```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
!pip install pyspark
```

```
import pandas as pd
import seaborn as sns

from pyspark.sql import SparkSession
from pyspark.sql import functions as f

spark = (
    SparkSession.builder.appName("ModelTraining")
    .config("spark.executor.memory", "6g")
    .getOrCreate()
)
```

```
import html
schema = "polarity FLOAT, id LONG, date_time TIMESTAMP, query STRING, user STRING, text STRING"
timestampformat = "EEE MMM dd HH:mm:ss zzz yyyy"
spark.sql("set spark.sql.legacy.timeParserPolicy=LEGACY")

IN_PATH_RAW = "/kaggle/input/twitter-nlp/training.1600000.processed.noemoticon.csv"
IN_PATH_TEST = "/kaggle/input/twitter-nlp/testdata.manual.2009.06.14.csv"
#OUT_PATH_CLEAN = "CLEAN"

spark_reader = spark.read.schema(schema)

user_regex = r"(@\w{1,15})"
hashtag_regex = "(#\w{1,})"
url_regex=r"((https?|ftp|file):\/{2,3})+([-\w+&@#/%=~|$?!:,.]*)|(www.)+([-\w+&@#/%=~|$?!:,.]*)"
email_regex=r"[\w.-]+@[\w.-]+\.[a-zA-Z]{1,}"

@f.udf
def html_unescape(s: str):
    if isinstance(s, str):
        return html.unescape(s)
    return s
```

```
def clean_data(df):
    df = (
        df
        .withColumn("original_text", f.col("text"))
        .withColumn("text", f.regexp_replace(f.col("text"), url_regex, ""))
        .withColumn("text", f.regexp_replace(f.col("text"), email_regex, ""))
        .withColumn("text", f.regexp_replace(f.col("text"), user_regex, ""))
        .withColumn("text", f.regexp_replace(f.col("text"), "#", " "))
        .withColumn("text", html_unescape(f.col("text")))
        .filter("text != ''")
    )
    return df

df_train_raw = spark_reader.csv(IN_PATH_RAW, timestampFormat=timestampformat)
df_train_clean = clean_data(df_train_raw)
df_test_raw = spark_reader.csv(IN_PATH_TEST, timestampFormat=timestampformat)
df_test_clean = clean_data(df_test_raw)


df_train_clean.show(10,True)
df_test_clean.show(10,True)


traindf = (
    df_train_clean
    # Remove all numbers
    .withColumn("text", f.regexp_replace(f.col("text"), "[^a-zA-Z']", " "))
    # Remove all double/multiple spaces
    .withColumn("text", f.regexp_replace(f.col("text"), " +", " "))
    # Remove leading and trailing whitespaces
    .withColumn("text", f.trim(f.col("text")))
    # Ensure we don't end up with empty rows
    .filter("text != ''")
)

traindata = traindf.select("text", "polarity").coalesce(1).cache()


df_test = (
    df_test_clean
    # Remove all numbers
    .withColumn("text", f.regexp_replace(f.col("text"), "[^a-zA-Z']", " "))
    # Remove all double/multiple spaces
    .withColumn("text", f.regexp_replace(f.col("text"), " +", " "))
    # Remove leading and trailing whitespaces
    .withColumn("text", f.trim(f.col("text")))
    # Ensure we don't end up with empty rows
    .filter("text != ''")
)

testdata = df_test.select("text", "polarity").coalesce(1).cache()


%%time
from pyspark.ml.feature import (
    StopWordsRemover,
    Tokenizer,
    HashingTF,
```

```
    IDF,
)
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline

tokenizer = Tokenizer(inputCol="text", outputCol="words1")
stopwords_remover = StopWordsRemover(
    inputCol="words1",
    outputCol="words2",
    stopWords=StopWordsRemover.loadDefaultStopWords("english")
)
hashing_tf = HashingTF(
    inputCol="words2",
    outputCol="term_frequency",
)
idf = IDF(
    inputCol="term_frequency",
    outputCol="features",
    minDocFreq=5,
)
lr = LogisticRegression(labelCol="polarity")


semantic_analysis_pipeline = Pipeline(
    stages=[tokenizer, stopwords_remover, hashing_tf, idf, lr]
)

semantic_analysis_model = semantic_analysis_pipeline.fit(traindata)


semantic_analysis_model.transform(testdata).show()
```