bda-implementation

April 2, 2025

[1]:
```
# First import SparkSession
from pyspark.sql import SparkSession

# Then create the Spark session
spark = SparkSession.builder \
    .appName("TweetAnalysis") \
    .config("spark.hadoop.io.compression.codecs", "org.apache.hadoop.io.
 ↪compress.GzipCodec") \
    .getOrCreate()

# Now you can use Spark
print("Spark session created successfully")
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
25/04/02 18:42:18 WARN NativeCodeLoader: Unable to load native-hadoop library
for your platform… using builtin-java classes where applicable
```

```
Spark session created successfully
```

[2]:
```
spark = SparkSession.builder \
    .config("spark.hadoop.io.compression.codecs", "org.apache.hadoop.io.
 ↪compress.GzipCodec") \
    .getOrCreate()
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("TweetAnalysis").getOrCreate()
df = spark.read.csv("hdfs://master-node:9000/user/dinesh/tweets_data_sample/
 ↪bda_train.csv", header=True, inferSchema=True)
#print(df)
df.show(5)
```

```
+---+-------+--------+--------------------+------+
| id|keyword|location|                text|target|
+---+-------+--------+--------------------+------+
|  1|   NULL|    NULL|Our Deeds are the…|     1|
|  4|   NULL|    NULL|Forest fire near …|     1|
```

```
|  5|   NULL|    NULL|All residents ask…|     1|
|  6|   NULL|    NULL|13,000 people rec…|     1|
|  7|   NULL|    NULL|Just got sent thi…|     1|
+---+-------+--------+------------------+------+
only showing top 5 rows
```

[2]:
```
print("Dinesh")
```

```
Dinesh
```

[3]:
```
import emoji
import pandas as pd

#Text processing libraries
import re
import string
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

#sklearn
from sklearn import model_selection
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics

#Libraries for plotting
import seaborn as sns

#Modules for plotting
from matplotlib import pyplot as plt
import geopandas as gpd
from shapely.geometry import Point , Polygon
import descartes
from wordcloud import WordCloud, STOPWORDS

#Import Nominatim for transform city names in coords
from geopy.geocoders import Nominatim
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/surendra/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[4]:
```
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType
import re
```

```
import string

# Register your cleaning functions as UDFs (User Defined Functions)
def clean_text(text):
    '''Make text lowercase, remove text in square brackets, remove links,
    remove punctuation and remove words containing numbers.'''
    if text is None:
        return None
    text = text.lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text

clean_text_udf = udf(clean_text, StringType())

def deEmojify(text):
    if text is None:
        return None
    regrex_pattern = re.compile(pattern = "["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           "]+", flags = re.UNICODE)
    return regrex_pattern.sub(r'', text)

deEmojify_udf = udf(deEmojify, StringType())

# Apply the cleaning functions using withColumn
df = df.withColumn('text', clean_text_udf(df['text']))
df = df.withColumn('text', deEmojify_udf(df['text']))

# For the location column (commented out in your original code)
# df = df.withColumn('location', deEmojify_udf(df['location']))

# Show the results
df.show()
```

```
[Stage 3:>                                                          (0 + 1) / 1]

+---+-------+--------+--------------------+------+
| id|keyword|location|                text|target|
+---+-------+--------+--------------------+------+
|  1|   NULL|    NULL|our deeds are the…|     1|
```

```
|  4|   NULL|    NULL|forest fire near …|     1|
|  5|   NULL|    NULL|all residents ask…|     1|
|  6|   NULL|    NULL| people receive w…|     1|
|  7|   NULL|    NULL|just got sent thi…|     1|
|  8|   NULL|    NULL|rockyfire update …|     1|
| 10|   NULL|    NULL|flood disaster he…|     1|
| 13|   NULL|    NULL|im on top of the …|     1|
| 14|   NULL|    NULL|theres an emergen…|     1|
| 15|   NULL|    NULL|im afraid that th…|     1|
| 16|   NULL|    NULL|three people died…|     1|
| 17|   NULL|    NULL|haha south tampa …|     1|
| 18|   NULL|    NULL|raining flooding …|     1|
| 19|   NULL|    NULL|flood in bago mya…|     1|
| 20|   NULL|    NULL|damage to school …|     1|
| 23|   NULL|    NULL|        whats up man|     0|
| 24|   NULL|    NULL|       i love fruits|     0|
| 25|   NULL|    NULL|    summer is lovely|     0|
| 26|   NULL|    NULL|   my car is so fast|     0|
| 28|   NULL|    NULL|what a goooooooaa…|     0|
+---+-------+--------+--------------------+------+
only showing top 20 rows
```

[5]:
```python
from pyspark.sql.functions import count, countDistinct, desc

# Calculate the metrics
keyword_stats = df.agg(
    count('keyword').alias('count'),
    countDistinct('keyword').alias('unique')
).collect()[0]

# Find the most frequent keyword and its count
top_keyword_df = df.groupBy('keyword').agg(count('*').alias('freq')) \
                   .orderBy(desc('freq')).limit(1).collect()[0]

# Print the results in the format you want
print(f"count            {keyword_stats['count']}")
print(f"unique           {keyword_stats['unique']}")
print(f"top      {top_keyword_df['keyword']}")
print(f"freq            {top_keyword_df['freq']}")
print("Name: keyword, dtype: object")
```

```
count            7987
unique           223
top      None
freq            400
```

```
Name: keyword, dtype: object
```

[6]:
```
# Get unique non-null keywords and sort
unique_keywords = df.select('keyword').na.drop().distinct().rdd.flatMap(lambda
 ↪x: x).collect()
unique_keywords_sorted = sorted(unique_keywords)
unique_keywords_sorted
```

[6]:
```
['0',
 '1',
 'ablaze',
 'accident',
 'aftershock',
 'airplane%20accident',
 'ambulance',
 'annihilated',
 'annihilation',
 'apocalypse',
 'armageddon',
 'army',
 'arson',
 'arsonist',
 'attack',
 'attacked',
 'avalanche',
 'battle',
 'bioterror',
 'bioterrorism',
 'blaze',
 'blazing',
 'bleeding',
 'blew%20up',
 'blight',
 'blizzard',
 'blood',
 'bloody',
 'blown%20up',
 'body%20bag',
 'body%20bagging',
 'body%20bags',
 'bomb',
 'bombed',
 'bombing',
 'bridge%20collapse',
 'buildings%20burning',
```

```
'buildings%20on%20fire',
'burned',
'burning',
'burning%20buildings',
'bush%20fires',
'casualties',
'casualty',
'catastrophe',
'catastrophic',
'chemical%20emergency',
'cliff%20fall',
'collapse',
'collapsed',
'collide',
'collided',
'collision',
'crash',
'crashed',
'crush',
'crushed',
'curfew',
'cyclone',
'damage',
'danger',
'dead',
'death',
'deaths',
'debris',
'deluge',
'deluged',
'demolish',
'demolished',
'demolition',
'derail',
'derailed',
'derailment',
'desolate',
'desolation',
'destroy',
'destroyed',
'destruction',
'detonate',
'detonation',
'devastated',
'devastation',
'disaster',
'displaced',
```

```
'drought',
'drown',
'drowned',
'drowning',
'dust%20storm',
'earthquake',
'electrocute',
'electrocuted',
'emergency',
'emergency%20plan',
'emergency%20services',
'engulfed',
'epicentre',
'evacuate',
'evacuated',
'evacuation',
'explode',
'exploded',
'explosion',
'eyewitness',
'famine',
'fatal',
'fatalities',
'fatality',
'fear',
'fire',
'fire%20truck',
'first%20responders',
'flames',
'flattened',
'flood',
'flooding',
'floods',
'forest%20fire',
'forest%20fires',
'hail',
'hailstorm',
'harm',
'hazard',
'hazardous',
'heat%20wave',
'hellfire',
'hijack',
'hijacker',
'hijacking',
'hostage',
'hostages',
```

```
'hurricane',
'injured',
'injuries',
'injury',
'inundated',
'inundation',
'landslide',
'lava',
'lightning',
'loud%20bang',
'mass%20murder',
'mass%20murderer',
'massacre',
'mayhem',
'meltdown',
'military',
'mudslide',
'natural%20disaster',
'nuclear%20disaster',
'nuclear%20reactor',
'obliterate',
'obliterated',
'obliteration',
'oil%20spill',
'outbreak',
'pandemonium',
'panic',
'panicking',
'police',
'quarantine',
'quarantined',
'radiation%20emergency',
'rainstorm',
'razed',
'refugees',
'rescue',
'rescued',
'rescuers',
'riot',
'rioting',
'rubble',
'ruin',
'sandstorm',
'screamed',
'screaming',
'screams',
'seismic',
```

```
'sinkhole',
'sinking',
'siren',
'sirens',
'smoke',
'snowstorm',
'storm',
'stretcher',
'structural%20failure',
'suicide%20bomb',
'suicide%20bomber',
'suicide%20bombing',
'sunk',
'survive',
'survived',
'survivors',
'terrorism',
'terrorist',
'threat',
'thunder',
'thunderstorm',
'tornado',
'tragedy',
'trapped',
'trauma',
'traumatised',
'trouble',
'tsunami',
'twister',
'typhoon',
'upheaval',
'violent%20storm',
'volcano',
'war%20zone',
'weapon',
'weapons',
'whirlwind',
'wild%20fires',
'wildfire',
'windstorm',
'wounded',
'wounds',
'wreck',
'wreckage',
'wrecked']
```

[6]:
```
from pyspark.sql.functions import col, regexp_replace
from pyspark.sql.types import StringType

# 1. Replace "%20" with spaces in the keyword column
df = df.withColumn('keyword', regexp_replace(col('keyword'), '%20', ' '))

# 2. Cast to string type (though PySpark columns are already StringType by␣
 ↪default for text)
df = df.withColumn('keyword', col('keyword').cast(StringType()))

# 3. Get all unique values (with null handling)
unique_keywords = (df.select('keyword')
                   .na.drop()  # Remove nulls
                   .distinct()
                   .rdd.flatMap(lambda x: x)
                   .collect())
unique_keywords_sorted = sorted([k for k in unique_keywords if k is not None])

# Show results
unique_keywords_sorted
```

[6]:
```
['0',
 '1',
 'ablaze',
 'accident',
 'aftershock',
 'airplane accident',
 'ambulance',
 'annihilated',
 'annihilation',
 'apocalypse',
 'armageddon',
 'army',
 'arson',
 'arsonist',
 'attack',
 'attacked',
 'avalanche',
 'battle',
 'bioterror',
 'bioterrorism',
 'blaze',
 'blazing',
 'bleeding',
 'blew up',
```

```
'blight',
'blizzard',
'blood',
'bloody',
'blown up',
'body bag',
'body bagging',
'body bags',
'bomb',
'bombed',
'bombing',
'bridge collapse',
'buildings burning',
'buildings on fire',
'burned',
'burning',
'burning buildings',
'bush fires',
'casualties',
'casualty',
'catastrophe',
'catastrophic',
'chemical emergency',
'cliff fall',
'collapse',
'collapsed',
'collide',
'collided',
'collision',
'crash',
'crashed',
'crush',
'crushed',
'curfew',
'cyclone',
'damage',
'danger',
'dead',
'death',
'deaths',
'debris',
'deluge',
'deluged',
'demolish',
'demolished',
'demolition',
'derail',
```

```
'derailed',
'derailment',
'desolate',
'desolation',
'destroy',
'destroyed',
'destruction',
'detonate',
'detonation',
'devastated',
'devastation',
'disaster',
'displaced',
'drought',
'drown',
'drowned',
'drowning',
'dust storm',
'earthquake',
'electrocute',
'electrocuted',
'emergency',
'emergency plan',
'emergency services',
'engulfed',
'epicentre',
'evacuate',
'evacuated',
'evacuation',
'explode',
'exploded',
'explosion',
'eyewitness',
'famine',
'fatal',
'fatalities',
'fatality',
'fear',
'fire',
'fire truck',
'first responders',
'flames',
'flattened',
'flood',
'flooding',
'floods',
'forest fire',
```

```
'forest fires',
'hail',
'hailstorm',
'harm',
'hazard',
'hazardous',
'heat wave',
'hellfire',
'hijack',
'hijacker',
'hijacking',
'hostage',
'hostages',
'hurricane',
'injured',
'injuries',
'injury',
'inundated',
'inundation',
'landslide',
'lava',
'lightning',
'loud bang',
'mass murder',
'mass murderer',
'massacre',
'mayhem',
'meltdown',
'military',
'mudslide',
'natural disaster',
'nuclear disaster',
'nuclear reactor',
'obliterate',
'obliterated',
'obliteration',
'oil spill',
'outbreak',
'pandemonium',
'panic',
'panicking',
'police',
'quarantine',
'quarantined',
'radiation emergency',
'rainstorm',
'razed',
```

```
'refugees',
'rescue',
'rescued',
'rescuers',
'riot',
'rioting',
'rubble',
'ruin',
'sandstorm',
'screamed',
'screaming',
'screams',
'seismic',
'sinkhole',
'sinking',
'siren',
'sirens',
'smoke',
'snowstorm',
'storm',
'stretcher',
'structural failure',
'suicide bomb',
'suicide bomber',
'suicide bombing',
'sunk',
'survive',
'survived',
'survivors',
'terrorism',
'terrorist',
'threat',
'thunder',
'thunderstorm',
'tornado',
'tragedy',
'trapped',
'trauma',
'traumatised',
'trouble',
'tsunami',
'twister',
'typhoon',
'upheaval',
'violent storm',
'volcano',
'war zone',
```

```
 'weapon',
 'weapons',
 'whirlwind',
 'wild fires',
 'wildfire',
 'windstorm',
 'wounded',
 'wounds',
 'wreck',
 'wreckage',
 'wrecked']
```

[7]:
```
print("Dinesh")
```

```
Dinesh
```

[7]:
```
from pyspark.sql.functions import col, lit, concat, when

# Add quotes around location values with proper null handling
df = df.withColumn("location",
                   when(col("location").isNull(), lit("''"))
                   .otherwise(concat(lit("'"), col("location"), lit("'"))))

# Show the result
df.show()
```

```
+---+-------+--------+--------------------+------+
| id|keyword|location|                text|target|
+---+-------+--------+--------------------+------+
|  1|   NULL|      ''|our deeds are the…|     1|
|  4|   NULL|      ''|forest fire near …|     1|
|  5|   NULL|      ''|all residents ask…|     1|
|  6|   NULL|      ''| people receive w…|     1|
|  7|   NULL|      ''|just got sent thi…|     1|
|  8|   NULL|      ''|rockyfire update …|     1|
| 10|   NULL|      ''|flood disaster he…|     1|
| 13|   NULL|      ''|im on top of the …|     1|
| 14|   NULL|      ''|theres an emergen…|     1|
| 15|   NULL|      ''|im afraid that th…|     1|
| 16|   NULL|      ''|three people died…|     1|
| 17|   NULL|      ''|haha south tampa …|     1|
| 18|   NULL|      ''|raining flooding …|     1|
| 19|   NULL|      ''|flood in bago mya…|     1|
| 20|   NULL|      ''|damage to school …|     1|
| 23|   NULL|      ''|        whats up man|     0|
| 24|   NULL|      ''|       i love fruits|     0|
| 25|   NULL|      ''|    summer is lovely|     0|
| 26|   NULL|      ''|   my car is so fast|     0|
```

```
| 28|   NULL|      ''|what a gooooooooaa…|     0|
+---+-------+--------+--------------------+------+
only showing top 20 rows
```

[8]:
```python
from pyspark.sql.functions import udf, col
from pyspark.sql.types import StringType
import emoji

# Define the UDF version of your function
def emojiflag_to_text(flag):
    '''Convert the emoji flag to a string with their name'''
    if flag is not None and len(flag) == 4:  # Added null check
        try:
            return emoji.demojize(flag)
        except:
            return flag
    return flag

# Register the UDF
emoji_udf = udf(emojiflag_to_text, StringType())

# Apply the transformation
df = df.withColumn('location', emoji_udf(col('location')))

# Show results (first 20 rows)
df.select('location').show(20, truncate=False)
```

```
+--------+
|location|
+--------+
|''      |
|''      |
|''      |
|''      |
|''      |
|''      |
|''      |
|''      |
|''      |
|''      |
|''      |
|''      |
|''      |
|''      |
|''      |
|''      |
|''      |
|''      |
|''      |
|''      |
```

```
|''      |
|''      |
|''      |
+--------+
only showing top 20 rows
```

[8]:
```python
from pyspark.sql.functions import col, regexp_replace, when, udf
from pyspark.sql.types import StringType
import re

# 1. Remove single quotes
df = df.withColumn('location', regexp_replace(col('location'), "'", ""))

# 2. Remove colons
df = df.withColumn('location', regexp_replace(col('location'), ":", ""))

# 3. Replace "nan" strings with actual nulls
df = df.withColumn('location',
                   when(col('location') == 'nan', None)
                   .otherwise(col('location')))

# 4. Remove remaining emojis using UDF
def remove_special_chars(text):
    if text is None:
        return None
    return re.sub(r'[^\w\s#@/:%.,_-]', '', text)

remove_chars_udf = udf(remove_special_chars, StringType())
df = df.withColumn('location', remove_chars_udf(col('location')))

# Show cleaned results (first 20 rows without truncation)
df.select('location').show(20, truncate=False)
```

```
+--------+
|location|
+--------+
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
```

```
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
+--------+
only showing top 20 rows
```

[43]:

Dinesh Bekkam

[9]:
```
from pyspark.sql.functions import col, concat, lit

# 1. Cast to string and add quotes (with null handling)
df = df.withColumn("location",
                   concat(lit("'"), col("location").cast("string"), lit("'")))

# 2. Show the updated DataFrame
df.show()
```

```
+---+-------+--------+--------------------+------+
| id|keyword|location|                text|target|
+---+-------+--------+--------------------+------+
|  1|   NULL|      ''|our deeds are the…|     1|
|  4|   NULL|      ''|forest fire near …|     1|
|  5|   NULL|      ''|all residents ask…|     1|
|  6|   NULL|      ''| people receive w…|     1|
|  7|   NULL|      ''|just got sent thi…|     1|
|  8|   NULL|      ''|rockyfire update …|     1|
| 10|   NULL|      ''|flood disaster he…|     1|
| 13|   NULL|      ''|im on top of the …|     1|
| 14|   NULL|      ''|theres an emergen…|     1|
| 15|   NULL|      ''|im afraid that th…|     1|
| 16|   NULL|      ''|three people died…|     1|
| 17|   NULL|      ''|haha south tampa …|     1|
| 18|   NULL|      ''|raining flooding …|     1|
| 19|   NULL|      ''|flood in bago mya…|     1|
| 20|   NULL|      ''|damage to school …|     1|
| 23|   NULL|      ''|        whats up man|     0|
| 24|   NULL|      ''|       i love fruits|     0|
| 25|   NULL|      ''|    summer is lovely|     0|
| 26|   NULL|      ''|   my car is so fast|     0|
```

```
| 28|   NULL|      ''|what a gooooooaa…|     0|
+---+-------+--------+--------------------+------+
only showing top 20 rows
```

[10]:
```python
from pyspark.sql.functions import udf, col
from pyspark.sql.types import StringType
import emoji

# Define the UDF version of your function
def emojiflag_to_text(flag):
    '''Convert the emoji flag to a string with their name'''
    if flag and len(flag) == 4:  # Added null check
        return emoji.demojize(flag)
    return flag

# Register the UDF
emoji_udf = udf(emojiflag_to_text, StringType())

# Apply the transformation
df = df.withColumn('location', emoji_udf(col('location')))

# Show results
df.show()
```

```
+---+-------+--------+--------------------+------+
| id|keyword|location|                text|target|
+---+-------+--------+--------------------+------+
|  1|   NULL|    ''''|our deeds are the…|     1|
|  4|   NULL|    ''''|forest fire near …|     1|
|  5|   NULL|    ''''|all residents ask…|     1|
|  6|   NULL|    ''''| people receive w…|     1|
|  7|   NULL|    ''''|just got sent thi…|     1|
|  8|   NULL|    ''''|rockyfire update …|     1|
| 10|   NULL|    ''''|flood disaster he…|     1|
| 13|   NULL|    ''''|im on top of the …|     1|
| 14|   NULL|    ''''|theres an emergen…|     1|
| 15|   NULL|    ''''|im afraid that th…|     1|
| 16|   NULL|    ''''|three people died…|     1|
| 17|   NULL|    ''''|haha south tampa …|     1|
| 18|   NULL|    ''''|raining flooding …|     1|
| 19|   NULL|    ''''|flood in bago mya…|     1|
| 20|   NULL|    ''''|damage to school …|     1|
| 23|   NULL|    ''''|        whats up man|     0|
| 24|   NULL|    ''''|       i love fruits|     0|
| 25|   NULL|    ''''|    summer is lovely|     0|
| 26|   NULL|    ''''|   my car is so fast|     0|
| 28|   NULL|    ''''|what a gooooooaa…|     0|
```

```
+---+-------+--------+--------------------+------+
only showing top 20 rows
```

[11]:
```python
from pyspark.sql.functions import col, regexp_replace, when, lit, udf
from pyspark.sql.types import StringType
import re

# Step 1: Remove single quotes
df = df.withColumn('location', regexp_replace(col('location'), "'", ""))

# Step 2: Remove colons
df = df.withColumn('location', regexp_replace(col('location'), ":", ""))

# Step 3: Handle 'nan' strings (now with proper imports)
df = df.withColumn('location',
                   when(col('location') == 'nan', lit(None))
                   .otherwise(col('location')))

# Step 4: Define and register UDF for emoji removal
def remove_emojis(text):
    if text is None:
        return None
    return re.sub(r'[^\w\s#@/:%.,_-]', '', text, flags=re.UNICODE)

remove_emojis_udf = udf(remove_emojis, StringType())

# Step 5: Remove remaining emojis
df = df.withColumn('location', remove_emojis_udf(col('location')))

# Show results
df.select('location').show(50,truncate=False)
```

```
+----------------------------+
|location                    |
+----------------------------+
|                            |
|                            |
|                            |
|                            |
|                            |
|                            |
|                            |
|                            |
|                            |
|                            |
|                            |
|                            |
```

```
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|Birmingham                   |
|Est. September 2012 - Bristol|
|AFRICA                       |
|Philadelphia, PA             |
|London, UK                   |
|Pretoria                     |
|World Wide                   |
|                             |
|Paranaque City               |
|Live On Webcam               |
|                             |
|                             |
|milky way                    |
|                             |
|                             |
|                             |
|GREENSBORO,NORTH CAROLINA    |
|                             |
|Live On Webcam               |
+-----------------------------+
only showing top 50 rows
```

[12]:
```
print("DataFrame Schema:")
df.printSchema()

# Show dimensions
print("\nDataFrame Dimensions:")
```

```
print(f"Rows: {df.count()}")
print(f"Columns: {len(df.columns)}")

# Show data types
print("\nData Types")
print("__"*12)
for field in df.schema.fields:
    print(f"{field.name.ljust(20)}: {str(field.dataType)}")
```

```
DataFrame Schema:
root
 |-- id: string (nullable = true)
 |-- keyword: string (nullable = true)
 |-- location: string (nullable = true)
 |-- text: string (nullable = true)
 |-- target: integer (nullable = true)


DataFrame Dimensions:
Rows: 8387
Columns: 5

Data Types
------------------------
id                   : StringType()
keyword              : StringType()
location             : StringType()
text                 : StringType()
target               : IntegerType()
```

[13]:
```
from pyspark.sql.functions import col, sum as spark_sum

# Count null values in 'location' column
null_locations = df.filter(col("location").isNull()).count()

# Alternative method using aggregation
null_locations_alt = df.agg(spark_sum(col("location").isNull().
 ↪cast("integer"))).collect()[0][0]

print(f'We have {null_locations} missing locations')
```

```
We have 0 missing locations
```

[14]:
```
from pyspark.sql.types import _parse_datatype_string

# 1. Show current data types
```

```
print("Current Data Types:")
print("__"*12)
for field in df.schema.fields:
    print(f"{field.name:<20} {str(field.dataType)}")

# 2. Automatic type inference (similar to convert_dtypes)
print("\nInferred Data Types:")
print("__"*12)
for col_name in df.columns:
    # Get sample of data to infer type
    sample = df.select(col_name).filter(col(col_name).isNotNull()).take(1)
    if sample:
        inferred_type = type(sample[0][0]).__name__
        print(f"{col_name:<20} {inferred_type}")
    else:
        print(f"{col_name:<20} Could not infer (all nulls)")

# 3. Convert types (example for string to timestamp)
# df = df.withColumn("date_column", col("date_column").cast("timestamp"))
```

```
Current Data Types:

------------------------
id                   StringType()
keyword              StringType()
location             StringType()
text                 StringType()
target               IntegerType()

Inferred Data Types:

------------------------
id                   str
keyword              str
location             str
text                 str
target               int
```

[15]:
```
from pyspark.sql.functions import countDistinct

# Count distinct values for each column
unique_counts = df.agg(*(countDistinct(col(c)).alias(c) for c in df.columns))

# Show the results
unique_counts.show(vertical=True)
```

```
[Stage 35:============================================================(1 + 0) / 1]

-RECORD 0--------
 id       | 8302
```

```
keyword  | 223
location | 3318
text     | 6893
target   | 2
```

[16]:
```python
from pyspark.sql.functions import col

# Count duplicated text values (exact duplicates)
dup_text_count = df.groupBy("text").count().filter("count > 1").count()

print(f'We have {dup_text_count} duplicated texts')
```

```
[Stage 41:>                                                         (0 + 1) / 1]
```

```
We have 318 duplicated texts
```

[17]:
```python
from pyspark.sql.functions import length, col, min, max, mean, stddev

# 1. Add text length column
df = df.withColumn("length", length(col("text")))

# 2. Get basic statistics
length_stats = df.select(
    mean("length").alias("mean_length"),
    stddev("length").alias("stddev_length"),
    min("length").alias("min_length"),
    max("length").alias("max_length")
).collect()[0]

print("Text Length Statistics:")
print(f"Average length: {length_stats['mean_length']:.1f} characters")
print(f"Standard deviation: {length_stats['stddev_length']:.1f}")
print(f"Shortest text: {length_stats['min_length']} characters")
print(f"Longest text: {length_stats['max_length']} characters")

# 3. Show relation with target (if you have a target column)
if "target" in df.columns:
    print("\nLength by Target:")
    df.groupBy("target").agg(
        mean("length").alias("avg_length"),
        stddev("length").alias("stddev_length")
    ).show()
```

```
Text Length Statistics:
```

```
Average length: 78.9 characters
Standard deviation: 32.2
Shortest text: 0 characters
Longest text: 145 characters

Length by Target:
+------+-----------------+------------------+
|target|       avg_length|     stddev_length|
+------+-----------------+------------------+
|  NULL|47.96551724137931| 30.25043255807656|
|     1|84.47322297955209|28.412488342229736|
|     0|78.08083028083028| 33.11491385439553|
+------+-----------------+------------------+
```

[32]:
```
display(df.select("length"))
```

```
DataFrame[length: int]
```

[18]:
```
from pyspark.sql.functions import col

# Display tweets longer than 130 characters
print("Tweets with length > 130 characters:")
df.filter(col("length") > 130).show(truncate=False)

print("\n" + "="*80 + "\n")

# Display tweets shorter than 20 characters
print("Tweets with length < 20 characters:")
df.filter(col("length") < 20).show(truncate=False)
```

```
Tweets with length > 130 characters:
+---+-------------------+-------------------+----------------------------------
----------------------------------------------------------------------------
----------------------+------+------+
|id |keyword            |location           |text
|target|length|
+---+-------------------+-------------------+----------------------------------
----------------------------------------------------------------------------
----------------------+------+------+
|71 |ablaze             |England.           |first night with retainers in its
quite weird better get used to it i have to wear them every single night for the
next year at least     |0     |133   |
|96 |accident           |CLVLND             |i cant have kids cuz i got in a
bicycle accident amp split my testicles its impossible for me to have kids
michael you are the father     |0     |133   |
|129|accident           |Maldives           |rt naayf first accident in years
turning onto chandanee magu from near mma taxi rammed into me while i was
```

```
halfway turned everyone conf û |1     |137   |
|138|accident           |Baton Rouge, LA    |has an accident changed your life
we will help you determine options that can financially support life care plans
and ongoing treatment   |0     |135   |
|145|accident           |Nairobi, Kenya     |i still have not heard church
leaders of kenya coming forward to comment on the accident issue and
disciplinary measuresarrestpastornganga|0     |138   |
|184|aftershock         |304                |remembering that you are going to
die is the best way i know to avoid the trap of thinking you have something to
lose  ûò steve jobs      |0     |132   |
|211|airplane%20accident|                   |experts in france begin examining
airplane debris found on reunion island french air accident experts on wednesday
began examining t      |1     |132   |
|238|airplane%20accident|                   |experts in france begin examining
airplane debris found on reunion island french air accident experts on wednesday
began examining t      |1     |132   |
|252|ambulance          |West Wales         |anyone travelling
aberystwythshrewsbury right now theres been an incident services at a halt just
outside shrews ambulance on scene       |1     |131   |
|334|annihilated        |                   |tomcatarts thus explaining why you
were all annihilated but the few or in this case you the only survivor evolved
and became godlike      |1     |132   |
|336|annihilated        |                   |tomcatarts who then were
annihilated by the legion itself the survivors of the imperfect hybrid project
quickly formed a new secret cell  |0     |136   |
|341|annihilated        |                   |thatdes ok i wasnt completely
forthright i may have also been in a food coma bc of the kebabtahinipickles i
also annihilated wfries       |0     |131   |
|383|annihilation       |Connecticut        |sonofbaldwin and hes the current
nova in the bookslast i checkedhe was tied into the books in  after rider died
during annihilation       |0     |131   |
|424|apocalypse         |San Antonio-ish, TX|dad bought a dvd that looks like a
science doc on the front but i read the back and its actually about the
impending biblical apocalypse  |1     |136   |
|444|apocalypse         |Tokyo              |enjoyed liveaction attack on titan
but every time i see posters im reminded how freshly clean and coiffed everyone
is in the apocalypse   |0     |135   |
|686|attack             |#UNITE THE BLUE    |blazerfan not everyone can see
ignoranceshe is latinoand that is all she can ever benothing morebut an attack
dog  a hate group gop       |0     |131   |
|706|attacked           |atx                |i cant believe a fucking cis female
is going to somehow claim to be offended over a transgendered female whos been
attacked by media      |0     |132   |
|818|battle             |Wisconsin          |sexydragonmagic ive come to the
realization that i just dont have the attention span for mass battle games both
painting and playing      |0     |132   |
|851|bioterror          |United States      |rt alisonannyoung exclusive fedex
no longer to transport research specimens of bioterror pathogens in wake of
```

```
anthrax lab mishaps  û      |0     |132   |
|896|bioterrorism       |Philadelphia, PA   |meyerbjoern thelonevirologi
mackayim of a major american newspaper running a series on all the alleged
bioterrorism research going on     |1     |134   |
+---+-------------------+-------------------+--------------------------------
--------------------------------------------------------------------------------
----------------------+------+------+
only showing top 20 rows


================================================================================

Tweets with length < 20 characters:
+---+-----------+--------------+------------------+------+------+
|id |keyword    |location      |text              |target|length|
+---+-----------+--------------+------------------+------+------+
|23 |NULL       |              |whats up man      |0     |12    |
|24 |NULL       |              |i love fruits     |0     |13    |
|25 |NULL       |              |summer is lovely  |0     |16    |
|26 |NULL       |              |my car is so fast |0     |17    |
|31 |NULL       |              |this is ridiculous|0     |18    |
|32 |NULL       |              |london is cool    |0     |15    |
|33 |NULL       |              |love skiing       |0     |11    |
|36 |NULL       |              |looooool          |0     |8     |
|39 |NULL       |              |love my girlfriend|0     |18    |
|40 |NULL       |              |cooool            |0     |7     |
|41 |NULL       |              |do you like pasta |0     |17    |
|44 |NULL       |              |the end           |0     |7     |
|163|aftershock |Belgium       |aftershock        |0     |11    |
|190|aftershock |              |aftershock        |0     |11    |
|302|annihilated|              |annihilated abs   |1     |18    |
|402|apocalypse |Texas         |apocalypse please |0     |17    |
|443|apocalypse |              |short reading     |NULL  |13    |
|467|armageddon |Here And There|armageddon        |1     |11    |
|469|armageddon |Derry, 17     | armageddon       |0     |11    |
|524|army       |Campinas Sp   |you da one        |NULL  |11    |
+---+-----------+--------------+------------------+------+------+
only showing top 20 rows
```

[19]:
```
from pyspark.sql.functions import desc

# Get value counts and filter for locations appearing 10 times
location_counts = (df.groupBy("location")
                   .count()
                   .filter("count >= 10")
                   .orderBy(desc("count")))
```

```
# Show top 20 locations
location_counts.show(20, truncate=False)
```

```
[Stage 55:>                                                         (0 + 1) / 1]
```

```
+---------------+-----+
|location       |count|
+---------------+-----+
|               |3334 |
|USA            |104  |
|New York       |71   |
|United States  |50   |
|London         |45   |
|Canada         |29   |
|Nigeria        |28   |
|UK             |27   |
|Los Angeles, CA|26   |
|India          |24   |
|Mumbai         |22   |
|Washington, DC |21   |
|Kenya          |20   |
|Worldwide      |19   |
|Chicago, IL    |18   |
|Australia      |18   |
|California     |17   |
|Everywhere     |15   |
|California, USA|15   |
|New York, NY   |15   |
+---------------+-----+
only showing top 20 rows
```

[20]:
```
# We are going to group some places, so we create a mapping dictionary to␣
 ↪replace the locations
mapping = {'United States':'USA',
           'New York':'USA',
           "London":'UK',
           "Los Angeles, CA":'USA',
           "Washington, D.C.":'USA',
           "California":'USA',
           "Chicago, IL":'USA',
           "Chicago":'USA',
           "New York, NY":'USA',
           "California, USA":'USA',
           "FLorida":'USA',
```

```
            "Nigeria":'Africa',
            "Kenya":'Africa',
            "Everywhere":'Worldwide',
            "San Francisco":'USA',
            "Florida":'USA',
            "United Kingdom":'UK',
            "Los Angeles":'USA',
            "Toronto":'Canada',
            "San Francisco, CA":'USA',
            "NYC":'USA',
            "Seattle":'USA',
            "Earth":'Worldwide',
            "Ireland":'UK',
            "London, England":'UK',
            "New York City":'USA',
            "Texas":'USA',
            "London, UK":'UK',
            "Atlanta, GA":'USA',
            "England, United Kingdom":'UK',
            "Mumbai, India":'India',
            "Melbourne,Victoria":'Australia'}
```

[21]:
```
from pyspark.sql.functions import col, udf
from pyspark.sql.types import StringType

# Assuming 'mapping' is your Python dictionary
# e.g., mapping = {'New York': 'NY', 'California': 'CA'}

# 1. Create a UDF (User Defined Function)
def location_mapper(loc):
    return mapping.get(loc, loc)  # Returns mapped value or original if not␣
 ↪found

location_mapper_udf = udf(location_mapper, StringType())

# 2. Apply the mapping
df = df.withColumn('location', location_mapper_udf(col('location')))

# 3. Show results
df.select('location').show(50, truncate=False)
```

```
+-----------------------------+
|location                     |
+-----------------------------+
|                             |
|                             |
|                             |
```

```
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|                              |
|Birmingham                    |
|Est. September 2012 - Bristol|
|AFRICA                        |
|Philadelphia, PA              |
|UK                            |
|Pretoria                      |
|World Wide                    |
|                              |
|Paranaque City                |
|Live On Webcam                |
|                              |
|                              |
|milky way                     |
|                              |
|                              |
|                              |
|GREENSBORO,NORTH CAROLINA     |
|                              |
|Live On Webcam                |
+------------------------------+
```

only showing top 50 rows

[22]:
```
from pyspark.sql.functions import col

# 1. Create new DataFrame (note: in Spark this is just a reference)
ndf = df

# 2. Drop rows with null values in any column
ndf = ndf.na.drop()

# Alternative: Only drop rows where 'location' is null
# ndf = ndf.dropna(subset=['location'])

# 3. Get value counts for location (appearing 10 times)
location_counts = (ndf.groupBy('location')
                   .count()
                   .filter(col('count') >= 10)
                   .orderBy('count', ascending=False))

# Show results
location_counts.show(truncate=False)

# To collect as a dictionary (like Pandas):
location_dict = {row['location']: row['count'] for row in location_counts.
 ↪collect()}
print(location_dict)
```

```
+--------------+-----+
|location      |count|
+--------------+-----+
|              |2368 |
|USA           |427  |
|UK            |111  |
|Africa        |51   |
|Worldwide     |41   |
|Canada        |39   |
|India         |24   |
|Mumbai        |21   |
|Washington, DC|21   |
|Australia     |17   |
|Indonesia     |13   |
|ss            |10   |
+--------------+-----+
```

```
{'': 2368, 'USA': 427, 'UK': 111, 'Africa': 51, 'Worldwide': 41, 'Canada': 39,
'India': 24, 'Mumbai': 21, 'Washington, DC': 21, 'Australia': 17, 'Indonesia':
13, 'ss': 10}
```

[ ]:

[22]:
```
from pyspark.sql.functions import desc

# Get value counts for 'location' column
countries_mask = (df.groupBy("location")
                  .count()
                  .orderBy(desc("count")))

# Show the results
countries_mask.show()
```

```
+----------------+-----+
|        location|count|
+----------------+-----+
|                | 3334|
|             USA|  445|
|              UK|  120|
|          Africa|   51|
|       Worldwide|   45|
|          Canada|   41|
|           India|   25|
|          Mumbai|   22|
|  Washington, DC|   21|
|       Australia|   18|
|       Indonesia|   14|
|  Sacramento, CA|   10|
|              ss|   10|
|      Manchester|    9|
|           World|    9|
|   Nashville, TN|    9|
|      Dallas, TX|    9|
|              US|    9|
|   San Diego, CA|    9|
|Denver, Colorado|    9|
+----------------+-----+
only showing top 20 rows
```

[23]:
```
from pyspark.sql.functions import desc

# Get value counts for 'location' column
countries_mask = (df.groupBy("location")
```

```
                    .count()
                    .orderBy(desc("count")))

# Show the results
countries_mask.show()
```

```
+---------------+-----+
|       location|count|
+---------------+-----+
|                | 3334|
|            USA|  445|
|             UK|  120|
|         Africa|   51|
|      Worldwide|   45|
|         Canada|   41|
|          India|   25|
|         Mumbai|   22|
| Washington, DC|   21|
|      Australia|   18|
|      Indonesia|   14|
| Sacramento, CA|   10|
|             ss|   10|
|     Manchester|    9|
|          World|    9|
|  Nashville, TN|    9|
|     Dallas, TX|    9|
|             US|    9|
|  San Diego, CA|    9|
|Denver, Colorado|    9|
+---------------+-----+
only showing top 20 rows
```

[43]:
```
from pyspark.sql.functions import col

# Assuming you have a PySpark DataFrame 'df' with a 'location' column
# and you've already done value counts like this:
location_counts = df.groupBy("location").count().orderBy("count",␣
 ↪ascending=False)

# Get just the location names (equivalent to list(location.index) in Pandas)
location_names = [row["location"] for row in location_counts.select("location").
 ↪collect()]
location_names
```

```
[43]: ["''",
       "'USA'",
       "'New York'",
       "'United States'",
       "'London'",
       "'Canada'",
       "'Nigeria'",
       "'UK'",
       "'Los Angeles, CA'",
       "'India'",
       "'Mumbai'",
       "'Washington, DC'",
       "'Kenya'",
       "'Worldwide'",
       "'Australia'",
       "'Chicago, IL'",
       "'California'",
       "'New York, NY'",
       "'Everywhere'",
       "'California, USA'",
       "'United Kingdom'",
       "'Florida'",
       "'Indonesia'",
       "'San Francisco'",
       "'Los Angeles'",
       "'Washington, D.C.'",
       "'Ireland'",
       "'NYC'",
       "'Toronto'",
       "'San Francisco, CA'",
       "'Earth'",
       "'Seattle'",
       "'Chicago'",
       "'New York City'",
       "'London, England'",
       "'ss'",
       "'Texas'",
       "'Sacramento, CA'",
       "'London, UK'",
       "'Atlanta, GA'",
       "'San Diego, CA'",
       "'Denver, Colorado'",
       "'Manchester'",
       "'World'",
       "'Nashville, TN'",
       "'US'",
       "'304'",
```

```
"'Dallas, TX'",
"'South Africa'",
"'Scotland'",
"'Houston, TX'",
"'Pennsylvania, USA'",
"'Tennessee'",
"'Sydney'",
"'Memphis, TN'",
"'Seattle, WA'",
"'Denver, CO'",
"'Austin, TX'",
"'worldwide'",
"'Oklahoma City, OK'",
"'Atlanta'",
"'Singapore'",
"'Orlando, FL'",
"'Massachusetts'",
"'Colorado'",
"'Planet Earth'",
"'Brooklyn, NY'",
"'Morioh, Japan'",
"'Charlotte, NC'",
"'Portland, OR'",
"' Road to the Billionaires Club'",
"'Paterson, New Jersey '",
"'California, United States'",
"'Pedophile hunting ground'",
"'Global'",
"'Calgary, Alberta'",
"'Texas, USA'",
"'Southern California'",
"'The Netherlands'",
"'Lagos, Nigeria'",
"'Pennsylvania'",
"'Asheville, NC'",
"'Tampa, FL'",
"'Coventry'",
"'Melbourne, Australia'",
"'New Hampshire'",
"'NY'",
"'Cleveland, OH'",
"'Puerto Rico'",
"'Wisconsin'",
"'North Carolina'",
"' '",
"'Lagos'",
"'Leeds, England'",
```

"'New York, USA'",
"'WorldWide'",
"'Pakistan'",
"'Philippines'",
"' '",
"'New Jersey'",
"'Florida, USA'",
"'Boston, MA'",
"'Vancouver, BC'",
"'San Jose, CA'",
"'Indiana'",
"'Newcastle'",
"'in the Word of God'",
"'Calgary, AB'",
"'Brasil'",
"'Cape Town'",
"'Calgary'",
"'MAD as Hell'",
"'Midwest'",
"'Leicester'",
"'Atlanta Georgia '",
"'Jamaica'",
"'San Francisco Bay Area'",
"'Oakland, CA'",
"'Republic of Texas'",
"'Sydney, Australia'",
"'Tokyo'",
"'Japan'",
"'Paignton'",
"'Melbourne'",
"'Michigan'",
"'Nigeria '",
"'Birmingham'",
"'Happily Married with 2 kids '",
"'Sacramento'",
"'Switzerland'",
"'london'",
"'Geneva'",
"'Bend, Oregon'",
"'canada'",
"'Manchester, England'",
"'North Carolina, USA'",
"'Portland, Oregon'",
"'Maryland'",
"'U.S.A'",
"'Nairobi-KENYA'",
"'World Wide'",

```
"'Oregon'",
"'Port Harcourt, Nigeria'",
"'Sydney, New South Wales'",
"'china'",
"'Haddonfield, NJ'",
"'Seattle, Washington'",
"'Mumbai, Maharashtra'",
"'Barbados'",
"'British Columbia, Canada'",
"'Kansas City'",
"'Jakarta/Kuala Lumpur/Spore'",
"'Phoenix, AZ'",
"'Edinburgh'",
"'Philadelphia, PA'",
"'Massachusetts, USA'",
"'Las Vegas, Nevada'",
"'State College, PA'",
"'St. Louis, MO'",
"'Kama  18  France '",
"'Tulsa, Oklahoma'",
"'Haysville, KS'",
"'Nairobi'",
"'California '",
"'Gotham City'",
"'New Orleans ,Louisiana'",
"'Georgia'",
"'Oklahoma City'",
"'Los Angeles, California'",
"'Africa'",
"'Las Vegas'",
"'Arizona'",
"'Financial News and Views'",
"'Karachi Pakistan'",
"'New Orleans, LA'",
"'America'",
"'iTunes'",
"'NJ'",
"'Memphis'",
"'East Coast'",
"'Ontario, Canada'",
"'NYC  Ex- #Islamophobe'",
"'Buy Give Me My Money '",
"'Johannesburg, South Africa'",
"'CA'",
"'new york'",
"'Anonymous'",
"'England'",
```

"'Oregon, USA'",
"'Arlington, TX'",
"'Winston-Salem, NC'",
"'uk'",
"'America of Founding Fathers'",
"'Columbus, OH'",
"'Virginia'",
"'Stockholm, Sweden'",
"'Asia'",
"'Karachi'",
"'Utah'",
"'Washington D.C.'",
"'Manhattan, NY'",
"'Des Moines, IA'",
"'Narnia'",
"'IraqAfghanistan RSA Baghdad'",
"'Hong Kong'",
"'india'",
"'Milwaukee, WI'",
"'Illinois, USA'",
"'21.462446,-158.022017'",
"'Buffalo, NY'",
"'PA'",
"'MA'",
"'Five down from the Coffeeshop'",
"'Glasgow'",
"'Sweden'",
"'Washington DC'",
"'San Diego'",
"'Nowhere. Everywhere.'",
"'Germany'",
"'#FLIGHTCITY UK '",
"'CA via Brum'",
"'Anchorage, AK'",
"'Dublin, Ireland'",
"'Austin, Texas'",
"'Nottingham, England'",
"'Taylor Swift'",
"'Jerusalem'",
"'Upstairs.'",
"'Brazil'",
"'Baltimore, MD'",
"'Toronto, Ontario'",
"'Inexpressible Island '",
"'South, USA'",
"'Victoria, BC'",
"'Hawaii, USA'",

```
"'Boston'",
"'Victoria, British Columbia'",
"'Indianapolis, IN'",
"'Malaysia'",
"'Virginia, USA'",
"'Calgary, AB, Canada'",
"'Adelaide, Australia'",
"'Madison, WI'",
"'Italy'",
"'Raleigh, NC'",
"'Kansas City, MO'",
"'nyc'",
"'Vancouver, British Columbia'",
"'Rio de Janeiro'",
"'Birmingham, England'",
"'China'",
"'Nairobi, Kenya'",
"'Naperville'",
"'ARGENTINA'",
"'Subconscious LA'",
"'London UK'",
"'Kent'",
"'mumbai'",
"'Europe'",
"'online '",
"'Bukittinggi   Sumatera Barat'",
"'Maldives'",
"'Belgium'",
"'Sarasota, FL'",
"'Portland, Ore. '",
"'Santa Cruz, CA'",
"'Kingston, Jamaica'",
"'scandinavia'",
"'To The Right of You'",
"'Berlin, Germany'",
"'Halifax'",
"'Blackpool'",
"'Breaking News'",
"'Yogya Berhati Nyaman'",
"'New Hanover County, NC'",
"'Ewa Beach, HI'",
"'far away'",
"'Detroit, MI'",
"'Location'",
"'somewhere USA '",
"'Silicon Valley'",
"'SWMO'",
```

```
"'the insane asylum. '",
"'Somewhere in the Canada'",
"'REPUBLICA DOMINICANA'",
"'Natinixw / Hoopa, Berkeley'",
"'Boston MA'",
"'Chevy Chase, MD'",
"'Redding, California, USA'",
"'Bronx NY'",
"'Knoxville, TN'",
"'Mongolia'",
"'Cumming, GA'",
"'tx'",
"'U.K.'",
"'Somewhere Only We Know '",
"'Chicago, Illinois'",
"'Dublin City, Ireland'",
"'Melbourne Australia'",
"'San Antonio, TX'",
"'Heaven'",
"'Toronto, Canada'",
"'Alabama'",
"'West'",
"'Pittsburgh'",
"'El Dorado, Arkansas'",
"'Above the snake line - #YoNews'",
"'Connecticut'",
"'Ottawa, Canada'",
"'Lincoln, NE'",
"'los angeles, ca'",
"'Korea'",
"'Columbus'",
"'A little house in the outback.'",
"'In Hell'",
"'Glasgow, Scotland'",
"'Global-NoLocation'",
"'Lyallpur, Pakistan'",
"'Dubai, UAE'",
"'Leesburg, FL'",
"'Buenos Aires'",
"'Washington, DC  Charlotte, NC'",
"'Alaska'",
"'Manchester, UK'",
"'Does it really matter'",
"'Olympia, WA'",
"'Buenos Aires, Argentina'",
"'Chile'",
"'Victoria, Australia, Earth'",
```

"'Macon, GA'",
"'Scotland, United Kingdom'",
"'West Virginia, USA'",
"'Philadelphia'",
"'Enfield, UK'",
"'Incognito'",
"'19.600858, -99.047821'",
"'NC'",
"'Dubai'",
"'Adelaide, South Australia'",
"'Madison, GA'",
"'Paris'",
"'Roanoke, VA'",
"'HTX'",
"'New Delhi, Delhi'",
"'Manila, Philippines'",
"'New York '",
"'All around the world'",
"'Lagos Nigeria'",
"'Newcastle Upon Tyne, England'",
"'Kashmir'",
"'ph'",
"'Bakersfield, California'",
"'Huntsville, AL'",
"'Lisbon, Portugal'",
"'Kolkata'",
"'Newark, NJ'",
"'Vietnam'",
"'Mo.City'",
"'Jupiter'",
"'Rock Hill, SC'",
"'Wales'",
"'Swaning Around'",
"'Washington'",
"'Melbourne, Victoria'",
"'Eldoret, kenya'",
"'DC'",
"'Selma2Oakland'",
"'Columbia, SC'",
"'Suplex City'",
"'Derby'",
"'Mesa, AZ'",
"'Accra,Ghana'",
"'Niagara Falls, Ontario'",
"'Liverpool'",
"'New Zealand'",
"'Orlando, Fl'",

"'Durham, NC'",
"'The American Wasteland MV'",
"'http//www.amazon.com/dp/B00HR'",
"'Baton Rouge, LA'",
"'Winnipeg'",
"'Vancouver BC'",
"'New Sweden'",
"'Fresno, CA'",
"'Mackay, QLD, Australia'",
"'Jakarta'",
"'United States of America'",
"'Orange County, California'",
"'Unknown'",
"'SÌo Paulo'",
"' '",
"'Thailand'",
"'Desde Republica Argentina'",
"'va'",
"'St Paul, MN'",
"'Muntinlupa City, Philippines'",
"'Washington, USA'",
"'Macclesfield'",
"'Costa Rica'",
"'Arizona '",
"'Your screen'",
"'The Universe'",
"'New England'",
"'TX'",
"'Venezuela'",
"'Dimes Palace'",
"'Epic City, BB.'",
"'South Stand'",
"'Dundee'",
"'Helsinki'",
"'Israel'",
"'Hackney, London'",
"'Boise, Idaho'",
"'rome'",
"'Reddit '",
"'Federal Capital Territory'",
"'Spare Oom'",
"'Portugal'",
"'Yamaku Academy, Class 3-4'",
"'EastCarolina'",
"'anzio,italy'",
"'Palo Alto, CA'",
"'Michigan, USA'",

```
"'Riyadh'",
"'Manchester, NH'",
"'USA Formerly @usNOAAgov'",
"'Karachi, Pakistan'",
"'Winnipeg, Manitoba'",
"'Live On Webcam'",
"'Norway'",
"'Auckland'",
"'Mumbai , India'",
"'434'",
"'New Delhi, India'",
"'Erie, PA'",
"'Alexandria, VA'",
"'Bournemouth'",
"'Houston TX'",
"'Miami, FL'",
"'Greensboro, North Carolina'",
"'107-18 79TH STREET'",
"'Bandung'",
"'Philadelphia, PA '",
"'Los Angeles '",
"'Spain'",
"'Houston'",
"'Charlotte NC'",
"'New York, New York'",
"'SF Bay Area'",
"'Trinidad and Tobago'",
"'Brisbane'",
"'Ukraine'",
"'Ohio, USA'",
"'Evanston, IL'",
"'Rocky Mountains'",
"'In the potters hands'",
"'Dallas Fort-Worth'",
"'EIC'",
"'Odawara, Japan'",
"'Wilmington, NC'",
"'England, United Kingdom'",
"'Georgia, USA'",
"'Baltimore'",
"'ÌÏT 10.614817868480726,12.195582811791382'",
"'Charlotte'",
"'WA State'",
"' Queensland, Australia'",
"'Sand springs oklahoma'",
"'St Charles, MD'",
"'617-BTOWN-BEATDOWN'",
```

"'Colombia'",
"'Honduras'",
"'Berlin - Germany'",
"'CA physically- Boston Strong'",
"'Silver Spring, MD'",
"'302'",
"'Renfrew, Scotland'",
"'dallas'",
"'Palo Alto, California'",
"'Trinidad Tobago'",
"'Detroit'",
"'140920-21 150718-19 BEIJING'",
"'Somalia'",
"'they/them '",
"'Oakland'",
"'Cleveland, OH - San Diego, CA'",
"' Dreamz'",
"'Sydney Australia'",
"'Spokane, WA'",
"'San Diego California 92101'",
"'#HarleyChick#PJNT#RunBenRun'",
"'hell'",
"'World Wide Web'",
"'Louisiana'",
"'GLOBAL'",
"'everywhere'",
"'Phoenix'",
"'Roaming around the world'",
"'Budapest, Hungary'",
"'Cherry Creek Denver CO'",
"'Street of Dallas'",
"'texas'",
"'Merica'",
"'Bangalore, India'",
"'Decatur, GA'",
"'Ontario Canada'",
"'Ashburn, VA'",
"'Minna, Nigeria'",
"'world'",
"'San Jose, California'",
"'Gold Coast, Australia'",
"'Moncton, New Brunswick'",
"'Moscow'",
"'Pretoria'",
"'Here.'",
"'Washington State'",
"'Gainesville, FL'",

"'Istanbul'",
"'Auckland, New Zealand'",
"'Instagram - @heyimginog '",
"'Perth, Western Australia'",
"'vancouver usa'",
"'NIGERIA'",
"'MY RTs ARE NOT ENDORSEMENTS'",
"'Slappin and Smackin '",
"'Madrid, Comunidad de Madrid'",
"'Melbourne-ish'",
"'Pocatello, ID'",
"'FSC 19'",
"'emily helen shelley '",
"'Utica NY'",
"'TonyJ@Centralizedhockey.com'",
"'Honolulu,Hawaii '",
"'Beacon Hills'",
"'Sao Paulo'",
"'Noida, NCR, India'",
"'Galveston, Texas'",
"'Est. September 2012 - Bristol'",
"'the own zone layer '",
"'Colorado/WorldWide'",
"'balvanera'",
"'Kuwait'",
"'Harpurhey, Manchester, UK'",
"'The Globe'",
"'Plano, Texas'",
"'Guatemala'",
"'Cavite, Philippines'",
"'Westerland'",
"'Belgrade'",
"'Gloucester'",
"'Lancaster, CA'",
"'front row at a show'",
"'twitch.tv/naturalemblem26'",
"'Iowa, USA'",
"'San Diego CA'",
"'U.S'",
"'261 5th Avenue New York, NY '",
"'Brackley Beach, PE, Canada'",
"'Pueblo, CO'",
"'Jerseyville, IL'",
"'San Diego, California'",
"'liverpool '",
"'Waterloo, ON'",
"'San Luis Obispo, CA'",

```
"'Positive 852'",
"'Namjoons pants'",
"'Ciudad AutÌ_noma de Buenos Aires, Argentina'",
"'Soufside'",
"'Fredericksburg, Virginia'",
"'Over the Moon…'",
"'North Ferriby, East Yorkshire'",
"'NY  live easy '",
"'Tennessee/Gallifrey'",
"'  News'",
"'Fort Wayne, IN'",
"'Stanford University'",
"'South of D.C.'",
"'London, Riyadh'",
"'Somewhere out there'",
"'NYCNJ'",
"'st.louis county missouri '",
"'Kamloops, BC'",
"'San Diego, Calif.'",
"'Pioneer Village, KY'",
"'Tarragona'",
"'Land Of The Kings'",
"'planet earth'",
"'The  below '",
"'Christchurch New Zealand'",
"'New Your'",
"'indiana'",
"'Lytham St Annes '",
"'Kenton, Ohio'",
"'College Station, TX'",
"'Funtua, Nigeria'",
"'Washington DC / Nantes, France'",
"'Montreal'",
"'Portsmouth, VA'",
"'The Epicenter, and Beyond'",
"'Asia European Continent Korea '",
"'on to the next adventure'",
"'Surry Hills, Sydney'",
"'Orlando '",
"'Fife, WA'",
"' Bouvet Island'",
"'Rural Northern Nevada'",
"'Lincoln, IL'",
"'Paulton, England'",
"' Philly Baby '",
"'In the moment'",
"'Pacific Northwest'",
```

```
"' Eugene, Oregon'",
"'Phoenix Az'",
"'Belfast'",
"'ljp/4'",
"'Huntington, WV'",
"'South Carolina, USA'",
"'cork'",
"'Somewhere Out There'",
"'Tacoma,Washington'",
"'Dorset, United Kingdom'",
"'Banbridge'",
"'Palestine '",
"'Astley, Manchester'",
"'Charleston, WV'",
"'Miami, Florida'",
"'El Paso, TX'",
"'Magnolia, Fiore '",
"'Leicester, England'",
"'Lynchburg, VA'",
"'#freegucci'",
"'La Puente, CA'",
"'West Hollywood'",
"'norcal'",
"' Cloud Mafia '",
"'Spain but Opa-Locka, FL'",
"'Groningen, Netherlands, Europe'",
"'Fort Walton Beach, Fl'",
"'Live4Heed'",
"'taking bath do not disturb'",
"'Hillsville/Lynchburg, VA'",
"'Santo Domingo Alma Rosa '",
"' The World'",
"'Area 8 '",
"'ECSU16'",
"' CA Û GA  '",
"'Rhyme Or Reason'",
"'Ktx'",
"'Vail Valley'",
"'Sale, England'",
"'khanna'",
"'Inverness, Nova Scotia'",
"'Instagram trillrebel_'",
"'  å_ '",
"'Right next to Compton'",
"'France'",
"'NYC metro'",
"'Tring '",
```

```
"'Burlington, VT'",
"'Western New York'",
"'Cimahi,West Java,Indonesia'",
"'on twitter '",
"'Here, unless there.  '",
"'wherever the  at'",
"'Selangor'",
"'Wellington, New Zealand'",
"'On the court '",
"'Boulder'",
"'@notoriousD12'",
"'94123'",
"'Antioch, CA '",
"'Brooklyn'",
"'Dagenham, Essex'",
"'RP'",
"'wisco'",
"'New Jersey/New York'",
"'Cambridge, MA'",
"'in Dimitris arms'",
"'Cobblestone'",
"'ANYWEHERE '",
"'627'",
"'Paname City'",
"'The Harbinger.'",
"'Bahrain'",
"'Chicago Heights, IL'",
"'Giddy, Greenland'",
"'The Multiverse'",
"'The green and pleasant land.'",
"'Minneapolis/St. Paul'",
"'Plano,TX'",
"'Holly, MI'",
"'#KaumElite#FVOR#SMOFC'",
"'AKRON OHIO USA'",
"'weit Û ixwin'",
"'Maryland,Baltimore'",
"'Gujranwala, Pakistan'",
"'Guayaquil'",
"'Top secret bunker '",
"'Soul Somalia/Body Montreal'",
"'Oblivion'",
"'Benton City, Washington'",
"'Bishops Stortford, England'",
"'Louisville, KY'",
"'Utah, USA'",
"'right here'",
```

```
"'heccfidmss@gmail.com'",
"' Blonde Bi Fry. '",
"'Peshawar'",
"'Colombo,Sri Lanka.'",
"'Nashua NH'",
"'CT  NYC'",
"'Chicago, but Philly is home'",
"'Wakefield, West Yorkshire'",
"'my house'",
"'i beg vines sorry '",
"'Hampstead, London.'",
"'Charlotte, N.C.'",
"'nj'",
"'52.479722, 62.184971'",
"'Hailing from Dayton '",
"'Elmwood Park, NJ'",
"'oman muscat al seeb '",
"'Miami Beach, Fl'",
"'Palma, Islas Baleares'",
"'Whole World '",
"'Mumbai india'",
"'The Hammock, FL, USA'",
"'Playa del Carmen, Mexico'",
"'Arlington, VA and DC'",
"'ÌÏT 33.209923,-87.545328'",
"'New Jersey/ D.R.'",
"'New Britain, CT'",
"'Nakhon Si Thammarat'",
"'Madison, WI  St. Louis MO'",
"'ON'",
"'Orlando'",
"'Rochester Hills, MI'",
"'Still. S.A.N.D.O.S'",
"'Not Of This World'",
"'The Suns Corona'",
"'sri lanka'",
"'toronto'",
"'Arkansas, Jonesboro'",
"'Youngstown, OH'",
"'Alliston Ontario'",
"'NIFC'",
"'Photo  Blue Mountains '",
"'Wandsworth, London'",
"'USA, North Dakota'",
"'Bhubneshwar'",
"'Moore, OK'",
"'West Palm Beach, Florida'",
```

```
"'EARTH'",
"'Amsterdam'",
"'PSN Pipbois '",
"'Highland Park, CA'",
"'Enterprise, Alabama'",
"'One World'",
"'Poconos'",
"'Pune, Maharashtra'",
"'Miami via Lima'",
"'Montana, USA'",
"'POFFIN'",
"'DMV'",
"'WAISTDEEP, TX'",
"'Washington, DC NATIVE'",
"'Newcastle upon Tyne'",
"'Philadelphia, PA USA'",
"'Durand, MI'",
"'Earth '",
"'Brentwood, NY'",
"'Woodcreek HS, Roseville, CA'",
"'Cameroon'",
"'Los Angeles for now'",
"'Dundee, UK'",
"'Hemel Hempstead'",
"'Norwalk, CT'",
"'Manhattan'",
"'Broadview Heights, Ohio'",
"'La Grange Park, IL'",
"'Saudi Arabia'",
"'Ljubljana, Slovenia'",
"'British girl in Texas'",
"'Melbourne, Florida'",
"'Atlanta,Ga'",
"'Aix-en-Provence, France'",
"'Cleveland, Ohio'",
"'Some pum pum'",
"'Whippany, NJ'",
"'Pomfret/Providence'",
"'IN our hearts  Earth Global '",
"'Mid West'",
"'Elsewhere, NZ'",
"'ATLANTA , GEORGIA '",
"'Florida Forever'",
"'… -.- -.--'",
"'Minneapolis, MN'",
"'Olathe, KS'",
"'Former Yugoslav Republic of Macedonia'",
```

"'#Bummerville otw'",
"'Adelaide'",
"'SaudI arabia - riyadh '",
"'japon'",
"'Serva Fidem'",
"'Gidi'",
"'www.twitch.tv/PKSparkxx'",
"'NY Capital District'",
"'denver colorado'",
"'Caribbean'",
"'El Paso, Texas'",
"'Nashville, Tennessee'",
"'Mysore, Karnataka'",
"'Innerhalb der LÌ_cke'",
"'Vidalia GA'",
"'Adventuring in Narnia'",
"'Charleston S.C.'",
"'God.Family.Money'",
"'Danville, VA'",
"'have car will travel'",
"'travelling to taes pants'",
"'Greeley, CO'",
"'Pontevedra, Galicia'",
"'Alexandria, VA, USA'",
"'Nevada wishing for Colorado'",
"'Tampa-St. Petersburg, FL'",
"'ÌÏT 40.562796,-75.488849'",
"'Marbella. Spain'",
"'Los Angles, CA'",
"'Attock'",
"'Albany/NY'",
"'US-PR'",
"'Brisbane.'",
"'Hampshire UK'",
"'Watch Those Videos -'",
"'South Pasadena, CA'",
"'Proudly frozen Canuck eh '",
"'Suburban Detroit, Michigan'",
"'Kuala Lumpur'",
"'glasgow'",
"'Dallas, TX '",
"'Georgia Tennessee'",
"'Vancouver, Canada'",
"'Concord, CA'",
"'Nowhere Islands/Smash Manor'",
"'lagos nigeria'",
"'Chicago Area'",

"'Canberra, Australian Capital Territory'",
"'36 38'",
"'Raleigh Garner/Cleveland NC'",
"'i luv raquel'",
"'Where the money at'",
"'Earth, Milky Way, Universe'",
"'Rogersville, MO'",
"'CA DC'",
"'The TARDIS'",
"'Gumptown'",
"'Rutherfordton, NC'",
"'Hyderabad Telangana INDIA'",
"'Bolton Tewkesbury, UK'",
"'Canadian bread'",
"'Rafael castillo'",
"'Rheinbach / Germany'",
"'someplace living my life'",
"'Dammam- KSA'",
"'Lima, OH'",
"'fujo garbage heaven '",
"'SÌo Paulo, Brasil'",
"'Stowmarket'",
"'Watertown, Mass.'",
"'statesboro/vidalia'",
"'Detroit, MI, United States'",
"'Santiago Bernabeau'",
"'Viterbo BFA Acting 18'",
"'UGA 15 Alumnus - Economics '",
"'Washington, D.C. '",
"'New South Wales, Australia'",
"'PekanbaruåBatam IslandåMedan'",
"'UK, Republic of Ireland and Australia'",
"'sÌo luis'",
"'Pawnee'",
"'Hame'",
"'WORDLDWIDE'",
"'Downtown Churubusco, Indiana'",
"'iPhone 33.104393,-96.628624'",
"'cody, austin follows '",
"'Not a U.S resident'",
"'Aarhus, Central Jutland'",
"'sss a'",
"'518'",
"'#MadeInNorthumberland'",
"'Lancashire, United Kingdom'",
"'prob turning up with sheen'",
"'The Wood'",

```
"'nap queen'",
"'mind ya business'",
"'U.S.A and Canada'",
"'Phila.'",
"'Sugar Land, TX'",
"'Port Jervis, NY'",
"'Dubai, United Arab Emirates'",
"'World news'",
"'golborne, north west england.'",
"'Monterrey, MÌxico'",
"'infj '",
"'In Your Notifications '",
"'West Midlands'",
"'Somewhere in China.'",
"'Cassadaga Florida'",
"'London / Birmingham'",
"'NC  OR'",
"'Gurgaon, Haryana. '",
"'709'",
"'Jersey City, New Jersey'",
"'Erbil'",
"'Traverse City, MI'",
"'Sioux Falls, SD'",
"'SEC Country'",
"'Im In Route '",
"'Sioux Falls, S.D. '",
"'Suginami-ku, Tokyo, Japan'",
"'Mariveles, Bataan'",
"'Emirates'",
"'Torry Alvarez love forever  '",
"'Cardiff, UK'",
"'with Doflamingo'",
"'The Web'",
"'death star'",
"'Freeport il '",
"'Extraterrestrial Highway'",
"'Center for Domestic Preparedness'",
"'Mooresville, NC'",
"'Valle Del Sol'",
"'nearest trash can '",
"'@ ForSL/RP'",
"'my deli'",
"'russia'",
"'Back East in PA'",
"'Gwersyllt, Wales'",
"'OES 4th Point. sisSTAR  TI'",
"'London/Outlaw Country '",
```

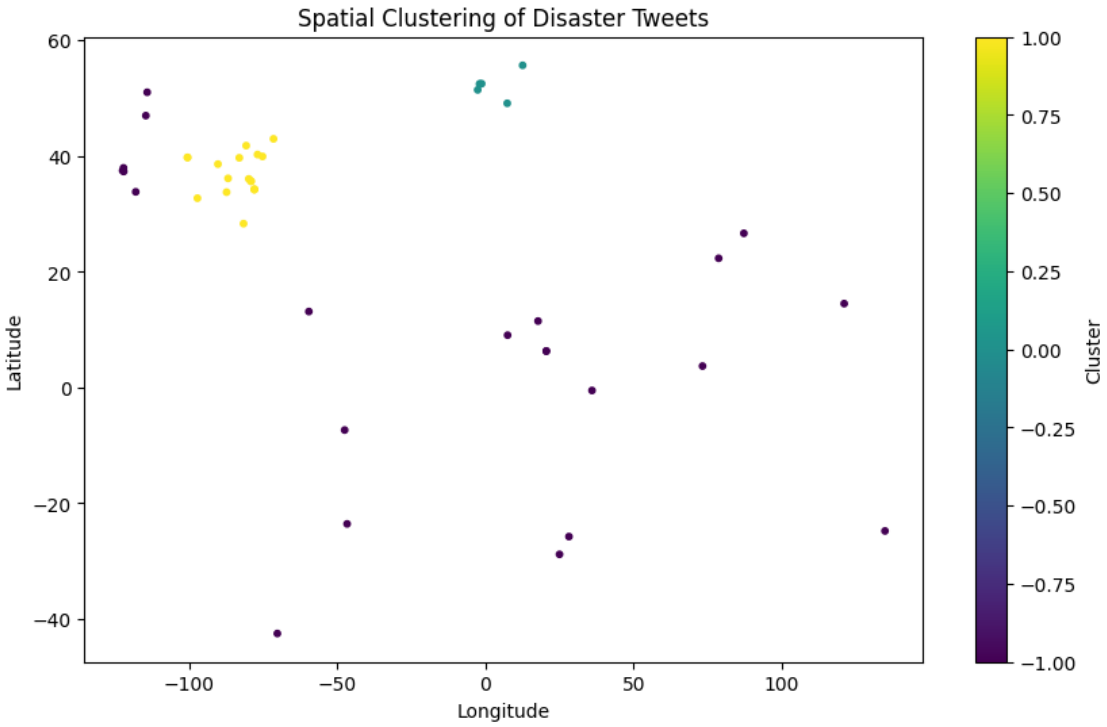
```
"'IDN'",
"'Earthling For now'",
"'Aro Diaspora'",
"'QUEENS.'",
"'IG AyshBanaysh'",
"'Christiana,Tennessee'",
"'where the wild things are'",
"'Lucknow, India'",
"'Ashford, Kent, United Kingdom'",
"', Thailand '",
"'justin  ari follow  tvd'",
"'italy'",
"'somewhere over a rainbow'",
"'Viejo'",
"'Antigua  NYC '",
"'Wood Buffalo, Alberta'",
"'Maricopa, AZ'",
"'Ecuador'",
"'www.facebook.com/stuntfm'",
"'North Memphis/Global Citizen'",
"'New York - Connecticut'",
"'Bronx, NY'",
"'Halifax, Nova Scotia'",
"'Central Illinois'",
"'Itirapina, SÌo Paulo'",
"'#WhereverImAt'",
"'See the barn of bleakness'",
"'48.870833,2.399227'",
"'Dreieich, Germany'",
"'Jacksonville Beach, FL'",
"'Kuwait '",
"'Somewhere in Jersey'",
"'Telangana'",
"'Thibodaux, LA'",
"'Lahore'",
"'Philadelphia, Pennsylvania USA'",
"' New Delhi '",
"'#GDJB #ASOT'",
"'ÌÏT 39.982988,-75.261624'",
"'Bangalore. India'",
"'Alaska, USA'",
"'not so cool KY'",
"'Ventura'",
"'Darnley, Prince Edward Island'",
"'In Space'",
"'garowe puntland somalia'",
"'Alger-New York-San Francisco'",
```

```
 "'dublin '",
 "'The Canopy Kingdom'",
 "'Bleak House'",
 "'Netherlands,Amsterdam-Virtual '",
 "'Temecula, CA'",
 "'Insula Barataria'",
 "'Eugene, Oregon'",
 "'he/him'",
 "'GOT7SupportPH'",
 "'Garden Grove'",
 "'New Haven, Connecticut'",
 "'    .'",
 "'di langit 7 bidadari , '",
 …]
```

[24]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf, col
from pyspark.sql.types import DoubleType
from opencage.geocoder import OpenCageGeocode
import pandas as pd
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Initialize Spark
spark = SparkSession.builder.appName("Geocoding").getOrCreate()

# Initialize OpenCage Geocoder
API_KEY = "eeb5797da3e944068c99c92df273961d"  # Replace with your actual key
geocoder = OpenCageGeocode(API_KEY)

# Define UDF for geocoding
def get_coordinates(location):
    try:
        if location:
            result = geocoder.geocode(location)
            if result:
                return (float(result[0]['geometry']['lat']),␣
 ↪float(result[0]['geometry']['lng']))
    except Exception:
        pass
    return (None, None)

# Register UDF
get_coords_udf = udf(get_coordinates, "struct<lat:double,lon:double>")
```

```
# Apply geocoding to first 500 rows
geo_df = df.limit(100).withColumn("coords", get_coords_udf(col("location")))

# Extract latitude and longitude
geo_df = geo_df.select(
    "*",
    col("coords.lat").alias("latitude"),
    col("coords.lon").alias("longitude")
).drop("coords")

# Drop rows with missing coordinates
geo_df = geo_df.na.drop(subset=["latitude", "longitude"])

# Convert to Pandas for clustering (only if dataset is small)
pdf = geo_df.select("latitude", "longitude").toPandas()

# Standardize coordinates
scaler = StandardScaler()
coords_scaled = scaler.fit_transform(pdf[['latitude', 'longitude']])

# Apply DBSCAN
dbscan = DBSCAN(eps=0.3, min_samples=5)
pdf['cluster'] = dbscan.fit_predict(coords_scaled)

# Plot clusters
plt.figure(figsize=(10, 6))
plt.scatter(pdf['longitude'], pdf['latitude'], c=pdf['cluster'],␣
 ↪cmap='viridis', s=10)
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Spatial Clustering of Disaster Tweets')
plt.colorbar(label='Cluster')
plt.show()
```

```
25/04/02 18:44:50 WARN SparkSession: Using an existing Spark session; only
runtime SQL configurations will take effect.
```



[25]:
```
from sklearn.cluster import DBSCAN
import numpy as np

# Convert PySpark DataFrame to Pandas
geo_pd_df = geo_df.toPandas()

# Ensure no missing values in latitude and longitude
geo_pd_df = geo_pd_df.dropna(subset=['latitude', 'longitude'])

# Run DBSCAN clustering (assuming you haven't assigned clusters yet)
dbscan = DBSCAN(eps=0.3, min_samples=5)
geo_pd_df['cluster'] = dbscan.fit_predict(geo_pd_df[['latitude', 'longitude']])

# Now try plotting
import folium

# Create a Folium world map
m = folium.Map(location=[20, 0], zoom_start=2)

# Define colors for clusters
colors = ['red', 'blue', 'green', 'purple', 'orange', 'darkred', 'lightblue']

# Add cluster markers to the map
```

```
for _, row in geo_pd_df.iterrows():
    cluster_id = row['cluster']
    color = colors[cluster_id % len(colors)] if cluster_id != -1 else "gray"

    folium.CircleMarker(
        location=[row['latitude'], row['longitude']],
        radius=5,
        color=color,
        fill=True,
        fill_color=color,
        fill_opacity=0.7,
        popup=f"Cluster {cluster_id}"
    ).add_to(m)

# Show the map
m
```

[25]:
```
<folium.folium.Map at 0x7fc59f655eb0>
```

[26]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType, IntegerType
import re
import string
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd

# Initialize Spark Session
spark = SparkSession.builder.appName("ML_Pipeline").getOrCreate()

# Drop null values in text and target columns
df = df.dropna(subset=["text", "target"])

# Convert target column to integer
df = df.withColumn("target", df["target"].cast(IntegerType()))

# Define text cleaning function
def clean_text(text):
    if text is None:
        return ""
    text = text.lower()
```

```
    text = re.sub(f"[{string.punctuation}]", "", text)  # Remove punctuation
    text = re.sub(r"\d+", "", text)  # Remove numbers
    return text

# Convert function to PySpark UDF
clean_text_udf = udf(clean_text, StringType())

# Apply UDF to create a new cleaned text column
df = df.withColumn("clean_text", clean_text_udf(df["text"]))

# Convert Spark DataFrame to Pandas for ML processing
pdf = df.select("clean_text", "target").toPandas().dropna()

# Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(pdf["clean_text"],␣
 ↪pdf["target"], test_size=0.2, random_state=42)

# Build ML Pipeline
pipeline = Pipeline([
    ("tfidf", TfidfVectorizer()),
    ("classifier", RandomForestClassifier(n_estimators=100, random_state=42))
])

# Train Model
pipeline.fit(X_train, y_train)

# Predictions
y_pred = pipeline.predict(X_test)

# Evaluate Model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Model Accuracy: {accuracy:.2f}")
print(report)
```

```
25/04/02 18:47:36 WARN SparkSession: Using an existing Spark session; only
runtime SQL configurations will take effect.
```

```
Model Accuracy: 0.78
              precision    recall  f1-score   support

           0       0.75      0.91      0.82       781
           1       0.86      0.63      0.73       655

    accuracy                           0.78      1436
   macro avg       0.80      0.77      0.77      1436
weighted avg       0.80      0.78      0.78      1436
```

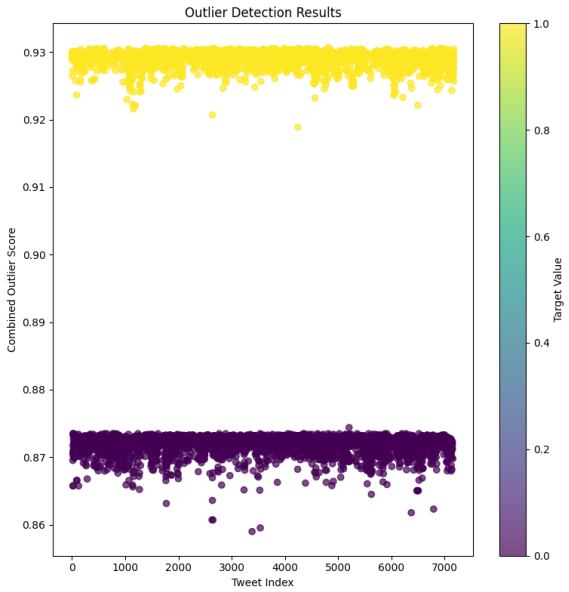
[27]:
```
from pyspark.sql import SparkSession
import pandas as pd
import numpy as np
import networkx as nx
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from scipy.stats import zscore
import matplotlib.pyplot as plt

# Initialize Spark Session
spark = SparkSession.builder.appName("EnhancedOutlierDetection").getOrCreate()

# Load data with correct column names
pandas_data = df.select("id", "clean_text", "target").toPandas()

# 1. Enhanced Text Similarity Analysis
print("Calculating text similarity...")
vectorizer = TfidfVectorizer(max_features=5000, stop_words='english')
tfidf_matrix = vectorizer.fit_transform(pandas_data['clean_text'])
similarity_matrix = cosine_similarity(tfidf_matrix)

# 2. Graph Construction with Adaptive Threshold
print("Building similarity graph...")
mean_sim_value = np.mean(similarity_matrix)  # This is already a numpy float
# Use np.maximum instead of max to avoid PySpark interception
threshold = np.maximum(0.2, mean_sim_value * 1.5)  # Dynamic threshold

G = nx.Graph()
for idx, row in pandas_data.iterrows():
    G.add_node(row['id'], text=row['clean_text'], target=row['target'])

for i in range(len(pandas_data)):
    for j in range(i+1, len(pandas_data)):
        if similarity_matrix[i,j] > threshold:
            G.add_edge(pandas_data.iloc[i]['id'], pandas_data.iloc[j]['id'],
                       weight=similarity_matrix[i,j])

# 3. Enhanced Outlier Detection
print("Calculating outlier scores...")
# Graph-based metrics
degree_centrality = nx.degree_centrality(G)
betweenness = nx.betweenness_centrality(G, weight='weight')  # Use edge weights

# Calculate combined outlier score
pandas_data['graph_outlier_score'] = pandas_data['id'].apply(
```

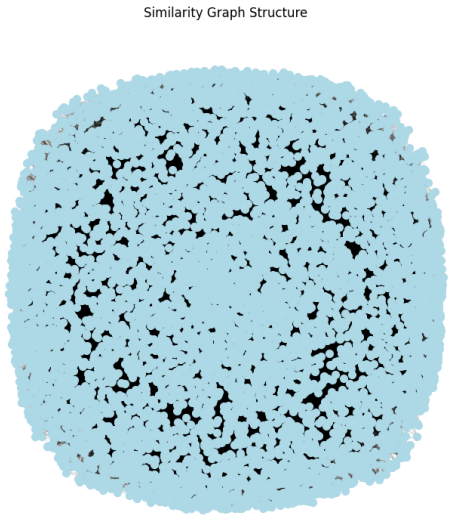
```
    lambda x: 0 if x not in degree_centrality else
    (1 - degree_centrality[x]) * (1 + betweenness.get(x, 0))
)

# Statistical analysis of target values
pandas_data['target_zscore'] = zscore(pandas_data['target'])
pandas_data['target_iqr'] = 0  # Initialize
Q1, Q3 = np.percentile(pandas_data['target'], [25, 75])
IQR = Q3 - Q1
if IQR > 0:  # Only calculate if there's variation
    pandas_data['target_iqr'] = (
        (pandas_data['target'] < (Q1 - 1.5*IQR)) |
        (pandas_data['target'] > (Q3 + 1.5*IQR))
    ).astype(int)

# Combine metrics into final outlier score
pandas_data['combined_outlier_score'] = (
    pandas_data['graph_outlier_score'] * 0.7 +
    pandas_data['target_zscore'].abs() * 0.2 +
    pandas_data['target_iqr'] * 0.1
)

# 4. Visualization
print("Generating visualizations...")
plt.figure(figsize=(15, 8))

# Plot 1: Outlier scores
plt.subplot(1, 2, 1)
plt.scatter(
    range(len(pandas_data)),
    pandas_data['combined_outlier_score'],
    c=pandas_data['target'],
    cmap='viridis',
    alpha=0.7
)
plt.colorbar(label='Target Value')
plt.xlabel('Tweet Index')
plt.ylabel('Combined Outlier Score')
plt.title('Outlier Detection Results')

# Plot 2: Graph visualization (simplified)
plt.subplot(1, 2, 2)
pos = nx.spring_layout(G, k=0.15, iterations=20)
nx.draw_networkx_nodes(G, pos, node_size=50, node_color='lightblue')
nx.draw_networkx_edges(G, pos, alpha=0.1)
plt.title('Similarity Graph Structure')
plt.axis('off')
```

```
plt.tight_layout()
plt.show()

# 5. Results Analysis
print("\n=== Results Summary ===")
print(f"Identified {len(G.nodes())} nodes and {len(G.edges())} edges")
print(f"Similarity threshold used: {threshold:.4f}")
print(f"Average degree centrality: {np.mean(list(degree_centrality.values())):.
  ↪4f}")
print(f"Average betweenness centrality: {np.mean(list(betweenness.values())):.
  ↪4f}")

# Get top outliers
top_outliers = pandas_data.nlargest(10, 'combined_outlier_score')[['id',␣
  ↪'clean_text', 'target', 'combined_outlier_score']]
print("\nTop 10 Potential Outliers:")
print(top_outliers.to_string(index=False))
```

```
25/04/02 18:48:02 WARN SparkSession: Using an existing Spark session; only
runtime SQL configurations will take effect.
```

```
Calculating text similarity…
Building similarity graph…
Calculating outlier scores…
Generating visualizations…
```





```
=== Results Summary ===
Identified 7176 nodes and 73816 edges
Similarity threshold used: 0.2000
Average degree centrality: 0.0029
Average betweenness centrality: 0.0004

Top 10 Potential Outliers:
   id
clean_text  target  combined_outlier_score
 7306                                                  us navy sidelines
newest subs   defensenewscomus navy sidelines  newest subsd  navy       1
0.930663
  905 stationcdrkelly any support sys  usagov auth taken hostage by blk us
clergyforced  exist youngerampgrossly disfigured by bioterrorismap       1
0.930581
 2089                                                    rt greenharvard
documenting climate changes first major casualty  via greenharvard       1
0.930574
 2657
jorrynja   your bfgfcrush  terell        1                0.930574
 4032                                 gurmeetramrahim  green s welfare force ke
appx  members har time disaster victim ki help ke liye tyar hai       1
0.930574
 6015
theblackshag dannyoneil too toxiccancerdiseasehazardous wastenoxious       1
0.930574
 6407
my back is so sunburned        1                0.930574
10310                                rare insight into terror and how to fight it
cameroon usa whitehouse es fr nigeria uk africa de ca au jp       1
0.930574
10519                                    solitude fire update august   solitude
wildfire summary this lightningcaused fire is being   utfire       1
0.930574
 5717                                  ashenforest floorburnt manzanita amp
timber on johnson fire along ridge on forest road  routecomplex        1
0.930482
```

[30]:
```
df
```

[30]:
```
DataFrame[id: string, keyword: string, location: string, text: string, target:
 int, length: int, clean_text: string]
```

[33]:
```
pandas_df = df.toPandas()
# Create dummy timestamps (sequential hours)
pandas_df['timestamp'] = pd.date_range(start='2023-01-01',␣
 ↪periods=len(pandas_df), freq='H')
```

```
anomalies_df = detect_keyword_spikes(pandas_df['text'], pandas_df['timestamp'])
anomalies_df
```

[33]:

	keyword	time	z_score	count
0	help	2023-01-03 19:00:00	6.381339	1.0
1	help	2023-01-03 20:00:00	6.381339	1.0
2	help	2023-01-04 20:00:00	6.381339	1.0
3	help	2023-01-04 21:00:00	6.381339	1.0
4	help	2023-01-11 03:00:00	6.381339	1.0
…	…	…	…	…
1518	danger	2023-06-19 06:00:00	7.204228	1.0
1519	danger	2023-08-03 05:00:00	7.204228	1.0
1520	danger	2023-08-03 06:00:00	7.204228	1.0
1521	danger	2023-10-09 00:00:00	7.204228	1.0
1522	danger	2023-10-09 01:00:00	7.204228	1.0

[1523 rows x 4 columns]

[34]:
```
import numpy as np
import pandas as pd
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, udf
from pyspark.sql.types import FloatType, StringType, ArrayType
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from scipy import stats
from collections import defaultdict

# Try to import sentiment analysis packages with fallbacks
try:
    from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
    vader_available = True
except ImportError:
    vader_available = False
    print("VADER sentiment not available - using TextBlob only")

try:
    from textblob import TextBlob
    textblob_available = True
except ImportError:
    textblob_available = False
    print("TextBlob not available - sentiment analysis disabled")

# Initialize Spark
```

```
spark = SparkSession.builder.appName("DisasterDetection").getOrCreate()

# Load data (example - replace with your data loading code)
# df = spark.read.json("tweets.json")
# For testing, let's create a small sample DataFrame
#data = [("1", "Help needed! Flood in downtown area", "2023-01-01 12:00:00"),
        #("2", "Fire reported in the industrial district", "2023-01-01 12:05:
 ↪00"),
        #("3", "Casual weather update for the weekend", "2023-01-01 12:10:00")]
#df = spark.createDataFrame(data, ["id", "text", "timestamp"])


# 1. Anomaly Detection via Keyword Frequency Spikes
def detect_keyword_spikes(text_series, timestamps, window_size=2,␣
 ↪z_threshold=3):
    keywords = ['help', 'emergency', 'disaster', 'fire', 'flood', 'earthquake',
                'rescue', 'urgent', 'need', 'sos', 'missing', 'danger']

    # Create a DataFrame with timestamps
    df = pd.DataFrame({'text': text_series, 'timestamp': pd.
 ↪to_datetime(timestamps)})
    df = df.set_index('timestamp').sort_index()

    keyword_counts = defaultdict(list)

    for kw in keywords:
        df[kw] = df['text'].str.contains(kw, case=False, na=False).astype(int)
        df[kw + '_count'] = df[kw].rolling(window=window_size, min_periods=1).
 ↪sum()

    anomalies = []
    for kw in keywords:
        counts = df[kw + '_count'].dropna()
        if len(counts) < 2:  # Avoid zero-division error
            continue

        z_scores = np.abs(stats.zscore(counts))
        spike_indices = np.where(z_scores > z_threshold)[0]

        for idx in spike_indices:
            anomalies.append({
                'keyword': kw,
                'time': counts.index[idx],
                'z_score': z_scores[idx],
                'count': counts.iloc[idx]
            })
```

```
    anomalies_df = pd.DataFrame(anomalies)
    return anomalies_df

# Use corrected function
anomalies_df = detect_keyword_spikes(pandas_df['text'], pandas_df['timestamp'])

# Check if anomalies exist before sorting
if not anomalies_df.empty and 'z_score' in anomalies_df.columns:
    print("\nDetected anomalies:")
    print(anomalies_df.sort_values('z_score', ascending=False).head(10))
else:
    print("\nNo anomalies detected.")

# 2. Topic Modeling with LDA
def perform_topic_modeling(texts, n_topics=3):
    vectorizer = CountVectorizer(max_df=0.95, min_df=1, stop_words='english')
    tf = vectorizer.fit_transform(texts.fillna(""))

    lda = LatentDirichletAllocation(n_components=n_topics, random_state=42)
    lda.fit(tf)

    feature_names = vectorizer.get_feature_names_out()
    topics = []
    for topic_idx, topic in enumerate(lda.components_):
        topics.append({
            "topic_id": topic_idx,
            "top_terms": [feature_names[i] for i in topic.argsort()[:-10 - 1:
 ↪-1]]
        })

    return pd.DataFrame(topics), lda.transform(tf)

topics_df, topic_scores = perform_topic_modeling(pandas_df['text'])
print("\nDisaster Topics Identified:")
print(topics_df)

# 3. Sentiment Analysis with fallbacks
def analyze_sentiment(text):
    distress_signals = 0
    distress_phrases = ["help needed", "urgent assistance", "sos", "rescue␣
 ↪needed"]

    # Check for distress signals
    if isinstance(text, str):
        text_lower = text.lower()
        distress_signals = int(any(phrase in text_lower for phrase in␣
 ↪distress_phrases))
```

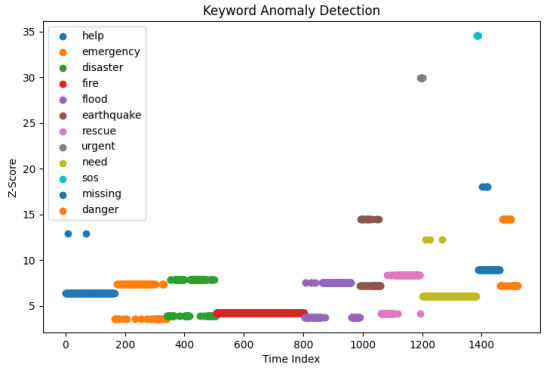
```
    # Initialize default scores
    vader_score = 0.0
    blob_score = 0.0

    # Calculate VADER score if available
    if vader_available and isinstance(text, str):
        vader_score = SentimentIntensityAnalyzer().
 ↪polarity_scores(text)['compound']

    # Calculate TextBlob score if available
    if textblob_available and isinstance(text, str):
        blob_score = TextBlob(text).sentiment.polarity

    return [float(vader_score), float(blob_score), int(distress_signals)]

# Add sentiment analysis to DataFrame
sentiment_udf = udf(analyze_sentiment, ArrayType(FloatType()))
df = df.withColumn("sentiment", sentiment_udf(col("text")))
df = df.withColumn("vader_score", col("sentiment")[0])
df = df.withColumn("blob_score", col("sentiment")[1])
df = df.withColumn("distress_signal", col("sentiment")[2])

# Show results
print("\nSentiment Analysis Results:")
df.select("text", "vader_score", "blob_score", "distress_signal").
 ↪show(truncate=False)

# 4. Visualization
plt.figure(figsize=(15, 10))

# Plot 1: Anomaly detection (placeholder - would need temporal data)
plt.subplot(2, 2, 1)
if not anomalies_df.empty:
    for kw in anomalies_df['keyword'].unique():
        subset = anomalies_df[anomalies_df['keyword'] == kw]
        plt.scatter(subset.index, subset['z_score'], label=kw)
    plt.title('Keyword Anomaly Detection')
    plt.xlabel('Time Index')
    plt.ylabel('Z-Score')
    plt.legend()
else:
    plt.text(0.5, 0.5, 'No anomalies detected', ha='center')

# Plot 2: Topic modeling
plt.subplot(2, 2, 2)
for i, row in topics_df.iterrows():
```

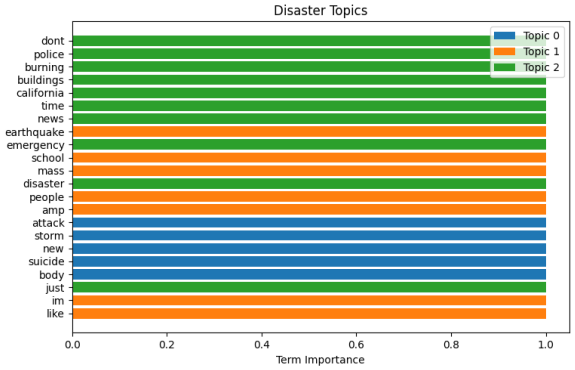
```
    plt.barh(row['top_terms'], [1]*len(row['top_terms']), label=f"Topic {i}")
plt.title('Disaster Topics')
plt.xlabel('Term Importance')
plt.legend()

# Plot 3: Sentiment distribution
plt.subplot(2, 2, 3)
if vader_available:
    plt.hist(df.select("vader_score").toPandas(), bins=10, alpha=0.5,␣
 ↪label='VADER')
if textblob_available:
    plt.hist(df.select("blob_score").toPandas(), bins=10, alpha=0.5,␣
 ↪label='TextBlob')
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment Score')
plt.legend()

plt.tight_layout()
plt.show()

# For classification, you would need labeled data
# This is just a placeholder structure
if 'severity_label' in df.columns:
    from pyspark.ml.feature import VectorAssembler
    from pyspark.ml.classification import RandomForestClassifier
    from pyspark.ml.pipeline import Pipeline

    feature_cols = ["vader_score", "blob_score", "distress_signal"]
    assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
    rf = RandomForestClassifier(labelCol="severity_label",␣
 ↪featuresCol="features")
    pipeline = Pipeline(stages=[assembler, rf])
    # model = pipeline.fit(df)
else:
    print("\nNote: No severity labels found - classification model not trained")
```

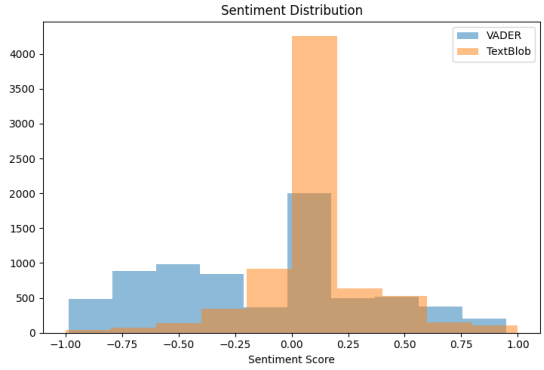
```
Detected anomalies:
     keyword                time    z_score  count
1383     sos 2023-03-29 08:00:00  34.568772    1.0
1384     sos 2023-03-29 09:00:00  34.568772    1.0
1385     sos 2023-05-11 07:00:00  34.568772    1.0
1386     sos 2023-05-11 08:00:00  34.568772    1.0
1387     sos 2023-09-02 04:00:00  34.568772    1.0
1388     sos 2023-09-02 05:00:00  34.568772    1.0
1196  urgent 2023-04-20 14:00:00  29.933259    1.0
1194  urgent 2023-02-16 22:00:00  29.933259    1.0
```

```
1201  urgent 2023-07-09 21:00:00  29.933259    1.0
1200  urgent 2023-07-09 20:00:00  29.933259    1.0

Disaster Topics Identified:
   topic_id                                          top_terms
0         0  [like, im, just, body, suicide, new, storm, at…
1         1  [amp, just, people, im, like, disaster, mass, …
2         2  [news, emergency, time, just, california, buil…

Sentiment Analysis Results:


+--------------------------------------------------------------------------------
-------------------------------------------------+-----------+----------+-----
----------+
|text
|vader_score|blob_score|distress_signal|
+--------------------------------------------------------------------------------
-------------------------------------------------+-----------+----------+-----
----------+
|our deeds are the reason of this earthquake may allah forgive us all
|0.2732     |0.0       |NULL           |
|forest fire near la ronge sask canada
|-0.34      |0.1       |NULL           |
|all residents asked to shelter in place are being notified by officers no other
evacuation or shelter in place orders are expected|-0.296     |-0.01875  |NULL
|
| people receive wildfires evacuation orders in california
|0.0        |0.0       |NULL           |
|just got sent this photo from ruby alaska as smoke from wildfires pours into a
school                                            |0.0        |0.0       |NULL
|
|rockyfire update  california hwy  closed in both directions due to lake county
fire  cafire wildfires                            |-0.34      |-0.1125   |NULL
|
|flood disaster heavy rain causes flash flooding of streets in manitou colorado
springs areas                                     |-0.6249    |-0.2      |NULL
|
|im on top of the hill and i can see a fire in the woods
|-0.1531    |0.5       |NULL           |
|theres an emergency evacuation happening now in the building across the street
|-0.3818    |0.0       |NULL           |
|im afraid that the tornado is coming to our area
|0.0        |-0.6      |NULL           |
|three people died from the heat wave so far
|-0.5574    |0.1       |NULL           |
|haha south tampa is getting flooded hah wait a second i live in south tampa
what am i gonna do what am i gonna do fvck flooding   |0.4588
```

```
|0.11212121|NULL           |
|raining flooding florida tampabay tampa  or  days ive lost count
|-0.3182    |0.0       |NULL           |
|flood in bago myanmar we arrived bago
|0.0        |0.0       |NULL           |
|damage to school bus on  in multi car crash breaking
|-0.7096    |0.0       |NULL           |
|whats up man
|0.0        |0.0       |NULL           |
|i love fruits
|0.6369     |0.5       |NULL           |
|summer is lovely
|0.5859     |0.5       |NULL           |
|my car is so fast
|0.0        |0.2       |NULL           |
|what a gooooooooaaaaaal
|0.0        |0.0       |NULL           |
+-------------------------------------------------------------------------------------
-----------------------------------------------------+-----------+----------+-----
----------+
only showing top 20 rows
```







```
Note: No severity labels found - classification model not trained
```

[35]:
```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, when
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.pipeline import Pipeline



columns = ["text", "vader_score", "blob_score", "distress_signal"]


# Fill missing values to prevent errors
df = df.fillna({"vader_score": 0.0, "blob_score": 0.0, "distress_signal": 0})

# Define Severity Label
df = df.withColumn(
    "severity_label",
    when((col("distress_signal") == 1) | (col("vader_score") < -0.5), 2)  # High severity
    .when((col("vader_score") < 0) | (col("blob_score") < 0), 1)  # Medium severity
    .otherwise(0)  # Low severity
)

df.select("text", "vader_score", "blob_score", "distress_signal", "severity_label").show(truncate=False)

# Feature Vector Assembler
feature_cols = ["vader_score", "blob_score", "distress_signal"]
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features", handleInvalid="keep")

# Random Forest Classifier
rf = RandomForestClassifier(labelCol="severity_label", featuresCol="features", numTrees=10)

# Create Pipeline
pipeline = Pipeline(stages=[assembler, rf])

# Train/Test Split
train_df, test_df = df.randomSplit([0.8, 0.2], seed=42)

# Train Model
```

```
model = pipeline.fit(train_df)

# Predictions
predictions = model.transform(test_df)

# Evaluate Accuracy
evaluator = MulticlassClassificationEvaluator(
    labelCol="severity_label", predictionCol="prediction", metricName="accuracy"
)
accuracy = evaluator.evaluate(predictions)
print(f"\nModel Accuracy: {accuracy:.2f}")

# Show Predictions
predictions.select("text", "severity_label", "prediction").show(truncate=False)
```

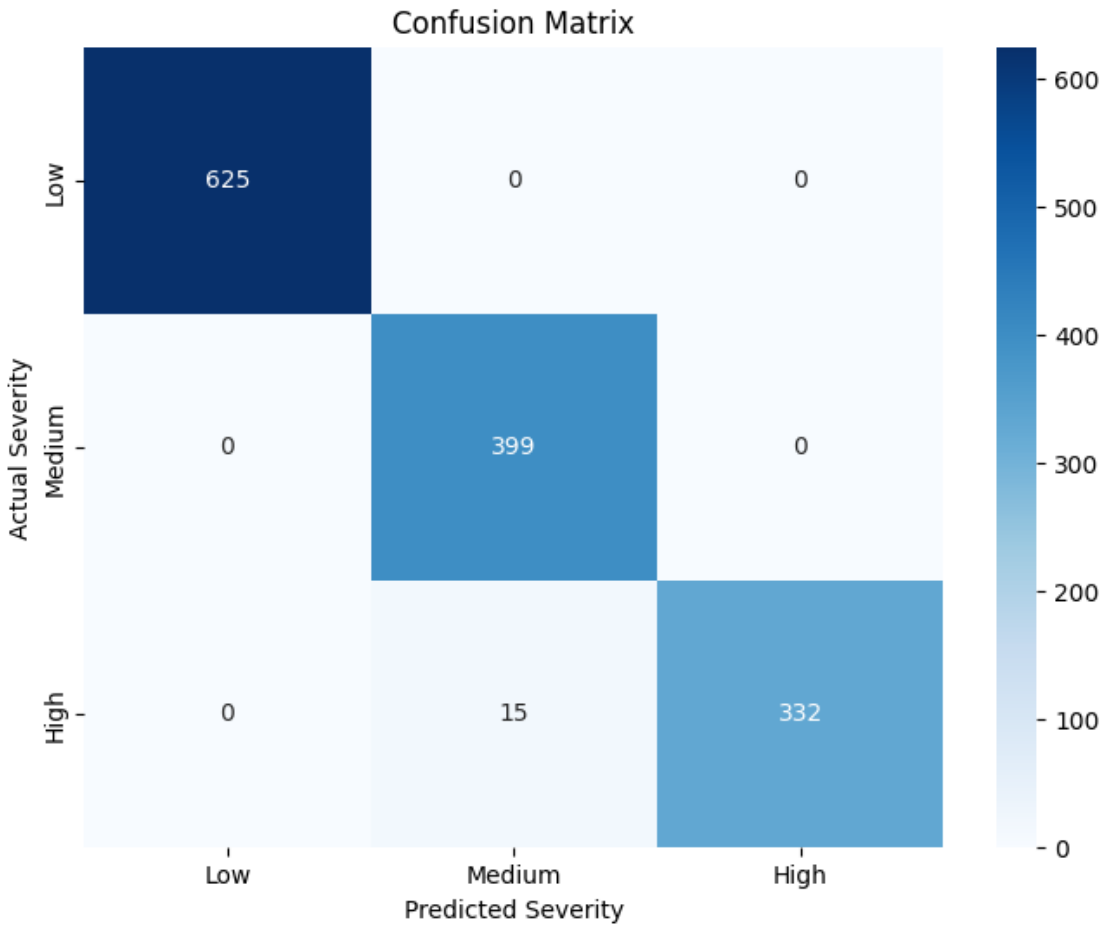
```
+-------------------------------------------------------------------------------
-----------------------------------------------------+-----------+----------+-----
----------+--------------+
|text
|vader_score|blob_score|distress_signal|severity_label|
+-------------------------------------------------------------------------------
-----------------------------------------------------+-----------+----------+-----
----------+--------------+
|our deeds are the reason of this earthquake may allah forgive us all
|0.2732     |0.0       |0.0            |0             |
|forest fire near la ronge sask canada
|-0.34      |0.1       |0.0            |1             |
|all residents asked to shelter in place are being notified by officers no other
evacuation or shelter in place orders are expected|-0.296     |-0.01875  |0.0
|1             |
| people receive wildfires evacuation orders in california
|0.0        |0.0       |0.0            |0             |
|just got sent this photo from ruby alaska as smoke from wildfires pours into a
school                                               |0.0        |0.0       |0.0
|0             |
|rockyfire update  california hwy  closed in both directions due to lake county
fire  cafire wildfires                               |-0.34      |-0.1125   |0.0
|1             |
|flood disaster heavy rain causes flash flooding of streets in manitou colorado
springs areas                                        |-0.6249    |-0.2      |0.0
|2             |
|im on top of the hill and i can see a fire in the woods
|-0.1531    |0.5       |0.0            |1             |
|theres an emergency evacuation happening now in the building across the street
|-0.3818    |0.0       |0.0            |1             |
|im afraid that the tornado is coming to our area
```

```
|0.0        |-0.6      |0.0            |1              |
|three people died from the heat wave so far
|-0.5574    |0.1       |0.0            |2              |
|haha south tampa is getting flooded hah wait a second i live in south tampa
what am i gonna do what am i gonna do fvck flooding   |0.4588
|0.11212121|0.0            |0             |
|raining flooding florida tampabay tampa  or  days ive lost count
|-0.3182    |0.0       |0.0            |1              |
|flood in bago myanmar we arrived bago
|0.0        |0.0       |0.0            |0              |
|damage to school bus on  in multi car crash breaking
|-0.7096    |0.0       |0.0            |2              |
|whats up man
|0.0        |0.0       |0.0            |0              |
|i love fruits
|0.6369     |0.5       |0.0            |0              |
|summer is lovely
|0.5859     |0.5       |0.0            |0              |
|my car is so fast
|0.0        |0.2       |0.0            |0              |
|what a gooooooooaaaaaal
|0.0        |0.0       |0.0            |0              |
+-------------------------------------------------------------------------------
-------------------------------------------------+-----------+----------+-----
----------+--------------+
only showing top 20 rows



Model Accuracy: 0.99
[Stage 101:>                                                          (0 + 1) / 1]
+-------------------------------------------------------------------------------
-------------------------------------------------------------+--------------+
----------+
|text
|severity_label|prediction|
+-------------------------------------------------------------------------------
-------------------------------------------------------------+--------------+
----------+
| had a personalinjury accident this summer read our advice amp see how a
solicitor can help otleyhour                                              |1
|1.0       |
|greenlacey godslove amp thanku my sister for rt of new video  the coming
apocalyptic us earthquake amp tsunami                                     |2
|2.0       |
|erictsunami worry about yourself
```

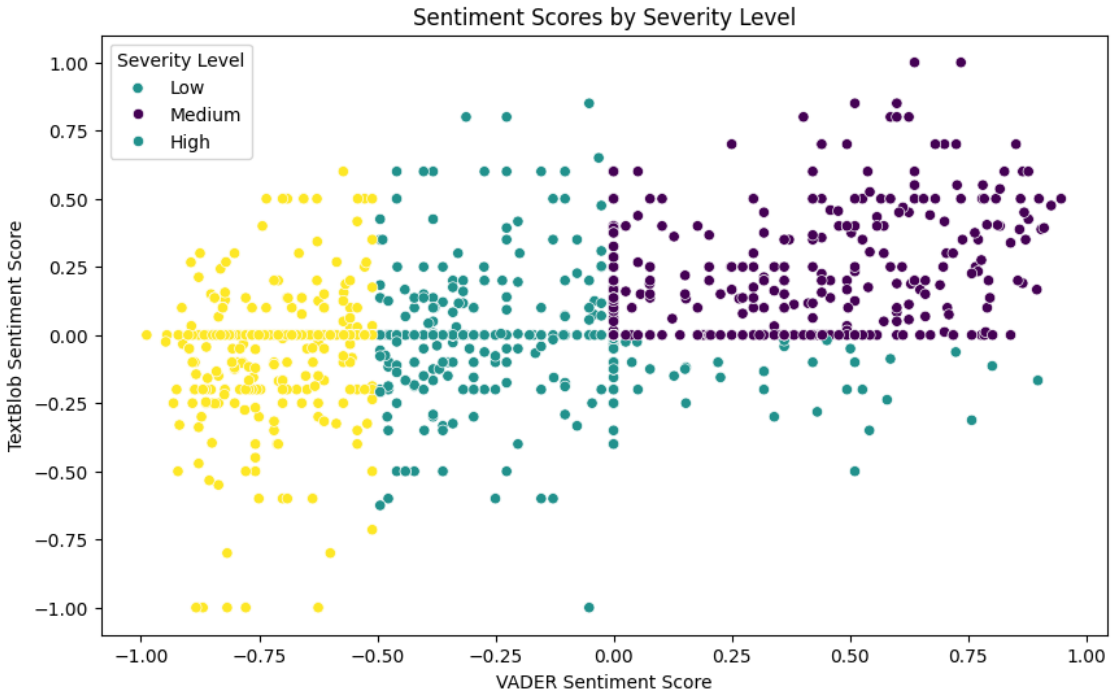
```
|1             |1.0       |
|just stop fucking saying  ûïa whole  û÷nother û  it just sounds fucking stupid
you fucking mean  ûïa whole other û  not a fucking tonguetwister|2
|2.0       |
|why some traffic is freezing cold and some blazing hot  ûò and how to heat up
some of your traffic                                                |1
|1.0       |
|last second outbid rt    catlow  twister swivels   fuel line made in usa please
favorite amp shar                                                 |0
|0.0       |
|crazy mom threw teen daughter a nude twister sex party according to her  gt
|1             |1.0       |
|tornadoes el nino may give canadas twister season a boost toronto
|0             |0.0       |
|mrsbinker  davidlabrava mine are diesel and twister both small for the breed
but very strong i have a beautiful pug too                          |0
|0.0       |
|thebuffshow thetwisterokc my boyfriend wants too see shaniatwain too
twisterlovesshania                                                        |0
|0.0       |
|some curls come out so pretty and some look like i just finished shooting the
movie twister                                                     |0
|0.0       |
|jrlallo my narrator will have to say chemically interesting lavatory for  not
quite the tongue twister but certainly odd p                      |0
|0.0       |
|map typhoon soudelors predicted path as it approaches taiwan expected to make
landfall over southern china by s û                               |1
|1.0       |
|obama declares disaster for typhoondevastated saipan obama signs disaster
declaration for northern marians a                                   |2
|2.0       |
|obama declares disaster for typhoondevastated saipan obama signs disaster
declaration for northern marians a                                   |2
|2.0       |
|losdelsonido obama declares disaster for typhoondevastated saipan obama signs
disaster declaration for northern ma  ivanberroa                  |2
|2.0       |
|photos typhoon soudelor has its aim set on taiwan andåêchina
|0             |0.0       |
|an indepth look at the new world of work and how young people businesses and
economies are coping with huge upheaval                            |0
|0.0       |
|rt livescience ancient mayan tablet with hieroglyphics honors lowly king
|0             |0.0       |
|august  your daily horoscope a relationship upheaval over the next few months
may be disruptive but in the                                      |1
|1.0       |
```

```
+--------------------------------------------------------------------------------
--------------------------------------------------------------+--------------+
----------+
only showing top 20 rows
```

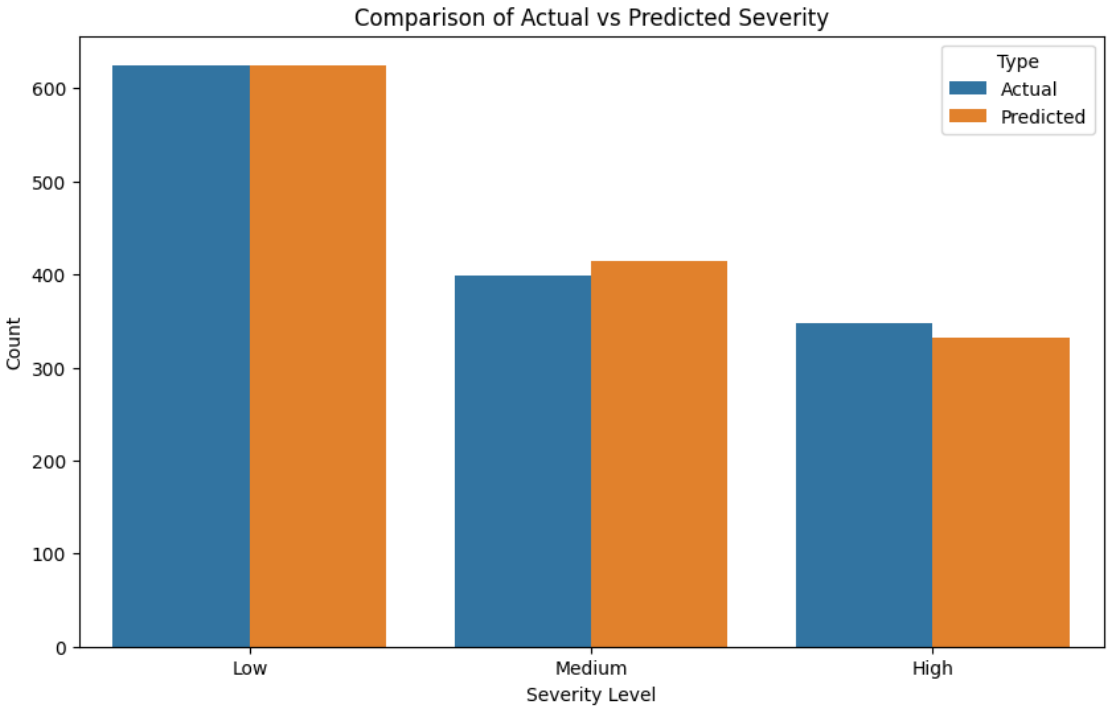
[37]:
```python
from pyspark.mllib.evaluation import MulticlassMetrics
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Convert predictions and labels to float type
predictionAndLabels = predictions.select(
    col("prediction").cast("float"),
    col("severity_label").cast("float")
).rdd

# Instantiate metrics object
metrics = MulticlassMetrics(predictionAndLabels)

# Confusion matrix
confusion_matrix = metrics.confusionMatrix().toArray()

# Plot
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix, annot=True, fmt='g',
            xticklabels=['Low', 'Medium', 'High'],
            yticklabels=['Low', 'Medium', 'High'],
            cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Severity')
plt.ylabel('Actual Severity')
plt.show()
```



[39]:
```
# Convert to pandas for easier plotting
plot_df = predictions.select("vader_score", "blob_score", "severity_label").
  ↪toPandas()

# Plot
plt.figure(figsize=(10, 6))
sns.scatterplot(data=plot_df, x="vader_score", y="blob_score",
                hue="severity_label", palette="viridis")
plt.title('Sentiment Scores by Severity Level')
plt.xlabel('VADER Sentiment Score')
plt.ylabel('TextBlob Sentiment Score')
plt.legend(title='Severity Level', labels=['Low', 'Medium', 'High'])
plt.show()
```



[38]:
```
comparison_df = predictions.select("severity_label", "prediction").toPandas()

plt.figure(figsize=(10, 6))
sns.countplot(data=comparison_df.melt(), x='value', hue='variable',
              palette={'severity_label': '#1f77b4', 'prediction': '#ff7f0e'})
plt.title('Comparison of Actual vs Predicted Severity')
plt.xlabel('Severity Level')
plt.ylabel('Count')
plt.legend(title='Type', labels=['Actual', 'Predicted'])
plt.xticks([0, 1, 2], ['Low', 'Medium', 'High'])
plt.show()
```



[41]:
```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, udf, struct, when
from pyspark.sql.types import StringType, FloatType
import pandas as pd
import numpy as np

# Initialize Spark (if not already done)
spark = SparkSession.builder.appName("DisasterSeverity").getOrCreate()

# 1. Enhanced Severity Scoring Function
def calculate_severity_score(row):
    """Calculate comprehensive severity score combining multiple factors"""
    # Base weights (adjust based on your domain knowledge)
    weights = {
        'keyword_urgency': 0.35,
        'sentiment': 0.3,
        'distress_terms': 0.25,
        'repetition': 0.1
    }

    # 1. Keyword urgency analysis
    urgent_keywords = ['emergency', 'urgent', 'critical', 'immediate', 'help']
    keyword_score = sum(1 for kw in urgent_keywords if kw in row['text'].
 ↪lower()) / len(urgent_keywords)
```

```
    # 2. Sentiment analysis (using existing vader_score)
    sentiment_score = 1 - row['vader_score']  # More negative = more severe

    # 3. Distress term detection
    distress_terms = ['sos', 'rescue', 'trapped', 'danger', 'evacuate']
    distress_score = sum(1 for term in distress_terms if term in row['text'].
 ↪lower()) / len(distress_terms)

    # 4. Message repetition (if you have user/source info)
    repetition_score = 0  # Could be enhanced with groupBy user/location

    # Calculate composite score
    composite_score = (
        weights['keyword_urgency'] * keyword_score +
        weights['sentiment'] * sentiment_score +
        weights['distress_terms'] * distress_score +
        weights['repetition'] * repetition_score
    )

    return float(composite_score)

# Register UDF
severity_score_udf = udf(calculate_severity_score, FloatType())

# 2. Generate Severity Labels
# Calculate severity scores - FIXED: Added struct import and proper column␣
 ↪reference
df = df.withColumn("severity_score", severity_score_udf(struct([col(c) for c in␣
 ↪df.columns])))

# Define severity thresholds (adjust based on your data distribution)
df = df.withColumn("severity_label",
                   when(col("severity_score") >= 0.75, "critical")
                   .when(col("severity_score") >= 0.45, "severe")
                   .otherwise("minor"))

# Show distribution
print("Severity Label Distribution:")
df.groupBy("severity_label").count().orderBy("count", ascending=False).show()
```

```
25/04/02 19:44:50 WARN SparkSession: Using an existing Spark session; only
runtime SQL configurations will take effect.

Severity Label Distribution:

[Stage 110:>                                                        (0 + 1) / 1]

+--------------+-----+
```

```
|severity_label|count|
+--------------+-----+
|         minor| 5210|
|        severe| 1966|
+--------------+-----+
```

[3]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf, col
from pyspark.sql.types import StringType
import re
import emoji
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer

# Initialize Spark Session
spark = SparkSession.builder.appName("TweetAnalysis").getOrCreate()

# Download necessary NLTK resources
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

# Load dataset from HDFS
#df = spark.read.csv("hdfs://master-node:9000/user/dinesh/tweets_data_sample/
 ↪bda_train.csv",
                     #header=True,
                     #inferSchema=True)

# Handle missing values
df = df.na.fill("", subset=["keyword"])
df = df.na.fill("Unknown", subset=["location"])

# Initialize NLP tools
stop_words = set(stopwords.words('english'))
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# Function to clean text
def clean_text(text):
    if text is None:
        return ""
    text = str(text).lower()
    text = re.sub(r"http\S+|www\S+", "", text)  # Remove URLs
```

```
    text = re.sub(r"@\w+|#\w+", "", text)  # Remove mentions/hashtags
    text = re.sub(r"[^\w\s]", "", text)  # Remove special characters
    text = emoji.replace_emoji(text, replace="")  # Remove emojis
    return text

# Function to preprocess text
def preprocess_text(text):
    if text is None:
        return ""
    words = word_tokenize(text)
    words = [word for word in words if word not in stop_words]
    words = [stemmer.stem(word) for word in words]
    words = [lemmatizer.lemmatize(word) for word in words]
    return " ".join(words)

#  Register UDFs **BEFORE** using them
clean_text_udf = udf(clean_text, StringType())
preprocess_text_udf = udf(preprocess_text, StringType())

#  Apply transformations correctly
df = df.withColumn("cleaned_text",␣
 ↪preprocess_text_udf(clean_text_udf(col("text"))))

# Show results
df.show(truncate=False)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/surendra/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /home/surendra/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /home/surendra/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[Stage 4:>                                                          (0 + 1) / 1]
+---+-------+--------+-----------------------------------------------------------
-----------------------------------------------------------------------------+----
--+-----------------------------------------------------------------------------
-+
|id |keyword|location|text
|target|cleaned_text
|
+---+-------+--------+-----------------------------------------------------------
-----------------------------------------------------------------------------+----
--+-----------------------------------------------------------------------------
-+
|1  |       |Unknown |Our Deeds are the Reason of this #earthquake May ALLAH
Forgive us all                                                                |1
```

```
|deed reason may allah forgiv u                                                  |
|4  |       |Unknown |Forest fire near La Ronge Sask. Canada
|1     |forest fire near la rong sask canada
|
|5  |       |Unknown |All residents asked to 'shelter in place' are being
notified by officers. No other evacuation or shelter in place orders are
expected|1     |resid ask shelter place notifi offic evacu shelter place order
expect         |
|6  |       |Unknown |13,000 people receive #wildfires evacuation orders in
California
|1     |13000 peopl receiv evacu order california
|
|7  |       |Unknown |Just got sent this photo from Ruby #Alaska as smoke from
#wildfires pours into a school                                                |1
|got sent photo rubi smoke pour school                                           |
|8  |       |Unknown |#RockyFire Update => California Hwy. 20 closed in both
directions due to Lake County fire - #CAfire #wildfires                        |1
|updat california hwi 20 close direct due lake counti fire                       |
|10 |       |Unknown |#flood #disaster Heavy rain causes flash flooding of
streets in Manitou, Colorado Springs areas
|1     |heavi rain caus flash flood street manit colorado spring area
|
|13 |       |Unknown |I'm on top of the hill and I can see a fire in the
woods…
|1     |im top hill see fire wood
|
|14 |       |Unknown |There's an emergency evacuation happening now in the
building across the street
|1     |there emerg evacu happen build across street
|
|15 |       |Unknown |I'm afraid that the tornado is coming to our area…
|1     |im afraid tornado come area
|
|16 |       |Unknown |Three people died from the heat wave so far
|1     |three peopl die heat wave far
|
|17 |       |Unknown |Haha South Tampa is getting flooded hah- WAIT A SECOND I
LIVE IN SOUTH TAMPA WHAT AM I GONNA DO WHAT AM I GONNA DO FVCK #flooding    |1
|haha south tampa get flood hah wait second live south tampa gon na gon na fvck|
|18 |       |Unknown |#raining #flooding #Florida #TampaBay #Tampa 18 or 19
days. I've lost count
|1     |18 19 day ive lost count
|
|19 |       |Unknown |#Flood in Bago Myanmar #We arrived Bago
|1     |bago myanmar arriv bago
|
|20 |       |Unknown |Damage to school bus on 80 in multi car crash #BREAKING
|1     |damag school bu 80 multi car crash
```

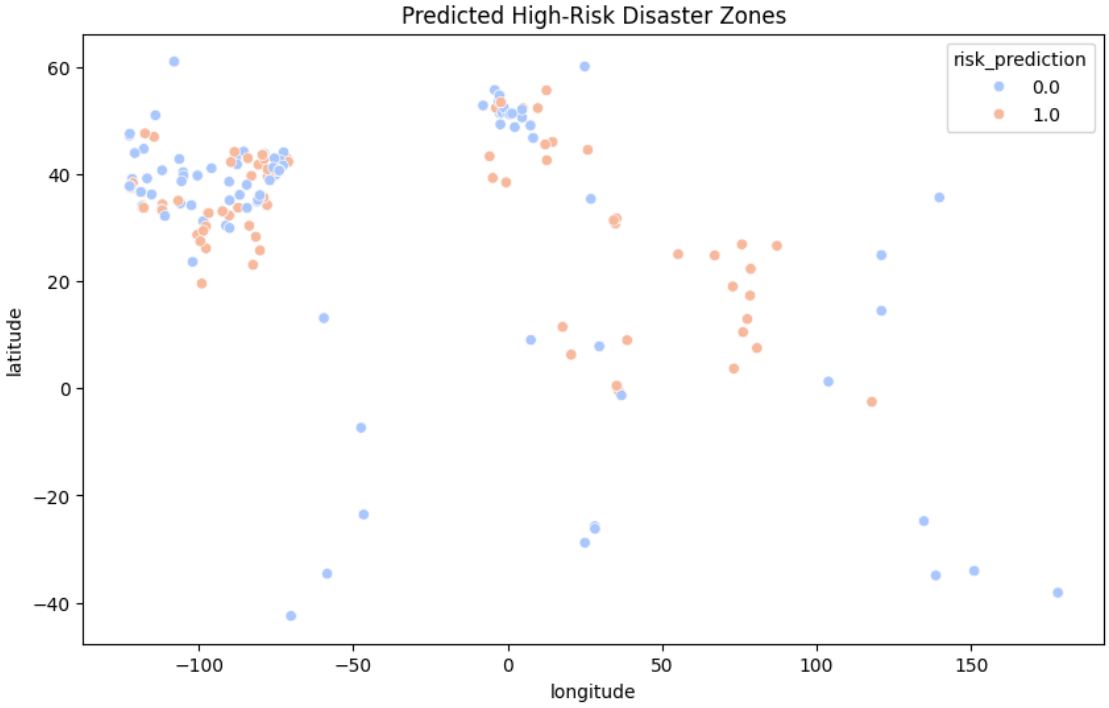
```
|
|23 |       |Unknown |What's up man?
|0     |what man
|
|24 |       |Unknown |I love fruits
|0     |love fruit
|
|25 |       |Unknown |Summer is lovely
|0     |summer love
|
|26 |       |Unknown |My car is so fast
|0     |car fast
|
|28 |       |Unknown |What a goooooooaaaaaal!!!!!!
|0     |goooooooaaaaaal
|
+---+-------+--------+-------------------------------------------------------------------
-------------------------------------------------------------------------+----
--+-------------------------------------------------------------------------
-+
only showing top 20 rows
```

[ ]:

[83]:
```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import pandas as pd
import numpy as np

# Initialize Spark Session
spark = SparkSession.builder.appName("SpatialPrediction").getOrCreate()
# Load dataset
df = spark.read.csv("hdfs://master-node:9000/user/dinesh/tweets_data_sample/
 ↪bda_train.csv",
                    header=True, inferSchema=True)

# Add geocoded coordinates (reusing your geocoding logic)
from opencage.geocoder import OpenCageGeocode
API_KEY = "eeb5797da3e944068c99c92df273961d"  # Replace with your API key
geocoder = OpenCageGeocode(API_KEY)

def get_coordinates(location):
```

```
    try:
        if pd.notna(location) and isinstance(location, str):
            result = geocoder.geocode(location)
            if result and len(result) > 0:
                return result[0]['geometry']['lat'],␣
 ↪result[0]['geometry']['lng']
    except:
        return None, None
    return None, None

# Convert to Pandas for ML
pdf = df.limit(500).toPandas()  # Limiting to 500 rows to avoid API overuse
pdf[['latitude', 'longitude']] = pdf['location'].apply(lambda x: pd.
 ↪Series(get_coordinates(x)))

# Drop rows with missing coordinates or target
pdf = pdf.dropna(subset=['latitude', 'longitude', 'target'])

# Features: latitude, longitude, and target (disaster or not)
X = pdf[['latitude', 'longitude']]
y = pdf['target']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Train Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict and evaluate
y_pred = rf_model.predict(X_test)
#print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
#print(classification_report(y_test, y_pred))

# Predict high-risk zones (example: entire dataset)
pdf['risk_prediction'] = rf_model.predict(X)

# Visualize predictions
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10, 6))
sns.scatterplot(x='longitude', y='latitude', hue='risk_prediction', data=pdf,␣
 ↪palette='coolwarm')
plt.title("Predicted High-Risk Disaster Zones")
plt.show()
```

```
25/04/02 14:11:13 WARN SparkSession: Using an existing Spark session; only
runtime SQL configurations will take effect.
INFO:backoff:Backing off _opencage_request(…) for 0.5s
(requests.exceptions.ConnectionError:
HTTPSConnectionPool(host='api.opencagedata.com', port=443): Max retries exceeded
with url: /geocode/v1/json?q=Birmingham&key=eeb5797da3e944068c99c92df273961d
(Caused by NameResolutionError("<urllib3.connection.HTTPSConnection object at
0x7fa81dc85820>: Failed to resolve 'api.opencagedata.com' ([Errno -3] Temporary
failure in name resolution")))
```



[52]:
```python
# Initialize Spark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("DisasterDashboard").getOrCreate()

# ========== DATA PREPARATION ==========
# Assuming df is your loaded Spark DataFrame with columns:
# ['id', 'keyword', 'location', 'text', 'target', 'length', 'clean_text',␣
 ↪'vader_score', 'blob_score', 'distress_signal', 'severity_score',␣
 ↪'severity_label']

# Add dummy latitude/longitude if not present (for mapping)
if 'lat' not in df.columns or 'lon' not in df.columns:
    from pyspark.sql.functions import rand
```

```
    df = df.withColumn('lat', rand() * 180 - 90)  # Random latitudes between -90 and 90
    df = df.withColumn('lon', rand() * 360 - 180)  # Random longitudes between -180 and 180

# Register temp view for SQL queries
df.createOrReplaceTempView("disaster_data")

# ========== INTERACTIVE WIDGETS ==========
import ipywidgets as widgets
from IPython.display import display, HTML
import pandas as pd

# Create interactive filters
location_options = ['All'] + sorted([row['location'] for row in df.select("location").distinct().collect() if row['location']])
location_filter = widgets.Dropdown(
    options=location_options,
    description='Location:',
    style={'description_width': 'initial'}
)

severity_options = ['All'] + sorted([row['severity_label'] for row in df.select("severity_label").distinct().collect() if row['severity_label']])
severity_filter = widgets.Dropdown(
    options=severity_options,
    description='Severity:',
    style={'description_width': 'initial'}
)

keyword_search = widgets.Text(
    description='Keyword:',
    style={'description_width': 'initial'}
)

# ========== VISUALIZATION FUNCTIONS ==========
import plotly.express as px
import folium
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
import nltk
try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    nltk.download('stopwords')
```

```
# Create output areas
out_map = widgets.Output()
out_stats = widgets.Output()
out_wordcloud = widgets.Output()

def update_dashboard(change):
    # Build SQL query based on filters
    query = "SELECT * FROM disaster_data WHERE 1=1"

    if location_filter.value != 'All':
        query += f" AND location = '{location_filter.value}'"
    if severity_filter.value != 'All':
        query += f" AND severity_label = '{severity_filter.value}'"
    if keyword_search.value:
        query += f" AND (LOWER(text) LIKE LOWER('%{keyword_search.value}%') OR␣
 ↪LOWER(clean_text) LIKE LOWER('%{keyword_search.value}%'))"

    filtered_df = spark.sql(query)

    # Check if data exists
    if filtered_df.count() == 0:
        with out_map:
            display(HTML("<p style='color:red'>No data matching filters</p>"))
        with out_stats:
            display(HTML("<p style='color:red'>No data matching filters</p>"))
        with out_wordcloud:
            display(HTML("<p style='color:red'>No data matching filters</p>"))
        return

    pdf = filtered_df.toPandas()

    # Clear previous outputs
    out_map.clear_output()
    out_stats.clear_output()
    out_wordcloud.clear_output()

    # 1. Update Map Visualization
    with out_map:
        try:
            # Calculate center point
            center_lat = pdf['lat'].mean()
            center_lon = pdf['lon'].mean()

            m = folium.Map(location=[center_lat, center_lon], zoom_start=6)

            # Add markers
            for _, row in pdf.iterrows():
```

```
                folium.CircleMarker(
                    location=[row['lat'], row['lon']],
                    radius=row['severity_score']*10 + 5,  # Scale for visibility
                    color='red' if row['severity_score'] > 0.7 else 'orange' if␣
↪row['severity_score'] > 0.4 else 'green',
                    fill=True,
                    fill_opacity=0.7,
                    popup=folium.Popup(f"""
                        <b>{row.get('keyword', 'N/A')}</b><br>
                        Location: {row.get('location', 'N/A')}<br>
                        Severity: {row.get('severity_label', 'N/A')}<br>
                        Score: {row.get('severity_score', 0):.2f}<br>
                        <hr>
                        {row.get('text', 'No text')[:100]}...
                        """, max_width=300)
                ).add_to(m)

            display(m)
        except Exception as e:
            display(HTML(f"<p style='color:red'>Error generating map: {str(e)}</
↪p>"))

    # 2. Update Statistical Visualizations
    with out_stats:
        try:
            # Sentiment Distribution
            fig1 = px.histogram(pdf, x='vader_score', color='severity_label',
                                title='Sentiment Distribution by Severity',
                                nbins=20,
                                labels={'vader_score': 'Sentiment Score',␣
↪'severity_label': 'Severity'})

            # Severity Distribution
            severity_counts = pdf['severity_label'].value_counts().reset_index()
            severity_counts.columns = ['severity_label', 'count']
            fig2 = px.pie(severity_counts, values='count',␣
↪names='severity_label',
                          title='Severity Level Distribution')

            # Display plots side by side
            display(widgets.HBox([fig1, fig2]))
        except Exception as e:
            display(HTML(f"<p style='color:red'>Error generating stats:␣
↪{str(e)}</p>"))

    # 3. Update Word Cloud
    with out_wordcloud:
```

```
        try:
            if 'clean_text' in pdf.columns:
                stop_words = set(stopwords.words('english'))
                custom_stopwords = ['http', 'https', 'com', 'www']  # Add␣
 ↪domain-specific stopwords
                text = ' '.join(pdf['clean_text'].dropna().astype(str))

                if text.strip():
                    wc = WordCloud(width=800, height=400,
                                   background_color='white',
                                   stopwords=stop_words.union(custom_stopwords),
                                   collocations=False)
                    wc.generate(text)

                    plt.figure(figsize=(12, 6))
                    plt.imshow(wc, interpolation='bilinear')
                    plt.axis("off")
                    plt.title("Most Frequent Words in Filtered Reports")
                    plt.show()
                else:
                    display(HTML("<p>No text data available for word cloud</
 ↪p>"))
            else:
                display(HTML("<p>No clean_text column available</p>"))
        except Exception as e:
            display(HTML(f"<p style='color:red'>Error generating word cloud:␣
 ↪{str(e)}</p>"))

# ========== DASHBOARD LAYOUT ==========
# Set up observer for filters
for widget in [location_filter, severity_filter, keyword_search]:
    widget.observe(update_dashboard, names='value')

# Display the dashboard
display(HTML("""
<style>
    .widget-label { min-width: 120px !important; }
    .widget-dropdown { min-width: 200px !important; }
    .widget-text { min-width: 300px !important; }
</style>
<h1 style='text-align:center; color:#e74c3c;'> Disaster Monitoring Dashboard</
 ↪h1>
"""))

# Create filter row
filter_row = widgets.HBox([
    widgets.VBox([location_filter]),
```

```
    widgets.VBox([severity_filter]),
    widgets.VBox([keyword_search])
])

# Main layout
dashboard = widgets.VBox([
    filter_row,
    widgets.HTML("<hr>"),
    widgets.HTML("<h3 style='text-align:center'>Geographical Distribution</
 ↪h3>"),
    out_map,
    widgets.HTML("<hr>"),
    widgets.HTML("<h3 style='text-align:center'>Statistical Analysis</h3>"),
    out_stats,
    widgets.HTML("<hr>"),
    widgets.HTML("<h3 style='text-align:center'>Text Analysis</h3>"),
    out_wordcloud
])

display(dashboard)

# Initial render
update_dashboard(None)
```

```
<IPython.core.display.HTML object>
```

```
VBox(children=(HBox(children=(VBox(children=(Dropdown(description='Location:',␣
 ↪options=('All', ' ', '  ', '   …
```

[ ]:
```
# Initialize Spark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("DisasterDashboard").getOrCreate()

# ========== DATA PREPARATION ==========
# Assuming df is your loaded Spark DataFrame with columns:
# ['id', 'keyword', 'location', 'text', 'target', 'length', 'clean_text',␣
 ↪'vader_score', 'blob_score', 'distress_signal', 'severity_score',␣
 ↪'severity_label']

# Add dummy latitude/longitude if not present (for mapping)
if 'lat' not in df.columns or 'lon' not in df.columns:
    from pyspark.sql.functions import rand
    df = df.withColumn('lat', rand() * 180 - 90)  # Random latitudes between␣
 ↪-90 and 90
```

```
    df = df.withColumn('lon', rand() * 360 - 180)  # Random longitudes between -180 and 180

# Register temp view for SQL queries
df.createOrReplaceTempView("disaster_data")

# ========== INTERACTIVE WIDGETS ==========
import ipywidgets as widgets
from IPython.display import display, HTML
import pandas as pd

# Create interactive filters
location_options = ['All'] + sorted([row['location'] for row in df.select("location").distinct().collect() if row['location']])
location_filter = widgets.Dropdown(
    options=location_options,
    description='Location:',
    style={'description_width': 'initial'}
)

severity_options = ['All'] + sorted([row['severity_label'] for row in df.select("severity_label").distinct().collect() if row['severity_label']])
severity_filter = widgets.Dropdown(
    options=severity_options,
    description='Severity:',
    style={'description_width': 'initial'}
)

keyword_search = widgets.Text(
    description='Keyword:',
    style={'description_width': 'initial'}
)

# ========== VISUALIZATION FUNCTIONS ==========
import plotly.express as px
import folium
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
import nltk
try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    nltk.download('stopwords')

# Create output areas
out_map = widgets.Output()
```

```
out_stats = widgets.Output()
out_wordcloud = widgets.Output()

def update_dashboard(change):
    # Build SQL query based on filters
    query = "SELECT * FROM disaster_data WHERE 1=1"

    if location_filter.value != 'All':
        query += f" AND location = '{location_filter.value}'"
    if severity_filter.value != 'All':
        query += f" AND severity_label = '{severity_filter.value}'"
    if keyword_search.value:
        query += f" AND (LOWER(text) LIKE LOWER('%{keyword_search.value}%') OR␣
 ↪LOWER(clean_text) LIKE LOWER('%{keyword_search.value}%'))"

    filtered_df = spark.sql(query)

    # Check if data exists
    if filtered_df.count() == 0:
        with out_map:
            display(HTML("<p style='color:red'>No data matching filters</p>"))
        with out_stats:
            display(HTML("<p style='color:red'>No data matching filters</p>"))
        with out_wordcloud:
            display(HTML("<p style='color:red'>No data matching filters</p>"))
        return

    pdf = filtered_df.toPandas()

    # Clear previous outputs
    out_map.clear_output()
    out_stats.clear_output()
    out_wordcloud.clear_output()

    # 1. Update Map Visualization
    with out_map:
        try:
            # Calculate center point
            center_lat = pdf['lat'].mean()
            center_lon = pdf['lon'].mean()

            m = folium.Map(location=[center_lat, center_lon], zoom_start=6)

            # Add markers
            for _, row in pdf.iterrows():
                folium.CircleMarker(
                    location=[row['lat'], row['lon']],
```

```
                    radius=row['severity_score']*10 + 5,  # Scale for visibility
                    color='red' if row['severity_score'] > 0.7 else 'orange' if␣
↪row['severity_score'] > 0.4 else 'green',
                    fill=True,
                    fill_opacity=0.7,
                    popup=folium.Popup(f"""
                        <b>{row.get('keyword', 'N/A')}</b><br>
                        Location: {row.get('location', 'N/A')}<br>
                        Severity: {row.get('severity_label', 'N/A')}<br>
                        Score: {row.get('severity_score', 0):.2f}<br>
                        <hr>
                        {row.get('text', 'No text')[:100]}...
                        """, max_width=300)
                ).add_to(m)

            display(m)
        except Exception as e:
            display(HTML(f"<p style='color:red'>Error generating map: {str(e)}</
↪p>"))

    # 2. Update Statistical Visualizations
    with out_stats:
        try:
            # Sentiment Distribution
            fig1 = px.histogram(pdf, x='vader_score', color='severity_label',
                                title='Sentiment Distribution by Severity',
                                nbins=20,
                                labels={'vader_score': 'Sentiment Score',␣
↪'severity_label': 'Severity'})

            # Severity Distribution
            severity_counts = pdf['severity_label'].value_counts().reset_index()
            severity_counts.columns = ['severity_label', 'count']
            fig2 = px.pie(severity_counts, values='count',␣
↪names='severity_label',
                          title='Severity Level Distribution')

            # Display plots side by side
            display(widgets.HBox([fig1, fig2]))
        except Exception as e:
            display(HTML(f"<p style='color:red'>Error generating stats:␣
↪{str(e)}</p>"))

    # 3. Update Word Cloud
    with out_wordcloud:
        try:
            if 'clean_text' in pdf.columns:
```

```
                stop_words = set(stopwords.words('english'))
                custom_stopwords = ['http', 'https', 'com', 'www']  # Add␣
 ↪domain-specific stopwords
                text = ' '.join(pdf['clean_text'].dropna().astype(str))

                if text.strip():
                    wc = WordCloud(width=800, height=400,
                                   background_color='white',
                                   stopwords=stop_words.union(custom_stopwords),
                                   collocations=False)
                    wc.generate(text)

                    plt.figure(figsize=(12, 6))
                    plt.imshow(wc, interpolation='bilinear')
                    plt.axis("off")
                    plt.title("Most Frequent Words in Filtered Reports")
                    plt.show()
                else:
                    display(HTML("<p>No text data available for word cloud</
 ↪p>"))
            else:
                display(HTML("<p>No clean_text column available</p>"))
        except Exception as e:
            display(HTML(f"<p style='color:red'>Error generating word cloud:␣
 ↪{str(e)}</p>"))

# ========== DASHBOARD LAYOUT ==========
# Set up observer for filters
for widget in [location_filter, severity_filter, keyword_search]:
    widget.observe(update_dashboard, names='value')

# Display the dashboard
display(HTML("""
<style>
    .widget-label { min-width: 120px !important; }
    .widget-dropdown { min-width: 200px !important; }
    .widget-text { min-width: 300px !important; }
</style>
<h1 style='text-align:center; color:#e74c3c;'> Disaster Monitoring Dashboard</
 ↪h1>
"""))

# Create filter row
filter_row = widgets.HBox([
    widgets.VBox([location_filter]),
    widgets.VBox([severity_filter]),
    widgets.VBox([keyword_search])
```

```
])

# Main layout
dashboard = widgets.VBox([
    filter_row,
    widgets.HTML("<hr>"),
    widgets.HTML("<h3 style='text-align:center'>Geographical Distribution</
 ↪h3>"),
    out_map,
    widgets.HTML("<hr>"),
    widgets.HTML("<h3 style='text-align:center'>Statistical Analysis</h3>"),
    out_stats,
    widgets.HTML("<hr>"),
    widgets.HTML("<h3 style='text-align:center'>Text Analysis</h3>"),
    out_wordcloud
])

display(dashboard)

# Initial render
update_dashboard(None)
```

```
<IPython.core.display.HTML object>
```

```
VBox(children=(HBox(children=(VBox(children=(Dropdown(description='Location:',␣
 ↪options=('All', ' ', '  ', '   …
```

```
[Stage 218:>                                                        (0 + 1) / 1]
```

[ ]: