MAKERERE UNIVERSITY

SCHOOL OF COMPUTING AND IMFORMATICS TECHNOLOGY

COLLEGE OF COMPUTING AND INFORMATION SCIENCES

MASTERS IN DATA COMMUNICATION AND SOFWTEWARE ENGINEERING

MCN7110 INTERNET OF THINGS

FULL NAME:	**NSENGIYUMVA WILBERFORCE**
REGISTRATION NUMBER:	**2024/HD05/22078U**
STUDENT NUMBER:	**240022078**
LECTURER:	**DR. MARRIETTE KATARAHWEIRE**
GITHUB	**IMPLEMENTATION CODE FOR ABSTRACTIONS**

Question One

(a) The (linear-model xs y) function demonstrates multiple layers of data and procedural abstraction. At the data level, the input lists xs and y represent the independent and dependent variable values, abstracting the raw numerical data into an organized format. These lists are then transformed into structured matrix representations using functions like list*->matrix and ->col-matrix, which abstract the underlying complexity of creating matrices and allow the data to be prepared for mathematical computations. The matrix X includes a column of ones for the intercept term and the independent variable values, while Y is constructed as a column matrix representing the dependent variable, ensuring the data adheres to the standard form required for linear regression. On the procedural abstraction level, the function encapsulates the mathematical operations needed to compute the regression coefficients in a clear and modular manner. The formula A=((XTX)−1)XTYA = ((X^T X)^-1) X^T Y, where XTX^T represents the transpose and (XTX)−1(X^T X)^-1 represents the inverse of a matrix, is implemented step by step using abstractions for matrix transposition, multiplication, and inversion. Each of these operations is handled by dedicated procedures, ensuring the logic is both reusable and independent of the specific details of matrix algebra. Finally, the result is converted back from a matrix to a list using matrix->list, abstracting the presentation of the computed regression coefficients as a simple list for user consumption. This layered abstraction simplifies the implementation, making the code more readable, reusable, and focused on the high-level task of computing a linear regression model.

(b) The (list->sentiment lst #:lexicon [lexicon 'nrc]) function operates at multiple levels of data and procedural abstraction. At the data level, the input lst is a list of words and their associated frequencies, representing raw textual data alongside a lexicon argument that defaults to 'nrc', which abstracts the choice of sentiment lexicon to be used in analysis. This input is transformed into a structured format by the nested pack-sentiment function, which iterates through the input list and associates each word with its sentiment classification and frequency by calling token->sentiment. Here, the sentiment lexicon abstraction hides the internal complexity of how words are classified into sentiments, allowing the program to focus on high-level sentiment association. Procedural abstraction is demonstrated in the design of pack-sentiment, which maps each word and frequency pair in the input list to a structured format consisting of "word," "sentiment," and "frequency," appending them into a unified data structure. The use of map and nested let statements modularizes the steps, making the logic reusable and easy to follow. The outer function ensures that the processed sentiment data is returned only if meaningful results exist (i.e., the sentiment list has more than one element), providing an additional abstraction that simplifies error handling and ensures the output is well-structured and relevant to the analysis. By organizing data into a human-readable format and encapsulating operations for token analysis and result packaging, the function achieves a clear separation of concerns, making the code modular, extensible, and easy to integrate into broader applications for sentiment analysis.

(c) The (read-csv file-path #:->number? [->number? #f] #:header? [header? #t]) function demonstrates layered data and procedural abstraction to handle the reading and parsing of CSV files efficiently. At the data level, the file-path argument abstracts the location of the CSV file, while the optional keyword arguments ->number? and header? add flexibility by allowing users to specify whether to convert values to numbers and whether the CSV contains a header row, respectively. The internal csv-reader abstraction, created using make-csv-reader-maker, hides the complexity of handling CSV-specific parsing details, such as skipping lines with comments (#). The data is read and initially converted into a list of rows using csv->list, abstracting the file-handling and line-reading processes into a single call. At the procedural abstraction level, the function modularizes the logic for handling numbers and headers. If ->number? is true, it attempts to convert all data elements to numbers using string->number, but if header? is also true, it ensures the header row is left as strings by splitting the processing into separate steps for the header and data rows. If ->number? is false, the function leaves all elements as strings, demonstrating abstraction in preserving data integrity based on user preference. The use of nested lambdas and higher-order functions like map encapsulates the row-wise and element-wise transformations, ensuring that each aspect of data processing is reusable and isolated from the lower-level details of reading and parsing. This layered abstraction enables the function to provide a robust and flexible interface for CSV file handling, making it adaptable for diverse datasets while keeping the implementation concise and user-focused.

(d) The (qq-plot* lst #:scale? [scale? #t]) function demonstrates data and procedural abstraction in the context of creating a quantile-quantile (Q-Q) plot. At the data level, the input lst represents the sample data for which the Q-Q plot is to be generated. The optional keyword argument scale? allows users to specify whether the plot should scale the sample data or not, providing an abstraction that tailors the plot to different needs without modifying the core logic. Procedurally, the function encapsulates the process of plotting a Q-Q plot within a higher-level function. The internal call to (qq-plot lst #:scale? scale?) handles the actual computation and creation of the plot, abstracting away the details of how the quantiles are calculated and visualized. The plot function further abstracts the rendering process, taking the results from qq-plot and adding additional customization, such as axis labels (#:x-label and #:y-label). This separation allows the user to focus on specifying the input data and desired plot characteristics (like scaling) without worrying about the intricacies of the plotting process. The overall effect is a clean, reusable interface for generating Q-Q plots, encapsulating both the statistical computation and graphical presentation.

(e) The (hist lst) function showcases both data and procedural abstraction in its simplicity and functionality. At the data level, lst represents the raw input data, which could be any collection of values (such as a list of numbers). The function abstracts the complexity of processing this data by passing it to the sorted-counts function, which likely handles the counting and sorting of the elements in lst. This transforms the raw data into a frequency distribution, ready for visualization. On the procedural abstraction side, the function uses discrete-histogram, which abstracts the actual process of creating a histogram from the sorted counts. The details of how the histogram is constructed and displayed are hidden, allowing the user to focus on providing the data (lst) without worrying about the underlying implementation of the histogram creation. By combining these abstractions, the function offers a high-level interface that makes generating a histogram easy and intuitive, while the internal workings (counting, sorting, and plotting) are neatly encapsulated.

Question 2

a) json-lines->json-array

The json-lines->json-array function is designed to read a JSON file containing multiple lines of tweet data and convert each line into a structured JSON record, which is then stored in an array. This function uses a loop to read the file line by line, allowing it to process the data incrementally. The motivation behind this procedure is to manage JSON data efficiently, especially when working with larger datasets that are too big to fit into memory at once. By processing each line individually and converting it into an array, the program can handle large-scale datasets more easily, making it suitable for small datasets as well. This abstraction simplifies the overall file reading process and structures the data in a way that makes it easier to manipulate and analyze later.

b) process-text

The process-text function is responsible for cleaning and preprocessing the tweet text by removing punctuation, URLs, and converting the text to lowercase. This function is essential because tweet texts can be noisy, containing unnecessary characters such as links, punctuation marks, and inconsistent capitalizations that could interfere with further analysis. By normalizing the text to a consistent format, the function ensures that only meaningful text remains for subsequent steps, such as tokenization and sentiment analysis. The motivation behind this function is to standardize the text data, making it easier to perform reliable text analysis, such as sentiment analysis, without being misled by extraneous elements.

c) tokenize-and-remove-stopwords

The tokenize-and-remove-stopwords function breaks down the processed tweet text into individual words (tokens) and removes common stopwords, such as "the," "is," and "in," that do not contribute significant meaning in sentiment analysis. Tokenization is a crucial step because it allows the program to analyze the individual components of the tweet, such as keywords and phrases, instead of working with entire sentences. Removing stopwords further refines the data, ensuring that only relevant words are considered when performing tasks like sentiment analysis. The motivation for this procedure is to focus the analysis on the meaningful content of the tweet, improving the quality and accuracy of the subsequent sentiment classification process by eliminating irrelevant terms.

d) analyze-sentiment

The analyze-sentiment function takes a list of tokens (words) and uses the NRC lexicon to classify their sentiment into categories like positive, negative, or neutral. Sentiment analysis is a core part of the program's goal to understand the emotional tone behind tweets from different countries. By using the NRC lexicon, the program leverages a predefined set of emotional labels to evaluate each word's sentiment, which is then applied to the entire tweet. The motivation behind this procedure is to automatically assess the mood or sentiment of tweets without requiring manual labeling, which is time-consuming and prone to human bias. This function abstracts the sentiment analysis process, making it a repeatable and automated step that can be applied to all tweets, regardless of the country.

e) aggregate-sentiment-counts

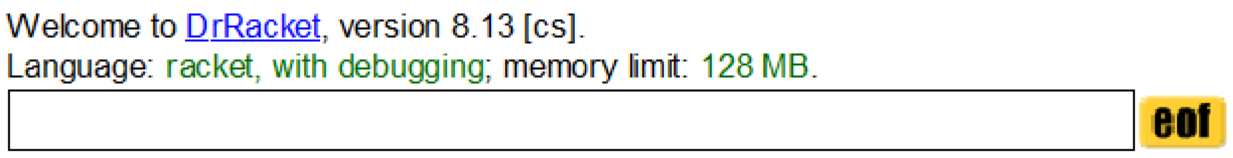
The aggregate-sentiment-counts function is used to summarize the sentiment data by counting the frequency of each sentiment label (such as positive, negative, or neutral). It aggregates the sentiment analysis results by summing the frequencies of sentiment labels for each country's tweet dataset. This function is important because it consolidates the results of sentiment analysis into a more digestible format, enabling easier interpretation and comparison of the data. The motivation behind this procedure is to provide an overview of how tweets from different countries are emotionally perceived, allowing for further analysis and comparison between countries. Aggregating the sentiment counts makes it easier to visualize trends and draw conclusions from the data.

f) plot-sentiment-analysis

The plot-sentiment-analysis function is responsible for creating a visual representation of the aggregated sentiment data. It generates a discrete histogram, which displays the frequency of each sentiment label for a specified country's tweets. The function takes the aggregated sentiment counts and plots them on a graph, allowing users to visually interpret the emotional tone of the tweets. The motivation behind this function is to provide an intuitive and accessible way to analyze sentiment data. By visualizing the sentiment counts, users can quickly grasp patterns, such as which sentiment dominates in a particular country's tweets, and compare sentiment across different countries. This visual abstraction allows for more insightful, at-a-glance analysis and can help identify trends or anomalies in the data.

Outputs of the system

When the program is executed, it shows an input where the user can put the name of the country that they want to view the moods. The countries are either Uganda, Egypt or Kuwait



```
Welcome to DrRacket, version 8.13 [cs].
Language: racket, with debugging; memory limit: 128 MB.
```

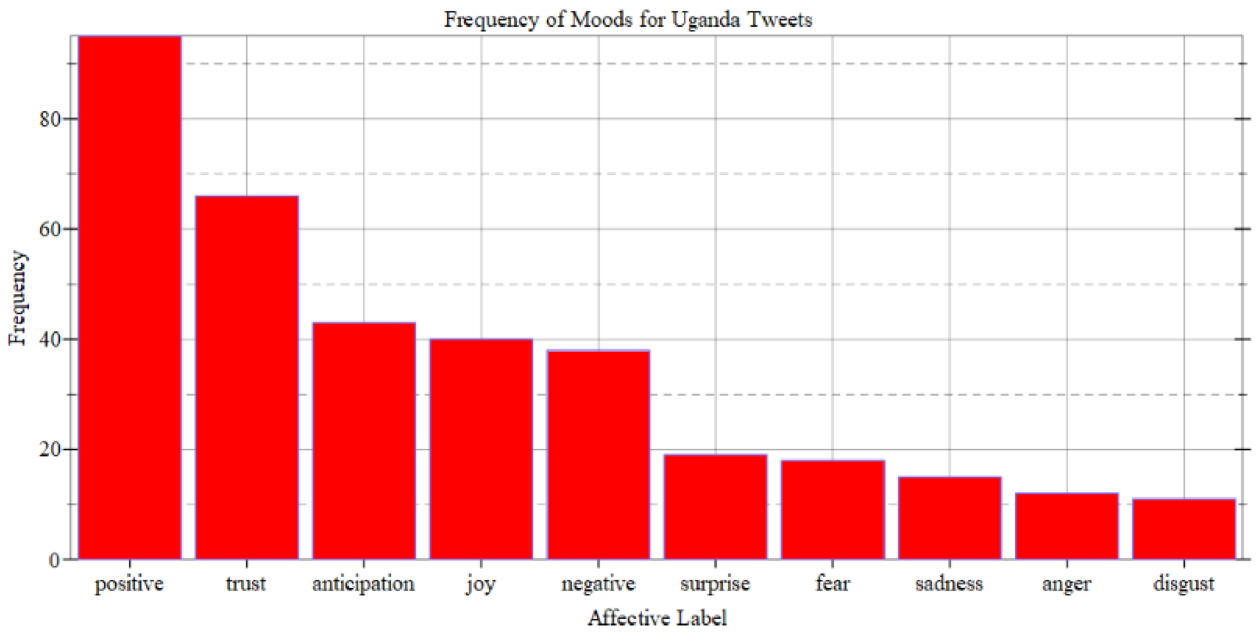


Figure 1-1When the user inputs Uganda

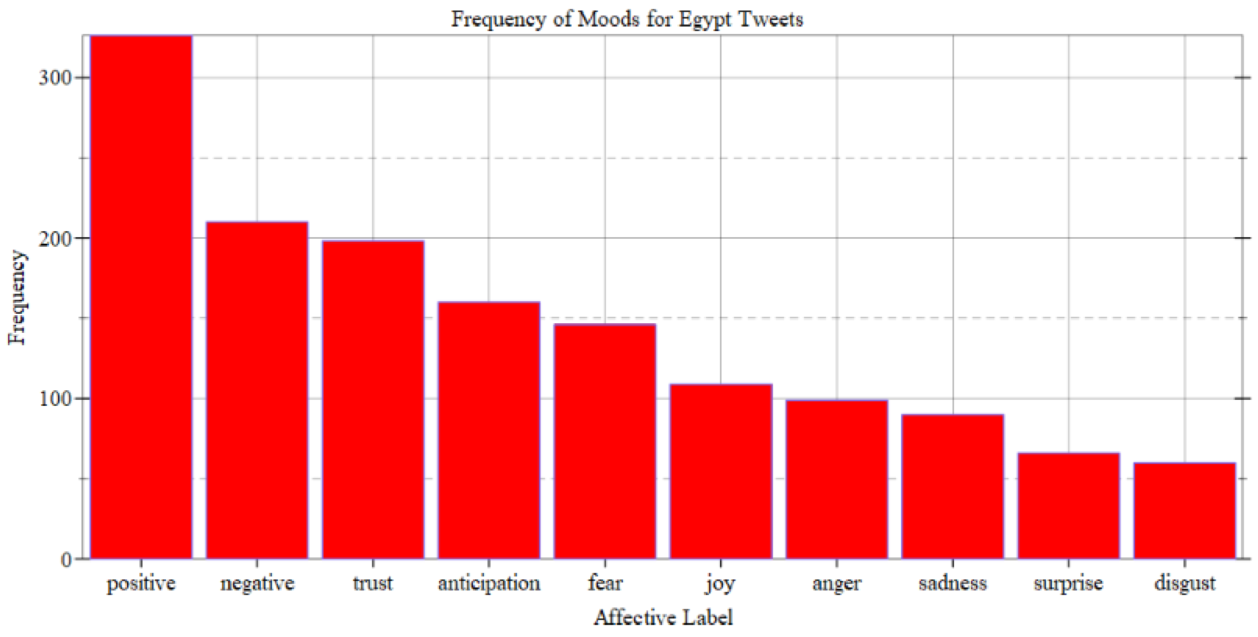


Figure 2-2 When the user inputs Egypt