Lesson 25 – Software Engineering for Data Scientists

**Questions for Mentor:**

**How to be a better data scientist:**

* Project design
  + Know business domain
  + Understand the ask
    - How data was extracted, by whom, are there biases?
  + Do the work that needs to be done
* Code design
  + Clearness beats cleverness
  + Names matter
    - Lists, variables, etc.
  + Functions/classes
  + Code stubs
  + Docstrings/comments
  + Exception handling
    - Have it return a helpful error message
  + Unit tests
* Tools for the job
  + IDEs/Environment
* 10x Extras
  + Pattern matching
    - Noticing similar problems that you’ve seen before
  + Learn how to explain your code to yourself and others
    - More than just “that’s what I got from stack overflow”
  + Learn when to quit/start over
  + Create own stock of gists
    - Commonly used code to reuse over time
* Productionizing model
  + Different options
    - AWS ML
    - Azure ML
    - Google Cloud ML
    - Rackspace
    - Algorithmia
    - Domino lab
    - Continuum
  + Consider pricing – some charge for hosting
  + Where is code going after you create your model?

**Data Science is Software:**

* Project structure
  + Clear structure in directories
* Environments and reproducibility
  + Watermark extension
    - !pip install watermark
    - %load\_ext watermark
  + Set up new environment for project
  + Include requirements for the environment in requirements.txt
* Coding for reusability
  + Jupyter Notebook
    - Code I expect to be executed or just read by a human
  + Python file (.py)
    - Code I expect to be executed by multiple notebooks or a machine
  + Python package index
    - Code I expect to be executed by many machines (or humans)
  + % debug allows you to navigate to your code that’s driving the error
    - Make sure you type ‘q’ when you’re done the debugger to quit
    - %pdb 1 turns on interactive debugger every time
* Testing
  + Tools to easily debug
  + Rubber duck – talk through your code
    - Like writing email to mentor giving ideas for project
  + Want to test any parts of code that can be isolated
  + Fixtures are static data that can be used for testing (df)
    - @pytest.fixture
* Collaboration
  + Important things companies look for
    - Version control
      * Github or other – just need to use it
    - Issue tracking
      * Formally keep track of bugs
      * Github issue is a good issue tracker
      * Establish workflow around issues and version control
        + Use master branch if its just you or you and a couple other people
        + Feature branches with slightly larger teams
        + Git flow

Another level up on complexity from feature branches

* + - Continuous integration
      * Establish norms
      * Code reviews
        + Ask someone else to take outside look as sensibility check

**Code Reviews:**

* Build an excellent product
* Build people
* Build yourself
* Creative work is powered by enthusiasm
* Two important factors
  + Truth
    - Clarity of explanation
      * Code
      * Links (specific commits)
      * Higher-bandwidth communications
      * Write down result!
  + Kindness
    - Tact hacks
      * Question mark – instead of stern statement, phrase as question
      * You, we, & this – make it ‘we’, not ‘you’
      * Compliments
      * Humor
* TL; DR; LGTM
  + Solutions
    - Prose overview of patch
    - Long commit messages
    - Small commits
    - Comments, docstrings, naming
* Nitpicks
  + Be generous
  + Give credit where due
  + Compliments in public, critiques in private
  + Spend reviews on things that matter – not Nitpicks
* While you’re at it
  + HaHaOnlySerious
    - Touching ugly code – it can become yours – avoid that!
  + Getting better
    - Not being perfect
* Slow turnarounds - avoid
  + Energizing to have fast turnarounds
  + Comprehensiveness not required
  + Respect working memory
  + Quick “No”s
* Insecurity
  + Imposter syndrome
  + Everybody wrapped up in themselves
  + Nobody should lose in code review
  + Why are you feeling what you’re feeling? What’s the worst that can happen?
* Feeling short on time
  + Always more to do than can be done
  + 2 minute rule
    - If it takes less than 2 minutes, just do it
  + Leveling up newcomers
* The trust bank
  + Establish good relationships with team when in person

**Maintainable Code in Data Science:**

* What is maintainable code?
  + Easy to modify
* Each class/function should do one thing for our model
* In Data Science we have extra constraints
* Pieces of DS project
  + Process data/engineer new features
    - How easy is it to modify / remove / add steps?
  + Tune hyperparameters
    - How easy to re-tune model after data has changed?
  + Train model
    - How easy to retrain model after data has changed?
  + Generate predictions on new data
    - Can you run all same steps easily on any new data w/o repeating code?
* Feature engineering/Data processing
  + One hot encoding
    - Encode same way every time it runs (France, UK, China, not just France and UK in video example)
    - Using .transform() implements change that was created with .fit() method
  + Column Transformer
    - Applies different methods to different columns
      * PCA for numerical columns, onehot for categorical columns
  + Pipelines
    - Multiple operations that can be mapped sequentially
    - Can be used with columntransformer
      * Apply pipeline on numerical and onehot on the categorical
    - Can be used directly with alrogithms as well as gridsearch, cross validation etc.
      * Avoids data leakage
    - Can apply same pipeline to new data
* Model Persistence
  + Pickle and joblib
    - Serializing models – saving on disk
    - Limitations
      * Dependencies are not saved
        + Must import libraries used and external files with classes created

**Getting Started Testing:**

* Goals
  + Show a way to test
  + Remove mystery
* Why test?
  + Know if code works
  + Save time
  + Better code
    - More modular
  + Remove fear
  + “Debugging is hard, testing is easy”
* Good tests
  + Automated
  + Fast
  + Reliable
  + Informative
  + Focused
* Unittest
  + Python standard library
  + Infrastructure for well-structured tests
  + Patterned on xUnit
* Test isolation
  + Every test gets a new test object
  + Tests can’t affect each other
  + Failure doesn’t stop next tests
  + SetUp and teardown
* Tests are real code
  + Helper functions, classes etc
  + Can become significant
  + Might need tests
* How to test live data?
  + Unpredictable
  + Slow?
  + Unavailability?
  + Question should be:
    - Assuming yahoo.com is working,
    - Is my code working?
* Test doubles
  + Powerful: isolates code
  + Focuses tests
  + Removes speed bumps and randomness
  + BUT: fragile tests!
* Tools
  + addCleanup – nicer than tearDown
  + doctest – only for testing docs!!!
  + Nose, py.test – better test runners
  + Ddt: data-driven tests
  + Coverage – you’ll wonder how you lived without it
  + Selenium – in-browser test
* Topics
  + TDD
  + BDD
  + Integration tests
* Testing is:
  + Complicated
  + Important
  + Worthy
  + Rewarding

**Python Debugging:**

* Syntax error
  + Error found at location of the error message
* Runtime error
  + Error not always found at location of error message
* Debugging techniques
  1. Read error message
  2. Print statements
  3. Make a hypothesis (prediction) why we’re seeing what we’re seeing
  4. Explain the problem to someone – verbalize issue
  5. Clean up the code
     + Get rid of unused code
  6. Use assert statements
  7. Pdb/ipdb
     + Interactive debugger
     + Locals() shows all variables and contents
     + Conditional break points (b <line>, cond)
  8. Minimizing input
     + Don’t define variables twice etc.
  9. Logging
  10. Debugging is hard! Sometimes you need to take a break
* Debugging summary: must know
  + Error messages in python not always helpful
  + Syntax errors are when python does not do anything
  + Some errors cause a program to stop with an exception
  + Read error messages from bottom to top
  + Semantic errors: the program does not do the right thing
  + Errors are distinct from the underlying defects
  + Defects propagate through the program