Lunch and Learn and Quarto

Paul Villanueva

8/13/2021

Table of Contents

# 1 Lunch and Learn and Quarto

This is an example Quarto project made for RStudio’s Lunch and Learn on 8/3/2021. To learn more about Quarto visit <https://quarto.org>.

# 2 First Quarto

This is an example Quarto document. Note the qmd extension - this tells Quarto that this is a Markdown files that contains computations.

Since Quarto based on Markdown, we can **bold** and *italicize* text. We can also make headers.

# 3 Let’s make a table

| Meal | Food |
| --- | --- |
| Breakfast | Coffee |
| Lunch | Leftovers |
| Dinner | Spam Musubi |

# 4 Example images

Table :

|  |
| --- |
| My support system  Figure : My support system |

Table :

|  |  |
| --- | --- |
| Lunch: Leftovers  Figure : Lunch: Leftovers | Dinner: Spam musubi  Figure : Dinner: Spam musubi |

# 5 Code and chunk options

Quarto is based on **R**Markdown, so you can do all the R stuff you’re used to as well.

library(tidyverse)  
  
standard\_curves <- readxl::read\_xlsx('std\_curve.xlsx', sheet = "everything") %>%   
 janitor::clean\_names() %>%   
 filter(amoa < 40)  
  
lm\_eqn = function(df){  
 m = lm(log\_qty ~ ct, df);  
 data.frame(  
 a = format(as.numeric(coef(m)[1]), digits = 2),  
 b = format(as.numeric(coef(m)[2]), digits = 2),  
 r2 = format(summary(m)$r.squared, digits = 3)  
 )  
}  
  
st\_splits <- standard\_curves %>%   
 group\_by(amoa, run) %>%   
 group\_split()   
  
eqs <- st\_splits %>%   
 lapply(., lm\_eqn) %>%   
 bind\_rows()  
  
labels <- lapply(st\_splits, slice\_head, n = 1) %>%   
 bind\_rows() %>%   
 select(amoa, run) %>%   
 bind\_cols(eqs) %>%   
 mutate(amoa = paste0("amoA\_AOB\_p", amoa)) %>%  
 mutate(eq\_label = paste0("y = ", a, " - ", abs(as.numeric(b)), "x<br>r^2 = ", r2))  
  
  
standard\_curves %>%   
 mutate(amoa = paste0("amoA\_AOB\_p", amoa)) %>%   
 ggplot(aes(log\_qty, ct)) +  
 geom\_point() +  
 facet\_grid(run ~ amoa, scales = "free") +   
 theme(  
 panel.border = element\_rect(color = "black", size = 1, fill = NA),  
 panel.grid.minor.x = element\_blank(),  
 panel.grid.minor.y = element\_blank(),  
 panel.grid.major.x = element\_line(color = "gray", size = 0.5, linetype = "dashed"),  
 panel.grid.major.y = element\_line(color = "gray", size = 0.5, linetype = "dashed"),  
 panel.spacing = unit(0.5, "lines"),  
 panel.background = element\_blank(),  
 strip.background = element\_rect(color = "black", size = 1, fill = NA),  
 ) +   
 labs(  
 x = "Log(gene copies per reaction)",  
 y = "Ct"  
 ) +   
 scale\_x\_continuous(limits = c(0, 7), breaks = seq(0, 7, 1), expand = c(0, 0)) +   
 scale\_y\_continuous(limits = c(0, 25)) +  
 geom\_smooth(aes(group=1), method="lm", se=FALSE) +   
 ggtext::geom\_richtext(data = labels, aes(x = 3, y = 5, label = eq\_label),  
 size = 4, fontface = "bold", inherit.aes = FALSE)

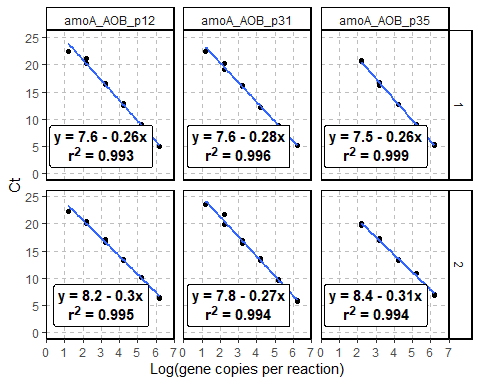


Figure : Figure 5.1: Standard curves for the LAMPS crop priming experiment

We can also throw some Python in here:

xs = [x for x in range(10)]  
  
print(\*(f'{x} squared is {x ^ 2}.' for x in xs), sep='\n')

0 squared is 2.  
1 squared is 3.  
2 squared is 0.  
3 squared is 1.  
4 squared is 6.  
5 squared is 7.  
6 squared is 4.  
7 squared is 5.  
8 squared is 10.  
9 squared is 11.

import matplotlib.pyplot as plt  
import numpy as np  
  
Z = np.random.rand(6, 10)  
x = [x + 0.5 for x in xs]  
y = np.arange(4.5, 11, 1)   
  
fig, ax = plt.subplots();  
ax.pcolormesh(x, y, Z)

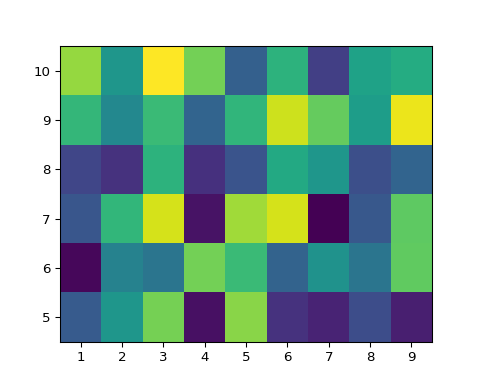


Figure : Figure 5.2: That’s a heatmap, baby!

## 5.1 Math stuff

We can also write math stuff! For example, here is a definition:

**Definition 5.1 (Continuity)** The function is *continuous at a point* if for all there exists such that if , then . If this is true for all such in the domain of , we say that is a *continuous function.*

#### 5.1 Example

Define by:

Prove that is not a continuous function.

**Proof.** Let and choose any . For any , we can find some such that since the irrationals are dense in . But then, showing that is not continuous at .

## 5.2 Adding references

We can also add references. For instance, the following definition of -partially colored comes from this paper: (Blair et al. 2020)

**Definition 5.2 (-partially colored)** Let be a diagram of a link with crossings. We call -*partially colored* if we have specified a subset of the strands of and a function . We refer to this partial coloring by the tuple . Given -partial colorings and of , we say is the result of a coloring move on if

1. and for some strand in ;
2. ;
3. is adjacent to at some crossing , and ;
4. the over-strand at is an element of ;
5. .

## 5.3 Cross-references

Along the way, we’ve been giving each of the items above labels. The Visual Editor knows about these labels and we can call them up for cross referencing. For example:

* We were pretty happy about the standard curves in [fig. 5.1](#fig-std-curves)
* I love me some heatmaps like [fig. 5.2](#fig-tiles)
* [def. 5.2](#def-merid-coloring) is trivially true for the unknot.

# 6 Bibliography

# 7 Second: A Pure Python qmd

This is a pure Python qmd document. Since there are no R code chunks, it is executed via the Jupyter kernel.

## 7.1 Adding days per month from date range to a dataframe

Suppose you have a dataset with a column of start dates and column of end dates. For example:

import pandas as pd  
import calendar  
  
date\_df = pd.DataFrame({  
 "START\_TM": ['2/15/2010', '2/15/2010', '3/16/2010'],  
 "END\_TM": ['4/18/2010', '2/18/2010', '5/20/2010']  
})  
date\_df["START\_TM"] = date\_df["START\_TM"].astype('datetime64')  
date\_df["END\_TM"] = date\_df["END\_TM"].astype('datetime64')  
date\_df

START\_TM END\_TM  
0 2010-02-15 2010-04-18  
1 2010-02-15 2010-02-18  
2 2010-03-16 2010-05-20

Our goal is to count the number of days in each month this range of dates falls over.

We start by adding columns for each month:

months = {calendar.month\_name[i]:[0 for \_ in range(date\_df.shape[0])] for i in range(1, 13)}  
for m in months:  
 date\_df[m] = [0 for \_ in range(date\_df.shape[0])]  
date\_df

START\_TM END\_TM January February March April May June July \  
0 2010-02-15 2010-04-18 0 0 0 0 0 0 0   
1 2010-02-15 2010-02-18 0 0 0 0 0 0 0   
2 2010-03-16 2010-05-20 0 0 0 0 0 0 0   
  
 August September October November December   
0 0 0 0 0 0   
1 0 0 0 0 0   
2 0 0 0 0 0

### 7.1.1 Helper functions

def insert\_days\_per\_month(outer\_row):  
 dpm = days\_per\_month(outer\_row)  
 for index, inner\_row in dpm.iterrows():  
 outer\_row[inner\_row['Month']] = inner\_row['NumDays']  
 return(outer\_row)  
  
def days\_per\_month(row):  
 s = pd.Series(index = pd.date\_range(row[0], row[1]))[1: ]   
 days\_in\_month = s.resample('MS').size().to\_period('m').\  
 rename\_axis('Month').reset\_index(name = 'NumDays')  
 days\_in\_month['Month'] = days\_in\_month['Month'].apply(  
 lambda x: calendar.month\_name[x.month])  
 return(days\_in\_month)

We can get the desired result with apply:

date\_df = date\_df.apply(lambda x: insert\_days\_per\_month(x), axis = 1)  
date\_df

DeprecationWarning:  
  
The default dtype for empty Series will be 'object' instead of 'float64' in a future version. Specify a dtype explicitly to silence this warning.

START\_TM END\_TM January February March April May June July \  
0 2010-02-15 2010-04-18 0 13 31 18 0 0 0   
1 2010-02-15 2010-02-18 0 3 0 0 0 0 0   
2 2010-03-16 2010-05-20 0 0 15 30 20 0 0   
  
 August September October November December   
0 0 0 0 0 0   
1 0 0 0 0 0   
2 0 0 0 0 0

## 7.2 EXTRA EXTRA READ ALL ABOUT IT HOT OFF THE PRESSES

Quarto has support for interactive documents. Support formats include:

* JavaScript: Observable JS
* R: Shiny
* Python: Jupyter Widgets are all supported, such as IPyLeaflet and Plotly

## 7.3 IPyLeaflet

from ipyleaflet import Map, Marker  
  
csg\_loc = (33.772819, -117.9694484)  
  
cham\_soot\_gol = Map(center=csg\_loc, scroll\_wheel\_zoom=True)  
cham\_soot\_gol.add\_layer(Marker(location=csg\_loc, title="Cham Soot Gol"))  
cham\_soot\_gol

Map(center=[33.772819, -117.9694484], controls=(ZoomControl(options=['position', 'zoom\_in\_text', 'zoom\_in\_titl…

Can do everything you’re used to with Python but with the awesome Visual Editor stuff:

## 7.4 Plotly

import plotly.express as px  
df = px.data.iris()  
fig = px.scatter(df, x="sepal\_width", y="sepal\_length",   
 color="species",   
 marginal\_y="violin", marginal\_x="box",   
 trendline="ols", template="simple\_white")  
fig.show()

Unable to display output for mime type(s): text/html

Unable to display output for mime type(s): text/html

# 8 Example Jupyter Notebook

The editing experience with Jupyter + Quarto is very similar to the RStudio editing experience.

I’m not lying!

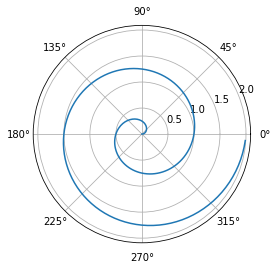
When we make changes and save here, the preview will update. Here’s some code:

for x in range(10):  
 print(f'{x} squared is {x ^2}.')

0 squared is 2.  
1 squared is 3.  
2 squared is 0.  
3 squared is 1.  
4 squared is 6.  
5 squared is 7.  
6 squared is 4.  
7 squared is 5.  
8 squared is 10.  
9 squared is 11.

Here’s a figure.

import numpy as np  
import matplotlib.pyplot as plt  
  
r = np.arange(0, 2, 0.01)  
theta = 2 \* np.pi \* r  
fig, ax = plt.subplots(subplot\_kw={'projection': 'polar'})  
ax.plot(theta, r)  
ax.set\_rticks([0.5, 1, 1.5, 2])  
  
ax.grid(True)  
plt.show()



And we can add chunk options just like we did in RStudio.

See [fig. 8.1](#fig-polar) for an example of a projection of a straight line into polar coordinates.

# 9 DNA String Stuff

Here are some functions to do some basic DNA string calculations.

import pandas as pd  
def reverse\_complement(nuc\_sequence: str) -> str:  
 """  
 Returns the reverse complement of a nucleotide sequence.  
 >>> reverse\_complement('ACGT')  
 'ACGT'  
 >>> reverse\_complement('ATCGTGCTGCTGTCGTCAAGAC')  
 'GTCTTGACGACAGCAGCACGAT'  
 >>> reverse\_complement('TGCTAGCATCGAGTCGATCGATATATTTAGCATCAGCATT')  
 'AATGCTGATGCTAAATATATCGATCGACTCGATGCTAGCA'  
 """  
 complements = {  
 "A": "T",  
 "C": "G",  
 "G": "C",  
 "T": "A"  
 }  
 rev\_seq = "".join([complements[s] for s in nuc\_sequence.upper()[::-1]])  
 return rev\_seq  
  
def gc\_content(nuc\_sequence: str) -> float:  
 """  
 Calculates the GC content of a nucleotide sequence.  
 >>> gc\_content('ACGT')  
 0.5  
 """  
 gc\_tally = 0  
 for nuc in nuc\_sequence.lower():  
 if nuc == 'g' or nuc == 'c':  
 gc\_tally += 1  
 return gc\_tally / len(nuc\_sequence)  
  
def random\_dna\_string(seq\_length: int = 10) -> str:  
 """  
 Generates a random DNA string seq\_length bp long  
 >>> len(random\_dna\_string())  
 10  
 >>> len(random\_dna\_string(20))  
 20  
 """  
 from random import choice  
   
 dna\_string = ""  
 for \_ in range(seq\_length):  
 dna\_string += choice("ACGT")  
 return dna\_string  
  
def make\_strings\_df(num\_strings: int = 10, str\_length: int = 10) -> pd.DataFrame:  
 """  
 Generates a pandas dataframe with num\_strings DNA sequences of length str\_length with   
 columns "Sequence", "GC Content", "Reverse Complement"  
 >>> df = make\_strings\_df(100, 37)  
 >>> df.shape  
 (100, 3)  
 >>> len(df['Sequence'][0])  
 37  
 """  
 dna\_strings\_list = [random\_dna\_string(str\_length) for \_ in range(num\_strings)]  
 strings\_df = pd.DataFrame({  
 "Sequence": dna\_strings\_list  
 })  
 strings\_df['GC Content'] = strings\_df['Sequence'].apply(gc\_content)  
 strings\_df['Reverse Complement'] = strings\_df['Sequence'].apply(reverse\_complement)  
 return strings\_df  
  
import doctest  
doctest.testmod(verbose=0)

TestResults(failed=0, attempted=9)

But that’s a lot of function definitions and code testing that a lot of people probably don’t care about. Let’s set fold and summary to hide this chunk.

Let’s use the function and create a histogram of the GC contents for the simulated sequences.

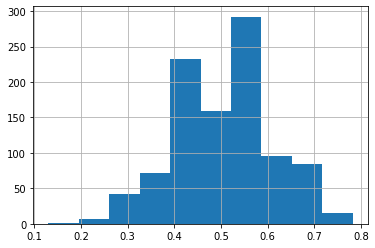
strings\_df = make\_strings\_df(1000, 23)  
print(f'strings\_df has {strings\_df.shape[0]} rows and {strings\_df.shape[1]} columns.')

strings\_df has 1000 rows and 3 columns.

strings\_df.head(10)

Sequence GC Content Reverse Complement  
0 TAATAATGGGCTAAACTATGTTT 0.260870 AAACATAGTTTAGCCCATTATTA  
1 GACCGTGACCCAAGGCAGATGGG 0.652174 CCCATCTGCCTTGGGTCACGGTC  
2 TAGGGTTGTGCTTTACCTTACAT 0.391304 ATGTAAGGTAAAGCACAACCCTA  
3 GCAAGGCCGGATACGCGTATAAT 0.521739 ATTATACGCGTATCCGGCCTTGC  
4 ACCACTCCTCAAACGTTACTGAT 0.434783 ATCAGTAACGTTTGAGGAGTGGT  
5 CCTCGTCAGTTGTCACTTCTATG 0.478261 CATAGAAGTGACAACTGACGAGG  
6 ACAATGATCGCAGCCGAGGTATA 0.478261 TATACCTCGGCTGCGATCATTGT  
7 GTTGGATATTCCGCAGCAGAGGA 0.521739 TCCTCTGCTGCGGAATATCCAAC  
8 CGCTTAAAATCCCTGCATAGACC 0.478261 GGTCTATGCAGGGATTTTAAGCG  
9 AGACCACTACTGGGTGGAGACGG 0.608696 CCGTCTCCACCCAGTAGTGGTCT

strings\_df['GC Content'].hist();



Blair, R., A. Kjuchukova, R. Velazquez, and P. Villanueva. 2020. “Wirtinger Systems of Generators of Knot Groups.” *Communications in Analysis and Geometry* 28 (2): 243–62. <https://doi.org/10.4310/cag.2020.v28.n2.a2>.