(https://dato.com)

Evan Samanas
Uploaded February 21, 2014

DOWNLOAD

Introduction to SFrames

What is an SFrame?

Note: This notebook uses GraphLab Create 1.7.

An SFrame is a tabular data structure. If you are familiar with R or the pandas python package, SFrames behave similarly to the dataframes available in those frameworks. SFrames act like a table by consisting of 0 or more columns. Each column has its own datatype and every column of a particular SFrame must have the same number of entries as the other columns that already exist. There are two things that make SFrames very different from other dataframes:

- Each column is an SArray, which is a series of elements stored on disk. This makes SFrames disk-based and therefore able to hold datasets that are too large to fit in your system's memory. You'll see this come in to play throughout this demo.
- An SFrame's data is located on the server that is running the GraphLab toolkits, which is not necessarily on your client machine. While this example does not demonstrate working with a GraphLab server on a different machine, you can see that in action here (https://dato.com/learn/notebooks/running_in_the_cloud.html).

This tutorial shows you how to import data into an SFrame, do some basic data cleaning/exploration, and save your work for later. If you are someone that likes to learn these things through reading comprehensive documentation instead of tutorials, then you can visit our API Reference (https://dato.com/products/create/docs/generated/graphlab.SFrame.html) first. If not, read on!

Getting Started: Creating SFrames

First we will get set up with import statements for this tutorial.

In [1]:

```
import graphlab as gl
```

Reading a csv file from an S3 bucket is just one way to import your data into an SFrame. The read_csv function gives you lots of control over where to read your data from and how to parse it, which you can read about here (https://dato.com/products/create/docs/generated/graphlab.SFrame.read_csv.html#graphlab.SFrame.read_csv). The column_type_hints option is important to highlight though, as without hints SFrame will simply decide that every column it finds is a string. Here, only the year column is of type string.

(The csv file of song metadata comes from the Million Song Dataset (http://labrosa.ee.columbia.edu/millionsong/). This data set was used for a Kaggle challenge (https://www.kaggle.com/c/msdchallenge) and includes data from The Echo Nest (http://the.echonest.com/), SecondHandSongs (http://www.secondhandsongs.com/), musiXmatch (http://musixmatch.com/), and Last.fm (http://www.last.fm/).)

In [2]:

```
# In order to interact with S3 we need to set our AWS credentials.
# You can use your own credentials or use the ones below.
gl.aws.set_credentials('AKIAJMHKEZGY6YP24BXA', 'vf/miz2Zx7V7VkCai9ZeJR45ZSimqu6/W7qdRLmN')

# The below will download a 78 MB file.
song_sf = gl.SFrame.read_csv('s3://dato-datasets/millionsong/song_data.csv',
                             column_type_hints = {'year' : int})
```

```
PROGRESS: Finished parsing file s3://dato-datasets/millionsong/song_data.csv
PROGRESS: Parsing completed. Parsed 100 lines in 0.781915 secs.
PROGRESS: Read 637410 lines. Lines per second: 156538
PROGRESS: Finished parsing file s3://dato-datasets/millionsong/song_data.csv
PROGRESS: Parsing completed. Parsed 1000000 lines in 4.2745 secs.
```

In [3]:

```
song_sf.num_rows()
```

Out[3]:

```
1000000
```

If the csv file we want to read does not have a header, we can still provide column_type_hints, but with GraphLab's default column names. Below is the code that would accomplish this, but I have commented it out because I don't want to affect the dataset we work with in the rest of this tutorial.

In [4]:

```
#song_sf = gl.SFrame.read_csv('s3://dato-datasets/millionsong/song_data.csv', header=False,
#                              column_type_hints = {'X5' : int})
#song_sf.head(1)
```

Before we start playing with this data, I want to highlight that you can save and load an SFrame for later use. This is great if you don't want to re-download a file from S3 a bunch of times, or re-parse a large csv file. Here's how to save to your current directory:

In [5]:

```
song_sf.save('orig_song_data')
```

That save operation takes some time because it copies the files SFrame uses to the given location (in this case, an auto-created directory called 'orig_song_data'). The load operation, however, is instantaneous. This is one of the perks of using a disk-backed dataframe.

In [6]:

```
song_sf = gl.load_sframe('orig_song_data')
```

Viewing data

I can emit several commands to see that we are working with a fairly tame dataset. After all, we only have five columns.

In [7]:

```
song_sf.head(5)
```

Out[7]:

song_id	title	release	artist_name	year
SOQMMHC12AB0180CB8	Silent Night	Monster Ballads X-Mas	Faster Pussy cat	2003
SOVFVAK12A8C1350D9	Tanssi vaan	Karkuteillä	Karkkiautomaatti	1995
SOGTUKN12AB017F4F1	No One Could Ever	Butter	Hudson Mohawke	2006
SOBNYVR12A8C13558C	Si Vos Querés	De Culo	Yerba Brava	2003
SOHSBXH12A8C13B0DF	Tangle Of Aspens	Rene Ablaze Presents Winter Sessions ...	Der Mystic	0

[5 rows x 5 columns]

In [8]:

```
song_sf.tail(5)
```

Out[8]:

song_id	title	release	artist_name	year
SOTXAME12AB018F136	O Samba Da Vida	Pacha V.I.P.	Kiko Navarro	0
SOXQYIQ12A8C137FBB	Jago Chhadeo	Naale Baba Lassi Pee Gya	Kuldeep Manak	0
SOHODZI12A8C137BB3	Novemba	Dub_Connected: electronic music ...	Gabriel Le Mar	0
SOLXGOR12A81C21EB7	Faraday	The Trance Collection Vol. 2 ...	Elude	0
SOWXJXQ12AB0189F43	Fernweh feat. Sektion Kuchikäschtli ...	So Oder So	Texta	2004

[5 rows x 5 columns]

In [9]:

```
song_sf.num_rows(), len(song_sf)
```

Out[9]:

```
(1000000, 1000000)
```

In [10]:

```
song_sf.num_cols()
```

Out[10]:

```
5
```

In [11]:

```
song_sf.column_names()
```

Out[11]:

```
['song_id', 'title', 'release', 'artist_name', 'year']
```

In [12]:

```
song_sf.column_types()
```

Out[12]:

```
[str, str, str, str, int]
```

Modifying an SFrame

Alright, I want a little more out of this SFrame. I want to add a few columns. Let's say I care about the length of the title of each song, what I've rated the song, and how many years old I was when the song was created.

In [13]:

```
year_i_was_born = 1988

# Count the number of words in each song title and add the word count as a new feature
song_sf['title_length'] = song_sf['title'].apply(lambda x: len(x.split()))

# Count how old I was when this song came out
song_sf.add_column(song_sf.select_column('year').apply(lambda x: x - year_i_was_born),
                   'how_old_was_i')

# Add a 0 rating for every song
song_sf['my_rating'] = 0
song_sf.head(5)
```

Out[13]:

song_id	title	release	artist_name	year	title_length	how_old_was_i
SOQMMHC12AB0180CB8	Silent Night	Monster Ballads X-Mas	Faster Pussy cat	2003	2	15
SOVFVAK12A8C1350D9	Tanssi vaan	Karkuteillä	Karkkiautomaatti	1995	2	7
SOGTUKN12AB017F4F1	No One Could Ever	Butter	Hudson Mohawke	2006	4	18
SOBNYVR12A8C13558C	Si Vos Querés	De Culo	Yerba Brava	2003	3	15
SOHSBXH12A8C13B0DF	Tangle Of Aspens	Rene Ablaze Presents Winter Sessions ...	Der Mystic	0	3	-1988

my_rating
0
0
0
0
0

[5 rows x 8 columns]

Clearly songs with a '0' year are a problem, but we'll cover that later.

A few things to cover from the snippet above:

- I can either use 'add_column'/'select_column', or just use python's index syntax to complete the same task.
- It is easy to create new columns from an existing one by using 'apply' to apply a function to each element.
- If you want a column to have the same value for every entry, just assign a single value to it. We can do this with existing columns as well:

In [14]:

```
song_sf['my_rating'] = 1
song_sf.head(5)
```

Out[14]:

song_id	title	release	artist_name	year	title_length	how_old_was_i
SOQMMHC12AB0180CB8	Silent Night	Monster Ballads X-Mas	Faster Pussy cat	2003	2	15
SOVFVAK12A8C1350D9	Tanssi vaan	Karkuteillä	Karkkiautomaatti	1995	2	7
SOGTUKN12AB017F4F1	No One Could Ever	Butter	Hudson Mohawke	2006	4	18
SOBNYVR12A8C13558C	Si Vos Querés	De Culo	Yerba Brava	2003	3	15
SOHSBXH12A8C13B0DF	Tangle Of Aspens	Rene Ablaze Presents Winter Sessions ...	Der Mystic	0	3	-1988

my_rating
1
1
1
1
1

[5 rows x 8 columns]

We can also add several columns at a time:

In [15]:

```
song_sf[['dumb_col','dumb_col2']] = [song_sf['title_length'],song_sf['my_rating']]
song_sf.head(5)
```

Out[15]:

song_id	title	release	artist_name	year	title_length	how_old_was_i
SOQMMHC12AB0180CB8	Silent Night	Monster Ballads X-Mas	Faster Pussy cat	2003	2	15
SOVFVAK12A8C1350D9	Tanssi vaan	Karkuteillä	Karkkiautomaatti	1995	2	7
SOGTUKN12AB017F4F1	No One Could Ever	Butter	Hudson Mohawke	2006	4	18
SOBNYVR12A8C13558C	Si Vos Querés	De Culo	Yerba Brava	2003	3	15
SOHSBXH12A8C13B0DF	Tangle Of Aspens	Rene Ablaze Presents Winter Sessions	Der Mystic	0	3	-1988
		...				

my_rating	dumb_col	dumb_col2
1	2	1
1	2	1
1	4	1
1	3	1
1	3	1

[5 rows x 10 columns]

But maybe that was a dumb idea. Let's get rid of those. Before I do that, I'll show you how to rename and swap column ordering. Why not?

In [16]:

```
song_sf.rename({'dumb_col2' : 'another_dumb_col'})
song_sf.swap_columns('dumb_col', 'another_dumb_col')
del song_sf['dumb_col']
del song_sf['another_dumb_col']
song_sf.head(5)
```

Out[16]:

song_id	title	release	artist_name	year	title_length	how_old_was_i
SOQMMHC12AB0180CB8	Silent Night	Monster Ballads X-Mas	Faster Pussy cat	2003	2	15
SOVFVAK12A8C1350D9	Tanssi vaan	Karkuteillä	Karkkiautomaatti	1995	2	7
SOGTUKN12AB017F4F1	No One Could Ever	Butter	Hudson Mohawke	2006	4	18
SOBNYVR12A8C13558C	Si Vos Querés	De Culo	Yerba Brava	2003	3	15
SOHSBXH12A8C13B0DF	Tangle Of Aspens	Rene Ablaze Presents Winter Sessions	Der Mystic	0	3	-1988
		...				

my_rating
1
1
1
1
1

[5 rows x 8 columns]

Still with me? Notice that the column types for the transformed columns are correct.

In [17]:

```
song_sf.column_types()
```

Out[17]:

```
[str, str, str, str, int, int, int, int]
```

Hold on though, I think I'd actually like the rating to be a float.

In [18]:

```
song_sf['my_rating'] = song_sf['my_rating'].astype(float)
song_sf.column_types()
```

Out[18]:

```
[str, str, str, str, int, int, int, float]
```

To create even more interesting feature columns, you may want to apply a function using multiple (or all) columns. When you apply a function to an SFrame (instead of just an SArray like I did earlier), the input to the function is a dictionary where the keys are your column names. Here I'd like to know what combination of song title, album title, and artist name mentions the word 'love' the most:

In [19]:

```
song_sf['love_count'] = song_sf[['release', 'title', 'artist_name']].apply(
    lambda row: sum(x.lower().split().count('love') for x in row.values()))
song_sf.topk('love_count').head(5)
```

Out[19]:

song_id	title	release	artist_name	year	title_length
SOBYJSK12A8C132021	Love To Love To Love You Baby ...	Love To Love To Love You Baby & Break 4 Love ...	David Vendetta	0	7
SOXAVWF12A8AE4922C	One Piece	Low Low Low La La La Love Love Love ...	Low Low Low La La La Love Love Love ...	2007	2
SOKXWMK12A8AE45BDB	Only Let Things Be	Low Low Low La La La Love Love Love ...	Low Low Low La La La Love Love Love ...	2007	4
SOHCQVM12A8AE49226	Venus In The Evening I Have Disappeared Again ...	Low Low Low La La La Love Love Love ...	Low Low Low La La La Love Love Love ...	2007	8
SOEXSIX12A58A77BA2	Rest Your Arms	Low Low Low La La La Love Love Love ...	Low Low Low La La La Love Love Love ...	2007	3

how_old_was_i	my_rating	love_count
-1988	1.0	7
19	1.0	6
19	1.0	6
19	1.0	6
19	1.0	6

[5 rows x 9 columns]

We can see from these examples that adding and deleting columns is a simple task for an SFrame. This is because an SFrame is essentially the keeper of references to columns (SArrays), so adding and deleting columns is a very cheap operation. However, the fact that SFrames store their data on disk produces some important limitations when thinking about editing an SFrame:

- SFrames are immutable with respect to column size and data.
- SFrames do not support random access of elements and are not indexed.

Sequential access is king on disk, and this is very useful to remember when working with SFrames. This means that inspecting a specific row would perform quite poorly and writing to a specific row is not possible. However, while working with SFrames you'll find that you can still accomplish nearly all of what you would have done with a more classic dataframe using transform and filter operations, yet you'll still reap the huge benefit of creating SFrames that are larger than the size of your machine's memory. So let's learn about filtering!

Filtering and Missing Values

I think I want to take care of those invalid year entries now. I don't really know how many there are, so I'll find out, as the answer to that may change what I do.

In [20]:

```
year_count = song_sf.groupby('year', gl.aggregate.COUNT)
print year_count.head()
print "Number of unique years: " + str(len(year_count))
print "Number of invalid years: "
year_count.topk('year', reverse=True, k=1)
```

```
+------+--------+
| year | Count  |
+------+--------+
| 1982 |  3597  |
| 1930 |   40   |
| 1975 |  2482  |
| 1942 |   24   |
| 1959 |  592   |
| 1990 |  7258  |
| 1974 |  2186  |
|  0   | 484424 |
| 1947 |   57   |
| 1996 | 14135  |
+------+--------+
[10 rows x 2 columns]

Number of unique years: 90
Number of invalid years:
```

Out[20]:

year	Count
0	484424

[1 rows x 2 columns]

Yikes, that's almost half of my dataset. Maybe I don't want to just get rid of that data. SFrames support missing values, and these are represented using 'None' . We will transform the apporpriate values to missing here:

In [21]:

```
song_sf['year'] = song_sf['year'].apply(lambda x :None if x == 0 else x)
song_sf.head(5)
```

Out[21]:

song_id	title	release	artist_name	year	title_length	how_old_was_i
SOQMMHC12AB0180CB8	Silent Night	Monster Ballads X-Mas	Faster Pussy cat	2003	2	15
SOVFVAK12A8C1350D9	Tanssi vaan	Karkuteillä	Karkkiautomaatti	1995	2	7
SOGTUKN12AB017F4F1	No One Could Ever	Butter	Hudson Mohawke	2006	4	18
SOBNYVR12A8C13558C	Si Vos Querés	De Culo	Yerba Brava	2003	3	15
SOHSBXH12A8C13B0DF	Tangle Of Aspens	Rene Ablaze Presents Winter Sessions ...	Der Mystic	None	3	-1988

my_rating	love_count
1.0	0
1.0	0
1.0	0
1.0	0
1.0	0

[5 rows x 9 columns]

To show that normal operations work on columns with missing values, we will do the 'how_old_was_i' transformation again.

In [22]:

```
song_sf['how_old_was_i'] = song_sf['year'].apply(lambda x : x - year_i_was_born)
song_sf.head(5)
```

Out[22]:

song_id	title	release	artist_name	year	title_length	how_old_was_i
SOQMMHC12AB0180CB8	Silent Night	Monster Ballads X-Mas	Faster Pussy cat	2003	2	15
SOVFVAK12A8C1350D9	Tanssi vaan	Karkuteillä	Karkkiautomaatti	1995	2	7
SOGTUKN12AB017F4F1	No One Could Ever	Butter	Hudson Mohawke	2006	4	18
SOBNYVR12A8C13558C	Si Vos Querés	De Culo	Yerba Brava	2003	3	15
SOHSBXH12A8C13B0DF	Tangle Of Aspens	Rene Ablaze Presents Winter Sessions	Der Mystic	None	3	None
		...				

my_rating	love_count
1.0	0
1.0	0
1.0	0
1.0	0
1.0	0

[5 rows x 9 columns]

However, if I actually did want to filter out these missing values, that is easy too.

In [23]:

```
song_sf_valid_years = song_sf[song_sf['year'] > 0]
print "Length of trimmed data: " + str(len(song_sf_valid_years))
song_sf_valid_years.head(5)
```

```
Length of trimmed data: 515576
```

Out[23]:

song_id	title	release	artist_name	year	title_length	how_old_was_i
SOQMMHC12AB0180CB8	Silent Night	Monster Ballads X-Mas	Faster Pussy cat	2003	2	15
SOVFVAK12A8C1350D9	Tanssi vaan	Karkuteillä	Karkkiautomaatti	1995	2	7
SOGTUKN12AB017F4F1	No One Could Ever	Butter	Hudson Mohawke	2006	4	18
SOBNYVR12A8C13558C	Si Vos Querés	De Culo	Yerba Brava	2003	3	15
SOEYRFT12AB018936C	2 Da Beat Ch'yall	Da Bomb	Kris Kross	1993	4	5

my_rating	love_count
1.0	0
1.0	0
1.0	0
1.0	0
1.0	0

[5 rows x 9 columns]

What I'm showing off here is that we can filter an SFrame by an SArray, where only the SFrame rows that correspond to the given SArray entries evaluating to True will make it through the filter. This happens when an SArray is given as the index of an SFrame. Furthermore, we can create a new SArray from an existing one by using any of the comparison operators. To execute this filter we did both of these things, but you can do them in isolation as well. Here I show the resulting SArray from running the '> 0' operation in isolation:

In [24]:

```
tmp = song_sf['year'] > 0
tmp
```

Out[24]:

```
dtype: int
Rows: 1000000
[1L, 1L, 1L, 1L, None, None, None, 1L, None, None, 1L, 1L, None, 1L, None, 1L, 1L, 1L, 1L, None, None, None, 1L, 1L, None, None, 1L, 1L, None, None,
None, None, None, 1L, 1L, 1L, None, 1L, None, 1L, None, 1L, None, 1L, 1L, 1L, None, 1L, None, None, None, None, 1L, 1L, 1L, None, None, 1L, None, No
ne, 1L, 1L, None, 1L, 1L, 1L, None, None, None, 1L, None, None, 1L, None, 1L, 1L, None, None, None, None, 1L, None, None, 1L, 1L, None, 1L, None, 1
L, None, None, None, 1L, 1L, 1L, 1L, None, 1L, 1L, 1L, ... ]
```

Keep in mind that the SArray must be the same length of the SFrame in order to filter. This also works with more complicated, chained filters with logical operators. Here's a list of songs that came out while I was in high school by a couple of my favorite bands in that period of my life:

In [25]:

```
my_fav_hs_songs = song_sf[((song_sf['artist_name'] == 'Relient K')
                           | (song_sf['artist_name'] == 'Streetlight Manifesto'))
                          & (song_sf['how_old_was_i'] >= 14) & (song_sf['how_old_was_i'] <= 18)]
my_fav_hs_songs
```

Out[25]:

song_id	title	release	artist_name	year	title_length
SOQNUHJ12A6D4F9E19	Let It All Out (Album Version) ...	MMHMM	Relient K	2004	6
SOIONAH12A58A76FD1	We Are The Few (Album Version) ...	Everything Goes Numb	Streetlight Manifesto	2003	6
SOUUYHK12A6D4F9E16	I So Hate Consequences (Album Version) ...	MMHMM	Relient K	2004	6
SOCBUJT12A6D4F9E1A	Who I Am Hates Who I've Been (mmhmm Album ...	MMHMM	Relient K	2004	10
SOBUHJR12A6D4FDC7C	Here's To Life (Album Version) ...	Everything Goes Numb	Streetlight Manifesto	2003	5
SONKBWG12A6D4FB91D	Giving Up_ Giving In (LP Version) ...	Keasbey Nights	Streetlight Manifesto	2006	6
SOSFAVU12A6D4FDC6A	Everything Went Numb (Album Version) ...	Everything Goes Numb	Streetlight Manifesto	2003	5
SORYDMW12A6D4FB923	This One Goes Out To.... (LP Version) ...	Keasbey Nights	Streetlight Manifesto	2006	7
SOFMBIT12A6D4F9E1C	This Week The Trend (Album Version) ...	MMHMM	Relient K	2004	6
SOINPKF12A6D4FDC75	A Better Place_ A Better Time (Album Version) ...	Everything Goes Numb	Streetlight Manifesto	2003	8

how_old_was_i	my_rating	love_count
16	1.0	0
15	1.0	0
16	1.0	0
16	1.0	0
15	1.0	0
18	1.0	0
15	1.0	0
18	1.0	0
16	1.0	0
15	1.0	0

[? rows x 9 columns]
Note: Only the head of the SFrame is printed. This SFrame is lazily evaluated.
You can use len(sf) to force materialization.

That's not all of them, but that's a pretty decent selection for a dataset of a million songs. Notice that I had to use the bitwise operators instead of the 'and'/'or' keyword. Python does not allow the overloading of logical operators, so remember to use the bitwise ones.

Descriptive Statistics

The descriptive statistics below are operations done on the SArray, and cannot be done on the SFrame.

In [26]:

```
# Look at lots of descriptive statistics of title_length
print "mean: " + str(song_sf['title_length'].mean())
print "std: " + str(song_sf['title_length'].std())
print "var: " + str(song_sf['title_length'].var())
print "min: " + str(song_sf['title_length'].min())
print "max: " + str(song_sf['title_length'].max())
print "sum: " + str(song_sf['title_length'].sum())
print "number of non-zero entries: " + str(song_sf['title_length'].nnz())
```

```
mean: 3.369894
std: 2.17226527587
var: 4.71873642876
min: 0
max: 47
sum: 3369894
number of non-zero entries: 999985
```

We can accomplish essentially the same thing by getting a sketch_summary on this column. This will give the exact values of the descriptive statistics I asked for above, and then give approximate values of some other useful stuff like quantiles and counts of unique values. These values are approximate because performing the real operation on a dataset that is larger than your memory size could exhaust your memory or take too long to compute. Each operation has well-defined bounds on how wrong the answer will be, which are listed in our API Reference (https://dato.com/products/create/docs/generated/graphlab.Sketch.html).

In [27]:

```
approx_sketch = song_sf['title_length'].sketch_summary()
print approx_sketch
```

```
+--------------------+---------------+----------+
|        item        |     value     | is exact |
+--------------------+---------------+----------+
|       Length       |    1000000    |   Yes    |
|        Min         |      0.0      |   Yes    |
|        Max         |      47.0     |   Yes    |
|        Mean        |    3.369894   |   Yes    |
|        Sum         |   3369894.0   |   Yes    |
|      Variance      | 4.71873642876 |   Yes    |
| Standard Deviation | 2.17226527587 |   Yes    |
|  # Missing Values  |       0       |   Yes    |
|  # unique values   |       44      |    No    |
+--------------------+---------------+----------+

Most frequent items:
+-------+--------+--------+--------+--------+-------+-------+-------+-------+
| value |   2    |   3    |   1    |   4    |   5   |   6   |   7   |   8   |
+-------+--------+--------+--------+--------+-------+-------+-------+-------+
| count | 241587 | 217674 | 164124 | 152569 | 92148 | 54674 | 30777 | 17826 |
+-------+--------+--------+--------+--------+-------+-------+-------+-------+
+-------+------+
|   9   |  10  |
+-------+------+
| 10465 | 6633 |
+-------+------+

Quantiles:
+-----+-----+-----+-----+-----+-----+-----+------+------+
|  0% |  1% |  5% | 25% | 50% | 75% | 95% | 99%  | 100% |
+-----+-----+-----+-----+-----+-----+-----+------+------+
| 0.0 | 1.0 | 1.0 | 2.0 | 3.0 | 4.0 | 7.0 | 11.0 | 47.0 |
+-----+-----+-----+-----+-----+-----+-----+------+------+
```

Saving the return value from sketch_summary gives you a graphlab.Sketch object, which can be queried further (details here (https://dato.com/products/create/docs/generated/graphlab.Sketch.html)). Here, I can drill deeper into those quantiles:

In [28]:

```
print approx_sketch.quantile(.25)
print approx_sketch.quantile(.75)
print approx_sketch.quantile(.993)
print approx_sketch.quantile(.995)
print approx_sketch.quantile(.997)
```

```
2.0
4.0
12.0
13.0
14.0
```

But wow...47 words?!? I gotta see what that song is.

In [29]:

```
top10_titles = song_sf.topk('title_length')
top10_titles
```

Out[29]:

song_id	title	release	artist_name	year
SOAALOO12AC468C4ED	Resolution Island Suite I) A Vessel Sublime II) ...	Allegory Of Hearing	Roy Montgomery	None
SOSEWOR12AB018BDF3	Guayacan Mix: Amor Traicionero / Te Amo_ Te ...	Como en un baile	Guayacan	None
SOEBPPZ12AB0183730	Son Of Scheherazade: Pt.1- Fanfare; Pt 2 - ...	British Tour '76	Renaissance	None
SODRVPW12A8C13DDCB	And by our own hand did every last bird lie ...	Every Red Heart Shines Toward the Red Sun ...	Red Sparowes	2006
SODEYRT12A8AE47972	Any Place I Hang My Hat is Home - On the ...	Too Marvelous For Words - The Songs Of Johnny ...	Lee Lessack	None
SOEICJI12AC3DFAAD4	Throw Away Comedy Medley: Dance With A Dolly / You ...	Live At The Sands	Dean Martin	None
SORHPYP12AB017C661	Happenings' Medley: Oh! The Grand Old Duke Of ...	The Cat & The Fiddle - 66 Nursery Rhyme Favourites ...	The Mother Goose Singers	None
SOKNNWV12AB017B473	If I Had My Way/Irish Rose/Daisy/Down Our ...	1999 International Barbershop Quartet ...	Swing City	None
SOWKFQF12AAA8C85E4	Manitoba Ne Répond Plus (Comme Un Lego_ Dans Un ...	Manitoba Ne Répond Plus	Gérard Manset	2008
SONMHGO12A6D4F8E6A	Marcha Do Cordão Da Bola Preta / Me Dá Dinheiro ...	Elizeth No Bola Preta	Elizeth Cardoso	None

title_length	how_old_was_i	my_rating	love_count
47	None	1.0	0
46	None	1.0	0
46	None	1.0	1
45	18	1.0	0
44	None	1.0	0
44	None	1.0	1
40	None	1.0	0
40	None	1.0	1
39	20	1.0	0
39	None	1.0	0

[10 rows x 9 columns]

In [30]:

```
top10_titles['title'][0]
```

Out[30]:

```
'Resolution Island Suite I) A Vessel Sublime II) And But A Gentle Swell III) Hubris Fills The Rash And Young Iv) Now The Reef-Dashed Mariner V) The Sirens_ They Feel Pity Vi) Wind Upon The Sails_ Light Upon The Sea Vii) Cast Away The Island_ Cruel S'
```

Makes sense...looks like a song with several movements. I'm somewhat curious about the titles with no words too.

In [31]:

```
song_sf.topk('title_length', k=5, reverse=True)
```

Out[31]:

song_id	title	release	artist_name	year	title_length	how_old_was_i	my_rating	love_cou
SOAGRAA12AB018D567		Puce de luxe	Sébastien Roch	None	0	None	1.0	0
SOOAFJX12AB018A028		Puce de luxe	Sébastien Roch	None	0	None	1.0	0
SOLDTFD12AB018AFE6		Puce de luxe	Sébastien Roch	None	0	None	1.0	0
SOASSAS12AB018AFCF		Puce de luxe	Sébastien Roch	None	0	None	1.0	0
SOQUGMS12AB018B01D		Puce de luxe	Sébastien Roch	None	0	None	1.0	0

[5 rows x 9 columns]

Here are a couple boolean operations too, with which I can prove that there were, in fact, songs before I was born. Just not all of them.

In [32]:

```
before_i_was_born = song_sf['how_old_was_i'] < 0
before_i_was_born.all(), before_i_was_born.any()
```

Out[32]:

```
(False, True)
```

Perhaps let's try some deeper analysis, like what albums have the most songs?

In [33]:

```
song_sf.groupby(['artist_name', 'release'], {'num_songs_in_album' : gl.aggregate.COUNT}).topk('num_songs_in_album')
```

Out[33]:

artist_name	release	num_songs_in_album
Fanny	First Time In A Long Time: The Reprise ...	85
Bernard Herrmann	The Twilight Zone	81
Spanky & Our Gang	The Complete Mercury Recordings ...	75
The Smashing Pumpkins	Rarities & B-Sides	72
Big Star	Keep An Eye On The Sky	71
Jacques Dutronc	Intégrale Les Cactus	69
The Stooges	1970: The Complete Fun House Sessions ...	67
Lull	Moments	64
Willie Clancy	Willie Clancy The Gold Ring ...	61
Jack Dangers	Forbidden Planet Explored / Sci-Fi Sound Effects ...	60

[10 rows x 3 columns]

Our groupby function only supports aggregation after grouping. The aggregation functions you can use are listed here (https://dato.com/products/create/docs/generated/graphlab.data_structures.html#module-graphlab.aggregate).

You can only go so far in analyzing this data though. We might want to match this data with user information, like how many times a certain person played one of these songs. For that, we need to the join function, but first we need to read this data in as an SFrame:

In [34]:

```
usage_data = gl.SFrame.read_csv("http://s3.amazonaws.com/dato-datasets/millionsong/10000.txt", header=False, delimiter='\t', column_type_hints={'X
3':int})
usage_data.rename({'X1':'user_id', 'X2':'song_id', 'X3':'listen_count'})
```

```
PROGRESS: Downloading http://s3.amazonaws.com/dato-datasets/millionsong/10000.txt to C:/Users/Evan/AppData/Local/Temp/graphlab-Evan/6216/bb0b5555-6152-493b-b13c-a4a68e0d376a.txt
PROGRESS: Finished parsing file http://s3.amazonaws.com/dato-datasets/millionsong/10000.txt
PROGRESS: Parsing completed. Parsed 100 lines in 0.680234 secs.
PROGRESS: Read 844838 lines. Lines per second: 1.19595e+006
PROGRESS: Finished parsing file http://s3.amazonaws.com/dato-datasets/millionsong/10000.txt
PROGRESS: Parsing completed. Parsed 2000000 lines in 1.12524 secs.
```

Out[34]:

user_id	song_id	listen_count
b80344d063b5ccb3212f76538f3d9e43d87dca9e ...	SOAKIMP12A8C130995	1
b80344d063b5ccb3212f76538f3d9e43d87dca9e ...	SOBBMDR12A8C13253B	2
b80344d063b5ccb3212f76538f3d9e43d87dca9e ...	SOBXHDL12A81C204C0	1
b80344d063b5ccb3212f76538f3d9e43d87dca9e ...	SOBYHAJ12A6701BF1D	1
b80344d063b5ccb3212f76538f3d9e43d87dca9e ...	SODACBL12A8C13C273	1
b80344d063b5ccb3212f76538f3d9e43d87dca9e ...	SODDNQT12A6D4F5F7E	5
b80344d063b5ccb3212f76538f3d9e43d87dca9e ...	SODXRTY12AB0180F3B	1
b80344d063b5ccb3212f76538f3d9e43d87dca9e ...	SOFGUAY12AB017B0A8	1
b80344d063b5ccb3212f76538f3d9e43d87dca9e ...	SOFRQTD12A81C233C0	1
b80344d063b5ccb3212f76538f3d9e43d87dca9e ...	SOHQWYZ12A6D4FA701	1

[2000000 rows x 3 columns]
Note: Only the head of the SFrame is printed.
You can use print_rows(num_rows=m, num_columns=n) to print more rows and columns.

I could just join the listen data with the song data, but maybe I'll do something a bit more interesting. Let's find out how many users from this dataset have listened to any one of those songs from my high school times, compared to the total number of users. First we need the total number of users:

In [35]:

```
num_users = len(usage_data['user_id'].unique())
print num_users
```

```
76353
```

In [36]:

```
fav_hs_listen_data = my_fav_hs_songs.join(usage_data, 'song_id')
num_fav_hs_users = len(fav_hs_listen_data['user_id'].unique())
print num_fav_hs_users
print float(num_fav_hs_users) / float(num_users)
```

```
287
0.0037588568884
```

That's really small. Those other people don't know what they're missing. Maybe the small proportion is simply because I'm only using a list of 42 songs. For kicks, what is the most popular song of that set of songs?

In [37]:

```
most_popular = fav_hs_listen_data.groupby(['song_id'], {'total_listens':gl.aggregate.SUM('listen_count'),
                                                        'num_unique_users':gl.aggregate.COUNT('user_id')})
most_popular.join(song_sf, 'song_id').topk('total_listens',k=20)
```

Out[37]:

song_id	num_unique_users	total_listens	title	release
SOCVOVH12A6D4FB912	107	555	Keasbey Nights (LP Version) ...	Keasbey Nights
SOPSQOS12A6D4F9E15	79	207	High Of 75 (Album Version) ...	MMHMM
SOINPKF12A6D4FDC75	63	196	A Better Place_ A Better Time (Album Version) ...	Everything Goes Numb
SOCLCYG12A6D4FDC71	67	188	Point/Counterpoint (Album Version) ...	Everything Goes Numb
SOUUYHK12A6D4F9E16	79	171	I So Hate Consequences (Album Version) ...	MMHMM

artist_name	year	title_length	how_old_was_i	my_rating	love_count
Streetlight Manifesto	2006	4	18	1.0	0
Relient K	2004	5	16	1.0	0
Streetlight Manifesto	2003	8	15	1.0	0
Streetlight Manifesto	2003	3	15	1.0	0
Relient K	2004	6	16	1.0	0

[5 rows x 11 columns]

...and only 5 even got listens, but "Keasbey Nights" wins from this small subset. Now, suppose I was a cheater and wanted to make this look a little better? I'll pretend I am so you can see 'append' in action.

In [38]:

```
me = gl.SFrame({'user_id':['evan'],'song_id':['SOSFAVU12A6D4FDC6A'],'listen_count':[4000]})
usage_data = usage_data.append(me)
fav_hs_listen_data = my_fav_hs_songs.join(usage_data, 'song_id')
most_popular = fav_hs_listen_data.groupby(['song_id'], {'total_listens':gl.aggregate.SUM('listen_count'),
                                                        'num_unique_users':gl.aggregate.COUNT('user_id')})
most_popular.join(song_sf, 'song_id').topk('total_listens',k=20)
```

Out[38]:

song_id	num_unique_users	total_listens	title	release
SOSFAVU12A6D4FDC6A	1	4000	Everything Went Numb (Album Version) ...	Everything Goes Numb
SOCVOVH12A6D4FB912	107	555	Keasbey Nights (LP Version) ...	Keasbey Nights
SOPSQOS12A6D4F9E15	79	207	High Of 75 (Album Version) ...	MMHMM
SOINPKF12A6D4FDC75	63	196	A Better Place_ A Better Time (Album Version) ...	Everything Goes Numb
SOCLCYG12A6D4FDC71	67	188	Point/Counterpoint (Album Version) ...	Everything Goes Numb
SOUUYHK12A6D4F9E16	79	171	I So Hate Consequences (Album Version) ...	MMHMM

artist_name	year	title_length	how_old_was_i	my_rating	love_count
Streetlight Manifesto	2003	5	15	1.0	0
Streetlight Manifesto	2006	4	18	1.0	0
Relient K	2004	5	16	1.0	0
Streetlight Manifesto	2003	8	15	1.0	0
Streetlight Manifesto	2003	3	15	1.0	0
Relient K	2004	6	16	1.0	0

[6 rows x 11 columns]

Splitting and Sampling

We're almost done with the tour of features. For easy splitting into training and test sets, we have the random_split function:

In [39]:

```
# Randomly split data rows into two subsets
first_set, second_set = song_sf.random_split(0.8, seed = 1)
first_set.num_rows(), second_set.num_rows()
```

Out[39]:

```
(800135, 199865)
```

If you want to split on a predicate though, you'll have to do that manually.

In [40]:

```
songs_before = song_sf[song_sf['how_old_was_i'] < 0]
songs_after = song_sf[song_sf['how_old_was_i'] >= 0]
songs_before.num_rows(), songs_after.num_rows()
```

Out[40]:

```
(69968, 445608)
```

We can also get a random sample of the dataset.

In [41]:

```
pct37 = song_sf.sample(.37)
pct37.num_rows()
```

Out[41]:

```
370008
```

Other Cool Features

SArrays support lots of mathematical operations. They can be performed with a scalar

In [42]:

```
sa = gl.SArray([1,2,3])
sa2 = sa * 2
print sa2
```

```
[2L, 4L, 6L]
```

...or they can be performed element-wise with another SArray.

In [43]:

```
add = sa + sa2
div = sa / sa2
print add
print div
```

```
[3L, 6L, 9L]
[0.5, 0.5, 0.5]
```

You can also iterate over SArrays and SFrames. When iterating over an SFrame, the returned element is a Python dictionary.

In [44]:

```
for i in song_sf:
    if i['title_length'] >= 45:
        print "Whoa that's long!"
```

```
Whoa that's long!
Whoa that's long!
Whoa that's long!
Whoa that's long!
```

Saving Our Work

I think I'm done exploring this dataset, but I'd like to save it for later. There's a couple ways I can do this. I can save it to a csv:

In [45]:

```
song_sf.save('new_song_data.csv', format='csv')
```

Or I can just save it as an SFrame as I showed earlier.

In [46]:

```
song_sf.save('new_song_data_sframe')
```

And of course, we can do all of this on S3. Note that if you download this notebook and run it, you won't be able to save to our dato-datasets bucket. Simply set your AWS credentials and uncomment the code below (replacing our S3 bucket with yours) to see this in action.

In [47]:

```
# In order to save to S3, you will need to use your own bucket and your own credentials.
# You can set your AWS credentials using the below function:
# graphlab.aws.set_credentials(<access_key_id>, <secret_access_key>)

#song_sf.save('s3://dato-datasets/my_sframes/new_song_sframe')       # S3://<bucket-name>/<file-path>
```

Now to load an SFrame back, we use the handy 'load_sframe' function like before. This takes the name of the sframe's top level directory:

In [48]:

```
# The below will download about 78 MB.
#hello_again = gl.load_sframe('s3://dato-datasets/my_sframes/new_song_sframe')
```

SArrays can be saved in a similar fashion. That's it! Our API reference (https://dato.com/products/create/docs) covers every function associated with SFrames and SArrays in detail.

Contact Sales (https://dato.com/company/contact.html)
Contact Support (https://dato.com/support/create-support-ticket.html)
Request Academic Use (https://dato.com/download/academic.html)

(HTTPS://DATO.COM/DOWNLOAD/)

(https://dato.com/solutions/machine-learning-algorithms/data-matching.html)
Deep Learning (https://dato.com/solutions/machine-learning-algorithms/deep-learning.html)
Logistic Regression (https://dato.com/solutions/machine-learning-algorithms/logistic-regression.html)
Nearest Neighbors (https://dato.com/solutions/machine-learning-algorithms/nearest-neighbor.html)
Text Analysis (https://dato.com/solutions/machine-learning-algorithms/text-analysis.html)

cases/fraud-detection.html)
Sentiment Analysis (https://dato.com/solutions/use-cases/sentiment-analysis.html)

CONNECT WITH US

(https://[illegible]P3XX03Rl_nQAz0eqXXAYw)

SUBSCRIBE TO OUR DATA SCIENCE BLOG

Enter email to subscribe

SUBSCRIBE