# An (extremely) Abbreviated Introduction to Data Science with sklearn

Doug Lloyd
CS50 Hackathon Amsterdam
14 June 2018

# Slides

- Slides and code are available at cs50.ly/amsterdam-ds. No need to write everything down now!

# Data Science

- According to Wikipedia, it is “an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from data in various forms, both structured and unstructured.”

# Data Science

- According to Wikipedia, it is “an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from data in various forms, both structured and unstructured.”

- You probably know this better as ***statistics***. And data science typically adds some machine learning, too.

# Data Science

- According to Wikipedia, it is “an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from data in various forms, both structured and unstructured.”

- You probably know this better as ***statistics***. And data science typically adds some machine learning, too.

- For us, talking about applying programming techniques to process data for various purposes.

# Data

- Select a set of data of interest to you (or your team, or your employer…)

# Data

- Select a set of data of interest to you (or your team, or your employer…)

- Import that data
  - For small data sets, likely can do it at runtime and not consume enormous amounts of memory.
  - For large data sets, might need to create a database file.

# Data

- Select a set of data of interest to you (or your team, or your employer…)
- Import that data
  - For small data sets, likely can do it at runtime and not consume enormous amounts of memory.
  - For large data sets, might need to create a database file.
- Come up with questions about that data you want to answer, then write code to answer those questions.

# Sample Data

- For today's examples, we're going to use the so-called 1978 Boston Housing data set.

# Sample Data

- For today's examples, we're going to use the so-called 1978 Boston Housing data set.

- 100% complete (no "empty data")

# Sample Data

- For today's examples, we're going to use the so-called 1978 Boston Housing data set.

- 100% complete (no "empty data")
- Fairly small, only ~500 rows and 14 columns worth of data.

# Sample Data

- For today's examples, we're going to use the so-called 1978 Boston Housing data set.

- 100% complete (no "empty data")
- Fairly small, only ~500 rows and 14 columns worth of data.
- Downloadable in CSV form at cs50.ly/amsterdam-ds. (You'll see why this is available separately in a moment.)

# Sample Data

- Collected by the U.S. federal government as ancillary census data.

# Sample Data

- Collected by the U.S. federal government as ancillary census data.

- Useful because it has a number of variables that can be used as part of different regressions, almost always used as a test case for predicting *median home value* in general.

# Sample Data

- Collected by the U.S. federal government as ancillary census data.

- Useful because it has a number of variables that can be used as part of different regressions, almost always used as a test case for predicting *median home value* in general.

- Extremely popular data set to introduce data science (see also, *Titanic* data).

# 1978 Boston Housing Data

- `crim` – per capita crime rate
- `zn` - % of land zoned for lots over 25k sq. ft.
- `indus` - % of land in town zoned for non-retail business
- `chas` – boolean for if property borders Charles River
- `nox` – concentration of nitric oxide
- `rm` – avg rooms per dwelling
- `age` - % of units built prior to 1940

# 1978 Boston Housing Data

- `dis` – weighted distance to major employment centers
- `rad` – accessibility to radial highways
- `tax` – tax rate per $10,000
- `ptratio` – student-teacher ratio
- `b` – a formula determining minority population
- `lstat` - % of the town population that is “lower class”
- `medv` – median value of owner occupied homes x1000

# 1978 Boston Housing Data

- `dis` – weighted distance to major employment centers
- `rad` – accessibility to radial highways
- `tax` – tax rate per $10,000
- `ptratio` – student-teacher ratio
- `b` – a formula determining minority population
- `lstat` - % of the town population that is “lower class”
- `medv` – median value of owner occupied homes x1000

# `sklearn`

- The other reason to choose the Boston data set is that it is natively available as part of **`sklearn`**. This means we can dive right in to seeing what **`sklearn`** can do without having to import data!

# `sklearn`

- The other reason to choose the Boston data set is that it is natively available as part of **`sklearn`**. This means we can dive right in to seeing what **`sklearn`** can do without having to import data!

- **`sklearn`** is a very popular module that can be used for data regression and analysis, simplifying many of the calculations that you might otherwise do on your own!

# Step by Step

- Now, let’s walk through some basic introductory steps one might take for tinkering with this data.

# boston-0.py

- Here, let's see if we can get the data properly imported so we can test some other stuff later.

- Looking at the output of this program, generated by line 13's `print(boston)`, what is the data type of the Boston data?

# boston-1.py

- Let’s dig in a little bit deeper, and check out just the size of our data set, to confirm that it matches what we’re expecting.

# boston-1.py

- Let's dig in a little bit deeper, and check out just the size of our data set, to confirm that it matches what we're expecting.

- Hmm…. That doesn't seem quite right. Our dataset is supposed to have 14 columns.

- As it turns out, when used for testing, regression usually happens on the median value, the missing 14$^{th}$ column, which is actually in the "target" key-value pair. We could splice it in, but let's leave it out for now!

# boston-2.py

- Using a popular Python data analysis module, **pandas**, we can determine some relatively basic calculations for each of the columns, such as mean, median, mode, and different quartile values.

- We can examine lots of different things about the data this way.

# boston-3.py

- But what we really want to do is ***analyze***! And we also probably want to actually see the data looking nice in a graph.

- This is where the machine learning aspect of things comes into play, and before we look at the code, we need to talk about *training sets* and *test sets*.

# Training Set

- In a machine learning context, our aim is to write a model (with data that perfect conforms to that model, and typically that data is made-up), and we “feed” that data to the computer.

# Training Set

- In a machine learning context, our aim is to write a model (with data that perfect conforms to that model, and typically that data is made-up), and we “feed” that data to the computer.

- By doing a pairing of expected inputs to expected outputs on the machine’s behalf (and having it perform analysis on the “perfect” data), the machine learns your model and readies itself to apply the model to more interesting (real) data.

# Test Set

- The so-called *test set*, as opposed to the training set, is the actual data that you want to have analyzed. It is not “perfect” as the training set might be, but it’s what you actually care about.

# Test Set

- The so-called *test set*, as opposed to the training set, is the actual data that you want to have analyzed. It is not "perfect" as the training set might be, but it's what you actually care about.

- Ordinarily, your test set and your training set aren't the same data. Here we only have the one set, so we'll randomly partition the data and call some of it our training set and some of it our test set.

# Test Set

- The so-called *test set*, as opposed to the training set, is the actual data that you want to have analyzed. It is not "perfect" as the training set might be, but it's what you actually care about.

- Ordinarily, your test set and your training set aren't the same data. Here we only have the one set, so we'll randomly partition the data and call some of it our training set and some of it our test set.

- Don't make the training set too large!

# Target

- As with many examples you may find with respect to the Boston housing data, we are going to use the first thirteen columns worth of data to see if there is some pattern that can be used to predict the fourteenth column, median home value.

# boston-3.py

- Let’s create our datasets!

# boston-4.py

- Now let’s plot it all out and see how our predictions compare to the actual data.

# boston-4.py

- Now let's plot it all out and see how our predictions compare to the actual data.

- What we are visualizing here is the difference between what our training data taught us about what the median value *should be* versus what the actual data said the value *actually was*.

# boston-4.py

- Now let's plot it all out and see how our predictions compare to the actual data.

- What we are visualizing here is the difference between what our training data taught us about what the median value *should be* versus what the actual data said the value *actually was*.

- In a perfect world, the data would follow a single straight line or at least some polynomial.

# boston-4.py

- When we `fit()` the data on line 29, what's effectively happening is we are coming up with an equation that satisfies $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n = t$, where t is the target value, among the *training* set.

# boston-4.py

- When we `fit()` the data on line 29, what's effectively happening is we are coming up with an equation that satisfies $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n = t$, where t is the target value, among the *training* set.
  - **`boston-4a.py`** adds a few lines of code that actually print out these coefficients, if you're interested in seeing the actual numbers!

# boston-4.py

- When we `fit()` the data on line 29, what's effectively happening is we are coming up with an equation that satisfies $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n = t$, where t is the target value, among the *training* set.

- When we `predict()` the data on line 30, we are basically applying that equation to the *test* set.

# boston-4.py

- When we `fit()` the data on line 29, what's effectively happening is we are coming up with an equation that satisfies $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n = t$, where t is the target value, among the *training* set.

- When we `predict()` the data on line 30, we are basically applying that equation to the *test* set.

- When we scatterplot the data on line 36, we have on the x-axis the true (observed) value of the *test* set's data, versus what the model predicted.

# Things to Try!

- What happens if we think some of the columns are irrelevant (like `chas`)? And we don't want them in our data?
  - `frame.drop(labels = 3, axis = 1)`

# Things to Try!

- What happens if we think some of the columns are irrelevant (like `chas`)? And we don't want them in our data?
  - `frame.drop(labels = 3, axis = 1)`

- What if we want to actually plot out the trendline itself? This is why we have `numpy`!
  - See `boston-5.py`, where we try to plot a *linear regression* and calculate a mean-squared error.

# Things to Try!

- What happens if we think some of the columns are irrelevant (like `chas`)? And we don't want them in our data?
  - `frame.drop(labels = 3, axis = 1)`

- What if we want to actually plot out the trendline itself? This is why we have `numpy`!
  - See `boston-5.py`, where we try to plot a *linear regression* and calculate a mean-squared error.

- What if we try a polynominal regression (e.g., n = 2)? Dig into `sklearn`!

# Things to Try!

- What if we try a polynominal regression (e.g., n = 2)? Dig into `sklearn`!

- What if we want to see if any of the other columns (e.g., `crim`) is better predicted by the other 13 than `medv`? Or if a subset of the columns is a better predictor?
  - See `boston-6.py`.

# Thanks!