lab-jupyter-linear-models-baselinse

April 15, 2023

Predict Hourly Rented Bike Count using Basic Linear Regression Models

Estimated time needed: **90** minutes

0.1 Lab Overview:

Now that you have performed exploratory analysis on the bike sharing demand dataset and obtained some insights on the attributes, it's time to build predictive models to predict the hourly rented bike count using related weather and date information.

In this lab, you will be asked to use `tidymodels` to build some baseline linear regression models: - **TASK: Split data into training and testing datasets - TASK: Build a linear regression model using only the weather variables - TASK: Build a linear regression model using both weather and date variables - TASK: Evaluate the models and identify important variables**

Let's start!

First install and import the necessary libraries

[1]:
```
remove.packages("rlang")
remove.packages("tidymodels")
```

```
Removing package from ‘/home/jupyterlab/conda/envs/r/lib/R/library’
(as ‘lib’ is unspecified)
Updating HTML index of packages in '.Library'
Making 'packages.html' … done
Removing package from ‘/home/jupyterlab/conda/envs/r/lib/R/library’
(as ‘lib’ is unspecified)
Updating HTML index of packages in '.Library'
Making 'packages.html' … done
```

[2]:
```
# It may take several minutes to install those libraries in Watson Studio
install.packages("rlang")
install.packages("tidymodels")
```

```
Updating HTML index of packages in '.Library'
Making 'packages.html' … done
Updating HTML index of packages in '.Library'
Making 'packages.html' … done
```

[3]:
```
library("tidymodels")
library("tidyverse")
library("stringr")
```

```
 Attaching packages                          tidymodels 1.0.0
broom        1.0.4      recipes      1.0.5
dials        1.1.0      rsample      1.1.1
dplyr        1.1.0      tibble       3.2.0
ggplot2      3.4.1      tidyr        1.3.0
infer        1.0.4      tune         1.0.1
modeldata    1.1.0      workflows    1.1.3
parsnip      1.0.4      workflowsets 1.0.0
purrr        1.0.1      yardstick    1.1.0
 Conflicts                            tidymodels_conflicts()
purrr::discard() masks scales::discard()
dplyr::filter()  masks stats::filter()
dplyr::lag()     masks stats::lag()
recipes::step()  masks stats::step()
• Dig deeper into tidy modeling with R at https://www.tmwr.org
 Attaching packages                           tidyverse 1.3.0
readr   1.3.1      forcats 0.5.0
stringr 1.5.0
 Conflicts                              tidyverse_conflicts()
readr::col_factor() masks scales::col_factor()
purrr::discard()    masks scales::discard()
dplyr::filter()     masks stats::filter()
stringr::fixed()    masks recipes::fixed()
dplyr::lag()        masks stats::lag()
readr::spec()       masks yardstick::spec()
```

The `seoul_bike_sharing_converted_normalized.csv` will be our main dataset which has following variables:

The response variable: - `RENTED BIKE COUNT`- Count of bikes rented at each hour

Weather predictor variables: - `TEMPERATURE` - Temperature in Celsius - `HUMIDITY` - Unit is `%` - `WIND_SPEED` - Unit is `m/s` - `VISIBILITY` - Multiplied by 10m - `DEW_POINT_TEMPERATURE` - The temperature to which the air would have to cool down in order to reach saturation, unit is Celsius - `SOLAR_RADIATION` - MJ/m2 - `RAINFALL` - mm - `SNOWFALL` - cm

Date/time predictor variables: - `DATE` - Year-month-day - `HOUR`- Hour of he day - `FUNCTIONAL DAY` - NoFunc(Non Functional Hours), Fun(Functional hours) - `HOLIDAY` - Holiday/No holiday - `SEASONS` - Winter, Spring, Summer, Autumn

Let's read the dataset as a dataframe first:

[4]:
```
# Dataset URL
dataset_url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
↪cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/
↪seoul_bike_sharing_converted_normalized.csv"
```

```r
bike_sharing_df <- read_csv(dataset_url)
spec(bike_sharing_df)
```

```
Parsed with column specification:
cols(
  .default = col_double(),
  DATE = col_character(),
  FUNCTIONING_DAY = col_character()
)
See spec(...) for full column specifications.
```

```
cols(
  DATE = col_character(),
  RENTED_BIKE_COUNT = col_double(),
  TEMPERATURE = col_double(),
  HUMIDITY = col_double(),
  WIND_SPEED = col_double(),
  VISIBILITY = col_double(),
  DEW_POINT_TEMPERATURE = col_double(),
  SOLAR_RADIATION = col_double(),
  RAINFALL = col_double(),
  SNOWFALL = col_double(),
  FUNCTIONING_DAY = col_character(),
  `0` = col_double(),
  `1` = col_double(),
  `10` = col_double(),
  `11` = col_double(),
  `12` = col_double(),
  `13` = col_double(),
  `14` = col_double(),
  `15` = col_double(),
  `16` = col_double(),
  `17` = col_double(),
  `18` = col_double(),
  `19` = col_double(),
  `2` = col_double(),
  `20` = col_double(),
  `21` = col_double(),
  `22` = col_double(),
  `23` = col_double(),
  `3` = col_double(),
  `4` = col_double(),
  `5` = col_double(),
  `6` = col_double(),
  `7` = col_double(),
  `8` = col_double(),
  `9` = col_double(),
  AUTUMN = col_double(),
```

```
  SPRING = col_double(),
  SUMMER = col_double(),
  WINTER = col_double(),
  HOLIDAY = col_double(),
  NO_HOLIDAY = col_double()
)
```

We won't be using the DATE column, because 'as is', it basically acts like an data entry index. (However, given more time, we could use the DATE colum to create a 'day of week' or 'isWeekend' column, which we might expect has an affect on preferred bike rental times.) We also do not need the FUNCTIONAL DAY column because it only has one distinct value remaining (YES) after missing value processing.

[9]:
```
bike_sharing_df <- bike_sharing_df %>%
                   select(-DATE, -FUNCTIONING_DAY)
```

```
Error in `select()`:
! Can't subset columns that don't exist.
 Column `DATE` doesn't exist.
Traceback:

1. bike_sharing_df %>% select(-DATE, -FUNCTIONING_DAY)
2. withVisible(eval(quote(`_fseq`(`_lhs`)), env, env))
3. eval(quote(`_fseq`(`_lhs`)), env, env)
4. eval(quote(`_fseq`(`_lhs`)), env, env)
5. `_fseq`(`_lhs`)
6. freduce(value, `_function_list`)
7. withVisible(function_list[[k]](value))
8. function_list[[k]](value)
9. select(., -DATE, -FUNCTIONING_DAY)
10. select.data.frame(., -DATE, -FUNCTIONING_DAY)
11. tidyselect::eval_select(expr(c(…)), data = .data, error_call = error_call)
12. eval_select_impl(data, names(data), as_quosure(expr, env), include = include,
  .     exclude = exclude, strict = strict, name_spec = name_spec,
  .     allow_rename = allow_rename, allow_empty = allow_empty, allow_predicates
 ↪= allow_predicates,
  .     error_call = error_call, )
13. with_subscript_errors(out <- vars_select_eval(vars, expr, strict = strict,
  .     data = x, name_spec = name_spec, uniquely_named = uniquely_named,
  .     allow_rename = allow_rename, allow_empty = allow_empty, allow_predicates
 ↪= allow_predicates,
  .     type = type, error_call = error_call), type = type)
14. try_fetch(expr, vctrs_error_subscript = function(cnd) {
  .     cnd$subscript_action <- subscript_action(type)
  .     cnd$subscript_elt <- "column"
  .     cnd_signal(cnd)
  . })
15. withCallingHandlers(expr, condition = function(cnd) {
```

```
  .     {
  .         .__handler_frame__. <- TRUE
  .         .__setup_frame__. <- frame
  .         if (inherits(cnd, "message")) {
  .             except <- c("warning", "error")
  .         }
  .         else if (inherits(cnd, "warning")) {
  .             except <- "error"
  .         }
  .         else {
  .             except <- ""
  .         }
  .     }
  .     while (!is_null(cnd)) {
  .         if (inherits(cnd, "vctrs_error_subscript")) {
  .             out <- handlers[[1L]](cnd)
  .             if (!inherits(out, "rlang_zap"))
  .                 throw(out)
  .         }
  .         inherit <- .subset2(.subset2(cnd, "rlang"), "inherit")
  .         if (is_false(inherit)) {
  .             return()
  .         }
  .         cnd <- .subset2(cnd, "parent")
  .     }
  . })
16. vars_select_eval(vars, expr, strict = strict, data = x, name_spec =␣
 ↪name_spec,
  .     uniquely_named = uniquely_named, allow_rename = allow_rename,
  .     allow_empty = allow_empty, allow_predicates = allow_predicates,
  .     type = type, error_call = error_call)
17. walk_data_tree(expr, data_mask, context_mask)
18. eval_c(expr, data_mask, context_mask)
19. reduce_sels(node, data_mask, context_mask, init = init)
20. walk_data_tree(new, data_mask, context_mask)
21. as_indices_sel_impl(out, vars = vars, strict = strict, data = data,
  .     allow_predicates = allow_predicates, call = error_call, arg =␣
 ↪as_label(expr))
22. as_indices_impl(x, vars, call = call, arg = arg, strict = strict)
23. chr_as_locations(x, vars, call = call, arg = arg)
24. vctrs::vec_as_location(x, n = length(vars), names = vars, call = call,
  .     arg = arg)
25. (function ()
  . stop_subscript_oob(i = i, subscript_type = subscript_type, names = names,
  .     subscript_action = subscript_action, subscript_arg = subscript_arg,
  .     call = call))()
26. stop_subscript_oob(i = i, subscript_type = subscript_type, names = names,
  .     subscript_action = subscript_action, subscript_arg = subscript_arg,
```

```
  .     call = call)
27. stop_subscript(class = "vctrs_error_subscript_oob", i = i, subscript_type =
↪subscript_type,
  .     …, call = call)
28. abort(class = c(class, "vctrs_error_subscript"), i = i, …,
  .     call = vctrs_error_call(call))
29. signal_abort(cnd, .file)
30. signalCondition(cnd)
31. (function (cnd)
  . {
  .     {
  .         .__handler_frame__. <- TRUE
  .         .__setup_frame__. <- frame
  .         if (inherits(cnd, "message")) {
  .             except <- c("warning", "error")
  .         }
  .         else if (inherits(cnd, "warning")) {
  .             except <- "error"
  .         }
  .         else {
  .             except <- ""
  .         }
  .     }
  .     while (!is_null(cnd)) {
  .         if (inherits(cnd, "vctrs_error_subscript")) {
  .             out <- handlers[[1L]](cnd)
  .             if (!inherits(out, "rlang_zap"))
  .                 throw(out)
  .         }
  .         inherit <- .subset2(.subset2(cnd, "rlang"), "inherit")
  .         if (is_false(inherit)) {
  .             return()
  .         }
  .         cnd <- .subset2(cnd, "parent")
  .     }
  . })(structure(list(message = "", trace = structure(list(call = list(
  .     IRkernel::main(), kernel$run(), handle_shell(), executor$execute(msg),
  .     tryCatch(evaluate(request$content$code, envir = .GlobalEnv,
  .         output_handler = oh, stop_on_error = 1L), interrupt = function(cond)
↪interrupted <<- TRUE,
  .         error = .self$handle_error), tryCatchList(expr, classes,
  .         parentenv, handlers), tryCatchOne(tryCatchList(expr,
  .         names[-nh], parentenv, handlers[-nh]), names[nh], parentenv,
  .         handlers[[nh]]), doTryCatch(return(expr), name, parentenv,
  .         handler), tryCatchList(expr, names[-nh], parentenv, handlers[-nh]),
  .     tryCatchOne(expr, names, parentenv, handlers[[1L]]),
↪doTryCatch(return(expr),
  .         name, parentenv, handler), evaluate(request$content$code,
```

```
.         envir = .GlobalEnv, output_handler = oh, stop_on_error = 1L),
.     evaluate_call(expr, parsed$src[[i]], envir = envir, enclos = enclos,
.         debug = debug, last = i == length(out), use_try = stop_on_error !=
.             2L, keep_warning = keep_warning, keep_message = keep_message,
.         output_handler = output_handler, include_timing = include_timing),
.     timing_fn(handle(ev <- withCallingHandlers(withVisible(eval(expr,
.         envir, enclos)), warning = wHandler, error = eHandler,
.         message = mHandler))), handle(ev <-␣
↪withCallingHandlers(withVisible(eval(expr,
.         envir, enclos)), warning = wHandler, error = eHandler,
.         message = mHandler)), try(f, silent = TRUE), tryCatch(expr,
.         error = function(e) {
.             call <- conditionCall(e)
.             if (!is.null(call)) {
.                 if (identical(call[[1L]], quote(doTryCatch)))
.                   call <- sys.call(-4L)
.                 dcall <- deparse(call)[1L]
.                 prefix <- paste("Error in", dcall, ": ")
.                 LONG <- 75L
.                 sm <- strsplit(conditionMessage(e), "\n")[[1L]]
.                 w <- 14L + nchar(dcall, type = "w") + nchar(sm[1L],
.                   type = "w")
.                 if (is.na(w))
.                   w <- 14L + nchar(dcall, type = "b") + nchar(sm[1L],
.                     type = "b")
.                 if (w > LONG)
.                   prefix <- paste0(prefix, "\n  ")
.             }
.             else prefix <- "Error : "
.             msg <- paste0(prefix, conditionMessage(e), "\n")
.             .Internal(seterrmessage(msg[1L]))
.             if (!silent && isTRUE(getOption("show.error.messages"))) {
.                 cat(msg, file = outFile)
.                 .Internal(printDeferredWarnings())
.             }
.             invisible(structure(msg, class = "try-error", condition = e))
.         }), tryCatchList(expr, classes, parentenv, handlers),
.     tryCatchOne(expr, names, parentenv, handlers[[1L]]),␣
↪doTryCatch(return(expr),
.         name, parentenv, handler), withCallingHandlers(withVisible(eval(expr,
.         envir, enclos)), warning = wHandler, error = eHandler,
.         message = mHandler), withVisible(eval(expr, envir, enclos)),
.     eval(expr, envir, enclos), eval(expr, envir, enclos), bike_sharing_df %>%
.         select(-DATE, -FUNCTIONING_DAY),␣
↪withVisible(eval(quote(`_fseq`(`_lhs`)),
.         env, env)), eval(quote(`_fseq`(`_lhs`)), env, env),␣
↪eval(quote(`_fseq`(`_lhs`)),
.         env, env), `_fseq`(`_lhs`), freduce(value, `_function_list`),
```

```
.     withVisible(function_list[[k]](value)), function_list[[k]](value),
.     select(., -DATE, -FUNCTIONING_DAY), select.data.frame(.,
.         -DATE, -FUNCTIONING_DAY), tidyselect::eval_select(expr(c(…)),
.         data = .data, error_call = error_call), eval_select_impl(data,
.         names(data), as_quosure(expr, env), include = include,
.         exclude = exclude, strict = strict, name_spec = name_spec,
.         allow_rename = allow_rename, allow_empty = allow_empty,
.         allow_predicates = allow_predicates, error_call = error_call,
.         ), with_subscript_errors(out <- vars_select_eval(vars,
.         expr, strict = strict, data = x, name_spec = name_spec,
.         uniquely_named = uniquely_named, allow_rename = allow_rename,
.         allow_empty = allow_empty, allow_predicates = allow_predicates,
.         type = type, error_call = error_call), type = type),
.     try_fetch(expr, vctrs_error_subscript = function(cnd) {
.         cnd$subscript_action <- subscript_action(type)
.         cnd$subscript_elt <- "column"
.         cnd_signal(cnd)
.     }), withCallingHandlers(expr, condition = function(cnd) {
.         {
.             .__handler_frame__. <- TRUE
.             .__setup_frame__. <- frame
.             if (inherits(cnd, "message")) {
.                 except <- c("warning", "error")
.             }
.             else if (inherits(cnd, "warning")) {
.                 except <- "error"
.             }
.             else {
.                 except <- ""
.             }
.         }
.         while (!is_null(cnd)) {
.             if (inherits(cnd, "vctrs_error_subscript")) {
.                 out <- handlers[[1L]](cnd)
.                 if (!inherits(out, "rlang_zap"))
.                   throw(out)
.             }
.             inherit <- .subset2(.subset2(cnd, "rlang"), "inherit")
.             if (is_false(inherit)) {
.                 return()
.             }
.             cnd <- .subset2(cnd, "parent")
.         }
.     }), vars_select_eval(vars, expr, strict = strict, data = x,
.         name_spec = name_spec, uniquely_named = uniquely_named,
.         allow_rename = allow_rename, allow_empty = allow_empty,
.         allow_predicates = allow_predicates, type = type, error_call =␣
↪error_call),
```

```
.     walk_data_tree(expr, data_mask, context_mask), eval_c(expr,
.         data_mask, context_mask), reduce_sels(node, data_mask,
.         context_mask, init = init), walk_data_tree(new, data_mask,
.         context_mask), as_indices_sel_impl(out, vars = vars,
.         strict = strict, data = data, allow_predicates = allow_predicates,
.         call = error_call, arg = as_label(expr)), as_indices_impl(x,
.         vars, call = call, arg = arg, strict = strict), chr_as_locations(x,
.         vars, call = call, arg = arg), vctrs::vec_as_location(x,
.         n = length(vars), names = vars, call = call, arg = arg),
.     `<fn>`(), stop_subscript_oob(i = i, subscript_type = subscript_type,
.         names = names, subscript_action = subscript_action, subscript_arg =
↪subscript_arg,
.         call = call), stop_subscript(class = "vctrs_error_subscript_oob",
.         i = i, subscript_type = subscript_type, …, call = call),
.     abort(class = c(class, "vctrs_error_subscript"), i = i, …,
.         call = vctrs_error_call(call))), parent = c(0L, 1L, 2L,
. 3L, 4L, 5L, 6L, 7L, 6L, 9L, 10L, 4L, 12L, 13L, 13L, 15L, 16L,
. 17L, 18L, 19L, 13L, 13L, 13L, 23L, 0L, 25L, 25L, 27L, 28L, 29L,
. 30L, 30L, 32L, 32L, 34L, 35L, 36L, 37L, 38L, 36L, 40L, 41L, 42L,
. 43L, 44L, 45L, 46L, 47L, 0L, 49L, 50L, 51L), visible = c(TRUE,
. TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE,
. TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE,
. TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE,
. FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,
. FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE
. ), namespace = c("IRkernel", NA, "IRkernel", NA, "base", "base",
. "base", "base", "base", "base", "base", "evaluate", "evaluate",
. "evaluate", "evaluate", "base", "base", "base", "base", "base",
. "base", "base", "base", "base", NA, "base", "base", "base", NA,
. "magrittr", "base", NA, "dplyr", "dplyr", "tidyselect", "tidyselect",
. "tidyselect", "rlang", "base", "tidyselect", "tidyselect", "tidyselect",
. "tidyselect", "tidyselect", "tidyselect", "tidyselect", "tidyselect",
. "vctrs", "vctrs", "vctrs", "vctrs", "rlang"), scope = c("::",
. NA, "local", NA, "::", "local", "local", "local", "local", "local",
. "local", "::", ":::", "local", "local", "::", "::", "local",
. "local", "local", "::", "::", "::", "::", NA, "::", "::", "::",
. NA, "::", "::", NA, "::", ":::", "::", ":::", ":::", "::", "::",
. ":::", ":::", ":::", ":::", ":::", ":::", ":::", ":::", "::",
. "local", ":::", ":::", "::"), error_frame = c(FALSE, FALSE, FALSE,
. FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,
. FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,
. FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,
. FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, FALSE, FALSE,
. FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,
. FALSE, FALSE, FALSE, FALSE)), row.names = c(NA, -52L), version = 2L, class =
↪c("rlang_trace",
. "rlib_trace", "tbl", "data.frame")), parent = NULL, i = "DATE",
.     subscript_type = "character", names = c("RENTED_BIKE_COUNT",
```

```
.     "TEMPERATURE", "HUMIDITY", "WIND_SPEED", "VISIBILITY",␣
↪"DEW_POINT_TEMPERATURE",
.     "SOLAR_RADIATION", "RAINFALL", "SNOWFALL", "0", "1", "10",
.     "11", "12", "13", "14", "15", "16", "17", "18", "19", "2",
.     "20", "21", "22", "23", "3", "4", "5", "6", "7", "8", "9",
.     "AUTUMN", "SPRING", "SUMMER", "WINTER", "HOLIDAY", "NO_HOLIDAY"
.     ), subscript_action = NULL, subscript_arg = "DATE", rlang = list(
.         inherit = TRUE), call = select(., -DATE, -FUNCTIONING_DAY)), class = ␣
↪c("vctrs_error_subscript_oob",
. "vctrs_error_subscript", "rlang_error", "error", "condition")))
32. handlers[[1L]](cnd)
33. cnd_signal(cnd)
34. signal_abort(cnd)
```

1 TASK: Split training and testing data

First, we need to split the full dataset into training and testing datasets.

The training dataset will be used for fitting regression models, and the testing dataset will be used to evaluate the trained models.

TODO: Use the `initial_split()`, `training()`, and `testing()` functions to generate a training dataset consisting of 75% of the original dataset, and a testing dataset using the remaining 25%.

[10]:
```
# Use the `initial_split()`, `training()`, and `testing()` functions to split␣
↪the dataset
# With seed 1234
set.seed(1234)
bike_sharing_split<- initial_split(bike_sharing_df, prop=3/4)
# prop = 3/4
# train_data
train_data<-training(bike_sharing_split)
# test_data
test_data<-testing(bike_sharing_split)
```

2 TASK: Build a linear regression model using weather variables only

As you could imagine, weather conditions may affect people's bike renting decisions. For example, on a cold and rainy day, you may choose alternate transportation such as a bus or taxi. While on a nice sunny day, you may want to rent a bike for a short-distance travel.

Thus, can we predict a city's bike-sharing demand based on its local weather information? Let's try to build a regression model to do that.

TODO: Build a linear regression model called `lm_model_weather` using the following variables:

- `TEMPERATURE` - Temperature in Celsius

- `HUMIDITY` - Unit is `%`
- `WIND_SPEED` - Unit is `m/s`
- `VISIBILITY` - Multiplied by 10m
- `DEW_POINT_TEMPERATURE` - The temperature to which the air would have to cool down in order to reach saturation, unit is Celsius
- `SOLAR_RADIATION` - MJ/m2
- `RAINFALL` - mm
- `SNOWFALL` - cm

Define a linear regression model specification.

[14]:
```
# Use `linear_reg()` with engine `lm` and mode `regression`
lm_spec<- linear_reg() %>% set_engine(engine="lm")
lm_spec
```

```
Linear Regression Model Specification (regression)

Computational engine: lm
```

Fit a model with the response variable `RENTED_BIKE_COUNT` and predictor variables `TEMPERATURE + HUMIDITY + WIND_SPEED + VISIBILITY + DEW_POINT_TEMPERATURE + SOLAR_RADIATION + RAINFALL + SNOWFALL`

[15]:
```
# Fit the model called `lm_model_weather`
# RENTED_BIKE_COUNT ~ TEMPERATURE + HUMIDITY + WIND_SPEED + VISIBILITY +
 ↪DEW_POINT_TEMPERATURE + SOLAR_RADIATION + RAINFALL + SNOWFALL,  with the
 ↪training data
lm_model_weather<- lm_spec %>% fit(RENTED_BIKE_COUNT ~ TEMPERATURE + HUMIDITY +
 ↪WIND_SPEED + VISIBILITY + DEW_POINT_TEMPERATURE + SOLAR_RADIATION + RAINFALL
 ↪+ SNOWFALL, data=train_data)
```

Print the fit summary for the `lm_model_weather` model.

[16]:
```
# print(lm_model_weather$fit)
print(lm_model_weather$fit)
```

```
Call:
stats::lm(formula = RENTED_BIKE_COUNT ~ TEMPERATURE + HUMIDITY +
    WIND_SPEED + VISIBILITY + DEW_POINT_TEMPERATURE + SOLAR_RADIATION +
    RAINFALL + SNOWFALL, data = data)

Coefficients:
          (Intercept)            TEMPERATURE               HUMIDITY
              171.370               2210.928              -1003.792
           WIND_SPEED             VISIBILITY  DEW_POINT_TEMPERATURE
              422.516                  4.582               -100.903
      SOLAR_RADIATION               RAINFALL               SNOWFALL
             -407.845              -2090.642                331.864
```

You should see the model details such as formula, residuals, and coefficients.

3 TASK: Build a linear regression model using all variables

In addition to weather, there could be other factors that may affect bike rental demand, such as the time of a day or if today is a holiday or not.

Next, let's build a linear regression model using all variables (weather + date/time) in this task.

TODO: Build a linear regression model called `lm_model_all` using all variables `RENTED_BIKE_COUNT ~ .`

[17]:
```
# Fit the model called `lm_model_all`
# `RENTED_BIKE_COUNT ~ .` means use all other variables except for the response␣
↪variable
lm_model_all<- lm_spec %>% fit(RENTED_BIKE_COUNT ~ ., data=train_data)
```

Print the fit summary for `lm_model_all`.

[18]:
```
# summary(lm_model_all$fit)
summary(lm_model_all$fit)
```

```
Call:
stats::lm(formula = RENTED_BIKE_COUNT ~ ., data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-1410.1  -217.0    -8.7   200.7  2025.1

Coefficients: (3 not defined because of singularities)
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)            254.625     50.277   5.064 4.21e-07 ***
TEMPERATURE            590.970    208.972   2.828 0.004699 **
HUMIDITY              -974.192     97.813  -9.960  < 2e-16 ***
WIND_SPEED               3.264     39.813   0.082 0.934662
VISIBILITY               3.338     19.826   0.168 0.866281
DEW_POINT_TEMPERATURE  796.057    217.519   3.660 0.000255 ***
SOLAR_RADIATION        291.995     41.101   7.104 1.34e-12 ***
RAINFALL             -2317.209    166.583 -13.910  < 2e-16 ***
SNOWFALL               263.825     96.700   2.728 0.006384 **
`0`                    -16.895     33.520  -0.504 0.614249
`1`                   -128.323     33.375  -3.845 0.000122 ***
`10`                  -220.389     32.375  -6.807 1.09e-11 ***
`11`                  -220.593     33.175  -6.649 3.19e-11 ***
`12`                  -203.231     34.280  -5.928 3.22e-09 ***
`13`                  -179.463     34.248  -5.240 1.66e-07 ***
`14`                  -183.466     33.911  -5.410 6.52e-08 ***
```

```
`15`                       -108.625     33.647  -3.228 0.001251 **
`16`                         34.530     33.664   1.026 0.305062
`17`                        336.930     33.580  10.034  < 2e-16 ***
`18`                        797.708     34.028  23.443  < 2e-16 ***
`19`                        495.566     34.124  14.522  < 2e-16 ***
`2`                        -220.132     33.173  -6.636 3.49e-11 ***
`20`                        449.443     33.551  13.396  < 2e-16 ***
`21`                        454.048     33.940  13.378  < 2e-16 ***
`22`                        344.139     33.625  10.234  < 2e-16 ***
`23`                        103.665     33.697   3.076 0.002104 **
`3`                        -307.956     33.413  -9.217  < 2e-16 ***
`4`                        -387.267     33.047 -11.719  < 2e-16 ***
`5`                        -367.805     33.257 -11.059  < 2e-16 ***
`6`                        -203.367     33.188  -6.128 9.45e-10 ***
`7`                          92.548     33.193   2.788 0.005316 **
`8`                         454.810     32.460  14.012  < 2e-16 ***
`9`                              NA         NA      NA       NA
AUTUMN                      360.852     20.138  17.919  < 2e-16 ***
SPRING                      202.465     19.188  10.552  < 2e-16 ***
SUMMER                      212.849     28.906   7.364 2.02e-13 ***
WINTER                           NA         NA      NA       NA
HOLIDAY                     -99.749     21.812  -4.573 4.89e-06 ***
NO_HOLIDAY                       NA         NA      NA       NA
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 372.8 on 6312 degrees of freedom
Multiple R-squared:  0.6671,	Adjusted R-squared:  0.6653
F-statistic: 361.4 on 35 and 6312 DF,  p-value: < 2.2e-16
```

Now you have built two basic linear regression models with different predictor variables, let's evaluate which model has better performance,

4 TASK: Model evaluation and identification of important variables

Now that you have built two regression models, `lm_model_weather` and `lm_model_all`, with different predictor variables, you need to compare their performance to see which one is better.

In this project, you will be asked to use very important metrics that are often used in Statistics to determine the performance of a model:

1. R^2 / R-squared
2. Root Mean Squared Error (RMSE)

R-squared

R squared, also known as the coefficient of determination, is a measure to indicate how close the data is to the fitted regression line. The value of R-squared is the percentage of variation of the response variable (y) that is explained by a linear model.

Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{MSE}$$

As you know, the Mean Squared Error measures the average of the squares of errors, where 'error' is the difference between the actual value (y) and the estimated value (). Another metric that is related to MSE is **Root Mean Squared Error (RMSE)** and is simply the square root of MSE.

We first need to test the `lm_model_weather` and `lm_model_all` models against the test dataset `test_data`, and generate `RENTED_BIKE_COUNT` prediction results.

TODO: Make predictions on the testing dataset using both `lm_model_weather` and `lm_model_all` models

[19]:
```
# Use predict() function to generate test results for `lm_model_weather` and
↪`lm_model_all`
# and generate two test results dataframe with a truth column:

# test_results_weather for lm_model_weather model
test_results_weather <- lm_model_weather %>% predict(new_data=test_data) %>%
↪mutate(truth=test_data$RENTED_BIKE_COUNT)
# test_results_all for lm_model_all
test_results_all <- lm_model_all %>% predict(new_data=test_data) %>%
↪mutate(truth=test_data$RENTED_BIKE_COUNT)
```

```
Warning message in predict.lm(object = object$fit, newdata = new_data, type =
"response"):
"prediction from a rank-deficient fit may be misleading"
```

NOTE: if you happen to see a warning like : `prediction from a rank-deficient fit may be misleading`, it may be casued by collinearity in the predictor variables. Collinearity means that one predictor variable can be predicted from other predictor variables to some degree. For example, `RAINFALL` could be predicted by `HUMIDITY`.

But dont worry, you will address `glmnet` models (Lasso and Elastic-Net Regularized Generalized Linear Models) instead of regular `regression` models to solve this issue and futher improve the model performance.

Next, let's calculate and print the R-squared and RMSE for the two test results

TODO: Use `rsq()` and `rmse()` functions to calculate R-squared and RMSE metrics for the two test results

[20]:
```
# rsq_weather <- rsq(...)
# rsq_all <- rsq(...)
rsq_weather <- rsq(test_results_weather, truth=truth, estimate=.pred)
rsq_all <- rsq(test_results_all, truth=truth, estimate=.pred)
# rmse_weather <- rmse(...)
# rmse_all <- rmse(...)
```

```
rmse_weather <- rmse(test_results_weather, truth=truth, estimate=.pred)
rmse_all <- rmse(test_results_all, truth=truth, estimate=.pred)
```

From these tables, you should find that the test results from `lm_model_all` are much better. It means that using both weather and datetime variables in the model generates better prediction results.

Since `lm_model_all` has many predictor variables, let's check which predictor variables have larger coefficients. Variables with larger coefficients in the model means they attribute more in the prediction of `RENTED_BIKE_COUNT`. In addition, since all predictor variables are normalized to the same scale, 0 to 1, we thus can compare their coefficients directly.

You could try building another regression model using the non-normalized `seoul_bike_sharing_converted.csv` dataset, and you would find that the coefficients are much different.

First let's print all coefficients:

[21]:
```
lm_model_all$fit$coefficients
```

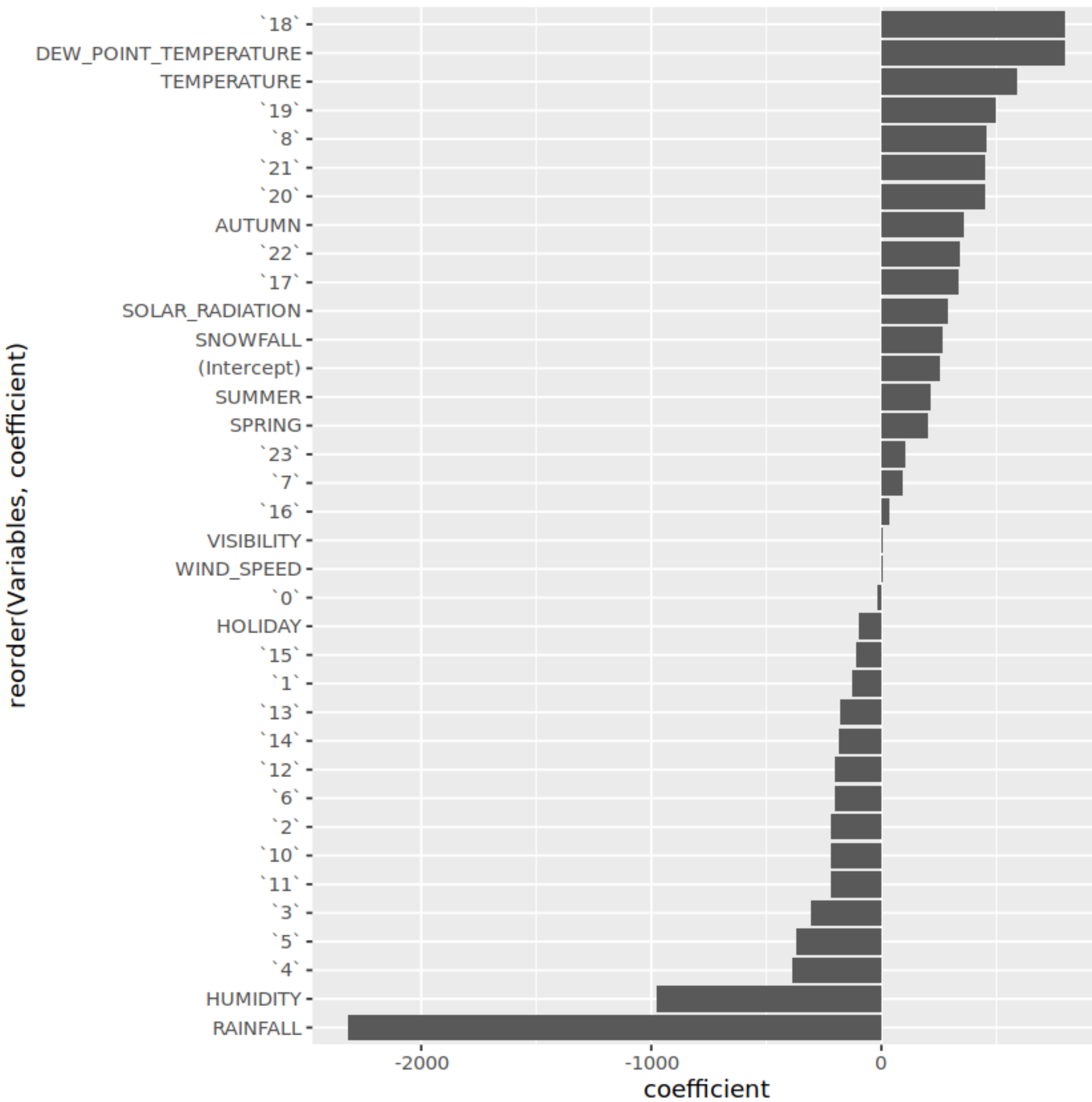
(Intercept) 254.624713841887 **TEMPERATURE** 590.970013169704 **HUMIDITY** -974.191665853887 **WIND__SPEED** 3.26403322110029 **VISIBILITY** 3.33846758815491 **DEW__POINT__TEMPERATURE** 796.057154996279 **SOLAR__RADIATION** 291.99491187606 **RAINFALL** -2317.20898113899 **SNOWFALL** 263.824807690223 **'0'** -16.895308187225 **'1'** -128.323156930108 **'10'** -220.388845946249 **'11'** -220.593196981749 **'12'** -203.230668526479 **'13'** -179.463293490874 **'14'** -183.46633365467 **'15'** -108.625201366895 **'16'** 34.5303135068688 **'17'** 336.929992543532 **'18'** 797.708010787832 **'19'** 495.566233185417 **'2'** -220.132469200938 **'20'** 449.442738955973 **'21'** 454.047517540653 **'22'** 344.138641820235 **'23'** 103.665012190421 **'3'** -307.955780172193 **'4'** -387.267047105129 **'5'** -367.804644005323 **'6'** -203.366761849977 **'7'** 92.5484678840224 **'8'** 454.810323199464 **'9'** <NA> **AUTUMN** 360.852377661853 **SPRING** 202.464657365026 **SUMMER** 212.849374276925 **WINTER** <NA> **HOLIDAY** -99.7491194802854 **NO__HOLIDAY** <NA>

hmm, it's not very clear to compare the coefficients from a long and unsorted list. Next, you need to sort and visualize them using a bar chart

TODO: Sort the coefficient list in descending order and visualize the result using `ggplot` and `geom_bar`

[22]:
```
# Sort coefficient list
coefficient<-sort(lm_model_all$fit$coefficients, decreasing=TRUE)
Coefficients_df<-data.frame(coefficient)
Variables<-factor(row.names(Coefficients_df))
```

[23]:
```
# Visualize the list using ggplot and geom_bar
ggplot(Coefficients_df,aes(reorder(Variables,coefficient) , coefficient))+
 ↪geom_col()+  coord_flip()
```



You should see a sorted coefficient bar chart like the following example:

Mark down these 'top-ranked variables by coefficient', which will be used for model refinments in the next labs.

Note that here the main reason we use absolute value is to easily identify important variables, i.e. variables with large magnitudes, no matter it's negative or positive. If we want to interprete the model then it's better to seperate the positive and negative coefficients.

5 Next Steps

Great! Now you have built a baseline linear regression model to predict hourly bike rent count, with reasonably good performance. In the next lab, you will be refining the baseline model to improve its performance.

5.1 Authors

Yan Luo

5.1.1 Other Contributors

Jeff Grossman

5.2 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-04-08	1.0	Yan	Initial version created

##