NFleischhauer_Task2

November 22, 2025

1 Task 2: Predictors of Human Harm - Aggregation Job

Student: Nicholas Fleischhauer
Date: November 23, 2025

1.1 Research Question

Which factors are the strongest predictors of human harm? Can we determine if 'human' factors (location) are more predictive than 'storm' factors (EVENT_TYPE, MAGNITUDE_TYPE)?

1.2 Summarization/Aggregation Job Overview

This notebook performs PySpark aggregation operations on the NOAA Storm Events dataset to:
1. **Aggregate human harm** (injuries + deaths) by storm factors (EVENT_TYPE, MAGNITUDE_TYPE) 2. **Aggregate human harm** by location factors (STATE, CZ_NAME)
3. **Count event frequency** per location as a population density proxy 4. **Analyze combined factors** (EVENT_TYPE × STATE) to identify patterns

Dataset: NOAA Storm Events (2020-2025 subset, ~371K rows, 51 columns)
Operations: GroupBy aggregations using PySpark RDD transformations (map, filter, reduceByKey)

[3]:
```
from pyspark.sql import SparkSession
import pyspark
from pyspark import SparkContext
import math
```

[ ]:
```
# Create SparkSession for CSV reading, then get SparkContext for RDD operations
# Configure for GCS access (uncomment auth configs after I get the service account key)

# Base configuration
spark_builder = SparkSession.builder \
    .appName("HumanHarmAnalysis") \
    .config("spark.jars", "/opt/spark/jars/gcs-connector-hadoop3-2.2.11.jar")

# TODO:
# Uncomment these lines when using the gcs-key.json in the project root:
# spark_builder = spark_builder \
```

```
#     .config("spark.hadoop.google.cloud.auth.service.account.enable", "true") \
#     .config("spark.hadoop.google.cloud.auth.service.account.json.keyfile", "/
 ↪home/sparkdev/app/gcs-key.json")

spark = spark_builder.getOrCreate()
sc = spark.sparkContext

# Silence verbose Spark logs - only show warnings and errors
sc.setLogLevel("WARN")
```

[ ]:
```
# Load data using Spark's CSV reader (handles quotes, escaping properly)
# Then convert to RDD for RDD operations

# Local subset file (for testing)
csv_path = "/home/sparkdev/app/Task2/storm_g2020.csv"

# TODO: Use this after I get the service account key
# Full dataset from GCS (uncomment when GCS is configured)
# csv_path = "gs://msds-694-cohort-14-group12/storm_data.csv"


# Read CSV with proper handling of headers, quotes, and escaping
df = spark.read \
    .option("header", "true") \
    .option("inferSchema", "true") \
    .option("quote", "\"") \
    .option("escape", "\"") \
    .option("multiLine", "true") \
    .csv(csv_path)

# Convert DataFrame to RDD of Row objects
rdd = df.rdd

print(f"RDD loaded: {rdd.count()} rows")
print(f"Number of partitions: {rdd.getNumPartitions()}")
```

```
[Stage 8:>                                                          (0 + 1) / 1]

RDD loaded: 371544 rows
Number of partitions: 1
```

[10]:
```
# Preview the data - RDD of Row objects
print("First few rows:")
for row in rdd.take(3):
    print(row)
```

```python
# Get column names from DataFrame for reference
print(f"\nColumn names ({len(df.columns)} total):")
print(df.columns)
```

```
First few rows:

[Stage 9:>                                                          (0 + 1) / 1]

Row(BEGIN_YEARMONTH=202006, BEGIN_DAY=24, BEGIN_TIME=1620, END_YEARMONTH=202006,
END_DAY=24, END_TIME=1620, EPISODE_ID=149684.0, EVENT_ID=902190,
STATE='GEORGIA', STATE_FIPS=13.0, YEAR=2020, MONTH_NAME='June',
EVENT_TYPE='Thunderstorm Wind', CZ_TYPE='C', CZ_FIPS=321, CZ_NAME='WORTH',
WFO='TAE', BEGIN_DATE_TIME='24-JUN-20 16:20:00', CZ_TIMEZONE='EST-5',
END_DATE_TIME='24-JUN-20 16:20:00', INJURIES_DIRECT=0, INJURIES_INDIRECT=0,
DEATHS_DIRECT=0, DEATHS_INDIRECT=0, DAMAGE_PROPERTY='0.00K',
DAMAGE_CROPS='0.00K', SOURCE='911 Call Center', MAGNITUDE=50.0,
MAGNITUDE_TYPE='EG', FLOOD_CAUSE=None, CATEGORY=None, TOR_F_SCALE=None,
TOR_LENGTH=None, TOR_WIDTH=None, TOR_OTHER_WFO=None, TOR_OTHER_CZ_STATE=None,
TOR_OTHER_CZ_FIPS=None, TOR_OTHER_CZ_NAME=None, BEGIN_RANGE=1.0,
BEGIN_AZIMUTH='W', BEGIN_LOCATION='DOLES', END_RANGE=1.0, END_AZIMUTH='W',
END_LOCATION='DOLES', BEGIN_LAT=31.7, BEGIN_LON=-83.89, END_LAT=31.7,
END_LON=-83.89, EPISODE_NARRATIVE='As is typical during summer, scattered
afternoon thunderstorms produced a few instances of damaging winds.',
EVENT_NARRATIVE='A power line was blown down on Highway 32W.  Hail was also
noted, but the size was unknown.', DATA_SOURCE='CSV')
Row(BEGIN_YEARMONTH=202006, BEGIN_DAY=20, BEGIN_TIME=1930, END_YEARMONTH=202006,
END_DAY=20, END_TIME=1930, EPISODE_ID=149048.0, EVENT_ID=898391, STATE='KANSAS',
STATE_FIPS=20.0, YEAR=2020, MONTH_NAME='June', EVENT_TYPE='Hail', CZ_TYPE='C',
CZ_FIPS=137, CZ_NAME='NORTON', WFO='GLD', BEGIN_DATE_TIME='20-JUN-20 19:30:00',
CZ_TIMEZONE='CST-6', END_DATE_TIME='20-JUN-20 19:30:00', INJURIES_DIRECT=0,
INJURIES_INDIRECT=0, DEATHS_DIRECT=0, DEATHS_INDIRECT=0, DAMAGE_PROPERTY=None,
DAMAGE_CROPS=None, SOURCE='Public', MAGNITUDE=1.0, MAGNITUDE_TYPE=None,
FLOOD_CAUSE=None, CATEGORY=None, TOR_F_SCALE=None, TOR_LENGTH=None,
TOR_WIDTH=None, TOR_OTHER_WFO=None, TOR_OTHER_CZ_STATE=None,
TOR_OTHER_CZ_FIPS=None, TOR_OTHER_CZ_NAME=None, BEGIN_RANGE=8.0,
BEGIN_AZIMUTH='SE', BEGIN_LOCATION='CALVERT', END_RANGE=8.0, END_AZIMUTH='SE',
END_LOCATION='CALVERT', BEGIN_LAT=39.7571, BEGIN_LON=-99.6684, END_LAT=39.7571,
END_LON=-99.6684, EPISODE_NARRATIVE='Supercells in small clusters formed during
the afternoon to early evening hours. The storms dropped large hail up to tennis
ball in size across northern Norton County.', EVENT_NARRATIVE='Penny to quarter
size hail reported and ongoing at time of the report.', DATA_SOURCE='CSV')
Row(BEGIN_YEARMONTH=202006, BEGIN_DAY=3, BEGIN_TIME=1550, END_YEARMONTH=202006,
END_DAY=3, END_TIME=1550, EPISODE_ID=149149.0, EVENT_ID=899120, STATE='KANSAS',
STATE_FIPS=20.0, YEAR=2020, MONTH_NAME='June', EVENT_TYPE='Hail', CZ_TYPE='C',
CZ_FIPS=23, CZ_NAME='CHEYENNE', WFO='GLD', BEGIN_DATE_TIME='03-JUN-20 15:50:00',
CZ_TIMEZONE='CST-6', END_DATE_TIME='03-JUN-20 15:50:00', INJURIES_DIRECT=0,
INJURIES_INDIRECT=0, DEATHS_DIRECT=0, DEATHS_INDIRECT=0, DAMAGE_PROPERTY=None,
DAMAGE_CROPS=None, SOURCE='Trained Spotter', MAGNITUDE=0.75,
MAGNITUDE_TYPE=None, FLOOD_CAUSE=None, CATEGORY=None, TOR_F_SCALE=None,
```

```
TOR_LENGTH=None, TOR_WIDTH=None, TOR_OTHER_WFO=None, TOR_OTHER_CZ_STATE=None,
TOR_OTHER_CZ_FIPS=None, TOR_OTHER_CZ_NAME=None, BEGIN_RANGE=14.0,
BEGIN_AZIMUTH='NW', BEGIN_LOCATION='ST FRANCIS', END_RANGE=14.0,
END_AZIMUTH='NW', END_LOCATION='ST FRANCIS', BEGIN_LAT=39.9137,
BEGIN_LON=-101.9753, END_LAT=39.9137, END_LON=-101.9753,
EPISODE_NARRATIVE='Thunderstorms formed in eastern Colorado during the afternoon
moving northeast behind an outflow boundary. As the storms moved northeast into
Cheyenne County, they produced hail up to penny in size.', EVENT_NARRATIVE='Dime
to penny sized hail reported at the location.', DATA_SOURCE='CSV')

Column names (51 total):
['BEGIN_YEARMONTH', 'BEGIN_DAY', 'BEGIN_TIME', 'END_YEARMONTH', 'END_DAY',
'END_TIME', 'EPISODE_ID', 'EVENT_ID', 'STATE', 'STATE_FIPS', 'YEAR',
'MONTH_NAME', 'EVENT_TYPE', 'CZ_TYPE', 'CZ_FIPS', 'CZ_NAME', 'WFO',
'BEGIN_DATE_TIME', 'CZ_TIMEZONE', 'END_DATE_TIME', 'INJURIES_DIRECT',
'INJURIES_INDIRECT', 'DEATHS_DIRECT', 'DEATHS_INDIRECT', 'DAMAGE_PROPERTY',
'DAMAGE_CROPS', 'SOURCE', 'MAGNITUDE', 'MAGNITUDE_TYPE', 'FLOOD_CAUSE',
'CATEGORY', 'TOR_F_SCALE', 'TOR_LENGTH', 'TOR_WIDTH', 'TOR_OTHER_WFO',
'TOR_OTHER_CZ_STATE', 'TOR_OTHER_CZ_FIPS', 'TOR_OTHER_CZ_NAME', 'BEGIN_RANGE',
'BEGIN_AZIMUTH', 'BEGIN_LOCATION', 'END_RANGE', 'END_AZIMUTH', 'END_LOCATION',
'BEGIN_LAT', 'BEGIN_LON', 'END_LAT', 'END_LON', 'EPISODE_NARRATIVE',
'EVENT_NARRATIVE', 'DATA_SOURCE']
```

1.3 Key Columns (access by name using row['COLUMN_NAME'])

- **STATE** - State name
- **EVENT_TYPE** - Type of storm event
- **CZ_NAME** - County/Zone name
- **INJURIES_DIRECT** - Direct injuries
- **INJURIES_INDIRECT** - Indirect injuries
- **DEATHS_DIRECT** - Direct deaths
- **DEATHS_INDIRECT** - Indirect deaths
- **MAGNITUDE** - Storm magnitude
- **MAGNITUDE_TYPE** - Type of magnitude measurement

Note: Since we read CSV as DataFrame then converted to RDD, rows are Row objects. Access columns by name: `row['STATE']` or `row.STATE`

[12]:
```python
def safe_int(x):
    """Safely convert to int, return 0 if None/null"""
    if x is None:
        return 0
    try:
        return int(float(x))
    except (ValueError, TypeError):
        return 0
```

```
def calculate_total_harm(row):
    """Calculate total human harm: injuries + deaths (direct + indirect)"""
    # Row objects support dictionary-style access: row['COLUMN'] returns None␣
 ↪if missing/null
    injuries_direct = safe_int(row['INJURIES_DIRECT'])
    injuries_indirect = safe_int(row['INJURIES_INDIRECT'])
    deaths_direct = safe_int(row['DEATHS_DIRECT'])
    deaths_indirect = safe_int(row['DEATHS_INDIRECT'])
    return injuries_direct + injuries_indirect + deaths_direct + deaths_indirect

# Create RDD with total harm calculated
rdd_with_harm = rdd.map(lambda row: (row, calculate_total_harm(row)))

# Filter to only rows with harm > 0
rdd_harm = rdd_with_harm.filter(lambda x: x[1] > 0)

# Calculate metrics
events_with_harm = rdd_harm.count()
total_harm_sum = int(rdd_harm.map(lambda x: x[1]).sum())

# Display results with clean formatting
print("="*60)
print("DATASET OVERVIEW: Human Harm Analysis")
print("="*60)
print(f"Total events with harm:        {events_with_harm:,} events")
print(f"Total people harmed:           {total_harm_sum:,} people")
print(f"Average harm per event:        {total_harm_sum/events_with_harm:.2f}␣
 ↪people/event")
print("="*60)
```

```
[Stage 13:>                                                         (0 + 1) / 1]
```

```
============================================================
DATASET OVERVIEW: Human Harm Analysis
============================================================
Total events with harm:        4,604 events
Total people harmed:           17,541 people
Average harm per event:        3.81 people/event
============================================================
```

1.4 Analysis 1: Human Harm by Storm Factors

[18]:
```
# Aggregate total harm by EVENT_TYPE
harm_by_event = (
    rdd_harm
```

```
    .map(lambda x: (x[0]['EVENT_TYPE'] if x[0]['EVENT_TYPE'] else 'UNKNOWN',␣
 ↪x[1]))
    .filter(lambda x: x[0] and x[0] != "")  # Only events with valid type
    .mapValues(lambda v: (v, 1))  # (harm, count)
    .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))  # Sum harm and count
    .mapValues(lambda x: (x[0], x[1], x[0] / x[1] if x[1] > 0 else 0))  #␣
 ↪(total_harm, count, avg_harm)
)

# Sort by total harm descending
harm_by_event_sorted = harm_by_event.sortBy(lambda x: x[1][0], ascending=False)

# Total is the total harm for the event type, so if there are 2 hurricanes with␣
 ↪100 deaths each, this will be 200
# Count is the number of events with harm, so if there are 2 hurricanes with␣
 ↪100 deaths each, this will be 2
# Avg is the average harm per event, soif there are 2 hurricanes with 100␣
 ↪deaths each, this will be 100
print("Top 10 Event Types by Total Human Harm:")
for event_type, (total_harm, count, avg_harm) in harm_by_event_sorted.take(10):
    print(f"{event_type}: Total={int(total_harm)}, Count={count}, Avg={avg_harm:
 ↪.2f}")
```

```
Top 10 Event Types by Total Human Harm:
[Stage 24:>                                                         (0 + 1) / 1]
Tornado: Total=4146, Count=515, Avg=8.05
Excessive Heat: Total=3561, Count=332, Avg=10.73
Heat: Total=1646, Count=569, Avg=2.89
Thunderstorm Wind: Total=1193, Count=577, Avg=2.07
Winter Weather: Total=931, Count=303, Avg=3.07
Wildfire: Total=742, Count=131, Avg=5.66
Rip Current: Total=590, Count=382, Avg=1.54
Flash Flood: Total=549, Count=256, Avg=2.14
Lightning: Total=402, Count=229, Avg=1.76
Winter Storm: Total=393, Count=139, Avg=2.83
```

[19]:
```
# Aggregate total harm by MAGNITUDE_TYPE
harm_by_magnitude = (
    rdd_harm
    .map(lambda x: (x[0]['MAGNITUDE_TYPE'] if x[0]['MAGNITUDE_TYPE'] else␣
 ↪'NONE', x[1]))
    .filter(lambda x: x[0] and x[0] != "")
    .mapValues(lambda v: (v, 1))
    .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))
    .mapValues(lambda x: (x[0], x[1], x[0] / x[1] if x[1] > 0 else 0))
```

```
)

harm_by_magnitude_sorted = harm_by_magnitude.sortBy(lambda x: x[1][0],␣
 ↪ascending=False)

print("\nHarm by Magnitude Type:")
for mag_type, (total_harm, count, avg_harm) in harm_by_magnitude_sorted.
 ↪collect():
    print(f"{mag_type}: Total={int(total_harm)}, Count={count}, Avg={avg_harm:.
 ↪2f}")
```

```
Harm by Magnitude Type:
[Stage 26:>                                                         (0 + 1) / 1]
NONE: Total=15906, Count=3766, Avg=4.22
EG: Total=1418, Count=738, Avg=1.92
MG: Total=208, Count=98, Avg=2.12
ES: Total=9, Count=2, Avg=4.50
```

1.4.1 Key Findings: Storm Factors and Human Harm

MAGNITUDE_TYPE Analysis:

The MAGNITUDE_TYPE field records the type of magnitude measurement for storm events. The categories include: - **EG** (Estimated Gust) - estimated wind speed in knots - **MG** (Measured Gust) - measured wind speed in knots - **ES** (Estimated Sustained) - estimated sustained wind speed - **NONE** - no magnitude measurement recorded

Critical: Events with **NONE** as the magnitude type account for the vast majority of human harm (15,906 people across 3,766 events, averaging 4.22 people per event). This significantly outpaces events with recorded wind measurements (EG averages only 1.92 people/event).

This suggests that **non-wind/hail storm events** — such as floods, tornadoes without wind speed data, extreme temperatures, and other weather phenomena that don't record magnitude — are actually **more dangerous to humans** than events with measurable wind speeds or hail sizes. This finding supports the hypothesis that different storm factors may predict harm differently, and that magnitude measurements alone are insufficient predictors of human impact.

For our Random Forest modeling in later phases, this indicates that: 1. EVENT_TYPE (the type of storm) may be a stronger predictor than MAGNITUDE_TYPE 2. Location factors (STATE, CZ_NAME) could be even more important if they correlate with severe non-wind events 3. We should engineer features that capture the severity of events beyond just wind/hail measurements

1.5 Analysis 2: Human Harm by Location Factors

[20]:
```
# Aggregate total harm by STATE
harm_by_state = (
    rdd_harm
    .map(lambda x: (x[0]['STATE'] if x[0]['STATE'] else 'UNKNOWN', x[1]))
    .filter(lambda x: x[0] and x[0] != "")
    .mapValues(lambda v: (v, 1))
    .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))
    .mapValues(lambda x: (x[0], x[1], x[0] / x[1] if x[1] > 0 else 0))
)

harm_by_state_sorted = harm_by_state.sortBy(lambda x: x[1][0], ascending=False)

print("Top 10 States by Total Human Harm:")
for state, (total_harm, count, avg_harm) in harm_by_state_sorted.take(10):
    print(f"{state}: Total={int(total_harm)}, Count={count}, Avg={avg_harm:.
 ↪2f}")
```

```
Top 10 States by Total Human Harm:
[Stage 28:>                                                         (0 + 1) / 1]
TEXAS: Total=2915, Count=316, Avg=9.22
ARIZONA: Total=2181, Count=709, Avg=3.08
CALIFORNIA: Total=967, Count=303, Avg=3.19
KENTUCKY: Total=967, Count=113, Avg=8.56
MISSOURI: Total=888, Count=172, Avg=5.16
TENNESSEE: Total=803, Count=130, Avg=6.18
MISSISSIPPI: Total=617, Count=110, Avg=5.61
OKLAHOMA: Total=591, Count=95, Avg=6.22
FLORIDA: Total=573, Count=259, Avg=2.21
GEORGIA: Total=435, Count=97, Avg=4.48
```

1.6 Note About Location

It will probably be important to take into account population count and population densities per state when using this kind of analysis. Its possible that they can skew the statistics so that it appears like one state may have higher harm, simply because it has more people or more population density in high risk regions.

TODO: Adjust for these concerns in a future exploration.

[23]:
```
# Aggregate total harm by STATE and CZ_NAME (County/Zone)
harm_by_cz = (
    rdd_harm
    .map(lambda x: ((
        x[0]['STATE'] if x[0]['STATE'] else 'UNKNOWN',
```

```
        x[0]['CZ_NAME'] if x[0]['CZ_NAME'] else 'UNKNOWN'
    ), x[1]))
    .filter(lambda x: x[0][0] and x[0][0] != "" and x[0][1] and x[0][1] != "")
    .mapValues(lambda v: (v, 1))
    .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))
    .mapValues(lambda x: (x[0], x[1], x[0] / x[1] if x[1] > 0 else 0))
)

harm_by_cz_sorted = harm_by_cz.sortBy(lambda x: x[1][0], ascending=False)

print("\nTop 10 Counties/Zones by Total Human Harm:")
for (state, cz), (total_harm, count, avg_harm) in harm_by_cz_sorted.take(10):
    print(f"{state}, {cz}: Total={int(total_harm)}, Count={count},␣
 ↪Avg={avg_harm:.2f}")
```

```
Top 10 Counties/Zones by Total Human Harm:
[Stage 34:>                                                         (0 + 1) / 1]
TEXAS, DALLAS: Total=1427, Count=14, Avg=101.93
ARIZONA, CENTRAL PHOENIX: Total=1027, Count=188, Avg=5.46
MISSOURI, DOUGLAS: Total=352, Count=2, Avg=176.00
TEXAS, DENTON: Total=315, Count=23, Avg=13.70
OKLAHOMA, TULSA: Total=309, Count=23, Avg=13.43
NEVADA, LAS VEGAS VALLEY: Total=271, Count=58, Avg=4.67
KENTUCKY, GRAVES: Total=239, Count=5, Avg=47.80
KENTUCKY, HOPKINS: Total=234, Count=3, Avg=78.00
TENNESSEE, DAVIDSON: Total=226, Count=12, Avg=18.83
ARIZONA, TUCSON METRO AREA: Total=225, Count=84, Avg=2.68
```

1.7 Analysis 3: Event Frequency by Location (Proxy for Population Density)

This will partially address the concerns in analysis 2.

[25]:
```
# Count events per CZ_NAME - more events = likely more populated area
# This will be used as a proxy for population density in later modeling
event_count_by_cz = (
    rdd
    .map(lambda row: ((
        row['STATE'] if row['STATE'] else 'UNKNOWN',
        row['CZ_NAME'] if row['CZ_NAME'] else 'UNKNOWN'
    ), 1))
    .filter(lambda x: x[0][0] and x[0][0] != "" and x[0][1] and x[0][1] != "")
    .reduceByKey(lambda a, b: a + b)
)
```

```
event_count_sorted = event_count_by_cz.sortBy(lambda x: x[1], ascending=False)

print("\nTop 10 Counties/Zones by Event Count (Population Proxy):")
for (state, cz), count in event_count_sorted.take(10):
    print(f"{state}, {cz}: {count} events")
```

```
Top 10 Counties/Zones by Event Count (Population Proxy):
[Stage 38:>                                                         (0 + 1) / 1]
ILLINOIS, COOK: 797 events
ATLANTIC SOUTH, VOLUSIA-BREVARD COUNTY LINE TO SEBASTIAN INLET 0-20NM: 754
events
PENNSYLVANIA, ALLEGHENY: 753 events
ALABAMA, LAUDERDALE: 714 events
ARIZONA, MARICOPA: 710 events
ALABAMA, COLBERT: 618 events
ATLANTIC NORTH, CHESAPEAKE BAY SANDY PT TO N BEACH MD: 609 events
OKLAHOMA, OKLAHOMA: 579 events
COLORADO, EL PASO: 578 events
TEXAS, TARRANT: 572 events
```

1.8 Analysis 4: Combined Storm + Location Factors

[26]:
```
# Aggregate harm by (EVENT_TYPE, STATE) pairs
# This shows interaction between storm type and location
harm_by_event_state = (
    rdd_harm
    .map(lambda x: ((
        x[0]['EVENT_TYPE'] if x[0]['EVENT_TYPE'] else 'UNKNOWN',
        x[0]['STATE'] if x[0]['STATE'] else 'UNKNOWN'
    ), x[1]))
    .filter(lambda x: x[0][0] and x[0][0] != "" and x[0][1] and x[0][1] != "")
    .mapValues(lambda v: (v, 1))
    .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))
    .mapValues(lambda x: (x[0], x[1], x[0] / x[1] if x[1] > 0 else 0))
)

harm_by_event_state_sorted = harm_by_event_state.sortBy(lambda x: x[1][0],␣
 ↪ascending=False)

print("\nTop 10 (Event Type, State) Combinations by Total Harm:")
for (event_type, state), (total_harm, count, avg_harm) in␣
 ↪harm_by_event_state_sorted.take(10):
    print(f"{event_type} in {state}: Total={int(total_harm)}, Count={count},␣
 ↪Avg={avg_harm:.2f}")
```

```
Top 10 (Event Type, State) Combinations by Total Harm:
[Stage 40:>                                                         (0 + 1) / 1]
Excessive Heat in TEXAS: Total=1428, Count=29, Avg=49.24
Excessive Heat in ARIZONA: Total=1163, Count=190, Avg=6.12
Heat in ARIZONA: Total=871, Count=454, Avg=1.92
Tornado in KENTUCKY: Total=763, Count=26, Avg=29.35
Tornado in TENNESSEE: Total=627, Count=41, Avg=15.29
Tornado in TEXAS: Total=495, Count=56, Avg=8.84
Tornado in MISSISSIPPI: Total=472, Count=47, Avg=10.04
Heat in TEXAS: Total=406, Count=37, Avg=10.97
Wildfire in CALIFORNIA: Total=364, Count=43, Avg=8.47
Drought in MISSOURI: Total=350, Count=1, Avg=350.00
```

1.9 Summary & Next Steps

This Task 2 analysis provides: 1. **Baseline aggregations** comparing storm factors (EVENT_TYPE, MAGNITUDE_TYPE) vs. location factors (STATE, CZ_NAME) 2. **Event frequency proxy** for population density (more events = likely more populated) 3. **Combined factor analysis** showing interactions between storm and location factors

For Future Phases (Random Forest Modeling): - Use these aggregations to engineer features - Join with actual population density data (external source) - Train Random Forest model with: - Storm features: EVENT_TYPE, MAGNITUDE_TYPE, MAGNITUDE - Location features: STATE, CZ_NAME, EVENT_COUNT_PER_CZ (population proxy) - Interaction features: EVENT_TYPE × STATE, MAGNITUDE × EVENT_COUNT - Calculate feature importance to determine which factors are strongest predictors: - Stuff like SHAP, Permutation importance, gini, etc.

[27]:
```
# Stop Spark session
spark.stop()
```