Intro to piping and Data Manipulation

JEPA

09/28/2017

Contents

Libraries and Data

```r
#If you downloaded tidyverse
#install.packages('tidyverse')
#library(tidyverse)

#install.packages('dplyr')
library(dplyr)

#install.packages('tidyr')
library(tidyr)

#install.packages('ggplot2')
library(ggplot2)

Alaska <- read.csv("./Data/Alaska.csv") #Sea around Us data for Alaska
USA <- read.csv("./Data/USAP.csv") #Sea around Us data for USA
```

Data Manipulation with Dplyr and Tidyr

Despite being separate, these two packages work together as one. Their main function is to manipulate data frames and keep things "tydi". In some cases you can also make basic data creation. Both packages follow the same syntax and can use the pipe operator, I normally don't even know which function is from what package so I often just call both.

Plus: Most functions are self explanatory like `select` or `filter`!

Dplyr

Arrange

The `arrange`function allows you to, literally, arrange your data by any value of a column

Basic structure:

New__Table <- arrange(Data, column__to__arrange__by)

Note: If you want to do from Top <- Bottom you can use `desc()` within the function

Note: when doing multiple variables the order is important since it will start with the first one

```
#You can arrange by characters (A -> Z)
Arrange_Example <- arrange(Alaska,common_name)

head(Arrange_Example[5:7], 3)
```

```
##   year scientific_name common_name
## 1 1964        Haliotis    Abalones
## 2 1964        Haliotis    Abalones
## 3 1966        Haliotis    Abalones
```

```
#You can arrange by characters (A <- Z) using desc()
Arrange_Example2 <- arrange(Alaska,desc(common_name))


head(Arrange_Example2[5:7], 3)
```

```
##   year   scientific_name         common_name
## 1 1984 Sebastes flavidus Yellowtail rockfish
## 2 1985 Sebastes flavidus Yellowtail rockfish
## 3 1987 Sebastes flavidus Yellowtail rockfish
```

```
# you can do multiple characters:
Arrange_Example3 <- arrange(Alaska,common_name,functional_group, desc(commercial_group))

head(Arrange_Example3[7:9],3)
```

```
##   common_name             functional_group commercial_group
## 1    Abalones Other demersal invertebrates         Molluscs
## 2    Abalones Other demersal invertebrates         Molluscs
## 3    Abalones Other demersal invertebrates         Molluscs
```

```
# And naturally, you can also arrange by numeric factors

Arrange_Example4 <- arrange(Alaska, uncertainty_score, desc(tonnes))
```

```
head(Arrange_Example4[4:6],3)
```

```
##   uncertainty_score year          scientific_name
## 1                 1 2010 Oncorhynchus tshawytscha
## 2                 1 1989 Oncorhynchus tshawytscha
## 3                 1 1988 Oncorhynchus tshawytscha
```

Filter

The `filter` function allows you to, literally, filter your data by any category or number.

Basic structure:

New_Table <- filter(Data, column_to_filter_by == “category”)

`filter` operators:

- `a == b` `a` is equal to `b`
- `a != b` `a` is not equal to `b`
- `a > b` `a` is greater than `b`
- `a < b` `a` is less than `b`
- `a >= b` `a` is greater than or equal to `b`
- `a <= b` `a` is less than or equal to `b`
- `a %in% b` `a` is an element in `b`

```
#You can filter by character
Filter_Example <- filter(Alaska,
                         common_name =="Clams")

head(Filter_Example[1:5], 5)
```

```
##                 area_name area_type                   data_layer
## 1 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
## 2 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
## 3 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
## 4 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
## 5 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
##   uncertainty_score year
## 1                 1 1950
## 2                 1 1951
## 3                 1 1952
## 4                 1 1953
## 5                 1 1954
```

```
#You can filter by numeric inputs too
Filter_Example2 <- filter(Alaska,
                         year == 2009)
head(Filter_Example2[1:5], 5)
```

```
##                 area_name area_type                   data_layer
## 1 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
## 2 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
## 3 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
## 4 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
## 5 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
##   uncertainty_score year
## 1                 4 2009
## 2                 4 2009
## 3                 2 2009
## 4                 4 2009
## 5                 4 2009
```

```
# Note: you can do =>, <= or !=

# you can do multiple characters:

Selection <- c("Clams","Octopuses")

Filter_Example3 <- filter(Alaska,
                          common_name %in% Selection)

head(Filter_Example3[4:8], 5)
```

```
##   uncertainty_score year scientific_name common_name
## 1                 1 1950        Bivalvia       Clams
## 2                 1 1950     Octopodidae   Octopuses
## 3                 1 1951        Bivalvia       Clams
## 4                 1 1951     Octopodidae   Octopuses
## 5                 1 1952        Bivalvia       Clams
##               functional_group
## 1 Other demersal invertebrates
## 2                  Cephalopods
## 3 Other demersal invertebrates
## 4                  Cephalopods
## 5 Other demersal invertebrates
```

```
# NOTE: remember that in R there are multiple ways to get to the same result!

#Wait! What if I want to filter by multiple columns!?

Filter_Example4 <- filter(Alaska,common_name == "Clams",
                            reporting_status =="Unreported") #Will give me all clams
# that are unreported

#You can also filter by NA

Filter_NA_Example1 <- filter(Alaska,is.na(uncertainty_score)) #Extract only NA's

head(Filter_NA_Example1[1:4],3)
```

```
##                  area_name area_type             data_layer
## 1 USA (Alaska, Subarctic)       eez Inferred foreign catch
## 2 USA (Alaska, Subarctic)       eez Inferred foreign catch
## 3 USA (Alaska, Subarctic)       eez Inferred foreign catch
##   uncertainty_score
## 1                NA
## 2                NA
## 3                NA
```

```
Filter_NA_Example2 <- filter(Alaska,!is.na(uncertainty_score)) #Clear NA's
```

Group_by* (plus summarise)

The group_byfunction allows you to group your data by common variables for future (immediate) calculations. This function needs the "pipe operator"

Basic structure:

New_Table <- Data %>% group_by(column_1,column_2…) %>% second_function()

```
#Simple group_by
Group_by_Example <- Alaska %>%
  group_by(common_name) %>%
  summarise(n()) #tells you how many rows of each "common_name"" you have

head(Group_by_Example, 3)
```

```
## # A tibble: 3 × 2
##      common_name `n()`
##           <fctr> <int>
## 1       Abalones    52
## 2  Alaska plaice     9
## 3 Alaska pollock   290
```

```
#Multiple
Group_by_Example2 <- Alaska %>%
  group_by(common_name,uncertainty_score) %>%
  summarise(n()) %>% #tells you how many rows of each "common_name"" you have
  arrange(uncertainty_score)

head(Group_by_Example, 3)
```

```
## # A tibble: 3 × 2
##      common_name `n()`
##           <fctr> <int>
## 1       Abalones    52
## 2  Alaska plaice     9
## 3 Alaska pollock   290
```

Mutate

The mutate function allows you to create a new column in the data-set. The new column can have characters or numbers.

Basic structure:

New_Table <- mutate(Data, Name_New_Column = action)

```
#Functions
Mutate_Example1 <- mutate(Alaska, Log = log(tonnes))

head(Mutate_Example1[13:16], 3)
```

```
##   reporting_status    tonnes landed_value      Log
## 1       Unreported   13.8030      20235.2 2.624886
## 2         Reported 1483.9740    2175505.9 7.302479
## 3       Unreported  389.9891     571724.0 5.966119
```

```
#In data calculations (per row)
Mutate_Example2 <- mutate(Alaska, Price_plus_Ton = (landed_value+tonnes))

head(Mutate_Example2[13:16], 3)
```

```
##   reporting_status    tonnes landed_value Price_plus_Ton
## 1       Unreported   13.8030      20235.2          20249
## 2         Reported 1483.9740    2175505.9        2176990
## 3       Unreported  389.9891     571724.0         572114
```

```
#Or characters...
Mutate_Example3 <- mutate(Alaska, Country = "USA")

head(Mutate_Example3[13:16], 3)
```

```
##   reporting_status    tonnes landed_value Country
## 1       Unreported   13.8030      20235.2     USA
## 2         Reported 1483.9740    2175505.9     USA
## 3       Unreported  389.9891     571724.0     USA
```

```
Mutate_Example4 <- mutate(Mutate_Example3, Country = paste("In",year,Country,"harvested",
                                                           round(tonnes,2), "tonnes of", common_name))

paste(Mutate_Example4[1,16])
```

```
## [1] "In 1950 USA harvested 13.8 tonnes of Marine fishes nei"
```

```
paste(Mutate_Example4[5387,16])
```

```
## [1] "In 1979 USA harvested 18.7 tonnes of Squids"
```

Rename

The `rename` function is another “self explanatory” it allows you to rename the columns

Basic structure:

New_Table <- rename(Data,New_Name = Old_Name)

```
Rename_Example <- rename(Alaska, Weight = tonnes)
```

Select

The `select`function is one of those “of-course it does that” function cus it allows you to, wait for it… SELECT any column you want.

Basic structure:

New_Table <- select(Data,number or name of column)

Note: Re-ordering of values happens here!

```r
#Select by column number
Select_Example1 <- select(Alaska, 6)

head(Select_Example1,3)
```

```
##                scientific_name
## 1 Marine fishes not identified
## 2 Marine fishes not identified
## 3 Marine fishes not identified
```

```r
#Select by multiple column numbers
Select_Example2 <- select(Alaska, 4,5,6,7)

head(Select_Example2, 3)
```

```
##   uncertainty_score year              scientific_name       common_name
## 1                 1 1950 Marine fishes not identified Marine fishes nei
## 2                 3 1950 Marine fishes not identified Marine fishes nei
## 3                 3 1950 Marine fishes not identified Marine fishes nei
```

```r
# You can also do (4:7) and even (4:6,15)

#Select by name
Select_Example3 <- select(Alaska, area_name,year,scientific_name,tonnes)

head(Select_Example3, 3)
```

```
##                  area_name year              scientific_name    tonnes
## 1 USA (Alaska, Subarctic) 1950 Marine fishes not identified   13.8030
## 2 USA (Alaska, Subarctic) 1950 Marine fishes not identified 1483.9740
## 3 USA (Alaska, Subarctic) 1950 Marine fishes not identified  389.9891
```

```r
# You can drop columns from a dataframe

Select_Example4 <- select(Select_Example3, -area_name,year)

head(Select_Example4, 3)
```

```
##   year              scientific_name    tonnes
## 1 1950 Marine fishes not identified   13.8030
## 2 1950 Marine fishes not identified 1483.9740
## 3 1950 Marine fishes not identified  389.9891
```

```r
#Note, you can also drop using -

#And you can also re-order your columns!

Select_Example5 <- select(Select_Example3, scientific_name,year,tonnes,area_name)

head(Select_Example5, 3)
```

```
##                scientific_name year    tonnes               area_name
## 1 Marine fishes not identified 1950   13.8030 USA (Alaska, Subarctic)
## 2 Marine fishes not identified 1950 1483.9740 USA (Alaska, Subarctic)
## 3 Marine fishes not identified 1950  389.9891 USA (Alaska, Subarctic)
```

```
#And you don't have to write everything

Select_Example6 <- select(Select_Example5, scientific_name,
                          everything())

head(Select_Example5, 3)
```

```
##                scientific_name year    tonnes               area_name
## 1 Marine fishes not identified 1950   13.8030 USA (Alaska, Subarctic)
## 2 Marine fishes not identified 1950 1483.9740 USA (Alaska, Subarctic)
## 3 Marine fishes not identified 1950  389.9891 USA (Alaska, Subarctic)
```

slice

The `slice`function works like the `select`function but for rows. So, if you want to extract an specific row, a set of rows, or a range between values, use slice!

Basic Structure

New_Data <- slice(Old_Data, number)

```
#Select by row number
Slice_Example1 <- slice(Alaska, 3948)

Slice_Example1
```

```
##                  area_name area_type                   data_layer
## 1 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
##   uncertainty_score year          scientific_name     common_name
## 1                 3 1973 Clupea pallasii pallasii Pacific herring
##                functional_group commercial_group fishing_entity
## 1 Medium pelagics (30 - 89 cm)    Herring-likes            USA
##   fishing_sector catch_type reporting_status  tonnes landed_value
## 1     Industrial   Landings         Reported 15792.9     23152391
```

```
#Select by multiple rows
Slice_Example2 <- slice(Alaska, 1000:3948)

head(Slice_Example2, 3)
```

```
##                  area_name area_type                   data_layer
## 1 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
## 2 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
## 3 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
##   uncertainty_score year          scientific_name     common_name
## 1                 3 1957  Hippoglossus stenolepis Pacific halibut
## 2                 1 1957  Hippoglossus stenolepis Pacific halibut
## 3                 3 1957 Clupea pallasii pallasii Pacific herring
##                functional_group commercial_group fishing_entity
## 1   Large flatfishes (>=90 cm)       Flatfishes            USA
## 2   Large flatfishes (>=90 cm)       Flatfishes            USA
## 3 Medium pelagics (30 - 89 cm)    Herring-likes            USA
##   fishing_sector catch_type reporting_status      tonnes landed_value
## 1      Artisanal   Landings         Reported 12564.60000  18419703.60
## 2   Recreational   Landings       Unreported    11.14694     16341.42
## 3     Industrial   Landings         Reported 53656.10001  78659842.61
```

Joining Data with dplyr

The "bind" family

These functions will help us bind two or more data-sets in one depending on different variables.

bind_cols

The `bind_cols` function allows us to bind two data-sets by column.

Basic Structure

New_Data <- bind_cols(Data1, Data2)

```
#Lets just asume that we have two different data sets
Data1 <- select(Alaska, 1)
Data2 <- select(Alaska, 2)

# View(Data2)



#Now we bind the columns together
Bind_Cols_1 <- bind_cols(Data1,Data2)

head(Bind_Cols_1, 3)
```

```
##                  area_name area_type
## 1 USA (Alaska, Subarctic)       eez
## 2 USA (Alaska, Subarctic)       eez
## 3 USA (Alaska, Subarctic)       eez
```

bind_rows

The `bind_rows` function is a sister-function of bind_cols but for binding rows.

Basic Structure

New_Data <- bind_rows(Data1, Data2)

```
#Lets just assume that we have two different data sets
Data1 <- slice(Alaska, 1:3)
Data2 <- slice(Alaska, 10800:10802)

#Now we bind the columns together
Bind_Row_1 <- bind_cols(Data1,Data2)

head(Bind_Row_1, 6)
```

```
##                  area_name area_type                  data_layer
## 1 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
## 2 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
## 3 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
##   uncertainty_score year              scientific_name       common_name
## 1                 1 1950 Marine fishes not identified Marine fishes nei
## 2                 3 1950 Marine fishes not identified Marine fishes nei
## 3                 3 1950 Marine fishes not identified Marine fishes nei
##                 functional_group       commercial_group fishing_entity
## 1 Medium demersals (30 - 89 cm) Other fishes & inverts            USA
## 2 Medium demersals (30 - 89 cm) Other fishes & inverts            USA
## 3 Medium demersals (30 - 89 cm) Other fishes & inverts            USA
##   fishing_sector catch_type reporting_status    tonnes landed_value
## 1    Subsistence   Landings       Unreported   13.8030      20235.2
## 2      Artisanal   Landings         Reported 1483.9740    2175505.9
## 3      Artisanal   Landings       Unreported  389.9891     571724.0
##                  area_name area_type                  data_layer
## 1 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
## 2 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
## 3 USA (Alaska, Subarctic)       eez Reconstructed domestic catch
##   uncertainty_score year    scientific_name common_name
## 1                 4 2009 Anoplopoma fimbria   Sablefish
## 2                 2 2009 Anoplopoma fimbria   Sablefish
## 3                 4 2009 Anoplopoma fimbria   Sablefish
##                  functional_group commercial_group fishing_entity
## 1 Large bathydemersals (>=90 cm)   Scorpionfishes            USA
## 2 Large bathydemersals (>=90 cm)   Scorpionfishes            USA
## 3 Large bathydemersals (>=90 cm)   Scorpionfishes            USA
##   fishing_sector catch_type reporting_status       tonnes landed_value
## 1     Industrial   Landings         Reported  1074.516856  1575241.711
## 2    Subsistence   Landings       Unreported     5.002588     7333.794
## 3      Artisanal   Landings         Reported 11175.083144 16382671.889
```

The "join" family

anti_join

This function will allow you to select all variables that are **not** the same within two data-sets. Note, both data-sets must have at least one similar category/column.

Basic Structure

Data_Name <- anti_join(Dataset1,Dataset2, by="similar category")

Lets us know what variables from one data-set are not present in some other data-set

```r
#Lets asume we want to know how many species are fished in Alaska
#and not in the continental US
Diff_Species <- anti_join(Alaska, USA, by="scientific_name")

#Lets assume we want to know how many species are fished in Alaska
#and not in the continental US
Similar_Species <- anti_join(Alaska, USA, by="scientific_name")
```

```r
#You can also do it by more than one variable
Diff_Species2 <- anti_join(Alaska, USA, by=c("scientific_name","reporting_status"))
```

semi_join

This function does the opposite as the anti join, letting you select those variables shared by two data-sets.

Basic Structure

Data_Name <- semi_join(Dataset1, Dataset2, by="similar category")

```r
#Now we want to know how many species are fished in BOTH Alaska and the continental US
Same_Species <- semi_join(Alaska, USA, by="scientific_name")

#Note: just like anti_join, you can do it for more than one variable
```

Inner_join

`Inner_join` will let you combine variables (rows) from different data-sets into one data-set based on a category/column that you choose

Basic Structure

Data_Name <- inner_join(Dataset1, Dataset2, by="similar category")

```r
#Now we want to know how many species are fished in BOTH Alaska and the continental US
Inner_Species <- inner_join(Alaska, USA, by="scientific_name")
```

```
## Warning in inner_join_impl(x, y, by$x, by$y, suffix$x, suffix$y): joining
## factors with different levels, coercing to character vector
```

```r
#Note: just like anti_join, you can do it for more than one variable

#Lets just asume that we have two different data sets
Data1 <- select(Alaska, 7,8)
Data2 <- select(Alaska, 7,11)

#Both Data 1 have two columns from witch one is "common_name".
# In the case of Data 1 the second column is "functional_group" and in the case of Data2 its "fishing_s

Inner_Example <- inner_join(Data1, Data2, by="common_name")

# The result will be a data-set with the "common_name",
# "functional_group" and "fishing_sector"

head(Inner_Example,3)
```

```
##         common_name              functional_group fishing_sector
## 1 Marine fishes nei Medium demersals (30 - 89 cm)    Subsistence
## 2 Marine fishes nei Medium demersals (30 - 89 cm)      Artisanal
## 3 Marine fishes nei Medium demersals (30 - 89 cm)      Artisanal
```

Left_join

Basic Structure

Data_Name <- left_join(Dataset1, Dataset2, by="similar category")

```
#Now we want to know how many species are fished in BOTH Alaska and the continental US
Left_Species <- left_join(Alaska, USA, by="scientific_name")
```

```
## Warning in left_join_impl(x, y, by$x, by$y, suffix$x, suffix$y): joining
## factors with different levels, coercing to character vector
```

```
#Note: just like anti_join, you can do it for more than one variable
```

Right_join

Basic Structure

Data_Name <- right_join(Dataset1, Dataset2, by="similar category")

```
#Now we want to know how many species are fished in BOTH Alaska and the continental US
Right_Species <- right_join(Alaska, USA, by="scientific_name")
```

```
## Warning in right_join_impl(x, y, by$x, by$y, suffix$x, suffix$y): joining
## factors with different levels, coercing to character vector
```

```
#Note: just like anti_join, you can do it for more than one variable
```

Tidyr

Gather and Spread

The `gather` function allows us to convert long data to short format. This is specifically helpful for plotting since it will allow you to set categories to data.

Note: The `spread` function is exactly the opposite to `gather` and has the same structure

Basic Structure

Data_Name <- gather(Dataset, key ="Some_Name", value ="Other_name", x:x)

```
# For example, if you want to have a divission between scientific and common name to plot
# the tonnes you'll do something like this:
Data1<- select(Alaska, 6,7,15)
Gather_Example <- gather(Data1, key='Name_Type', value='Species', 1:2)
```

```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```

```
head(Gather_Example,5)
```

```
##   landed_value       Name_Type                          Species
## 1    20235.198 scientific_name        Marine fishes not identified
## 2  2175505.884 scientific_name        Marine fishes not identified
## 3   571724.021 scientific_name        Marine fishes not identified
## 4    12045.723 scientific_name        Marine fishes not identified
## 5      664.751 scientific_name Miscellaneous aquatic invertebrates
```

Unite and Separate

These functions are used to unite or spread dates on a data-set

Basic Structure

Data_name <- separate(Data, TemporalColumn, c("year", "month", "day"), sep = "-")

Note: The date structure will depend on your data, as well as the `sep =`

```
#Assuming that our data set had a dat volumn with year/month/day this is how we would do it...
Separate_Example <- separate(Alaska,year,c("year", "month", "day"), sep = "-")

#Note: ignore the warning message, is because we don't have a month/day format

head(Separate_Example[5:7],3)
```

```
##   year month  day
## 1 1950  <NA> <NA>
## 2 1950  <NA> <NA>
## 3 1950  <NA> <NA>
```

```
# And then we can also go backwords

Unite_Example <- unite(Separate_Example,"Date",year, month, day, sep = "-")

head(Unite_Example[4:6],3)
```

```
##   uncertainty_score       Date               scientific_name
## 1                 1 1950-NA-NA Marine fishes not identified
## 2                 3 1950-NA-NA Marine fishes not identified
## 3                 3 1950-NA-NA Marine fishes not identified
```

```
#Note that, because month and day are NA's, the new column has them together
```

The Piping opperator %>%

Many R packages like `dplyr`, `tidyr` `ggplot2` and `leaflet`, allows you to use the pipe (`%>%`) operator to chain functions together. Chaining code allows you to streamline your workflow and make it easier to read.

When using the `%>%` operator, first specify the data frame that all following functions will use. For the rest of the chain the data frame argument can be omitted from the remaining functions.

NOTE: for Mac users the pipe symbol "%>%" shortcut is: command + shit + m. For windows users is: Ctrol + Shift + m

```
Pipie_Example <- Alaska %>%
  filter(year >= 2000) %>% #Lets filter the years above 2000
  select(area_name,scientific_name,tonnes,year) %>% #We only care about these data
  group_by(scientific_name,year) %>%
  summarise(Mean = mean(tonnes),
            SD = sd(tonnes),
            N = n()) %>% #Give me the mean and sd of each species each year
  mutate(Round_Mean = round(Mean,2), #create a log version of mean
         Round_SD = round(SD,2)) %>% #... and the sd
  transmute(Log_Mean = log(Round_Mean,2),
            Log_SD = log(Round_SD,2)) %>%
  semi_join(USA,
            by="scientific_name")
```

```
## Adding missing grouping variables: `scientific_name`
```

Combo!

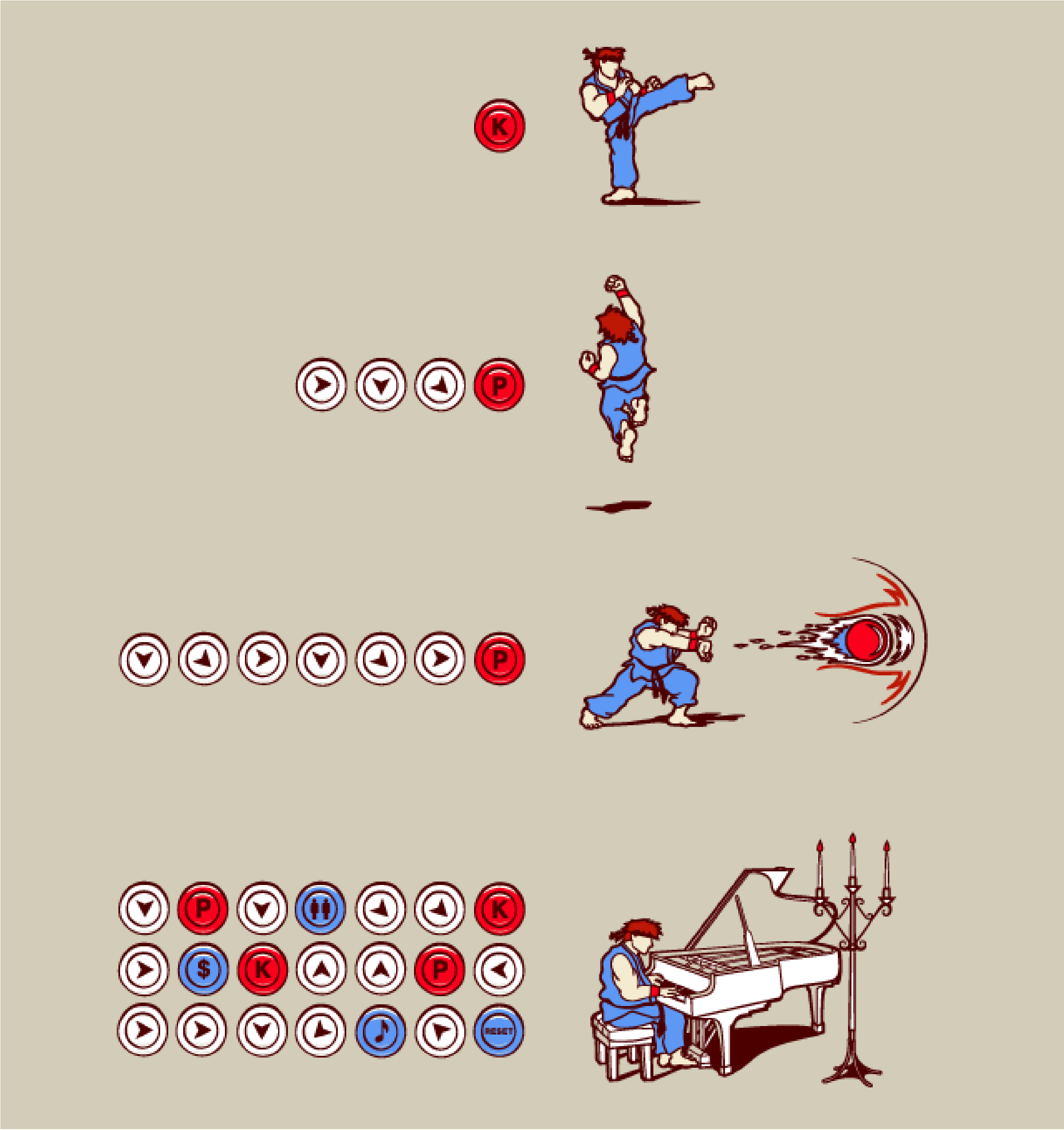
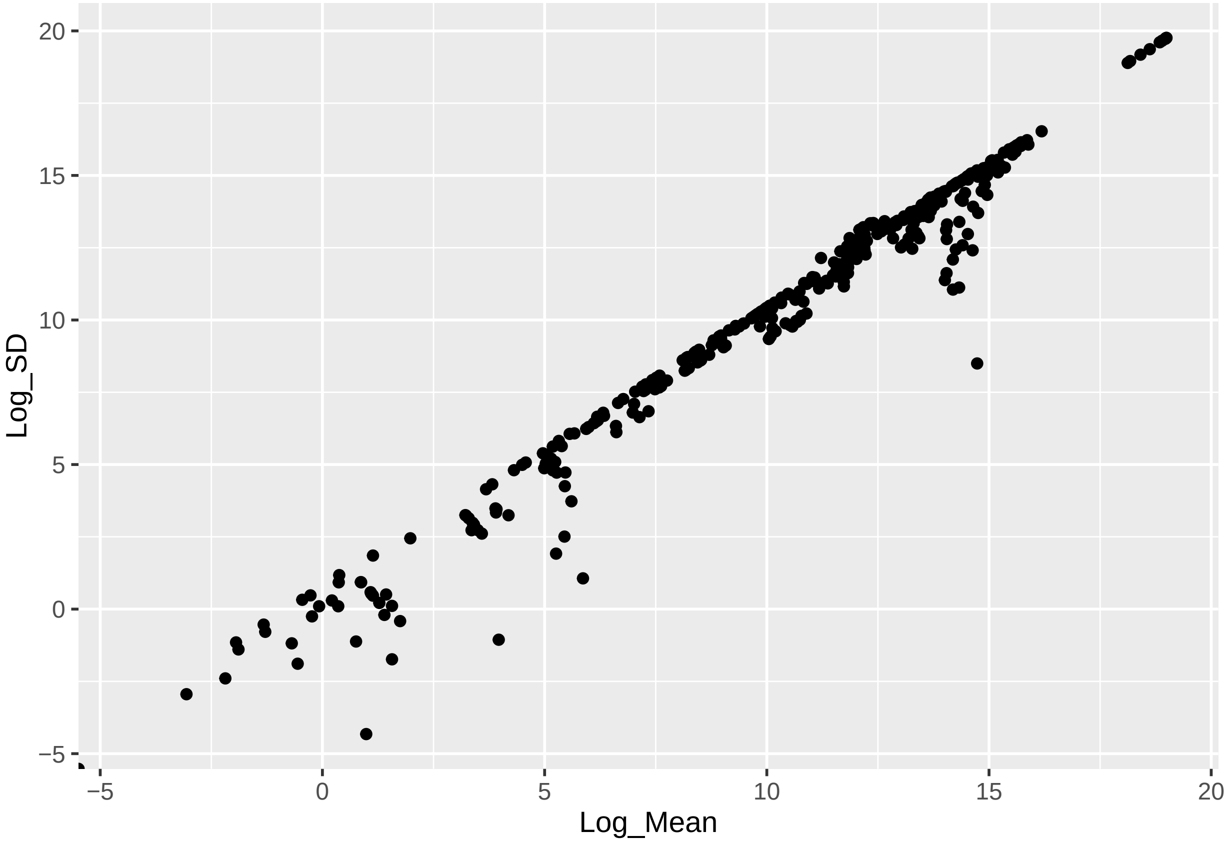


Figure 1: S

One of the beauties of `tydiverse` is that you can mix several packages in one code like this graph:

```
Pipie_Example <- Alaska %>%
  filter(year >= 2000) %>% #Lets filter the years above 2000
  select(area_name,scientific_name,tonnes,year) %>% #We only care about these data
  group_by(scientific_name,year) %>%
  summarise(Mean = mean(tonnes),
            SD = sd(tonnes),
            N = n()) %>% #Give me the mean and sd of each species each year
  mutate(Round_Mean = round(Mean,2), #create a log version of mean
         Round_SD = round(SD,2)) %>% # and the sd
  transmute(Log_Mean = log(Round_Mean,2),
            Log_SD = log(Round_SD,2)) %>%
  ggplot(., #It tells ggplot2 to use the data you are piping
         aes(
           x=Log_Mean,
           y=Log_SD
         )) +
  geom_point()

Pipie_Example
```


Log_SD
Log_Mean