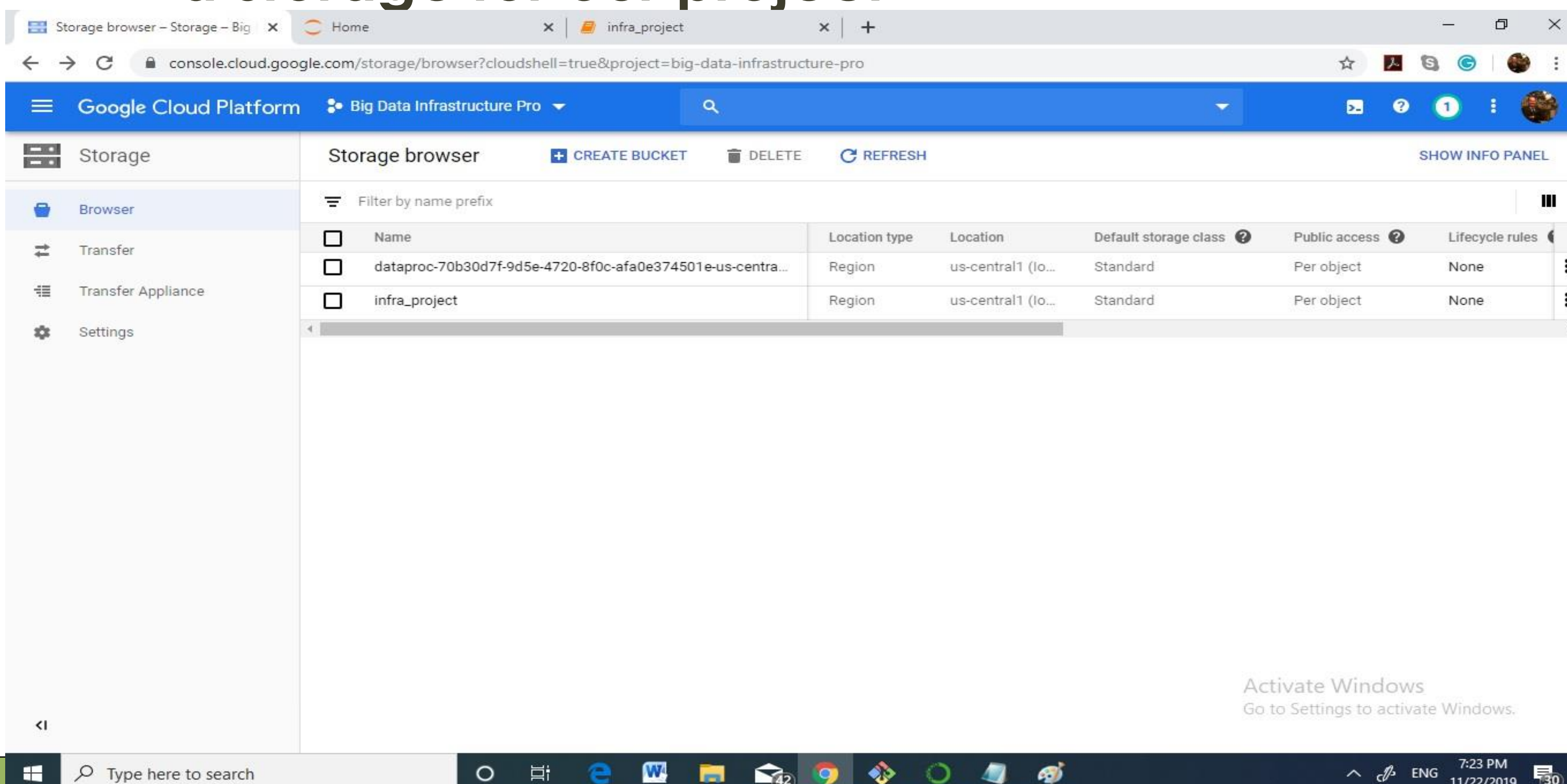
Monthly prediction for Bixi rides in Montreal

The final project for big data infrastructure course by
Hamid Reza Taremian

Setting up Google cloud platform

- 1. First we need to set up a bucket to have a storage for our project



- **2. Then we need to make a cluster for our project using command shell which will allow jupyter notebook using the followinf command:**
- gcloud beta dataproc clusters create infra_projcet

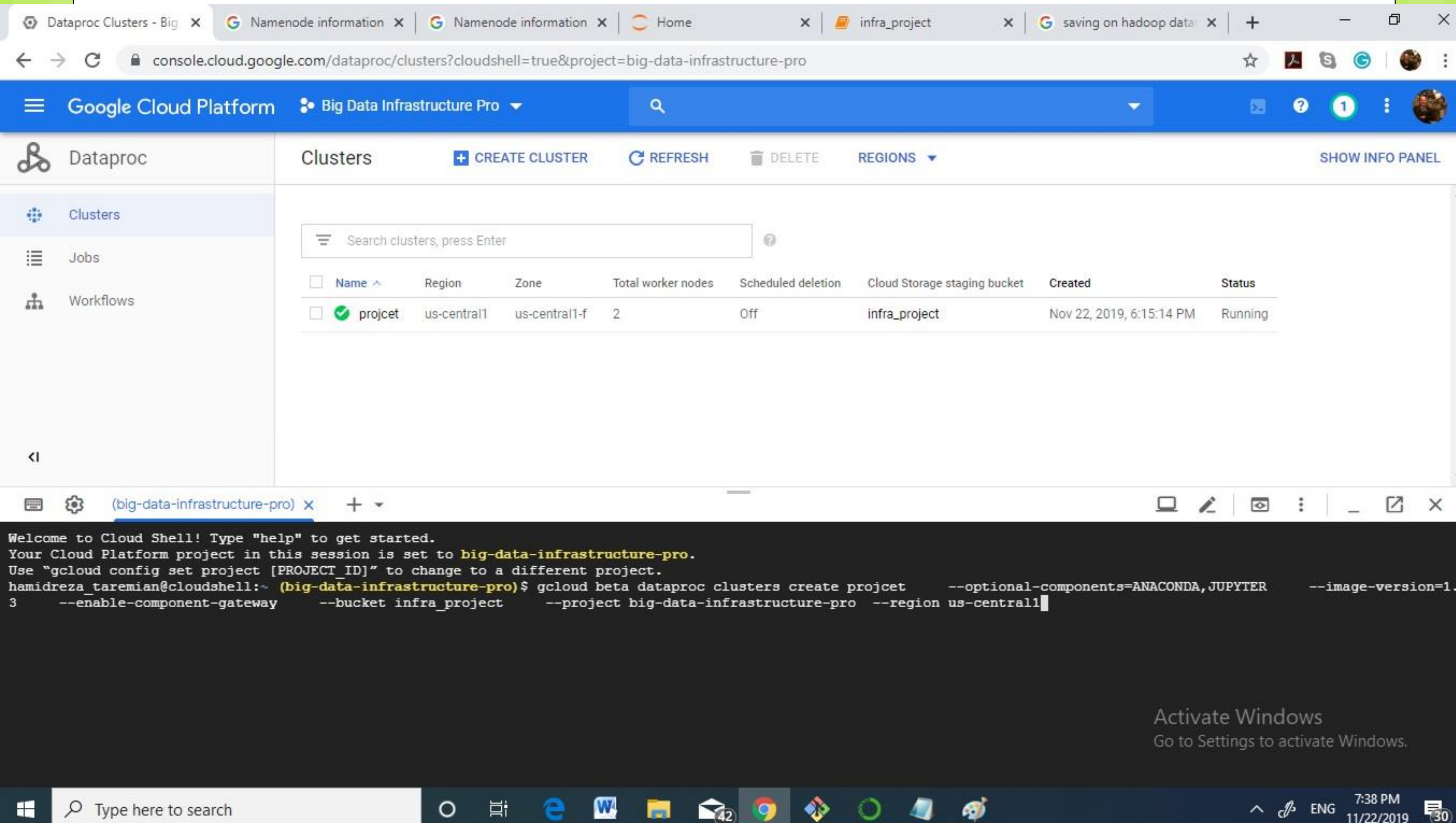
\ --optional-components=ANACONDA,JUPYTER

\ --image-version=1.3

\ --enable-component-gateway

\ --bucket infra_project

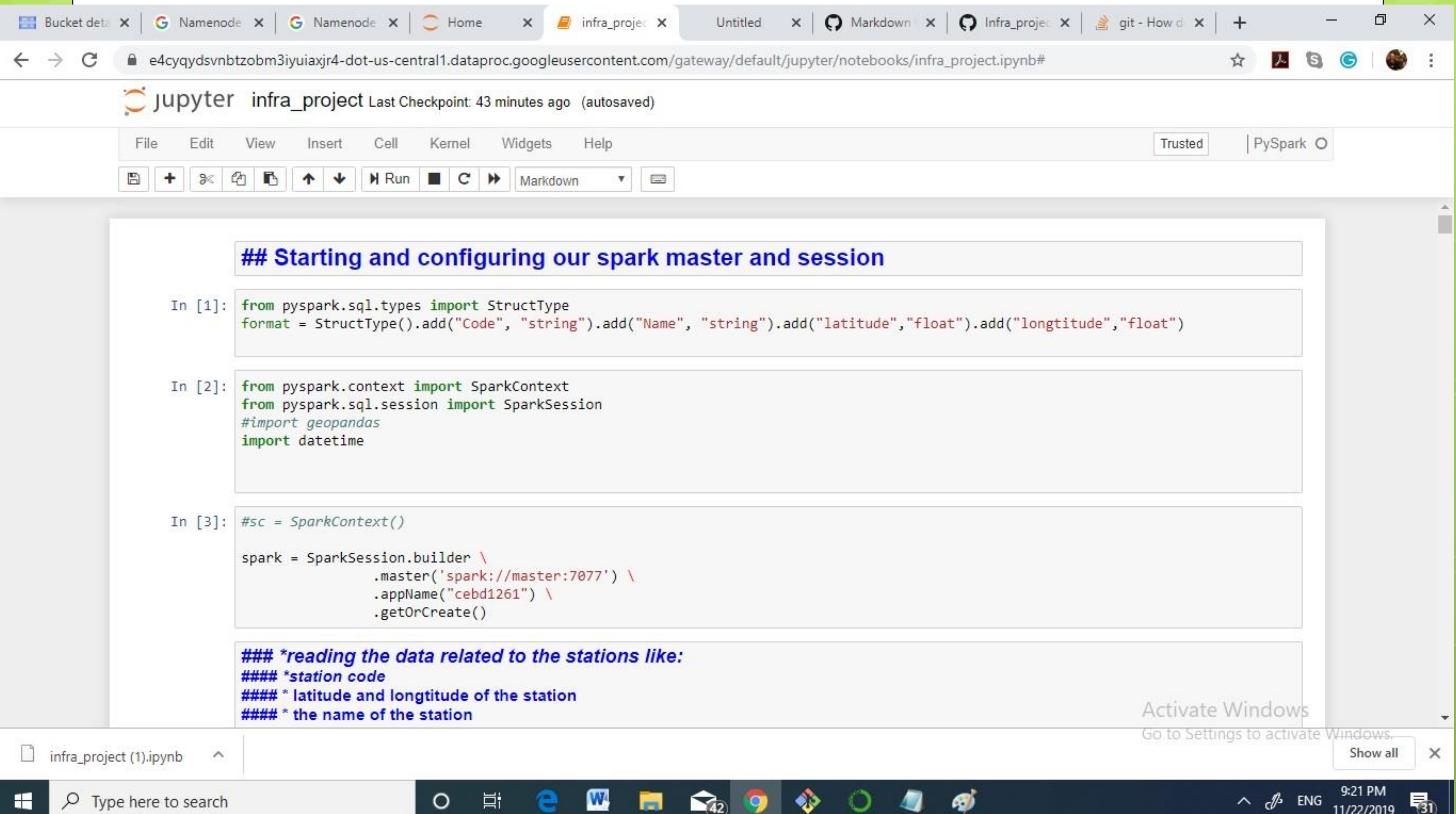
\ --project big-data-infrastructure-pro


Dataproc Clusters - Big
Namenode information
Namenode information
Home
infra_project
saving on hadoop data
console.cloud.google.com/dataproc/clusters?cloudshell=true&project=big-data-infrastructure-pro
Google Cloud Platform
Big Data Infrastructure Pro
Dataproc
Clusters
CREATE CLUSTER
REFRESH
DELETE
REGIONS
SHOW INFO PANEL
Clusters
Jobs
Workflows
Search clusters, press Enter
Name
Region
Zone
Total worker nodes
Scheduled deletion
Cloud Storage staging bucket
Created
Status
projcet
us-central1
us-central1-f
2
Off
infra_project
Nov 22, 2019, 6:15:14 PM
Running
(big-data-infrastructure-pro)
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to big-data-infrastructure-pro.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
hamidreza_taremian@cloudshell:~ (big-data-infrastructure-pro)$ gcloud beta dataproc clusters create projcet --optional-components=ANACONDA,JUPYTER --image-version=1.
3 --enable-component-gateway --bucket infra_project --project big-data-infrastructure-pro --region us-central1
Activate Windows
Go to Settings to activate Windows.
Type here to search
ENG
7:38 PM
11/22/2019

Some Analysis

- we can see how many rides were done from each station and link the station geographical information for further usage.

- 3. Now we can have access to jupyter notebook to write our code



Starting and configuring our spark master and session

In [1]:
```
from pyspark.sql.types import StructType
format = StructType().add("Code", "string").add("Name", "string").add("latitude","float").add("longtitude","float")
```

In [2]:
```
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
#import geopandas
import datetime
```

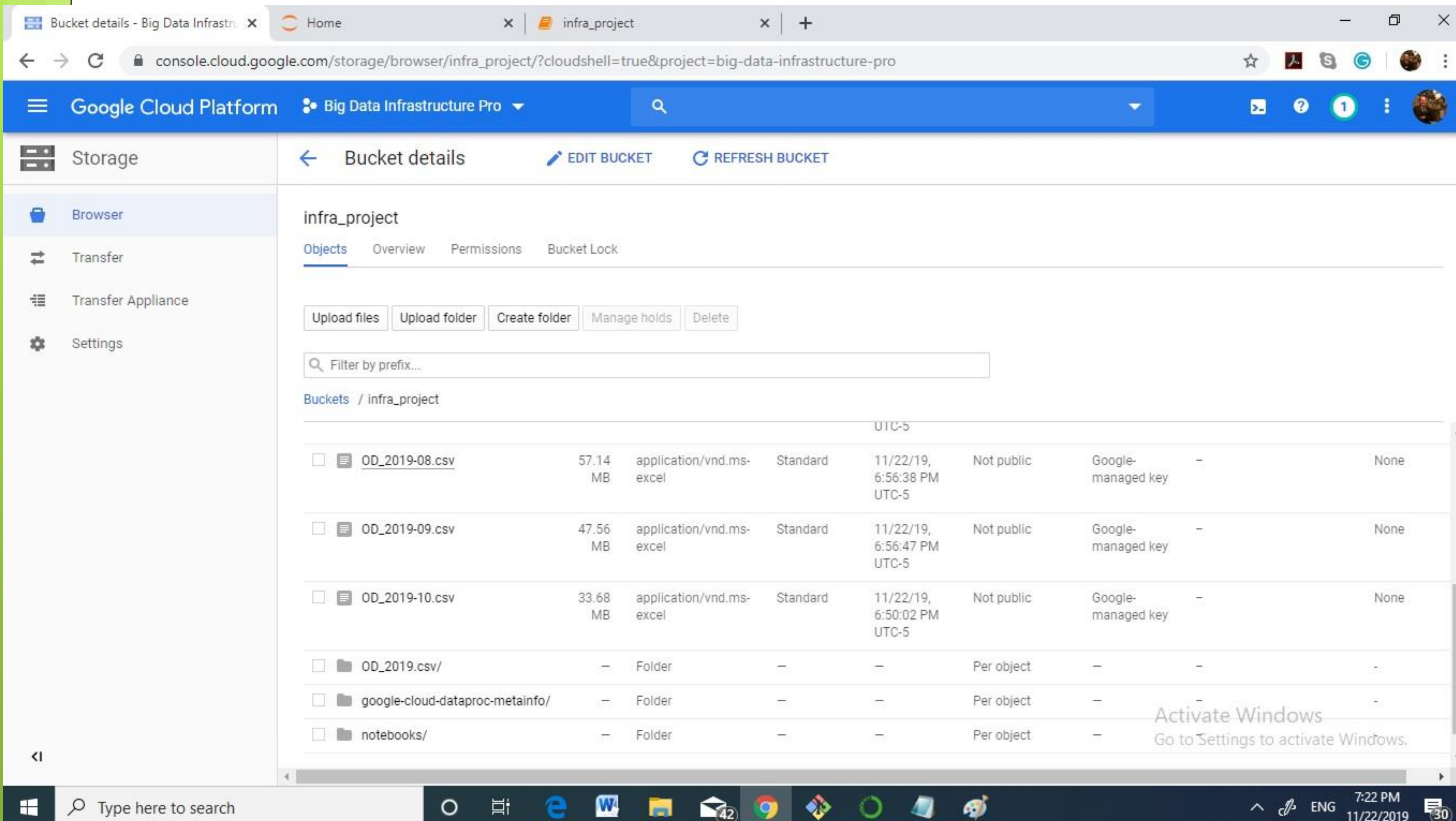
In [3]:
```
#sc = SparkContext()

spark = SparkSession.builder \
            .master('spark://master:7077') \
            .appName("cebd1261") \
            .getOrCreate()
```

*reading the data related to the stations like:
*station code
* latitude and longtitude of the station
* the name of the station

The python+spark code

- our data consists of one CSV file for the information related to the each station and CSV files for monthly rides. so first we upload our files into the bucket that we made before.


Bucket details - Big Data Infrastr
Home
infra_project
console.cloud.google.com/storage/browser/infra_project/?cloudshell=true&project=big-data-infrastructure-pro
Google Cloud Platform
Big Data Infrastructure Pro
Storage
Browser
Transfer
Transfer Appliance
Settings
Bucket details
EDIT BUCKET
REFRESH BUCKET
infra_project
Objects
Overview
Permissions
Bucket Lock
Upload files
Upload folder
Create folder
Manage holds
Delete
Filter by prefix...
Buckets / infra_project
OD_2019-08.csv 57.14 MB application/vnd.ms-excel Standard 11/22/19, 6:56:38 PM UTC-5 Not public Google-managed key – None
OD_2019-09.csv 47.56 MB application/vnd.ms-excel Standard 11/22/19, 6:56:47 PM UTC-5 Not public Google-managed key – None
OD_2019-10.csv 33.68 MB application/vnd.ms-excel Standard 11/22/19, 6:50:02 PM UTC-5 Not public Google-managed key – None
OD_2019.csv/ – Folder – – Per object – –
google-cloud-dataproc-metainfo/ – Folder – – Per object – –
notebooks/ – Folder – – Per object – –
Activate Windows
Go to Settings to activate Windows.
Type here to search
ENG
7:22 PM
11/22/2019

In [7]:
```
station_cnt = df_2019_10.groupBy('start_station_code').agg({'start_station_code':'count'})
station_cnt.show()
```

```
+------------------+------------------------+
|start_station_code|count(start_station_code)|
+------------------+------------------------+
|              6194|                    2301|
|              6731|                     271|
|              6240|                     383|
|              6248|                    3293|
|              6366|                     537|
|              6903|                    2108|
|              7056|                     301|
|              6081|                     358|
|              6227|                    2535|
|              7054|                     696|
|              6380|                     547|
|              6106|                     520|
|              6732|                     608|
|              6143|                    1911|
|              6252|                    1152|
|              6402|                     722|
|              7014|                    1635|
|              7013|                     375|
|              6033|                     424|
|              7093|                      42|
+------------------+------------------------+
only showing top 20 rows
```

- we can read the data for stations and each month separately.

In [4]:
```
df_station = spark.read.csv('gs://infra_project/Stations_2019.csv',schema=format, header="true")
df_station.show(5)
```

```
+-----+--------------------+---------+----------+
| Code|                Name| latitude|longtitude|
+-----+--------------------+---------+----------+
|10002|Métro Charlevoix ...| 45.47823| -73.56965|
| 4000|Jeanne-d'Arc / On...|  45.5496| -73.54188|
| 4001|  Graham / Brookfield|45.520073|-73.629776|
| 4002|   Graham / Wicksteed|45.516937| -73.64048|
| 5002|St-Charles / Mont...|45.533684| -73.51526|
+-----+--------------------+---------+----------+
only showing top 5 rows
```

Code	Name	latitude	longtitude	October count
6194	Métro Atwater (At...	45.489475	-73.584564	2301
6731	28e avenue / Rose...	45.564354	-73.57124	271
6240	Parc Kent (de Ken...	45.505722	-73.629456	383
6248	St-Dominique / Ra...	45.518593	-73.581566	3293
6366	Wilderton / Van ...	45.510143	-73.62475	537
6903	St-Dominique / Na...	45.516663	-73.57722	2108
7056	Bibliothèque de V...	45.44826	-73.57786	301
6081	Mackay / Ste-Cath...	45.49571	-73.57695	358
6227	de l'Esplanade / ...	45.521038	-73.59491	2535
7054	de la Côte St-Pau...	45.467667	-73.59392	696
6380	Parc J.-Arthur-Ch...	45.551582	-73.56191	547
6106	Papineau / René-L...	45.52114	-73.54926	520
6732	Fortune / Wellington	45.477924	-73.55904	608
6143	Rachel / de Brébeuf	45.52689	-73.57264	1911
6252	Mozart / St-Laurent	45.53318	-73.61544	1152
6402	Ste-Émilie / Sir-...	45.472668	-73.58539	722
7014	Métro Université ...	45.504276	-73.61797	1635
7013	Benny / Sherbrooke	45.464878	-73.626595	375
6033	16e avenue / Beau...	45.55828	-73.58316	424
7093	Laforest / Dudemaine	45.539806	-73.68726	42

only showing top 20 rows

- we can decide which rides happened during the same day and how many were more than one day

```
In [9]:  def sameday(day1,day2):
             if day1==day2:
                 same_day=1
                 print('yes')
             else:
                 same_day=0
             return(same_day)

         spark.udf.register("Check_same_day", sameday)

Out[9]:  <function __main__.Check_same_day>

In [10]: from  pyspark.sql.functions import month,year,dayofmonth,monotonically_increasing_id,udf,struct
         from pyspark.sql.types import IntegerType



         q=df_2019_10.select(year(df_2019_10.start_date),month(df_2019_10.start_date),dayofmonth(df_2019_10.start_date),dayofmonth(df_2019_


         q=q.withColumnRenamed('month(start_date)','month').withColumnRenamed('year(start_date)','year').withColumnRenamed('dayofmonth(sta

         q = q.withColumn('index',monotonically_increasing_id())
         df_2019_10 = df_2019_10.withColumn('index',monotonically_increasing_id())

         df_2019_10=df_2019_10.join(q,on='index')
```

index	start_date	start_station_code	duration_sec	is_member	year	month	sday	end_day	same
0	2019-10-01 00:00:08	6174	199	1	2019	10	1	1	1
1	2019-10-01 00:01:13	6196	205	1	2019	10	1	1	1
2	2019-10-01 00:01:34	6033	212	1	2019	10	1	1	1
3	2019-10-01 00:02:33	6136	256	1	2019	10	1	1	1
4	2019-10-01 00:02:34	6204	104	1	2019	10	1	1	1
5	2019-10-01 00:02:49	6052	596	1	2019	10	1	1	1
6	2019-10-01 00:04:07	6149	503	1	2019	10	1	1	1
7	2019-10-01 00:04:25	7032	438	1	2019	10	1	1	1
8	2019-10-01 00:04:51	6118	262	1	2019	10	1	1	1
9	2019-10-01 00:05:01	6204	519	0	2019	10	1	1	1
10	2019-10-01 00:05:01	6095	540	1	2019	10	1	1	1
11	2019-10-01 00:05:52	6753	1111	1	2019	10	1	1	1
12	2019-10-01 00:06:23	6387	359	1	2019	10	1	1	1
13	2019-10-01 00:06:26	6181	168	1	2019	10	1	1	1
14	2019-10-01 00:06:41	6254	475	1	2019	10	1	1	1
15	2019-10-01 00:07:11	6432	1517	0	2019	10	1	1	1
16	2019-10-01 00:07:46	6021	375	1	2019	10	1	1	1
17	2019-10-01 00:09:19	6404	1176	1	2019	10	1	1	1
18	2019-10-01 00:09:51	6100	466	1	2019	10	1	1	1
19	2019-10-01 00:11:00	6184	654	1	2019	10	1	1	1

only showing top 20 rows

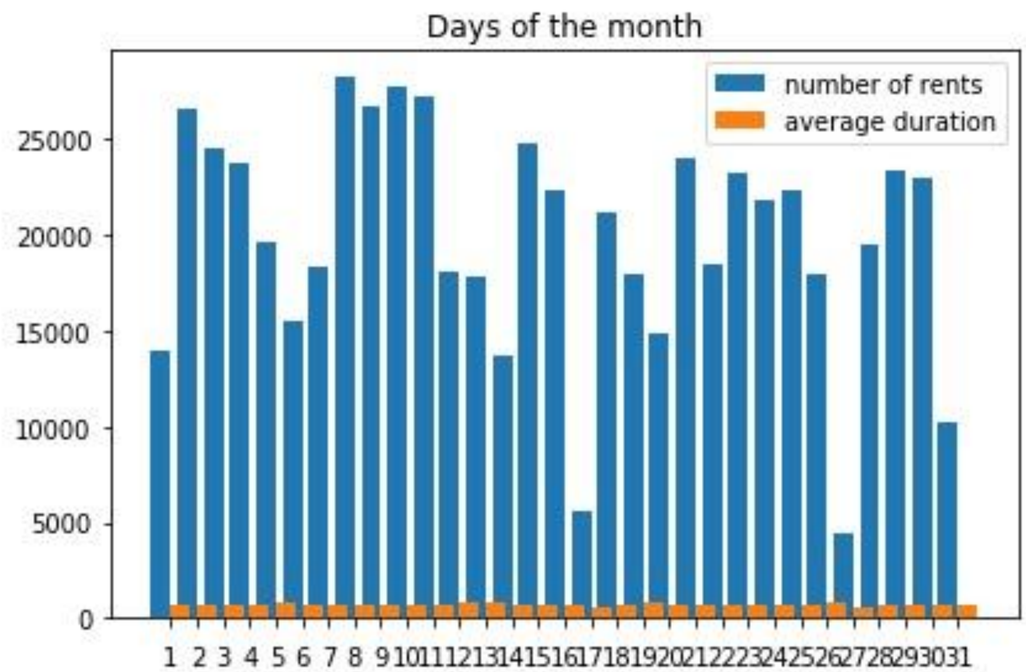
predictions

- The data at hand is time related so we have to run time series predictions on it. The step will be as follows:
  - extract the average time per ride and total number of rides per month for each station
  - put all these together to have the information for year 2019
  - run time series prediction

- we can extract the average time and number of rides per station using the code below:
-

```
In [13]: df_2019_10_agg=df_2019_10.groupBy('year','month','sday').agg({'sday':'count','duration_sec':'mean'})
         df_2019_10_agg=df_2019_10_agg.orderBy('sday')
         df_2019_10_agg.show(31)
```

```
+----+-----+----+-----------+-----------------+
|year|month|sday|count(sday)|avg(duration_sec)|
+----+-----+----+-----------+-----------------+
|2019|   10|   1|      14017| 636.348148676607|
|2019|   10|   2|      26547|679.7920292311749|
|2019|   10|   3|      24542|670.0463694890392|
|2019|   10|   4|      23822| 685.766392410377|
|2019|   10|   5|      19613|773.0914189568143|
|2019|   10|   6|      15544|688.2258749356665|
|2019|   10|   7|      18289|664.0854612061895|
|2019|   10|   8|      28226|722.1517395309289|
|2019|   10|   9|      26730|702.8726898615788|
|2019|   10|  10|      27761|713.6801988400994|
|2019|   10|  11|      27242| 733.204573819837|
|2019|   10|  12|      18073|797.5678636640292|
|2019|   10|  13|      17879|840.4145086414229|
|2019|   10|  14|      13734|721.5892675112858|
|2019|   10|  15|      24828|693.0045513130336|
|2019|   10|  16|      22370|682.5941439427805|
|2019|   10|  17|       5616| 541.031339031339|
|2019|   10|  18|      21218|664.1704213403714|
|2019|   10|  19|      17944|756.7014043691485|
|2019|   10|  20|      14862|741.1306015341138|
|2019|   10|  21|      23982| 702.249311983988|
```


Days of the month
number of rents
average duration
25000
20000
15000
10000
5000
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

- after extracting all the month information now we can have all information in one table

```
In [39]: df_2019_agg= df_2019_04_agg.union(df_2019_05_agg)
         df_2019_agg=df_2019_agg.union(df_2019_06_agg)
         df_2019_agg=df_2019_agg.union(df_2019_07_agg)
         df_2019_agg=df_2019_agg.union(df_2019_08_agg)
         df_2019_agg=df_2019_agg.union(df_2019_09_agg)
         df_2019_agg=df_2019_agg.union(df_2019_10_agg)

         df_2019_agg.show(200)
```

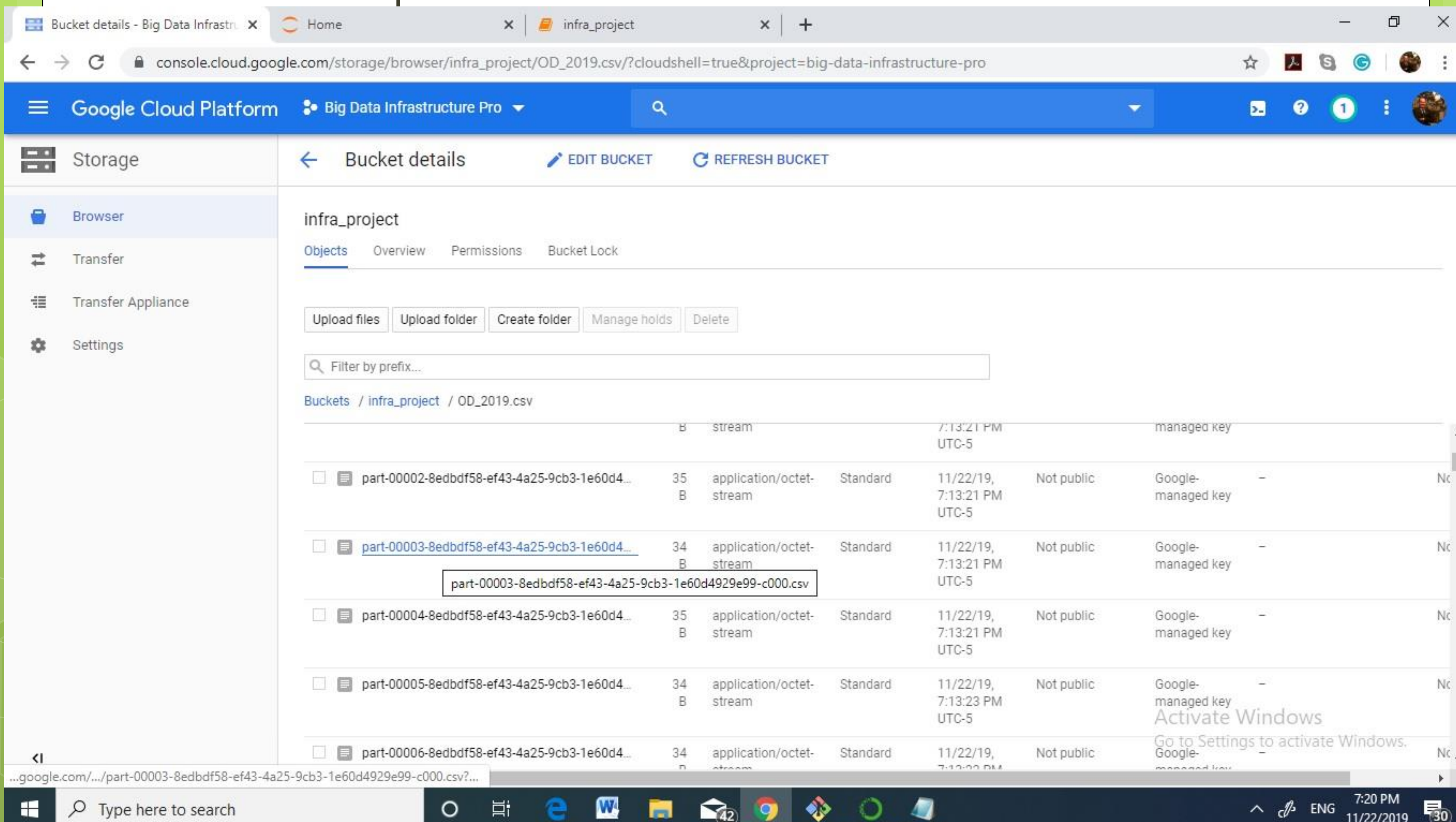
```
+----+-----+----+-----------+-----------------+
|year|month|sday|count(sday)|  avg(duration_sec)|
+----+-----+----+-----------+-----------------+
|2019|    4|  14|       9143| 868.4090561084982|
|2019|    4|  15|       7310| 653.9667578659371|
|2019|    4|  16|      13672| 715.0547103569339|
|2019|    4|  17|      19726|  806.916607523066|
|2019|    4|  18|      13505| 676.8959644576083|
|2019|    4|  19|       4673| 618.5182965974749|
|2019|    4|  20|       6604| 639.3069351907934|
|2019|    4|  21|      16306|1013.2635226297068|
|2019|    4|  22|      21354| 927.1408635384471|
|2019|    4|  23|      20897|  791.423505766378|
|2019|    4|  24|      10624| 648.4322289156627|
|2019|    4|  25|      20155| 743.7738526420243|
|2019|    4|  26|       8179| 623.4257244161878|
|2019|    4|  27|       7518| 605.6012237297153|
|2019|    4|  28|      14192| 792.2765642615558|
|2019|    4|  29|      20319| 757.1533540036419|
```

- now we can do the final cleaning

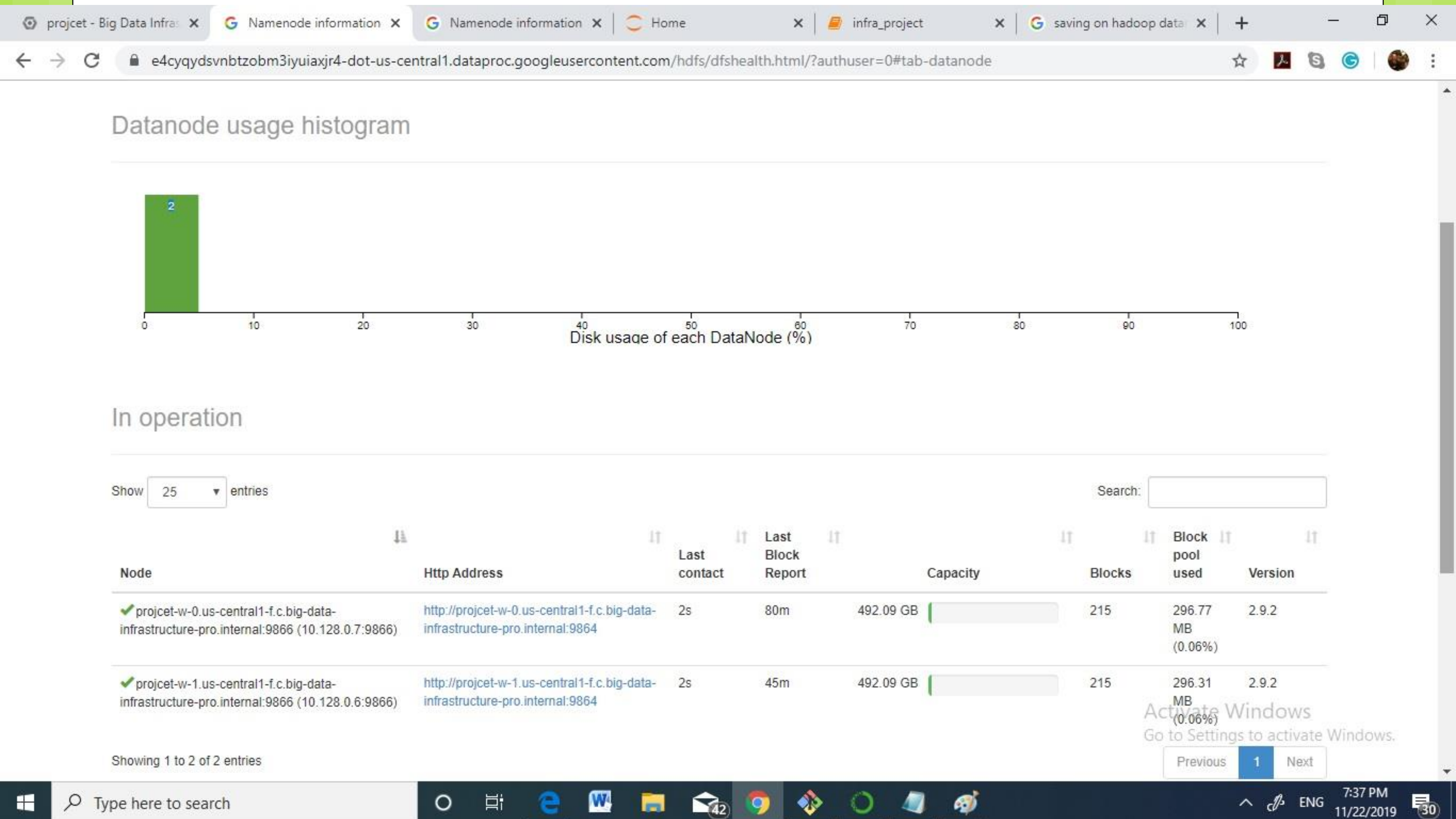
rides per day	average ride time/seconds	Date
9143	868.4090561084982	2019/4/14
7310	653.9667578659371	2019/4/15
13672	715.0547103569339	2019/4/16
19726	806.916607523066	2019/4/17
13505	676.8959644576083	2019/4/18
4673	618.5182965974749	2019/4/19
6604	639.3069351907934	2019/4/20
16306	1013.263522629706 8	2019/4/21
21354	927.1408635384471	2019/4/22
20897	791.423505766378	2019/4/23
10624	648.4322289156627	2019/4/24
20155	743.7738526420243	2019/4/25
8179	623.4257244161878	2019/4/26
7518	605.6012237297153	2019/4/27
14192	792.2765642615558	2019/4/28
20319	757.1533540036419	2019/4/29
23365	767.3380269634068	2019/4/30
16549	685.0921505831168	2019/5/1
19354	712.4081843546554	2019/5/2
14502	688.2785133085092	2019/5/3

only showing top 20 rows

- we can save the resulting table in Google cloud bucket that we made for our project and below we can see its partitions



Hdfs


projcet - Big Data Infras
Namenode information
Namenode information
Home
infra_project
saving on hadoop data
e4cyqydsvnbtzobm3iyuiaxjr4-dot-us-central1.dataproc.googleusercontent.com/hdfs/dfshealth.html/?authuser=0#tab-datanode
Datanode usage histogram
2
0
10
20
30
40
50
60
70
80
90
100
Disk usage of each DataNode (%)
In operation
Show 25 entries
Search:
Node
Http Address
Last contact
Last Block Report
Capacity
Blocks
Block pool used
Version
projcet-w-0.us-central1-f.c.big-data-infrastructure-pro.internal:9866 (10.128.0.7:9866)
http://projcet-w-0.us-central1-f.c.big-data-infrastructure-pro.internal:9864
2s
80m
492.09 GB
215
296.77 MB (0.06%)
2.9.2
projcet-w-1.us-central1-f.c.big-data-infrastructure-pro.internal:9866 (10.128.0.6:9866)
http://projcet-w-1.us-central1-f.c.big-data-infrastructure-pro.internal:9864
2s
45m
492.09 GB
215
296.31 MB (0.06%)
2.9.2
Showing 1 to 2 of 2 entries
Previous
1
Next
Activate Windows
Go to Settings to activate Windows.
Type here to search
ENG
7:37 PM
11/22/2019

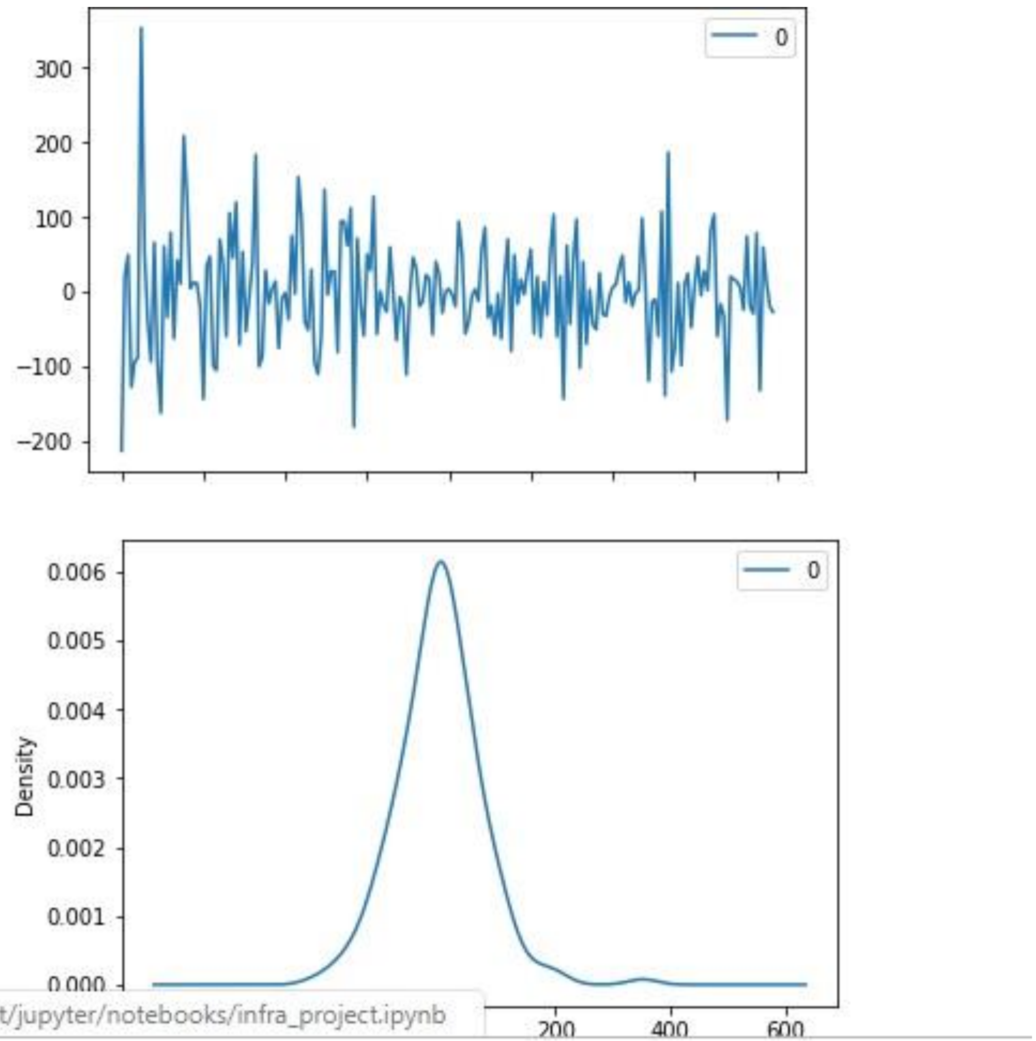
- after the data is ready we can run the ARIMA time series prediction

```
In [50]: df_2019_pd=df_2019_agg.toPandas()

         data = df_2019_pd['average ride time/seconds']
         data.index=df_2019_pd[['Date']]

         model = ARIMA(data, order=(5,1,0),dates=df_2019_pd['Date'])
         model_fit = model.fit(disp=0)
         print(model_fit.summary())
         # plot residual errors
         residuals = DataFrame(model_fit.resid)
         residuals.plot()
         pyplot.show()
         residuals.plot(kind='kde')
         pyplot.show()
         print(residuals.describe())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:    D.average ride time/seconds   No. Observations:          200
Model:                         ARIMA(5, 1, 0)   Log Likelihood       -1141.916
Method:                               css-mle   S.D. of innovations     72.775
Date:                        Sat, 23 Nov 2019   AIC                   2297.832
Time:                                01:38:02   BIC                   2320.921
Sample:                            01-15-2019   HQIC                  2307.176
                                 - 01-31-2019
=====================================================================================================
                                        coef    std err          z      P>|z|      [0.025      0.975]
-----------------------------------------------------------------------------------------------------
const                                -0.4001      1.666     -0.240      0.810      -3.665       2.865
ar.L1.D.average ride time/seconds    -0.4369      0.067     -6.541      0.000      -0.568      -0.306
ar.L2.D.average ride time/seconds    -0.5138      0.069     -7.467      0.000      -0.649      -0.379
ar.L3.D.average ride time/seconds    -0.4426      0.070     -6.309      0.000      -0.580      -0.305
ar.L4.D.average ride time/seconds    -0.3492      0.069     -5.074      0.000      -0.484      -0.214
ar.L5.D.average ride time/seconds    -0.3774      0.067     -5.620      0.000      -0.509      -0.246
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            0.6566            -0.9596j            1.1627           -0.1545
AR.2            0.6566            +0.9596j            1.1627            0.1545
AR.3           -1.2785            -0.0000j            1.2785           -0.5000
AR.4           -0.4800            -1.1413j            1.2381           -0.3134
AR.5           -0.4800            +1.1413j            1.2381            0.3134
```



0
300
200
100
0
−100
−200
0
0.006
0.005
0.004
0.003
0.002
0.001
0.000
Density
200
400
600
efault/jupyter/notebooks/infra_project.ipynb