```
from google.colab import drive
from google.colab import files
drive.mount("/content/drive/", force_remount=True)

     Mounted at /content/drive/
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="white", color_codes=True)
import warnings # current version of seaborn generates a bunch of warnings that we'll ignore
warnings.filterwarnings("ignore")
import matplotlib.ticker as ticker
```

HW1: http://webpages.csus.edu/fitzgerald/homework-streaming-large-text-file/

1 - subset

5 pts

Download a small subset of the data (100 rows is plenty) to your personal computer, and examine it using any software you like. Briefly describe this subset of the data by picking out a couple rows that look interesting to you.

1. How many columns are there?

> There are 61 columns in this sample dataset.

2. Do the data values in each column seem to match the column definitions?

> Sometimes.There are a large number of empty values for numerous columns and one article regarding President Trump and his administration had Actor2Code: COP and Actor2Name: DEPUTY when there is no mention of either in the article.

3. What character delimits the records?

> It seems TAB delimits the record.

4. What is the CAMEO event code, what event does this correspond to, and what is the Goldstein score?

> A raw CAMEO event code describes the action that Actor1 (person, place, etc.) performed upon Actor2 during an event. A Goldstein score is an assigned numberic score between -10 and +10, which tries to capture the potential impact that the type of event will have on the stability of a country for every CAMEO event code.

5. Are the URL's to the news articles still live, and do they match the CAMEO event code?

> Some URL's still work and others do not.

6. Does the Goldstein score appear to be doing what it was designed to do?

> It's hard to tell. There seems to be multiple duplicate articles and some of them do not have the same Goldstein scores. Such as the article which recieved both a Goldstein score of -9.2 **and** 2.8.

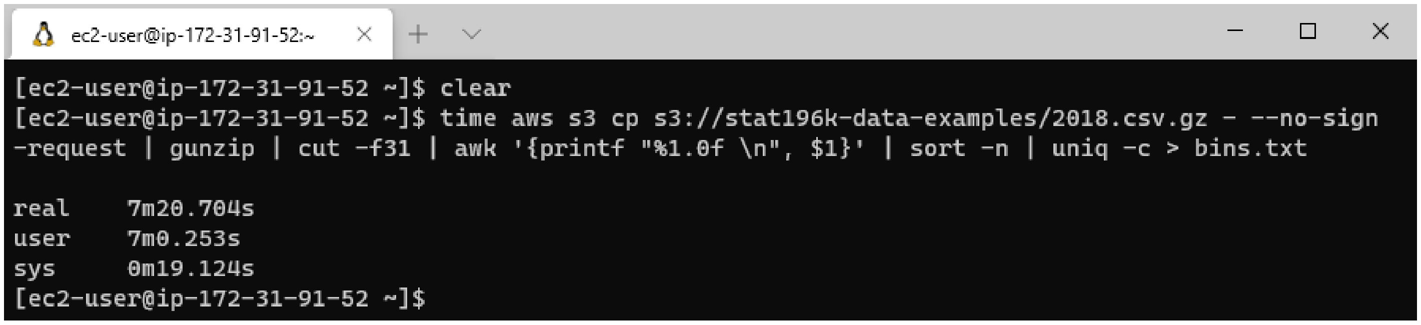
2 - histogram

10 pts

Create a histogram of the Goldstein scores for all of 2018, using the integers as bin endpoints for the histogram. It's possible to do this in less than 10 minutes using a single shell pipeline on a t2 micro instance with 1 vCPU, 1 GiB memory, and 8 GiB storage.

```
bins = pd.read_csv('/content/drive/My Drive/Sac State/STAT196K/HW1/bins.csv')
```

1. How long does your program take to run?



```
[ec2-user@ip-172-31-91-52 ~]$ clear
[ec2-user@ip-172-31-91-52 ~]$ time aws s3 cp s3://stat196k-data-examples/2018.csv.gz - --no-sign
-request | gunzip | cut -f31 | awk '{printf "%1.0f \n", $1}' | sort -n | uniq -c > bins.txt

real    7m20.704s
user    7m0.253s
sys     0m19.124s
[ec2-user@ip-172-31-91-52 ~]$
```

2. Explain in detail what each command in the pipeline does and how they work together.

> **time**: Times how long it takes for a given script or command to run.
>
> **aws s3 cp**: Copy file from S3 storage to my EC2 instance.
>
> **gunzip**: unzip the file from S3 storage.
>
> **cut -f31**: Filter to the 31th column of the file.
>
> **awk '{printf "%1.0f \n", $1}'**: Round the values to the nearest whole number.
>
> **sort -n**: Sort the values numerically in ascending order.
>
> **uniq -c**: Count the number of times a line was repeated.

My pipeline consists of six commands that serve as input of another command starting with **copying** the 2018.csv.gz file from AWS S3 storage. Then it will **unzip** the file so it can be **cut** in order to filter the file to a specific column field for the Goldstein score. Afterwards, the float values will be **rounded** then **sorted** in ascending order. Finally, it will **count** the number of times a value was repeated and print the results as a .txt file.

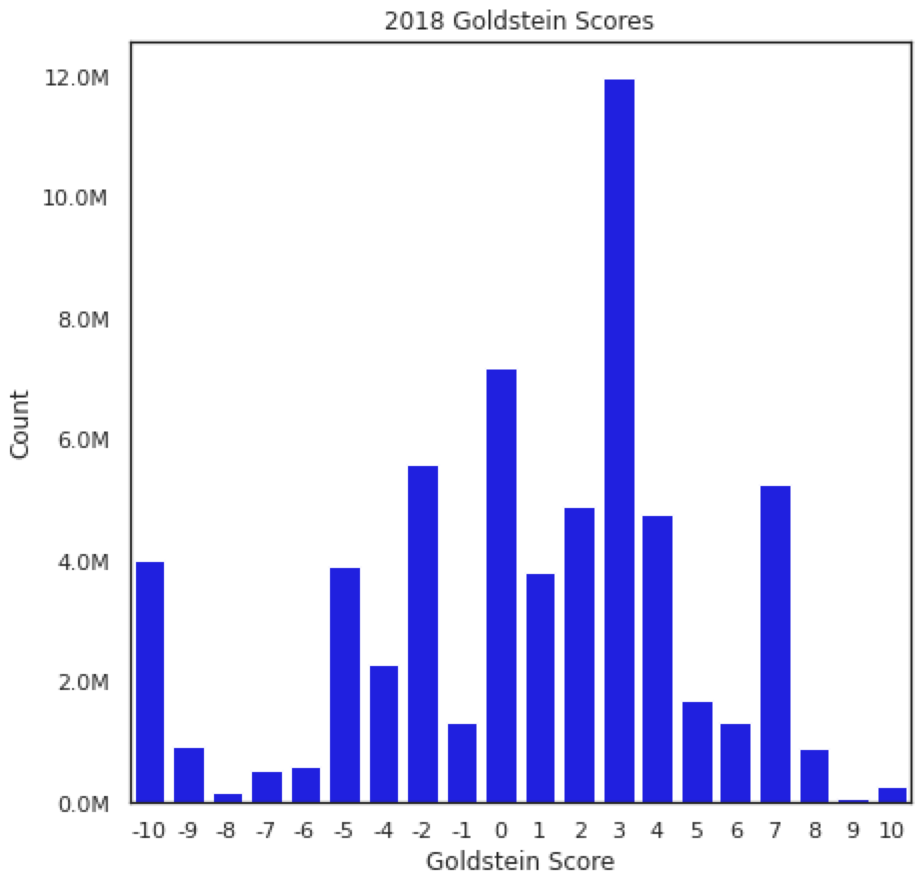
3. Plot and interpret the histogram. You'll probably want to download the summary statistics (around 20 numbers) to your personal computer to plot the histogram. Do you notice anything strange?

> The first thing that is strange to me is that +3 scores are the most frequent but there are no scores for -3. The second most frequent score is 0, what is the meaning of a 0 score - unrelated news? For scores -10 and +10, there are significantly more -10 scores than there are +10 scores. Since the scores are supposed to represent the potential impact an event has on the stability of a country, how come it appears more common for maximum negative impact scores to occur significanly more often than maximum positive impacts?

```
def millions(x, pos):
    'The two args are the value and ticker position'
    return '%1.1fM' % (x*1e-6)


fig = plt.figure(figsize=(7,7))
ax = sns.barplot(data=bins, x="score", y="count", color="blue")
ax.set_title('2018 Goldstein Scores')
ax.set_xlabel('Goldstein Score')
```

```
    ax.set_xlabel('Goldstein Score')
    ax.set_ylabel('Count')
    ax.yaxis.set_major_formatter(ticker.FuncFormatter(millions));
    plt.show()
```



4. Exactly how many events (rows) are in this data?

```
print('There are %s events(rows) in this data' % bins['count'].sum())
```

```
    There are 61544481 events(rows) in this data
```

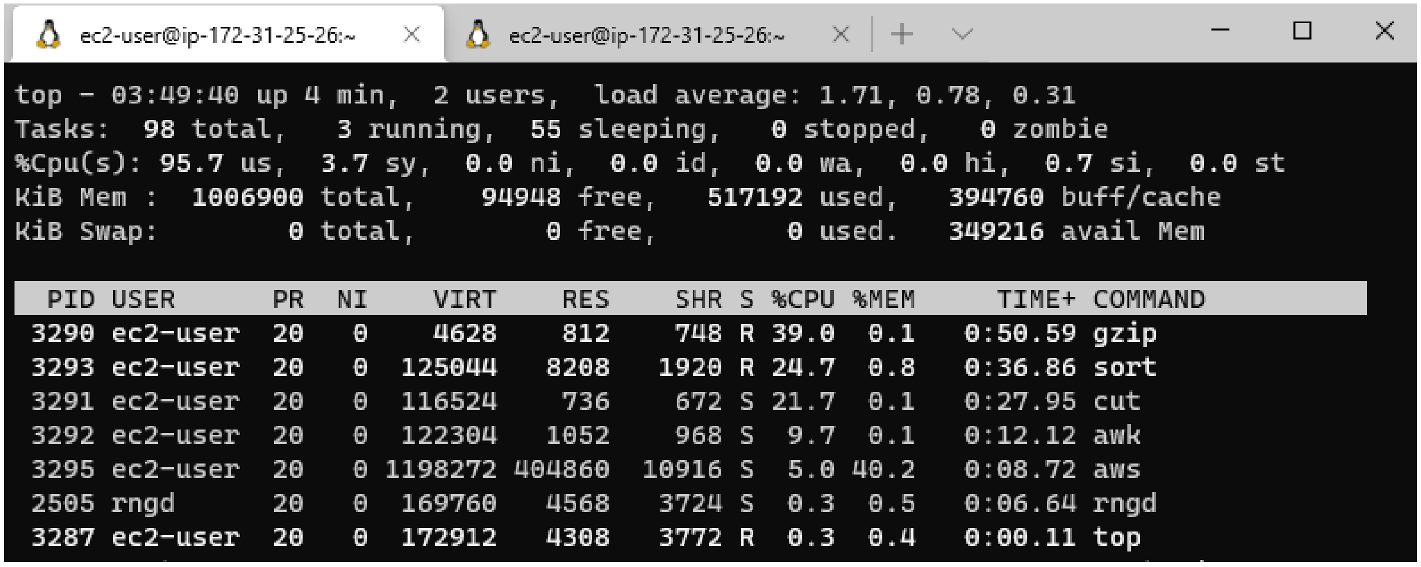
3 - performance

5 pts

Print and interpret the output of **top** while your program is running.

1. What are the bottlenecks? **t2.micro**

> The main bottleneck is gzip(gunzip), which is understandable given how large the file is.

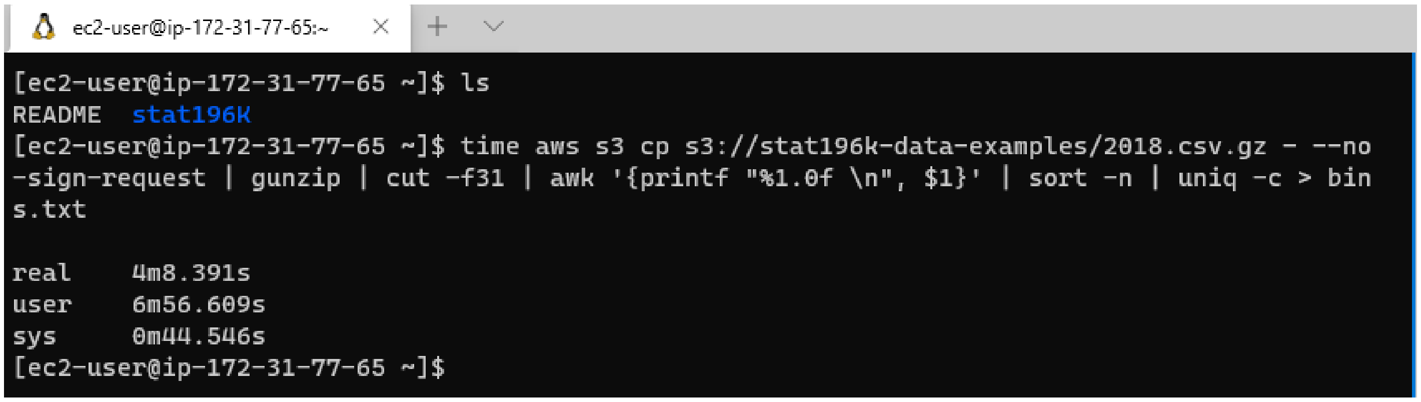


```
top - 03:49:40 up 4 min,  2 users,  load average: 1.71, 0.78, 0.31
Tasks:  98 total,   3 running,  55 sleeping,   0 stopped,   0 zombie
%Cpu(s): 95.7 us,  3.7 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.7 si,  0.0 st
KiB Mem :  1006900 total,    94948 free,   517192 used,   394760 buff/cache
KiB Swap:        0 total,        0 free,        0 used.   349216 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 3290 ec2-user  20   0    4628    812    748 R 39.0  0.1   0:50.59 gzip
 3293 ec2-user  20   0  125044   8208   1920 R 24.7  0.8   0:36.86 sort
 3291 ec2-user  20   0  116524    736    672 S 21.7  0.1   0:27.95 cut
 3292 ec2-user  20   0  122304   1052    968 S  9.7  0.1   0:12.12 awk
 3295 ec2-user  20   0 1198272 404860  10916 S  5.0 40.2   0:08.72 aws
 2505 rngd      20   0  169760   4568   3724 S  0.3  0.5   0:06.64 rngd
 3287 ec2-user  20   0  172912   4308   3772 R  0.3  0.4   0:00.11 top
```

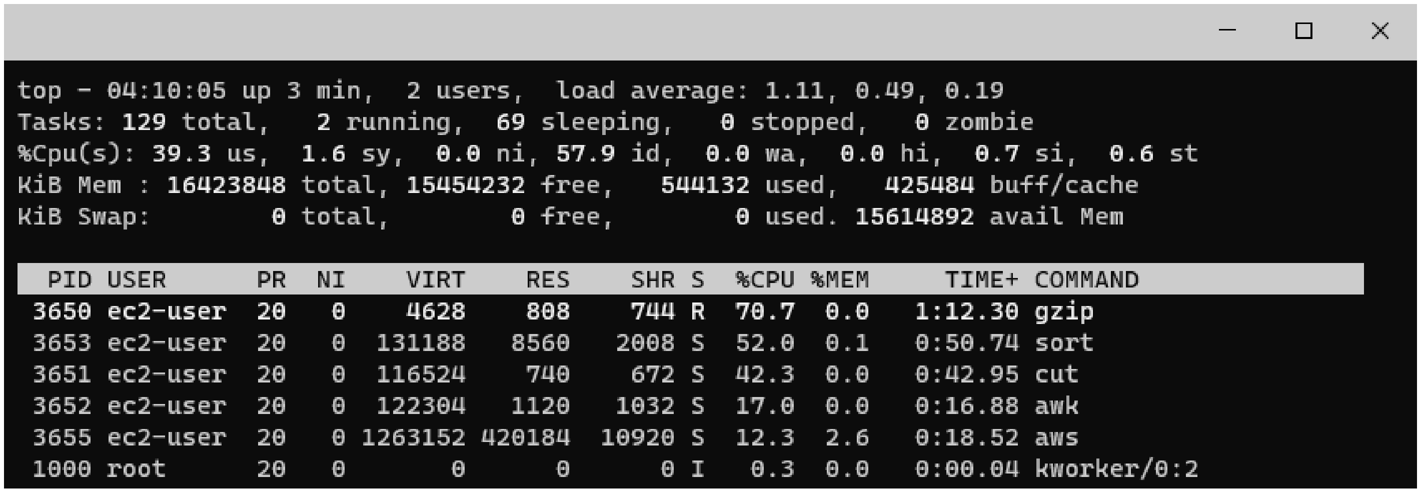
2. Run and time your program on an EC2 instance with more vCPU's and a faster network and show the results of **top** once more. Is the program faster on the more expensive instance? **t2.xlarge**

> Yes! The t2.xlarge instance was much faster (x1.7) than the t2.micro instance.



```
[ec2-user@ip-172-31-77-65 ~]$ ls
README  stat196K
[ec2-user@ip-172-31-77-65 ~]$ time aws s3 cp s3://stat196k-data-examples/2018.csv.gz - --no
-sign-request | gunzip | cut -f31 | awk '{printf "%1.0f \n", $1}' | sort -n | uniq -c > bin
s.txt

real    4m8.391s
user    6m56.609s
sys     0m44.546s
[ec2-user@ip-172-31-77-65 ~]$
```



```
top - 04:10:05 up 3 min,  2 users,  load average: 1.11, 0.49, 0.19
Tasks: 129 total,   2 running,  69 sleeping,   0 stopped,   0 zombie
%Cpu(s): 39.3 us,  1.6 sy,  0.0 ni, 57.9 id,  0.0 wa,  0.0 hi,  0.7 si,  0.6 st
KiB Mem : 16423848 total, 15454232 free,   544132 used,   425484 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 15614892 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 3650 ec2-user  20   0    4628    808    744 R  70.7  0.0   1:12.30 gzip
 3653 ec2-user  20   0  131188   8560   2008 S  52.0  0.1   0:50.74 sort
 3651 ec2-user  20   0  116524    740    672 S  42.3  0.0   0:42.95 cut
 3652 ec2-user  20   0  122304   1120   1032 S  17.0  0.0   0:16.88 awk
 3655 ec2-user  20   0 1263152 420184  10920 S  12.3  2.6   0:18.52 aws
 1000 root      20   0       0      0      0 I   0.3  0.0   0:00.04 kworker/0:2
```

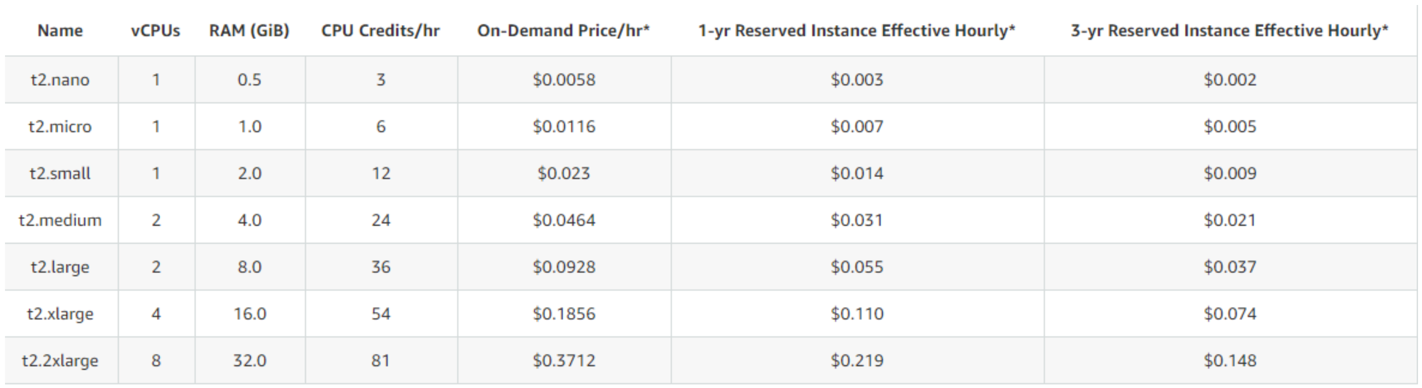
3. Are you benefitting from pipeline parallelism?

> Although it is not optimized given that t2.xlarge's max mapacity 400%CPU, I am still benefiting from some pipeline parallelism because by adding up the %CPU of my commands - it amounts to 194%.

4. What’s the bottleneck now? **t2.xlarge**

> The bottleneck for the t2.xlarge instance is still the same as the t2.micro instance, which is gzip(gunzip). Which makes sense because I need the data from gzip to use my remaining commands.

5. Compare and comment on the financial cost of using a more expensive instance versus the t2.micro. Is it worth it?

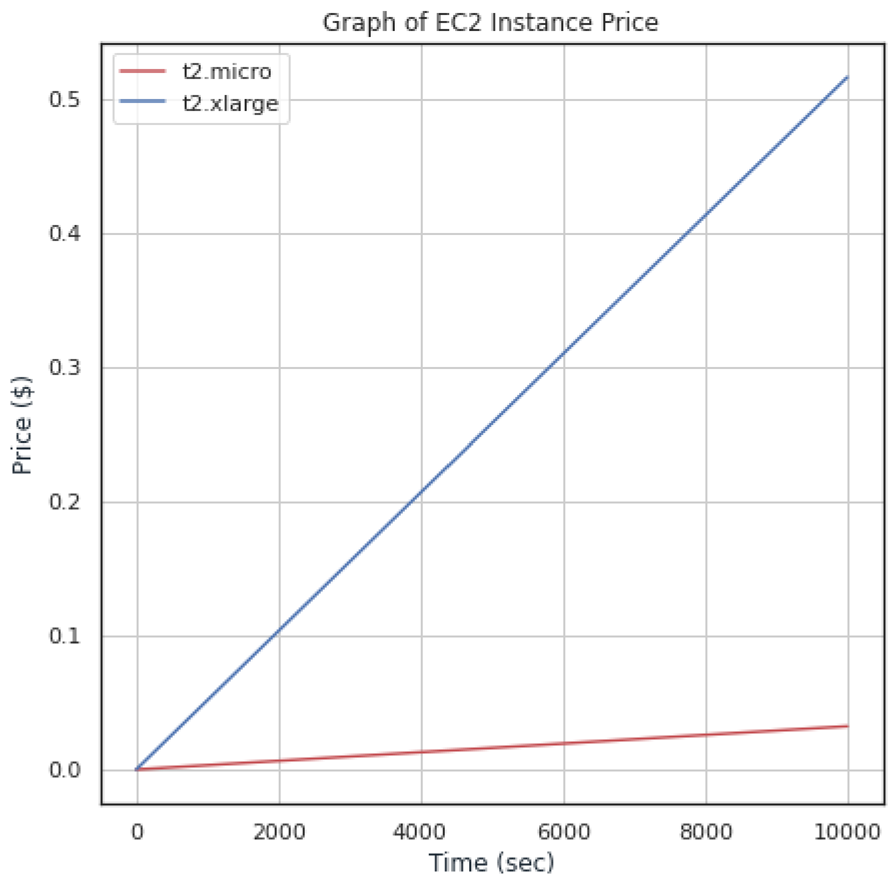


Name	vCPUs	RAM (GiB)	CPU Credits/hr	On-Demand Price/hr*	1-yr Reserved Instance Effective Hourly*	3-yr Reserved Instance Effective Hourly*
t2.nano	1	0.5	3	$0.0058	$0.003	$0.002
t2.micro	1	1.0	6	$0.0116	$0.007	$0.005
t2.small	1	2.0	12	$0.023	$0.014	$0.009
t2.medium	2	4.0	24	$0.0464	$0.031	$0.021
t2.large	2	8.0	36	$0.0928	$0.055	$0.037
t2.xlarge	4	16.0	54	$0.1856	$0.110	$0.074
t2.2xlarge	8	32.0	81	$0.3712	$0.219	$0.148

> While the price of the t2.xlarge is **relatively** larger than the cost of using the t2.instance, the higher cost is negliable compared to the time that is saved based on the results of my pipeline (248 sec vs 422 sec), the t2.xlarge is **1.7** times faster. Comparatively, for a process that would take 10,000 seconds (2.7 hrs) to run on t2.micro - the amount of time saved by using t2.xlarge would be 5,882 seconds (1.6 hrs) while the amount of money saved by using t2.micro would be around $0.20. Thus, it is worth using t2.xlarge over t2.micro instances because the amount of time saved outweighs the cost.

```
fig = plt.figure(figsize=(7,7))
x = np.linspace(0,10000)
plt.plot(x, 0.0116*(x/3600), '-r', label='t2.micro')
plt.plot(x, 0.1856*(x/3600), '-b', label='t2.xlarge')
plt.title('Graph of EC2 Instance Price')
```

```
    plt.title('Graph of EC2 Instance Price')
    plt.xlabel('Time (sec)', color='#1C2833')
    plt.ylabel('Price ($)', color='#1C2833')
    plt.legend(loc='upper left')
    plt.grid()
    plt.show()
```


Graph of EC2 Instance Price
t2.micro
t2.xlarge
Price ($)
Time (sec)