City University of New York (CUNY)

CUNY Academic Works

Open Educational Resources

City College of New York

2018

Jupyter: Intro to Data Science - Lecture 1 What Data Science Can Tell Us About the World

Grant Long
CUNY City College

NYC Tech-in-Residence Corps

How does access to this work benefit you? Let us know!

More information about this work at: https://academicworks.cuny.edu/cc_oers/170

Discover additional works at: https://academicworks.cuny.edu

This work is made publicly available by the City University of New York (CUNY).
Contact: AcademicWorks@cuny.edu

Lecture 1: What Data Science Can Tell Us About the World

For our first exercise, we'll be using Jupyter, Python 3.6, and the Pandas package to analyze a data set.

Preamble.

This exercise will be an extreme crash course in Python. If you're not familiar with Python, don't worry, it's easy! Future datacamp exercises will focus on the fundamentals of the language. Today's exercise will just involve tweaking some pre-written code.

A couple of notes:

- We're using a jupyter notebook, a web-browser based interactive programming environment that allows you to execute "cells" or blocks of code in chunks. Any variables you assign will persist across different cells.
- To execute a block of code, hit "shift" + "enter". Alternatively, you can use the "Run cell" button in the toolbar above.

The first thing we do when creating a new Jupyter notebook is import all of the packages we'll need for our analysis. For today's example, we just need the pandas package, which we'll alias as pd.

In [ ]:
```
import pandas as pd

%matplotlib inline
```

Great. We're in business. But first, it's time for:

A Very, Very Brief Overview of Python

Just for practice, we can assign a variable, print a statement, and modify the variable.

In [ ]:
```
a = 3
print('Using Python like a rockstar')
```

In [ ]:
```
a = a + 2
a
```

As a reminder, variables in Python are dynamically typed, so we can assign a string to our *a* variable, which was previously a *int*.

In [ ]:
```
a = 'crazy!'
print('This is ' + a)
```

Our Data.

This course is focused on using data to make sense of the world, so we'll do lots of hands on work with data. Each week, we'll explore a data set together, and see what we can learn about the world.

H1-B Visa Data.

H1-B visas are among the most commonly used visas to hire skilled foreign workers in technology, finance, and other sectors. The government publicly discloses roles, titles, and pay for 1,000s of workers in the U.S. This publicly available gives us a sense of the going rate for data analysts, scientist, and engineers around the U.S.

H1B data is made publicly available by the U.S. Department of Labor, and can be found here (https://www.foreignlaborcert.doleta.gov/performancedata.cfm). Record layouts, including column definitions, are avilable here (https://www.foreignlaborcert.doleta.gov/pdf/PerformanceData/2018/H-1B_FY18_Record_Layout_Q3.pdf) The CSV downloaded in this file in an extract of cases submitted in Q2 2018 from the most recent update to the data available in August 2018.

Load data.

Let's use the pandas package to load our data file into a dataframe. We're reading this from the web, but that's not problem, and we can handle the whole operation easily in one line. Note that pandas may warn us that the columns in the file contain mixed data types, but that's not something we need to worry about right now.

In [ ]:
```
df = pd.read_csv('https://grantmlong.com/data/H1B_data.csv')
```

Now that we've got the data loaded, we can look quickly at both the first few rows and the column titles to get a sense of what we're dealing with.

In [ ]:
```
df.head(5)
```

If you look closely, you'll notice that some of the columns in the middle were hidden. Fear not! Pandas allows us to print a full list of columns:

In [ ]:
```
print(list(df))
```

Digging A Little Deeper

Pandas has plenty of built-in functionality that makes it easy to find summary stats about the data we're working with.

```
In [ ]: print(df.shape)
        print(len(df))
```

`.describe()` (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.describe.html) gives us basic summary stats for one or more columns.

```
In [ ]: df.WAGE_RATE_OF_PAY_FROM.describe().astype(int)
```

`.value_counts()` (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.value_counts.html) gives us a series of values and the number of time they appear in a categorical data set.

```
In [ ]: df['CASE_STATUS'].value_counts()
```

```
In [ ]: df.WORKSITE_CITY.value_counts().head(10)
```

```
In [ ]: df.JOB_TITLE.value_counts().head(10)
```

Finally, pandas also includes has some basic visualization functionality, including `.hist()` (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.hist.html)

```
In [ ]: df.WAGE_RATE_OF_PAY_FROM.hist()
```

***Exercise:* Which ten employers hired the most workers?**

```
In [ ]: # todo: top 10 employers by visa count
```

Slicing and Dicing

To get information relevant to our class, we'll need find a little more detail on particular employers, jobs, and locations.

Pandas has a deep arrays of tools that allow for a wide range of in-memory filtering and data extraction, including SQL-like functionality. As is the case with many pandas functions, there is often more than one way to do the same thing.

Generally, the best way to query data frames is using the `.loc[]` (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.loc.html) functionality, essentially allows the user to enter conditions filtering the rows and columns.

```
In [ ]: df.loc[(df.WORKSITE_CITY=='NEW YORK'), 'JOB_TITLE'].value_counts().head(
        10)
```

```
In [ ]: df.EMPLOYER_NAME.loc[(df.JOB_TITLE=='DATA SCIENTIST')].value_counts().he
        ad(10)
```

Exercise: **Which ten employers hired the most workers with salaries over $100,000?**

```
In [ ]: # todo: top 10 employers by visa count for salaries over $100,000
```

Advanced Functionality

Pandas also has functionality to handle more complex datatypes, including:

1. Time functionality to handle dates.
2. Regular expression functionality to handle text

Let's first do some basic transformations of some of our columns to make them easier to work with:

```
In [ ]: # Some of our date columns are a little problematic:
        print(df.CASE_SUBMITTED.values[:5])

        # These looks a lot like strings, but we can transform them to dates, or
        "datetime" objects in pandas
        print(pd.to_datetime(df.CASE_SUBMITTED).values[:5])

        # To make these easier to work with, we can create a new column for the
         transformed date
        df['submission_date'] = pd.to_datetime(df.CASE_SUBMITTED)
```

Now let's look at how we can use the datetime functionality to do some advanced filtering:

```
In [ ]: # Let's filter our set down to all of the records submitted in June 2018
        time_mask = (df.submission_date.dt.year==2018) & (df.submission_date.dt.
        month==6)

        df.loc[time_mask, 'JOB_TITLE'].value_counts().head(10)
```

Pandas also includes built-in string handling, including [regular expression (https://pythex.org/)](https://pythex.org/) searches.

We're primarily interested in data roles, so let's only look at the top firms hiring for those roles.

```
In [ ]: data_roles = df.JOB_TITLE.str.contains(r'DATA')

        df.loc[data_roles, 'EMPLOYER_NAME'].value_counts().head(10)
```

We can combine the mask we created in the preceding cell to find the top data roles with applications submitted in June.

```
In [ ]: df.loc[data_roles & time_mask, 'JOB_TITLE'].value_counts().head(10)
```

Exercise: **What are the top worksites for "engineers"?**

```
In [ ]: # todo: top 10 cities for workers in "engineer" roles
```

All super useful tricks. One last useful set:

Grouping and Calculating Summary Stats

Our last task will be to explore the `groupby()` (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.groupby.html) and `.agg()` (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.core.groupby.DataFrameGroupBy.agg.html) functionality to calculate some basic stats on slices of data, along with the `sort_values()` (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.sort_values.html) to get a sense to the top ranking values.

Let's face it, we're interest in how much folks in data roles get paid. Let's take a look.

```
In [ ]: df.loc[data_roles, 'WAGE_RATE_OF_PAY_FROM'].median()
```

Great, that gives a general sense of the median pay for data roles, but how does that differ by role?

```
In [ ]: df.loc[data_roles, ['JOB_TITLE', 'WAGE_RATE_OF_PAY_FROM']].groupby('JOB_
        TITLE').median()
```

Cool, but way too much information. Let's sort this by the pay column.

```
In [ ]: (df.loc[data_roles, ['JOB_TITLE', 'WAGE_RATE_OF_PAY_FROM']]
         .groupby('JOB_TITLE')
         .median()
         .sort_values(by='WAGE_RATE_OF_PAY_FROM', ascending=False)
         .head(10)
        )
```

Big numbers, but how many visas were issued for this purpose? For this, we can use `agg()` :

In [ ]:
```
(df.loc[data_roles, ['JOB_TITLE', 'WAGE_RATE_OF_PAY_FROM']]
 .groupby('JOB_TITLE')
 .agg(['median', 'count'])
 .sort_values(by=('WAGE_RATE_OF_PAY_FROM', 'median'), ascending=False)
 .head(10)
)
```

Exercise: **What are the top firms for data scientists? How many data scientists are being hired there? How much are they getting paid?**

In [ ]:
```
# todo: top 10 firms for data scientists, with median pay and visa count
```

Time Permitting: What's Interesting Here?

In [ ]:
```
```