EGM722: Assignment Part 1: The How-to Guide

Lighting Column Upgrades and Drainage Asset Cleaning on the M20 Motorway, Ireland

Name: Roy Coates

Student ID: B00996803

GitHub Repository Link: RoyCoates/EGM722Project: Repository for EGM722 Assignment Project

1.0 Introduction:

TII (Transport Infrastructure Ireland) is the authority responsible for the maintenance of all the motorways in the Republic of Ireland. The motorway network contains of sections totalling 916km which are sub-divided into three sections and managed on behalf of TII by private contractors.

The second region (Network B) is managed by Colas Joint Venture Ltd. It includes the Midlands/West of the country with approximately 256 km of carriageway and includes parts of the N/M4, N/M6, M7, N/M18, N19, M20 and N85 (TII Website – Road Maintenance)

As well as regular maintenance works such as grass-cutting, drain cleaning, incident response, gritting etc, TII task Colas Joint Venture (CJV) with additional works such as road pavement repairs, signage scheme replacements and lighting column upgrades (Replacing older Halogen Lighting Columns with more efficient LED ones). This project will focus on a Lighting Column Upgrade Project for the 5 Junctions of the M20 motorway.

Disclaimer: **This is not an actual project designated by TII. It is merely a pretend scenario for this college project.**

In this scenario, CJV have been tasked with an LED upgrade scheme on the M20. Information on which Lighting Columns require upgrading has been received in Shapefile format. CJV are also planning to utilise the Traffic Management that will be required to complete these upgrades to carry out cleaning and maintenance works on the near-by drainage assets. CJV has existing shapefiles for Gullies and Filter Drains for the M20.

The code has been designed to take the Lighting Point and Drainage asset shapefiles so that it can:

- Create an interactive map that:
    - Displays the lighting points and drainage assets on a map
    - Allows for the basemap to be changed from Open Street Maps to the ESRI World Imagery
    - Colour code the lighting columns to highlight the ones scheduled for upgrade
    - Creates Pop-Ups for each asset showing the information stored in their respective shapefiles.
    - Display a legend of all assets for the user.
    - Allows for Assets to be switched on and off

- Create a chart showing the total number of lighting columns to be changed at each Junction on the M20

- Perform a cost savings analysis and display the results in a table

- Allow the user to extract the information from the Lighting Column Shapefile as an excel file.

The code is provided as a JupyterLab file (EGM722Project.ipynb) and as a programme file (EGM722Project.py)

2.0 Setup and Installation:

To run the Python code and get the items listed above, there are a number of steps that need to be followed. Please complete these in the order as set out below.

2.1 Install git and Anaconda

To run the python code, you will need to install both **git** and **Anaconda** on your computer. The link and instructions for installing "git" can be found at this website: **https://git-scm.com/downloads**

The link and instructions for installing Anaconda can be found at this website: **https://docs.anaconda.com/anaconda/install/**

2.2 Download/clone this repository

Once you have the two items above installed, you can clone this repository to your computer by completing the following:

2.2.1 Open GitHub Desktop and select File > Clone Repository.

2.2.2 Select the URL tab, then enter the URL for this repository which is: https://github,com/RoyCoates/EGMProject

2.2.3 Open Git Bash from the Start menu, then navigate to your folder.

2.2.4 Execute the following command:
`git clonehttps://github.com/RoyCoates/EGM722Project.git`.
You should see messages relating to downloading/unpacking files, and the repository should be set up.

2.3 Create a conda environment

Once you have successfully cloned the repository, you now must create a conda environment.

Select the "environment.yml" file provided in the repository. Using Anaconda Navigator, select **"Import"** from the bottom of the "Environments" panel.

Alternatively, you can open a command prompt (on Windows, you may need to select an Anaconda command prompt). Navigate to the folder where you cloned this repository and run the following command:

C:\Users\RoyCoates> conda env create -f environment.yml

This step will only have to be performed once.

2.4 Start Jupyter Lab

From the Anaconda Navigator, you can launch Jupyter Lab. Make sure that your "EGM722Project" environment is activated (See Trouble Shooting section of this document for further information)

2.4.1 From the command-line, first open a terminal window or an Anaconda Prompt, and navigate to the folder where you have cloned the repository.

2.4.2 Activate your newly created environment "conda activate EGM722Project"

2.4.3 Run Jupyter Lab ,which should launch a web browser window, which should give you an overview of the current folder.

2.4.4 Navigate and select the file `EGM722Project.ipynb` under "https://github.com/RoyCoates/EGM722Project.git"

Note: The files **"EGM722Project.ipynb"** & "**EGM722Project.py"** both run the same code. The **".ipynb"** file can be used with **Jupyter Lab**. The **".py file"** can be used with **PyCharm** or many other IDE's (Integrated Development Environments)

2.5 PyCharm - Download & Set-Up

To view and execute the programme file `EGM722Project.py` that is in the GIT repository. I suggest you download and install PyCharm. PyCharm is an Integrated Development Environment (IDE)

2.5.1 You can download PyCharm from:
https://www.jetbrains.com/pycharm/download/other.html
Select the version for your current operating system (Windows, MacOS, or Linux). When installed, open and select "Create New Project"

2.5.2 Set **"Location**" to the folder where you cloned the EGM722Project repository

2.5.3 Set-up a python interpreter. Use the conda environment that you have set up. (See section 2.3 - Create a conda environment)

2.5.4 In the New Project pane, go to **"Environment"** and select **"Select Existing"**

2.5.5 Under **"Type"** select **"Conda"**.

2.5.6 The path to the **"conda"** program is **" ~/Anaconda3/bin/conda"** or **"~/Anaconda3/condabin/conda.bat"**

2.5.7 When the path is set, click **"Create"** and select **"Create from Existing Sources"**

2.6. Run the code

The project code (.py file) will load an interactive map of the M20 motorway in Ireland. Lighting columns scheduled for LED upgrading as well as Drainage Assets that require inspection will be available as layers on the map.

All the shapefile required to run the code are stored in the file" **data_files"** in the GIT repository. The list of shapefiles I as follows:

Use **Jupyter Lab** to run the **"EGM722Project.ipynb"** file or **PyCharm** to run the **"EGM722Project.py`file"**

2.7 Core Dependencies

Core dependencies are essential third-party libraries or modules required for a project to function. These are external packages that must be installed as they don't come with Python's standard library. This will be discussed further in **3.0 Methods** section. The following is a list of Core Dependencies that are used to run the Python Code:

- **geopandas** - for spatial data handling
- **pandas** - for data manipulation
- **matplotlib** - for charts/plots
- **folium** - for interactive web maps

3.0 Methods

The code was developed using JupyterLab. This allowed for the different segments of the code to be tested before copying it to the python programme file. This section of the How-to Guide will go through and explain the code sequentially. The figures are screenshots from the JupyterLab notebook used to develop the code.

3.1 Importing Packages/Dependencies

"Many developers have written their own modules, extending Python's capabilities beyond what is provided by the standard library of modules packaged with Python" (Sweigart, A. 2020). Using the import statement, this project makes use of several of these packages or dependencies (See Figure 1)

Import the required packages

```
# Imports the required packages
import os
import pandas as pd                # Used for data handling
import geopandas as gpd            # Core for spatial data operations
import matplotlib.pyplot as plt    # For creating the bar chart
import folium                      # Main mapping library
from folium.plugins import MeasureControl  # For measurement tool
from folium import FeatureGroup, TileLayer  # For layer control and base maps
```

Figure 1: List of required packages/dependencies required to run the code.

3.2 Load Geospatial data and Convert Geospatial Data Co-ordinates

The next section of code uses functions to load the shapefiles that are required to be displayed on the map.
"A function is like a mini program within a program" (Sweigart, A. 2020). "*Python Functions is a block of statements that return the specific task"* (https://www.geeksforgeeks.org/python-functions/).

The first function, load_geospatial_data, reads the shapefiles from the file "data_file" and returns them as GeoDataFrames in a tuple (See Figure 2)
The second function, convert_to_wgs84, takes the GeoDataFrames and converts their coordinate reference system (CRS) to WGS84. This is so the data will load in the correct position on the basemaps "ESRI World Imagery" & "Open Street Maps" (OSM) (See Figure 2)

The code section "*if __name__ == "__main__":*" uses these functions to load the data, convert the data and unpack the data so it is ready to use. (See Figure 2)

```python
#Load the required Shapefiles stored in the data_file file
def load_geospatial_data() -> tuple:
    """Loads all geospatial datasets from specified file paths.

    Returns:
        tuple: Contains 6 GeoDataFrames in this order:
            (MarkerPosts, Filter_Drains, Gully, Junctions,
             Lighting_Column, Boundary)
    """
    # These are the shapefiles stored in the "data_files" file that we want to display on the map
    MarkerPosts = gpd.read_file('data_files/MarkerPost_100M.shp')
    Filter_Drains = gpd.read_file('data_files/FD_Filter_Drain.shp')
    Gully = gpd.read_file('data_files/GY_Gully.shp')
    Junctions = gpd.read_file('data_files/Junctions.shp')
    Lighting_Column = gpd.read_file('data_files/LP_Lighting_Point.shp')
    Boundary = gpd.read_file('data_files/MMaRC_B_Boundary.shp')

    return MarkerPosts, Filter_Drains, Gully, Junctions, Lighting_Column, Boundary
# The shapefiles are in ITM. This will convert them to WGS84 so they can be used with open street map
def convert_to_wgs84(*gdfs: gpd.GeoDataFrame) -> list:
    """Converts GeoDataFrames to WGS84 (EPSG:4326) coordinate system.

    Args:
        *gdfs: Variable number of GeoDataFrames to convert

    Returns:
        list: Converted GeoDataFrames in same order as input
    """
    return [gdf.to_crs("EPSG:4326") for gdf in gdfs]

# To run the code
if __name__ == "__main__":
    # Load and prepare data
    raw_data = load_geospatial_data()
    wgs84_data = convert_to_wgs84(*raw_data)
    MarkerPosts, Filter_Drains, Gully, Junctions, Lighting_Column, Boundary = wgs84_data
```

Function 1 – load_geostatial_data

Function 2 – convert_to_wgs84

Figure 2: Loading required Shapefiles and converting them into WGS84

3.3 Map Creation and Adding Data to the Map

Next, we create a basemap and display our geospatial data(shapefiles). Additional elements such as symbology, labelling, map legend and map tools will be discussed

3.3.1 Folium

The code uses a folium. *"Folium makes it easy to visualize data that's been manipulated in Python on an interactive leaflet map."* (https://python-visualization.github.io/folium/latest/). The basemap is set to ESRI World Imagery but the option of switching the basemap to OSM is also incorporated into the code (See Figure 3)

The map is designed to open on the M20 Junction 5 (The first junction to have the upgrade works)

```python
# Create base map that zooms into Junction 5 - The first junction that works are scheduled for.
m = folium.Map(
    location=[52.5856, -8.7211],
    zoom_start=18,
    tiles="Esri.WorldImagery"
)

# Add OpenStreetMap as an additional base layer. The basemaps can be changed
TileLayer('OpenStreetMap', name='OpenStreetMap').add_to(m)
```

M20, Junction 5 Co-ordinates

Ability to switch basemap to OSM

Figure 3: Loading Folium Map with ESRI World Imagery set as default basemap

3.3.2 Adding Layers with different Symbology

Each asset type requires its own symbology to stand out in the map. Figure 4 shows an example of two layer with different symbology. The code has utilised the "Scheduled F" column in Lighting Column Shapefile to allow different symbology (colouring) between lighting columns that will be upgraded (green) and lighting columns remaining as is (Orange)
The Marker Post are symbolised a red circles.

```
# Add the Lighting Columns - The Lighting columns to be upgraded will be coloured green. The columns staying the same will be coloured orange.
Lighting_Column.explore(
    m=m,
    column='ScheduledF',
    categorical=True,
    categories=['Yes', 'No'],
    cmap=['green', 'orange'],
    marker_kwds={'radius': 7},
    name='Lighting Columns'
)

# Add the Marker Posts
MarkerPosts.explore(
    m=m,
    color='red',
    marker_kwds={'radius': 5},
    name='Marker Posts'
)
```

Lighting Column Symbology

Marker Post Symbology

Figure 4: Example of Layer Symbology's.

3.3.3 Map Legend

Folium does not have a legend function. *"The importance of legends cannot be overstated, and their absence in Folium has led to a variety of workarounds"* (https://www.geeksforgeeks.org/create-a-legend-on-a-folium-map-a-comprehensive-guide/)

A custom legend was created using Cascading Style Sheets (CSS). Figure 5 and 5A display the code used:

```
# Create a Legend for the map using CSS classes
legend_html = """
<div class="legend-container">
    <h4 class="legend-title">Legend</h4>

    <div class="legend-item">
        <i class="fa fa-map-pin legend-icon junction-icon"></i>
        <span>Junctions</span>
    </div>

    <div class="legend-item">
        <div class="legend-line boundary-line"></div>
        <span>Boundary</span>
    </div>

    <div class="legend-item">
        <div class="legend-icon gully-icon"></div>
        <span>Gully</span>
    </div>

    <div class="legend-item">
        <div class="legend-line drain-line"></div>
        <span>Filter Drains</span>
    </div>

    <div class="legend-item">
        <div class="legend-icon lamp-upgraded"></div>
        <span>Lighting Columns To Be Upgraded</span>
    </div>

    <div class="legend-item">
        <div class="legend-icon lamp-existing"></div>
        <span>Lighting Columns to Remain As Is</span>
    </div>

    <div class="legend-item">
        <div class="legend-icon marker-icon"></div>
        <span>Marker Posts - M20</span>
    </div>
```

```
    <style>
        .legend-container {
            position: fixed;
            bottom: 50px;
            left: 50px;
            z-index: 1000;
            background: white;
            border: 2px solid grey;
            border-radius: 5px;
            padding: 10px;
            font-size: 12px;
            max-width: 250px;
        }

        .legend-title {
            margin: 0 0 5px 0;
            font-size: 14px;
        }

        .legend-item {
            display: flex;
            align-items: center;
            margin-bottom: 5px;
        }

        .legend-icon {
            width: 20px;
            height: 20px;
            margin-right: 5px;
            border: 1px solid black;
            border-radius: 50%;
        }

        .junction-icon {
            color: lightred;
            font-size: 18px;
            border: none;
        }

        .legend-line {
            width: 20px;
            height: 0;
            margin-right: 5px;
        }

        .boundary-line { border: 2px solid red; }
        .drain-line { border-bottom: 2px solid blue; }
        .gully-icon { background: yellow; }
        .lamp-upgraded { background: green; }
        .lamp-existing { background: orange; }
        .marker-icon { background: red; }
    </style>
</div>
"""
m.get_root().html.add_child(folium.Element(legend_html))
```

Add Legend to the Map

Figures 5 & 5A – Creating a Legend using CSS

3.3.4 Custom Labels

Custom labels were created for each layer. For each shapefile, a column was selected to use as the label. Figure 6 shows the code used for the Marker Post Layer. Marker Post labels are sourced from the "mVal" column, were set to font size 12, coloured white and positioned above the marker. The other layers were labelled using similar coding.

```
# Add the Marker Post Labels
for idx, row in MarkerPosts.iterrows():
    if not row.geometry.is_empty and pd.notnull(row['mVal']):
        label = folium.DivIcon(
            html=f"""<div style="
                color: white;
                font-size: 12px;
                font-weight: bold;
                transform: translate(-50%, -100%);
                z-index: 9999;">{row['mVal']}</div>"""
        )
        folium.Marker(
            location=[row.geometry.y, row.geometry.x],
            icon=label,
            icon_size=(0, 0)
        ).add_to(m)
```

Figure 6 – Label creation and parameters

3.3.5 Map Tools

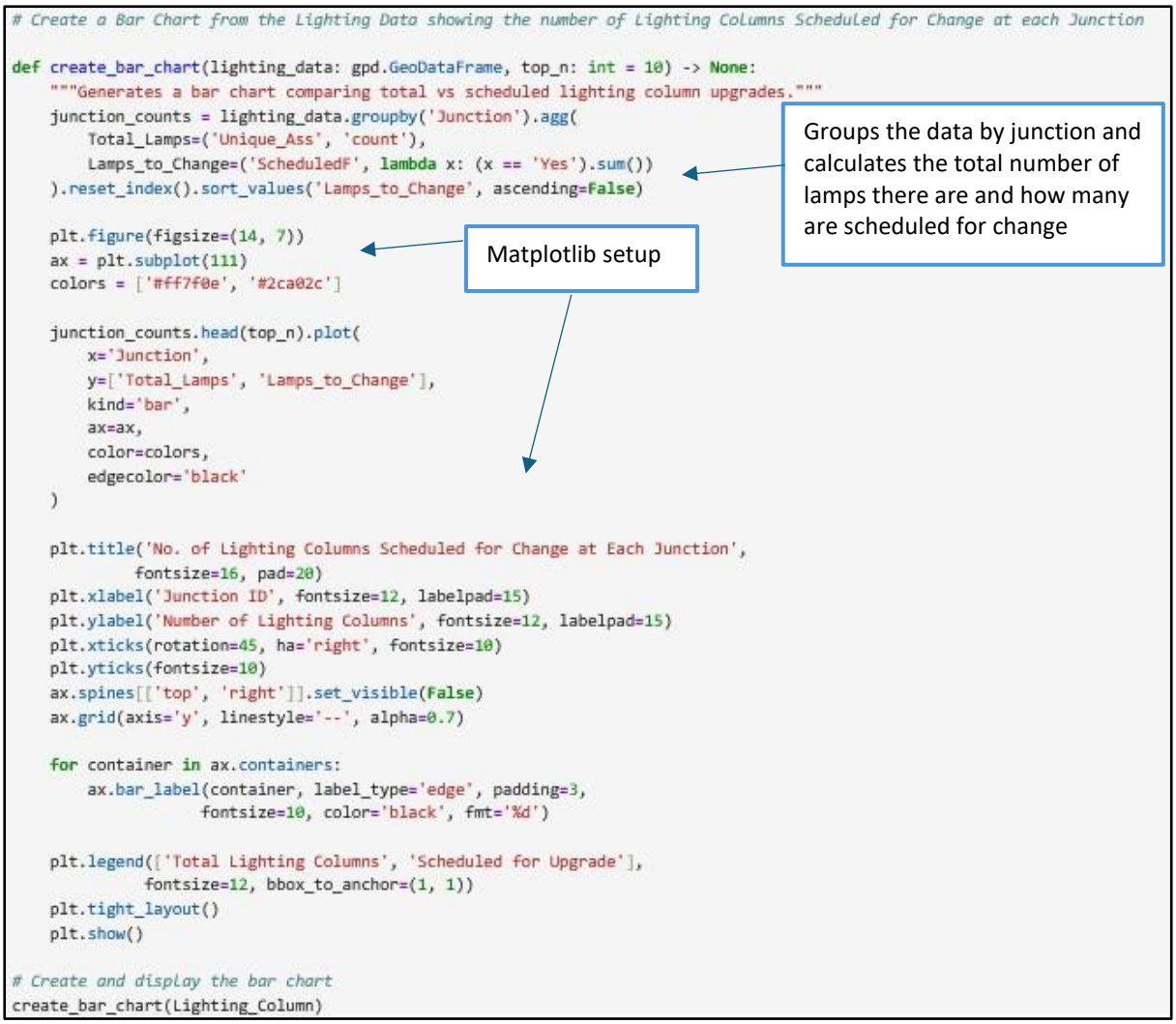
A distance measuring tool was added to the map to allow the user measure distances between different assets. This would allow the operatives to plan the Traffic Management required to access a certain number of lighting columns and drainage assets at one time (See Figure 7)

```
# Add a distance tool to the map. This will allow the user to measure distances between the different assets.
MeasureControl(position="bottomleft", primary_length_unit="meters").add_to(m)
folium.LayerControl().add_to(m)
```

Figure 7 – Code to implement a distance measuring tool

3.4 Bar Chart Creation – No. of Lighting Columns to be Upgraded

The code is designed to display a bar chart showing the total number of Lighting Columns per Junction and the number of Lighting Columns to be upgraded at each Junction.
The code uses matplotlib to generate the bar chart with parameters of the chart having to be set out (Title, Colours, Labels, Legend etc)
Code examples from https://pythonbasics.org/matplotlib-bar-chart/ were modified and added to, to create the bar chart. (See Figure 7)




```python
# Create a Bar Chart from the Lighting Data showing the number of Lighting Columns Scheduled for Change at each Junction

def create_bar_chart(lighting_data: gpd.GeoDataFrame, top_n: int = 10) -> None:
    """Generates a bar chart comparing total vs scheduled lighting column upgrades."""
    junction_counts = lighting_data.groupby('Junction').agg(
        Total_Lamps=('Unique_Ass', 'count'),
        Lamps_to_Change=('ScheduledF', lambda x: (x == 'Yes').sum())
    ).reset_index().sort_values('Lamps_to_Change', ascending=False)

    plt.figure(figsize=(14, 7))
    ax = plt.subplot(111)
    colors = ['#ff7f0e', '#2ca02c']

    junction_counts.head(top_n).plot(
        x='Junction',
        y=['Total_Lamps', 'Lamps_to_Change'],
        kind='bar',
        ax=ax,
        color=colors,
        edgecolor='black'
    )

    plt.title('No. of Lighting Columns Scheduled for Change at Each Junction',
              fontsize=16, pad=20)
    plt.xlabel('Junction ID', fontsize=12, labelpad=15)
    plt.ylabel('Number of Lighting Columns', fontsize=12, labelpad=15)
    plt.xticks(rotation=45, ha='right', fontsize=10)
    plt.yticks(fontsize=10)
    ax.spines[['top', 'right']].set_visible(False)
    ax.grid(axis='y', linestyle='--', alpha=0.7)

    for container in ax.containers:
        ax.bar_label(container, label_type='edge', padding=3,
                     fontsize=10, color='black', fmt='%d')

    plt.legend(['Total Lighting Columns', 'Scheduled for Upgrade'],
               fontsize=12, bbox_to_anchor=(1, 1))
    plt.tight_layout()
    plt.show()

# Create and display the bar chart
create_bar_chart(Lighting_Column)
```

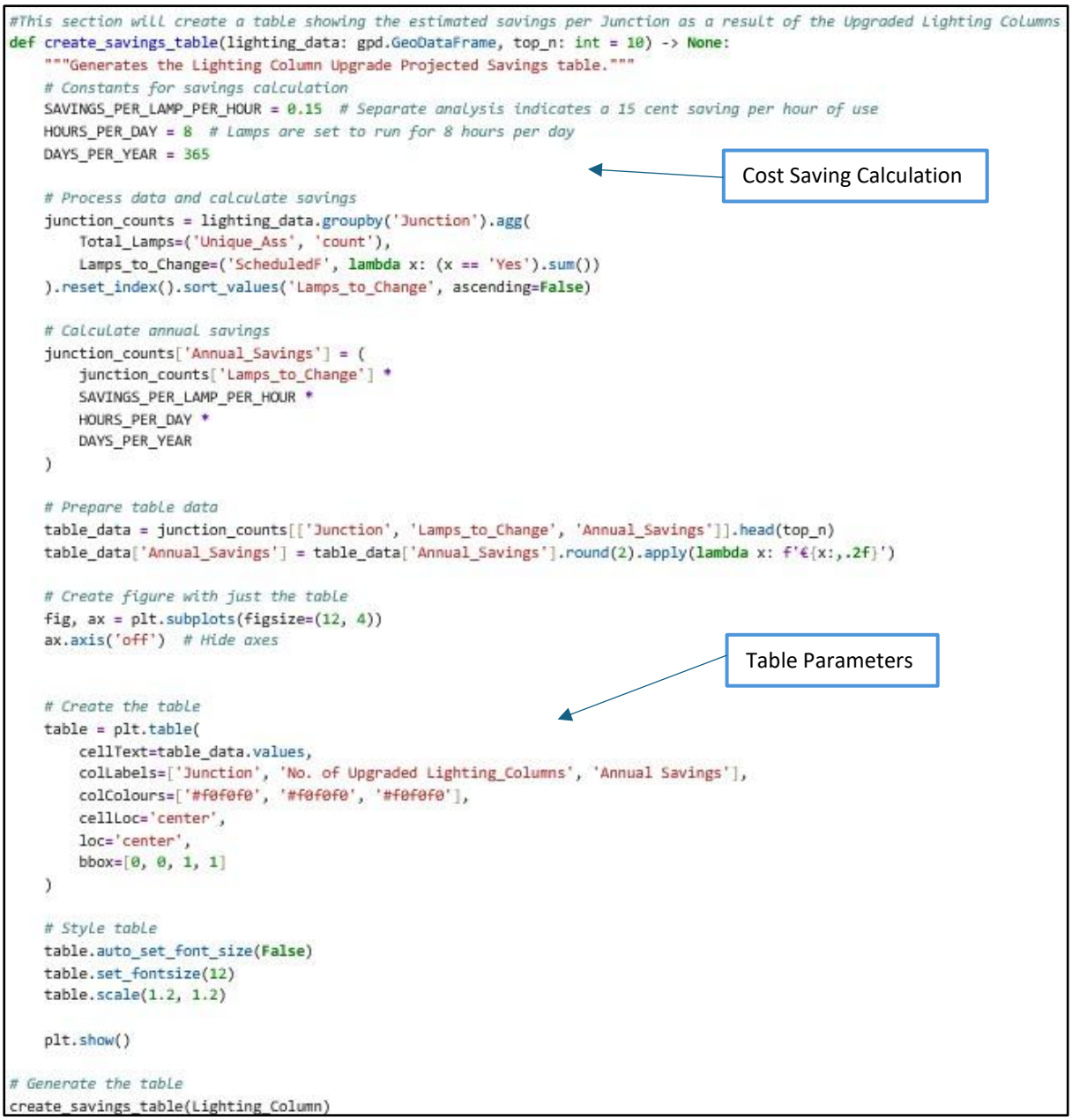
Figure 7 – Code for Bar Chart

3.5 Cost Savings Analysis

The upgraded lighting columns use an LED bulb which uses less electricity and is more cost effective. The cost saving per one hour use of the led bulb versus the standard SON bulb is 35 cents.
Every lighting column is programmed to run for 8 hours a day, 365 days a year. Using this information, it is easy to calculate the savings of one lamp (€0.15 * 8 * 365) = €438

This calculation is set out in the coding and the total number of lamps to be upgraded is used to calculate the annual saving per Junction.

A table is generated with the column headers “Junction”, “No. of Upgraded Lighting Columns” & “Annual Saving”.




```python
#This section will create a table showing the estimated savings per Junction as a result of the Upgraded Lighting Columns
def create_savings_table(lighting_data: gpd.GeoDataFrame, top_n: int = 10) -> None:
    """Generates the Lighting Column Upgrade Projected Savings table."""
    # Constants for savings calculation
    SAVINGS_PER_LAMP_PER_HOUR = 0.15  # Separate analysis indicates a 15 cent saving per hour of use
    HOURS_PER_DAY = 8  # Lamps are set to run for 8 hours per day
    DAYS_PER_YEAR = 365

    # Process data and calculate savings
    junction_counts = lighting_data.groupby('Junction').agg(
        Total_Lamps=('Unique_Ass', 'count'),
        Lamps_to_Change=('ScheduledF', lambda x: (x == 'Yes').sum())
    ).reset_index().sort_values('Lamps_to_Change', ascending=False)

    # Calculate annual savings
    junction_counts['Annual_Savings'] = (
        junction_counts['Lamps_to_Change'] *
        SAVINGS_PER_LAMP_PER_HOUR *
        HOURS_PER_DAY *
        DAYS_PER_YEAR
    )

    # Prepare table data
    table_data = junction_counts[['Junction', 'Lamps_to_Change', 'Annual_Savings']].head(top_n)
    table_data['Annual_Savings'] = table_data['Annual_Savings'].round(2).apply(lambda x: f'€{x:,.2f}')

    # Create figure with just the table
    fig, ax = plt.subplots(figsize=(12, 4))
    ax.axis('off')  # Hide axes


    # Create the table
    table = plt.table(
        cellText=table_data.values,
        colLabels=['Junction', 'No. of Upgraded Lighting_Columns', 'Annual Savings'],
        colColours=['#f0f0f0', '#f0f0f0', '#f0f0f0'],
        cellLoc='center',
        loc='center',
        bbox=[0, 0, 1, 1]
    )

    # Style table
    table.auto_set_font_size(False)
    table.set_fontsize(12)
    table.scale(1.2, 1.2)

    plt.show()

# Generate the table
create_savings_table(Lighting_Column)
```

Figure 8 - Code for Cost Savings Analysis

4.0 Expected Results

When the EGM722Project.py`file is run in PyCharm. The following will be generated:

1. A browser window containing the interactive map will open. The positions of the Lighting Column and Drainage Assets will be clearly displayed. The window will zoom in to Junction 5 on the M20 as this is the first junction scheduled for the works. Figure 1 below shows the map and contains notes on the different map features.

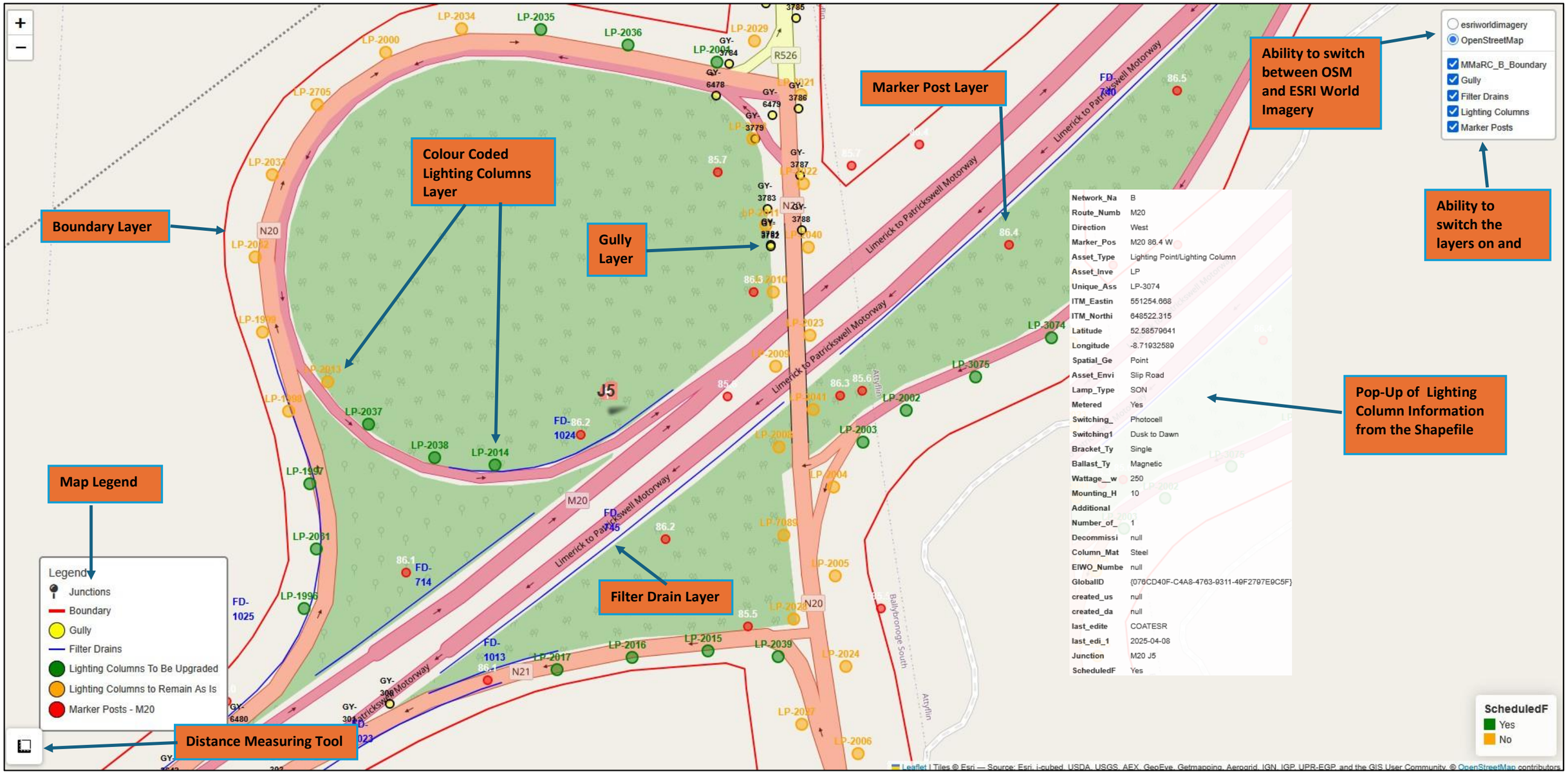


Figure 9 : Lighting Column & Drainage Data on Interactive Map

2. A bar chart graphic showing the number of Lighting Columns Schedules for Change at each Junction (See Figure 2)

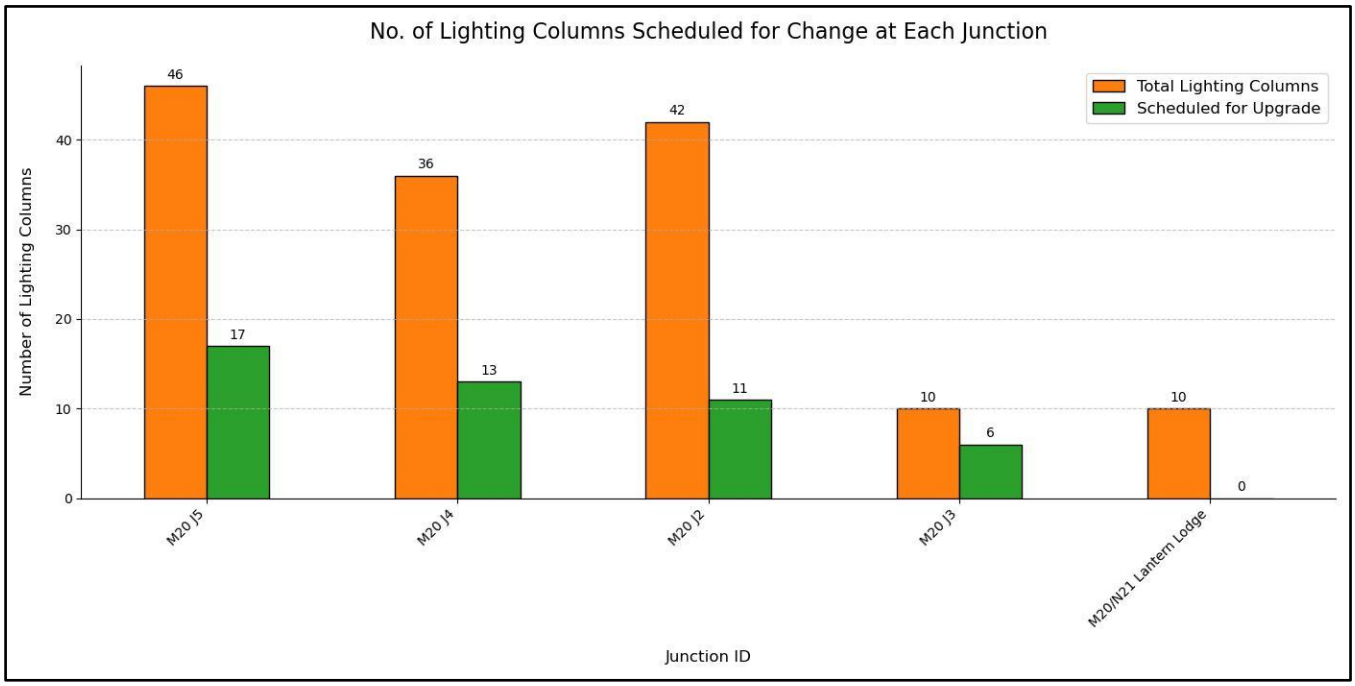


Figure 10 : Bar Chart showing the Number of Lighting Columns Scheduled for Change at Each Junction

3. A table (See Table 1) showing the results of the cost savings analysis (See section 3.5)

Junction	No. of Upgraded Lighting_Columns	Annual Savings
M20 J5	17	€7,446.00
M20 J4	13	€5,694.00
M20 J2	11	€4,818.00
M20 J3	6	€2,628.00
M20/N21 Lantern Lodge	0	€0.00

Table 1 : Table summarising the Annual Savings of Upgraded Lighting Columns

4. A .CSV file called “lighting_column_data.csv” will be created in the GitHub repository and be available for download. The file requires some additional steps to be taken in Excel. This will be discussed in **Section 5.0 Trouble Shooting**

5.0 Troubleshooting

This section will address three issues the author has encountered when using the Python Code:

1. **Error Message when using PyCharm**

   When testing the code , the following error message appeared:

```
import geopandas as gpd
ModuleNotFoundError: No module named 'geopandas'

Process finished with exit code 1
```

   This error signified that the interpreter was not set to the EGM722Project environment. This can be checked by opening the Terminal window in PyCharm. If the code starts with (base) it signifies that it is set to the base conda environment and not the EGM722Project environment.
   To switch to the EGM722Project environment, type " **conda activate EGM722Project**" (See Figure)
   The code should now run with no errors.

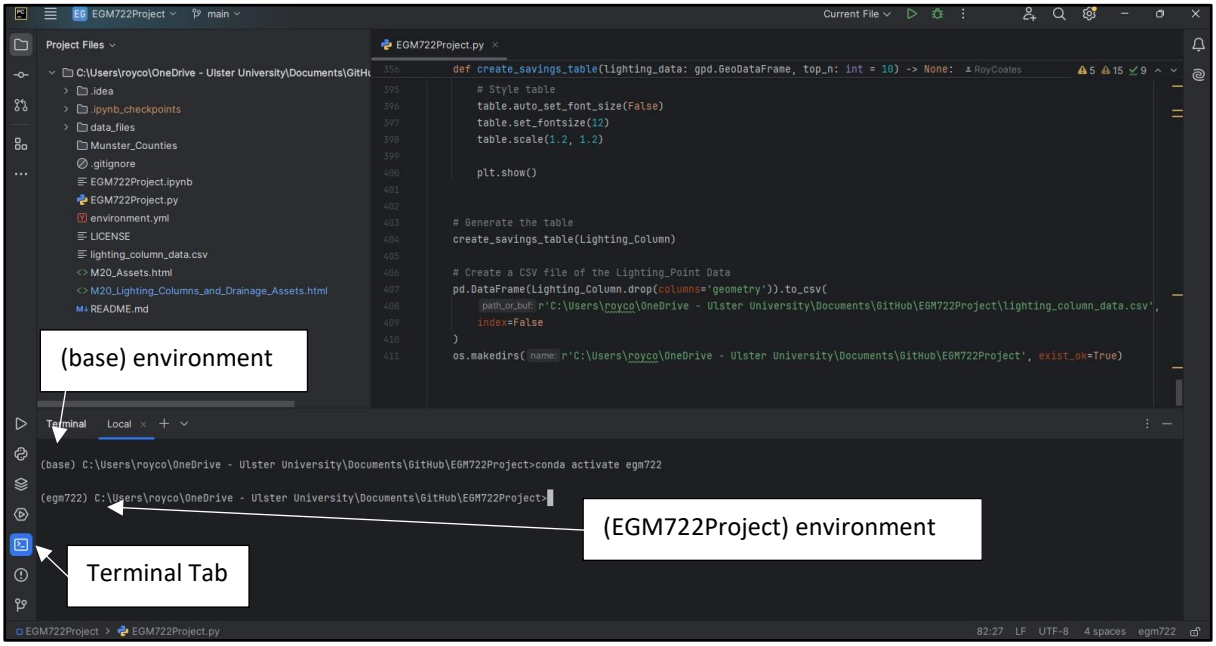


Figure : PyCharm terminal interface – (base) interpreter vs (EGM722Project) interpreter

2. **Using PyCharm, the bar chart graph and map appear but the table does not:**
   At first it appears that the code did not run fully, and the table was not created. However, all you must do is close the bar chart graph and the table will appear.

3. **Converting the csv into a workable file.**
   The author tried to import **Openpyxl** to use and export an excel file, but consistent error messages forced the author to abandon this and export a csv file instead. The data in the csv file needs to be transformed and split by a comma delimiter.

6.0 References

EarthLab Website: Available at:
https://www.earthdatascience.org/courses/scientists-guide-to-plotting-data-in-python/plot-spatial-data/customize-vector-plots/python-customize-map-legends-geopandas/

Folium: Available at:
https://python-visualization.github.io/folium/latest/

Geeks for Geeks. Available at:
https://www.geeksforgeeks.org/python-functions/
https://www.geeksforgeeks.org/create-a-legend-on-a-folium-map-a-comprehensive-guide/

GitHub (2017) Git Handbook
https://docs.github.com/en/get-started/using-git/about-git

McNabb, R, Holloway, P.(2025) EGM722-Programming for GIS and Remote Sending Course Notes and Practical's

Python GIS Tutorials Website: Available at:
https://python-gis-tutorials.readthedocs.io/en/main/index.html

Python Basics – matplotlib
https://pythonbasics.org/matplotlib-bar-chart/

Sweigart, A. (2020) Automate the boring stuff with python, 2nd edition, San Francisco, No Starch Press

The Python Standard Library Website: Available at:
https://docs.python.org/3/library/

TII – Transport Infrastructure Ireland Website: Available at:
https://www.tii.ie/en/roads-tolling/operations-and-maintenance/road-maintenance/

W3 schools. Learn to Code Website. Available at:
https://w3schools.com/python/default.asp
https://w3schools.com/python/matplotlib_bars.asp

Word Count: 2066