Prophet Forecasting:

The process involves reading and filtering the data, grouping it by date, and preparing it for time series forecasting with Prophet. The forecast detects anomalies (breakouts) and visualizes trends. Finally, the data, including forecasts and anomalies, is stored in a PostgreSQL database for further use.

In the context of keyword extraction, the list of `covid_keywords` plays a vital role in identifying and isolating relevant content associated with the COVID-19 pandemic. These terms—such as "covid", "pandemic", "vaccine", "quarantine", and others—are carefully chosen to represent various facets of the pandemic, including health-related issues, government responses, public safety measures, and societal impacts.

The core purpose of these keywords is to filter and analyze text data that pertains specifically to the COVID-19 pandemic. In the provided code, this is achieved through a filtering mechanism that scans the text and hashtags of social media posts, news articles, or other types of content for the presence of these keywords. If a match is found, the content is flagged as relevant for further processing.

The specific keywords in the list represent a wide array of pandemic-related topics:

- **Health and Disease**: Words like "covid", "covid19", "coronavirus", "vaccine", and "vaccination" focus on the virus itself, its spread, and the global vaccination efforts.
- **Preventive Measures**: Keywords such as "lockdown", "quarantine", "mask", and "socialdistancing" highlight the measures taken to mitigate the spread of the virus.
- **Public Awareness and Response**: Terms like "wuhan", "stayhome", and "pandemic" reflect societal responses to the outbreak and the global effort to manage the crisis.

By applying these keywords in keyword extraction, one can easily detect and isolate content that discusses the pandemic, which is particularly useful in the analysis of large datasets, such as social media posts, news articles, or any publicly available textual content.

For instance, in the provided code, a function called `contains_covid_keywords` checks each text entry in a dataset (such as tweets) for any occurrence of these keywords. If any keyword is found, the content is considered relevant and is extracted for further analysis. This ensures that only content directly related to COVID-19 is retained, filtering out irrelevant data.

Moreover, the keyword extraction approach can be particularly beneficial when performing **sentiment analysis** or **trend analysis**. For example, by focusing on posts related to vaccines or lockdown measures, one can gauge public sentiment, identify emerging trends in public opinion, and monitor how the discussion evolves over time. This could provide valuable insights into how different populations are responding to COVID-related topics or how government policies are being perceived.

In summary, the use of `covid_keywords` in keyword extraction is a powerful tool for isolating and analyzing COVID-related content in large datasets. It helps identify relevant discussions, provides a foundation for sentiment and trend analysis, and ensures that the analysis focuses on the most critical aspects of the pandemic, from health concerns to public safety measures. This approach is particularly useful in understanding how the COVID-19 crisis has impacted public discourse across various platforms.

```
covid_keywords = {"covid", "covid19", "coronavirus", "pandemic", "lockdown",
                  "quarantine", "mask", "wuhan", "vaccine", "vaccination",
                  "socialdistancing", "stayhome"}
```

Sentiment Analysis :

This code performs multiple stages of processing for sentiment analysis on a COVID-19 dataset, using various machine learning models in PySpark. Here's a summary of the key steps:

1. **Data Loading and Sampling**: The code loads a CSV dataset (`Covid-19(Apr-Jun 2020).csv`), samples 35,000 entries randomly.
2. **Text Preprocessing**: A function `clean_tweets` is applied to clean the text data by removing URLs, mentions, hashtags, and special characters, while converting the text to lowercase.
3. **Data Splitting**: The cleaned dataset is split into training and test sets, with 85% of the data used for training and 15% for testing.There are 29898 rows in the training set, and 5101 in the test set
4. **Sentiment Indexing**: Sentiment labels are converted to numerical values using `StringIndexer`, which is applied to both the training and test sets.
5. **Model Training**: A pipeline is created for training machine learning models, which includes tokenization, CountVectorizer transformation, and model classification (Logistic Regression). The model is trained using the `Pipeline` API.
6. **Model Evaluation**: The code evaluates the models on the training and test sets using accuracy, F1 score, recall, and precision. It also plots confusion matrices for each model.
7. **Saving Models**: The trained models are saved to a specific directory for future use.
8. **Results**: The evaluation metrics for each model are compiled into a DataFrame and displayed, showing performance on both the training and test sets.

This approach offers a comprehensive way to process text data, train machine learning models, and evaluate them, providing valuable insights into sentiment analysis of COVID-19-related content.

```
LogisticRegression -> Train Acc: 0.8267777108836711, Test Acc: 0.8168986473240542
LogisticRegression -> Train F1: 0.823881303675293, Test F1: 0.8138348225875812
LogisticRegression -> Train Recall: 0.8267777108836711, Test Recall: 0.816898647324054
LogisticRegression -> Train Precision: 0.8610912799161194, Test Precision: 0.855312723998569

Classification Report for LogisticRegression:
              precision    recall  f1-score   support

    negative       0.71      0.99      0.83      2032
     neutral       0.93      0.76      0.84      1613
    positive       0.97      0.63      0.76      1456
```

accuracy			0.82	5101
macro avg	0.87	0.80	0.81	5101
weighted avg	0.86	0.82	0.81	5101

Spark Stream:

In this project, we designed a **real-time data pipeline** that reads COVID-19-related comments from a **Kafka** stream, processes and classifies the data using **PySpark**, performs **Named Entity Recognition (NER)** with fuzzy matching, predicts **sentiment** with a pre-trained model, and finally stores the results into a **PostgreSQL** database for further analysis.

The pipeline consists of the following key stages:

---

1. Logging Setup and Spark Session Initialization

The pipeline begins by setting up logging to track system activities and initializing a **SparkSession** named `"StreamPredict"`. This allows the system to manage real-time streaming and batch processing efficiently.

---

2. Reading Data from Kafka

We connect to a local **Kafka** broker and subscribe to the topic `CovidF`. The incoming Kafka stream contains JSON data with two fields:

- `event_timestamp`
- `comment`

The JSON data is parsed and prepared for further processing.

---

3. Data Cleaning and Preprocessing

To prepare the comments for analysis:

- **Lowercasing** is applied.
- **Removal of noise** such as retweet tags (`rt`), URLs, mentions, hashtags, non-alphanumeric characters, and extra spaces.
- Rows with **null or empty comments** are filtered out.

This ensures that the downstream models work on clean and standardized input data.

---

4. Filtering COVID-19-Related Comments

Using a defined set of **COVID-related keywords** (e.g., "covid", "pandemic", "vaccine"), comments are filtered to retain only those discussing the pandemic. This step ensures domain-specific focus.

The timestamp is also transformed to **date format** (without time details) for better aggregation later.

---

5. Fuzzy Named Entity Recognition (NER)

To enhance the analysis:

- We perform **fuzzy matching** to detect references to **diseases, drugs, viruses, and vaccines** within the comments.
- A **broadcast variable** is used to share the keyword dictionary efficiently across Spark workers.
- RapidFuzz's `partial_ratio` scorer identifies the best-matching entity if the match score exceeds **95%**.
- The results are exploded to create one record per identified entity type and value.

This NER step captures implicit mentions even when the words are slightly misspelled or differently phrased.

---

6. Sentiment Analysis Using a Pre-trained Model

We load a **pre-trained logistic regression model** from local storage. For each comment:

- The model predicts the probability distribution across sentiment classes.
- A simple thresholding mechanism maps probabilities into three sentiment classes:
    - **Positive** (probability > 0.6)
    - **Negative** (probability < 0.4)
    - **Neutral** (otherwise)

Thus, every COVID-related comment is associated with a **sentiment label**.

---

7. Storing the Results into PostgreSQL

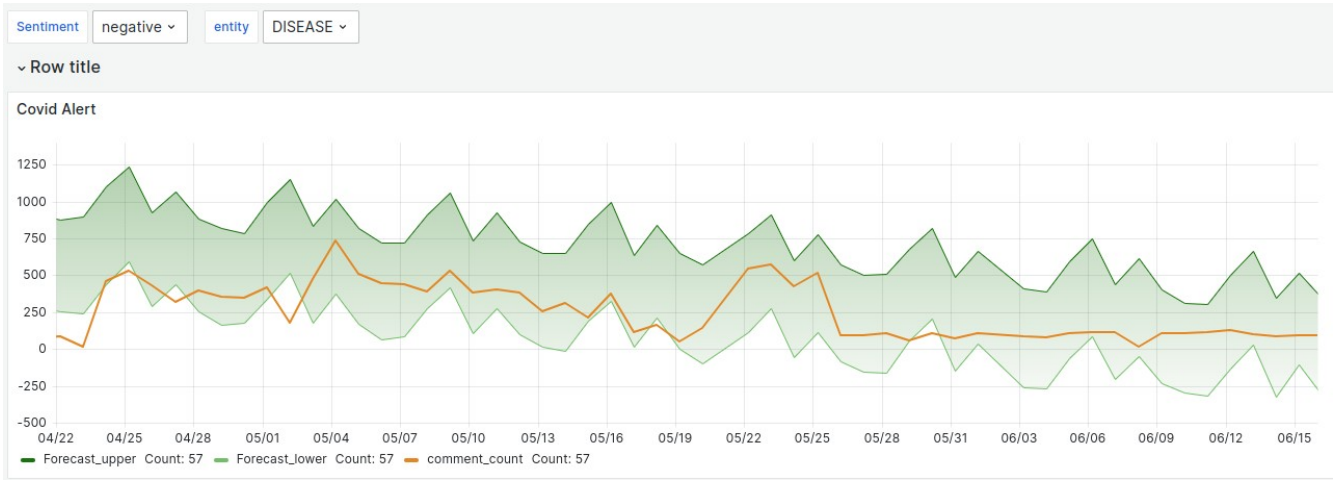
Using a `foreachBatch` function:

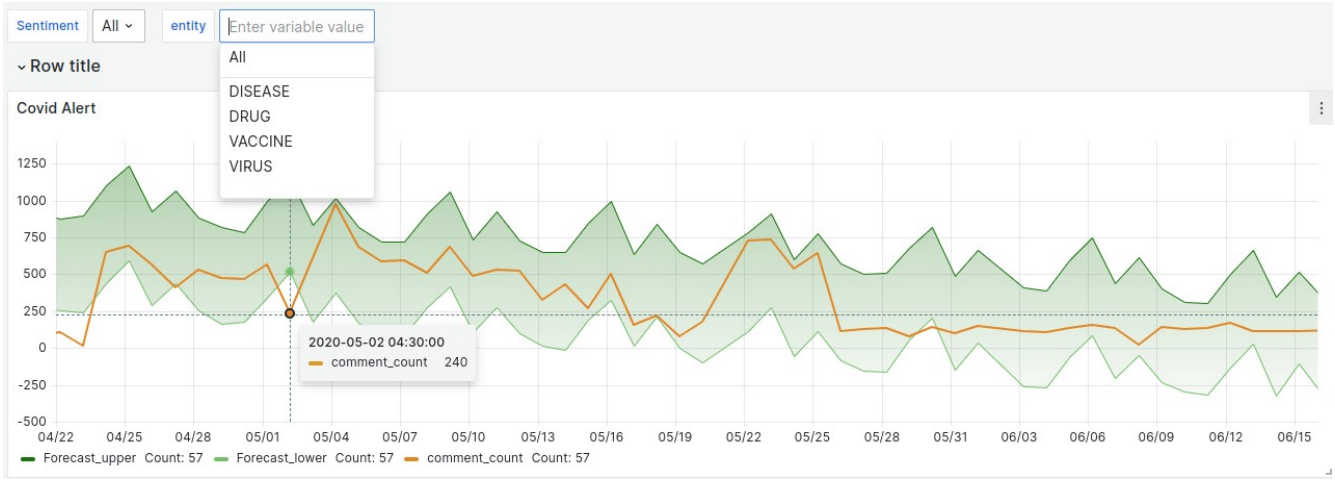
- Each micro-batch is **deduplicated** based on the combination of timestamp, comment, entity type, and entity value.

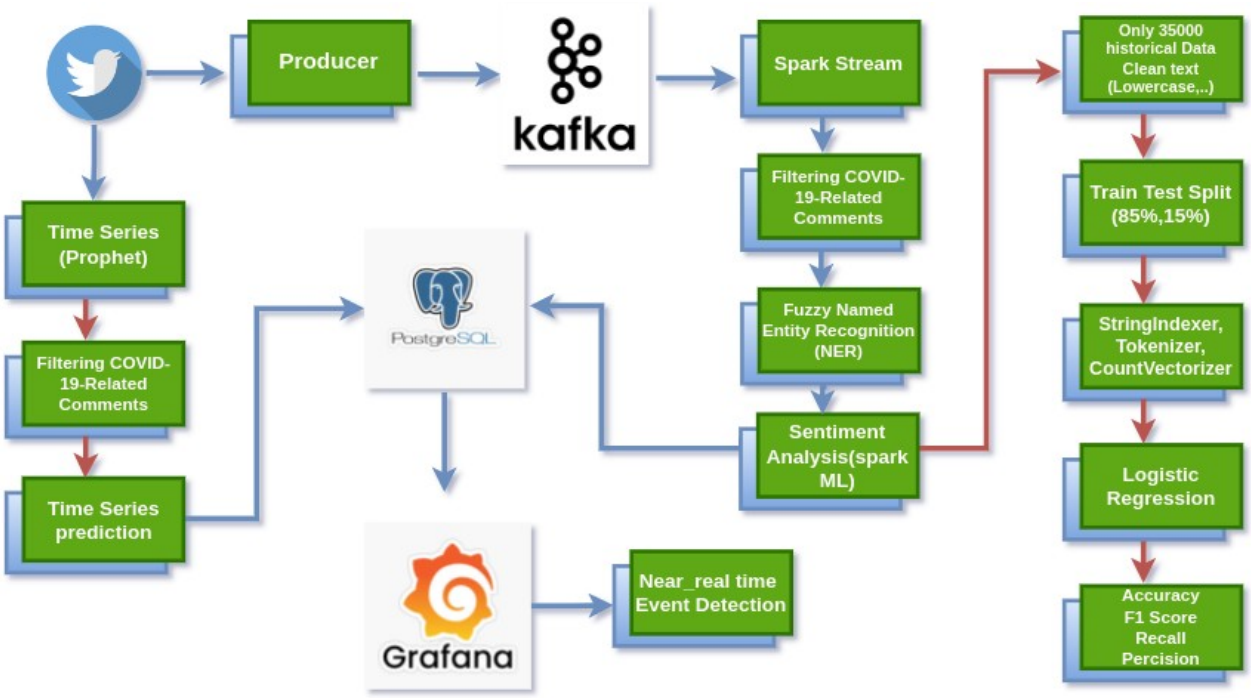
- Data is written into a **PostgreSQL table** named `covid_commentsF` in **append mode**.
- **Checkpointing** ensures fault tolerance and exactly-once processing semantics.

The saved fields include:

- event timestamp
- comment
- predicted sentiment label
- data type (test)
- recognized entity type
- recognized entity value






Producer
kafka
Spark Stream
Only 35000 historical Data Clean text (Lowercase,..)
Time Series (Prophet)
Filtering COVID-19-Related Comments
Fuzzy Named Entity Recognition (NER)
Train Test Split (85%,15%)
PostgreSQL
Filtering COVID-19-Related Comments
Sentiment Analysis(spark ML)
StringIndexer, Tokenizer, CountVectorizer
Time Series prediction
Logistic Regression
Grafana
Near_real time Event Detection
Accuracy
F1 Score
Recall
Percision