Technical Interview Notes

Josh Allen

2025-02-04

Table of contents

Preface

This is a structured notebook for technical interviews. I will keep this up to date as much as possible

A Note to Keep Yourself Focused

These notes spend a lot of time going through the some of the technical components of the job eg models and techniques to do your job. All of these are important to actually implement what you want but not neccesarilly why you will be hired. While not neccessarily the same as what you do in causal inference it is generally important to define your estimand.

What do we mean by this? Lundberg, Johnson, and Stewart (2021) argue that before you touch the data we should first go ahead and define an estimand or what your model is actually estimating. By doing this at the beggining we are freeing ourselves from the constraints of various models or getting bogged down on the part that some data scientists care about the most which seems to be make models go brrrrrrrr.

There are two components to the estimand:

- Unit specific quantity: What is the counterfactual?
- What is the effect of introducing new cards into circulation? Would would person i be less likely to have experienced credit card fraud if they had received the new card?
- The target population of interst: Who does this number apply too?
- Does this quantity of interest apply to every person with a debit card or a city where we roll out a new measure?

Underriding this estimand is a very clear research question. If we think about the team or division that is major research agenda. For this job interview that you are preparing for the teams goal is to identify and reduce fraud. That is a huge question people have been doing fraud since 300 BC when two merchants sank their ship to collect insurance money. The first documented bank fraud happened in 193 AD when the Praetorian guard sold the emperorship something they didn't actually own.

However, there are lot of little bites of the puzzle that we can take. Lets take identity theft for example. We can't just stop identity theft because well people have been trying it forever and well for the most it has worked. However, there are lots of little interventions we can make.

Lets say somebody steals the social security number of somebody tries to open a credit card. In an olden time this may have worked. However, if we are doing analysis of cases where we detected fraud we may be interested in whether changes in location rapidly is a good predictor of this.

A well constructed research question is how does location change velocity effect credit card fraud? Where the theoretical estimand would credit fraud if velocity took on a specific value. The population would be the US population that opens credit cards. We may find that location velocity has no effect because fraudsters turn off the location on their services or use a VPN. We may want to actually force people to not be able to do this or force more verification.

Careful consideration of the who the intervention applies to is informative because we can more clearly define what a targeted roll out in one place tells us about other places or why we spent resources studying a weird case.

Your job is not to be the best python programmer or best statistician. Your job is to ultimately define these estimands and apply the best tools to faithfully represent these estimands. This implies a ton of work on the front and back end.

1 Data Viz with Matplot

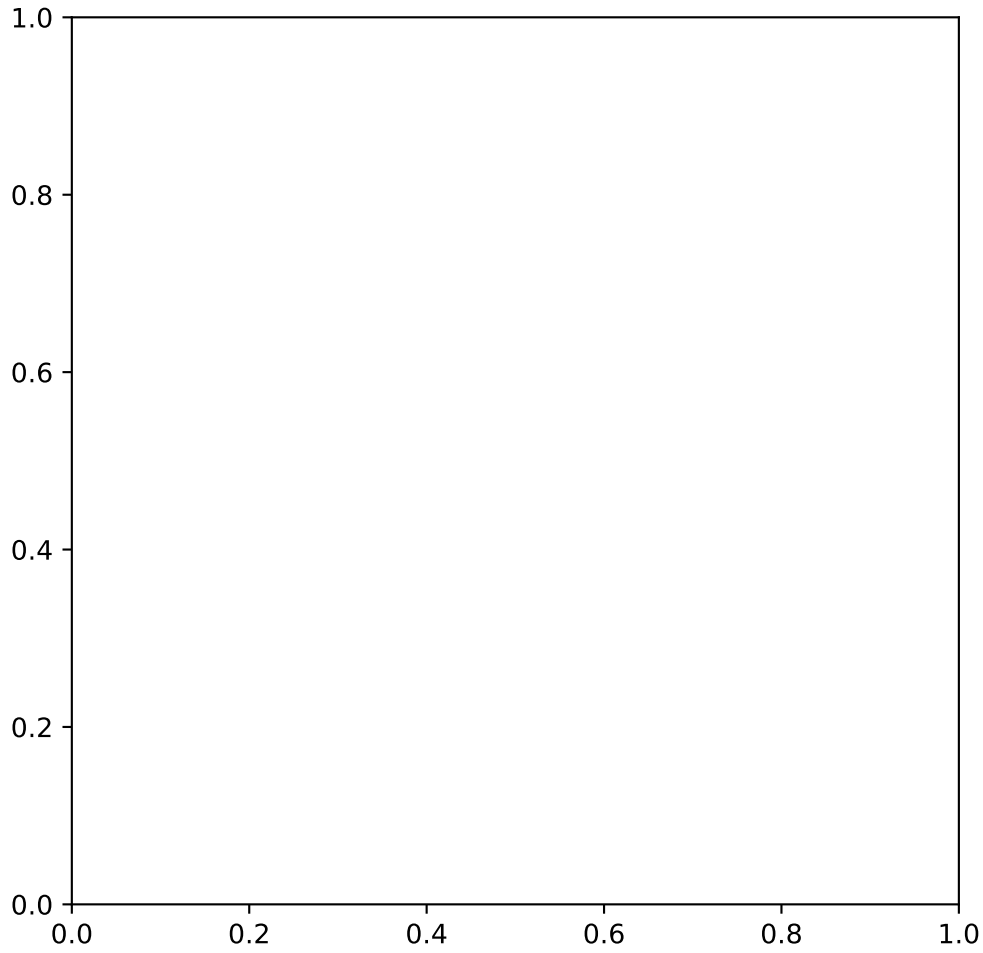
It is time to learn the grammar of matplot. Since it is the most popular library and you are going to use data viz a fair amount

1.1 Initializing a plot

```
import polars as pl
import matplotlib.pyplot as plt
from palmerpenguins import load_penguins

# Load dataset into a Polars DataFrame
penguins = pl.from_pandas(load_penguins()).drop_nulls()
```

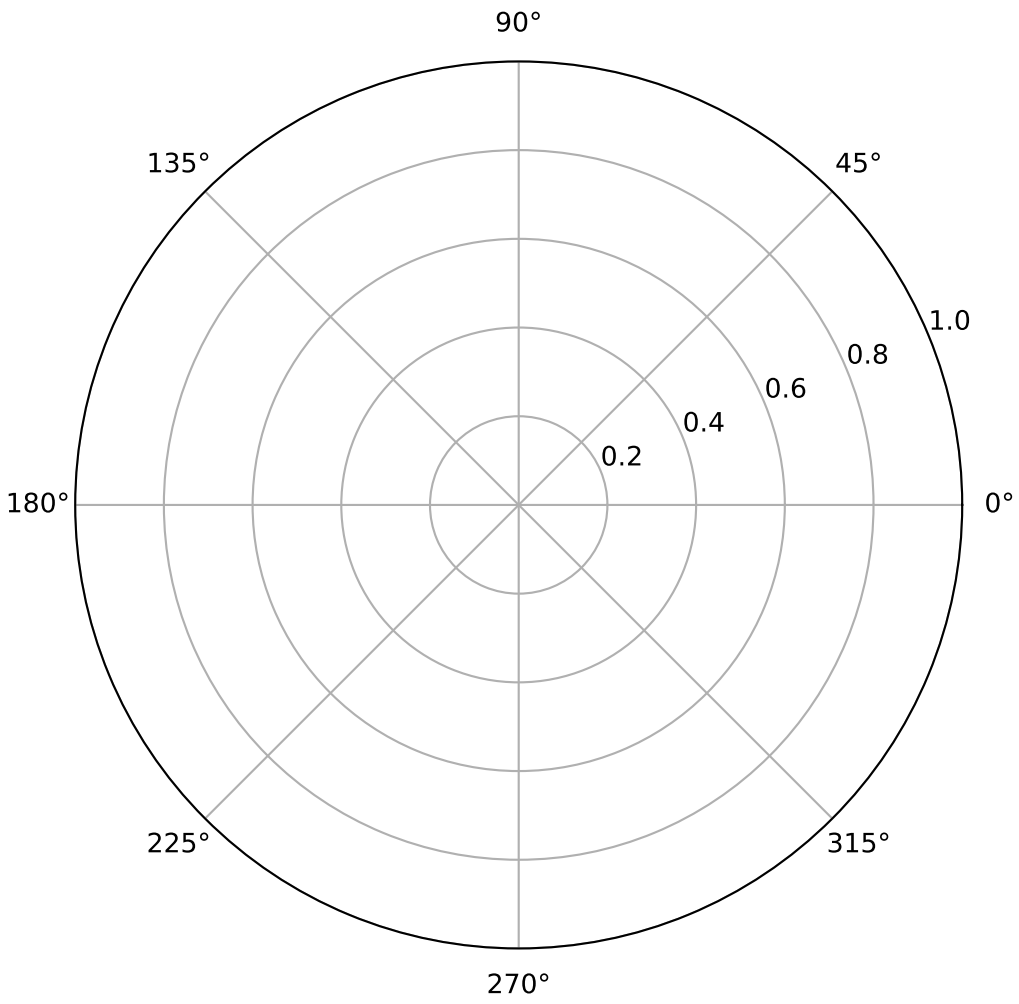
The basics of matplot are kind of like ggplot just reshuffled a little bit. To initialize a plot we first create an object.

```
fig,ax = plt.subplots(figsize = (6,6), subplot_kw = {'aspect':1})
```



So in a matplot fig we have control over the axis and what goes in the figure. The figure parts control the actual plotting of the data similar to geometries in ggplot. We initialize the plot with an aspect ratio of 6 inches wide by 6 inches tall. There are some finer points on the axis but basically we have control over the traditional x and y axis. However, matplot thinks of the the major grid and minor grids, and major labels and minor labels as axis as well. The axis also control the projections of the plot. So if we wanted a polor projection we could do

```
fig,ax = plt.subplots(figsize = (12, 6),subplot_kw = {'projection': 'polar', 'aspect': 1})
```

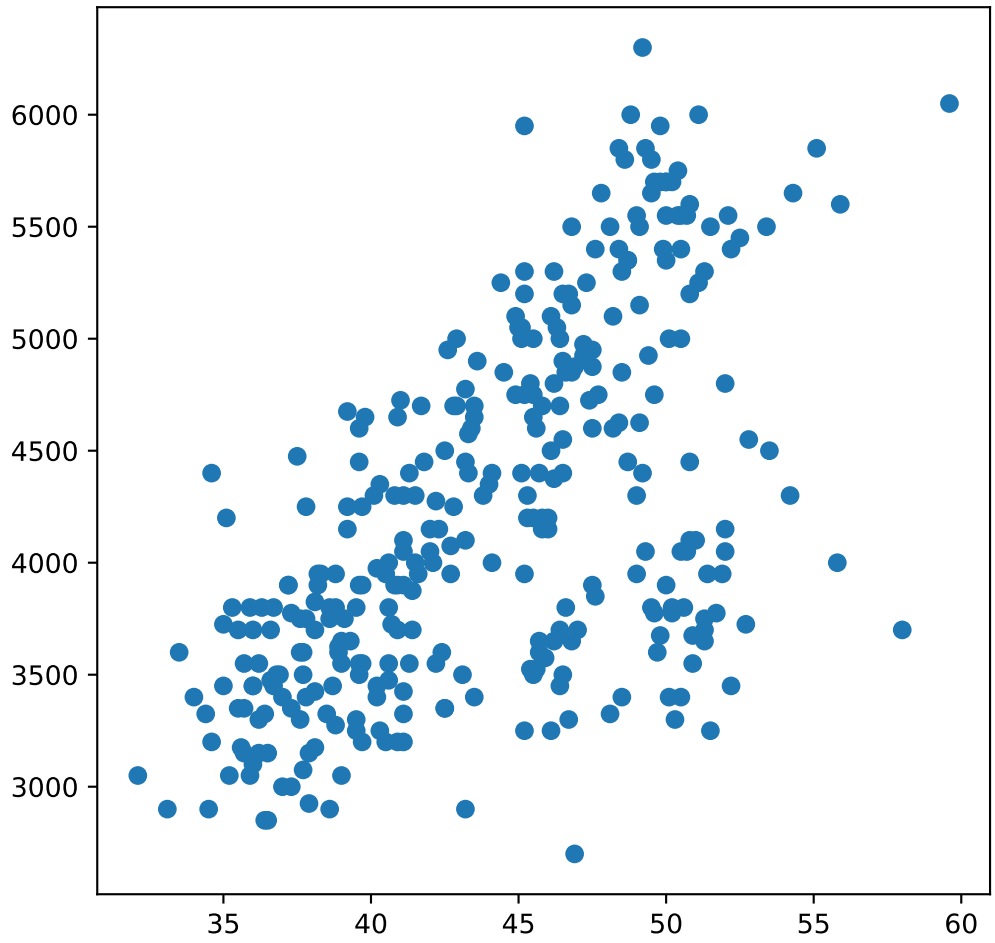


Which is pretty straight forward. There are some nuances within this but we are just going to keep it pushing.

1.2 Adding geometries

The next step is to start adding things so we are just going to initialize a new plot. We kinds of just add things with a dot instead of a + so lets make a simple scatter plot.

```
fig,ax = plt.subplots(figsize = (6,6))

plt.scatter(x = penguins['bill_length_mm'], y = penguins['body_mass_g'])
```



Matplot is a little bit dumber than `ggplot` meaning that it will just do exactly what you tell it. So instead of putting basic labels on the x and y axis like ggplot it will just plot the data. So if we wanted to add labels to the axis than we need to add the labels

```
plt.xlabel('Bill Depth(mm)')

plt.ylabel('Body Mass (g)')

plt.close()
```

We can also add more geometries like this

```
import numpy as np
from statsmodels.nonparametric.smoothers_lowess import lowess
```

```
x = penguins['bill_depth_mm'].to_numpy()

y = penguins['body_mass_g'].to_numpy()
smoothed = lowess(y, x, frac=0.3)  # Adjust `frac` for smoothing degree
x_smooth = smoothed[:, 0]
y_smooth = smoothed[:, 1]

plt.scatter(x, y)

plt.plot(x_smooth, y_smooth)
plt.xlabel('Bill Depth (mm)')
plt.ylabel('Body Mass (g)')

plt.close()
```

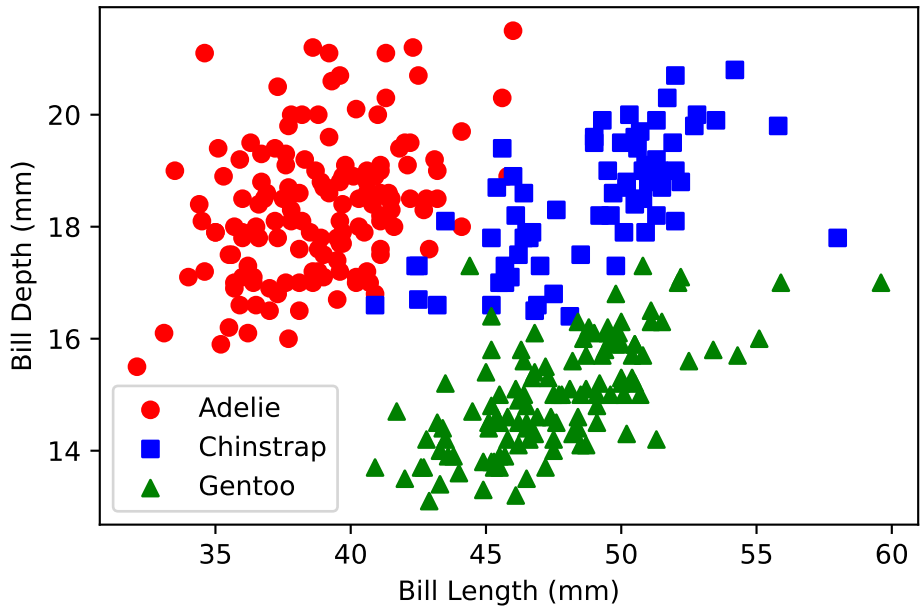
If we wanted to change the method we could do

```
from sklearn.linear_model import LinearRegression
x_reshape = x.reshape(-1,1)

model = LinearRegression()

model.fit(x_reshape, y = y)

y_pred = model.predict(x_reshape)




plt.scatter(x, y, s=10, alpha=0.7, label="Original Data")
plt.plot(x, y_pred, color="red", label="Linear Regression")
plt.xlabel("Bill Length (mm)")
plt.ylabel("Body Mass (g)")
plt.title("Linear Regression: Bill Length vs Body Mass")
plt.close()
```

So the pattern emerges that we have some differences based on subgroups. So we need to plot these by species and then fit a regression line. In ggplot this process is a little bit more compact. But for matplot it is a lot like base R where we loop over and then do this.

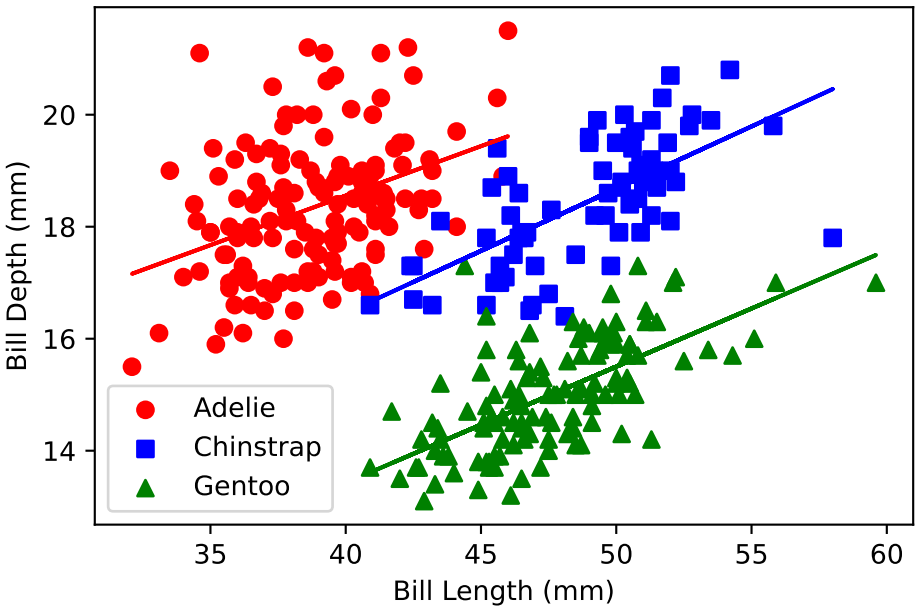
```
species_unique = penguins.unique(subset='species').sort('species')['species']

penguins.glimpse()

markers = ['o', 's', '^']

colors = ['red', 'blue', 'green']

for species, marker, color in zip(species_unique, markers, colors):
    species_data = penguins.filter(pl.col('species') == species)
    plt.scatter(x = species_data['bill_length_mm'], y = species_data['bill_depth_mm'],
    label = species, marker = marker, color = color )
    plt.xlabel('Bill Length (mm)')
    plt.ylabel('Bill Depth (mm)')
    plt.legend()
```

```
Rows: 333
Columns: 8
$ species           <str> 'Adelie', 'Adelie', 'Adelie', 'Adelie', 'Adelie', 'Adelie', 'Adelie
$ island            <str> 'Torgersen', 'Torgersen', 'Torgersen', 'Torgersen', 'Torgersen', '
$ bill_length_mm    <f64> 39.1, 39.5, 40.3, 36.7, 39.3, 38.9, 39.2, 41.1, 38.6, 34.6
$ bill_depth_mm     <f64> 18.7, 17.4, 18.0, 19.3, 20.6, 17.8, 19.6, 17.6, 21.2, 21.1
$ flipper_length_mm <f64> 181.0, 186.0, 195.0, 193.0, 190.0, 181.0, 195.0, 182.0, 191.0, 198
$ body_mass_g       <f64> 3750.0, 3800.0, 3250.0, 3450.0, 3650.0, 3625.0, 4675.0, 3200.0, 380
$ sex               <str> 'male', 'female', 'female', 'female', 'male', 'female', 'male', 'fe
$ year              <i64> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007
```



In a similar way we need to do this for the the linear regression lines

```
for species, marker, color in zip(species_unique, markers, colors):
    # Filter data for the current species
    species_data = penguins.filter(pl.col('species') == species)

    # Extract x and y values
    x = species_data['bill_length_mm'].to_numpy()
    y = species_data['bill_depth_mm'].to_numpy()

    # Scatter plot
    plt.scatter(x, y, label=species, marker=marker, color=color)

    # Fit linear regression (1st-degree polynomial)
    m, b = np.polyfit(x, y, 1)

    # Plot regression line
    plt.plot(x, m * x + b, color=color)

# Add labels and legend
plt.xlabel('Bill Length (mm)')
plt.ylabel('Bill Depth (mm)')
plt.legend()
plt.show()
```


Bill Depth (mm)
20
18
16
14
35
40
45
50
55
60
Bill Length (mm)
Adelie
Chinstrap
Gentoo

2 Probability

We have our general sense of probabilities where we are just the number of times things occur. This is kind of helpful.

```
import numpy as np
from scipy import stats as stats
from matplotlib import pyplot as plt
import polars as pl

numbs = np.array([1,3,4,3])

1/numbs.sum()
```

```
0.09090909090909091
```

However, for the most part in the real world we don't neccessarily care about the probability of a single event happening unconditionally. Conditioanl probability is genereallly a little weird but not totally different than just counting.

2.1 Conditional Probability

There are two ways we generally think of basic condiitonal probability Frequentistly and Bayesianly. For the most part these are fairly similar. The key difference is how we incorporate what we know about the world.

2.1.1 Freqeuentist

In Frequentism people often say that we don't impose any priors on the data. This is not true in any real sense because we impose a flat prior. So the prior is kind of incorporated for us. Typically we see conditional probabilities expressed in two ways

$$\text{P(A|B)} = \frac{\text{Probability of A and B Hapenning}}{\text{Probability of B happening}} \quad \text{P(A|B)} = \frac{\text{P(A)} \cap \text{P(B)}}{\text{P(B)}}$$