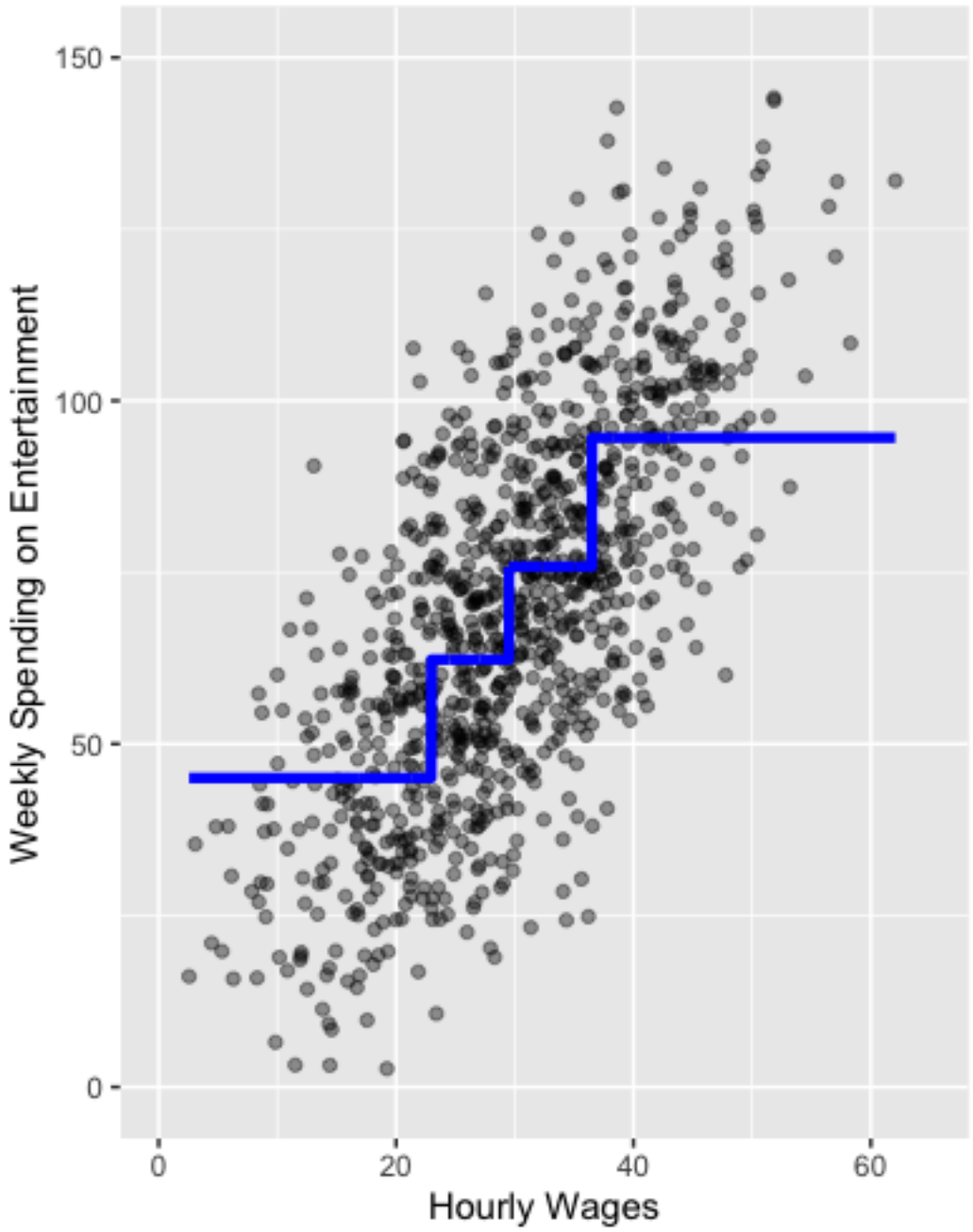
Using Regression for Prediction

Will Doyle

Overview

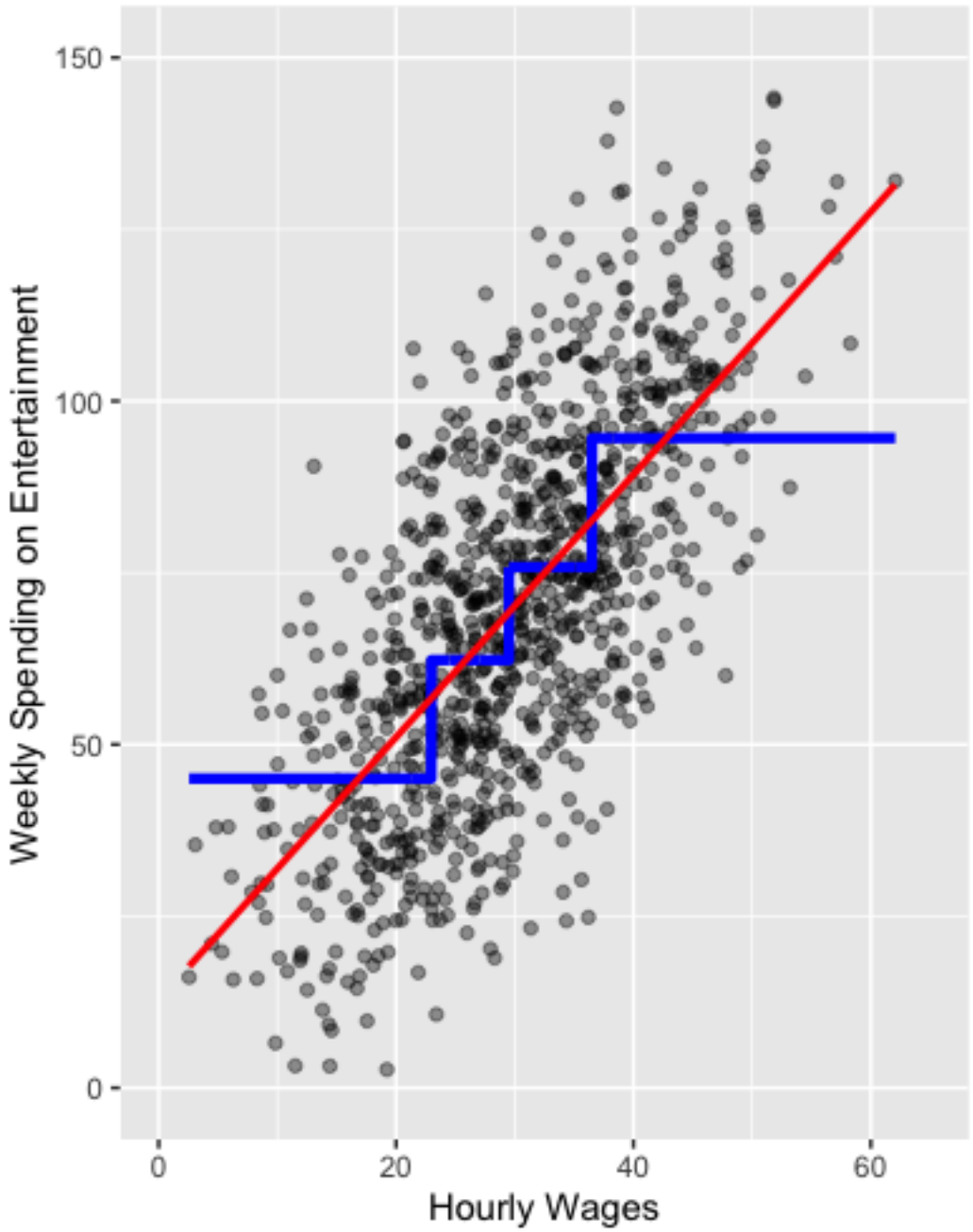
So far, we've been using just the simple mean to make predictions. Today, we'll continue using the simple mean to make predictions, but now in a complicated way. Before, when we calculated conditional means, we did so in certain "groupings" of variables. When we run linear regression, we no longer need to do so. Instead, linear regression allows us to calculate the conditional mean of the outcome at *every* value of the predictor. If the predictor takes on just a few values, then that's the number of conditional means that will be calculated. If the predictor is continuous and takes on a large number of values, we'll still be able to calculate the conditional mean at every one of those values.

As an example, consider the following plot of weekly spending on entertainment (movies, video games, streaming, etc) as a function of hourly wages.



If we're asked to summarize the relationship between wages and spending, we could use conditional means. That's what the blue line above does: for each of the four quartiles of hourly wages, it provides the average spending on entertainment. We could expand this logic and include both more levels of hourly wages and other variables with which to calculate the mean– for example, hourly wages and level of education. The problem is that this approach gives us lots of numbers back.

The graphic below fits a line (in red) to the data:



The line has the general function y=12+(2*x). We're saying that if hourly wages were 0, the person would be predicted to spend 12 on entertainment (who knows where they got it). As hourly wages go up, the line predicts that weekly spending goes up $2 for every $1 increase in hourly wages. The line we fit to this data summarizes the relationship using just two numbers: the intercept (value of the y when x=0) and slope– the amount that y increases as a function of x. We call the slope the *coefficient* on x.

The general model we posit for regression is as follows:

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ...\beta_k x_k + \epsilon$$

It's just a linear, additive model. Y increases or decreases as a function of x, with multiple x's included. ϵ is the extent to which an individual value is above or below the line created. β is the amount that Y increases

or decreases for a one unit change in x. With the simplifying assumption of linearity, we can summarize what we know about the relationship between the independent and dependent variable in a very parsimonious way. The trade-off is that an assumption of linearity may not always be warranted.

Motivation: predicting movie revenues

"Nobody knows anything..... Not one person in the entire motion picture field knows for a certainty what's going to work. Every time out it's a guess and, if you're lucky, an educated one."

-William Goldman, screenwriter of The Princess Bride and All the President's Men.

In this series of lectures we're going to see if we can predict which movies will bring in more money. Predicting movie revenues is known to be a very difficult problem, as some movies vastly outperform expectations, while other (very expensive) movies flop badly. Unlike other areas of the economy, it's not always easy to know which characteristics of movies are associated with higher gross revenues. Nevertheless, we shall persist!

It's typical for an investor group to have a model to understand the range of performance for a given movie. Investors want to know what range of return they might expect for an investment in a given movie. We'll try and get started on just such a model.

The Data

Load in libraries.

```
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(plotly)
library(scales)
```

Load in data.

```
mv<-read_rds("mv.Rds")%>%
  filter(!is.na(budget))

glimpse(mv)
```

```
## Rows: 3,191
## Columns: 20
## $ title         <chr> "Almost Famous", "American Psycho", "Gladiator", "Requie~
## $ rating        <chr> "R", "R", "R", "Unrated", "R", "PG-13", "R", "PG-13", "P~
## $ genre         <chr> "Adventure", "Comedy", "Action", "Drama", "Mystery", "Ad~
## $ year          <dbl> 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 20~
## $ released      <chr> "September 22, 2000 (United States)", "April 14, 2000 (U~
## $ score         <dbl> 7.9, 7.6, 8.5, 8.3, 8.4, 7.8, 6.2, 6.4, 5.7, 7.4, 6.4, 7~
## $ votes         <dbl> 260000, 514000, 1400000, 786000, 1200000, 542000, 238000~
## $ director      <chr> "Cameron Crowe", "Mary Harron", "Ridley Scott", "Darren ~
```

```
## $ writer        <chr> "Cameron Crowe", "Bret Easton Ellis", "David Franzoni", ~
## $ star          <chr> "Billy Crudup", "Christian Bale", "Russell Crowe", "Elle~
## $ country       <chr> "United States", "United States", "United States", "Unit~
## $ budget        <dbl> 93289619, 10883789, 160147179, 6996721, 13993443, 139934~
## $ gross         <dbl> 73677478, 53278578, 723586629, 11490339, 62266278, 66800~
## $ company       <chr> "Columbia Pictures", "Am Psycho Productions", "Dreamwork~
## $ runtime       <dbl> 122, 101, 155, 102, 113, 143, 88, 130, 100, 104, 130, 16~
## $ id            <dbl> 877, 64, 1163, 2050, 52, 3795, 5544, 9301, 1093, 101, 68~
## $ imdb_id       <chr> "0181875", "0144084", "0172495", "0180093", "0209144", "~
## $ bechdel_score <dbl> 3, 3, 0, 3, 1, 2, 3, 2, 3, 1, 2, 3, 3, 0, NA, NA, 1, NA,~
## $ boxoffice_a   <dbl> 50586063, 23431686, 291849463, 5652546, 39717849, 363257~
## $ language      <chr> "English, French", "English, Spanish, Cantonese", "Engli~
```

This data comes from the Internet Movie Data Based, with contributions from several other sources.

Name	Description
title	Film Title
rating	MPAA Rating
genre	Genre: Adventure, Action etc
year	Year
released	Date released
score	IMDB Score
votes	Votes on IMDB
director	Director
writer	Screenwriter (top billed)
star	Top billed actor
country	Country where produced
runtime	Running time in minutes
id	Alternate ID
imdb_id	IMDB unique ID
bechdel_score	1=two women characters, 2=they speak to each other, 3= about something other than a man, 0=none of the above
box_office_a	Box office take
language	Languages spoken in the movie

Can we make money in the movie business?

There are a couple of ways we can answer this question. First of all, let's look at gross minus budget (I'm avoiding the word profit because movie finance is deeply weird).

```
mv%>%
  mutate(gross_less_budget=gross-budget)%>%
  summarize(dollar(mean(gross_less_budget,na.rm=-TRUE)))
```

```
## # A tibble: 1 x 1
##   `dollar(mean(gross_less_budget, na.rm = -TRUE))`
##   <chr>
## 1 $113,277,345
```

Looks good! But wait a second, some movies must lose money right? Let's see how many that is:

```
mv%>%
  mutate(gross_less_budget=gross-budget)%>%
  mutate(made_money=ifelse(gross_less_budget>0,1,0))%>%
```

```
  group_by(made_money)%>%
  drop_na()%>%
  count()%>%
  ungroup()%>%
  mutate(prop=n/sum(n))
```

```
## # A tibble: 2 x 3
##   made_money     n  prop
##        <dbl> <int> <dbl>
## 1          0   336 0.170
## 2          1  1645 0.830
```

Oof, so something like 18 percent of movies in this dataset lost money. How much did they lose?

```
mv%>%
  mutate(gross_less_budget=gross-budget)%>%
  mutate(made_money=ifelse(gross_less_budget>0,1,0))%>%
  group_by(made_money)%>%
  summarize(dollar(mean(gross_less_budget)))%>%
  drop_na()
```

```
## # A tibble: 2 x 2
##   made_money `dollar(mean(gross_less_budget))`
##        <dbl> <chr>
## 1          0 -$16,705,383
## 2          1 $155,755,006
```

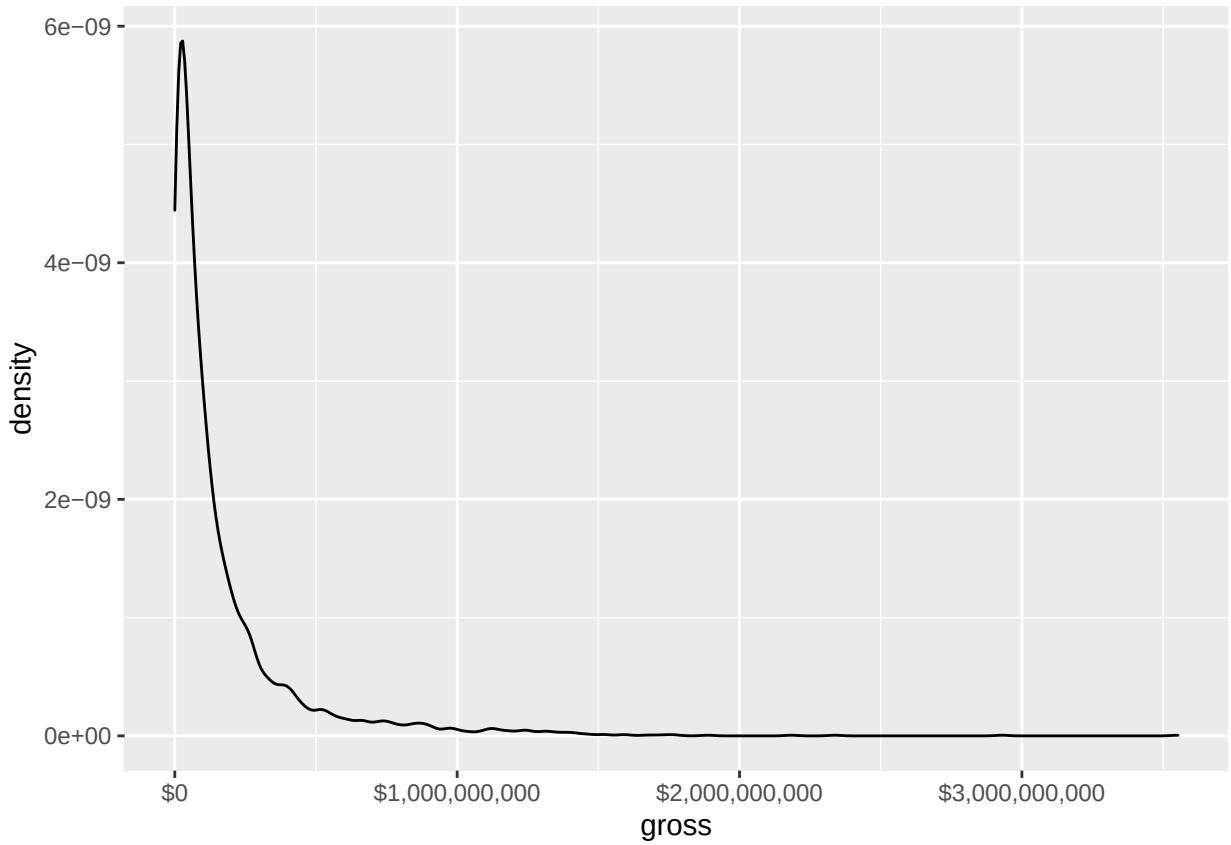
16 million on average. We better get this right! Okay, Let's get started on our model.

Dependent Variable

Our dependent variable will be gross from the movie which is a measure of revenue from all sources: box office, streaming, dvds etc.

```
mv%>%
  ggplot(aes(gross))+
  geom_density()+
  scale_x_continuous(labels=dollar_format())
```

```
## Warning: Removed 12 rows containing non-finite values (stat_density).
```



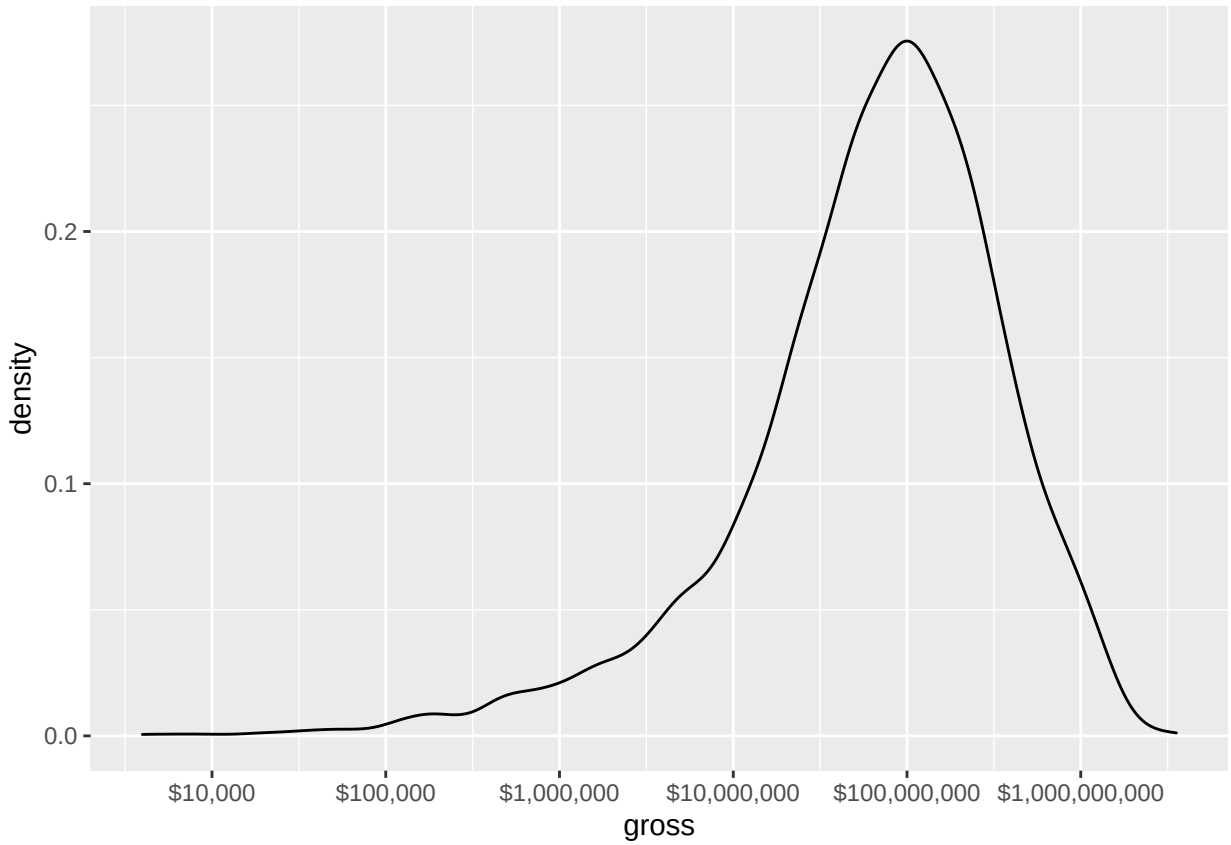
Well, that looks weird. Some movies make A LOT. Most, not so much. We need to work with this on an appropriate scale. When working with almost any data that has to do with money we're going to see things on an exponential scale. Our solution will be to transform this to be on a "log scale". Really what we're doing is taking the natural log of a number, which is the amount that Euler's constant e would need to be raised to in order to equal the nominal amount.

$$log_e(y) = x, \equiv e^x = y$$

Let's transform gross to be on a log scale. We can do this within ggplot to allow for some nicer labeling.

```
mv%>%
  ggplot(aes(x=gross))+
  geom_density()+
  scale_x_continuous(trans="log",
                     labels=dollar_format(),
                     breaks=breaks_log(n=6))
```

```
## Warning: Removed 12 rows containing non-finite values (stat_density).
```



That looks somewhat better. Notice how the steps on the log scale imply very different amounts. Effectively what we're doing is changing our thinking to be aboout percent changes. A 10% increase in gross above 10,000 is 1000. A 10% increase in gross above 100,000 is 10,000. On a log scale, these are (sort of) equivalent steps. Transforming this on the front end will have some implications on the back end, but we'll deal with that in good time.

Gross as a Function of Budget

It's quite likely that gross revenues will be related to the budget, but how likely? Let's plot the relationship and find out.

```
gg<-mv%>%
  ggplot(aes(x=budget,y=gross,text=paste(title,"<br>",
                                         "Budget:", dollar(budget), "<br>",
                                         "Gross:" ,dollar(gross))))+
    geom_point(size=.25,alpha=.5)+

  scale_x_continuous(trans="log",
                     labels=label_dollar(),
                     breaks=log_breaks())+
   scale_y_continuous(trans="log",
                     labels=label_dollar(),
                     breaks=log_breaks())+
  xlab("Budget")+
  ylab("Gross")

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

You can navigate this plot and find out what the titles are. This is made possible by the ggplotly command, which uses the plotly library.

To make things easier, I'm going to create a new variable that just the log of gross revenues

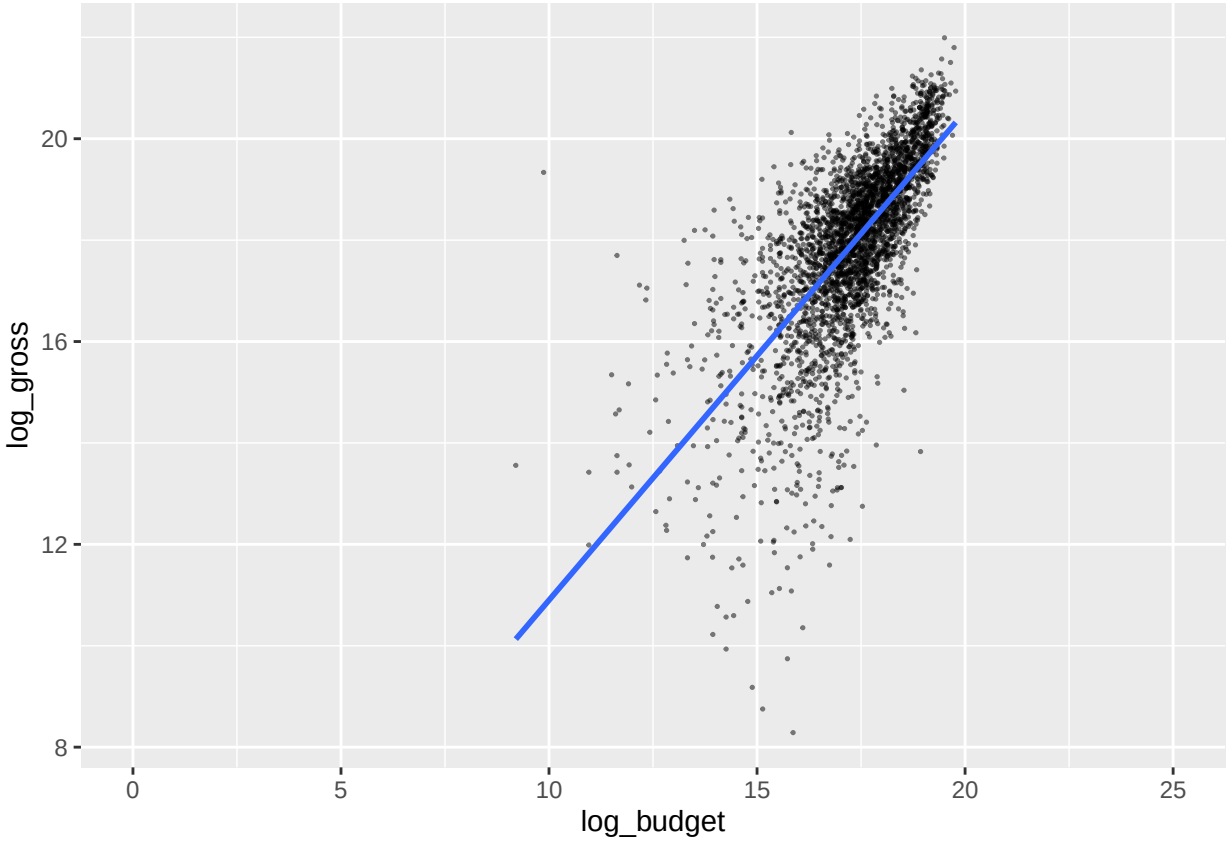
```
mv<-mv%>%
  mutate(log_gross=log(gross))
```

Basic Linear Model

This plot shows a line fit to the data, with the log of budget on the x axis and the log of gross budget on the y axis.

```
mv%>%
  mutate(log_budget=log(budget))%>%
  ggplot(aes(y=log_gross,x=log_budget))+
  geom_point(size=.25,alpha=.5)+
  geom_smooth(method="lm",se=FALSE)+
  xlim(0,25)
```

```
## `geom_smooth()` using formula 'y ~ x'
## Warning: Removed 12 rows containing non-finite values (stat_smooth).
## Warning: Removed 12 rows containing missing values (geom_point).
```



Does an assumption of linearity work in this case? Why or why not? Can we summarize this whole dataset in just two numbers?

Root Mean Squared Error

To understand how far off we are, we need to summarize our errors. We will use a very standard method, Root Mean Squared Error, or RMSE. An error term is the distance between each point and its prediction. We square the errors, take the average, then take the square root of the result. The name RMSE is exactly what RMSE is– neat, huh?

$$RMSE(\hat{Y}) = \sqrt{1/n \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2}$$

The errors in the above graphic are simply the vertical distances from the actual point to the point plotted by the line. We take those distance (positive for the points above the line, negative for the points below the line) and square them, then take the average of all of those squares, then take the square root of that average: voila, rmse.

Luckily we have an r function that can do this for us, helpfully named `rmse`.

Calculating rmse from the data we used to fit the line is actually not correct, because it doesn't help us learn about out of sample data. To solve this problem, we need another concept: training and testing.

Training and Testing

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 only worked 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. We could then train our model on half the data, then test it on the other portion. This would tell us whether our measure of model fit (e.g. rmse) is similar or different when we apply our model to out of sample data. That's what we're going to do in this lesson. We'll split the data randomly in two, with one part used to train our models and the other part used to test the model against out of sample data.

I use the `initial_split` command to designate the way that the data should be split– in this case 75/25. The testing data (which is a random quarter of the original dataset) is stored as `mv_test`. The training data is stored as `mv_train`. We'll run our models on `mv_train`.

Note: the `set.seed` command ensures that your random split should be the same as my random split.

Training and Testing Data

```
set.seed(35202)

split_data<-mv%>%initial_split() # default is .75

mv_train<-training(split_data)

mv_test<-testing(split_data)
```

Start a Workflow

For much of our modeling purposes, we're going to use the `tidymodels` approach to fiting models. It's a really nice way to structure our approach to fitting models. The key to the tidynmodels approach is the workflow: a standard set of steps for fitting a model and understanding its predictive capacity. We'll start by creating our modeling workflow, called `movie_wf`.

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

Define Model

The next step is to define the model. The type of model I want is a linear regression, which I specify below. This is also sometimes called "ordinary least squares" regression.

```
lm_fit <-
  linear_reg() %>%
  set_engine("lm")%>%
  set_mode("regression")
```

..and then we'll add that model to our workflow

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

Set Formula

To start training our model, we need to specify a formula. The dependent variable always goes on the left hand side, while the independent variable or variables always go on the right hand side. The formula below is for a model with gross revenues on the left hand side, and budget on the right hand side.

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

Define the Recipe

A recipe takes the formula and applies it to the dataset and then specifies a set of *pre-processing* steps: steps that the analyst wants to take to adjust the data prior to running the model. In our model, we've decided that budget and gross should be log transformed before being included.

```
movie_rec<-recipe(movie_formula,mv_train)%>%
    step_log(budget)
```

Having specified how the data should be processed, we can add the recipe to the workflow.

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

Fit Model to the Data

Now we're ready to actually fit the model to the data. In the `fit` command below I specify which formula should be used and which dataset to fit the linear regression specified above.

```
movie_fit<-movie_wf%>%
  fit(data=mv_train)
```

Look at the Results

To examine the results, we can use the `tidy` command.

```
movie_fit%>%
  pull_workflow_fit()%>%
  tidy()
```

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

```
## # A tibble: 2 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)    1.33     0.351      3.78 0.000158
## 2 budget         0.960    0.0202     47.4  0
```

What this tells us is that for each additional one percent increase in budgets, gross is predicted to increase by about .95 percent.

This means different things at different budgets. It also means that there aren't increasing returns to scale, at least looking at this sample of movies.

To look at measures of model fit, we can use `glance`:

```
movie_fit%>%
  pull_workflow_fit()%>%
  glance()
```

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

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.485         0.485  1.26     2250.       0     1 -3946. 7898. 7915.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

Evaluate Predictions in the Testing Data

All of that information is about how the model performed in the *training* data. Now it's time to take a look at how it performs in the *testing* data. To do this, we use the model to make predictions in the testing data

then calculate RMSE. What's happening here is that R is using the simple calculation from our previous model: intercept plus slope of the line (coefficient) on budget to predict our outcome of log gross revenues.

This code adds predictions to the dataset.

```
mv_test<-movie_fit%>%
  predict(mv_test)%>% ## Add predictions
  rename(.pred1=.pred)%>%
  bind_cols(mv_test) ## add to the existing testing dataset
```

With those predictions in hand, we can calculate the rmse in the testing dataset.

```
mv_test%>%
  rmse(truth=log_gross,estimate=.pred1)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        1.33
```

Is this good? Who knows! RMSE is very context dependent. One way to think about it for us is that the model predicts that a 10 million dollar investment will generate about 15 million: sounds good, right?

```
quick_est<-exp(1.2+ (.95* log(1e7)))
dollar(quick_est)
```

```
## [1] "$14,830,418"
```

Except the rmse says that gross could be between 51 million (yay) and 4.2 million. Right now, that's our prediction– a 10 million dollar investment, base on this model, will make somewhere between 4.2 million and 51 million. Your investors are probably not thrilled and are thinking about just putting the money in T Bills.

```
quick_upper_bound<-exp(1.2+ .95*log(1e7) +1.28)

dollar(quick_upper_bound)
```

```
## [1] "$53,339,669"
```

```
quick_lower_bound<-exp(1.2+ .95* log(1e7) -1.28)

dollar(quick_lower_bound)
```

```
## [1] "$4,123,409"
```

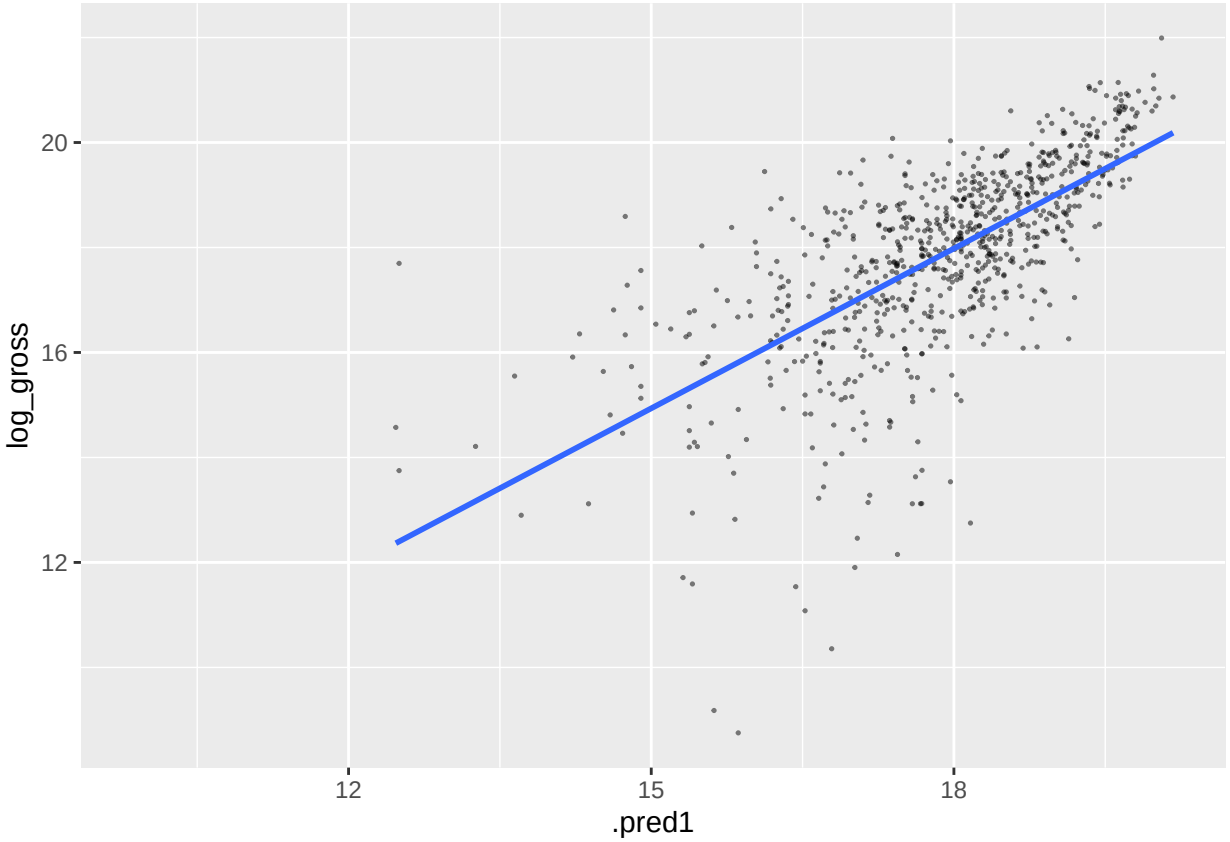
Here's a comparison of the predicted value with the actual value in the testing dataset.

```
mv_test%>%
  ggplot(aes(y=log_gross,x=.pred1))+
  geom_point(size=.25,alpha=.5)+
  geom_smooth(method="lm",se=FALSE)
```

```
## `geom_smooth()` using formula 'y ~ x'

## Warning: Removed 6 rows containing non-finite values (stat_smooth).

## Warning: Removed 6 rows containing missing values (geom_point).
```



Quick Exercise Run a regression using a single different continuous predictor (there aren't many in this dataset). Calculate rmse and see if you can beat my score.

```
# INSERT CODE HERE
```

Multiple regression

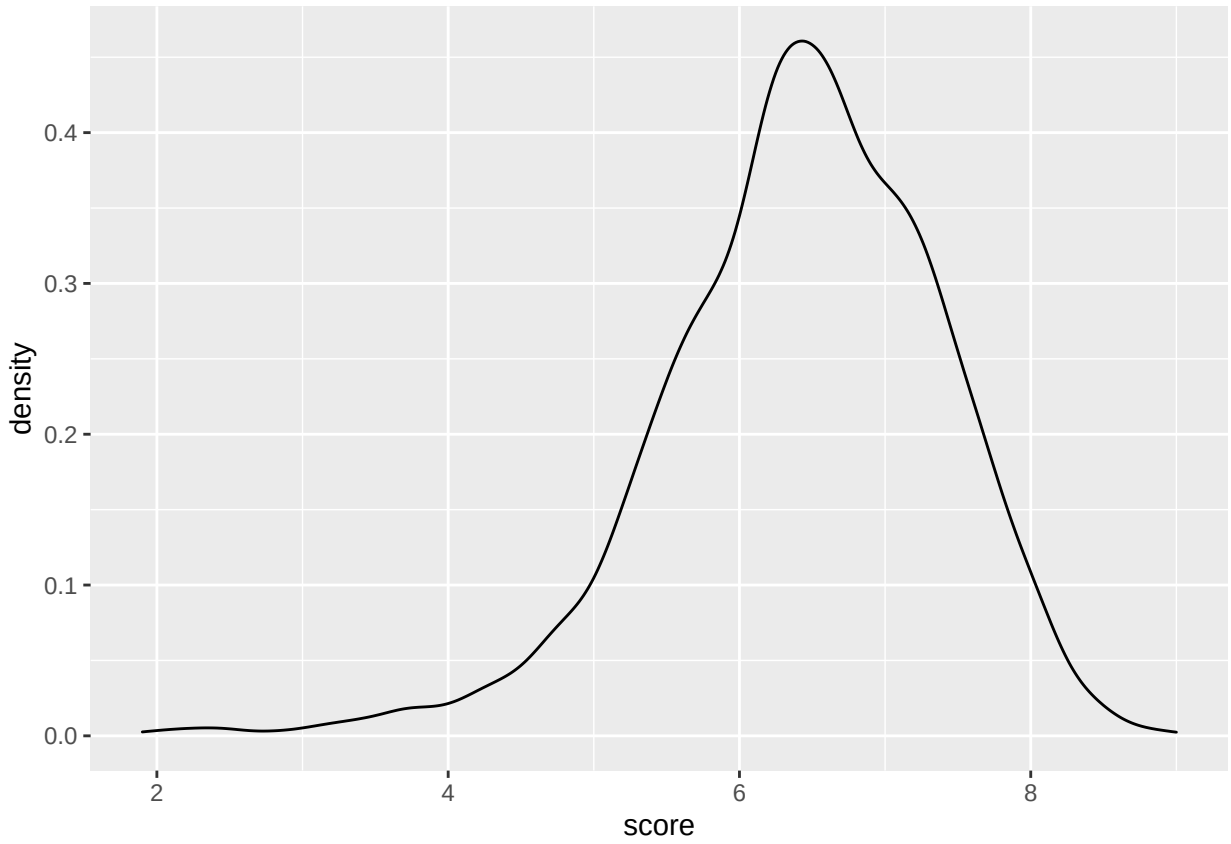
We can continue refining our model by adding more variables. Regression with more than one predictor is called multiple regression. The nice thing is that the steps will all be the same, we just need to change the formula and the rest stays pretty much as it was.

I'm going to add IMDB score as a predictor. While this isn't a perfect reflection of critical reception, the basic idea here is that better-reviewed movies may be more likely to make more money back.

Let's take a look at IMDB scores:

```
mv%>%
  ggplot(aes(x=score))+
  geom_density()
```

```
## Warning: Removed 3 rows containing non-finite values (stat_density).
```



Unlike budget or gross, there isn't a strong need to transform the score.

Set Formula

I add score to the formula simply by using "+" in the formula.

```
mv_formula<-as.formula("log_gross~budget+score")
```

I now need to update the recipe to include the new formula. Notice that I am not using the `step_log` command on the score variable, based on what I observed above.

```
mv_recipe<-recipe(mv_formula,mv_train)%>%
  step_log(budget)
```

Now I can update the worklfow with the new formula.

```
movie_wf<-movie_wf%>%
  update_recipe(mv_recipe)
```

And with that, I'm ready to fit the model again.

```
movie_fit<-movie_wf%>%
  fit(data=mv_train)
```

Let's take a look at the results.

```
movie_fit%>%
  pull_workflow_fit()%>%
  tidy()
```

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

```
## # A tibble: 3 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)   -0.679    0.387      -1.75 7.94e- 2
## 2 budget         0.966    0.0197     48.9  0
## 3 score          0.297    0.0266     11.1  3.57e-28
```

There is a positive and statistically significant relationship between score and gross revenues, *after* controlling for budget. So this means that at any budget level, higher ratings on the IMDB scale predict higher gross. Small-budget movies with higher scores, and high-budget movies with higher scores are all predicted to have higher gross revenues.

```
movie_fit%>%
  pull_workflow_fit()%>%
  glance()
```

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

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.511         0.511  1.23     1245.       0     2 -3885. 7779. 7802.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

Calculate RMSE

```
mv_test<-movie_fit%>%
  predict(mv_test)%>%
  rename(.pred2=.pred)%>%
  bind_cols(mv_test)
```

```
mv_test%>%
  rmse(truth=log_gross,estimate=.pred2)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        1.30
```

That . . . didn't do much! We're not getting a big increase in model fit as a result of including score, even though it IS related to the outcome.

What does this rmse *actually* mean? We can get a sense by using the parameters of the model to calculate the predicted gross for a 10 million dollar movie with average IMDB scores, then add or subtract the rmse.

```
mv_sum<-mv%>%
  summarize(intercept=1,
            mean_log_budget=log(1e7),
            mean_score=mean(score,na.rm=TRUE))%>%
  pivot_longer(cols=everything(), names_to="type")

mv_rmse<-mv_test%>%
  rmse(truth=log_gross,estimate=.pred2)
```

```
movie_fit%>%
  pull_workflow_fit()%>%
  tidy()%>%
  select(term,estimate)%>%
  bind_cols(mv_sum)%>%
  mutate(product=estimate*value)%>%
  summarize(prediction=sum(product))%>%
  mutate(low_range=dollar(exp(prediction-mv_rmse$.estimate)),
         mid_range=dollar(exp(prediction)),
         hi_range=dollar(exp(prediction+mv_rmse$.estimate))
  )
```

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

```
## # A tibble: 1 x 4
##   prediction low_range  mid_range   hi_range
##        <dbl> <chr>      <chr>       <chr>
## 1       16.8 $5,406,801 $19,798,245 $72,495,827
```

So, for a 10 million dollar investment, this model says we could make between 5 million and 66 million. Remember William Goldman?

Quick Exercise Run a regression using two different predictors. Calculate rmse and see if you can beat my score.

```
# INSERT CODE HERE
```

You MUST remember: correlation is not causation. All you can pick up on using this tool is associations, or common patterns. You can't know whether one thing causes another. Remember that the left hand side variable could just as easily be on the right hand side.