Meta Musical Memes

Megan Jasek, James King, Sean Underwood

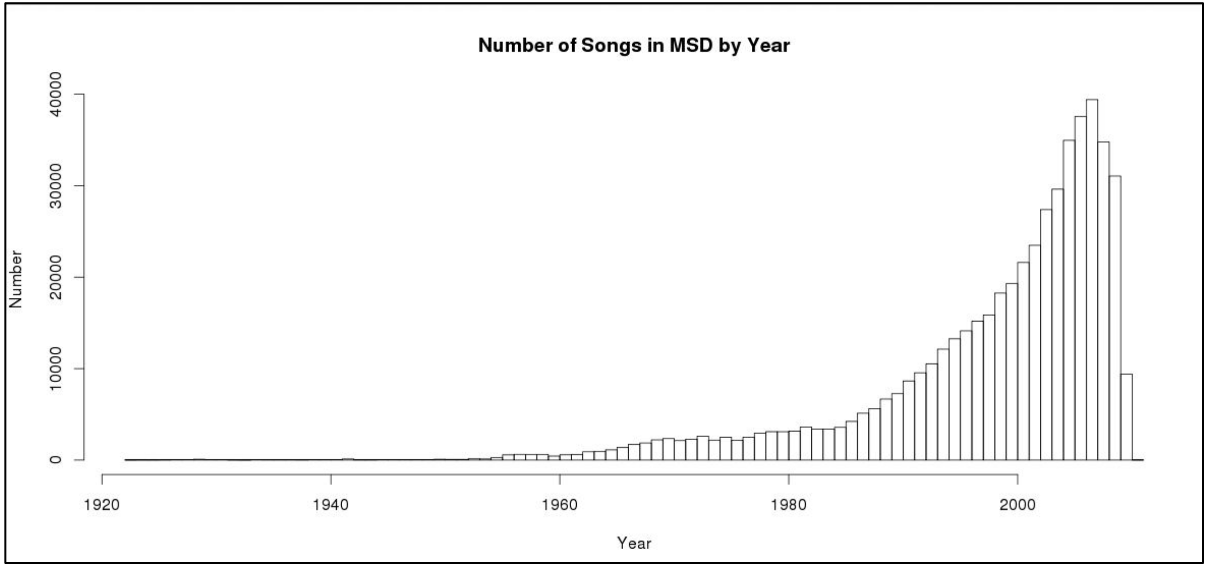
W205 Data Storage and Retrieval - Fall 2015

Problem Definition

Is there a basic formula that helps predict a song’s commercial success?

Data

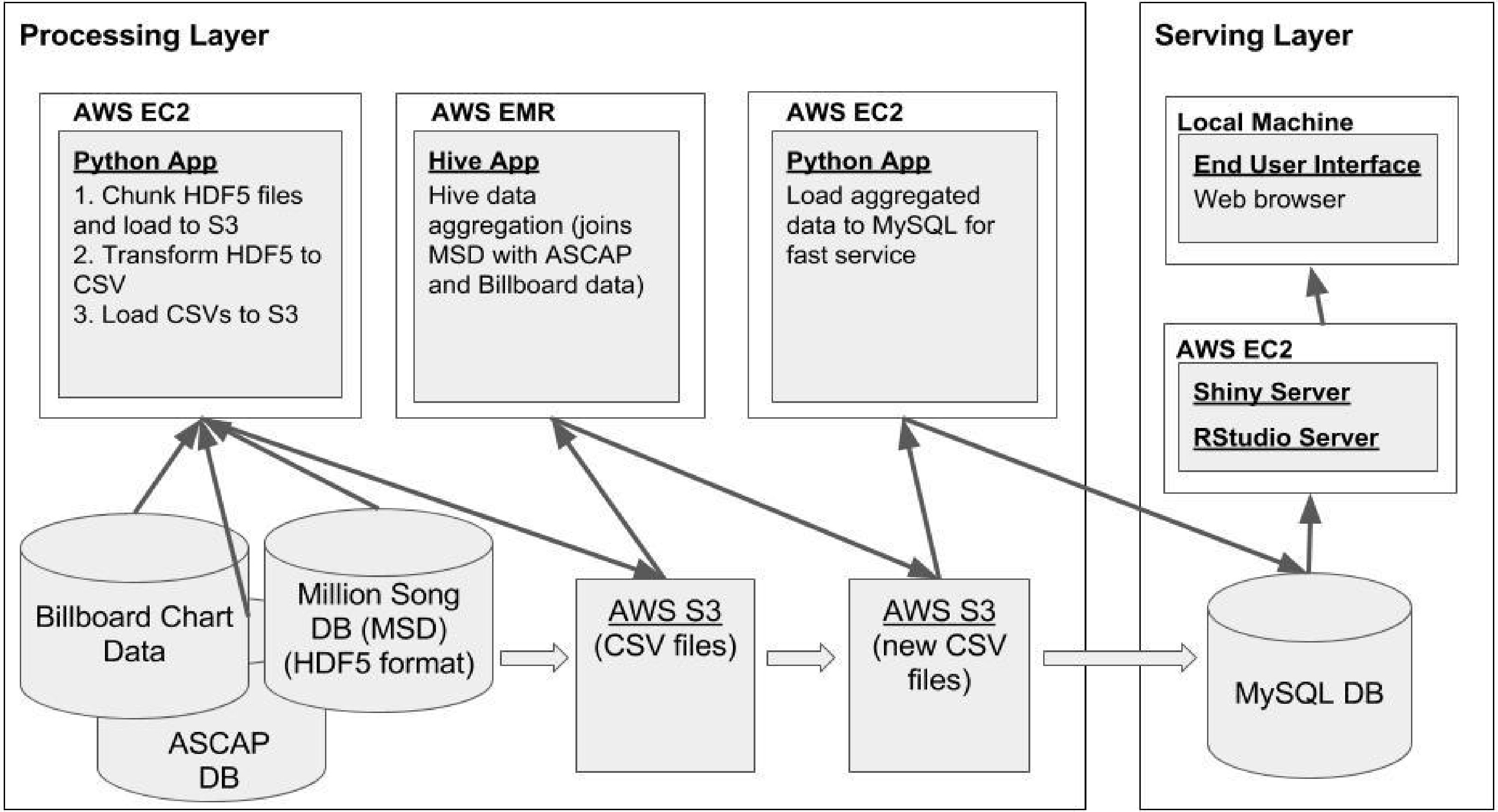
- Million Song Dataset
  - Song credits
  - Music theory info (key, tempo, etc.)
  - Audio features
  - Popularity estimates (“Hotttnesss”)
- Billboard charts
  - Performer credits
  - Peak position
  - Peak year
- ASCAP records
  - Writer credits



Data - Acquisition and Organization

- Million Song Dataset:
  - Provided by Columbia University’s LabROSA
  - Publicly available on Amazon AWS as an EBS Snapshot
  - File format: HDF5 (one file per song)
  - 500 GB
- Billboard Data
  - Fan-maintained
  - Excel format
  - 29 MB
- ASCAP Data
  - Provided by ASCAP
  - CSV format
  - 1.6 GB

Architecture



Processing Layer

AWS EC2

Python App

1. Chunk HDF5 files and load to S3
2. Transform HDF5 to CSV
3. Load CSVs to S3

AWS EMR

Hive App

Hive data aggregation (joins MSD with ASCAP and Billboard data)

AWS EC2

Python App

Load aggregated data to MySQL for fast service

Serving Layer

Local Machine

End User Interface

Web browser

AWS EC2

Shiny Server

RStudio Server

Processing Layer

- ETL: Million Song Dataset (500 GB)
    - HDF5 - Does not interact well with HDFS.
        - HDF5 is optimized for high performance when being used from a single process that does not need to load the whole file into memory at once.
        - When a file gets split across a block boundary, there’s no good way for workers to re-assemble the data.
        - h5py python library
    - Extraction - multiprocessing python library - run code in parallel
        - xLarge AWS instance with 8 VCPUs
        - Still took 12 hours to extract data
- ETL: Billboard Data (30 MB)
    - MS Excel file
    - Extracted 4 attributes (song, artist, peak position, peak year) using python
- ETL: ASCAP Data (1.5 GB)
    - CSV file in long format, unusual name formatting: “last middle first”
    - Extracted 3 attributes (song, artist, writer)

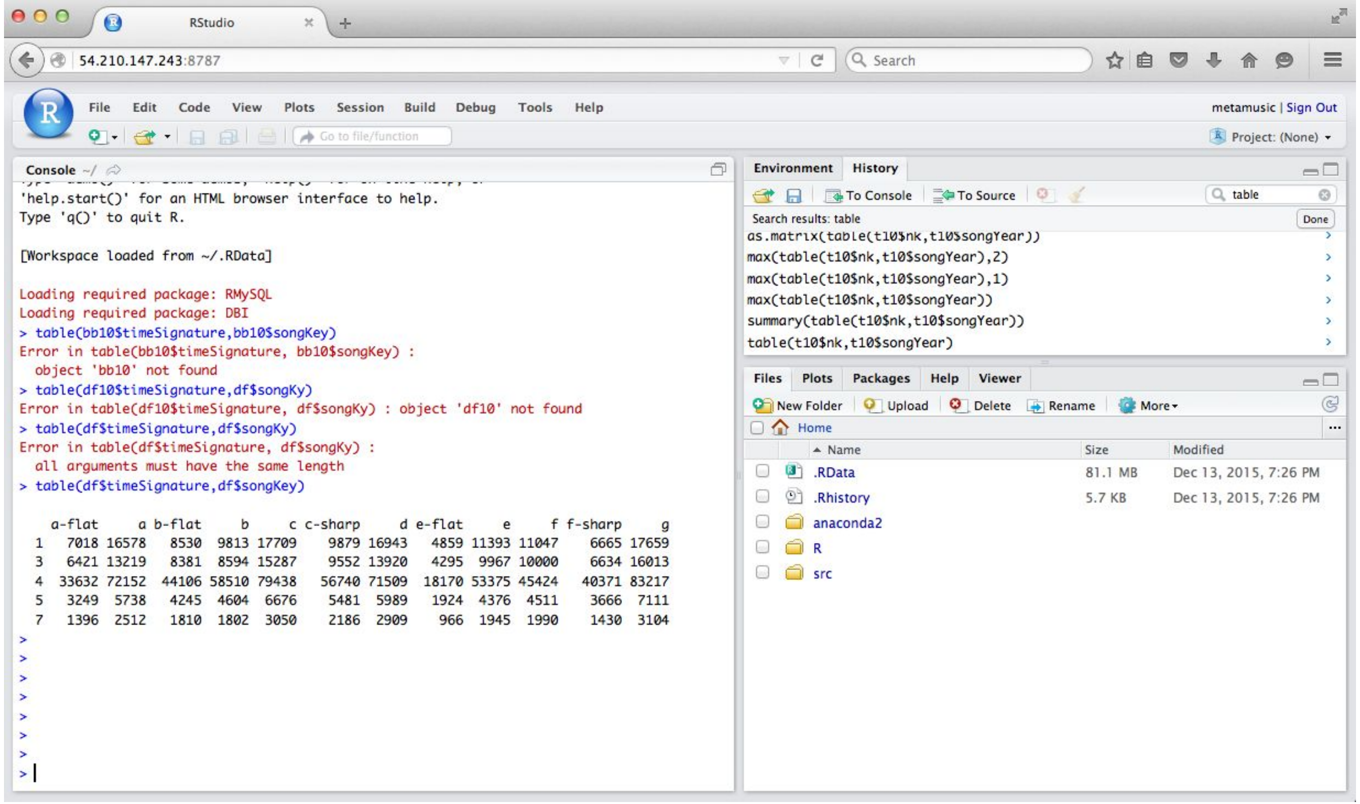
Processing Layer (cont.)

- Data Merge
  - All data converted to CSV format
  - Uploaded to Amazon S3
  - Amazon Elastic Map Reduce (EMR)
    - Hive job to clean and merge datasets
      - Massaged MSD 0 values
      - Converted numeric encodings for key and mode to text
    - Output uploaded to Amazon S3
    - 3 m3.xlarge instances - total time 5 minutes
  - Hive trickiness
    - Hive does not proactively validate data
    - It’s very important to manually validate
- ASCAP: There’s no escape from incomplete data
  - We failed to find a reliable way to bridge the lack of performer credits in the ASCAP data.
- Produced 2 datasets
  - One with ASCAP and one without

Serving Layer

- Merged data in MySQL DB
- Interactive Output
  - RStudio Server
    - Running on EC2 instance
    - Exposed full R ecosystem in a web browser
  - Shiny
    - Toolkit for writing web applications in R
    - Create interactive UI widgets hooked up to R commands
  - Shiny Server
    - Running on EC2 instance
    - Exposes Shiny applications in a web browser

RStudio Server

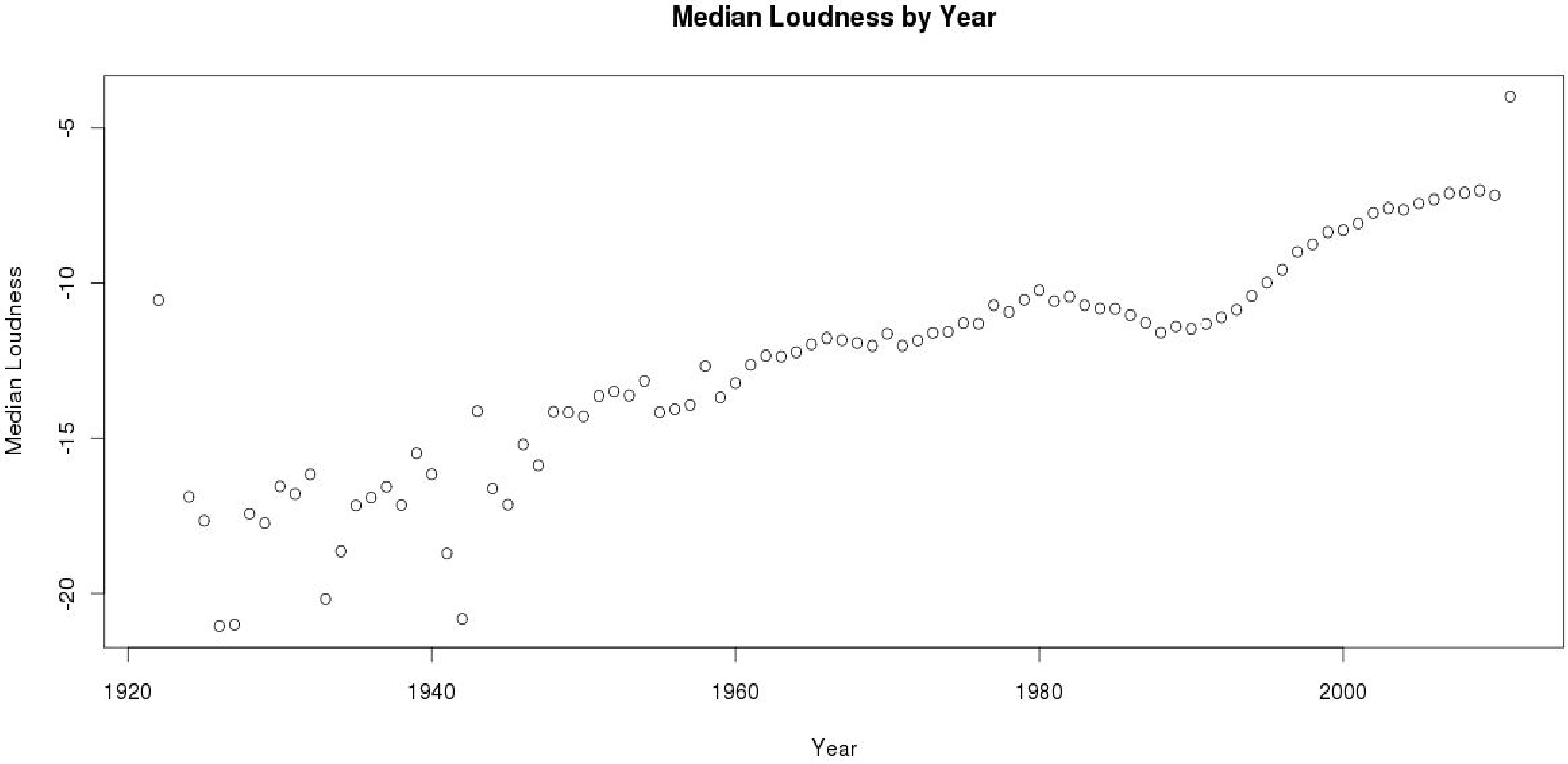


```
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

Loading required package: RMySQL
Loading required package: DBI
> table(bb10$timeSignature,bb10$songKey)
Error in table(bb10$timeSignature, bb10$songKey) : 
  object 'bb10' not found
> table(df10$timeSignature,df$songKy)
Error in table(df10$timeSignature, df$songKy) : object 'df10' not found
> table(df$timeSignature,df$songKy)
Error in table(df$timeSignature, df$songKy) : 
  all arguments must have the same length
> table(df$timeSignature,df$songKey)
   
    a-flat     a b-flat     b     c c-sharp     d e-flat     e     f f-sharp     g
  1   7018 16578   8530  9813 17709    9879 16943   4859 11393 11047    6665 17659
  3   6421 13219   8381  8594 15287    9552 13920   4295  9967 10000    6634 16013
  4  33632 72152  44106 58510 79438   56740 71509  18170 53375 45424   40371 83217
  5   3249  5738   4245  4604  6676    5481  5989   1924  4376  4511    3666  7111
  7   1396  2512   1810  1802  3050    2186  2909    966  1945  1990    1430  3104
> 
> 
> 
> 
> 
> 
> 
> 
```

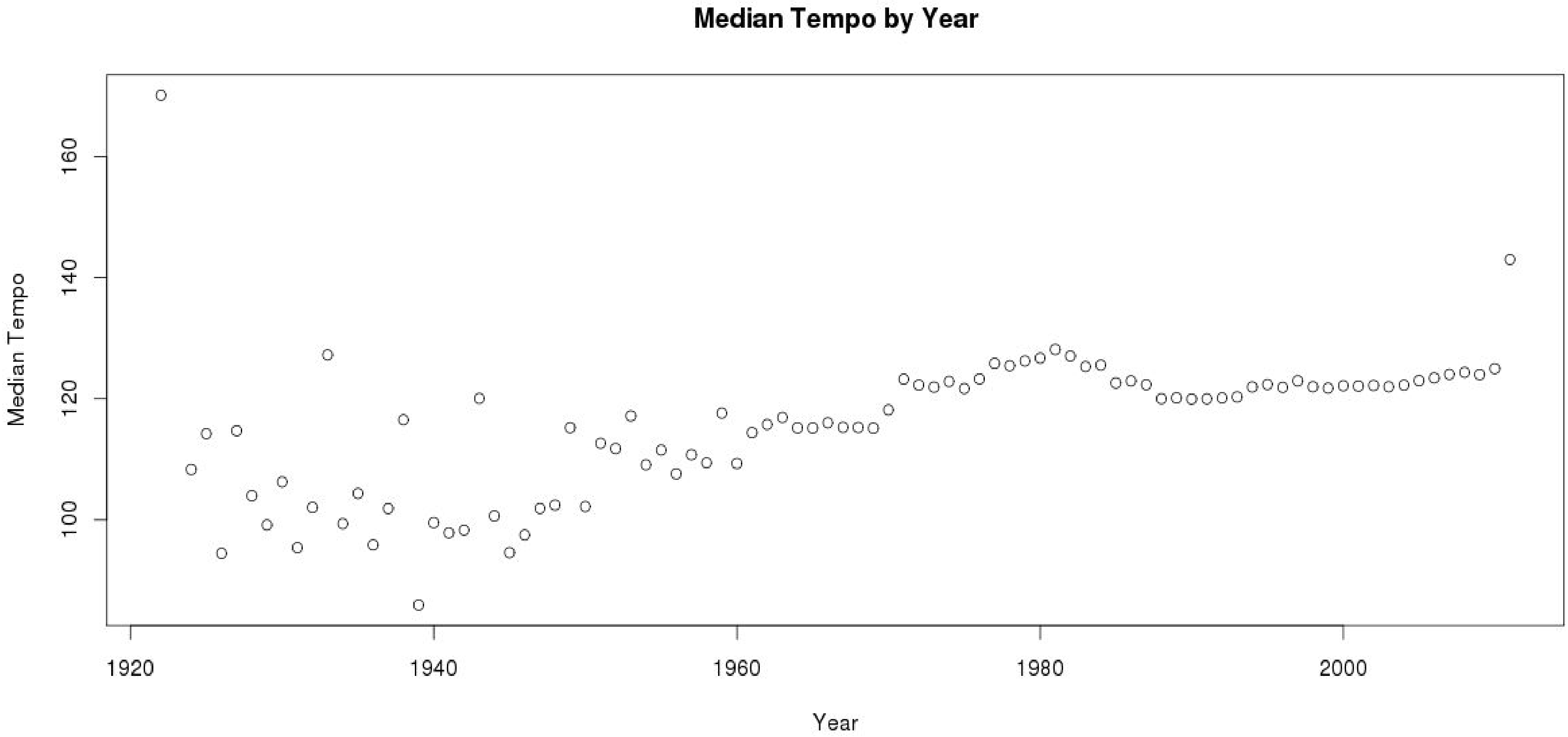
```
as.matrix(table(t10$nk,t10$songYear))
max(table(t10$nk,t10$songYear),2)
max(table(t10$nk,t10$songYear),1)
max(table(t10$nk,t10$songYear))
summary(table(t10$nk,t10$songYear))
table(t10$nk,t10$songYear)
```

	Name	Size	Modified
□	.RData	81.1 MB	Dec 13, 2015, 7:26 PM
□	.Rhistory	5.7 KB	Dec 13, 2015, 7:26 PM
□	anaconda2		
□	R		
□	src		

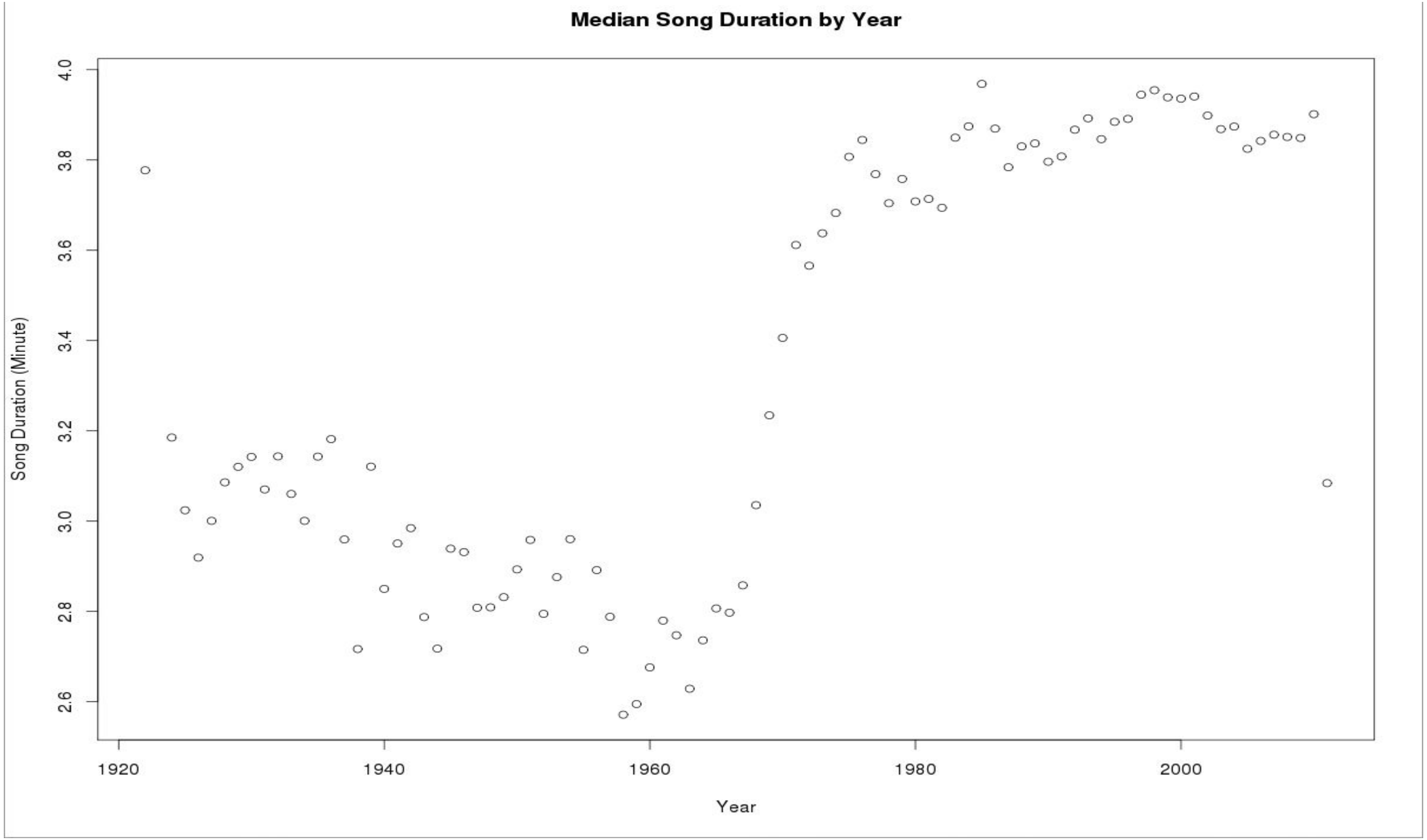
RStudio Results: Songs are Getting Louder…..What?



RStudio Results: Songs. Aren't. Getting. Any. Faster.

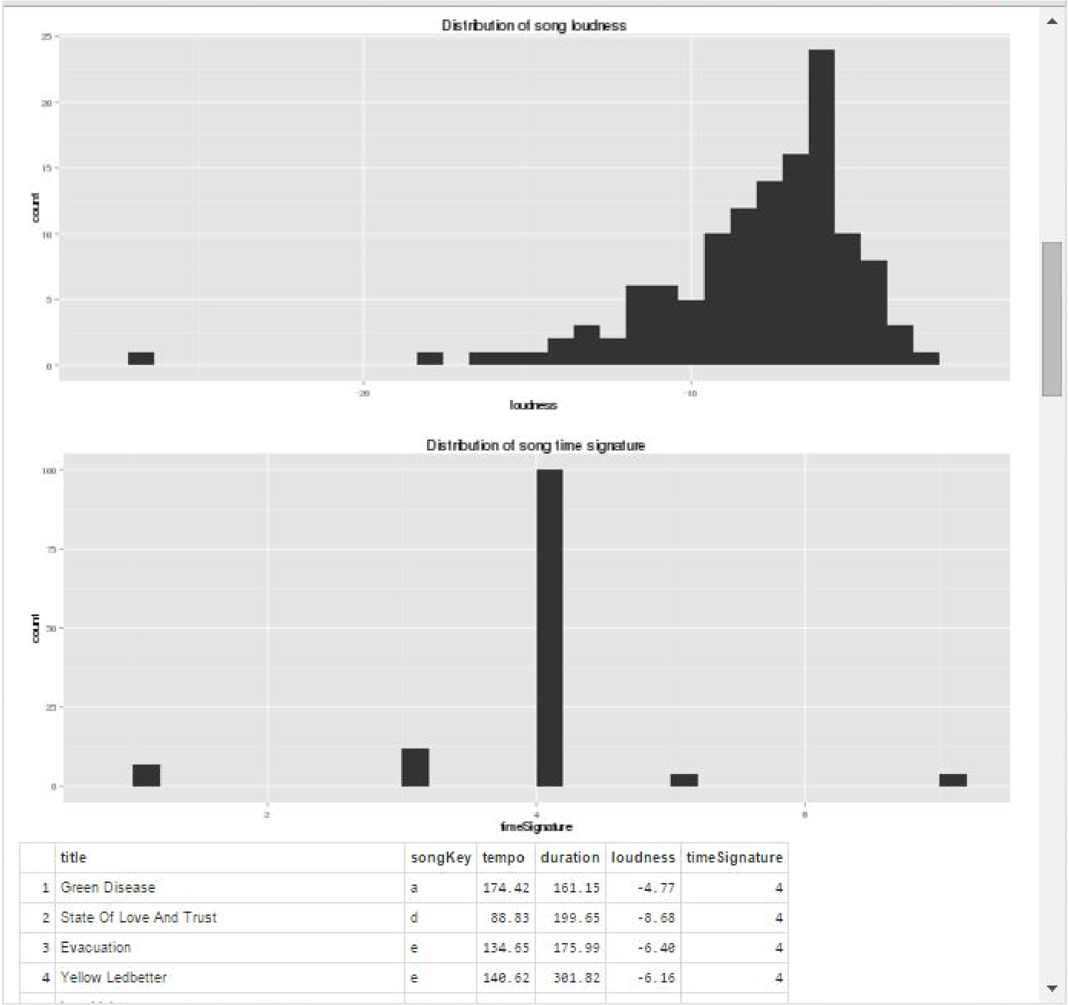


RStudio Results: Songs are Getting Lonnnnnnnger...



Analysis: Shiny Example 1:

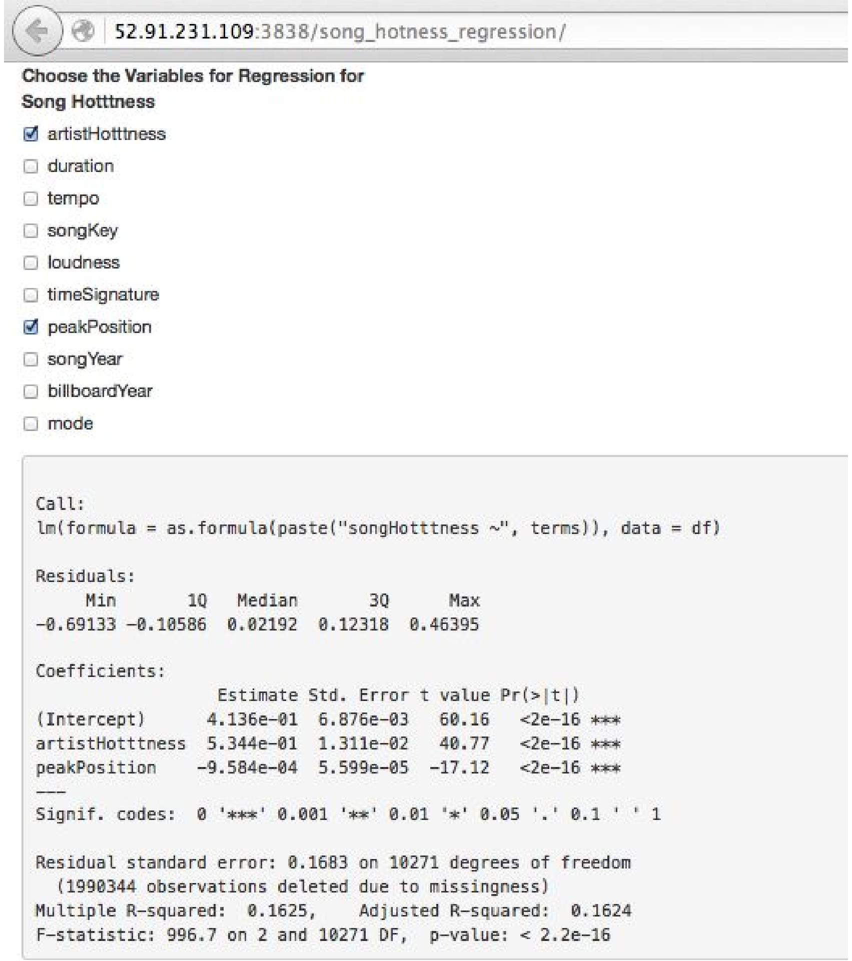
Song Statistics by Artist


Distribution of song loudness
count
25
20
15
10
5
0
-20
-10
loudness
Distribution of song time signature
count
100
75
50
25
0
2
4
6
timeSignature

	title	songKey	tempo	duration	loudness	timeSignature
1	Green Disease	a	174.42	161.15	-4.77	4
2	State Of Love And Trust	d	88.83	199.65	-8.68	4
3	Evacuation	e	134.65	175.99	-6.40	4
4	Yellow Ledbetter	e	140.62	301.82	-6.16	4

Analysis: Shiny Example 2:

Song Hotness Regression

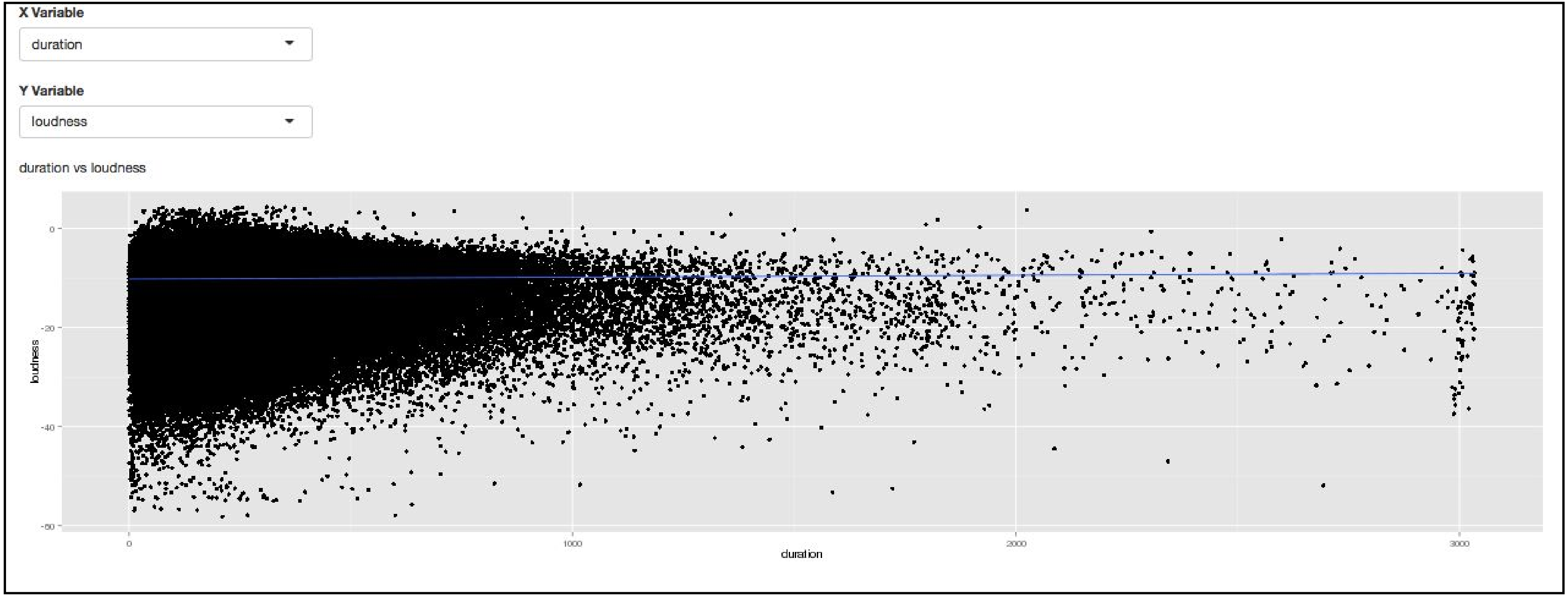


Choose the Variables for Regression for Song Hotttness

☑ artistHotttness
□ duration
□ tempo
□ songKey
□ loudness
□ timeSignature
☑ peakPosition
□ songYear
□ billboardYear
□ mode

```
Call:
lm(formula = as.formula(paste("songHotttness ~", terms)), data = df)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.69133 -0.10586  0.02192  0.12318  0.46395 

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)    
(Intercept)      4.136e-01  6.876e-03   60.16   <2e-16 ***
artistHotttness  5.344e-01  1.311e-02   40.77   <2e-16 ***
peakPosition    -9.584e-04  5.599e-05  -17.12   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1683 on 10271 degrees of freedom
  (1990344 observations deleted due to missingness)
Multiple R-squared:  0.1625,    Adjusted R-squared:  0.1624 
F-statistic: 996.7 on 2 and 10271 DF,  p-value: < 2.2e-16
```

Analysis: Shiny Example 3: Scatter Plots



X Variable

duration

Y Variable

loudness

duration vs loudness

Architecture - Alternatives

- Processing Layer
  - Leave the single-song HDF5 files as-is
    - The “small files problem” wasn’t nearly as problematic as the Hadoop/HDF5 problem.
  - Run the entire job in Spark
  - Convert HDF5 files to another format (e.g., JSON) from the outset
    - Human-readable
    - Hadoopable
    - Bigger data size, but ¯_(ツ)_/¯
- Serving Layer
  - DB - any relational database
  - BlinkDB - sampling database to speed up R visualizations
  - Tableau
  - Python with visualizations in D3

Future of Project - Scaling

- As expected, the Hive portions of the job scale very well.
- HDF5 file processing was largely sequential.
    - Scaling would require devising a way to store HDF5 files in a data lake
    - Alternatively, they could be converted to a more compatible format at import time
- Shiny had difficulty with the volume of data
    - R loads all data points into RAM, which imposes scaling limits
    - Regression and plotting of such large volumes of data is slow
    - Could instead use a sampling database such as BlinkDB to control the amount of data