Predicting Tip Size

PSAT 131 Final Project

Bella Genolio

2024-05-07

Contents

Introduction

The purpose of this project is to predict the price of a tip at a restaurant. Tips are useful to predict since they are often a large portion of a waiter's paycheck. Waiters cannot always anticipate the amount of money they will make in a shift, so this model will make predicting that value easier. This project will also examine the values that make these predictions possible. The variables we will be examining include whether the bill payer was a smoker, their gender, the total bill, how much the bill cost per person, the time of the meal, the day of the week, and the size of the party. Some questions that hopefully will be answered by the end of this project include:

- What predictors will do the best job of predicting tips?
- Which predictors are irrelevant?
- What model allows us to best predict the tip value?
- How accurately can we predict the tip value?

Our main goal is to predict the variable `tip` using our other variables, let's discuss how we will fit the right model by the end of this project. Once we load the data, we will have to see if there are any missing values, add any relevant predictors, and remove any unnecessary values. After tidying the data we can look at how the predictors interact by using exploratory data analysis. This visualization will help further understand the data and enable us to fit the best models possible. We will then split the data into testing and training sets stratified on `tip` to ensure we have equal proportions of the data for both training our models and testing their ability. Next we can build a recipe for our models to use. Then we will create the models, workflows, and tuning grids for each of the different models we want to use. Since the value we are predicting is quantitative, we want to use regression models. We will fit each of our models to the training set using k-fold cross validation to ensure we have a well trained model. Once we tune our models, we can look at their plots and the best results of the tuning. This comparison will allow us to choose our final model, to do this we will compare rmse's and see which is the lowest. We will then fit the final model to our training set one last time and then augment it to our testing set. Finally, we will look at the final results of our model to see how it did!

Data Citation

The source of the data set is the Kaggle data set, "Waiter's Tips Dataset" by Zahra Amini.This data set is a collection of information about tips recorded by one waiter over a couple of months working in a restaurant.

Loading and Exploring Raw Data

Here we will load the necessary packages for this project

```r
library(ggplot2)
library(tidyverse)
library(tidymodels)
library(corrplot)
library(corrr)
library(ggthemes)
library(visdat)
library(dbplyr)
library(ranger)
library(vip)
library(janitor)
tidymodels_prefer()
```

Now we can load the data set provided by Kaggle and inspect the data.

```r
tips <- read.csv("~/pstat_131_project_BG/tips.csv")
head(tips)
```

```
##   total_bill  tip    sex smoker day   time size price_per_person
## 1      16.99 1.01 Female     No Sun Dinner    2             8.49
## 2      10.34 1.66   Male     No Sun Dinner    3             3.45
## 3      21.01 3.50   Male     No Sun Dinner    3             7.00
## 4      23.68 3.31   Male     No Sun Dinner    2            11.84
## 5      24.59 3.61 Female     No Sun Dinner    4             6.15
## 6      25.29 4.71   Male     No Sun Dinner    4             6.32
##            Payer.Name    CC.Number Payment.ID
## 1 Christy Cunningham 3.560325e+15    Sun2959
## 2     Douglas Tucker 4.478071e+15    Sun4608
## 3     Travis Walters 6.011812e+15    Sun4458
## 4   Nathaniel Harris 4.676138e+15    Sun5260
## 5        Tonya Carter 4.832733e+15    Sun2251
## 6          Erik Smith 2.131404e+14    Sun9679
```

Upon inspection, we can see that many of the categorical variables are stored as characters and we can turn them into factors. We also can make the names easier to refer to by using `clean_names()`

```
tips <- tips %>%
  mutate(sex = as.factor(sex),
         smoker = as.factor(smoker),
         day = as.factor(day),
         time = as.factor(time))

tips <- clean_names(tips)
```

We can also look at the size of the data set in order to see how many observations and variables we are working with

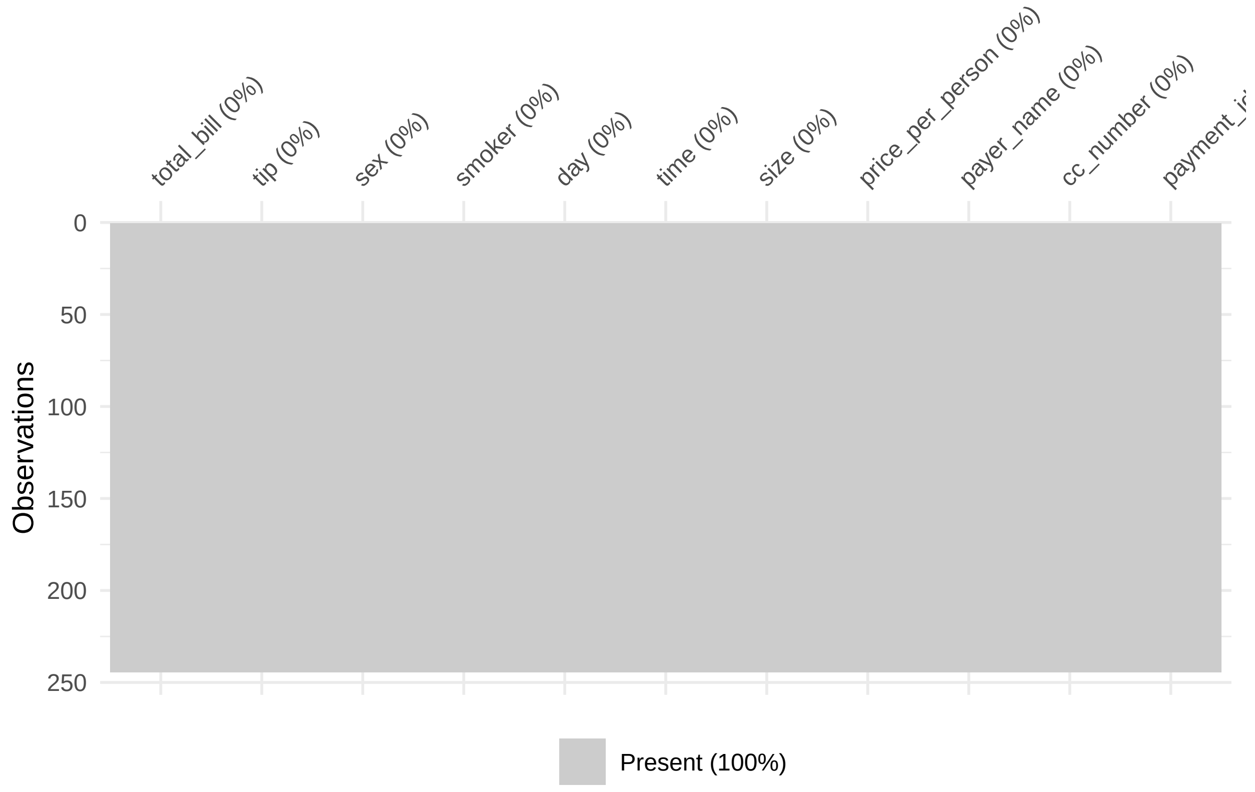
```
dim(tips)
```

```
## [1] 244  11
```

There are 244 observations and 11 variables, a very manageable sized data set, though not all of these variables will necessary be useful for predicting the tip amount. We will keep them in for now and trim as we look at the data more.

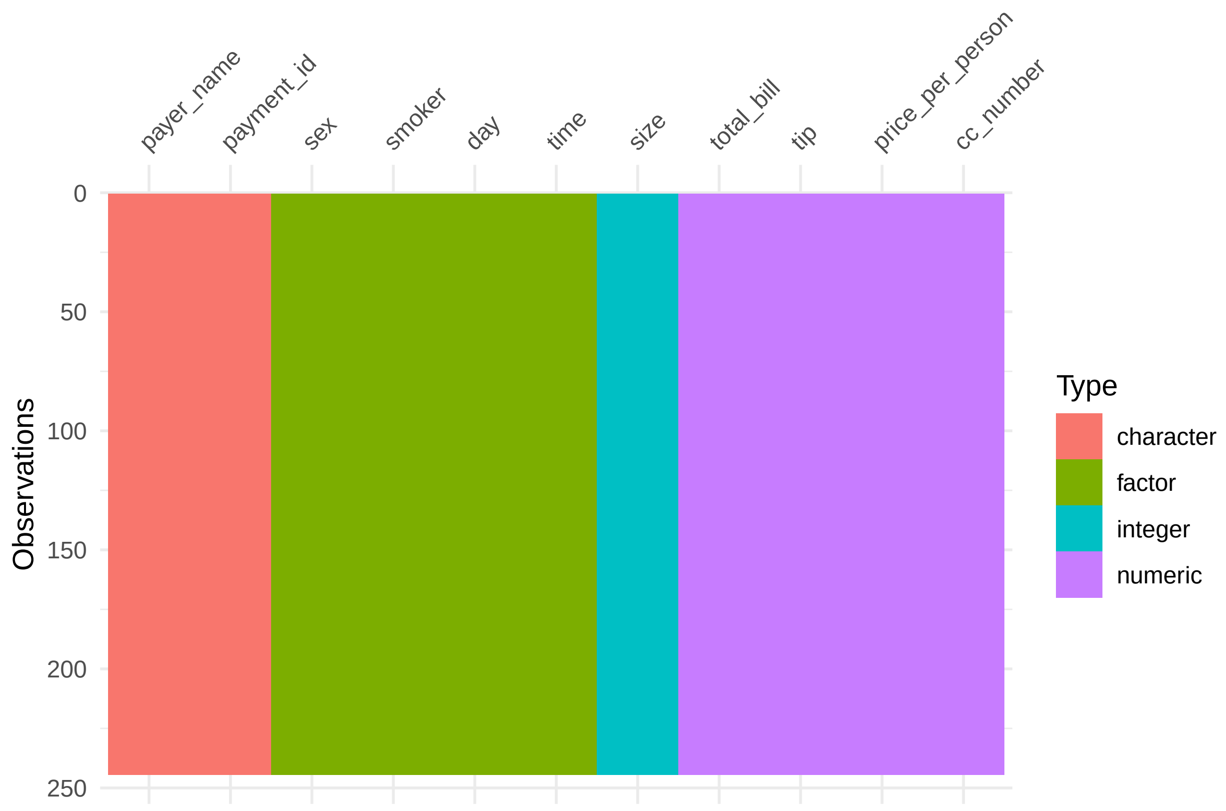
To start, let's examine the missing data from the data set. We can use `vis_miss` in order to visualize what's missing.

```
vis_miss(tips)
```



Luckily it looks like we have no missing data in this data set! It is a smaller data set, so missing data would have to be imputed. Imputation is when you substitute missing data with a different value, usually something within the data set that has other similar predictor values. We can check our data another way using `vis_dat()` which will give us an area with `NA` for missing values and will tell us what types of variables our data breaks down into.

```
vis_dat(tips)
```



Again, we have no missing data, so there is no `NA` column, but this is still useful to see the breakdown of the different types of variables.

Let's make a new data value called `percentage` that takes the `tip` and divides it by the `total_bill` to see what percent of the bill the tip is. This will mainly help with exploratory data analysis, we cannot use it for our models since it is directly correlated with our response variable `tip`.

```
tips$percent <- 100*(tips$tip/tips$total_bill)
```

For this data set, we are predicting the numeric value of tips, so we will be taking a regression approach. Intuitively we can think of what variables will be useful and not useful in predicting tips. We can remove unnecessary values - `CC.Number`, `Payer.Name`, `Payment.ID`. These are unique to each of the observations and will not help us predict the tip amount.

```
tips <- tips[, c("total_bill", "tip", "sex", "smoker", "day", "time", "size", "price_per_person", "perc

head(tips)
```

```
##   total_bill  tip    sex smoker day   time size price_per_person   percent
## 1      16.99 1.01 Female     No Sun Dinner    2             8.49  5.944673
## 2      10.34 1.66   Male     No Sun Dinner    3             3.45 16.054159
## 3      21.01 3.50   Male     No Sun Dinner    3             7.00 16.658734
## 4      23.68 3.31   Male     No Sun Dinner    2            11.84 13.978041
## 5      24.59 3.61 Female     No Sun Dinner    4             6.15 14.680765
## 6      25.29 4.71   Male     No Sun Dinner    4             6.32 18.623962
```

Looking at the data, we have no missing data, we eliminated any unnecessary values and added a relevant value. Take a look at the `tips_codebook.txt` file to get information on each of the variables. It seems like we are all set to start exploring our data!

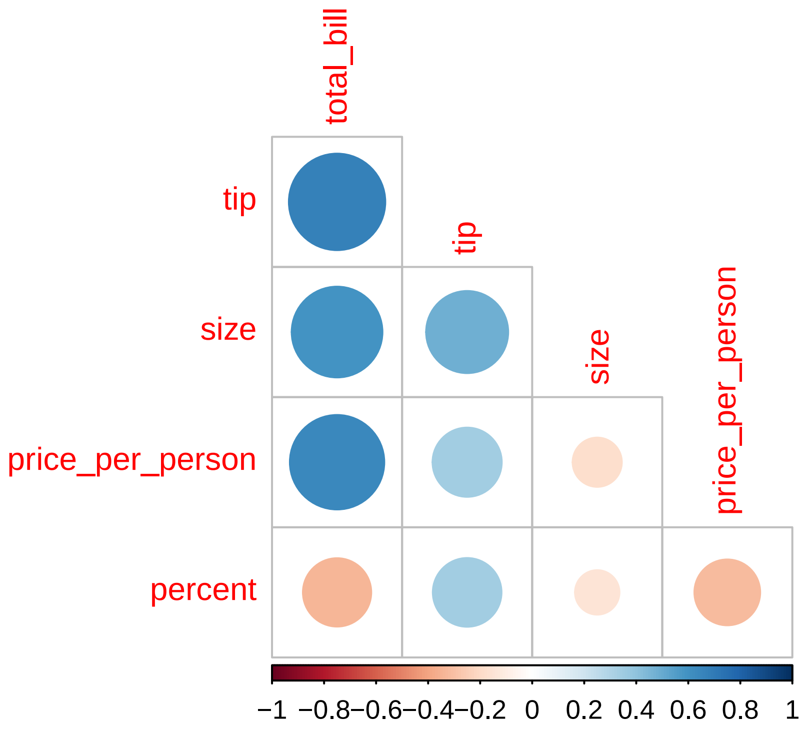
Exploratory Data Analysis

Now that we have examined and tidied the data set, we can visualize our data to begin examining patterns and trends in the data. Keep in mind our response variable is `tip` .

Correlation Plot

We can use a correlation plot to look at the correlation between all of our variables.

```
#start by choosing all numeric and integer values and turning them into a correlation matrix
tips %>%
  select(is.numeric) %>%
  cor() %>%
  corrplot(type = "lower", diag = F)
```



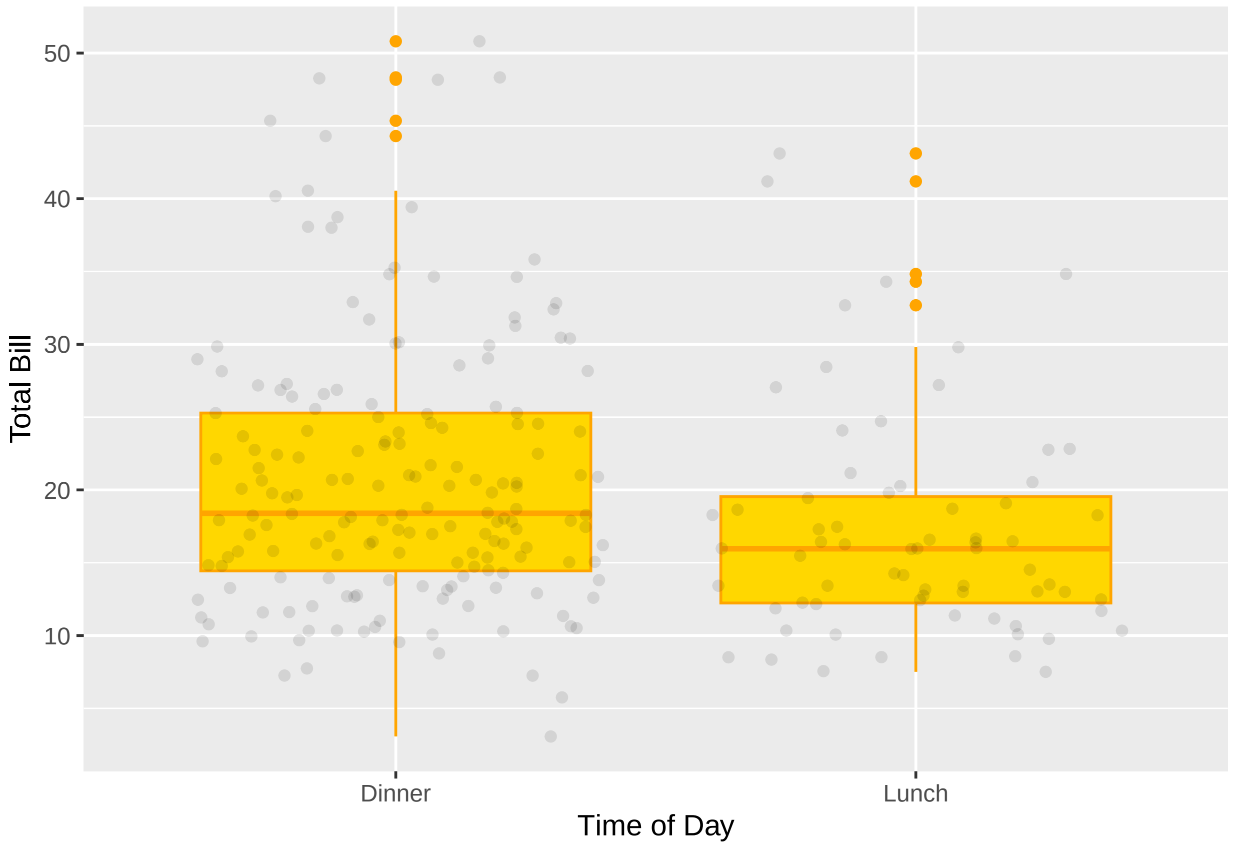
This correlation plot shows us that `tip`, `size`, `price_per_person` are all highly positively correlated with the `total_bill`. `size` is slightly negatively correlated with `price_per_person`. `percent` is negatively correlated with `total_bill` and `price_per_person`, which is understandable since the `percent` is `tip` over `total_bill`. None of the correlations appear to be over 0.8 which means we won't have to combine any of the variables to decrease complexity and avoid error.

Box Plot

Lets examine how one of our non numeric values, time of day, impacts the total bill.

```
ggplot(tips, aes(factor(time), total_bill)) +
  geom_boxplot(fill = "gold", color = "orange") +
  geom_jitter(alpha = 0.1)+
```

```r
  ylab("Total Bill") +
  xlab("Time of Day")
```

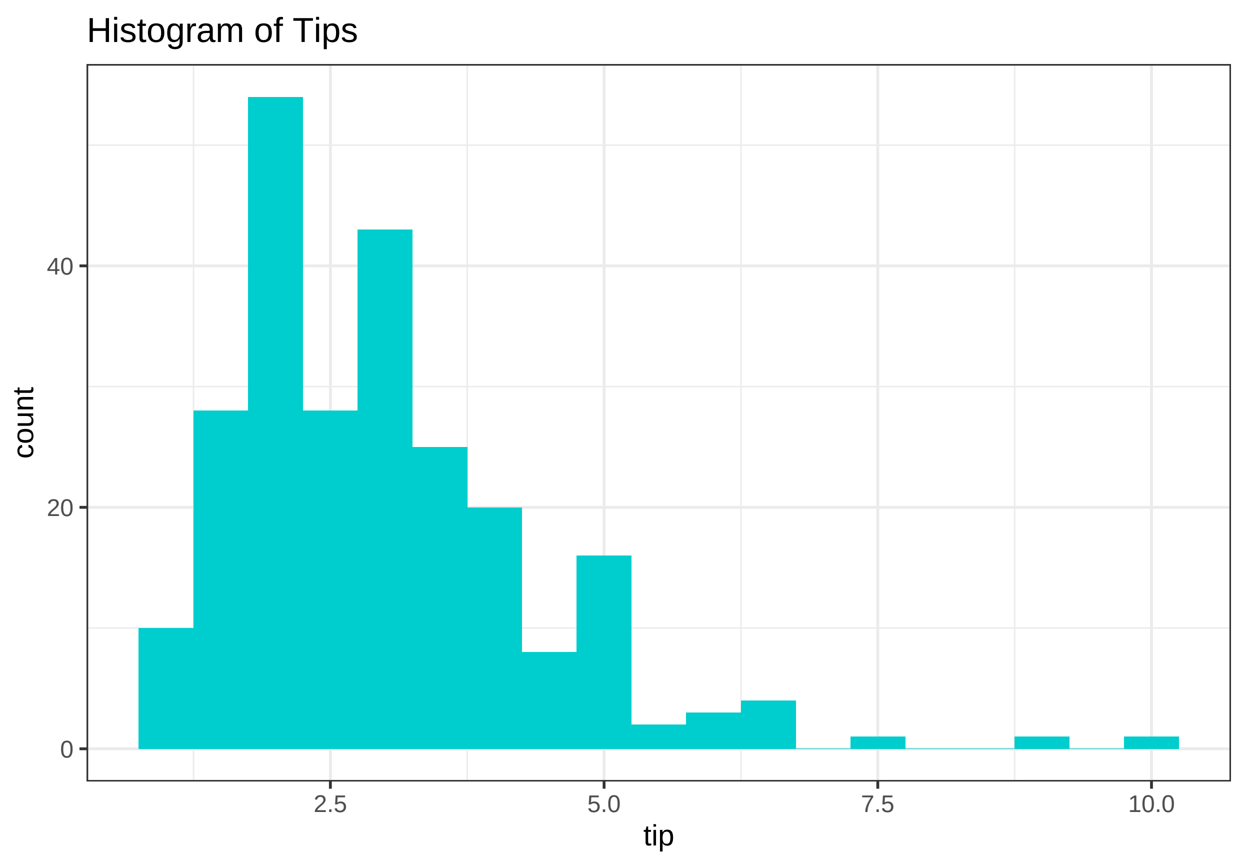


We can see that more of the data is collected from dinner and the total bill amount tends to be higher than at lunch. This means that the dinner data will be more useful in predicting new observations since there is more of it.

Histograms

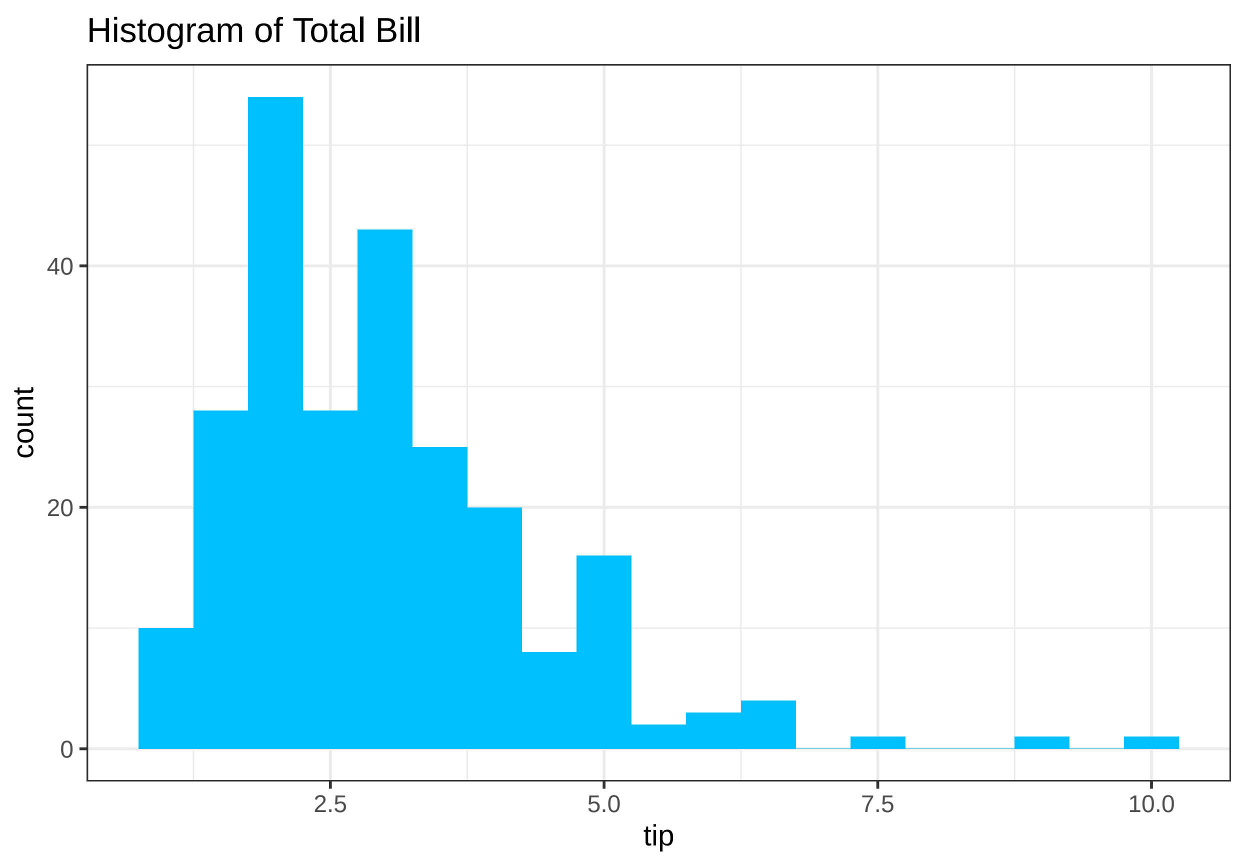
We also can look at a histogram of tip values and compare it to a histogram of the total bill.

```r
ggplot(tips, aes(x = tip)) +
  geom_histogram(binwidth = 0.5, fill = "cyan3") +
  theme_bw() +
  labs(title = "Histogram of Tips")
```



Just looking at the tips histogram, most of the tips are from 50 cents to 5 dollars. The most common value is one dollar.

```r
ggplot(tips, aes(x = tip)) +
  geom_histogram(binwidth = 0.5, fill = "deepskyblue") +
  theme_bw() +
  labs(title = "Histogram of Total Bill")
```

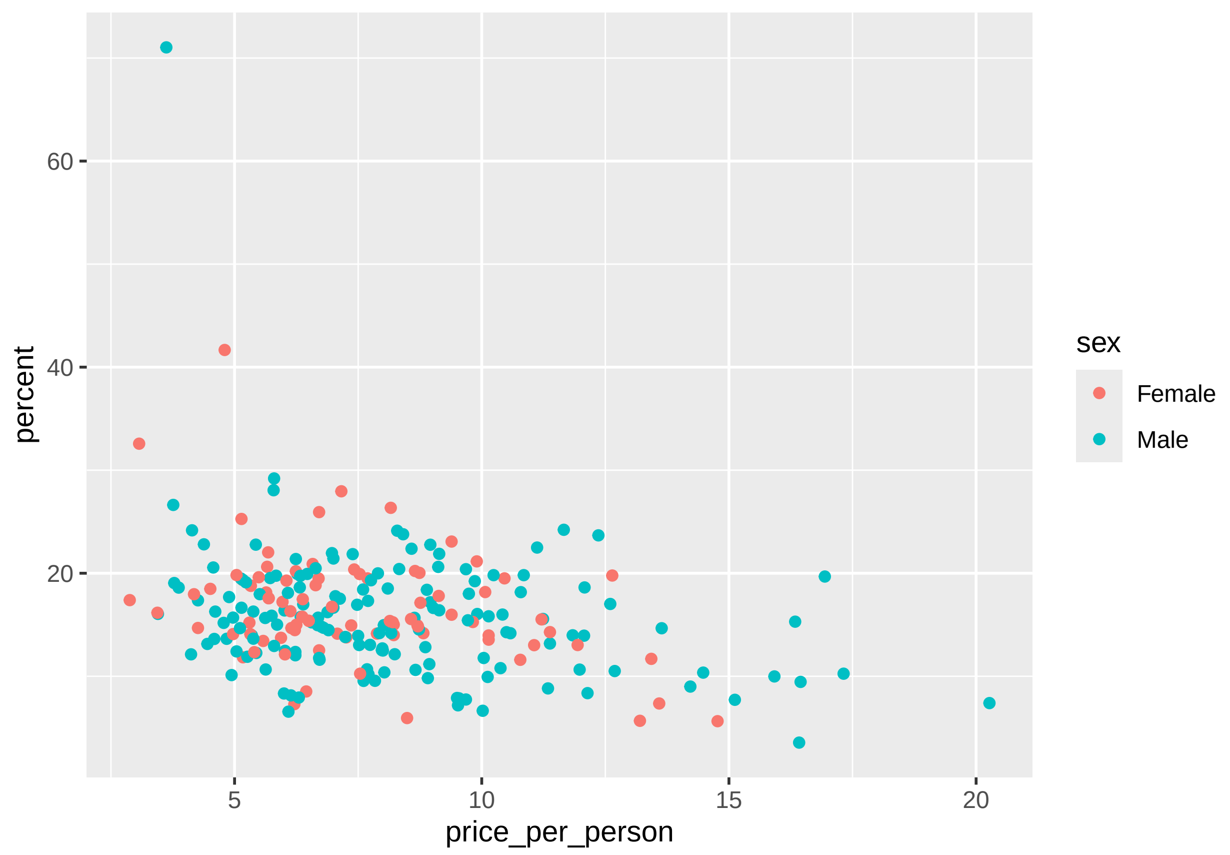


It looks as though these two plots follow a similar shape but the tip amounts spike at values that are whole numbers. This makes sense since tips paid in cash are often a couple of loose dollars. This tells us that the total bill variable will be instrumental in predicting our tip amount.

Scatter Plot

We can look at a graph that compares the `price_per_person` and the `percent` based on `sex`

```r
ggplot(tips, aes(x = price_per_person, y = percent, color = sex))+
  geom_point()
```



This graph may not show a linear relationship between `percent` and `price_per_person` but it does show us a few interesting things about the data. We can see that more of the observations are male than female. We can also see that the tip percent of the total bill majorly falls in the range of 10-20%. We can see that the percentage generally stays the same depending on the the price per person, overall it decreases slightly but that can be explained by rounding up for smaller tip amounts.

Setting Up Models

Now that we've visualized our data, we can begin to set up our models. We generally know how our variables impact tips, so now we can continue on to split the data, create the recipe, and use cross validation on our models.

Train/Test Split

Remember that this dataset is relatively small, so we need to ensure there is enough data for both our testing and training sets. I am going with a 75/25 split so that the testing set will have enough observations to form a proper model while leaving some data solely for testing so that we avoid overfitting. We set a seed at the start to make sure the split is the same every time. We will stratify on the response variable `tips`. This will allow our testing and training sets to generally have a similar range of tip values.

```
set.seed(522)
tips_split <- tips %>%
  initial_split(prop = 0.75, strata = tip)

tips_train <- training(tips_split)
tips_test <- testing(tips_split)
```

We now can look at the dimensions of each of the data sets.

```
dim(tips_train)
```

```
## [1] 181   9
```

```
dim(tips_test)
```

```
## [1] 63  9
```

We see that the training set has 181 observations and the testing set has 63. This verifies the size of the testing and training sets and lets us know how much data we are working with for both.

Recipe Building

Now that we have the testing and training sets we can create a recipe for all of our models to work with. A recipe is a way to set up your data for any form of modeling. The predictors we want to include in our recipe are `total_bill`, `size`, `price_per_person`, `sex`, `smoker`, `day`, and `time`. The last four predictors are all categorical, so we will have to use a dummy predictor for these values. We also will use `step_center` and `step_scale` to standardize our predictors.

```
tips_recipe <- recipe(tip ~ total_bill + size + price_per_person + sex + smoker + day + time, data = ti
  step_dummy(all_nominal_predictors()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())

tips_recipe %>%
  prep()%>%
  bake(new_data = tips_train)
```

```
## # A tibble: 181 x 10
##    total_bill   size price_per_person   tip sex_Male smoker_Yes day_Sat day_Sun
##         <dbl>  <dbl>            <dbl> <dbl>    <dbl>      <dbl>   <dbl>   <dbl>
##  1     -1.06   0.452          -1.52    1.66    0.743     -0.783  -0.743   1.48
##  2     -0.533 -0.599          -0.126   1.96    0.743     -0.783  -0.743   1.48
##  3     -1.07  -0.599          -0.943   1.71    0.743     -0.783  -0.743   1.48
##  4     -0.490 -0.599          -0.0611  1.57    0.743     -0.783  -0.743   1.48
##  5     -0.721 -0.599          -0.415   2       0.743     -0.783   1.34   -0.671
##  6     -1.15  -0.599          -1.07    1.45    0.743     -0.783   1.34   -0.671
##  7     -0.390  0.452          -0.840   2       0.743     -0.783   1.34   -0.671
##  8     -1.14  -0.599          -1.05    1.32    0.743     -0.783  -0.743   1.48
##  9     -1.11  -0.599          -1.00    1.56    0.743     -0.783  -0.743   1.48
## 10      0.744 -0.599           1.82    1.5    -1.34      -0.783   1.34   -0.671
## # i 171 more rows
## # i 2 more variables: day_Thur <dbl>, time_Lunch <dbl>
```

We can prep and bake the recipe to make sure the recipe is running smoothly throughout our project. It's important to prep and bake early so we can catch any problems with the recipe as soon as possible so that we can handle any issues when modeling.

K-Fold Cross Validation

We now will employ a resampling technique called K-fold cross validation. How it works is each observation is assigned randomly to a fold out of k folds, then the model will hold one fold out and assess using the rest of the k-1 folds to fit the model. This process is repeated k times in order to ensure that every observation in the training set is used to fit the model and gives an assessment of test error. This process will allow our models to be tested thoroughly and helps us avoid overfitting. For this data set, we will be using 5 folds and will be stratifying on our response variable, `tip`.

```r
tips_folds <- vfold_cv(tips_train, v = 5, strata = tip)
```

Model Building

In order to find the best way to predict our data, we have to fit it to a number of models. The models we will be fitting include: k-nearest neighbors, elastic net, linear regression, and

For each of these models we have to:

1. Set up the model.

```r
# Model 1: K Nearest Neighbors
tips_knn <- nearest_neighbor(neighbors = tune())%>%
  set_engine("kknn")%>%
  set_mode("regression")

# Model 2: Elastic Net
tips_en <- linear_reg(mixture = tune(),
                       penalty = tune()) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

# Model 3: Linear Regression
tips_lr <- linear_reg() %>%
  set_engine("glm")

# Model 4: Random Forest
tips_rf <- rand_forest(mtry = tune(),
                       trees = tune(),
                       min_n = tune()) %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("regression")
```

2. Set up a workflow for each model, add the specific model, and add the recipe.

```r
# knn workflow
knn_wrk <- workflow() %>%
  add_recipe(tips_recipe) %>%
  add_model(tips_knn)

# en workflow
en_wrk <- workflow() %>%
  add_model(tips_en) %>%
  add_recipe(tips_recipe)

# lr workflow
lr_wrk <- workflow() %>%
  add_model(tips_lr) %>%
  add_recipe(tips_recipe)

# rf workflow
rf_wrk <- workflow() %>%
  add_model(tips_rf) %>%
  add_recipe(tips_recipe)
```

3. Set up a tuning grid for the tuneable models (all but linear regression), specify the tuning variables and the levels tuning.

```r
# knn grid, tuning neighbors
knn_grid <- grid_regular(neighbors(c(1,30)), levels = 10)

# en grid, tuning penalty and mixture
en_grid <- grid_regular(penalty(range = c(-5,5)),
                        mixture(range = c(0,1)),
                        levels = 10)

# rf grid, tuning mtry, trees, and min_n
rf_grid <- grid_regular(mtry(range = c(1, 7)),
                        trees(range = c(200, 600)),
                        min_n(range = c(5, 20)),
                        levels = 5)
```

4. Tune the models using`tune_grid()` for the tune-able models and `fit_resamples()` otherwise. Include the k-fold cross validation set up earlier to resample the models.

```r
#install.packages("kknn")
library(kknn)
#install.packages("glmnet")
library(glmnet)
# tuning the knn grid using k-fold cv
tune_knn <- tune_grid(
  object = knn_wrk,
  resamples = tips_folds,
  grid = knn_grid
 )

# tuning the en grid using k-fold cv
tune_en <- tune_grid(
  object = en_wrk,
  resamples = tips_folds,
  grid = en_grid
)

# fitting lr using k-fold cv
tune_lr <- fit_resamples(
  lr_wrk,
  resamples = tips_folds
)

# tune the rf grid using k-fold cv
tune_rf <- tune_grid(
  object = rf_wrk,
  resamples = tips_folds,
  grid = rf_grid
)
```

5. Save all tuned models to an RDS file to avoid rerunning the model.

```r
# knn
save(tune_knn, file = "tune_knn.rda")

# en
save(tune_en, file = "tune_en.rda")

# no need to save linear regression

# rf
save(tune_rf, file = "tune_rf.rda")
```

6. Load back the saved files.

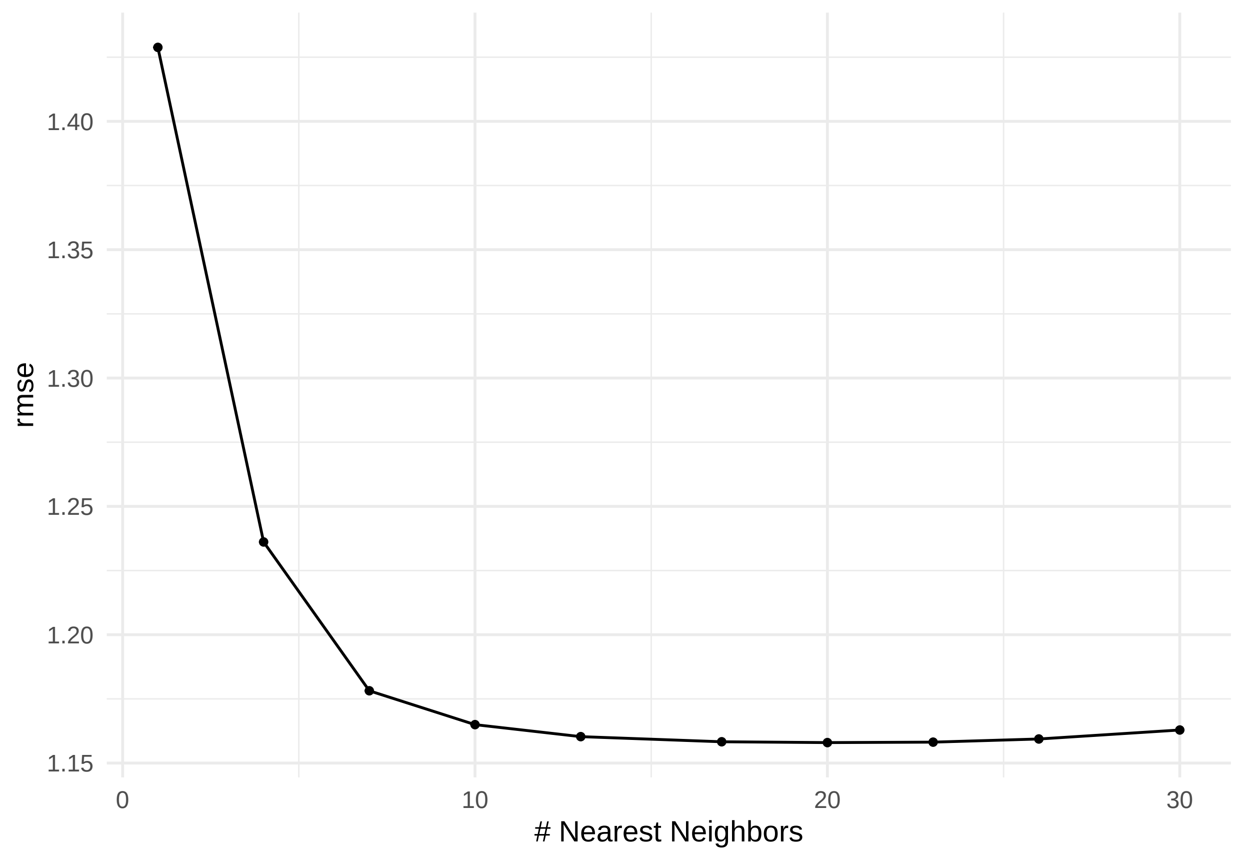
```r
# knn
load(file = "tune_knn.rda")

# en
load(file = "tune_en.rda")

# no need for linear regression

# rf
load(file = "tune_rf.rda")
```

Model Autoplots

Now that we've set up our workflows and tuned each of our models, we can move on to looking at the results of these models. We can look at the auto plots for the tuned models to get an idea for how the results come out before looking at any numbers.

K-Nearest Neighbors

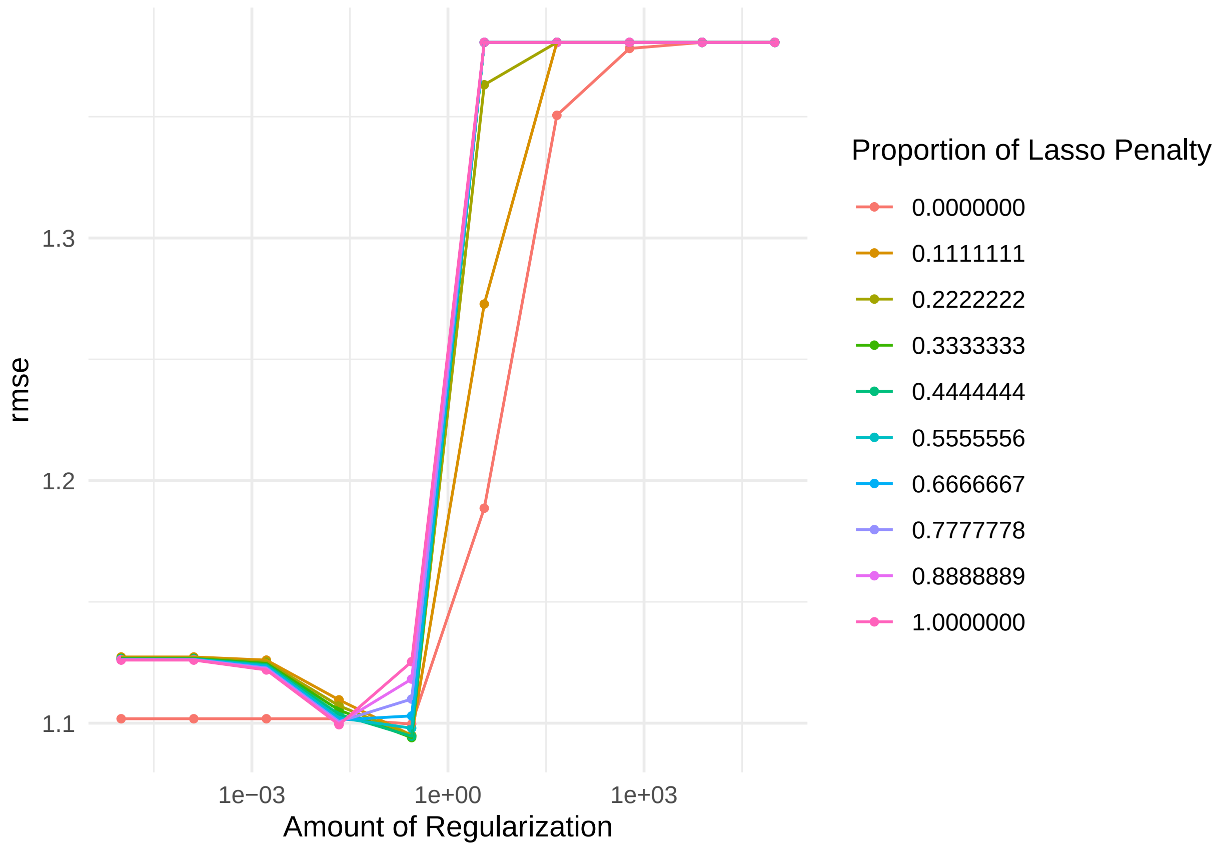
```r
autoplot(tune_knn, metric = "rmse")+ theme_minimal()
```



Notice that the plot continues to decrease past 10 neighbors and starts to go back up around n = 25. Still, the lowest rmse is the highest among the other "best" picks for each of our models. This model will likely not be our final pick.

Elastic Net

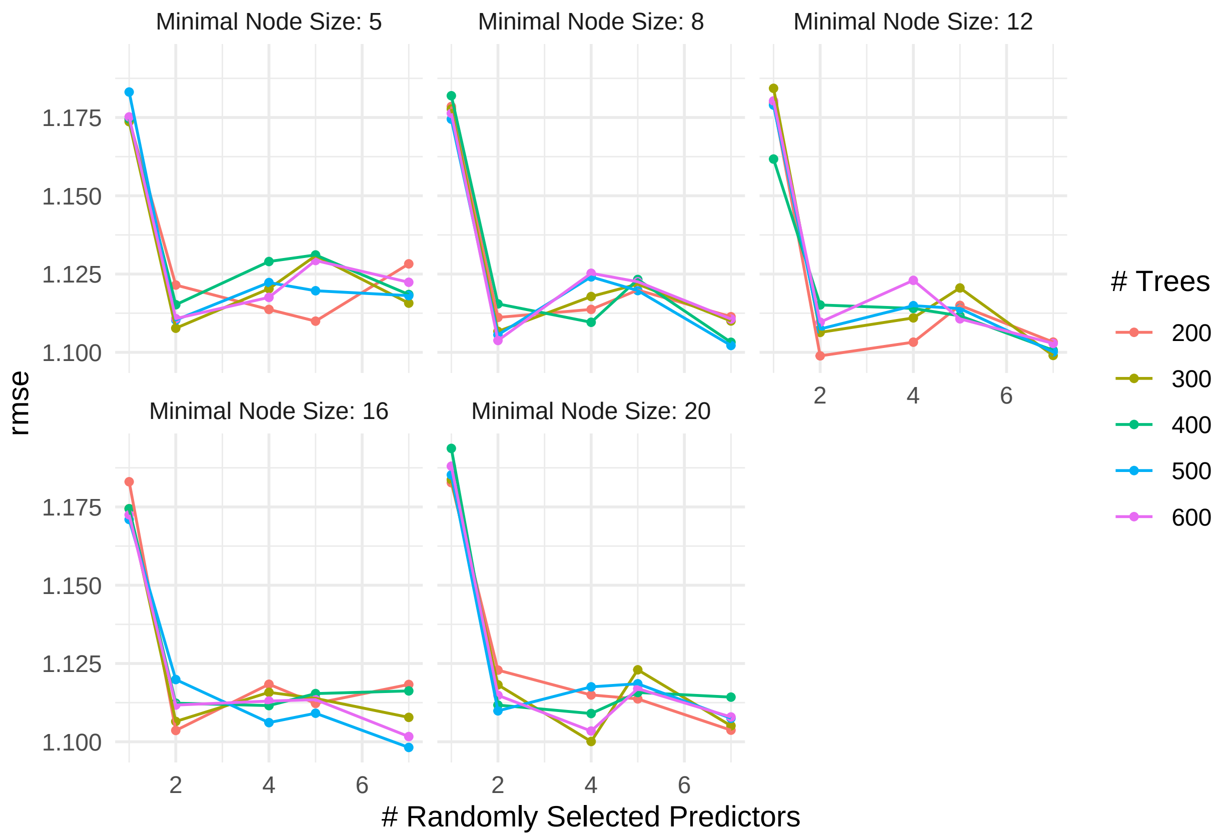
```r
autoplot(tune_en, metric = "rmse")+ theme_minimal()
```



For the Elastic Net model we tuned penalty, the proportion of lasso penalty, and mixture, the amount of regularization. This plot shows us that a 0.333 penalty with a mixture between 0.01 and 1 performed the best. We can see that as the amount of regularization increases, past this lowest point the rmse dramatically increases. This model has a very low rmse, but we cannot be sure until we compare the values of each model's best rmse.

Random Forest

```
autoplot(tune_rf, metric = "rmse")+ theme_minimal()
```



For the random forest model, we tuned three different parameters: `mtry` - the number of randomly selected predictors given to the tree to make its decisions, `min_n` - the minimum number of data values needed to make each split (nodes if you will), and `trees` - the number of trees to develop in the forest. This graph shows that regardless of minimal node size, the RMSE generally decreases as the number of randomly selected predictors increases. The number of trees follow different patterns, in certain plots a group will be parallel for a while and then break off and in others each amount of trees follow entirely different paths. Just from reading these graphs (not using `collect_metrics()` to look at the actual values), the plot with the lowest rmse appears to be the random forest with 2 randomly selected predictors, a minimal node size of 20, and 200 trees. This is one of the more effective models, though it's difficult to tell whether the Elastic Net or the Random Forest will be the best fit model.

Model Results

We will now collect the models metrics looking at `rmse` by using `show_best()` to get the best of the tuned models. We will go one by one, picking the best model from each of the tuned.

```
knn_best <- show_best(tune_knn, metric = "rmse")%>%
  slice(3)

en_best <- show_best(tune_en, metric = "rmse")%>%
  slice(1)

# we can just use collect metrics for linear regression since we aren't tuning
lr_best <- collect_metrics(tune_lr)%>%
  slice(1)
```

```r
rf_best <- show_best(tune_rf, metric = "rmse")%>%
  slice(1)
```

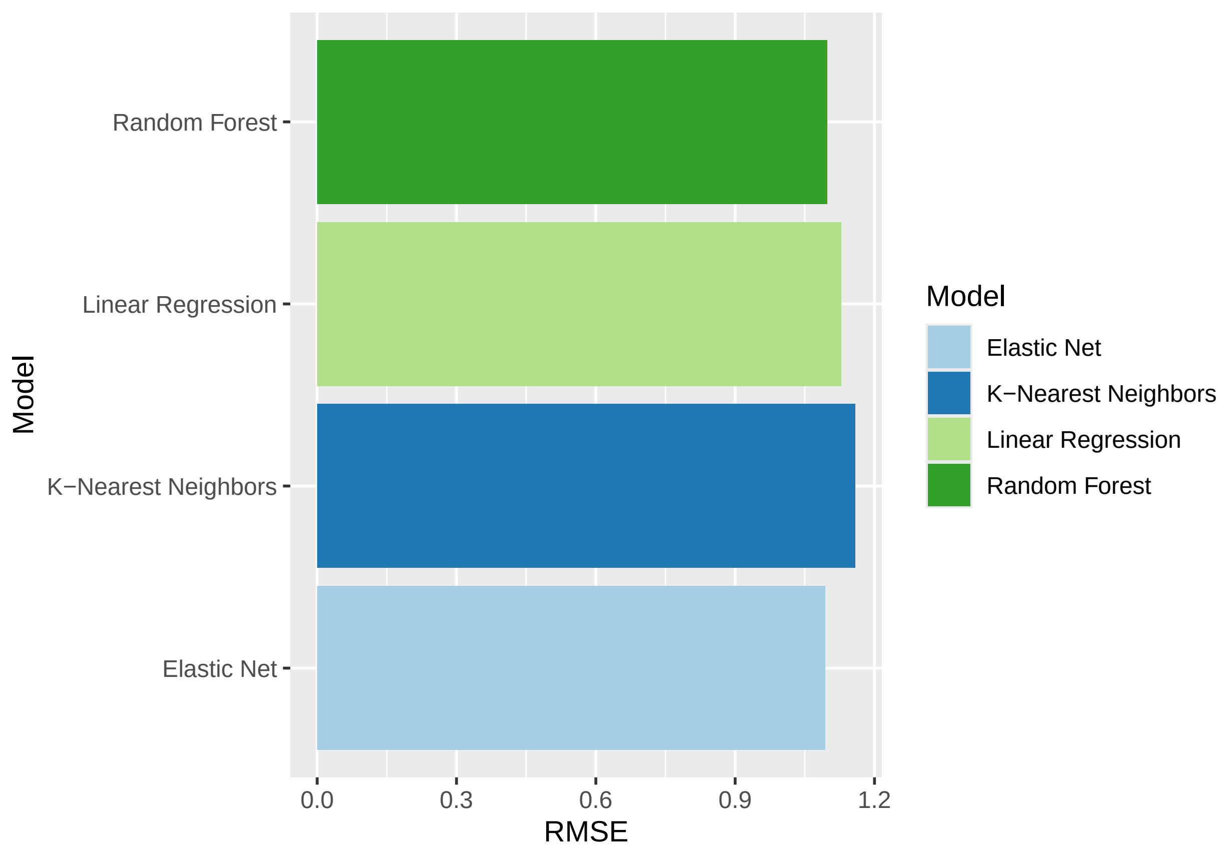
With the results we just collected, we can compare the models and see which performed best.

```r
# creating a tibble of the models and their rmse's to look at side by side
best_compare_tib <- tibble(
  Model = c("K-Nearest Neighbors", "Elastic Net", "Linear Regression",  "Random Forest"),
  RMSE = c(knn_best$mean, en_best$mean, lr_best$mean, rf_best$mean))

# arrange the tibble by lowest rmse
best_compare_tib <- best_compare_tib %>%
  arrange(RMSE)

# print the tibble
best_compare_tib
```

```
## # A tibble: 4 x 2
##   Model                RMSE
##   <chr>               <dbl>
## 1 Elastic Net          1.09
## 2 Random Forest        1.10
## 3 Linear Regression    1.13
## 4 K-Nearest Neighbors  1.16
```

We can also visualize these results graphically with a bar plot.

```r
models_df <- data.frame(best_compare_tib)

ggplot(models_df, aes(y = Model,x=RMSE))+
  geom_histogram(stat = "identity", aes(fill = Model))+
  scale_fill_brewer(palette = "Paired")
```



From these two visualizations, we can see that the elastic net model is our best performing model! We will use this as our final model to fit our training data. These comparisons also show us that the linear and simpler models, K-Nearest Neighbors and Linear Regression, did the worst which means that the data is probably non-linear.

Final Results

Let's take a look at the parameters for the best model that we chose.

```
en_best
```

```
## # A tibble: 1 x 8
##   penalty mixture .metric .estimator  mean     n std_err .config
##     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1   0.278   0.333 rmse    standard    1.09     5  0.0811 Preprocessor1_Model035
```

The final model we are choosing is an elastic net model with 0.2783 for the penalty and the mixture is 0.333. The mean rmse is 1.094 and the standard error is 0.08113.

Fitting to Training Data

We will now take this model and fit it to the training data. This way we can fit our best model on the entire training data set rather than portions from our folded data.

```
best_workflow <- finalize_workflow(en_wrk, en_best)
best_tips_model <- fit(best_workflow, data = tips_train)
```

Fitting to Testing Data

Now that we've finalized our model and trained it with our full training data set, we can fit our model to our testing data set, the portion of the data we have not used in our model building at all.

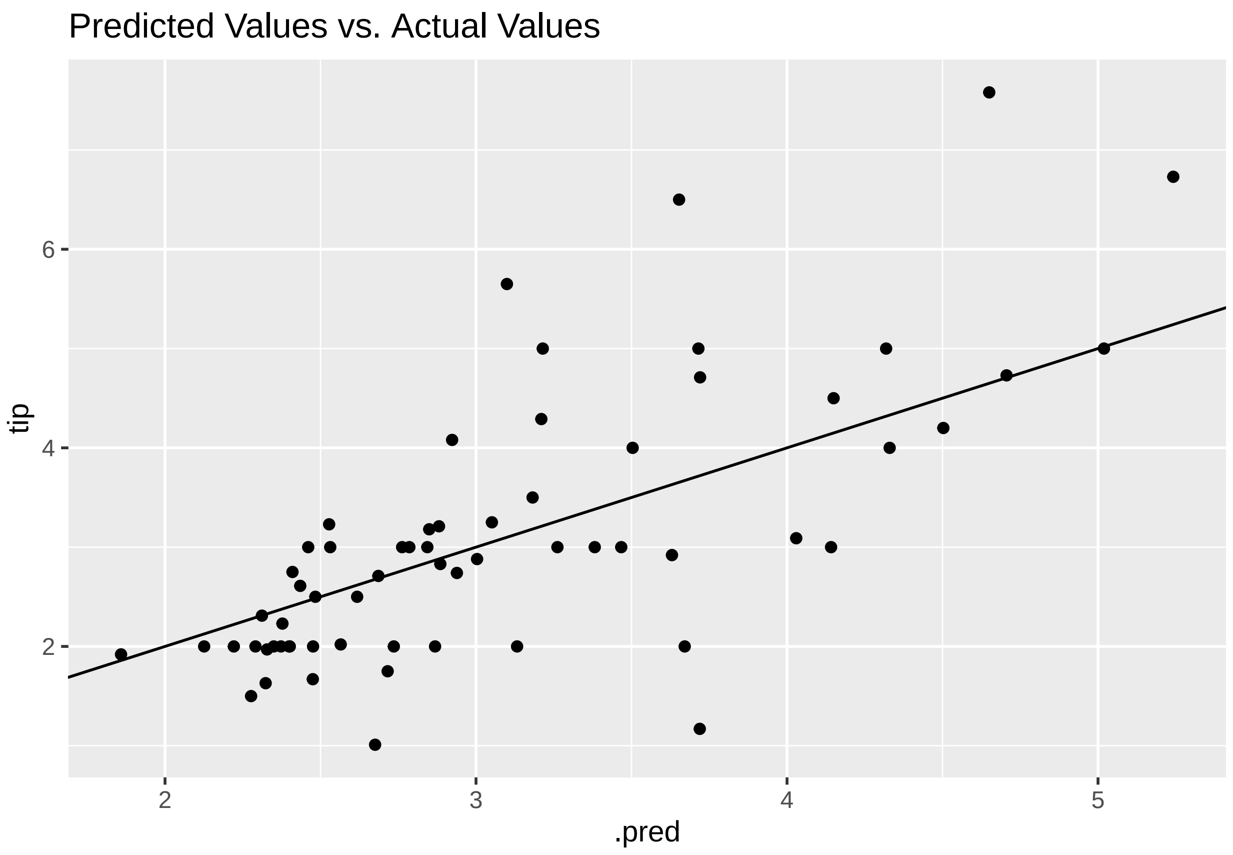
```r
# Augment and look at final rmse
augment(best_tips_model, new_data = tips_train) %>%
  rmse(truth = tip, .pred)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        1.05
```

We can see that the rmse is 1.049, which is quite a bit lower than our training rmse! This still is not great considering our values range from 0-10.

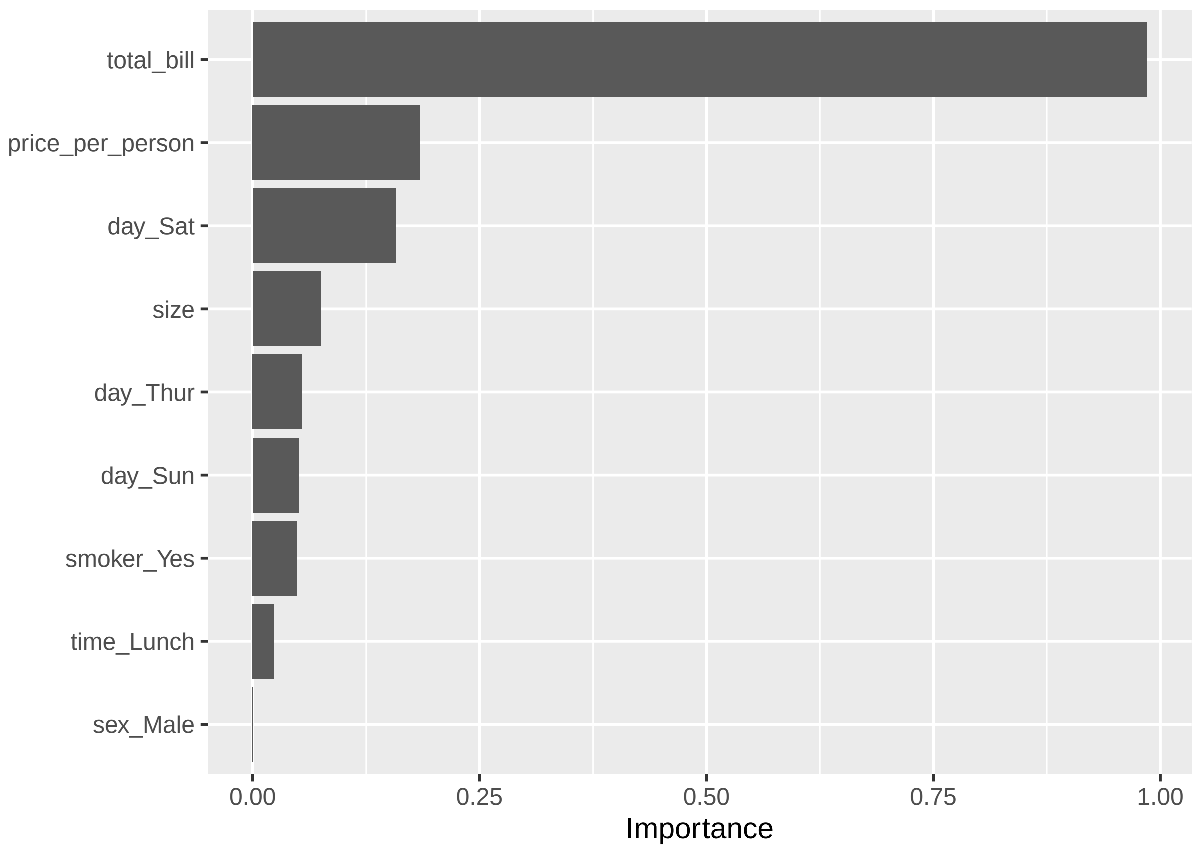
We also can plot the predicted values versus the actual values:

```r
# Make a predicted vs. actual value tibble
tips_tibble <- predict(best_tips_model, new_data = tips_test %>%
                         select(-tip))
tips_tibble <- bind_cols(tips_tibble, tips_test %>%
                           select(tip))

# create a plot for predicted vs actual values
tips_tibble %>%
  ggplot(aes(x = .pred, y = tip))+
  geom_point()+
  geom_abline()+
  labs(title = "Predicted Values vs. Actual Values")
```



Though there are only a few points on the through line, many others cluster around it. We can see that this model is not perfect but it does a better job than the other models we previously fit.

```
vip(best_tips_model)
```



From this graph, we can see that `total_bill` was the most important predictor by a lot.

Conclusion

The final results of our modeling yielded the elastic net model with a rmse of 1.049 as our final model. This model did not do a perfect job fitting our data but it performed better than the random forest, linear regression, and k-nearest neighbors models that we had fit. I did not predict the elastic net model to be the best fit for our data, I expected it to be the random forest model since it is nonparametric and makes no assumptions about the outcome - it's typically a good fit for most data.

As far as improvements go, there are many other models we could have used on the data including lasso regression, ridge regression, polynomial regression, and boosted trees. All of these fits are possible, however I think that the best way we could make a model to predict tip amount would be to have more data from more than one waiter over a larger period of time. This data set was somewhat limiting, many of the predictors only had a few levels or a small range of values. It also would be useful to add predictors of the gender of the waiter, a factor variable of the quality of the restaurant, and the time the customer spent at the restaurant.

At the end of this project, attempting to predict the tip amount using this data set has taught me a lot about machine learning and data analysis. This project allowed me to explore visualizing data, fitting models, looking at their outputs, choosing the best model, and assessing how it fits to a testing set.I've also become more accustomed to looking at a data set and understanding what predictors are valuable and thinking of new predictors that could help fit the best model. Overall, this project has taught me a lot and even though the elastic net model was not a perfect fit, building the model itself was a valuable experience.