

VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

Ho Chi Minh City University of Technology

Faculty of Computer Science and Engineering



PROBABILITY AND STATISTICS (MT2013)

Assignment Report - Class CC03 - Group 07

***“ANOVA and Regression Analysis
of CPU’s Thermal Design Power:
Supplementary Document”***

Supervisor: PhD. Phan Thi Huong

Students: Le Kieu Anh Vu – 2353340
Do Thanh Tan – 2353073
Le Pham Tien Long – 2352688
Pham Quang Minh – 2352761
Nguyen Huu Minh Khoi - 2352614
Hoang Thuy Tram – 2353202

Ho Chi Minh City, December 2024

```
1 # Required Libraries
2 library(datasets)      # Standard R datasets and data manipulation functions
3 library(graphics)      # Basic plotting functions and graphical parameters
4 library(dplyr)         # Data manipulation and transformation (pipes, filters, grouping)
5 library(tidyr)         # Data tidying and reshaping functions
6 library(zoo)           # Time series and rolling window calculations
7 library(ggplot2)       # Advanced and elegant graphics generation
8 library(corrplot)      # Visualization of correlation matrices
9 library(readr)         # Fast and friendly data reading functions
10 library(FSA)           # For Dunn's test (post-hoc test for Kruskal-Wallis)
11 library(agricolae)     # Statistical analysis tools (Tukey's HSD, experimental designs)
12 library(caTools)       # Utility functions for data splitting and sampling
13 library(xgboost)       # Advanced gradient boosting machine learning algorithm
14
15 #-----
16 # SECTION 1: DATA PREPROCESSING
17 #-----
18
19 # 1.1 Import Raw Data
20 intelCPU <- read.csv("Intel_CPUs.csv")
21 head(intelCPU, 15)
22 cat("\n")
23 intelCPU <- read.csv("Intel_CPUs.csv", na.strings = c("", "N/A"))
24 head(intelCPU, 15)
25
26 # 1.2 Select Relevant Columns
27 cpuInfo <- intelCPU[, c("Product_Collection",
28                        "Vertical_Segment",
29                        "Status", "Launch_Date", "Lithography",
30                        "Recommended_Customer_Price", "nb_of_Cores",
31                        "nb_of_Threads", "Processor_Base_Frequency", "Cache", "TDP",
32                        "Max_Memory_Size", "Max_Memory_Bandwidth", "
33                        Graphics_Base_Frequency", "Graphics_Max_Dynamic_Frequency",
34                        "Instruction_Set")]
35 names(cpuInfo) #check if the wanted information was selected successfully
36 head(cpuInfo, 15) #check 15 first rows in file cpuInfo
37
38 # 1.3 Handle Missing Values
39 # Calculate the sum and ratio of NA in each column
40 print(colSums(is.na(cpuInfo)))
41 cat("\n")
42 print(apply(is.na(cpuInfo), 2, mean)*100)
43 cat("\n")
44 #Converting the ratio of NA into a data frame for easier plotting
45 naRatio <- apply(is.na(cpuInfo), 2, mean)*100
46 naData <- data.frame(
47   Column = names(naRatio),
48   Percentage = naRatio
49 )
50 #Creating a bar plot
51 ggplot(naData, aes(x = reorder(Column, -Percentage), y = Percentage)) +
52   geom_bar(stat = "identity", fill = "orange") +
53   labs(
54     title = "Percentage of Missing Values (NA) in Each Column",
55     x = "Columns",
56     y = "Percentage (%)"
57   ) + theme_minimal() +
58   theme(
59     axis.text.x = element_text(angle = 90, hjust = 1),
60     plot.title = element_text(hjust = 0.5)
61   )
62
63 # Delete NA in columns with NA ratio below 5% and delete columns with NA ratio above 50%
64 naRatio <- apply(is.na(cpuInfo), 2, mean)*100
65 checkCol <- names(naRatio[naRatio < 5])
66 cpuInfo <- cpuInfo[complete.cases(cpuInfo[, checkCol]), ]
67 removeCol <- names(naRatio[naRatio > 50])
68 cpuInfo <- cpuInfo[, !(names(cpuInfo) %in% removeCol)]
69
70 # Check the sum of NA and NA ratio in each column after filtering
71 print(colSums(is.na(cpuInfo)))
```

```
71 cat("\n")
72 print(apply(is.na(cpuInfo), 2, mean)*100)
73 cat("\n")
74 #Converting the ratio of NA into a data frame for easier plotting
75 naRatio <- apply(is.na(cpuInfo), 2, mean)*100
76 naData <- data.frame(
77   Column = names(naRatio),
78   Percentage = naRatio
79 )
80 #Creating a bar plot
81 ggplot(naData, aes(x = reorder(Column, -Percentage), y = Percentage)) +
82   geom_bar(stat = "identity", fill = "orange") +
83   labs(
84     title = "Percentage of Missing Values (NA) in Each Column",
85     x = "Columns",
86     y = "Percentage (%)"
87   ) + theme_minimal() +
88   theme(
89     axis.text.x = element_text(angle = 90, hjust = 1),
90     plot.title = element_text(hjust = 0.5)
91   )
92
93 # 1.4 Data Transformation
94 # Check the data type of each column before transformation
95 str(cpuInfo)
96 cat("\n")
97
98 #Launch_Date
99 year <- as.integer(substr(cpuInfo$Launch_Date, nchar(cpuInfo$Launch_Date) - 1, nchar(
100   cpuInfo$Launch_Date)))
101 cpuInfo$Launch_Date <- ifelse(year >= 90, 1900 + year, 2000 + year)
102
103 #Lithography
104 cpuInfo$Lithography <- as.integer(gsub(" nm", "", cpuInfo$Lithography))
105 colnames(cpuInfo)[which(colnames(cpuInfo) == "Lithography")] <- "Lithography (nm)"
106
107 #Recommended_Customer_Price
108 price <- function(x){
109   if (grepl("-", x)){
110     x <- strsplit(x, "-")[[1]]
111     return(mean(as.double(x)))
112   }
113   return(as.double(x))
114 }
115 cpuInfo$Recommended_Customer_Price <- gsub("\\$", "", cpuInfo$Recommended_Customer_Price)
116 cpuInfo$Recommended_Customer_Price <- gsub(",", "", cpuInfo$Recommended_Customer_Price)
117 cpuInfo$Recommended_Customer_Price <- sapply(cpuInfo$Recommended_Customer_Price, price)
118 colnames(cpuInfo)[which(colnames(cpuInfo) == "Recommended_Customer_Price")] <- "
  Recommended_Customer_Price (USD)"
119
120 #Processor_Base_Frequency
121 mhzFormat <- function(x){
122   if (grepl("M", x)){
123     return(as.double(gsub(" MHz", "", x)))
124   }
125   return(as.double(gsub(" GHz", "", x)) * 1000)
126 }
127 cpuInfo$Processor_Base_Frequency <- sapply(cpuInfo$Processor_Base_Frequency, mhzFormat)
128 colnames(cpuInfo)[which(colnames(cpuInfo) == "Processor_Base_Frequency")] <- "
  Processor_Base_Frequency (MHz)"
129
130 #Cache
131 mbFormat <- function(x){
132   if (grepl("M", x)){
133     return(as.double(gsub(" M", "", x)))
134   }
135   return(as.double(gsub(" K", "", x)) / 1024)
136 }
137 cpuInfo <- separate(cpuInfo, Cache, into = c("Cache_Size (MB)", "Cache_Type"), sep = "B")
138 cpuInfo$`Cache_Size (MB)` <- sapply(cpuInfo$`Cache_Size (MB)`, mbFormat)
139 cpuInfo$Cache_Type <- ifelse(cpuInfo$Cache_Type == "", "Original", sub(" ", "",
```

```
cpuInfo$Cache_Type))
139
140 #TDP
141 cpuInfo$TDP <- as.double(gsub(" W", "", cpuInfo$TDP))
142 colnames(cpuInfo)[which(colnames(cpuInfo) == "TDP")] <- "TDP (Watts)"
143
144 #Max_Memory_Size
145 gbFormat <- function(x){
146   if (grepl("G", x)){
147     return(as.double(gsub(" GB", "", x)))
148   }
149   return(as.double(gsub(" TB", "", x)) * 1024)
150 }
151 cpuInfo$Max_Memory_Size <- sapply(cpuInfo$Max_Memory_Size, gbFormat)
152 colnames(cpuInfo)[which(colnames(cpuInfo) == "Max_Memory_Size")] <- "Max_Memory_Size (GB)"
153
154 #Max_Memory_Bandwidth
155 cpuInfo$Max_Memory_Bandwidth <- as.double(gsub(" GB/s", "", cpuInfo$Max_Memory_Bandwidth)
156 )
157 colnames(cpuInfo)[which(colnames(cpuInfo) == "Max_Memory_Bandwidth")] <- "
158   Max_Memory_Bandwidth (GB/s)"
159
160 #Instruction_Set
161 cpuInfo$Instruction_Set <- gsub("Itanium ", "", cpuInfo$Instruction_Set)
162 cpuInfo$Instruction_Set <- gsub("-bit", "", cpuInfo$Instruction_Set)
163 cpuInfo$Instruction_Set <- as.integer(cpuInfo$Instruction_Set)
164 colnames(cpuInfo)[which(colnames(cpuInfo) == "Instruction_Set")] <- "Instruction_Set (bit)"
165
166 # Check the data type of each column after transformation
167 str(cpuInfo)
168 head(cpuInfo, 15)
169
170 # 1.5. Filling missing values
171 # Filling in missing values in Recommended_Customer_Price column
172 cpuInfo <- cpuInfo %>%
173   group_by(Product_Collection) %>% fill(`Recommended_Customer_Price (USD)`, .direction
174     = "updown") %>%
175   group_by(Vertical_Segment) %>% fill(`Recommended_Customer_Price (USD)`, .direction =
176     "updown")
177
178 # Filling in missing values in Launch_Date column
179 cpuInfo <- cpuInfo %>%
180   group_by(Product_Collection) %>% fill(Launch_Date, .direction = "downup") %>%
181   group_by(Vertical_Segment) %>% fill(Launch_Date, .direction = "updown")
182
183 # Filling in missing values in Instruction_Set column
184 mode <- function(x){
185   uniq <- unique(x)
186   uniq[which.max(tabulate(match(x, uniq)))]
187 }
188 cpuInfo$`Instruction_Set (bit)`[is.na(cpuInfo$`Instruction_Set (bit)`)] <- mode(cpuInfo$`
189   Instruction_Set (bit)`
190
191 # Filling in missing values in other un-checked columns
192 cpuInfo$nb_of_Threads <- ifelse(is.na(cpuInfo$nb_of_Threads), cpuInfo$nb_of_Cores * 2,
193   cpuInfo$nb_of_Threads)
194 cpuInfo$`Max_Memory_Size (GB)`[is.na(cpuInfo$`Max_Memory_Size (GB)`)] <- median(cpuInfo$`
195   Max_Memory_Size (GB)` , na.rm = TRUE)
196 cpuInfo$`Max_Memory_Bandwidth (GB/s)`[is.na(cpuInfo$`Max_Memory_Bandwidth (GB/s)`)] <-
197   median(cpuInfo$`Max_Memory_Bandwidth (GB/s)` , na.rm = TRUE)
198
199 # Checking if there are still any NAs that haven't been filtered yet
200 print(colSums(is.na(cpuInfo))) #total number of NAs in each column
201 cat("\n")
202 # Converting the ratio of NA into a data frame for easier plotting
203 naRatio <- apply(is.na(cpuInfo), 2, mean)*100
204 naData <- data.frame(
205   Column = names(naRatio),
206   Percentage = naRatio
207 )
```

```

199 )
200 # Creating a bar plot
201 ggplot(naData, aes(x = reorder(Column, -Percentage), y = Percentage)) +
202   geom_bar(stat = "identity", fill = "orange") +
203   labs(
204     title = "Percentage of Missing Values (NA) in Each Column",
205     x = "Columns",
206     y = "Percentage (%)"
207   ) + theme_minimal() +
208   theme(
209     axis.text.x = element_text(angle = 90, hjust = 1),
210     plot.title = element_text(hjust = 0.5)
211   )
212
213 # Check if we have filtered successfully
214 head(cpuInfo, 15)
215 cat("\n")
216
217 # 1.6 Data Storing
218 # Choose only columns in int & num values for storing
219 cpuFinal <- subset(cpuInfo, select = -c(Product_Collection, Vertical_Segment, Status,
220   Cache_Type))
221 cat("\n")
222 # Check the data type of each column before storing
223 str(cpuFinal)
224 cat("\n")
225 # Store the cleaned data into a new CSV file
226 write_csv(cpuFinal, "Intel_CPUs_cleaned.csv")
227
228 #-----
229 # SECTION 2: DESCRIPTIVE STATISTICS
230 #-----
231 # 2.1. Import cleaned data and print statistical summary
232 data_descriptive <- read_csv("Intel_CPUs_cleaned.csv")
233 # Print statistical summary
234 print(summary(data_descriptive))
235
236 # 2.2. Generate histograms and boxplots for each numerical variable
237 #Launch_date
238 hist(
239   cpuFinal$Launch_Date, #plot launch_date variable
240   main = "Histogram of CPU Launch Date's distribution", #graph name
241   xlab = "Launch Date", #x value
242   border = "black"
243 )
244 boxplot(
245   cpuFinal$Launch_Date, #plot launch_date variable
246   main = "Boxplot of Launch Date", #graph name
247   ylab = "Year", #x value
248   col = "grey"
249 )
250
251 #Lithography
252 hist(
253   cpuFinal$Lithography (nm)`,
254   main = "Histogram of Lithography",
255   xlab = "Lithography (nm)",
256   border = "black"
257 )
258 boxplot(
259   cpuFinal$Lithography (nm)`,
260   main = "Boxplot of Lithography",
261   ylab = "Lithography (nm)",
262   col = "grey"
263 )
264
265 #Recommended_customer_price
266 hist(
267   cpuFinal$Recommended_Customer_Price (USD)`,
268   main = "Histogram of Recommended Customer Price",
269   xlab = "Price (USD)",

```

```
269     border = "black"
270 )
271 boxplot(
272     cpuFinal$`Recommended_Customer_Price (USD)`,
273     main = "Boxplot of Recommended Customer Price",
274     ylab = "Price (USD)",
275     col = "grey"
276 )
277
278 #nb_of_Cores
279 hist(
280     cpuFinal$`nb_of_Cores`,
281     main = "Histogram of nb_of_Cores",
282     xlab = "Number of Cores",
283     border = "black"
284 )
285 boxplot(
286     cpuFinal$`nb_of_Cores`,
287     main = "Boxplot of nb_of_Cores",
288     ylab = "Number of Cores",
289     col = "grey"
290 )
291
292 #nb_of_Threads
293 hist(
294     cpuFinal$`nb_of_Threads`,
295     main = "Histogram of nb_of_Threads",
296     xlab = "Number of Threads",
297     border = "black"
298 )
299 boxplot(
300     cpuFinal$`nb_of_Threads`,
301     main = "Boxplot of nb_of_Threads",
302     ylab = "Number of Threads",
303     col = "grey"
304 )
305
306 #Processor_base_frequency
307 hist(
308     cpuFinal$`Processor_Base_Frequency (MHz)`,
309     main = "Histogram of Processor Base Frequency",
310     xlab = "Frequency (MHz)",
311     border = "black"
312 )
313 boxplot(
314     cpuFinal$`Processor_Base_Frequency (MHz)`,
315     main = "Boxplot of Processor Base Frequency",
316     ylab = "Frequency (MHz)",
317     col = "grey"
318 )
319
320 #TDP
321 hist(
322     cpuFinal$`TDP (Watts)`,
323     main = "Histogram of TDP (Watts)",
324     xlab = "Power Consumption (Watts)",
325     border = "black"
326 )
327 boxplot(
328     cpuFinal$`TDP (Watts)`,
329     main = "Boxplot of TDP",
330     ylab = "Power Consumption (Watts)",
331     col = "grey"
332 )
333
334 #Cache_size
335 hist(
336     cpuFinal$`Cache_Size (MB)`,
337     main = "Histogram of Cache Size (MB)",
338     xlab = "Cache Size (MB)",
339     border = "black"
```

```
340 )
341 boxplot(
342   cpuFinal$`Cache_Size (MB)` ,
343   main = "Boxplot of Cache Size (MB)",
344   ylab = "Cache Size (MB)",
345   col = "grey"
346 )
347
348 #Max_memory_bandwidth
349 hist(
350   cpuFinal$`Max_Memory_Bandwidth (GB/s)` ,
351   main = "Histogram of Max Memory Bandwidth (GB/s)",
352   xlab = "Bandwidth (GB/s)",
353   border = "black"
354 )
355 boxplot(
356   cpuFinal$`Max_Memory_Bandwidth (GB/s)` ,
357   main = "Boxplot of Max Memory Bandwidth (GB/s)",
358   ylab = "Bandwidth (GB/s)",
359   col = "grey"
360 )
361
362 #Correlation plot
363 cpufinal = cor(cpuFinal)
364 corrrplot(
365   cpufinal, method = "color", #add square
366   tl.cex = 0.7, #change text size
367   number.cex = 0.7, #change number size
368   col = colorRampPalette(c("green","white","red"))(100), #change color
369   addCoef.col = "black" #add numbers
370 )
371
372 # 2.3 Analyse TDP by Lithography level
373 # Calculate and display TDP summary statistics by lithography
374 data_TDP <- data_descriptive %>%
375   group_by(Lithography..nm.) %>%
376   summarize(
377     `5% Quantile` = quantile(TDP..Watts., probs = 0.05, na.rm = TRUE),
378     `95% Quantile` = quantile(TDP..Watts., probs = 0.95, na.rm = TRUE),
379     Mean = mean(TDP..Watts., na.rm = TRUE),
380     SD = sd(TDP..Watts., na.rm = TRUE)
381   )
382 print(data_TDP)
383
384 # Define x-axis positions
385 x_positions <- seq_along(data_TDP$Lithography..nm.)
386 # Create the plot
387 plot(
388   x_positions, data_TDP$Mean, type = "o",
389   col = "blue", ylim = range(c(data_TDP$Mean, data_TDP$SD)),
390   xlab = "Lithography (nm)", ylab = "Value",
391   xaxt = "n", main = "Trends in Mean and Standard Deviation of TDP"
392 )
393 lines(x_positions, data_TDP$SD, type = "o", col = "red")
394 points(x_positions, data_TDP$SD, col = "red")
395 # Customize the x-axis
396 axis(1, at = x_positions, labels = data_TDP$Lithography..nm., las = 2)
397 # Add legend
398 legend(
399   "topright", legend = c("Mean", "SD"), col = c("blue", "red"),
400   lty = 1, pch = 1, bty = "n"
401 )
402
403 #-----
404 # SECTION 3: INFERENTIAL STATISTICS
405 #-----
406 # 3.1 One-way ANOVA
407 # Import data and create a factor variable for Lithography
408 data_anova <- read.csv("Intel_CPUs_cleaned.csv")
409 data_anova$Lithography..nm. <- as.factor(data_anova$Lithography..nm.)
410
```

```
411 # Perform one-way ANOVA
412 litho_anova_model <- aov(TDP..Watts. ~ Lithography..nm., data = data_anova)
413 print(summary(litho_anova_model))
414
415 # Normality test of residuals
416 shapiro_test_result <- shapiro.test(residuals(litho_anova_model))
417 print(shapiro_test_result)
418
419 # Homoscedasticity test of residuals
420 bartlett_test_result <- bartlett.test(TDP..Watts. ~ Lithography..nm., data = data_anova)
421 print(bartlett_test_result)
422
423 # Post-hoc test (Tukey's HSD)
424 tukey_result <- TukeyHSD(litho_anova_model)
425 print(tukey_result)
426 plot(tukey_result, las = 1)
427
428 # 3.2 Multiple Linear Regression
429 # Import data and remove outliers
430 data_regression <- read.csv("Intel_CPUs_cleaned.csv")
431 remove_outliers <- function(df, column) {
432   Q1 <- quantile(df[[column]], 0.25) # First quartile
433   Q3 <- quantile(df[[column]], 0.75) # Third quartile
434   IQR <- Q3 - Q1                    # Interquartile range
435   lower_bound <- Q1 - 1.5 * IQR      # Lower bound
436   upper_bound <- Q3 + 1.5 * IQR      # Upper bound
437   return(df[df[[column]] >= lower_bound & df[[column]] <= upper_bound, ])
438 }
439 data_regression <- remove_outliers(data_regression, "TDP..Watts.")
440
441 # Split data into training and testing sets
442 set.seed(123)
443 split <- sample.split(data_regression, SplitRatio = 0.8)
444 training_set <- subset(data_regression, split == TRUE)
445 test_set <- subset(data_regression, split == FALSE)
446
447 # Fit the multiple linear regression model
448 regressor <- lm(formula = TDP..Watts. ~ Launch_Date +
449                 Recommended_Customer_Price..USD. +
450                 Lithography..nm. +
451                 nb_of_Cores +
452                 Processor_Base_Frequency..MHz. +
453                 Cache_Size..MB. +
454                 Max_Memory_Size..GB. +
455                 Max_Memory_Bandwidth..GB.s. +
456                 Instruction_Set..bit.,
457                 data = training_set)
458 print(summary(regressor))
459
460 # Assumptions testing
461 plot(regressor)
462
463 # Perform predictions on the test set
464 y_pred <- predict(regressor, newdata = test_set)
465 # Calculate performance metrics
466 MAE <- mean(abs(y_pred - test_set$TDP..Watts.))
467 MSE <- mean((y_pred - test_set$TDP..Watts.)^2)
468 RMSE <- sqrt(MSE)
469 # Print metrics
470 cat("MAE:", round(MAE, 2), "\n")
471 cat("MSE:", round(MSE, 2), "\n")
472 cat("RMSE:", round(RMSE, 2), "\n")
473 # Scatter plot of actual vs predicted values
474 plot(test_set$TDP..Watts., y_pred,
475      col = "blue",
476      pch = 16,
477      xlab = "Actual TDP (Watts)",
478      ylab = "Predicted TDP (Watts)",
479      main = "Actual vs Predicted Values")
480 abline(0, 1, col = "red", lty = 2) # Diagonal for perfect prediction
481 legend("topleft", legend = "Perfect Prediction", col = "red", lty = 2)
```



```
482
483 #-----
484 # SECTION 4: EXTENSIONS
485 #-----
486 # 4.1. Non-parametric ANOVA test
487 # Import data and create a factor variable for Lithography
488 data_anova_extension <- read.csv("Intel_CPUs_cleaned.csv")
489 data_anova_extension$Lithography..nm. <- as.factor(data_anova$Lithography..nm.)
490
491 # Compare TDP across different Lithography groups
492 kruskal_test_result <- kruskal.test(TDP..Watts. ~ Lithography..nm., data =
  data_anova_extension)
493 print(kruskal_test_result)
494
495 # Post-hoc test (Dunn's test)
496 dunn_test_result <- dunnTest(TDP..Watts. ~ Lithography..nm., data = data_anova_extension,
  method = "bonferroni")
497 print(dunn_test_result)
498
499 # 4.2. Machine Learning Model for regression
500 # Import data and remove outliers
501 data_regression_extension <- read.csv("Intel_CPUs_cleaned.csv")
502 remove_outliers <- function(df, column) {
503   Q1 <- quantile(df[[column]], 0.25) # First quartile
504   Q3 <- quantile(df[[column]], 0.75) # Third quartile
505   IQR <- Q3 - Q1                    # Interquartile range
506   lower_bound <- Q1 - 1.5 * IQR      # Lower bound
507   upper_bound <- Q3 + 1.5 * IQR      # Upper bound
508   return(df[df[[column]] >= lower_bound & df[[column]] <= upper_bound, ])
509 }
510 data_regression_extension <- remove_outliers(data_regression_extension, "TDP..Watts.")
511
512 # Split dataset into training and testing sets
513 set.seed(123)
514 split <- sample.split(data_regression_extension, SplitRatio = 0.8)
515 training_set <- subset(data_regression_extension, split == TRUE)
516 test_set <- subset(data_regression_extension, split == FALSE)
517
518 # Convert data to matrices, as XGBoost requires matrix inputs
519 train_matrix <- as.matrix(training_set[, c("Launch_Date",
520                                             "Recommended_Customer_Price..USD.",
521                                             "Lithography..nm.",
522                                             "nb_of_Cores",
523                                             "Processor_Base_Frequency..MHz.",
524                                             "Cache_Size..MB.",
525                                             "Max_Memory_Size..GB.",
526                                             "Max_Memory_Bandwidth..GB.s.",
527                                             "Instruction_Set..bit.")])
528 train_labels <- training_set$TDP..Watts.
529
530 test_matrix <- as.matrix(test_set[, c("Launch_Date",
531                                       "Recommended_Customer_Price..USD.",
532                                       "Lithography..nm.",
533                                       "nb_of_Cores",
534                                       "Processor_Base_Frequency..MHz.",
535                                       "Cache_Size..MB.",
536                                       "Max_Memory_Size..GB.",
537                                       "Max_Memory_Bandwidth..GB.s.",
538                                       "Instruction_Set..bit.")])
539 test_labels <- test_set$TDP..Watts.
540
541 # Train an XGBoost regression model
542 xgb_model <- xgboost(data = train_matrix,
543                      label = train_labels,
544                      nrounds = 100, # Number of boosting rounds
545                      objective = "reg:squarederror", # Regression task
546                      eta = 0.1, # Learning rate
547                      max_depth = 6, # Tree depth
548                      subsample = 0.8, # Subsample ratio for training data
549                      colsample_bytree = 0.8, # Subsample ratio for features
550                      verbose = 0) # Suppress output
```

```
551
552 # Make predictions on the test set
553 y_pred <- predict(xgb_model, newdata = test_matrix)
554 # Calculate performance metrics
555 MAE <- mean(abs(y_pred - test_labels))
556 MSE <- mean((y_pred - test_labels)^2)
557 RMSE <- sqrt(MSE)
558 # Print metrics
559 cat("MAE:", round(MAE, 2), "\n")
560 cat("MSE:", round(MSE, 2), "\n")
561 cat("RMSE:", round(RMSE, 2), "\n")
562 # Scatter plot of actual vs predicted values
563 plot(test_labels, y_pred,
564       col = "blue",
565       pch = 16,
566       xlab = "Actual TDP (Watts)",
567       ylab = "Predicted TDP (Watts)",
568       main = "Actual vs Predicted Values")
569 abline(0, 1, col = "red", lty = 2) # Diagonal for perfect prediction
570 legend("topleft", legend = "Perfect Prediction", col = "red", lty = 2)
571
572 # Examine feature importance
573 importance_matrix <- xgb.importance(model = xgb_model, feature_names = colnames(
574   train_matrix))
575 print(importance_matrix)
576 xgb.plot.importance(importance_matrix)
```

Listing 1: R code implemented for the report