Predicting Tumor Diagnoses

Using Machine Learning to Predict Whether or Not a Breast Cancer Tumor is Benign or Malignant

Elena Markova

UCSB Spring 2023

Introduction

Alright, for this one prepare yourself for a very uplifting topic: cancer. First let's start with how cancer comes about. A tumor is a collection of cells that form an abnormal mass of tissue. When a tumor is discovered, the first and very important step is to figure out whether the mass is benign or malignant. Benign tumors, while possibly painful and dangerous, do not carry the same danger as malignant tumors. If a tumor is benign, it generally won't spread, but if a tumor is malignant the tumor cells are more likely to metastasize or travel to other areas of the body. This tendency to invade other parts of the body is what makes malignant tumors very threatening. The aim of this project is to create a machine learning model that predicts whether a breast cancer tumor is benign or malignant based off of various physical measurements and features of the tumor. The data used in these models has been taken from the Diagnostic Wisconsin Breast Cancer Database but was originally found on Kaggle/UCI Machine Learning Repository (https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data).

Inspiration and Motive

Everyone is aware of cancer and the chaos that surrounds it but not everyone has had the personal experience to truly internalize the process of the disease. Many times, people find malignant tumors too late and the treatment becomes infinitely more complicated. In my case, I am unfortunately all too familiar with the details of cancer diagnoses - after seeing two other family members go through cancer diagnoses with varying levels of success, my sister got diagnosed with Stage III breast cancer at 24 years old. The very first test to determine that was checking if the tumor they found was benign or malignant. Being surrounded by the troubles of cancer over the past several years has really lit a fire in me to try and find ways to use my skills to help cancer patients. This project marks the beginning of my attempts in using machine learning to help people like my sister who get blindsided by these diagnoses. It may seem like a simple question - is a tumor benign or malignant - but in reality it makes all the difference in people's cancer experiences. So the goal of this project is to begin the process of using machine learning to aid cancer patients through their journeys and hopefully even help people prevent or lower their chances of getting diagnosed with cancer.

Exploring the Data

Loading and Tidying the Dataset

So with that, we begin our deep dive into this project and dataset. We'll begin by loading the dataset and tidying it up - when I downloaded it there was an extra empty row so we'll take that out and clean up the variable names for ease of use.

```r
data <- read.csv("data/ben:mal.csv")
data <- data[,1:32]
data <- data %>%
  clean_names() %>%
  mutate(diagnosis = factor(diagnosis, levels=c('M', 'B')))
```

An important first question here is checking to see if there is any missing data since we are taking data that someone else has already compiled. Since it is simply physical measurements and it is part of university research, we are lucky to find out that there is no missing data in this dataset. YAY! That means a whole lot less problems for us and a complete base of data to work with.

```
sum(is.na(data))
```

```
## [1] 0
```

Describing the Predictors

With this dataset, we are using a variety of physical measurements of the tumor features in order to predict whether it is benign or malignant. There are 10 phhysical measurements of the tumor and for each we have a mean, standard error, and a "worst" value. Here is a list of the variables used in these models:

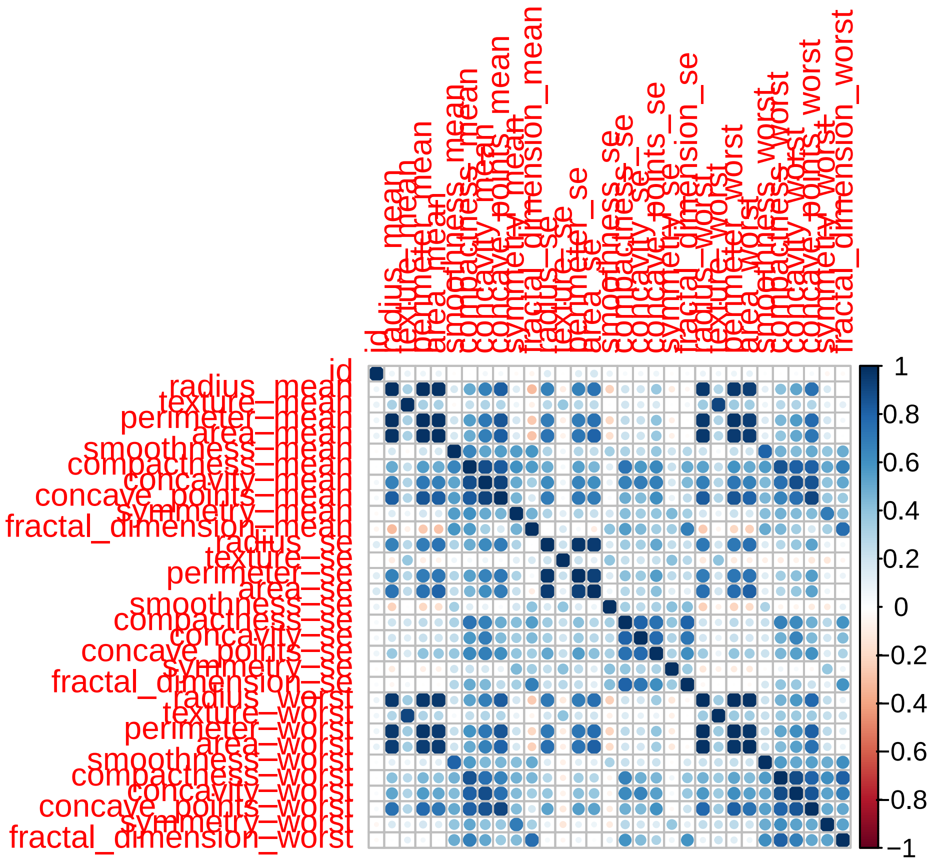
- `id`: ID number of the tumor specimen in the sample that was conducted
- `diagnosis`: The diagnosis of the tumor : either Benign (0) or Malignant (1)
- `radius_mean`: The mean radius of the tumor (mean of distances from center to points on the perimeter)
- `radius_se`: The standard error of the radius of the tumor (mean of distances from center to points on the perimeter)
- `radius_worst`: The worst radius of the tumor (mean of distances from center to points on the perimeter)
- `texture_mean`: The mean texture of the tumor (standard deviation of gray-scale values)
- `texture_se`: The standard error of the texture of the tumor (standard deviation of gray-scale values)
- `texture_worst`: The worst texture of the tumor (standard deviation of gray-scale values)
- `perimeter_mean`: The mean perimeter of the tumor
- `perimeter_se`: The standard error of the perimeter of the tumor
- `perimeter_worst`: The worst perimeter of the tumor
- `area_mean`: The mean area of the tumor
- `area_se`: The standard error of the area of the tumor
- `area_worst`: The worst area of the tumor
- `smoothness_mean`: The mean smoothness of the tumor (local variation in radius lengths)
- `smoothness_se`: The standard error of the smoothness of the tumor (local variation in radius lengths)
- `smoothness_worst`: The worst smoothness of the tumor (local variation in radius lengths)
- `compactness_mean`: The mean compactness of the tumor (perimeterˆ2 / area - 1.0)
- `compactness_se`: The standard error of the compactness of the tumor (perimeterˆ2 / area - 1.0)
- `compactness_worst`: The worst compactness of the tumor (perimeterˆ2 / area - 1.0)
- `concavity_mean`: The mean concavity of the tumor (severity of concave portions of the contour)
- `concavity_se`: The standard error of the concavity of the tumor (severity of concave portions of the contour)
- `concavity_worst`: The worst concavity of the tumor (severity of concave portions of the contour)
- `concave_points_mean`: The mean number of concave portions of the contour
- `concave_points_se`: The standard error of the number of concave portions of the contour
- `concave_points_worst`: The worst number of concave portions of the contour
- `symmetry_mean`: The mean symmetry of the tumor
- `symmetry_se`: The standard error of the symmetry of the tumor

- `symmetry_worst`: The worst symmetry of the tumor
- `fractal_dimension_mean`: The mean fractal dimension ("coastline approximation" - 1)
- `fractal_dimension_se`: The standard error of the fractal dimension ("coastline approximation" - 1)
- `fractal_dimension_worst`: The worst fractal dimension ("coastline approximation" - 1)

Visual Exploratory Data Analysis

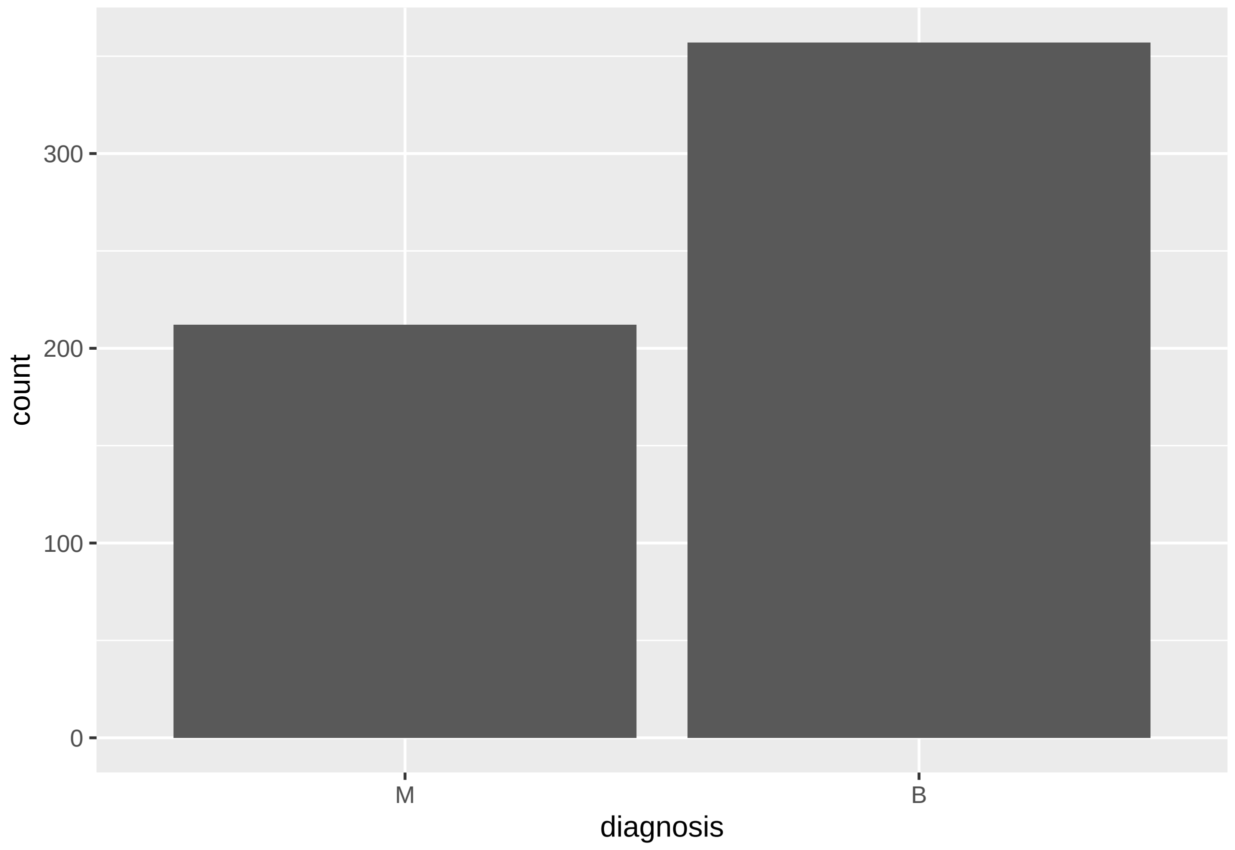
Now that we know what our variables are, we want to look at some graphs and plots to visualize what the relationships are between the predictors and our outcome variable, `diagnosis`. Our first step is to create a correlation matrix plot to see which of the predictors seem to be the most correlated with our outcome variable of diagnosis. We can see that a lot of the variables are positively correlated with the outcome which makes sense since physical attributes of a tumor will definitely be correlated with whether the tumor is benign or malignant.

```
data %>%
  select(where(is.numeric)) %>%
  cor() %>%
  corrplot()
```



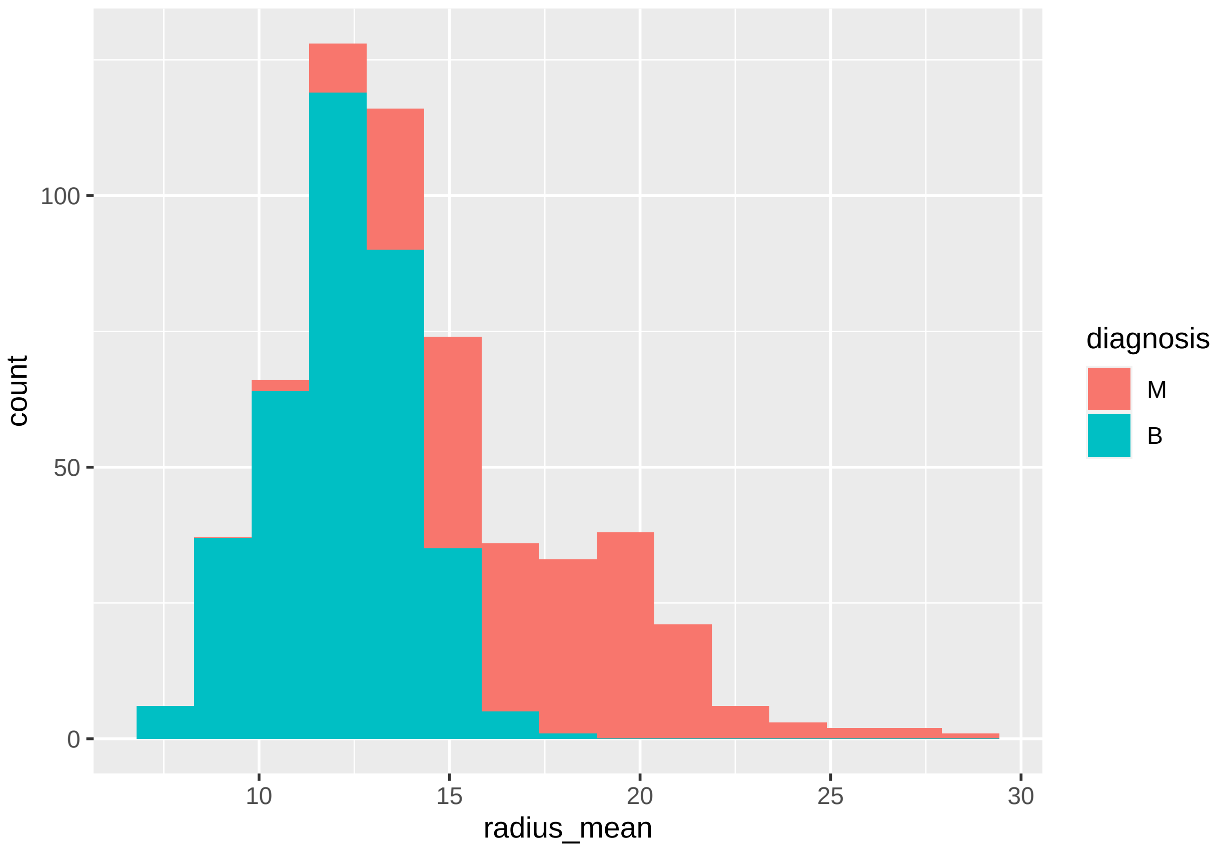
Next, we want to look at the general distribution of the dataset - we can see that there are almost double the amount of Benign diagnoses in this datset compared to Malignant diagnoses.

```
data %>%
  ggplot(aes(x=diagnosis)) +
  geom_bar()
```

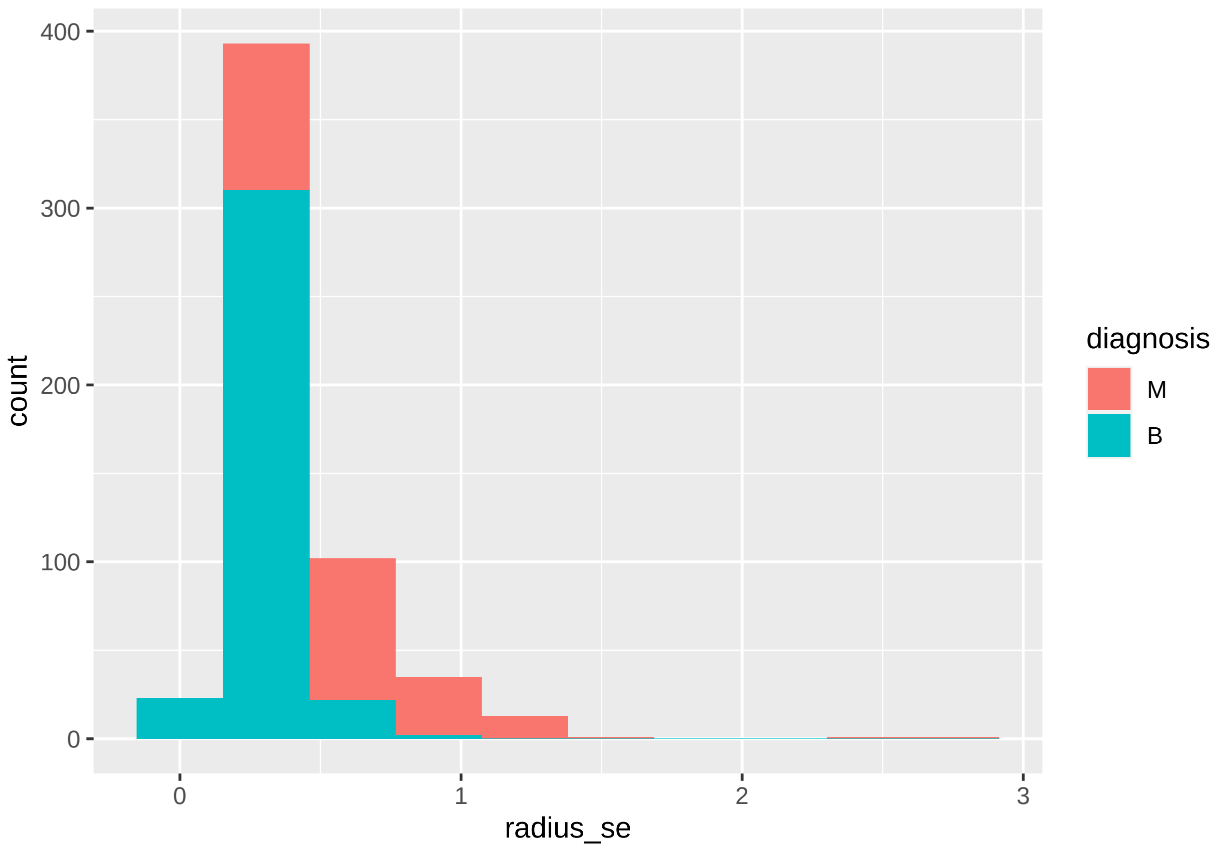


Here, we want to get a sense of the difference in values of the “mean”, “se”, and “worst” categories.

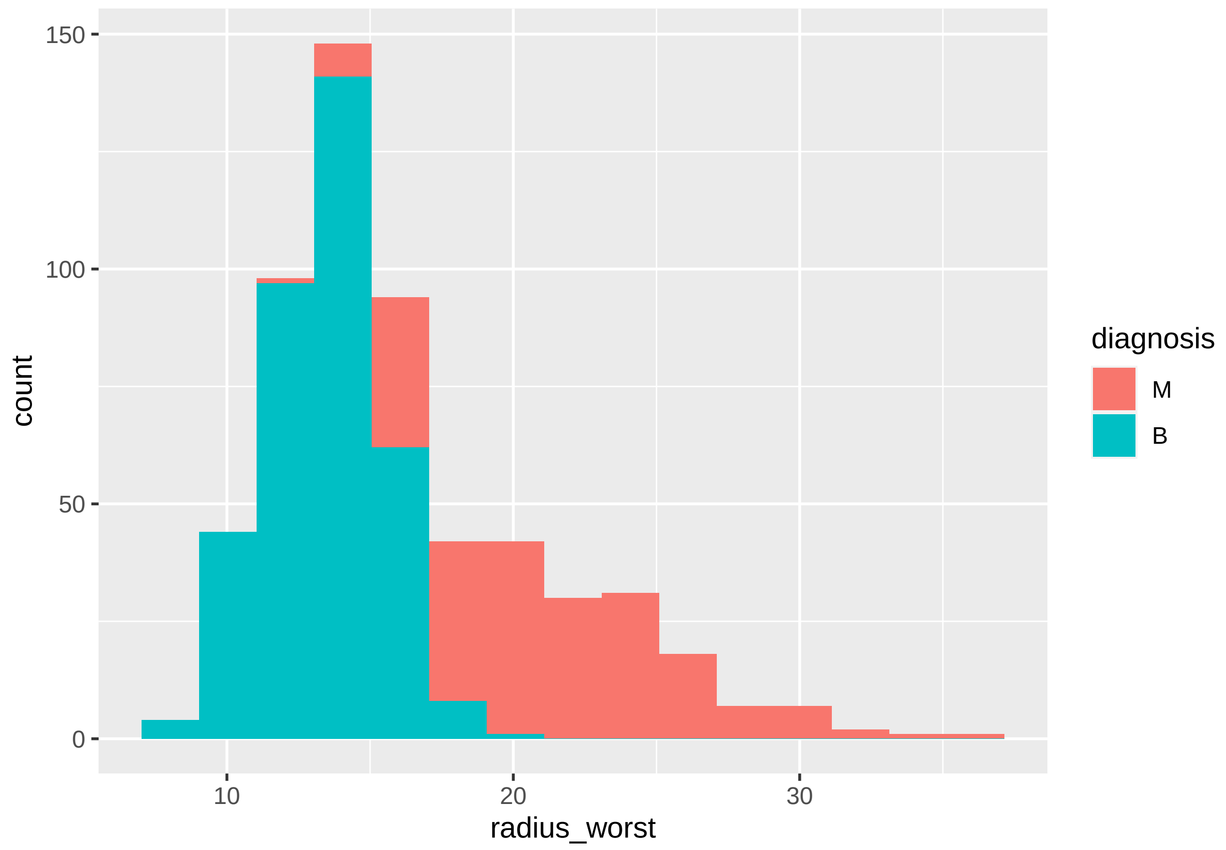
```
data %>%
  ggplot(aes(x=radius_mean, fill=diagnosis)) +
  geom_histogram(bins = 15)
```



```
data %>%
  ggplot(aes(x=radius_se, fill=diagnosis)) +
  geom_histogram(bins=10)
```

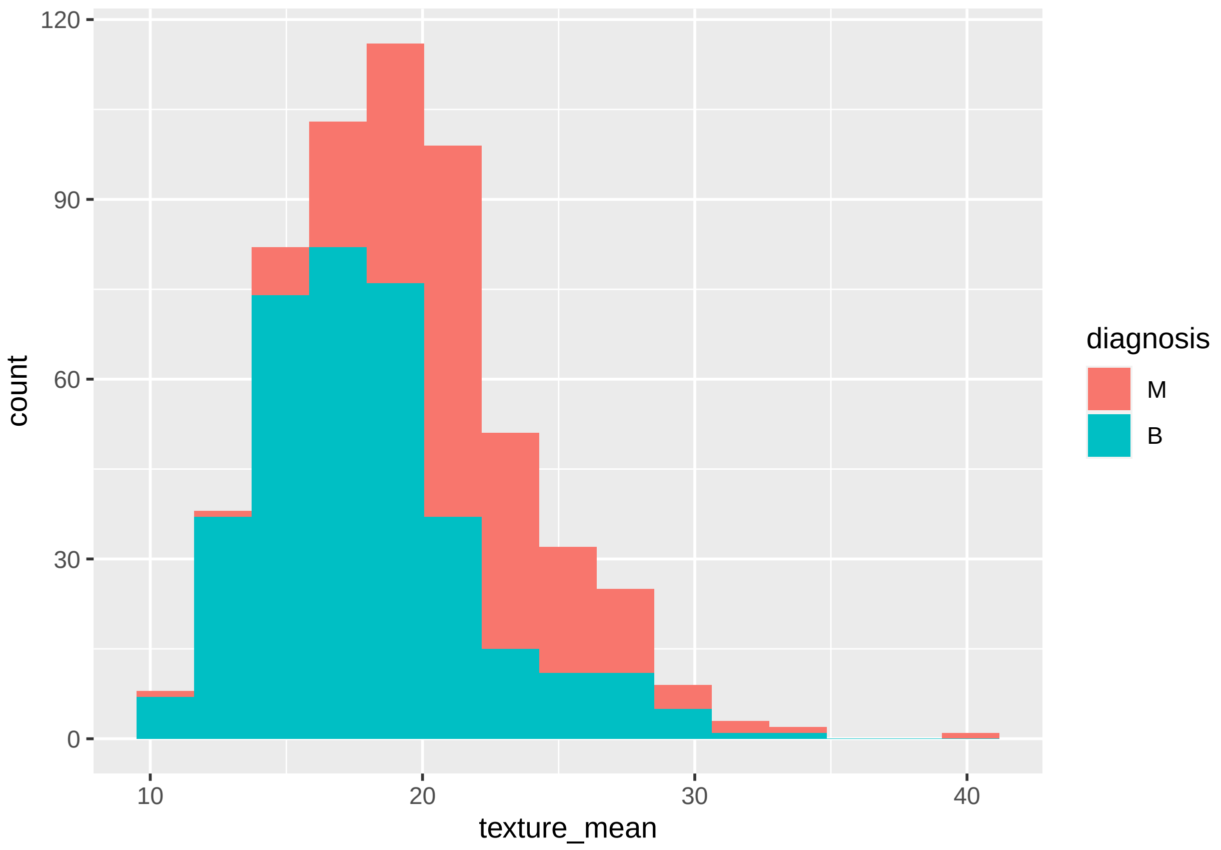


```
data %>%
  ggplot(aes(x=radius_worst, fill=diagnosis)) +
  geom_histogram(bins=15)
```

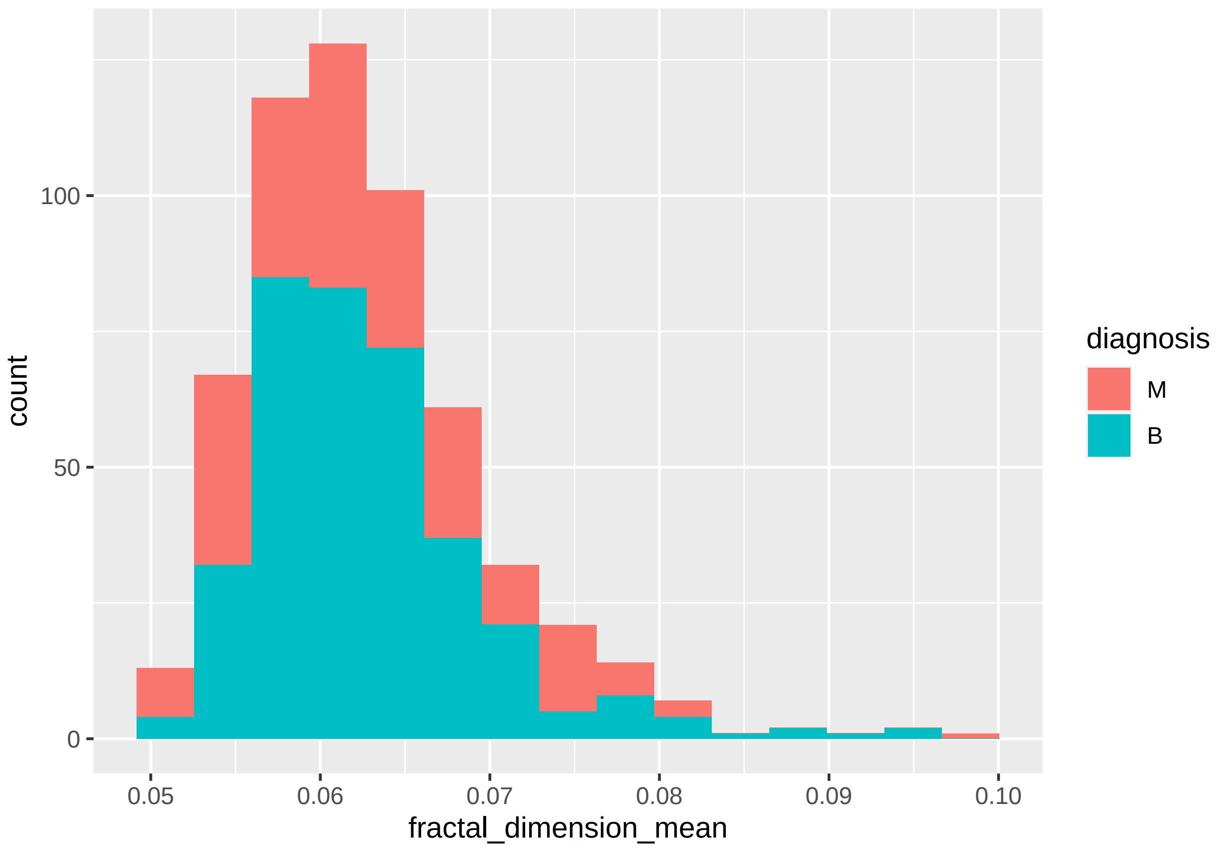


Just from this check alone, we can start to see the tumors with larger radii tend to be Malignant. Let's do a similar check with some of the other variables. This time, we can see that the fractal dimension measurement doesn't show a clear relationship, however, we do see that tumors with more variation in texture tend to be Malignant.

```
data %>%
  ggplot(aes(x=texture_mean, fill=diagnosis)) +
  geom_histogram(bins = 15)
```



```
data %>%
  ggplot(aes(x=fractal_dimension_mean, fill=diagnosis)) +
  geom_histogram(bins = 15)
```



Setting Up Models

Now that we understand our data a little bit better, we can move onto setting up some models. To do this, we are going to do a train/test split, create a recipe, and establish cross-validation.

Training and Testing Data Split

When creating a model based off of one dataset, we need to split up our data into a training set and a test set, stratifying on the response variable `diagnosis`. The training set allows us to use part of the data and work through our models. Then we can use that fitted model on our separate testing data to get a sense of whether or not the model works on new data.

```
set.seed(3435)
data_split <- initial_split(data, strata = diagnosis, prop = 0.75)
train_data <- training(data_split)
test_data <- testing(data_split)
```

I chose a proportion 0.75 so that we can have quite a bit of data to train the model on but still enough data left over to test on and have the calculations be as accurate as possible. We can see here that the training set has 426 observations and the testing set has 143 observations.

```
dim(train_data)
```

```
## [1] 426  32
```

```
dim(test_data)
```

```
## [1] 143  32
```

Recipe Building

Now that we have our training set all ready to go, we are going to take all of those variables and throw them together. They're all going to act as ingredients in our recipe to make a final product of a diagnosis.

```r
recipe <- recipe(diagnosis ~ radius_mean + texture_mean + perimeter_mean + area_mean + smoothness_mean
                 + compactness_mean + concavity_mean + concave_points_mean + symmetry_mean
                 + fractal_dimension_mean + radius_se + texture_se + perimeter_se + area_se
                 + smoothness_se + compactness_se + concavity_se + concave_points_se + symmetry_se
                 + fractal_dimension_se + radius_worst + texture_worst + perimeter_worst + area_worst
                 + smoothness_worst + compactness_worst + concavity_worst + concave_points_worst
                 + symmetry_worst + fractal_dimension_worst, data = data) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_predictors())
```

K-Fold Cross Validation

An important step here before creating the models, is to implement cross validation. Since we are working on a smaller dataset, we want to set up a process of resampling so that we can get a result as if we had a much larger dataset. This will ensure that we have a more general model as our final product. We need to stratify on our response variable and we are going to create 5 folds.

```r
folds <- vfold_cv(train_data, v = 5, strata = diagnosis)
```

Building Prediction Models

So, since these models can get complicated fairly quickly and take up a lot of computing time, we want to do all of the actual grunt work in separate files so that we don't end up running the model every single time we want to play around in this Rmd file. Each model has its own R script using our model setup we made in the steps above and then that information was stored in an RDA file. Now we can simply load those results and explore!

The models that I have run for this prediction are: K-Nearest Neighbors, Linear Discriminant Analysis, Quadratic Discriminant Analysis, and Elastic Net Logistic Regression.

```r
save(folds, recipe, train_data, test_data, file="model_setup.rda")
load("knn_model.rda")
load("lda_model.rda")
load("qda_model.rda")
load("enlr_model.rda")
```

Model Building Process

The models all follow this basic set up:

1. We specify the type of model (classification in our case) and then we specify the engine and mode.
2. We set up a workflow that includes our new model and the recipe we created above.
3. Then we use a tuning grid in order to tune all of our hyperparameters.
4. We pick the most accurate model and fit our finalized workflow to the training set.

Model Results

As a measure of how well our model is working, we will be using the metric of `roc_auc` with this metric, the closer the value is to 1 the better our model predicts the outcome.

K-Nearest Neighbors

The k-nearest neighbors algorithm (KNN) is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. The results of our model show that the optimal number of neighbors to look at is 7 and this results in an `roc_auc` value of 0.988 which is amazing! This model ends up predicting the outcome very well.

```
# collect_metrics(tune_res_knn)
show_best(tune_res_knn, metric = "roc_auc")
```

```
## # A tibble: 5 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1         7 roc_auc binary     0.988     5 0.00546 Preprocessor1_Model07
## 2         6 roc_auc binary     0.987     5 0.00503 Preprocessor1_Model06
## 3         5 roc_auc binary     0.987     5 0.00474 Preprocessor1_Model05
## 4         9 roc_auc binary     0.987     5 0.00587 Preprocessor1_Model09
## 5         8 roc_auc binary     0.987     5 0.00556 Preprocessor1_Model08
```

Linear Discriminant Analysis (LDA)

Linear discriminant analysis (LDA) is a generalization of Fisher's linear discriminant, a method used in statistics and other fields, to find a linear combination of features that characterizes or separates two or more classes of objects or events. The results of our model show that this results in an `roc_auc` value of 0.988 again. Our models are doing very well so far - I don't know how we'll pick!

```
# collect_metrics(lda_kfold_fit)
show_best(lda_kfold_fit, metric = "roc_auc")
```

```
## # A tibble: 1 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 roc_auc binary     0.988     5 0.00503 Preprocessor1_Model1
```

Quadratic Discriminant Analysis (QDA)

A quadratic discriminant analysis is a statistical classifier model that uses a quadratic decision surface to separate measurements of two or more classes of objects or events. It is a more advanced version of a linear model as it is used to find a non-linear boundary between our classifiers, and assumes that each class follows a Gaussian distribution. The results of our model show that this results in an `roc_auc` value of 0.99 which is a bit higher than the others!

```
# collect_metrics(qda_kfold_fit)
show_best(qda_kfold_fit, metric = "roc_auc")
```

```
## # A tibble: 1 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 roc_auc binary     0.989     5 0.00519 Preprocessor1_Model1
```

Elastic Net Logistic Regression

Elastic Net Logistic Regression is a model that essentially allows us to fit both Lasso, Ridge, and any mix of the two regressions. The results of our model show that the optimal values of the hyperparameters penalty and mixture are both 0 - this means that a Ridge regression is optimal. This model results in an `roc_auc` value of 0.995. I think we've found our winner among winners folks!

```r
# collect_metrics(tune_res_enlr)
show_best(tune_res_enlr, metric = "roc_auc")
```

```
## # A tibble: 5 x 8
##   penalty mixture .metric .estimator  mean     n std_err .config
##     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1   0       0     roc_auc binary     0.995     5 0.00289 Preprocessor1_Model001
## 2   0.111   0     roc_auc binary     0.994     5 0.00343 Preprocessor1_Model002
## 3   0       0.111 roc_auc binary     0.993     5 0.00372 Preprocessor1_Model011
## 4   0.111   0.111 roc_auc binary     0.992     5 0.00387 Preprocessor1_Model012
## 5   0.222   0     roc_auc binary     0.992     5 0.00398 Preprocessor1_Model003
```

Putting the Model to the Test

After tuning the Elastic Net Logistic Regression Model, we find that the `roc_auc` on the training data is 0.995. Let's try it on the testing data and see if our model REALLY predicts the outcome well. . . .

```r
best_enlr <- select_by_one_std_err(tune_res_enlr, metric = "roc_auc", penalty, mixture)
best_enlr
```

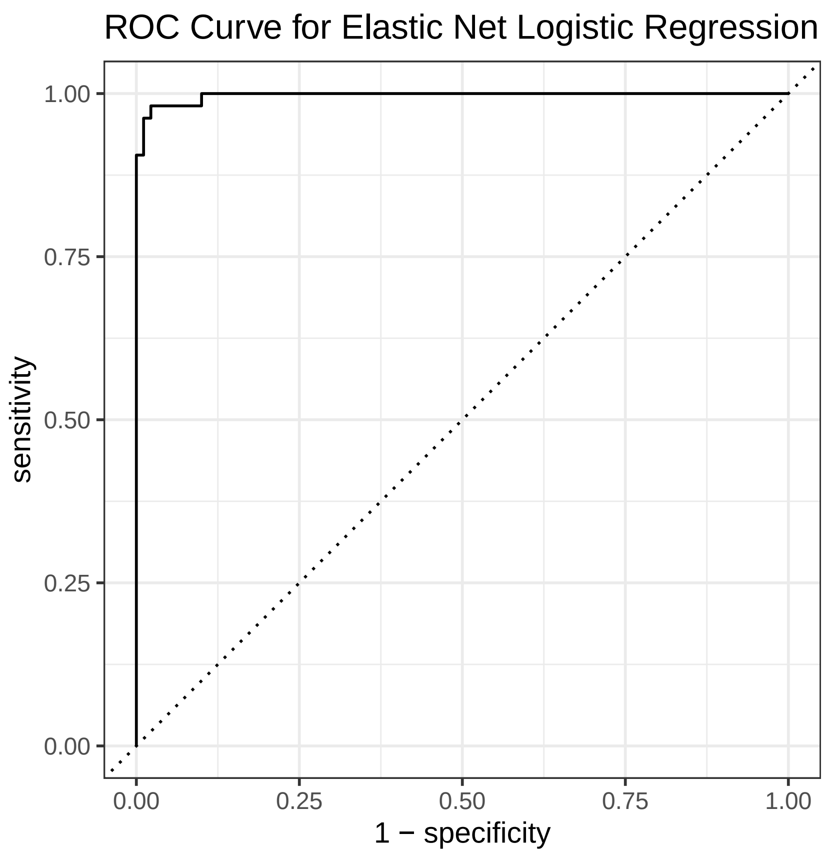
```
## # A tibble: 1 x 10
##   penalty mixture .metric .estimator  mean     n std_err .config    .best .bound
##     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>      <dbl>  <dbl>
## 1       0       0 roc_auc binary     0.995     5 0.00289 Preproces~ 0.995  0.992
```

Here we will fit the model to the testing data and see what happens!!

```r
final_wf_enlr <- finalize_workflow(enlr_wkflow, best_enlr)
final_fit_enlr <- fit(final_wf_enlr, train_data)
augment(final_fit_enlr, new_data = test_data) %>%
  roc_auc(truth = diagnosis, estimate = .pred_M)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.997
```

```r
augment(final_fit_enlr, new_data = test_data) %>%
  roc_curve(diagnosis, .pred_M) %>%
  autoplot() + ggtitle("ROC Curve for Elastic Net Logistic Regression")
```



SUCCESS!!!! Our model ended up getting an `roc_auc` of 0.997 on the TEST data which is crazy amazing! This means that our model didn't it too closely to the training data - it stayed broad enough to accept new data and still predict with very good accuracy.

Conclusion

Surprisingly, we ended up with truly spectacular results at the end of this project! As it turns out, all 4 of the models that we tried out had an `roc_auc` value ABOVE 0.95 which is amazing. This just goes to show us that there is a very strong prediction ability of physical measurements on the outcome variable of `diagnosis`. Seeing that there weren't any models that really performed poorly was very surprising at first. However, thinking about it further it would make sense that these variables specifically would very strongly correlate to the characteristics of a tumor.

All in all, this project was definitely a success. We found that there are many options in terms of models that can predict whether a tumor is benign or malignant. In terms of extension or improvement here, there is always room for improvement but that comes with the tradeoff of simplicity and computing power so really it depends on the case. I would say that we've ended up with a remarkable and truly promising result. It's very encouraging to see that a simple but important question such as this could be predicted well with machine learning.

Coming away from this project I have renewed faith in the direction that technology is heading in, especially in connection with medicine. These computing capabilities have the power to change people's lives for the better. Cancer is one of those life events that has a completely draining effect on a person and everyone connected to it - family, friends, doctors, etc. So if we can harness this power to reduce the severity of cancer, we have the chance to create palpable change in the lives of humans everywhere. The final extension I have is that it's not just cancer - there are so many applications within the medical world and I, for one, am very excited to explore the possibilities and help people live freely and healthily!