Fall 2022 QMST 3339 Final Project

Marcus Block

Jamal Coleman

Jacob Neeley

GPU After-Market Price Analysis

References:

Tom's Hardware, GPU Prices: Tracking Graphics Cards Sold on eBay;

https://www.tomshardware.com/news/gpu-pricing-index

BlockChain.com | Charts – Bitcoin MKT Price (USD), Last 3 years;

https://www.blockchain.com/explorer/charts/market-price

GPU prices are still on a decline: Is Bitcoin's sorrow gamers' joy?

https://www.cointelegraph.com/news/gpu-prices-are-still-on-a-decline-is-bitcoin-s-sorrow-gamers-joy

Market capitalization of Bitcoin (BTC) from April 2013 to November 14, 2022

https://www.statista.com/statistics/377382/bitcoin-market-capitalization/

Table of Definitions

Crypto	A digital currency designed to work as a medium of exchange through a computer network not reliant on any central authority (government or banks)
GPU	Abbreviation for Graphics Processing Unit: A processor originally designed to accelerate graphics renderings
QTY Sold	Quantity: amount or number of a GPUs Sold
GPU[Column]	Model Name of a specific GPU.
AMD	A computer corporation who makes Radeon CPU (Central Processing Unit) and GPUs
NVIDIA	Graphics Card Manufacturer in California, United States with market capital size of $389.34
GeForce	NVIDIA brand GPUs
Radeon	AMD produced GPUs
BTC	Bitcoin: One of the biggest crypto or digit currencies on the market. BTC had a market capital size of $319.19 Billion USD on November 14th, 2022
Btc_Price (Variable)	Price of BTC from a specific point in time
eBay Price	After Market price of a gpu recorded from eBay. Provided by (courtesy of Tom's Hardware)

Executive Summary

It is essential to note that several factors can influence Graphics Cards' after-market prices. These factors include, but are not limited to, the following:

- Inflation
- 2020-Present global supply chain crisis.
    - Global Chip Shortage
- Potential trend relating to BTC Prices

The focus of this study was to investigate the correlation between Bitcoin valuations and various price points that GPU models sold for on the popular eBay marketspace. After our analysis, we reject our null hypothesis that the valuation of Bitcoin does not affect after-market GPU prices.

Crypto experienced an all-time high back in November 2021, now 1 year later Bitcoin has fallen to about $16K from its all-time high of $67,572 (back in Nov. 08, 2021). During the global pandemic, GPU's have been well above MSRP. Now that crypto currencies like Bitcoin have fallen, GPU prices have begun to fall. Correlation does not equal causation, but it was interesting to see since powerful GPUs are needed to mine cryptocurrency effectively.

Assume we are a company that sells new and secondhand GPUs. With our observations, we built a model using bit coin prices from the last 3 years and the average GPU prices from the last 25 months (about 2 years). Unfortunately, we found that our model would not be a great predictor for after-market GPU prices. In the context of our data, the model violated several linear regression assumptions. We found that the correlation between the eBay Prices and BTC_Prices that we observed was only .4929. When adding QTY sold and factoring for each GPU in our model, we increased the quality of our model (Adj. R-Squared increased) and found that we could explain 86.43% of the variations. We were able to conclude that Bitcoin price does in fact influence after-market (ebay) GPU prices, but as mentioned earlier, our model is not a great predictor. Our model's predictions severally underpredict values when compared to what we observed. In some cases, we greatly overpredict values in comparison to what we observed. Based on these findings, we believe that our model is not currently capable of correctly predicting eBay Prices when controlling for the independent variable BTC_Price and outside factors. An effective model could help us maximize our profits as efficiently as possible by

giving us insights into what prices we should be selling our products at, especially high performing GPUs.

Problem and Data Description

Problem:

A big problem in the GPU market are scalpers who wish to profit off new product releases, and to profit off the Crypto miners looking to acquire more GPUs to reduce mining time. Given the problem, our goal is to model the changes in GPU eBay prices with the changes in Bitcoin Price and the model (product name) of GPUs. Doing so, and correctly predicting prices in response to Bitcoin price, would allow our company to adequately price our product and maintain a consistent turnover so that we are not pricing our products too low. If we leave profits on the table, then we are not pricing our products high enough, leaving our products with fewer customers willing to buy.

To achieve this, we have constructed a Forward-Built MLR model that splits our observed data into training and test data sets to predict GPU eBay Prices given a specific Bitcoin price and determine if our model's guesses are correct. Our observations include Bitcoin prices from the last 3 years from blockchain.com (linked in references) via a downloadable csv, and average monthly GPU Price data scraped from Tom's Hardware that includes the average prices of the top-selling GPUs from the past 2 years (also linked in references).

Data Description:

The raw data we have collected comes from two sources: Blockchain.com and Tom's Hardware.com.

Blockchain.com is a cryptocurrency financial services company. They allow a user to view all kinds of data related to different currencies under the explore page. Under the charts tab, a user can explore and view different stats of different currencies. Our Bitcoin price data is from the market price (USD) over time chart. Under the data tab, a user can choose to download the data they are looking at in the form of csv. We downloaded the price data from the last 3 years. The raw csv had data that filled 2 columns, time (which was measured using Unix Epoch Time in Milliseconds) and price.

Tom's Hardware has a blog page that provides monthly updates on Average GPU prices from eBay. The author, Jared Walton, has been updating this page since February 2021. The prices were scraped from eBay, likely using a python script, and averaged. The author then arranges them into tables. We collected the same data by creating our own python script that would scrape the tables that the author had published publicly online. The data scraped by Jared Walton originally had just 3 columns: GPU, eBay price, QTY sold. Each row was populated with the product name, its respective average price for the month, and the quantity sold. In more recent months, 4 additional columns were added to keep things consistent. We only scrapped the original 3 columns because if we decided to include the 4 additional columns, we would have to come up with a method to fill them. Something we decided was out of scope for this project.

Describing and Visualizing Data

Python Scripts:

For this project, Python scripts were written to scrape data from Tom's Hardware using BeautifulSoup and paganation to collect data from multiple pages. The script can handle data from the Bitcoin CSV downloaded from blockchain.com, mine and create new data, and clean the data before creating Pandas data frames that were then used to export our refined data as CSVs. All Python code will be referenced from the appendix.

collect_gpuData.py - python script to collect GPU data from tables: Average GPU price was scraped from Tom's Hardware. All the data was nicely arranged in tables with columns, rows and headers on the website. [1&2] Using BeautifulSoup, it was easy to parse the webpage, find the table using .find() function, then using a for loop to populate a list containing the headers from the table which was then used to create a Pandas' data frame. [3] Each page represents 1 month; we need to collect as many pages as we see fit. We decided 25/30 would be good as the tables were introduced until page 25 (page 1 represents the most recent month). We needed to grab the information for each month from each page, this is paganation. A while loop was constructed and in that while loop is a for loop that collects data row by row and appends it to the data frame. The while loop will work if the variable current_page is less than max_page. See appendix for more specifics. [4&5] data is clean a bit, [6] then exported as a csv.

BtcPrice_Time.py: Bitcoin market prices from the last 3 years were downloaded as a csv from blockchain.com. All that was given was an x and y value. Some context needed to be added

before this information could be used. [0] The csv was read into a Pandas' data frame object 'BTC.' for loop was constructed to iterate through each row in btc['x'] and convert each value to DateTime. [1] Time was given in column as Unix Epoch Time in Milliseconds, x was re-initialized to BTC['x'] = x/1000 to convert time to Unix epoch time in seconds. [2] Columns were renamed. [3] New column was created called DateTime and inserted into that column were the date time's appended to Date_Time list done in [0]. [4] DataFrame was exported as a csv.

DataClean.py - Read in raw data, clean it up: [0] Read in CSVs from Collect_gpuData.py and BtcPrice_Time.py as their own respective dataframe objects. [0.a] Any value less than March 2021 (based on epoch time) will be dropped using a conditional drop statement. [1] There are 362 values in GPU, 25 months and approximately 15 distinct GPUs. BTC contains daily values from March 2021 to October 2022. To match these daily values to gpu, Prices by index we decided to take a sample of the BTC prices to represent the averages of their respective months. A Dataframe object called bsamps was created. [2] String values were assigned to each datatime to make things a little easier to read. [3 & 4] data was cleaned, conversions were made, and data that did not meet certain conditions were dropped. [5] A dataframe object was created which contains information from the other two dataframes which will be used to build our model.

btc_gpu_pricemodel.R - Forward Built MLR

Figure 1 shows the residual vs. our predictions for the model. The graph shows that the variance is mostly constant, but as residuals increase, we begin to witness an increase and decrease in the variance. We believe the model is not a good fit for the data and the price difference with some of the cards from eBay is creating bad predictions in our model. This can be explained by the variation in prices and the presence of statistically insignificant cards present in the model. Residuals with constant variation fall between 500 to 1000.

Here at figure 2, we have a lm plot showing the linear association between eBay Prices and BTC_Prices. The green line represents the mean (average) of eBay price while the red line represents the regression model we are using for the project (eBay.Price~btc_Prices+QTY.Sold+factor(GPU)). Market conditions and poor data can potentially explain why the data is plotted all over the place. We can see that the data does not look very linear.

Figure 3 shows us fitted values (estimated values) when we factor in the GPU. The redline can show us where we are under and over predicting. 0 to 500 we are seeing some underpredictions. 500 to 1000 we observe some over predictions but fitted values 1500+ are severely under predicting.

Analysis

Correlation: Before we begin building our model, we need to understand a few things about our data. eBay. Prices will be our dependent variable, what we will predict and attempt to model. We will start with figure 4 down in the appendix. This is our correlation. We can see that, relative to eBay.Price, BTC prices have a positive correlation of .4929 and while not as strong, QTY sold has a positive correlation of .0863.

Linear Regression equation – summarize our data: [figure 5] building linear model object "o1" we can summarize our data. Given a reasonable alpha value .05 and the null hypothesis: Bitcoin prices have no effect on gpu eBay Prices (beta_1 = 0). Viewing the p-value relative to the f-stat we reject the null hypothesis that Bitcoin prices have no effect on eBay prices. P-value = 2.26e-16 which would be smaller than any reasonable alpha-value. The intercept of the dependent variable is 1.425e+02, if btc_Prices were 0 than eBay. Prices would be equal to the intercept. 1 unit increase in btc_Prices would increase eBay.Price by 2.289e-2. While btc_price has an effect we can only explain about 24% of our variations in prices.

Figure 6.b shows the coefficient estimates summary of lm(eBay.Price~btc_Prices+factor(GPU)). Using a forward building approach, we want to see if we can improve our model by adding features. Each eBay.Price has an associated string value notated as the GPU model. This value can be evaluated separately by R using the factor function. Figure 6.a shows the coefficient estimates of each GPU. It is important to note that not all of them are statistically significant, in other words they are not sensitive to changes in btc_Price. This is because some budget cards are not as desirable, especially to customers interested in crypto mining. Keep this in mind we will come back to this later.

Figure 7 shows us the coefficients estimates of lm(eBay.Price~btc_Prices+QTY sold). Their summary output is nothing remarkable. It does not appear statistically significant, and the adjusted R-Squared value is only .2388, worse in comparison to o1, even worse in comparison to model from figure 6. b. Figure 7. B and 7.c shows us the coefficient estimate of

lm(eBay.Price~btc_Prices+QTY sold+factor(GPU)) and the summary output, respectively. In this model QTY sold becomes statistically significant, and the Adjusted R squared value sits at .855, higher than lm(eBay.Price~btc_Prices+factor(GPU)) from figure 6.b. Attempting to build models with month and datetime variables did not produce any significant. Thus, the model presented in figure 7.b and 7.c is what we decided to proceed with.

Returning to figure 2, we can see once we plot our regression equation its fit appears rather poor. It is also easy to see that our data does not appear very linear. It can be argued that visually, it violates the linearity assumption. From this we begin to accept that our model is not a better predictor than the average value of the dependent variables' coefficient. From here on out we will see issues our model has.

Training and Test Set Evaluations: A for loop can be found in the R script that attempts to build a training and test set that we could better understand our model. Data in the test set will contain values we observed. When compared, will our models' guesses be correct? No. After successfully running the loop to build our test and training set, we found that our model was significantly overpredicting and significantly underpredicting data, at a given point. This is because our upper and lower bounds of predictions were to spread out. The difference between upper and lower bounds of the predictions is often close to 1000. Figure 8 shows pred [,1] - eBay.Price, what was observed subtracted from the predictions. We can see they are quite bad.

Linear Regression Assumption Violations: Does our model violate any of the regression assumptions? Yes. Figure 1 shows us constant variance in residuals, but it is fair to say that we can see some increases here and there. Figure 9 is a Residual vs. Fitted plot that shows us where in our model we are underpredicting our values. Assuming the null hypothesis that residuals are normally distributed, when running the [figure 10] lillie test we fail to reject the null. The p-value is .05138, greater than a reasonable alpha. Residuals are not autocorrelated, see figure 11.

Conclusion. Given our observations, our model sucks.

Final Summary

With our observations, we constructed a forward-Built MLR. Using bitcoin prices from the last 3 years, and average GPU prices from the last 25 months (about 2 years). Our model violated 2 linear assumption tests. It is reasonable to believe that the statistically insignificant

GPUs present in our data caused our model to suffer, and that our model does not predict values better than the dependent variables intercept, even though our complex model was a better predictor than our simple model. The only thing we were able to conclude in this study is that we can reject the Null Hypothesis that states that btc_Price has no effect on GPU eBay Prices; however, given our observations we were not able to adequately create predictions based on btc_Prices that would provide business value for the company at this time.

Appendix

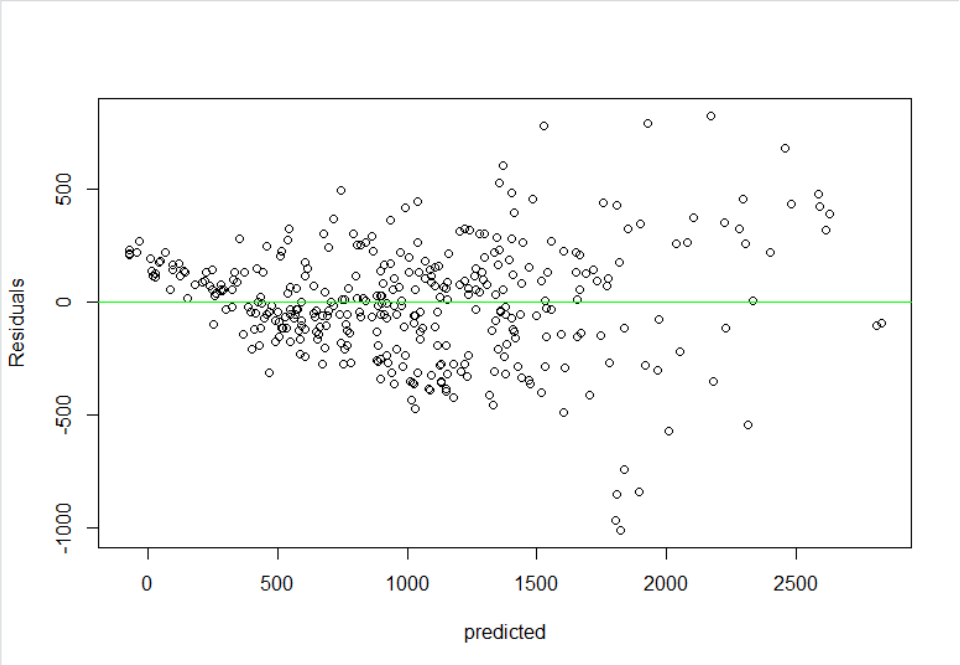

Figure 1

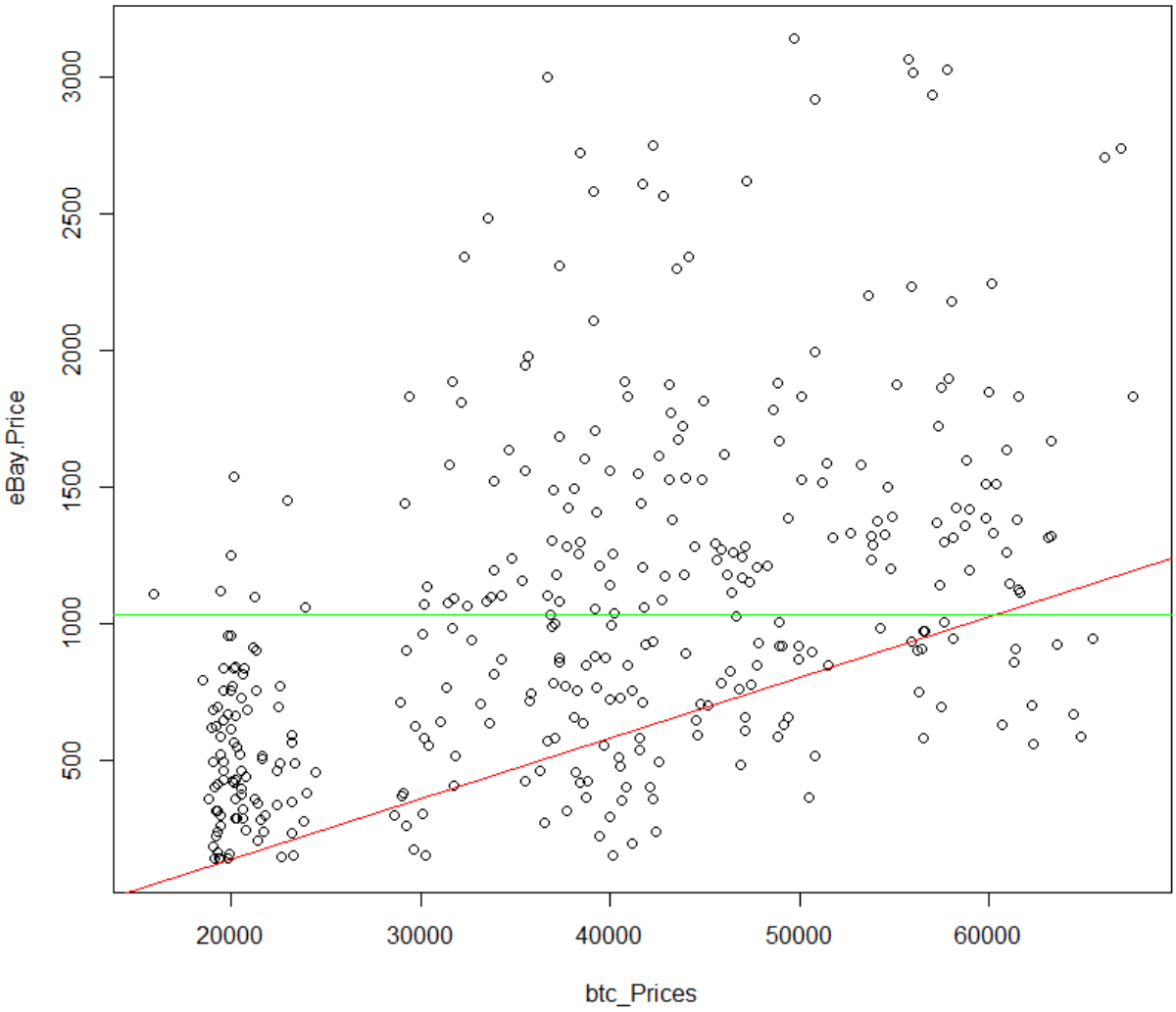
eBay.Price
500
1000
1500
2000
2500
3000
20000
30000
40000
50000
60000
btc_Prices

Figure 2

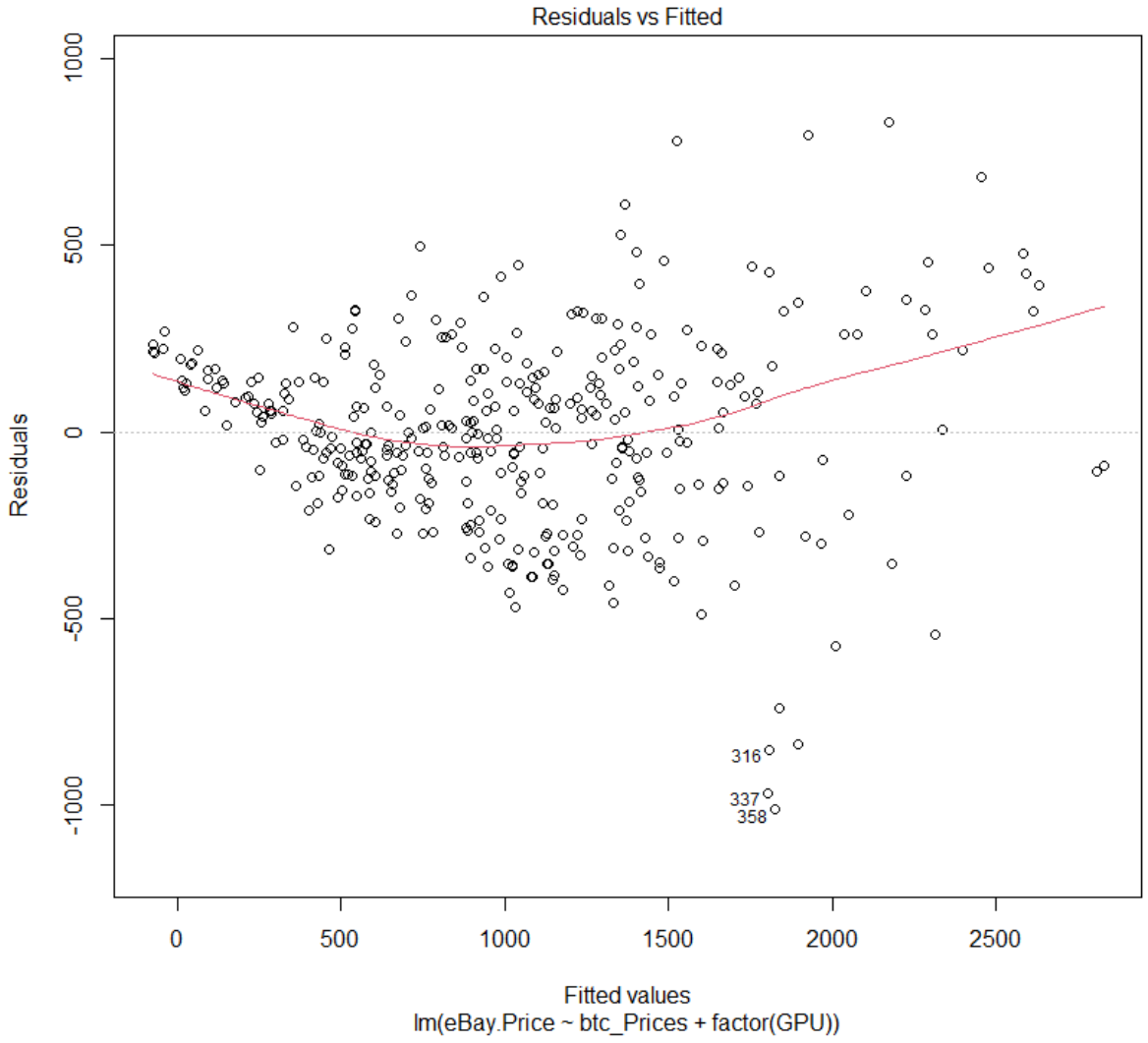

Figure 3

```
> round(cor(mktsamples[c(-1,-2,-4)]),4)
            btc_Prices eBay.Price QTY.Sold
btc_Prices      1.0000     0.4929   0.1937
eBay.Price      0.4929     1.0000   0.0863
QTY.Sold        0.1937     0.0863   1.0000
```

Figure 4

```
Call:
lm(formula = eBay.Price ~ btc_Prices)

Residuals:
     Min       1Q   Median       3Q      Max
-1038.61  -381.90   -91.14   236.43  2020.33

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.425e+02  8.785e+01   1.622    0.106
btc_Prices  2.289e-02  2.138e-03  10.704   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 553 on 357 degrees of freedom
Multiple R-squared:  0.243,     Adjusted R-squared:  0.2408
F-statistic: 114.6 on 1 and 357 DF,  p-value: < 2.2e-16

> #p-value = 2.2e-16; smaller than any reasonable alpha. Reject Null hypothesis.
>
```

Figure 5

```
Coefficients:
                                Estimate Std. Error t value Pr(>|t|)
(Intercept)                   -3.272e+02  8.933e+01  -3.663  0.00029 ***
btc_Prices                     2.167e-02  1.108e-03  19.557  < 2e-16 ***
factor(GPU)GeForce RTX 3060    1.467e+02  1.348e+02   1.088  0.27727
factor(GPU)GeForce RTX 3060 12GB  6.473e+01  1.033e+02   0.627  0.53131
factor(GPU)GeForce RTX 3060 T     3.160e+02  9.819e+01   3.218  0.00141 **
factor(GPU)GeForce RTX 3070       4.314e+02  9.821e+01   4.393 1.50e-05 ***
factor(GPU)GeForce RTX 3070 T     4.661e+02  1.004e+02   4.641 4.99e-06 ***
factor(GPU)GeForce RTX 3080       9.229e+02  9.819e+01   9.400  < 2e-16 ***
factor(GPU)GeForce RTX 3080 12GB  6.513e+02  1.198e+02   5.436 1.05e-07 ***
factor(GPU)GeForce RTX 3080 T     1.044e+03  1.006e+02  10.382  < 2e-16 ***
factor(GPU)GeForce RTX 3090       1.706e+03  9.823e+01  17.364  < 2e-16 ***
factor(GPU)GeForce RTX 3090 T     1.424e+03  1.238e+02  11.500  < 2e-16 ***
factor(GPU)Radeon RX 6400        -7.548e+01  1.348e+02  -0.560  0.57581
factor(GPU)Radeon RX 6500 XT     -1.619e+02  1.166e+02  -1.389  0.16587
factor(GPU)Radeon RX 6600        -1.272e+02  1.083e+02  -1.174  0.24120
factor(GPU)Radeon RX 6600 XT     -4.677e+01  1.043e+02  -0.448  0.65421
factor(GPU)Radeon RX 6650 XT      1.151e+02  1.348e+02   0.854  0.39381
factor(GPU)Radeon RX 6700 XT      1.299e+02  9.844e+01   1.319  0.18797
factor(GPU)Radeon RX 6750 XT      2.307e+02  1.347e+02   1.713  0.08770 .
factor(GPU)Radeon RX 6800         4.260e+02  9.836e+01   4.331 1.96e-05 ***
```

Figure 6.a

```
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 260.7 on 336 degrees of freedom
Multiple R-squared:  0.8417,    Adjusted R-squared:  0.8313
F-statistic: 81.18 on 22 and 336 DF,  p-value: < 2.2e-16

> #Multiple R-squared:  ~.8417, Adjusted R-squared:  .8313!!!
>
```

Figure 6.b

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 145.232948  89.018687   1.631    0.104
btc_Prices    0.022975   0.002183  10.526   <2e-16 ***
QTY.Sold     -0.009875   0.048888  -0.202    0.840
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 553.8 on 356 degrees of freedom
Multiple R-squared:  0.243,     Adjusted R-squared:  0.2388
F-statistic: 57.15 on 2 and 356 DF,  p-value: < 2.2e-16
```

Figure 7

```
Coefficients:
                                 Estimate Std. Error t value Pr(>|t|)
(Intercept)                    -3.272e+02  8.933e+01  -3.663  0.00029 ***
btc_Prices                      2.167e-02  1.108e-03  19.557  < 2e-16 ***
factor(GPU)GeForce RTX 3060     1.467e+02  1.348e+02   1.088  0.27727
factor(GPU)GeForce RTX 3060 12GB 6.473e+01  1.033e+02   0.627  0.53131
factor(GPU)GeForce RTX 3060 T   3.160e+02  9.819e+01   3.218  0.00141 **
factor(GPU)GeForce RTX 3070     4.314e+02  9.821e+01   4.393 1.50e-05 ***
factor(GPU)GeForce RTX 3070 T   4.661e+02  1.004e+02   4.641 4.99e-06 ***
factor(GPU)GeForce RTX 3080     9.229e+02  9.819e+01   9.400  < 2e-16 ***
factor(GPU)GeForce RTX 3080 12GB 6.513e+02  1.198e+02   5.436 1.05e-07 ***
factor(GPU)GeForce RTX 3080 T   1.044e+03  1.006e+02  10.382  < 2e-16 ***
factor(GPU)GeForce RTX 3090     1.706e+03  9.823e+01  17.364  < 2e-16 ***
factor(GPU)GeForce RTX 3090 T   1.424e+03  1.238e+02  11.500  < 2e-16 ***
factor(GPU)Radeon RX 6400      -7.548e+01  1.348e+02  -0.560  0.57581
factor(GPU)Radeon RX 6500 XT   -1.619e+02  1.166e+02  -1.389  0.16587
factor(GPU)Radeon RX 6600      -1.272e+02  1.083e+02  -1.174  0.24120
factor(GPU)Radeon RX 6600 XT   -4.677e+01  1.043e+02  -0.448  0.65421
factor(GPU)Radeon RX 6650 XT    1.151e+02  1.348e+02   0.854  0.39381
factor(GPU)Radeon RX 6700 XT    1.299e+02  9.844e+01   1.319  0.18797
factor(GPU)Radeon RX 6750 XT    2.307e+02  1.347e+02   1.713  0.08770 .
```

Figure 7.b

```
Residual standard error: 241.7 on 335 degrees of freedom
Multiple R-squared:  0.8643,    Adjusted R-squared:  0.855
F-statistic: 92.75 on 23 and 335 DF,  p-value: < 2.2e-16
```

Figure 7.c

5	7	9	10	12	15	18	20	21	24
-1002.257798	248.652541	1411.049858	-1135.150890	-1504.023504	-166.388387	1097.615903	-143.537157	-885.968232	-17.523577
27	28	31	32	36	42	43	45	50	56
1804.534839	-251.792958	681.348799	321.524755	1409.787884	-639.514676	-503.564723	287.787683	-1410.524371	690.477746
59	61	62	64	66	68	74	76	80	81
-374.312465	-1457.414252	723.751544	-293.302258	307.451960	602.965967	-2131.092795	-529.615284	447.208516	-204.094631
84	86	89	98	99	105	106	108	113	121
-186.204388	-1228.412382	488.731127	-939.926880	-48.170506	-706.746840	-992.497783	116.522275	774.936830	890.817630
126	128	130	133	134	143	144	145	151	157
785.269011	1146.450684	-659.989347	1137.038393	1129.980933	-52.680161	-503.850449	-1158.283726	743.854588	113.486534
163	169	171	173	177	179	183	184	185	187
-209.611870	-369.614485	57.916743	376.807375	-810.492293	-1777.741620	1597.099572	-828.220565	-1962.421601	329.451367
192	194	197	198	199	207	213	217	225	231
145.951658	490.799921	-366.521947	1232.321595	1667.601447	-474.839656	1018.789617	69.466927	409.349334	1092.215202
235	236	238	248	252	254	255	256	263	264
30.714079	-81.589041	-411.512956	-738.738192	1106.813164	1360.803564	-584.410430	-202.048244	429.054047	443.418801
265	271	275	288	290	291	292	297	299	300
216.588701	-410.374384	1297.573075	9.162089	207.499093	223.678329	523.347929	-579.895120	-719.142434	-621.191037
305	306	309	314	317	318	324	326	332	341
604.424909	581.633614	-137.474260	227.633150	728.239116	-388.703703	-249.086241	-118.587504	106.085663	-384.852582
344	345	347	348	349	354	356	357		
-268.686093	-166.527798	-135.553179	605.569747	591.924757	-14.810209	76.784183	13.886110		

Figure 8

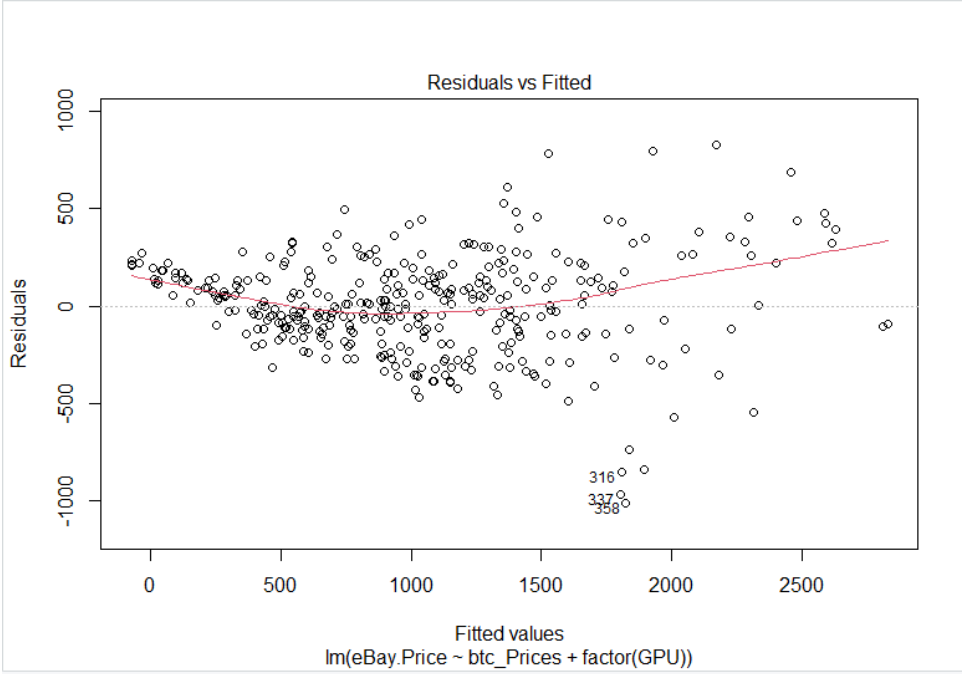

Figure 9

```
        Lilliefors (Kolmogorov-Smirnov) normality test

data:  ro2
D = 0.047328, p-value = 0.05138
```

Figure 10

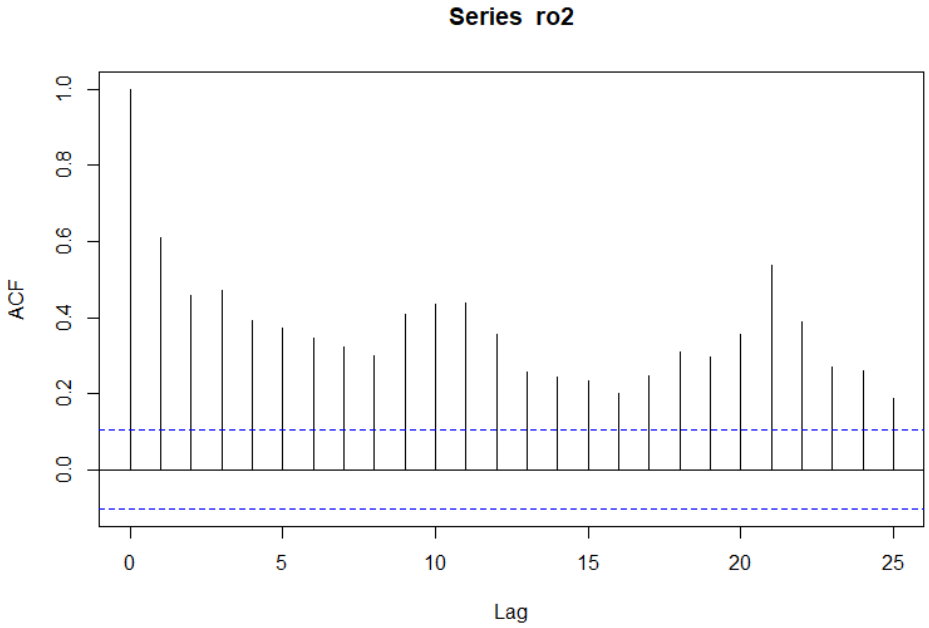

Figure 11

collect_gpuData.py - python script to collect GPU data from tables

```
"""
Created on Mon Nov 14 14:44:50 2022
@author: Jacne

"""
import pandas as pd
from bs4 import BeautifulSoup as bs
import requests
import csv
import time

url = 'https://www.tomshardware.com/news/gpu-pricing-index'

1. #Parse data from html website
       page = requests.get(url)
       headpg = bs(page.content,"html.parser")
       Headpg.title.text

2. #code to grab header data from tables
       headtable = headpg.find('table',{'class':'table__wrapper table__wrapper--
       inbodyContent table__wrapper--sticky table__wrapper--divider'})
       headers =[]
       for i in headtable.find_all('th'):
              col = i.text.strip()
              headers.append(col)
       headers = headers[:3]
       df = pd.DataFrame(columns = headers)
```

```
3. #paganation to scrape data from multiple pages
      max_pages = 24
      current_page = 1
      while current_page <= max_pages:
            current_url= f'{url}/{current_page}'
            html = requests.get(current_url)
            soup = bs(html.text,'html.parser')
            table = soup.find('table',{'class':'table__wrapper table__wrapper--
            inbodyContent table__wrapper--sticky table__wrapper--divider'})
            tTitle=soup.find('caption',{'class':'table__caption table__caption--top
            table__caption--left'})
            print(current_url)
            for row in table.find_all('tr')[1:]:#exclude header
                  data = row.find_all('td')[:3]
                  row_data = [td.text.strip() for td in data]
                  length = len(df)
                  df.loc[length] = row_data
            #time.sleep(5) #sleep for 5 seconds to prevent sending too many requests
            current_page+=1


4. #clean some things up
      df.rename(columns = {'Oct eBay Price':'eBay Price'},inplace = True)
      df=df.replace('N/A','nan')
5. #reverse the order
      df = df.iloc[::-1
6. #save as csv
      export_csv = df.to_csv(r'averageMonthlyGPUPrice_ebay.csv',index = None, header
= True)
```

BtcPrice_Time.py - mine more context from blockchain.com csv

```
"""
```

```
Created on Mon Nov 14 20:28:47 2022

@author: Jacne
"""
0. import numpy as n
import pandas as pd
import csv
import datetime
#read read in market-price.csv
btc = pd.read_csv(r'C:/User/market-price.csv',sep=',')


      date_time = []
      for i in btc['x']
            date_time.append(datetime.datetime.fromtimestamp(i/1000))
      date_time


1. #convert milliseconds to Seconds
      btc['x'] = btc['x']/1000
2. #rename columns to something more meaningful.
      btc.rename(columns = {'x':'Unix Epoch Time(Seconds)','y':'Price'},inplace =
      True)
3. #Insert DateTime into DataFrame
      btc.insert(
      loc = 0,
      column = 'DateTime',
      value = date_time)
4. #save as csv
      btc.to_csv(r'btc_mktPrice.csv',index = None, header = True)
```

DataClean.py - Read in raw data, clean it up.

```
      """
```

```
Created on Tue Nov 15 11:44:20 2022

@author: Jacne

"""

import numpy as np

import pandas as pd

from collections import OrderedDict

import csv

import datetime


0. btc = pd.read_csv(r'C:/yourpath/btc_mktPrice.csv',sep=',')

gpu = pd.read_csv(r'C:/yourpath/averageMonthlyGPUPrice_ebay.csv',sep=',')


0.a btc = btc.drop(btc[btc['Unix Epoch Time(Seconds)'] <= 1616198400].index)


1.#grab a sample of prices, build a dataframe from samples

       bsamp = btc.sample(n=362)

       bsamp['Unix Epoch Time(Seconds)'] = bsamp['Unix Epoch
       Time(Seconds)'].astype(int)

       bsamp['Price'] = bsamp['Price'].astype(float)

       bsamp = bsamp.reset_index(drop=True)

       bsamples = pd.DataFrame(bsamp)

       bsamples = bsamples.sort_values(by='Unix Epoch Time(Seconds)',ascending
       = True)

       bsamples = bsamples.reset_index(drop=True)


2.#not very efficient, works for now. Needs to be dynamic in the future.

#switch statement would be great, but python 3.9 does not have that...

       months = []

       for i in bsamples['DateTime']:

              if i.startswith('1/'):

              i='January'
```

```
months.append(i)
if i.startswith('2/'):
i='Febuary'
months.append(i)
if i.startswith('3/'):
i='March'
months.append(i)
if i.startswith('4/'):
i='April'
months.append(i)
if i.startswith('5/'):
i='May'
months.append(i)
if i.startswith('6/'):
i='June'
months.append(i)
if i.startswith('7/'):
i='July'
months.append(i)
if i.startswith('8/'):
i='August'
months.append(i)
if i.startswith('9/'):
i='September'
months.append(i)
if i.startswith('10/'):
i = 'October'
months.append(i)
if i.startswith('11/'):
i='November'
months.append(i)
```

```
            if i.startswith('12/'):
            i='December'
        months.append(i)
        bsamples['DateTime'] = months
        bsamples.rename(columns = {'Price':'btc_Prices','DateTime':'Month'
            ,'Unix Epoch Time(Seconds)':'DateTime'},inplace=True)


3.#conversions
        prices = []
        gpu['eBay Price'] = gpu['eBay Price'].astype(str)
        for i in gpu['eBay Price']:
        prices.append(float(i.strip('$').replace(',','')))
        gpu['eBay Price']= prices
        gpus = []
        for i in gpu['GPU']:
        gpus.append(i.strip('(opens in new tab) '))
        gpu['GPU'] = gpus
        gpu['QTY Sold'] = gpu['QTY Sold'].astype(float)


4.#values that need to be dropped
        gpu = gpu.dropna()
        #Drop Values that appear only once, b/c we can't predict them in our
        model.
        gpu = gpu[gpu.groupby('GPU').GPU.transform(len) > 1]
        gpu = gpu.reset_index(drop=True)
5.#mkt sample
        btc_gpu_samples = pd.concat([bsamples,gpu],axis = 1,join='inner')
        export_csv = btc_gpu_samples.to_csv(r'btc_gpu_mktSamples.csv',index =
        None, header = True)
```

btc_gpu_pricemodel.R - Forward Built MLR

```
#samples model - bit coin prices and GPU ebay prices were taken over the last 2~
years.
       Path<-"C:yourpath/btc_gpu_mktSamples.csv"
       mktsamples<-read.table(file=Path,sep=',',header = TRUE)
       attach(mktsamples)
       head(mktsamples)
       round(cor(mktsamples[c(-1,-2,-4)]),4)
       summary(mktsamples)
#GPU eBAY Price relative to bitcoin Price
       plot(eBay.Price~btc_Prices)
       abline(a=mean(eBay.Price),b=0,col="green")
#Linear Regression Model - Summarize data
       o1<-lm(eBay.Price~btc_Prices)
#Let's say alpha = .05. H_null: bit coin prices have no affect on gpu ebay prices
       summary(o1)
#p-value = 2.2e-16; smaller than any reasonable alpha. Reject Null hypothesis.
#we can only explain about ~24% of our eBay prices currently...
#Adjusted R-squared: ~.24
#what about our expected eBay prices if btc was 0?
#b0 = 1.425e+02 = 142.50
#add 1000 units
#142.50* 1000 = 142500
#can using factor(month) improve the model?
       f1<-lm(eBay.Price~btc_Prices+factor(Month))
       summary(f1)
#Multiple R-squared: ~.28,Adjusted R-squared: ~.26...
#relative to factor(GPU) it is not a good idea...
#can using factor(GPU) improve the model?
       f1<-lm(eBay.Price~btc_Prices+factor(GPU))
       summary(f1)
#Multiple R-squared: ~.8417,     Adjusted R-squared: .8313!!!
################################################################################
```

```
#forward building our model. Do we include QTY Sold?

#let's look at correlation again.

      round(cor(mktsamples[c(-1,-2,-4)]),4)

#QTY sold is not as strongly correlated as btc_price

#Will creating a more complex model, a model that includes QTY sold, be a better
predictor than our previous model?

      o2<-lm(eBay.Price~btc_Prices+QTY.Sold)

      summary(o2)

#Multiple R-squared: ~.24,Adjusted R-squared: 0.2388 ; bad in-comparison to o1

#but...

      f2<-lm(eBay.Price~btc_Prices+QTY.Sold+factor(GPU))

      summary(f2)

#Multiple R-squared: ~.8643,     Adjusted R-squared: ~.85; when we factor gpu, we
should include QTY sold.

#regression line that will help us predict values we have not observed

      plot(eBay.Price~btc_Prices)

      abline(a=lm(eBay.Price~btc_Prices+QTY.Sold+factor(GPU)),b=0,col='red')

      abline(a=mean(eBay.Price),b=0,col="green")

#our model is not a better predictor than the average value of dependent variable's
coefficient...

################################################################################

#create training and test sets to better understand our model

#do fitted values fall in between the lwr and upr boundaries of our predictions?

#run this first then the for loop. might result in error, that is ok, run the for
loop, then run pred again.

      pred<- predict(f2,newdata = test, interval = "prediction")

      K=50

      MAE = array(dim=c(K))

      MSE = array(dim=c(K))

      out = array(dim=c(K))

      for (i in 1:K){

            randrows<- sample(1:nrow(mktsamples),nrow(mktsamples)*(0.7))

            #randrows go into training, rest goes into test
```

```
            training<-mktsamples[randrows,] #70% of the data to represent training

            test<-mktsamples[-randrows,] #30% of the data to represent test

            attach(training)

            predict(f2,newdata = test

            #get eBay.Prices of test

            attach(test)

            #assuming we have a valid model, MAE will tell you how many errors on
            average

                  MAE[i]<-mean(abs(eBay.Price-predict(f2,newdata = test)))

                  MSE[i]<-mean(abs(eBay.Price-predict(f2,newdata = test))^2)

                  out[i]<-
                  sum((eBay.Price<pred[,2])|(eBay.Price>pred[,3]))/length(eBay.Pric
                  e)

            }
#data in the testing set already contains known values for the eBay.Price attribute,

#determine whether the model's guesses are correct.

#Boolean check to see how good our predictions are.

      pred

      pred[,1] - eBay.Price #predictions are not good, model sucks :(

#statistically insignificant cards cause our model to suffer...

################################################################################
#################

#Can the assumptions tell us why our model and data suck?

#Assumptions - Does our model violate any of the Linear Regression assumptions?

#predict and residual equations

      p1<-predict(lm(eBay.Price~btc_Prices))

      ro2<-residuals(lm(eBay.Price~btc_Prices))

      p1<-predict(lm(eBay.Price~btc_Prices+factor(GPU)))

      ro2<-residuals(lm(eBay.Price~btc_Prices+factor(GPU))

#residuals vs predicted, are the residuals' variance increasing/decreasing or
constant?

      plot(ro2~p1,xlab="predicted",ylab="Residuals"

      abline(a=mean(ro2),b=0,col='green')
```

```
#most data is constant, scattered randomly, constant variance.

#create residuals vs fitted

      plot(f1)

      plot(f2) #when we include QTY sold our residuals constant variance doesn't
      really differ from f1.

#does the model violate normality of residuals?

#Null hypothesis is that the residuals are normally distributed

      require(nortest)

      library(nortest)

      lillie.test(ro2)

#are the residuals autocorrelated?

      acf(ro2) #fail assumption.

#linear assumption - fails linear assumption

      plot(eBay.Price~btc_Prices)

      abline(a=lm(eBay.Price~btc_Prices+QTY.Sold+factor(GPU)),b=0,col='red')

      abline(a=mean(eBay.Price),b=0,col="green")
```