Data Frames and Data Pipelines

In Python and R

Agenda

1. Pandas and Dplyr
    - The Data Frame
    - Data Pipelines
2. Examples: Homework 3
3. EDA

What are the *data structures*?

Pandas Data Structures

There are three fundamental data structures in pandas:

- Data Frame: 2D data tabular data.
- Series: 1D data. I usually think of it as columnar data.
- Index: A sequence of row labels.

Data Frame

	Candidate	Party	%	Year	Result
0	Obama	Democratic	52.9	2008	win
1	McCain	Republican	45.7	2008	loss
2	Obama	Democratic	51.1	2012	win
3	Romney	Republican	47.2	2012	loss
4	Clinton	Democratic	48.2	2016	loss
5	Trump	Republican	46.1	2016	win

Series

```
0       Obama
1      McCain
2       Obama
3      Romney
4     Clinton
5       Trump
Name: Candidate, dtype: object
```

Index

Analogous Data Structures in R

- Data Frame: 2D tabular data.
- Atomic Vectors: Column of data of the same type.
- Row names: a sequence of row labels.

```
##   Candidate      Party Percentage
## 0     Obama Democratic       52.9
## 1    McCain Republican       45.7
## 2     Obama Democratic       51.1
## 3    Romney Republican       47.2
```

What is a data frame, generally?

- 2D data structure
- type heterogeneous
- columns = variables, rows = observations
- implicit row and column indices

What *isn't* a data frame?

Matrix

- 2D data structure
- type homogeneous
- implicit row and column indices

Relation (in SQL)

- 2D data structure
- type heterogeneous enforced via schema
- columns = variables, rows = observations
- no row or column indices

Why do we have data frames?

> “We have introduced into S a class of objects called data.frames, which can be used if convenient to organize all of the variables relevant to a particular analysis …”

J. Chambers, T. Hastie, and D. Pregibon, (1990), *Statistical Models in S*

> “Data frames are more general than matrices in the sense that matrices in S assume all elements to be of the same mode — all numeric, all logical,all character string, etc.” and “… data frames support matrix-like computation, with variables as columns and observations as rows,and, in addition, they allow computations in which the variables act as separate objects, referred to by name.”

J. M. Chambers, T. J. Hastie, et al. (1992), *Statistical Models in S*

Accessing data by name

Pandas data frame

Use `.loc[]`

```
pandas_df.loc[[0, 1],["Candidate", "Percentage"]]
```

```
##    Candidate  Percentage
## 0      Obama        52.9
## 1     McCain        45.7
```

R data frame

Use `[]`

```
r_df[c("0", "1"), c("Candidate", "Percentage")]
```

```
##   Candidate Percentage
## 0     Obama       52.9
## 1    McCain       45.7
```

Accessing data by position

Pandas data frame

Use `.iloc[]`

```
pandas_df.iloc[[0, 1],[0, 2]]
```

```
##   Candidate  Percentage
## 0     Obama        52.9
## 1    McCain        45.7
```

R data frame

Use `[]`

```
r_df[c(1, 2), c(1, 3)]
```

```
##   Candidate Percentage
## 0     Obama       52.9
## 1    McCain       45.7
```

Data Wrangling with `dplyr`

- A grammar for data wrangling with a small number of functions that can be composed in powerful ways.
- Inspired by SQL - declarative.
- Focus constructing *pipelines* to get from raw data to the data product you're aiming for.

Accessing data

```r
select(r_df, Candidate, Percentage)
```

```
##   Candidate Percentage
## 0     Obama       52.9
## 1    McCain       45.7
## 2     Obama       51.1
## 3    Romney       47.2
```

```r
slice(r_df, c(1, 2))
```

```
##   Candidate      Party Percentage
## 1     Obama Democratic       52.9
## 2    McCain Republican       45.7
```

```r
slice(select(r_df, Candidate, Percentage), c(1, 2))
```

```
##   Candidate Percentage
## 1     Obama       52.9
## 2    McCain       45.7
```

Building Pipelines for a Nursery Rhyme

Most data wrangling requires multiple *operations*, just as a nursery rhyme has multiple *verbs*:

> Little Bunny Foo Foo,
>
> Hopping through the forest,
>
> Scooping up the field mice,
>
> And bopping them on the head.

Building Pipelines, take 1

One approach is to break it down step by step and take the output and overwrite the input.

```
foo_foo <- hop(foo_foo, through = forest)
foo_foo <- scoop(foo_foo, up = field_mice)
foo_foo <- bop(foo_foo, on = head)
```

(example from *R for Data Science* (Wickham and Grolemund))

Building Pipelines, take 2

Another approach is to nest the functions inside one another.

```
bop(
  scoop(
    hop(foo_foo, through = forest),
    up = field_mice
  ),
  on = head
)
```

Building Pipelines, take 3

Another more readable approach is to use the pipe operator (%>%) to pass the output of one function as the input to the next.

```
foo_foo %>%
  hop(through = forest) %>%
  scoop(up = field_mice) %>%
  bop(on = head)
```

Relies upon the system being closed under these operations: *data frame in, data frame out*.

```
r_df %>%
  select(Candidate, Percentage) %>%
  slice(1, 2)
```

```
##   Candidate Percentage
## 1     Obama       52.9
## 2    McCain       45.7
```

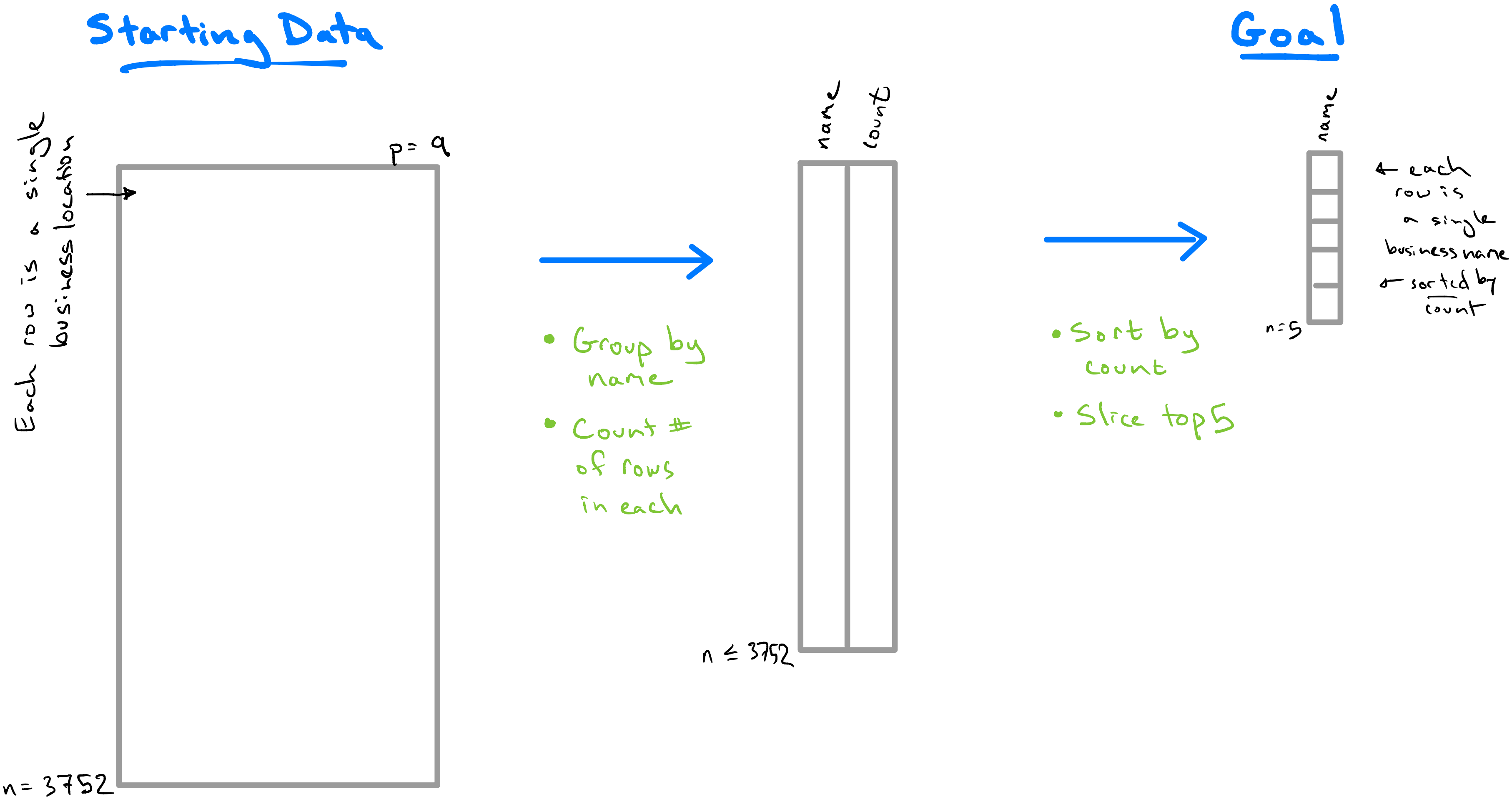
Example: Food Safety

```
bus
```

```
## # A tibble: 3,752 x 9
##      bid name   address  city  state postal_code latitude longitude phone_n
##    <dbl> <chr>  <chr>    <chr> <chr> <chr>          <dbl>     <dbl>
##  1  3183 NEW E… 907 Irv… San … CA    94122           37.8     -122.
##  2 91931 Gourm… 4605 Ge… San … CA    94118        -9999     -9999   141557
##  3 91826 The W… OFF THE… San … CA    -9999        -9999     -9999   141583
##  4 94935 94635… 24 Will… San … CA    94107        -9999     -9999
##  5 70425 Peet'… 1509 SL… San … CA    94132        -9999     -9999   141505
##  6  2249 Ramzi… 0044 Mo… San … CA    94104           37.8     -122.
##  7 99845 EAT C… 1450 AR… San … CA    94124        -9999     -9999   141506
##  8 93959 Willi… 2055 Si… San … CA    94124        -9999     -9999
##  9 77404 Shabu… 219 Kin… San … CA    94107        -9999     -9999
## 10 89282 Taque… Mission… San … CA    -9999        -9999     -9999   141508
## # … with 3,742 more rows
```

Question 1c: Assign top_names to the top 5 most frequently used business names, from most frequent to least frequent.

Question 1c: Assign top_names to the top 5 most frequently used business names, from most frequent to least frequent.



```
bus %>%
  group_by(name) %>%
  summarize(cnt = n()) %>%
  arrange(desc(cnt)) %>%
  slice(1:5)
```

```
## summarise() ungrouping output (override with .groups argument)

## # A tibble: 5 x 2
##   name                          cnt
##   <chr>                       <int>
## 1 Peet's Coffee & Tea            14
## 2 Starbucks Coffee                9
## 3 STARBUCKS                       7
## 4 Proper Food                     6
## 5 Specialty's Cafe & Bakery       6
```

```
bus %>%
  count(name) %>%
  arrange(desc(n)) %>%
  slice(1:5) %>%
  select(name)
```

```
## # A tibble: 5 x 1
##   name
##   <chr>
## 1 Peet's Coffee & Tea
## 2 Starbucks Coffee
```

Pandas and dplyr

```
bus["name"].value_counts()[:5].index.values
```

```
## array(["Peet's Coffee & Tea", 'Starbucks Coffee', 'STARBUCKS',
##        'Proper Food', 'Starbucks'], dtype=object)
```

Notes on pandas:

- Data structures change: data frame > series > index > array.
- Combines operators (`[]`) and methods.

```
bus %>%
  count(name) %>%
  arrange(desc(n)) %>%
  slice(1:5) %>%
  pull(name)
```

Notes on dplyr

- Data structure doesn't change: the dataframe/tibble.
- Uses only functions.

A Pipeline in Pandas

```
bus["name"].value_counts()[:5].index.values
```

```
## array(["Peet's Coffee & Tea", 'Starbucks Coffee', 'STARBUCKS',
##        'Proper Food', 'Starbucks'], dtype=object)
```

vs

```
(bus["name"]
  .value_counts()[:5]
  .index
  .values)
```

```
## array(["Peet's Coffee & Tea", 'Starbucks Coffee', 'STARBUCKS',
##        'Proper Food', 'Starbucks'], dtype=object)
```

The pipeline form is ensures each operation is easily readible and distinct.

Question 6a

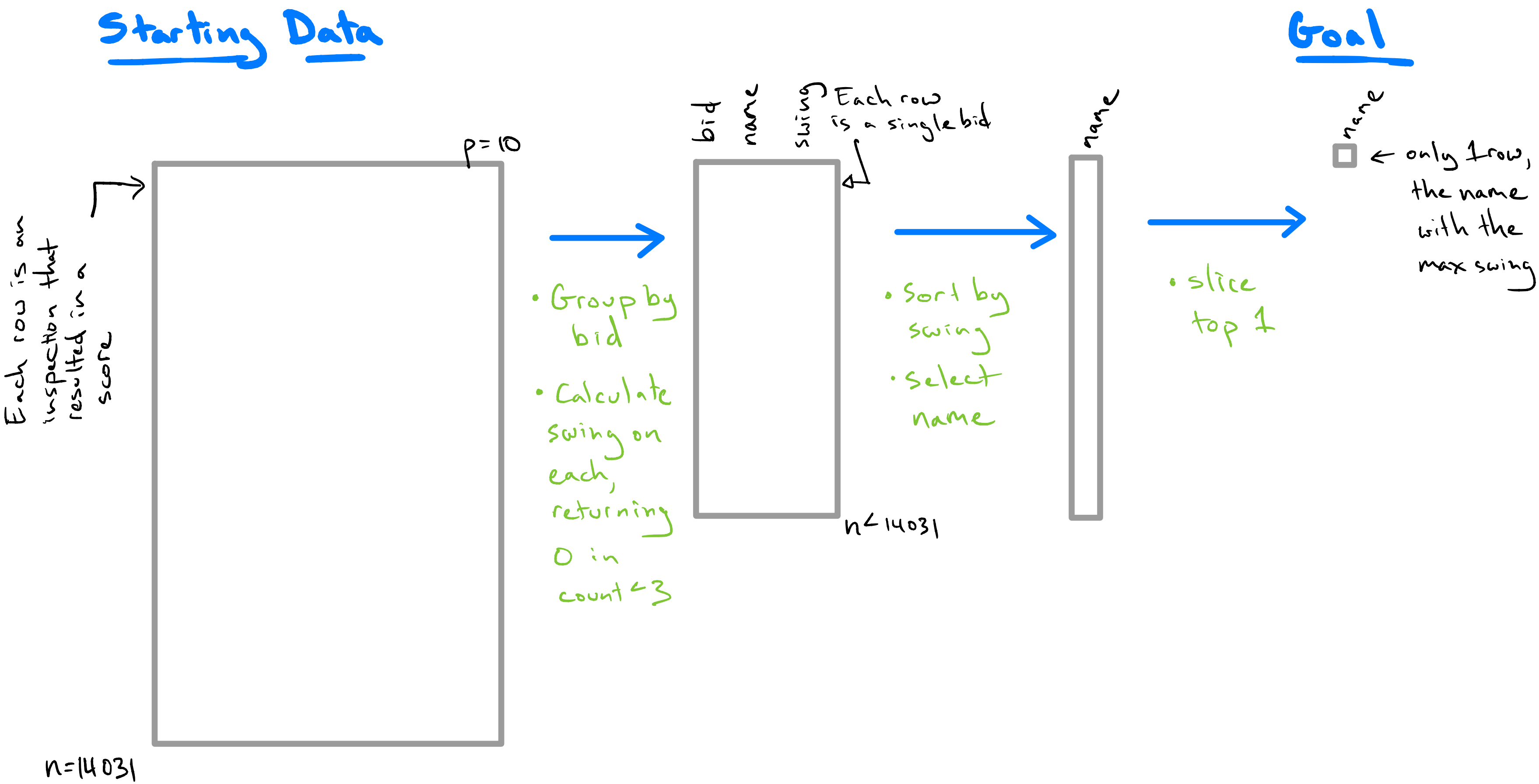
> Let's see which restaurant has had the most extreme improvement in its rating, aka scores. Let the "swing" of a restaurant be defined as the difference between its highest-ever and lowest-ever rating. Only consider restaurants with at least 3 ratings, aka rated for at least 3 times (3 scores)! Using whatever technique you want to use, assign max_swing to the name of restaurant that has the maximum swing.
>
> Note: The "swing" is of a specific business. There might be some restaurants with multiple locations; each location has its own "swing".
>
> The city would like to know if the state of food safety has been getting better, worse, or about average. This is a pretty vague and broad question, which you should expect as part of your future job as a data scientist! However for the ease of grading for this assignment, we are going to guide you through it and offer some specific directions to consider.

The start of the pipeline

```
ins_named
```

```
## # A tibble: 14,031 x 10
##    iid   date  score type     bid timestamp   year Missing Score name  addr
##    <chr> <chr> <dbl> <chr>  <dbl> <date>     <dbl> <lgl>          <chr> <c
##  1 1000… 04/0…   100 Rout… 100010 2019-04-03  2019 FALSE          ILLY… PI
##  2 1000… 08/1…    91 Rout… 100017 2019-08-16  2019 FALSE          AMIC… 47
##  3 1000… 05/2…    83 Rout… 100041 2019-05-20  2019 FALSE          UNCL… 36
##  4 1000… 04/2…    98 Rout… 100055 2019-04-25  2019 FALSE          Twir… 33
##  5 1000… 09/1…    82 Rout… 100055 2019-09-12  2019 FALSE          Twir… 33
##  6 1000… 08/1…    89 Rout… 100058 2019-08-16  2019 FALSE          SF P… 47
##  7 1000… 08/1…    76 Rout… 100059 2019-08-15  2019 FALSE          DUMP… 25
##  8 1000… 09/0…   100 Rout… 100069 2019-09-06  2019 FALSE          Miss… 14
##  9 1000… 03/2…    89 Rout… 100072 2019-03-26  2019 FALSE          SUBW… 23
## 10 1000… 08/2…    98 Rout… 100079 2019-08-27  2019 FALSE          POSI… 47
## # … with 14,021 more rows
```

Constructing a pipeline (take 2)



A pipeline in R, take 1

```
ins_named %>%
  group_by(bid) %>%
  mutate(n = n()) %>%
  filter(n >= 3) %>%
  mutate(swing = max(score) - min(score)) %>%
  ungroup() %>%
  arrange(desc(swing)) %>%
  select(name) %>%
  slice(1)
```

```
## # A tibble: 1 x 1
##   name
##   <chr>
## 1 Lollipot
```

A pipeline in R, take 2

```r
swing <- function(x) {
  if (length(x) < 3) {
    return(0)
  } else {
    return(max(x) - min(x))
    }
}

ins_named %>%
  group_by(bid) %>%
  mutate(swing = swing(score)) %>%
  ungroup() %>%
  arrange(desc(swing)) %>%
  select(name) %>%
  slice(1)
```

```
## # A tibble: 1 x 1
##   name
##   <chr>
## 1 Lollipot
```

Question 6b

> What's the relationship between the first and second scores for the businesses with 2 inspections in a year? Do they typically improve? For simplicity, let's focus on only 2018 for this problem.
>
> Plot these scores. That is, make a scatter plot to display these pairs of scores. Include on the plot a reference line with slope 1.

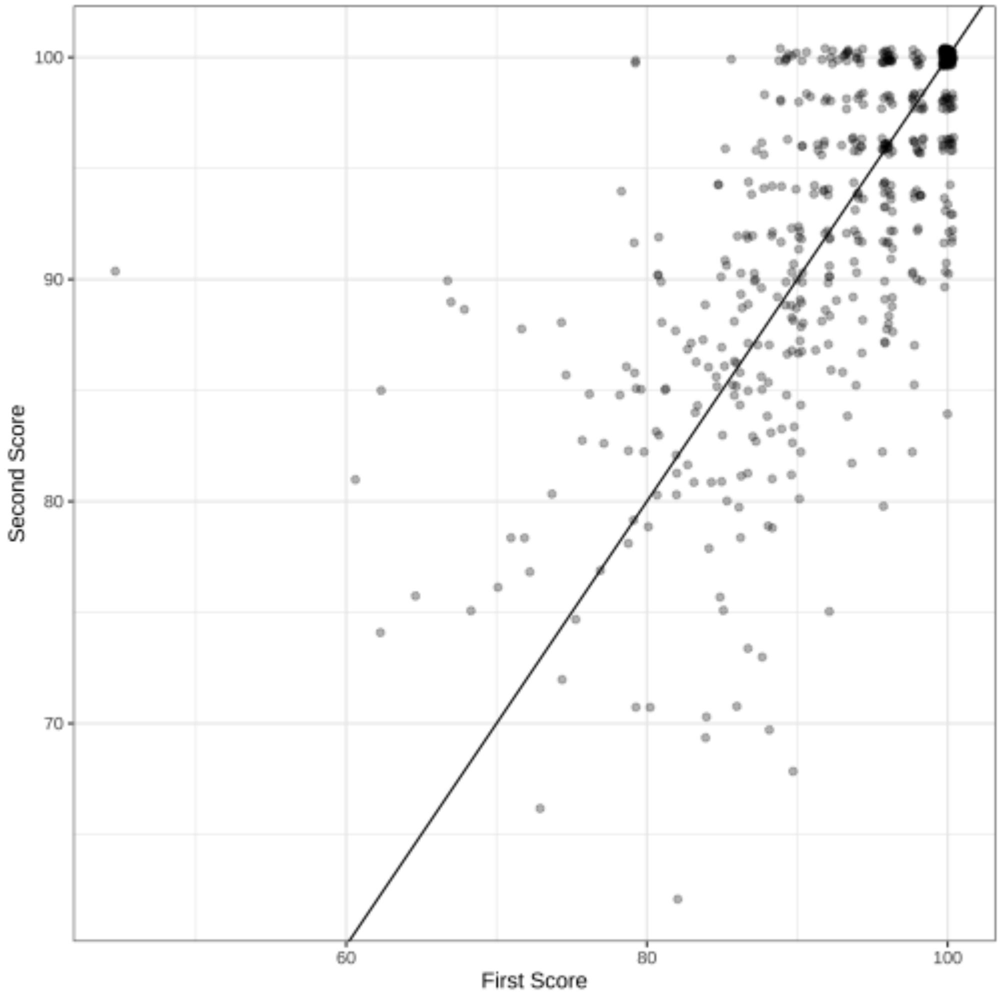
The start of the pipeline

```
ins
```

```
## # A tibble: 14,031 x 8
##    iid        date         score type              bid timestamp   year Missing Sc
##    <chr>      <chr>        <dbl> <chr>           <dbl> <date>     <dbl> <lgl>
##  1 100010_2… 04/03/201…   100 Routine -… 100010 2019-04-03  2019 FALSE
##  2 100017_2… 08/16/201…    91 Routine -… 100017 2019-08-16  2019 FALSE
##  3 100041_2… 05/20/201…    83 Routine -… 100041 2019-05-20  2019 FALSE
##  4 100055_2… 04/25/201…    98 Routine -… 100055 2019-04-25  2019 FALSE
##  5 100055_2… 09/12/201…    82 Routine -… 100055 2019-09-12  2019 FALSE
##  6 100058_2… 08/16/201…    89 Routine -… 100058 2019-08-16  2019 FALSE
##  7 100059_2… 08/15/201…    76 Routine -… 100059 2019-08-15  2019 FALSE
##  8 100069_2… 09/06/201…   100 Routine -… 100069 2019-09-06  2019 FALSE
##  9 100072_2… 03/26/201…    89 Routine -… 100072 2019-03-26  2019 FALSE
## 10 100079_2… 08/27/201…    98 Routine -… 100079 2019-08-27  2019 FALSE
## # … with 14,021 more rows
```

Constructing a pipeline into a plot

```
ins %>%
  filter(year == 2018) %>%
  group_by(bid) %>%
  mutate(n = n()) %>%
  filter(n == 2) %>%
  arrange(bid, timestamp) %>%
  ungroup() %>%
  mutate(order = rep(c("first_inspection", "second_inspection"), 535)) %
  select(bid, score, order) %>%
  pivot_wider(names_from = order,
              values_from = score) %>%
  ggplot(aes(x = first_inspection,
             y = second_inspection)) +
  geom_jitter() +
  theme_bw()
```


100
90
80
70
Second Score
60
80
100
First Score

Bonus: Spark Data Frames

```
textFile = sc.textFile("hdfs://...")

# Creates a DataFrame having a single column named "line"
df = textFile.map(lambda r: Row(r)).toDF(["line"])
errors = df.filter(col("line").like("%ERROR%"))
# Counts all the errors
errors.count()
# Counts errors mentioning MySQL
errors.filter(col("line").like("%MySQL%")).count()
# Fetches the MySQL errors as an array of strings
errors.filter(col("line").like("%MySQL%")).collect()
```