**Sentiment Analysis Code: A Detailed Overview**

**Objective and Context:**

**The following code is a sophisticated solution designed for analysing sentiments and emotions and categorisation on the basis of them and several other impact parameters from textual data regarding social media.**

**This code uses cutting-edge machine learning and natural language processing (NLP) techniques to collect, analyze, and present information from text. To produce a thorough analysis tool, it integrates sophisticated measures such as sentiment tones, emotional affects, and word-level frequencies. A Graphical User Interface (GUI) is also included to guarantee user-friendly interaction, which makes the tool accessible and adaptable for both technical and non-technical users.**

**Libraries and Tools Used**

* **Data Handling and Analysis:** 
  + **pandas for data manipulation.**
  + **collections.defaultdict for handling dictionary-based operations.**
* **Natural Language Processing:** 
  + **nltk for sentiment analysis (SentimentIntensityAnalyzer) and stopword handling.**
  + **spacy for multilingual stopword extraction.**
  + **Hugging Face Transformers for emotion classification (j-hartmann/emotion-English-distilroberta-base model).**
* **Data Visualization:** 
  + **matplotlib.pyplot for creating bar, line, and pie charts.**
* **GUI Development:** 
  + **tkinter and ttk for a user-friendly interface.**
* **Miscellaneous:** 
  + **re for regex-based text processing.**

**Features of the Code**

1. **Stopword Aggregation:**
   * **Combines stopwords from both NLTK and spaCy to ensure multilingual support, enabling robust text cleaning across various languages.**
2. **Data Loading:**
   * **Loads a CSV file (sentiment\_dataset.csv) containing the text data along with metadata such as likes and comments.**
3. **Sentiment Analysis:**
   * **Uses SentimentIntensityAnalyzer from NLTK to calculate sentiment scores for each text and stores the compound score as a "tone" column in the dataset.**
4. **Emotion Classification:**
   * **Employs the Hugging Face Transformers pipeline to classify text into emotions (joy, sadness, anger, surprise, fear). These scores are added as separate columns to the dataset.**
5. **Frequency Analysis:**
   * **Counts the frequency of each word across the dataset, excluding stopwords, and stores the cumulative word frequency for each row.**
6. **Impact Calculation:**
   * **Combines the tone and emotions with metadata (likes and comments) to calculate an "impact" metric for each text and individual emotions.**
7. **Word-Level Analysis:**

**Produces a second dataset (words.csv) that contains comprehensive word-level metrics including tone, frequency, and influence on a particular emotion.**

1. **Data Export:**
   * **Saves processed datasets (texts.csv and words.csv) for external analysis or visualization.**
2. **Graphical User Interface (GUI):**

**Offers an interactive interface for data selection and visualization through the use of Tkinter.**

**enables users to select, sort, and display results according to a number of parameters (e.g., emotion-specific metrics, tone, and effect).**

1. **Visualization Options:**
   * **Users can create bar, line, or pie charts to visualize data trends.**
   * **Supports customizable thresholds (e.g., highest, lowest, or extreme values).**

**Output Files**

1. **texts.csv:**
   * **Contains processed text data with the following key columns:** 
     + **text: Original text data.**
     + **tone: Sentiment score.**
     + **impact: Calculated overall impact of the text.**
     + **frequency: Cumulative word frequency for each text.**
     + **emotion\_joy, emotion\_sadness, etc.: Emotion-specific scores for each text.**
     + **impact\_joy, impact\_sadness, etc.: Impact values based on individual emotions.**
2. **words.csv:**
   * **Provides word-level metrics with the following columns:** 
     + **word: Individual words from the dataset.**
     + **frequency: Frequency of the word in the dataset.**
     + **likes and comments: Aggregated metadata associated with the word.**
     + **tone: Average sentiment score for the word.**
     + **emotion\_joy, emotion\_sadness, etc.: Emotion-specific scores for the word.**
     + **impact\_joy, impact\_sadness, etc.: Emotion-specific impact values for the word.**

**Use Cases**

1. **Social Media Analysis:**
   * **Analyze user sentiments and emotions from posts or comments.**
   * **Identify impactful texts or keywords driving engagement.**
2. **Marketing Insights:**
   * **Understand audience reactions to campaigns or products.**
   * Take out the terms and expressions that evoke the strongest feelings.
3. **Content Moderation:**
   * **Identify writings that have strong emotional effects or negative tones.**
   * **Identify specific words contributing to undesirable sentiments.**
4. **Research and Reporting:**
   * See patterns in the emotion and sentiment of text across datasets.
   * Give stakeholders thorough reports that include information they can use.

**Limitations and Improvements**

1. **Language Dependency:**

**Emotion analysis is limited to the pre-trained English model (j-hartmann/emotion-English-distilroberta-base). Support for other languages could be added with multilingual models.**

1. **Stopword Handling:**

**SpaCy and NLTK stopword lists may not cover all unwanted words, especially domain-specific terms.**

1. **GUI Enhancements:**

**Currently limited to basic selection and plotting. Features like saving plots, adding more filters, or customizing visuals could improve usability.**

1. **Computational Efficiency:**

**The script processes each text and word iteratively, which might be slow for very large datasets. Optimization through parallel processing could be explored.**

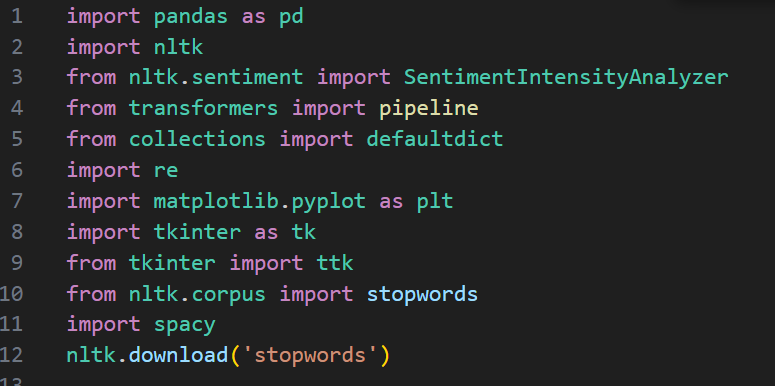
1. **Challenges in Model Selection:**

**It was difficult to find an appropriate open-source sentiment analysis model. Despite its limits in understanding complex human emotions such as sarcasm, jokes, humor, and subtle moods, the selected model was found to be the most suitable alternative after rigorous testing.**

**With insights at the word and text levels, this script offers a thorough method for mood and emotion analysis. It is a flexible tool for NLP-related activities and analytics because of its interactive GUI and comprehensive output datasets.**

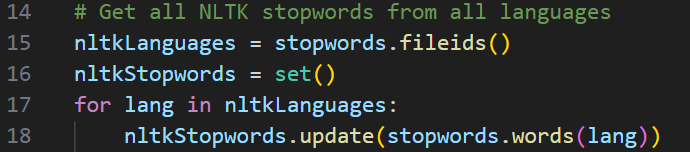
**Snippet Descriptions for the Code File: Sentiment Analysis.py**

**1. Importing Necessary Libraries:**

 This snippet imports necessary libraries and tools to allow the program's many features:

* **pandas**: Handles data manipulation, particularly useful for reading, storing, and modifying datasets.
* **nltk**: A natural language processing library that offers methods for handling linguistic data, such as stopwords, and the SentimentIntensityAnalyzer for scoring of different types of emotions.
* **transformers**: Facilitates advanced machine learning models, including pre-trained models for emotion classification.
* **collections.defaultdict**: Enables the efficient creation and management of nested dictionaries.
* **re**: A module for text pattern matching and regular expression operations.
* **matplotlib.pyplot**: Creates visualizations such as bar charts, line charts, and pie charts for data insights.
* **tkinter**: Implements a graphical user interface (GUI) to allow users to interact with the analysis.
* **spacy**: Provides support for handling multilingual text data, particularly for retrieving stopwords.
* **nltk.download('stopwords')**: Ensures the stopwords corpus is available for use in the program.

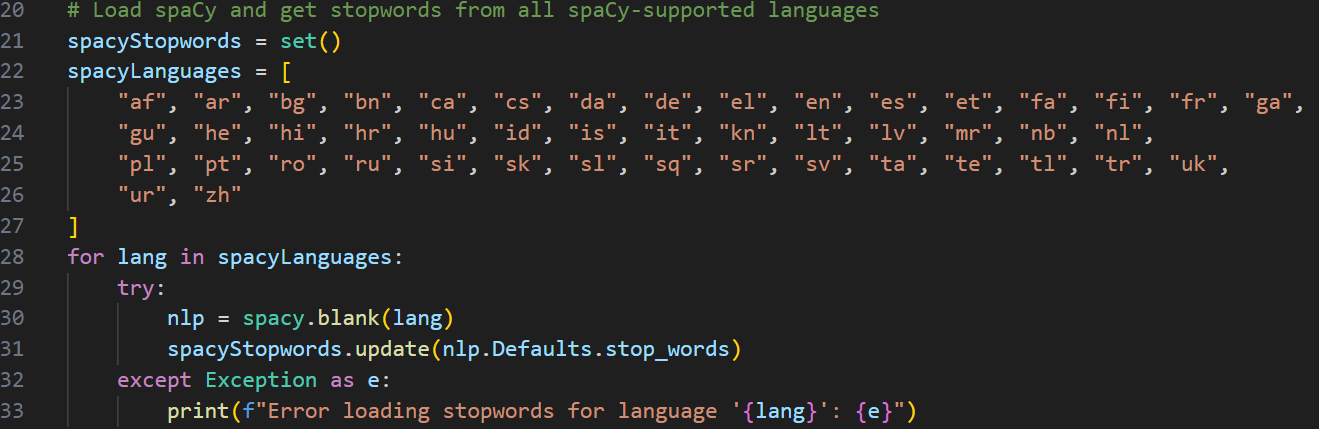
**2. Extracting Stopwords from NLTK:**



**This function generates an exhaustive list of stopwords for every language that is accessible from NLTK.**

* **Stopword Retrieval: Retrieves stopwords for every language by using fileids to retrieve all language identifiers supplied by NLTK.**
* **Set Creation: To guarantee uniqueness, the obtained stopwords are added to a Python set.**
* **Result: As a result, a comprehensive list of stopwords suitable for multilingual text processing is generated.**

**3. Extracting Stopwords from spaCy:**

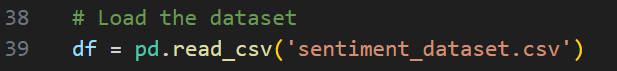
This code extracts stopwords from spaCy for multiple languages.

* **Language List**: Hardcodes supported languages.
* **Model Loading**: Attempts to load a blank spaCy language model and retrieve its predefined stopwords.
* **Error Management**: Gracefully handles exceptions if a language model fails to load.
* **Outcome**: Produces an additional stopword set specific to spaCy-supported languages.

**4. Merging Stopwords:**

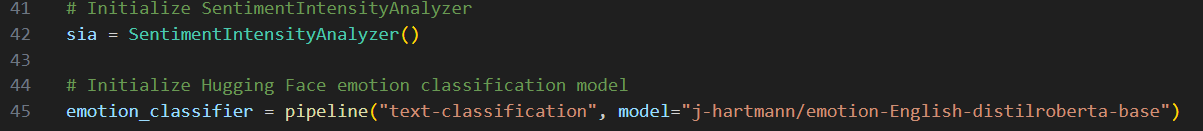
This line combines NLTK and spaCy stopwords into a single set, allowing unified filtering of common words from text data.

**5. Loading Dataset:**

This code reads a dataset into a pandas DataFrame:

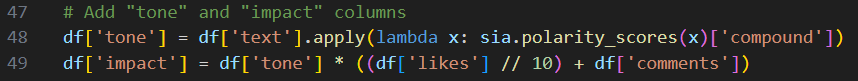
* **File Format**: Assumes the dataset is in CSV format and located in the same directory.
* **Expected Columns**: Likely contains fields such as text, likes, and comments.
* **Purpose**: Provides the raw data needed for subsequent processing and analysis.

**6. Initializing Sentiment Analysis Tools:**

This snippet initializes sentiment analysis tools:

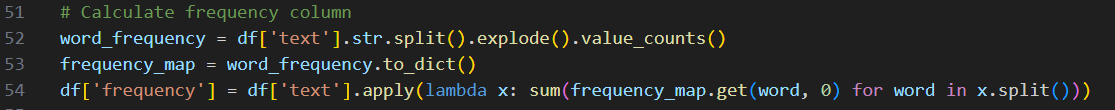
* SentimentIntensityAnalyzer (VADER): Provides a rule-based approach for calculating a compound polarity score for text sentiment.
* **Hugging Face Emotion Classifier**: The Hugging Face Emotion Classifier uses a transformer model that has already been trained to identify and rate emotions in text.

**7. Adding Tone and Impact Columns:**

 Two additional columns are added to the DataFrame in this section:

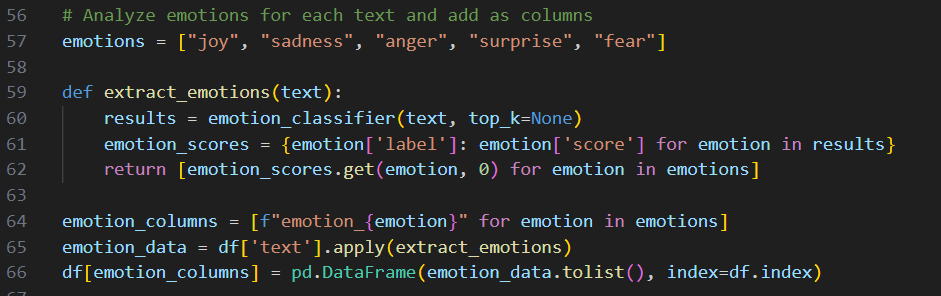
* **tone**: calculates a compound sentiment score for every text row using VADER..
* **impact**: Calculates an engagement-weighted score by multiplying tone with engagement metrics (likes scaled by a factor of 10 and comments).

**8. Calculating Word Frequency:**

This section calculates a **frequency column** that aggregates word occurrences:

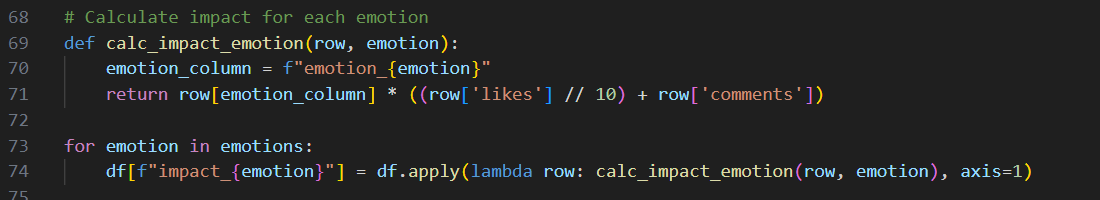
* **Frequency Mapping**: Splits text into words, flattens the structure using explode, and computes occurrences of each unique word using value\_counts().
* **Frequency Column**: For each row of text, sums the total occurrences of its words based on the computed frequency map.

**9. Extracting and Analyzing Emotions:**

This code analyzes the emotional content of each text row:

* **Emotion Extraction**: Uses the emotion classifier to score predefined emotions (e.g., joy, sadness) for each text.
* **Emotion Columns**: Adds a new column for each emotion to store its intensity.

**10. Calculating Emotion Impact:**



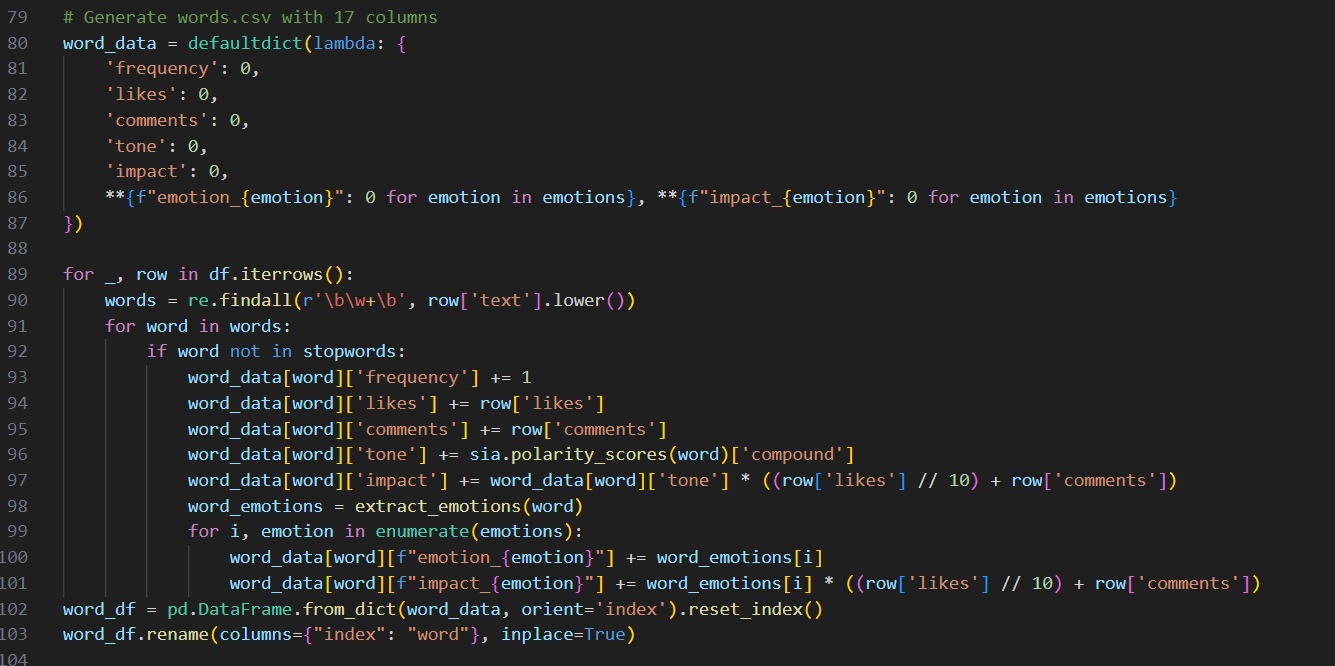
This section computes emotion-specific impact scores:

* **Impact Formula**: Multiplies the emotion intensity by engagement metrics (likes and comments).
* **Dynamic Columns**: Creates separate impact columns for each emotion.

**11. Saving Text Analysis Results:**

This saves the This creates a CSV file called texts.csv that has the transformed DataFrame with all of the new columns included.

**12. Generating Word-Level Analysis:**

This code performs detailed word-level analysis:

* **Data Aggregation**: Collects statistics for each word, including frequency, engagement, tone, and emotion scores.
* **Stopword Filtering**: Excludes stopwords to focus on meaningful words.
* **Emotion Attribution**: Distributes emotion scores proportionally based on engagement metrics.

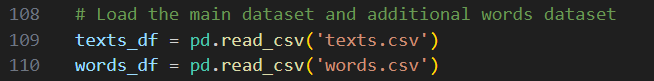
**13. Saving Word Analysis Results:**



Exports the word-level analysis to a CSV file (words.csv) for external use.

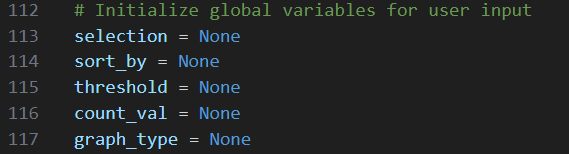
**Explanation of the Code for User Interaction and Visualization (GUI)**

**1. Loading Processed Data:**

 Two pre-processed datasets are loaded into Pandas using this snippet. DataFrames:  
Text-level analysis, including metrics like tone, effect, and emotional intensities, is contained in texts.csv.  
Word-level analysis is included in words.csv, which includes information on each word's frequency, likes, comments, tone, and emotion-specific effects.  
The data is now prepared for display and user engagement.

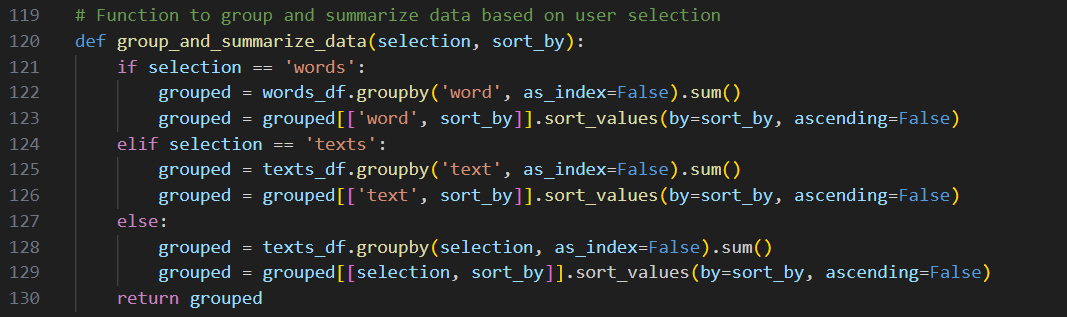
The data is now ready for user interactions and visualization.

**2. Initializing Global Variables for User Inputs:**

This section sets up global variables to store user inputs from the GUI:

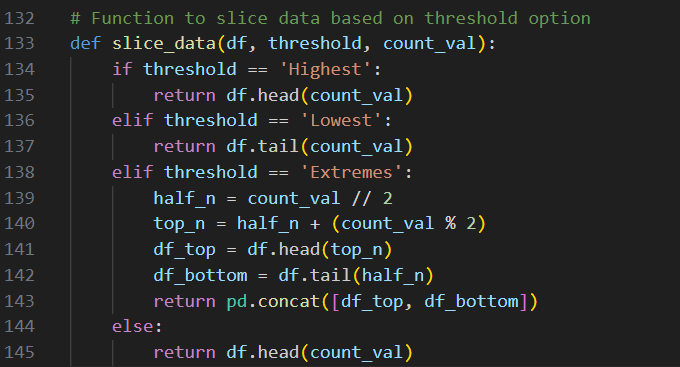
* **selection**: Determines whether to analyze texts, words, or other attributes.
* sort\_by: Indicates the measure (such as tone or impact) that will be used to sort the data.  
  threshold: Defines if the highest, lowest, or extreme numbers should be shown.  
  count\_val: Regulates how many entries are shown.  
  graph\_type: Selects the visualization type, such as a pie chart, line chart, or bar chart.

**3. Grouping and Summarizing Data:**

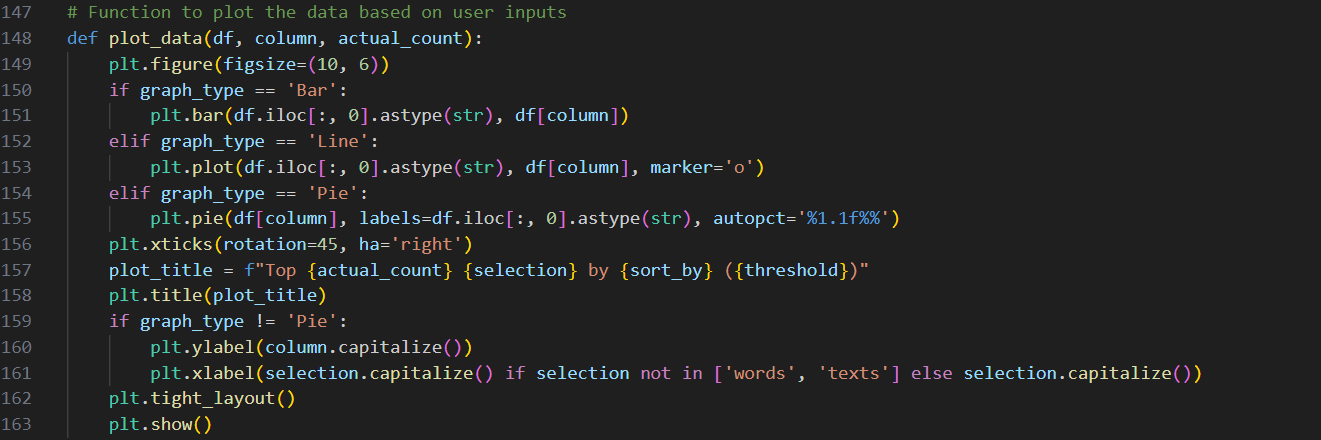


**This feature arranges and rates data according to user-specified standards:  
Grouping: Combines information according to the chosen characteristic (e.g., words or paragraphs).  
Sorting: Arranges the combined data according to the designated measure (e.g., tone or impact).  
A condensed DataFrame for visualization is the output.**

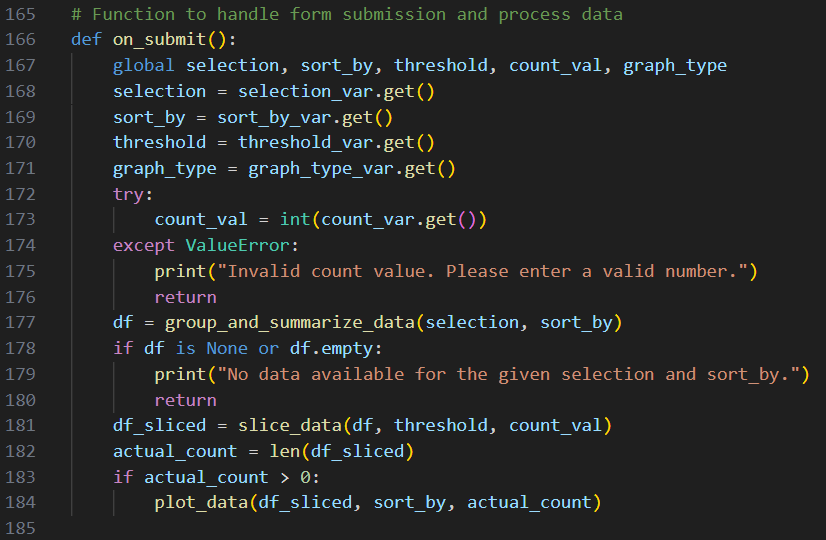
**4. Slicing Data for Display:**

Depending on user settings, this method pulls out a particular subset of data:  
Highest: Gets the best entries based on the chosen metric.  
Lowest: Gets the entries at the bottom.  
Extremes: Provides a balanced view by combining the top and bottom entries.  
By default, if no threshold is set, the output is restricted to the top entries.

**5. Plotting Data:**

Using user input, this function displays the sliced data:  
Data is displayed on a bar chart, where values are represented by bars.  
Line Chart: Displays trends using a line plot.  
Pie Chart: Uses a circular arrangement to highlight proportions.  
Dynamic Titles: Modifies titles and labels to correspond with user choices.

**6. Handling User Submissions:**

This feature interprets user input from the graphical user interface and presents the findings:  
Validation: Verifies the validity of the count value.  
Data preparation involves slicing and summarizing data according to user preferences.  
Visualization: Creates graphs by invoking the plotting function.

**7. Creating the GUI:**



T7. Building the GUI: A graphical user interface (GUI) is created by this snippet:  
User Input boxes: Offers buttons, text boxes, and drop-down menus to record user preferences.  
Layout: Clearly and intuitively arranges input components.  
Event Handling: Connects user input to the data processing on\_submit function.  
  
In brief  
The aforementioned algorithm allows for interactive study of pre-processed datasets by combining data processing, visualization, and an intuitive user interface. Through the GUI, users can view results in a variety of graphical representations and personalize their analysis.