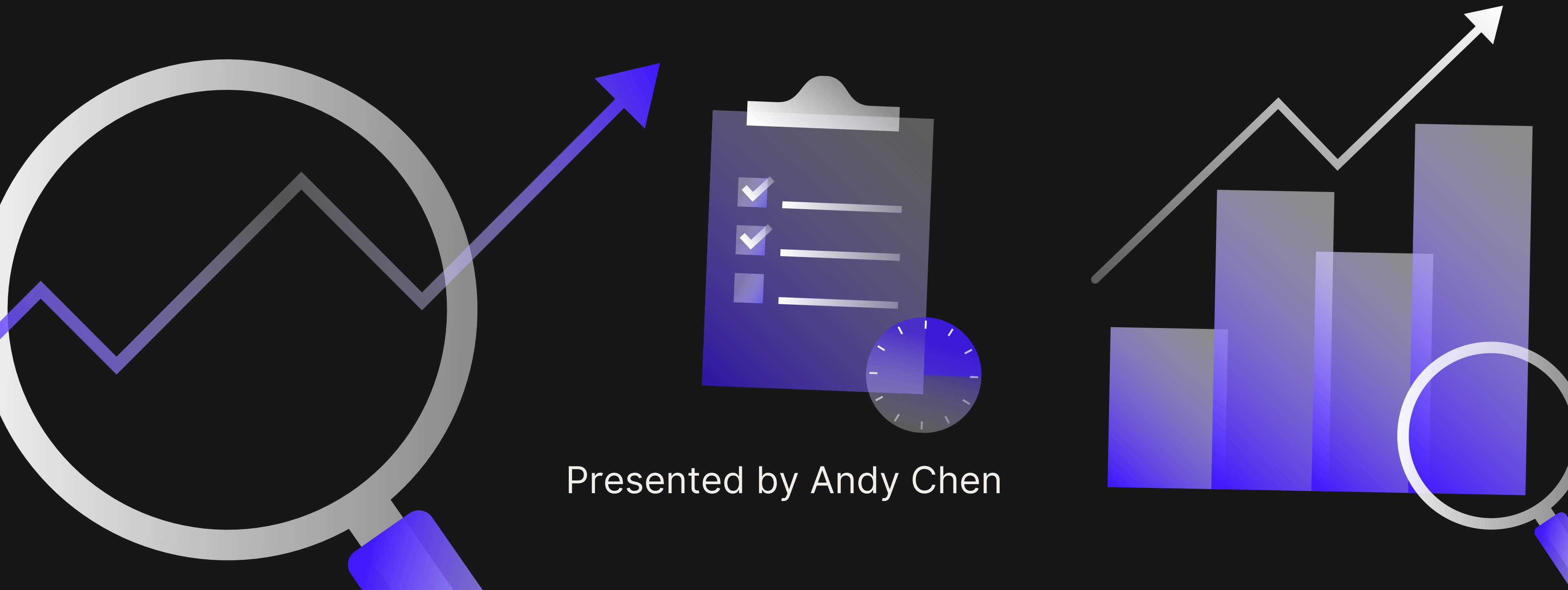
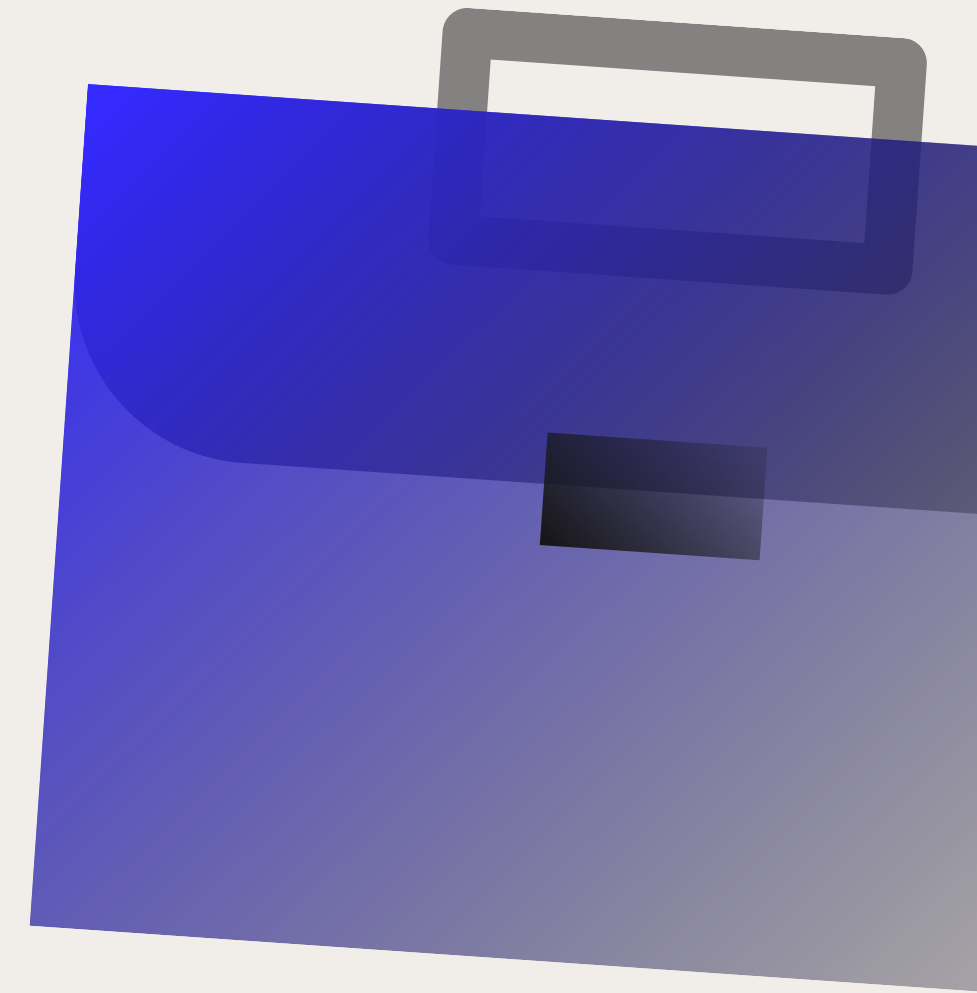


Stock Price Analysis Using GBM & OU Model



Presented by Andy Chen

Background and Objective of Stock Price Modeling



Stock price modeling helps understand and predict price behaviors in different market contexts. This project applies **Geometric Brownian Motion** (GBM) and **Ornstein-Uhlenbeck** (OU) processes to model stock prices of Intel and AMD, two companies in the same industry, and Netflix, a company from a different industry, for comparison. By contrasting similar and different sectors, this analysis highlights how industry-specific and broader market dynamics influence stock price movements.



Dataset

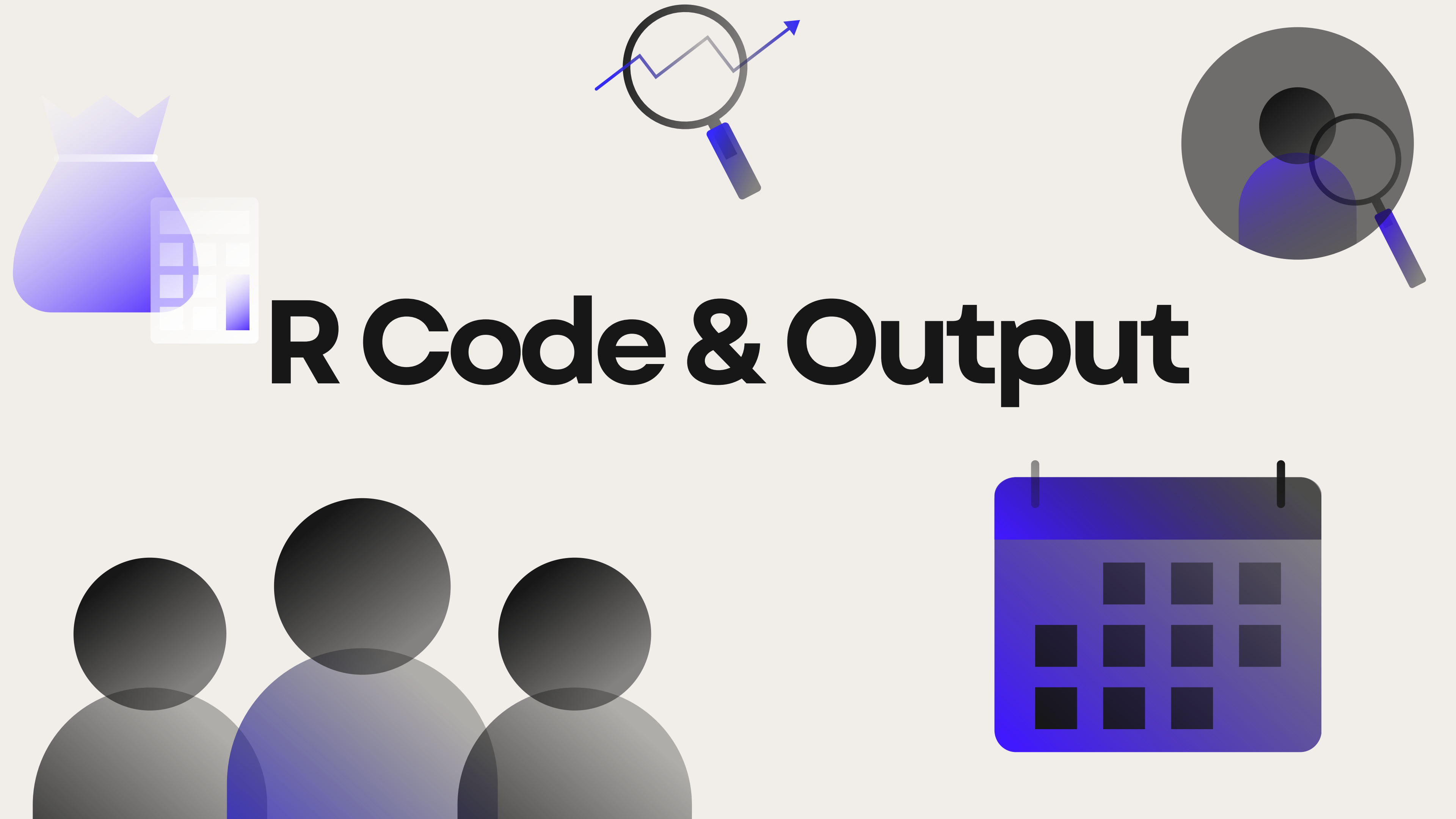


ticker	date	open	high	low	close
AMD	2024/8/12	134.44	137.99	133.23	136.77
AMD	2024/8/13	138.12	141.19	137.52	141.13
AMD	2024/8/14	143	143.25	137.64	140.75
AMD	2024/8/15	143	147.96	141.81	147.36
AMD	2024/8/16	145.7	149.369	145.2	148.56
AMD	2024/8/19	148.43	155.365	147.72	155.28
AMD	2024/8/20	156.49	162.04	154.51	156.4
AMD	2024/8/21	156.12	158.65	155.07	157.81
AMD	2024/8/22	158.87	159.84	151.01	151.7
AMD	2024/8/23	153.6	156.39	151.83	154.98
AMD	2024/8/26	154.7	158.28	148.91	150
AMD	2024/8/27	150.13	151.7	148.44	150.5

ticker	date	open	high	low	close
NFLX	2002/5/23	1.16	1.24	1.15	1.2
NFLX	2002/5/24	1.21	1.23	1.2	1.21
NFLX	2002/5/28	1.21	1.23	1.16	1.16
NFLX	2002/5/29	1.16	1.16	1.09	1.1
NFLX	2002/5/30	1.11	1.11	1.07	1.07
NFLX	2002/5/31	1.08	1.08	1.07	1.08
NFLX	2002/6/3	1.08	1.15	1.08	1.13
NFLX	2002/6/4	1.14	1.14	1.11	1.12
NFLX	2002/6/5	1.11	1.16	1.11	1.15
NFLX	2002/6/6	1.15	1.23	1.15	1.18
NFLX	2002/6/7	1.18	1.18	1.1	1.12
NFLX	2002/6/10	1.13	1.17	1.13	1.16
NFLX	2002/6/11	1.16	1.19	1.13	1.15

ticker	date	open	high	low	close
INTC	1972/1/7	0.013861	0.013861	0.013861	0.013861
INTC	1972/1/14	0.006887	0.006887	0.006887	0.006887
INTC	1972/1/21	0.006887	0.006887	0.006887	0.006887
INTC	1972/1/24	0.006887	0.006887	0.006887	0.006887
INTC	1972/1/25	0.006887	0.006887	0.006887	0.006887
INTC	1972/1/26	0.006887	0.006887	0.006887	0.006887
INTC	1972/1/27	0.013861	0.013861	0.013861	0.013861
INTC	1972/1/28	0.013861	0.013861	0.013861	0.013861
INTC	1972/1/31	0.013861	0.013861	0.013861	0.013861
INTC	1972/2/1	0.013861	0.013861	0.013861	0.013861
INTC	1972/2/2	0.013861	0.013861	0.013861	0.013861
INTC	1972/2/3	0.013861	0.013861	0.013861	0.013861
INTC	1972/2/4	0.013861	0.013861	0.013861	0.013861
INTC	1972/2/7	0.013861	0.013861	0.013861	0.013861





R Code & Output

Correlation

```
# Load necessary libraries
library(dplyr)
library(ggplot2)

# Step 1: Load and Align Data by Date
intc_data <- read.csv("intc.csv")
amd_data <- read.csv("amd.csv")
nflx_data <- read.csv("nflx.csv")

# Convert dates to Date format
intc_data$date <- as.Date(intc_data$date, format = "%Y-%m-%d")
amd_data$date <- as.Date(amd_data$date, format = "%Y-%m-%d")
nflx_data$date <- as.Date(nflx_data$date, format = "%Y-%m-%d")

# Merge datasets by date (inner join to keep common dates)
merged_data <- merge(intc_data, amd_data, by = "date", suffixes = c("_intc", "_amd"))
merged_data <- merge(merged_data, nflx_data, by = "date")

# Ensure prices are numeric
merged_data <- merged_data[, c("date", "close_intc", "close_amd", "close")]
names(merged_data) <- c("date", "Intel", "AMD", "Netflix")

# Step 2: Compute Log>Returns
log_returns <- merged_data %>%
  mutate(
    Intel = log(Intel / lag(Intel)),
    AMD = log(AMD / lag(AMD)),
    Netflix = log(Netflix / lag(Netflix))
  ) %>%
  na.omit() # Remove rows with NA values

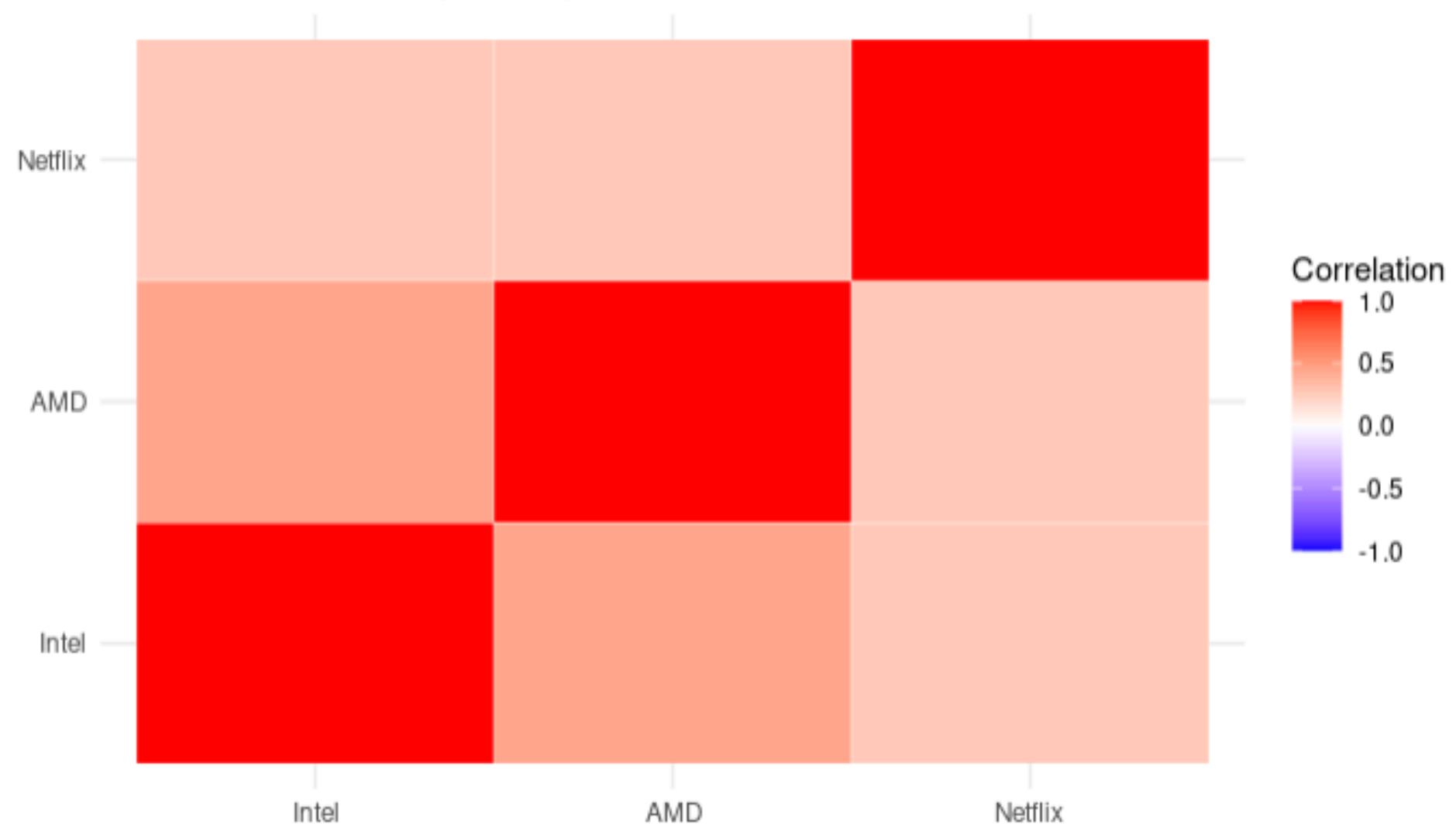
# Step 3: Compute Correlation Coefficients
log_returns_df <- log_returns[, c("Intel", "AMD", "Netflix")]
correlation_matrix <- cor(log_returns_df)

# Print the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)
```

[1] "Correlation Matrix:"

	Intel	AMD	Netflix
Intel	1.0000000	0.4572235	0.2698502
AMD	0.4572235	1.0000000	0.2668896
Netflix	0.2698502	0.2668896	1.0000000

Correlation Heatmap of Log>Returns



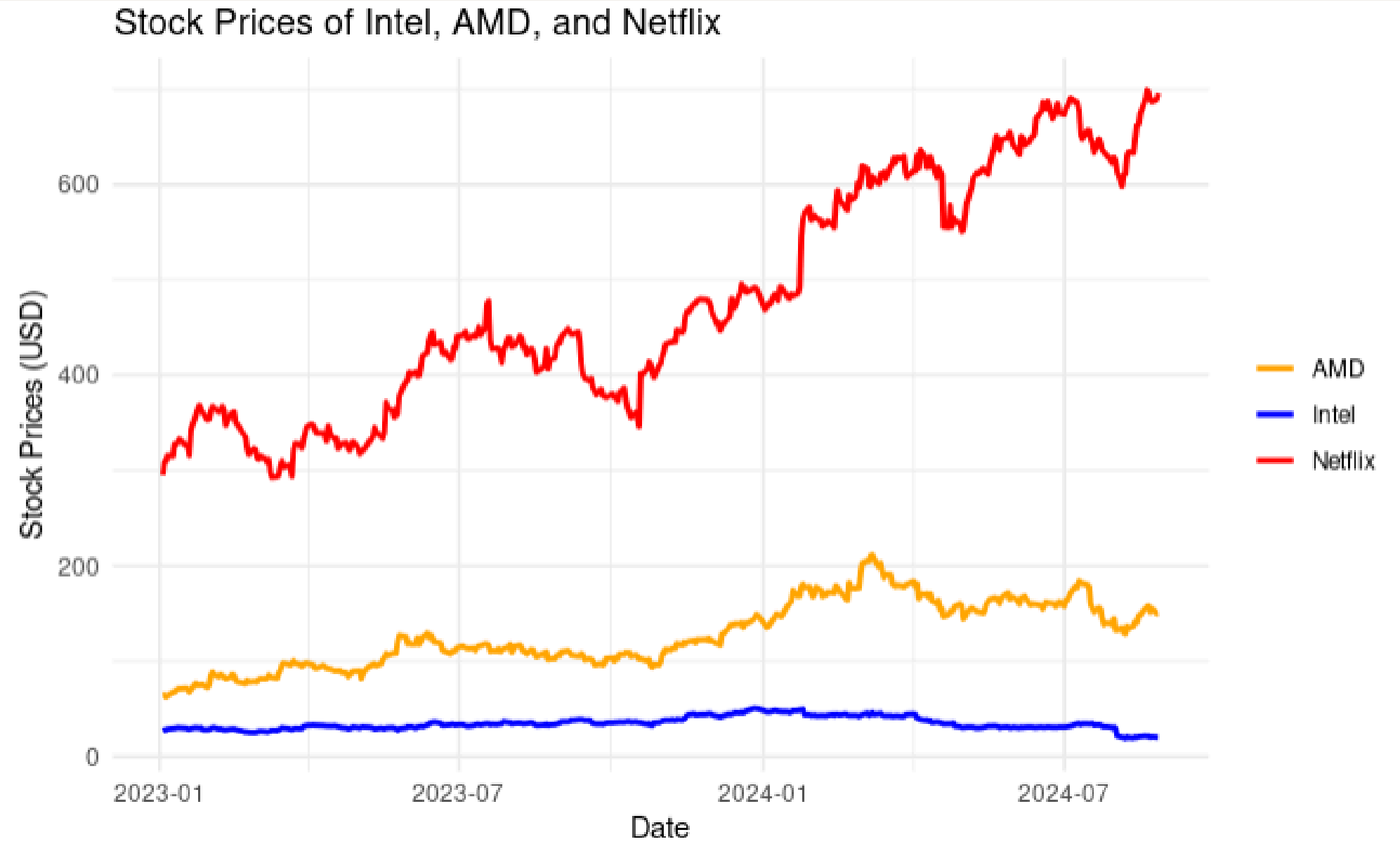
```
# Visualize Correlation Matrix as Heatmap
correlation_heatmap <- as.data.frame(as.table(correlation_matrix))
colnames(correlation_heatmap) <- c("Company1", "Company2", "Correlation")

ggplot(correlation_heatmap, aes(x = Company1, y = Company2, fill = Correlation)) +
  geom_tile(color = "white") +
  scale_fill_gradient2(low = "blue", high = "red", mid = "white", midpoint = 0, limit = c(-1, 1), space = "Lab") +
  labs(title = "Correlation Heatmap of Log>Returns", x = "", y = "") +
  theme_minimal()
```

Stock Price Visualization: Intel, AMD, and Netflix 2023 ~2024

```
# Filter the data to include only dates from 2023
merged_data_2023 <- merged_data %>%
  filter(date >= as.Date("2023-01-01"))

# Step 3: Plot the stock prices of Intel, AMD, and Netflix from 2023
ggplot(merged_data_2023, aes(x = date)) +
  geom_line(aes(y = Intel, color = "Intel"), size = 1) +
  geom_line(aes(y = AMD, color = "AMD"), size = 1) +
  geom_line(aes(y = Netflix, color = "Netflix"), size = 1) +
  labs(title = "Stock Prices of Intel, AMD, and Netflix",
       x = "Date",
       y = "Stock Prices (USD)") +
  scale_color_manual(values = c("Intel" = "blue", "AMD" = "orange", "Netflix" = "red")) +
  theme_minimal() +
  theme(legend.title = element_blank())
```



Parameter Estimation for Stock Price Modeling

```
# Intel
# Step: Compute increments of log-prices for Intel
compute_log_increments <- function(data) {
  log_inc <- log(data$close[-1] / head(data$close, -1))
  return(log_inc)
}

log_inc_intel <- compute_log_increments(intc_data)

# Step 3: Estimate parameters (mu and sigma) for Intel
compute_parameters <- function(log_inc) {
  mu_hat <- mean(log_inc) # Mean of log-increments
  sigma_hat <- sd(log_inc) # Standard deviation of log-increments
  return(list(mu_hat = mu_hat, sigma_hat = sigma_hat))
}

params_intel <- compute_parameters(log_inc_intel)

# Step 4: Print Intel results
print("Intel Parameters:")
print(paste("Mu (Mean):", params_intel$mu_hat))
print(paste("Sigma (Std Dev):", params_intel$sigma_hat))
```

```
[1] "Intel Parameters:"
[1] "Mu (Mean): 0.000548695891176054"
[1] "Sigma (Std Dev): 0.0338975448287733"
```

```
# Step: Compute increments of log-prices for AMD
log_inc_amd <- compute_log_increments(amd_data)

# Step 3: Estimate parameters (mu and sigma) for AMD
params_amd <- compute_parameters(log_inc_amd)

# Step 4: Print AMD results
print("AMD Parameters:")
print(paste("Mu (Mean):", params_amd$mu_hat))
print(paste("Sigma (Std Dev):", params_amd$sigma_hat))
```

```
[1] "AMD Parameters:"
[1] "Mu (Mean): 0.000345074576648941"
[1] "Sigma (Std Dev): 0.0373636266111063"
```

```
[1] "Netflix Parameters:"
[1] "Mu (Mean): 0.00113557482130037"
[1] "Sigma (Std Dev): 0.0355196030727573"
```

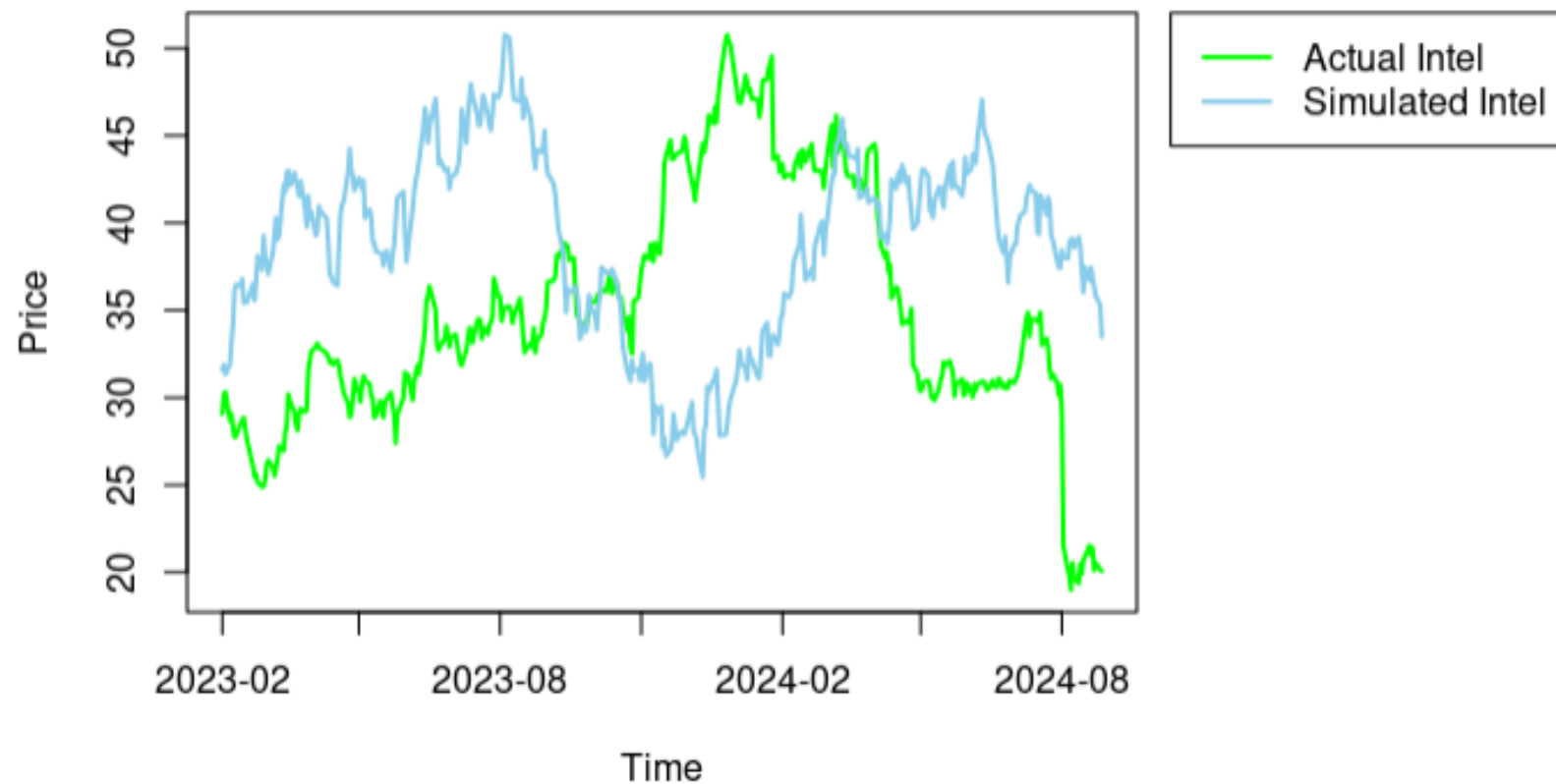
```
# Step: Compute increments of log-prices for Netflix
log_inc_nflx <- compute_log_increments(nflx_data)

# Step 3: Estimate parameters (mu and sigma) for Netflix
params_nflx <- compute_parameters(log_inc_nflx)

# Step 4: Print Netflix results
print("Netflix Parameters:")
print(paste("Mu (Mean):", params_nflx$mu_hat))
print(paste("Sigma (Std Dev):", params_nflx$sigma_hat))
```

Simulating Ornstein-Uhlenbeck (OU) Process for Intel

Intel: Actual vs Simulated



```
# Intel
# Filter data to start
intc_data <- subset(intc_data, date >= as.Date("2023-02-01"))

# Step 2: Estimate OU Parameters for Intel
intc_inc <- diff(intc_data$close) # Increments for Intel
intc_fit <- glm(intc_inc ~ head(intc_data$close, -1))
theta_hat_intc <- -intc_fit$coefficients[2]
mu_hat_intc <- intc_fit$coefficients[1] / theta_hat_intc
sigma_hat_intc <- sigma(intc_fit)

# Step 3: Simulate OU Process for Intel
set.seed(5536667)
OU_intc <- numeric(nrow(intc_data))
OU_intc[1] <- intc_data$close[1]
for (i in 2:nrow(intc_data)) {
  OU_intc[i] <- OU_intc[i - 1] + theta_hat_intc * (mu_hat_intc - OU_intc[i - 1]) +
    sigma_hat_intc * rnorm(1)
}

# Step 4: Scale the simulated Intel prices so the maximum is within the actual price range
max_intc_actual <- max(intc_data$close)
max_intc_simulated <- max(OU_intc)
scaling_factor_intc <- max_intc_actual / max_intc_simulated
OU_intc_scaled <- OU_intc * scaling_factor_intc

# Step 5: Plot Intel Data
par(mar = c(5, 5, 4, 10)) # Add space on the right for legend
plot(as.Date(intc_data$date), intc_data$close, type = "l", lty = 1, lwd = 2,
     col = "green", xlab = "Time", ylab = "Price", main = "Intel: Actual vs Simulated", xaxt = "n")
lines(as.Date(intc_data$date), OU_intc_scaled, lwd = 2, col = "skyblue")

axis.Date(1, at = seq(min(intc_data$date), max(intc_data$date), by = "3 months"),
         format = "%Y-%m")

# Add legend outside on the right
legend("topright", inset = c(-0.45, 0), xpd = TRUE, # Move legend outside
      legend = c("Actual Intel", "Simulated Intel"),
      lty = 1, lwd = 2, col = c("green", "skyblue"))
```

Simulating Ornstein-Uhlenbeck (OU) Process for AMD

```
# AMD
# Filter data to start
amd_data <- subset(amd_data, date >= as.Date("2023-02-01"))

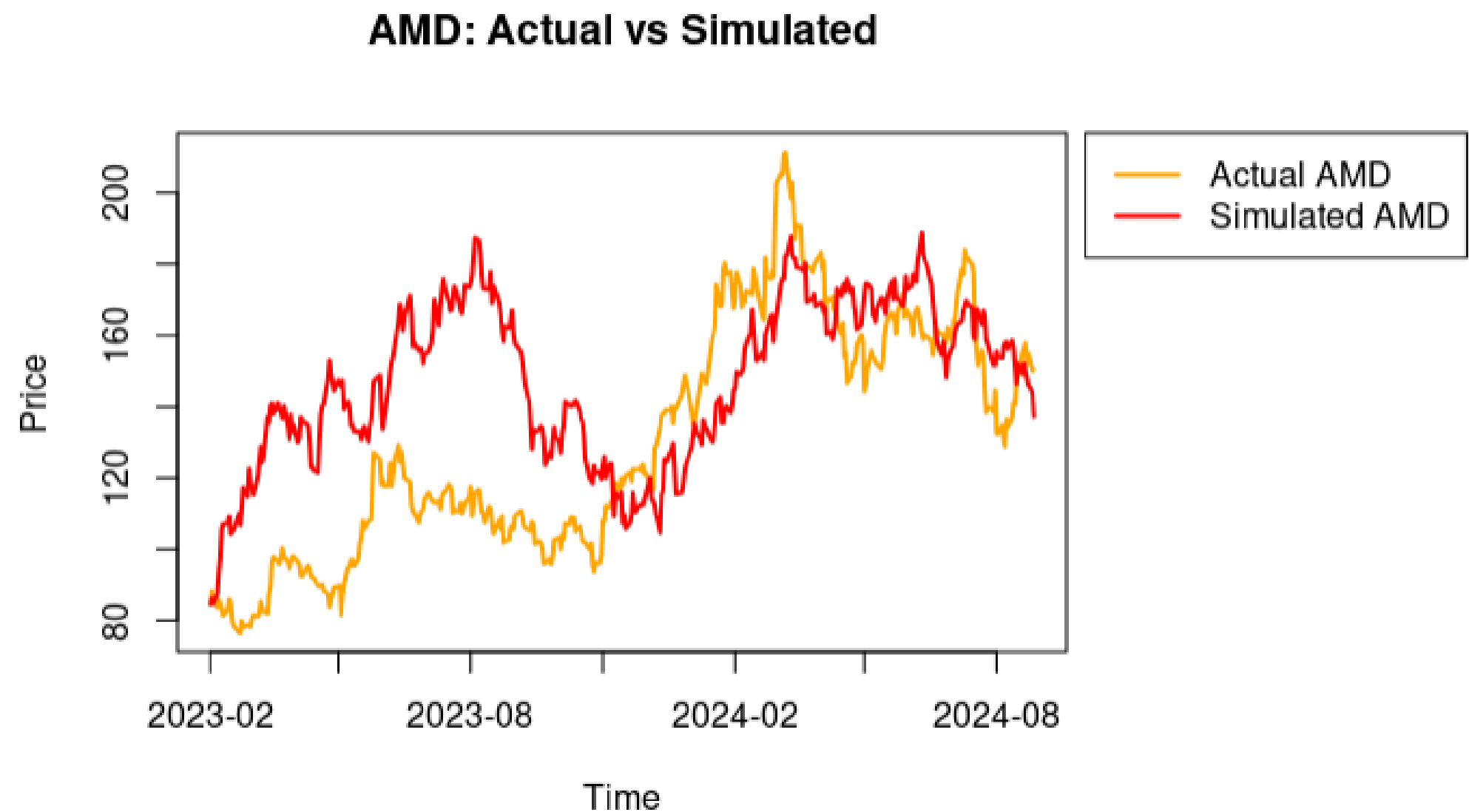
# Step 6: Estimate OU Parameters for AMD
amd_inc <- diff(amd_data$close) # Increments for AMD
amd_fit <- glm(amd_inc ~ head(amd_data$close, -1))
theta_hat_amd <- -amd_fit$coefficients[2]
mu_hat_amd <- amd_fit$coefficients[1] / theta_hat_amd
sigma_hat_amd <- sigma(amd_fit)

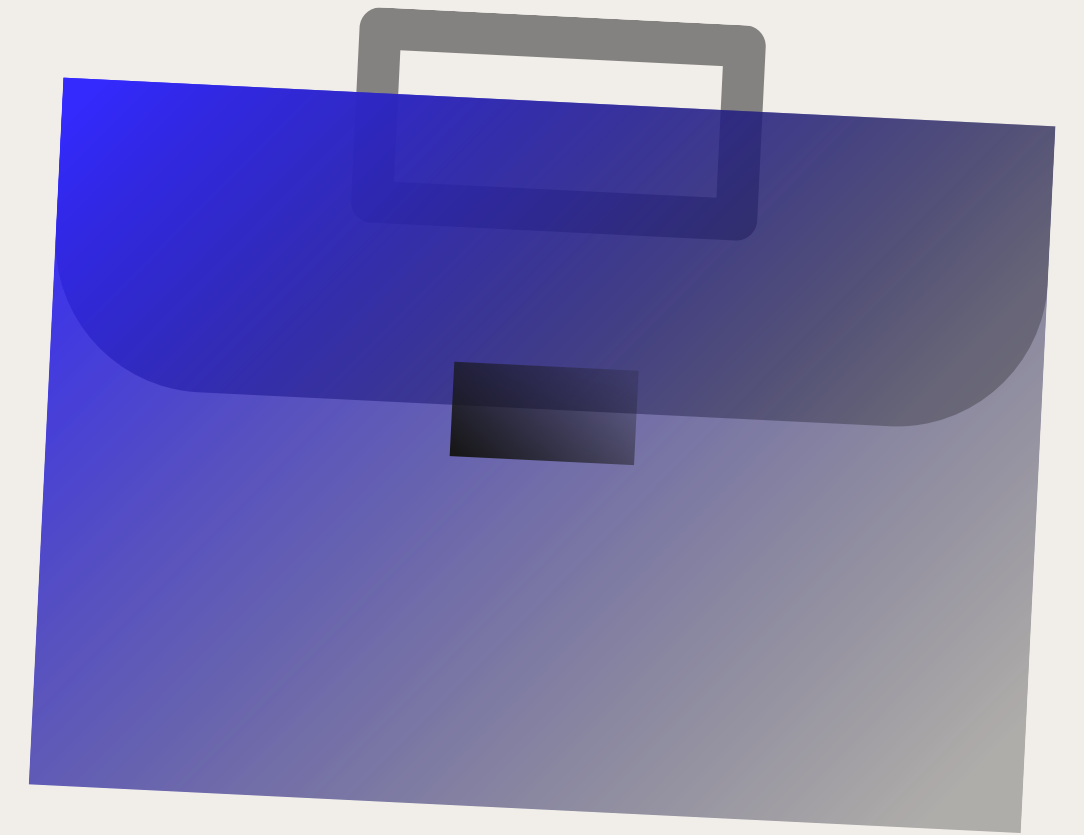
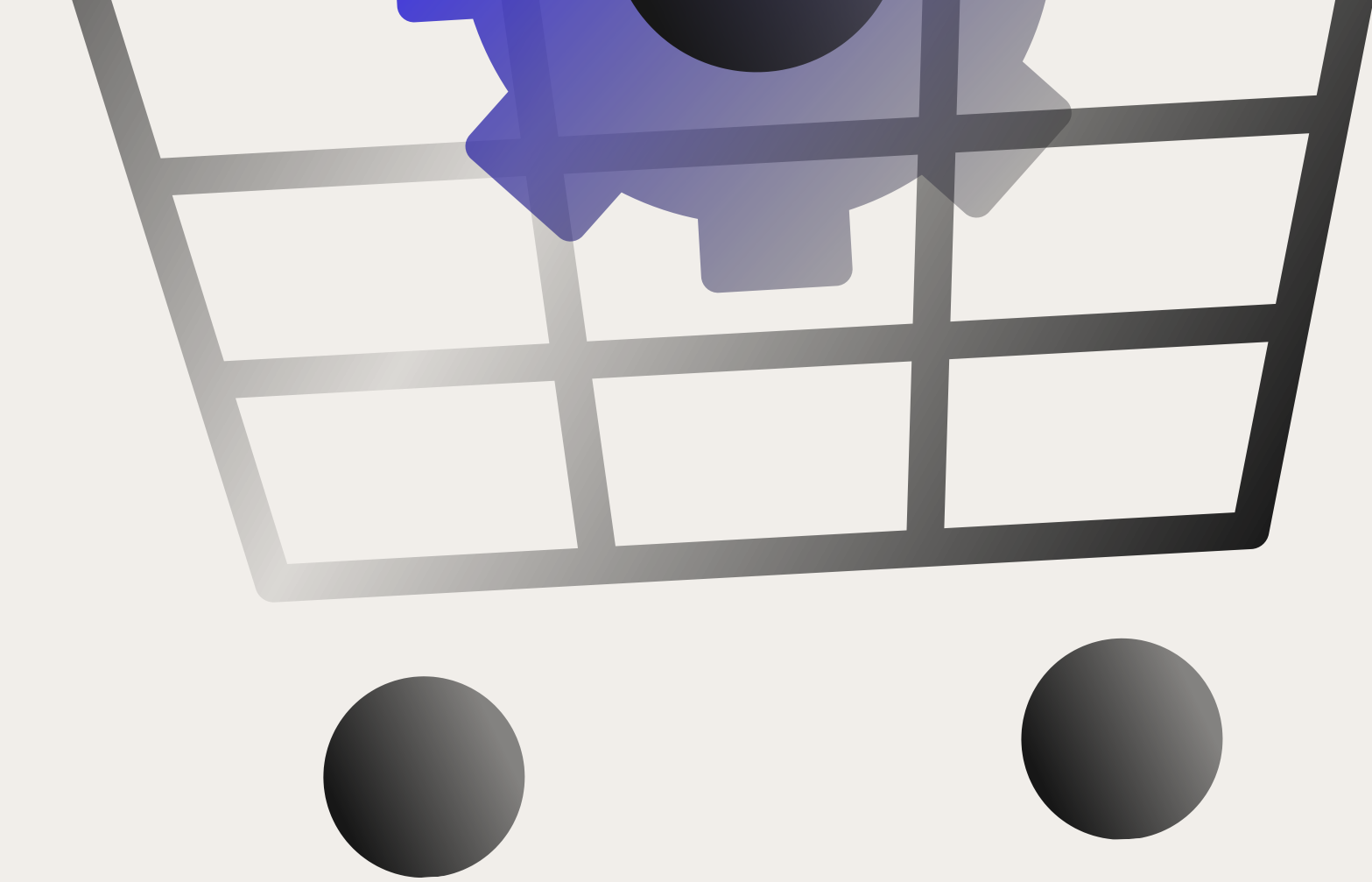
# Step 7: Simulate OU Process for AMD
set.seed(5536667)
OU_amd <- numeric(nrow(amd_data))
OU_amd[1] <- amd_data$close[1]
for (i in 2:nrow(amd_data)) {
  OU_amd[i] <- OU_amd[i - 1] + theta_hat_amd * (mu_hat_amd - OU_amd[i - 1]) +
    sigma_hat_amd * rnorm(1)
}

# Step 8: Plot AMD Data
par(mar = c(5, 5, 5, 10)) # Add space on the right for legend
plot(as.Date(amd_data$date), amd_data$close, type = "l", lty = 1, lwd = 2,
     col = "orange", xlab = "Time", ylab = "Price", main = "AMD: Actual vs Simula",
     lines(as.Date(amd_data$date), OU_amd, lwd = 2, col = "red"))

axis.Date(1, at = seq(min(amd_data$date), max(amd_data$date), by = "3 months"),
         format = "%Y-%m")

# Add legend outside on the right
legend("topright", inset = c(-0.45, 0), xpd = TRUE, # Move legend outside
      legend = c("Actual AMD", "Simulated AMD"),
      lty = 1, lwd = 2, col = c("orange", "red"))
```





Thank You