

**whois**

# My name is Vincent

The screenshot shows a Reddit post titled "my linkedin profile" by user fishnets88. The post has 724 comments and was submitted 4 hours ago. The post content includes a large list of technologies and tools: R, python, javascript, shiny, dplyr, purrr, ditto, ggplot, d3, canvas, spark, sawk, pyspark, sparklyR, lodash, lazy, bootstrap, jupyter, vulpix, git, flask, numpy, pandas, feebas, scikit, pgm, bayes, h2o.ai, sparkling-water, tensorflow, keras, onyx, ekans, hadoop, scala, unity, metapod, gc, c#/c++, krebbase, neo4j, hadoop. Below the list, it says, "I typically ask recruiters to point out which of these are pokemon." The post is promoted by TamTamNL.



# whois

My name is Vincent  
I solve data problems, AMA!

- PyData Chair
- Rstudio Partner
- Meetup Organiser
- koaning.io
- bayesian

Fan of ING, thanks!  
for sponsoring ALL THE THINGS



# **FOR the HORde WoRLd of WaR-and SpaRKcRaft**



Vincent D. Warmerdam - GDD - koaning.io - @fishnets88

**AKA**

# **A Talk About Rlang: The Great Parts**

# **This R language**

Python people are like dog people.  
R people are like cat people.

The problem starts when a dog person looks at a cat  
expecting dog behavior.

'That is not how data science is supposed to work!'

— **Python User**

'Your dog is broken.'  
— **Python User**



## Paraprasing.

R is a language with strange parts, just like these cats that live in my house, but it more than compensates with some great parts.

I love python. It is a scripting language with great taste. But I really believe that I am better in my career in the field because I've invested enough time learning other languages.



# **Today**

My goal is to talk about the great parts today.

We'll see different backends in the mix.

We'll discuss how to deal with keras/spark.

We'll understand more advanced R tricks.

We'll even talk about the DSL for a different breed of ML.

# **Today**

My goal is to talk about the great parts today.

We'll see different backends in the mix.

We'll discuss how to deal with keras/spark.

We'll understand more advanced R tricks.

We'll even talk about the DSL for a different breed of ML.

There will also be special announcements at the end.

# **Today**

My goal is to talk about the great parts today.

We'll see different backends in the mix.

We'll discuss how to deal with keras/spark.

We'll understand more advanced R tricks.

We'll even talk about the DSL for a different breed of ML.

There will also be special announcements at the end.

Oh, and a fun dataset.

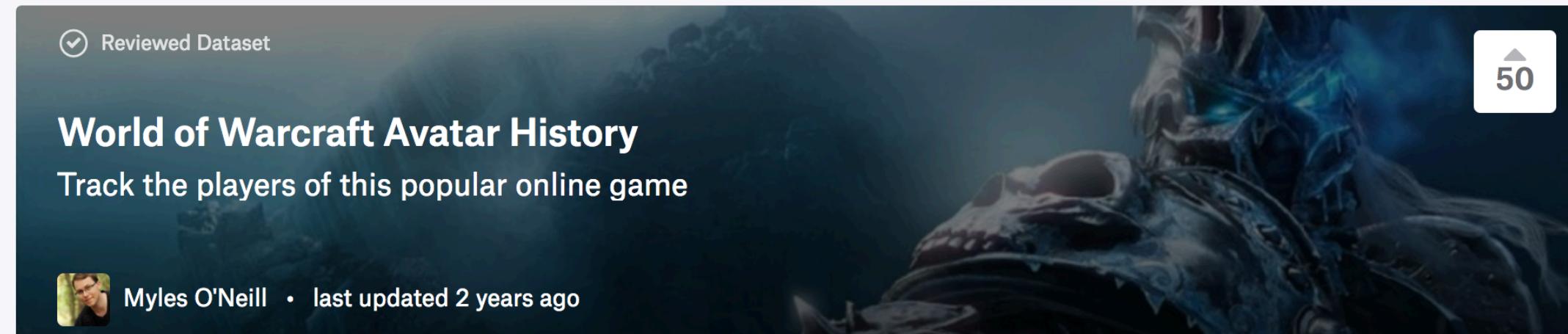
Reviewed Dataset

# World of Warcraft Avatar History

Track the players of this popular online game

Myles O'Neill • last updated 2 years ago

Overview Data Kernels Discussion Activity Download (88 MB) New Kernel



A small white box in the top right corner contains the number '50' with an upward arrow icon.

Tags video games games and toys medium featured

Top Contributors >

Rank	User	Contributor
1st		Thiago Balbo
2nd		33Vito
3rd		dmi3kno

Kernels >

Kernel Name	Run Time	Votes
Wow Dataset - Exploratory A...	run 2 years ago	51 votes
A World of Warcraft	run 2 years ago	22 votes
A Bipartite Chord Diagram fo...	run 2 years ago	7 votes

Discussion >

Topic	Run Time	Replies
In Tableau	a year ago	1 reply
Use this dataset in educatio...	a year ago	3 replies
Session parser?	2 years ago	2 replies

Description Help us describe this dataset Edit

## Overview

The World of Warcraft Avatar History Dataset is a collection of records that detail information about player characters in the game over time. It includes information about their character level, race, class, location, and social guild. The Kaggle version of this dataset includes only the information from 2008 (and the dataset in general only includes information from the 'Horde' faction of players in the game from a single game server).

Vincent D. Warmerdam - [\[@fishnets88\]](#) - GoDataDriven + [koaning.io](#)

# Dataset Preview

# Source: table<df> [?? x 7]						
# Database: spark_connection						
# Ordered by: char, timestamp						
	char	level	race	charclass	zone	timestamp
	<int>	<int>	<chr>	<chr>	<chr>	<dttm>
1	2	18	Orc	Shaman	The Barrens	2008-12-03 10:41:47
2	7	54	Orc	Hunter	Feralas	2008-01-15 21:47:09
3	7	54	Orc	Hunter	Un'Goro Crater	2008-01-15 21:56:54
4	7	54	Orc	Hunter	The Barrens	2008-01-15 22:07:23
5	7	54	Orc	Hunter	Badlands	2008-01-15 22:17:08
6	7	54	Orc	Hunter	Badlands	2008-01-15 22:26:52
7	7	54	Orc	Hunter	Badlands	2008-01-15 22:37:25
8	7	54	Orc	Hunter	Swamp of Sorrows	2008-01-15 22:47:10
9	7	54	Orc	Hunter	The Temple of Atal'Hakkar	2008-01-15 22:56:53
10	7	54	Orc	Hunter	The Temple of Atal'Hakkar	2008-01-15 23:07:25

# **Dataset Stats**

Data from a single World of Warcraft Server.

- 37,354 players
- 10,826,734 rows
- min\_timestamp = 2008-01-01 00:02:04
- max\_timestamp = 2008-12-31 23:50:18

# Stats Query

Generating these stats in R is a breeze. For example:

```
df %>%  
  summarise(maxdate = max(timestamp),  
            mindate = min(timestamp),  
            n_char = n_distinct(char),  
            n = ())
```

# Stats Query

```
df %>%  
  summarise(maxdate = max(timestamp),  
            mindate = min(timestamp),  
            n_char = n_distinct(char),  
            n = ())
```

There's two interesting parts in this query though.

The first part is this %>% operator.

# **Modern R code: %>%-operator**

To get these verbs to work, it helps to explain the %>%.

```
money <- function(amount, interest){  
  amount * (1 + interest)  
}
```

Then the %>% operator makes the following statements equivalent.

```
money(100, 3)  
100 %>% money(3)
```

# **Modern R code: %>%-operator**

Why is this such a great deal? Compare:

```
money(money(money(money(100, 3),1),2),1)
```

```
100 %>%  
  money(3) %>%  
  money(1) %>%  
  money(2) %>%  
  money(1)
```

One can be read from top to bottom, left to right ...

# Why this is nice: keRas

Yep, R has support for that nowadays.

```
model <- keras_model_sequential() %>%
  layer_input(input_shape = c(784)) %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'sigmoid') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = 'softmax')
```

It is nice and readable.

## **Modern R code: dplyr**

The main usecase of %>% is dplyr though.

```
ddf %>%  
  group_by(charclass, race) %>%  
  summarise(n = n_distinct(char),  
            mean_lvl = mean(level)) %>%  
  arrange(-n)
```

But there is something very strange about this query.  
What?

# **Modern R code: dplyr**

```
ddf %>%  
  group_by(charclass, race) %>%  
  summarise(n = n_distinct(char),  
            mean_lv1 = mean(level)) %>%  
  arrange(-n)
```

The char and level variables are not declared anywhere!

## Modern R code: dplyr

```
ddf %>%  
  group_by(charclass, race) %>%  
  summarise(n = n_distinct(char),  
            mean_lv1 = mean(level)) %>%  
  arrange(-n)
```

The char and level variables are not declared anywhere!

The internal trick that is used here is that such a code block is lazily evaluated. We can assign context to the variables that are not declared, later.

# Capture that AST

Example of this delayed evaluation.

```
> expr <- quo(x + y)
> rlang::eval_tidy(expr)
# Error: object 'x' not found
```

# Capture that AST

Example of this delayed evaluation.

```
> expr <- quo(x + y)
> rlang::eval_tidy(expr)
# Error: object 'x' not found
> x <- 1
> rlang::eval_tidy(expr)
# Error: object 'y' not found
```

# Capture that AST

Example of this delayed evaluation.

```
> expr <- quo(x + y)
> rlang::eval_tidy(expr)
# Error: object 'x' not found
> x <- 1
> rlang::eval_tidy(expr)
# Error: object 'y' not found
> y <- 2
> rlang::eval_tidy(expr)
[1] 3
```

# **Example of this trick.**

```
show_size <- function(dataf, ...){  
  exprs <- quos(...)  
  dataf %>%  
    group_by(!!!exprs) %>%  
    summarise(n = n())  
}
```

```
df %>% show_size(race)  
df %>% show_size(char)  
df %>% show_size(char, race)
```

# Modern R code: dplyr

```
ddf %>%  
  group_by(charclass, race) %>%  
  summarise(n = n_distinct(char),  
            mean_lvl = mean(level)) %>%  
  arrange(-n)
```

The internals are interesting, but let's get back to analysis.

	charclass	race	n	mean_lvl
	<chr>	<chr>	<dbl>	<dbl>
1	Warrior	Orc	3506	62.42852
2	Paladin	Blood Elf	3199	59.67628
	...			

# **Let's write something useful!**

We have a cool tool/language. Let's do some cool analytics.

- are people playing more in weekends?
- how long does it take to get to level 60?
- what things can we do to level up quicker?

For the next part I will discuss some analysis patterns using dplyr and what you need to do if the dataset becomes very large.

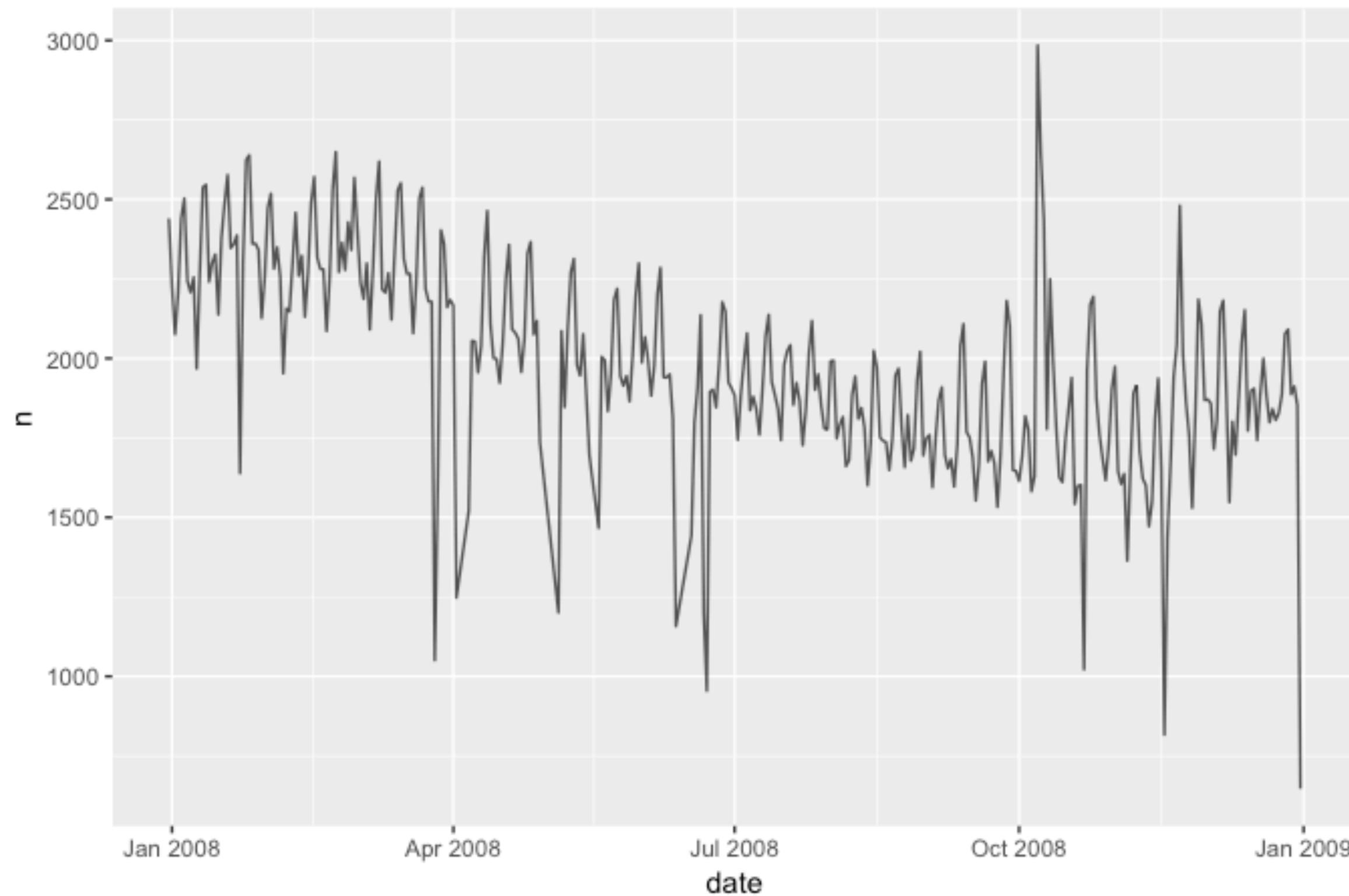
# Results

First make a query per date (good for plotting).

```
df <- df_all %>%
  group_by(date = date(timestamp)) %>%
  summarise(n = n_distinct(char))
```

Next let's look at the code that makes a plot.

```
ggplot() +
  geom_line(data=df, aes(date, n), alpha=0.5)
```



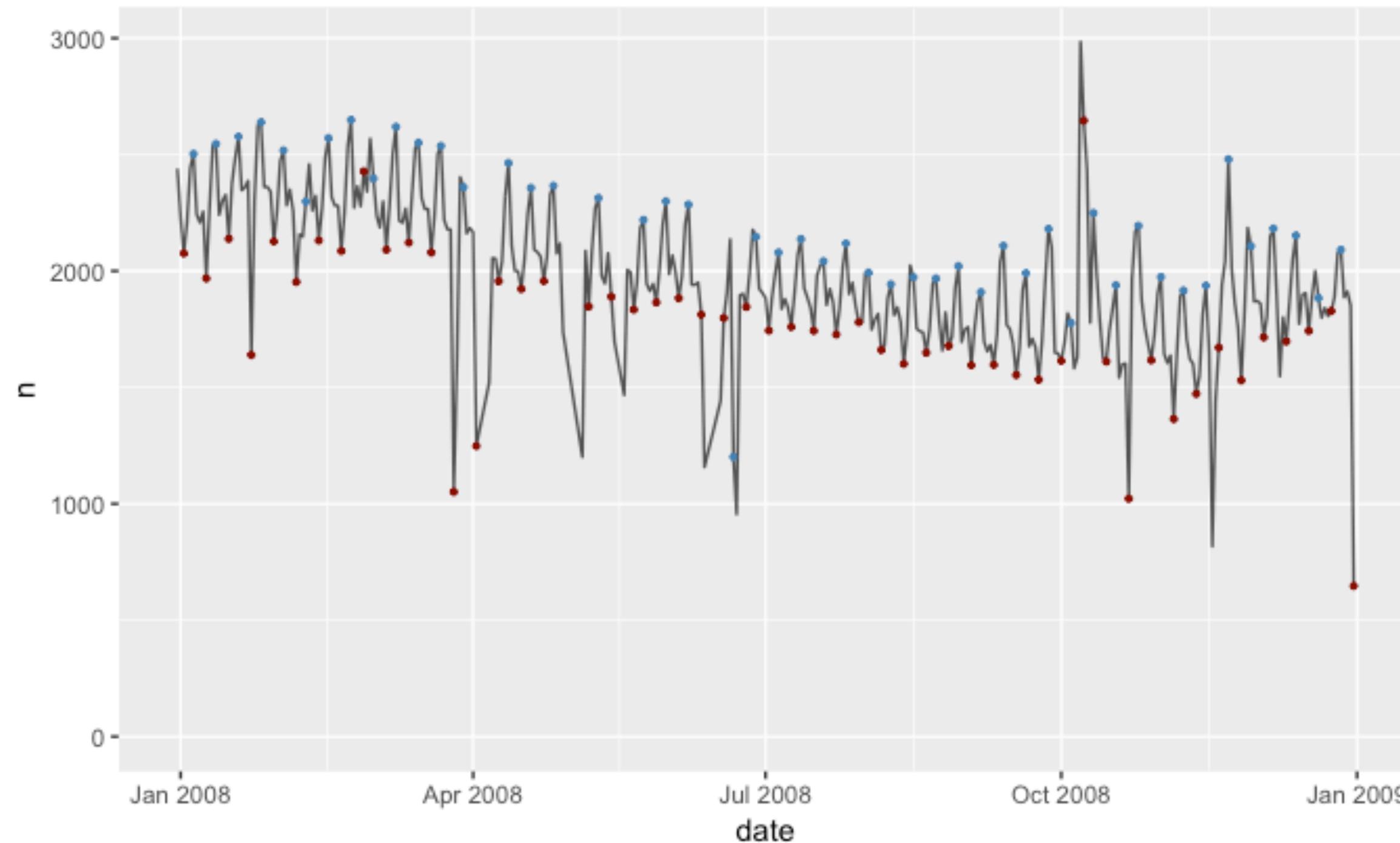
# Results

The chart is nice but it can be **much better**.

```
ggplot() +  
  geom_line(data=df, aes(date, n), alpha=0.7) +  
  geom_point(data=df %>% grab_only("Sat"),  
             aes(date, n), colour="steelblue", size=0.7) +  
  geom_point(data=df %>% grab_only("Wed"),  
             aes(date, n), colour="red4", size=0.7) +  
  ggtitle("WoW Characters Online over Time",  
          subtitle = "Note the weekly pattern.") +  
  ylim(0, NA)
```

## WoW Characters Online over Time

Note the weekly pattern, a small decline and a large spike in October.



# **Downside of Dataset**

Running this query took a fair amount of time, even though the dataset is only 600Mb. I want my investigation to be a bit more real-time if possible. So let's see if I can rewrite this in spark.

# Downside of Dataset

Running this query took a fair amount of time, even though the dataset is only 600Mb. I want my investigation to be a bit more real-time if possible. So let's see if I can rewrite this in spark.

"BUT WAIT!" I hear you say. That might mean that you'd need to rewrite all your code! We've seen that pandas code and pyspark code is very different!

# Downside of Dataset

Running this query took a fair amount of time, even though the dataset is only 600Mb. I want my investigation to be a bit more real-time if possible. So let's see if I can rewrite this in spark.

"BUT WAIT!" I hear you say. That might mean that you'd need to rewrite all your code! We've seen that pandas code and pyspark code is very different!

This statement is true but luckily for us, R solves all that for us. We'll see that in our next example.

## **Next Task: Sessionising**

In this next portion we'll try to calculate the total amount of time it might take to reach level 60. Before we can do that, we need to sessionize the dataset.

We'll implement all of this in `sparklyr` but let's first talk about what needs to happen.

# Next Task: Sessionising

	char	level	race	charclass	zone	guild	date	ts
1	9	70	Orc	Hunter	The Barrens	79	2008-01-01	2008-01-01 12:02:20
2	9	70	Orc	Hunter	The Barrens	79	2008-01-01	2008-01-01 12:12:07
3	9	70	Orc	Hunter	The Barrens	79	2008-01-01	2008-01-01 12:22:40
4	9	70	Orc	Hunter	The Barrens	79	2008-01-01	2008-01-01 12:32:29
5	9	70	Orc	Hunter	The Barrens	79	2008-01-01	2008-01-01 12:42:18
6	9	70	Orc	Hunter	The Barrens	79	2008-01-01	2008-01-01 12:52:47
7	9	70	Orc	Hunter	Ashenvale	79	2008-01-01	2008-01-01 13:02:29
8	9	70	Orc	Hunter	Ashenvale	79	2008-01-01	2008-01-01 13:12:18
9	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 13:22:44
10	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 13:32:32
11	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 16:02:31
12	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 16:12:18
13	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 16:22:44
14	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 16:32:32
15	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 16:42:20
16	9	70	Orc	Hunter	Blackfathom Deeps	79	2008-01-01	2008-01-01 16:52:08
17	9	70	Orc	Hunter	Shattrath City	79	2008-01-01	2008-01-01 17:02:43

# Next Task: Sessionising

		date		ts	diff_mins	new_session	session_id
1		2008-01-01	2008-01-01	12:02:20	NA	TRUE	1
2		2008-01-01	2008-01-01	12:12:07	10	FALSE	1
3		2008-01-01	2008-01-01	12:22:40	10	FALSE	1
4		2008-01-01	2008-01-01	12:32:29	10	FALSE	1
5		2008-01-01	2008-01-01	12:42:18	10	FALSE	1
6		2008-01-01	2008-01-01	12:52:47	10	FALSE	1
7		2008-01-01	2008-01-01	13:02:29	10	FALSE	1
8		2008-01-01	2008-01-01	13:12:18	10	FALSE	1
9		2008-01-01	2008-01-01	13:22:44	10	FALSE	1
10		2008-01-01	2008-01-01	13:32:32	10	FALSE	1
11		2008-01-01	2008-01-01	16:02:31	10	FALSE	1
12		2008-01-01	2008-01-01	16:12:18	120	TRUE	2
13		2008-01-01	2008-01-01	16:22:44	10	FALSE	2
14		2008-01-01	2008-01-01	16:32:32	10	FALSE	2
15		2008-01-01	2008-01-01	16:42:20	10	FALSE	2
16		2008-01-01	2008-01-01	16:52:08	10	FALSE	2
17		2008-01-01	2008-01-01	17:02:43	10	FALSE	2

# Simple Dplyr Statement

In this query `wowdf` is a local dataframe and it would do the trick we want.

```
wowdf %>%
  arrange(char, timestamp) %>%
  group_by(char) %>%
  mutate(unix_ts = as.integer(timestamp),
        unix_diff = unix_ts - lag(unix_ts)) %>%
  mutate(new_sess = ifelse(is.na(unix_diff), TRUE, unix_diff > 1000))
```

Let's now rewrite this into a spark query...

# Simple Dplyr Statement

In this query `wowdf` is not a local dataframe, it is a connection to a datasource, which could be ... anything SQL really ...

```
q <- wowddf %>%
  arrange(char, timestamp) %>%
  group_by(char) %>%
  mutate(unix_ts = unix_timestamp(timestamp),
        unix_diff = unix_ts - lag(unix_ts)) %>%
  mutate(new_sess = ifelse(is.na(unix_diff), TRUE, unix_diff > 1000))

explain(q)
```

# Translation towards SparkSql

```
SELECT
  `char`, `level`, `race`, `charclass`, `zone`,
  `guild`, `timestamp`, `unix_ts`, `unix_diff`,
  CASE WHEN (((`unix_diff`)) IS NULL)) THEN (TRUE) ELSE (`unix_diff` > 1000.0) END AS `new_sess`
FROM (
  SELECT
    `char`, `level`, `race`, `charclass`,
    `zone`, `guild`, `timestamp`, `unix_ts`,
    `unix_ts` - LAG(`unix_ts`, 1, NULL)
      OVER (PARTITION BY `char` ORDER BY `char`, `timestamp`) AS `unix_diff`
  FROM (
    SELECT `char`, `level`, `race`, `charclass`,
      `zone`, `guild`, `timestamp`,
      UNIX_TIMESTAMP(`timestamp`) AS `unix_ts`
    FROM (
      SELECT * FROM `df`
      ORDER BY `char`, `timestamp` ) `rkdvulqmdg` ) `efzujnwfsu` ) `dbcscslutz`
```

# Translation towards BigQuery

```
SELECT
    [char],[level],[race],[charclass],
    [zone],[guild],[timestamp],[unix_ts],[unix_diff],
    IF((([unix_diff]) IS NULL), TRUE, [unix_diff] > 1000.0) AS [new_sess]
FROM (
    SELECT
        [char],[level],[race],[charclass],
        [zone],[guild],[timestamp],[unix_ts],
        [unix_ts] - LAG([unix_ts], 1, NULL)
            OVER (PARTITION BY [char] ORDER BY [char], [timestamp]) AS [unix_diff]
    FROM (
        SELECT
            [char],[level],[race],[charclass],
            [zone],[guild],[timestamp],
            UNIX_TIMESTAMP([timestamp]) AS [unix_ts]
        FROM (
            SELECT * FROM [wow.cleansess]
            ORDER BY [char], [timestamp])))
```

## Dat AST

Wondering how this works?

Remember that trick with capturing an expression and being able to delay the evaluation? You can also take the expression and translate it.

This is what is happening internally. There's a bit of magic here and there in the tidyverse but internally this is something that plays a very large part.

## **Independant Backend**

Part of the design of tidyverse is to allow for a different backend. The user interface needs to be intuitive such that you as a developer can be very expressive without needing to worry about performance.

Currently, these backends are supported:

**data.frame, tibble, SQLite, PostgreSQL, Redshift**

## **Independant Backend**

Part of the design of dplyr is to allow for a different backend. The user interface needs to be intuitive such that you as a developer can be very expressive without needing to worry about performance.

Currently, these backends are supported:

**data.frame, tibble, SQLite, PostgreSQL, Redshift,  
MySQL, MariaDB, MonetDB, Presto, Spark**

## **Independant Backend**

Part of the design of dplyr is to allow for a different backend. The user interface needs to be intuitive such that you as a developer can be very expressive without needing to worry about performance.

Currently, these backends are supported:

**data.frame, tibble, SQLite, PostgreSQL, Redshift, MySQL, MariaDB, MonetDB, Presto, Spark, Hive, Impala, Vertica, Teradata, Google BigQuery**

## Few Downsides

There are **lots** of SQL dialects and it is impossible to fully support all of them via the dplyr spec.

## Silly Example

Not every engine handles things the same way. Spark will apply a `window.lag()` to any datatype while BigQuery will not allow for anything besides INT64 to be passed to the `LAG()` function.

## Few Upsides

Nevertheless. Holy cow, thats a whole lotta backends!

The functions that are passed will not be handled by R at all. This can be used as an advantage: R will do the SQL translation even when the function does not exist locally.

Let's consider an example.

## Spark/Hive Tricks

Can anybody see a function in this SQL query that does not belong to the tidyverse?

```
ddf %>%
  filter(level <= 60) %>%
  arrange(char, timestamp) %>%
  group_by(char) %>%
  mutate(unix_ts = unix_timestamp(timestamp),
        unix_diff = unix_ts - lag(unix_ts))
```

## Spark/Hive Tricks

Can anybody see a function in this SQL query that does not belong to the tidyverse?

```
ddf %>%
  filter(level <= 60) %>%
  arrange(char, timestamp) %>%
  group_by(char) %>%
  mutate(unix_ts = unix_timestamp(timestamp),
        unix_diff = unix_ts - lag(unix_ts))
```

Even though `unix_timestamp` does not exist, no error!

## Spark/Hive Tricks

Translation to Spark example.

```
ddf %>%
  filter(level <= 60) %>%
  arrange(char, timestamp) %>%
  group_by(char) %>%
  mutate(unix_ts = megatondinosaurhead(timestamp),
        unix_diff = unix_ts - lag(unix_ts))
```

This will translate to ...

```
SELECT
`char`, `level`, `race`, `charclass`, `zone`,
`guild`, `timestamp`, `unix_ts`, `unix_diff`,
CASE WHEN (((`unix_diff`)) IS NULL)) THEN (TRUE)
ELSE (`unix_diff` > 1000.0) END AS `new_sess`
FROM (
SELECT
`char`, `level`, `race`, `charclass`,
`zone`, `guild`, `timestamp`, `unix_ts`,
`unix_ts` - LAG(`unix_ts`, 1, NULL)
    OVER (PARTITION BY `char` ORDER BY `char`, `timestamp`) AS `unix_diff`
FROM (
SELECT `char`, `level`, `race`, `charclass`,
`zone`, `guild`, `timestamp`,
MEGATONDINOSAURHEAD(`timestamp`) AS `unix_ts`
FROM (
SELECT * FROM `df`
ORDER BY `char`, `timestamp` ) `rkdvulqmdg` ) `efzujnwfsu` ) `dbcscslutz`
```

## **Sparklyr Functions**

Why is this awesome?

All HIVE functions should be available to you if they are available from SparkSQL. You can see these functions defined [here](#).

R will try help you from rewriting dplyr code. If you use `as.character(colname)` in dplyr it will get translated to `CAST(colname AS STRING)` implicitly in the SQL. Anything not recognizes will be translated like before.

## All Session Code

```
agg_ddf <- ddf %>%
  filter(level <= 60) %>%
  arrange(char, timestamp) %>%
  group_by(char) %>%
  mutate(unix_ts = unix_timestamp(timestamp),
        unix_diff = unix_ts - lag(unix_ts)) %>%
  filter(unix_ts < 1220227200) %>%
  mutate(new_sess = ifelse(is.na(unix_diff), TRUE, unix_diff > 1000),
        session = new_sess %>% as.numeric() %>% cumsum()) %>%
  ungroup() %>%
  group_by(char, session, level, race, charclass, guild) %>%
  summarise(min_ts = min(unix_ts), max_ts = max(unix_ts)) %>%
  mutate(session_time = max_ts - min_ts) %>%
  ungroup() %>%
  group_by(char, level, race, charclass, guild = (guild != -1)) %>%
  summarise(level_time = sum(session_time)/60/60) %>%
  ungroup()
```

## Apply Session Part

```
agg_ddf <- ddf %>%
  filter(level <= 60) %>%
  arrange(char, timestamp) %>%
  group_by(char) %>%
  mutate(unix_ts = unix_timestamp(timestamp),
        unix_diff = unix_ts - lag(unix_ts)) %>%
  filter(unix_ts < 1220227200) %>%
  mutate(new_sess = ifelse(is.na(unix_diff), TRUE, unix_diff > 1000),
        session = new_sess %>% as.numeric() %>% cumsum()) %>%
  ungroup() %>%
  group_by(char, session, level, race, charclass, guild) %>%
  summarise(min_ts = min(unix_ts), max_ts = max(unix_ts)) %>%
  mutate(session_time = max_ts - min_ts) %>%
  ungroup() %>%
  group_by(char, level, race, charclass, guild = (guild != -1)) %>%
  summarise(level_time = sum(session_time)/60/60) %>%
  ungroup()
```

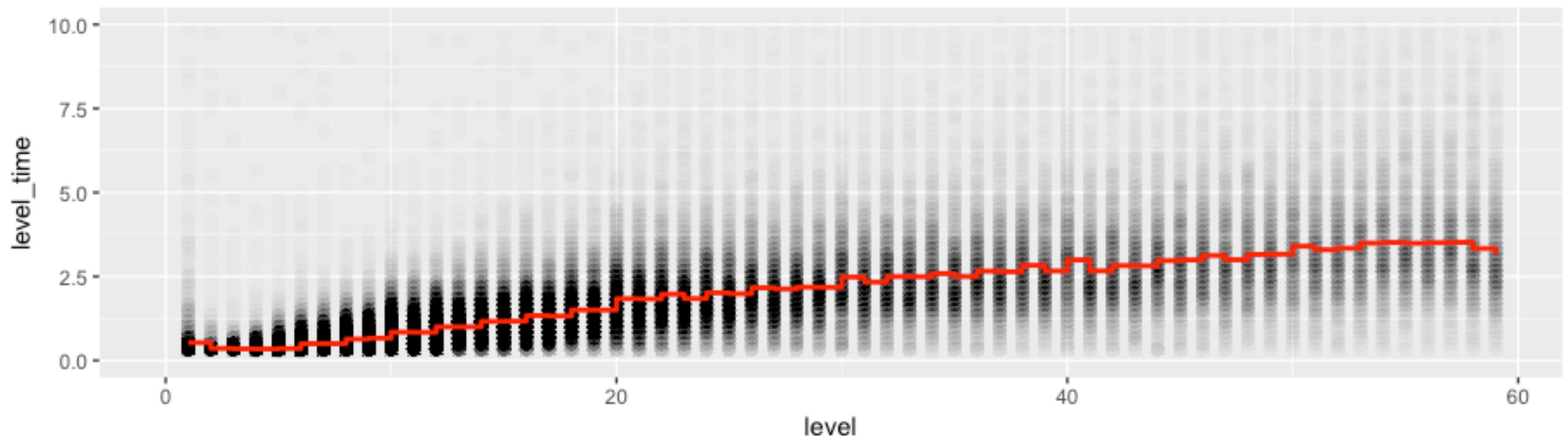
## Apply Statistic Part

```
agg_ddf <- ddf %>%
  filter(level <= 60) %>%
  arrange(char, timestamp) %>%
  group_by(char) %>%
  mutate(unix_ts = unix_timestamp(timestamp),
        unix_diff = unix_ts - lag(unix_ts)) %>%
  filter(unix_ts < 1220227200) %>%
  mutate(new_sess = ifelse(is.na(unix_diff), TRUE, unix_diff > 1000),
        session = new_sess %>% as.numeric() %>% cumsum()) %>%
  ungroup() %>%
  group_by(char, session, level, race, charclass, guild) %>%
  summarise(min_ts = min(unix_ts), max_ts = max(unix_ts)) %>%
  mutate(session_time = max_ts - min_ts) %>%
  ungroup() %>%
  group_by(char, level, race, charclass, guild = (guild != -1)) %>%
  summarise(level_time = sum(session_time)/60/60) %>%
  ungroup()
```

# Result!

Time it takes per level

whoa ... all that time wasted on humanity



total time  $\approx 132.6\text{hrs} > 3 \text{ weeks fte.}$

## Result!

This etl code in R is:

- readable from left to right, top to bottom
- describable via do this **then** that
- executable on many backends
- optimizable internally (AST!)

Being able to create a DSL in a language is a cool thing.

## Running Sparklyr Locally

You can read .csv or .parquet locally. Even start the 'cluster'.

```
sc <- spark_connect("local", version = "2.2.0")  
  
ddf <- spark_read_csv(sc, 'df', 'wowclean.csv',  
                      col_types = cols(  
                        char = col_integer(),  
                        level = col_integer(),  
                        charclass = col_character(),  
                        zone = col_character(),  
                        guild = col_integer(),  
                        timestamp = col_datetime()))
```

## **What About ML?**

The next few slides explain how you can model the time it takes to level in H20.

I'll skip it in the interest of time since it is mostly syntax anyway, feel free to remember that H20 is a great tool.

## **What About ML?**

R has your back as far as spark is concerned. There are two options; running it from Spark or with H2o on top of Spark. They both work directly on spark dataframes from sparklyr but I prefer to use the H2o varient.

1. grid search is exploratory
2. more hyperparams per model
3. pojo output

```
partitions <- agg_ddf %>%
  filter(level != 60) %>%
  sdf_partition(training = 0.5, test = 0.5, seed = 42)

train_frame <- as_h2o_frame(sc, partitions$training)
test_frame <- as_h2o_frame(sc, partitions$test)

hyper_parameters <- list(alpha = seq(0, 1, 0.1))

model_glm_grid <- h2o.grid(
  algorithm = "glm",
  grid_id = "glm_grid",
  hyper_params = hyper_parameters,
  training_frame = train_frame,
  validation_frame = test_frame,
  x = c('race', 'charclass', 'guild', 'level'),
  y = c('level_time')
)
```

# Observations

- The code we just describe really works and it is rather clear.
- That gridsearch feature is pretty sweet, much better than Spark's

It is a library though, not a grammar and still has limits. We'll get back to this point in a moment.

Before we do that, we should appreciate what we have by running this. The UI is pretty sweet.



Flow ▾

Cell ▾

Data ▾

Model ▾

Score ▾

Admin ▾

Help ▾

## Untitled Flow



## Models

<input type="checkbox"/>	Key
<input type="checkbox"/>	<a href="#">glm_grid_model_0</a>
<input type="checkbox"/>	<a href="#">glm_grid_model_1</a>
<input type="checkbox"/>	<a href="#">glm_grid_model_10</a>
<input type="checkbox"/>	<a href="#">glm_grid_model_2</a>
<input type="checkbox"/>	<a href="#">glm_grid_model_3</a>
<input type="checkbox"/>	<a href="#">glm_grid_model_4</a>
<input type="checkbox"/>	<a href="#">glm_grid_model_5</a>
<input type="checkbox"/>	<a href="#">glm_grid_model_6</a>
<input type="checkbox"/>	<a href="#">glm_grid_model_7</a>
<input type="checkbox"/>	<a href="#">glm_grid_model_8</a>

Algorithm
Generalized Linear Modeling

Actions
<a href="#"> Predict...</a> <a href="#"> Inspect</a>

OUTLINE FLOWS

## Help

Using Flow for

[Quickstart](#)Or, [view example Flows](#) H<sub>2</sub>O.

STAR H2O ON GITHUB!

[Star](#) 2,833

GENERAL

[Flow Web UI ...](#)[... Importing Data](#)

Vince [D] Warmerdam (@fishnets88) - Generalized Linear Modeling

## Untitled Flow



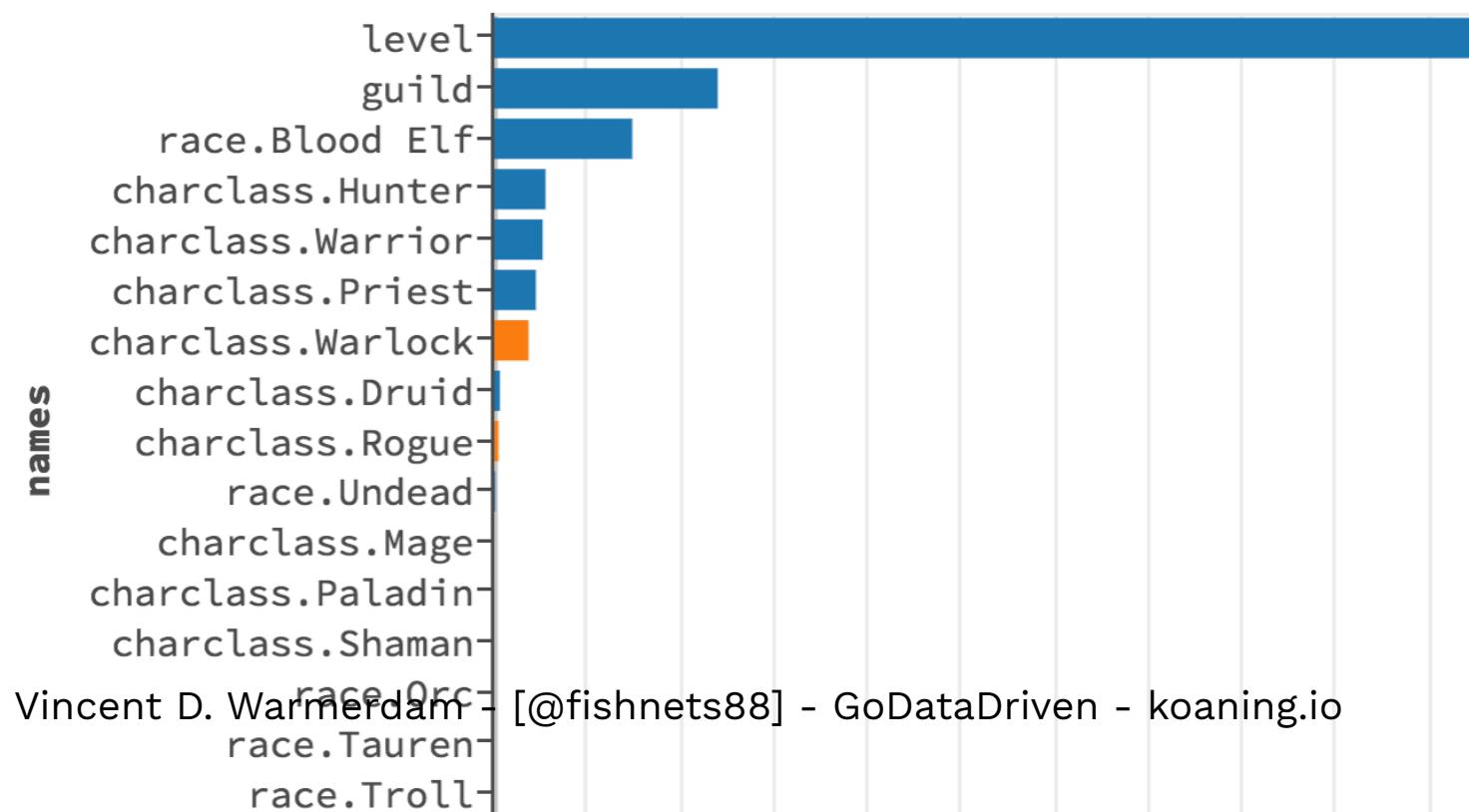
Actions: [Refresh](#) [Predict...](#) [Download POJO](#) [Download Model Deployment Package \(MOJO\)](#) [Export](#) [Inspect](#) [Delete](#)

[Download Gen Model](#)

► MODEL PARAMETERS

▼ SCORING HISTORY

▼ STANDARDIZED COEFFICIENT MAGNITUDES



## **Recap.**

We've seen that modern R packages prefer grammars and how this offer flexibility. It makes it easy to learn, is very expressive and being independant of backend is great.

I hope it's clear that currently, R actually has some nice features that are language-specific and also task specific (data/analytics).

If you don't like R, you can run non-R code from R.

## **Recap.**

But I've only been talking about somewhat current R stuff. Surely you all want to see something more special at this stage. I've only told you what is exiting about R today and have yet to show some ideas that are novel.

## Recap.

But I've only been talking about somewhat current R stuff. Surely you all want to see something more special at this stage. I've only told you what is exiting about R today and have yet to show some ideas that are novel.

To show why R still has features for the future, let's discuss that our common method of modelling ... is ... after some thinking ... **really** horrible.

The problem with the model is rather general and it explain better with a simpler dataset.

## General Modelling: the Problem

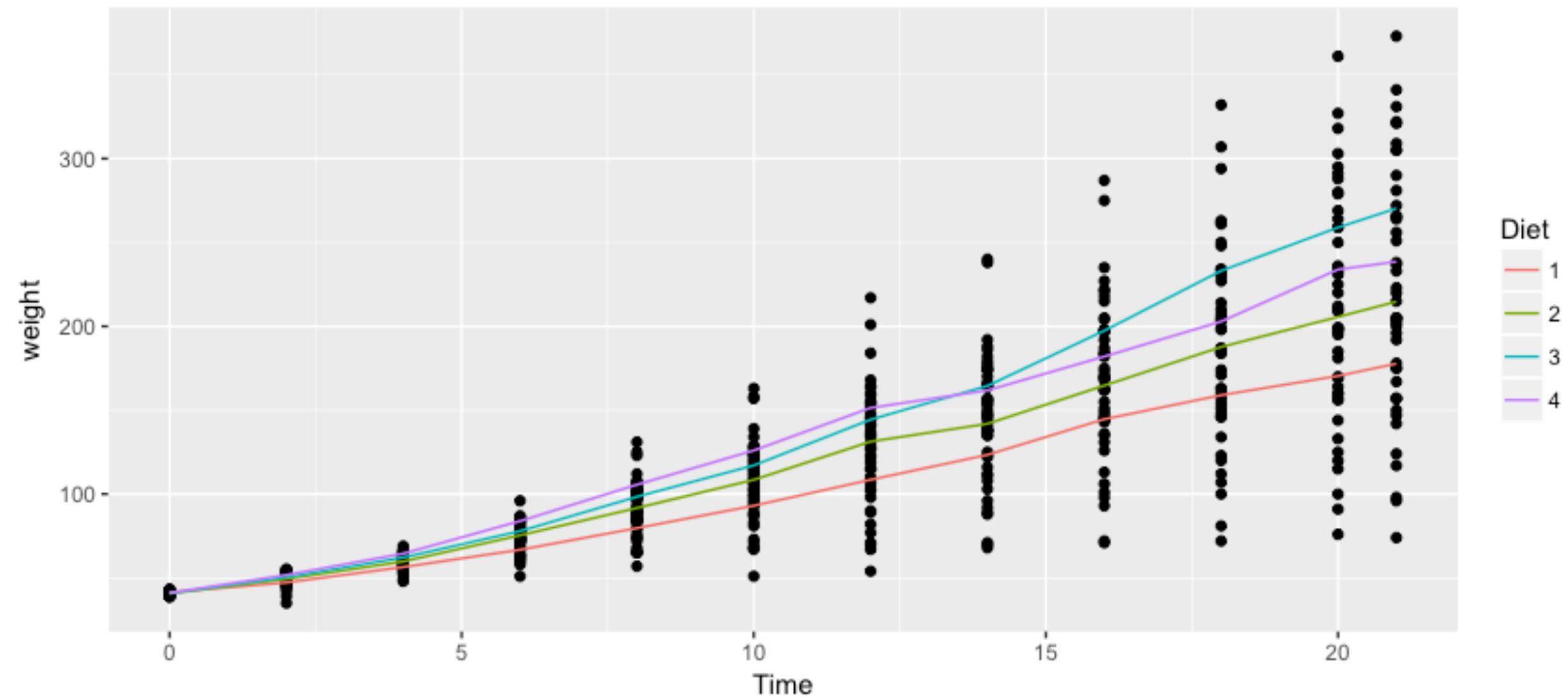
Suppose that I have a dataset with chickens.

	weight	Time	Chick	Diet
1	42	0	1	1
2	51	2	1	1
3	59	4	1	1
...				
576	234	18	50	4
577	264	20	50	4
578	264	21	50	4

# ChickWeight: the dataset

Weight of Chickens over Time

Certain Diets seem to make the Chickens more fat.



## Model 1: Base Regression

We could model it with a linear regression.

```
> model <- lm(weight ~ Time + Diet, data=chickweight)
> model %>% summary()

             Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.9244    3.3607   3.251  0.00122 **
Time         8.7505    0.2218  39.451 < 2e-16 ***
Diet2        16.1661   4.0858   3.957 8.56e-05 ***
Diet3        36.4994   4.0858   8.933 < 2e-16 ***
Diet4        30.2335   4.1075   7.361 6.39e-13 ***
```

No matter what backend you use, the model is **all wrong**.

## Model 2: R-trick: Nested Regression

```
> chickweight %>%
  group_by(Diet) %>%
  nest() %>%
  mutate(mod = data %>% map(~ lm(weight ~ Time, data=.)))
```

	Diet	data	mod
	<fctr>	<list>	<list>
1	1	<tibble [220 x 3]>	<S3: lm>
2	2	<tibble [120 x 3]>	<S3: lm>
3	3	<tibble [120 x 3]>	<S3: lm>
4	4	<tibble [118 x 3]>	<S3: lm>

Better, but this is still **wrong**.

## **The problem**

In Machine Learning it feels like we can pour data into a predefined model but it doesn't feel like we can define the model much.

We're usually constrained to perhaps feature engineering and hyperparam tuning (which granted, is good enough for lots of problems).

## **The problem**

In Machine Learning it feels like we can pour data into a predefined model but it doesn't feel like we can define the model much.

We're usually constrained to perhaps feature engineering and hyperparam tuning (which granted, is good enough for lots of problems).

Popular ML libraries don't offer a real DSL for models.

## Model 3: Domain Model

I wrote what I want on a piece of paper:

$$\omega \sim \beta_0 + \beta_1 t + \varepsilon$$

↑  
same for all chickens

This intercept  
is different per Diet

I want to basically try this, in a few lines of code.

'Your cat is broken.'  
— Rlang Model Purist

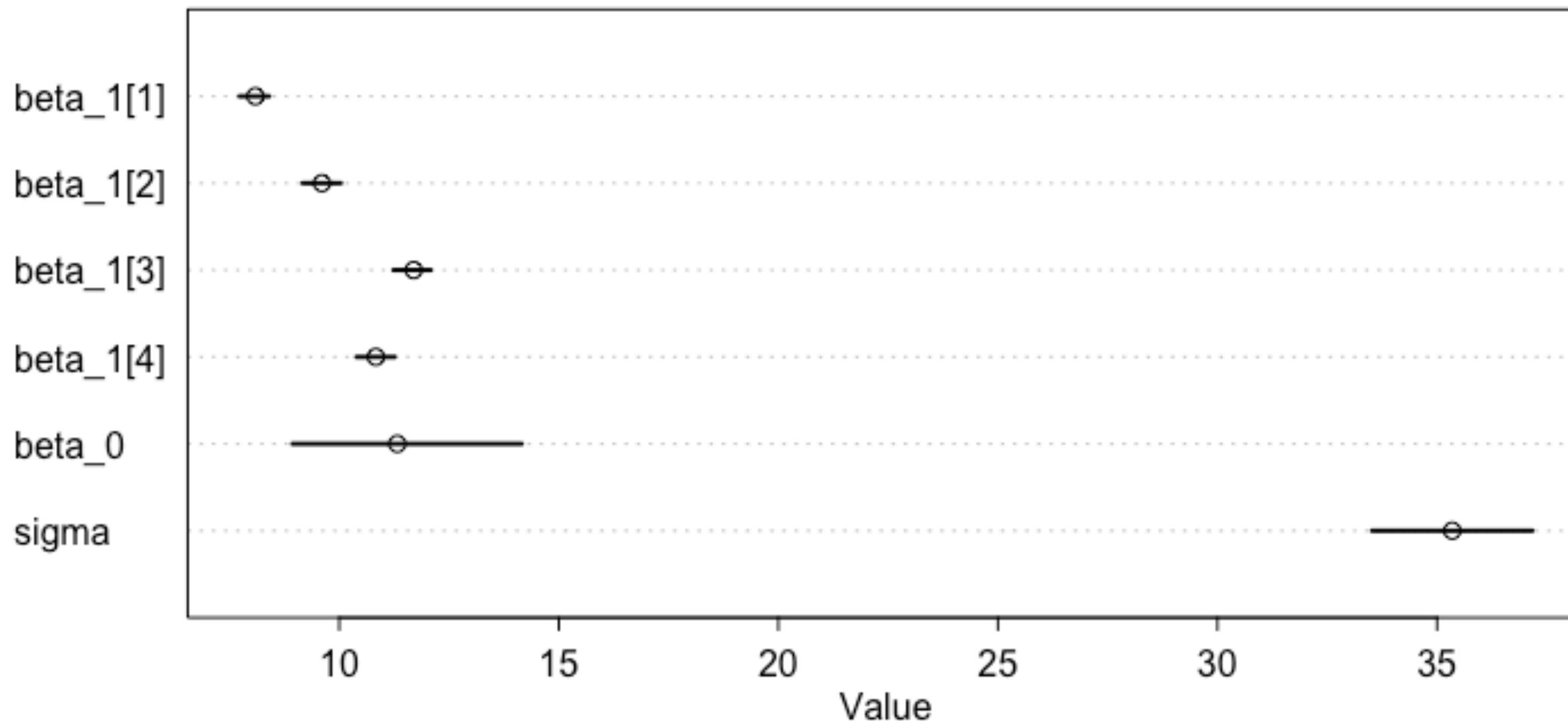


## **Model 3: Domain Model rethinking**

```
mod <- map2stan(  
  alist(  
    weight ~ dnorm(mu, sigma),  
    mu <- intercept + slope[Diet]*Time,  
    slope[Diet] ~ dnorm(0, 2),  
    intercept ~ dnorm(0, 2),  
    sigma ~ dunif(0, 10)  

```

## Model 3: Domain Model rethinking



## Model 3: Domain Model rethinking

$$\omega \sim \beta_0 + \beta_1 t + \varepsilon$$

↑  
Same for all chickens

$$\varepsilon = \alpha_0 + \alpha_1 t +$$

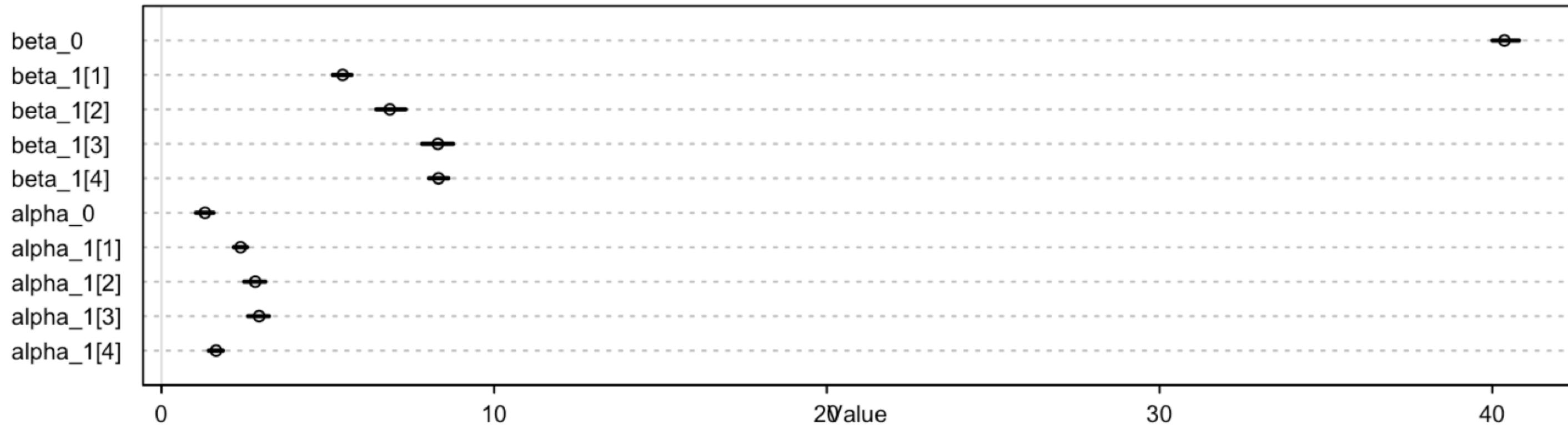
This intercept  
is different per Diet

## Model 3: Domain Model rethinking

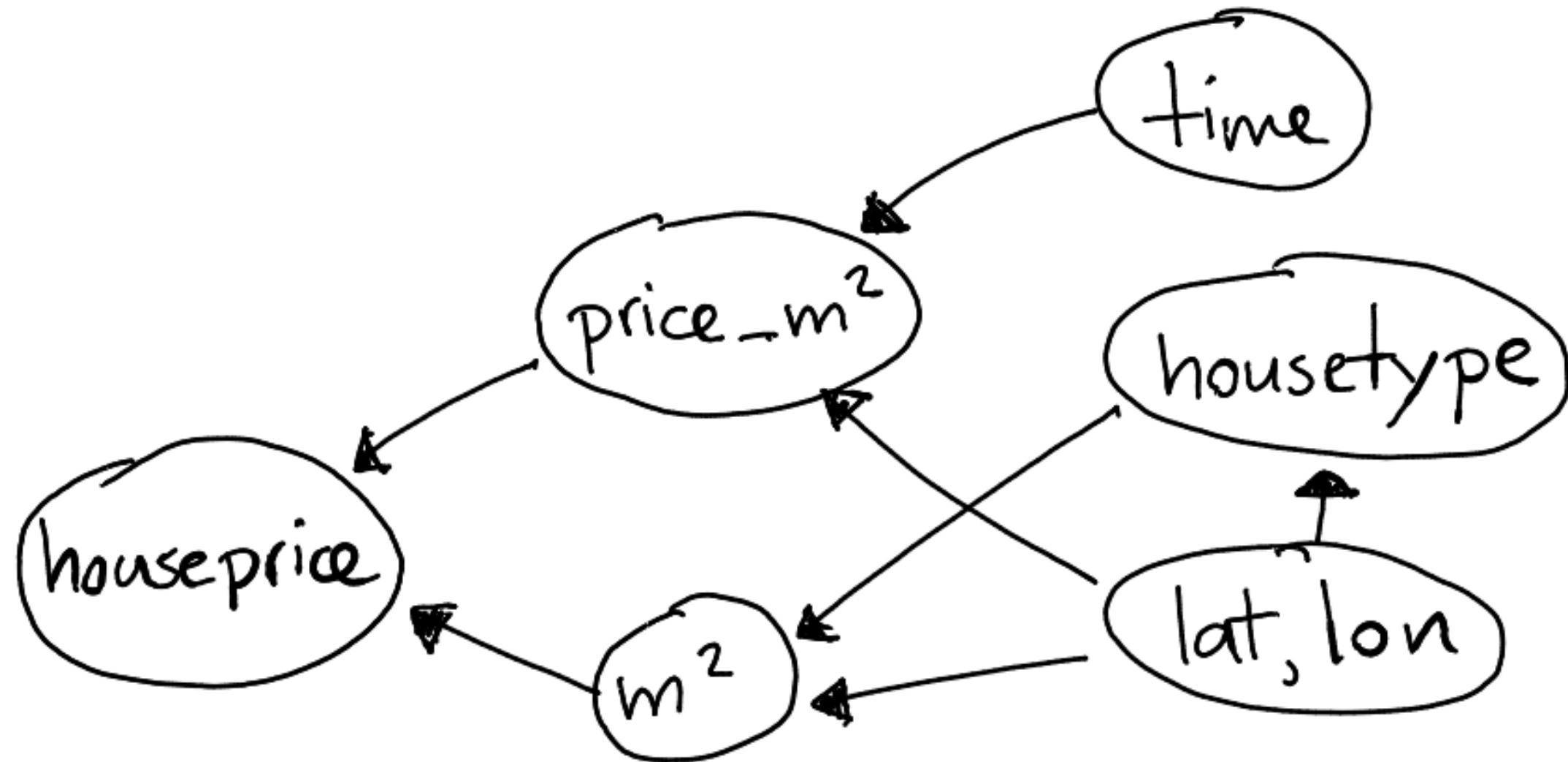
```
mod <- map2stan(  
  alist(  
    weight ~ dnorm(mu, sigma),  
    mu <- beta_0 + beta_1[Diet]*Time,  
    beta_0 ~ dnorm(0, 2),  
    beta_1[Diet] ~ dnorm(0, 2),  
    sigma <- alpha_0 + alpha_1[Diet]*Time,  
    alpha_0 ~ dunif(0, 10),  
    alpha_1[Diet] ~ dunif(0, 10)  

```

## Model 3: Domain Model rethinking



# Freedom



## **MetaModels!**

Most cool things we saw today are all a result of the nonstandard evaluation that occurs in R. It is exactly this what makes ggplot2, dplyr, dbplyr, sparklyr and bigrquery possible. Without it, the tools would be less expressive.

A logical next step in the ecosystem might be a convenient language of models. It can be designed to be independant of backend and it would be a huge feature for both industry and applied academia.

## Make a DSL for Models

This is merely a proposal of what it might look like.

```
ml_df %>%  
  modmaker({  
    p(Y_t | S_{t-1}) ~ N(mu[S{t-1}], sigma)  
    p(S_{t} | S_{t-1}) ~ Categorical(p[S-{t-1}])  
    mu ~ N(0, 1)  
    p ~ UniformDiscrete()  
  }) %>%  
  infer(backend='tensorflow', method='MCMC')
```

## Make a DSL for Models

This is merely a proposal of what it might look like.

```
ml_df %>%  
  modmaker({  
    p(Y_t | S_{t-1}) ~ N(mu[S{t-1}], sigma)  
    p(S_{t} | S_{t-1}) ~ Categorical(p[S-{t-1}])  
    mu ~ N(0, 1)  
    p ~ UniformDiscrete()  
  }) %>%  
  infer(backend='pytorch', method='adam')
```

## Make a DSL for Models

This is merely a proposal of what it might look like.

```
ml_df %>%  
  modmaker({  
    p(Y_t | S_{t-1}) ~ N(mu[S{t-1}], sigma)  
    p(S_{t} | S_{t-1}) ~ Categorical(p[S-{t-1}])  
    mu ~ N(0, 1)  
    p ~ UniformDiscrete()  
  }) %>%  
  infer(backend='H2O::Linear')
```

## **The Future still needs to be Made**

Designing a DSL is hard though. Can you come up with verbs such that you can simply declare the following models:

- latent dirichlet allocation
- trueskill
- hidden markov models
- domain recommenders

## **The Future still needs to be Made**

Designing a DSL is hard though. Can you come up with verbs such that you can simply declare the following models:

- latent dirichlet allocation
- trueskill
- hidden markov models
- domain recommenders
- your own creativity

## **The Future still needs to be Made**

Once the UI is dreamt up things get exiting. We're getting near a standard format for exchanging dataframes via apache arrow. We're getting near a standard for numerics too: tensors.

I hope it is clear that R won't need to care about backend too much.

The grammar for models may become a thing. I wouldn't be suprised if the 1st usable version of it is written in R.

## **The Future still needs to be Made**

It is good to note that there's plenty of bad things about R and. Silent errors might've been helping statisticians in the 90'ies but they're breaking docker containers today.

Stuff like logging and stuff like;

```
c(1,2,3,4) + 1  
c(1,2,3,4) + c(1,2,3)
```

Try it, you'll get dissapointed.

## **The Future still needs to be Made**

I'm a fan of R, it made me a better professional and a happier human.

Having said all this there's parts I don't like and **obviously** I love python too. I even really like to do golang and parts of javascript (d3) too.

It sometimes feels a bit strange to have to defend R to Python people though, it's all cats and dogs really.



# Announcements!



# Announcement



Let's get people working  
together to make the future  
more better.

## Questions?

