International Institute of Information Technology Bangalore

ज्ञानमुत्तमम्

International Institute of Information Technology, Bangalore

Reading Elective Report

Authors:
Saikiran Reddy
IMT2017030
Rohith
IMT2017044

Supervisor:
Prof. Jaya Sreevalsan Nair

June 23, 2021

Figure 1: Our Web Application.

Contents

List of Figures

1 Introduction

1.1 Problem Statement

Dataset: The GenCat Mobile Coverage app is an initiative of the Government of Catalonia to crowdsource data collection on the state of mobile telephone network coverage in Catalonia. The source of the dataset is here. There is two main analysis we can from this data.

- We can use it to analyze the quality of mobile coverage in Catalonia of the four main operators (Movistar, Vodafone, Orange, and Yoigo) and filter data according to the technology used (2G, 3G, or 4G).
- Additionally the data enables the identification of areas in Catalonia that need to improve their mobile coverage with the final goal of helping to improve the efficiency of basic services for the general public.

The dataset contains each day's data from 2015 and is still being updated daily. Each data file contains a day's cell coverage data which is approximately 2.11 GB. We have two variants of data, US and EU.

Each day's dataset contains 11,744,914 rows, 21 columns. These columns include information such as

- date and time
- network provider and operator
- net type
- the location (postal code, town name)
- activity (whether the user is IN_VEHICLE, STILL, ON_FOOT, TILTING, UNKNOWN, ON_BICYCLE)
- the precise latitude and longitude along with the position_geometry

Other metrics are also provided such as upload and download speeds, the strength of the signal, precision, and status.

This comprehensive dataset really gives an insight into the network coverage of Catalonia which is something we wish to visualize in our project.

The demo Application which we have deployed on the gh-pages for the Vue Arcgis Application can be viewed here. The Entire Codebase for all the frameworks for this elective can be found in this github link.

2 Extraction

There is no straightforward way for downloading the dataset. We were asked to provide our international credit card details to just query and download the dataset. So the first major hurdle was to find ways to download the required dataset.

After researching for quite a while to find a suitable solution, we stumbled upon the python script, bigquery-downloader, which enabled us to query the dataset and download it for free of cost. This enabled us to query and download the data as a CSV file which was later consumed by our Apache cluster. We are planning on making this extraction process programmatic such that at a later stage we can provide it as an API for the web app such that users can automatically query and download data and visualize it.

2.1 Procedure

The following steps were taken to query and download a week's worth of dataset.

1. The Script requires a google cloud account with bigquery API enabled and its credential details as a JSON file.
2. The SQL Query we used was to query the entire table of the dataset.

By providing these two files we were able to query data backward from present-day to a week's worth of data which was downloaded a tar.gz file.

In the future, we want to automate this process on click in the web application if the query the user asks for data in a further time scale which triggers the entire extraction and then provides the necessary dataset to the user.

3 Setting Up Environment

We are using Apache Spark which is a unified analytics engine for large-scale data processing. We are specifically running and managing the cluster in Spark Standalone Mode. As this was the first time we are using a cluster technology we encountered various challenged and difficulties when setting up the cluster environment across multiple devices. Specifically setting up in cross-platform environments like Linux and windows was very hard and the documentation was too much detritus and didn't clearly explain how to configure the environment. We are providing a brief description of the challenges we faced and how we overcame them.

3.1 Procedure

3.2 Challenges and Solutions

These are problems we faced during the initial setup of the cluster

- The first challenge we faced in a Linux environment after installing the Apache spark package was when starting the master node it produced an insufficient resource error which after searching the stack overflow for a long time found out that it was due to lack of permissions for a working directory which is usually read and write by the root only. As running an elevated privileged process was not advisable we removed and installed a stand-alone one in the local directory instead of the package manager and using **chown** we changed required directories user groups and permissions.
- Similarly in windows environment as the majority of spark commands were returned as bash shell scripts there was no clear way even in the documentation of starting and running the master and slave nodes of the cluster in windows. After again researching we found that the only way to run the cluster nodes in windows was through **spark-class** binary which had to be provided additional arguments as specifying the type of node, resources, etc.

```
spark-class org.apache.spark.deploy.master.Master in\
spark/bin file to start master

 spark-class org.apache.spark.deploy.worker.Worker\
 spark://ip:port
```

- Also in the windows environment we were getting another error which caused the cluster node to crash. After again searching based on the error we found out about installing Hadoop binary with win-utils exe file which was required to run the spark-standalone cluster. This solved the installing of packages in both the environment.
- After initializing the master and slave nodes in the cluster and when trying to load the dataset we found out there was inherent resource allocated to the cluster. Even After adding all the specified environment variables mentioned in the docs required about the resource allocation we still couldn't allocate it at all. After searching quite a lot
- Each Node was utilizing 1 GB memory even after setting spark Context driver memory. We had to set it as a global environment variable in the spark-defaults.conf file.

4 Plotting the Data

We have used Arcgis as our plotting library. It is a powerful library useful for spatial analysis, mapping, and GIS. Installation was easy using pip. The Main Challenge we faced was the API of Arcgis required our data to be spatially enabled and integrating it into our Vue Web application.

4.1 Using Apache Spark

We use the Apache spark cluster by using the pyspark library. To use it in the jupyter notebook we had required another library which is called findspark which will point to the spark installation directory.

```
import findspark
findspark.init("/home/saikiran/tools/spark")
```

Then we create a spark configuration which contains all the settings necessary like spark cluster url to connect, application name, memory limits etc.

```
conf = pyspark.SparkConf().setMaster('spark://localhost:7077')
.setAppName("Big Data Visualization")
.set("spark.sql.execution.arrow.pyspark.enabled", "true")
.set('spark.executor.memory','10g')
.set('spark.driver.memory','20g')
.set("spark.driver.maxResultSize",0)
```

Then we create a Spark Context with which we can initialize spark SQL Context which is used to create spark dataframe which helps us in utilizes the cluster resources to visualize the dataset.

```
sc = pyspark.SparkContext()
sc.setLogLevel('ERROR')
sql = pyspark.SQLContext(sc)
df = sql.read.csv("./feb03.csv", sep='\t', header = True)
```

4.2 Loading the Dataset

We are loading our dataset which is in the form of CSV into a Spark dataframe. We initially used the old SQL Context reading method which caused a deprecation error. We later found in the tutorial online to use latest read_csv method. Also, we found out later that file is using a tab separator instead of a comma delimiter.

We enabled the environment variable *spark.sql.execution.arrow.pyspark.enabled* which allows us to convert our spark SQL context dataframe to pandas dataframe. Using Pandas Dataframe we convert our dataframe into a spatially enabled dataframe using

```
pdf = pd.DataFrame.spatial.from_xy(pdf,
    x_column = 'long', y_column = 'lat')
```

Then we plot the dataframe by running an ArcGIS map instance and binding it to the pandas dataframe.

```
pdf.spatial.plot(map_widget= m)
```

We can also plot a time series animation using ArcGIS API which provides a time slider and animation. It requires us to have a DateTime object which can be created by using the date and hour columns in the Data CSV File. Incidentally, pandas allow for to specify columns that can be taken as arguments for DateTime parsing.

```
pd.read_csv("./feb03.csv", sep='\t',error_bad_lines=False,
  parse_dates=[['date', 'hour']])
```

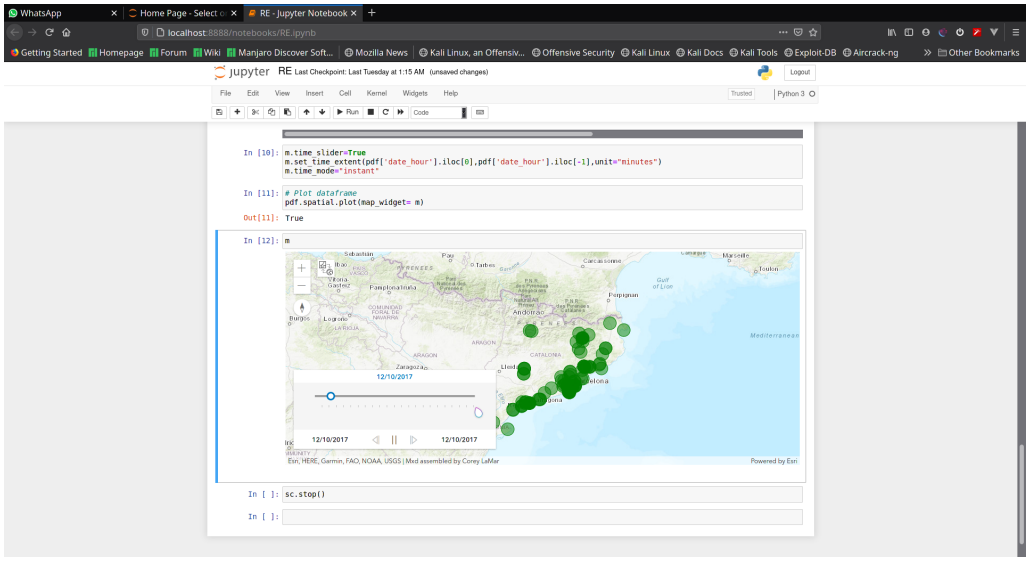

Figure 4.1: Temporal Vizualisation with Time Slider

4.3 Loading the Dataset in WebApp

For the Webapp we use ArcGis Javascript API with which we had to create a new Map using a base-map and ground. We can set the camera on Catalonia by giving the coordinates i.e. Longitude and Latitude ([1.7056684862856024, 41.77040227096264]) as arguments in the center and apply a proper scale and adjust it to convenience. The default value of scale that we found comfortable is 1749933.8984067584.

Widgets such as compass and zoom-in and zoom-out can be added by "view.ui.add".

The input file is given in the .csv format and the Arcgis API framework take the dataset in an URL form. Due to the large size of the dataset, for demo purposes, we had included only the first one hundred rows of the dataset for now and we later will use apache spark with flask server to visualize the entire Dataset.

4.4 Creating Visualizations

We have used Vue Framework to create our Web application which happens to work well with both Apache Spark and ArcGIS API. With Vue, we created a home and about page on the home page, for now, we displaying a visualization that can interactively be queried for an attribute value that can be selected in the drop-down menu which we created using Bootstrap CSS and render the resulting visualization in the Application. We will further add many visualizations and with Flask as Backend, even analysis reports to our Web application.

This is the TypeScript code that uses the ArcGIS API which provides us with a base map and camera positions that are used as the base for our upcoming visualizations.

```
this.map = new Map({
  basemap: "hybrid",
  ground: "world-elevation",
});

this.view = new MapView({
  container: "viewDiv",
  map: this.map,
  // zoom: 10,
  scale: 1749933.8984067584,
  center: [1.7056684862856024, 41.77040227096264],
  constraints: {
    minScale: 2311162.217155,
  },
});
this.view.ui.add([
  {
    component: "compass",
```

```
        position: "top-left",
        index: 0,
      },
    ]);
```

In the beginning, the visualization we have created was a visualization of the speed attribute using the ArcGIS API framework. This is done by querying the dataset for which we are providing URL and the framework creates a Layer of visualization by consuming the dataset and understanding the type : "**spatial reference**" in the dataframe which we have already created using pandas in pyspark before providing it to the Web application and we then add that layer to the Basemap which we carefully calibrated to zoom and scale to the required location and then we display it. To dynamically query it we had to provide an attribute parameter with a query operator like shown below

```
definitionExpression: "speed >= " + this.rangeValue,
```

We can then adjust the magnitude of the attribute by using the definitionExpression attribute to generate a layer over which we render the graphs on the base map dynamically. Then we manually added a slider using the bootstrap framework with the range of the attribute so that the value can be dynamically changed by the user and render the changes over the base map. Then we wanted to be able to make any attribute with a number type to be visualized in a similar fashion.

Then we created a drop-down component where the user can select the attribute that they want to be visualized on the base map. This was done by creating an interface of all the attributes present in the dataset and their respective types and providing it dynamically. Then we also had to get dynamically, the minimum and maximum values of each value of these attributes which we parsed the entire dataset using Papa.parse() and found the maximum and minimum values which were provided to the slider and thus we were able to dynamically visualize any attribute on the map in the Web Application.

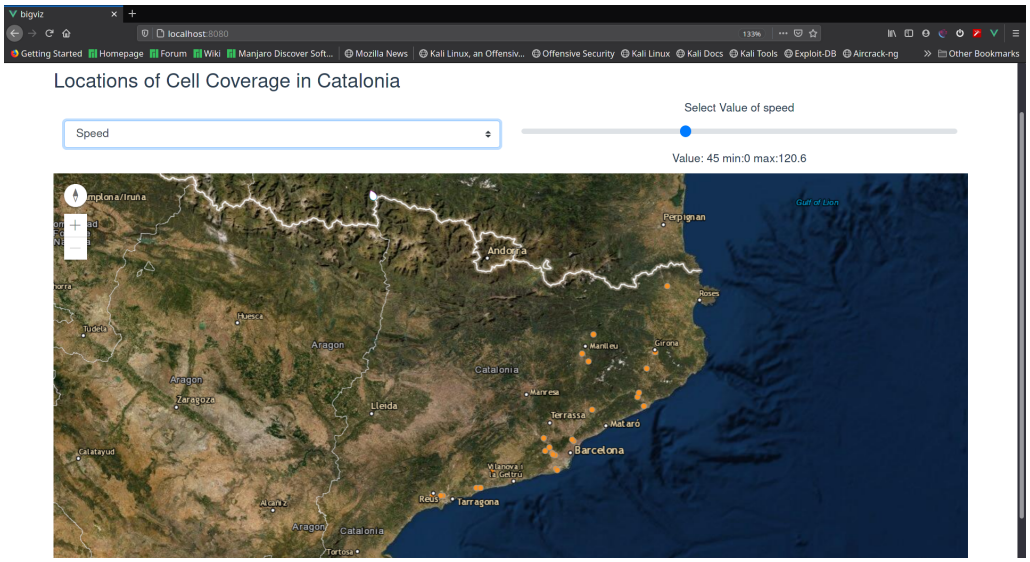

Figure 4.2: Our Web Application with our custom slider and drop down option.

4.5 Using KeplerGL

As we have mentioned previously, we have been using data is smaller quantites until now. As we have created the visualizations, we decided to include the entire dataset into the visualization.

For this we decided to use "Apache Hive", an open datawarehousing software for reading, writing and managing large data set files that are stored directly in either the Apache Hadoop Distributed File System (HDFS) or other data storage systems such as Apache HBase.

After going through the Hive documentation, we decided to integrate our Hive with the ArcGIS API. But unfortunately, the ArcGIS API allows upto only 10000 points in the visualization. And using any more points is restricted by a paywall. After trying to look for a way through we decided it was best to move on to another open-source library.

As we went through multiple libraries, we found a library called "KeplerGL", an advanced geospatial visualization tool, to render large-scale interactive maps. It is incredibly easier to use than ArcGIS API and has no payment requirements for visualizing large datasets.

4.6 Visualizations using KeplerGL

Flask, a micro web framework written in Python is required to use Keplergl. So we installed keplergl and flask using "pip". Now we needed to write the python code for visualizations.

After importing flask, keplergl, pandas and os(for reading files), we load the dataframe from our dataset.

```
    df = pd.read_csv(os.path.join(os.path.dirname(__file__),'feb03.csv'),
sep='\t',error_bad_lines=False, nrows=59999)
```

Figure 4.3: Loading the Dataframe.

In the last argument "nrows" we can actually specify the number of rows we are including. We could add more rows to the data framewww but our local systems were unable to handle the computation.

Now we call a basemap "map1" and onto that we add our data frame to be visualized.

Figure 4.4: Kepler Map Displaying 999999 rows easily without any cluster

Now the code will open the app in port:5000 by default. After we create our front end i.e. our react app, this will send a post request of our data to the front-end. This means that every time we need to load the data frame into the visualization, our back-end will do it for us.

4.7 React app with Keplergl framework

To use keplergl functionalities in react, we use an npm package called **kepler.gl**. After that we create a keplergl map by using the **mapboxApiAccessToken**.

We have a sidebar that contains the different visualizations of the data. In our Sidebar, we have multiple visualizations like Point, Polygon and Heatmap. Selecting anyone of these will automatically provide the visualization in request.

By selecting the point preset in the navigation bar, the point visualization is created by setting the id to "point" and giving the appropriate latitude and longitude.

```
map_1 = KeplerGl()
map_1.add_data(data=df, name='CellCoverage')
```

Figure 4.5: Adding data to the underlying basemap

```
<KeplerGl
        id="keplerMap"
        width={window.innerWidth}
mapboxApiAccessToken="pk.eyJ1Ijoia2luZ2F2YXRhciIsImEiOiJja25ycjdlZjgwMGQyMnZyMGl2djc2ZjBsIn0.f7ZSGKKyYf
Sr4cg9krk_Cg"
        height={window.innerHeight}
      />
```

Figure 4.6: Accessing the Kepler Map.

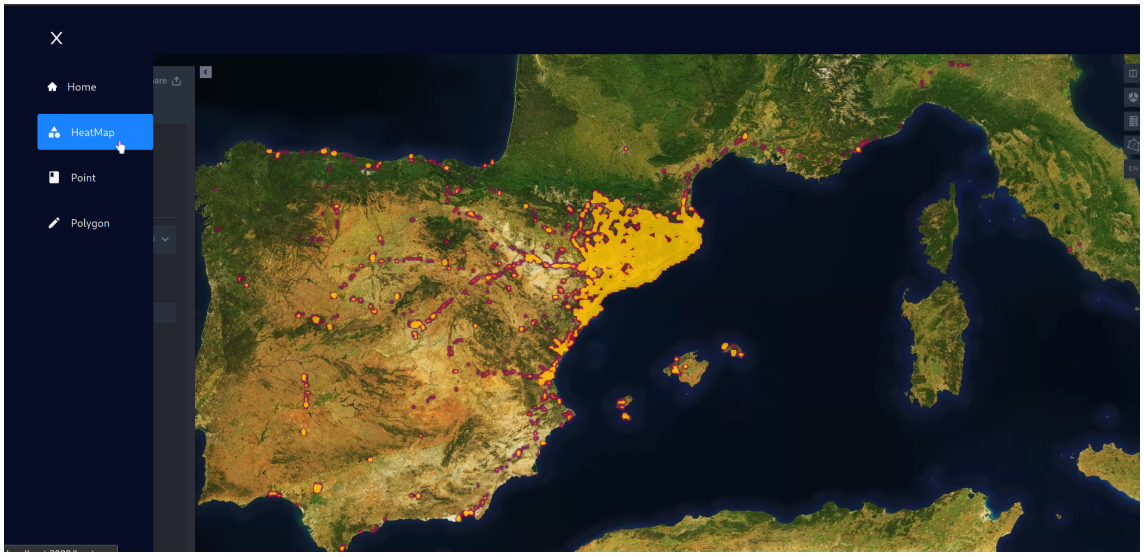

Figure 4.7: React App showing Heatmap preset

The same goes for Polygon preset as well. Instead of asking the traditional latitude and longitude attributes for location, it instead asks for GeoJson value which is the positional_geom column in our csv file.

These visualizations can be done by using a configuration called **mapConfig** provided by the Keplergl framework itself. The framework provides a configuration of the visualization it has created and changing the configuration in the react app can change the visualization of data.

Figure 4.8: React App showing point preset

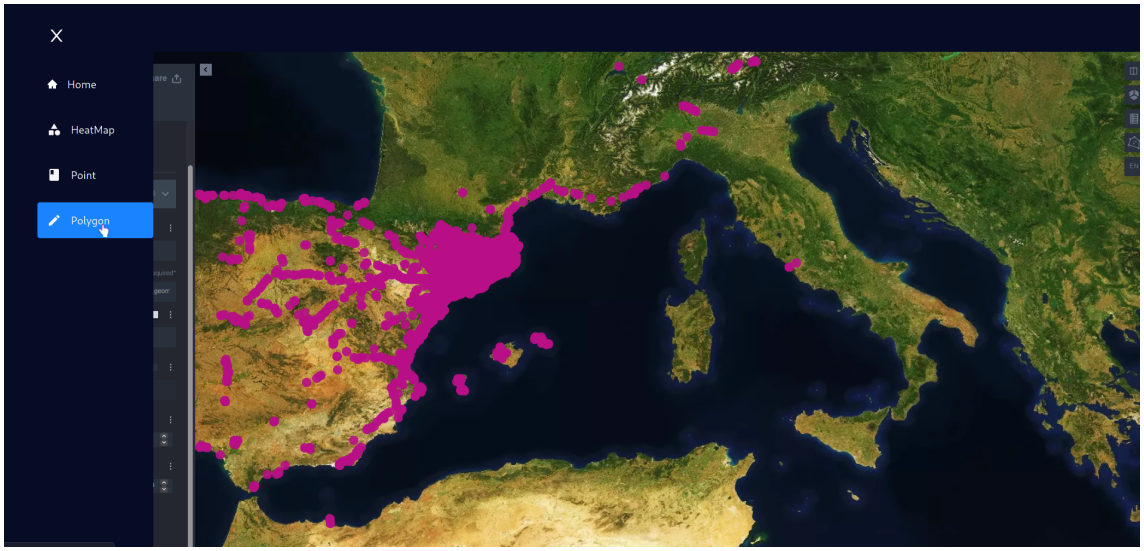

Figure 4.9: React App showing Polygon preset

4.8 Using QGIS

QGIS is a free and open-source cross-platform desktop geographic information system application that supports viewing, editing, and analysis of geospatial data.

Unlike the other two libraries, we used a desktop app for this framework.

We can upload the our dataset in all formats unlike the GIS only format in ArcGIS. To add a layer, go to Layer in the top bar and select "Add layer" and under that select "Add Delimited Text layer". We get a new window asking us to add our dataset. QGIS is intelligent enough to recognize GIS format datatypes. If you want to manually set the delimiter, you have that option also. For us, our dataset is separated by tabs. After adding our layer, we can add a basemap by using the OpenLayers Plugin. We used Google Hybrid as our basemap.

QGIS uses the GDAL/OGR library to read and write GIS data formats. QGIS gives you a couple basemaps with the OpenLayers plugin.

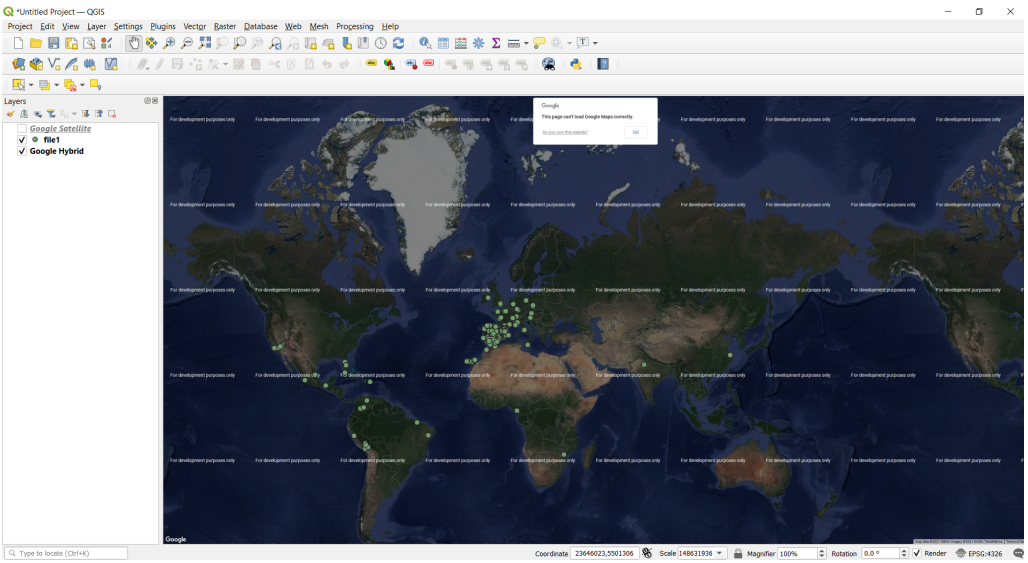

Figure 4.10: QGis visualization with a partial dataset.

4.9 Insights

Now after plotting a million points of the dataset we have found out that there are points not just in Catalonia but all over the globe. The points are heavily concentrated around Spain and Europe but there were also points in United States of America, India and many other countries.

5 Analysis

We have implemented three different frameworks for our project: ArcGIS API, kepergl and QGIS.

All three of them have their own benefits and flaws and as we worked on our project, we personally feel that the library keplergl is the best out of all three as it has great features, easier to use from a beginner's perspective which means it is easier to learn than its competitors and is smarter than the rest.

Analysis of Frameworks		
ArcGis	Keplergl	QGis
All three frameworks use "shapefile" format, a format where geometric position is calculated and stored as a separate column and type. This file format makes it easier for the frameworks to load the dataset.	All three frameworks use "shapefile" format, a format where geometric position is calculated and stored as a separate column and type. This file format makes it easier for the frameworks to load the dataset.	All three frameworks use "shapefile" format, a format where geometric position is calculated and stored as a separate column and type. This file format makes it easier for the frameworks to load the dataset.
ArcGis works on a "Layers" method. Where the basemap, values and other variables are superimposed on one another to create the final visualization.	Keplergl works on a "Layers" method. Where the basemap, values and other variables are superimposed on one another to create the final visualization.	QGis works on a "Layers" method. Where the basemap, values and other variables are superimposed on one another to create the final visualization.
ArcGis support integration with Python as well	Keplergl support integration with Python as well	QGis support integration with Python as well
Arcgis handles huge data files well and offers many features but is behind the paywall.	keplergl handles huge data files well and offers more features compared to the other two while also being free.	QGis handles huge data files well and offers fewer features compared to the other two.
ArcGis provide over 2887 GIS datasets	keplergl has five default basemaps to use.	QGIS gives you a couple basemaps with the OpenLayers plugin

6 Conclusion

During the course of this elective, we have learnt how big data is visualized and how the tools required are installed and set up. We have analyzed multiple different frameworks over the course of electives and created multiple web applications over these frameworks using spark clusters and pyspark.

References

[1] Arcgis javascript api. https://developers.arcgis.com/javascript/latest/. (Accessed on 27/02/2021).

[2] Csvlayer visualization. https://swaggypyang.github.io/arcgisapi/sdk/latest/api-reference/esri-layers-CSVLayer.html. (Accessed on 21/03/2021).

[3] Arcgis python api. https://developers.arcgis.com/python/. (Accessed on 17/02/2021).

[4] Pyspark sparkcontext. https://www.tutorialspoint.com/pyspark/pyspark_sparkcontext.htm. (Accessed on 25/01/2021).