Self-Study Guidance

Pritam Dalal

Cheesy Cliches

- A course like 5990 or 5091, no matter how intense it might feel, can only ever really be a light introduction to the task of doing professional work via writing code.
- Improving your coding skills is a lifelong journey.
- Classes are a good way to start, but eventually they will be an inefficient use of time. Although the social element of talks and seminars will always be valuable.
- It is essential to become a competent self-learner: online books, tutorials, MOOCs, Udemy.

Learning Programming is Painful

- Learning your first programming language is the hardest, but the next one is that much easier.
- Whenever I learn a new programming language, it seems like every line of code I write takes an inordinate amount of effort. I felt this way with Python earlier this year.
- After **A LOT** of hard work this dissipates and you feel like you can “express” yourself with code.
    - But as soon as you try to pick up a new language, the feeling comes back. So don’t get too comfortable.

Forget Axioms

- Many folks with math/science backgrounds approach programming with this idea of gaining a deep understanding about the logic of the syntax and computer programming concepts.
- I was definitely in this camp. Borrowing a paradigm from math, I wanted to learn the “axioms” of programming.
- A more useful way of going about programming involves:
  - getting sh@$ done
  - a experimental engineer
  - iterate, iterate iterate

Doing Sh$@ vs Learning

- A typical way to get started with this craft is to have a vague notion of: *I want to learn more about programming*. And then you probably sign up for a class.
- This is OK, and is to be expected when you get started, but try to get away from just *learning* as quickly as you can.
- The best way to improve at programming is by having a project (as small or as big as you want) that you actually care about, and try to do it.
  - Anything you don't know how to do, you learn along the way.

Pritam's Example Project - GARCH(1,1)

- I am backtesting a trading strategy for a client where the trade decisions are based on volatility estimates that they want to calculate using GARCH(1,1)
- Here is I am going to break down this project into units of coding work:
  1. I'll use the `rugarch` package in R.
  2. Follow the tutorial and get some results for SPY as quickly as possible, for one date.
  3. Iterate through a bunch of dates and calculate for SPY. (write a for-loop around the previous step)
  4. Iterate over a bunch of underlyings, for a bunch of dates. (write a for-loop around the previous step)
  5. analyze the results using `dplyr` and `ggplot2`.
  6. communicate results to client.
  7. Read theoretical background material.

Oscillate Between These

1. A project that is meaningful to you (more than 60% of your time).
2. Easy study material, tutorials (e.g. R4DS, Udemy videos).
    - for when your brain hurts but you want to work on code
3. Advanced study material (e.g. Advanced R, creating packages).
    - for when you're bored with your project and you want some intellectual stimulation
4. Reading theory (this should be less that 10% of your time)
    - when you want to waste some time and you don't have any Netflix shows to binge

Preview for Spring 2019

1. Learn the basics of data analysis in Python in the context of the Delta-Neutral options data.
2. Do a very similar backtest analysis project within about 4-6 weeks of the start of the class.
3. Classical Machine Learning Topics (both R and Python)
   - supervised: regression and variants, k-nearest neighbors
   - unsupervised: principal components analysis, k-means clustering
4. Deep Learning (if time permits)