The Statistics of Pokemon

null

null

Contents


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

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. For our final project for S&DS361, we decided that this wealth of 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 a myriad of 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.
- Growth Rate: Each Pokemon has a growth rate associated with it.
- 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.

API

To obtain these data, we made us 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, after which we made a separate request for each Pokemon to obtain more detailed information. The JSON files provided by the API were then processed into a DataFrame for subsequent 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.

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

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

head(pokemons_df)
```

```
##   X       name base_experience height weight    species type_primary
## 1 1  bulbasaur              64      7     69  bulbasaur       poison
```

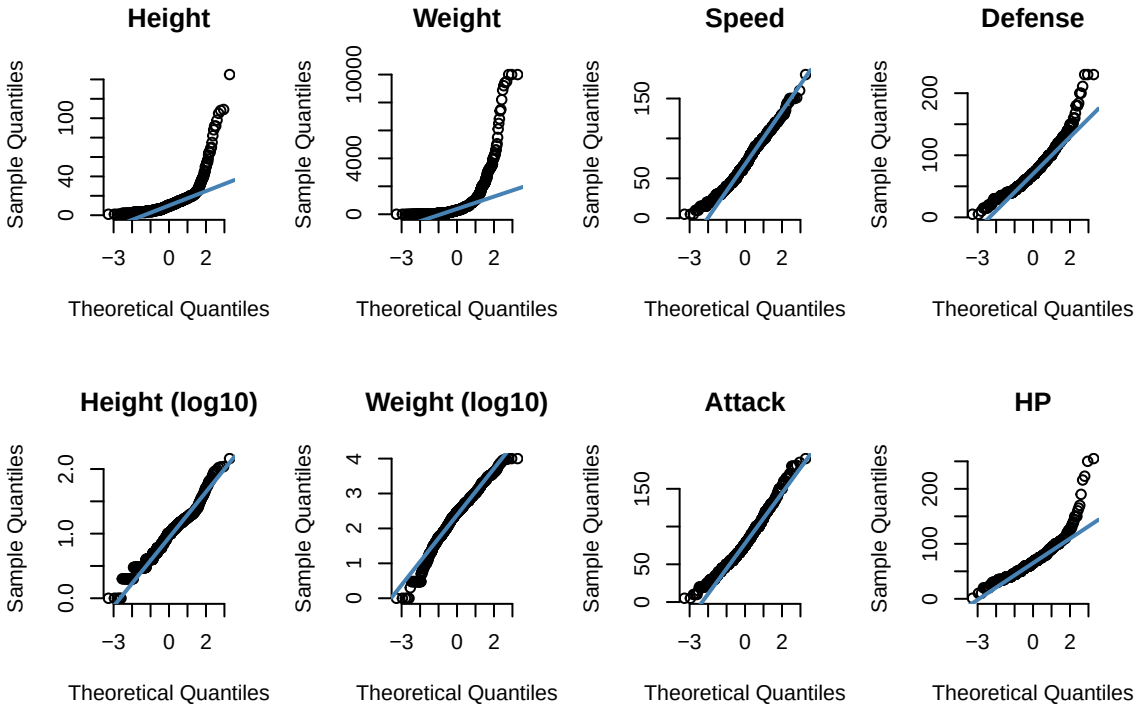
```
## 2 2    ivysaur             142     10    130    ivysaur       poison
## 3 3   venusaur             236     20   1000   venusaur       poison
## 4 4 charmander              62      6     85 charmander         fire
## 5 5 charmeleon             142     11    190 charmeleon         fire
## 6 6  charizard             240     17    905  charizard       flying
##   type_secondary speed_base speed_effort special.defense_base
## 1          grass         45            0                  65
## 2          grass         60            0                  80
## 3          grass         80            0                 100
## 4           <NA>         65            1                  50
## 5           <NA>         80            1                  65
## 6           fire        100            0                  85
##   special.defense_effort special.attack_base special.attack_effort defense_base
## 1                      0                  65                     1           49
## 2                      1                  80                     1           63
## 3                      1                 100                     2           83
## 4                      0                  60                     0           43
## 5                      0                  80                     1           58
## 6                      0                 109                     3           78
##   defense_effort attack_base attack_effort hp_base hp_effort is_legendary
## 1              0          49             0      45         0        FALSE
## 2              0          62             0      60         0        FALSE
## 3              0          82             0      80         0        FALSE
## 4              0          52             0      39         0        FALSE
## 5              0          64             0      58         0        FALSE
## 6              0          84             0      78         0        FALSE
##   legendary log_height log_weight
## 1    Normal  0.8450980   1.838849
## 2    Normal  1.0000000   2.113943
## 3    Normal  1.3010300   3.000000
## 4    Normal  0.7781513   1.929419
## 5    Normal  1.0413927   2.278754
## 6    Normal  1.2304489   2.956649
```

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 of less normally-distributed (and height and weight after a log-transformation). However, we also see that there is a skew towards Pokemon with hig hHP and defense.

```
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.

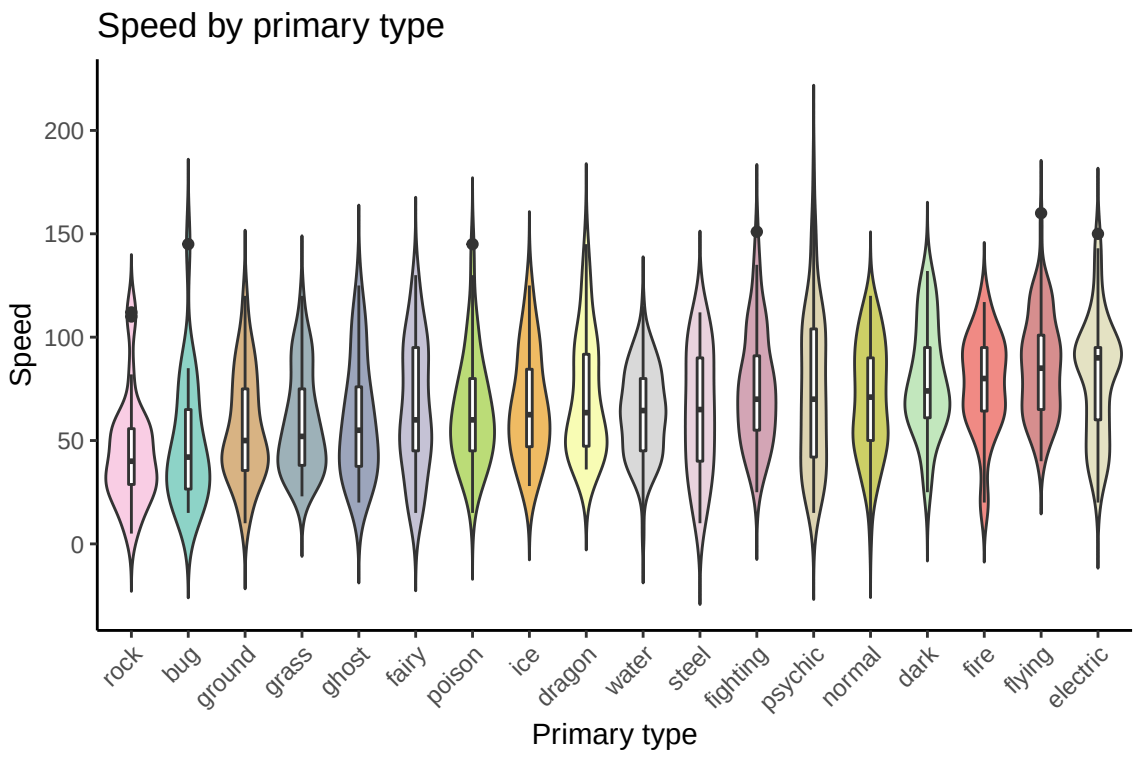
```
plot_by_primary <- function(var,var_name){

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

  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.

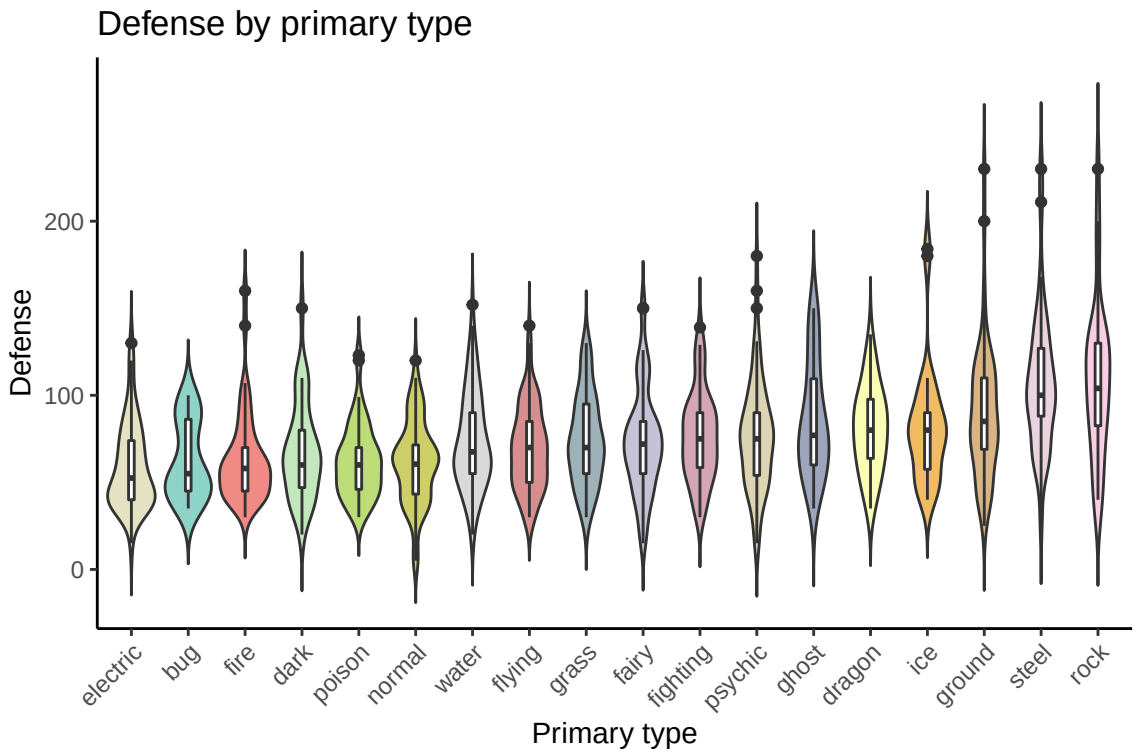
```
plot_by_primary("speed_base","Speed")
```



Base defense

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

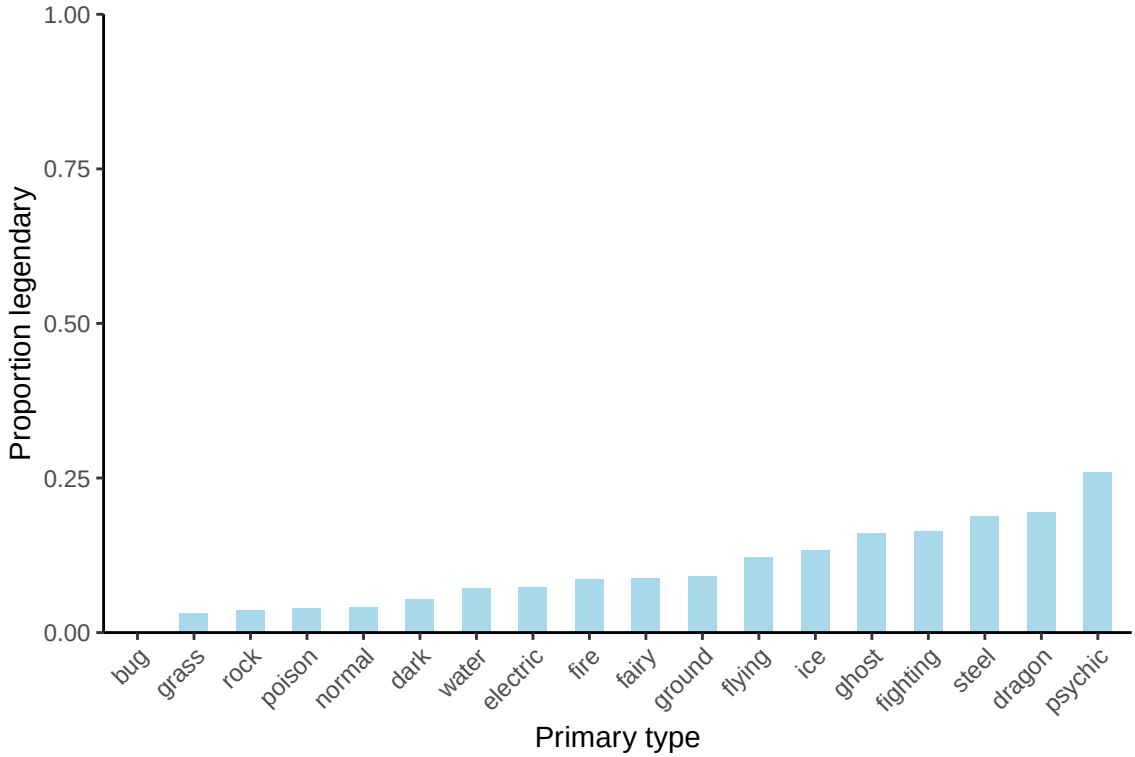
```
plot_by_primary("defense_base","Defense")
```



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.

```
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")
```



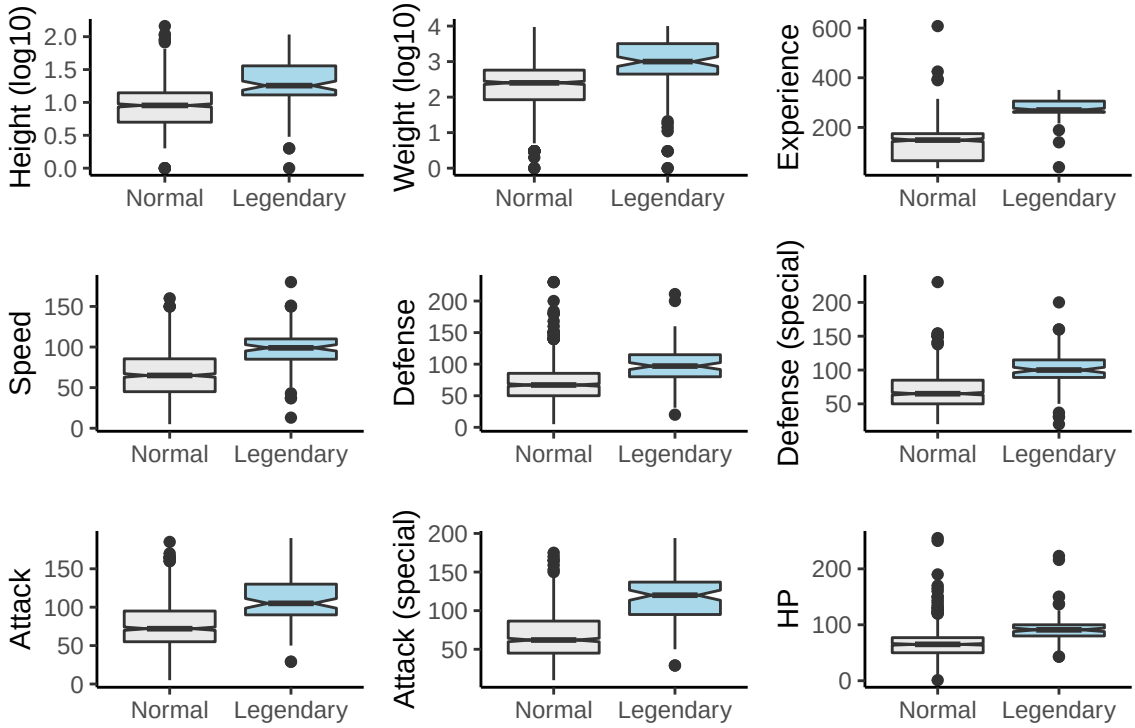
Comparisons of mormal and legendary Pokemon statistics

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.

```
library(gridExtra)

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"))
}

plots <- do.call(grid.arrange, c(p, ncol=3))
```



Statistical analysis

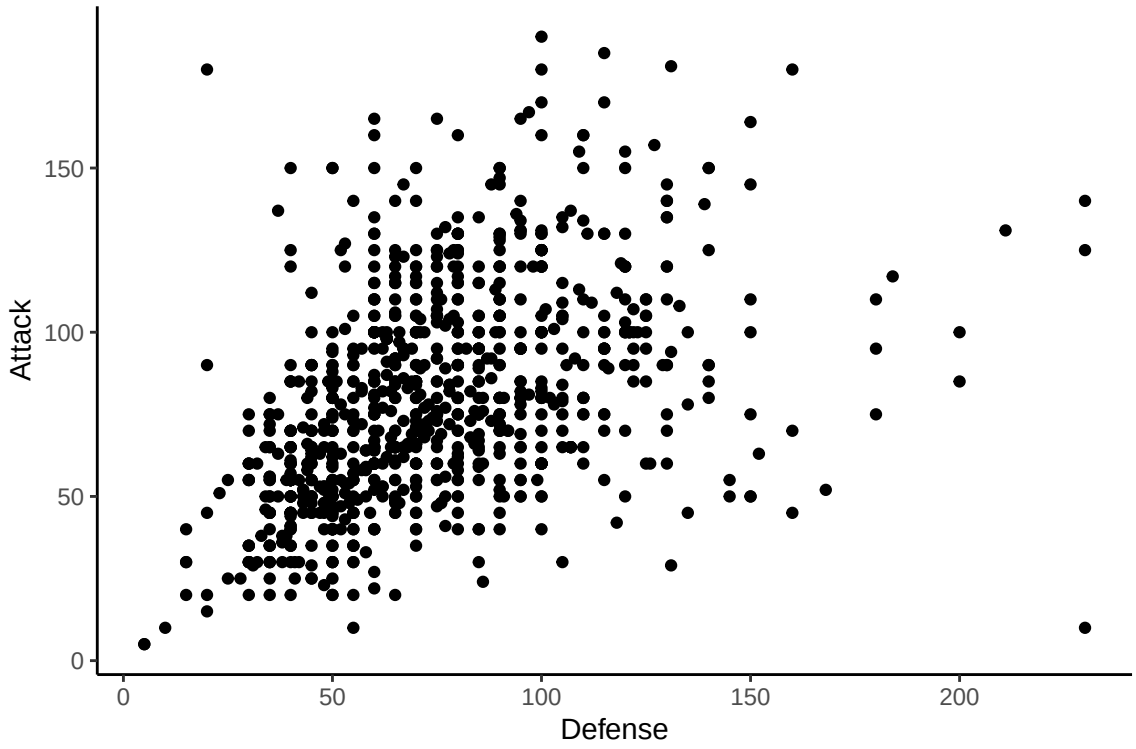
Bootstrap analysis

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")
```



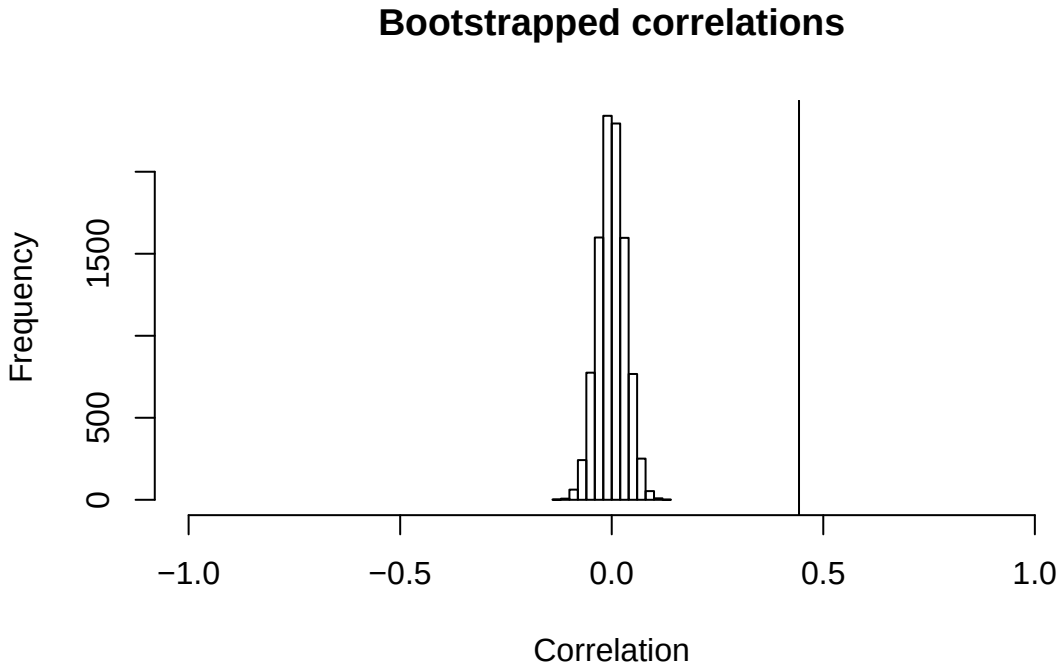

```
corr <- cor(pokemons_df$defense_base, pokemons_df$attack_base)
corr
```

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

Testing significance with bootstrap

```
n <- 10000
corr_bs <- rep(0, n)
for (i in 1:n){
  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))
```

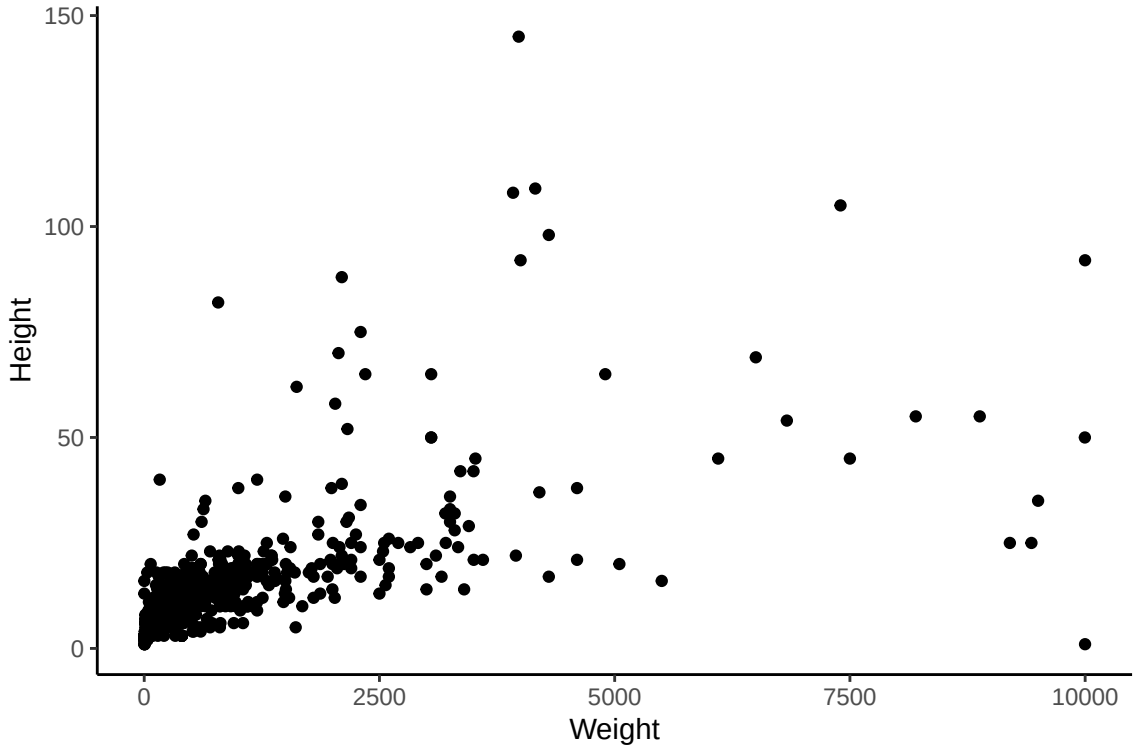
```
## [1] 0
```

From this result, we see that there were no bootstrapped samples with a correlation greater than the observed correlation in the null distribution.

Height and weight

Plot

```
ggplot(pokemons_df, aes(x = weight, y = height))+
  geom_point() +
  theme_classic() +
  labs(x="Weight", y = "Height")
```



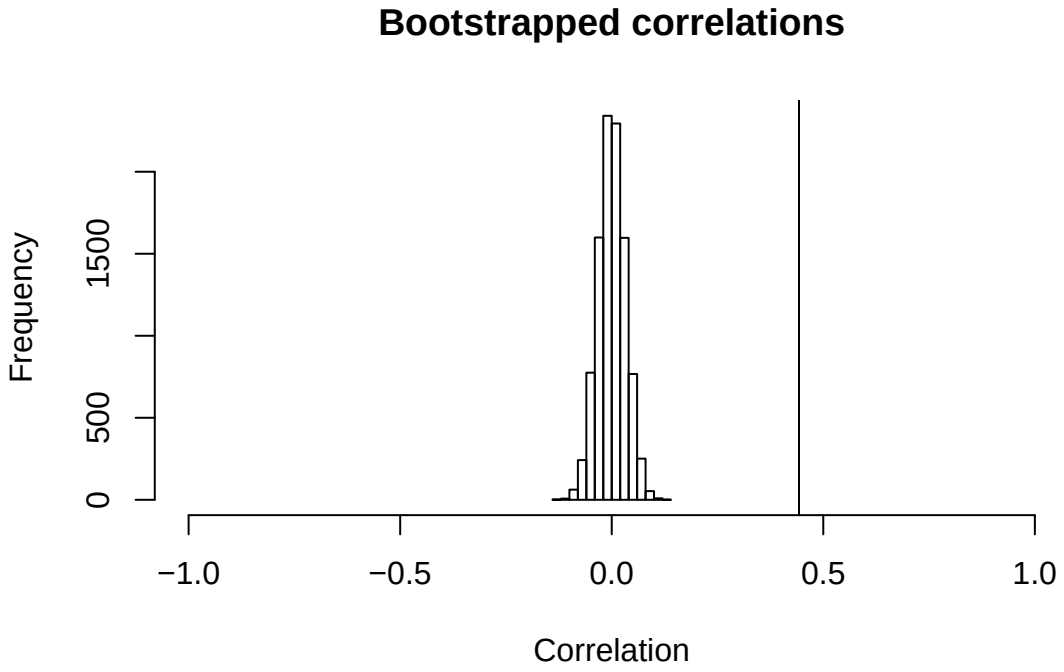

```
corr <- cor(pokemons_df$weight, pokemons_df$height)
corr
```

```
## [1] 0.6631623
```

Testing significance with bootstrap

```
n <- 10000
corr_bs <- rep(0, n)
for (i in 1:n){
  weight_sample <- sample(pokemons_df$weight, length(pokemons_df$weight), replace = T)
  height_sample <- sample(pokemons_df$height, length(pokemons_df$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.

Linear regression: predicting base health

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

Naive full regression

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

```
summary(nfr_hp)
```

```
##
## Call:
## lm(formula = hp_base ~ . - X - name - species, data = pokemons_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -67.835  -7.346  -0.390   7.005  65.597
##
## Coefficients: (1 not defined because of singularities)
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)            40.9117499  7.9682291   5.134 4.16e-07 ***
## base_experience         0.3301697  0.0430558   7.668 1.02e-13 ***
## height                 -0.0597679  0.1015332  -0.589 0.556377
## weight                  0.0020619  0.0008125   2.538 0.011485 *
## type_primarydark        2.8392859  6.3722035   0.446 0.656111
```

```
## type_primarydragon      2.7507907  6.5904580   0.417 0.676585
## type_primaryelectric   10.0300689  7.0786461   1.417 0.157164
## type_primaryfairy       2.2251681  6.2547248   0.356 0.722182
## type_primaryfighting    9.1434548  6.3238042   1.446 0.148881
## type_primaryfire        1.5612555  6.9555581   0.224 0.822496
## type_primaryflying      9.6002057  5.9386470   1.617 0.106646
## type_primaryghost       2.3324389  6.5372137   0.357 0.721407
## type_primarygrass       6.9994430  6.4137911   1.091 0.275697
## type_primaryground      2.7356142  6.1858238   0.442 0.658520
## type_primaryice        15.4265453  6.7713673   2.278 0.023164 *
## type_primarynormal      8.7655103  7.9662477   1.100 0.271754
## type_primarypoison      2.3251305  6.3286519   0.367 0.713489
## type_primarypsychic     9.1294625  6.2550116   1.460 0.145086
## type_primaryrock       11.3416169  7.0095233   1.618 0.106330
## type_primarysteel       1.9362569  6.3259535   0.306 0.759679
## type_primarywater       1.0933885  6.6213319   0.165 0.868912
## type_secondarydark      8.6287101  3.4828100   2.478 0.013582 *
## type_secondarydragon    0.1475247  3.6306706   0.041 0.967606
## type_secondaryelectric  1.7457230  3.7371454   0.467 0.640626
## type_secondaryfairy     1.2785470 10.0376881   0.127 0.898699
## type_secondaryfighting  3.6517942  5.0707605   0.720 0.471782
## type_secondaryfire      2.4242770  3.3347245   0.727 0.467602
## type_secondaryflying    9.3748154 10.3530532   0.906 0.365660
## type_secondaryghost     6.9684377  3.3053038   2.108 0.035540 *
## type_secondarygrass     4.3198236  2.8397499   1.521 0.128885
## type_secondaryground    4.0429993  3.6516644   1.107 0.268791
## type_secondaryice       5.5761036  4.2233364   1.320 0.187378
## type_secondarynormal    3.4284170  2.9337800   1.169 0.243160
## type_secondarypoison    5.6331222  3.6883721   1.527 0.127371
## type_secondarypsychic   0.1449710  3.5057130   0.041 0.967032
## type_secondaryrock      3.0766873  2.7676206   1.112 0.266850
## type_secondarysteel    -4.2258602  3.7605064  -1.124 0.261696
## type_secondarywater     5.1737321  2.6943713   1.920 0.055440 .
## speed_base             -0.1588987  0.0381480  -4.165 3.70e-05 ***
## speed_effort           -8.3375054  2.7013719  -3.086 0.002146 **
## special.defense_base   -0.1339550  0.0430524  -3.111 0.001976 **
## special.defense_effort -7.2768590  2.7476264  -2.648 0.008360 **
## special.attack_base    -0.0375552  0.0395147  -0.950 0.342395
## special.attack_effort  -8.8114239  2.6451564  -3.331 0.000933 ***
## defense_base           -0.1291198  0.0429463  -3.007 0.002785 **
## defense_effort         -9.9198130  2.7579559  -3.597 0.000356 ***
## attack_base            -0.0536587  0.0393604  -1.363 0.173454
## attack_effort          -6.3987494  2.6568801  -2.408 0.016410 *
## hp_effort               4.5799744  2.8123745   1.629 0.104090
## is_legendaryTRUE       -0.7102800  2.5788975  -0.275 0.783115
## legendaryNormal                NA         NA      NA       NA
## log_height              7.7621879  5.5550232   1.397 0.162977
## log_weight              4.2725329  1.8568702   2.301 0.021834 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.51 on 468 degrees of freedom
##   (444 observations deleted due to missingness)
## Multiple R-squared:   0.72,  Adjusted R-squared:  0.6895
```

```
## F-statistic:  23.6 on 51 and 468 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.

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

```
summary(hp1)
```

```
##
## Call:
## lm(formula = hp_base ~ . - X - type_primary - type_secondary -
##     name - species, data = pokemons_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -74.029  -7.703  -0.434   6.543  61.941
##
## Coefficients: (1 not defined because of singularities)
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)             4.874e+01  5.009e+00   9.732  < 2e-16 ***
## base_experience         3.355e-01  4.137e-02   8.109 3.95e-15 ***
## height                 -1.255e-01  9.913e-02  -1.266 0.205931
## weight                  1.881e-03  7.864e-04   2.393 0.017095 *
## speed_base             -1.315e-01  3.493e-02  -3.763 0.000188 ***
## speed_effort           -1.013e+01  2.639e+00  -3.840 0.000139 ***
## special.defense_base   -1.369e-01  4.123e-02  -3.320 0.000966 ***
## special.defense_effort -8.892e+00  2.677e+00  -3.321 0.000961 ***
## special.attack_base    -3.075e-02  3.655e-02  -0.841 0.400533
## special.attack_effort  -1.115e+01  2.573e+00  -4.335 1.76e-05 ***
## defense_base           -1.446e-01  3.803e-02  -3.801 0.000162 ***
## defense_effort         -1.172e+01  2.665e+00  -4.396 1.35e-05 ***
## attack_base            -5.137e-02  3.765e-02  -1.364 0.173096
## attack_effort          -8.456e+00  2.604e+00  -3.247 0.001243 **
## hp_effort               2.914e+00  2.769e+00   1.052 0.293132
## is_legendaryTRUE       -1.423e+00  2.477e+00  -0.574 0.565895
## legendaryNormal                NA         NA      NA       NA
## log_height              1.100e+01  5.300e+00   2.076 0.038405 *
## log_weight              4.703e+00  1.738e+00   2.707 0.007031 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.82 on 502 degrees of freedom
##   (444 observations deleted due to missingness)
## Multiple R-squared:  0.6857, Adjusted R-squared:  0.6751
## F-statistic: 64.43 on 17 and 502 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:
```

```r
hp1_sum <- summary(hp1)
hp1_sum$which
```

```
##    (Intercept) base_experience height weight speed_base speed_effort
## 1         TRUE            TRUE  FALSE  FALSE      FALSE        FALSE
## 2         TRUE            TRUE  FALSE  FALSE      FALSE        FALSE
## 3         TRUE            TRUE  FALSE  FALSE      FALSE        FALSE
## 4         TRUE            TRUE  FALSE  FALSE      FALSE        FALSE
## 5         TRUE            TRUE  FALSE  FALSE      FALSE        FALSE
## 6         TRUE            TRUE  FALSE   TRUE      FALSE        FALSE
## 7         TRUE            TRUE  FALSE   TRUE       TRUE        FALSE
## 8         TRUE            TRUE  FALSE  FALSE       TRUE        FALSE
## 9         TRUE            TRUE  FALSE   TRUE       TRUE        FALSE
## 10        TRUE            TRUE  FALSE  FALSE       TRUE         TRUE
## 11        TRUE            TRUE  FALSE   TRUE       TRUE         TRUE
## 12        TRUE            TRUE  FALSE   TRUE       TRUE         TRUE
## 13        TRUE            TRUE  FALSE   TRUE       TRUE         TRUE
## 14        TRUE            TRUE   TRUE   TRUE       TRUE         TRUE
## 15        TRUE            TRUE   TRUE   TRUE       TRUE         TRUE
## 16        TRUE            TRUE   TRUE   TRUE       TRUE         TRUE
##    special.defense_base special.defense_effort special.attack_base
## 1                 FALSE                  FALSE               FALSE
## 2                 FALSE                  FALSE               FALSE
## 3                 FALSE                  FALSE               FALSE
## 4                 FALSE                  FALSE               FALSE
## 5                 FALSE                  FALSE               FALSE
## 6                 FALSE                  FALSE               FALSE
## 7                 FALSE                  FALSE               FALSE
## 8                  TRUE                  FALSE               FALSE
## 9                  TRUE                  FALSE               FALSE
## 10                 TRUE                   TRUE               FALSE
## 11                 TRUE                   TRUE               FALSE
## 12                 TRUE                   TRUE               FALSE
## 13                 TRUE                   TRUE               FALSE
## 14                 TRUE                   TRUE               FALSE
## 15                 TRUE                   TRUE               FALSE
## 16                 TRUE                   TRUE                TRUE
##    special.attack_effort defense_base defense_effort attack_base attack_effort
## 1                  FALSE        FALSE          FALSE       FALSE         FALSE
## 2                  FALSE        FALSE          FALSE       FALSE         FALSE
## 3                  FALSE        FALSE          FALSE       FALSE         FALSE
## 4                  FALSE         TRUE          FALSE       FALSE         FALSE
## 5                  FALSE         TRUE          FALSE       FALSE          TRUE
## 6                  FALSE         TRUE          FALSE       FALSE          TRUE
## 7                  FALSE         TRUE          FALSE       FALSE          TRUE
## 8                   TRUE         TRUE           TRUE       FALSE         FALSE
## 9                   TRUE         TRUE           TRUE       FALSE         FALSE
## 10                  TRUE         TRUE           TRUE       FALSE          TRUE
## 11                  TRUE         TRUE           TRUE       FALSE          TRUE
## 12                  TRUE         TRUE           TRUE       FALSE          TRUE
## 13                  TRUE         TRUE           TRUE       FALSE          TRUE
## 14                  TRUE         TRUE           TRUE       FALSE          TRUE
## 15                  TRUE         TRUE           TRUE        TRUE          TRUE
## 16                  TRUE         TRUE           TRUE        TRUE          TRUE
```

```
##    hp_effort is_legendaryTRUE legendaryNormal log_height log_weight
## 1      FALSE             FALSE           FALSE      FALSE      FALSE
## 2       TRUE             FALSE           FALSE      FALSE      FALSE
## 3       TRUE             FALSE           FALSE      FALSE       TRUE
## 4       TRUE             FALSE           FALSE      FALSE       TRUE
## 5       TRUE             FALSE           FALSE      FALSE       TRUE
## 6       TRUE             FALSE           FALSE      FALSE       TRUE
## 7       TRUE             FALSE           FALSE      FALSE       TRUE
## 8       TRUE             FALSE           FALSE      FALSE       TRUE
## 9       TRUE             FALSE           FALSE      FALSE       TRUE
## 10     FALSE             FALSE           FALSE      FALSE       TRUE
## 11     FALSE             FALSE           FALSE      FALSE       TRUE
## 12      TRUE             FALSE           FALSE      FALSE       TRUE
## 13      TRUE             FALSE           FALSE       TRUE       TRUE
## 14      TRUE             FALSE           FALSE       TRUE       TRUE
## 15      TRUE             FALSE           FALSE       TRUE       TRUE
## 16      TRUE             FALSE           FALSE       TRUE       TRUE
```

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

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

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

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

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

```
##  [1] 5063.840 4754.106 4628.071 4583.243 4554.873 4535.290 4523.412 4514.194
##  [9] 4499.767 4493.710 4486.209 4478.427 4473.712 4470.020 4466.633 4465.310
```

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 saw before 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 with train-test splits, showing that we can reach a high accuracy of $AUROC > 0.975$ 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

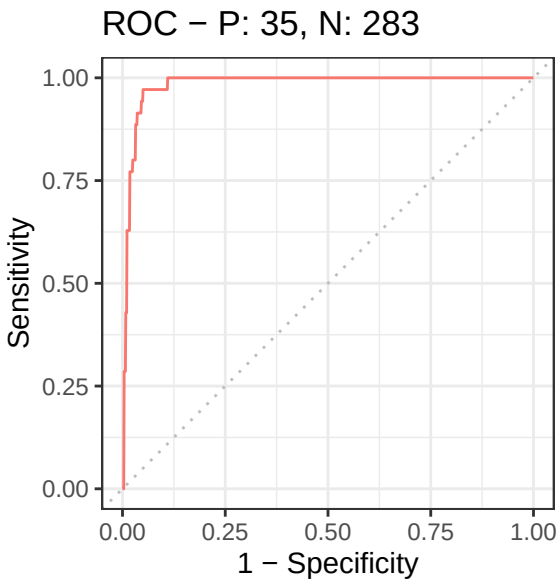
```
logit_mod <- glm(is_legendary ~ base_experience +
                   height + weight + speed_base +
                   defense_base + attack_base + hp_base,
                 data = train_pokemon,
                 family = binomial)

summary(logit_mod)
```

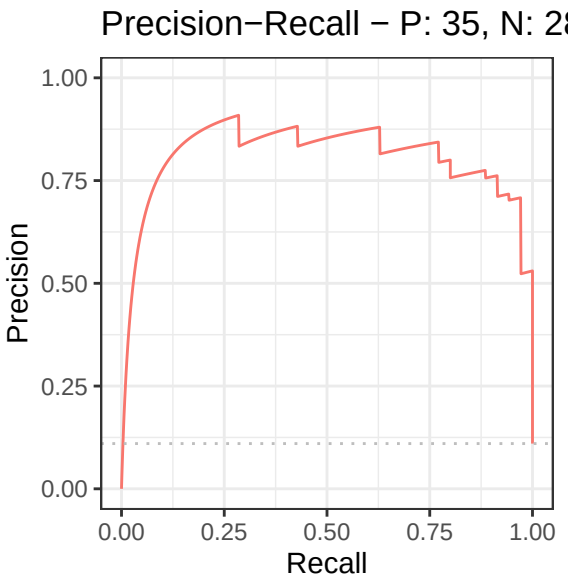
```
##
## Call:
## glm(formula = is_legendary ~ base_experience + height + weight +
##     speed_base + defense_base + attack_base + hp_base, family = binomial,
##     data = train_pokemon)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.1933  -0.2461  -0.1055  -0.0294   4.1589
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -1.053e+01  1.437e+00  -7.325 2.39e-13 ***
## base_experience  2.949e-02  4.913e-03   6.002 1.95e-09 ***
## height          -3.604e-04  1.902e-02  -0.019  0.98488
## weight           4.410e-04  1.539e-04   2.865  0.00417 **
## speed_base       2.591e-02  8.820e-03   2.937  0.00331 **
## defense_base     2.582e-03  7.528e-03   0.343  0.73158
## attack_base     -4.679e-03  6.207e-03  -0.754  0.45099
## hp_base         -4.695e-03  9.360e-03  -0.502  0.61596
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 426.13  on 645  degrees of freedom
## Residual deviance: 214.03  on 638  degrees of freedom
## AIC: 230.03
##
## Number of Fisher Scoring iterations: 7
```

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

ROC and precision-recall curve

```
library(precrec)

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






```
auc(curves)
```

```
##   modnames dsids curvetypes      aucs
## 1       m1     1        ROC 0.9830389
## 2       m1     1        PRC 0.7963164
```