**Code Explanation**

**Cars Dataset**

* summary(cars)
	+ # This function provides a summary of the `cars` dataset, which includes key descriptive statistics like mean, median, and range for numerical variables. The `cars` dataset is preloaded in R and contains speed and stopping distances for cars.

**Data Types and Structures**

* v.x <- vector()
	+ # Creates a generic empty vector `v.x` without specifying its type.
* v.y <- c(1:10)
	+ # Creates a numerical vector `v.y` containing integers from 1 to 10.
* str(v.x)
	+ # Displays the structure of `v.x`, showing its type and length.
* str(v.y)
	+ # Displays the structure of `v.y`.
* m.x <- matrix(data = c(1:12), nrow = 3, ncol = 4, byrow = TRUE)
	+ # Creates a 3x4 matrix `m.x` filled row-wise with integers 1 to 12.
* str(m.x)
	+ # Displays the structure of `m.x`, including its dimensions and data type.

**Data Frame Creation and Operations**

* v.a <- c(14:17)
	+ # A numerical vector containing integers from 14 to 17.
* v.b <- c("A", "b", "C", "D")
	+ # A character vector with four elements.
* v.c <- c(2.5)
	+ # A numerical vector initialized with a single value (should likely contain more values for matching length).
* df.abc <- data.frame(v.a, v.b, v.c)
	+ # Combines the vectors into a data frame `df.abc`.
* str(df.abc)
	+ # Displays the structure of `df.abc`, showing variable types and dimensions.
* sum(df.abc$v.a)
	+ # Calculates the sum of the values in the `v.a` column of `df.abc`.
* List.ab5 <- list(v.a, v.b, v.c, v.5)
	+ # Creates a list `List.ab5` containing multiple objects.
* as.matrix(df.abc)
	+ # Converts the data frame `df.abc` to a matrix, coercing elements to a common data type.
* str(as.matrix(df.abc))
	+ # Displays the structure of the matrix derived from `df.abc`.

**Anscombe's Dataset Analysis**

* data(anscombe)
	+ # Loads the `anscombe` dataset, which includes preloaded data demonstrating relationships between variables.
* str(anscombe)
	+ # Shows the structure of `anscombe`.
* apply(anscombe, 2, mean)
	+ # Computes the mean for each column (2 indicates column-wise operation).
* cor(anscombe$x1, anscombe$y1)
	+ # Computes the correlation between `x1` and `y1`.
* lm(anscombe$y1 ~ anscombe$x1)
	+ # Fits a linear regression model predicting `y1` using `x1`.
* plot(anscombe$x1, anscombe$y1)
	+ # Creates a scatterplot of `x1` and `y1`.
* abline(lm(anscombe$y1 ~ anscombe$x1))
	+ # Adds a best-fit line to the scatterplot.

**Matrix Operations**

* ans.mx <- as.matrix(anscombe)
	+ # Converts the `anscombe` dataset to a matrix format.
* class(ans.mx)
	+ # Checks the class of `ans.mx`.
* ans.df <- as.data.frame(ans.mx)
	+ # Converts the matrix back to a data frame.
* class(ans.df)
	+ # Checks the class of the resulting data frame.
* summary(ans.df)
	+ # Provides a summary of the data frame.
* ans.ls <- as.list(anscombe)
	+ # Converts the `anscombe` dataset to a list format.
* class(ans.ls)
	+ # Checks the class of the list.

**Custom Matrix and Histogram**

* new.mx <- matrix(runif(1000), nrow = 10, ncol = 100)
	+ # Generates a 10x100 matrix filled with 1000 random numbers from a uniform distribution.
* hist(new.mx[1, ], breaks = 9, main = "First row of matrix")
	+ # Plots a histogram of the first row of the matrix.
* cs <- colSums(new.mx)
	+ # Computes column sums of the matrix.
* hist(cs, breaks = 9, main = "Histogram of Column Sums", xlab = "Column sums", ylab = "Frequency")
	+ # Plots a histogram of column sums.

**Simpson’s Paradox**

* Friend1 <- read.csv("mydata.csv")
* Friend2 <- read.csv("mydata-2.csv")
	+ # Reads two datasets from CSV files.
* cg <- rbind(Friend1, Friend2)
	+ # Combines the datasets by stacking rows.
* plot(Friend1$X, Friend1$Y, main = "Friend1 plot")
	+ # Plots `X` vs `Y` for `Friend1` with a title.
* abline(lm(Y ~ X, data = Friend1))
	+ # Adds a trend line to the plot for `Friend1`.
* plot(cg$X, cg$Y, main = "Combined plot")
	+ # Plots the combined dataset with a trend line to demonstrate Simpson’s Paradox.

**CODE**

* personality <- rnorm(100, 10, 4)
	+ # Generates a random sample of 100 values from a normal distribution with a mean of 10 and standard deviation of 4, representing "personality."
* looks <- rnorm(100, 10, 3)
	+ # Generates a random sample of 100 values from a normal distribution with a mean of 10 and standard deviation of 3, representing "looks."
* df.p1 <- data.frame(looks, personality)
	+ # Creates a data frame `df.p1` with "looks" as the first column and "personality" as the second column.
* plot(df.p1$looks, df.p1$personality)
	+ # Creates a scatterplot of "looks" (x-axis) against "personality" (y-axis).
* abline(15, -0.9)
	+ # Adds a line with intercept 15 and slope -0.9 to the scatterplot. This line could represent a "threshold" or "benchmark."
* cor(df.p1)
	+ # Computes the correlation matrix between "looks" and "personality," quantifying the linear relationship between these variables.

**Why It Works**:

* rnorm() draws values from a normal distribution.
* data.frame() organizes related variables into a tabular structure.
* plot() visually explores relationships between variables.
* cor() evaluates how strongly two variables are linearly related.

**What is a p-value**

This section explains the concept of p-values, which measure the probability of observing results as extreme as the data under the null hypothesis. It also explores how p-values behave under different conditions through simulation.

**Code:**

* nrow = 1000
* ncol = 6
* p.mat <- matrix(data = runif(nrow \* ncol), nrow = nrow, ncol = ncol)
	+ # Creates a 1000x6 matrix filled with random numbers uniformly distributed between 0 and 1.
* p.mat[, 4:6] <- runif(n = 3000, min = 0.5, max = 1.5)
	+ # Replaces columns 4 to 6 with random numbers between 0.5 and 1.5.
* p.out <- apply(p.mat, 1, function(x) as.numeric(t.test(x[1:3], x[4:6])$p.value))
	+ # For each row, performs a t-test comparing the first three columns with the last three columns and extracts the p-value.
* hist(p.out, breaks = 99, main = "Histogram of P value", xlab = "p-value", col = 'gray')
	+ # Plots a histogram of the p-values obtained from the t-tests.
* abline(v = 0.05, col = 'red', lty = 2, lwd = 3)
	+ # Adds a vertical red dashed line at 0.05, marking the threshold for significance.
* sum(p.out <= 0.05) / nrow
	+ # Calculates the proportion of p-values below the 0.05 significance threshold.
* min(p.out)
	+ # Finds the minimum p-value from the simulated tests.
* p.adjust(p.out1, method = "BH")
	+ This adjusts the p-values using the Benjamini-Hochberg procedure to control the False Discovery Rate (FDR).

**Comparing Sample Sizes**

By increasing sample size or the number of columns being compared, the code demonstrates how statistical power increases, making it easier to detect small differences.

**Key Observations:**

* Larger sample sizes lead to p-value distributions concentrating closer to significant thresholds.
* This is a demonstration of how increasing statistical power reduces false negatives.

**Benjamini-Hochberg Correction**

**Code:**

* min(p.adjust(p.out1))
	+ Adjusts p-values using the Benjamini-Hochberg (BH) method, which controls the FDR.
	+ Returns the minimum adjusted p-value, reflecting the most significant finding after correction.

**Why It Works**:

* The BH method ranks raw p-values, adjusting them to account for the multiple comparisons problem.
* Helps mitigate the risk of false discoveries in multiple testing scenarios.

How is a vector made

* Create a vector using c()
	+ numeric\_vector <- c(1, 2, 3, 4, 5)
	+ character\_vector <- c("apple", "banana", "cherry")

How is a matrix made

* matrix(data, nrow, ncol, byrow = TRUE)
	+ byrow= False if filing column wise
	+ mat <- matrix(1:9, nrow = 3, ncol = 3)
	+ 
* Can use cbind() to combine column or rbind to combine rows
	+ vec1 <- c(1, 2, 3)
	+ vec2 <- c(4, 5, 6)
	+ mat <- rbind(vec1, vec2)

How to subset a vector, matrix, or dataframe

Logic for a for loop

Logic of simple function