02_drone_impact_analyse_final

December 11, 2025

1 Dronewars Dataset

1.1 Data conversie en setup

[1]:
```
# --- BLOK 1: SETUP & DATA LOADING ---
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import folium
import os
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, year, month, sum as spark_sum, avg, count
```

[2]:
```
# --- BLOK 1: SETUP & DATA LOADING ---


# 1. Start Spark
spark = SparkSession.builder.appName("Drone Impact Analysis").getOrCreate()
sns.set_theme(style="whitegrid")
```

[12]:
```
excel_file = "/home/jovyan/work/drone-project/DroneWarsData.xlsx"
xl = pd.ExcelFile(excel_file)
print(xl.sheet_names)
```

```
['Afghanistan', 'Somalia', 'Yemen', 'Pakistan', 'Variables', 'All',
'Unknown_Locations', 'All_WithoutUnknown', 'All_WithUnknown', 'Copy of
All_WithoutUnknown', 'US Confirmed', 'US Confirmed (updated)', 'Confirmed vs
Unconfirmed', 'Copy of Confirmed vs Unconfirme', 'Afghanistan(2)', 'Somalia(2)',
'Yemen(2)', 'All(2)']
```

[13]:
```
# EENMALIG !

# 2. Excel naar CSV Converteren (Auto-detectie)
# Pas deze bestandsnaam aan naar jouw exacte bestandsnaam!
excel_file = "/home/jovyan/work/drone-project/DroneWarsData.xlsx"
csv_file = "/home/jovyan/work/drone-project/dronewars_data_all.csv"
sheet_name = "All"
```

```
if not os.path.exists(csv_file):
    print(f"Converteren van {excel_file} naar CSV...")
    # We lezen sheet 0. Als er meerdere tabbladen zijn, check dit even!
    try:
        pdf_raw = pd.read_excel(excel_file, sheet_name=sheet_name)
        pdf_raw.to_csv(csv_file, index=False)
        print("Succesvol geconverteerd.")
    except Exception as e:
        print(f"FOUT bij converteren: {e}")
        print("Check of het bestand in de map staat en de naam klopt.")
else:
    print(f"CSV bestand {csv_file} bestaat al. We slaan conversie over.")
```

```
Converteren van /home/jovyan/work/drone-project/DroneWarsData.xlsx naar CSV…
Succesvol geconverteerd.
```

1.2 Data Verkennen

[3]:
```
csv_file = "/home/jovyan/work/drone-project/dronewars_data_all.csv"

df_strikes = spark.read.csv(csv_file, header=True, inferSchema=True)

print("Schema van de Drone Wars dataset:")
df_strikes.printSchema()
```

```
Schema van de Drone Wars dataset:
root
 |-- Strike ID: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- Date (MM-DD-YYYY): string (nullable = true)
 |-- President: string (nullable = true)
 |-- Most Specific Location: string (nullable = true)
 |-- Most Specific Lat/Long: string (nullable = true)
 |-- Latitude: string (nullable = true)
 |-- Longitude: string (nullable = true)
 |-- Minimum total people killed: string (nullable = true)
 |-- Maximum total people killed: string (nullable = true)
 |-- Minimum civilians reported killed: string (nullable = true)
 |-- Maximum civilians reported killed: string (nullable = true)
 |-- Minimum children reported killed: string (nullable = true)
 |-- Maximum children reported killed: string (nullable = true)
 |-- Minimum reported injured: string (nullable = true)
 |-- Ratio of Civilians to Total Killed: double (nullable = true)
 |-- Other (Non-civilian/children Killed): string (nullable = true)
 |-- Unnamed: 17: double (nullable = true)
```

1.3 Data cleaning

[4]:
```
from pyspark.sql.functions import to_date, coalesce, lit
```

[5]:
```
df_strikes.columns
```

[5]:
```
['Strike ID',
 'Country',
 'Date (MM-DD-YYYY)',
 'President',
 'Most Specific Location',
 'Most Specific Lat/Long',
 'Latitude',
 'Longitude',
 'Minimum total people killed',
 'Maximum total people killed',
 'Minimum civilians reported killed',
 'Maximum civilians reported killed',
 'Minimum children reported killed',
 'Maximum children reported killed',
 'Minimum reported injured',
 'Ratio of Civilians to Total Killed',
 'Other (Non-civilian/children Killed)',
 'Unnamed: 17']
```

[6]:
```
# --- BLOK 2: DATA CLEANING (volledig) ---
from pyspark.sql.functions import to_date, coalesce, lit, col, year,expr,trim,␣
 ↪regexp_replace

rename_map = {
    "Minimum total people killed": "min_killed",
    "Maximum total people killed": "max_killed",
    "Minimum civilians reported killed": "min_civilians",
    "Maximum civilians reported killed": "max_civilians",
    "Minimum children reported killed": "min_children",
    "Maximum children reported killed": "max_children",
    "Minimum reported injured": "min_injured",
    "Ratio of Civilians to Total Killed": "ratio_civ_to_total",
    "Other (Non-civilian/children Killed)": "other_killed",
}

# 1) Datum parsing + basisfilter
df_clean = (
    df_strikes
    .withColumn("parsed_ts", expr("try_to_timestamp(`Date (MM-DD-YYYY)`,␣
 ↪'dd-MM-yyyy')"))
    .filter(col("parsed_ts").isNotNull())
```

```
    .withColumn("parsed_date", col("parsed_ts").cast("date"))
    .withColumn("year", year(col("parsed_date")))
)

# 1b) Trim whitespace van string kolommen
string_cols = ["Country","President"]
for c in string_cols:
    if c in df_clean.columns:
        df_clean = df_clean.withColumn(c, trim(col(c)))

if "Latitude" in df_clean.columns:
    df_clean = df_clean.withColumn("Latitude", regexp_replace(col("Latitude"),␣
 ↪"^,", ""))
    df_clean = df_clean.withColumn("Latitude", regexp_replace(col("Latitude"),␣
 ↪",", "."))

if "Longitude" in df_clean.columns:
    df_clean = df_clean.withColumn("Longitude",␣
 ↪regexp_replace(col("Longitude"), "^,", ""))
    df_clean = df_clean.withColumn("Longitude",␣
 ↪regexp_replace(col("Longitude"), ",", "."))
# 2) Kolomnamen verkorten
for old, new in rename_map.items():
    if old in df_clean.columns:
        df_clean = df_clean.withColumnRenamed(old, new)

# 3) Casting
int_cols = [
    "min_killed", "max_killed",
    "min_civilians", "max_civilians",
    "min_children", "max_children",
    "min_injured", "other_killed"
]


for c in int_cols:
    if c in df_clean.columns:
        df_clean = df_clean.withColumn(c, col(c).cast("integer"))

if "ratio_civ_to_total" in df_clean.columns:
    df_clean = df_clean.withColumn("ratio_civ_to_total",␣
 ↪expr("try_cast(ratio_civ_to_total as double)"))

# Coördinaten: gebruik try_cast om ongeldige waarden naar NULL te maken
if "Latitude" in df_clean.columns:
    df_clean = df_clean.withColumn("Latitude", expr("try_cast(Latitude as␣
 ↪double)"))
```

```
if "Longitude" in df_clean.columns:
    df_clean = df_clean.withColumn("Longitude", expr("try_cast(Longitude as␣
 ↪double)"))

# 4) Nulls opvullen (alleen op integer-kolommen)
df_clean = df_clean.fillna(0, subset=[c for c in int_cols if c in df_clean.
 ↪columns])

print("Data Cleaning voltooid.")
print(f"Aantal geldige rijen: {df_clean.count()}")
df_clean.printSchema()
```

```
Data Cleaning voltooid.
Aantal geldige rijen: 682
root
 |-- Strike ID: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- Date (MM-DD-YYYY): string (nullable = true)
 |-- President: string (nullable = true)
 |-- Most Specific Location: string (nullable = true)
 |-- Most Specific Lat/Long: string (nullable = true)
 |-- Latitude: double (nullable = true)
 |-- Longitude: double (nullable = true)
 |-- min_killed: integer (nullable = true)
 |-- max_killed: integer (nullable = true)
 |-- min_civilians: integer (nullable = true)
 |-- max_civilians: integer (nullable = true)
 |-- min_children: integer (nullable = true)
 |-- max_children: integer (nullable = true)
 |-- min_injured: integer (nullable = true)
 |-- ratio_civ_to_total: double (nullable = true)
 |-- other_killed: integer (nullable = true)
 |-- Unnamed: 17: double (nullable = true)
 |-- parsed_ts: timestamp (nullable = true)
 |-- parsed_date: date (nullable = true)
 |-- year: integer (nullable = true)
```

1.4 Data verkennen

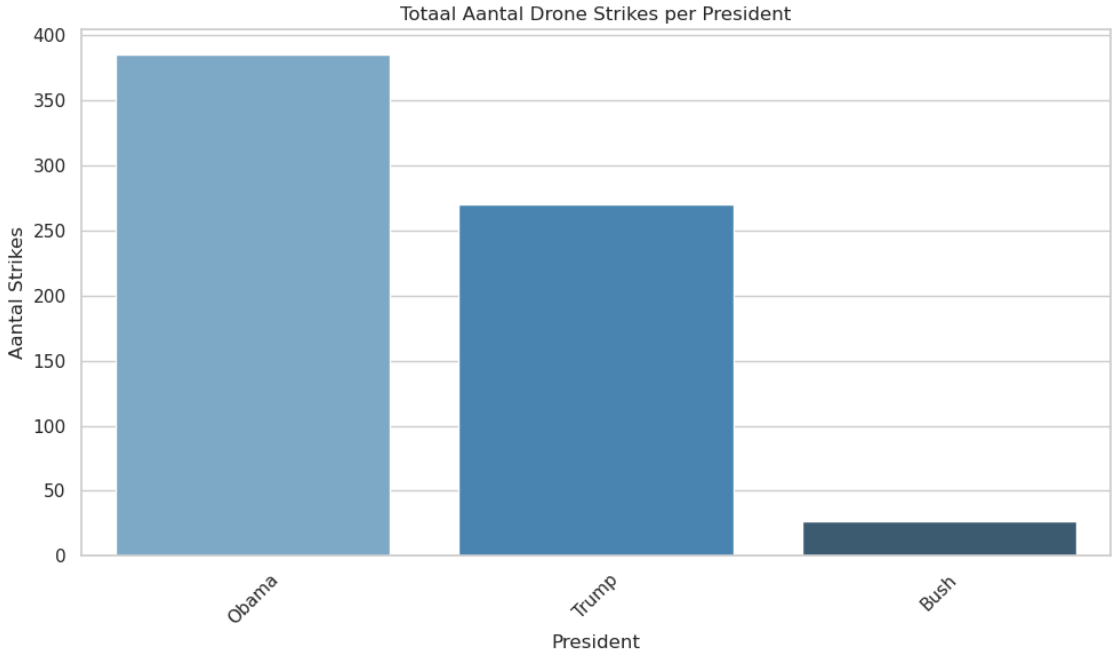
1.4.1 Aantal drone strikes per President

Deze visualisatie toont welk presidentschap de meeste drone-activiteit kende.

[12]:
```
# --- Visualisatie: Aantal Strikes per President ---
# Stap 1: Aggregeren met Spark
# We groeperen op President en tellen het aantal rijen (strikes)
```

```
df_pres_count = (
    df_clean
    .groupBy("President")
    .count()
    .orderBy("count", ascending=False)
)

# Stap 2: Naar Pandas voor visualisatie
pdf_pres_count = df_pres_count.toPandas()

# Stap 3: Plotten (Barplot)
plt.figure(figsize=(10, 6))
sns.barplot(data=pdf_pres_count, x="President", y="count", palette="Blues_d")

plt.title("Totaal Aantal Drone Strikes per President")
plt.xlabel("President")
plt.ylabel("Aantal Strikes")
plt.xticks(rotation=45) # Labels draaien voor leesbaarheid
plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_24428/182300162.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(data=pdf_pres_count, x="President", y="count", palette="Blues_d")
```

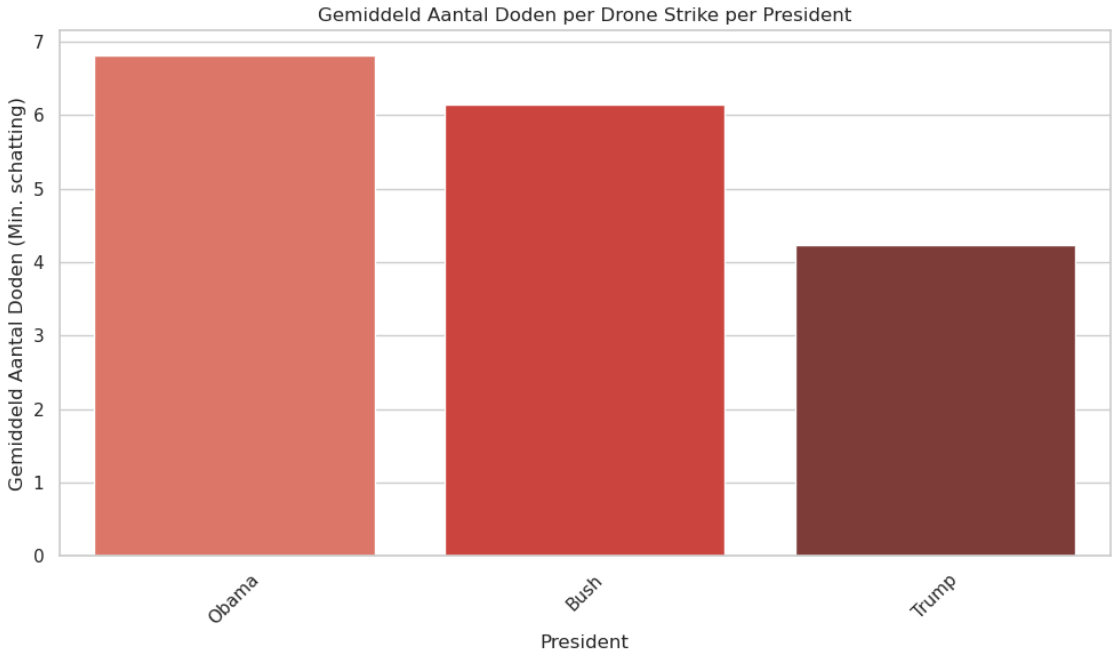


1.4.2 Gemiddeld aantal doden per strike per president

Deze grafiek toont de intensiteit of letaliteit van de aanvallen per administratie, in plaats van puur het volume.

[13]:
```
# --- Visualisatie: Gemiddelde Lethaliteit per President ---
from pyspark.sql.functions import avg

# Stap 1: Aggregeren (Gemiddelde berekenen)
df_pres_avg = (
    df_clean
    .groupBy("President")
    .agg(avg("min_killed").alias("avg_deaths"))
    .orderBy("avg_deaths", ascending=False)
)

# Stap 2: Naar Pandas
pdf_pres_avg = df_pres_avg.toPandas()

# Stap 3: Plotten
plt.figure(figsize=(10, 6))
sns.barplot(data=pdf_pres_avg, x="President", y="avg_deaths", palette="Reds_d")

plt.title("Gemiddeld Aantal Doden per Drone Strike per President")
plt.xlabel("President")
plt.ylabel("Gemiddeld Aantal Doden (Min. schatting)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_24428/1170993934.py:17: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(data=pdf_pres_avg, x="President", y="avg_deaths",
palette="Reds_d")
```

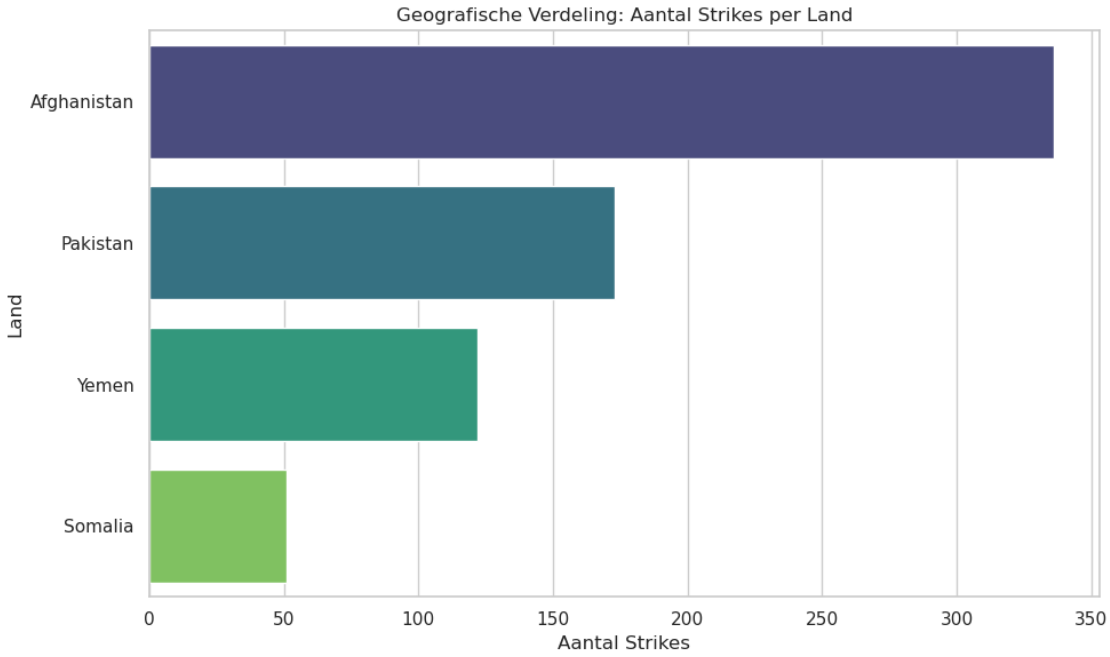


1.4.3 Aantal Drone Strikes Per Land

Deze visualisatie geeft de geografische focus van de drone-oorlogvoering weer.

[14]:
```
# --- Visualisatie: Totaal Aantal Strikes per Land ---
# Stap 1: Aggregeren
df_country_count = (
    df_clean
    .groupBy("Country")
    .count()
    .orderBy("count", ascending=False)
)

# Stap 2: Naar Pandas
pdf_country_count = df_country_count.toPandas()

# Stap 3: Plotten (Horizontale Barplot voor betere leesbaarheid landnamen)
plt.figure(figsize=(10, 6))
sns.barplot(data=pdf_country_count, y="Country", x="count", palette="viridis")

plt.title("Geografische Verdeling: Aantal Strikes per Land")
plt.xlabel("Aantal Strikes")
plt.ylabel("Land")
plt.tight_layout()
plt.show()
```

```
/tmp/ipykernel_24428/3167773748.py:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(data=pdf_country_count, y="Country", x="count", palette="viridis")
```



1.5 Data analyses

1.5.1 Word Count op Locaties

Doel: Analyseer welke locaties (dorpen/wijken) het vaakst voorkomen in de beschrijvingen. Dit traint je map en reduceByKey skills.

Concept: We gebruiken RDD's omdat tekst ongestructureerd is. We pakken de kolom Most Specific Location, splitten de tekst en tellen de woorden.

[36]:
```
df_clean.select("Most Specific Location").show(20,truncate=False)
```

```
+-------------------------------------------------------------------+
|Most Specific Location                                             |
+-------------------------------------------------------------------+
|Kari Kot, South Waziristan, Pakistan                               |
|Ghundikala, North Waziristan, Pakistan                             |
|Mandi Khel, North Waziristan, Pakistan                             |
|Mandi Khel, North Waziristan, Pakistan                             |
|Datta Khel, North Waziristan, Pakistan                             |
```

```
|Boya, North Waziristan, Pakistan                                                |
|Shabwa, Yemen                                                                   |
|Spera, Khost, Afghanistan                                                       |
|Madin Village, South Waziristan, Pakistan                                       |
|Lajhmarai area, Tehsil Birmal, South Waziristan, Pakistan                       |
|Shaltan Darra, Shegal, Kunar, Afghanistan                                       |
|Mogadishu                                                                       |
|Mosaki, North Waziristan, Pakistan                                              |
|Ghundi Killi, Tappi area, Tehsil Miranshah, North Waziristan, Pakistan          |
|Gayan, Paktika, Afghanistan                                                     |
|Spera, Khost, Afghanistan                                                       |
|Radaa, Bayda, Yemen                                                             |
|Alwara Mandi, Datta Khel or Shawal, North Waziristan, Pakistan                  |
|Sanzali, North Waziristan, Pakistan                                             |
|Sanzali, North Waziristan, Pakistan                                             |
+--------------------------------------------------------------------------------+
only showing top 20 rows
```

[7]:
```python
# --- RDD: Word Count op Locaties ---
# Stap 1: Selecteer de tekstkolom en zet om naar RDD
rdd_locations = df_clean.select("Most Specific Location") \
    .filter(col("Most Specific Location").isNotNull()) \
    .rdd.flatMap(lambda row: [loc.strip() for loc in row[0].split(',')]) # Split op komma en verwijder spaties

# Stap 2: Map (woord -> 1) en Reduce (tel op)
# We filteren ook korte woordjes weg (bv. "in", "of", "near")
stop_words = [
    "in", "of", "the", "near", "district", "province", "village", "area", "unknown",
    "afghanistan", "pakistan", "yemen", "somalia", "iraq", "syria", "iran",
    "north", "south", "east", "west", "central", "northern", "southern", "eastern", "western",
    "region", "county", "state", "city", "town", "and", "or", "at", "on"
]

word_counts = rdd_locations \
    .map(lambda word: word.lower().strip(",.")) \
    .filter(lambda word: len(word) > 3 and word not in stop_words) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b) \
    .sortBy(lambda x: x[1], ascending=False)

# Stap 3: Resultaat tonen
print("Top 10 Meest Genoemde Locaties/Termen:")
for word, count in word_counts.take(10):
    print(f"{word}: {count}")
```

```
Top 10 Meest Genoemde Locaties/Termen:
north waziristan: 129
nangarhar: 94
helmand: 43
south waziristan: 38
achin: 34
kunar: 32
abyan: 30
shabwa: 24
datta khel: 21
paktika: 18
```

1.5.2 Burger Dodelijk Ratio

Doel: Bereken de "Civillian Lethality Ratio" per land met SQL.

Concept: SQL in Spark is krachtig voor aggregaties. We registreren de DataFrame eerst als "TempView".

[8]:
```
# --- SQL: Civilian Ratio per Land ---
# Stap 1: Maak SQL Table
df_clean.createOrReplaceTempView("strikes")

# Stap 2: SQL Query
# We sommeren de totalen en berekenen dan het percentage
sql_query = """
    SELECT
        Country,
        SUM(min_killed) as total_deaths,
        SUM(min_civilians) as civilian_deaths,
        ROUND((SUM(min_civilians) / SUM(min_killed)) * 100, 2) as␣
 ↪civilian_ratio_pct
    FROM strikes
    GROUP BY Country
    ORDER BY total_deaths DESC
"""

df_ratios = spark.sql(sql_query)
df_ratios.show()
```

```
+-----------+------------+---------------+------------------+
|    Country|total_deaths|civilian_deaths|civilian_ratio_pct|
+-----------+------------+---------------+------------------+
|Afghanistan|        2020|            153|              7.57|
|   Pakistan|         981|            115|             11.72|
|      Yemen|         499|             50|             10.02|
|    Somalia|         428|             14|              3.27|
+-----------+------------+---------------+------------------+
```

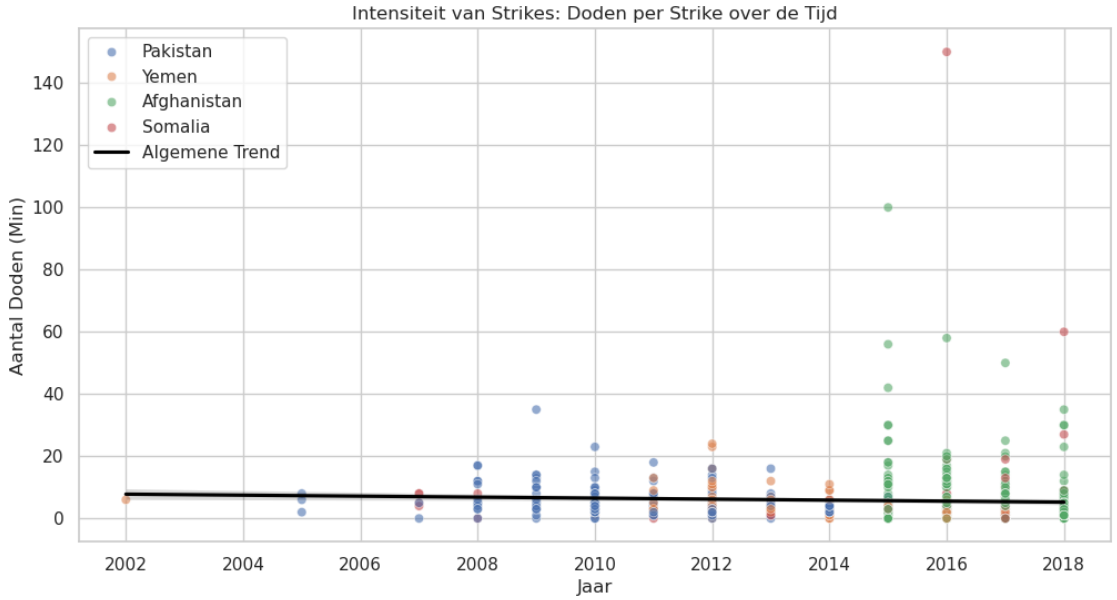
[ ]:

1.5.3 Scatterplot & Regressie: Tijd vs Slachtoffers

Doel: Is er een correlatie tussen tijd en aantal slachtoffers per strike? (Escalatie analyse).

Concept: We plotten elke strike als een punt. regplot voegt een trendlijn toe.

[9]:
```
# --- Visualisatie: Scatterplot met Trendlijn ---
# Stap 1: Data naar Pandas (alleen nodige kolommen om memory te sparen)
# We zetten de datum om naar een numerieke waarde (jaar) voor de regressie
pdf_scatter = df_clean.select("year", "min_killed", "Country").toPandas()

# Stap 2: Plotten
plt.figure(figsize=(12, 6))

# Scatterplot: Elk punt is een strike
sns.scatterplot(data=pdf_scatter, x="year", y="min_killed", hue="Country",␣
 ↪alpha=0.6)

# Regressielijn (Trend): Gaat het dodental per strike omhoog of omlaag?
sns.regplot(data=pdf_scatter, x="year", y="min_killed", scatter=False,␣
 ↪color="black", label="Algemene Trend")

plt.title("Intensiteit van Strikes: Doden per Strike over de Tijd")
plt.xlabel("Jaar")
plt.ylabel("Aantal Doden (Min)")
plt.legend()
plt.show()
```

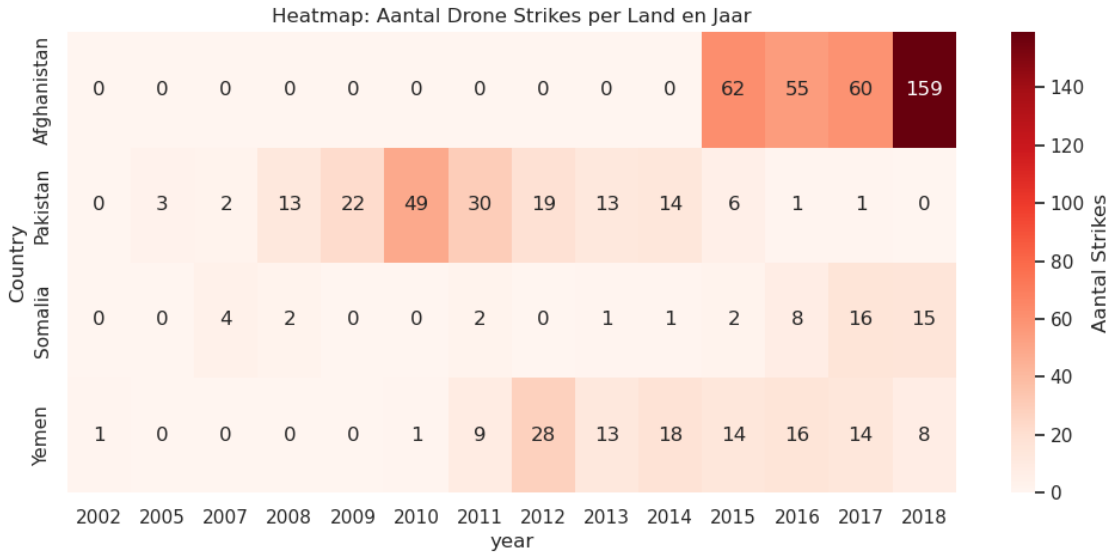


1.5.4 Heatmap: Jaar vs Land intensiteit

Doel: Wanneer was het conflict waar het hevigst?

Concept: Pivot tables zijn essentieel voor heatmaps.

[10]:
```
# --- Visualisatie: Heatmap van Conflict Intensiteit ---
# Stap 1: Aggregeren in Spark
df_heatmap = df_clean.groupBy("Country", "year") \
    .count() \
    .orderBy("year")

# Stap 2: Pivot in Pandas (Rijen=Land, Kolommen=Jaar, Waarde=Aantal Strikes)
pdf_heatmap = df_heatmap.toPandas()
matrix = pdf_heatmap.pivot(index="Country", columns="year", values="count").
 ↪fillna(0)

# Stap 3: Plotten
plt.figure(figsize=(12, 5))
sns.heatmap(matrix, annot=True, fmt='g', cmap="Reds", cbar_kws={'label':␣
 ↪'Aantal Strikes'})
plt.title("Heatmap: Aantal Drone Strikes per Land en Jaar")
plt.show()
```



1.5.5 Folium Interactieve kaart

[60]:
```
import folium
from folium.plugins import HeatMap

# --- Geo Visualisatie: Markers & Heatmap ---
# Stap 1: Data ophalen (Locatie + Info voor popup)
# Filter ongeldige coordinaten (0,0 of null)
df_geo = df_clean.filter((col("Latitude") != 0) & (col("Latitude").
 ↪isNotNull())) \
                 .select("Latitude", "Longitude", "min_killed",␣
 ↪"min_civilians", "Date (MM-DD-YYYY)", "Country")

pdf_geo = df_geo.toPandas()

# Stap 2: Kaart opzetten
m = folium.Map(location=[32, 65], zoom_start=4, tiles="CartoDB dark_matter")

# Laag 1: Heatmap (Toont concentratie)
# We maken een lijst van [lat, lon, gewicht]
heat_data = [[row['Latitude'], row['Longitude'], row['min_killed']] for index,␣
 ↪row in pdf_geo.iterrows()]
HeatMap(heat_data, radius=15, blur=10, name="Strike Density").add_to(m)

# Laag 2: Individuele Markers
marker_group = folium.FeatureGroup(name="Individual Strikes")

for _, row in pdf_geo.iterrows():
    # Kleurcodering (Rood = Burgerslachtoffers, Blauw = Geen)
    color = "red" if row['min_civilians'] > 0 else "blue"

    # HTML Popup tekst
    popup_html = f"""
    <b>Datum:</b> {row['Date (MM-DD-YYYY)']}<br>
    <b>Land:</b> {row['Country']}<br>
    <b>Totaal Doden:</b> {row['min_killed']}<br>
    <b>Burgers:</b> {row['min_civilians']}
    """

    folium.CircleMarker(
        location=[row['Latitude'], row['Longitude']],
        radius=3, # Klein houden voor overzicht
        color=color,
        fill=True,
        fill_opacity=0.7,
        popup=folium.Popup(popup_html, max_width=200)
    ).add_to(marker_group)
```

```
marker_group.add_to(m)

# Layer Control toevoegen (zodat je kan switchen)
folium.LayerControl().add_to(m)

# Opslaan
m.save("/home/jovyan/work/drone-project/drone_impact_advanced.html")
print("Geavanceerde kaart opgeslagen als 'drone_impact_advanced.html'")
```

```
Geavanceerde kaart opgeslagen als 'drone_impact_advanced.html'
```

[55]:
```
df_clean.select("Latitude","Longitude").show(10)
```

```
+---------+---------+
| Latitude|Longitude|
+---------+---------+
|32.270651|69.554179|
|32.959854|70.156371|
|33.001031|70.364339|
|33.001031|70.364339|
|33.150049|70.433361|
|32.943336|69.899944|
| 14.75463|46.516261|
| 33.19392|69.515022|
|32.320237| 69.85974|
|32.378567|69.430782|
+---------+---------+
only showing top 10 rows
```

1.5.6 Jupyter Widgets demo: Filteren op Land en President

Doel: Filter data op basis van Land en President.

Concept: interact koppelt een functie aan een dropdown. Elke keer als je kiest, wordt de grafiek opnieuw getekend.

[ ]:
```
import ipywidgets as widgets
from IPython.display import display, clear_output

# --- Dashboard: Interactieve Filtering ---

# Stap 1: Dropdown opties ophalen
countries = [row.Country for row in df_clean.select("Country").distinct().
 ↪collect() if row.Country]
countries.sort()

# Stap 2: De Widget
dropdown = widgets.Dropdown(
```

```
    options=countries,
    description='Kies Land:',
    disabled=False,
)

# Stap 3: De Functie die update
def plot_country_stats(country):
    # Filter data in Spark/Pandas
    # We gebruiken hier Pandas op de reeds ingeladen df_clean (geconverteerd␣
 ↪naar pandas voor snelheid in widget)
    # Voor een ECHTE Big Data app zou je hier een nieuwe Spark query doen, maar␣
 ↪voor widgets is local pandas sneller.
    pdf_subset = df_clean.filter(col("Country") == country).select("year",␣
 ↪"min_killed", "min_civilians").toPandas()

    if pdf_subset.empty:
        print(f"Geen data voor {country}")
        return

    # Aggregeren per jaar
    pdf_agg = pdf_subset.groupby("year").sum().reset_index()

    # Plotten
    plt.figure(figsize=(10, 5))
    sns.lineplot(data=pdf_agg, x="year", y="min_killed", label="Totaal Doden",␣
 ↪marker="o")
    sns.lineplot(data=pdf_agg, x="year", y="min_civilians", label="Burger␣
 ↪Doden", marker="o", color="red")

    plt.title(f"Slachtoffers in {country} over tijd")
    plt.ylabel("Aantal")
    plt.grid(True)
    plt.show()

# Stap 4: Koppelen
widgets.interact(plot_country_stats, country=dropdown);
```

```
interactive(children=(Dropdown(description='Kies Land:', options=('Afghanistan',␣
 ↪'Pakistan', 'Somalia', 'Yemen…
```