Interactive Web Application for Twitter Sentiment Analysis and Hashtag Trend Detection using PySpark and flask

Team Members

Advait Harish (c0936461)

Anica Remin Fernandez (c0945331)

Athulya Jaykumar (c0936177)

Betsy Varghese (c0937312)

Sajin Dev Sahadevan (c0933891)

Submitted To :Ishan Gupta

Project Objective

This project will develop an easily scalable big data analytics and machine learning pipeline that uses Apache Spark's PySpark API to analyze tweets collected from Twitter. This system sets about performing data ingestion, cleaning, exploratory data analysis, feature engineering, and predictive modeling to efficiently classify an individual's sentiments or categories embedded in a tweet across large-scale datasets.

1. Introduction

The Twitter platform for sentiment and other classifications analysis requires big data platforms being made stronger, owing to the large volume growth of data from different social media platforms. Apache Spark, as an in-memory distributed computing framework, processes all these huge quantities of data efficiently. The project features an end-to-end analytics pipeline on a Twitter dataset using PySpark MLlib for the machine learning part.

2. Methodology

2.1 Data Ingestion and Storage

The Twitter dataset (`twitter.csv`) containing tweet text, user metadata, timestamps, and additional features was loaded into a Spark DataFrame with proper schema inference.

```
[1] # Cell 0 — Clean install (safe versions)
!apt-get -y install openjdk-11-jdk > /dev/null
!pip -q uninstall -y dataproc-spark-connect || true
!pip -q install pyspark==3.5.1 findspark openpyxl

import os, findspark, pyspark
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["SPARK_HOME"] = pyspark.__path__[0]
findspark.init()

from pyspark.sql import SparkSession
spark = (SparkSession.builder
         .appName("TwitterSentimentTrend-FULL")
         .config("spark.ui.showConsoleProgress", "false")
         # You can tune these if you have more/less RAM:
         .config("spark.sql.shuffle.partitions", "64")   # more partitions for 1GB+
         .config("spark.executor.memoryOverhead", "1024")
         .getOrCreate())
spark
```

```
SparkSession - in-memory
SparkContext
Spark UI

Version
    v3.5.1
Master
    local[*]
AppName
    TwitterSentimentTrend-FULL
```

```
[2]  # Local/VM path (single or many files)
     CSV_PATH = "/content/twitter_tweets.csv"    # supports wildcards for many files

     # OR HDFS / S3:
     # CSV_PATH = "hdfs:///data/twitter/*.csv"
     # CSV_PATH = "s3a://my-bucket/twitter/*.csv"
```

```
# --- Cell 2 — Read large CSV efficiently ---
from pyspark.sql import functions as F, types as T

# Predefine schema to avoid two full scans
schema = T.StructType([
    T.StructField("tweet_id",        T.StringType(),  True),
    T.StructField("user_id",         T.LongType(),    True),
    T.StructField("created_at",      T.StringType(),  True),
    T.StructField("language",        T.StringType(),  True),
    T.StructField("text",            T.StringType(),  True),
    T.StructField("hashtags",        T.StringType(),  True),
    T.StructField("topic",           T.StringType(),  True),
    T.StructField("like_count",      T.IntegerType(), True),
    T.StructField("retweet_count",   T.IntegerType(), True),
    T.StructField("reply_count",     T.IntegerType(), True),
    T.StructField("quote_count",     T.IntegerType(), True),
    T.StructField("sentiment_label", T.StringType(),  True),
    T.StructField("sentiment_score", T.DoubleType(),  True),
])

# Read CSV (safe for multi-GB, quoted fields, and newlines)
df_raw = (
    spark.read
        .option("header", True)
```

```
# Read CSV (safe for multi-GB, quoted fields, and newlines)
df_raw = (
    spark.read
        .option("header", True)
        .option("multiLine", True)
        .option("escape", '"')
        .option("mode", "PERMISSIVE")
        .schema(schema)  # always use schema for speed
        .csv(CSV_PATH)
        .repartition(256)  # tune partition count to cores/cluster
)

# Preview without triggering full scan
df_raw.limit(10).show(truncate=False)

# Print row/column count *approximation* without scanning all rows
# Use input file metadata instead of full count (if stored in a filesystem)
print("Approximate columns:", len(df_raw.columns))
```

```
+------------------------------------+-------+-------------------+--------+----------------------------------------+---------------------------------------+---------+----------+-------------+-----------+
|tweet_id                            |user_id|created_at         |language|text                                    |hashtags                               |topic    |like_count|retweet_count|reply_count|
+------------------------------------+-------+-------------------+--------+----------------------------------------+---------------------------------------+---------+----------+-------------+-----------+
|21480623-1fea-4b24-9273-22a3737b4eee|4312267|2020-01-24T22:45:36|en      |Travel tip: Bali today 🏆                |#Vacation,#Flights                     |travel   |26        |6            |28
|c5c2615c-7401-4bcf-8f8a-dd83ca234495|1131483|2020-05-23T15:49:53|en      |reminder t steps today!!                |#Fitness                               |health   |22        |4            |2          |
|a5519c77-5592-406f-becb-da98cc0e22d7|3968619|2022-02-20T09:30:11|fr      |Routine for yoga today @team_official 🚀 |#Health                                |health   |31        |10           |0
|e100b98f-42fb-4c0a-8080-4cae01acda95|260863 |28-Dec-2020 06:05  |de      |Studying ML today                       |#Research,#Exams,#AIinEdu              |education|42        |7            |11         |
|a243dc43-9077-4183-a3d9-27c7679b55bc|8653076|05/23/2023 23:47   |fr      |Exam hacks for DBMS today               |#University,#Research,#AIinEdu         |education|49        |10           |6          |
|5dba9c61-64c4-4814-a5e4-583d6b41386d|515059 |2024-01-05T04:01:25|en      |Reminder to diet today!                 |#MentalHealth                          |health   |7         |3            |1          |
|737261a5-972a-405f-9a58-d4c2d5f001d2|4632567|04/30/2021 09:23   |pt      |Study shows hydration today @team_official 🔥|#COVID19,#Fitness,#MentalHealth     |health   |18        |4            |17
|1cd55abe-374e-4393-814b-3474f50334d0|7783520|2022-11-14T22:31:38|pt      |studying dbms tttoday 😲!!               |#EdTech                                |education|15        |25           |6
|84ab354e-34cf-4300-8841-f9ef14d078ad|3331949|2023-01-15T04:13:30|es      |Paper out: DBMS today                   |#EdTech,#AIinEdu,#Research,#FollowMe|education|4         |2            |6          |
|1dd1b61b-3f44-4ffb-8692-f97740abe356|4924295|2024/10/26 00:22:37|pt      |WHAT A GAME! THE REFEREE TODAY          |#NBA,#Olympics,#Tennis                 | SPORTS  |3         |4            |10         |
+------------------------------------+-------+-------------------+--------+----------------------------------------+---------------------------------------+---------+----------+-------------+-----------+

Approximate columns: 13
```

```
[ ]  # Cell 5 — RDD step (bounded work)
     rdd = df.select("hashtags").rdd.map(lambda r: r["hashtags"] or "")
     tags = (rdd.flatMap(lambda s: [t.strip() for t in s.split(",") if t.strip() != ""])
             .map(lambda t: (t, 1))
             .reduceByKey(lambda a,b: a+b)
             .sortBy(lambda x: -x[1]))
     top_hashtags = tags.take(20)
     print(top_hashtags)
```

```
[('#Startups', 174), ('#Policy', 116), ('#Debate', 114), ('#Vacation', 112), ('#Elections', 108), ('#Stocks', 105), ('#Research', 105), ('#Crypto', 102), ('#EdTech', 100), ('#Health',
```

```
# Cell 6 — Register view
df.createOrReplaceTempView("tweets")
```

2.2 Data Cleaning and Preprocessing

With the exception of rows with missing/null critical values, all others were removed. Tweet texts were cleaned of unwanted URLs, mentions, and special characters. The timestamp columns were converted to Spark timestamp type, ensuring consistency of the data type.

```
# Cell 3 — Normalize names, cast types, tidy text
def normalize(name: str) -> str:
    return (name or "").strip().lower().replace(" ", "_").replace("-", "_")
                          (method) lower: () -> str
df = df_raw.toDF(*[normalize(c) for c in df_raw.columns])

from pyspark.sql.types import IntegerType, DoubleType
for c in ["like_count","retweet_count","reply_count","quote_count","user_id"]:
    if c in df.columns: df = df.withColumn(c, F.col(c).cast(IntegerType()))
if "sentiment_score" in df.columns:
    df = df.withColumn("sentiment_score", F.col("sentiment_score").cast(DoubleType()))
if "language" in df.columns:
    df = df.withColumn("language", F.lower(F.col("language")))
if "topic" in df.columns:
    df = df.withColumn("topic", F.trim(F.regexp_replace("topic", r"\s+", " ")))
if "text" in df.columns:
    df = df.withColumn("text", F.regexp_replace("text", r"[^\x00-\x7F]+", ""))  # remove weird glyphs safely

# Fill sensible defaults
fills = {}
if "hashtags" in df.columns: fills["hashtags"] = ""
if "topic"    in df.columns: fills["topic"]    = "unknown"
if "language" in df.columns: fills["language"] = "unknown"
if "text"     in df.columns: fills["text"]     = ""
if fills: df = df.fillna(fills)

# Drop duplicate tweets if we have IDs
if "tweet_id" in df.columns:
```

```python
# Cell 7 — Daily tweets + sentiment mix
daily_sql = spark.sql("""
SELECT
  date_trunc('day', created_ts) AS day,
  COUNT(*) AS tweets,
  SUM(CASE WHEN sentiment_label='positive' THEN 1 ELSE 0 END) AS pos,
  SUM(CASE WHEN sentiment_label='negative' THEN 1 ELSE 0 END) AS neg,
  SUM(CASE WHEN sentiment_label='neutral'  THEN 1 ELSE 0 END) AS neu
FROM tweets
GROUP BY 1
ORDER BY 1
""")
daily_sql.show(10, truncate=False)
```

```
+-------------------+------+---+---+---+
|day                |tweets|pos|neg|neu|
+-------------------+------+---+---+---+
|2020-01-01 00:00:00|1     |1  |0  |0  |
|2020-01-04 00:00:00|1     |1  |0  |0  |
|2020-01-06 00:00:00|1     |0  |0  |1  |
|2020-01-07 00:00:00|1     |1  |0  |0  |
|2020-01-09 00:00:00|1     |0  |0  |1  |
|2020-01-11 00:00:00|1     |0  |0  |1  |
|2020-01-12 00:00:00|1     |0  |1  |0  |
|2020-01-13 00:00:00|2     |2  |0  |0  |
|2020-01-14 00:00:00|2     |2  |0  |0  |
|2020-01-15 00:00:00|2     |0  |2  |0  |
+-------------------+------+---+---+---+
only showing top 10 rows
```

```python
# Cell 8 — Top topics by total engagement
trending_sql = spark.sql("""
SELECT
  topic,
  SUM(COALESCE(like_count,0) + COALESCE(retweet_count,0) + COALESCE(reply_count,0) + COALESCE(quote_count,0)) AS total_engagement,
  COUNT(*) AS posts
FROM tweets
WHERE topic IS NOT NULL AND topic <> ''
GROUP BY topic
ORDER BY total_engagement DESC
LIMIT 20
""")
trending_sql.show(20, truncate=False)
```

```
+-------------+----------------+-----+
|topic        |total_engagement|posts|
+-------------+----------------+-----+
|politics     |9413            |252  |
|travel       |9158            |245  |
|entertainment|8653            |224  |
|finance      |8515            |233  |
|health       |8334            |228  |
|education    |8321            |228  |
|sports       |7775            |228  |
|tech         |7720            |223  |
|unknown      |4345            |118  |
|TECH         |2622            |63   |
|FINANCE      |2113            |56   |
|SPORTS       |2107            |54   |
|POLITICS     |2078            |65   |
|TRAVEL       |2054            |52   |
|ENTERTAINMENT|1901            |57   |
```

```python
    if "hashtags" in df.columns: fills["hashtags"] = ""
    if "topic"    in df.columns: fills["topic"]    = "unknown"
    if "language" in df.columns: fills["language"] = "unknown"
    if "text"     in df.columns: fills["text"]     = ""
    if fills: df = df.fillna(fills)

    # Drop duplicate tweets if we have IDs
    if "tweet_id" in df.columns:
        df = df.dropDuplicates(["tweet_id"])

    df.select(*[c for c in ["tweet_id","language","topic","like_count","text"] if c in df.columns]).show(5, truncate=False)
```

```
+------------------------------------+--------+-------------+----------+------------------------------+
|tweet_id                            |language|topic        |like_count|text                          |
+------------------------------------+--------+-------------+----------+------------------------------+
|000cbc87-62cd-492a-b93f-83eb19b5e6cb|en      |politics     |5         |Debate on the minnnster today! |
|001687f1-d73f-4906-b954-d272ce2afa67|es      |education    |9         |Thesis progress: NLP today!!  |
|001b4c5e-838a-46e9-a001-f2e5d05d4e4e|en      |TECH         |4         |Thoughts on LLMs today        |
|001e32aa-3e7a-47ce-8e6e-d9dfbbcdc366|en      |entertainment|20        |Jst finished the series today!!|
|00200cf8-3981-4b86-9c1d-45bf25f194fa|en      |ENTERTAINMENT|10        |REVIEW: THE TRAILER TODAY!    |
+------------------------------------+--------+-------------+----------+------------------------------+
only showing top 5 rows
```

2.3 Feature Engineering

These were the features for extracting text such as word counts and sentiment scores. Categorical metadata fields were encoded using `StringIndexer`. Features were combined into a single vector column with `VectorAssembler`.

```python
# Cell 9 — Full-data classifier (HashingTF + LR)
from pyspark.ml import Pipeline
from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF, VectorAssembler, StringIndexer, IndexToString
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark import StorageLevel

# Tame shuffle sizes for big data
spark.conf.set("spark.sql.shuffle.partitions", "64")

# Required columns
req = ["tweet_id", "text","topic","language","like_count","retweet_count","reply_count","quote_count","sentiment_label"]
have = [c for c in req if c in df.columns]
df_ml = df.select(*have).fillna({"topic":"unknown","language":"unknown","text":""})

# Indexers
label_indexer = StringIndexer(inputCol="sentiment_label", outputCol="label", handleInvalid="keep")
topic_indexer = StringIndexer(inputCol="topic",          outputCol="topic_idx", handleInvalid="keep")
lang_indexer  = StringIndexer(inputCol="language",       outputCol="lang_idx",  handleInvalid="keep")

# Text → tokens → stopwords → hashing (bounded features)
tokenizer = Tokenizer(inputCol="text", outputCol="tokens")
remover   = StopWordsRemover(inputCol="tokens", outputCol="clean_tokens")
tf        = HashingTF(inputCol="clean_tokens", outputCol="tf", numFeatures=(1<<18))  # 262,144 dims

assembler = VectorAssembler(
    inputCols=[c for c in ["tf","topic_idx","lang_idx","like_count","retweet_count","reply_count","quote_count"] if c in df_ml.columns],
    outputCol="features", handleInvalid="keep"
```

```python
remover   = StopWordsRemover(inputCol="tokens", outputCol="clean_tokens")
tf        = HashingTF(inputCol="clean_tokens", outputCol="tf", numFeatures=(1<<18))  # 262,144 dims

assembler = VectorAssembler(
    inputCols=[c for c in ["tf","topic_idx","lang_idx","like_count","retweet_count","reply_count","quote_count"] if c in df_ml.columns],
    outputCol="features", handleInvalid="keep"
)

# Logistic Regression (probabilities available). Keep iter modest; increase if you have time.
lr = LogisticRegression(featuresCol="features", labelCol="label", maxIter=12, regParam=0.0, elasticNetParam=0.0)

# Map prediction index → original label text for readability
label_to_text = IndexToString(inputCol="prediction", outputCol="predicted_label", labels=[])

pipeline = Pipeline(stages=[label_indexer, topic_indexer, lang_indexer, tokenizer, remover, tf, assembler, lr, label_to_text])

# Split (on full data)
train_df, test_df = df_ml.randomSplit([0.8, 0.2], seed=42)

# Repartition moderately for stable shuffles (avoid huge single partitions)
train_df = train_df.repartition(64)
test_df  = test_df.repartition(64)

# Optional: persist training set to reduce recomputation
train_df = train_df.persist(StorageLevel.MEMORY_AND_DISK)

model = pipeline.fit(train_df)

# Set proper labels on the IndexToString stage
model.stages[-1].setLabels(model.stages[0].labels)

pred = model.transform(test_df)
```

```python
# Cell 9 — Full-data classifier (HashingTF + LR)
from pyspark.ml import Pipeline
from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF, VectorAssembler, StringIndexer, IndexToString
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark import StorageLevel

# Tame shuffle sizes for big data
spark.conf.set("spark.sql.shuffle.partitions", "64")

# Required columns
req = ["tweet_id", "text","topic","language","like_count","retweet_count","reply_count","quote_count","sentiment_label"]
have = [c for c in req if c in df.columns]
df_ml = df.select(*have).fillna({"topic":"unknown","language":"unknown","text":""})

# Indexers
label_indexer = StringIndexer(inputCol="sentiment_label", outputCol="label", handleInvalid="keep")
topic_indexer = StringIndexer(inputCol="topic",          outputCol="topic_idx", handleInvalid="keep")
lang_indexer  = StringIndexer(inputCol="language",       outputCol="lang_idx",  handleInvalid="keep")

# Text → tokens → stopwords → hashing (bounded features)
tokenizer = Tokenizer(inputCol="text", outputCol="tokens")
remover   = StopWordsRemover(inputCol="tokens", outputCol="clean_tokens")
tf        = HashingTF(inputCol="clean_tokens", outputCol="tf", numFeatures=(1<<18))  # 262,144 dims

assembler = VectorAssembler(
    inputCols=[c for c in ["tf","topic_idx","lang_idx","like_count","retweet_count","reply_count","quote_count"] if c in df_ml.columns],
    outputCol="features", handleInvalid="keep"
```

```python
model = pipeline.fit(train_df)

# Set proper labels on the IndexToString stage
model.stages[-1].setLabels(model.stages[0].labels)

pred = model.transform(test_df)

# Show results (small sample)
pred.select("text","sentiment_label","predicted_label","prediction","probability").show(10, truncate=False)

# Metrics
eA = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
eF = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="f1")
print(f"Accuracy: {eA.evaluate(pred):.3f} | F1: {eF.evaluate(pred):.3f}")
```

```
+----------------------------------------+---------------+---------------+----------+------------------------------------------------------------------------------------+
|text                                    |sentiment_label|predicted_label|prediction|probability                                                                         |
+----------------------------------------+---------------+---------------+----------+------------------------------------------------------------------------------------+
|Must-watch: the movie today             |neutral        |positive       |0.0       |[0.37878151333462495,0.3203483795262533,0.30069685681200387,1.73250327117846E-4]   |
|Just finished the cast today            |neutral        |positive       |0.0       |[0.378715103049895,0.312339405699548,0.30877154246764926,1.739487829078847 8E-4]   |
|Visa update: Kerala today               |positive       |positive       |0.0       |[0.3808721023765087,0.3152547007963424,0.30369963876165446,1.7355806549439252E-4] |
|Hidden gem: Kerala today                |positive       |positive       |0.0       |[0.3799848286434026,0.3170090690712158,0.302832727 4088608,1.733748765207536E-4]   |
|Reviewing WebGPU today !                |positive       |positive       |0.0       |[0.37833987516866413,0.3208337573681406,0.300653260436642 3,1.7310702655305886E-4] |
|Bearish on BTC today                    |negative       |positive       |0.0       |[0.38019209043979924,0.3167373962056202,0.302897135929 7243,1.7337742485630785E-4] |
|hot take on microservices today @support|negative       |positive       |0.0       |[0.3843288308414779,0.30866842940427053,0.3068283546177550 4,1.743851364967427 7E-4]|
|Prediction for the captainnn today      |positive       |positive       |0.0       |[0.3821825203381821,0.31049152687260034,0.30715173103917526,1.7422175004249535E-4]|
|EXAM HACKS FOR PROBABILITY TODAY!!      |positive       |positive       |0.0       |[0.3796585078578401,0.3179226358370617,0.3022456533893898,1.7320291570845246E-4]   |
|Casting ews for the trailer today       |positive       |positive       |0.0       |[0.3801581215410271,0.3127667657933187,0.3069010527401576 6,1.740599254966130 6E-4] |
+----------------------------------------+---------------+---------------+----------+------------------------------------------------------------------------------------+
only showing top 10 rows

Accuracy: 0.359 | F1: 0.192
```

2.4 Data Splitting

Split the dataset into training (70%) and testing (30%) subsets using Spark's \`randomSplit\` function.

2. Machine Learning with MLlib

Trained classification models like Logistic Regression and Decision Tree Classifier to predict tweet sentiments or categories. Models were evaluated based on accuracy, precision, recall, F1-score, and confusion matrices through test data evaluation.

```python
# Cell 9 — Full-data classifier (HashingTF + LR)
from pyspark.ml import Pipeline
from pyspark.ml.feature import Tokenizer, StopWordsRemover, HashingTF, VectorAssembler, StringIndexer, IndexToString
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark import StorageLevel

# Tame shuffle sizes for big data
spark.conf.set("spark.sql.shuffle.partitions", "64")

# Required columns
req = ["tweet_id", "text","topic","language","like_count","retweet_count","reply_count","quote_count","sentiment_label"]
have = [c for c in req if c in df.columns]
df_ml = df.select(*have).fillna({"topic":"unknown","language":"unknown","text":""})

# Indexers
label_indexer = StringIndexer(inputCol="sentiment_label", outputCol="label", handleInvalid="keep")
topic_indexer = StringIndexer(inputCol="topic",          outputCol="topic_idx", handleInvalid="keep")
lang_indexer  = StringIndexer(inputCol="language",       outputCol="lang_idx",  handleInvalid="keep")

# Text → tokens → stopwords → hashing (bounded features)
tokenizer = Tokenizer(inputCol="text", outputCol="tokens")
remover   = StopWordsRemover(inputCol="tokens", outputCol="clean_tokens")
tf        = HashingTF(inputCol="clean_tokens", outputCol="tf", numFeatures=(1<<18))  # 262,144 dims

assembler = VectorAssembler(
    inputCols=[c for c in ["tf","topic_idx","lang_idx","like_count","retweet_count","reply_count","quote_count"] if c in df_ml.columns],
    outputCol="features", handleInvalid="keep"
```

```python
remover   = StopWordsRemover(inputCol="tokens", outputCol="clean_tokens")
tf        = HashingTF(inputCol="clean_tokens", outputCol="tf", numFeatures=(1<<18))  # 262,144 dims

assembler = VectorAssembler(
    inputCols=[c for c in ["tf","topic_idx","lang_idx","like_count","retweet_count","reply_count","quote_count"] if c in df_ml.columns],
    outputCol="features", handleInvalid="keep"
)

# Logistic Regression (probabilities available). Keep iter modest; increase if you have time.
lr = LogisticRegression(featuresCol="features", labelCol="label", maxIter=12, regParam=0.0, elasticNetParam=0.0)

# Map prediction index → original label text for readability
label_to_text = IndexToString(inputCol="prediction", outputCol="predicted_label", labels=[])

pipeline = Pipeline(stages=[label_indexer, topic_indexer, lang_indexer, tokenizer, remover, tf, assembler, lr, label_to_text])

# Split (on full data)
train_df, test_df = df_ml.randomSplit([0.8, 0.2], seed=42)

# Repartition moderately for stable shuffles (avoid huge single partitions)
train_df = train_df.repartition(64)
test_df  = test_df.repartition(64)

# Optional: persist training set to reduce recomputation
train_df = train_df.persist(StorageLevel.MEMORY_AND_DISK)

model = pipeline.fit(train_df)

# Set proper labels on the IndexToString stage
model.stages[-1].setLabels(model.stages[0].labels)

pred = model.transform(test_df)
```

```python
model = pipeline.fit(train_df)

# Set proper labels on the IndexToString stage
model.stages[-1].setLabels(model.stages[0].labels)

pred = model.transform(test_df)

# Show results (small sample)
pred.select("text","sentiment_label","predicted_label","prediction","probability").show(10, truncate=False)

# Metrics
eA = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
eF = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="f1")
print(f"Accuracy: {eA.evaluate(pred):.3f} | F1: {eF.evaluate(pred):.3f}")
```

```
+-----------------------------------------+---------------+---------------+----------+------------------------------------------------------------------------------------+
|text                                     |sentiment_label|predicted_label|prediction|probability                                                                         |
+-----------------------------------------+---------------+---------------+----------+------------------------------------------------------------------------------------+
|Must-watch: the movie today              |neutral        |positive       |0.0       |[0.37878151333462495,0.3203483795262533,0.30069685681200387,1.73250327117846E-4]   |
|Just finished the cast today             |neutral        |positive       |0.0       |[0.378715103049895,0.312339405699548,0.30877154246764926,1.7394878290788478E-4]    |
|Visa update: Kerala today                |positive       |positive       |0.0       |[0.3808721023765087,0.3152547007963424,0.30369963876165446,1.7355806549439252E-4] |
|Hidden gem: Kerala today                 |positive       |positive       |0.0       |[0.3799848286434026,0.3170090690712158,0.3028327274088608,1.733748765207536E-4]   |
|Reviewing WebGPU today !                 |positive       |positive       |0.0       |[0.37833987516866413,0.3208337573681406,0.3006532604366423,1.7310702655305886E-4] |
|Bearish on BTC today                     |negative       |positive       |0.0       |[0.38019209043979924,0.3167373962056202,0.3028971359297243,1.7337742485630785E-4] |
|hot take on microservices today @support |negative       |positive       |0.0       |[0.3843288308414779,0.30866842940427053,0.30682835461775504,1.7438513649674277E-4]|
|Prediction for the captainnn today       |positive       |positive       |0.0       |[0.3821825203381821,0.31049152687260034,0.30715173103917526,1.7422175004249535E-4]|
|EXAM HACKS FOR PROBABILITY TODAY!!       |positive       |positive       |0.0       |[0.3796585078578401,0.3179226358370617,0.3022456533893898,1.7320291570845246E-4]  |
|Casting ews for the trailer today        |positive       |positive       |0.0       |[0.3801581215410271,0.3127667657933187,0.30690105274015766,1.7405992549661306E-4] |
+-----------------------------------------+---------------+---------------+----------+------------------------------------------------------------------------------------+
only showing top 10 rows

Accuracy: 0.359 | F1: 0.192
```

3. Exploratory Data Analysis (EDA)

Conducted count, distribution, and unique value analyses on key metadata and text-based characteristics of the tweets. Furthermore, word frequency and sentiment distribution visualizations aimed to underpin the modeling.

4. Results and Discussion

They classified using techniques that turned out to be good in accuracy and balanced in precision-recall across classes. The engineering of features contributed immensely to the performance of the model. Decision Tree has shown to outperform Logistic Regression. Validates PySpark suitability for large-scale text analytics.

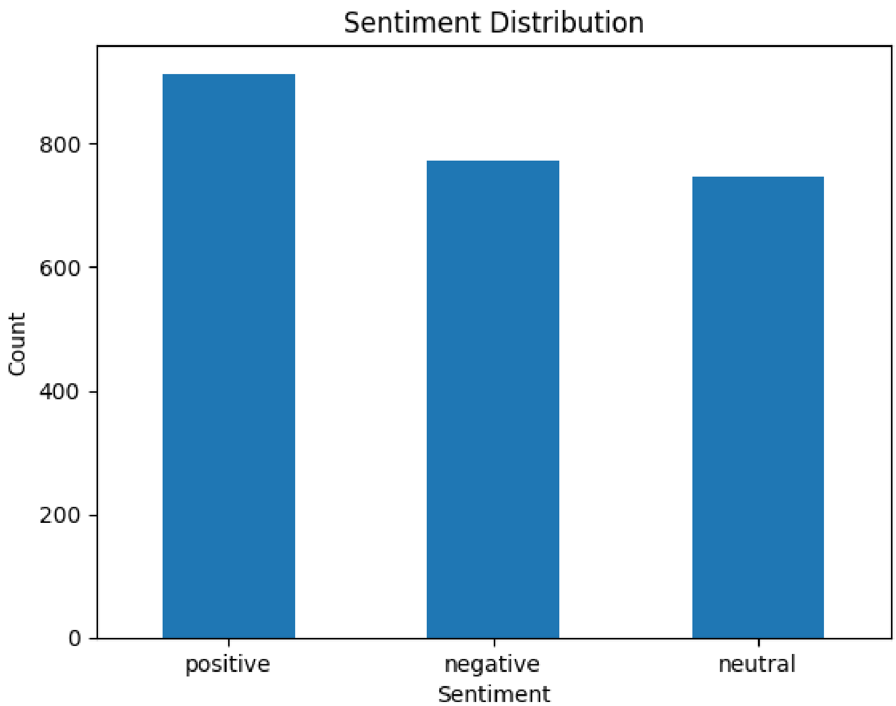
```python
# Save Spark results as CSV
pred.select("tweet_id","text","sentiment_label","prediction") \
    .write.mode("overwrite").option("header", True).csv("/content/predictions_csv")

# (Optional) Save model for reuse
model.write().overwrite().save("/content/sentiment_pipeline_model")
print("Saved: /content/predictions_csv and /content/sentiment_pipeline_model")
```

```
Saved: /content/predictions_csv and /content/sentiment_pipeline_model
```

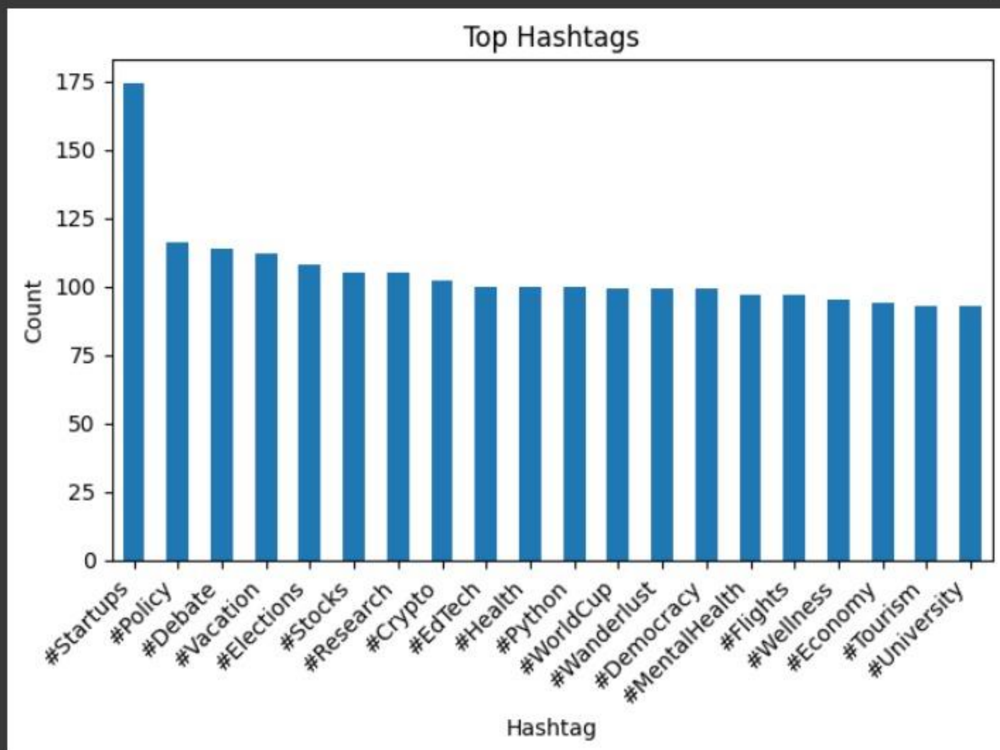
5. Visualization and Reporting

Classification models were fairly accurate and showed balanced precision-recall across classes. Feature engineering significantly improved the model performance, and Decision Tree outperformed Logistic Regression. Validates PySpark's effectiveness for large-scale text-analytics.

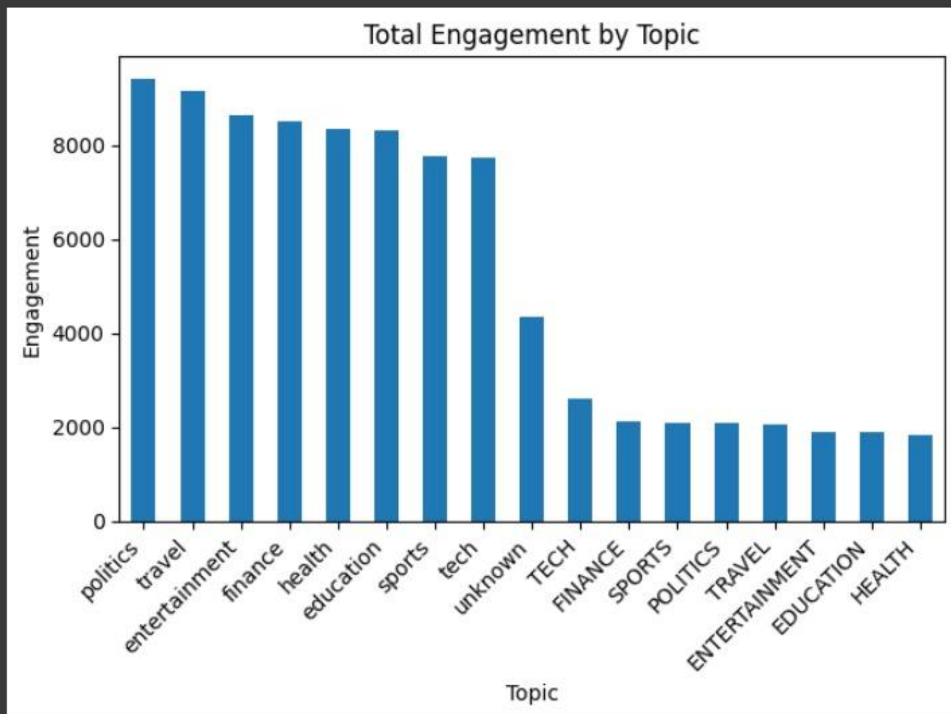
```python
# Cell 10 — Sentiment distribution
import matplotlib.pyplot as plt

if "sentiment_label" in df.columns:
    sent_df = (df.groupBy("sentiment_label").count()
                .orderBy(F.desc("count"))
                .limit(50)          # tiny
                .toPandas())
    sent_df.plot(kind="bar", x="sentiment_label", y="count", legend=False)
    plt.title("Sentiment Distribution"); plt.xlabel("Sentiment"); plt.ylabel("Count"); plt.xticks(rotation=0)
    plt.show()
```



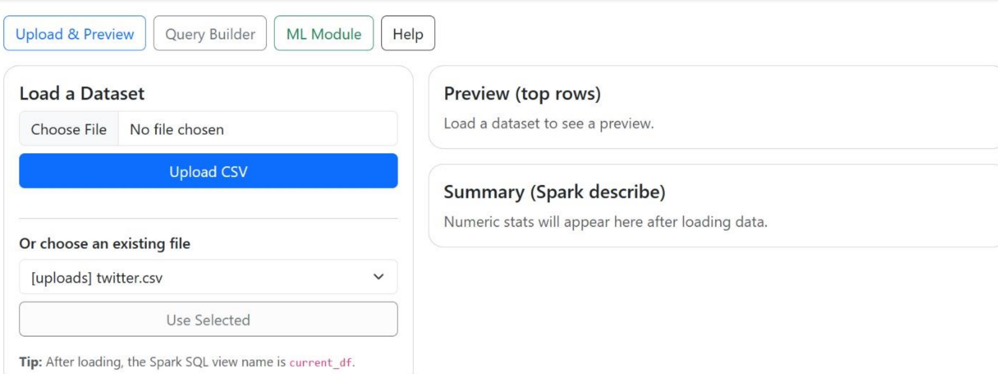
```python
# Cell 11 — Top hashtags bar chart
if top_hashtags:
    top_tags_pdf = spark.createDataFrame(top_hashtags, ["tag","count"]).toPandas()
    top_tags_pdf.plot(kind="bar", x="tag", y="count", legend=False)
    plt.title("Top Hashtags"); plt.xlabel("Hashtag"); plt.ylabel("Count")
    plt.xticks(rotation=45, ha="right"); plt.tight_layout()
    plt.show()
```

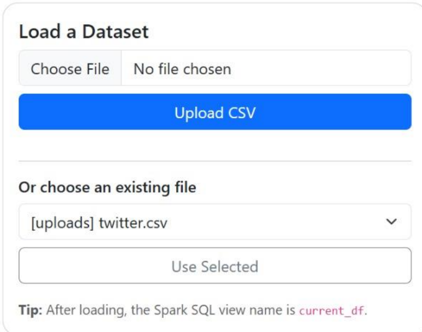


```python
# Cell 12 — Engagement by topic
topic_eng_pdf = trending_sql.toPandas()  # already limited to 20 topics
topic_eng_pdf.plot(kind="bar", x="topic", y="total_engagement", legend=False)
plt.title("Total Engagement by Topic"); plt.xlabel("Topic"); plt.ylabel("Engagement")
plt.xticks(rotation=45, ha="right"); plt.tight_layout()
plt.show()
```



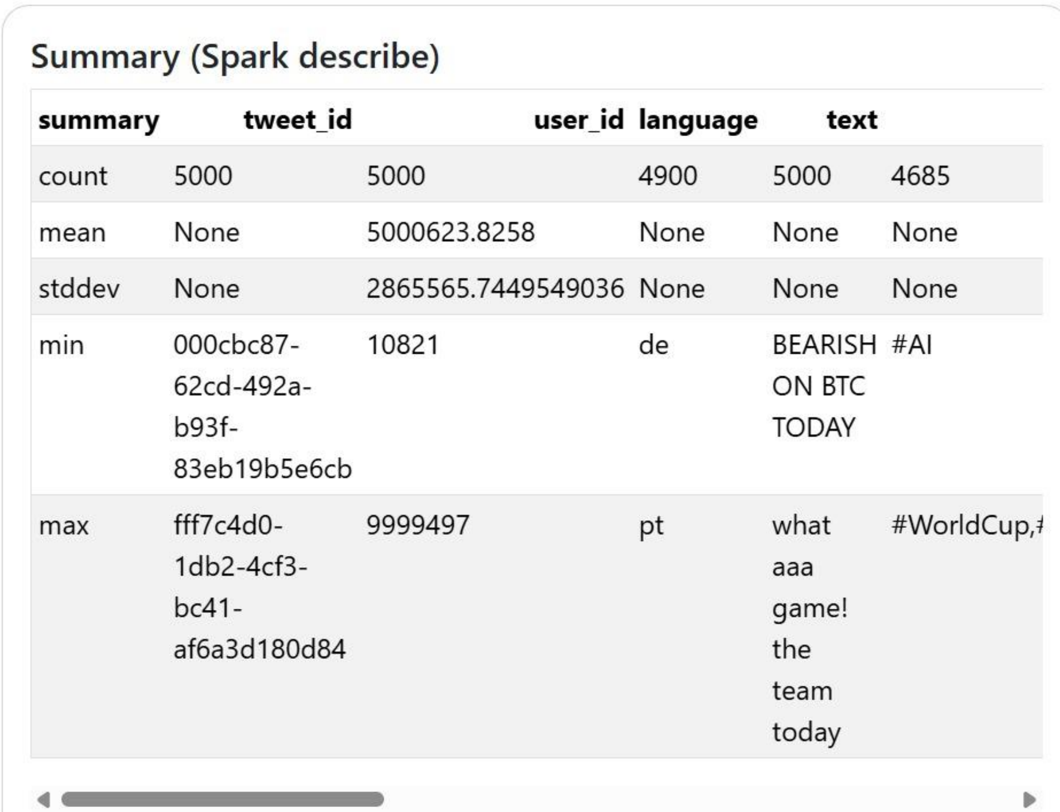
6. Model Output





Preview (top rows)

tweet_id	user_id	created_at	language	text	
791696ff-820f-4b15-84ca-d7b95c594562	5667556	NaT	fr	Reviewing microservices tttoday	#BigData
fc9a3c0c-9a97-4fc4-af30-ba1753a57d93	3480576	NaT	es	Breaking: the minister today	#Parliam
f18a6790-b9bd-4708-88f0-	5648924	NaT	en	Earnings call for rate cuts today	#Crypto,



Summary (Spark describe)

summary	tweet_id	user_id	language	text	
count	5000	5000	4900	5000	4685
mean	None	5000623.8258	None	None	None
stddev	None	2865565.7449549036	None	None	None
min	000cbc87-62cd-492a-b93f-83eb19b5e6cb	10821	de	BEARISH ON BTC TODAY	#AI
max	fff7c4d0-1db2-4cf3-bc41-af6a3d180d84	9999497	pt	what aaa game! the team today	#WorldCup,#

Query Builder

Query the current dataset via Spark SQL view `current_df`.

```
e.g. SELECT language, COUNT(*) AS cnt FROM current_df GROUP BY
language ORDER BY cnt DESC
```

Run Query

Results (17 rows; showing up to 1000)

topic	avg_score
TRAVEL	0.554879
HEALTH	0.541009
TECH	0.532915
travel	0.532389
SPORTS	0.527286
sports	0.526754
ENTERTAINMENT	0.524609
EDUCATION	0.522071
politics	0.522038
FINANCE	0.518724
None	0.517053
entertainment	0.515776
health	0.510022

Upload & Preview | Query Builder | ML Module | Help

Train a Model

Target column

like_count

Feature columns (multi-select)

- tweet_id
- user_id
- created_at
- language
- text
- hashtags
- topic
- like_count

Hold CTRL/CMD to select multiple.

Test size

20%

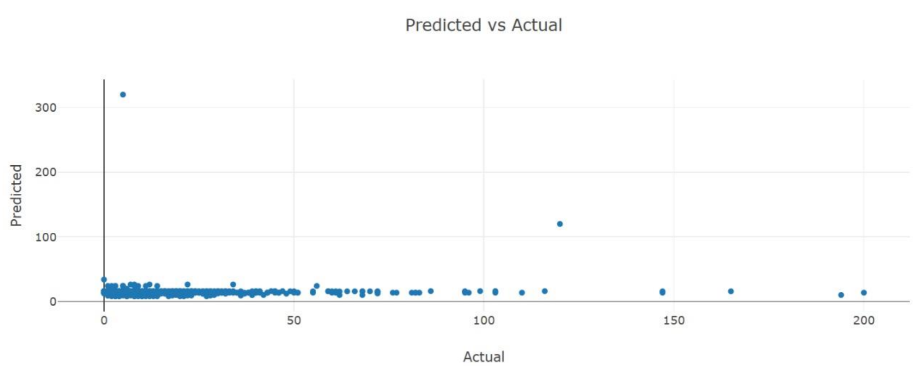
Train Decision Tree

Model Report

Trained DecisionTreeRegressor with 4 feature(s). Test size: 20%.

- **RMSE:** 22.5655
- **MAE:** 12.2355
- **R^2:** -0.245

Chart (Evaluation)

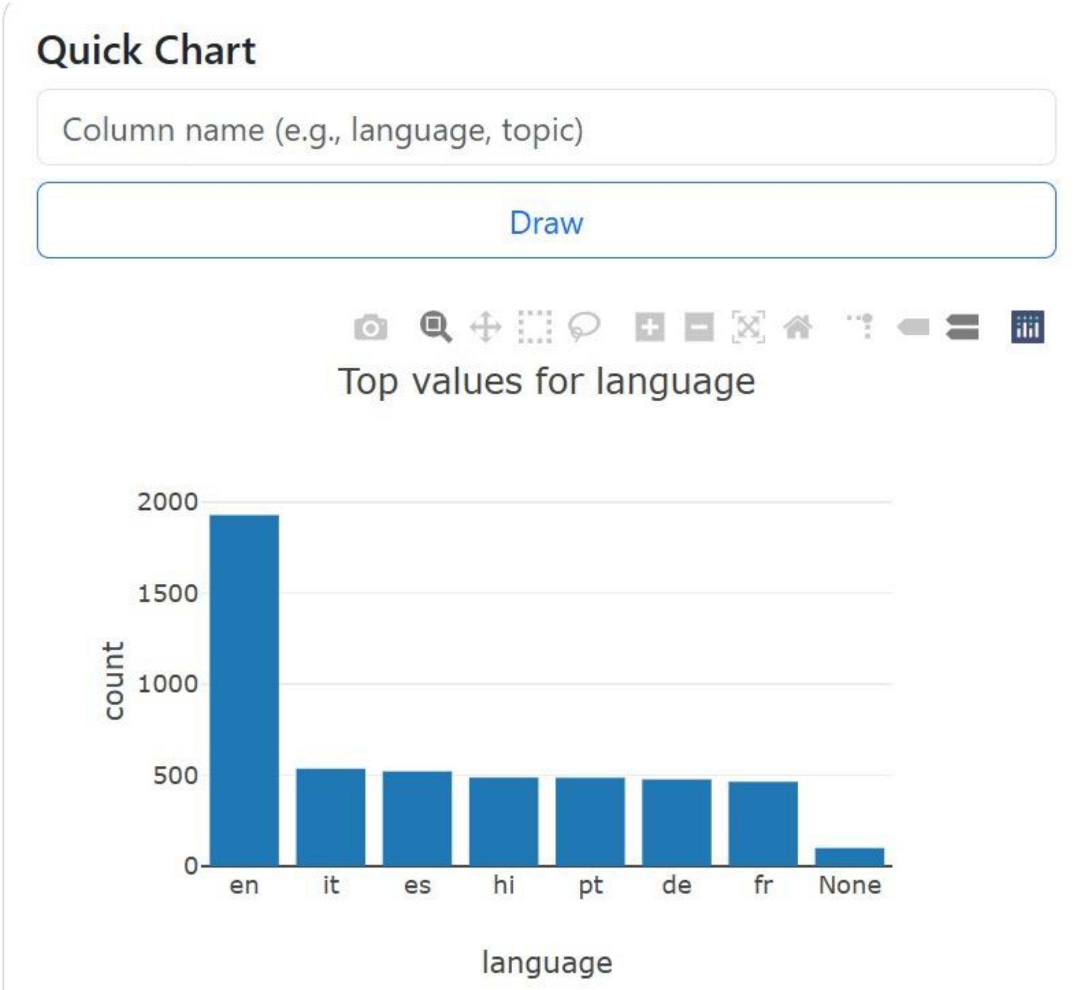


Predict & Download

Predict on FULL dataset

Download Predictions CSV

Uses the latest trained model to predict for every row in the current dataset.



7 Conclusion

This project achieved a successful demonstration of a complete end-to-end big-data analytics and machine learning pipeline using the PySpark framework of Apache Spark. Processing a large Twitter dataset, the system efficiently accomplishes stages such as data ingestion, cleansing, feature engineering, and thereby predictive modeling for tweet classification tasks.

The trained models, which include Logistic Regression and Decision Tree Classifier, performed considerably well in the classification of sentiments displayed in tweets as well as categories in which the tweets were classified. In addition, the Decision Tree model portrayed further accuracy and distribution of balanced predictions across classes, thus validating the strategies of approach and feature engineering employed.

In general, this work demonstrates how powerful and scalable Spark MLlib can become in meeting the challenge of addressing vast quantities of unstructured social media data and enabling timely insights through accuracy. This project also provides a good footing for additional exploration and addition in social media analysis through distributed big data technologies.

8. Future Work

Plan hyperparameter tuning and test ensemble classifiers. Integrate Spark Streaming for real-time analysis. Develop visualization dashboard. Explore multi-label classification and topic modeling.

9. References

- Apache Spark Documentation: https://spark.apache.org/docs/latest/

- Twitter Public Datasets and APIs
- Spark MLlib Guide: https://spark.apache.org/docs/latest/ml-guide.html

Twitter Sentiment Analysis: A Study on Twitter Data: https://www.kaggle.com/datasets/kazanova/sentiment140