Configuration Manual

MSc Research Project
MSc in Data Analytics

Samya Bose
Student ID: 18180523

School of Computing
National College of Ireland

Supervisor: Dr. Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Mr. Samya Bose		
Student ID:	18180523		
Programme:	MSc in Data Analytics	**Year:**	2020
Module:	MSc Research Project		
Supervisor:	Dr. Vladimir Milosavlijevic		
Submission Due Date:	17th August, 2020		
Project Title:	Correlation between Solar Wind Ionic Variation and Earthquake		
Word Count:	5126	**Page Count:**	15

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Samya Bose

Date: 16th August, 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Samya Bose

18180523
MSc in Data Analytics
School of Computing
National College of Ireland

1. Introduction

The configuration manual includes information about the hardware and software requirements, the libraries that needs to be installed. This manual also mentions the steps that need to be followed for the glitch free running of the code. Also, it gives information about the datasets, and description of the variables, and the changes to the dataset. Following this manual will result is replication of the results.

2. Hardware Specifications

- Operating System: Windows 10 Home Edition, 64-bit
- Processor: Intel(R) Core(TM) i5-8300H @ 2.30GHz
- RAM: 16GB

3. Software Specifications

- Anaconda Navigator (1.9.7)
- pgAdmin (for PostgreSQL)
- Jupyter Notebook (6.0.0)
- Jupyter Lab (1.0.2)
- Python (3.5)
- Power BI

4. Environment Setup

The project was completed using Python coding language. This is because Python consists of a large number of libraries and packages which makes working with it easier. Second advantage is the availability of Jupyter notebook and lab using Anaconda, which allows the running of code snippets without running the whole code. This makes the process a whole lot faster. Anaconda includes different IDEs for Python like Jupyter Lab, Jupyter notebook, Spyder etc. The following screenshot shows the Anaconda and Jupyter Lab interface.

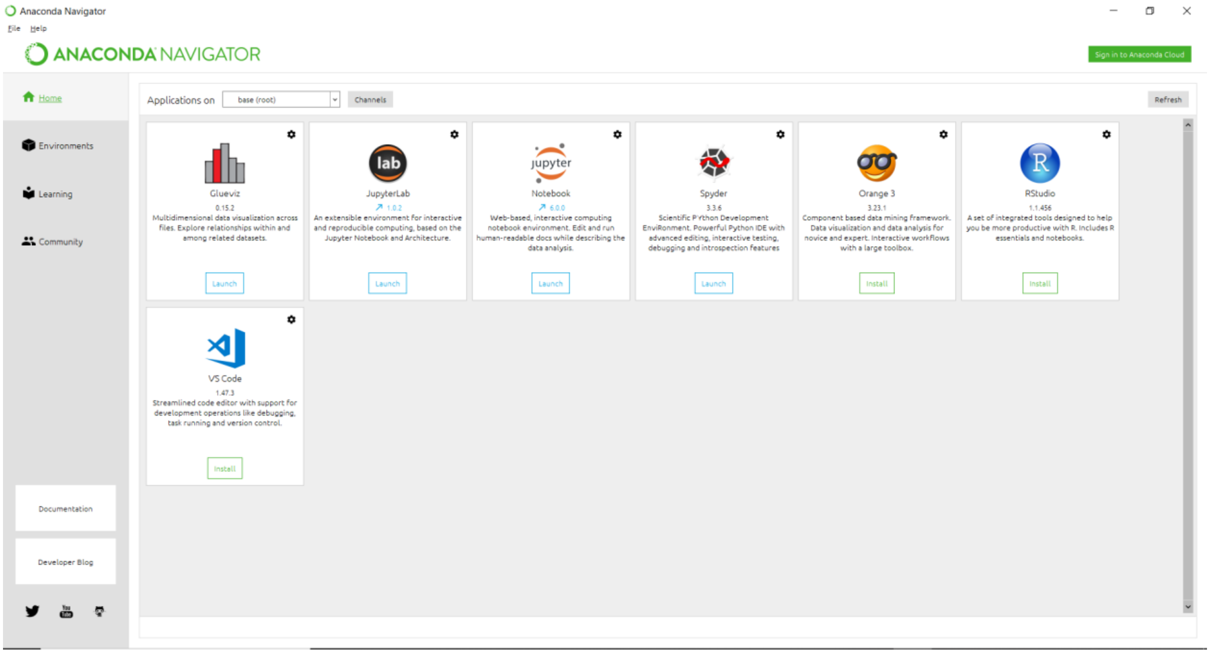

Fig 1: Anaconda interface

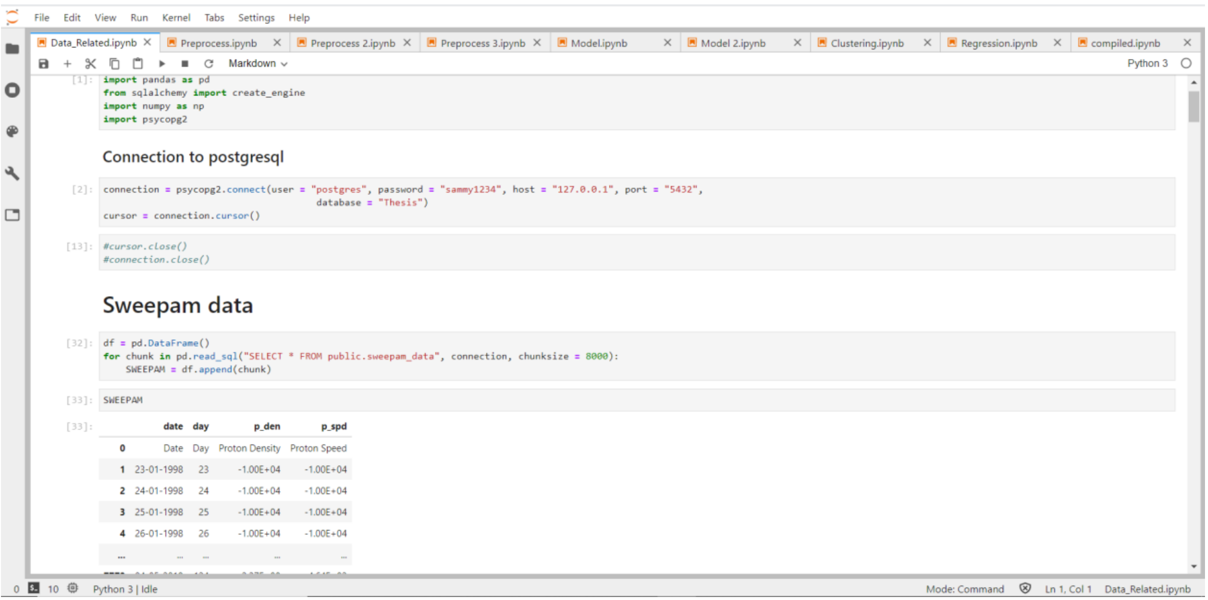

Fig 2: Jupyter Lab interface

Before installing Anaconda on the machine, Python needs to be installed. I have used version 3.5 of Python, since it is one of the widest used versions. Python can be installed from the following link: https://www.python.org/downloads/ . After download and installation of Python, Anaconda can be installed in the same location as Python was installed, otherwise it can lead to creation of multiple Python environments. Anaconda can be downloaded and installed from the following link: https://www.anaconda.com/products/individual .

Next, for the database part, PostgreSQL needs to be installed and for the interface, I have used pgAdmin, which gives an interactive way to create databases and view them instead of a console view.

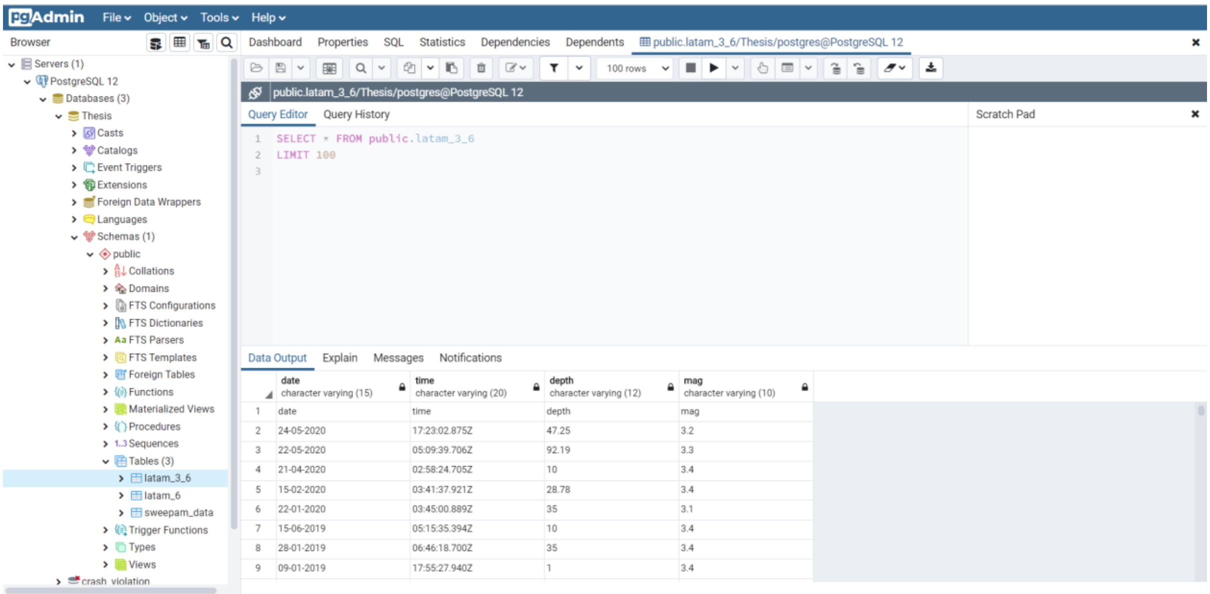

Fig 3: pgAdmin interface

PostgreSQL and pgAdmin can be downloaded from the following links:

- Postgresql: https://www.postgresql.org/download/
- pgAdmin: https://www.pgadmin.org/download/

5. Data Source

The data for this project has been acquired from 2 sites. First for earthquake data, I have used USGS official site, and for solar wind data, I have used CalTech's database for ACE satellite data. Following are the links to the data sources:

- USGS: https://earthquake.usgs.gov/earthquakes/search/
- ACE: http://www.srl.caltech.edu/ACE/ASC/level2/lvl2DATA_SWEPAM.html

6. Libraries

The following is a list of libraries along with their uses which are required to be imported to the Python notebook. If any of these are not installed, then pip install might be required first for the import to be possible. Details are given as below.

- Pandas: Most important library as we will be using pandas for everything related to the dataframe. *(Use: import pandas as pd)*
- Numpy: Numpy will be required to do calculations, and also for doing list and array operations. *(Use: import numpy as np)*
- Matplotlib: This library will be required to create practically all the graphs in the study. *(Use: import matplotlib.pyplot as plt || %matplotlib inline)*
- Operator: This library will be used for few operations in the loops, mainly for the optimisation purpose. *(Use: from operator import itemgetter)*
- Math: Used for all the mathematical operations. *(Use: import math)*

- Sklearn: Main library for all the functions of Scikit learn. Provides an array of predefined functions for machine learning. *(Use: from sklearn.preprocessing import MinMaxScaler || from sklearn.cluster import KMeans || from sklearn.decomposition import PCA)*
- Plotly: Used for high definition graphs. In this project used for generating the cluster graphs. *(Use: import plotly.offline as pyo || import plotly.graph_objs as go || from plotly.offline import init_notebook_mode, iplot || pyo.init_notebook_mode())* [Note: plotly graphs won't show or load in Jupyter Lab. For that purpose, Jupyter Notebook to be used]
- Sqlalchemy and Psycopg2: Used to get the data from text file to PostgreSQL and then integrating PostgreSQL with python to get the data as pandas dataframe. *(Use: from sqlalchemy import create_engine || import psycopg2)*

7. Project

- **Data Load**

The solar wind data was received as a text file. This had description of the variables and their units. This data needed to be cleaned and date needed to be added. Following is a screenshot of the initial data.

```
BEGIN METADATA
SWEPAM Daily Averaged Solar Wind Parameters
Data downloaded from ACE Science Center on Tue Jun  9 16:48:20 2020
SWEPAM Team Software Version: 3.20
SWEPAM Team Time/Date Processed:
ACE Science Center Level2 Software Version: 1.4
ACE Science Center Processing Date/Time: Mon Oct 21 17:39:37 PDT 2019


Note: All timestamps are UT, and refer to the start of the time period.
year,day,hr,min,sec: year, day of year, hour of day, minutes, seconds.
fp_year       : fractional year.
fp_doy        : fractional day-of-year.
ACEepoch      : seconds since Jan 1 00:00:00 UT 1996.
```

```
day proton_density proton_speed
BEGIN DATA
 23 -9.9999e+03 -9.9999e+03
 24 -9.9999e+03 -9.9999e+03
 25 -9.9999e+03 -9.9999e+03
 26 -9.9999e+03 -9.9999e+03
 27 -9.9999e+03 -9.9999e+03
 28 -9.9999e+03 -9.9999e+03
 29 -9.9999e+03 -9.9999e+03
 30 -9.9999e+03 -9.9999e+03
 31 -9.9999e+03 -9.9999e+03
 32 -9.9999e+03 -9.9999e+03
 33 -9.9999e+03 -9.9999e+03
 34 -9.9999e+03 -9.9999e+03
```

Fig 4: Initial solar wind data

The data in txt file is imported to PostgreSQL and created into a dataframe readable format. Once that is done, the database is connected to Python, which is shown as per the code below.

Connection to postgresql

```
[2]:  connection = psycopg2.connect(user = "postgres", password = "sammy1234", host = "127.0.0.1", port = "5432",
                                     database = "Thesis")
      cursor = connection.cursor()

[13]: cursor.close()
      connection.close()
```

Fig 5: Connection to PostgreSQL

Once the database is connected, the data is pulled to Python and stored to a pandas dataframe, as shown in the below screenshot. Once the data is pulled, it is a good practice to do the cursor.close() and connection.close() commands, which will disconnect the database. Keeping the database connected can lead to memory leaks, which might end up using more RAM of the system.

Sweepam data

```
[32]: df = pd.DataFrame()
      for chunk in pd.read_sql("SELECT * FROM public.sweepam_data", connection, chunksize = 8000):
          SWEEPAM = df.append(chunk)
```

```
[33]: SWEEPAM
```

[33]:

	date	day	p_den	p_spd
0	Date	Day	Proton Density	Proton Speed
1	23-01-1998	23	-1.00E+04	-1.00E+04
2	24-01-1998	24	-1.00E+04	-1.00E+04
3	25-01-1998	25	-1.00E+04	-1.00E+04
4	26-01-1998	26	-1.00E+04	-1.00E+04

Fig 6: Storing the data to Pandas dataframe

- **Exploratory Data Analysis**

For the EDA part, subset creation of the dataset is needed since it has over 6000 rows of data. This is mainly to create graphs and understand the trend of solar winds before and after an event of earthquake. Creation of subsets is as per the following screenshot.

```
[8]: df_dict = {}
     for i in range(len(latam.date)):
         dat1 = latam.date[i]
         #df_list.append('df'+str(i))
         for j in range(len(sweepam.date)):
             dat2 = sweepam.date[j]
             if (dat1 == dat2):
                 #print("yes")
                 df_dict['df'+str(i)] = sweepam.iloc[j-11:j+10]
                 #df_name = 'df'+ str(i)
```

```
[48]: df_dict['df0']
```

Fig 7: Creating subsets of the dataset

The above code chunk splits the dataset based on the date of the earthquake. It takes 20-day duration of solar wind data i.e. from 10 days before the earthquake to 10 days after, and creates lists, and stores into a dictionary.

Next, the creation of graphs is shown as per the screenshot below.

```
[63]: fig,ax1 = plt.subplots(figsize=(30,98))
      ax2 = ax1.twinx()
      plt.subplots_adjust(hspace = 0.5)
      for i in range(len(df_dict)):
          #print(df_dict['df'+str(i)])
          data = df_dict['df'+str(i)]
          mark = latam.loc[[i],['date']]
          plt.subplot(41,6,i+1)
          ax1=plt.gca()
          ax2=plt.gca()
          ax1.plot(data['date'],data['p_den'], color = 'blue')
          ax2.plot(mark,5,color = 'red',marker = 'o',markersize = 4)
          plt.xticks(rotation = 30)
```

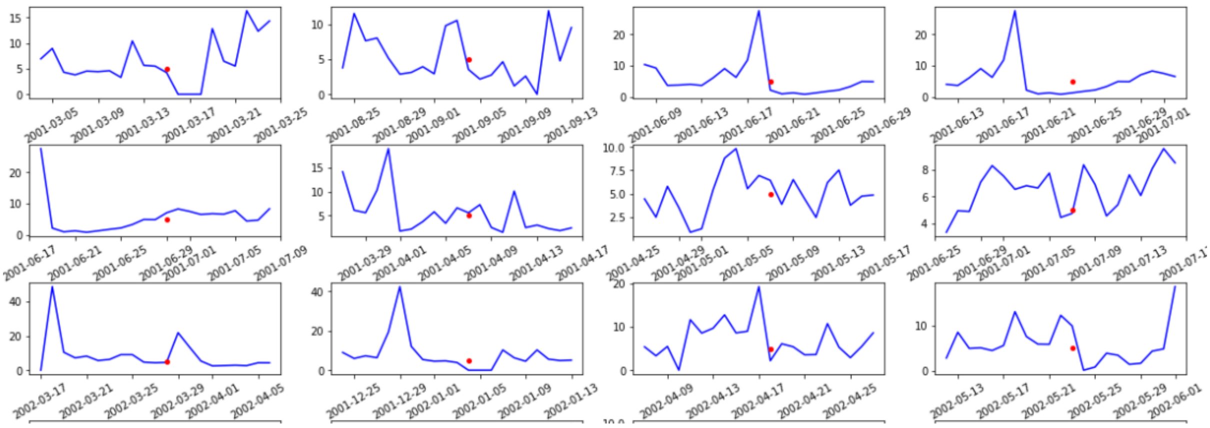

Fig 8: Creation of EDA graphs

The blue line represents the trend of the proton density, and the red dot shows the event of an earthquake. The red dot is not a constant location because I have hardcoded it to be a specific value just to mark the mid-point.

- **Model Build and Fitting: Clustering**

The following code was used for the clustering and graph creation.

```
In [4]: patRecDF2 = pd.read_csv('patRecDF3.csv')
```

```
In [14]: #patRecDF2.to_csv(r'C:\Users\samya\PycharmProjects\Thesis\ref.csv',index = False)
```

```
In [25]: patRecDF2 = patRecDF2.drop(columns = ['date'], axis = 1 )
```

```
In [26]: cluster = KMeans(n_clusters = 4)
```

```
In [32]: patRecDF2['cluster1'] = cluster.fit_predict(patRecDF2[patRecDF2.columns[0:2]])
```

Fig 9: Clustering

The scikit learn provides k-means function for clustering. The default parameters are n_clusters = 8, n_init = 10, max_iter = 300, tol = 0.0001. I have changed only the number of clusters to 4. After the clusters are created, and added to the dataset, PCA is done to get the x and y coordinates. Mentioned columns have been taken for the PCA.

```
In [33]: cols = patRecDF2.columns[0:4]

In [29]: cols

Out[29]: Index(['diff1', 'diff2', 'mag', 'cluster1'], dtype='object')

In [34]: pca = PCA(n_components = 2)
         patRecDF2['x'] = pca.fit_transform(patRecDF2[cols])[:,0]
         patRecDF2['y'] = pca.fit_transform(patRecDF2[cols])[:,1]
```

Fig 10: Code for PCA

Next, visualisations of the clustering is done using plotly graphs, as shown in the screenshots.

```
In [35]: trace0 = go.Scatter(x = patRecDF2[patRecDF2.cluster1 == 0]["x"],
                             y = patRecDF2[patRecDF2.cluster1 == 0]["y"],
                             name = "Cluster1", mode = "markers",
                             marker = dict(size = 10, color = "rgba(15,152,152,0.5)",
                                           line = dict(width = 1, color = "rgb(0,0,0)")))
         trace1 = go.Scatter(x = patRecDF2[patRecDF2.cluster1 == 1]["x"],
                             y = patRecDF2[patRecDF2.cluster1 == 1]["y"],
                             name = "Cluster2", mode = "markers",
                             marker = dict(size = 10, color = "rgba(180,18,180,0.5)",
                                           line = dict(width = 1, color = "rgb(0,0,0)")))
         trace2 = go.Scatter(x = patRecDF2[patRecDF2.cluster1 == 2]["x"],
                             y = patRecDF2[patRecDF2.cluster1 == 2]["y"],
                             name = "Cluster3", mode = "markers",
                             marker = dict(size = 10, color = "rgba(132,132,132,0.8)",
                                           line = dict(width = 1, color = "rgb(0,0,0)")))
         trace3 = go.Scatter(x = patRecDF2[patRecDF2.cluster1 == 3]["x"],
                             y = patRecDF2[patRecDF2.cluster1 == 3]["y"],
                             name = "Cluster4", mode = "markers",
                             marker = dict(size = 10, color = "rgba(148,148,13,0.8)",
                                           line = dict(width = 1, color = "rgb(0,0,0)")))
         data = [trace0,trace1,trace2,trace3]
         pyo.iplot(data)
```

Fig 11: Code for creation of the graph

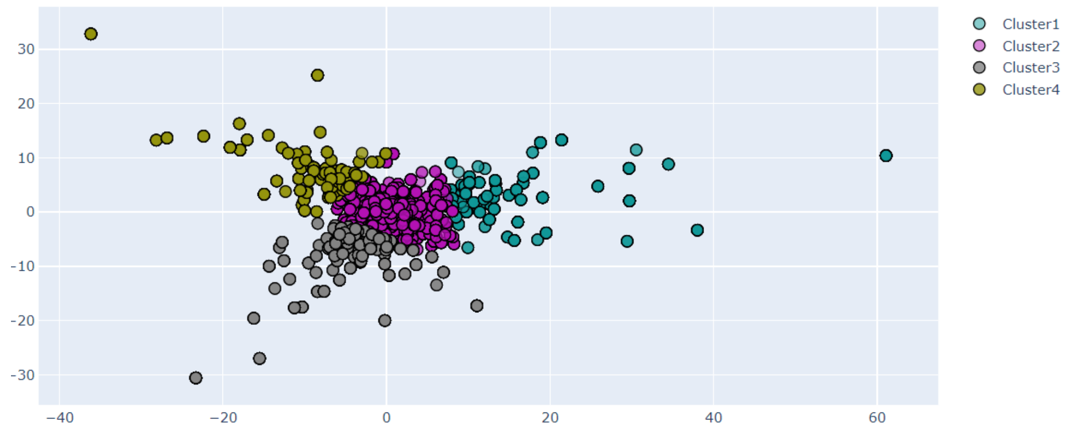

Fig 12: Clustering result

Finally, graphs were generated using Power BI for the evaluation. The datasets used for the graph generation was the same as that of clustering.

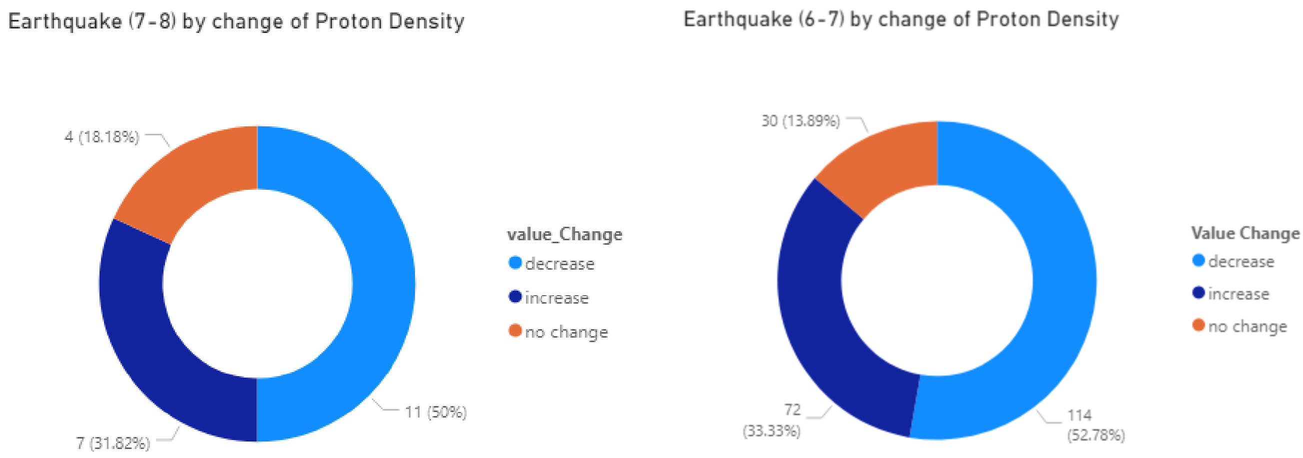

Fig 13: PowerBI graphs