PSTAT131 Final

Henry Hsia

2023-12-14

Introduction

The purpose of this project is to develop a machine learning model that can predict the mass of a meteorite based on various factors.

Background

Meteorites are pieces of debris from space that survives its descent through the atmosphere to reach the surface of Earth. Smaller meteorites can cause property damage and injury, while larger meteorites, although infrequent, can cause major damage and destruction.

What are we trying to do?

It seems plausible to me that the chance of a meteorite being large enough to cause substantial damage is directly correlated with when and where it landed. With this project, we can map out which region of the world and which time periods had the most substantial meteorites. Additionally, with our machine learning model, we can try to predict the mass of a meteorite based on its longitude, latitude, year, its composition, and whether the meteorite was observed falling or was found.

Loading, Exploring, and Tidying the Raw Data

First, let's load some basic packages and the raw data.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.4     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(tidymodels)
```

```
## -- Attaching packages -------------------------------------- tidymodels 1.1.1 --
## v broom        1.0.5     v rsample      1.2.0
## v dials        1.2.0     v tune         1.1.2
## v infer        1.0.5     v workflows    1.1.3
## v modeldata    1.2.0     v workflowsets 1.0.1
## v parsnip      1.1.1     v yardstick    1.2.0
```

```
## v recipes      1.0.8
## -- Conflicts ---------------------------------------- tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## * Use suppressPackageStartupMessages() to eliminate package startup messages
```

```
library(janitor)
```

```
##
## Attaching package: 'janitor'
##
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```
meteorite_data <- read.csv("Meteorite_Landings.csv")
head(meteorite_data)
```

```
##       name  id nametype    recclass mass..g. fall year    reclat    reclong
## 1   Aachen   1    Valid          L5       21 Fell 1880  50.77500    6.08333
## 2   Aarhus   2    Valid          H6      720 Fell 1951  56.18333   10.23333
## 3     Abee   6    Valid         EH4   107000 Fell 1952  54.21667 -113.00000
## 4 Acapulco  10    Valid Acapulcoite     1914 Fell 1976  16.88333  -99.90000
## 5  Achiras 370    Valid          L6      780 Fell 1902 -33.16667  -64.95000
## 6 Adhi Kot 379    Valid         EH4     4239 Fell 1919  32.10000   71.80000
##            GeoLocation
## 1    (50.775, 6.08333)
## 2 (56.18333, 10.23333)
## 3   (54.21667, -113.0)
## 4    (16.88333, -99.9)
## 5  (-33.16667, -64.95)
## 6         (32.1, 71.8)
```

This dataset was obtained from data.gov, a website housing the US government's open datasets. The dataset was published and is maintained by NASA, and can be found at the URL: https://catalog.data.gov/dataset/meteorite-landings

The dataset contains information for 34,500 meteorites, with their name, id, type, mass, whether they were observed falling before being found, year, latitude, longitude, and geolocation.

Variable Selection

We don't need all 9 of the initial columns from the raw data. Both name and id seem unnecessary. We will also remove nametype and geolocation in this step. Nametype is related to the name. Geolocation is just a combination of the latitude and longitude variables, so we can just safely remove it.

```
meteorite_data2 <- subset(meteorite_data, select = -c(name, id, nametype, GeoLocation))
head(meteorite_data2)
```

```
##      recclass mass..g. fall year    reclat    reclong
## 1          L5       21 Fell 1880  50.77500    6.08333
## 2          H6      720 Fell 1951  56.18333   10.23333
## 3         EH4   107000 Fell 1952  54.21667 -113.00000
## 4 Acapulcoite     1914 Fell 1976  16.88333  -99.90000
```

```
## 5          L6      780 Fell 1902 -33.16667  -64.95000
## 6         EH4     4239 Fell 1919  32.10000   71.80000
```

Let's clean up the variable names now using the clean_names function from the janitor library now that we have a better filtered dataset.

```
meteorite_data2 <- meteorite_data2 %>%
  clean_names()
```

Converting Factors

Let's convert the categorical variables (recclass and fall) to factors so that they can actually be manipulated by a machine learning model. Right now, these two variables are character variables and can't be handled because machine learning algorithms work with numeric variables. Using the as.factor() function converts these categorical variables into numeric categories so that they can be manipulated by the models we will use in future steps.

```
meteorite_data2$recclass <- as.factor(meteorite_data2$recclass)
meteorite_data2$fall <- as.factor(meteorite_data2$fall)
```

Missing Data

Let's get an even better idea of the dataset with the summary function, specifically how much missing data the dataset contains.

```
meteorite_data2 %>%
  summary()
```

```
##     recclass         mass_g            fall            year
##  L6     : 8285   Min.   :       0   Fell : 1107   Min.   : 860
##  H5     : 7142   1st Qu.:       7   Found:44609   1st Qu.:1987
##  L5     : 4796   Median :      33                 Median :1998
##  H6     : 4528   Mean   :   13278                 Mean   :1992
##  H4     : 4211   3rd Qu.:     203                 3rd Qu.:2003
##  LL5    : 2766   Max.   :60000000                 Max.   :2101
##  (Other):13988   NA's   :131                      NA's   :291
##      reclat          reclong
##  Min.   :-87.37   Min.   :-165.43
##  1st Qu.:-76.71   1st Qu.:   0.00
##  Median :-71.50   Median :  35.67
##  Mean   :-39.12   Mean   :  61.07
##  3rd Qu.:  0.00   3rd Qu.: 157.17
##  Max.   : 81.17   Max.   : 354.47
##  NA's   :7315     NA's   :7315
```

According to the function, there are over 45,000 observations. There are also 7,300 missing values in the latitude and longitude variables. Since in a future step we will likely reduce the size of our data so that it will be easier to work with anyway, let's just remove the pieces of data that contain any missing values.

```
meteorite_data2 <- na.omit(meteorite_data2)

meteorite_data2 %>%
  summary()
```

```
##     recclass         mass_g            fall            year
##  L6     : 7519   Min.   :       0   Fell : 1065   Min.   : 860
##  H5     : 6243   1st Qu.:       7   Found:37050   1st Qu.:1986
##  H6     : 3898   Median :      29                 Median :1996
```

```
##  H4     : 3880   Mean   :   15601                  Mean   :1990
##  L5     : 3264   3rd Qu.:     187                  3rd Qu.:2002
##  LL5    : 2199   Max.   :60000000                  Max.   :2101
##  (Other):11112
##      reclat          reclong
##  Min.   :-87.37   Min.   :-165.43
##  1st Qu.:-76.72   1st Qu.:   0.00
##  Median :-71.50   Median :  35.67
##  Mean   :-39.60   Mean   :  61.31
##  3rd Qu.:  0.00   3rd Qu.: 157.17
##  Max.   : 81.17   Max.   : 178.20
##
```

Removing Null Geolocations

Finally, looking through the data manually reveals that there are way too many observations with a longitude and latitude both at 0.00000. I doubt there are that many meteorites landing and being found at null island off the coast of West Africa, more likely they were not able to pinpoint the location of a meteorite so NASA just set the latitude and longitude to null. While this may slightly skew the data, let's remove all the meteorites with a latitude and longitude of 0.00000.

```
meteorite_data3 <- meteorite_data2[!(meteorite_data2$reclat == 0 & meteorite_data2$reclong == 0), ]

head(meteorite_data3)
```

```
##      recclass mass_g fall year    reclat    reclong
## 1          L5     21 Fell 1880  50.77500    6.08333
## 2          H6    720 Fell 1951  56.18333   10.23333
## 3         EH4 107000 Fell 1952  54.21667 -113.00000
## 4 Acapulcoite   1914 Fell 1976  16.88333  -99.90000
## 5          L6    780 Fell 1902 -33.16667  -64.95000
## 6         EH4   4239 Fell 1919  32.10000   71.80000
```

```
meteorite_data3 %>%
  summary()
```

```
##     recclass        mass_g            fall            year
##  L6     :6523   Min.   :       0   Fell : 1064   Min.   : 860
##  H5     :5586   1st Qu.:       6   Found:30865   1st Qu.:1982
##  H4     :3324   Median :      30                 Median :1991
##  H6     :3231   Mean   :   18543                 Mean   :1987
##  L5     :2723   3rd Qu.:     202                 3rd Qu.:2000
##  LL5    :1897   Max.   :60000000                 Max.   :2013
##  (Other):8645
##      reclat          reclong
##  Min.   :-87.37   Min.   :-165.43
##  1st Qu.:-79.68   1st Qu.:  26.00
##  Median :-72.00   Median :  56.82
##  Mean   :-47.27   Mean   :  73.19
##  3rd Qu.: 18.33   3rd Qu.: 159.39
##  Max.   : 81.17   Max.   : 178.20
##
```

Removing meteorites with a null geolocation removed around 7,000 meteorites. However, with the dataset size still being at 31,000, there should be little statistically significant impact on a dataset this large. Now, with that being done, our data should be ready for analysis and manipulation.

Visual EDA

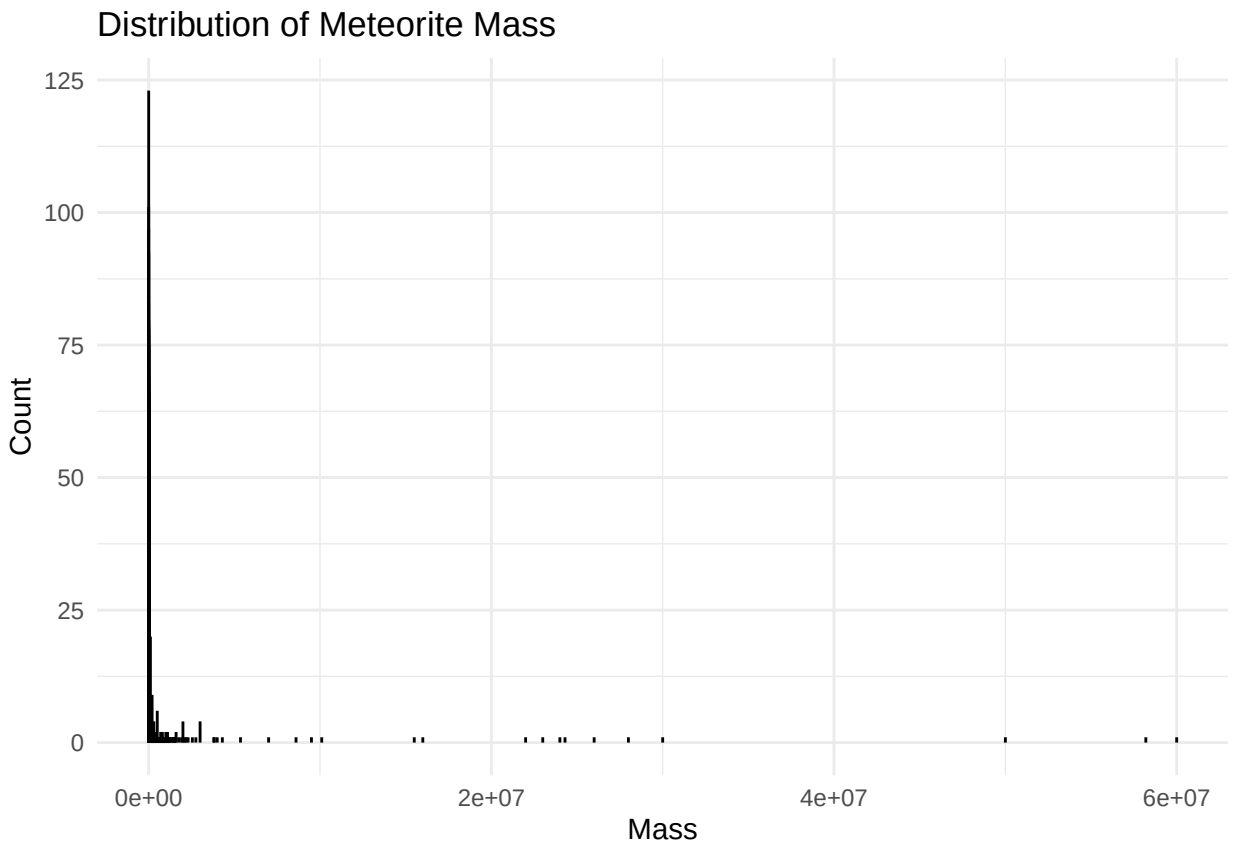
Now, let's visually explore the relationship between the variables.

Distribution of Mass

First, let's explore the distribution of the outcome variable, mass_g.

```
library(ggplot2)

ggplot(meteorite_data3, aes(x = mass_g)) +
  geom_bar(fill = "skyblue", color = "black") +
  labs(title = "Distribution of Meteorite Mass", x = "Mass", y = "Count") +
  theme_minimal()
```

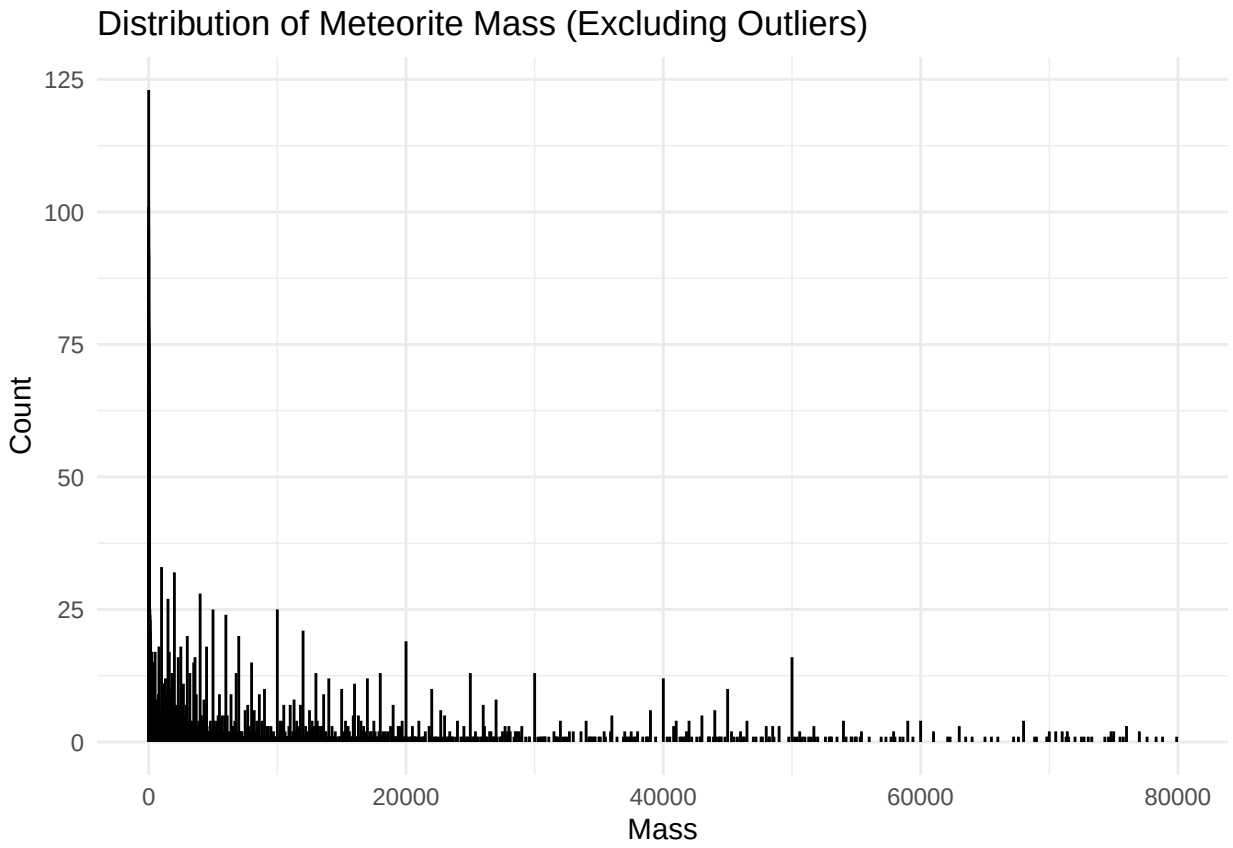


The values of mass range from 0 to 60 million. Looking at the graph, it seems that the majority of observations are on the lower end, which makes sense. However, let's get some more information on the distribution by filtering out the 1st and 99th percentile of values.

```
lower_limit <- quantile(meteorite_data3$mass_g, probs = 0.01)
upper_limit <- quantile(meteorite_data3$mass_g, probs = 0.99)

filtered_data <- subset(meteorite_data3, mass_g > lower_limit & mass_g < upper_limit)

ggplot(filtered_data, aes(x = mass_g)) +
  geom_bar(fill = "skyblue", color = "black") +
  labs(title = "Distribution of Meteorite Mass (Excluding Outliers)", x = "Mass", y = "Count") +
  theme_minimal()
```



Excluding the outliers gives us a better look at the distribution of the other 98% of the data. From our second graph, it seems that the meteorite masses follows a normal distribution centered around a mass of 0, however there are still plenty of meteorites at most mass value intervals up to 80,000.

Meteorite Type

The next variable we will take a look at is recclass. This variable specifies the type and composition of the meteorite, such as being stony, containing small or large amounts of iron, etc. If the classification begins with an L, H or EH, it has a mostly standard meteorite composition, with L specifying low iron, and H and EH specifying a relatively high amount of iron. Other meteorite classifications will usually specify the most prevalent mineral on the meteorite, such as Eucrite, Aubrite, Diogenite, etc.

Let's examine the relation meteorite types have to mass. Before we generate a graph though, there is likely too many types to fit in a graph comfortably. So let's create a new column in meteorite_data3 that groups all of the different types by only their prefix, such as grouping all of the L's, H's, Iron's, Diogenite's, etc.

```
unique_recclasses <- length(unique(meteorite_data3$recclass))
unique_recclasses
```

```
## [1] 392
```

392 unique classes before our grouping.

```
extract_group <- function(recclass) {
  prefix <- gsub("[^A-Za-z]+.*", "", toupper(recclass))
  return(prefix)
}

meteorite_data3$recclass_group <- sapply(meteorite_data3$recclass, extract_group)
```

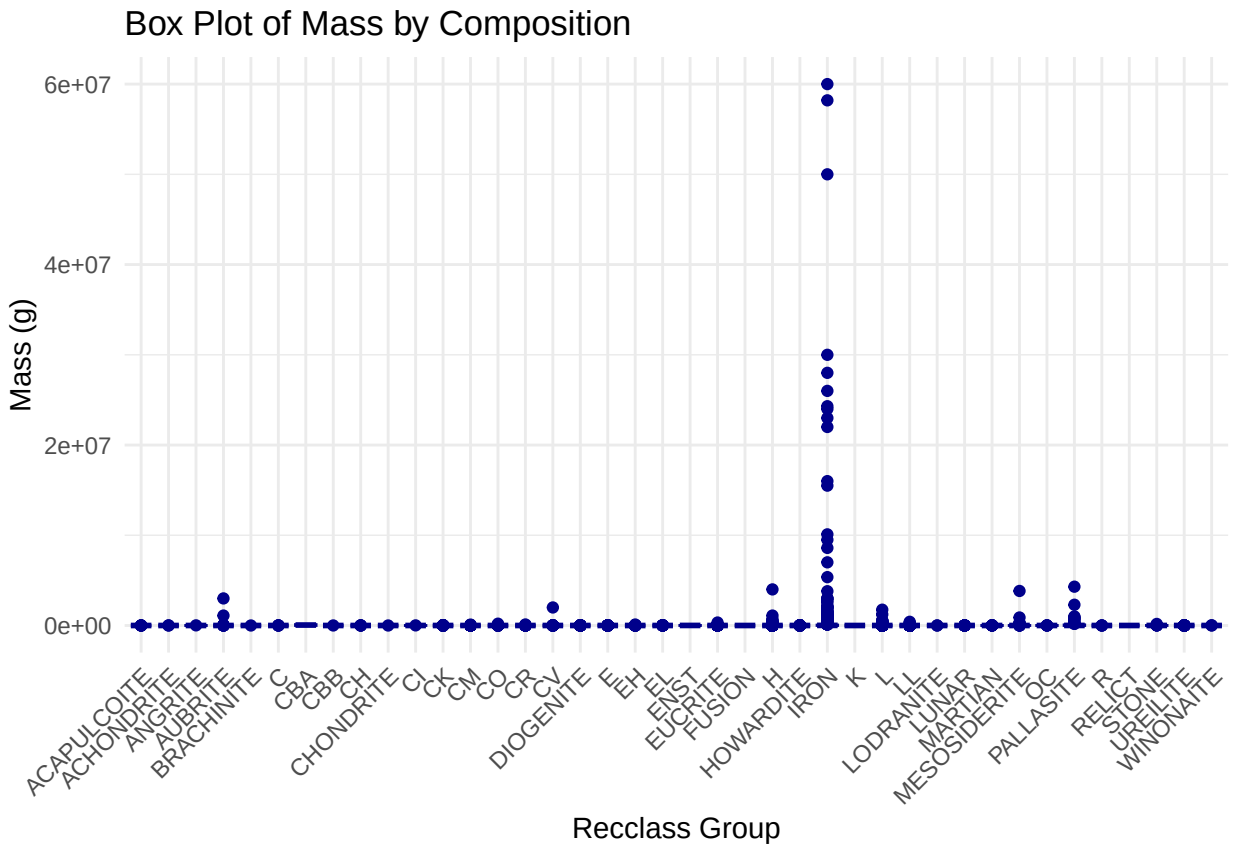
```
unique_recclass_groups <- length(unique(meteorite_data3$recclass_group))
unique_recclass_groups
```

```
## [1] 40
```

40 grouped classes after our function.

Now, let's generate the plot.

```
ggplot(meteorite_data3, aes(x = recclass_group, y = mass_g)) +
  geom_boxplot(fill = "skyblue", color = "darkblue") +
  labs(title = "Box Plot of Mass by Composition", x = "Recclass Group", y = "Mass (g)") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Unsurprisingly, from the boxplot it seems that pretty much every heavy meteorite is of the iron class.

Let's classify our new column as a factor before moving on to the next graph.

```
meteorite_data3$recclass_group <- as.factor(meteorite_data3$recclass_group)

head(meteorite_data3)
```

```
##      recclass mass_g fall year    reclat    reclong recclass_group
## 1          L5     21 Fell 1880  50.77500    6.08333              L
## 2          H6    720 Fell 1951  56.18333   10.23333              H
## 3         EH4 107000 Fell 1952  54.21667 -113.00000             EH
## 4 Acapulcoite   1914 Fell 1976  16.88333  -99.90000    ACAPULCOITE
## 5          L6    780 Fell 1902 -33.16667  -64.95000              L
## 6         EH4   4239 Fell 1919  32.10000   71.80000             EH
```

Correlation Plot

Let's explore the overall relationships between all of the numeric variables using a correlation plot.

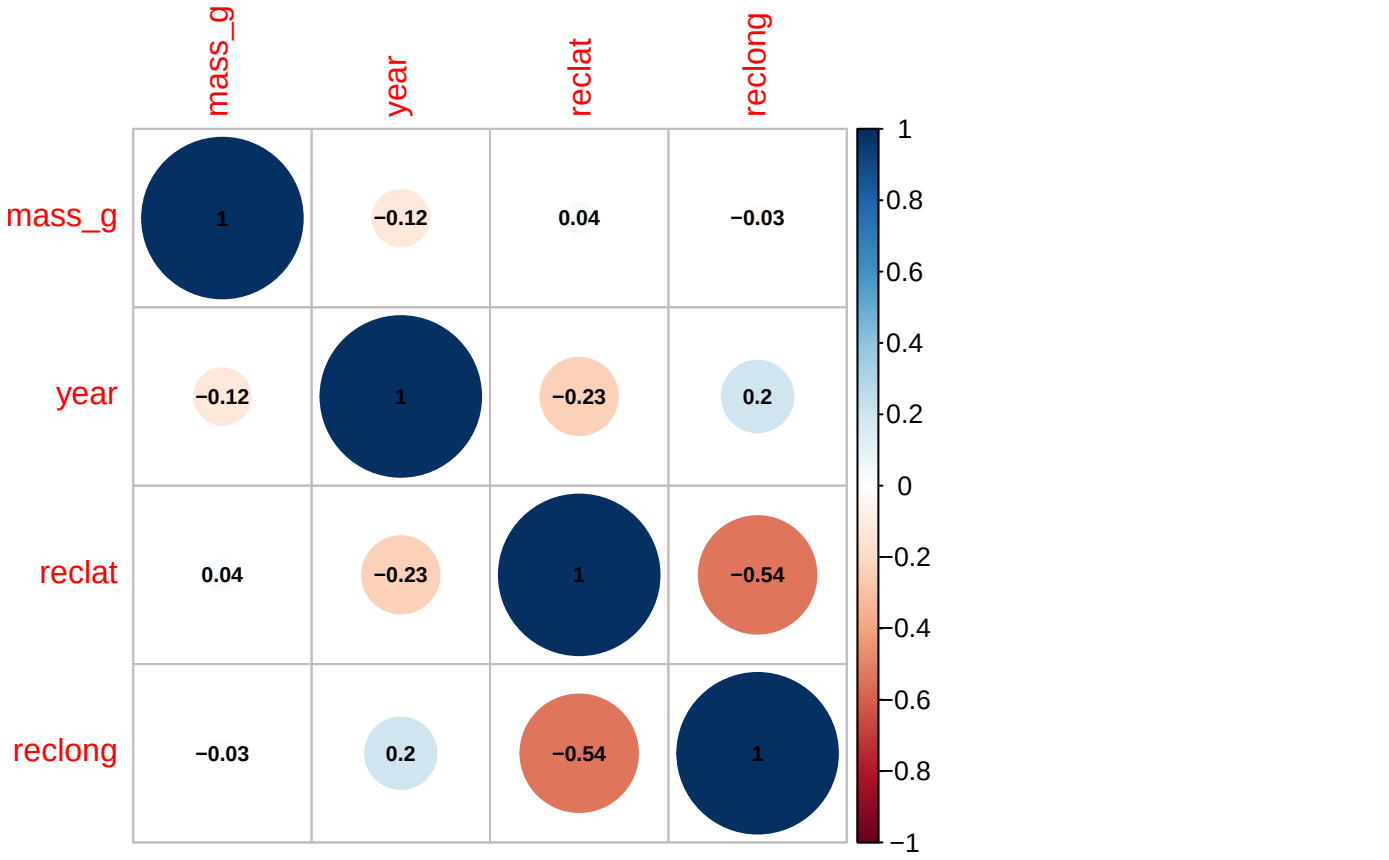
```
library(corrr)
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
meteorite_numeric <- meteorite_data3 %>%
  select_if(is.numeric)

meteorite_cor <- cor(meteorite_numeric)

meteorite_corrplot <- corrplot(meteorite_cor, method = "circle", addCoef.col = 1, number.cex = 0.7)
```



From the correlation plot it seems that there is very little relation between the mass and where a meteorite lands in the world. There might be some small relation between the year and the mass though. There exists more significant relations between the latitude and longitude, and between the year and location.

Conclusions from Visual EDA

Overall, from our visual data analysis, it seems that the most promising variable to predict mass with is recclass (the type/composition of the meteorite). Our initial hypothesis of being able to predict where larger meteorites land might be impossible due to very little relation being present between mass and latitude/longitude. The relation between the year and fell variables and the mass variable is currently less known, so we may still be able to predict mass with these variables as well.

Setting up for Modeling

Let's move on to the modeling step, where we can see if we can actually predict the mass of a meteorite with our predictors. First however, we have to set up our data for modeling by splitting, creating a recipe, and creating folds for k-fold cross validation.

Splitting the Data

The first step before modeling and fitting our data is to split the data into a training and testing set. This is a crucial step for many reasons. Firstly, in order to assess how well our future model actually does at predicting the outcome variable, we need the testing to see how the model does on data it has not seen before. Otherwise, there will be overfitting issues if the model is evaluated with the same data used to train it. Additionally, data splitting prevents overfitting in other ways. Overfitting occurs when the model captures noise and patterns that do not actually exist in the data by fitting the data used to train it too precisely. By having a second set, the testing set, we can assess whether the model has learned the underlying patterns or just memorized the training data. Finally, splitting the data allows us to better estimate how the data might perform on new, previously unseen data.

We will use a 70/30 data split, which is I believe is pretty standard. Our data split will be stratified on our outcome variable, mass_g. This ensures that the training and testing sets have an equal distribution of the outcome variable, mass_g.

```
library(rsample)

set.seed(9721044)

split_data <- initial_split(meteorite_data3, prop = 0.7, strata = "mass_g")

training_data <- training(split_data)
testing_data <- testing(split_data)
```

Let's verify the split before moving on to the next step.

```
nrow(training_data)/nrow(meteorite_data3)
```

```
## [1] 0.699928
```

```
nrow(testing_data)/nrow(meteorite_data3)
```

```
## [1] 0.300072
```

The data was split correctly.

Creating the Recipe

We will be using the same predictors, conditions, and outcomes throughout our modeling and fitting. Therefore, the next step would be to create a recipe to use for all of our models. We will designate mass_g as our outcome variable in all other variables as predictors. Additionally, we will normalize our variables by centering and scaling. Finally, we use step_zv to remove columns with zero variance, as that will cause problems for us later on otherwise.

```
library(recipes)

data_recipe <- recipe(mass_g ~ . , data = training_data) %>%
  step_dummy(all_nominal(), -all_outcomes()) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_zv(all_predictors())
```

```
summary(data_recipe)
```

```
## # A tibble: 7 x 4
##   variable       type      role      source
##   <chr>          <list>    <chr>     <chr>
## 1 recclass       <chr [3]> predictor original
## 2 fall           <chr [3]> predictor original
## 3 year           <chr [2]> predictor original
## 4 reclat         <chr [2]> predictor original
## 5 reclong        <chr [2]> predictor original
## 6 recclass_group <chr [3]> predictor original
## 7 mass_g         <chr [2]> outcome   original
```

K-Fold Cross Validation

Let's create 10 folds to conduct K-Fold stratified cross validation. R will take the training data and assign each observation in the data from folds one to ten. A new testing set is created in each fold, and we will end up with ten folds. Then whenever we test a model, it will be fit to each fold's testing and training set, and come out with ten different accuracy scores. By using K-fold cross validation, we get a better estimate of testing accuracy then simply fitting the data to the entire training and testing sets. We can reduce variation by taking the mean accuracy from several samples than just one. We will also stratify on our outcome variable again, mass_g, to attempt to make sure the data is balanced.

```
k_folds <- vfold_cv(training_data, v = 10, strata = mass_g)
```

Modeling

Finally, we have arrived at the modeling step. Let's use Root Mean Squared Error (RMSE) as our primary metric. RMSE is one of the most commonly used metrics in machine learning, it measures the performance of regression based models by showing how far the model's predictions are from the true value. The lower the RMSE the better, as it represents a shorter distance between a model's predictions and the true value.

Fitting the Models

Most models have a similar process.

Step 1: Specify model parameters, engine, and mode

```
library(kknn)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-8
```

```
# Linear Regression
lm_model <- linear_reg() %>%
  set_engine("lm")

# Ridge Regression
```

```
ridge_spec <- linear_reg(mixture = 0,
                         penalty = tune()) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

# LASSO Regression
lasso_spec <- linear_reg(penalty = tune(),
                         mixture = 1) %>%
  set_mode("regression") %>%
  set_engine("glmnet")

# Elastic Net
elastic_spec <- linear_reg(penalty = tune(),
                           mixture = tune()) %>%
  set_mode("regression") %>%
  set_engine("glmnet")
```

Step 2: Set up workflow for model and add model and recipe

```
lm_workflow <- workflow() %>%
  add_model(lm_model) %>%
  add_recipe(data_recipe)

ridge_workflow <- workflow() %>%
  add_recipe(data_recipe) %>%
  add_model(ridge_spec)

lasso_workflow <- workflow() %>%
  add_recipe(data_recipe) %>%
  add_model(lasso_spec)

elastic_workflow <- workflow() %>%
  add_recipe(data_recipe) %>%
  add_model(elastic_spec)
```

Step 3: Create tuning grid to specify ranges of parameters

```
# Linear Regression needs no grid

# Ridge and LASSO grid
penalty_grid <- grid_regular(penalty(range = c(-5,5)), levels = 50)

# Elastic Net
elastic_grid <- grid_regular(penalty(range = c(-5, 5)), mixture(range = c(0,1)), levels = 10)
```

Step 4: Tune the model

```
# No tuning for Linear Regression

# Ridge
ridge_tune <- tune_grid(
  ridge_workflow,
  resamples = k_folds,
  grid = penalty_grid
)
```

RMSE for that tuned model is.

```
# Linear Regression
#lm_fit <- fit_resamples(lm_workflow, resamples = k_folds)
#lm_rmse <- collect_metrics(lm_fit) %>%
#  slice(1)

# Ridge
ridge_rmse <- collect_metrics(ridge_tune) %>%
  arrange(mean) %>%
  slice(49)

# LASSO
lasso_rmse <- collect_metrics(lasso_tune) %>%
  arrange(mean) %>%
  slice(35)
```

Results

Let's compare the results of our models and see which one performed the best.

```
library(tibble)

final_compare <- tibble(Model = c("Ridge Regression", "Lasso Regression"), RMSE = c(ridge_rmse$mean, las

final_compare <- final_compare %>%
  arrange(RMSE)

final_compare
```

```
## # A tibble: 2 x 2
##   Model              RMSE
##   <chr>             <dbl>
## 1 Lasso Regression 0.0227
## 2 Ridge Regression 0.0242
```

The model that performed the best was Lasso regression. Let's fit this best performing model to the testing set.

```
best_lasso <- select_best(lasso_tune, metric = 'rmse')
lasso_final_workflow <- finalize_workflow(lasso_workflow, best_lasso)
lasso_final_fit <- fit(lasso_final_workflow, data = training_data)
```

```
## New names:
## * `recclass_H` -> `recclass_H...119`
## * `recclass_H...4` -> `recclass_H...126`
## * `recclass_H.L4.5` -> `recclass_H.L4.5...138`
## * `recclass_H.L4.5` -> `recclass_H.L4.5...139`
## * `recclass_H3.4` -> `recclass_H3.4...149`
## * `recclass_H3.5` -> `recclass_H3.5...150`
## * `recclass_H3.6` -> `recclass_H3.6...151`
## * `recclass_H3.4` -> `recclass_H3.4...164`
## * `recclass_H3.5` -> `recclass_H3.5...167`
## * `recclass_H3.6` -> `recclass_H3.6...169`
## * `recclass_H3.8.4` -> `recclass_H3.8.4...176`
## * `recclass_H3.8.4` -> `recclass_H3.8.4...181`
```

```
## `recclass_C5.6.ung`, `recclass_CB`, `recclass_CH.CBb`, `recclass_CH3.`,
## `recclass_CK`, `recclass_CK3.8`, `recclass_CK3.4`, `recclass_CM`,
## `recclass_CM2.an`, `recclass_CO3.`, `recclass_CO3.1`, `recclass_CR.an`,
## `recclass_CR1`, `recclass_CR7`, `recclass_EH`, `recclass_EH3.4.an`,
## `recclass_EH6`, `recclass_EH7`, `recclass_EL.melt.rock`, `recclass_EL3.4`,
## `recclass_EL6.`, `recclass_EL6.7`, `recclass_Enst.achon`,
## `recclass_H.melt.breccia`, `recclass_H.L.3`, `recclass_H.L.4`,
## `recclass_H.L3.4`, `recclass_H.L3.6`, `recclass_H.L3.7`,
## `recclass_H.L4.5...138`, `recclass_H.L4.5...139`, `recclass_H.L5`,
## `recclass_H.L6.melt.rock`, `recclass_H3.`, `recclass_H3.05`, `recclass_H3.15`,
## `recclass_H3.2.3.7`, `recclass_H3.4.5`, `recclass_H4.melt.breccia`,
## `recclass_H5.an`, `recclass_H6.7`, `recclass_Howardite.an`,
## `recclass_Impact.melt.breccia`, `recclass_Iron..IAB.sHL.an`,
## `recclass_Iron..IID.an`, `recclass_Iron..IIE.`, `recclass_Iron.`,
## `recclass_L.H.3`, `recclass_L.LL..4`, `recclass_L.LL...261`,
## `recclass_L.LL.melt.rock`, `recclass_L.LL3.6`, `recclass_L.LL3.5...273`,
## `recclass_L.LL4.5`, `recclass_L.LL5.6...279`, `recclass_L.LL5.6...280`,
## `recclass_L.4.6`, `recclass_L3.4...289`, `recclass_L3.7...292`,
## `recclass_L3.melt.breccia`, `recclass_L3.0`, `recclass_L3.00`,
## `recclass_L3.4.3.7`, `recclass_L3.5.3.7`, `recclass_L3.8.5`, `recclass_L3.9.5`,
## `recclass_L4.melt.breccia`, `recclass_L4.melt.rock`, `recclass_L5.7`,
## `recclass_L5.melt.breccia`, `recclass_L6.melt.rock`, `recclass_LL.imp.melt`,
## `recclass_LL.melt.breccia`, `recclass_LL.melt.rock`, `recclass_LL.L.3.1`,
## `recclass_LL.3.5`, `recclass_LL.4`, `recclass_LL.4.5`, `recclass_LL3.4...364`,
## `recclass_LL3.05`, `recclass_LL3.1.3.5`, `recclass_LL3.10`, `recclass_LL3.7.6`,
## `recclass_LL3.8.4`, `recclass_LL3.9.4`, `recclass_LL3.4...386`,
## `recclass_LL4.6...391`, `recclass_LL6.an`, `recclass_LL6.melt.breccia`,
## `recclass_LL6.7`, `recclass_LL7...`, `recclass_Lunar..bas.gab.brec.`,
## `recclass_Lunar..norite.`, `recclass_Martian`,
## `recclass_Martian..basaltic.breccia.`, `recclass_Mesosiderite.A3`,
## `recclass_Mesosiderite.A3.4`, `recclass_OC3`, `recclass_R3.5`,
## `recclass_R3.4...449`, `recclass_R3.5.4`, `recclass_R3.7`, `recclass_R3.8.5`,
## `recclass_R3.9`, `recclass_R3.4...458`, `recclass_R4.5`, `recclass_R6`,
## `recclass_Relict.H`, `recclass_Relict.iron` and `recclass_Unknown`. Consider
## using `step_zv()` to remove those columns before normalizing
```

```
#m_tibble <- predict(lasso_final_fit, new_data = testing_data %>% select(-mass_g))
#m_tibble <- bind_cols(m_tibble, testing_data %>% select(mass_g))

#metric <- metric_set(rmse)

#tibble_metrics <- metric(m_tibble, truth = mass_g, estimate = .pred)
#tibble_metrics
```

Conclusion

In conclusion, after fitting various models, our best model to predict the mass of a meteorite is Lasso regression. Lasso stands for least absolute shrinkage and selection operator. It differs from a normal linear regression by eliminating less important variables from the model with a regularization term. Therefore, it makes sense that Lasso would be our best model because as we observed from the correlation plot earlier, many of the numeric variables such as year, latitude, and longitude have very little correlation with mass, and thus would likely be eliminated or minimized by a lasso regression. The regression would then focus on a much more highly correlated variable, like recclass, thus resulting in a decent RMSE score of around 0.022.

Going back to our hypothesis, it seems that mass and the location a meteorite lands are not very correlated. Instead, the best predictor to predict a meteorite is it's type/composition.

There are many avenues of improvement for this project, such as testing more models like K-nearest neighbors, random forest, etc. Also, we can use a dataset with more predictor variables, as other factors of meteorites might be better predictor variables for mass than year and location.

Overall, attempeting to predict the mass of meteorites using this NASA dataset has greatly improved my machine learning and modeling skills.