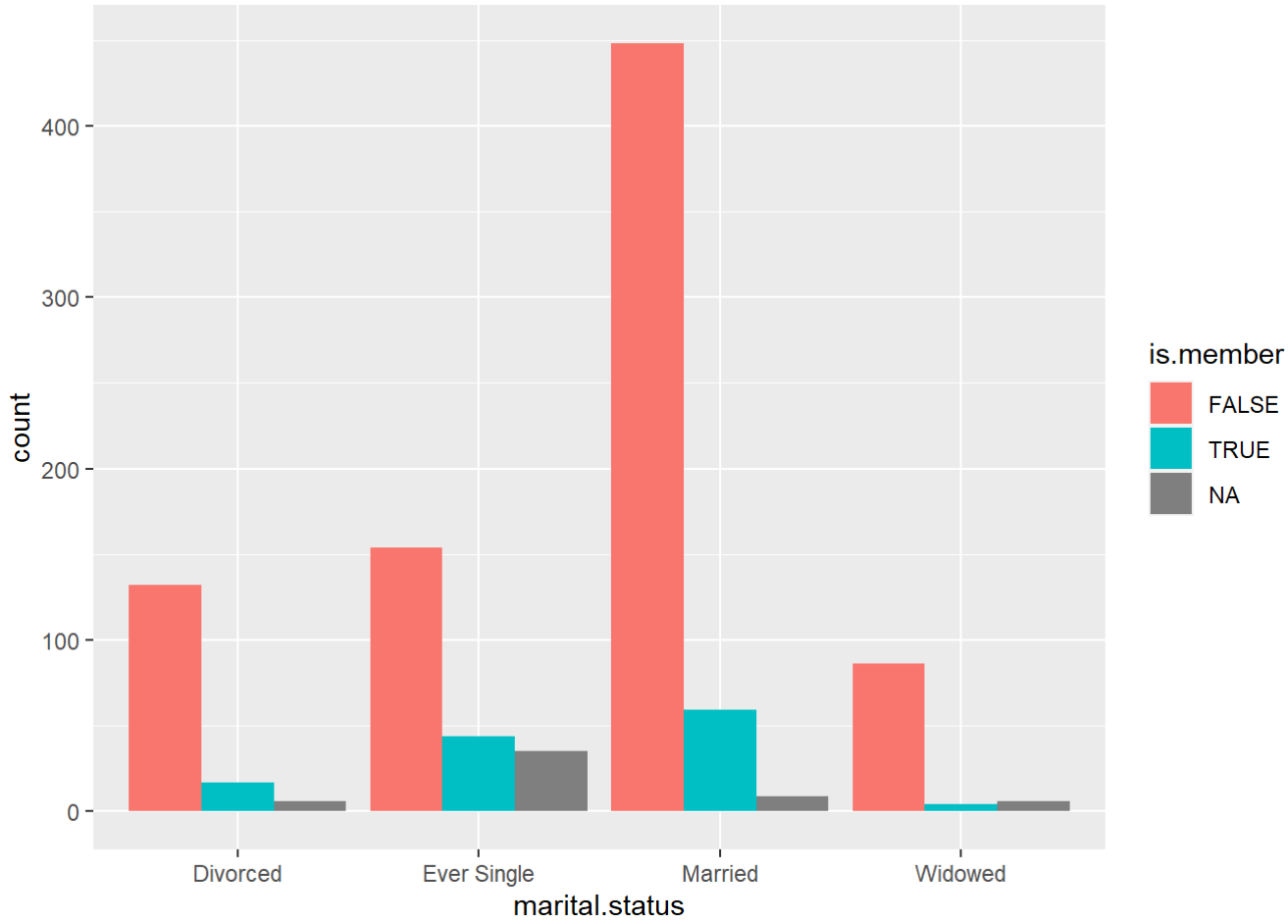
Positioning

- identity (default for most geoms)
- jitter (default for geom_jitter)
- dodge (default of geom_boxplot)
- stack (default of geom_bar, geom_histogram, geom_area)
- fill (useful for geom_bar, geom_histogram, geom_area)

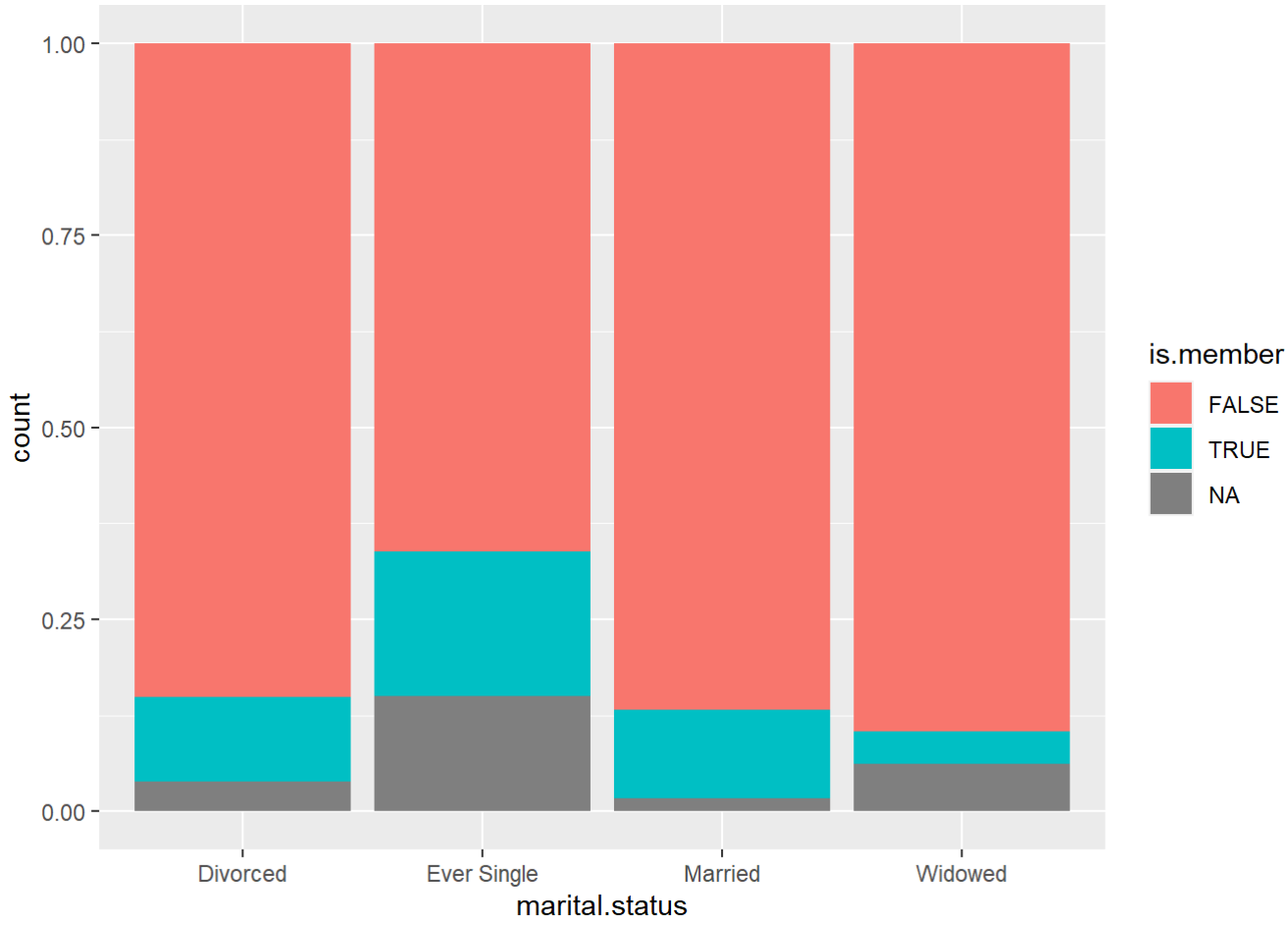
```
custdata = read.csv("./Data/Customers.csv")

ggplot(custdata) + geom_bar(aes(x=marital.status, fill=is.member))
```




```
ggplot(custdata) + geom_bar(aes(x=marital.status, fill=is.member), position="dodge")
```



```
ggplot(custdata) + geom_bar(aes(x=marital.status, fill=is.member), position="fill")
```



gganimate

Animate the changes in plot by year using **transition_time**

```
library("gganimate")
```

```
## Warning: package 'gganimate' was built under R version 4.1.1
```

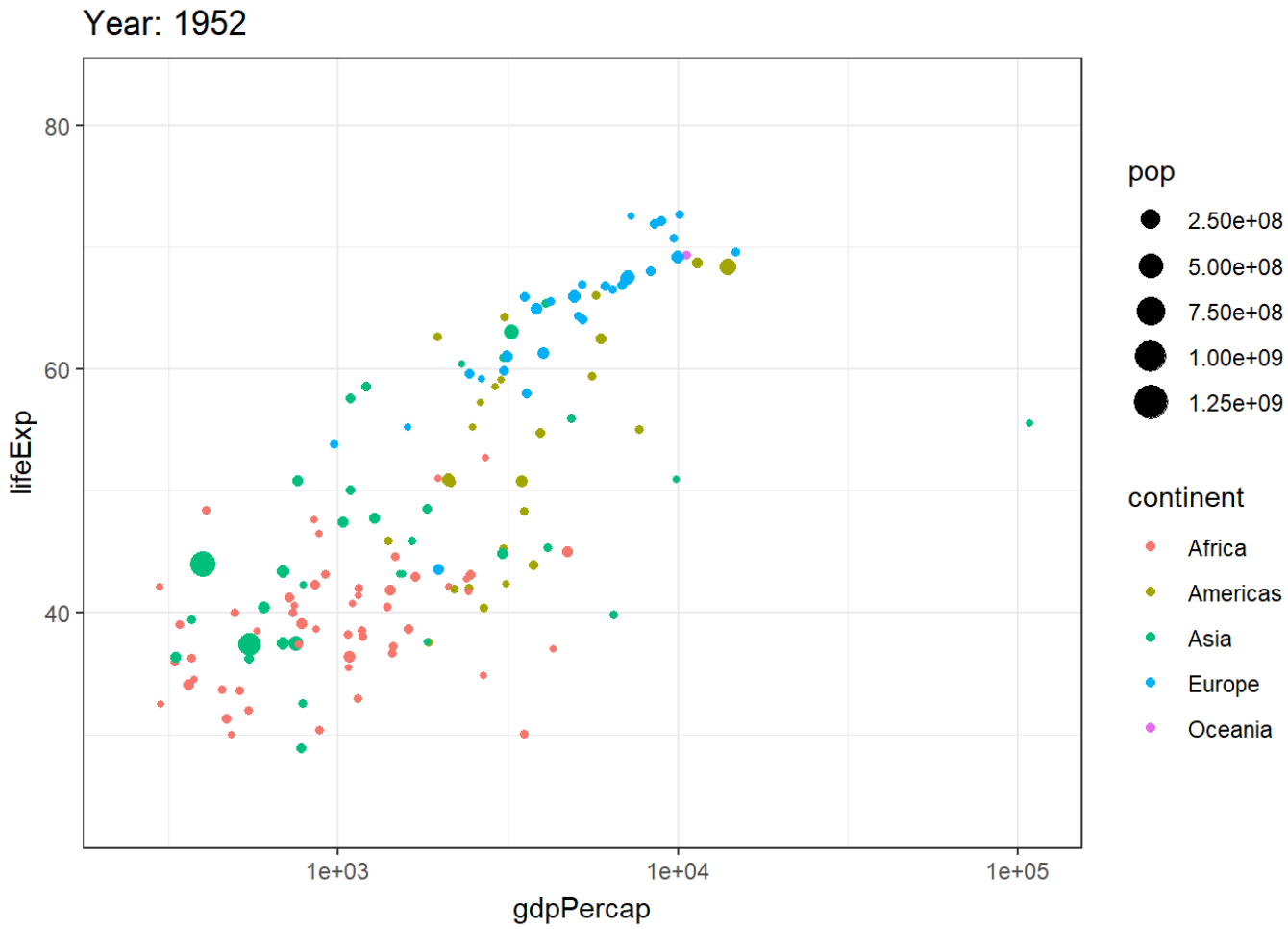
```
library("png")
```

```
## Warning: package 'png' was built under R version 4.1.1
```

```
library("gifski")
```

```
## Warning: package 'gifski' was built under R version 4.1.1
```

```
p = ggplot(data=gapminder, aes(x=gdpPercap, y=lifeExp, size=pop, color=continent)) +
  geom_point() + scale_x_log10() + theme_bw()

p + transition_time(year) + labs(title="Year: {frame_time}")
```


Year: 1952
lifeExp
80
60
40
1e+03
1e+04
1e+05
gdpPercap
pop
2.50e+08
5.00e+08
7.50e+08
1.00e+09
1.25e+09
continent
Africa
Americas
Asia
Europe
Oceania

Lecture 8

Data Transformation

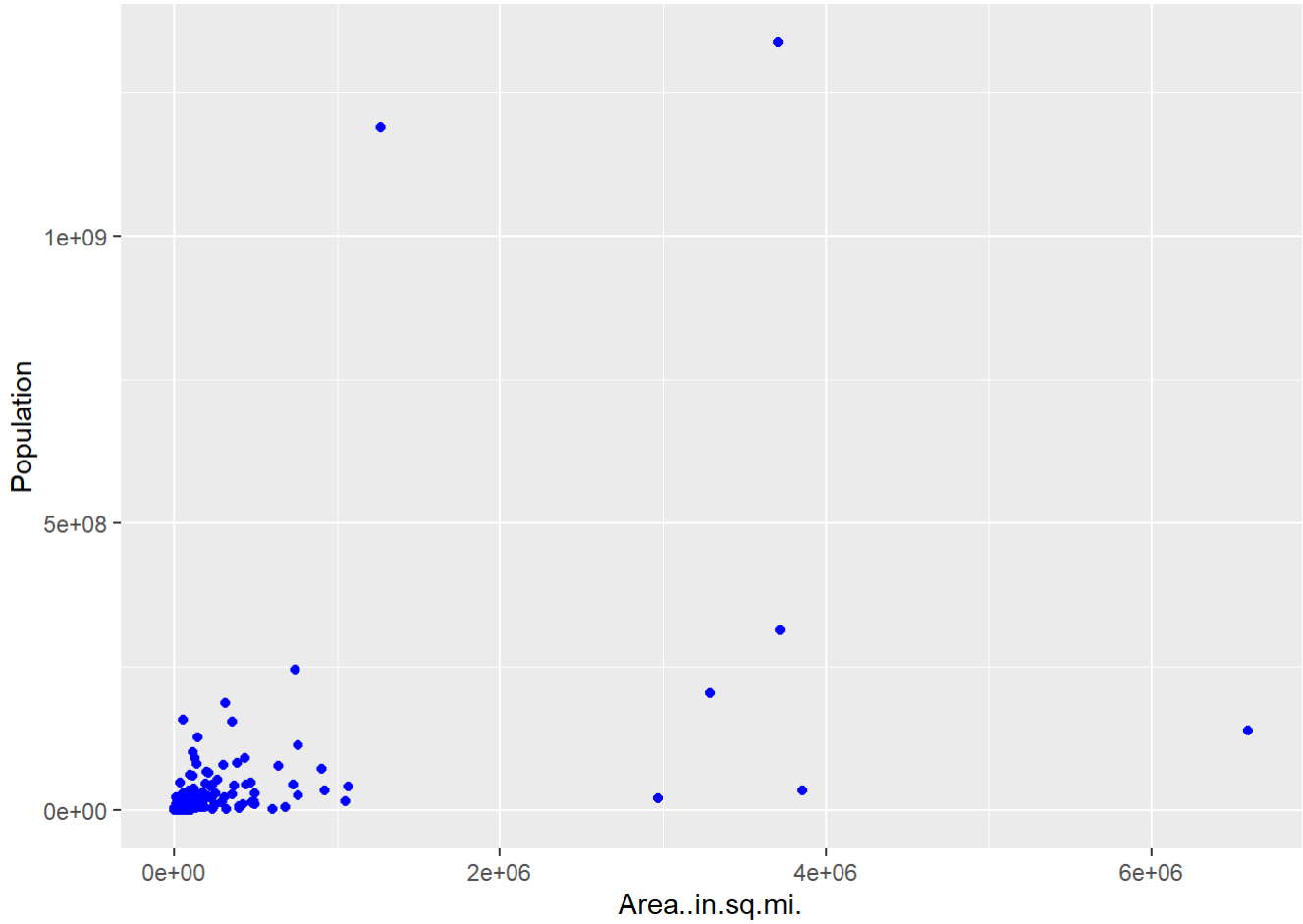
1. Modelling Needs
2. Better Visualisation
3. Better Interpretation

1. Mathematical Transformation

```
library("ggplot2")
```

```
## Warning: package 'ggplot2' was built under R version 4.1.1
```

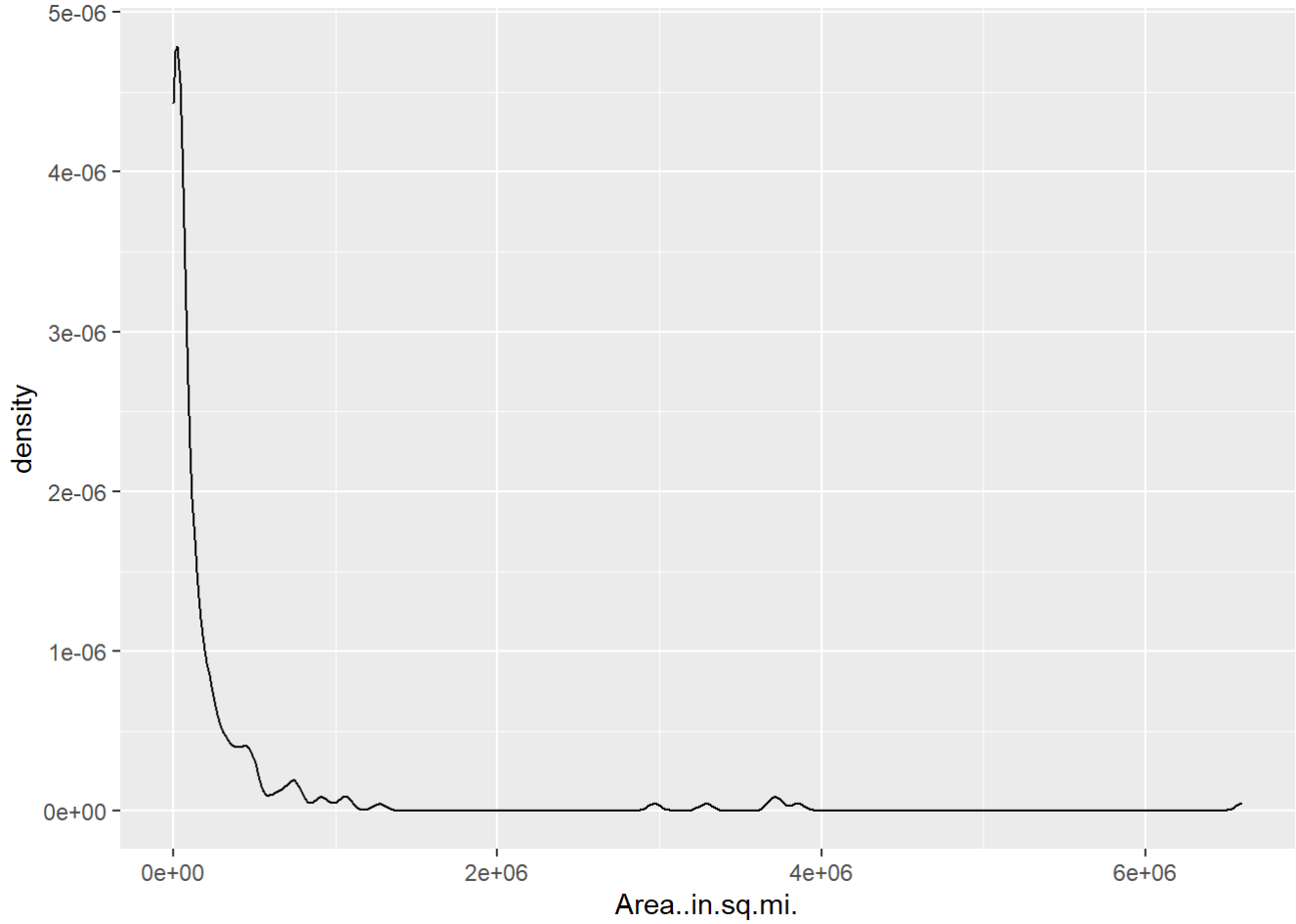
```
data = read.csv("./Data/area&population-raw.csv")
ggplot(data, aes(x=Area..in.sq.mi., y=Population)) + geom_point(colour="blue")
```



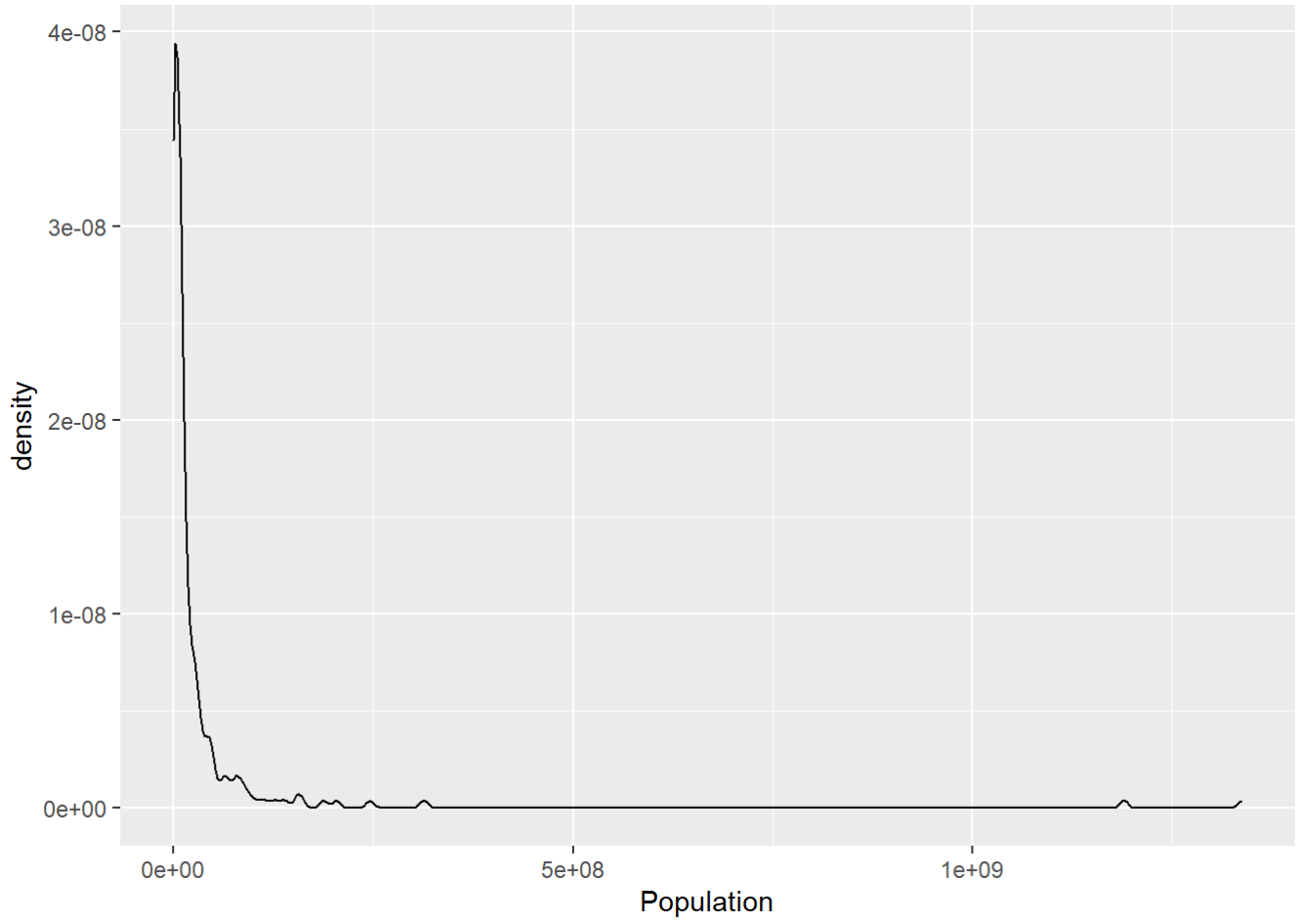
The visualisation is poor. Most points are concentrated in the bottom left corner, making it difficult to draw any reasonable conclusions. To figure out how to transform the data, we need to understand it first.

Density plot of area:

```
ggplot(data, aes(x=Area..in.sq.mi.)) + geom_density()
```



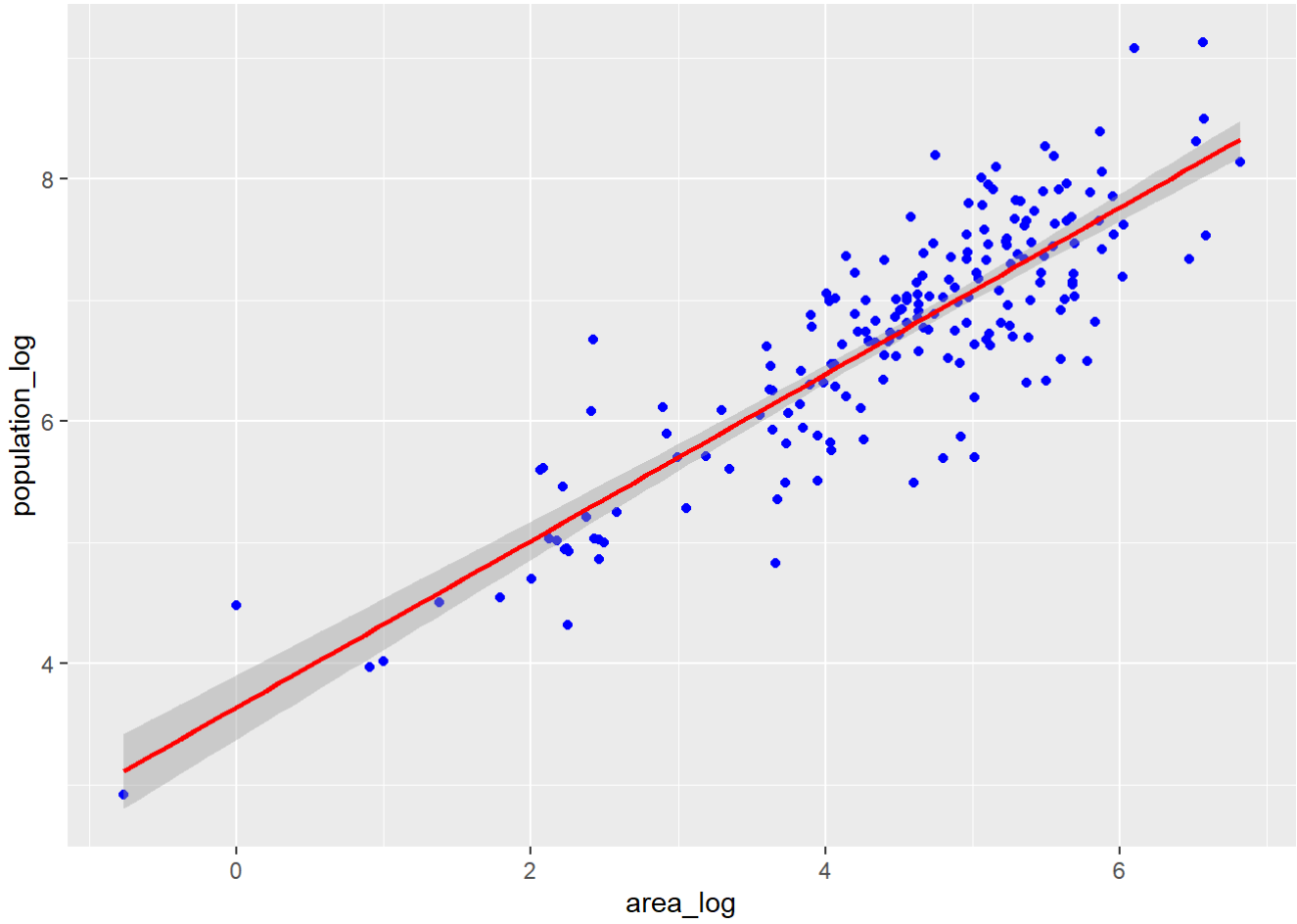

```
ggplot(data, aes(x=Population)) + geom_density()
```



The graphs look logarithmic. We can now transform the graphs.

```
data$area_log = log10(data$Area..in.sq.mi.)
data$population_log = log10(data$Population)

ggplot(data, aes(x=area_log, y=population_log)) + geom_point(colour="blue") + geom_smooth(met
hod="lm", color="red")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



2. Categorisation

Sometimes, data is split into categories. For numerical values, they are usually split into intervals of equal lengths.

Note that this method might carry risks. For example, 0-2 years old vs 20-22 years old. Babies change a lot in 3 years while young adults do not change as dramatically. We must always evaluate what the meaning of the intervals we choose.

Numerical values are also split along convenient numbers (e.g. 1-1000, 1000-5000). Such methods have no statistical foundation. This is also risky as there might not be a pronounced difference between the 2 groups (e.g. 999 vs 1001).

One way to base our categorisations is with visualisation:

```
custdata = read.table("./Data/custdata.tsv", sep="\t", header=T)

ggplot(custdata, aes(x=income, y=as.numeric(health.ins))) + geom_point(position=position_jitt
er(w=0.05, h=0.05)) + scale_x_log10(breaks=c(100,1000,10000,100000)) + geom_smooth() + annota
tion_logticks(sides="bt")
```

```
## Warning in self$trans$transform(x): NaNs produced
```

```
## Warning: Transformation introduced infinite values in continuous x-axis
```

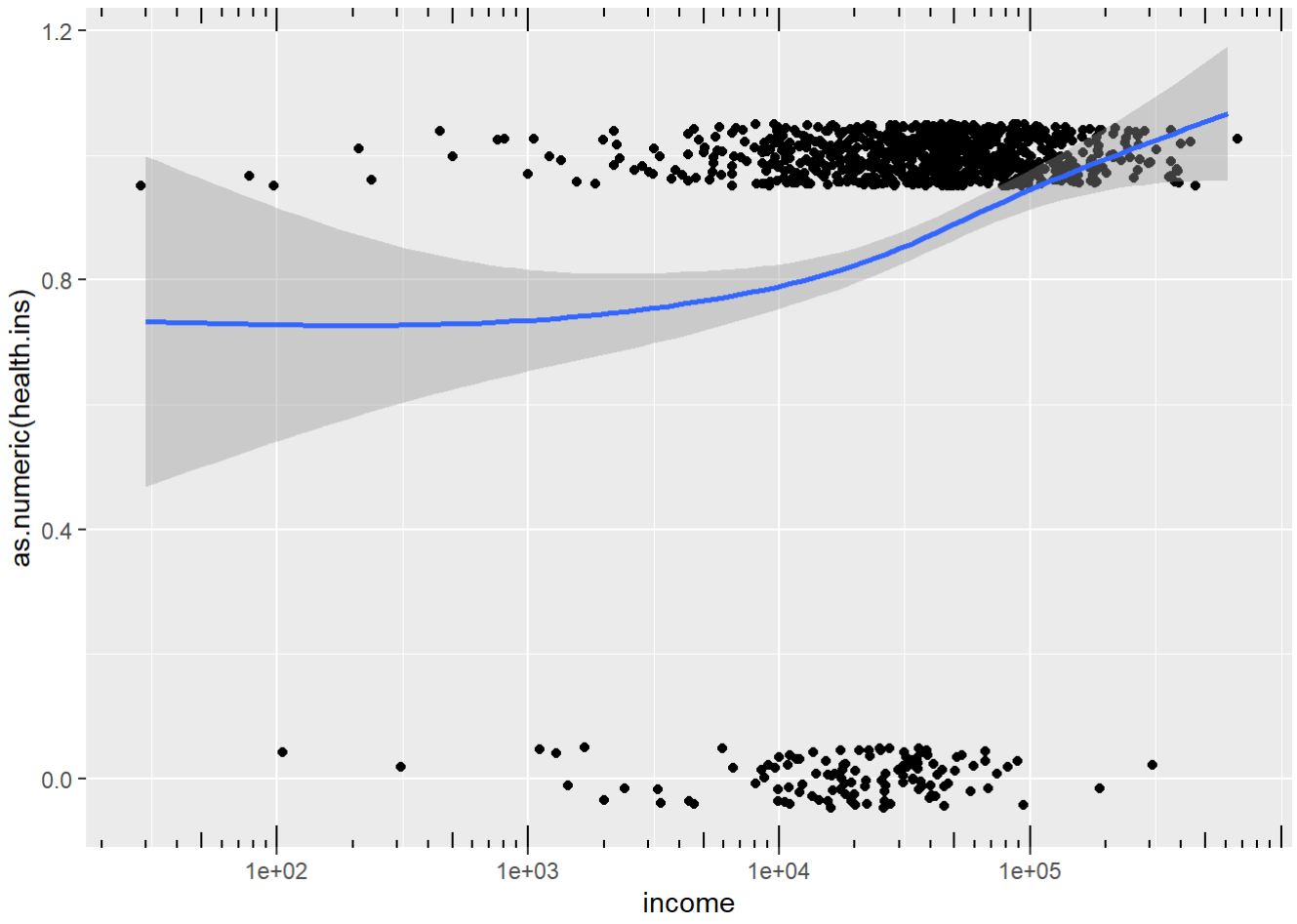
```
## Warning in self$trans$transform(x): NaNs produced
```

```
## Warning: Transformation introduced infinite values in continuous x-axis
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```

```
## Warning: Removed 79 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 79 rows containing missing values (geom_point).
```



The blue line is the best fit line of customers' likelyhood of buying insurance with respect to their income. It has a clear increase between 20,000 to 100,000. Hence, it is best to have 3 groups: <20000, 20000-100000, >100000.

3. Normalisation

When we want to compare data that is affected by other environmental factors that cause its weight to differ, we need to normalise it. For example, it is difficult to draw any reasonable conclusions about the income in Indonesia vs in Singapore using just the raw data. Other factors like exchange rate and purchasing power greatly affect our results.

We should instead use the median/mean as benchmarks within each country to normalise their respective income data.

```
# join data
custdata = merge(custdata, median.income, by.x="State.of.res", by.y="State")

# normalise
custdata$income.normalised = with(custdata, income/Per.Capita.Income)
```

ggmap

ggmap is a package developed on top of ggplot2. Hence all plots from ggmap can be overlayed with plots from ggplot2.

```
library(ggmap)
```

```
## Warning: package 'ggmap' was built under R version 4.1.1
```

```
## Google's Terms of Service: https://cloud.google.com/maps-platform/terms/.
```

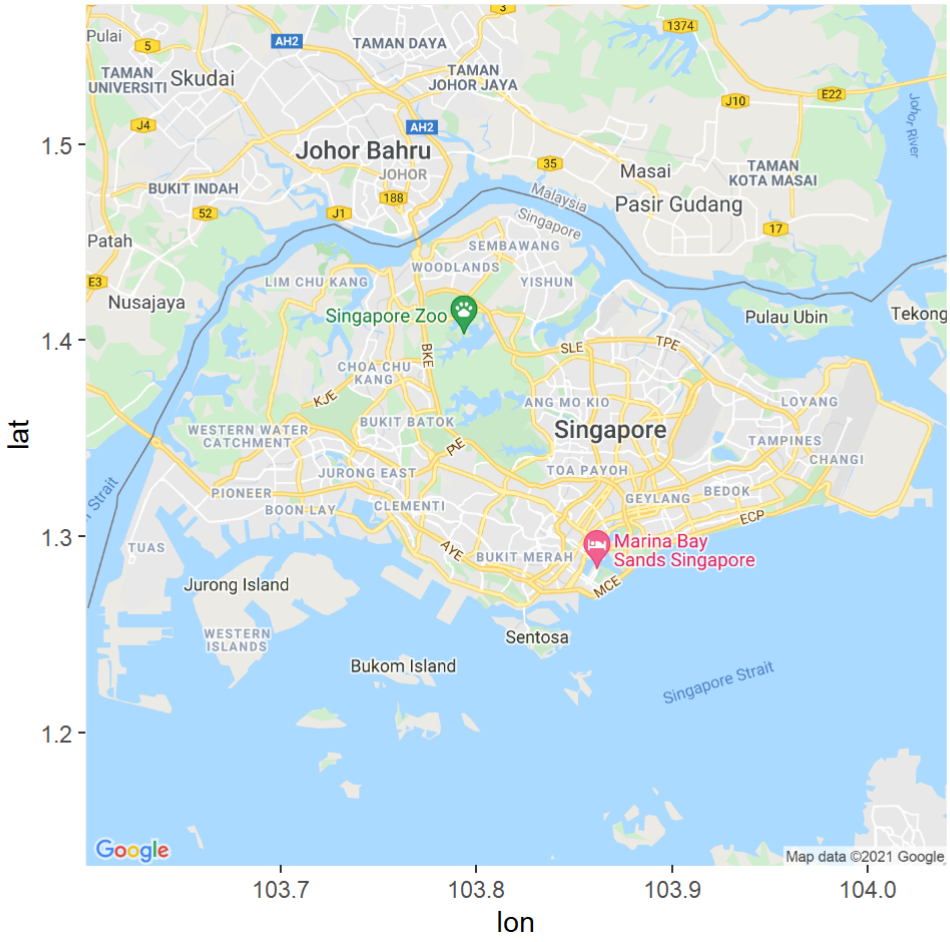
```
## Please cite ggmap if you use it! See citation("ggmap") for details.
```

```
ggmap::register_google(key='AIzaSyCnGFj3-tmhyhkx2Suxw3HNa6P0c0fjHc0')

m = get_map("Singapore", source="google", zoom=11, maptype="roadmap")
```

```
## Source : https://maps.googleapis.com/maps/api/staticmap?center=Singapore&zoom=11&size=640x
640&scale=2&maptype=roadmap&language=en-EN&key=xxx-tmhyhkx2Suxw3HNa6P0c0fjHc0
```

```
## Source : https://maps.googleapis.com/maps/api/geocode/json?address=Singapore&key=xxx-tmhyh
kx2Suxw3HNa6P0c0fjHc0
```

```
ggmap(m)
```



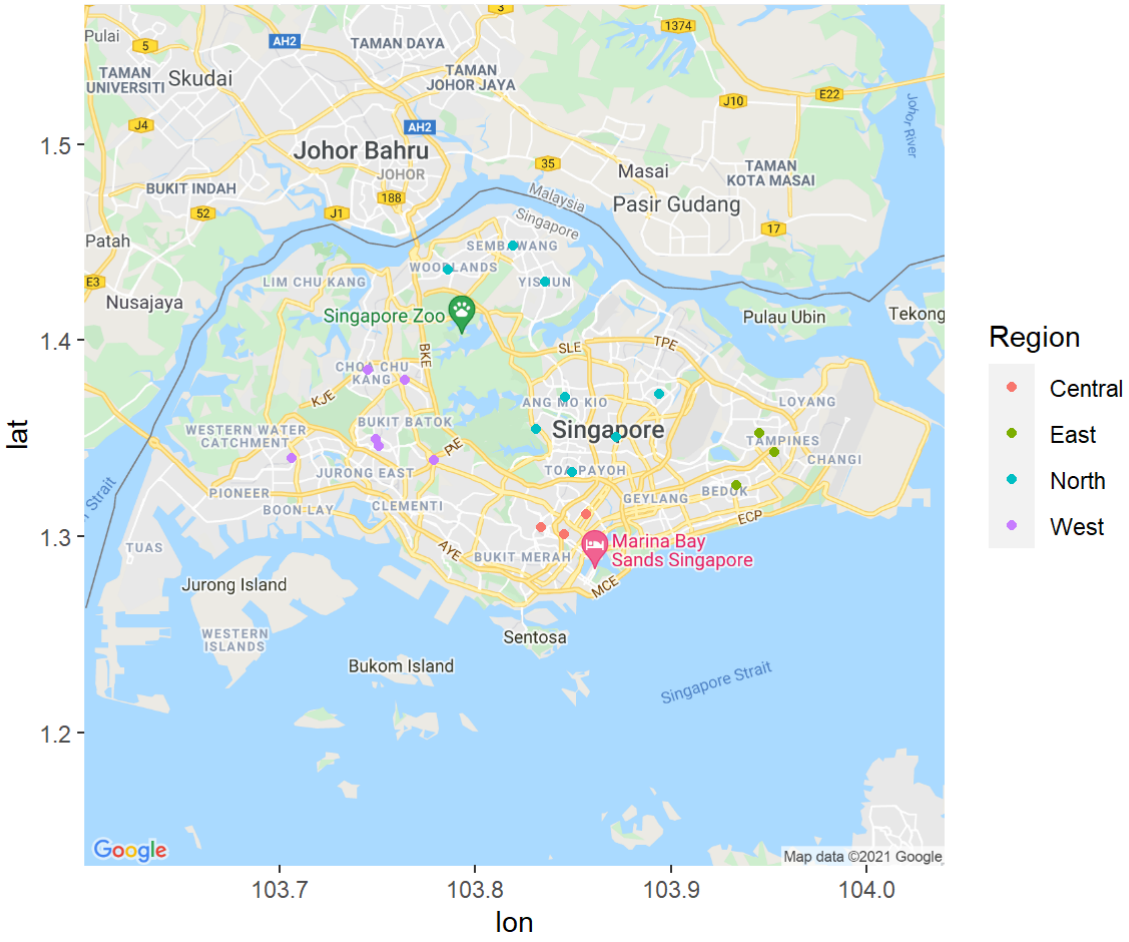
Integrating with ggplot2

```
pizzahut.location = read.csv("./Data/PizzaHut.csv", header=T, colClasses=c("character", "char
acter", "factor", "character", "numeric", "numeric"))

# get singapore map
map = get_map("Singapore", zoom=11, source="google")
```

```
## Source : https://maps.googleapis.com/maps/api/staticmap?center=Singapore&zoom=11&size=640x
640&scale=2&maptype=terrain&language=en-EN&key=xxx-tmhyhkx2Suxw3HNa6P0c0fjHc0
```

```
## Source : https://maps.googleapis.com/maps/api/geocode/json?address=Singapore&key=xxx-tmhyh
kx2Suxw3HNa6P0c0fjHc0
```

```
# plot latitude and longitude and combine with map
m1 = ggmap(m, base_layer = ggplot(data=pizzahut.location, aes(x=lon, y=lat)))

# plot points by region
m1 + geom_point(aes(color=Region))
```



Region Borders

Sometimes, we want a map, not by the country but by the regions. This cannot be done with only ggmap. We will use **raster** and **rgdal**.

```
library("raster")
```

```
## Warning: package 'raster' was built under R version 4.1.1
```

```
## Loading required package: sp
```

```
## Warning: package 'sp' was built under R version 4.1.1
```

```
library("rgdal")
```

```
## Warning: package 'rgdal' was built under R version 4.1.1
```

```
## Please note that rgdal will be retired by the end of 2023,
## plan transition to sf/stars/terra functions using GDAL and PROJ
## at your earliest convenience.
##
## rgdal: version: 1.5-27, (SVN revision 1148)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 3.2.1, released 2020/12/29
## Path to GDAL shared files: C:/Users/Carissa Ying/Documents/R/win-library/4.1/rgdal/gdal
## GDAL binary built with GEOS: TRUE
## Loaded PROJ runtime: Rel. 7.2.1, January 1st, 2021, [PJ_VERSION: 721]
## Path to PROJ shared files: C:/Users/Carissa Ying/Documents/R/win-library/4.1/rgdal/proj
## PROJ CDN enabled: FALSE
## Linking to sp version:1.4-5
## To mute warnings of possible GDAL/OSR exportToProj4() degradation,
## use options("rgdal_show_exportToProj4_warnings"="none") before loading sp or rgdal.
## Overwritten PROJ_LIB was C:/Users/Carissa Ying/Documents/R/win-library/4.1/rgdal/proj
```

```
library("XML")
```

```
## Warning: package 'XML' was built under R version 4.1.1
```

```
SG = getData('GADM', country="SG", level=1) # GADM is the source of the map data
class(SG) # SpatialPolygonsDataFrame
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
# Access elements using @
class(SG@data) # data frame
```

```
## [1] "data.frame"
```

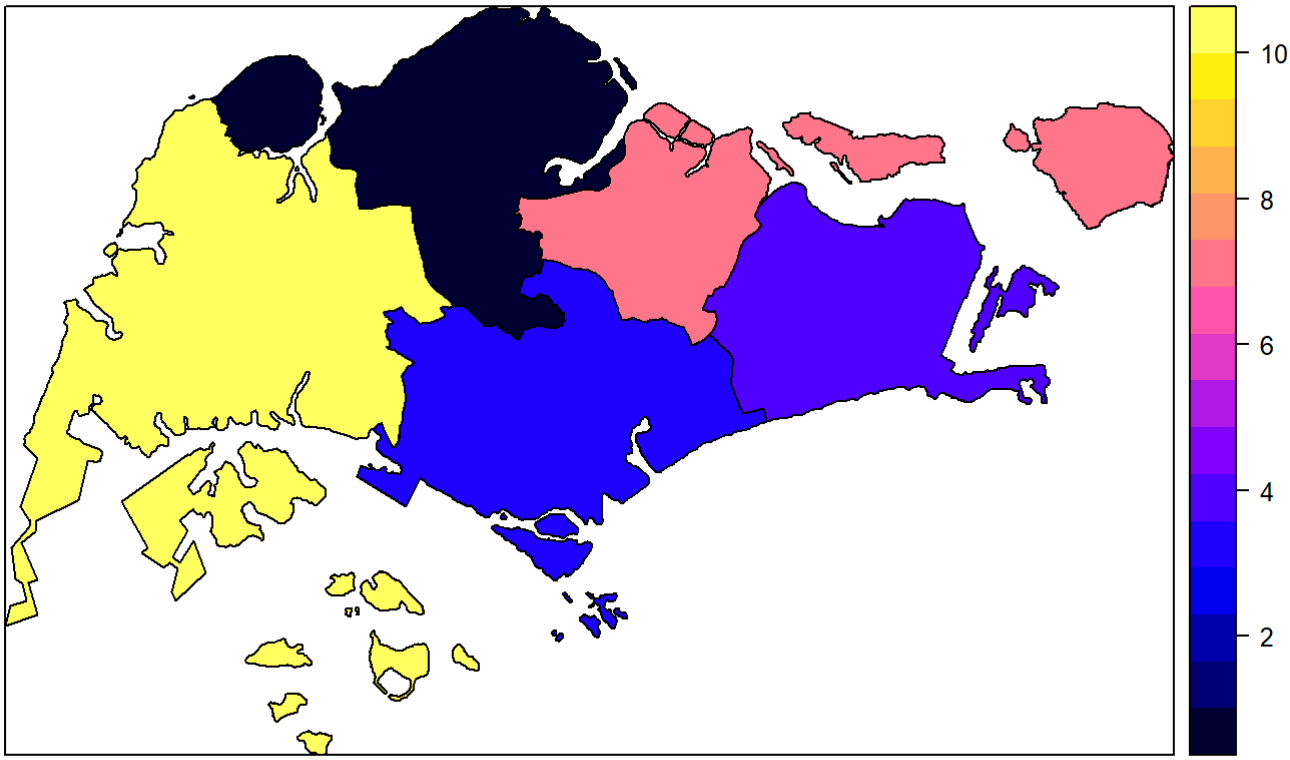
```
head(SG@data) # used to communicate with other data
```

```
##        GID_0    NAME_0   GID_1     NAME_1 VARNAME_1 NL_NAME_1 TYPE_1 ENGTYPE_1
## 260003   SGP Singapore SGP.1_1    Central      <NA>      <NA> Region    Region
## 260001   SGP Singapore SGP.2_1       East      <NA>      <NA> Region    Region
## 259999   SGP Singapore SGP.3_1      North      <NA>      <NA> Region    Region
## 260000   SGP Singapore SGP.4_1 North-East      <NA>      <NA> Region    Region
## 260002   SGP Singapore SGP.5_1       West      <NA>      <NA> Region    Region
##        CC_1 HASC_1
## 260003 <NA>   <NA>
## 260001 <NA>   <NA>
## 259999 <NA>   <NA>
## 260000 <NA>   <NA>
## 260002 <NA>   <NA>
```

```
class(SG@polygons) # list
```

```
## [1] "list"
```

```
# SG@polygons[1] # contains coordinate data about region boundaries

data = data.frame(Region=c("Central", "East", "North", "North-East", "West"),
                  value=c(3,4,1,7,10))
SG@data = merge(SG@data, data, by.x="NAME_1", by.y="Region")

# plot
spplot(SG, "value") # colours using values in the value column of SG@data
```



leaflet

leaflet is an open-source javascript library for interactive maps.

```
library("leaflet")
```

```
## Warning: package 'leaflet' was built under R version 4.1.1
```

```
library("tidyverse")
```

```
## Warning: package 'tidyverse' was built under R version 4.1.1
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

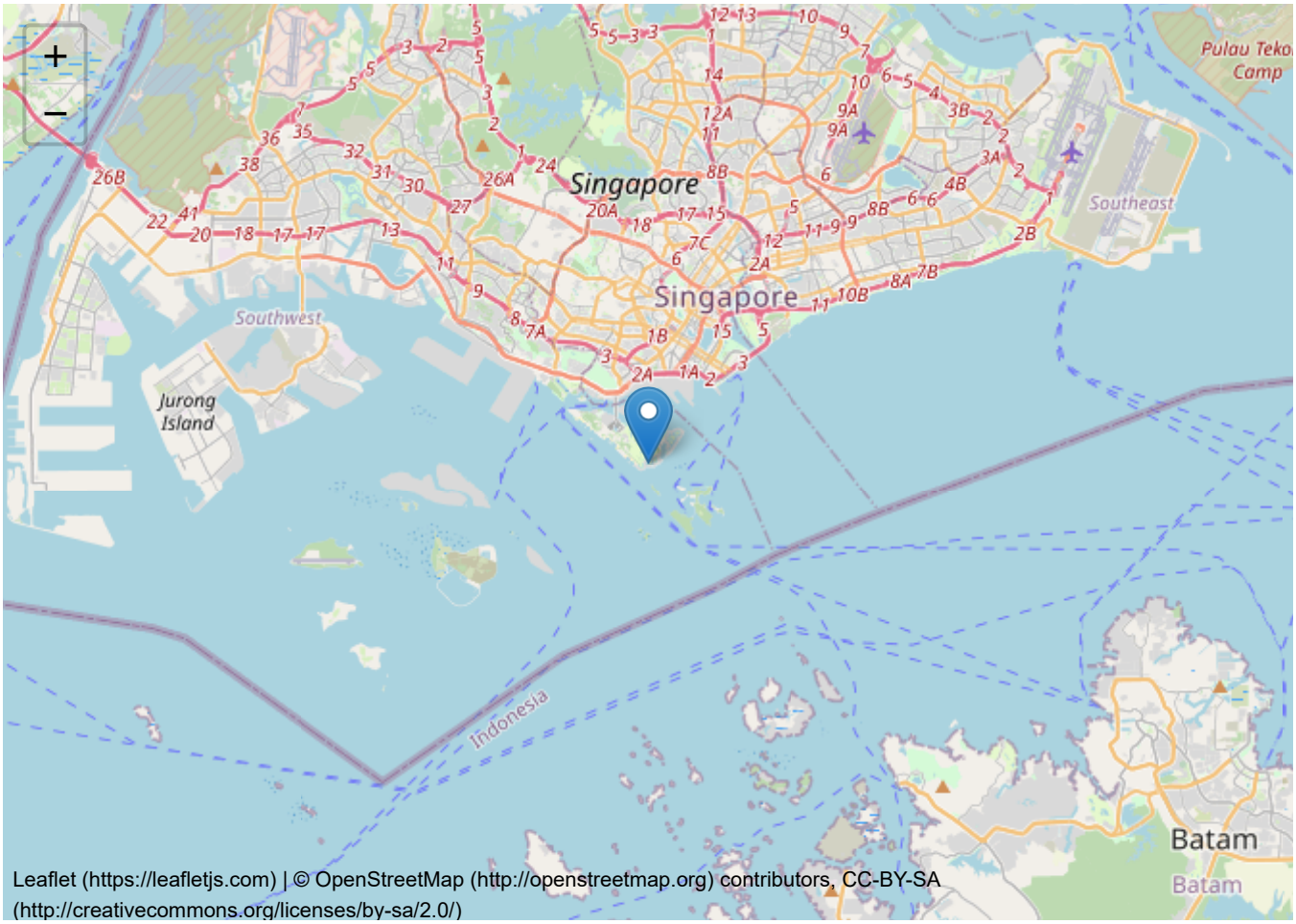
```
## v tibble  3.1.3     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1
## v purrr   0.3.4
```

```
## Warning: package 'readr' was built under R version 4.1.1
```

```
## Warning: package 'forcats' was built under R version 4.1.1
```

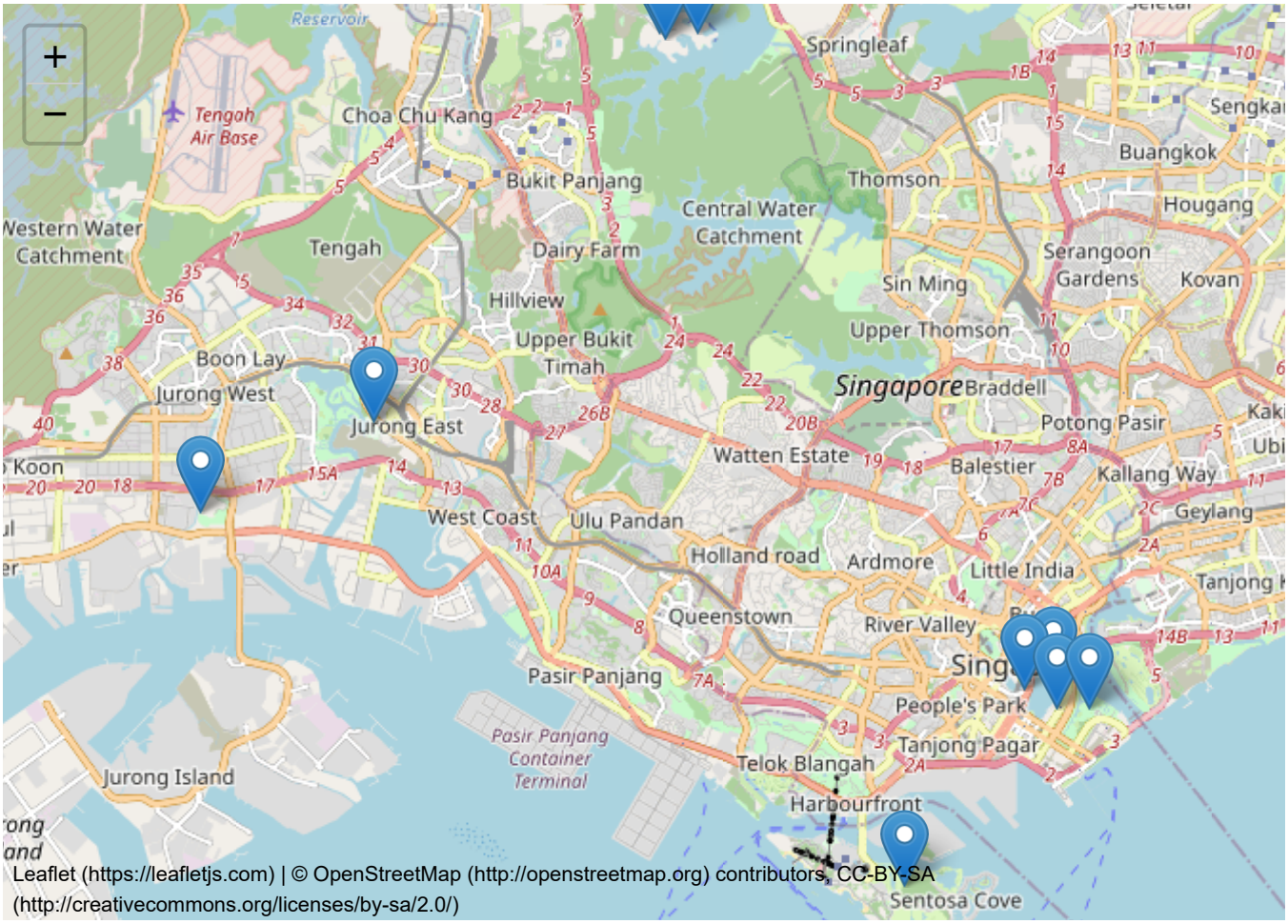
```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x tidyr::extract() masks raster::extract()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x dplyr::select()  masks raster::select()
```

```
m = leaflet()

# adding a map layer
m = addTiles(m)

# adding markers
m = addMarkers(m, lat=1.239660, lng=103.835381, popup="Sentosa Cove")

# preset zooming level
m = setView(m, lat=1.239660, lng=103.835381, zoom=11)

m
```



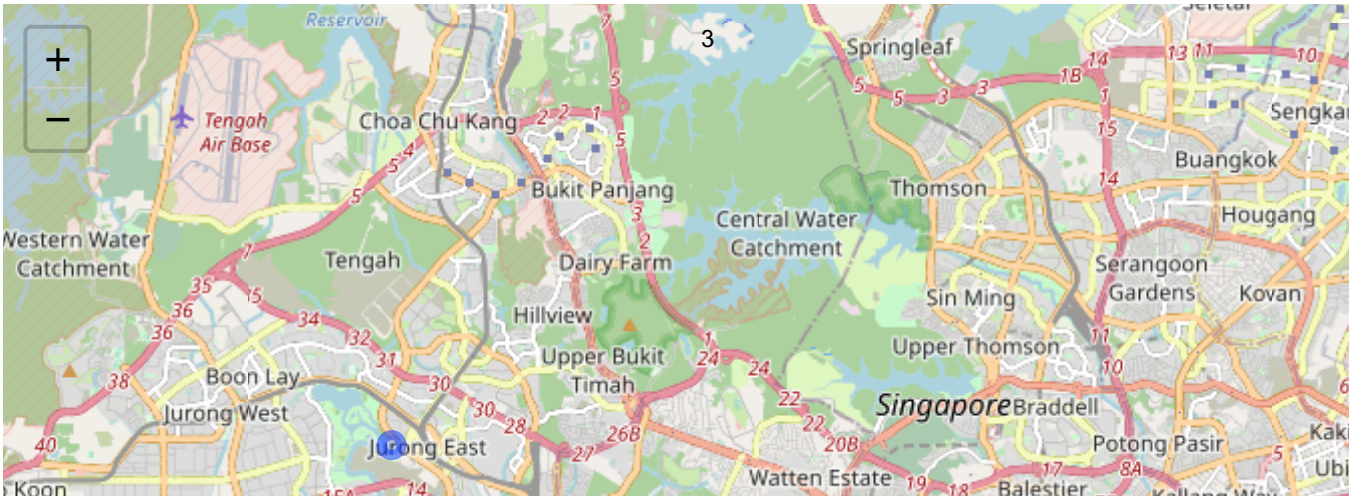
Adding Multiple Markers

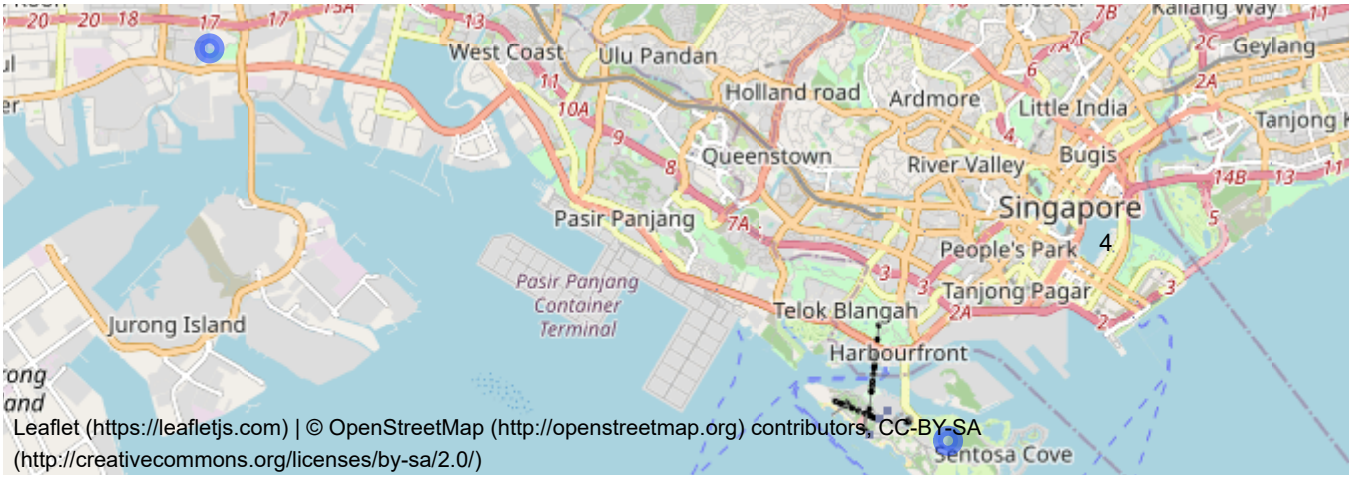
```
df = read.csv("./Data/Singapore_Tourist_Locations.csv", colClasses=c("numeric", "numeric", "c
haracter"))
leaflet() %>% addTiles() %>% addMarkers(data=df, lng=~longitude, lat=~latitude, popup=~name)
```



Adding Clusters

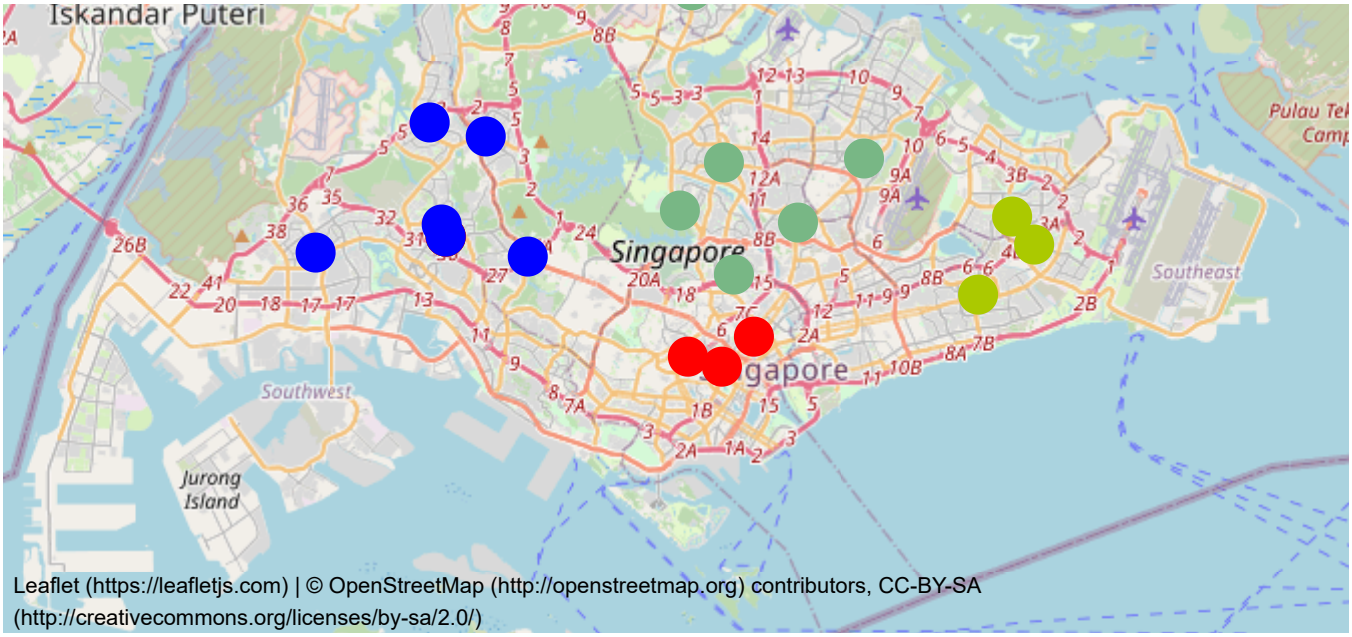
```
# showing clusters rather than individual points
leaflet() %>% addTiles() %>% addCircleMarkers(data=df, lng=~longitude, lat=~latitude,
                                              radius=5, clusterOptions=markerClusterOptions
(),
                                              popup=~name)
```





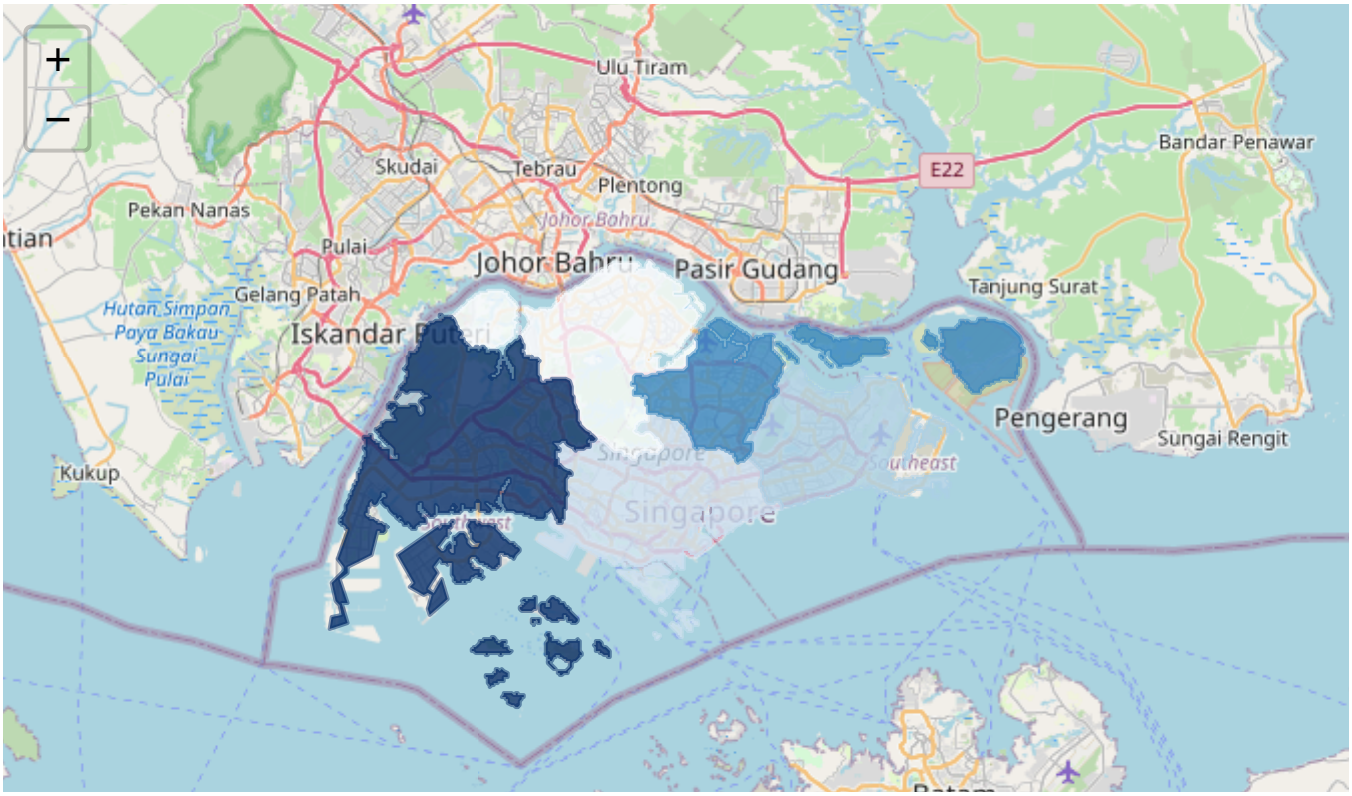
Overlaying Plots

```
library(htmlwidgets)

pizzahut.location = read.csv("./Data/PizzaHut.csv")
region.list = c("North", "East", "Central", "West")
colorFactors = colorFactor(c("red", "green", "blue"), domain=pizzahut.location$Region)

m = leaflet() %>% addTiles()

for (i in 1:4) {
  pizzahut.region = pizzahut.location[pizzahut.location$Region == region.list[i], ]

  m = addCircleMarkers(m, lng=pizzahut.region$lon, lat=pizzahut.region$lat,
                       radius=10, stroke=F, fillOpacity=1,
                       color = colorFactors(pizzahut.region$Region),
                       group=region.list[i]) # the group is to add the checkbox options later
}

# add map types
m = m %>% addTiles(group="Default") %>%
  addProviderTiles("Esri.WorldImagery", group="Esri") %>%
  addProviderTiles("Stamen.Toner", group="Toner") %>%
  addProviderTiles("Stamen.TonerLite", group="Toner Lite")

# add radio and checkbox controls
## base groups can only show 1 type at a time (radio)
## overlay groups can be shown together (checkbox)
m = addLayersControl(m, baseGroups=c("Default", "Esri", "Toner", "Toner Lite"),
                     overlayGroups=region.list)

saveWidget(m, file="m.html")
m
```

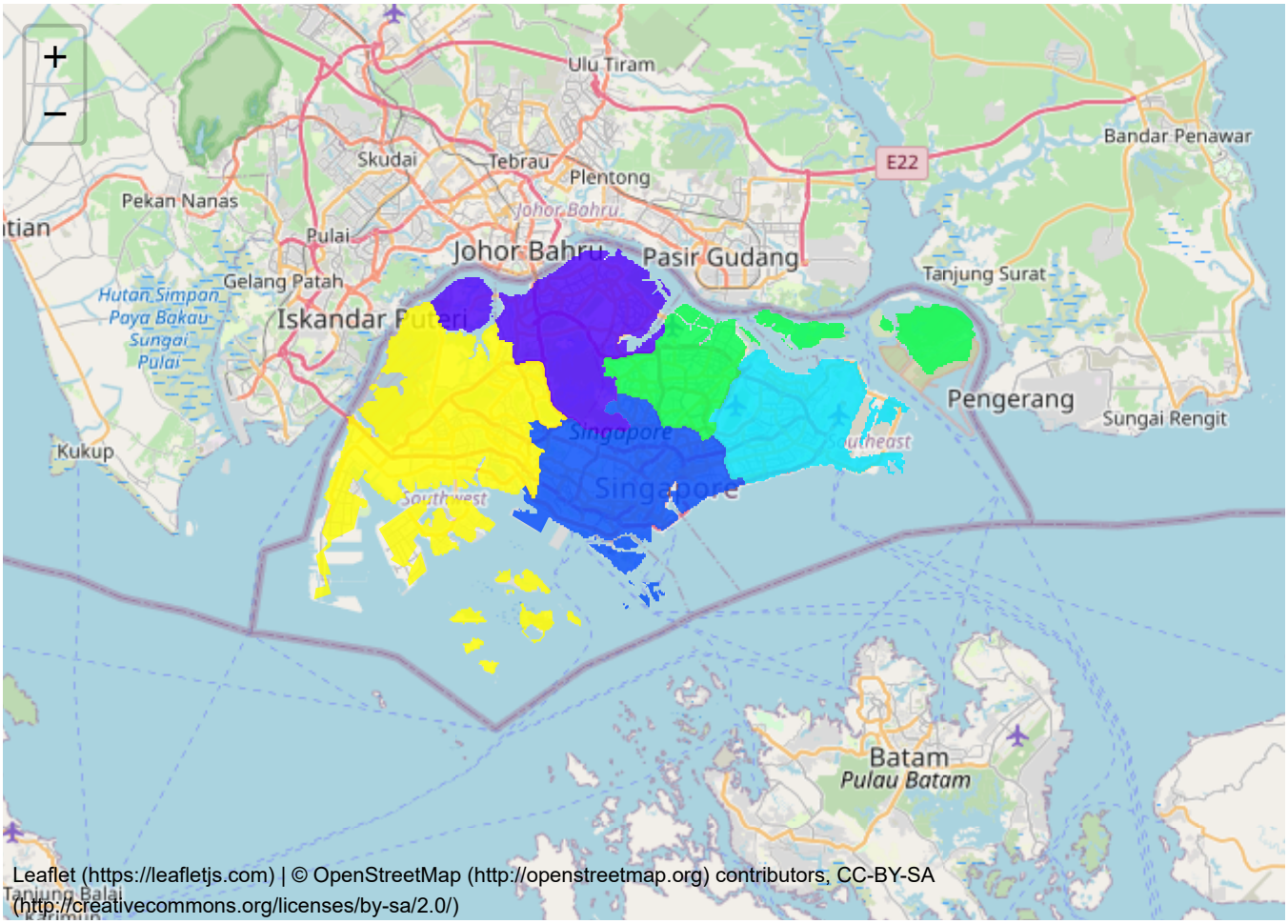


Plotting Polygons

```
library(raster)

data = data.frame(Region=c("Central", "East", "North", "North-East", "West"),
                  value=c(3,4,1,7,10))
SG = getData("GADM", country="SG", level=1)
SG@data = merge(SG@data, data, by.x="NAME_1", by.y="Region")
popup = paste0("<strong>Name: </strong>", SG$NAME_1)

# create numeric palette
pal = colorNumeric(palette="Blues", domain=SG$value)
m = leaflet() %>% addTiles() %>% addPolygons(data=SG, weight=2, stroke=T,
                                             smoothFactor=0.1, fillOpacity=0.8,
                                             color=~pal(value), popup=popup)

m
```




```
# create static palette
factpal = colorFactor(topo.colors(5), SG$value)
m = leaflet() %>% addTiles() %>% addPolygons(data=SG, weight=2, stroke=F,
                                             smoothFactor=0.2, fillOpacity=0.8,
                                             color=~factpal(value), popup=popup)
m
```

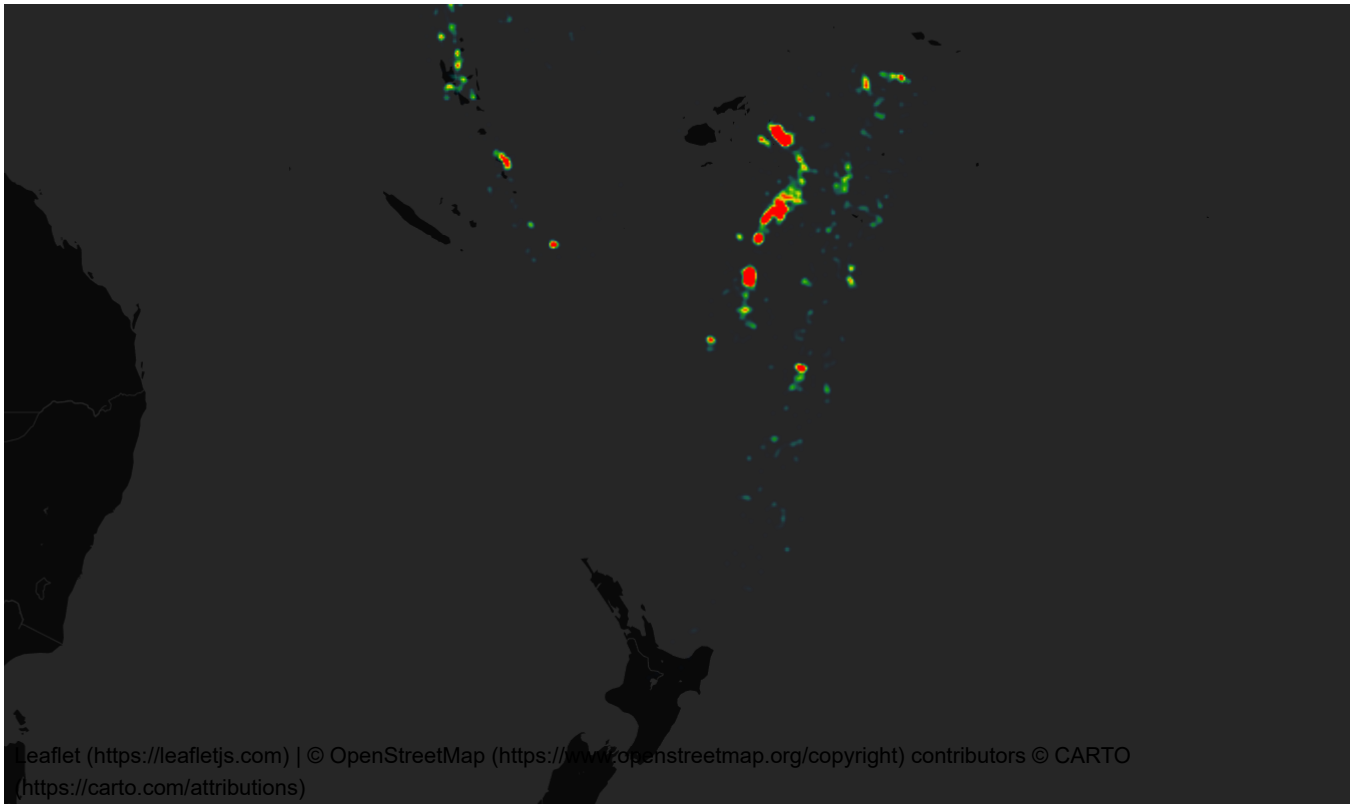


Generate Heatmap

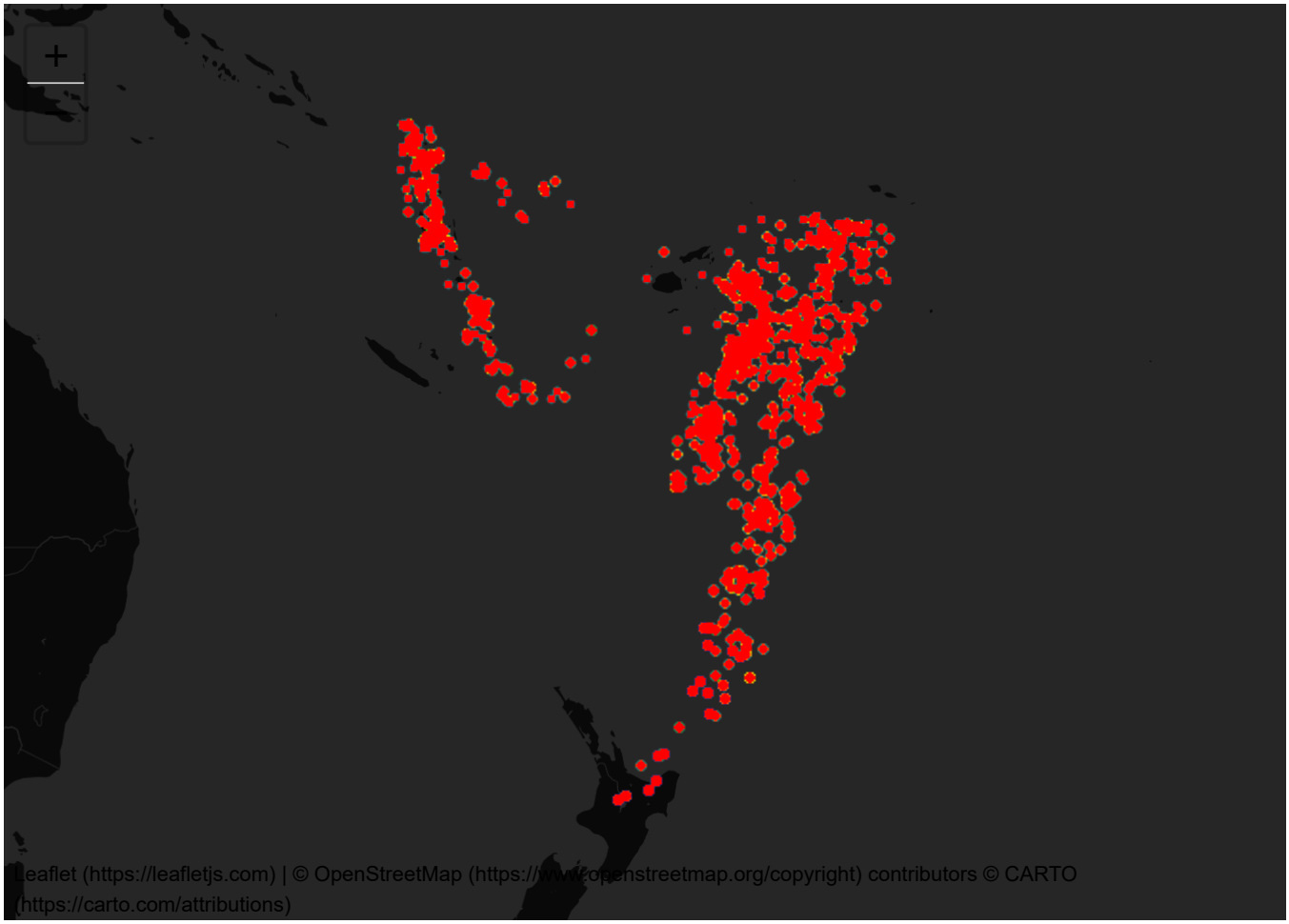
```
library("leaflet.extras")
```

```
## Warning: package 'leaflet.extras' was built under R version 4.1.1
```

```
# heatmap based on number of earthquakes in each area
leaflet(quakes) %>% addProviderTiles(providers$CartoDB.DarkMatter) %>%
  addWebGLHeatmap(lng=~long, lat=~lat, size=60000) # size is in meters
```




```
# heatmap based on eartquake magnitude
leaflet(quakes) %>% addProviderTiles(providers$CartoDB.DarkMatter) %>%
  addWebGLHeatmap(lng=~long, lat=~lat, intensity=~mag, size=60000)
```



Lecture 10

Shiny

Shiny apps consist of 2 components: 1. UI 2. Server

UI is the interface that users see and interact with. What users do on the UI are then sent to the server as an input. The server processes this then sends an output to the UI. The UI processes and shows this output to the user.

Creating a Shiny App

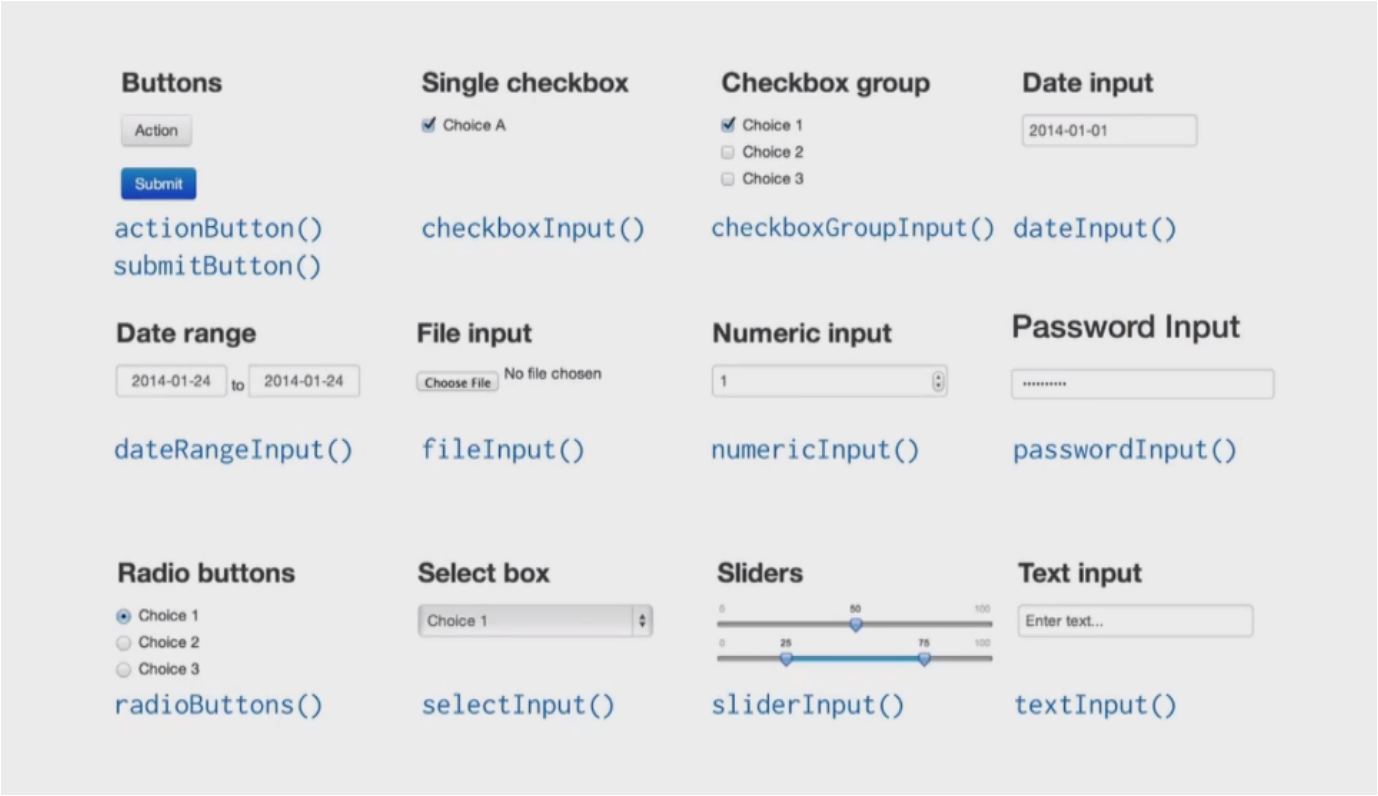
Create a R script called “app.R”. Note that R automatically recognises this as the base script for the app.

Start with this template:

```
library("shiny")

# Define UI
ui = fluidPage()

# Define server logic
server = function(input, output) {}

# Instantiate app
shinyApp(ui=ui, server=server)
```

1. UI

A typical UI usually consists of 2 parts: - Controls (allow user interaction with the system) - Outputs



Some controls in Shiny

Function	Inserts
dataTableOutput()	An interactive table
htmlOutput()	Raw HTML
imageOutput()	Image
plotOutput()	Plot
tableOutput()	Table
textOutput()	Text
uiOutput()	A shiny UI element
verbatimTextOutput()	Text
leafletOutput()	Leaflet map

Some outputs in Shiny

```
ui = fluidPage(sliderInput(inputId="zoomlevel", # needs an id to identify
                           label="Map Zooming Level",
                           value=11,
                           min=1,
                           max=20),
               leafletOutput(outputId="plotMap")) # also needs an output id
```

2. Server

User actions on UI controls trigger a event. The event passes the corresponding change to the server, which has listeners to react to specific events. The listener then produces the output we expect to see.

```
server = function(input, output){
  output$plotMap = renderLeaflet({ # handle the output
    leaflet() %>% setView(lng=-103,835381, lat=1.239660,
                          zoom=input$zoomlevel %>% # given zoomlevel input
                            addTiles() %>%
                            addMarkers(lng=-103,835381, lat=1.239660,
                                       popup="Sentosa Cove")
  })
}
```

User Experience

UI should be easy to use and look and feel good to use. We can control how our app looks using layouts.

Sidebar

```
ui = pageWithSidebar(
  # on the top
  headerPanel("How customer's choices on buying health insurance are affected?"),
  # on the side
  sidebarPanel(selectInput(inputId="attribute",
                           label="Customer Attribute",
                           choices=c("housing.type", "sex", "is.employed",
                                     "marital.stat")),
               radioButtons(inputId="position",
                            label="Bar Positioning",
                            choices=c("stack", "dodge", "fill"))),
  # the main display panel
  mainPanel(plotOutput("custPlot"))
)

server = function(input, output) {
  output$custPlot = renderPlot({
    ggplot(custdata) +
      # note that we must use aes_string since input$attribute is a string, not a column
      geom_bar(aes_string(x=input$attribute, fill="is.member"),
               position=input$position)
  })
}
```

Deploy on a Cloud

Apps can be hosted on a cloud to allow others to use them as well. This is done on shinyapps.io. On the app.R file, click publish app. Upload all relevant files into the same folder.