Homework 4: Machine Learning

Jacqui Cole

2023-06-01

Table of Contents

# The Bechdel Test

<https://fivethirtyeight.com/features/the-dollar-and-cents-case-against-hollywoods-exclusion-of-women/>

The [Bechdel test](https://bechdeltest.com) is a way to assess how women are depicted in Hollywood movies. In order for a movie to pass the test:

1. It has to have at least two [named] women in it
2. Who talk to each other
3. About something besides a man

There is a nice article and analysis you can find here <https://fivethirtyeight.com/features/the-dollar-and-cents-case-against-hollywoods-exclusion-of-women/> We have a sample of 1394 movies and we want to fit a model to predict whether a film passes the test or not.

bechdel <- read\_csv(here::here("data", "bechdel.csv")) %>%   
 mutate(test = factor(test))

## Rows: 1394 Columns: 10  
## ── Column specification ────────────────────────────────────────────────────────  
## Delimiter: ","  
## chr (4): title, test, rated, genre  
## dbl (6): year, budget\_2013, domgross\_2013, intgross\_2013, metascore, imdb\_ra...  
##   
## ℹ Use `spec()` to retrieve the full column specification for this data.  
## ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

glimpse(bechdel)

## Rows: 1,394  
## Columns: 10  
## $ year <dbl> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 20…  
## $ title <chr> "12 Years a Slave", "2 Guns", "42", "47 Ronin", "A Good …  
## $ test <fct> Fail, Fail, Fail, Fail, Fail, Pass, Pass, Fail, Pass, Pa…  
## $ budget\_2013 <dbl> 2.00, 6.10, 4.00, 22.50, 9.20, 1.20, 1.30, 13.00, 4.00, …  
## $ domgross\_2013 <dbl> 5.3107035, 7.5612460, 9.5020213, 3.8362475, 6.7349198, 1…  
## $ intgross\_2013 <dbl> 15.8607035, 13.2493015, 9.5020213, 14.5803842, 30.424919…  
## $ rated <chr> "R", "R", "PG-13", "PG-13", "R", "R", "PG-13", "PG-13", …  
## $ metascore <dbl> 97, 55, 62, 29, 28, 55, 48, 33, 90, 58, 52, 78, 83, 53, …  
## $ imdb\_rating <dbl> 8.3, 6.8, 7.6, 6.6, 5.4, 7.8, 5.7, 5.0, 7.5, 7.4, 6.2, 7…  
## $ genre <chr> "Biography", "Action", "Biography", "Action", "Action", …

How many films fail/pass the test, both as a number and as a %?

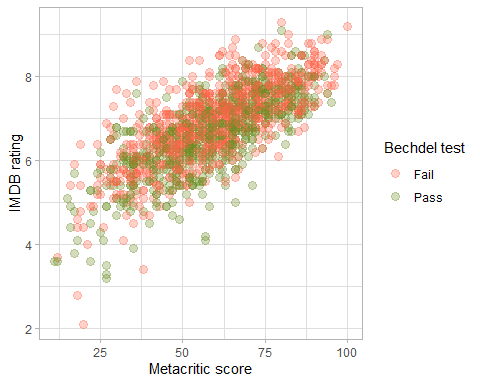
bechdel %>% #read in dataset of movies  
 count(test) %>% #calculate proportion of movies that fail/pass the test  
 mutate(percentfail = 100\*n/sum(n)) #calculate corresponding percentage

## # A tibble: 2 × 3  
## test n percentfail  
## <fct> <int> <dbl>  
## 1 Fail 772 55.4  
## 2 Pass 622 44.6

#The answers are found in the table below:

## Movie scores

ggplot(data = bechdel, aes(  
 x = metascore,  
 y = imdb\_rating,  
 colour = test  
)) +  
 geom\_point(alpha = .3, size = 3) +  
 scale\_colour\_manual(values = c("tomato", "olivedrab")) +  
 labs(  
 x = "Metacritic score",  
 y = "IMDB rating",  
 colour = "Bechdel test"  
 ) +  
 theme\_light()



# Split the data

# \*\*Split the data\*\*  
  
set.seed(123)  
  
data\_split <- initial\_split(bechdel, # updated data  
 prop = 0.8,   
 strata = test)  
  
bechdel\_train <- training(data\_split)   
bechdel\_test <- testing(data\_split)

Check the counts and % (proportions) of the test variable in each set.

bechdel\_train %>% #read in training dataset of movies  
 count(test) %>% #calculate proportion of movies that fail/pass the test  
 mutate(percentfail = 100\*n/sum(n)) #calculate corresponding percentage

## # A tibble: 2 × 3  
## test n percentfail  
## <fct> <int> <dbl>  
## 1 Fail 617 55.4  
## 2 Pass 497 44.6

#The training data set has the following counts and proportion of Fail/Pass:

bechdel\_test %>% #read in test dataset of movies  
 count(test) %>% #calculate proportion of movies that fail/pass the test  
 mutate(percentfail = 100\*n/sum(n)) #calculate corresponding percentage

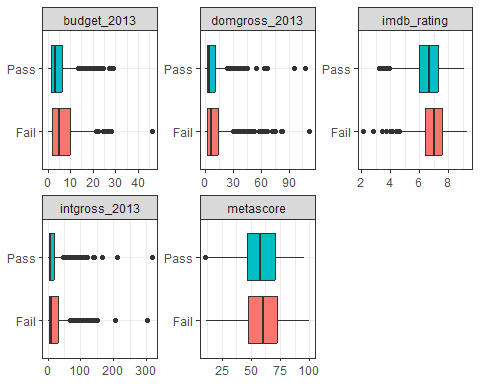
## # A tibble: 2 × 3  
## test n percentfail  
## <fct> <int> <dbl>  
## 1 Fail 155 55.4  
## 2 Pass 125 44.6

#The training data set has the following counts and proportion of Fail/Pass:

## Feature exploration

## Any outliers?

bechdel %>%   
 select(test, budget\_2013, domgross\_2013, intgross\_2013, imdb\_rating, metascore) %>%   
  
 pivot\_longer(cols = 2:6,  
 names\_to = "feature",  
 values\_to = "value") %>%   
 ggplot()+  
 aes(x=test, y = value, fill = test)+  
 coord\_flip()+  
 geom\_boxplot()+  
 facet\_wrap(~feature, scales = "free")+  
 theme\_bw()+  
 theme(legend.position = "none")+  
 labs(x=NULL,y = NULL)



#There are no massive outliers, although the maximum value of budget 2013 seems to be disproportionately larger than others. So I find the maximum value of this outlier:  
 bechdel %>%  
 summarise(maxi = max(budget\_2013))

## # A tibble: 1 × 1  
## maxi  
## <dbl>  
## 1 46.1

#The identity of this outlier is deduced by:  
  
 bechdel %>%  
 filter (budget\_2013 > 46)

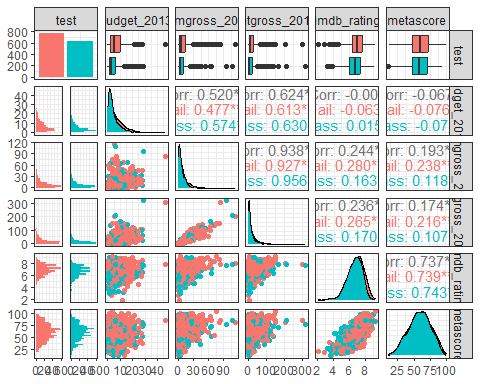
## # A tibble: 1 × 10  
## year title test budget\_2013 domgross\_2013 intgross\_2013 rated metascore  
## <dbl> <chr> <fct> <dbl> <dbl> <dbl> <chr> <dbl>  
## 1 2009 Avatar Fail 46.1 82.6 302. PG-13 83  
## # ℹ 2 more variables: imdb\_rating <dbl>, genre <chr>

#The outlier is Avatar. I note that its domestic gross (domgross\_2013 is also the maximum value of that field (302), but it is not the max value of the intgross (although it is high at 82.5)). Overall, when considered over the full range of the variables plotted in the above boxplots, it is not an outlier in all regards. So, I keep it in the dataset in going forward. Though I just note its disproportionately large value for a few of its variables.

## Scatterplot - Correlation Matrix

Write a paragraph discussing the output of the following

bechdel %>%   
 select(test, budget\_2013, domgross\_2013, intgross\_2013, imdb\_rating, metascore)%>%   
 ggpairs(aes(colour=test), alpha=0.2)+  
 theme\_bw()



#The output of the following correlation matrix tests reveal that:  
#(a) the variables have quite different distributions   
#- metascore is somewhere between a trianglular and normal distribution;  
#- imbb\_rating appears to be a fairly normal distribution albeit with a small left skew;  
#-the budget\_2013, domgross\_2013 and intgross\_2013 variables show a near Voigt function that is cut off on the left and which is highly right skewed, as most data are of low values.  
#(b) The test data (Fail and Pass) seem to be fairly even in number and the distribution of Fail and Pass data, as judged by the similar profiles showing in the plots that comprise the left hand column below the test histogram. This evenness is important because it shows that the test data are balanced when it comes to a machine-learning application. A corrolaary to this is t also shows that a random partitioning of the data into a training and testing dataset should result in a balanced set of split datasets.  
#(c) The correlation values for the total datasets (in black font) show that there are significant levels of correlation between the following variables:  
#  
# 52% correlation between domgross\_2013 and budget\_2013  
# 62% correlation between intgross\_2013 and budget\_2013  
# 94% correlation between intgross\_2013 and domgross\_2013  
# 74% correlation between metascore and imdb\_rating  
#  
# This makes intuitive sense because one would hope that the larger the budget for a film, the more that the film will profit in both domestic and international gross. As a corrolary, one would expect a correlation between domestic and international gross, assuming that most films are international (as hollywood films tend to be).  
#  
# Meanwhile, one would intuitively expect there to be a correlation in ratings: the imdb\_rating and the metascore in this case.  
#  
# Note that there are no correlations between the scoring-based variables and the budget variable, which means that the audience appreciation level is independent of the budget spent on the film!  
#  
# These ratings are slightly correlated to the gross profits of the films which suggests that customers choose to see big budget films a bit more than others, perhaps expecting a good experience. However, the lack of correlation between ratings and budget suggest that the audience is occasionally let down by a big budget film.  
#  
# There is little variation between the correlations of the total data versus those that Pass or Fail these 3 criteria. However, there are a few observable differences: e.g. domgross\_2013 and budget\_2013 are slightly less correlated if the criteria are not met (Fail) and slightly more correlated if the criteria are met.   
# There is also a slightly lower correlation between film ratings/scores and gross profits of films if the criteria are met (Pass) than if they are not met (Fail).  
#   
#This all means that there appears to be a modest but statistically significant relationship in movies and these criteria being met.

## Categorical variables

Write a paragraph discussing the output of the following

bechdel %>%   
 group\_by(genre, test) %>%  
 summarise(n = n()) %>%   
 mutate(prop = n/sum(n))

## `summarise()` has grouped output by 'genre'. You can override using the  
## `.groups` argument.

## # A tibble: 24 × 4  
## # Groups: genre [14]  
## genre test n prop  
## <chr> <fct> <int> <dbl>  
## 1 Action Fail 260 0.707  
## 2 Action Pass 108 0.293  
## 3 Adventure Fail 52 0.559  
## 4 Adventure Pass 41 0.441  
## 5 Animation Fail 63 0.677  
## 6 Animation Pass 30 0.323  
## 7 Biography Fail 36 0.554  
## 8 Biography Pass 29 0.446  
## 9 Comedy Fail 138 0.427  
## 10 Comedy Pass 185 0.573  
## # ℹ 14 more rows

bechdel %>%   
 group\_by(rated, test) %>%  
 summarise(n = n()) %>%   
 mutate(prop = n/sum(n))

## `summarise()` has grouped output by 'rated'. You can override using the  
## `.groups` argument.

## # A tibble: 10 × 4  
## # Groups: rated [5]  
## rated test n prop  
## <chr> <fct> <int> <dbl>  
## 1 G Fail 16 0.615  
## 2 G Pass 10 0.385  
## 3 NC-17 Fail 5 0.833  
## 4 NC-17 Pass 1 0.167  
## 5 PG Fail 115 0.561  
## 6 PG Pass 90 0.439  
## 7 PG-13 Fail 283 0.529  
## 8 PG-13 Pass 252 0.471  
## 9 R Fail 353 0.568  
## 10 R Pass 269 0.432

#  
# <Write a paragraph>  
#  
#

# Train first models. test ~ metascore + imdb\_rating

lr\_mod <- logistic\_reg() %>%   
 set\_engine(engine = "glm") %>%   
 set\_mode("classification")  
  
lr\_mod

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

tree\_mod <- decision\_tree() %>%   
 set\_engine(engine = "C5.0") %>%   
 set\_mode("classification")  
  
tree\_mod

## Decision Tree Model Specification (classification)  
##   
## Computational engine: C5.0

lr\_fit <- lr\_mod %>% # parsnip model  
 fit(test ~ metascore + imdb\_rating, # a formula  
 data = bechdel\_train # dataframe  
 )  
  
tree\_fit <- tree\_mod %>% # parsnip model  
 fit(test ~ metascore + imdb\_rating, # a formula  
 data = bechdel\_train # dataframe  
 )

## Logistic regression

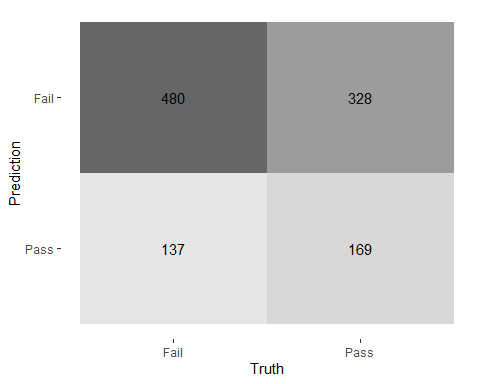
lr\_fit %>%  
 broom::tidy()

## # A tibble: 3 × 5  
## term estimate std.error statistic p.value  
## <chr> <dbl> <dbl> <dbl> <dbl>  
## 1 (Intercept) 2.80 0.494 5.68 1.35e- 8  
## 2 metascore 0.0207 0.00536 3.86 1.13e- 4  
## 3 imdb\_rating -0.625 0.100 -6.24 4.36e-10

lr\_preds <- lr\_fit %>%  
 augment(new\_data = bechdel\_train) %>%  
 mutate(.pred\_match = if\_else(test == .pred\_class, 1, 0))

### Confusion matrix

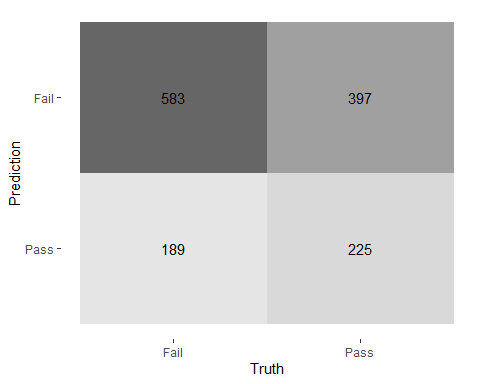
lr\_preds %>%   
 conf\_mat(truth = test, estimate = .pred\_class) %>%   
 autoplot(type = "heatmap")



## Decision Tree

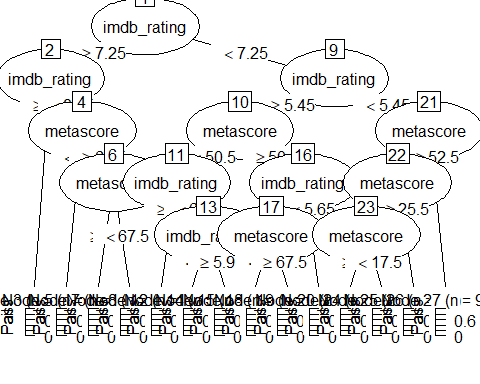
tree\_preds <- tree\_fit %>%  
 augment(new\_data = bechdel) %>%  
 mutate(.pred\_match = if\_else(test == .pred\_class, 1, 0))

tree\_preds %>%   
 conf\_mat(truth = test, estimate = .pred\_class) %>%   
 autoplot(type = "heatmap")



## Draw the decision tree

draw\_tree <-   
 rpart::rpart(  
 test ~ metascore + imdb\_rating,  
 data = bechdel\_train, # uses data that contains both birth weight and `low`  
 control = rpart::rpart.control(maxdepth = 5, cp = 0, minsplit = 10)  
 ) %>%   
 partykit::as.party()  
plot(draw\_tree)



# Cross Validation

Run the code below. What does it return?

set.seed(123)  
bechdel\_folds <- vfold\_cv(data = bechdel\_train,   
 v = 2,   
 strata = test)  
bechdel\_folds

## # 2-fold cross-validation using stratification   
## # A tibble: 2 × 2  
## splits id   
## <list> <chr>  
## 1 <split [556/558]> Fold1  
## 2 <split [558/556]> Fold2

# This code returns the labelling of a 10-fold cross-validation.

## fit\_resamples()

Trains and tests a resampled model.

lr\_fit <- lr\_mod %>%  
 fit\_resamples(  
 test ~ metascore + imdb\_rating,  
 resamples = bechdel\_folds  
 )  
  
  
tree\_fit <- tree\_mod %>%  
 fit\_resamples(  
 test ~ metascore + imdb\_rating,  
 resamples = bechdel\_folds  
 )

## collect\_metrics()

Unnest the metrics column from a tidymodels fit\_resamples()

collect\_metrics(lr\_fit)

## # A tibble: 2 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.574 2 0.0169 Preprocessor1\_Model1  
## 2 roc\_auc binary 0.602 2 0.0148 Preprocessor1\_Model1

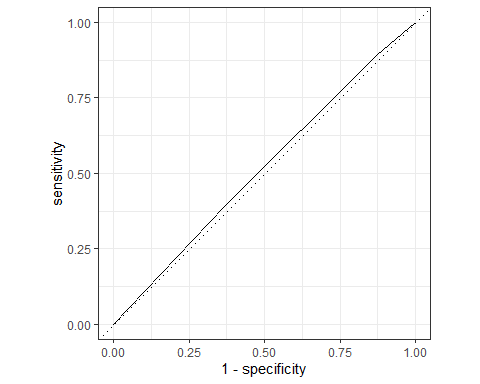
collect\_metrics(tree\_fit)

## # A tibble: 2 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.551 2 0.00260 Preprocessor1\_Model1  
## 2 roc\_auc binary 0.510 2 0.0103 Preprocessor1\_Model1

tree\_preds <- tree\_mod %>%   
 fit\_resamples(  
 test ~ metascore + imdb\_rating,   
 resamples = bechdel\_folds,  
 control = control\_resamples(save\_pred = TRUE) #<<  
 )  
  
# What does the data for ROC look like?  
tree\_preds %>%   
 collect\_predictions() %>%   
 roc\_curve(truth = test, .pred\_Fail)

## # A tibble: 5 × 3  
## .threshold specificity sensitivity  
## <dbl> <dbl> <dbl>  
## 1 -Inf 0 1   
## 2 0.391 0 1   
## 3 0.554 0.131 0.890  
## 4 0.607 0.632 0.389  
## 5 Inf 1 0

# Draw the ROC  
tree\_preds %>%   
 collect\_predictions() %>%   
 roc\_curve(truth = test, .pred\_Fail) %>%   
 autoplot()



# The data for ROC show that the model is not very accurate at all, because the True Positive and True Negatives (matrix diagonal) in the confusion matrix are not very distinguished from the False Negatives and False Positives (off diagonals). Indeed, the results are barely above the dotted line (x = y) which would be the case for data from random guessing.

# Build a better training set with recipes

## Preprocessing options

* Encode categorical predictors
* Center and scale variables
* Handle class imbalance
* Impute missing data
* Perform dimensionality reduction
* … …

## To build a recipe

1. Start the recipe()
2. Define the variables involved
3. Describe **prep**rocessing [step-by-step]

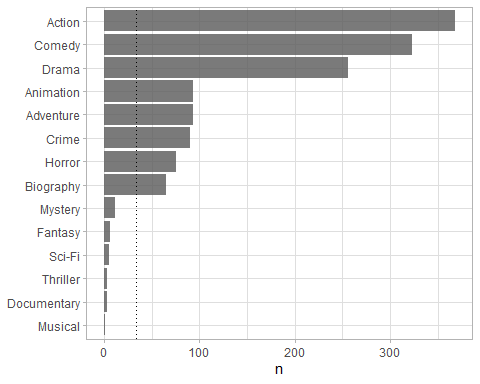
## Collapse Some Categorical Levels

Do we have any genre with few observations? Assign genres that have less than 3% to a new category ‘Other’

bechdel %>% #reading in the full dataset  
 count(genre) %>% # looking at the distribution of genres  
 mutate(m = 100\*n/sum(n)) %>% # calculating the genre proportion  
 filter (m < 3) # identifying the genres that appear less than 3 percent.

## # A tibble: 6 × 3  
## genre n m  
## <chr> <int> <dbl>  
## 1 Documentary 3 0.215   
## 2 Fantasy 6 0.430   
## 3 Musical 1 0.0717  
## 4 Mystery 12 0.861   
## 5 Sci-Fi 5 0.359   
## 6 Thriller 3 0.215

#  
# The resulting table below identifies the six genre categories that make up less than 3 percent of the dataset.



movie\_rec <-  
 recipe(test ~ .,  
 data = bechdel\_train) %>%  
   
 # Genres with less than 5% will be in a catewgory 'Other'  
 step\_other(genre, threshold = .03)

## Before recipe

## # A tibble: 14 × 2  
## genre n  
## <chr> <int>  
## 1 Action 293  
## 2 Comedy 254  
## 3 Drama 213  
## 4 Adventure 75  
## 5 Animation 72  
## 6 Crime 68  
## 7 Horror 68  
## 8 Biography 50  
## 9 Mystery 7  
## 10 Fantasy 5  
## 11 Sci-Fi 3  
## 12 Thriller 3  
## 13 Documentary 2  
## 14 Musical 1

## After recipe

movie\_rec %>%   
 prep() %>%   
 bake(new\_data = bechdel\_train) %>%   
 count(genre, sort = TRUE)

## # A tibble: 9 × 2  
## genre n  
## <fct> <int>  
## 1 Action 293  
## 2 Comedy 254  
## 3 Drama 213  
## 4 Adventure 75  
## 5 Animation 72  
## 6 Crime 68  
## 7 Horror 68  
## 8 Biography 50  
## 9 other 21

## step\_dummy()

Converts nominal data into numeric dummy variables

movie\_rec <- recipe(test ~ ., data = bechdel) %>%  
 step\_other(genre, threshold = .03) %>%   
 step\_dummy(all\_nominal\_predictors())   
  
movie\_rec

##

## ── Recipe ──────────────────────────────────────────────────────────────────────

##

## ── Inputs

## Number of variables by role

## outcome: 1  
## predictor: 9

##

## ── Operations

## • Collapsing factor levels for: genre

## • Dummy variables from: all\_nominal\_predictors()

## Let’s think about the modelling

What if there were no films with rated NC-17 in the training data?

#Then, the model will not consider this NC-17 rating.

* Will the model have a coefficient for rated NC-17?

# The model will not have any coefficient for rated NC-17.

* What will happen if the test data includes a film with rated NC-17?

# The model will presumably ignore the rated NC-17 class of data.

## step\_novel()

Adds a catch-all level to a factor for any new values not encountered in model training, which lets R intelligently predict new levels in the test set.

movie\_rec <- recipe(test ~ ., data = bechdel) %>%  
 step\_other(genre, threshold = .03) %>%   
 step\_novel(all\_nominal\_predictors) %>% # Use \*before\* `step\_dummy()` so new level is dummified  
 step\_dummy(all\_nominal\_predictors())

## step\_zv()

Intelligently handles zero variance variables (variables that contain only a single value)

movie\_rec <- recipe(test ~ ., data = bechdel) %>%  
 step\_other(genre, threshold = .03) %>%   
 step\_novel(all\_nominal(), -all\_outcomes()) %>% # Use \*before\* `step\_dummy()` so new level is dummified  
 step\_dummy(all\_nominal(), -all\_outcomes()) %>%   
 step\_zv(all\_numeric(), -all\_outcomes())

## step\_normalize()

Centers then scales numeric variable (mean = 0, sd = 1)

movie\_rec <- recipe(test ~ ., data = bechdel) %>%  
 step\_other(genre, threshold = .03) %>%   
 step\_novel(all\_nominal(), -all\_outcomes()) %>% # Use \*before\* `step\_dummy()` so new level is dummified  
 step\_dummy(all\_nominal(), -all\_outcomes()) %>%   
 step\_zv(all\_numeric(), -all\_outcomes()) %>%   
 step\_normalize(all\_numeric())

## step\_corr()

Removes highly correlated variables

movie\_rec <- recipe(test ~ ., data = bechdel) %>%  
 step\_other(genre, threshold = .03) %>%   
 step\_novel(all\_nominal(), -all\_outcomes()) %>% # Use \*before\* `step\_dummy()` so new level is dummified  
 step\_dummy(all\_nominal(), -all\_outcomes()) %>%   
 step\_zv(all\_numeric(), -all\_outcomes()) %>%   
 step\_normalize(all\_numeric())   
 # step\_corr(all\_predictors(), threshold = 0.75, method = "spearman")   
  
  
  
movie\_rec

##

## ── Recipe ──────────────────────────────────────────────────────────────────────

##

## ── Inputs

## Number of variables by role

## outcome: 1  
## predictor: 9

##

## ── Operations

## • Collapsing factor levels for: genre

## • Novel factor level assignment for: all\_nominal(), -all\_outcomes()

## • Dummy variables from: all\_nominal(), -all\_outcomes()

## • Zero variance filter on: all\_numeric(), -all\_outcomes()

## • Centering and scaling for: all\_numeric()

# Define different models to fit

## Model Building  
  
# 1. Pick a `model type`  
# 2. set the `engine`  
# 3. Set the `mode`: regression or classification  
  
# Logistic regression  
log\_spec <- logistic\_reg() %>% # model type  
 set\_engine(engine = "glm") %>% # model engine  
 set\_mode("classification") # model mode  
  
# Show your model specification  
log\_spec

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

# Decision Tree  
tree\_spec <- decision\_tree() %>%  
 set\_engine(engine = "C5.0") %>%  
 set\_mode("classification")  
  
tree\_spec

## Decision Tree Model Specification (classification)  
##   
## Computational engine: C5.0

# Random Forest  
library(ranger)  
  
rf\_spec <-   
 rand\_forest() %>%   
 set\_engine("ranger", importance = "impurity") %>%   
 set\_mode("classification")  
  
  
# Boosted tree (XGBoost)  
library(xgboost)

##   
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':  
##   
## slice

xgb\_spec <-   
 boost\_tree() %>%   
 set\_engine("xgboost") %>%   
 set\_mode("classification")   
  
# K-nearest neighbour (k-NN)  
knn\_spec <-   
 nearest\_neighbor(neighbors = 4) %>% # we can adjust the number of neighbors   
 set\_engine("kknn") %>%   
 set\_mode("classification")

# Bundle recipe and model with workflows

log\_wflow <- # new workflow object  
 workflow() %>% # use workflow function  
 add\_recipe(movie\_rec) %>% # use the new recipe  
 add\_model(log\_spec) # add your model spec  
  
# show object  
log\_wflow

## ══ Workflow ════════════════════════════════════════════════════════════════════  
## Preprocessor: Recipe  
## Model: logistic\_reg()  
##   
## ── Preprocessor ────────────────────────────────────────────────────────────────  
## 5 Recipe Steps  
##   
## • step\_other()  
## • step\_novel()  
## • step\_dummy()  
## • step\_zv()  
## • step\_normalize()  
##   
## ── Model ───────────────────────────────────────────────────────────────────────  
## Logistic Regression Model Specification (classification)  
##   
## Computational engine: glm

## A few more workflows  
  
tree\_wflow <-  
 workflow() %>%  
 add\_recipe(movie\_rec) %>%   
 add\_model(tree\_spec)   
  
rf\_wflow <-  
 workflow() %>%  
 add\_recipe(movie\_rec) %>%   
 add\_model(rf\_spec)   
  
xgb\_wflow <-  
 workflow() %>%  
 add\_recipe(movie\_rec) %>%   
 add\_model(xgb\_spec)  
  
knn\_wflow <-  
 workflow() %>%  
 add\_recipe(movie\_rec) %>%   
 add\_model(knn\_spec)

HEADS UP

1. How many models have you specified?

#Five models have been specified in the above code run and these take the form of various types of classification: logistic regression (classification), a decision tree (tree), and various tree options: a random forest (rf), a booststrapping option: gradient boosting (xgb) and k-nearest neighbours (knn).

1. What’s the difference between a model specification and a workflow?

#A model specification is essentially a function that one can use to apply unseen data that can fit well to the model if the data used to generate the function are similar to the unseen data. In this case, each model is a classifier which means that it partitions data into certain categories according to its relative fit to the underpinning model that was defined by the training data.  
  
#In contrast, a workflow is a sequence of steps or operations that lead to a result. In the workflows above, a model is contained within each of them.

1. Do you need to add a formula (e.g., test ~ .) if you have a recipe?

# Model Comparison

You now have all your models. Adapt the code from slides code-from-slides-CA-housing.R, line 400 onwards to assess which model gives you the best classification.

## Evaluate Models  
  
## Logistic regression results{.smaller}  
  
log\_res <- log\_wflow %>%   
 fit\_resamples(  
 resamples = bechdel\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas, accuracy,  
 kap, roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE))

## → A | warning: glm.fit: algorithm did not converge

## There were issues with some computations A: x1 → B | warning: prediction from rank-deficient fit; attr(\*, "non-estim") has doubtful cases  
## There were issues with some computations A: x1There were issues with some computations A: x1 B: x1There were issues with some computations A: x2 B: x1There were issues with some computations A: x2 B: x2There were issues with some computations A: x2 B: x2

# Show average performance over all folds (note that we use log\_res):  
log\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.467 2 0.0315 Preprocessor1\_Model1  
## 2 f\_meas binary 0.386 2 0.0344 Preprocessor1\_Model1  
## 3 kap binary -0.0286 2 0.0819 Preprocessor1\_Model1  
## 4 precision binary 0.553 2 0.0654 Preprocessor1\_Model1  
## 5 recall binary 0.308 2 0.0621 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.484 2 0.0411 Preprocessor1\_Model1  
## 7 sens binary 0.308 2 0.0621 Preprocessor1\_Model1  
## 8 spec binary 0.664 2 0.148 Preprocessor1\_Model1

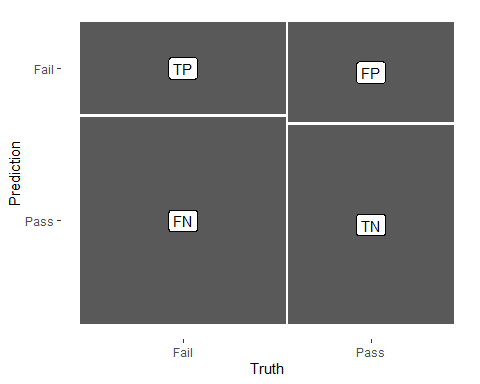
# Show performance for every single fold:  
log\_res %>% collect\_metrics(summarize = FALSE)

## # A tibble: 16 × 5  
## id .metric .estimator .estimate .config   
## <chr> <chr> <chr> <dbl> <chr>   
## 1 Fold1 recall binary 0.246 Preprocessor1\_Model1  
## 2 Fold1 precision binary 0.618 Preprocessor1\_Model1  
## 3 Fold1 f\_meas binary 0.352 Preprocessor1\_Model1  
## 4 Fold1 accuracy binary 0.498 Preprocessor1\_Model1  
## 5 Fold1 kap binary 0.0533 Preprocessor1\_Model1  
## 6 Fold1 sens binary 0.246 Preprocessor1\_Model1  
## 7 Fold1 spec binary 0.811 Preprocessor1\_Model1  
## 8 Fold1 roc\_auc binary 0.525 Preprocessor1\_Model1  
## 9 Fold2 recall binary 0.370 Preprocessor1\_Model1  
## 10 Fold2 precision binary 0.487 Preprocessor1\_Model1  
## 11 Fold2 f\_meas binary 0.421 Preprocessor1\_Model1  
## 12 Fold2 accuracy binary 0.435 Preprocessor1\_Model1  
## 13 Fold2 kap binary -0.111 Preprocessor1\_Model1  
## 14 Fold2 sens binary 0.370 Preprocessor1\_Model1  
## 15 Fold2 spec binary 0.516 Preprocessor1\_Model1  
## 16 Fold2 roc\_auc binary 0.443 Preprocessor1\_Model1

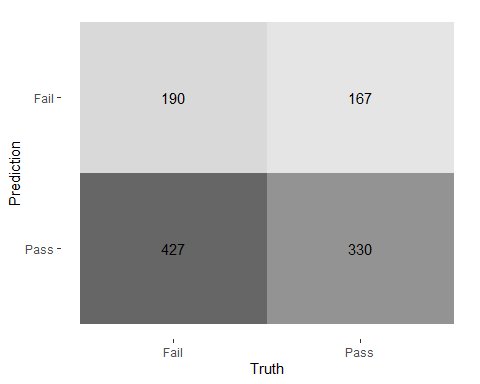
## `collect\_predictions()` and get confusion matrix{.smaller}  
  
log\_pred <- log\_res %>% collect\_predictions()  
  
log\_pred %>% conf\_mat(test, .pred\_class)

## Truth  
## Prediction Fail Pass  
## Fail 190 167  
## Pass 427 330

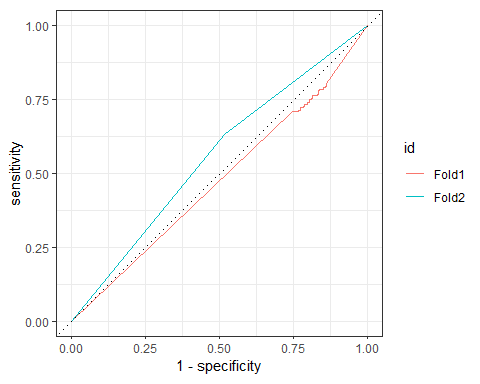
log\_pred %>%   
 conf\_mat(test, .pred\_class) %>%   
 autoplot(type = "mosaic") +  
 geom\_label(aes(  
 x = (xmax + xmin) / 2,   
 y = (ymax + ymin) / 2,   
 label = c("TP", "FN", "FP", "TN")))



log\_pred %>%   
 conf\_mat(test, .pred\_class) %>%   
 autoplot(type = "heatmap")



## ROC Curve  
  
log\_pred %>%   
 group\_by(id) %>% # id contains our folds  
 roc\_curve(test, .pred\_Pass) %>%   
 autoplot()



## Decision Tree results  
  
tree\_res <-  
 tree\_wflow %>%   
 fit\_resamples(  
 resamples = bechdel\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas,   
 accuracy, kap,  
 roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE)  
 )   
  
tree\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.576 2 0.0133 Preprocessor1\_Model1  
## 2 f\_meas binary 0.638 2 0.0317 Preprocessor1\_Model1  
## 3 kap binary 0.130 2 0.0147 Preprocessor1\_Model1  
## 4 precision binary 0.605 2 0.000506 Preprocessor1\_Model1  
## 5 recall binary 0.679 2 0.0718 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.568 2 0.0200 Preprocessor1\_Model1  
## 7 sens binary 0.679 2 0.0718 Preprocessor1\_Model1  
## 8 spec binary 0.449 2 0.0593 Preprocessor1\_Model1

## Random Forest  
  
rf\_res <-  
 rf\_wflow %>%   
 fit\_resamples(  
 resamples = bechdel\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas,   
 accuracy, kap,  
 roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE)  
 )   
  
rf\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.634 2 0.00963 Preprocessor1\_Model1  
## 2 f\_meas binary 0.695 2 0.00703 Preprocessor1\_Model1  
## 3 kap binary 0.244 2 0.0206 Preprocessor1\_Model1  
## 4 precision binary 0.645 2 0.00826 Preprocessor1\_Model1  
## 5 recall binary 0.752 2 0.00526 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.642 2 0.0273 Preprocessor1\_Model1  
## 7 sens binary 0.752 2 0.00526 Preprocessor1\_Model1  
## 8 spec binary 0.487 2 0.0151 Preprocessor1\_Model1

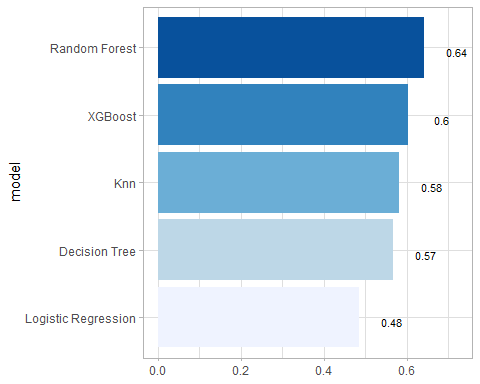
## Boosted tree - XGBoost  
  
xgb\_res <-   
 xgb\_wflow %>%   
 fit\_resamples(  
 resamples = bechdel\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas,   
 accuracy, kap,  
 roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE)  
 )   
  
xgb\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.603 2 0.000712 Preprocessor1\_Model1  
## 2 f\_meas binary 0.647 2 0.00282 Preprocessor1\_Model1  
## 3 kap binary 0.194 2 0.00339 Preprocessor1\_Model1  
## 4 precision binary 0.638 2 0.00318 Preprocessor1\_Model1  
## 5 recall binary 0.656 2 0.00917 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.604 2 0.00939 Preprocessor1\_Model1  
## 7 sens binary 0.656 2 0.00917 Preprocessor1\_Model1  
## 8 spec binary 0.537 2 0.0130 Preprocessor1\_Model1

## K-nearest neighbour  
  
knn\_res <-   
 knn\_wflow %>%   
 fit\_resamples(  
 resamples = bechdel\_folds,   
 metrics = metric\_set(  
 recall, precision, f\_meas,   
 accuracy, kap,  
 roc\_auc, sens, spec),  
 control = control\_resamples(save\_pred = TRUE)  
 )   
  
knn\_res %>% collect\_metrics(summarize = TRUE)

## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.552 2 0.00170 Preprocessor1\_Model1  
## 2 f\_meas binary 0.711 2 0.00141 Preprocessor1\_Model1  
## 3 kap binary -0.00359 2 0.00359 Preprocessor1\_Model1  
## 4 precision binary 0.553 2 0.000708 Preprocessor1\_Model1  
## 5 recall binary 0.997 2 0.00325 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.581 2 0.0113 Preprocessor1\_Model1  
## 7 sens binary 0.997 2 0.00325 Preprocessor1\_Model1  
## 8 spec binary 0 2 0 Preprocessor1\_Model1

## Model Comparison  
  
log\_metrics <-   
 log\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 # add the name of the model to every row  
 mutate(model = "Logistic Regression")   
  
tree\_metrics <-   
 tree\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "Decision Tree")  
  
rf\_metrics <-   
 rf\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "Random Forest")  
  
xgb\_metrics <-   
 xgb\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "XGBoost")  
  
knn\_metrics <-   
 knn\_res %>%   
 collect\_metrics(summarise = TRUE) %>%  
 mutate(model = "Knn")  
  
# create dataframe with all models  
model\_compare <- bind\_rows(log\_metrics,  
 tree\_metrics,  
 rf\_metrics,  
 xgb\_metrics,  
 knn\_metrics)   
  
#Pivot wider to create barplot  
 model\_comp <- model\_compare %>%   
 select(model, .metric, mean, std\_err) %>%   
 pivot\_wider(names\_from = .metric, values\_from = c(mean, std\_err))   
  
# show mean are under the curve (ROC-AUC) for every model  
model\_comp %>%   
 arrange(mean\_roc\_auc) %>%   
 mutate(model = fct\_reorder(model, mean\_roc\_auc)) %>% # order results  
 ggplot(aes(model, mean\_roc\_auc, fill=model)) +  
 geom\_col() +  
 coord\_flip() +  
 scale\_fill\_brewer(palette = "Blues") +  
 geom\_text(  
 size = 3,  
 aes(label = round(mean\_roc\_auc, 2),   
 y = mean\_roc\_auc + 0.08),  
 vjust = 1  
 )+  
 theme\_light()+  
 theme(legend.position = "none")+  
 labs(y = NULL)



## `last\_fit()` on test set  
  
# - `last\_fit()` fits a model to the whole training data and evaluates it on the test set.   
# - provide the workflow object of the best model as well as the data split object (not the training data).   
   
last\_fit\_xgb <- last\_fit(xgb\_wflow,   
 split = data\_split,  
 metrics = metric\_set(  
 accuracy, f\_meas, kap, precision,  
 recall, roc\_auc, sens, spec))  
  
last\_fit\_xgb %>% collect\_metrics(summarize = TRUE)

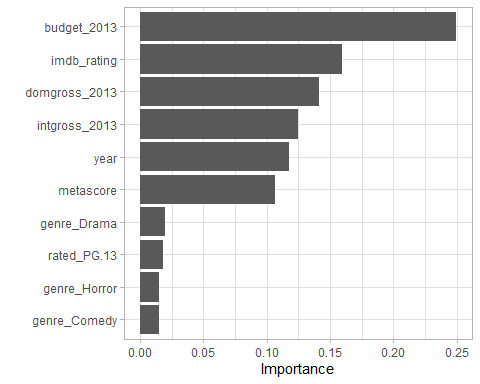
## # A tibble: 8 × 4  
## .metric .estimator .estimate .config   
## <chr> <chr> <dbl> <chr>   
## 1 accuracy binary 0.568 Preprocessor1\_Model1  
## 2 f\_meas binary 0.630 Preprocessor1\_Model1  
## 3 kap binary 0.114 Preprocessor1\_Model1  
## 4 precision binary 0.599 Preprocessor1\_Model1  
## 5 recall binary 0.665 Preprocessor1\_Model1  
## 6 sens binary 0.665 Preprocessor1\_Model1  
## 7 spec binary 0.448 Preprocessor1\_Model1  
## 8 roc\_auc binary 0.610 Preprocessor1\_Model1

#Compare to training  
xgb\_res %>% collect\_metrics(summarize = TRUE)

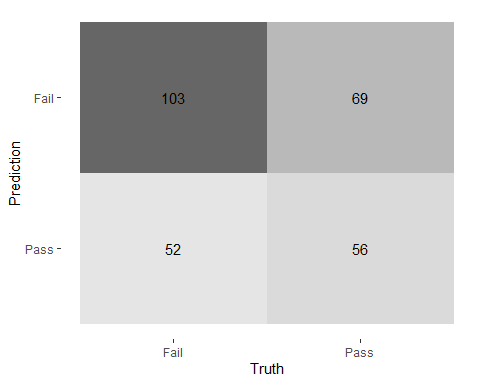
## # A tibble: 8 × 6  
## .metric .estimator mean n std\_err .config   
## <chr> <chr> <dbl> <int> <dbl> <chr>   
## 1 accuracy binary 0.603 2 0.000712 Preprocessor1\_Model1  
## 2 f\_meas binary 0.647 2 0.00282 Preprocessor1\_Model1  
## 3 kap binary 0.194 2 0.00339 Preprocessor1\_Model1  
## 4 precision binary 0.638 2 0.00318 Preprocessor1\_Model1  
## 5 recall binary 0.656 2 0.00917 Preprocessor1\_Model1  
## 6 roc\_auc binary 0.604 2 0.00939 Preprocessor1\_Model1  
## 7 sens binary 0.656 2 0.00917 Preprocessor1\_Model1  
## 8 spec binary 0.537 2 0.0130 Preprocessor1\_Model1

## Variable importance using `{vip}` package  
  
library(vip)  
  
last\_fit\_xgb %>%   
 pluck(".workflow", 1) %>%   
 pull\_workflow\_fit() %>%   
 vip(num\_features = 10) +  
 theme\_light()

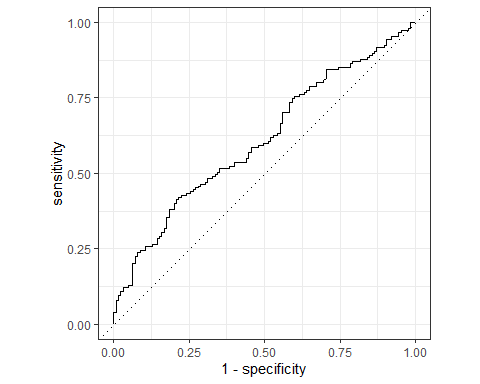
## Warning: `pull\_workflow\_fit()` was deprecated in workflows 0.2.3.  
## ℹ Please use `extract\_fit\_parsnip()` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.



## Final Confusion Matrix  
  
last\_fit\_xgb %>%  
 collect\_predictions() %>%   
 conf\_mat(test, .pred\_class) %>%   
 autoplot(type = "heatmap")



## Final ROC curve  
last\_fit\_xgb %>%   
 collect\_predictions() %>%   
 roc\_curve(test, .pred\_Fail) %>%   
 autoplot()



# Deliverables

There is a lot of explanatory text, comments, etc. You do not need these, so delete them and produce a stand-alone document that you could share with someone. Knit the edited and completed R Markdown (Rmd) file as a Word or HTML document (use the “Knit” button at the top of the script editor window) and upload it to Canvas. You must be commiting and pushing your changes to your own Github repo as you go along.

# Details

* Who did you collaborate with: TYPE NAMES HERE
* Approximately how much time did you spend on this problem set: ANSWER HERE
* What, if anything, gave you the most trouble: ANSWER HERE

**Please seek out help when you need it,** and remember the [15-minute rule](https://dsb2023.netlify.app/syllabus/#the-15-minute-rule). You know enough R (and have enough examples of code from class and your readings) to be able to do this. If you get stuck, ask for help from others, post a question on Slack– and remember that I am here to help too!

As a true test to yourself, do you understand the code you submitted and are you able to explain it to someone else?

# Rubric

13/13: Problem set is 100% completed. Every question was attempted and answered, and most answers are correct. Code is well-documented (both self-documented and with additional comments as necessary). Used tidyverse, instead of base R. Graphs and tables are properly labelled. Analysis is clear and easy to follow, either because graphs are labeled clearly or you’ve written additional text to describe how you interpret the output. Multiple Github commits. Work is exceptional. I will not assign these often.

8/13: Problem set is 60–80% complete and most answers are correct. This is the expected level of performance. Solid effort. Hits all the elements. No clear mistakes. Easy to follow (both the code and the output). A few Github commits.

5/13: Problem set is less than 60% complete and/or most answers are incorrect. This indicates that you need to improve next time. I will hopefully not assign these often. Displays minimal effort. Doesn’t complete all components. Code is poorly written and not documented. Uses the same type of plot for each graph, or doesn’t use plots appropriate for the variables being analyzed. No Github commits.