**WhatsApp Chat Analyzer Documentation**

**Table of Contents**

1. Introduction
2. Preprocessing
3. Streamlit App (**app.py**)
4. Helper Functions
5. Model Training
6. Conclusion

**1. Introduction**

The WhatsApp Chat Analyzer is a tool designed to analyze and visualize chat data from WhatsApp conversations. This documentation provides a comprehensive overview of the entire project, including preprocessing, the Streamlit app, helper functions, and model training for sentiment analysis.

**2. Preprocessing**

The preprocessing module contains the code for cleaning and structuring the raw chat data. The **preprocess** function extracts relevant information such as user messages, dates, and times, and creates a structured DataFrame. The function performs the following steps:

* Extracts date, time, and message using regular expressions.
* Converts dates to 24-hour format and creates DataFrame columns for various date components.
* Extracts user names and messages from the user messages.
* Generates additional columns for date components and message periods.
* Returns a cleaned and structured DataFrame.

**3. Streamlit App (app.py)**

The Streamlit app (**app.py**) is the user interface for the WhatsApp Chat Analyzer. It allows users to upload chat data, select specific users for analysis, and visualize various insights and statistics. The app consists of the following sections:

**Sidebar**

* Users can upload chat data and select a user for analysis.

**Statistics and Timeline**

* Displays metrics such as total messages, words, media, and links.
* Visualizes monthly and daily timelines of message activity.

**Activity Maps**

* Displays the busiest day and month based on message activity.
* Presents a heatmap showing activity throughout the week and day.

**Most Busy Users**

* Identifies and displays the most active users in the chat.

**Wordcloud and Most Common Words**

* Generates a word cloud and bar plot for common words.

**Emoji Analysis**

* Analyzes and presents emoji usage statistics.

**Sentiment Analysis**

* Performs sentiment analysis using both Keras and TextBlob models.
* Displays overall sentiment of the chat.

**4. Helper Functions**

The **helper** module likely contains functions that assist in generating visualizations and insights within the Streamlit app. These functions perform various tasks, such as calculating statistics and creating plots. The functions include:

**fetch\_stats(selected\_user, df)**

* Calculates and returns message-related statistics.
* Parameters: selected\_user (user for analysis), df (DataFrame).

**monthly\_timeline(selected\_user, df)**

* Creates a timeline plot of message distribution over months.
* Parameters: selected\_user (user for analysis), df (DataFrame).

**daily\_timeline(selected\_user, df)**

* Generates a timeline plot of message distribution over days.
* Parameters: selected\_user (user for analysis), df (DataFrame).

**week\_activity\_map(selected\_user, df)**

* Provides information about the busiest day of the week.
* Parameters: selected\_user (user for analysis), df (DataFrame).

**month\_activity\_map(selected\_user, df)**

* Presents data on the busiest month in terms of activity.
* Parameters: selected\_user (user for analysis), df (DataFrame).

**activity\_heatmap(selected\_user, df)**

* Creates a heatmap showing message activity by hour and day.
* Parameters: selected\_user (user for analysis), df (DataFrame).

**most\_busy\_users(df)**

* Identifies and returns the most active users in the chat.
* Parameters: df (DataFrame).

**create\_wordcloud(selected\_user, df)**

* Generates a word cloud visualization based on user messages.
* Parameters: selected\_user (user for analysis), df (DataFrame).

**most\_common\_words(selected\_user, df)**

* Calculates and returns the most common words used by a user.
* Parameters: selected\_user (user for analysis), df (DataFrame).

**emoji\_helper(selected\_user, df)**

* Analyzes and returns information about emoji usage.
* Parameters: selected\_user (user for analysis), df (DataFrame).

**5. Model Training**

The model training section focuses on building a sentiment analysis model using TensorFlow and Keras. This model is designed to analyze text data and predict whether the sentiment of a given text is positive or negative.

**Data Loading and Preprocessing**

1. Data Loading: The sentiment analysis model is trained on a dataset containing labeled text data. The dataset is loaded from a CSV file using the Pandas library. The loaded DataFrame consists of columns such as sentiment and text, where sentiment represents the sentiment label (positive or negative) and text contains the text messages.
2. Label Decoding: The sentiment labels in the dataset are originally represented as numerical values (0 for negative, 4 for positive). A label decoding function is applied to convert these numerical labels into human-readable sentiments ("Negative" and "Positive").
3. Text Cleaning and Preprocessing: The text data is preprocessed to remove special characters, URLs, and non-alphanumeric characters using regular expressions. Additionally, stopwords are removed, and stemming may be applied to reduce words to their base form.

**Model Architecture**

1. Word Embeddings (GloVe): Word embeddings provide a way to represent words as numerical vectors in a high-dimensional space. Pre-trained GloVe word embeddings are used in this model. These embeddings capture semantic relationships between words and are used as input features for the neural network.
2. Embedding Layer: The model architecture begins with an embedding layer. The embedding layer maps words to their corresponding word vectors based on the pre-trained GloVe embeddings. It is important to note that the embeddings are not trainable in this case to retain the original learned representations.
3. Convolutional and LSTM Layers: After the embedding layer, the model includes a Convolutional Neural Network (CNN) layer with ReLU activation to capture local patterns in the text data. A Bidirectional Long Short-Term Memory (BiLSTM) layer is used to capture sequential dependencies in the text.
4. Dense Layers: The output from the LSTM layer is passed through a series of dense (fully connected) layers. These layers apply non-linear transformations to the extracted features and allow the model to learn complex patterns.
5. Output Layer: The final layer is a dense output layer with a sigmoid activation function. It produces a single value between 0 and 1, indicating the predicted sentiment score. A score above 0.5 is classified as "Positive," and a score below 0.5 is classified as "Negative."

**Model Compilation and Training**

1. Model Compilation: The model is compiled using the Adam optimizer and the binary cross-entropy loss function. The optimizer adjusts the model's weights during training to minimize the loss function, thus improving the model's predictions.
2. Learning Rate Reduction: A callback, ReduceLROnPlateau, is used to dynamically adjust the learning rate during training based on validation loss. This can help improve convergence and prevent overshooting.
3. Training: The model is trained using the training data. The training process involves feeding the model input data (text sequences) and target data (sentiment labels). The model updates its weights through backpropagation and gradient descent.
4. Batch Size and Epochs: Training is performed in batches of data, with the batch size specified. The training process iterates through the entire dataset for a defined number of epochs. An epoch is a single pass through the entire training dataset.

**Model Evaluation and Visualization**

1. Training Progress Visualization: During training, the model's accuracy and loss on both the training and validation datasets are monitored. These metrics are visualized using line plots to track the model's learning progress.
2. Sentiment Prediction and Decoding: After training, the model is used to predict sentiments on the test data. Predicted sentiment scores are decoded into human-readable sentiments (positive or negative).
3. Confusion Matrix: The performance of the sentiment analysis model is assessed using a confusion matrix. The confusion matrix provides insight into the model's classification accuracy, false positives, false negatives, and true positives.
4. Plotting the Confusion Matrix: The confusion matrix is visualized as a heatmap, where rows represent true labels and columns represent predicted labels. Each cell in the heatmap contains a value indicating the proportion of correctly classified samples.

**6. Conclusion**

The WhatsApp Chat Analyzer is a versatile tool for analyzing and visualizing chat data from WhatsApp conversations. With features like statistics, timelines, activity maps, sentiment analysis, and more, users can gain valuable insights from their chat data. The inclusion of helper functions simplifies the process of generating visualizations and statistics within the Streamlit app, making it user-friendly and informative.