

# **Customer Sentiment and Trend Analysis**

---

**Team Members:**

---

**Mahmoud Kamel Salman**

---

**Sarah Abdel Hamid Abdel Ghany**

---

**Fatma Ahmed Mohamed**

---

**Youssef Ashraf Shaaban**

---

**Ahmed Mahmoud Abdelaziz**

# Data Preprocessing and Cleaning for NLP

# Loading the Data

Done with Pandas, with coding processing and bypassing corrupted rows

```
#loading the data and read
twitter_data = pd.read_csv('/content/training.1600000.processed.noemoticon.csv',
                           encoding='ISO-8859-1',
                           on_bad_lines='skip',
                           engine='python')
```

# Data exploration and cleaning:

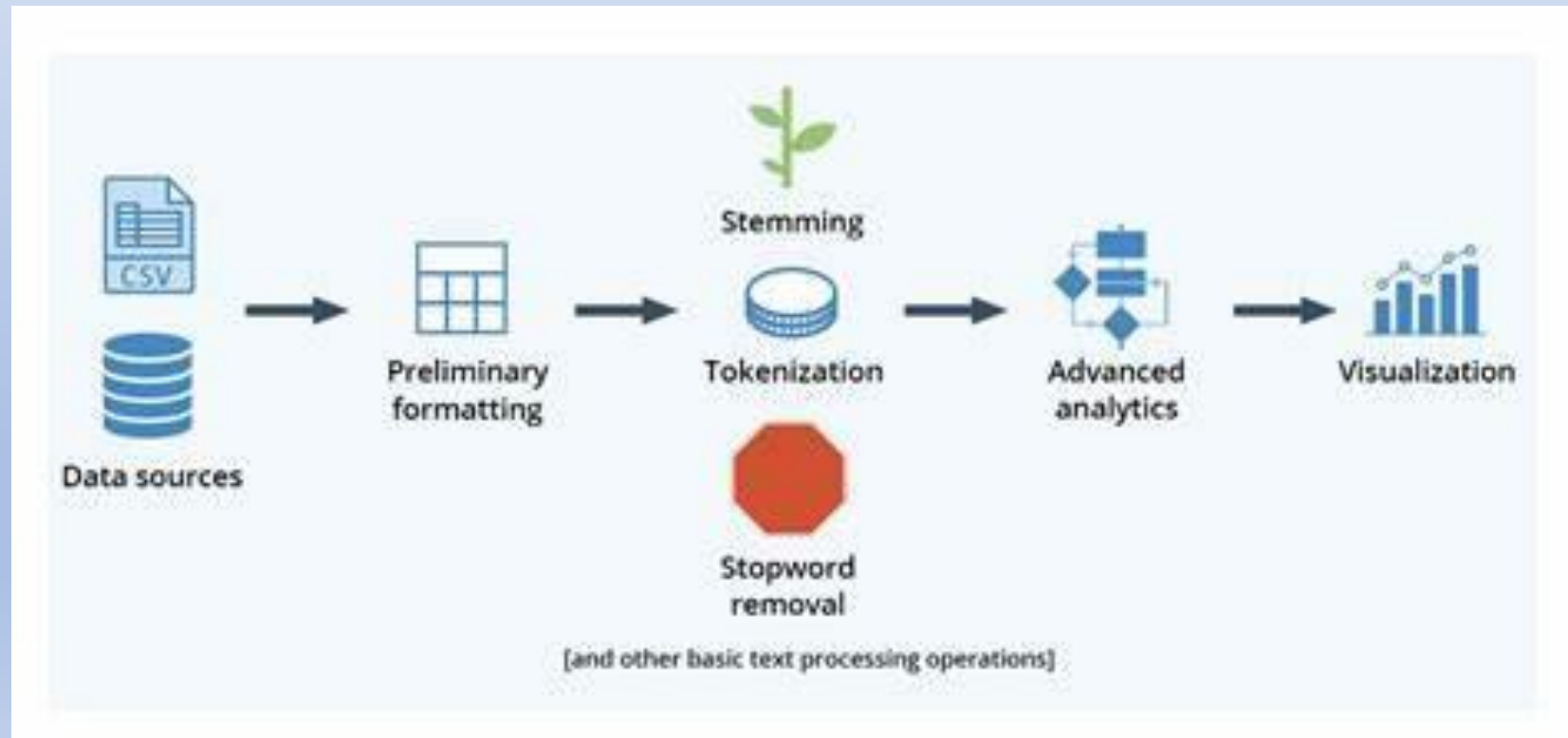
The data was explored to check for duplicates, missing values, and column structure. The categorization column was modified to facilitate binary categorization.

```
False  
duplicat is : 0
```

```
twitter_data.info()  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1600000 entries, 0 to 1599999  
Data columns (total 6 columns):  
#   Column      Non-Null Count  Dtype  
---  -  
0   target      1600000 non-null  int64  
1   ids         1600000 non-null  int64  
2   data        1600000 non-null  object  
3   flag        1600000 non-null  object  
4   user        1600000 non-null  object  
5   text        1600000 non-null  object  
dtypes: int64(2), object(4)  
memory usage: 73.2+ MB
```


# Text processing

A function was created to derive words and clean up tweets by removing non-alphabetic characters, common words, and reducing words to their roots.



# Summary and next step

By the end of these steps, you have a clean and processed dataset containing text ready for analysis, which can be used to build a Sentiment Analysis model using machine learning algorithms.



	target	stemmed_content
0	0	switchfoot http twitpic com zl awww bummer sho...
1	0	upset updat facebook text might cri result sch...
2	0	kenichan dive mani time ball manag save rest g...
3	0	whole bodi feel itchi like fire
4	0	nationwideclass behav mad see

# **Exploratory Data Analysis (EDA)**

# Perform exploratory data analysis (EDA) to understand the dataset.

## Dataset Summary:

- Number of records and features.
- Overview of categorical and numerical variables.
- Initial observations on missing data and distributions.
- Handling missing values (removal or imputation).

```
In [45]: df = pd.read_csv('cleaned_twitter_data (3).csv')

In [46]: df.head()

Out[46]:
```

	target	stemmed_content
0	0	switchfoot http twitpic.com zI awww bummer sho...
1	0	upset updat facebook text might cri result sch...
2	0	kenichan dive mani time ball manag save rest g...
3	0	whole bodi feel itchi like fire
4	0	nationwideclass behav mad see

```
In [47]: df.shape

Out[47]: (1600000, 2)

In [48]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1600000 entries, 0 to 1599999
Data columns (total 2 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   target          1600000 non-null int64
1   stemmed_content 1599505 non-null object
dtypes: int64(1), object(1)
memory usage: 24.4+ MB

In [50]: print(df.isnull().sum())

target          0
stemmed_content 495
dtype: int64

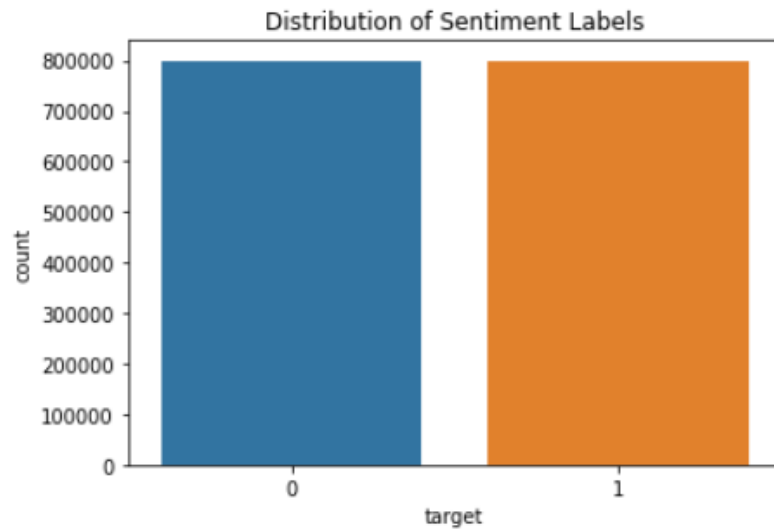
In [51]: df.dropna(inplace=True)
```



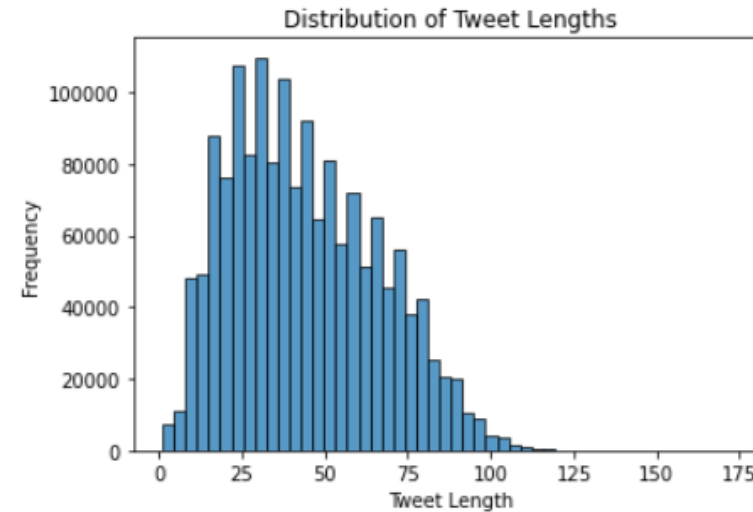
# Univariate and Bivariate Analysis Key Features:

Distribution of important variables through histograms.

```
plt.figure(figsize=(6, 4))
sns.countplot(x='target', data=df)
plt.title('Distribution of Sentiment Labels')
plt.show()
```



```
plt.hist(df['tweet_length'], bins=50, alpha=0.75, edgecolor='black')
plt.title('Distribution of Tweet Lengths')
plt.xlabel('Tweet Length')
plt.ylabel('Frequency')
plt.show() # right skewed , almost 25 word in one tweet.
```



# Extracting Most Common Words and Counts

Identify and analyze the 20 most common words in positive and negative tweets.

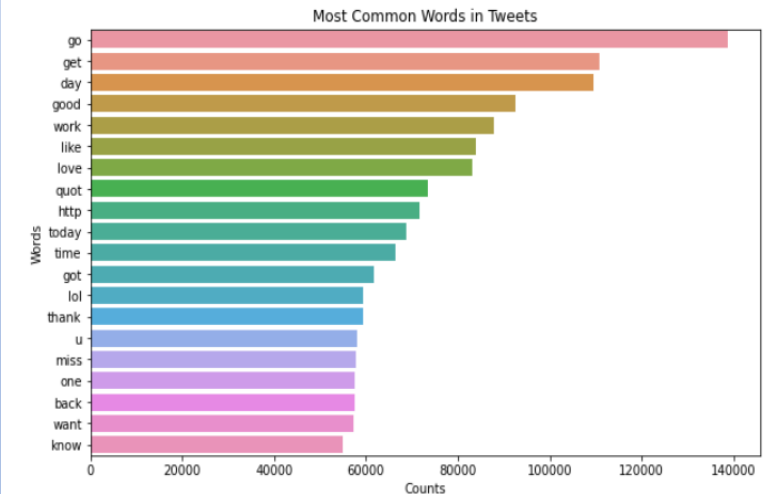
```
] from collections import Counter
```

```
] positive_tweets = ' '.join(df[df['target'] == 1]['stemmed_content'])  
negative_tweets = ' '.join(df[df['target'] == 0]['stemmed_content'])
```

```
] # Count the most common words in both + & -  
pos_word_counts = Counter(positive_tweets.split())  
neg_word_counts = Counter(negative_tweets.split())
```

```
] # most common words in pos and neg  
most_common_positive_words = pos_word_counts.most_common(20)  
most_common_negative_words = neg_word_counts.most_common(20)
```

```
: words = list(words)  
counts = list(counts)  
  
plt.figure(figsize=(10, 6))  
sns.barplot(x=counts, y=words)  
plt.title('Most Common Words in Tweets')  
plt.xlabel('Counts')  
plt.ylabel('Words')  
plt.show()  
# in all tweets pos & neg
```

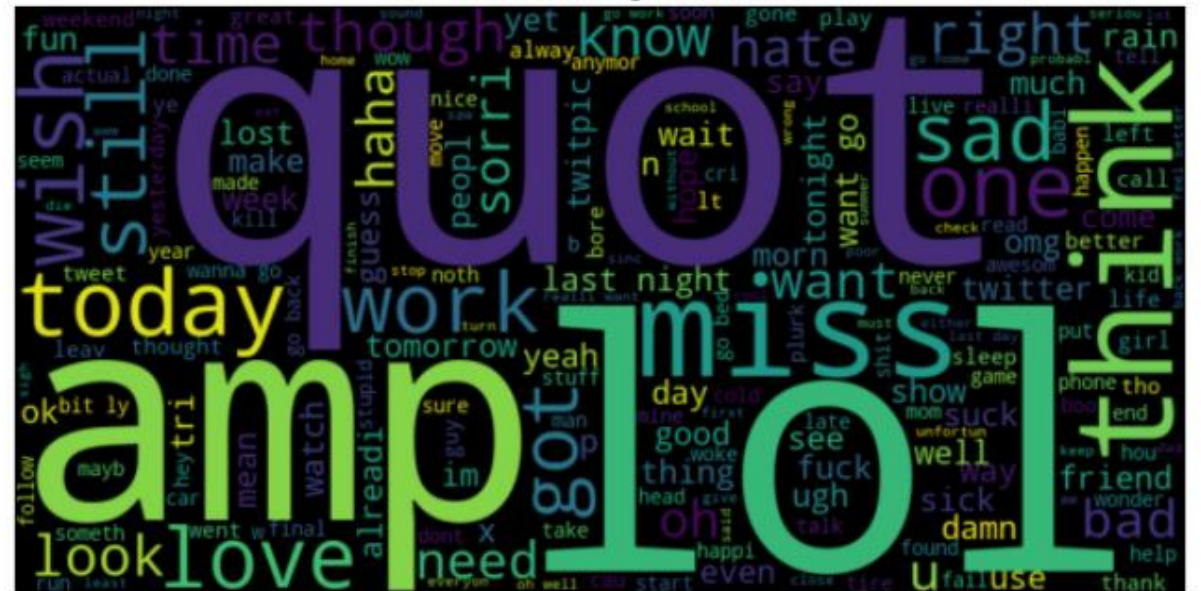


Word clouds for both positive and negative sentiments.

### Word Cloud for Positive Tweets



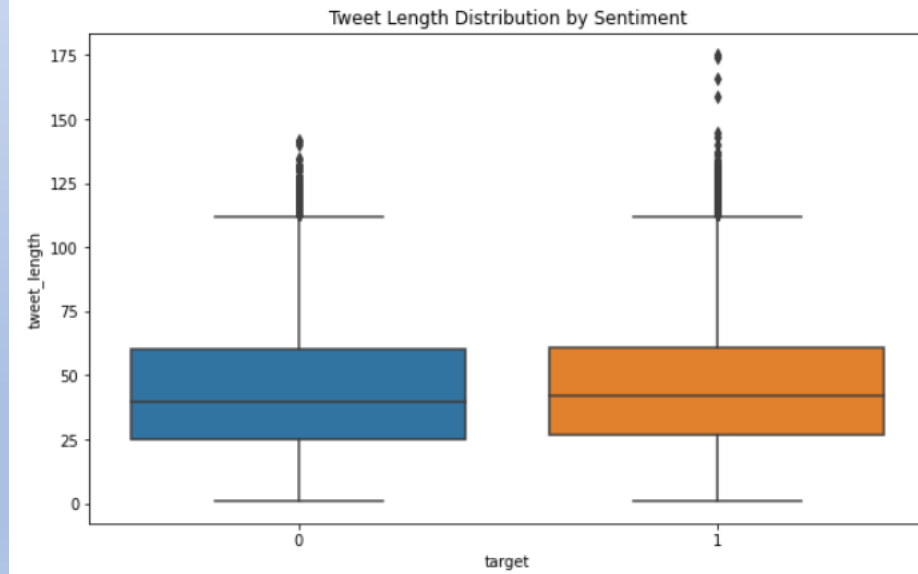
### Word Cloud for Negative Tweets



# identifying and handling outliers

## Identifying outliers

```
plt.figure(figsize=(10, 6))
sns.boxplot(x='target', y='tweet_length', data=df)
plt.title('Tweet Length Distribution by Sentiment')
plt.show()
```



## Handling outliers

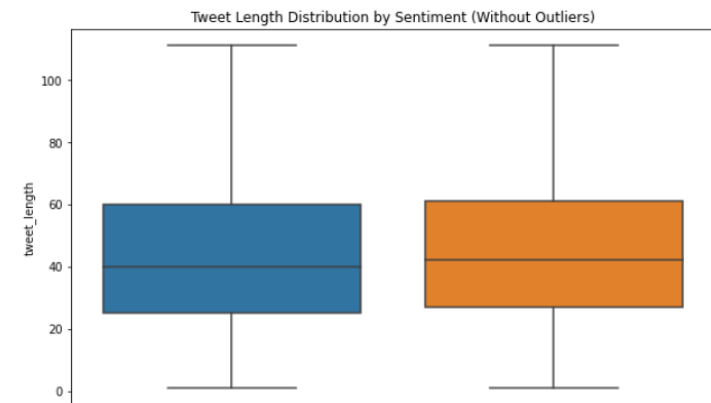
```
In [68]: # Calculate Q1(25%) and Q3(75%)
Q1 = df['tweet_length'].quantile(0.25)
Q3 = df['tweet_length'].quantile(0.75)

IQR = Q3 - Q1

# Defining the outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Removing outliers
df_filtered = df[(df['tweet_length'] >= lower_bound) & (df['tweet_length'] <= upper_bound)]

plt.figure(figsize=(10, 6))
sns.boxplot(x='target', y='tweet_length', data=df_filtered)
plt.title('Tweet Length Distribution by Sentiment (Without Outliers)')
plt.show()
```



**Milflow**

# MLFLOW

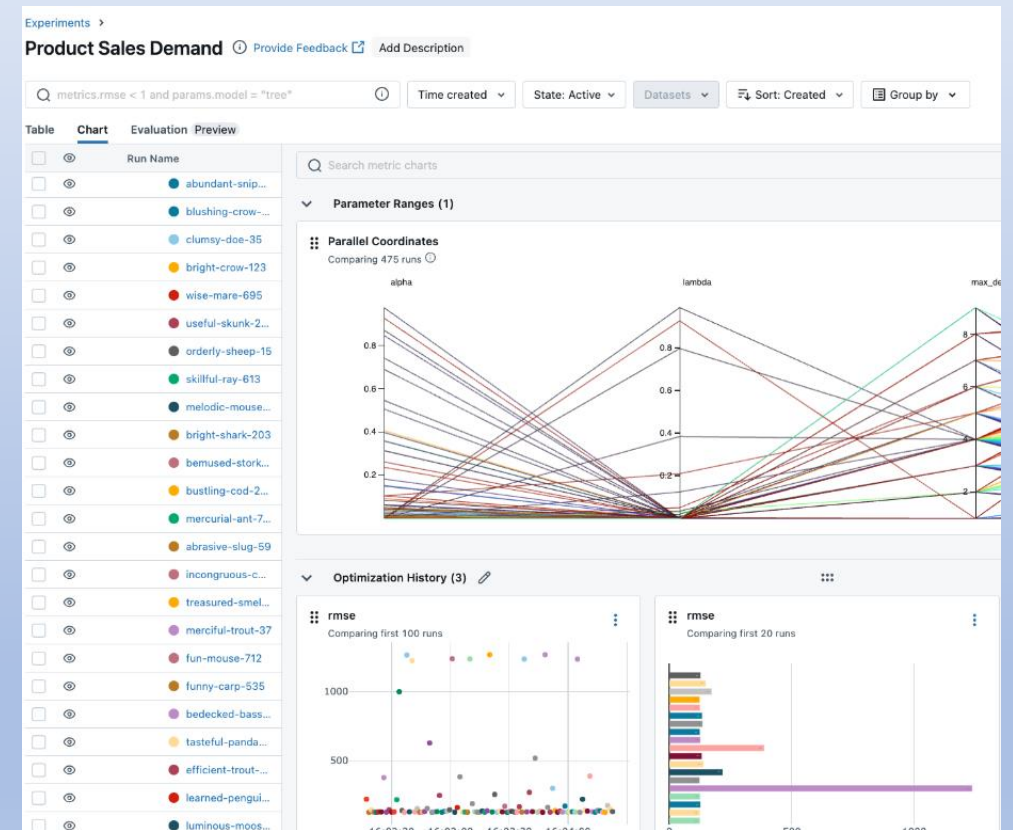
MLflow is an open-source platform designed to manage the machine learning lifecycle, providing tools to help with:

**1.Experiment tracking**

**2.Model management**

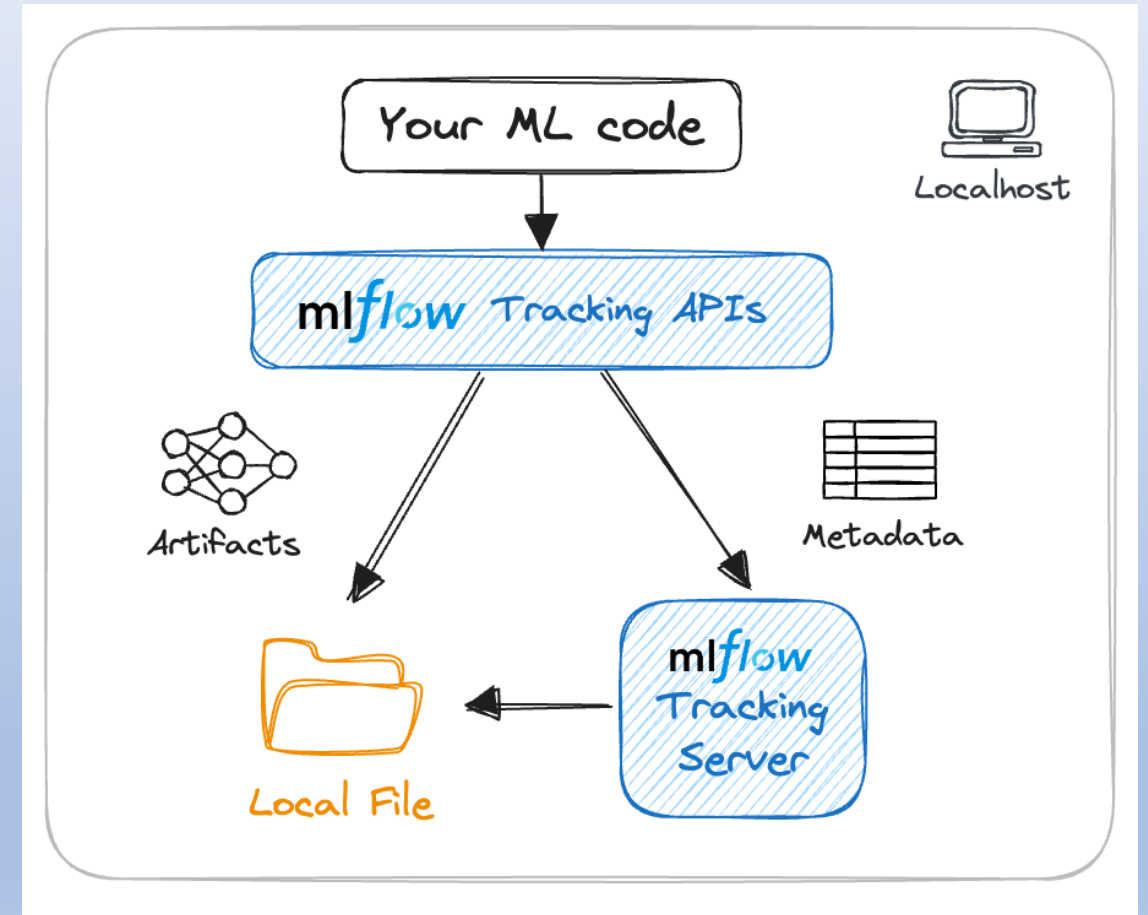
**3.Reproducibility**

**4.Deployment**



## Cloud or Server is Needed for MLflow

By using a cloud-based MLflow server (on platforms like **AWS, Google Cloud, or Azure**), the entire team can access the system remotely from any location, making it easier for distributed teams to collaborate.



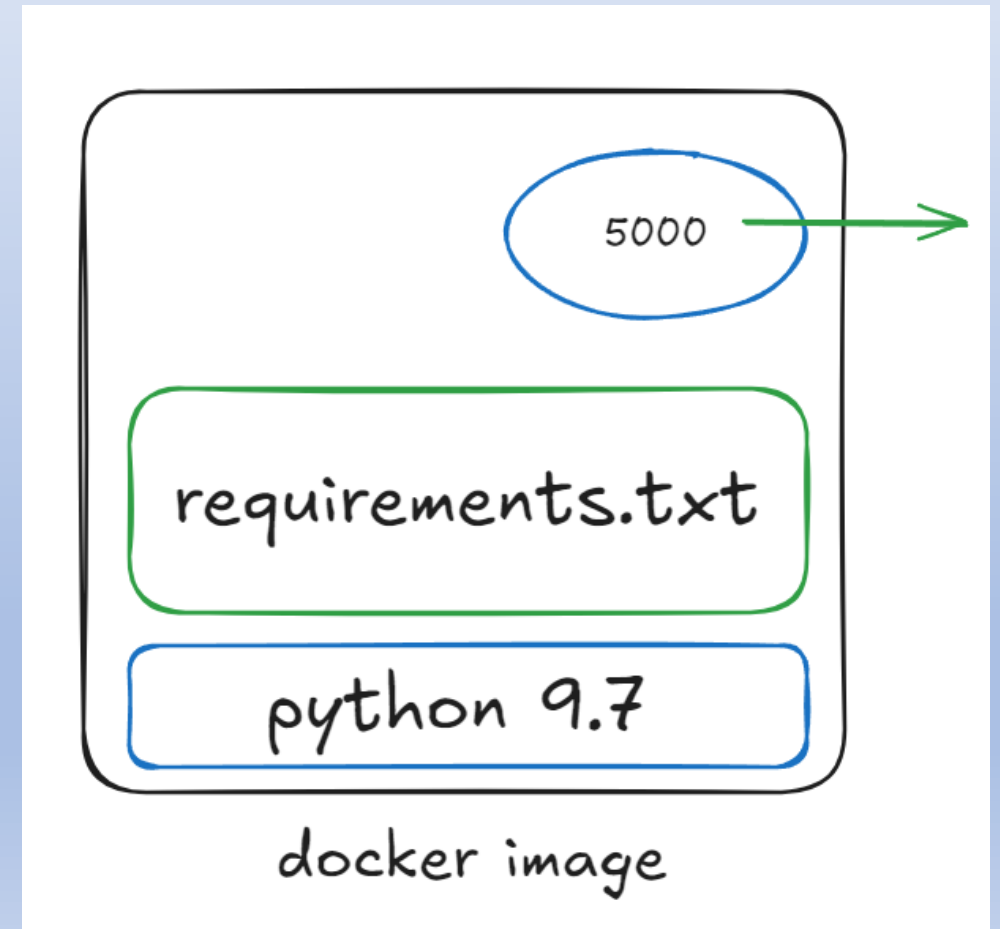


# DOCKER FOR TEAM COLLABORATION

## 1. Docker offers:

**Docker** containers offer a lightweight, consistent environment that can be shared across team members, enabling collaboration without the complexity of setting up individual environments.

```
1 FROM python:3.9-slim
2
3 ENV PIP_NO_CACHE_DIR=1
4
5 RUN pip install --upgrade pip
6 RUN pip install mlflow boto3
7
8 WORKDIR /mlflow-app
9
10 COPY . /mlflow-app
11
12 EXPOSE 5000
13
```





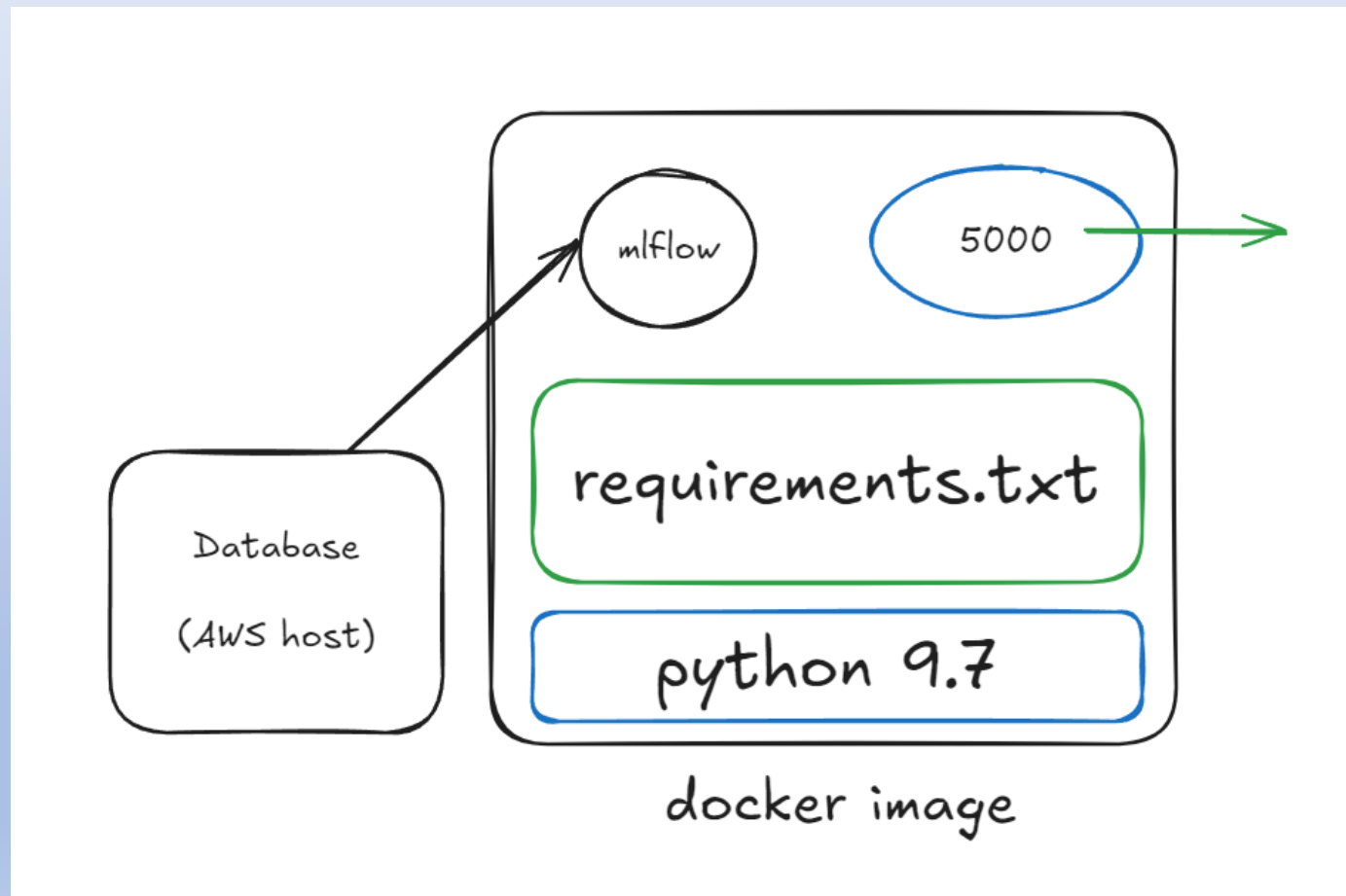
# DATABASE HOST FOR TEAM COLLABORATION

## MLflow with PostgreSQL:

docker-compose to run MLflow along with a PostgreSQL database for logging results, all inside containers.

## Small data share:

No need to share the big docker images with a team to work on the same mlflow server with same data.



# Train-Test Split

- **Objective:** Split the data to train models and evaluate their performance.
- **Split Ratio:**
  - **80%** for training
  - **20%** for testing.
- The training set teaches the models to recognize patterns.
- The testing set helps us evaluate how well the models generalize to new data.
- **Code Example**

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

# Model Training & Evaluation

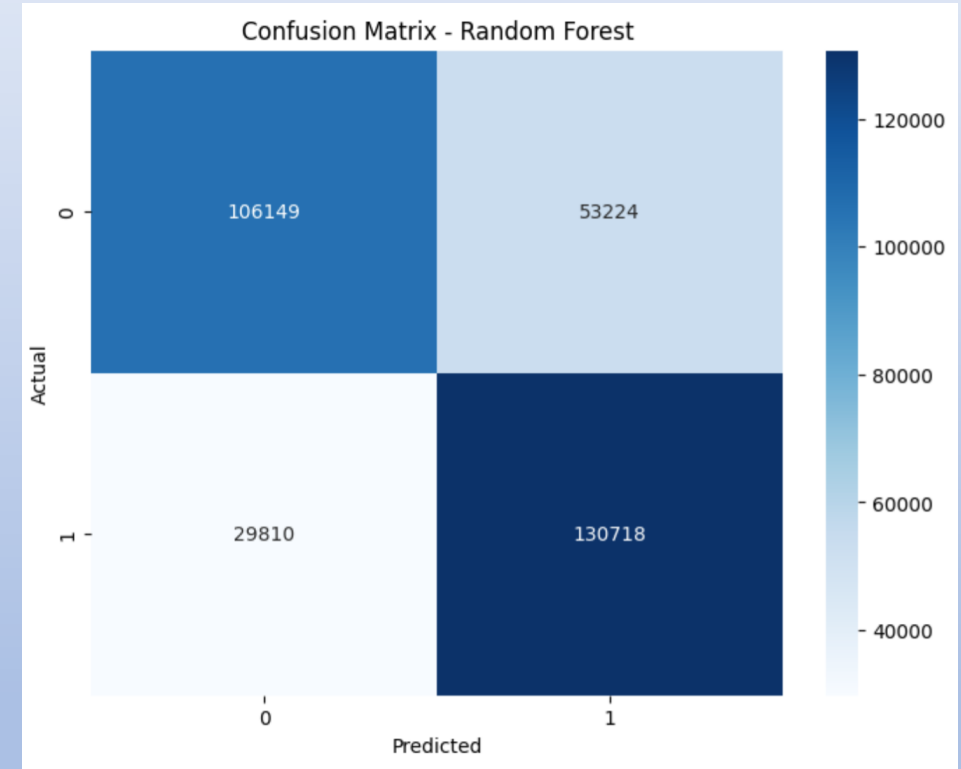
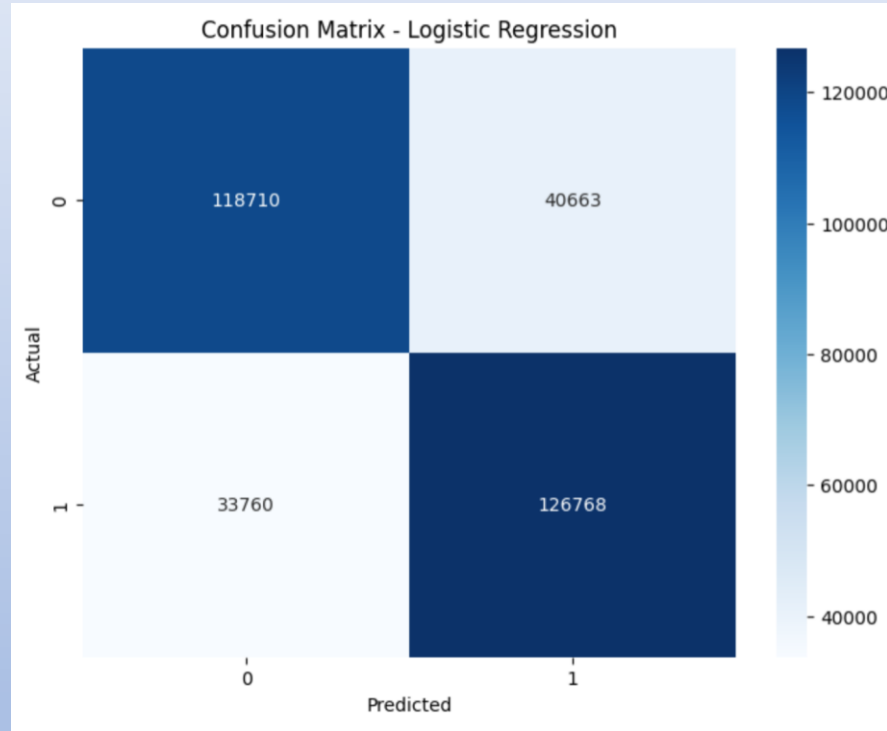
## Models Used:

- **Logistic Regression:** A linear model effective for text-based data.
- **Random Forest:** An ensemble model combining decision trees for better performance.

## Metrics Evaluated:

- **Accuracy Score:** Percentage of correct predictions.
- **Classification Report:** Includes precision, recall, and F1-score.
- **Confusion Matrix:** Visualizes true vs. predicted labels.

# Example Confusion Matrix Plot:



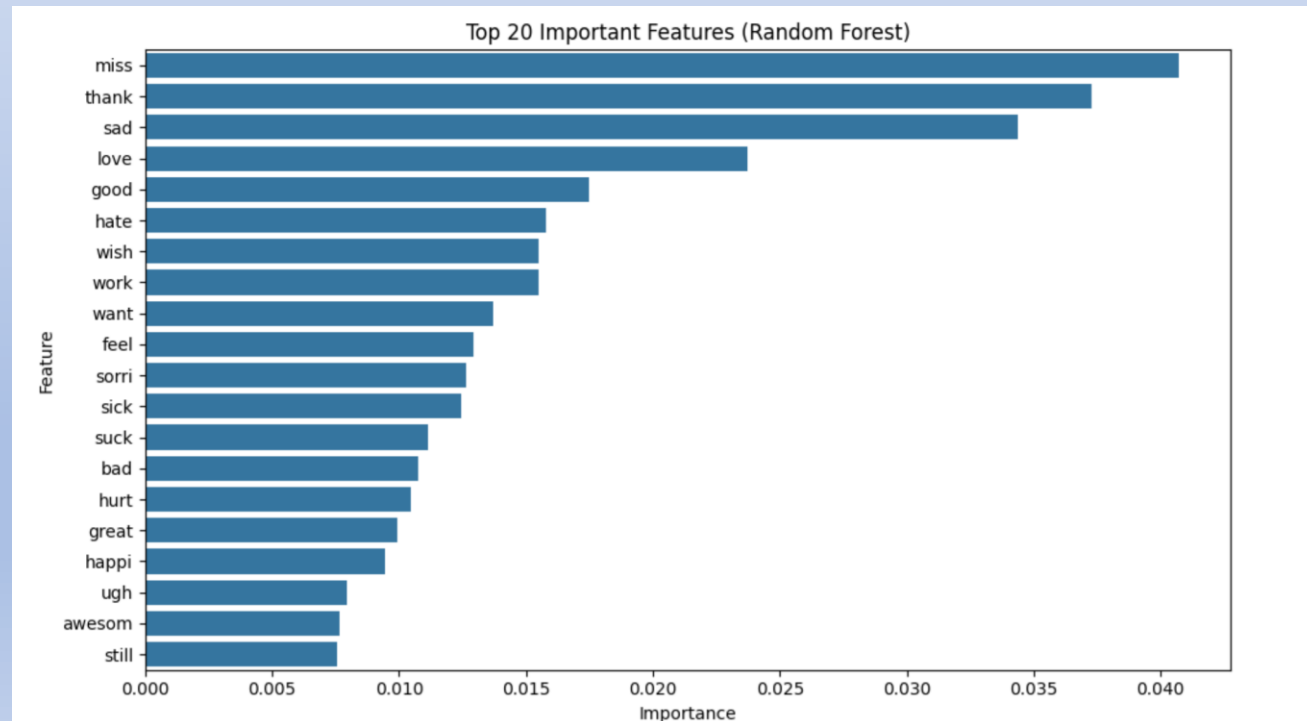
# Top Features from Random Forest

## What is Feature Importance?

It shows which words or phrases contribute most to the model's decisions.

## Steps:

- Extract top 20 important features using Random Forest.
- Visualize them with a bar plot.



# Predicting Sentiment of New Tweets

- **Purpose:** Test the models on new, unseen tweets.

- **Example Tweets:**

“I love this product!” → **Positive**

“Worst experience ever!” → **Negative**

**Code Example:**

```
# Example usage
new_tweets = [
    "I love this product! It's amazing!",
    "This is the worst experience I've ever had.",
]
```

**Output Example:**

```
Sentiment Predictions for New Tweets:
Tweet: I love this product! It's amazing!
Logistic Regression Prediction: Positive
Random Forest Prediction: Positive

Tweet: This is the worst experience I've ever had.
Logistic Regression Prediction: Negative
Random Forest Prediction: Negative
```

# Bidirectional Encoder Representations from Transformers (BERT)

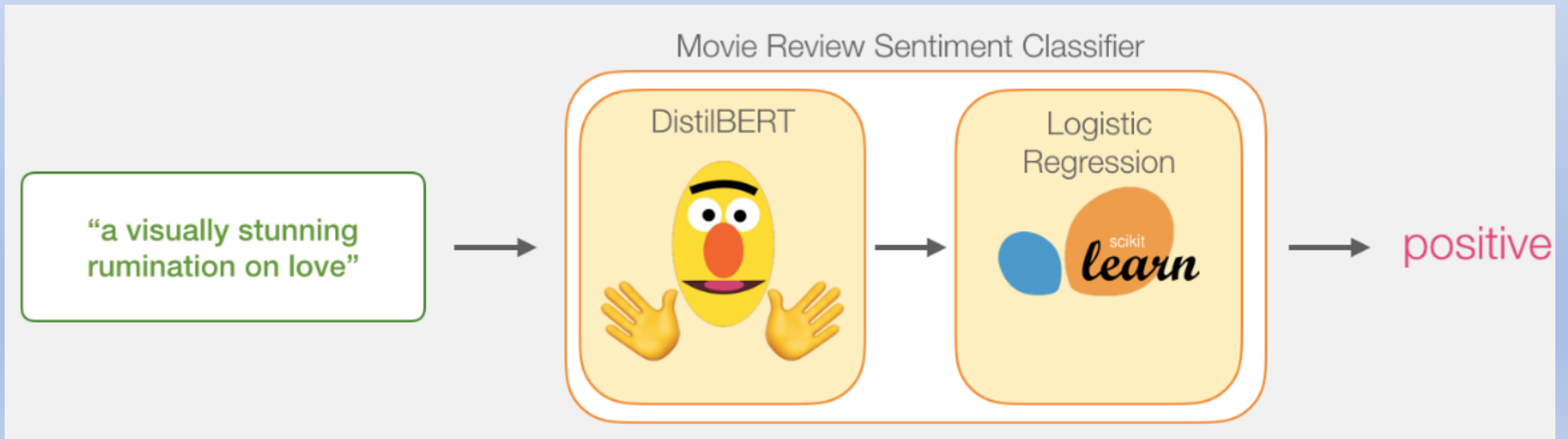
# Sentence Sentiment Classification

Our goal is to create a model that takes a sentence (just like the ones in our dataset) and produces either 1 (indicating the sentence carries a positive sentiment) or a 0 (indicating the sentence carries a negative sentiment). We can think of it as looking like this:





- Under the hood, the model is actually made up of two model.
- processes the sentence and passes along some information it extracted from it on to the next model. DistilBERT is a smaller version of BERT developed and open sourced by the team at HuggingFace. It's a lighter and faster version of BERT that roughly matches its performance.
- The next model, a basic Logistic Regression model from scikit learn will take in the result of DistilBERT's processing, and classify the sentence as either positive or negative (1 or 0, respectively).



# load a BERT tokenizer and model

- BERT understands text as a series of token IDs. Without tokenization, the model wouldn't be able to interpret the input.
- Tokenization also handles special cases like subwords, punctuation, and unknown tokens, which allows BERT to process a wide variety of text inputs effectively.

```
# Load the tokenizer and BERT model

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

# Tokenize the data
train_encodings = tokenizer(train_texts, truncation=True, padding=True, max_length=250)
test_encodings = tokenizer(test_texts, truncation=True, padding=True, max_length=250)
```

Raw text dataset

0
a stirring , funny and finally transporting re...
apparently reassembled from the cutting room f...
they presume their audience wo n't sit still f...
this is a visually stunning rumination on love...
jonathan parker 's bartleby should have been t...

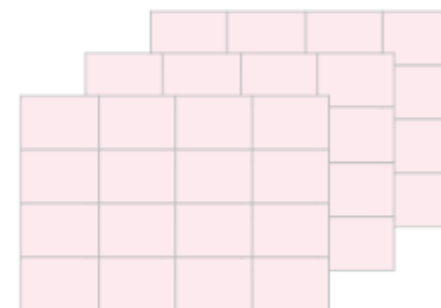
Tokenized input  
tensor

	0	1	...	66
0	101	1037	...	0
1	101	2027	...	0
...	...	...	...	...
1,999	101	1996	...	0

DistilBERT



BERT Output  
Tensor/**predictions**



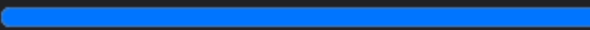
## our train

```
# Convert to tensor format
class SentimentDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)
```

```
# Initialize Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    compute_metrics=compute_metrics,
    callbacks=callbacks,
    optimizers=(optimizer, lr_scheduler)
)
```



[10000/10000 3:04:23, Epoch 1/1]

Step	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
2000	0.268900	0.473229	0.801063	0.800918	0.802085	0.801063
4000	0.307200	0.453801	0.796669	0.796408	0.798389	0.796669
6000	0.411900	0.410535	0.813969	0.813952	0.814040	0.813969
8000	0.411400	0.401085	0.816063	0.816008	0.816537	0.816063
10000	0.403700	0.394789	0.819581	0.819580	0.819584	0.819581

# **Generative Adversarial network (GAN)**

# Generative Adversarial network (GAN)

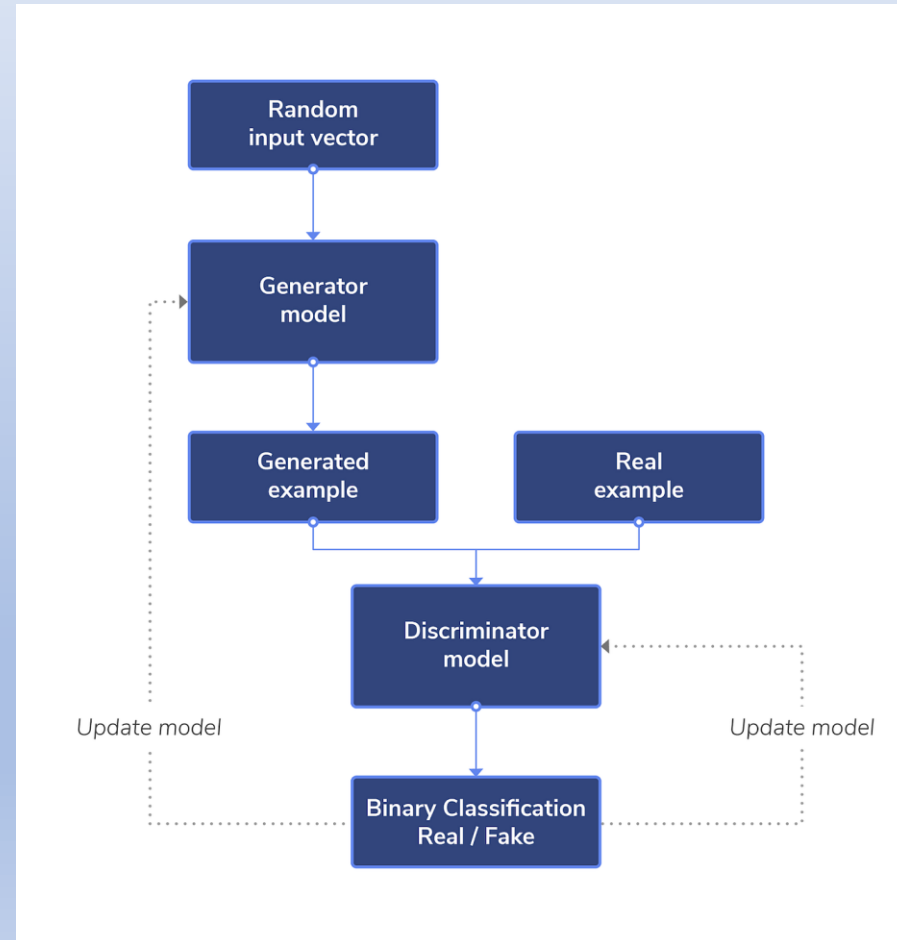
GANs consist of two networks:

- Generator (creates synthetic data) :

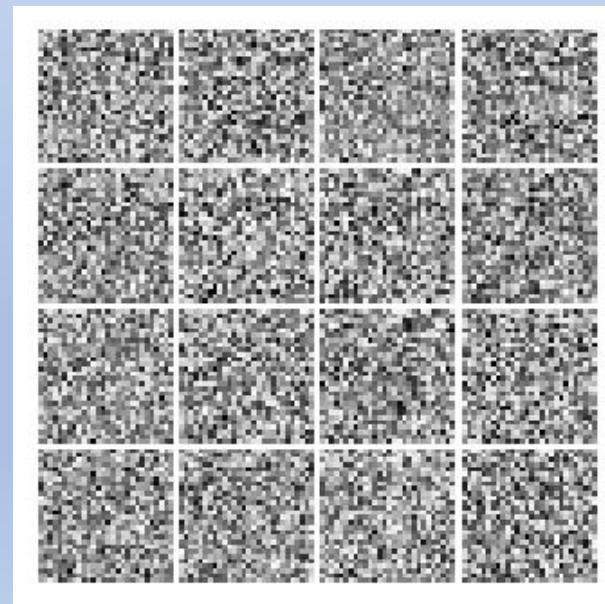
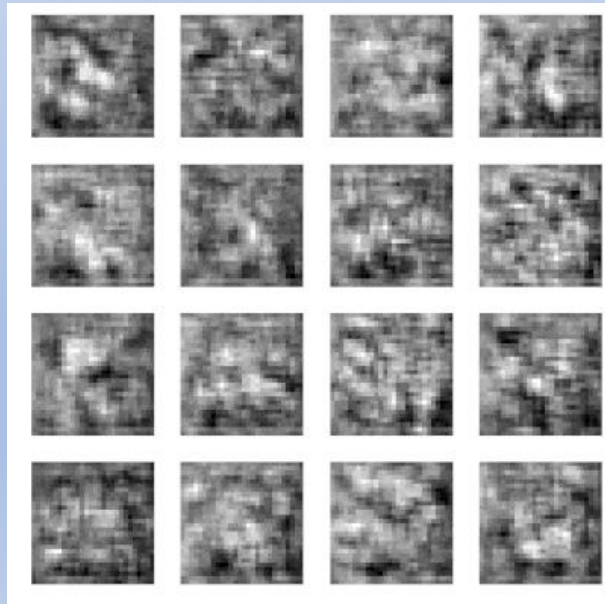
Its job in a GAN is creates synthetic data by learning to mimic the real data, aiming to fool the discriminator into thinking the generated data is authentic.

- Discriminator(evaluates data authenticity).

Its job is to evaluates whether the input data is real (from the dataset) or fake (generated by the generator), guiding the generator's improvement.



GANs revolutionized the field of visual data generation but faced significant challenges when applied to text due to the **sequential and discrete nature of language**, making it harder for the model to generate coherent and meaningful text.



# What are the challenges of using GANs for text generation

## 1. Discrete Nature of Text:

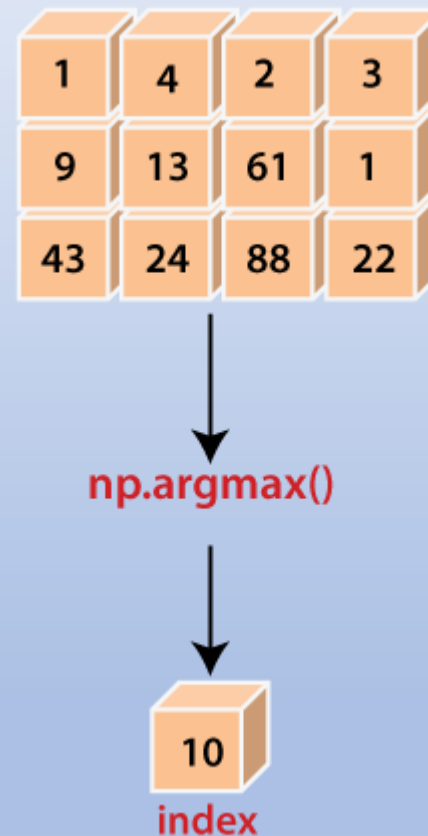
Unlike images, where pixel values are continuous, text is composed of discrete tokens (words or characters). This makes it difficult for GANs to calculate gradients effectively during the training process, as backpropagation relies on continuous values.

## Others:

- **Sequential Structure**
- **Mode Collapse**
- **Evaluation Challenges:**



- We usually use `argmax` function to convert the output of the generator into a vector of generated words.
- But the problem with `argmax` function is that it is not differentiable, which obstructs the process of backpropagation in the neural network.
- That's what makes using GANs for text a challenge.



# The solution

- Gumbel softmax
- Reinforcement learning
- Variational autoencoder

```
660
661     if not filtered_grads:
--> 662         raise ValueError("No gradients provided for any variable.")
663     if missing_grad_vars:
664         warnings.warn(
```

**ValueError:** No gradients provided for any variable.

# Our simple GAN

```
# Discriminator model
def build_discriminator(vocab_size, seq_length):
    model = Sequential(name='discriminator')
    model.add(Input(shape=(seq_length,))) # Expecting a sequence of word indices
    model.add(Embedding(input_dim=vocab_size, output_dim=50, input_length=seq_length)) # Embedding layer
    model.add(LSTM(128)) # Recurrent layer for processing sequences
    model.add(Dense(1, activation='sigmoid')) # Binary classification: real or fake
    model.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))
    model.summary()
    return model
```

```
# Generator model
def build_generator(vocab_size, seq_length):
    model = Sequential(name='generator')
    model.add(Dense(120, input_dim=100)) # Random noise vector length is 100
    model.add(Reshape((seq_length, 12))) # Reshape to (sequence_length, features)
    model.add(GRU(128, return_sequences=True)) # Recurrent layer for sequence generation
    model.add(Dense(vocab_size)) # Output logits for vocab distribution
    # Use Gumbel-Softmax to generate a sequence of word indices
    model.add(Lambda(lambda x: gumbel_softmax_sample(x, temperature=0.5)))
    model.compile(loss='categorical_crossentropy', optimizer=Adam(0.0002, 0.5))
    model.summary()
    return model
```

```
# GAN model: Combined Generator and Discriminator
def build_gan(generator, discriminator):
    discriminator.trainable = False # We freeze the discriminator when training the generator
    model = Sequential(name='GAN')
    model.add(generator)
    # model.add(Reshape((10,)))
    model.add(discriminator)
    model.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))
    print(model.summary())
    return model
```

# Advanced alternative solutions for GANs.

- **SeqGAN** - [SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient](#)
- **LeakGAN** - [Long Text Generation via Adversarial Training with Leaked Information](#)
- **SentiGAN** - [SentiGAN: Generating Sentimental Texts via Mixture Adversarial Networks](#)

# Deployment

# Frontend

We utilize the React framework for the layout and Tailwind for styling and colors, and we deploy our application using Vercel.

## Sentence Classifier

Classify

She received an award for her outstanding performance.

Positive

This restaurant has received poor reviews.

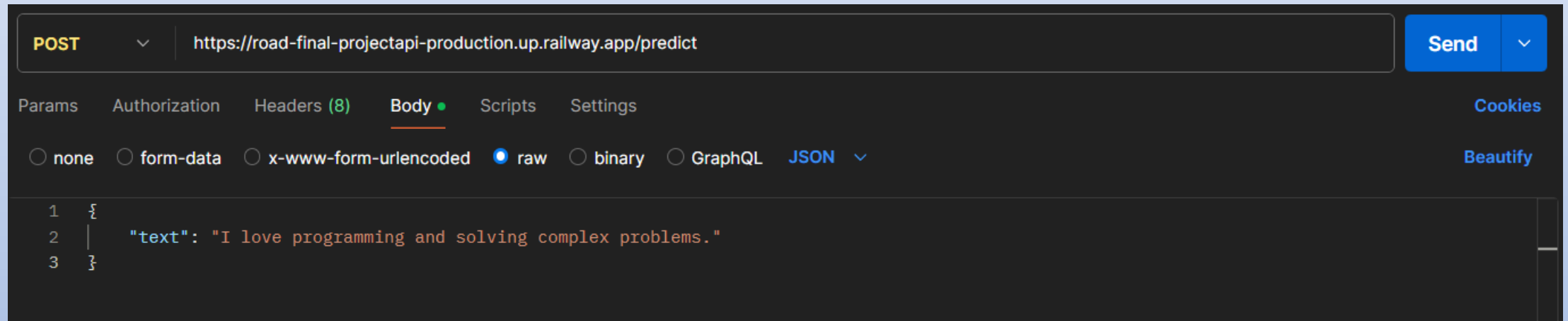
Negative

This book is truly inspiring and well-written.

Positive

# Backend

We use Flask to create our API and deploy the backend with CORS (AWS) on Railway.



This is the final deployment link

<https://road-final-project-front.vercel.app/>



**Thank you**