Stat 650/750 Course Description and Syllabus Fall 2016

Vital Info

Class Schedule	TR 12-1:30PM, BH 332P
Instructors	Christopher Genovese 232E Baker Hall (x8-7836) genovese@cmu.edu Alex Reinhart areinhar@stat.cmu.edu
Office Hours	To be determined empirically and by appointment
TAs	Philipp Burckhardt and Taylor Pospisil
Home Page	http://www.cmu.edu/blackboard
Text	Class notes and handouts

Course Objectives and Scope

Computing is an essential—and increasingly important—part of statistical practice and methodology, but computing's role in the typical graduate Statistics curriculum (including ours) has not been commensurate with its importance. For most students, the primary opportunity to develop computing skills comes during research projects, but this tends to put completion above learning and tends to cover ideas that are focused on the particular needs of the project. While students may be able to take a computing-related course here or there, it is rare for such a course to cover the fundamental concepts in computing and develop critical skills in a way that will pay off during later research, work experience, and beyond. This course, 36-650/750, aims to fill that gap.

The premise of this course is that building a broad and solid foundation in computing will pay significant dividends throughout a student's research career. We will focus on four main themes: (a) effective programming practices; (b) fundamental principles of software design; (c) important algorithms, data structures, and representations; and (d) essential tools and methods. Along the way, we will also consider a variety of applications and techniques that are important for statistical practice. However, the focus of the course is not on specific statistical techniques per se. Similarly, although you will practice new tools and approaches to programming, the purpose of this course is *not* teaching you to program.

A second premise of this course is that practice is the key to developing strong computing skills. To this end, the work will consist of many programming tasks of various sizes, both between and during class. You will have access to a repository of exercises varying in complexity so that you can target your work to your experience level, and you will have an opportunity to work interactively with your peers. Prior programming experience is not a requirement, but if you have not programmed before (or much), you will be expected to work toward learning a chosen language. This course is, for the most part, language agnostic, and indeed, you will be asked to do several (simpler) tasks in a language outside your comfort zone to gain the valuable perspective that this offers.

By the end of this course, you should be better able to:

– develop correct, well-structured, and readable code;

– design useful tests at all stages of development;

– effectively use development tools such as editors/IDEs, debuggers, profilers, testing frameworks, and a version control system;

– build small-to-medium scale software system that is well-designed and that facilitates code reuse and generalization;

– select algorithms and data structures for several common families of statistical and other problems;

– write small programs in a new language.

Class Mechanics

Classes will feature a combination of lectures, interactive discussions, and (single and group) programming activities. We will often discuss, edit, and run code from a variety of sources, and we will do a fair amount of real-time programming. Hence, ***you should bring your laptop to every class***.

We will have mechanisms for sharing code snippets interactively and turning in homeworks interactively. For this purpose, you should sign up for a (free) account at `github.com`, and then sign up for the free educational account at `education.github.com`.

Participation and attendance are important parts of the class and as such will constitute a nontrivial portion (15%) of your final grade.

Resources

Class notes will be made available on Blackboard shortly after class. In addition, there will be occasional handouts and readings, all of which will be posted to Blackboard.

There is no primary text for the class, but several useful (though not required) books have been put on reserve in the Engineering and Science Library (in Wean Hall). These are:

- *Code Complete (2nd Edition)* by Steve McConnell. Provides (copious) detail on software construction.

- *The Algorithm Design Manual (2nd Edition)* by Steven Skiena. Provides details on a wide range of algorithms, from sorting to graph traversal, from computing convex hulls to solving Sudoku exercises.
- *Seven Languages in Seven Weeks* by Bruce Tate. Provides primers on the programming languages `Ruby`, `Io`, `Prolog`, `Scala`, `Erlang`, `Clojure`, and `Haskell`, thereby expanding one's horizons beyond oft-used languages such as `R`, `C`, and `Python`.

In addition, various electronic documents and links to other online resources will be available through Blackboard.

The instructors and TAs will hold regular office hours. The times of those sessions will be announced on Blackboard.

Assignments

The main work in this course consists of programming and related exercises. There will be three main types of exercises:

- Stand-alone. These are short, self-contained tasks that illustrate or build upon an idea we have discussed in class. Many of these involve writing a short program, but others will involve some related task (e.g., debugging, reasoning).
- Vignettes. These are groups of exercises around a central theme, data set, or idea. Within a vignette, all exercises that are not marked optional should be completed eventually (although this will usually take place over several assignments).
- Challenges. These are larger programming tasks that integrate several different ideas and skills. There will be two challenges assigned over the semester.

All of these exercises will be available in the course repository on GitHub.

Short assignments will be graded on a simple rubric as either "Mastered" or "Not yet mastered". Challenges can also be graded as "Sophisticated", the criteria for which will be specified in their rubrics. You will have the opportunity to revise your assignments to address issues found during code review; each assignment and project may be revised at most twice. Rubrics will be posted on Blackboard.

You will usually have free choice of which exercise to do, within some relevant subset of the repository. Occasionally, though, an exercise will be specified that everyone should do. There will be a required (but ungraded) assignment and a diagnostic assessment due the first day of class, posted on Blackboard.

Grading

Course grading is based on the rubric mentioned above. Rather than a points-based scheme, we will use a simple set of requirements to set your grade. The semester will have two scheduled Challenge projects; assignments can be submitted ***at most twice per week***, including revised assignments, so do not procrastinate. (There are fifteen weeks, so you have 28 opportunities to submit, not including the first week. You will be working on the Challenges for several of these weeks.) The TAs will grade the first two assignments or revisions submitted each week, leaving any excess submissions to be graded in subsequent weeks.

Grade	Requirements
R	Fail to meet requirements for D
D	Earn Mastered on at least 8 assignments
C	Earn Mastered on at least 10 assignments
B	Earn C **and** Mastered on one Challenge
B+	Earn B **and** Mastered on at least 12 assignments, **or** earn B **and** Sophisticated on one Challenge
A-	Earn Mastered on at least 12 assignments **and** Mastered on both Challenges
A	Earn A- **and** Sophisticated on one Challenge
A+	Earn A- **and** Sophisticated on both Challenges

This grade may be adjusted upwards for students who demonstrate outstanding participation in the class.

Policies

Email. When sending an email, please put either "[650]" or "[750]" at the beginning of the subject line, so that we can easily identify the message. Also, please be advised that merely sending email ***does not eliminate your responsibility for completing assignments on time***.

Disability Resources. If you require a special accommodation, see the online guidelines from the Office of Disability Resources (link). Please give the instructor documentation with reasonable notice before accommodation is needed.

Collaboration, Cheating, and Plagiarism. Discussing assignments with your fellow students is allowed and encouraged, but it is important that every student get practice working on these problems. This means that ***all the work you turn in must be your own***. You must devise and write your own solutions and carry out your own tests. The general policy on homework collaboration is:

> No student should ask for assistance from any other student or offer assistance to any other student until that student has made a serious effort to solve the problem. All work must be written up individually. Direct copying or dictation of another student's solution to a problem or obtaining a solution from an outside source will be considered cheating. This applies also to computer source, output, and tests.

In addition, you ***should not consult on-line or other sources*** that discuss solutions to problems related to those in the homework. You are of course free to refer to programming language documentation and discussion of general concepts and approaches. Feel free to come talk to us if you have any questions about this.

Any form of cheating is typically grounds for course failure. We are obliged in these situations to report the incident to the appropriate University authorities. Please refer to university policies on academic integrity (link).

POLICY UPDATES. Updates to policies and course information will be posted in updated versions of this syllabus and announced on the course homepage.