Object Oriented Programming for Data Science

Data Filtering and Smoothing.

- Rishikesh Tiwari (11257044)
- Suheb Khan (11266079)
- Kumpal Khokhariya (11266110)
- Milankumar Mavani (11280466)
- Nouman Ahmad (11273745)

Technology
Arts Sciences
TH Köln

Overview

- Outlier Recognition
- Interpolation
- Data Smoothing
- GUI
- Code Implementation and Results

Outlier Recognition

Incorrect data or Scientifically interesting?

- Z-score: Measures the deviation of data points from the mean in terms of standard deviations.
- Interquartile Range (IQR): Defines outliers as data points outside the range of threshold times the IQR above the third quartile or below the first quartile.
- Isolation Forest: Identifies outliers based on their isolation from the majority of the data points.

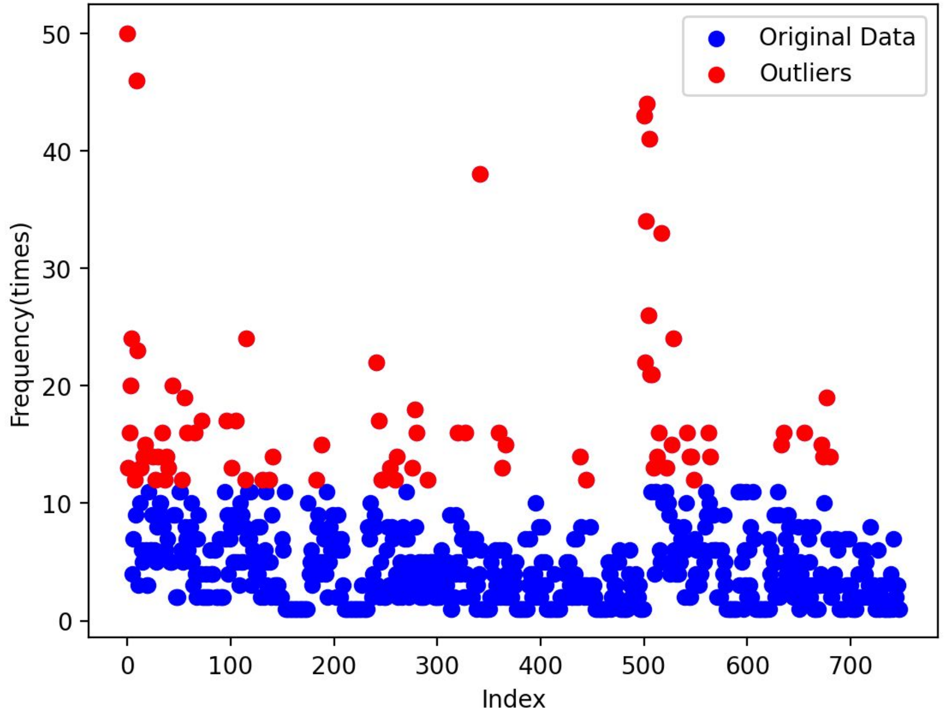


Image Source - Project code UI output with the data provided by Prof. Dr. Wolf.

Z-Score

Z = (x−μ)/σ

Where:

x is the individual data point,

μ is the mean of the dataset,

σ is the standard deviation of the dataset.

```
"""
Class for outlier detection methods
"""
class OutlierDetection:
    def __init__(self, df):                                  # Initializing class with a DataFrame
        self.df = df                                         # Assigning the DataFrame to an attribute
    """
    Method for detecting outliers using Z-score
    """
    def detect_outliers_zscore_drop(self, column_names, threshold):
        cleaned_data_df = self.df.copy()                     # Make a copy of the DataFrame
        for column in column_names:                          # Iterate through specified column names
            column_data = self.df[column]                    # Extract data from the specified column
            z_scores = zscore(column_data)                   # Calculate z-scores for the column data
            z_score_outliers = np.abs(z_scores) > threshold  # Identify outliers based on z-score and threshold
            cleaned_data_df.loc[z_score_outliers, column] = np.nan  # Replace outliers with NaN values

        return cleaned_data_df                               # Return the DataFrame with outliers replaced by NaN values
```

To identify outliers:

- Z > threshold the data point is considered an outlier above the mean.
- Z < −threshold the data point is considered an outlier below the mean.

Image Source - Project code.

IQR

- Column df is arranged into ascending order.

- IQR = Q3 - Q1

- Lower bound (Q1 - t * IQR)
- Upper bound (Q3 + t * IQR)

t = threshold

```python
"""
Method for detecting outliers using Inter Quartile Range
"""
def detect_outliers_iqr_drop(self, column_names, threshold):
    cleaned_data_df = self.df.copy()                        # Make a copy of the DataFrame
    for column in column_names:                             # Iterate through specified column names
        column_data = self.df[column]                       # Extract data from the specified column
        q1 = column_data.quantile(0.25)                     # Calculating first quartile
        q3 = column_data.quantile(0.75)                     # Calculating third quartile
        iqr = q3 - q1                                       # Calculating interquartile range
                                                            # Identify outliers based on IQR and threshold
        iqr_outliers = self.df[(column_data < q1 - threshold * iqr) | (column_data > q3 + threshold * iqr)].index.tolist()
                                                            # Replace outliers with NaN values
        cleaned_data_df.loc[iqr_outliers, column] = np.nan
    return cleaned_data_df
```

To identify outliers:

- Values that lies outside the bounds are identified as outliers

Isolation Forest

Let's consider a dataset containing the following values:

{10, 20, 30, 40, 50, 100}

```
"""
Method for detecting outliers using Isolation Forest
"""

def detect_outliers_isof_drop(self, column_names, contamination_rate):
    cleaned_data_df = self.df.copy()                                # Make a copy of the DataFrame
    for column_name in column_names:                                # Iterate through specified column names
        column_data = self.df[column_name]                          # Extracting column data
        isolation_forest = IsolationForest(max_samples=100, contamination=contamination_rate, random_state=42)
        isolation_forest_outliers = isolation_forest.fit_predict(column_data.values.reshape(-1, 1))
        outliers_mask = isolation_forest_outliers == -1             # Ceates a boolean mask where True indicates the presence of an outlier
        cleaned_data_df.loc[outliers_mask, column_name] = np.nan # Replace outliers with NaN values
    return cleaned_data_df
```

To identify outliers:

- Trees in the forest randomly select features and split values to isolate anomalies.
- Points that require fewer splits to isolate are considered outliers.

The algorithm scores each point based on its isolation, and those with lower scores are flagged as outliers.

Image Source - Project code.

Interpolation.

- Linear Interpolation.
- Quadratic Interpolation.
- Cubic Interpolation.

Linear Interpolation

- Linear interpolation is a form of interpolation, which involves the generation of new values based on an existing set of values.
- Linear interpolation is achieved by geometrically rendering a straight line between two adjacent points on a graph or plane.
- All points on the line other than the original two can be considered interpolated values.

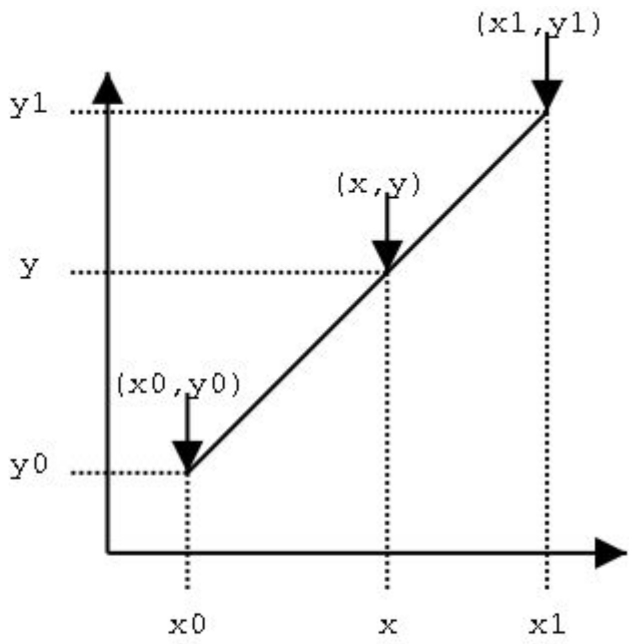


Image Source - toppr.com
Source - toppr.com

Quadratic Interpolation

- Quadratic interpolation is a method of estimating values between two known data points using a quadratic polynomial. Given three points, the goal is to find a quadratic function that passes through these points, allowing you to make predictions for values within the range of the given points.
- The general form of a quadratic polynomial is

  y = ax2 + bx + c.

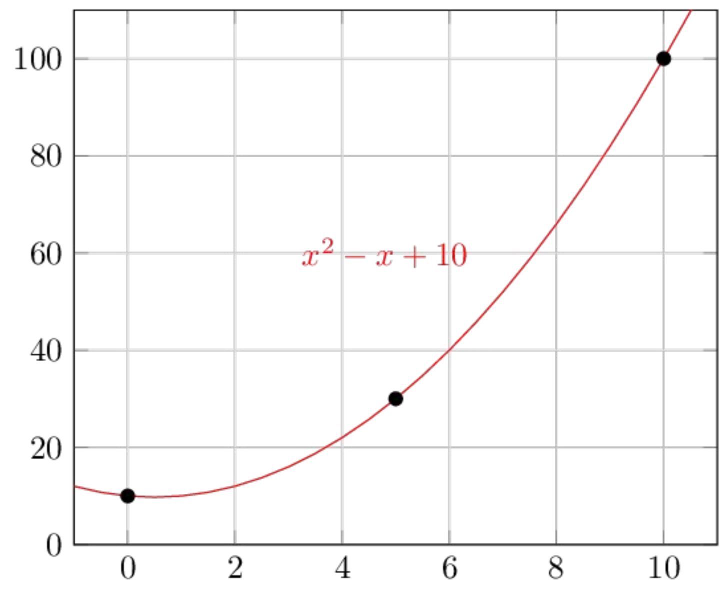
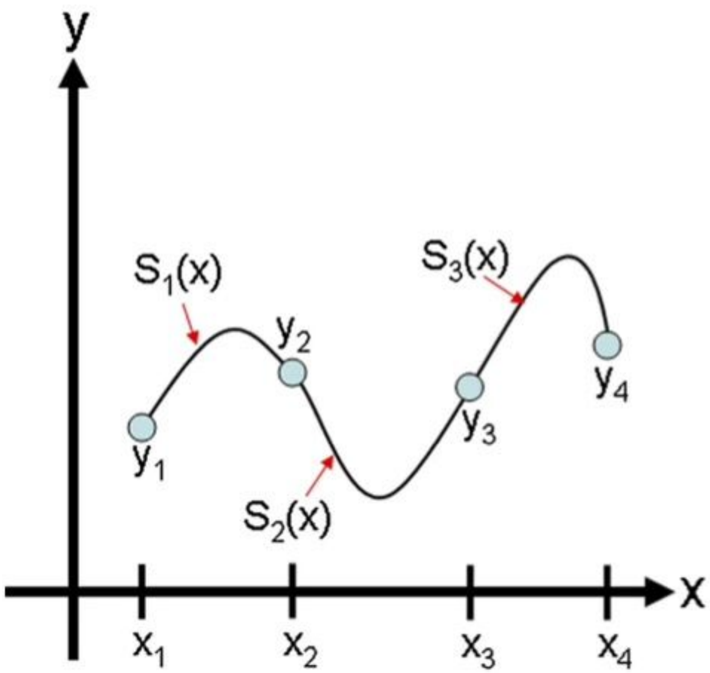


Image Source - https://www.baeldung.com/cs/polynomial-interpolation
Source - https://www.geeksforgeeks.org/quadratic-interpolation/

Cubic Interpolation

- In cubic spline interpolation the interpolating function is a divide in a set of piecewise cubic functions and this function use to connect adjacent data points.
- Each points (xi,yi) and (xi+1,yi+1) are joined by a cubic polynomial Si(x)=aix3+bix2+cix+di that is valid for xi≤x≤xi+1 for i=1,…,n−1.
- To find the interpolating function, we must first determine the coefficients ai,bi,ci,di for each of the cubic functions. For n points, there are n−1 cubic functions to find, and each cubic function requires four coefficients. Therefore we have a total of 4(n−1) unknowns, and so we need 4(n−1) independent equations to find all the coefficients.



Source - https://pythonnumericalmethods.studentorg.berkeley.edu/notebooks/chapter17.03-Cubic-Spline-Interpolation.html
image Source - https://pythonnumericalmethods.studentorg.berkeley.edu/notebooks/chapter17.03-Cubic-Spline-Interpolation.html

Interpolation Class

```
class Interpolation:
    def __init__(self, available_methods=['linear', 'quadratic', 'cubic']):
        self.available_methods = available_methods                  # Initializing available interpolation methods

    def interpolation(self, df, column_names, method='linear'):
        for column_name in column_names:                              # Iterate through specified column names
                                                                      # Performing interpolation on the specified column
            if method in self.available_methods:                      # Checking if the interpolation method is valid
                                                                      # Interpolating using specified method
                df[column_name] = df[[column_name]].interpolate(method=method)
                                                                      # Filling any remaining NaN values
                df[column_name] = df[[column_name]].interpolate(method='bfill')
                df[column_name] = df[[column_name]].interpolate(method='ffill')
            else:
                                                                      # Raise an error if the interpolation method is invalid
                raise ValueError("Invalid interpolation method. Please choose one of the following: {}".format(self.available_methods))
        return df                                                     # Returning the interpolated DataFrame
```

- **DataFrame.interpolate(method='linear', *, axis=0, limit=None, inplace=False, limit_direction=None, limit_area=None, downcast=_NoDefault.no_default,, **kwargs).**

Source - https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.interpolate.html

Data Smoothing.

- Moving Average.
- Savitzky-Golay Filter.

Moving Average.

- Moving Average are a common data filtering technique used to smooth time-series data.
- They help revel trends and patterns by reducing noises and fluctuations.
- Parameters-
  - ‘df’ - Input Dataset.
  - ‘filter_length’- Size of the Moving window set.
  - ‘min_period’ - The initial size of Input dataset.

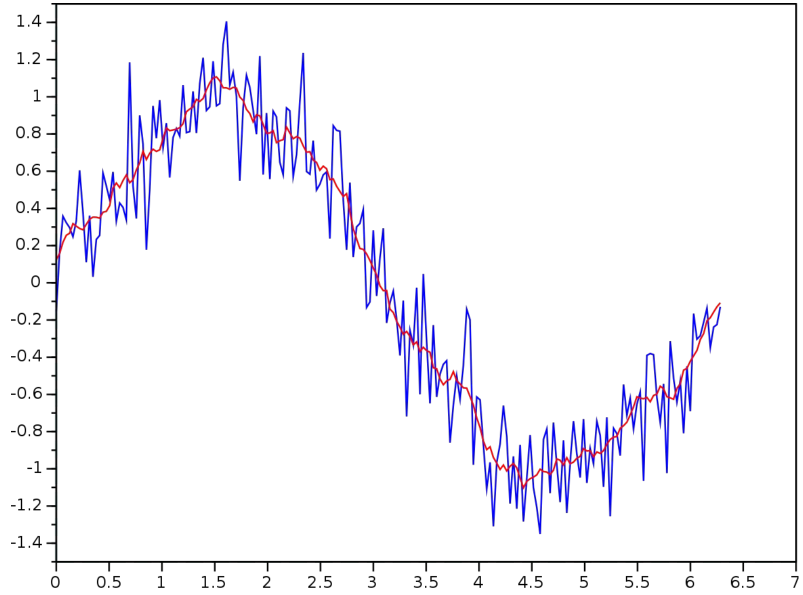


Image Source -Wikipedia
Source - Wikipedia

Technology
Arts Sciences
TH Köln

Moving Average

- **Let’s see the effect of filter_length and min_period**
  - Adjusting the filter_length parameter controls the smoothing effect and min_period tells us from which data point it should consider taking the mean.
  - Let’s take an example for better understandment, let's say the filter length is 7 and we have a time series with only 5 data points: [10,30,45,50,60]
  - For the first data point, it will take the mean of the available data points [10] (as there's only one data point).
  - For the second data point, it will take the mean of the available data points [10, 30] (as there are two data points).

Moving Average

- **DataFrame.rolling(*window*, *min_periods*=*None*, *center*=*False*, *axis*=*_NoDefault.no_default*, *closed*=*None*, *method*=*'single'*)**

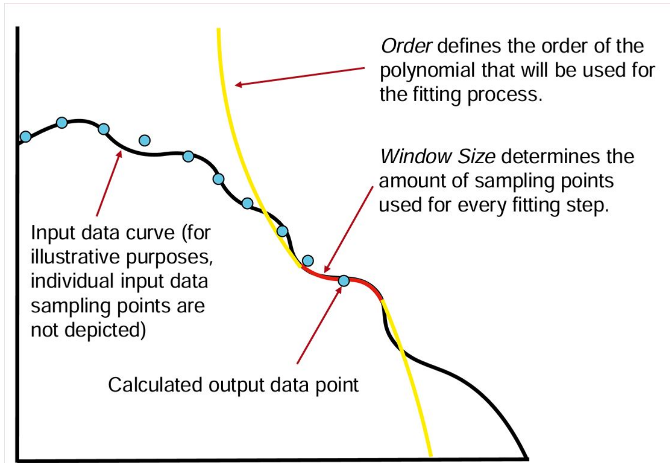
```python
class Smoothing:

    def __init__(self, df):
        self.df = pd.DataFrame(df)                        # Initializing class with a DataFrame

    """
    Method for applying moving average filter to smooth data
    """
    def moving_average(self, column_name, filter_length):
                                                          # Output DataFrame to store results
        df_var = pd.DataFrame()
                                                          # Iterate over each column in the DataFrame
        df_var = self.df.copy()                           # Creating a copy of the DataFrame
        for column in column_name:                        # Iterating through specified column names
                                                          # Calculating moving average and updating the column with smoothed values
            df_var[column] = self.df[column].rolling(filter_length, min_periods=1).mean()
        return df_var                                     # Returning the smoothed DataFrame
```

Savitzky-Golay filter.

- Savitzky–Golay filter is a digital filter that can be applied to a set of data points for the purpose of smoothing the data.
- **Moving Window Approach:** The Savitzky-Golay filter operates on the principle of a moving window, where a window of a specified length slides along the data set.
- **Polynomial Fitting:** Within each window, a polynomial of a predetermined degree (specified by the user) is fitted to the data points.
  The polynomial is typically fitted using least squares regression, minimizing the sum of the squared differences between the observed and predicted values.

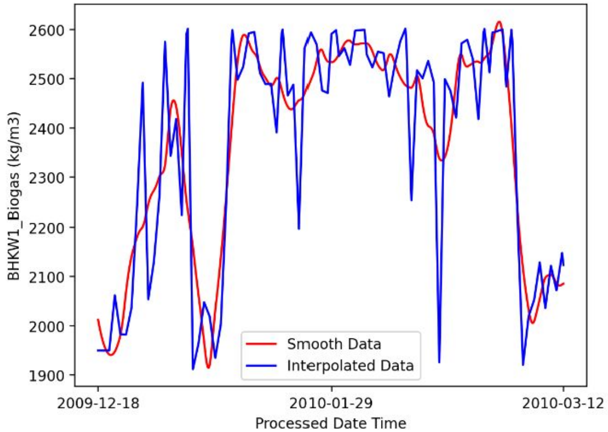


Source: https://en.wikipedia.org/wiki/Savitzky%E2%80%93Golay_filter
Image Source: https://www.lighttrans.com/use-cases/feature-use-cases/savitzky-golay-filter-function.html

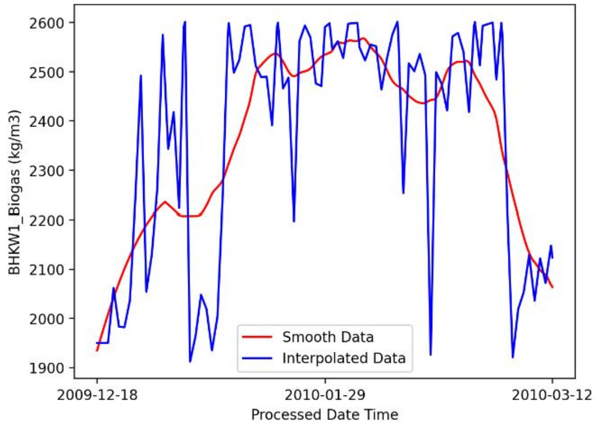
Savitzky-Golay filter.

Effects on the Filter - Window Size

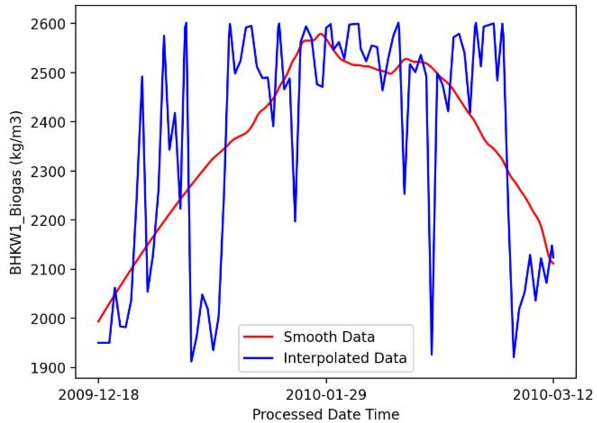
Window size determines how many neighboring points are considered in the fitting process. A larger window size will result in a smoother output but may potentially blur out important details.



Filter Length 100 & Order 2



Filter Length 300 & Order 2



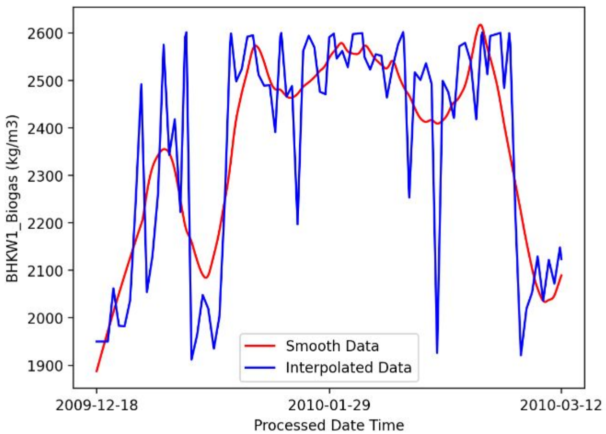
Filter Length 500 & Order 2

Image Source - Project code UI output with the data provided by Prof. Dr. Wolf.

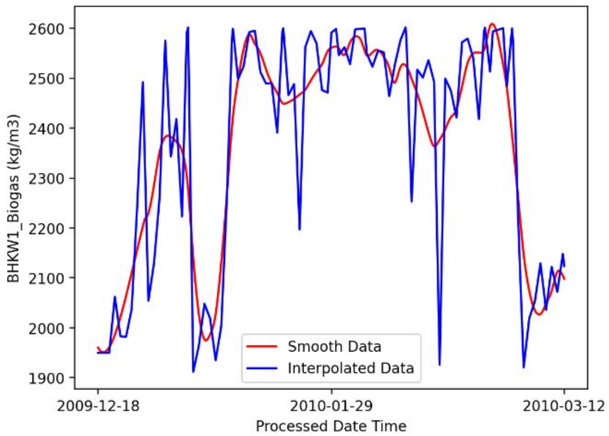
Savitzky-Golay filter.

Effects on the Filter - Order of Polynomial

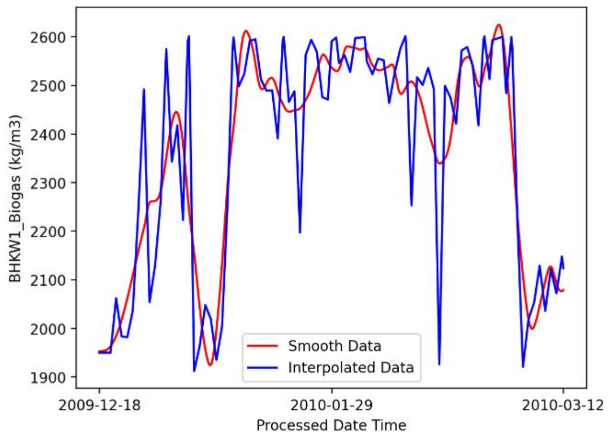
Higher polynomial orders allow for more flexibility in fitting complex data, but they can also introduce more variability if the data is noisy. Typically, lower-order polynomials (e.g., 2nd or 3rd order) are used for smoothing.



Filter Length 200 & Order 2



Filter Length 200 & Order 4



Filter Length 200 & Order 6

Image Source - Project code UI output with the data provided by Prof. Dr. Wolf.

Savitzky-Golay filter.

scipy.signal.savgol_filter

scipy.signal.savgol_filter(x, window_length, polyorder, deriv=0, delta=1.0, mode='interp')

```
"""
Method for applying Savitzky-Golay filter to smooth data
"""
def savitzky_golay(self, column_name, filter_length, order):
    df_var = self.df.copy()
    for column in column_name:

        df_var[column] = savgol_filter(df_var[column], filter_length, order)
    return df_var
```

'mirror' Mode: Repeats signal values at the edges in reverse order, excluding the closest value.
'nearest' Mode: Extends the signal with the nearest existing value at the edges.
'constant' Mode: Pads the signal with a constant value specified by the cval parameter.
'wrap' Mode: Wraps signal values from the opposite end of the array to extend the signal.
'interp' Mode(Default): Interpolates values at the edges based on existing data points for a smooth transition.

Source:https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.savgol_filter.html
Image Source - Project code

GUI - Introduction

- Definition: A Graphical User Interface (GUI) is a visual way of interacting with a computer using graphical elements such as windows, icons, buttons, and menus.
- Purpose: GUIs are designed to enhance user experience by providing an intuitive and user-friendly interface for interacting with software applications.
- Key Components: GUIs consist of various graphical elements, including windows, menus, buttons, text fields, sliders, and checkboxes.
- Importance: GUIs play a crucial role in software usability, accessibility, and user satisfaction, leading to increased productivity and efficiency.

GUI - Why Streamlit?

- Introduction:
  - Streamlit is a user-friendly Python framework for building interactive web applications with ease and speed.
- Key Benefits:
  - Ease of Use: Streamlit's simple syntax and built-in widgets enable rapid development without complex coding.
  - Rapid Prototyping: Fast iteration allows quick experimentation and visualization of data.
  - Rich Ecosystem: Extensive library support for data visualization, machine learning, and data processing enhances functionality.
  - Deployment Flexibility: Seamless deployment to various platforms for easy sharing and production use.
  - Community Support: Active community provides resources, tutorials, and assistance for development challenges.

GUI

- First look of the GUI

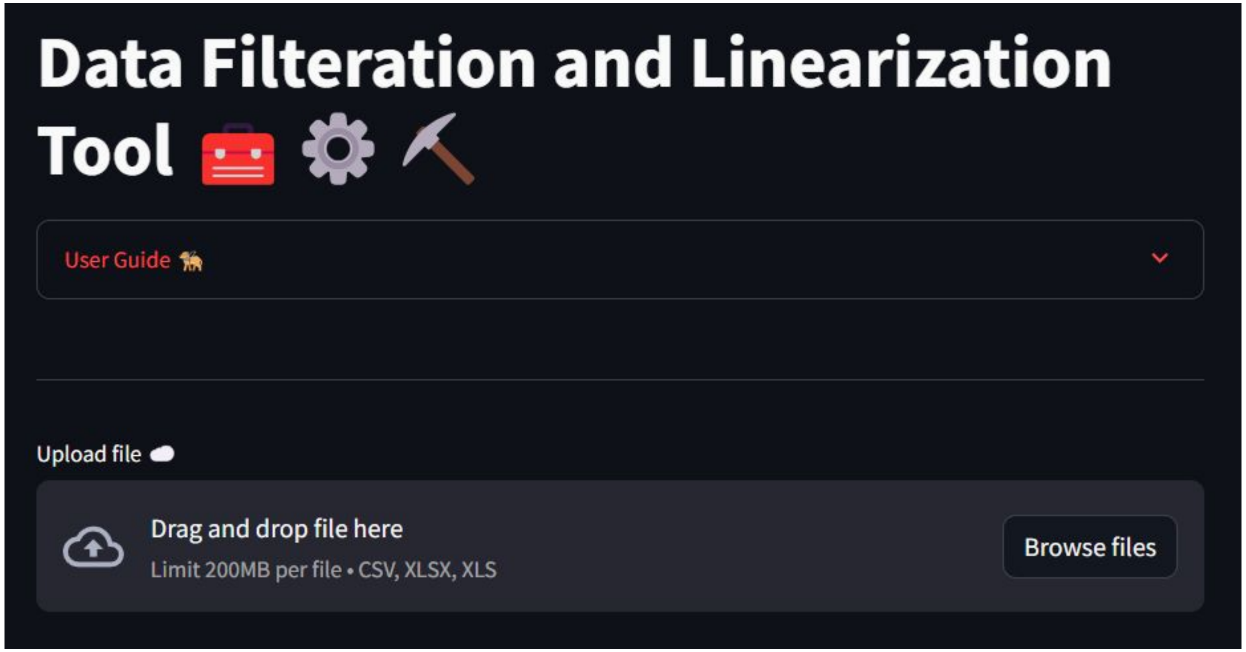


Image source - From our GUI code

GUI

- After uploading the file

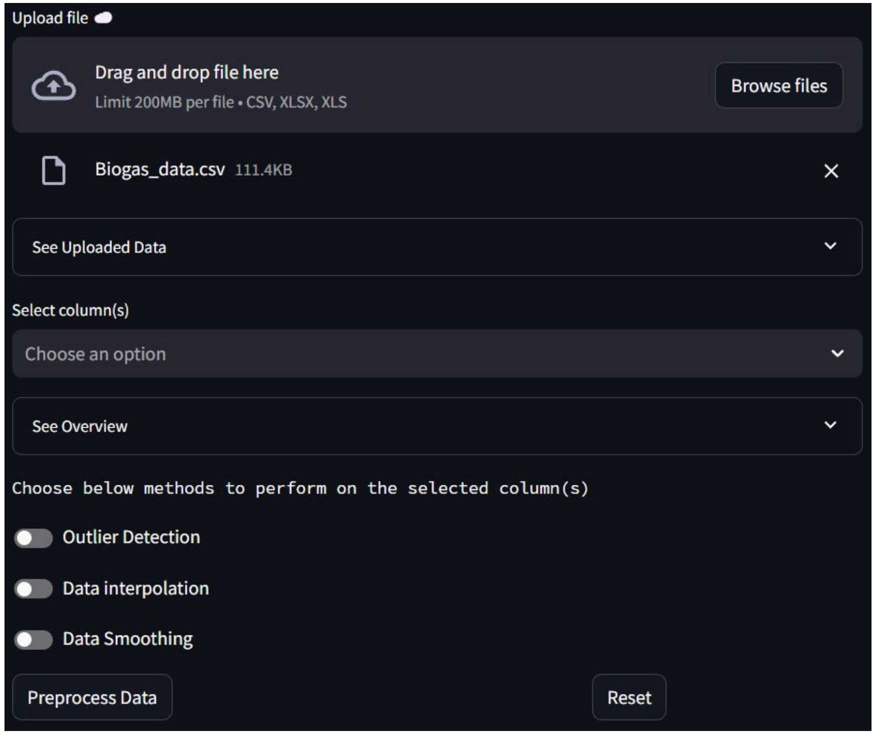


Image source - From our GUI code

GUI

- We can select the columns needs to be pre-processed and see an overview of all the methods offered.

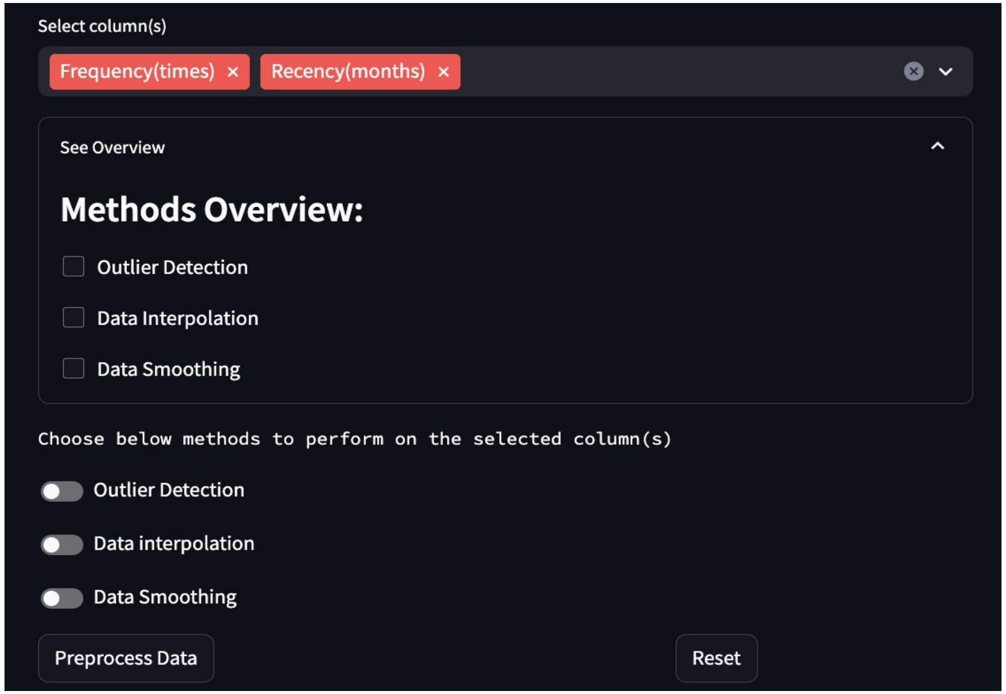
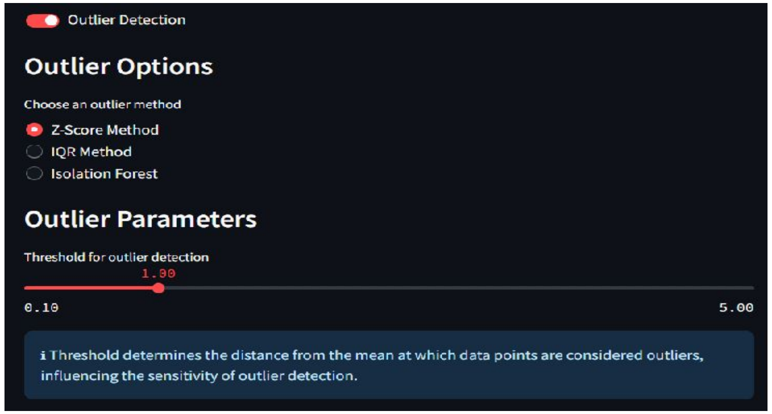
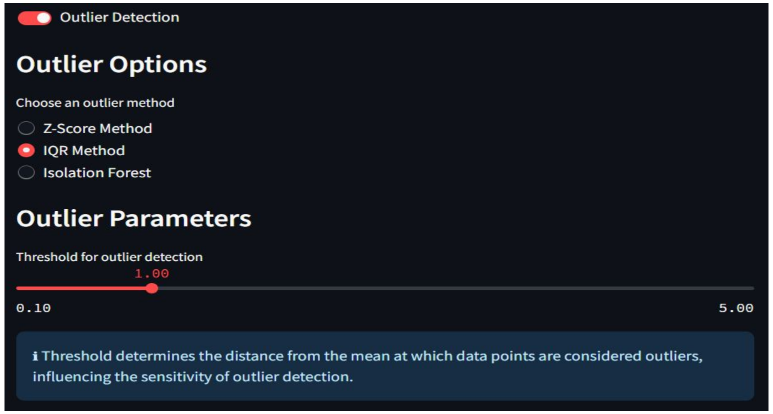


Image source - From our GUI code

GUI





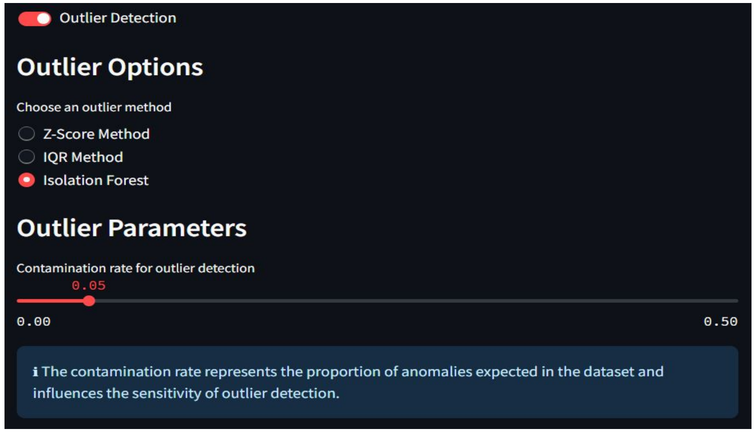


Image source - From our GUI code

GUI

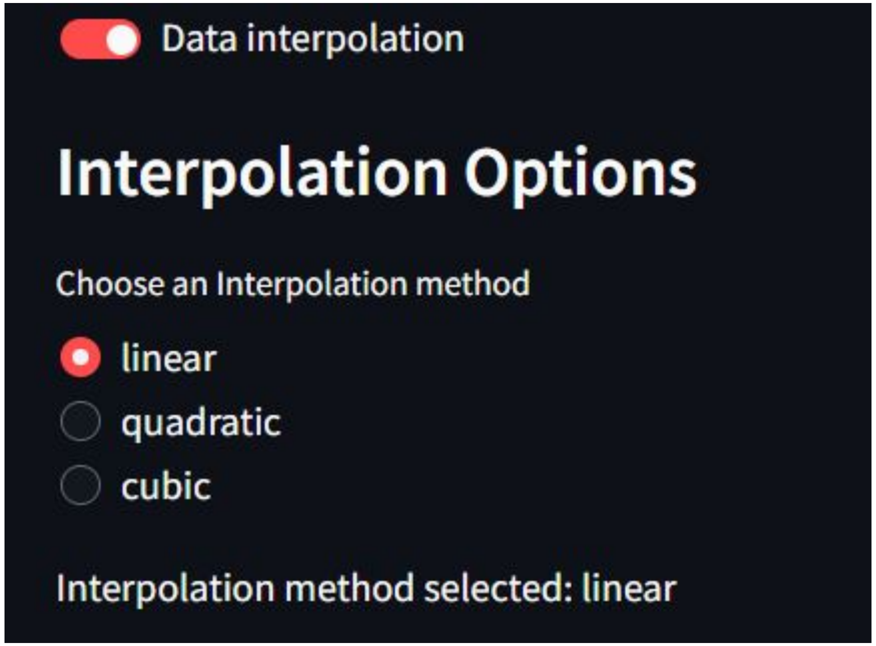


Image source - From our GUI code

GUI

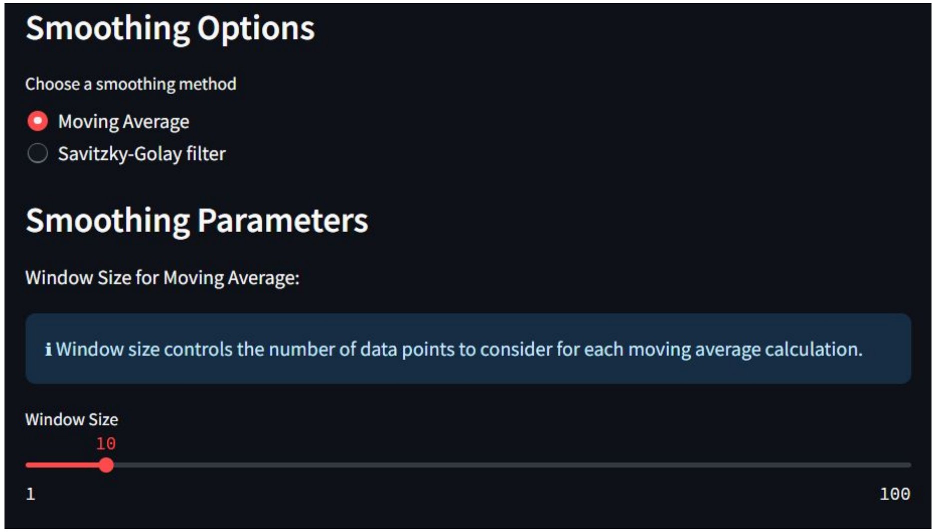
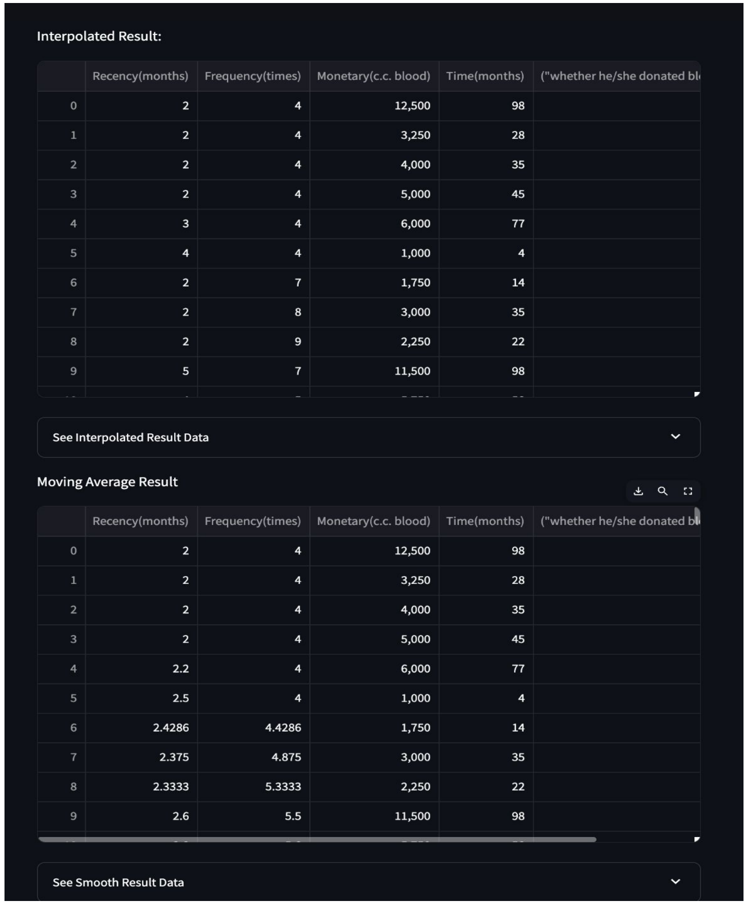


Image source - From our GUI code

GUI



Interpolated Result:

	Recency(months)	Frequency(times)	Monetary(c.c. blood)	Time(months)	("whether he/she donated bl
0	2	4	12,500	98	
1	2	4	3,250	28	
2	2	4	4,000	35	
3	2	4	5,000	45	
4	3	4	6,000	77	
5	4	4	1,000	4	
6	2	7	1,750	14	
7	2	8	3,000	35	
8	2	9	2,250	22	
9	5	7	11,500	98	

See Interpolated Result Data

Moving Average Result

	Recency(months)	Frequency(times)	Monetary(c.c. blood)	Time(months)	("whether he/she donated bl
0	2	4	12,500	98	
1	2	4	3,250	28	
2	2	4	4,000	35	
3	2	4	5,000	45	
4	2.2	4	6,000	77	
5	2.5	4	1,000	4	
6	2.4286	4.4286	1,750	14	
7	2.375	4.875	3,000	35	
8	2.3333	5.3333	2,250	22	
9	2.6	5.5	11,500	98	

See Smooth Result Data

Image source - From our GUI code

GUI

- The final part of our GUI.

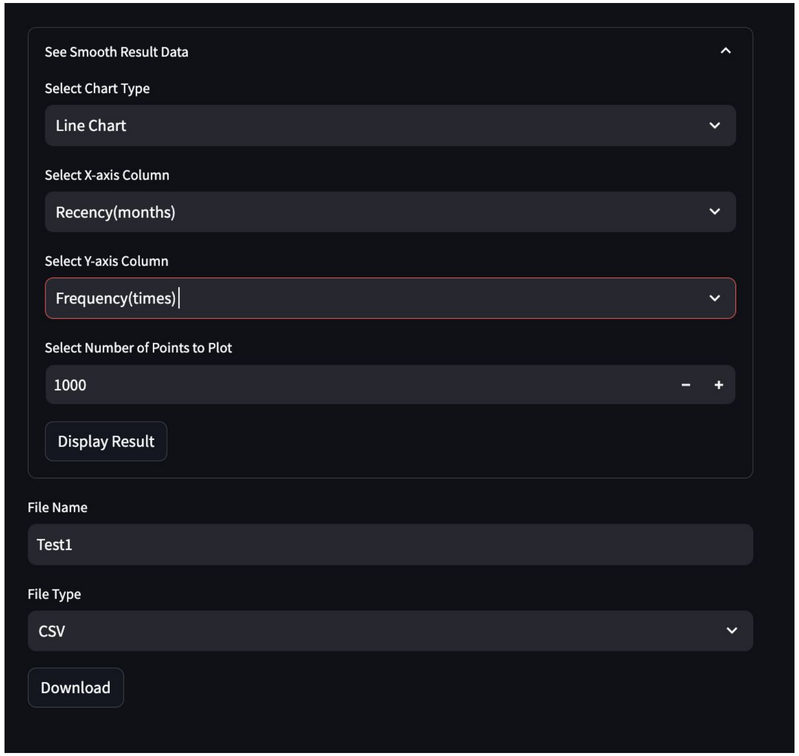


Image source - From our GUI code

Code Implementation and Results.

Let’s move over to our code for the results.

Thank You!