Data Wrangling and Visualisation Exercises

RUG @ HSG

Below, you will find some exercises to apply and improve your data wrangling and visualisation skills. Generally, they increase in difficulty as you go along. Don't hesitate to ask questions in the Q&A WhatsApp group, we or your fellow students will be happy to answer them!

The packages we loaded for this exercise:

```
library(tidyverse)
library(tidymodels)
library(tidytext)
```

The data set

In these exercises we will be working with data on **London AirBnBs** (Click here to download and save to csv). Each row represents one listing, and there are a variety of columns with information on the listing, such as the name, host, price and more. This dataset could be used to study patterns in Airbnb pricing, to understand how Airbnbs are being used in London, or to compare different neighbourhoods in London. (Source)

Download the data, store it in the same folder as your RScript or Notebook and read in the `listings.csv` file only.

```
listings <- read_csv("listings.csv")
```

```
## Rows: 83850 Columns: 17
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr   (4): name, host_name, neighbourhood, room_type
## dbl  (11): index, id, host_id, latitude, longitude, price, minimum_nights, n...
## lgl   (1): neighbourhood_group
## date  (1): last_review
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Task 1: Getting an overview over the data

(a) In the `tidyverse` there are multiple functions that allow you to gain a quick overview over the data that you are working with. (*Tip:* Try out the function `glimpse()` to get all of the information in one step.)

(b) Check for missing and duplicated values.

(c) Count all nominal values and print summary information on all numerical values. (*Hint:* The function `count()` gives you the value counts for the specified variables. Can you additionally to sort the results?)

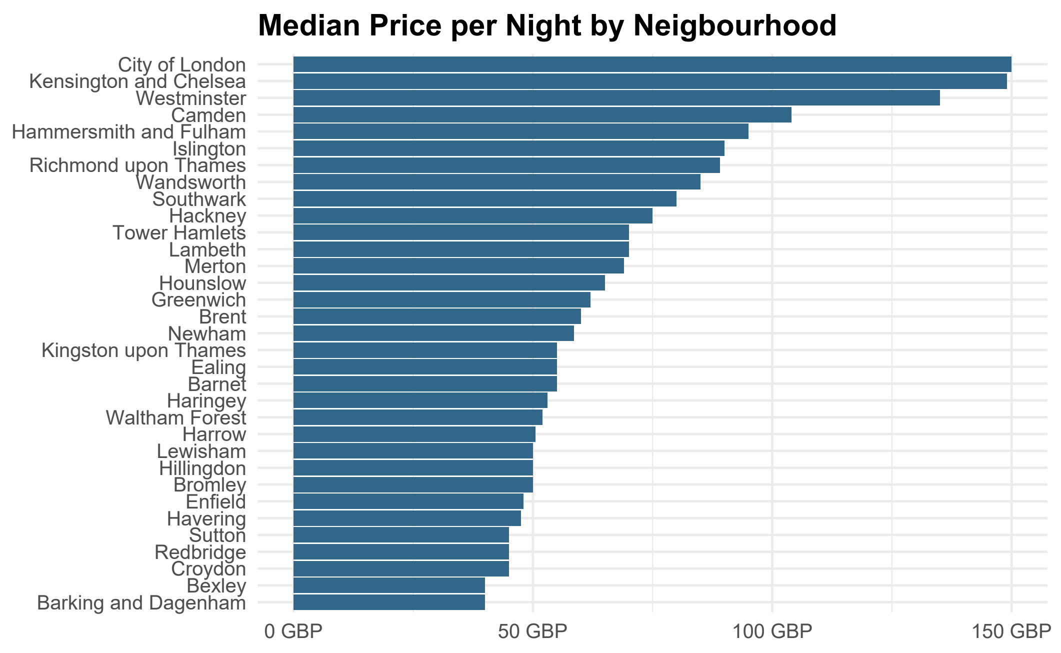
Task 2: Handling/Wrangling/Munging/Massaging and visualising the data

Now that you have an idea about the data, this part will get interesting. The bread and butter of data analysis is visualisation, it is always wise to take a look at some charts before you do any modelling or inference. In the spirit of the data visualisation godfather Edward Tufte:

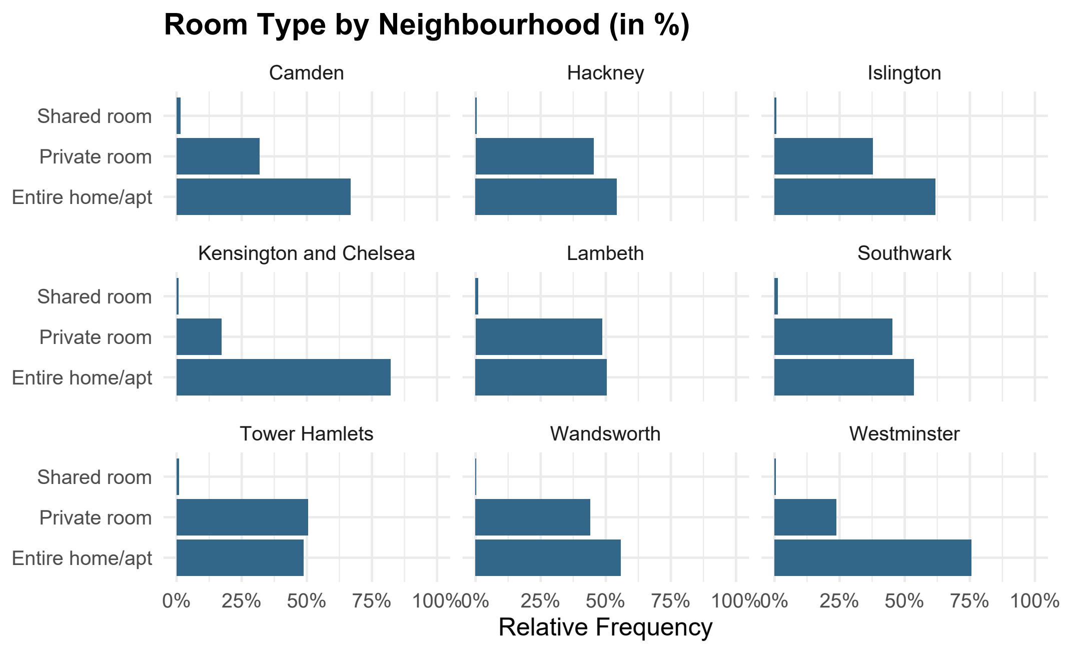
> "Above all else show the data."

If you feel like you don't know how to do it all yet, just have a look at the solution and try to understand every step (see what happens if you run the code line by line).

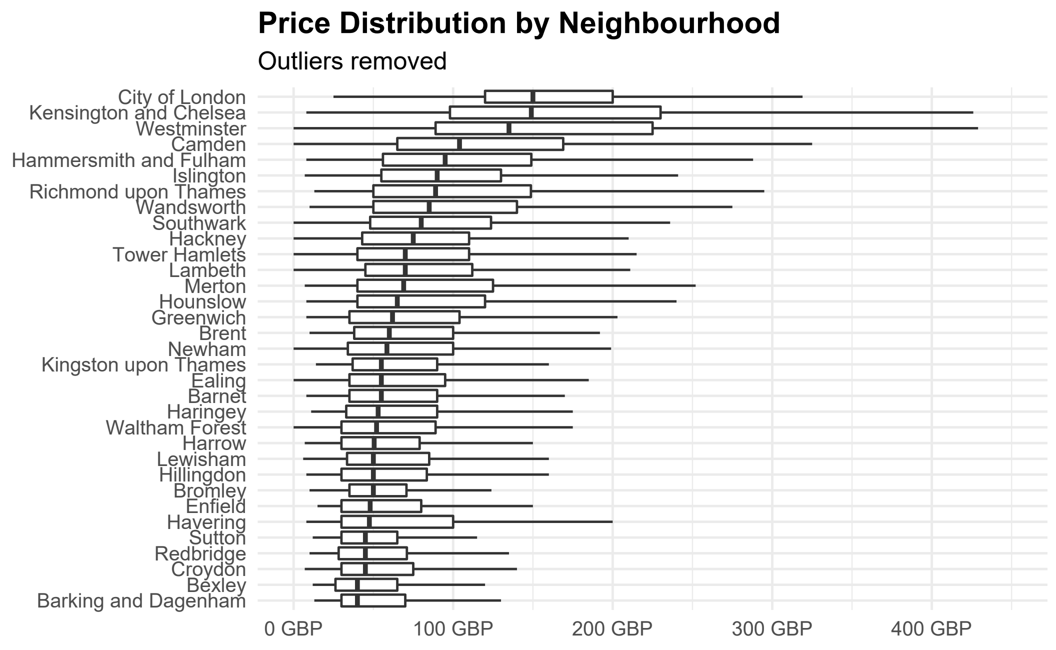
(a) Calculate the median price for each neighbourhood and plot it in a sorted bar plot like below.



(b) Count the room types for the nine most frequent neighbourhoods, normalise their frequency to a percentage and plot them into a bar chart. (*Hint:* The function `reorder_within()` from the `tidytext` package can be used for ordering the bars in the different facets. Make sure to also add `scales='free'` in the `facet_wrap` line and to add a `scale_y_reordered`, should you decide to order it.).



(c) Plot the boxplots by neighbourhood of the price per night. There is no preparation of the data required for this step, you can pipe the dataset straight into the `ggplot` call. In any price data, the distribution often has a long right tail, therefore using a log scale can make the chart more legible. In this example, use `coord_cartesian()` to zoom into the chart instead. As opposed to the `scale_`, `coord_cartesian()` does not drop the observations and shift the axis and does exactly what we want: merely zoom in.



Task 3: A basic model

So far we have only analysed some aspects of the data. Let's get to the next step of making a basic model for predictions. We will show the `tidymodels` package, which makes data preparation really easy.

The first step is always splitting the data into a training and testing set. The training data set will be where we let the model learn patterns in the data. The testing data set will be where we want to see if the model can make predictions on previously unseen data. Testing the model on the dataset it has learned from doesn't make sense, as the goal is to take the model and use it on new data.

```
set.seed(1) # setting random seed to make results reproducible
split <- initial_split(listings, frac = 0.75)

train_data <- training(split)
test_data <- testing(split)
```

What the above code did was randomly sampling 75% from the total data and storing it in `train_data`. Then, filtering out the observations from the dataset that are now in the training set and taking the rest and storing it in `test_data`.

```
split
```

```
## <Training/Testing/Total>
## <62887/20963/83850>
```

We can see in the split object how many rows were used for training and testing.

Now we can select a few variables, that we will use in a linear regression, let's say these three to predict the price:

```
listings %>%
  select(neighbourhood, room_type, number_of_reviews, price ) %>%
  glimpse()
```

```
## Rows: 83,850
## Columns: 4
## $ neighbourhood     <chr> "Islington", "Kensington and Chelsea", "Westminster"~
## $ room_type         <chr> "Private room", "Entire home/apt", "Entire home/apt"~
## $ number_of_reviews <dbl> 16, 85, 41, 93, 28, 122, 62, 79, 115, 5, 187, 70, 50~
## $ price             <dbl> 65, 100, 300, 175, 65, 29, 147, 147, 34, 100, 45, 19~
```

We will use a linear regression. As linear regressions require numeric predictors, we cannot use `neighbourhood` and `room_type` in their current form. We can use dummy encoding, i.e. represent them in binary form with zeros and ones. `Tidymodels` makes this preprocessing step very easy:

```
preprocessing_recipe <- recipe(price ~ neighbourhood + room_type +
                                 number_of_reviews,
                               data = train_data) %>%
  step_dummy(all_nominal_predictors())
```

We can apply the preprocessing steps to the testing data like such and see what comes out the other side:

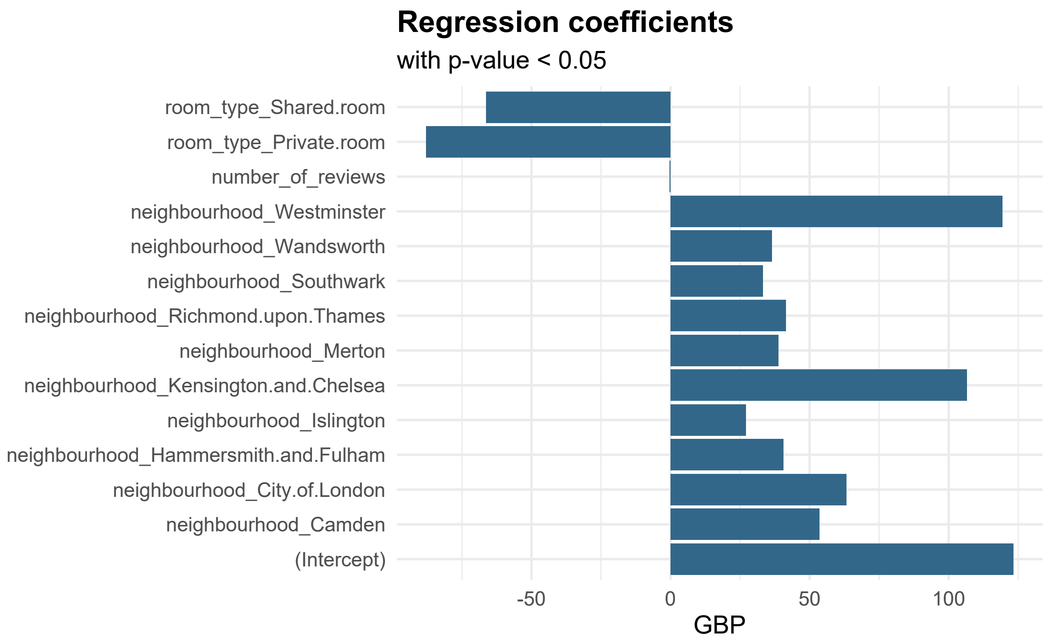
```
preprocessing_recipe %>%
  prep() %>%
  juice()
```

```
## # A tibble: 62,887 x 36
##    numbe~1 price neigh~2 neigh~3 neigh~4 neigh~5 neigh~6 neigh~7 neigh~8 neigh~9
##      <dbl> <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1      44    55       0       0       0       0       0       0       0       0
##  2      15    50       0       0       0       0       0       0       0       0
##  3       3    53       0       0       0       0       0       0       0       0
##  4       2   110       0       0       0       0       0       0       0       0
##  5       0   100       0       0       0       0       0       0       0       0
##  6       3   560       0       0       0       0       0       0       0       0
##  7       8   115       0       0       0       0       0       0       0       0
##  8       1    29       0       0       0       0       0       0       0       0
##  9      29    80       0       0       0       0       0       0       0       0
## 10       7   189       0       0       0       0       0       0       0       0
## # ... with 62,877 more rows, 26 more variables: neighbourhood_Enfield <dbl>,
## #   neighbourhood_Greenwich <dbl>, neighbourhood_Hackney <dbl>,
## #   neighbourhood_Hammersmith.and.Fulham <dbl>, neighbourhood_Haringey <dbl>,
## #   neighbourhood_Harrow <dbl>, neighbourhood_Havering <dbl>,
## #   neighbourhood_Hillingdon <dbl>, neighbourhood_Hounslow <dbl>,
## #   neighbourhood_Islington <dbl>, neighbourhood_Kensington.and.Chelsea <dbl>,
## #   neighbourhood_Kingston.upon.Thames <dbl>, neighbourhood_Lambeth <dbl>, ...
```

Perfect, now we can fit the model:

```
trained_model <- workflow() %>%
  add_model(linear_reg()) %>%
  add_recipe(preprocessing_recipe) %>%
  fit(train_data)
```

Let's look at the coefficients:



Careful: We can't interpret these intercepts causally without further assumptions, however we can see that with the given variables, the model picked up on some signal from more expensive neighbourhoods (like Westminster and Kensington and Chelsea). Additionally, the dummy variables have to be interpreted relative to the level which has been left out. For example: The share room and private room are less expensive compared to a full house, which is the level left out by the dummy encoding to avoid multicollinearity.

Now, we can make predictions on the testing data to see if the model is any good:

```
trained_model %>%
  # This means: stick predictions for each line to the right
  augment(test_data) %>%
  rsq(truth = price, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rsq     standard      0.0785
```

```
trained_model %>%
  # This means: stick predictions for each line to the right
  augment(test_data) %>%
  mae(truth = price, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 mae     standard        65.8
```

As you can see, the R^2 of the predictions is under 10%, so not very impressive. The mean average error is about 65 pounds - also not impressive.

(a) Your task: Can you create a model (include more variables, other transformations, other model, ...) that performs better than that? Let us know on our socials (Instagram)!

Solutions

Task 1

(a)

```
## Rows: 83,850
## Columns: 17
## $ index                          <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1~
## $ id                             <dbl> 13913, 15400, 17402, 24328, 25023, 2512~
## $ name                           <chr> "Holiday London DB Room Let-on going", ~
## $ host_id                        <dbl> 54730, 60302, 67564, 41759, 102813, 103~
## $ host_name                      <chr> "Alina", "Philippa", "Liz", "Joe", "Ama~
## $ neighbourhood_group            <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ neighbourhood                  <chr> "Islington", "Kensington and Chelsea", ~
## $ latitude                       <dbl> 51.56802, 51.48796, 51.52098, 51.47298,~
## $ longitude                      <dbl> -0.11121, -0.16898, -0.14002, -0.16376,~
## $ room_type                      <chr> "Private room", "Entire home/apt", "Ent~
## $ price                          <dbl> 65, 100, 300, 175, 65, 29, 147, 147, 34~
## $ minimum_nights                 <dbl> 1, 3, 3, 30, 4, 10, 3, 2, 1, 1, 1, 3, 2~
## $ number_of_reviews              <dbl> 16, 85, 41, 93, 28, 122, 62, 79, 115, 5~
## $ last_review                    <date> 2019-06-10, 2019-05-05, 2019-06-19, 20~
## $ reviews_per_month              <dbl> 0.15, 0.73, 0.41, 0.88, 0.69, 1.08, 0.5~
## $ calculated_host_listings_count <dbl> 3, 1, 14, 1, 1, 3, 5, 5, 1, 1, 1, 1, 2,~
## $ availability_365               <dbl> 347, 203, 269, 329, 2, 222, 355, 342, 3~
```

(b)

This will give you a sorted fraction of missing values by column, i.e. "What percentage of missing values in each column?":

```
colMeans(is.na(listings)) %>%
  enframe() %>%
  arrange(-value)
```

```
## # A tibble: 17 x 2
##    name                              value
##    <chr>                             <dbl>
##  1 neighbourhood_group            1
##  2 last_review                    0.251
##  3 reviews_per_month              0.251
##  4 host_name                      0.000561
##  5 name                           0.000310
##  6 index                          0
##  7 id                             0
##  8 host_id                        0
##  9 neighbourhood                  0
## 10 latitude                       0
## 11 longitude                      0
## 12 room_type                      0
## 13 price                          0
## 14 minimum_nights                 0
## 15 number_of_reviews              0
## 16 calculated_host_listings_count 0
## 17 availability_365               0
```

For duplicate rows check if the dimensions of the original dataset and when selecting distinct rows are the same:

```
listings %>%
  dim()
```

```
## [1] 83850    17
```

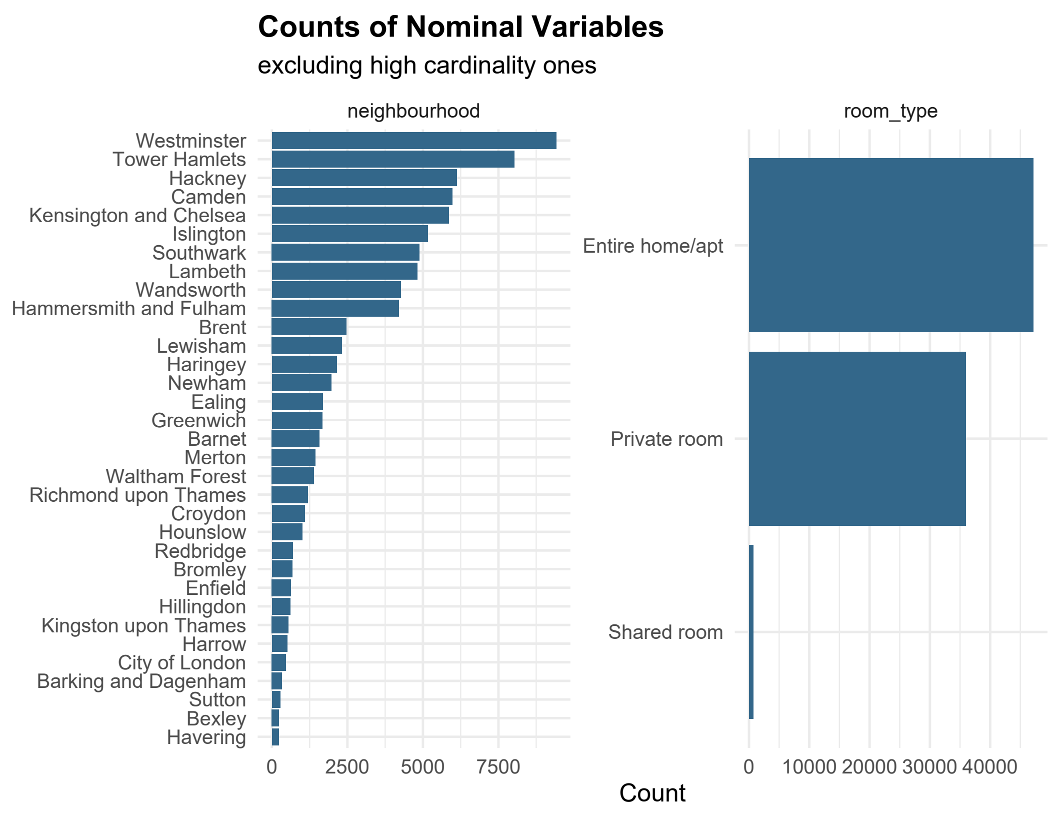
```
listings %>%
  distinct() %>%
  dim()
```

```
## [1] 83850    17
```

As they are, it is safe to say that there aren't any duplicate rows.

(c)

Nominal counts (removing high cardinality variables, that is, those that are nominal but have too many unique values to make sense to count):



Task 2

(a)

```r
listings %>%
  group_by(neighbourhood) %>%
  summarise(median_price = median(price)) %>%
  ggplot(aes(x = median_price,
             y = neighbourhood %>% fct_reorder(median_price))) +
  geom_col(fill = "#33678A") +
  labs(title = "Median Price per Night by Neigbourhood",
       x = NULL, y = NULL) +
  scale_x_continuous(labels = scales::comma_format(suffix = " GBP")) +
  theme_minimal() +
  theme(plot.title = element_text(face = "bold"))
```

(b)

```r
five_nh <- listings %>%
  count(neighbourhood, sort = T) %>%
  head(8) %>%
  pull(neighbourhood)

listings %>%
  filter(neighbourhood %in% five_nh) %>%
  count(neighbourhood, room_type) %>%
  group_by(neighbourhood) %>%
  mutate(n = n/sum(n)) %>%
  ungroup() %>%
  ggplot(aes(x = n,
             y = room_type)) +
  geom_col(fill = "#33678A") +
  labs(title = "Room Type by Neighbourhood (in %)",
       y = NULL, x = "Relative Frequency") +
  facet_wrap(~ neighbourhood) +
  scale_x_continuous(labels = scales::percent_format(),
                     limits = c(0,1)) +
  theme_minimal() +
  theme(plot.title = element_text(face = "bold"))
```

(c)

```r
listings %>%
  ggplot(aes(x = price,
             y = neighbourhood %>% fct_reorder(price))) +
  geom_boxplot(outlier.colour = NA) +
  labs(title = "Price Distribution by Neighbourhood",
       subtitle = "Outliers removed", x = NULL, y = NULL) +
  scale_x_continuous(labels = scales::comma_format(suffix = " GBP")) +
  coord_cartesian(xlim = c(0, 450)) +
  theme_minimal() +
  theme(plot.title = element_text(face = "bold"))
```

Task 3

We can highly recommend watching Julia Silge's Youtube Videos if you are interested in learning more about supervised machine learning with `tidymodels`. Also, we have our own session on `tidymodels`, which will come up soon, if you'll vote for it in the upcoming polls.