


Introduction to Data Science

Welcome to [Zipfian Academy's \(http://zipfianacademy.com\)](http://zipfianacademy.com) Introduction to Data Science course. Thank you for attending, we hope you enjoyed the lecture (we sure had fun presenting). This exercise will give you hands-on experience with the concepts covered, and will help solidify your understanding of the process of data science.

Getting Help

If you have any questions throughout this exercise or need help at all, please ask on [Piazza \(https://piazza.com/class#summer2013/101\)](https://piazza.com/class#summer2013/101) and join the conversation. Chances are that if you have a question or get stuck, someone else is in the same situation. If you post to Piazza we (the instructors) can answer the question for everyone, and other students can provide insights as well.

 As always, feel free to email us about anything at all (questions, issues, concerns, feedback) at [class@zipfianacademy.com \(mailto:class@zipfianacademy.com\)](mailto:class@zipfianacademy.com). We would love to hear how you liked the class, whether the content was technical enough (or too technical), or any other topics you wish were covered.

Next Steps

We hope you have fun with this exercise! If you want to learn more or dive deeper into any of these subjects, we are always happy to discuss (and can talk for days about these subjects). We will be continuing the data journey with the subsequent classes in the [series \(https://team-zipfian-statistics.eventbrite.com/\)](https://team-zipfian-statistics.eventbrite.com/). We encourage you to sign up for any class(es) covering topics which you would like to dive deeper into and continue your learning. And if you just can't get enough of this stuff (and want a completely immersive environment), you can [apply \(http://zipfiancollective.wufoo.com/forms/z7x3p3/\)](http://zipfiancollective.wufoo.com/forms/z7x3p3/) for our intensive data science bootcamp starting September 16th.

Learning Python

This assignment assumes a basic familiarity with Python and is intended to teach you how to leverage it for data science. If you do not feel comfortable enough with Python (and programming in general) I recommend these resources:

- [Think Python \(http://www.greenteapress.com/thinkpython/thinkpython.pdf\)](http://www.greenteapress.com/thinkpython/thinkpython.pdf)
- [MIT Open Courseware: A Gentle Introduction to Programming Using Python \(http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-189-a-gentle-introduction-to-programming-using-python-january-iap-2008/index.htm\)](http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-189-a-gentle-introduction-to-programming-using-python-january-iap-2008/index.htm)
- [Learn Python the Hard Way \(http://learnpythonthehardway.org/book/\)](http://learnpythonthehardway.org/book/)
- [Python Koans \(https://github.com/gregmalcolm/python_koans/wiki\)](https://github.com/gregmalcolm/python_koans/wiki)

Setup and Environment

This exercise is written in an [IPython \(http://ipython.org/\)](http://ipython.org/) [notebook \(http://ipython.org/notebook.html\)](http://ipython.org/notebook.html) and uses many of wonderful libraries from the scientific Python [community \(http://strata.oreilly.com/2013/03/python-data-tools-just-keep-getting-better.html\)](http://strata.oreilly.com/2013/03/python-data-tools-just-keep-getting-better.html). While you do not need IPython locally to complete the exercise (there are PDF and .ipynb versions of these instructions), I recommend setting it up on your computer if you plan to continue learning and playing with data. IPython [notebooks \(http://ipython.org/ipython-doc/stable/interactive/htmlnotebook.html\)](http://ipython.org/ipython-doc/stable/interactive/htmlnotebook.html) not only provide an interface to interactively run (and debug) code in a web browser, but also to document your file as you go along. Below are the steps to setup a scientific Python environment on your computer to complete this (and all future class's) exercise. If you have tips or suggestions to make this process easier, please reach out either on Piazza or via email.

Version control and Environment Isolation

- [Git \(http://git-scm.com/\)](http://git-scm.com/): Distributed Version Control to keep track of changes and updates to files/data.
- [virtualenv \(http://www.virtualenv.org/en/latest/\)](http://www.virtualenv.org/en/latest/): Python environment isolation to help manage dependencies with packages and versions.
- [pythonbrew \(https://github.com/utahta/pythonbrew\)](https://github.com/utahta/pythonbrew): Manage and install multiple versions of Python. Can be handy if you want to experiment with Python 3.x.

Scientific Python packages

- [Enthought Python Distribution \(https://www.enthought.com/products/epd/free/\)](https://www.enthought.com/products/epd/free/): A freely available packaged environment for scientific Python.
- [Scipy Superpack \(http://fonnesbeck.github.io/ScipySuperpack/\)](http://fonnesbeck.github.io/ScipySuperpack/): Only for Mac OSX, but a one line shell script that installs all the fundamental scientific computing packages.
- [pandas \(http://pandas.pydata.org/\)](http://pandas.pydata.org/): Data analysis and statistical library providing functionality in Python similar to R (<http://www.r-project.org/>).

if you are on OSX, you may need to install Xcode (http://developer.apple.com/library/ios/#documentation/DeveloperTools/Conceptual/WhatsNewXcode/Articles/xcode_4_3.html) (with command line utilities) or install gcc (<https://medium.com/kr-projects/6e54e8c50dc8>) directly

[Tutorial \(https://sites.google.com/site/pythonbootcamp/preparation/software\)](https://sites.google.com/site/pythonbootcamp/preparation/software) walking you through the installation of these tools, with **[tests \(https://sites.google.com/site/pythonbootcamp/preparation/testing-that-it-all-works\)](https://sites.google.com/site/pythonbootcamp/preparation/testing-that-it-all-works)** to make sure it all works.

Exercise: Happy Healthy Hungry -- San Francisco

Now that your environment is all setup up... the fun begins! In this exercise we will walk through the data science [process \(http://zipfianacademy.com/data/data-science-process.png\)](http://zipfianacademy.com/data/data-science-process.png) covered in lecture. We will be analyzing the inspections of San Francisco restaurants using publicly available [data \(http://www.sfdph.org/dph/EH/Food/score/default.asp\)](http://www.sfdph.org/dph/EH/Food/score/default.asp) from the Department of Public health. We will learn to explore this data to map the cleanliness of the city, and get a better

perspective on the relative meaning of these scores by looking at summary statistics of the data. We will also use simple regression in addition to more advanced Machine Learning methods to attempt to predict the score of restaurants that have pending inspections. Once we understand how SF fares, we will compare its restaurants to those of [NYC](https://nyc.opendata.socrata.com/Health/Restaurant-Inspection-Results/4ykw-7nck) (<https://nyc.opendata.socrata.com/Health/Restaurant-Inspection-Results/4ykw-7nck>).

This case study will be explored through the class [series](http://team-zipfian-statistics.eventbrite.com/) (<http://team-zipfian-statistics.eventbrite.com/>). Each subsequent class will go into much more depth of each specific topic. This specific exercise is an introduction to the problem, and focuses on the data science [process](http://zipfianacademy.com/data/data-science-workflow/animate.gif) (<http://zipfianacademy.com/data/data-science-workflow/animate.gif>) itself more than individual subjects.

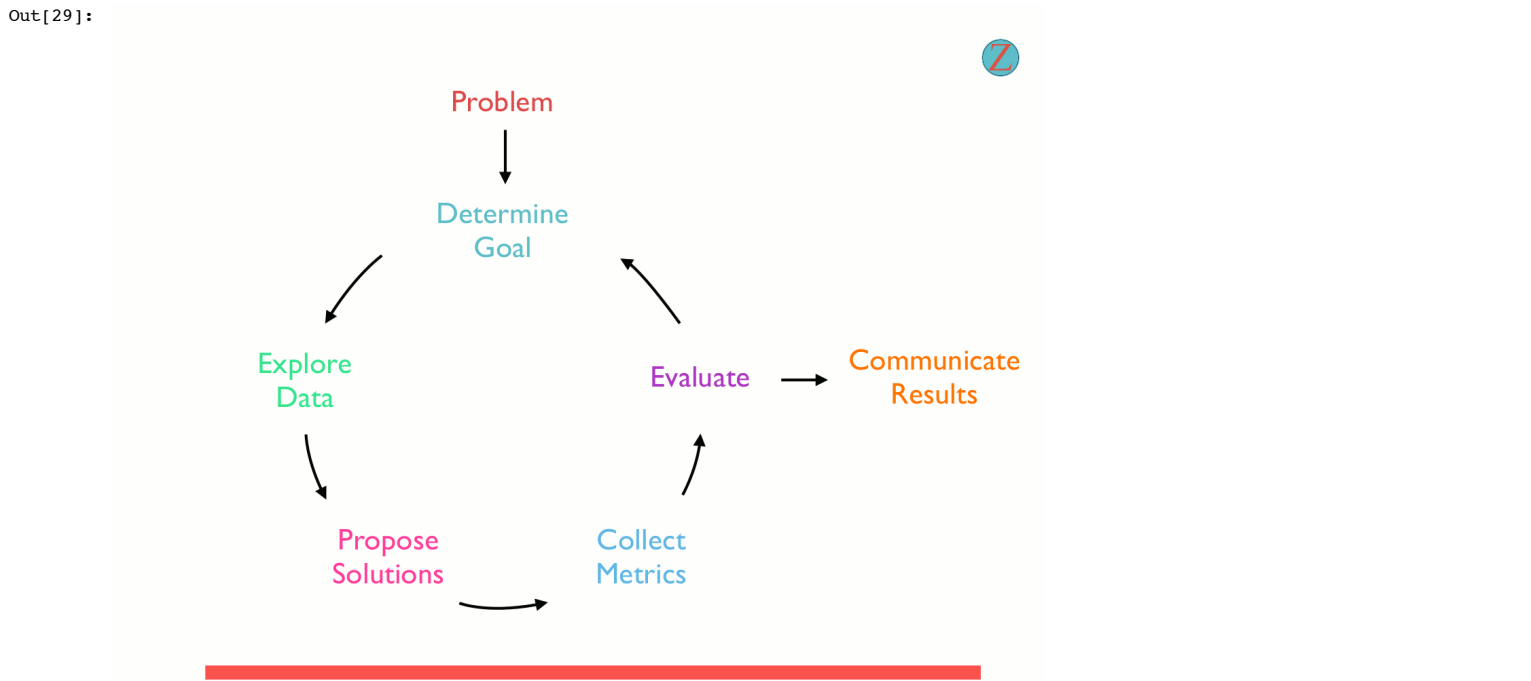
```
In [28]: # Import pylab to provide scientific Python libraries (NumPy, SciPy, Matplotlib)
import pylab

# import the Image display module
from IPython.display import Image

# inline allows us to embed matplotlib figures directly into the IPython notebook
%pylab inline
```

Welcome to pylab, a matplotlib-based Python environment [backend: module://IPython.kernel.zmq.pylab.backend_inline].
For more information, type 'help(pylab)'.

```
In [29]: Image(url='http://zipfianacademy.com/data/data-science-workflow/animate.gif', width=700)
```



Problem

The first step of the **Process** is to define the problem we want to address. To do so let us review what we have set out to accomplish and begin exploring questions we want answered.

How clean are SF restaurants?

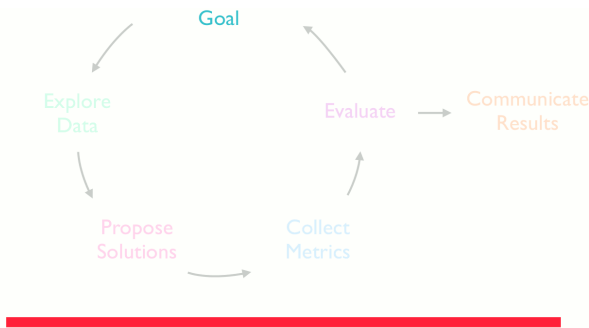
It is often best to arrive at a simple yet illuminating question to give you direction. Of course there are a number of sub-questions we may have that relate to our over arching problem, we can address these when we determine our goals for the analysis.

Let us review the important points to keep in mind when defining our problem:

- The question can be **qualitative**, but the approach, must be **quantifiable**
- What am I **looking** for?
- What do I want to **learn**?
- Alright to be **exploratory** (and the best analysis often are)

```
In [30]: Image(url='http://zipfianacademy.com/data/data-science-workflow/goal.png', width=500)
```





Determine Goal

Now that we have a problem we hope to solve, let us begin to quantify our analysis. Since our *Problem Statement* is often qualitative and broad, we can ask further questions to better define what we hope to achieve.

How does an individual restaurants' score compare to the whole/aggregate of SF?

Are SF's inspections better or worse than other cities?

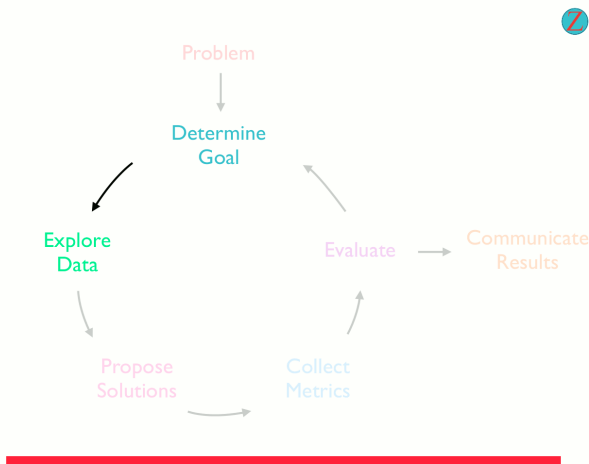
If a restaurant has not yet been inspected, can we approximate/predict what score it will receive?

Some things to note about our goals and approach:

- Determine what defines **success**, and to what degree.
- Brainstorm **metrics** to visualize and/or calculate.
- Ask **questions** that have (or can have) a definitive answer.
- Be careful what you wish for, be aware of possible **correlations**, and take caution with how you measure (http://en.wikipedia.org/wiki/Observer-expectancy_effect) it.

In [31]: `Image(url='http://zipfianacademy.com/data/data-science-workflow/explore.png', width=500)`

Out[31]:



Explore Data

To recap where we are in our analysis:

- We have determined what we want to learn -- **How clean are SF restaurants?**
- We explored quantifiable metrics to collect -- **Individual scores, summary statistics about distribution of scores, other cities' inspection data to compare**

The **Explore** stage of the analysis is where we will most likely spend most of our time (<http://strataconf.com/stratany2012/public/schedule/detail/27495>). Now comes the fun part (in my opinion)! At this stage we will use a variety of tools (the documentation of each linked to inline) to figure out where and how to obtain data, what it looks like once we have it, and how to use it to answer of questions to achieve our goals.

Acquire

Luckily, San Francisco has much of its public government data freely accessible (<https://data.sfgov.org/>) online. There are also great initiatives (<http://www.yelp.com/healthscores>) by SF companies (<http://officialblog.yelp.com/2013/01/introducing-lives.html>) collaborating with non-profits (<http://codeforamerica.org/>) and government (<http://sfgov.org/>) to develop open data standards (<http://foodinspectiondata.us/>). Such standardization allows for much more transparency, leading ultimately to a more engaged citizenry.

The relevant data (<http://www.sfdph.org/dph/EH/Food/score/default.asp>) has been downloaded for your convenience and can be found on Piazza in the Resources section (https://piazza.com/zipfian_academy/summer2013/101/resources).

Examine

If you are working with this IPython notebook, download the data files into the same directory which you ran the `ipython notebook` command

Now that we have found the relevant data we can begin to peer inside to understand what we are working with. I recommend starting with an iterative approach, using the quickest/easiest tools first and slowly build to more complicated analyses. UNIX provides us with many powerful tools and can carry us quite far by itself. In our case the dataset came with [documentation \(https://s3.amazonaws.com/piazza-resources/hgtp0qhps1d5/hhfilqv3gii2l8/File_Specifications.pdf?AWSAccessKeyId=AKIAJKOQYKAYOBKKVTKQ&Expires=1370258028&Signature=ZsHGKBMNwbdv9Ptio3b6GYrhB08%3D\)](https://s3.amazonaws.com/piazza-resources/hgtp0qhps1d5/hhfilqv3gii2l8/File_Specifications.pdf?AWSAccessKeyId=AKIAJKOQYKAYOBKKVTKQ&Expires=1370258028&Signature=ZsHGKBMNwbdv9Ptio3b6GYrhB08%3D) of its contents, but it still is essential to look at the raw data and compare it to the docs.

```
In [32]: # Let us display a few lines of data to understand its format and fields

# Any command prefixed with '!' instructs IPython to run a shell command
# http://ipython.org/ipython-doc/rel-0.13.1/interactive/reference.html#system-shell-access

!head -n 5 SFBusinesses/businesses.csv

"business_id","name","address","city","state","postal_code","latitude","longitude","phone_number"
10,"TIRAMISU KITCHEN","033 BELDEN PL","San Francisco","CA","94104","37.791116","-122.403816",""
12,"KIKKA","250 EMBARCADERO 7/F","San Francisco","CA","94105","37.788613","-122.393894",""
17,"GEORGE'S COFFEE SHOP","2200 OAKDALE AVE ","San Francisco","CA","94124","37.741086","-122.401737","+14155531470"
19,"NRGIZE LIFESTYLE CAFE","1200 VAN NESS AVE, 3RD FLOOR","San Francisco","CA","94109","37.786848","-122.421547",""
```

`head` is a UNIX command to print only the first few lines of a file (in this case 5). This is very useful for exploring very large files (which happens a lot in data science) very quickly and easily. You can read more about it [here \(http://en.wikipedia.org/wiki/Head_%28Unix%29\)](http://en.wikipedia.org/wiki/Head_%28Unix%29) or by consulting its manual pages on the command-line: `man head`

```
In [33]: # [PROTIP]: IPython has built in tab completion for commands.
# Partially type a command or file name and hit tab.

!head -n 5 SFBusinesses/inspections.csv

"business_id","Score","date","type"
10,"98","20121114","routine"
10,"98","20120403","routine"
10,"100","20110928","routine"
10,"96","20110428","routine"
```

```
In [34]: !head -n 5 data/SFBusinesses/violations.csv

head: data/SFBusinesses/violations.csv: No such file or directory
```

```
In [35]: # It is a little hard to read since the headers are much
# shorter than the data. Lets see if we can prettify it.

!head -n 5 SFBusinesses/violations.csv | column -t -s ','

"business_id"  "date"          "description"
10             "20121114"      "Unclean or degraded floors walls or ceilings [ date violation corrected:  ]"
10             "20120403"      "Unclean or degraded floors walls or ceilings [ date violation corrected: 9/20/2012 ]"
10             "20110428"      "Inadequate and inaccessible handwashing facilities [ date violation corrected: 6/1/2011 ]"
12             "20120420"      "Food safety certificate or food handler card not available [ date violation corrected: 11/20/2012 ]"
```

`column` is used to format its input into multiple columns. It also is useful for [formatting \(http://linux.about.com/library/cmd/blcmdl1_column.htm\)](http://linux.about.com/library/cmd/blcmdl1_column.htm) columns already present in delimited data (CSV for example). With this command we used UNIX [pipes \(http://en.wikipedia.org/wiki/Pipeline_%28Unix%29\)](http://en.wikipedia.org/wiki/Pipeline_%28Unix%29), one of the most powerful and useful aspects of working in a terminal.

```
In [36]: !head -n 5 SFBusinesses/ScoreLegend.csv | column -t -s ','

"Minimum_Score"  "Maximum_Score"  "Description"
0                70               "Poor"
71               85               "Needs Improvement"
86               90               "Adequate"
91               100              "Good"
```

There are two different data directories, each of which has similar files. Let's try to figure out the difference between the two, since the documentation on the data does not mention anything.

```
In [37]: %%bash

# We can use IPython cell 'magics' to run a cell in a subprocess. In this case we run
# the entire cell in a bash process (notice no exclamation point before the shell command)

head -n 5 SFFoodProgram_Complete_Data/businesses_plus.csv

"business_id","name","address","city","state","postal_code","latitude","longitude","phone_no","TaxCode","business_certificate","applic

10,"TIRAMISU KITCHEN","033 BELDEN PL","San Francisco","CA","94104","37.791116","-122.403816","H24",,"Tiramisu LLC","33 Belden
St","San Francisco","CA","94104"
12,"KIKKA","250 EMBARCADERO 7/F","San Francisco","CA","94105","37.788613","-122.393894","H24",,7/12/2002 0:00:00,"KIKKA ITO,
INC.",,431 South Isis Ave.,,Inglewood","CA","90301"
17,"GEORGE'S COFFEE SHOP","2200 OAKDALE AVE ","San Francisco","CA","94124","37.741086","-122.401737","(141) 555-5314","H24",,4/5/1975
0:00:00,"LIEUW, VICTOR & CHRISTINA C","648 MACARTHUR DRIVE","DALY CITY","CA","94015"
```

```
19,"NRGIZE LIFESTYLE CAFE","1200 VAN NESS AVE, 3RD FLOOR","San Francisco","CA","94109","37.786848","-122.421547",,"H24",,"24 Hour Fitness Inc","1200 Van Ness Ave, 3rd Floor","San Francisco","CA","94109"
```

IPython cell [magics \(http://nbviewer.ipython.org/url/github.com/ipynbviewer/ipynbviewer/raw/master/examples/notebooks/Cell%20Magics.ipynb\)](http://nbviewer.ipython.org/url/github.com/ipynbviewer/ipynbviewer/raw/master/examples/notebooks/Cell%20Magics.ipynb) and other tricks

```
In [38]: # Can we somehow compare the columns?

!head -n 1 SFFoodProgram_Complete_Data/businesses_plus.csv | awk -F, '{ print NF }'
!head -n 1 SFBusinesses/businesses.csv | awk -F, '{ print NF }'

17
9
```

We see here that in SFFoodProgram_Complete_Data/ the files seem to be augmented with additional data. The file has almost twice as many fields present.

We used [awk \(http://en.wikipedia.org/wiki/AWK\)](http://en.wikipedia.org/wiki/AWK) to figure this out by passing in the header row from the file to a simple awk script to count the number of fields (NF).

AWK is actually a programming language (in addition to a command line utility) and can be quite powerful if used correctly. This is going to be one of the standard tools at our disposal as a data scientist to explore and manipulate data.

```
In [39]: # In addition to the extra fields, is anything else different?

!wc SFBusinesses/businesses.csv SFFoodProgram_Complete_Data/businesses_plus.csv

 6384   44426   656875 SFBusinesses/businesses.csv
 6353   85003  1215896 SFFoodProgram_Complete_Data/businesses_plus.csv
12737  129429 1872771 total
```

[wc \(http://en.wikipedia.org/wiki/Wc_%28Unix%29\)](http://en.wikipedia.org/wiki/Wc_%28Unix%29) is another standard UNIX utility that we will find ourselves coming back to time and time again. Here we compare the line counts (number of records) and sizes of the files.

The first column is the number of newlines, or how many records are contained. The second column is word count and the last the number of bytes

Some interesting things to note from this very simple (yet illustrative) exploration of our data. As we might have guessed, the files in the SFFoodProgram_Complete_Data/ have more information added to the original SFBusinesses/ files. While the 'complete' data has more columns, there are actually fewer records (**6353 compared to 6384**) possibly due to the fact that it is harder to get the additional data for the businesses. But while a few businesses might be missing (~30), there is almost twice as much data (**656KB compared to 1.2MB**) in the 'complete' files if we look at byte counts.

We can endlessly explore and compare these files and contents, I encourage you to perform similar comparisons with the other extra files (SFFoodProgram_Complete_Data/) in the directories and dive deeper into each file itself. For our examination of the data, I am happy with what we have accomplished. Given these new insights, we have enough information to continue on with our analysis.

Prepare

This is typically what people refer to as data 'munging' (or 'wrangling') and often is the most tedious process when working with messy data. Due to increasing awareness of the importance of data quality, the city of SF has been making great strides in more open and [accessible \(http://www.datasf.org/\)](http://www.datasf.org/) data. If you (the city of SF) know the [format \(http://www.yelp.com/healthscores\)](http://www.yelp.com/healthscores) you will need going into the data collection process (inspecting restaurants) you can hopefully avoid a lot of pain later in the analysis process.

The preparation process of our analysis is not as long and cumbersome as it typically might be due to the high quality of the raw data. Because of this, I will spare you much of the tedium of this step so we can focus on the more interesting aspects of the analysis. If you want to see (and experience) the pain (all you masochists out there), we will get much deeper into data acquisition and scrubbing techniques in our data wrangling [class \(http://team-zipfian-data-wrangling.eventbrite.com/\)](http://team-zipfian-data-wrangling.eventbrite.com/) of this series.

Transform

Now that we know the structure of our data, we can start to begin examining it statistically to get a macroscopic look at its distribution. This part of our tutorial will use much of the powerful built in functionality of [NumPy \(http://www.numpy.org/\)](http://www.numpy.org/), [SciPy \(http://www.scipy.org/\)](http://www.scipy.org/), [matplotlib \(http://matplotlib.org/\)](http://matplotlib.org/), and [pandas \(http://pandas.pydata.org/\)](http://pandas.pydata.org/). If you want to get more experience with these, there are great [resources \(http://fperez.org/py4science/starter_kit.html\)](http://fperez.org/py4science/starter_kit.html) and [tutorials \(http://www.rexx.com/~dkuhlman/scipy_course_01.html\)](http://www.rexx.com/~dkuhlman/scipy_course_01.html) covering these libraries in much more [depth \(http://scipy-lectures.github.io/\)](http://scipy-lectures.github.io/) than I will here. I highly recommend taking a look at these if this analysis interests you even in the least bit.

```
In [40]: '''
To perform some interesting statistical analyses, we first need to "join" our CSV files in order to associate businesses
with their inspection scores. This data currently resides in SFBusinesses/businesses.csv and SFBusinesses/inspections.csv
'''

# import pandas library which provides an R like environment for python.
# if you do not have it installed: sudo easy_install pandas.
from pandas import Series, DataFrame

import pandas as pd

# store relevant file paths in variables since we may use them frequently
root_dir = 'SFBusinesses/'
businesses = root_dir + 'businesses.csv'
inspections = root_dir + 'inspections.csv'

# load each file into a Pandas DataFrame, pandas automatically converts the first line into a header for the columns

df_business = pd.read_csv(businesses)
```

```
df_inspection = pd.read_csv(inspections)

# inspect the first 10 rows of the DataFrame
df_inspection.head(10)
```

Out[40]:

	business_id	Score	date	type
0	10	98	20121114	routine
1	10	98	20120403	routine
2	10	100	20110928	routine
3	10	96	20110428	routine
4	10	100	20101210	routine
5	12	100	20121120	routine
6	12	98	20120420	routine
7	12	100	20111018	routine
8	12	100	20110401	routine
9	17	100	20120823	routine

```
In [41]: '''
we can 'join' DataFrames just as we would database tables
pandas uses a left-outer join by default, meaning that all
the records from the businesses will be present even if there
is not a corresponding row in the inspections.
'''

# join the two DataFrames on the 'business_id' column
big_table = df_business.merge(df_inspection, on='business_id')

# the joined DataFrame columns: frame1 columns + frame2 columns
# in our case it is the concatenation of df_business and df_inspection columns
print 'Business:\t' + str(df_business.columns) + '\n'
print 'Inspection:\t' + str(df_inspection.columns) + '\n'
print 'Big Table:\t' + str(big_table.columns)

# allows for row and column indexing succinctly
big_table.iloc[:10, :4]
```

```
Business:      Index([business_id, name, address, city, state, postal_code, latitude, longitude, phone_number], dtype=object)

Inspection:    Index([business_id, Score, date, type], dtype=object)

Big Table:     Index([business_id, name, address, city, state, postal_code, latitude, longitude, phone_number, Score, date, type],
dtype=object)
```

Out[41]:

	business_id	name	address	city
0	10	TIRAMISU KITCHEN	033 BELDEN PL	San Francisco
1	10	TIRAMISU KITCHEN	033 BELDEN PL	San Francisco
2	10	TIRAMISU KITCHEN	033 BELDEN PL	San Francisco
3	10	TIRAMISU KITCHEN	033 BELDEN PL	San Francisco
4	10	TIRAMISU KITCHEN	033 BELDEN PL	San Francisco
5	12	KIKKA	250 EMBARCADERO 7/F	San Francisco
6	12	KIKKA	250 EMBARCADERO 7/F	San Francisco
7	12	KIKKA	250 EMBARCADERO 7/F	San Francisco
8	12	KIKKA	250 EMBARCADERO 7/F	San Francisco
9	17	GEORGE'S COFFEE SHOP	2200 OAKDALE AVE	San Francisco

Now that we have our joined data, we can start exploring it

```
In [42]: # let us first group our data by business so we can find its most recent score for the inspections

grouped_business = big_table.groupby('business_id')

# a function that takes a DataFrame and returns the row with the newest date
def most_recent(df, column='date'):
    return df.sort_index(by=column)[-1:]

# input to most_recent is the DataFrame of each group, in this case
# all of the rows and columns for each business (grouped on business_id).
most_recent_inspection_results = grouped_business.apply(most_recent)

# We applied the most_recent function to extract the row
```

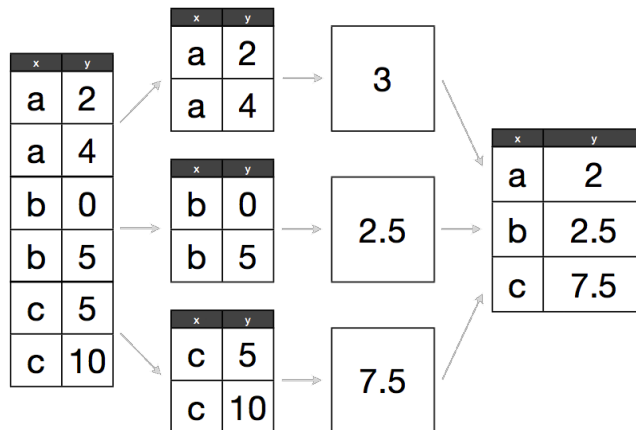
```
# of the DataFrame with the most recent inspection.
most_recent_inspection_results[['name', 'date', 'Score', 'type']].head()
```

Out[42]:

		name	date	Score	type
business_id					
10	0	TIRAMISU KITCHEN	20121114	98	routine
12	5	KIKKA	20121120	100	routine
17	9	GEORGE'S COFFEE SHOP	20120823	100	routine
19	13	NRGIZE LIFESTYLE CAFE	20121127	100	routine
24	18	OMNI S.F. HOTEL - 2ND FLOOR PANTRY	20121018	100	routine

In [43]: `Image(url='http://inundata.org/R_talks/meetup/images/splitapply.png', width=500)`

Out[43]:



Split-Apply-Combine

A visual representation of how group-by, aggregate, and apply semantics work

We can bin the restaurants by scores to understand the distribution of inspections better. Here we create a histogram to understand the distribution of scores better

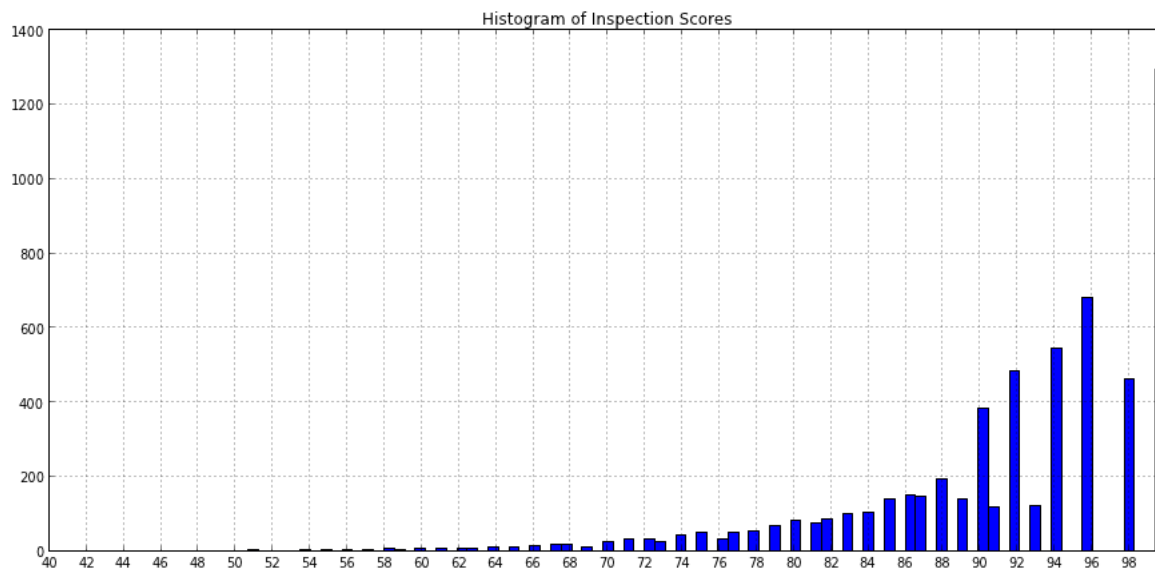
```
In [44]: # create a matplotlib figure with size [15,7]
figure(figsize=[15,7])

# pandas built-in histogram function automatically distributes and counts bin values
h = most_recent_inspection_results['Score'].hist(bins=100)

# create x-axis ticks of even numbers 0-100
xticks(np.arange(40, 100, 2))

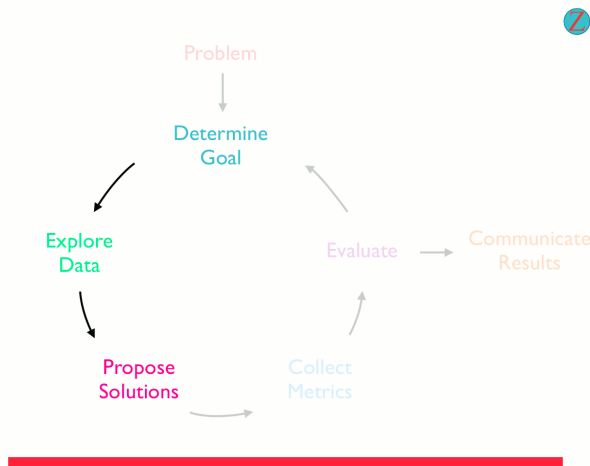
# add a title to the current figure, our histogram
h.set_title("Histogram of Inspection Scores")
```

Out[44]: <matplotlib.text.Text at 0xc023850>



In [45]: `Image(url='http://zipfianacademy.com/data/data-science-workflow/solutions.png', width=500)`

Out[45]:



Propose Solutions

Since we have explored our data and have a better idea of its nature, we can begin to devise a plan to answer our questions. This is usually the most iterative part of the entire process: as we learn more about our data we modify our approach, and as modify our solutions we must re-examine our data.

Goals:

- How does an individual restaurants' score compare to the whole/aggregate of SF?
- Are SF's inspections better or worse than other cities?
- If a restaurant has not yet been inspected, can we approximate/predict what score it will receive?

Solutions:

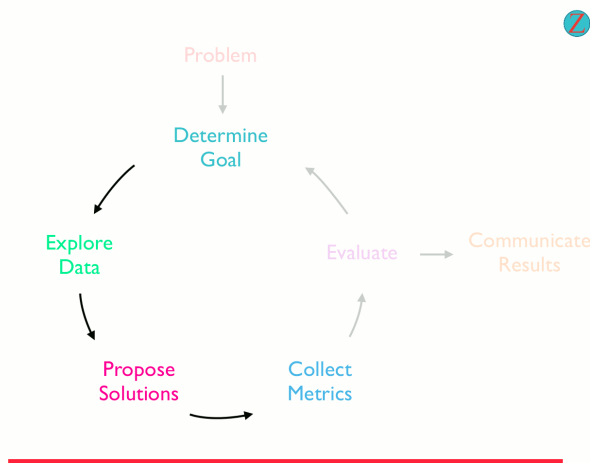
- Collect summary statistics (mean, median, standard deviation, etc.) about distribution of scores.
- Acquire data on inspection scores for other cities, compare distribution of cities.
- Perform a linear regression on historic data on past inspections combined with scores from other 'similar' restaurants.

Some things to note about formulating solutions:

- Ask, how can I address the problem?
- In what ways can I use the data to achieve my goals?
- The simplest solution is often best (and the one you should try first)
- Quantize the metrics needed for your analysis

In [46]: `Image(url='http://zipfianacademy.com/data/data-science-workflow/metrics.png', width=500)`

Out[46]:



Collect Metrics

This is the step where derivative values are often calculated, including **summary statistics**, **transformations** on the data, and **correlations**. There also is a bit of traditional **data mining** involved as most machine learning occurs in the solutions and metrics stages (in our formulation). We could even go so far as to say that the results of predictive models are simply additional metrics: the **probability** of defaulting on a loan, the **cluster** a new product belongs in, or the **score** of a restaurant that hasn't been inspected yet.

The purpose of this part of the process is to calculate the information you need to begin evaluating and testing you solutions and hypotheses.

```
In [47]: # compute descriptive summary statistics of the inspection scores
most_recent_inspection_results['Score'].describe()
```

```
Out[47]: count      5824.000000
         mean        91.590659
         std         8.296706
         min         44.000000
         25%         88.000000
         50%         94.000000
         75%         98.000000
         max        100.000000
         dtype: float64
```

```
In [48]: # recall that in the Score Legend, each numeric score corresponds to a more qualitative description
!head -n 5 SFBusinesses/ScoreLegend.csv | column -t -s ','
```

"Minimum_Score"	"Maximum_Score"	"Description"
0	70	"Poor"
71	85	"Needs Improvement"
86	90	"Adequate"
91	100	"Good"

```
In [49]: '''
let us compute a histogram of these descriptions as well
'''

# first we need to discretize the numerical values, this can be
# thought of as converting a continuous variable into a categorical one.
descriptions = ['Poor', 'Needs Improvement', 'Adequate', 'Good']
bins = [-1, 70, 85, 90, 100]

# copy the scores from our grouped DataFrame, DataFrames manipulate
# in place if we do not explicitly copy them.
scores = most_recent_inspection_results['Score'].copy()
score_transform = most_recent_inspection_results.copy()

# built-in pandas function which assigns each data point in
# 'scores' to an interval in 'bins' with labels of 'descriptions'
discretized_scores = pd.cut(scores, bins ,labels=descriptions)

print discretized_scores
```

```
Categorical: Score
array([Good, Good, Good, ..., Good, Adequate, Good], dtype=object)
Levels (4): Index([Poor, Needs Improvement, Adequate, Good], dtype=object)
```

```
In [50]: # tranform the original DataFrame's "Score" column with the new descriptions
score_transform['Score'] = discretized_scores

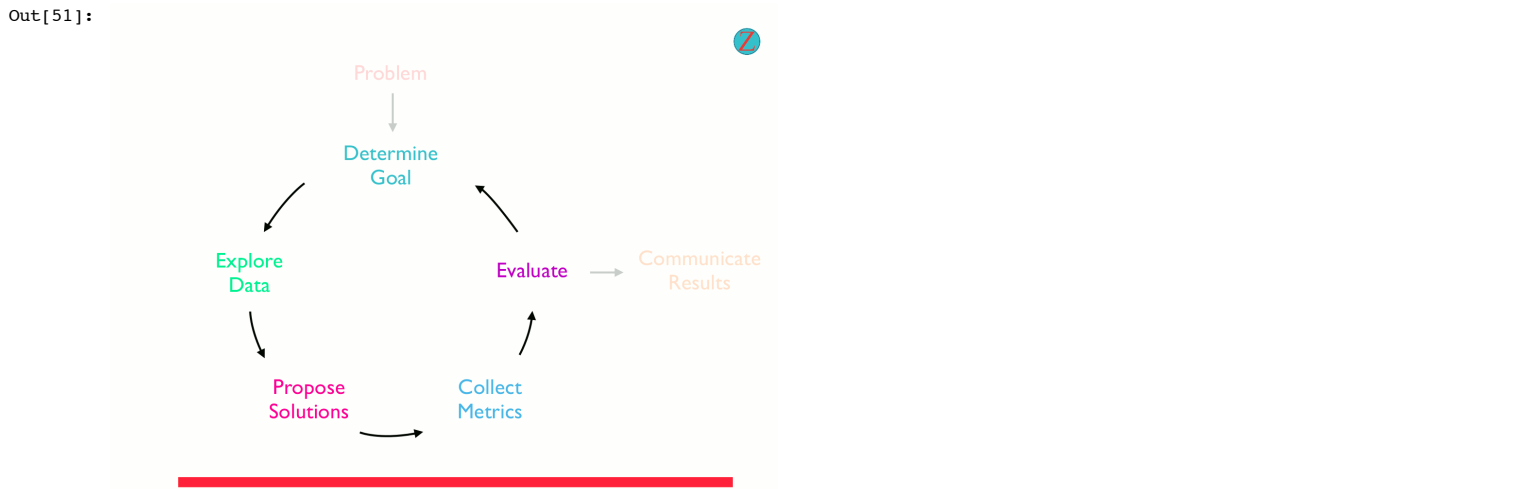
score_transform[['name', 'date','Score']].head(15)
```

Out[50]:

		name	date	Score
business_id				
10	0	TIRAMISU KITCHEN	20121114	Good
12	5	KIKKA	20121120	Good
17	9	GEORGE'S COFFEE SHOP	20120823	Good
19	13	NRGIZE LIFESTYLE CAFE	20121127	Good
24	18	OMNI S.F. HOTEL - 2ND FLOOR PANTRY	20121018	Good
29	23	CHICO'S PIZZA	20121228	Adequate
31	28	NORMAN'S ICE CREAM AND FREEZES	20121217	Good
37	33	CAFE BISTRO	20130130	Good
40	38	MO'S COFFEE BAR	20121213	Good
45	42	CHARLIE'S DELI CAFE	20121126	Good
48	48	ART'S CAFE	20130327	Adequate
50	53	SUSHI ZONE	20130328	Good
54	59	RHODA GOLDMAN PLAZA	20130326	Good
56	64	CAFE X + O	20130327	Good

By quantizing the scores of the restaurant inspections, we can get a better qualitative insight into the ratings. Let us compare this new distribution of quantized scores to the raw numeric values.

In [51]: `Image(url='http://zipfianacademy.com/data/data-science-workflow/evaluate.png', width=500)`



Evaluate

With the metrics we need properly calculated, it is time to draw some conclusions from our analyses. We need to evaluate whether the result we have arrived at:

- Answers our original question to an acceptable level of confidence.
- Has allowed us to achieve our goals?

```

In [52]: '''
plot a histogram of the discretized scores
'''

# create a figure with 2 subplots
fig = figure(figsize=[30,7])
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)

# count each occurrence of descriptions in the 'Score' column,
# and reverse this result so 'Poor' is left most and 'Good' right most
counts = score_transform['Score'].value_counts()[::-1]
plt = counts.plot(kind='bar', ax=ax2)

# decorate the plot and axis with text
ax2.set_title("Restaurant Inspections (%i total)" % sum(counts))
ax2.set_ylabel("Counts")
ax2.set_xlabel("Description")

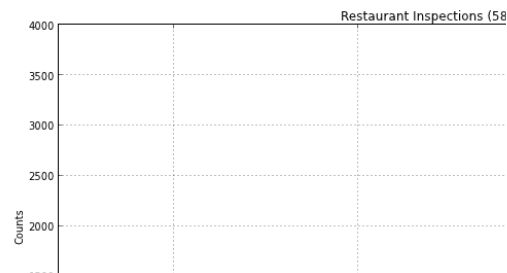
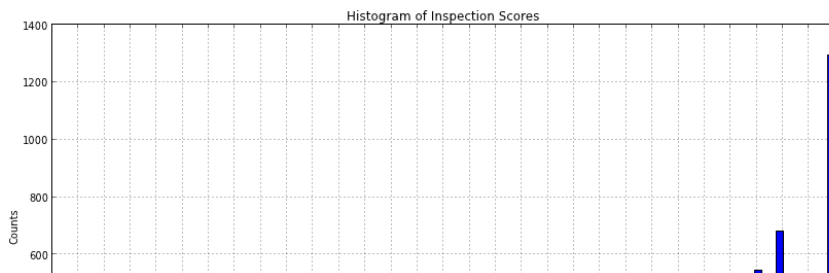
# let us add some labels to each bar
for x, y in enumerate(counts):
    plt.text(x + 0.5, y + 200, '%.f' % y, ha='left', va='top')

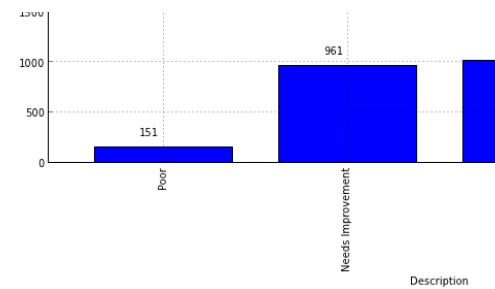
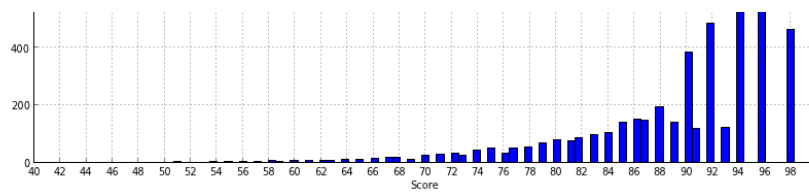
# plot the original raw scores (same graph as earlier)
most_recent_inspection_results['Score'].hist(bins=100, ax=ax1)
# create x-axis ticks of even numbers 0-100
ax1.set_xticks(np.arange(40, 100, 2))

# add a title to the current figure, our histogram
ax1.set_title("Histogram of Inspection Scores")
ax1.set_ylabel("Counts")
ax1.set_xlabel("Score")

```

Out[52]: `<matplotlib.text.Text at 0xab60f0>`





We can see that a majority of restaurants are **adequate** or **good**, according to the quantiles only **25%** have scores less than **88**. While the histogram of the numeric scores gives us a more granular look at the data, it can be quite difficult to derive value from it. Is an **86** a filthy/unhealthy restaurant or did it simply forget a few nuanced inspection rules? The Score Legend provides us a mapping from a raw score to a meaningful value, similar to the scaling of standardized test raw scores.

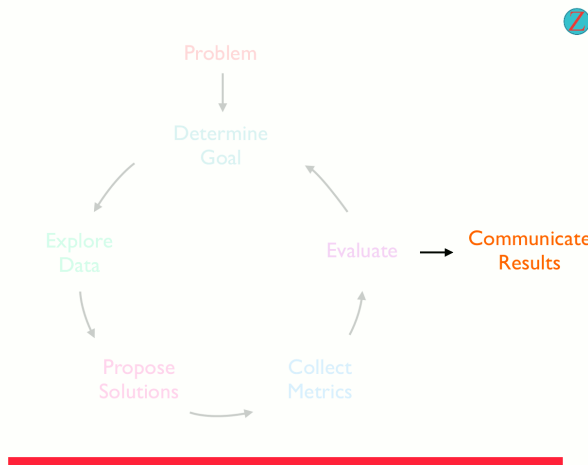
If we are not satisfied with our evaluation, we need to iterate on our approach:

- Do I need more/better data?
- Do I need to try a different proposed solution?
- Do I need to calculate different metrics?

I will leave the process of iteration (out of the scope of this exercise) as an additional task for the student. With such analyses as these you can endlessly refine your solution and find new questions to ask... and I whole heartedly encourage all of you to do so!

In [53]: `Image(url='http://zipfianacademy.com/data/data-science-workflow/communicate.png', width=500)`

Out[53]:



Communicate Results

This is the step where derivative values are often calculated, including **summary statistics**, **transformations** on the data, and **correlations**. There also is a bit of traditional **data mining** involved as most machine learning occurs in the solutions and metrics stages (in our formulation). We could even go so far as to say that the results of predictive models are simply additional metrics: the **probability** of defaulting on a loan, the **cluster** a new product belongs in, or the **score** of a restaurant that hasn't been inspected yet.

The purpose of this part of the process is to calculate the information you need to begin evaluating and testing you solutions and hypotheses.