Course Wrap-Up

Pritam Dalal

April 30, 2019

History & Background

- Python is a general purpose programming language that was created in 1990 by Guido van Rossum.
- It was **not** initially created as a scientific computing or data analysis programming environment.
  - R and Matlab were created specifically for this purpose
- Python is an interpreted language (no compilation step) which makes it well suited for interactive computing.
- Development of Python as scientific computing tool began in 2001 when Fernando Perez launched of the IPython project.

Python, IPython, Qt Console

- The simplest way of running python code is from the python console application:
  - >>>
- IPython is an enhanced (but still primitive) interactive console application that is more suited for scientific computing:
  - `In [1]:`
  - still didn't include in-line plotting
- The Qt Console is a more modern command-line interface that allows for inline plotting.

The Juypter Project

- Two notebook formats that grew in popularity in the 1990s and early 2000s were Maple and Mathematica.
    - both were very expensive
- The first IPython notebook was created in 2011.
- From 2011-2014, IPython notebooks started supporting other languages, including R and Julia.
- In 2014, all the language agnostic parts of IPython were rebranded as Project Jupyter.
    - Jupyter Notebook
    - Jupter Lab

Various Ways We Ran Python

- Python command line tool: `>>>`
  - demo only
- IPyton command line tool: `In [1]:`
  - demo only
- Jupyter Notebook: `$ jupyter notebook`
  - first half of class
- Jupyter Lab: `$ jupyter lab`
  - second half of class
- VS Code: `$ code`

Built-In Data Structures

- List - `[1, 2, 'a']` - a simple container of multiple values.
  - the values can be of multiple types
  - we mostly used this for indexing `DataFrame` columns or supplying arguments to functions
- Tuple - `(2022, 7)` - an immutable sequence of values
  - e.g. the output format of `DataFrame.shape`
- Dict - `{'SPY':290.15, 'IWM':160.25}` - Python's implementation of a hash table.
  - we mostly used this for function arguments

Control-Flow and Functions

- `if-elif-else:`
  - conditional execution
- `for :`
  - repeating a block of code a fixed number of times
- `def func_name(<inputs>): <code> return <output>`

Packages

Python is an open source language and it has been greatly extended by thousands of paackages. Here are the ones we used in this course:

- `numpy` - performant linear alegbra
- `pandas` - data analysis, basic visualization
- `matplotlib` - low-level visualization library
- `seaborn` - high level statistical visualization library
- `sklearn` - machine learning

Numpy

- This is the foundational package that most scientific Python packages (e.g. `pandas`) are built on top of.
- `ndarray` data structure that can be used to represent vectors and matrices
- Arrays can be created manually with the `np.array()` function.
  - we didn't do this very much because our data comes from external sources
- Element-wise function broadcasting allows for vectorized code.
- Implementation of standard linear algebra libraries like BLAS and LA-PACK.

Pandas

- Create by Wes McKinney in 2008, when he was working as quant at AQR, a quantitative asset management firm.
- Built on top of `numpy`.
- Introduces two primary data structures.
  - `Series` - an indexed one-dimensional array
  - `DataFrame` - a collection of names `Series`
- The `DataFrame` structure was borrowed from R (which in turn was inspired by SQL).
  - one major difference is that R data frames do not have an index.

DataFrame Indexing

- Indexing generally refers to selecting subsets of a `DataFrame` by positional arguments.
- To select columns: `df['col1']` or `df[['col1', 'col2']]`.
- To select rows by index: `df.loc[]`.
- To select rows by row number: `df.iloc[]`.
- `pandas` has a variety of row indexers, which can be a little confusing.

DataFrame Masking

- Masking allows you to grab all rows of a `DataFrame` that meet certain conditions.
- The basic idea is to feed an array of booleans into a `DataFrame`.
- The booleans are generated by some kind of comparison operation, usually involving the columns of the `DataFrame`
- Eg. `df_spy[df_spy['trade_date'] > '2019-01-01']`
- This is analagous to querying a database table in SQL with a `WHERE` clause.

Vectorized Code

- Vectorized code is an important concept in data analysis.
- In essence, code vectorization means writing code that is absent of a lot of `for` loops.
  - `numpy` broadcasting
  - `numpy` universal functions
  - `DataFrame.apply()`
- The `for` loops exist, they are just written by the package developers, not you.
- Code vectorization has several benefits:
  - faster development times
  - improves code readability
  - improves performance by pushing calculation to the C layer.

Merging and Joining

- Data is often spread across multiple data frame and needs to be combined before further analysis.
- This is done the with `.merge()` function in `pandas`.
- Three variants:
  - `inner` - keeps matches in both tables
  - `left` - keeps all rows in left table, returns matches in right
  - `right` - keeps all rows in right table, returns matches in left
- I found joins to be mentally taxing when I was learning them.

Grouping and Aggregation

- The basic data wrangling toolbox is rounded out with performing aggregation calculations (sum, average, stdev) on subgroups of a data frame.
- This is accomplished by a combination of `.group_by()` and `.agg()`.
- Creating custom aggregation functions is great way of adding flexibility of this technique.
- A huge amount of basic data analysis/wrangling is repeated combination of indexing, masking, joining, groupby, aggregation, and a bit of visualization.

Visualization

- No treatment of data analysis is complete without a discussion of data visualization.
- My personal opinion is that at an introductory level, the basic wrangling techniques discussed above have a higher return on time investment.
- Finished graphs or complex visualization can take a long time to develop.
- Simple graphs that are quick and dirty can be generated quickly in python, and are a good return on time.
- In this class we focused on the graphing capabilities in `pandas` and `seaborn`.

Matplotlib

- There are many visualization packages in the python ecosystem, most of them are built on top of `matplotlib`.
- `matplotlib` is a low-level package.
- `matplotlib.pyplot` is a module that contains functions that collectively create a MATLAB-like plotting interface.
- `pyplot` is a legacy feature which was implemented to compete with MATLAB.
- Use the `%matplotlib inline` magic command in Jupyter to see plots inline.

Pandas Plotting

- `pandas` has simple built-in plotting capabilities, which allow you to easily produce graphs while working with `DataFrames`.
- We created both line graphs and bar charts with the `DataFrame.plot()` method.
- To create scatter plots we used the `DataFrame.plot.scatter()` method.
- You can apply customizations to the graphs via arguments to these functions.

Seaborn

- `seaborn` is a visualization package that is written on top of `matplotlib`.
- The idea behind `seaborn` is to be higher-level than `matplotlib` and to also focus on statistical applications.
- `seaborn` plots tend to be a bit more stylish with default settings.
- In tutorial 16, we recreated all of our `pandas` plots with `seaborn`.
- Later in the course we used `sns.pairplot()` to create pair-wise scatter plots for a collection of features.

Monte-Carlo Simulation

- Monte-Carlo simulation is a general technique used in finance for a variety of tasks including option pricing.
- The basic idea is to generate a variety of simulated market price paths that you believe reflect the future.
- Aside from general finance interest, we used Monte-Carlo as a way to explore a different style of programming in Python.
- `numpy.random` was used to generate psuedo-random numbers.

Beyond Notebooks

- In this course, almost all of our work was done in Jupyter Notebook.
- This is typical of initial exploratory data analysis (which is highly interactive) and *final-stage* data analysis (which involves rich-text, code, and visualization).
- In real-world settings, a lot of time is spent in between these two stages, an consists of a large amount of invovled data wrangling.
- This type of coding is less interactive, and feels more like programming than data analysis.

Modular Programming

- Larger data analysis efforts require spreading code over multiple text files, which is known as *modular programming*.
- Python has three constructs to facilitate modular programming:
  - function
  - module - a `.py` file that contains Python code
  - package - a folder that contains modules
- `from <package>.<module> import <function>`
  - e.g. `from sklearn.decomposition import PCA`
- Check out the `package` folder to see some examples.

Machine Learning

- Machine Learning is fundamentally about prediction and pattern recognition using data.
- In a *supervised learning* task we seek to predict a *label* from a given set of *features*.
  - classification: discrete labels
  - regression: continuous labels
- In an *unsupervised learning* task we seek to find interesting structure with-in the features themselves.
  - there are no labels
  - this is harder and more subjective

sklearn

- `sklearn` is a popular library that encompasses many machine learning techniques.
- One very nice feature of `sklearn` is that all learning models share the same API and workflow.
  - feature selection: organized in a `DataFrame` or `ndarray`
  - scaling: normalize features
  - model selection: instantiate model with constructor
  - fit the model: using the `.fit()` method of model object
  - analyze accuracy: using the `.score()` method
- The simple API of `sklearn` has made python the de facto standard language in machine learning.

Finance Examples

- Simple Linear Regression
  - implied leverage effect (implied vs returns)
  - volatility clustering (realized-vol vs realize-vol)
- K Nearest Neighbors
  - predicting gain/loss by looking at VIX term structure
  - good same-day predictor; additional points adds little
  - had little predictive power for next day (exercise 9)
- Principal Components Analysis
  - recovered level, slope, and curvature from VIX term-structure
  - 95% of variance is from level, the first principle component
  - you will see similar results in stock returns (exercise 10)

Key Concepts

- Variance-Bias Trade-Off
  - a model that is too simple, will not capture structure in the data
  - a model that is too complex, will overfit the data
  - in KNN, think `n_components = 1` vs `n_components = 500`
- Hold-Out Sets
  - training set score will inflate accuracy
  - solution - use 80% of the data to fit the model, and then test the accuracy of the model with the remaining 20%.
  - `train_test_split()`
- Cross Validation
  - hold-out sets reduce the amount of data that the model can be trained with
  - solution - break the data set into 5 parts and do and 80/20 train-test-split 5 times; use some aggregate of the 5 accuracies.
  - `cross_validation()`

Delta-Neutral Data

www.historicaloptiondata.com

- All the option data from this course was donated by Delta-Neutral.
- This is high quality EOD equity option data.
- Over 4000 underlyings covered.
- http://www.deltaneutral.com/