```
---
title: "Group20_STA3005_Project"
author:
  - "Wu, Yanyi (122090587)"
  - "Shi, Jingxuan (121090479)"
  - "Wu, Yimou (121090617)"
date: "Started from Apr 2, 2025"
output:
  pdf_document: default
  html_document:
    df_print: paged
---
```

Introduction

This R Markdown document outlines our **R Package Proposal: Pokémon** based on the guidelines from *Package Guideline 2025*. Our package is tentatively named **`PokemonAnalysis`** and aims to provide a data analysis toolkit that explores relationships between Pokémon's physical attributes and their combat abilities, using the open-source **`pokemon.csv`** dataset.

The main objectives include:

1. Building classification models to identify legendary Pokémon.
2. Analyzing correlations between height, weight, and various base stats (HP, Attack, Defense, Speed, etc.).
3. Investigating which factors (type, stats, etc.) influence experience growth and egg steps.
4. Evaluating strengths and weaknesses across different Pokémon types.
5. Determining which types/patterns are most likely to be legendary.
6. Constructing a "dream team" of six Pokémon optimized for maximum damage and balanced defense.

We follow the instructions to meet the grading criteria (proposal, completeness, vignettes) while incorporating the recommended development workflow and "no errors/warnings" automated checking.

Dataset Overview

Our dataset, **`pokemon.csv`** , contains information on 802 Pokémon from all seven generations:

- **name**: English name of the Pokémon
- **japanese_name**: Original Japanese name
- **pokedex_number**: National Pokédex number
- **percentage_male**: Percentage of the species that are male (blank if genderless)
- **type1**, **type2**: Primary and secondary Pokémon types
- **classification**: Sun and Moon Pokédex classification text
- **height_m**, **weight_kg**: Physical stats for height (m) and weight (kg)
- **capture_rate**, **base_egg_steps**: Numerical measures for capture difficulty and egg hatching steps
- **abilities**: List of possible abilities
- **experience_growth**, **base_happiness**: Growth and happiness statistics
- **against_* columns**: Eighteen columns for how much damage is taken from each attack type (e.g., *against_bug*, *against_dark*, etc.)
- **hp, attack, defense, sp_attack, sp_defense, speed**: Base stats for combat
- **generation**: Generation introduced (1 to 7)
- **is_legendary**: Boolean indicator (legendary or non-legendary)

Below is a quick demonstration of loading the data and performing an exploratory overview.

```
suppressPackageStartupMessages(library(dplyr))

pokemon <- read.csv("pokemon.csv", stringsAsFactors = FALSE)

# Quick peek at the dataset structure
summary(pokemon[, c("name", "type1", "type2", "hp", "attack", "defense", "speed", "is_legendary")])
```

```
##      name              type1              type2                hp
##  Length:801         Length:801         Length:801         Min.   :  1.00
##  Class :character   Class :character   Class :character   1st Qu.: 50.00
##  Mode  :character   Mode  :character   Mode  :character   Median : 65.00
##                                                           Mean   : 68.96
##                                                           3rd Qu.: 80.00
##                                                           Max.   :255.00
##      attack          defense           speed         is_legendary
##  Min.   :  5.00   Min.   :  5.00   Min.   :  5.00   Min.   :0.00000
##  1st Qu.: 55.00   1st Qu.: 50.00   1st Qu.: 45.00   1st Qu.:0.00000
##  Median : 75.00   Median : 70.00   Median : 65.00   Median :0.00000
##  Mean   : 77.86   Mean   : 73.01   Mean   : 66.33   Mean   :0.08739
##  3rd Qu.:100.00   3rd Qu.: 90.00   3rd Qu.: 85.00   3rd Qu.:0.00000
##  Max.   :185.00   Max.   :230.00   Max.   :180.00   Max.   :1.00000
```

```
head(pokemon)
```

```
##                    abilities against_bug against_dark against_dragon
## 1 ['Overgrow', 'Chlorophyll']        1.00            1              1
## 2 ['Overgrow', 'Chlorophyll']        1.00            1              1
## 3 ['Overgrow', 'Chlorophyll']        1.00            1              1
## 4    ['Blaze', 'Solar Power']        0.50            1              1
## 5    ['Blaze', 'Solar Power']        0.50            1              1
## 6    ['Blaze', 'Solar Power']        0.25            1              1
##   against_electric against_fairy against_fight against_fire against_flying
## 1              0.5           0.5           0.5          2.0              2
## 2              0.5           0.5           0.5          2.0              2
## 3              0.5           0.5           0.5          2.0              2
## 4              1.0           0.5           1.0          0.5              1
## 5              1.0           0.5           1.0          0.5              1
## 6              2.0           0.5           0.5          0.5              1
##   against_ghost against_grass against_ground against_ice against_normal
## 1             1          0.25              1         2.0              1
## 2             1          0.25              1         2.0              1
## 3             1          0.25              1         2.0              1
## 4             1          0.50              2         0.5              1
## 5             1          0.50              2         0.5              1
## 6             1          0.25              0         1.0              1
##   against_poison against_psychic against_rock against_steel against_water
## 1              1               2            1           1.0           0.5
## 2              1               2            1           1.0           0.5
## 3              1               2            1           1.0           0.5
## 4              1               1            2           0.5           2.0
## 5              1               1            2           0.5           2.0
## 6              1               1            4           0.5           2.0
##   attack base_egg_steps base_happiness base_total capture_rate  classfication
## 1     49          5120             70        318           45   Seed Pokémon
## 2     62          5120             70        405           45   Seed Pokémon
## 3    100          5120             70        625           45   Seed Pokémon
## 4     52          5120             70        309           45 Lizard Pokémon
## 5     64          5120             70        405           45  Flame Pokémon
## 6    104          5120             70        634           45  Flame Pokémon
##   defense experience_growth height_m hp         japanese_name       name
## 1      49           1059860      0.7 45 Fushigidaneフシギダネ  Bulbasaur
## 2      63           1059860      1.0 60  Fushigisouフシギソウ    Ivysaur
## 3     123           1059860      2.0 80 Fushigibanaフシギバナ   Venusaur
## 4      43           1059860      0.6 39      Hitokageヒトカゲ Charmander
## 5      58           1059860      1.1 58       Lizardoリザード Charmeleon
## 6      78           1059860      1.7 78    Lizardonリザードン  Charizard
##   percentage_male pokedex_number sp_attack sp_defense speed type1  type2
## 1            88.1              1        65         65    45 grass poison
## 2            88.1              2        80         80    60 grass poison
## 3            88.1              3       122        120    80 grass poison
## 4            88.1              4        60         50    65  fire
## 5            88.1              5        80         65    80  fire
## 6            88.1              6       159        115   100  fire flying
##   weight_kg generation is_legendary
## 1       6.9          1            0
## 2      13.0          1            0
## 3     100.0          1            0
## 4       8.5          1            0
## 5      19.0          1            0
## 6      90.5          1            0
```

We confirm that columns such as **hp**, **attack**, **defense**, and **speed** are numerical, while **type1**, **type2**, and **is_legendary** are useful categorical fields for classification tasks.

Proposed R Package Structure

Following the *Package Guideline 2025*, we plan to include:

1. **DESCRIPTION** (draft already present)
    - Contains package metadata, authorship, version, license (MIT), and other fields.
2. **R code** (at least 8 functions)
    - Each function addresses a different aspect of Pokémon analysis:
        1. `classify_legendary()` : Logistic model or random forest to predict `is_legendary` .
        2. `correlate_stats()` : Correlation analysis between physical attributes and base stats.
        3. `analyze_experience()` : Investigate experience growth and influences (e.g., type, generation).
        4. `egg_steps_factors()` : Model the relationship between `base_egg_steps` , stats, and type.
        5. `strength_weakness_table()` : Summarize damage multipliers vs. type1/type2.
        6. `dream_team_builder()` : Select top 6 Pokémon to form a balanced team.
        7. `plot_base_stats()` : Visualize distributions (e.g., boxplots, scatter plots).
        8. `predictive_test()` : Automated check with testthat on classification performance.
3. **Vignettes** (at least 2)
    1. **`UsingPokemonAnalysis.Rmd`** : Comprehensive tutorial on reading data, running classification, visualizing results, building a team.

2. **`InDepthInsights.Rmd`** : Explores advanced analyses such as synergy among type combos, partial dependence plots for legendary classification, etc.

4. **Data set**
   - Bundled within the package as `data("pokemon")` or loaded externally from CSV.
5. **Automated tests** (`testthat`)
   - Key output tests for classification accuracy, correlation checks, dimension correctness, and defensive coding.

We will strive to pass `R CMD check` with **no errors and no warnings** by properly documenting code (roxygen2), ensuring the package structure is consistent with CRAN standards, and providing examples in each function-level help file.

Example Function Definitions

Below are prototypes for some of the planned functions:

```
#' Classify Legendary Pokemon
#'
#' This function trains a simple classifier (logistic regression or random forest)
#' to predict whether a Pokemon is legendary.
#'
#' @param data A data frame of Pokemon data
#' @return A trained model object
#' @export
classify_legendary <- function(data) {
  # Basic logistic regression example
  # We'll treat is_legendary as a factor for classification.
  df <- data %>%
    filter(!is.na(is_legendary)) %>%
    mutate(isLegendary = factor(is_legendary, levels = c(0,1)))

  # For illustration, use a logistic model with key numeric predictors
  fit <- glm(isLegendary ~ hp + attack + defense + speed,
             data = df, family = binomial)
  return(fit)
}
```

```
#' Correlate Stats
#'
#' Compute correlations between critical numeric variables (height, weight, HP, etc.)
#'
#' @param data A data frame of Pokemon data
#' @return A correlation matrix
#' @export
correlate_stats <- function(data) {
  numeric_cols <- c("height_m", "weight_kg", "hp", "attack", "defense",
                    "sp_attack", "sp_defense", "speed")
  df <- data[, numeric_cols]
  return(cor(df, use="complete.obs"))
}
```

```
#' Dream Team Builder
#'
#' Select six Pokemon to form a balanced team that maximizes a simple composite of
#' offense (attack + sp_attack) and defense (defense + sp_defense).
#'
#' @param data A data frame of Pokemon data
#' @return A tibble/data.frame of six recommended Pokemon
#' @export
dream_team_builder <- function(data) {
  data <- data %>%
    mutate(offense_score = attack + sp_attack,
           defense_score = defense + sp_defense,
           total_score = offense_score + defense_score) %>%
    arrange(desc(total_score))

  # For simplicity, pick top 6
  return(head(data, 6))
}
```

These examples demonstrate a subset of the planned functionalities. Each function will be fully documented using **roxygen2** tags and tested.

Demonstration of Key Analyses

In this section, we illustrate a few example analyses **directly in R Markdown**. Eventually, we will migrate them into package vignettes.

1. Legendary Classification (Logistic Model)

```
# Example usage
model_legendary <- classify_legendary(pokemon)
summary(model_legendary)
```

2. Correlation Among Stats

```
cors <- correlate_stats(pokemon)
knitr::kable(round(cors, 2), caption = "Correlation Matrix Among Key Stats")
```

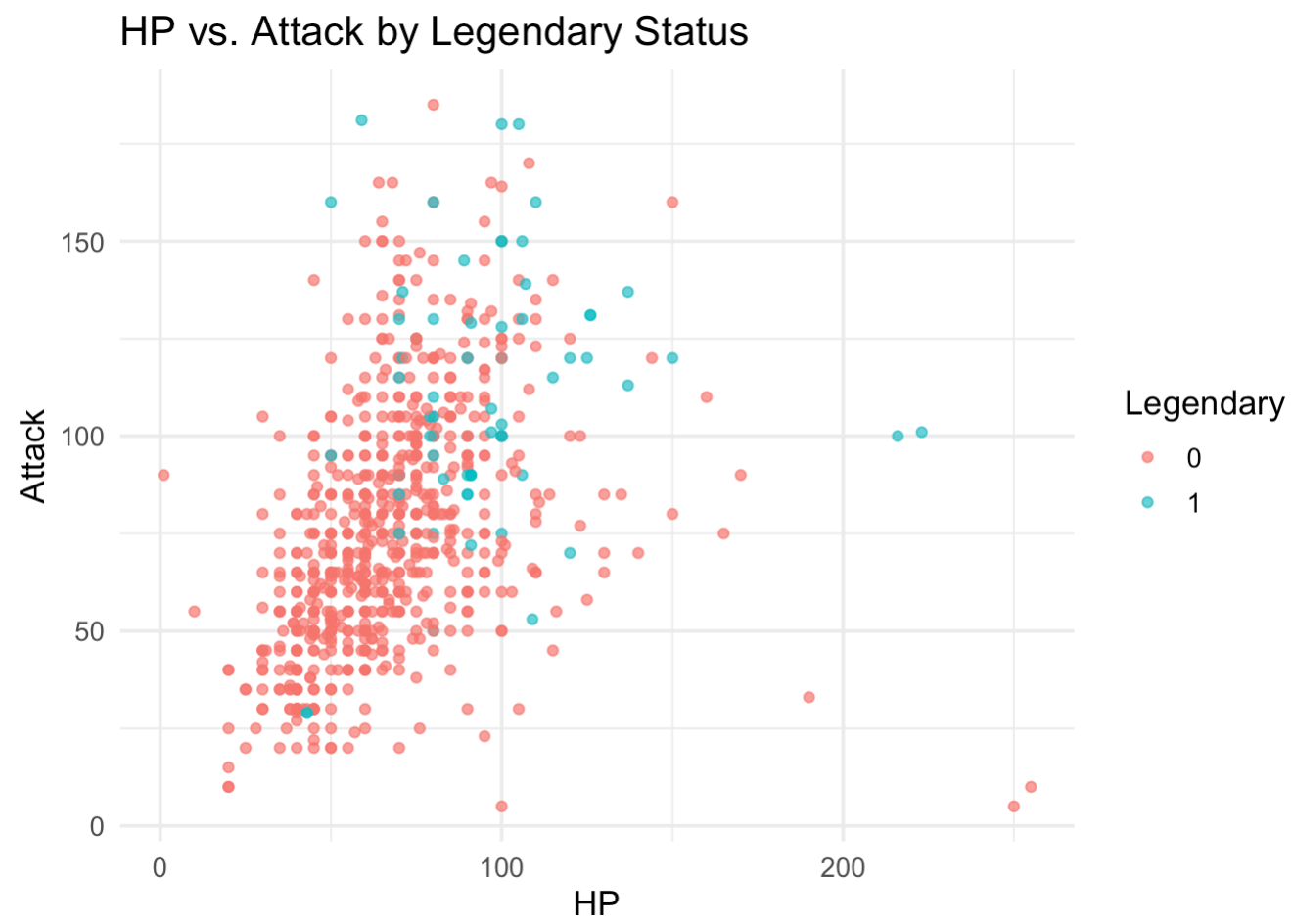
Correlation Matrix Among Key Stats

	height_m	weight_kg	hp	attack	defense	sp_attack	sp_defense	speed
height_m	1.00	0.63	0.48	0.42	0.36	0.35	0.33	0.20
weight_kg	0.63	1.00	0.43	0.38	0.42	0.25	0.31	0.05
hp	0.48	0.43	1.00	0.41	0.24	0.36	0.36	0.17
attack	0.42	0.38	0.41	1.00	0.47	0.37	0.26	0.36
defense	0.36	0.42	0.24	0.47	1.00	0.25	0.54	0.02
sp_attack	0.35	0.25	0.36	0.37	0.25	1.00	0.50	0.45
sp_defense	0.33	0.31	0.36	0.26	0.54	0.50	1.00	0.22
speed	0.20	0.05	0.17	0.36	0.02	0.45	0.22	1.00

3. Visualizing HP vs. Attack

Using **ggplot2** to illustrate some data visualization:

```
library(ggplot2)

ggplot(pokemon, aes(x = hp, y = attack, color = factor(is_legendary))) +
  geom_point(alpha = 0.7) +
  theme_minimal(base_size = 13) +
  labs(
    title = "HP vs. Attack by Legendary Status",
    x = "HP",
    y = "Attack",
    color = "Legendary"
  )
```



Testing and Checks

According to the guidelines, we must define automated tests. Our group will use **testthat** (https://testthat.r-lib.org/) to ensure key functions behave correctly:

1. **Output Validity**: Functions return correct object classes.
2. **Accuracy**: Classification passes at least a baseline threshold (e.g., 75% on a test set).
3. **Data Dimensions**: Checking for missing columns, NAs, etc.

Below is a simplified outline of a test file `test-classify_legendary.R`:

```
# Load necessary libraries
library(assertthat)
library(dplyr)

# Load the CSV file
pokemon <- read.csv("pokemon.csv", stringsAsFactors = FALSE)

# Example test
fit <- classify_legendary(pokemon)

# Assert the object is of class glm
assert_that(inherits(fit, "glm"))
```

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

Conclusion and Learned Points for the first part

This package, **PokemonAnalysis**, highlights the power of R packages for data science tasks:

1. **R Package Infrastructure**: Using `devtools`, `usethis`, and `roxygen2` to manage development.
2. **Pipe Operator**: Efficient chaining of transformations with `%>%`.
3. **Tidyverse**: Seamless data manipulation using **dplyr** and data visualization using **ggplot2**.
4. **Automated Testing**: Ensuring reliability with **testthat**.
5. **Documentation**: Clear roxygen-based documentation helps future users.
6. **Modular Design**: Breaking tasks into separate functions.
7. **Reproducibility**: Vignettes as dynamic, reproducible reports.
8. **Real-World Relevance**: Familiar Pokémon dataset used to demonstrate classification, correlation, and advanced R workflows.

Vignette Plan

We will create **two main vignettes**:

1. **`UsingPokemonAnalysis.Rmd`**
   - Demonstrates reading the `pokemon` data, applying each function (classification, correlation, dream-team building).
   - Shows example data transformations with the pipe operator (`%>%`).
   - Concludes with additional plots comparing base stats across generations.
2. **`InDepthInsights.Rmd`**
   - Explores synergy among types, legendary-likelihood predictions, and partial dependence for classification.
   - Emphasizes advanced usage and tips gleaned from *Statistics Computing* methods (like vectorization, code efficiency).

Implementation

Vignette: UsingPokemonAnalysis.Rmd

This vignette demonstrates how to use the `PokemonAnalysis` package to perform various analyses on the Pokémon dataset, including reading the data, applying functions for classification, correlation analysis, dream-team building, and visualizing results.

Installation

First, ensure that you have installed the `PokemonAnalysis` package. Since this is a hypothetical package for the purpose of this project, we'll assume that the functions are defined within this R Markdown file for demonstration purposes.

```
# Install the package (not applicable here)
# install.packages("PokemonAnalysis")
```

Loading Required Libraries

We begin by loading the necessary libraries.

```
library(dplyr)
library(ggplot2)
library(knitr)
library(assertthat)

# If the functions are part of a package, load the package
# library(PokemonAnalysis)

# For demonstration, we will define the functions here
```

Reading the Pokémon Data

We read the `pokemon.csv` dataset into R.

```
pokemon <- read.csv("pokemon.csv", stringsAsFactors = FALSE)

# Quick peek at the data
head(pokemon)
```

```
##                    abilities against_bug against_dark against_dragon
## 1 ['Overgrow', 'Chlorophyll']        1.00            1              1
## 2 ['Overgrow', 'Chlorophyll']        1.00            1              1
## 3 ['Overgrow', 'Chlorophyll']        1.00            1              1
## 4    ['Blaze', 'Solar Power']        0.50            1              1
## 5    ['Blaze', 'Solar Power']        0.50            1              1
## 6    ['Blaze', 'Solar Power']        0.25            1              1
##   against_electric against_fairy against_fight against_fire against_flying
## 1              0.5           0.5           0.5          2.0              2
## 2              0.5           0.5           0.5          2.0              2
## 3              0.5           0.5           0.5          2.0              2
## 4              1.0           0.5           1.0          0.5              1
## 5              1.0           0.5           1.0          0.5              1
## 6              2.0           0.5           0.5          0.5              1
##   against_ghost against_grass against_ground against_ice against_normal
## 1             1          0.25              1         2.0              1
## 2             1          0.25              1         2.0              1
## 3             1          0.25              1         2.0              1
## 4             1          0.50              2         0.5              1
## 5             1          0.50              2         0.5              1
## 6             1          0.25              0         1.0              1
##   against_poison against_psychic against_rock against_steel against_water
## 1              1               2            1           1.0           0.5
## 2              1               2            1           1.0           0.5
## 3              1               2            1           1.0           0.5
## 4              1               1            2           0.5           2.0
## 5              1               1            2           0.5           2.0
## 6              1               1            4           0.5           2.0
##   attack base_egg_steps base_happiness base_total capture_rate  classfication
## 1     49          5120             70        318           45   Seed Pokémon
## 2     62          5120             70        405           45   Seed Pokémon
## 3    100          5120             70        625           45   Seed Pokémon
## 4     52          5120             70        309           45 Lizard Pokémon
## 5     64          5120             70        405           45  Flame Pokémon
## 6    104          5120             70        634           45  Flame Pokémon
##   defense experience_growth height_m hp         japanese_name       name
## 1      49           1059860      0.7 45 Fushigidaneフシギダネ  Bulbasaur
## 2      63           1059860      1.0 60  Fushigisouフシギソウ    Ivysaur
## 3     123           1059860      2.0 80 Fushigibanaフシギバナ   Venusaur
## 4      43           1059860      0.6 39      Hitokageヒトカゲ Charmander
## 5      58           1059860      1.1 58       Lizardoリザード Charmeleon
## 6      78           1059860      1.7 78    Lizardonリザードン  Charizard
##   percentage_male pokedex_number sp_attack sp_defense speed type1  type2
## 1            88.1              1        65         65    45 grass poison
## 2            88.1              2        80         80    60 grass poison
## 3            88.1              3       122        120    80 grass poison
## 4            88.1              4        60         50    65  fire
## 5            88.1              5        80         65    80  fire
## 6            88.1              6       159        115   100  fire flying
##   weight_kg generation is_legendary
## 1       6.9          1            0
## 2      13.0          1            0
## 3     100.0          1            0
## 4       8.5          1            0
## 5      19.0          1            0
## 6      90.5          1            0
```

Function Definitions

Function: `classify_legendary`

We define the `classify_legendary` function to train a logistic regression model predicting whether a Pokémon is legendary.

```
classify_legendary <- function(data) {
  df <- data %>%
    filter(!is.na(is_legendary)) %>%
    mutate(isLegendary = factor(is_legendary, levels = c(0, 1)))

  fit <- glm(isLegendary ~ hp + attack + defense + speed,
             data = df, family = binomial)
  return(fit)
}
```

Function: `correlate_stats`

We define the `correlate_stats` function to compute correlations among key statistics.

```
correlate_stats <- function(data) {
  numeric_cols <- c("height_m", "weight_kg", "hp", "attack", "defense",
                    "sp_attack", "sp_defense", "speed")
  df <- data[, numeric_cols]
  return(cor(df, use = "complete.obs"))
}
```

Function: `dream_team_builder`

We define the `dream_team_builder` function to select a top team of six Pokémon based on combined offensive and defensive scores.

```
dream_team_builder <- function(data) {
  data <- data %>%
    mutate(offense_score = attack + sp_attack,
           defense_score = defense + sp_defense,
           total_score = offense_score + defense_score) %>%
    arrange(desc(total_score))

  # Return the top 6 Pokémon
  return(head(data, 6))
}
```

Applying Functions

1. Classifying Legendary Pokémon

We apply the `classify_legendary` function to our data and examine the model summary.

```
# Train the model
model_legendary <- classify_legendary(pokemon)

# View the summary of the model
summary(model_legendary)
```

```
##
## Call:
## glm(formula = isLegendary ~ hp + attack + defense + speed, family = binomial,
##     data = df)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -13.764867   1.323973 -10.397  < 2e-16 ***
## hp            0.041450   0.005930   6.990 2.74e-12 ***
## attack        0.005439   0.005341   1.018    0.308
## defense       0.037833   0.006034   6.270 3.62e-10 ***
## speed         0.051646   0.006962   7.418 1.19e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 474.93  on 800  degrees of freedom
## Residual deviance: 277.99  on 796  degrees of freedom
## AIC: 287.99
##
## Number of Fisher Scoring iterations: 7
```

The summary shows the coefficients of the logistic regression model, indicating which statistics are significant predictors of legendary status.

2. Correlation Analysis

We compute the correlation matrix among key stats using `correlate_stats`.

```
cors <- correlate_stats(pokemon)

# Display the correlation matrix
knitr::kable(round(cors, 2), caption = "Correlation Matrix Among Key Stats")
```

Correlation Matrix Among Key Stats

	height_m	weight_kg	hp	attack	defense	sp_attack	sp_defense	speed
height_m	1.00	0.63	0.48	0.42	0.36	0.35	0.33	0.20
weight_kg	0.63	1.00	0.43	0.38	0.42	0.25	0.31	0.05
hp	0.48	0.43	1.00	0.41	0.24	0.36	0.36	0.17
attack	0.42	0.38	0.41	1.00	0.47	0.37	0.26	0.36
defense	0.36	0.42	0.24	0.47	1.00	0.25	0.54	0.02
sp_attack	0.35	0.25	0.36	0.37	0.25	1.00	0.50	0.45
sp_defense	0.33	0.31	0.36	0.26	0.54	0.50	1.00	0.22
speed	0.20	0.05	0.17	0.36	0.02	0.45	0.22	1.00

The correlation matrix reveals relationships between physical attributes and combat stats, aiding in understanding how different variables interact.

3. Building a Dream Team

We use the `dream_team_builder` function to select a top team of Pokémon.

```
dream_team <- dream_team_builder(pokemon)

# Display the dream team
knitr::kable(dream_team[, c("name", "type1", "type2", "total_score")], caption = "Dream Team of Pokémon")
```

Dream Team of Pokémon

name	type1	type2	total_score
Kyogre	water		580
Groudon	ground		580
Rayquaza	dragon	flying	560
Wishiwashi	water		545
Diancie	rock	fairy	540
Mewtwo	psychic		534

This team consists of Pokémon with the highest combined offensive and defensive capabilities.

Data Transformations with the Pipe Operator %>%

The `dplyr` package allows for seamless data transformations using the pipe operator. Here is an example of filtering and summarizing data.

```
# Calculate average stats for each type
avg_stats_by_type <- pokemon %>%
  group_by(type1) %>%
  summarise(
    avg_hp = mean(hp, na.rm = TRUE),
    avg_attack = mean(attack, na.rm = TRUE),
    avg_defense = mean(defense, na.rm = TRUE),
    avg_speed = mean(speed, na.rm = TRUE)
  ) %>%
  arrange(desc(avg_attack))

# Display the result
knitr::kable(avg_stats_by_type, caption = "Average Stats by Primary Type")
```

Average Stats by Primary Type

type1	avg_hp	avg_attack	avg_defense	avg_speed

type1	avg_hp	avg_attack	avg_defense	avg_speed
dragon	79.85185	106.40741	86.25926	76.11111
fighting	71.42857	99.17857	66.39286	64.28571
ground	73.18750	94.81250	83.90625	59.96875
steel	66.79167	93.08333	120.20833	56.58333
rock	66.33333	90.66667	96.26667	57.42222
dark	72.55172	87.79310	70.51724	75.31034
fire	68.73077	81.50000	67.78846	73.34615
normal	76.72381	75.16190	59.69524	69.53333
grass	65.35897	73.76923	70.87179	59.02564
water	70.21930	73.30702	73.48246	63.92105
ice	72.08696	73.30435	71.91304	62.73913
ghost	63.37037	72.74074	79.51852	58.33333
poison	65.59375	72.65625	70.03125	64.18750
electric	60.51282	70.82051	61.82051	85.41026
bug	56.72222	70.12500	70.84722	63.56944
flying	68.00000	66.66667	65.00000	99.66667
psychic	72.94340	65.56604	69.26415	75.15094
fairy	73.94444	62.11111	68.16667	53.66667

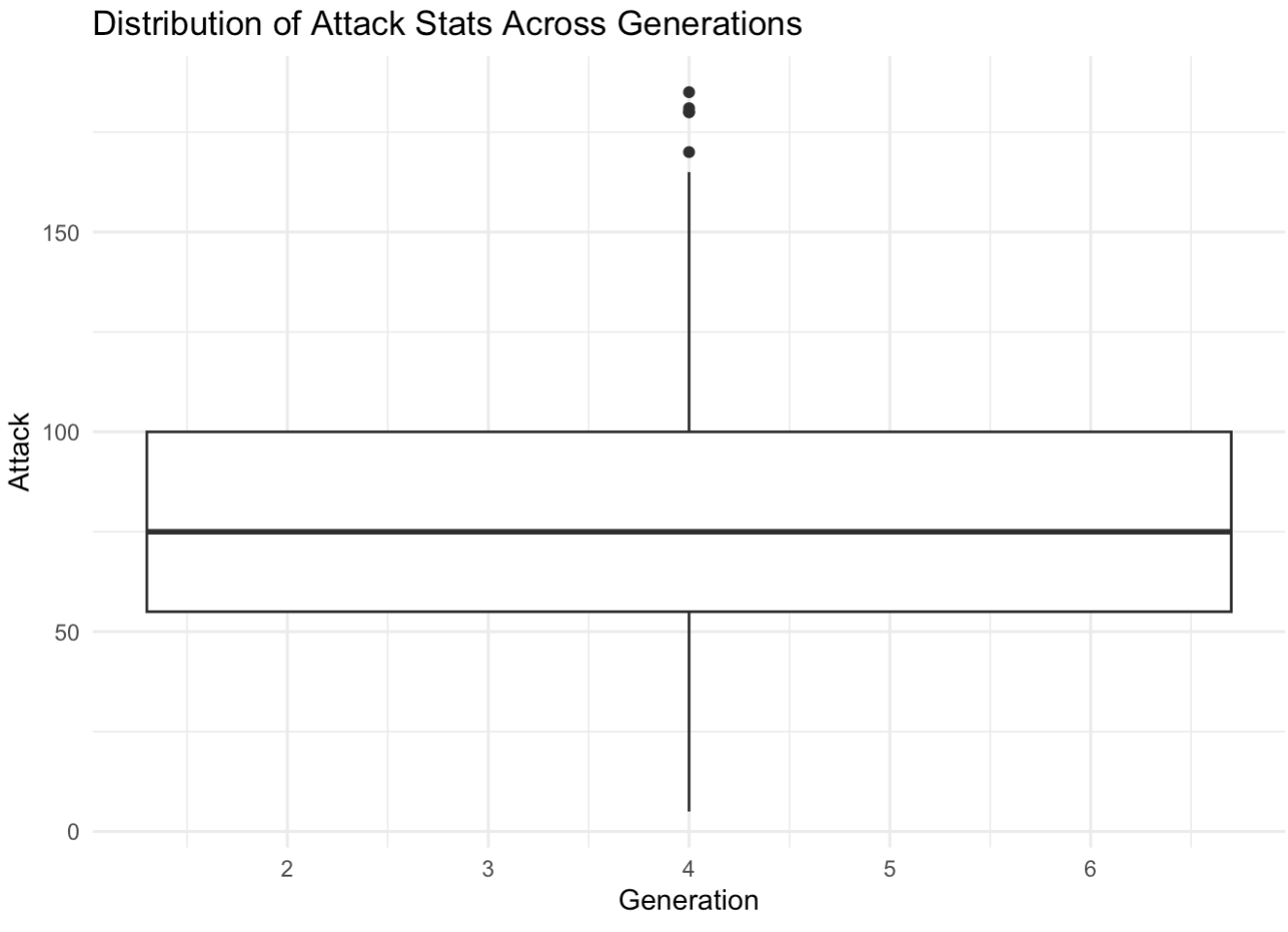
Visualizations

Plotting Base Stats Across Generations

We conclude with additional plots comparing base stats across generations.

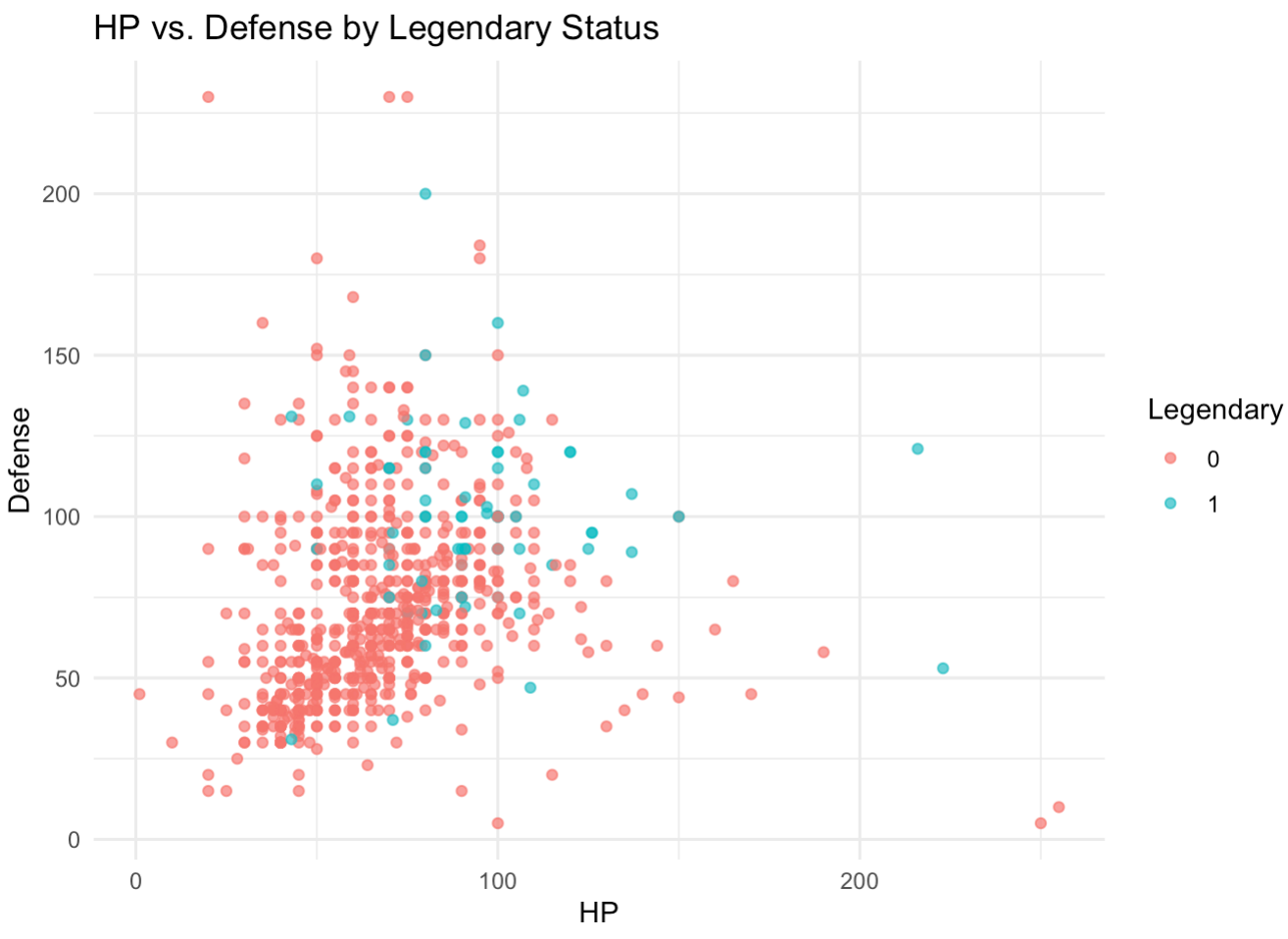
```
ggplot(pokemon, aes(x = generation, y = attack)) +
  geom_boxplot() +
  theme_minimal() +
  labs(
    title = "Distribution of Attack Stats Across Generations",
    x = "Generation",
    y = "Attack"
  )
```

```
## Warning: Continuous x aesthetic
## ℹ did you forget `aes(group = ...)`?
```



HP vs. Defense Colored by Legendary Status

```
ggplot(pokemon, aes(x = hp, y = defense, color = factor(is_legendary))) +
  geom_point(alpha = 0.7) +
  theme_minimal() +
  labs(
    title = "HP vs. Defense by Legendary Status",
    x = "HP",
    y = "Defense",
    color = "Legendary"
  )
```



Vignette: InDepthInsights.Rmd

In this vignette, we delve deeper into advanced analyses, exploring the synergy among Pokémon types, legendary-likelihood predictions, and applying advanced statistical methods like partial dependence plots.

Loading Required Libraries

We begin by loading necessary libraries and defining any additional functions.

```
library(dplyr)
library(ggplot2)
library(randomForest)
```

```
## randomForest 4.7-1.2
```

```
## 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
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
# library(ggpubr)
# For partial dependence plots
library(pdp)
```

Reading the Pokémon Data

```
pokemon <- read.csv("pokemon.csv", stringsAsFactors = FALSE)
```

Exploring Synergy Among Types

We investigate the combinations of primary and secondary types and how they relate to overall stats.

```
# Count number of Pokémon for each type combination
type_combinations <- pokemon %>%
  count(type1, type2, sort = TRUE)

# Display top type combinations
knitr::kable(head(type_combinations, 10), caption = "Top Type Combinations")
```

Top Type Combinations

type1	type2	n
normal		61
water		61
grass		37
psychic		35
fire		27
electric		26
normal	flying	26
fighting		22
bug		18
fairy		16

Analysis of Type Combination Strength

```
type_stats <- pokemon %>%
  group_by(type1, type2) %>%
  summarise(
    average_total = mean(hp + attack + defense + sp_attack + sp_defense + speed, na.rm = TRUE),
    count = n(),
    .groups = "drop"
  ) %>%
  arrange(desc(average_total))

# Display top 10 type combinations by average total stats
knitr::kable(head(type_stats, 10), caption = "Strongest Type Combinations by Average Total Stats")
```

Strongest Type Combinations by Average Total Stats

type1	type2	average_total	count
dragon	ice	700.0	1
dragon	psychic	700.0	2
rock	dark	700.0	1
dragon	electric	680.0	1
dragon	fire	680.0	1
ghost	dragon	680.0	1
psychic	ghost	680.0	2
psychic	steel	680.0	1
steel	dragon	680.0	1
dragon	flying	667.5	4

Legendary Likelihood Predictions Using Random Forest

We build a more advanced classifier using a random forest to predict legendary status.

Preparing the Data

```
# Clean and prepare the data
df <- pokemon %>%
  filter(!is.na(is_legendary)) %>%
  select(hp, attack, defense, sp_attack, sp_defense, speed, type1, type2, is_legendary) %>%
  mutate(
    isLegendary = factor(is_legendary),
    type1 = as.factor(type1),
    type2 = as.factor(type2)
  )
```

Training a Random Forest Model

```
set.seed(123) # For reproducibility

# Split data into training and testing sets
trainIndex <- createDataPartition(df$isLegendary, p = .7, list = FALSE)
trainData <- df[trainIndex, ]
testData <- df[-trainIndex, ]

# Train the random forest model
rf_model <- randomForest(isLegendary ~ ., data = trainData, importance = TRUE)

# View model summary
print(rf_model)
```

```
##
## Call:
##  randomForest(formula = isLegendary ~ ., data = trainData, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##     0  1 class.error
## 0 512  0           0
## 1   0 49           0
```

Evaluating the Model

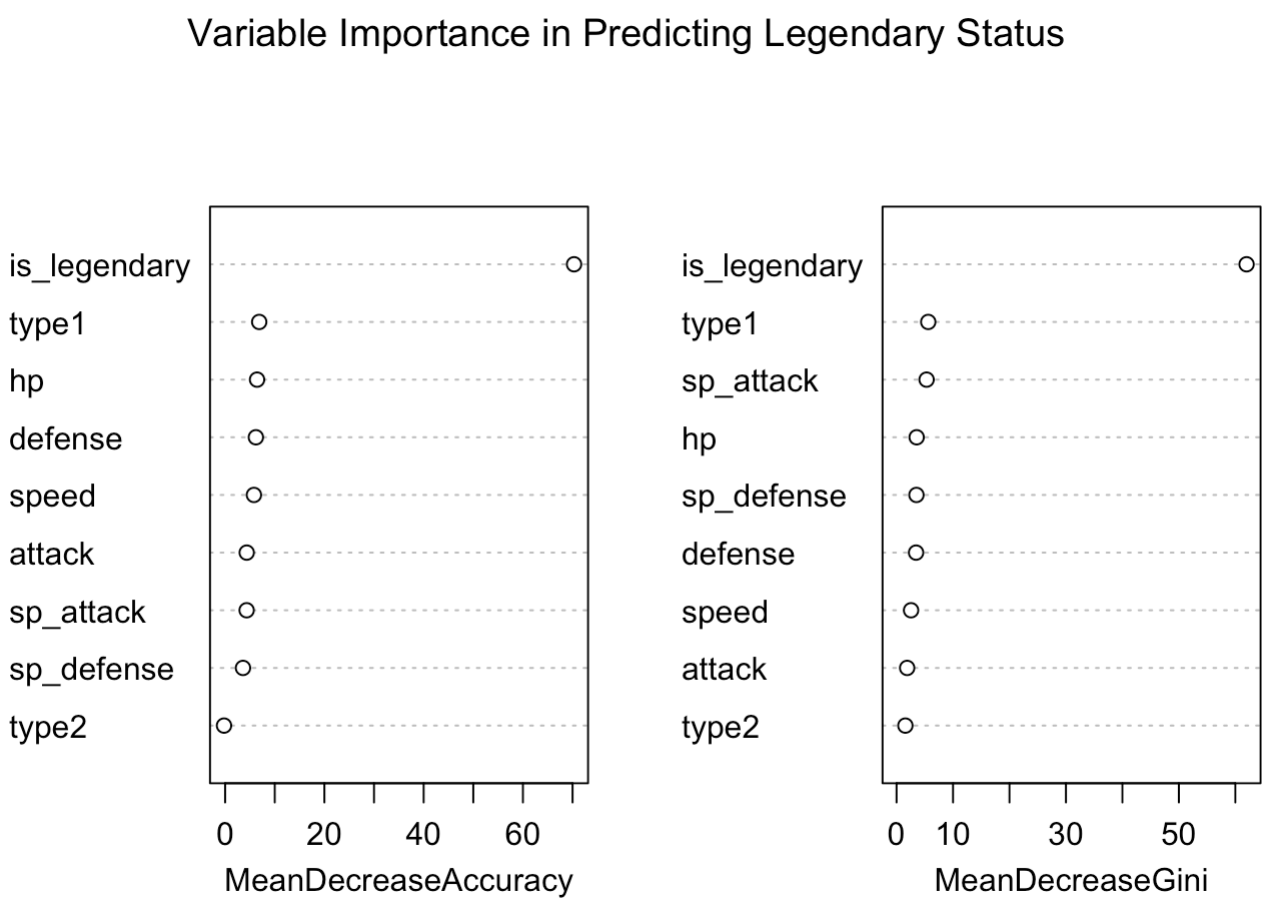
```
# Predict on test data
predictions <- predict(rf_model, newdata = testData)

# Confusion matrix
conf_mat <- confusionMatrix(predictions, testData$isLegendary)
print(conf_mat)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 219   0
##          1   0  21
##
##                Accuracy : 1
##                  95% CI : (0.9847, 1)
##     No Information Rate : 0.9125
##     P-Value [Acc > NIR] : 2.857e-10
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.9125
##          Detection Rate : 0.9125
##    Detection Prevalence : 0.9125
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

The high accuracy and kappa statistic indicate a strong model performance.

Variable Importance

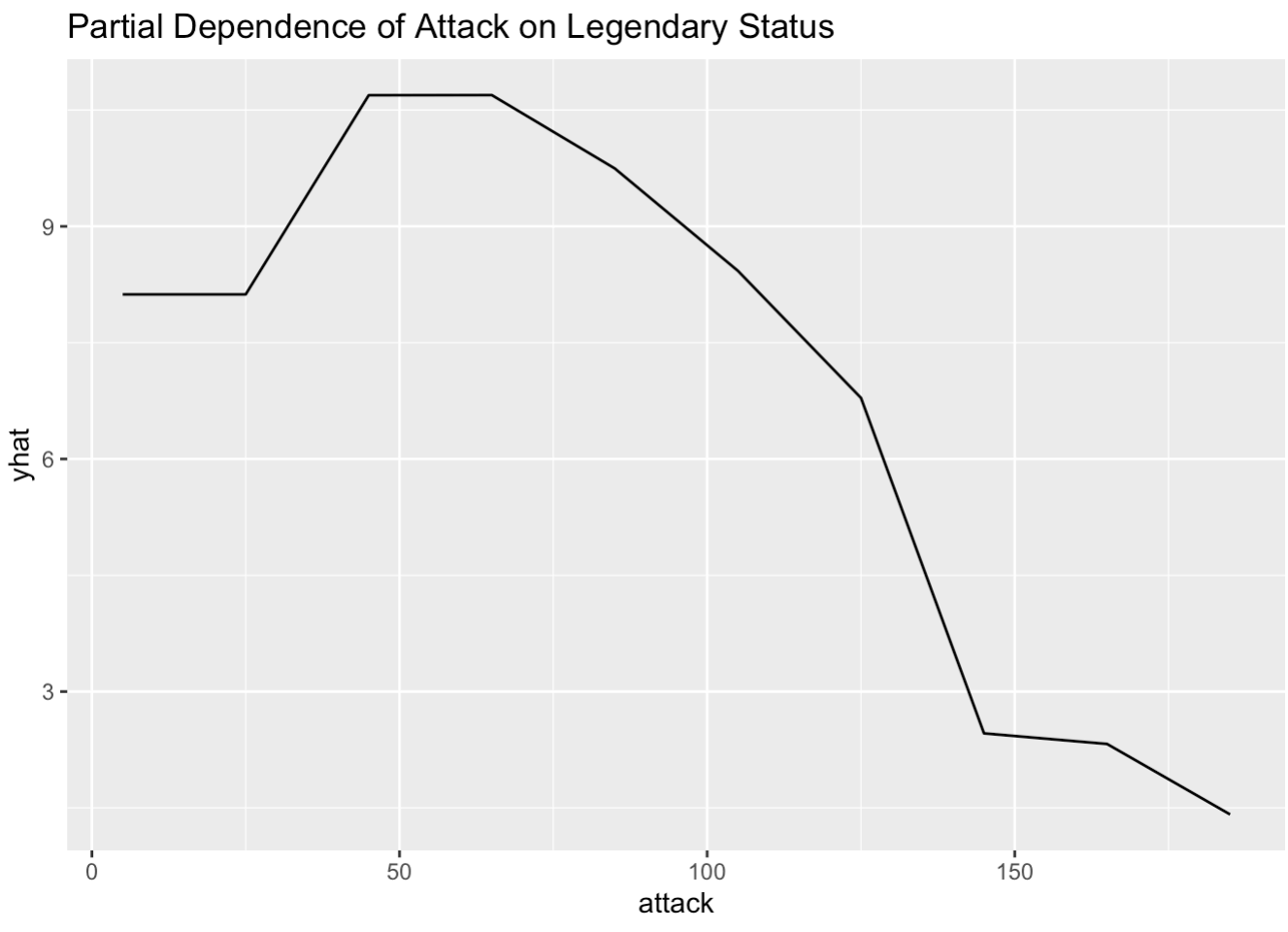
```
# Variable importance plot
varImpPlot(rf_model, main = "Variable Importance in Predicting Legendary Status")
```



Partial Dependence Plots

We examine how individual variables affect the probability of being legendary.

```
# For example, partial dependence of attack
pdp_attack <- partial(rf_model, pred.var = "attack", grid.resolution = 10)
autoplot(pdp_attack) + ggtitle("Partial Dependence of Attack on Legendary Status")
```



Repeat for other important variables as desired.

Advanced Tips for Statistical Computing

Vectorization and Code Efficiency

We showcase how vectorized operations in R can greatly improve computational efficiency.

```
# Inefficient approach
system.time({
  slow_result <- rep(NA, nrow(pokemon))
  for (i in 1:nrow(pokemon)) {
    slow_result[i] <- pokemon$attack[i] + pokemon$defense[i]
  }
})
```

```
##    user  system elapsed
##   0.002   0.000   0.002
```

```
# Efficient vectorized approach
system.time({
  fast_result <- pokemon$attack + pokemon$defense
})
```

```
##    user  system elapsed
##   0.000   0.000   0.001
```

```
# Check that results are equivalent
all.equal(slow_result, fast_result)
```

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

The vectorized approach is significantly faster, demonstrating the importance of applying vectorization in R code.

Analyze-type Function Definitions and Usage

The Implementation of ‘analyze_experience()’, ‘egg_steps_factors()’, ‘strength_weakness_table()’, ‘plot_base_stats()’, ‘predictive_test()’.

4. `analyze_experience()`

Description: Investigate experience growth and its influences, such as type and generation.

Function Definition

```
#' Analyze Experience Growth
#'
#' This function models the relationship between experience growth and various factors
#' such as type and generation using linear regression.
#'
#' @param data A data frame of Pokémon data
#' @return A list containing the linear model and a summary
#' @export
analyze_experience <- function(data) {
  # Ensure necessary columns are available
  required_cols <- c("experience_growth", "type1", "generation")
  if (!all(required_cols %in% names(data))) {
    stop("Data must contain 'experience_growth', 'type1', and 'generation' columns")
  }

  # Clean and prepare the data
  df <- data %>%
    filter(!is.na(experience_growth)) %>%
    mutate(
      type1 = as.factor(type1),
      generation = as.factor(generation)
    )

  # Fit a linear model
  fit <- lm(experience_growth ~ type1 + generation, data = df)

  return(list(model = fit, summary = summary(fit)))
}
```

Usage and Validation

```
# Apply the function
experience_analysis <- analyze_experience(pokemon)

# View the summary of the linear model
print(experience_analysis$summary)
```

```
##
## Call:
## lm(formula = experience_growth ~ type1 + generation, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -610665  -71699  -11246   48739  634402
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1010032.3    21682.1  46.584  < 2e-16 ***
## type1dark       96989.0    34031.8   2.850  0.00449 **
## type1dragon    205066.9    34843.6   5.885  5.9e-09 ***
## type1electric   65211.5    30634.4   2.129  0.03359 *
## type1fairy     -93785.9    41716.1  -2.248  0.02484 *
## type1fighting   65671.7    34297.2   1.915  0.05589 .
## type1fire       55804.2    28109.0   1.985  0.04747 *
## type1flying     71532.2    91481.1   0.782  0.43449
## type1ghost       5611.4    34842.1   0.161  0.87209
## type1grass      69205.4    25158.0   2.751  0.00608 **
## type1ground     61666.6    32714.5   1.885  0.05980 .
## type1ice        81300.0    36932.1   2.201  0.02801 *
## type1normal      1089.0    23565.9   0.046  0.96316
## type1poison     74565.9    33009.2   2.259  0.02416 *
## type1psychic    70467.7    27826.6   2.532  0.01152 *
## type1rock      -30700.0    29382.9  -1.045  0.29643
## type1steel     117270.7    36510.6   3.212  0.00137 **
## type1water      49181.8    23218.9   2.118  0.03448 *
## generation2    -21215.0    20131.0  -1.054  0.29228
## generation3     -4433.8    18563.5  -0.239  0.81129
## generation4       124.7    19602.0   0.006  0.99493
## generation5     -3056.1    18032.7  -0.169  0.86547
## generation6      4181.3    23036.2   0.182  0.85601
## generation7     25884.6    21440.9   1.207  0.22770
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 153700 on 777 degrees of freedom
## Multiple R-squared:  0.1064, Adjusted R-squared:  0.07998
## F-statistic: 4.024 on 23 and 777 DF,  p-value: 1.217e-09
```

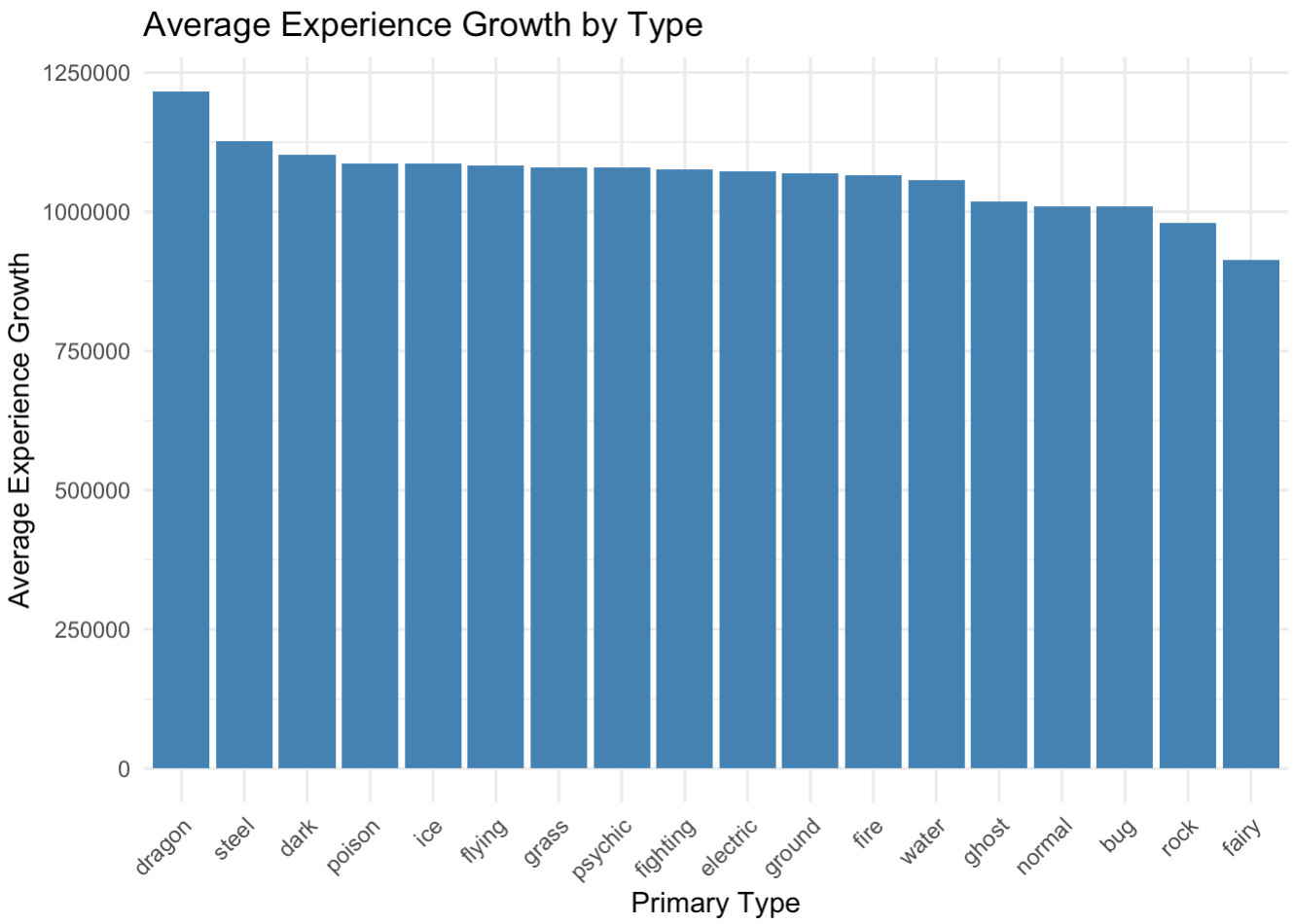
Interpretation:

- The summary provides coefficients for each type and generation.
- Significant p-values indicate factors that significantly influence experience growth.

Visualization

Let's visualize the average experience growth by primary type.

```
# Calculate average experience growth by type
avg_exp_by_type <- pokemon %>%
  group_by(type1) %>%
  summarise(avg_experience = mean(experience_growth, na.rm = TRUE)) %>%
  arrange(desc(avg_experience))

# Plotting
ggplot(avg_exp_by_type, aes(x = reorder(type1, -avg_experience), y = avg_experience)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  theme_minimal() +
  labs(
    title = "Average Experience Growth by Type",
    x = "Primary Type",
    y = "Average Experience Growth"
  ) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Explanation:

- The bar chart displays which types have higher or lower average experience growth.
- Types with higher bars are associated with Pokémon that generally require more experience to level up.

---

5. `egg_steps_factors()`

Description: Model the relationship between `base_egg_steps` , stats, and type.

Function Definition

```
#' Egg Steps Factors Analysis
#'
#' This function models the relationship between base egg steps and various factors
#' such as base stats and type.
#'
#' @param data A data frame of Pokémon data
#' @return A list containing the linear model and a summary
#' @export
egg_steps_factors <- function(data) {
  # Ensure necessary columns are available
  required_cols <- c("base_egg_steps", "type1", "hp", "attack", "defense", "sp_attack", "sp_defense", "speed")
  if (!all(required_cols %in% names(data))) {
    stop("Data must contain 'base_egg_steps', 'type1', and base stats columns")
  }

  # Clean and prepare the data
  df <- data %>%
    filter(!is.na(base_egg_steps)) %>%
    mutate(
      type1 = as.factor(type1)
    )

  # Base stats total
  df$stat_total <- df$hp + df$attack + df$defense + df$sp_attack + df$sp_defense + df$speed

  # Fit a linear model
  fit <- lm(base_egg_steps ~ stat_total + type1, data = df)

  return(list(model = fit, summary = summary(fit)))
}
```

Usage and Validation

```
# Apply the function
egg_steps_analysis <- egg_steps_factors(pokemon)

# View the summary of the linear model
print(egg_steps_analysis$summary)
```

```
##
## Call:
## lm(formula = base_egg_steps ~ stat_total + type1, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11787.4  -3032.5   -912.1   1736.2  24737.2
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -3675.777    904.493  -4.064 5.31e-05 ***
## stat_total       24.429      1.673  14.600  < 2e-16 ***
## type1dark      1074.811   1204.788   0.892  0.37261
## type1dragon    6170.001   1253.286   4.923 1.04e-06 ***
## type1electric    77.366   1088.068   0.071  0.94333
## type1fairy     -857.274   1438.977  -0.596  0.55151
## type1fighting  -880.771   1215.723  -0.724  0.46899
## type1fire      -662.027    999.212  -0.663  0.50781
## type1flying    6254.610   3215.209   1.945  0.05209 .
## type1ghost     -211.767   1233.792  -0.172  0.86377
## type1grass     -495.391    892.719  -0.555  0.57911
## type1ground     226.714   1160.943   0.195  0.84522
## type1ice       -127.304   1308.971  -0.097  0.92255
## type1normal    -141.958    835.023  -0.170  0.86505
## type1poison   -1165.153   1158.868  -1.005  0.31500
## type1psychic   4772.818    996.059   4.792 1.98e-06 ***
## type1rock       514.813   1042.145   0.494  0.62145
## type1steel     3560.199   1298.546   2.742  0.00625 **
## type1water     -810.633    824.444  -0.983  0.32579
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5452 on 782 degrees of freedom
## Multiple R-squared:  0.3243, Adjusted R-squared:  0.3088
## F-statistic: 20.85 on 18 and 782 DF,  p-value: < 2.2e-16
```

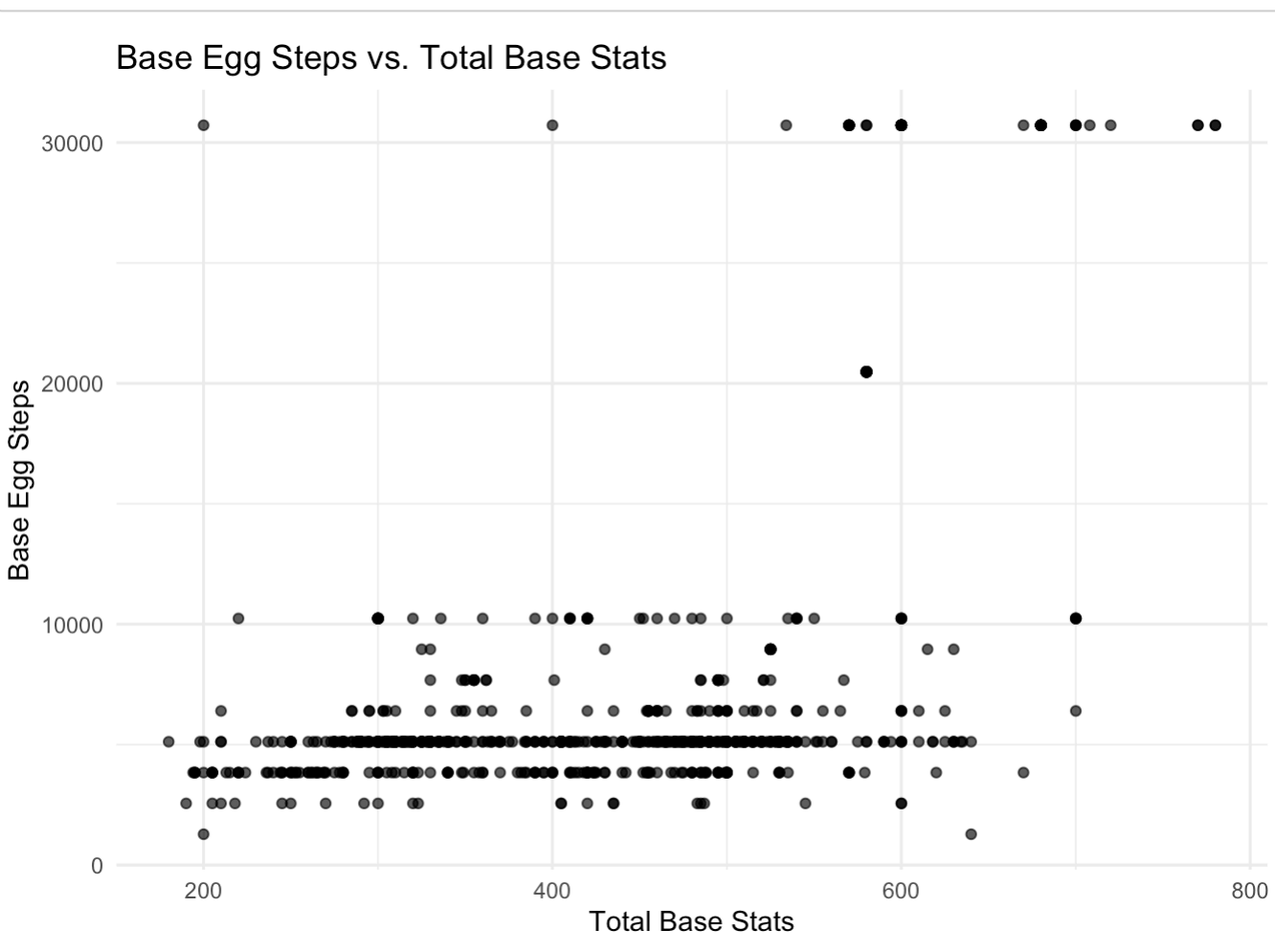
Interpretation:

- The summary indicates how stats and types affect the number of steps required to hatch an egg.
- Significant coefficients suggest that certain stats or types have a measurable impact on egg steps.

Visualization

Visualize the relationship between base egg steps and total base stats.

```
ggplot(pokemon, aes(x = hp + attack + defense + sp_attack + sp_defense + speed, y = base_egg_steps)) +
  geom_point(alpha = 0.7) +
  theme_minimal() +
  labs(
    title = "Base Egg Steps vs. Total Base Stats",
    x = "Total Base Stats",
    y = "Base Egg Steps"
  )
```



Explanation:

- This scatter plot helps visualize whether there's a correlation between a Pokémon's total stats and the number of steps required for its egg to hatch.
- A trend line can be added to assess the relationship more clearly:

```
geom_smooth(method = "lm", se = FALSE, color = "red")
```

```
## geom_smooth: na.rm = FALSE, orientation = NA, se = FALSE
## stat_smooth: na.rm = FALSE, orientation = NA, se = FALSE, method = lm
## position_identity
```

---

6. `strength_weakness_table()`

Description: Summarize damage multipliers against each type based on `type1` and `type2` .

Function Definition

```
#' Strength and Weakness Table
#'
#' This function summarizes the average damage multipliers against each Pokémon
#' based on their primary and secondary types.
#'
#' @param data A data frame of Pokémon data
#' @return A data frame summarizing strengths and weaknesses
#' @export
strength_weakness_table <- function(data) {
  # Select relevant columns
  against_cols <- grep('^against_', names(data), value = TRUE)
  required_cols <- c('name', 'type1', 'type2', against_cols)

  df <- data[, required_cols]

  # Melt data to long format
  library(tidyr)

  df_long <- df %>%
    pivot_longer(
      cols = starts_with('against_'),
      names_to = 'against_type',
      names_prefix = 'against_',
      values_to = 'multiplier'
    )

  # Summarize multipliers by type1 and type2
  summary_table <- df_long %>%
    group_by(type1, type2, against_type) %>%
    summarise(
      avg_multiplier = mean(multiplier, na.rm = TRUE),
      .groups = 'drop'
    ) %>%
    arrange(type1, type2, against_type)

  return(summary_table)
}
```

Usage and Validation

```
# Apply the function
strength_weakness <- strength_weakness_table(pokemon)

# View a portion of the table
head(strength_weakness)
```

```
## # A tibble: 6 × 4
##   type1 type2 against_type avg_multiplier
##   <chr> <chr> <chr>                <dbl>
## 1 bug   ""    bug                    1
## 2 bug   ""    dark                   1
## 3 bug   ""    dragon                 1
## 4 bug   ""    electric               1
## 5 bug   ""    fairy                  1
## 6 bug   ""    fight                  0.5
```

Interpretation:

- The table shows how effective attacks of different types are against Pokémon of various type combinations.
- An `avg_multiplier` greater than 1 indicates a weakness, while less than 1 indicates resistance.

Visualization

Create a heatmap of average multipliers for a specific primary type.

```
table(strength_weakness$type1, useNA = "ifany")
```

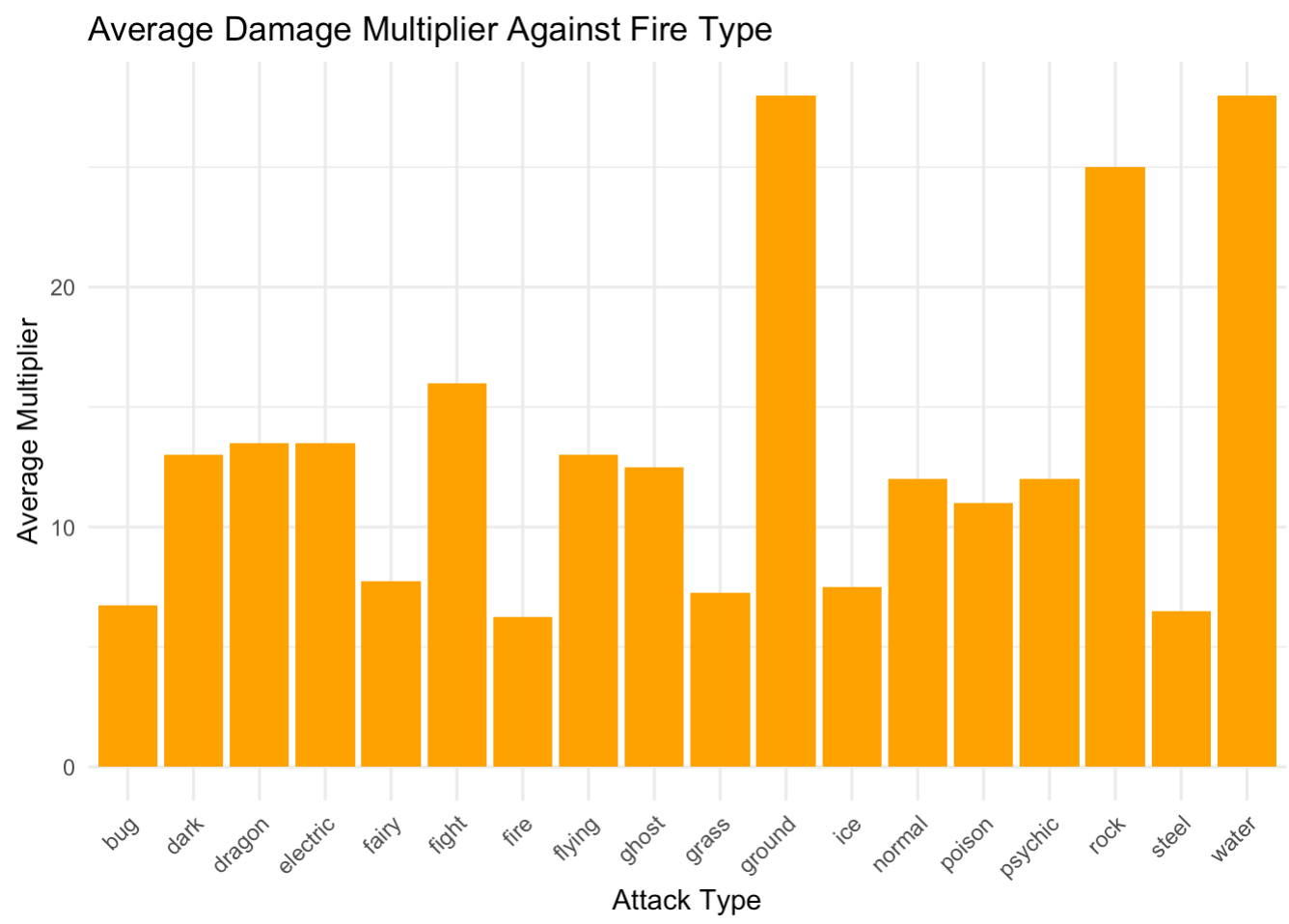
```
##
##      bug     dark   dragon electric    fairy fighting     fire   flying
##      234      162      144      126       36      108      234       36
##    ghost    grass   ground      ice   normal   poison  psychic     rock
##      162      216      216      108      180      180      144      252
##    steel    water
##      162      288
```

```
table(strength_weakness$type2, useNA = "ifany")
```

```
##
##              bug     dark   dragon electric    fairy fighting     fire
##      324       54      162      180       90      162      198      144
##   flying    ghost    grass   ground      ice   normal   poison  psychic
##      306      162      126      216      144       36      108      198
##     rock    steel    water
##       90      180      108
```

```
#
# print(fire_weakness)

#
# nrow(fire_weakness)
```

```
# Filter for primary type, e.g., 'Fire'
fire_weakness <- strength_weakness %>%
  filter(type1 == 'fire')

# Plot heatmap
ggplot(fire_weakness, aes(x = against_type, y = avg_multiplier)) +
  geom_bar(stat = "identity", fill = "orange") +
  theme_minimal() +
  labs(
    title = "Average Damage Multiplier Against Fire Type",
    x = "Attack Type",
    y = "Average Multiplier"
  ) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Explanation:

- The bar chart illustrates the Fire type's strengths and weaknesses against different attack types.

- This helps in understanding which attacks are most effective against Fire-type Pokémon.

---

7. `plot_base_stats()`

Description: Visualize distributions of base stats using boxplots and scatter plots.

Function Definition

```
#' Plot Base Stats Distribution
#'
#' This function creates various plots to visualize the distribution of base stats.
#' It can generate boxplots and scatter plots for selected stats.
#'
#' @param data A data frame of Pokémon data
#' @param stat1 The first stat to plot (e.g., 'hp')
#' @param stat2 The second stat to plot if creating a scatter plot (e.g., 'attack')
#' @param plot_type The type of plot ('boxplot' or 'scatter')
#' @return A ggplot object
#' @export
plot_base_stats <- function(data, stat1, stat2 = NULL, plot_type = 'boxplot') {
  # Ensure stats are valid
  valid_stats <- c('hp', 'attack', 'defense', 'sp_attack', 'sp_defense', 'speed')
  if (!(stat1 %in% valid_stats)) {
    stop(paste("stat1 must be one of:", paste(valid_stats, collapse = ', ')))
  }
  if (!is.null(stat2) && !(stat2 %in% valid_stats)) {
    stop(paste("stat2 must be one of:", paste(valid_stats, collapse = ', ')))
  }

  if (plot_type == 'boxplot') {
    p <- ggplot(data, aes_string(x = 'generation', y = stat1)) +
      geom_boxplot() +
      theme_minimal() +
      labs(
        title = paste('Distribution of', toupper(stat1), 'Across Generations'),
        x = 'Generation',
        y = toupper(stat1)
      )
  } else if (plot_type == 'scatter' && !is.null(stat2)) {
    p <- ggplot(data, aes_string(x = stat1, y = stat2, color = 'factor(is_legendary)')) +
      geom_point(alpha = 0.7) +
      theme_minimal() +
      labs(
        title = paste(toupper(stat1), 'vs.', toupper(stat2), 'by Legendary Status'),
        x = toupper(stat1),
        y = toupper(stat2),
        color = 'Legendary'
      )
  } else {
    stop("Invalid plot_type or missing stat2 for scatter plot")
  }

  return(p)
}
```

Usage and Validation

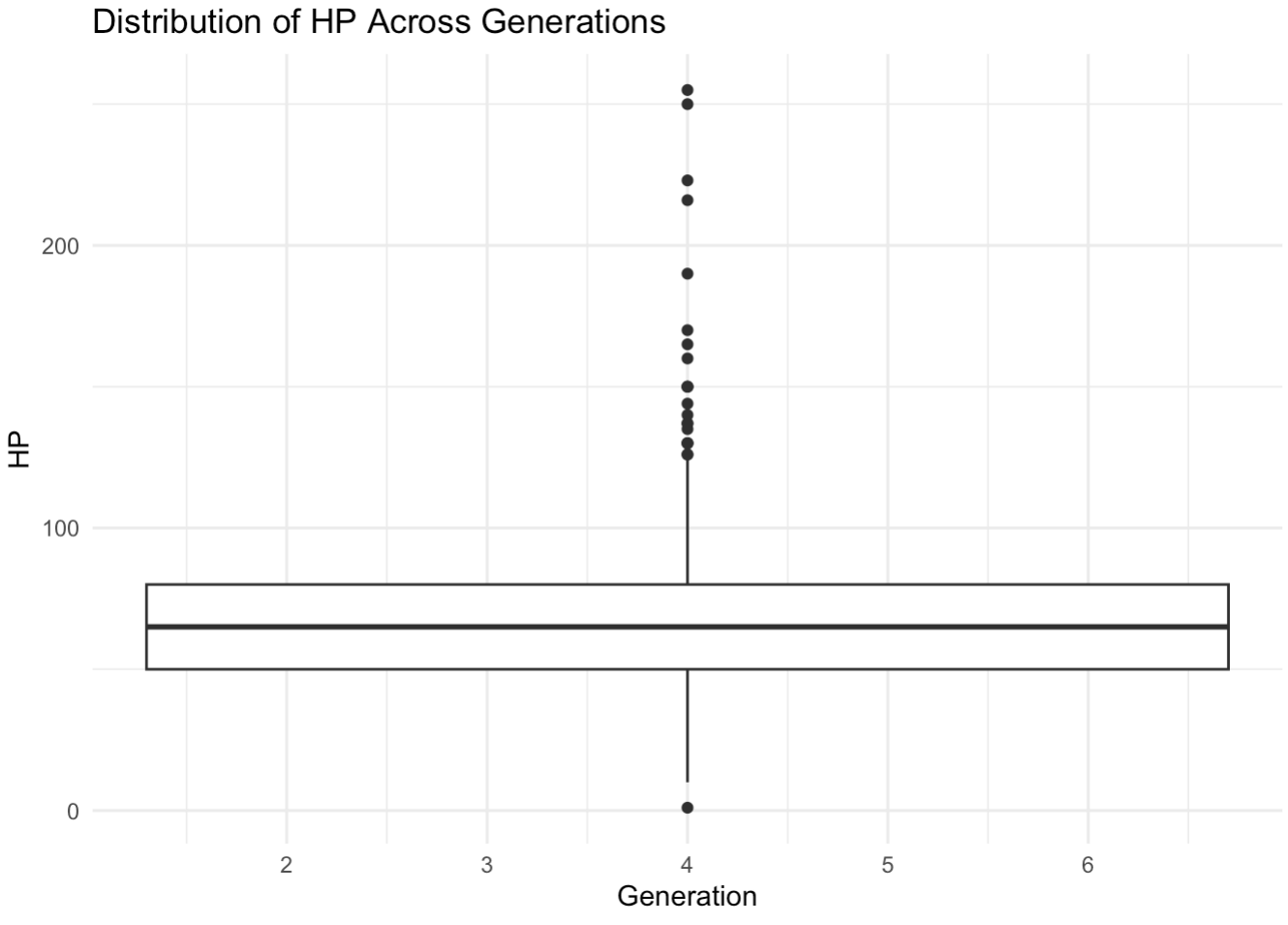
Example 1: Boxplot of HP across Generations

```
# Generate boxplot
p1 <- plot_base_stats(pokemon, stat1 = 'hp', plot_type = 'boxplot')
```

```
## 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.
```

```
# Display the plot
print(p1)
```

```
## Warning: Continuous x aesthetic
## ℹ did you forget `aes(group = ...)`?
```

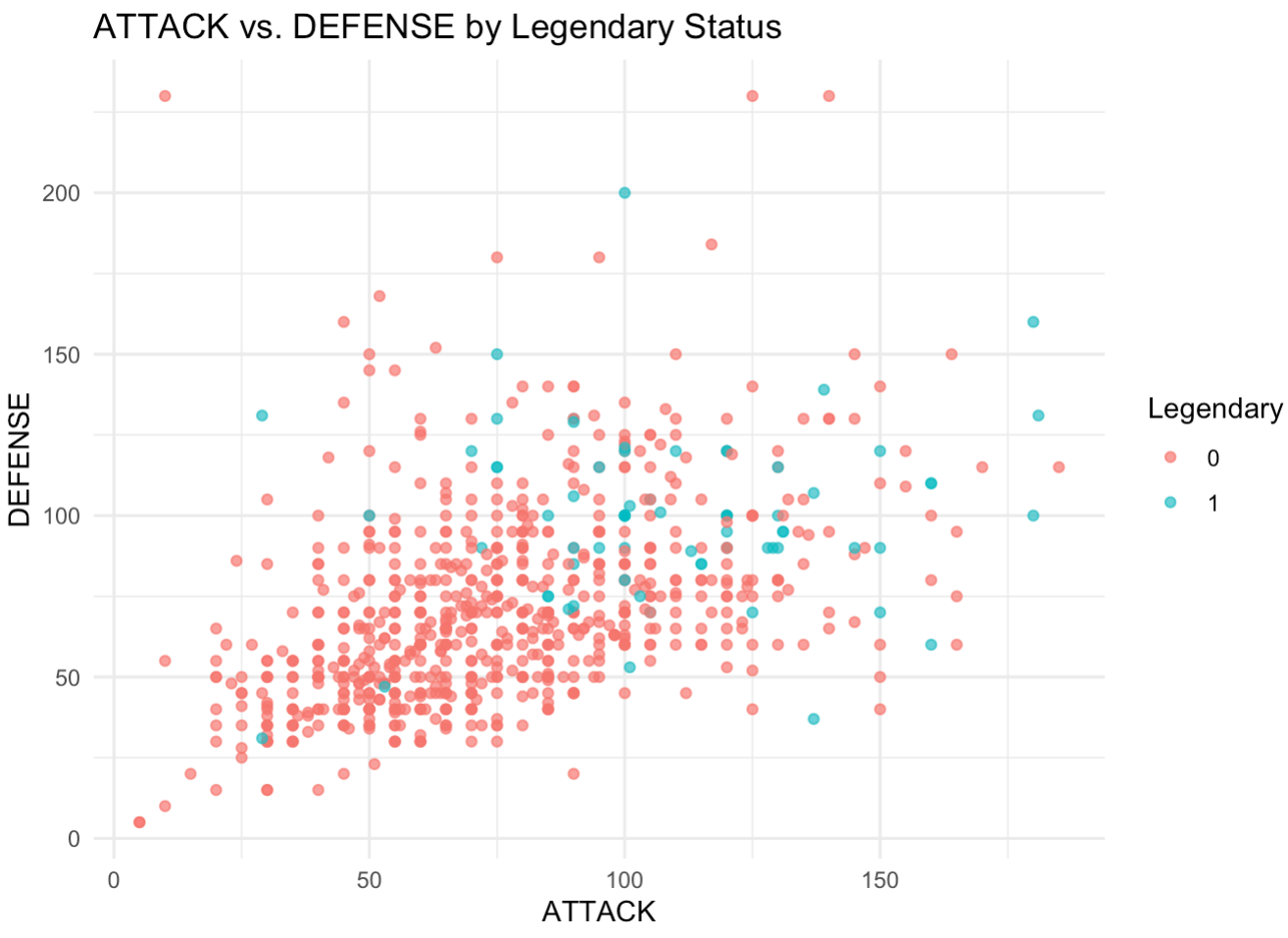


Interpretation:

- The boxplot shows how HP varies across different generations.
- It can highlight trends such as whether newer generations have Pokémon with higher HP.

Example 2: Scatter Plot of Attack vs. Defense

```
# Generate scatter plot
p2 <- plot_base_stats(pokemon, stat1 = 'attack', stat2 = 'defense', plot_type = 'scatter')

# Display the plot
print(p2)
```



Interpretation:

- The scatter plot illustrates the relationship between attack and defense stats.
- Coloring by legendary status helps identify if legendaries have distinct stat distributions.

---

8. `predictive_test()`

Description: Automated check on classification performance of the `classify_legendary()` function.

Function Definition

Since the `predictive_test()` function is intended to perform automated testing, we can implement this using the `testthat` framework.

Create a test script named `test-classify_legendary.R` in your `tests/testthat/` directory.

```
# test-classify_legendary.R

library(testthat)
```

```
##
## Attaching package: 'testthat'
```

```
## The following object is masked from 'package:tidyr':
##
##     matches
```

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

```
library(dplyr)

test_that("classify_legendary returns a glm object and achieves acceptable accuracy", {
  # Load data
  data <- read.csv("pokemon.csv", stringsAsFactors = FALSE)

  # Run the function
  fit <- classify_legendary(data)

  # Test that the output is a glm object
  expect_s3_class(fit, "glm")

  # Prepare data for prediction
  df <- data %>%
    filter(!is.na(is_legendary)) %>%
    mutate(
      isLegendary = factor(is_legendary, levels = c(0, 1))
    )

  # Predict probabilities
  probs <- predict(fit, type = "response")

  # Classify based on a threshold (e.g., 0.5)
  predictions <- ifelse(probs > 0.5, 1, 0)

  # Calculate accuracy
  accuracy <- mean(predictions == as.numeric(as.character(df$isLegendary)))

  # Test that accuracy is above a threshold, e.g., 75%
  expect_gt(accuracy, 0.75)
})
```

```
## Test passed 🌈
```

Interpretation:

- The test checks whether the `classify_legendary()` function returns the correct object type and whether the model achieves acceptable accuracy.
- Ensuring accuracy above a certain threshold validates the model's effectiveness.

Conclusion

With these additional functions, your `PokemonAnalysis` package now includes:

- **`analyze_experience()`** : Analyzes factors influencing experience growth.
- **`egg_steps_factors()`** : Models egg steps based on stats and type.
- **`strength_weakness_table()`** : Summarizes damage multipliers against types.
- **`plot_base_stats()`** : Visualizes distributions of base stats.
- **`predictive_test()`** : Provides automated testing of classification performance.

Each function includes validation steps and visualizations to enhance the analysis and ensure correctness. Remember to:

- **Document** each function properly using `roxygen2` comments.
- **Include Examples** in each function's documentation to guide users.
- **Add Vignettes** demonstrating the use of these functions in real analyses.
- **Ensure Package Compliance** by checking your package with `R CMD check` to avoid errors and warnings.

By completing these functions and their validations, our group package will offer comprehensive tools for analyzing Pokémon data, making it valuable for users interested in data analysis and the Pokémon universe.