# The Statistics of Pokemon

null

null

## Contents

## Introduction

Pokemon is an RPG (role-playing game) developed by GameFreak. The main video game series centers around the concept of "Pokemon Battles," where titular Pokemon–fantastical creatures that display a wide range of characteristics–owned by "Pokemon Trainers" engage in turn-based battles until all Pokemon available to one side "faint." As the franchise has developed, the range of Pokemon available in-game has expanded greatly. As of the current generation of Pokemon games, there are 807 canon Pokemon creatures, each with unique characteristics, which are outlined in a canonical Pokedex, present in the games and recorded in our API of choice. Each Pokemon has a wealth of associated data, some of which we outline in the following section. For our final project for S&DS361, we decided that these data may be interesting to analyze. In particular, with the rapid rise of Pokemon in popular culture, such analysis may be engaging for statisticians and non-statisticians alike.

### The data

Pokemon have myriad characteristics associated with them. Below are the most well-characterized traits, which we decided to focus our analyses on:

- Type: Each Pokemon is a member of one or two Pokemon "types", corresponding to elements or concepts (i.e. water, grass, ghost, normal). Each type is weak to certain types, and effective against other types, resulting in a rock-paper-scissors-like game dynamic.
- Height: Each Pokemon has a height associated with it in the Pokedex.
- Weight: Each Pokemon has a weight associated with it in the Pokedex.
- Base Experience: When defeated, each Pokemon yields some amount of experience.
- Base Stats: Pokemon have attack/defense (normal and special) and health points (HP) stats that determine their behavior in battle. Attack/defense values also each have an associated *effort value* that determines how experience points are rewarded when battling this Pokemon. Effort values are usually distributed such that the strongest stats have the highest effort points.
- Legendary status: a small number of Pokemon are *legendary*, which are unusually powerful and rare and associated with the lore of the game.

### API

To obtain these data, we made use of the PokeAPI, a free-to-use API containing information from all generations of the Pokemon games, complete with wrappers in numerous languages. However, an official R wrapper is not available, and while a wrapper package exists, we made direct API calls to retrieve our data.

## Data acquisition

The downloading of the data was done through LoadJSON requests from the PokeAPI. We first made a call which listed all the Pokemon in the database. Using this list of all Pokemon, we then made a separate request for each Pokemon to obtain more detailed information, receiving data about the various characteristics, such as type, height, weight, growth rate, base experience, base stats, and legendary status. The JSON files provided by the API were then processed and saved into a DataFrame for subsequent analyses. After the completion of this step, the processed dataframe (attached as supplementary data) was imported for all analyses.

## Visualization

### Load CSV

We processed the JSON files returned by the API into a single DataFrame, where each row denotes a Pokemon and columns correspond to the various attributes. For brevity, we have excluded these preprocessing steps from the report.

```r
library(GGally)
library(ggplot2)
library(RColorBrewer)
require(plyr)
library(leaps)
library(knitr)
```

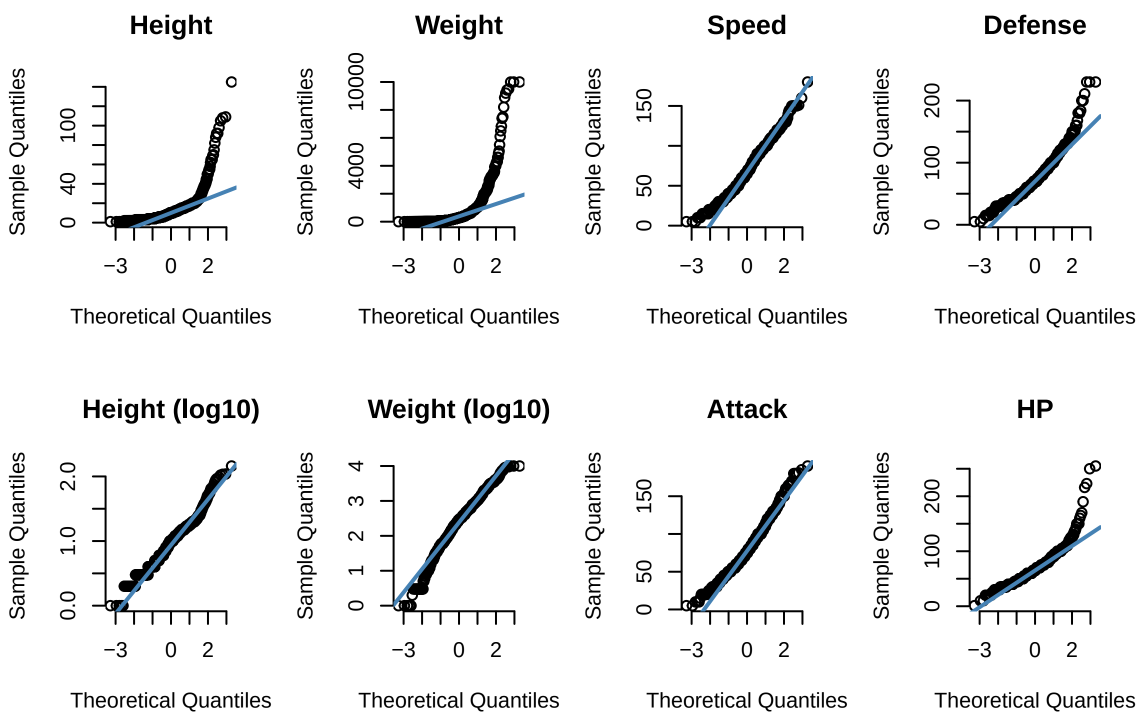
```r
pokemons_df <- read.csv("pokemons.csv",stringsAsFactors=FALSE)
pokemons_df$legendary <- mapvalues(pokemons_df$is_legendary,
          from=c(TRUE,FALSE),
          to=c("Legendary","Normal"))

# calculate log-transformed heights and weights
```

```r
pokemons_df$log_height <- log10(pokemons_df$height)
pokemons_df$log_weight <- log10(pokemons_df$weight)
```

## Q-Q plots of each characteristic

To see examine the normality of each statistic, we used q-q plots to plot the theoretical normal quantiles against the sample quantiles. We see that most of the traits are more or less normally-distributed (and height and weight after a log-transformation). However, we also see that there is a skew towards Pokemon with high HP and defense.

```r
# define subplots
par(mfrow=c(2,4))

qqnorm(pokemons_df$height, pch = 1, frame = FALSE,main="Height")
qqline(pokemons_df$height, col = "steelblue", lwd = 2)

qqnorm(pokemons_df$weight, pch = 1, frame = FALSE,main="Weight")
qqline(pokemons_df$weight, col = "steelblue", lwd = 2)

qqnorm(pokemons_df$speed_base, pch = 1, frame = FALSE,main="Speed")
qqline(pokemons_df$speed_base, col = "steelblue", lwd = 2)

qqnorm(pokemons_df$defense_base, pch = 1, frame = FALSE,main="Defense")
qqline(pokemons_df$defense_base, col = "steelblue", lwd = 2)

qqnorm(pokemons_df$log_height, pch = 1, frame = FALSE,main="Height (log10)")
qqline(pokemons_df$log_height, col = "steelblue", lwd = 2)

qqnorm(pokemons_df$log_weight, pch = 1, frame = FALSE,main="Weight (log10)")
qqline(pokemons_df$log_weight, col = "steelblue", lwd = 2)

qqnorm(pokemons_df$attack_base, pch = 1, frame = FALSE,main="Attack")
qqline(pokemons_df$attack_base, col = "steelblue", lwd = 2)

qqnorm(pokemons_df$hp_base, pch = 1, frame = FALSE,main="HP")
qqline(pokemons_df$hp_base, col = "steelblue", lwd = 2)
```



## Type-stratified visualization

In general, a Pokemon's type determines its relative strength against another in battle. For instance, fire-type Pokemon tend to perform well against ice types, but are vulnerable to attacks by water types. Pokemon actually have up to two types, but for the purpose of this report, we will consider only the primary one.

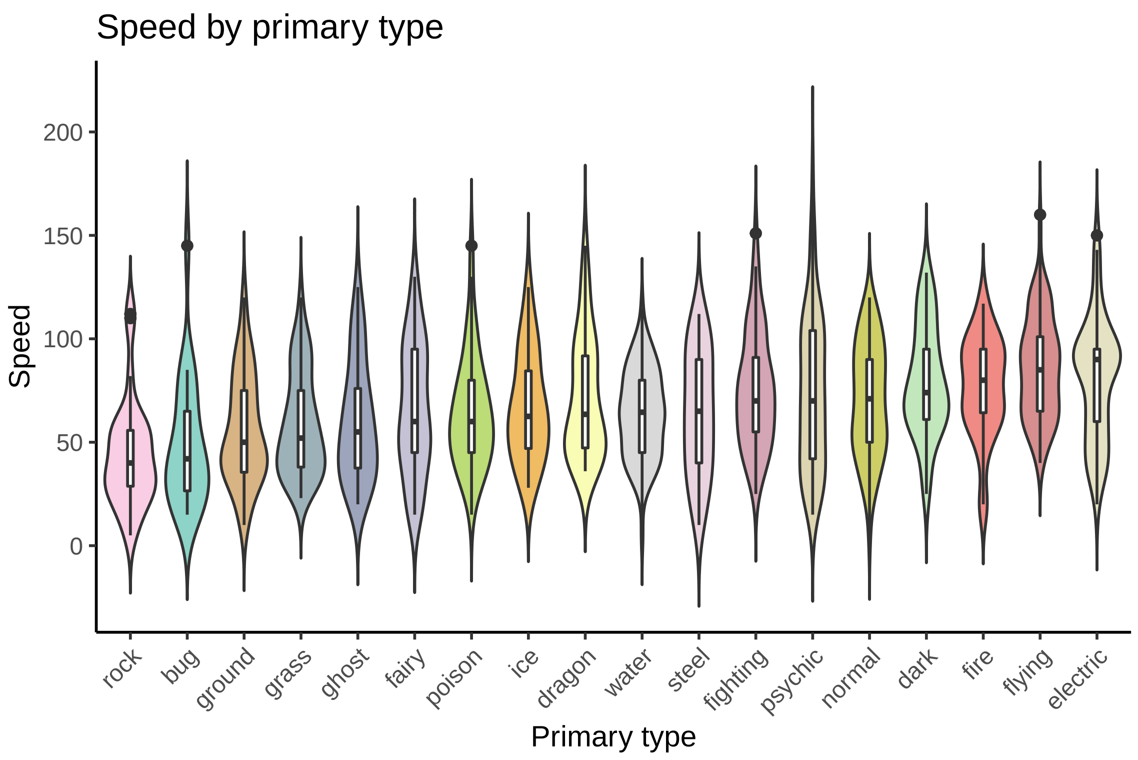
Here, we examine the distributions of speed and defense (for brevity) with respect to type, first by defining a helper function for generating the plots by each type.

```r
plot_by_primary <- function(var,var_name){

  # make the plot by type
  colourCount = length(unique(pokemons_df$type_primary))
  getPalette = colorRampPalette(brewer.pal(9, "Set3"))

  # order types by medians for variable
  bymedian <- reorder(pokemons_df$type_primary, pokemons_df[,var], median)

  plt <- ggplot(pokemons_df, aes_string(x=bymedian, y=var, fill="type_primary")) +
    geom_violin(trim=FALSE)+
    geom_boxplot(width=0.1, fill="white")+
    labs(title=paste(var_name,"by primary type"),x="Primary type", y = var_name) +
    scale_fill_manual(values = getPalette(colourCount)) +
    theme_classic() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1),legend.position = "none")

  return(plt)
}
```

**Speed**

With regards to speed, we see some differences that we would expect by common sense - electric and flying types have the highest median speed, and rock, bug, and ground types have the lowest.

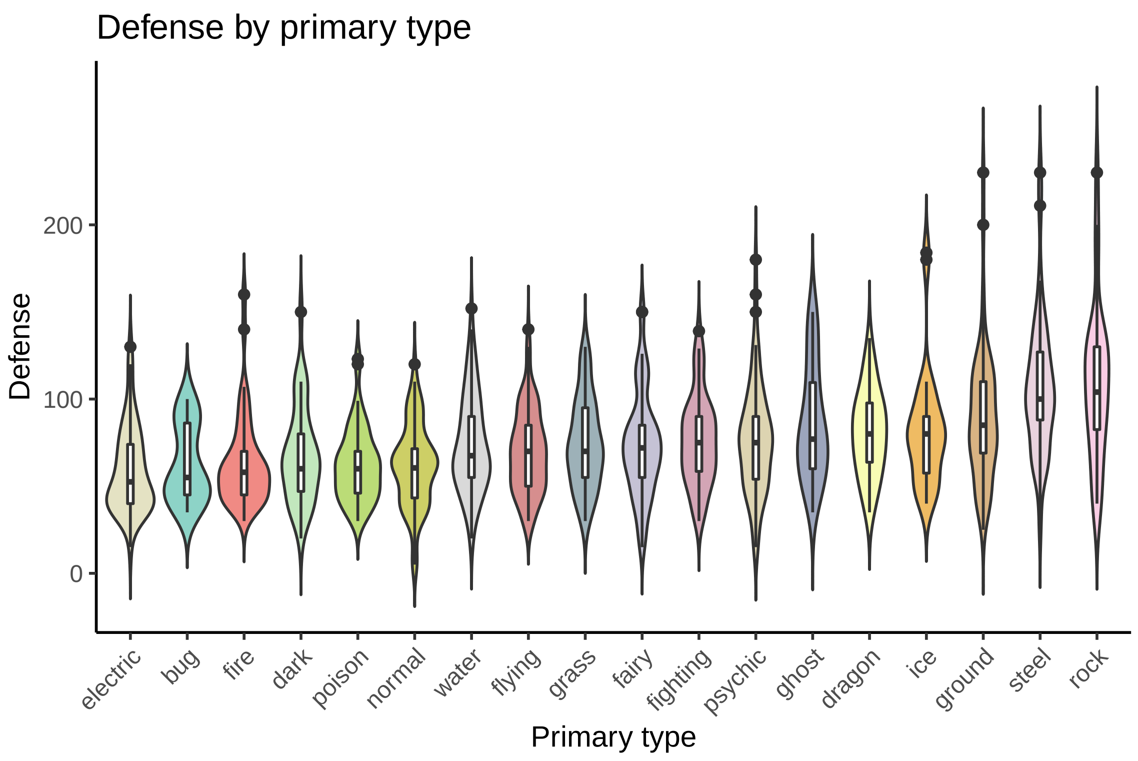
```r
plot_by_primary("speed_base","Speed")
```



**Base defense**

The distributions of defense by primary type seem to have a few types with high medians, namely ground, steel, and rock.

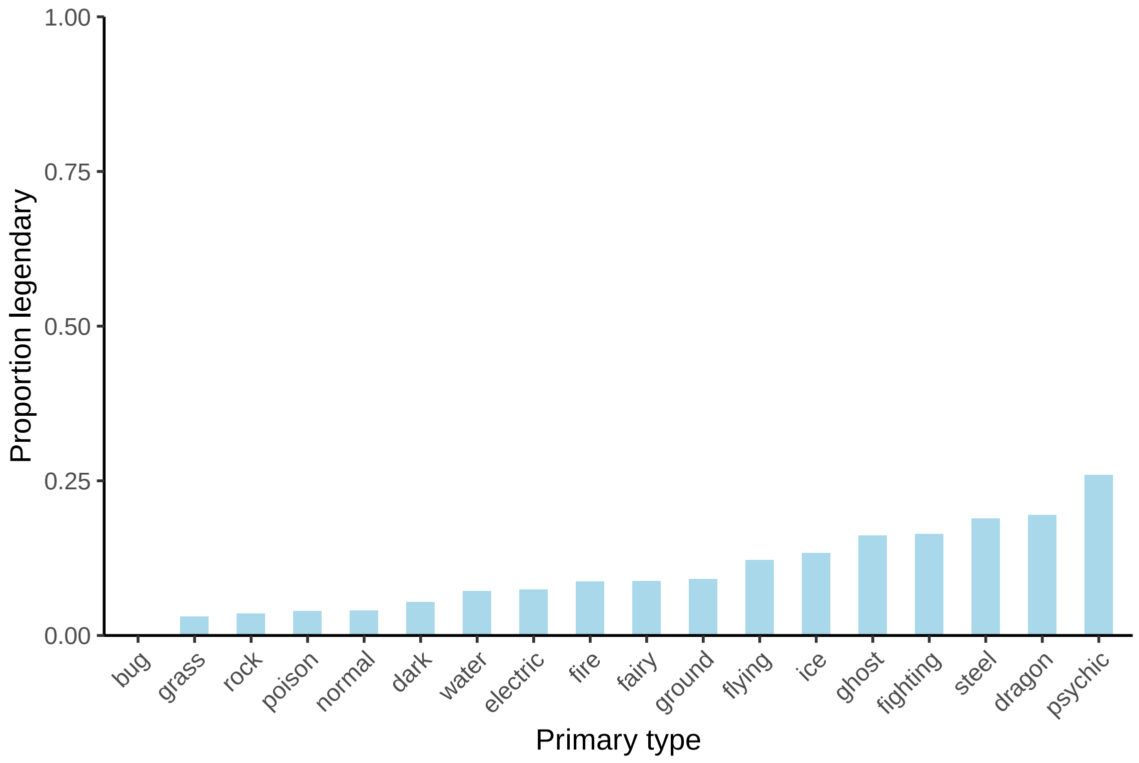
```r
plot_by_primary("defense_base","Defense")
```



**Proportions of legendary pokemon by type**

Using the primary types, we can also take a preliminary look at the proportions of Pokemon within each type that are legendary, giving us a suggestion as to if type could be indicative of legendary status. We indeed see that some types (e.g. steel, dragon, and psychic) have especially high proportions of legendaries, whereas others (bug, grass, and rock) have none or very little. This is consistent with the Pokemon lore, which often include Pokemon possessing mythical attributes or abilities (dragons, psychic abilities, etc) as legendary Pokemon.

```r
bymean <- reorder(pokemons_df$type_primary, pokemons_df$is_legendary, mean)
order <- factor(pokemons_df$legendary,levels=c("Normal","Legendary"))

ggplot(pokemons_df,aes(x = bymean,fill = order)) +
    geom_bar(position = "fill",width=0.5) +
    scale_fill_manual(values=c("white", "#a8d8ea")) +
    theme_classic() +
    ylim(0,1)+
    scale_y_continuous(expand = c(0,0)) +
    xlab("Primary type") +
    ylab("Proportion legendary") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1),
          legend.position = "none")
```

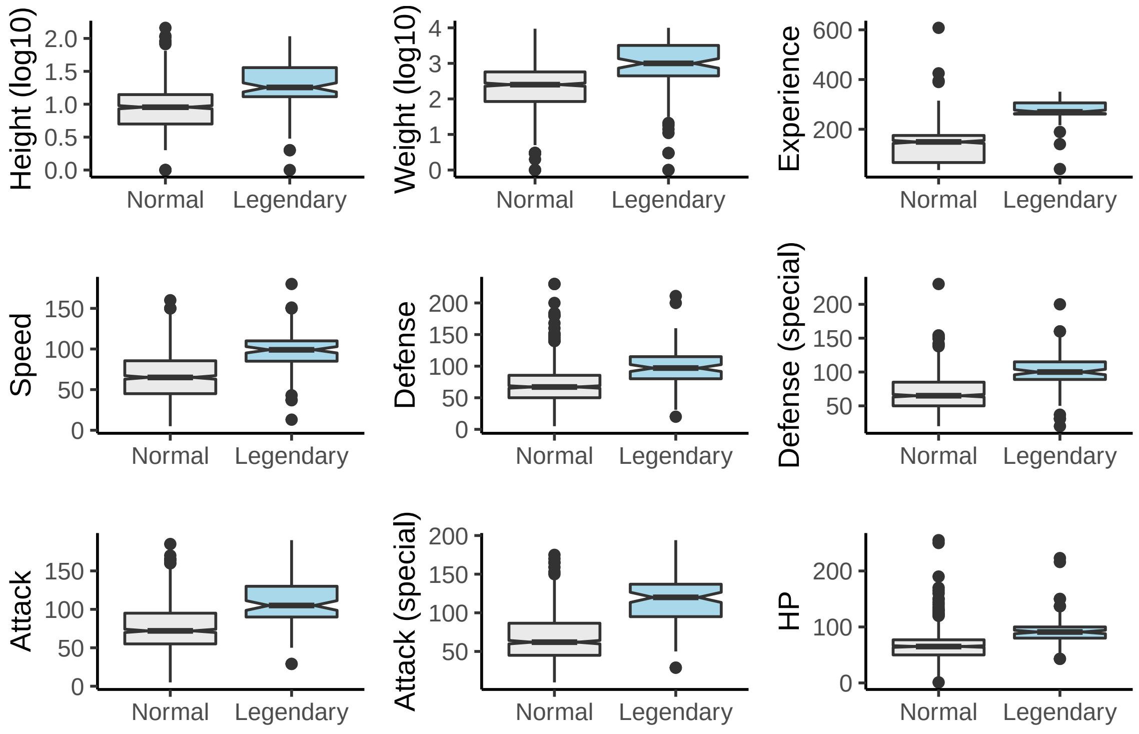


## Comparisons of mormal and legendary Pokemon statistics

Legendary-type Pokemon are generally stronger than normal ones in battle. Here, we take a look at these differences in the performance indicators with respect to legendary status.

As one would expect given the reputation of legendary Pokemon as being especially powerful, legendaries tend to have higher stats in every single statistic, weight and height included.

```r
library(gridExtra)

# plots
p <- list()

select_vars = c("log_height",
                "log_weight",
                "base_experience",
                "speed_base",
                "defense_base",
                "special.defense_base",
                "attack_base",
                "special.attack_base",
                "hp_base")

select_names = c("Height (log10)",
                 "Weight (log10)",
                 "Experience",
                 "Speed",
                 "Defense",
                 "Defense (special)",
                  "Attack",
                  "Attack (special)",
                  "HP")

order <- factor(pokemons_df$legendary,levels=c("Normal","Legendary"))

for (i in 1:length(select_vars)) {

  var <- select_vars[i]
  name <- select_names[i]

  p[[i]] <- ggplot(data=pokemons_df, aes_string(x=order, y=var)) +
            geom_boxplot(aes(fill=order),notch=TRUE) + guides(fill=FALSE) +
            theme(axis.title.y = element_text(size=14)) +
            xlab("") + ylab(name) +
            theme_classic() +
            scale_fill_manual(values=c("#eaeaea", "#a8d8ea"))
}

# arrange the boxplots in a grid
plots <- do.call(grid.arrange, c(p, ncol=3))
```



## Statistical analysis

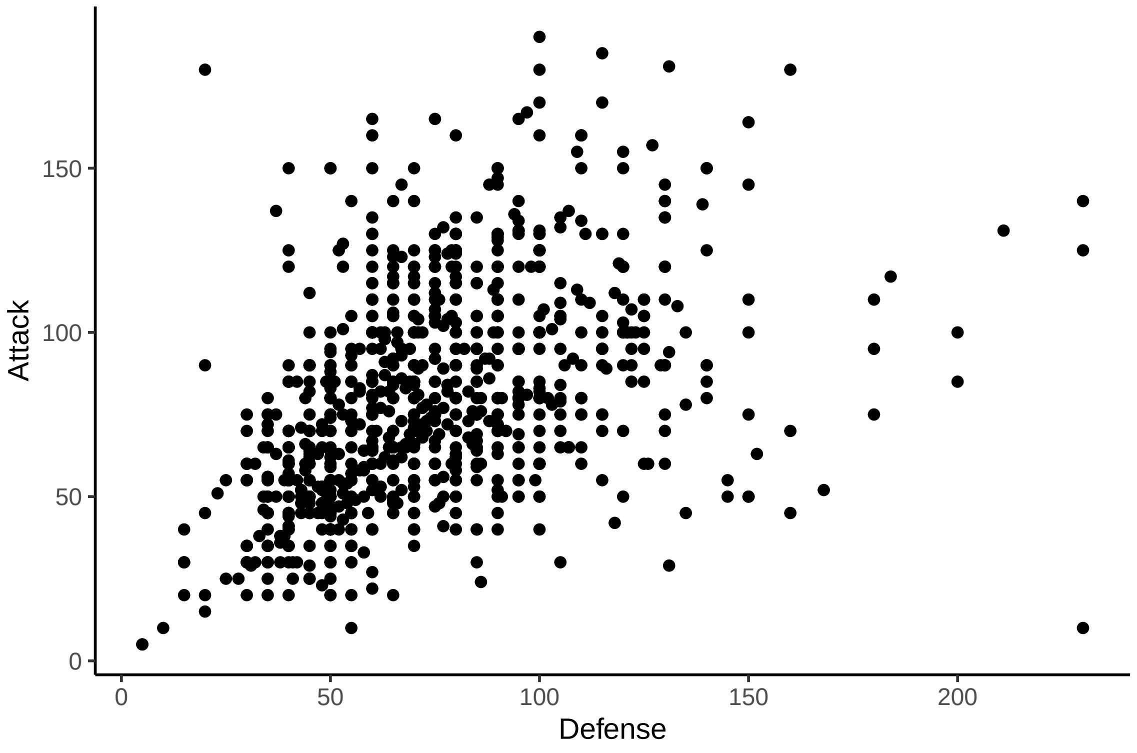
### Bootstrap analysis

We would expect certain traits to be correlated with each other - for instance, stronger Pokemon tend to have a stronger defense and attack, and taller Pokemon tend to weigh more as well. Here we use boostrap tests to examine the significance of these correlations.

### Defense and attack

#### Plot

```
ggplot(pokemons_df, aes(x = defense_base, y = attack_base))+
  geom_point() +
  theme_classic() +
  labs(x="Defense", y = "Attack")
```



From a brief view at this data, it seems that there is some positive relationship between defense and attack (which would be consistent with Pokemon gameplay, where more valuable and powerful Pokemon often have both higher attack and defense).

```
corr <- cor(pokemons_df$defense_base, pokemons_df$attack_base)
corr
```

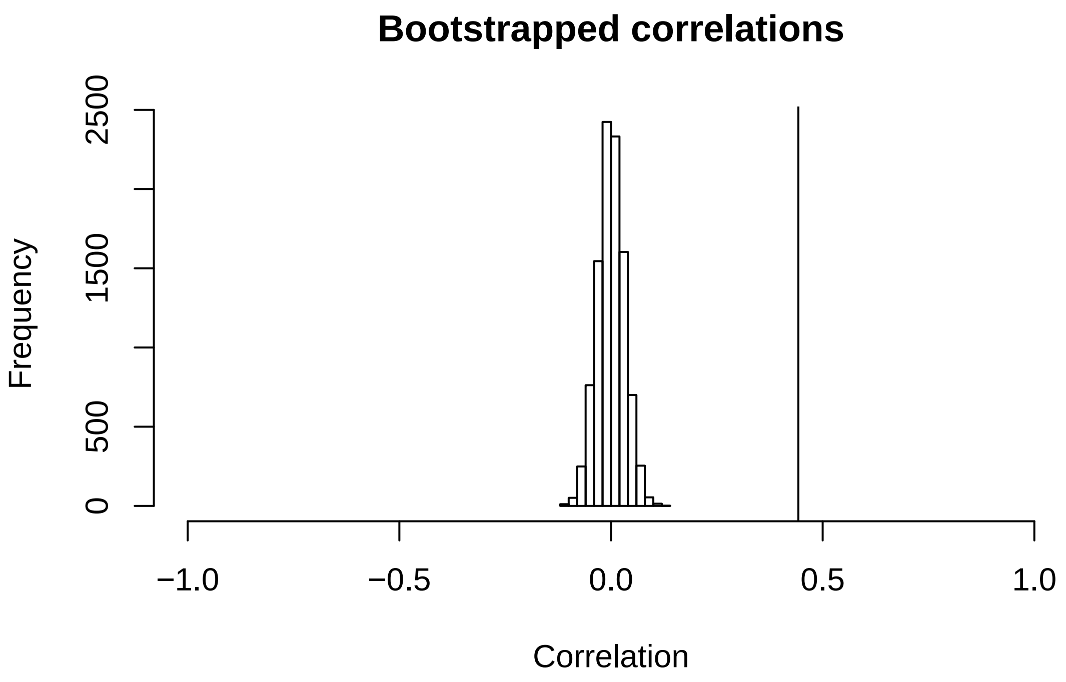
```
## [1] 0.4425994
```

As expected, there is correlation observed between defense and attack. However, using bootstrap analysis, we can perform hypothesis testing against a null hypothesis where there is no correlation between the two, and perform a more rigorous analysis.

#### Testing significance with bootstrap

```
n <- 10000

# keep track of the bootstrapped correlations
corr_bs <- rep(0, n)

for (i in 1:n){
  # sample the bootstrap
  defense_sample <- sample(pokemons_df$defense_base,
                           length(pokemons_df$defense_base),
                           replace = T)
  attack_sample <- sample(pokemons_df$attack_base,
                          length(pokemons_df$attack_base),
                          replace = T)

  corr_bs[i] <- cor(defense_sample, attack_sample)
}
```

```
hist(corr_bs, xlim=c(-1,1), main="Bootstrapped correlations",xlab="Correlation")
abline(v = corr)
```



```
mean(as.numeric(corr_bs > corr))
```

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

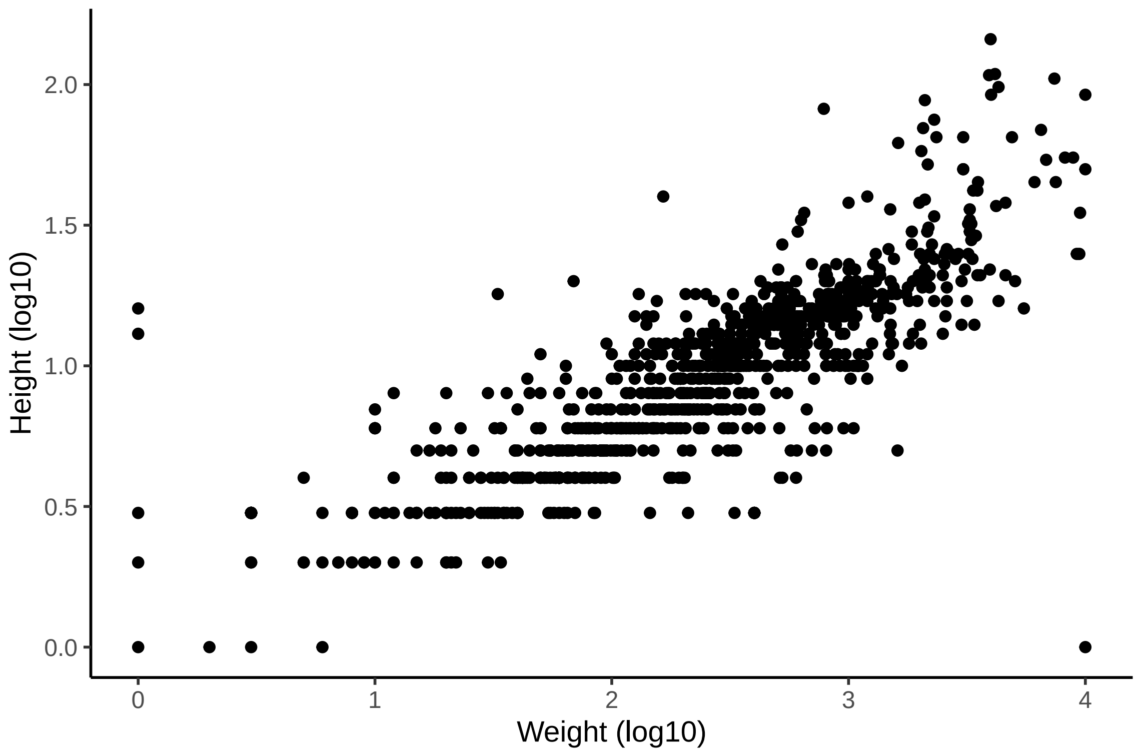
From this result, we see that there were no bootstrapped samples with a correlation greater than the observed correlation in the null distribution, corresponding to a P-value of 0. Thus, we reject the null, and there seems to be a statistically significant correlation between attack and defense.

### Height and weight

#### Plot

We then consider height and weight (log-transformed). We see a particularly strong relationship between these two variables, which makes sense as taller Pokemon would be expected to also weigh more.

```
ggplot(pokemons_df, aes(x = log_weight, y = log_height))+
  geom_point() +
  theme_classic() +
  labs(x="Weight (log10)", y = "Height (log10)")
```



Indeed, there is high correlation observed between weight and height.

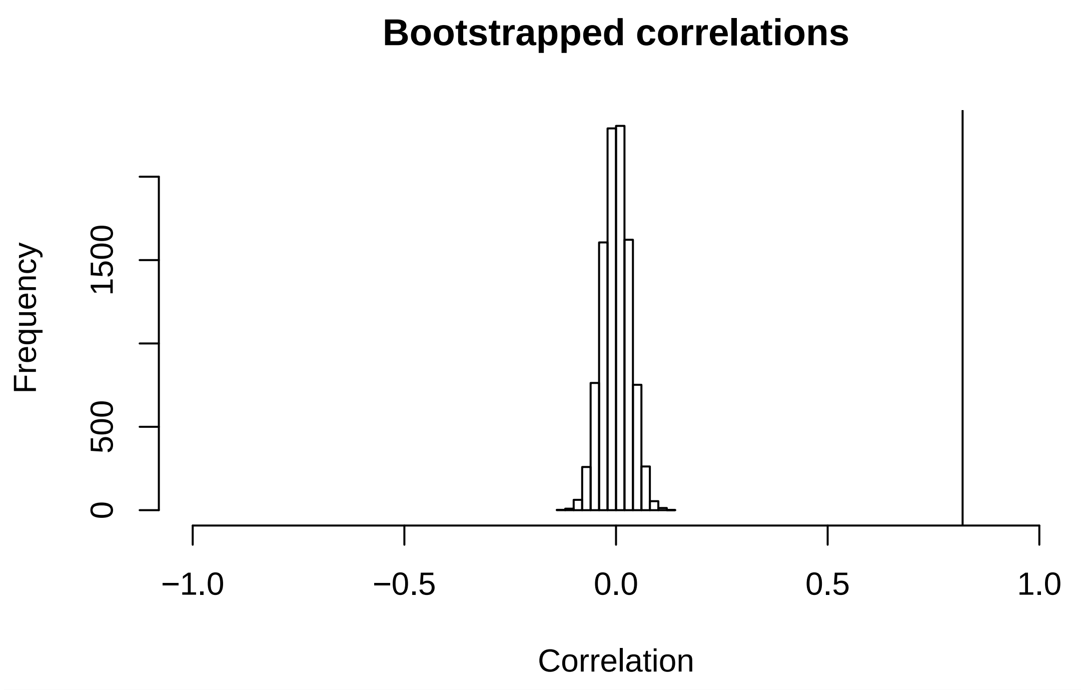
```
corr <- cor(pokemons_df$log_weight, pokemons_df$log_height)
corr
```

```
## [1] 0.818774
```

#### Testing significance with bootstrap

```
n <- 10000

# keep track of the bootstrapped correlations
corr_bs <- rep(0, n)
for (i in 1:n){

  # sample the bootstrap
  weight_sample <- sample(pokemons_df$log_weight,
                          length(pokemons_df$log_weight),
                          replace = T)
  height_sample <- sample(pokemons_df$log_height,
                          length(pokemons_df$log_height),
                          replace = T)

  corr_bs[i] <- cor(weight_sample, height_sample)
}
```

```
hist(corr_bs, xlim=c(-1,1), main="Bootstrapped correlations",xlab="Correlation")
abline(v = corr)
```



```
mean(as.numeric(corr_bs > corr))
```

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

Again, we found no bootstrapped samples with a correlation above the observed correlation in the null distribution. Thus, we reject the null hypothesis that there is no correlation between weight and height.

## Principal Component Analysis

To gain an understanding of the overall landscape of Pokemon, we used PCA to examine the major directions of variation among numeric factors. For PCA, we only used the log10-transformed heights and weights, as we found the raw ones to dominate the projections.

```
pca_df <- pokemons_df[,-c(1,4,5)] # remove indices and raw heights and weights

# keep only the continuous variables
num_only <- unlist(lapply(pca_df, is.numeric))
pca_df <- pca_df[ , num_only]

# compute the PCA
pc1 <- prcomp(pca_df)
```

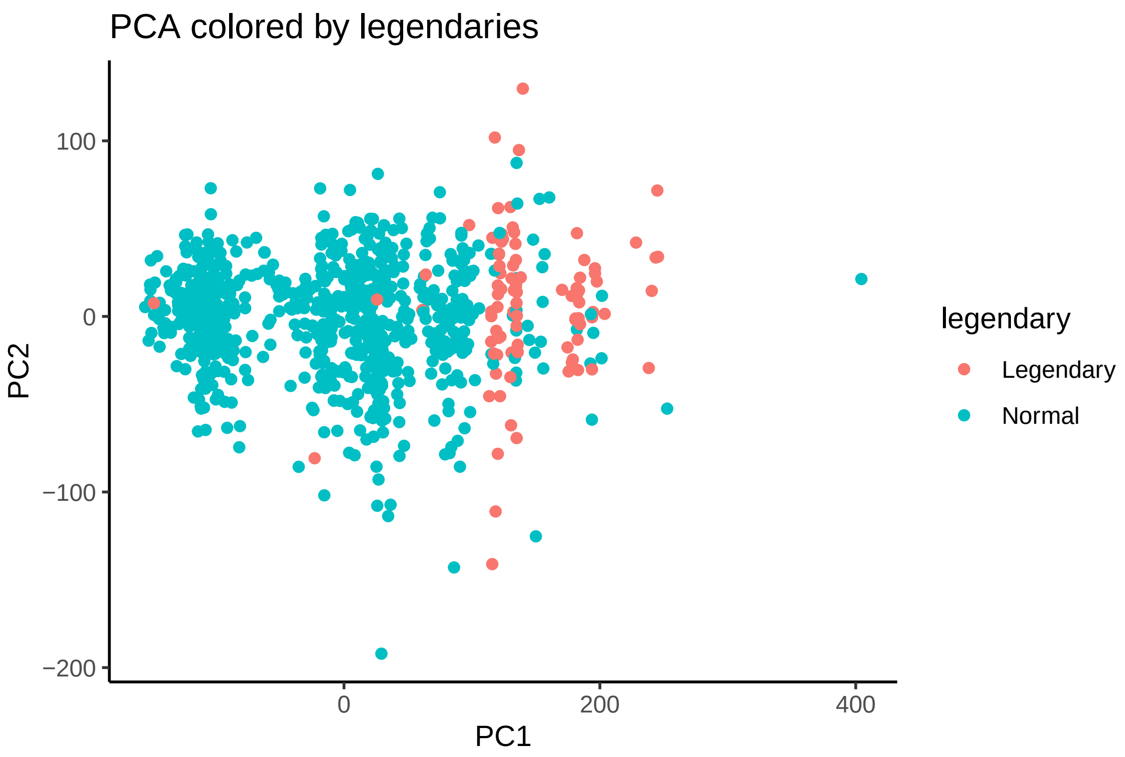
```
summary(pc1)
```

```
## Importance of components:
##                           PC1      PC2     PC3      PC4      PC5      PC6
## Standard deviation     90.969 32.57219 27.0600 23.44183 20.96206 15.39053
## Proportion of Variance  0.721  0.09244  0.0638  0.04788  0.03828  0.02064
## Cumulative Proportion   0.721  0.81344  0.8772  0.92511  0.96340  0.98403
##                             PC7     PC8     PC9    PC10    PC11    PC12    PC13
## Standard deviation     13.45724 0.70249 0.67863 0.61435 0.55004 0.51808 0.46117
## Proportion of Variance  0.01578 0.00004 0.00004 0.00003 0.00003 0.00002 0.00002
## Cumulative Proportion   0.99981 0.99986 0.99990 0.99993 0.99995 0.99998 1.00000
##                           PC14    PC15
## Standard deviation     0.1546 0.1251
## Proportion of Variance 0.0000 0.0000
## Cumulative Proportion  1.0000 1.0000
```

It seems that principal component 1 contains most (about 72%) of the observed variance, after which the remaining components taper off.

When we plot the first two components, we see that there is a clear clustering effect in the first component. Some of these clusters appear to be accounted for by legendary Pokemon, but the remaining ones appear to be backed by a different factor.

```r
pokemons_df['PC1'] = pc1$x[,1]
pokemons_df['PC2'] = pc1$x[,2]

ggplot(data = pokemons_df, aes_string(x='PC1', y='PC2')) +
    geom_point(aes(color = legendary)) +
    guides(fill = guide_legend(ncol = 2)) +
    labs(title= 'PCA colored by legendaries') +
    theme_classic()
```



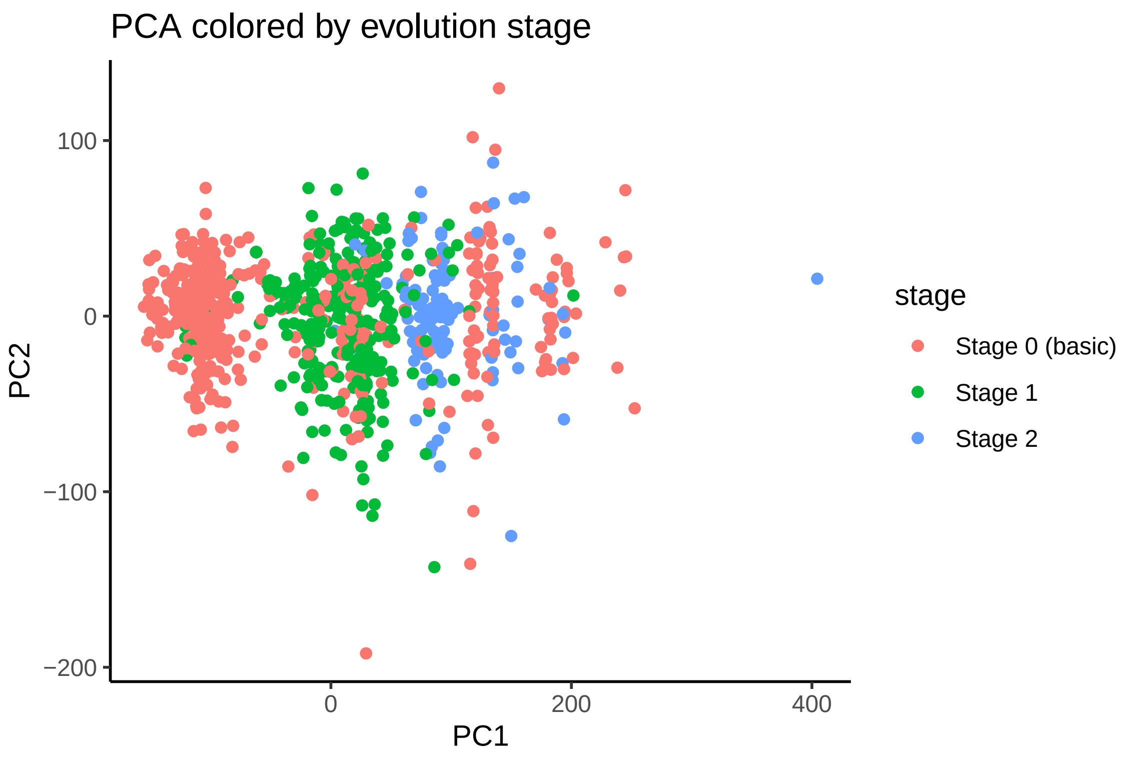
We hypothesized that these clusters may in fact reflect the evolution mechanic in Pokemon. Pokemon have different levels of evolution in which they evolve into progressively stronger species - for instance, a Psyduck may evolve into a Golduck. The maximum number of evolutions is two, so we can assign each Pokemon three stages of evolution: basic (stage 0), stage 1, and stage 2. Evolution stages are not directly provided by the PokeAPI as part of Pokemon info, so we instead scraped the bulbapedia site (bulbapedia.bulbagarden.net).

```r
# read in Pokemon with each stage
basic <- read.table("basic.txt",header=FALSE,stringsAsFactors=FALSE)$V1
stage_1 <- read.table("stage_1.txt",header=FALSE,stringsAsFactors=FALSE)$V1
stage_2 <- read.table("stage_2.txt",header=FALSE,stringsAsFactors=FALSE)$V1

# make stages dataframe
```

```r
stages <- data.frame(species=tolower(c(basic,stage_1,stage_2)))
stages$stage <- c(rep("Stage 0 (basic)",length(basic)),
                  rep("Stage 1",length(stage_1)),
                  rep("Stage 2",length(stage_2))
                  )

stages <- stages[!duplicated(stages$species),]

rownames(stages) <- stages$species

pokemons_df$stage <- stages[pokemons_df$species,"stage"]
```

When colored by evolution stage, we see that many of these clusters are clearly distinguished by evolution stage. Moreover, because legendary Pokemon cannot evolve (with a handful of exceptions), these can be seen as exceptions to the rest of the "basic" cluster.

```r
pokemons_df['PC1'] = pc1$x[,1]
pokemons_df['PC2'] = pc1$x[,2]

ggplot(data = pokemons_df, aes_string(x='PC1', y='PC2')) +
    geom_point(aes(color = stage)) +
    guides(fill = guide_legend(ncol = 2)) +
    labs(title= 'PCA colored by evolution stage') +
    theme_classic()
```



## Linear regression: predicting base health

Here, we used linear regression to predict the base HP of each Pokemon.

### Naive full regression

```r
nfr_hp <- lm(hp_base ~ . -X -name -species, pokemons_df)
```

```r
summary(nfr_hp)
```

```
##
## Call:
## lm(formula = hp_base ~ . - X - name - species, data = pokemons_df)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.452e-11 -7.710e-14 -7.000e-16  9.520e-14  9.666e-13
##
## Coefficients: (2 not defined because of singularities)
##                          Estimate Std. Error    t value Pr(>|t|)
## (Intercept)             1.211e+03  2.874e-12  4.211e+14  < 2e-16 ***
## base_experience        -4.616e+00  1.231e-14 -3.749e+14  < 2e-16 ***
## height                  5.360e-15  5.291e-15  1.013e+00  0.31152
## weight                 -2.309e-17  4.276e-17 -5.400e-01  0.58948
## type_primarydark       -6.911e-14  3.317e-13 -2.080e-01  0.83505
## type_primarydragon     -2.798e-14  3.434e-13 -8.100e-02  0.93509
## type_primaryelectric   -7.205e-14  3.691e-13 -1.950e-01  0.84531
## type_primaryfairy      -5.840e-14  3.277e-13 -1.780e-01  0.85865
## type_primaryfighting   -2.263e-14  3.305e-13 -6.800e-02  0.94543
## type_primaryfire       -4.908e-14  3.625e-13 -1.350e-01  0.89236
## type_primaryflying     -1.268e-15  3.113e-13 -4.000e-03  0.99675
## type_primaryghost      -2.180e-14  3.404e-13 -6.400e-02  0.94897
## type_primarygrass      -1.045e-13  3.339e-13 -3.130e-01  0.75449
## type_primaryground     -5.256e-14  3.216e-13 -1.630e-01  0.87025
## type_primaryice         9.405e-14  3.548e-13  2.650e-01  0.79103
## type_primarynormal      1.740e-13  4.146e-13  4.200e-01  0.67483
## type_primarypoison     -4.785e-13  3.292e-13 -1.453e+00  0.14678
## type_primarypsychic     8.726e-14  3.272e-13  2.670e-01  0.78984
## type_primaryrock       -1.109e-13  3.668e-13 -3.020e-01  0.76262
## type_primarysteel      -8.074e-15  3.287e-13 -2.500e-02  0.98041
## type_primarywater      -3.054e-14  3.442e-13 -8.900e-02  0.92934
## type_secondarydark     -1.280e-13  1.822e-13 -7.020e-01  0.48275
## type_secondarydragon   -1.402e-13  1.900e-13 -7.380e-01  0.46112
## type_secondaryelectric -1.317e-13  1.984e-13 -6.640e-01  0.50705
## type_secondaryfairy    -2.732e-13  5.228e-13 -5.230e-01  0.60149
## type_secondaryfighting -1.539e-13  2.638e-13 -5.830e-01  0.56005
## type_secondaryfire     -1.525e-13  1.735e-13 -8.790e-01  0.38002
## type_secondaryflying    5.947e-14  5.383e-13  1.100e-01  0.91208
## type_secondaryghost     1.475e-14  1.732e-13  8.500e-02  0.93217
## type_secondarygrass    -4.106e-13  1.481e-13 -2.773e+00  0.00578 **
## type_secondaryground   -9.433e-14  1.900e-13 -4.960e-01  0.61984
## type_secondaryice      -7.903e-14  2.199e-13 -3.590e-01  0.71946
## type_secondarynormal   -1.480e-13  1.528e-13 -9.680e-01  0.33338
## type_secondarypoison   -4.400e-14  1.924e-13 -2.290e-01  0.81920
## type_secondarypsychic  -8.475e-14  1.835e-13 -4.620e-01  0.64446
## type_secondaryrock     -3.842e-14  1.456e-13 -2.640e-01  0.79200
## type_secondarysteel    -1.269e-13  1.964e-13 -6.460e-01  0.51866
## type_secondarywater    -1.230e-13  1.408e-13 -8.730e-01  0.38297
## speed_base              -9.060e-01  2.685e-15 -3.374e+14  < 2e-16 ***
## speed_effort             5.134e-04  1.724e-13  2.979e+09  < 2e-16 ***
## special.defense_base    -1.145e+00  3.329e-15 -3.439e+14  < 2e-16 ***
## special.defense_effort -5.069e-03  1.689e-13 -3.001e+10  < 2e-16 ***
## special.attack_base     -1.434e+00  3.941e-15 -3.639e+14  < 2e-16 ***
## special.attack_effort  -1.621e-02  1.663e-13 -9.751e+10  < 2e-16 ***
## defense_base            -1.028e+00  3.115e-15 -3.301e+14  < 2e-16 ***
## defense_effort          -1.499e-03  1.748e-13 -8.579e+09  < 2e-16 ***
## attack_base             -1.291e+00  3.633e-15 -3.554e+14  < 2e-16 ***
## attack_effort           -1.186e-02  1.686e-13 -7.036e+10  < 2e-16 ***
## hp_effort               -5.324e-03  1.712e-13 -3.109e+10  < 2e-16 ***
## is_legendaryTRUE        -7.465e-14  2.025e-13 -3.690e-01  0.71259
## legendaryNormal                 NA         NA         NA       NA
## log_height              -1.271e-02  2.952e-13 -4.306e+10  < 2e-16 ***
## log_weight              -2.234e-02  9.743e-14 -2.293e+11  < 2e-16 ***
## PC1                      5.406e+00  1.317e-14  4.106e+14  < 2e-16 ***
## PC2                             NA         NA         NA       NA
## stageStage 1             9.647e-14  1.106e-13  8.720e-01  0.38341
## stageStage 2             4.695e-14  1.978e-13  2.370e-01  0.81243
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.019e-13 on 465 degrees of freedom
##   (444 observations deleted due to missingness)
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 1.147e+28 on 54 and 465 DF,  p-value: < 2.2e-16
```

From our naive regression, it seems that certain statistics (namely base experience, base speed, and weight) are significant coefficients. It also seems that we may drop the type factors, as these do not seem to be as strong indicators.

```r
hp1 <- lm(hp_base ~ . -X -type_primary -type_secondary -name -species, pokemons_df)
```

```r
summary(hp1)
```

```
##
## Call:
## lm(formula = hp_base ~ . - X - type_primary - type_secondary -
##     name - species, data = pokemons_df)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.516e-11 -2.990e-14  3.220e-14  8.770e-14  7.064e-13
##
## Coefficients: (2 not defined because of singularities)
##                          Estimate Std. Error    t value Pr(>|t|)
## (Intercept)              1.211e+03  2.641e-12  4.583e+14   <2e-16 ***
## base_experience         -4.616e+00  1.145e-14 -4.031e+14   <2e-16 ***
## height                   4.401e-15  5.001e-15  8.800e-01    0.379
## weight                  -6.881e-18  3.990e-17 -1.720e-01    0.863
## speed_base              -9.060e-01  2.479e-15 -3.654e+14   <2e-16 ***
## speed_effort             5.134e-04  1.591e-13  3.228e+09   <2e-16 ***
## special.defense_base    -1.145e+00  3.075e-15 -3.723e+14   <2e-16 ***
## special.defense_effort -5.069e-03  1.559e-13 -3.251e+10   <2e-16 ***
## special.attack_base     -1.434e+00  3.644e-15 -3.935e+14   <2e-16 ***
## special.attack_effort  -1.621e-02  1.543e-13 -1.051e+11   <2e-16 ***
## defense_base           -1.028e+00  2.757e-15 -3.729e+14   <2e-16 ***
## defense_effort         -1.499e-03  1.615e-13 -9.281e+09   <2e-16 ***
## attack_base            -1.291e+00  3.367e-15 -3.835e+14   <2e-16 ***
## attack_effort          -1.186e-02  1.567e-13 -7.570e+10   <2e-16 ***
## hp_effort              -5.324e-03  1.601e-13 -3.325e+10   <2e-16 ***
## is_legendaryTRUE       -1.078e-13  1.819e-13 -5.930e-01    0.554
## legendaryNormal                NA         NA         NA       NA
## log_height             -1.271e-02  2.749e-13 -4.625e+10   <2e-16 ***
## log_weight             -2.234e-02  8.813e-14 -2.535e+11   <2e-16 ***
## PC1                     5.406e+00  1.224e-14  4.417e+14   <2e-16 ***
## PC2                            NA         NA         NA       NA
## stageStage 1            6.510e-14  1.012e-13  6.440e-01    0.520
## stageStage 2           -5.150e-14  1.809e-13 -2.850e-01    0.776
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.929e-13 on 499 degrees of freedom
##   (444 observations deleted due to missingness)
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 3.179e+28 on 20 and 499 DF,  p-value: < 2.2e-16
```

#### Finding an optimal model

```
hp1 <- regsubsets(hp_base ~ . -X -name -type_primary -type_secondary -species,
                  data = pokemons_df,
                  nvmax = 15)
```

```
## Reordering variables and trying again:
```

```
hp1_sum <- summary(hp1)
```

```
which(hp1_sum$cp == max(hp1_sum$cp))
```

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

```
which(hp1_sum$bic == max(hp1_sum$bic))
```

```
## integer(0)
```

```
hp1_sum$aic <- length(pokemons_df$X) * log(hp1_sum$rss/length(pokemons_df$X)) + 2*15
hp1_sum$aic
```

```
##  [1]    5050.346   4718.780   4603.764   4546.242   4476.782   3187.102
##  [7]   -8300.256  -9323.877 -10460.895 -11403.449 -11890.094 -12810.442
## [13] -14374.403 -18243.183 -54772.123        NaN
```

From Cp, BIC, and AIC, it would seem that the submodel with only one parameter (as base experience) is the optimal model.

## Logistic regression for legendaries

We previously observed that legendary Pokemon tend to have high stats across all traits, suggesting that these combined could produce an accurate indicator of legendary status. Here, we train a logistic model for legendary status with train-test splits, showing that we can reach a high accuracy of $AUROC > 0.9$ on the test set.

### Train-test split

```
test_frac <- 0.33

total_n <- length(pokemons_df[,1])
test_n <- as.integer(total_n*test_frac)
train_n <- total_n - test_n

is_test <- c(rep(TRUE,each=test_n),rep(FALSE,each=train_n))
is_test <- sample(is_test)

train_pokemon <- pokemons_df[!is_test,]
test_pokemon <- pokemons_df[is_test,]
```

### Fit model

```
logit_mod <- glm(is_legendary ~ base_experience +
                   height + weight + speed_base +
                   defense_base + attack_base + hp_base,
               data = train_pokemon,
               family = binomial)

summary(logit_mod)
```

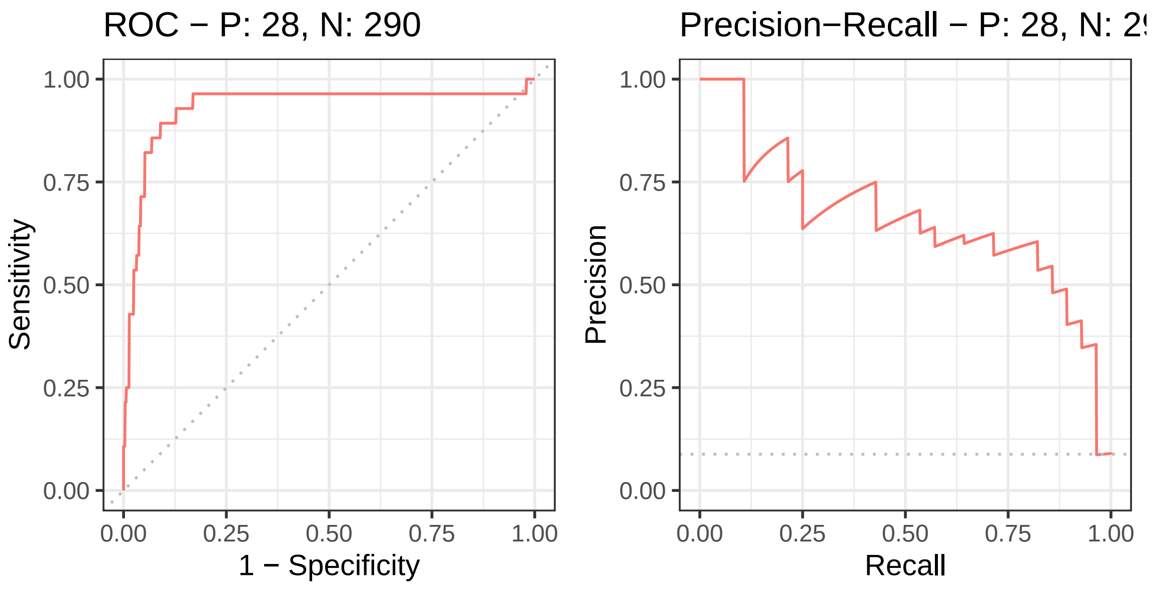
```
##
## Call:
## glm(formula = is_legendary ~ base_experience + height + weight +
##     speed_base + defense_base + attack_base + hp_base, family = binomial,
##     data = train_pokemon)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.9165  -0.1968  -0.0620  -0.0123   2.9028
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -1.402e+01  1.785e+00  -7.851 4.14e-15 ***
## base_experience  3.137e-02  5.531e-03   5.672 1.41e-08 ***
## height          -1.670e-02  1.595e-02  -1.047  0.29502
## weight           5.005e-04  1.516e-04   3.302  0.00096 ***
## speed_base       4.328e-02  1.022e-02   4.234 2.30e-05 ***
## defense_base     1.894e-02  7.922e-03   2.391  0.01678 *
## attack_base     -6.600e-03  7.247e-03  -0.911  0.36244
## hp_base          4.821e-04  9.692e-03   0.050  0.96033
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 455.75  on 645  degrees of freedom
## Residual deviance: 193.08  on 638  degrees of freedom
## AIC: 209.08
##
## Number of Fisher Scoring iterations: 8
```

```r
predicted <- predict(logit_mod, test_pokemon, type="response")
```

**ROC and precision-recall curve**

```r
library(precrec)

curves <- evalmod(scores = predicted, labels = test_pokemon$is_legendary)
autoplot(curves)
```



```r
auc(curves)
```

```
##   modnames dsids curvetypes      aucs
## 1       m1     1        ROC 0.9307882
## 2       m1     1        PRC 0.6568189
```

## Conclusion

Our analysis of the PokeAPI is by no means exhaustive, and there are many other interesting analyses that can be performed. However, we hope that the analyses contained in this project provide some level of insight into the relationships between different Pokemon statistics. In our project, we used various techniques covered in class, beginning with data visualization with `ggplot2`, generating null distributions with bootstrap, principal component analysis, linear regression, and finally logistic regression. It must be noted that these statistics are derived from the Pokemon games, and are thus not truly natural data, in a sense. Game developers chose the various characteristics of each Pokemon, in line with the Pokemon's appearance and lore. Even though these data are artificial, we feel that given Pokemon's popularity in recent years, as well as the plethora of data presented in the PokeAPI databased, such analyses are still worthwhile, and provide an opportunity to use data analysis techniques covered in S&DS 361 on a unique, interesting dataset. Another missed opportunity for more in-depth statistical analysis is the lack of actual samples in the data. Due to the nature of the Pokemon games, there is no variability between individuals of a specific Pokemon species. That is, each Bulbasaur will have the same statistics as the population values. If some variability were introduced in the game, a more realistic analysis could have been performed, using observations of different individuals even in the same Pokemon species. Fixed effects models, as well as other more sophisticated statistical techniques could then have been used. However, we found that even with the somewhat simplified scenario, the dataset still proved interesting to analyze.