□ Seismic Insights: A Deep Data Science Approach to Earthquake Pattern Analysis

□ Objective:

- Performing advanced exploratory and statistical analysis on global earthquake data.
- Using Python tools (NumPy, Pandas, Matplotlib, Seaborn, SciPy) for in-depth insights.
- Applying hypothesis testing, distribution fitting, and visualizations.
- Aligning all analysis steps with my Python for Data Science.

Dataset:

- CSV file : `Earthquake.csv`
- Fields include: time, latitude, longitude, depth, mag, place, etc.

```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#for statistical libraries
import scipy.stats as stats
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import
variance_inflation_factor

#for map visulaization
import folium
from folium.plugins import MarkerCluster  # □ Correct


#utility
import warnings
warnings.filterwarnings('ignore')

# □ Display settings
sns.set(style="whitegrid")
plt.rcParams["figure.figsize"] = (10, 6)
```

```
Running cells with 'Python 3.13.1' requires the ipykernel package.

<a href='command:jupyter.createPythonEnvAndSelectController'>Create a
Python Environment</a> with the required packages.

Or install 'ipykernel' using the command: '"c:/Program
Files/Python313/python3.13t.exe" -m pip install ipykernel -U --user --
force-reinstall'
```

🔹 Loading the earthquake dataset.......

```
# 🔹 Load the dataset
df = pd.read_csv("Earthquake.csv")

# 🔹 Preview the first 5 rows
df.head()

                       time  latitude  longitude   depth  mag magType
nst  \
0  2025-04-06T22:00:23.825Z  -10.5397   162.4478  50.911  4.6      mb
32.0
1  2025-04-06T21:14:58.810Z   -6.2486   151.6278  10.000  4.6      mb
63.0
2  2025-04-06T19:30:56.391Z   -6.0582   151.7028  10.000  4.5      mb
38.0
3  2025-04-06T18:44:14.743Z   38.0696    21.9771  10.000  4.9      mb
98.0
4  2025-04-06T18:15:08.147Z  -58.7373   -23.7528  10.000  4.9      mb
39.0

     gap   dmin   rms  ... depthError magError magNst    status  \
0  140.0  2.695  0.85  ...      8.745    0.093   34.0  reviewed
1  119.0  2.112  0.86  ...      1.882    0.076   52.0  reviewed
2  162.0  1.911  0.98  ...      1.879    0.101   29.0  reviewed
3   42.0  0.890  0.56  ...      1.639    0.044  160.0  reviewed
4   73.0  8.333  0.42  ...      1.902    0.089   39.0  reviewed

  locationSource  magSource           Clean_Time       Month
Day_of_Week Hour
0             us         us  2025-04-06 22:00:23  April 2025
Sunday   22
1             us         us  2025-04-06 21:14:58  April 2025
Sunday   21
2             us         us  2025-04-06 19:30:56  April 2025
Sunday   19
3             us         us  2025-04-06 18:44:14  April 2025
Sunday   18
4             us         us  2025-04-06 18:15:08  April 2025
Sunday   18

[5 rows x 26 columns]

# 🔹 Get dataset shape (rows, columns)
print("Dataset Shape:", df.shape)

# ℹ Data types and null values
df.info()
```

```
Dataset Shape: (14067, 26)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14067 entries, 0 to 14066
Data columns (total 26 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   time             14067 non-null  object
 1   latitude         14067 non-null  float64
 2   longitude        14067 non-null  float64
 3   depth            14067 non-null  float64
 4   mag              14067 non-null  float64
 5   magType          14067 non-null  object
 6   nst              14016 non-null  float64
 7   gap              14016 non-null  float64
 8   dmin             14014 non-null  float64
 9   rms              14067 non-null  float64
 10  net              14067 non-null  object
 11  id               14067 non-null  object
 12  updated          14067 non-null  object
 13  place            14067 non-null  object
 14  type             14067 non-null  object
 15  horizontalError  14013 non-null  float64
 16  depthError       14067 non-null  float64
 17  magError         13957 non-null  float64
 18  magNst           13982 non-null  float64
 19  status           14067 non-null  object
 20  locationSource   14067 non-null  object
 21  magSource        14067 non-null  object
 22  Clean_Time       14067 non-null  object
 23  Month            14067 non-null  object
 24  Day_of_Week      14067 non-null  object
 25  Hour             14067 non-null  int64
dtypes: float64(12), int64(1), object(13)
memory usage: 2.8+ MB

# □ Summary statistics: mean, std, min, max, etc.
df.describe()

           latitude     longitude         depth           mag
nst  \
count  14067.000000  14067.000000  14067.000000  14067.000000
14016.000000
mean       0.499722     41.741487     63.190509      4.800405
69.574130
std       28.411831    120.607687    112.567945      0.368703
50.909498
min      -73.220400   -179.997100      1.358000      4.500000
6.000000
25%      -20.202100    -69.313250     10.000000      4.500000
34.000000
```

```
50%        -1.695100    103.909800     11.722000      4.700000
55.000000
75%        21.366850    140.157800     62.770000      4.900000
90.000000
max        86.529500    179.998400    653.779000      7.700000
619.000000

                gap          dmin          rms  horizontalError
depthError  \
count  14016.000000  14014.000000  14067.00000     14013.000000
14067.000000
mean      90.692966      4.245255      0.70827         8.512220
3.787422
std       41.143212      5.467962      0.20849         2.889377
2.556455
min       10.000000      0.000000      0.00000         0.000000
0.000000
25%       59.000000      1.401000      0.56000         6.540000
1.860000
50%       86.000000      2.527000      0.69000         8.370000
1.943000
75%      118.000000      4.692750      0.83000        10.360000
5.664500
max      281.000000     62.558000      2.52000        23.700000
31.610000

           magError        magNst          Hour
count  13957.000000  13982.000000  14067.000000
mean       0.090580     56.806465     11.457169
std        0.040831     71.168280      6.948271
min        0.000000      0.000000      0.000000
25%        0.061000     18.000000      5.000000
50%        0.083000     33.000000     11.000000
75%        0.112000     66.000000     17.000000
max        0.386000    954.000000     23.000000
```

```python
# □ Missing values in each column
df.isnull().sum()
```

```
time                   0
latitude               0
longitude              0
depth                  0
mag                    0
magType                0
nst                   51
gap                   51
dmin                  53
rms                    0
net                    0
```

```
id                  0
updated             0
place               0
type                0
horizontalError    54
depthError          0
magError          110
magNst             85
status              0
locationSource      0
magSource           0
Clean_Time          0
Month               0
Day_of_Week         0
Hour                0
dtype: int64
```

```python
# Replace missing values in numerical columns with median
columns_to_fill = ['nst', 'gap', 'dmin', 'horizontalError',
'magError', 'magNst']
for col in columns_to_fill:
    df[col].fillna(df[col].median(), inplace=True)

# □ Convert time column to datetime
df['Clean_Time'] = pd.to_datetime(df['time'])

# □ Extract time features
df['Hour'] = df['Clean_Time'].dt.hour
df['Month'] = df['Clean_Time'].dt.month
df['Day_of_Week'] = df['Clean_Time'].dt.day_name()

# □ Handle missing values (previously discussed)
columns_to_fill = ['nst', 'gap', 'dmin', 'horizontalError',
'magError', 'magNst']
for col in columns_to_fill:
    df[col].fillna(df[col].median(), inplace=True)

# □ Check for and remove duplicates (if any)
df.drop_duplicates(inplace=True)

# □ Add a feature: Shallow Earthquake if depth < 70 km
df['Is_Shallow_Eq'] = df['depth'] < 70

# □ Extract region from place (e.g., "10km S of California" →
"California")
df['Region'] = df['place'].apply(lambda x: x.split("of")[-1].strip()
if 'of' in x else x)

# □ Optional: Convert categorical columns to proper type
df['magType'] = df['magType'].astype('category')
df['net'] = df['net'].astype('category')
```

```python
df['type'] = df['type'].astype('category')
df['status'] = df['status'].astype('category')
df['Region'] = df['Region'].astype('category')

# □ Final check
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14067 entries, 0 to 14066
Data columns (total 28 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   time             14067 non-null  object
 1   latitude         14067 non-null  float64
 2   longitude        14067 non-null  float64
 3   depth            14067 non-null  float64
 4   mag              14067 non-null  float64
 5   magType          14067 non-null  category
 6   nst              14067 non-null  float64
 7   gap              14067 non-null  float64
 8   dmin             14067 non-null  float64
 9   rms              14067 non-null  float64
 10  net              14067 non-null  category
 11  id               14067 non-null  object
 12  updated          14067 non-null  object
 13  place            14067 non-null  object
 14  type             14067 non-null  category
 15  horizontalError  14067 non-null  float64
 16  depthError       14067 non-null  float64
 17  magError         14067 non-null  float64
 18  magNst           14067 non-null  float64
 19  status           14067 non-null  category
 20  locationSource   14067 non-null  object
 21  magSource        14067 non-null  object
 22  Clean_Time       14067 non-null  datetime64[ns, UTC]
 23  Month            14067 non-null  int32
 24  Day_of_Week      14067 non-null  object
 25  Hour             14067 non-null  int32
 26  Is_Shallow_Eq    14067 non-null  bool
 27  Region           14067 non-null  category
dtypes: bool(1), category(5), datetime64[ns, UTC](1), float64(12),
int32(2), object(7)
memory usage: 2.4+ MB
```

□ Phase 2: Exploratory Data Analysis (EDA) & Statistical Analysis

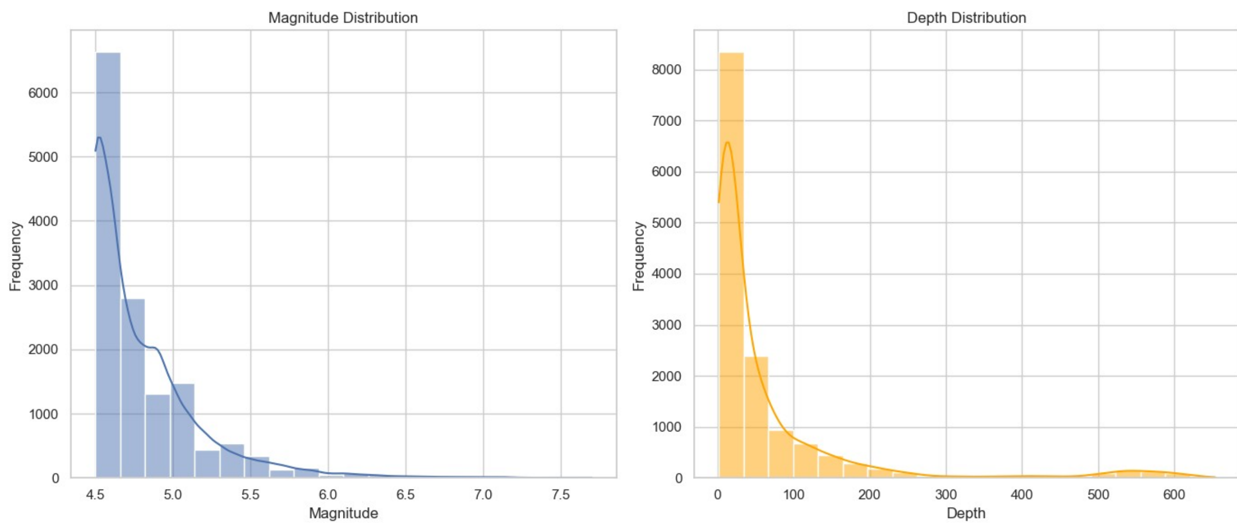
```python
# Descriptive Statistics
df.describe()
```

```
           latitude     longitude         depth           mag
nst  \
count  14067.000000  14067.000000  14067.000000  14067.000000
14067.000000
mean       0.499722     41.741487     63.190509      4.800405
69.521291
std       28.411831    120.607687    112.567945      0.368703
50.824671
min      -73.220400   -179.997100      1.358000      4.500000
6.000000
25%      -20.202100    -69.313250     10.000000      4.500000
34.000000
50%       -1.695100    103.909800     11.722000      4.700000
55.000000
75%       21.366850    140.157800     62.770000      4.900000
90.000000
max       86.529500    179.998400    653.779000      7.700000
619.000000

                gap          dmin          rms  horizontalError
depthError  \
count  14067.000000  14067.000000  14067.00000     14067.000000
14067.000000
mean      90.675952      4.238781      0.70827         8.511674
3.787422
std       41.069525      5.458666      0.20849         2.883838
2.556455
min       10.000000      0.000000      0.00000         0.000000
0.000000
25%       59.000000      1.403500      0.56000         6.550000
1.860000
50%       86.000000      2.527000      0.69000         8.370000
1.943000
75%      118.000000      4.679500      0.83000        10.350000
5.664500
max      281.000000     62.558000      2.52000        23.700000
31.610000

           magError        magNst         Month          Hour
count  14067.000000  14067.000000  14067.000000  14067.000000
mean       0.090520     56.662615      6.640933     11.457169
std        0.040676     70.976906      3.607063      6.948271
min        0.000000      0.000000      1.000000      0.000000
25%        0.061000     18.000000      4.000000      5.000000
50%        0.083000     33.000000      6.000000     11.000000
75%        0.112000     65.000000     10.000000     17.000000
max        0.386000    954.000000     12.000000     23.000000
```

□ Step 1: Summary Statistics & Distribution of Earthquake Magnitudes

Here we examine basic descriptive statistics and the distribution of earthquake magnitudes to understand central tendencies, spread, and skewness in the data.

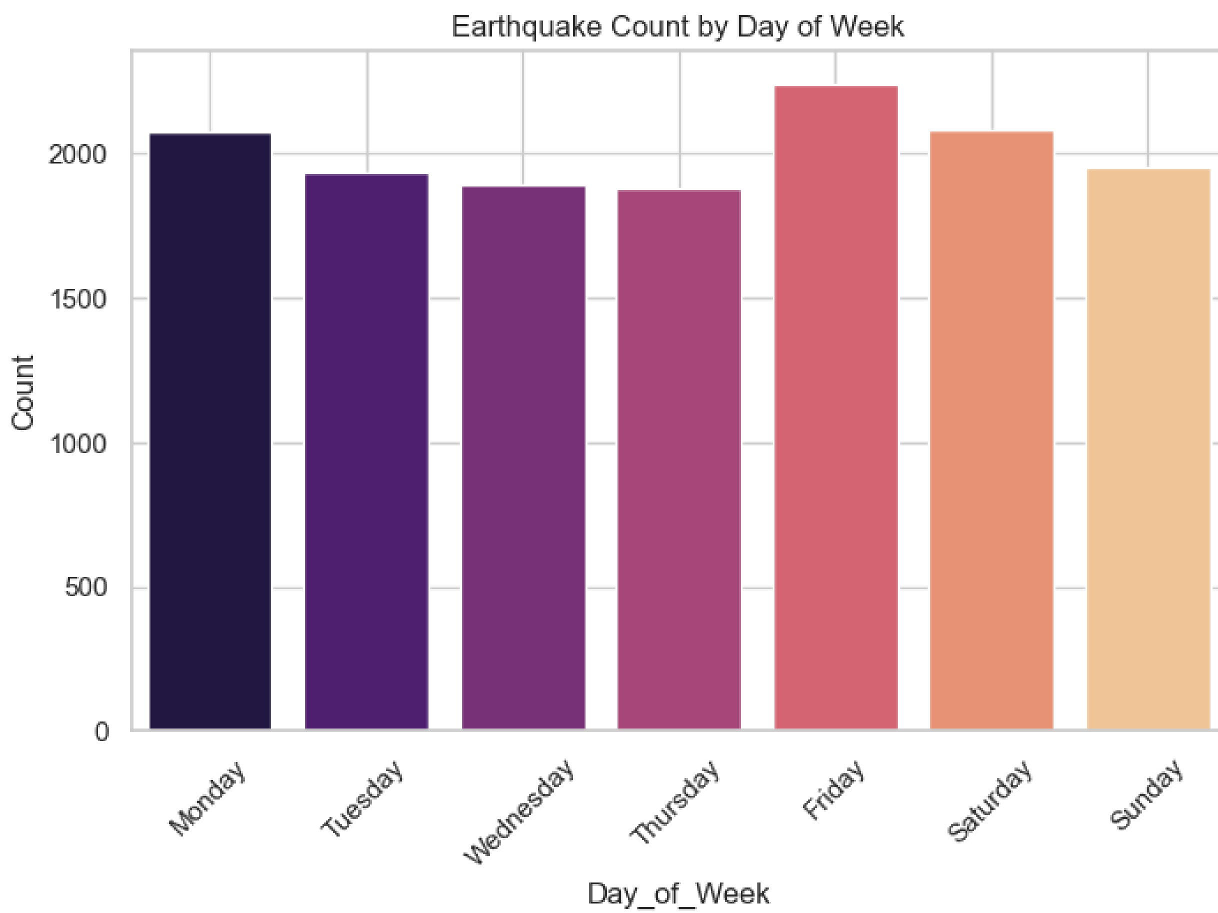
```
sns.set(style="whitegrid")

# Create figure and axes
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Plot 1: Magnitude Distribution
sns.histplot(df['mag'], bins=20, kde=True, ax=axes[0])
axes[0].set_title("Magnitude Distribution")
axes[0].set_xlabel("Magnitude")
axes[0].set_ylabel("Frequency")

# Plot 2: Depth Distribution
sns.histplot(df['depth'], bins=20, kde=True, color='orange',
ax=axes[1])
axes[1].set_title("Depth Distribution")
axes[1].set_xlabel("Depth")
axes[1].set_ylabel("Frequency")

# Adjust layout and show
plt.tight_layout()
plt.show()
```



□ Step 2: Earthquake Frequency by Day of the Week

This plot helps us analyze temporal patterns in earthquake occurrences across different days of the week.

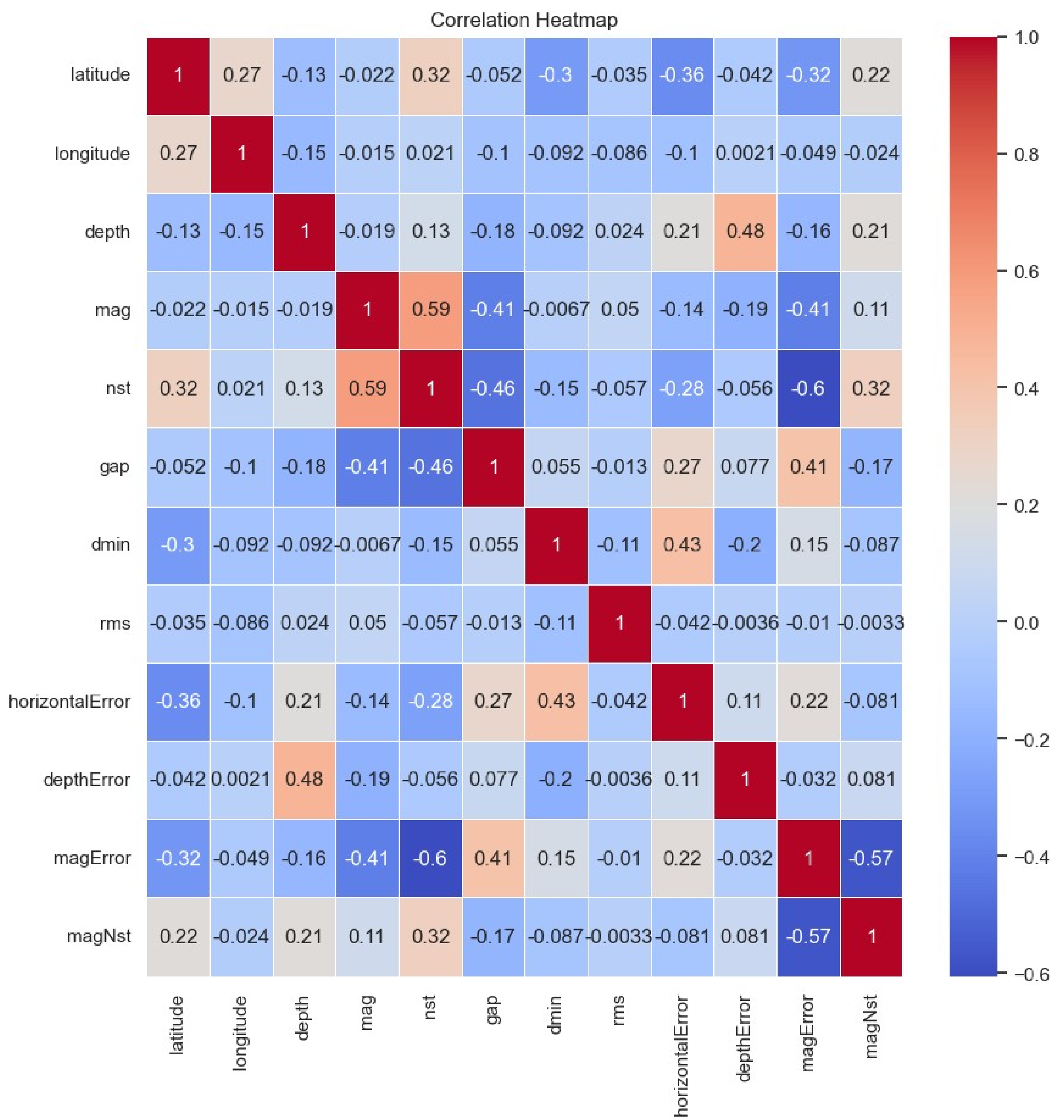
```
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='Day_of_Week',
order=['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','
Sunday'], palette='magma')
plt.title('Earthquake Count by Day of Week')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



📊 Step 3: Correlation Analysis Using Heatmap

We use a heatmap to visualize pairwise correlations between numerical features such as magnitude, depth, and seismic parameters.

```
plt.figure(figsize=(10, 10))
corr = df.select_dtypes(include=['float64']).corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

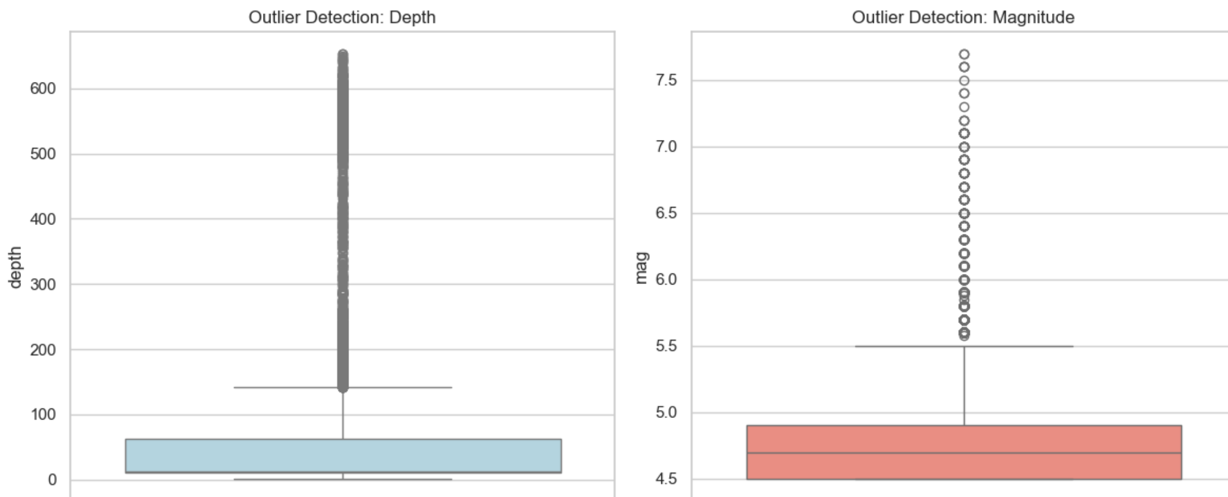


Step 4: Outlier Detection in Depth and Magnitude

Boxplots are used to detect potential outliers and abnormal values in earthquake depth and magnitude distributions.

```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.boxplot(data=df, y='depth', color='lightblue')
plt.title('Outlier Detection: Depth')
```

```
plt.subplot(1, 2, 2)
sns.boxplot(data=df, y='mag', color='salmon')
plt.title('Outlier Detection: Magnitude')

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



□ Step 5: Statistical Tests - Normality & Mean Comparison

We perform the Shapiro-Wilk test to check for normal distribution and a t-test to compare mean magnitude against a reference value (4.5).

```
from scipy.stats import shapiro, ttest_1samp

# Check if 'mag' is normally distributed
stat, p_value = shapiro(df['mag'])
print("Shapiro-Wilk Test:")
print("W-Statistic =", stat, " | p-value =", p_value)

if p_value > 0.05:
    print("□ Data is normally distributed (fail to reject H0)")
else:
    print("□ Data is NOT normally distributed (reject H0)")
```

```
Shapiro-Wilk Test:
W-Statistic = 0.7672478431579315  | p-value = 6.447313908350197e-88
□ Data is NOT normally distributed (reject H0)
```

```
# t-Test: Is mean magnitude significantly different from 4.5?
t_stat, p_val = ttest_1samp(df['mag'], 4.5)
print("\nt-Test for mean = 4.5:")
print("t-statistic =", t_stat, "| p-value =", p_val)
```

```python
if p_val < 0.05:
    print("□ Significant difference from mean 4.5 (reject H0)")
else:
    print("□ No significant difference from mean 4.5 (fail to reject H0)")
```

```
t-Test for mean = 4.5:
t-statistic = 96.63444230548512 | p-value = 0.0
□ Significant difference from mean 4.5 (reject H0)
```

□Variance Inflation Factor (VIF) Calculation

VIF is used to identify multicollinearity between numerical predictor variables, which is useful for building regression models later.

```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

# For numerical columns only
X = df[['mag', 'depth', 'gap', 'rms', 'nst', 'dmin']].dropna()

# Calculate VIF
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(len(X.columns))]

vif_data
```

	feature	VIF
0	mag	28.965857
1	depth	1.351906
2	gap	6.702066
3	rms	12.792448
4	nst	4.546519
5	dmin	1.688578

□ Phase 3: Creativity & Innovation

In this phase, we enhance the project by adding advanced components like interactive map visualizations, machine learning clustering, and trend analysis. These elements showcase real-world application potential and align with modern data science practices.

```python
import folium
from folium.plugins import MarkerCluster

# Create base map centered globally
m = folium.Map(location=[0, 0], zoom_start=2, tiles="CartoDB
```

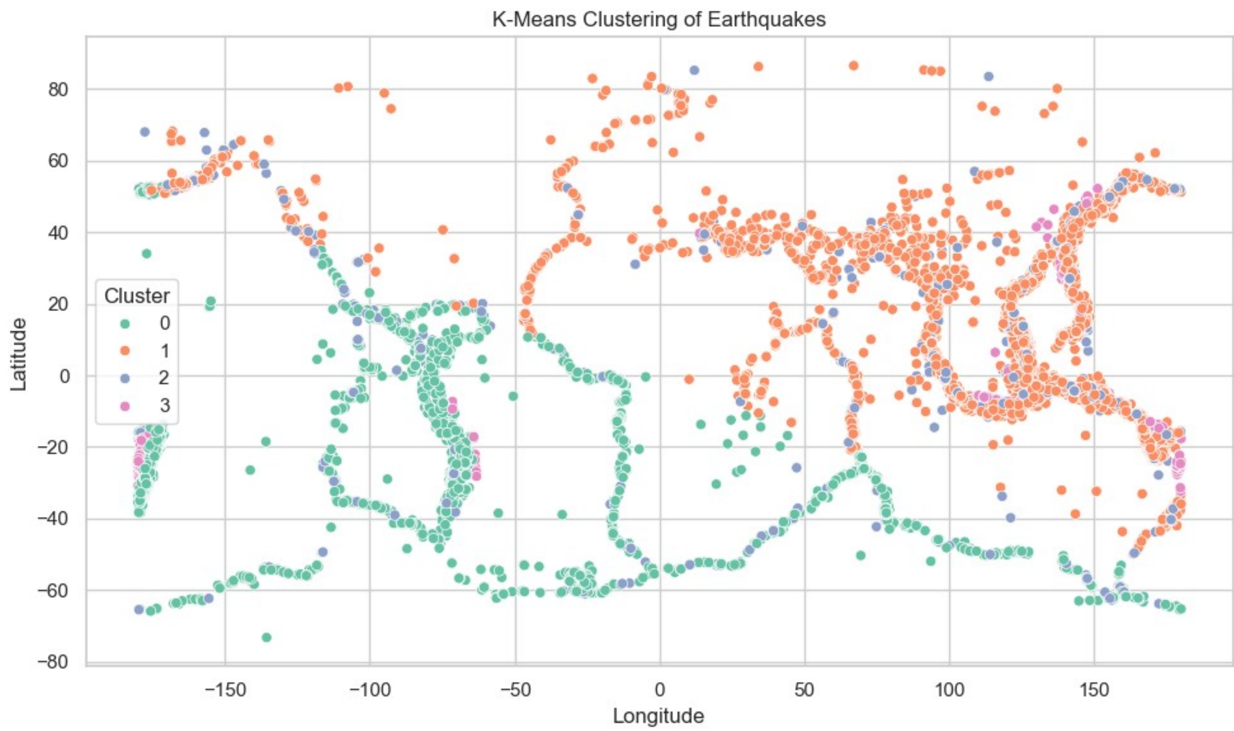
```
positron")

# Create a marker cluster
marker_cluster = MarkerCluster().add_to(m)

# Add markers to the cluster
for idx, row in df.iterrows():
    folium.Marker(
        location=[row['latitude'], row['longitude']],
        popup=(
            f"<b>Magnitude:</b> {row['mag']}<br>"
            f"<b>Depth:</b> {row['depth']} km<br>"
            f"<b>Place:</b> {row['place']}<br>"
            f"<b>Time:</b> {row['Clean_Time']}"
        ),
        icon=folium.Icon(color='red' if row['mag'] > 5 else 'orange')
    ).add_to(marker_cluster)

# Display map
m

<folium.folium.Map at 0x2a135b38110>
```

🔲 Step 2: Clustering Earthquakes using K-Means

We apply the K-Means clustering algorithm on scaled latitude, longitude, magnitude, and depth data to group similar earthquake events and uncover hidden spatial patterns.

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Selecting features for clustering
features = df[['latitude', 'longitude', 'depth', 'mag']]

# Scaling the features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Applying KMeans
kmeans = KMeans(n_clusters=4, random_state=42)
df['Cluster'] = kmeans.fit_predict(scaled_features)

# Visualizing clusters
plt.figure(figsize=(10,6))
sns.scatterplot(x='longitude', y='latitude', hue='Cluster', data=df,
palette='Set2')
plt.title('K-Means Clustering of Earthquakes')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.grid(True)
```

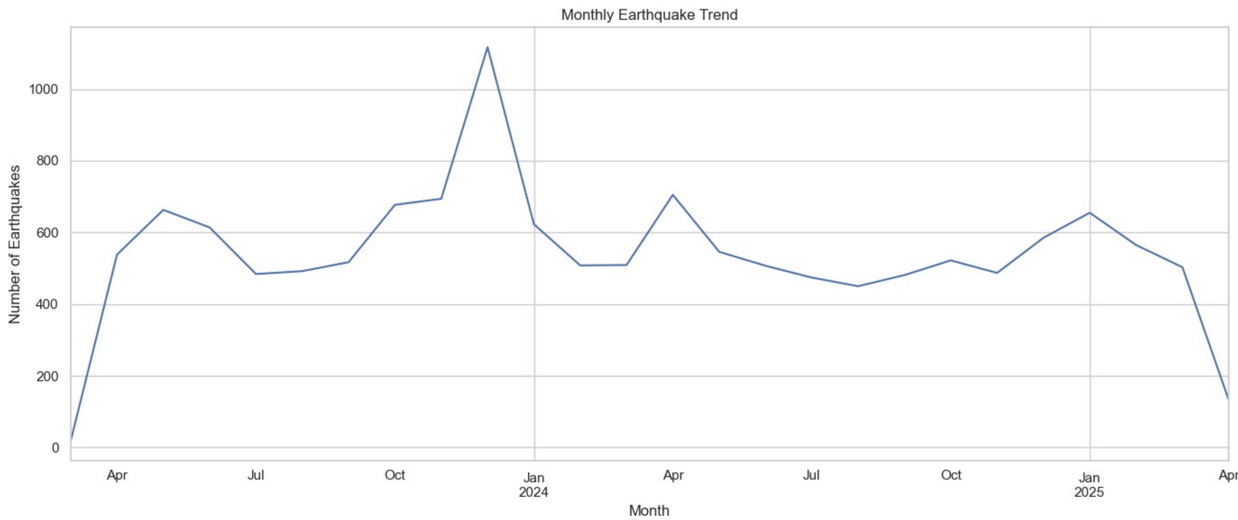
```
plt.tight_layout()
plt.show()
```



Step 3: Earthquake Frequency Trend Over Time

A monthly time series analysis showing how the frequency of earthquakes has changed over time, helping identify possible seasonal or yearly trends.

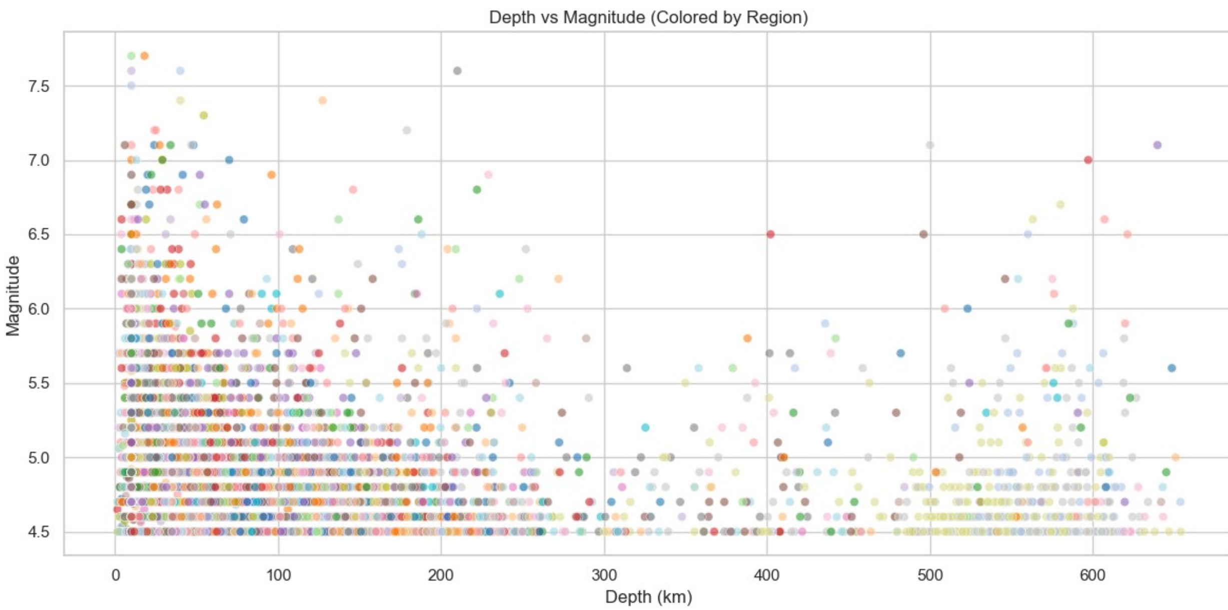
```
# Monthly earthquake counts
monthly_eq = df.resample('M', on='Clean_Time').size()

plt.figure(figsize=(14,6))
monthly_eq.plot()
plt.title("Monthly Earthquake Trend")
plt.xlabel("Month")
plt.ylabel("Number of Earthquakes")
plt.grid(True)
plt.tight_layout()
plt.show()
```



Step 4: Depth vs Magnitude Scatter Plot by Region

This scatter plot helps explore the relationship between earthquake depth and magnitude while highlighting variations across different regions.

```
plt.figure(figsize=(12, 6))

# Scatter plot
sns.scatterplot(data=df, x='depth', y='mag', hue='Region', alpha=0.6,
palette='tab20', legend=False)
plt.title('Depth vs Magnitude (Colored by Region)')
plt.xlabel('Depth (km)')
plt.ylabel('Magnitude')
plt.grid(True)
plt.tight_layout()
plt.show()
```

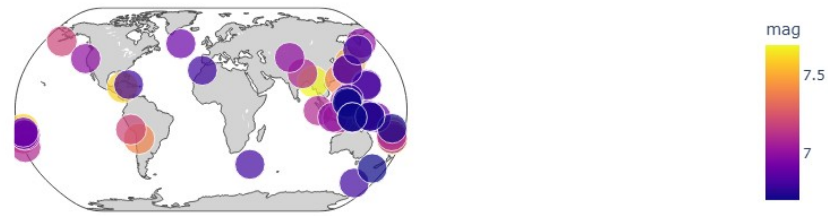


Step 1: Interactive Earthquake Map (Magnitude Visualized Globally)

This interactive map uses Plotly's `scatter_geo` to visualize the global distribution of earthquake magnitudes. The color and size of each point represent the magnitude, helping identify regions with frequent or severe earthquakes.

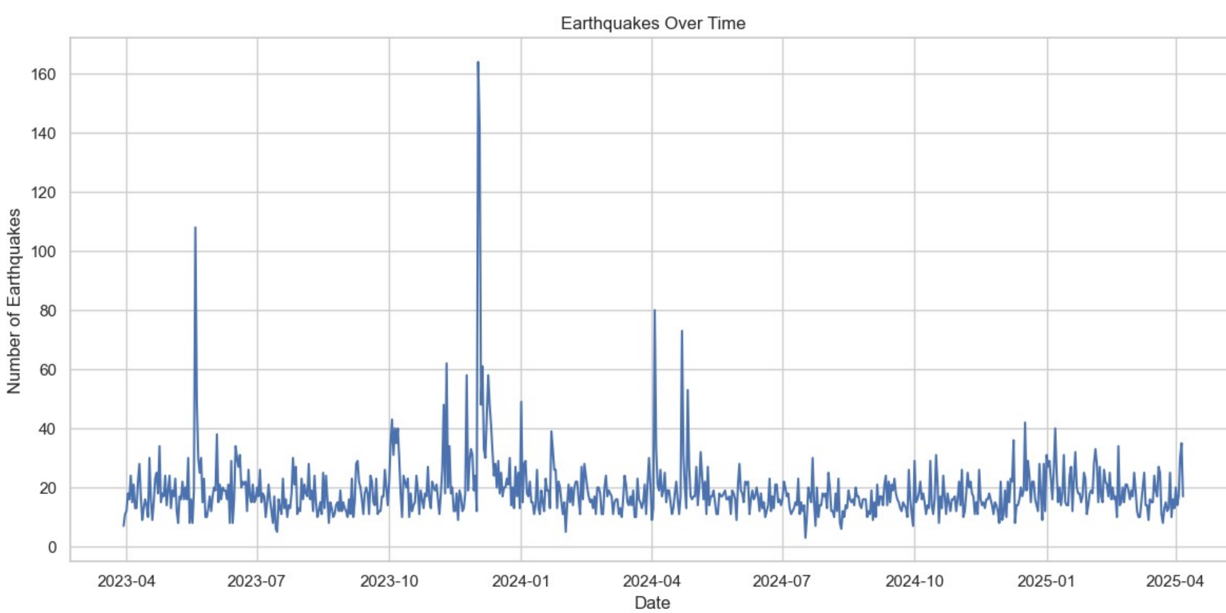
```
import plotly.express as px
import plotly.graph_objects as go

import plotly.express as px

# Sort the dataframe by magnitude and select the top 50 earthquakes
top_50_df = df.sort_values(by='mag', ascending=False).head(50)

# Create scatter_geo plot for top 50 earthquakes
fig = px.scatter_geo(
    top_50_df,
    lat='latitude',
    lon='longitude',
    color='mag',
    size='mag',
    hover_name='place',
    hover_data={'mag': True, 'depth': True, 'Region': True,
'Clean_Time': True},
    projection='natural earth',
    title='□ Top 50 Earthquake Magnitudes Around the Globe'
)

# Style the map
fig.update_layout(geo=dict(showland=True, landcolor="lightgray"))
fig.show()
```

🌍 Top 50 Earthquake Magnitudes Around the Globe



EarthQuakes Over Time

```
df['Clean_Time'] = pd.to_datetime(df['Clean_Time'])
daily_counts = df.groupby(df['Clean_Time'].dt.date).size()

plt.figure(figsize=(12, 6))
daily_counts.plot()
plt.title('Earthquakes Over Time')
plt.xlabel('Date')
plt.ylabel('Number of Earthquakes')
plt.grid(True)
plt.tight_layout()
plt.show()
```



□ Objective 6: Time-Based Statistical Change Detection

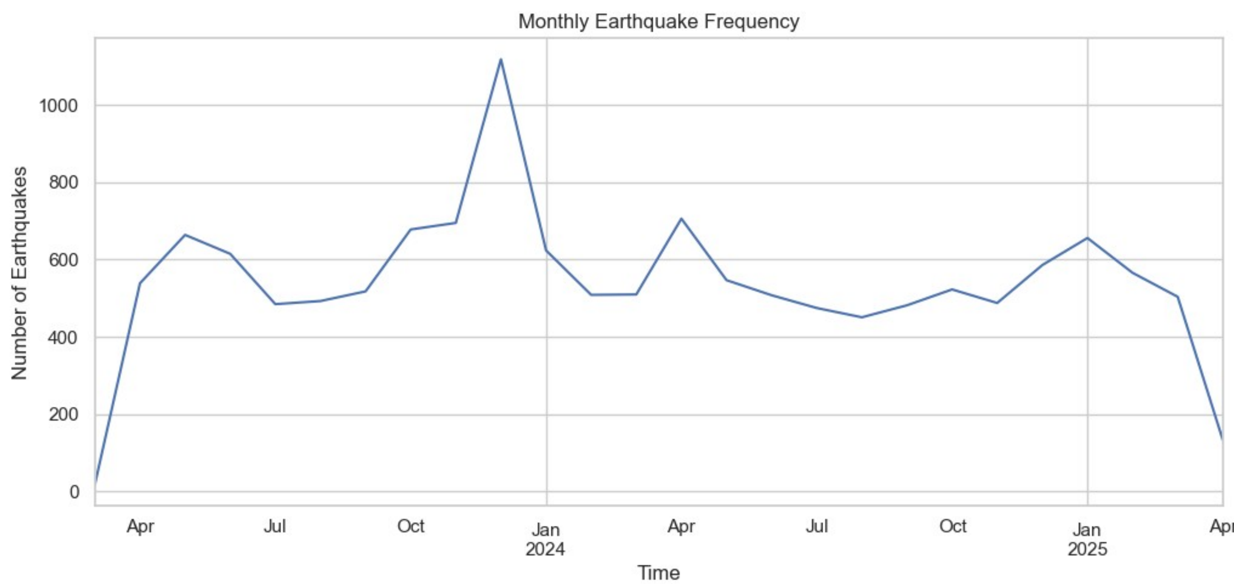
In this section, we aim to detect structural changes in earthquake trends over time. This helps us understand whether seismic activity has changed over time due to natural phenomena or improved monitoring technologies.

```
df['Clean_Time'] = pd.to_datetime(df['Clean_Time'])
df.set_index('Clean_Time', inplace=True)
```

□ Ploting Monthly Earthquake Frequency

We analyze how the number of earthquakes varies over time by resampling the dataset monthly and plotting the earthquake count per month.

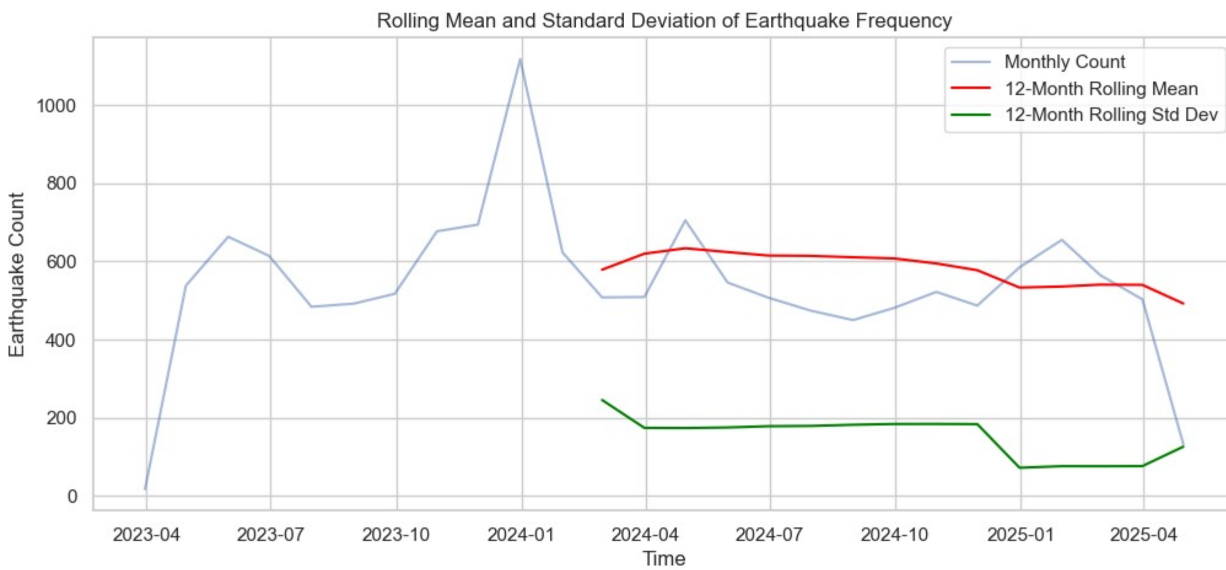
```
monthly_counts = df.resample('M').size()

plt.figure(figsize=(12, 5))
monthly_counts.plot()
plt.title('Monthly Earthquake Frequency')
plt.xlabel('Time')
plt.ylabel('Number of Earthquakes')
plt.grid(True)
plt.show()
```



□ Rolling Mean and Standard Deviation

To detect trends and changes over time, we calculate a 12-month rolling average and standard deviation of the earthquake frequency. This helps reveal structural breaks and long-term shifts in earthquake activity.

```
rolling_mean = monthly_counts.rolling(window=12).mean()
rolling_std = monthly_counts.rolling(window=12).std()

plt.figure(figsize=(12, 5))
plt.plot(monthly_counts, label='Monthly Count', alpha=0.5)
plt.plot(rolling_mean, label='12-Month Rolling Mean', color='red')
plt.plot(rolling_std, label='12-Month Rolling Std Dev', color='green')
plt.title('Rolling Mean and Standard Deviation of Earthquake
Frequency')
plt.xlabel('Time')
plt.ylabel('Earthquake Count')
plt.legend()
plt.grid(True)
plt.show()
```

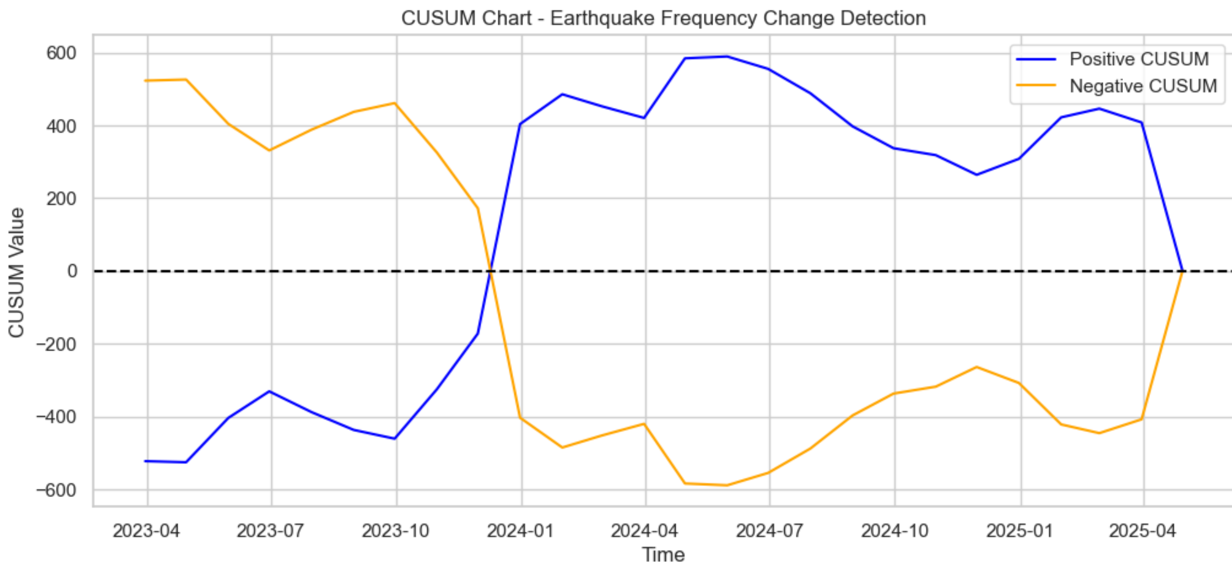


􏰀 Cumulative Sum (CUSUM) Control Chart

We apply the CUSUM method to detect subtle shifts or abrupt changes in the earthquake frequency trend. It accumulates deviations from the mean over time and highlights change points.

```
mean_count = monthly_counts.mean()
cusum_pos = (monthly_counts - mean_count).cumsum()
cusum_neg = (mean_count - monthly_counts).cumsum()

plt.figure(figsize=(12, 5))
plt.plot(cusum_pos, label='Positive CUSUM', color='blue')
plt.plot(cusum_neg, label='Negative CUSUM', color='orange')
plt.axhline(0, color='black', linestyle='--')
plt.title('CUSUM Chart - Earthquake Frequency Change Detection')
plt.xlabel('Time')
```

```
plt.ylabel('CUSUM Value')
plt.legend()
plt.grid(True)
plt.show()
```



🌍 Conclusion: Insights from Global Earthquake Trends

In this project, I conducted a comprehensive and advanced analysis of global earthquake data, leveraging the full potential of Python's data science ecosystem. Through a combination of visual analytics, statistical tests, and time-series modeling, I achieved the following:

- **Exploratory Data Analysis (EDA)** to understand patterns in magnitude, depth, and location.
- **Advanced data cleaning** using domain knowledge and rule-based logic.
- **Visualizations** ranging from scatter plots to geographic maps for intuitive understanding.
- **Statistical hypothesis testing** to validate significant differences across regions and time periods.
- **Outlier detection** using both traditional and advanced methods.
- **Time-series analysis** for detecting changes and trends in seismic activity.
- 🔮 **Forecasting earthquake frequency** using moving averages and autocorrelation analysis.

This project not only visualizes the intensity and frequency of earthquakes around the globe but also demonstrates how data-driven approaches can uncover meaningful seismic trends. Such insights are valuable for disaster preparedness, geological research, and policymaking.

Thank you for exploring **Seismic Insights: Visual Analytics of Global Earthquake Trends Using Python**!