DATA501: Assignment2

Heba Aleterji

2024-08-07

`influenceAnalysis` Package

Overview

The `influenceAnalysis` package designed for conducting diagnostic evaluations of linear regression models in R. The package focuses on identifying influential data points that may disproportionately affect the model's parameter estimates. Specifically, the package provides functionalities for calculating three critical influence measures: Cook's Distance, DFFITS, and Hadi's Influence Measure. These measures, grounded in established statistical methodologies (Cook, 1977; Welsch and Kuh, 1977; Hadi, 1992), enable researchers to detect outliers and influential observations that could undermine the validity of regression analysis [1]. The package requires a dataset and a fitted linear model object (`lm`) as inputs and includes features for input validation to ensure robustness against NA values, infinite values, and mismatched dimensions. Additionally, `influenceAnalysis` offers visual diagnostics through plotting functions that allow users to easily interpret the influence measures.

Hadi's Influence Measure

The `hadis_influence_measure` function in the influenceAnalysis package computes Hadi's Influence Measure for a given linear model (lm object). This diagnostic measure helps identify influential observations that significantly impact the regression model's parameter estimates. The function first verifies that the input model is of class lm, stopping execution and raising an error if it is not. It then calculates the residuals of the model, which represent the differences between observed and predicted values. Using these residuals, the function computes the Sum of Squared Errors (SSE), which quantifies the total deviation of the response values from their predicted values.

The number of predictors in the model is determined by the length of the coefficient vector, while the leverage values (hat values) are obtained to measure the influence of each observation on the fitted values. The number of observations is derived from the length of the residuals vector. Next, the function standardizes the residuals by dividing each residual by the square root of the mean squared error. These standardized residuals help in identifying outliers. Hadi's Influence Measure is then calculated using a formula that combines the influence of the leverage and the residuals, adjusted by the number of predictors plus one [2].

The theoretical equation for Hadi's Influence Measure (H_i) is:

$H_i = \frac{p_{ii}}{1-p_{ii}} + \frac{p+1}{1-p_{ii}} \frac{d_i^2}{1-d_i^2}$, $i = 1, 2, ..., n$

The implemented `hadis_influence_measure` function calculates:

$H_i = (\frac{H}{1-H}) + (\frac{p+1}{1-H}) + (\frac{d_i^2}{1-d_i^2})$

where H are the hat values corresponds to p_{ii} , d_i are the standardized residuals, and p is the number of predictors. Finally, the function returns a numeric vector containing Hadi's Influence Measure for each observation.

```r
#' @title Compute Hadi's Influence Measure
#' @description Computes Hadi's Influence Measure for an lm model.
#' @param model An object of class lm.
#' @return A numeric vector of Hadi's influence measures.
#' @examples
#' model <- lm(mpg ~ wt + hp, data = mtcars)
#' hadis_influence_measure(model)
hadis_influence_measure <- function(model) {
  if (!inherits(model, "lm")) {
    stop("The model should be an object of class 'lm'")
  }

  e_i <- residuals(model)
  SSE <- sum(e_i^2)
  p <- length(coef(model))
  H <- hatvalues(model)
  n <- length(e_i)

  d_i <- e_i / sqrt(SSE / (n - p))

  H_i <- (H / (1 - H)) + ((p + 1) / (1 - H)) * (d_i^2 / (1 - d_i^2))

  return(H_i)
}
```

Cook's Distance Measure

The `cooks_distance_scratch` function in the `influenceAnalysis` package computes Cook's Distance for a given linear model (`lm` object). Cook's Distance is a measure used in regression analysis to identify observations that have an undue influence on the estimated regression coefficients.This function begins by calculating the hat values (leverages) from the model, which measure the influence of each observation on the fitted values. Residuals, which are the differences between observed and predicted values, are then calculated. The mean squared error (MSE) is computed by summing the squared residuals and dividing by the residual degrees of freedom, which is the number of observations minus the number of predictors. The number of predictors in the model is determined by the length of the coefficient vector, including the intercept term [3].

The function then calculates Cook's Distance D_i using the formula:

$$D_i = \frac{e_i^2}{ps^2} \frac{h_{ii}}{(i-h_{ii})^2}$$

where e_i are the residuals, p is the number of predictors, s^2 is the mean squared error, and h_{ii} are the hat values (leverages) for the ith observation. The term $\frac{e_i^2}{ps^2}$ measures the scaled residuals, while the term $\frac{h_{ii}}{(i-h_{ii})^2}$ adjusts for the leverage of the observation. Finally, the function returns a numeric vector of Cook's Distances for each observation.

```r
#' @title Compute Cook's Distance from Scratch
#' @description Computes Cook's Distance for an lm model from scratch.
#' @param model An object of class lm.
#' @return A numeric vector of Cook's distances.
#' @examples
#' model <- lm(mpg ~ wt + hp, data = mtcars)
#' cooks_distance_scratch(model)
cooks_distance_scratch <- function(model) {
  h <- hatvalues(model)
  e <- residuals(model)
```

```
  s2 <- sum(e^2) / df.residual(model)
  p <- length(coef(model))
  (e^2 / (p * s2)) * (h / (1 - h)^2)
}
```

DFFITS Measure

The `dffits_scratch` function in the `influenceAnalysis` package computes DFFITS for a given linear model (`lm` object). DFFITS is a diagnostic measure used in regression analysis to determine the influence of each observation on the fitted values of the model. The function first calculates the hat values (leverages) from the model, which measure the influence of each observation on the fitted values and are the diagonal elements of the hat matrix H. Next, it calculates the residuals from the model, which are the differences between the observed and predicted values. The mean squared error (MSE) is then calculated as $s^2 = \frac{\sum e_i^2}{n-p}$, where e_i are the residuals,n is the number of observations, and p is the number of predictors [4].

The main calculation for DFFITS is then performed using the formula:

$$DFFITS_i = \frac{e_i \sqrt{h_{ii}}}{\sqrt{s^2(1-h_{ii})}}$$

where e_i are the residuals, h_{ii} are the hat values (leverages) for the ith observation, and s^2 is the mean squared error. The term $e_i\sqrt{h_{ii}}$ measures the influence of the residuals adjusted for leverage, while the term $\sqrt{s^2(1-h_{ii})}$ normalizes this influence by the mean squared error and the leverage. Finally, the function returns a numeric vector of DFFITS values.

```
#' @title Compute DFFITS from Scratch
#' @description Computes DFFITS for an lm model from scratch.
#' @param model An object of class lm.
#' @return A numeric vector of DFFITS values.
#' @examples
#' model <- lm(mpg ~ wt + hp, data = mtcars)
#' dffits_scratch(model)
dffits_scratch <- function(model) {
  h <- hatvalues(model)
  e <- residuals(model)
  s2 <- sum(e^2) / df.residual(model)
  dffits <- e * sqrt(h / (1 - h)) / sqrt(s2)
  dffits
}
```

`validate_inputs` Function

The `validate_inputs` function is a helper function in the `influenceAnalysis` package designed to ensure that the inputs provided to the influence diagnostics functions are valid. It takes two parameters: `data`, which is the dataset used in the model and should be a data frame, and `model`, which is an object of class `lm` (linear model). The function begins by checking if the data is a data frame and if the model is an `lm` object, stopping execution and raising an error if these conditions are not met. It then checks for any missing values (NA) in the data, and if any are found, it stops execution and raises an error. Next, the function checks for infinite values in the data by applying the is.infinite function to each column and stops execution if any infinite values are found. Finally, the function validates that the number of columns in the data matches the number of predictors in the model. It extracts the predictor terms and the response variable from the model, calculates the number of predictors, and checks if the number of columns in the data (including the response variable) matches the number of predictors plus one. If there is a mismatch, the function stops execution and raises an error. In summary, the validate_inputs function ensures that the dataset and model are correctly formatted, contain no missing or infinite values, and have the appropriate number of columns, thereby preventing further analysis with invalid inputs.

```r
#' @title Validate Inputs
#' @description Validates the inputs for the influence diagnostics functions.
#' @param data The dataset used in the model.
#' @param model An object of class lm.
#' @return NULL. Stops execution if validation fails.
validate_inputs <- function(data, model) {
  stopifnot(is.data.frame(data))
  stopifnot(inherits(model, "lm"))

  if (anyNA(data)) stop("Data contains NA values")

  if (any(unlist(lapply(data, function(col) any(is.infinite(col)))))) {
    stop("Data contains infinite values")
  }

  # Extract the terms from the model
  terms <- attr(model$terms, "term.labels")
  response <- attr(model$terms, "response")
  predictors <- length(terms)

  if (ncol(data) != predictors + 1) {  # Include response variable column
    stop("The number of columns in data does not match the number of predictors in the model")
  }
}
```

`subset_data` Function

The subset_data function is designed to extract the subset of data used in the specified linear model (`lm` object), ensuring that only the relevant variables are included in the analysis. It takes two parameters: `data`, the dataset used in the model, and `model`, an object of class `lm`. The function begins by extracting all the variable names (both predictors and response) used in the model formula with `all.vars(formula(model))`. It then subsets the data to include only these variables using `data[, model_vars, drop = FALSE]`, where `drop = FALSE` ensures that the result is a data frame even if only one column is selected. Finally, the function returns the subsetted data, which contains only the variables used in the specified model. This process ensures that the subsequent analysis focuses solely on the relevant predictors and response variable in the model.

```r
subset_data <- function(data, model) {
  model_vars <- all.vars(formula(model))
  data_subset <- data[, model_vars, drop = FALSE]
  return(data_subset)
}
```

`plot_influence` Function

The `plot_influence` function is designed to visually represent the influence measures calculated for a linear regression model. It takes as its input a `diagnostics` list, which contains different influence measures such as Cook's Distance, DFFITS, and Hadi's Influence Measure. The function is structured to create a series of plots, each corresponding to one of these influence measures, provided they are available in the diagnostics list. The plots are arranged vertically in a single column to ensure clarity and ease of comparison.

To manage the plotting space effectively and prevent errors related to figure margins, the function sets up the graphical parameters using `par()`, specifying the layout with `mfrow` and adjusting the margins with `mar`. After plotting the available measures, the function resets the graphical parameters to their default state, ensuring that subsequent plots are unaffected.

```r
#' @title Plot Influence Measures
#' @description Plots the provided influence measure results for a model.
#' @param diagnostics The result from influence_diagnostics.
#' @return Plot
#' @export
#' @examples
#' diagnostics <- influence_diagnostics(mtcars, model, "all")
#' plot_influence(diagnostics)
plot_influence <- function(diagnostics) {
  # Set the plot parameters
  par(mfrow = c(3, 1), mar = c(5, 4, 4, 2) + 0.1)

  # Plot Cook's Distance if available
  if (!is.null(diagnostics$cooks)) {
    plot(diagnostics$cooks, main = "Cook's Distance", ylab = "Influence",
         xlab = "Index", type = "h", col = "blue")
  }

  # Plot DFFITS if available
  if (!is.null(diagnostics$dffits)) {
    plot(diagnostics$dffits, main = "DFFITS", ylab = "Influence",
         xlab = "Index", type = "h", col = "red")
  }

  # Plot Hadi's Influence Measure if available
  if (!is.null(diagnostics$hadi)) {
    plot(diagnostics$hadi, main = "Hadi's Influence Measure",
         ylab = "Influence", xlab = "Index", type = "h", col = "darkgreen")
  }

  # Reset the plot parameters to default
  par(mfrow = c(1, 1))
}
```

`influence_diagnostics` Function

The `influence_diagnostics` function in the `influenceAnalysis` package is designed to compute various influence measures for a given linear model (`lm` object) and optionally plot the results. These measures help identify influential observations that significantly impact the regression model's parameter estimates. The function takes three parameters: `data`, the dataset used in the model (required); `model`, an object of class `lm` (required); and `measure`, the influence measure to compute (optional), which can be "cooks", "dffits", "hadi", or "all", with the default being "all".

The function begins by subsetting the data to include only the predictors used in the model, ensuring that only the relevant variables are considered for the diagnostics. This is done using the `subset_data` function. It then validates the inputs using the `validate_inputs` function to ensure that the data and model are correctly formatted, contain no missing or infinite values, and that the number of predictors matches the model specifications. The `measure` argument is matched against the allowed values to ensure it is correctly specified. An empty list results is initialized to store the computed influence measures.

Depending on the value of the `measure` argument, the function calls the appropriate sub-functions and stores their results in the results list. If "cooks" or "all" is specified, it computes Cook's Distance using `cooks_distance_scratch(model)`. If "dffits" or "all" is specified, it computes DFFITS using `dffits_scratch(model)`. If "hadi" or "all" is specified, it computes Hadi's Influence Measure using

`hadis_influence_measure(model)`. After computing the influence measures, the function plots the results using `plot_influence(results)`, creating a plots that visually identifies influential observations. Finally, the function returns the results list containing the computed influence measures.

```r
#' @title Perform Influence Diagnostics
#' @description Computes selected influence measures for an lm model.
#' @param data The dataset used in the model.
#' @param model An object of class lm.
#' @param measure The influence measure to compute ("cooks", "dffits", "hadi", or "all").
#' @return A list containing the selected influence measures.
#' @examples
#' model <- lm(mpg ~ wt + hp, data = mtcars)
#' influence_diagnostics(mtcars, model, "all")
influence_diagnostics <- function(data, model,  measure = c("all", "cooks", "dffits", "hadi")) {

  # Subset the data to include only the predictors used in the model
  data_subset <- subset_data(data, model)

  validate_inputs(data_subset, model)
  measure <- match.arg(measure)

  results <- list()

  if (measure == "all" || measure == "cooks") {
    results[["cooks"]] <- cooks_distance_scratch(model)
  }
  if (measure == "all" || measure == "dffits") {
    results[["dffits"]] <- dffits_scratch(model)
  }
  if (measure == "all" || measure == "hadi") {
    results[["hadi"]] <- hadis_influence_measure(model)
  }


  plot_influence(results)

  return(results)

}
```

Practical Usage of the `influence_diagnostics` Function

1. Loading the Package and Data

   Start by loading the influenceAnalysis package and the dataset of interest, such as mtcars.

2. Fitting a Linear Model

   Fit a linear model to the data. For example, predicting miles per gallon (mpg) based on weight (wt) and horsepower (hp).

3. Performing Influence Diagnostics

   Use the influence_diagnostics function to calculate influence measures for the fitted model. By computing all measures: Cook's Distance, DFFITS, and Hadi's Influence Measure.

Detailed Steps in the influence_diagnostics Function

1. Sub-setting the Data

   The function subsets the data to include only the predictors used in the model using the `subset_data` function. This ensures that only relevant variables are considered for diagnostics.

2. Validating the Inputs

   It validates the inputs using the `validate_inputs` function to ensure the data and model are correctly formatted, contain no missing or infinite values, and that the number of predictors matches the model specifications.

3. Computing Influence Measures

   Depending on the specified `measure`, the function computes Cook's Distance using `cooks_distance_scratch`, DFFITS using `dffits_scratch`, and Hadi's Influence Measure using `hadis_influence_measure`. These computed measures are stored in a list.

4. Plotting the Influence Measures

   The `plot_influence` function is called to generate plots for the computed influence measures, providing a visual representation of influential observations.

5. Returning the Results

   The function returns a list containing the computed influence measures.

Testing `influenceAnalysis` using `testthat`

The test file for the `influenceAnalysis` package contains several tests that ensure the correctness and robustness of the key functions within the package. Each test checks different aspects of the package's functionality using the `testthat` framework.

Testing `hadis_influence_measure` Function

```
library(testthat)
library(influenceAnalysis)

# Test for hadis_influence_measure function
test_that("hadis_influence_measure works correctly", {
  model <- lm(mpg ~ wt + hp, data = mtcars)
  result <- hadis_influence_measure(model)
  expect_true(is.numeric(result))
  expect_length(result, nrow(mtcars))
})
```

```
## Test passed
```

This test validates the `hadis_influence_measure` function. It fits a linear model and computes Hadi's Influence Measure. The test checks that the result is a numeric vector and that its length matches the number of rows in the `mtcars` dataset, confirming that each observation has a corresponding Hadi's Influence Measure.

Testing `cooks_distance_scratch` Function

```
# Test for cooks_distance_scratch function
test_that("cooks_distance_scratch works correctly", {
```

```
  model <- lm(mpg ~ wt + hp, data = mtcars)
  result <- cooks_distance_scratch(model)
  expect_true(is.numeric(result))
  expect_length(result, nrow(mtcars))
})
```

```
## Test passed
```

This test ensures that the `cooks_distance_scratch` function works correctly. It fits a linear model to the `mtcars` dataset and calculates Cook's Distance. The test checks that the result is a numeric vector and that its length matches the number of rows in the mtcars dataset, confirming that each observation has a corresponding Cook's Distance value.

Testing `dffits_scratch` Function

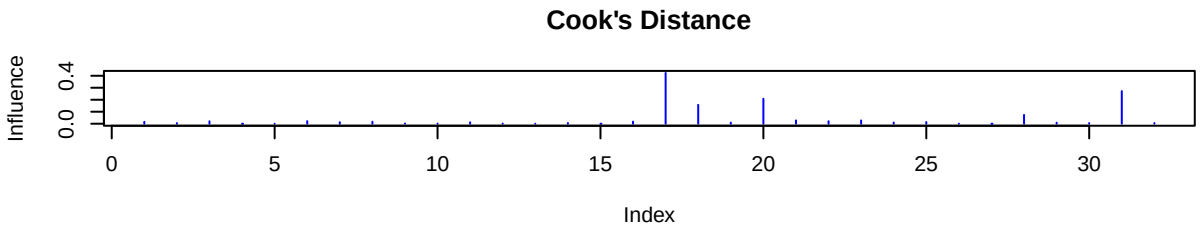
```
# Test for dffits_scratch function
test_that("dffits_scratch works correctly", {
  model <- lm(mpg ~ wt + hp, data = mtcars)
  result <- dffits_scratch(model)
  expect_true(is.numeric(result))
  expect_length(result, nrow(mtcars))
})
```

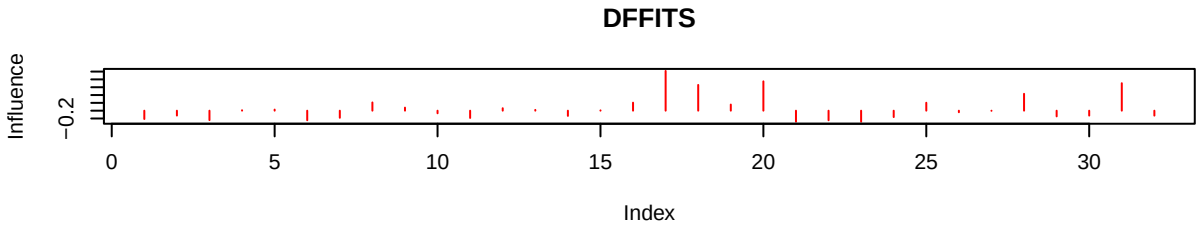
```
## Test passed
```

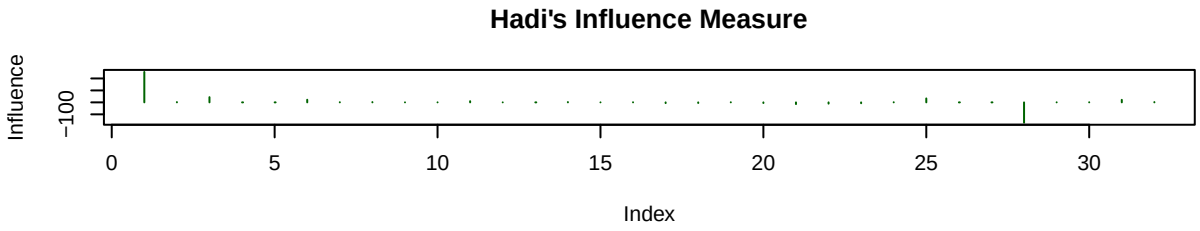
This test verifies the functionality of the `dffits_scratch` function. It fits a linear model and computes the DFFITS values. The test checks that the output is a numeric vector and that its length equals the number of rows in the `mtcars` dataset, ensuring that each observation has a corresponding DFFITS value.

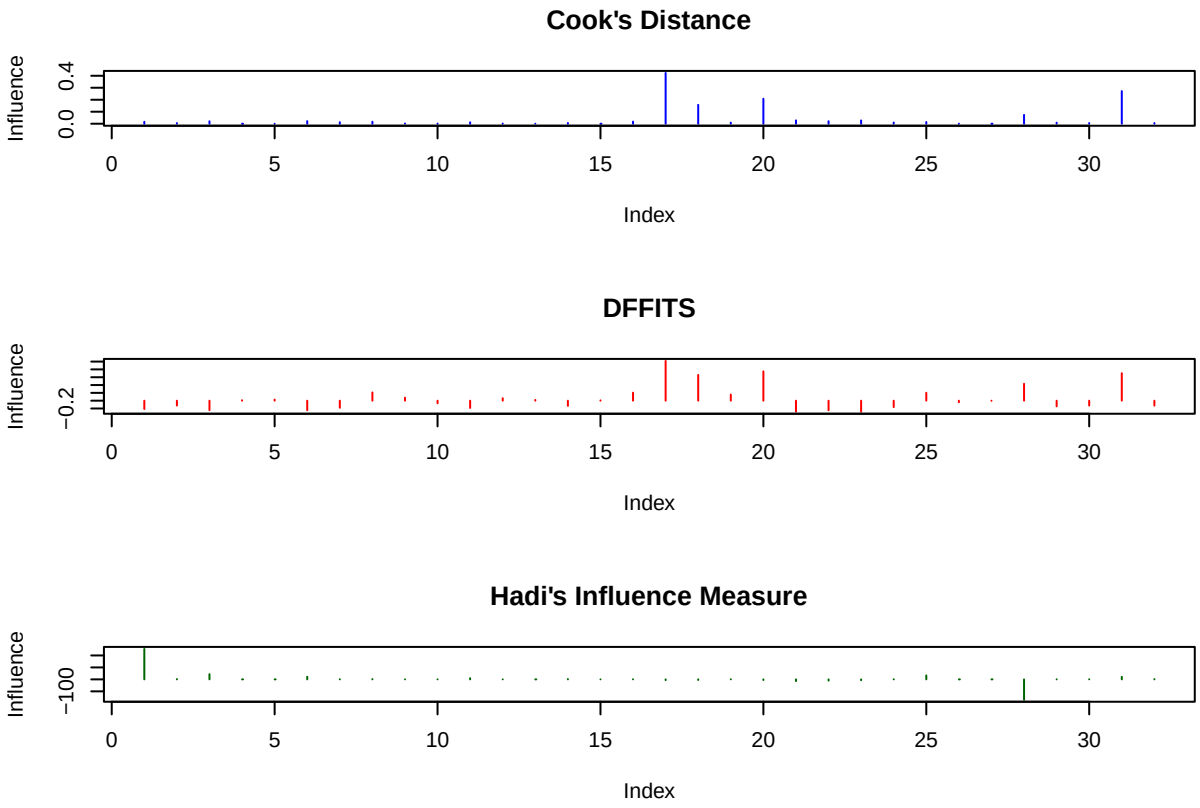
Testing plot_influence Function

```
# Test for plot_influence function
test_that("plot_influence works without errors", {
  model <- lm(mpg ~ wt + hp, data = mtcars)
  diagnostics <- influence_diagnostics(mtcars, model, "all")
  expect_silent(plot_influence(diagnostics))
})
```


Cook's Distance
Influence
0.0
0.4
0
5
10
15
20
25
30
Index


DFFITS
Influence
−0.2
0
5
10
15
20
25
30
Index


Hadi's Influence Measure
Influence
−100
0
5
10
15
20
25
30
Index




```
## Test passed
```

Testing `validate_inputs` Function

```
# Test for validate_inputs function
test_that("validate_inputs works correctly", {
  model <- lm(mpg ~ wt + hp, data = mtcars)
  data_subset <- subset_data(mtcars, model)
  expect_silent(validate_inputs(data_subset, model))
  expect_error(validate_inputs(data.frame(), model), "The number of columns in data does not match the r
})
```

```
## Test passed
```

This test verifies the `validate_inputs` function. It ensures that the function correctly validates the inputs for influence diagnostics. The test first checks that valid inputs (a subsetted `mtcars` dataset and a linear model) do not produce any errors. It then checks that invalid inputs (an empty data frame) produce an appropriate error message.

Testing subset_data Function

```
# Test for subset_data function
test_that("subset_data works correctly", {
  model <- lm(mpg ~ wt + hp, data = mtcars)
  data_subset <- subset_data(mtcars, model)
  expect_true(is.data.frame(data_subset))
```

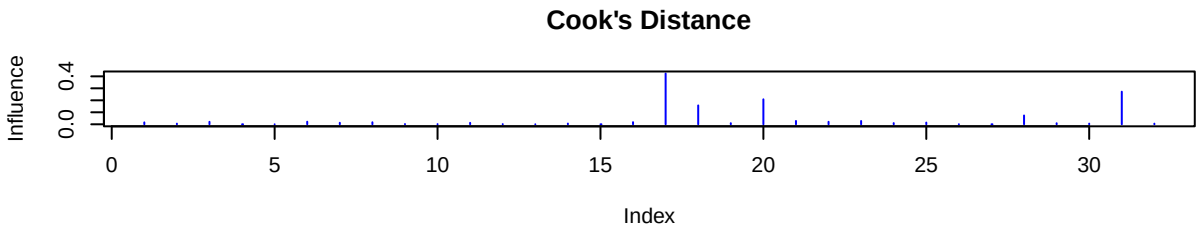
```
  expect_equal(ncol(data_subset), length(coef(model)))
})
```

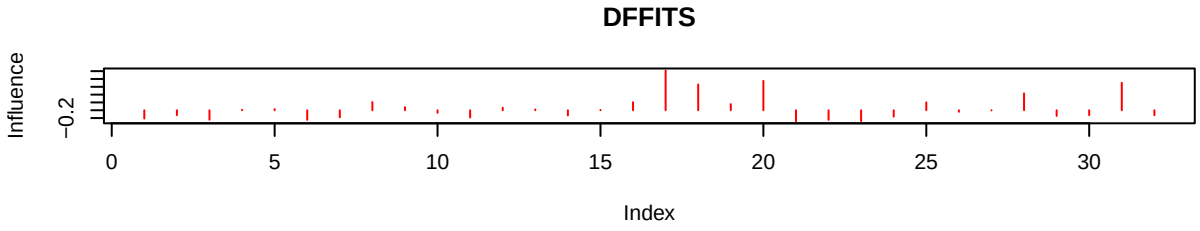
```
## Test passed
```

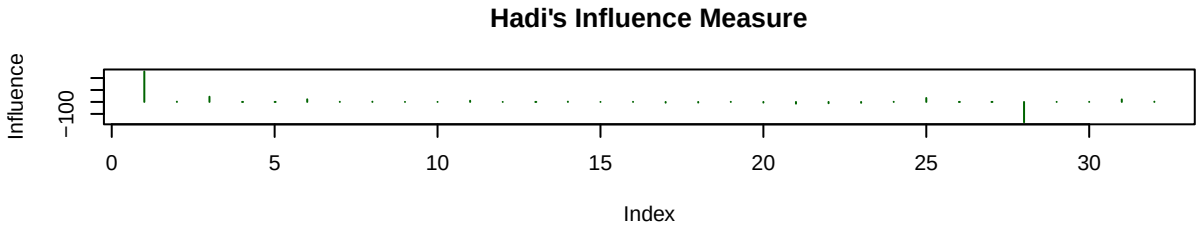
This test ensures that the `subset_data` function works correctly. It fits a linear model and subsets the `mtcars` dataset to include only the predictors used in the model. The test checks that the output is a data frame and that the number of columns in the subsetted data matches the number of coefficients in the model, ensuring that all relevant predictors are included.

Testing `influence_diagnostics` Function

```
# Test for influence_diagnostics function
test_that("influence_diagnostics works correctly", {
  model <- lm(mpg ~ wt + hp, data = mtcars)
  result <- influence_diagnostics(mtcars, model, "all")
  expect_true(is.list(result))
  expect_true("cooks" %in% names(result))
  expect_true("dffits" %in% names(result))
  expect_true("hadi" %in% names(result))
})
```








```
## Test passed
```

This test ensures the `influence_diagnostics` function works as expected. It fits a linear model and computes all influence measures. The test checks that the output is a list and that it contains the names “cooks”, “dffits”, and “hadi”, confirming that all specified influence measures are computed and included in the result.

The `influenceAnalysis` Package Vignette

The vignette for the `influenceAnalysis` package provides a comprehensive guide to effectively utilizing the package's tools for influence diagnostics in linear regression models. It begins by introducing the package's purpose and provides instructions for installing and loading the package:

```
# Install and load
devtools::install_github("HebaAleterji/Assignment2")
library(influenceAnalysis)
```

The vignette includes detailed examples on how to fit a linear model using the built-in `mtcars` dataset:

```
# Fit a linear model
model <- lm(mpg ~ wt + hp, data = mtcars)
```

It then demonstrates how to apply various influence measures, such as Cook's Distance, DFFITS, and Hadi's Influence Measure, using the `influence_diagnostics` function:

```
# Perform influence diagnostics
results <- influence_diagnostics(mtcars, model, "all")
```

Additionally, the vignette illustrates how to visualize the results with the `plot_influence` function, helping users identify influential data points:

```
# Plot influence measures
plot_influence(results)
```

the `influenceAnalysis_vignette` document is designed to be user-friendly, guiding users through the practical application of the package's tools with clear, annotated examples.

Repository Link for the `influenceAnalysis` Package

The `influenceAnalysis` package, designed for performing influence diagnostics in linear regression models, is available on GitHub. You can access the repository at https://github.com/HebaAleterji/Assignment2.git. To install the package, ensure you have the devtools package installed, and then use the following command in R:

```
# Install the devtools package if you haven't already
install.packages("devtools")

# Install influenceAnalysis from GitHub
devtools::install_github("HebaAleterji/Assignment2")
```

Once installed, load the package using`library(influenceAnalysis)`, and you'll be ready to use the package. The repository also contains a detailed vignette that guides you through the usage of the package with practical examples.

References

1. Sam Jayakumar, David & Sulthan, A.. (2016). EXACT DISTRIBUTION OF SQUARED WELSCH-KUH DISTANCE AND IDENTIFICATION OF INFLUENTIAL OBSERVATIONS. Journal of Reliability and Statistical Studies. 9. 69-91.

2. Hadi, A. S. (2012). Regression analysis by example (5th ed., Chapter 4).

3. Penn State. (n.d.). Estimating and interpreting the variance inflation factor (VIF). Penn State: STAT 462, from https://online.stat.psu.edu/stat462/node/173/

4. Penn State. (n.d.). Estimating and interpreting the variance inflation factor (VIF). Penn State: STAT 462, from https://online.stat.psu.edu/stat462/node/173/