Modeling with tidymodels in R

aswanijehangeer

12-12-2022

Machine Learning with tidymodels

Regression

tidymodels is a collection of machine learning packages designed to simplify the machine learning workflow in R.

In this exercise, you will assign each package within the **tidymodels ecosystem** to its corresponding process within the machine learning workflow.

```
# - Creating training and testing data sets ----

# - The rsample package is designed to create training and test datasets.

# - We will create training and test datasets from the home_sales data. This data contains information

# - The outcome variable in this data is selling_price.

# - Create a data split object

home_split <- initial_split(home_sales,
                            prop = 0.7,
                            strata = selling_price)


# training set

home_training <- home_split %>%
  training()

# test set

home_test <- home_split %>%
  testing()



# Checking number of rows in test and train sets

nrow(home_training)
```

```
## [1] 1042
```

```r
nrow(home_test)
```

```
## [1] 450
```

```r
# - Distribution of outcome variables ----

# - In training data set

home_training %>%
  summarize(min_sell_price = min(selling_price),
            max_sell_price = max(selling_price),
            mean_sell_price = mean(selling_price),
            sd_sell_price = sd(selling_price))
```

```
## # A tibble: 1 x 4
##   min_sell_price max_sell_price mean_sell_price sd_sell_price
##            <dbl>          <dbl>           <dbl>         <dbl>
## 1         350000         650000         478852.        80860.
```

```r
# - In test data set

home_test %>%
  summarize(min_sell_price = min(selling_price),
            max_sell_price = max(selling_price),
            mean_sell_price = mean(selling_price),
            sd_sell_price = sd(selling_price))
```

```
## # A tibble: 1 x 4
##   min_sell_price max_sell_price mean_sell_price sd_sell_price
##            <dbl>          <dbl>           <dbl>         <dbl>
## 1         350000         650000         479624.        81342.
```

```r
# - Excellent work! The minimum and maximum selling prices in both data sets are the same. The mean and






# - Linear regression models with tidymodels ----

# - The parsnip package provides a unified syntax for the model fitting process in R.


# Initialize a linear regression object, linear_model

linear_model <- linear_reg() %>%
  # Set the model engine
  set_engine('lm') %>%
  # Set the model mode
  set_mode('regression')
```

```
# - Train our model to predict selling_price using home_age and sqft_living as predictor variables from

# Fit the model using the training data
lm_fit <- linear_model %>%
  fit(selling_price ~ home_age + sqft_living,
      data = home_training)

# - We have defined our model with linear_reg() and trained it to predict selling_price using home_age


# - Exploring estimated model parameters ---

tidy(lm_fit)
```

```
## # A tibble: 3 x 5
##   term        estimate std.error statistic   p.value
##   <chr>          <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)  291264.   7412.       39.3  1.08e-207
## 2 home_age      -1597.    174.       -9.17 2.39e- 19
## 3 sqft_living     104.      2.70     38.6  5.91e-203
```

```
# - The standard error, std.error, for the sqft_living predictor variable is 2.72.
# - The estimated parameter for the home_age predictor variable is -1419.
# - The estimated parameter for the sqft_living predictor variable is 102.
# - The estimated intercept is 292528.2.

# - The tidy() function automatically creates a tibble of estimated model parameters. Since sqft_living


# - Predicting home selling prices ----


# - After fitting a model using the training data, the next step is to use it to make predictions on th

# Predict selling price

home_predictions <- predict(lm_fit,
                            new_data = home_test)

home_predictions
```

```
## # A tibble: 450 x 1
##      .pred
##      <dbl>
##  1 434492.
##  2 380091.
##  3 474560.
##  4 401723.
##  5 509147.
##  6 458395.
##  7 621203.
##  8 443373.
```

```
##  9 411270.
## 10 403283.
## # ... with 440 more rows
```

```r
# - Create a tibble with the selling_price, home_age, and sqft_living columns from the test data set an

home_test_results <- home_test %>%
  select(selling_price, home_age, sqft_living) %>%
  bind_cols(home_predictions)

home_test_results
```

```
## # A tibble: 450 x 4
##    selling_price home_age sqft_living   .pred
##            <dbl>    <dbl>       <dbl>   <dbl>
##  1        465000       10        1530 434492.
##  2        411000       18        1130 380091.
##  3        380000       24        2130 474560.
##  4        356000       24        1430 401723.
##  5        495000        3        2140 509147.
##  6        450000       25        1990 458395.
##  7        624000       26        3570 621203.
##  8        400000        9        1600 443373.
##  9        366000       19        1445 411270.
## 10        415000       24        1445 403283.
## # ... with 440 more rows
```

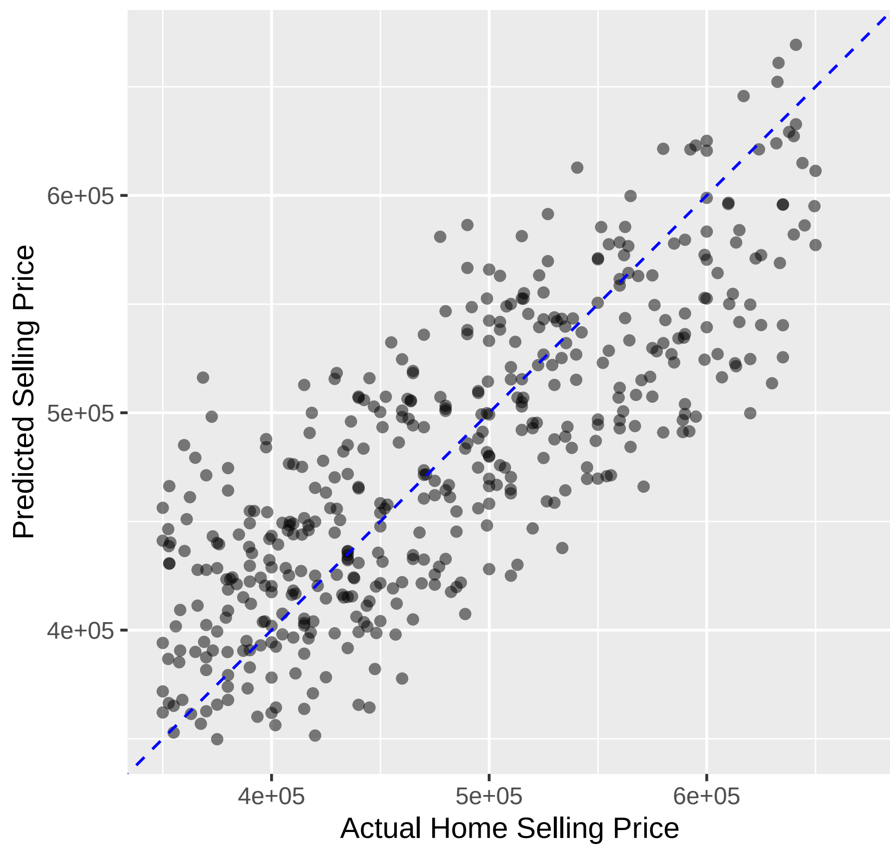
```r
# - We have trained a linear regression model and used it to predict the selling prices of homes in the



# - Evaluating Model Performance

# - Using home_test_results, calculate the RMSE and R squared metrics.

# - Calculate the RMSE metric
home_test_results %>%
  rmse(truth = selling_price,
       estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      49006.
```

```r
# Calculate the R squared metric
home_test_results %>%
  rsq(truth = selling_price, estimate = .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rsq     standard       0.637
```

```r
# - Great job! The RMSE metric indicates that the average prediction error for home selling prices is a


# - R Squared Plot ----

ggplot(home_test_results, aes(x = selling_price, y = .pred)) +
  geom_point(alpha = 0.5) +
  geom_abline(color = 'blue', linetype = 2) +
  coord_obs_pred() +
  labs(x = 'Actual Home Selling Price', y = 'Predicted Selling Price')
```




```r
# - Complete model fitting process with last_fit() ----

linear_fit <- linear_model %>%
  last_fit(selling_price ~ .,
           split = home_split)


# - Collect predictions and view results

predictions_df <- linear_fit %>% collect_predictions()
predictions_df
```

```
## # A tibble: 450 x 5
```

```
##    id                 .pred  .row selling_price .config
##    <chr>              <dbl> <int>         <dbl> <chr>
##  1 train/test split 422473.     2        465000 Preprocessor1_Model1
##  2 train/test split 398635.     3        411000 Preprocessor1_Model1
##  3 train/test split 411968.     5        380000 Preprocessor1_Model1
##  4 train/test split 408826.    11        356000 Preprocessor1_Model1
##  5 train/test split 479162.    12        495000 Preprocessor1_Model1
##  6 train/test split 477483.    21        450000 Preprocessor1_Model1
##  7 train/test split 626778.    22        624000 Preprocessor1_Model1
##  8 train/test split 438605.    27        400000 Preprocessor1_Model1
##  9 train/test split 402629.    36        366000 Preprocessor1_Model1
## 10 train/test split 397272.    38        415000 Preprocessor1_Model1
## # ... with 440 more rows
```

```r
# - Make an R squared plot using predictions_df

ggplot(predictions_df, aes(x = selling_price, y = .pred)) +
  geom_point(alpha = 0.5) +
  geom_abline(color = 'blue', linetype = 2) +
  coord_obs_pred() +
  labs(x = 'Actual Home Selling Price', y = 'Predicted Selling Price')
```

Classification Models

Learn how to predict categorical outcomes by training classification models. Using the skills you've gained so far, you'll predict the **likelihood of customers canceling their service with a telecommunications company.**

```r
# - Create data split object

telecom_split <- rsample::initial_split(telecom_df,
                                        prop = 0.75,
                                        strata = canceled_service)

# - Training set
telecom_training <- telecom_split %>%
  training()

# - test set
telecom_test <- telecom_split %>%
  testing()

# - Check the number of rows in training and test set

nrow(telecom_training)
```

```
## [1] 731
```

```r
nrow(telecom_test)
```

```
## [1] 244
```

```r
# - Fitting a logistic Model ----

# - Specify and logistic model

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

# - Overview

logistic_model
```

```
## Logistic Regression Model Specification (classification)
##
## Computational engine: glm
```

```r
# - Fit to training data
logistic_fit <- logistic_model %>%
  fit(canceled_service ~ avg_call_mins + avg_intl_mins + monthly_charges,
      data = telecom_training)

# - Print model fit object
logistic_fit
```

```
## parsnip model object
##
##
## Call:  stats::glm(formula = canceled_service ~ avg_call_mins + avg_intl_mins +
##     monthly_charges, family = stats::binomial, data = data)
##
## Coefficients:
##     (Intercept)    avg_call_mins    avg_intl_mins  monthly_charges
##       1.9882723       -0.0099791        0.0221365       -0.0005621
##
## Degrees of Freedom: 730 Total (i.e. Null);  727 Residual
## Null Deviance:       932.4
## Residual Deviance: 810.4     AIC: 818.4
```

```
# - Predict outcome categories

class_preds <- predict(logistic_fit, new_data = telecom_test,
                       type = "class")

# - Predict estimated probabilities for each outcome

class_probs <- predict(logistic_fit, new_data = telecom_test,
                       type = "prob")

# - Combine test results

telecom_results <- telecom_test %>%
  select(canceled_service) %>%
  bind_cols(class_preds, class_probs)


telecom_results
```

```
## # A tibble: 244 x 4
##    canceled_service .pred_class .pred_yes .pred_no
##    <fct>            <fct>           <dbl>    <dbl>
##  1 no               no             0.370     0.630
##  2 yes              no             0.138     0.862
##  3 no               no             0.231     0.769
##  4 no               no             0.112     0.888
##  5 yes              yes            0.541     0.459
##  6 yes              no             0.176     0.824
##  7 no               no             0.388     0.612
##  8 no               no             0.0849    0.915
##  9 no               no             0.233     0.767
## 10 no               no             0.0196    0.980
## # ... with 234 more rows
```

```
# -  We have created a tibble of model results using the test data set. Our results tibble contains all


# - Assessing Model Fit ----
```

```
# - Calculate the confusion matrix

yardstick::conf_mat(telecom_results,
         truth = canceled_service,
         estimate = .pred_class)
```

```
##           Truth
## Prediction yes  no
##        yes  27  11
##        no   55 151
```

```
# - Calculate the accuracy of our model

yardstick::accuracy(telecom_results,
                    truth = canceled_service,
                    estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.730
```

```
# - Calculate the sensitivity

sens(telecom_results, truth = canceled_service,
     estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    binary         0.329
```

```
# - Calculate the specificity

spec(telecom_results, truth = canceled_service,
     estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 spec    binary         0.932
```

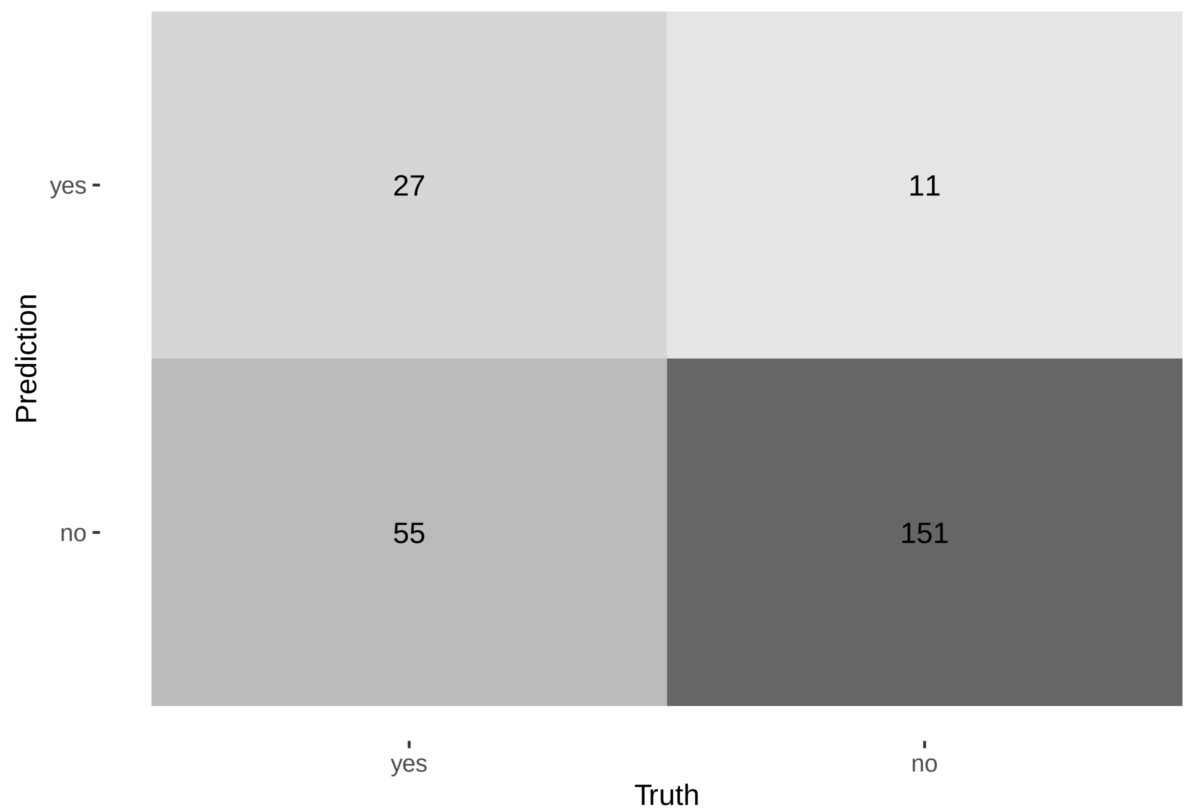
```
# - The specificity of your logistic regression model is 0.926, which is more than double the sensitivi

# - Custom Performance Metric Sets ----

telecom_metrics <- metric_set(accuracy, sens, spec)

# -Calculate metrics using model results tibble

telecom_metrics(telecom_results,
                truth = canceled_service,
                estimate = .pred_class)
```

```
## # A tibble: 3 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.730
## 2 sens     binary         0.329
## 3 spec     binary         0.932
```

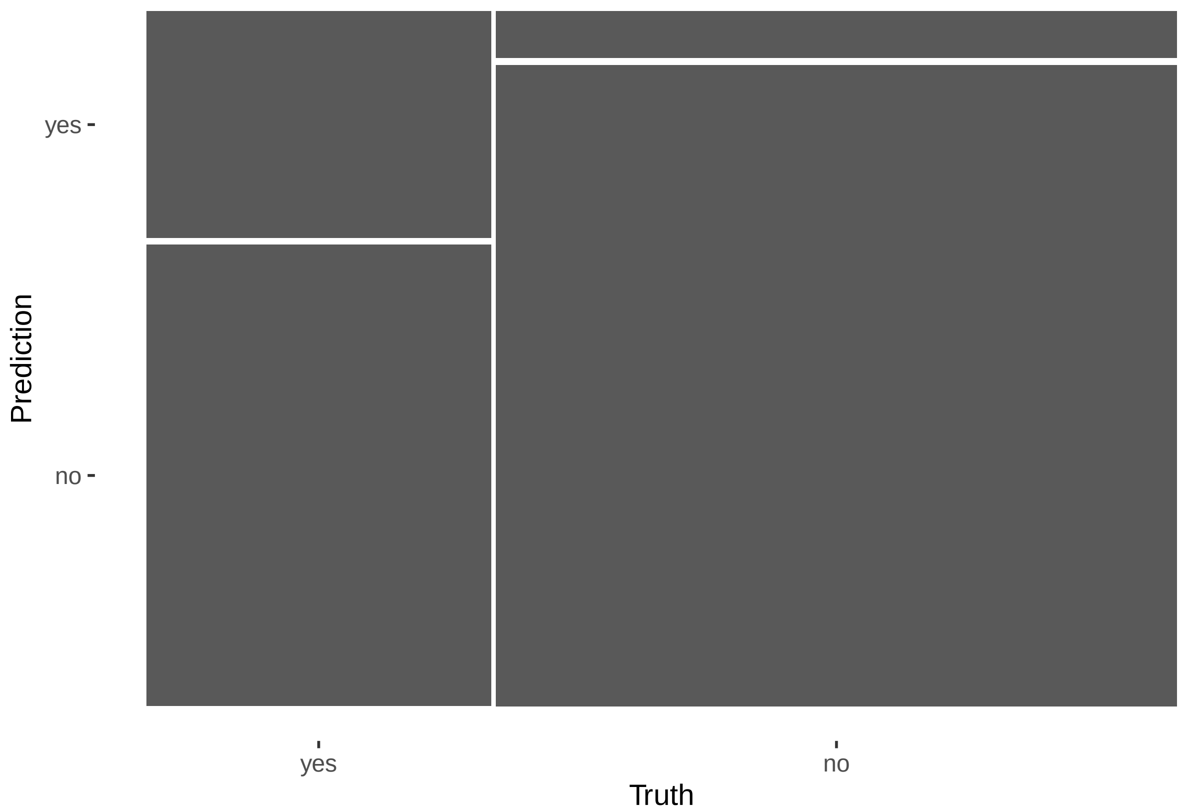
```
# - Create a confusion matrix
conf_mat(telecom_results,
         truth = canceled_service,
         estimate = .pred_class) %>%
  # - Pass to the summary() function
  summary()
```

```
## # A tibble: 13 x 3
##    .metric              .estimator .estimate
##    <chr>                <chr>          <dbl>
##  1 accuracy             binary         0.730
##  2 kap                  binary         0.301
##  3 sens                 binary         0.329
##  4 spec                 binary         0.932
##  5 ppv                  binary         0.711
##  6 npv                  binary         0.733
##  7 mcc                  binary         0.340
##  8 j_index              binary         0.261
##  9 bal_accuracy         binary         0.631
## 10 detection_prevalence binary         0.156
## 11 precision            binary         0.711
## 12 recall               binary         0.329
## 13 f_meas               binary         0.45
```

```
# - Visualizing Model Performance ----

conf_mat(telecom_results,
         truth = canceled_service,
         estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```




```r
# - OR

conf_mat(telecom_results,
         truth = canceled_service,
         estimate = .pred_class) %>%
  autoplot(type = "mosaic")
```

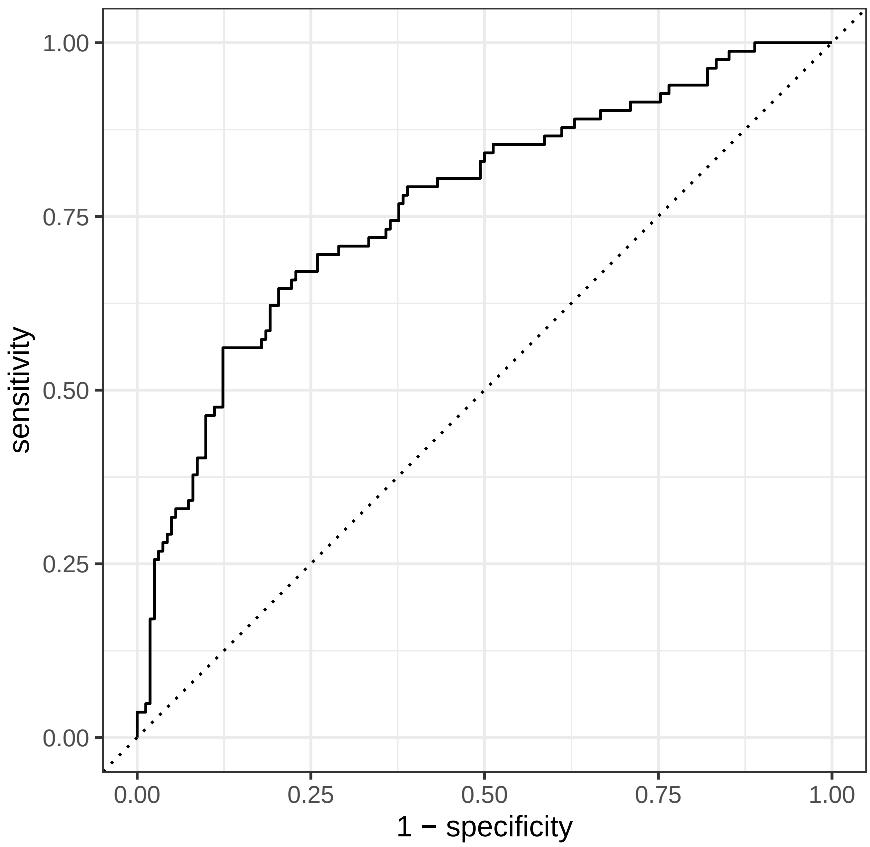



```r
# - Create a tibble, threshold_df, which contains the sensitivity and specificity of your classificatio

threshold_df <- telecom_results %>%
  roc_curve(truth = canceled_service,
            estimate = .pred_yes)

threshold_df
```

```
## # A tibble: 246 x 3
##    .threshold specificity sensitivity
##         <dbl>       <dbl>       <dbl>
##  1  -Inf          0                 1
##  2     0.0196     0                 1
##  3     0.0337     0.00617           1
##  4     0.0391     0.0123            1
##  5     0.0431     0.0185            1
##  6     0.0496     0.0247            1
##  7     0.0516     0.0309            1
##  8     0.0573     0.0370            1
##  9     0.0595     0.0432            1
## 10     0.0601     0.0494            1
## # ... with 236 more rows
```

```r
# - ROC Curve
```

```
threshold_df %>%
  autoplot()
```




```
# - Calculate the area under the ROC curve using the telecom_results tibble.

telecom_results %>%
  roc_auc(truth = canceled_service,
          estimate = .pred_yes)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.766
```

```
# - The area under the ROC curve is 0.77. This indicates that your model gets a C in terms of overall p



# - Automating the Modeling Workflow ----

# - The last_fit() function is designed to streamline the modeling workflow in tidymodels. Instead of t


# - Train model with last_fit()
```

```
telecom_last_fit <- logistic_model %>%
  last_fit(canceled_service ~ avg_call_mins + avg_intl_mins + monthly_charges,
           split = telecom_split)

# - Collecting performance metrics

telecom_last_fit %>%
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.730 Preprocessor1_Model1
## 2 roc_auc  binary         0.766 Preprocessor1_Model1
```

```
# - Collecting Predictions

last_fit_results <- telecom_last_fit %>%
  collect_predictions()

last_fit_results
```

```
## # A tibble: 244 x 7
##    id               .pred_yes .pred_no  .row .pred_class canceled_serv~1 .config
##    <chr>                <dbl>    <dbl> <int> <fct>       <fct>           <chr>
##  1 train/test split    0.370     0.630     3 no          no              Prepro~
##  2 train/test split    0.138     0.862     4 no          yes             Prepro~
##  3 train/test split    0.231     0.769     6 no          no              Prepro~
##  4 train/test split    0.112     0.888     8 no          no              Prepro~
##  5 train/test split    0.541     0.459    11 yes         yes             Prepro~
##  6 train/test split    0.176     0.824    14 no          yes             Prepro~
##  7 train/test split    0.388     0.612    15 no          no              Prepro~
##  8 train/test split    0.0849    0.915    19 no          no              Prepro~
##  9 train/test split    0.233     0.767    20 no          no              Prepro~
## 10 train/test split    0.0196    0.980    22 no          no              Prepro~
## # ... with 234 more rows, and abbreviated variable name 1: canceled_service
```

```
# - Custom metrics function

last_fit_metrics <- metric_set(accuracy, sens,
                               spec, roc_auc)

# - Calculate Metrics

last_fit_metrics(last_fit_results,
                 truth = canceled_service,
                 estimate = .pred_class, .pred_yes)
```

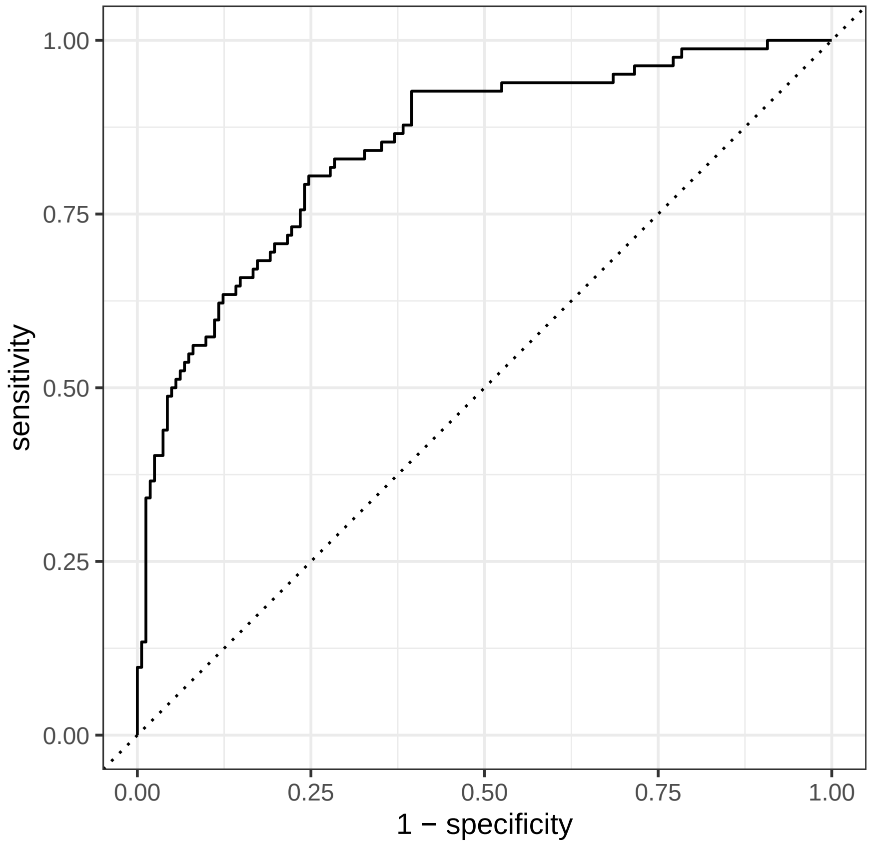
```
## # A tibble: 4 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
```

```
## 1 accuracy binary         0.730
## 2 sens     binary         0.329
## 3 spec     binary         0.932
## 4 roc_auc  binary         0.766
```

```r
# - We were able to train and evaluate your logistic regression model in half the time!


# - Complete modeling workflow

# - Train a logistic regression model

logistic_fit <- logistic_model %>%
  last_fit(canceled_service ~
             avg_call_mins + avg_intl_mins + monthly_charges + months_with_company,
           split = telecom_split)

# - Collect metrics
logistic_fit %>%
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.799 Preprocessor1_Model1
## 2 roc_auc  binary         0.847 Preprocessor1_Model1
```

```r
# - Collect model predictions

logistic_fit %>%
  collect_predictions() %>%
  # - Plot ROC curve
  roc_curve(truth = canceled_service, .pred_yes) %>%
  autoplot()
```



Feature Engineering

Find out how to bake feature engineering pipelines with the **recipes** package. You'll prepare numeric and categorical data to help machine learning algorithms **optimize** your predictions.

```
# - Exploring recipe objects

telecom_rec <- recipe(canceled_service ~ .,
                      data = telecom_df) %>%
  step_log(avg_call_mins, log = 10)



# - Specify feature engineering recipe
telecom_log_rec <- recipe(canceled_service ~ .,
                          data = telecom_training) %>%
  # - Add log transformation step
  step_log(avg_call_mins, avg_intl_mins, base = 10)

# - Print recipe object
telecom_log_rec
```

```
## Recipe
##
## Inputs:
```

```
##
##       role #variables
##    outcome          1
##  predictor          8
##
## Operations:
##
## Log transformation on avg_call_mins, avg_intl_mins
```

```r
# - View variable roles and data types
telecom_log_rec %>%
  summary()
```

```
## # A tibble: 9 x 4
##   variable           type      role      source
##   <chr>              <list>    <chr>     <chr>
## 1 cellular_service   <chr [3]> predictor original
## 2 avg_data_gb        <chr [2]> predictor original
## 3 avg_call_mins      <chr [2]> predictor original
## 4 avg_intl_mins      <chr [2]> predictor original
## 5 internet_service   <chr [3]> predictor original
## 6 contract           <chr [3]> predictor original
## 7 months_with_company <chr [2]> predictor original
## 8 monthly_charges    <chr [2]> predictor original
## 9 canceled_service   <chr [3]> outcome   original
```

```r
# - Train your telecom_log_rec object using the telecom_training data set.

telecom_log_rec_prep <- telecom_log_rec %>%
  prep(training = telecom_training)

telecom_log_rec_prep
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          8
##
## Training data contained 731 data points and no missing data.
##
## Operations:
##
## Log transformation on avg_call_mins, avg_intl_mins [trained]
```

```r
# - Apply to training data

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

```
## # A tibble: 731 x 9
##    cellular_se~1 avg_d~2 avg_c~3 avg_i~4 inter~5 contr~6 month~7 month~8 cance~9
##    <fct>           <dbl>   <dbl>   <dbl> <fct>   <fct>     <dbl>   <dbl> <fct>
##  1 multiple_lin~    8.05    2.52    2.09 digital two_ye~      50    75.2 no
##  2 multiple_lin~    9.96    2.53    2.13 fiber_~ month_~      61   106.  no
##  3 multiple_lin~   10.2     2.60    2.06 fiber_~ month_~      17    92.7 no
##  4 single_line      9.37    2.58    1.94 fiber_~ month_~       4    94.9 no
##  5 multiple_lin~    4.11    2.57    1.81 digital two_ye~      72    55.3 no
##  6 multiple_lin~    7.86    2.58    2.21 digital one_ye~      23    73.8 no
##  7 multiple_lin~    9.24    2.59    2.15 fiber_~ month_~      69    95.4 no
##  8 multiple_lin~   11.0     2.59    1.89 fiber_~ two_ye~      70   112.  no
##  9 single_line      8.02    2.38    1.98 fiber_~ month_~       9    88.4 no
## 10 single_line      5.03    2.54    2.14 digital two_ye~      39    49.8 no
## # ... with 721 more rows, and abbreviated variable names 1: cellular_service,
## #   2: avg_data_gb, 3: avg_call_mins, 4: avg_intl_mins, 5: internet_service,
## #   6: contract, 7: months_with_company, 8: monthly_charges,
## #   9: canceled_service
```

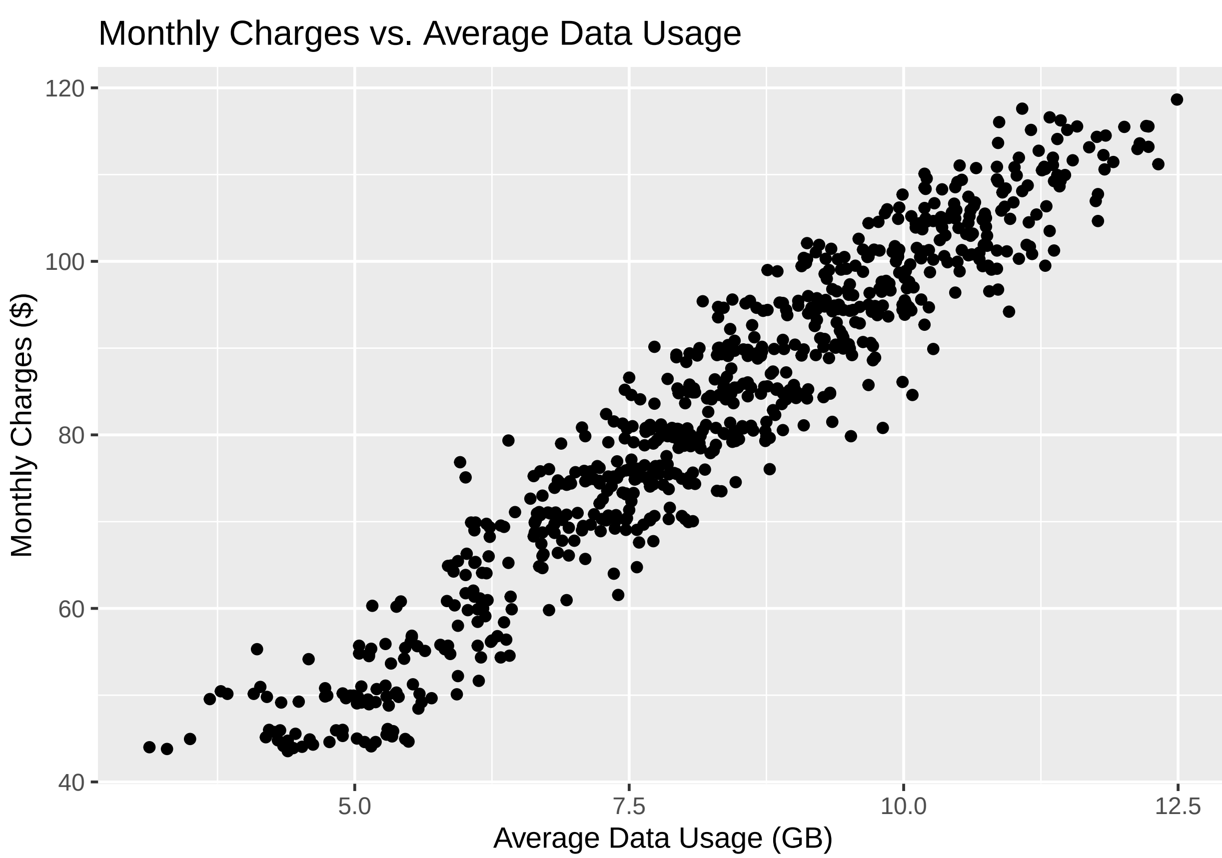
```
# - Apply to the test data

telecom_log_rec_prep %>%
  bake(new_data = telecom_test)
```

```
## # A tibble: 244 x 9
##    cellular_se~1 avg_d~2 avg_c~3 avg_i~4 inter~5 contr~6 month~7 month~8 cance~9
##    <fct>           <dbl>   <dbl>   <dbl> <fct>   <fct>     <dbl>   <dbl> <fct>
##  1 single_line     10.3     2.42    1.74 fiber_~ one_ye~      50   103.  no
##  2 multiple_lin~    5.08    2.40    2.03 digital one_ye~      53    60.0 yes
##  3 single_line      9.3     2.51    2.06 fiber_~ month_~      25    95.7 no
##  4 multiple_lin~    9.4     2.49    2.17 fiber_~ one_ye~      50    99.4 no
##  5 single_line      5.87    2.61    1.94 digital one_ye~      45    54.2 yes
##  6 single_line      7.07    2.40    1.97 fiber_~ month_~      19    73.2 yes
##  7 single_line      6.69    2.55    1.96 digital month_~       6    59.2 no
##  8 multiple_lin~   10.6     2.45    2.17 fiber_~ two_ye~      54   108    no
##  9 multiple_lin~    5.17    2.53    2.08 digital month_~       6    49.0 no
## 10 single_line      8.67    1.97    2.12 fiber_~ two_ye~      55    88.8 no
## # ... with 234 more rows, and abbreviated variable names 1: cellular_service,
## #   2: avg_data_gb, 3: avg_call_mins, 4: avg_intl_mins, 5: internet_service,
## #   6: contract, 7: months_with_company, 8: monthly_charges,
## #   9: canceled_service
```

```
# - We successfully trained your recipe to be able to transform new data sources and applied it to the


# - Numeric Predictors

# - When two variables are highly correlated, their values change linearly with each other and hence pr

# - Create a correlation matrix of the numeric columns of telecom_training.

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

```
##                     avg_data_gb avg_call_mins avg_intl_mins months_with_company
## avg_data_gb           1.0000000    0.18102694    0.16049131          0.43670640
## avg_call_mins         0.1810269    1.00000000    0.08568142          0.03460455
## avg_intl_mins         0.1604913    0.08568142    1.00000000          0.22033894
## months_with_company   0.4367064    0.03460455    0.22033894          1.00000000
## monthly_charges       0.9576652    0.18503239    0.16397301          0.45838946
##                     monthly_charges
## avg_data_gb               0.9576652
## avg_call_mins             0.1850324
## avg_intl_mins             0.1639730
## months_with_company       0.4583895
## monthly_charges           1.0000000
```

```
# - Plot correlated predictors

ggplot(telecom_training, aes(x = avg_data_gb, y = monthly_charges)) +
  geom_point()  +
  labs(title = 'Monthly Charges vs. Average Data Usage',
       y = 'Monthly Charges ($)', x = 'Average Data Usage (GB)')
```




```
# - Removing correlated predictors with recipes.

# - Addin a preprocessing step that removes highly correlated predictor variables using the all_numeric
```

```r
# - Specify a recipe object

telecom_cor_rec <- recipe(canceled_service ~ .,
                          data = telecom_training) %>%
  # - Remove correlated variables
  step_corr(all_numeric(), threshold = 0.8)

# - Train the recipe on telecom training data set

telecom_cor_rec_prep <- telecom_cor_rec %>%
  prep(data = telecom_training)

# - Applying training and test data

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

```
## # A tibble: 731 x 8
##    cellular_service avg_data_gb avg_ca~1 avg_i~2 inter~3 contr~4 month~5 cance~6
##    <fct>                  <dbl>    <dbl>   <dbl> <fct>   <fct>     <dbl> <fct>
##  1 multiple_lines          8.05      328     122 digital two_ye~      50 no
##  2 multiple_lines          9.96      340     136 fiber_~ month_~      61 no
##  3 multiple_lines         10.2       402     116 fiber_~ month_~      17 no
##  4 single_line             9.37      382      87 fiber_~ month_~       4 no
##  5 multiple_lines          4.11      371      64 digital two_ye~      72 no
##  6 multiple_lines          7.86      378     164 digital one_ye~      23 no
##  7 multiple_lines          9.24      392     142 fiber_~ month_~      69 no
##  8 multiple_lines         11.0       390      78 fiber_~ two_ye~      70 no
##  9 single_line             8.02      240      95 fiber_~ month_~       9 no
## 10 single_line             5.03      343     138 digital two_ye~      39 no
## # ... with 721 more rows, and abbreviated variable names 1: avg_call_mins,
## #   2: avg_intl_mins, 3: internet_service, 4: contract, 5: months_with_company,
## #   6: canceled_service
```

```r
telecom_cor_rec_prep %>%
  bake(new_data = telecom_test)
```

```
## # A tibble: 244 x 8
##    cellular_service avg_data_gb avg_ca~1 avg_i~2 inter~3 contr~4 month~5 cance~6
##    <fct>                  <dbl>    <dbl>   <dbl> <fct>   <fct>     <dbl> <fct>
##  1 single_line            10.3       262      55 fiber_~ one_ye~      50 no
##  2 multiple_lines          5.08      250     107 digital one_ye~      53 yes
##  3 single_line             9.3       326     114 fiber_~ month_~      25 no
##  4 multiple_lines          9.4       312     147 fiber_~ one_ye~      50 no
##  5 single_line             5.87      408      88 digital one_ye~      45 yes
##  6 single_line             7.07      249      94 fiber_~ month_~      19 yes
##  7 single_line             6.69      352      91 digital month_~       6 no
##  8 multiple_lines         10.6       281     147 fiber_~ two_ye~      54 no
##  9 multiple_lines          5.17      341     119 digital month_~       6 no
## 10 single_line             8.67       93     131 fiber_~ two_ye~      55 no
## # ... with 234 more rows, and abbreviated variable names 1: avg_call_mins,
## #   2: avg_intl_mins, 3: internet_service, 4: contract, 5: months_with_company,
## #   6: canceled_service
```

```r
# - We have trained your recipe to remove all correlated predictors that exceed the 0.8 correlation thr



# - Multiple Feature Engineering Steps ----

telecom_norm_rec <- recipe(canceled_service ~ .,
                           data = telecom_training) %>%
  step_corr(all_numeric(), threshold = 0.8) %>%
  step_normalize(all_numeric())


# - Train the recipe

telecom_norm_rec_prep <- telecom_norm_rec %>%
  prep(training = telecom_training)

# Test the recipe on test data set

telecom_norm_rec_prep %>%
  bake(new_data = telecom_test)
```

```
## # A tibble: 244 x 8
##    cellular_service avg_data_gb avg_ca~1 avg_i~2 inter~3 contr~4 month~5 cance~6
##    <fct>                  <dbl>    <dbl>   <dbl> <fct>   <fct>     <dbl> <fct>
##  1 single_line            1.05   -1.18   -1.73   fiber_~ one_ye~   0.667 no
##  2 multiple_lines        -1.68   -1.34   -0.0588 digital one_ye~   0.786 yes
##  3 single_line            0.517  -0.355   0.167  fiber_~ month_~  -0.328 no
##  4 multiple_lines         0.569  -0.536   1.23   fiber_~ one_ye~   0.667 no
##  5 single_line           -1.27    0.703  -0.671  digital one_ye~   0.468 yes
##  6 single_line           -0.645  -1.35   -0.477  fiber_~ month_~  -0.567 yes
##  7 single_line           -0.842  -0.0199 -0.574  digital month_~  -1.08  no
##  8 multiple_lines         1.22   -0.936   1.23   fiber_~ two_ye~   0.826 no
##  9 multiple_lines        -1.63   -0.162   0.328  digital month_~  -1.08  no
## 10 single_line            0.189  -3.36    0.714  fiber_~ two_ye~   0.866 no
## # ... with 234 more rows, and abbreviated variable names 1: avg_call_mins,
## #   2: avg_intl_mins, 3: internet_service, 4: contract, 5: months_with_company,
## #   6: canceled_service
```

```r
# - Nominal Predictors, Applying step_dummy() to the predictors.


# - Specify the telecom_recipe_1 object to normalize all numeric predictors and then create dummy varia

telecom_recipe1 <- recipe(canceled_service ~ avg_data_gb + contract,
                          data = telecom_training) %>%
  step_normalize(all_numeric()) %>%
  step_dummy(all_nominal(), -all_outcomes())

# - Train the recipe and apply on test data

telecom_recipe1 %>%
  prep(training = telecom_training) %>%
```

```r
  bake(new_data = telecom_test)
```

```
## # A tibble: 244 x 4
##    avg_data_gb canceled_service contract_one_year contract_two_year
##          <dbl> <fct>                        <dbl>             <dbl>
##  1       1.05  no                               1                 0
##  2      -1.68  yes                              1                 0
##  3       0.517 no                               0                 0
##  4       0.569 no                               1                 0
##  5      -1.27  yes                              1                 0
##  6      -0.645 yes                              0                 0
##  7      -0.842 no                               0                 0
##  8       1.22  no                               0                 1
##  9      -1.63  no                               0                 0
## 10       0.189 no                               0                 1
## # ... with 234 more rows
```

```r
# - Now specify telecom_recipe_2 to create dummy variables for all nominal predictors and then normaliz

telecom_recipe2 <- recipe(canceled_service ~ avg_data_gb + contract,
                          data = telecom_training) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_normalize(all_nominal(), -all_outcomes())

# - Train the recipe and apply on the test data

telecom_recipe2 %>%
  prep(training = telecom_training) %>%
  bake(new_data = telecom_test)
```

```
## # A tibble: 244 x 4
##    avg_data_gb canceled_service contract_one_year contract_two_year
##          <dbl> <fct>                        <dbl>             <dbl>
##  1       10.3  no                               1                 0
##  2        5.08 yes                              1                 0
##  3        9.3  no                               0                 0
##  4        9.4  no                               1                 0
##  5        5.87 yes                              1                 0
##  6        7.07 yes                              0                 0
##  7        6.69 no                               0                 0
##  8       10.6  no                               0                 1
##  9        5.17 no                               0                 0
## 10        8.67 no                               0                 1
## # ... with 234 more rows
```

```r
# - Complete Feature Engineering Pipeline ----

# - Create a recipe that predicts canceled_service using the training data
telecom_recipe_final <- recipe(canceled_service ~ ., data = telecom_training) %>%
  # - Remove correlated predictors
  step_corr(all_numeric(), threshold = 0.8) %>%
  # - Normalize numeric predictors
```

```r
  step_normalize(all_numeric()) %>%
  # - Create dummy variables
  step_dummy(all_nominal(), -all_outcomes())

# - Train recipe
telecom_recipe_prep <- telecom_recipe_final %>%
  prep(training = telecom_training)

# - Transform training data
telecom_training_prep <- telecom_recipe_prep %>%
  bake(new_data = NULL)

# - Transform test data
telecom_test_prep <- telecom_recipe_prep %>%
  bake(new_data = telecom_test)

telecom_test_prep
```

```
## # A tibble: 244 x 9
##    avg_data_gb avg_cal~1 avg_i~2 month~3 cance~4 cellu~5 inter~6 contr~7 contr~8
##          <dbl>     <dbl>   <dbl>   <dbl> <fct>     <dbl>   <dbl>   <dbl>   <dbl>
##  1       1.05    -1.18   -1.73     0.667 no            1       0       1       0
##  2      -1.68    -1.34   -0.0588   0.786 yes           0       1       1       0
##  3       0.517   -0.355   0.167   -0.328 no            1       0       0       0
##  4       0.569   -0.536   1.23     0.667 no            0       0       1       0
##  5      -1.27     0.703  -0.671    0.468 yes           1       1       1       0
##  6      -0.645   -1.35   -0.477   -0.567 yes           1       0       0       0
##  7      -0.842   -0.0199 -0.574   -1.08  no            1       1       0       0
##  8       1.22    -0.936   1.23     0.826 no            0       0       0       1
##  9      -1.63    -0.162   0.328   -1.08  no            0       1       0       0
## 10       0.189   -3.36    0.714    0.866 no            1       0       0       1
## # ... with 234 more rows, and abbreviated variable names 1: avg_call_mins,
## #   2: avg_intl_mins, 3: months_with_company, 4: canceled_service,
## #   5: cellular_service_single_line, 6: internet_service_digital,
## #   7: contract_one_year, 8: contract_two_year
```

```r
# - Train your logistic_model object to predict canceled_service using all available predictor variables

# - Train logistic model

logistic_fit <- logistic_model %>%
  fit(canceled_service ~ ., data = telecom_training_prep)

# - Obtain class predictions
class_preds <- predict(logistic_fit, new_data = telecom_test_prep,
                       type = 'class')

# - Obtain estimated probabilities
prob_preds <- predict(logistic_fit, new_data = telecom_test_prep,
                      type = 'prob')

# - Combine test set results
```

```r
telecom_results <- telecom_test_prep %>%
  select(canceled_service) %>%
  bind_cols(class_preds, prob_preds)

telecom_results
```

```
## # A tibble: 244 x 4
##    canceled_service .pred_class .pred_yes .pred_no
##    <fct>            <fct>           <dbl>    <dbl>
##  1 no               no            0.139      0.861
##  2 yes              no            0.0167     0.983
##  3 no               no            0.323      0.677
##  4 no               no            0.0479     0.952
##  5 yes              no            0.0942     0.906
##  6 yes              no            0.285      0.715
##  7 no               no            0.381      0.619
##  8 no               no            0.0202     0.980
##  9 no               no            0.315      0.685
## 10 no               no            0.00234    0.998
## # ... with 234 more rows
```

```r
# - Performance Metrics

# - Create a confusion matrix
telecom_results %>%
  conf_mat(truth = canceled_service, estimate = .pred_class)
```

```
##           Truth
## Prediction yes  no
##        yes  46  11
##        no   36 151
```

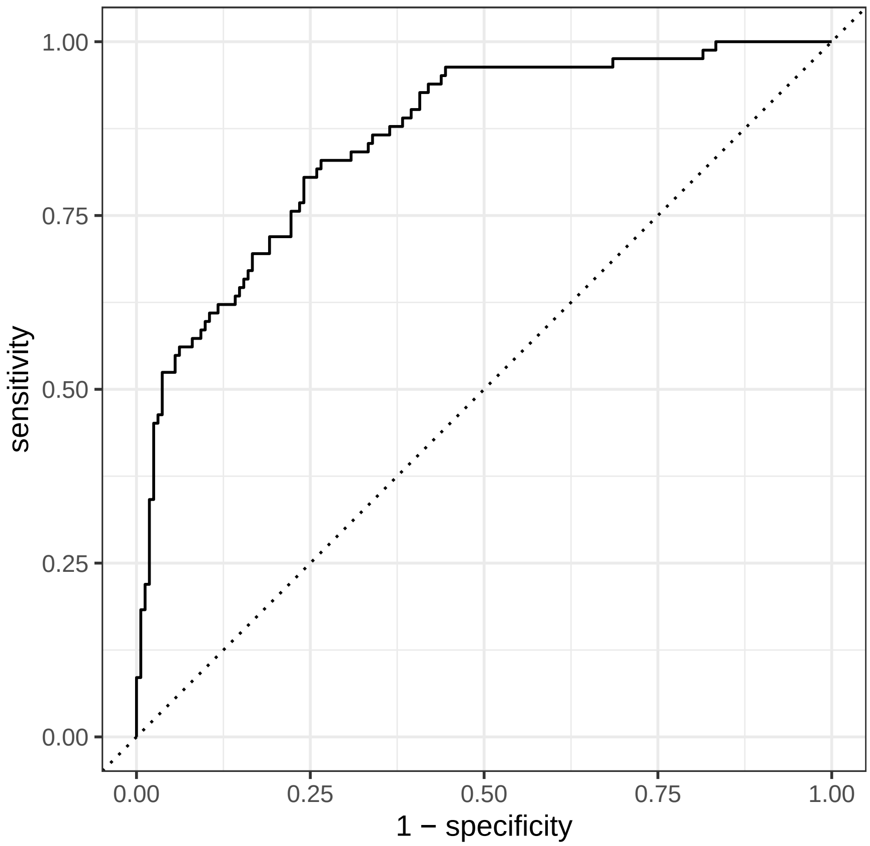
```r
# - Calculate sensitivity
telecom_results %>%
  sens(truth = canceled_service, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    binary         0.561
```

```r
# - Calculate specificity
telecom_results %>%
  spec(truth = canceled_service, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 spec    binary         0.932
```

```r
# - Plot ROC curve
telecom_results %>%
  roc_curve(truth = canceled_service, .pred_yes) %>%
  autoplot()
```



Workflows and Hyperparameter Tuning.

Now it's time to streamline the modeling process using workflows and fine-tune models with **cross-validation** and **hyper parameter tuning**. You'll learn how to tune a **decision tree classification model** to predict whether a **bank's customers are likely to default on their loan.**

Machine Learning Workflows The **workflows** package provides the ability to bundle **parsnip** models and **recipe** objects into a single modeling workflow object. This makes managing a machine learning project much easier and removes the need to keep track of multiple modeling objects.

We will working with the loans_df data set, which contains financial information on consumer loans at a bank. The outcome variable in this data is **loan_default**.

We will create a decision tree model object and specify a feature engineering pipeline for the loan data.

```r
# - Create the data split

loans_split <- initial_split(loans_df,
                             strata = loan_default)
```

```r
# - Training data
loans_training <- loans_split %>%
  training()

# - Test data
loans_test <- loans_split %>%
  testing()


# - Checking correlation between numeric variables

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

```
##                loan_amount interest_rate installment annual_income
## loan_amount     1.00000000   -0.01916938  0.93423788    0.33731395
## interest_rate  -0.01916938    1.00000000  0.01851923   -0.07722095
## installment     0.93423788    0.01851923  1.00000000    0.28993304
## annual_income   0.33731395   -0.07722095  0.28993304    1.00000000
## debt_to_income  0.12184227    0.13315552  0.18527637   -0.19069631
##                debt_to_income
## loan_amount         0.1218423
## interest_rate       0.1331555
## installment         0.1852764
## annual_income      -0.1906963
## debt_to_income      1.0000000
```

```r
# - we have created your training and test data sets and discovered that loan_amount and installment ar


# - Specifying the recipe and model

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

dt_model
```

```
## Decision Tree Model Specification (classification)
##
## Computational engine: rpart
```

```r
# - Create a recipe object with the loans_training data. Use all available predictor variables to predi
# - Add a correlation filter to remove multicollinearity at a 0.85 threshold, normalize all numeric pre


loans_recipe <- recipe(loan_default ~ .,
                       data = loans_training) %>%
  step_corr(all_numeric(), threshold = 0.85) %>%
  step_normalize(all_numeric()) %>%
  step_dummy(all_nominal(), - all_outcomes())
```

```r
loans_recipe
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          7
##
## Operations:
##
## Correlation filter on all_numeric()
## Centering and scaling for all_numeric()
## Dummy variables from all_nominal(), -all_outcomes()
```

```r
# - Create a workflow object, loans_dt_wkfl, that combines your decision tree model and feature enginee

loans_dt_wkfl <- workflow() %>%
  add_model(dt_model) %>%
  add_recipe(loans_recipe)

loans_dt_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
```

```r
# - Train loans_dt_wkfl on last_fit() function

loans_dt_wkfl_fit <- loans_dt_wkfl %>%
  last_fit(split = loans_split)

# - Performance metrics

loans_dt_wkfl_fit %>%
  collect_metrics()
```

```
## # A tibble: 2 x 4
```

```
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.804 Preprocessor1_Model1
## 2 roc_auc  binary         0.766 Preprocessor1_Model1
```

```
# - We have trained a workflow with last_fit() that created training and test data sets, trained and ap
```

Estimating performance with cross validation Cross validation is a method that uses training data to provide multiple estimates of model performance. When trying different model types on your data, it is important to study their performance profile to help decide which model type performs consistently well.

```
# - Create cross validation folds

set.seed(123)

# - creating 10 folds
loans_folds <- vfold_cv(loans_training,
                        v = 10,
                        strata = loan_default)

loans_folds
```

```
## #  10-fold cross-validation using stratification
## # A tibble: 10 x 2
##    splits           id
##    <list>           <chr>
##  1 <split [586/67]> Fold01
##  2 <split [587/66]> Fold02
##  3 <split [588/65]> Fold03
##  4 <split [588/65]> Fold04
##  5 <split [588/65]> Fold05
##  6 <split [588/65]> Fold06
##  7 <split [588/65]> Fold07
##  8 <split [588/65]> Fold08
##  9 <split [588/65]> Fold09
## 10 <split [588/65]> Fold10
```

```
# - creating custom metric function

loans_metric <- metric_set(roc_auc, sens, spec)

# - Using decision tree workflow to perform cross validation using folds and custom metric function.

# - Fit resamples

loans_dt_rs <- loans_dt_wkfl %>%
  fit_resamples(resamples = loans_folds,
                metrics = loans_metric)

# - Performance metrics

loans_dt_rs %>%
  collect_metrics()
```

```
## # A tibble: 3 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 roc_auc binary     0.785    10  0.0150 Preprocessor1_Model1
## 2 sens    binary     0.638    10  0.0229 Preprocessor1_Model1
## 3 spec    binary     0.888    10  0.0220 Preprocessor1_Model1
```

```
# - We have used cross validation to evaluate the performance of your decision tree workflow. Across th
```

```
# - Logistics Model
logistic_model <- logistic_reg() %>%
  set_engine("glm") %>%
  set_mode("classification")

# - Create Workflow

loans_logistic_wkfl <- workflow() %>%
  add_model(logistic_model) %>%
  add_recipe(loans_recipe)

# - Fit resamples

loans_logistic_rs <- loans_logistic_wkfl %>%
  fit_resamples(resamples = loans_folds,
                metrics = loans_metric)

# - View performance metrics

loans_logistic_rs %>%
  collect_metrics()
```

Cross validation with logistic regression

```
## # A tibble: 3 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 roc_auc binary     0.842    10  0.0157 Preprocessor1_Model1
## 2 sens    binary     0.626    10  0.0385 Preprocessor1_Model1
## 3 spec    binary     0.860    10  0.0226 Preprocessor1_Model1
```

```
# - Great Job!
```

Comparing model performance profiles The benefit of the **collect_metrics()** function is that it returns a tibble of cross validation results. This makes it easy to calculate custom summary statistics with the **dplyr** package.

```
# - Detailed cross validation results - Decision Tree

dt_rs_results <- loans_dt_rs %>%
```

```
  collect_metrics(summarize = FALSE)

# - Explore model performance for decision tree
dt_rs_results %>%
  group_by(.metric) %>%
  summarize(min = min(.estimate),
            median = median(.estimate),
            max = max(.estimate))
```

```
## # A tibble: 3 x 4
##   .metric   min median   max
##   <chr>   <dbl>  <dbl> <dbl>
## 1 roc_auc 0.732  0.771 0.878
## 2 sens    0.52   0.62  0.769
## 3 spec    0.775  0.913 0.975
```

```
# - Detailed cross validation results - Logistic Model

logistic_rs_results <- loans_logistic_rs %>%
  collect_metrics(summarize = FALSE)

# - Explore model performance for logistic regression
logistic_rs_results %>%
  group_by(.metric) %>%
  summarize(min = min(.estimate),
            median = median(.estimate),
            max = max(.estimate))
```

```
## # A tibble: 3 x 4
##   .metric   min median   max
##   <chr>   <dbl>  <dbl> <dbl>
## 1 roc_auc 0.749  0.841 0.918
## 2 sens    0.44   0.62  0.808
## 3 spec    0.775  0.888 0.95
```

```
# - Great Job!
```

Hyperparameter Tuning Hyper parameter tuning is a method for fine-tuning the performance of your models. In most cases, the default hyper parameters values of parsnip model objects will not be the optimal values for maximizing model performance.

```
# - a parsnip decision tree model and set all three of its hyperparameters for tuning.
# - Use the rpart engine.

# - Set tuning hyperparameters

dt_tune_model <- decision_tree(cost_complexity = tune(),
                               tree_depth = tune(),
                               min_n = tune()) %>%
  # - Specify engine
  set_engine('rpart') %>%
```

```r
  # - Specify mode
  set_mode('classification')

dt_tune_model
```

```
## Decision Tree Model Specification (classification)
##
## Main Arguments:
##   cost_complexity = tune()
##   tree_depth = tune()
##   min_n = tune()
##
## Computational engine: rpart
```

```r
# - Create a tuning workflow
loans_tune_wkfl <- loans_dt_wkfl %>%
  # - Replace model
  update_model(dt_tune_model)

loans_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
```

Random Grid Search The most common method of hyperparameter tuning is grid search. This method creates a tuning grid with unique combinations of hyperparameter values and uses cross validation to evaluate their performance. **The goal of hyperparameter tuning is to find the optimal combination of values for maximizing model performance.**

```r
# - Hyperparameter tuning with grid search

set.seed(456)

dt_grid <- grid_random(parameters(dt_tune_model),
```

```r
                       size = 5)

dt_grid
```

```
## # A tibble: 5 x 3
##   cost_complexity tree_depth min_n
##             <dbl>      <int> <int>
## 1        6.40e-10         12    10
## 2        7.85e- 9         11    16
## 3        3.95e- 4          9    27
## 4        4.67e- 3         14    20
## 5        1.25e- 3         15    24
```

```r
# - Hyperparameter tuning

dt_tuning <- loans_tune_wkfl %>%
  tune_grid(resamples = loans_folds,
            grid = dt_grid,
            metrics = loans_metric)


# - View Results

dt_tuning %>%
  collect_metrics()
```

```
## # A tibble: 15 x 9
##    cost_complexity tree_depth min_n .metric .estim~1  mean     n std_err .config
##              <dbl>      <int> <int> <chr>   <chr>    <dbl> <int>   <dbl> <chr>
##  1        6.40e-10         12    10 roc_auc binary   0.770    10 0.0215  Prepro~
##  2        6.40e-10         12    10 sens    binary   0.631    10 0.0260  Prepro~
##  3        6.40e-10         12    10 spec    binary   0.798    10 0.0164  Prepro~
##  4        7.85e- 9         11    16 roc_auc binary   0.820    10 0.0144  Prepro~
##  5        7.85e- 9         11    16 sens    binary   0.674    10 0.0287  Prepro~
##  6        7.85e- 9         11    16 spec    binary   0.803    10 0.0202  Prepro~
##  7        3.95e- 4          9    27 roc_auc binary   0.833    10 0.00699 Prepro~
##  8        3.95e- 4          9    27 sens    binary   0.626    10 0.0270  Prepro~
##  9        3.95e- 4          9    27 spec    binary   0.836    10 0.0148  Prepro~
## 10        4.67e- 3         14    20 roc_auc binary   0.799    10 0.0159  Prepro~
## 11        4.67e- 3         14    20 sens    binary   0.682    10 0.0187  Prepro~
## 12        4.67e- 3         14    20 spec    binary   0.835    10 0.0111  Prepro~
## 13        1.25e- 3         15    24 roc_auc binary   0.820    10 0.0118  Prepro~
## 14        1.25e- 3         15    24 sens    binary   0.639    10 0.0249  Prepro~
## 15        1.25e- 3         15    24 spec    binary   0.823    10 0.0124  Prepro~
## # ... with abbreviated variable name 1: .estimator
```

```r
# - Collect detailed tuning results
dt_tuning_results <- dt_tuning %>%
  collect_metrics(summarize = FALSE)

dt_tuning_results
```

```
## # A tibble: 150 x 8
##    id     cost_complexity tree_depth min_n .metric .estimator .estimate .config
##    <chr>            <dbl>      <int> <int> <chr>   <chr>          <dbl> <chr>
##  1 Fold01        6.40e-10         12    10 sens    binary         0.654 Preproc~
##  2 Fold01        6.40e-10         12    10 spec    binary         0.707 Preproc~
##  3 Fold01        6.40e-10         12    10 roc_auc binary         0.743 Preproc~
##  4 Fold02        6.40e-10         12    10 sens    binary         0.538 Preproc~
##  5 Fold02        6.40e-10         12    10 spec    binary         0.775 Preproc~
##  6 Fold02        6.40e-10         12    10 roc_auc binary         0.711 Preproc~
##  7 Fold03        6.40e-10         12    10 sens    binary         0.72  Preproc~
##  8 Fold03        6.40e-10         12    10 spec    binary         0.8   Preproc~
##  9 Fold03        6.40e-10         12    10 roc_auc binary         0.804 Preproc~
## 10 Fold04        6.40e-10         12    10 sens    binary         0.6   Preproc~
## # ... with 140 more rows
```

```
# - Explore detailed ROC AUC results for each fold

dt_tuning_results %>%
  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>
##  1 Fold01       0.743          0.847       0.859
##  2 Fold02       0.711          0.779       0.802
##  3 Fold03       0.804          0.843       0.851
##  4 Fold04       0.678          0.835       0.846
##  5 Fold05       0.834          0.838       0.86
##  6 Fold06       0.817          0.844       0.856
##  7 Fold07       0.779          0.827       0.837
##  8 Fold08       0.821          0.835       0.875
##  9 Fold09       0.760          0.786       0.832
## 10 Fold10       0.667          0.726       0.794
```

Selecting the best model To incorporate hyperparameter tuning into your modeling process, an optimal hyperparameter combination must be selected based on the average value of a performance metric. Then you will be able to finalize your tuning workflow and fit your final model.

```
# - Display the 5 best performing hyperparameter combinations from your tuning results based on the are

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

```
## # A tibble: 5 x 9
##   cost_complexity tree_depth min_n .metric .estima~1  mean     n std_err .config
##             <dbl>      <int> <int> <chr>   <chr>     <dbl> <int>   <dbl> <chr>
## 1        3.95e- 4          9    27 roc_auc binary    0.833    10 0.00699 Prepro~
## 2        1.25e- 3         15    24 roc_auc binary    0.820    10 0.0118  Prepro~
```

```
## 3        7.85e- 9        11    16 roc_auc binary    0.820    10 0.0144  Prepro~
## 4        4.67e- 3        14    20 roc_auc binary    0.799    10 0.0159  Prepro~
## 5        6.40e-10        12    10 roc_auc binary    0.770    10 0.0215  Prepro~
## # ... with abbreviated variable name 1: .estimator
```

```r
# - Choosing the best model

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

best_dt_model
```

```
## # A tibble: 1 x 4
##   cost_complexity tree_depth min_n .config
##             <dbl>      <int> <int> <chr>
## 1        0.000395          9    27 Preprocessor1_Model3
```

```r
# - Finalize the workflow

final_loans_wkfl <- loans_tune_wkfl %>%
  finalize_workflow(best_dt_model)

final_loans_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 = 0.000395000288718773
##   tree_depth = 9
##   min_n = 27
##
## Computational engine: rpart
```

```r
# - Our workflow is now ready for model fitting and prediction on new data sources!
```

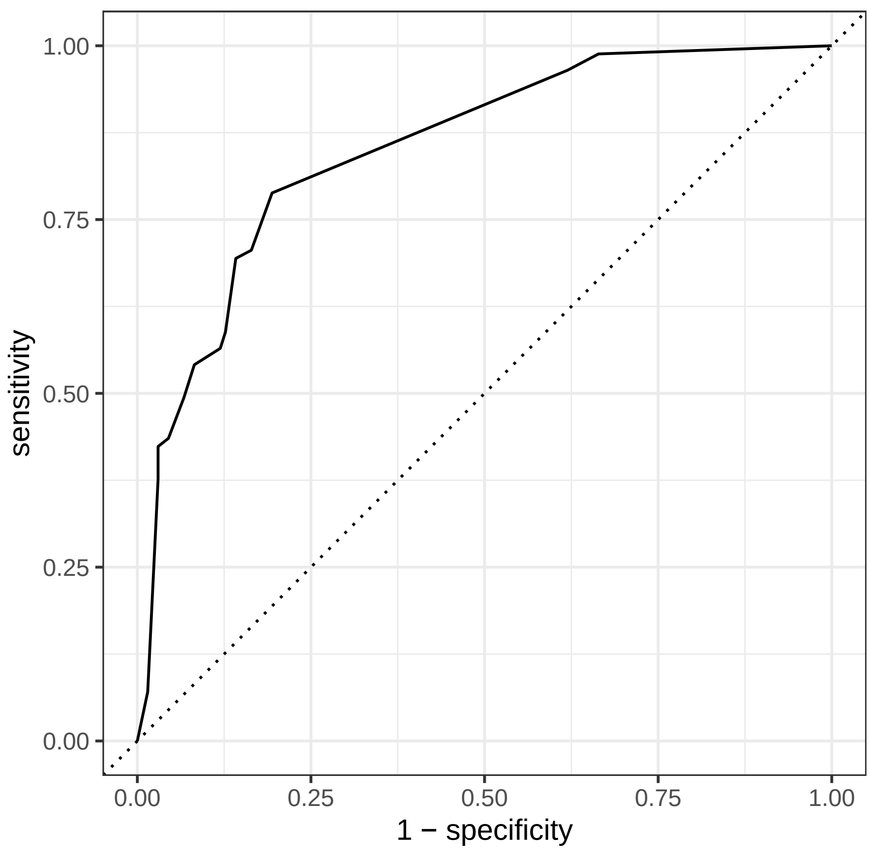
```r
# - Train finalized decision tree workflow
```

```r
loans_final_fit <- final_loans_wkfl %>%
  last_fit(split = loans_split)

# - View performance metrics

loans_final_fit %>%
  collect_metrics()
```

Training final workflow

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.763 Preprocessor1_Model1
## 2 roc_auc  binary         0.851 Preprocessor1_Model1
```

```r
# - Create an ROC curve
loans_final_fit %>%
  # - Collect predictions
  collect_predictions() %>%
  # - Calculate ROC curve metrics
  roc_curve(truth = loan_default, .pred_yes) %>%
  # - Plot the ROC curve
  autoplot()
```



```
# -  We were able to train your finalized workflow with last_fit() and generate predictions on the test
```