**An Analysis of Multiple Variables’ Impact on Valorant Win Prediction**

1. **Introduction**

Valorant is a competitive first-person shooter game developed by Riot Games that was released in June 2020. Valorant is a team-based tactical shooter that emphasizes precise aim, strategic coordination, and the effective utilization of unique agent abilities. The game features a diverse cast of agents to choose from, each with unique abilities and different weapons that can be used strategically to gain an advantage over the enemy team. Players are divided into two teams, the attackers and defenders, and engage in a series of rounds with the objective of either planting or defusing a bomb or eliminating the enemy team. The shooting mechanics are a crucial part of playing Valorant, requiring players to have good aim and recoil control. The game also emphasizes strategic positioning, map control, and intelligent use of utilities. Players must communicate and coordinate with their team, making split-second decisions and adapting to the ever-changing dynamics of the round. After every match, players can get a detailed report of their performance during the match. This leads to the question, can one player’s stats predict the outcome of the match. Which stats are important in predicting whether a match ends in a win or loss?

A similar analysis was done by Michael Song to predict what agent/role a player plays based on their statistics. The data consists of various statistics such as headshot percent, clutch percent, KDA, and various round-level statistics from 1,647 professional Valorant players world-wide (Song, 2022). The analysis focuses on several key statistics like Average Damage per Round (ADR), Assists per Round (APR), Clutch Win Rate, etc., to understand the relationship between agent selection and player performance. To predict the most played agent/role based on player statistics, a support vector machines (SVM) was trained. The model achieved higher accuracy than a baseline model, indicating that certain statistics can provide insights into agent selection.

1. **Data Source**

The data consists of various player statistics about the match, such as average combat score (ACS), agent I played, map the match was on, kill/death ratio, etc. This data was from my personal competitive history obtained on tracker.gg (Hotu Potu#Yum’s Competitive Multiplayer Overview - Valorant Tracker, n.d.). The data was cleaned using R. The data frame has 16 variables:

Agent: 21 agents

Map: 9 maps randomly assigned to each match

Placement: First to tenth in comparison to other player’s ACS

Duration (min): length of game

Average Rank of Match: averages the rank of all the players in the match

Average Combat Score (ACS): based on kills, multi-kills, and damage done to the enemy team (Everything You Need to Know About Average Combat Score in Valorant, n.d.)

A (Assists): teammate killing an enemy you’ve done at least 50 damage to

K/D (Kill/Death ratio)

Average Damage Per Round (ADR)

HS%: head shot percentage

KAST%: percent of rounds where you got a kill, assist, survived, or traded

First Kill: first person on the team to get a kill during a round

First Death: first person to die during a round

MK (Multikill): get more than 1 kill during a round

Econ: in-game currency based on performance

Results: binary response variable where win is 1 and lose is 0.

All character variables were changed to win rates which can be seen in Appendix A.

1. **Analysis**

Support vector machines and logistic regression were used to analyze the dataset. Support vector machines (SVMs) are machine learning algorithms used for classification and regression tasks. SVMs are effective in dealing with complex datasets and can handle both linearly and non-linearly separable data. A hyperplane is a decision boundary that separates the data points into different classes in SVM (Augmented Startups, 2017). Support vectors are the crucial data points closest to the hyperplane, from both the positive and negative classes that influence the position and orientation of the hyperplane (Wikipedia contributors, 2023). Different kernels for the SVM are used based on the type of relationship between the response variables and predictors. A logistic regression model is used to predict the probability of a binary outcome based on one or more predictor variables. The stepwise selection method was used in the logistic regression model to evaluate the importance of each predictor variable.

The logistic regression and stepwise selection computations were done in R, as seen in Appendix B. After the stepwise selection was done on the logistic regression using the full model with all the predictor variables, another logistic regression was done using the reduced model with some of the original predictor variables removed by stepwise selection. Using the AIC values to compare the logistic regression of the full model and the reduced model, it can be seen that the reduced model is the better model since the AIC value is smaller for the reduced model.

SVM computations were done in R and Python. The SVM model was then cross-validated to assess its performance and generalize the results of the model. The results of the SVM model in Python can be seen in Appendix C. The SVM computations were done on the reduced and full models. In R, the accuracy of the reduced and full models was equivalent. However, in Python, the reduced model was shown to have consistently higher accuracy than the full model. Both the cross-validation accuracy scores were similar for the reduced and full models in R and Python.

1. **Prediction**

SVM is the machine learning model used in this analysis to predict the result of the match based on a single player’s statistics. To determine what kernel was best for the model, I tested a non-linear kernel, gamma, as well as the linear kernel. The SVM model using the linear kernel produced greater accuracy, which means the relationship between the response variable and predictor variables is most likely linear. The data was then split into 80/20 train/test sets, where 80% of the data was used to train the SVM model and 20% was used to test the model. This was done on the full model, which has all the predictor variables listed in “Data Source”, and the reduced model, which only has the variables: Agent, Map, Placement, Average Rank of Match, Average Combat Score, K/D, KAST%, First Death, and Results. In R, the accuracy of the model resulted in 73% for both the reduced and full models. In Python, the accuracy of the reduced model was around 85%, whereas in the full model, the accuracy was about 75%.

1. **Conclusion**

To conclude, there are significant player statistics that can be used to predict the outcome of a match with great accuracy. Even though Valorant is a team-based game, this shows that one player can make a difference in whether the match ends in a win or a loss.

Further work can be done to improve this model by incorporating more detailed statistics on how the player plays during each round, like weapon choice, utility usage, etc., to accurately predict the outcome of the match before the match ends or even starts.

In evaluating a player’s performance, these observations provide us with intriguing insights. Some players do have their super bad games and good games, but overall, their performance can be used for many analyses, such as gambling purposes, legally.

1. **Appendix A. Win, Losses, and Frequency of Each Agent, Map, Placement, and Average Rank of Match**

This appendix consists of graphical representations computed in R.

A picture containing screenshot, colorfulness, rectangle

Description automatically generated

Figure 1. The figure shows how frequently I play each agent as well as how often I win or lose playing them. The y-axis represents frequency, and the x-axis represents the outcome of the match.

A picture containing screenshot, colorfulness, text, rectangle

Description automatically generated

Figure 2. The figure shows how frequently I play on each map as well as how often I win or lose playing on them. The y-axis represents frequency, and the x-axis represents the outcome of the match.

A picture containing screenshot, text, colorfulness, rectangle

Description automatically generated

Figure 3. The figure shows how frequently where I placed at the end of the match and how often I win or lose depending on where I placed. The y-axis represents frequency, and the x-axis represents the outcome of the match.

A picture containing screenshot, colorfulness, text, rectangle

Description automatically generated

Figure 3. The figure shows how frequently I play in that average rank of the match and how often I win or lose depending on that average. The y-axis represents frequency, and the x-axis represents the outcome of the match.

1. **Appendix B. Logistic Regression and Stepwise Selection Computations**

This appendix consists of computations from R.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 1. This computation shows the logistic regression of the dataset.

A screenshot of a computer

Description automatically generated

Figure 2. This computation shows the stepwise selection method used on the logistic regression model.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 3. This computation shows the logistic regression with reduced predictor variables based on the stepwise selection.

1. **Appendix C. Support Vector Machines Computation Results**

This appendix consists of SVM computations and the cross-validation results.

A picture containing text, screenshot, software, display

Description automatically generated

Figure 1. This figure represents the results of the SVM model using the full model.

A picture containing text, screenshot, font

Description automatically generated

Figure 2. This figure represents the cross-validation results of the SVM model using the full model.

A screenshot of a computer program

Description automatically generated with medium confidence

Figure 3. This figure represents the results of the SVM model using the reduced model from stepwise selection method.

A picture containing text, screenshot, font

Description automatically generated

Figure 2. This figure represents the cross-validation results of the SVM model using the reduced model from stepwise selection method.

1. **Appendix D. R Code Used to Clean the Dataset and Logistic Regression and Stepwise Selection Computation**

library(readxl)

library(easypackages)

library(dplyr)

library(olsrr)

library(caret)

library(leaps)

library(MASS)

library(tidyverse)

library(car)

libraries("ggplot2","MASS","aod","ResourceSelection")

df1 <- read\_excel("C:/Users/ddinh/OneDrive/Documents/EXST 7087/Final Project/Data/DD\_CompetitiveCareer\_02\_25\_2023.xlsx")

str(df1)

df2 <- df1[, -c(1,5,10,11)]

df2$Results[df2$Results == "W"] <- 1

df2$Results[df2$Results == "L"] <- 0

df2$Results[df2$Results == "D"] <- ""

df2$Agent[df2$Agent == "sage"] <- "Sage"

df2$Agent[df2$Agent == "reyna"] <- "Reyna"

df2$Agent[df2$Agent == "brimstone"] <- "Brimstone"

df2

df2[df2 == ""] <- NA

df3 <- df2[complete.cases(df2), ]

str(df3)

response\_index <- which(names(df3) == "Results")

df <- df3[, c(response\_index, 1:(response\_index-1), (response\_index+1):ncol(df3))]

#df3<- df2$Results <- ifelse(df2$Results == "W", 1, ifelse(df2$Results == "L", 0, df2$Results))

#barplot(df3$Results, names.arg = df3$Agent, main = "Bar Plot", xlab = "Agent", ylab = "Results")

freq\_table <- table(df$Agent, df$Results)

freq\_table

row\_perc <- prop.table(freq\_table, margin = 1) \* 100

Breach <- row\_perc[1,2]

Brimstone <- row\_perc[2,2]

Cypher <- row\_perc[3,2]

Jett <- row\_perc[4,2]

Killjoy <- row\_perc[5,2]

Neon <- row\_perc[6,2]

Omen <- row\_perc[7,2]

Raze <- row\_perc[8,2]

Reyna <- row\_perc[9,2]

Sage <- row\_perc[10,2]

Skye <- row\_perc[11,2]

Sova <- row\_perc[12,2]

Viper <- row\_perc[13,2]

plot <- barplot(freq\_table, beside = FALSE, legend = TRUE, col = rainbow(13), main = "Agent Win Rate")

ggsave("C:/Users/ddinh/OneDrive/Documents/EXST 7087/Final Project/Results/AgentWR.jpeg",plot)

freq\_table2 <- table(df$Map, df$Results)

freq\_table2

row\_perc2 <- prop.table(freq\_table2, margin = 1) \* 100

Ascent <- row\_perc2[1,2]

Bind <- row\_perc2[2,2]

Breeze <- row\_perc2[3,2]

Fracture <- row\_perc2[4,2]

Haven <- row\_perc2[5,2]

Icebox <- row\_perc2[6,2]

Lotus <- row\_perc2[7,2]

Pearl <- row\_perc2[8,2]

Split <- row\_perc2[9,2]

plot2 <- barplot(freq\_table2, beside = FALSE, legend = TRUE, col = rainbow(9), main = "Map Win Rate")

ggsave(filename = "C:/Users/ddinh/OneDrive/Documents/EXST 7087/Final Project/Results/MapWR.jpg", plot = plot2, dpi = 300)

freq\_table3 <- table(df$Placement, df$Results)

freq\_table3

row\_perc3 <- prop.table(freq\_table3, margin = 1) \* 100

Placement1 <- row\_perc3[1,2]

Placement2 <- row\_perc3[2,2]

Placement3 <- row\_perc3[3,2]

Placement4 <- row\_perc3[4,2]

Placement5 <- row\_perc3[5,2]

Placement6 <- row\_perc3[6,2]

Placement7 <- row\_perc3[7,2]

Placement8 <- row\_perc3[8,2]

Placement9 <- row\_perc3[9,2]

Placement10 <- row\_perc3[10,2]

plot3 <- barplot(freq\_table3, beside = FALSE, legend = TRUE, col = rainbow(10), main = "Placement Win Rate")

ggsave(filename = "C:/Users/ddinh/OneDrive/Documents/EXST 7087/Final Project/Results/PlacementWR.jpg", plot = plot3, dpi = 300)

freq\_table4 <- table(df$`Average Rank of Match`, df$Results)

freq\_table4

row\_perc4 <- prop.table(freq\_table4, margin = 1) \* 100

BronzeI <- row\_perc4[1,2]

BronzeII <- row\_perc4[2,2]

BronzeIII <- row\_perc4[3,2]

GoldI <- row\_perc4[4,2]

GoldII <- row\_perc4[5,2]

GoldIII <- row\_perc4[6,2]

PlatinumI <- row\_perc4[7,2]

PlatinumII <- row\_perc4[8,2]

PlatinumIII <- row\_perc4[9,2]

SilverI <- row\_perc4[10,2]

SilverII <- row\_perc4[11,2]

SilverIII <- row\_perc4[12,2]

plot4<- barplot(freq\_table4, beside = FALSE, legend = TRUE, col = rainbow(12), main = "Average Rank of Match Win Rate")

ggsave(filename = "C:/Users/ddinh/OneDrive/Documents/EXST 7087/Final Project/Results/RankWR.jpg", plot = plot3, dpi = 300)

overallwinrate <- mean(df$Results == 1) \* 100

df$Agent[df$Agent == "Breach"] <- Breach

df$Agent[df$Agent == "Brimstone"] <- Brimstone

df$Agent[df$Agent == "Cypher"] <- Cypher

df$Agent[df$Agent == "Jett"] <- Jett

df$Agent[df$Agent == "Killjoy"] <- Killjoy

df$Agent[df$Agent == "Neon"] <- Neon

df$Agent[df$Agent == "Omen"] <- Omen

df$Agent[df$Agent == "Raze"] <- Raze

df$Agent[df$Agent == "Reyna"] <- Reyna

df$Agent[df$Agent == "Sage"] <- Sage

df$Agent[df$Agent == "Skye"] <- Skye

df$Agent[df$Agent == "Sova"] <- Sova

df$Agent[df$Agent == "Viper"] <- Viper

df$Map[df$Map == "Ascent"] <- Ascent

df$Map[df$Map == "Bind"] <- Bind

df$Map[df$Map == "Breeze"] <- Breeze

df$Map[df$Map == "Fracture"] <- Fracture

df$Map[df$Map == "Haven"] <- Haven

df$Map[df$Map == "Icebox"] <- Icebox

df$Map[df$Map == "Lotus"] <- Lotus

df$Map[df$Map == "Pearl"] <- Pearl

df$Map[df$Map == "Split"] <- Split

df$Placement[df$Placement == "1"] <- Placement1

df$Placement[df$Placement == "2"] <- Placement2

df$Placement[df$Placement == "3"] <- Placement3

df$Placement[df$Placement == "4"] <- Placement4

df$Placement[df$Placement == "5"] <- Placement5

df$Placement[df$Placement == "6"] <- Placement6

df$Placement[df$Placement == "7"] <- Placement7

df$Placement[df$Placement == "8"] <- Placement8

df$Placement[df$Placement == "9"] <- Placement9

df$Placement[df$Placement == "10"] <- Placement10

df$`Average Rank of Match`[df$`Average Rank of Match` == "Bronze I"] <- BronzeI

df$`Average Rank of Match`[df$`Average Rank of Match` == "Bronze II"] <- BronzeII

df$`Average Rank of Match`[df$`Average Rank of Match` == "Bronze III"] <- BronzeIII

df$`Average Rank of Match`[df$`Average Rank of Match` == "Gold I"] <- GoldI

df$`Average Rank of Match`[df$`Average Rank of Match` == "Gold II"] <- GoldII

df$`Average Rank of Match`[df$`Average Rank of Match` == "Gold III"] <- GoldIII

df$`Average Rank of Match`[df$`Average Rank of Match` == "Platinum I"] <- PlatinumI

df$`Average Rank of Match`[df$`Average Rank of Match` == "Platinum II"] <- PlatinumII

df$`Average Rank of Match`[df$`Average Rank of Match` == "Platinum III"] <- PlatinumIII

df$`Average Rank of Match`[df$`Average Rank of Match` == "Silver I"] <- SilverI

df$`Average Rank of Match`[df$`Average Rank of Match` == "Silver II"] <- SilverII

df$`Average Rank of Match`[df$`Average Rank of Match` == "Silver III"] <- SilverIII

cols\_to\_convert <- names(df)[-1]

df[, cols\_to\_convert] <- lapply(df[, cols\_to\_convert], as.numeric)

str(df)

write.csv(df, file = "C:/Users/ddinh/OneDrive/Documents/EXST 7087/Final Project/Data/DD\_CleanCompetitiveCareer\_05\_01\_2023.csv", row.names = FALSE)

logmodel <- glm(as.factor(Results) ~ ., data = df, family = "binomial")

stepboth.model <- stepAIC(logmodel, direction = "both",

trace = FALSE)

stepwise\_summary <- summary(stepboth.model)

log\_summary <- summary(logmodel)

stepwise\_summary

log\_summary

#log\_coefficients <- coef(log\_summary)

#log\_statistics <- coef(summary(log\_summary))

#log\_summary\_data <- data.frame(Coefficients = log\_coefficients, log\_statistics)

#write.csv(log\_summary\_data, file = "C:/Users/ddinh/OneDrive/Documents/EXST 7087/Final Project/Results/logistic\_regression\_summary.csv", row.names = FALSE)

#stepwise\_coefficients <- coef(stepwise\_summary)

#stepwise\_statistics <- coef(summary(stepwise\_summary))

#stepwise\_summary\_data <- data.frame(Coefficients = stepwise\_coefficients, stepwise\_statistics)

#write.csv(stepwise\_summary\_data, file = "C:/Users/ddinh/OneDrive/Documents/EXST 7087/Final Project/Results/stepwise\_summary.csv", row.names = FALSE)

reduced\_df <- df[, -c(5,8,10,11,13,15,16)]

write.csv(reduced\_df, file = "C:/Users/ddinh/OneDrive/Documents/EXST 7087/Final Project/Data/DD\_ReducedCleanCompetitiveCareer\_05\_01\_2023.csv", row.names = FALSE)

logmodel2 <- glm(as.factor(Results) ~ ., data = reduced\_df, family = "binomial")

summary(logmodel2)

1. **Appendix E. R Code to Compute the SVM on the Full Model**

library(e1071)

library(caret)

library(kernlab)

df= read.csv("C:/Users/ddinh/OneDrive/Documents/EXST 7087/Final Project/Data/DD\_CleanCompetitiveCareer\_05\_01\_2023.csv")

# Split the data into training and testing sets

set.seed(123) # Set a seed for reproducibility

train\_indices <- sample(nrow(df), nrow(df) \* 0.8) # 70% for training

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

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

# Train the SVM model with all columns except the first as predictors

svm\_model <- svm(Results ~ ., data = train\_data, type = "C-classification", kernel = "linear")

# Make predictions on the test data

svm\_pred <- predict(svm\_model, newdata = test\_data)

# Evaluate the SVM model

accuracy <- sum(svm\_pred == test\_data$Results) / length(svm\_pred)

print(paste("Accuracy:", accuracy))

# Perform cross-validation

svm\_cv <- train(Results ~ ., data = df, method = "svmLinear", trControl = trainControl(method = "cv", number = 5))

# View the cross-validated results

print(svm\_cv)

# Lower RMSE values indicate better model performance, as it means the model's predictions are closer to the actual values on average.

# Confusion matrix after cross validating

test\_data[, 1] <- factor(test\_data[, 1], levels = levels(svm\_pred))

# Create the confusion matrix

con\_mat <- confusionMatrix(svm\_pred, test\_data[, 1])

# Print the confusion matrix

print(con\_mat)

1. **Appendix F. R Code to Compute the SVM on the Reduced Model**

library(e1071)

library(caret)

library(kernlab)

df= read.csv("C:/Users/ddinh/OneDrive/Documents/EXST 7087/Final Project/Data/DD\_ReducedCleanCompetitiveCareer\_05\_01\_2023.csv")

# Split the data into training and testing sets

set.seed(123) # Set a seed for reproducibility

train\_indices <- sample(nrow(df), nrow(df) \* 0.8) # 70% for training

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

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

# Train the SVM model with all columns except the first as predictors

svm\_model <- svm(Results ~ ., data = train\_data, type = "C-classification", kernel = "linear")

# Make predictions on the test data

svm\_pred <- predict(svm\_model, newdata = test\_data)

# Evaluate the SVM model

accuracy <- sum(svm\_pred == test\_data$Results) / length(svm\_pred)

print(paste("Accuracy:", accuracy))

# Perform cross-validation

svm\_cv <- train(Results ~ ., data = df, method = "svmLinear", trControl = trainControl(method = "cv", number = 5))

# View the cross-validated results

print(svm\_cv)

# Lower RMSE values indicate better model performance, as it means the model's predictions are closer to the actual values on average.

# Confusion matrix after cross validating

test\_data[, 1] <- factor(test\_data[, 1], levels = levels(svm\_pred))

# Create the confusion matrix

con\_mat <- confusionMatrix(svm\_pred, test\_data[, 1])

# Print the confusion matrix

print(con\_mat)

1. **Appendix G. Python Code to Compute the SVM on the Full Model**

import pandas as pd

import numpy as np

#import os

#import matplotlib.pyplot as plt

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

from sklearn.svm import SVC

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.model\_selection import StratifiedShuffleSplit

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import cross\_val\_score, cross\_val\_predict

#from sklearn.model\_selection import KFold

#%%===== Loading data pre-processing and re arranging ===========================================================

df1 = pd.read\_csv("C:/Users/ddinh/OneDrive/Documents/EXST 7087/Final Project/Data/DD\_CleanCompetitiveCareer\_05\_01\_2023.csv")

X = df1.drop('Results', axis=1)

y = df1['Results']

#%%===============Using Linear Kernel ===============================================================

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20)

svclassifier = SVC(kernel='linear')

svclassifier.fit(X\_train, y\_train)

y\_pred = svclassifier.predict(X\_test)

print(y\_pred)

print(confusion\_matrix(y\_test,y\_pred))

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

#%%===================================== Using a gamma kernel ===========================================

C\_range = np.logspace(-3, 10, 3)

gamma\_range = np.logspace(-2, 3, 2)

param\_grid = dict(gamma=gamma\_range, C=C\_range)

cv = StratifiedShuffleSplit(n\_splits=5, test\_size=0.2, random\_state=42)

grid = GridSearchCV(SVC(), param\_grid=param\_grid, cv=cv)

grid.fit(X\_train, y\_train)

print("The best parameters are %s with a score of %0.2f"

% (grid.best\_params\_, grid.best\_score\_))

svclassifier2 = SVC(C=3162.2776601683795,gamma=0.01)

svclassifier2.fit(X\_train, y\_train)

y\_pred = svclassifier2.predict(X\_test)

print(confusion\_matrix(y\_test,y\_pred))

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

# Using the linear kernel provided better accuracy than the gamma bc the relationship between the response and predictor variables are more linear

#%%========================= Cross Validating =====================================================

cross\_val\_scores = cross\_val\_score(svclassifier, X\_train, y\_train, cv=5)

# View the cross-validated scores

print("Cross-Validated Accuracy Scores:", cross\_val\_scores)

print("Mean Accuracy:", cross\_val\_scores.mean())

# Get cross-validated predictions

cross\_val\_predictions = cross\_val\_predict(svclassifier, X\_train, y\_train, cv=5)

# Calculate the confusion matrix

conf\_mat = confusion\_matrix(y\_train, cross\_val\_predictions)

# Print the confusion matrix

print("Confusion Matrix:", conf\_mat)

1. **Appendix H. Python Code to Compute the SVM on the Reduced Model**

#%%===== Load Appropriate Libraries ===============================================================================

import pandas as pd

#import numpy as np

#import os

#import matplotlib.pyplot as plt

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

from sklearn.svm import SVC

from sklearn.metrics import classification\_report, confusion\_matrix

#from sklearn.model\_selection import StratifiedShuffleSplit

#from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import cross\_val\_score, cross\_val\_predict

#from sklearn.model\_selection import KFold

#%%===== Loading data pre-processing and re arranging ===========================================================

df1 = pd.read\_csv("C:/Users/ddinh/OneDrive/Documents/EXST 7087/Final Project/Data/DD\_ReducedCleanCompetitiveCareer\_05\_01\_2023.csv")

X = df1.drop('Results', axis=1)

y = df1['Results']

#%%===============Using Linear Kernel ===============================================================

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20)

svclassifier = SVC(kernel='linear')

svclassifier.fit(X\_train, y\_train)

y\_pred = svclassifier.predict(X\_test)

print(y\_pred)

print(confusion\_matrix(y\_test,y\_pred))

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

#%%========================= Cross Validating =====================================================

cross\_val\_scores = cross\_val\_score(svclassifier, X\_train, y\_train, cv=5)

# View the cross-validated scores

print("Cross-Validated Accuracy Scores:", cross\_val\_scores)

print("Mean Accuracy:", cross\_val\_scores.mean())

# Get cross-validated predictions

cross\_val\_predictions = cross\_val\_predict(svclassifier, X\_train, y\_train, cv=5)

# Calculate the confusion matrix

conf\_mat = confusion\_matrix(y\_train, cross\_val\_predictions)

# Print the confusion matrix

print("Confusion Matrix:", conf\_mat)

1. **References**

Augmented Startups. (2017, August 15). *Support Vector Machine (SVM) in 7 minutes - Fun Machine Learning* [Video]. YouTube. <https://www.youtube.com/watch?v=Y6RRHw9uN9o>

*Everything You Need To Know About Average Combat Score In Valorant*. (n.d.). EsportsDriven.com. <https://esportsdriven.com/valorant/guides/394/everything-you-need-to-know-about-average-combat-score-in-valorant>

*hotu potu#yum’s Competitive Multiplayer Overview - Valorant Tracker*. (n.d.). Valorant Tracker. <https://tracker.gg/valorant/profile/riot/hotu%20potu%23yum/overview>

Song, M. (2022, January 7). An Analysis of Agent Impact on Performance in Pro Valorant. *Medium*. https://micsong.medium.com/an-analysis-of-agent-impact-on-performance-in-pro-valorant-c3b3cfa4aa2f#1866

Wikipedia contributors. (2023). Support vector machine. *Wikipedia*. <https://en.wikipedia.org/wiki/Support_vector_machine#References>