# Welcome to your new notebook

# Type here in the cell editor to add code!

%pip install requests

%pip install TextBlob

**Step 1: Data Collection**

# Import Libraries

import requests

from pyspark.sql import SparkSession

# Initialize Spark session

spark = SparkSession.builder.appName("NewsSentimentAnalysis").getOrCreate()

# Your API key

api\_key = "apykey"

url = f"https://newsapi.org/v2/everything?q=global&language=en&apiKey={api\_key}"

# Fetch data from NewsAPI

response = requests.get(url)

data = response.json()

# Extract relevant fields from articles

articles = [

    {

        "title": article["title"],

        "content": article["content"],

        "source": article["source"]["name"],

        "author": article.get("author"),

        "published\_date": article.get("publishedAt"),

        "url": article.get("url"),

        "summary": article.get("description")

    }

    for article in data["articles"]

]

# Convert the list of articles to a Spark DataFrame

df = spark.createDataFrame(articles)

# Show the collected data

df.show()

# Overwrite the existing Delta table in the "Tables" folder with the current DataFrame

# If there are schema differences, merge the new schema with the existing one

df.write.format("delta").option("mergeSchema", "true").mode("overwrite").save("Tables/news\_data")

**Step 2: Data Cleaning and Preparation**

# Import necessary functions

from pyspark.sql.functions import col, lower, regexp\_replace, to\_date, lit, when, sum as \_sum

# Load data from Delta table

df = spark.read.format("delta").load("Tables/news\_data")

# Drop duplicate entries based on 'title' and 'content'

df = df.dropDuplicates(["title", "content"])

# Drop rows where 'title' or 'content' are null

df = df.na.drop(subset=["title", "content"])

# Standardize text: Convert to lowercase and remove special characters

df = df.withColumn("title", lower(col("title")))

df = df.withColumn("content", lower(col("content")))

df = df.withColumn("content", regexp\_replace(col("content"), "[^a-zA-Z0-9\\s]", ""))

# Convert 'published\_date' to DateType

df = df.withColumn("published\_date", to\_date(col("published\_date")))

# Handle missing values for 'author' and 'summary' columns

df = df.withColumn("author", when(col("author").isNull(), lit("Unknown")).otherwise(col("author")))

df = df.withColumn("summary", when(col("summary").isNull(), lit("No summary available")).otherwise(col("summary")))

# Write cleaned data back to Delta table

df.write.format("delta").mode("overwrite").save("Tables/cleaned\_news\_data")

# Load cleaned data for further processing

df = spark.read.format("delta").load("Tables/cleaned\_news\_data")

# Show a sample of the cleaned data

df.show(truncate=False)

# Print schema validation

df.printSchema()

# Check for missing values across all columns (only once, eliminating redundancy)

null\_counts = df.select([\_sum(col(c).isNull().cast("int")).alias(c) for c in df.columns])

null\_counts.show()

**Step 3: Sentiment Analysis**

#Using TextBlob Library to analyze sentiment for text columns (e.g., title or summary).

from textblob import TextBlob

from pyspark.sql.functions import col

# Define a function to calculate sentiment

def get\_sentiment(text):

    return TextBlob(text).sentiment.polarity

# Apply the function to the 'summary' column

from pyspark.sql.functions import udf

from pyspark.sql.types import FloatType

sentiment\_udf = udf(get\_sentiment, FloatType())

df = df.withColumn("sentiment\_score", sentiment\_udf(col("summary")))

#Step 4: Categorize Sentiment

#You can classify sentiment into categories like Positive, Neutral, or Negative based on the sentiment score:

from pyspark.sql.functions import when

df = df.withColumn(

    "sentiment\_category",

    when(df.sentiment\_score > 0.1, "Positive")

    .when(df.sentiment\_score < -0.1, "Negative")

    .otherwise("Neutral")

)

# Step 5: Inspect Results

# Preview the DataFrame to validate the added sentiment columns:

df.select("title", "summary", "sentiment\_score", "sentiment\_category").show(truncate=False)

#Step 6: Save the Data

#Store the enriched DataFrame (with sentiment scores) back into a Delta table for persistence:

df.write.format("delta").mode("overwrite").save("Tables/news\_with\_sentiment")

**Step 4: Data Storage**

# Store the processed data and sentiment scores in an Azure SQL Database for persistent storage and querying

import pyodbc

import pandas as pd

#from pyspark.sql import SparkSession

# Initialize Spark session

#spark = SparkSession.builder.appName("SentimentAnalysis").getOrCreate()

#df = spark.read.format("delta").load("Tables/news\_with\_sentiment")

df.show()

# Define the connection string

conn\_str = (

    "jdbc:sqlserver://server\_name:1433;"

    "database=Sentiment\_db;user=usernane;"

    "password=password;"

    "encrypt=true;"

    "trustServerCertificate=false;"

    "hostNameInCertificate=\*.database.windows.net;"

    "loginTimeout=30;"

)

# Write DataFrame to Azure SQL Database

df.write.format("jdbc") \

    .option("url", conn\_str) \

    .option("dbtable", "news\_with\_sentiment") \

    .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \

    .mode("append") \

    .save()