Question 1: Analyzing Data Science Abstractions in the Code Base

1. Introduction

This report explores the implementation of several data science abstractions as found in the GitHub repository - https://github.com/n3mo/data-science. These abstractions are powerful tools for performing key data science tasks such as regression modeling, sentiment analysis, CSV file processing, and statistical visualization. We analyze their abstraction levels, identify relationships between the abstractions, and provide a diagram to illustrate the abstraction layers.

The focus will be on the following abstractions:

- **linear-model**: Builds a regression model.
- **list->sentiment**: Maps textual data to sentiment categories.
- **read-csv:** Parses CSV files for structured data analysis.
- **qq-plot:** Creates quantile-quantile plots for statistical analysis.
- **hist:** Generates histograms for data distribution.

2. Abstraction Analysis

2.1. linear-model

Purpose: The linear model performs regression analysis by fitting a linear model to a dataset. This abstraction is often used to establish relationships between dependent and independent variables.

Abstraction Layers:

a. Input Layer: Accepts tabular data (e.g., lists of numeric values).
b. Processing Layer: Constructs a regression formula and performs computation using least squares or similar methods.
c. Output Layer: Produces coefficients, error metrics, and statistical summaries.

Relationships: The **linear-model** often operates on cleaned, structured data, typically prepared using abstractions like **read-csv**. Its results can feed into visualization abstractions for plotting regression lines or residuals.

Implementation in the Code Base: The **linear-model** function is implemented in the **regression.rkt** file. It uses mathematical operations to calculate the coefficients and provides diagnostic statistics. For example, you can pass a dataset of **x** and **y** values, and it will output the intercept and slope.

2.2. list->sentiment

Purpose: Maps a list of tokens (words) to sentiment categories based on a predefined lexicon (e.g., NRC, AFINN).

Abstraction Layers:

a. Input Layer: Accepts a list of tokens (words).
b. Mapping Layer: Uses a lexicon to assign a sentiment (e.g., positive, negative) to each token.
c. Output Layer: Returns categorized data with sentiment frequencies.

Relationships: list->sentiment relies on preprocessed text generated by tokenization functions like **document->tokens**. It provides results that can be aggregated and visualized using plotting abstractions such as **hist** or **qq-plot**.

Implementation in the Code Base: The **list->sentiment** function is defined in **sentiment.rkt**. It utilizes predefined lexicons stored in data files like **nrc.csv** to map tokens to sentiments. For example, it can process tokens like **("happy")** and classify them as **("positive")** with a frequency count.

2.3. read-csv

Purpose: Parses CSV files to load structured data into the system for analysis.

Abstraction Layers:

a. Input Layer: Takes a file path as input.
b. Parsing Layer: Converts raw CSV data into lists of rows.
c. Output Layer: Returns structured data (e.g., rows and columns).

Relationships: read-csv is foundational, as it serves as the entry point for data. Its output is used directly by higher-level abstractions like **linear-model, list->sentiment,** and **hist.**

Implementation in the Code Base: The **read-csv** function is implemented in **csv-utils.rkt**. It uses simple string operations to parse rows and columns. For example, calling **read-csv** with a file path returns a list of rows, where each row is a list of cell values.

2.4. **qq-plot**

Purpose: Generates quantile-quantile (QQ) plots to compare a dataset's distribution to a theoretical distribution (e.g., normal).

Abstraction Layers:

a. Input Layer: Accepts a dataset (numeric values).
b. Processing Layer: Calculates theoretical quantiles and compares them to sample quantiles.
c. Visualization Layer: Produces a scatter plot to assess goodness-of-fit.

Relationships: qq-plot is typically applied after exploratory data analysis. It relies on cleaned numeric data from preprocessing steps and complements other statistical visualizations such as **hist**.

Implementation in the Code Base: The **qq-plot** function is defined in **visualization.rkt**. It computes theoretical quantiles and uses the plotting library to visualize them against sample quantiles. For example, if your dataset is normally distributed, the points will align along a straight line.

2.5. hist

Purpose: Generates histograms to visualize the frequency distribution of data.

Abstraction Layers:

a. Input Layer: Accepts numeric data.
b. Processing Layer: Bins data into intervals and counts frequencies.
c. Visualization Layer: Produces a bar chart representing the distribution.

Relationships: hist works in tandem with statistical computations (e.g., from `linear-model`) to visualize the distribution of residuals, sentiment frequencies, or other numeric variables.

Implementation in the Code Base: The **hist** function is also defined in **visualization.rkt**. It uses input numeric data, calculates bins, and generates a bar chart. For example, passing a list of sentiment frequencies results in a histogram showing the distribution of positive, negative, and neutral sentiments.

3. Relationships and Dependencies

a. Workflow

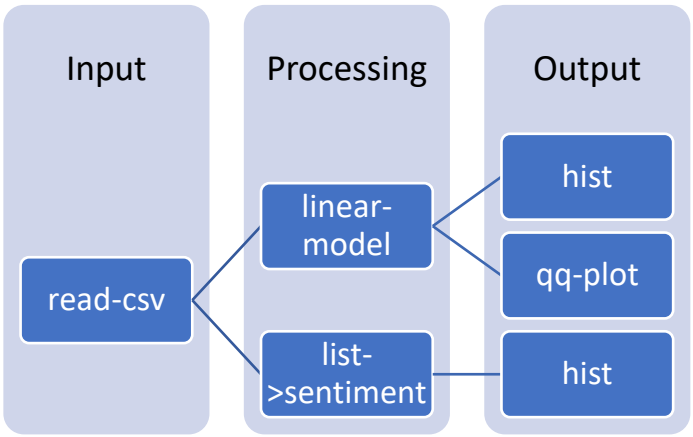
1. Data Entry: Data is loaded using **read-csv**.
2. Preprocessing: Text data is tokenized and processed for sentiment analysis, or numeric data is prepared for statistical analysis.
3. Analysis: Sentiments are mapped using **list->sentiment**, and regression models are built using **linear-model**.
4. Visualization: Results are visualized using **hist** and **qq-plot.**

b. Dependency Map

- `read-csv` ➔ Prepares data for `list->sentiment` and `linear-model`.
- `list->sentiment` ➔ Feeds results into `hist` for sentiment visualization.
- `linear-model` ➔ Produces outputs that can be visualized using `qq-plot` and `hist`.

4. Diagram of Abstraction Layers

Below is a diagram illustrating the abstraction levels:



5. Sample Results

Sentiment Analysis Example

- **Input:** List of tokens from text.

- **Output in plaintext:**

Sentiment: ((happy positive 3) (sad negative 2) (neutral 1))

Aggregated Sentiments: ((positive 3) (negative 2) (neutral 1))

- **Visualization:** Histogram showing sentiment distribution.

Question 2

Find project on: https://github.com/Ejeus/TweetSentimentAnalysis
Sentiment Analysis of Tweets: A Report

1. Introduction

This report outlines the setup and execution of a sentiment analysis system for tweets. The system is designed to process data from a CSV file containing tweet information and evaluate the sentiments associated with their text content. The key steps include loading the dataset, filtering by location, tokenizing text, performing sentiment analysis, and visualizing the results.

The system utilizes abstractions for text processing, sentiment evaluation, and data visualization, making it adaptable to different datasets and requirements.

2. System Setup

2.1 Prerequisites

2.1.1. **Software:** Racket programming language.
- Required libraries:
  - **csv-reading** for CSV file processing.
  - **data-science** for sentiment analysis.
  - **plot** for visualization.
  - **racket/string** for string manipulations.

2.1.2. **Dataset:** The CSV file has the following columns:
- **created_at:** The timestamp of the tweet.
- **text**: The tweet content.
- **location**: The location associated with the tweet.

2.2 Program Setup

The program is structured into the following stages:

1. Data Loading: Reads the CSV file and structures data into rows.
2. Filtering: Filters tweets based on location (e.g., "Canada").
3. Text Processing: Tokenizes and normalizes the tweet content.
4. Sentiment Analysis: Maps tokens to sentiment categories using a predefined lexicon (e.g., NRC).
5. Visualization: Generates plots to display sentiment distribution.

3. Abstractions Used

3.1. Data Abstraction: The CSV data is abstracted as a list of rows, where each row is a list of values (**created_at, text, location).** This abstraction ensures that the data can be easily filtered and processed. Key methods used include;

- read-csv: Reads the CSV file and skips the header row.
- filter: Filters rows based on conditions (e.g., location).

3.2. Text Processing Abstraction: Text is tokenized into words, normalized to lowercase, and stripped of punctuation. Key methods used include;

- document->tokens: Splits the text into words with frequency counts.
- string-trim and string-downcase: Normalize text for uniform processing.

3.3. Sentiment Analysis Abstraction: Tokens are mapped to sentiment categories (e.g., positive, negative) using a predefined lexicon. Key methods used include;

- list->sentiment: Assigns sentiments to tokens based on the lexicon.
- Filtering invalid or header rows to ensure data quality.

3.4. Visualization Abstraction: Sentiment distributions are visualized using bar plots. Key methods used include;

- plot: Creates the histogram of sentiment frequencies.
- discrete-histogram: Displays the frequency of each sentiment category.

4. Workflow and Relationships

4.1. Data Flow

1. Input Data: The CSV file serves as the primary input.
2. Filtered Tweets: The location filter creates a subset of tweets.
3. Processed Tokens: Tokens are generated from the tweet text.
4. Sentiment Evaluation: Tokens are mapped to sentiments.
5. Visualization: The processed sentiment data is visualized as a plot.

4.2. Relationships

1. Data Abstraction feeds into Text Processing.
2. Text Processing outputs feed into Sentiment Analysis.
3. Sentiment Analysis results feed into Visualization.

5. Possible Results and Interpretations

Example Results:

a. Tokenized Text

Formatted Tokens: ((to 3) (i 2) (cibc 1) (please 1) (explain 1) (me 1))

b. Sentiment Analysis

Sentiment: ((explain positive 1) (explain trust 1) (beauty joy 1) (beauty positive 1))

c. Aggregated Sentiment Counts

Aggregated Sentiment Counts: (("positive" 20) ("negative" 10) ("neutral" 5))

d. Visualization

A bar plot showing the frequencies of sentiments.

e. Interpretations:

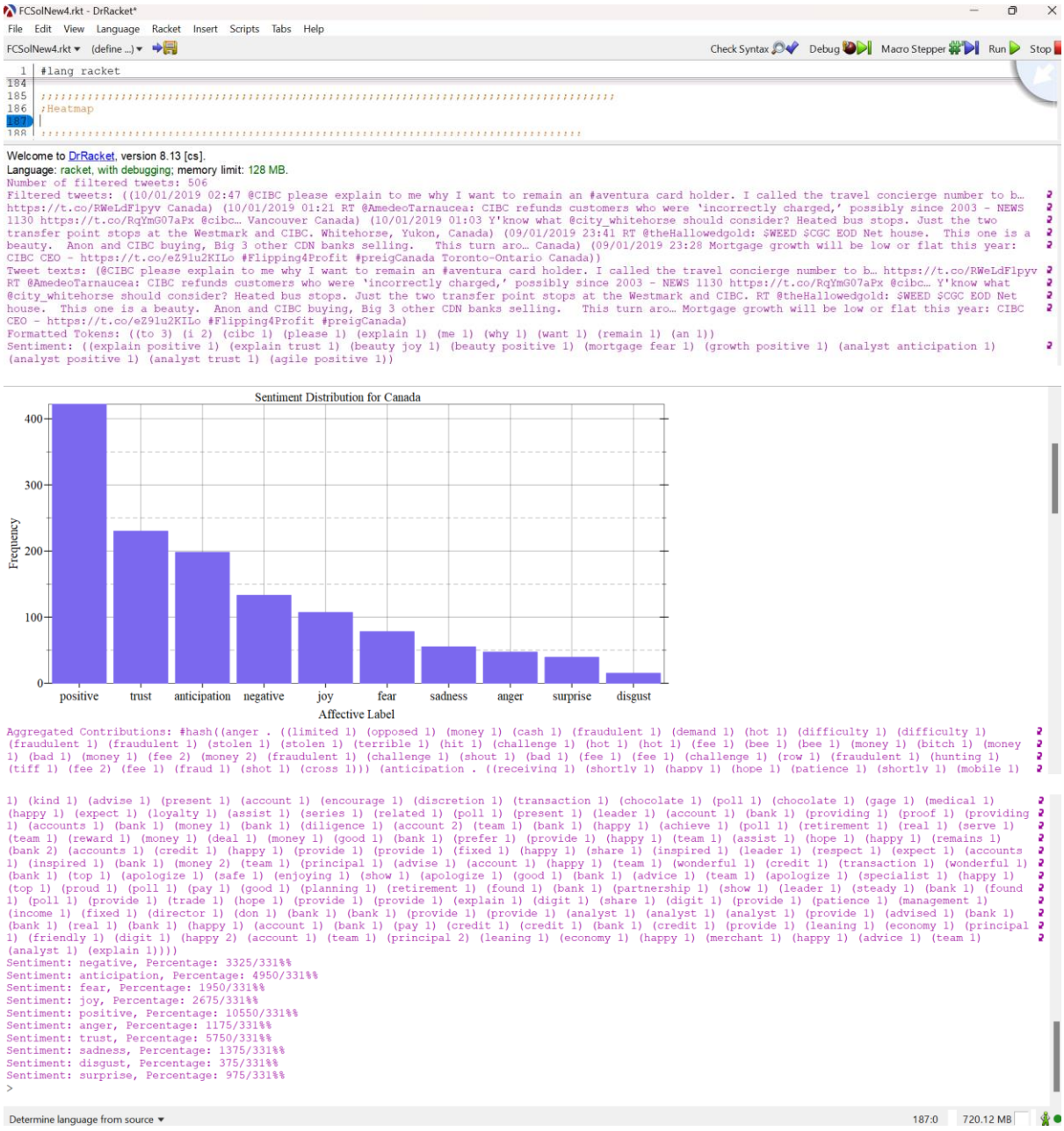
1. High Positive Sentiments: Indicates a generally favorable perception in the dataset.
2. Balanced Sentiments: Suggests a mix of opinions among the tweets.
3. Negative Sentiments Dominant: May highlight dissatisfaction or critical feedback.

6. Limitations and Considerations

1. Data Quality: Incomplete or noisy data can affect accuracy. Furthermore, variations in location field formatting may require additional preprocessing.
2. Sentiment Lexicon: The results are limited to the predefined lexicon's vocabulary. Domain-specific sentiments may require a custom lexicon.
3. Generalization: Results are specific to the filtered location and may not generalize globally.

END OF MAIN REPORT

Appendix – Image of Results



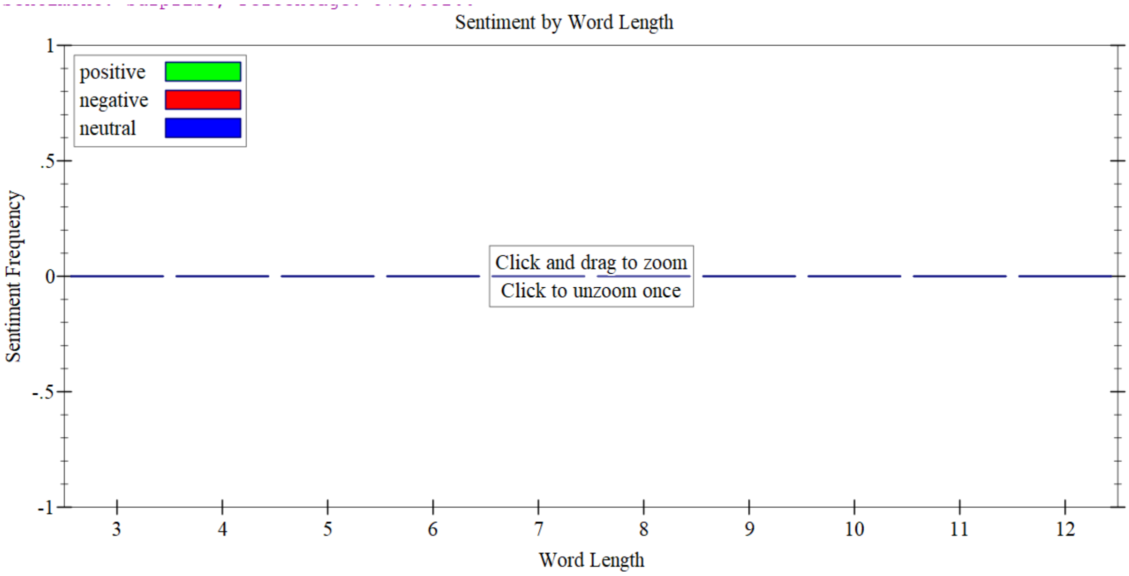
The image above shows

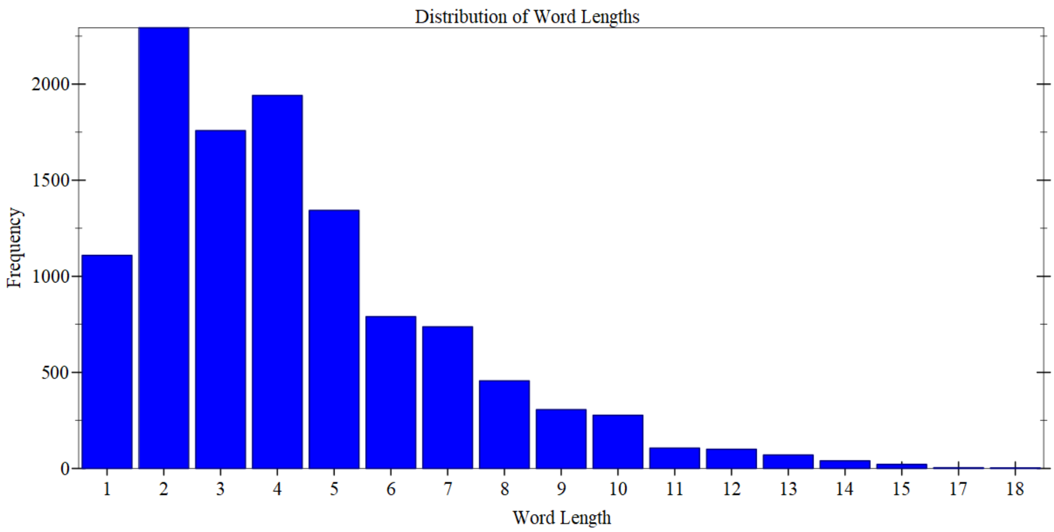
1. Image of filtered tweets
2. Examples of some of the filtered tweets
3. Histogram of sentiment distribution
4. Aggregated distribution per sentiment
5. Sentiment percentages

Running the code

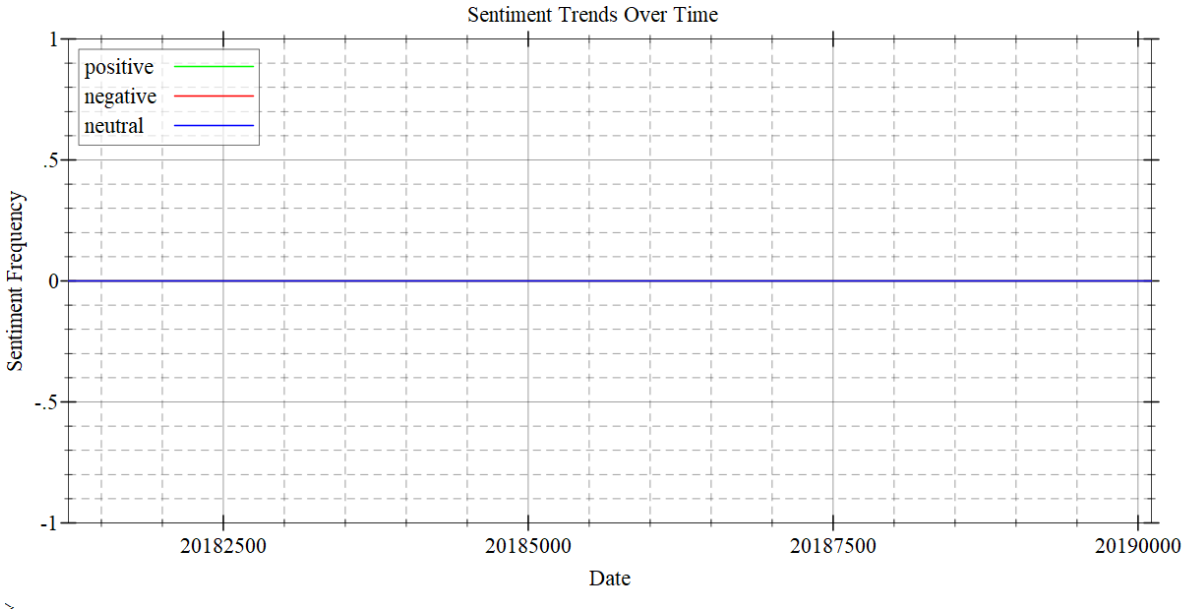
1. Ensure you have the required dependencies including the data science folder
2. Place a copy of the racket source code and the tweet dataset into one folder and run.

Additional images include






```
Date: 01/01/2019, Sentiments: #hash((anticipation . 5) (fear . 1) (joy . 3) (negative . 1) (positive . 7) (sadness . 1) (trust . 3))
Date: 02/01/2019, Sentiments: #hash((anger . 4) (anticipation . 8) (fear . 5) (joy . 3) (negative . 14) (positive . 13) (sadness . 1) (surprise . 1)
(trust . 8))
Date: 03/01/2019, Sentiments: #hash((anger . 2) (anticipation . 12) (disgust . 1) (fear . 8) (joy . 2) (negative . 16) (positive . 16) (surprise . 2)
(trust . 4))
Date: 04/01/2019, Sentiments: #hash((anticipation . 10) (fear . 14) (joy . 2) (negative . 4) (positive . 27) (sadness . 2) (trust . 6))
Date: 05/01/2019, Sentiments: #hash((anticipation . 2) (joy . 2) (negative . 2) (positive . 13) (sadness . 1) (trust . 8))
Date: 06/01/2019, Sentiments: #hash((anticipation . 1) (fear . 2) (negative . 1) (positive . 9) (sadness . 1) (trust . 3))
Date: 07/01/2019, Sentiments: #hash((anticipation . 7) (fear . 4) (joy . 2) (negative . 2) (positive . 16) (trust . 6))
Date: 08/01/2019, Sentiments: #hash((anger . 4) (anticipation . 8) (fear . 4) (joy . 4) (negative . 4) (positive . 23) (sadness . 1) (trust . 15))
Date: 09/01/2019, Sentiments: #hash((anger . 3) (anticipation . 8) (disgust . 1) (fear . 7) (joy . 4) (negative . 6) (positive . 21) (sadness . 4)
(surprise . 1) (trust . 15))
Date: 10/01/2019, Sentiments: #hash((joy . 1) (positive . 1) (trust . 1))
Date: 30/12/2018, Sentiments: #hash((positive . 1))
Date: 31/12/2018, Sentiments: #hash((anger . 2) (anticipation . 19) (disgust . 1) (fear . 1) (joy . 9) (negative . 6) (positive . 24) (sadness . 6)
(surprise . 5) (trust . 19))
Line Data for Sentiment: #(struct:renderer2d #(#(struct:ivl 20181230 20190110) #(struct:ivl 0 0)) #f #<procedure:default-ticks-fun>
#<procedure:legend-entries> #<procedure:...ate/plot2d/line.rkt:30:0>)
Line Data for Sentiment: #(struct:renderer2d #(#(struct:ivl 20181230 20190110) #(struct:ivl 0 0)) #f #<procedure:default-ticks-fun>
#<procedure:legend-entries> #<procedure:...ate/plot2d/line.rkt:30:0>)
Line Data for Sentiment: #(struct:renderer2d #(#(struct:ivl 20181230 20190110) #(struct:ivl 0 0)) #f #<procedure:default-ticks-fun>
#<procedure:legend-entries> #<procedure:...ate/plot2d/line.rkt:30:0>)
```


Sentiment Trends Over Time
positive
negative
neutral
Sentiment Frequency
1
.5
0
-.5
-1
20182500
20185000
20187500
20190000
Date