Dan Toomey

Jupyter for Data Science

Exploratory analysis, statistical modeling, machine learning, and data visualization with Jupyter

Packt

Contents

Chapter 1. Jupyter and Data Science

The Jupyter product was derived from the IPython project. The IPython project was used to provide interactive online access to Python. Over time it became useful to interact with other programming languages, such as R, in the same manner. With this split from only Python, the tool grew into its current manifestation of Jupyter. IPython is still an active tool available for use.

Jupyter is available as a web application for a wide variety of platforms. It can also be used on your desktop/laptop over a wide variety of installations. In this book, we will be exploring using Jupyter from a Windows PC and over the internet for other providers.

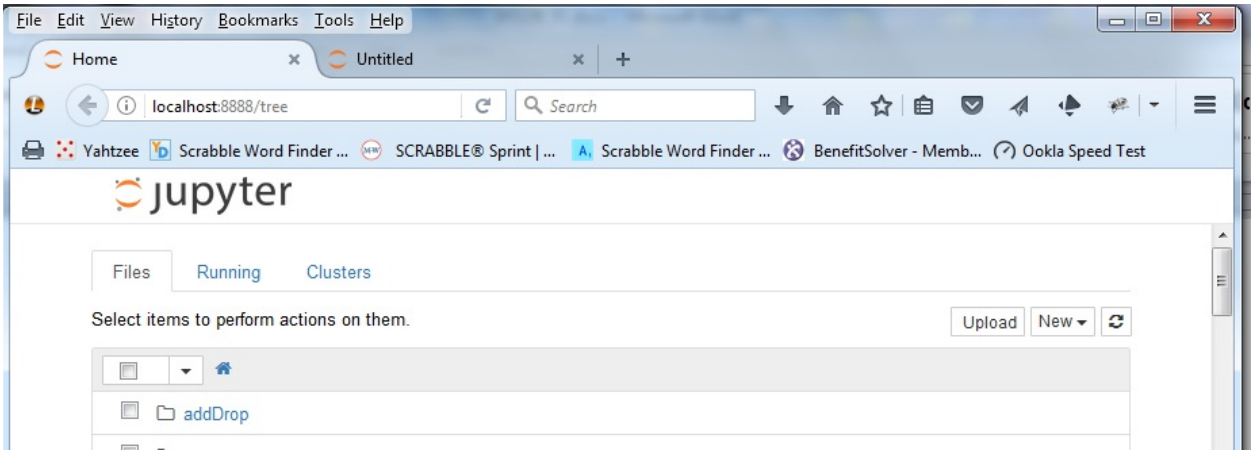
Jupyter concepts

Jupyter is organized around a few basic concepts:

- **Notebook**: A collection of statements (in a language). For example, this could be a complete R script that loads data, analyzes it, produces a graph, and records results elsewhere.
- **Cell**: the lowest granular piece of a Jupyter Notebook that can be worked with:
  - **Current Cell**: The current cell being edited or the one(s) selected
- **Kernel**: each notebook is associated with a specific language implementation. The part of Jupyter which processes the specific language involved is called a kernel.

A first look at the Jupyter user interface

We can jump right in and see what Jupyter has to offer. A Jupyter screen looks like this:



Note

So, Jupyter is deployed as a website that can be accessed on your machine (or can be accessed like any other website across the internet).

We see the URL of the page, `http://localhost:8888/tree`. `localhost` is a pseudonym for a web server running on your machine. The website we are accessing on the web server is in a *tree* display. This is the default display. This conforms to the display of the projects within Jupyter. Jupyter displays objects in a tree layout much like Windows File Explorer. The main page lists a number of projects; each project is its own subdirectory and contains a further delineation of content for each. Depending on where you start Jupyter, the existing contents of the current directory will be included in the display as well.

Detailing the Jupyter tabs

On the web page, we have the soon to be familiar Jupyter logo and three tabs:

- **`Files`**
- **`Running`**
- **`Clusters`**

The **`Files`** tab lists the objects available to Jupyter. The files used by Jupyter are stored as regular files on your disk. Jupyter...

Summary

In this chapter, we looked into the details of the Jupyter user interface: what objects does it work with, what actions can be taken by Jupyter, what does the display tell us about the data, and what tools are available? Next, we looked at some real-life examples from industry showing R and Python coding from several industries. Then we saw some of the ways to share our notebook with other users and, correspondingly, how to protect our notebook with different security mechanisms.

In the next chapter, we will see how far we can go using Python in a Jupyter Notebook.

Chapter 2. Working with Analytical Data on Jupyter

Jupyter does none of the heavy lifting for analyzing data: all the work is done by programs written in a selected language. Jupyter provides the framework to run a variety of programming language modules. So, we have a choice how we analyze data in Jupyter.

A popular choice for data analysis programming is Python. Jupyter does have complete support for Python programming. We will look at a variety of programming solutions that might tax such a support system and see how Jupyter fairs.

Data scraping with a Python notebook

A common tool for data analysis is gathering the data from a public source such as a website. Python is adept at scraping websites for data. Here, we look at an example that loads stock price information from Google Finance data.

In particular, given a stock symbol, we want to retrieve the last year of price ranges for that symbol.

One of the pages on the Google Finance site will give the last years' worth of price data for a security company. For example, if we were interested in the price points for **Advanced Micro Devices** (**AMD**), we would enter the following URL:

https://www.google.com/finance/historical?q=NASDAQ:AMD

Here, NASDAQ is the stock exchange that carries the AMD security. On the resultant Google page, there is a table of data points of interest, as seen in the following partial screenshot.

Like many sites that you will be attempting to access, there is a lot of other information on the page as well, like headers and footers and ads, as you can see in the following screenshot. The web pages are built for human readers. Fortunately, Google and these other companies realize you are scraping...

Using heavy-duty data processing functions in Jupyter

Python has several groups of processing functions that can tax computer system power. Let us use some of these in Jupyter and determine if the functionality performs as expected.

Using NumPy functions in Jupyter

NumPy is a package in Python providing multidimensional arrays and routines for array processing. We bring in the NumPy package using `import * from numpy` statement. In particular, the NumPy package defines the `array` keyword, referencing a NumPy object with extensive functionality.

The NumPy array processing functions run from the mundane, such as `min()` and `max()` functions (which provide the minimum and maximum values over the array dimensions provided), to more interesting utility functions for producing histograms and calculating correlations using the elements of a data frame.

With NumPy, you can manipulate arrays in many ways. For example, we will go over some of these functions with the following scripts, where we will use NumPy to:

- Create an array
- Calculate the max value in the array
- Calculate the min value in the array
- Determine the sum across the second axis

```
# numpy...
```

Using SciPy in Jupyter

SciPy is an open source library for mathematics, science and, engineering. With such a wide scope, there are many areas we can explore using SciPy:

- Integration
- Optimization
- Interpolation
- Fourier transforms
- Linear algebra
- There are several other intense sets of functionality as well, such as signal processing

Using SciPy integration in Jupyter

A standard mathematical process is integrating an equation. SciPy accomplishes this using a callback function to iteratively calculate out the integration of your function. For example, suppose that we wanted to determine the integral of the following equation:

$$\int 2pi + 1$$

We would use a script like the following. We are using the definition of *pi* from the standard `math` package.

```
from scipy.integrate import quadimport mathdef integrand(x, a, b):    return a*math.pi + ba =
2b = 1quad(integrand, 0, 1, args=(a,b))
```

Again, this coding is very clean and simple, yet almost impossible to do in many languages. Running this script in Jupyter we see the results quickly:

```
In [5]: from scipy.integrate import quad
        import math

        def integrand(x, a, b):
            return a*math.pi + b

        a = 2
        b = 1
        quad(integrand, 0, 1, args=(a,b))
Out[5]: (7.283185307179586, 8.08596002064242e-14)
```

I was curious how the `integrand` function is used during the execution. I am using this to exercise a call back function. To see this...

Expanding on panda data frames in Jupyter

There are more functions built-in for working with data frames than we have used so far. If we were to take one of the data frames from a prior example in this chapter, the Titanic dataset from an Excel file, we could use additional functions to help portray and work with the dataset.

As a repeat, we load the dataset using the script:

```
import pandas as pddf =
pd.read_excel('http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.xls')
```

We can then inspect the data frame using the `info` function, which displays the characteristics of the data frame:

```
df.info()
```

```
In [12]: print(df.info())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 14 columns):
pclass       1309 non-null int64
survived     1309 non-null int64
name         1309 non-null object
sex          1309 non-null object
age          1046 non-null float64
sibsp        1309 non-null int64
parch        1309 non-null int64
ticket       1309 non-null object
fare         1308 non-null float64
cabin        295 non-null object
embarked     1307 non-null object
boat         486 non-null object
body         121 non-null float64
home.dest    745 non-null object
dtypes: float64(3), int64(4), object(7)
memory usage: 143.2+ KB
None
```

Some of the interesting points are as follows:

- 1309 entries
- 14 columns
- Not many fields with valid data in the `body` column—most were lost
- Does give a good overview of the types of data involved

We can also use the `describe` function, which gives us a statistical breakdown of the number columns in the data frame.

```
df.describe()
```

This produces the following tabular display:

```
In [4]: df.describe()
```

Out[4]:

	pclass	survived	age	sibsp	parch	fare	body
count	1309.000000	1309.000000	1046.000000	1309.000000	1309.000000	1308.000000	121.000000
mean	2.294882	0.381971	29.881135	0.498854	0.385027	33.295479	160.809917
std	0.837836	0.486055	14.413500	1.041658	0.865560	51.758668	97.696922
min	1.000000	0.000000	0.166700	0.000000	0.000000	0.000000	1.000000
25%	2.000000	0.000000	21.000000	0.000000	0.000000	7.895800	72.000000
50%	3.000000	0.000000	28.000000	0.000000	0.000000	14.454200	155.000000
75%	3.000000	1.000000	39.000000	1.000000	0.000000	31.275000	256.000000
max	3.000000	1.000000	80.000000	8.000000	9.000000	512.329200	328.000000

For each numerical column we have:

- Count
- Mean
- Standard deviation
- 25, 50, and 75 percentile points
- Min, max values for the item

We can slice rows of interest using the syntax

Summary

In this chapter, we looked at some of the more compute intensive tasks that might be performed in Jupyter. We used Python to scrape a website to gather data for analysis. We used Python NumPy, pandas, and SciPy functions for in-depth computation of results. We went further into pandas and explored manipulating data frames. Lastly, we saw examples of sorting and filtering data frames.

In the next chapter, we will make some predictions and use visualization to validate our predictions.

Chapter 3. Data Visualization and Prediction

Making predictions is usually precarious. However, there are methods that have been in use that provide some confidence in your results. Under Jupyter, we can use Python and/or R for predictions with readily available functionality.

Make a prediction using scikit-learn

scikit-learn is a machine learning toolset built using Python. Part of the package is supervised learning, where the sample data points have attributes that allow you to assign the data points into separate classes. We use an estimator that assigns a data point to a class and makes predictions as to other data points with similar attributes. In scikit-learn, an estimator provides two functions, `fit()` and `predict()`, providing mechanisms to classify data points and predict classes of other data points, respectively.

As an example, we will be using the housing data from https://uci.edu/ (I think this is data for the Boston area). There are a number of factors including a price factor.

We will take the following steps:

- We will break up the dataset into a training set and a test set
- From the training set, we will produce a model
- We will then use the model against the test set and evaluate how well our model fits the actual data for predicting housing prices

The attributes in the dataset (in corresponding order in our data frame) are:

CRIM per capita crime rate by town

`ZN` proportion of residential land zoned...

Make a prediction using R

We can perform the same analysis using R in a notebook. The functions are different for the different language, but the functionality is very close.

We use the same algorithm:

- Load the dataset
- Split the dataset into training and testing partitions
- Develop a model based on the training partition
- Use the model to predict from the testing partition
- Compare predicted versus actual testing

The coding is as follows:

```
#load in the data set from uci.edu (slightly different from other housing model)housing <-
read.table("http://archive.ics.uci.edu/ml/machine-learning-
databases/housing/housing.data")#assign column namescolnames(housing) <- c("CRIM", "ZN",
"INDUS", "CHAS", "NOX",                  "RM", "AGE", "DIS", "RAD", "TAX", "PRATIO",
"B", "LSTAT", "MDEV")#make sure we have the right data being loadedsummary(housing)      CRIM
ZN             INDUS            CHAS         Min.   : 0.00632   Min.   :  0.00   Min.   : 0.46
Min.   :0.00000   1st Qu.: 0.08204   1st Qu.:  0.00   1st Qu.: 5.19   1st Qu.:0.00000   Median
: 0.25651   Median :  0.00   Median : 9.69   Median :0.00000   Mean   :...
```

Interactive visualization

There is a Python package, Bokeh, that can be used to generate a figure in your notebook where the user can interact and change the figure.

In this example, I am using the same data from the histogram example later in this chapter (also included in the file set for this chapter) to display an interactive Bokeh histogram.

The coding is as follows:

```
from bokeh.io import show, output_notebook
from bokeh.charts import Histogram
import numpy as np
import pandas as pd
# this step is necessary to have display inline in a notebook
output_notebook()
# load the counts from other histogram example
from_counts = np.load("from_counts.npy")
# convert array to a dataframe for Histogram
df = pd.DataFrame({'Votes':from_counts})
# make sure dataframe is working correctly
print(df.head())
   Votes
0     23
1     29
2     23
3    302
4     24
# display the Bokeh histogram
hist = Histogram(from_counts, \
title="How Many Votes Made By Users", \
bins=12)
show(hist)
```

We can see the histogram displayed as follows. There is little being done automatically to clean up the graph, such as move counters around or the uninteresting axes...

Plotting using Plotly

Plotly is an interesting mix. It is a subscription website that provides significant data analysis graphing functionality. You can use the free version of the software, but you still need to log in with credentials to use it. The graphics functions are available in a variety of languages from Node.js to Python and the like.

Further, the graphics generated are available in Plotly and in your local notebook. If you mark the graphic as public, then you can access it from the notebook, just like any other graphic over the internet. Similarly, as a web graphic, you can select from the display and save locally as needed.

In this example, we use the voting histogram again, but using Plotly's capabilities.

The script becomes the following:

```
import plotlyimport plotly.graph_objs as goimport plotly.plotly as pyimport pandas as pdimport numpy as np#once you set credentials they are stored in local space and referenced automatically#you need to subscribe to the site to get the credentials#the api key would need to be replaced with your key#plotly.tools.set_credentials_file(username='DemoAccount', api_key='lr17zw81')#we are...
```

Creating a human density map

I had originally planned on producing a worldwide human density map, but the graphics available don't allow for setting the color of each country. So, I built a density map for the United States.

The algorithm is:

1. Obtain graphic shapes for each of the states.
2. Obtain the density for each state.
3. Decide on a color range and apply the lowest density to one end of the range and the highest to the other end.
4. For each state:
    - Determine it's density
    - Lookup that density value in the range and select a color
    - Draw the state

This is coded with the following (comments embedded as the code proceeds):

```
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
from matplotlib.patches import Polygon
import pandas as pd
import numpy as np
import matplotlib
# create the map
map = Basemap(llcrnrlon=-119,llcrnrlat=22,urcrnrlon=-64,urcrnrlat=49,
        projection='lcc',lat_1=33,lat_2=45,lon_0=-95)
# load the shapefile, use the name 'states'
# download from
https://github.com/matplotlib/basemap/tree/master/examples/st99_d00.dbf,shx,shp
map.readshapefile('st99_d00', name='states', drawbounds=True)
#...
```

Draw a histogram of social data

There are a wide variety of social sites that produce datasets. In this example, we will gather one of the datasets and produce a histogram from the data. The specific dataset is the voting behavior on WIKI from https://snap.stanford.edu/data/wiki-Vote.html. Each data item shows user number N voted for user number X. So, we produce some statistics in a histogram to analyze voting behavior by:

- Gathering all of the voting that took place
- For each vote:
    - Increment a counter that says who voted
    - Increment a counter that says who was voted for
    - Massage the data so we can display it in two histograms

The coding is as follows:

```
%matplotlib inline
# import all packages being used
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import matplotlib

# load voting data drawn from https://snap.stanford.edu/data/wiki-Vote.html
df = pd.read_table('wiki-Vote.txt', sep=r"\s+", index_col=0)

# produce standard summary info to validate
print(df.head())
print(df.describe())
```

Note

Python will automatically assign the first column as the index into the table, regardless of whether the index is re-used (as is...

Plotting 3D data

Many of the data analysis packages (R, Python, and so on) have significant data visualization capabilities. An interesting one is to display data in three dimensions. Often, when three dimensions are used, unexpected visualizations appear.

For this example, we are using the car dataset from https://uci.edu/. It is a well-used dataset with several attributes for vehicles, for example, `mpg`, `weight`, and `acceleration`. What if we were to plot three of those data attributes together and see if we can recognize any apparent rules?

The coding involved is as follows:

```
%matplotlib inline
# import tools we are using
import pandas as pd
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
# read in the car 'table' – not a csv, so we need
# to add in the column names
column_names = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration',
'year', 'origin', 'name']
df = pd.read_table('http://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-
mpg.data', \
                   sep=r"\s+", index_col=0, header=None, names = column_names)
print(df.head())
     ...
```

Summary

In this chapter, we used prediction models from Python and R under Jupyter. We used Matplotlib for data visualization. We used interactive plotting (under Python). And we covered several graphing techniques available in Jupyter. We created a density map with SciPy. We used histograms to visualize social data. Lastly, we generated a 3D plot under Jupyter.

In the next chapter, we will look at accessing data in different ways under Jupyter.

Chapter 4. Data Mining and SQL Queries

PySpark exposes the Spark programming model to Python. Spark is a fast, general engine for large-scale data processing. We can use Python under Jupyter. So, we can use Spark in Jupyter.

Installing Spark requires the following components to be installed on your machine:

- Java JDK.
- Scala from http://www.scala-lang.org/download/.
- Python recommend downloading Anaconda with Python (from http://continuum.io).
- Spark from https://spark.apache.org/downloads.html.
- `winutils`: This is a command-line utility that exposes Linux commands to Windows. There are 32-bit and 64-bit versions available at:
  - 32-bit `winutils.exe` at https://code.google.com/p/rrd-hadoop-win32/source/checkout
  - 64-bit `winutils.exe` at https://github.com/steveloughran/winutils/tree/master/hadoop-2.6.0/bin

Then set environment variables that show the position of the preceding components:

- `JAVA_HOME`: The bin directory where you installed JDK
- `PYTHONPATH`: Directory where Python was installed
- `HADOOP_HOME`: Directory where `winutils` resides
- `SPARK_HOME`: Where Spark is installed

These components are readily available over the internet for a variety of operating...

Special note for Windows installation

Spark (really Hadoop) needs a temporary storage location for its working set of data. Under Windows this defaults to the `\tmp\hive` location. If the directory does not exist when Spark/Hadoop starts it will create it. Unfortunately, under Windows, the installation does not have the correct tools built-in to set the access privileges to the directory.

You should be able to run `chmod` under `winutils` to set the access privileges for the `hive` directory. However, I have found that the `chmod` function does not work correctly.

A better idea has been to create the `tmp\hive` directory yourself in admin mode. And then grant full privileges to the hive directory to all users, again in admin mode.

Without this change, Hadoop fails right away. When you start `pyspark`, the output (including any errors) are displayed in the command line window. One of the errors will be insufficient access to this directory.

Using Spark to analyze data

The first thing to do in order to access Spark is to create a `SparkContext`. The `SparkContext` initializes all of Spark and sets up any access that may be needed to Hadoop, if you are using that as well.

The initial object used to be a `SQLContext`, but that has been deprecated recently in favor of `SparkContext`, which is more open-ended.

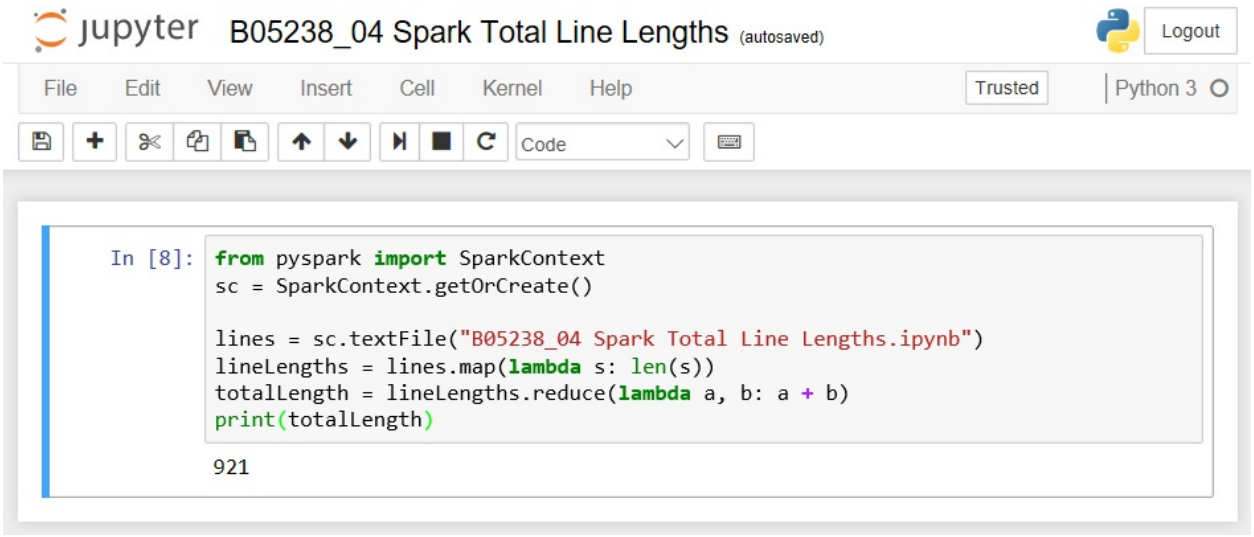
We could use a simple example to just read through a text file as follows:

```
from pyspark import SparkContextsc = SparkContext.getOrCreate()lines = sc.textFile("B05238_04 Spark Total Line Lengths.ipynb")lineLengths = lines.map(lambda s: len(s))totalLength = lineLengths.reduce(lambda a, b: a + b)print(totalLength)
```

In this example:

- We obtain a `SparkContext`
- With the context, read in a file (the Jupyter file for this example)
- We use a Hadoop `map` function to split up the text file into different lines and gather the lengths
- We use a Hadoop `reduce` function to calculate the length of all the lines
- We display our results

Under Jupyter this looks like the following:



Another MapReduce example

We can use MapReduce in another example where we get the word counts from a file. A standard problem, but we use MapReduce to do most of the heavy lifting. We can use the source code for this example. We can use a script similar to this to count the word occurrences in a file:

```
import pysparkif not 'sc' in globals():    sc = pyspark.SparkContext()text_file =
sc.textFile("Spark File Words.ipynb")counts = text_file.flatMap(lambda line: line.split(" "))
\             .map(lambda word: (word, 1)) \             .reduceByKey(lambda a, b: a + b)for x
in counts.collect():    print x
```

Note

We have the same preamble to the coding.

Then we load the text file into memory.

Note

`text_file` is a Spark **RDD** (**Resilient Distributed Dataset**), not a data frame.

It is assumed to be massive and the contents distributed over many handlers.

Once the file is loaded we split each line into words, and then use a `lambda` function to tick off each occurrence of a word. The code is truly creating a new record for each word occurrence, such as *at appears 1*, *at appears 1*. For example, if the word *at* appears twice each occurrence would have a record...

Using SparkSession and SQL

Spark exposes many SQL-like actions that can be taken upon a data frame. For example, we could load a data frame with product sales information in a CSV file:

```
from pyspark.sql import SparkSession spark = SparkSession(sc) df = spark.read.format("csv") \
.option("header", "true") \        .load("productsales.csv");df.show()
```

The example:

- Starts a `SparkSession` (needed for most data access)
- Uses the session to read a CSV formatted file, that contains a header record
- Displays initial rows

```
from pyspark.sql import SparkSession
spark = SparkSession(sc)

df = spark.read.format("csv") \
        .option("header", "true") \
        .load("productsales.csv");
df.show()
```

```
+------+-------+-------+------+---------+---------+-------+-------+----+-----+
|ACTUAL|PREDICT|COUNTRY|REGION| DIVISION| PRODTYPE|PRODUCT|QUARTER|YEAR|MONTH|
+------+-------+-------+------+---------+---------+-------+-------+----+-----+
|   925|    850| CANADA|  EAST|EDUCATION|FURNITURE|   SOFA|      1|1993|12054|
|   999|    297| CANADA|  EAST|EDUCATION|FURNITURE|   SOFA|      1|1993|12085|
|   608|    846| CANADA|  EAST|EDUCATION|FURNITURE|   SOFA|      1|1993|12113|
|   642|    533| CANADA|  EAST|EDUCATION|FURNITURE|   SOFA|      2|1993|12144|
|   656|    646| CANADA|  EAST|EDUCATION|FURNITURE|   SOFA|      2|1993|12174|
```

We have a few interesting columns in the sales data:

- Actual sales for the products by division
- Predicted sales for the products by division

If this were a bigger file, we could use SQL to determine the extent of the product list. Then the following is the Spark SQL to determine the product list:

```
df.groupBy("PRODUCT").count().show()
```

The data frame `groupBy` function works very similar to the SQL `Group By` clause. `Group By` collects the items in the dataset according to the values in the column specified. In this case the `PRODUCT` column. The `Group By` results in a dataset being established with the results. As a dataset, we can query...

Combining datasets

So, we have seen moving a data frame into Spark for analysis. This appears to be very close to SQL tables. Under SQL it is standard practice not to reproduce items in different tables. For example, a product table might have the price and an order table would just reference the product table by product identifier, so as not to duplicate data. So, then another SQL practice is to join or combine the tables to come up with the full set of information needed. Keeping with the order analogy, we combine all of the tables involved as each table has pieces of data that are needed for the order to be complete.

How difficult would it be to create a set of tables and join them using Spark? We will use example tables of `Product`, `Order`, and `ProductOrder`:

Table	Columns
Product	Product ID, Description, Price
Order	Order ID, Order Date
ProductOrder	Order ID, Product ID, Quantity

So, an `Order` has a list of `Product`/`Quantity` values associated.

We can populate the data frames and move them into Spark:

```
from pyspark import SparkContextfrom pyspark.sql import SparkSessionsc =
SparkContext.getOrCreate()spark = SparkSession(sc) # load product set
```

Loading JSON into Spark

Spark can also access JSON data for manipulation. Here we have an example that:

- Loads a JSON file into a Spark data frame
- Examines the contents of the data frame and displays the apparent schema
- Like the other preceding data frames, moves the data frame into the context for direct access by the Spark session
- Shows an example of accessing the data frame in the Spark context

The listing is as follows:

Our standard includes for Spark:

```
from pyspark import SparkContextfrom pyspark.sql import SparkSession sc =
SparkContext.getOrCreate()spark = SparkSession(sc)
```

Read in the JSON and display what we found:

```
#using some data from file from https://gist.github.com/marktyers/678711152b8dd33f6346df =
spark.read.json("people.json")df.show()
```

```
from pyspark import SparkContext
from pyspark.sql import SparkSession

sc = SparkContext.getOrCreate()
spark = SparkSession(sc)
```

```
#using some data from file from https://gist.github.com/marktyers/678711152b8dd33f6346
df = spark.read.json("people.json")
df.show()
```

```
+----+----+--------------------+---------------+
| age|born|                fame|           name|
+----+----+--------------------+---------------+
|null|null|                null|        Michael|
|  30|null|                null|           Andy|
|  19|null|                null|         Justin|
|null|1955|co-founder of App...|     Steve Jobs|
|null|1955|                null|Tim Berners-Lee|
|null|1815|                null|   George Boole|
+----+----+--------------------+---------------+
```

I had a difficult time getting a standard JSON to load into Spark. Spark appears to expect one record of data per list of the JSON file versus most JSON I have seen pretty much formats the record layouts with indentation and the like.

Note

Notice the use of null values where an attribute was not specified in an instance.

Display the interpreted schema for the data:

`df.printSchema()`

```
df.printSchema()
```

```
root
 |-- age: long (nullable = true)
 |-- born: long (nullable = true)
 |-- fame: string (nullable = true)
 |-- name: string (nullable = true)
```

The default for...

Using Spark pivot

The `pivot()` function allows you to translate rows into columns while performing aggregation on some of the columns. If you think about it you are physically adjusting the axes of a table about a pivot point.

I thought of an easy example to show how this all works. I think it is one of those features that once you see it in action you realize the number of areas that you could apply it.

In our example, we have some raw price points for stocks and we want to convert that table about a pivot to produce average prices per year per stock.

The code in our example is:

```
from pyspark import SparkContextfrom pyspark.sql import SparkSessionfrom pyspark.sql import
functions as funcsc = SparkContext.getOrCreate()spark = SparkSession(sc)# load product
setpivotDF = spark.read.format("csv") \        .option("header", "true") \
.load("pivot.csv");pivotDF.show()pivotDF.createOrReplaceTempView("pivot")# pivot data per the
year to get average prices per stock per yearpivotDF \    .groupBy("stock") \
.pivot("year",[2012,2013]) \    .agg(func.avg("price")) \    .show()
```

This looks as follows in Jupyter:

```
from pyspark import SparkContext
from pyspark.sql import SparkSession
from pyspark.sql import functions as func

sc = SparkContext.getOrCreate()
spark = SparkSession(sc)
```

All the standard includes...

Summary

In this chapter, we got familiar with obtaining a `SparkContext`. We saw examples of using Hadoop MapReduce. We used SQL with Spark data. We combined data frames and operated on the resulting set. We imported JSON data and manipulated it with Spark. Lastly, we looked at using a pivot to gather information about a data frame.

In the next chapter, we will look at using R programming under Jupyter.

Chapter 5. R with Jupyter

In this chapter we will be using R coding within Jupyter. I think R is one of the primary languages expected to be used within Jupyter. The full extent of the language is available to Jupyter users.

How to set up R for Jupyter

In the past, it was necessary to install the separate components of Jupyter, Python, and so on to have a working system. With Continuum Analytics, the process of installing Jupyter and adding the R engine to the solution set for Jupyter is easy and works on both Windows and Mac.

Assuming you have installed conda already, we have one command to add support for R programming to Jupyter:

```
conda install -c r r-essentials
```

Note

At this point, when you start Jupyter, one of the kernels listed will now be `R`.

R data analysis of the 2016 US election demographics

To get a flavor of the resources available to R developers, we can look at the 2016 election data. In this case, I am drawing from Wikipedia (https://en.wikipedia.org/wiki/United_States_presidential_election,_2016), specifically the table named 2016 presidential vote by demographic subgroup. We have the following coding below.

Define a helper function so we can print out values easily. The new `printf` function takes any arguments passed `(...)` and passes them along to `sprintf`:

```
printf <- function(...)print(sprintf(...))
```

I have stored the separate demographic statistics into different **TSV** (**tab-separated value)** files, which can be read in using the following coding. For each table, we use the `read.csv` function and specify the field separator as a tab instead of the default comma. We then use the `head` function to display information about the data frame that was loaded:

```
age <- read.csv("Documents/B05238_05_age.tsv", sep="\t")head(age)education <-
read.csv("Documents/B05238_05_education.tsv", sep="\t")head(education)gender <-
read.csv("Documents/B05238_05_gender.tsv", sep="\t")ideology <-...
```

Analyzing 2016 voter registration and voting

Similarly, we can look at voter registration versus actual voting (using census data from https://www.census.gov/data/tables/time-series/demo/voting-and-registration/p20-580.html).

First, we load our dataset and display head information to visually check for accurate loading:

```
df <- read.csv("Documents/B05238_05_registration.csv")summary(df)
```

```
df <- read.csv("Documents/B05238_05_registration.csv")
summary(df)
```

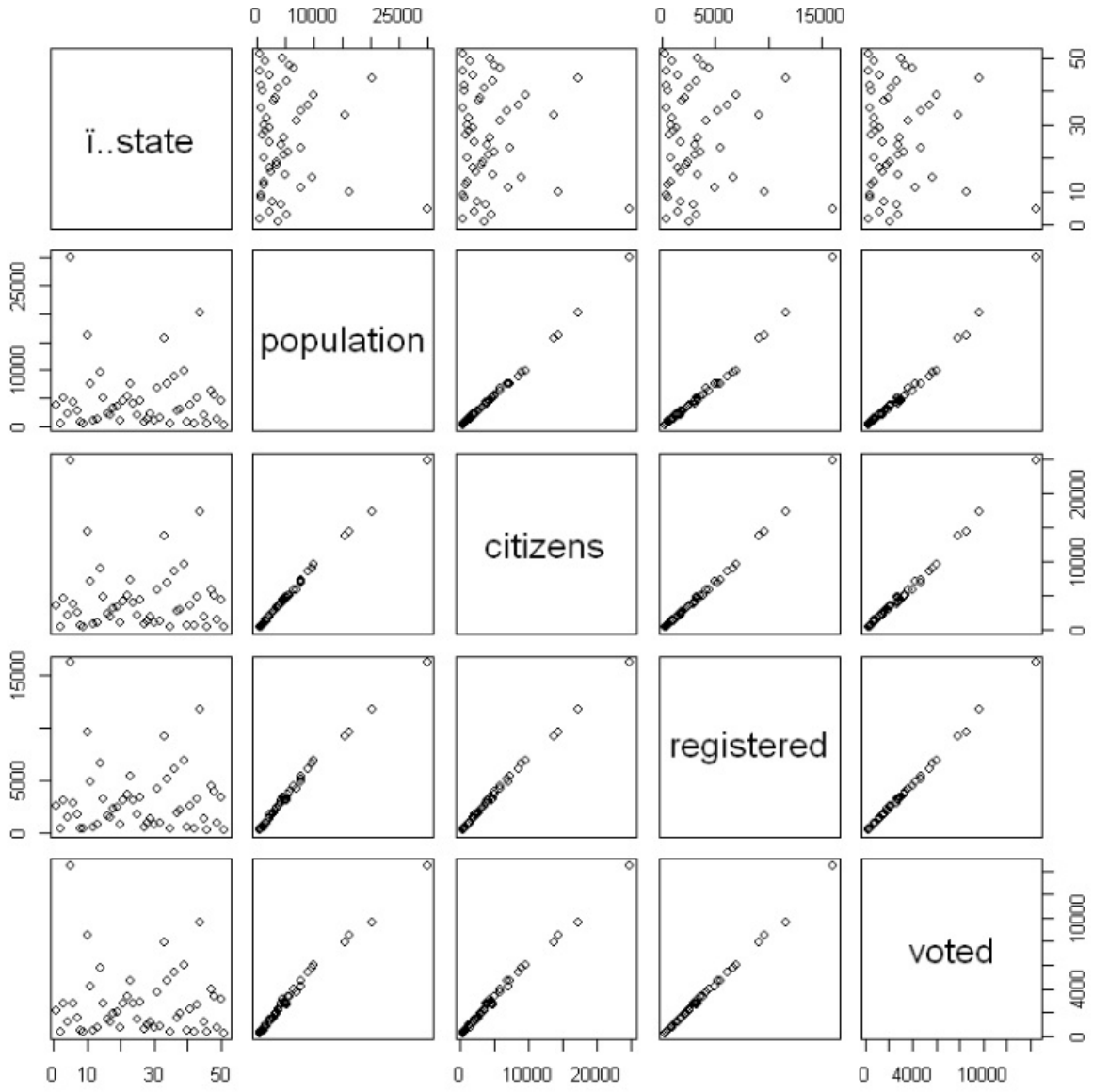
```
      ï..state     population       citizens       registered   
 ALABAMA   : 1   Min.   :  436   Min.   :  427   Min.   :  304.0  
 ALASKA    : 1   1st Qu.: 1316   1st Qu.: 1243   1st Qu.:  871.5  
 ARIZONA   : 1   Median : 3348   Median : 3246   Median : 2253.0  
 ARKANSAS  : 1   Mean   : 4814   Mean   : 4393   Mean   : 3090.1  
 CALIFORNIA: 1   3rd Qu.: 5483   3rd Qu.: 5036   3rd Qu.: 3783.0  
 COLORADO  : 1   Max.   :29894   Max.   :24890   Max.   :16096.0  
 (Other)   :45                                                    
     voted        
 Min.   :  277.0  
 1st Qu.:  738.5  
 Median : 1942.0  
 Mean   : 2696.8  
 3rd Qu.: 3348.5  
 Max.   :14416.0  
```

So, we have some registration and voting information by state. Use R to automatically plot all the data in *x* and *y* format using the `plot` command:

```
plot(df)
```

We are specifically looking at the relationship between registering to vote and actually voting. We can see in the following graphic that most of the data is highly correlated (as evidenced by the 45 degree angles of most of the relationships):

```
plot(df)
```



We can produce somewhat similar results using Python, but the graphic display is not even close.

Import all of the packages we are using for the example:

```
from numpy import corrcoef, sum, log, arange
from numpy.random import rand
from pylab import pcolor, show, colorbar, xticks, yticks
import pandas as pd
import matplotlib
from matplotlib import pyplot as plt...
```

Analyzing changes in college admissions

We can look at trends in college admissions acceptance rates over the last few years. For this analysis, I am using the data on https://www.ivywise.com/ivywise-knowledgebase/admission-statistics.

First, we read in our dataset and show the summary points, from head to validate:

```
df <- read.csv("Documents/acceptance-rates.csv")summary(df)head(df)
```

We see the summary data for school acceptance rates as follows:

```
df <- read.csv("Documents/acceptance-rates.csv")
summary(df)
head(df)
```

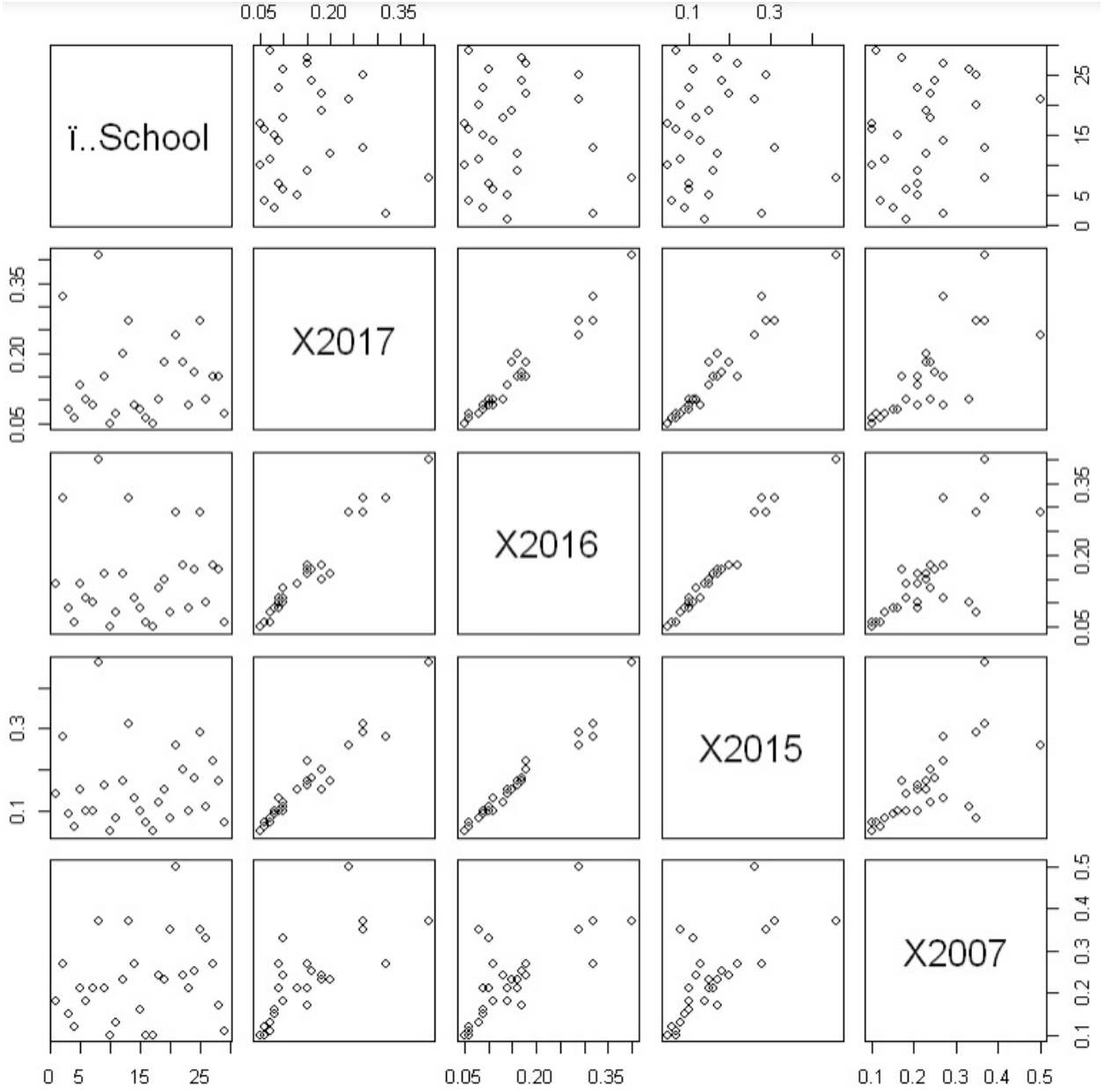
```
              ï..School      X2017           X2016           X2015
 Amherst College    : 1   Min.   :0.0500   Min.   :0.0500   Min.   :0.0500
 Boston College     : 1   1st Qu.:0.0800   1st Qu.:0.0900   1st Qu.:0.0900
 Brown University   : 1   Median :0.1000   Median :0.1300   Median :0.1300
 Columbia University: 1   Mean   :0.1444   Mean   :0.1493   Mean   :0.1534
 Cornell University : 1   3rd Qu.:0.1800   3rd Qu.:0.1700   3rd Qu.:0.1800
 Dartmouth College  : 1   Max.   :0.4100   Max.   :0.4000   Max.   :0.4600
 (Other)            :23   NA's   :2
     X2007
 Min.   :0.1000
 1st Qu.:0.1600
 Median :0.2100
 Mean   :0.2279
 3rd Qu.:0.2700
 Max.   :0.5000
```

ï..School	X2017	X2016	X2015	X2007
Amherst College	NA	0.14	0.14	0.18
Boston College	0.32	0.32	0.28	0.27
Brown University	0.08	0.09	0.09	0.15
Columbia University	0.06	0.06	0.06	0.12
Cornell University	0.13	0.14	0.15	0.21
Dartmouth College	0.10	0.11	0.10	0.18

It's interesting to note that the acceptance rate varies so widely, from a low of 5 percent to a high of 41 percent in 2017.

Let us look at the data plots, again, to validate that the data points are correct:

```
plot(df)
```



From the correlation graphics shown, it does not look like we can use the data points from 2007. The graphs show a big divergence between 2007 and the other years, whereas the other three have good correlations.

So, we have 3 consecutive years of data from 25 major US universities. We can convert the data into a time series using a few steps.

First, we create a vector of the average acceptance rates for these colleges over the years 2015-2017. We use the mean function to determine...

Predicting airplane arrival time

R has built-in functionality for splitting up a data frame between training and testing sets, building a model based on the training set, predicting results using the model and the testing set, and then visualizing how well the model is working.

For this example, I am using airline arrival and departure times versus scheduled arrival and departure times from http://stat-computing.org/dataexpo/2009/the-data.html for 2008. The dataset is distributed as a `.bz2` file that unpacks into a CSV file. I like this dataset, as the initial row count is over 7 million and it all works nicely in Jupyter.

We first read in the airplane data and display a summary. There are additional columns in the dataset that we are not using:

```
df <- read.csv("Documents/2008-airplane.csv")summary(df)...CRSElapsedTime      AirTime
ArrDelay          DepDelay       Min.   :-141.0   Min.   :   0     Min.   :-519.00   Min.
:-534.00   1st Qu.:  80.0   1st Qu.:  55     1st Qu.: -10.00   1st Qu.:  -4.00   Median :
110.0   Median :  86     Median :  -2.00   Median :  -1.00   Mean   : 128.9   Mean   : 104
Mean   :   8.17   Mean...
```

Summary

In this chapter, we first set up R as one of the engines available for a notebook. Then we used some rudimentary R to analyze voter demographics for the presidential election. We looked at voter registration versus actual voting. Next, we analyzed the trend in college admissions. Finally, we looked at using a predictive model to determine whether flights would be delayed or not.

In the next chapter, we will look into wrangling data in different ways under Jupyter.

Chapter 6. Data Wrangling

In this chapter, we look at data in several different forms and pry useful statistics. The tools to access the data have been well developed and allow for data to be missing headings or data points in some of the records.

Reading a CSV file

One of the standards for file formats is CSV. In this section, we will walk through the process of reading a CSV and adjusting the dataset to arrive at some conclusions about the data. The data I am using is from the Heating System Choice in California Houses dataset, found at https://vincentarelbundock.github.io/Rdatasets/datasets.html:

```
#read in the CSV file as available on the siteheating <-
read.csv(file="Documents/heating.csv", header=TRUE, sep=",")# make sure the data is laid out
the way we expecthead(heating)
```

X	idcase	depvar	ic.gc	ic.gr	ic.ec	ic.er	ic.hp	oc.gc	oc.gr	...	oc.hp	income
1	1	gc	866.00	962.64	859.90	995.76	1135.50	199.69	151.72	...	237.88	7
2	2	gc	727.93	758.89	796.82	894.69	968.90	168.66	168.66	...	199.19	5
3	3	gc	599.48	783.05	719.86	900.11	1048.30	165.58	137.80	...	171.47	4
4	4	er	835.17	793.06	761.25	831.04	1048.70	180.88	147.14	...	222.95	2
5	5	er	755.59	846.29	858.86	985.64	883.05	174.91	138.90	...	178.49	2
6	6	gc	666.11	841.71	693.74	862.56	859.18	135.67	140.97	...	209.27	6

The data appears to be as expected; however, a number of the columns have acronym names and are somewhat duplicated. Let us change the names of interest that we want to be more readable and remove the extras we are not going to use:

```
# change the column names to be more readablecolnames(heating)[colnames(heating)=="depvar"] <-
"system"colnames(heating)[colnames(heating)=="ic.gc"] <- "install_cost"colnames(heating)
[colnames(heating)=="oc.gc"] <- "annual_cost"colnames(heating)[colnames(heating)=="pb.gc"] <-
"ratio_annual_install"# remove columns which are not usedheating$idcase <- NULLheating$ic.gr
<- NULL
```

Reading another CSV file

We can look at another CSV in the same dataset to see what kind of issues we run across. Using the yearly batting records for all Major League Baseball players that we previously downloaded from the same site, we can use coding like the following to start analyzing the data:

```
players <- read.csv(file="Documents/baseball.csv", header=TRUE, sep=",")head(players)
```

This produces the following head display:

X	id	year	stint	team	lg	g	ab	r	h	...	rbi	sb	cs	bb	so	ibb	hbp	sh	sf	gidp
4	ansonca01	1871	1	RC1		25	120	29	39	...	16	6	2	2	1	NA	NA	NA	NA	NA
44	forceda01	1871	1	WS3		32	162	45	45	...	29	8	0	4	0	NA	NA	NA	NA	NA
68	mathebo01	1871	1	FW1		19	89	15	24	...	10	2	1	2	0	NA	NA	NA	NA	NA
99	startjo01	1871	1	NY2		33	161	35	58	...	34	4	2	3	0	NA	NA	NA	NA	NA
102	suttoez01	1871	1	CL1		29	128	35	45	...	23	3	1	1	0	NA	NA	NA	NA	NA
106	whitede01	1871	1	CL1		29	146	40	47	...	21	2	2	4	1	NA	NA	NA	NA	NA

There are many statistics for baseball players in this dataset. There are also many `NA` values. R is pretty good at ignoring `NA` values. Let us first look at the statistics for the data using:

```
summary(players)
```

This generates statistics on all the fields involved (there are several more that are not in this display):

```
       X                id              year          stint
 Min.   :    4   mcguide01:   31   Min.   :1871   Min.   :1.000
 1st Qu.:27080   henderi01:   29   1st Qu.:1937   1st Qu.:1.000
 Median :48997   newsobo01:   29   Median :1970   Median :1.000
 Mean   :46655   johnto01 :   28   Mean   :1961   Mean   :1.093
 3rd Qu.:65780   kaatji01 :   28   3rd Qu.:1988   3rd Qu.:1.000
 Max.   :89534   ansonca01:   27   Max.   :2007   Max.   :4.000
                 (Other)  :21527
      team          lg              g                ab              r
 CHN    : 1179     :   65   Min.   :  0.00   Min.   :  0.0   Min.   :  0.00
 NYA    : 1100   AA:  171   1st Qu.: 29.00   1st Qu.: 25.0   1st Qu.:  2.00
 SLN    : 1094   AL:10007   Median : 59.00   Median :131.0   Median : 15.00
 PHI    : 1070   FL:   37   Mean   : 72.82   Mean   :225.4   Mean   : 31.78
 CIN    : 1030   NL:11378   3rd Qu.:125.00   3rd Qu.:435.0   3rd Qu.: 58.00
 PIT    : 1020   PL:   32   Max.   :165.00   Max.   :705.0   Max.   :177.00
 (Other):15206   UA:    9
       h               X2b              X3b              hr
 Min.   :  0.00   Min.   : 0.00   Min.   : 0.000   Min.   : 0.000
 1st Qu.:  4.00   1st Qu.: 0.00   1st Qu.: 0.000   1st Qu.: 0.000
 Median : 32.00   Median : 5.00   Median : 1.000   Median : 1.000
 Mean   : 61.76   Mean   :10.45   Mean   : 2.194   Mean   : 5.234
 3rd Qu.:119.00   3rd Qu.:19.00   3rd Qu.: 3.000   3rd Qu.: 7.000
 Max.   :257.00   Max.   :64.00   Max.   :28.000   Max.   :73.000
```

A number of interesting points are visible in the preceding display that are worth noting:

- **We have about 30 data points per player**
- **It is interesting that the player data goes back to 1871**
- **There are about 1,000 data points per team**
- **American League and National League are clearly more popular**
- **The range of some of the data points is surprising:**
    - **At bats range from 0 to 700**
    - **Runs (r) range from 0 to 177**
    - **Hits (h) range...**

Manipulating data with dplyr

The `dplyr` package for R is described as a package providing a grammar for data manipulation. It has the entry points you would expect for wrangling your data frame in one package. We will use the `dplyr` package against the baseball player statistics we used earlier in this chapter.

We read in the player data and show the first few rows:

```
players <- read.csv(file="Documents/baseball.csv", header=TRUE, sep=",")
head(players)
```

X	id	year	stint	team	lg	g	ab	r	h	...	rbi	sb	cs	bb	so	ibb	hbp	sh	sf	gidp
4	ansonca01	1871	1	RC1		25	120	29	39	...	16	6	2	2	1	NA	NA	NA	NA	NA
44	forceda01	1871	1	WS3		32	162	45	45	...	29	8	0	4	0	NA	NA	NA	NA	NA
68	mathebo01	1871	1	FW1		19	89	15	24	...	10	2	1	2	0	NA	NA	NA	NA	NA
99	startjo01	1871	1	NY2		33	161	35	58	...	34	4	2	3	0	NA	NA	NA	NA	NA
102	suttoez01	1871	1	CL1		29	128	35	45	...	23	3	1	1	0	NA	NA	NA	NA	NA
106	whitede01	1871	1	CL1		29	146	40	47	...	21	2	2	4	1	NA	NA	NA	NA	NA

We will be using the `dplyr` package, so we need to pull the package into our notebook:

```
library(dplyr)
```

Converting a data frame to a dplyr table

The `dplyr` package has functions to convert your data object into a `dplyr` table. A `dplyr` table stores data in a compact format using much less memory. Most of the other `dplyr` functions can operate directly on the table as well.

We can convert our data frame to a table using:

```
playerst <- tbl_df(players)playerst
```

This results in a very similar display pattern:

X	id	year	stint	team	lg	g	ab	r	h	...	rbi	sb	cs	bb	so	ibb	hbp	sh	sf	gidp
4	ansonca01	1871	1	RC1		25	120	29	39	...	16	6	2	2	1	NA	NA	NA	NA	NA
44	forceda01	1871	1	WS3		32	162	45	45	...	29	8	0	4	0	NA	NA	NA	NA	NA
68	mathebo01	1871	1	FW1		19	89	15	24	...	10	2	1	2	0	NA	NA	NA	NA	NA
99	startjo01	1871	1	NY2		33	161	35	58	...	34	4	2	3	0	NA	NA	NA	NA	NA
102	suttoez01	1871	1	CL1		29	128	35	45	...	23	3	1	1	0	NA	NA	NA	NA	NA
106	whitede01	1871	1	CL1		29	146	40	47	...	21	2	2	4	1	NA	NA	NA	NA	NA
113	yorkto01	1871	1	TRO		29	145	36	37	...	23	2	2	9	1	NA	NA	NA	NA	NA
121	ansonca01	1872	1	PH1		46	217	60	90	...	50	6	6	16	3	NA	NA	NA	NA	NA

Getting a quick overview of the data value ranges

Another function available in `dplyr` is the `glimpse()` function. It takes every column and displays the range of values present for that variable. We use...

Sampling a dataset

The `dplyr` package has a function to gather a sample from your dataset, `sample()`. You pass in the dataset to operate against and how many samples you want drawn, `sample_n()`, and the fraction percentage, `sample_frac()`, as in this example:

```
data <- sample_n(players, 30)glimpse(data)
```

We see the results as shown in the following screenshot:

```
Observations: 30
Variables: 23
$ X     <int> 3531, 49852, 83330, 44753, 29473, 43131, 48665, 62357, 74109,...
$ id    <fctr> mathebo01, reedro01, guarded01, jenkife01, troutdi01, burges...
$ year  <int> 1887, 1971, 2002, 1965, 1941, 1963, 1970, 1985, 1995, 1930, 1...
$ stint <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
$ team  <fctr> PH4, ATL, MIN, PHI, DET, PIT, SDN, CAL, CHN, CLE, SLA, DET, ...
$ lg    <fctr> AA, NL, AL, NL, AL, NL, NL, AL, NL, AL, AL, AL, NL, NL, AL, ...
$ g     <int> 7, 32, 4, 7, 40, 91, 139, 150, 50, 86, 110, 138, 65, 35, 139,...
$ ab    <int> 25, 74, 0, 1, 50, 264, 534, 520, 162, 265, 390, 492, 176, 16,...
$ r     <int> 5, 3, 0, 0, 5, 20, 79, 80, 24, 30, 56, 75, 19, 0, 71, 64, 80,...
$ h     <int> 5, 11, 0, 0, 9, 74, 156, 137, 31, 78, 93, 136, 43, 1, 138, 86...
$ X2b   <int> 0, 0, 0, 0, 1, 10, 34, 23, 2, 23, 13, 23, 10, 0, 12, 10, 29, ...
$ X3b   <int> 0, 0, 0, 0, 1, 1, 1, 1, 0, 2, 6, 3, 2, 0, 7, 3, 4, 4, 0, 0, 5...
$ hr    <int> 0, 0, 0, 0, 0, 6, 23, 20, 6, 2, 6, 24, 1, 0, 3, 0, 11, 0, 1, ...
$ rbi   <int> 0, 1, 0, 0, 5, 37, 89, 85, 14, 37, 58, 82, 20, 0, 58, 36, 91,...
$ sb    <int> 0, 0, 0, 0, 0, 0, 5, 5, 0, 2, 0, 3, 1, 0, 7, NA, 9, 10, 1, 0,...
$ cs    <int> NA, 0, 0, 0, 0, 1, 3, 3, 0, 3, 5, 2, 0, NA, 7, NA, 5, NA, 0, ...
$ bb    <int> 4, 0, 0, 0, 4, 24, 34, 78, 36, 18, 24, 76, 15, 0, 61, 9, 91, ...
$ so    <int> NA, 20, 0, 1, 10, 14, 78, 61, 51, 17, 37, 84, 18, 5, 65, 28, ...
$ ibb   <int> NA, 0, 0, 0, NA, 8, 8, 3, 1, NA, NA, 7, 3, NA, 0, NA, 6, NA, ...
$ hbp   <int> 0, 0, 0, 0, 0, 1, 0, 13, 1, 1, 1, 5, 0, 0, 1, NA, 2, 2, 2, 0,...
$ sh    <int> NA, 6, 0, 0, 0, 0, 3, 5, 0, 4, 3, 4, 1, 1, 13, NA, 1, 5, 0, 2...
$ sf    <int> NA, 0, 0, 0, NA, 4, 6, 4, 1, NA, NA, 3, 0, NA, 8, NA, 12, NA,...
$ gidp  <int> NA, 0, 0, 0, 1, 12, 14, 12, 8, NA, 7, 6, 2, 0, 9, NA, 11, NA,...
```

Note that there are 30 observations in the results set, as requested.

Filtering rows in a data frame

Another function we can use is the `filter` function. The `filter` function takes a data frame as an argument and a filtering statement. The function passes over each row of the data frame and returns those rows that meet the filtering statement:

```
#filter only players with over 200 hits in a season
over200 <- filter(players, h > 200)
head(over200)
nrow(over200)
```

X	id	year	stint	team	lg	g	ab	r	h	...	rbi	sb	cs	bb	so	ibb	hbp	sh	sf	gidp
3454	brownpe01	1887	1	LS2	AA	134	547	137	220	...	118	103	NA	55	NA	NA	8	NA	NA	NA
3765	thompsa01	1887	1	DTN	NL	127	545	118	203	...	166	22	NA	32	19	NA	9	NA	NA	NA
4370	glassja01	1889	1	IN3	NL	134	582	128	205	...	85	57	NA	31	10	NA	5	NA	NA	NA
5832	delahed01	1893	1	PHI	NL	132	595	145	219	...	146	37	NA	47	20	NA	10	NA	NA	NA
5841	duffyhu01	1893	1	BSN	NL	131	560	147	203	...	118	44	NA	50	13	NA	1	NA	NA	NA
6023	thompsa01	1893	1	PHI	NL	131	600	130	222	...	126	18	NA	50	17	NA	6	NA	NA	NA

274

it looks like many players were capable of 200 hits a season. How about if we look at those players that could also get over 40 home runs in a season?

```
over200and40hr <- filter(players, h > 200 & hr > 40)
head(over200and40hr)
nrow(over200and40hr)
```

X	id	year	stint	team	lg	g	ab	r	h	...	rbi	sb	cs	bb	so	ibb	hbp	sh	sf	gidp
18834	ruthba01	1921	1	NYA	AL	152	540	177	204	...	171	17	13	145	81	NA	4	4	NA	NA
19528	hornsro01	1922	1	SLN	NL	154	623	141	250	...	152	17	12	65	50	NA	1	15	NA	NA
19883	ruthba01	1923	1	NYA	AL	152	522	151	205	...	131	17	21	170	93	NA	4	3	NA	NA
21925	gehrilo01	1927	1	NYA	AL	155	584	149	218	...	175	10	8	109	84	NA	3	21	NA	NA
23312	kleinch01	1929	1	PHI	NL	149	616	126	219	...	145	5	NA	54	61	NA	0	9	NA	NA
23524	gehrilo01	1930	1	NYA	AL	154	581	143	220	...	174	12	14	101	63	NA	3	18	NA	NA

16

It's a very small list. I know that player names are somewhat mangled, but you can recognize a...

Tidying up data with tidyr

The `tidyr` package is available to clean up/tidy your dataset. The use of `tidyr` is to rearrange your data so that:

- **Each column is a variable**
- **Each row is an observation**

When your data is arranged in this manner, it becomes much easier to analyze. There are many datasets published that mix columns and rows with values. You then must adjust them accordingly if you use the data in situ.

`tidyr` provides three functions for cleaning up your data:

- `gather`
- `separate`
- `spread`

The `gather()` function takes your data and arranges the data into key-value pairs, much like the Hadoop database model. Let's use the standard example of stock prices for a date using the following:

```
library(tidyr)
stocks <- data_frame(
  time = as.Date('2017-08-05') + 0:9,
  X = rnorm(10, 20, 1), #how many numbers, mean, std dev
  Y = rnorm(10, 20, 2),
  Z = rnorm(10, 20, 4)
)
```

This will generate data that looks like this:

time	X	Y	Z
2017-08-05	20.12058	20.56542	17.55471
2017-08-06	20.51702	18.40603	16.00274
2017-08-07	19.90123	20.34891	20.20725
2017-08-08	19.95212	21.27073	15.75052
2017-08-09	20.18198	18.75737	23.01953
2017-08-10	18.08772	18.10624	14.54853

Every row has a timestamp and the prices of the three stocks at that time.

We first use `gather()` to split out key-value pairs for the stocks. The `gather()` function is called with the data frame that it will work with, the output column...

Summary

In this chapter, we read in CSV files and performed a quick analysis of the data, including visualizations to help understand the data. Next, we considered some of the functions available in the `dplyr` package, including drawing a glimpse of the ranges of the data items, sampling a dataset, filtering out data, adding columns using mutate, and producing a summary. While doing so, we also started to use piping to more easily transfer the results of one operation into another operation. Lastly, we looked into the `tidyr` package to clean or tidy up our data into distinct columns and observations using the associated gather, separate, and spread functions.

In the next chapter, we will look at producing a dashboard under Jupyter.

Chapter 7. Jupyter Dashboards

A dashboard is a mechanism to put together multiple displays into one for use in a presentation. For example, you can take 3 graphical displays of production lines drawn from different data and display them on screen in one frame or dashboard. With Jupyter we can draw upon many mechanisms for retrieving and visualizing data which can then be put together into a single presentation.

Visualizing glyph ready data

A glyph is a symbol. In this section, we are looking to display glyphs at different points in a graph rather than the standard dot as the glyph should provide more visual information to the viewer. Often there is an attribute about a data point that can be used to turn the data point into a useful glyph, as we will see in the following examples.

The `ggplot2` package is useful for visualizing data in a variety of ways. `ggplot` is described as a plotting system for R. We will look at an example that displays volcano data points across the globe. I used the information from the National Center for Environmental Information at https://www.ngdc.noaa.gov/nndc. I selected volcano information post-1964.

This generated a set of data that I copied into a local CSV file:

```
#read in the CSV file as available as described previouslyvolcanoes =
read.csv("volcanoes.csv")head(volcanoes)
```

Number	Volcano.Name	Country	Region	Latitude	Longitude	Elev	Type
1402-08=	Acatenango	Guatemala	Guatemala	14.501	-90.876	3976	Stratovolcano
0803-17=	Adatara	Japan	Honshu-Japan	37.620	140.280	1718	Stratovolcano
0604-02=	Agung	Indonesia	Lesser Sunda Is	-8.342	115.508	3142	Stratovolcano
1000-123	Akademia Nauk	Russia	Kamchatka	53.980	159.450	1180	Stratovolcano
0805-07=	Akan	Japan	Hokkaido-Japan	43.380	144.020	1499	Caldera
0803-23=	Akita-Komaga-take	Japan	Honshu-Japan	39.750	140.800	1637	Stratovolcano

If we just plot out the points on a world map we can see where the volcanoes are located. We are using the `mapdata` package for the map:

Note

The latest R syntax requires specifying the location of the repository (or mirror) to find the package...

Publishing a notebook

You can publish a notebook/dashboard using markdown. Markdown involves adding annotations to cells in your notebook that are interpreted by Jupyter and converted into the more standard HTML representations that you see in other published materials.

In all cases, we create the cell with the markdown type. We then enter the syntax for markdown in the cell. Once we run markdown cells the display of the cell changes to the effective markdown representation. You should also note there is no line number designation for markdown cells, as there is no code executing in markdown cells.

Font markdown

You can adjust font style information using `italic` and `bold` HTML notations. For example, if we have the code format of a cell as follows. You can use markdown that has markdown tags for italics (`<i>`) and bold (`<b>`):

```
You can use markdown that has
<i>italic</i> and <b>bold</b> text
```

When we run the cell we see the effective markdown as:

> You can use markdown that has *italic* and **bold** text

List markdown

We can use lists such as the following, where we start an un-numbered list (we could have used `nl` for a numbered list) with two list items enclosed with the list item (`<li>`) tag:

```
In [ ]: <ul>
            <li>list item 1</li>
            <li>list item 2</li>
        </ul>
```

When we run this cell the resulting markdown is displayed:

- list item 1
- list item 2

Heading...

Creating a Shiny dashboard

Shiny is a web application framework for R. It does not require the user to code HTML. There are normally two sets of code: the server and the user interface (UI). Both sets of code work on top of a Shiny server. A Shiny server can reside on one of your machines or in the cloud (via several hosting companies).

The Shiny server code set deals with accessing data, computing results, obtaining direction from the user, and interacting with other server code set to change results. The UI code set deals with layout of the presentation.

For example, if you had an application that produced a histogram of data the server set would obtain the data and produce results, display the results, and interact with the user to change the result—for example, it might change the number of buckets or range of data being displayed. The UI code set would strictly be concerned with layout.

Shiny code does not run under Jupyter. You can develop the coding using RStudio. RStudio is an integrated development environment (IDE) for developing R scripts.

With RStudio you can develop the code sets for the `server.R` and `ui.R` components...

Building standalone dashboards

Using Node.js, developers have come up with a way to host your dashboard/notebook without Jupyter on jupyter-dashboard-server.

Installation requires installing Node.js (as the server is written in Node.js). This is a larger installation set.

Once you have Node.js installed, one of the tools installed is npm-node product manager. You can use npm to install the dashboard server with the following command:

```
npm install -g jupyter-dashboards-server
```

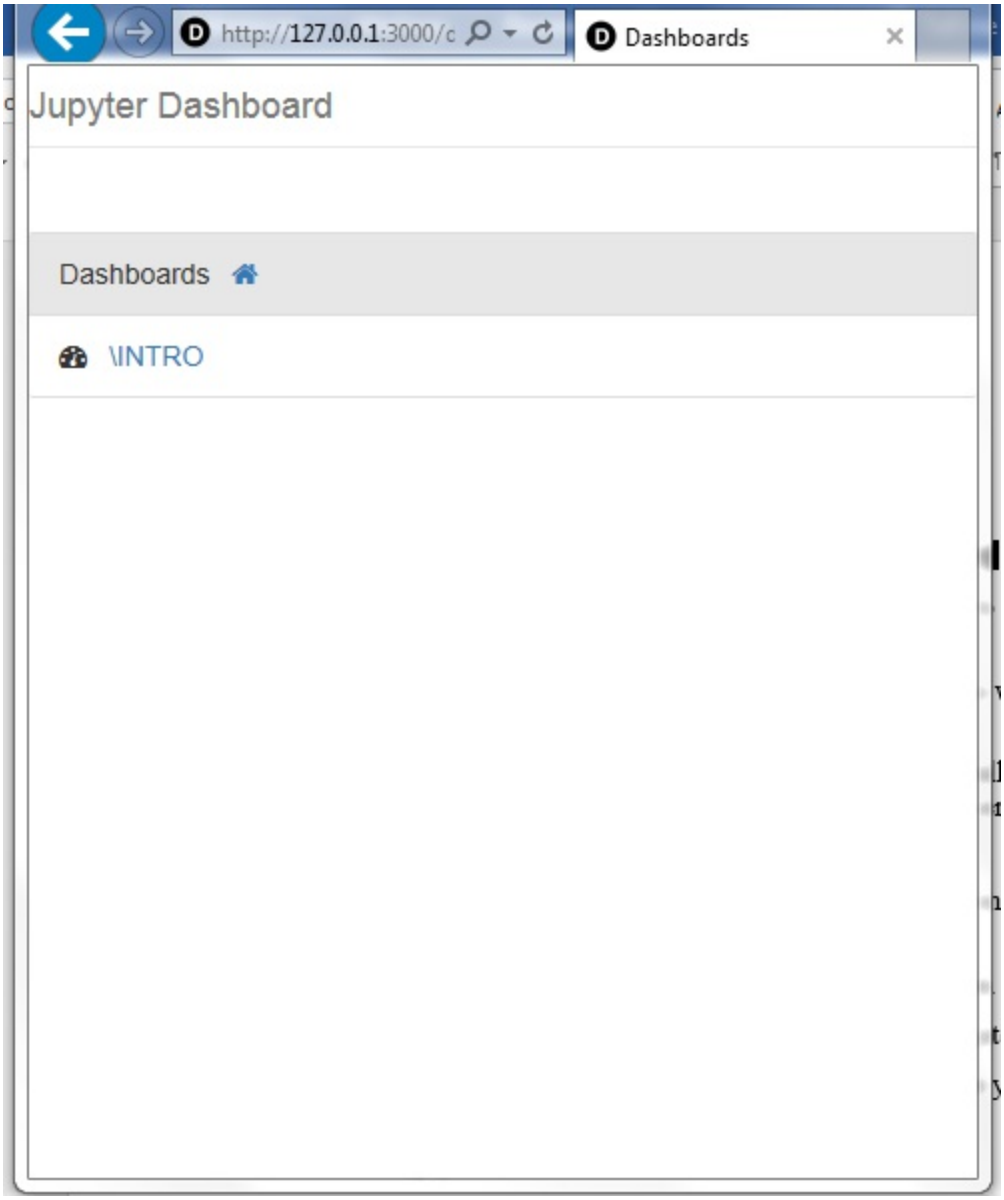
Once installed you can run the server with the following command:

```
C:\Users\Dan>jupyter-dashboards-server --KERNEL_GATEWAY_URL=http://my.gateway.com
```

`mygateway.com` **is a dummy. You would use your gateway server (if needed). At this point the server is running on the environment you mentioned and will output a few lines:**

```
Using generated SESSION_SECRET_TOKENJupyter dashboard server listening on 127.0.0.1:3000
```

You can open a browser to the URL (`http://127.0.0.1:3000/dashboards`) and see what the server console looks like:



As for developing a dashboard you can host on the server, we need to install more:

```
conda install jupyter_dashboards -c conda-forge
```

Then enable the extension...

Summary

In this chapter, we visualized data graphically using glyphs to emphasize important aspects of the data. We used markdown to annotate a notebook page. We used Shiny to generate an interactive application. And we saw a way to host notebooks outside of Jupyter.

In the next chapter, we will look at statistical modeling under Jupyter.

Chapter 8. Statistical Modeling

In this chapter we are taking raw data and attempting to interpret the information by building a statistical model. Once we have a model built then it is usually easier to see commonalities. We can determine trends as well.

Converting JSON to CSV

For this chapter, we will be using the Yelp data available from the challenge at https://www.yelp.com/dataset/challenge. This section uses the dataset from *round 9* of the challenge. For background, Yelp is a site for rating different products and services where Yelp publishes the ratings to users.

The dataset file is a very large (a few gigabytes) amount of ratings. There are several sets of rating information in the download-for business ratings, reviews, tips (as in this would be a nice place to visit), and a user set. We are interested in the review data.

When dealing with such large files it may be useful to find and use a large file editor so you can poke into the data file. On Windows, most of the standard editors are limited to a few megabytes. I used the Large Text File Viewer program to open these JSON files.

All of the files are in JSON format. JSON is a human readable format with structured elements—for example, a city object containing street objects. While it is convenient to read JSON the format is clumsy when dealing with large numbers of elements. In the `reviews` file there are a few million...

Evaluating Yelp reviews

We read in the processed Yelp reviews using this script and print out some statistics of the data:

```
reviews <- read.csv("c:/Users/Dan/yelp_academic_dataset_review.csv")
```

I usually take a look at some of the data once loaded to visually check that things are working as expected. We can do this with a `head()` function call:

```
head(reviews)
```

funny	user_id	review_id	business_id	stars	date	useful	type	cool
0	KpkOkG6RIf4Ra25Lhhxf1A	NxL8SIC5yqOdnlXCg18IBg	2aFiy99vNLklCx3T_tGS9A	5	2011-10-10	0	review	0
0	bQ7fQq1otn9hKX-gXRsrgA	pXbbIgOXvLuTi_SPs1hQEQ	2aFiy99vNLklCx3T_tGS9A	5	2010-12-29	1	review	0
0	r1NUhdNmL6yU9Bn-Yx6FTw	wslW2Lu4NYylb1jEapAGsw	2aFiy99vNLklCx3T_tGS9A	5	2011-04-29	0	review	0
0	aW3ix1KNZAvoM8q-WghA3Q	GP6YEearUWrzPtQYSF1vVg	2LfluF3_sX6uwe-IR-P0jQ	5	2014-07-14	0	review	1
0	YOo-Cip8HqvKp_p9nEGphw	25RlYGq2s5qShi-pn3ufVA	2LfluF3_sX6uwe-IR-P0jQ	4	2014-01-15	0	review	0
0	bgl3j8yJcRO-00NkUYsXGQ	Uf1Ki1yyH_JDKhLvn2e4FQ	2LfluF3_sX6uwe-IR-P0jQ	5	2013-04-28	2	review	1

Summary data

All of the columns appear to be correctly loading. Now, we can look at summary statistics for the data:

```
summary(reviews)
```

```
     funny                          user_id
 Min.   :  0.0000   CxDOIDnH8gp9KXzpBHJYXw:   3327
 1st Qu.:  0.0000   bLbSNkLggFnqwNNzzq-Ijw:   1795
 Median :  0.0000   PKEzKWv_FktMm2mGPjwd0Q:   1509
 Mean   :  0.4195   QJI9OSEn6ujRCtrX06vs1w:   1316
 3rd Qu.:  0.0000   DK57YibC5ShBmqQl97CKog:   1266
 Max.   :632.0000   d_TBs6J3twMy9GChqUEXkg:   1091
                    (Other)               :4142846
                 review_id                        business_id
 ----X0BIDP9tA49U3RvdSQ:      1   4JNXUYY8wbaaDmk3BPzlWw:   6414
 ---0hl58W-sjVTKi5LghGw:      1   RESDUcs7fIiihp38-d6_6g:   5715
 ---3OXpexMp0oAg77xWfYA:      1   K7lWdNUhCbcnEvI0NhGewg:   5216
 ---65iIIGzHj96QnOh89EQ:      1   cYwJA2A6I12KNkm2rtXd5g:   5116
 ---7WhU-FtzSUOje87Y4uw:      1   DkYS3arLOhA8si5uUEmHOw:   4655
 ---94vtJ_5o_nikEs6hUjg:      1   hihud--QRriCYZw1zZvW4g:   4120
 (Other)               :4153144   (Other)               :4121914
     stars               date             useful             type
 Min.   :1.000   2016-02-15:   3632   Min.   :   0.000   review:4153150
 1st Qu.:3.000   2017-01-15:   3600   1st Qu.:   0.000
 Median :4.000   2016-08-14:   3593   Median :   0.000
 Mean   :3.723   2016-04-03:   3588   Mean   :   1.008
 3rd Qu.:5.000   2016-07-31:   3566   3rd Qu.:   1.000
 Max.   :5.000   2016-07-10:   3559   Max.   :1125.000
                 (Other)   :4131612
      cool
 Min.   :  0.0000
 1st Qu.:  0.0000
 Median :  0.0000
 Mean   :  0.5262
 3rd Qu.:  0.0000
 Max.   :513.0000
```

There are several points in the summary worth noting:

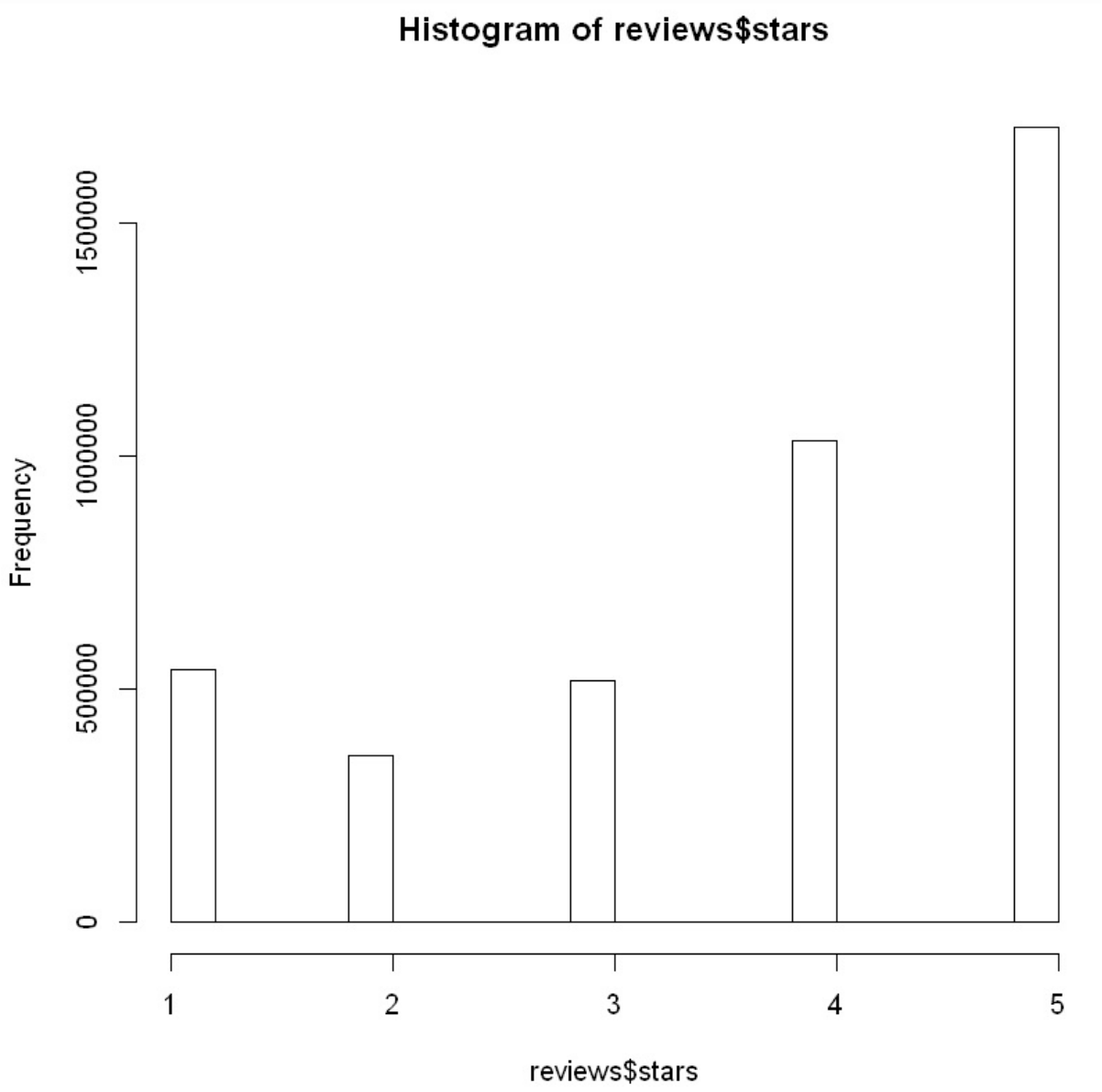
- **Some of the data points I had assumed would be just `TRUE`/`FALSE`, `0`/`1` have ranges instead; for example, `funny` has a max value over 600; `useful` has a max 1100, `cool` has 500.**
- **All of the IDs (users, businesses) have been mangled. We could use the user file and the business file to come up with exact references.**
- **Star ratings are `1-5`, as expected. However, the mean and median are about a `4`, which I take as many people only take the time to write good reviews.**

Review spread

We can get an idea of the spread of the reviews using a simple histogram using:

```
hist(reviews$stars)
```

Which generates the histogram inline as:


Histogram of reviews$stars
Frequency
0
500000
1000000
1500000
1
2
3
4
5
reviews$stars

Again,...

Using Python to compare ratings

In the previous examples we used R to work through data frames that were built from converted JSON to CSV files. If we were to use the Yelp businesses rating file we could use Python directly, as it is much smaller and produces similar results.

In this example, we gather cuisines from the Yelp file based on whether the business category includes restaurants. We accumulate the ratings for all cuisines and then produce averages for each.

We read in the JSON file into separate lines and convert each line into a Python object:

Note

We convert each line to Unicode with the `errors=ignore` option. This is due to many erroneous characters present in the data file.

```
import json#filein = 'c:/Users/Dan/business.json'filein =
'c:/Users/Dan/yelp_academic_dataset_business.json'lines = list(open(filein))
```

We use a dictionary for the ratings for a cuisine. The key of the dictionary is the name of the cuisine. The value of the dictionary is a list of ratings for that cuisine:

```
ratings = {}for line in lines:    line = unicode(line, errors='ignore')    obj =
json.loads(line)    if obj['categories'] == None:        continue   ...
```

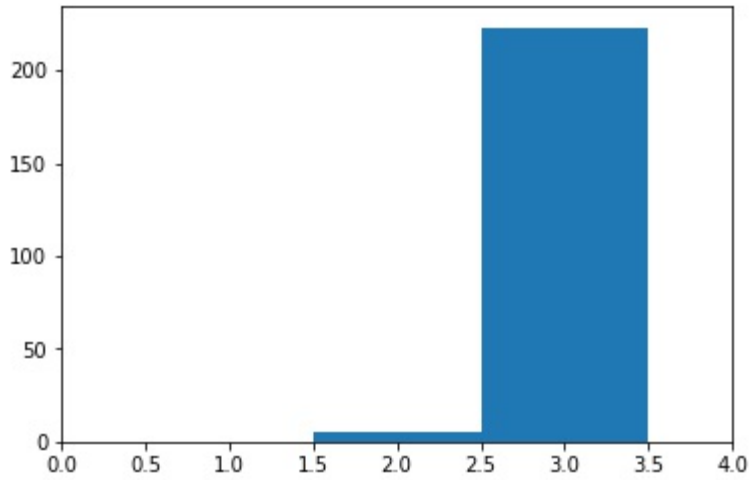
Visualizing average ratings by cuisine

Now that we have the cuisine averages computed, we can display them in a histogram to get an idea of their spread. We first convert the dictionary to a data frame. Then plot the `Rating` column of the data frame into a histogram:

Note

We are using five bins to correspond to the five possible ratings.

```
import pandas as pdimport numpy as npdf = pd.DataFrame(columns=['Cuisine', 'Rating'])for
cuisine in cuisines:    df.loc[len(df)]=[cuisine, cuisines[cuisine]]hist, bin_edges =
np.histogram(df['Rating'], bins=range(5))import matplotlib.pyplot as
pltplt.bar(bin_edges[:-1], hist, width = 1)plt.xlim(min(bin_edges), max(bin_edges))plt.show()
```



Again, we see a clear mark towards high average values. I had tried to get a better gradient on the data display to no avail.

Arbitrary search of ratings

Since we have the data in an easily loadable format we can search for arbitrary conditions, such as `Personal Chefs` that allow dogs-maybe they will custom cook for your dog.

We could use a script as follows: for line in lines:

```
    line = unicode(line, errors='ignore')    obj = json.loads(line)    if obj['categories'] ==
None:        continue    if 'Personal Chefs' in obj['categories']:        if obj['attributes']
== None:            continue        for attr in obj['attributes']:            print (attr)
```

Where we do something useful with the items that filter out. This script would display the attributes only for `Personal Chefs`. As can be seen in the following display:

```
BusinessAcceptsCreditCards: True
BikeParking: True
BusinessAcceptsCreditCards: True
BusinessParking: {'garage': False, 'street': True,
Caters: True
GoodForMeal: {'dessert': False, 'latenight': False,
RestaurantsDelivery: False
RestaurantsPriceRange2: 1
RestaurantsTakeOut: True
WheelchairAccessible: True
BusinessAcceptsCreditCards: True
RestaurantsPriceRange2: 2
RestaurantsPriceRange2: 4
BusinessAcceptsCreditCards: True
RestaurantsPriceRange2: 2
BusinessAcceptsCreditCards: True
BusinessAcceptsCreditCards: True
RestaurantsPriceRange2: 2
```

We could just as easily performed some calculation or other manipulation to narrow down and focus on a very specific portion of the data easily.

Determining relationships between number of ratings and ratings

Given the preceding results it appears that people mostly only vote in a positive manner. We can look to see if there is a relationship between how many votes a company has received and their rating.

First, we accumulate the dataset using the following script, extracting the number of votes and rating for each firm:

```
#determine relationship between number of reviews and star ratingimport pandas as pdfrom
pandas import DataFrame as df  import numpy as np  dfr2 = pd.DataFrame(columns=['reviews',
'rating'])mynparray = dfr2.valuesfor line in lines:    line = unicode(line, errors='ignore')
obj = json.loads(line)    reviews = int(obj['review_count'])    rating = float(obj['stars'])
arow = [reviews,rating]    mynparray = np.vstack((mynparray,arow)) dfr2 = df(mynparray)print
(len(dfr2))
```

This coding just builds the data frame with our two variables. We are using NumPy as it more easily adds a row to a data frame. Once we are done with all records we convert the NumPy data frame back to a pandas data frame.

The column names have been lost in the translation, so we put those...

Chapter 9. Machine Learning Using Jupyter

In this chapter, we will use several algorithms for machine learning under Jupyter. We have coding in both R and Python to portray the breadth of options available to the Jupyter developer.

Naive Bayes

Naive Bayes is an algorithm that uses probability to classify the data according to Bayes theorem for strong independence of the features. Bayes theorem estimates the probability of an event based on prior conditions. So, overall, we use a set of feature values to estimate a value assuming the same conditions hold true when those features have similar values.

Naive Bayes using R

Our first implementation of naive Bayes uses the R programming language. The R implementation of the algorithm is encoded in the `e1071` library. `e1071` appears to have been the department identifier at the school where the package was developed.

We first install the package, and load the library:

```
#install.packages("e1071", repos="http://cran.r-project.org")
library(e1071)
library(caret)
set.seed(7317)
data(iris)
```

Some notes on these steps:

- **The `install.packages` call is commented out as we don't want to run this every time we run the script.**
- **`e1071` is the naive Bayes algorithm package.**
- **The `caret` package contains a method to partition a dataset randomly.**
- **We set the `seed` so as to be able to reproduce the results.**
- **We are using the `iris` dataset for this...**

Nearest neighbor estimator

Using nearest neighbor, we have an unclassified object and a set of objects that are classified. We then take the attributes of the unclassified object, compare against the known classifications in place, and select the class that is closest to our unknown. The comparison distances resolve to Euclidean geometry computing the distances between two points (where known attributes fall in comparison to the unknown's attributes).

Nearest neighbor using R

For this example, we are using the housing data from `ics.edu`. First, we load the data and assign column names:

```
housing <- read.table("http://archive.ics.uci.edu/ml/machine-learning-
databases/housing/housing.data")
colnames(housing) <- c("CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD", "TAX",
"PRATIO", "B", "LSTAT", "MDEV")
summary(housing)
```

We reorder the data so the key (the housing price `MDEV`) is in ascending order:

```
housing <- housing[order(housing$MDEV),]
```

Now, we can split the data into a training set and a test set:

```
#install.packages("caret")
library(caret)
set.seed(5557)
indices <- createDataPartition(housing$MDEV, p=0.75, list=FALSE)...
```

Decision trees

In this section, we will use decision trees to predict values. A decision tree has a logical flow where the user makes decisions based on attributes following the tree down to a root level where a classification is then provided.

For this example, we are using automobile characteristics, such as vehicle weight, to determine whether the vehicle will produce good mileage. The information is extracted from the page at https://alliance.seas.upenn.edu/~cis520/wiki/index.php?n=Lectures.DecisionTrees. I copied the data out to Excel and then wrote it as a CSV for use in this example.

Decision trees in R

We load the libraries to use `rpart` and `caret`. `rpart` has the decision tree modeling package. `caret` has the data partition function:

```
library(rpart)
library(caret)
set.seed(3277)
```

We load in our `mpg` dataset and split it into a training and testing set:

```
carmpg <- read.csv("car-mpg.csv")
indices <- createDataPartition(carmpg$mpg, p=0.75, list=FALSE)
training <- carmpg[indices,]
testing <- carmpg[-indices,]
nrow(training)
nrow(testing)
33
9
```

We develop a model to predict `mpg` acceptability based on the other factors:

```
fit <-...
```

Neural networks

We can model the housing data as a neural network where the different data elements are inputs into the system and the output of the network is the house price. With a neural net we end up with a graphical model that provides the factors to apply to each input in order to arrive at our housing price.

Neural networks in R

There is a neural network package available in R. We load that in:

```
#install.packages('neuralnet', repos="http://cran.r-project.org")
library("neuralnet")
```

Load in the housing data:

```
filename = "http://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data"
housing <- read.table(filename)
colnames(housing) <- c("CRIM", "ZN", "INDUS", "CHAS", "NOX",
                       "RM", "AGE", "DIS", "RAD", "TAX", "PRATIO",
                       "B", "LSTAT", "MDEV")
```

Split up the housing data into training and test sets (we have seen this coding in prior examples):

```
housing <- housing[order(housing$MDEV),]
#install.packages("caret")
library(caret)
set.seed(5557)
indices <- createDataPartition(housing$MDEV, p=0.75, list=FALSE)
training <- housing[indices,]
testing <- housing[-indices,]...
```

Random forests

The random forests algorithm attempts a number of random decision trees and provides the tree that works best within the parameters used to drive the model.

Random forests in R

With R we include the packages we are going to use:

```
install.packages("randomForest", repos="http://cran.r-project.org")
library(randomForest)
```

Load the data:

```
filename = "http://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data"
housing <- read.table(filename)
colnames(housing) <- c("CRIM", "ZN", "INDUS", "CHAS", "NOX",
                       "RM", "AGE", "DIS", "RAD", "TAX", "PRATIO",
                       "B", "LSTAT", "MDEV")
```

Split it up:

```
housing <- housing[order(housing$MDEV),]
#install.packages("caret")
library(caret)
set.seed(5557)
indices <- createDataPartition(housing$MDEV, p=0.75, list=FALSE)
training <- housing[indices,]
testing <- housing[-indices,]
nrow(training)
nrow(testing)
```

Calculate our model:

```
forestFit <- randomForest(MDEV ~ CRIM + ZN + INDUS + CHAS + NOX
                  + RM + AGE + DIS + RAD + TAX + PRATIO
                  + B + LSTAT, data=training)
forestFit
Call:
...
```

Summary

In this chapter, we used several machine learning algorithms, some of them in R and Python to compare and contrast. We used naive Bayes to determine how the data might be used. We applied nearest neighbor in a couple of different ways to see our results. We used decision trees to come up with an algorithm for predicting. We tried to use neural network to explain housing prices. Finally, we used the random forest algorithm to do the same—with the best results! In the next chapter, we will look at optimizing Jupyter notebooks.

Chapter 10. Optimizing Jupyter Notebooks

Before a Jupyter Notebook is developed you should confront optimizations that should occur before the public starts their access. Optimizations cover a gamut of options running from language-specific issues (use best practice R coding style) to deploying your notebook in a highly available environment.

Deploying notebooks

A Jupyter Notebook is a website. You could host a website on the computer that you are using to display this document. There may be a machine available in your department that is in use as a web server.

If you were to deploy on a local machine you would have a single user website where additional users would be blocked from access or would collide with each other. The first step towards publishing your notebook involves using a hosting service that provides multiple user access.

Deploying to JupyterHub

The predominant Jupyter hosting product currently is JupyterHub. To be clear, JupyterHub is installed into a machine under your control. It provides multi-user access to your notebooks. This means you could install JupyterHub on a machine in your environment and only internal users (multiple internal users) could access it.

When JupyterHub starts it begins a hub or controlling agent. The hub will start an instance of a listener or proxy for Jupyter requests. When the proxy gets requests for Jupyter it turns them over to the hub. If the hub decides this is a new user it will generate a new instance of the Jupyter...

Optimizing your script

There are optimizations that you can make to have your notebook scripts run more efficiently. The optimizations are script language dependent. We have covered using Python and R scripts in our notebooks and will cover optimizations that can be made for those two languages.

Jupyter does support additional languages, such as Scala and Spark. The other languages have their own optimization tools and strategies.

Optimizing your Python scripts

Performance tuning your Python scripts can be done using several tools:

- `timeit`
- **Python regular expressions**
- **String handling**
- **Loop optimizations**
- `hotshot` **profiling**

Determining how long a script takes

The `timeit` function in Python takes a line of code and determines how long it takes to execute. You can also repeatedly execute the same script to see if there are start-up issues that need to be addressed.

`timeit` **is used in this manner:**

```
import timeitt = timeit.Timer("myfunction('Hello World')", "import myfunction")   t.timeit()
3.32132323232t.repeat(2, 2000000)    [#, #, #]
```

The `repeat()` function is telling `timeit` to execute the timed instruction two times at 2,000,000 times each....

Monitoring Jupyter

As with the earlier discussions in this chapter on optimization, you can also use programming tools to monitor the overall interactions of your notebook. The predominant tool for Linux/Mac environments is `memory_profiler`. If you start this tool then your notebook, the profiler will keep track of memory use of your notebook.

With this record of information points you may be able to adjust your programmatic memory allocation to be smaller in profile if you find a large memory use occurring. For example, the profiler may highlight that you are creating (and dropping) a large memory item continuously inside of a loop. When you go back to your coding you realize this memory access could be pulled out of the loop and just done once or that size of the allocation could be minimized easily.

Caching your notebook

Caching is a common programming practice to speed up performance. If the computer does not have to reload a section of code or variable or file, but can just access directly from a cache this will improve performance.

There is a mechanism to cache your notebook if you are deploying into a Docker space. Docker is a mechanism for virtualizing code over many instances in one machine. It has become common practice to do so in the Java programming world. Luckily, Docker is very flexible and a method has been determined to use Jupyter in Docker as well. Once in Docker, it is a minor adjustment to automatically cache your pages in Docker. The underlying tool used is `memcached`, yet another widespread common tool for caching anything, in this case Jupyter Notebooks.

Securing a notebook

Securing a notebook can be accomplished by several methods such as:

- **Manage authorization**
- **Securing notebook content**

Managing notebook authorization

A notebook can be secured to use username/password authorization. Authorization is on by default in your notebook. Under Jupyter it is token/password instead of username/password as a token is more open to interpretation. See Jupyter documentation on implementing authorization as this has changed slightly over time.

Securing notebook content

A notebook has possible security issues with several parts of standard content that are secured automatically by Jupyter:

- **Untrusted HTML is sanitized**
- **Untrusted JavaScript is not executed**
- **HTML and JavaScript in markdown cells is not trusted**
- **Notebook output is not trusted**
- **Other HTML or JavaScript in the notebook is not trusted**

Where trust comes down to the question: Did the user do this or did the Jupyter script? Untrusted means it will not be generated.

Sanitized code is wrapped to force the values to be text display only—no executed code will be generated. For example, if your notebook cell were to produce HTML, such as an additional `H1`...

Scaling Jupyter Notebooks

Scaling is the process of providing very large numbers of concurrent users to a notebook without a degradation in performance. The one vendor that is doing this today is Azure. They have thousands of pages and users working at scale daily.

Most amazingly this is a free service.

Sharing Jupyter Notebooks

Jupyter Notebooks can be shared by placing the notebook on a server (there are several kinds) or converting the notebook to another format (it will not be interactive, but the content will be available).

Sharing Jupyter Notebook on a notebook server

Built into the notebook configuration are extensions that can be used to expose a notebook server, directly. The notebook configuration can be generated using the following command:

```
Jupyter Notebook -generate-config
```

In the resulting `jupyter_notebook_config.py` file there are settings that can be used to set:

- **IP/port address of your notebook**
- **Encryption certificate location**
- **Password**

By setting this and starting Jupyter you should be able to access the notebook at the IP address specified from other machines in your network.

Note

You should work with your network security personnel before doing so.

Sharing encrypted Jupyter Notebook on a notebook server

If you specify the certificate information correctly in the previous configuration file the notebook will only be accessible over HTTPS or a secure, encrypted channel.

Sharing notebook on a web server

Another part of the...

Converting a notebook

You can also share a notebook with others by converting the notebook to a readable form for recipients. Notebooks can be converted to a number of formats using the `Download As` feature in the notebook `File` menu.

Notebooks can be converted in this way to the formats:

- **<language> format: This option is dependent on the language used to create the notebook. For example, an R notebook would have the choice to `Download as R script`.**
- **`HTML`: This representation is the HTML encoding to display the page as it appears in your notebook using HTML constructs.**
- **`Markdown`: Markdown is a simple display tag format used by some older Linux systems.**
- **`reST`: Another markdown type of format that has simpler display constructs than HTML.**
- **`PDF`.**

Versioning a notebook

A common practice in the programming world is to maintain a history of the changes made to a program. Over time the different versions of the program are maintained in a software repository where the programmer can retrieve prior versions to return to an older, working state of their program.

In the previous section we mentioned placing your notebook on GitHub. Git is a software repository in wide use. GitHub is an internet-based instance of Git. Once you have any software in Git it will automatically be versioned. The next time you update your notebook in GitHub. Git will take the current instance, store it as a version in your history, and place the new instance as the current—where anyone accessing your GitHub repository will see the latest version by default.

Summary

In this chapter, we deployed our notebook to a set of different environments. We looked into optimizations that can be made to our notebook scripts. We learned about different ways to share our notebook. Lastly, we looked into converting our notebook for users without access to Jupyter.