"DATA SCIENCE SIMPLIFIED USING ARCGIS API FOR PYTHON"

LEAD CONSULTANT, INFOSYS LIMITED

SEZ Survey No. 41 (pt) 50 (pt), Singapore Township PO, Ghatkesar Mandal, Hyderabad, Telengana 500088

Abstract:

Data science, a buzz word in the market, is the science of interpreting and processing data and extracting knowledge and insights from the data in various forms. But analytics and generating insights have been part of ArcGIS since many years and its evolving year on year with more powerful analytical extensions available today. The ArcGIS API for Python is the best companion for a geo-spatial data scientist looking to reap the benefits of the vast analytical capabilities of ArcGIS. The API not only allows you to perform common tasks such as thematic mapping, geocoding, network analysis etc. but also simplifies complex vector and raster analysis. The API allows integration with a vast library of the scientific Python ecosystem. It has rich support for Pandas, an open source library providing high performance data structures and data analysis tools. The ArcGIS API for Python is well integrated with Jupyter notebook, an open source, interactive web platform that allows you to share documents containing live code, equations, visualizations and narrative text. The combination can be a powerful simulation, visualization and statistical modelling environment.

The paper explores a case study of Risk or Emergency Management in case of a major leak on a Gas network using the ArcGIS API for Python and the Jupyter notebook.

About the Author:

Mr. TEJASVI NAGARAJ

With over 14 years of experience in GIS based software development, I specialize in Geospatial technologies for Utilities and Municipalities. I have been a key player in analyzing the business and implementing technology in various projects across the globe. I have successfully implemented complex GIS Programs in the Utilities and Municipal Sector. Thought Leader in the GIS space, I help organizations in building the GIS roadmap, and adopting the latest technology trends.

E mail ID: tejasvi.nagaraj@infosys.com

Contact: +919642121922

Introduction

ArcGIS has become a preferred platform for many organizations to visualize spatial data, plan their assets and manage their operations as well as to perform real time analytics. Successful GIS analysis however requires selecting the most appropriate tools to operate on your data and ArcGIS API for Python is much more than just an analysis tool.

The ArcGIS API for Python is a powerful Python library powered by web GIS for spatial analysis, mapping and automation.
The API gives users full access to most of the sophisticated analysis tools that are available to web GIS users and also supports common GIS tasks such as map making, geocoding, routing and directions etc.
It allows you to easily combine your own data with ready to use maps and curated geographic data from Esri and other authoritative sources. It allows you to use ArcGIS GeoAnalytics tools to perform complex analysis on vector and raster data and makes working with big data sources easier.

The great time-saving automation capabilities in the API can also be used to organize and manage a web GIS with users, groups and information items, administer your Web GIS through managing credits and building an entire distributed GIS by creating GIS collaborations, where a web GIS can either be the ArcGIS Online or an ArcGIS Enterprise.

Pandas, a Python package providing fast, flexible, and expressive data structures designed to make working with relational or labeled data both easy and intuitive is commonly used to perform complex data manipulations when working with large data sets.
The ArcGIS API for Python has full support for Pandas built in. At version 1.2 the API introduced the Spatial Dataframe which allowed combining all the powerful features of a Pandas Dataframe with geometries. A Pandas Dataframe can now be instantiated directly from feature classes, cloud files and feature layers or vice-versa allowing you to perform complex data analysis and manipulations inherently suited to the Dataframe or plot the results of the data manipulations on the map as heat maps or thematic maps.
This combination of ArcGIS geometry structures with Pandas Dataframe processing now makes the Spatial Dataframe the perfect structure for integrating ArcGIS with the countless plotting, machine learning, and data science libraries of the whole Python data science ecosystem.

The ArcGIS API for Python can be best experienced on the ArcGIS Pro environment. ArcGIS Pro 2.2 comes pre-installed with the ArcGIS API for Python 1.4.1. The package also comes with Jupyter notebook, an interactive computing environment that enables users to author notebook documents that include Live code, Interactive widgets, Plots, Narrative text, Equations, Images, Video and most importantly maps powered by ArcGIS.
These documents provide a complete and self-contained record of a computation that can be converted to various formats and easily shared with others in the form of HTML pages or PDFs or as the notebook itself.

This paper further substantiates and demonstrates the various capabilities of the ArcGIS API for Python by implementing a few critical use cases for a Gas Distribution Utility related to Emergency Management and Emergency Planning using the API simulated on a Jupyter Notebook.

Mapping and Visualization using Jupyter notebook and the ArcGIS API for Python

To get started with the analysis, we can display an interactive map of the area of interest by simply specifying the name of the place and a zoom level. Layers can be added from a web GIS with a few lines of python scripts as shown below:

```
# Display the map of Chicago

In [6]: import arcgis
        from arcgis.gis import GIS
        from getpass import getpass
        password=getpass()

### Connect to your web GIS

In [11]: gis = GIS("https://tejasvi-nagaraj.maps.arcgis.com", "Tejasvi.sn", password, verify_cert=False)

### Display map and add layers

In [18]: map = gis.map('Chicago Golf Club', zoomlevel=13)  # you can specify the zoom level when creating a map
         serviceCenters=gis.content.search('title: ServiceCenter')[1]
         pipes=gis.content.search('title: P_Pipes')[0]
         valves=gis.content.search('title: P_Valve')[0]
         buildings=gis.content.search('title: Naperville_Buildings')[0]
         gasleaks=gis.content.search('title: P_GasLeak')[0]
         map.add_layer(gasleaks)
         map.add_layer(serviceCenters)
         map.add_layer(pipes)
         map.add_layer(valves)
```

Fig: 1 – Python snippet to connect to a web GIS

Fig: 2 – Map within Jupyter notebook

You could also directly open a Web Map shared with you on your web GIS as shown below:

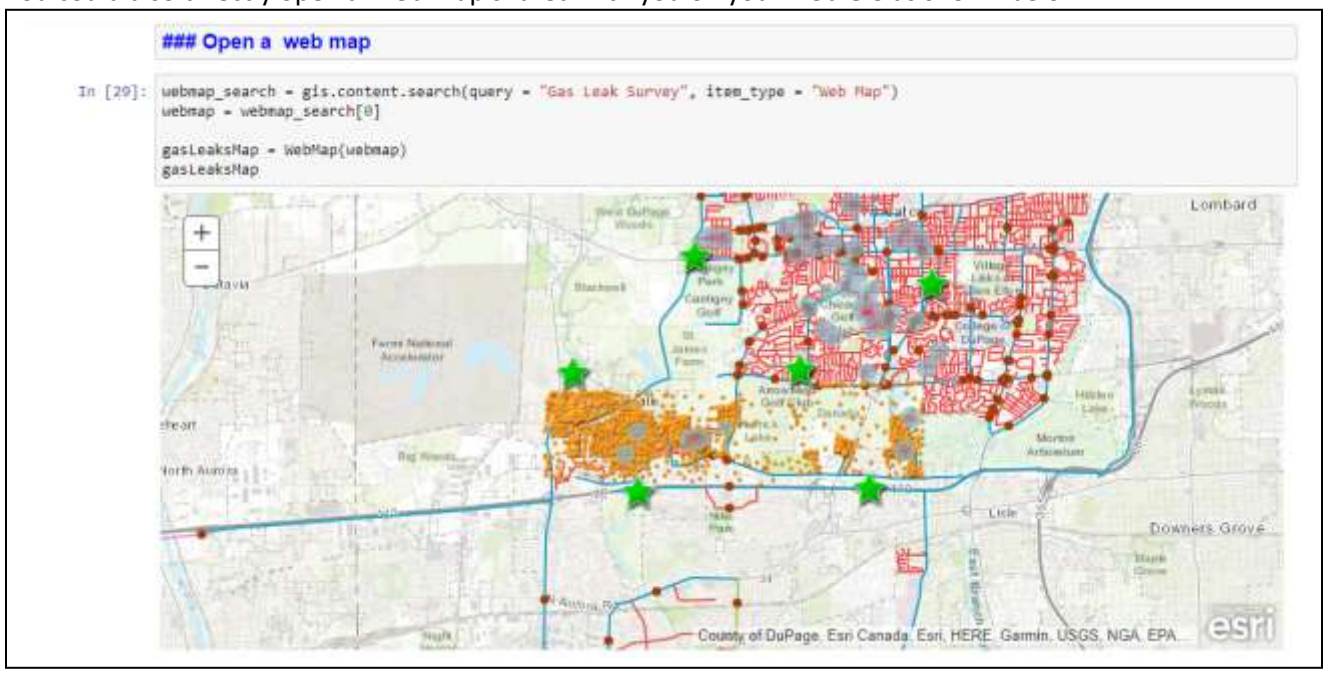

Fig: 3 – Adding an ArcGIS Online Web Map within Jupyter notebook

Emergency Response

Leaking pipes can quickly become the source of explosions, and people getting hurt or killed. A leak classified as Grade 1 is a leak that represents an existing or probable hazard to persons or property, and requires immediate repair and attention.
Information about such leaks can come from various sources such as a Customer complaint or from Technicians doing leak surveys where the they capture the location of the leaks and the corresponding gas leak readings. Let's see how a GIS analyst could use this information to provide quick analysis using the ArcGIS API for Python.

Step 1: Upload active leaks data
The Pandas library can be used to upload active leaks information received as a flat file (e.g. a CSV file), and with a simple script it can be converted to a GIS Layer and added to the map or displayed in a tabular format as shown below:

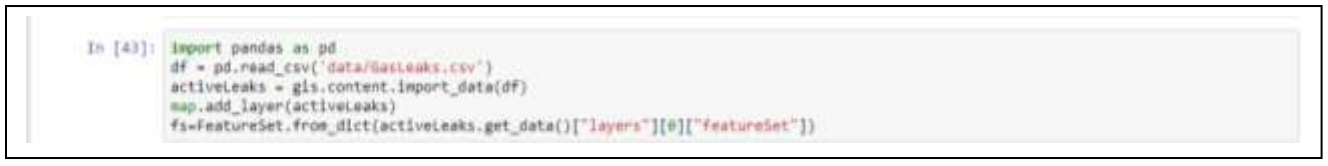

Fig: 4 – Using Pandas to read active leaks as CSV data

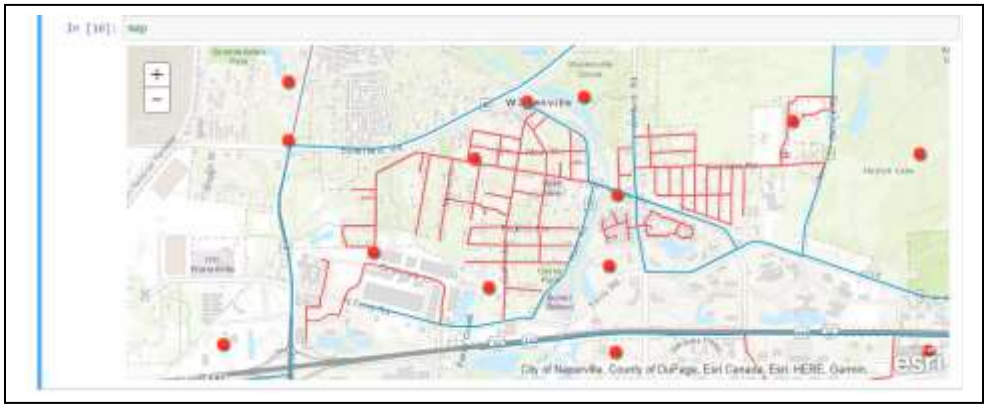

Fig: 5 – Mapping active leaks loaded from CSV within Jupyter notebook

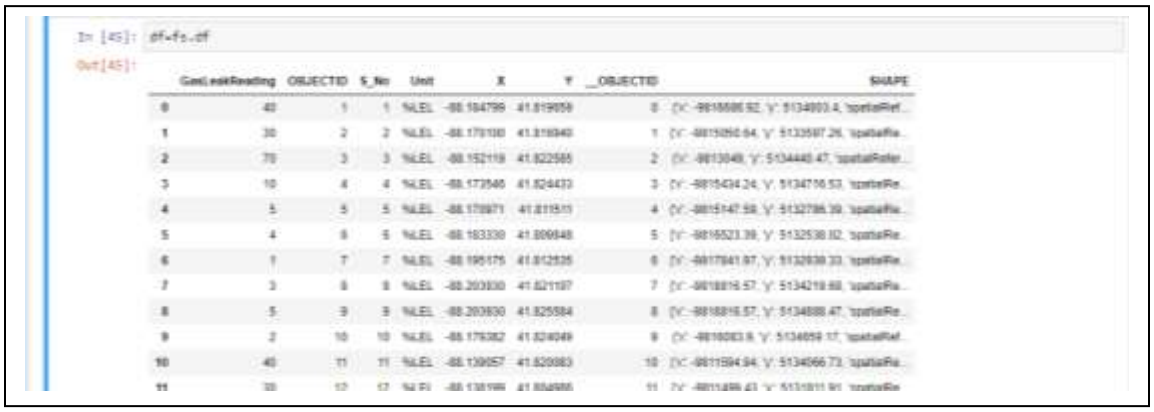

Fig: 6 – CSV data displayed in tabular format within Jupyter notebook

Step 2: Find leak addresses - We can now reverse geocode each of these locations to get the addresses that can be shared with the responder crew as shown below.

```
In [38]: for leak in fs.features:
             result = geocoding.reverse_geocode({"x": leak.attributes['X'],"y": leak.attributes['Y']})
             leak.attributes['Address'] = result['address']['LongLabel']
```

Fig: 7 – Python snippet to reverse geocode leak locations

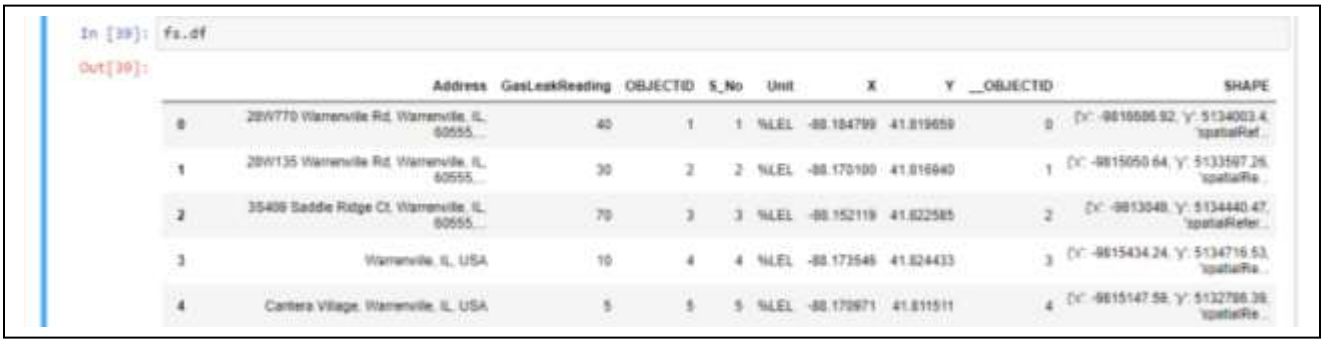

Fig: 8 – Reverse geocoded addresses for leak locations

Step 3: Derive and Visualize Gas Leak spread

Using the Interpolate Points Analysis tool provided in ArcGIS Online, we can predict Gas Leak Reading values at surrounding locations based on measurements found in the collection of points uploaded above. The interpolation results can be seen below, where darker shades are used to highlight the areas where higher Gas Leak Readings were found:

```
In [ ]: from arcgis.features.analyze_patterns import interpolate_points
        interpolated_leaks = interpolate_points(activeLeaks, field='GasLeakReading')
        map.add_layer(interpolated_leaks['result_layer'])

        gasLeaksMap.add_layer(interpolated_leaks['result_layer'], options={'title':'Interpolated Leaks'})
```

Fig: 9 – Python snippet to interpolate leaks

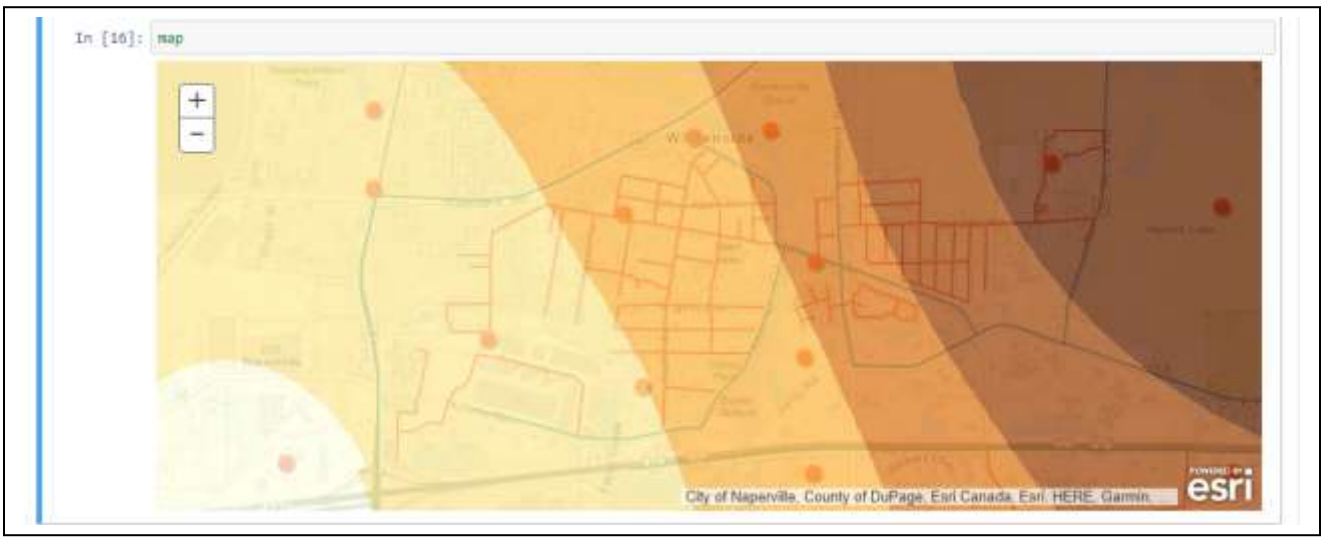

Fig: 10 – Gas leak spread interpolated using Geoanalytics

Step 4: Find buildings in the high risk zone

Using the result of the analysis performed in the previous step, we could find the buildings that intersect with the high risk zone shown in the darker shade which is classified as class higher than 5 in this example.

```
### Find interpolated polygon with classes higher than 5
fs=interpolated_leaks['result_layer'].query('Classes>5')
### Perform spatial query to find buildings at high risk
buildingsFs = buildingsFl.query(geometry=fs.features[0].geometry)
buildingsFs.df[['OWNERNME1','PSTLADDRES','PSTLCITY']]
```

Fig: 11 – Python snippet to query buildings in high risk zone

	OWNERNME1	PSTLADDRES	PSTLCITY
0	Patricia Ann Kelley	Naperville Wheaton Rd	Naperville
1	Ns Mpg Inc	2211 York Rd	Oak Brook
2	Michael Alfonso	Butterfield Rd	Naperville
3	Michael Alfonso	Butterfield Rd	Naperville
4	Elviraamesquita	Butterfield Rd	Naperville
5	Forest Preserve District		Wheaton
6	Off The Street Club	25 N Karlov	Chicago
7	Off The Street Club	25 N Karlov Ave	Chicago
8	Patricia Ann Kelley	Herrick Rd	Naperville

Fig: 12 – Query result for buildings in high risk zone

Step 5: Find the closest Service Center

(i) Using the Network Analysis tools provided by ArcGIS Online, we will now find the Service Center that can attend to a particular leak in the shortest period of time.

(ii) We will then get the optimal route that can be taken by an emergency responder from this Service Center to the Leak Location.

```
In [7]: cf_layer = network.ClosestFacilityLayer(analysis_url, gis=gis)

In [41]: result = cf_layer.solve_closest_facility(incidents=currentLeakFS,facilities=facilities,default_target_facility_count=4,return_fac
                                                  impedance_attribute_name='TravelTime',
                                                  accumulate_attribute_names=['Kilometers','TravelTime'])

In [42]: line_feat_list = []
         for line_dict in result['routes']['features']:
             f1 = Feature(line_dict['geometry'], line_dict['attributes'])
             line_feat_list.append(f1)

In [43]: routes_fset = FeatureSet(line_feat_list, geometry_type=result['routes']['geometryType'],spatial_reference= result['routes']['spat

In [44]: map.draw(routes_fset)

In [45]: map
```

Fig: 13 – Python snippet to find closest service center to leak location

Fig: 14 – Shortest route between leak location and closest service center

(iii) We will also get the analysis of the distance and travel time required to reach the leak location from each Service Center.

```
In [50]: routes_fset.df.drop(columns=['FacilityCurbApproach','SHAPE','ObjectID','IncidentCurbApproach','Shape_Length'])
```

Out[50]:

	FacilityID	FacilityRank	IncidentID	Name	Total_Kilometers	Total_Miles	Total_TravelTime
0	4	1	1	Location 1 - Red Maple Ln	4.510260	2.802546	5.943606
1	5	2	1	Location 1 - Diehl Rd	5.058817	3.143403	7.303025
2	1	3	1	Location 1 - Williams Street	7.186905	4.465735	8.201684
3	7	4	1	Location 1 - Batavia Rd	5.032629	3.127130	8.352558

Fig: 15 – Distance and Travel time from leak location to each service center

Emergency planning

Spatial analytics can be used to strategically plan the location of the Service Centers or Emergency Response centers such that the crew from any of these service centers can reach most part of the service area within an acceptable timeframe to attend any hazardous event or outages.
The ArcGIS API for Python can be used to derive Drive Time based Service Areas for these Service Centers considering various factors such as Drive Times, Travel Mode, Typical Traffic Conditions at different times of a day etc. as shown below.

Each of these analyses can be shared as an item on ArcGIS online, or can be added as a layer in a Web Map directly from the ArcGIS API for Python which is particularly useful when the analysis is automated and the results are then shared to end users who can view them at their convenience on ArcGIS Online or Portal.

```
### Service Area Analysis using Truck Mode

service_area_url = gis.properties.helperServices.serviceArea.url
sa_layer = network.ServiceAreaLayer(service_area_url, gis=gis)
travel_modes = sa_layer.retrieve_travel_modes()
truck_mode = [t for t in travel_modes['supportedTravelModes'] if t['name'] == 'Trucking Time'][0]
result = sa_layer.solve_service_area(serviceCenters, default_breaks=[5,10,15],
                                     travel_direction='esriNATravelDirectionFromFacility',
                                     travel_mode=truck_mode)

### Draw result on map

poly_feat_list = []
for polygon_dict in result['saPolygons']['features']:
    f1 = Feature(polygon_dict['geometry'], polygon_dict['attributes'])
    poly_feat_list.append(f1)
service_area_fset = FeatureSet(poly_feat_list,
                       geometry_type=result['saPolygons']['geometryType'],
                       spatial_reference= result['saPolygons']['spatialReference'])
colors = {5: [0, 128, 0, 90], 10: [255, 255, 0, 90], 15: [255, 0, 0, 90]}
fill_symbol = {"type": "esriSFS","style": "esriSFSSolid","color": [115,76,0,255]}

for service_area in service_area_fset.features:
    #set color based on drive time
    fill_symbol['color'] = colors[service_area.attributes['ToBreak']]
    #set popup
    popup={"title": "Service area",
           "content": "{} minutes".format(service_area.attributes['ToBreak'])}
    map.draw(service_area.geometry, symbol=fill_symbol, popup=popup)
```

Fig: 16 – Python snippet to find and display Drive Time based Service Areas for service centers

The analysis results for 5 min, 10 min and 15 min drive time based service areas for all the service centers combined are plotted on the map in different shades as shown below.

Fig: 17 – Drive time based Service Areas plotted on map within Jupyter notebook

Further planning of the number of service centers or number of crew to be staffed in each service center can be done by looking at the following factors:

(i) Load on each service center in terms of number of leaks or outages or complaints they are servicing. The Spatial Dataframe of the ArcGIS API for Python can easily be plotted using matplotlib as shown below:

```
In [24]: %matplotlib inline
         import matplotlib.pyplot as pd

         summarydf = fs.df[['SERVICECEN']]
         counts = summarydf.groupby(['SERVICECEN']).size()

         counts.plot(kind='pie', subplots=True)

Out[24]: array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f31ecb76898>],
               dtype=object)
```

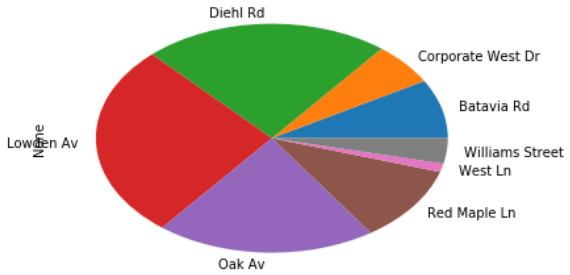

Fig: 18 – Pie chart showing percentages of leaks serviced by each service center within Jupyter notebook

(ii) Average time taken by each service center to respond to critical incidents like leaks

```
%matplotlib inline
import matplotlib.pyplot as pd

summarydf = fs.df[['SERVICECEN','RESPONSETI']]
summarydf = summarydf.groupby(['SERVICECEN'])['RESPONSETI'].mean()
summarydf.plot(kind='bar')

<matplotlib.axes._subplots.AxesSubplot at 0x7fe700650588>
```

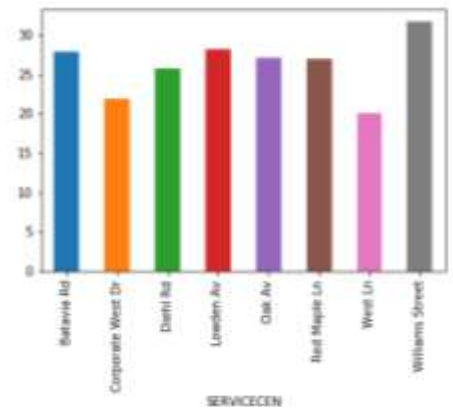

Fig: 19 – Bar chart showing average response time for service center within Jupyter notebook

Conclusion

Python is widely accepted as a highly suitable programming language for data exploration and analysis with its rich ecosystem of libraries such as NumPy, SciPy, pandas, maptplolib, scikit-learn, etc. and interactive visualization environments such as Jupyter notebooks. The ArcGIS Python API follows suite in being your library for comprehensive analyses of geospatial data. With an intuitive design and easy to use syntax, the API opens up access to rich geoprocessing services and big data analysis capabilities of spatial data.

The samples shown here are just a glimpse of what the ArcGIS Web platform and the ArcGIS API for Python are capable of. The ArcGIS API for python can crunch vector, raster and non-spatial data from a variety of sources to bring out mission critical analysis and gives the power to organizations to automate, publish and share results with a wider audience.

The recent and upcoming versions of the ArcGIS API for Python are very promising in terms of exploratory data analysis and machine learning with geospatial data and a go to platform for geospatial data scientists.

References

1. https://en.wikipedia.org/wiki/Pandas_(software)
2. https://en.wikipedia.org/wiki/Project_Jupyter#Jupyter_Notebook
3. http://notebooks.esri.com/user/PujGGtRQ8OfDsUvPZngCHeSpi/tree/samples/04_gis_analysts_data_scientists
4. https://developers.arcgis.com/python/