pokemon-CK

Chaitanya Kharche

2024-04-16

```
# Load required libraries
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.3.3

## Loading required package: lattice
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.3.3

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 4.3.3
```

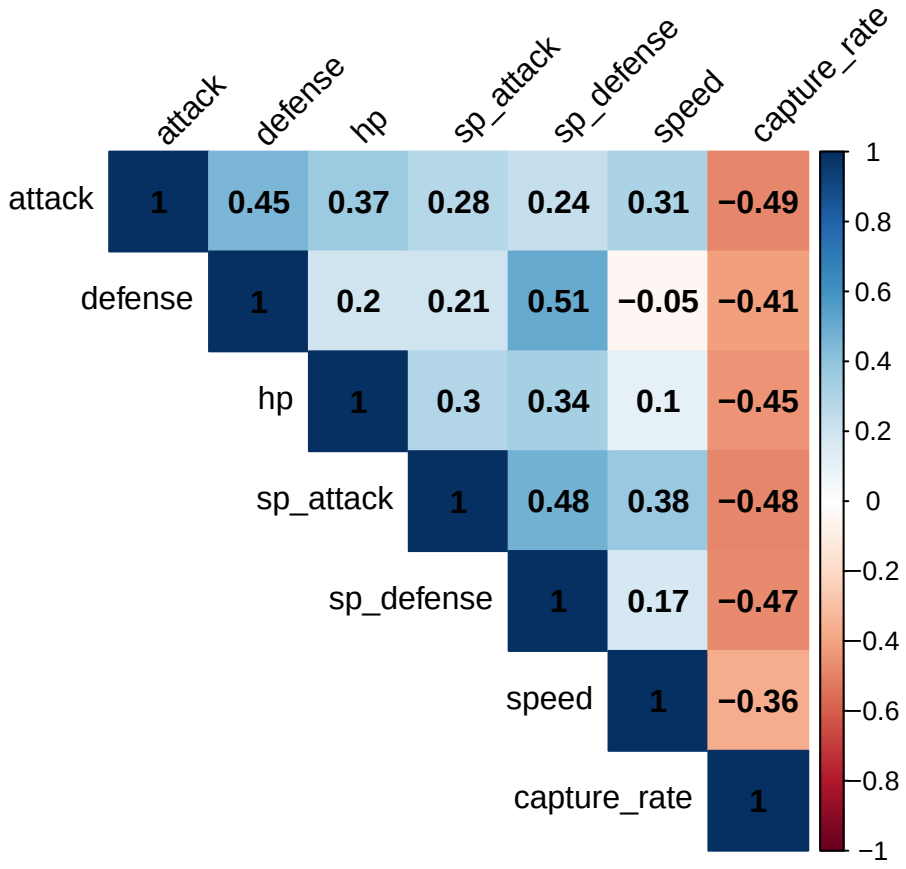
```
##
## Attaching package: 'xgboost'
```

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

```r
# Load the dataset
pokemon_data <- read.csv("C:/Users/91965/Downloads/pokemon.csv")

# Data cleaning and preprocessing
pokemon_data <- pokemon_data %>%
  mutate(capture_rate = as.integer(capture_rate),
         generation = as.integer(generation),
         is_legendary = as.integer(is_legendary),
         type1 = as.factor(type1),
         type2 = as.factor(type2)) %>%
  na.omit()
```

```
## Warning: There was 1 warning in `mutate()`.
## i In argument: `capture_rate = as.integer(capture_rate)`.
## Caused by warning:
## ! NAs introduced by coercion
```

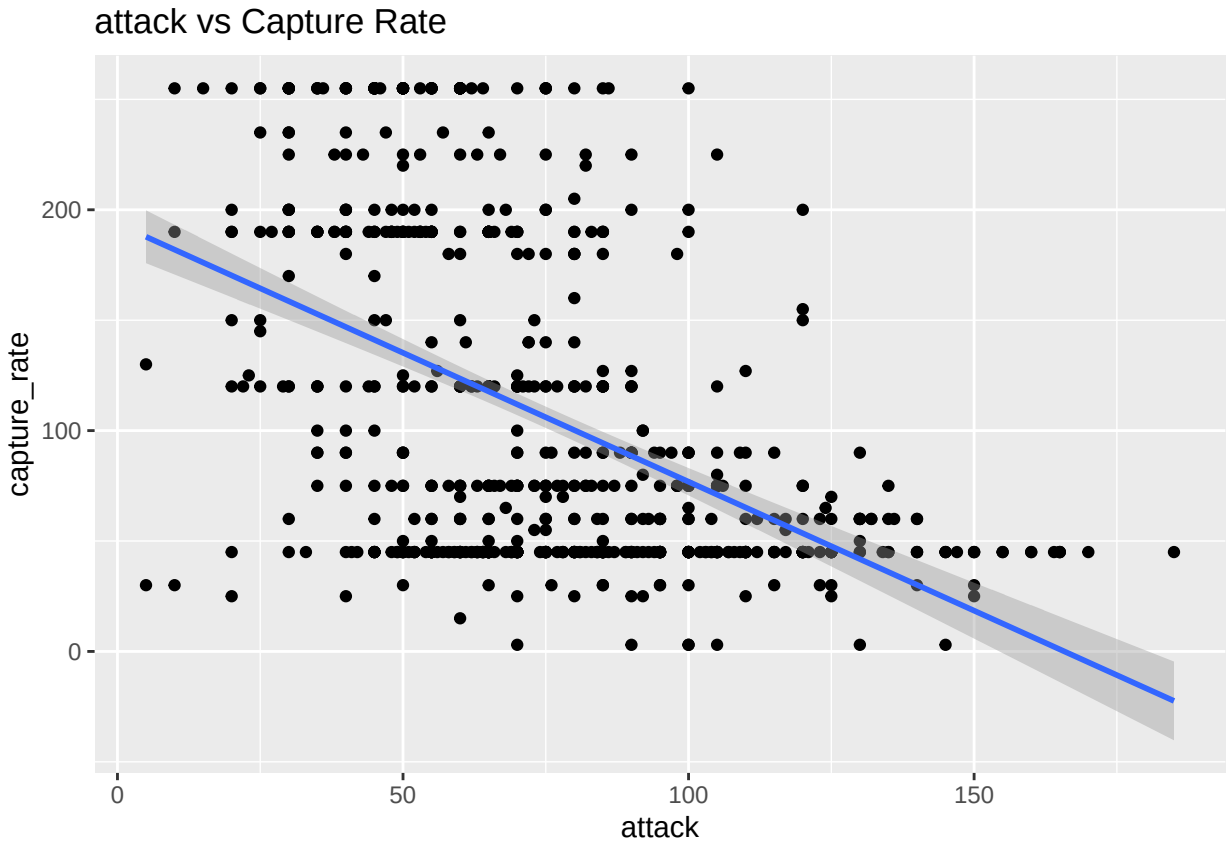
```r
# Encode categorical variables
pokemon_data <- pokemon_data %>%
  mutate(type1_encoded = as.integer(type1),
         type2_encoded = as.integer(type2))

# Correlation analysis
base_stats <- c("attack", "defense", "hp", "sp_attack", "sp_defense", "speed")
corr_matrix <- cor(pokemon_data[, c(base_stats, "capture_rate")])
corrplot::corrplot(corr_matrix, method = "color", type = "upper",
                   addCoef.col = "black", tl.col = "black", tl.srt = 45)
```



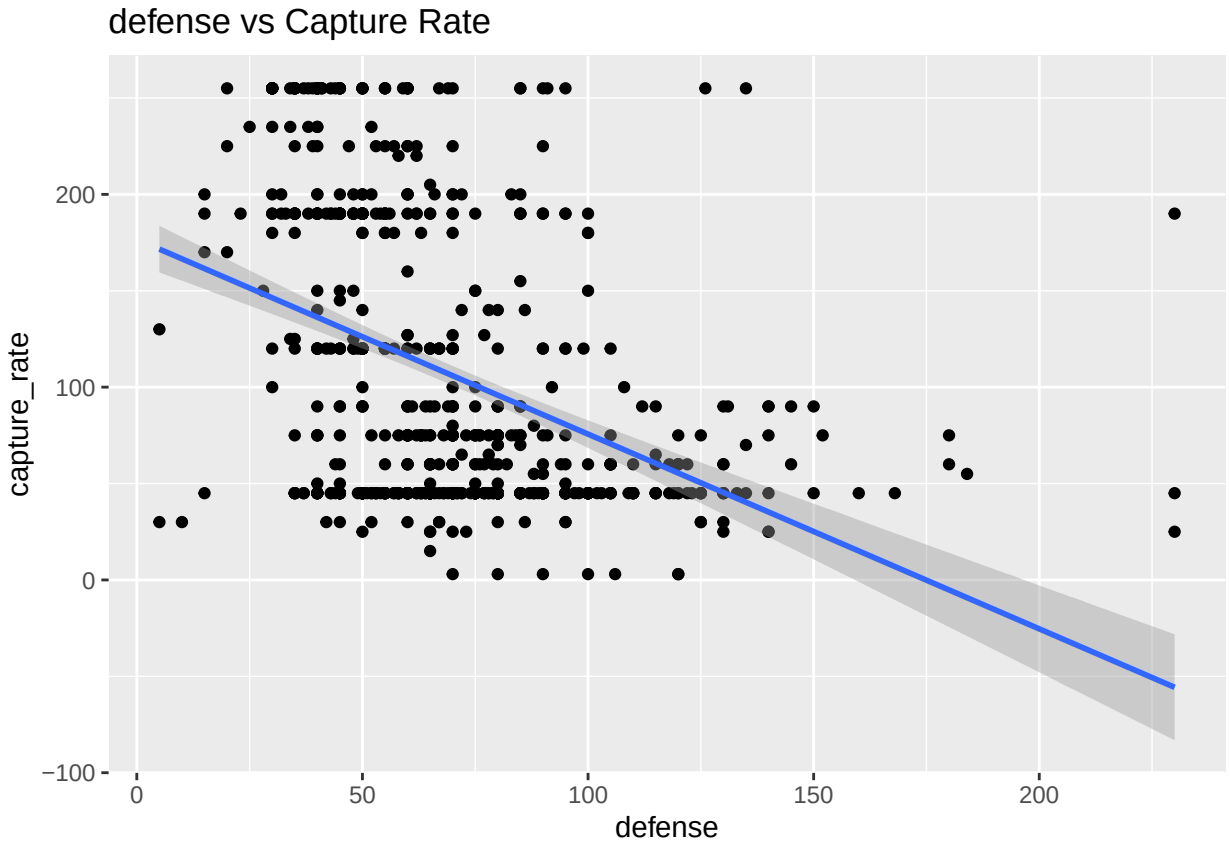

```
# Scatter plots
for (stat in base_stats) {
  plot <- ggplot(pokemon_data, aes_string(x = stat, y = "capture_rate")) +
    geom_point() +
    geom_smooth(method = "lm") +
    ggtitle(paste(stat, "vs Capture Rate"))
  print(plot)
}
```

```
## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
## i Please use tidy evaluation idioms with `aes()`.
## i See also `vignette("ggplot2-in-packages")` for more information.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

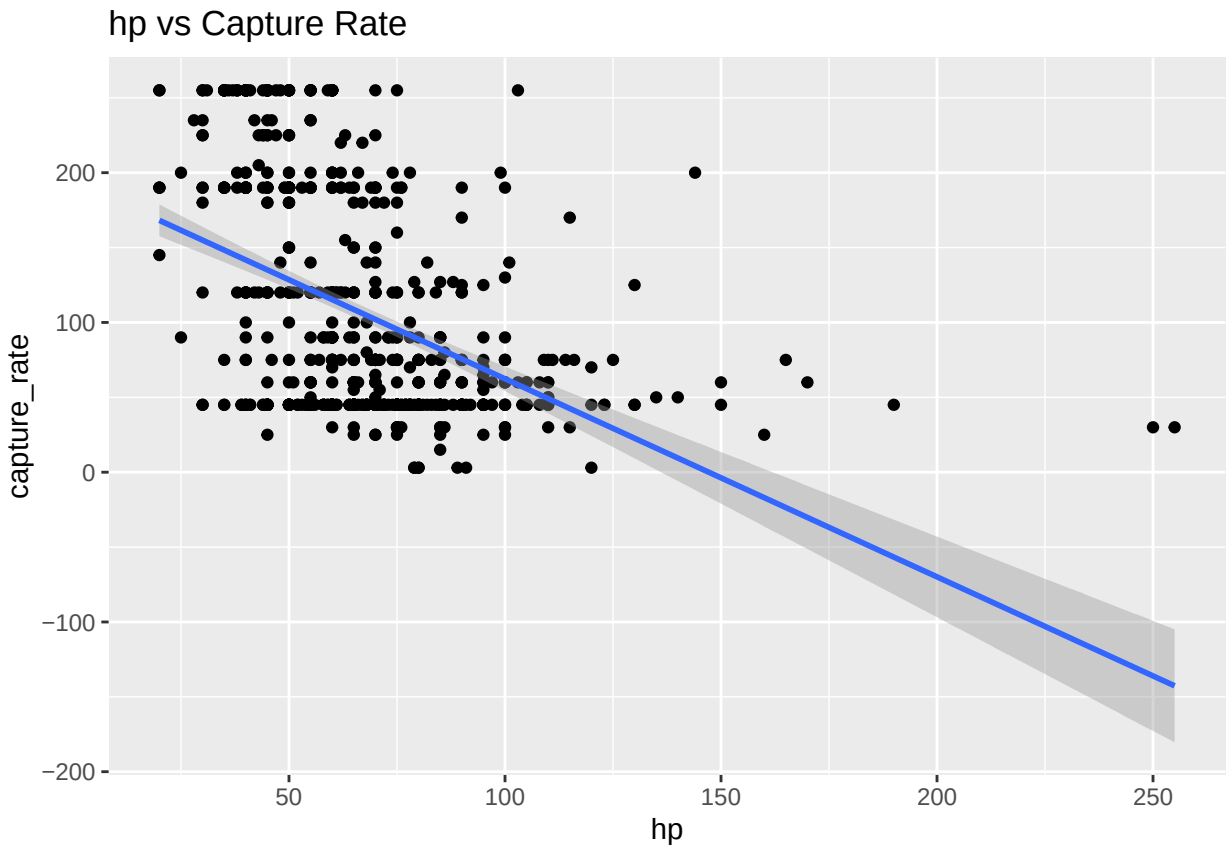
```
## `geom_smooth()` using formula = 'y ~ x'
```



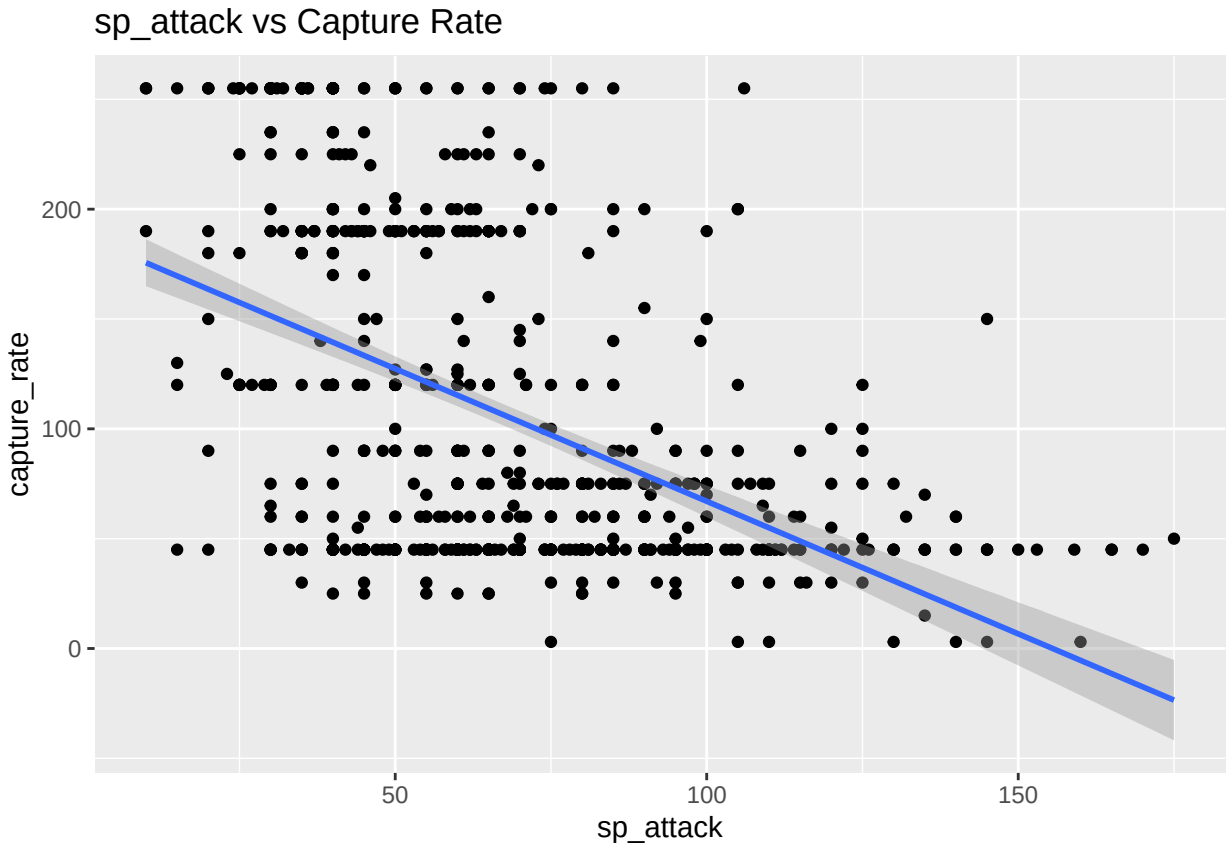

```
## `geom_smooth()` using formula = 'y ~ x'
```



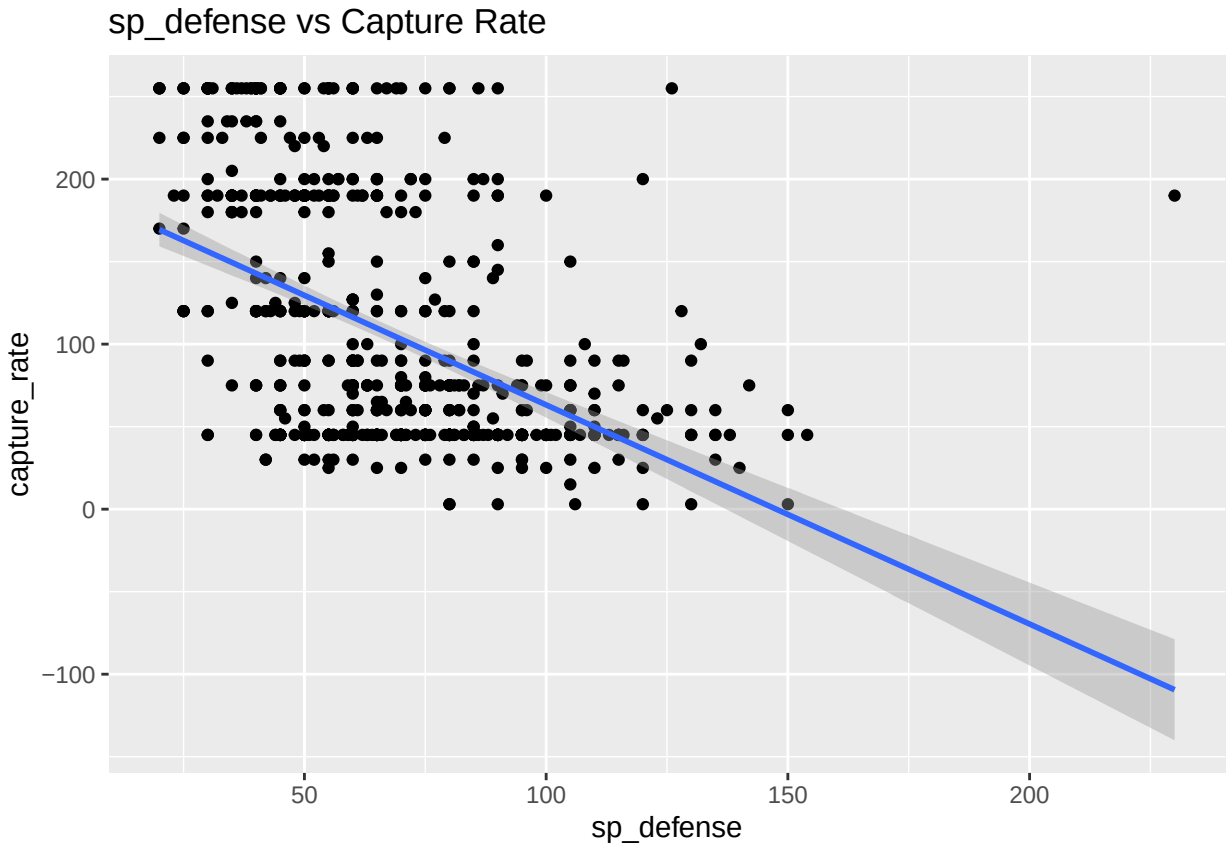

```
## `geom_smooth()` using formula = 'y ~ x'
```



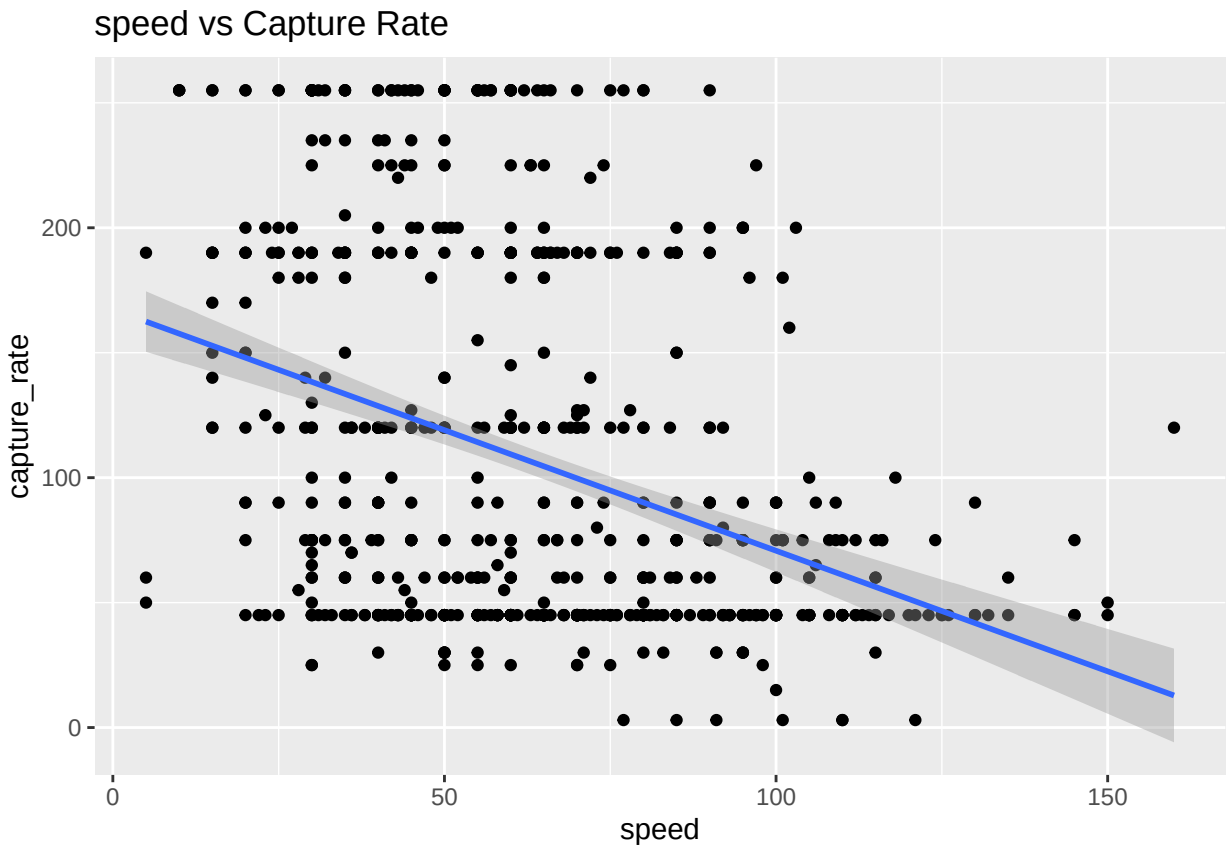

```
## `geom_smooth()` using formula = 'y ~ x'
```



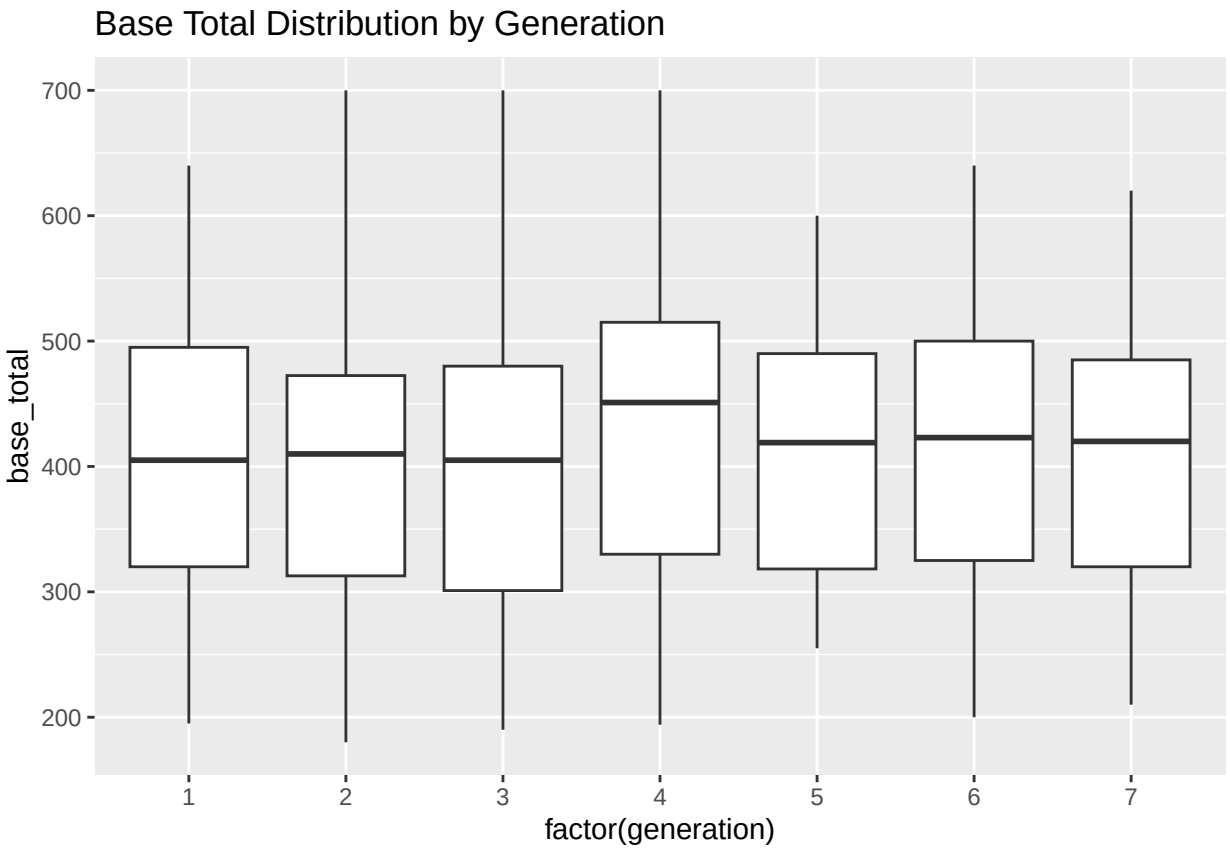

```
## `geom_smooth()` using formula = 'y ~ x'
```



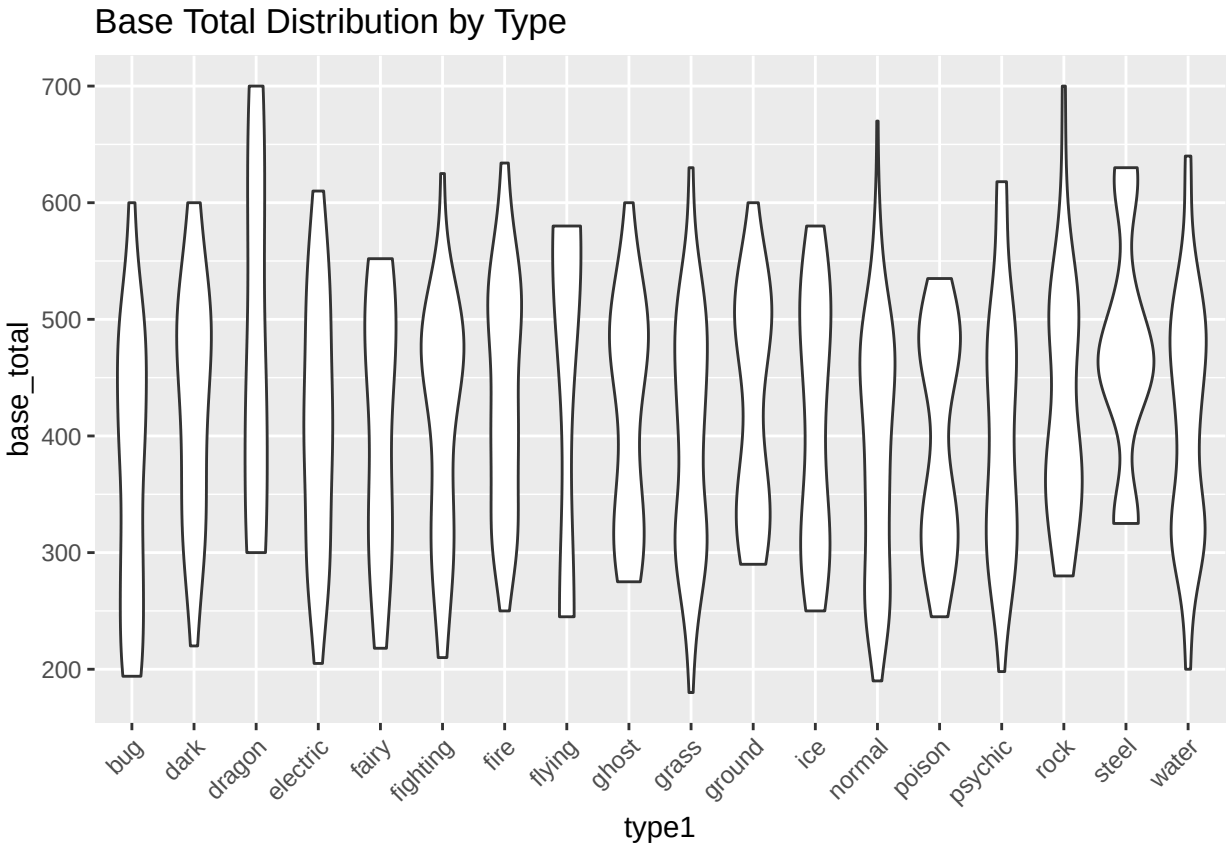

```
## `geom_smooth()` using formula = 'y ~ x'
```



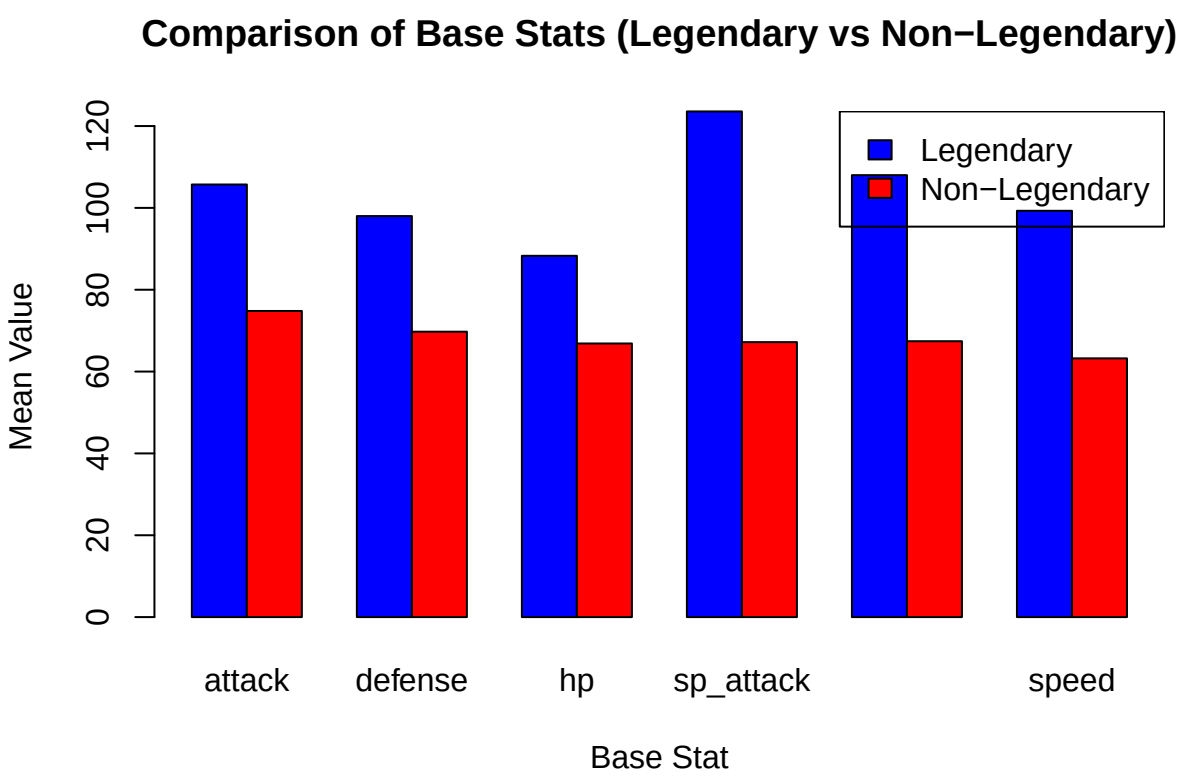

```
# Box plots for generations
ggplot(pokemon_data, aes(x = factor(generation), y = base_total)) +
  geom_boxplot() +
  ggtitle("Base Total Distribution by Generation")
```




```
# Violin plots for types
ggplot(pokemon_data, aes(x = type1, y = base_total)) +
  geom_violin() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  ggtitle("Base Total Distribution by Type")
```




```r
# Legendary vs non-legendary comparison
legendary_stats <- colMeans(pokemon_data[pokemon_data$is_legendary == 1, base_stats])
nonlegendary_stats <- colMeans(pokemon_data[pokemon_data$is_legendary == 0, base_stats])

barplot(rbind(legendary_stats, nonlegendary_stats),
        beside = TRUE, col = c("blue", "red"),
        names.arg = base_stats, xlab = "Base Stat", ylab = "Mean Value",
        main = "Comparison of Base Stats (Legendary vs Non-Legendary)")
legend("topright", legend = c("Legendary", "Non-Legendary"), fill = c("blue", "red"))
```




```
# Prepare data for modeling
X <- pokemon_data[, base_stats]
y <- pokemon_data$is_legendary

# Split data into train and test sets
set.seed(42)
trainIndex <- createDataPartition(y, p = 0.8, list = FALSE)
X_train <- X[trainIndex, ]
X_test <- X[-trainIndex, ]
y_train <- y[trainIndex]
y_test <- y[-trainIndex]

# Train random forest model
rf_model <- randomForest(x = X_train, y = factor(y_train), importance = TRUE)

# Train XGBoost model
xgb_model <- xgboost(data = as.matrix(X_train), label = y_train,
                     nrounds = 100, objective = "binary:logistic")
```

```
## [1]  train-logloss:0.449195
## [2]  train-logloss:0.314027
## [3]  train-logloss:0.228981
## [4]  train-logloss:0.170650
## [5]  train-logloss:0.129748
## [6]  train-logloss:0.100454
## [7]  train-logloss:0.079603
```

```
## [8]  train-logloss:0.063409
## [9]  train-logloss:0.051677
## [10] train-logloss:0.043142
## [11] train-logloss:0.036314
## [12] train-logloss:0.031417
## [13] train-logloss:0.027332
## [14] train-logloss:0.024214
## [15] train-logloss:0.021765
## [16] train-logloss:0.019668
## [17] train-logloss:0.018136
## [18] train-logloss:0.017018
## [19] train-logloss:0.015889
## [20] train-logloss:0.015064
## [21] train-logloss:0.014264
## [22] train-logloss:0.013495
## [23] train-logloss:0.012902
## [24] train-logloss:0.012398
## [25] train-logloss:0.012031
## [26] train-logloss:0.011470
## [27] train-logloss:0.011155
## [28] train-logloss:0.010884
## [29] train-logloss:0.010669
## [30] train-logloss:0.010516
## [31] train-logloss:0.010286
## [32] train-logloss:0.010070
## [33] train-logloss:0.009888
## [34] train-logloss:0.009472
## [35] train-logloss:0.009344
## [36] train-logloss:0.009233
## [37] train-logloss:0.009126
## [38] train-logloss:0.009031
## [39] train-logloss:0.008858
## [40] train-logloss:0.008718
## [41] train-logloss:0.008643
## [42] train-logloss:0.008585
## [43] train-logloss:0.008508
## [44] train-logloss:0.008420
## [45] train-logloss:0.008376
## [46] train-logloss:0.008337
## [47] train-logloss:0.008297
## [48] train-logloss:0.008261
## [49] train-logloss:0.008229
## [50] train-logloss:0.008191
## [51] train-logloss:0.008161
## [52] train-logloss:0.008129
## [53] train-logloss:0.008096
## [54] train-logloss:0.008063
## [55] train-logloss:0.008033
## [56] train-logloss:0.008004
## [57] train-logloss:0.007975
## [58] train-logloss:0.007947
## [59] train-logloss:0.007919
## [60] train-logloss:0.007893
## [61] train-logloss:0.007865
```

```
## [62] train-logloss:0.007840
## [63] train-logloss:0.007815
## [64] train-logloss:0.007789
## [65] train-logloss:0.007763
## [66] train-logloss:0.007739
## [67] train-logloss:0.007715
## [68] train-logloss:0.007693
## [69] train-logloss:0.007670
## [70] train-logloss:0.007646
## [71] train-logloss:0.007622
## [72] train-logloss:0.007600
## [73] train-logloss:0.007578
## [74] train-logloss:0.007557
## [75] train-logloss:0.007537
## [76] train-logloss:0.007514
## [77] train-logloss:0.007496
## [78] train-logloss:0.007474
## [79] train-logloss:0.007456
## [80] train-logloss:0.007435
## [81] train-logloss:0.007418
## [82] train-logloss:0.007398
## [83] train-logloss:0.007380
## [84] train-logloss:0.007361
## [85] train-logloss:0.007343
## [86] train-logloss:0.007323
## [87] train-logloss:0.007306
## [88] train-logloss:0.007288
## [89] train-logloss:0.007271
## [90] train-logloss:0.007254
## [91] train-logloss:0.007236
## [92] train-logloss:0.007220
## [93] train-logloss:0.007203
## [94] train-logloss:0.007188
## [95] train-logloss:0.007173
## [96] train-logloss:0.007158
## [97] train-logloss:0.007141
## [98] train-logloss:0.007125
## [99] train-logloss:0.007110
## [100]    train-logloss:0.007095
```

```r
# Make predictions on test set
y_pred_rf <- predict(rf_model, X_test)
y_pred_xgb <- predict(xgb_model, as.matrix(X_test))
y_pred_xgb <- ifelse(y_pred_xgb > 0.5, 1, 0)

# Specify the levels explicitly
levels <- c(0, 1)

# Convert to factors with specified levels
y_pred_rf <- factor(y_pred_rf, levels = levels)
y_pred_xgb <- factor(y_pred_xgb, levels = levels)
y_test <- factor(y_test, levels = levels)

# Create confusion matrices
```

```r
cm_rf <- confusionMatrix(y_pred_rf, y_test, positive = "1")
print(cm_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 136   0
##          1   0   0
##
##                Accuracy : 1
##                  95% CI : (0.9732, 1)
##     No Information Rate : 1
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : NaN
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : NA
##             Specificity :  1
##          Pos Pred Value : NA
##          Neg Pred Value : NA
##              Prevalence :  0
##          Detection Rate :  0
##    Detection Prevalence :  0
##       Balanced Accuracy : NA
##
##        'Positive' Class : 1
##
```

```r
cm_xgb <- confusionMatrix(y_pred_xgb, y_test, positive = "1")
print(cm_xgb)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 135   0
##          1   1   0
##
##                Accuracy : 0.9926
##                  95% CI : (0.9597, 0.9998)
##     No Information Rate : 1
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity :       NA
##             Specificity : 0.992647
```

```
##          Pos Pred Value :       NA
##          Neg Pred Value :       NA
##              Prevalence : 0.000000
##          Detection Rate : 0.000000
##    Detection Prevalence : 0.007353
##       Balanced Accuracy :       NA
##
##        'Positive' Class : 1
##
```