Engineering For Data Science Documentation

Joshua Cook

May 27, 2018

CONTENTS

CHAPTER

ONE

CONFIGURATION

1.1 Configuration

1.1.1 The Modeling Problem and the Engineering Problem

As a practicing data scientist, it is likely that you spend the bulk of your time working toward the development of a model for a particular inference or prediciton application. It is less likely that you spend time thinking of the equally complex problems stemming from your system infrastructure. We might trivially think of these two often orthogonal concerns as the **modeling problem** and the **engineering problem**. The typical data scientist is trained to solve the former, often in an extremely rigorous manner, but can often wind up developing a series of ad hoc solutions to the latter.

Since its introduction in 2013, Docker has quickly become a fundamental tool in the design and deployment of robust engineering infrastructure for many applications. From the smallest tech shops to Google, Docker is being used to

- modernize traditional software
- leverage cloud resources for application architecture
- streamline continuous integration and delopyment pipelines
- build out microservices

These will certainly seem downright esoteric to the Data Scientist who is typically concerned with feature importances or how many epochs to run to train a certain neural network.

That said, developing a robust engineering practice with Docker at its core can only make for better data science. This includes immediate concerns such as environment configuration and replicability and presentation of results, but in learning how to use Docker properly the data scientist can ensure that their work is deployed correctly as part and product of their team's software.

Here, I discuss Docker as a tool for the data scientist, in particular in conjunction with the popular interactive programming platform Jupyter and the cloud computing platform Amazon Web Services (AWS). Using Docker, Jupyter and AWS, the data scientist can take control of their environment configuration, prototype scalable data architectures, and trivially clone their work toward replicability and communication.

1.1.2 Getting Started

In this first chapter, you will configure your local system and create an AWS instance in order to do data science work. To do this work, you will use the package management system `conda`, the containerization technology Docker, the version control software `git` and its counterpart cloud-based backup service Github.com, and the system design tool Docker Compose.

There is something of a bootstrap moment in this first chapter. I'm writing this book using Jupyter notebooks. The goal is that you are able to read it and execute the very same code from Jupyter notebooks while you're reading it. That said, you may not even have a Jupyter running on your system. The steps here I designed to take you through step-by-step to install and configure all of the tools you will need to do the work in this book.

Prerequisites

We will assume a basic knowledge of working in bash. This should include things like knowing that ~ is an alias for your home directory, that `pwd` shows your current location, `cd` changes directories and `ls` can be used to list files and in a directory.

Useful tools

Bash

If you are using a Mac OS X system or a Linux system, you will already have Bash available to you in an application called Terminal. If you are using a Windows system, you can use the Anaconda Prompt that will be installed next. If you are on a Mac, you may want to install iTerm (https://iterm2.com).

Conda

It is recommended that you install Conda on your local system. Conda is a package and environment management system for Linux, Mac, and Windows and is perfect for managing our local Python packages. Detailed instructions for installing Conda on your system can be found here: https://conda.io/docs/user-guide/install/index.html

Atom

A simple but extensible text editor such as Atom (https://atom.io) can be an invaluable tool. Atom is available for any modern operating system.

vim

Nearly all of the work we will be doing will be on remote systems. It can be useful to be able to edit text files in place on these remote systems. We have essentially three options to do this:

- Vim
- nano
- Emacs

I will empahsize Vim. Vim is a text editor, like Atom. It is extensible, if not exactly simple. It is available on nearly every system by default, but it can be very challenging to learn. With a little guidance, we will be able to do but we need to do. It should be noted that we will only be using Vim on remote systems. If you would like to spend some time acquainting yourself with Vim, you can type `vimtutor` at a Bash prompt.

SSH Key Pairs

All of the work that you will be doing will take place remotely. As such, there is very little configuration to be done for the local system. The one thing that you will need to do is configure a set of SSH Keys to enable secure connection to the remote system you bring online. We will be using the Secure Shell protocol (SSH) to do the vast majority of

our command line work. It is considered a best practice to use an SSH key pair for authentication when using SSH. An ssh key pair is a set of two long character strings: a private key and a public key. Though they are both called keys, I prefer to think of them as a key and lock.

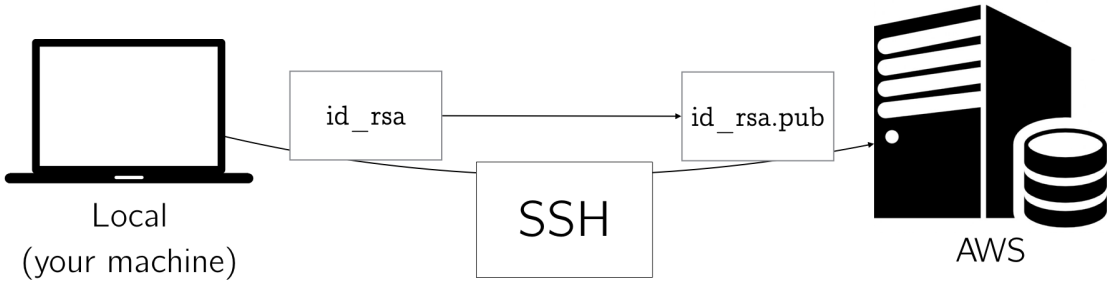


Fig. 1: Connecting with SSH Keys

An SSH Key is a password-less method of authenticating to a remote system using public-key cryptography. Authentication is done using a key pair consisting of a public key (`id_rsa.pub`), which can be shared publicly, and a private key (`id_rsa`), which is known only to the user (See fig **:raw-latex:‘\ref{fig:ssh_keys}‘**). One might think of the public key as the lock on your front door, accessible to anyone, and the private key as the key in your pocket so that only you are able to open your door and gain access to your home.

You will generate this key pair on our local system and then provide the public key to AWS so that it can be added to any system you wish to launch. You will keep the private key on our local system and use it whenever you wish to gain access.

Check to See if SSH Key Pair Exists

You will use the Bash tool `ssh-keygen` to create a new key pair. To begin open a new terminal session:raw-latex:*footnote{On a Mac or Linux system, simply open the Terminal application. On Windows, open Anaconda Prompt.}*, where you will examine whether or not you already have a key pair. Launching a new Bash session will put us in our home directory. The canonical location for storing SSH Keys is in a folder called `~/.ssh` in our home directory. Note that this directory begins with a `.` which makes it a hidden directory. In Bash Command **:raw-latex:‘\ref{lst:ls_home}‘**, you use `ls -la` to display all of the contents of your home directory (`~`) in a list.

We will first check to make sure that you do not already have an SSH key pair.

```
In [1]: ls -la ~
```

```
...
drwxr-xr-x   21 joshuacook  staff    714 Jul 31  2017 .pylint.d
drwx------   11 joshuacook  staff    374 Jan 28 18:49 .ssh
drwxr-xr-x    6 joshuacook  staff    204 Feb  2 22:01 .vim
-rw-------    1 joshuacook  staff  20788 Feb 10 08:54 .viminfo
-rw-r--r--@   1 joshuacook  staff   1263 Jul 26  2017 .vimrc
```

```
drwx------@   5 joshuacook  staff    170 Aug 26 09:12 Applications
drwx------+  19 joshuacook  staff    646 Feb 11 09:32 Desktop
drwx------+   6 joshuacook  staff    204 Feb  4 12:18 Documents
...
```

My local system is running Mac OS X and has the `.ssh` folder already. As the directory already exists, in Bash Command **:raw-latex:‘\ref{lst:ls_ssh}‘**, I list the contents of my `.ssh` directory.

```
In [4]: ls -la ~/.ssh
```

```
total 24
drwxr-xr-x  8 jovyan users  272 Apr 24 23:07 .
drwxr-xr-x 21 jovyan users  714 Apr 25 00:39 ..
-rw-------  1 jovyan users 1679 Apr 24 23:06 id_rsa
-rw-r--r--  1 jovyan users  418 Apr 24 23:06 id_rsa.pub
```

As can be seen, I already have an SSH Keypair named `id_rsa` and `id_rsa.pub`. If this is true for you, as well, you should skip the next step and not create new SSH Keys. If you do not have these keys, proceed to Bash Command **:raw-latex:‘\ref{lst:create_new_ssh_key}‘**.

Create a new SSH Key Pair

If when listing the home directory, you do not see a folder called `.ssh` or when displaying the contents of `.ssh` you do not see an SSH Keypair named `id_rsa` and `id_rsa.pub`, a new SSH Keypair will need to be created. In Bash Command **:raw-latex:‘\ref{lst:create_new_ssh_key}‘**, you create a new SSH Keypair using the `ssh-keygen` command line utility.

During the creation of the SSH Keypair, you will be prompted three times. The first asks where you should save the SSH Keypair, defaulting to the `.ssh/id_rsa` in our home directory. In Bash Command **:raw-latex:‘\ref{lst:create_new_ssh_key}‘**, you see that this is being done at `/Users/joshuacook/.ssh/id_rsa` on my local system where my username is `joshuacook`. The second and third prompts will ask for a passphrase to be added to the key. For our purposes, leaving this passphrase empty will be fine. In other words, the default options are preferable and you may simply hit `<ENTER>` three times.

```
In [1]: ssh-keygen
```

```
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/joshuacook/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in id_rsa.
Your public key has been saved in id_rsa.pub.
The key fingerprint is:
SHA256:p5KeEomPt6izFC5gaFphfx3zw8aAB+D8RiUA1/nEsUc joshuacook@LOCAL
The key's randomart image is:
+---[RSA 2048]----+
|  ..++ooo.E      |
|   +  o=oo       |
|  o o oo* .      |
|.. o o o.O       |
|o+...+ S B       |
|*.o oo . + .     |
|oo o .o .        |
|+ ..+. o         |
|o+...oo          |
+----[SHA256]-----+
```

You can verify the SSH Keypair you just created by displaying the Public Key in our shell (See Bash Command **:raw-latex:‘\ref{lst:cat_pub_key}‘**). Here, you use the `cat` command, which concatenates the contents of `id_rsa.pub`

to the shell output.

```
In [1]: cat ~/.ssh/id_rsa.pub
```

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDdnHPEiq1a4OsDDY+g9luWQS8pCjBmR
64MmsrQ9MaIaE5shIcFB1Kg3pGwJpypiZjoSh9pS55S9LckNsBfn8Ff42ALLj
R8y+WlJKVk/0DvDXgGVcCc0t/uTvxVx0bRruYxLW167J89UnxnJuRZDLeY9fD
OfIzSR5eglhCWVqiOzB+OsLqR1W04Xz1oStID78UiY5msW+EFg25Hg1wepYMC
JG/Zr43ByOYPGseUrbCqFBS1KlQnzfWRfEKHZbtEe6HbWwz1UDL2NrdFXxZAI
XYYoCVtl4WXd/WjDwSjbMmtf3BqenVKZcP2DQ9/W+geIGGjvOTfUdsCHennYI
EUfEEP joshuacook@LOCAL
```

That is the sum of the local configuration you will need to do in order to get started.

1.1.3 Amazon Web Services

If you have not already done so, set up an AWS account:raw-latex:*footnote{As of 2017/12/19, detailed instructions for doing this can be obtained here: https://aws.amazon.com/premiumsupport/knowledge-center/create-and-activate-aws-account/}*. You will be using AWS to manage the hardware upon which your data science platform will run. We will leave the details of what exactly "hardware" means to AWS. This is to say that AWS may be allocating resources as a virtual machine, but for your purposes, the experience will be as if you are using a physical system across the room from you.

The most popular service offered by Amazon Web Services is the Elastic Compute Cloud (EC2), "a web service that provides secure, resizable compute capacity in the cloud"**:raw-latex:`\footnote{https://aws.amazon.com/ec2/}`**. For our purposes, compute capacity means a cloud-based computer you will use to run your platform. What we learn should generalize to other cloud providers such as DigitalOcean or Google cloud platform.

If you are new to AWS you will be able to work through this text using the AWS Free Tier:raw-latex:*footnote{https://aws.amazon.com/free/}*. For the first 12 months following sign up, new users receive 750 Hours per month of EC2 time. This amounts to 31.25 days of availability and, provided that readers keep only one server running at a time, ensures that readers can work through this text at no cost.

Configure your AWS Account

The next thing you will need to do is configure your AWS Account.

This will involve:

1. Configure a Key Pair
2. Creating a Security Group

The AWS Key Pair is slightly misnamed as it is not in fact a pair, but rather is simply the public portion of the SSH Key Pair you have on our local system. You will simply add the Public Key from the SSH Key Pair you just created.

To begin, log in to your AWS control panel and navigate to the EC2 control panel (fig. **:raw-latex:`\ref{fig:access_ec2_dash}`**). First, access "Services" (fig. **:raw-latex:`\ref{fig:access_ec2_dash}`**, #1) then access "EC2" (fig. **:raw-latex:`\ref{fig:access_ec2_dash}`**, #2). The Services link can be accessed from any page in the AWS website.

Configure a Key Pair

Once at the EC2 control panel, access the Key Pairs pane using either link (fig. **:raw-latex:`\ref{fig:access_key_pairs}`**).

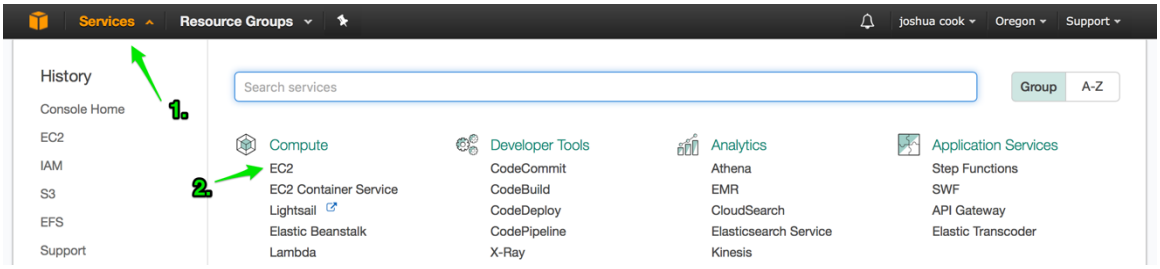


Fig. 2: Access EC2 Dashboard

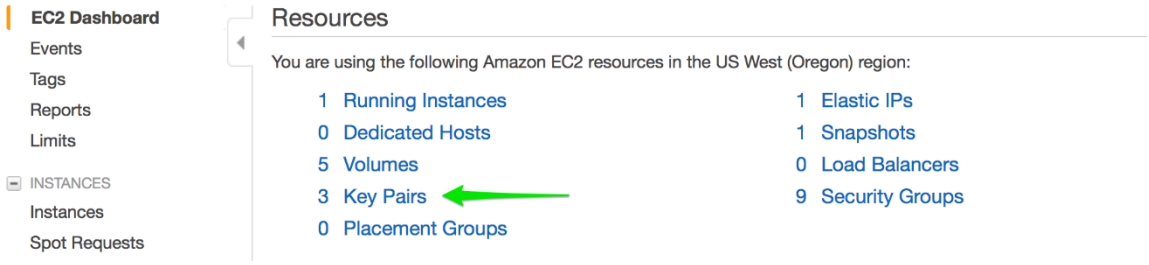


Fig. 3: Access Key Pairs from the EC2 control panel

From the Key Pairs pane, choose "Import Key Pair." This will activate a modal that you can use to create a new key pair associated with a region on your AWS account. Make sure to give the key pair a computer-friendly name, like `from-MacBook-2018`. Paste the contents of your public key (`id_rsa.pub`) into the public key contents. Prior to clicking Import, your key should appear as in fig. **:raw-latex:'\ref{fig:import_public_key}'**. Click Import to create the new key.

You have just created a key pair between AWS and your local system. When you create a new instance, you will instruct AWS to provision the instance with this public key and thus you will be able to access the cloud-based system from your local system using your private key.

Ports & Security Groups

A security group is a set of ports public facing Internet. This may be a new concept for you and we will not dig very deeply into it. Suffice it to say that we need a short list of ports to be available to us for accessing the different services we might configure. A port is a number appended to an IP address or domain name with a colon like this

```
192.168.99.100:3000
```

Here, `192.168.99.100` is the IP address and `3000` is the port. A port can be thought of as a channel over which a service will listen for requests. You have already been using a few ports without knowing that you are. All internet traffic is routed using ports 80 and 443 by default. What this means is that visiting http://google.com:80 is equivalent to visiting http://google.com and visiting http://google.com:443 is equivalent to visiting https://google.com.

As we are largely concerned with learning, we are not concerned with the specifics of high–availability, and for us networking best practices will consist of making sure that things work. If you intend on putting work that you do here into production, you should certainly consult with your local Site Reliability Engineer.

For security purposes, by default Amazon closes all ports to outside traffic. This is why we will need to create a security group to open the ports that we need. These include the following:

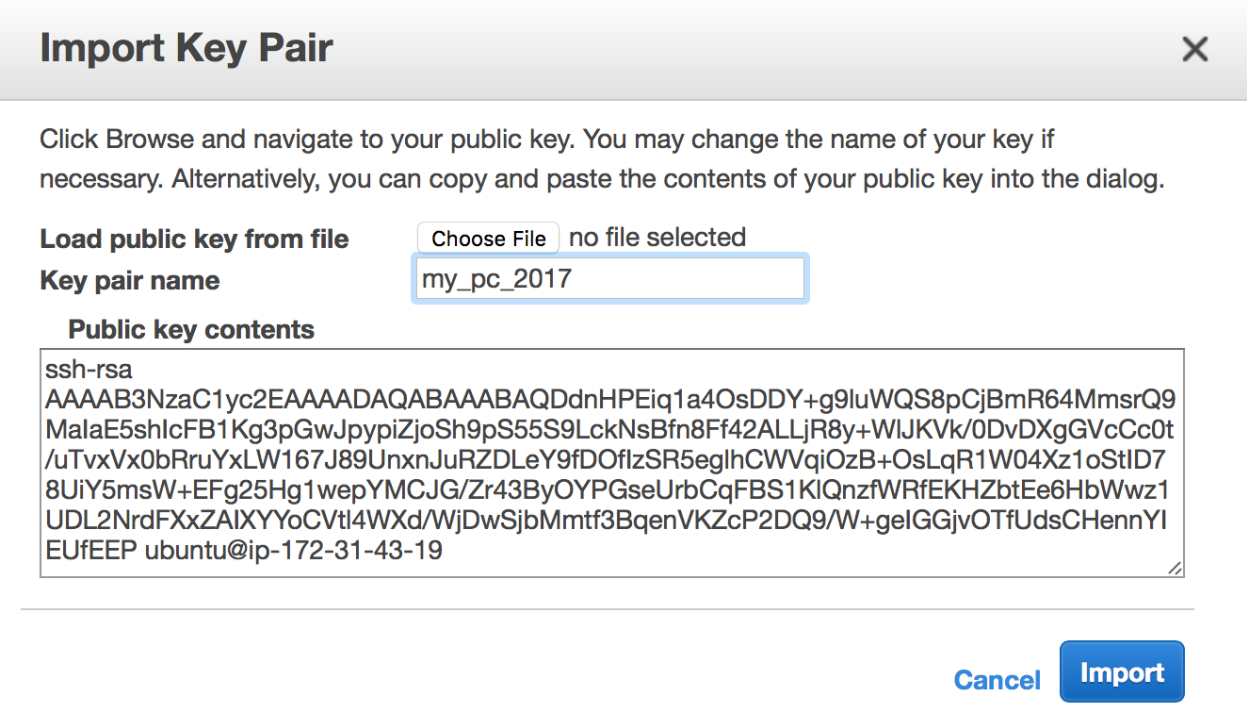


Fig. 4: Import a New Public Key

port	service
22	ssh
80	http
443	https
5000	miscellaneous
5432	PostgreSQL
6379	Redis
8888	Jupyter
27017	Mongo

Create a New Security Group

From the EC2 Control panel, access Security Groups (See fig. **:raw-latex:‘\ref{fig:ch-01-access_security_group}‘**).

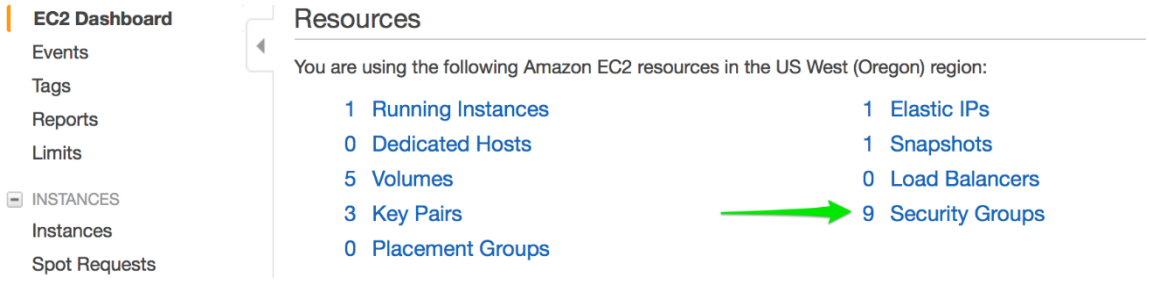


Fig. 5: Access Security Groups from the EC2 control panel

From the Security Group pane, click "Create Security Group." Give the security group a computer friendly group name like `ds_engineering`. Give the security group a description like "Open access to important Data Science related services". Use the default VPC.

Access the Inbound tab (See fig. **:raw-latex:`\ref{fig:ch_01-create_new_security_group}`**, #1) and configure the security rules listed above. **Make sure to set Source to "Anywhere"**.

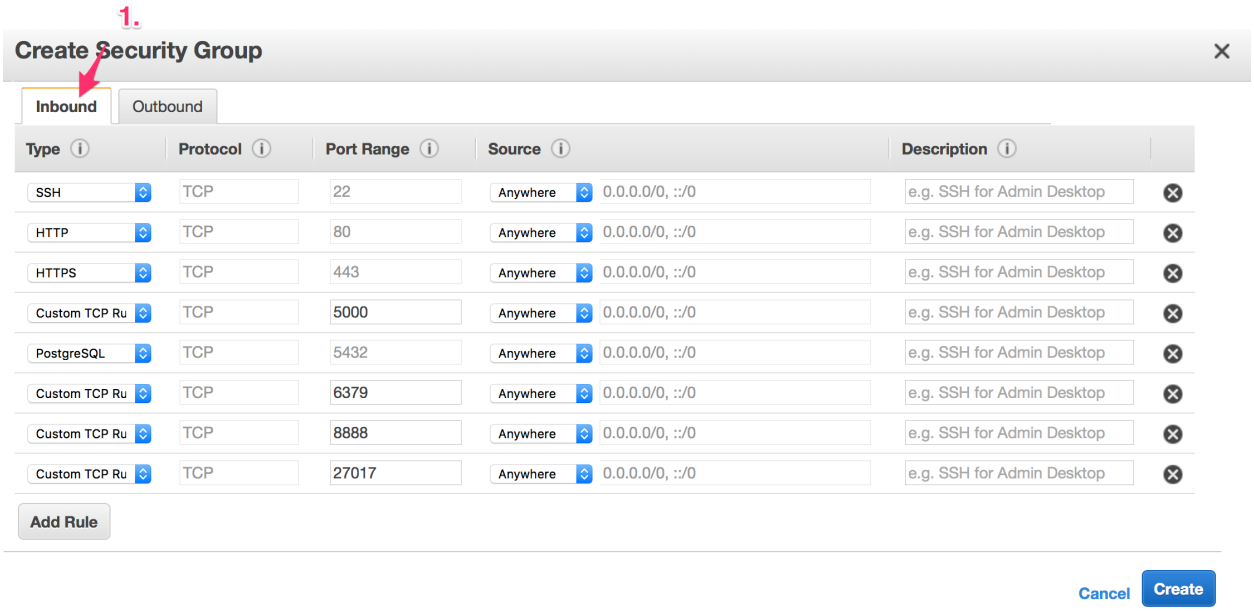


Fig. 6: Create a New Security Group

CHAPTER

TWO

ELASTIC COMPUTE CLOUD

2.1 Elastic Compute Cloud

2.1.1 Launch a New AWS EC2 Instance

AWS EC2 `t2.micro`

AWS EC2 virtual machines are available to meet a host of different applications and purposes. The `m` series and the `t` series are considered in general purpose and are adequate for our needs. Additionally, the `t2.micro` from the `t` series is considered a machine "free-tier" and can be run for free under certain circumstances. We will use this type of machine as often as possible.

Launch Instance

To create a new instance, start from the EC2 control panel and click the Launch Instance button (fig. **:raw-latex:'\ref{fig:launch_instance}'**).

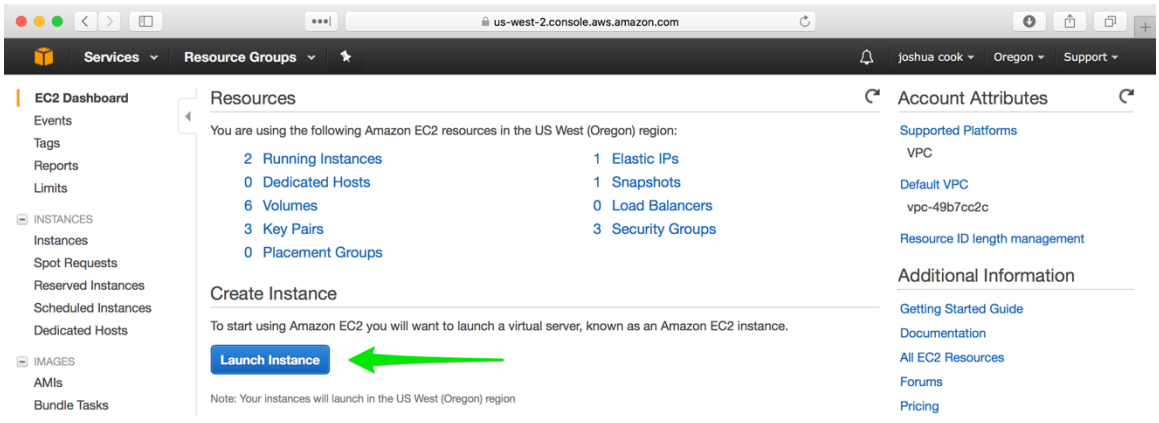


Fig. 1: Begin the launch process for a new instance

Step 1: Choose an Amazon Machine Image (AMI)

The launching of a new instance is a multi-step process that walks the user through all configurations necessary. The **first tab** is "Choose AMI." An AMI is an Amazon Machine Image:raw-latex:*footnote{http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html}*. and contains the software you

will need to run your sandbox machine. I recommend choosing the latest stable Ubuntu Server release that is free-tier eligible. At the time of writing, this was ami-efd0428f, Ubuntu Server 16.04 LTS (HVM), SSD Volume Type (fig. **:raw-latex:‘\ref{fig:latest_ubuntu}‘**).

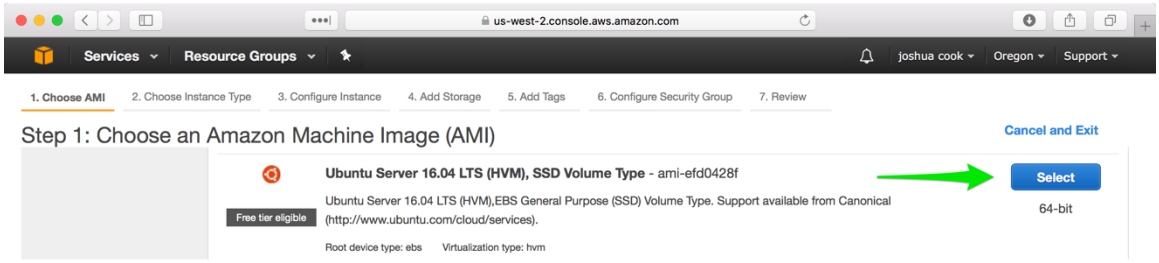


Fig. 2: Choose the latest stable Ubuntu Server release as AMI

Step 2: Choose Instance Type

The **second tab** is “Choose Instance Type.” In practice, I have found that the free tier, `t2.micro` (fig. **:raw-latex:‘\ref{fig:choose_type}‘**), is sufficient for many applications. Furthermore, the instance type may always be changed later should the need present itself.

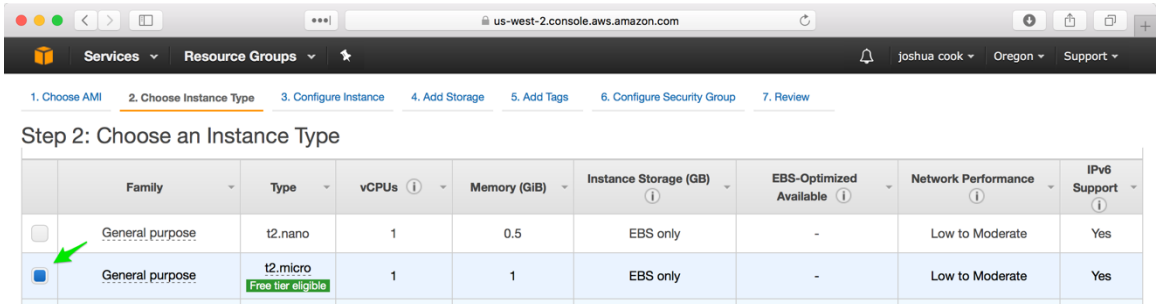


Fig. 3: Choose `t2.micro` for Instance Type

Step 3: Configure Instance Details

The **third tab**, “Configure Instance,” can be safely ignored.

Step 4: Add Storage

The **fourth tab** is “Add Storage.” This option is also specific to intended usage. It should be noted that Jupyter Docker images can take up more than 5GB of disk space in the local image cache. For this reason, it is recommended to raise the value from the default 8GB to 30GB. Furthermore, as noted on this tab:

> Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage.

Step 5: Add Tags

The fifth tab, “Add Tags,” can be safely ignored.

Step 6: Configure Security Group

The sixth tab, "Configure Security Group," is critical for the proper functioning of your systems. By default this tab will be set up to "Create a **new** security group". This will not work for us! Ultimately, we will be accessing our system via a web browser which we require at a minimum that port 80 is open. We recommend simply using the default group which will open our system on all ports. If greater security is required for your specific application a more restrictive security group may be defined and used.

Select the "default" security group (fig. **:raw-latex:`\ref{fig:default_security_group}`**).

Fig. 4: Choose the latest stable Ubuntu Server release as AMI

☐ **Note:** You may receive a Warning stating, "Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only." This is expected and is okay.

Step 7: Review Instance Launch

Finally, click "Review and Launch." Here, you see the specific configuration of the EC2 instance you will be creating. Verify that you are creating a `t2.micro` (fig. **:raw-latex:`\ref{fig:review_and_launch}`**, #2)running the latest free tier-eligible version of Ubuntu Server (fig. **:raw-latex:`\ref{fig:review_and_launch}`**, #1)and that it is available to all traffic (fig. **:raw-latex:`\ref{fig:review_and_launch}`**, #3), and then click the Launch button (fig. **:raw-latex:`\ref{fig:review_and_launch}`**, #4).

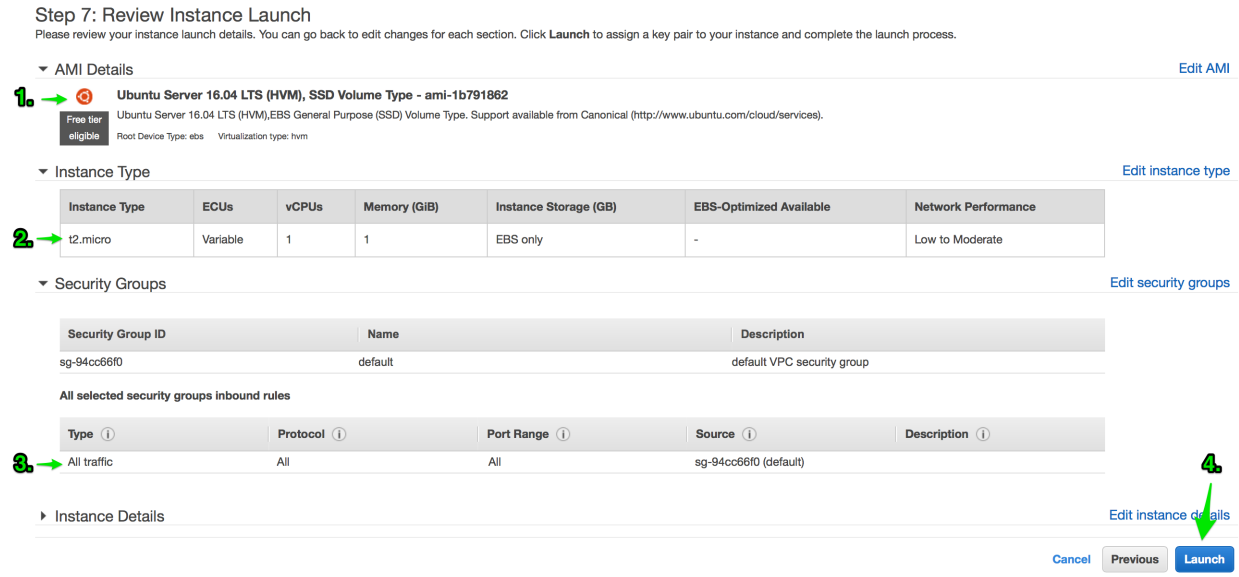


Fig. 5: Review and launch the new instance

Add an SSH Key

In a final confirmation step, you will see a modal titled "Select an existing key pair or create a new key pair." Select the key pair you previously created. Check the box acknowledging access to that key pair and launch the instance (fig. **:raw-latex:`\ref{fig:add_key_pair}`**).

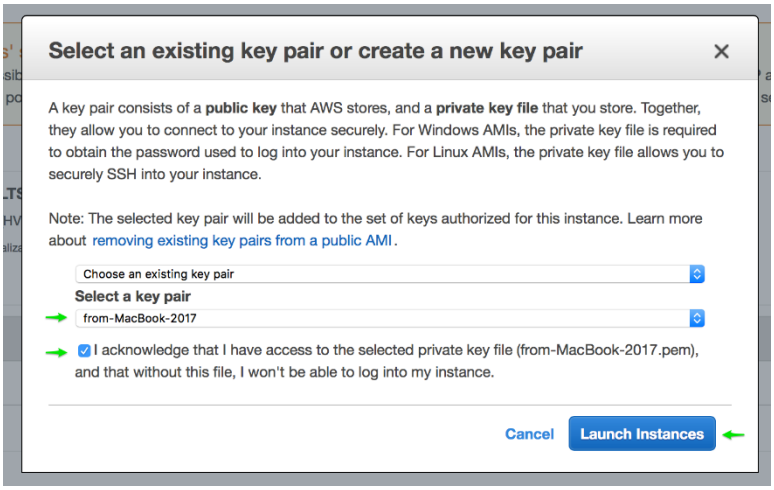


Fig. 6: Add a key pair to the instance

☐ **Note:** If this step is not done correctly, that is, if the correct key pair is not added to the launching instance, the instance will need to be terminated and a new instance will need to be launched. There is now way to add a key pair to a running instance.

You should see a notification that the instance is now running. Click the View Instances tab in the lower right corner to be taken to the EC2 Dashboard Instances pane, where you should see your new instance running.

Examing the Newly Launched Instance

Make note of the IP address of the new instance (fig. **:raw-latex:`\ref{fig:new_ip}`**).

Configure Git & Github

Armed with our new ssh key, we will add the public key to Github so that we can use the key to access our github account.

1. copy the public key
2. choose

The first thing we will do is create a new local `git` repository and a remote Github repository. The local repository is where we will do all of our work. We will use the remote Github repository to track all of the work that we will do. It is a common beginning misconception to think of `git` and Github as the same thing. `git` is a commond line tool we will use to track changes to our project. Github is a cloud-based service designed to work with `git` to help us to make sure our work is always backed up on an additional system.

Create the new repository on Github

At github.com, in the upper-right hand corner, click the plus icon and select "New Repository".

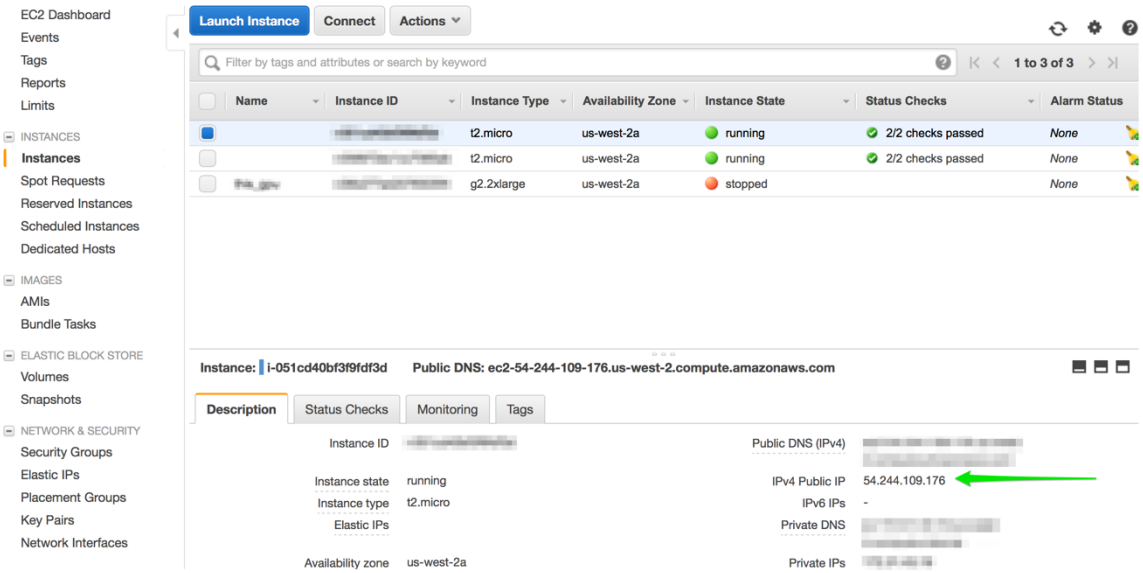
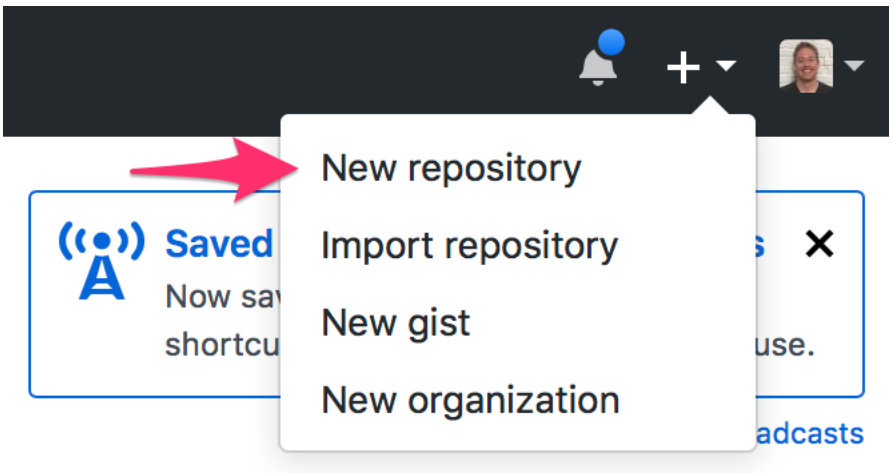
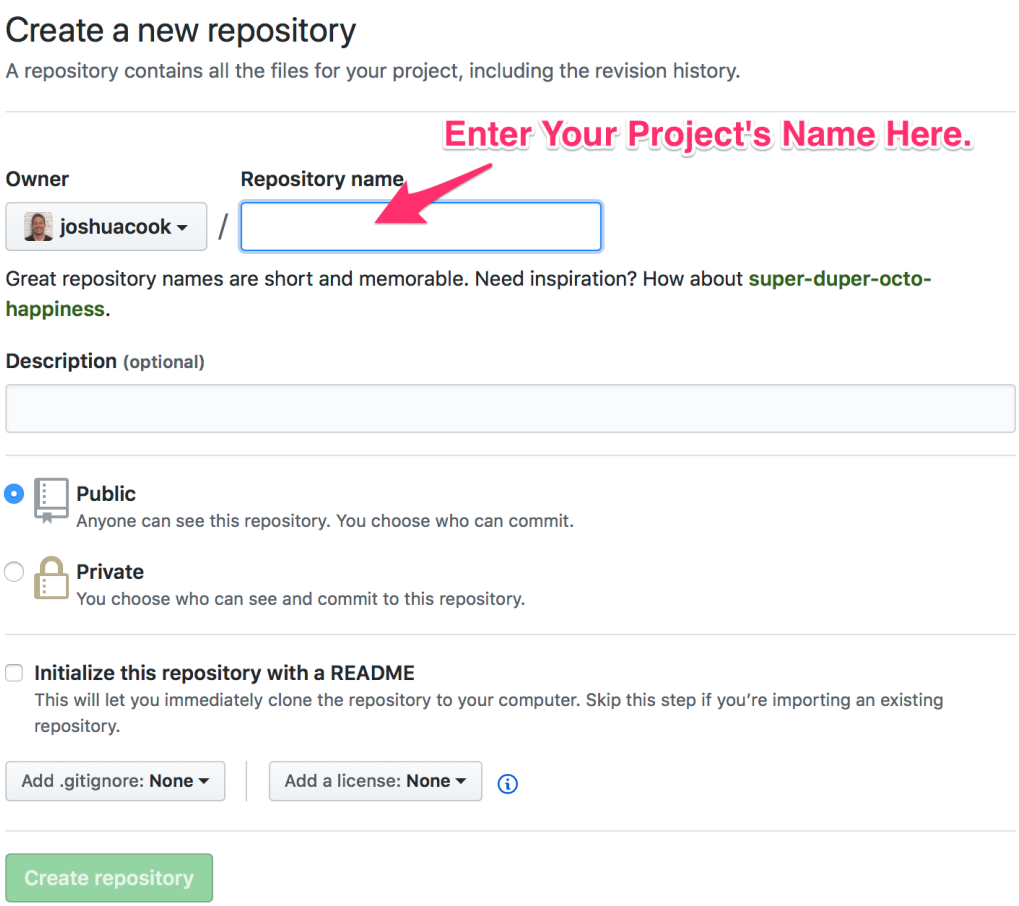


Fig. 7: Add a key pair to the instance



Give your new repository a name. It does not need to match `engineering-for-data-science`, but it is generally considered a best practice to make sure the repository on Github and the local directory containing your files have the same name. You do not need to provide a description nor create a `README` file.

Finally, click "Create Repository".



Make a new directory

We use the `mkdir` command to create a new directory to hold our work.

```
In [9]: mkdir -p engineering-for-data-science
```

Change directories to the new directory

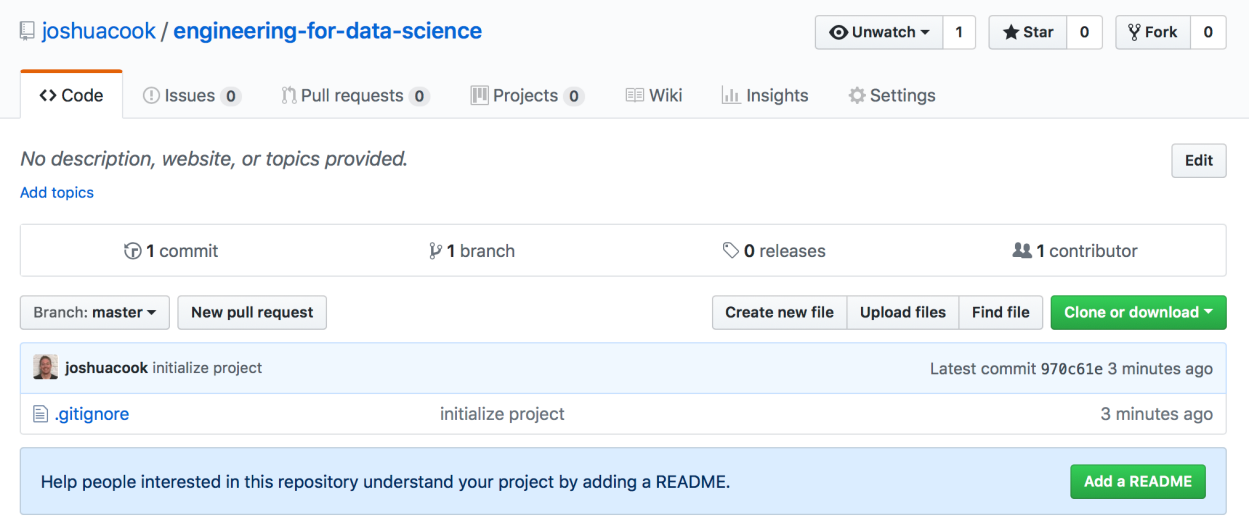
Next, use the `cd` (change directory) command to change directories to the new directory we just created.

```
In [10]: cd engineering-for-data-science
In [11]: git init
Initialized empty Git repository in /home/jovyan/engineering-for-data-science/.git/
In [12]: mkdir chapter-01-introduction
In [13]: echo ".ssh" > .gitignore
```

```
In [14]: ls

chapter-01-introduction

In [15]: ls -la

total 4
drwxr-xr-x  5 jovyan users 170 Apr 24 23:53 .
drwxr-xr-x 20 jovyan users 680 Apr 24 23:51 ..
drwxr-xr-x  2 jovyan users  68 Apr 24 23:52 chapter-01-introduction
drwxr-xr-x 10 jovyan users 340 Apr 24 23:51 .git
-rw-r--r--  1 jovyan users   5 Apr 24 23:53 .gitignore

In [16]: git add .gitignore && git commit -m 'initialize project'


*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: empty ident name (for <jovyan@eb5eb401ee58.(none)>) not allowed

In [17]: git config --global user.email "me@joshuacook.me"

In [18]: git config --global user.name "Joshua Cook"

In [19]: git add .gitignore && git commit -m 'initialize project'

[master (root-commit) 970c61e] initialize project
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore

In [20]: git remote add origin git@github.com:joshuacook/engineering-for-data-science.git

In [21]: git push -u origin master

Counting objects: 3, done.
Writing objects: 100% (3/3), 222 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:joshuacook/engineering-for-data-science.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

2.1.2 Git and Github

As you work through this text, you will be developing a series of data science projects. Tracking software development work is typically done using version control software. One of the most popular version control tools is `git`. Additonally, it can be useful to use a version control hosting service as a remote backup for work being tracked using `git`. The remote service we will use is Github.com. In my experience, learners who are new to version control often confuse `git` and Github, so it bares repeating – we will use `git` to track changes we make to our code and Github as a remote backup for these changes.



2.1.3 Configuring Github

We will assume that you have a Github account. Once this has been done, you will need to configue an SSH connection between AWS and Github. This next part may potentially create a confusion. We are actually going to need a new SSH Keypair, this one associated with our AWS instance. This is because it is our AWS instance that will be connecting to Github, not our local machine (See fig. **:raw-latex:‘\ref{fig:ssh_local_remote}‘**).

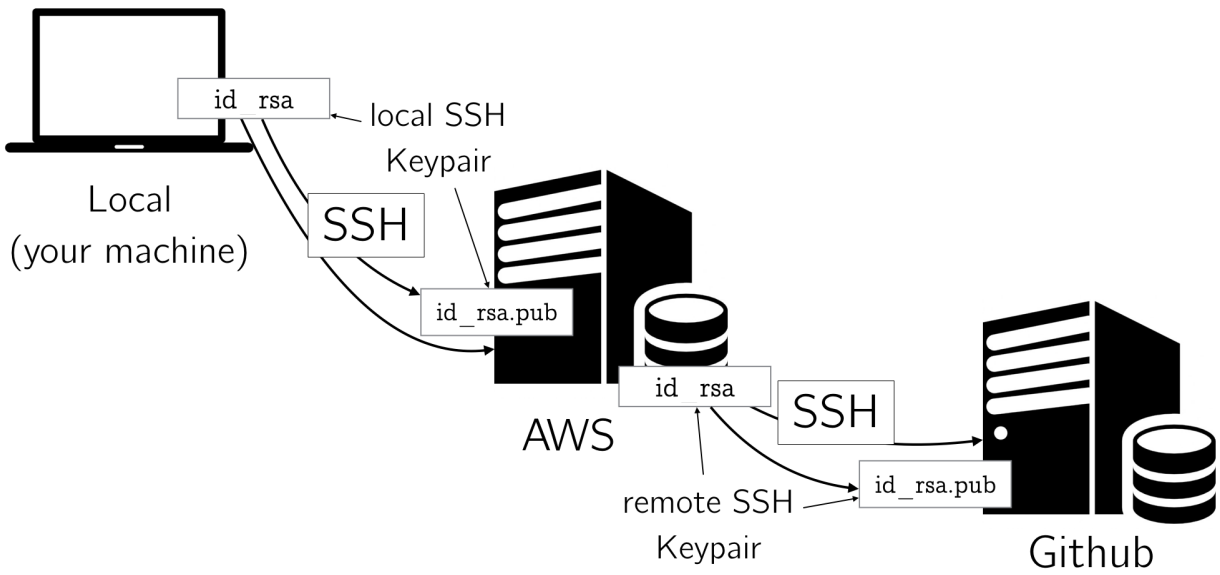


Fig. 8: SSH Connections

Create a New Key Pair

In [@lst:create_new_ssh_key_remote], you create a new key pair on your remote AWS instance. In [@lst:ssh_into_new_instance], you connect to your new AWS instance. To do this we will use the IP address we

made note of in fig. **:raw-latex:‘\ref{fig:new_ip}‘**. We use SSH to connect to our remote AWS instance. Note that we use the username, `ubuntu`, the default username for the Ubuntu 16 AMI provided by AWS.

Listing: Create a new SSH Keypair

```
$ ssh ubuntu@54.244.109.176
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-64-generic x86_64)
```

☐ **Note:** The first time you access your EC2 instance, you should see the following message: `The authenticity of host '54.244.109.176 (54.244.109.176)' can't be established ... Are you sure you want to continue connecting (yes/no)? This is expected. You should hit <ENTER> to accept or type yes and hit <ENTER>.`

In [@lst:create_new_ssh_key_remote], you create a new SSH Keypair on our remote AWS instance. Again, during the creation of the SSH Keypair, you will be prompted three times. The first asks where you should save the SSH Keypair, defaulting to the `.ssh/id_rsa` in our home directory. In [@lst:create_new_ssh_key_remote], you see that this is being done at `/home/ubuntu/.ssh/id_rsa`**:raw-latex:‘\footnote{This should be the same for everyone now, as you should be working on an AWS \texttt{t2.micro} running an Ubuntu system where the user’s name is \texttt{ubuntu}}‘**. The second and third prompts will ask for a passphrase to be added to the key. For our purposes, leaving this passphrase empty will be fine. In other words, the default options are preferable and you may simply hit `<ENTER>` three times.

Listing: Create a new SSH Keypair

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ubuntu/.ssh/id_rsa):
Created directory '/home/ubuntu/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ubuntu/.ssh/id_rsa.
Your public key has been saved in /home/ubuntu/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:ZSpFpgSRgRqlQom8yVBG2dZo1tgkPQdrmUGgMXGDtRY
ubuntu@ip-172-31-43-19
The key's randomart image is:
+---[RSA 2048]----+
|o=XBE/*.o        |
|==+=O**O.        |
|=o++o *o. o      |
|o+ . . . +       |
|       . S       |
|        .        |
|                 |
|                 |
|                 |
+----[SHA256]-----+
```

As before, you can verify the SSH Keypair you just created by displaying the Public Key in your shell ([@lst:cat_pub_key_remote]). Again, you use the `cat` command, which concatenates the contents of `id_rsa.pub` to the shell output.

Listing: Display Public SSH Key

```
$ cat ~/.ssh/id_rsa.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDQ896GUMgCMAIW79gwF3ojRjcUYCKUKc8b+q
iQlah2jtr7s0K4WRGjktOy3lCCHO+1UK/GrzY1Y4VxCKoKJDH3G9N5UzyGhlxa/2Ah
```

(continues on next page)

(continued from previous page)

```
kKxzHht1knyh/mkVGqYUhuHpXfxUQAstCFrIdp3G0MDPiko2qeJcBF7JSv1lLMbIuM
XuVU/Mzq6BU+tEogScYytmLckyEe1j8RJ+e5nBURwmkgj3UAN1DzmU/lVwLlltEpmC
DlOel4yEXAw8yBwM3GwjahfiBThvBHpsc43HxWrkM8Yi/kdDnvsDZYxU4zhXZPsPab
UY/LfxEod9c6Sui5W8GtAfdi6krnqbzxrKt81Mradh ubuntu@ip-172-31-43-19
```

Add the Public Key to Github

Previously, you added your local SSH public key to your AWS account. Now, you will add your AWS SSH public key to your Github account:raw-latex:*footnote{https://help.github.com/articles/adding-a-new-ssh-key-to-your-github-account/}*. First, access the **Settings** for your account by clicking the profile photo in the upper-right corner of any page on Github. Next, in the user settings sidebar, select **SSH and GPG keys**. On the SSH and GPG Keys page, click **New SSH key**. On the next page, give your key a descriptive title e.g. "AWS Feb 2018" and then paste your AWS public key in to the "Key" field . Finally, click **Add SSH key** and confirm your Github password , if prompted.

Learning to read the Bash Prompt

During your work you will no doubt notice that an idle SSH connection may become disconnected and/or unresponsive. Should this happen, simply close the terminal session, launch a new one, and reconnect to the remote instance.

The most important thing is that you are aware of which system your current shell session is connected to. Shell prompts are designed to relay this information to you immediately. If you are new to working with Bash, you may need to train yourself to being aware of the prompt when typing. [@lst:default_AWS_prompt] shows the default AWS Bash prompt. The information contained is the username, `ubuntu`, and the private IP address of the AWS instance. **This is not the public address you use to connect**. What is useful about this, is that we can immediately see that the user is `ubuntu`. This tells us we are connected to AWS.

Listing: The default AWS Bash prompt

```
ubuntu@ip-172-31-21-89:~$
```

Your local system will no doubt display something different (See [@lst:other_prompt]). Again, the important thing is to take note of what is displayed by the prompt and to learn to associate that prompt with the correct system. As you become a more advanced Bash user, you may wish to personalize your prompt, but for now it is imperative that you learn to read the prompt in order to always know to which system you are connected.

Listing: A local Bash prompt

```
joshuas-macbook-pro:~$
```

Test your SSH Connection to Github

Having added you AWS Public Key to your Github account, you should verify your SSH connection from your AWS instance. In [@lst:verify_github_ssh], we attempt to connect to Github via SSH. As before, we receive a message about the authenticity of the connection. Again, type `yes`, and continue. If successful, you will see a message telling you have successfully authenticated but that Github does not provide shell access.

Listing: Verify Github SSH Key

```
ubuntu@ip-172-31-21-89:~$ ssh -T git@github.com
The authenticity of host 'github.com (IP ADDRESS)' can't be
established.
RSA key fingerprint is
```

(continues on next page)

(continued from previous page)

```
16:27:ac:a5:76:28:2d:36:63:1b:56:4d:eb:df:a6:48.
Are you sure you want to continue connecting (yes/no)? yes
Hi username! You've successfully authenticated, but GitHub does
not provide shell access.
```

Docker & Docker Compose

Having configured our SSH connections and provisioned a new AWS EC2 instance, it is time to get to the business of building your data science platform. To do this you will use the containerization platform Docker and its Docker Compose tool. While Docker is very easy to use, it can be difficult to understand for the uninitiated. In an earlier work, *Docker for Data Science*:raw-latex:‘\footnote{https://www.apress.com/us/book/9781484230114}‘, I wrote:

> [Using Docker] we add a layer of complexity to our software, but in doing so gain the advantage of ensuring that our local development environment will be identical to any possible environment into which we would deploy the application.

It may be simpler, however, to simply think about using Docker as a way to manage a running process. Your system will be running two processes: an IPython shell and a PostgreSQL server. Were you to not use Docker, you would need to ensure that the AWS instance had all of the libraries required to run both of those processes (and keep those libraries up to date).

Instead, you will let Docker manage the processes using a container for each process. Each respective container will be run using a predefined image built using best practices and ready to run their respective process. The exchange is this: you will take on the congitive burden of *understanding* what Docker is doing and Docker (and the Docker community) will take over the burden of making sure that your processes run.

2.1.4 Docker Compose

Docker Compose is a tool built for managing an application consisting of multiple containers. Using Docker Compose, it is possible to completely define an application using a simple text file. To make this conversation less abstract, let’s have a look at the `docker-compose.yml` file you will use to define your first application (See [@lst:docker_compose]).

Listing: Your Data Science Application

```
version: "3"
services:
  ipython_shell:
    image: jupyter/scipy-notebook
  database:
    image: postgres
    volumes:
      - postgres_data:/var/lib/postgresql/data
volumes:
  postgres_data
```

That’s it. This simple file completely defines a fully-functioning Data Science Application. In it, we define the two services we need: `ipython_shell` and `database`. These two services are defined using the `jupyter/scipy-notebook` and `postgres` images. When we launch the application, the images will be pulled from Docker Hub into our local memory and then launched. The one other thing we do is create a data volume `postgres_data`. We will use this as the data volume for our database server so that if for some reason we have to shut our system down, we do not lose our data. The data will exist on this volume independent of the services.

☐ **Note:** Throughout this text, when discussing infrastructure, I may casually refer to containers, services, and processes. At the risk of annoying your local site reliability engineer, you may treat these as terms as synonomous. Care should be taken, however, not to confuse services/containers/processes and images. An image defines a service, but a service should be thought of as a living and active thing. You may loosely compare the service-image relationship to the object-class relationship in Object-Oriented Programming. A service is a running container defined by an image, just like an object is an instance of a class that exists in memory.

2.1.5 Installing and Configuring Docker

Installing Docker on your AWS instance is a downright trivial process. It consists of running an install script that can be obtained from Docker and then adding your user to the Docker group. In [@lst:install_docker], we run these two commands. First, we download the install script from https://get.docker.com, then immediately pipe the script into the shell (`| sh`).

☐ **Note:** It is generally considered to be a significant security vulnerability to execute arbitrary code obtained from an unknown, or untrusted source. For our purposes, the source (https://get.docker/com) is considered trustworthy, we are using SSL to perform the curl, and in practice this is the method I use to install Docker. Still, it may make the security minded more comfortable to `curl` the script, inspect, and then run it.

Listing: Install Docker via a Shell Script

```
$ curl -sSL https://get.docker.com/ | sh
# Executing docker install script, commit: 1d31602
+ sudo -E sh -c apt-get update -qq >/dev/null
...

Client:
 Version:   18.02.0-ce
 API version:   1.36
 Go version:    go1.9.3
 Git commit:    fc4de44
 Built: Wed Feb  7 21:16:33 2018
 OS/Arch:   linux/amd64
 Experimental:  false
 Orchestrator:  swarm

Server:
 Engine:
  Version:  18.02.0-ce
  API version:  1.36 (minimum version 1.12)
  Go version:   go1.9.3
  Git commit:   fc4de44
  Built:    Wed Feb  7 21:15:05 2018
  OS/Arch:  linux/amd64
  Experimental: false

...
```

When the script completes there is one last thing to be done. In [@lst:add_to_docker_group], you add the `ubuntu` user to the `docker` group. By default, the command line docker client will require sudo access in order to issue commands to the docker daemon. You can add the `ubuntu` user to the `docker` group in order to allow the `ubuntu` user to issue commands to docker without sudo.

Listing: Add the Ubuntu User to the Docker Group

```
$ sudo usermod -aG docker ubuntu
```

Finally, in order to force the changes to take effect, you should disconnect and reconnect to their remote system. You can achieve this by typing `exit` or `ctrl-d` and then reconnecting via ssh to your EC2 instance.

2.1.6 Installing and Configuring Docker

Recall that regardless of your local operating system, you are working on an AWS EC2 Instance running the Linux variant, Ubuntu. As such, `docker-compose`can be installed using the instructions provided here: https://github.com/docker/compose/releases, which are written specifically for Linux machines. As of the writing of this book, this consists of two steps.

In [@lst:curl_docker_compose], you use `curl` to retrieve the `docker-compose` binary from Github. As of the writing of this book. the latest version of `docker-compose` was `1.19.0`. You should retrieve the latest version from the above url.

Listing: Retrieve `docker-compose` binary from Github

```
$ sudo curl -L https://github.com/docker/compose/releases/download/1.19.0/docker-
↪compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
```

In [@lst:chmod_docker-compose], we use the `chmod`**:raw-latex:‘\footnote{The unix "change mode" utility. I pronounce it "shmod".}‘** utility to allow `docker-compose` to be executed (`+x`).

Listing: Enable Docker Compose to be Executed

```
$ sudo chmod +x /usr/local/bin/docker-compose
```

Finally, in [@lst:docker_compose_version], we check the version of `docker-compose` against what we expect to have installed.

```
$ docker-compose -v
docker-compose version 1.19.0, build 9e633ef
```

TODO

1. What went wrong here? What should you do?

   ```
   failed to register layer: Error processing tar file(exit status 1):
   write / usr/bin/python2.7: no space left on device
   ```

2. What went wrong here? What should you do?

   ```
   docker: Error response from daemon: driver failed programming external
   connectivity on endpoint cocky_swartz (08124b75d2f031def6d36c6bc819549c009
   391e3bd76f3fe3b4e06e11be6fbad): Bind for 0.0.0.0:80 failed: port is
   already allocated.
   ```

3. What doe this command do?

   ```
   curl -sSL https://get.docker.com | sh
   ```

4. What are two ways to display the contents of a text file from the command line?
5. What are the two modes in vim?
6. How do you save the changes in a file in vim?
7. How would you find all files in your current directory that contain the string “bash”?
8. What are the steps to launching a Jupyter Notebook Server on a running AWS instance?

CHAPTER

THREE

MONGODB

```
In [1]: !pip install pymongo

Collecting pymongo
  Using cached https://files.pythonhosted.org/packages/30/f9/78dd244df932309299288a452d1c3524f6f7746f
Installing collected packages: pymongo
Successfully installed pymongo-3.6.1
You are using pip version 9.0.1, however version 10.0.1 is available.You should consider upgrading vi

In [2]: from pymongo import MongoClient

In [3]: mongo_client = MongoClient('18.236.138.158', 27016)
        database_reference = mongo_client.twitter
        collection_reference = database_reference.tweets

In [4]: cursor = collection_reference.find()

In [7]: next(cursor)

Out[7]: {'_id': ObjectId('5a4d56d9bfe2f9001587a130'),
         'created_at': 'Wed Jan 03 22:16:04 +0000 2018',
         'id': 948679379957223427,
         'id_str': '948679379957223427',
         'text': 'It's okay to invest in YOUrself. \nI beat myself up whenever i don't cross out ever
         'source': '<a href="http://instagram.com" rel="nofollow">Instagram</a>',
         'truncated': False,
         'in_reply_to_status_id': None,
         'in_reply_to_status_id_str': None,
         'in_reply_to_user_id': None,
         'in_reply_to_user_id_str': None,
         'in_reply_to_screen_name': None,
         'user': {'id': 61891630,
          'id_str': '61891630',
          'name': 'IG: Lorenamour64',
          'screen_name': 'Lorena64o',
          'location': 'NORTH HOLLYWOOD',
          'url': 'https://www.goherbalife.com/lramirez2/en-US',
          'description': 'LA I turned bad habits into good habits. Health Coach | karaoke queen | noc
          'translator_type': 'none',
          'protected': False,
          'verified': False,
          'followers_count': 337,
          'friends_count': 239,
          'listed_count': 10,
          'favourites_count': 574,
          'statuses_count': 3164,
          'created_at': 'Fri Jul 31 22:28:02 +0000 2009',
          'utc_offset': -28800,
```

```
          'time_zone': 'Pacific Time (US & Canada)',
          'geo_enabled': True,
          'lang': 'en',
          'contributors_enabled': False,
          'is_translator': False,
          'profile_background_color': '050108',
          'profile_background_image_url': 'http://pbs.twimg.com/profile_background_images/812409981/a
          'profile_background_image_url_https': 'https://pbs.twimg.com/profile_background_images/8124
          'profile_background_tile': False,
          'profile_link_color': 'FF0000',
          'profile_sidebar_border_color': 'FFFFFF',
          'profile_sidebar_fill_color': 'E3E2DE',
          'profile_text_color': '634047',
          'profile_use_background_image': True,
          'profile_image_url': 'http://pbs.twimg.com/profile_images/921581872206954496/QksJ_AK__norma
          'profile_image_url_https': 'https://pbs.twimg.com/profile_images/921581872206954496/QksJ_AK
          'profile_banner_url': 'https://pbs.twimg.com/profile_banners/61891630/1423650146',
          'default_profile': False,
          'default_profile_image': False,
          'following': None,
          'follow_request_sent': None,
          'notifications': None},
         'geo': {'type': 'Point', 'coordinates': [34.04491223, -118.44241261]},
         'coordinates': {'type': 'Point', 'coordinates': [-118.44241261, 34.04491223]},
         'place': {'id': '3b77caf94bfc81fe',
          'url': 'https://api.twitter.com/1.1/geo/id/3b77caf94bfc81fe.json',
          'place_type': 'city',
          'name': 'Los Angeles',
          'full_name': 'Los Angeles, CA',
          'country_code': 'US',
          'country': 'United States',
          'bounding_box': {'type': 'Polygon',
           'coordinates': [[[-118.668404, 33.704538],
             [-118.668404, 34.337041],
             [-118.155409, 34.337041],
             [-118.155409, 33.704538]]]},
          'attributes': {}},
         'contributors': None,
         'is_quote_status': False,
         'quote_count': 0,
         'reply_count': 0,
         'retweet_count': 0,
         'favorite_count': 0,
         'entities': {'hashtags': [],
          'urls': [{'url': 'https://t.co/9kQpNWp4F9',
            'expanded_url': 'https://www.instagram.com/p/BdgPSa5hQrc/',
            'display_url': 'instagram.com/p/BdgPSa5hQrc/',
            'indices': [96, 119]}],
          'user_mentions': [],
          'symbols': []},
         'favorited': False,
         'retweeted': False,
         'possibly_sensitive': False,
         'filter_level': 'low',
         'lang': 'en',
         'timestamp_ms': '1515017764449',
         'user_processed': True}
```

```
In [8]: cursor = collection_reference.find()
```

```
In [9]: cursor.count()

Out[9]: 2157100

In [10]: cursor_sampl = collection_reference.aggregate(
             [{'$sample': {'size': 20}}]
         )

In [12]: list(range(5))

Out[12]: [0, 1, 2, 3, 4]

In [13]: len(list(cursor_sampl))

Out[13]: 20

In [14]: next(cursor_sampl)

---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-14-96be4d43b38c> in <module>()
----> 1 next(cursor_sampl)

/opt/conda/lib/python3.6/site-packages/pymongo/command_cursor.py in next(self)
    290             return coll.database._fix_outgoing(self.__data.popleft(), coll)
    291         else:
--> 292             raise StopIteration
    293
    294     __next__ = next

StopIteration:

In [15]: cursor_sampl = collection_reference.aggregate([{'$sample': {'size': 1}}])

In [16]: tw = next(cursor_sampl)

In [17]: tw.keys()

Out[17]: dict_keys(['_id', 'created_at', 'id', 'id_str', 'text', 'display_text_range', 'source', 'tru

In [18]: list(tw.values())[:5]

Out[18]: [ObjectId('5a78aacc15ba4c00015664b7'),
          'Mon Feb 05 19:04:44 +0000 2018',
          960590028253446144,
          '960590028253446144',
          'YAAASSSSS!!!! LOVE MY TEAM AND MY CITY @Eagles!!! CONGRATS!! WELL DESERVED LONG AWAITED BU

In [19]: import pandas as pd

In [20]: cursor = collection_reference.aggregate([{'$sample': {'size': 5}}])

         tw_sample_df = pd.DataFrame(list(cursor))

In [21]: tw_sample_df.dtypes

Out[21]: _id                        object
         contributors               object
         coordinates                object
         created_at                 object
         display_text_range         object
         entities                   object
         extended_entities          object
         favorite_count              int64
         favorited                    bool
         filter_level               object
         geo                        object
```

```
id                            int64
id_str                       object
in_reply_to_screen_name      object
in_reply_to_status_id       float64
in_reply_to_status_id_str    object
in_reply_to_user_id         float64
in_reply_to_user_id_str      object
is_quote_status                bool
lang                         object
place                        object
possibly_sensitive           object
quote_count                   int64
reply_count                   int64
retweet_count                 int64
retweeted                      bool
source                       object
text                         object
timestamp_ms                 object
truncated                      bool
user                         object
user_processed               object
dtype: object
```

```
In [22]: tw_sample_df.select_dtypes([int])
```

```
Out[22]: favorite_count                  id  quote_count  reply_count  retweet_count
         0               0  949128180421423110            0            0              0
         1               0  952395937166446592            0            0              0
         2               0  950168161424359425            0            0              0
         3               0  955545267616473089            0            0              0
         4               0  958617919558098944            0            0              0
```

```
In [23]: tw_sample_df.select_dtypes([float])
```

```
Out[23]: in_reply_to_status_id  in_reply_to_user_id
         0                    NaN                  NaN
         1                    NaN                  NaN
         2                    NaN                  NaN
         3           9.555443e+17          583038984.0
         4                    NaN                  NaN
```

```
In [24]: tw_sample_df.select_dtypes(['object'])
```

```
Out[24]: _id contributors coordinates  \
         0  5a4ef81fbfe2f9001587ff5a         None        None
         1  5a5adb7436dd5f00015df15b         None        None
         2  5a52c0ae36dd5f000158f176         None        None
         3  5a66507f36dd5f000164b402         None        None
         4  5a717e2136dd5f00016b72de         None        None

                           created_at display_text_range  \
         0  Fri Jan 05 03:59:26 +0000 2018            [0, 38]
         1  Sun Jan 14 04:24:20 +0000 2018                NaN
         2  Mon Jan 08 00:51:57 +0000 2018            [0, 68]
         3  Mon Jan 22 20:58:39 +0000 2018           [35, 73]
         4  Wed Jan 31 08:28:16 +0000 2018                NaN

                                                   entities  \
         0  {'hashtags': [], 'urls': [], 'user_mentions': ...
         1  {'hashtags': [], 'urls': [], 'user_mentions': ...
         2  {'hashtags': [{'text': 'MyTwitterAnniversary',...
```

```
3  {'hashtags': [], 'urls': [], 'user_mentions': ...
4  {'hashtags': [{'text': 'Tweetlikethe1600s', 'i...

                                  extended_entities filter_level   geo  \
0  {'media': [{'id': 949128175698657281, 'id_str'...          low  None
1                                                NaN          low  None
2  {'media': [{'id': 950168155208347648, 'id_str'...          low  None
3                                                NaN          low  None
4                                                NaN          low  None

              id_str      ...       in_reply_to_status_id_str  \
0  949128180421423110      ...                            None
1  952395937166446592      ...                            None
2  950168161424359425      ...                            None
3  955545267616473089      ...             955544344269869056
4  958617919558098944      ...                            None

  in_reply_to_user_id_str lang  \
0                    None   en
1                    None   en
2                    None   en
3               583038984   en
4                    None   en

                                              place possibly_sensitive  \
0  {'id': '3b77caf94bfc81fe', 'url': 'https://api...              False
1  {'id': 'fbd6d2f5a4e4a15e', 'url': 'https://api...                NaN
2  {'id': '3b77caf94bfc81fe', 'url': 'https://api...              False
3  {'id': '3b77caf94bfc81fe', 'url': 'https://api...                NaN
4  {'id': '3b77caf94bfc81fe', 'url': 'https://api...                NaN

                                             source  \
0  <a href="http://twitter.com/download/iphone" r...
1  <a href="http://twitter.com/download/iphone" r...
2  <a href="http://twitter.com/download/iphone" r...
3  <a href="http://twitter.com/download/android" ...
4  <a href="http://twitter.com/download/iphone" r...

                                               text   timestamp_ms  \
0  This damn album was so overlooked . https:/...  1515124766818
1  Can't get over how fine p rod was in 2009 lowk...  1515903860738
2  Do you remember when you joined Twitter? I do!...  1515372717611
3  @johnpaulstonard @aiww @TateEtcMag No one has ...  1516654719629
4  Let me stop perusing #Tweetlikethe1600s before...  1517387296939

                                               user user_processed
0  {'id': 935285892, 'id_str': '935285892', 'name...            NaN
1  {'id': 2476155914, 'id_str': '2476155914', 'na...           True
2  {'id': 2281376473, 'id_str': '2281376473', 'na...           True
3  {'id': 823668383132487681, 'id_str': '82366838...            NaN
4  {'id': 26182204, 'id_str': '26182204', 'name':...            NaN

[5 rows x 21 columns]
```

```
In [25]: tw_sample_df['created_at'] = pd.to_datetime(tw_sample_df['created_at'])

In [28]: tw_sample_df.created_at

Out[28]: 0   2018-01-05 03:59:26
         1   2018-01-14 04:24:20
```

```
2   2018-01-08 00:51:57
3   2018-01-22 20:58:39
4   2018-01-31 08:28:16
Name: created_at, dtype: datetime64[ns]
```

3.1 Tweets Containing Geo Information

3.1.1 DO NOT RUN THIS

```
In [31]: contain_geo_search = collection_reference.find({'geo' : { '$ne' : None}})
         contain_geo_search
```

```
Out[31]: <pymongo.cursor.Cursor at 0x7f6b32f11f28>
```

```
In [30]: # contain_geo_search.count()
```

```
318373
```

3.2 Distinct Users

```
In [32]: collection_reference.distinct('user')
```

```
---------------------------------------------------------------------------
OperationFailure                          Traceback (most recent call last)
<ipython-input-32-1172ea4afe07> in <module>()
----> 1 collection_reference.distinct('user')

/opt/conda/lib/python3.6/site-packages/pymongo/collection.py in distinct(self, key, filter, session,
   2446             return self._command(sock_info, cmd, slave_ok,
   2447                                  read_concern=self.read_concern,
-> 2448                                  collation=collation, session=session)["values"]   2449
   2450     def map_reduce(self, map, reduce, out, full_response=False, session=None,

/opt/conda/lib/python3.6/site-packages/pymongo/collection.py in _command(self, sock_info, command, sl
    243                 session=s,
    244                 client=self.__database.client,
--> 245                 retryable_write=retryable_write)    246
    247     def __create(self, options, collation, session):

/opt/conda/lib/python3.6/site-packages/pymongo/pool.py in command(self, dbname, spec, slave_ok, read_
    515                             self.max_bson_size, read_concern,
    516                             parse_write_concern_error=parse_write_concern_error,
--> 517                             collation=collation)    518         except OperationFailure:
    519             raise

/opt/conda/lib/python3.6/site-packages/pymongo/network.py in command(sock, dbname, spec, slave_ok, is
    123             helpers._check_command_response(
    124                 response_doc, None, allowable_errors,
--> 125                 parse_write_concern_error=parse_write_concern_error)    126     except Except
    127         if publish:

/opt/conda/lib/python3.6/site-packages/pymongo/helpers.py in _check_command_response(response, msg, a
    143
    144             msg = msg or "%s"
--> 145             raise OperationFailure(msg % errmsg, code, response)
```

```
    146
    147
```

```
OperationFailure: distinct too big, 16mb cap
```

```
In [1]: # !pip install pymongo
```

```
In [2]: from pymongo import MongoClient
```

```
In [3]: matplotlib inline
```

```
In [4]: mongo_client = MongoClient('18.236.138.158', 27016)
        database_reference = mongo_client.twitter
```

```
In [5]: collection_reference = database_reference.instructor_test_group
```

```
In [6]: collection_reference.count()
```

```
Out[6]: 20000
```

```
In [7]: cursor_sampl = collection_reference.aggregate([{'$sample': {'size': 20}}])
```

```
In [8]: len(list(cursor_sampl))
```

```
Out[8]: 20
```

_id	truncated	user	extended_tweet	favorited
created_at	in_reply_to_status_id	geo	quote_count	retweeted
id	in_reply_to_status_id_str	coordinates	reply_count	filter_level
id_str	in_reply_to_user_id	place	retweet_count	lang
text	in_reply_to_user_id_str	contributors	favorite_count	timestamp_ms
source	in_reply_to_screen_name	is_quote_status	entities	

```
In [2]: cursor_sampl = collection_reference.aggregate([{'$sample': {'size': 20000}}])
```

```
In [3]: sample_group = list(cursor_sampl)
```

```
In [4]: database_reference.instructor_test_group.insert_many(sample_group)
Out[4]: <pymongo.results.InsertManyResult at 0x7f84d63324c8>
```

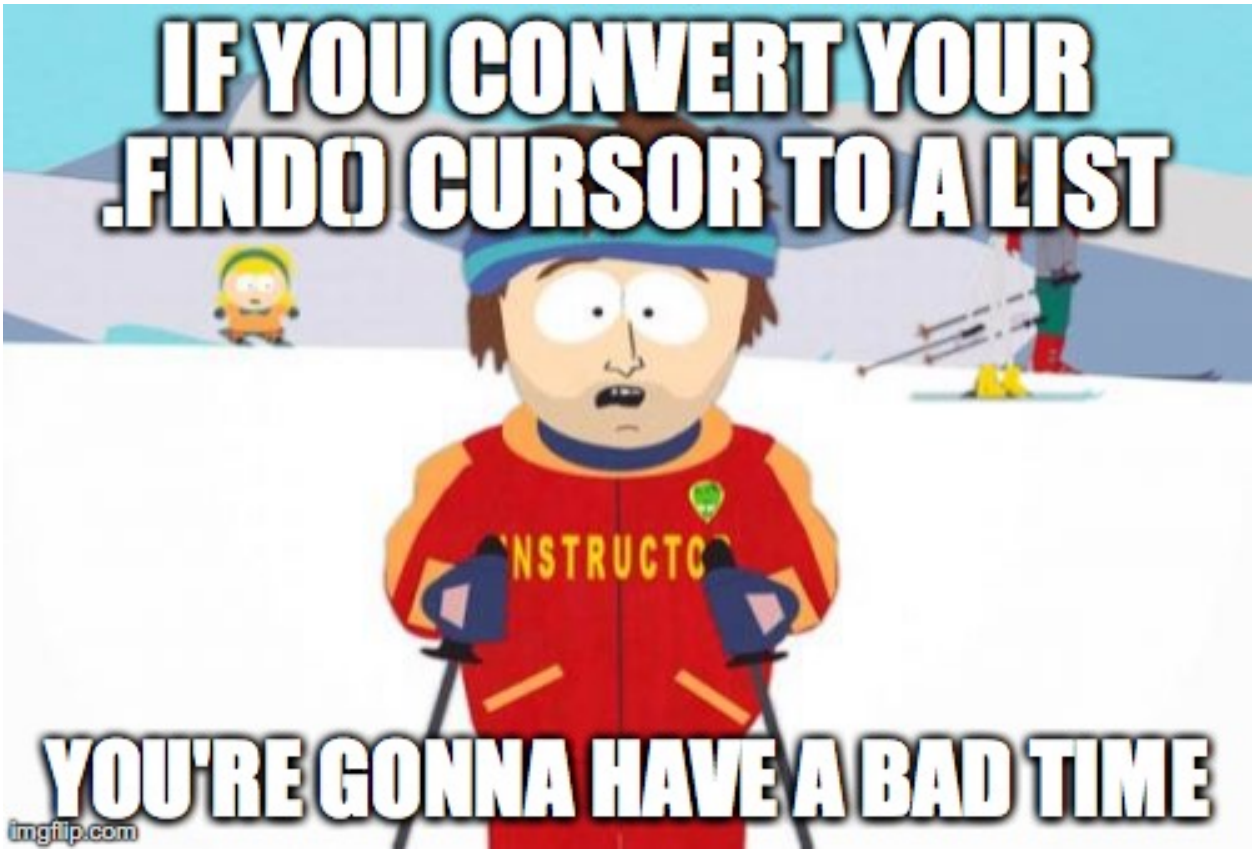
```
In [5]: database_reference.instructor_test_group.count()
Out[5]: 20000
```

3.3 The Aggregation Pipeline

A call to the aggregation framework defines a pipeline (figure 6.1), the **aggregation pipeline**, where the output from each step in the pipeline provides input to the next step. Each step executes a single operation on the input documents to transform the input and generate output documents.

3.3.1 Useful Aggregation Pipeline Operations

- `$project` // Specify fields to be placed in the output document.
- `$match` // Select documents to be processed, similar to find().
- `$limit` // Limit the number of documents to be passed to the next step.
- `$skip` // Skip a specified number of documents.


IF YOU CONVERT YOUR
.FINDO CURSOR TO A LIST
YOU'RE GONNA HAVE A BAD TIME
imgflip.com


Input documents
Operation 1
Operation 2
...
Operation n
Output documents

- `$unwind` // Expand an array, generating one output document for each array entry.
- `$group` // Group documents by a specified key.
- `$sort` // Sort documents.
- `$geoNear` // Select documents near a geospatial location.
- `$out` // Write the results of the pipeline to a collection (new in v2.6).
- `$redact` // Control access to certain data (new in v2.6).

```
In [9]: PROJECT = "$project"
        MATCH   = "$match"
        LIMIT   = "$limit"
        UNWIND  = "$unwind"
        GROUP   = "$group"
        SORT    = "$sort"
        COUNT   = "$count"

In [10]: test_group = database_reference.instructor_test_group

In [ ]: not_empty = { "$ne" : None }

In [17]: cursor = test_group.aggregate([
             { MATCH : { "geo" : not_empty }},
             { COUNT : "geo"}
         ])

In [18]: next(cursor)

Out[18]: {'geo': 2952}

In [19]: match_non_null_geo = { MATCH : { "geo" : not_empty }}
         count_geo = { COUNT : "geo"}

         dag_count_non_null_geo = [
             match_non_null_geo,
             count_geo
         ]

In [20]: next(test_group.aggregate(dag_count_non_null_geo))

Out[20]: {'geo': 2952}
```

3.3.2 Group Template

```
{ $group: { _id: <expression>, <field1>: { <accumulator1> : <expression1> }, ... } }
```

Accumulators

- `$sum`
- `$avg`
- `$first`
- `$last`
- `$max`
- `$min`
- `$stdDevPop`

- $stdDevSamp

```
In [33]: group_and_count("$text")
```

```
Out[33]: {'$group': {'_id': '$text', 'count': {'$sum': 1}}}
```

```
In [22]: greater_than_10 = { "$gt" : 10 }
         sum_1 = { "$sum" : 1 }

         def group_and_count(key):
             return { GROUP : {
                           "_id"   : key,
                           "count" : sum_1
                          }
                    }

         match_count_gt_10 = { MATCH : { "count" : greater_than_10 } }

         sort_by_count_descending = { SORT : { "count" : -1 } }

         def limit(val):
             return { LIMIT : val }
```

```
In [26]: list(test_group.aggregate(
             [
                 group_and_count('$lang'),
                 match_count_gt_10,
                 sort_by_count_descending,
                 limit(5)
             ]
         ))
```

```
Out[26]: [{'_id': 'en', 'count': 16996},
          {'_id': 'und', 'count': 1815},
          {'_id': 'es', 'count': 295},
          {'_id': 'tl', 'count': 126},
          {'_id': 'fr', 'count': 121}]
```

```
In [27]: not_an_empty_array             = { "$ne" : [] }
         match_non_empty_hashtag_arrays = { MATCH : { "entities.hashtags" : not_an_empty_array } }
         project_to_text_only           = { PROJECT : { "text" : "$entities.hashtags.text", "_id" :0
         unwind_text                    = { UNWIND : "$text" }
```

```
In [28]: list(test_group.aggregate(
             [
                 match_non_empty_hashtag_arrays,
                 project_to_text_only,
                 unwind_text,
                 limit(10)
             ]
         ))
```

```
Out[28]: [{'text': 'photos'},
          {'text': 'Artist'},
          {'text': 'LosAngeles'},
          {'text': 'Accounting'},
          {'text': 'Job'},
          {'text': 'Jobs'},
          {'text': 'Hiring'},
          {'text': 'CareerArc'},
          {'text': 'sanrio'},
```

```
          {'text': 'turquoise'}]
```

```
In [29]: list(test_group.aggregate(
             [
                 match_non_empty_hashtag_arrays,
                 project_to_text_only,
                 unwind_text,
                 group_and_count('$text'),
                 match_count_gt_10,
                 sort_by_count_descending,
                 limit(10)
             ]
         ))
```

```
Out[29]: [{'_id': 'job', 'count': 395},
          {'_id': 'Hiring', 'count': 308},
          {'_id': 'LosAngeles', 'count': 286},
          {'_id': 'CareerArc', 'count': 240},
          {'_id': 'hiring', 'count': 149},
          {'_id': 'Job', 'count': 107},
          {'_id': 'Jobs', 'count': 107},
          {'_id': 'earthquake', 'count': 67},
          {'_id': 'LA', 'count': 56},
          {'_id': 'losangeles', 'count': 49}]
```

```
In [34]: job_hashtags      = ['job', 'jobs', 'hiring', 'careerarc']
         location_hashtags = ['california', 'losangeles', 'la', 'santamonica', 'glendale', 'paloalto
         project_to_lower  = { PROJECT : { "text" : {"$toLower" : "$text"} } }
         match_not_in_bad  = { MATCH : { "_id" : { "$nin" : job_hashtags + location_hashtags}}}
```

```
In [35]: list(test_group.aggregate(
             [
                 match_non_empty_hashtag_arrays,
                 project_to_text_only,
                 unwind_text,
                 project_to_lower,
                 group_and_count('$text'),
                 match_not_in_bad,
                 match_count_gt_10,
                 sort_by_count_descending,
                 limit(50)
             ]
         ))
```

```
Out[35]: [{'_id': 'earthquake', 'count': 67},
          {'_id': 'goldenglobes', 'count': 56},
          {'_id': 'quake', 'count': 46},
          {'_id': 'art', 'count': 40},
          {'_id': 'healthcare', 'count': 38},
          {'_id': 'superbowl', 'count': 28},
          {'_id': 'retail', 'count': 26},
          {'_id': 'sales', 'count': 25},
          {'_id': 'rn', 'count': 25},
          {'_id': 'marketing', 'count': 25},
          {'_id': 'gonancygo', 'count': 24},
          {'_id': 'hospitality', 'count': 23},
          {'_id': 'grammys', 'count': 22},
          {'_id': 'repost', 'count': 22},
          {'_id': 'it', 'count': 21},
          {'_id': 'releasethememo', 'count': 21},
          {'_id': 'timesup', 'count': 19},
```

```
 {'_id': 'nsng', 'count': 19},
 {'_id': 'love', 'count': 19},
 {'_id': 'clerical', 'count': 18},
 {'_id': 'businessmgmt', 'count': 17},
 {'_id': 'tv', 'count': 17},
 {'_id': 'hollywood', 'count': 16},
 {'_id': 'trumpshutdown', 'count': 16},
 {'_id': 'script', 'count': 16},
 {'_id': 'beverlyhills', 'count': 15},
 {'_id': 'dtla', 'count': 15},
 {'_id': 'comedy', 'count': 15},
 {'_id': 'actorslife', 'count': 14},
 {'_id': 'nursing', 'count': 14},
 {'_id': 'iheartawards', 'count': 14},
 {'_id': 'trump', 'count': 14},
 {'_id': 'tbt', 'count': 13},
 {'_id': 'finance', 'count': 13},
 {'_id': 'fashion', 'count': 13},
 {'_id': 'photography', 'count': 13},
 {'_id': 'music', 'count': 12},
 {'_id': 'veterans', 'count': 11},
 {'_id': 'actor', 'count': 11},
 {'_id': 'pilot', 'count': 11},
 {'_id': 'actors', 'count': 11},
 {'_id': 'smile', 'count': 11},
 {'_id': 'lincoln', 'count': 11}]
```

```
In [1]: import pandas as pd
        from pymongo import MongoClient
        import random

        from mongo_aggregation_verbs import *

        mongo_client = MongoClient('18.236.138.158', 27016)
        database_reference = mongo_client.twitter
```

```
In [2]: database_reference.collection_names()

        collection_reference = database_reference.instructor_test_group

        test_group = database_reference.instructor_test_group
```

```
In [3]: match_empty_url_arrays = { MATCH : { "entities.urls" : [] } }

        list(test_group.aggregate(
            [
                match_empty_url_arrays,
                { COUNT : "text" }
            ]
        ))
```

```
Out[3]: [{'text': 11121}]
```

```
In [4]: job_hashtags = ['job', 'jobs', 'hiring', 'careerarc']
        location_hashtags = ['california', 'losangeles', 'la', 'santamonica', 'glendale', 'paloalto'
        match_not_in_bad = { MATCH : { "text" : { "$in" : job_hashtags + location_hashtags } } }
        project_to_text_keep_id = { PROJECT : { "text" : "$entities.hashtags.text" } }
        project_to_id = { PROJECT : { "_id" : 1 } }

        bad_ids = list(test_group.aggregate(
            [
```

```
                match_non_empty_hashtag_arrays,
                project_to_text_keep_id,
                unwind_text,
                project_to_lower,
                match_not_in_bad,
                project_to_id
            ]
        ))
        bad_ids[:10], len(bad_ids)
```

```
Out[4]: ([{'_id': ObjectId('5a73683636dd5f00016c7fad')},
          {'_id': ObjectId('5a73683636dd5f00016c7fad')},
          {'_id': ObjectId('5a73683636dd5f00016c7fad')},
          {'_id': ObjectId('5a73683636dd5f00016c7fad')},
          {'_id': ObjectId('5a73683636dd5f00016c7fad')},
          {'_id': ObjectId('5a6da1bd36dd5f0001690696')},
          {'_id': ObjectId('5a6da1bd36dd5f0001690696')},
          {'_id': ObjectId('5a6da1bd36dd5f0001690696')},
          {'_id': ObjectId('5a6df39136dd5f0001691533')},
          {'_id': ObjectId('5a6df39136dd5f0001691533')}],
         1835)
```

```
In [5]: bad_ids = [bad_id['_id'] for bad_id in bad_ids]
        bad_ids[:10]
```

```
Out[5]: [ObjectId('5a73683636dd5f00016c7fad'),
         ObjectId('5a73683636dd5f00016c7fad'),
         ObjectId('5a73683636dd5f00016c7fad'),
         ObjectId('5a73683636dd5f00016c7fad'),
         ObjectId('5a73683636dd5f00016c7fad'),
         ObjectId('5a6da1bd36dd5f0001690696'),
         ObjectId('5a6da1bd36dd5f0001690696'),
         ObjectId('5a6da1bd36dd5f0001690696'),
         ObjectId('5a6df39136dd5f0001691533'),
         ObjectId('5a6df39136dd5f0001691533')]
```

```
In [6]: not_in_bad_ids = { "$nin" : bad_ids }
```

```
In [8]: not_in_bad_ids_and_no_url = {
            "_id"           : not_in_bad_ids,
            "entities.urls" : []
        }

        just_the_text = {
            "text" : 1,
            "_id"  : 0
        }
```

```
In [9]: test_group.find_one(
            not_in_bad_ids_and_no_url,
            just_the_text
        )
```

```
Out[9]: {'text': "@SincerelyLegit Lol why at night? If it's boring I'll fall asleep"}
```

```
In [10]: cur  = test_group.find(
             not_in_bad_ids_and_no_url,
             just_the_text
         )

         tweets = list(cur)
         tweet_text = pd.DataFrame(tweets)
```

```
In [11]: len(tweet_text)
```

```
Out[11]: 11102
```

```
In [12]: tweet_text.head()
```

```
Out[12]:                                                text
         0  @SincerelyLegit Lol why at night? If it's bori...
         1                                  @godtributes Lol
         2  @KingBeyonceStan I need to binge on both now b...
         3  @_QUEENSharnay Naw I'd i can't put this skip o...
         4  @sannicolaso I have things to do at home but y...
```

```
In [13]: tweet_text.text = tweet_text.text.str.replace('http\S+|www.\S+', '', case=False)
```

```
In [ ]: from sklearn.datasets import fetch_20newsgroups
```

```
In [14]: from sklearn.feature_extraction.text import TfidfVectorizer
```

```
In [15]: tfidf = TfidfVectorizer(stop_words='english')
         tfidf.fit(tweet_text.text)
         word_occurence = tfidf.transform(tweet_text.text).todense()
```

```
In [16]: word_occurence.shape
```

```
Out[16]: (11102, 20582)
```

```
In [17]: words = tfidf.get_feature_names()
         word_sample = random.sample(words, 20)
         word_occurence_m = pd.DataFrame(word_occurence, columns=words)
         word_occurence_m[word_sample].head()
```

```
Out[17]: tires  awesomeeeee  odds  clout  100brbr  total  holy  bluejays  newyork  \
         0    0.0          0.0   0.0    0.0      0.0    0.0   0.0       0.0      0.0
         1    0.0          0.0   0.0    0.0      0.0    0.0   0.0       0.0      0.0
         2    0.0          0.0   0.0    0.0      0.0    0.0   0.0       0.0      0.0
         3    0.0          0.0   0.0    0.0      0.0    0.0   0.0       0.0      0.0
         4    0.0          0.0   0.0    0.0      0.0    0.0   0.0       0.0      0.0

            thatkiddez  picklerinn  fortunate  witnessed  rl_miller  bob  carmella  \
         0         0.0         0.0        0.0        0.0        0.0  0.0       0.0
         1         0.0         0.0        0.0        0.0        0.0  0.0       0.0
         2         0.0         0.0        0.0        0.0        0.0  0.0       0.0
         3         0.0         0.0        0.0        0.0        0.0  0.0       0.0
         4         0.0         0.0        0.0        0.0        0.0  0.0       0.0

            danrather  westla  betsy  brinamoniquee
         0        0.0     0.0    0.0            0.0
         1        0.0     0.0    0.0            0.0
         2        0.0     0.0    0.0            0.0
         3        0.0     0.0    0.0            0.0
         4        0.0     0.0    0.0            0.0
```

```
In [18]: from sklearn.decomposition import LatentDirichletAllocation
```

```
In [19]: lda = LatentDirichletAllocation(n_topics=10, learning_method='batch')
         lda.fit(word_occurence)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/decomposition/online_lda.py:294: DeprecationWarning: r
  DeprecationWarning)
```

```
Out[19]: LatentDirichletAllocation(batch_size=128, doc_topic_prior=None,
                      evaluate_every=-1, learning_decay=0.7,
                      learning_method='batch', learning_offset=10.0,
                      max_doc_update_iter=100, max_iter=10, mean_change_tol=0.001,
                      n_components=10, n_jobs=1, n_topics=10, perp_tol=0.1,
```

```
                    random_state=None, topic_word_prior=None,
                    total_samples=1000000.0, verbose=0)

In [20]: lda_df = pd.DataFrame(lda.components_, columns=words).T

In [21]: def filter_topic(lda_df, index, threshold):
             return (lda_df[lda_df[index] > threshold][index]
                     .sort_values(ascending=False))

In [22]: filter_topic(lda_df, 0, 10)

Out[22]: realdonaldtrump    26.924098
         life               14.433196
         today              13.796060
         like               12.909203
         tonight            12.440967
         hi                 11.932545
         lol                11.809768
         just               11.709171
         thank              10.415115
         lo                 10.157741
         Name: 0, dtype: float64

In [23]: filter_topic(lda_df, 1, 10)

Out[23]: good       17.543596
         god        16.114882
         morning    13.859482
         just       13.609985
         day        13.163371
         thank      10.265703
         Name: 1, dtype: float64

In [24]: filter_topic(lda_df, 2, 10)

Out[24]: don     17.529542
         mood    15.397384
         wow     13.017690
         time    12.512149
         gt      12.051757
         like    11.812582
         just    11.767592
         Name: 2, dtype: float64

In [30]: filter_topic(lda_df, 3, 5)

Out[30]: good     11.035873
         gonna     9.677585
         la        9.457188
         right     8.257726
         just      7.964738
         true      6.918964
         need      6.860816
         like      6.785791
         ass       6.449890
         heart     5.911852
         tired     5.898605
         music     5.106587
         omg       5.063022
         Name: 3, dtype: float64

In [26]: filter_topic(lda_df, 4, 10)
```

```
Out[26]: love     17.597504
         just     15.784500
         like     11.736119
         wait     11.312292
         today    10.499123
         Name: 4, dtype: float64

In [27]: filter_topic(lda_df, 5, 10)

Out[27]: fuck      19.533498
         just      19.283515
         yes       16.218575
         people    15.788082
         know      15.775503
         like      15.591380
         love      13.543071
         don       12.154129
         bitch     11.227335
         wanna     10.108777
         lol       10.100001
         Name: 5, dtype: float64

In [28]: filter_topic(lda_df, 6, 6)

Out[28]: love       24.931024
         good       11.150954
         angeles     9.569691
         sleep       9.182882
         omg         9.061345
         los         8.237702
         best        7.957185
         going       7.851733
         really      7.655026
         like        7.567627
         lol         7.551425
         couldn      7.519786
         care        7.317565
         just        7.006165
         fall        6.931113
         smh         6.906545
         today       6.660393
         time        6.586684
         getting     6.421132
         day         6.249168
         Name: 6, dtype: float64

In [29]: filter_topic(lda_df, 7, 7)

Out[29]: like      20.469175
         lmao      14.012189
         got       13.858762
         really    12.019327
         just      11.643017
         great     11.011661
         want      10.680874
         need       9.494232
         money      9.130007
         make       8.750701
         didn       8.128451
         lol        7.969240
         love       7.931216
```

```
crazy      7.606814
say        7.390890
know       7.127195
Name: 7, dtype: float64
```

```
In [ ]: filter_topic(lda_df, 8, 7)
```

```
In [ ]: filter_topic(lda_df, 9, 7)
```

```
In [1]: !pip install folium --quiet
```

```
In [2]: import pandas as pd
        from pymongo import MongoClient

        %matplotlib inline

        mongo_client = MongoClient('18.236.138.158', 27016)
        database_reference = mongo_client.twitter
```

_id	truncated	user	extended_tweet	favorited
created_at	in_reply_to_status_id	geo	quote_count	retweeted
id	in_reply_to_status_id_str	coordinates	reply_count	filter_level
id_str	in_reply_to_user_id	place	retweet_count	lang
text	in_reply_to_user_id_str	contributors	favorite_count	timestamp_ms
source	in_reply_to_screen_name	is_quote_status	entities	

```
In [2]: cursor_sampl = collection_reference.aggregate([{'$sample': {'size': 20000}}])

In [3]: sample_group = list(cursor_sampl)

In [4]: database_reference.instructor_test_group.insert_many(sample_group)
Out[4]: <pymongo.results.InsertManyResult at 0x7f84d63324c8>

In [5]: database_reference.instructor_test_group.count()
Out[5]: 20000
```

```
In [3]: collection_reference = database_reference.instructor_test_group
```

```
In [4]: collection_reference.count()
```

```
Out[4]: 20000
```

```
In [6]: from mongo_aggregation_verbs import *
```

3.4 Tweets By Day

```
In [7]: datestring_created_at = { "dateString" : "$created_at"}
        date_from_string = {"$dateFromString" : datestring_created_at }

        date_to_id = {
            PROJECT : {
                "_id" : 0,
                "year"  : {"$year"       : {"date" : date_from_string}},
                "month" : {"$month"      : {"date" : date_from_string}},
                "day"   : {"$dayOfMonth" : {"date" : date_from_string}},
            }
        }
```

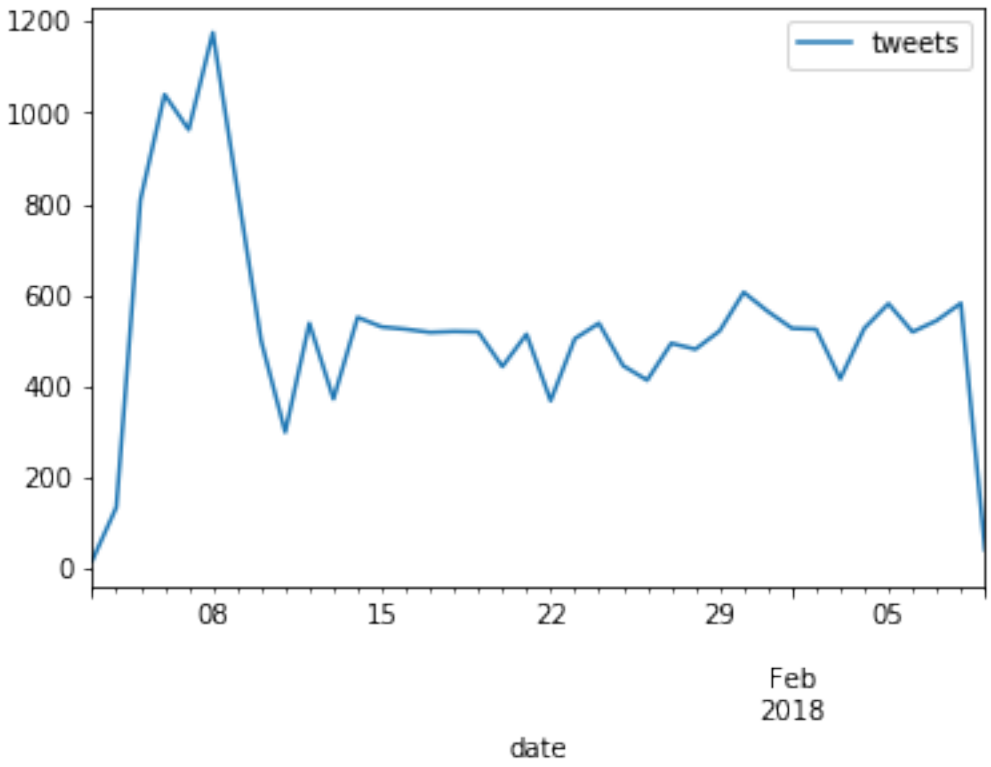
```
        group_by_date = {
            GROUP : {
                "tweets" : { "$sum" : 1 },
                "_id" : {
                    "year"  : "$year",
                    "month" : "$month",
                    "day"   : "$day"
                },
            }
        }
```

```
In [8]: def dictionary_to_datestring(x):
            month = x['month']
            day   = x['day']
            year  = x['year']
            return "{}-{}-{}".format(month, day, year)
```

```
In [9]: cursor = collection_reference.aggregate([
            date_to_id,
            group_by_date
        ])

        daily_tweets = pd.DataFrame(list(cursor))
```

```
In [10]: datestrings = daily_tweets['_id'].apply(dictionary_to_datestring)
         daily_tweets['date'] = pd.to_datetime(datestrings)

         daily_tweets.drop('_id', axis=1, inplace=True)
         daily_tweets.sort_values('date', inplace=True)
         daily_tweets.set_index('date', inplace=True)
         daily_tweets.plot()
```

```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7f33956f1f28>
```



3.5 Tweet Locations

```
In [11]: not_null = { '$ne' : None }
         nonnull_geo = {'geo' : not_null }
         keep_geo = { 'geo' : 1 }

         cursor = collection_reference.find(nonnull_geo, keep_geo)
         cursor.count()
```

```
Out[11]: 2952
```

```
In [12]: geo_tweets = pd.DataFrame(list(cursor))
```

```
In [23]: list(geo_tweets.head(5)['geo'].values)
```

```
Out[23]: [{'type': 'Point', 'coordinates': [34.18016221, -118.60321147]},
          {'type': 'Point', 'coordinates': [33.98752, -118.4538]},
          {'type': 'Point', 'coordinates': [34.1517492, -118.5214282]},
          {'type': 'Point', 'coordinates': [34.6495, -120.07927]},
          {'type': 'Point', 'coordinates': [33.9275, -118.27722]}]
```

```
In [21]: def parse_geo_from_tweets(tweets):
             geo = pd.DataFrame(list(tweets['geo'].values))
             return geo
```

```
In [22]: geo = parse_geo_from_tweets(geo_tweets)
         geo.sample(5)
```

```
Out[22]: coordinates   type
         1269          [32.6865, -117.1583]  Point
         1082  [34.13641722, -118.35336765]  Point
```

```
         1575            [36.758, -121.587]  Point
         1385         [34.26867, -118.1257]  Point
         731   [36.48863024, -119.72972051]  Point
```

```
In [24]: import folium
         starting_loc = [34.0689, -118.4452]
         la_map = folium.Map(location=starting_loc, zoom_start=13)
```

```
In [26]: geo.coordinates.values
```

```
Out[26]: array([[34.18016221, -118.60321147], [33.98752, -118.4538],
                [34.1517492, -118.5214282], ..., [33.96789, -118.32806],
                [33.74972064, -116.7024919], [34.313861, -118.534847]], dtype=object)
```

```
In [27]: for loc in geo.coordinates:
             folium.Marker(loc).add_to(la_map)
```

```
In [28]: la_map
```

```
Out[28]: <folium.folium.Map at 0x7f3394cdfd30>
```

```
In [1]: import pandas as pd
        import numpy as np
        from pymongo import MongoClient

        %matplotlib inline

        mongo_client = MongoClient('18.236.138.158', 27016)
        database_reference = mongo_client.twitter
```

```
In [2]: from mongo_aggregation_verbs import *
```

```
In [3]: collection_reference = database_reference.instructor_test_group
```

```
In [4]: collection_reference.count()
```

```
Out[4]: 20000
```

```
In [6]: INSTAGRAM = '<a href="http://instagram.com" rel="nofollow">Instagram</a>'
        source_is_instagram = { 'source' : INSTAGRAM }
        source_is_not_instagram = { 'source' : {'$ne' : INSTAGRAM } }
```

```
In [7]: (collection_reference.find(source_is_instagram).count(),
         collection_reference.find(source_is_not_instagram).count())
```

```
Out[7]: (1907, 18093)
```

3.6 Tweet Locations

```
In [8]: not_null = { '$ne' : None }
        nonnull_geo = {'geo' : not_null }
        keep_geo_and_text = { 'geo' : 1, 'text' : 1 , '_id' : 0}

        match_insta = {
            MATCH : source_is_instagram
        }

        match_not_insta = {
            MATCH : source_is_not_instagram
        }
```

```
In [9]: cursor = collection_reference.aggregate([
            { MATCH : source_is_instagram },
```

```
            { MATCH : nonnull_geo},
            { COUNT : "geo"}
        ])
        next(cursor)
```

```
Out[9]: {'geo': 1907}
```

```
In [10]: cursor = collection_reference.aggregate([
             { MATCH : source_is_not_instagram },
             { MATCH : nonnull_geo},
             { COUNT : "geo"}
         ])
         next(cursor)
```

```
Out[10]: {'geo': 1045}
```

```
In [11]: def group_and_count(key):
             return { GROUP : {
                         "_id"   : key,
                         "count" : { "$sum" : 1 }
                        }
                    }
```

```
In [12]: def parse_geo_from_tweets(tweets):
             tweets = pd.DataFrame(tweets)
             geo = pd.DataFrame(list(tweets['_id'].values))
             geo['count'] = tweets['count']
             return geo
```

```
In [14]: cursor = collection_reference.aggregate([
             { MATCH : source_is_not_instagram },
             { MATCH : nonnull_geo},
             group_and_count('$geo'),
             { MATCH : { "count" : { "$gt" : 14 } } },
             { SORT  : { "count" : -1 } }
         ])
         not_insta = parse_geo_from_tweets(list(cursor))
         not_insta
```

```
Out[14]:   coordinates   type  count
         0  [34.0522342, -118.2436849]  Point    206
         1  [37.3813444, -122.1802812]  Point     39
         2   [34.1425078, -118.255075]  Point     31
         3   [36.778261, -119.4179324]  Point     21
         4        [35.426667, -116.89]  Point     17
         5   [34.0508369, -118.263032]  Point     16
         6  [34.0194543, -118.4911912]  Point     15
```

```
In [15]: cursor = collection_reference.aggregate([
             { MATCH : source_is_instagram },
             { MATCH : nonnull_geo},
             group_and_count('$geo'),
             { MATCH : { "count" : { "$gt" : 14 } } },
             { SORT  : { "count" : -1 } }
         ])
         insta = parse_geo_from_tweets(list(cursor))
         insta
```

```
Out[15]:   coordinates   type  count
         0           [34.0522, -118.243]  Point    465
         1  [36.48863024, -119.72972051]  Point     37
         2  [34.09799334, -118.33866453]  Point     35
         3  [34.07305556, -118.39944444]  Point     29
```

```
4           [34.0221, -118.481]  Point     27
5    [34.0402214, -118.2545227]  Point     16
6    [33.9442368, -118.3975983]  Point     15
```

```
In [16]: import folium
         starting_loc = [34.0689, -118.4452]
         la_map = folium.Map(location=starting_loc, zoom_start=12)
```

```
In [17]: for loc, count in not_insta[['coordinates','count']].values:
             popup = folium.Popup(str(count), parse_html=True)
             folium.Marker(loc, popup=popup, icon=folium.Icon(color='red')).add_to(la_map)
         for loc, count in insta[['coordinates','count']].values:
             popup = folium.Popup(str(count), parse_html=True)
             folium.Marker(loc, popup=popup, icon=folium.Icon(color='blue')).add_to(la_map)
```

```
In [18]: la_map
```

```
Out[18]: <folium.folium.Map at 0x7ffa7de7e1d0>
```

```
In [19]: def parse_geo_from_tweets(tweets):
             tweets = pd.DataFrame(tweets)
             geo = pd.DataFrame(list(tweets['_id'].values))
             geo['count'] = tweets['count']
             return geo
```

```
In [20]: cursor = collection_reference.aggregate([
             { MATCH : source_is_not_instagram },
             { MATCH : nonnull_geo},
             group_and_count('$user.id'),
             { MATCH : { "count" : { "$gt" : 14 } } },
             { SORT  : { "count" : -1 } },
             { LIMIT : 10 }
         ])
         not_insta_top_users = pd.DataFrame(list(cursor))
         not_insta_top_users
```

```
Out[20]: _id  count
         0  4549072827     29
         1   787687147     29
         2  1414684496     27
         3  3066057658     27
         4   789990810     27
         5  4191239027     25
         6    21298660     21
         7  3864064936     19
         8    21298373     19
         9  3380828067     17
```

```
In [21]: cursor = collection_reference.aggregate([
             { MATCH : source_is_instagram },
             { MATCH : nonnull_geo},
             group_and_count('$user.id'),
         #     { MATCH : { "count" : { "$gt" : 10 } } },
             { SORT  : { "count" : -1 } },
             { LIMIT : 10 }
         ])
         insta_top_users = pd.DataFrame(list(cursor))
         insta_top_users
```

```
Out[21]: _id  count
         0          1455659006     10
```

```
1           613833206      8
2  843390093012353024      6
3          4561143733      6
4            19640448      5
5           226456467      5
6            37016954      4
7  760160463833313280      4
8            30723561      4
9          2267807461      4
```

```
In [22]: not_insta_top_users_ids = not_insta_top_users._id.values
         insta_top_users_ids = insta_top_users._id.values
```

```
In [23]: not_insta_top_users_ids_list = list(not_insta_top_users_ids)
         not_insta_top_users_ids_list = [int(i) for i in not_insta_top_users_ids_list]
         insta_top_users_ids_list = list(insta_top_users_ids)
         insta_top_users_ids_list = [int(i) for i in insta_top_users_ids_list]
```

```
In [24]: def parse_geo_from_tweets(tweets):
             tweets = pd.DataFrame(tweets)
             tmp = pd.DataFrame(list(tweets['_id'].values))
             geo = pd.DataFrame(list(tmp['geo'].values))
             geo['user_id'] = tmp['user_id']
             geo['count'] = tweets['count']
             return geo
```

```
In [25]: cursor = collection_reference.aggregate([
             { MATCH : source_is_not_instagram },
             { MATCH : nonnull_geo},
             { PROJECT : { "user_id" : "$user.id", "geo" : 1, "text" : 1, "_id" :0 } },
             { MATCH : { "user_id" : { "$in" : not_insta_top_users_ids_list }}},
             group_and_count({"user_id" : "$user_id", "geo" : "$geo"}),
         ])

         not_insta_top_user_geo = parse_geo_from_tweets(list(cursor))
```

```
In [26]: cursor = collection_reference.aggregate([
             { MATCH : source_is_instagram },
             { MATCH : nonnull_geo},
             { PROJECT : { "user_id" : "$user.id", "geo" : 1, "text" : 1, "_id" :0 } },
             { MATCH : { "user_id" : { "$in" : insta_top_users_ids_list }}},
             group_and_count({"user_id" : "$user_id", "geo" : "$geo"}),
         ])
         insta_top_user_geo = parse_geo_from_tweets(list(cursor))
```

```
In [27]: not_insta_top_user_geo.head()
```

```
Out[27]: coordinates   type     user_id  count
         0  [34.19743613, -118.58178967]  Point  4549072827      1
         1        [34.03491, -118.27746]  Point  4191239027      1
         2     [35.7476654, -118.060997]  Point  1414684496      1
         3         [34.0995, -118.32813]  Point  4191239027      1
         4     [34.187044, -118.3812562]  Point   789990810      1
```

```
In [28]: insta_top_user_geo.head()
```

```
Out[28]: coordinates   type     user_id  count
         0  [34.04453451, -118.26677639]  Point   226456467      1
         1  [34.06895637, -118.40267947]  Point  1455659006      1
         2           [34.0221, -118.481]  Point  1455659006      1
         3  [34.07305556, -118.39944444]  Point  1455659006      5
         4  [34.08718311, -118.46354276]  Point    19640448      1
```

```
In [55]: colors_insta = {
             760160463833313280 : 'red',
             30723561 : 'blue',
             613833206 : 'green',
             2267807461 : 'purple',
             4561143733 : 'orange',
             1455659006 : 'darkred',
             37016954 : 'lightred',
             19640448 : 'beige',
             843390093012353024 : 'darkblue',
             226456467 : 'darkgreen',
         }

         # colors_insta = {
         #     760160463833313280 : '#0000ff',
         #     30723561 : '#0010ff',
         #     613833206 : '#0020ff',
         #     2267807461 : '#0030ff',
         #     4561143733 : '#0040ff',
         #     1455659006 : '#0050ff',
         #     37016954 : '#0060ff',
         #     19640448 : '#0070ff',
         #     843390093012353024 : '#0080ff',
         #     226456467 : '#0090ff',
         # }

In [56]: # not_insta_top_user_geo['color'] = not_insta_top_user_geo.user_id.apply(lambda x: colors_n
         insta_top_user_geo['color'] = insta_top_user_geo.user_id.apply(lambda x: colors_insta[x])

In [57]: insta_top_user_geo.sample(10)

Out[57]:
```

	coordinates	type	user_id	count	color
14	[34.0304, -118.779]	Point	2267807461	1	purple
2	[34.0221, -118.481]	Point	1455659006	1	darkred
5	[34.0981334, -118.32668656]	Point	4561143733	2	orange
12	[34.0522, -118.243]	Point	2267807461	2	purple
19	[34.06635491, -118.41345382]	Point	19640448	1	beige
7	[34.122322, -118.223444]	Point	760160463833313280	4	red
21	[33.7358, -118.291]	Point	226456467	4	darkgreen
11	[34.07517256, -118.35229982]	Point	2267807461	1	purple
15	[34.0567207, -118.4424515]	Point	19640448	1	beige
9	[34.0522, -118.243]	Point	30723561	4	blue

```
In [58]: starting_loc = [34.0689, -118.4452]
         la_map = folium.Map(location=starting_loc, zoom_start=12)

In [59]: for loc, color, count in insta_top_user_geo[['coordinates','color','count']].values:
             popup = folium.Popup(str(count), parse_html=True)
             if count < 3:
                 folium.Marker(loc, popup=popup, icon=folium.Icon(color=color)).add_to(la_map)
         #     else:
         #         folium.Marker(loc, popup=popup, icon=folium.Icon(color=color, icon='warning')).ad
         # for loc, count in insta[['coordinates','count']].values:
         #     folium.Marker(loc, popup=popup, icon=folium.Icon(color='blue')).add_to(la_map)



In [60]: la_map

Out[60]: <folium.folium.Map at 0x7ffa7654c1d0>
```

```
760160463833313280 : 'red',
30723561 : 'blue',
613833206 : 'green',
2267807461 : 'purple',
4561143733 : 'orange',
1455659006 : 'darkred',
37016954 : 'lightred',
19640448 : 'beige',
843390093012353024 : 'darkblue',
226456467 : 'darkgreen',
```

```
In [65]: cur = collection_reference.find({"user.id" : 37016954})
         tw = list(cur)
         pd.DataFrame(tw)[['text', 'user']]
```

```
Out[65]: text  \
         0  Thanks for the support • Be A Leader • Shop &a...
         1  They'll quit on themselves to find excuses to ...
         2  Thanks for the support • The latest singles fr...
         3  Thanks for the support • The latest singles fr...

                                                         user
         0  {'id': 37016954, 'id_str': '37016954', 'name':...
         1  {'id': 37016954, 'id_str': '37016954', 'name':...
         2  {'id': 37016954, 'id_str': '37016954', 'name':...
         3  {'id': 37016954, 'id_str': '37016954', 'name':...
```

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search