Analysis of Global Terrorism Incidents

The Global Terrorism Database (GTD) is a comprehensive dataset containing information on terrorist incidents worldwide. This report aims to analyze the dataset to identify trends, patterns, and insights related to global terrorism.

1-Data Acquisition and Preprocessing:

The dataset was loaded into a Pandas DataFrame and explored to understand its structure and features. Missing values were handled, and data cleaning was performed to ensure accuracy in analysis.

jupyter Untitled1 Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

Markdown JupyterLab Python 3 (ipykernel)

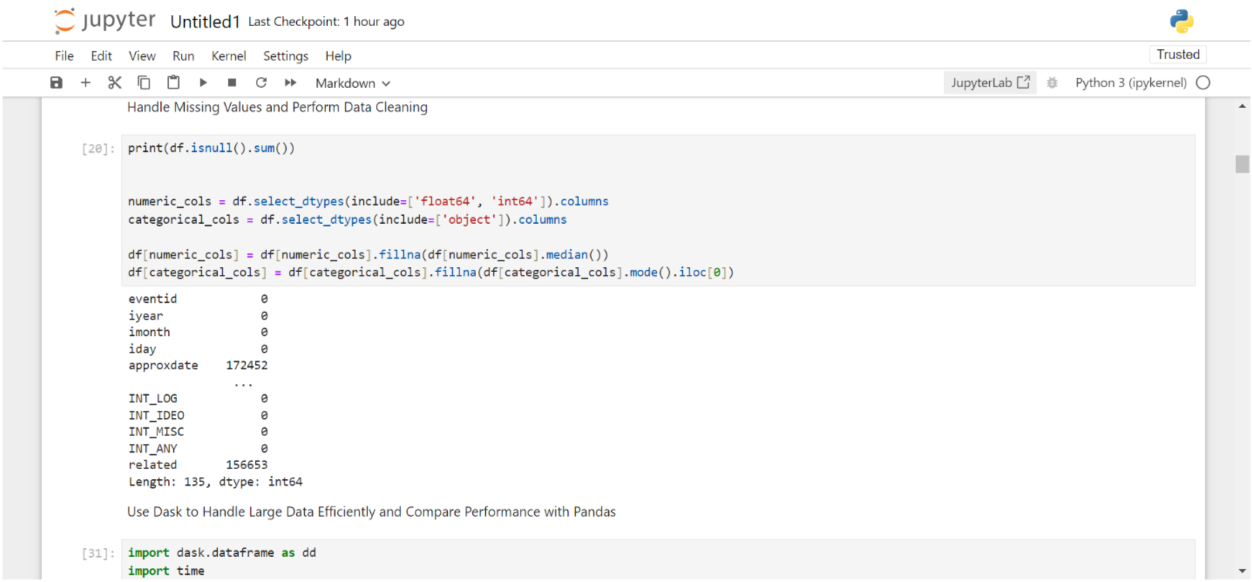
[1]:
```
import pandas as pd
```

[9]:
```
df = pd.read_csv('globalterrorismdb_0718dist.csv', encoding='latin1', low_memory=False)
```

Explore the Dataset to Understand its Structure and Features:

[11]:
```
print(df.head())
```

```
        eventid  iyear  imonth  iday approxdate  extended resolution  country  \
0  197000000001   1970       7     2        NaN         0        NaN       58
1  197000000002   1970       0     0        NaN         0        NaN      130
2  197001000001   1970       1     0        NaN         0        NaN      160
3  197001000002   1970       1     0        NaN         0        NaN       78
4  197001000003   1970       1     0        NaN         0        NaN      101

          country_txt  region  ... addnotes scite1 scite2  scite3  dbsource  \
0  Dominican Republic       2  ...      NaN    NaN    NaN     NaN      PGIS
1              Mexico       1  ...      NaN    NaN    NaN     NaN      PGIS
2         Philippines       5  ...      NaN    NaN    NaN     NaN      PGIS
3              Greece       8  ...      NaN    NaN    NaN     NaN      PGIS
4               Japan       4  ...      NaN    NaN    NaN     NaN      PGIS

  INT_LOG  INT_IDEO INT_MISC INT_ANY  related
0       0         0        0       0      NaN
1       0         1        1       1      NaN
2      -9        -9        1       1      NaN
3      -9        -9        1       1      NaN
4      -9        -9        1       1      NaN
```



jupyter Untitled1 Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

Markdown JupyterLab Python 3 (ipykernel)

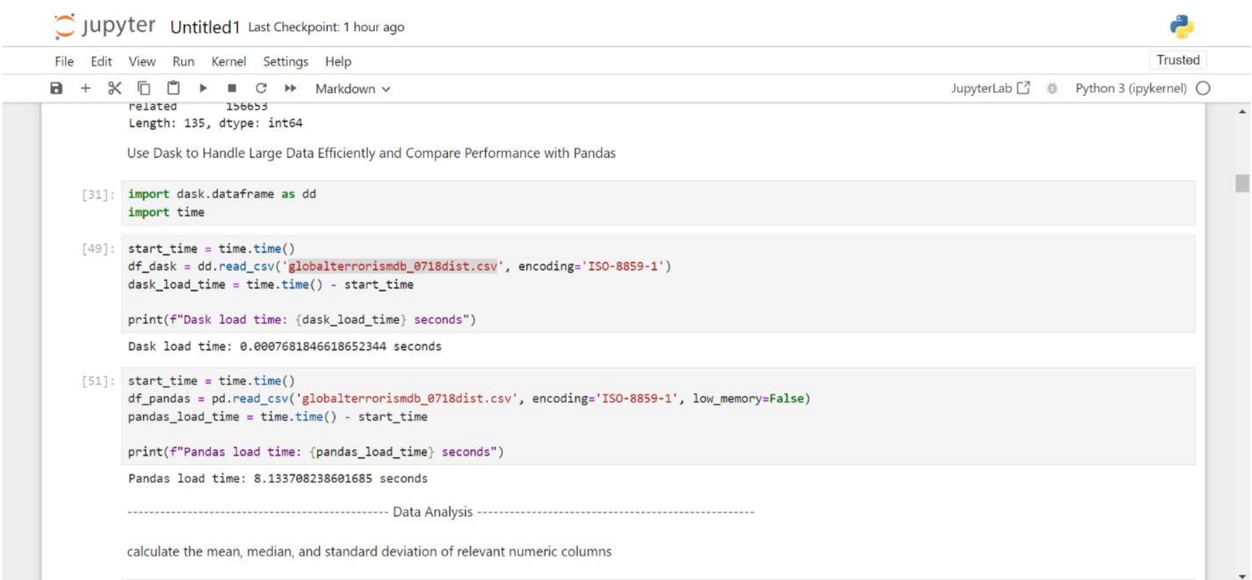
Handle Missing Values and Perform Data Cleaning

[20]:
```
print(df.isnull().sum())


numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = df.select_dtypes(include=['object']).columns

df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].median())
df[categorical_cols] = df[categorical_cols].fillna(df[categorical_cols].mode().iloc[0])
```

```
eventid            0
iyear              0
imonth             0
iday               0
approxdate    172452
               ...
INT_LOG            0
INT_IDEO           0
INT_MISC           0
INT_ANY            0
related       156653
Length: 135, dtype: int64
```

Use Dask to Handle Large Data Efficiently and Compare Performance with Pandas

[31]:
```
import dask.dataframe as dd
import time
```



jupyter Untitled1 Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

Markdown JupyterLab Python 3 (ipykernel)

```
related       156653
Length: 135, dtype: int64
```

Use Dask to Handle Large Data Efficiently and Compare Performance with Pandas

[31]:
```
import dask.dataframe as dd
import time
```

[49]:
```
start_time = time.time()
df_dask = dd.read_csv('globalterrorismdb_0718dist.csv', encoding='ISO-8859-1')
dask_load_time = time.time() - start_time

print(f"Dask load time: {dask_load_time} seconds")
```

```
Dask load time: 0.0007681846618652344 seconds
```

[51]:
```
start_time = time.time()
df_pandas = pd.read_csv('globalterrorismdb_0718dist.csv', encoding='ISO-8859-1', low_memory=False)
pandas_load_time = time.time() - start_time

print(f"Pandas load time: {pandas_load_time} seconds")
```

```
Pandas load time: 8.133708238601685 seconds
```

-- Data Analysis --

calculate the mean, median, and standard deviation of relevant numeric columns

2-Data Analysis :

Basic Statistical Analysis Using Numpy, we calculated the mean, median, and standard deviation for relevant numeric columns. The most frequent values in categorical columns were also identified.

Detailed Analysis Using Pandas : Number of Attacks Per Year The number of terrorist attacks per year was calculated, revealing a significant increase in incidents over the past few decades.

Jupyter Untitled1 Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

Markdown JupyterLab Python 3 (ipykernel)

--- Data Analysis ---

calculate the mean, median, and standard deviation of relevant numeric columns

[54]:
```
import numpy as np
```

[56]:
```
mean_values = df[numeric_cols].mean()
median_values = df[numeric_cols].median()
std_dev_values = df[numeric_cols].std()
```

[58]:
```
print("Mean values:\n", mean_values)
print("Median values:\n", median_values)
print("Standard deviation values:\n", std_dev_values)
```

```
Mean values:
 eventid      2.002705e+11
iyear        2.002639e+03
imonth       6.467277e+00
iday         1.550564e+01
extended     4.534622e-02
                 ...
nreleased   -1.661007e+00
INT_LOG     -4.543731e+00
INT_IDEO    -4.464398e+00
INT_MISC     9.000996e-02
INT_ANY     -3.945952e+00
Length: 77, dtype: float64
```

Jupyter Untitled1 Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

Markdown JupyterLab Python 3 (ipykernel)

```
Name: 0, dtype: object
```

Use Pandas to Group Data and Calculate Aggregate Statistics

[63]:
```
# Group by year and calculate the number of attacks per year
attacks_per_year = df.groupby('iyear').size()
print("Attacks per year:\n", attacks_per_year)
```

```
Attacks per year:
 iyear
1970     651
1971     471
1972     568
1973     473
1974     581
1975     740
1976     923
1977    1319
1978    1526
1979    2662
1980    2662
1981    2586
1982    2544
1983    2870
1984    3495
1985    2915
1986    2860
1987    3183
1988    3721
```

jupyter Untitled1 Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

Markdown JupyterLab Python 3 (ipykernel)

[65]:
```python
# Group by region and calculate the number of attacks per region
attacks_per_region = df.groupby('region_txt').size()
print("Attacks per region:\n", attacks_per_region)
```

```
Attacks per region:
 region_txt
Australasia & Oceania            282
Central America & Caribbean    10344
Central Asia                     563
East Asia                        802
Eastern Europe                  5144
Middle East & North Africa     50474
North America                   3456
South America                  18978
South Asia                     44974
Southeast Asia                 12485
Sub-Saharan Africa             17550
Western Europe                 16639
dtype: int64
```

[67]:
```python
attacks_per_attack_type = df.groupby('attacktype1_txt').size()
print("Attacks per attack type:\n", attacks_per_attack_type)
```

```
Attacks per attack type:
 attacktype1_txt
Armed Assault                          42669
Assassination                          19312
Bombing/Explosion                      88255
Facility/Infrastructure Attack         10356
Hijacking                                659
```


jupyter Untitled1 Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

Markdown JupyterLab Python 3 (ipykernel)

[65]:
```python
# Group by region and calculate the number of attacks per region
attacks_per_region = df.groupby('region_txt').size()
print("Attacks per region:\n", attacks_per_region)
```

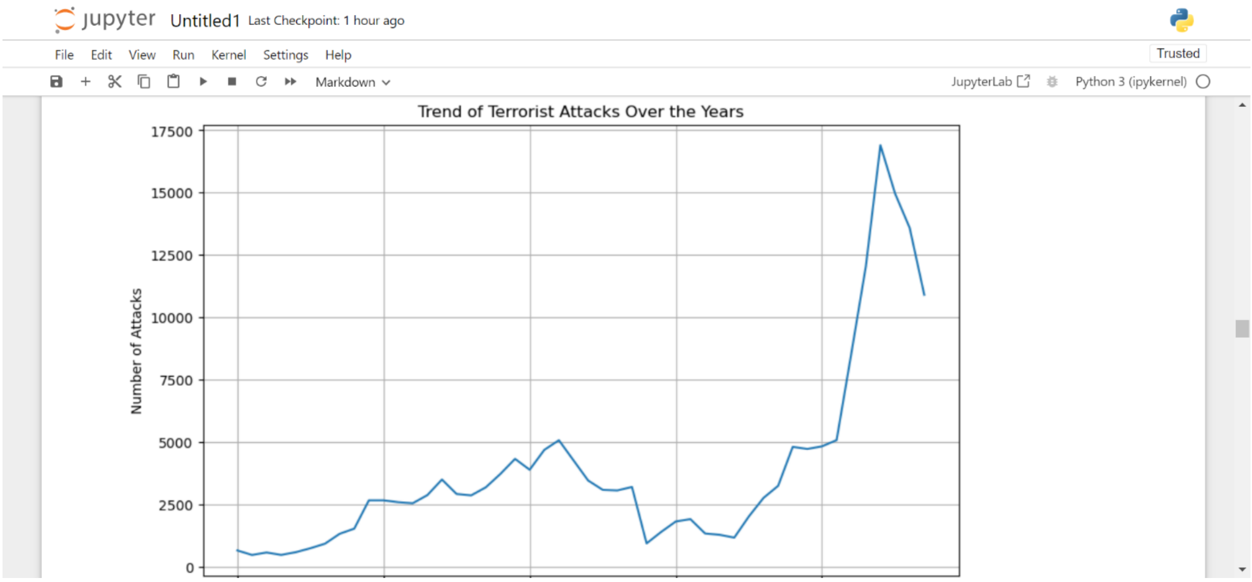
```
Attacks per region:
 region_txt
Australasia & Oceania            282
Central America & Caribbean    10344
Central Asia                     563
East Asia                        802
Eastern Europe                  5144
Middle East & North Africa     50474
North America                   3456
South America                  18978
South Asia                     44974
Southeast Asia                 12485
Sub-Saharan Africa             17550
Western Europe                 16639
dtype: int64
```

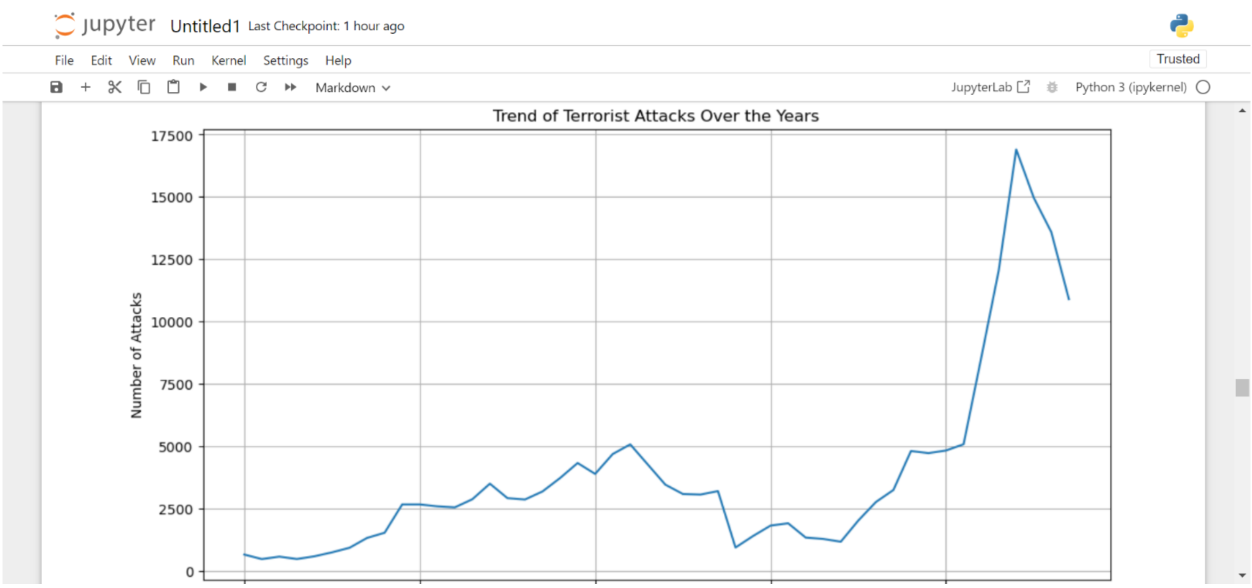
[67]:
```python
attacks_per_attack_type = df.groupby('attacktype1_txt').size()
print("Attacks per attack type:\n", attacks_per_attack_type)
```

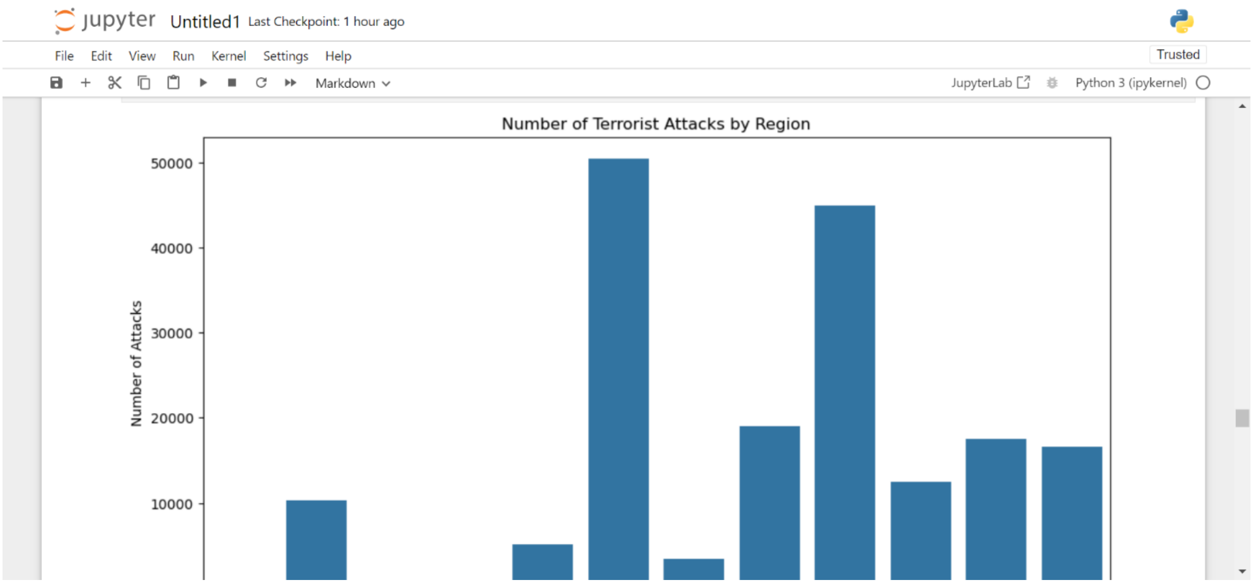
```
Attacks per attack type:
 attacktype1_txt
Armed Assault                          42669
Assassination                          19312
Bombing/Explosion                      88255
Facility/Infrastructure Attack         10356
Hijacking                                659
```

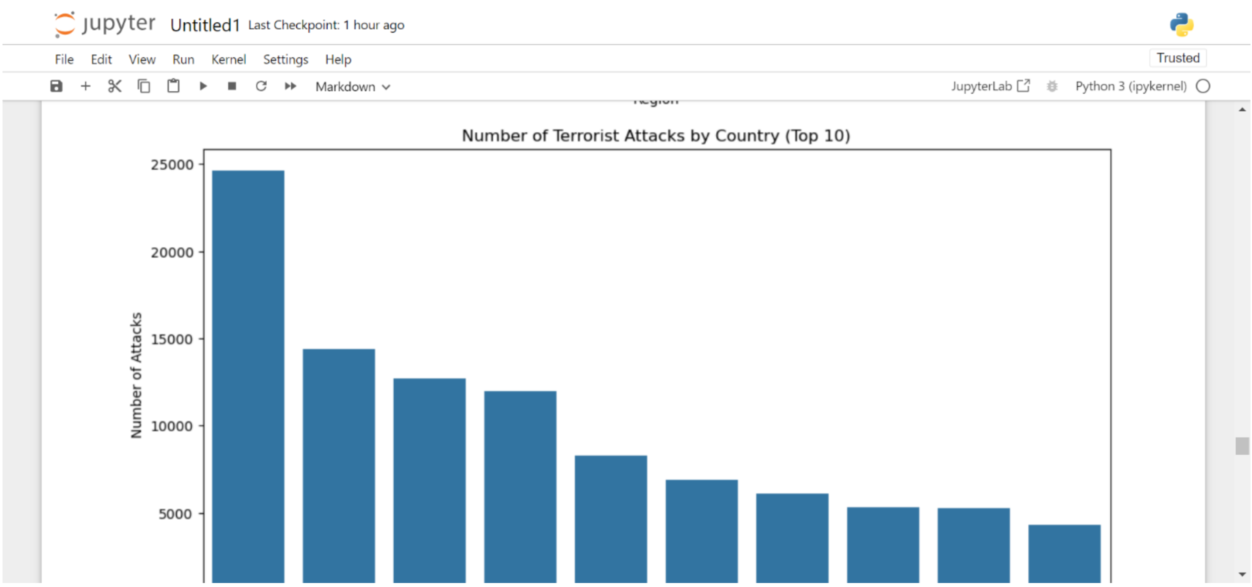
Data Visualizations :

Correlation Between Features A heatmap was created to visualize the correlation between different features in the dataset.



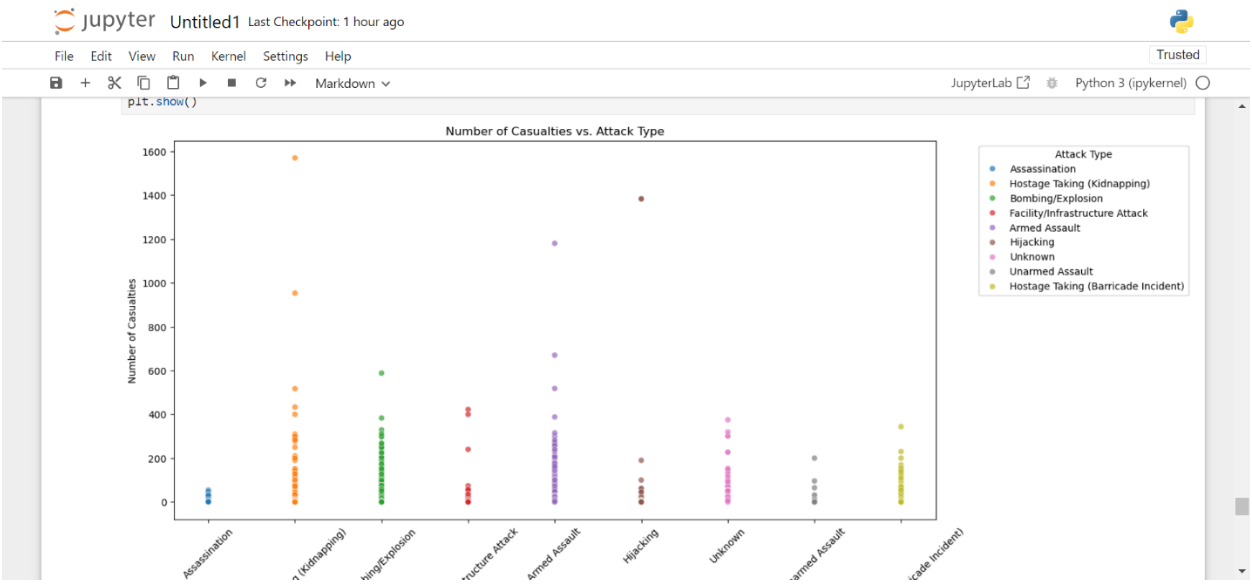






Casualties vs. Attack Type :

A scatter plot was used to show the relationship between the number of casualties and the type of attack.



Performance Comparison with Dask :

Dask was used to perform similar operations on the dataset, demonstrating its efficiency in handling large data. Comparisons between Pandas and Dask were made in terms of time and memory usage.

jupyter Untitled1 Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help — Trusted

Code — JupyterLab — Python 3 (ipykernel)

Load the Dataset Using Dask

[119]:
```
import pandas as pd
import dask.dataframe as dd
import time
```

[121]:
```
file_path = 'globalterrorismdb_0718dist.csv'
df = pd.read_csv(file_path, encoding='ISO-8859-1', low_memory=False)
```

[123]:
```
dask_df = dd.read_csv(file_path, encoding='ISO-8859-1')
```

[125]:
```
start_time = time.time()
pandas_mean = df['nkill'].mean()
pandas_time = time.time() - start_time
print(f"Pandas mean calculation time: {pandas_time} seconds")
```

```
Pandas mean calculation time: 0.002473115921020508 seconds
```

[127]:
```
# Time taken for Dask operation
start_time = time.time()
dask_mean = dask_df['nkill'].mean().compute()
dask_time = time.time() - start_time
print(f"Dask mean calculation time: {dask_time} seconds")
```

```
Dask mean calculation time: 1.6403729915618896 seconds
```

[131]:
```
start_time = time.time()
```

jupyter Untitled1 Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
dask_mean = dask_df['nkill'].mean().compute()
dask_time = time.time() - start_time
print(f"Dask mean calculation time: {dask_time} seconds")
```

```
Dask mean calculation time: 1.6403729915618896 seconds
```

[131]:
```
start_time = time.time()
dask_attacks_per_year = dask_df.groupby('iyear').size().compute()
dask_time = time.time() - start_time
print(f"Dask attacks per year calculation time: {dask_time} seconds")
```

```
Dask attacks per year calculation time: 1.5819976329803467 seconds
```

[133]:
```
start_time = time.time()
dask_attacks_per_region = dask_df.groupby('region_txt').size().compute()
dask_time = time.time() - start_time
print(f"Dask attacks per region calculation time: {dask_time} seconds")
```

```
Dask attacks per region calculation time: 1.295663833618164 seconds
```

[ ]: