Introduction

The overarching goal of this exam is to estimate two deterministic functions of the two independent variables X_1 and X_2. The functions to be found are the mean $\mu(X_1, X_2) = E(Y)$ and variance $V(X_1, X_2) = Var(Y)$. There were two data sets given, one was the training, and the other was the testing. Both data sets consist of all numerical data points.
The specific problem of this exam is to use machine learning methods to predict the mean and variance as a function of X_1 and X_2. The predictions are then going to be tested against unknown "true" prediction values known only by the teaching staff.

EDA

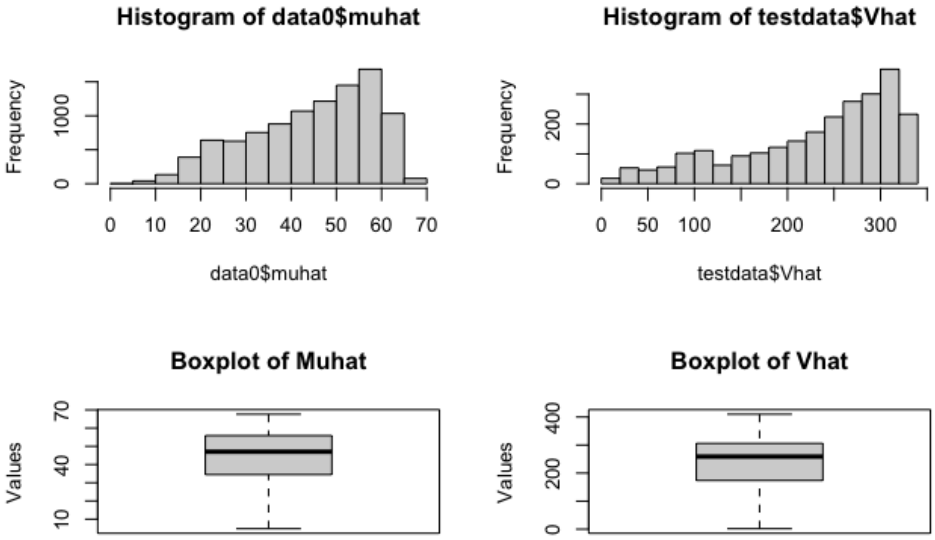
The training dataset contains 10,000 observations, each with 202 pieces of information. The first two columns represent two predictor variables, X_1, and X_2. The remaining columns are used to calculate two important statistical measures: the average (referred to as muhat) and the variance (referred to as Vhat) for each observation. The average gives an overall measure of the central tendency of the data, while the variance measures the spread or variability. For each observation, the average and variance are calculated based on the values in the remaining columns. After these calculations, the dataset is organized into a new table (data0) that includes X_1, X_2, and the calculated average and variance for each observation, providing a comprehensive summary of the data. This table will be used for further analysis in the exam.

The testing dataset contains 2,500 observations, each with 2 pieces of information. These two pieces represent the predictor variables X_1 and X_2. This dataset will be used to make predictions or perform further analysis, based on the previously defined models built from the data0 information.

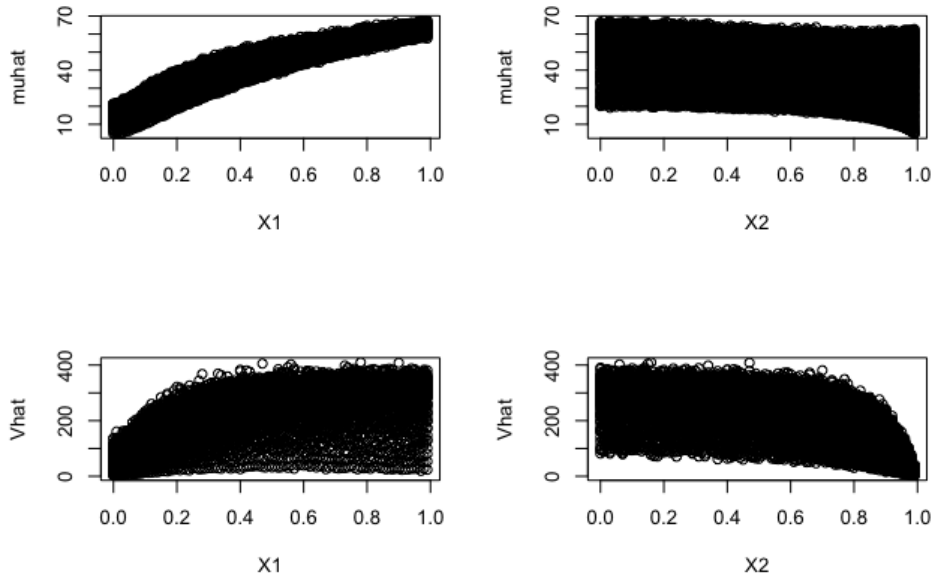
The data0 table was explored further. First off, the spread of the muhat and Vhat variables are as follows:

	Min	1st Qu	Median	Mean	3rd Qu	Max
muhat	4.75	34.47	47.03	44.41	55.90	67.63
Vhat	1.723	173.757	258.930	235.148	305.940	409.924

Looking at these numbers, it is seen that both medians for the muhat and Vhat are larger than the mean. This indicates that the data is left skewed. To make sure of this, a histogram of each column was created:



As expected, both columns are right skewed. Boxplots were also looked at to make sure there were not outliers – there were not any. How X_1 and X_2 interacted with muhat and Vhat was then looked at. They followed the following distributions:



As shown above, as X_1's value increases, so does the muhat value and the Vhat value until X_1 reaches about 0.5, and as X_2's value increases, muhat seems to slightly decrease and so does the Vhat value until around 0.7 where is sharply decreases.

Methods/Methodology

To predict the mean (muhat) and the variance (Vhat) of a set of outcomes using two input variables (X_1 and X_2), I explored and compared several machine learning models. Each model was selected for its unique strengths in capturing different types of patterns in the data. Throughout this process, 10-fold cross-validation was used to tune the models and evaluate their performance while guarding against overfitting.

In 10-fold cross-validation, the training dataset is divided into 10 equal parts or "folds." For each model, training is performed on 9 folds while the remaining fold is used for validation. This process is repeated 10 times, ensuring that each fold serves as the validation set exactly once. The average performance across all 10 iterations is then used as the basis for comparing models.

Target Transformations

The prediction targets were the row-wise mean (muhat) and variance (Vhat) of 200 outcome variables per row. Rather than modeling these values directly, I chose to predict their logarithms: log(muhat) and log(Vhat). This decision was based on the distribution of both targets – muhat was moderately skewed and contained some large values, while Vhat was highly left-skewed with extreme outliers. Applying a log transformation to both targets helped stabilize their variances and made their distributions closer to normal. This transformation improved model performance and made the residuals more homoscedastic, which benefits many regression methods.

Random Forest (RF)

Random Forest is an ensemble method that creates a large number of decision trees and aggregates their predictions, in this instance 500 trees were used. This approach reduces the risk of overfitting that individual trees may exhibit. The key tuning parameter for Random Forest is mtry, which determines how many predictor variables are randomly sampled at each split. Since the input space in this task consists of only two predictors (X_1 and X_2), I tested mtry values of 1 and 2. This allowed the model to consider both limited and full information at each decision point without unnecessary complexity.

Gradient Boosting Machines (GBM)

GBM is another tree-based ensemble method that builds trees sequentially, with each new tree correcting the errors of the ones before it. This makes GBM highly flexible and powerful in capturing complex patterns. Several hyperparameters were tuned for this model:

- n.trees (the total number of trees)
- interaction.depth (the maximum depth of each tree)
- shrinkage (the learning rate controlling the influence of each tree)
  The tuning grid was designed to provide a good balance between model flexibility and training efficiency. n.minobsinnode (minimum number of observations in the

terminal nodes) was set to the default number of 10 for computational performance reasons.

Support Vector Machines (SVM)
Support Vector Machines with a radial basis function (RBF) kernel were included to model potential nonlinear relationships. SVMs work by transforming the input space and finding an optimal separating boundary. I tuned two key parameters:

- sigma, which controls the spread of the RBF kernel
- C, which governs the trade-off between fitting the training data and maintaining generalizability
  Testing multiple values of C allowed me to adjust the model's tolerance for error and control overfitting.

Neural Networks (NNET)
Neural networks provide a flexible, layered approach to learning relationships between variables. For this project, I used single-layer feedforward neural networks and tuned the number of hidden units (size) and the regularization parameter (decay) that penalizes large weights to avoid overfitting. The models were configured for regression by setting linout = TRUE, and cross-validation was used to find the optimal architecture and regularization strength.

By using a range of modeling techniques – from tree-based ensembles to kernel methods and neural networks – I was able to compare both linear and highly flexible nonlinear approaches. Cross-validation ensured that the performance estimates were robust, and it helped guide the tuning of model parameters for reliable out-of-sample predictions. The final models were then applied to a separate test dataset, and for the predictions, results were exponentiated to return them to their original scale.

Results

Below are the results of each model's performance and best tuning parameters on muhat and Vhat. The RMSE highlighted in yellow for each outcome indicates the best model, and the one highlighted in red the worst:

Outcome	Model	Tuned Parameters	Best Tuning Values	Lowest RMSE	Associated R^2 with Lowest RMSE
Log(muhat)	RF	mtry	mtry = 2	0.02873191	0.9943608
Log(muhat)	GBM	n.trees, interaction.depth, shrinkage	n.trees = 500, interaction.depth = 3, shrinkage = 0.1	0.02728357	0.9949258

Log(muhat)	SVM	sigma, C	sigma = 0.01, C = 10	0.06659318	0.9715093
Log(muhat)	NNET	size, decay	size = 4, decay = 0.1	0.04142411	0.9882794
Log(Vhat)	RF	mtry	mtry = 1	0.1178528	0.958978
Log(Vhat)	GBM	n.trees, interaction.depth, shrinkage	n.trees = 500, interaction.depth = 3, shrinkage = 0.05	0.1118634	0.9626794
Log(Vhat)	SVM	sigma, C	sigma = 0.01, C = 10	0.2541374	0.8245969
Log(Vhat)	NNET	size, decay	size = 4, decay = 0.1	0.1400666	0.9419971

Random Forest (RF)

The Random Forest (RF) model performed well on both outcomes, particularly in terms of stability and ease of tuning. For the prediction of log(muhat), RF achieved a low RMSE of 0.0287 with an associated R^2 of 0.9944, indicating a strong fit. Similarly, for log(Vhat), the RF model yielded an RMSE of 0.1179 and an R^2 of 0.9590. These results demonstrate RF's capacity to model the relationships in the data effectively.

The tuning process for RF was relatively simple, involving only the mtry parameter – the number of predictors considered at each split. The best value was mtry = 2 for log(muhat) and mtry = 1 for log(Vhat). Although RF was not ultimately the best-performing model, its competitive results and straightforward tuning make it a strong candidate in many applied contexts. However, while RF was solid, it was outperformed by Gradient Boosting Machines in both accuracy and explained variance.

Gradient Boosting Machine (GBM)

Gradient Boosting Machines (GBM) emerged as the best-performing model across both outcomes, making it the final chosen method. For log(muhat), GBM achieved the lowest RMSE of 0.0273 and the highest R^2 of 0.9949, while for log(Vhat), it recorded an RMSE of 0.1119 and an R^2 of 0.9627 – the best scores across all models tested. These results underscore GBM's superior predictive power and its ability to capture complex patterns in the data.

The tuning process for GBM involved optimizing three hyperparameters: the number of trees (n.trees), tree depth (interaction.depth), and learning rate (shrinkage). For log(muhat), the optimal combination was 500 trees, depth of 3, and a learning rate of 0.1. For log(Vhat), the same number of trees and depth were used, with a slightly lower

shrinkage of 0.05. The consistency in optimal tuning across the two tasks adds to GBM's appeal, as it suggests generalizable performance.

Support Vector Machine (SVM)
The Support Vector Machine (SVM) model underperformed relative to the other models in this study. For log(muhat), the SVM achieved an RMSE of 0.0666 and an R^2 of 0.9715, while for log(Vhat), it yielded the highest RMSE of 0.2541 and the lowest R^2 of 0.8246 among all models. These results indicate that SVM struggled to capture the underlying structure of the data, particularly for variance estimation.

Despite these limitations, the tuning process for SVM was carefully managed. The model was optimized using the sigma and C parameters, with the best results obtained at sigma = 0.01 and C = 10. However, even with tuning, SVM failed to deliver competitive accuracy.

Neural Network (NNET)
The Neural Network (NNET) model delivered strong but not best-in-class results. For log(muhat), NNET recorded an RMSE of 0.0414 and an R^2 of 0.9883, while for log(Vhat) it achieved an RMSE of 0.1401 and an R^2 of 0.9420. These values are better than those of SVM but still fall short of the performance achieved by GBM and RF.

The tuning of NNET involved adjusting the number of hidden units (size) and the weight decay (decay). The best performance was obtained with size = 4 and decay = 0.1 for both targets. While the neural network showed flexibility and reasonable predictive power, it is also known for sensitivity to initialization and overfitting.

Findings/Conclusion

The evaluation of four machine learning models – Random Forest (RF), Gradient Boosting Machine (GBM), Support Vector Machine (SVM), and Neural Network (NNET) – revealed distinct strengths and weaknesses, shaping the final choice of GBM as the optimal model for predicting log-transformed outcomes.

Gradient Boosting Machine (GBM) emerged as the most effective model, delivering the best performance in terms of both predictive accuracy and explained variance. Its ability to consistently achieve low RMSE and high R^2 values across both outcomes (log(muhat) and log(Vhat)) showcases its ability to model complex relationships in the data. The flexibility in tuning hyperparameters further contributes to its robustness, providing a clear advantage over the other models.

Random Forest (RF), while slightly behind GBM in predictive accuracy, demonstrated strong performance and ease of use. Its competitive results, particularly in predicting log(muhat), and its straightforward tuning process (via mtry) make it a reliable alternative for scenarios where model simplicity and stability are prioritized.

Neural Network (NNET), though performing reasonably well, did not match the predictive power of GBM or RF. While it exhibited flexibility in handling data, its susceptibility to overfitting and the challenge of tuning in the face of high-dimensional data are limitations that hinder its overall performance. This suggests that while neural networks can be powerful, their sensitivity may require additional consideration in future modeling efforts.

Support Vector Machine (SVM), on the other hand, struggled to capture the underlying data structure, particularly for variance prediction (log(Vhat)). Despite careful tuning, it performed poorly in comparison to the other models, suggesting that SVM may not be the best choice for this dataset.

In conclusion, GBM stands out as the most robust and effective model, offering the best balance of predictive power and tuning flexibility. While RF remains a solid choice for simpler implementations, NNET and SVM underperformed in this study. Future work may focus on refining the tuning processes for these models or exploring hybrid approaches to improve performance further.

Appendix

```
# Load required libraries
library(caret)
library(randomForest)
library(gbm)
library(e1071)
library(nnet)

set.seed(42)

##### (A) Load and Prepare Training Data
traindata <- read.table(file = "7406train.csv", sep = ",")
X1 <- traindata[, 1]
X2 <- traindata[, 2]
muhat <- apply(traindata[, 3:202], 1, mean)
Vhat  <- apply(traindata[, 3:202], 1, var)

# Apply log transformation to muhat and Vhat
log_muhat <- log(muhat)
log_Vhat  <- log(Vhat)

# Add log-transformed columns for training
data0 <- data.frame(X1 = X1, X2 = X2, muhat = muhat, Vhat = Vhat, log_muhat = log_muhat,
log_Vhat = log_Vhat)

## we can plot 4 graphs in a single plot
par(mfrow = c(2, 2))
plot(X1, muhat)
plot(X2, muhat)
plot(X1, Vhat)
plot(X2, Vhat)

hist(data0$muhat)
boxplot(data0$muhat, main="Boxplot of Muhat", ylab="Values")
summary(data0$muhat)
hist(data0$Vhat)
summary(data0$Vhat)
boxplot(data0$Vhat, main="Boxplot of Vhat", ylab="Values")

##### (B) Cross-Validation Setup
cv_control <- trainControl(method = "cv", number = 10)

# Define tuning grids
```

```
rf_grid  <- expand.grid(mtry = c(1, 2))
svm_grid <- expand.grid(sigma = 0.01, C = c(1, 10))
gbm_grid <- expand.grid(
  n.trees = c(50, 100, 500),
  interaction.depth = c(1, 3),
  shrinkage = c(0.05, 0.1),
  n.minobsinnode = 10
)
nnet_grid <- expand.grid(size = c(2, 4), decay = c(0.1, 0.5))

##### (C) Train models for log(muhat)
models_logmu <- list()

models_logmu[["RF"]] <- train(
  log_muhat ~ X1 + X2, data = data0, method = "rf",
  trControl = cv_control, tuneGrid = rf_grid, ntree = 500
)

models_logmu[["GBM"]] <- train(
  log_muhat ~ X1 + X2, data = data0, method = "gbm",
  trControl = cv_control, tuneGrid = gbm_grid, verbose = FALSE
)

models_logmu[["SVM"]] <- train(
  log_muhat ~ X1 + X2, data = data0, method = "svmRadial",
  trControl = cv_control, tuneGrid = svm_grid
)

models_logmu[["NNET"]] <- train(
  log_muhat ~ X1 + X2, data = data0, method = "nnet",
  trControl = cv_control, tuneGrid = nnet_grid,
  linout = TRUE, trace = FALSE
)

# Select best log(muhat) model
results_logmu <- sapply(models_logmu, function(mod) min(mod$results$RMSE^2))
best_model_logmu <- names(which.min(results_logmu))
cat("Best model for log(muhat):", best_model_logmu, "\n")
models_logmu[[best_model_logmu]]$bestTune


##### (D) Train models for log(Vhat)
models_logvar <- list()
```

```
models_logvar[["RF"]] <- train(
  log_Vhat ~ X1 + X2, data = data0, method = "rf",
  trControl = cv_control, tuneGrid = rf_grid, ntree = 500
)

models_logvar[["GBM"]] <- train(
  log_Vhat ~ X1 + X2, data = data0, method = "gbm",
  trControl = cv_control, tuneGrid = gbm_grid, verbose = FALSE
)

models_logvar[["SVM"]] <- train(
  log_Vhat ~ X1 + X2, data = data0, method = "svmRadial",
  trControl = cv_control, tuneGrid = svm_grid
)

models_logvar[["NNET"]] <- train(
  log_Vhat ~ X1 + X2, data = data0, method = "nnet",
  trControl = cv_control, tuneGrid = nnet_grid,
  linout = TRUE, trace = FALSE
)

# Select best log(Vhat) model
results_logvar <- sapply(models_logvar, function(mod) min(mod$results$RMSE^2))
best_model_logvar <- names(which.min(results_logvar))
cat("Best model for log(Vhat):", best_model_logvar, "\n")
models_logvar[[best_model_logvar]]$bestTune


##### (E) Load Test Data and Predict
testX <- read.table(file = "7406test.csv", sep = ",")
colnames(testX) <- c("X1", "X2")

# Predict log(muhat)
logmu_pred <- predict(models_logmu[[best_model_logmu]], newdata = testX)
mu_pred <- exp(logmu_pred)  # Revert to original scale

# Predict log(Vhat) and revert to original scale
log_var_pred <- predict(models_logvar[[best_model_logvar]], newdata = testX)
var_pred <- exp(log_var_pred)

cat("Results for models predicting log(muhat):\n")
for (model_name in names(models_logmu)) {
  cat("\n---", model_name, "---\n")
  print(models_logmu[[model_name]]$results)
```

```
  cat("Best tuning parameters:\n")
  print(models_logmu[[model_name]]$bestTune)
  cat("Lowest RMSE:\n")
  print(min(models_logmu[[model_name]]$results$RMSE))
}
logmu_summary <- sapply(models_logmu, function(m) min(m$results$RMSE))
print(logmu_summary)

cat("\n\nResults for models predicting log(Vhat):\n")
for (model_name in names(models_logvar)) {
  cat("\n---", model_name, "---\n")
  print(models_logvar[[model_name]]$results)
  cat("Best tuning parameters:\n")
  print(models_logvar[[model_name]]$bestTune)
  cat("Lowest RMSE:\n")
  print(min(models_logvar[[model_name]]$results$RMSE))
}

logvar_summary <- sapply(models_logvar, function(m) min(m$results$RMSE))
print(logvar_summary)


##### (F) Finalize Submission Data and Write CSV
testdata <- data.frame(
  X1 = testX$X1,
  X2 = testX$X2,
  muhat = round(mu_pred, 6),
  Vhat  = round(var_pred, 6)
)



hist(testdata$muhat)
summary(testdata$muhat)
hist(testdata$Vhat)
summary(testdata$Vhat)


write.table(testdata, file="1.Malcolm.Olan.csv",
          sep=",",  col.names=F, row.names=F)
```