Mini Project 1: Shark attacks

## **Business idea:** Surfing Insurance Provider

A specialized insurance company that offers coverage for surfers against shark attacks and related water accidents.

### How to Use Shark Attack Data:

* **Identify high-risk locations:** Filter the dataset for ‘Activity’ = Surfing and analyze which countries, states, and beaches have the highest incidence of shark attacks.
* **Demographic analysis:** Analyze age, sex, and time variables to identify risk groups.
* **Seasonality and timing:** Look for patterns in months/times when attacks are more frequent.

### Roadmap:

1. **Filter** the dataset for rows where ‘Activity’ contains ‘Surfing’.
2. **Clean** missing values in key columns: ‘Location’, ‘Country’, ‘State’, ‘Date’, ‘Injury’, ‘Fatal Y/N’.
3. **Aggregate** attack counts by location and time (year, month).
4. **Visualize** hotspots and trends (e.g., heatmaps, bar charts).
5. **Profile** risk groups by age, sex, and time of day.
6. **Deliverable:** Statistical report and interactive dashboard for insurance pricing

## General Steps

### Data Cleaning:

* Handle missing values (drop, impute or flag).
* Drop irrelevant columns (‘Unnamed: 21’, ‘Unnamed: 22’, etc.).
* Correct duplicated or inconsistent entries.
* Standardize categorical values (e.g., country and activity names).
* Convert date fields to datetime objects.

### Data Reduction:

* Filter rows matching your business scenario (e.g., only specific activities or locations).
* Create new DataFrames focused on relevant data.

### EDA and Visualization:

* Use aggregation and grouping to analyze statistics.
* Visualize findings with bar charts, maps, or heatmaps.

### Reporting/Presentation:

* Prepare concise slides showing objectives, methods, findings, and business implications.
* Build a **“Minimum Viable Product” (MVP)**, such as a dashboard or report.

## Example: Shark Repellent Roadmap (Expanded)

1. Filter:  
   df\_us = df[df['Country'] == 'USA']
2. Clean:  
   Drop columns with excessive missing data, standardize locations.
3. Group/Analyze:  
   df\_state = df\_us.groupby('State').size().reset\_index(name='AttackCount')
4. Visualize:  
   State-wise attack frequency bar chart.
5. Report:  
   Summarize high-risk states and propose repellent deployment plan.

## How to Approach the Project

* Scope: What data is relevant? Who is the user/customer?
* Roadmap: List the steps from raw data to actionable business insights.
* MVP: What is the simplest version of your product/analysis?
* Present: Show your findings, recommendations, and business impact.

## Dataset Structure and Initial Observations

You loaded the dataset from an Excel file (GSAF5.xls) using Pandas. The first few rows and the .info() output provide valuable insight into the data's structure and quality.

### Columns Overview

The dataset contains 23 columns:

* **Date**: Date of the attack (varied formats, e.g., "16th August 2025" or "Before 1903")
* **Year**: Numeric year (float; some are 0, indicating missing or unknown)
* **Type**: Attack type ("Provoked", "Unprovoked", etc.)
* **Country, State, Location**: Geographical info (sometimes missing or inconsistent)
* **Activity**: What the person was doing (e.g., "Surfing", "Fishing")
* **Name, Sex, Age**: Victim details (frequent nulls or vague entries like "Not stated")
* **Injury**: Description of injuries (unstructured text)
* **Fatal Y/N**: Fatality indicator (values like "Y", "N", "FATAL", missing)
* **Time**: Time of attack (inconsistent formats, many nulls)
* **Species**: Shark species (many "Undetermined" or missing)
* **Source, pdf, href formula, href**: References and links (mostly for documentation)
* **Case Number, Case Number.1, original order**: Case identifiers (frequent nulls)
* **Unnamed: 21, Unnamed: 22**: Seemingly empty columns

### Data Types

* Most columns are detected as object (string), except for Year and original order (numeric).
* There are many missing values, esp. in columns like Age, Time, Species, and the last two unnamed columns.

## Quality Issues and Problems Identified

### A. Missing Data

* **Many Nulls:** Columns such as State, Location, Age, Time, Species, pdf, etc. have substantial missing values.
* **Zero or Placeholder Values:** Some rows have Year as 0, or Age as "?" or NaN.

### B. Inconsistent Formatting

* **Date:** Multiple formats, e.g., “7th August”, “Before 1903”, “16th August 2025”, “1883-1889”.
* **Fatal Y/N:** Uses mixed values ("Y", "N", "FATAL", "FATAL. ...", null).
* **Time:** Variations like "1055 hrs", "0730hrs", "Not stated", NaN.
* **Country/State:** Variations in case ("USA" vs "usa"), spelling, and missing values.

### C. Unstructured or Messy Text

* **Injury:** Long descriptive text, not standardized.
* **Activity:** Some unique activities, misspellings, or multiple activities.

### D. Redundant or Useless Columns

* **Unnamed: 21, Unnamed: 22:** Almost entirely empty.
* **pdf, href formula, href:** Reference links may be useful for further research but not for core analysis.
* **Case Number, Case Number.1:** May be redundant if only one is used.

## Data Cleaning and Transformation Plan

### Step 1: Column Assessment & Reduction

* **Drop Unneeded Columns:** Remove Unnamed: 21, Unnamed: 22, and possibly pdf, href formula, href unless required for analysis.
* **Assess Redundant Columns:** Keep only one of Case Number/Case Number.1 or combine if needed.

### Step 2: Standardize and Impute Data

### Dates

* Use Pandas to\_datetime() with custom parsing for varied date formats.
* Extract year from Date where missing in Year.

### Categorical Variables

* Standardize values in Country, State, Type, Fatal Y/N, Sex.
* Lowercase, strip whitespace, and unify spelling.

### Numerical Variables

* Convert Age to numeric, set invalid values ("?", NaN, "Unkwnon") to null.
* Set year as integer; if Year is 0, try to impute from Date.

### Missing Values

* Decide on imputation (e.g., mode for categorical, median for numeric) or leave as null.
* For critical columns (Country, Type, Date), consider dropping rows if missing.

### Step 3: Feature Engineering

* **Create Flags:** For fatal vs non-fatal (Fatal Y/N), provoked vs unprovoked (Type).
* **Extract Info:** From Injury, Activity (e.g., keywords, severity).
* **Geographical Parsing:** Normalize geographical fields for mapping.

### Step 4: Finalize DataFrame

* **Remove Duplicates:** Check for duplicated records by key columns.
* **Rename Columns:** For clarity and consistency.
* **Validate Data:** Check for outliers or impossible values (e.g., Age < 0, future dates).

## Summary Table of Columns

| **Column** | **Type** | **Issues** | **Action** |
| --- | --- | --- | --- |
| Date | object | Inconsistent formats | Parse, standardize |
| Year | float | Some 0/NaN, redundant with Date | Impute/clean |
| Type | object | Mixed values | Standardize |
| Country | object | Spelling/case variation, missing | Standardize/impute |
| State | object | Missing, inconsistent | Standardize/impute |
| Location | object | Missing, varied detail | Standardize/impute |
| Activity | object | Varied, unstructured | Standardize/categorize |
| Name | object | "Not stated", missing | Standardize |
| Sex | object | "M", "F", missing | Standardize/impute |
| Age | object | "?", NaN, outliers | Convert, impute |
| Injury | object | Unstructured text | Extract/categorize |
| Fatal Y/N | object | "Y", "N", "FATAL", missing | Standardize |
| Time | object | Many nulls, varied formats | Parse, standardize |
| Species | object | "Undetermined", missing | Standardize |
| Source | object | Text, possible duplicates | Standardize |
| pdf, href\* | object | Mostly null, documentation | Drop if not needed |
| Case Number\* | object | Redundant, missing | Keep one, clean |
| original order | float | Often null, unclear purpose | Assess |
| Unnamed: 21/22 | object | Empty | Drop |

## Recommended Next Steps

1. Clean up column names and drop empty/redundant columns.
2. Standardize categorical and text fields.
3. Convert columns to correct data types.
4. Handle missing values appropriately.
5. Document assumptions and cleaning steps for reproducibility.
6. Generate summary statistics and visualizations to further assess data quality.

### 1. Clean up column names and drop empty/redundant columns

### Analysis

* Some columns are likely unnecessary for analysis (e.g., Unnamed: 21, Unnamed: 22, pdf, href formula, href, which are mostly empty or metadata).
* Column names may have varying formats (spaces, punctuation, etc.) that are best standardized for programming.

### Code & Explanation

Python

# Clean up column names: remove spaces, convert to lowercase, replace spaces with underscores

df.columns = [col.strip().lower().replace(' ', '\_') for col in df.columns]

# Drop empty/redundant columns

# Check which columns have almost all missing values

print(df.isnull().sum())

# Let's drop columns that are almost entirely NaN or are meta-data

drop\_cols = ['unnamed:\_21', 'unnamed:\_22', 'pdf', 'href\_formula', 'href', 'case\_number.1', 'original\_order']

df = df.drop(columns=drop\_cols, errors='ignore')

### Explanation:

* The first line cleans up column names for easier referencing.
* The second block prints how many missing values each column has, helping visually confirm which to drop.
* The third line drops columns that are mostly empty or redundant. errors='ignore' prevents errors if a column is missing.

### 2. Standardize categorical and text fields

### Analysis

* Many categorical columns (type, country, state, sex, etc.) have inconsistent capitalization, spelling, or extra whitespace.
* Example: "M" vs "male" vs "Male" for sex, or "USA" vs "usa".

### Code & Explanation

Python

# Standardize categorical columns

# Example for Sex and Country

df['sex'] = df['sex'].str.strip().str.lower().replace({'male': 'm', 'female': 'f', 'nan': None})

df['country'] = df['country'].str.strip().str.title() # Makes "usa" -> "Usa", "AUSTRALIA" -> "Australia"

# For columns like 'type', unify spelling and capitalization

df['type'] = df['type'].str.strip().str.capitalize()

**Explanation:**

* .str.strip() removes leading/trailing spaces.
* .str.lower() or .str.title() standardizes capitalization.
* .replace() can be used to map variants to consistent values.
* This makes categories consistent for analysis, grouping, and filtering.

### 3. Convert columns to correct data types

### Analysis

* Some columns are stored as strings but should be numeric or dates (e.g., year as float, age as string, date as string).
* Dates are in multiple formats (e.g., “16th August 2025”, “1900-1905”, “Before 1903”).

### Code & Explanation

Python

# Convert 'year' to integer, handle missing/invalid as NaN

df['year'] = pd.to\_numeric(df['year'], errors='coerce').astype('Int64')

# Convert 'age' to integer, set invalid entries ('?', 'Not stated') as NaN

df['age'] = pd.to\_numeric(df['age'], errors='coerce').astype('Int64')

# Convert 'date' to datetime (custom parsing required for inconsistent formats)

# Try to parse common date formats, fallback to NaT

df['date\_parsed'] = pd.to\_datetime(df['date'], errors='coerce', infer\_datetime\_format=True)

**Explanation:**

* pd.to\_numeric(..., errors='coerce') converts values to numbers, setting invalid values to NaN.
* .astype('Int64') ensures the column is integer type and can hold NaNs.
* pd.to\_datetime(..., errors='coerce') tries to convert strings to datetime, setting failures to NaT (Not a Time).
* For very inconsistent date formats, need a more advanced parser (e.g., using dateutil.parser or custom regex).

## Summary Table

| **Step** | **Action** | **Example Code** | **Explanation** |
| --- | --- | --- | --- |
| 1 | Clean columns | df.columns = [col.strip().lower().replace(' ', '\_') for col in df.columns] | Standardizes column names for coding |
| 1 | Drop empties | df.drop(columns=[...], errors='ignore') | Removes columns with little/no info |
| 2 | Standardize categories | df['sex'] = df['sex'].str.strip().str.lower().replace({'male': 'm', 'female': 'f'}) | Makes categories consistent |
| 3 | Convert types | df['year'] = pd.to\_numeric(df['year'], errors='coerce').astype('Int64') | Ensures year is numeric and handles missing |
| 3 | Parse dates | df['date\_parsed'] = pd.to\_datetime(df['date'], errors='coerce', infer\_datetime\_format=True) | Converts string dates to datetime |

### Tip:

After each cleaning step, always check results with df.info(), df.head(), or df.describe() to confirm correctness.