INTRODUCTION:

Census tracts are relatively permanent small-area geographic divisions of a county or statistically equivalent entity defined for the tabulation and presentation of data from the decennial census and selected other statistical programs

Census tracts defined by these criteria will also be used to tabulate and publish estimates from the American Community Survey (ACS) and potentially data from other Bureau of the Census (Census Bureau) censuses and surveys. In addition to census tracts, the Participant Statistical Areas Program (PSAP) encompasses the review and update of census block groups, census designated places, and census county divisions.

Census tracts generally have a population size between 1,200 and 8,000 people, with an optimum size of 4,000 people. A census tract usually covers a contiguous area; however, the spatial size of census tracts varies widely depending on the density of settlement. Census tract boundaries are delineated with the intention of being maintained over a long time so that statistical comparisons can be made from census to census. Census tracts occasionally are split due to population growth or merged as a result of substantial population decline.

IMPORTANCE OF CENCUS TRACT IN DATA ANALYSIS :

.. census tracts are comprised on one or more coterminous block groups.
.. on average, a census tract is comprised of three block groups.
• Census tracts are used by many Federal, state and local governments for compliance and program management

Census tracts are important for many reasons. It is easy to misidentify or misunderstand patterns and characteristics within cities, counties and metros which become obfuscated using these higher levels, more aggregate, geographies. Many cities and counties that might be experiencing demographic-economic decline will often have bright spots that are groups of a few or many census tracts.

- Importance of Census Tracts for Data Analytics
  Census tracts are important for many reasons. A partial list of reasons is provided below.
  • Covering the U.S. wall-to-wall, census tracts are the preferred "small area" geography for superior data analytics.
  • The Census Bureau now produces annual tract demographic-economic data from the American Community Survey; there is an evolving time-series at the tract level creating new analytical opportunities.
  • Originally developed to equivalence neighborhoods, many still do.
  • Defined by the Census Bureau in collaboration with local groups, tracts typically reflect boundaries meaningful for local area analysis.
  • Defined generally for use with each new decennial census, most tract boundaries are stable and non-changing for ten years and many much longer.
  • Designed to average 4,000 population, there are more than twice as many census tracts (73,056) than ZIP code areas (33,129).
  • Tract boundaries are well-defined; unlike ZIP code areas which are subject to multi-sourced geographic definitions.
  • Many data developers (e.g., epidemiologists) use census tract geography to tabulate their own small area data enabling more effective

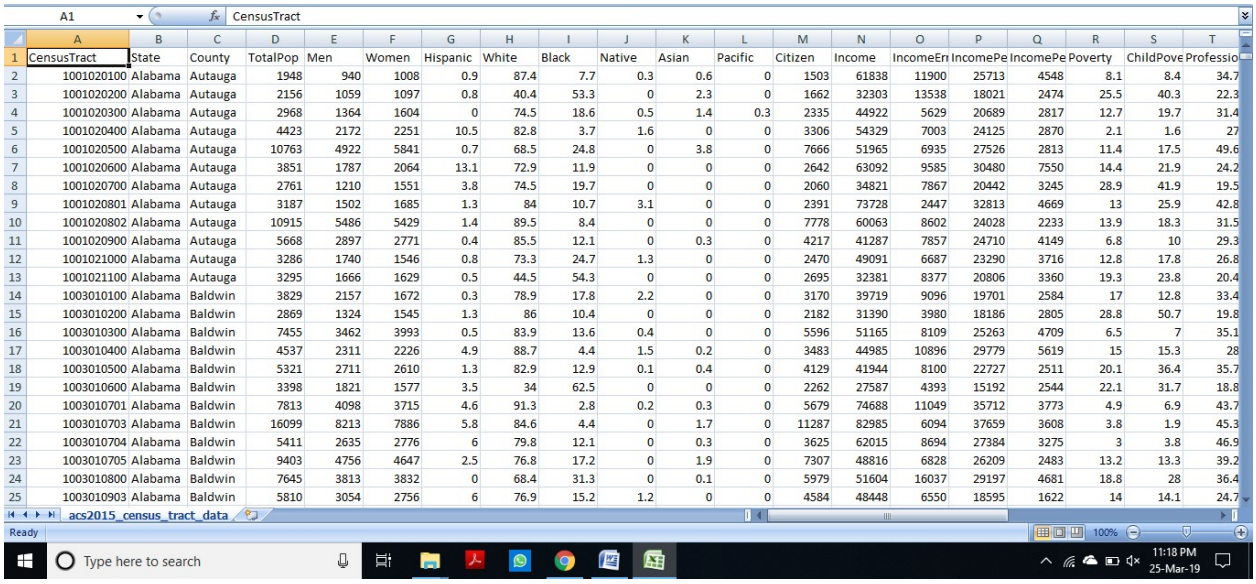
use of those data with Census Bureau census tract data.

• As a statistical geographic area (in contrast to politically defined areas, census tracts are coterminous with counties; data at the census tract level can be aggregated to the county level.

• Small area estimates for tracts are typically more reliable than for block groups..

DATA SET DESCRIPTION:

Link for dataset:



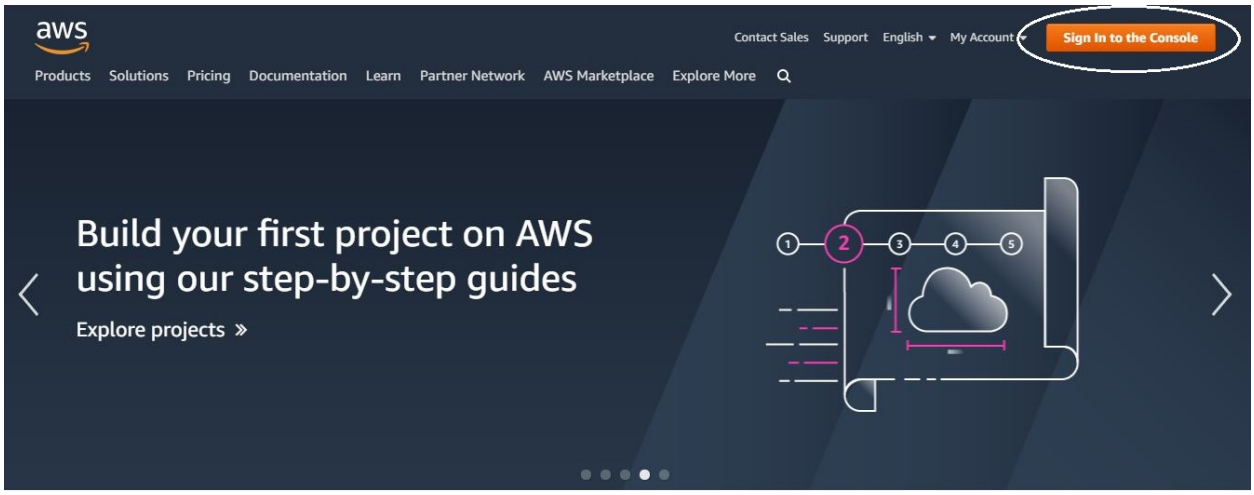
CensusTract	State	County	TotalPop	Men	Women	Hispanic	White	Black	Native	Asian	Pacific	Citizen	Income	IncomeEr	IncomePe	IncomePe	Poverty	ChildPove	Professio
1001020100	Alabama	Autauga	1948	940	1008	0.9	87.4	7.7	0.3	0.6	0	1503	61838	11900	25713	4548	8.1	8.4	34.7
1001020200	Alabama	Autauga	2156	1059	1097	0.8	40.4	53.3	0	2.3	0	1662	32303	13538	18021	2474	25.5	40.3	22.3
1001020300	Alabama	Autauga	2968	1364	1604	0	74.5	18.6	0.5	1.4	0.3	2335	44922	5629	20689	2817	12.7	19.7	31.4
1001020400	Alabama	Autauga	4423	2172	2251	10.5	82.8	3.7	1.6	0	0	3306	54329	7003	24125	2870	2.1	1.6	27
1001020500	Alabama	Autauga	10763	4922	5841	0.7	68.5	24.8	0	3.8	0	7666	51965	6935	27526	2813	11.4	17.5	49.6
1001020600	Alabama	Autauga	3851	1787	2064	13.1	72.9	11.9	0	0	0	2642	63092	9585	30480	7550	14.4	21.9	24.2
1001020700	Alabama	Autauga	2761	1210	1551	3.8	74.5	19.7	0	0	0	2060	34821	7867	20442	3245	28.9	41.9	19.5
1001020801	Alabama	Autauga	3187	1502	1685	1.3	84	10.7	3.1	0	0	2391	73728	2447	32813	4669	13	25.9	42.8
1001020802	Alabama	Autauga	10915	5486	5429	1.4	89.5	8.4	0	0	0	7778	60063	8602	24028	2233	13.9	18.3	31.5
1001020900	Alabama	Autauga	5668	2897	2771	0.4	85.5	12.1	0	0.3	0	4217	41287	7857	24710	4149	6.8	10	29.3
1001021000	Alabama	Autauga	3286	1740	1546	0.8	73.3	24.7	1.3	0	0	2470	49091	6687	23290	3716	12.8	17.8	26.8
1001021100	Alabama	Autauga	3295	1666	1629	0.5	44.5	54.3	0	0	0	2695	32381	8377	20806	3360	19.3	23.8	20.4
1003010100	Alabama	Baldwin	3829	2157	1672	0.3	78.9	17.8	2.2	0	0	3170	39719	9096	19701	2584	17	12.8	33.4
1003010200	Alabama	Baldwin	2869	1324	1545	1.3	86	10.4	0	0	0	2182	31390	3980	18186	2805	28.8	50.7	19.8
1003010300	Alabama	Baldwin	7455	3462	3993	0.5	83.9	13.6	0.4	0	0	5596	51165	8109	25263	4709	6.5	7	35.1
1003010400	Alabama	Baldwin	4537	2311	2226	4.9	88.7	4.4	1.5	0.2	0	3483	44985	10896	29779	5619	15	15.3	28
1003010500	Alabama	Baldwin	5321	2711	2610	1.3	82.9	12.9	0.1	0.4	0	4129	41944	8100	22727	2511	20.1	36.4	35.7
1003010600	Alabama	Baldwin	3398	1821	1577	3.5	34	62.5	0	0	0	2262	27587	4393	15192	2544	22.1	31.7	18.8
1003010701	Alabama	Baldwin	7813	4098	3715	4.6	91.3	2.8	0.2	0.3	0	5679	74688	11049	35712	3773	4.9	6.9	43.7
1003010703	Alabama	Baldwin	16099	8213	7886	5.8	84.6	4.4	0	1.7	0	11287	82985	6094	37659	3608	3.8	1.9	45.3
1003010704	Alabama	Baldwin	5411	2635	2776	6	79.8	12.1	0	0.3	0	3625	62015	8694	27384	3275	3	3.8	46.9
1003010705	Alabama	Baldwin	9403	4756	4647	2.5	76.8	17.2	0	1.9	0	7307	48816	6828	26209	2483	13.2	13.3	39.2
1003010800	Alabama	Baldwin	7645	3813	3832	0	68.4	31.3	0	0.1	0	5979	51604	16037	29197	4681	18.8	28	36.4
1003010903	Alabama	Baldwin	5810	3054	2756	6	76.9	15.2	1.2	0	0	4584	48448	6550	18595	1622	14	14.1	24.7

STEPS FOR SETTING UP OF CLOUD ENVIRONMENT:

1. AUTHENTICATION:

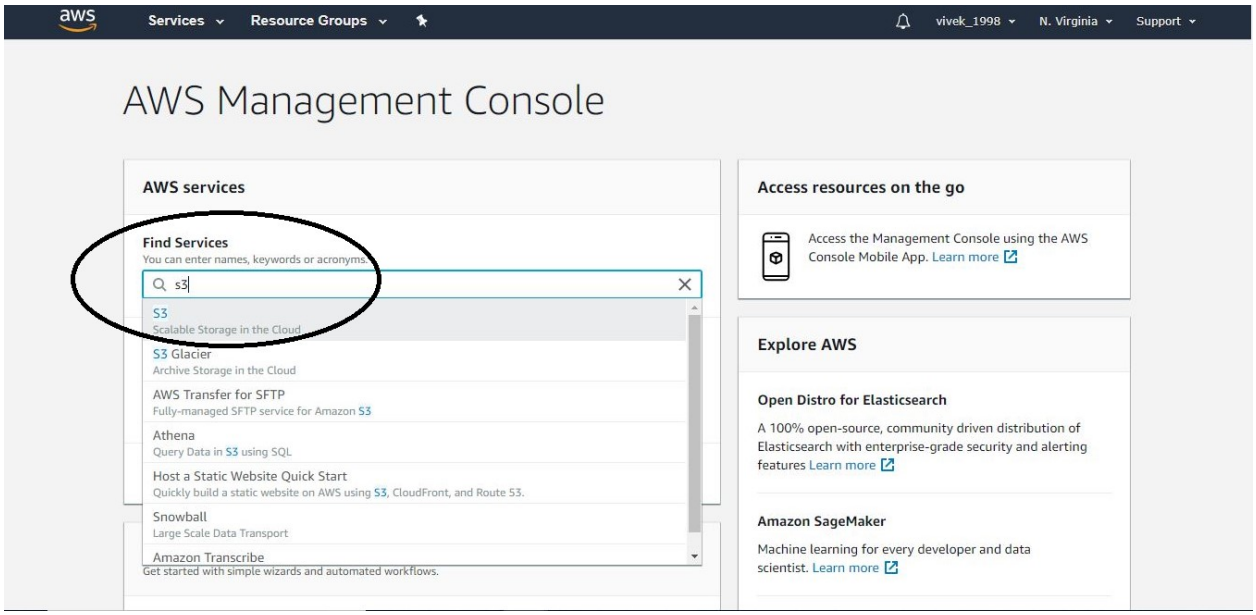
STEP 1.1: create a account at https://aws.amazon.com/

STEP 1.2: Sign into your AWS Amazon console using your credentials

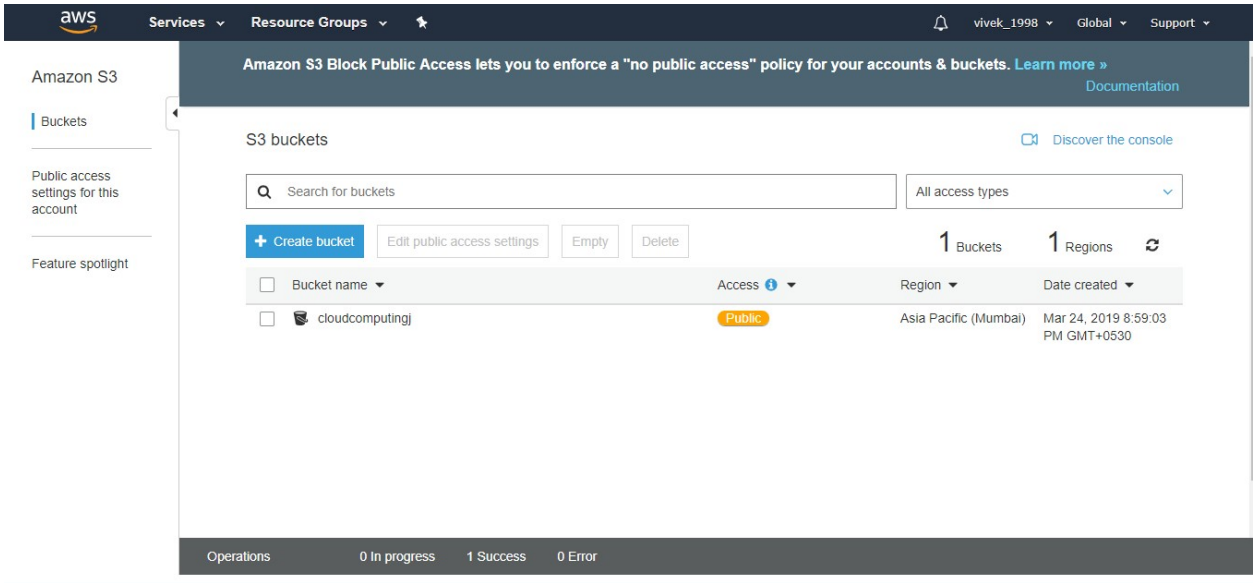


2. CONSOLE (OPENING S3)

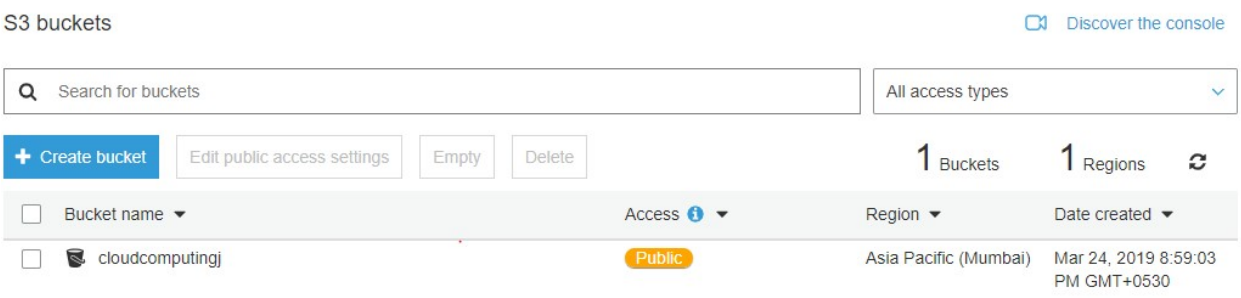
STEP 2.1: Search for S3 storage and open it



RESULT: This screen will be displayed



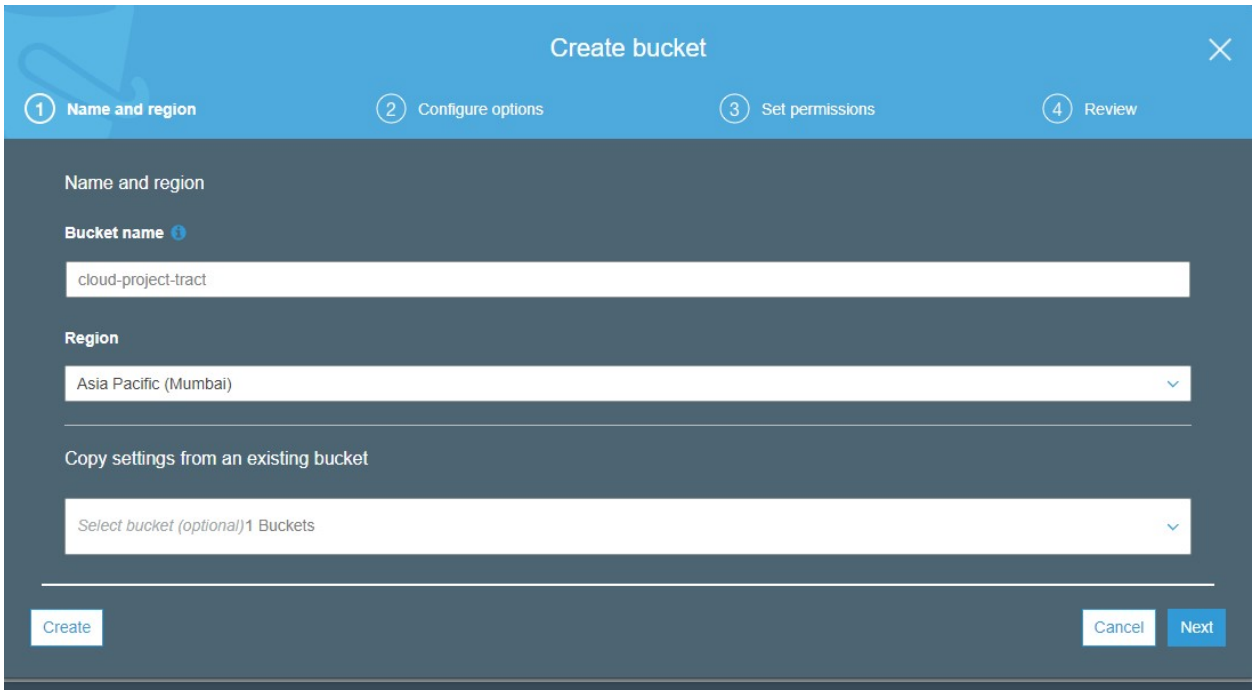
BUCKET INFORMATION WILL BE DISPLAYED HERE



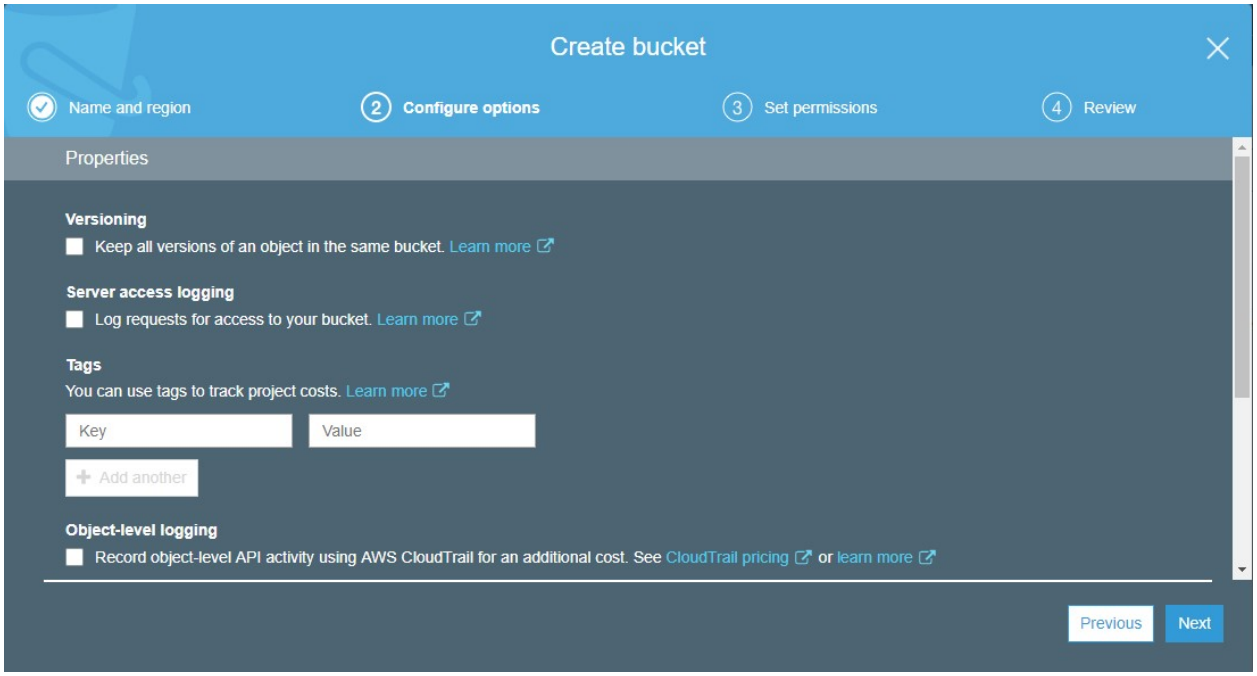
3. CREATING OF BUCKET WITH PUBLIC ACCESS

STEP 3.1: Click on create button option

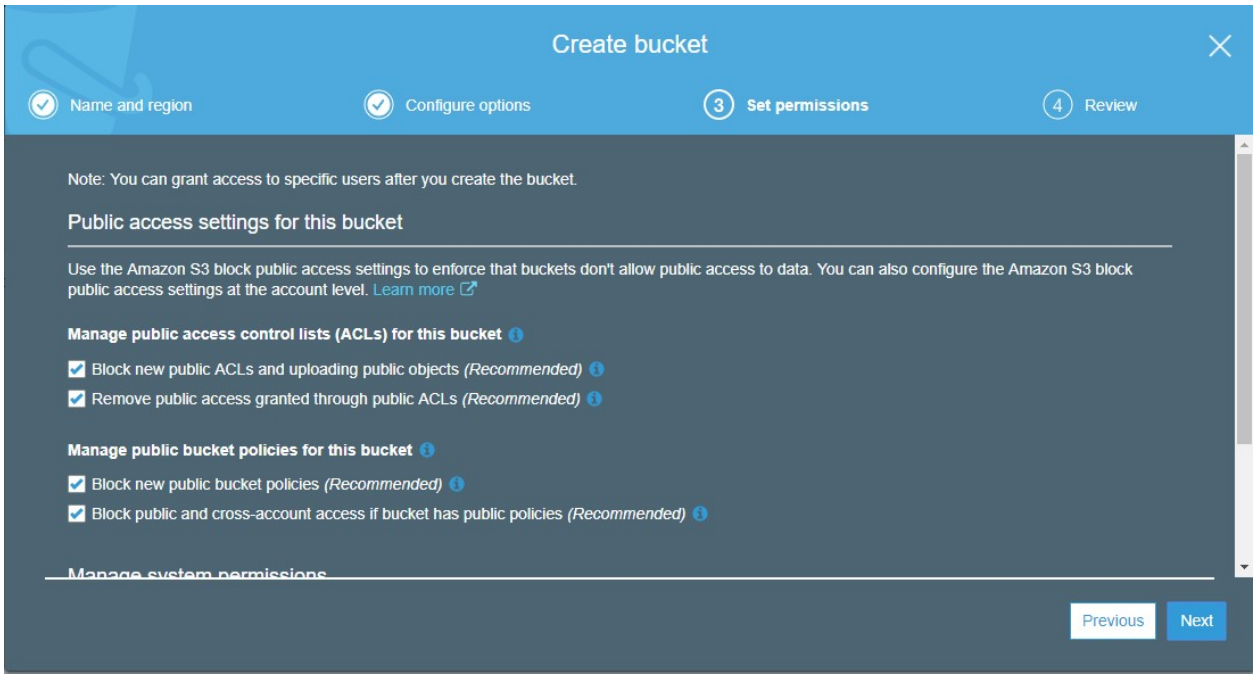
STEP 3.2: Fill details like name and select region you want to deploy your Storage.



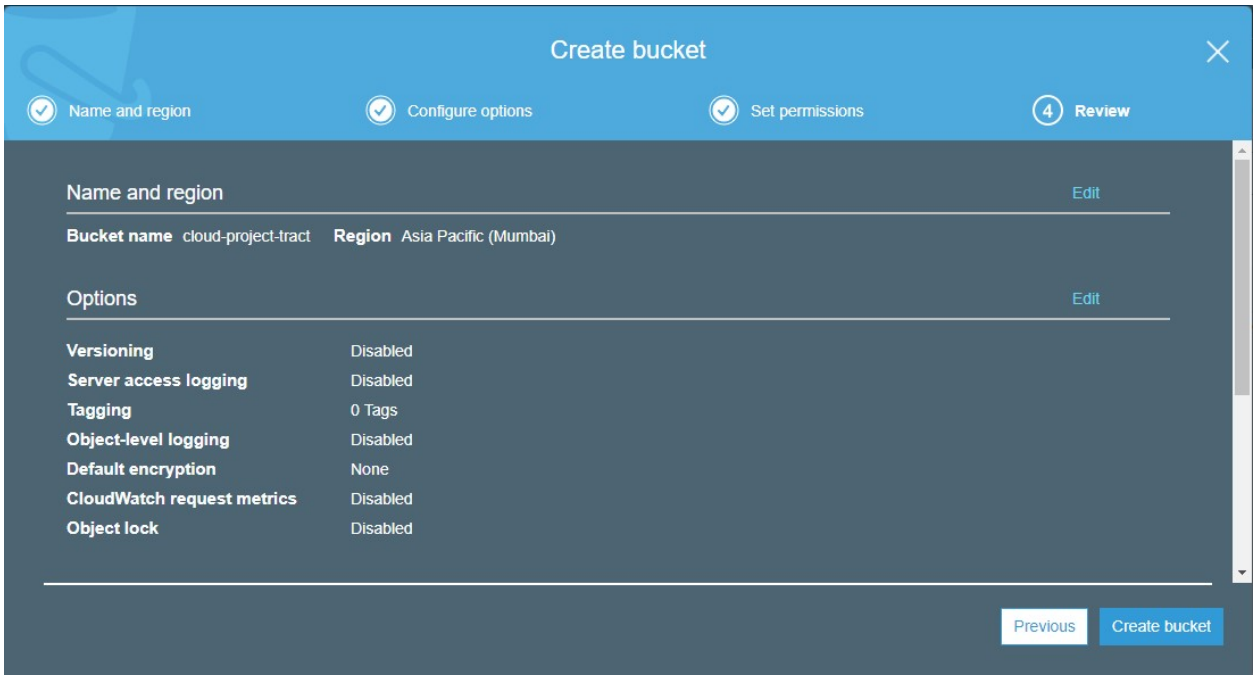
STEP 3.3: Configure key value if required



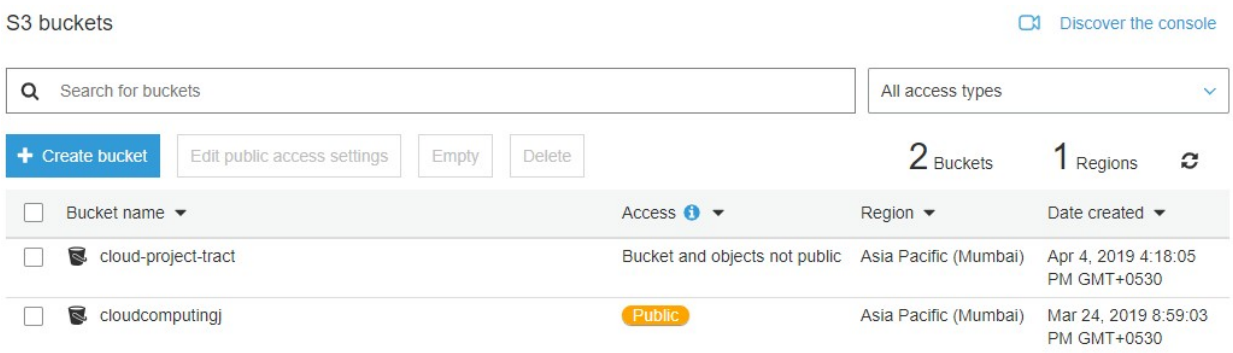
STEP 3.4: Set up permissions(these will allow to make your bucket public)



STEP 3.5: Click on create bucket after reviewing bucket details

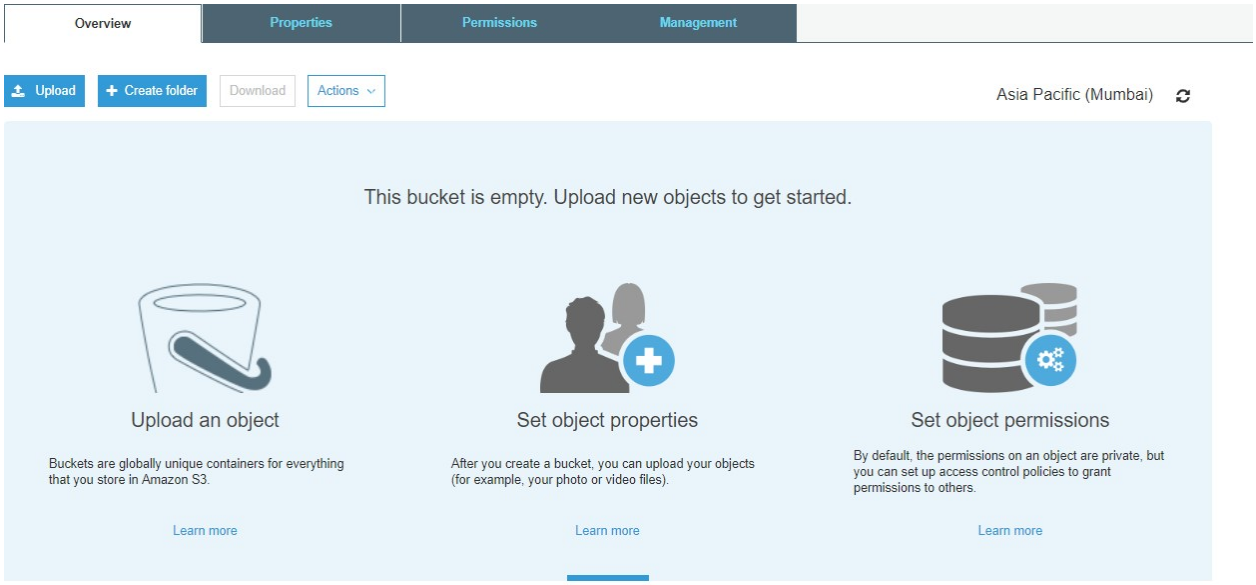


RESULT:

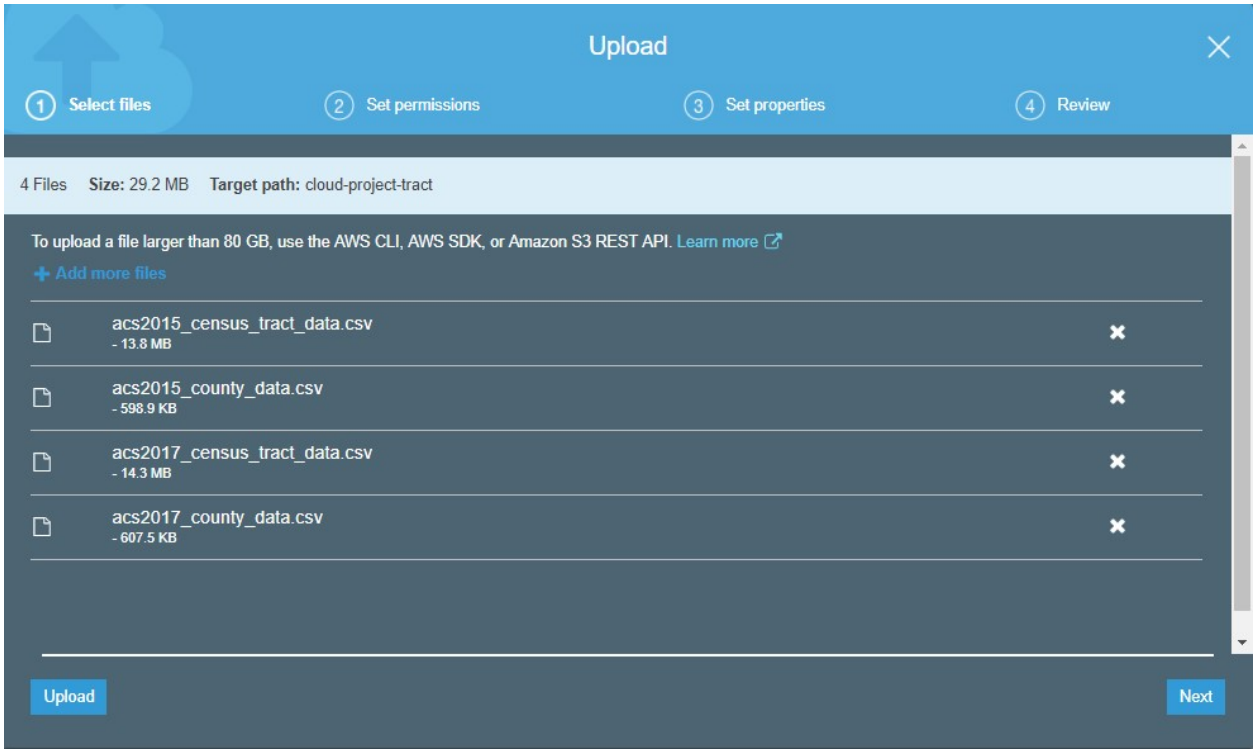

S3 buckets
Discover the console
Search for buckets
All access types
Create bucket
Edit public access settings
Empty
Delete
2 Buckets
1 Regions
Bucket name
Access
Region
Date created
cloud-project-tract
Bucket and objects not public
Asia Pacific (Mumbai)
Apr 4, 2019 4:18:05 PM GMT+0530
cloudcomputingj
Public
Asia Pacific (Mumbai)
Mar 24, 2019 8:59:03 PM GMT+0530

4. UPLOADING OBJECTS TO S3:

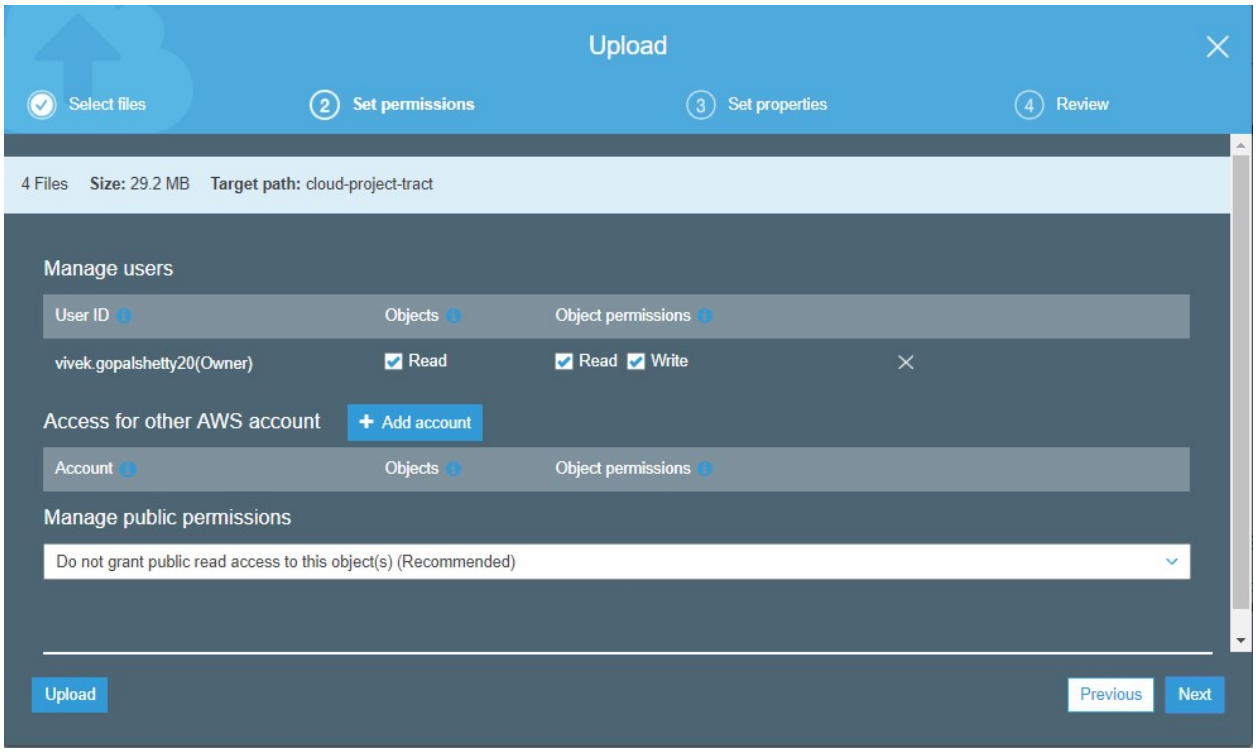
STEP 4.1: Click on upload



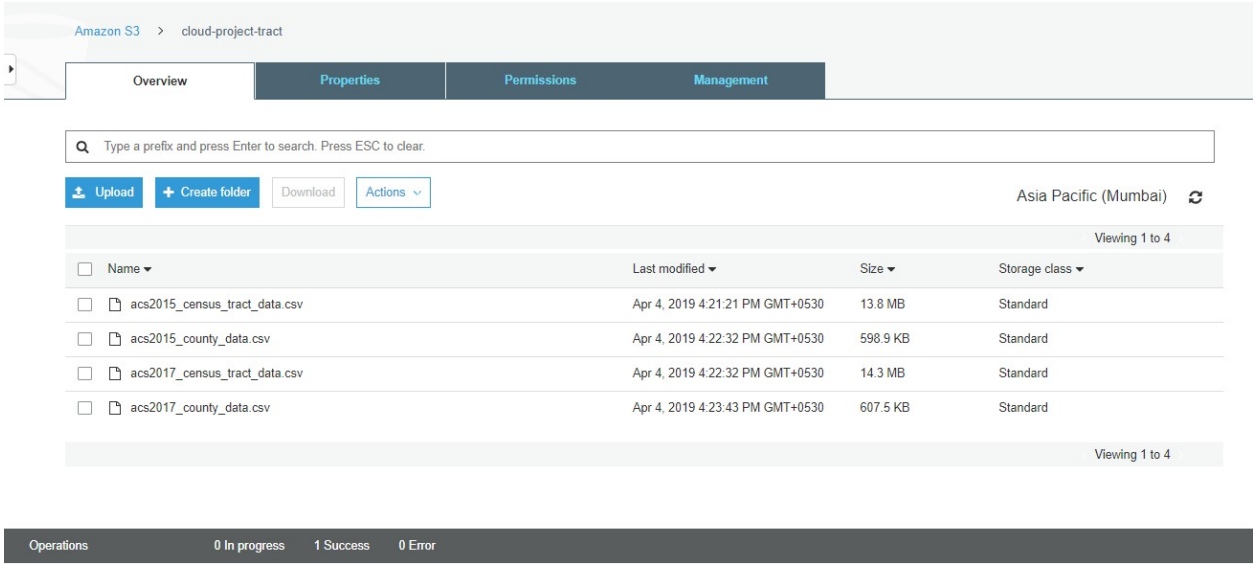
STEP 4.2: Select files from local system and drop them into bucket



STEP 4.3: While collaboration you can add other AWS accounts and give access to them.

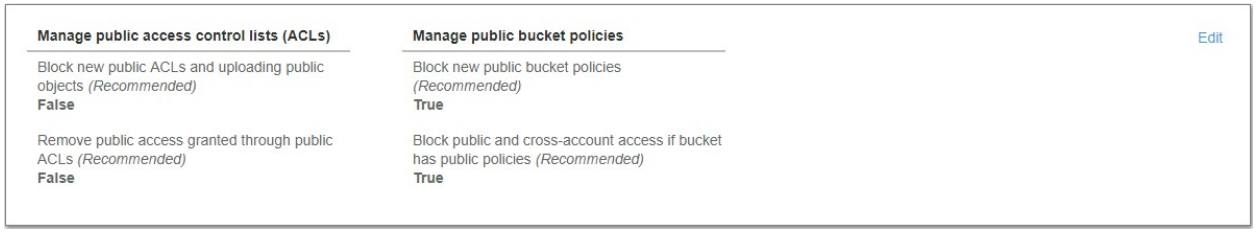


RESULT:

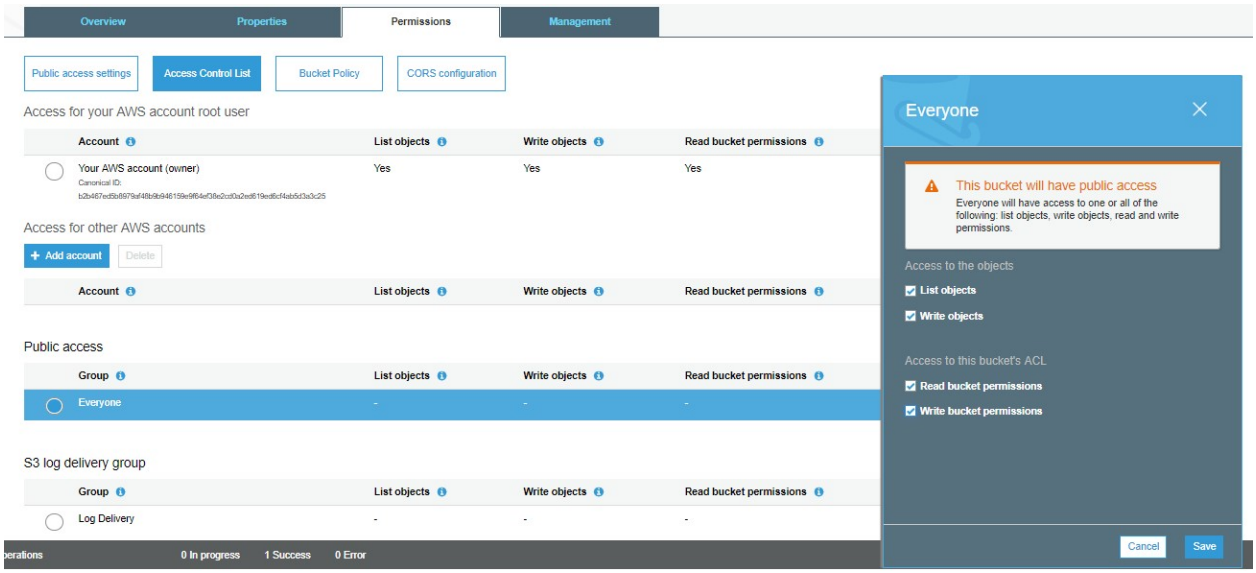


5. MAKING BUKCET PUBLIC

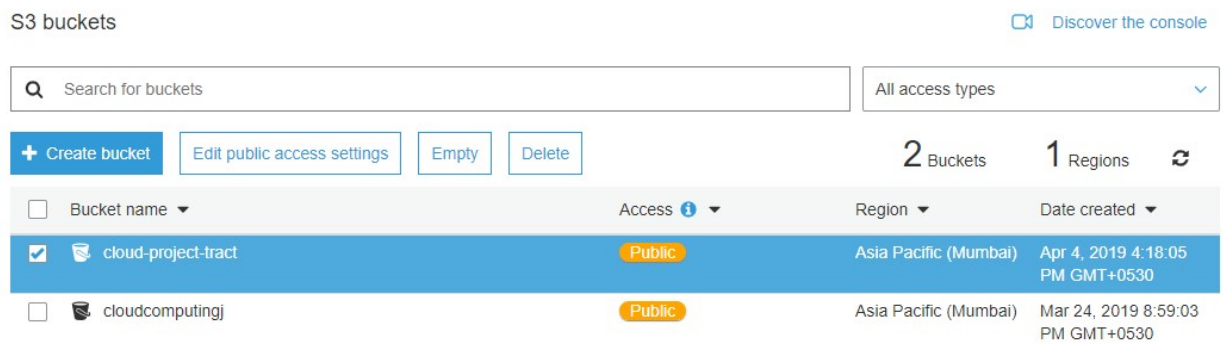
Important: Make sure your initial configuration is as shown in picture.



STEP 5: Select bucket and then go to permissions and then click on Public access-->everyone and then tick mark (read/write access) and confirm.



RESULT:



SETTING UP OF CODING ENVIRONMENT:

We have done data processing with data from AWS S3.

A small rest API has been provided to you. We have implemented one function that API needs to call to find the average census tract population size for a given city.

Setup

Before you start this lab you should make a new virtual environment in this directory and start it. Next, notice that there is a file called `requirements.txt`. This is a format that pip understands and that contains all the python packages needed for this project. You can install them with `pip install -r requirements.txt`

The REST API

This API only has two endpoints. The / endpoint just returns a string saying "Welcome to cloud computing project!"

The second endpoint is documented as follows

`/average_pop?city`

City: name of a US city. Can be one of

' Alaska',

'Arizona',

'Arkansas',

'California',

'Colorado',

'Connecticut',

'Delaware',

'District of Columbia',

'Florida',

'Georgia',

'Hawaii',

'Idaho',

'Illinois',

'Indiana',

'Iowa',

'Kansas',

'Kentucky',

'Louisiana',

'Maine',

'Maryland',

'Massachusetts',

'Michigan',

'Minnesota',

'Mississippi',

'Missouri',

'Montana',

'Nebraska',

'Nevada',

'New Hampshire',

'New Jersey',

'New Mexico',

'New York',

'North Carolina',

'North Dakota',

'Ohio',

'Oklahoma',

'Oregon',

'Pennsylvania',

'Puerto Rico',

'Rhode Island',

'South Carolina',

'South Dakota',

'Tennessee',

'Texas',

'Utah',

'Vermont',

'Virginia',

'Washington',

'West Virginia',

'Wisconsin',

'Wyoming'

Virtual Environments

Virtual environments are a popular tool with python developers for self containing dependencies within a directory. You simply make a new virtual environment, and anything you install or remove with pip will only apply to that directory, not your global pip packages.

To make a new virtual environment: `virtualenv ./project_directory`

To start that environment: _**On Mac/Linux**_ `source ./project_directory/bin/activate` _**On windows**_ `.\project_directory\Source\activate`

To stop that environment: _**On Mac/Linux**_ `source ./project_directory/bin/deactivate` _**On Windows**_ `.\project_directory\Source\activate`

Testing Code

To test your code, we have written a small program in `test.py` which just calls the `average_pop` endpoint for every city and prints out the results.

sThe goal is to set up for Elastic Beanstalk, where we will run this API on a cluster of multiple machines to increase performance.

STEPS FOR SETTING UP OF BEANSTALK:

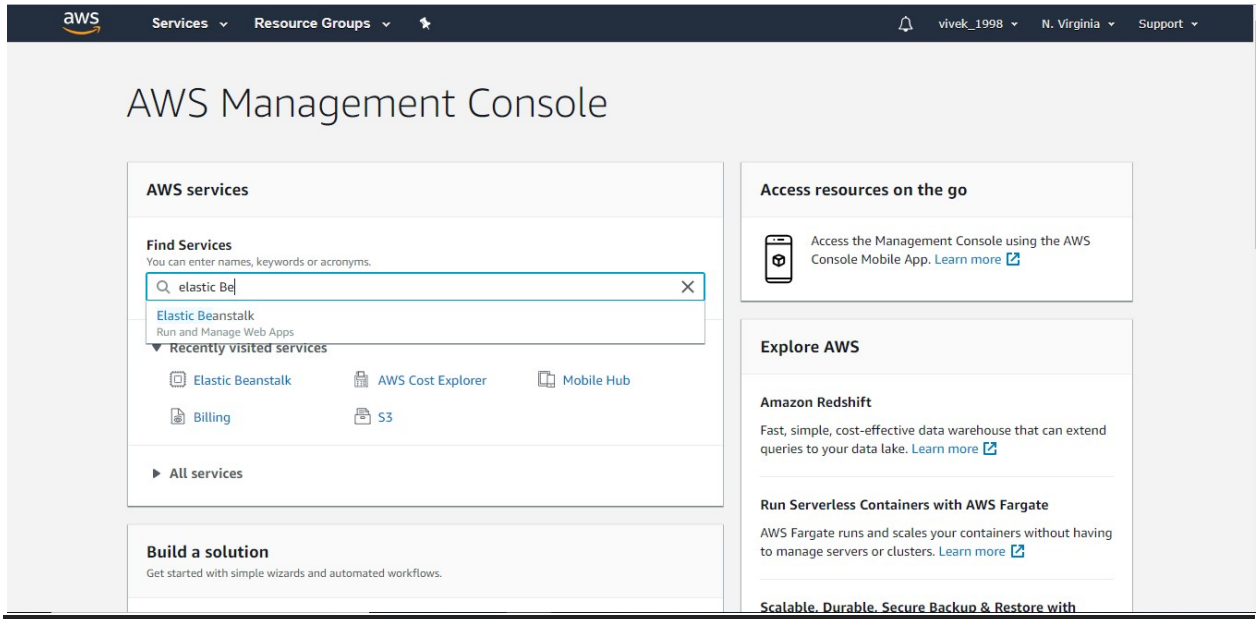
1. AUTHENTICATION:

STEP 1.1: create a account at https://aws.amazon.com/

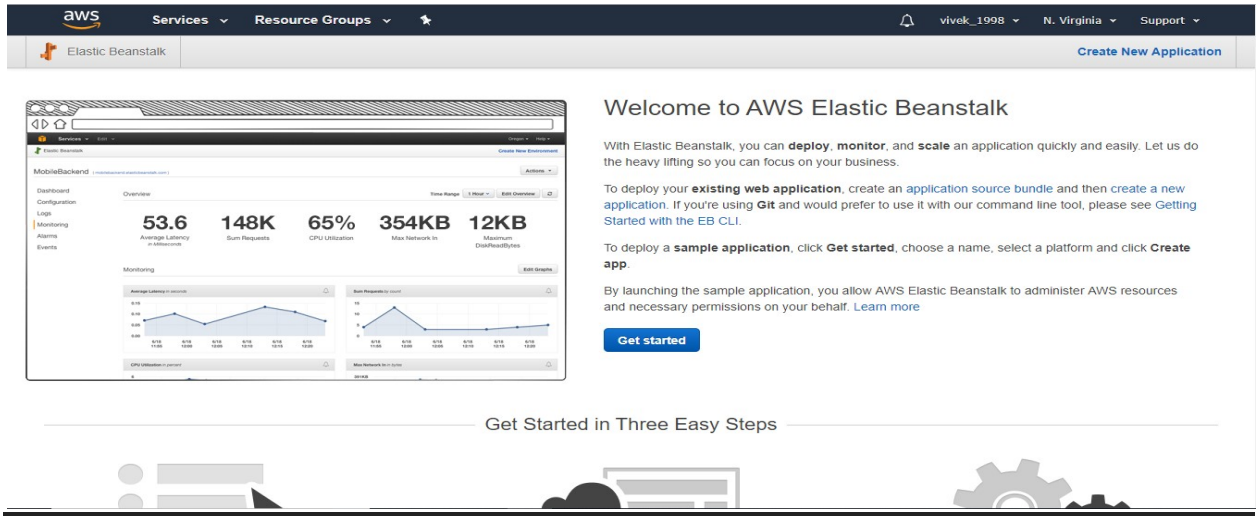
STEP 1.2: Sign into your AWS Amazon console using your credentials

2. CONSOLE (OPENING Elastic Beanstalk)

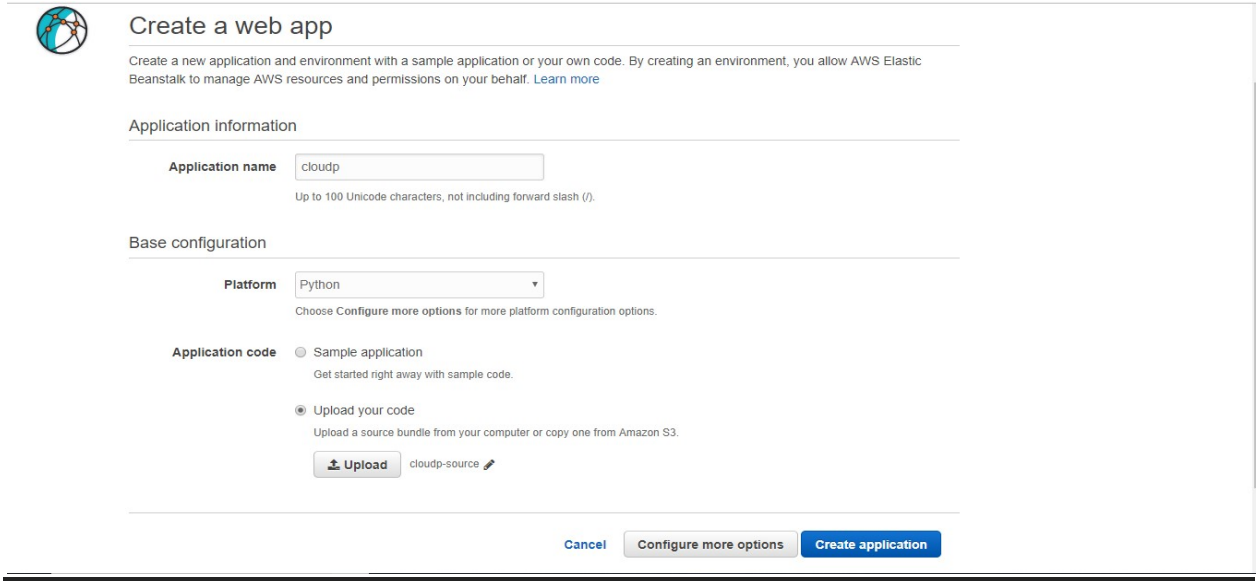
STEP 2.1: Search for **Elastic Beanstalk** and open it



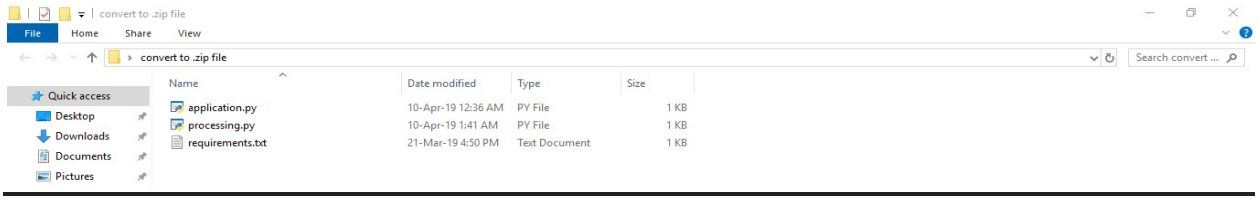
RESULT :



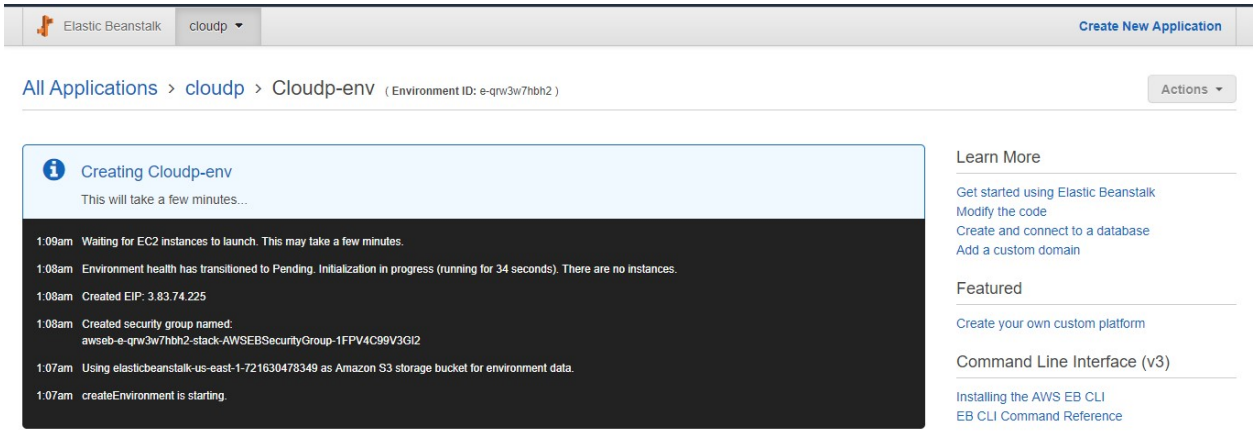
4. Now click on create and application and fill details as in image exactly



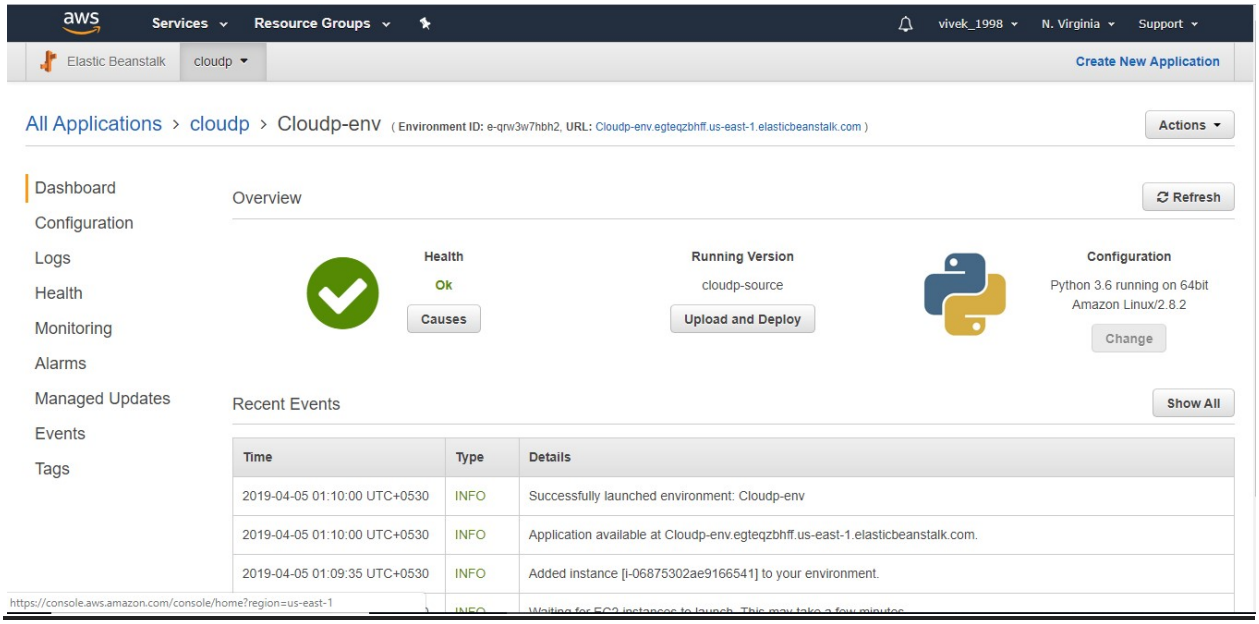
MAKE ZIP FILE OF IT AND UPLOAD IN APPLICATION CODE



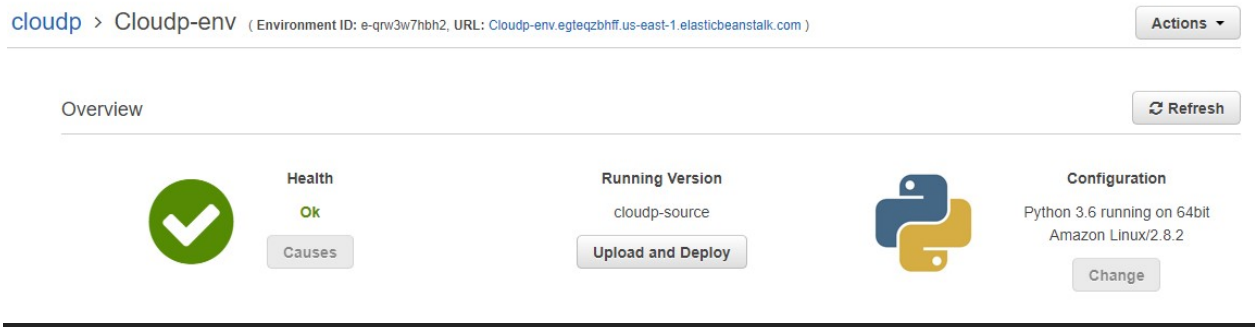
RESULT: wait for 5-10 mins for your web app to deploy on cloud



RESULT: you can check health and status also



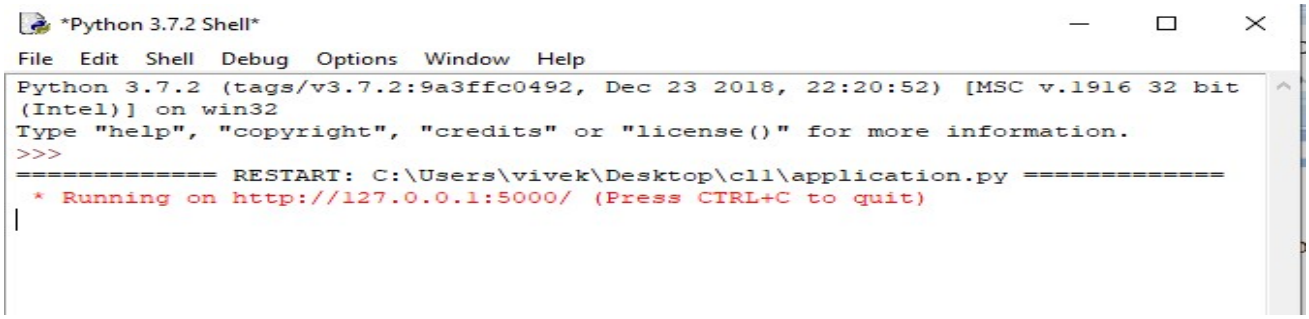
ACTION: Use link given at URL in test.py now to get average population



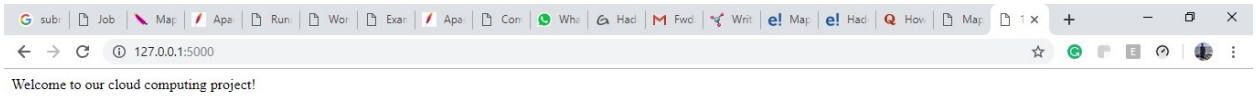
FLASK END TO END API:

```
from flask import Flask, request
import processing as p
app = Flask(__name__)


@app.route("/")
def index():
    return "Welcome to our cloud computing project!"


# /average_pop?city
@app.route("/average_pop")
def process():
    cityname = request.args.get('city')
    if cityname == None:
        return "No city name"
    avg = p.get_avg_pop(cityname)
    return str(avg)


if __name__ == "__main__":
    app.run()
```

OUTPUT:



ON WEB APP:

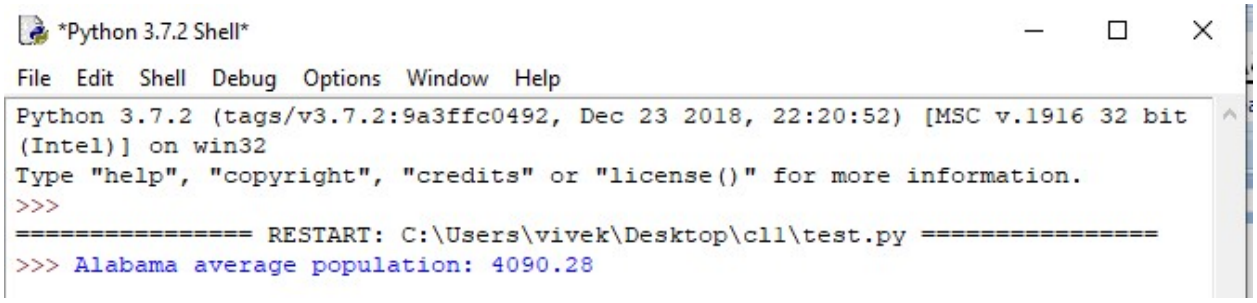


PROCESSING :

```
import boto3 as aws
from botocore.handlers import disable_signing
import pandas as pd
import sys

if sys.version_info[0] < 3:
    from StringIO import StringIO                    # Check for python version to use correct
library
else:
    from io import StringIO

def get_census_data(city):          #given city name extract that data from AWS S3 storage
    filename = "acs2015.csv"
    s3 = aws.resource('s3')
    s3.meta.client.meta.events.register('choose-signer.s3.*', disable_signing)  #disable signing
    obj = s3.Object(bucket_name="cloudcomputingj", key=filename)      #connecting to our
bucket
    res = obj.get()
    raw_data = res['Body'].read().decode('UTF-8')    #decode data once fetched from server
    data = StringIO(raw_data)
    df = pd.read_csv(data)
    return df          #return dataFrame required

def get_avg_pop(city):        #given a city name finding average population of state using cencus
tract
    df = get_census_data(city) # Makes a call to get the city data from S3
    condition=(df['State']==city)
    df=df[condition]
    # Calculate and return the average census tract population
    total_pop=df['TotalPop'].sum(axis=0)
    num_tracts=df.shape[0]
    return str(round(total_pop / num_tracts,2))
```

```
TEST  (input place )
import requests as r
import threading

cities = ['Alabama']
'''
 'Alaska',
 'Arizona',
 'Arkansas',
 'California',
 'Colorado',
 'Connecticut',
 'Delaware',
 'District of Columbia',
 'Florida',
 'Georgia',
 'Hawaii',
 'Idaho',
 'Illinois',
 'Indiana',
 'Iowa',
 'Kansas',
 'Kentucky',
 'Louisiana',
 'Maine',
 'Maryland',
 'Massachusetts',
 'Michigan',
 'Minnesota',
 'Mississippi',
 'Missouri',
 'Montana',
 'Nebraska',
 'Nevada',
 'New Hampshire',
 'New Jersey',
 'New Mexico',
 'New York',
 'North Carolina',
 'North Dakota',
```

```
 'Ohio',
 'Oklahoma',
 'Oregon',
 'Pennsylvania',
 'Puerto Rico',
 'Rhode Island',
 'South Carolina',
 'South Dakota',
 'Tennessee',
 'Texas',
 'Utah',
 'Vermont',
 'Virginia',
 'Washington',
 'West Virginia',
 'Wisconsin',
 'Wyoming']
'''

def get_avg(c):
    res = r.get('http://localhost:5000/average_pop?city=' + c)
    print(c + " average population: " + res.text)

for c in cities:
    threading.Thread(target=get_avg, args=(c,)).start()
```



ON SERVER REQUESTS

```
127.0.0.1 - - [10/Apr/2019 01:54:08] "GET /average_pop?city=AlabamaAlaska HTTP/1.1" 200 -
127.0.0.1 - - [10/Apr/2019 01:56:22] "GET /average_pop?city=Arkansas HTTP/1.1" 200 -
127.0.0.1 - - [10/Apr/2019 01:57:34] "GET /average_pop?city=Alabama HTTP/1.1" 200 -
127.0.0.1 - - [10/Apr/2019 01:58:37] "GET /average_pop?city=Delaware HTTP/1.1" 200 -
127.0.0.1 - - [10/Apr/2019 01:59:41] "GET /average_pop?city=Alaska HTTP/1.1" 200 -
127.0.0.1 - - [10/Apr/2019 02:00:29] "GET /average_pop?city=Arkansas HTTP/1.1" 200 -
127.0.0.1 - - [10/Apr/2019 02:01:04] "GET /average_pop?city=Arizona HTTP/1.1" 200 -
127.0.0.1 - - [10/Apr/2019 02:01:47] "GET /average_pop?city=District%20of%20Columbia HTTP/1.1" 200 -
127.0.0.1 - - [10/Apr/2019 02:02:30] "GET /average_pop?city=California HTTP/1.1" 200 -
127.0.0.1 - - [10/Apr/2019 02:03:42] "GET /average_pop?city=Colorado HTTP/1.1" 200 -
127.0.0.1 - - [10/Apr/2019 02:04:34] "GET /average_pop?city=Connecticut HTTP/1.1" 200 -
127.0.0.1 - - [10/Apr/2019 02:05:54] "GET /average_pop?city=Hawaii HTTP/1.1" 200 -
127.0.0.1 - - [10/Apr/2019 02:06:45] "GET /average_pop?city=Florida HTTP/1.1" 200 -
```

USING DEPLOYED WEB APP FROM AWS SERVER

NOTE: Make no changes to application.py or processing .py
In test.py change url given to you on AWS console of webapp

TEST.py

```
import requests as r
import threading

cities = ['Alabama',
 'Alaska',
 'Arizona',
 'Arkansas',
 'California',
 'Colorado',
 'Connecticut',
 'Delaware',
 'District of Columbia',
 'Florida',
 'Georgia',
 'Hawaii',
 'Idaho',
 'Illinois',
 'Indiana',
 'Iowa',
 'Kansas',
 'Kentucky',
 'Louisiana',
 'Maine',
 'Maryland',
 'Massachusetts',
 'Michigan',
 'Minnesota',
 'Mississippi',
 'Missouri',
 'Montana',
 'Nebraska',
 'Nevada',
 'New Hampshire',
```

```
 'New Jersey',
 'New Mexico',
 'New York',
 'North Carolina',
 'North Dakota',
 'Ohio',
 'Oklahoma',
 'Oregon',
 'Pennsylvania',
 'Puerto Rico',
 'Rhode Island',
 'South Carolina',
 'South Dakota',
 'Tennessee',
 'Texas',
 'Utah',
 'Vermont',
 'Virginia',
 'Washington',
 'West Virginia',
 'Wisconsin',
 'Wyoming']

link=" "    #link given to you on aws webapp console.
def get_avg(c):
    res = r.get('http://link)
    print(c + " average population: " + res.text)

for c in cities:
    threading.Thread(target=get_avg, args=(c,)).start()
```

THREADING AS A ENHANCER:

Usage of asynchronous requests enhances the cloud request process by usage of the threading process.
An synchronous request doesn't work on the "FIRST COME FIRST SERVE BASIS".
It identifies the faster networks and caters to their requests first to reduce load on the server by sending outputs as soon as the outputs are generated.

Furthermore, the multi-threading process speeds up the working of the system.
A multi-threaded application is an application whose architecture takes advantage of the multi-threading provided by the operating system. Such applications assign specific jobs to individual threads within the process and the threads communicate, using various means, to synchronize their actions. For example, a data processing application might be designed so that the graphical user interface is completely handled by one thread, while the actual work of the application is performed by another thread. This architecture would enable the complete separation of the business logic from the user-interface logic within the application.

CONCLUSION:

The census tracts calculated above simply the population based analysis process to great extent.
The easy and statistically accurate approaches used lead to solid and reliable reports which can be used in the development of a nation as well as the welfare of the people.

Cloud computing being the backbone of all future Information Technology Endeavours enhances the above process to extents of utter ease. The data and results being calculated and communicated using the cloud the technologies takes us one step closer to fully digitised nations, data and accounts.

The usage of this tutorial for the educating the people about cloud computing and data analysis together can generate curiosity in young minds and build towards a brighter future.