Regression: Feature Selection, Part 3

Will Doyle

Introduction

In this last lecture we'll introduce a couple of new concepts: cross validation and feature selection.

```
library(tidyverse)
```

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'pillar'
```

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'hms'
```

```
## Warning: package 'tibble' was built under R version 4.1.2
```

```
library(tidymodels)
```

```
## Warning: package 'recipes' was built under R version 4.1.2
```

```
library(knitr)
library(plotly)
library(modelr)
```

The Data

We're going to work with a subset of the data that only includes a few variables, which are selected below.

```
mv<-readRDS("mv.Rds")%>%
  filter(!is.na(budget))%>%
  mutate(log_gross=log(gross))%>%
  mutate(year=as_factor(year))%>%
  select(
    title,
    log_gross,
         budget,
         rating,
         genre,
         runtime,
         year)%>%
  drop_na()
```

As usual, we'll split the data 75/25.

```
split_data<-initial_split(mv)

mv_train<-training(split_data)

mv_test<-testing(split_data)
```

Cross Validation

The essence of prediction is discovering the extent to which our models can predict outcomes for data that does not come from our sample. Many times this process is temporal. We fit a model to data from one time period, then take predictors from a subsequent time period to come up with a prediction in the future. For instance, we might use data on team performance to predict the likely winners and losers for upcoming soccer games.

This process does not have to be temporal. We can also have data that is out of sample because it hadn't yet been collected when our first data was collected, or we can also have data that is out of sample because we designated it as out of sample.

The data that is used to generate our predictions is known as *training* data. The idea is that this is the data used to train our model, to let it know what the relationship is between our predictors and our outcome. So far, we have worked mostly with training data.

That data that is used to validate our predictions is known as *testing* data. With testing data, we take our trained model and see how good it is at predicting outcomes using out of sample data.

One very simple approach to this would be to cut our data in half. This is what we've done so far. We could then train our model on half the data, then test it on the other half. This would tell us whether our measure of model fit (e.g. rmse, auc) is similar or different when we apply our model to out of sample data.

But this would only be a "one-shot" approach. It would be better to do this multiple times, cutting the data into two parts: training and testing, then fitting the model to the training data, and then checking its predictions against the testing data. That way, we could generate a large number of rmse's to see how well the model fits on lots of different possible out-of-sample predictions.

This process is called *cross validation*, and it involves two important decisions: first, how will the data be cut, and how many times will the validation run.

We're going to cut our training dataset 75/25, and we'll repeat that 25 times. This is so our code will run faster– we would really want to do this more like 1,000 times in practice.

Monte Carlo Resampling

```
mv_rs<-mc_cv(mv_train,times=25) ## More like 1000 in practice
```

Lasso for Feature Selection

One of the key decisions for an analyst is which variables to include. We can make decisions about this using theory, or our understanding of the context, but we can also rely on computational approaches. This is known as *regularization* and it involves downweighting the importance of coefficients from a model based on the contribution that a predictor makes. We're going to make use of a regularization penalty known as the "lasso." The lasso downweights variables mostly be dropping variables that are highly correlated with one another, leaving only one of the correlated variables as contributors to the model. We set the degree to which this penalty will be implemented by setting the "penalty" variable in the model specification.

```
penalty_spec<-.1

mixture_spec<-1

lasso_fit<-
  linear_reg(penalty=penalty_spec,
             mixture=mixture_spec) %>%
```

```
  set_engine("glmnet")%>%
  set_mode("regression")
```

Define the Workflow

```
movie_wf<-workflow()
```

Add the Model

```
movie_wf<-movie_wf%>%
  add_model(lasso_fit)
```

Set Formula

In setting the recipe for this model, we're now going to include ever variable in the dataset. This is very common in these kinds of applications.

```
movie_formula<-as.formula("log_gross~.")
```

Recipe

Because we have so many predictors, we need to generalize our process for feature engineering. Instead of running steps on particular variables, we're going to use the capabilities of tidymodels to select types of variables.

```
movie_rec<-recipe(movie_formula,mv)%>%
  update_role(title,new_role="id variable")%>%
  update_role(log_gross,new_role="outcome")%>%  ## specify dv
  step_log(budget)%>%
  step_other(all_nominal_predictors(),threshold = .01)%>%
  step_dummy(all_nominal_predictors())%>%
  step_normalize(all_predictors())%>% ## Convert all to Z scores
  step_naomit(all_predictors()) ## drop missing
```

To see what this does, we can use the `prep` and `bake` commands

```
mv_processed<-movie_rec%>%prep()%>%bake(mv_train)
```

Now we can add our model to the recipe.

```
movie_wf<-movie_wf%>%
  add_recipe(movie_rec)
```

Fit resamples

To fit a model to resampled data, we use `fit_resamples.` It's now going to fit the model to the training data and then test the model against the testing data for all 25 of our resampled datasets.

```
movie_lasso_fit<-movie_wf%>%
  fit_resamples(mv_rs)
```

```
## Warning: package 'rlang' was built under R version 4.1.2
## Warning: package 'vctrs' was built under R version 4.1.2
## Warning: package 'glmnet' was built under R version 4.1.2
```

Examine resamples and fit

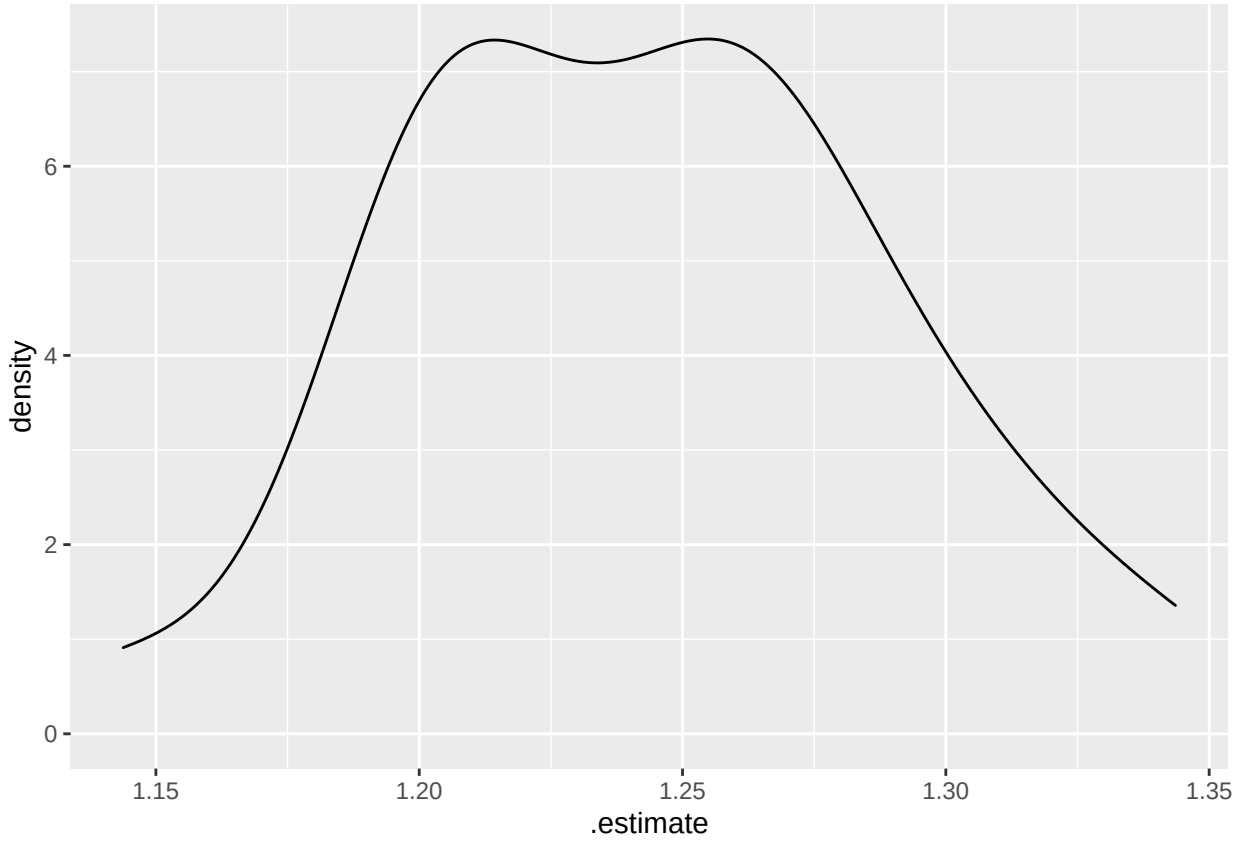
Once we run this model, we get a measure of model fit from every testing dataset. `collect_metrics` will let us see the average.

```
movie_lasso_fit%>%
  collect_metrics()
```

```
## # A tibble: 2 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   1.24     25 0.00922 Preprocessor1_Model1
## 2 rsq     standard   0.498    25 0.00487 Preprocessor1_Model1
```

CV results

```
movie_lasso_fit%>%
  unnest(.metrics)%>%
  filter(.metric=="rmse")%>%
  ggplot(aes(x=.estimate))+
  geom_density()
```



Finalize Workflow

```
movie_lasso_final <- finalize_workflow(movie_wf,
                                       select_best(movie_lasso_fit,metric="rmse")) %>%
  fit(mv_test)
```

Parameter Estimates

```
movie_lasso_final%>%
  extract_fit_parsnip()%>%
  tidy()%>%
  kable()
```

term	estimate	penalty
(Intercept)	17.9900207	0.1
budget	1.0371408	0.1
runtime	0.0662626	0.1
rating_Not.Rated	-0.1481886	0.1
rating_PG	0.0000000	0.1
rating_PG.13	0.0000000	0.1
rating_R	-0.1783678	0.1
rating_other	-0.1300777	0.1
genre_Adventure	0.0000000	0.1
genre_Animation	0.0495060	0.1
genre_Biography	0.0000000	0.1
genre_Comedy	0.0000000	0.1
genre_Crime	0.0000000	0.1
genre_Drama	-0.0135335	0.1
genre_Horror	0.2123768	0.1
genre_other	0.0000000	0.1
year_X2001	0.0000000	0.1
year_X2002	0.0000000	0.1
year_X2003	0.0000000	0.1
year_X2004	0.0000000	0.1
year_X2005	0.0000000	0.1
year_X2006	0.0000000	0.1
year_X2007	0.0000000	0.1
year_X2008	0.0000000	0.1
year_X2009	0.0000000	0.1
year_X2010	0.0000000	0.1
year_X2011	0.0000000	0.1
year_X2012	0.0000000	0.1
year_X2013	0.0000000	0.1
year_X2014	0.0000000	0.1
year_X2015	0.0000000	0.1
year_X2016	0.0000000	0.1
year_X2017	0.0000000	0.1
year_X2018	0.0000000	0.1
year_X2019	0.0000000	0.1
year_other	0.0000000	0.1

Prediction in the testing dataset

```
movie_lasso_final%>%last_fit(split_data)%>%
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       1.27  Preprocessor1_Model1
## 2 rsq     standard       0.497 Preprocessor1_Model1
```

Model Tuning

The problem with the above is that I arbitrarily set the value of penalty to be .1. Do I know this was correct? No! What we need to do is try out a bunch of different values of the penalty, and see which one gives us the best model fit. This process has the impressive name of "hyperparameter tuning" but it could just as easily be called "trying a bunch of stuff to see what works."

Below I'm going to give the argument `tune()` for the value of penalty. This will allow us to "fill in" values later.

```
lasso_tune_fit<-
  linear_reg(penalty=tune(),mixture=mixture_spec)%>%
  set_engine("glmnet")
```

Update Workflow

```
movie_wf<-movie_wf%>%
  update_model(lasso_tune_fit)
```

Create Grid for Model Tuning

A tuning grid is a set of numbers we might want to use. I use the function `grid_regular` and ask it to give me 10 possible values of the penalty.

```
lasso_grid<-grid_regular(parameters(lasso_tune_fit) ,levels=10)
```

Fit Using the Grid

```
movie_lasso_tune_fit <-
  movie_wf %>%
    tune_grid(mv_rs,grid=lasso_grid)
```

Examine Results

Lets' take a look and see which models fit better.

```
movie_lasso_tune_fit%>%
  collect_metrics()%>%
  filter(.metric=="rmse")%>%
  arrange(mean)
```

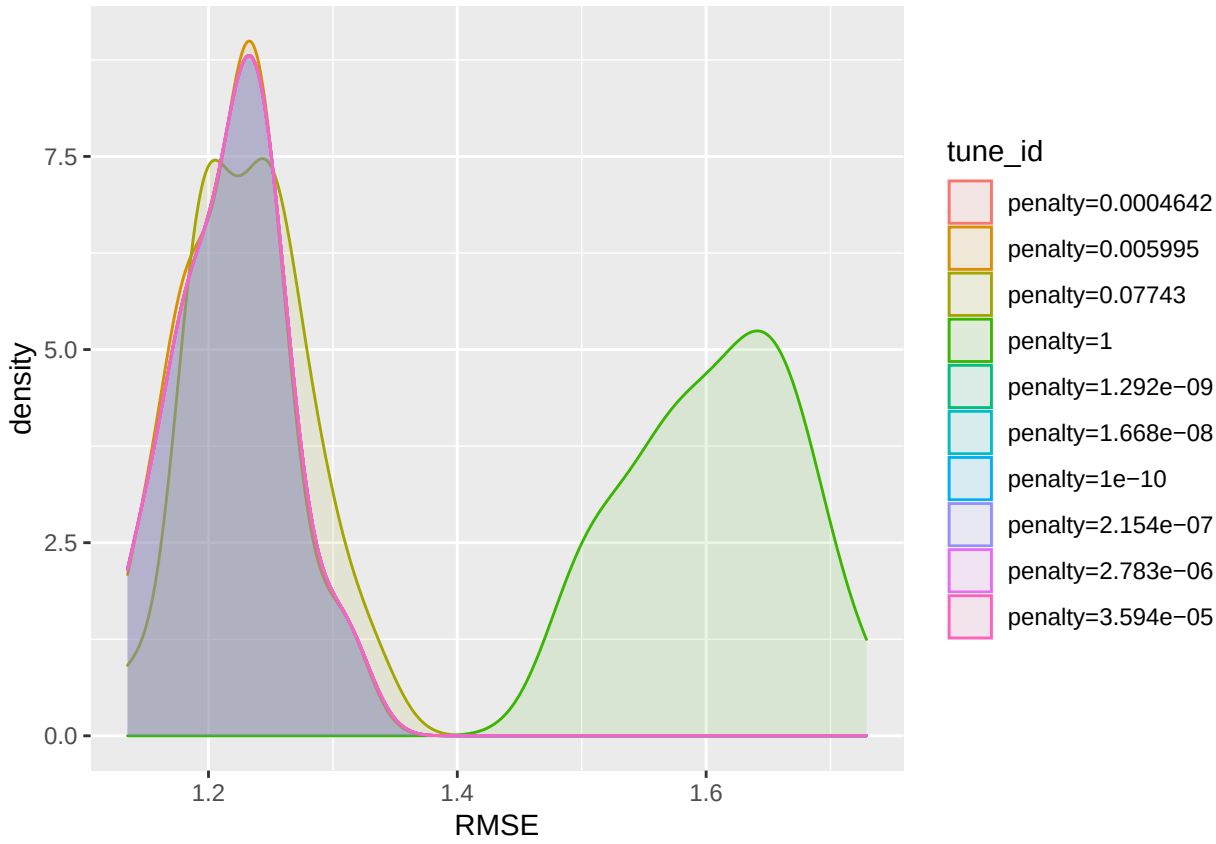
```
## # A tibble: 10 x 7
##          penalty .metric .estimator  mean     n std_err .config
##            <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1 0.00599       rmse    standard    1.22    25 0.00874 Preprocessor1_Model08
##  2 0.000464      rmse    standard    1.22    25 0.00884 Preprocessor1_Model07
##  3 0.0000000001  rmse    standard    1.22    25 0.00884 Preprocessor1_Model01
##  4 0.00000000129 rmse    standard    1.22    25 0.00884 Preprocessor1_Model02
##  5 0.0000000167  rmse    standard    1.22    25 0.00884 Preprocessor1_Model03
##  6 0.000000215   rmse    standard    1.22    25 0.00884 Preprocessor1_Model04
##  7 0.00000278    rmse    standard    1.22    25 0.00884 Preprocessor1_Model05
##  8 0.0000359     rmse    standard    1.22    25 0.00884 Preprocessor1_Model06
##  9 0.0774        rmse    standard    1.23    25 0.00915 Preprocessor1_Model09
## 10 1             rmse    standard    1.61    25 0.0131  Preprocessor1_Model10
```

It looks like using a very small penalty (like basically none) is the way to go.

Plot Results

Let's confirm what we learned by plotting the results.

```
movie_lasso_tune_fit%>%
unnest(.metrics)%>%
  filter(.metric=="rmse")%>%
  mutate(tune_id=paste0("penalty=",prettyNum(penalty,digits=4))) %>%
  select(tune_id,.estimate)%>%
  rename(RMSE=.estimate)%>%
  ggplot(aes(x=RMSE,color=tune_id,fill=tune_id))+
  geom_density(alpha=.1)
```



What this is telling us is that big penalty values are a really bad idea, and that most of the lower penalty values fit the data just fine.

Choose best model and fit to training data

We can choose the best model and then fit it to our training dataset.

```
movie_final<-
  finalize_workflow(movie_wf,
                    select_best(movie_lasso_tune_fit,
                                metric="rmse"))%>%
  fit(mv_train)
```

Examine Parameter Estimates

```
movie_final%>%
  extract_fit_parsnip()%>%
  tidy()%>%
  mutate(penalty=prettyNum(penalty,digits=4))%>%
  kable()
```

term	estimate	penalty
(Intercept)	17.9066641	0.005995
budget	1.0208128	0.005995
runtime	0.1897202	0.005995

term	estimate	penalty
rating_Not.Rated	-0.1753524	0.005995
rating_PG	0.0271672	0.005995
rating_PG.13	-0.0006412	0.005995
rating_R	-0.2163684	0.005995
rating_other	-0.0252554	0.005995
genre_Adventure	0.0210392	0.005995
genre_Animation	0.1183175	0.005995
genre_Biography	-0.0993921	0.005995
genre_Comedy	-0.0020505	0.005995
genre_Crime	-0.0497932	0.005995
genre_Drama	-0.0883640	0.005995
genre_Horror	0.1481076	0.005995
genre_other	0.0000000	0.005995
year_X2001	-0.0045242	0.005995
year_X2002	0.0000000	0.005995
year_X2003	0.0080653	0.005995
year_X2004	0.0341622	0.005995
year_X2005	0.0000684	0.005995
year_X2006	0.0000000	0.005995
year_X2007	0.0148053	0.005995
year_X2008	0.0201085	0.005995
year_X2009	0.0000000	0.005995
year_X2010	-0.0016501	0.005995
year_X2011	0.0694180	0.005995
year_X2012	0.0296979	0.005995
year_X2013	0.0592749	0.005995
year_X2014	0.0595661	0.005995
year_X2015	0.0302126	0.005995
year_X2016	0.0227654	0.005995
year_X2017	0.0942815	0.005995
year_X2018	0.0670247	0.005995
year_X2019	0.0435173	0.005995
year_other	0.0453819	0.005995

This is almost identical to what we would have gotten had we just used OLS, but we know now for certain!

Make Prediction

```
pred_df<-movie_final%>%
  predict(mv_test)%>%
  rename(`Predicted Gross`=.pred)%>%
  bind_cols(mv_test)%>%
  rename(`Actual Gross`=log_gross)
```

```
rmse_final<-yardstick::rmse(pred_df,truth = `Actual Gross`,estimate = `Predicted Gross`)

rmse_final
```

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

```
## 1 rmse    standard        1.20
```

```
gg<-pred_df%>%
  ggplot(aes(y=`Actual Gross`,x=`Predicted Gross`,text=paste(title,"<br>",
                                                            `Actual Gross`,
                                                            `Predicted Gross`)))+
  geom_point(alpha=.25,size=.5)+
  scale_x_continuous(trans="log",labels=label_dollar())+
  scale_y_continuous(trans="log",labels=label_dollar())


ggplotly(gg,tooltip="text")
```

Or Just

```
movie_final%>%last_fit(split_data)%>%
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       1.20  Preprocessor1_Model1
## 2 rsq     standard       0.549 Preprocessor1_Model1
```

Choose best model and fit to full data

Once we're sure that we at our very last model, we can get estimates from the full dataset.

```
movie_final<-
  finalize_workflow(movie_wf,
                    select_best(movie_lasso_tune_fit,
                                metric="rmse"))%>%
  fit(mv) ## FULL dataset
```

```
movie_final%>%
  pull_workflow_fit()%>%
  tidy()%>%
  mutate(penalty=prettyNum(penalty))%>%
  kable()
```

```
## Warning: `pull_workflow_fit()` was deprecated in workflows 0.2.3.
## Please use `extract_fit_parsnip()` instead.
```

term	estimate	penalty
(Intercept)	17.9275230	0.005994843
budget	1.0125321	0.005994843
runtime	0.2042725	0.005994843
rating_Not.Rated	-0.1965304	0.005994843
rating_PG	0.0174066	0.005994843
rating_PG.13	-0.0295522	0.005994843
rating_R	-0.2588306	0.005994843
rating_other	-0.0781389	0.005994843
genre_Adventure	0.0000000	0.005994843
genre_Animation	0.1150546	0.005994843
genre_Biography	-0.1007891	0.005994843

term	estimate	penalty
genre_Comedy	-0.0051435	0.005994843
genre_Crime	-0.0556307	0.005994843
genre_Drama	-0.0972500	0.005994843
genre_Horror	0.1880654	0.005994843
genre_other	-0.0035946	0.005994843
year_X2001	-0.0076118	0.005994843
year_X2002	0.0000000	0.005994843
year_X2003	0.0128948	0.005994843
year_X2004	0.0298426	0.005994843
year_X2005	0.0003864	0.005994843
year_X2006	0.0000000	0.005994843
year_X2007	0.0321195	0.005994843
year_X2008	0.0175200	0.005994843
year_X2009	0.0000000	0.005994843
year_X2010	0.0000000	0.005994843
year_X2011	0.0537007	0.005994843
year_X2012	0.0185697	0.005994843
year_X2013	0.0571450	0.005994843
year_X2014	0.0598982	0.005994843
year_X2015	0.0342997	0.005994843
year_X2016	0.0250014	0.005994843
year_X2017	0.0886255	0.005994843
year_X2018	0.0672555	0.005994843
year_X2019	0.0569767	0.005994843
year_other	0.0380641	0.005994843

This is what we would use for any incoming data.Let's say the proposal is to make either a horror or adventure movie for 10 million. We've also been pitched movies that will be rated R– which of course better reflects the director's "artistic vision" and movies that will be rated PG-13. Let's see what our model says about these.

```
newdata<-data_grid(mv,
          title="Data Science 3: The Tuning",
          budget=1e7,
          rating=c("R","PG-13"),
          genre=c("Horror","Adventure"),
          runtime=100,
          year=as_factor(2020))

movie_final%>%
  predict(newdata)%>%
  bind_cols(newdata)%>%
  mutate(low_dollar_amount=dollar(exp(.pred-rmse_final$.estimate)))%>%
  mutate(mean_dollar_amount=dollar(exp(.pred)))%>%
  mutate(hi_dollar_amount=dollar(exp(.pred+rmse_final$.estimate)))
```

```
## # A tibble: 4 x 10
##   .pred title                budget rating genre runtime year  low_dollar_amou~
##   <dbl> <chr>                 <dbl> <chr>  <chr>   <dbl> <fct> <chr>
## 1  17.8 Data Science 3: The ~    1e7 PG-13  Adve~     100 2020  $15,422,413
## 2  18.6 Data Science 3: The ~    1e7 PG-13  Horr~     100 2020  $37,727,841
## 3  17.3 Data Science 3: The ~    1e7 R      Adve~     100 2020  $9,724,327
## 4  18.2 Data Science 3: The ~    1e7 R      Horr~     100 2020  $23,788,615
```

```
## # ... with 2 more variables: mean_dollar_amount <chr>, hi_dollar_amount <chr>
```

What should we recommend as a result?