# Differential Privacy and Encryption for Secure Data Analysis

By Stephen Sam

## 1. Introduction

This project presents a structured approach to examining the smart metering data collected in Tetouan, Morocco. The primary objective is to delve into comprehending and enhancing energy consumption trends, all the while ensuring robust safeguards for consumer privacy. Recognizing the delicate nature of energy consumption data, which holds the potential to unveil intimate details about individuals, we advocate for the utilization of a dual strategy incorporating cryptographic protocols and data anonymization methodologies. By employing these techniques, we aim to thwart unauthorized access and the reduction of personal information. Ultimately, this endeavor seeks to furnish practical insights for bolstering energy efficiency measures while steadfastly upholding the privacy rights of the residents.
The application is crafted to safeguard individuals' privacy within a dataset by introducing random noise to the data. This protective measure serves to hinder the identification of individuals within the dataset while preserving the capacity for statistical analysis. Utilizing PyQt5 as the GUI framework, the application seamlessly integrates with the IBM Diffprivlib library, leveraging its suite of differential privacy mechanisms.

## 2. Dataset

The dataset comprises 52,416 entries documenting energy usage in Tetouan, Morocco, with readings recorded every 10 minutes across three different zones. It includes environmental parameters such as temperature, humidity, and wind speed, alongside energy consumption figures, making it a comprehensive resource for investigating the influence of various factors on energy usage trends.
Key features of the dataset include:

- **Date Time**: Each entry is timestamped, allowing for precise time series analysis at 10-minute intervals.
- **Temperature**: Ambient temperature data is provided, offering insights into how weather conditions impact energy consumption.
- **Humidity**: Recorded humidity levels accompany temperature readings, aiding in understanding how climate affects energy usage patterns.
- **Wind Speed**: Measurements of wind speed are included, which can be relevant in scenarios involving heating or cooling processes.

- **General Diffuse Flows & Diffuse Flows**: These features likely pertain to specific environmental or energy flow characteristics specific to the region, providing additional variables for analysis.
- **Zone 1, Zone 2, and Zone 3 Power Consumption**: Detailed energy consumption data is provided for each of the three zones in Tetouan, facilitating zone-specific analysis and comparisons.

# 3. Privacy and Security Techniques

Performing differential privacy on columns selected by user based on:

a. Laplace
b. Gaussian

And implementing Partially Homomorphic Encryption
To implement differential privacy on the columns we selected, we have utilized the Laplace and Gaussian mechanisms. Additionally, we employed Homomorphic Encryption for enhanced privacy protection.

**1. Laplace Mechanism:**

- We calculated the sensitivity of each selected column, representing the impact of individual data on the output.
- Adding Laplace noise to the data in each column ensured differential privacy.
- We determined the scale of Laplace noise based on column sensitivity and desired privacy level, allocating privacy budget accordingly.

**2. Gaussian Mechanism:**

- Like Laplace, we calculated column sensitivity and added Gaussian noise for privacy.
- Mean and standard deviation of Gaussian noise were determined based on sensitivity and desired privacy level.
- Privacy budget was distributed across selected columns as needed.

**3.Homomorphic Encryption:**

- Using Homomorphic Encryption, we performed computations on encrypted data without decryption.
- Encryption of selected columns was done using a suitable scheme.
- After necessary computations (e.g., aggregation, analysis), decryption occurred, ensuring data remained confidential throughout.
- Selection of the encryption scheme depended on computational requirements and desired security level.

- By implementing these approaches, we ensured that the selected columns-maintained privacy while still allowing for meaningful analysis and computation, managing privacy budget and considering trade-offs between privacy and utility.

# 4. Libraries used

In our project, we've incorporated two essential libraries for performing differential privacy and homomorphic encryption: `diffprivlib` for implementing differential privacy mechanisms and `tenseal` for homomorphic encryption.

1. **diffprivlib**:

- This library provides implementations of various differential privacy mechanisms, including Gaussian and Laplace mechanisms.
- Gaussian and Laplace mechanisms add noise to data to preserve privacy while allowing meaningful analysis.
- We can use these mechanisms to ensure that our data analysis tasks maintain privacy guarantees, especially when dealing with sensitive information.
- By importing and utilizing these mechanisms from `diffprivlib`, we can easily incorporate differential privacy into our project workflow.

2**. tenseal:**

- `tenseal` is a library for performing homomorphic encryption, enabling computations on encrypted data without decryption.
- Homomorphic encryption is crucial for preserving the confidentiality of sensitive data while still allowing for meaningful computations.
- With `tenseal`, we can encrypt our data before performing computations, ensuring that the original data remains confidential throughout the analysis process.
- This library enables us to explore advanced privacy-preserving techniques and securely perform computations on sensitive data.

Learning about and incorporating these libraries into our project helped us in understanding and implementing privacy-preserving techniques in real-world scenarios. By practically implementing these libraries, we gain hands-on experience with differential privacy and homomorphic encryption, deepening our understanding of their principles and applications. This approach not only enhances our project's security and privacy measures but also enriches our knowledge and skills in privacy-preserving data analysis.

# 5. Code Snippets

To study the use of Differential Privacy and Partially Homomorphic Encryption on Tetouan's power consumption dataset, we decided to build a graphical user interface in Python. The goal of this GUI is not to only allow analysis and anonymization of data for this dataset, but also for any dataset that an individual might want to study anonymization and encryption techniques on. We created two separate GUI's in this case to achieve this outcome. One for Differential Privacy and another for Homomorphic encryption. The idea was to combine these two programs into a single program, however, there were issues with package dependencies in Python which led us to develop it as two separate GUI's. We will go over this in future works as well.

## Differential Privacy Implementation

This application is designed to protect the privacy of individuals in a dataset by adding random noise to the data. This mechanism helps prevent the identification of individuals from the dataset while allowing for statistical analysis. The application utilizes PyQt5 for the GUI framework and integrates differential privacy mechanisms from the IBM Diffprivlib library.

```python
def prepare_dp_controls(self):
    try:
        # Add labels for clarity
        label_layout = QHBoxLayout()
        label_layout.addWidget(QLabel('Column'))
        label_layout.addWidget(QLabel('Mechanism'))
        label_layout.addWidget(QLabel('Epsilon'))
        label_layout.addWidget(QLabel('Sensitivity'))
        self.layout.addLayout(label_layout)

        for widget in self.dp_controls:
            widget.deleteLater()
        self.dp_controls.clear()

        for i, column in enumerate(self.data.columns):
            hbox = QHBoxLayout()
            checkbox = QCheckBox(f"{column}")
            hbox.addWidget(checkbox)

            combo_box = QComboBox()
            combo_box.addItems(["Laplace", "Gaussian"])
            hbox.addWidget(combo_box)

            epsilon_spinbox = QDoubleSpinBox()
            epsilon_spinbox.setRange(0.01, 10.0)
            epsilon_spinbox.setSingleStep(0.1)
            epsilon_spinbox.setValue(0.5)
            hbox.addWidget(epsilon_spinbox)

            sensitivity_spinbox = QDoubleSpinBox()
            sensitivity_spinbox.setRange(0.1, 10.0)
            sensitivity_spinbox.setSingleStep(0.1)
            sensitivity_spinbox.setValue(1.0)
            hbox.addWidget(sensitivity_spinbox)

            self.layout.addLayout(hbox)
            self.dp_controls.append((checkbox, combo_box, epsilon_spinbox, sensitivity_spinbox))
    except Exception as e:
        QMessageBox.critical(self, 'Error', 'Failed to prepare differential privacy controls: ' + str(e))
```

The GUI includes dynamic controls that allow the user to select and configure the differential privacy mechanisms for each column in the dataset. These controls are generated based on the columns present in the loaded dataset. The key components here are:

- Checkbox: Allows the selection of columns to apply differential privacy.
- Combo Box: Provides a choice between Laplace and Gaussian mechanisms.
- Epsilon Spinbox: Allows setting the epsilon value, which controls the level of privacy (lower values mean more privacy).
- Sensitivity Spinbox: Allows setting the sensitivity, which impacts the amount of noise based on the data's variability.

```python
def apply_dp(self):
    try:
        self.anonymized_data = self.data.copy()
        for i, controls in enumerate(self.dp_controls):
            if controls[0].isChecked():
                epsilon = controls[2].value()
                sensitivity = controls[3].value()
                mechanism_type = controls[1].currentText()
                column = self.data.columns[i]

                if mechanism_type == "Laplace":
                    mechanism = Laplace(epsilon=epsilon, sensitivity=sensitivity)
                elif mechanism_type == "Gaussian":
                    mechanism = Gaussian(epsilon=epsilon, sensitivity=sensitivity, delta=1e-5)

                self.anonymized_data[column] = [mechanism.randomise(val) for val in self.data[column]]

        self.display_data(self.anonymized_data)
        QMessageBox.information(self, 'Info', 'Differential privacy applied successfully!')
    except Exception as e:
        QMessageBox.critical(self, 'Error', 'Failed to apply differential privacy: ' + str(e))
```

Upon activating the 'Apply Differential Privacy' button, the application processes each selected column using the specified differential privacy mechanism and parameters. The important mechanisms here include, but are not limited to:

- Laplace Mechanism: Adds noise drawn from a Laplace distribution.
- Gaussian Mechanism: Adds noise drawn from a Gaussian distribution, requiring an additional delta parameter, which provides a measure of the probability of additional privacy loss.

```python
def plot_data_comparison(self):
    if not hasattr(self, 'anonymized_data'):
        QMessageBox.warning(self, 'Error', 'Apply differential privacy before plotting.')
        return

    # Clear existing plots to avoid overlay of graphs on re-plotting
    for i in reversed(range(self.scroll_layout.count())):
        widgetToRemove = self.scroll_layout.itemAt(i).widget()
        if widgetToRemove is not None:
            widgetToRemove.setParent(None)
            widgetToRemove.deleteLater()

    # Plot new data comparisons
    for i, column in enumerate(self.data.columns):
        controls = self.dp_controls[i]
        checkbox = controls[0]
        if checkbox.isChecked():
            fig = Figure(figsize=(10, 8))
            canvas = FigureCanvas(fig)
            axs = fig.subplots(2, 2)

            # Histogram with title including column name
            axs[0, 0].hist(self.data[column], bins=20, alpha=0.5, label='Original')
            axs[0, 0].hist(self.anonymized_data[column], bins=20, alpha=0.5, label='Anonymized')
            axs[0, 0].set_title(f'Histogram of {column}')
            axs[0, 0].legend()

            # Box Plot with title including column name
            axs[0, 1].boxplot([self.data[column], self.anonymized_data[column]], labels=['Original', 'Anonymized'])
            axs[0, 1].set_title(f'Box Plot of {column}')

            # Density Plot with title including column name
            sns.kdeplot(self.data[column], ax=axs[1, 0], fill=True, label='Original', alpha=0.5)
            sns.kdeplot(self.anonymized_data[column], ax=axs[1, 0], fill=True, label='Anonymized', alpha=0.5)
            axs[1, 0].set_title(f'Density Plot of {column}')
            axs[1, 0].legend()

            # Line Plot with title including column name
            axs[1, 1].plot(self.data[column], label='Original')
            axs[1, 1].plot(self.anonymized_data[column], label='Anonymized')
            axs[1, 1].set_title(f'Line Plot of {column}')
            axs[1, 1].legend()

            self.scroll_layout.addWidget(canvas)

            # Add metrics text area
            metrics_text = QTextEdit()
            metrics_text.setPlainText(
                f"Original Mean: {np.mean(self.data[column]):.2f}, Anonymized Mean: {np.mean(self.anonymized_data[column]):.2f}\n"
                f"Original Std Dev: {np.std(self.data[column]):.2f}, Anonymized Std Dev: {np.std(self.anonymized_data[column]):.2f}"
            )
            metrics_text.setReadOnly(True)
            self.scroll_layout.addWidget(metrics_text)
```

We created a plot_Data_comparison() function here to plot the data we have received from the newly updated (from DP) columns only. This can be easily extended upon to add more graphs as currently it only generates 4 types of graphs to compare the original and modified data, and metrics to show the mean and standard deviation.

Regarding error handling too, the application includes managing and reporting issues that occur during the application of differential privacy, ensuring the application does not crash and provides feedback on errors encountered.

## Partially Homomorphic Encryption Implementation

This GUI Configures the main window, setting its title and dimensions, and initializes a layout with multiple buttons for various operations. It creates a cohesive user interface for interacting with encryption functionalities.

```
def setup_context(self):
    self.context = ts.context(
        ts.SCHEME_TYPE.CKKS,
        poly_modulus_degree=4096,
        coeff_mod_bit_sizes=[30, 30, 30]
    )
    self.context.global_scale = 2**30
    self.context.generate_galois_keys()
```

Regarding the Encryption Context setup, Initializes the homomorphic encryption settings using the CKKS scheme, which supports approximate arithmetic on encrypted floating-point numbers. It also sets up key parameters like the polynomial modulus degree and coefficient modulus sizes, essential for defining the security and efficiency of encryption operations. We intentionally kept it to smaller values to reduce computational costs and time taken to encrypt/decrypt.

```
def async_encrypt_data(self):
    threading.Thread(target=self.encrypt_data).start()

def encrypt_data(self):
    temperatures = self.data['Temperature'].tolist()
    self.encrypted_vector = ts.ckks_vector(self.context, temperatures)
    self.data['Temperature'] = ['Encrypted value' for _ in temperatures]  # Placeholder text for encrypted data
    self.display_data(self.data)
    self.show_status("Data encrypted!")

def modify_encrypted_data(self):
    if self.e
        self.
        self.
        self.display_data(self.data)
        self.show_status("Encrypted data modified!")
    else:
        self.show_status("Encrypt data first!")

def decrypt_data(self):
    if self.encrypted_vector:
        decrypted_temperatures = self.encrypted_vector.decrypt()
        self.data['Temperature'] = decrypted_temperatures
        self.display_data(self.data)
        self.show_status("Data decrypted!")
    else:
        self.show_status("Encrypt and modify data first!")
```

(method) def display_data(data: Any) -> None

display_data

ted vector

e' for _ in self.data['Temperature']]  # Placeholder text for modified data

- async_encrypt_data(self)

This method uses the threading library to run the encrypt_data method in a separate thread. This is done to prevent the main GUI thread from becoming unresponsive while the encryption process, which can be computationally intensive, is running.

- encrypt_data(self)

It retrieves the temperatures from the 'Temperature' column of the data (assuming it's a pandas DataFrame) and converts them into a list.

It then encrypts these temperature values using TenSEAL's ts.ckks_vector method with the previously configured context. The context includes parameters suitable for encrypting floating-point numbers.

It replaces the original temperature values in the DataFrame with a placeholder text 'Encrypted value' for each entry. This is to visually indicate that the values have been encrypted.

The GUI is updated to display this new data, and a status message "Data encrypted!" is printed or displayed in the application.

- modify_encrypted_data(self)

Checks if there's an existing encrypted vector (self.encrypted_vector). If it exists, the method adds 5 to each element of this encrypted vector. This showcases the ability of partially homomorphic encryption to perform computations on ciphertexts directly.

After modification, it updates the GUI to display a placeholder 'Modified encrypted value' for each temperature to indicate that the encrypted data has been altered.

A status message "Encrypted data modified!" is then displayed or printed.

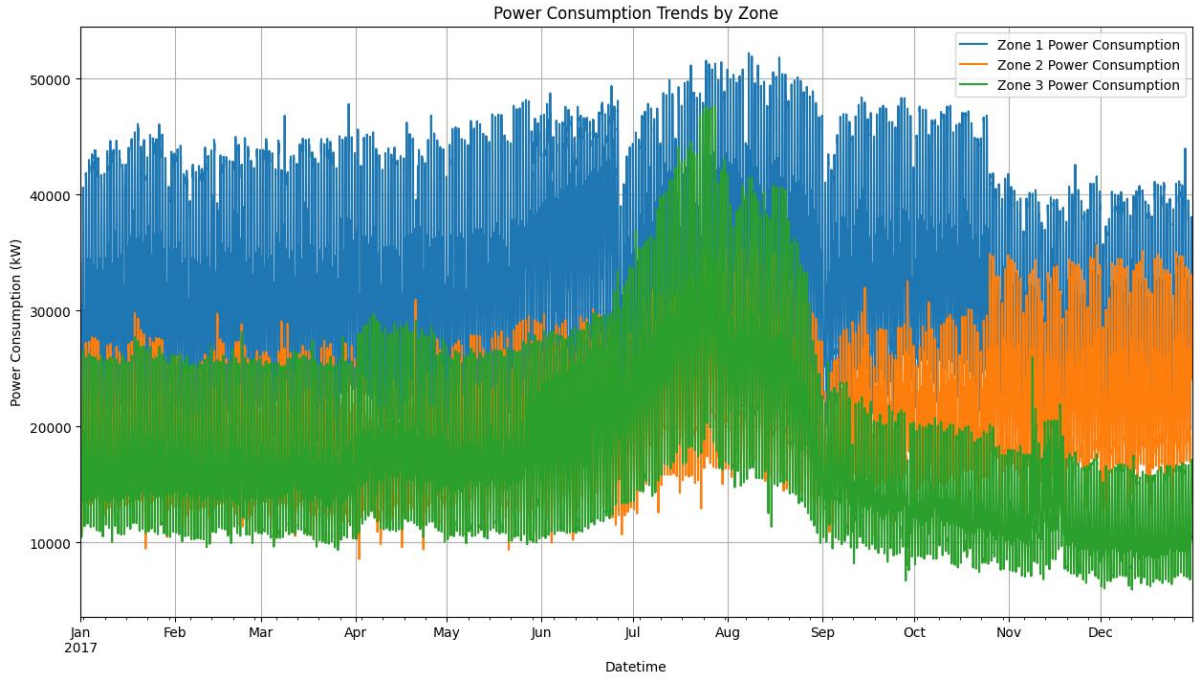
- decrypt_data(self)

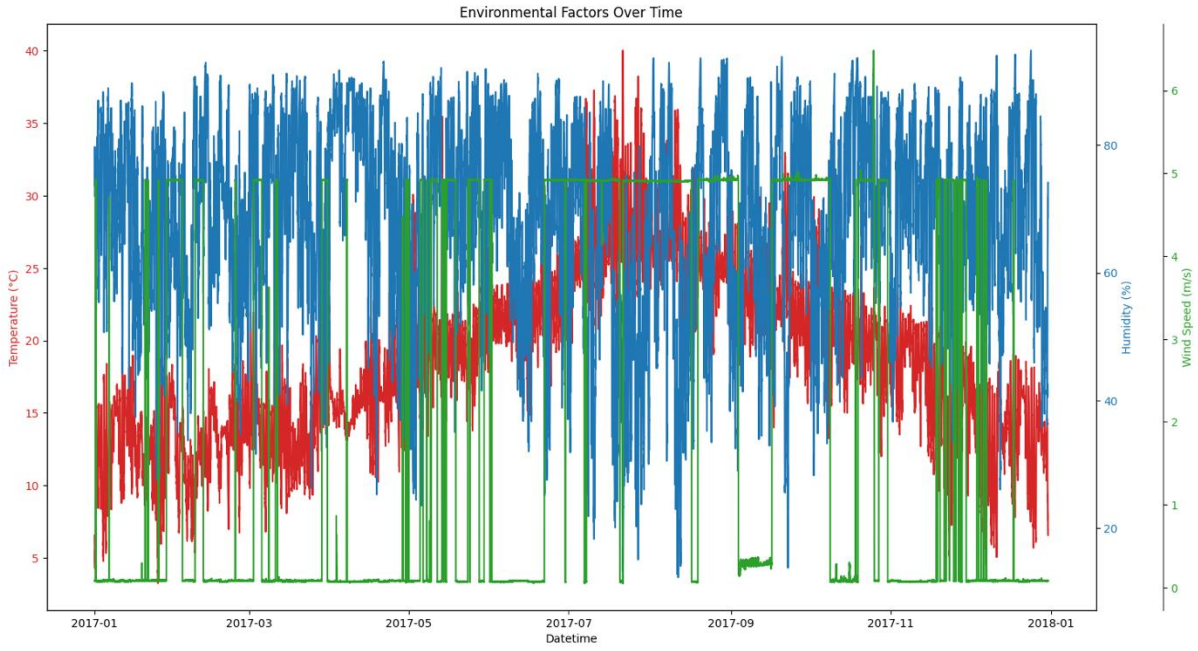
This method checks if there is an encrypted vector available. If so, it decrypts this vector using the decrypt () method of the encrypted vector.

The decrypted temperatures are then stored back in the DataFrame, replacing the encrypted or modified placeholders.

The GUI is refreshed to show the decrypted temperatures, and a status message "Data decrypted!" is shown or printed.

## 6. Different Trends




When exploring the trends in the data we can see:

- <u>Seasonal and Diurnal Patterns:</u>

Given the parameters like temperature and solar radiation, one could analyze seasonal and diurnal patterns in power consumption. For example, higher temperatures might correlate with increased use of cooling systems, thus higher power consumption.

- Weather Impact on Power Usage:

Changes in weather conditions (like increased cloudiness reflected in lower solar radiation values) could influence energy consumption, particularly in areas relying on solar power.

Potential attack Vectors when looking from a semi-honest adversary model from this dataset include:

- Data Snooping:

An attacker could infer usage patterns, peak load times, or even specific behaviors of zones based on power consumption data.

- Temporal Correlations:

By analyzing timestamps and power consumption, an attacker might deduce operational hours and activity levels, potentially identifying sensitive information about industrial or security practices.

The role of anonymization or encryption in this case would involve:

- Anonymization:

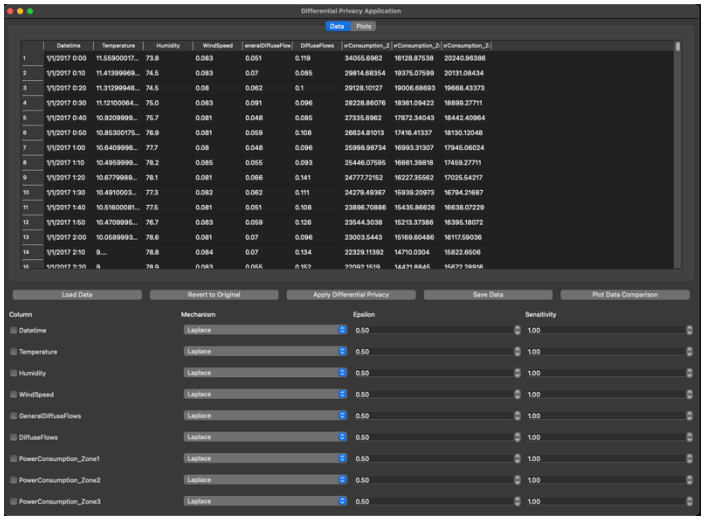
  Removing or disguising identifiers like zone specifics, exact timestamps, or any direct links to specific real-world locations or entities can reduce the risk of sensitive information leakage.

- Encryption:

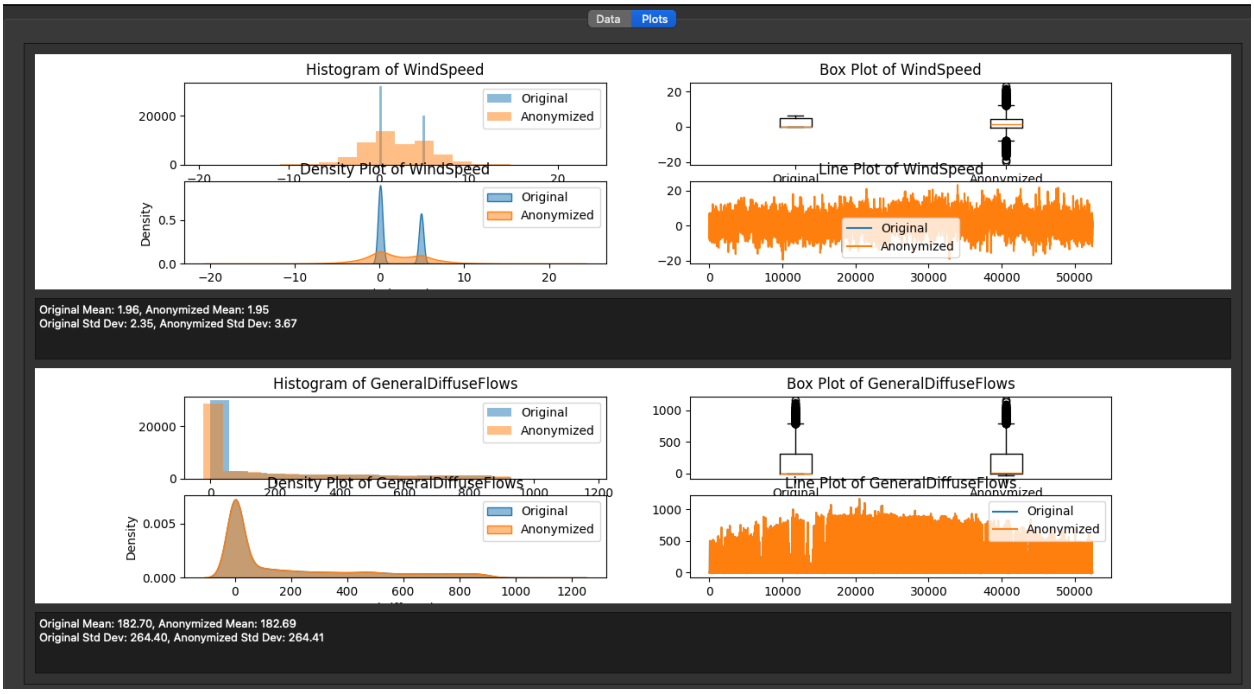
  Using encryption methods to secure the data ensures that even if data interception occurs, the information remains unintelligible without the proper decryption keys. This is particularly useful for protecting data in transit or stored in cloud environments.

# 7. GUI

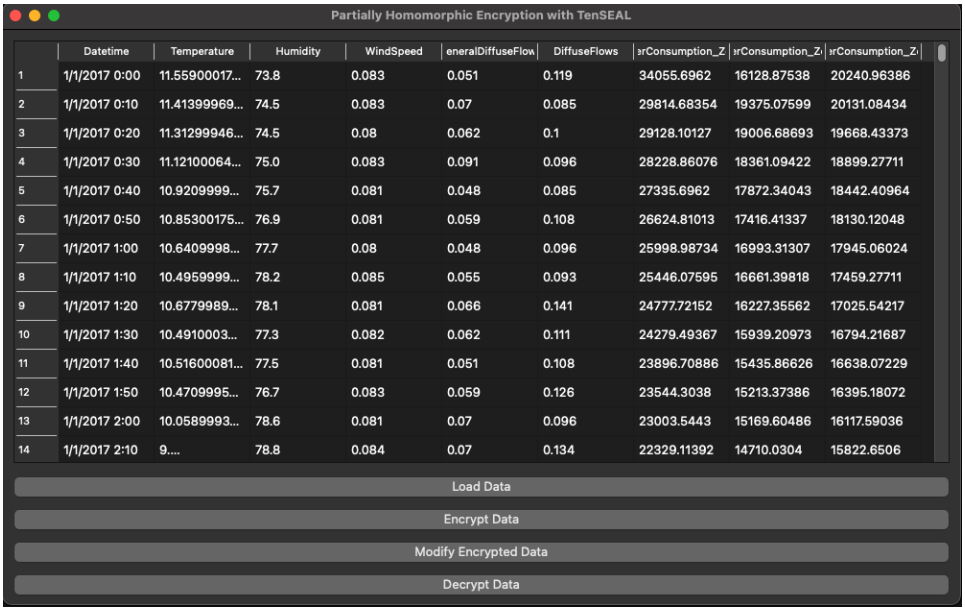
## Differential Privacy GUI




This is the layout of the GUI after loading a dataset. It generates columns once the dataset has completed loading and the user can select which column they want to perform differential privacy on. There are buttons such as “Revert to Original”, which will revert the data back to its original state before any sort of manipulation. The “Apply differential Privacy” performs said privacy method. “Save data” saves the data to a specified folder for further analysis after applying differential privacy.




# Partially Homomorphic GUI



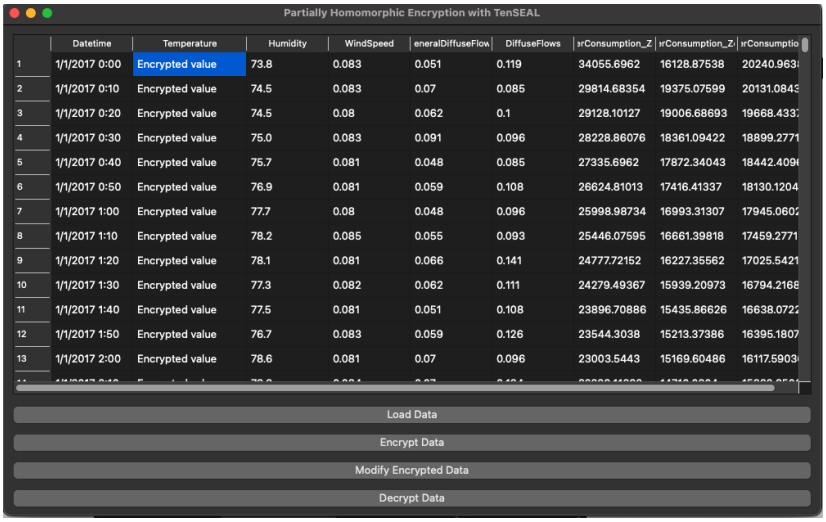

| | Datetime | Temperature | Humidity | WindSpeed | eneralDiffuseFlow | DiffuseFlows | rConsumption_Z | rConsumption_Z | rConsumption_Z |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1/1/2017 0:00 | 11.55900017... | 73.8 | 0.083 | 0.051 | 0.119 | 34055.6962 | 16128.87538 | 20240.96386 |
| 2 | 1/1/2017 0:10 | 11.41399969... | 74.5 | 0.083 | 0.07 | 0.085 | 29814.68354 | 19375.07599 | 20131.08434 |
| 3 | 1/1/2017 0:20 | 11.31299946... | 74.5 | 0.08 | 0.062 | 0.1 | 29128.10127 | 19006.68693 | 19668.43373 |
| 4 | 1/1/2017 0:30 | 11.12100064... | 75.0 | 0.083 | 0.091 | 0.096 | 28228.86076 | 18361.09422 | 18899.27711 |
| 5 | 1/1/2017 0:40 | 10.9209999... | 75.7 | 0.081 | 0.048 | 0.085 | 27335.6962 | 17872.34043 | 18442.40964 |
| 6 | 1/1/2017 0:50 | 10.85300175... | 76.9 | 0.081 | 0.059 | 0.108 | 26624.81013 | 17416.41337 | 18130.12048 |
| 7 | 1/1/2017 1:00 | 10.6409998... | 77.7 | 0.08 | 0.048 | 0.096 | 25998.98734 | 16993.31307 | 17945.06024 |
| 8 | 1/1/2017 1:10 | 10.4959999... | 78.2 | 0.085 | 0.055 | 0.093 | 25446.07595 | 16661.39818 | 17459.27711 |
| 9 | 1/1/2017 1:20 | 10.6779989... | 78.1 | 0.081 | 0.066 | 0.141 | 24777.72152 | 16227.35562 | 17025.54217 |
| 10 | 1/1/2017 1:30 | 10.4910003... | 77.3 | 0.082 | 0.062 | 0.111 | 24279.49367 | 15939.20973 | 16794.21687 |
| 11 | 1/1/2017 1:40 | 10.51600081... | 77.5 | 0.081 | 0.051 | 0.108 | 23896.70886 | 15435.86626 | 16638.07229 |
| 12 | 1/1/2017 1:50 | 10.4709995... | 76.7 | 0.083 | 0.059 | 0.126 | 23544.3038 | 15213.37386 | 16395.18072 |
| 13 | 1/1/2017 2:00 | 10.0589993... | 78.6 | 0.081 | 0.07 | 0.096 | 23003.5443 | 15169.60486 | 16117.59036 |
| 14 | 1/1/2017 2:10 | 9.... | 78.8 | 0.084 | 0.07 | 0.134 | 22329.11392 | 14710.0304 | 15822.6506 |

In the application, the initial step involves reading the data from a CSV file, loading it into memory for further processing. Once the data is loaded, users are presented with the option to encrypt the data, specifically targeting the "Temperature" column for encryption.

When the user selects the "Encrypt data" option, the application initiates the process of homomorphic encryption on the "Temperature" column. Homomorphic encryption allows

computations to be performed directly on encrypted data without the need for decryption. In this context, the temperature values are encrypted using a homomorphic encryption scheme.

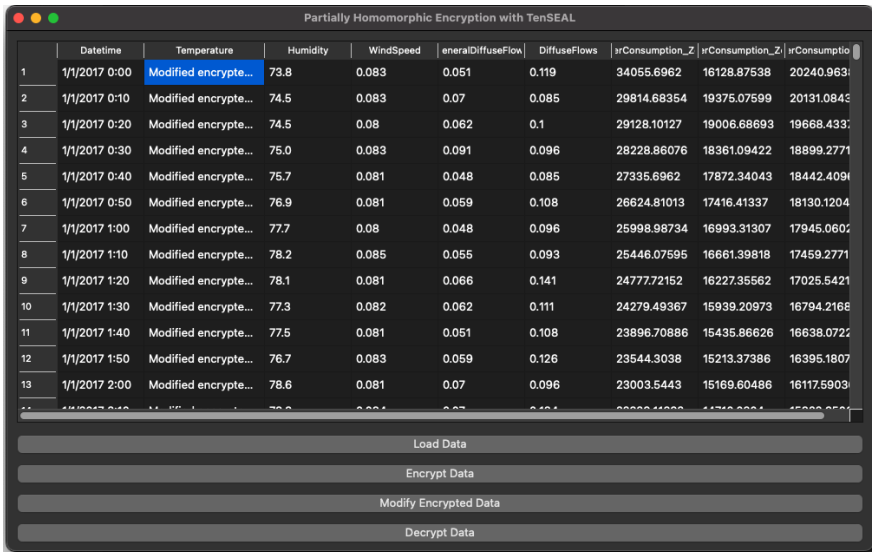
By encrypting the "Temperature" column, the original temperature values are transformed into encrypted ciphertexts, ensuring that the sensitive information remains confidential throughout the analysis process. This encryption process ensures that even though the data is being manipulated for analysis, the original temperature readings remain protected from unauthorized access or disclosure.



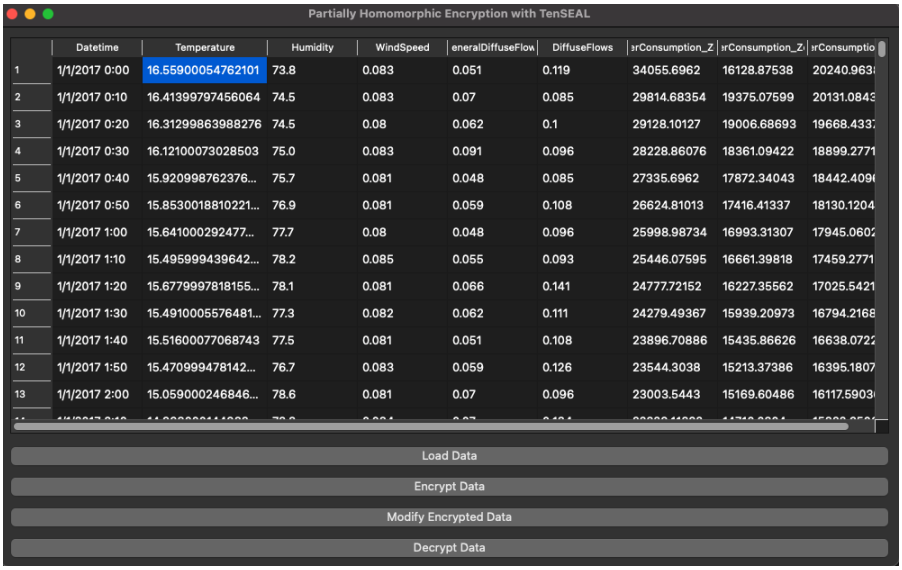

| | Datetime | Temperature | Humidity | WindSpeed | eneralDiffuseFlow | DiffuseFlows | rConsumption_Z | rConsumption_Z | rConsumptio |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1/1/2017 0:00 | Encrypted value | 73.8 | 0.083 | 0.051 | 0.119 | 34055.6962 | 16128.87538 | 20240.963 |
| 2 | 1/1/2017 0:10 | Encrypted value | 74.5 | 0.083 | 0.07 | 0.085 | 29814.68354 | 19375.07599 | 20131.0843 |
| 3 | 1/1/2017 0:20 | Encrypted value | 74.5 | 0.08 | 0.062 | 0.1 | 29128.10127 | 19006.68693 | 19668.433 |
| 4 | 1/1/2017 0:30 | Encrypted value | 75.0 | 0.083 | 0.091 | 0.096 | 28228.86076 | 18361.09422 | 18899.2771 |
| 5 | 1/1/2017 0:40 | Encrypted value | 75.7 | 0.081 | 0.048 | 0.085 | 27335.6962 | 17872.34043 | 18442.409 |
| 6 | 1/1/2017 0:50 | Encrypted value | 76.9 | 0.081 | 0.059 | 0.108 | 26624.81013 | 17416.41337 | 18130.1204 |
| 7 | 1/1/2017 1:00 | Encrypted value | 77.7 | 0.08 | 0.048 | 0.096 | 25998.98734 | 16993.31307 | 17945.0602 |
| 8 | 1/1/2017 1:10 | Encrypted value | 78.2 | 0.085 | 0.055 | 0.093 | 25446.07595 | 16661.39818 | 17459.2771 |
| 9 | 1/1/2017 1:20 | Encrypted value | 78.1 | 0.081 | 0.066 | 0.141 | 24777.72152 | 16227.35562 | 17025.5421 |
| 10 | 1/1/2017 1:30 | Encrypted value | 77.3 | 0.082 | 0.062 | 0.111 | 24279.49367 | 15939.20973 | 16794.2168 |
| 11 | 1/1/2017 1:40 | Encrypted value | 77.5 | 0.081 | 0.051 | 0.108 | 23896.70886 | 15435.86626 | 16638.0722 |
| 12 | 1/1/2017 1:50 | Encrypted value | 76.7 | 0.083 | 0.059 | 0.126 | 23544.3038 | 15213.37386 | 16395.1807 |
| 13 | 1/1/2017 2:00 | Encrypted value | 78.6 | 0.081 | 0.07 | 0.096 | 23003.5443 | 15169.60486 | 16117.5903 |

After the data encryption process is completed, the application updates the display to reflect the encrypted status of the "Temperature" column. Specifically, instead of displaying the original temperature values, the application now showcases the encrypted data with the word "Encrypted" in place of the numerical values.

This alteration in the display ensures transparency regarding the encryption status of the "Temperature" column. Users can readily discern that the data in this column has undergone encryption and is now represented in its protected form. By replacing the actual numerical values with a clear indicator like "Encrypted," the application effectively communicates to users that the sensitive information has been safeguarded using homomorphic encryption.




When users choose to modify the encrypted data, the application conducts a multiplication operation directly on the encrypted values using homomorphic encryption. This allows mathematical operations without decryption. After the operation, the display indicates the modification by replacing the "Encrypted" label with "Modified Encrypted Values," ensuring transparency and showcasing the versatility of homomorphic encryption. This feature enhances user confidence and understanding, emphasizing privacy preservation while enabling analysis of encrypted data.




Selecting the "Decrypt" option triggers the decryption process for the encrypted "Temperature" data. Once decrypted, the original numerical values replace the encrypted representations in the dataset, making them accessible for analysis or visualization. This feature maintains privacy during encryption while ensuring data usability for users.

# 8. Future Works

As part of future enhancements, implementing the Exponential Mechanism would be beneficial. This mechanism is particularly useful for scenarios involving non-numeric data or non-additive utility functions. To implement it, we need to carefully consider the utility function and sensitivity of the selection process. By exploring and incorporating the Exponential Mechanism, we can enhance the application's ability to handle diverse data types and decision-making processes while maintaining privacy and preserving utility.

# 9. Summary

Through this project, we've gained invaluable insights into the significance of privacy, particularly in handling sensitive energy consumption data. Our understanding of privacy-preservation techniques significantly expanded as we implemented both differential privacy mechanisms and homomorphic encryption methods, ensuring robust privacy while facilitating insightful analysis. Leveraging libraries like diffprivlib and tenseal, we seamlessly integrated these techniques into our application, underscoring the practicality of privacy preservation in real-world scenarios. Moreover, we comprehended the delicate balance between privacy and utility, navigating privacy budgets and trade-offs to maintain data utility while safeguarding privacy. Looking ahead, we've identified avenues for further improvement, such as exploring the Exponential Mechanism for handling non-numeric data, reflecting our commitment to continuous learning and advancement in privacy-preserving techniques.

# 10. References

1.https://www.researchgate.net/publication/332827026_Comparison_of_Machine_Learning_Algorithms_for_the_Power_Consumption_Prediction_-_Case_Study_of_Tetouan_city_-

2. diffpriv: An R Package for Easy Differential Privacy Benjamin I. P. Rubinstein

3. Francesco Ald`a and Benjamin I. P. Rubinstein. The Bernstein mechanism: Function release under differential privacy. In Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI’2017), pages 1705–1711, 2017