Lunch and Learn and Quarto

Paul Villanueva

8/13/2021

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.

Part I.

Section title

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

Figure 4.1.: My support system

Figure 4.2.: Lunch: Leftovers

Figure 4.3.: 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.

```r
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)
```

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.
```

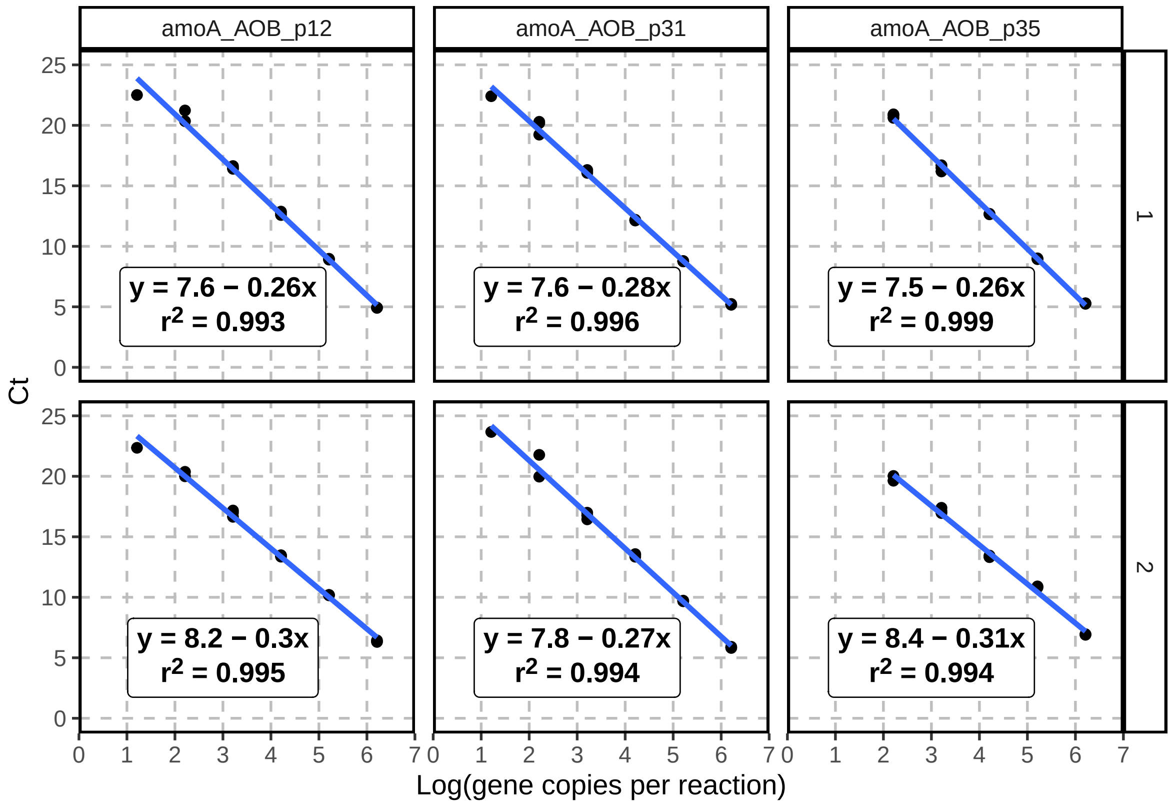


Figure 5.1.: Standard curves for the LAMPS crop priming experiment

```
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)
```

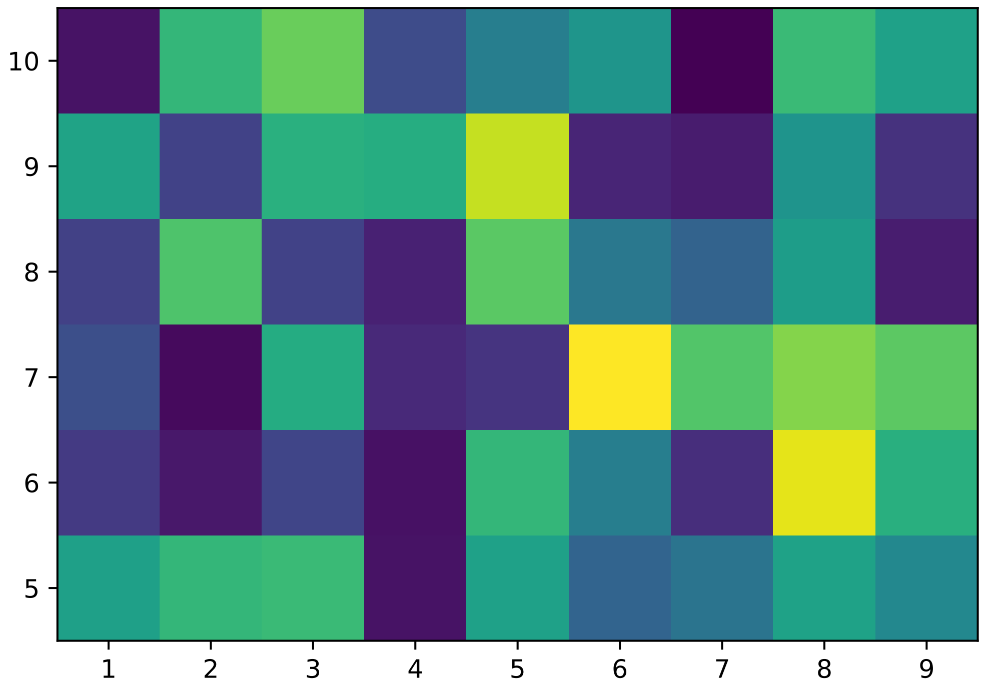


Figure 5.2.: That's a heatmap, baby!

5.1. Math stuff

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

Definition 5.1 (Continuity)**.** The function $f : \mathbb{R}^n \to \mathbb{R}^m$ is *continuous at a point* $x \in \mathbb{R}^n$ if for all $\varepsilon > 0$ there exists $\delta > 0$ such that if $|x - x_0| < \delta$, then $|f(x) - f(x_0)| < \varepsilon$. If this is true for all such x in the domain of f, we say that f is a *continuous function.*

5.1.0.1. Example

Define $f : \mathbb{R} \to \mathbb{R}$ by:

$$f(x) = \begin{cases} 1, x \in \mathbb{Q}, \\ 0, x \notin \mathbb{Q} \end{cases}$$

Prove that f is not a continuous function.

Proof. Let $\varepsilon = \frac{1}{2}$ and choose any $x \in \mathbb{Q}$. For any $\delta > 0$, we can find some $c \notin \mathbb{Q}$ such that $|x - c| < \delta$ since the irrationals are dense in $\mathbb{R}$. But then$|f(x) - f(c)| = |1 - 0| = 1 > \frac{1}{2}$, showing that f is not continuous at x. ■

5.2. Adding references

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

Definition 5.2 (k-partially colored)**.** Let D be a diagram of a link L with n crossings. We call D *k-partially colored* if we have specified a subset A of the strands of D and a function $f :\to \{1, 2, \ldots, k\}$. We refer to this partial coloring by the tuple (A, f). Given k-partial colorings (A_1, f_1) and (A_2, f_2) of D, we say (A_2, f_2) is the result of a coloring move on (A_1, f_1) if

1. $A_1 \subset A_2$ and $A_2 \ A_1 = \{s_j\}$ for some strand s_j in D;
2. $f_2|_{A_1} = f_1$;
3. s_j is adjacent to s_i at some crossing $c \in v(D)$, and $s_i \in A_1$;
4. the over-strand s_k at c is an element of A_1;
5. $f_1(s_i) = f_2(s_j)$.

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
- I love me some heatmaps like fig. 5.2
- def. 5.2 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	August	Septemb
0	2010-02-15	2010-04-18	0	0	0	0	0	0	0	0	
1	2010-02-15	2010-02-18	0	0	0	0	0	0	0	0	
2	2010-03-16	2010-05-20	0	0	0	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
```

	START_TM	END_TM	January	February	March	April	May	June	July	August	Septemb
0	2010-02-15	2010-04-18	0	13	31	18	0	0	0	0	
1	2010-02-15	2010-02-18	0	3	0	0	0	0	0	0	
2	2010-03-16	2010-05-20	0	0	15	30	20	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

```python
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_te
```

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

7.4. Plotly

```python
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()
```

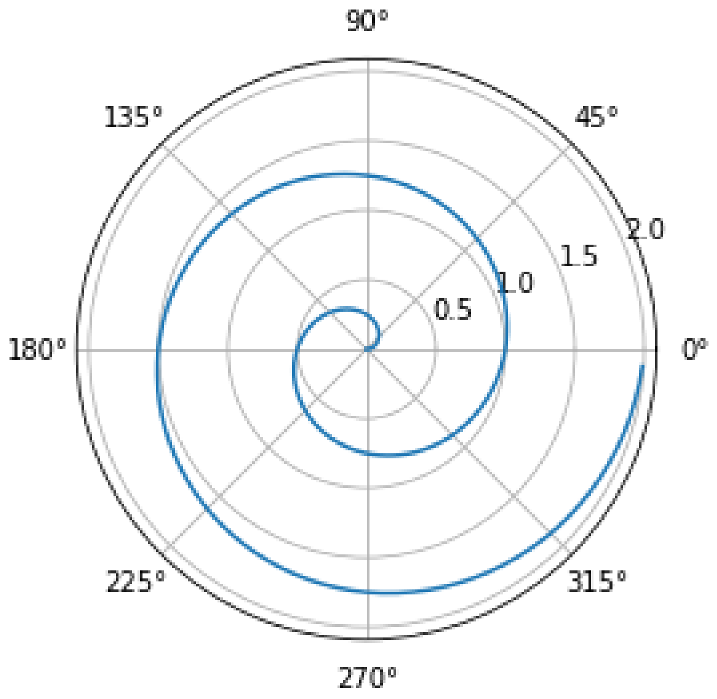


Figure 8.1.: **?(caption)**

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

See fig. 8.1 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.

```python
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.

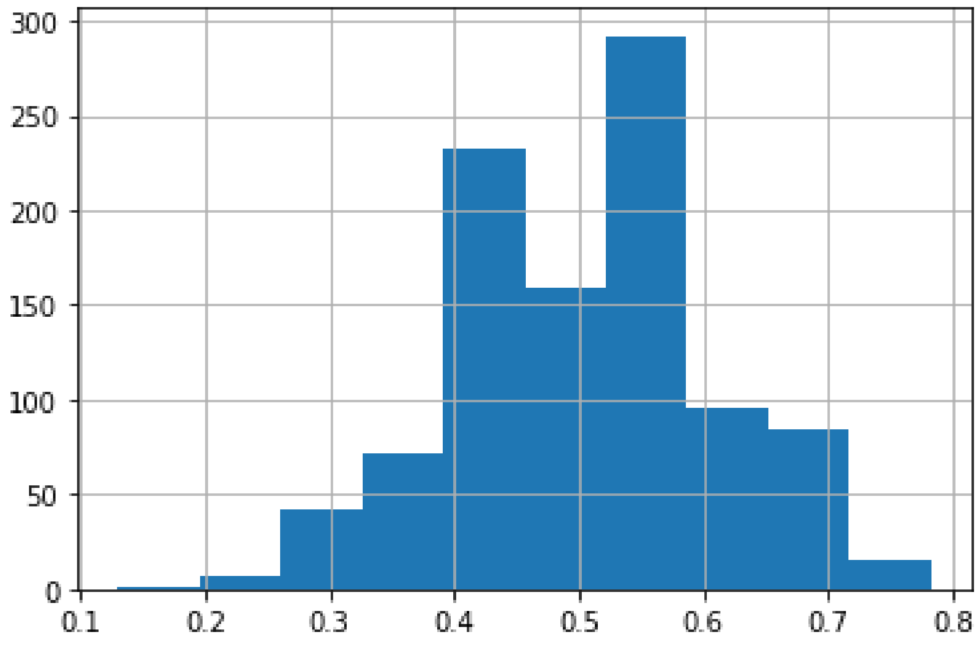
```
strings_df = make_strings_df(1000, 23)
print(f'strings_df has {strings_df.shape[0]} rows and {strings_df.shape[1]} columns.')
```

```
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.