MAKERERE UNIVERSITY

SEMESTER ONE 2024/2025 ACADEMIC YEAR

SCHOOL COMPUTING AND INFORMATICS TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

MASTER OF SCIENCE IN COMPUTER SCIENCE

MCN 7105

Structure & Interpretation of computer programs

Final Assignment

NDIBALEKERA DIANA RHODA

2024/HD05/21951U

2400721951

Question 1

Analysis of Data Science Abstractions

The functions analyzed include those for linear regression, sentiment analysis, reading CSV data, quantile-quantile plotting, and histogram generation. Each function is studied in terms of its data and procedure abstraction layers, describing how data is represented, processed, and output.

1. Linear Model

The linear-model function performs linear regression using matrix operations, specifically solving for the model parameters (coefficients) via the normal equation. The linear model function extends this by adding additional model diagnostics such as residuals, mean squared error (MSE), and root MSE.

Data Abstraction: The model is represented by coefficients (intercept and slopes) calculated from the input data. Data is stored in matrices for both the input variables and model parameters.

Procedure Abstraction: linear model solves for the regression coefficients using matrix inversions and multiplications. It returns the model's coefficients as a list. linear model extends this by calculating residuals, MSE, and root MSE and returns the results in a hash.

2. Sentiment Analysis (list->sentiment)

The list->sentiment function analyzes a list of tokens (words) using a predefined lexicon (NRC, Bing, or AFINN) and returns sentiment scores or labels.

Data Abstraction: Input is a list of tokens (words), and output is a sentiment score or label for each token. Sentiment results are stored as either labels or numerical values depending on the chosen lexicon.

Procedure Abstraction: The list->sentiment function computes the sentiment of each token using the token->sentiment function and returns the aggregated sentiment results for the list.

3. CSV Reading (read-csv)

The read-csv function reads a CSV file and converts its content into a list of lists, with each list representing a row. It also converts values to either strings or numbers based on the specified parameters.

Data Abstraction: Data is represented as a list of lists, where each inner list corresponds to a row of the CSV. It is then stored in either string or numeric form, based on the ->number? parameter.

Procedure Abstraction: The read-csv function reads and processes the file content into a list of lists, supporting optional headers and comments.

4. Quantile-Quantile Plot (qq-plot)

The qq-plot function compares the quantiles of a sample dataset to those of a normal distribution. It visually assesses whether the sample follows a normal distribution.

Data Abstraction: The sample data and the theoretical normal distribution quantiles are stored as separate lists.

Procedure Abstraction: The qq-plot function computes and compares quantiles of the sample with those of a normal distribution, generating a plot for visual comparison.

5. Histogram (hist)

The hist function generates a discrete histogram of numerical values by grouping them into bins and counting their frequency.

Data Abstraction: Data is represented as a list of numerical values, and the histogram is represented as a list of bin-count pairs.

Procedure Abstraction: The hist function sorts the data, creates bins, and counts the frequency of values in each bin. The result is a histogram represented as (bin, count) pairs.

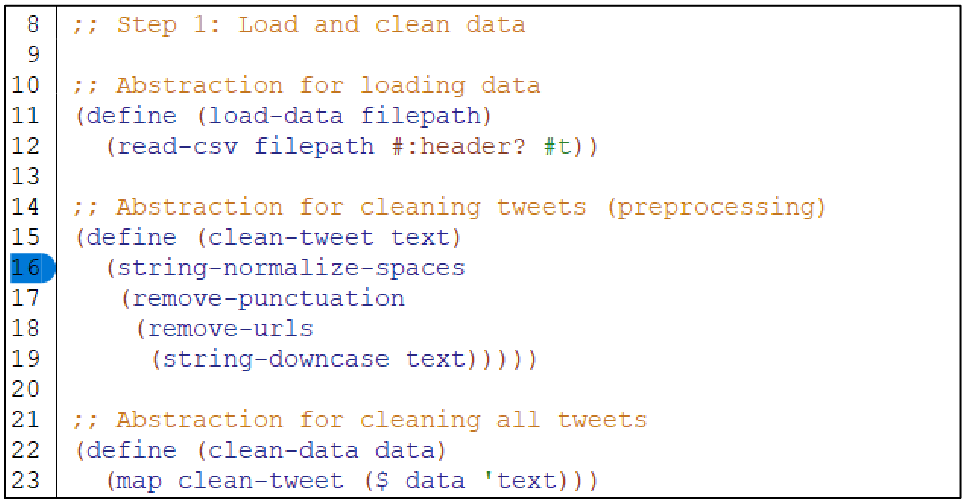
Question 2

This required to build a set of abstractions that can be used by developers to build software systems that analyze the moods of tweets for a given country (e.g., Uganda) over a period of 12 months. The dataset used was extracted from Kaggle and through it we only selected Uganda as a country and saved our new dataset in a CSV. This dataset only included data for 3 months, i.e., July, September, and October.

Data abstractions created included data loading and cleaning, sentiment analysis, aggregation of sentiment data and visualization of sentiment distribution.

Data Loading and Cleaning

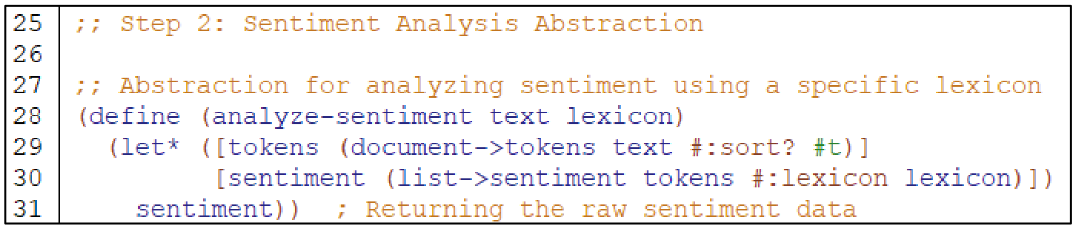
The code reads tweet data from a CSV file and applies necessary cleaning steps to normalize spaces, remove punctuation, and eliminate URLs.   This preprocessing ensures that the data is in a usable format for sentiment analysis.



```
;; Step 1: Load and clean data

;; Abstraction for loading data
(define (load-data filepath)
  (read-csv filepath #:header? #t))

;; Abstraction for cleaning tweets (preprocessing)
(define (clean-tweet text)
  (string-normalize-spaces
   (remove-punctuation
    (remove-urls
     (string-downcase text)))))

;; Abstraction for cleaning all tweets
(define (clean-data data)
  (map clean-tweet ($ data 'text)))
```

Sentiment analysis

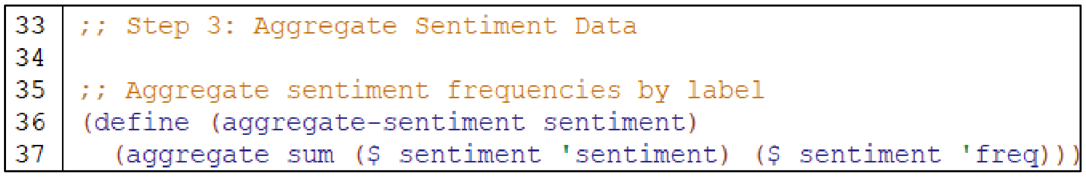
The `analyze-sentiment` function leverages a sentiment lexicon (e.g., NRC) to analyze the sentiment of each token (word) in the tweets.   This analysis provides sentiment labels or scores for each word in the tweets.



```
;; Step 2: Sentiment Analysis Abstraction

;; Abstraction for analyzing sentiment using a specific lexicon
(define (analyze-sentiment text lexicon)
  (let* ([tokens (document->tokens text #:sort? #t)]
         [sentiment (list->sentiment tokens #:lexicon lexicon)])
    sentiment))  ; Returning the raw sentiment data
```

Aggregate Sentiment Data

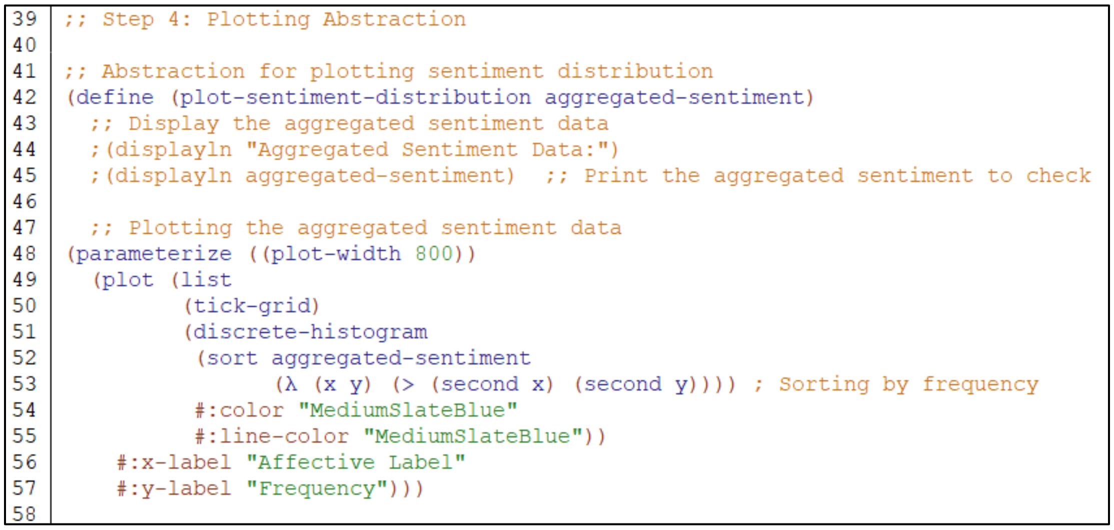
Sentiment data is aggregated by summing the frequencies of sentiment labels providing an overall sentiment distribution for the dataset.



```
;; Step 3: Aggregate Sentiment Data

;; Aggregate sentiment frequencies by label
(define (aggregate-sentiment sentiment)
  (aggregate sum ($ sentiment 'sentiment) ($ sentiment 'freq)))
```

Visualization of sentiment distribution

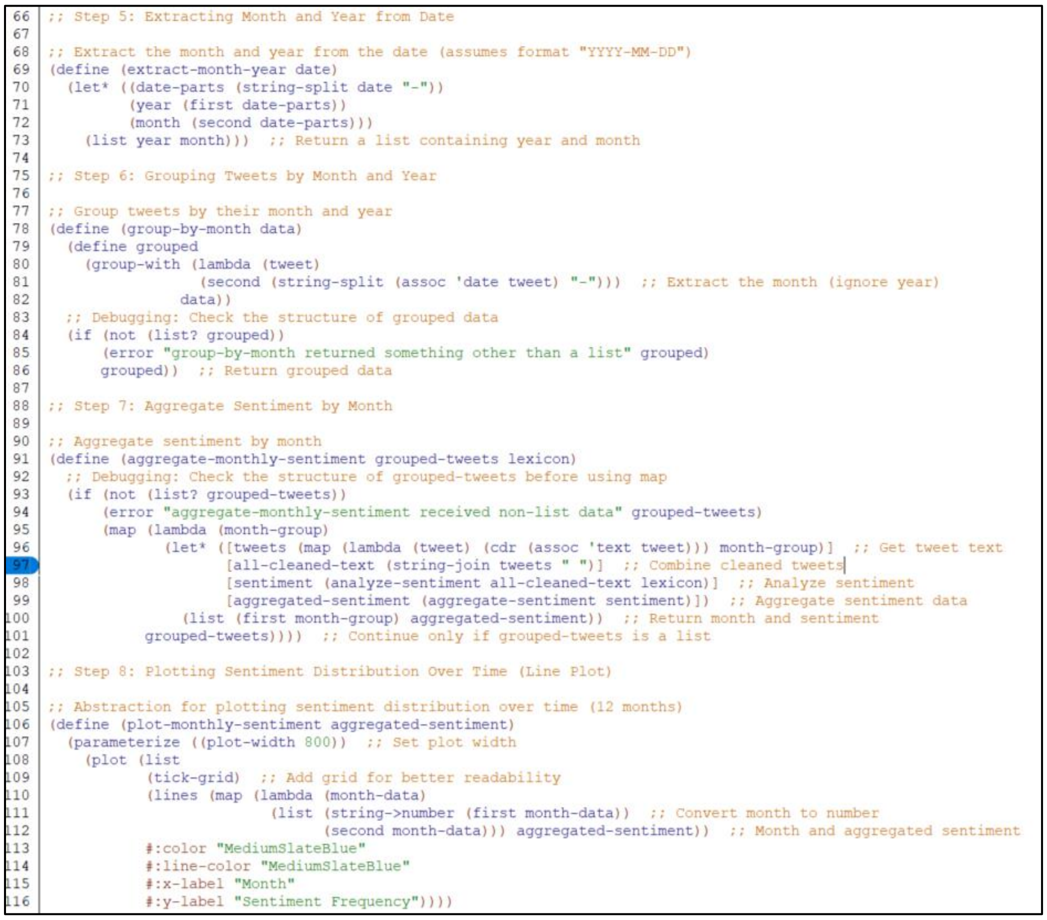
Visualization of the sentiment distribution is done using a discrete histogram, displaying the frequency of each sentiment label.



```
;; Step 4: Plotting Abstraction

;; Abstraction for plotting sentiment distribution
(define (plot-sentiment-distribution aggregated-sentiment)
  ;; Display the aggregated sentiment data
  ;(displayln "Aggregated Sentiment Data:")
  ;(displayln aggregated-sentiment)  ;; Print the aggregated sentiment to check

  ;; Plotting the aggregated sentiment data
(parameterize ((plot-width 800))
  (plot (list
         (tick-grid)
         (discrete-histogram
          (sort aggregated-sentiment
                (λ (x y) (> (second x) (second y)))) ; Sorting by frequency
          #:color "MediumSlateBlue"
          #:line-color "MediumSlateBlue"))
     #:x-label "Affective Label"
     #:y-label "Frequency")))

```

Grouping Tweets by Month

The group-by-month function is designed to group tweets based on the month they were posted, without considering the year. This allows for the analysis of sentiment trends across months, irrespective of the year.



```
;; Step 5: Extracting Month and Year from Date

;; Extract the month and year from the date (assumes format "YYYY-MM-DD")
(define (extract-month-year date)
  (let* ((date-parts (string-split date "-"))
         (year (first date-parts))
         (month (second date-parts)))
    (list year month)))  ;; Return a list containing year and month

;; Step 6: Grouping Tweets by Month and Year

;; Group tweets by their month and year
(define (group-by-month data)
  (define grouped
    (group-with (lambda (tweet)
                  (second (string-split (assoc 'date tweet) "-")))  ;; Extract the month (ignore year)
                data))
  ;; Debugging: Check the structure of grouped data
  (if (not (list? grouped))
      (error "group-by-month returned something other than a list" grouped)
      grouped))  ;; Return grouped data

;; Step 7: Aggregate Sentiment by Month

;; Aggregate sentiment by month
(define (aggregate-monthly-sentiment grouped-tweets lexicon)
  ;; Debugging: Check the structure of grouped-tweets before using map
  (if (not (list? grouped-tweets))
      (error "aggregate-monthly-sentiment received non-list data" grouped-tweets)
      (map (lambda (month-group)
             (let* ([tweets (map (lambda (tweet) (cdr (assoc 'text tweet))) month-group)]  ;; Get tweet text
                    [all-cleaned-text (string-join tweets " ")]  ;; Combine cleaned tweets
                    [sentiment (analyze-sentiment all-cleaned-text lexicon)]  ;; Analyze sentiment
                    [aggregated-sentiment (aggregate-sentiment sentiment)])  ;; Aggregate sentiment data
               (list (first month-group) aggregated-sentiment))  ;; Return month and sentiment
           grouped-tweets))))  ;; Continue only if grouped-tweets is a list

;; Step 8: Plotting Sentiment Distribution Over Time (Line Plot)

;; Abstraction for plotting sentiment distribution over time (12 months)
(define (plot-monthly-sentiment aggregated-sentiment)
  (parameterize ((plot-width 800))  ;; Set plot width
    (plot (list
           (tick-grid)  ;; Add grid for better readability
           (lines (map (lambda (month-data)
                         (list (string->number (first month-data))  ;; Convert month to number
                               (second month-data))) aggregated-sentiment))  ;; Month and aggregated sentiment
           #:color "MediumSlateBlue"
           #:line-color "MediumSlateBlue"
           #:x-label "Month"
           #:y-label "Sentiment Frequency"))))
```

Design decision drivers

Data Structure Choice: Chose alist (association lists) to store tweet data because they provide an efficient way to pair the date and text fields. Each tweet is represented as a key-value pair, which is easy to access and manipulate for further processing.

Data Cleaning: Preprocessing steps like lowercasing, punctuation removal, and URL stripping were chosen to standardize the text. These steps ensure that irrelevant information (like URLs) is removed, and inconsistencies (such as case sensitivity or extra spaces) are resolved, leading to more accurate sentiment analysis.

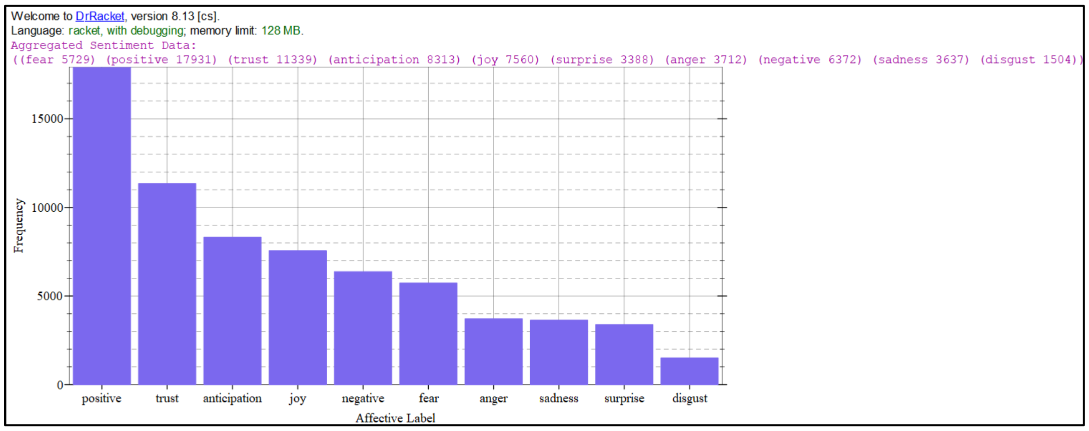
Sentiment Analysis Lexicon: Used the NRC lexicon for sentiment analysis, as it provides a detailed set of emotional labels (e.g., joy, anger, sadness), which allow for a more nuanced understanding of sentiment compared to simpler positive/negative categorizations. This is especially useful for analyzing complex, diverse social media content like tweets.

Grouping by Month: Tweets were grouped by month and year to analyze sentiment over time. This decision allows for the observation of trends and patterns in sentiment across a 12-month period, providing a temporal context that's useful for understanding how sentiment changes with seasons or specific events.

Sentiment Aggregation: Aggregation of sentiment frequencies by month helps summarize the overall sentiment for each period, making the data easier to interpret. This decision simplifies analysis and enables the creation of clear visualizations of sentiment distribution.

Visualization: Used a histogram to visualize sentiment distribution because it effectively represents the frequency of different sentiment categories (positive, negative, neutral) and allows for easy comparison of sentiment across different months. This type of visualization helps identify dominant sentiment trends over time.

And below are the results from the above



```
Welcome to DrRacket, version 8.13 [cs].
Language: racket, with debugging; memory limit: 128 MB.
Aggregated Sentiment Data:
((fear 5729) (positive 17931) (trust 11339) (anticipation 8313) (joy 7560) (surprise 3388) (anger 3712) (negative 6372) (sadness 3637) (disgust 1504))
```