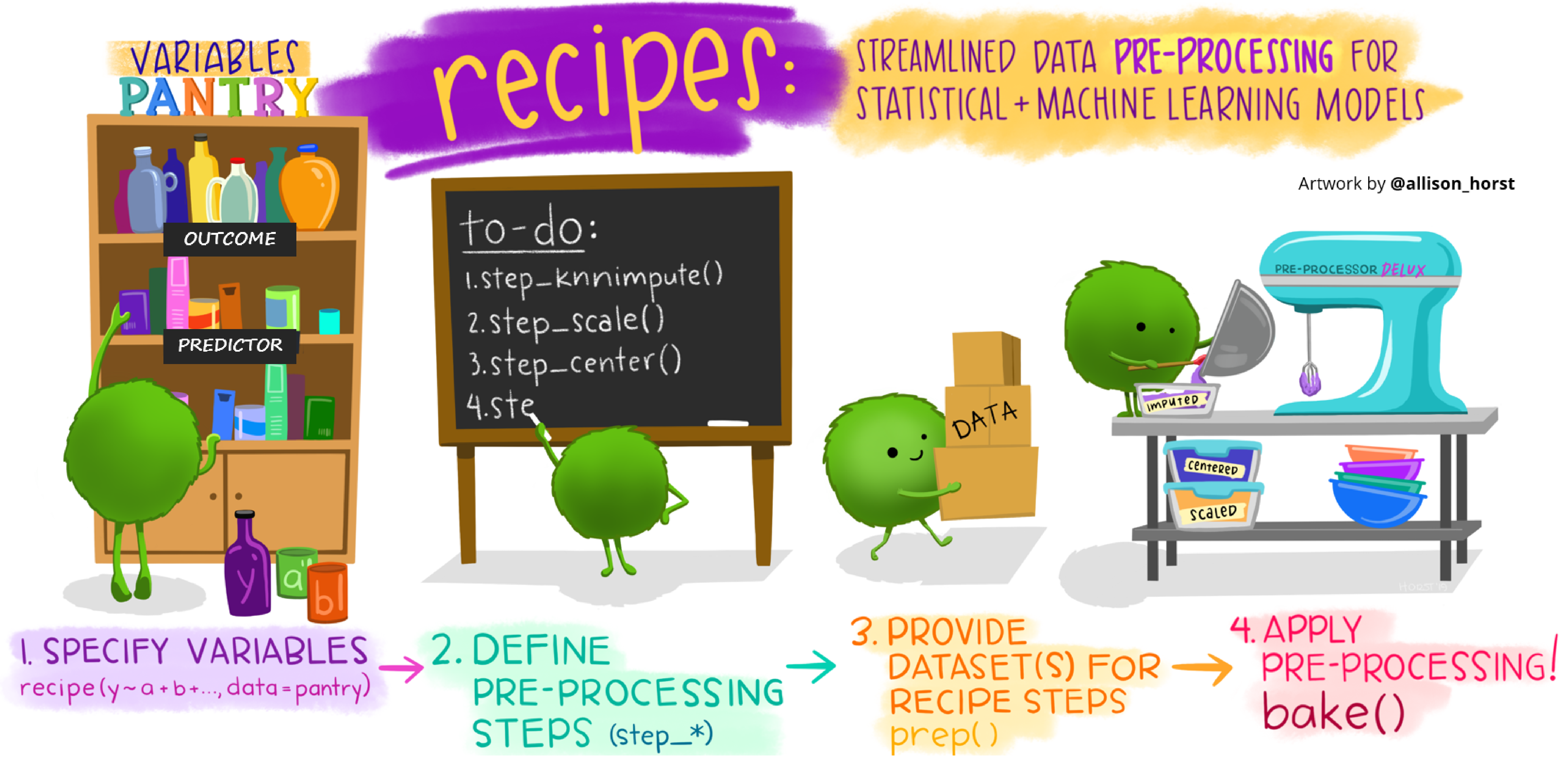
Feature engineering

MODELING WITH TIDYMODELS IN R

David Svancer
Data Scientist

datacamp

Feature engineering with the recipes package



Artwork by **@allison_horst**

Specifying variable types and roles

Define column roles

- Assign outcome or predictor role to all variables

Determine variable data types

- Numeric data
- Categorical data

Accomplished with the `recipe()` function

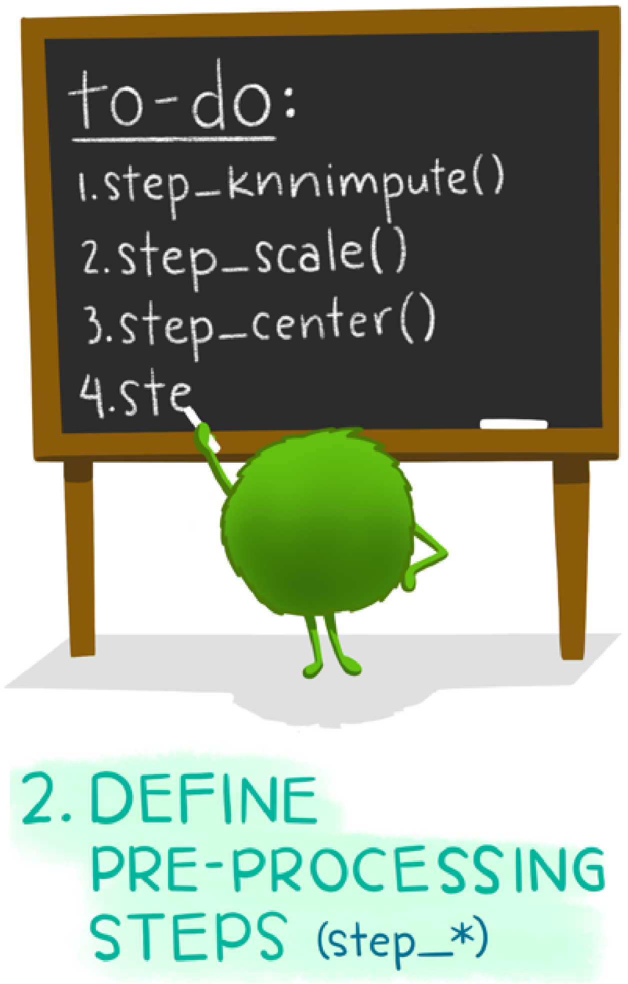
Artwork by **@allison_horst**

Data preprocessing steps

Add required data preprocessing steps

- Imputation of missing data
- Data transformations
  - Centering and scaling numeric variables
- Creating new variables
  - Calculating ratios of variables
- And many more...

Each step is added with a unique `step_*()` function



Artwork by **@allison_horst**

[1] https://recipes.tidymodels.org/reference/index.html

Training preprocessing steps

`recipe` objects are trained on a data source, typically the training dataset

- Data transformations are estimated
  - Mean and standard deviation of numeric columns for centering and scaling
  - Formulas for creating new columns are stored for applying to new data

Recipes are trained with the `prep()` function

Artwork by @allison_horst

Applying recipes to new data

Apply all trained data preprocessing transformations

- To the training and test datasets for modeling
- To new sources of data for future predictions
  - Machine learning algorithms require the same data format as was used during training to predict new values

Artwork by **@allison_horst**

Recipes are applied with the `bake()` function

Simple feature engineering pipeline

Log transform `total_time` in lead scoring data

- Common transformation for large data values
- Compresses the range of data values and reduces variability

```
leads_training
```

```
# A tibble: 996 x 7
   purchased  total_visits  total_time  pages_per_visit  total_clicks  lead_source    us_location
   <fct>             <dbl>       <dbl>            <dbl>          <dbl>          <fct>           <fct>
 1 yes                7         1148          7               59         direct_traffic     west
 2 no                 5          228          2.5             25         email              southeast
 3 no                 7          481          2.33            21         organic_search     west
 4 no                 4          177          4               37         direct_traffic     west
 5 no                 2         1273          2               26         email              midwest
# ... with 991 more rows
```

Building a recipe object

The `recipe()` function

- Model formula
  - Assigns variable roles
- `data` argument
  - Determines variable data types

Pass `recipe` object to `step_log()` to add logarithm transformation step

- Select variable for transformation, `total_time`, and specify logarithm base

```
leads_log_rec <- recipe(purchased ~ .,
                        data = leads_training) %>%
  step_log(total_time, base = 10)
```

```
leads_log_rec
```

```
Data Recipe
Inputs:
      role #variables
   outcome          1
 predictor          6

Operations:

Log transformation on total_time
```

Explore variable roles and types

Passing a `recipe` object to the `summary()` function

- Creates a tibble with variable information
- `type` column
  - Captures data type of variable
  - 'nominal' represents categorical variables
- `role` column
  - Captures variable roles for modeling
  - Assigned based on input model formula

```
leads_log_rec %>%
  summary()
```

```
# A tibble: 7 x 4
  variable        type    role      source
  <chr>           <chr>   <chr>     <chr>
1 total_visits    numeric predictor original
2 total_time      numeric predictor original
3 pages_per_visit numeric predictor original
4 total_clicks    numeric predictor original
5 lead_source     nominal predictor original
6 us_location     nominal predictor original
7 purchased       nominal outcome   original
```

Training a recipe object

The `prep()` function

- Takes a `recipe` object as the first argument
- `training` argument
  - Specifies the data on which to train data preprocessing steps

```
leads_log_rec_prep <- leads_log_rec %>%
  prep(training = leads_training)
```

```
leads_log_rec_prep
```

```
Data Recipe
Inputs:
      role #variables
   outcome          1
 predictor          6

Training data contained 996 data points and
no missing data.

Operations:
Log transformation on total_time [trained]
```

Printing a trained `recipe` object

- Trained steps are indicated by `[trained]`

Transforming the training data

The `bake()` function

- First argument is a trained `recipe` object
- `new_data` argument
  - Data on which to apply trained `recipe`
- Training data
  - `leads_training` was used to train the `recipe`
  - By default, transformed data is retained by `prep()` function
  - Pass `NULL` to `new_data` to extract
- Returns a tibble with transformed data

```
leads_log_rec_prep %>%
  bake(new_data = NULL)
```

```
# A tibble: 996 x 7
  total_visits total_time ... us_location  purchased
    <dbl>        <dbl>    ...   <fct>        <fct>
 1      7         3.06    ...   west          yes
 2      5         2.36    ...   southeast    no
 3      7         2.68    ...   west         no
 4      4         2.25    ...   west         no
 5      2         3.10    ...   midwest      no
# ... with 991 more rows
```

Transforming new data

Transforming datasets not used during `recipe` training

- Pass dataset to `new_data` argument
- Trained `recipe` will apply all steps to new data sources

```
leads_log_rec_prep %>%
  bake(new_data = leads_test)
```

```
# A tibble: 332 x 7
 total_visits  total_time ... us_location  purchased
     <dbl>        <dbl>    ...  <fct>        <fct>
 1     8            2      ...  west          no
 2     4            3.13   ...  northeast     yes
 3     3            2.25   ...  west          no
 4     2            1.20   ...  midwest       no
 5     9            3.01   ...  west          yes
# ... with 327 more rows
```

Let's get baking!

MODELING WITH TIDYMODELS IN R

Numeric predictors

MODELING WITH TIDYMODELS IN R

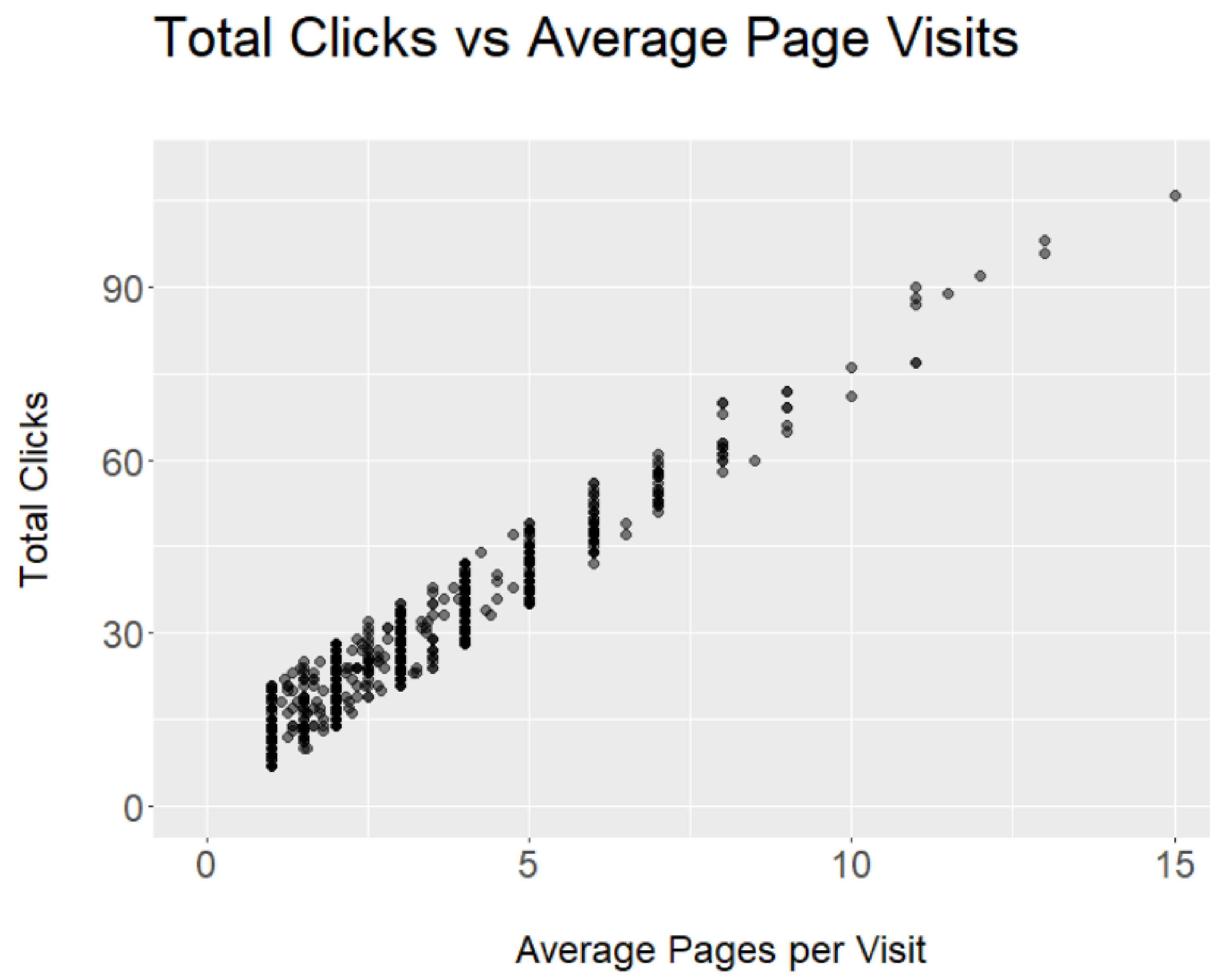
David Svancer
Data Scientist

Correlated predictor variables

Correlation measures the strength of a linear relationship between two numeric variables

- Ranges from -1 to 1
- Highly correlated predictors near -1 or 1
  - Provide *redundant information*
  - Model fitting problems (*multicollinearity*)

```
ggplot(leads_training,
       aes(x = pages_per_visit, y = total_clicks)) +
  geom_point()  +
  labs(title = 'Total Clicks vs Average Page Visits',
       y = 'Total Clicks', x = 'Average Pages per Visit')
```



Finding correlated predictor variables

Calculate a *correlation matrix*

- Pass dataset to `select_if()` function
  - Provide `is.numeric` as argument
- Pass to `cor()` function

```
leads_training %>%
  select_if(is.numeric) %>%
  cor()
```

```
                total_visits total_time pages_per_visit total_clicks
total_visits           1.00       0.01            0.43         0.42
total_time             0.01       1.00            0.02         0.01
pages_per_visit        0.43       0.02            1.00         0.96
total_clicks           0.42       0.01            0.96         1.00
```

Processing correlated predictors

Removing multicollinearity with `recipes`

- Specify `recipe` object with `recipe()` function
- Pass to `step_corr()`
  - Add all numeric columns
    - Column names separated by commas
  - Provide correlation `threshold`
    - Absolute value
    - Threshold of 0.9 removes correlations at 0.9 or more and -0.9 or less

```
leads_cor_rec <- recipe(purchased ~ .,
                        data = leads_training) %>%
  step_corr(total_visits, total_time,
            pages_per_visit, total_clicks,
            threshold = 0.9)

leads_cor_rec
```

```
Data Recipe
Inputs:
      role #variables
   outcome          1
 predictor          6

Operations:
Correlation filter on total_visits,..., total_clicks
```

Selecting predictors by type

- `all_outcomes()`
  - Selects the outcome variable
- `all_numeric()`
  - Selects all numeric variables
    - Will include the outcome variable if it is numeric

To select numeric predictors for `recipe` steps

- Pass `all_numeric()` to `step_*()` functions
- If outcome variable is numeric, also pass `-all_outcomes()`

```
leads_cor_rec <- recipe(purchased ~ .,
                        data = leads_training) %>%
  step_corr(all_numeric(), threshold = 0.9)

leads_cor_rec
```

```
Data Recipe
Inputs:
      role #variables
   outcome          1
 predictor          6

Operations:
Correlation filter on all_numeric()
```

Training and applying the recipe

- Train with `prep()`
  - Provide `leads_training` for training
- Apply with `bake()`
  - `pages_per_visit` removed from `leads_test`
  - `pages_per_visit` will be removed from all future data as well

```
leads_cor_rec %>%
  prep(training = leads_training) %>%
  bake(new_data = leads_test)
```

```
# A tibble: 332 x 6
total_visits total_time total_clicks ... purchased
    <dbl>        <dbl>        <dbl>    ...   <fct>
 1   8           100           24      ...    no
 2   4           1346          22      ...    yes
 3   3           176           27      ...    no
 4   2           16            12      ...    no
 5   9           1022          12      ...    yes
# ... with 327 more rows
```

Normalization

Centering and scaling numeric variables

- Subtract the mean
- Divide by the standard deviation
- Transforms data to standard deviation units
  - Transformed variable will have a mean of 0 and standard deviation of 1

The `total_time` variable in `leads_training`

- Spending 1,273 seconds on the website is 1.19 standard deviations greater than the average time spent by customers

total_time
1148
228
481
177
1273
711
...

$$\left(\frac{\text{total\_time} - 631}{539.5} \right) \rightarrow$$

total_time_norm
0.96
-0.75
-0.28
-0.84
1.19
0.15
...

Combining data preprocessing steps

Normalizing numeric predictors with `recipes`

- `step_normalize()`
  - Column names or `all_numeric()` selector
  - Means and standard deviations from training data columns applied to new data sources

Multiple `step_*()` functions can be added to a `recipe`

- Order matters

```
leads_norm_rec <- recipe(purchased ~ .,
                         data = leads_training) %>%
  step_corr(all_numeric(), threshold = 0.9) %>%
  step_normalize(all_numeric())

leads_norm_rec
```

```
Data Recipe
Inputs:
      role #variables
   outcome          1
 predictor          6

Operations:
Correlation filter on all_numeric()
Centering and scaling for all_numeric()
```

Transforming the test data

`pages_per_vist` is removed and numeric predictors are normalized

```
leads_norm_rec %>%
  prep(training = leads_training) %>%
  bake(new_data = leads_test)
```

```
# A tibble: 332 x 6
 total_visits  total_time  total_clicks  lead_source   us_location  purchased
        <dbl>       <dbl>         <dbl>    <fct>           <fct>       <fct>
 1     0.864      -0.984     -0.360        direct_traffic   west          no
 2    -0.151       1.33      -0.506        direct_traffic   northeast     yes
 3    -0.405      -0.843     -0.140        organic_search   west          no
 4    -0.659      -1.14      -1.24         email            midwest       no
 5     1.12        0.725     -1.24         direct_traffic   west          yes
# ... with 327 more rows
```

Let's practice!

MODELING WITH TIDYMODELS IN R

Nominal predictors

MODELING WITH TIDYMODELS IN R

David Svancer
Data Scientist

Nominal data

Data that encodes characteristics or groups

- No meaningful order

Examples

- Department within a company
  - Marketing, Finance, Technology
- Native language
  - English, Czech, Spanish ...
- Car type
  - SUV, sedan, compact ...

Transforming nominal predictors

Nominal data must be transformed to numeric data for modeling

One-Hot Encoding

- Maps categorical values to a sequence of [0/1] indicator variables
- Indicator variable for each unique value in original data

department
finance
marketing
technology

→

department_finance	department_marketing	department_technology
1	0	0
0	1	0
0	0	1

Transforming nominal predictors

Dummy Variable Encoding

- Excludes *one value* from original set of data values
  - ***n*** distinct values produce (***n - 1***) indicator variables
- Preferred method for modeling
  - Default in `recipes` package

department
finance
marketing
technology

→

department_marketing	department_technology
0	0
1	0
0	1

Lead scoring data

Nominal predictor variables - `lead_source` and `us_location`

```
leads_training
```

```
# A tibble: 996 x 7
 purchased total_visits total_time pages_per_visit total_clicks lead_source    us_location
   <fct>       <dbl>       <dbl>           <dbl>            <dbl>        <fct>            <fct>
 1 yes          7          1148            7                59          direct_traffic    west
 2 no           5          228             2.5              25          email             southeast
 3 no           7          481             2.33             21          organic_search    west
 4 no           4          177             4                37          direct_traffic    west
 5 no           2          1273            2                26          email             midwest
# ... with 991 more rows
```

Creating dummy variables

The `step_dummy()` function

- Creates dummy variables from nominal predictor variables

```
recipe(purchased ~ ., data = leads_training) %>%
  step_dummy(lead_source, us_location) %>%
  prep(training = leads_training) %>%
  bake(new_data = leads_test)
```

```
# A tibble: 332 x 12
   total_visits ... lead_source_email  lead_source_organic_search  lead_source_direct_traffic  us_location_southeast ... us_location_west
        <dbl>    ...      <dbl>                    <dbl>                       <dbl>                       <dbl>                   <dbl>
1        8      ...        0                        0                           1                           0                       1
2        4      ...        0                        0                           1                           0                       0
3        3      ...        0                        1                           0                           0                       1
4        2      ...        1                        0                           0                           0                       0
5        9      ...        0                        0                           1                           0                       1

# ... with 327 more rows
```

Selecting columns by type

Selecting by column type using `all_nominal()` and `all_outcomes()` selectors

- `-all_outcomes()` excludes the nominal outcome variable, `purchased`

```
recipe(purchased ~ ., data = leads_training) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  prep(training = leads_training) %>%
  bake(new_data = leads_test)
```

```
# A tibble: 332 x 12
   total_visits ... lead_source_email  lead_source_organic_search  lead_source_direct_traffic ... us_location_west
       <dbl>    ...      <dbl>                     <dbl>                       <dbl>                       <dbl>
1        8      ...        0                         0                           1                           1
2        4      ...        0                         0                           1                           0
3        3      ...        0                         1                           0                           1
4        2      ...        1                         0                           0                           0
5        9      ...        0                         0                           1                           1
# ... with 327 more rows
```

Preprocessing nominal predictor variables

Modeling engines in R

- Many include automatic dummy variable creation
  - Possible to use nominal predictors without preprocessing with `step_dummy()`
- Not consistent across all engines
  - One-hot vs dummy variables
  - Naming of new variables

The `recipes` package provides a standardized way to prepare nominal predictors for modeling

Let's practice!

MODELING WITH TIDYMODELS IN R

Complete modeling workflow

MODELING WITH TIDYMODELS IN R

David Svancer
Data Scientist

Data resampling

Creating training and test datasets

- `initial_split()`
  - Create data split object
- `training()`
  - Build training dataset
- `testing()`
  - Build test dataset

```
leads_split <- initial_split(leads_df,
                             strata = purchased)

leads_training <- leads_split %>%
  training()

leads_test <- leads_split %>%
  testing()
```

Model specification

Specify model with `parsnip`

- `logistic_reg()`
  - General interface to logistic regression models
- `set_engine()`
  - 'glm' engine
- `set_mode()`
  - `purchased` is a nominal outcome variable
  - Mode should be 'classification'

```
logistic_model <- logistic_reg() %>%
  set_engine('glm') %>%
  set_mode('classification')
```

```
Logistic Regression Model
Specification (classification)

Computational engine: glm
```

Feature engineering

Specify feature engineering steps with `recipes`

- `recipe()`
  - Model formula and training data
- `step_*()` functions
  - Sequential preprocessing steps

```
leads_recipe <- recipe(purchased ~ .,
                       data = leads_training) %>%
  step_corr(all_numeric(), threshold = 0.9) %>%
  step_normalize(all_numeric()) %>%
  step_dummy(all_nominal(), -all_outcomes())
```

```
leads_recipe
```

```
Data Recipe
Inputs:
      role #variables
   outcome          1
 predictor          6

Operations:
Correlation filter on all_numeric()
Centering and scaling for all_numeric()
Dummy variables from all_nominal(), -all_outcomes()
```

Recipe training

Train feature engineering steps on the training data

- `prep()`
  - Pass `recipe` object to `prep()`
  - Add `leads_training` for training data

```
leads_recipe_prep <- leads_recipe %>%
  prep(training = leads_training)
```

```
leads_recipe_prep
```

```
Data Recipe
Inputs:
      role #variables
   outcome          1
 predictor          6
Training data contained 996 data points
and no missing data.

Operations:
Correlation filter removed pages_per_visit [trained]
Centering and scaling for total_visits ... [trained]
Dummy variables from lead_source, us_location [trained]
```

Preprocess training data

Apply trained `recipe` to the training data and save the results for modeling fitting

```
leads_training_prep <- leads_recipe_prep %>%
  bake(new_data = NULL)
leads_training_prep
```

```
# A tibble: 996 x 11
total_visits  total_time   ... lead_source_email  lead_source_organic_search ...  us_location_west
     <dbl>      <dbl>                <dbl>                           <dbl>                    <dbl>
 1    0.611      0.958      ...        0                               0           ...          1
 2    0.103     -0.747      ...        1                               0           ...          0
 3    0.611     -0.278      ...        0                               1           ...          1
 4   -0.151     -0.842      ...        0                               0           ...          1
 5   -0.659      1.19       ...        1                               0           ...          0
# ... with 991 more rows
```

Preprocess test data

Apply trained `recipe` to the test data and save the results for modeling evaluation

```
leads_test_prep <- leads_recipe_prep %>%
  bake(new_data = leads_test)

leads_test_prep
```

```
# A tibble: 332 x 11
 total_visits  total_time  ...  lead_source_email  lead_source_organic_search ...  us_location_west
     <dbl>       <dbl>                <dbl>                          <dbl>                     <dbl>
 1    0.864      -0.984     ...        0                              0             ...           1
 2   -0.151       1.33      ...        0                              0             ...           0
 3   -0.405      -0.843     ...        0                              1             ...           1
 4   -0.659      -1.14      ...        1                              0             ...           0
 5    1.12        0.725     ...        0                              0             ...           1
# ... with 327 more rows
```

Model fitting and predictions

Train logistic regression model with `fit()`

- Use the **preprocessed training dataset**, `leads_training_prep`

```
logistic_fit <- logistic_model %>%
  fit(purchased ~ .,
      data = leads_training_prep)
```

Obtain model predictions with `predict()`

- Predict outcome values and estimated probabilities
- Use the **preprocessed test dataset**, `leads_test_prep`

```
class_preds <- predict(logistic_fit,
                       new_data = leads_test_prep,
                       type = 'class')

prob_preds <- predict(logistic_fit,
                      new_data = leads_test_prep,
                      type = 'prob')
```

Combining prediction results

Combine predictions into a results dataset for `yardstick` metric functions

- Select the actual outcome variable, `purchased` from the test dataset
- Bind the predictions with `bind_cols()`

```
leads_results <- leads_test %>%
  select(purchased) %>%
  bind_cols(class_preds, prob_preds)

leads_results
```

```
# A tibble: 332 x 4
   purchased .pred_class .pred_yes .pred_no
   <fct>     <fct>           <dbl>    <dbl>
 1 no        no             0.257     0.743
 2 yes       yes            0.896     0.104
 3 no        no             0.0852    0.915
 4 no        no             0.183     0.817
 5 yes       yes            0.776     0.224
# ... with 327 more rows
```

Model evaluation

Evaluate model performance with `yardstick`

- The results data can be used with all `yardstick` metric functions for model evaluation
- Confusion matrix, sensitivity, specificity, and other metrics

```
leads_results %>%
  conf_mat(truth = purchased,
           estimate = .pred_class)
```

```
          Truth
Prediction yes  no
       yes  77  34
       no   43 178
```

Let's practice!

MODELING WITH TIDYMODELS IN R