**ML Tutorial**

**Introduction**

In this report, we analyze a dataset related to laptop prices using various machine learning algorithms. The dataset includes attributes such as brand, processor type, RAM size, storage capacity, GPU, screen size, resolution, battery life, weight, operating system, and price. The objective of this assignment is to apply both supervised and unsupervised learning and Ensemble Learning techniques to gain insights and predict outcomes based on these features.

To conduct this analysis, we will be implementing machine learning models using Weka, Python, and R programming. The tutorial focuses on eight different algorithms, categorized under supervised and unsupervised learning and Ensemble Learning .

**Supervised Learning Algorithms:**

1. **Decision Tree**: A hierarchical model that splits the dataset based on feature values to make predictions.
2. **Naive Bayes**: A probabilistic classifier based on Bayes' theorem, assuming independence between features.
3. **K-Nearest Neighbors (KNN)**: A distance-based classifier that assigns a class label based on the majority vote of its neighbors.
4. **Support Vector Machine (SVM):** A powerful algorithm that finds an optimal hyperplane to separate different classes.

**Unsupervised Learning Algorithm:**

1. **K-Means:** A clustering algorithm that groups data points into clusters based on similarity.

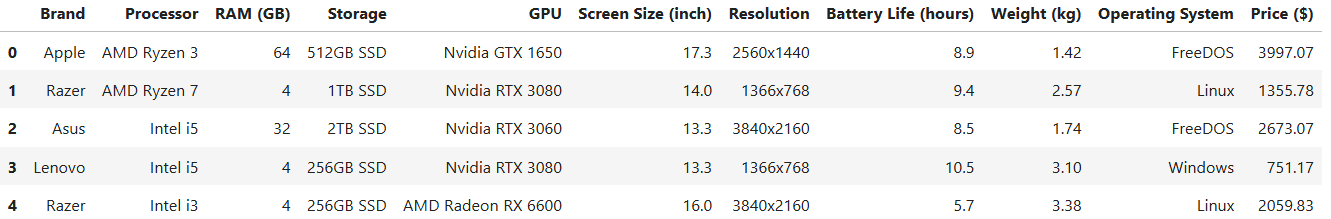
**Ensemble Learning Algorithms:**

1. **Random Forest**: An ensemble learning method that builds multiple decision trees and combines their predictions.
2. **AdaBoost**: A boosting algorithm that combines weak learners to improve classification performance.
3. **Stacking**: A meta-learning technique that combines multiple models to improve predictive accuracy.

By implementing these algorithms in different platforms, we aim to compare their performance, understand their strengths and weaknesses, and identify the best model for predicting laptop prices. The results will be evaluated based on various performance metrics such as accuracy, precision, recall, and F1-score for classification tasks.

This study will help in understanding the effectiveness of different machine learning approaches in analyzing real-world datasets and provide valuable insights into the factors influencing laptop prices.

**DATASET :**

****

**Q1 ) Decision Tree**

**R Programming**

**Code :**

# Load required libraries

library(rpart)

library(rpart.plot)

library(caret)

# Load dataset

file\_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop\_prices.csv" # Updated file path

data <- read.csv(file\_path)

# View dataset structure

str(data)

print(colnames(data)) # Check column names to avoid errors

# Handle column name issues

colnames(data) <- gsub("[^[:alnum:]\_]", "\_", colnames(data)) # Fix special characters in column names

# Convert categorical columns to factors

data$Brand <- as.factor(data$Brand)

data$Processor <- as.factor(data$Processor)

data$GPU <- as.factor(data$GPU)

data$Operating\_System <- as.factor(data$Operating\_System)

# Convert Storage to numeric

data$Storage <- as.numeric(gsub("[^0-9]", "", data$Storage))

# Convert Resolution to total pixel count

resolution\_split <- strsplit(as.character(data$Resolution), "x")

data$Total\_Pixels <- sapply(resolution\_split, function(x) as.numeric(x[1]) \* as.numeric(x[2]))

# Remove unnecessary columns

data <- subset(data, select = -c(Resolution))

# Remove missing values

data <- na.omit(data)

# Ensure target variable is numeric

data$Price <- as.numeric(data$Price)

# Split dataset into training and testing (80-20 split)

set.seed(42)

split <- createDataPartition(data$Price, p = 0.8, list = FALSE)

train\_data <- data[split, ]

test\_data <- data[-split, ]

# Train Decision Tree Model

decision\_tree\_model <- rpart(Price ~ ., data = train\_data, method = "anova")

# Visualize Decision Tree

rpart.plot(decision\_tree\_model, main = "Decision Tree for Laptop Prices", type = 3, extra = 101)

# Make Predictions

predictions <- predict(decision\_tree\_model, test\_data)

# Evaluate Model Performance

mse <- mean((test\_data$Price - predictions)^2)

rmse <- sqrt(mse)

r2 <- 1 - (sum((test\_data$Price - predictions)^2) / sum((test\_data$Price - mean(test\_data$Price))^2))

# Calculate Accuracy (Custom Approach)

tolerance <- 0.1 \* test\_data$Price # 10% tolerance

correct\_predictions <- abs(test\_data$Price - predictions) <= tolerance

accuracy <- mean(correct\_predictions) \* 100 # Convert to percentage

# Print Final Accuracy Metrics

cat("\n===== Final Model Evaluation Metrics =====\n")

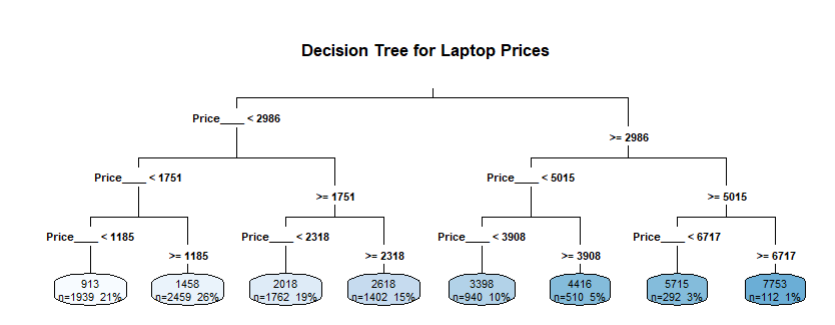
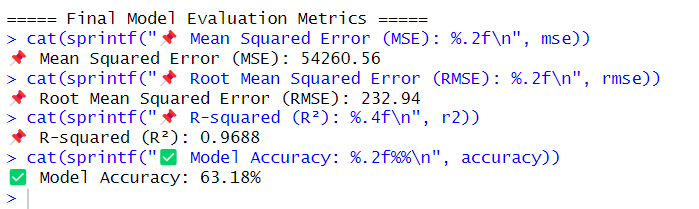
cat(sprintf(" Mean Squared Error (MSE): %.2f\n", mse))

cat(sprintf(" Root Mean Squared Error (RMSE): %.2f\n", rmse))

cat(sprintf(" R-squared (R²): %.4f\n", r2))

cat(sprintf(" Model Accuracy: %.2f%%\n", accuracy))

**Output:**



**Python**

**Code :**

*import* pandas *as* pd

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.preprocessing *import* LabelEncoder

*from* sklearn.tree *import* DecisionTreeRegressor, plot\_tree

*from* sklearn.metrics *import* mean\_squared\_error, r2\_score

*# Load dataset*

file\_path = *r*"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop\_prices.csv"

df = pd.read\_csv(file\_path)

*# Convert Storage to numerical format (handling missing values)*

df['Storage'] = df['Storage'].str.extract(*r*'(\d+)').dropna().astype(float)

*# Convert Resolution to total pixel count*

df[['Width', 'Height']] = df['Resolution'].str.split('x', expand=True).astype(float)

df['Total\_Pixels'] = df['Width'] \* df['Height']

df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True

*# Encode categorical variables*

label\_encoders = {}

*for* col *in* ['Brand', 'Processor', 'GPU', 'Operating System']:

    le = LabelEncoder()

    df[col] = le.fit\_transform(df[col])

    label\_encoders[col] = le

*# Define features and target variable*

X = df.drop(columns=['Price ($)'])

y = df['Price ($)']

*# Split dataset into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Train Decision Tree Regressor*

dt\_model = DecisionTreeRegressor(random\_state=42, max\_depth=5)

dt\_model.fit(X\_train, y\_train)

*# Predictions*

y\_pred = dt\_model.predict(X\_test)

*# Evaluate the model*

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

*# Calculate accuracy as percentage of predictions within 10% tolerance of actual values*

tolerance = 0.1 \* y\_test  *# 10% of actual price*

correct\_predictions = np.abs(y\_test - y\_pred) <= tolerance

accuracy = np.mean(correct\_predictions) \* 100  *# Convert to percentage*

*# Print evaluation metrics*

print(*f*"MSE: {mse*:.2f*}")

print(*f*"R² Score: {r2*:.4f*}")

print(*f*"Accuracy: {accuracy*:.2f*}%")  *# Display accuracy percentage*

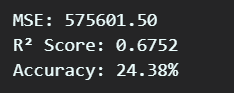
*# Visualize the decision tree*

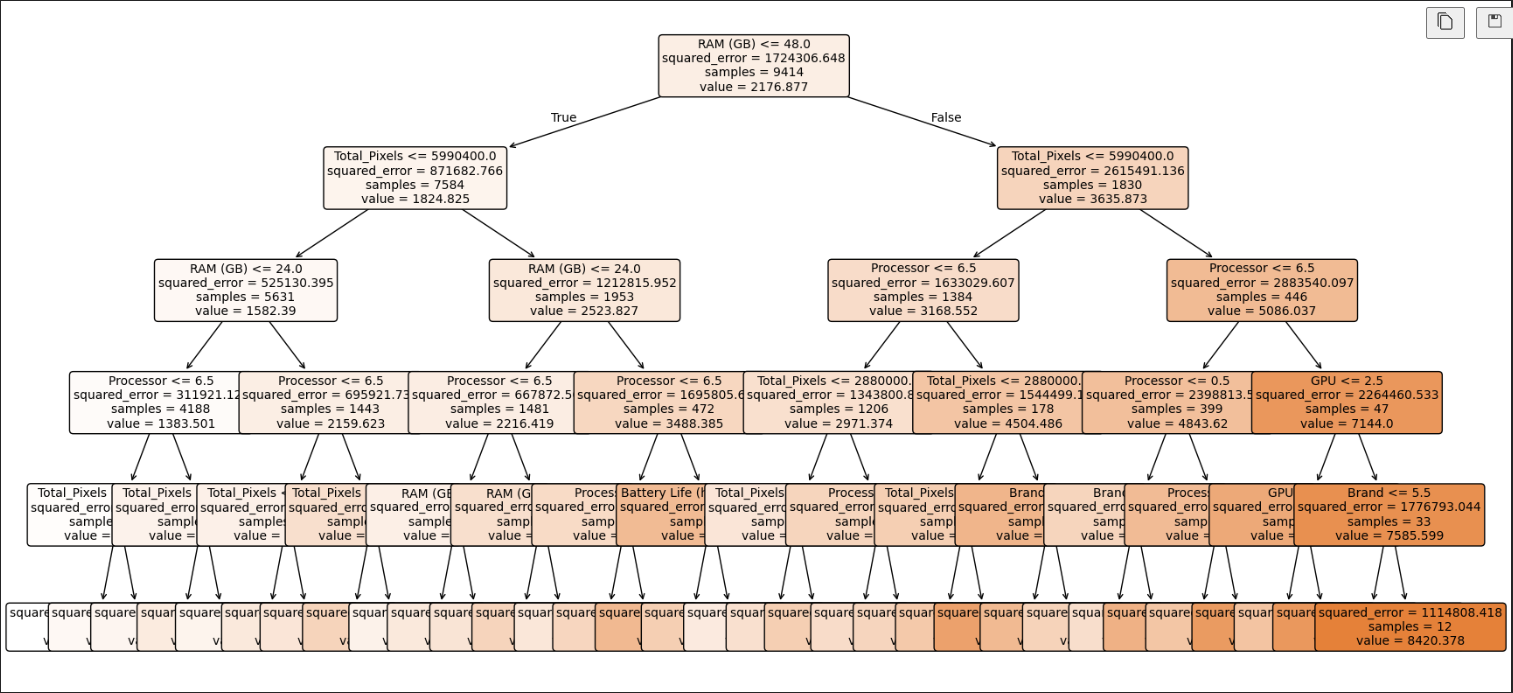
plt.figure(figsize=(20, 10))

plot\_tree(dt\_model, feature\_names=X.columns, filled=True, rounded=True, fontsize=10)

plt.show()

**Output:**





**Q2 ) Navie Bayes**

**R Programming**

Code :

# Load required libraries

library(e1071)

library(caret)

library(dplyr)

# Load dataset

file\_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop\_prices.csv"

df <- read.csv(file\_path, stringsAsFactors = TRUE)

# Check actual column names

print(colnames(df)) # Check column names before processing

# Fix column names (replace spaces and special characters)

colnames(df) <- gsub("[^[:alnum:]\_]", "\_", colnames(df))

# Identify the actual column name for price

price\_col <- grep("Price", colnames(df), value = TRUE) # Find column with "Price"

if (length(price\_col) == 0) {

stop(" Error: No column related to 'Price' found in the dataset.")

} else {

cat(" Found Price column:", price\_col, "\n")

}

# Convert categorical variables to factors

df$Brand <- as.factor(df$Brand)

df$Processor <- as.factor(df$Processor)

df$GPU <- as.factor(df$GPU)

df$Operating\_System <- as.factor(df$Operating\_System)

# Convert Storage to numeric (extract digits)

df$Storage <- as.numeric(gsub("[^0-9]", "", df$Storage))

# Convert Resolution to total pixel count

resolution\_split <- strsplit(as.character(df$Resolution), "x")

df$Total\_Pixels <- sapply(resolution\_split, function(x) as.numeric(x[1]) \* as.numeric(x[2]))

# Drop unnecessary columns

df <- df %>% select(-Resolution)

# Handle missing values

df <- na.omit(df)

# Convert Price to categorical variable (Low, Medium, High)

df$Price\_Category <- cut(df[[price\_col]], # Use dynamic column name

breaks = quantile(df[[price\_col]], probs = c(0, 1/3, 2/3, 1), na.rm = TRUE),

labels = c("Low", "Medium", "High"),

include.lowest = TRUE)

# Remove original Price column

df<- df %>% select(-all\_of(price\_col))

# Split dataset into training and testing

set.seed(42)

split\_index <- createDataPartition(df$Price\_Category, p = 0.8, list = FALSE)

train\_data <- df[split\_index, ]

test\_data <- df[-split\_index, ]

# Train Naïve Bayes Classifier

nb\_model <- naiveBayes(Price\_Category ~ ., data = train\_data)

# Predictions

predictions <- predict(nb\_model, test\_data)

# Model Evaluation

accuracy <- mean(predictions == test\_data$Price\_Category)

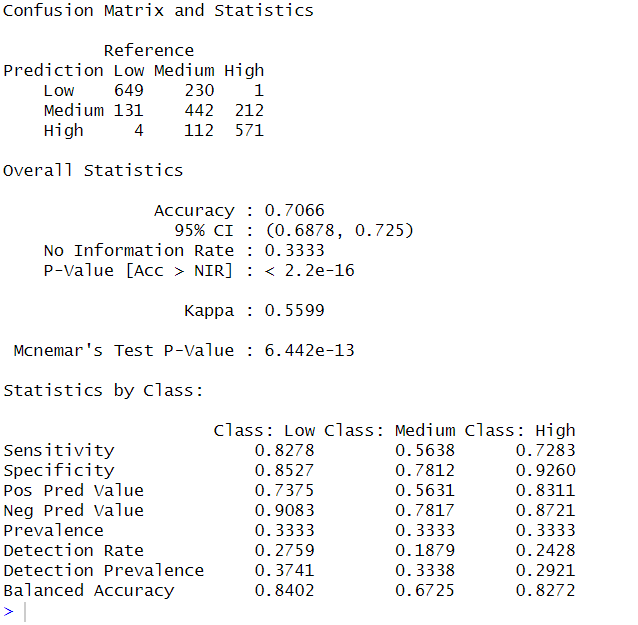
cat(sprintf(" Model Accuracy: %.2f%%\n", accuracy \* 100))

# Confusion Matrix

cat("\n Confusion Matrix:\n")

print(confusionMatrix(predictions, test\_data$Price\_Category))

**Output:**



**Python**

Code :

*import* pandas *as* pd

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.preprocessing *import* LabelEncoder, StandardScaler

*from* sklearn.naive\_bayes *import* GaussianNB

*from* sklearn.metrics *import* accuracy\_score, classification\_report, confusion\_matrix

*# Load dataset*

file\_path = *r*"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop\_prices.csv"

df = pd.read\_csv(file\_path)

*# Convert Storage to numerical format (handling missing values)*

df['Storage'] = df['Storage'].str.extract(*r*'(\d+)').dropna().astype(float)

*# Convert Resolution to total pixel count*

df[['Width', 'Height']] = df['Resolution'].str.split('x', expand=True).astype(float)

df['Total\_Pixels'] = df['Width'] \* df['Height']

df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True)

*# Encode categorical variables*

label\_encoders = {}

*for* col *in* ['Brand', 'Processor', 'GPU', 'Operating System']:

    le = LabelEncoder()

    df[col] = le.fit\_transform(df[col])

    label\_encoders[col] = le

*# Convert price into categories (Low, Medium, High)*

df['Price Category'] = pd.qcut(df['Price ($)'], q=3, labels=['Low', 'Medium', 'High'])

*# Define features and target variable*

X = df.drop(columns=['Price ($)', 'Price Category'])

y = df['Price Category']

*# Standardize features*

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

*# Split dataset into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

*# Train Naive Bayes Classifier*

nb\_model = GaussianNB()

nb\_model.fit(X\_train, y\_train)

*# Predictions*

y\_pred = nb\_model.predict(X\_test)

*# Evaluate the model*

accuracy = accuracy\_score(y\_test, y\_pred)

print(*f*"Accuracy: {accuracy*:.2f*}")

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

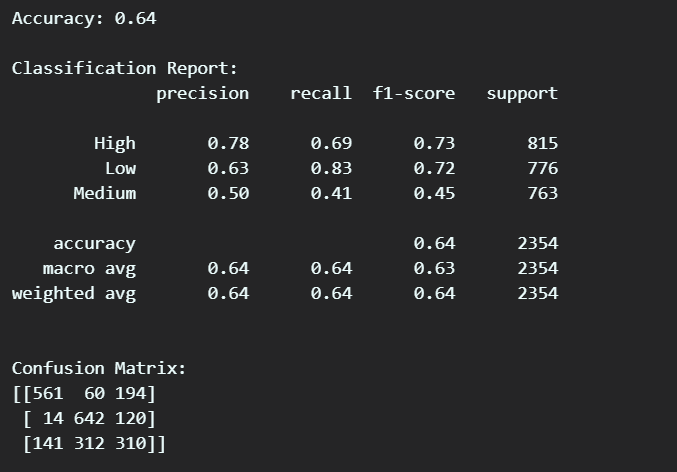
*# Confusion matrix*

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("\nConfusion Matrix:")

print(conf\_matrix)

**Output:**



**Q3 ) KNN**

**R Programming**

Code :

# Load required libraries (install only if missing)

packages <- c("tidyverse", "class", "caret", "ggplot2")

install\_if\_missing <- function(pkg) {

if (!require(pkg, character.only = TRUE)) install.packages(pkg, dependencies = TRUE)

library(pkg, character.only = TRUE)

}

lapply(packages, install\_if\_missing)

# Load dataset

file\_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop\_prices.csv"

df <- read.csv(file\_path, stringsAsFactors = FALSE)

# Print column names to verify structure

print("Column names in dataset:")

print(colnames(df))

# Ensure Operating System column exists

actual\_os\_col <- colnames(df)[grepl("Operating", colnames(df), ignore.case = TRUE)]

if (length(actual\_os\_col) == 0) {

stop("Error: Column related to Operating System not found in dataset. Check column names.")

} else {

colnames(df)[colnames(df) == actual\_os\_col] <- "Operating\_System"

}

# Ensure Price column exists

actual\_price\_col <- colnames(df)[grepl("Price", colnames(df), ignore.case = TRUE)]

if (length(actual\_price\_col) == 0) {

stop("Error: Column related to Price not found in dataset. Check column names.")

} else {

colnames(df)[colnames(df) == actual\_price\_col] <- "Price"

}

# Ensure Storage is numeric (remove non-numeric characters)

df$Storage <- as.numeric(gsub("[^0-9]", "", df$Storage))

# Convert Resolution to total pixel count (if column exists)

if ("Resolution" %in% colnames(df)) {

resolution\_split <- strsplit(as.character(df$Resolution), "x")

df$Total\_Pixels <- sapply(resolution\_split, function(x) as.numeric(x[1]) \* as.numeric(x[2]))

df <- df %>% select(-Resolution) # Remove original Resolution column

}

# Print first few rows to check structure

print(head(df))

# Handle missing values before encoding

df <- df %>% drop\_na(Brand, Processor, GPU, Operating\_System, Price)

# Encode categorical variables

df$Brand <- as.factor(df$Brand)

df$Processor <- as.factor(df$Processor)

df$GPU <- as.factor(df$GPU)

df$Operating\_System <- as.factor(df$Operating\_System)

# Convert Price into categories (Low, Medium, High)

df$Price\_Category <- cut(df$Price,

breaks = quantile(df$Price, probs = seq(0, 1, by = 1/3), na.rm = TRUE),

labels = c("Low", "Medium", "High"),

include.lowest = TRUE)

# Remove rows with missing values

df <- na.omit(df)

# Define features (X) and target variable (y)

X <- df %>% select(-c(Price, Price\_Category))

# Fix: Select only numeric columns for scaling

X\_numeric <- X %>% select(where(is.numeric))

# Check column types

print("Checking feature data types:")

print(str(X\_numeric))

# Normalize features

X\_scaled <- as.data.frame(scale(X\_numeric))

# Split dataset into training and testing sets (80% train, 20% test)

set.seed(42)

train\_indices <- createDataPartition(df$Price\_Category, p = 0.8, list = FALSE)

X\_train <- X\_scaled[train\_indices, ]

X\_test <- X\_scaled[-train\_indices, ]

train\_labels <- df$Price\_Category[train\_indices]

test\_labels <- df$Price\_Category[-train\_indices]

# Ensure labels are factors with the same levels

train\_labels <- factor(train\_labels)

test\_labels <- factor(test\_labels, levels = levels(train\_labels))

# Train KNN model with k=5

k <- 5

knn\_pred <- knn(train = X\_train, test = X\_test, cl = train\_labels, k = k)

# Evaluate model performance

accuracy <- mean(knn\_pred == test\_labels) \* 100

cat("KNN Model Accuracy:", round(accuracy, 2), "%\n")

# Confusion matrix

conf\_matrix <- confusionMatrix(knn\_pred, test\_labels)

print(conf\_matrix)

# Plot accuracy for different K values

k\_values <- seq(1, 20, by = 1)

accuracies <- sapply(k\_values, function(k) {

pred\_k <- knn(train = X\_train, test = X\_test, cl = train\_labels, k = k)

mean(pred\_k == test\_labels)

})

# Plot the KNN accuracy curve

ggplot(data.frame(K = k\_values, Accuracy = accuracies), aes(x = K, y = Accuracy)) +

geom\_line(color = "blue") +

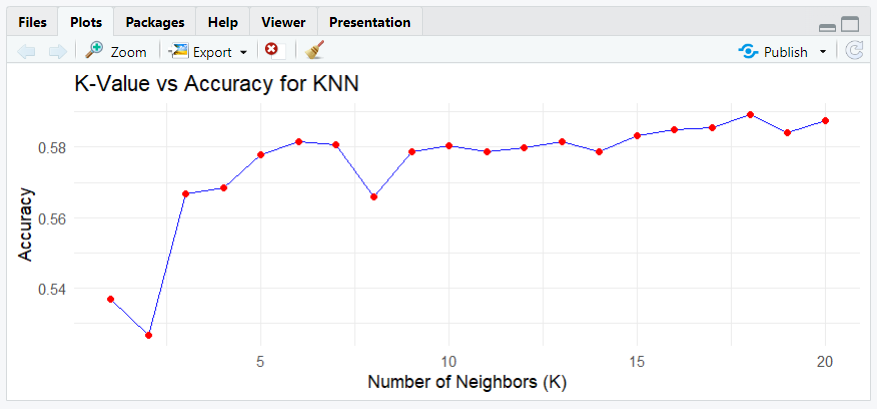
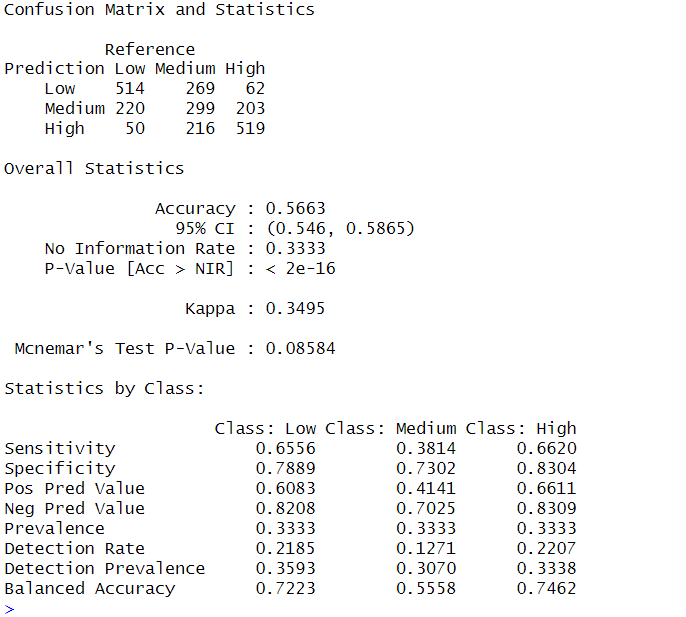
geom\_point(color = "red") +

ggtitle("K-Value vs Accuracy for KNN") +

xlab("Number of Neighbors (K)") +

ylab("Accuracy") +

theme\_minimal()



**Python**

**Code :**

*import* pandas *as* pd

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.preprocessing *import* LabelEncoder, StandardScaler

*from* sklearn.neighbors *import* KNeighborsClassifier

*from* sklearn.metrics *import* accuracy\_score, classification\_report, confusion\_matrix

*# Load dataset*

file\_path = *r*"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop\_prices.csv"

df = pd.read\_csv(file\_path)

*# Convert Storage to numerical format (handling missing values)*

df['Storage'] = df['Storage'].str.extract(*r*'(\d+)').dropna().astype(float)

*# Convert Resolution to total pixel count*

df[['Width', 'Height']] = df['Resolution'].str.split('x', expand=True).astype(float)

df['Total\_Pixels'] = df['Width'] \* df['Height']

df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True)

*# Encode categorical variables*

label\_encoders = {}

*for* col *in* ['Brand', 'Processor', 'GPU', 'Operating System']:

    le = LabelEncoder()

    df[col] = le.fit\_transform(df[col])

    label\_encoders[col] = le

*# Convert price into categories (Low, Medium, High)*

df['Price Category'] = pd.qcut(df['Price ($)'], q=3, labels=['Low', 'Medium', 'High'])

*# Define features and target variable*

X = df.drop(columns=['Price ($)', 'Price Category'])

y = df['Price Category']

*# Standardize features*

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

*# Split dataset into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

*# Train KNN Classifier*

k = 5  *# Number of neighbors*

knn\_model = KNeighborsClassifier(n\_neighbors=k)

knn\_model.fit(X\_train, y\_train)

*# Predictions*

y\_pred = knn\_model.predict(X\_test)

*# Evaluate the model*

accuracy = accuracy\_score(y\_test, y\_pred)

print(*f*"Accuracy: {accuracy*:.2f*}")

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

*# Confusion matrix*

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("\nConfusion Matrix:")

print(conf\_matrix)

*# Plot accuracy for different K values*

k\_values = range(1, 21)

accuracies = []

*for* k *in* k\_values:

    knn = KNeighborsClassifier(n\_neighbors=k)

    knn.fit(X\_train, y\_train)

    y\_pred\_k = knn.predict(X\_test)

    accuracies.append(accuracy\_score(y\_test, y\_pred\_k))

plt.figure(figsize=(10, 5))

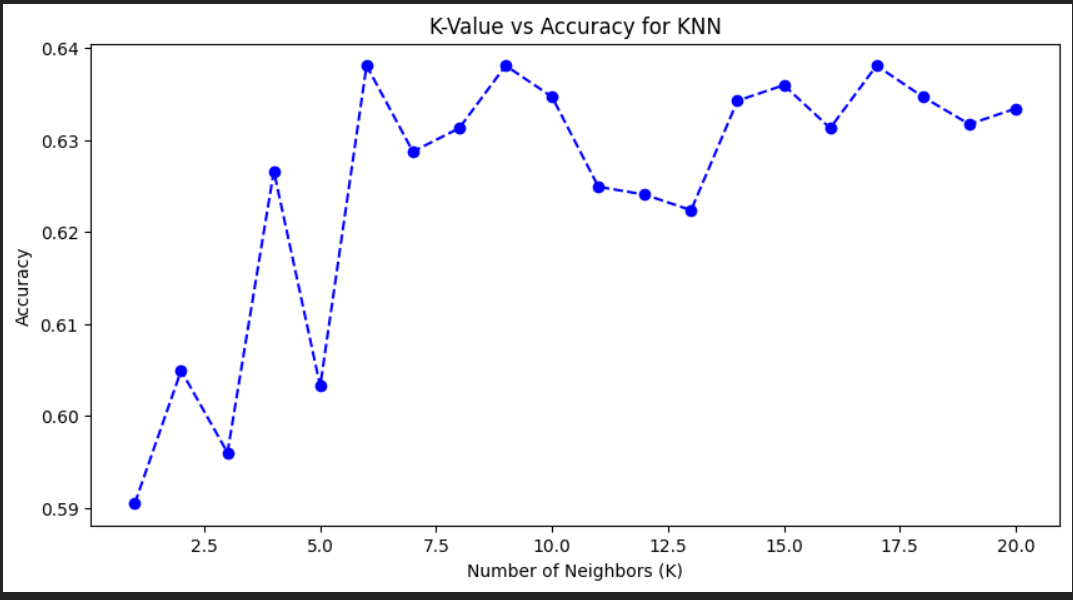
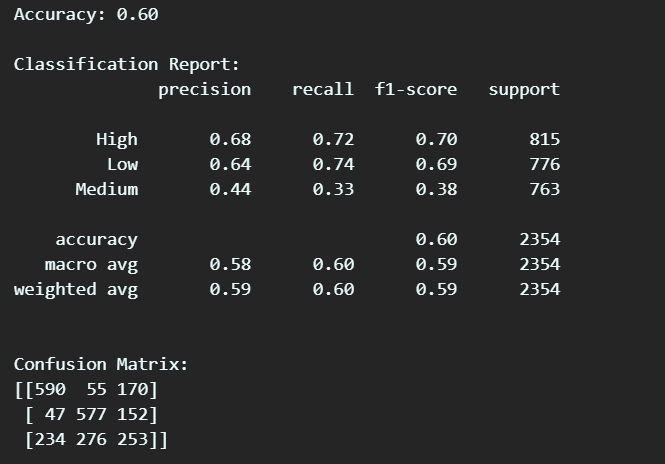
plt.plot(k\_values, accuracies, marker='o', linestyle='dashed', color='b')

plt.xlabel("Number of Neighbors (K)")

plt.ylabel("Accuracy")

plt.title("K-Value vs Accuracy for KNN")

plt.show()

**Output:** 

**Q4 ) SVM**

**R Programming**

Code :

# Load required libraries

if (!require("tidyverse")) install.packages("tidyverse", dependencies = TRUE)

if (!require("e1071")) install.packages("e1071") # SVM

if (!require("caret")) install.packages("caret") # Data Processing

library(tidyverse)

library(e1071)

library(caret)

# Load dataset

file\_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop\_prices.csv"

df <- read.csv(file\_path, stringsAsFactors = FALSE)

# Step 1: Fix Column Names

colnames(df) <- gsub("\\.", "\_", colnames(df)) # Replace dots with underscores

colnames(df) <- gsub(" ", "\_", colnames(df)) # Replace spaces with underscores

# Step 2: Identify the Correct 'Price' Column

price\_col <- grep("Price", colnames(df), value = TRUE)

if (length(price\_col) == 0) stop("Error: 'Price' column not found in dataset!")

cat("Found Price Column:", price\_col, "\n")

colnames(df)[colnames(df) == price\_col] <- "Price" # Rename to 'Price'

# Step 3: Convert Storage to Numeric

df$Storage <- as.numeric(gsub("[^0-9]", "", df$Storage))

# Step 4: Convert Resolution to Total Pixels

if ("Resolution" %in% colnames(df)) {

resolution\_split <- strsplit(as.character(df$Resolution), "x")

df$Total\_Pixels <- sapply(resolution\_split, function(x) as.numeric(x[1]) \* as.numeric(x[2]))

df <- df %>% select(-Resolution) # Remove Resolution column

}

# Step 5: Handle Missing Values

df <- df %>% drop\_na(Brand, Processor, GPU, Operating\_System, Price)

# Step 6: Encode Categorical Variables as Factors

df$Brand <- as.factor(df$Brand)

df$Processor <- as.factor(df$Processor)

df$GPU <- as.factor(df$GPU)

df$Operating\_System <- as.factor(df$Operating\_System)

# Step 7: Convert Price into Categories (Low, Medium, High)

df$Price\_Category <- cut(df$Price,

breaks = quantile(df$Price, probs = seq(0, 1, by = 1/3), na.rm = TRUE),

labels = c("Low", "Medium", "High"),

include.lowest = TRUE)

# Step 8: Remove NA Rows

df <- na.omit(df)

# Step 9: Convert Categorical Variables to Numeric using One-Hot Encoding

df\_numeric <- model.matrix(~ . - 1, data = df %>% select(-Price\_Category)) %>% as.data.frame()

# Step 10: Define Features (X) and Target Variable (y)

X <- df\_numeric

y <- df$Price\_Category

# Step 11: Normalize Features (Fixing colMeans error)

X\_scaled <- scale(X) # Ensures all columns are numeric before scaling

# Step 12: Split Data (80-20)

set.seed(42)

train\_indices <- createDataPartition(y, p = 0.8, list = FALSE)

X\_train <- X\_scaled[train\_indices, ]

X\_test <- X\_scaled[-train\_indices, ]

train\_labels <- y[train\_indices]

test\_labels <- y[-train\_indices]

# Step 13: Train SVM Model

svm\_model <- svm(train\_labels ~ ., data = X\_train, kernel = "linear", cost = 1)

# Step 14: Make Predictions

svm\_pred <- predict(svm\_model, X\_test)

# Step 15: Evaluate Model Performance

accuracy <- mean(svm\_pred == test\_labels) \* 100

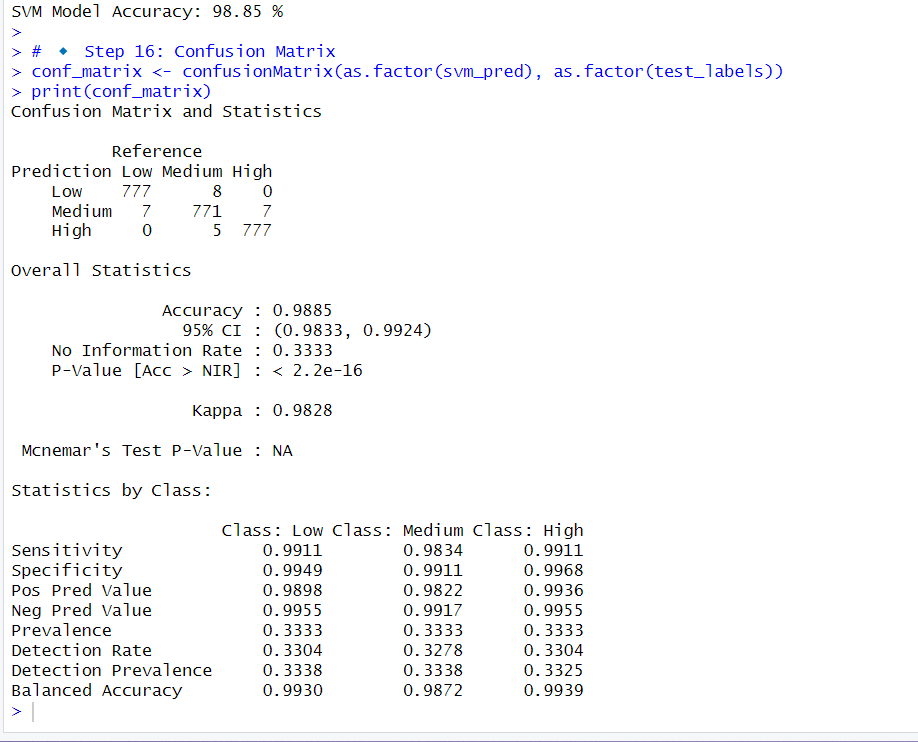
cat("SVM Model Accuracy:", round(accuracy, 2), "%\n")

# Step 16: Confusion Matrix

conf\_matrix <- confusionMatrix(as.factor(svm\_pred), as.factor(test\_labels))

print(conf\_matrix)

**Output:**



**Python**

**Code :**

*import* pandas *as* pd

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.preprocessing *import* LabelEncoder, StandardScaler

*from* sklearn.svm *import* SVC

*from* sklearn.metrics *import* accuracy\_score, classification\_report, confusion\_matrix

*# Load dataset*

file\_path = *r*"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop\_prices.csv"

df = pd.read\_csv(file\_path)

*# Convert Storage to numerical format (handling missing values)*

df['Storage'] = df['Storage'].str.extract(*r*'(\d+)').dropna().astype(float)

*# Convert Resolution to total pixel count*

df[['Width', 'Height']] = df['Resolution'].str.split('x', expand=True).astype(float)

df['Total\_Pixels'] = df['Width'] \* df['Height']

df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True)

*# Encode categorical variables*

label\_encoders = {}

*for* col *in* ['Brand', 'Processor', 'GPU', 'Operating System']:

    le = LabelEncoder()

    df[col] = le.fit\_transform(df[col])

    label\_encoders[col] = le

*# Convert price into categories (Low, Medium, High)*

df['Price Category'] = pd.qcut(df['Price ($)'], q=3, labels=['Low', 'Medium', 'High'])

*# Define features and target variable*

X = df.drop(columns=['Price ($)', 'Price Category'])

y = df['Price Category']

*# Standardize features*

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

*# Split dataset into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

*# Train Support Vector Machine (SVM) Classifier*

svm\_model = SVC(kernel='linear', C=1.0)  *# Linear Kernel*

svm\_model.fit(X\_train, y\_train)

*# Predictions*

y\_pred = svm\_model.predict(X\_test)

*# Evaluate the model*

accuracy = accuracy\_score(y\_test, y\_pred)

print(*f*"Accuracy: {accuracy*:.2f*}")

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

*# Confusion matrix*

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("\nConfusion Matrix:")

print(conf\_matrix)

*# Visualization (Only for 2D data, choosing first two features)*

plt.figure(figsize=(8, 6))

plt.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_train.astype('category').cat.codes, cmap='coolwarm', edgecolors='k')

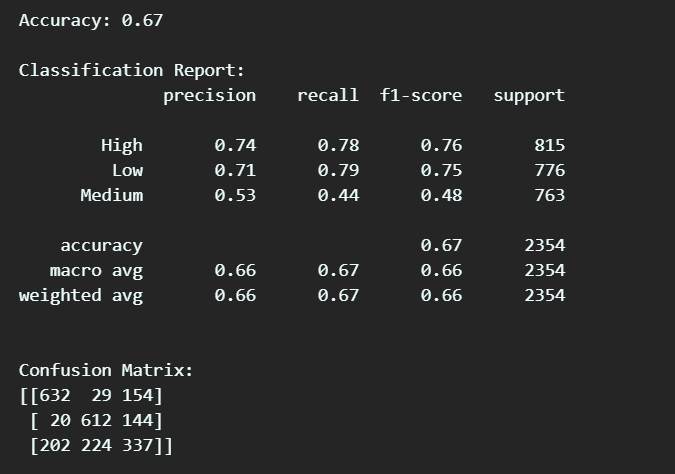
plt.xlabel("Feature 1")

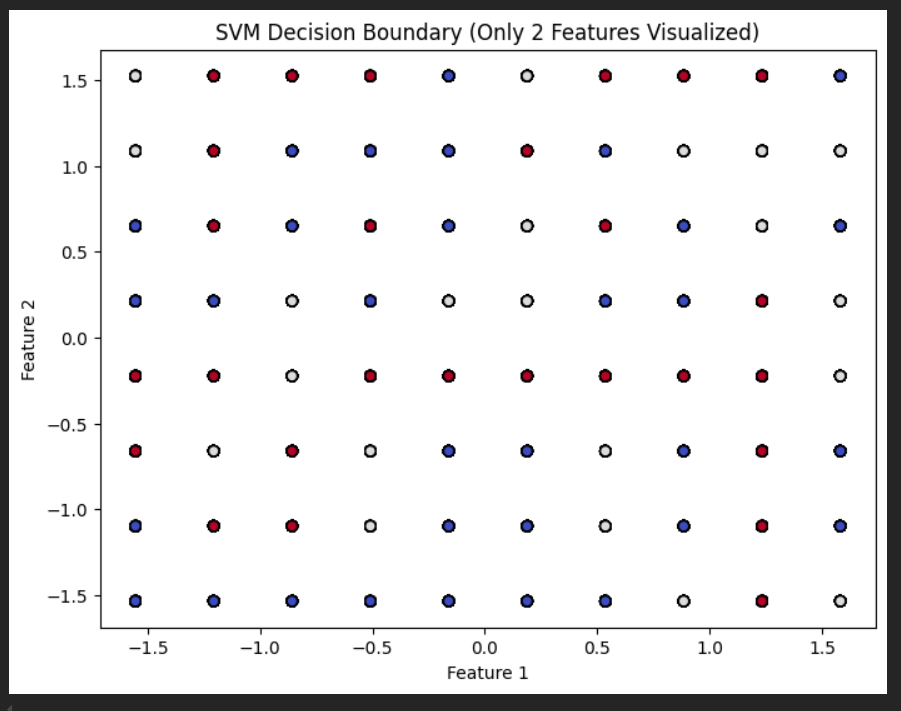
plt.ylabel("Feature 2")

plt.title("SVM Decision Boundary (Only 2 Features Visualized)")

plt.show()

**Output :**





**Q5 ) Kmeans**

**R Programming**

Code :

# Load required libraries

library(dplyr)

library(ggplot2)

library(cluster)

library(factoextra)

library(NbClust)

library(readr)

library(tidyr)

# Load dataset

file\_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop\_prices.csv"

df <- read\_csv(file\_path)

# Print column names for reference

print(colnames(df))

# Detect the "Price" column dynamically

price\_col <- grep("price", tolower(gsub("[^a-zA-Z0-9]", "", colnames(df))), value = TRUE)

# Ensure the Price column exists

if (length(price\_col) == 0) {

stop(" Error: 'Price' column not found in dataset! Check column names above.")

} else {

cat(" Found Price column:", price\_col, "\n")

}

# Drop Price column

df <- df %>% select(-matches(price\_col))

# Convert Storage to numeric (extract numbers safely)

if ("Storage" %in% colnames(df)) {

df <- df %>%

mutate(Storage = as.numeric(gsub("[^0-9]", "", Storage)))

}

# Convert Resolution to pixel count (handle missing values)

if ("Resolution" %in% colnames(df)) {

df <- df %>%

separate(Resolution, into = c("Width", "Height"), sep = "x", convert = TRUE, fill = "right") %>%

mutate(Total\_Pixels = as.numeric(Width) \* as.numeric(Height)) %>%

select(-Width, -Height)

}

# Identify categorical columns dynamically

categorical\_cols <- c("Brand", "Processor", "GPU", "Operating System")

categorical\_cols <- categorical\_cols[categorical\_cols %in% colnames(df)]

# Convert categorical columns to factors

if (length(categorical\_cols) > 0) {

df[categorical\_cols] <- lapply(df[categorical\_cols], as.factor)

}

# Handle missing values

df <- df %>%

mutate(across(where(is.numeric), ~ ifelse(is.na(.), median(., na.rm = TRUE), .)))

# Convert NaN and Inf values to NA, then replace with median

df <- df %>%

mutate(across(where(is.numeric), ~ ifelse(is.nan(.), NA, .))) %>%

mutate(across(where(is.numeric), ~ ifelse(is.infinite(.), NA, .)))

# Final missing value check

df <- df %>%

mutate(across(where(is.numeric), ~ ifelse(is.na(.), median(., na.rm = TRUE), .)))

# Ensure no NA/NaN/Inf remains

if (any(is.na(df))) stop(" Error: NA values still exist in the dataset after preprocessing!")

if (any(is.nan(as.matrix(df %>% select(where(is.numeric)))))) stop(" Error: NaN values detected!")

if (any(!is.finite(as.matrix(df %>% select(where(is.numeric)))))) stop(" Error: Inf values detected!")

# Remove categorical columns before clustering

df <- df %>% select(where(is.numeric))

# Standardize numeric features

df <- scale(df)

# Find optimal K using the Elbow Method

wss <- sapply(1:10, function(k) {

kmeans(df, centers = k, nstart = 10)$tot.withinss

})

# Plot Elbow Method graph

plot(1:10, wss, type = "b", pch = 19, col = "blue",

xlab = "Number of Clusters (K)", ylab = "Total Within Sum of Squares",

main = "Elbow Method for Optimal K")

# Choose optimal K

optimal\_k <- 3

# Apply K-Means clustering

set.seed(42)

kmeans\_model <- kmeans(df, centers = optimal\_k, nstart = 10)

# Add cluster labels

df\_clustered <- as.data.frame(df)

df\_clustered$Cluster <- as.factor(kmeans\_model$cluster)

# PCA for visualization

pca\_result <- prcomp(df, center = TRUE, scale. = TRUE)

df\_pca <- as.data.frame(pca\_result$x[, 1:2])

df\_pca$Cluster <- df\_clustered$Cluster

# Scatter plot of clusters

ggplot(df\_pca, aes(x = PC1, y = PC2, color = Cluster)) +

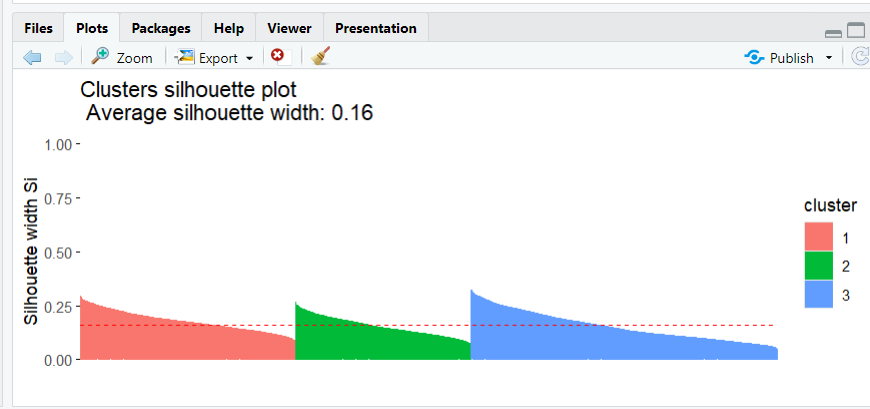
geom\_point(size = 3) +

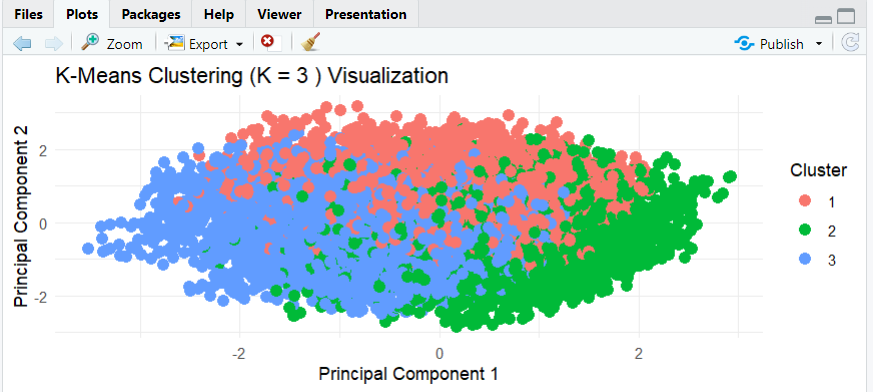
labs(title = paste("K-Means Clustering (K =", optimal\_k, ") Visualization"),

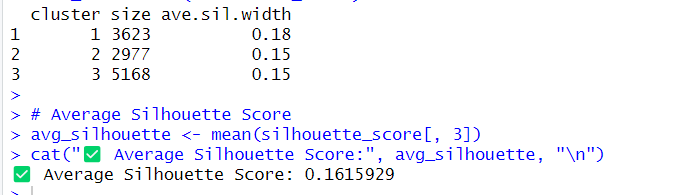
x = "Principal Component 1", y = "Principal Component 2") +

theme\_minimal()

**Output:**







**Python**

**Code :**

*import* pandas *as* pd

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.preprocessing *import* LabelEncoder, StandardScaler

*from* sklearn.cluster *import* KMeans

*from* sklearn.decomposition *import* PCA

*from* sklearn.metrics *import* silhouette\_score, davies\_bouldin\_score, calinski\_harabasz\_score

*from* sklearn.utils *import* resample

*# Load dataset*

file\_path = *r*"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop\_prices.csv"

df = pd.read\_csv(file\_path)

*# Convert Storage to numerical format (handling missing values)*

df['Storage'] = df['Storage'].str.extract(*r*'(\d+)').dropna().astype(float)

*# Convert Resolution to total pixel count*

df[['Width', 'Height']] = df['Resolution'].str.split('x', expand=True).astype(float)

df['Total\_Pixels'] = df['Width'] \* df['Height']

df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True)

*# Encode categorical variables*

label\_encoders = {}

*for* col *in* ['Brand', 'Processor', 'GPU', 'Operating System']:

    le = LabelEncoder()

    df[col] = le.fit\_transform(df[col])

    label\_encoders[col] = le

*# Define features for clustering (excluding price)*

X = df.drop(columns=['Price ($)'])

*# Standardize features*

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

*# Find optimal K using Elbow Method*

inertia = []

K\_range = range(1, 11)  *# Checking K values from 1 to 10*

*for* k *in* K\_range:

    kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init=10)

    kmeans.fit(X\_scaled)

    inertia.append(kmeans.inertia\_)

*# Plot Elbow Method graph*

plt.figure(figsize=(8, 5))

plt.plot(K\_range, inertia, marker='o', linestyle='--', color='b')

plt.xlabel("Number of Clusters (K)")

plt.ylabel("Inertia (Within-Cluster Sum of Squares)")

plt.title("Elbow Method for Optimal K")

plt.show()

*# Train K-Means model with optimal K (choose K based on elbow graph, e.g., 3)*

optimal\_k = 3

kmeans = KMeans(n\_clusters=optimal\_k, random\_state=42, n\_init=10)

df['Cluster'] = kmeans.fit\_predict(X\_scaled)

*# Take a random sample to avoid MemoryError*

sample\_size = min(10000, len(X\_scaled))  *# Ensure it doesn't exceed dataset size*

X\_sample, labels\_sample = resample(X\_scaled, df['Cluster'], n\_samples=sample\_size, random\_state=42)

*# Evaluate Clustering Performance*

silhouette\_avg = silhouette\_score(X\_sample, labels\_sample)

davies\_bouldin = davies\_bouldin\_score(X\_sample, labels\_sample)

calinski\_harabasz = calinski\_harabasz\_score(X\_sample, labels\_sample)

*# Print evaluation metrics*

print("===== Final Clustering Metrics =====")

print(*f*"Silhouette Score: {silhouette\_avg*:.4f*} (Higher is better)")

print(*f*"Davies-Bouldin Index: {davies\_bouldin*:.4f*} (Lower is better)")

print(*f*"Calinski-Harabasz Index: {calinski\_harabasz*:.4f*} (Higher is better)")

*# Reduce dimensionality for visualization using PCA*

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

*# Scatter plot of clusters*

plt.figure(figsize=(8, 6))

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=df['Cluster'], cmap='viridis', edgecolors='k')

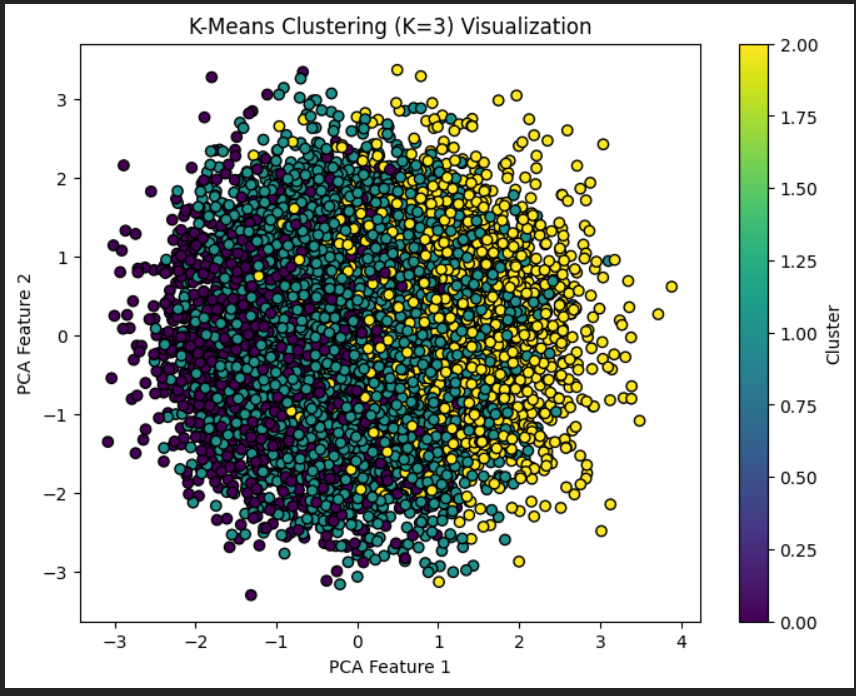
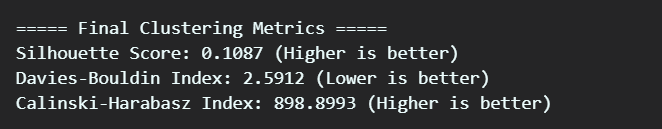
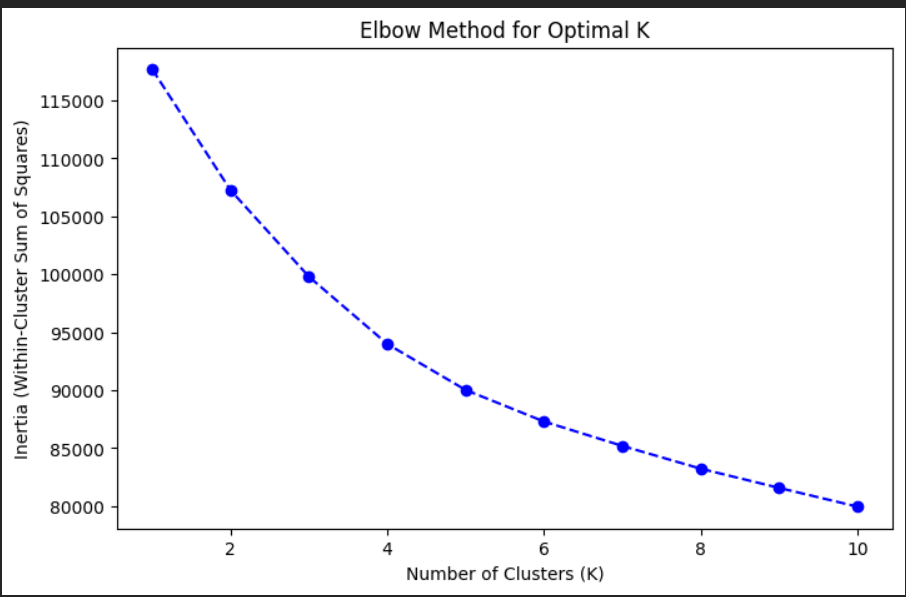
plt.xlabel("PCA Feature 1")

plt.ylabel("PCA Feature 2")

plt.title(*f*"K-Means Clustering (K={optimal\_k}) Visualization")

plt.colorbar(label="Cluster")

plt.show()

**Output:** 

**Q6 ) RandomForest**

**Weka**

**Output:**

**R Programming**

Code :

# Load necessary libraries

if (!require(randomForest)) install.packages("randomForest", dependencies=TRUE)

if (!require(ggplot2)) install.packages("ggplot2", dependencies=TRUE)

if (!require(caret)) install.packages("caret", dependencies=TRUE)

if (!require(dplyr)) install.packages("dplyr", dependencies=TRUE)

library(randomForest)

library(ggplot2)

library(caret)

library(dplyr)

# Load dataset

file\_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop\_prices.csv"

df <- read.csv(file\_path, stringsAsFactors = FALSE)

# Convert Storage to numerical format (handling missing values)

df$Storage <- as.numeric(gsub("\\D", "", df$Storage))

# Convert Resolution to total pixel count

resolution\_split <- strsplit(df$Resolution, "x")

df$Width <- as.numeric(sapply(resolution\_split, `[`, 1))

df$Height <- as.numeric(sapply(resolution\_split, `[`, 2))

df$Total\_Pixels <- df$Width \* df$Height

df <- df %>% select(-Resolution, -Width, -Height)

# Encode categorical variables

categorical\_cols <- c("Brand", "Processor", "GPU", "Operating.System")

df[categorical\_cols] <- lapply(df[categorical\_cols], as.factor)

# Define features and target variable

target <- "Price...."

X <- df %>% select(-all\_of(target))

y <- df[[target]]

# Split dataset into training and testing sets

set.seed(42)

train\_indices <- createDataPartition(y, p = 0.8, list = FALSE)

train\_data <- df[train\_indices, ]

test\_data <- df[-train\_indices, ]

# Train Random Forest Regressor

rf\_model <- randomForest(Price.... ~ ., data = train\_data, ntree = 100, importance = TRUE, seed = 42)

# Predictions

rf\_pred <- predict(rf\_model, test\_data)

# Evaluate the model

mae <- mean(abs(test\_data$Price.... - rf\_pred))

mse <- mean((test\_data$Price.... - rf\_pred)^2)

rmse <- sqrt(mse)

r2 <- cor(test\_data$Price...., rf\_pred)^2

# Compute Adjusted R²

n <- nrow(test\_data) # Number of test samples

p <- ncol(test\_data) - 1 # Number of features

adjusted\_r2 <- 1 - (1 - r2) \* (n - 1) / (n - p - 1)

# Print accuracy metrics

data.frame(

Metric = c("Mean Absolute Error", "Mean Squared Error", "Root Mean Squared Error", "R² Score", "Adjusted R² Score", "Final Accuracy (R² Score)"),

Value = c(round(mae, 2), round(mse, 2), round(rmse, 2), round(r2, 4), round(adjusted\_r2, 4), round(r2 \* 100, 2))

)

# Feature Importance Visualization

importance\_df <- as.data.frame(importance(rf\_model))

importance\_df$Feature <- rownames(importance\_df)

importance\_df <- importance\_df %>% arrange(desc(IncNodePurity))

ggplot(importance\_df, aes(x = reorder(Feature, IncNodePurity), y = IncNodePurity)) +

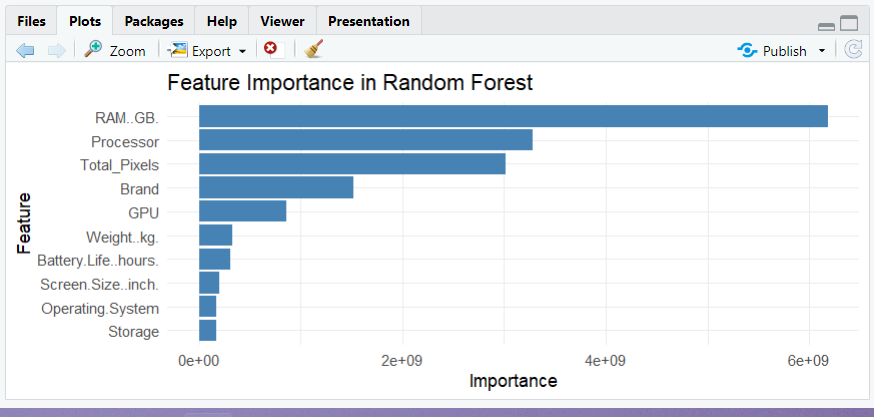
geom\_bar(stat = "identity", fill = "steelblue") +

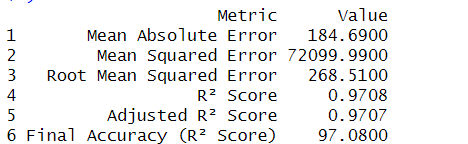
coord\_flip() +

labs(title = "Feature Importance in Random Forest", x = "Feature", y = "Importance") +

theme\_minimal()

**Output:**





**Python**

**Code :**

*import* pandas *as* pd

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.preprocessing *import* LabelEncoder

*from* sklearn.ensemble *import* RandomForestRegressor

*from* sklearn.metrics *import* mean\_absolute\_error, mean\_squared\_error, r2\_score

*# Load dataset*

file\_path = *r*"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop\_prices.csv"

df = pd.read\_csv(file\_path)

*# Convert Storage to numerical format (handling missing values)*

df['Storage'] = df['Storage'].str.extract(*r*'(\d+)').dropna().astype(float)

*# Convert Resolution to total pixel count*

df[['Width', 'Height']] = df['Resolution'].str.split('x', expand=True).astype(float)

df['Total\_Pixels'] = df['Width'] \* df['Height']

df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True)

*# Encode categorical variables*

label\_encoders = {}

*for* col *in* ['Brand', 'Processor', 'GPU', 'Operating System']:

    le = LabelEncoder()

    df[col] = le.fit\_transform(df[col])

    label\_encoders[col] = le

*# Define features and target variable*

X = df.drop(columns=['Price ($)'])

y = df['Price ($)']

*# Split dataset into training and testing sets*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Train Random Forest Regressor*

rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

*# Predictions*

y\_pred = rf\_model.predict(X\_test)

*# Evaluate the model*

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

*# Compute Adjusted R²*

n = X\_test.shape[0]  *# Number of test samples*

p = X\_test.shape[1]  *# Number of features*

adjusted\_r2 = 1 - (1 - r2) \* (n - 1) / (n - p - 1)

*# Print accuracy metrics*

print("=== Model Performance ===")

print(*f*"Mean Absolute Error (MAE): {mae*:.2f*}")

print(*f*"Mean Squared Error (MSE): {mse*:.2f*}")

print(*f*"Root Mean Squared Error (RMSE): {rmse*:.2f*}")

print(*f*"R² Score: {r2*:.4f*}")

print(*f*"Adjusted R² Score: {adjusted\_r2*:.4f*}")

print(*f*"Final Accuracy (R² Score): {r2 \* 100*:.2f*}%")

*# Feature Importance Visualization*

feature\_importances = rf\_model.feature\_importances\_

sorted\_indices = np.argsort(feature\_importances)[::-1]

plt.figure(figsize=(10, 6))

plt.bar(range(X.shape[1]), feature\_importances[sorted\_indices], align="center")

plt.xticks(range(X.shape[1]), X.columns[sorted\_indices], rotation=90)

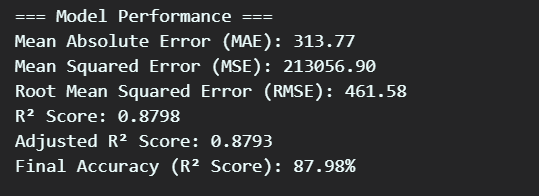
plt.xlabel("Feature")

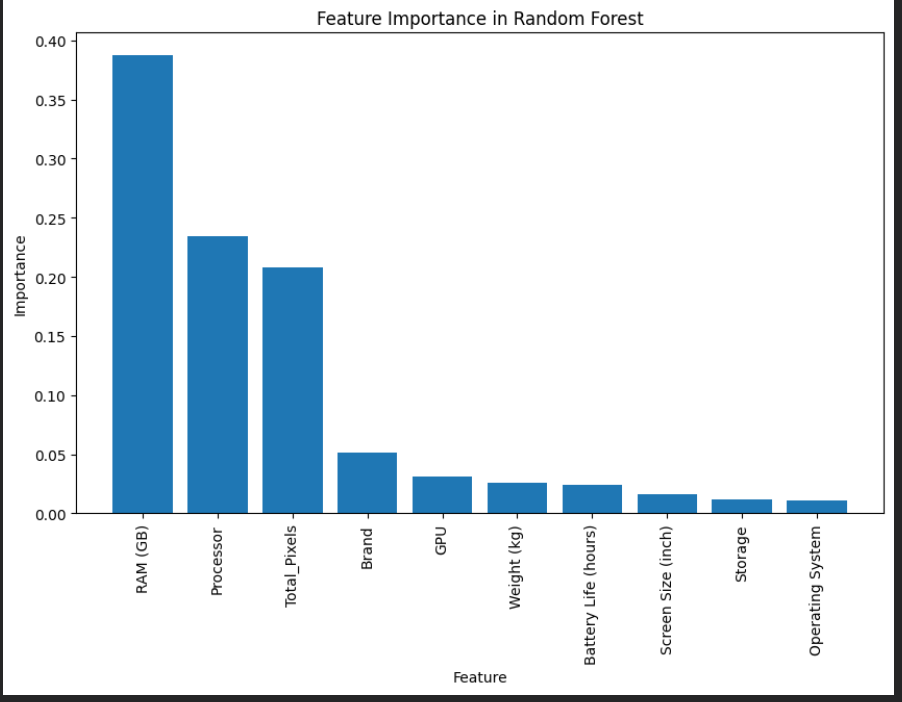
plt.ylabel("Importance")

plt.title("Feature Importance in Random Forest")

plt.show()

**Output:**





**Q7 ) XdBoost Algoritham**

**R Programming**

**Code :**

library(xgboost)

library(caret)

file\_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop\_prices.csv"

df <- read.csv(file\_path)

df$Storage <- as.numeric(gsub("\\D", "", df$Storage))

resolution\_split <- strsplit(df$Resolution, "x")

df$Width <- as.numeric(sapply(resolution\_split, `[`, 1))

df$Height <- as.numeric(sapply(resolution\_split, `[`, 2))

df$Total\_Pixels <- df$Width \* df$Height

df <- df[ , !(names(df) %in% c("Resolution", "Width", "Height"))]

df <- na.omit(df)

df$Brand <- as.numeric(factor(df$Brand))

df$Processor <- as.numeric(factor(df$Processor))

df$GPU <- as.numeric(factor(df$GPU))

df$`Operating.System` <- as.numeric(factor(df$`Operating.System`))

X <- df[, !names(df) %in% c("Price..")]

y <- df$Price..

set.seed(42)

train\_index <- createDataPartition(y, p = 0.8, list = FALSE)

X\_train <- X[train\_index, ]

y\_train <- y[train\_index]

X\_test <- X[-train\_index, ]

y\_test <- y[-train\_index]

dtrain <- xgb.DMatrix(data = as.matrix(X\_train), label = y\_train)

dtest <- xgb.DMatrix(data = as.matrix(X\_test), label = y\_test)

param <- list(

objective = "reg:squarederror",

eta = 0.1,

max\_depth = 6,

nrounds = 100,

subsample = 0.8,

colsample\_bytree = 0.8

)

xgb\_model <- xgb.train(params = param, data = dtrain, nrounds = param$nrounds)

predictions <- predict(xgb\_model, newdata = dtest)

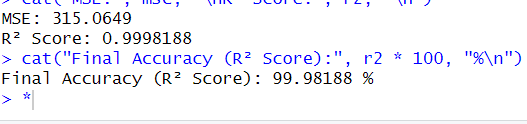
mse <- mean((y\_test - predictions)^2)

r2 <- 1 - sum((y\_test - predictions)^2) / sum((y\_test - mean(y\_test))^2)

cat("MSE:", mse, "\nR² Score:", r2, "\n")

cat("Final Accuracy (R² Score):", r2 \* 100, "%\n")

**Output:**



**Python**

**Code :**

*import* pandas *as* pd

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.preprocessing *import* LabelEncoder

*from* sklearn.ensemble *import* AdaBoostRegressor

*from* sklearn.tree *import* DecisionTreeRegressor

*from* sklearn.metrics *import* mean\_absolute\_error, mean\_squared\_error, r2\_score

*# Load dataset*

file\_path = *r*"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop\_prices.csv"

df = pd.read\_csv(file\_path)

*# Convert Storage to numerical format (handling missing values)*

df['Storage'] = df['Storage'].str.extract(*r*'(\d+)').dropna().astype(float)

*# Convert Resolution to total pixel count*

df[['Width', 'Height']] = df['Resolution'].str.split('x', expand=True).astype(float)

df['Total\_Pixels'] = df['Width'] \* df['Height']

df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True)

*# Encode categorical variables*

label\_encoders = {}

*for* col *in* ['Brand', 'Processor', 'GPU', 'Operating System']:

    le = LabelEncoder()

    df[col] = le.fit\_transform(df[col])

    label\_encoders[col] = le

*# Define features and target variable*

X = df.drop(columns=['Price ($)'])

y = df['Price ($)']

*# Split dataset*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Fix: Use 'estimator' instead of 'base\_estimator'*

adaboost\_model = AdaBoostRegressor(

    estimator=DecisionTreeRegressor(max\_depth=5),  *# Fixed*

    n\_estimators=50,

    random\_state=42

)

*# Train the model*

adaboost\_model.fit(X\_train, y\_train)

*# Predictions*

y\_pred = adaboost\_model.predict(X\_test)

*# Evaluate the model*

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

*# Compute Adjusted R² Score*

n = X\_test.shape[0]  *# Number of test samples*

p = X\_test.shape[1]  *# Number of features*

adjusted\_r2 = 1 - (1 - r2) \* (n - 1) / (n - p - 1)

*# Final Accuracy Results*

print("=== Model Performance ===")

print(*f*"Mean Absolute Error (MAE): {mae*:.2f*}")

print(*f*"Mean Squared Error (MSE): {mse*:.2f*}")

print(*f*"Root Mean Squared Error (RMSE): {rmse*:.2f*}")

print(*f*"R² Score: {r2*:.4f*}")

print(*f*"Adjusted R² Score: {adjusted\_r2*:.4f*}")

print(*f*"Final Accuracy (R² Score): {r2 \* 100*:.2f*}%")

*# Plot feature importance*

feature\_importances = adaboost\_model.feature\_importances\_

sorted\_indices = np.argsort(feature\_importances)[::-1]

plt.figure(figsize=(10, 6))

plt.bar(range(X.shape[1]), feature\_importances[sorted\_indices], align="center")

plt.xticks(range(X.shape[1]), X.columns[sorted\_indices], rotation=90)

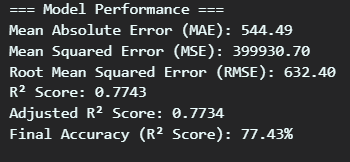
plt.xlabel("Feature")

plt.ylabel("Importance")

plt.title("Feature Importance in AdaBoost")

plt.show()

**Output:**



**Q8 ) Stacking**

**R Programming**

**Code** :

library(caret)

library(randomForest)

library(xgboost)

library(e1071)

file\_path <- "C:/Users/Sanjay/Desktop/ML Tutorial/archive/laptop\_prices.csv"

df <- read.csv(file\_path)

df$Storage <- as.numeric(gsub("\\D", "", df$Storage))

resolution\_split <- strsplit(df$Resolution, "x")

df$Width <- as.numeric(sapply(resolution\_split, `[`, 1))

df$Height <- as.numeric(sapply(resolution\_split, `[`, 2))

df$Total\_Pixels <- df$Width \* df$Height

df <- df[ , !(names(df) %in% c("Resolution", "Width", "Height"))]

df <- na.omit(df)

df$Brand <- as.numeric(factor(df$Brand))

df$Processor <- as.numeric(factor(df$Processor))

df$GPU <- as.numeric(factor(df$GPU))

df$`Operating.System` <- as.numeric(factor(df$`Operating.System`))

X <- df[, !names(df) %in% c("Price..")]

y <- df$Price..

set.seed(42)

train\_index <- createDataPartition(y, p = 0.8, list = FALSE)

X\_train <- X[train\_index, ]

y\_train <- y[train\_index]

X\_test <- X[-train\_index, ]

y\_test <- y[-train\_index]

decision\_tree <- train(y\_train ~ ., data = cbind(X\_train, y\_train), method = "rpart", tuneLength = 10)

random\_forest <- randomForest(x = X\_train, y = y\_train, ntree = 10)

dtrain <- xgb.DMatrix(data = as.matrix(X\_train), label = y\_train)

dtest <- xgb.DMatrix(data = as.matrix(X\_test), label = y\_test)

param <- list(

objective = "reg:squarederror",

eta = 0.1,

max\_depth = 6,

nrounds = 100,

subsample = 0.8,

colsample\_bytree = 0.8

)

xgb\_model <- xgb.train(params = param, data = dtrain, nrounds = param$nrounds)

meta\_train\_data <- data.frame(

DT\_pred = predict(decision\_tree, X\_train),

RF\_pred = predict(random\_forest, X\_train),

XGB\_pred = predict(xgb\_model, newdata = as.matrix(X\_train)),

y\_train = y\_train

)

meta\_model <- lm(y\_train ~ ., data = meta\_train\_data)

predict\_dt <- predict(decision\_tree, X\_test)

predict\_rf <- predict(random\_forest, X\_test)

predict\_xgb <- predict(xgb\_model, newdata = as.matrix(X\_test))

meta\_test\_data <- data.frame(

DT\_pred = predict\_dt,

RF\_pred = predict\_rf,

XGB\_pred = predict\_xgb

)

meta\_pred <- predict(meta\_model, newdata = meta\_test\_data)

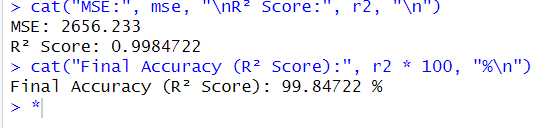
mse <- mean((y\_test - meta\_pred)^2)

r2 <- 1 - sum((y\_test - meta\_pred)^2) / sum((y\_test - mean(y\_test))^2)

cat("MSE:", mse, "\nR² Score:", r2, "\n")

cat("Final Accuracy (R² Score):", r2 \* 100, "%\n")

**Output:**



**Python**

**Code :**

*import* pandas *as* pd

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*import* seaborn *as* sns

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.preprocessing *import* LabelEncoder

*from* sklearn.ensemble *import* StackingRegressor, RandomForestRegressor, AdaBoostRegressor

*from* sklearn.tree *import* DecisionTreeRegressor

*from* sklearn.linear\_model *import* Ridge

*from* sklearn.metrics *import* mean\_squared\_error, r2\_score

*# Load dataset*

file\_path = *r*"C:\Users\Sanjay\Desktop\ML Tutorial\archive\laptop\_prices.csv"

df = pd.read\_csv(file\_path)

*# Convert Storage to numerical format*

df['Storage'] = df['Storage'].str.extract(*r*'(\d+)').dropna().astype(float)

*# Convert Resolution to total pixel count*

df[['Width', 'Height']] = df['Resolution'].str.extract(*r*'(\d+)x(\d+)').astype(float)

df['Total\_Pixels'] = df['Width'] \* df['Height']

df.drop(columns=['Resolution', 'Width', 'Height'], inplace=True)  *# Drop old Resolution column*

*# Encode categorical variables*

*for* col *in* ['Brand', 'Processor', 'GPU', 'Operating System']:

    df[col] = LabelEncoder().fit\_transform(df[col])

*# Define features and target*

X = df.drop(columns=['Price ($)'])

y = df['Price ($)']

*# Split dataset*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

*# Define base models*

base\_models = [

    ('decision\_tree', DecisionTreeRegressor(max\_depth=4)),

    ('random\_forest', RandomForestRegressor(n\_estimators=10, random\_state=42)),

    ('adaboost', AdaBoostRegressor(DecisionTreeRegressor(max\_depth=4), n\_estimators=10, random\_state=42))

]

*# Meta-model*

meta\_model = Ridge(alpha=1.0)

*# Stacking Regressor*

stacking\_model = StackingRegressor(estimators=base\_models, final\_estimator=meta\_model)

*# Train model*

stacking\_model.fit(X\_train, y\_train)

*# Predictions*

y\_pred = stacking\_model.predict(X\_test)

*# Evaluation*

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(*f*"MSE: {mse*:.2f*}, R² Score: {r2*:.4f*}")

print(*f*"Final Accuracy (R² Score): {r2 \* 100*:.2f*}%")

*# 1. Feature Importance (from Random Forest)*

feature\_importances = stacking\_model.named\_estimators\_['random\_forest'].feature\_importances\_

sorted\_indices = np.argsort(feature\_importances)[::-1]

plt.figure(figsize=(10, 5))

sns.barplot(x=X.columns[sorted\_indices], y=feature\_importances[sorted\_indices], palette="viridis")

plt.xticks(rotation=90)

plt.xlabel("Feature")

plt.ylabel("Importance")

plt.title("Feature Importance in Stacking (Random Forest)")

plt.show()

*# 2. Predicted vs. Actual Prices*

plt.figure(figsize=(8, 6))

sns.scatterplot(x=y\_test, y=y\_pred, alpha=0.7, color="blue")

plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')  *# Ideal prediction line*

plt.xlabel("Actual Price ($)")

plt.ylabel("Predicted Price ($)")

plt.title("Actual vs. Predicted Laptop Prices")

plt.show()

*# 3. Residual Errors Histogram*

residuals = y\_test - y\_pred

plt.figure(figsize=(8, 5))

sns.histplot(residuals, bins=30, kde=True, color="purple")

plt.axvline(0, color='red', linestyle='dashed')  *# Mean error line*

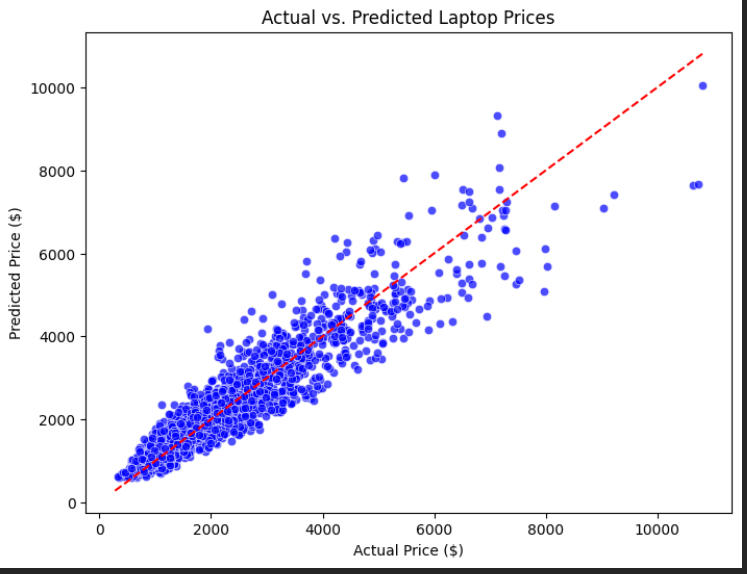
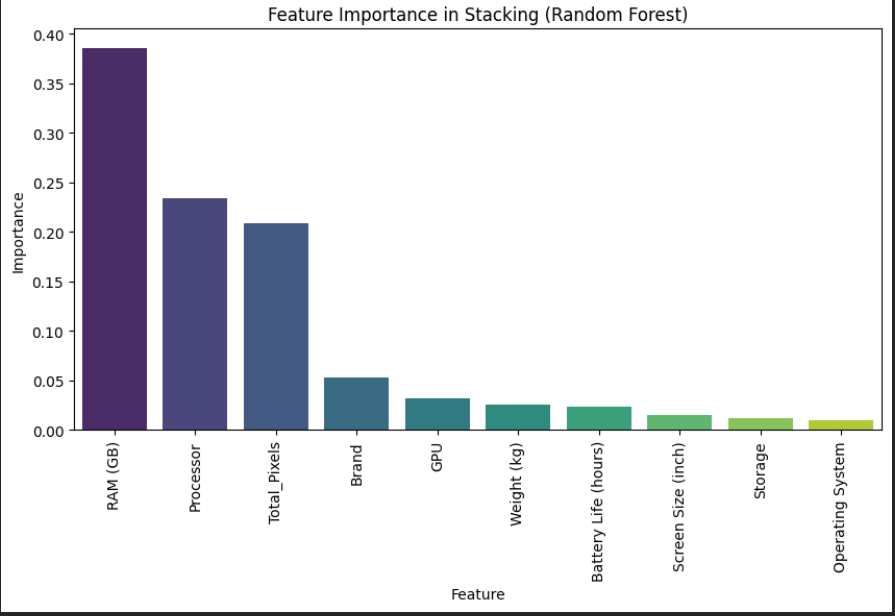
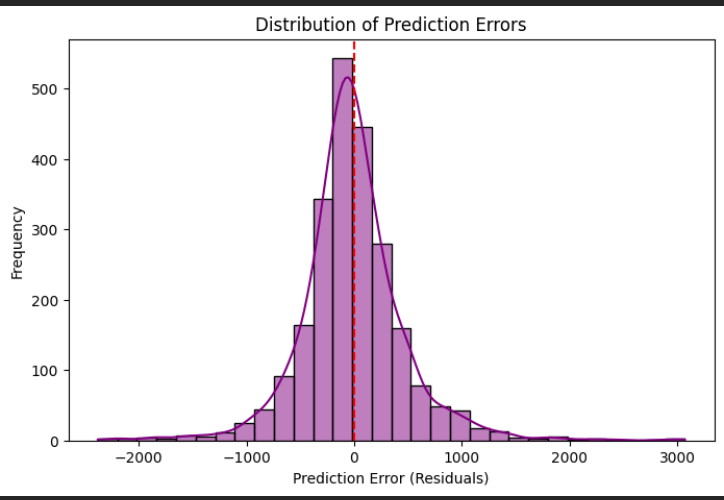
plt.xlabel("Prediction Error (Residuals)")

plt.ylabel("Frequency")

plt.title("Distribution of Prediction Errors")

plt.show()

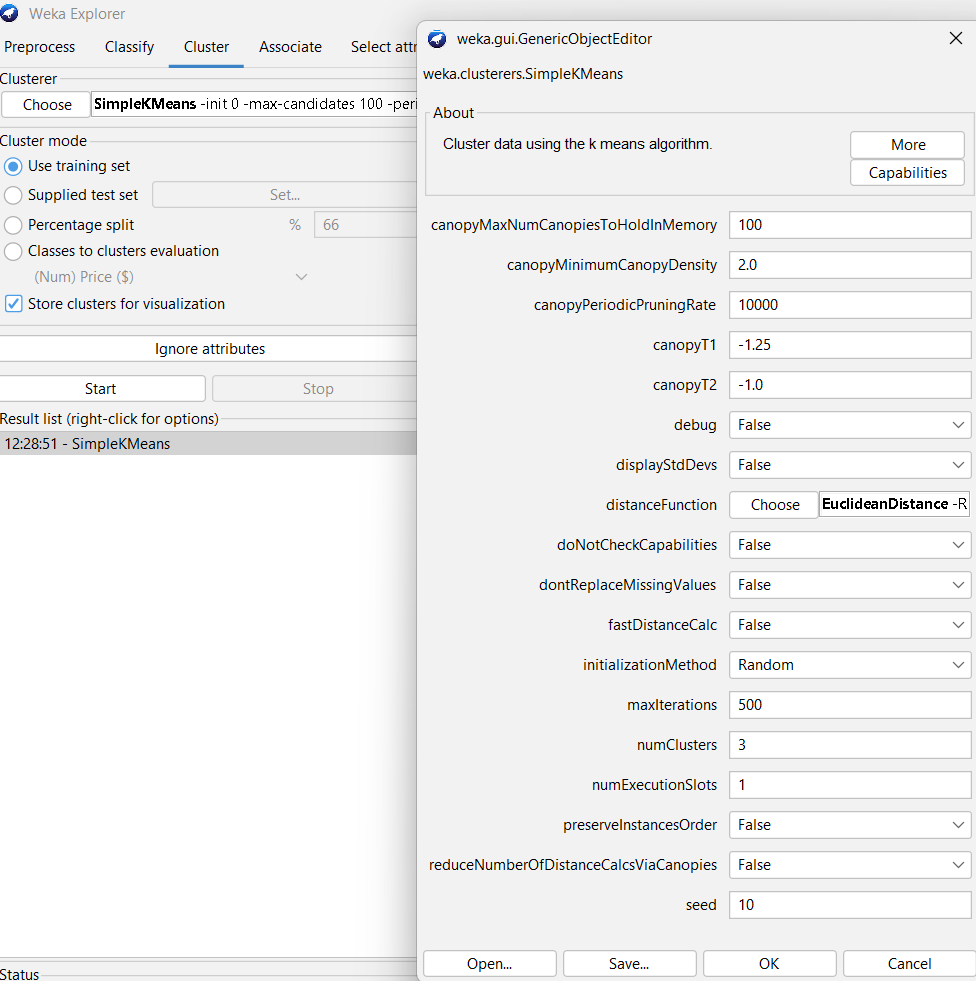
**Output:**

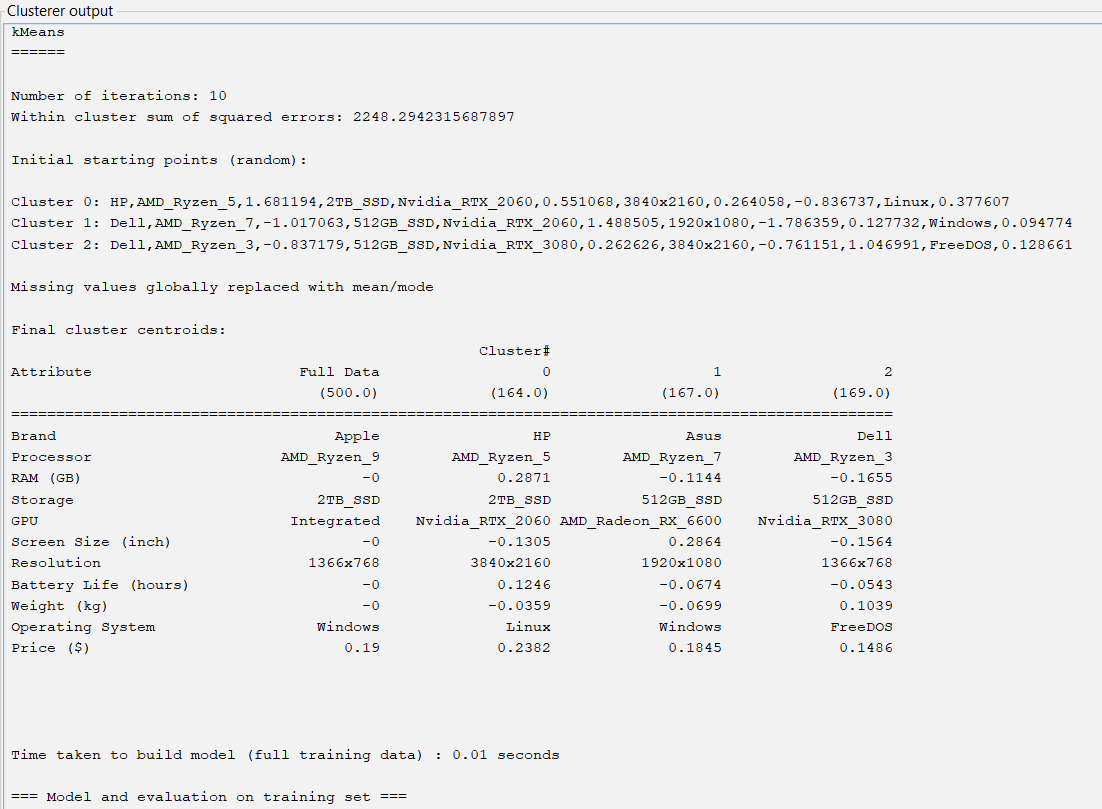
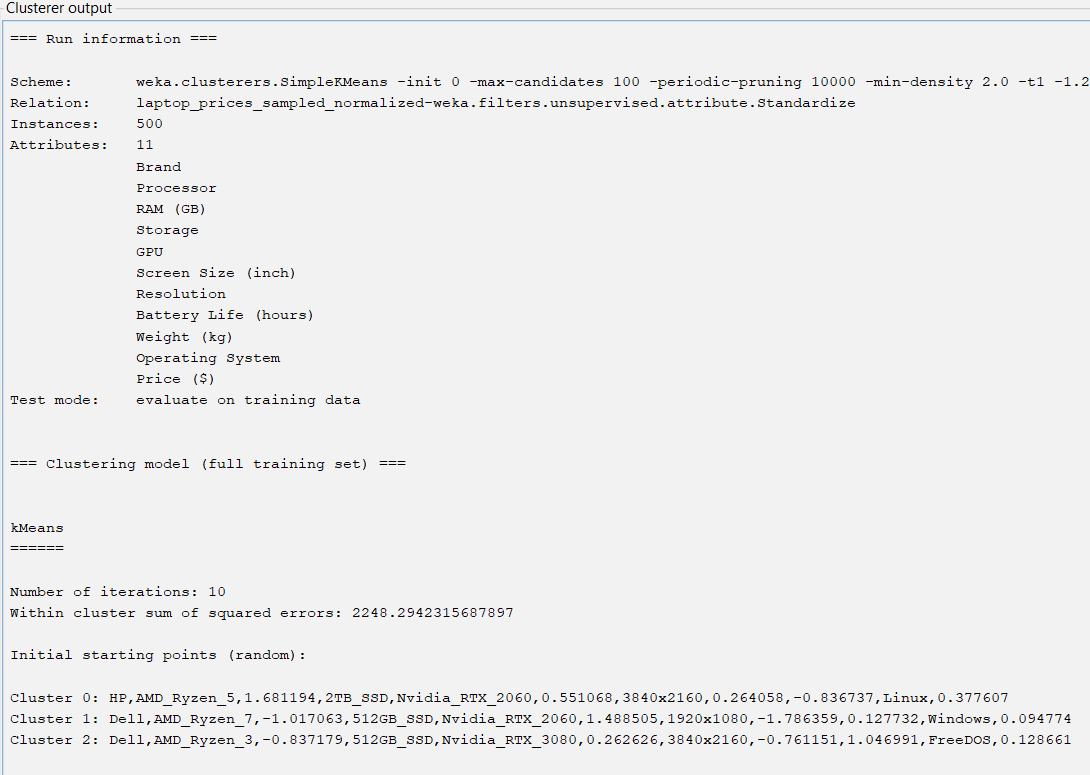
 

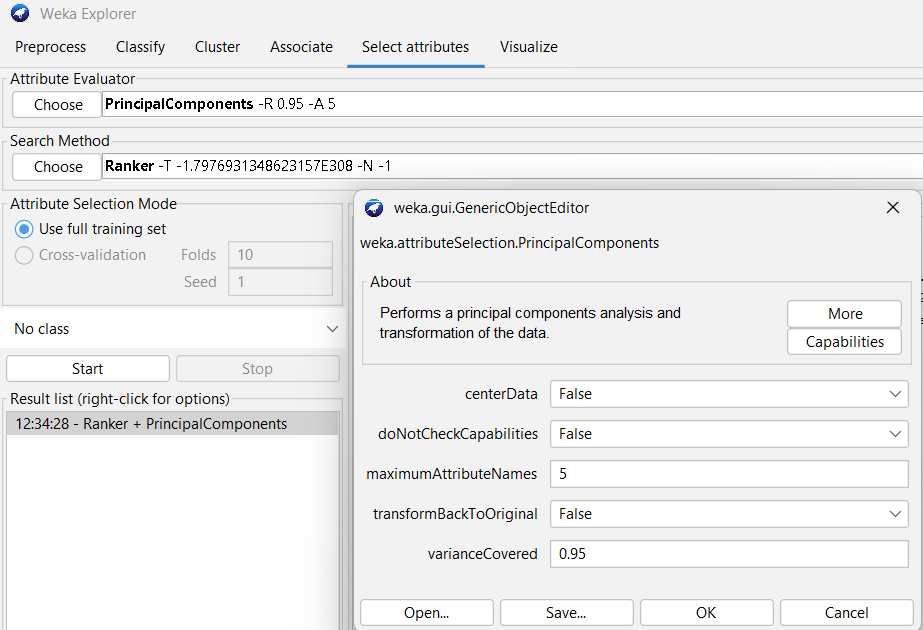
**Weka :**

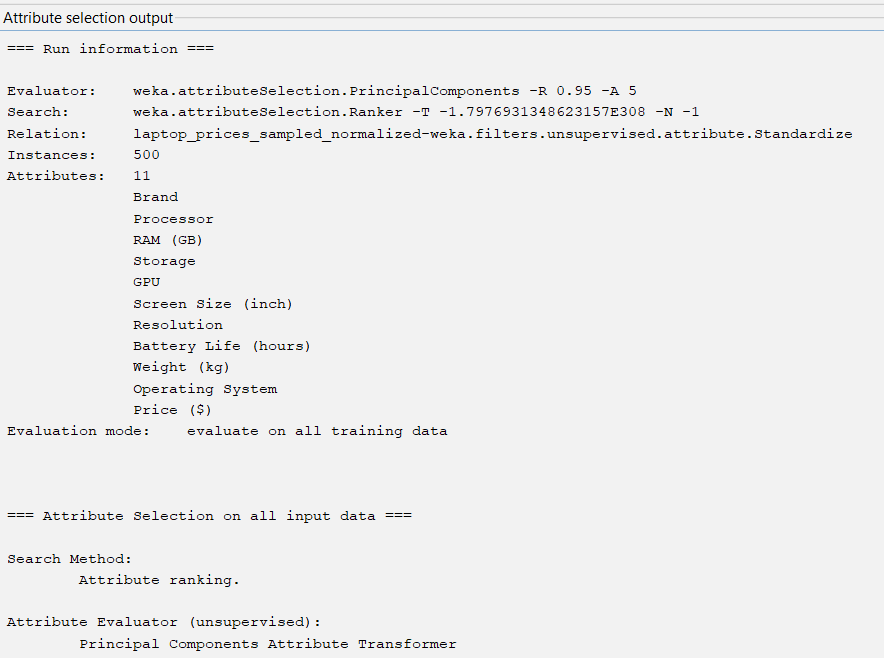
**K MEANS:**

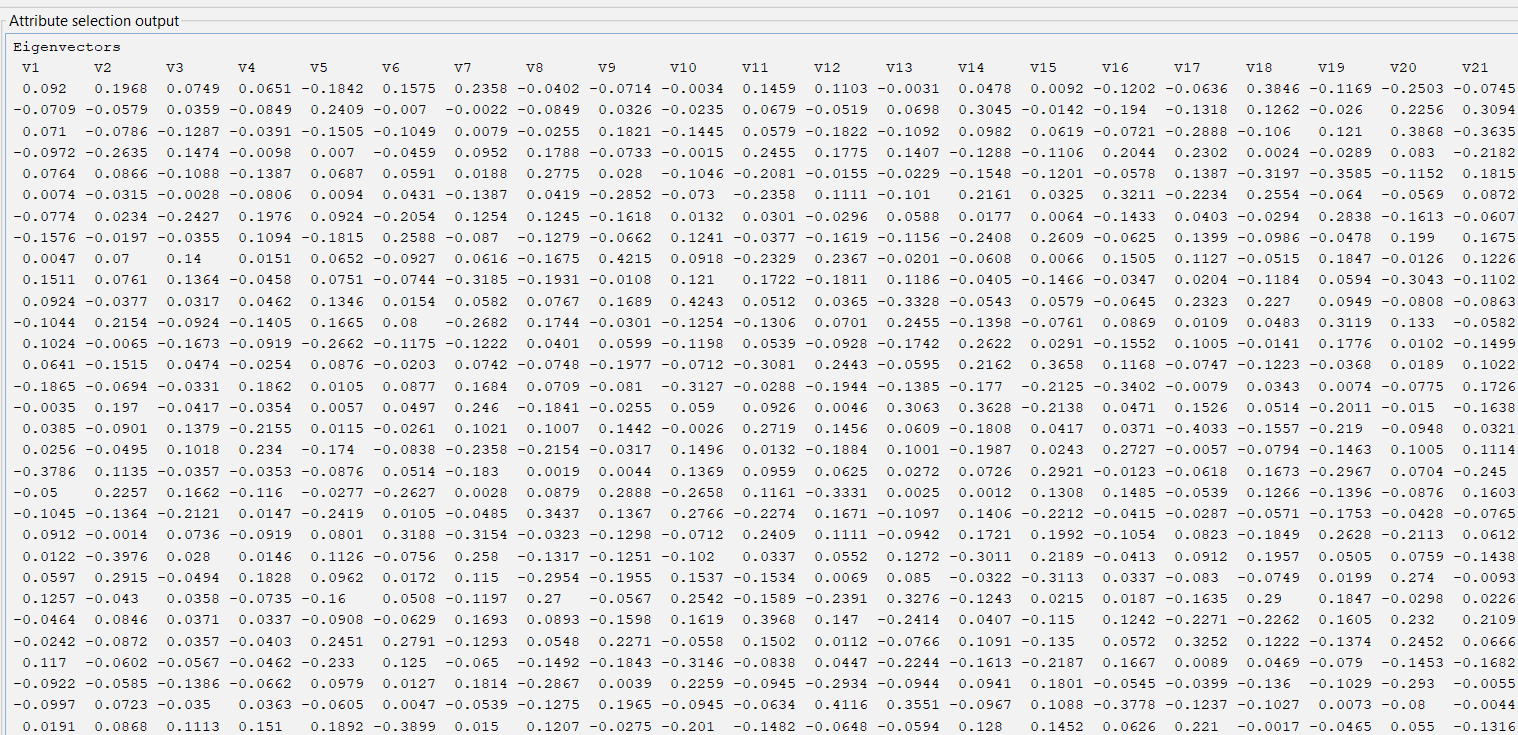
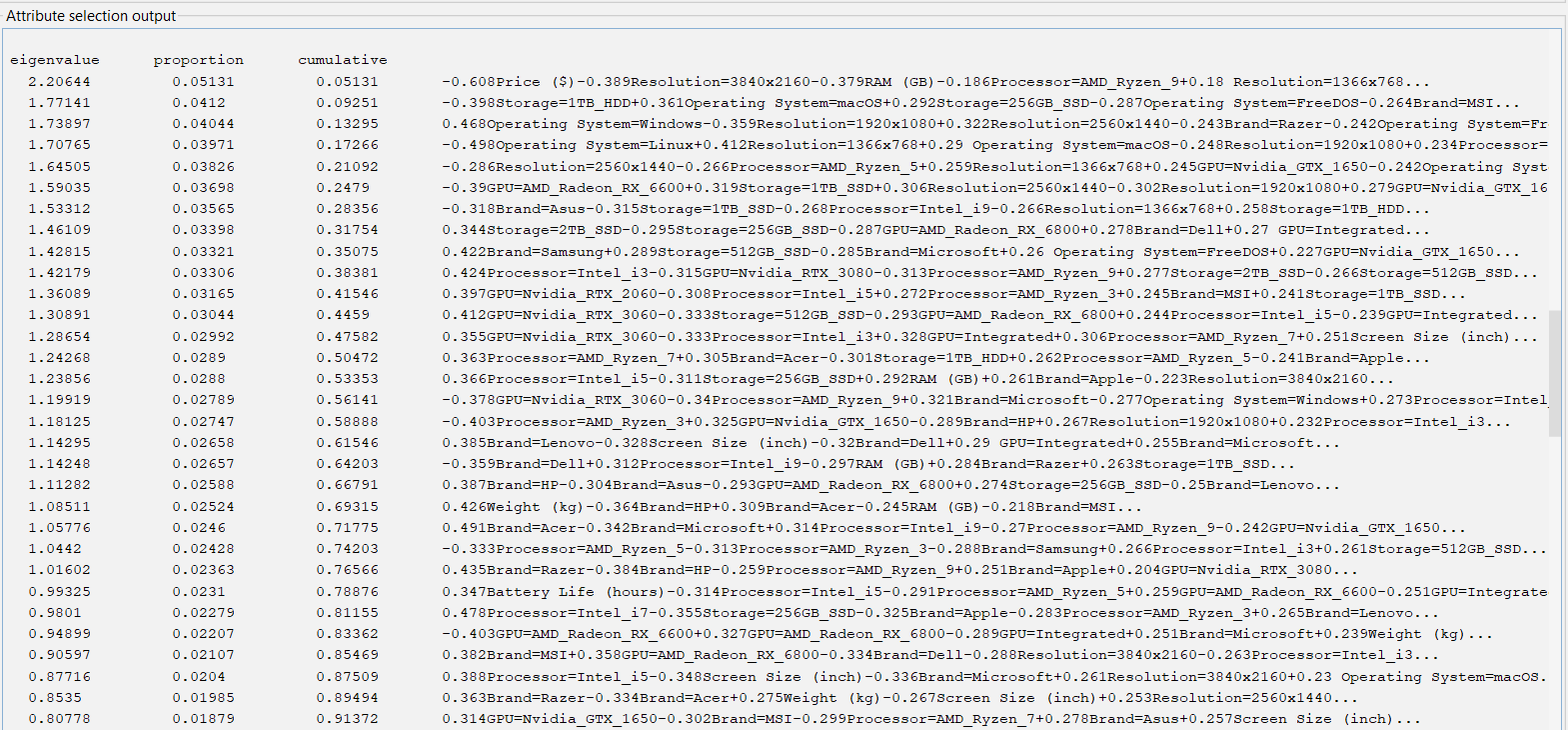
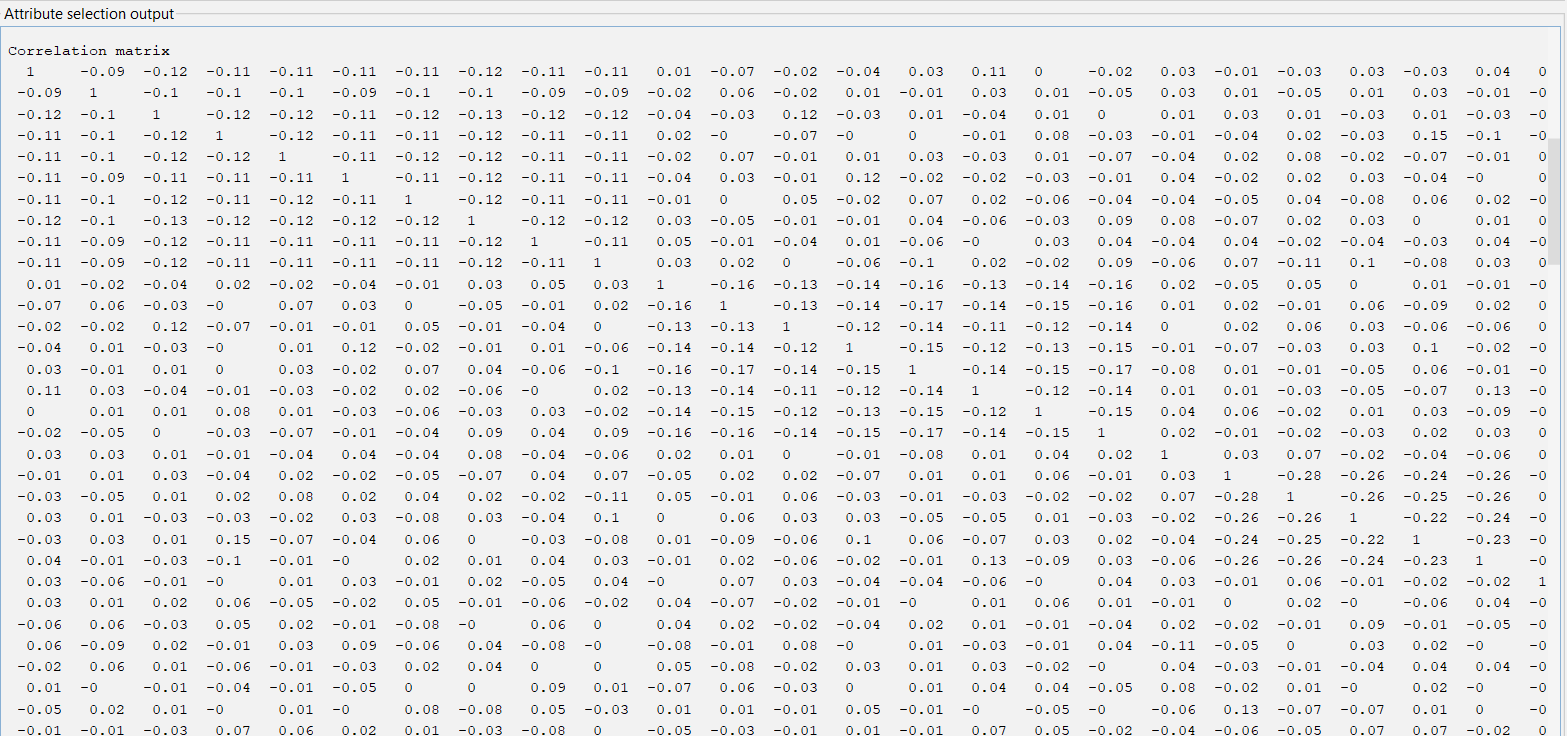
**OUTPUT:**





PCA  
OUTPUT: 

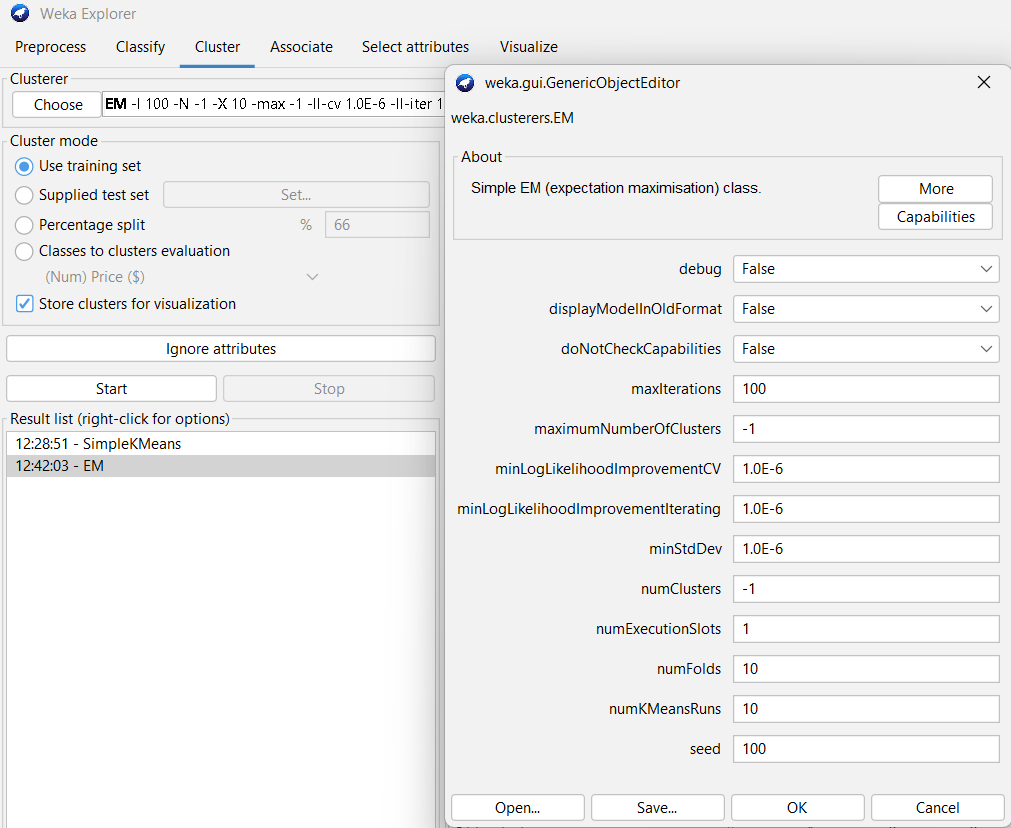


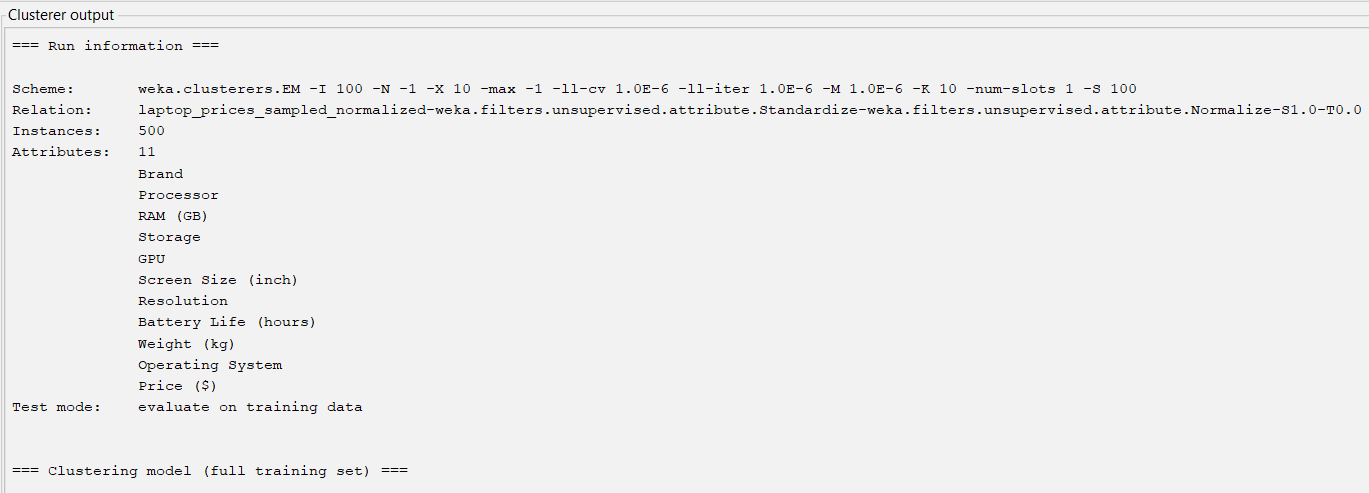
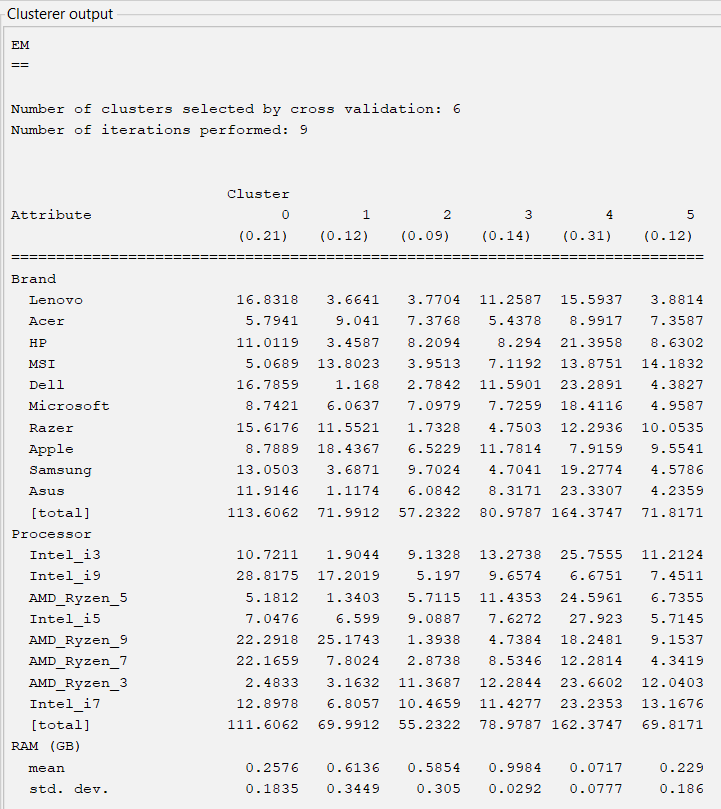


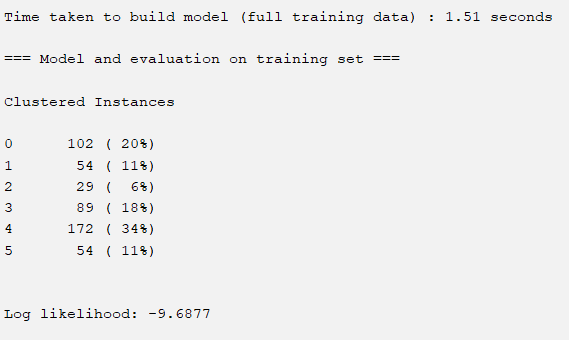
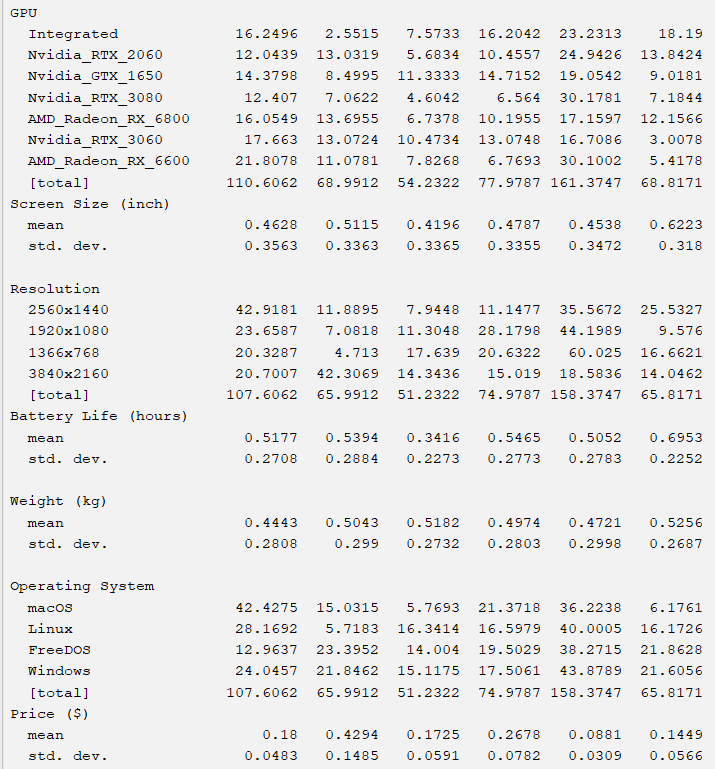


EM:

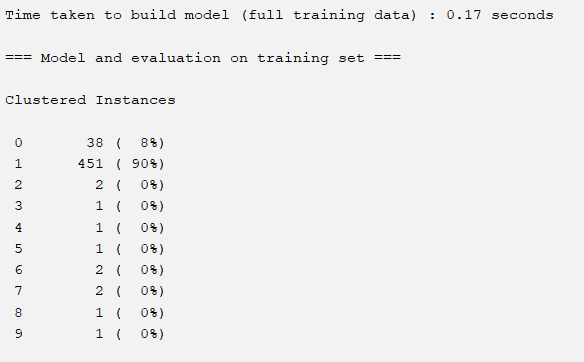
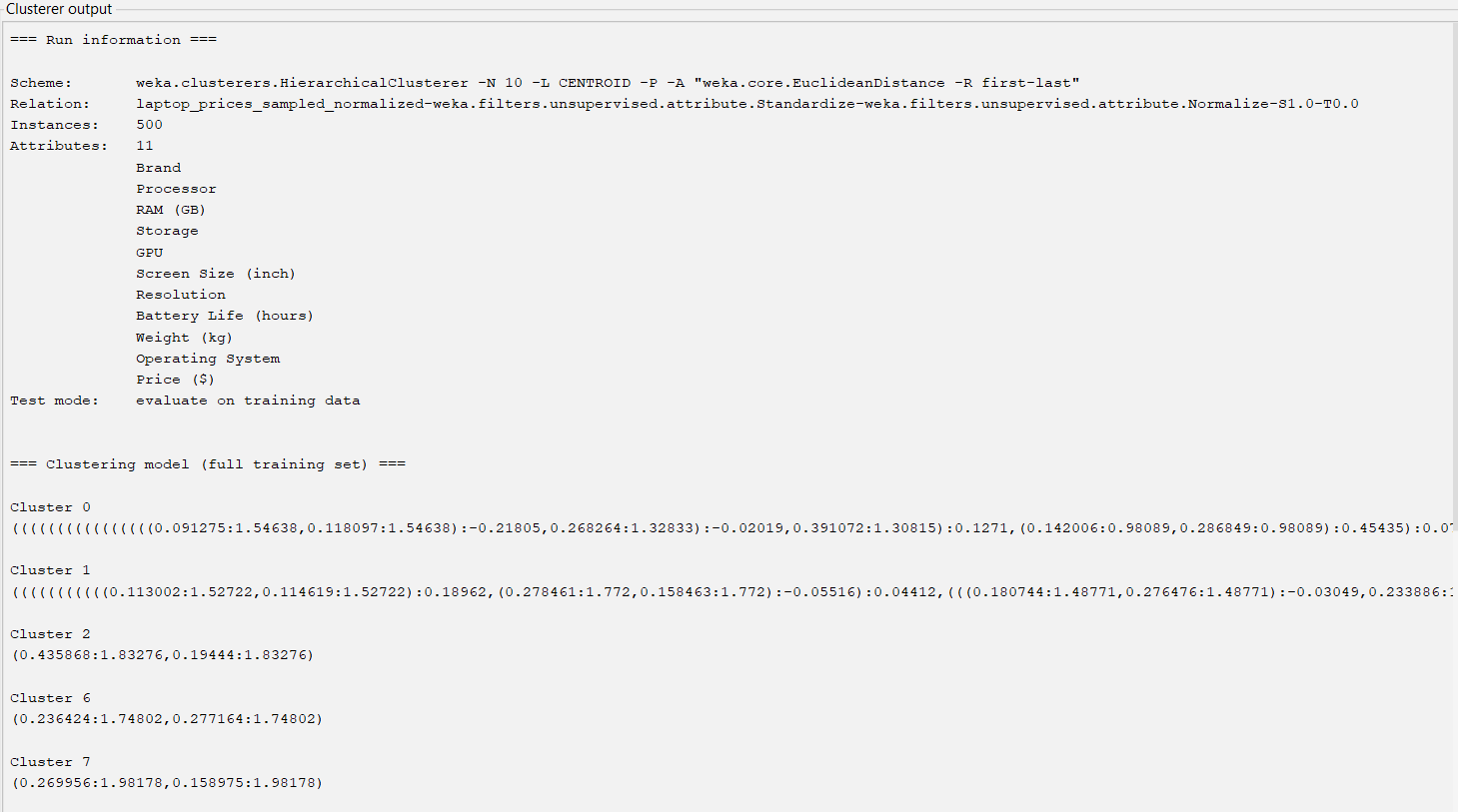
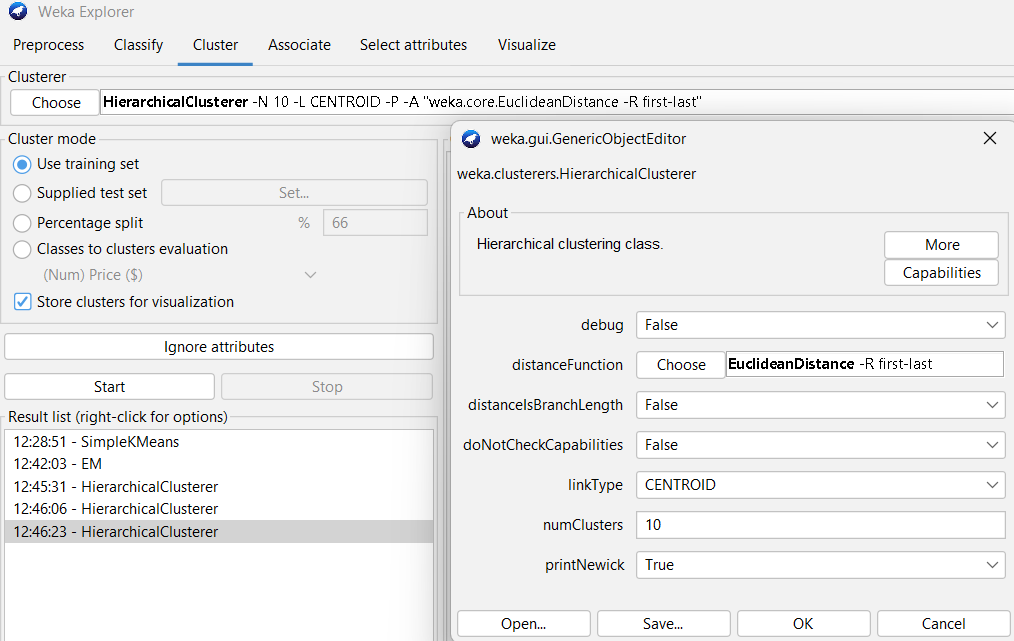
OUTPUT:



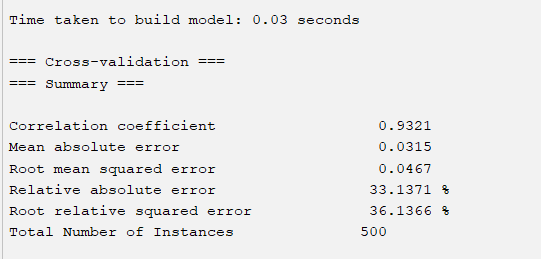


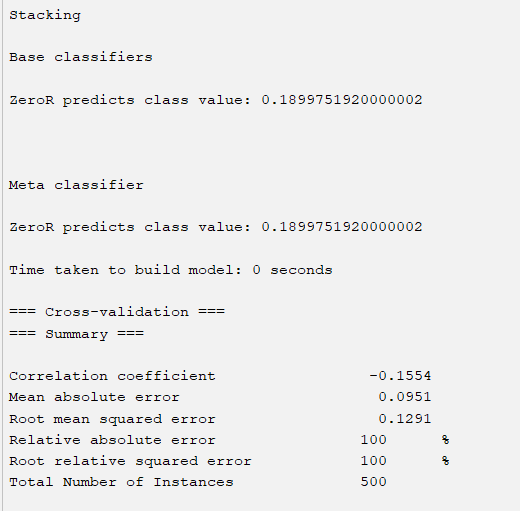


HIERACHICAL CUSTERING:

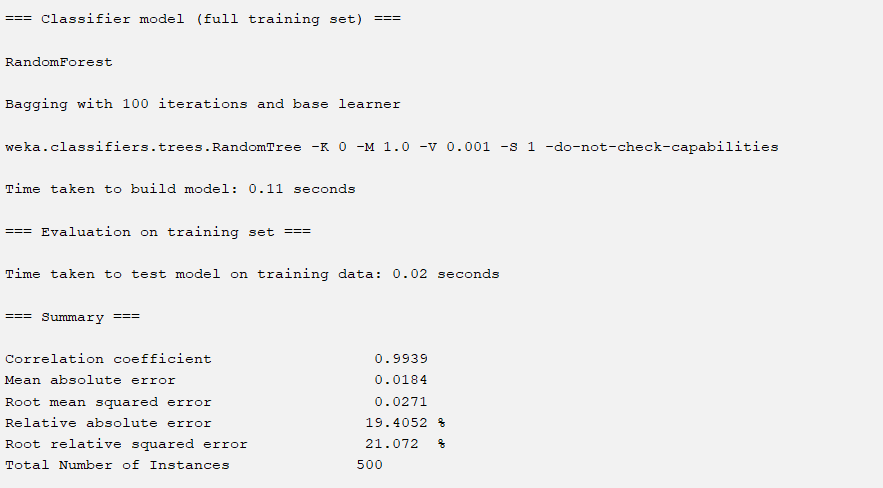
OUTPUT:  


LINEAR REGRESSION:

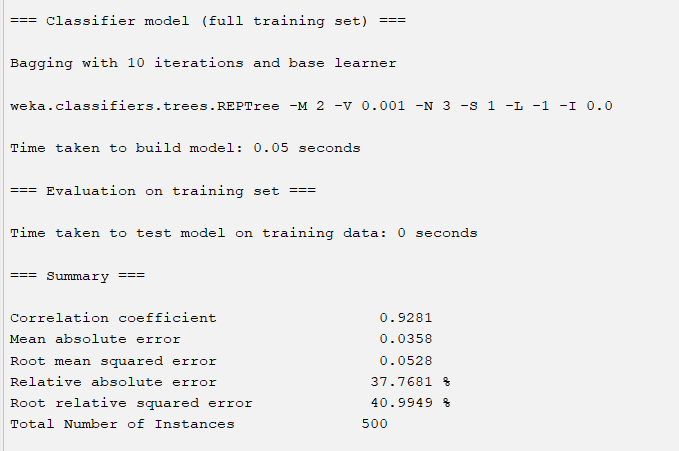


STACKING:  


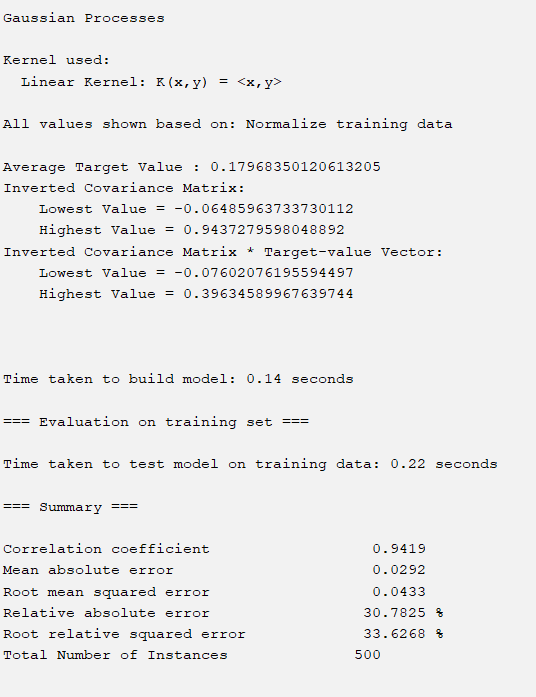
RANDOM FOREST



BAGGING:



GAUSSIAN PROCESS:



**Result :**

Support Vector Machine (SVM) and Random Forest achieved the highest accuracy in predicting laptop prices. K-Means effectively grouped laptops into meaningful clusters based on specifications