Quarto

Exploration & Reference Document

Dominic DiSanto

Table of contents

Preface (YAML Options) & Set-Up

- Default chunk options can be set under the `execute` section of the YAML header
  - Affects both `r` and `python` chunks!
- Under format we've also included an option `df-print` to specify default behavior for displaying data frames, using `knitr::kable` (YAML Data Frame options)
- Below I also load some packages in R and Python that are used in chunks throughout this document:

R Packages:

```r
library(reticulate)
library(ggplot2)
library(magrittr)
library(palmerpenguins)
```

Python Modules

```
import seaborn as sns # violin plot, Example 3
import matplotlib.pyplot as plt # survival curves, Example 3
import numpy as np # used generally among python chunks
import pandas as pd # violin plot, Example 3
from matplotlib.ticker import ScalarFormatter
```

Misc. Quarto References

- General Quarto Reference Guide
- List of YAML document header options (PDF)
- Websites in Quarto
- RStudio
  - *Can include both R and Python code via* `reticulate` *package*
  - General formatting with R
  - Cell options & syntax (slightly different than normal RMD)
  - Figure generation, labelling/aliasing, calling (including multi plots)
  - Presentations (reveal.js and beamer included)
  - Advanced Writing Tips
    - General & Technical Formatting
    - Citations (useful to set-up Zotero, not yet completed)
- Jupyter/Python
  - I've not set-up/explored yet

Example 1 (LaTeX Writing and Equations)

Let us find the log-likelihood of a model of the form $logit(\pi_i) = \mathbf{x}_i^T\beta$, where $\pi_i = P(Y_i = 1)$ for $Y_i \sim Bernoulli(\pi_i)$. We know the likelihood function of our model:

$$L(\beta_0, \beta_1, |\mathbf{y}) = \prod_{i=1}^{n} \pi_i^{y_i}(1-\pi_i)^{1-y_i}$$

and the resulting log-likelihood, $\ell(\beta_0, \beta_1, |\mathbf{y})$ is:

$$\ell(\beta_0, \beta_1, |\mathbf{y}) = \sum_{i=1}^{n} y_i log(\pi_i) + (1-y_i) log(1-\pi_i)$$

Example 2 (Mingling Code)

Mingling Python & R

Installation

- Default/"smart" installation of miniconda using `reticulate` continued to fail, so I installed the miniconda package directly from GitHub (`remotes::install_github("hafen/rminiconda")`)
- Afterwards, used the package to install miniconda (`rminiconda::install_miniconda()`)
- Python chunk/reticulate failed first try, but then worked second try thereafter
- Able to execute python in RStudio console by running `reticulate::repl_python()`, and return back to R passing `exit`

R

```r
mean(c(0,2,3,4,6))
```

```
[1] 3
```

Python

```python
np.mean([0,2,3,4,6])
```

```
3.0
```

Example 3 (Figures, ggplot, base R, seaborn, and matplotlib)

Galton-Watson Survival Curves

Base R

Note I've included only the code relevant to visualization. The code that generates the survival curves is contained in an Appendix at the end of the document.

```
legend_text <- c(expression(paste(lambda, "=0.9")),
                 expression(paste(lambda, "=1.0")),
                 expression(paste(lambda, "=1.05")),
                 expression(paste(lambda, "=1.1")))

plot(l09$Generation, l09$SurvProb, type="b", lty=4, col="red", lwd=2
     , xlim = c(1, 500), ylim=c(0,0.7), log='x'
     , xlab="Generation", ylab="Survival Probability of Infection/Process"
     , main = "Galton-Watson Survival Curves for Poisson Brancing\nWith Differing Rate Par
lines(l1$Generation, l1$SurvProb, type="b", col="green", lwd=2)
lines(l105$Generation, l105$SurvProb, type="b", col="skyblue", lwd=2)
lines(l11$Generation, l11$SurvProb, type="b", col="blue", lwd=2)
legend(x = 50, y=0.5,
       col = c("red", "green", "skyblue", "blue"),
       lty = rep(13, 4), pch=rep(1, 4),
       legend = legend_text)
```

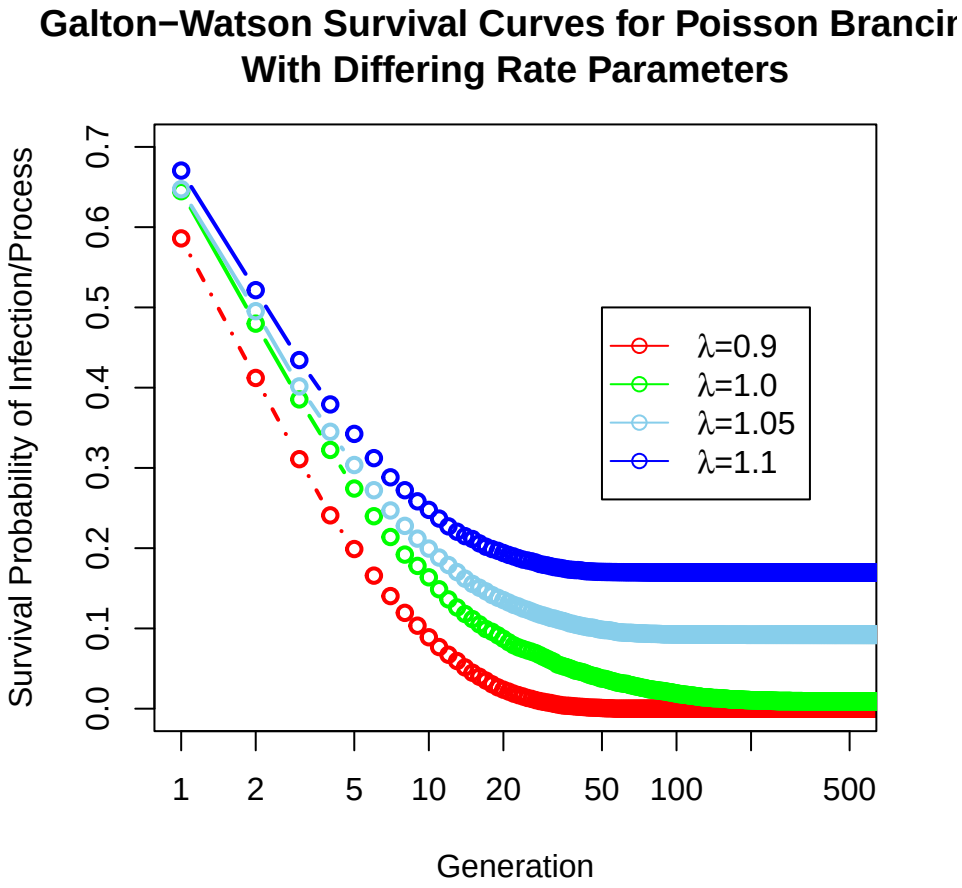


Figure 1: Galton-Watson Curve (Base R Plot)

matplotlib

We will inherit the R objects above and create the same plot using Python's `matplotlib`.

```
from matplotlib.ticker import ScalarFormatter

gs_fig, gs_ax = plt.subplots()

l09plt = gs_ax.plot(r.l09['Generation'], r.l09['SurvProb'], '.-', color='red')
l1plt = gs_ax.plot(r.l1['Generation'], r.l1['SurvProb'], '.-', color='green')
l105plt = gs_ax.plot(r.l105['Generation'], r.l105['SurvProb'], '.-', color='lightblue')
l11plt = gs_ax.plot(r.l11['Generation'], r.l11['SurvProb'], '.-', color='blue')

axlabs = gs_ax.set(xscale='log', xlabel="Generation"
          , ylabel="Survival Probability"
          , title="Galton-Watson Survival Curves for Poisson Brancing\nWith Differing Rate

leg = gs_ax.legend(['$\lambda$=0.9' , '$\lambda$=1' , '$\lambda$=1.05', '$\lambda$=1.1'])

for axis in [gs_ax.xaxis, gs_ax.yaxis]:
    axlab = axis.set_major_formatter(ScalarFormatter())

plt.show()
```

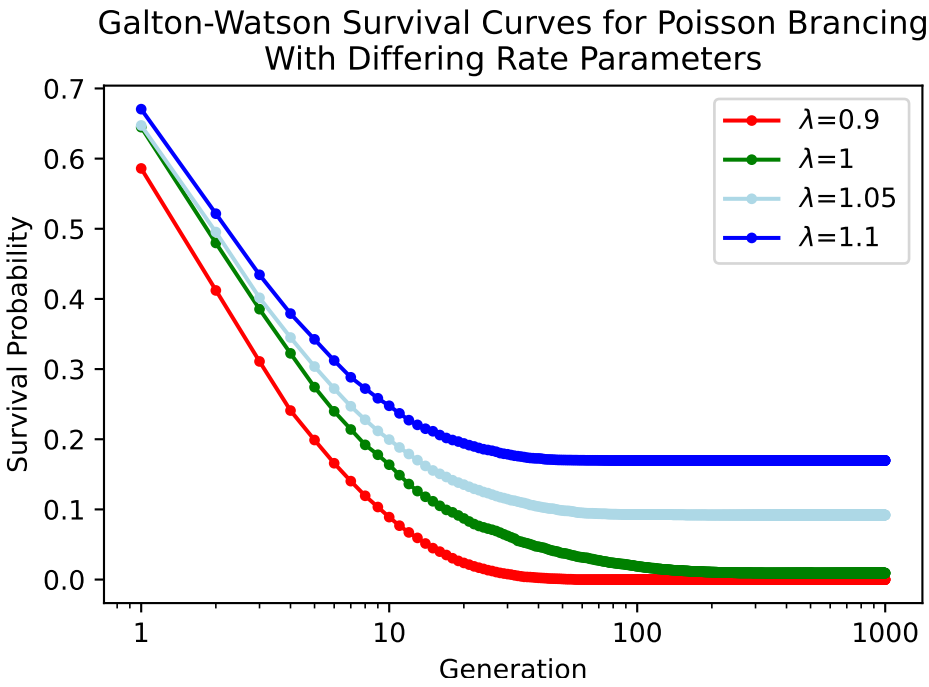
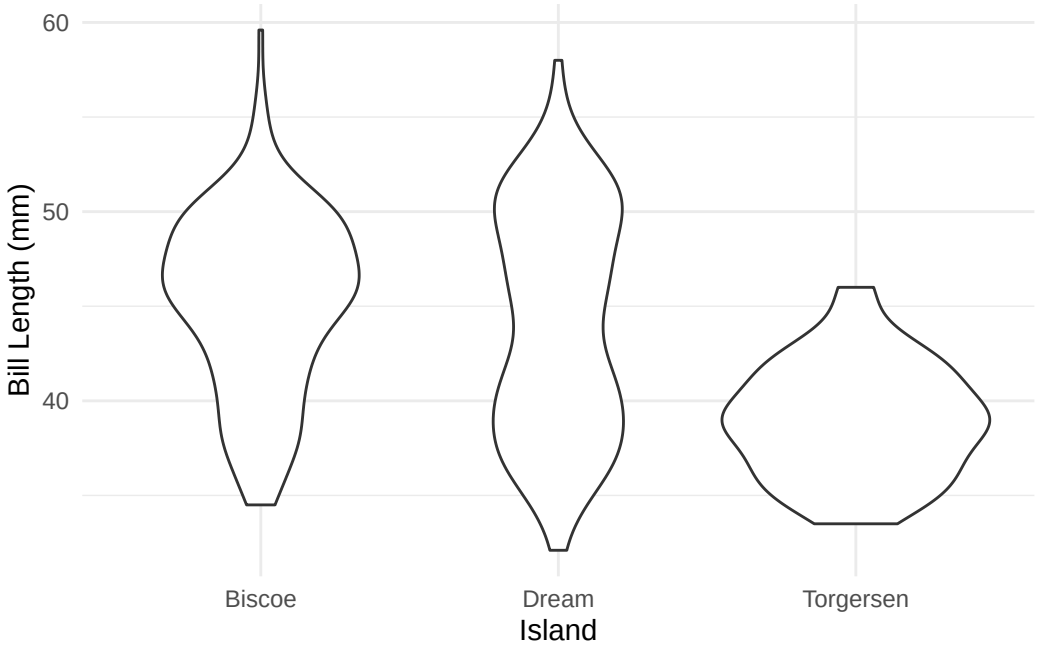


Figure 2: Galton-Watson Curve (Matplotlib Plot)

Violin Plots of Penguin Bill Length

ggplot

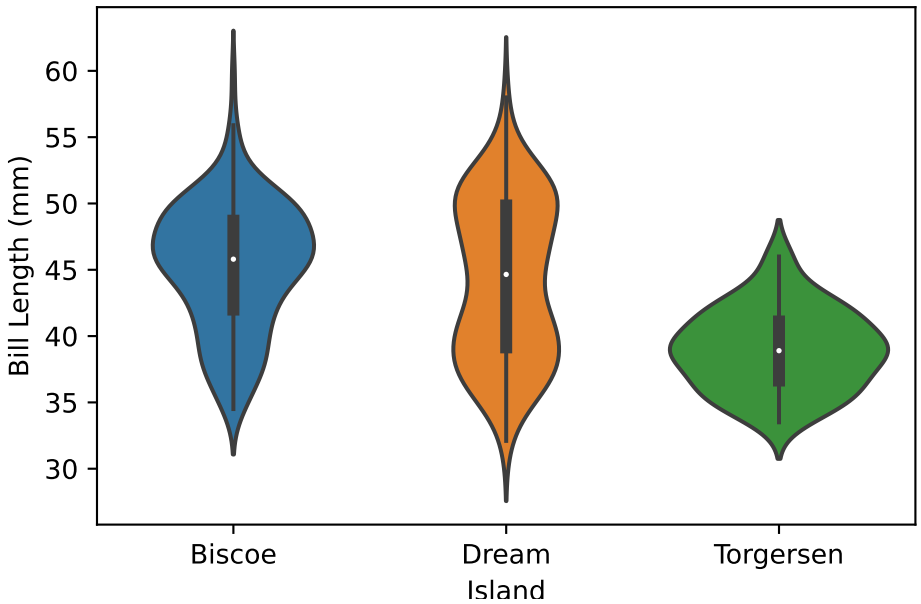
```
penguins %>%
  ggplot(aes(x=island, y=bill_length_mm)) +
  geom_violin() +
  theme_minimal() +
  xlab("Island") + ylab("Bill Length (mm)")
```



seaborn

The matplotlib code for a violin plot from a dataframe is especially ugly, so I have opted for seaborn for simplicity:

```
fig1_violin, ax1_violin = plt.subplots()
fig1_violin = sns.violinplot(x = 'island', y = 'bill_length_mm'
                             , data = r.penguins, ax = ax1_violin)
labs = ax1_violin.set(xlabel="Island", ylabel="Bill Length (mm)")
plt.show()
```



Example 4 (Tables)

Quarto has some wonderful point-and-click table insertion tools for Markdown tables in the Visual editor, that generates the corresponding Markdown syntax in the Source editor as well:

Table 1: GUI Generated Table (results are fabricated)

Model	Mean Bias	MSE
Left aligned	*Centered*	*Right aligned*
Logistic Regression	-0.235	1.23
Poisson Link	0.501	2.35
Firth Estimation	0.172	1.10

Below we simply call an R data frame (a preview of our penguins data set), that is displaying using `knitr::kable()` by default from our specified YAML options.

```
head(penguins[,c("species", "island", "bill_length_mm", "bill_depth_mm")])
```

species	island	bill_length_mm	bill_depth_mm
Adelie	Torgersen	39.1	18.7
Adelie	Torgersen	39.5	17.4
Adelie	Torgersen	40.3	18.0
Adelie	Torgersen	NA	NA
Adelie	Torgersen	36.7	19.3
Adelie	Torgersen	39.3	20.6

My knowledge of pretty Python tables is limited, but one can easily generate a table in Python and then call it from within an r chunk.

First we create the table in a chunk of Python code:

```
cases_df = pd.DataFrame({'Unit': [1, 2, 3], 'Cases_per_100k':np.random.poisson(64, 3)})
```

then call in an chunk of R code, which will again display as a kable:

```
py$cases_df
```

Unit	Cases_per_100k
1	46
2	66
3	60

Appendix (Survival Curve Code)

```r
poisson_bp <- function(L, generations, sims){

  extinct_gen <- rep(NA, sims)

  for (sim in 1:sims){
    x_t <- 1 # assume one infection occurred

    for (gen in 1:generations){

      (x_t <- sum(rpois(x_t, lambda=L)))

      if(x_t==0 & is.na(extinct_gen[sim])) extinct_gen[sim] <- gen

      if(x_t>100) {break} # escape if the population explodes, valid assumption?
    }
  }

  extinct_gen[is.na(extinct_gen)] <- Inf  # no extinction occurred


  surv_prob <- c()

  for (i in 1:generations){
      surv_prob <- c(surv_prob, mean(extinct_gen>i))
  }

  output <- data.frame(Generation = 1:generations,
                       SurvProb = surv_prob)

  return(output)

}

```

```r
nsim <- 10000
ngen <- 1000

l09 <- poisson_bp(0.9, generations = ngen, sims = nsim)
l1 <- poisson_bp(1, generations = ngen, sims = nsim)
l105 <- poisson_bp(1.05, generations = ngen, sims = nsim)
l11 <- poisson_bp(1.1, generations = ngen, sims = nsim)
```