Reducing dimensionality

FEATURE ENGINEERING IN R

Jorge Zazueta

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

datacamp

Zero variance features

Some datasets include columns with constant values or zero variance. We can filter out those features by adding `step_zv()` to our `recipe()`.

Col_1	Col_2	...	Col_n
0.9099	0.9738	0.2959	0.8945
0.1757	0.9738	0.0519	0.9337
0.8688	0.9738	0.8156	0.4716
0.0136	0.9738	0.1120	0.8219
0.3765	0.9738	0.3083	0.0309

Near-zero variance features

Near-zero variance features include predictors with a single value **and** predictors with both of the following characteristics:

- Very few unique values relative to the number of samples
- The ratio of the frequency of the most common value to the frequency of the second most common value is large

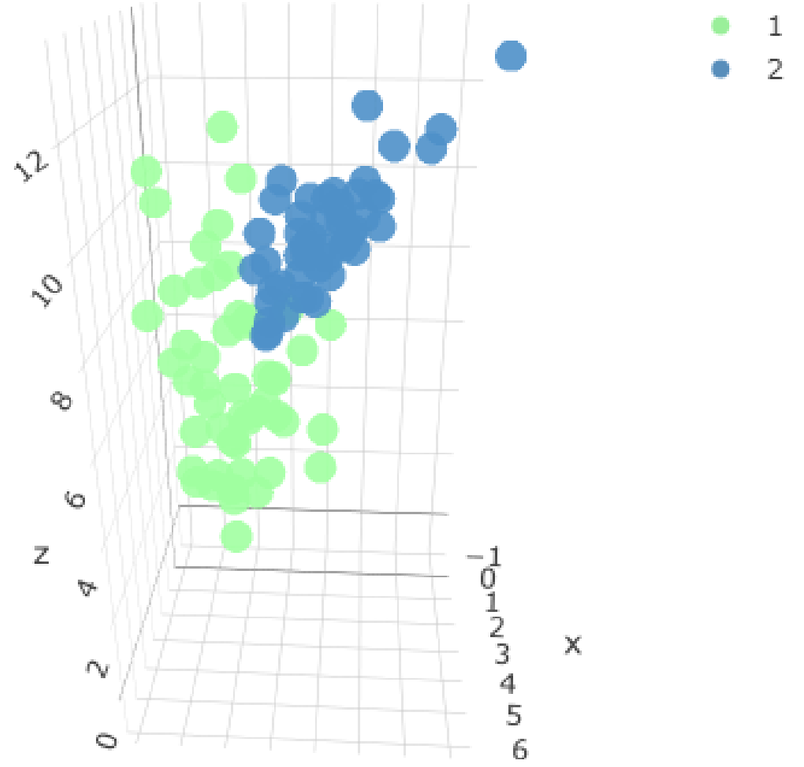
Example of near-zero variance:

- For 100 observations there are two different values but one occurs only once.

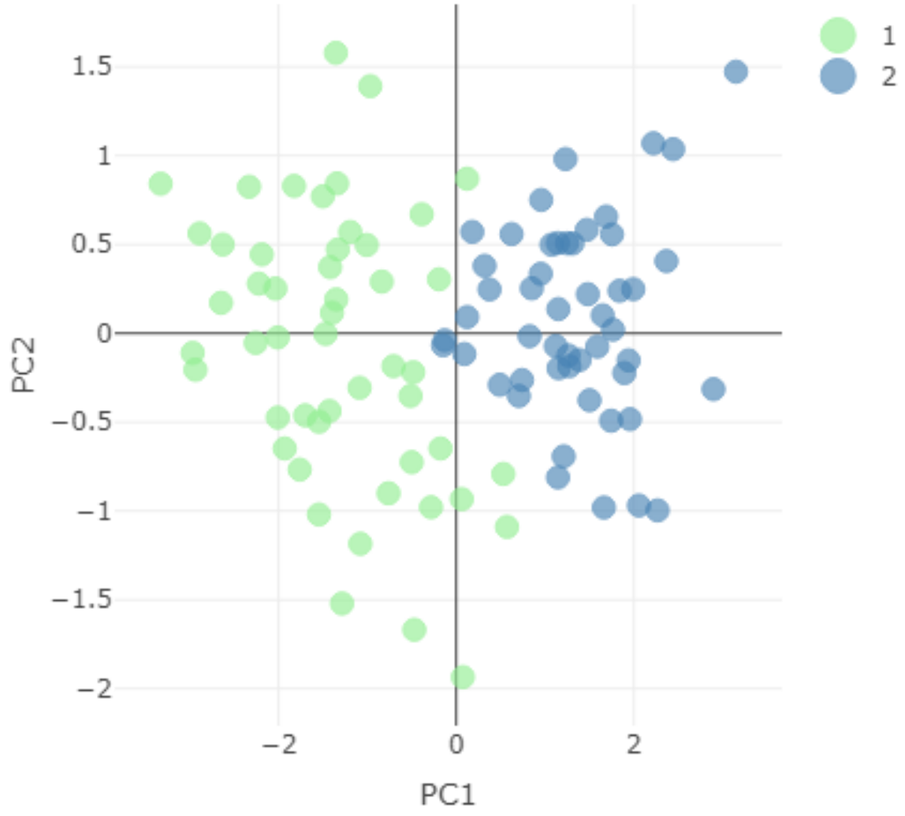
`step_nzv()` identifies and removes predictors with these characteristics.

Principal Component Analysis (PCA)

Original three-dimensional dataset with two associated classes.



Reduced dataset representing data using first two principal components.



Let's prep a recipe

Creating a recipe to perform PCA and retrieving its output via `prep()`.

```
pc_recipe <-
recipe(~., data = loans_num) %>%
  step_nzv(all_numeric()) %>%
  step_normalize(all_numeric()) %>%
  step_pca(all_numeric())
```

```
pca_output <- prep(pc_recipe)
```

We can look at the information available by calling `names()` on `pca_output`.

```
names(pca_output)
```

```
 [1] "var_info"        "term_info"
 [3] "steps"           "template"
 [5] "levels"          "retained"
 [7] "requirements"    "tr_info"
 [9] "orig_lvls"       "last_term_info"
```

Unearthing variance explained

Extract standard deviation from the pca_output object and compute variance explained.

```
stdv <- pca_output$steps[[3]]$res$sdev
```

```
var_explained <- stdv^2/sum(stdv^2)
```

```
PCA = tibble(PC = 1:length(stdv),
             var_explained = var_explained,
             cumulative = cumsum(var_explained))
```

A table showing variance explained by principal component.

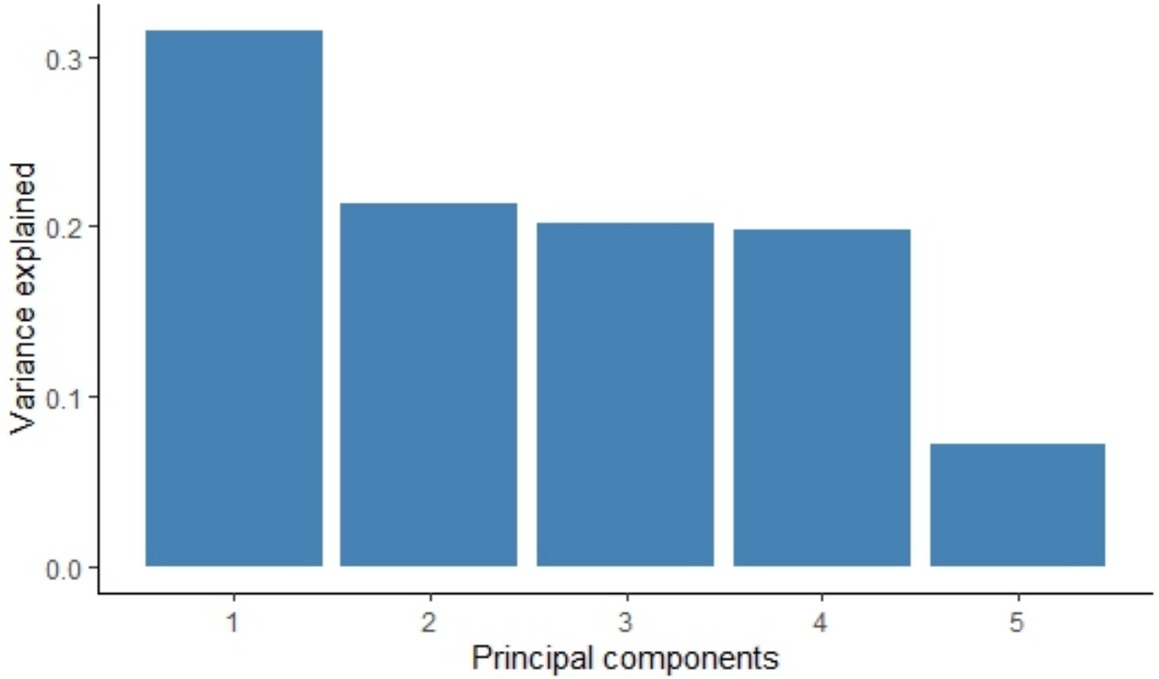
```
# A tibble: 5 × 3
     PC var_explained cumulative
  <int>          <dbl>      <dbl>
1     1         0.315       0.315
2     2         0.214       0.529
3     3         0.202       0.730
4     4         0.198       0.928
5     5         0.0722      1
```

Visualizing variance explained

We can graph the output as a column chart using `ggplot2`.

```
PCA %>%
ggplot(aes(x = PC,
           y = var_explained)) +
  geom_col(fill = "steelblue") +
  xlab("Principal components") +
  ylab("Variance explained")
```

Variance explained by principal component.



Let's practice!

FEATURE ENGINEERING IN R

Feature hashing

FEATURE ENGINEERING IN R

Jorge Zazueta

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

What is feature hashing?

- Transforms a text variable into a set of numerical variables
- Uses *hash values* as feature indices
- Low memory representation of the data
- Helpful when we expect new categories when new data is seen

Assign an index number to each carrier based on text values.

carrier		dummy_hash
UA	->	30
WN	->	32
DL	->	27
EV	->	44
B6	->	18
AA	->	26

How many carriers are there?

The `flights` dataset includes carriers as factors, but we don't know if new carriers will appear when we look at new data.

```
flights %>%
  select(carrier) %>%
  table()
```

```
carrier
  9E   AA   AS   B6   DL   EV   F9   FL   HA   MQ   OO   UA   US   VX   WN   YV
 859 1744   26 2503 2619 3014   38  186   14 1540    2 3367 1228  244  757   41
```

Let us hash that feature

We can assign create dummy hashes to represent the factor values. Using the `textrecipes` package.

```
recipe <- recipe(~carrier,
                 data = flights_train) %>%
  step_dummy_hash(carrier, prefix = NULL,
                  signed = FALSE,
                  num_terms = 50L)
# Prep the recipe
object <- recipe %>%
  prep()

# Bake the recipe object with new data
baked <- bake(object,
              new_data = flights_test)
```

A peak at the `step_dummy_hash()` representation.

```
bind_cols(flights_test$carrier,baked)[1:6,c(1,18:20)]
```

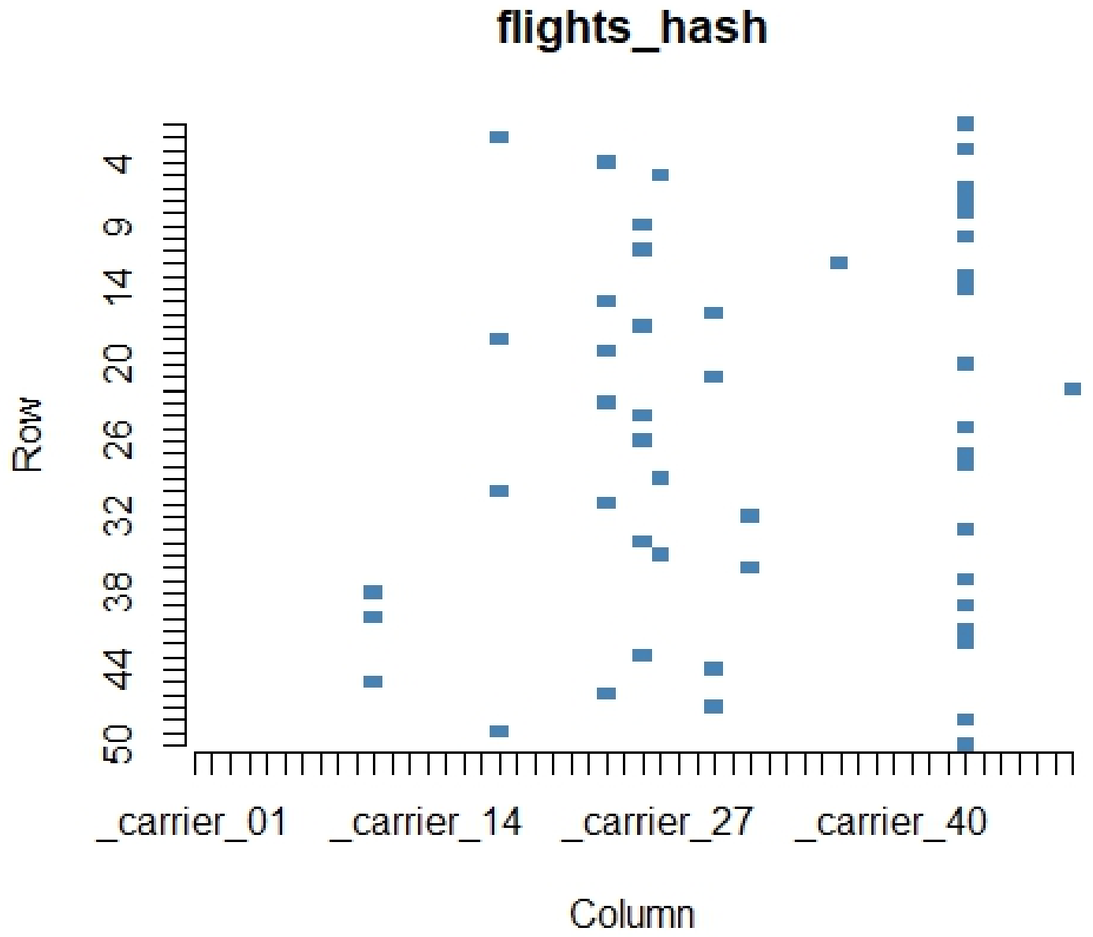
```
New names:
• `` -> `...1`
# A tibble: 10 × 4
   ...1  `_carrier_17` `_carrier_18` `_carrier_19`
   <chr>         <int>         <int>         <int>
 1 EV                0             0             0
 2 B6                0             1             0
 3 EV                0             0             0
 4 MQ                0             0             0
 5 DL                0             0             0
 6 EV                0             0             0
```

Visualizing the hashing

We can take a look at the matrix with the help of the `plot.matrix` package.

```
flights_hash <-
    as.matrix(baked)[1:50,]

plot(flights_hash,
     col = c("white","steelblue"),
     key = NULL,
     border = NA)
```



Let's practice!

FEATURE ENGINEERING IN R

Encoding categorical data using supervised learning

FEATURE ENGINEERING IN R

Jorge Zazueta

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

datacamp

Introducing supervised encoding

Supervised encoding, in contrast, uses the outcome values to derive numeric features from nominal predictors.

Introducing supervised encoding

Supervised encoding uses the outcome values to derive numeric features from nominal predictors.

Some supervised encoding functions available in the `embed` package

Function	Definition
step_lencode_glm()	Uses likelihood encodings to convert a nominal predictor into a single set of scores derived from a generalized linear model.
step_lencode_bayes()	Applies Bayesian likelihood encodings to convert a nominal predictor into a single set of scores derived from a generalized linear model estimated using Bayesian analysis.
step_lencode_mixed()	Converts nominal predictors into a single set of scores derived from a generalized linear mixed model.

Predicting grant application success

We are interested in predicting grant application success based solely on sponsor code.

```
lr_model <- logistic_reg() # declare model

lr_recipe_glm <- # Set recipe glm
  recipe(class ~ sponsor_code,
         data = grants_train) %>%
  step_lencode_glm(sponsor_code,
                   # Declare outcome variable
                   outcome = vars(class))

lr_workflow_glm <- # Create Workflow
  workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_glm)
```

Workflow summary

```
lr_workflow_glm
```

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

-- Preprocessor --------------------------------
1 Recipe Step

- step_lencode_glm()

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

Computational engine: glm
```

Fitting, augmenting, and assessing

We fit and evaluate our model

```
lr_fit_glm <- # Fit
  lr_workflow_glm %>%
  fit(grants_train)

lr_aug_glm <- # Augment
  lr_fit_glm %>%
  augment(grants_test)

glm_model <- lr_aug_glm %>% # Assess
  class_evaluate(truth = class,
                 estimate = .pred_class,
                 .pred_successful)
```

Performance results are stored in `glm_model`

```
glm_model
```

```
# A tibble: 2 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.728
2 roc_auc  binary         0.684
```

Binding models together

We build `bayes_model` and `mixed_model` to compare the performance of the corresponding steps.

```
# Define model names
model <- c("glm", "glm",
           "bayes","bayes",
           "mixed", "mixed")
# Bind models in a tibble
models <-
bind_rows(glm_model,
          bayes_model,
          mixed_model)%>%
  add_column(model = model)%>%
  select(-.estimator) %>%
  spread(model,.estimate)
```

A convenient performance table

```
models
```

```
# A tibble: 2 × 4
  .metric  bayes   glm mixed
  <chr>    <dbl> <dbl> <dbl>
1 accuracy 0.718 0.728 0.720
2 roc_auc  0.686 0.684 0.682
```

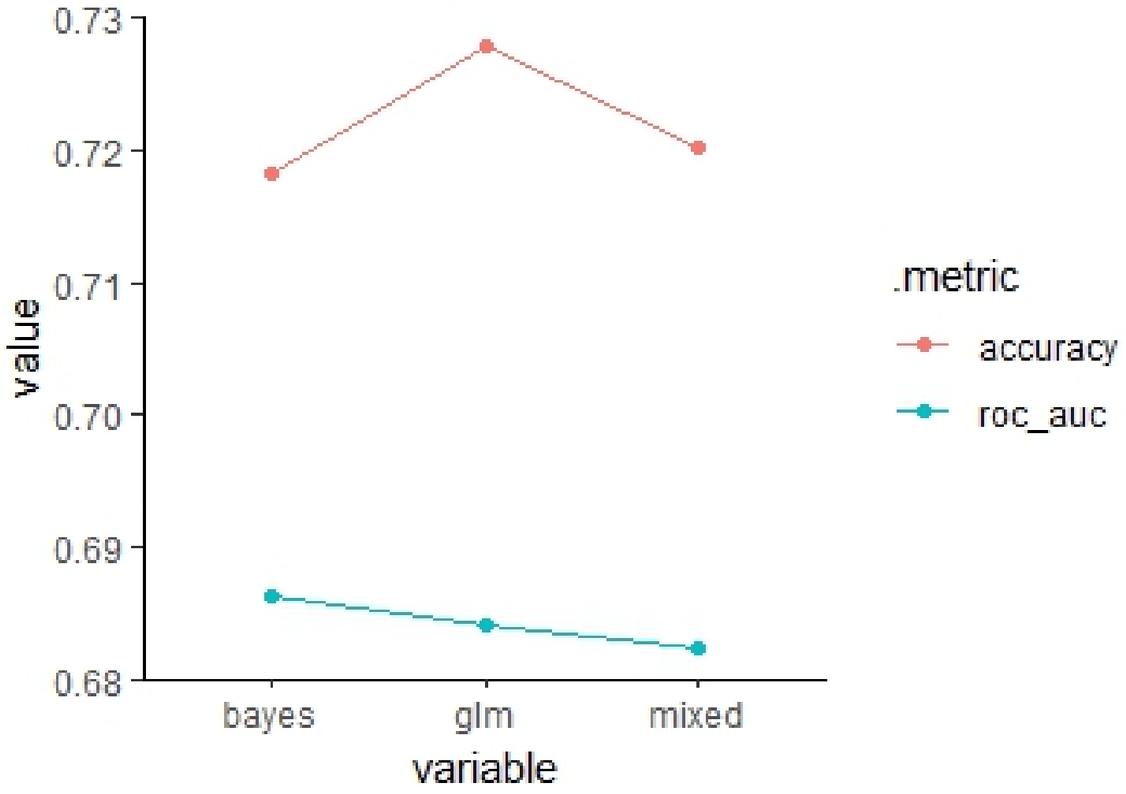
Visualizing our results

Visualize results in a parallel coordinates chart from the `Gally` package.

```
# Libraries
library(GGally)

# Parallel coordinates chart
ggparcoord(models,
           columns = 2:4,
           groupColumn = 1,
           scale="globalminmax",
           showPoints = TRUE)
```

Parallel coordinates chart of accuracy and roc_auc comparing all models.



Let's practice!

FEATURE ENGINEERING IN R

Variable Importance

FEATURE ENGINEERING IN R

Jorge Zazueta

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

Adding more predictors

A more complete model includes many variables.

```
lr_model <- logistic_reg()
lr_recipe <-
  recipe(class~ sponsor_code +
         contract_value_band +
         category_code,
         data = grants_train) %>%
  step_lencode_glm(sponsor_code,
                   contract_value_band,
                   category_code,
                   outcome = vars(class))
```

With more appealable results.

```
lr_aug %>% class_evaluate(truth = class,
                estimate = .pred_class,
                .pred_successful)
```

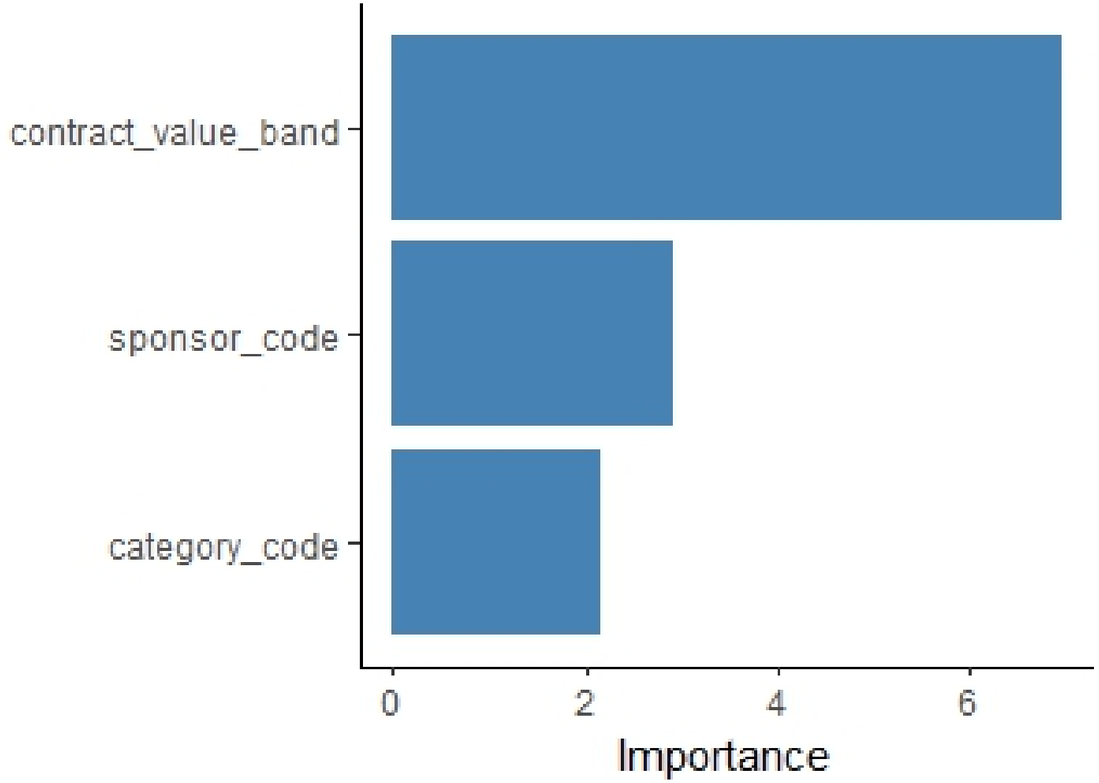
```
# A tibble: 2 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.890
2 roc_auc  binary         0.951
```

Which variables matter most?

We can plot features ranked by importance with help from the `vip()` package.

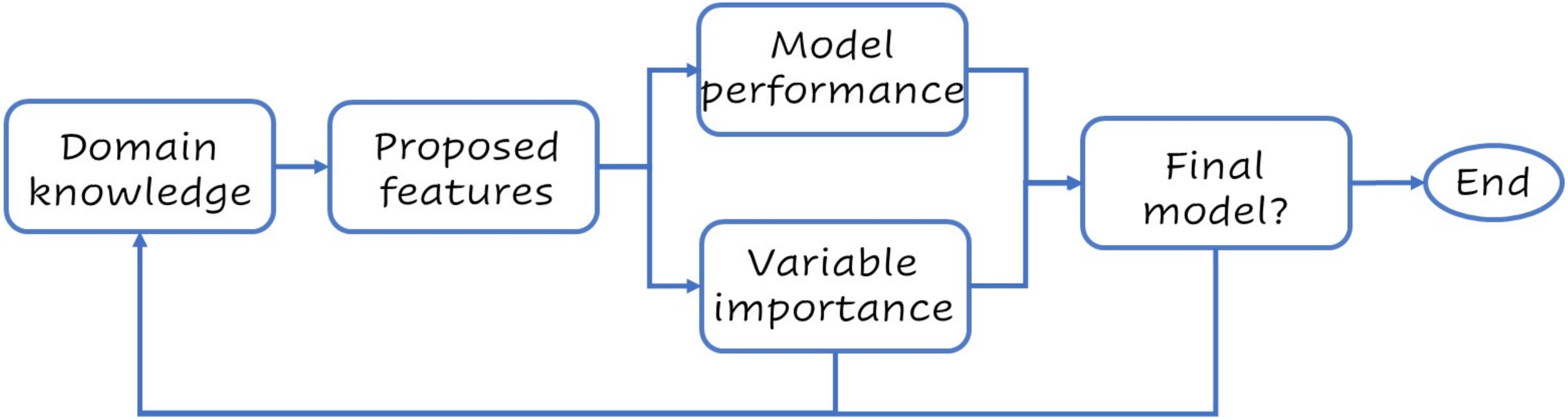
```
lr_fit %>%
  extract_fit_parsnip() %>%
  vip(aesthetics =
      list(fill = "steelblue"))
```

Variable importance chart



Variable importance and feature engineering

Variable importance can be a powerful feedback mechanism for refining feature engineering based on domain knowledge.



Let's practice!

FEATURE ENGINEERING IN R