Week 2 - Codebook

October 15, 2024

1 Week 2: Importing Data

1.1 Clear the entire workspace

[1]:
```
rm(list=ls())
```

1.2 Load required libraries

[3]:
```
ReqdLibs =␣
 ↪c("here","purrr","readxl","Hmisc","chron","ggplot2","ggthemes","dplyr")
invisible(lapply(ReqdLibs, library, character.only = TRUE))
```

1.3 Theme defaults

This is only for the Jupyter Notebook so you the figures axes are larger

[5]:
```
thm = theme(
          strip.text.x=element_text(size=20,face="bold"),
          strip.text.y=element_text(size=20,face="bold"),
          legend.text=element_text(size=16,face="bold"),
          legend.position = "top",
          legend.title=element_text(size=16,face="bold"),
          title =element_text(size=14, face='bold'),
          text = element_text(colour = "black",size=18),
          plot.title = element_text(colour = "black",size = 22, face = "bold"),
          axis.ticks.length = unit(0.3,"cm"),
          axis.line = element_line(colour = "black",size=0.85),
          axis.ticks = element_line(colour = "black",size=0.85),
          axis.text = element_text(colour = "black",size=24),
          axis.title=element_text(size=25))
```

1.4 Retrieve the folder path to the data files, a.k.a, the *Root Folder*

Note that this section is slightly different from the R project folder because the Jupyter notebook is outside teh R project folder and is read within the root folder `ReproRehab_Bootcamp` so we are going to need to direct it to the `Materials/Week 2/R project` folder.

[6]:
```
folder_path = here("Materials/Week 2/R project", "Raw Data")
# output the folder name
print(folder_path)
# make sure it exists
dir.exists(folder_path)
```

```
[1] "/Users/rinivarghese/Documents/My Documents/JHU/Lab/Professional
Development/ReproRehab 2024/ReproRehab_Bootcamp/Materials/Week 2/R project/Raw
Data"
```

TRUE

1.5 What subfolders and files are within the root folder? Let's check.

OK, once we have the *'folder path'* to the root folder, i.e., `Raw Data`, it is important to know exactly what the structure within is. In our case, the structure looks something like this: `Raw Data/` `sub1/` `sub1_rest.xlsx` `sub1_trial1.xlsx` `sub1_trial2.xlsx` ... `sub2/` `sub2_rest.xlsx` `sub2_trial1.xlsx` `sub2_trial2.xlsx` ... `sub3/` `sub3_rest.xlsx` `sub3_trial1.xlsx` `sub3_trial2.xlsx` ... ...

[7]:
```
options(warn=0)
subfolder_path = here(folder_path,'Sub1')

files.test=list.files(subfolder_path)
files.test

#Let's read in one file to see how ugly the data are
temp0=suppressMessages(read_excel(here(subfolder_path,files.test[1]),))
head(temp0)
```

1. 'Sub1_rest.xlsx' 2. 'Sub1_trial1.xlsx' 3. 'Sub1_trial2.xlsx' 4. 'Sub1_trial3.xlsx' 5. 'Sub1_trial4.xlsx' 6. 'Sub1_trial5.xlsx' 7. 'Sub1_trial6.xlsx'

A tibble: 6 × 129

ID code:	11	...3	Test number:	88	...6
<chr>	<chr>	<lgl>	<chr>	<chr>	<l
Last name:	SUBJECT	NA	Test date:	6/21/2017	NA
First name:	NO1	NA	Test time:	10:16	NA
Sex:	M	NA	N. of steps:	53	NA
Age:	26	NA	Duration (hh:mm:ss):	00:02:59	NA
Height (in):	72.834645669291334	NA	BSA (m^2):	2.1000404912839743	NA
Weight (lb):	189.59770013580487	NA	BMI (Kg/m^2):	25.127830533235937	NA

1.6 So, as you see above, the data are quite messy

we will modify the code so we only import a certain range Let's just do the first 5 rows where the data is in long format

[8]:
```
temp=suppressMessages(read_excel(here(subfolder_path,files.test[1]),range =␣
↪cell_cols("J:O")))
head(temp)
# the first two rows are header-like information so remove it
temp=temp[-c(1,2),-2]
head(temp)
```

A tibble: 6 × 6

t	...2	Rf	VT	VE	VO2
<chr>	<dbl>	<chr>	<chr>	<chr>	<chr>
hh:mm:ss	NA	b/min	l	l/min	ml/mi
0	NA	NA	NA	NA	NA
00:00:02	2	18.518518518518519	0.75485305361151733	13.978760252065134	408.69
00:00:05	3	20.979020979020977	0.55389892987980249	11.620257270205645	304.78
00:00:08	3	18.237082066869302	0.70588961229617564	12.873366789595908	347.70
00:00:10	2	25.531914893617021	0.78239498935139695	19.97604228131226	614.81

A tibble: 6 × 5

t	Rf	VT	VE	VO2
<chr>	<chr>	<chr>	<chr>	<chr>
00:00:02	18.518518518518519	0.75485305361151733	13.978760252065134	408.698704538219
00:00:05	20.979020979020977	0.55389892987980249	11.620257270205645	304.786751427508
00:00:08	18.237082066869302	0.70588961229617564	12.873366789595908	347.704321728538
00:00:10	25.531914893617021	0.78239498935139695	19.97604228131226	614.813454296942
00:00:13	21.978021978021978	0.49575484331783432	10.895710842150205	253.643090772051
00:00:16	18.927444794952681	0.78749534782174502	14.90527472217814	437.096660485628

1.7 Now we do this iteratively.

We go to our root folder `Raw Data` then we loop through all the subjects' folders within it to repeat the steps described above.

1.7.1 create a list of all the folder names within the root folder

[9]:
```
dir.list = dir(folder_path)
dir.list
```

1. 'Sub1' 2. 'Sub10' 3. 'Sub11' 4. 'Sub12' 5. 'Sub13' 6. 'Sub2' 7. 'Sub3' 8. 'Sub4' 9. 'Sub5' 10. 'Sub6' 11. 'Sub7' 12. 'Sub8' 13. 'Sub9'

1.7.2 create an empty data.frame that can accommodate any variable type

`data.frame(list())`: Converts the empty list into a data frame. Lists can accommodate any data type, i.e., numeric, string, characters, booleans etc. Since the list is empty, the resulting data frame will have no columns and no rows. In other words, this command initializes an empty data frame, which can later be filled with data, but starts with no content (i.e., no columns and no rows).

[10]:
```
data.all = data.frame(list())
```

1.8 Method 1: using a `for` loop to compile the master dataset

We use a for loop in R to iteratively "stack" individual participant data tables. For each participant, the steps are the samea as we did above.

[11]:
```
for(i in 1:length(dir.list)){
  files.import=list.files(here(folder_path,dir.list[i]))
  for(j in 1:length(files.import)){
    #Give me only the rows I need
    temp=suppressMessages(read_excel(here(folder_path,dir.list[i],files.
 ↪import[j]),
                                     range = cell_cols("J:O")))
    #Remove the random stuff
    temp=temp[-c(1,2),-2]
    #Convert to numeric
    temp[,c(2:5)]=apply(temp[,c(2:5)],2,as.numeric)
    #Covert to seconds
    temp$t <- seconds(times(temp$t)) + (minutes(times(temp$t)) * 60)
    #Assign Sub id
    temp$Sub=dir.list[i]
    #Assign trial id
    if(nchar(files.import[j])<16){
    temp$trial="rest"
    }else{
      temp$trial=paste("trial",as.numeric(substr(files.import[j],nchar(files.
 ↪import[j])-5,nchar(files.import[j])-5)))
    }
    # this final step is where the 'stacking' happens
    data.all=rbind(data.all,temp)

  }
}
```

1.8.1 Last step! Check the compiled dataset dimensions

[12]:
```
head(data.all)
dim(data.all)
```

A tibble: 6 × 7

t <dbl>	Rf <dbl>	VT <dbl>	VE <dbl>	VO2 <dbl>	Sub <chr>	trial <chr>
2	18.51852	0.7548531	13.97876	408.6987	Sub1	rest
5	20.97902	0.5538989	11.62026	304.7868	Sub1	rest
8	18.23708	0.7058896	12.87337	347.7043	Sub1	rest
10	25.53191	0.7823950	19.97604	614.8135	Sub1	rest
13	21.97802	0.4957548	10.89571	253.6431	Sub1	rest
16	18.92744	0.7874953	14.90527	437.0967	Sub1	rest

1. 15062 2. 7

1.9 Method 2: using `map_df` in the *purrr* package

`map_df` is used in two layers: * **Outer Layer**: Iterates over each directory in ***dir.list***, listing files within each directory. * **Inner Layer**: Processes each file in the current directory by: 1) reading the Excel file, 2) cleaning the data, 3) converting time to seconds, and 4) assigning identifiers.

This nesting efficiently combines results from all files into a single data frame, allowing for streamlined data aggregation from multiple directories.

[13]:
```
data.all <- map_df(dir.list, function(dir_name) {
  # List all files in the current directory
  files.import <- list.files(here(folder_path, dir_name))

  map_df(files.import, function(file_name) {
    # Read the Excel file
    temp <- suppressMessages(read_excel(here(folder_path, dir_name, file_name),␣
 ↪range = cell_cols("J:O")))

    # Clean the data
    temp <- temp[-c(1, 2), -2]
    temp[, 2:5] <- apply(temp[, 2:5], 2, as.numeric)

    # Convert time to seconds
    temp$t <- seconds(times(temp$t)) + (minutes(times(temp$t)) * 60)

    # Assign Sub id
    temp$Sub <- dir_name

    # Assign trial id
    temp$trial <- ifelse(nchar(file_name) < 16, "rest", paste("trial", as.
 ↪numeric(substr(file_name, nchar(file_name) - 5, nchar(file_name) - 5))))

    return(temp)
  })
})

# Now data.all contains all the processed data
head(data.all)
dim(data.all)
```

A tibble: 6 × 7

t <dbl>	Rf <dbl>	VT <dbl>	VE <dbl>	VO2 <dbl>	Sub <chr>	trial <chr>
2	18.51852	0.7548531	13.97876	408.6987	Sub1	rest
5	20.97902	0.5538989	11.62026	304.7868	Sub1	rest
8	18.23708	0.7058896	12.87337	347.7043	Sub1	rest
10	25.53191	0.7823950	19.97604	614.8135	Sub1	rest
13	21.97802	0.4957548	10.89571	253.6431	Sub1	rest
16	18.92744	0.7874953	14.90527	437.0967	Sub1	rest

1. 15062 2. 7

1.10 Method 3: combined power of R libraries

We now use `map_df` with 3 other packages: `lubridate`,`dplyr`, and `tidyr` to do all the same things as above, but less hard-coded (more modular).

[15]:
```
library(lubridate)
library(tidyr)

data.all <- map_df(dir.list, function(dir_name) {
  # List all files in the current directory
  files.import <- list.files(here(folder_path, dir_name))

  map_df(files.import, function(file_name) {

    # Read the Excel file
    temp <- suppressMessages(read_excel(here(folder_path, dir_name, file_name),␣
 ↪range = cell_cols("J:O"))) %>%

      # Clean and transform the data
      slice(-c(1, 2)) %>%                    # Remove the first two rows
      select(-2) %>%                         # Remove the second column
      mutate(across(2:5, as.numeric),        # Convert columns 2 to 5 to numeric
             t = as.numeric(hms(t)),         # Convert time to seconds
             id = file_name) %>%             # Assign file name as identifier
      separate(id,into = c("Sub","trial","extn"),sep = "[_\\.]") %>%  # Now␣
 ↪separate the identifier into sub & trial
      select(!c("extn"))                     # Don't need file extensions in the␣
 ↪table!

    return(temp)
  })
})

# Now data.all contains all the processed data
head(data.all)
dim(data.all)
```

A tibble: 6 × 7

t <dbl>	Rf <dbl>	VT <dbl>	VE <dbl>	VO2 <dbl>	Sub <chr>	trial <chr>
2	18.51852	0.7548531	13.97876	408.6987	Sub1	rest
5	20.97902	0.5538989	11.62026	304.7868	Sub1	rest
8	18.23708	0.7058896	12.87337	347.7043	Sub1	rest
10	25.53191	0.7823950	19.97604	614.8135	Sub1	rest
13	21.97802	0.4957548	10.89571	253.6431	Sub1	rest
16	18.92744	0.7874953	14.90527	437.0967	Sub1	rest

1. 15062 2. 7

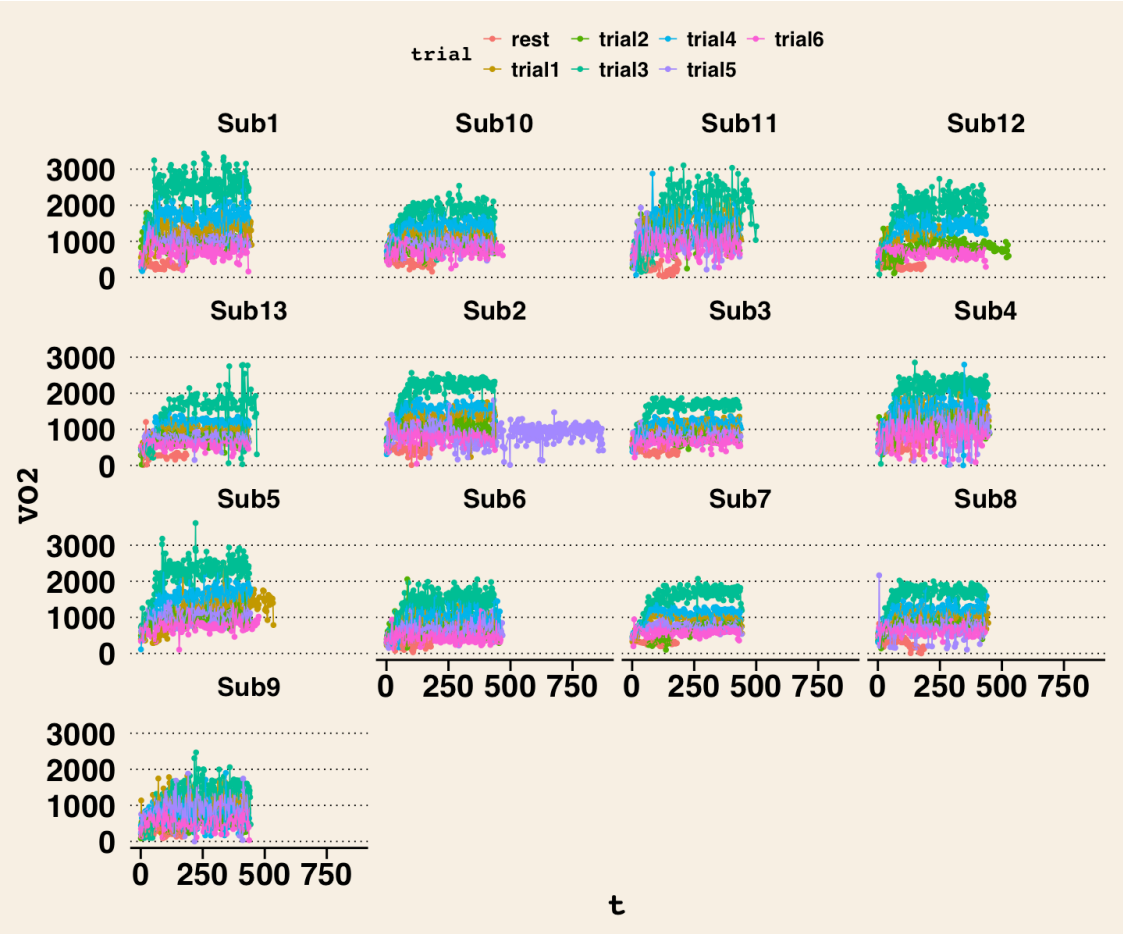
1.11 Visualization

Now that we have our compiled dataset, let's visualize it at different levels

1.11.1 Each timepoint for each trial for per participant

[16]:
```
options(repr.plot.width = 12, repr.plot.height = 10)

#Visualize raw data by subject
ggplot(data.all,aes(x=t,y=VO2,color=trial))+
  geom_point()+
  geom_line()+
  facet_wrap(~Sub) + theme_wsj() + thm
```

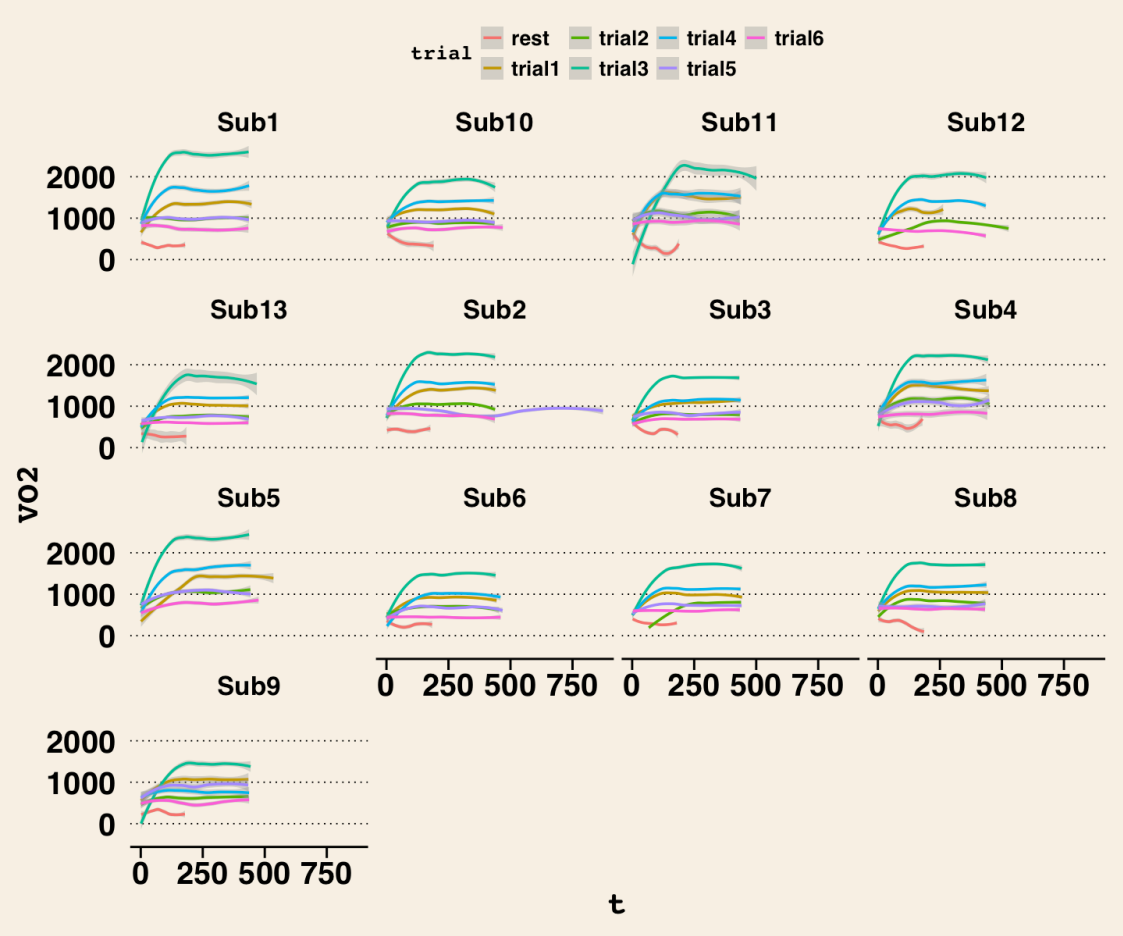


1.11.2 Smoothed traces across time for each trial per participant

In smoothing, we use a 'loess' procedure. This function is in-built within the geom_smooth function of ggplot. The 'loess' or 'lowess' procedure follows a kind of windowed smoothing procedure. It assigns weights to each point within the window, with closer points given higher weights, to calculate

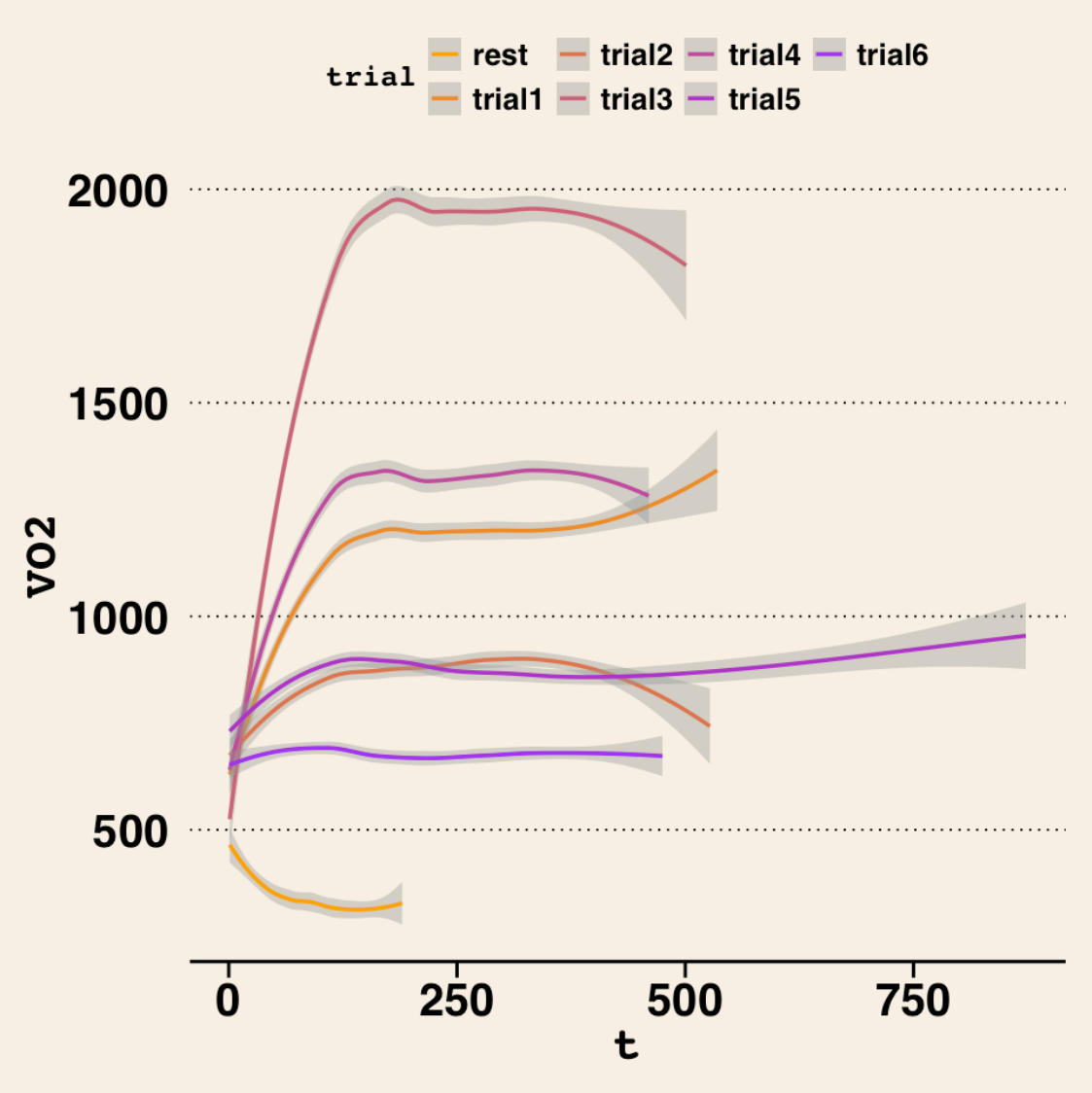
the smoothed value for each data point.

[17]:
```
#Visualize data using loess by subject
ggplot(data.all,aes(x=t,y=VO2,color=trial))+
  geom_smooth(method = 'loess', formula = 'y~x')+
  facet_wrap(~Sub) + theme_wsj() + thm
```



1.11.3 Each trial averaged across particpants

[18]:
```
options(repr.plot.width = 8, repr.plot.height = 8)
#create a color gradient
colfunc <- colorRampPalette(c("orange", "purple"))
#Visualize each trial across all participants
ggplot(data.all,aes(x=t,y=VO2,color=trial))+
  geom_smooth(method = 'loess', formula = 'y~x')+
  scale_colour_manual(values = c(colfunc(7))) + theme_wsj() + thm
```


trial
rest
trial1
trial2
trial3
trial4
trial5
trial6
2000
1500
1000
500
VO2
0
250
500
750
t

1.12 The End