Feature engineering

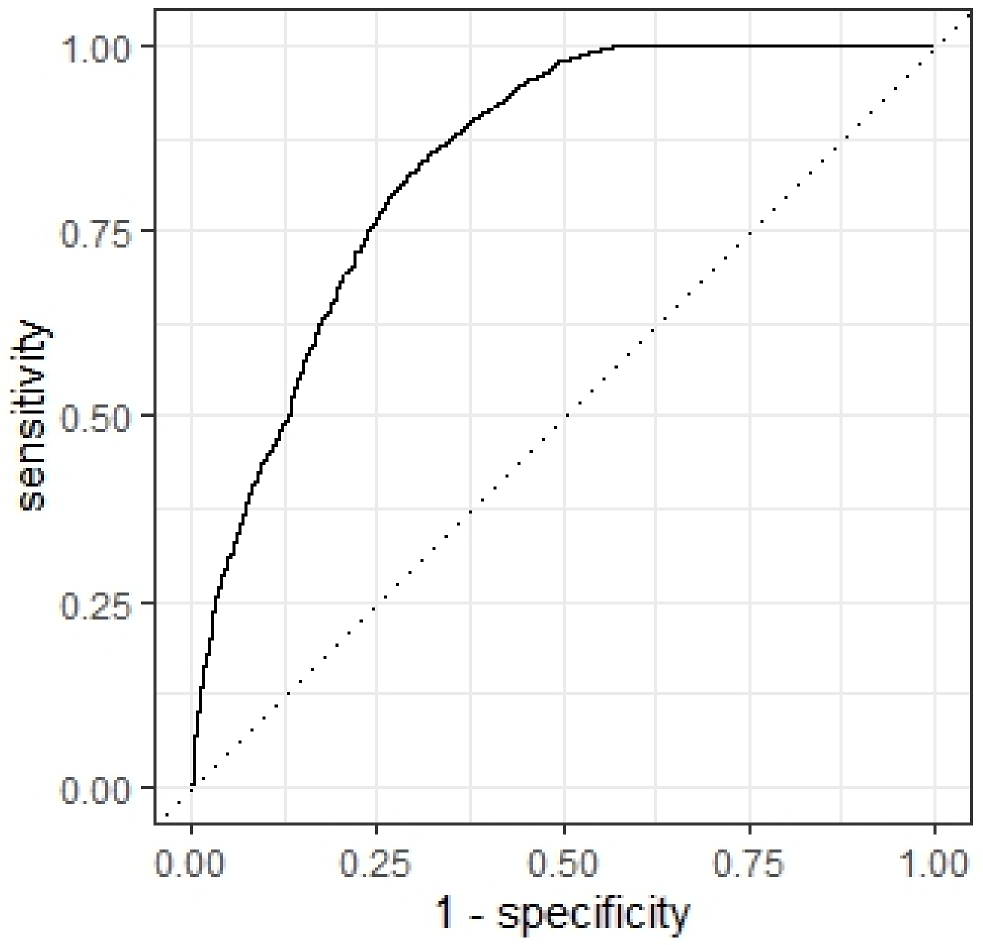
Feature engineering is the art and science of

- Creating,
- Transforming,
- Extracting, and
- Selecting variables

To improve model performance and interpretability

.

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

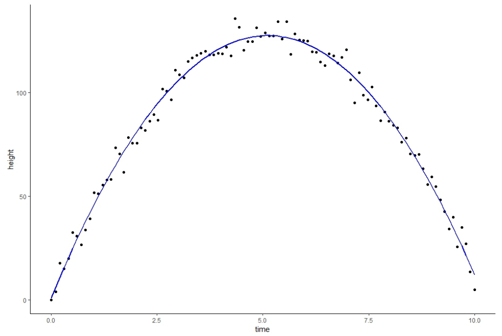


Jorge Zazueta
Research Professor and Head of the Modeling Group at the School of Economics, UASLP

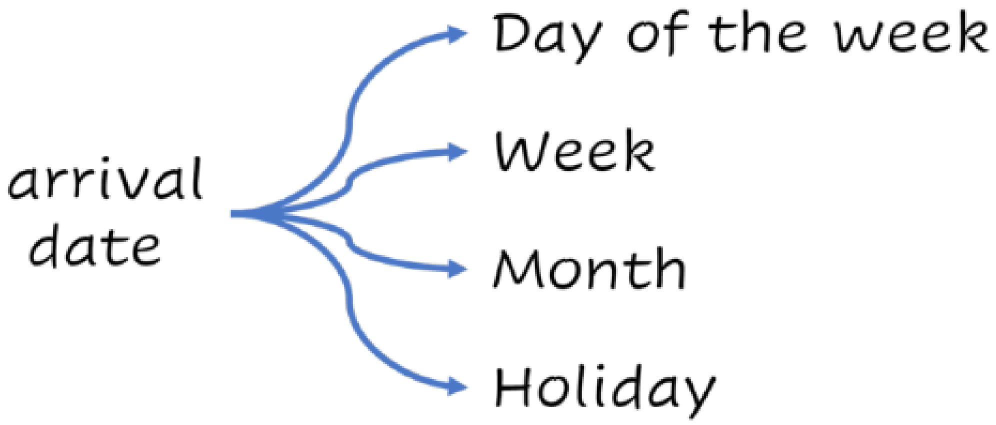
Domain knowledge

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

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



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

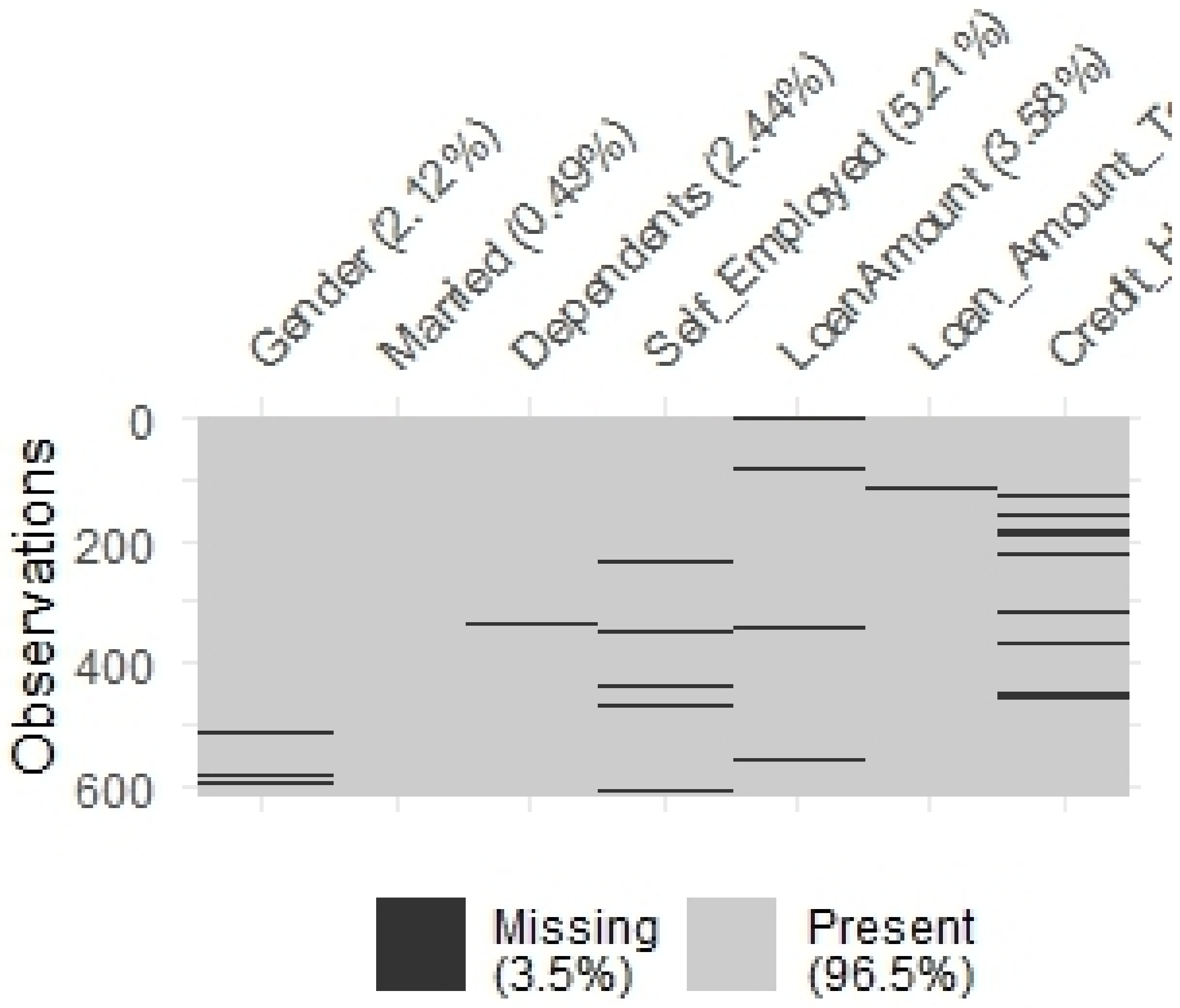



```
lr_recipe <-
  recipe(IsCanceled ~., data = train) %>%
  update_role(Agent, new_role = "ID" ) %>%
  step_date(arrival_date,
      features = c("dow", "week", "month")) %>%
  step_holiday(arrival_date,
      holidays = timeDate::listHolidays("US")) %>%
  step_rm(arrival_date) %>%
  step_dummy(all_nominal_predictors())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

Missing values and Dummy Variables

As values seem to be **missing completely at random** we can rely on traditional imputation methods.

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



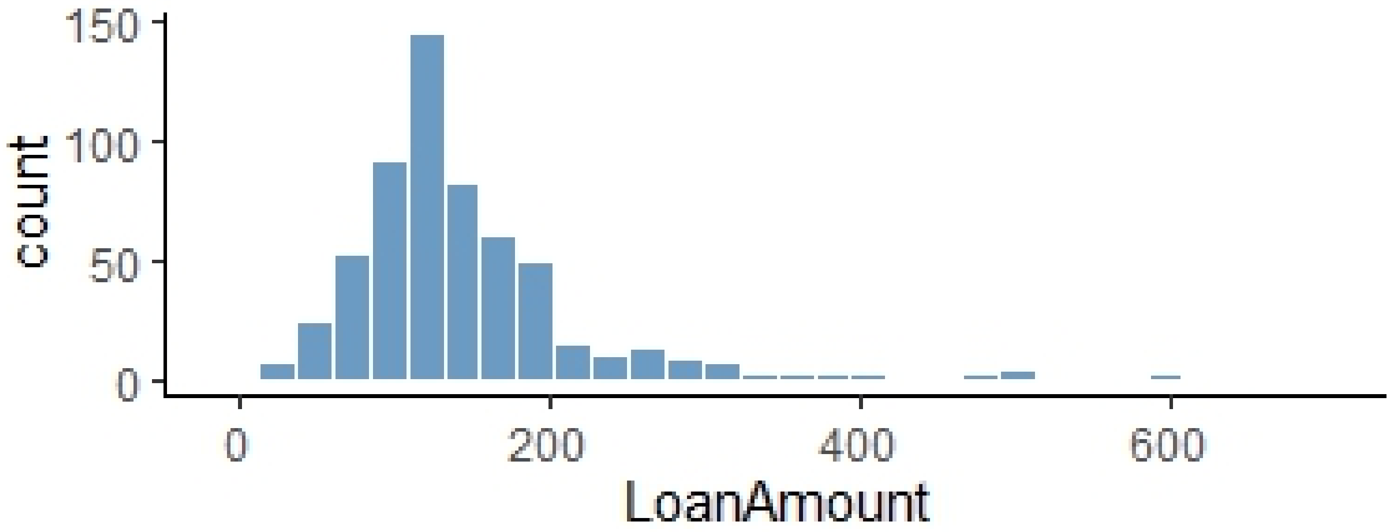
Factor_2	Factor_3	Factor_4
0	0	0
1	0	0
0	0	1
0	1	0
1	0	0

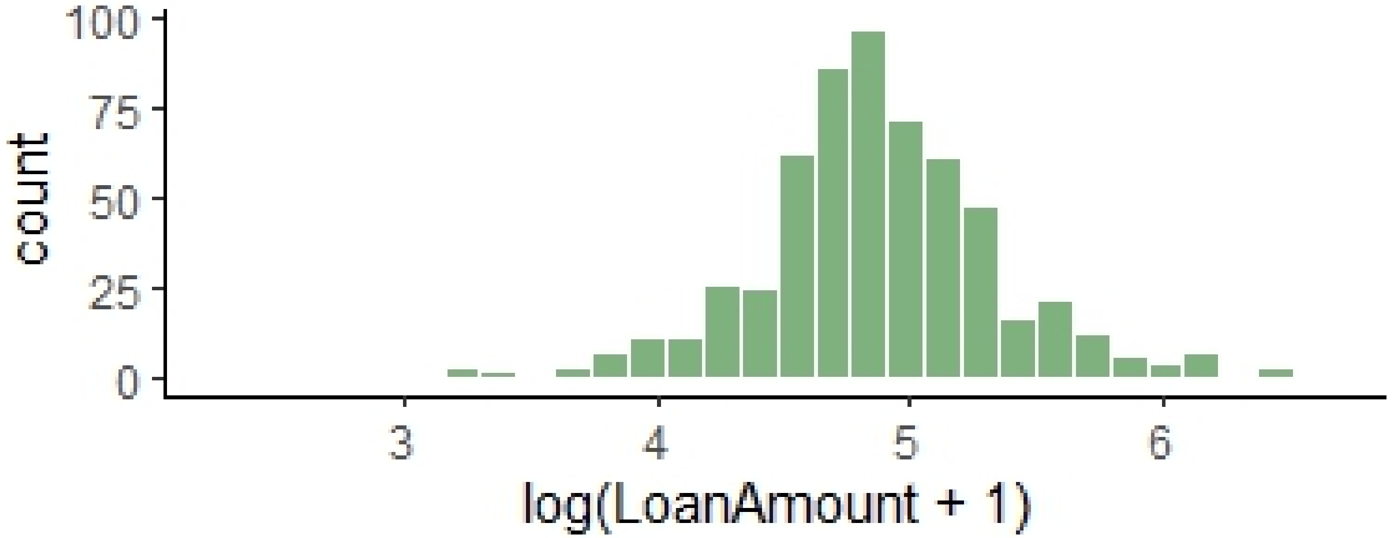
Log transformation

log-transform numerical features to:

- **Handle skewed data**
- **Reduce the impact of outliers**
- **Convert multiplicative relations into additive**
- Works only for positive values or log(variable + 1)
- Make the data more suitable for modeling

log-transformed loan amount data



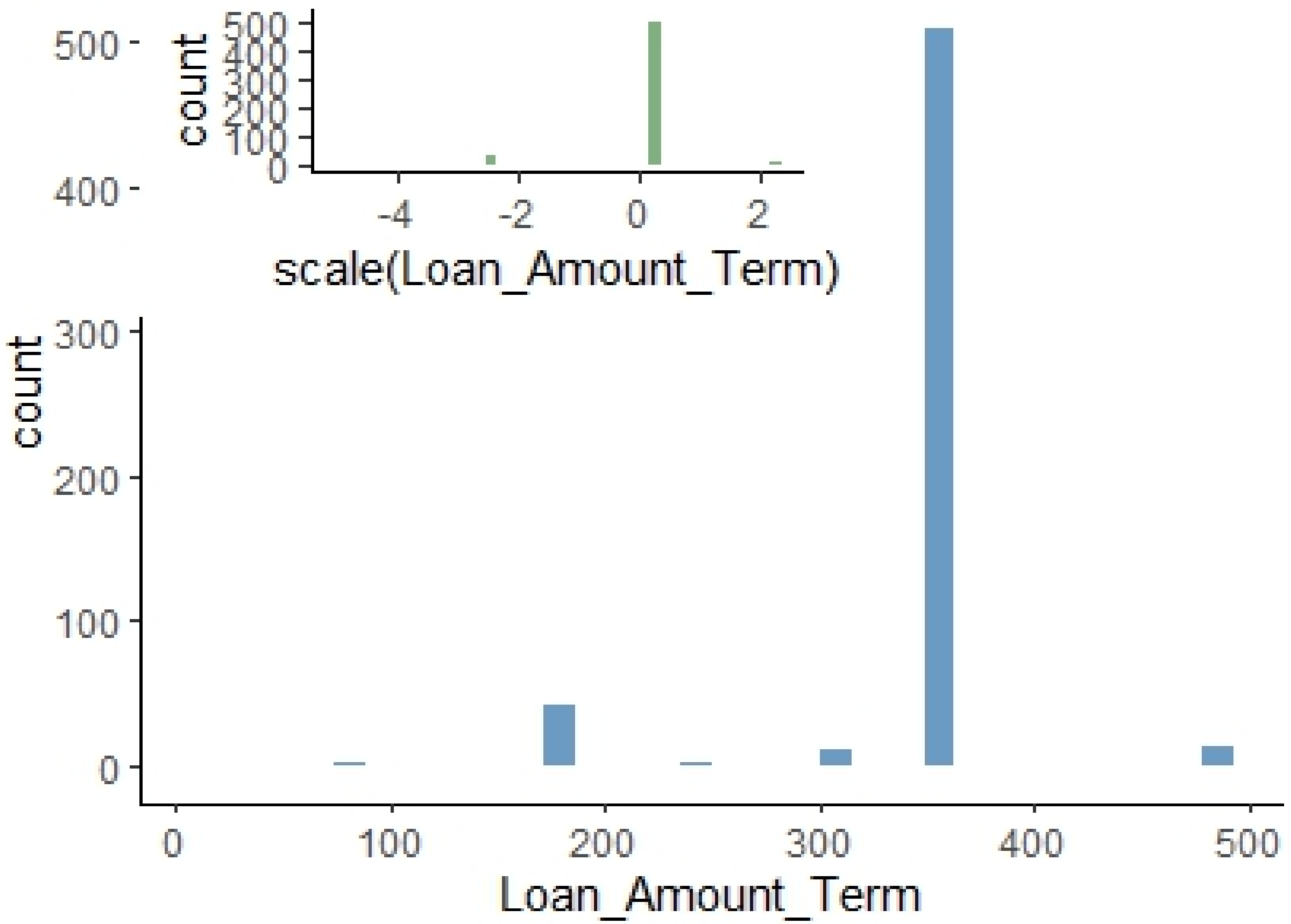


Normalization

Normalize or scale numerical features to:

- Prevent one feature from dominating the others
- Ease interpretation because it gives a comparable magnitude
- Make the data more suitable for modeling

e.g., loan amount term values shown vary significantly



Defining the model and the recipe

We can now declare a logistic regression model and add a recipe to impute, normalize and log-transform the relevant features.

```
lr_model <- logistic_reg()

lr_recipe <-
  recipe(Loan_Status ~.,
         data = train) %>%
  step_impute_knn(
    all_numeric_predictors())%>%
  step_normalize(Loan_Amount_Term) %>%
  step_log(all_numeric_predictors(),
           -Loan_Amount_Term, offset = 1)
```

```
class_evaluate <- metric_set(
  roc_auc, accuracy, sens)
```

```
lr_aug %>%
  class_evaluate(
    truth = Loan_Status,
    estimate = .pred_class,
    .pred_Y)
```

```
# A tibble: 3 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.813
2 sens     binary         0.467
3 roc_auc  binary         0.288
```

Applying transformations

Box-Cox recipe (take two)

```
lr_recipe_BC <- # Define recipe
  recipe(Loan_Status ~., data = train) %>%
  step_BoxCox(all_numeric(),
              -CoapplicantIncome)
lr_workflow_BC <- # Bundle workflows
  workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_BC)
lr_fit_BC <- # fit and augment
  lr_workflow_BC %>%
  fit(train)
```

Box-Cox

- Used to transform non-normal variable closer to normal
- As a family, it includes inverse, log, square and cubic roots as special cases
- Works for strictly positive values

$$\varphi(y,\lambda)=\begin{cases}\dfrac{y^{\lambda}-1}{y} & \lambda\neq 0, y>0\\ \log y & \lambda=0, y>0\end{cases}$$

Applying transformations

Yeo-Johnson recipe

```
lr_recipe_YJ <- # Define recipe
  recipe(Loan_Status ~., data = train) %>%
  step_YeoJohnson(all_numeric())
lr_workflow_YJ <- # Bundle workflows
  workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_YJ)
lr_fit_YJ <- # fit and augment
  lr_workflow_YJ %>%
  fit(train)
```

Yeo-Johnson

- Similar properties as Box-Cox
- Can handle zero and negative values
- For positive y is the same as Box-Cox of $y + 1$

$$\varphi(y,\lambda) = \begin{cases} \dfrac{(y+1)^{\lambda}-1}{\lambda} & \lambda \neq 0, y \geq 0 \\ \log(y+1) & \lambda = 0, y \geq 0 \\ -\dfrac{\left[(-y+1)^{2-\lambda}-1\right]}{2-\lambda} & \lambda \neq 2, y < 0 \\ -\log(-y+1) & \lambda = 2, y < 0 \end{cases}$$

The step_poly() function

`step_poly()` implements a polynomial expansion to one or more variables and passes it to our model.

```
lr_recipe_poly <-
  recipe(Loan_Status ~., data = train) %>%
  step_poly(all_numeric_predictors())
```

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

Results with `step_poly()`

```
# A tibble: 2 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.75
2 roc_auc  binary         0.703
```

The step_percentile() function

`step_percentile()` determines the empirical distribution of a variable based on the training set and converts all values to percentiles.

```
lr_recipe_perc <-
  recipe(Loan_Status ~., data = train) %>%
  step_percentile(all_numeric_predictors())
```

```
lr_workflow_perc <-
  workflow() %>%
  add_model(lr_model) %>%
  add_recipe(lr_recipe_perc)
```

Results with `step_percentile()`

```
# A tibble: 2 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.769
2 roc_auc  binary         0.677
```

There is no clear-cut rule for this

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.

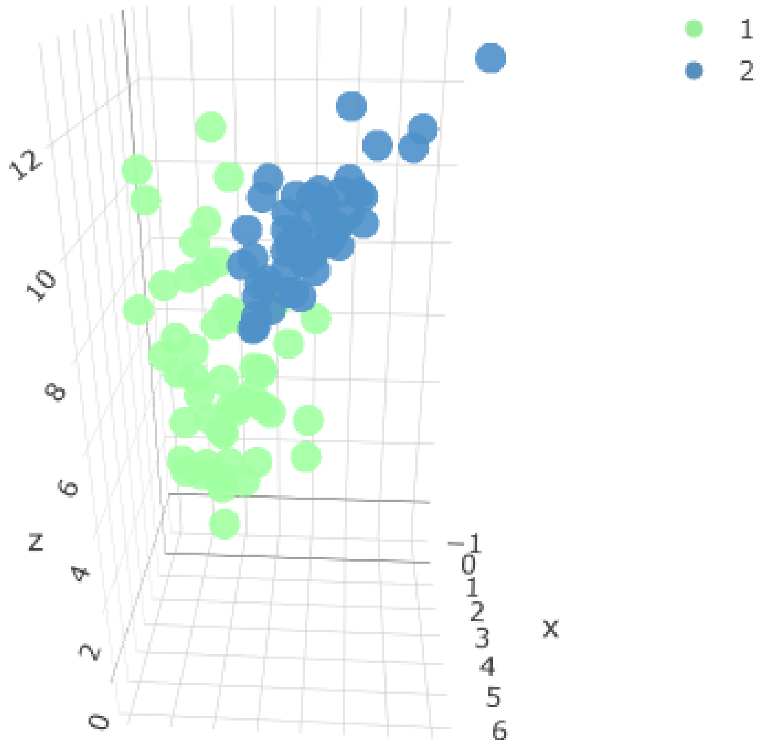
Principal Component Analysis (PCA)

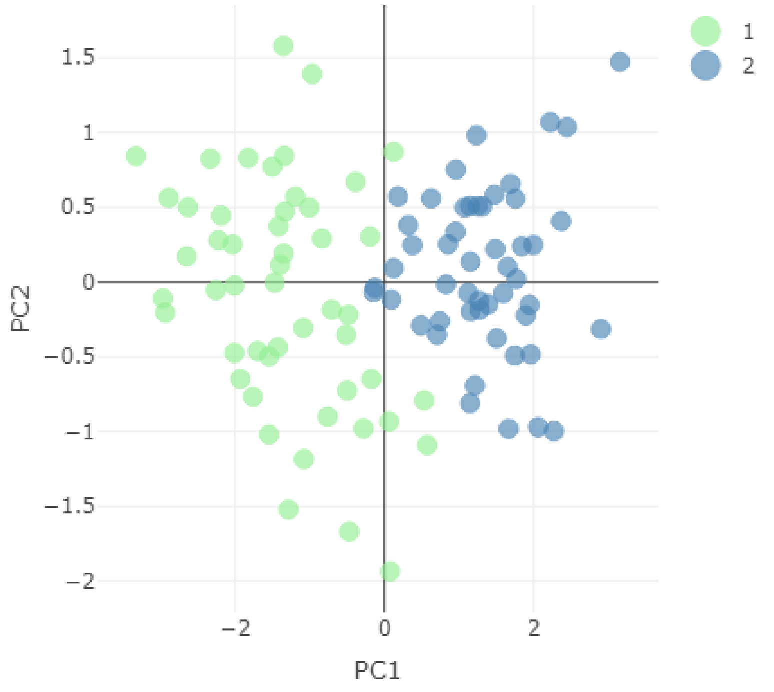
Creating a recipe to perform PCA and retrieving its output via `prep()` .

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

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

```
names(pca_output)
```





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
```

We need to keep at least 70% of variation

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

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)]
```

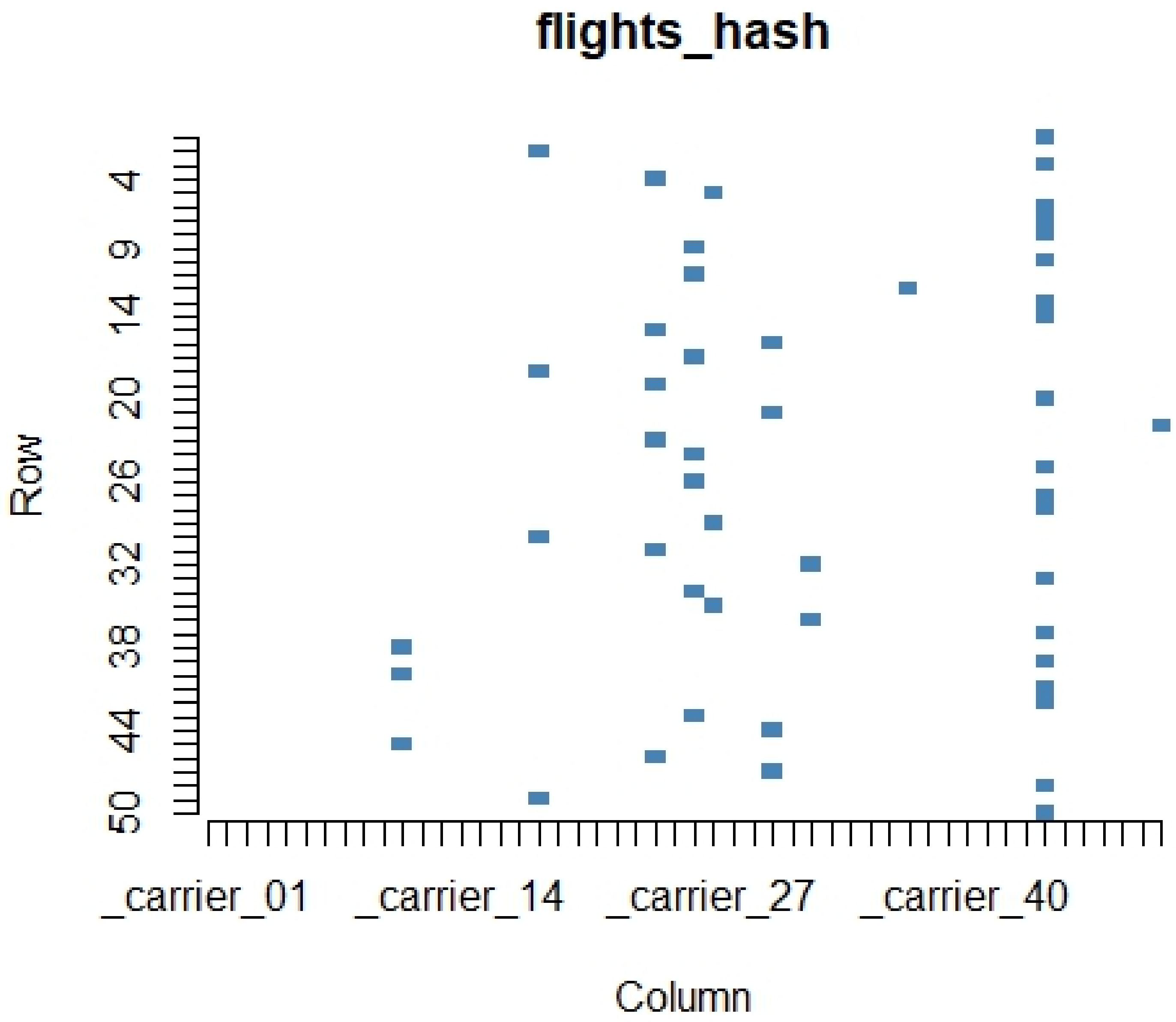
```
New names:
• `` -> `...1`
# A tibble: 10 × 4
   ...1  `_carrier_17` `_carrier_18` `_carrier_19`
   <chr>         <int>         <int>         <int>
 1 EV                0             0             0
 2 B6                0             1             0
 3 EV                0             0           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



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
```

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
```

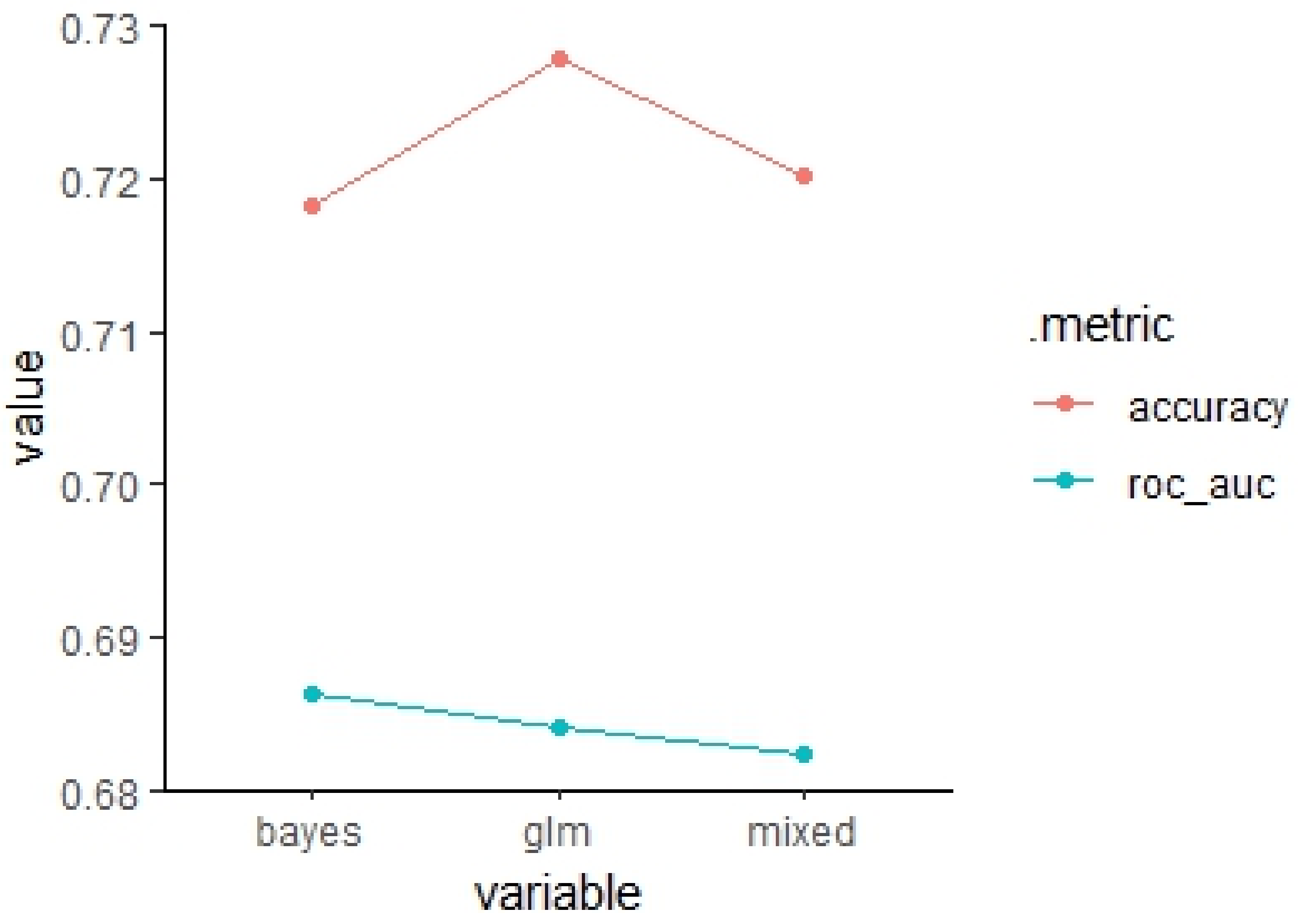
Visualizing our results

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.

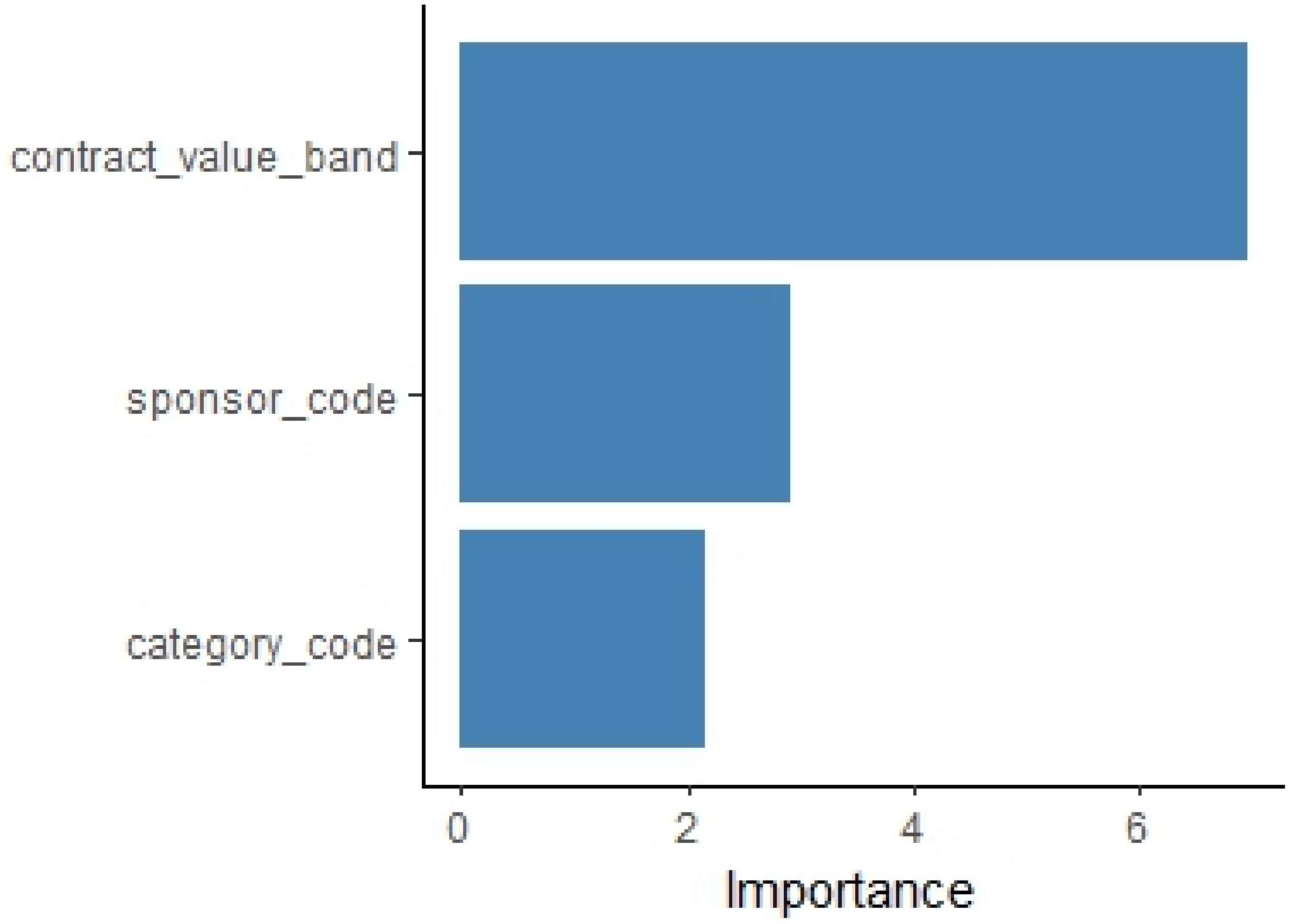


Which variables matter most?

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

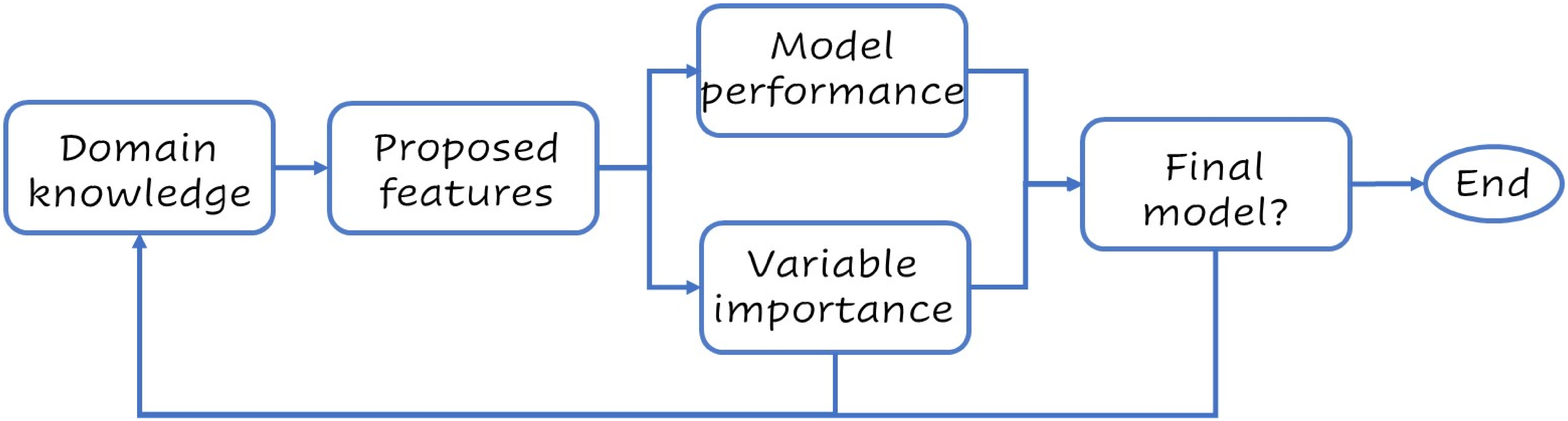
```
lr_fit %>%
  extract_fit_parsnip() %>%
  vip(aesthetics =
      list(fill = "steelblue"))
```

Variable importance chart



Variable importance and feature engineering

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



Reasons to reduce the number of features

Eliminating irrelevant or low-information variables can have benefits, including

- Reduce model variance without significantly increasing bias
- Increase out-of-sample model performance
- Reducing computation time
- Decreasing model complexity
- Improving interpretability

Build a reduced model by creating a features ve

A feature vector can be passed used to select features before training.

```
# Feature vector
features <- c("Credit_History", "Property_Area", "LoanAmount", "Loan_Status")

# Training and testing data
train_features <- train %>% select(all_of(features))
test_features <- test %>% select(all_of(features))

# Create recipe and bundle with model
recipe_features <- recipe(Loan_Status ~., data = train_features)
workflow_features <- workflow() %>% add_model(lr_model) %>%
  add_recipe(recipe_features)
```

Two common regularization techniques

Lasso

- Adds penalty term proportional to absolute value of model weights
- Encourages some weights to become exactly zero
- Effectively eliminates the corresponding features
- Can be an automated feature selection method

Ridge

- Adds penalty term proportional to square of model weights
- Does not shrink some coefficients to zero like Lasso
- But can effectively reduce overfitting

Hyperparameter tuning

Setting a model with tuning

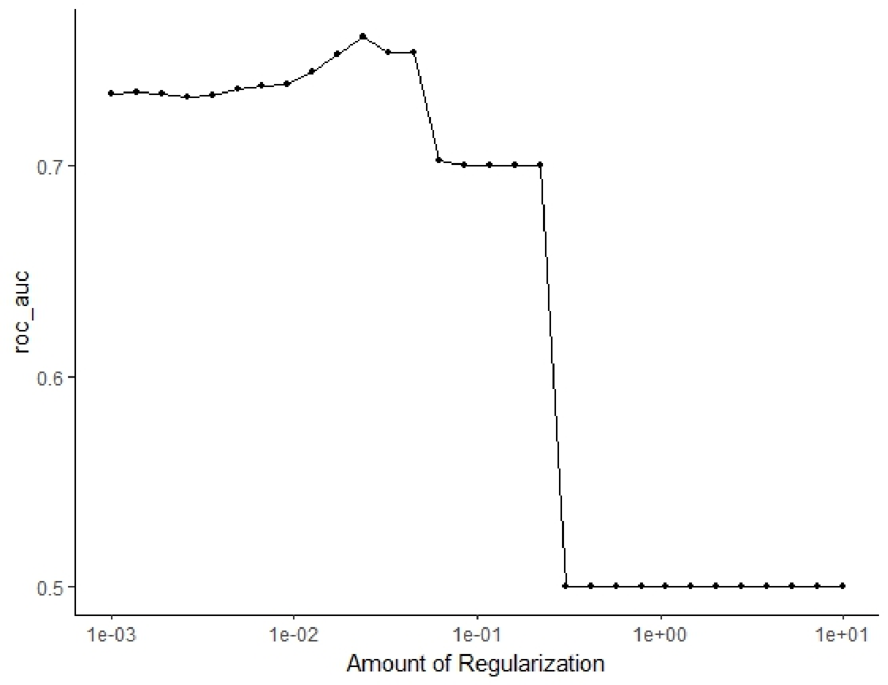
```
model_lasso_tuned <- logistic_reg() %>%
  set_engine("glmnet") %>%
  set_args(mixture = 1,
  penalty = tune())

workflow_lasso_tuned <-
  workflow() %>%
  add_model(model_lasso_tuned) %>%
  add_recipe(recipe)

penalty_grid <- grid_regular(
  penalty(range = c(-3, 1)),
  levels = 30)
```

Looking at the tuning output

```
tune_output <- tune_grid(
  workflow_lasso_tuned,
  resamples = vfold_cv(train, v = 5),
  metrics = metric_set(roc_auc),
  grid = penalty_grid)
autoplot(tune_output)
```



Exploring the results

Auto-chosen features

```
best_penalty <-
select_by_one_std_err(tune_output,
metric = 'roc_auc', desc(penalty))

# Fit Final Model
final_fit<-
finalize_workflow(workflow_lasso_tuned,
best_penalty) %>%
  fit(data = train)
```

```
final_fit_se %>% tidy()
```

```
# A tibble: 15 × 3
   term                      estimate penalty
   <chr>                        <dbl>   <dbl>
 1 (Intercept)                 -0.660  0.0452
 2 ApplicantIncome              0      0.0452
 3 CoapplicantIncome            0      0.0452
 4 LoanAmount                   0      0.0452
 5 Loan_Amount_Term             0      0.0452
 6 Credit_History              -0.948  0.0452
 7 Gender_Female                0      0.0452
 8 Married_Yes                 -0.191  0.0452
 9 Dependents_X1                0      0.0452
10 Dependents_X2                0      0.0452
11 Dependents_X3.               0      0.0452
12 Education_Not.Graduate       0      0.0452
13 Self_Employed_Yes            0      0.0452
14 Property_Area_Rural          0      0.0452
15 Property_Area_Semiurban     -0.163  0.0452
```

Simple logistic regression vs. tuned Lasso

Ridge regularization

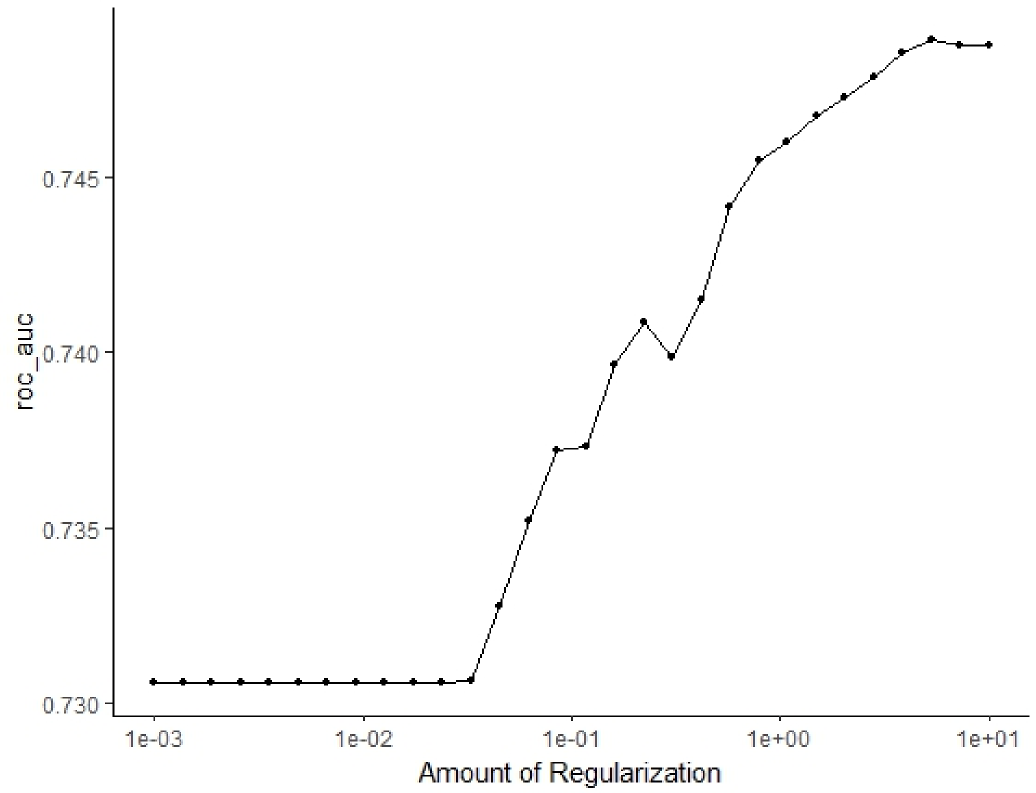
Ridge is the option when mixture = 0

```
model_ridge_tuned <- logistic_reg() %>%
  set_engine("glmnet") %>%
  set_args(mixture = 0, penalty = tune())

workflow_ridge_tuned <-
  workflow() %>%
  add_model(model_ridge_tuned) %>%
  add_recipe(recipe)

tune_output <- tune_grid(
  workflow_ridge_tuned,
  resamples = vfold_cv(train, v = 5),
  metrics = metric_set(roc_auc),
  grid = penalty_grid)
```

```
tune_output <- tune_grid(
  workflow_ridge_tuned,
  resamples = vfold_cv(train, v = 5),
  metrics = metric_set(roc_auc),
  grid = penalty_grid)
 autoplot(tune_output)
```



Ridge regularization

```
best_penalty <-
select_by_one_std_err(tune_output,
metric = 'roc_auc', desc(penalty))
best_penalty

final_fit<-
finalize_workflow(workflow_ridge_tuned,
best_penalty) %>%
  fit(data = train)
```

```
tidy(final_fit)
```

```
# A tibble: 15 × 3
   term                     estimate penalty
   <chr>                       <dbl>   <dbl>
 1 (Intercept)             -0.799         10
 2 ApplicantIncome          0.00232       10
 3 CoapplicantIncome        0.0000537     10
 4 LoanAmount               0.00291       10
 5 Loan_Amount_Term         0.00161       10
 6 Credit_History          -0.0245        10
 7 Gender_Female            0.00850       10
 8 Married_Yes             -0.0140        10
 9 Dependents_X1            0.00497       10
10 Dependents_X2           -0.0100        10
11 Dependents_X3.           0.00259       10
12 Education_Not.Graduate   0.00308       10
13 Self_Employed_Yes        0.00892       10
14 Property_Area_Rural      0.0109        10
```

Where to go from here?

Data science is a never ending journey that keeps refreshing itself. These are some datacamp courses that you might considering as next steps.

- Dimensionality reduction in R
- Advanced dimensionality reduction in R
- Modeling with tidymodels in R