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

```r
# Required Libraries
library(datasets)     # Standard R datasets and data manipulation functions
library(graphics)     # Basic plotting functions and graphical parameters
library(dplyr)        # Data manipulation and transformation (pipes, filters, grouping)
library(tidyr)        # Data tidying and reshaping functions
library(zoo)          # Time series and rolling window calculations
library(ggplot2)      # Advanced and elegant graphics generation
library(corrplot)     # Visualization of correlation matrices
library(readr)        # Fast and friendly data reading functions
library(FSA)          # For Dunn's test (post-hoc test for Kruskal-Wallis)
library(agricolae)    # Statistical analysis tools (Tukey's HSD, experimental designs)
library(caTools)      # Utility functions for data splitting and sampling
library(xgboost)      # Advanced gradient boosting machine learning algorithm

#---------------------------------------------------------
# SECTION 1: DATA PREPROCESSING
#---------------------------------------------------------

# 1.1 Import Raw Data
intelCPU <- read.csv("Intel_CPUs.csv")
head(intelCPU, 15)
cat("\n")
intelCPU <- read.csv("Intel_CPUs.csv", na.strings = c("", "N/A"))
head(intelCPU, 15)

# 1.2 Select Relevant Columns
cpuInfo <- intelCPU[, c("Product_Collection",
                        "Vertical_Segment",
                        "Status", "Launch_Date", "Lithography",
                        "Recommended_Customer_Price", "nb_of_Cores",
                        "nb_of_Threads", "Processor_Base_Frequency", "Cache", "TDP",
                        "Max_Memory_Size", "Max_Memory_Bandwidth", "Graphics_Base_Frequency", "Graphics_Max_Dynamic_Frequency",
                        "Instruction_Set")]
names(cpuInfo) #check if the wanted information was selected successfully
head(cpuInfo, 15) #check 15 first rows in file cpuInfo

# 1.3 Handle Missing Values
# Calculate the sum and ratio of NA in each collumn
print(colSums(is.na(cpuInfo)))
cat("\n")
print(apply(is.na(cpuInfo), 2, mean)*100)
cat("\n")
#Converting the ratio of NA into a data frame for easier plotting
naRatio <- apply(is.na(cpuInfo), 2, mean)*100
naData <- data.frame(
    Column = names(naRatio),
    Percentage = naRatio
)
#Creating a bar plot
ggplot(naData, aes(x = reorder(Column, -Percentage), y = Percentage)) +
    geom_bar(stat = "identity", fill = "orange") +
    labs(
        title = "Percentage of Missing Values (NA) in Each Column",
        x = "Columns",
        y = "Percentage (%)"
    ) + theme_minimal() +
    theme(
        axis.text.x = element_text(angle = 90, hjust = 1),
        plot.title = element_text(hjust = 0.5)
    )

# Delete NA in columns with NA ratio below 5% and delete columns with NA ratio above 50%
naRatio <- apply(is.na(cpuInfo), 2, mean)*100
checkCol <- names(naRatio[naRatio < 5])
cpuInfo <- cpuInfo[complete.cases(cpuInfo[, checkCol]), ]
removeCol <- names(naRatio[naRatio > 50])
cpuInfo <- cpuInfo[, !(names(cpuInfo) %in% removeCol)]

# Check the sum of NA and NA ratio in each column after filtering
print(colSums(is.na(cpuInfo)))
```

```r
cat("\n")
print(apply(is.na(cpuInfo), 2, mean)*100)
cat("\n")
#Converting the ratio of NA into a data frame for easier plotting
naRatio <- apply(is.na(cpuInfo), 2, mean)*100
naData <- data.frame(
    Column = names(naRatio),
    Percentage = naRatio
)
#Creating a bar plot
ggplot(naData, aes(x = reorder(Column, -Percentage), y = Percentage)) +
    geom_bar(stat = "identity", fill = "orange") +
    labs(
        title = "Percentage of Missing Values (NA) in Each Column",
        x = "Columns",
        y = "Percentage (%)"
    ) + theme_minimal() +
    theme(
        axis.text.x = element_text(angle = 90, hjust = 1),
        plot.title = element_text(hjust = 0.5)
    )

# 1.4 Data Transformation
# Check the data type of each column before transformation
str(cpuInfo)
cat("\n")

#Launch_Date
year <- as.integer(substr(cpuInfo$Launch_Date, nchar(cpuInfo$Launch_Date) - 1, nchar(cpuInfo$Launch_Date)))
cpuInfo$Launch_Date <- ifelse(year >= 90, 1900 + year, 2000 + year)

#Lithography
cpuInfo$Lithography <- as.integer(gsub(" nm", "", cpuInfo$Lithography))
colnames(cpuInfo)[which(colnames(cpuInfo) == "Lithography")] <- "Lithography (nm)"

#Recommended_Customer_Price
price <- function(x){
  if (grepl("-", x)){
    x <- strsplit(x, "-")[[1]]
    return(mean(as.double(x)))
  }
  return(as.double(x))
}
cpuInfo$Recommended_Customer_Price <- gsub("\\$", "", cpuInfo$Recommended_Customer_Price)
cpuInfo$Recommended_Customer_Price <- gsub(",", "", cpuInfo$Recommended_Customer_Price)
cpuInfo$Recommended_Customer_Price <- sapply(cpuInfo$Recommended_Customer_Price, price)
colnames(cpuInfo)[which(colnames(cpuInfo) == "Recommended_Customer_Price")] <- "Recommended_Customer_Price (USD)"

#Processor_Base_Frequency
mhzFormat <- function(x){
  if (grepl("M", x)){
    return(as.double(gsub(" MHz", "", x)))
  }
  return(as.double(gsub(" GHz", "", x)) * 1000)
}
cpuInfo$Processor_Base_Frequency <- sapply(cpuInfo$Processor_Base_Frequency, mhzFormat)
colnames(cpuInfo)[which(colnames(cpuInfo) == "Processor_Base_Frequency")] <- "Processor_Base_Frequency (MHz)"

#Cache
mbFormat <- function(x){
  if (grepl("M", x)){
    return(as.double(gsub(" M", "", x)))
  }
  return(as.double(gsub(" K", "", x)) / 1024)
}
cpuInfo <- separate(cpuInfo, Cache, into = c("Cache_Size (MB)", "Cache_Type"), sep = "B")
cpuInfo$`Cache_Size (MB)` <- sapply(cpuInfo$`Cache_Size (MB)`, mbFormat)
cpuInfo$Cache_Type <- ifelse(cpuInfo$Cache_Type == "", "Original", sub(" ", "",
```

```r
    cpuInfo$Cache_Type))

#TDP
cpuInfo$TDP <- as.double(gsub(" W", "", cpuInfo$TDP))
colnames(cpuInfo)[which(colnames(cpuInfo) == "TDP")] <- "TDP (Watts)"

#Max_Memory_Size
gbFormat <- function(x){
  if (grepl("G", x)){
    return(as.double(gsub(" GB", "", x)))
  }
  return(as.double(gsub(" TB", "", x)) * 1024)
}
cpuInfo$Max_Memory_Size <- sapply(cpuInfo$Max_Memory_Size, gbFormat)
colnames(cpuInfo)[which(colnames(cpuInfo) == "Max_Memory_Size")] <- "Max_Memory_Size (GB)
    "

#Max_Memory_Bandwidth
cpuInfo$Max_Memory_Bandwidth <- as.double(gsub(" GB/s", "", cpuInfo$Max_Memory_Bandwidth)
    )
colnames(cpuInfo)[which(colnames(cpuInfo) == "Max_Memory_Bandwidth")] <- "
    Max_Memory_Bandwidth (GB/s)"

#Instruction_Set
cpuInfo$Instruction_Set <- gsub("Itanium ", "", cpuInfo$Instruction_Set)
cpuInfo$Instruction_Set <- gsub("-bit", "", cpuInfo$Instruction_Set)
cpuInfo$Instruction_Set <- as.integer(cpuInfo$Instruction_Set)
colnames(cpuInfo)[which(colnames(cpuInfo) == "Instruction_Set")] <- "Instruction_Set (bit
    )"

# Check the data type of each column after transformation
str(cpuInfo)
head(cpuInfo, 15)

# 1.5. Filling missing values
# Filling in missing values in Recommended_Customer_Price column
cpuInfo <- cpuInfo %>%
    group_by(Product_Collection) %>% fill(`Recommended_Customer_Price (USD)`, .direction
        = "updown") %>%
    group_by(Vertical_Segment) %>% fill(`Recommended_Customer_Price (USD)`, .direction =
        "updown")

# Filling in missing values in Launch_Date column
cpuInfo <- cpuInfo %>%
    group_by(Product_Collection) %>% fill(Launch_Date, .direction = "downup") %>%
    group_by(Vertical_Segment) %>% fill(Launch_Date, .direction = "updown")

# Filling in missing values in Instruction_Set column
mode <- function(x){
    uniq <- unique(x)
    uniq[which.max(tabulate(match(x, uniq)))]
}
cpuInfo$`Instruction_Set (bit)`[is.na(cpuInfo$`Instruction_Set (bit)`)] <- mode(cpuInfo$`
    Instruction_Set (bit)`)

# Filling in missing values in other un-checked columns
cpuInfo$nb_of_Threads <- ifelse(is.na(cpuInfo$nb_of_Threads), cpuInfo$nb_of_Cores * 2,
    cpuInfo$nb_of_Threads)
cpuInfo$`Max_Memory_Size (GB)`[is.na(cpuInfo$`Max_Memory_Size (GB)`)] <- median(cpuInfo$`
    Max_Memory_Size (GB)`, na.rm = TRUE)
cpuInfo$`Max_Memory_Bandwidth (GB/s)`[is.na(cpuInfo$`Max_Memory_Bandwidth (GB/s)`)] <-
    median(cpuInfo$`Max_Memory_Bandwidth (GB/s)`, na.rm = TRUE)

# Checking if there are still any NAs that haven't been filtered yet
print(colSums(is.na(cpuInfo))) #total number of NAs in each column
cat("\n")
# Converting the ratio of NA into a data frame for easier plotting
naRatio <- apply(is.na(cpuInfo), 2, mean)*100
naData <- data.frame(
    Column = names(naRatio),
    Percentage = naRatio
```

```
)
# Creating a bar plot
ggplot(naData, aes(x = reorder(Column, -Percentage), y = Percentage)) +
    geom_bar(stat = "identity", fill = "orange") +
    labs(
        title = "Percentage of Missing Values (NA) in Each Column",
        x = "Columns",
        y = "Percentage (%)"
    ) + theme_minimal() +
    theme(
        axis.text.x = element_text(angle = 90, hjust = 1),
        plot.title = element_text(hjust = 0.5)
    )

# Check if we have filtered successfully
head(cpuInfo, 15)
cat("\n")

# 1.6 Data Storing
# Choose only columns in int & num values for stroring
cpuFinal <- subset(cpuInfo, select = -c(Product_Collection, Vertical_Segment, Status,
    Cache_Type))
cat("\n")
# Check the data type of each column before storing
str(cpuFinal)
cat("\n")
# Store the cleaned data into a new CSV file
write_csv(cpuFinal, "Intel_CPUs_cleaned.csv")

#---------------------------------------------------------
# SECTION 2: DESCRIPTIVE STATISTICS
#---------------------------------------------------------
# 2.1. Import cleaned data and print statistical summary
data_descritive <- read.csv("Intel_CPUs_cleaned.csv")
# Print statistical summary
print(summary(data_descritive))

# 2.2. Generate histograms and boxplots for each numerical variable
#Launch_date
hist(
  cpuFinal$Launch_Date, #plot launch_date variable
  main = "Histogram of CPU Launch Date's distribution", #graph name
  xlab = "Launch Date", #x value
  border = "black"
)
boxplot(
  cpuFinal$Launch_Date, #plot launch_date variable
  main = "Boxplot of Launch Date", #graph name
  ylab = "Year", #x value
  col = "grey"
)

#Lithography
hist(
  cpuFinal$`Lithography (nm)`,
  main = "Histogram of Lithography",
  xlab = "Lithography (nm)",
  border = "black"
)
boxplot(
  cpuFinal$`Lithography (nm)`,
  main = "Boxplot of Lithography",
  ylab = "Lithography (nm)",
  col = "grey"
)

#Recommended_customer_price
hist(
  cpuFinal$`Recommended_Customer_Price (USD)`,
  main = "Histogram of Recommended Customer Price",
  xlab = "Price (USD)",
```

```r
  border = "black"
)
boxplot(
  cpuFinal$`Recommended_Customer_Price (USD)`,
  main = "Boxplot of Recommended Customer Price",
  ylab = "Price (USD)",
  col = "grey"
)

#nb_of_Cores
hist(
  cpuFinal$`nb_of_Cores`,
  main = "Histogram of nb_of_Cores",
  xlab = "Number of Cores",
  border = "black"
)
boxplot(
  cpuFinal$`nb_of_Cores`,
  main = "Boxplot of nb_of_Cores",
  ylab = "Number of Cores",
  col = "grey"
)

#nb_of_Threads
hist(
  cpuFinal$`nb_of_Threads`,
  main = "Histogram of nb_of_Threads",
  xlab = "Number of Threads",
  border = "black"
)
boxplot(
  cpuFinal$`nb_of_Threads`,
  main = "Boxplot of nb_of_Threads",
  ylab = "Number of Threads",
  col = "grey"
)

#Processor_base_frequency
hist(
  cpuFinal$`Processor_Base_Frequency (MHz)`,
  main = "Histogram of Processor Base Frequency",
  xlab = "Frequency (MHz)",
  border = "black"
)
boxplot(
  cpuFinal$`Processor_Base_Frequency (MHz)`,
  main = "Boxplot of Processor Base Frequency",
  ylab = "Frequency (MHz)",
  col = "grey"
)

#TDP
hist(
  cpuFinal$`TDP (Watts)`,
  main = "Histogram of TDP (Watts)",
  xlab = "Power Consumption (Watts)",
  border = "black"
)
boxplot(
  cpuFinal$`TDP (Watts)`,
  main = "Boxplot of TDP",
  ylab = "Power Consumption (Watts)",
  col = "grey"
)

#Cache_size
hist(
  cpuFinal$`Cache_Size (MB)`,
  main = "Histogram of Cache Size (MB)",
  xlab = "Cache Size (MB)",
  border = "black"
```

```r
)
boxplot(
  cpuFinal$`Cache_Size (MB)`,
  main = "Boxplot of Cache Size (MB)",
  ylab = "Cache Size (MB)",
  col = "grey"
)

#Max_memory_bandwidth
hist(
  cpuFinal$`Max_Memory_Bandwidth (GB/s)`,
  main = "Histogram of Max Memory Bandwidth (GB/s)",
  xlab = "Bandwidth (GB/s)",
  border = "black"
)
boxplot(
  cpuFinal$`Max_Memory_Bandwidth (GB/s)`,
  main = "Boxplot of Max Memory Bandwidth (GB/s)",
  ylab = "Bandwidth (GB/s)",
  col = "grey"
)

#Correlation plot
cpufinal = cor(cpuFinal)
corrplot(
  cpufinal, method = "color", #add square
  tl.cex = 0.7, #change text size
  number.cex = 0.7, #change number size
  col = colorRampPalette(c("green","white","red"))(100), #change color
  addCoef.col = "black" #add numbers
  )

# 2.3 Analyse TDP by Lithography level
# Calculate and display TDP summary statistics by lithography
data_TDP <- data_descritive %>%
  group_by(Lithography..nm.) %>%
  summarize(
    `5% Quantile` = quantile(TDP..Watts., probs = 0.05, na.rm = TRUE),
    `95% Quantile` = quantile(TDP..Watts., probs = 0.95, na.rm = TRUE),
    Mean = mean(TDP..Watts., na.rm = TRUE),
    SD = sd(TDP..Watts., na.rm = TRUE)
  )
print(data_TDP)

# Define x-axis positions
x_positions <- seq_along(data_TDP$Lithography..nm.)
# Create the plot
plot(
  x_positions, data_TDP$Mean, type = "o",
  col = "blue", ylim = range(c(data_TDP$Mean, data_TDP$SD)),
  xlab = "Lithography (nm)", ylab = "Value",
  xaxt = "n", main = "Trends in Mean and Standard Deviation of TDP"
)
lines(x_positions, data_TDP$SD, type = "o", col = "red")
points(x_positions, data_TDP$SD, col = "red")
# Customize the x-axis
axis(1, at = x_positions, labels = data_TDP$Lithography..nm., las = 2)
# Add legend
legend(
  "topright", legend = c("Mean", "SD"), col = c("blue", "red"),
  lty = 1, pch = 1, bty = "n"
)

#---------------------------------------------------------
# SECTION 3: INFERENTIAL STATISTICS
#---------------------------------------------------------
# 3.1 One-way ANOVA
# Import data and create a factor variable for Lithography
data_anova <- read.csv("Intel_CPUs_cleaned.csv")
data_anova$Lithography..nm. <- as.factor(data_anova$Lithography..nm.)

```

```
# Perform one-way ANOVA
litho_anova_model <- aov(TDP..Watts. ~ Lithography..nm., data = data_anova)
print(summary(litho_anova_model))

# Normality test of residuals
shapiro_test_result <- shapiro.test(residuals(litho_anova_model))
print(shapiro_test_result)

# Homoscadasticity test of residuals
bartlett_test_result <- bartlett.test(TDP..Watts. ~ Lithography..nm., data = data_anova)
print(bartlett_test_result)

# Post-hoc test (Tukey's HSD)
tukey_result <- TukeyHSD(litho_anova_model)
print(tukey_result)
plot(tukey_result, las = 1)

# 3.2 Multiple Linear Regression
# Import data and remove outliers
data_regression <- read.csv("Intel_CPUs_cleaned.csv")
remove_outliers <- function(df, column) {
  Q1 <- quantile(df[[column]], 0.25) # First quartile
  Q3 <- quantile(df[[column]], 0.75) # Third quartile
  IQR <- Q3 - Q1                     # Interquartile range
  lower_bound <- Q1 - 1.5 * IQR      # Lower bound
  upper_bound <- Q3 + 1.5 * IQR      # Upper bound
  return(df[df[[column]] >= lower_bound & df[[column]] <= upper_bound, ])
}
data_regression <- remove_outliers(data_regression, "TDP..Watts.")

# Split data into training and testing sets
set.seed(123)
split <- sample.split(data_regression, SplitRatio = 0.8)
training_set <- subset(data_regression, split == TRUE)
test_set <- subset(data_regression, split == FALSE)

# Fit the multiple linear regression model
regressor <- lm(formula = TDP..Watts. ~ Launch_Date +
                  Recommended_Customer_Price..USD. +
                  Lithography..nm. +
                  nb_of_Cores +
                  Processor_Base_Frequency..MHz. +
                  Cache_Size..MB. +
                  Max_Memory_Size..GB. +
                  Max_Memory_Bandwidth..GB.s. +
                  Instruction_Set..bit.,
                data = training_set)
print(summary(regressor))

# Assumptions testing
plot(regressor)

# Perform predictions on the test set
y_pred <- predict(regressor, newdata = test_set)
# Calculate performance metrics
MAE <- mean(abs(y_pred - test_set$TDP..Watts.))
MSE <- mean((y_pred - test_set$TDP..Watts.)^2)
RMSE <- sqrt(MSE)
# Print metrics
cat("MAE:", round(MAE, 2), "\n")
cat("MSE:", round(MSE, 2), "\n")
cat("RMSE:", round(RMSE, 2), "\n")
# Scatter plot of actual vs predicted values
plot(test_set$TDP..Watts., y_pred,
     col = "blue",
     pch = 16,
     xlab = "Actual TDP (Watts)",
     ylab = "Predicted TDP (Watts)",
     main = "Actual vs Predicted Values")
abline(0, 1, col = "red", lty = 2)  # Diagonal for perfect prediction
legend("topleft", legend = "Perfect Prediction", col = "red", lty = 2)
```

```r

#--------------------------------------------------------
# SECTION 4: EXTENSIONS
#--------------------------------------------------------
# 4.1. Non-parametric ANOVA test
# Import data and create a factor variable for Lithography
data_anova_extension <- read.csv("Intel_CPUs_cleaned.csv")
data_anova_extension$Lithography..nm. <- as.factor(data_anova$Lithography..nm.)

# Compare TDP across different Lithography groups
kruskal_test_result <- kruskal.test(TDP..Watts. ~ Lithography..nm., data =
    data_anova_extension)
print(kruskal_test_result)

# Post-hoc test (Dunn's test)
dunn_test_result <- dunnTest(TDP..Watts. ~ Lithography..nm., data = data_anova_extension,
     method = "bonferroni")
print(dunn_test_result)

# 4.2. Machine Learning Model for regression
# Import data and remove outliers
data_regression_extension <- read.csv("Intel_CPUs_cleaned.csv")
remove_outliers <- function(df, column) {
  Q1 <- quantile(df[[column]], 0.25) # First quartile
  Q3 <- quantile(df[[column]], 0.75) # Third quartile
  IQR <- Q3 - Q1                     # Interquartile range
  lower_bound <- Q1 - 1.5 * IQR      # Lower bound
  upper_bound <- Q3 + 1.5 * IQR      # Upper bound
  return(df[df[[column]] >= lower_bound & df[[column]] <= upper_bound, ])
}
data_regression_extension <- remove_outliers(data_regression_extension, "TDP..Watts.")

# Split dataset into training and testing sets
set.seed(123)
split <- sample.split(data_regression_extension, SplitRatio = 0.8)
training_set <- subset(data_regression_extension, split == TRUE)
test_set <- subset(data_regression_extension, split == FALSE)

# Convert data to matrices, as XGBoost requires matrix inputs
train_matrix <- as.matrix(training_set[, c("Launch_Date",
                                           "Recommended_Customer_Price..USD.",
                                           "Lithography..nm.",
                                           "nb_of_Cores",
                                           "Processor_Base_Frequency..MHz.",
                                           "Cache_Size..MB.",
                                           "Max_Memory_Size..GB.",
                                           "Max_Memory_Bandwidth..GB.s.",
                                           "Instruction_Set..bit.")])
train_labels <- training_set$TDP..Watts.

test_matrix <- as.matrix(test_set[, c("Launch_Date",
                                           "Recommended_Customer_Price..USD.",
                                           "Lithography..nm.",
                                           "nb_of_Cores",
                                           "Processor_Base_Frequency..MHz.",
                                           "Cache_Size..MB.",
                                           "Max_Memory_Size..GB.",
                                           "Max_Memory_Bandwidth..GB.s.",
                                           "Instruction_Set..bit.")])
test_labels <- test_set$TDP..Watts.

# Train an XGBoost regression model
xgb_model <- xgboost(data = train_matrix,
                     label = train_labels,
                     nrounds = 100,  # Number of boosting rounds
                     objective = "reg:squarederror",  # Regression task
                     eta = 0.1,  # Learning rate
                     max_depth = 6,  # Tree depth
                     subsample = 0.8,  # Subsample ratio for training data
                     colsample_bytree = 0.8,  # Subsample ratio for features
                     verbose = 0)  # Suppress output
```

```r

# Make predictions on the test set
y_pred <- predict(xgb_model, newdata = test_matrix)
# Calculate performance metrics
MAE <- mean(abs(y_pred - test_labels))
MSE <- mean((y_pred - test_labels)^2)
RMSE <- sqrt(MSE)
# Print metrics
cat("MAE:", round(MAE, 2), "\n")
cat("MSE:", round(MSE, 2), "\n")
cat("RMSE:", round(RMSE, 2), "\n")
# Scatter plot of actual vs predicted values
plot(test_labels, y_pred,
     col = "blue",
     pch = 16,
     xlab = "Actual TDP (Watts)",
     ylab = "Predicted TDP (Watts)",
     main = "Actual vs Predicted Values")
abline(0, 1, col = "red", lty = 2)  # Diagonal for perfect prediction
legend("topleft", legend = "Perfect Prediction", col = "red", lty = 2)

# Examine feature importance
importance_matrix <- xgb.importance(model = xgb_model, feature_names = colnames(
    train_matrix))
print(importance_matrix)
xgb.plot.importance(importance_matrix)
```

Listing 1: R code implemented for the report