What is feature engineering?

FEATURE ENGINEERING IN R

Jorge Zazueta

Research Professor and Head of the Modeling Group at the School of Economics, UASLP

What is feature engineering?

Feature engineering is the art and science of

- creating,
- transforming,
- extracting, and
- selecting

variables to improve model performance and interpretability.

Height of an object as a function of time

```
# A tibble: 100 × 2
    time height
   <dbl>  <dbl>
 1 0        0
 2 0.101    3.85
 3 0.202   17.7
 4 0.303   15.1
 5 0.404   20.0
 6 0.505   32.6
 7 0.606   30.8
 8 0.707   26.6
```

Why engineer features?

We create a simple regression model of height

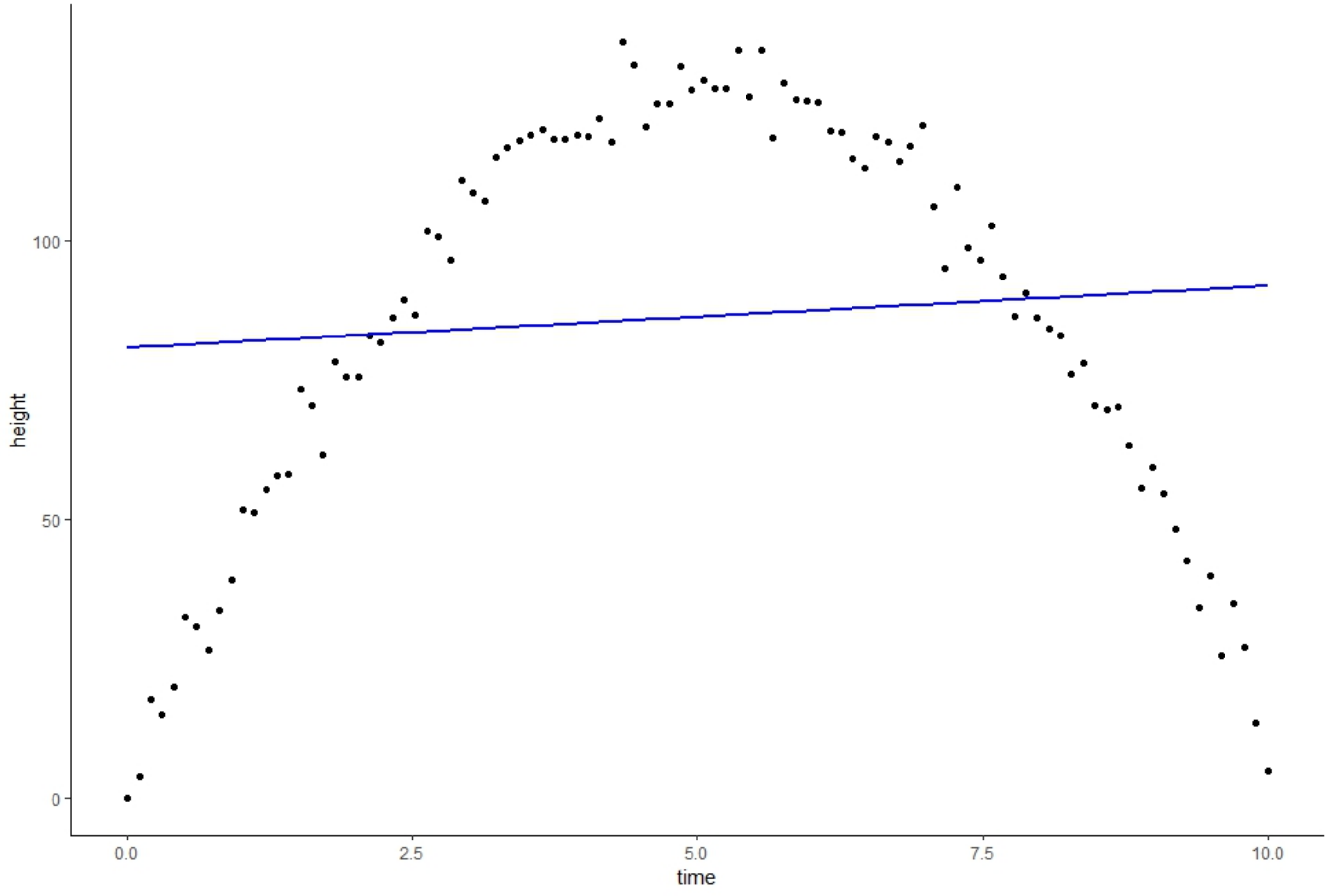
```
lr_height <- lm(height ~ time,
                data = height)
```

and graph it to assess accuracy by sight.

```
df <- height %>%
bind_cols(lr_pred = predict(lr_height))

df %>%
  ggplot(aes(x = time, y = height)) +
  geom_point() +
  geom_line(aes(y = lr_pred),
            color = "blue", lwd = .75)+
    theme_classic()
```

Our model fails miserably to represent the data!

Linear regression of height *vs.* time.



Using mutate()

The height of an object follows a parabolic path given by the following formula:

$$y(t) = y_0 + v_0 t - \frac{g}{2}t^2.$$

Where y represents the height of the object at time t, and y_0, v_0, and g are, respectively, the initial height, velocity, and acceleration due to gravity.

We can fit our model, recognizing the dependence of height on both time and the square of time.

`mutate()` takes a data frame as a first argument and the definition of a new variable to be added to the data frame.

```
df_2 <- df %>% mutate(time_2 = time^2)
```

```
# A tibble: 100 × 4
   time height lr_pred time_2
  <dbl>  <dbl>   <dbl>  <dbl>
1 0        0      80.8 0
2 0.101    3.85   80.9 0.0102
3 0.202   17.7    81.0 0.0408
4 0.303   15.1    81.1 0.0918
```

Predict using the engineered feature

We create another regression model, using our new feature along with the original one.

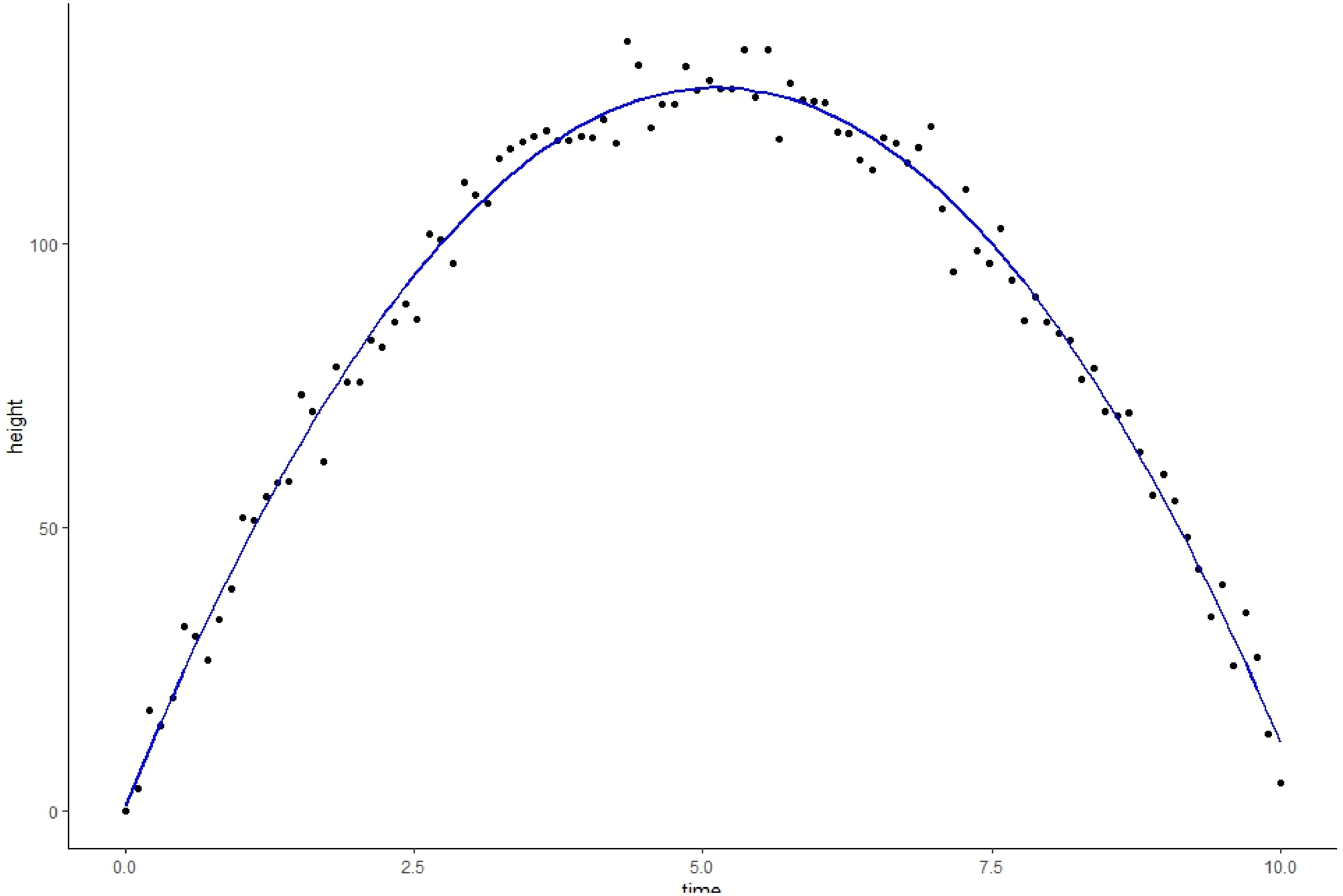
```
lr_height_2 <-
lm(height ~ time + time_2, data = df_2)
```

And graph our new prediction.

```
df_2 <- df_2 %>%
    bind_cols(lr2_pred = predict(lr_height_2))
df_2 %>%
  ggplot(aes(x = time, y = height)) +
  geom_point() +
  geom_line(aes(y = lr2_pred),
      col = "blue", lwd = .75) +
  theme_classic()
```

That is an impressive improvement without resorting to a different model.

Height vs. time and time_2



Let's practice!

FEATURE ENGINEERING IN R

Creating new features using domain knowledge

FEATURE ENGINEERING IN R

Jorge Zazueta

Research Professor and Head of the Modeling Group at the School of Economics, UASLP

The importance of domain knowledge

Domain knowledge enables us to identify and create relevant and useful features for a particular model or task.

Feature engineering is about creating new input features from existing ones.

Examples of domain knowledge:

- **Financial:** The critical determinants of bankruptcy
- **Medical:** Pre-existing conditions relevant to a specific treatment
- **Marketing:** Distinguishing features of a consumer group

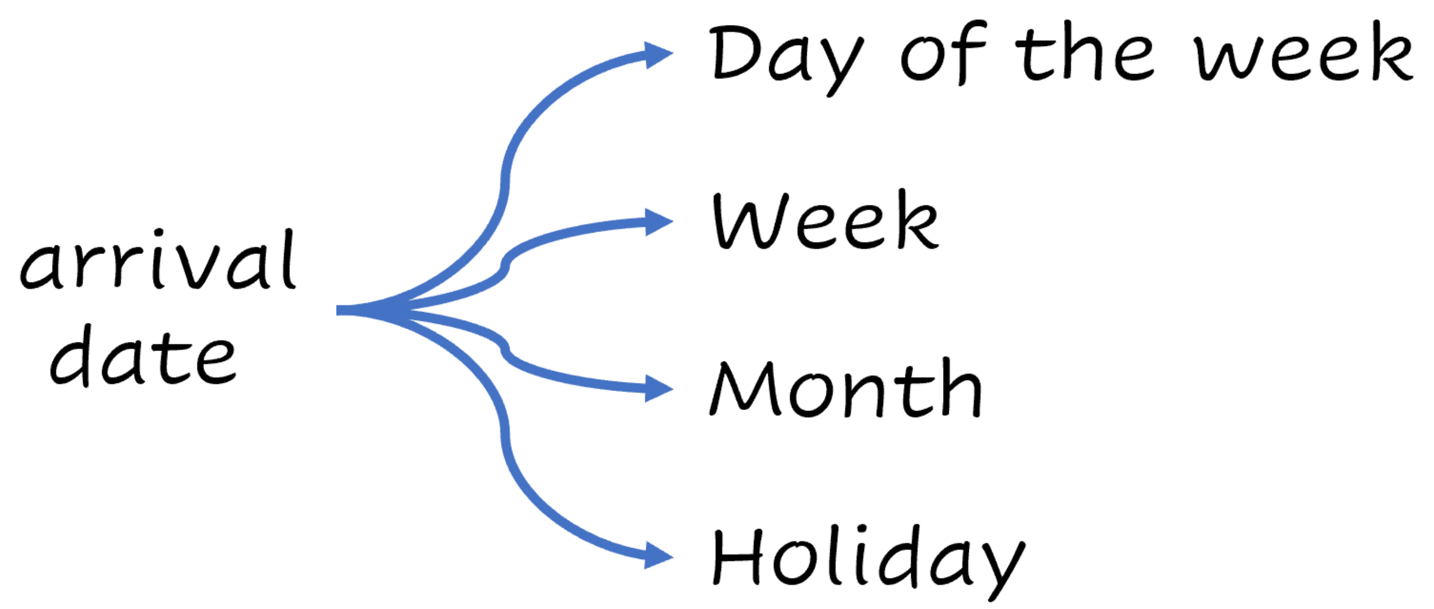
Creating variables based on professional experience

We want to predict hotel cancellations based on the following feature vector:

```r
features <-
c("IsCanceled", "LeadTime",
  "arrival_date",
  "StaysInWeekendNights",
  "StaysInWeekNights",
  "PreviousCancellations",
  "PreviousBookingsNotCanceled",
  "ReservedRoomType",
  "AssignedRoomType","BookingChanges",
  "DepositType","CustomerType",
  "ADR","TotalOfSpecialRequests")
```

Features form raw data

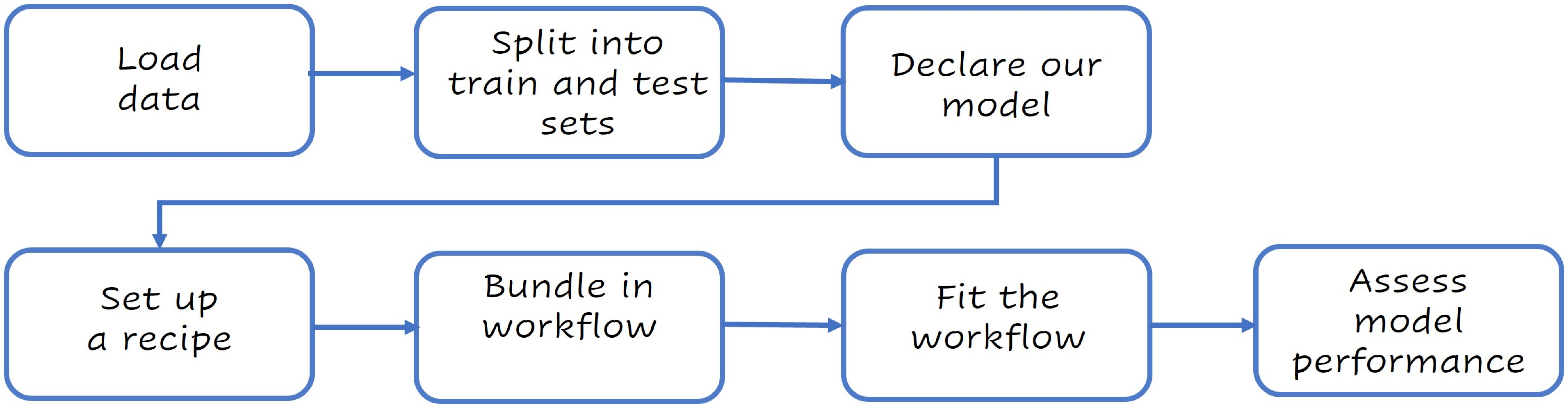
We can generate informative features from `arrival_date`.



But this becomes tedious quickly. We need to automate it!

The tidymodels framework

We'll use a workflow based on `tidymodels` , a collection of packages for modeling and machine learning using `tidyverse` principles (1) with emphasis on feature engineering.



We can learn more at **www.tidymodels.org**

[1] [Tidyverse guiding principles.](https://design.tidyverse.org/unifying-principles.html)

Setting up our data for analysis

Let's start by getting our data ready.

```
cancelations <-
  cancelations %>%
  mutate(across(where(is_character),as.factor))
```

```
set.seed(123)
split <- cancellations %>%
    initial_split(
    strata = "IsCanceled")
train <- training(split)
test <- testing(split)
```

The prop parameter can be used to change the train/test data split (the default is 3/4).

```
initial_split(data, prop = 3/4, strata = NULL)
```

Verify that `train` and `test` sets exhibit similar proportions of canceled reservations.

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

IsCanceled
        0         1
0.5826946 0.4173054
```

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

IsCanceled
        0         1
0.5827788 0.4172212
```

Building a workflow

Declare our model

```
lr_model <- logistic_reg()
```

Build a recipe

```
lr_recipe <-
  recipe(IsCanceled ~., data = train) %>%
  update_role(Agent, new_role = "ID" ) %>%
  step_date(arrival_date,
      features = c("dow", "week", "month")) %>%
  step_holiday(arrival_date,
      holidays = timeDate::listHolidays("US")) %>%
  step_rm(arrival_date) %>%
  step_dummy(all_nominal_predictors())
```

Print `lr_recipe`

```
Recipe
Inputs:

      role #variables
        ID          1
   outcome          1
 predictor         13

Operations:

Date features from arrival_date
Holiday features from arrival_date
Variables removed arrival_date
Dummy variables from all_nominal_predictors()
```

Building a workflow

Bundle the model and the recipe into a `workflow` object.

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

Fit the workflow

```
lr_fit <-
  lr_workflow %>%
  fit(data = train)
```

Building a workflow

We can use `tidy(lr_fit)` to summarize our model.

```
# A tibble: 65 × 5
   term                        estimate std.error statistic   p.value
   <chr>                          <dbl>     <dbl>     <dbl>     <dbl>
 1 (Intercept)                 -1.92     0.228        -8.43 3.57e- 17
 2 LeadTime                     0.00414  0.000268     15.4  1.16e- 53
 3 StaysInWeekendNights         0.0860   0.0382        2.25 2.45e-  2
 4 StaysInWeekNights            0.0804   0.0185        4.34 1.40e-  5
 5 PreviousCancellations        2.39     0.147        16.2  2.45e- 59
 6 PreviousBookingsNotCanceled -0.440    0.0450       -9.77 1.45e- 22
 7 BookingChanges              -0.449    0.0463       -9.69 3.18e- 22
 8 ADR                          0.0104   0.000782     13.2  4.85e- 40
 9 TotalOfSpecialRequests      -0.727    0.0316      -23.0  5.29e-117
10 arrival_date_week            0.0245   0.0171        1.43 1.53e-  1
# … with 55 more rows
```

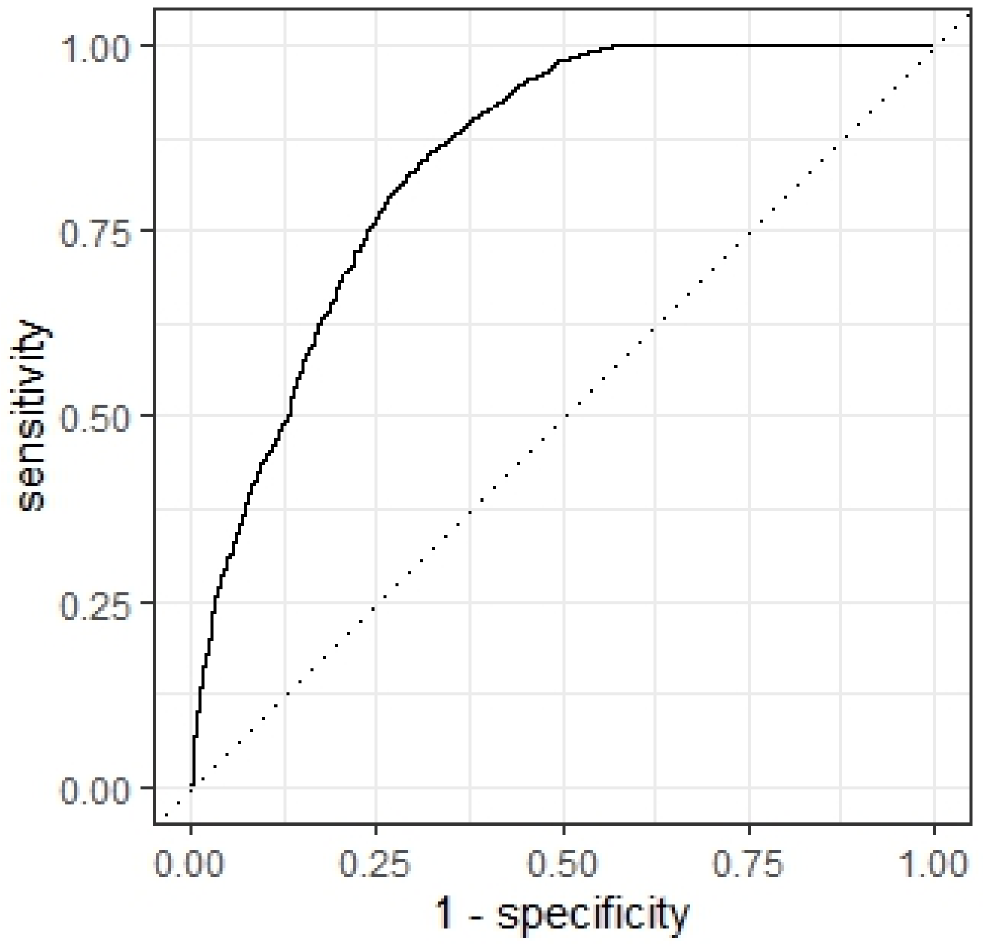
Assessing model performance

We can now assess our model's performance.

```
lr_aug <- lr_fit %>% augment(test)

bind_rows(
  lr_aug %>%
  roc_auc(truth = IsCanceled,.pred_0),
  lr_aug %>%
  accuracy(truth = IsCanceled,.pred_class))
```

```
# A tibble: 2 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 roc_auc  binary         0.842
2 accuracy binary         0.782
```

```
lr_aug %>%
  roc_curve(truth = IsCanceled, .pred_0) %>%
  autoplot()
```



Let's practice!

FEATURE ENGINEERING IN R

Increasing the information content of raw data

FEATURE ENGINEERING IN R

Jorge Zazueta
Research Professor and Head of the Modeling Group at the School of Economics, UASLP

Dealing with raw data

A typical dataset with missing values

Col_1	Col_2	...	Col_n
Data	Data	Data	NA
NA	Data	Data	Data
Data	NA	Data	Data
Data	NA	Data	Data
Data	Data	Data	NA

Values as factors

Col_1
Factor_1
Factor_2
Factor_4
Factor_3
Factor_2

Dealing with raw data

Dataset with imputed values

Col_1	Col_2	...	Col_n
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

Factors represented as dummy variables

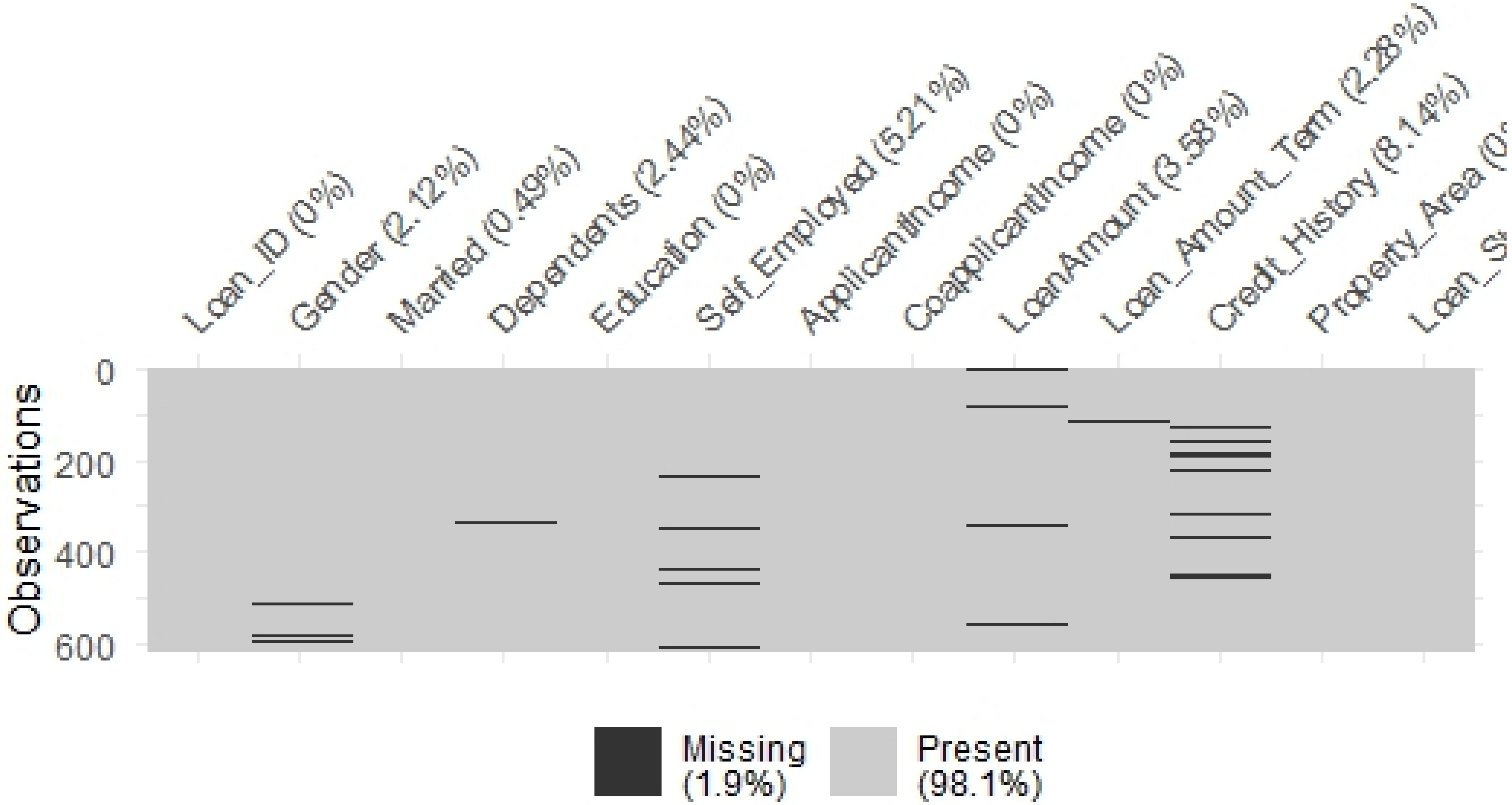
Factor_2	Factor_3	Factor_4
0	0	0
1	0	0
0	0	1
0	1	0
1	0	0

The loans dataset

```
# A tibble: 614 × 13
   Loan_ID  Gender Married Dependents Educa…¹ Self_…² Appli…³ Coapp…? LoanA…? Loan_…?
   <fct>    <fct>  <fct>   <fct>      <fct>   <fct>     <dbl>   <dbl>   <dbl>   <dbl>
 1 LP001002 Male   No      0          Gradua… No         5849       0      NA     360
 2 LP001003 Male   Yes     1          Gradua… No         4583    1508     128     360
 3 LP001005 Male   Yes     0          Gradua… Yes        3000       0      66     360
 4 LP001006 Male   Yes     0          Not Gr… No         2583    2358     120     360
 5 LP001008 Male   No      0          Gradua… No         6000       0     141     360
 6 LP001011 Male   Yes     2          Gradua… Yes        5417    4196     267     360
 7 LP001013 Male   Yes     0          Not Gr… No         2333    1516      95     360
 8 LP001014 Male   Yes     3+         Gradua… No         3036    2504     158     360
 9 LP001018 Male   Yes     2          Gradua… No         4006    1526     168     360
10 LP001020 Male   Yes     1          Gradua… No        12841   10968     349     360
# … with 604 more rows, 3 more variables: Credit_History <dbl>, Property_Area <fct>,
#   Loan_Status <fct>, and abbreviated variable names ¹?Education, ²?Self_Employed,
#   ³?ApplicantIncome, ??CoapplicantIncome, ??LoanAmount, ??Loan_Amount_Term
# ? Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```

Missing values

We can visually identify missing values in `loans` using `vis_miss(loans)` from the package `naniar`.

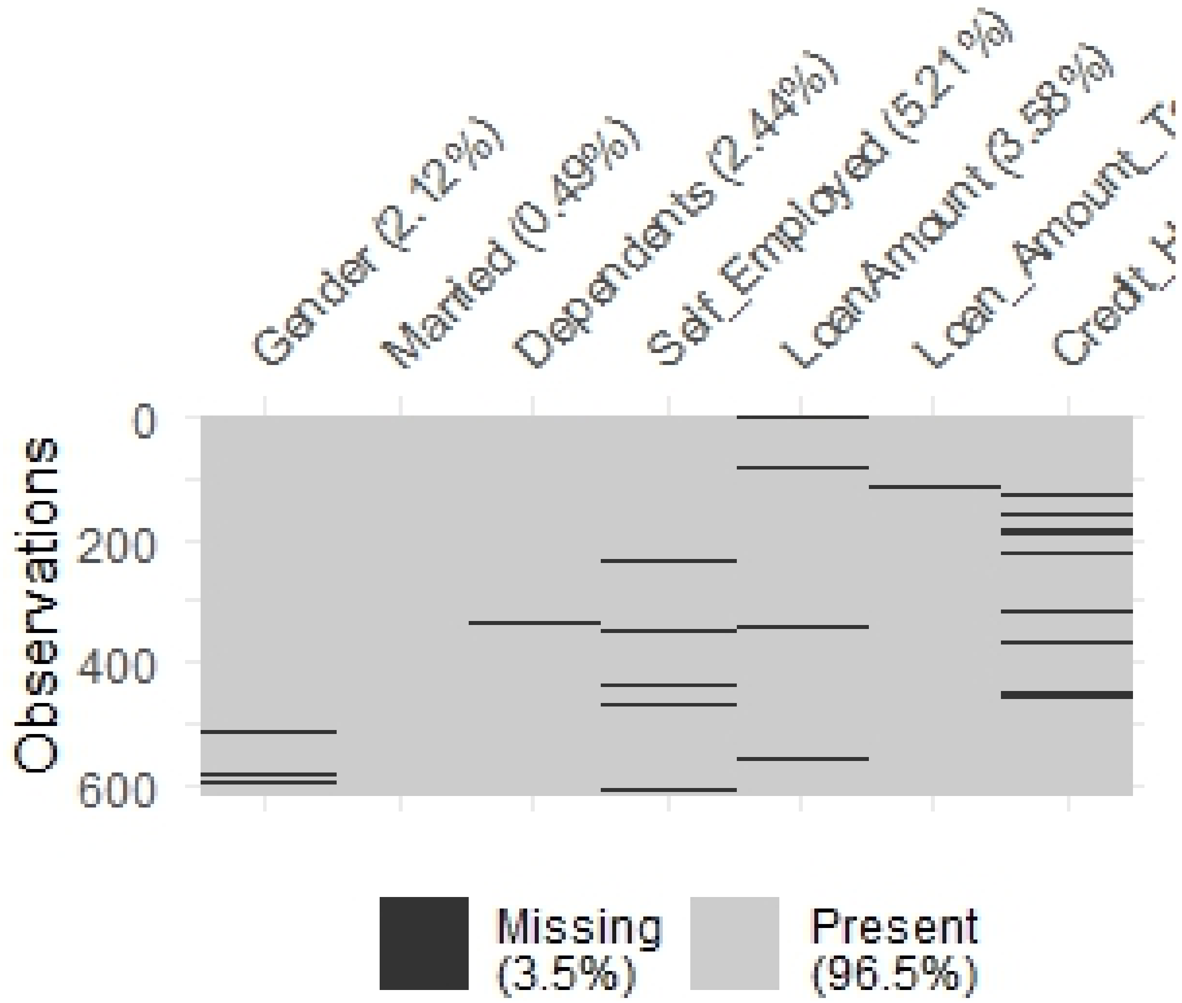


Missing values

We can zoom the table by selecting only the columns with missing values.

```
loans %>%
select(Gender,
       Married,
       Dependents,
       Self_Employed,
       LoanAmount,
       Loan_Amount_Term,
       Credit_History) %>%
  vis_miss()
```

A closer view of missing values



Missing values and dummy variables

We can address missing values and create dummy variables in the same recipe.

```
lr_recipe <-
  recipe(Loan_Status ~.,
         data = train) %>%
  update_role(Loan_ID,
              new_role = "ID" ) %>%
  step_impute_knn(all_predictors()) %>%
  step_dummy(all_nominal_predictors())
```

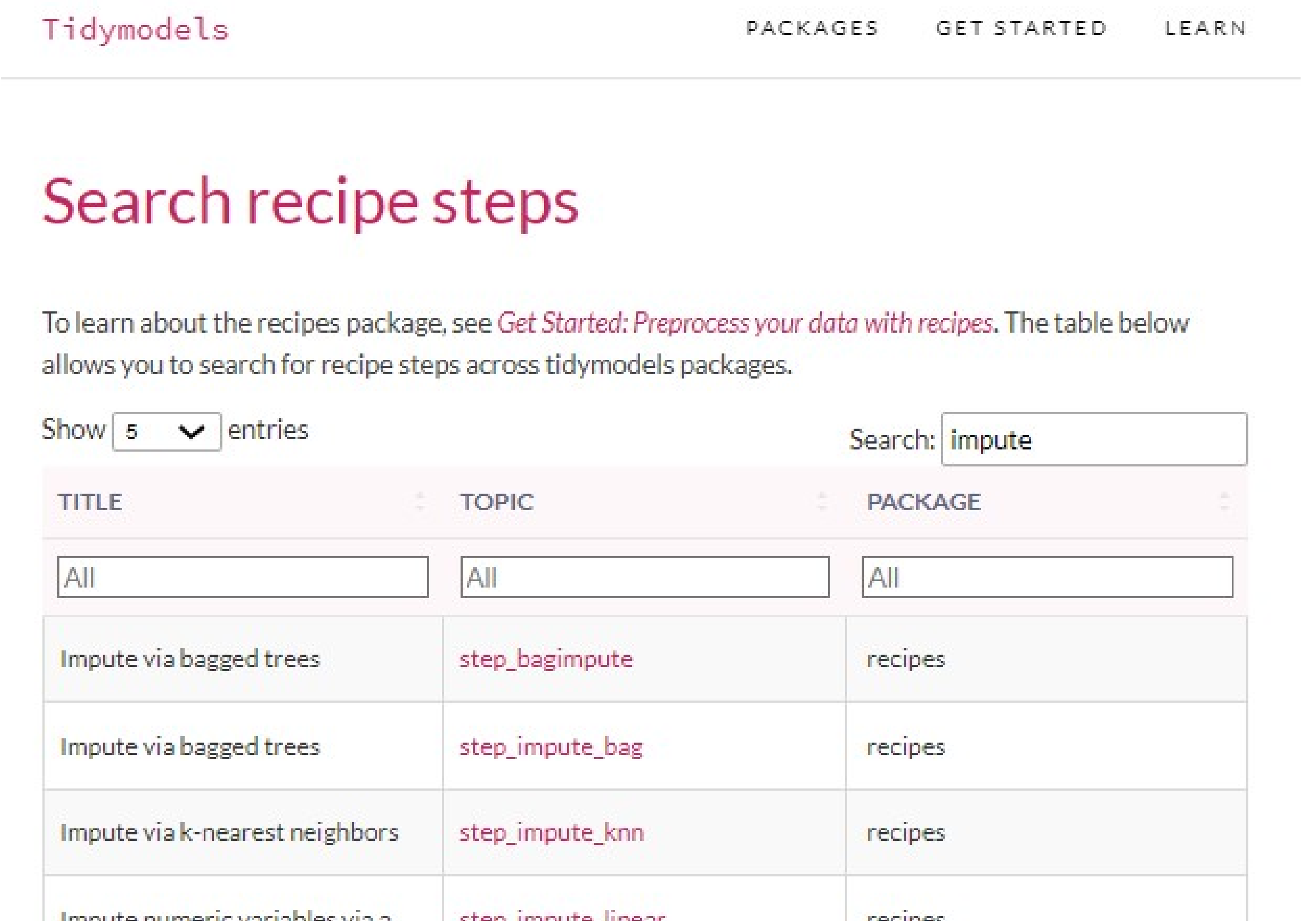
Print the recipe

```
lr_recipe
```

```
Recipe

Inputs:

      role #variables
        ID          1
   outcome          1
 predictor         30

Operations:

K-nearest neighbor imputation for all_predictors()
Dummy variables from all_nominal_predictors()
```

Finding the right recipe step

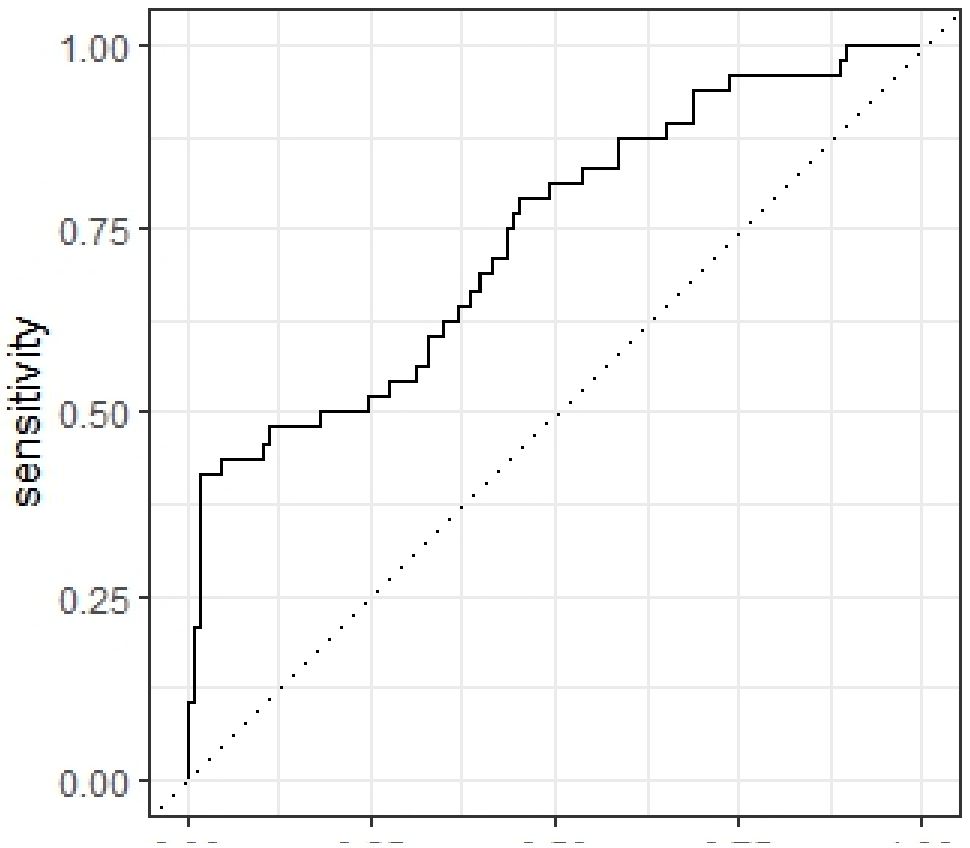
We can find other imputation methods and all recipe steps in the `tidymodels` documentations at **www.tidymodels.org/find/recipes**



Fitting and assessing our model

```
# Fit
lr_fit <-
  lr_workflow %>% fit(data = train)
lr_aug <-
  lr_fit %>% augment(test)
# Assess
lr_aug %>%
  roc_curve(truth = Loan_Status, .pred_N) %>%
  autoplot()

bind_rows(lr_aug %>%
            roc_auc(truth = Loan_Status,
                    .pred_N),
          lr_aug %>%
            accuracy(truth = Loan_Status,
                     .pred_class))
```

```
# A tibble: 2 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 roc_auc  binary         0.738
2 accuracy binary         0.792
```



Let's practice!

FEATURE ENGINEERING IN R