Closing Remarks

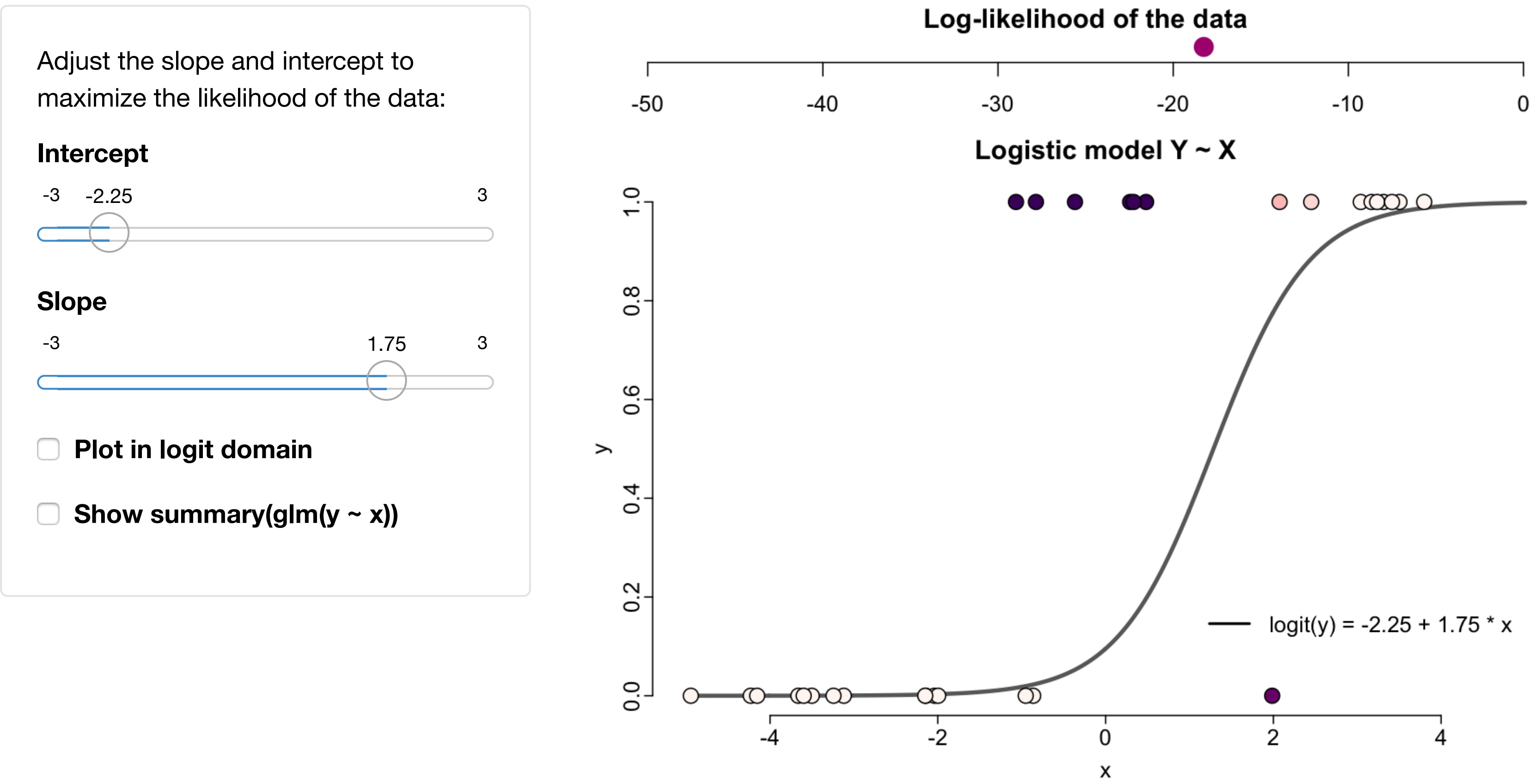
ISOM3390: Business Programming in R

A Wall of Hex Stickers

shiny

`shiny` (http://shiny.rstudio.com) is an R package that provides Web application framework for R

- R code -> interactive web page



Try to find values for the slope and intercept that minimize the residual error from the linear model.

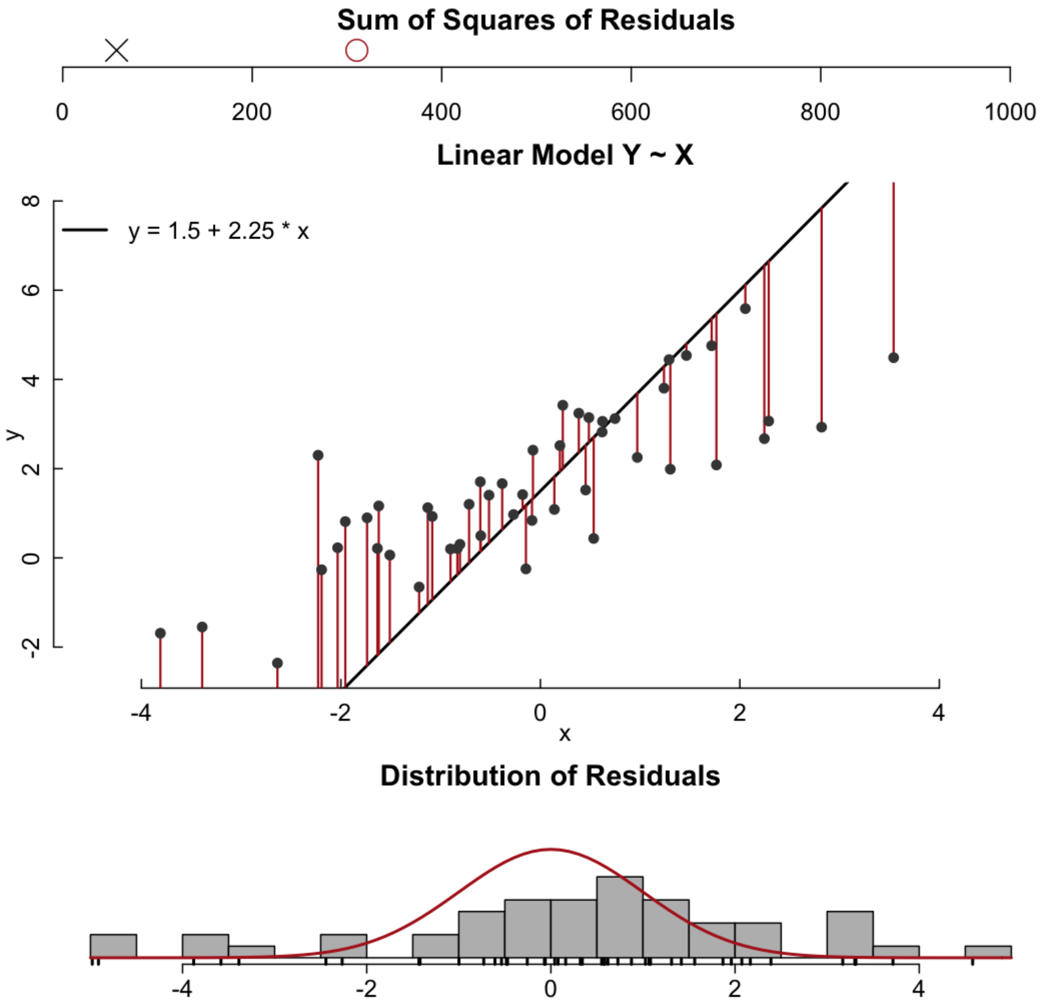
Intercept

-2 1.5 6

Slope

-1 2.25 3

☐ **Show summary(lm(y ~ x))**



The Template of `Shiny` Apps

```
library(shiny)
```

Define the user interface to take inputs and display output using `*Input()` and `*Output()` functions:

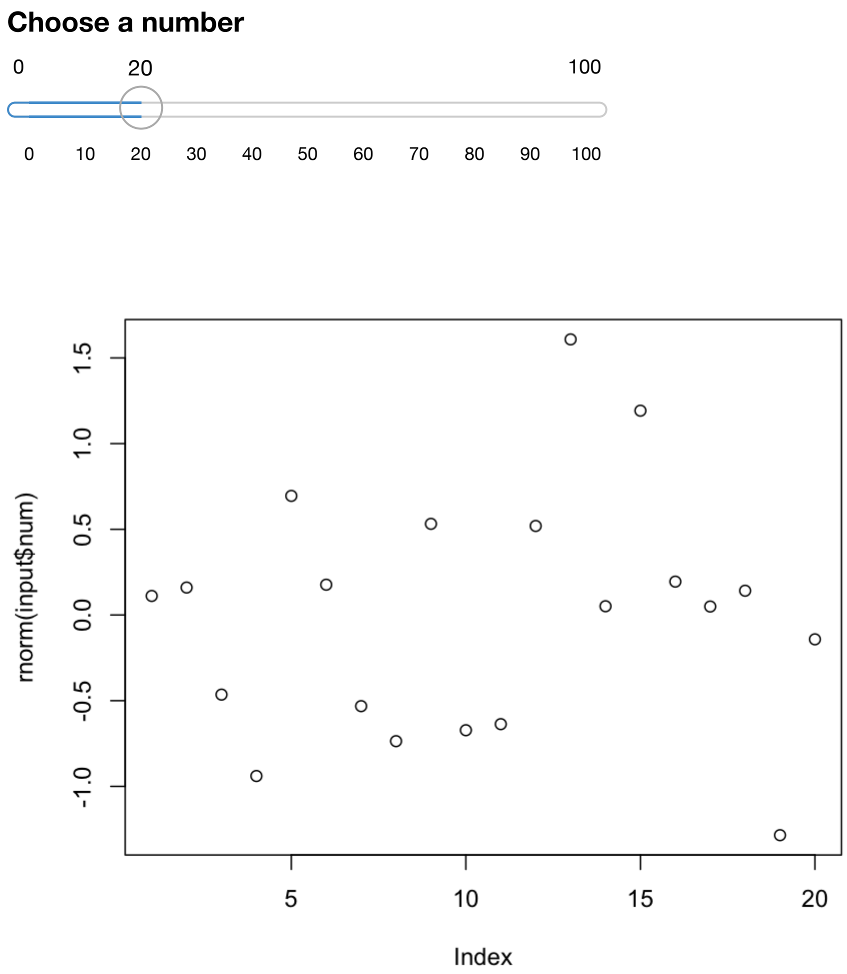
```
ui <- fluidPage(
  sliderInput("num", "Choose a number", min = 0, max = 100, value = 20),
  plotOutput("myplot"))
```

Define the sever reponse that assembles inputs into outputs with `render*()` functions:

```
server <- function(input, output) {
    output$myplot <- renderPlot({plot(rnorm(input$num))})
}
```

Create a `shiny` app object:

```
shinyApp(ui = ui, server = server)
```



`purrr` for Functional Programming

`purrr` (https://purrr.tidyverse.org/) is a core package in the `tidyverse` for functional programm

It provides a complete and consistent set of tools for working with functions and vectors.

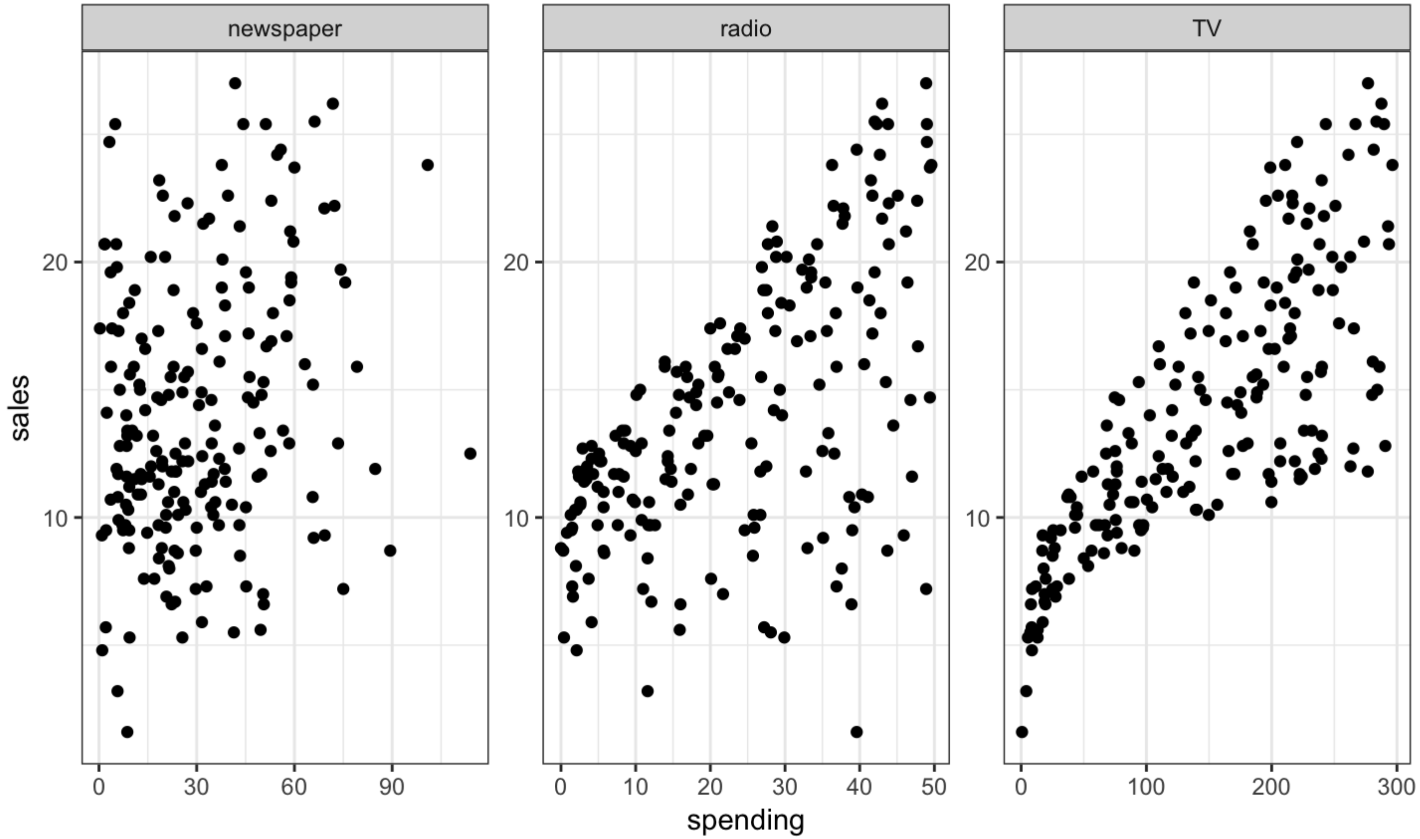
Here we have some simulated data on sales of some product in 200 different markets.

```
advertising
```

```
## # A tibble: 200 x 4
##        TV radio newspaper sales
##     <dbl> <dbl>     <dbl> <dbl>
##  1 230.   37.8      69.2  22.1
##  2  44.5  39.3      45.1  10.4
##  3  17.2  45.9      69.3   9.3
##  4 152.   41.3      58.5  18.5
##  5 181.   10.8      58.4  12.9
##  6   8.7  48.9      75     7.2
##  7  57.5  32.8      23.5  11.8
##  8 120.   19.6      11.6  13.2
##  9   8.6   2.1       1     4.8
## 10 200.    2.6      21.2  10.6
## # ... with 190 more rows
```

The advertising budgets are our **input variables**, also called **independent variables**, **features**, or **predictors**. The sales is the **output**, also called the **dependent variable** or **response**.

```
(advertising_long <- advertising %>% gather(type, spending, 1:3)) %>%
  ggplot(aes(spending, sales)) + geom_point() + facet_wrap(~ type, ncol = 3, scales = "free") + theme_bw()
```



We can split a data frame into pieces, and use `map()` in `purrr` to fit a model to each piece:

```
fit <- advertising_long %>% split(.$type) %>% map(~ lm(sales ~ spending, data = .))
```

```
## $newspaper
##
## Call:
## lm(formula = sales ~ spending, data = .)
##
## Coefficients:
## (Intercept)     spending
##    12.35141      0.05469
```

```
## $radio
##
## Call:
## lm(formula = sales ~ spending, data = .)
##
## Coefficients:
## (Intercept)     spending
##      9.3116       0.2025
```

```
## $TV
##
## Call:
## lm(formula = sales ~ spending, data = .)
##
## Coefficients:
## (Intercept)     spending
##     7.03259      0.04754
```

broom

broom (https://github.com/tidymodels/broom) is a `tidyverse` package for turning models into t

```
library(broom)
```

It provides 3 verbs to make it convenient to interact with model objects:

`tidy()` summarizes information about model components.

```
(fit_summary <- fit[[1]] %>% tidy())
```

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)  12.4       0.621      19.9  4.71e-49
## 2 spending      0.0547    0.0166      3.30 1.15e- 3
```

`glance()` reports information about the entire model.

```
fit[[1]] %>% glance()
```

```
## # A tibble: 1 x 11
##   r.squared adj.r.squared sigma statistic p.value    df logLik   AIC   BIC
## *     <dbl>         <dbl> <dbl>     <dbl>   <dbl> <int>  <dbl> <dbl> <dbl>
## 1    0.0521        0.0473  5.09      10.9 0.00115     2  -608. 1223. 1233.
## # ... with 2 more variables: deviance <dbl>, df.residual <int>
```

`augment()` adds informations about observations to a dataset.

```
fit[[1]] %>% augment()
```

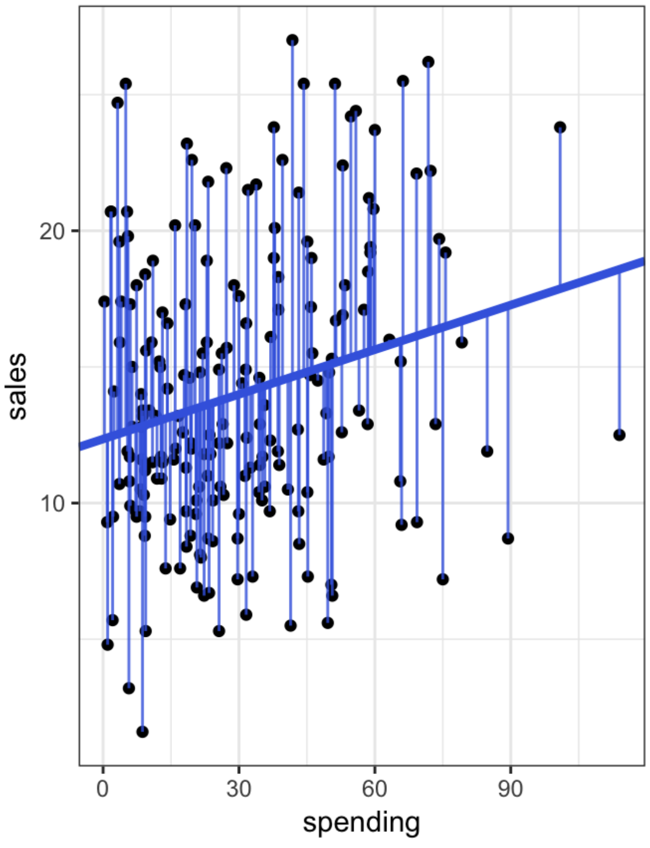
```
## # A tibble: 200 x 9
##    sales spending .fitted .se.fit .resid    .hat .sigma .cooksd .std.resid
##  * <dbl>    <dbl>   <dbl>   <dbl>  <dbl>   <dbl>  <dbl>   <dbl>      <dbl>
##  1  22.1     69.2    16.1   0.735  5.96  0.0208    5.09 1.49e-2     1.18
##  2  10.4     45.1    14.8   0.433 -4.42  0.00724   5.10 2.77e-3    -0.871
##  3   9.3     69.3    16.1   0.736 -6.84  0.0209    5.08 1.97e-2    -1.36
##  4  18.5     58.5    15.6   0.587  2.95  0.0133    5.10 2.29e-3     0.583
##  5  12.9     58.4    15.5   0.585 -2.65  0.0132    5.10 1.83e-3    -0.523
##  6   7.2     75      16.5   0.820 -9.25  0.0259    5.06 4.51e-2    -1.84
##  7  11.8     23.5    13.6   0.379 -1.84  0.00553   5.10 3.63e-4    -0.362
##  8  13.2     11.6    13.0   0.478  0.214 0.00881   5.11 7.93e-6     0.0422
##  9   4.8      1      12.4   0.608 -7.61  0.0143    5.08 1.64e-2    -1.50
## 10  10.6     21.2    13.5   0.392 -2.91  0.00593   5.10 9.80e-4    -0.573
## # ... with 190 more rows
```

`broom` tidies 100+ models from popular modelling packages and almost all of the model objects in the `stats` package that comes with base R. `vignette("available-methods")` lists method availabilty.

For more details, please see `vignette("broom")`

By plotting the variables against one another, we can see there is some sort of relationship between each medium's advertising spending and product sales:

```
fit[[1]] %>% augment() %>% ggplot(aes(spending, sales)) + geom_point() +
  geom_abline(intercept = fit_summary[[1, "estimate"]], slope = fit_summary[[2, "estimate"]], colour = "royalblue", size = 1.5) +
  geom_segment(aes(xend = spending, yend = .fitted), colour = "royalblue", alpha = 0.8) +
  coord_fixed(6) + theme_bw()
```



The Pipeline for Exploring Many Models

`tidyr::nest()` transforms the dataset into a dataframe such that every medium populats one row, and the corresponding data is stored in a list column:

```
(advertising_nested <- advertising_long %>% nest(-type))
```

```
## # A tibble: 3 x 2
##   type      data
##   <chr>     <list>
## 1 TV        <tibble [200 × 2]>
## 2 radio     <tibble [200 × 2]>
## 3 newspaper <tibble [200 × 2]>
```

Use `purrr::map()` to fit both models for each row:

```
(lm_fitted <- advertising_nested %>% mutate(model = map(data, function (df) lm(sales ~ spending, data = df))))
```

```
## # A tibble: 3 x 3
##   type      data               model
##   <chr>     <list>             <list>
## 1 TV        <tibble [200 × 2]> <S3: lm>
## 2 radio     <tibble [200 × 2]> <S3: lm>
## 3 newspaper <tibble [200 × 2]> <S3: lm>
```

Use `broom::augment()` and `broom::tidy()` to extracting models' details and associate them with corresponding rows:

```
(lm_fitted <- advertising_nested %>% mutate(model = map(data, function (df) lm(sales ~ spending, data = df)),
                                            prediction = model %>% map(augment), summary = model %>% map(tidy)))
```

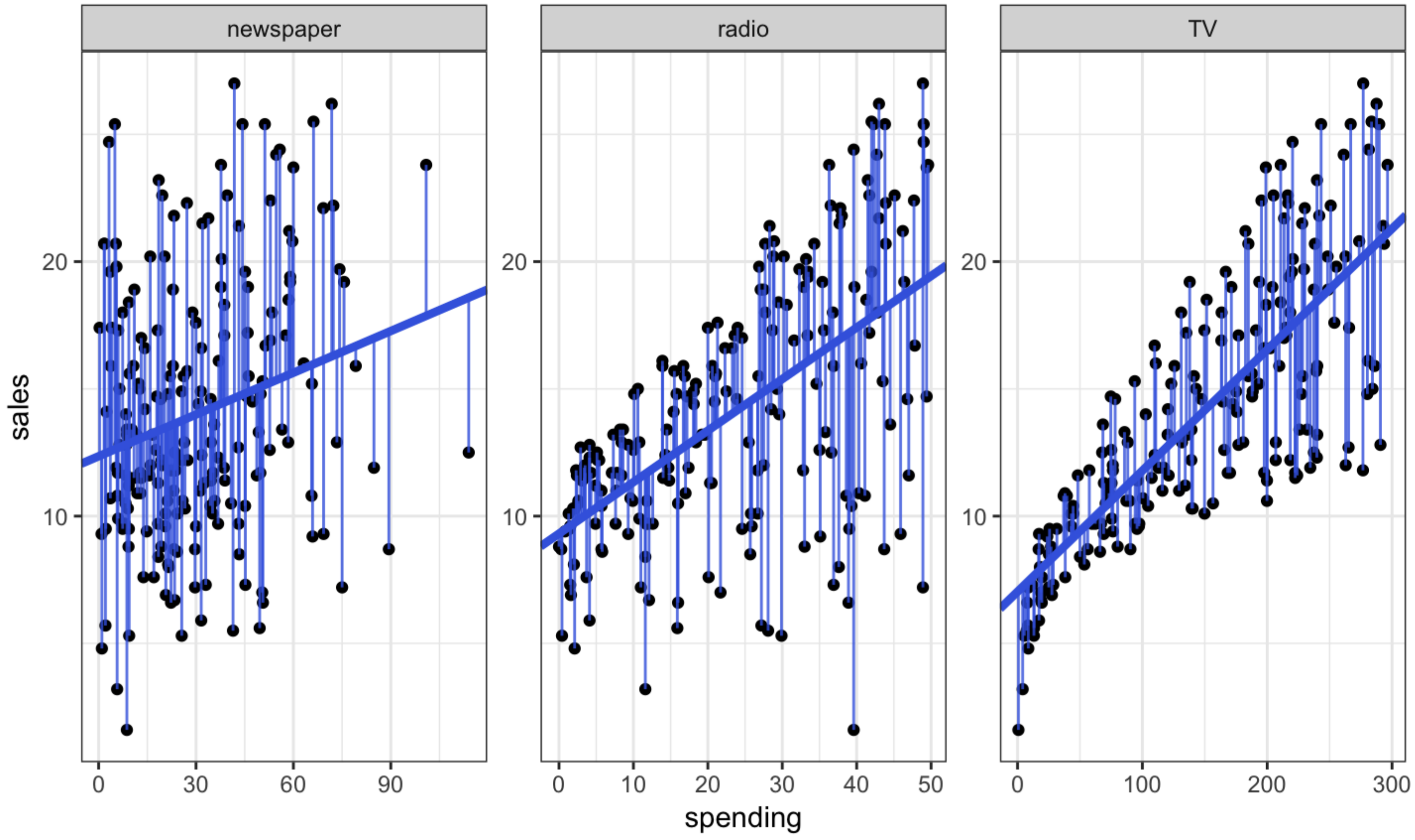
```
## # A tibble: 3 x 5
##   type      data               model    prediction         summary
##   <chr>     <list>             <list>   <list>             <list>
## 1 TV        <tibble [200 × 2]> <S3: lm> <tibble [200 × 9]> <tibble [2 × 5]>
## 2 radio     <tibble [200 × 2]> <S3: lm> <tibble [200 × 9]> <tibble [2 × 5]>
## 3 newspaper <tibble [200 × 2]> <S3: lm> <tibble [200 × 9]> <tibble [2 × 5]>
```

```
(fit_summary <- lm_fitted %>% select(type, summary) %>% unnest() %>%
  select(type, term, estimate) %>% spread(term, estimate) %>% rename(a = "(Intercept)", b = spending))
```

```
## # A tibble: 3 x 3
##   type          a      b
##   <chr>     <dbl>  <dbl>
## 1 newspaper 12.4  0.0547
## 2 radio      9.31 0.202
## 3 TV         7.03 0.0475
```

Plot model fittings with `ggplot()`:

```
lm_fitted %>% select(type, prediction) %>% unnest() %>%
  ggplot(aes(spending, sales)) + geom_point() +
  geom_abline(mapping = aes(slope = b, intercept = a), data = fit_summary, colour = "royalblue", size = 1.5) +
  geom_segment(aes(xend = spending, yend = .fitted), colour = "royalblue", alpha = 0.8) +
  facet_wrap(~ type, ncol = 3, scales = "free") + theme_bw()
```



Fitting Different Subsets with Different Models

K**-nearest neighbors regression** (K**NN regression**) uses a moving average to generate the regression line:

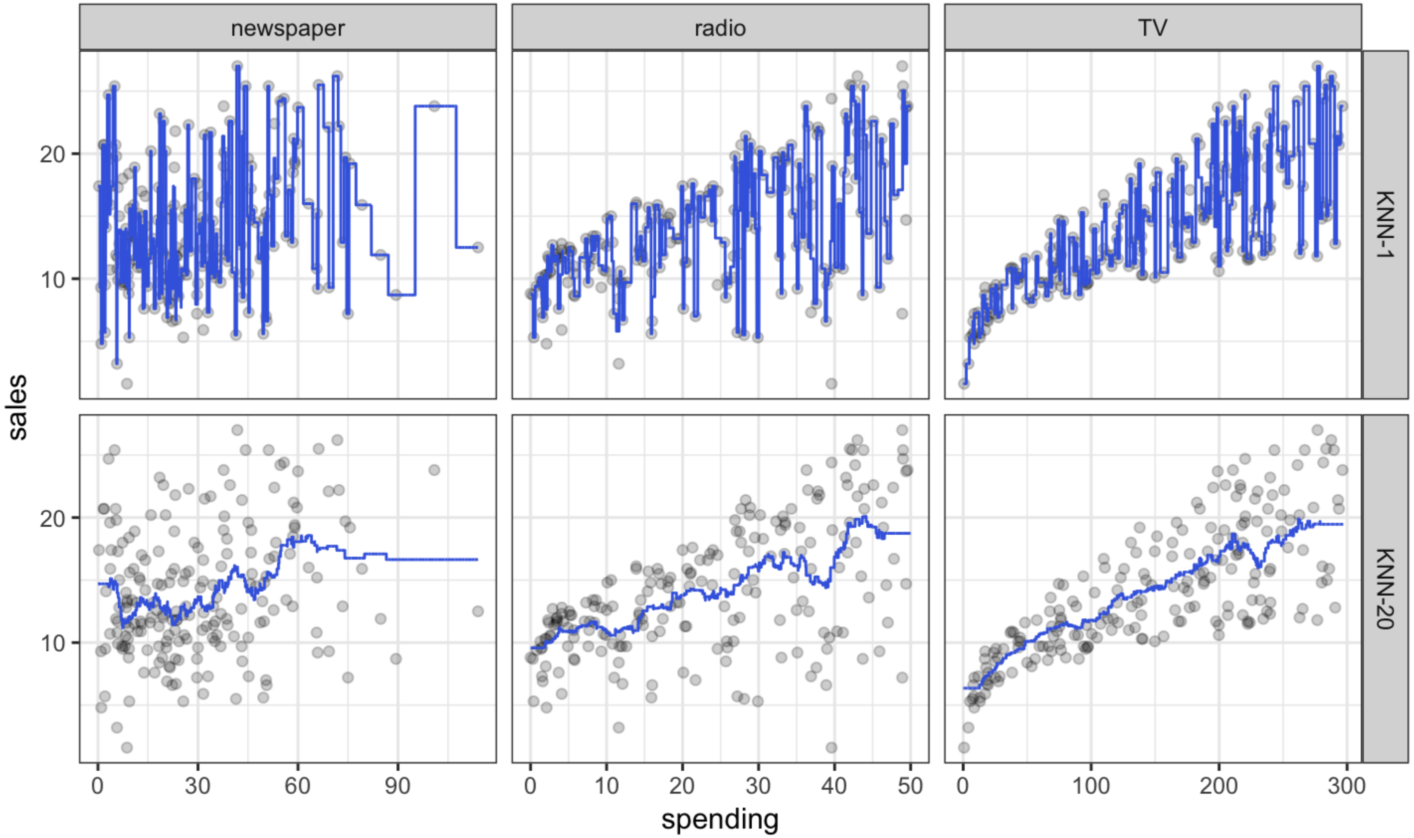
- Given a value for K and a prediction point x_0,
- Identifies the K observations closest to the prediction point x_0, and estimates a local regression line that is the average of these observations values for the outcome y.

```
(knn_fitted <- advertising_nested %>%
   mutate(knn1 = map(data, function (df) caret::knnreg(sales ~ spending, data = df, k = 1)),
          knn20 = map(data, function (df) caret::knnreg(sales ~ spending, data = df, k = 20))))
```

```
## # A tibble: 3 x 4
##   type      data               knn1         knn20
##   <chr>     <list>             <list>       <list>
## 1 TV        <tibble [200 × 2]> <S3: knnreg> <S3: knnreg>
## 2 radio     <tibble [200 × 2]> <S3: knnreg> <S3: knnreg>
## 3 newspaper <tibble [200 × 2]> <S3: knnreg> <S3: knnreg>
```

Plot model fittings with `ggplot()`:

```
ggplot(advertising_long, aes(spending, sales)) + geom_point(alpha = 0.2) +
  geom_line(data = plot_knn, mapping = aes(x = spending, y = prediction), colour = "royalblue") +
  facet_grid(model ~ type, scales = "free") +  theme_bw()
```



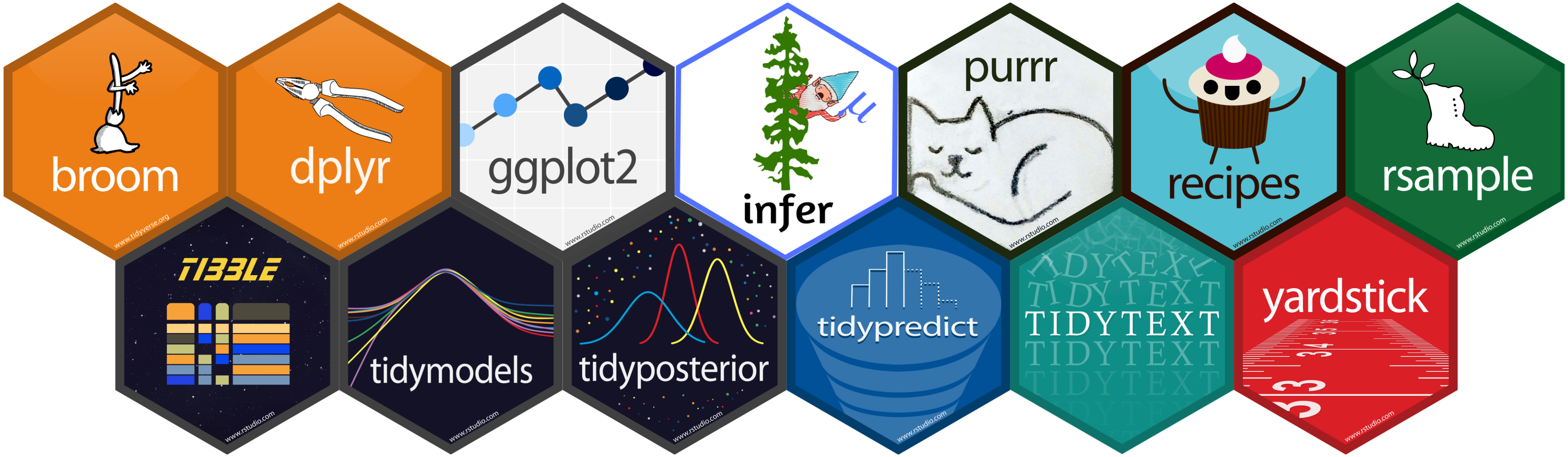
`tidymodels`

`tidymodels` (https://www.tidyverse.org/articles/2018/08/tidymodels-0-0-1/) is a "meta-package"
statistical analysis.

It shares the underlying design philosophy, grammar, and data structures of the `tidyverse`.

The core set of packages that are loaded on startup includes many of our acquainted tools: `tibble`, `broom`, `dplyr`, `ggplot2`, `purrr`, and more: `infer`, `recipes`, `rsample`, and `yardstick`.

There are a few modeling packages that are also installed along with `tidymodels`: `tidytext`, `tidypredict`, and `tidyposterior`.

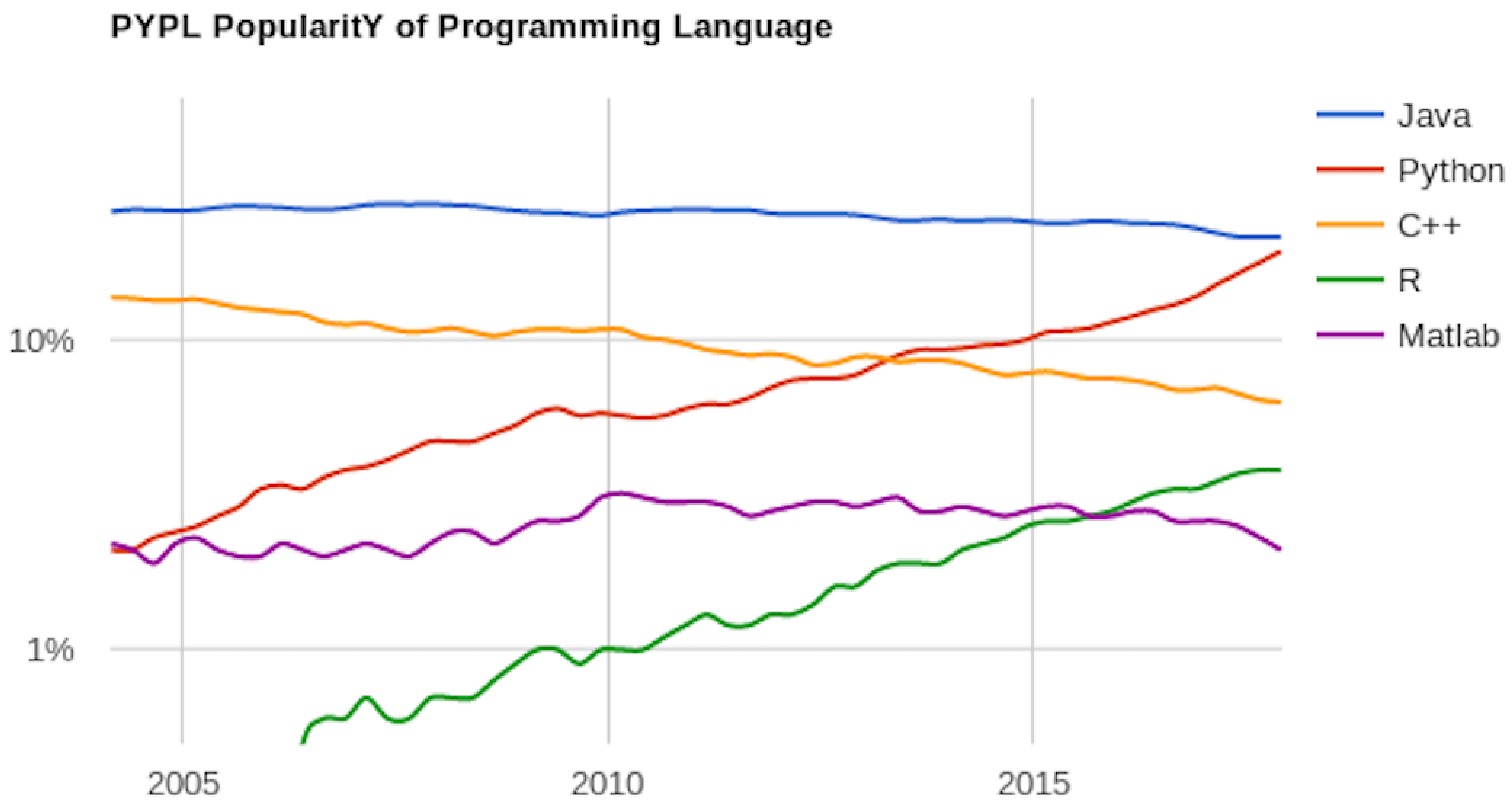


R Interfaces for Big Data Anlytics

Both **`SparkR`** (https://spark.apache.org/docs/latest/sparkr.html) (a part of the official **Spark** project) and **`sparklyr`** (https://spark.rstudio.com/) (created by the RStudio team) are R packages that provide light-weight frontends to use **Apache Spark** from R.

Who Use R?

Traditionally academics and research, now rapidly expanding into the enterprise market.



The Final Project

Problem 1: Create a choropleth map for the House election results of the 2018 US midterm election

- Scrape data from this website (https://www.bloomberg.com/graphics/2018-midterm-election-results/?view=H&filter=none)
- Map data for congressional districts can be found here (https://www.data.gov/)

Problem 2: Perform sentiment analysis on a movie review corpus created by scraping IMDb webpages

- The starting point could be this webpage (https://www.imdb.com/chart/moviemeter?sort=rk,asc&mode=simple&page=1)
- Collect user reviews (and ratings) for 100 movies, 100 reviews per movie (with spoilers excluded).
- Sample 2500 positive reviews and 2500 negative reviews to form a review corpus.
- Use list-based measures to predict sentiments, and evaluate outcomes by contrasting them with actual ratings.
- Find 10 most commonly-used positive and negative words in the corpus. For each word, plot its likelihood of occurring in reviews with different ratings.

Submission format: R markdown file with both code and detailed annotation.

Submission deadline: Dec. 20th