Feture Engineering in R

Natalie Fisher

Last modified on April 29, 2025 12:02:06 Eastern Daylight Time

Table of contents

Preface

This material is from the DataCamp course Feature Engineering in R by Jorge Zazueta.

Course Description: In this course, you'll learn about feature engineering, which is at the heart of many times of machine learning models. As the performance of any model is a direct consequence of the features it's fed, feature engineering places domain knowledge at the center of the process. You'll become acquainted with principles of sound feature engineering, helping to reduce the number of variables where possible, making learning algorithms run faster, improving interpretability, and preventing overfitting.

You will learn how to implement feature engineering techniques using the R tidymodels framework, emphasizing the recipe package that will allow you to create, extract, transform, and select the best features for your model.

When faced with a new dataset, you will be able to identify and select relevant features and disregard non-informative ones to make your model run faster without sacrificing accuracy. You will also become comfortable applying transformations and creating new features to make your models more efficient, interpretable, and accurate

Reminder to self: each `*.qmd` file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

1 Introducing Feature Engineering

Raw data does not always come in its best shape for analysis. In this opening chapter, you will get a first look at how to transform and create features that enhance your model's performance and interpretability.

What is feature engineering? - (video)

1.1 A Tentative Model

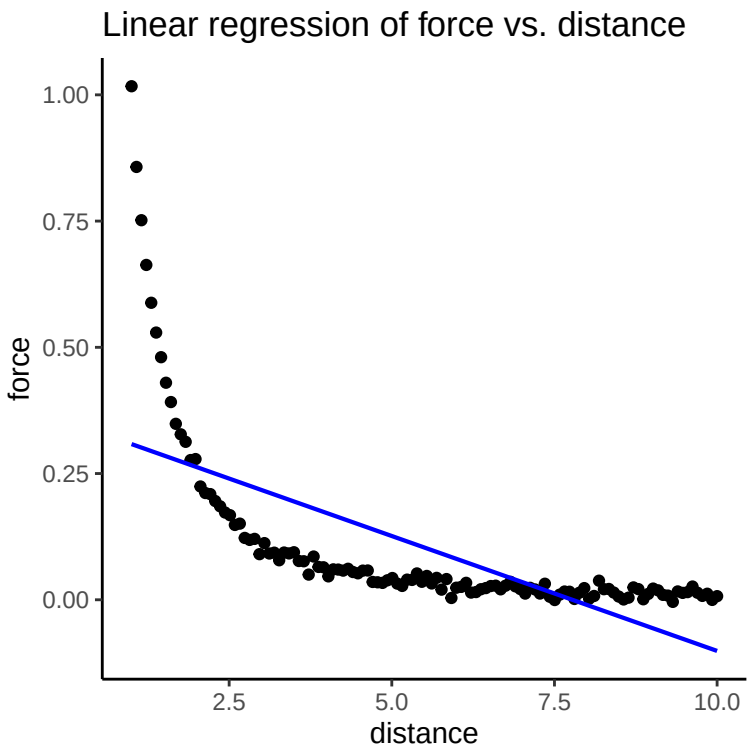
You are handed a data set with measures of the gravitational force between two bodies at different distances and are challenged to build a simple model to predict such force given a specific distance. Initially, you want to stick to simple linear regression. The data consist of 120 pairs of `distance` and `force`, and is loaded for you as `newton`.

Instructions

- Build a linear model for the `newton` data using the linear model from base `R` function and assign it to `lr_force`.
- Create a new data frame `df` by binding the prediction values to the original `newton` data.
- Generate a scatterplot of force versus distance using `ggplot()`.
- Add a regression line to the scatterplot with the fitted values.

```
# Build a linear model for the newton the data and assign it to lr_force
lr_force <- lm(force ~ distance, data = newton)

# Create a new data frame by binding the prediction values to the original data
df <- newton %>% bind_cols(lr_pred = predict(lr_force))

# Generate a scatterplot of force vs. distance
df %>%
```

```
  ggplot(aes(x = distance, y = force)) +
  geom_point() +
# Add a regression line with the fitted values
  geom_line(aes(y = lr_pred), color = "blue", lwd = .75) +
  ggtitle("Linear regression of force vs. distance") +
  theme_classic()
```



1.2 Manually engineering a feature

After doing some research with your team, you recall that the gravitational force of attraction between two bodies obeys Newton's formula:

$$F = G\frac{m_1 m_2}{r^2}$$

You can't use the formula directly because the masses are unknown, but you can fit a regression model of `force` as a function of `inv_square_distance`. The augmented dataset `df` you built in the previous exercise has been loaded for you.

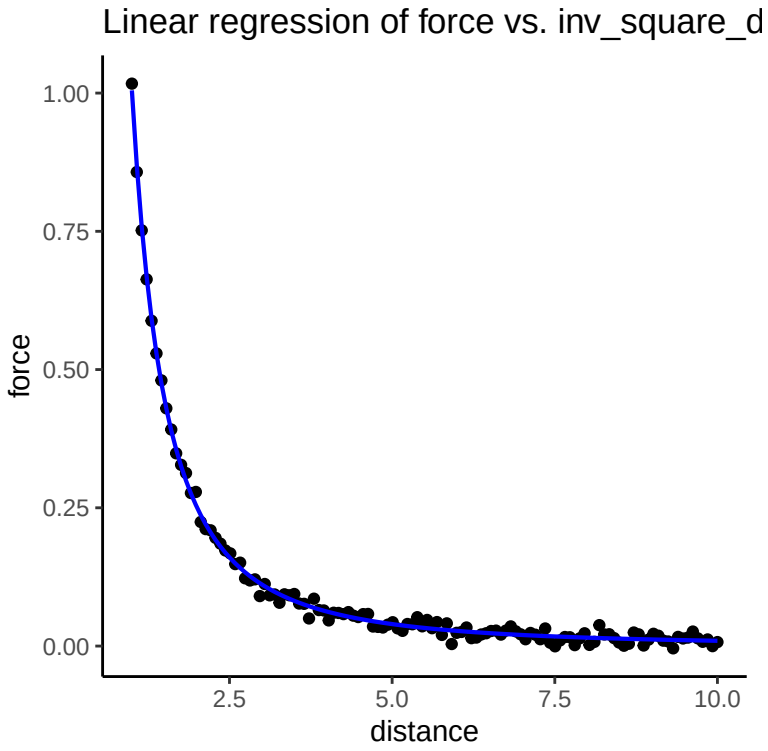
Instructions

- Create a new variable `inv_square_distance` defined as the reciprocal of the squared distance and add it to the `df` data frame.

*Build a simple regression model using `lm()` of force versus `inv_square_distance` and save it as `lr_force_2`.

*Bind your predictions to `df_inverse`.

```
# Create a new variable inv_square_distance
df_inverse <- df %>% mutate(inv_square_distance = 1/distance^2)

# Build a simple regression model
lr_force_2 <- lm(force ~ inv_square_distance, data = df_inverse)

# Bind your predictions to df_inverse
df_inverse <- df_inverse %>% bind_cols(lr2_pred = predict(lr_force_2))

df_inverse %>% ggplot(aes(x = distance, y = force)) +
  geom_point() +
  geom_line(aes(y = lr2_pred), col = "blue", lwd = .75) +
  ggtitle("Linear regression of force vs. inv_square_distance") +
  theme_classic()
```



Creating new features using domain knowledge - (video)

1.3 Setting up your data for analysis

ou will look at a version of the nycflights13 dataset, loaded as flights. It contains information on `flights` departing from New York City. You are interested in predicting whether or not they will arrive late to their destination, but first, you need to set up the data for analysis.

After discussing our model goals with a team of experts, you selected the following variables for your model: `flight`, `sched_dep_time`, `dep_delay`, `sched_arr_time`, `carrier`, `origin`, `dest`, `distance`, `date`, `arrival`.

You will also `mutate()` the date using `as.Date()` and convert character type variables to factors.

Lastly, you will split the data into `train` and `test` datasets.

Instructions

- Transform all character-type variables to factors.
- Split the flights data into test and train sets.

```
set.seed(24601)
flights <- flights %>%
  slice_sample(n= 18182)
#str(flights)
```

```
flights <- flights %>%
  mutate(date = make_date(year, month, day)) %>%
  mutate(arr = sched_arr_time - arr_time) %>%
  mutate(arrival = ifelse(arr <= -50, "late", "on-time")) %>%
  select(flight, sched_dep_time, dep_delay, sched_arr_time, carrier, origin, dest, distanc
  mutate(date = as.Date(date), across(where(is.character), as.factor))
```

```
# Split the flights data into test and train sets
set.seed(246)
split <- flights %>% initial_split(prop = 3/4, strata = arrival)
test <- testing(split)
train <- training(split)

test %>% select(arrival) %>% table() %>% prop.table()
```

```
arrival
     late   on-time
0.1640236 0.8359764
```

```
train %>% select(arrival) %>% table() %>% prop.table()
```

```
arrival
     late   on-time
0.1669435 0.8330565
```

1.4 Building a workflow

With your data ready for analysis, you will declare a `logistic_model()` to predict whether or not they will arrive late.

You assign the role of "ID" to the `flight` variable to keep it as a reference for analysis and debugging. From the `date` variable, you will create new features to explicitly model the effect of holidays and represent factors as `dummy` variables.

Bundling your *model* and `recipe()` together using `workflow()` will help ensure that subsequent fittings or predictions will implement consistent feature engineering steps.

Instructions

- Assign an "ID" role to `flight`.
- Bundle the model and the recipe into a `workflow` object.
- Fit `lr_workflow` to the `test` data.
- Tidy the fitted `workflow`.

```
lr_model <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

# Assign an "ID" role to flight
lr_recipe <- recipe(arrival ~., data = train) %>% update_role(flight, new_role = "ID") %>%
  step_holiday(date, holidays = timeDate::listHolidays("US")) %>% step_dummy(all_nominal_p

# Bundle the model and the recipe into a workflow object
lr_workflow <- workflow() %>% add_model(lr_model) %>% add_recipe(lr_recipe)
lr_workflow
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: logistic_reg()

-- Preprocessor ----------------------------------------------------------------
2 Recipe Steps

* step_holiday()
* step_dummy()

-- Model -----------------------------------------------------------------------
Logistic Regression Model Specification (classification)

Computational engine: glm
```

```r
# Fit lr_workflow workflow to the test data
lr_fit <- lr_workflow %>% fit(data = test)

# Tidy the fitted workflow
tidy(lr_fit)
```

```
# A tibble: 138 x 5
   term                          estimate   std.error statistic   p.value
   <chr>                            <dbl>       <dbl>     <dbl>     <dbl>
 1 (Intercept)                   -34.5      3956.         -0.00873 9.93e-  1
 2 sched_dep_time                 -0.000572    0.000162   -3.53    4.10e-  4
 3 dep_delay                      -0.0365      0.00165   -22.1     4.69e-108
 4 sched_arr_time                  0.000502    0.000149    3.37    7.52e-  4
 5 distance                        0.0307      0.0142      2.16    3.07e-  2
 6 date                           -0.000262    0.000503   -0.520   6.03e-  1
 7 date_USChristmasDay             0.935       1.24        0.752   4.52e-  1
 8 date_USColumbusDay             15.2       959.          0.0159  9.87e-  1
 9 date_USCPulaskisBirthday       15.9      1687.          0.00941 9.92e-  1
10 date_USDecorationMemorialDay    0.612       1.12        0.549   5.83e-  1
# i 128 more rows
```

Increasing the information content of raw data -(video)

1.5 Identifying missing values

Attrition is a critical issue for corporations, as losing an employee implies not only the cost of recruiting and training a new one, but constitutes a loss in tacit knowledge and culture that is hard to recover.

The `attrition` dataset has information on employee attrition including `Age`, `WorkLifeBalance`, `DistanceFromHome`, `StockOptionLevel`, and 27 others. Before continuing with your analysis, you want to detect any missing variables.

The package `naniar` and the `attrition` dataset are already loaded for you.

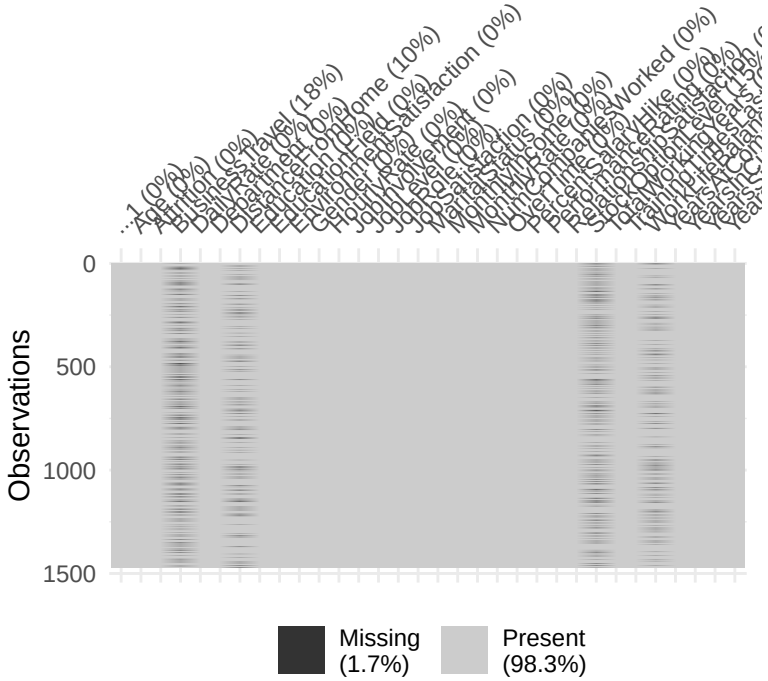
```r
library(naniar)
set.seed(345)
attrition <- attrition %>%
  mutate(BusinessTravel = replace(BusinessTravel, sample(row_number(), size = ceiling(.18*
                                  DistanceFromHome = replace(DistanceFromHome, sample(row_
```

```
                                    StockOptionLevel = replace(StockOptionLevel, sample(row_
                                    WorkLifeBalance = replace(WorkLifeBalance, sample(row_nu

attrition$...1 <- 1:1470
attrition <- attrition %>%
  relocate(...1, .before = Age)

set.seed(89)
attrition_split <- initial_split(attrition, prop = .75, strata = Attrition)
train <- training(attrition_split)
test <- testing(attrition_split)
```

Instructions

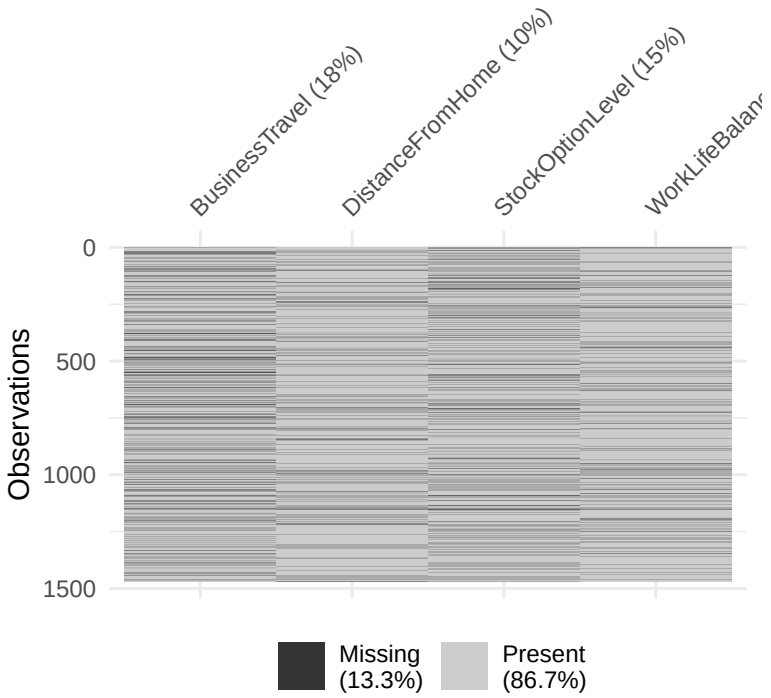
- Explore missing data visually to identify missing values on the `attrition` data.

```
# Explore missing data on the attrition dataset
vis_miss(attrition)
```



- Select the variables with missing values and visualize only those.

```
# Select the variables with missing values and rerun the analysis on those variables.
attrition %>%
  select("BusinessTravel", "DistanceFromHome",
         "StockOptionLevel", "WorkLifeBalance") %>%
  vis_miss()
```



1.6 Imputing missing values and creating dummy variables

After detecting missing values in the attrition `dataset` and determining that they are missing completely at random (MCAR), you decide to use K Nearest Neighbors (KNN) imputation. While configuring your feature engineering `recipe`, you decide to create dummy variables for all your nominal variables and update the role of the `...1` variable to "ID" so you can keep it in the dataset for reference, without affecting your model.

Instructions

- Update the role of `...1` to "ID".
- Impute values to all predictors where data are missing.

- Create dummy variables for all nominal predictors.

```
r_model <- logistic_reg()

lr_recipe <-
  recipe(Attrition ~., data = train) %>%

# Update the role of "...1" to "ID"
  update_role(...1, new_role = "ID" ) %>%

# Impute values to all predictors where data are missing
  step_impute_knn(all_nominal_predictors()) %>%

# Create dummy variables for all nominal predictors
  step_dummy(all_nominal_predictors())

lr_recipe
```

1.7 Fitting and assessing the model

Now that you have addressed missing values and created dummy variables, it is time to assess your model's performance!

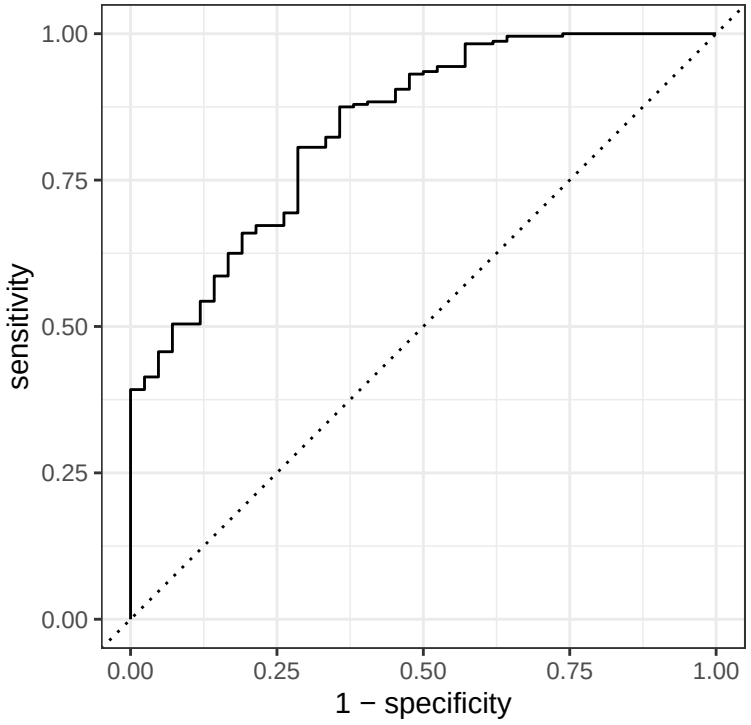
The `attrition` dataset, along with the `test` and `train` splits, the `lr_recipe` and your declared `logistic_model()` are all loaded for you.

Instructions

- Bundle model and recipe in workflow.
- Fit workflow to the train data.
- Generate an augmented data frame for performance assessment.

```
# Bundle model and recipe in workflow
lr_workflow <- workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe)

# Fit workflow to the train data
```

```r
lr_fit <- fit(lr_workflow, data = train)

# Generate an augmented data frame for performance assessment
lr_aug <- lr_fit %>% augment(test)

lr_aug %>% roc_curve(truth = Attrition, .pred_No) %>% autoplot()
```




```r
bind_rows(lr_aug %>% roc_auc(truth = Attrition, .pred_No),
          lr_aug %>% accuracy(truth = Attrition, .pred_class))
```

```
# A tibble: 2 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 roc_auc  binary         0.840
2 accuracy binary         0.865
```

1.8 Predicting Hotel Bookings

You just got a job at a hospitality research company, and your first task is to build a model that predicts whether or not a hotel stay will include children. To train your model, you will rely on a modified version of the hotel booking demands dataset by Antonio, Almeida, and Nunes (2019). You are restricting your data to the following features:

```
features <- c('adults',
              'children', 'meal',
              'reserved_room_type',
              'customer_type',
              'arrival_date')
```

```
library(lubridate)
hotel_booking <- read_csv("./DATA/hotel_booking.csv")
hotel_booking <- hotel_booking %>%
  mutate(month_no = match(arrival_date_month, month.name)) %>%
  mutate(arrival_date = make_date(arrival_date_year, month_no, arrival_date_day_of_month))
  mutate(children = ifelse( children == 0, "none", "children"))

set.seed(90)
hotel_booking %>%
  sample_n(size = 50000) %>%
  select("hotel", 'adults',
              'children', 'meal',
              'reserved_room_type',
              'customer_type',
              'arrival_date') %>%
  mutate_if(is.character, as.factor) -> hotelsOL

hotelsOL$...1 <- 1:50000
hotelsOL <- hotelsOL %>%
  relocate(...1, .before = hotel)
```

```
set.seed(4)
hotel_split <- initial_split(hotelsOL, prop = .75, strata = hotel)

train <- training(hotel_split)
test <- testing(hotel_split)

lr_model <- logistic_reg()
```

The data has been loaded for you as `hotels`, along with its corresponding `test` and `train` splits, and the model has been declared as `lr_model <- logistic_reg()`.

You will assess model performance by accuracy and area under the *ROC* curve or *AUC*.

##Instructions {-}

- Generate "day of the week", "week" and "month" features.
- Create dummy variables for all nominal predictors.

```
lr_recipe <-
  recipe(children ~., data = train) %>%
# Generate "day of the week", "week" and "month" features

  step_date(arrival_date, features = c("dow", "week", "month")) %>%

# Create dummy variables for all nominal predictors
  step_dummy(all_nominal_predictors())
```

- Bundle your model and recipe in a `workflow()`.
- Fit the workflow to the training data.

```
# Bundle your model and recipe in a workflow
lr_workflow <-workflow() %>% add_model(lr_model) %>% add_recipe(lr_recipe)

# Fit the workflow to the training data

lr_fit <-  lr_workflow %>%
  fit(data = train)
lr_aug <- lr_fit %>%
  augment(test)

bind_rows(roc_auc(lr_aug,truth = children, .pred_children),accuracy(lr_aug,truth = childre
```

```
# A tibble: 2 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 roc_auc  binary         0.853
2 accuracy binary         0.948
```

2 Transforming Features

In this chapter, you'll learn that, beyond manually transforming features, you can leverage tools from the tidyverse to engineer new variables programmatically. You'll explore how this approach improves your models' reproducibility and is especially useful when handling datasets with many features.

Why transform existing features? -(video)

2.1 Glancing at your data

```
library(modeldata)
attrition_num <- attrition %>%
  select(Attrition, Age, DailyRate, DistanceFromHome, HourlyRate, JobLevel, MonthlyIncome,
```

The dataset `attrition_num` has been loaded for you. Use the console to write code as necessary to answer the following questions: (1) How many observations are in the dataset? (2) How many numeric predictors? and (3) How many factor variables?

- **There are 1,470 observations, 16 numeric predictors and one factor variable.**
- There are 1,470 observations, 17 numeric predictors and one factor variable.
- There are 1,490 observations, 0 numeric predictors and two factor variables.

2.2 Normalizing and log-transforming

You are handed a dataset, `attrition_num` with numerical data about employees who left the company. Features include `Age`, `DistanceFromHome`, and `MonthlyRate`.

You want to use this data to build a model that can predict if an employee is likely to stay, denoted by `Attrition`, a binary variable coded as a `factor`. In preparation for modeling, you want to reduce possible skewness and prevent some variables from outweighing others due to variations in scale.

The `attrition_num` data and the `train` and `test` splits are loaded for you.

Instructions

- Normalize all numeric predictors.
- Log-transform all numeric features, with an offset of one.

```
attrition_split <- initial_split(attrition_num, prop = 3/4, strata = Attrition)

train <- training(attrition_split)
test <- testing(attrition_split)
```

```
lr_model <- logistic_reg()

lr_recipe <-
  recipe(Attrition~., data = train) %>%

# Normalize all numeric predictors
  step_normalize(all_numeric_predictors()) %>%

# Log-transform all numeric features, with an offset of one
  step_log(all_nominal_predictors(), offset = 1)

lr_workflow <-
  workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe)

lr_workflow
```

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: logistic_reg()

-- Preprocessor ----------------------------------------------------------------
2 Recipe Steps

* step_normalize()
* step_log()
```

```
-- Model ---------------------------------------------------------------------
Logistic Regression Model Specification (classification)

Computational engine: glm
```

2.3 Fit and augment

With your `lr_workflow` ready to go, you can fit it to the `test` data to make predictions.

For model assessment, it is convenient to augment your fitted object by adding predictions and probabilities using `augment()`.

Instructions

- Fit the workflow to the train data.
- Augment the lr_fit object using the test data to get it ready for assessment.

```
# Fit the workflow to the train data
lr_fit <-
  fit(lr_workflow, data = train)

# Augment the lr_fit object
lr_aug <-
  augment(lr_fit, new_data = test)

lr_aug
```

```
# A tibble: 369 x 20
   .pred_class .pred_No .pred_Yes Attrition   Age DailyRate DistanceFromHome
 * <fct>          <dbl>     <dbl> <fct>     <int>     <int>            <int>
 1 No             0.865   0.135   No           36      1299               27
 2 No             0.886   0.114   No           35       809               16
 3 No             0.747   0.253   No           22      1123               16
 4 No             0.955   0.0446  No           53      1219                2
 5 No             0.738   0.262   Yes          36      1218                9
 6 No             0.961   0.0393  No           34       419                7
 7 No             0.994   0.00593 No           53      1282                5
 8 No             0.725   0.275   Yes          24       813                1
 9 No             0.767   0.233   No           35       890                2
```

```
10 No              0.956   0.0439  No           33      1141                 1
# i 359 more rows
# i 13 more variables: HourlyRate <int>, JobLevel <int>, MonthlyIncome <int>,
#   MonthlyRate <int>, NumCompaniesWorked <int>, PercentSalaryHike <int>,
#   StockOptionLevel <int>, TotalWorkingYears <int>,
#   TrainingTimesLastYear <int>, YearsAtCompany <int>,
#   YearsInCurrentRole <int>, YearsSinceLastPromotion <int>,
#   YearsWithCurrManager <int>
```

2.4 Customize your model assessment

Creating custom assessment functions is quite convenient when iterating through various models. The `metric_set()` function from the yardstickpackage can help you to achieve this.

Define a function that returns `roc_auc`, `accuracy`, `sens(sensitivity)` and specificity `spec`(specificity) and use it to assess your model.

The augmented data frame `lr_augis` already loaded and ready to go.

Instructions

- Define a custom assessment function that returns `roc_auc`, `accuracy`, `sens`, and `spec`.
- Assess your model using your new function on `lr_augto` obtain the metrics you just chose.

```
# Define a custom assessment function
class_evaluate <- metric_set(roc_auc, accuracy, sens, spec)

# Assess your model using your new function
class_evaluate(lr_aug, truth = Attrition,
               estimate = .pred_class,
               .pred_No)
```

```
# A tibble: 4 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.832
2 sens     binary         0.984
3 spec     binary         0.05
4 roc_auc  binary         0.742
```

Common feature transformations -(video)

2.5 Common transformations

This transformation has no restrictions on the values it can take.

- Box-Cox
- **Yeo-Johnson**
- Both Yeo-Johnson and Box-Cox can take any values
- Neither. Both accept only positive values

2.6 Plain recipe

Using the `attrition_num` dataset with all numerical data about employees who have left the company, you want to build a model that can predict if an employee is likely to stay, using `Attrition`, a binary variable coded as a `factor`. To get started, you will define a plain recipe that does nothing other than define the model formula and the training data.

The `attrition_numdata`, the logistic regression `lr_model`, the user-defined `class-evaluate()` function, and the trainand test splits are loaded for you.

Instructions

- Create a plain recipe defining only the model formula.

```
# Create a plain recipe defining only the model formula
lr_recipe_plain <-
  recipe(Attrition ~., data = train)

lr_workflow_plain <- workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_plain)
lr_fit_plain <- lr_workflow_plain %>%
  fit(train)
lr_aug_plain <-
  lr_fit_plain %>% augment(test)
lr_aug_plain %>% class_evaluate(truth = Attrition,
                 estimate = .pred_class,.pred_No)
```

```
# A tibble: 4 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.832
2 sens     binary         0.984
3 spec     binary         0.05
4 roc_auc  binary         0.742
```

2.7 Box-Cox transformation

Using the `attrition_num` dataset with all numerical data about employees who have left the company, you want to build a model that can predict if an employee is likely to stay, using Attrition, a binary variable coded as a factor. To get the features to behave nearly normally, you will create a recipe that implements the Box-Cox transformation.

The `attrition_numdata`, the logistic regression `lr_model`, the user-defined `class-evaluate()` function, and the `train` and `test` splits are loaded for you.

Instructions

- Create a recipe that uses Box-Cox to transform all numeric features, including the target.

```
# Create a recipe that uses Box-Cox to transform all numeric features
lr_recipe_BC <-
  recipe(Attrition ~., data = train) %>%
 step_BoxCox(all_numeric())

lr_workflow_BC <- workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_BC)
lr_fit_BC <- lr_workflow_BC %>%
  fit(train)
lr_aug_BC <-
  lr_fit_BC %>% augment(test)
lr_aug_BC %>% class_evaluate(truth = Attrition,
                  estimate = .pred_class,.pred_No)
```

```
# A tibble: 4 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
```

```
1 accuracy binary         0.832
2 sens     binary         0.971
3 spec     binary         0.117
4 roc_auc  binary         0.741
```

2.8 Yeo-Johnson transformation

Using the `attrition_num` dataset with all numerical data about employees who have left the company, you want to build a model that can predict if an employee is likely to stay, using `Attrition`, a binary variable coded as a `factor`. To get the features to behave nearly normally, you will create a recipe that implements the Yeo-Johnson transformation.

The `attrition_numdata`, the logistic regression `lr_model`, the user-defined `class-evaluate()` function, and the `train` and `test` splits are loaded for you.

Instructions

- Create a recipe that uses Yeo-Johnson to transform all numeric features, including the target.

```
# Create a recipe that uses Yeo-Johnson to transform all numeric features
lr_recipe_YJ <-
  recipe(Attrition ~., data = train) %>%
  step_YeoJohnson(all_numeric())

lr_workflow_YJ <- workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_YJ)
lr_fit_YJ <- lr_workflow_YJ %>%
  fit(train)
lr_aug_YJ <-
  lr_fit_YJ %>% augment(test)
lr_aug_YJ %>% class_evaluate(truth = Attrition,
                  estimate = .pred_class,.pred_No)
```

```
# A tibble: 4 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.840
2 sens     binary         0.977
```

```
3 spec      binary         0.133
4 roc_auc   binary         0.746
```

Advanced transformations -(video)

2.9 Baseline

Continuing with the `attrition_num` dataset, you will create a baseline with a plain recipe to assess the effects of additional feature engineering steps. The `attrition_num` data, the logistic regression `lr_model`, the user-defined `class-evaluate()` function, and the `train` and `test` splits have already been loaded for you.

Instructions

- Bundle the model and recipe into a workflow.
- Augment the fitted workflow to get it ready for assessment.

```
lr_recipe_plain <- recipe(Attrition ~., data = train)

# Bundle the model and recipe
lr_workflow_plain <- workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_plain)
lr_fit_plain <- lr_workflow_plain %>%
  fit(train)

# Augment the fit workflow
lr_aug_plain <- lr_fit_plain %>%
  augment(test)
lr_aug_plain %>%
  class_evaluate(truth = Attrition,estimate = .pred_class,
                 .pred_No)
```

```
# A tibble: 4 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.832
2 sens     binary         0.984
```

```
3 spec     binary         0.05
4 roc_auc  binary         0.742
```

2.10 `step_poly()`

Now that you have a baseline, you can compare your model's performance if you add a polynomial transformation to all numerical values.

The `attrition_num` data, the logistic regression `lr_model`, the user-defined `class-evaluate()` function, and the `train` and `test` splits have already been loaded for you.

Instructions

- Add a polynomial transformation to all numeric predictors.
- Fit workflow to the `train` data.

```
lr_recipe_poly <-
  recipe(Attrition ~., data = train) %>%

# Add a polynomial transformation to all numeric predictors
  step_poly(all_numeric_predictors())

lr_workflow_poly <- workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_poly)

# Fit workflow to the train data
lr_fit_poly <- lr_workflow_poly %>% fit(train)
lr_aug_poly <- lr_fit_poly %>% augment(test)
lr_aug_poly %>% class_evaluate(truth = Attrition, estimate = .pred_class,.pred_No)
```

```
# A tibble: 4 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.829
2 sens     binary         0.968
3 spec     binary         0.117
4 roc_auc  binary         0.763
```

2.11 `step_percentile()`

How would applying a percentile transformation to your numeric variables affect model performance? Try it!

The `attrition_num` data, the logistic regression `lr_model`, the user-defined `class-evaluate()` function, and the `train` and `test` splits have already been loaded for you.

Instructions

- Apply a percentile transformation to all numeric predictors.

```
# Add percentile tansformation to all numeric predictors
lr_recipe_perc <-
  recipe(Attrition ~., data = train) %>%
  step_percentile(all_numeric_predictors())
lr_workflow_perc <-
  workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_perc)
lr_fit_perc <- lr_workflow_perc %>% fit(train)
lr_aug_perc <- lr_fit_perc %>% augment(test)
lr_aug_perc %>% class_evaluate(truth = Attrition,
                  estimate = .pred_class,.pred_No)
```

```
# A tibble: 4 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.839
2 sens     binary         0.984
3 spec     binary         0.1
4 roc_auc  binary         0.750
```

2.12 Who's staying?

It's time to practice combining several transformations to the `attrition_num` data. First, normalize or near-normalize numeric variables by applying a Yeo-Johnson transformation. Next, transform numeric predictors to percentiles, create dummy variables, and eliminate features with near zero variance.

Instructions

- Apply a Yeo-Johnson transformation to all numeric variables.
- Transform all numeric predictors into percentiles.
- Create dummy variables for all nominal predictors.

```
lr_recipe <- recipe(Attrition ~., data = train) %>%

# Apply a Yeo-Johnson transformation to all numeric variables
  step_YeoJohnson(all_numeric()) %>%

# Transform all numeric predictors into percentiles
 step_percentile(all_numeric_predictors()) %>%

# Create dummy variables for all nominal predictors
  step_dummy(all_nominal_predictors())

lr_workflow <- workflow() %>% add_model(lr_model) %>% add_recipe(lr_recipe)
lr_fit <- lr_workflow %>% fit(train)
lr_aug <- lr_fit %>% augment(test)
lr_aug %>% class_evaluate(truth = Attrition, estimate = .pred_class,.pred_No)
```

```
# A tibble: 4 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.842
2 sens     binary         0.987
3 spec     binary         0.1
4 roc_auc  binary         0.750
```

> **Note**
>
> ALLLLLL DONE

3 references