Konstantino Sparakis : U47131572
Shuwen Sun: U74918696
December 14, 2016
CS591/CS505

Final Project Report

GITHUB: https://github.com/AWS-Spot-Analysis/spot-analysis

Attached to the email are:
Sample data from api,
Sample data from archives,
And our working code in jupyter notebook

Abstract

This project aims to analysis the AWS EC2 spot instance marketplace. Throughout our work, we are trying to answer: What economic benefits can users get from using spot instances instead of regular EC2 Instances. We also aim to explore some work from research communities, and see if we can come up with similar results. The results and conclusions we have found can result in economic savings and better decision making when it comes to using using the AWS EC2 Spot Instance bid market.

Introduction

Our dataset is a time series of bid pricing for Amazon clouds EC2 Spot instances. EC2 is a service that allows you to purchase and use a custom virtual machine on an hourly basis that runs in a remote server and location, the cloud. EC2 virtual machines consist of different specs, allowing users to choose everything from ram, cpu's, ssd memory, OS and more options. Amazon also has multiple data centers throughout the world and these are considered regions.

What amazon introduced with Spot instances is that since not all their computing resources are being used up at any given moment. You can bid on these leftover machine resources starting at about 1/10th the original price. As the resources available is variable with decrease in supply, the demand goes up and so does the price of the resources. If you get outbid your instance is shutdown after 2 minutes.

Exploring the Datasets

Amazon has many different data centers located throughout the world. Each data center supports a subset of Instance types. Which are virtual machines with different specs, catering to the needs of users. Also each Instance type comes with a huge selection of operating systems, which are priced differently . For our purposes we will only be analyzing the basic linux operating system.

Example of locations and instance types in a table:

	1	2	3	4	5
Location	ap-northeast-2	ap-northeast-2	us-east-1	us-east-2	us-west-1
Instance Type	t2.nano	m1.medium	g2.8xlarge	r3.8xlarge	hi1.4xlarge

Spot Instance Price History

Data Date Range: *10/13/16 - 12/11/16, 59 days*
Total Number of Time Series: *1,525*
Method of obtaining: *AWS API*
Relevant Code in Github: *aws-data-grab.ipynb, from-aws-data-filter.py*

What is this data?

Spot Instance Price History data, is all pricing data for a specific Instance Type in a specific location for a time range. Each data point represents a change in pricing. This can happen every millisecond to once a day, truly depending on that markets activity. So what the data represents is a time series of pricing. Because there exists 61 instance types and 25 locations, the data we are working with is an array of 1,525 time series each representing an Instance Type in a specific location.

	TimeStamp	AvailabilityZone	InstanceType	SpotPrice
0	2016-12-12 10:44:30	ap-northeast-1a	t1.micro	0.0135
1	2016-12-12 10:44:30	ap-northeast-1a	t1.micro	0.0035
2	2016-12-12 10:43:50	ap-northeast-1a	t1.micro	0.0134

Deciding on the dataset and time range

Originally we had found an archived year's worth of data ranging from 2014-2015. But we ran into a few issues with this data. One of the issues was the sheer size of the data at about 74GB processing the data was too taxing for our local machines and we could not extract a subset as the data was not sorted. Along with this because the pricing of the actual ec2 instances change so frequently we would have had to normalized the data knowing the actual price of the ec2 instance but no archives with previous pricings could be found.

With all this in mind we decided that it was more advantageous for our analysis to use the most recent sixty days of data that we could obtain from AWS directly. Do note AWS only provides up to sixty days of their data.

Obtaining the Data

As for the actual data collection itself we wrote a script found in our github “data/Grab_aws_data.py” to automate this for us, as using the amazon command line tool was not sufficient.

Our script uses Boto a python library that connects to the AWS API. Having a list of all regions and machine types taken directly from the aws website, we then had the ability to iterate through and grab all the data. The issue came with the fact that it took way too long to collect data, our estimations say it would have easily taken us anywhere from 24-48+ hours, to grab data that was no larger than 6GB’s in total. So we paralyzed the job and were able to reduce the data collection to about 6 hours.

There is an AWS set limit to how many requests you can make per minute to their api, so we had to carefully balance the amount of threads in our parallelization to ten threads to

account for this, which was our main limitation and why it still took so long. Boto also has a limitation of fifty connection pools, which we found lookin at its open source code, but sadly we could not even take advantage of this. Even with our modest ten threads we ran into one instance where the download of one of the time series was timed out and we had to go back and download this one manually.

With the data downloaded in json format, our script converts it to a dataframe and then saves it as a csv file. Our parallelization meant we had a different csv files for every region and machine type combination. Which could easily be combined using a simple terminal command:

$ cat * > aws-recent-data.csv

Because of the concatenation of data the headers that we created in our previous code became an issue. So we had to write some code to delete all rows that had text where numbers should be which was rather simple. We also dropped some columns from the csv about irrelevant information, and we had to convert all the ISO8601 timestamps into python datetime timestamps using a custom method as the usual methods suggested were not working for us.

Another funny thing we found is that amazon doesn't specifically only provide 60 days of data. We were able to obtain 67 days, there seems to be a limit set more on the amount of data can be collected that the actual date range. Because of this we had to truncate our data to 59 days in order to insure we got rid of any time discrepancies between our time series.

Actual EC2 Spot Pricing

Relevant Code: *data/Grab_Actual_Pricing*
Method of obtaining: *Web Scraping with Beautifulsoup*
Date of Pricing Data: Last Update: *2016-12-14 16:23:48 UTC*

In order to make sense of EC2 Spot Instance prices, we needed the real hourly pricing of spot instances. This is where we had to obtain all actual prices for all regions and machine types. There are also different pricing models, with the On Demand price being the price the user pays for just spinning up a machine and paying hourly. But a user can also choose price plans, for example paying one or three year completely upfront, partially upfront and so on.

	AvailabilityZone	InstanceType	OnDemand	yrTerm1Standard.allUpfront	yrTerm1Standard.noUpfront
1	us-west-2	t1.micro	0.020	0.012	0.014
2	us-west-1	t1.micro	0.025	0.015	0.017
3	eu-west-1	t1.micro	0.020	0.015	0.016

Obtaining The Data

We discovered a website "http://www.ec2instances.info/" where they have scraped all price data from amazon and have added it into nice tables available for everyone. So we decided to scrape the data from them using python's BeautifulSoup. Conveniently the actually stored all the prices for one machine type in different locations in json right in the html. This made scraping the data very easy. Once we grabbed the data we turned it into csv format and saved into a file.

Cleaning

Not much cleaning was needed as the scraped data was obtained in a very structured format but since not all aws locations contain all types of machines we simply got rid of empty values.

Our Research

Relevant Code in Github: *spot_analysis_cr.ipnyb , spot_analysis_dtw.ipnyb*

Comparing EC2 Spot Prices with Real EC2 Pricing

Method: Analysing Mean, Median, Max, and Min

One of the first questions we wanted to answer was, If it is even worth using AWS Spot Instances. If they were really that cheap then why not just use the AWS Spot Instance forever instead of ever paying for other pricing options.

In order to answer this we compared the real price data with the Spot Instance Pricing data. We got the mean and max of every time series. And then subtracted these values from the actual pricing for that instance for that specific location. So here we end up with an array of the differences as such:

Mean Pricing Difference = Actual Price - Mean Spot Price of timeseries

Max Pricing Difference = Actual Price - Max Price of timeseries

We then got the Mean, Median, Min, and Max along with a positive value count and a negative value count for all the time series and created these tables:

Max Pricing Difference Table

Pricing	Mean	Median	Min	Max	# of Positive Values	# of Negative Values
On Demand	-5.769535	0.00184	-202.934	10.9348	531	766
1 Year Up Front	-6.625474	-0.22749	-209.735	2.7168	221	1076
3 Years Up Front	-7.044934	-0.49483	-214.462	1.05127	531	766

Mean Pricing Difference Table

Pricing	Mean	Median	Min	Max	# of Positive Values	# of Negative Values
On Demand	-0.42271	0.39690	-141.164	15.8683	986	311
1 Year Up Front	-1.28081	0.12496	146.775	7.65034	841	456
3 Years Up Front	-1.700269	0.00732	-149.798	2.02622	986	311

****Negative values means it's cheaper to use that option (Money Lost) Positive Values means it's cheaper to use Spot Instances (Money Saved)**

Analysis of the Mean Results Table

An analysis of the Mean results table actually tells us a lot of useful information. What the mean price of a timeseries tells us if we were to only use Spot Instances for these 60 days and then wanted to compute our total cost, we would just have to multiply our mean with the 60 days*24 hours. So if this mean price is cheaper than other plans then that means it actually makes no sense for that instance type in a specific location to use anything but the spot instance, as you can set your bid very high to ensure you don't get outbid and the probability of you paying more than any other payment plan is very low.

Looking at the number of positive values to negative values here shows us that for all three pricing plans there exists more markets where you can save money by using only spot instances then there exists markets where you would be losing money. Which is great news! As long as you know your instance is a good candidate there is no reason to pay actual prices and you should stick to spot instances. There is some bad news though looking at a comparison of min and max we see there is if we chose the wrong instance we could be losing about $150 dollars per hour worst case scenario while best case scenario we would only be saving $15 dollars an hour.

Another interesting fact is the fact looking at the number of negative values shows us that there are 311 marketplaces that you should avoid or tread cautiously because on average you will lose money in attempting to use there. In conclusion it is important to know the average price of your market before deciding to use spot instances.

Analysis of the Max Results Table

By observing the stats on the max price by hour subtracted to our real hourly prices we can answer an interesting question of whether the ec2 spot prices ever surpass the actual price. The answer is surprisingly yes, and it is very common happening to a large portion of the time series where it will once in awhile overpass its on demand pricing. This is interesting considering that you could just cut the spot instance and spin it up back on the on-demand pricing paying the cheaper price relatively easily, but depending on the job you are running perhaps companies can't afford to shut down the machine and end up sticking with them.

Another interesting fact that we learn here is that even if we only paid the max price for that spot instance time series there are still a large number of machines where you would be saving money still. That is shocking. We learn this by looking at the number of Positive Values in the Max results Table.

In Conclusion

In our analysis we primarily only look at the difference between the on demand pricing and the mean and max but our tables also include the pricing difference between 3 year upfront and 1 year up front pricing options. The 3 years upfront is the cheapest option offered by

amazon and since there exists spot instances that are even cheaper than this option there shows that in the spot instances there actually exists savings worth taking a look at.

Correlations

Methods: Pearson's Statistical Correlation, Dynamic Time Warping & Hierarchical clustering

Method: Pearson’s Statistical Correlations

The reason to find correlations is because they can be useful to tell you what marketplaces to avoid or go to when another market place is acting in a certain way. The reason we use pearson's statistical correlation is because it is the default method on pandas corr() function and accomplishes what we need. We get positive correlations telling us that these time series move together and we also get negative correlations telling us that these time series move in opposite directions. We can use these correlations to make predictions, and to even help us choose a market.

Normalizing the data for correlations:

One thing we did have to do was normalize the data. Each time series price changes at random, and that is when the price is recorded. In order to do a pearson's correlation we had to resample the data hourly averaging in order to be able to do the correlation. With this some information on price change activity is lost. On top of this we had to take the difference between each price value with the next one and then take the log. This allows us to better correlate price activity rather than actual prices. As now we can correlate really expensive machines with cheap ones.

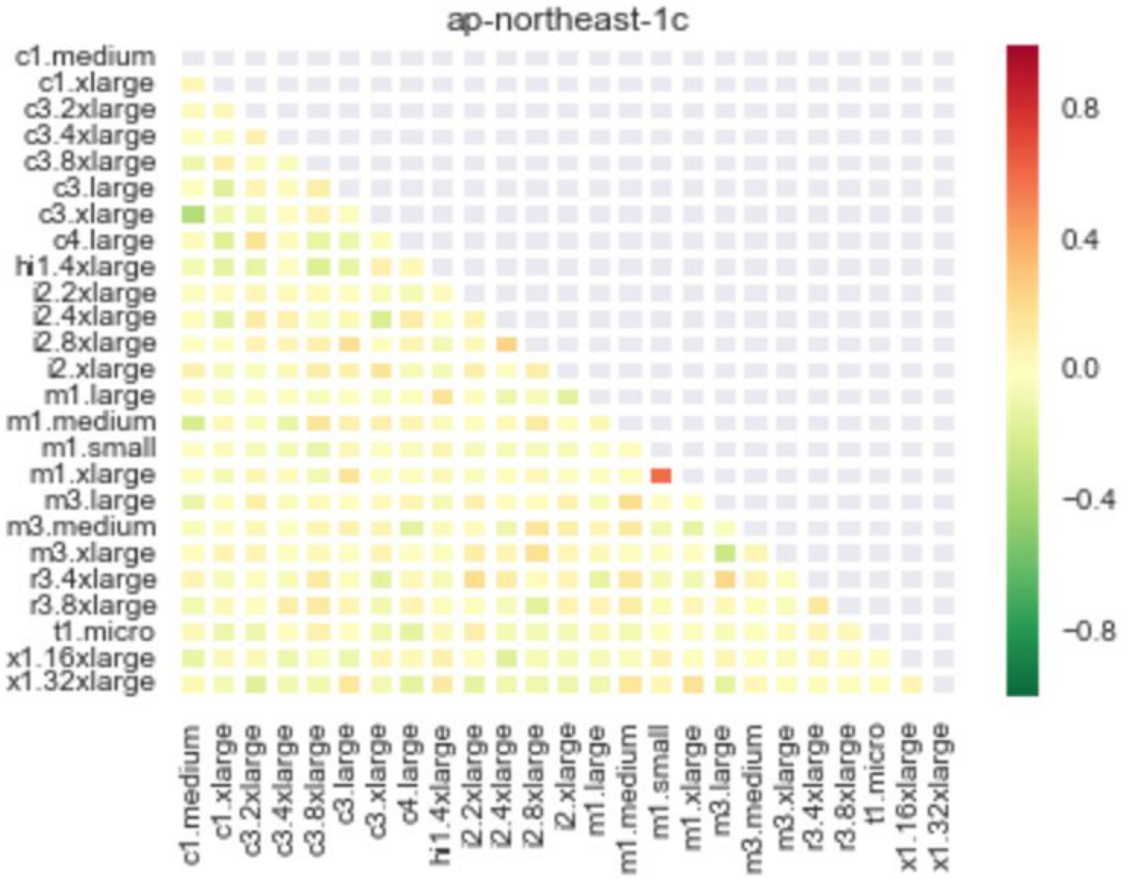
Correlating Instance Types Within a Region

Results & Analysis:
For all Heat maps and charts please refer to our github and look at **spot_analysis_corr.py** below are some classic examples of what we tend to see. There we have charted every combo of instance types and locations

At first we really assumed that there would be a lot of correlations here as the resources are shared so when the resources are low that means all market prices have to jump up. But to our surprise although some market places did have one or two correlation between instance types the majority of correlations were not strong enough to prove a good certainty. So perhaps even within one data center they are using different machines to support different virtual machine instance types. But since no information on infrastructure is available we can not answer this for

sure. These correlations could have been useful because if you see one machines price moving up within the same region you can guess you will get outbid soon on your correlated machine type, or if you have a negative machine you can assume your machines price will drop if another's is rising

Typical Example:

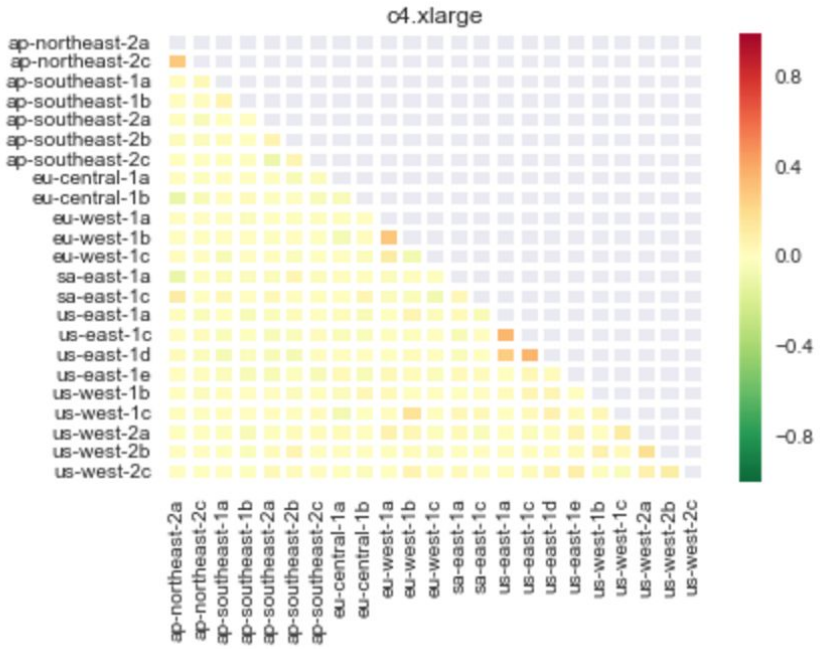


Correlating One Instance Type With itself in different regions.

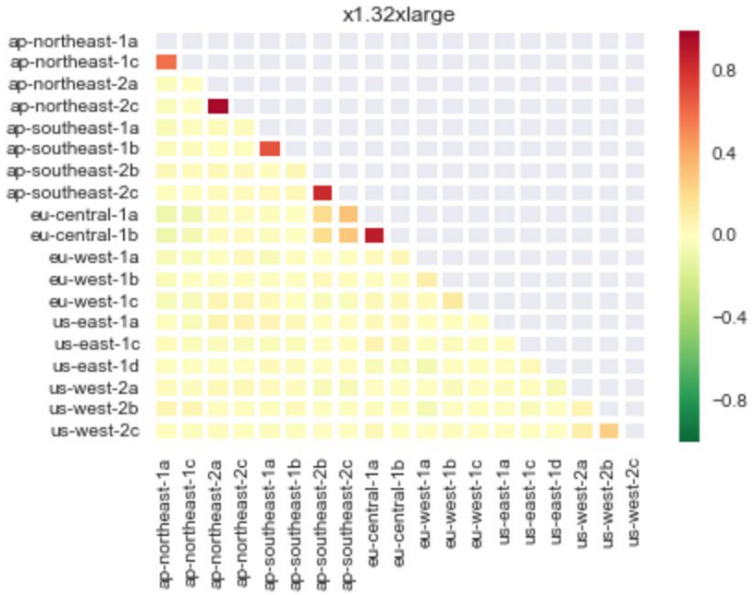
Results & Analysis:
For all Heat maps and charts please refer to our github and look at **spot_analysis_corr.py** below are some classic examples of what we tend to see. There we have charted every combo of instance types and locations

We found that the the same Instance in different market place, Is more often than not completely unrelated. This is great news! Because this lets you pick a location for your instance and know with a high certainty that even if another locations marketplace prices are getting extremely high your market will not be affected. This also means you can save money by moving your instance to a different region. This generated heat map is a typical example of what we see in our results:

Typical Results:

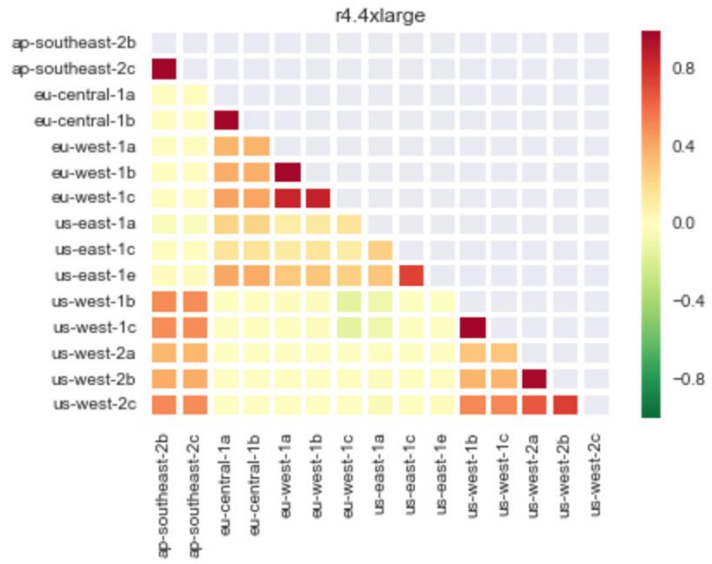


We did notice one interesting behavior as displayed by the following heat map.



Notice that there are correlations but they are between regions with the same name for example Eu-central-1a and EU-central-1b. Now you may have assumed that these data centers were named differently because they are completely separate, but with this information perhaps they are using the same resources? But we can not know this for sure as infrastructure data is not available. Or perhaps it shows that location of the instance is very important to the some customers and they need to stay within the region for whatever reason, and it's easy to pick between a 1a and 1b region.

Our machine type with the highest overall correlation was the r4.4xlarge but even then its correlations were about 0.4 showing that they are not even very statistically sound. Using correlation could have helped us in making predictions.

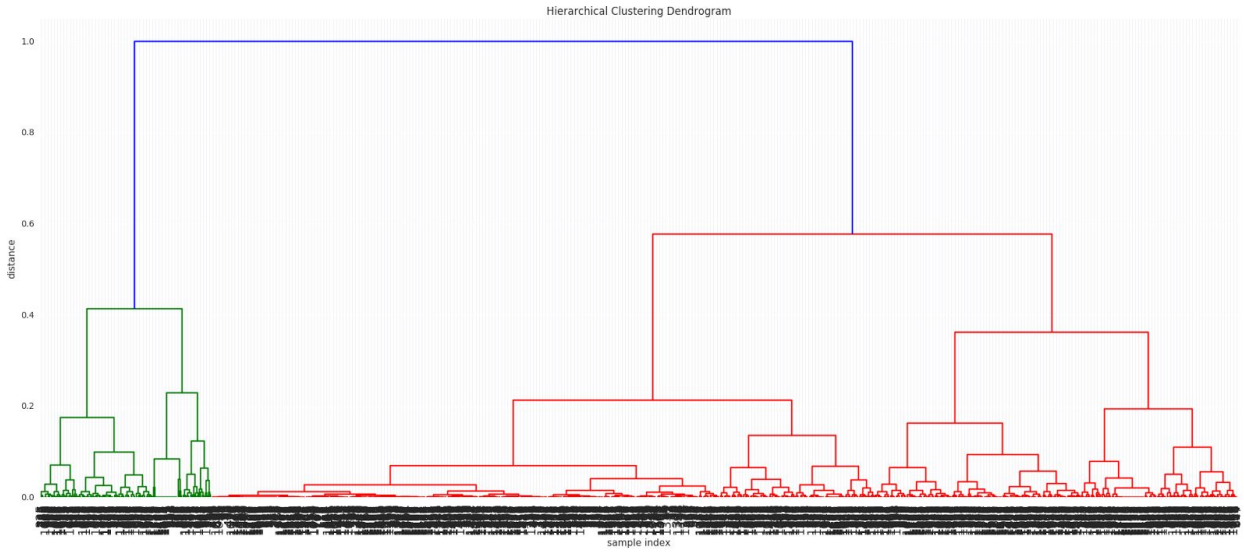


Method: Dynamic Time Warping & Hierarchical Clustering

Because we lose information of frequency of price change in the way we normalize the data using pearson's correlations, we wanted to try another method that would retain this information. And here we found the usage of Dynamic Time Warping (DTW). DTW compares two graphs and tries to move the graphs so that they best match each other. In other words giving us a distance between the two graphs. Using these distances we try\ to cluster them to see if we could get readable and useful results. This DTW value returned when small can tell us that the two time series are very similar and act as a correlator. One thing we did notice is that the official DTW algorithm is actually painfully slow, so we adopted a variation of it we found online known more for its speed. No python library exists that does DTW exists so we ended up spending a lot of time to just get this to work.

Results & Analysis:

Overall we were able to use DTW successfully and even use hierarchical clustering. The graph below is actually what the hierarchical clustering showed on dataset from two regions(ap-northeast-2a and ap-northeast-2c). And they clearly forms two clusters with similar structure in each clusters(small clusters demonstrate that they are various machine types).



Unfortunately though this became evident that the graph was not as useful as a heatmap would have been to display the same data and it really took us to the last minute to get DTW to work. So we were not able to get enough information to make any proper conclusions on the similarity of our time series that could affect any decisions in the market place.

Conclusion

In conclusion we consider our analysis successful and actually rather useful to the average customer. Although we were not able to complete a Dynamic Time Warping analysis on every time series pair to learn a more indepth information. We are confident that our pearson's correlation was rather accurate. Many research papers have stated that in fact AWS Spot instance markets are highly uncorrelated and we found the same results. Learning about how uncorrelated these marketplaces are can help people save money by simply moving their virtual machine to a different location or changing to a different similar spec instance type.

We were also to make some guesses about how amazon's infrastructure works thanks to our analysis of correlations showing that some different regions such as us-east-1a and us-east-1b might actually be the same from a datacenter perspective and something else is the reason they are named differently. Of course these are just speculations and as amazon keeps their information private we have no method of knowing.

Through our analysis of the means, median, mean and avg we were also able to learn that the Spot Instance Market is also a market worth investing in as there are instances where the savings are guaranteed and you should take advantage of that. But we also learned that you need to be carefull as there are instances where you could easily lose a lot of money if you don't manage your bidding correctly.

Overall this has made for a very interesting project and with many lessons learned we are glad to have come up with results that the public can use to help them make better educated bids to save money in their Spot instances.