DIMENSIONALITY REDUCTION FOR DATA VISUALIZATION

Python Data Visualization Assignment

B9DA106
DATA VISUALIZATION

Module Title:	Data Visualization
Module Code:	B9DA106
Assessment Title:	Continuous Assessment Two
Mode of Submission:	Moodle
Student Names:	Taiwo Abdulrahman Lamidi(10545249) Ajay Kumar(10546004) Sanjay Kannan(10545006)

Submitted to;

Lecturer Name: Kunwar Madan

INTRODUCTION

Dimensionality reduction is used in data science to overcome some machine learning problems. For example, in the case whereby the dataset is having a large number of features, modelling and training such data will take time and also visualizing the data wouldn't be easy. Dimensionality Reduction is the way toward chopping down the quantity of features to the most significant ones. Dimensionality Reduction assists with the removal of unwanted information present in the data set. Additionally, it makes the preparation quicker and get great visualization. Dropping the dimensionality down to a few, make it conceivable to visualize the information on a 2d or 3d plot, with the goal that significant bits of knowledge can be picked up by examining these in clusters and association. The types of dimensionality reduction are:

- PCA (principal component analysis)
- t-SNE (t-distributed stochastic neighbor embedding)
- UMAP (uniform manifold approximation and projection)

PRINCIPAL COMPONENT ANALYSIS (PCA)

PCA (Principal Component Analysis) works by distinguishing the hyperplane which lies nearest to the information and afterward extends the data on that hyperplane while holding the vast majority of the variety in the data set. The main segments clarify the greatest measure of the change in the training set.

T-DISTRIBUTED STOCHASTIC NEIGHBOUR EMBEDDING (t-SNE)

T-DISTRIBUTED STOCHASTIC NEIGHBOUR EMBEDDING is utilized for dimensionality Reduction that is especially appropriate to visualize high-dimensional datasets. (t-SNE) reduces high dimensional datasets to a low dimensional diagram that holds tons of original data. It does as such by giving every data point an area 2 or 3-dimension guide. This procedure discovers clusters in data accordingly ensuring that an embedding keeps the importance of the data. t-SNE lessens dimensionality while attempting to keep comparative instances close and divergent cases separated

UNIFORM MANIFOLD APPROXIMATION AND PROJECTION (UMAP)

This is a broadly useful complex learning and dimension reduction algorithm. It is utilized for nonlinear dimensionality reduction technique and is successful for the visualization of clusters and their relative vicinities. The huge contrast with TSNE is versatility, it tends to be applied straightforwardly to scanty frameworks in this way disposing the need to applying any Dimensionality Reduction, for example, PCA as an earlier pre-preparing step

UMAP(HYPERPARAMETERS)

1. n_neighbours ; monitors and controls how UMAP equate local and global data structures.
2. min_dist: controls the distances between points.
3. n_components: It helps to know the dimensionality of the reduced dimension space.
4. Metric: This boundary controls how separation is registered in the encompassing space of the input data.

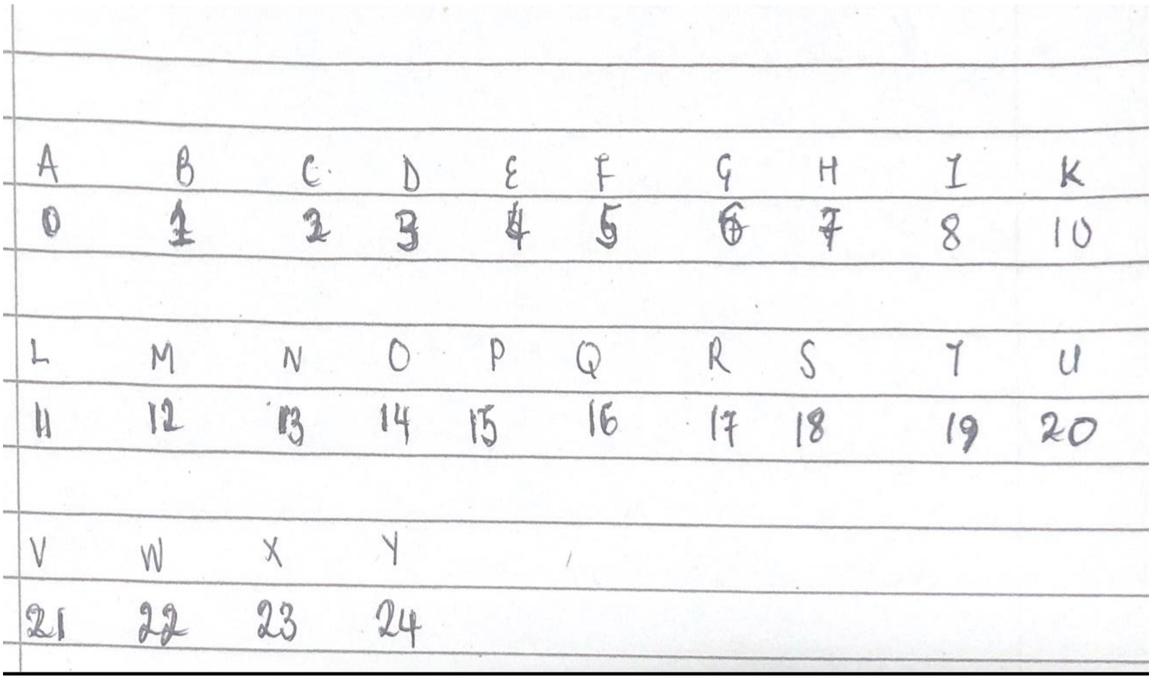
ASSIGNMENT EXPLAINATION

QUESTION 1 EXPLAINATION.

UMAP dimensionality-reduction technique will be used for visualizing natural groupings or clusters for the hand signs below;

Figure 1: Hand gestures in the American Sign Language representing 24 English alphabets

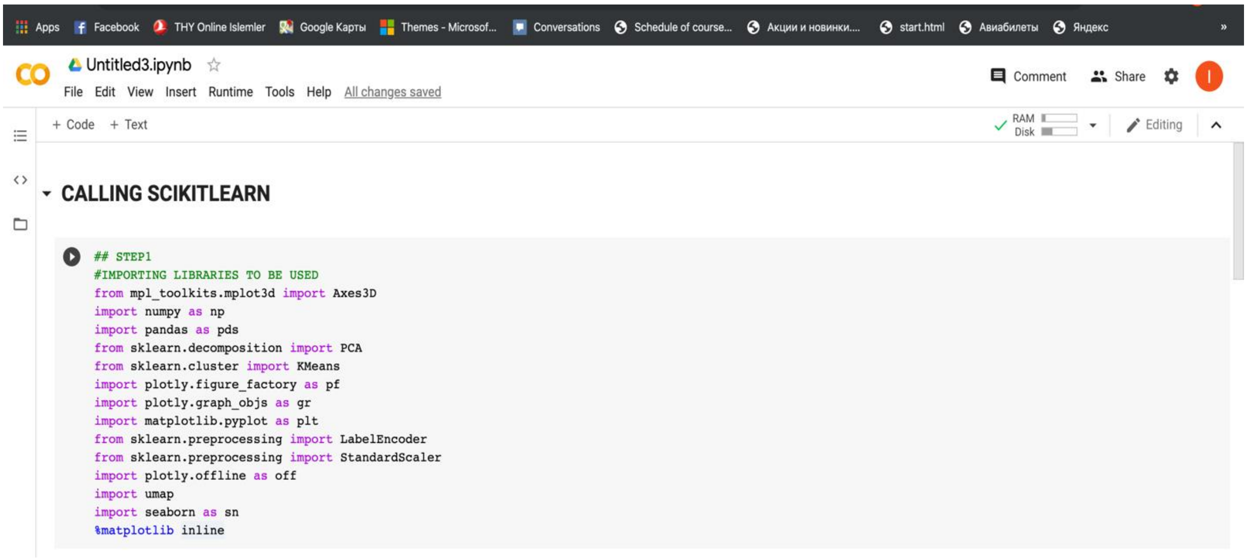
The data set is in its readable format (CSV) and each row in the dataset represents an image. Each independent variable represents a pixel (with value between 0-255). Label variable indicates which alphabet letter the image corresponds to. Letters A-Z are represented by numeric labels 0-25 (with no instances for 9=J or 25=Z as these alphabets cannot be represented by still images).



The above picture shows how the diagrams are going to be analyzed according to the clusters.

<u>Python codes and processes</u>

1. The first step is the calling and loading of all the needed libraries.



2. The next step includes the loading of data set. Faced the problem of loading it directly from the laptop. Had to upload it to a drive and with the help of few codes, was then able to put it in python. This step states the number of rows, data type, data memory size, data class and features.

```
import plotly.offline as off
[40] import umap
import seaborn as sn
%matplotlib inline
```

DATA IMPORTATION

```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

```
[39] auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
[4] downloaded = drive.CreateFile({'id':'19b48wn-5zshYaUVM69A8wfQ1Gwb2ZiDk'}) # replace the id with id of file you want to access
downloaded.GetContentFile('sign_mnist.csv')
```

```
[42] signmn = pds.read_csv('sign_mnist.csv')
print(signmn.head(1))
```

```
   label  pixel1  pixel2  pixel3  ...  pixel781  pixel782  pixel783  pixel784
0      3     107     118     127  ...       206       204       203       202

[1 rows x 785 columns]
```

```
[43] signmn.head()
```

The contains 10,000 records and 785 columns . The data belongs to the class Data frame and it ranges from 0- 255.

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14	pixel15	pixel16	pixel17	pixel18
0	3	107	118	127	134	139	143	146	150	153	156	158	160	163	165	159	166	168	170
1	6	155	157	156	156	156	157	156	158	158	157	158	156	154	154	153	152	151	149
2	2	187	188	188	187	187	186	187	188	187	186	185	185	185	184	184	184	181	181
3	2	211	211	212	212	211	210	211	210	210	211	209	207	208	207	206	203	202	201
4	13	164	167	170	172	176	179	180	184	185	186	188	189	189	190	191	189	190	190

5 rows × 785 columns

```
print(signmn.shape)
print(signmn.info())
print(signmn.describe())
```

```
(10000, 785)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Columns: 785 entries, label to pixel784
dtypes: int64(785)
memory usage: 59.9 MB
None
              label        pixel1  ...      pixel783      pixel784
count  10000.000000  10000.000000  ...  10000.000000  10000.000000
mean      12.310400    145.790700  ...    161.709500    160.692500
std        7.294599     41.912797  ...     63.005796     63.559782
min        0.000000      0.000000  ...      0.000000      0.000000
25%        6.000000    122.000000  ...    130.000000    129.000000
50%       13.000000    151.000000  ...    182.000000    182.000000
75%       19.000000    175.000000  ...    204.000000    204.000000
max       24.000000    255.000000  ...    255.000000    255.000000

[8 rows x 785 columns]
```

3. The next step involves classifying the dataset into features and label.

```
print(signmn.shape)
print(signmn.info())
print(signmn.describe())
```

```
(10000, 785)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Columns: 785 entries, label to pixel784
dtypes: int64(785)
memory usage: 59.9 MB
None
              label        pixel1  ...      pixel783      pixel784
count  10000.000000  10000.000000  ...  10000.000000  10000.000000
mean      12.310400    145.790700  ...    161.709500    160.692500
std        7.294599     41.912797  ...     63.005796     63.559782
min        0.000000      0.000000  ...      0.000000      0.000000
25%        6.000000    122.000000  ...    130.000000    129.000000
50%       13.000000    151.000000  ...    182.000000    182.000000
75%       19.000000    175.000000  ...    204.000000    204.000000
max       24.000000    255.000000  ...    255.000000    255.000000

[8 rows x 785 columns]
```

Untitled3.ipynb

File Edit View Insert Runtime Tools Help All changes saved

```
[8 rows x 785 columns]
```

input variables and output variables

```
[45] IPT= signmn.drop(['label'], axis= 1) #input variables
     OUP=signmn['label'] #output variable
```

```
[9] #print unique values of output variables

    print(np.unique(OUP))
```

```
[ 0  1  2  3  4  5  6  7  8 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]
```

```
[46] #CHEKING SANITY
     IPT.shape
```

```
(10000, 784)
```

```
[47] print ("The shape of IPT is " + str(IPT.shape))
     print ("The shape of OUP is " + str(OUP.shape))
```

```
The shape of IPT is (10000, 784)
The shape of OUP is (10000,)
```

```
[48] OUP_scld = StandardScaler().fit_transform(IPT)
     print(OUP_scld)
```

```
[[-0.92555597 -0.76586906 -0.62437311 ...  0.66073648  0.65537724
   0.64993246]
 [ 0.21973624  0.19996474  0.10972487 ... -0.95020514 -0.42394252
  -0.18396987]
 [ 0.98326438  0.96767879  0.91976401 ...  0.51718722  0.5125261
   0.53979441]
```

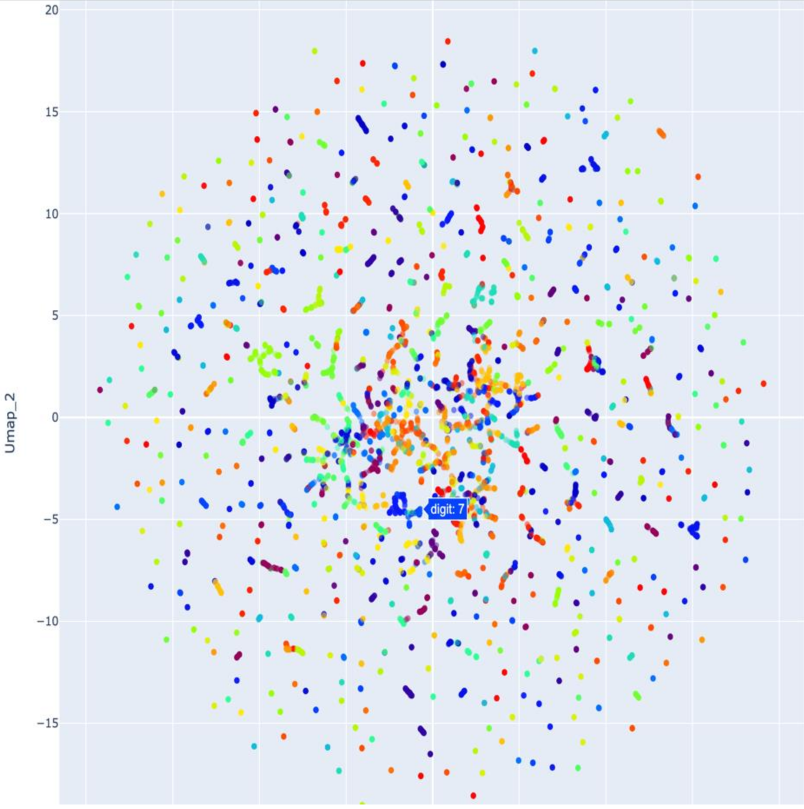
4. Next step is processes is normalizing and standardizing the features to fit Gaussian distribution (ensures the mean of the distribution is between 0 and 1).

```
[48] OUP_scld = StandardScaler().fit_transform(IPT)
     print(OUP_scld)
```

```
[[-0.92555597 -0.76586906 -0.62437311 ...  0.66073648  0.65537724
   0.64993246]
 [ 0.21973624  0.19996474  0.10972487 ... -0.95020514 -0.42394252
  -0.18396987]
 [ 0.98326438  0.96767879  0.91976401 ...  0.51718722  0.5125261
   0.53979441]
 ...
 [-0.42449063 -0.39439452 -0.42186333 ...  0.7723859   0.76648369
   0.7600705 ]
 [ 0.43447853  0.3980845   0.38817582 ...  0.35768805  0.3538026
   0.35098634]
 [ 0.00499395  0.02660996  0.00846997 ...  0.29388839  0.27444086
   0.27231631]]
```

5. This step includes the implementation of UMAP. UMAP was used because of the speed. Separate values were used for the n_neigbour and min_dist and the output was compared to

each other before we ended up using n_neigbour of 5 and min_dist of 0.1.. The most challenging is trying to get the proper n_neigbour, so as to get good picture from the visualization.



From the above diagram, it is that the sign languages are distant from one another except for the ones that are similar in hand gestures which were grouped together.

- It is interesting working with UMAP
    I. Because of its high speed which makes it easier to change the parameters
    II. It preserves the global structure

PERSONAL CONTRIBUTION:

My (Lamidi) main work was writing the algorithms and reporting. Also PowerPoint and audio record preparation. I also analysed the insight gotten from the visualization.

QUESTION 2 EXPLAINATION:

T-SNE Dimension Reduction technique will be used for visualizing natural groupings or clusters for the Customers dataset provide below:

Dataset 2: customers.csv

This dataset refers to customer data of a telecom company. Each row corresponds to a customer.

Number of Instances: 7033

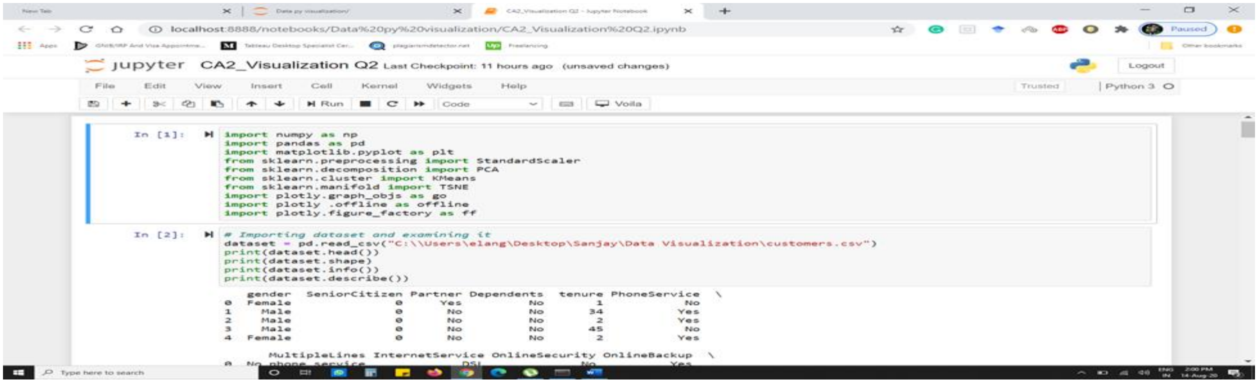
Number of Variables: 19 independent variables

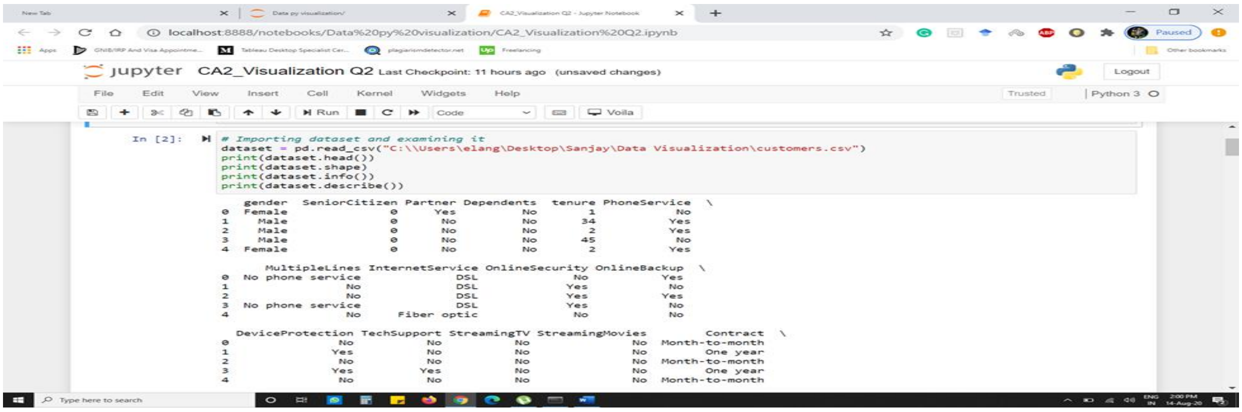
About the data set: the initial dataset has many categorical columns, so we had to change them into numerical columns and some of these are ordinal variables and some are nominal variables and for nominal variables we create dummy columns.

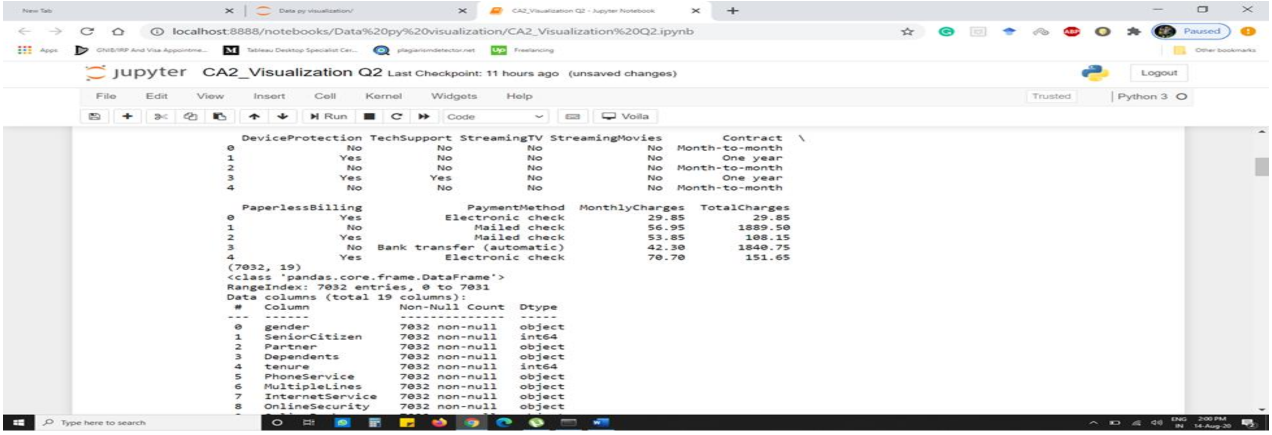
PYTHON CODING AND EXECUTATION STEP BY STEP:

STEP 1: **Importing libraries, importing data and quick glance:**

Importing all the libraries and library functions which are used like standard scalar etc, Next importing the data file to analyse the dataset. Libraries like pandas, NumPy, SK learn, Plotly for visualization of dataset are being used.





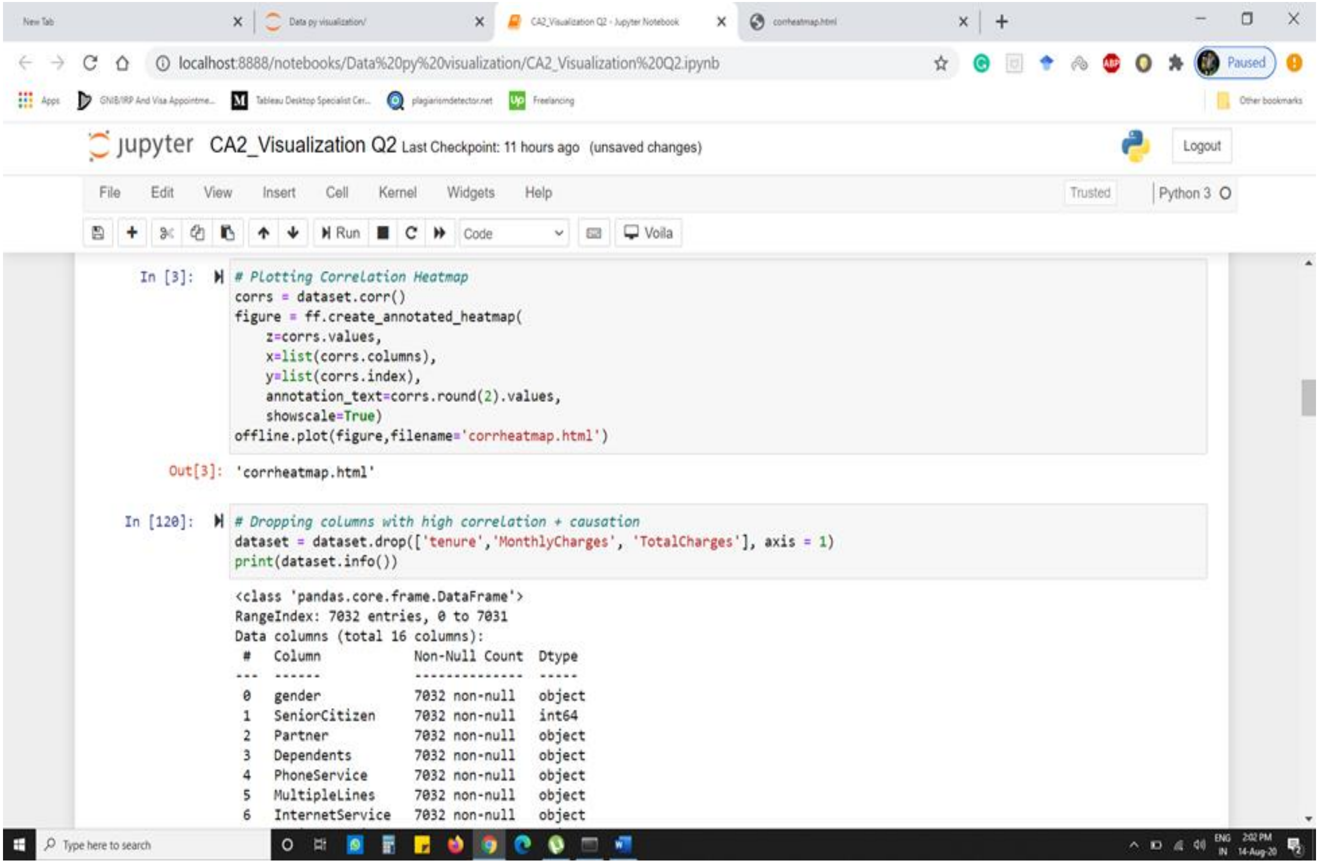


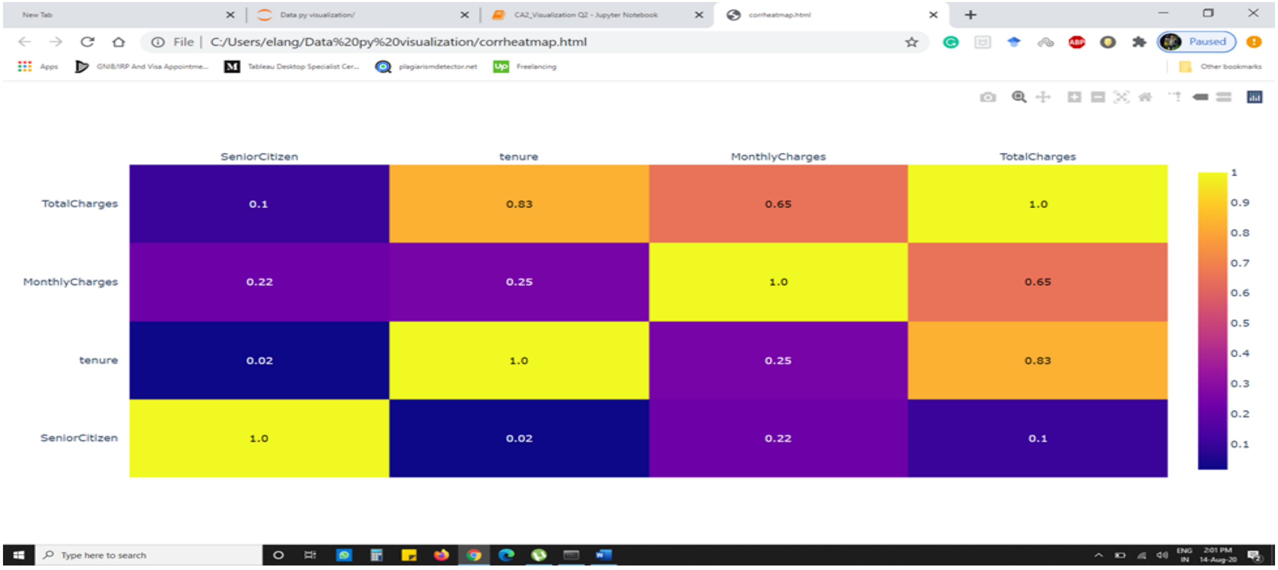
After importing the dataset, Some usually python coding have been used to have a quick glance on dataset for understanding and knowing about the data type of each column and for further proceedings towards step 2.

STEP 2: Heat Map, Correlation and Dropping columns:

Next step is to check the columns whether there is some correlation between columns or not, if there is high negative or high positive correlation between columns then model and algorithm will be biased and then the visualization will not be robust. To Eliminate the highly correlative columns we use heat map to check the values as shown below: "the monthly charges, tenure, and total charges has high correlation values"

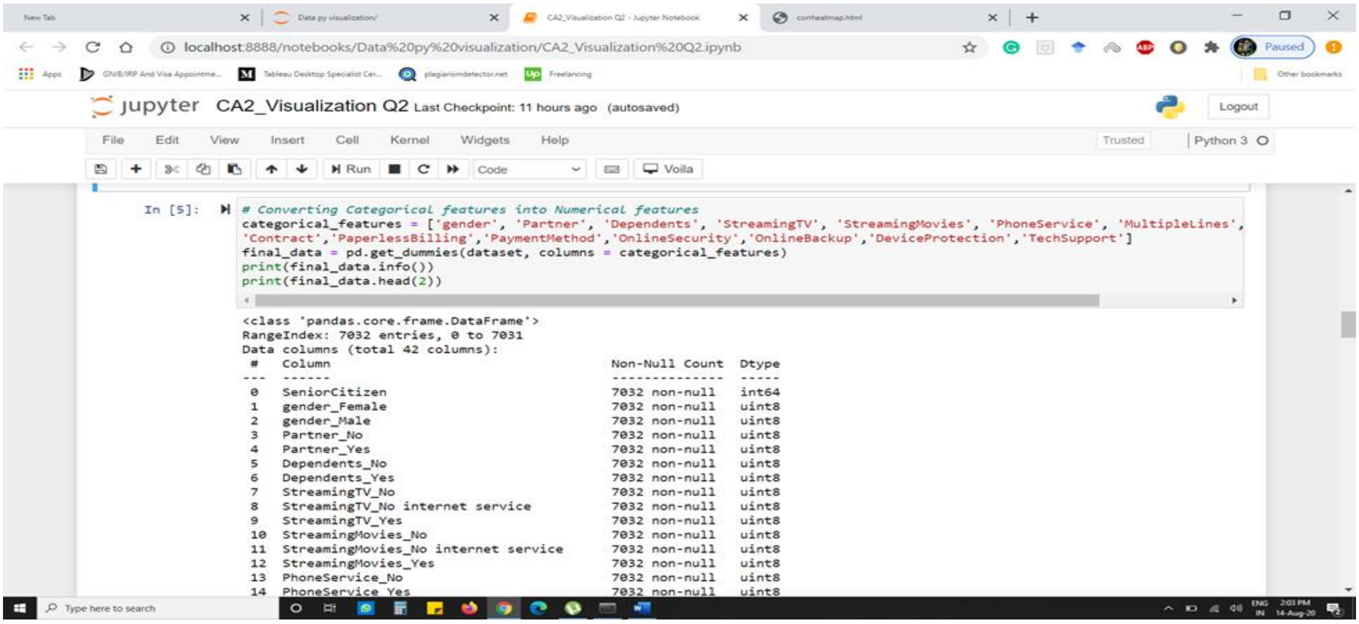
Hence those three columns are drop not only on the basics of correlation but also causation as well.





STEP 3: Converting categorical columns and creating dummy columns:

After dropping correlative columns, Next step is to covert the remaining categorical columns into numerical columns because no machine learning algorithm can perform operation on object data type. And for the nominal columns need dummy columns are being created as shown in figures:

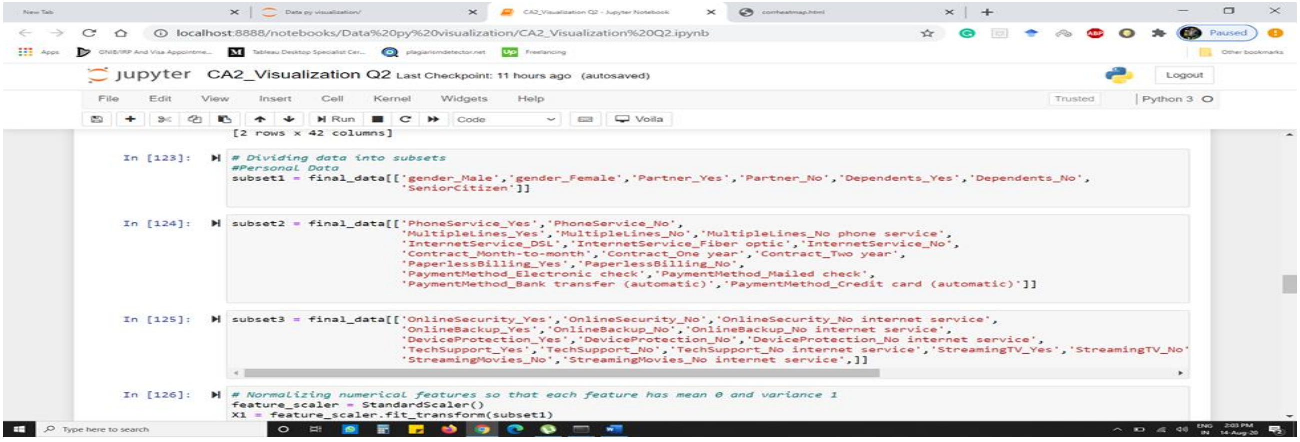


STEP 4: Dividing mass dataset into subset dataset:

As known, T-SNE is a very slow processing algorithm, so it better to divide this huge dataset into sub-categories and then perform the algorithms.

This dataset has been divided into three sub datasets:

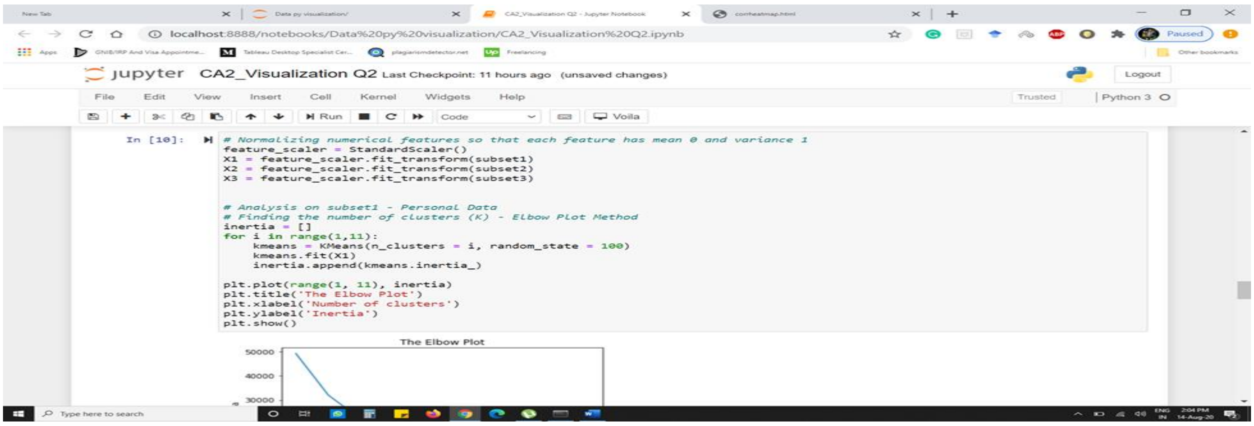
1. First subsets having personal data of customers.
2. Second subset is the hardware and different types of services provide from company to the customers.
3. Third subset is the online services provided from the company to the customers.

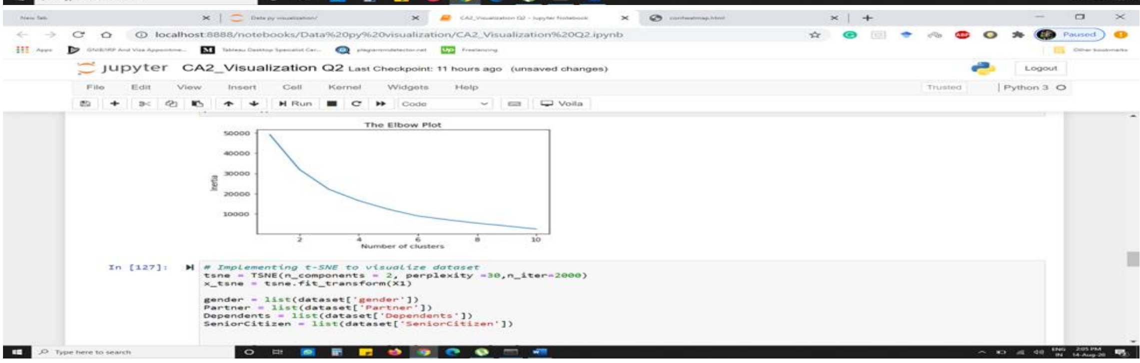


STEP5: Normalizing numerical and elbow plot:

Before performing the t-sne algorithm we need to make sure that the all numerical have same mean range and variance which is called normalizing, if we don't normalize the numerical then algorithm will give more significance to one variable that has higher value and we will lose information in visualization .

Furthermore, we perform elbow Plot we have an idea that how many clusters are in our subset.

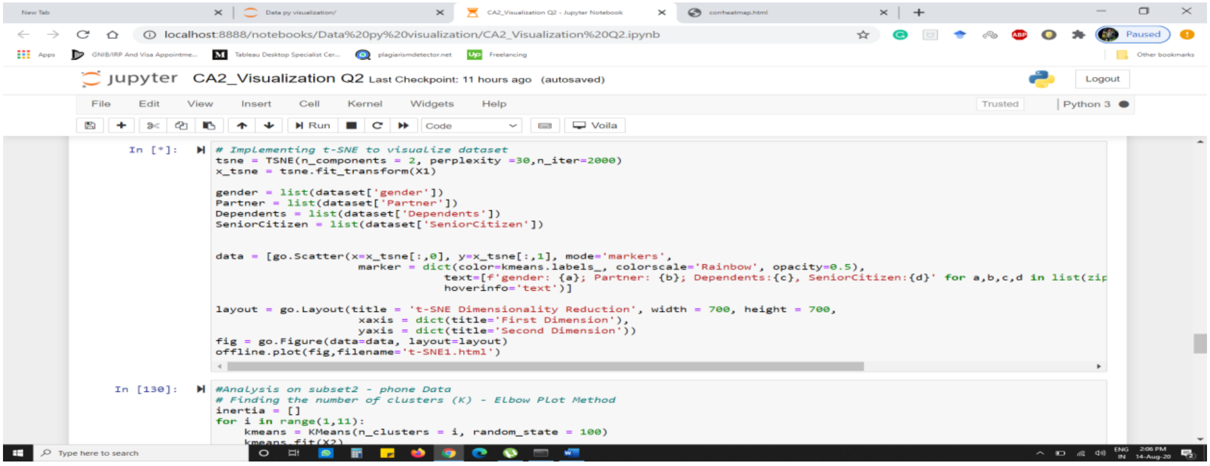


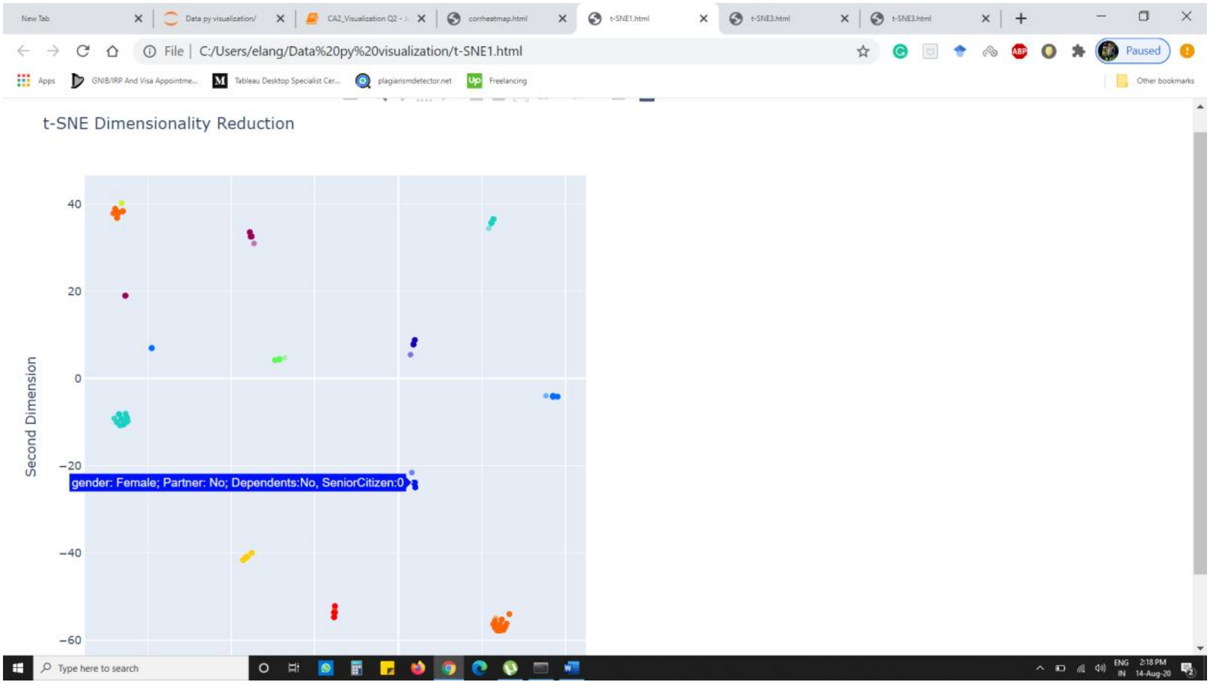


STEP 6: Perform the T-SNE Algorithm and plot visualization:

In this report we included the algorithm and visualization perform on first subset which is about the personal data of customers.

Note: all visualization has been included in the main zip with all in format html and python coding in .pynb file





<u>**PERSONAL CONTRIBUTION:**</u> Sanjay Kannan(10545006) work was writing the algorithms and PowerPoint and audio record preparation. Ajay Kumar worked on reporting of question 2, analysed the insight gotten from the visualization and audio recording.