Aaron Stange & Lorie Ninganza Due: 4/29/2024

Professor Khan

Stock Simulation Final Report

Introduction:

In this project, our original goal was to gain a deeper understanding of the factors that influence stock prices. Then, use that knowledge to develop a simulation in R that simulates/predicts the stock market through statistical analysis and techniques. Through the development and creation of our simulation, we hoped to both gain insight into the stock market and further our knowledge of simulation in R.

By analyzing the impact factors such as company performance, industry trends, and interest rate, we want to understand how these factors can influence shaping the a stock's price over time. A crucial aspect of our project is identifying and understanding the variables that affect stock prices, like volatility, drift, and interest rates. In order to accurately simulate the market we would have to develop a deep understanding of how each of these variables affects a stock's price. We would then have to both extract/calculate their values from historical data and recreate their effect through statistical techniques to predict future prices. To this end, we'd have to learn how to use packages and datasets to make our simulation realistic and be able to predict prices with a degree of accuracy. Finally, we would have to learn how to then use our simulated stock price data as a resource to make investment decisions.

Model and Simulation:

Our model has two primary components/functions, one goes back in time "n" trading days and simulates the prices of a specified stock throughout said days, comparing the simulated prices to the actual prices. The other function starts from the current date and predicts the stock for "n" days in the future. The parameters for both functions are; symbol (the symbol of the stock that they want to simulate ex. Amazon = "AMZN"), days (the number of trading days the simulation will run for, defaults to 30), paths (the number of times the stock will be simulated, defaults to 10), and interest_rate (allows the user to input a hypothetical market interest rate, defaults to last recorded market interest rate).

The base for both of our models uses a geometric Brownian motion function for predicting and simulating stock prices based on the drift and volatility of the stock. To get the data required, our model downloads historical data of a given stock. It then uses the historical data to find the daily returns and the initial price for the stock at the start of the simulation period, based on the daily closing prices. Next, it uses the returns to calculate the drift and volatility of the stock. The drift, volatility, and initial price are then run through the geometric Brownian motion function to get the predicted stock prices.

In total, our model has four functions and uses one package. The package we use is called "quantmod", and it is what allows us to download historical data of both stocks and interest rates, along with helping perform some of the analysis with its dailyReturns() and volatility() methods. The first two functions our model uses are the components mentioned earlier, Stock.Simulator() which compares our predicted stock prices to the real prices for "n" trading days in the past to the present and Future.Stock.Simulator() which simulates future stock prices for “n” trading days in the future. Both these functions download and extract the needed data but then use the other two functions, gbm.f() and Simulated.Prices(), to actually predict/simulate the stock.

The gbm.f() function, takes an initial price, drift and volatility of a stock, and using geometric Brownian motion, it generates predicted stock prices for “n” trading days from the initial price. The second function Simulated.Prices() creates and returns a matrix, with each column of the matrix holding a separate realization for the stock prices over the “n” trading days. It takes a “paths” parameter which determines how many columns or different realizations for the stock it will simulate. It then loops through code “paths” times using the gbm.f() function to generate a single path/realization for the stock and finally appends said path to the matrix

Features and Assumptions:

Some other feature that our model has that makes it stand out from other models are how it calculates the volatility, the incorporation of interest rates on drift, and the random chance that volatility will double. In the process of making and testing our model, we noticed that when compared to the path the actual stocks were taking, our simulated stock paths appeared to have a lower volatility. At the time we were calculating the volatility as the standard deviation of the returns. We decided to do some research and experiment with different ways of calculating the volatility, during which we found the built-in volatility function in “quantmod”. The problem with this function is that after testing it extensively it kept returning a value that appeared to be larger than the actual volatility. Seeing this we decided to find the mean or average of these two values and use that as the volatility, and though this solution seemed quite simple, it worked surprisingly well and when tested returned the best results.

Another feature our model has is its incorporation of interest rates. Our model downloads historical data of the interest rates leading up to the starting date of the simulation and compares the average interest rate over the entire period leading up to the start date to the interest rate on the start date. This difference will represent how the interest rates are currently changing and we then use said difference to adjust the drift of our stock accordingly. If the interest rates are currently dropping then the difference will be positive, if the interest rates are rising the difference will be negative. To use this number to adjust the drift we divide it by a thousand to mediate its impact as the drift is already a very small number, and then add it to the drift. If the user enters a custom interest rate then the model will use that interest rate at the current or latest interest rate.

The final feature of our model is the random one-in-a-thousand-chance that the volatility will double for a realization of the stock. This feature is implemented inside of the Simulated.Prices() function right before a realization/path of stock is created there is a 0.1% chance that the volatility double for that path. This simulates the true randomness of the stock market as an event could occur at any time that drastically affects either a particular stock or the entire stock market. Events like natural disasters, war, recession, or good events like breakthroughs, all of these events can have a drastic impact on stock prices over a short period of time. With this in consideration and the fact that the stock market tends to be a very volatile and hard to predict place, having a small chance of the stock volatility going up is more realistic than not.

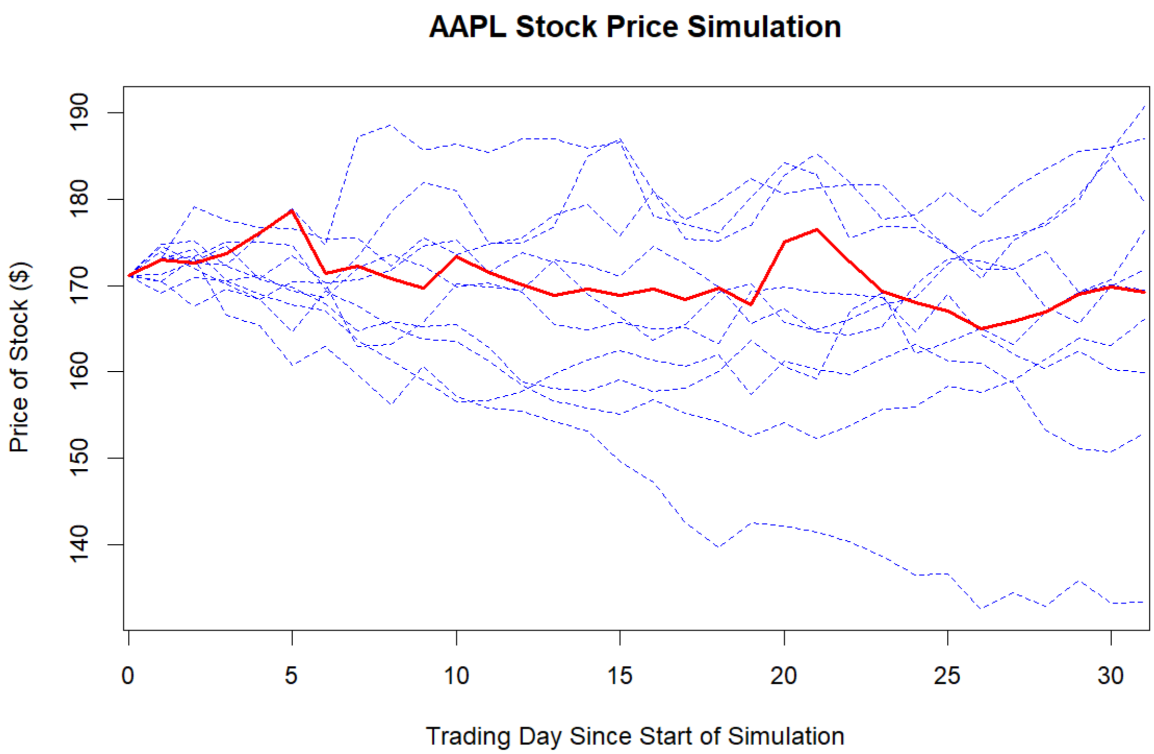
The main assumption that our model makes is that the historical data of the market leading up to the start of the simulation is a good representation of how the market will play out during the simulated time period. In addition to this, it assumes that what we are getting for the calculated drift and volatility of the stock is accurate to the actual drift and volatility. Though our numbers have appeared to be relatively what the real drift and volatility are, they are still just predicted calculations made based on the historical data of the stock.

Analysis and Discussion:

Our model takes in historical stock price and interest rate data from a relevant time frame for the period our simulation is running. It then extracts the daily closing prices of the stock from said data, taking the last closing price as the initial price of the stock. Then, our model calculates the daily returns of the stock from the closing prices using quantmod's dailyReturn() function. The daily returns are the percentage change of the stock on a given day and are what we use to calculate the volatility and drift of the stock. We have already discussed how to find volatility, as for the drift it is as simple as finding the average daily returns using the mean() function. The drift is then adjusted according to the interest rate as stated before.

After our model gets the drift, volatility, and initial price of a stock, it then makes use of our geometric Brownian motion function to simulate future stock prices. This function works by first creating a sequence/vector of "n + 1" time steps, n being the number of trading days our simulation runs and a time step being one trading day. This sequence represents the time points at which the stock price will be simulated. Then, the function calculates "dt" or the change in time/size of a time step, which will be one trading day divided by the total length of our simulation. Next, the function generates a vector of Brownian motion random variables, which is essentially a random positive or negative number that simulates the random fluctuation of a stock's price over time. It generates the vector using rnorm() to generate "n + 1" random normally distributed numbers with a standard deviation of one and a mean of zero. It then takes the cumulative sum of these values as the change in the stock's price will carry over to the next day. The vector is then multiplied by the square root of "dt" to scale the values appropriately. Finally, the function uses the geometric Brownian motion formula to generate future stock prices. This formula incorporates the drift (mu), volatility (sigma), initial price (s0), time sequence (t.s), and the Brownian motion variable (Bt) in order to model the stock's price motion over time.

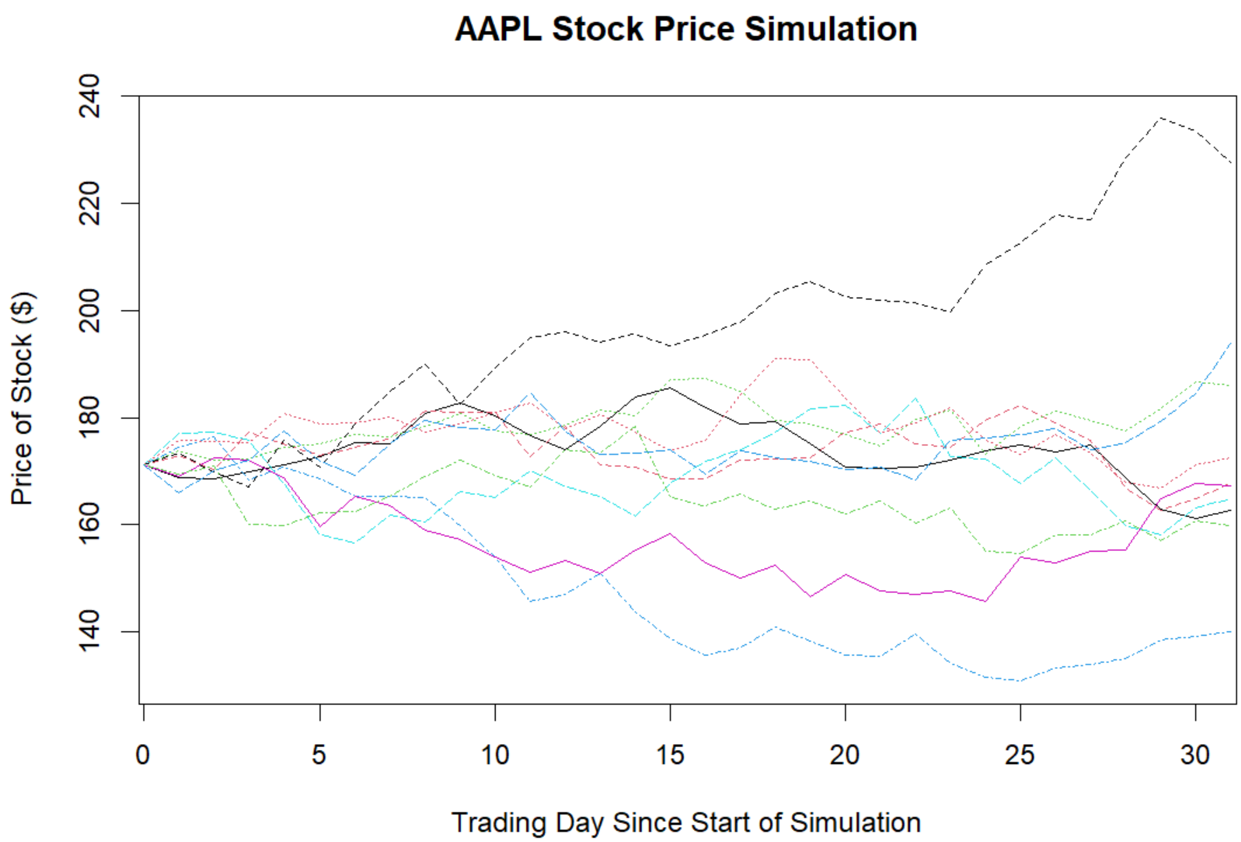
When running our Stock.Simulator() component of our model on default settings (days = 30, paths = 10, interest_rate = NULL) and passing Appl's symbol or "AAPL". Using historical data from before to the last thirty days to simulate the stock's price over the last thirty days and then compare that to the actual price trajectory. The simulated prices are represented by the dotted blue lines and the real stock is represented by the solid red line. In addition to making this graph, it returns a table of all the simulated prices and the real closing prices over the simulated time period.



	sim_prices	sim_prices.1	sim_prices.2	sim_prices.3	sim_prices.4	sim_prices.5	sim_prices.6	sim_prices.7	sim_prices.8	sim_prices.9	AAPL.Close
2024-03-13	171.1300	171.1300	171.1300	171.1300	171.1300	171.1300	171.1300	171.1300	171.1300	171.1300	171.13
2024-03-14	169.0652	173.9376	172.9571	173.0913	170.4814	173.1625	174.5780	174.8340	171.2765	170.5188	173.00
2024-03-15	170.8739	172.0221	173.3671	179.2056	173.1979	174.2305	172.6288	175.1232	172.7449	167.6036	172.62
2024-03-18	170.5781	170.1768	175.0848	177.5837	166.5897	170.5659	172.4444	172.3017	174.5439	169.6149	173.72
2024-03-19	171.2141	168.4379	175.0554	176.7984	165.3839	168.9396	175.8420	170.8440	170.6786	168.3825	176.08
2024-03-20	169.3911	170.4982	174.6237	176.6235	160.7588	167.7498	178.9047	173.4593	169.7896	164.5867	178.67
2024-03-21	168.6318	170.3362	169.5872	175.3828	162.9166	167.0735	174.7393	170.3859	167.9232	169.3449	171.37
2024-03-22	171.9621	170.7450	173.3014	175.5730	159.6416	163.7093	187.2083	162.9868	164.7680	167.6380	172.28
2024-03-25	173.5894	171.7694	178.5243	172.2740	156.1775	161.3992	188.6436	163.1890	165.8513	165.2336	170.85
2024-03-26	172.3021	174.6175	181.9731	175.5270	160.6213	159.0315	185.7235	165.7330	165.2869	163.8005	169.71
2024-03-27	169.8025	175.2463	181.0089	173.5639	157.1400	156.5925	186.4322	170.2404	165.4835	163.5682	173.31
2024-03-28	170.2725	171.4900	174.9659	174.8293	155.8247	156.6365	185.4247	169.8370	162.7834	161.2585	171.48
2024-04-01	169.2106	173.8071	174.9148	175.4883	155.4487	157.8057	187.0302	169.4329	158.7922	158.4477	170.03
2024-04-02	165.5357	172.7608	176.9099	178.1529	154.1875	159.7451	187.0482	173.0002	158.1563	156.6671	168.84
2024-04-03	164.8536	169.0576	184.9595	179.4449	153.1351	161.3909	185.8801	172.4286	157.8058	155.7692	169.65
2024-04-04	165.7674	166.4864	187.0302	175.7454	149.5842	162.4801	186.6904	171.0637	159.0277	155.1469	168.82
2024-04-05	165.0527	163.6479	180.9591	180.9364	147.1944	161.3319	178.0220	174.5600	157.7144	156.8127	169.58
2024-04-08	165.1257	165.4342	177.5829	175.4437	142.4673	160.5982	177.0577	172.5286	158.1247	155.2098	168.45
2024-04-09	163.3421	169.2821	179.7049	175.1174	139.5623	161.9433	176.1118	169.8797	160.0225	154.2515	169.67
2024-04-10	169.3178	170.1535	182.4088	176.9750	142.5376	157.4424	180.2988	165.5625	163.6936	152.5502	167.78
2024-04-11	169.8763	165.6973	180.6330	182.8467	142.1490	161.2473	184.2338	167.3588	160.6789	154.1304	175.04
2024-04-12	169.1970	164.8298	181.3741	185.3121	141.4730	160.2599	182.8682	164.6513	159.2390	152.3398	176.55
2024-04-15	168.9663	166.0558	181.6704	181.9817	140.3744	159.7180	175.5492	164.2752	167.0102	153.7360	172.69
2024-04-16	168.6387	167.8463	181.7526	177.7146	138.6974	161.5533	176.8534	165.2705	169.0924	155.7054	169.38
2024-04-17	164.6348	168.6340	177.7563	178.2452	136.3998	163.2212	176.7585	170.1070	162.1447	155.9127	168.00
2024-04-18	168.9405	172.4874	174.4242	180.7975	136.5934	161.2898	174.2887	173.0619	163.4644	158.3172	167.04
2024-04-19	164.4243	175.0217	170.7666	178.0321	132.4290	161.0720	171.9123	172.8635	164.9635	157.6102	165.00
2024-04-22	162.1225	175.7779	175.1854	181.2151	134.3634	158.6746	171.9317	171.9534	163.2396	158.9011	165.84
2024-04-23	160.4366	176.9963	177.3082	183.4834	132.8369	153.2504	173.9620	167.5220	166.9553	161.4631	166.90
2024-04-24	162.4914	179.8988	180.4889	185.6197	135.8553	151.1267	169.2881	165.5747	169.1908	163.9744	169.02
2024-04-25	160.2369	185.6746	184.9446	186.1068	133.2228	150.7341	170.0411	169.9242	170.6686	163.0675	169.89
2024-04-26	159.9656	190.7864	179.6347	187.0335	133.3494	152.8544	169.5184	171.8321	176.4188	166.1007	169.30

From this data, we can get an idea of the accuracy of our prediction model. In the process of making our project, creating this component of our model was an instrument for fine-tuning our model to be more accurate. For example, it is what allowed us to see that our volatility wasn't very accurate and make adjustments to how we went about calculating it. Overall it was an essential step in creating an effective model and if we were to continue this project and try to make our predictions even more accurate it would continue to assist us.

When running our other component Future.Stock.Simulator() on default settings (days = 30, paths = 10, interest_rate = NULL) and passing Appl's symbol or "AAPL". Using historical data from the previous thirty days to simulate the stock's price over the next thirty days. In this graph, each line represents a simulated stock price, and in each column of the corresponding table is the prices of a simulated path over the simulation period.



	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	171.1300	171.1300	171.1300	171.1300	171.1300	171.1300	171.1300	171.1300	171.1300	171.1300
2	168.8558	173.0021	173.8398	174.3944	177.1684	168.9499	173.3688	175.8895	169.4640	166.0909
3	168.6548	170.2672	171.9971	176.3586	177.3373	172.5360	169.9910	175.5674	171.0636	170.1674
4	169.8118	177.2517	172.3375	168.2971	175.7983	172.0229	166.9715	175.4618	160.0638	172.0431
5	171.2043	175.2464	174.5069	170.8763	167.6288	168.8070	175.7599	180.8063	159.8495	177.5572
6	172.7946	173.0708	175.1274	168.6127	158.1191	159.7146	170.8572	178.9071	162.3478	171.9156
7	175.4338	174.4684	176.8841	165.4376	156.6686	165.4302	178.7904	179.0288	162.5057	169.3024
8	175.1076	176.3271	176.3908	165.2953	161.9353	163.6015	184.9655	180.1897	165.3443	175.0344
9	180.7673	181.1538	178.3109	165.1929	160.5886	158.9047	189.8695	177.3534	169.1240	179.5814
10	182.7623	180.9848	180.9018	159.8757	166.2066	157.2302	182.5488	178.8075	172.1353	178.2949
11	180.3326	180.9785	177.4591	153.9786	165.0895	154.0709	189.2357	181.0464	169.2712	177.7554
12	176.6663	172.7320	176.8573	145.7602	170.1657	151.1173	195.0085	182.6908	167.0932	184.8072
13	174.0511	178.4121	178.7040	147.0262	167.3157	153.4421	196.0958	177.8850	174.1008	177.5880
14	178.3604	171.2170	181.5382	150.8872	165.2277	151.0502	194.0187	180.5840	173.3436	173.1472
15	183.9687	170.7386	180.3166	143.8607	161.7180	155.2209	195.6510	177.2683	178.5199	173.4040
16	185.6689	168.6398	187.0647	138.6360	167.8131	158.4109	193.4662	173.8204	165.1689	173.9521
17	181.8944	168.6617	187.2550	135.6801	171.9663	152.8930	195.4831	175.7262	163.5140	169.3803
18	178.9182	172.1424	185.0218	137.0835	174.0022	150.0581	197.9125	184.3221	165.6722	173.9052
19	179.3281	172.3464	179.5444	140.9471	177.3565	152.3834	203.2474	191.1092	162.9393	172.6265
20	175.2152	172.4221	178.8485	138.3332	181.7575	146.5327	205.3408	190.9064	164.3844	171.8981
21	170.8315	177.3576	176.9892	135.7574	182.3453	150.7851	202.6145	183.7193	162.0604	170.2413
22	170.5837	178.7514	174.7312	135.3947	177.0906	147.6347	201.8566	177.1278	164.3998	170.8840
23	170.8416	175.0382	179.8109	139.5736	183.5584	147.1117	201.6137	178.9639	160.3048	168.3360
24	172.0925	174.5314	181.4851	134.2645	172.8361	147.6758	199.8302	181.9963	163.1270	175.7731
25	173.8543	179.5807	173.2315	131.4668	172.4103	145.6995	208.4967	175.8526	155.0744	176.1441
26	174.8937	182.2947	178.4663	130.8210	167.8293	154.0263	212.6877	173.2350	154.6040	176.8507
27	173.6444	179.1429	181.2885	133.3189	172.4928	152.8887	217.8503	176.9141	158.2442	178.0295
28	174.9016	175.8433	179.5939	133.8484	166.3779	155.0746	217.0933	173.3950	158.0898	173.9886
29	168.9174	166.8920	177.4594	135.0128	159.9736	155.2412	228.4824	167.9595	160.8419	175.2739
30	162.8374	162.8075	181.6688	138.4431	158.1762	164.9839	235.9862	166.8570	156.9555	179.2914
31	161.2420	164.9318	186.5953	139.0851	163.1401	167.6498	233.6671	171.3226	160.7493	184.5347
32	162.7513	167.7293	186.0311	139.9539	164.9311	167.1887	227.6469	172.6248	159.8634	193.8667

Based on these simulated prices, we can look at the average ending price to determine if we think the stock is going to go up or down. The level of accuracy of this prediction can be obtained from our other function from before. In this case, I would conclude that the stock price of Apple will go down slightly over the next thirty days.

Questions and Answers:

When coming into this project neither of us had any real idea about how the stock market functioned other than that prices go up-and-down over time. We didn't know the factors that influenced a stock's price and we wanted to both learn about these factors and how we could then use them to predict future prices. We wanted to learn about various trading strategies, and based on our simulation what we would think to work the best, effectively using our simulation to make investment decisions. We ultimately wanted to gain a deeper understanding of the stock market and be able to predict future stock prices with some level of confidence. In addition to learning about the stock market, we wanted to learn and develop our simulation skills, which would have to in order to be able to accurately simulate a stock's price.

Through the completion of this project, I would say we answered all these questions and more. We now have extensive knowledge about factors that influence a stock's price, like company performance and interest rates. We also know the variables that are used to quantify a stock's trajectory, volatility and drift. Along with how to both calculate them based on past stock performance and then use them to predict future prices. During the extensive tests that we performed on our simulation using a multitude of different stocks, we concluded that in most cases making your investment choices based on the overall current drift of a stock while taking into account the current interest rate is usually the best choice when investing. In contrast to trying to day trade and make a profit off of a stock's volatility which is too random to try and account for. In addition to gaining a deeper understanding of the stock market and its movement, we also learned quite a bit about simulation and statistical analysis. We learned about geometric Brownian motion and how to build a function to simulate it in R. We how to use packages(quatmod) and online datasets(stock data) in R and how to then extract what we need from said datasets using said packages. In addition to this, we had to learn more about graphing in R by constructing and plotting matrices. Overall, through the project's development, we were able to address our initial questions while also uncovering additional questions that emerged throughout the process.

Conclusion:

In conclusion, our research aimed to deepen our understanding of the stock market and develop a simulation in R for predicting stock behavior. We analyzed factors such as company performance, industry trends, and interest rates to comprehend their impact on stock prices and figure out a way to simulate them. We developed functions to simulate past and future stock prices using historical data of the stock and the geometric Brownian motion formula. Throughout the project, we refined our model using our Stock.Simulator() function that compared our simulated prices to the real price of a stock. This allowed us to fine-tune aspects of our model like our volatility calculations and the incorporation of interest rates on drift.

Initially, we were unfamiliar with the stock market mechanisms, but despite that, we were able to gain extensive knowledge about factors affecting stock prices and simulation techniques. Ultimately, we were able to learn extensively about how the stock market functions and learned applicable knowledge like that buying-and-holding onto a stock that has a positive long-term drift is much safer and wiser than trying to buy and sell when a stock randomly fluctuates. This research not only addressed your initial inquiries but also expanded our understanding of the stock market and simulation methodologies.

In addition, throughout the creation of this project, we ended up running into some limitations. This included having more ideas for our model that we were unable to implement/ more questions that we were unable to answer in the given time frame for the project. One thing that we would be interested in implementing, if we were to continue the development of this project, is having an average path or endpoint of all simulated paths to give an overall idea of if it would be wise to invest in the stock. Another implementation that we think could have been useful would be to quantify or compute the accuracy of our predictions when we compare our simulated paths to the real path of a stock. In addition to these implementations, we also would like to have answered questions like what drift/ volatility would make a stock considered a worthy investment. This question would combine aspects like the stock's current trajectory versus the entire market. Overall, though there were some questions that we might be leaving unanswered and implementations not implemented, we are quite satisfied with our final product and can confidently say that we exceeded what we initially sought to do. This is especially prevalent when looking back at our first meeting where we had no clue on how we would even start this project or what it would entail.

Bibliography:

Stat Legend. (2021, November 21). *Geometric Brownian Motion (GBM) Simulation in R* [Video]. YouTube. https://www.youtube.com/watch?v=ZIRxkD6SOLQ

Pinsent, W. (2021, September 30). *Understanding stock prices and values*. Investopedia. https://www.investopedia.com/articles/stocks/08/stock-prices-fool.asp#:~:text=The%20stock%E2%80%99s%20price%20only%20tells,buyers%2C%20the%20price%20will%20drop

Yearner, M. (2024, January 16). How to simulate stock prices - Machine Yearner - medium. *Medium*. https://medium.com/@MachineLearningYearning/how-to-simulate-stock-prices-452042862989

Algovibes. (2020, December 19). *Simplified stock price simulation in Python [14 lines of code] using Monte Carlo methods* [Video]. YouTube. https://www.youtube.com/watch?v=LWc-9v8RVwM

Appendix: R Code

```
install.packages('quantmod')
library(quantmod)


# Simulate stock prices, then plots and returns said simulates prices against the real prices
# symbol - symbol of stock
# days - trading days of simulation
# paths - number of simulated realizations of stock
# interest rate - Allows user to enter custom interest rate,
#                defaults to true interest rate at start of simulation
Stock.Simulator <- function(symbol, days = 30, paths = 10, interest_rate = NULL) {

  # downloads historical prices for double the amount of trading days we're
  # simulating + 10(min amount of data for training), takes double since
  # it'll be split for training data
  getSymbols(symbol, from = Sys.Date() - 2.91*(days) - 10)

  # Gets closing values for stock
  closings <- Cl(get(symbol))

  # separates test and training data based on # of simulation days
  closing_train <- closings[1:(length(closings) - days)]
  closing_test <- closings[(length(closings) - days - 1):length(closings)]

  # Calculates daily returns
  returns <- dailyReturn(closing_train)

  # Calculates volatility based on last 30 days(tested for a while and this way gave best results)
  vol <- mean(c((as.numeric(tail(volatility(closing_train, n = days), 1))), sd(returns)))

  # Calculates drift
  drift <- mean(returns)
```

```
  # downloads historical treasury interest rate data,
  interest_rates <- getSymbols("DGS10", src = "FRED",
                              from = Sys.Date() - 6*(days) - 10, auto.assign = FALSE)

  # sections off the interest rate data so we are only using data from before start of simulation
  interest_rates <- interest_rates[1:(length(interest_rates) - days)]

  # checks if user inputted custom interest rate
  if (is.null(interest_rate)) {
    # Calculates last interest rate
    latest_interest_rate <- as.numeric(tail(interest_rates, 1))
  } else {
    latest_interest_rate <- interest_rate
  }

  # Calculates mean of past interest rates
  mean_interest_rate <- mean(interest_rates, na.rm = TRUE)

  # finds difference for avg interest rate vs current rate
  cur_rate_diff <- mean_interest_rate - latest_interest_rate

  # adjusts drift to account for current interest rate
  drift <- drift + (cur_rate_diff/1000)

  # gets last price of stock
  init_price <- as.numeric(closings[(length(closings) - days - 1)])

  # gets simulated prices
  sim_prices <- Simulated.Prices(drift, vol, init_price, days, paths)

  # adds real prices to matrix
  sim_prices_and_real <- cbind(sim_prices, closing_test)

  # plots simulated stock prices(dotted blue) vs real prices(bold red)
  matplot(c(0:(days + 1)), sim_prices_and_real, type = "l",
          xlab = "Trading Day Since Start of Simulation",
          ylab = "Price of Stock ($)",
          xlim = c(1, days),
          main = paste(symbol, "Stock Price Simulation"),
          col = c(rep("blue", paths), "red"),
          lty = c(rep(2, paths), 1),
          lwd = c(rep(1, paths), 2))

  return (sim_prices_and_real)

}

#test
apple_table <- Stock.Simulator("MSFT", days = 20, paths = 30)
```

```
# Simulates and plots/returns future stock prices
# symbol - symbol of stock
# days - trading days of simulation
# paths - number of simulated realizations of stock
# interest rate - Allows user to enter custom interest rate,
#               defaults to true interest rate at start of simulation
Future.Stock.Simulator <- function(symbol, days = 30, paths = 10, interest_rate = NULL) {

  # downloads historical prices based on the amount of trading days we're
  # simulating + 10(min amount of data for training)
  getSymbols(symbol, from = Sys.Date() - 1.45*(days) - 10)

  # Gets closing values for stock
  closings <- Cl(get(symbol))

  # Calculates daily returns
  returns <- dailyReturn(closings)

  # Calculates volatility based on last 30 days(tested for a while and this way gave best results)
  vol <- mean(c((as.numeric(tail(volatility(closings, n = days), 1))), sd(returns)))

  # Calculates mean return (drift)
  drift <- mean(returns)

  # downloads historical treasury interest rate data
  interest_rates <- getSymbols("DGS10", src = "FRED",
                               from = Sys.Date() - 3*(days) - 10, auto.assign = FALSE)

  # checks if user inputted custom interest rate
  if (is.null(interest_rate)) {
    # Calculates last interest rate
    latest_interest_rate <- as.numeric(tail(interest_rates, 1))
  } else {
    latest_interest_rate <- interest_rate
  }

  # Calculates mean of past interest rates
  mean_interest_rate <- mean(interest_rates, na.rm = TRUE)

  # finds difference for avg interest rate vs current rate
  cur_rate_diff <- mean_interest_rate - latest_interest_rate

  # adjusts drift to account for current interest rate
  drift <- drift + (cur_rate_diff/1000)

  # gets last price of stock
  init_price <- as.numeric(closings[(length(closings) - days - 1)])

  # gets simulated prices
  sim_prices <- Simulated.Prices(drift, vol, init_price, days, paths)
```

```
  # plots simulated stock prices
  matplot(c(0:(days + 1)), sim_prices, type = "l",
          xlab = "Trading Day Since Start of Simulation",
          ylab = "Price of Stock ($)",
          xlim = c(1, days),
          main = paste(symbol, "Stock Price Simulation"))

  return (sim_prices)

}

#test
future_apple_table <- Future.Stock.Simulator("AAPL")


# Simulates multiple price paths and store them in a matrix and returns said matrix
# drift - drift of the stock
# vol - volatility of the stock
# init_price - initial price of the stock
# days - trading days of simulation
# paths - number of simulated realizations of stock
Simulated.Prices <- function(drift, vol, init_price, days = 30, paths = 10) {

  # creates matrix to hold all possible realizations
  sim_price_matrix <- matrix(0, nrow = days + 2, ncol = paths)

  for (i in 0:paths) {

    # Unexpected event(1 in a 1000) making volatility double
    if (runif(1) < .001) {
      vol <- vol*2
    }

    # Uses GBM function to generate new stock prices
    sim_price <- c(init_price ,gbm.f(n = days, s0 = init_price, mu = drift, sigma = vol))

    # appends matrix to hold new path stock could take
    sim_price_matrix[,i] <- sim_price
  }
  return (sim_price_matrix)
}

# Source: Stat Legend - https://www.youtube.com/watch?v=ZIRxkD6SOLQ
# Geometric Brownian Motion Stock Price Simulator
# n - end time
# s0 - init price
# mu - drift
# sigma - volatility
gbm.f <- function(n, s0, mu, sigma) {
  #Get a time horizon, 1 day
```

```
  t <- 1

  # time step, makes sequence from 0 to end time by time horizon
  t.s <- seq(0,t,length=n+1)

  # change in time
  dt <- t/n

  # generating Brownian random variable
  Bt <- sqrt(dt)*cumsum(rnorm((n+1),0,1))

  # Uses GBM formula to generate future stock prices
  St <- s0*exp((mu-sigma^2/2)*t.s+sigma*Bt)

  return (St)

}
```