**Explore the data**

Our modeling goal here is to predict which hotel stays include children (vs. do not include children or babies) based on the other characteristics in this dataset such as which hotel the guests stay at, how much they pay, etc. This is mostly because more information is gathered when guests check in; the biggest contributor to these differences is not that people who cancel are different from people who do not.

To build our models, let’s filter to **only** the bookings that did not cancel and build a model to predict which hotel stays include children and which do not.

library(tidyverse)

hotels <- readr::read\_csv("<https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-02-11/hotels.csv>")

hotel\_stays <- hotels %>%

filter(is\_canceled == 0) %>%

mutate(

children = case\_when(

children + babies > 0 ~ "children",

TRUE ~ "none"

),

required\_car\_parking\_spaces = case\_when(

required\_car\_parking\_spaces > 0 ~ "parking",

TRUE ~ "none"

)

) %>%

select(-is\_canceled, -reservation\_status, -babies)

hotel\_stays

## # A tibble: 75,166 x 29

## hotel lead\_time arrival\_date\_ye… arrival\_date\_mo… arrival\_date\_we…

##

## 1 Reso… 342 2015 July 27

## 2 Reso… 737 2015 July 27

## 3 Reso… 7 2015 July 27

## 4 Reso… 13 2015 July 27

## 5 Reso… 14 2015 July 27

## 6 Reso… 14 2015 July 27

## 7 Reso… 0 2015 July 27

## 8 Reso… 9 2015 July 27

## 9 Reso… 35 2015 July 27

## 10 Reso… 68 2015 July 27

## # … with 75,156 more rows, and 24 more variables:

## # arrival\_date\_day\_of\_month , stays\_in\_weekend\_nights ,

## # stays\_in\_week\_nights , adults , children , meal ,

## # country , market\_segment , distribution\_channel ,

## # is\_repeated\_guest , previous\_cancellations ,

## # previous\_bookings\_not\_canceled , reserved\_room\_type ,

## # assigned\_room\_type , booking\_changes , deposit\_type ,

## # agent , company , days\_in\_waiting\_list ,

## # customer\_type , adr , required\_car\_parking\_spaces ,

## # total\_of\_special\_requests , reservation\_status\_date

hotel\_stays %>%

count(children)

## # A tibble: 2 x 2

## children n

##

## 1 children 6073

## 2 none 69093

There are more than 10x more hotel stays without children than with.

When I have a new dataset like this one, I often use the skimr package to get an overview of the dataset’s characteristics. The numeric variables here have different very different values and distributions (big vs. small).

library(skimr)

skim(hotel\_stays)

## ── Data Summary ────────────────────────

## Values

## Name hotel\_stays

## Number of rows 75166

## Number of columns 29

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Column type frequency:

## character 14

## Date 1

## numeric 14

## \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Group variables None

##

## ── Variable type: character ────────────────────────────────────────────────────

## skim\_variable n\_missing complete\_rate min max empty

## 1 hotel 0 1 10 12 0

## 2 arrival\_date\_month 0 1 3 9 0

## 3 children 0 1 4 8 0

## 4 meal 0 1 2 9 0

## 5 country 0 1 2 4 0

## 6 market\_segment 0 1 6 13 0

## 7 distribution\_channel 0 1 3 9 0

## 8 reserved\_room\_type 0 1 1 1 0

## 9 assigned\_room\_type 0 1 1 1 0

## 10 deposit\_type 0 1 10 10 0

## 11 agent 0 1 1 4 0

## 12 company 0 1 1 4 0

## 13 customer\_type 0 1 5 15 0

## 14 required\_car\_parking\_spaces 0 1 4 7 0

## n\_unique whitespace

## 1 2 0

## 2 12 0

## 3 2 0

## 4 5 0

## 5 166 0

## 6 7 0

## 7 5 0

## 8 9 0

## 9 10 0

## 10 3 0

## 11 315 0

## 12 332 0

## 13 4 0

## 14 2 0

##

## ── Variable type: Date ─────────────────────────────────────────────────────────

## skim\_variable n\_missing complete\_rate min max

## 1 reservation\_status\_date 0 1 2015-07-01 2017-09-14

## median n\_unique

## 1 2016-09-01 805

##

## ── Variable type: numeric ──────────────────────────────────────────────────────

## skim\_variable n\_missing complete\_rate mean sd

## 1 lead\_time 0 1 80.0 91.1

## 2 arrival\_date\_year 0 1 2016. 0.703

## 3 arrival\_date\_week\_number 0 1 27.1 13.9

## 4 arrival\_date\_day\_of\_month 0 1 15.8 8.78

## 5 stays\_in\_weekend\_nights 0 1 0.929 0.993

## 6 stays\_in\_week\_nights 0 1 2.46 1.92

## 7 adults 0 1 1.83 0.510

## 8 is\_repeated\_guest 0 1 0.0433 0.204

## 9 previous\_cancellations 0 1 0.0158 0.272

## 10 previous\_bookings\_not\_canceled 0 1 0.203 1.81

## 11 booking\_changes 0 1 0.293 0.736

## 12 days\_in\_waiting\_list 0 1 1.59 14.8

## 13 adr 0 1 100. 49.2

## 14 total\_of\_special\_requests 0 1 0.714 0.834

## p0 p25 p50 p75 p100 hist

## 1 0 9 45 124 737 ▇▂▁▁▁

## 2 2015 2016 2016 2017 2017 ▃▁▇▁▆

## 3 1 16 28 38 53 ▆▇▇▇▆

## 4 1 8 16 23 31 ▇▇▇▇▆

## 5 0 0 1 2 19 ▇▁▁▁▁

## 6 0 1 2 3 50 ▇▁▁▁▁

## 7 0 2 2 2 4 ▁▂▇▁▁

## 8 0 0 0 0 1 ▇▁▁▁▁

## 9 0 0 0 0 13 ▇▁▁▁▁

## 10 0 0 0 0 72 ▇▁▁▁▁

## 11 0 0 0 0 21 ▇▁▁▁▁

## 12 0 0 0 0 379 ▇▁▁▁▁

## 13 -6.38 67.5 92.5 125 510 ▇▆▁▁▁

## 14 0 0 1 1 5 ▇▁▁▁▁

How do the hotel stays of guests with/without children vary throughout the year? Is this different in the city and the resort hotel?

hotel\_stays %>%

mutate(arrival\_date\_month = factor(arrival\_date\_month,

levels = [month.name](http://month.name)

)) %>%

count(hotel, arrival\_date\_month, children) %>%

group\_by(hotel, children) %>%

mutate(proportion = n / sum(n)) %>%

ggplot(aes(arrival\_date\_month, proportion, fill = children)) +

geom\_col(position = "dodge") +

scale\_y\_continuous(labels = scales::percent\_format()) +

facet\_wrap(~hotel, nrow = 2) +

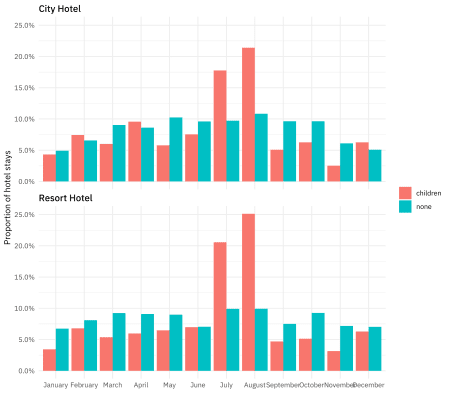
labs(

x = NULL,

y = "Proportion of hotel stays",

fill = NULL

)



Are hotel guests with children more likely to require a parking space?

hotel\_stays %>%

count(hotel, required\_car\_parking\_spaces, children) %>%

group\_by(hotel, children) %>%

mutate(proportion = n / sum(n)) %>%

ggplot(aes(required\_car\_parking\_spaces, proportion, fill = children)) +

geom\_col(position = "dodge") +

scale\_y\_continuous(labels = scales::percent\_format()) +

facet\_wrap(~hotel, nrow = 2) +

labs(

x = NULL,

y = "Proportion of hotel stays",

fill = NULL

)



There are many more relationships like this we can explore. In many situations I like to use the ggpairs() function to get a high-level view of how variables are related to each other.

library(GGally)

hotel\_stays %>%

select(

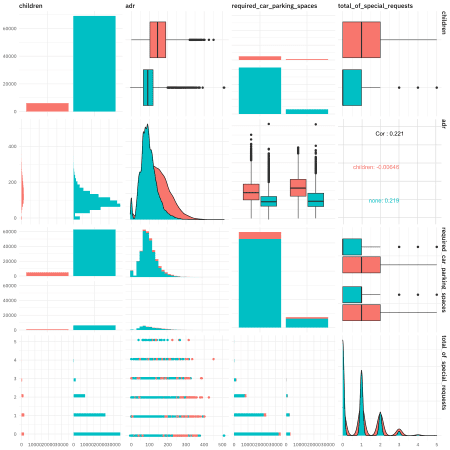
children, adr,

required\_car\_parking\_spaces,

total\_of\_special\_requests

) %>%

ggpairs(mapping = aes(color = children))



**Build models with recipes**

The next step for us is to create a dataset for modeling. Let’s include a set of columns we are interested in, and convert all the character columns to factors, for the modeling functions coming later.

hotels\_df <- hotel\_stays %>%

select(

children, hotel, arrival\_date\_month, meal, adr, adults,

required\_car\_parking\_spaces, total\_of\_special\_requests,

stays\_in\_week\_nights, stays\_in\_weekend\_nights

) %>%

mutate\_if(is.character, factor)

Now it is time for tidymodels! we split the data into training and testing sets using initial\_split(). Next, we use a recipe() to build a set of steps for data preprocessing and feature engineering.

* First, we must tell the recipe() what our model is going to be (using a formula here) and what our training data is.
* We then downsample the data, since there are about 10x more hotel stays without children than with. If we don’t do this, our model will learn very effectively about how to predict the negative case. 
* We then convert the factor columns into (one or more) numeric binary (0 and 1) variables for the levels of the training data.
* Next, we remove any numeric variables that have zero variance.
* As a last step, we normalize (center and scale) the numeric variables. We need to do this because some of them are on very different scales from each other and the model we want to train is sensitive to this.
* Finally, we prep() the recipe(). This means we actually do something with the steps and our training data; we estimate the required parameters from hotel\_train to implement these steps so this whole sequence can be applied later to another dataset.

We then can do exactly that, and apply these transformations to the testing data; the function for this is bake(). We won’t touch the testing set again until the very end.

library(tidymodels)

set.seed(1234)

hotel\_split <- initial\_split(hotels\_df)

hotel\_train <- training(hotel\_split)

hotel\_test <- testing(hotel\_split)

hotel\_rec <- recipe(children ~ ., data = hotel\_train) %>%

step\_downsample(children) %>%

step\_dummy(all\_nominal(), -all\_outcomes()) %>%

step\_zv(all\_numeric()) %>%

step\_normalize(all\_numeric()) %>%

prep()

hotel\_rec

## Data Recipe

##

## Inputs:

##

## role #variables

## outcome 1

## predictor 9

##

## Training data contained 56375 data points and no missing data.

##

## Operations:

##

## Down-sampling based on children [trained]

## Dummy variables from hotel, arrival\_date\_month, ... [trained]

## Zero variance filter removed no terms [trained]

## Centering and scaling for adr, adults, ... [trained]

test\_proc <- bake(hotel\_rec, new\_data = hotel\_test)

Now it’s time to **specify** and then **fit** our models. First we specify and fit a nearest neighbors classification model, and then a decision tree classification model. Check out what data we are training these models on: juice(hotel\_rec). The recipe hotel\_rec contains all our transformations for data preprocessing and feature engineering, *as well as* the data these transformations were estimated from. When we juice() the recipe, we squeeze that training data back out, transformed in the ways we specified including the downsampling. The object juice(hotel\_rec) is a dataframe with 9,176 rows while the our original training data hotel\_train has 56,375 rows.

knn\_spec <- nearest\_neighbor() %>%

set\_engine("kknn") %>%

set\_mode("classification")

knn\_fit <- knn\_spec %>%

fit(children ~ ., data = juice(hotel\_rec))

knn\_fit

## parsnip model object

##

## Fit time: 1.3s

##

## Call:

## kknn::train.kknn(formula = formula, data = data, ks = 5)

##

## Type of response variable: nominal

## Minimal misclassification: 0.2518527

## Best kernel: optimal

## Best k: 5

tree\_spec <- decision\_tree() %>%

set\_engine("rpart") %>%

set\_mode("classification")

tree\_fit <- tree\_spec %>%

fit(children ~ ., data = juice(hotel\_rec))

tree\_fit

## parsnip model object

##

## Fit time: 257ms

## n= 9176

##

## node), split, n, loss, yval, (yprob)

## \* denotes terminal node

##

## 1) root 9176 4588 children (0.5000000 0.5000000)

## 2) adr>=-0.03405154 4059 1092 children (0.7309682 0.2690318) \*

## 3) adr< -0.03405154 5117 1621 none (0.3167872 0.6832128)

## 6) total\_of\_special\_requests>=0.647359 944 416 children (0.5593220 0.4406780) \*

## 7) total\_of\_special\_requests< 0.647359 4173 1093 none (0.2619219 0.7380781)

## 14) adults< -2.852103 80 9 children (0.8875000 0.1125000) \*

## 15) adults>=-2.852103 4093 1022 none (0.2496946 0.7503054) \*

We trained these models on the downsampled training data; we have not touched the testing data.

**Evaluate models**

To evaluate these models, let’s build a validation set. We can build a set of Monte Carlo splits from the downsampled training data (juice(hotel\_rec)) and use this set of resamples to estimate the performance of our two models using the fit\_resamples() function. This function does *not* do any tuning of the model parameters; in fact, it does not even keep the models it trains. This function is used for computing performance metrics across some set of resamples like our validation splits. It will fit a model such as knn\_spec to each resample and evaluate on the heldout bit from each resample, and then we can collect\_metrics() from the result.

set.seed(1234)

validation\_splits <- mc\_cv(juice(hotel\_rec), prop = 0.9, strata = children)

validation\_splits

## # # Monte Carlo cross-validation (0.9/0.1) with 25 resamples using stratification

## # A tibble: 25 x 2

## splits id

##

## 1 Resample01

## 2 Resample02

## 3 Resample03

## 4 Resample04

## 5 Resample05

## 6 Resample06

## 7 Resample07

## 8 Resample08

## 9 Resample09

## 10 Resample10

## # … with 15 more rows

knn\_res <- fit\_resamples(

children ~ .,

knn\_spec,

validation\_splits,

control = control\_resamples(save\_pred = TRUE)

)

knn\_res %>%

collect\_metrics()

## # A tibble: 2 x 5

## .metric .estimator mean n std\_err

##

## 1 accuracy binary 0.74 25 0.00272

## 2 roc\_auc binary 0.804 25 0.00219

tree\_res <- fit\_resamples(

children ~ .,

tree\_spec,

validation\_splits,

control = control\_resamples(save\_pred = TRUE)

)

tree\_res %>%

collect\_metrics()

## # A tibble: 2 x 5

## .metric .estimator mean n std\_err

##

## 1 accuracy binary 0.722 25 0.00248

## 2 roc\_auc binary 0.741 25 0.00230

This validation set gives us a better estimate of how our models are doing than predicting the whole training set at once. The nearest neighbor model performs somewhat better than the decision tree. Let’s visualize these results.

knn\_res %>%

unnest(.predictions) %>%

mutate(model = "kknn") %>%

bind\_rows(tree\_res %>%

unnest(.predictions) %>%

mutate(model = "rpart")) %>%

group\_by(model) %>%

roc\_curve(children, .pred\_children) %>%

ggplot(aes(x = 1 - specificity, y = sensitivity, color = model)) +

geom\_line(size = 1.5) +

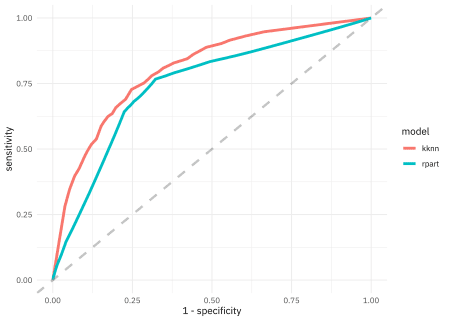
geom\_abline(

lty = 2, alpha = 0.5,

color = "gray50",

size = 1.2

)



We can also create a confusion matrix.

knn\_conf <- knn\_res %>%

unnest(.predictions) %>%

conf\_mat(children, .pred\_class)

knn\_conf

## Truth

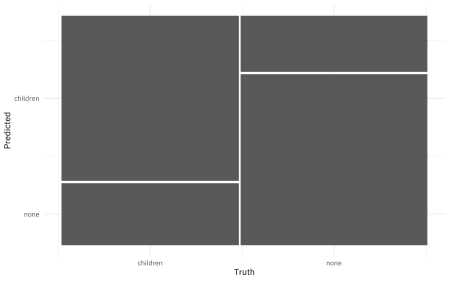
## Prediction children none

## children 8325 2829

## none 3125 8621

knn\_conf %>%

autoplot()



FINALLY, let’s check in with our transformed testing data and see how we can expect this model to perform on new data.

knn\_fit %>%

predict(new\_data = test\_proc, type = "prob") %>%

mutate(truth = hotel\_test$children) %>%

roc\_auc(truth, .pred\_children)

## # A tibble: 1 x 3

## .metric .estimator .estimate

##

## 1 roc\_auc binary 0.795

Notice that this AUC value is about the same as from our validation splits.

**Summary**

Let me know if you have questions or feedback about using recipes with tidymodels and how to get started. I am glad to be using these #TidyTuesday datasets for predictive modeling!