STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **14**, November 2025, pp 2663–2687.
Published online in International Academic Press (www.IAPress.org)

Characterization of Demand Profiles for Medium-Voltage Substations in Bogotá Using a Python Interface

María Camila Herrera-Briñez [1], Oscar Danilo Montoya [2,*], Walter Gil-González [3]

[1]*Maestría en Ingeniería Eléctrica, Facultad de Ingenierías, Universidad Tecnológica de Pereira, Pereira 660003, Colombia*
[2]*Grupo de Investigación en Compatibilidad e Interferencia Electromagnética, Facultad de Ingeniería, Universidad Distrital Francisco José de Caldas, Bogotá D.C. 110121, Colombia*
[3]*Departamento de Ingeniería Eléctrica, Facultad de Ingenierías, Universidad Tecnológica de Pereira, Pereira 660003, Colombia*

Abstract This document presents a comprehensive methodology for analyzing operational demand forecast data from XM's website using a Python script executed within the Google Colaboratory (Colab) environment. The script is designed to process forecast data stored in text files, which are compressed in a ZIP archive and uploaded to Colab for analysis. Its core functionalities include decompressing and organizing the files, filtering content based on specified keywords, calculating average power values, and normalizing these results. The script is structured into distinct sections: it begins by describing the libraries and functions employed, followed by a main section that integrates and executes these functions. The resulting vectors are then used to plot normalized active and reactive power curves, enabling the visualization of daily demand patterns over a 24-hour period. This graphical representation offers valuable insights into the operational dynamics of Colombia's National Interconnected System (SIN). The analysis focuses on forecast data spanning from May 2023 to May 2024, encompassing both active and reactive power. By integrating this script into the data processing workflow, the methodology streamlines the evaluation of XM's operational demand forecasts, thereby supporting more informed decision-making in energy planning and management. Ultimately, by converting raw forecast data into actionable insights, this approach enhances the understanding of demand behavior and contributes to the efficient management of Colombia's energy resources.

Keywords Operational Demand Forecast, Python Script, Power Demand Analysis, Data Processing Workflow, Colombian National Interconnected System

DOI: 10.19139/soic-2310-5070-2454

1. Introduction

Power demand projections are invaluable tools for identifying significant demand peaks and patterns. This analysis improves the accuracy of forecasting and resource allocation, enabling energy managers to predict high-demand periods with greater precision. By understanding these demand fluctuations, utilities can plan and manage energy resources more efficiently, ensuring stability and reliability in the power supply [1]. Furthermore, access to power demand patterns is crucial for improving grid stability, as it allows operators to anticipate and address potential issues related to overloads or outages during peak periods. It also contributes to cost savings; By forecasting high-demand times, utilities can secure energy at lower costs during off-peak periods and avoid purchasing expensive emergency power during peak times [5]. Furthermore, accurate demand analysis supports regulatory compliance by helping utilities meet requirements related to grid reliability and emission reduction, thus avoiding penalties and maintaining operational licenses [13]. Finally, precise demand forecasting enables utilities to provide more reliable

*Correspondence to: O. D. Montoya (Email: odmontoyag@udistrital.edu.co). Facultad de Ingeniería, Universidad Distrital Francisco José de Caldas, Bogotá D.C. 110121, Colombia

ISSN 2310-5070 (online) ISSN 2311-004X (print)

service to customers, reducing the likelihood of service interruptions and enhancing overall customer satisfaction [12].

The demand forecast for the Colombian National Interconnected System is available on Sinergox [11], a website managed by XM. This platform offers various demand trends, including industrial power demand and losses, categorized by regions and customer types (regulated and non-regulated). Users can also access real-time daily demand forecasts, providing valuable insights into demand patterns across the Colombian SIN (National Interconnected System). However, the site does not offer access to historical demand curves. Additionally, XM has developed an API that enables users to query relevant information related to the energy matrix, including available and used generation, energy market prices, CO2 emission calculations, and demand data [15]. While this API can be a powerful tool for accessing detailed information, it requires data processing skills and a solid understanding of API functionality to generate useful graphs and insights.

A key player in Colombia's energy industry is XM, which manages the Colombian National Interconnected System (SIN) through its National Dispatch Center (NDC) and oversees the Wholesale Energy Market. Effective management of the SIN is crucial for maintaining the stability and reliability of Colombia's power supply. To achieve this, XM relies on comprehensive demand forecast data collected from various Forecast Control Units (UCPs) spread across the country. Accurate demand forecasting is essential for efficient energy distribution and grid management. XM consolidates and publishes this forecast data on its official website [17], providing valuable insights into the expected operational demands of the SIN. The data is presented in monthly reports in .txt format, covering historical records up to January 2018. For this study, we focus on the forecast data from May 2023 to May 2024.

The methodology outlined in this document leverages Python to process and analyze the forecast data. Python, an open-source programming language, is widely used for data analysis due to its versatility and the extensive range of libraries available [8]. Google Colaboratory (Colab), a hosted Jupyter Notebook service provided by Google, is employed for this analysis. Colab offers a convenient, no-setup environment with free access to powerful computing resources, making it an ideal choice for processing and analyzing large datasets [2].

The primary objective of this study is to integrate and analyze XM's operational demand forecast data, focusing on reactive and active power. By employing Python scripts, we aim to filter relevant data, calculate average power values, and normalize these values to identify demand patterns over a 24-hour period. This approach provides actionable insights into energy demand, supporting more informed decision-making for energy management and planning.

The script described herein is designed to streamline the data processing workflow. It includes functions for reading and extracting data from .txt files, calculating averages, and normalizing power values. Additionally, it visualizes the results using plots to highlight trends and fluctuations in power demand. This methodology not only enhances our understanding of the SIN's demand patterns but also contributes to more effective energy resource management.

The implementation of this Python script for analyzing XM's operational demand forecast data is crucial for optimizing energy management in Bogotá. By leveraging advanced data processing and visualization capabilities, the script transforms raw datasets into actionable insights, revealing significant demand patterns and peaks. Ultimately, the implementation of this script not only improves the accuracy of demand forecasts but also contributes to more effective resource utilization, better grid management, and sustainable energy practices.

This paper is organized as follows. Section 2 details the methodology for implementing the Python script, including comprehensive instructions on how to locate and download the necessary source data. It also provides a step-by-step explanation of the functions used to generate the normalized active and reactive power curves. Section 3 presents the results and a discussion. Section 4 offers the main conclusions of this study. Finally, Annex I includes the monthly curves for the specified time of the use case.

2. Methodology

This section describes the methodology developed to analyze operational demand forecasts provided by XM, the operator of Colombia's National Interconnected System (SIN). The analysis focuses specifically on the demand data for Bogotá, which is published on XM's official website in monthly text file reports. To automate and streamline the processing of this information, a Python script was implemented and executed within Google Colaboratory. The script enables the reading, filtering, normalization, and visualization of active and reactive power demand curves, facilitating insights into daily consumption patterns. The following subsections detail the source of the data, the structure and logic of the script, and the procedures used to prepare and analyze the information.

2.1. Power demand information for Bogotá

XM is responsible for managing the SIN through the NDC and oversees the Wholesale Energy Market. To effectively manage the system and meet demand requirements, XM gathers demand forecast data from UCPs.

This data is consolidated and made available on XM's official website [16]. The website hosts the definitive forecast of operational demands for the SIN, sourced from UCP administrators. This data is crucial for conducting the Economic Dispatch from the upcoming Monday to the following Sunday, as mandated by Resolution CREG 025 of 1995, Agreement CNO 1303.

The forecast data is organized into monthly reports in .txt format, spanning from historical records up to January 2018. For the purposes of this study, data from May 2023 to May 2024 was selected. To use this data with the Python script provided below, simply download the .txt files from XM's website [16], without altering the folder or file names.

The Python script referenced below is designed to read and analyze these .txt files, facilitating the integration of XM's forecast data into various operational analyses and models.

2.2. Python script design description

This script is designed to be user-friendly and accessible, leveraging Google Colaboratory (Colab), a hosted Jupyter Notebook service. Colab requires no setup and offers free access to computing resources, making it an ideal environment for data analysis tasks [14]. By leveraging Colab, users can harness powerful computing resources and collaborate seamlessly on data-driven projects.

The script facilitates the integration of XM's operational demand forecast data into various analyses. Simply download the required .txt files from XM's website and upload them to your Colab session for processing. The script handles reading and analyzing these files, providing insights into demand patterns within the Colombian National Interconnected System. Figure 1, shows the workflow of the script implementation.

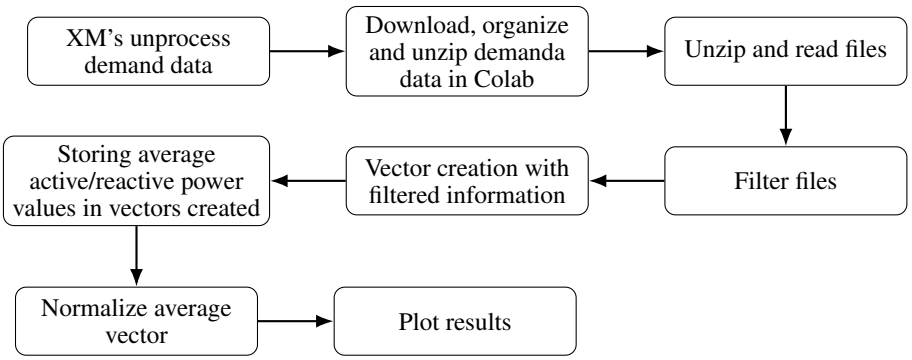

Figure 1. Implemented script workflow. Source: Self-made.

The steps outlined in Figure 1 are described as follows:

- XM's Unprocessed Demand Data: These are text files containing raw demand information, as detailed in Section 2.2.2. The data includes various parameters relevant to power demand but requires processing for analysis.
- Download, Organize, and Upload Demand Data in Colab: The first step involves downloading the text files and organizing them into a single folder. Afterward, the folder is uploaded to Google Colaboratory (Colab). Colab provides a fully functional Python environment within the browser, enabling users to write, execute, and refine Python code. Popular libraries such as Pandas, NumPy, and Matplotlib can be utilized for data manipulation, analysis, and visualization. This setup ensures a seamless platform for complex data analysis and insightful visualizations. Further details on this process are explained in Section 2.2.2.
- Unzip and Read Files: Once uploaded, the files are unzipped and read into the Colab environment. This step is crucial for making the data accessible for subsequent processing, as described in Section 2.2.2.
- Filter Files: In Section 2.2.3, the unzipped files are filtered based on specific keywords related to UPC names and the desired voltage levels of substations. This filtering process ensures that only the relevant information is extracted from the text files.
- Vector Creation with Filtered Information: As detailed in Section 2.2.4, vectors are created to store the filtered information. These vectors are designed to efficiently handle and organize the data for further analysis.
- Store Average Active/Reactive Power Values in Vectors: The active and reactive power values are stored in the previously created vectors. This storage step is essential for processing the data, as described in Section 2.2.4.
- Normalize Average Vectors: Normalization is performed to ensure comparability of the data. In Section 2.2.5, the average active and reactive power vectors are normalized to standardize the values and facilitate meaningful comparisons.
- Plot Results: Finally, in Section 2.2.7, the normalized vectors are visualized through plots. This step allows for the graphical representation of the data, highlighting trends and fluctuations in power demand.

2.2.1. Python Libraries: To ensure the script's functionality and enhance its capabilities for data analysis and visualization, three Python libraries are utilized, as showed in Code 1:

- os: This module provides a portable way of using operating system dependent functionality [9]. This library is useful for reading, opening, and modifying files within Python.
- zipfile: This module provides tools to create, read, write, append, and list a ZIP file [10].
- matplotlib.pyplot: Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python [4], we used it to plot the final demand graphics.
- gdown: is an open-source Python library designed to facilitate the downloading of files and folders from Google Drive, used to load the sample data.

```
1 import os
2 import zipfile
3 import matplotlib.pyplot as plt
4 import gdown
```

Code 1: Python Libraries used.

2.2.2. Data preparation: As outlined in Section 2.2.2, foundational data for the script is available on XM's website. Follow these steps to prepare and upload the information into Colab:

1. Download Operational Demand Data:
   - Visit XM's website and download the desired months of forecast operational demands using this link. XM provides open and public access to aggregated electricity data such as generation, demand, and market prices. This data is strictly technical and system-level, containing no personally identifiable information, and thus, there are no anonymization requirements or privacy restrictions typically associated with personal data. XM does not impose limitations on the use or redistribution of these

datasets, which are made available to promote transparency, research, and operational analysis. While users may consult the API usage terms for specific conditions, the general policy supports free academic, analytical, and institutional use of the information without restriction.
- It is advisable to download data for consecutive months to ensure reliable results. This process may take some time depending on the data size.
- Ensure data integrity by downloading during off-peak times.

2. Organize Files:
   - Once downloaded, organize the files into a single folder. For convenience in the script, zip the folder and name it "BogCurves.zip.".
3. To Upload the folder into your Google Drive and use it in Google Colaboratory, follow these steps:
   - Open Google Drive: Navigate to https://drive.google.com Sign in with the Google account linked to your Colab notebook.
   - Click + New > File Upload. Select the file (e.g., BogCurves.zip file) from your local computer. Wait for the upload to finish; the file will appear in the folder.

```
5  file_id = "1wSsWnN5JmoaWdk_v31mK3rTNuGmyV7Py"
6  zip_file_path = "/content/curves_data.zip"
7  gdown.download(f"https://drive.google.com/uc?id={file_id}", zip_file_path, quiet=False)
8
9  extracted_directory = '/content/curves/BogCurves'
10 if not os.path.exists(extracted_directory):
11     try:
12         with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
13             zip_ref.extractall('/content/curves')
14     except zipfile.BadZipFile:
15         print("[ERROR] The zip file is corrupted or unreadable.")
16         exit(1)
```

Code 2: Script to read and unzip files.

Code 2 performs the following key steps:

- **Define `file_id`:** This is the unique identifier of the file hosted on Google Drive. It can be extracted from the shareable link of the file. For instance, in the sample data URL https://drive.google.com/file/d/1wSsWnN5JmoaWdk_v31mK3rTNuGmyV7Py/view, the file ID is `1wSsWnN5JmoaWdk_v31mK3rTNuGmyV7Py`. Adjust the file id as needed, in line 5 of Code 2. Please make sure your Google Drive folder is set to public or anyone with the link can access it, and that it is not restricted by private sharing permissions.
- **Specify the download path:** The script defines the full local path where the file will be saved. In this case, it is stored as `/content/curves_data.zip` within the Colab environment, as shown in line 6.
- **Handle extraction errors:** If the ZIP file is corrupted or unreadable, the script raises a `zipfile.BadZipFile` exception. It then prints an error message and terminates the execution using `exit(1)`.

Finally, update line 9 by modifying the `extracted_directory` variable to reflect the name of the folder where your data is located in Google Drive. This ensures that the script can correctly locate and access the necessary files for analysis within your Colab environment. Make sure to verify and adjust all file paths and names in the script to match your specific directory structure and file organization.

Lines 17 and 18 in Code 3, a list called "directories"is created based on the folders in the donwloaded zip file (BogCurves.zip). This list will have a format such as each included month is shown as YYYY-MM. There is not a limit on the number of months the script can process, however for larger amounts of data performance time will increase.

```
17 directories = [os.path.join(extracted_directory, dir) for dir in os.listdir(
       extracted_directory)
```

```
18                 if os.path.isdir(os.path.join(extracted_directory, dir))]
```

Code 3: Script for directory creation.

2.2.3. Files Filtering: After unzipping a function to parse through all text files located at ”file_path” and extract specific data into a list named ”vectors” was developed. The function utilizes parameters ”word” and ”filter_word” as keywords to filter lines based on specified criteria

To illustrate this function, consider two example lines extracted from the file PRON_BARRA0531.txt found in the 2023-05 folder on XM's website.

Uandaki,FLORENC1,115,Ener.Act.
Uandaki,FLORENC1,115,Ener.Reac.

Each line displays the first five relevant items: The first item denotes the UPC name, the second item signifies the node name, the third item indicates the node's voltage, and the final item specifies the type of information contained in that line.

```
19 def search_and_append(file_path, words, filter_word, vectors):
20     try:
21         with open(file_path, 'r', encoding='utf-8') as file:
22             for line in file:
23                 if any(word in line.lower() for word in words):
24                     parts = line.strip().split(',')
25                     if len(parts) >= 5 and filter_word in line and "115" in parts[2]:
26                         try:
27                             vector = [float(item) for item in parts[4:]]
28                             vectors.append(vector)
29                         except ValueError:
30                             print(f"[WARN] Skipping invalid line in {file_path}: {line.
    strip()}")
31     except Exception as e:
32         print(f"[ERROR] Failed to read {file_path}: {e}")
```

Code 4: Python Filtering function.

The provided code in Code 4. demonstrates the implementation of a filtering function named search_and_append. Here is a detailed breakdown: In line 19, the function is defined with the following input parameters:

- file_path: Path to the text file that needs to be filtered and processed.
- words: List of keywords used to identify relevant lines within the text file. These keywords are matched in a case-insensitive manner.
- filter_word: Used to distinguish between diverse types of data when multiple lines contain keywords from words.
- vectors: Name of the list where filtered lines' data will be stored.

Lines 21 and 22 in Code 4 open the text file specified by “file_path” and initiate the process of iterating through each line (“line”) within the opened file (“file”). Line 23 checks if any word from the “words” list (converted to lowercase for case insensitivity) is present in the current line. If a match is found, the function proceeds to the next steps. Line 24 strips any leading or trailing whitespace from “line” and splits it into segments based on commas. This format is typical for the structure of text files being processed.

Line 25 validates several conditions: It ensures there are at least five elements in “parts” to prevent errors when accessing elements. It verifies the presence of “filter_word” in line. It checks if the third element in “parts” equals ”115”, indicating nodes with a voltage of 115kV. This last criterion can be adjusted to filter based on different voltages specified in the text files, please refer to Annex II, for a list of available voltage levels to filter.

If all conditions are met, line 27 creates a list named “vector” containing floating-point numbers converted from elements starting from the fifth element onward in “parts”. Line 28 appends the generated vector (list of floats) to the vectors list. This accumulates all valid vectors that meet the specified criteria from the file.

The script includes two exception-handling blocks to ensure robust data processing. The first block captures ValueError exceptions that may arise when attempting to convert a line of text into numerical values. This typically occurs when the line contains missing or malformed data. When such an error is encountered, the script issues a warning message as shown in line 29 and 30, allowing the process to continue without interruption by skipping only the problematic line. The second block is a broader Exception handler that captures any other unforeseen errors that may occur while opening or reading the file, such as encoding issues or file access problems. In such cases, an error message is displayed via in lines 31 and 32, indicating the file that failed and the cause of the failure.

2.2.4. Average calculation: This function efficiently computes the average value for each position (index) across a list of vectors. It handles cases where no vectors are provided by returning an empty list. The computation involves summing up values at each position across all vectors and then dividing these sums by the number of vectors to get the average value for each position. This approach assumes all vectors have the same length and that each vector contains numeric values that can be summed and averaged. The only input parameter is the vector containing the floating numbers obtained in Section 2.2.3.

```
33 def calculate_average(vectors):
34     if not vectors:
35         return []
36     vector_length = len(vectors[0])
37     if any(len(v) != vector_length for v in vectors):
38         raise ValueError("Mismatched vector lengths")
39     num_vectors = len(vectors)
40     position_sums = [0] * vector_length
41     for vector in vectors:
42         for i, value in enumerate(vector):
43             position_sums[i] += value
44     position_averages = [position_sum / num_vectors for position_sum in position_sums]
45     return position_averages
```

Code 5: Python function for average vector calculation.

In lines 34 and 35 of Code 5 the function verifies the existence and non-emptiness of the list "vectors." Lines 36 and 39 determine the length of each vector by accessing the first vector and count the total number of vectors in "vectors". En exception for mismatched vector length is used in lines 37 and 38.

In line 40 the list "position_sums " is initialized with zeros, where each element corresponds to the sum of values at a specific position (index) across all vectors.

Line 41 iterates through each vector in "vectors". Within this loop, line 42 iterates through each element ("value") in the current vector, capturing its index "i". Line 43 accumulates each "value" into the corresponding position sum in "position_sums".

Line 44 computes the average for each position (index) by dividing the sum of values ("position_sum") at each position by the total number of vectors ("num_vectors"). Finally, the function returns a list named "position_averages" containing the calculated averages for each position across all vectors.

2.2.5. Vector Normalization: This function is designed to normalize a given vector using Max value normalization, this is a technique where each element of a vector is divided by the maximum value in that vector. For a vector $\mathbf{x} = [x_1, x_2, ..., x_n]$, the normalized value is calculated as:

$$x'_i = \frac{x_i}{\max(\mathbf{x})}$$

This approach is particularly suitable for normalizing power values for the following reasons:

- Preservation of Relative Proportions: Max-value normalization maintains the ratio of the power value of each substation to the highest. For instance, if a substation consumes half the power of the most demanding one, its normalized value becomes 0.5. This ensures intuitive and meaningful comparisons.

- No Minimum Value Adjustment Required: Unlike full min-max normalization, which rescales data to the range $[0, 1]$ using both minimum and maximum values, max-value normalization only requires the maximum. This is advantageous when zero has physical meaning (e.g., zero power), and all values are non-negative.
- Robustness to Data Skew: In cases where a few substations have significantly higher demand than others (e.g., industrial zones), max normalization compresses the range appropriately without introducing distortion from low-value outliers.

For this specific use case, the input parameter is the vector containing the average values obtained in Section 2.2.4.

```
46 def normalize_vector(vector):
47     if not vector:
48         return []
49     max_value = max(vector)
50     return [v / max_value if max_value != 0 else 0 for v in vector]
```

Code 6: Python function normalize to vectors.

Line 49 of Code 6 identifies the maximum value within the "vector" list. This maximum value serves as the reference for normalizing all elements within the vector.

In line 50, the function returns the normalized vector, as a list comprehension constructs a new list. Within this comprehension, each element ("value") in the "vector" is divided by "max_value." This process iterates through each element of the vector, performing the division operation to normalize each value. This approach ensures that every element in the vector is scaled proportionally to the maximum value found within that vector.

2.2.6. Script implementation: This code iterates through a list of directories ("directories"), for each directory, it processes text files (ending with '.txt') found within. It extracts specific data ("vectors") related to reactive and active power from these files using the "search_and_append" function. It then calculates averages of these vectors and normalizes these averages.

```
51 for directory in directories:
52     react_p = []
53     active_p = []
54 
55     for filename in os.listdir(directory):
56         if filename.endswith('.txt'):
57             file_path = os.path.join(directory, filename)
58             search_and_append(file_path, ['bogota', 'codensa'], 'Ener.Reac.', react_p)
59             search_and_append(file_path, ['bogota', 'codensa'], 'Ener.Act.', active_p)
60 
61     try:
62         average_reac_power = calculate_average(react_p)
63         average_act_power = calculate_average(active_p)
64 
65         if not average_reac_power or not average_act_power:
66             print(f"[INFO] Skipping {directory}: not enough data.")
67             continue
68 
69         if len(average_reac_power) != 24 or len(average_act_power) != 24:
70             print(f"[WARN] Skipping {directory}: vector length != 24")
71             continue
72 
73         normalized_reac = normalize_vector(average_reac_power)
74         normalized_act = normalize_vector(average_act_power)
75 
76         positions = list(range(1, 25))
```

Code 7: Python main script implementation.

In line 51 of Code 7, a loop is initiated to iterate over each element in the list named "directories", which was previously established in Section 2.2.2. Lines 52 and 53 are dedicated to creating empty vectors for reactive and

active power, respectively. Line 55 further advances by iterating over each file within the current "directory". In line 56, a check is performed to verify if the file has a ".txt" extension, which signifies that it is a text file. Line 57 constructs the full path to the file by combining the "directory" path with the "filename".

Subsequently, lines 58 and 59 invoke the function "search_and_append" (refer to Section 2.2.3), using the filter parameters 'bogota' and 'codensa' to extract the required information. The function splits the information based on the keywords 'Ener.Reac.' for reactive power and 'Ener.Act.' for active power, saving the results in separate vectors: "react_p" and "active_p" respectively.

Lines 62 and 63 call the "calculate_average" function from Code 7. for the "react_p" and "active_p" vectors and store the results in "average_reac_power" and "average_act_power." Lines 65 to 67 display an informational message if either "average_reac_power" or "average_act_power" is missing, indicating that the data is insufficient to generate a plot. Lines 69 to 71 issue a warning if the length of the vectors is not equal to 24, alerting the user that the dataset may contain incomplete or misleading information. In lines 73 and 74, these average power lists are then normalized using the "normalize_vector" function, as described in Section 2.2.5. Average vector normalization. Finally, line 76 generates a list of integers ranging from 1 to 24 to facilitate the plotting of the results.

2.2.7. Plotting the results: The code generates a plot that visualizes the normalized average reactive and active power values over a range of 24 hours. It starts by creating a figure with specified dimensions and then plots the normalized data for both reactive and active power, using distinct markers and labels. The plot is enhanced with axis labels, a title that includes the current directory's name, and a legend to differentiate between the two data series. Data points are annotated with their corresponding average values, formatted to two decimal places, and colored accordingly for clarity. The completed plot is saved as a PNG file, named based on the directory specified in the "extracted_directory" path defined in Section 2.2.2, the figure is closed to release resources. This process helps in visualizing and analyzing the power data across different directories effectively. The code describe is shown in Code

```
77  fig, ax1 = plt.subplots(figsize=(10, 6))
78  ax1.plot(positions, normalized_reac, label='Reactive Power (P.U.)', color='blue',
        marker='o')
79  ax1.plot(positions, normalized_act, label='Active Power (P.U.)', color='green', marker=
        'o')
80  ax1.set_xlabel('Hour')
81  ax1.set_ylabel('Normalized Value (P.U.)')
82  ax1.set_xticks(range(0, 25, 2))  # Every 2 hours on X-axis
83  min_val = min(min(normalized_reac), min(normalized_act))
84  min_tick = round(min_val - 0.01, 2)
85  min_tick = max(0, min_tick)
86  yticks = [round(v, 2) for v in np.arange(min_tick, 1.05 + 0.001, 0.05)]
87  ax1.set_yticks(yticks)
88  ax1.set_ylim(min_tick, 1.05)
89  ax1.grid(True)
90  for i, (x, y_reac, y_act) in enumerate(zip(positions, normalized_reac, normalized_act),
        start=1):
91      real_reac = average_reac_power[i - 1]
92      real_act = average_act_power[i - 1]
93      ax1.text(x, y_reac, f"{real_reac:.1f}", ha='center', va='bottom', fontsize=8, color
        ='blue')
94      ax1.text(x, y_act, f"{real_act:.2f}", ha='center', va='top', fontsize=8, color='
        green')
95  max_val_reac = max(average_reac_power)
96  max_val_act = max(average_act_power)
97  ax1.text(0.5, 0.975, f"Max Reactive: {max_val_reac:.2f} MVAr", fontsize=10, color='blue
        ')
98  ax1.text(0.5, 0.95, f"Max Active: {max_val_act:.2f} MW", fontsize=10, color='green')
99  plt.title(f'Normalized Active and Reactive Power for {os.path.basename(directory)}')
100 ax1.legend(loc='upper left')
101 plt.tight_layout()
102 plt.savefig(f"{directory}_plot.png")
103 plt.close()
```

Code 8: Python to plot normalized demand curves.

2.2.8. Debugging compile errors: Common errors in the script may arise from issues related to file formatting or incorrect file uploads from XM's website. Although the script includes exception handling to manage missing information, formatting errors, or mismatched dimensions, some problems can still occur. Below are some suggestions for addressing frequent runtime or data processing errors:

- Verify File Format: Ensure that the files are in the correct format (e.g., .txt) and are properly structured as expected by the script.
- Check File Uploads: Confirm that the files have been uploaded correctly from XM's website in Google Drive. Re-upload if necessary to correct any issues.
- Validate Keywords: Ensure that the keywords used in the script match those present in the text files. Keywords are not case-sensitive, but any discrepancy can result in empty vectors, please review Annex II, for a full list of available keywords.
- Inspect File Paths: Double-check that the file paths are constructed correctly and that all files are accessible from the specified directories.
- Error: No such file or directory: '/content/BogCurves.zip" Filters not matching the files criteria: Ensure that steps mentioned in Section 2.2.2. were follow and the zipped files containing the text files in the right location and matches the name in line 9 of Code 2.
- Error: ValueError: max() arg is an empty sequence: Ensure that the keywords used in section iv, Filtering the Files Using Keywords, are present in the text files. Since the keywords are not case-sensitive, a case mismatch will still lead to empty vectors. Consequently, verify that the keywords 'Ener.Reac.' for reactive power and 'Ener.Act.' for active power are correctly specified and match the contents of the text files, please note that city and UPC name can also cause conflicts, please review Annex II, for a full list of available keywords. If the vectors are empty due to a keyword mismatch, the max() function will fail, leading to this error. Double-check and update the keywords if necessary to ensure that they are present in the files being processed.
- Error: ValueError: x and y must have same first dimension: Make sure the files uploaded contain at least a few lines reactive and active power for 24 hours.

2.2.9. Code and sample data accessibility: Open code in Colab is available in this link or go to this url: https://colab.research.google.com/drive/1KRYCy66cG6z__UjwqjBtB9RXvwkLPi95.

A sample dataset used for testing this script can be accessed via the following Google Drive link: https://drive.google.com/file/d/1wSsWnN5JmoaWdk_v31mK3rTNuGmyV7Py

3. Results and Discussion

The Python script successfully processes the forecast data by iterating through the provided directories and extracting relevant information from the text files. For each directory, it differentiates between reactive and active power data using specific keywords. The script then calculates the average power values for each type, normalizes these averages, and generates plots showing the normalized power values over 24 hours. The Python script functions as an exceptionally valuable tool for the ongoing analysis and monitoring of energy demand, significantly enhancing the efficiency and accuracy of energy management practices. Its integration into the data processing workflow provides a robust framework for continuously improving forecasting accuracy and resource planning efforts.

The script's capability to handle, process, and analyze large datasets from multiple sources ensures that energy demand can be monitored in real-time or at regular intervals. By automating the data extraction, filtering, and normalization processes, the script enables a streamlined approach to tracking and evaluating demand patterns. This continuous monitoring is crucial for maintaining up-to-date insights into energy consumption trends, allowing stakeholders to promptly address any emerging issues or changes in demand. By integrating the Python script

into the data processing workflow, organizations can leverage its analytical power to refine and enhance demand forecasting models.

The Python script's design for use within platforms like Google Colaboratory (Colab) ensures that it can be easily incorporated into existing workflows without requiring extensive setup or technical expertise. Its user-friendly interface and powerful computational capabilities make it accessible for regular use by energy analysts and managers, facilitating seamless integration into routine data processing and analysis tasks.

Figure 2 displays the reactive and active power outputs from May 2023 to May 2024 in Bogotá for 115kV nodes, respectively. To help in analysis, average values have been incorporated into each data point. Given that all the graphics are normalized, they serve as a potent tool for comparing demand forecasts for Bogotá over the specified period. These visualizations provide critical insights into the operational demand patterns of the SIN [7], thereby enhancing the ability to plan and manage energy resources more effectively.

Focusing exclusively on 115 kV substations for Bogotá establishes a consistent and representative baseline for the study. The 115 kV level is one of the most common voltage levels used in regional transmission and distribution systems in Colombia, providing a broad and uniform sample of system behavior across different areas. Additionally, selecting a single voltage level ensures that the results are not influenced by technical differences associated with higher (e.g., 230 kV, 500 kV) or lower (e.g., 34.5 kV) voltage levels, which may follow different operating regimes or usage profiles. This helps isolate trends in active and reactive power curves attributable to demand-side behavior or regional conditions, rather than equipment specifications.

The data reveals distinct patterns in active power consumption, with notable peaks occurring during two specific time intervals: from 11 am to 1 pm and from 6 pm to 9 pm. These times correspond to the well-documented peak power demand periods in Colombia. By analyzing these peaks, stakeholders can gain a clearer understanding of the demand dynamics and refine their strategies for energy distribution and resource allocation to better align with these high-demand periods. This comprehensive analysis underscores the importance of detailed monitoring and forecasting in optimizing energy management practices. The reactive power data exhibits variability that often corresponds with the active power peaks, but with different magnitudes and timings. This variation can be attributed to the different nature of reactive power, which is influenced by factors such as load types and power factor correction measures.

Reactive power profiles play a technically significant role in understanding the operational behavior and stability of power systems. Unlike active power, which represents the actual energy consumed by end-users, reactive power is essential for maintaining voltage levels and enabling the efficient transmission of active power across the network. Its patterns are heavily influenced by the nature of electrical loads, such as inductive components found in motors, transformers, and industrial machinery. In the analyzed data, the reactive power curves often exhibit peak values that align closely with those of active power, particularly during the typical high-demand intervals identified earlier—namely, from late morning to early afternoon and during the evening hours. This correlation suggests that as energy consumption increases, the reactive power demand also rises due to the simultaneous operation of devices requiring magnetizing current. However, the magnitude and timing of reactive power peaks can differ slightly, reflecting the impact of load diversity and the implementation of power factor correction mechanisms across different sectors. Understanding these profiles is crucial for system operators, as improper management of reactive power can lead to voltage instability, increased losses, or even system failures. Thus, detailed monitoring of both active and reactive power is essential for maintaining grid reliability and optimizing energy distribution strategies.

Even if similar analyses are not found in the literature—especially those that include both reactive and active power consumption specifically for the residential sector or for Bogotá—comparison can still be made with articles that examine patterns of active power. Additionally, in the article "Electricity Demand Curve for Residential Sector, Socioeconomic Stratum Three in Bogotá D.C. Using Characterization by Artificial Neural Networks" [6], even though the focus is solely on the residential sector in Bogotá, and the authors' curve is not normalized and thus cannot be directly compared with the curves from this study, similar patterns and behaviors can still be observed in the 24-hour demand curve.

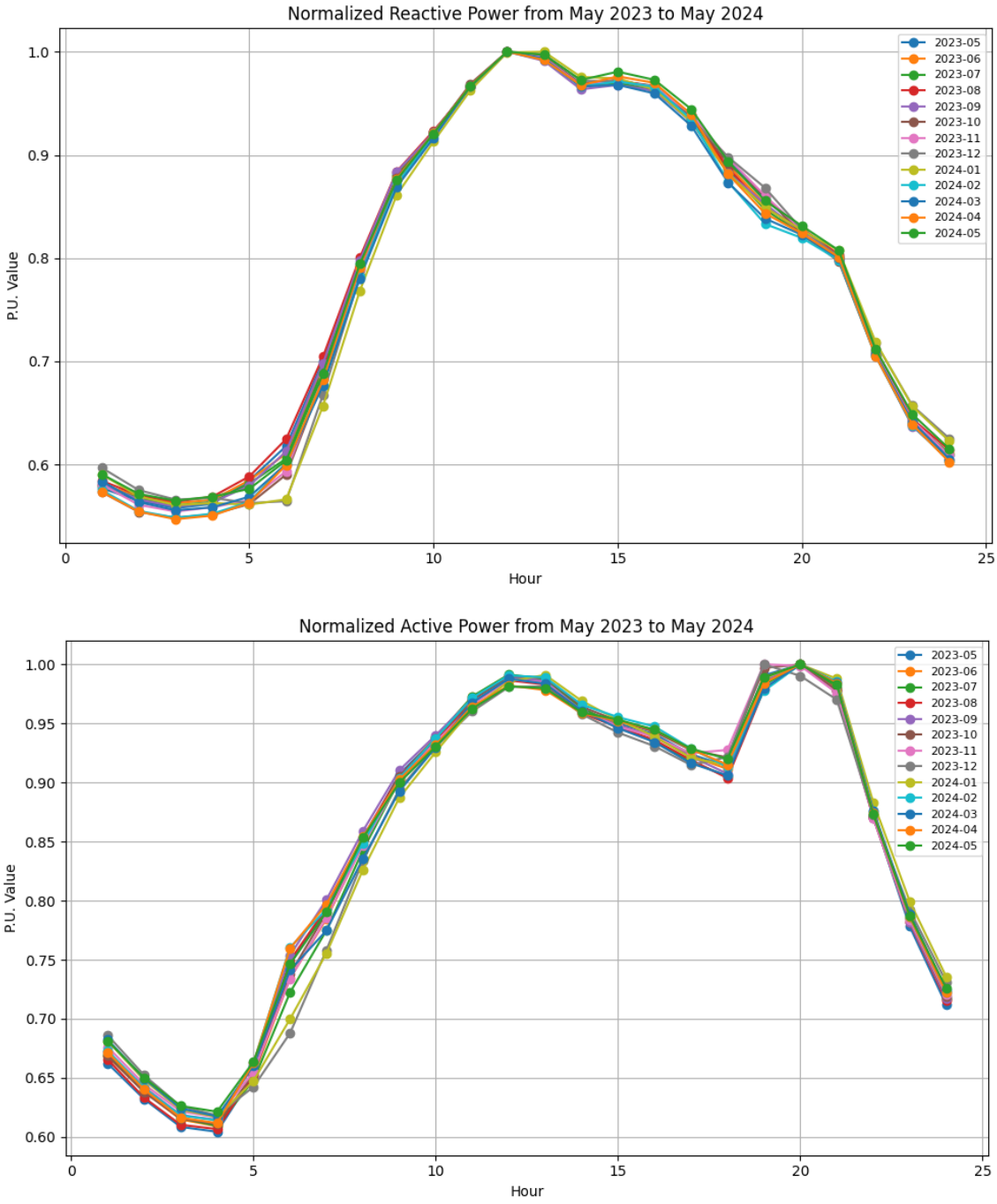

Figure 2. Normalized reactive and active power from May 2023 to May 2024 for Bogotá in 115kV nodes. Source: Self-made.

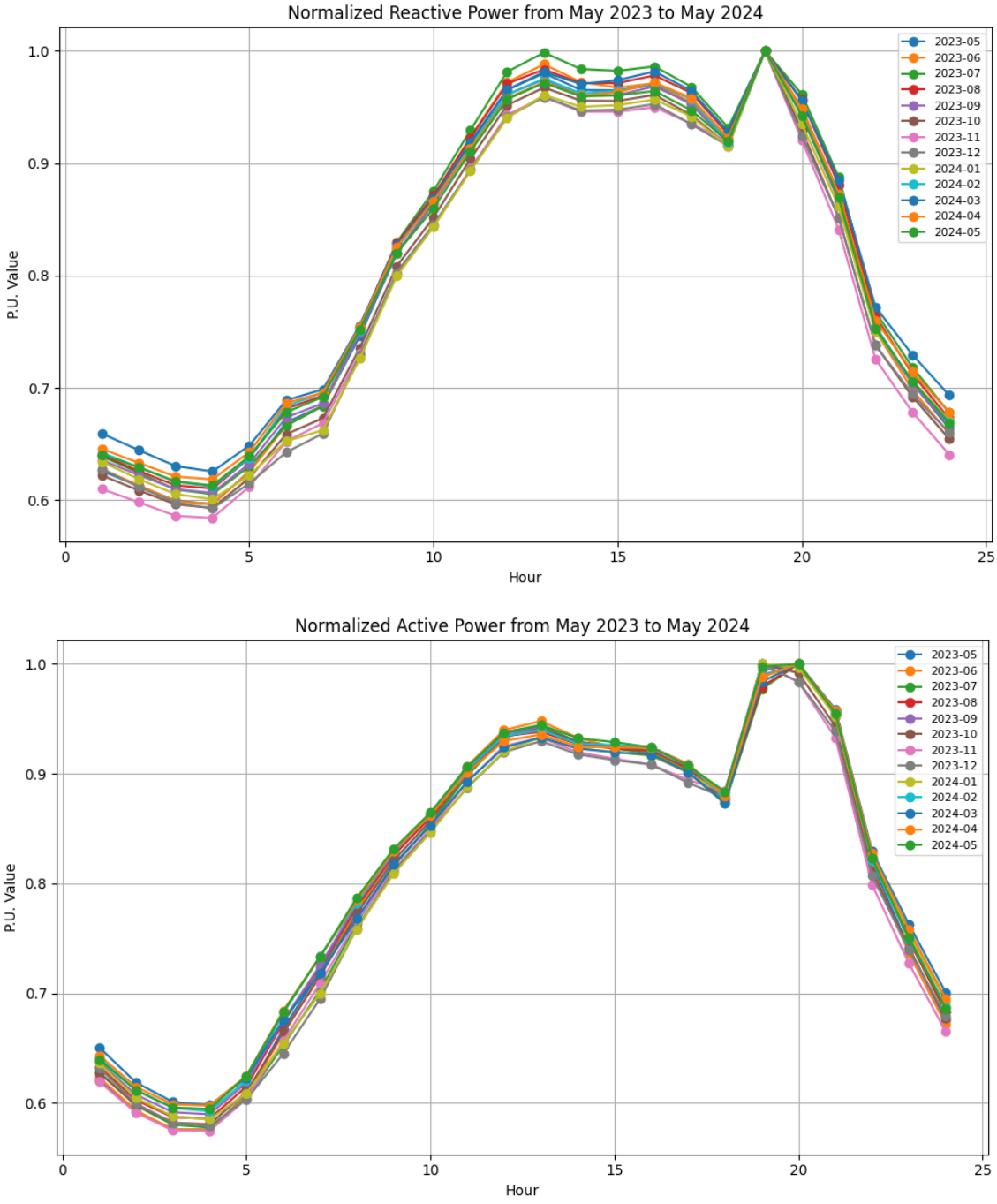

Figure 3. Normalized reactive and active power from May 2023 to May 2024 for Antioquia in 110kV nodes. Source: Self-made.

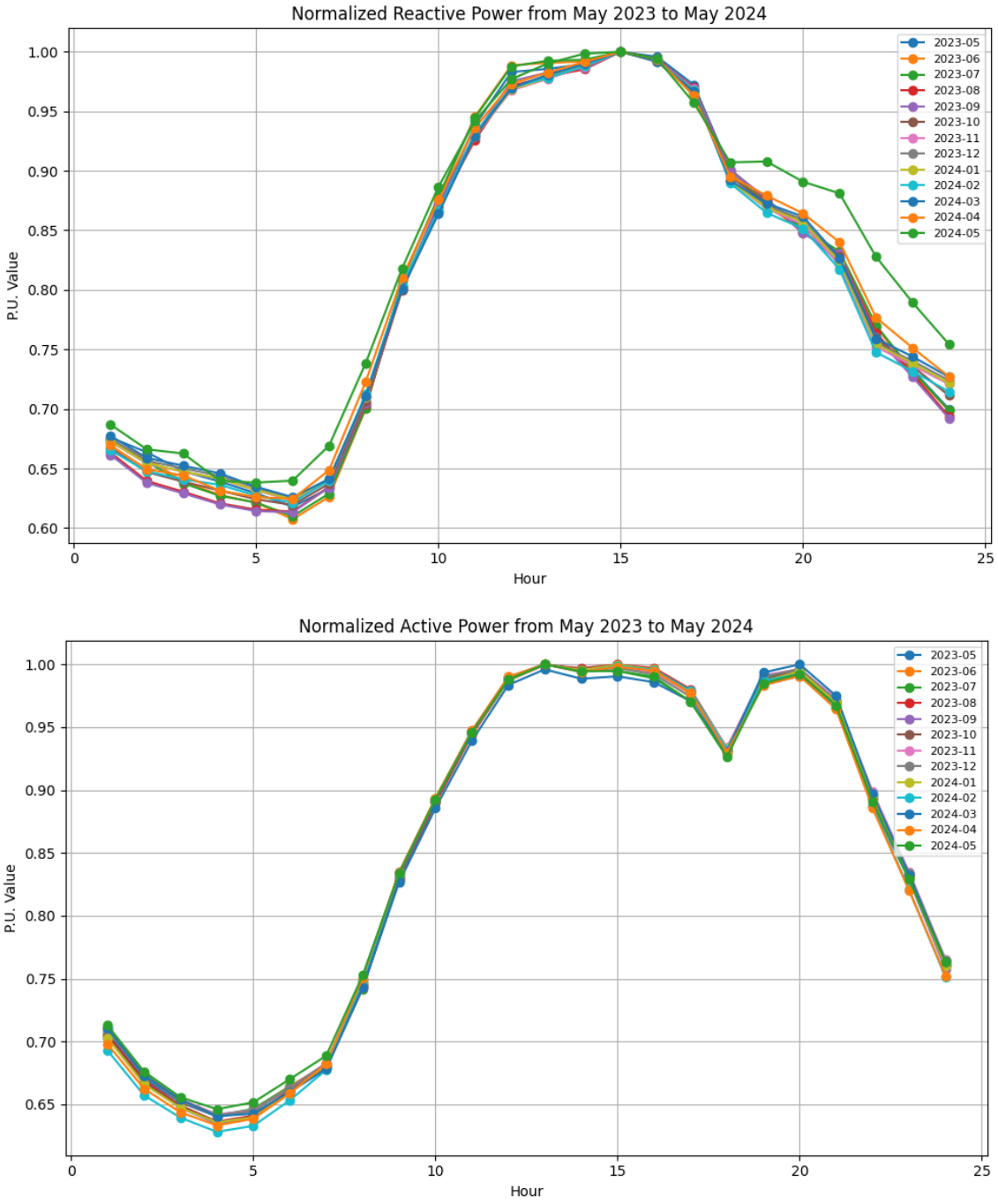

Figure 4. Normalized reactive and active power from May 2023 to May 2024 for Cali in 115kV nodes. Source: Self-made.

Figure 3 displays the reactive and active power outputs from May 2023 to May 2024 in Antioquia for 110kV nodes, for instance, a characteristic demand curve for the residential, commercial, and industrial sectors in Medellín was obtained using artificial neural networks and interpolation algorithms [3]. By observing and comparing peak demand times, similar patterns can be noted between the curves obtained with the script and those generated by the authors.

As a final output, Figure 4 present the normalized reactive and active power curves from May 2023 to May 2024 for 115 kV nodes in Cali. These results illustrate the flexibility and applicability of the script when using different filtering criteria.

3.1. Script Performance Evaluation

The performance analysis presented in Table 1 demonstrates that the script's execution time is mainly influenced by the amount of data processed, rather than the specific geographic filter applied. When analyzing 12 months of data, the execution time ranged from 25.66 seconds for Bogotá (115 kV) to 30.18 seconds for Antioquia (110 kV), reflecting a modest increase likely due to differences in file count or structure. In contrast, reducing the dataset to just 1 month significantly decreased the execution time to under 9 seconds for both cases—8.59 seconds for Bogotá and 8.53 seconds for Antioquia. These results indicate that the script scales efficiently with smaller datasets and maintains consistent performance across different regional filters, making it suitable for both large-scale and targeted analyses.

Keywords	Dataset Sizing	Execution Tine
Bogotá - 115kV	12 months	25.66 s
Antioquia - 110kV	12 months	30.18 s
Bogotá - 115 kV	1 month	8.59 s
Antioquia - 110 kV	1 month	8.53s

Table 1. Comparison of script execution times based on regional filters and dataset sizing.

3.2. Validation and Benchmarking

The validation and benchmarking of the results generated by this script are constrained by the absence of comparable studies or publicly accessible datasets that provide hourly normalized active and reactive power curves over the period from May 2023 to May 2024 for the specified voltage levels and geographic regions. Consequently, external benchmarking is currently not feasible.

Internal validation through statistical error metrics such as RMSE (Root Mean Square Error) has been incorporated to quantify the deviation of daily 24-hour load profiles from the monthly average curve. RMSE was computed by comparing each daily vector against the monthly average and then averaging the errors across all daily samples for a given month and power type (active or reactive). This metric gives an overall estimate of variability within each month and highlights how consistent daily operations are.

For example, Table 2 shows that the RMSE for reactive power in Bogotá (115 kV) ranged from 5.11 MVAr (January 2024) to 5.84 MVAr (April 2024), while the active power RMSE ranged from 15.19 MW to 17.38 MW across the same period. These values reflect operational diversity rather than errors in data processing, as they stem from aggregating naturally varying daily behavior into a representative monthly profile. It is important to emphasize that since the monthly average is already a smoothed representation, further comparison to individual samples may lead to misleading interpretations. The RMSE is not an accuracy measure of the method itself but a depiction of intra-month load variation.

3.3. Limitations

While the methodology presented offers a practical approach for analyzing energy consumption patterns across substations, it is important to recognize several limitations that may affect the interpretation and applicability of

Month	RMSE Reactive (MVAr)	RMSE Active (MW)
May 2023	5.23	15.95
June 2023	5.23	15.79
July 2023	5.21	15.60
August 2023	5.38	16.04
September 2023	5.42	16.16
October 2023	5.39	16.20
November 2023	5.39	16.07
December 2023	5.14	15.33
January 2024	5.11	15.19
February 2024	5.49	16.20
March 2024	5.73	17.00
April 2024	5.84	17.38
May 2024	5.14	15.53

Table 2. RMSE values per month for Bogotá (115 kV nodes).

the results. One key limitation lies in the temporal granularity of the data: the use of hourly averages may obscure significant intra-hour fluctuations in energy demand. Short-term spikes or drops—common in industrial zones or areas with high residential variability—can be smoothed out during averaging, potentially masking transient behaviors that are operationally relevant.

Additionally, the analysis does not account for external influencing factors such as temperature, humidity, or broader contextual variables like public holidays or atypical work-from-home periods. These elements can have a substantial impact on energy usage patterns, particularly in urban environments like Bogotá, where weather and human activity patterns are closely tied to power consumption. The absence of such variables limits the ability to attribute observed trends to specific causes and may oversimplify the underlying dynamics.

Finally, the study relies on historical textual data logs that may contain inconsistencies or biases introduced during data acquisition or formatting. Averaging across multiple files helps to mitigate the influence of individual anomalies, but cannot fully eliminate the risk of systemic bias or error. Furthermore, if this method were to be extended to forecasting tasks, the lack of uncertainty modeling and the exclusion of exogenous predictors could introduce significant forecast bias, reducing the reliability of predictive outcomes. These limitations suggest that while the presented approach is effective for broad pattern recognition and comparison, caution should be exercised when drawing detailed operational or predictive conclusions.

3.4. Future Work

Building on the current methodology, several avenues for future development could significantly enhance both the analytical capabilities and user accessibility of this work. One promising direction is the integration of machine learning models for real-time analysis and anomaly detection. By training on historical substation data, such models could identify consumption patterns, detect deviations, and even predict future load profiles with greater temporal precision, enabling more proactive decision-making in grid operations.

Another important enhancement involves establishing a direct API connection to the XM website or other authoritative sources to automate the acquisition of up-to-date power system data. This would reduce manual processing, improve data freshness, and facilitate continuous monitoring.

In terms of usability, developing a step-by-step Colab-based tutorial would be highly beneficial for non-technical users, such as planners or energy managers who may not have programming experience. Such a tutorial could guide users through data upload, parameter selection, and output interpretation using a simplified interface.

Complementing this, the creation of a graphical user interface (GUI) would make the tool significantly more accessible. A well-designed GUI could allow users to adjust parameters, such as filtering keywords, normalization type, or time intervals, without editing code directly, making the analysis pipeline more flexible and user-friendly for a wider audience.

Collectively, these improvements would not only increase the technical depth of the tool but also democratize its usage across stakeholders in the energy sector, fostering a more responsive and data-driven approach to power system management.

4. Conclusions

The Python script developed for analyzing XM's operational demand forecast data has provided valuable insights into Bogotá's power demand patterns. Utilizing Python's libraries and the Google Colaboratory environment, the methodology has effectively processed and visualized large datasets, uncovering significant trends in both active and reactive power. This robust approach enables the transformation of raw data into actionable insights, crucial for effective energy management. The script's capabilities in handling, filtering, and normalizing data highlight its effectiveness in managing extensive datasets. It processes multiple text files, ensuring that all relevant data is accurately captured and utilized, which is essential for maintaining data integrity and comprehensive analysis. Filtering isolates pertinent information, enhancing the accuracy of the results and streamlining the analysis process.

The analysis confirms significant demand peaks during well-established high-demand periods, such as between 11 am to 1 pm and 6 pm to 9 pm. These findings validate the accuracy of the forecast data and support the need for targeted energy management strategies. Recognizing these peak periods helps align forecasts with actual consumption patterns, improving the preparation for high-demand scenarios.

Normalized data provides a clearer understanding of demand variations, crucial for developing precise energy management strategies. By removing biases and standardizing data, normalization reveals trends and fluctuations that raw data might obscure. This enhanced clarity allows energy managers to identify periods of high and low demand with greater precision, facilitating more responsive management strategies.

Understanding demand patterns and variations supports strategic planning and resource management. The insights derived enable proactive measures, such as infrastructure upgrades and demand response programs, which enhance grid stability and efficiency. This understanding also contributes to long-term sustainability and operational excellence in energy management. Effective resource planning relies on the accurate analysis provided by the script. Continuous updates and integration of data into planning efforts ensure efficient resource use and better alignment with actual demand. This leads to improved overall system performance, optimizing resource deployment and grid management.

In summary, the methodology outlined demonstrates a robust approach to analyzing power demand forecasts, offering practical solutions for energy management in Bogotá. The insights gained enhance the operational efficiency of the SIN, ensuring a reliable power supply and supporting effective energy management practices.

Acknowledgement

The authors acknowledge the support provided by Thematic Network 723RT0150 "Red para la integración a gran escala de energías renovables en sistemas eléctricos (RIBIERSE-CYTED)" funded by the 2022 Call for Thematic Networks by the Ibero-American Program of Science and Technology for Development (CYTED in Spanish).

Annex I: Monthly normalized active and reactive curves from May 2023 to May 2024.

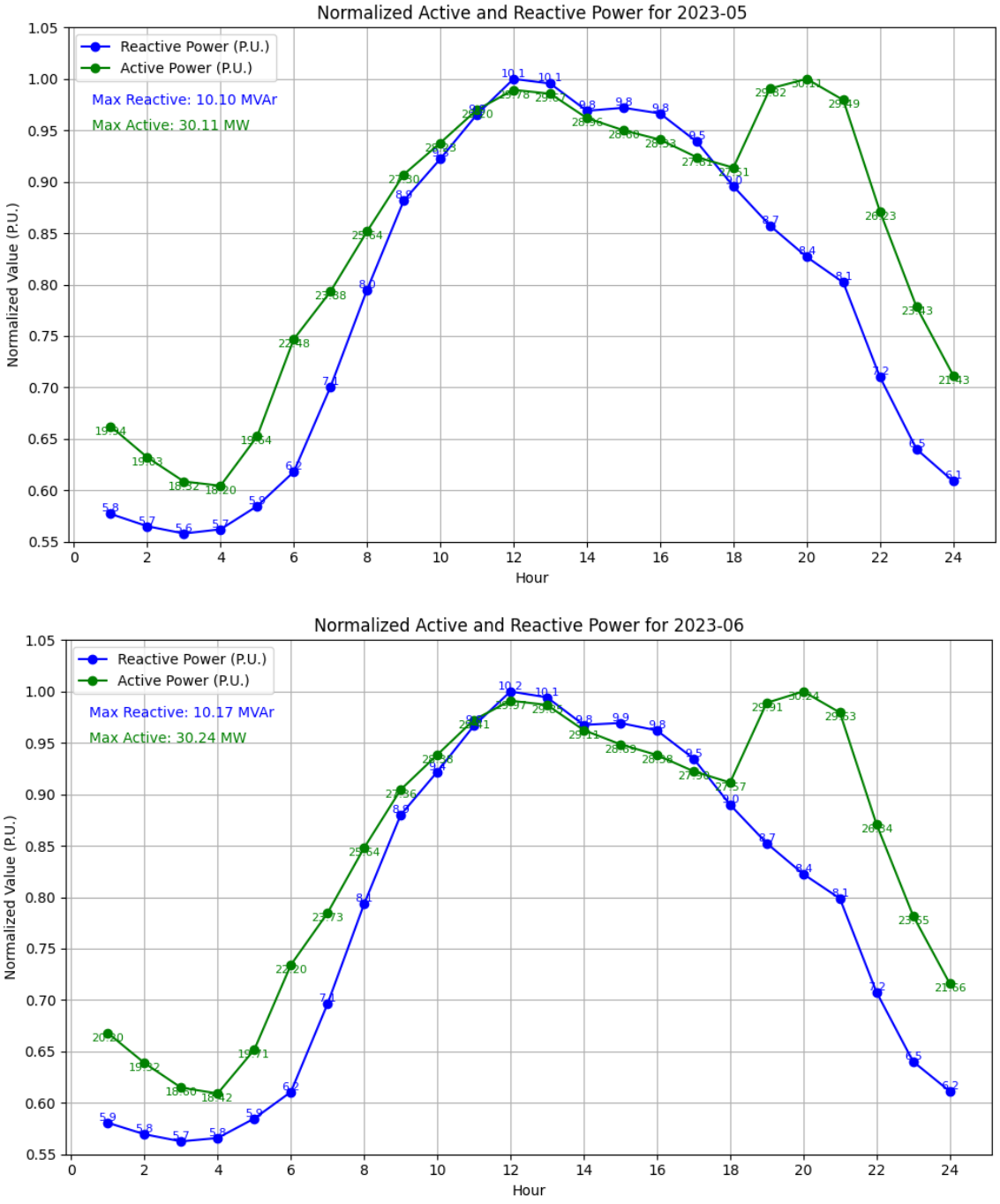

Figure 5. Normalized active and reactive power for May and June 2023. Source: Self-made.

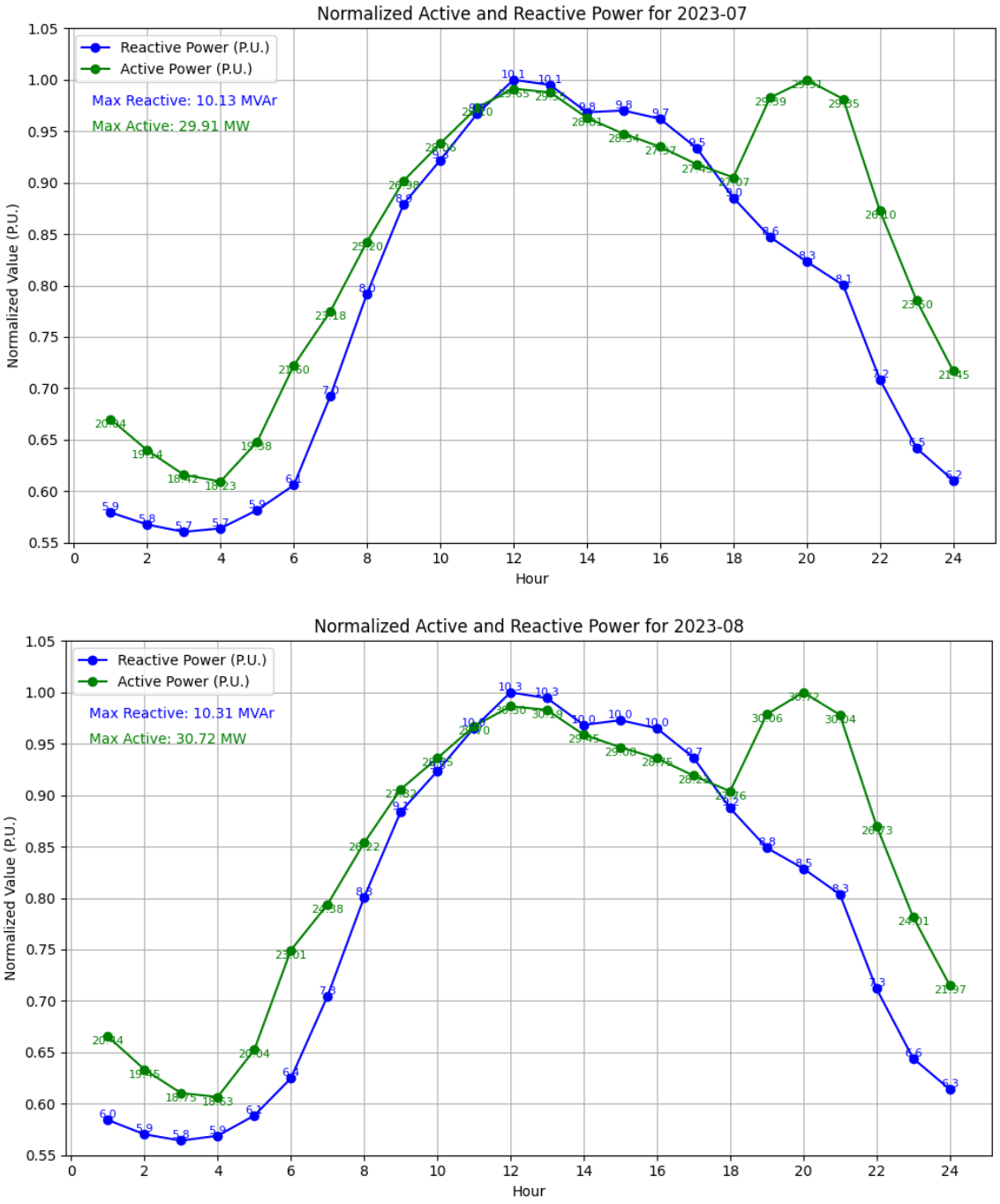

Figure 6. Normalized active and reactive power for July and August 2023. Source: Self-made.

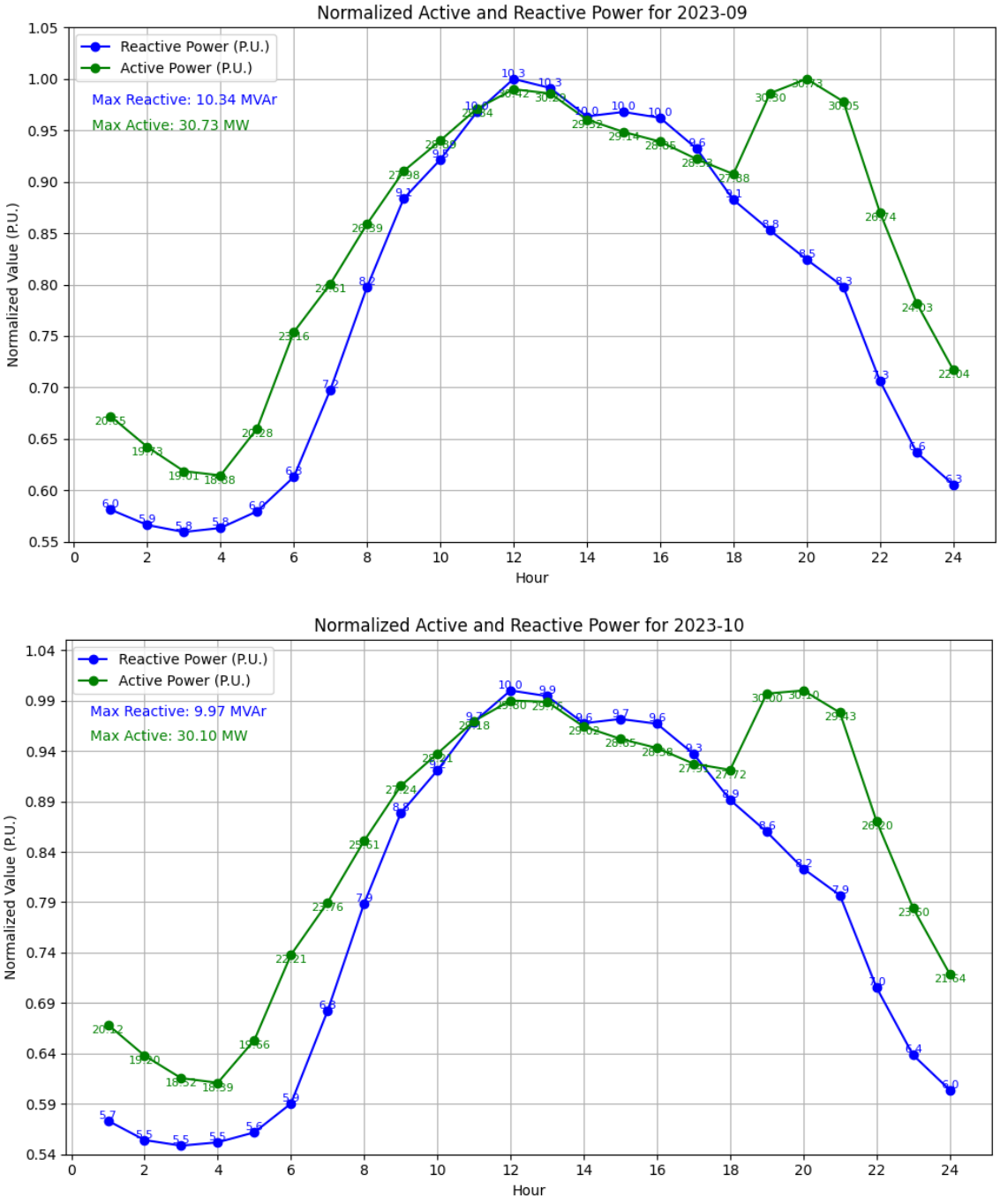

Figure 7. Normalized active and reactive power for September and October 2023. Source: Self-made.

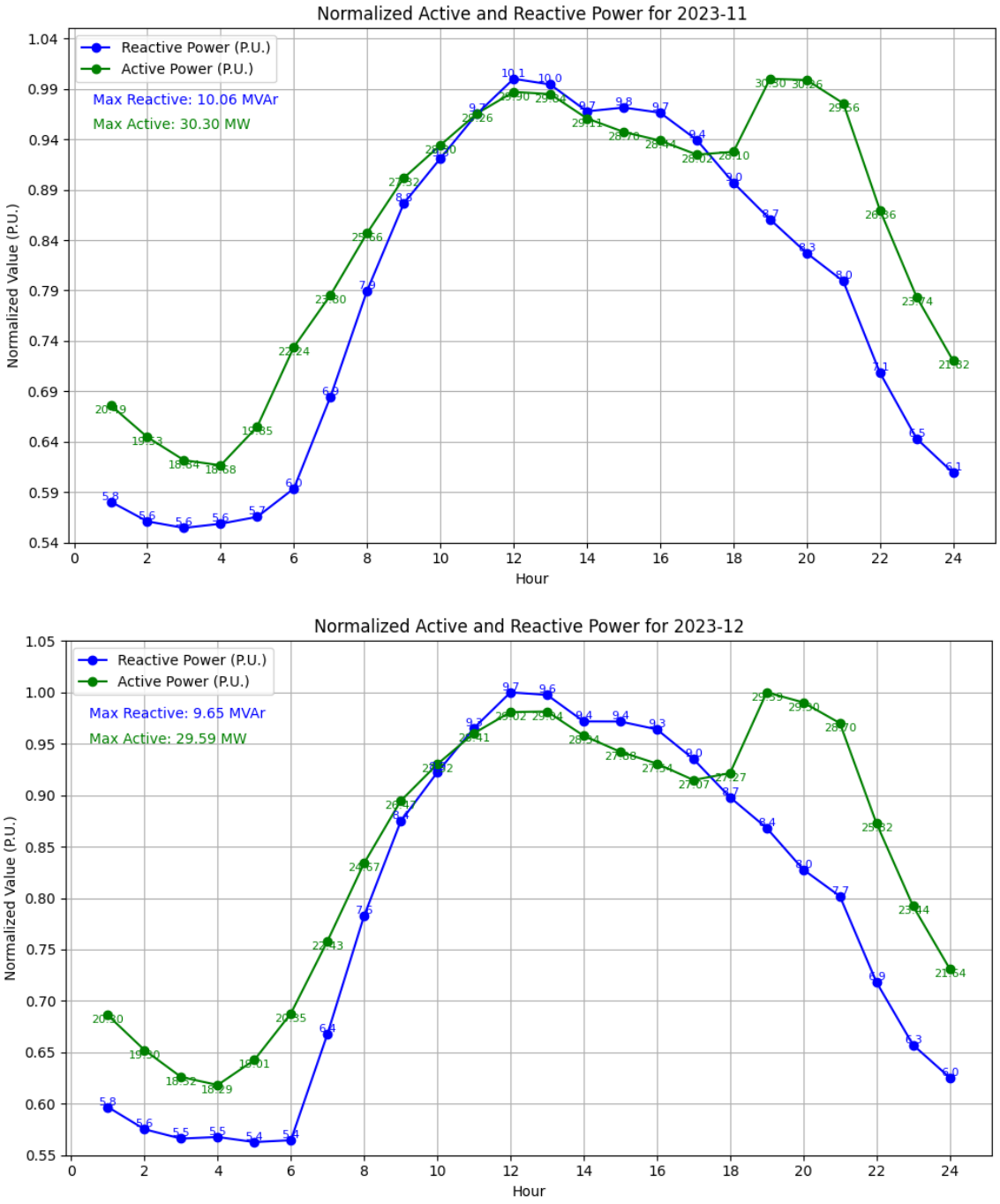

Figure 8. Normalized active and reactive power for November and December 2023. Source: Self-made.

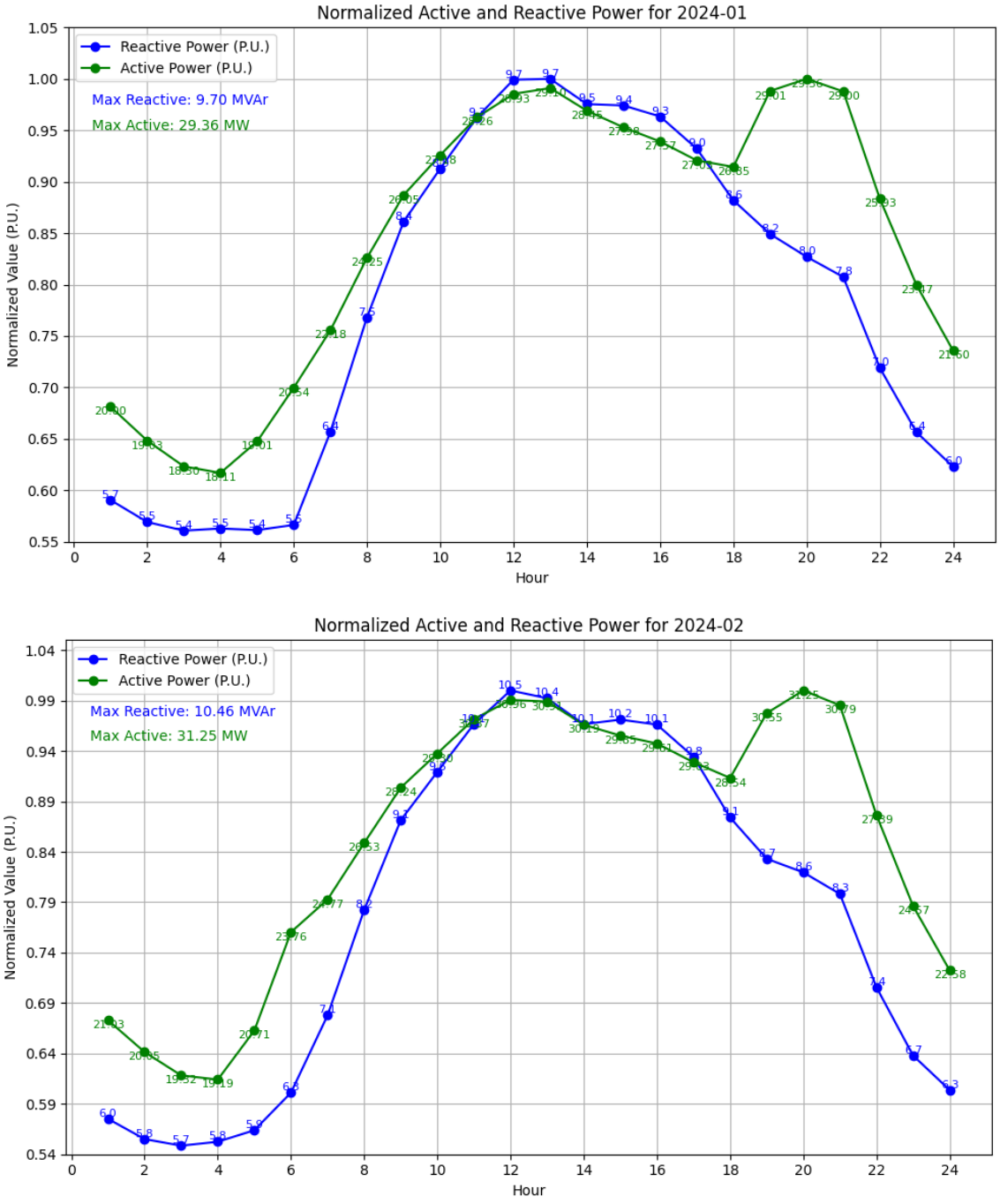

Figure 9. Normalized active and reactive power for January and February 2024. Source: Self-made.

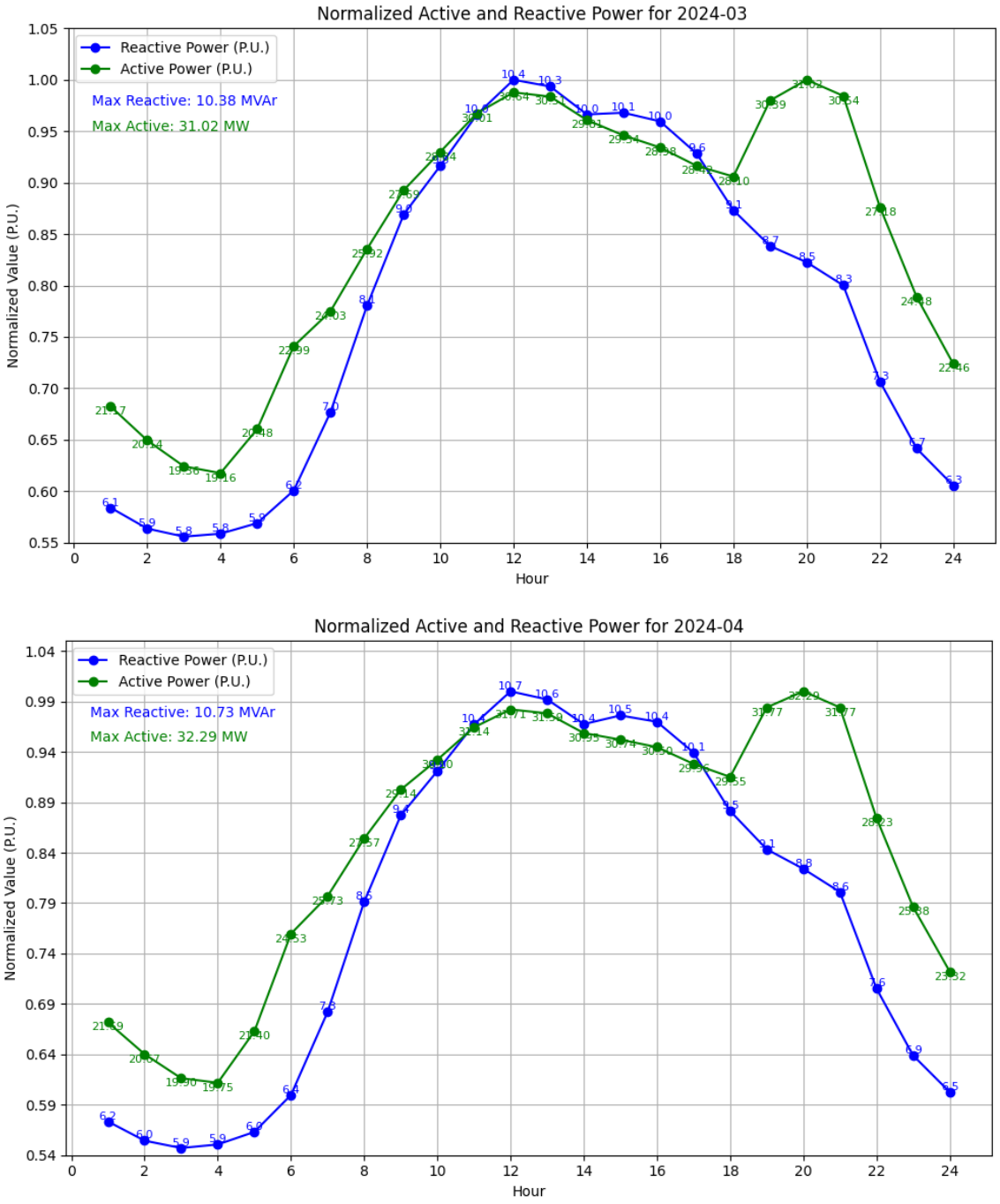

Figure 10. Normalized active and reactive power for March and April 2024. Source: Self-made.

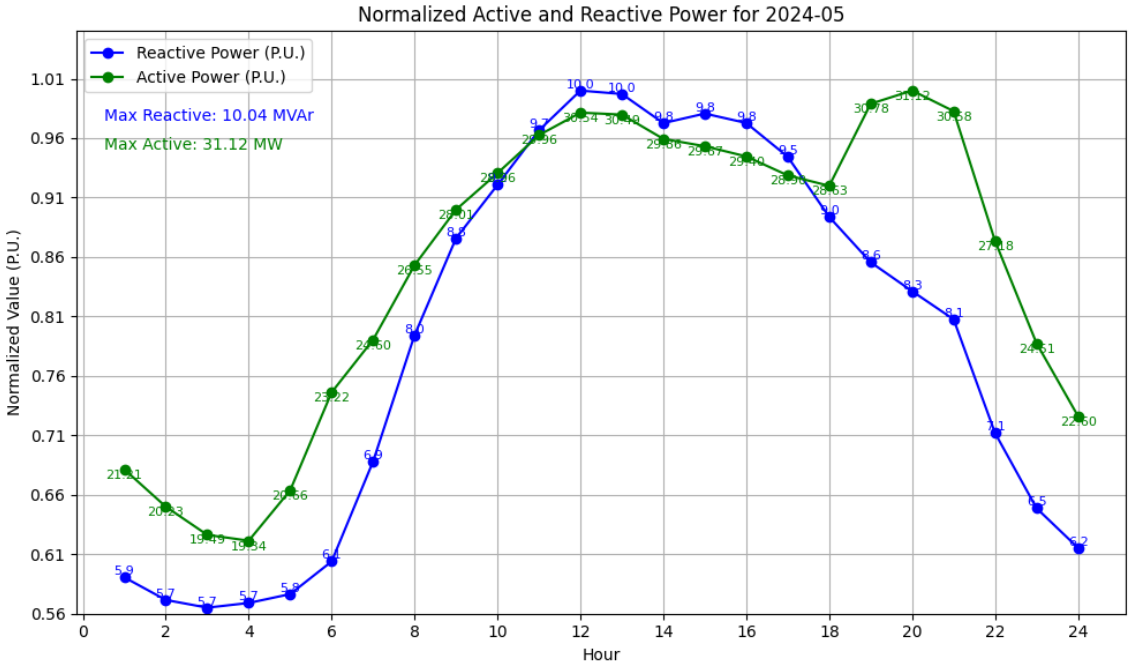

Figure 11. Normalized active and reactive power for May 2024. Source: Self-made.

Annex II: List of available keywords and voltage levels to filter using the python script.

UPCs names

- andaki
- antioquia
- barranquilla
- bputumayo
- cali
- cartagena
- cartago
- cedenar
- cens
- cerromatoso
- cesar
- chec
- choco
- cirainf
- codensa
- drummond
- drummondLoma
- ebsa
- emec
- emsa
- enelar
- enerca
- guajiram
- guaviare
- intercor
- oxyint
- pacande
- pacifico
- pereira
- pijaos
- planeta
- putumayo
- quindio
- rubiales
- Sanfernando
- santander
- sinu
- sur
- Ternium
- tubocaribe
- tulua

Voltage levels

- 115
- 110
- 44
- 220
- 34
- 14
- 66
- 230
- 58
- 500
- 13
- 33

REFERENCES

1. Biswajit Biswal, Subhasish Deb, Subir Datta, Taha Selim Ustun, and Umit Cali. Review on smart grid load forecasting for smart energy management using machine learning and deep learning techniques. *Energy Reports*, 12:3654–3670, dec 2024. doi:10.1016/j.egyr.2024.09.056.
2. Google. Google colaboratory, n.d. Accessed: June 27, 2025. URL: https://cloud.google.com/colab/docs.
3. Miriam Gómez Marín, Henry O. Sarmiento-Maldonado, Alba Nelly Ardila Arias, William Alonso Giraldo Aristizábal, and Rubén Darío Vásquez-Salazar. Using Artificial Intelligence Tools to Analyze Particulate Matter Data (PM2.5). *Atmosphere*, 16(6):635, May 2025. doi:10.3390/atmos16060635.
4. Matplotlib Development Team. Matplotlib: Visualization with python, n.d. Accessed: June 27, 2025. URL: https://matplotlib.org/.
5. Abraham Hizkiel Nebey. Recent advancement in demand side energy management system for optimal energy utilization. *Energy Reports*, 11:5422–5435, jun 2024. doi:10.1016/j.egyr.2024.05.028.
6. Carlos Peña-Guzmán and Juliana Rey. Forecasting residential electric power consumption for bogotá colombia using regression models. *Energy Reports*, 6:561–566, February 2020. doi:10.1016/j.egyr.2019.09.026.
7. Python Package Index. gdown — python package index, n.d. Accessed: June 27, 2025. URL: https://pypi.org/project/gdown/.
8. Python Software Foundation. Python language reference, n.d. Accessed: June 27, 2025. URL: https://www.python.org/doc/.
9. Python Software Foundation. `os` — miscellaneous operating system interfaces, n.d. Accessed: June 27, 2025. URL: https://docs.python.org/3/library/os.html.
10. Python Software Foundation. `zipfile` — work with zip archives, n.d. Accessed: June 27, 2025. URL: https://docs.python.org/3/library/zipfile.html.
11. Sinergox - Plataforma de Gestión Energética. Demand and boundaries [demandas y fronteras], n.d. Accessed: June 27, 2025. URL: https://sinergox.xm.com.co/Paginas/Home.aspx.
12. Dr. Sarah Thompson. The Impact of Demand Forecasting Accuracy on Customer Satisfaction. *International Journal of Supply Chain Management*, 9(5):55–66, December 2024. doi:10.47604/ijscm.3117.
13. Prosper O. Ugbehe, Ogheneruona E. Diemuodeke, and Daniel O. Aikhuele. Electricity demand forecasting methodologies and applications: a review. *Sustainable Energy Research*, 12(1), April 2025. doi:10.1186/s40807-025-00149-z.
14. Natalia Verde, Petros Patias, and Giorgos Mallinis. A Cloud-Based Mapping Approach Using Deep Learning and Very-High Spatial Resolution Earth Observation Data to Facilitate the SDG 11.7.1 Indicator Computation. *Remote Sensing*, 14(4):1011, February 2022. doi:10.3390/rs14041011.
15. XM - Operador del Sistema Interconectado Nacional. New metrics in the xm api [nuevas métricas en la api xm], n.d. Accessed: June 27, 2025. URL: https://github.com/EquipoAnaliticaXM/API_XM?tab=readme-ov-file#section2.
16. XM - Operator of the National Interconnected System. Official demand forecast (final) [pronóstico oficial de demanda (definitivo)], 2025. Accessed: June 27, 2025. URL: https://www.xm.com.co/consumo/informes-demanda/pronostico-oficial-de-demanda-definitivo.
17. XM - Operator of the National Interconnected System. Xm official website [sitio web oficial de xm], n.d. Accessed: June 27, 2025. URL: https://www.xm.com.co/.