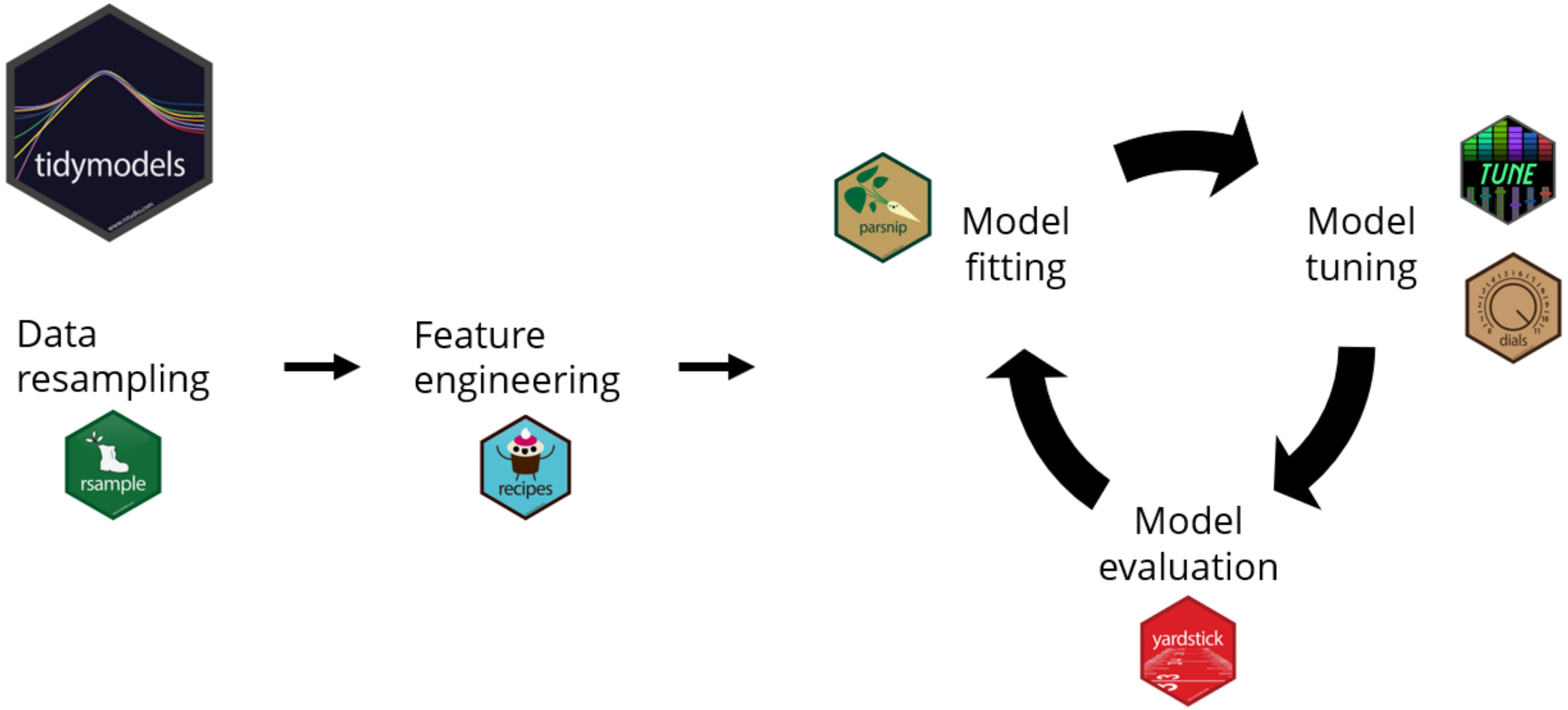
# The tidymodels ecosystem

MODELING WITH TIDYMODELS IN R

**David Svancer**
Data Scientist

datacamp

# Collection of machine learning packages



# Supervised machine learning

Branch of machine learning that uses labeled data for model fitting

**Regression**

- Predicting **quantitative** outcomes
  - Selling price of a home

**Classification**

- Predicting **categorical** outcomes
  - Whether an employee will leave a company

| left_company | miles_from_home | salary |
|---|---|---|
| no | 1 | 84500 |
| yes | 10 | 64820 |
| no | 5 | 76490 |
| yes | 19 | 68540 |

`tidymodels` **variable roles**

- *left_company* is an outcome variable
- *miles_from_home* and *salary* are predictor variables

# Data resampling with tidymodels

- `initial_split()`
  - Specifies instructions for creating training and test datasets
  - `prop` specifies the proportion to place into training
  - `strata` provides stratification by the outcome variable. Stratifying by the **outcome variable** ensures the model fitting process is performed on a representative sample of the original data.

```
library(tidymodels)
```

```
mpg_split <- initial_split(mpg,
                           prop = 0.75,
                           strata = hwy)
```

```
mpg_training <- mpg_split %>%
  training()
```

```
mpg_test <- mpg_split %>%
  testing()
```

# Linear regression with tidymodels

MODELING WITH TIDYMODELS IN R

**David Svancer**
Data Scientist

datacamp

# Linear regression model

Predicting `hwy` using `cty` as a predictor

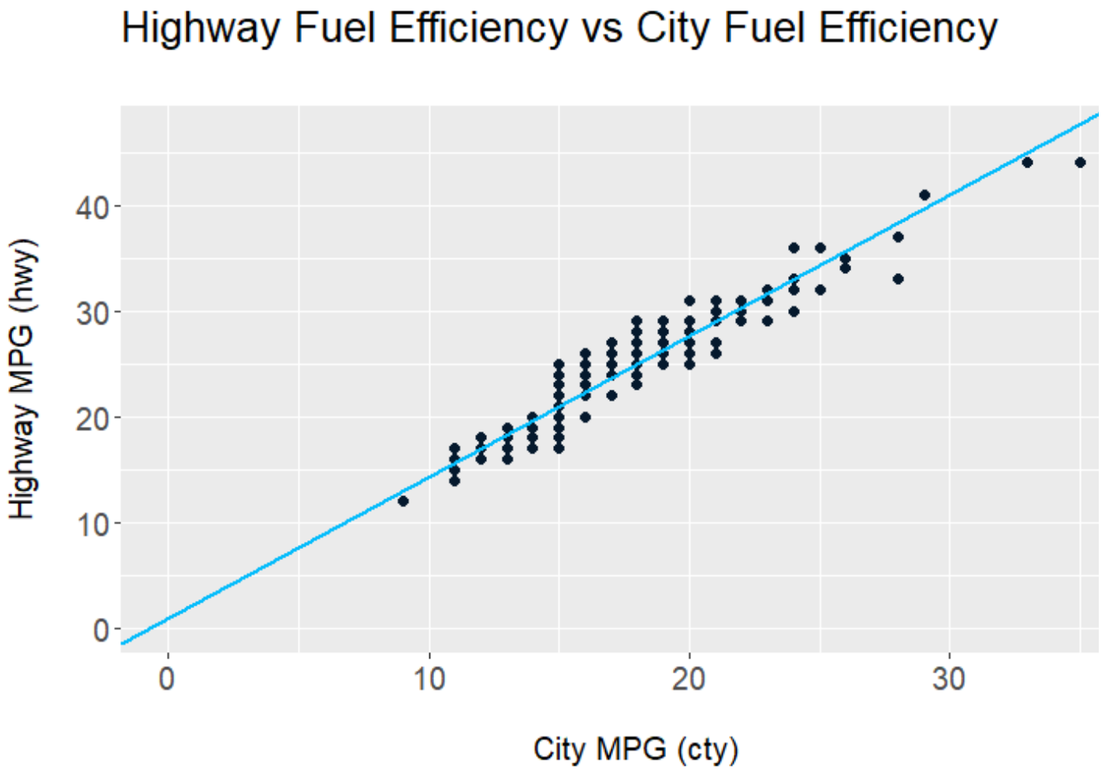
$$hwy = \beta_0 + \beta_1 cty$$

Model parameters

- $\beta_0$ is the intercept
- $\beta_1$ is the slope

Estimated paramters from training data

$$hwy = 0.77 + 1.35(cty)$$



# Model formulas

Model formulas in `parsnip`

- Used to assign column roles
  - Outcome variable
  - Predictor variables

General form

```
outcome ~ predictor_1 + predictor_2 + ...
```

Shorthand notation

```
outcome ~ .
```

Predicting `hwy` using `cty` as a predictor variable

```
hwy ~ cty
```

# Fitting a linear regression model

Define model specification with `parsnip`

- `linear_reg()`

```
lm_model <- linear_reg() %>%
  set_engine('lm') %>%
  set_mode('regression')
```

Pass `lm_model` to the `fit()` function

- Specify model formula
- `data` to use for model fitting

```
lm_fit <- lm_model %>%
  fit(hwy ~ cty, data = mpg_training)
```

# Obtaining the estimated parameters

The `tidy()` function

- Takes a trained `parsnip` model object
- Creates a model summary tibble
- `term` and `estimate` column provide estimated parameters

```
tidy(lm_fit)
```

```
# A tibble: 2 x 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)    0.769    0.528       1.46 1.47e- 1
2 cty            1.35     0.0305     44.2  6.32e-97
```

# Making predictions

```
hwy_predictions <- lm_fit %>%
  predict(new_data = mpg_test)

hwy_predictions
```

```
# A tibble: 57 x 1
   .pred
   <dbl>
 1  25.0
 2  27.7
 3  25.0
 4  25.0
 5  22.3
# ... with 47 more rows
```

```
mpg_test_results <- mpg_test %>%
  select(hwy, cty) %>%
  bind_cols(hwy_predictions)

mpg_test_results
```

```
# A tibble: 57 x 3
     hwy   cty .pred
   <int> <int> <dbl>
 1    29    18  25.0
 2    31    20  27.7
 3    27    18  25.0
 4    26    18  25.0
 5    25    16  22.3
# ... with 47 more rows
```

# Adding predictions to the test data

The `bind_cols()` function

- Combines two or more tibbles along the column axis
- Useful for creating a model results tibble

**Steps**

- Select `hwy` and `cty` from `mpg_test`
- Pass to `bind_cols()` and add predictions column

```
mpg_test_results <- mpg_test %>%
  select(hwy, cty) %>%
  bind_cols(hwy_predictions)

mpg_test_results
```

```
# A tibble: 57 x 3
     hwy   cty .pred
   <int> <int> <dbl>
 1    29    18  25.0
 2    31    20  27.7
 3    27    18  25.0
 4    26    18  25.0
 5    25    16  22.3
# ... with 47 more rows
```

# Evaluating model performance

MODELING WITH TIDYMODELS IN R

**David Svancer**
Data Scientist

datacamp

# Input to yardstick functions

All `yardstick` functions require a tibble with model results

- Column with the true outcome variable values
  - `hwy` for mpg data
- Column with model predictions
  - `.pred`

```
mpg_test_results
```

```
# A tibble: 57 x 3
     hwy   cty .pred
   <int> <int> <dbl>
 1    29    18  25.0
 2    31    20  27.7
 3    27    18  25.0
 4    26    18  25.0
 5    25    16  22.3
# ... with 47 more rows
```

# Root mean squared error (RMSE)

RMSE estimates the average prediction error

- Calculated with the `rmse()` function from `yardstick`
  - Takes a tibble with model results
  - `truth` is the column with true outcome values
  - `estimate` is the column with predicted outcome values

```
mpg_test_results %>%
  rmse(truth = hwy, estimate = .pred)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard        1.93
```

# R squared metric

Measures the squared correlation between actual and predicted values

- Also called the **coefficient of determination**
- Ranges from 0 to 1
  - When all predictions equal the true outcome values, R squared is 1
- Calculated with the `rsq()` function from `yardstick`

```
mpg_test_results %>%
  rsq(truth = hwy, estimate = .pred)
```

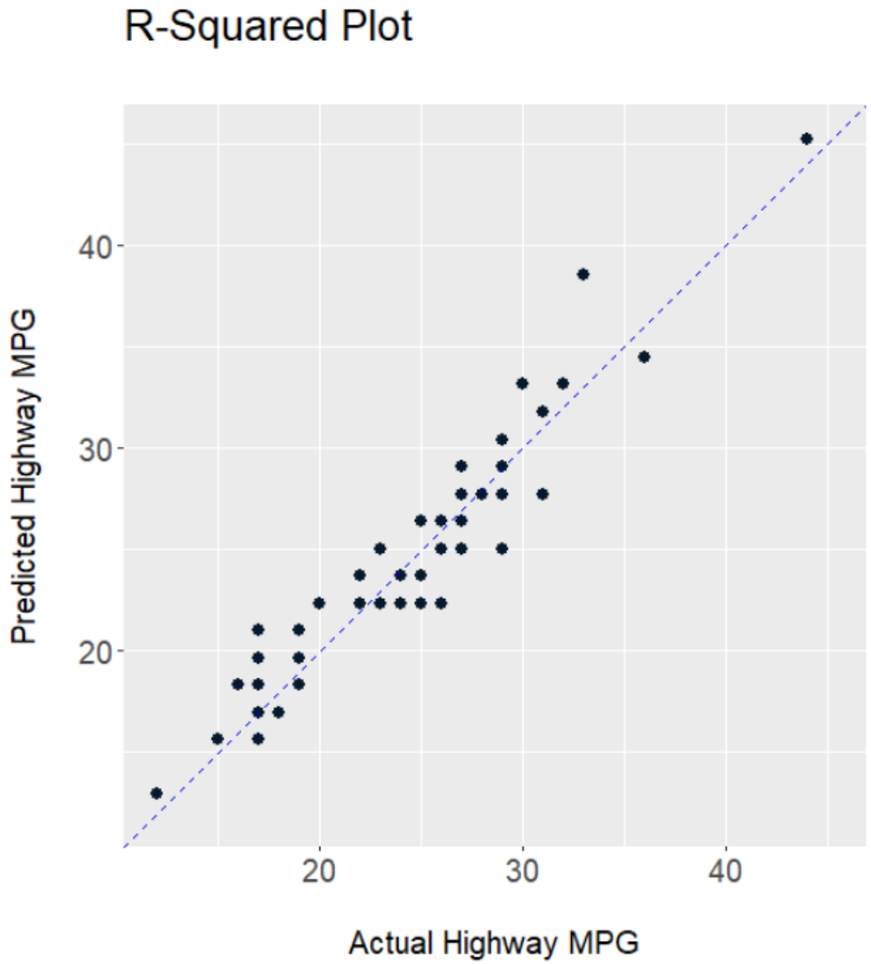
```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rsq     standard       0.904
```

# Plotting R squared plots

Making R squared plots with `ggplot2`

- Tibble of model results
- `geom_point()`
- `geom_abline()`
- `coord_obs_pred()`

```
ggplot(mpg_test_results, aes(x = hwy, y = .pred)) +
  geom_point() +
  geom_abline(color = 'blue', linetype = 2) +
  coord_obs_pred() +
  labs(title = 'R-Squared Plot',
       y = 'Predicted Highway MPG',
       x = 'Actual Highway MPG')
```



# Streamlining model fitting

The `last_fit()` function

- Takes a model specification, model formula, and data split object
- Performs the following:
  1. Creates training and test datasets
  2. Fits the model to the training data
  3. Calculates metrics and predictions on the test data
  4. Returns an object with all results

```
lm_last_fit <- lm_model %>%
  last_fit(hwy ~ cty,
           split = mpg_split)
```

```
lm_last_fit %>%
  collect_metrics()
```

```
# A tibble: 2 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard        1.93
2 rsq     standard        0.904
```

# Collecting predictions

The `collect_predictions()` function

- Takes the results of `last_fit()`
  - Returns a tibble with test dataset predictions
  - Predictions column is named `.pred`
  - Outcome variable and other row identifier columns included

```
lm_last_fit %>%
  collect_predictions()
```

```
# A tibble: 57 x 4
   id                .pred  .row   hwy
   <chr>             <dbl> <int> <int>
 1 train/test split  25.0      1    29
 2 train/test split  27.7      3    31
 3 train/test split  25.0      7    27
 4 train/test split  25.0      8    26
 5 train/test split  22.3      9    25
# ... with 47 more rows
```

# Classification models

MODELING WITH TIDYMODELS IN R

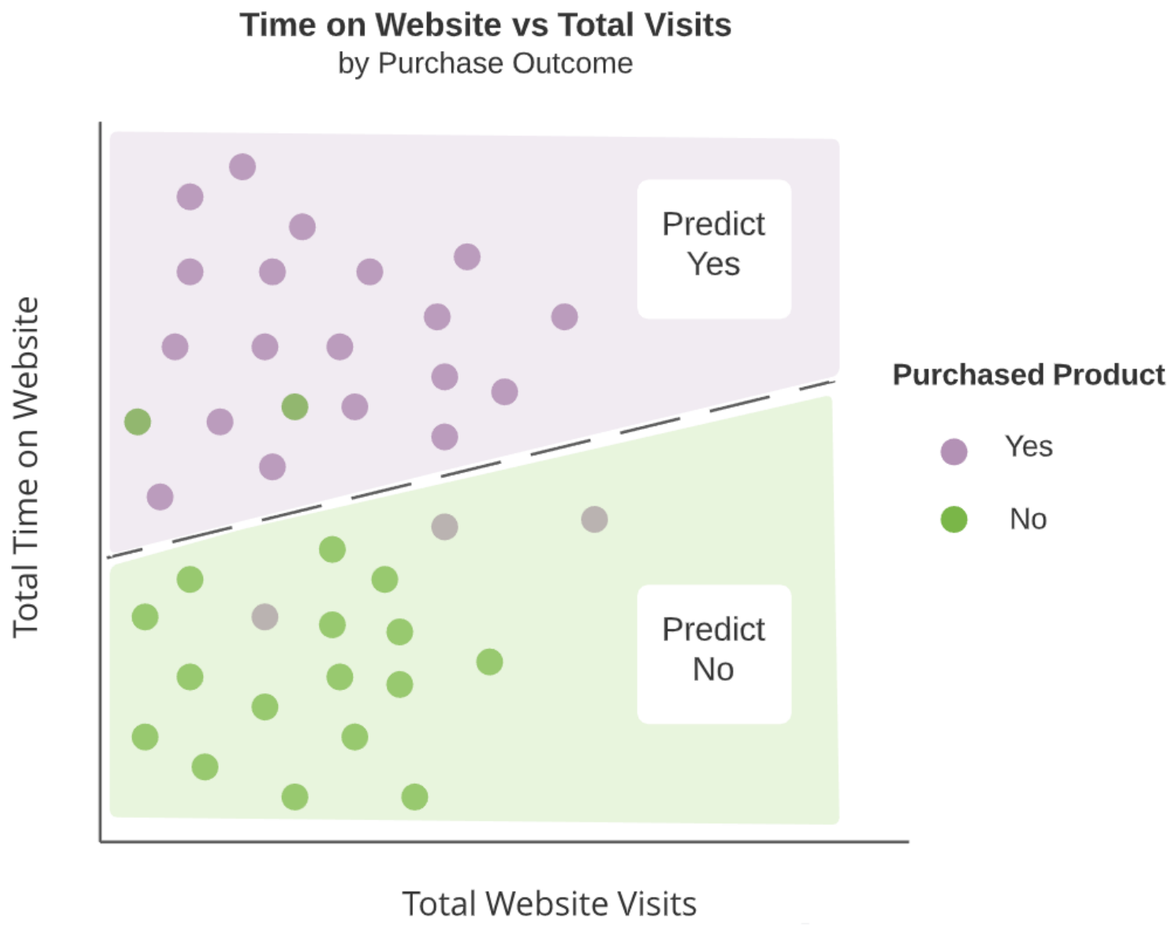
**David Svancer**
Data Scientist

datacamp

# Classification algorithms

**Goal**: Create distinct, non-overlapping regions along set of predictor variable values

- Predict the same categorical outcome in each region




## Logistic Regression

- Popular classification algorithm which creates a *linear* separation between outcome categories

# Logistic regression model specification

Model specification in `parsnip`

- `logistic_reg()`
  - General interface to logistic regression models in `parsnip`
  - Common engine is 'glm'
  - Mode is 'classification'

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

```
logistic_fit <- logistic_model %>%
  fit(purchased ~ total_visits + total_time,
      data = leads_training)
```

# Predicting outcome categories

```
class_preds <- logistic_fit %>%
  predict(new_data = leads_test,
          type = 'class')

class_preds
```

```
# A tibble: 332 x 1
   .pred_class
   <fct>
 1 no
 2 yes
 3 no
 4 no
 5 yes
 # ... with 327 more rows
```

```
prob_preds <- logistic_fit %>%
  predict(new_data = leads_test,
          type = 'prob')

prob_preds
```

```
# A tibble: 332 x 2
   .pred_yes .pred_no
       <dbl>    <dbl>
 1    0.134     0.866
 2    0.729     0.271
 3    0.133     0.867
 4    0.0916    0.908
 5    0.598     0.402
# ... with 327 more rows
```

# Combining results

For model evaluation with the `yardstick` package, a results tibble will be needed

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

The outcome variable from the test dataset and prediction tibbles can be combined with `bind_cols()`

```
leads_results
```

```
# A tibble: 332 x 4
   purchased .pred_class .pred_yes .pred_no
   <fct>     <fct>           <dbl>    <dbl>
 1 no        no             0.134     0.866
 2 yes       yes            0.729     0.271
 3 no        no             0.133     0.867
 4 no        no             0.0916    0.908
 5 yes       yes            0.598     0.402
# ... with 327 more rows
```

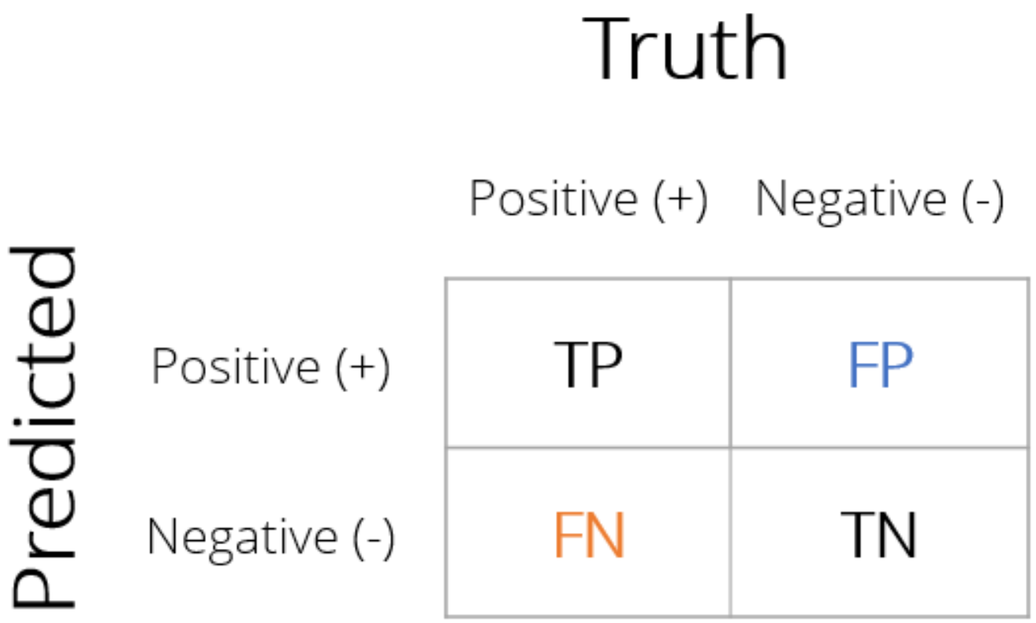
# Binary classification

- In `tidymodels` outcome variable needs to be a factor
  - **First level is positive class**
  - Check order with `levels()`

```
levels(leads_df['purchased'])
```

```
[1] "yes" "no"
```

# Confusion matrix with yardstick




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

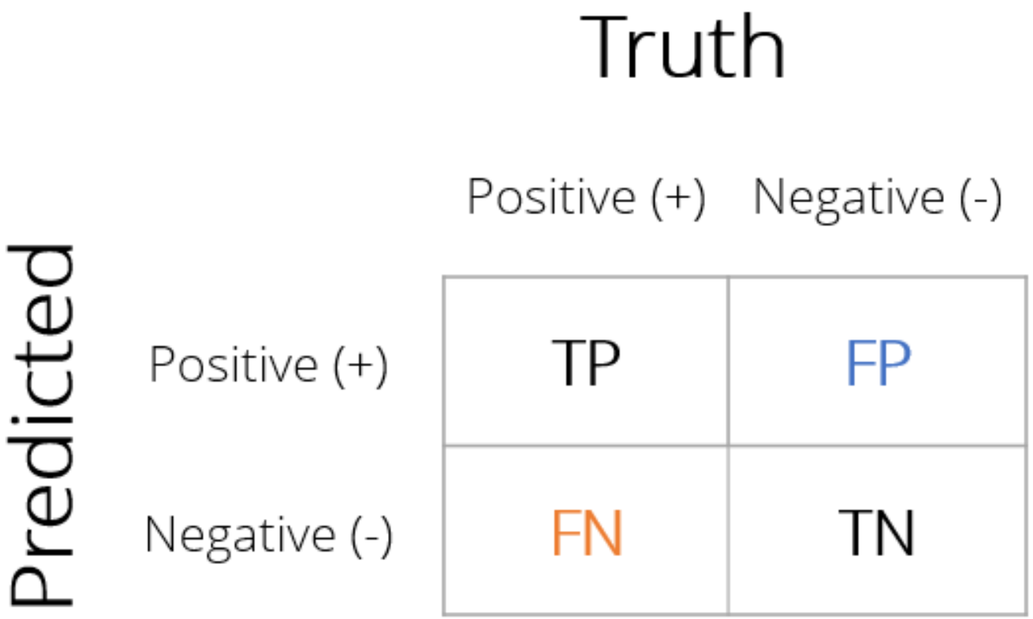
```
          Truth
Prediction yes  no
       yes  74  34
       no   46 178
```

# Classification accuracy

$$\frac{TP + TN}{TP + TN + FP + FN}$$

```
accuracy(leads_results,
         truth = purchased,
         estimate = .pred_class)
```

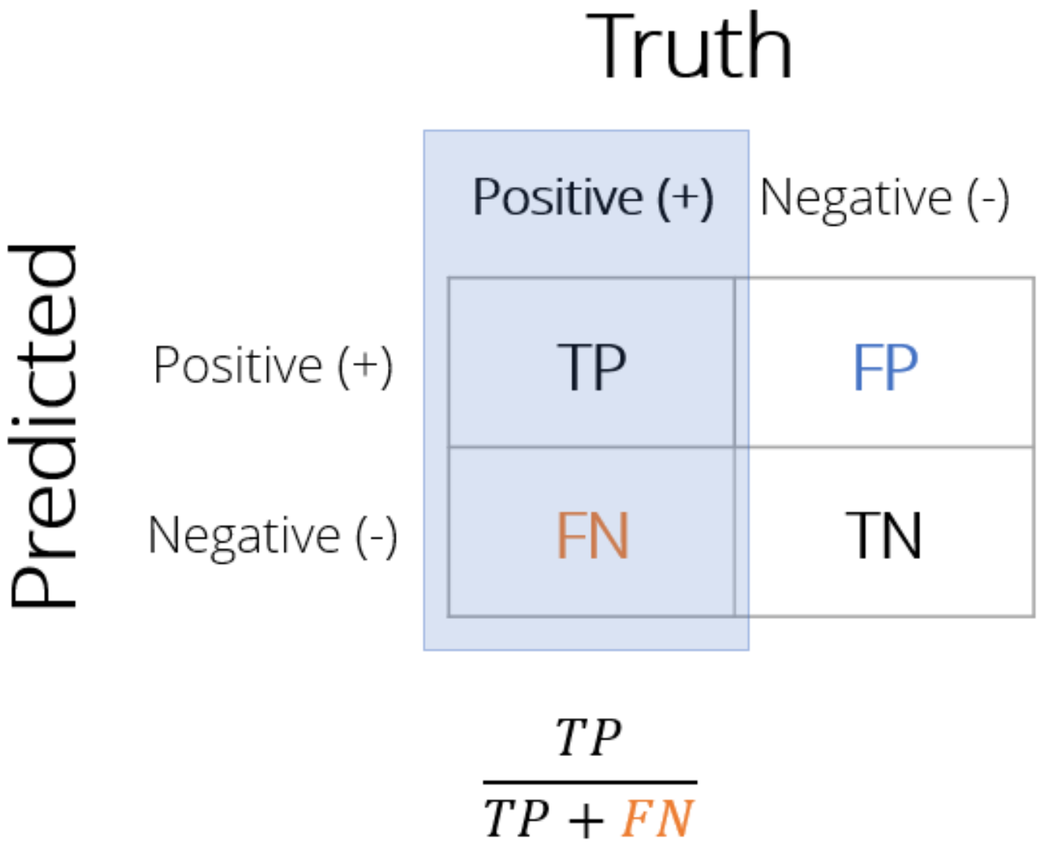
```
# A tibble: 1 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.759
```



# Sensitivity

```
sens(leads_results,
     truth = purchased,
     estimate = .pred_class)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 sens    binary         0.617
```

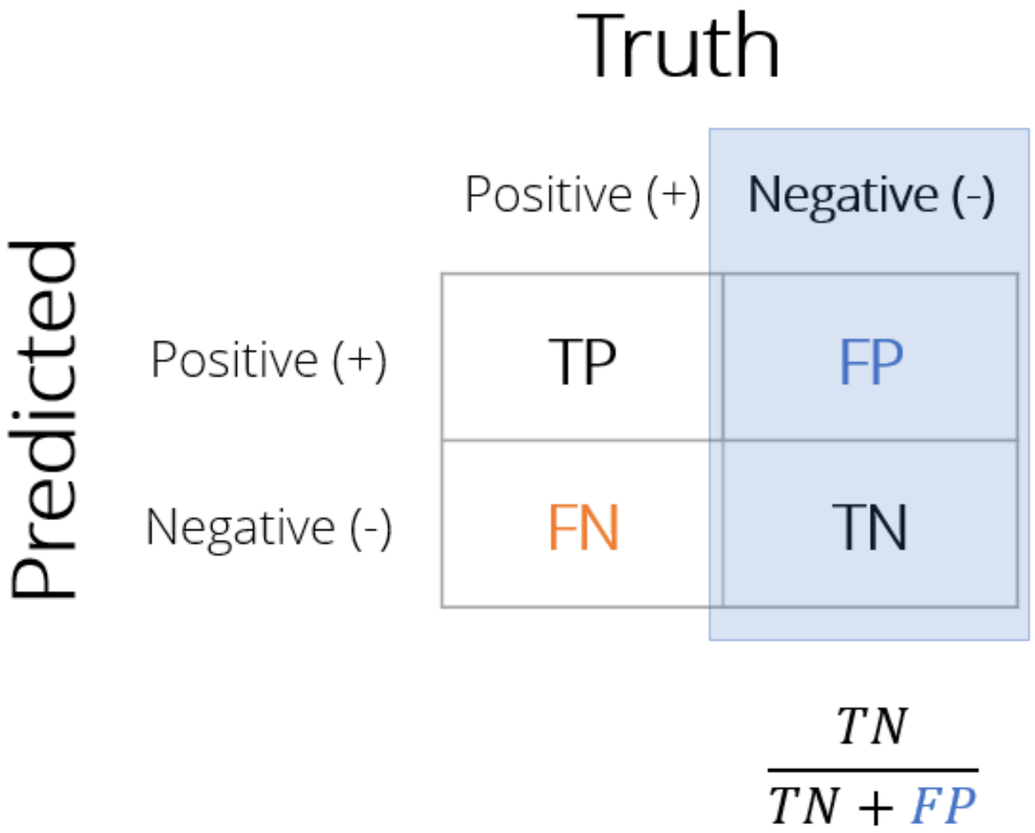



$$\frac{TP}{TP + FN}$$

# Specificity

```
spec(leads_results,
     truth = purchased,
     estimate = .pred_class)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 spec    binary         0.840
```




$$\frac{TN}{TN + FP}$$

# Many metrics

## Binary classification metrics

```
custom_metrics <-
  metric_set(accuracy, sens, spec)
```

```
custom_metrics(leads_results,
               truth = purchased,
               estimate = .pred_class)
```

```
# A tibble: 3 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.759
2 sens     binary         0.617
3 spec     binary         0.840
```

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

```
# A tibble: 13 x 3
   .metric              .estimator .estimate
   <chr>                <chr>          <dbl>
 1 accuracy             binary         0.759
 2 kap                  binary         0.466
 3 sens                 binary         0.617
 4 spec                 binary         0.840
 5 ppv                  binary         0.685
 6 npv                  binary         0.795
 7 mcc                  binary         0.468
 8 j_index              binary         0.456
 9 bal_accuracy         binary         0.728
10 detection_prevalence binary         0.325
11 precision            binary         0.685
12 recall               binary         0.617
13 f_meas               binary         0.649
```

# Custom metric sets

The `metric_set()` function

- `accuracy()`, `sens()`, and `spec()`
  - Require `truth` and `estimate` arguments
- `roc_auc()`
  - Requires `truth` and column of estimated probabilities

The `custom_metrics()` function will need all three, with `.pred_yes` as the last argument

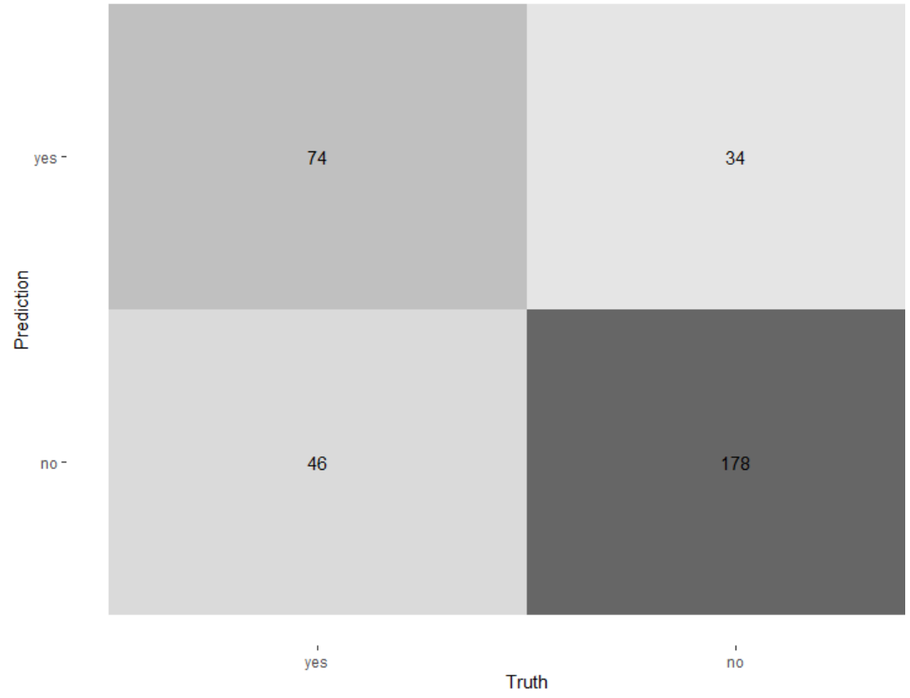
```
custom_metrics <- metric_set(accuracy, sens,
                             spec, roc_auc)
```

```
custom_metrics(last_fit_results,
               truth = purchased,
               estimate = .pred_class,
               .pred_yes)
```

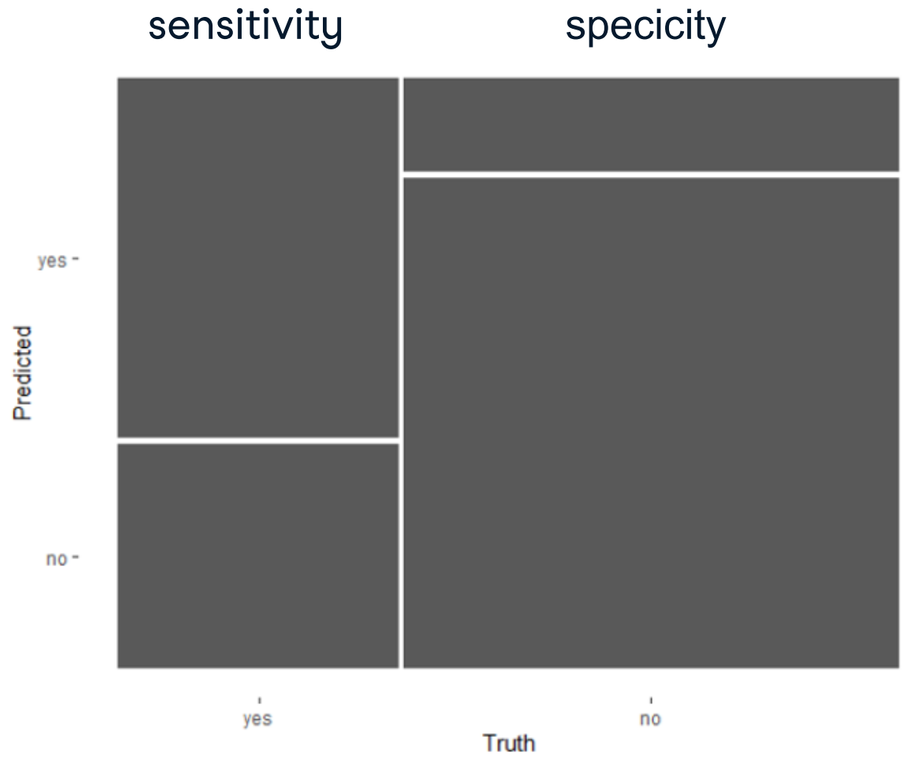
```
# A tibble: 4 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.759
2 sens     binary         0.617
3 spec     binary         0.840
4 roc_auc  binary         0.763
```

# Plotting the confusion matrix

```
conf_mat(leads_results,
         truth = purchased,
         estimate = .pred_class) %>%
  autoplot(type = 'heatmap')
```




```
conf_mat(leads_results,
         truth = purchased,
         estimate = .pred_class) %>%
  autoplot(type = 'mosaic')
```



# Exploring performance across thresholds

How does a classification model perform across a range of thresholds?

- Unique probability thresholds in the `.pred_yes` column of the test dataset results
  - Calculate specificity and sensitivity for each

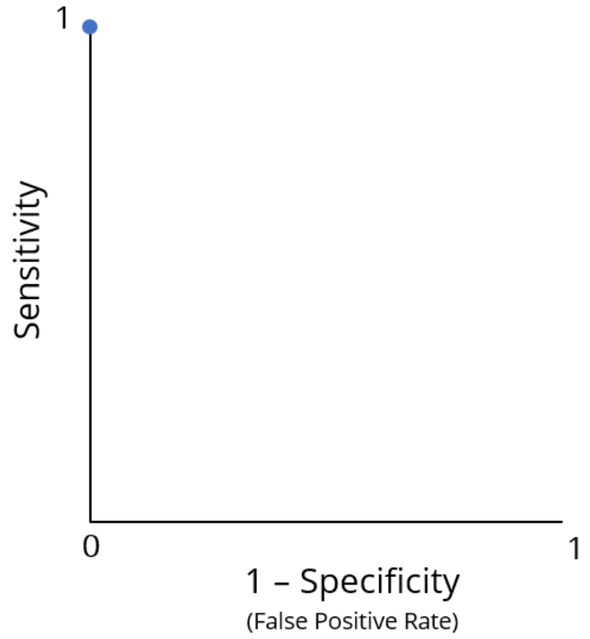
```
leads_results %>%
  roc_curve(truth = purchased, .pred_yes)
```

| threshold | specificity | sensitivity |
|---|---|---|
| 0 | 0 | 1 |
| 0.11 | 0.01 | 0.98 |
| 0.15 | 0.05 | 0.97 |
| ... | ... | ... |
| 0.84 | 0.89 | 0.08 |
| 0.87 | 0.94 | 0.02 |
| 0.91 | 0.99 | 0 |
| 1 | 1 | 0 |

# ROC curves

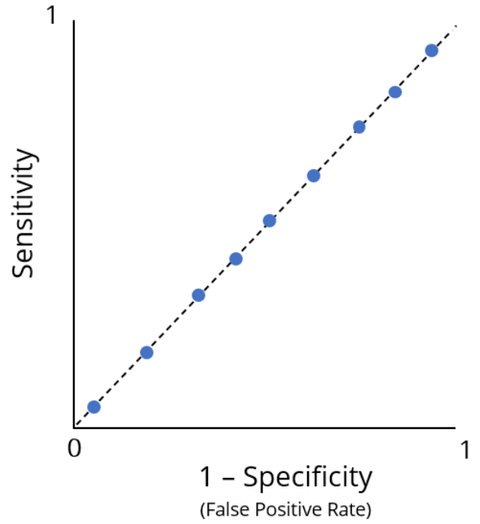
**Optimal performance** is at the point (0, 1)

- Ideally, a classification model produces points close to left upper edge across all thresholds




**Poor performance**

- Sensitivity and (1 - specificity) are equal across all thresholds
  - Corresponds to a classification model that predicts outcomes based on the result of randomly flipping a fair coin

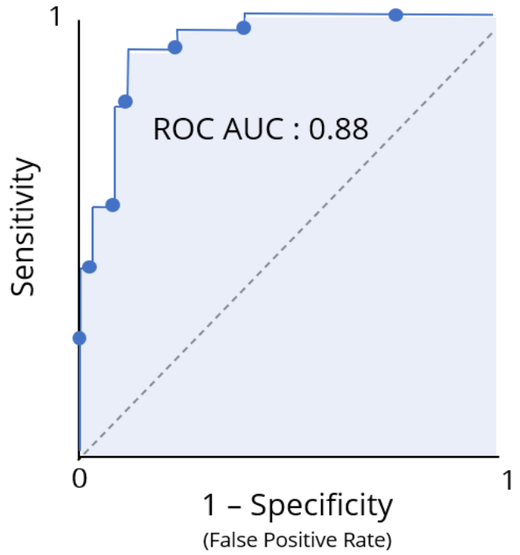


# Summarizing the ROC curve

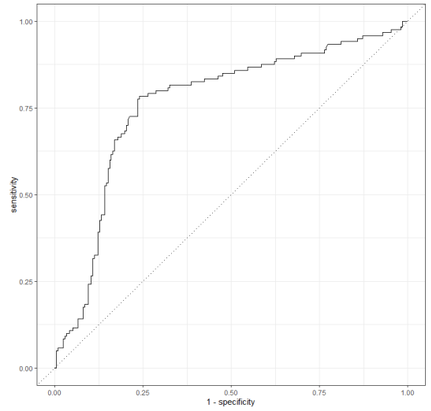
The area under the ROC curve (ROC AUC) captures the ROC curve information of a classification model in a single number

Useful interpretation as a letter grade of classification performance

- A - [0.9, 1]
- B - [0.8, 0.9)
- C - [0.7, 0.8)
- D - [0.6, 0.7)
- F - [0.5, 0.6)




```
leads_results %>%
  roc_curve(truth = purchased, .pred_yes) %>%
  autoplot()
```




```
roc_auc(leads_results,
        truth = purchased,
        .pred_yes)
```

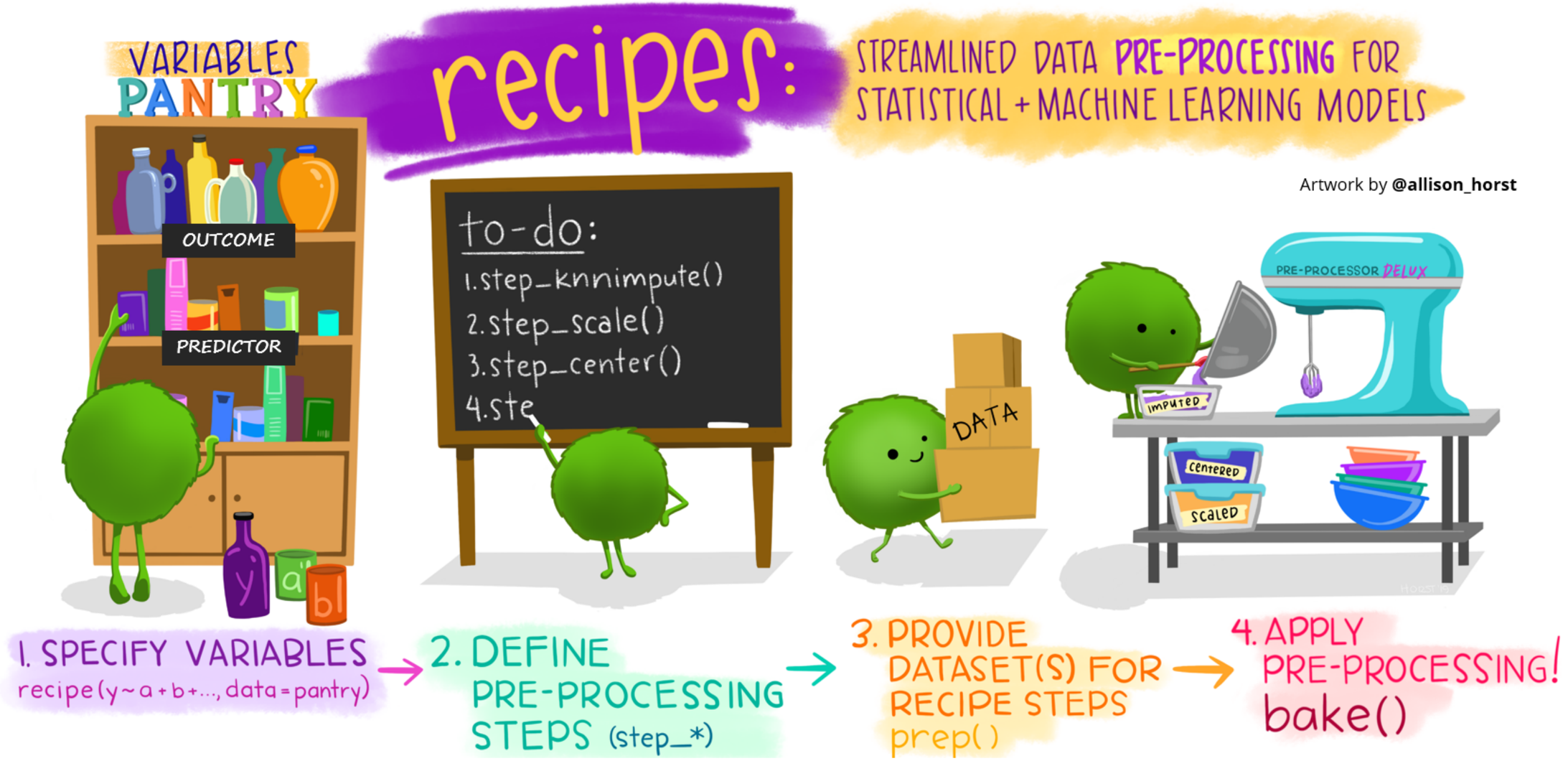
# Feature engineering

## MODELING WITH TIDYMODELS IN R

**David Svancer**
Data Scientist

datacamp

# Feature engineering with the recipes package



# 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 arestored for applying to new data

Recipes are trained with the `prep()` function

# 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


Recipes are applied with the `bake()` function

# 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

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

- 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

```
# 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
```

# 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()
```

# 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 |
| ... |

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

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

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

leads_norm_rec
```

# 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 |

# 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
```

# 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
```

# Machine learning workflows

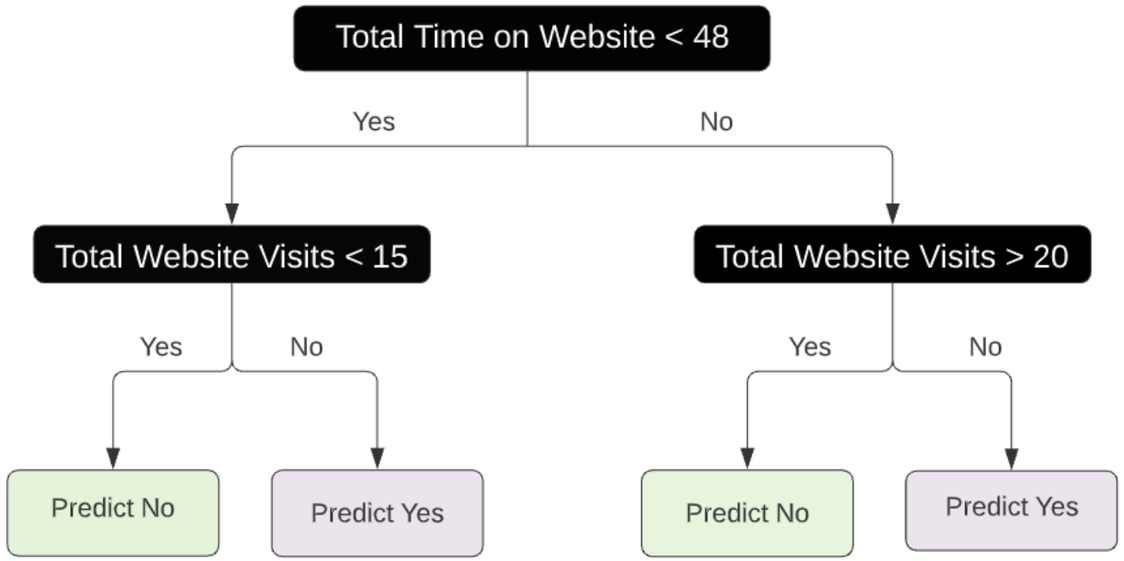
## MODELING WITH TIDYMODELS IN R

**David Svancer**
Data Scientist

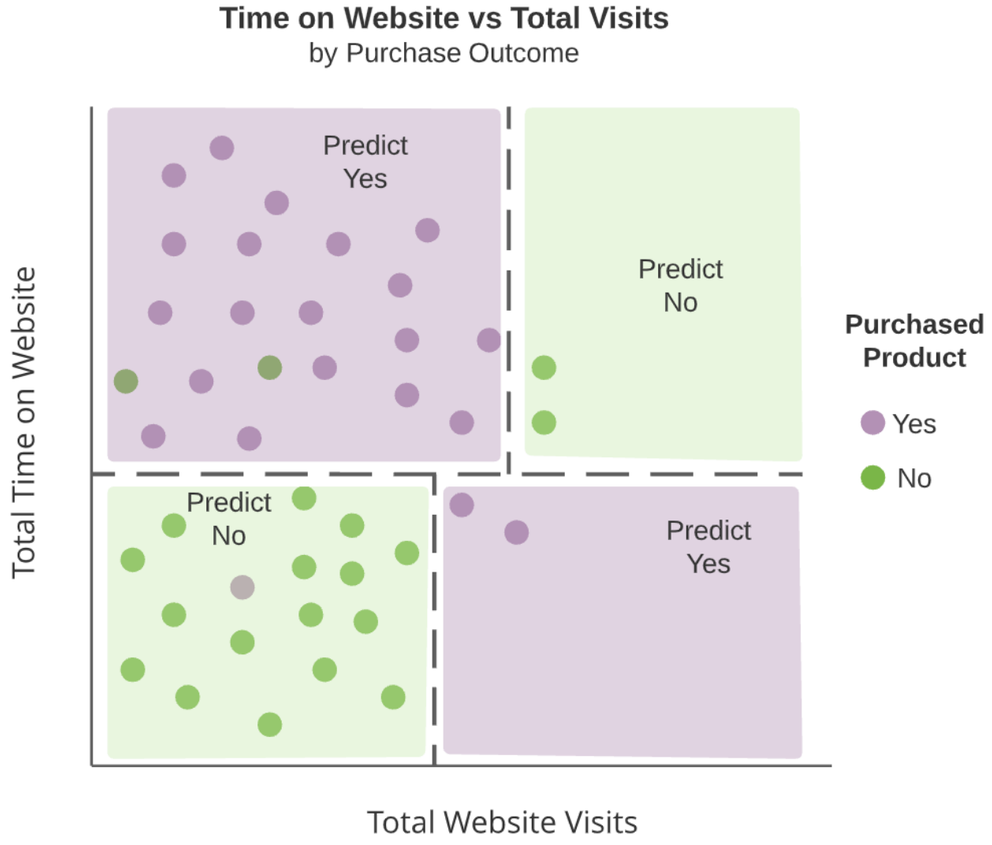
datacamp

# Tree diagrams

- **Interior nodes**
  - Decision tree splits (dark boxes)
- **Terminal nodes**
  - Regions which are not split further
  - Green and purple boxes




Interior nodes are dashed lines and terminal nodes are highlighted rectangular regions



# Model specification

Model specification in `parsnip`

- `decision_tree()`
  - General interface to decision tree models in `parsnip`
  - Common engine is `'rpart'`
  - Mode can be either `'classification'` or `'regression'`
    - For lead scoring data, we need `'classification'`

```
dt_model <- decision_tree() %>%
  set_engine('rpart') %>%
  set_mode('classification')
```

# Feature engineering recipe

Data transformations for lead scoring data

- Encoded in a `recipe` object
  - Remove multicollinearity
  - Normalize numeric predictors
  - Create dummy variables for nominal predictors

Two R objects to manage

- `parsnip` model and `recipe` specification
- Combining into one object would make life easier

```
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()
```

# Combining models and recipes

The `workflows` package is designed for streamlining the model process

- Combines a `parsnip` model and `recipe` object into a single `workflow` object

Initialized with the `workflow()` function

- Add model object with `add_model()`
- Add `recipe` object with `add_recipe()`
  - Must be specification, not a trained `recipe`

```
leads_wkfl <- workflow() %>%
  add_model(dt_model) %>%
  add_recipe(leads_recipe)

leads_wkfl
```

```
== Workflow ====================
Preprocessor: Recipe
Model: decision_tree()
-- Preprocessor ----------------
3 Recipe Steps
* step_corr()
* step_normalize()
* step_dummy()
-- Model -----------------------
Decision Tree Model Specification (classification)
Computational engine: rpart
```

# Model fitting with workflows

Training a `workflow` object

- Pass `workflow` to `last_fit()` and provide data split object
- View model evaluation results with `collect_metrics()`

**Behind the scenes**

- Training and test datasets created
- `recipe` trained and applied
- Decision tree trained with training data
- Predictions and metrics on test data

```
leads_wkfl_fit <- leads_wkfl %>%
  last_fit(split = leads_split)

leads_wkfl_fit %>%
  collect_metrics()
```

```
# A tibble: 2 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.771
2 roc_auc  binary         0.775
```

# Collecting predictions

A `workflow` trained with `last_fit()` can be passed to `collect_predictions()`

- Produces detailed results on the test data
- Like before, can be used with `yardstick` functions to explore performance custom metrics

```
leads_wkfl_preds <- leads_wkfl_fit %>%
  collect_predictions()

leads_wkfl_preds
```

```
# A tibble: 332 x 6
   id          .pred_yes .pred_no  .row .pred_class purchased
  <chr>           <dbl>   <dbl>    <int>   <fct>       <fct>
train/test split  0.120    0.880     2      no          no
train/test split  0.755    0.245    17      yes         yes
train/test split  0.120    0.880    21      no          no
train/test split  0.120    0.880    22      no          no
train/test split  0.755    0.245    24      yes         yes
# ... with 327 more rows
```

# Exploring custom metrics

Create a custom metric set with `metric_set()`

- Area under the ROC curve, sensitivity, and specificity

```
leads_metrics <- metric_set(roc_auc, sens, spec)

leads_wkfl_preds %>%
  leads_metrics(truth = purchased,
                estimate = .pred_class,
                .pred_yes)
```

Pass predictions datasets to `leads_metrics()` to calculate metrics

```
# A tibble: 3 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 sens    binary         0.75
2 spec    binary         0.783
3 roc_auc binary         0.775
```

# Estimating performance with cross validation

MODELING WITH TIDYMODELS IN R

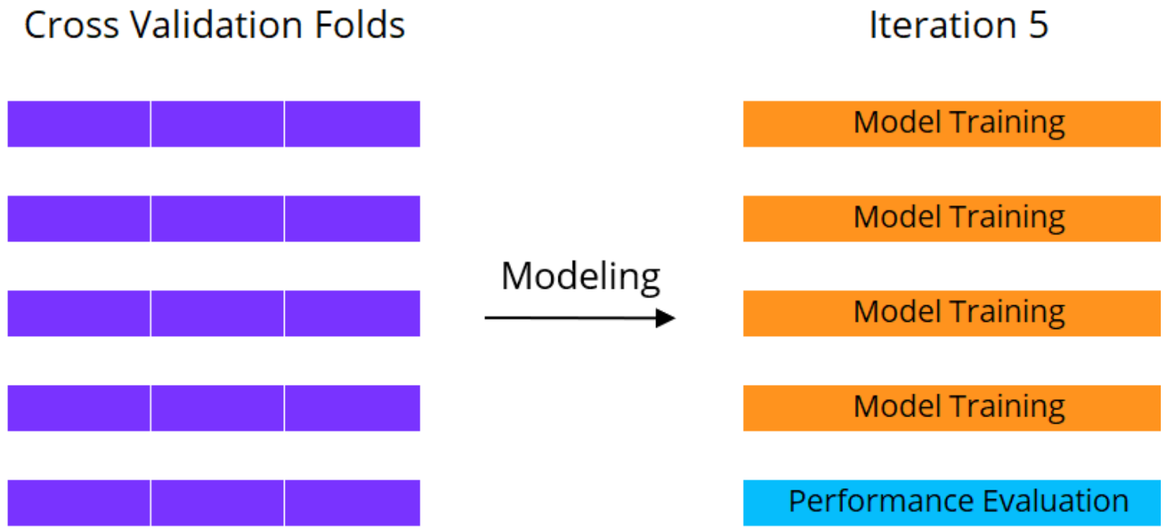
**David Svancer**
Data Scientist

datacamp

# Machine learning with cross validation

Performing 5-fold cross validation

- Five iterations of model training and evaluation
- Iteration 1
  - Fold 1 reserved for model evaluation and folds 2 through 5 for model training
- Iteration 2
  - Fold 2 reserved for model evaluation




**Five estimates** of model performance in total

# Creating cross validation folds

The `vfold_cv()` function

- Training data
- Number of folds, `v`
- Stratification variable, `strata`
- Execute `set.seed()` before `vfold_cv()` for reproducibility
- `splits`
  - **List column** with data split objects for creating fold

```
set.seed(214)
leads_folds <- vfold_cv(leads_training,
                        v = 10,
                        strata = purchased)
leads_folds
```

```
#  10-fold cross-validation using stratification
# A tibble: 10 x 2
   splits             id
   <list>             <chr>
 1 <split [896/100]> Fold01
 2 <split [896/100]> Fold02
 3 <split [896/100]> Fold03
 . ...............   ......
 9 <split [897/99]>  Fold09
10 <split [897/99]>  Fold10
```

# Model training with cross validation

The `fit_resamples()` function

- Train a `parsnip` model or `workflow` object
- Provide cross validation folds, `resamples`
- Optional custom metric function, `metrics`
  - Default is accuracy and ROC AUC

Each metric is estimated 10 times

- One estimate per fold
- Average value in `mean` column

```
leads_rs_fit <- leads_wkfl %>%
  fit_resamples(resamples = leads_folds,
                metrics = leads_metrics)

leads_rs_fit %>%
  collect_metrics()
```

```
# A tibble: 3 x 5
  .metric .estimator  mean     n std_err
  <chr>   <chr>      <dbl> <int>   <dbl>
1 roc_auc binary     0.823    10  0.0147
2 sens    binary     0.786    10  0.0203
3 spec    binary     0.855    10  0.0159
```

# Detailed cross validation results

The `collect_metrics()` function

- Passing `summarize = FALSE` will provide all metric estimates for every cross validation fold
- 30 total combinations (3 metrics x 10 folds)
  - `.metric` column identifies metric
  - `.estimate` column gives estimated value for each fold

```
rs_metrics <- leads_rs_fit %>%
  collect_metrics(summarize = FALSE)

rs_metrics
```

```
# A tibble: 30 x 4
   id     .metric .estimator .estimate
   <chr>  <chr>   <chr>          <dbl>
 1 Fold01 sens    binary         0.861
 2 Fold01 spec    binary         0.891
 3 Fold01 roc_auc binary         0.885
 4 Fold02 sens    binary         0.778
 5 Fold02 spec    binary         0.969
 6 Fold02 roc_auc binary         0.885
# ... with 24 more rows
```

# Hyperparameter tuning

MODELING WITH TIDYMODELS IN R

**David Svancer**
Data Scientist

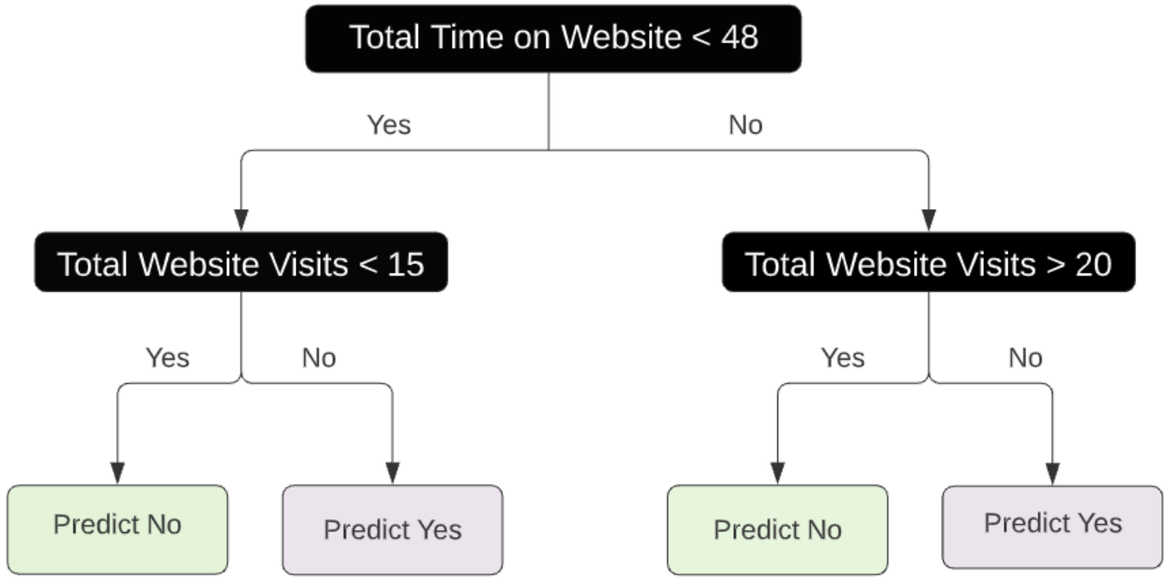
datacamp

# Hyperparameters

Model parameters whose values are set prior to model training and control model complexity

`parsnip` **decision tree**

- `cost_complexity`
  - Penalizes large number of terminal nodes
- `tree_depth`
  - Longest path from root to terminal node
- `min_n`
  - Minimum data points required in a node for further splitting



# Default hyperparameter values

`decision_tree()` function sets default hyperparameter values

- `cost_complexity` is set to 0.01
- `tree_depth` is set to 30
- `min_n` is set to 20

These may not be the best values for all datasets

- **Hyperparameter tuning**
  - Process of using cross validation to find the optimal set of hyperparameter values

```
dt_model <- decision_tree() %>%
  set_engine('rpart') %>%
  set_mode('classification')
```

```
dt_tune_model <- decision_tree(cost_complexity = tune(),
                               tree_depth = tune(),
                               min_n = tune()) %>%
  set_engine('rpart') %>%
  set_mode('classification')

dt_tune_model
```

# Creating a tuning workflow

`workflow` objects can be easily updated

- Prior `leads_wkfl`
  - Feature engineering steps for lead scoring data and decision tree model with default hyperparameters
- Pass `leads_wkfl` to `update_model()` and provide new decision tree model with tuning parameters

```
leads_tune_wkfl <- leads_wkfl %>%
  update_model(dt_tune_model)

leads_tune_wkfl
```

```
== Workflow ===============
Preprocessor: Recipe
Model: decision_tree()
-- Preprocessor -----------
3 Recipe Steps
* step_corr()
* step_normalize()
* step_dummy()
-- Model ------------------
Decision Tree Model Specification (classification)
Main Arguments: cost_complexity = tune()
                tree_depth = tune()
                min_n = tune()
Computational engine: rpart
```

# Saving a tuning grid

First step in hyperparameter tuning

- Create and save a tuning grid
- `dt_grid` contains 5 random combinations of hyperparameter values

```
set.seed(214)
dt_grid <- grid_random(parameters(dt_tune_model),
                       size = 5)

dt_grid
```

```
# A tibble: 5 x 3
  cost_complexity  tree_depth min_n
             <dbl>      <int>  <int>
1    0.0000000758         14     39
2    0.0243                5     34
3    0.00000443           11      8
4    0.000000600           3      5
5    0.00380               5     36
```

# Hyperparameter tuning with cross validation

The `tune_grid()` function performs hyperparameter tuning

Takes the following arguments:

- `workflow` or `parsnip` model
- Cross validation object, `resamples`
- Tuning grid, `grid`
- Optional `metrics` function

Returns tibble of results

- `.metrics`
  - List column with results for each fold

```
dt_tuning <- leads_tune_wkfl %>%
             tune_grid(resamples = leads_folds,
                       grid = dt_grid,
                       metrics = leads_metrics)
```

```
dt_tuning
```

```
# Tuning results
# 10-fold cross-validation using stratification
# A tibble: 10 x 4
   splits             id       .metrics          ..
   <list>             <chr>    <list>            ..
<split [896/100]>   Fold01   <tibble [15 x 7]>  ..
.................   ......   ................    ..
<split [897/99]>    Fold09   <tibble [15 x 7]>  ..
<split [897/99]>    Fold10   <tibble [15 x 7]>  ..
```

# Exploring tuning results

The `collect_metrics()` function provides summarized results by default

- Average estimated metric values across all folds per combination

```
dt_tuning %>%
  collect_metrics()
```

```
# A tibble: 15 x 9
   cost_complexity tree_depth min_n .metric .estimator  mean     n std_err .config
             <dbl>      <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
 1    0.0000000758         14    39 roc_auc binary     0.827    10 0.0147  Model1
 2    0.0000000758         14    39 sens    binary     0.728    10 0.0277  Model1
 3    0.0000000758         14    39 spec    binary     0.865    10 0.0156  Model1
 4    0.0243                5    34 roc_auc binary     0.823    10 0.0147  Model2
 .    ......               ..    .. ....    ......     .....    .. .....   ......
14    0.00380               5    36 sens    binary     0.747    10 0.0209  Model5
15    0.00380               5    36 spec    binary     0.858    10 0.0161  Model5
```

# Exploring tuning results

Selecting `summarise = FALSE` within `collect_metrics()` returns a tibble

- Easy to explore results with `dplyr`
- Exploring ROC AUC
  - Select `roc_auc` metric
  - Form groups by `id` column
  - Calculate `.estimate` summary statistics

**Wild fluctuations between folds would be an indicator of model overfitting**

```
dt_tuning %>%
  collect_metrics(summarize = FALSE) %>%
  filter(.metric == 'roc_auc') %>%
  group_by(id) %>%
  summarize(min_roc_auc = min(.estimate),
            median_roc_auc = median(.estimate),
            max_roc_auc = max(.estimate))
```

```
# A tibble: 10 x 4
 id     min_roc_auc  median_roc_auc  max_roc_auc
<chr>       <dbl>           <dbl>        <dbl>
Fold01      0.830           0.885        0.888
Fold02      0.857           0.882        0.885
Fold03      0.818           0.836        0.836
......      ....            ....         ....
Fold10      0.762           0.790        0.813
```

# Viewing the best performing models

The `show_best()` function

- Displays the top `n` performing models based on average value of `metric`
- `Model1` is the winner

```
dt_tuning %>%
  show_best(metric = 'roc_auc', n = 5)
```

```
# A tibble: 5 x 9
cost_complexity  tree_depth  min_n  .metric .estimator   mean    n    std_err  .config
    <dbl>           <int>    <int>    <chr>   <chr>      <dbl>  <int>  <dbl>    <chr>
0.0000000758         14       39     roc_auc  binary     0.827   10   0.0147   Model1
0.00380               5       36     roc_auc  binary     0.825   10   0.0146   Model5
0.0243                5       34     roc_auc  binary     0.823   10   0.0147   Model2
0.00000443           11       8      roc_auc  binary     0.816   10   0.00786  Model3
0.000000600           3       5      roc_auc  binary     0.814   10   0.0131   Model4
```

# Selecting a model

The `select_best()` function

- Pass `dt_tuning` results to `select_best()`
- Select the `metric` on which to evaluate performance

```
best_dt_model <- dt_tuning %>%
  select_best(metric = 'roc_auc')

best_dt_model
```

Returns a tibble with the best performing model and hyperparameter values

```
# A tibble: 1 x 4
cost_complexity tree_depth  min_n  .config
     <dbl>          <int>    <int>   <chr>
0.0000000758         14       39     Model1
```

# Finalizing the workflow

The `finalize_workflow()` function will finalize a `workflow` that contains a model object with tuning parameters

- Pass `workflow` object
- A tibble with one row of final model hyperparameter values
  - Column names must match hyperparameters in model object

Returns a `workflow` object with set hyperparameter values

```
final_leads_wkfl <- leads_tune_wkfl %>%
  finalize_workflow(best_dt_model)
final_leads_wkfl
```

```
== Workflow ========================================
Preprocessor: Recipe
Model: decision_tree()
-- Preprocessor ------------------------------------
3 Recipe Steps
* step_corr()
* step_normalize()
* step_dummy()
-- Model -------------------------------------------
Decision Tree Model Specification (classification)
Main Arguments:
  cost_complexity = 0.0000000758
  tree_depth = 14
  min_n = 39
Computational engine: rpart
```

# Model fitting

Finalized `workflow` object can be trained with `last_fit()` and original data split object, `leads_split`

```
leads_final_fit <- final_leads_wkfl %>%
  last_fit(split = leads_split)

leads_final_fit %>%
  collect_metrics()
```

**Behind the scenes**

- Training and test datasets created
- `recipe` trained and applied
- **Tuned decision tree** trained with entire training dataset
- Predictions and metrics on **test data**

```
# A tibble: 2 x 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.771
2 roc_auc  binary         0.793
```