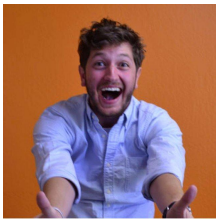
Intro to Data Science

Lesson 02 Notes

Data Wrangling

Welcome to Lesson 2



Welcome to the second lesson of Introduction to Data Science. In the first lesson, we discussed data science at a high level. So we talked about the skills that a data scientist usually has, and also some of the problems that you can solve using data science. One of the most important skills that a data scientist can have, is the ability to extract and clean data. This is usually referred to as data wrangling or data munging. Believe it or not, some data scientists say that they can spend up to 70% of their time data wrangling. This may sound crazy to you, but it's not hard to imagine this happening. Let's say you have some really cool analysis project that you want to do. Say, look at a variety of factors, and figure out why the life expectancy in City A is higher than City B. Well okay, let's say that all this data lives on a website. So, you write some scripts to go and pull this data from the website. Great, but then you need somewhere to store the data, so you're going to need a database. So okay, we, we use a database, we store all the data there. And then you look at the data and you realize, oh wait, there are a bunch of missing values, or a bunch of this data looks weird. So you need to develop some, some method to take all this data and clean it up. This is data munging. And all of this is necessary if we want to answer our original really cool question. Why is the life expectancy in city A, higher than the life expectancy in city B?

In this lesson, we'll discuss data munging basics using 3 data sets. One containing information on a bunch of baseball players, called the Lawmen baseball database. Another containing information on enrollments in an Indian identification program called Aadhar, and also data from the Last.fm API. Using these 3 data sets, we're going to discuss the formats that data can come in, how you can acquire data from these various data sources, and also how to inspects the data and see if it might have any missing or erroneous values. Alright, well let's get started.

Nick Introduction

Hello, my name is Nick Gustafson, and I'm a data scientist at Udacity. My background is I did my undergrad in neurobiology and I went on in graduate school to continue studying neurobiology, and neuroscience in particular, and kind of, computational neuroscience and looking at algorithmic and all mechanisms of learning and decision making. And while towards that, hm, tail end of my graduate work I, eventually kind of realized I did not want to stay in academia, however I still have this kind of strong passion for analyzing data, modeling data,

understanding You know, patterns within it, and kind of, you know, applying scientific methods, and data science just, just felt like a natural progression from there.

What Is Data Wrangling

You may ask yourself, what is data munging? Let's work through an example and demonstrate the concept to you. Imagine we're trying to figure out if right handed or left handed baseball players have a higher batting average and we're given this table of data to answer the question. Let's look at the table. Look, this value is incorrect. You can't have a batting average greater than 1,000. Also, it looks like we don't have any left or right-handedness for Ichiro Suzuki. Data wrangling is the art of dealing with and or converting missing or ill-formatted data into a format that more easily lends itself to analysis. But before we can do that analysis, we first have to get the data that we want to analyze. Three of the most common sources from which can get data are from files, from a database, or from websites through web APIs. We'll cover all three of these in this lesson.

Analyzing Messy Data 1

Before diving into the nitty-gritty details of data munging, I want to ask if you've ever had to analyze really messy or unorganized data? Even if you're not a data scientist, you've probably dealt with unwieldy data at some point in your life. Maybe you had trouble reading the receipts from a bake sale you organized, or you tried to keep track of your personal budget in an Excel spreadsheet. What methods or tools did you use to make your life easier? Please share your experience with the class by typing your answer in the forum. I've added a link to the discussion thread for you in the instructor comments.

Analyzing Messy Data 2

By now you've realized that data is often messy and unorganized. Because of that, a large amount of a data scientist's time is spent extracting and cleaning the data that they will use to perform analyses or make visualizations. Let's get our feet wet and first discuss the various ways that we might load our data. If we want to programmatically process and analyze our data, it's very important to understand the structure of the data itself.

Nicks Experience with Data Wrangling

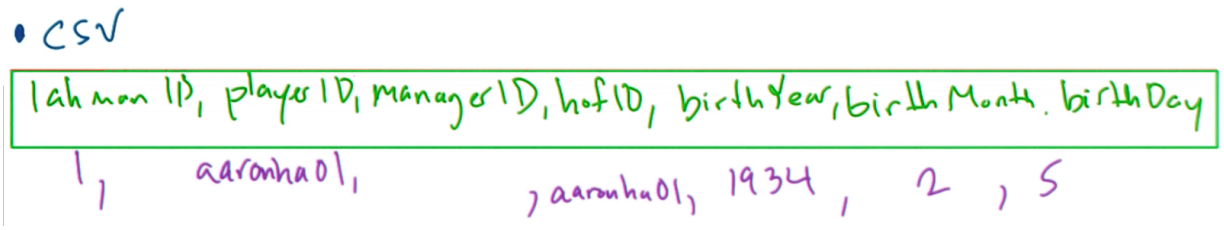
On average, over 50% of the time it's just kind of. Well, maybe be spent like, combing through the data and like just, figuring out maybe idiosyncrasies in it. And You know. You quickly apply an algorithm, you look at it. You don't get the results you expect, so you dig in a little closer and like see that there are these weird little kind of edge cases that are, are ending up more prominent in your data than you previously expected. So, you have to go back and scrape and comb through it again, and it's just this iterative back-and-forth process.

Acquiring Data

Often acquiring data to do analysis doesn't require any particularly fancy methodology. It's just a matter of finding the right file somewhere on the internet and downloading it. A lot of the data that exists out there, particularly on government websites, is stored in text files. For example, let's say we wanted to get a database of all major league baseball statistics. I kind find this data at http://www.SeanLahman.com/baseballarchives/statistics. If I visit this website, I can see that the data I want is available in a variety of formats. A comma delimited or CSV version, but also a Microsoft Access version, and a SQL version.

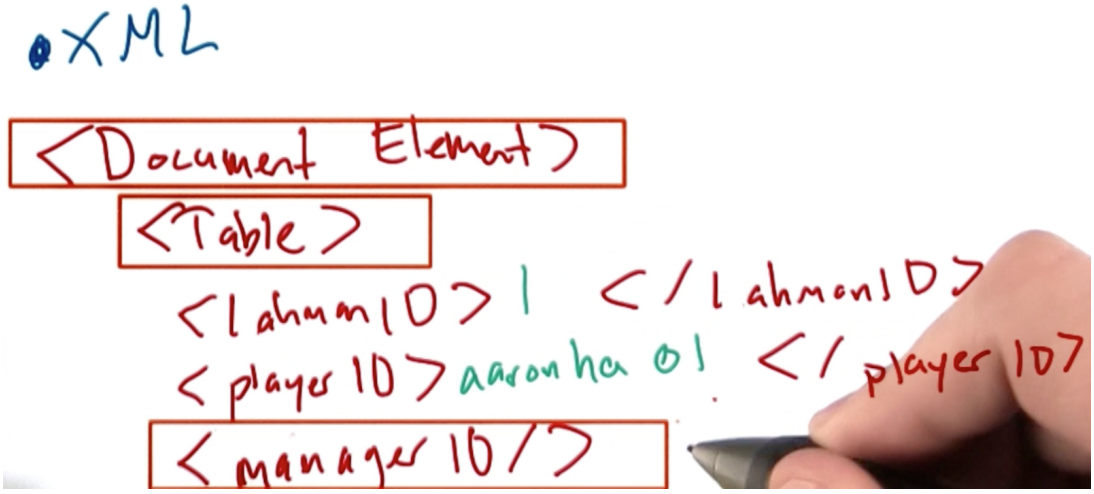
Common Data Formats

Three of the most common data formats are CSV, XML, and JSON. I want to quickly discuss what one record in the Lahman Baseball Database would look like in each of these different formats. It just so happens that the first row in this data set corresponds to a pretty famous baseball player. Hank Aaron. Let's first discuss what this data looks like in the format we originally downloaded it in, CSV. In the CSV format we usually have a series of rows. With each row corresponding to an entry. There is a header row at the top of our file. Where our comma separated values do not correspond to an entry but instead tell us what each entry means for the rest of the document. In the case of this baseball document, our first row tells us that every row will have some identifiers, the Lahman ID. The player ID, the manager ID, et cetera. Then we'll see birth year, birth month, birth day and then a bunch of other statistics. If we go and look at the entry for Hank Aaron. We see all of these values, 1 for Lahman ID, aaronha01 for player ID, et cetera, separated by commas. Note that if a particular player doesn't have a value for one field, for example Hank Aaron does not have a manager ID, we simply see two commas in a row.

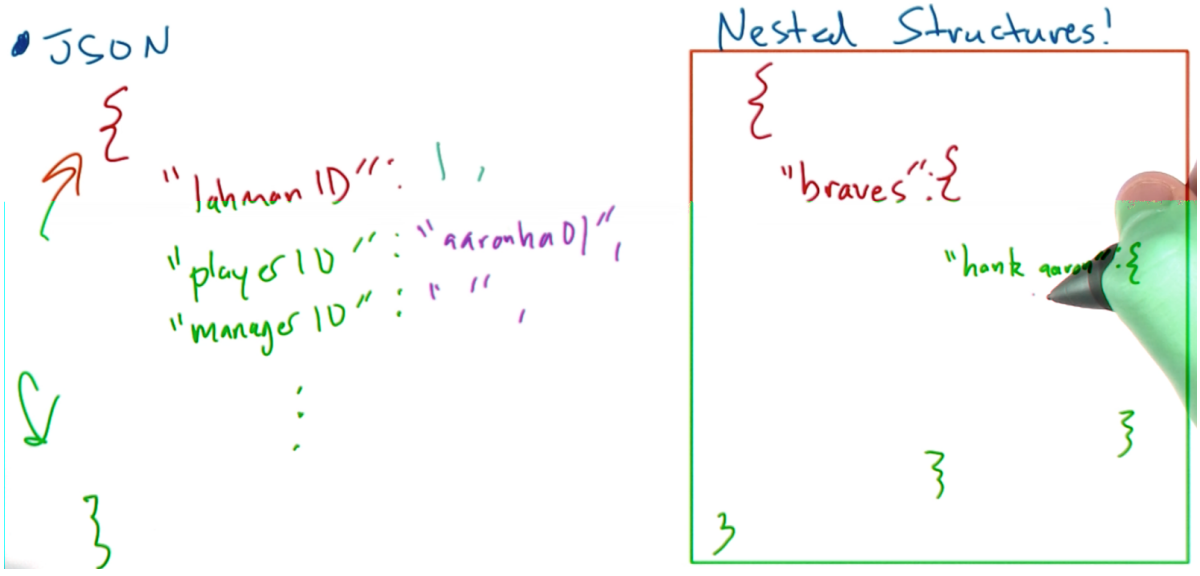


In the case of an XML document, we end up with something that pretty closely resembles HTML. We can have a document element, which is opened. It can then have a series of tables, which are also opened, as we see here. The table has a number of children, which correspond to the values that we just discussed in the CSV 1 for Lahman ID, aaronha01 for player Id, etc.

Note that when we're missing a value in XML, things are treated a bit differently. For example, since Hank Aaron was not a manager, we just open the manager ID field and then put a slash at the end.



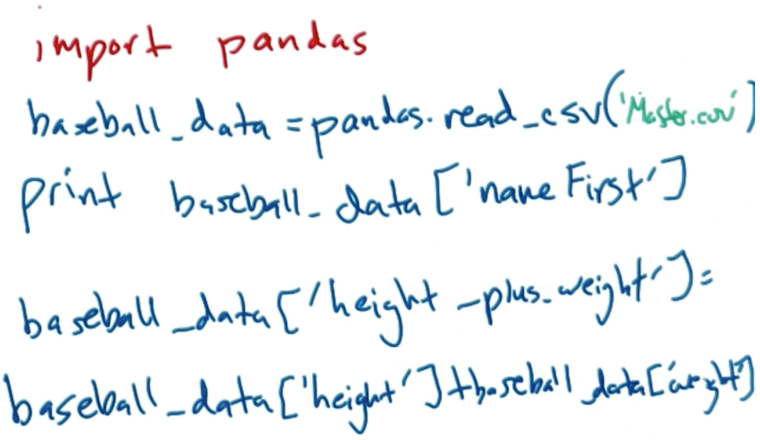
Finally, for a JSON document, we have a number of JSON objects indicated by these curly braces. A JSON object looks a lot like a Python dictionary. We have keys. Which correspond to what we would see in the header row in a CSV file followed by values. Note that in the case of a missing value, we simply open and close the quotation marks. Oftentimes, the benefit of XML or JSON is that they support nested structures in a way that CSV documents cannot. I don't want to go into this in great detail, but you can imagine that the value corresponding to a particular JSON key, say braves, could itself be another JSON object. Here's what that might look like. One final note, is to remember that when we talk about different formats that data can come in, it's not a matter of the file extension being .CSV or being .JSON. A file format really has to do with how the data is organized inside of a file. So we could easily have data that is formatted in JSON or CSV, but that comes in a file whose extension is .txt.



Since the online MTA subway data comes in CSV format, in the weather underground API responds to requests with a JSON object. We'll discuss how to load and process files in these

two formats. While we won't explicitly discuss loading and processing XML files, it isn't that different than the material will cover with regards to JSON and CSV. If you'd like to learn more about dealing with XML data in particular. I recommend you take the data wrangling class with Mongo DB taught here at Udacity. For now, let's discuss how to work with CSV files.

CSV Data 1



Let's revisit the CSV baseball data that we downloaded earlier. CSV is a very popular way to store data, likely because it's easy to read and comprehend for both humans and computers. It also probably doesn't hurt that Microsoft Excel can export data as a CSV. As you can see, every element of each row is separated by a comma. If we wanted to load this data into Pandas, we can do this in one line. It's super easy. We simply import pandas, and then write baseball_data equals pandas.read_csv Master.csv. This will load the comma separated data into what Pandas calls a data frame, where we can retrieve each column like this, print baseball_data name first. We can also create new columns on the data frame by manipulating the columns in a vectorized way. For example, if I wanted a new column that was the sum of each player's height and weight, called baseball_data height plus weight, I could write baseball_data height plus weight equals baseball_data height, plus baseball_data weight.

CSV Data 2

Now, say that I wanted to write my data to a new CSV file that included my new height plus weight column. Again, this is a simple one-line process using Pandas. I can simply call the data frame objects to CSV method, and provide as an argument the filename of the file I wish to write to. In this case, baseball data with weight height dot CSV. There are a number of optional keyword arguments I could provide here beyond the file name. For example, if I wanted to use a separator besides a comma, if I wanted to specify how to handle missing values, et cetera. You should check out the panda's documentation if you'd like to learn more.

Quiz: CSV exercise

Alright, we've shown you how to load some data into pandas from a CSV file. Now, let's see if you can load a csv into python and manipulate the data in a basic way. What we've provided here is the skeleton of a function called add full name, which takes in two arguments, path_to_csv and path_to_new_csv. Assume you'll be reading in a CSV file with the same columns as the Lahman baseball data set. Most importantly, there are columns called, nameFirst and nameLast. Write a function that reads the CSV located at path_to_csv into a

pandas dataframe, and then adds a new column called nameFull, with that player's full name. For example, in the row corresponding to Hank Aaron, nameFull would be Hank Aaron. After that, write the data in the Pandas dataFrame to a new CSV, located at path_to_new_csv. Your code should go here.

```
def add_full_name(path_to_csv, path_to_new_csv):
    #Assume you will be reading in a csv file with the same col
    #Lahman baseball data set has -- most importantly, there ar
    #called 'nameFirst' and 'nameLast'.
    #1) Write a function that reads a csv
    #located at "path_to_csv" into a pandas dataframe, adds a n
    #called 'nameFull' with a players full name.
    #
    #For example:
    #   for Hank Aaron, nameFull would be 'Hank Aaron',
        #
        #2) Write the data in the pandas dataFrame to a new csv
        #path_to_new_csv

    #WRITE YOUR CODE HERE
```

Answer:

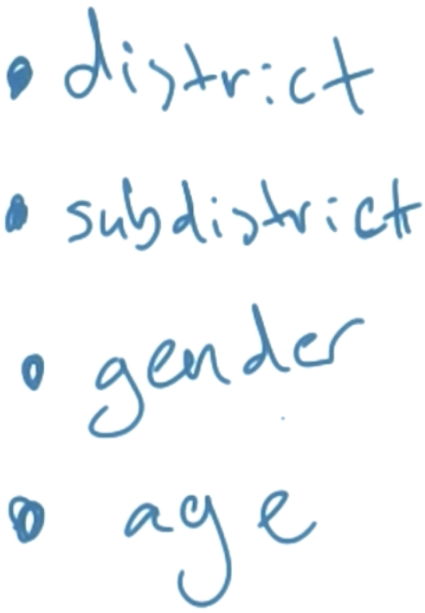
Alright, let's walk through the solution to this assessment. First, we import the pandas module, then we read in the csv located at path_to_csv, using the pandas read_csv function. Then we're going to create a new column in the dataframe called nameFull, which is the concatenation of the nameFirst column, a space, and the nameLast column. Finally, we're going to use pandas to_csv function, to write the new data frame to a new csv file located at path_to_new_csv.

What Are Relational Databases

We've talked about loading data from flat files by way of CSV, but a lot of the time, the data that data scientists work with are stored in relational databases. A relational database is similar to a collection of spreadsheets. In each spreadsheet, there are columns and rows. A column specifies a value and its type such as player ID or team ID or record. Each row contains values for each column such as 1 or 1 or the database world, we call each set of rows and columns a table rather than a spreadsheet. And the tables are usually related to each other in some way. It's very important to be somewhat familiar with relational databases. Since they're used in so many places and also since we'll be storing some of the data required for our class project in a small database. Let's take some time and talk briefly about how one can store and retrieve data from relational databases. In this course, we won't discuss the design of databases. Topics like normalization, indices, keys, things like that. Some of these topics will be covered in Data Wrangling with Mongo DB.

Aadhaar Data

To illustrate concepts for the relational databases, we are going to use some data on the aadhaar program ,that's acquired from India's online open data portal. Aadhaar is relatively new initiative in India. It is a 12 digit, unique number that enables identification for every resident of India. According to the Indian government, once instituted, the Aadhaar will provide a universal identity infrastructure, which can be used by any identity-based application. The data set we'll be using describes the number of people who enrolled in, or were rejected by the Aadhaar program. Cut by various characteristics, such as district, subdistrict, gender, and age across India. As data scientists, the Aadhaar program is interesting to us because it is a huge dataset representing one of the most significant populations on earth. We can study the Aadhaar enrollment dataset to learn a bit more about India's population. And also what type of people are applying for. And successfully enrolling in the Aadhaar program.



Quiz: Relational Databases

Before we begin our discussion of relational databases, why do you think a database might be useful? Check all options that apply. It is straightforward to extract aggregated data with complex filters. A database scales well. It ensures all data is consistently formatted. Data is redundant. Relational databases are a hot, new technology.

[x] It is straightforward to extract aggregated data with complex filters
[x] A database scales well
[x] It ensures all data is consistently formatted
[ ] Data is redundant
[ ] Relational databases are a hot, new technology

Answer:

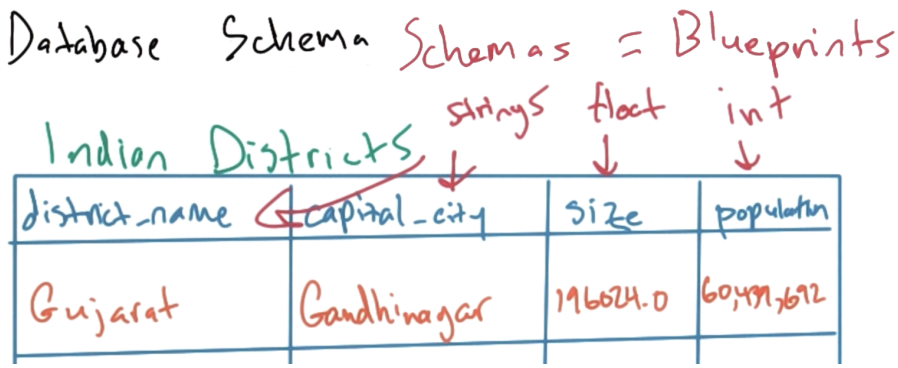
Alright. The correct answer is that relational databases are useful because it is straightforward to extract data with complex queries. A database scales well. And relational databases ensure data is consistently formatted. What do each of these mean? Well, first off it's easy to extract data from the database with a complex one line query. We can easily say choose all records for people where their age is greater than 50, and their weight is less than 50, and the city is equal to Mumbai. We can do this with flat files as well, but its a lot more work. Database is also scale well. Its not uncommon to have databases with hundreds of thousands or millions of entries. Since all information is ideally stored in one location, It's easy to update, delete, and add new data to the database in a scalable way. Think of the Aardhaar data for example. India has a population of 1.2 billion people. That's a really big data set. It's important to have a solution that

scales well. Finally, relational databases have a concept of something called a schema. Which basically says that each column in a table Is always of the same type you can't have some people's age be a string while the age of others is an int. Relational databases are built to have as little redundancy as possible that way if we want to update a value we only have to do it in one place and we can ensure that our data remains consistent throughout the entire database. Also, relational databases are well established and have been used for some time. Even if they were hot and new, that's not a great reason to use a technology. And now that we know why relational databases might be useful, let's discuss how to use them in more detail.

Aadhaar Data and Relational Databases

How does the Aadhaar data fit into the definition of relational database that we just covered? In the case of the Aadhaar data, our database would have just one table called Aadhaar Data. It has numerous columns for example, registrar, enrollment agency, state, district, and a number of others. We can imagine a related table might have information on all the Indian districts such as their populations and their size.

Introduction to Database Schemas



Relational databases always have a schema. What is a schema? In layman's terms a schema is a blueprint that tells the database how we plan to store our data. More specifically, a schema basically says that for a given table, every single row or entry will have the exact same number of columns that correspond to the same values and that each column's value will be formatted in the same way. So, for example, in this table, I can say that there are four columns: district name, capital city, size, and population. District name and capital city might always be strings, where size might be a float, and population might be an int. This list of columns and their requirements on how their values are formatted define the schema for this table. Now let's add some entries to the table. Let's say the first Aardhaar district I wish to enter is corresponds to the district Gujarat which has 60,439,692 people, has an area of 196,024 square kilometers and has a capital city of Gandhinagar. All of these entries fit the schema, so inserting this row into the table is fine. Now, let's say I had another I wish to add, the Maharashtra district, which has a capital city of Mumbai, and I provide no other values. This would also be allowed, but for all columns for which I don't specify values, the value would either be set to null or default values that are stored in the table definition.

Quiz: Database Schema

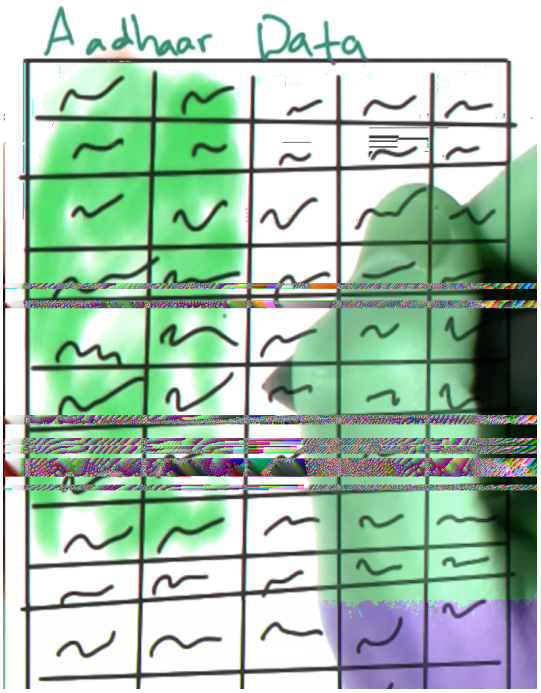
Now, say that we want to try a new row to our table. Madhya Pradesh, 28, 50.0, our table, given the schema? Check Yes or No.

[ ] No
[x] Yes

Answer:

Inserting the row, Madhya Pradesh, 28, 50.0, 60 into our aadhaar table would be allowed, despite the fact that it's sort of gibberish. Madhya Pradesh's size will be stored as 50.0 square kilometers and it's population would be stored as 60. The interesting thing here is that the capital would be stored as 28, a string which doesn't really make much sense. But, in any case, the data would still be stored. You should always be careful to define the data types for your columns in a sensible way. For example, storing numbers as strings is problematic. Since it disallows us from doing numerical computations on the column. For example, taking the average value or finding the minimum value.

Simple Queries



Alright, now let's say we have a bunch of our data in a relational database. Your next question is probably, how do we get it out? And what can we do with it? Well, let's try to learn about this using our aadhaar data. Now, data is usually retrieved from a relational database using one of a family of languages that I'll call SQL like languages. Well, there are some small variations between the languages. They have the same general concepts and syntax. Now, how could we query the Aadhaar enrollment data set with SQL like syntax, if data was in a database. Stored in a table called Aadhaar data. Well, the most simple thing that we might want to do is simply to select all of the data, that is, all rows and columns. If we wanted to do that, we could type something like select star from aadhaar data. Note that the SQL syntax is pretty human readable. We're just saying we want to select star or everything from the aadhaar data table. If we were to run this command as is, we would produce tens of thousands of rows which is probably not super constructive right now. So what we can do is, limit the number of rows by just saying limit 20 at the end of our SQL command. If we ran this query, let's see what we get back. Oh, great, the first 20 rows of the data. These are rows describing enrollments for the registrar Allahabad Bank, with various enrollment agencies, districts, age, genders, et cetera. Now, say we only wanted some of these columns. Say, district and subdistrict. We could just ask for those specific columns instead of star. For example, we might write, select district, subdistrict from aadhaar data, limit 20.

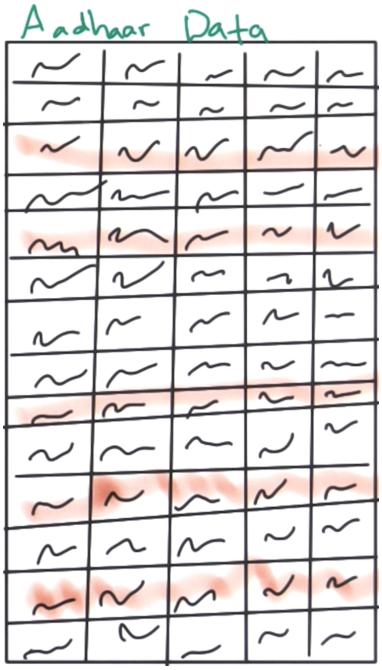
Programming Quiz: Write Your Own Simple Query

Alright, why don't you try writing a quick SQL query of your own on the India aadhaar data set. What we're going to do is read in an aadhaar data CSV to create a pandas dataframe which can be queried using SQL like syntax. We'll rename any columns that have spaces in them with underscores and set all characters to lowercase. So if the columns names more closely resemble columns names one might find in a table. Why don't you write query that will select out the first 50 values for registrar and enrollment agency in the aadhaar_data table. Your queries should go here. So that's how we pull all of the data out of SQL database.

Answer:

Alright, why don't we walk through a query that will generate the correct results? Note that we select the two columns that we want from the table, registrar and enrollment_agency, from aadhaar_data. In order to get only the first 50 rows, we add a LIMIT 50 to the end of our query. So the correct query is SELECT registrar, enrollment agency FROM aadhaar_data, LIMIT 50.

Complex Queries 1



Let's discuss a couple of more complex things that we want to do. We may not always want to pull thousands of rows, or even the first 20 rows in order. Maybe we want a specific subset of our data. For example, maybe we're interested in all data corresponding to the state Gujarat. In order to do that, we'd simply write, select star from aadhaar_data, where state equals Gujarat. Note that to limit our results, the where clause needs to go directly after the table name. SQL syntax really does mirror the way we might say things out loud very closely. Now, let's see what happens if we run this command. And there you see, we have all the data where the state is equal to Gujarat.

Complex Queries 2

The last thing that I want to quickly discuss ,are some functions that usually exist in query languages like SQL. Such as ,group bys and aggregate functions. So let's say that I wanted to create some transformed version of my data. For example, what are the total number of enrollments per district? . I could write something like this. Select district,. Sum (aadhaar - generated) from aadhaar - data, group by district. Let's talk about this a little bit more. What's happening exactly? . Sum is what we call an aggregate function. An aggregate function takes some set of values, usually numbers and performs a mathematical operation on them. We've used sum ,but other aggregate functions include count. Min, mean, max. You get the idea. Operations that one could perform on a collection of numbers. But wait. Every single row is only one number. So how do we get to collections of numbers? What we basically say here is take each distinct district. And

then for all of the different values of aadhaar_generated corresponding to a row ,for that district. Sum them up. So, we start with our aadhaar data table. Take each district ,and sum up the count aadhaar generated.

Now in order for our results to make sense, we are only going to want one row in our output for each district. So we throw in this group by clause on the end, which essentially says. Let's only have one row per district in our results. There can be numerous clauses in a group buy. So we could also say this, select district, subdistrict, sum aadhaar generated, from aadhaar data, group by district, subdistrict. Note that whatever columns we select, that we don't aggregate, we need to group by. In this case, district and sub-district. We could also put a where clause here, so in order to sum up aadhaar generated for people over 60 in each district, I can just add a where clause in after the table name, as we discussed earlier. If we were to run this query. Giving us select district, sub-district, sum aadhaar generated, from aadhaar data, where age greater than 60, group by district, sub-district. If we were to run this query, we would have a row for every combination of district and subdistrict. And we would also have for each row, account of how many aadhaar were generated ,for people over the age of 60.

Programming Quiz: Write Your Own Complex Query

So now that we've discussed how to write slightly more complicated sequel queries, why don't you give it a try? We're going to read in our aadhaar_data csv to a pandas dataframe. Afterwards, as we did before, we'll rename the columns by replacing any spaces with underscores, and by setting all characters to lowercase. So that the column names more closely resemble column names you might find in a sequel table. Can you write a sequel query that will select from the odd hard data table? How many men, and how many women, over the age of 50, have had aadharr generated for them in each district? Your query should go here.

Answer:

The correct answer is SELECT gender, district, sum aadhaar_generated FROM aadhaar_data WHERE age > 50 GROUP BY gender, district. Let's talk through this query. First, we wanted to know how many men and women in each district had aadhaar generated. So we select gender, district and sum aadhaar_generated from the aadhaar_data table. Since we're using an aggregate function, we need to include a group by, with our non-aggregated fields, in this case, district and gender. Finally, we want to limit this to men and women over the age of 50. So, we include the where clause, WHERE age > 50, after the table name.

APIs

Previously, we've shown you how to access data that sits in a file or a database. But what do you do when you want to access data that sits in some website, like Twitter data? We could just go to twitter.com and search for Udacity, but then we'd have to manually write down all the data. We could try to rate a web crawler to go through the pages HTML, but as you can imagine, that could get kind of complicated. Luckily, many tech companies like Twitter allow

users and developers to access data directly in a simple, machine readable format. They do so through something called an Application Programming Interface, or API. There are several different kinds of APIs, but one of the most common types, and the one used by Twitter, is a representational state transfer or REST API. I know I sound like a broken record here, but if you'd like to learn more about the nitty gritty behind REST APIs, I'd recommend you take the data wrangling class with MongoDB.

API Example

Let's use last.fm's API as an example, and let's see how we can interact with it to get data and find out the top albums in the US, China, and Spain. As you can see on the left hand side here, there are a few different last.fm API methods that we can talk to and retrieve information from. For example, let's check out the album.getInfo method. As you can see this page tells us the type of data that the API method will return like artist name, album name and a bunch of other information such as language or a music brains ID from the album. But the question is now how do we actually get to this data from last fm's API. It's actually really simple. We can simply copy and paste a link like this into the web browser. And look at all the information that last.fm returned to us.

Okay, now I want to point out a few interesting things about the URL. See all the parts after the question mark? That's how we define API parameters. Like the method. The API key, the artist name and the name of the album that you want data for. So for example, if we wanted information on a different Rihanna album, like the album Unapologetic, we could simply change the album parameter here from Loud to Unapologetic. Let's see if that works. Look, we now have the data from Rihanna's album, Unapologetic. Pretty simple, huh?

Data in JSON Format

I want to highlight another thing. Let's look at this data. It looks kind of, strange, right? This data's actually in JSON format. The same JSON that we discussed earlier in this class. As I insinuated earlier, we can kind of, think of JSON data as a Python dictionary. Let's make this data easier to read and I'll show you what I mean. See how each piece of data in this JSON object corresponds to a key, very much how like every value in a Python dictionary corresponds to a key. So our first key here is album. And the value is actually another JSON object. Here we have key name, value unapologetic, key artist, value of Rihanna, and so on. Although JSON is not equivalent to Python dictionaries, you can think of them as similar abstractions.

How to Access an API Efficiently

You may think it's pretty inefficient that we have to type a URL into the browser every time we want to grab some data from last.fm's API. There has to be a more efficient way to do this and you're right. We can write a simple Python program that uses the JSON and request libraries to do exactly what we did manually a few moments ago. If we run this program right here, let's think of what it returns. Look, the data that this Python program returned is exactly the same based on data that you saw in the browser a second ago. This program was less than 10 lines. First we specify a URL, then we simply say request that get, that URL and call.text, in order to get the text. We assign that value to data. We print type of data which we saw was Unicode, and we print data itself, which was the JSON object. But if we go back and look at the JSON object itself, we see that it's in this funky string format that makes it very hard for us to parse out the interesting information. We could write a Regex to parse out what we want, but that can get pretty painful really quickly. The JSON library will make interacting with JSON data as easy as 1, 2, 3. Let me show you how.

```
if __name__=="__main__":
    url = 'http://ws.audioscrobbler.com/2.0/?method=album.getin
fo&api_key=4beab33cc6d65b05800d51f5e83bde1b&artist=Cher&album=B
elieve&format=json'
    data = requests.get(url).text
    data = json.loads(data)
    print type(data)
    print data
    data['artist']
```

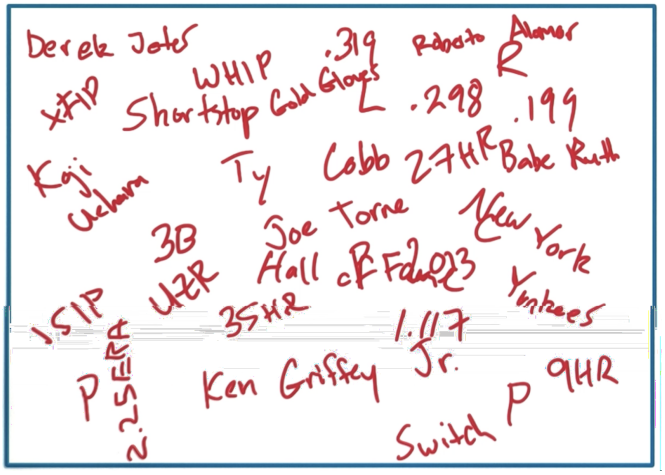
We've modified our script ever so slightly. After initially assigning data, we reassign it. We say data equals json.loads data. What json.loads does is it interprets a string and assumes that it's a JSON object. By doing so, we convert the JSON data into a Python dictionary. We'll see that when we print type data. Once we convert the data into a nicely structured dictionary, we can get the interesting bits out as if we're simply accessing a dictionary. For example, typing data artist. Let's see what this script produces now. As we can see, what we return here is that the type of the object is a dictionary and then we have a Python dictionary.

Programming Quiz: API Exercise

Now, let's put everything together. Let's use last.fm's geo.gettopartists API call to write a short program to find the top-performing artist in Spain. Here's a hint, on the page here it shows that we have to pass in the country parameter. Also, you may want to use the JSON library so it will be easier for you to read the JSON data. If you need to register for your own API key, you can do so here. Your code should go into this space.

Answer:

Alright, let's work through the solution. First we import the json and request libraries. Then we provide the URL that we're going to be making our API call to. Then we make our API call using the request library, and load the results into a dictionary. Finally, we print out the name of the number one top artist. In the data dictionary, we will look at the top artist's key. Then we look at the artist key there. We look at the first entry. And we look at the name parameter. As you can see, the URL is the biggest difference between this code and the sample code in the previous video. Mainly I've just changed the parameters and the URL variable. To make sure that I'm getting the top artists, I am calling the geo.gettopartists method. I'm also passing the country parameters in by setting the parameter country equal to Spain. And once I have retrieved the data, I'm loading it into a json file in this line. Data equals json.loads(data). This lets me find the top performing artist quickly and easily by accessing it almost like a python dictionary. You could also load the data as json and try to just parse it with the regex, but I personally think that would be really painful.



Sanity Checking Data

Now that we have acquired our data, whether the informal flat file sequel like relational database, or an API, you might think we are ready to dive in and do some analysis. Alas, I am afraid, we are not quite there yet. There can often be problems with our data. It might not be in a format that allows us to easily perform the desired analysis, or maybe there's a bunch of bad or missing values. In this next section, let's discuss how to deal with these issues so that we can get to the exciting part of any project, the analysis. Let's talk briefly about sanity checking our data. What does it mean to sanity check data? Well, put briefly, if we're sanity checking data we want to quickly determine, does the data make sense? Is there a problem? Does the data look like I expect it to? There's a ton of work you can do to sanity check your data. We can draw plots to visualize the data. You can run some basic analyses and tons of other things. I don't want to focus too much on this in this course. If you'd like to dive deep into the subject, I'd recommend taking you down to these exploratory data and analysis course. However, to do just the bare amount of sanity checking, Pandas DataFrames do have a useful method called the describe.

Pandas Describe Function

Let's illustrate how Panda's describe function works, using our Laman baseball data set. Say that our baseball data was loaded into a Panda's data frame called baseball, using Panda's read.csv function. Like so. If we call baseball dot describe, what do we get back? You can see that baseball that describe returns a data frame in it of itself. For every numerical column, we

```
>>> baseball = pandas.read_csv('../data/Master.csv')
>>> baseball.describe()
            lahmanID     birthYear    birthMonth
count  18125.000000  17879.000000  17661.000000  175
mean    9210.497655   1928.730186      6.627711
std     5461.545073     40.076883      3.465484
min        1.000000   1820.000000      1.000000
25%     4537.000000   1894.000000      4.000000
50%     9069.000000   1933.000000      7.000000
75%    13606.000000   1966.000000     10.000000
max    19420.000000   1993.000000     12.000000
```

see the count, mean, standard deviation, mean, values.
We can do some quick checking to make sure there are data generally make sense. Here. LahmanID has actually been read in as a number, which is a bit misleading. We won't be doing any arithmetic on it. But we see that the minimum birth month is one, and the maximum birth month is 12, as we would expect. We see that the minimum birth date is one, and the maximum birth date is 31. That makes sense. And we see that the mean birth year is actually 1928, which to me is surprising. I'd think that it would be a little bit later. Investigating values like this, we can tell pretty quickly what our data looks like, and whether there might be any significant outliers in our data. In other words, are the min or max way larger than the values corresponding with the 25th or 75th percentile. Although we won't discuss exploratory analysis of data in depth, there's one thing that you might notice when looking at a summary of your data. It may have a bunch of missing values. This is evidenced here by the differences in count for our various columns. Since this is a particularly common problem. Let's discuss why values may be missing. And different methods we can use to mitigate the effect of those missing values on our analysis.

Quiz: Why Are Values Missing

In just a moment we're going to discuss what to do when there are missing values in your data. But before we get into the technical details you might ask yourself why that data's missing in the first place. Why do you think that values may be missing from your data? Share your thoughts in the box below. Don't worry, your response won't be graded.

[x] Occasional system errors prevent data from being recorded

[x] Some subset of subjects or event types are systematically missing certain data attributes, or missing entirely

Answer:

There are a number of different reasons why values may be missing from your data. But here are a couple that I think are particularly common or important. So, the first is that occasional system errors may prevent data from being recorded. Another reason that data may be missing, is that some subset of subjects or event types are systematically missing certain data attributes. Or maybe missing entirely. Let's dive deeper into both of these reasons. One explanation for missing data is that something may be failing occasionally when collecting data. For example, if you were conducting a traditional door-to-door survey, maybe the person handling the documents loses a few or tends to miss some streets at random because they don't know the geography of the town well.

Missing Values

One explanation for missing data is that something may be failing occasionally when collecting your data. For example, if you were conducting a traditional door to door survey, maybe the person handling the documents loses a few, or tends to miss some streets at random because they don't know the geography of your town very well. This is a case of a researcher or collection technique causing missing data. The offline example of someone conducting a traditional door-to-door survey can easily be translated to the digital world. Imagine your data collection software fails one out of every 1,000 times, or you have a couple of service outages. You could have similar data loss. If this were to happen in our baseball player data set, we might see some missing values here and there, just because we made some error in collecting them at random. Another more harmful reason values may be missing is referred to as non response. In the case of non response, a certain subset of people chooses not to answer particular questions. Or don't respond at all. This can lead to biases in your data and consequently, false conclusions. In the case of baseball data, say we were interested in the batting average of players. Maybe, since they're, I don't know, self-conscious about their batting average. All outfielders just don't provide a value. This would make our results inaccurate and lead us to conclude that the average batting average was 312 even though that is based only one data point here, Derek Jeter's.

Name	Position	L or R	AVG
Derek Jeter	Shortstop	R	.312
Ichiro Suzuki	Outfield	L	.3~~10~~
Babe Ruth	Outfield	L	.3~~42~~
Barry Bonds	Outfield	L	.2~~18~~
Ken Griffey Jr.	Outfield	L	.2~~84~~

The important point here is this, if the missing values from your data are distributed at random. Your data may still be representative of the population. However, if values are missing systematically, it could invalidate your findings. It's important to design your experiment or data collection method to minimize these effects. Beyond this, it's very important to check your data to see if such effects are present.

Dealing With Missing Data

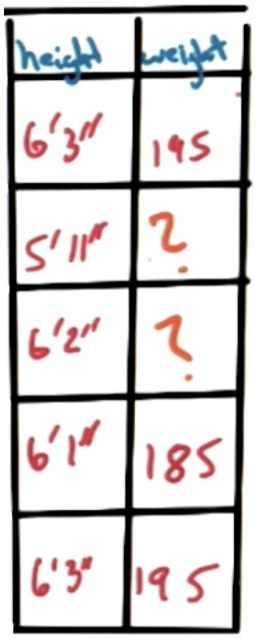
So let's say that there are missing values in your data and they're distributed at random. What do we do? There are two approaches that I want to discuss here, partial deletion and imputation. Partial deletion is exactly what it sounds like, limiting our data set for analysis to the data that we have available to us. Let's discuss partial deletion further. For the following discussion let's use the Sean Lawman database of baseball players. There are a number of players for whom many values are missing. Now let's say that we wanted to ask a couple of different questions. For example what are the average life span and average height of baseball players? There are many players that we can't include in the analysis, either because their birth date or death date is missing, or because they're still alive. In this case, we may want to perform partial deletion. One method we could use is called Listwise Deletion. In the case where we perform Listwise Deletion, we'd exclude a particular data point from all analyses even if some useful values were present. So if using Listwise Deletion, Barry Bonds, who has a listed height, but not death date, would not be included in either analysis, even if we were calculating the average height of players. On the

other hand, in the case where we might perform Pairwise Deletion, we would exclude a particular case from the analysis for tasks which are not possible with the data at hand. So we wouldn't include Barry Bonds in our sample when calculating life span since there's no death date, but we would still use his height when calculating the average height of all baseball players.

Why Impute

In scenarios where we don't have very much data, or where removing our missing values would compromise the representativeness of our sample, it might not make sense to throw away a bunch of our entries just because they're missing values. This could severely impact the statistical power of whatever analysis we were trying to perform. In this case, it likely makes sense to make an intelligent guess at the missing values in our data. The process of approximating these missing values is referred to as imputation. There are many different ways to impute missing values. And different techniques are constantly being developed. I want to quickly discuss some relatively simple ways to impute missing values in our data. Let's note that imputation is a really hard problem. Each of the methods we'll discuss introduce a certain biases or inaccuracies into your data set. We're discussing some of the most simple ways to impute data, but much more sophisticated and robust methods are out there.

Easy Imputation

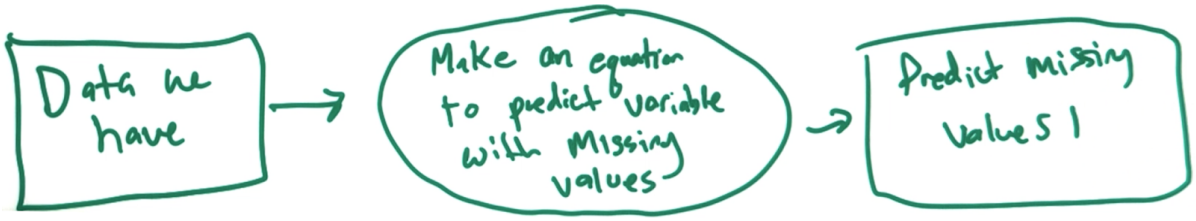


Let's first discuss what would seem to be the easiest way to impute a missing value in our data set. Just take the mean of our other data points and fill in the missing values. So, for example, let's say that Ichiro Suzuki and Babe Ruth are missing values for weight in our baseball data set. Well, okay, no problem. We can just take the mean of all other players weights and assign that value to Ichiro and Babe Ruth. In this case, we would assign Ichiro and Babe Ruth both a weight of 191.67. Wow, that seems really easy, right? There's gotta be a catch. Well, let's first discuss what's good about this method. We don't change the mean of the height across our sample, That's good. But let's say we were hoping to study the relationship between weight and birth year. Or height and weight. Just plugging the mean height into a bunch of our data points lessens the correlation between our imputed variable and any other variable.

Impute Using Linear Regression

Another method that we could use to impute missing values in a data set is to perform linear regression to estimate the missing values. We'll cover linear regression in more depth in the next lesson. But the general idea is that we would create an equation that predicts missing values in the data using information we do have, and then use that equation to fill in our missing values. Okay so, what are the drawbacks of using this linear regression type technique? Well, one negative side effect of imputing missing values in this way is that we would overemphasize

existing trends in the data. For example, if, if there is a relationship between date of birth and height in MLB players, all of our imputed values will amplify this trend. Additionally, this model will produce exact values for the missing entries, which would suggest a greater certainty in the missing values than we actually have. In any case, let's say we did want to fill in the missing values for weight in our baseball player data. We could train a linear model using the existing data that we have, and then use that model to fill in these missing values. Let's say we did want to fill in the missing values for weight in our baseball data. We could train a linear model using our existing data. That is, entries that have position, left or right handed batter, average, birthdate, deathdate, height and weight. And then use that model that we've created to fill in these missing values.



Programming Quiz: Imputation Exercise

Alright, why don't you try to impute some values on your own. Pandas dataframes have a method called fillna. Which take in a value and allow you to pass in a static value to replace any NAs that exist in a dataframe or series. You can call this function like so. Data frame column, fillna value. Using the numpy.mean function, which calculates the mean of an numpy array, why don't you impute any missing values in our Lahman baseball data sets weight column, by setting the missing values equal to the average weight. Your code should go here.

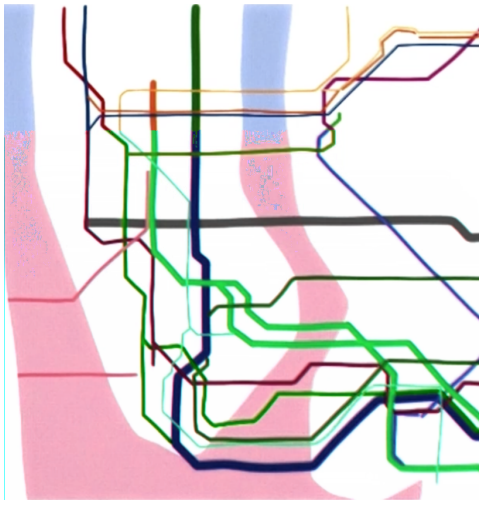
Answer:

All right, let's walk through the solution to this guy. It's a pretty straightforward one line solution. We simply say baseball weight equals baseball weight.fillna numpy.mean baseball weight. What we're doing here is saying let's compute the mean of all the existing non NA values in baseball weight. Then let's take all the NA's in baseball weight. And replace them with this mean value. Let's take the baseball weight column, and replace it with this new column, which doesn't have any NA values in it.

Tip of the Imputation Iceberg

These methods are literally just the tip of the iceberg when it comes to data amputation. I wanted to make sure that you were exposed to the concept, but there are much more robust and sophisticated methods to fill in missing values in your data. Filling in the missing values on a data set by computing the mean of the field in question, or fitting a linear model, are somewhat effective and simplistic methods, but they can both have negative side effects and amplify or attenuate preexisting trends in your data.

Assignment 2



Now you're equipped with some basic skills necessary to acquire and clean up data. We'll use these tools to start working on our class project. Analyzing the dynamics and workings of the New York city subway system. First you'll use your knowledge of how to acquire data via an API to obtain weather underground data on New York city for a month in SQL skills to run some basic queries on the data and get a sense for what our data looks like. Once you're comfortable with the weather data you'll gather a bunch of data related to the MTA subway itself from the MTA website. In addition to gaining some basic familiarity with the MTA data you'll do some processing and cleaning of that data such that it'll be very easy to do analysis of the data in the very near future.

Nick Coolest Project

The coolest one I've worked on so far. The one that I there are several that we've done kind of, on the team. But the one that sort of seemed most flushed out and seemed kind of have the best effect within the company thus far is the we made a coaching dashboard for the course managers, to kind of assign each student a risk index, which measures You know, roughly, like it looks at various features related to this, you know, behavior and usage of the site and tries to you know, figure out how well they're doing. And also on top of that, a sort of urgency index which then tracks sort of changes in risk and allows one to in theory kind of group students into trends.

Recap

To recap, over the course of this lesson we've discussed a bunch of the basic skills required to do data wrangling or data munging. So we've discussed some of the formats that data might come in. Such as CSV, JSON, and xml. We've also talked about how to get this data from a variety of sources. Such as SQL like databases, flat files, and API's. We've also talked about how to look at the data, see if things look all right, and if they don't, some of the methods to mitigate these problems, such as data imputation. Over the course of doing these basic acquisition and cleaning tasks, you'll usually discover some basic things about your data. But you might also want to do some more sophisticated analysis to learn some more subtle things about your data set. That's what we'll learn in the next lesson, and then we'll apply those insights to the New York City subway data.