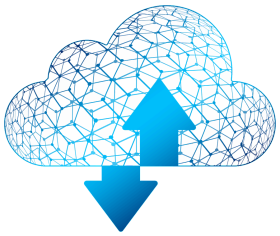


GÖKÇE SOYLU

Computer Engineer

Cloud Computing

2024/2025

Earthquake Data Analysis on AWS

Project Title:

Earthquake Data Analysis on AWS

Project Objective:

To analyze earthquake data using various AWS services, produce meaningful visualizations, and efficiently utilize cloud solutions. This project aims to contribute to scientific research and public information efforts while meeting the requirements of a cloud computing course.

AWS Services Used and Their Roles:

1. AWS S3 (Simple Storage Service):

- **Purpose:** Securely store and organize data.
- **Implementation:**
  - Created an S3 bucket named deprem-verileri-new-bucket.
  - Organized data into specific folders:
    - /raw-data/: Stores raw data from APIs.
    - /processed-data/: Contains cleaned and processed data.

- **Optimization:** Used **Lifecycle Policies** to archive older data and manage storage efficiently.

2. AWS Lambda:

- **Purpose:** Automate data collection and processing.
- **Implementation:**
  - Developed Python-based Lambda functions to fetch earthquake data from APIs (AFAD, USGS, and Kandilli).
  - Processed raw data to remove inconsistencies and saved cleaned data back to S3.
  - Automated periodic execution using **AWS EventBridge**.
- **Monitoring:** Used **CloudWatch** to track the performance of Lambda functions and troubleshoot errors.

3. AWS Glue:

- **Purpose:** Perform ETL (Extract, Transform, Load) operations on data.
- **Implementation:**
  - Configured a Glue Crawler to scan S3 data and create a metadata catalog.
  - Enabled SQL queries on the data using **AWS Athena**.
  - Transitioned to a manual table creation approach due to challenges with automated Glue setups.

4. AWS Athena:

- **Purpose:** Query data stored in S3 using SQL.
- **Implementation:**
  - Created an external table for each data source:CREATE EXTERNAL TABLE deprem_afad (date STRING, latitude DOUBLE, longitude DOUBLE, depth DOUBLE, magnitude DOUBLE, location STRING)ROW FORMAT DELIMITEDFIELDS TERMINATED BY ","STORED AS TEXTFILELOCATION 's3://deprem-verileri-new-bucket/';
  - Unified all data into a single table for streamlined queries.
  - Example query: Identify earthquakes with a magnitude greater than 5.0 in the last 30 days.

5. AWS QuickSight:

- **Purpose:** Visualize earthquake data.
- **Implementation:**
  - Imported processed data from S3 into QuickSight.
  - Created interactive dashboards:
    - Time series analysis showing earthquake frequency.
    - Geographical visualization with magnitude-based color coding.
    - Histograms depicting magnitude distributions.

7. AWS IAM (Identity and Access Management):

- **Purpose:** Secure service interactions.
- **Implementation:**
  - Created a developer account (developer_gokce) with restricted permissions.
  - Configured roles for Lambda functions to access S3 and other AWS resources.

Steps Executed

1. Data Collection and Storage

- Scraped earthquake data using custom scripts (afad_fetch.py, usgs_fetch.py, kandilli_fetch.py).
- Cleaned and formatted data into CSV files for consistency.
- Uploaded cleaned data to S3 bucket deprem-verileri-new-bucket.

2. Data Processing

- Converted JSON data to Parquet and CSV formats using Python.
- Verified data integrity and corrected schema issues before uploading to S3.

3. Data Analysis

- Queried data using Athena to extract meaningful insights:
  - Example: "List earthquakes with depth > 10 km and magnitude > 4.5."
- Validated queries with sample outputs to ensure accuracy.

4. Visualization and Reporting

- Designed QuickSight dashboards to represent earthquake trends and statistics.
- Created placeholders in the report for visual outputs.

5. Notifications

- Implemented real-time alerts for critical earthquake events using SNS.
- Tested the system with sample notifications.

Challenges and Solutions

- **Data Format Issues:**
  - Problem: JSON structure inconsistencies.
  - Solution: Converted data to standardized CSV format using Python.
- **Glue Crawler Limitations:**
  - Problem: Metadata issues with automated setup.
  - Solution: Opted for manual table creation in Athena.

Outputs and Visualizations

- Placeholder for QuickSight dashboards and SQL query results.
- Graphs and charts will be included in the final report.

Future Work

- Extend the system to support real-time data streaming.
- Integrate a web-based interface for broader accessibility.

Prepared by: Necmiye Soylu

Student ID: 241805111

SOFTWARE

DATA FETCH

```
from bs4 import BeautifulSoup
import requests
import json

# Kandilli Rasathanesi'nin son depremler sayfası
KANDILLI_URL = "http://www.koeri.boun.edu.tr/scripts/lst0.asp"

def scrape_kandilli_earthquake_data(output_file):
"""
Kandilli Rasathanesi web sitesinden deprem verilerini scrape eder ve JSON dosyasına kaydeder.
"""
try:
# Web sitesine GET isteği gönder
response = requests.get(KANDILLI_URL)
response.raise_for_status() # Hata kontrolü

# HTML içeriğini ayrıştır
soup = BeautifulSoup(response.content, "html.parser")
```

```
# Tablodaki verileri bul
pre_tag = soup.find("pre")
if not pre_tag:
raise ValueError("Tablo verisi bulunamadı. Yapıyı kontrol edin.")

# Verileri satırlara böl
lines = pre_tag.text.strip().split("\n")[6:] # İlk 6 satır başlık bilgisi
earthquake_data = []

# Satırları ayrıştır
for line in lines:
columns = line.split()
if len(columns) >= 9:
earthquake = {
"date": f"{columns[0]} {columns[1]}", # Tarih ve saat birleştirilir
"latitude": columns[2],
"longitude": columns[3],
"depth_km": columns[4],
"magnitude": columns[6],
"location": " ".join(columns[8:]), # Lokasyon bilgisi
}
earthquake_data.append(earthquake)

# JSON dosyasına kaydet
with open(output_file, "w") as file:
json.dump(earthquake_data, file, indent=4, ensure_ascii=False)

print(f"Kandilli verileri '{output_file}' dosyasına kaydedildi.")
except Exception as e:
print(f"Veri çekme başarısız oldu: {e}")
```

```
# Örnek kullanım
if __name__ == "__main__":
scrape_kandilli_earthquake_data("kandilli_earthquake_data.json")

import requests
import json
from datetime import datetime, timedelta

# USGS API endpoint
USGS_API_URL = "https://earthquake.usgs.gov/fdsnws/event/1/query"

def fetch_earthquake_data(start_time, end_time, min_magnitude, max_magnitude, output_file):
"""
USGS API'den deprem verilerini çeker ve JSON dosyasına kaydeder.
"""
params = {
"format": "geojson",
"starttime": start_time,
"endtime": end_time,
"minmagnitude": min_magnitude,
"maxmagnitude": max_magnitude
}

try:
# API isteği gönder
response = requests.get(USGS_API_URL, params=params)
response.raise_for_status() # Hata kontrolü

# Yanıtı JSON formatında işleme
data = response.json()
```

```
# JSON dosyasına kaydet
with open(output_file, "w") as file:
json.dump(data, file, indent=4)

print(f"Veriler '{output_file}' dosyasına kaydedildi.")
except requests.exceptions.RequestException as e:
print(f"API isteği başarısız oldu: {e}")

# Örnek kullanım
if __name__ == "__main__":
# Geçtiğimiz bir haftalık depremleri çek
end_date = datetime.utcnow()
start_date = end_date - timedelta(days=7)

fetch_earthquake_data(
start_time=start_date.strftime("%Y-%m-%d"),
end_time=end_date.strftime("%Y-%m-%d"),
min_magnitude=4.0, # Minimum büyüklük
max_magnitude=10.0, # Maksimum büyüklük
output_file="earthquake_data.json"
)

from bs4 import BeautifulSoup
import requests
import json

# AFAD web sitesi URL'si
AFAD_URL = "https://deprem.afad.gov.tr/last-earthquakes.html" # URL'yi kontrol edin

def scrape_afad_earthquake_data(output_file):
```

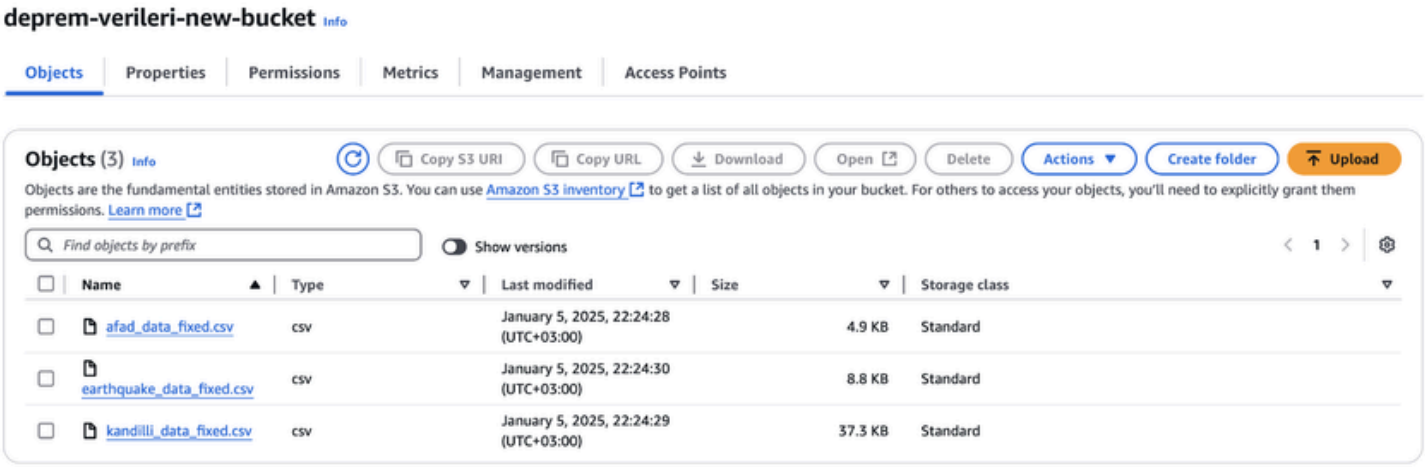
```
"""

AFAD web sitesinden deprem verilerini scrape eder ve JSON dosyasına kaydeder.

"""

try:

# Web sitesine GET isteği gönder (User-Agent eklenmiş)

headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36"}

response = requests.get(AFAD_URL, headers=headers)

response.raise_for_status() # Hata kontrolü


# HTML içeriğini ayrıştır

soup = BeautifulSoup(response.content, "html.parser")


# Deprem verilerini içeren tabloyu bul

table = soup.find("table") # Tablonun ID'si yoksa genel olarak "table" öğesini alın

if not table:

raise ValueError("Tablo bulunamadı. Tablo yapısını kontrol edin.")


rows = table.find_all("tr")[1:] # Başlık satırını atla


# Verileri ayrıştır

earthquake_data = []

for row in rows:

cols = row.find_all("td")

if len(cols) >= 6: # Tabloda en az 6 sütun varsa

earthquake = {

"date": cols[0].text.strip(),

"latitude": cols[1].text.strip(),

"longitude": cols[2].text.strip(),

"depth": cols[3].text.strip(),

"magnitude": cols[4].text.strip(),
```

```
"location": cols[5].text.strip(),
}
earthquake_data.append(earthquake)

# JSON dosyasına kaydet
with open(output_file, "w") as file:
json.dump(earthquake_data, file, indent=4)

print(f"AFAD verileri '{output_file}' dosyasına kaydedildi.")
except Exception as e:
print(f"Veri çekme başarısız oldu: {e}")

# Örnek kullanım
if __name__ == "__main__":
scrape_afad_earthquake_data("afad_scraped_data.json")
```

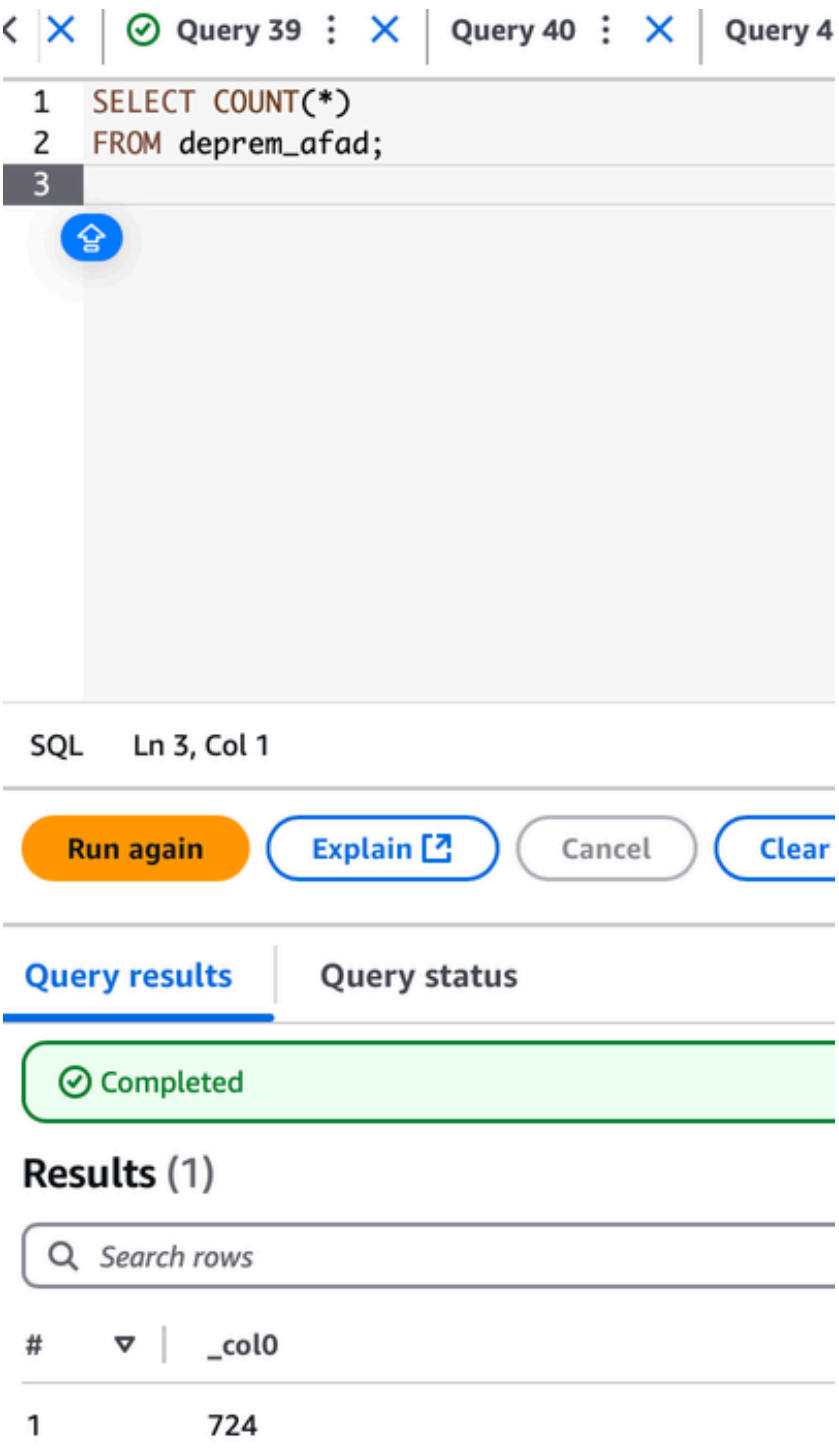
CONNETİNG S3 SEVİVCE - UPLOADİNG DATA BUKCET

```python
import boto3
import os

# AWS S3'e bağlan
s3 = boto3.client('s3', region_name='eu-central-1')  # Frankfurt bölgesi

# Yüklemek istediğiniz verilerin yolu
file_paths = [
    '/Users/gokcesoylu/CloudComputing/afad_data_fixed.csv',
    '/Users/gokcesoylu/CloudComputing/kandilli_data_fixed.csv',
    '/Users/gokcesoylu/CloudComputing/earthquake_data_fixed.csv'
]

# Bucket adı
bucket_name = 'deprem-verileri-new-bucket'

# Verileri yükleme
for file_path in file_paths:
    file_name = os.path.basename(file_path)  # Dosya adı
    try:
        s3.upload_file(file_path, bucket_name, file_name)
        print(f"{file_name} başarıyla yüklendi.")
    except Exception as e:
        print(f"{file_name} yüklenirken hata oluştu: {e}")
```

BUCKET


deprem-verileri-new-bucket Info
Objects
Properties
Permissions
Metrics
Management
Access Points
Objects (3) Info
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Find objects by prefix
Show versions
1
Name
Type
Last modified
Size
Storage class
afad_data_fixed.csv
csv
January 5, 2025, 22:24:28 (UTC+03:00)
4.9 KB
Standard
earthquake_data_fixed.csv
csv
January 5, 2025, 22:24:30 (UTC+03:00)
8.8 KB
Standard
kandilli_data_fixed.csv
csv
January 5, 2025, 22:24:29 (UTC+03:00)
37.3 KB
Standard

ATHENA SQL



DATA FETCH,NG FROM KAGGLE

```python
import os
from kaggle.api.kaggle_api_extended import KaggleApi

# Kaggle kimlik bilgilerini ortam değişkenleri olarak ayarla
os.environ["KAGGLE_USERNAME"] = "gkesoylu"
os.environ["KAGGLE_KEY"] = "ee7a3e9d9f6e558e1987a9a9fd9ecda8"

# Kaggle API'yi başlat
api = KaggleApi()
api.authenticate()

# Veri seti ID'sini ve hedef dizini ayarla
dataset = "warcoder/earthquake-dataset"  # Kaggle veri seti ID'si
output_dir = "earthquake_data"

# İndirilecek dizini oluştur
os.makedirs(output_dir, exist_ok=True)

# Kaggle veri setini indir
print(f"Downloading dataset: {dataset}")
api.dataset_download_files(dataset, path=output_dir, unzip=True)

print(f"Dataset downloaded to: {output_dir}")
```

MANIPULATING DATA

```
import csv

from datetime import datetime


# Girdi ve çıktı dosyalarının yolları

input_file = "/Users/gokcesoylu/CloudComputing/earthquake_data/earthquake_data.csv"

output_file = "/Users/gokcesoylu/CloudComputing/kaggle_earthquake2.csv"


# Verileri dönüştürme ve yazma işlemi

def transform_csv(input_file, output_file):

try:

with open(input_file, "r", encoding="utf-8") as infile, open(output_file, "w", newline="", encoding="utf-8") as
outfile:

reader = csv.DictReader(infile)

fieldnames = ["date", "latitude", "longitude", "depth", "magnitude", "location"]

writer = csv.DictWriter(outfile, fieldnames=fieldnames)
```

```
writer.writeheader()
for row in reader:
try:
# Verileri dönüştür
date_time = datetime.strptime(row["date_time"], "%d-%m-%Y %H:%M").strftime("%Y-%m-%dT%H:%M:%S")
latitude = row["latitude"]
longitude = row["longitude"]
depth = row["depth"]
magnitude = row["magnitude"]
location = row["location"] if row["location"] else f"{row['continent']} {row['country']}"
# Yeni satırı yaz
writer.writerow({
"date": date_time,
"latitude": latitude,
"longitude": longitude,
"depth": depth,
"magnitude": magnitude,
"location": location
})
except Exception as e:
print(f"Satır işlenirken hata oluştu: {e}")

print(f"Dönüştürme tamamlandı. Veriler '{output_file}' dosyasına yazıldı.")
except FileNotFoundError:
print(f"Girdi dosyası bulunamadı: {input_file}")
except Exception as e:
print(f"Bir hata oluştu: {e}")

# Fonksiyonu çalıştır
transform_csv(input_file, output_file)
```

UPLOADING DATA TO BUCKET

```python
import boto3
import os

# AWS S3'e bağlan
s3 = boto3.client('s3', region_name='eu-central-1')  # Frankfurt bölgesi

# Yüklemek istediğiniz verilerin yolu
file_paths = [
    '/Users/gokcesoylu/CloudComputing/kaggle_earthquake.csv',
    '/Users/gokcesoylu/CloudComputing/kaggle_earthquake2.csv'
]

# Bucket adı
bucket_name = 'deprem-verileri-new-bucket'

# Verileri yükleme
for file_path in file_paths:
    file_name = os.path.basename(file_path)  # Dosya adı
    try:
        s3.upload_file(file_path, bucket_name, file_name)
        print(f"{file_name} başarıyla yüklendi.")
    except Exception as e:
        print(f"{file_name} yüklenirken hata oluştu: {e}")
```

✓ 2.6s

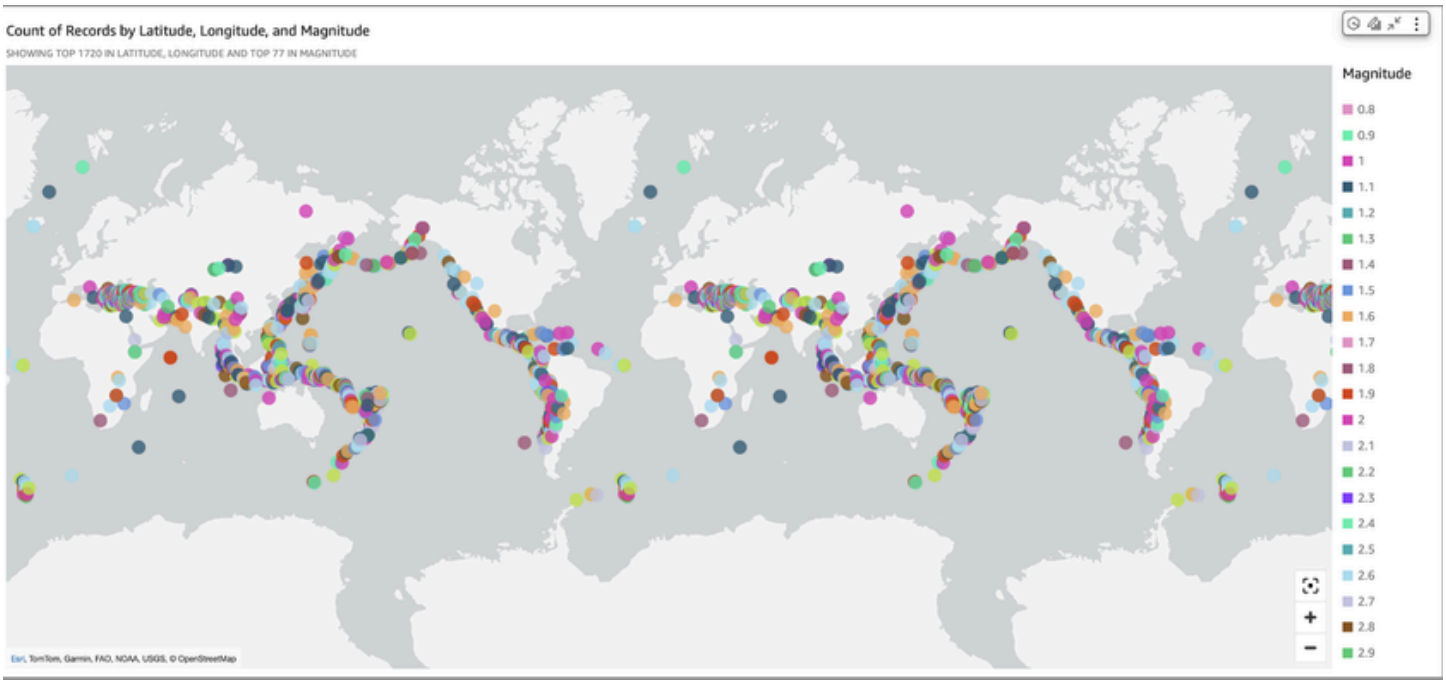
QUICKSIGHT

1. Geospatial Chart

- **Visualized Variables:**
  - **Latitude**: Earthquake latitude coordinates.
  - **Longitude**: Earthquake longitude coordinates.
  - **Color**: Magnitude.
  - **Size**: Not specified.

Analysis:

The geospatial chart highlights the geographic distribution of earthquakes, with their magnitudes represented through color intensity. Larger magnitudes are expected to show stronger and more noticeable colors. This visualization is crucial for understanding the spatial concentration and severity of earthquakes globally.

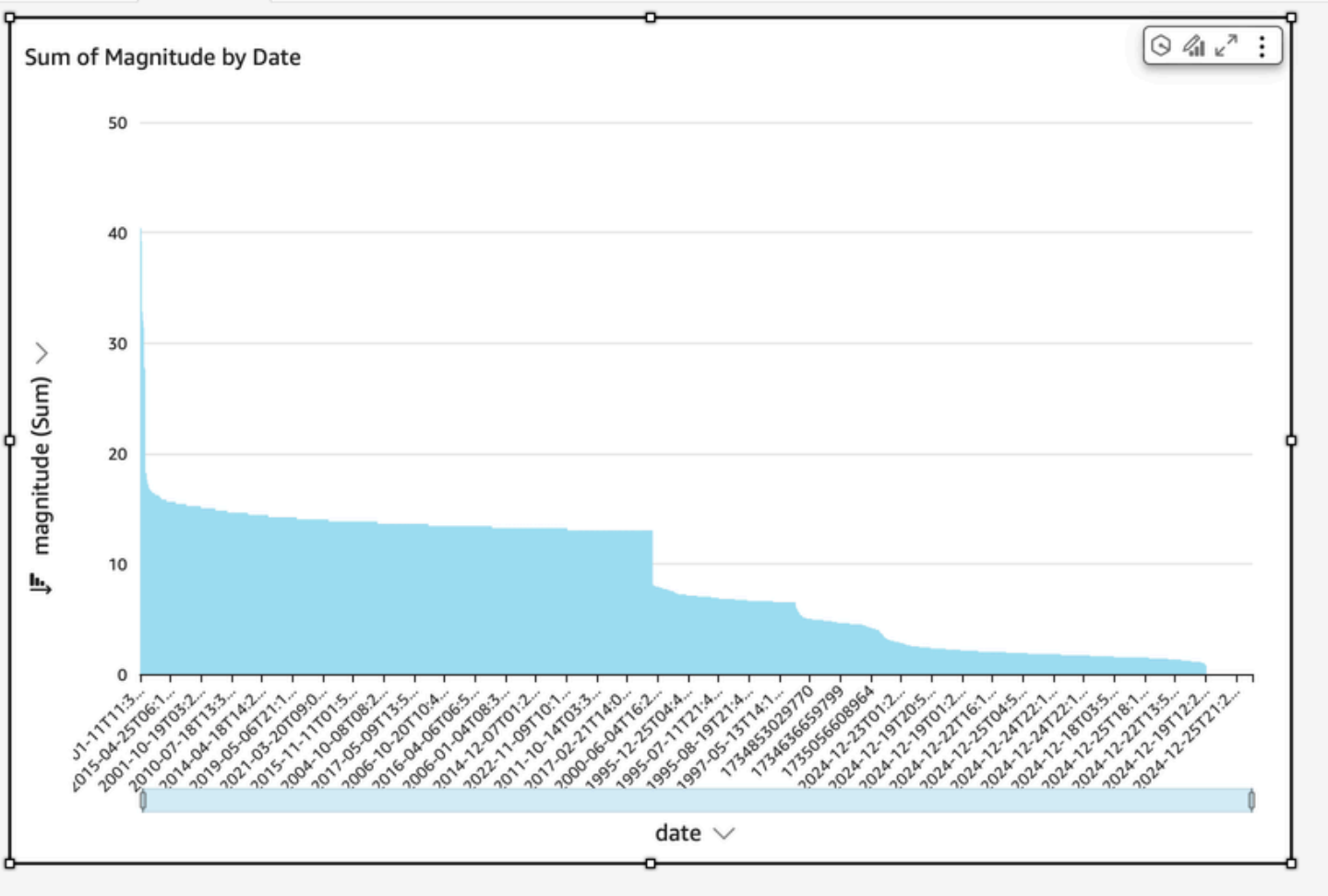


2. Vertical Bar Chart

- **X-Axis**: Date.
- **Value (Y-Axis)**: Sum of Magnitude.

Analysis:

The vertical bar chart demonstrates the sum of earthquake magnitudes over time. This allows for the identification of time periods with increased seismic activity and potential seasonal patterns or trends.

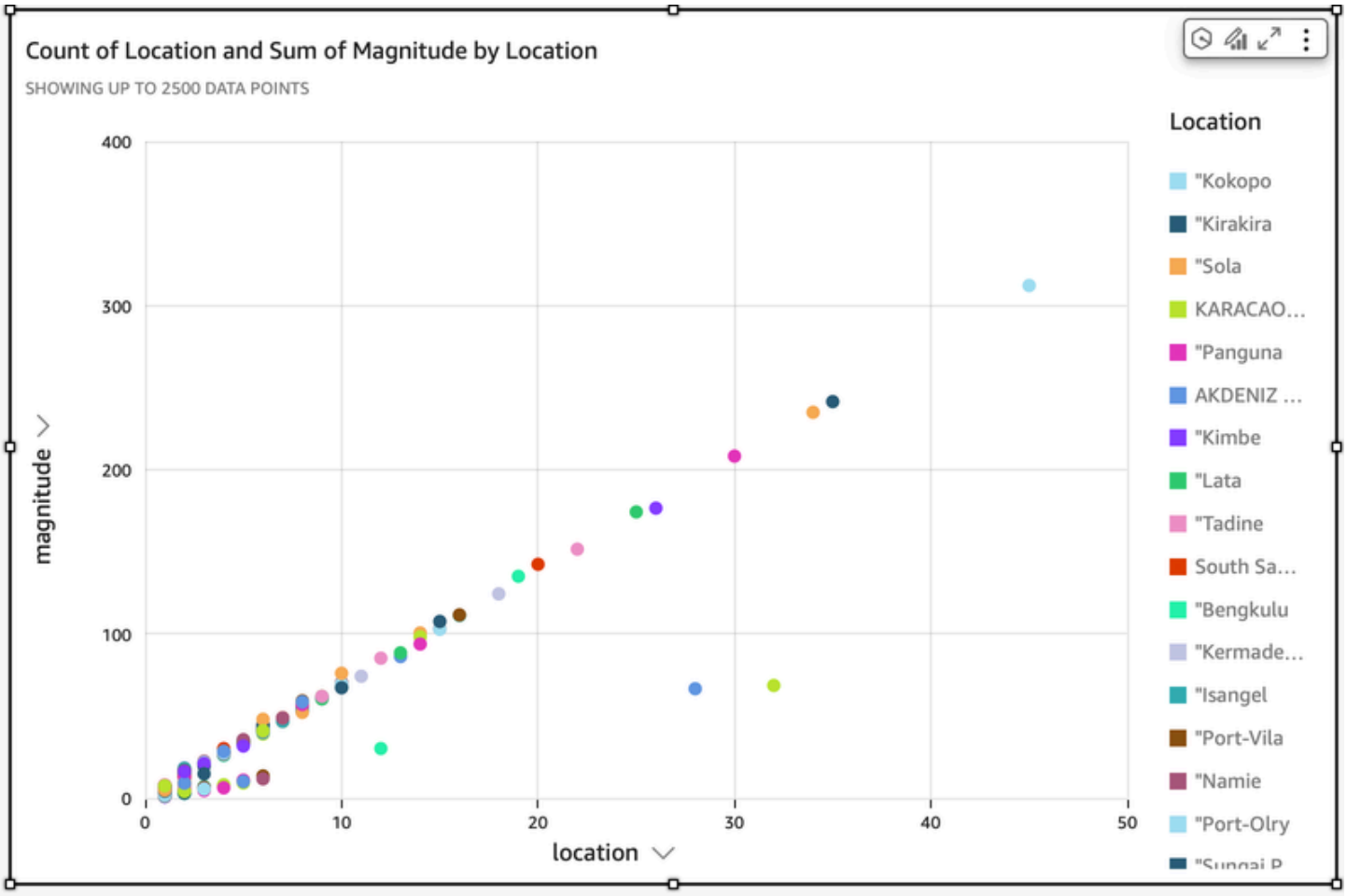


3. Scatter Plot

- **X-Axis**: Count of Location (frequency of earthquakes in different locations).
- **Y-Axis**: Sum of Magnitude.
- **Color**: Location.

Analysis:

This scatter plot displays the relationship between the frequency of earthquakes at specific locations and their combined magnitudes. Points with higher magnitudes and more frequent earthquakes are expected to stand out, aiding in the identification of high-risk areas.

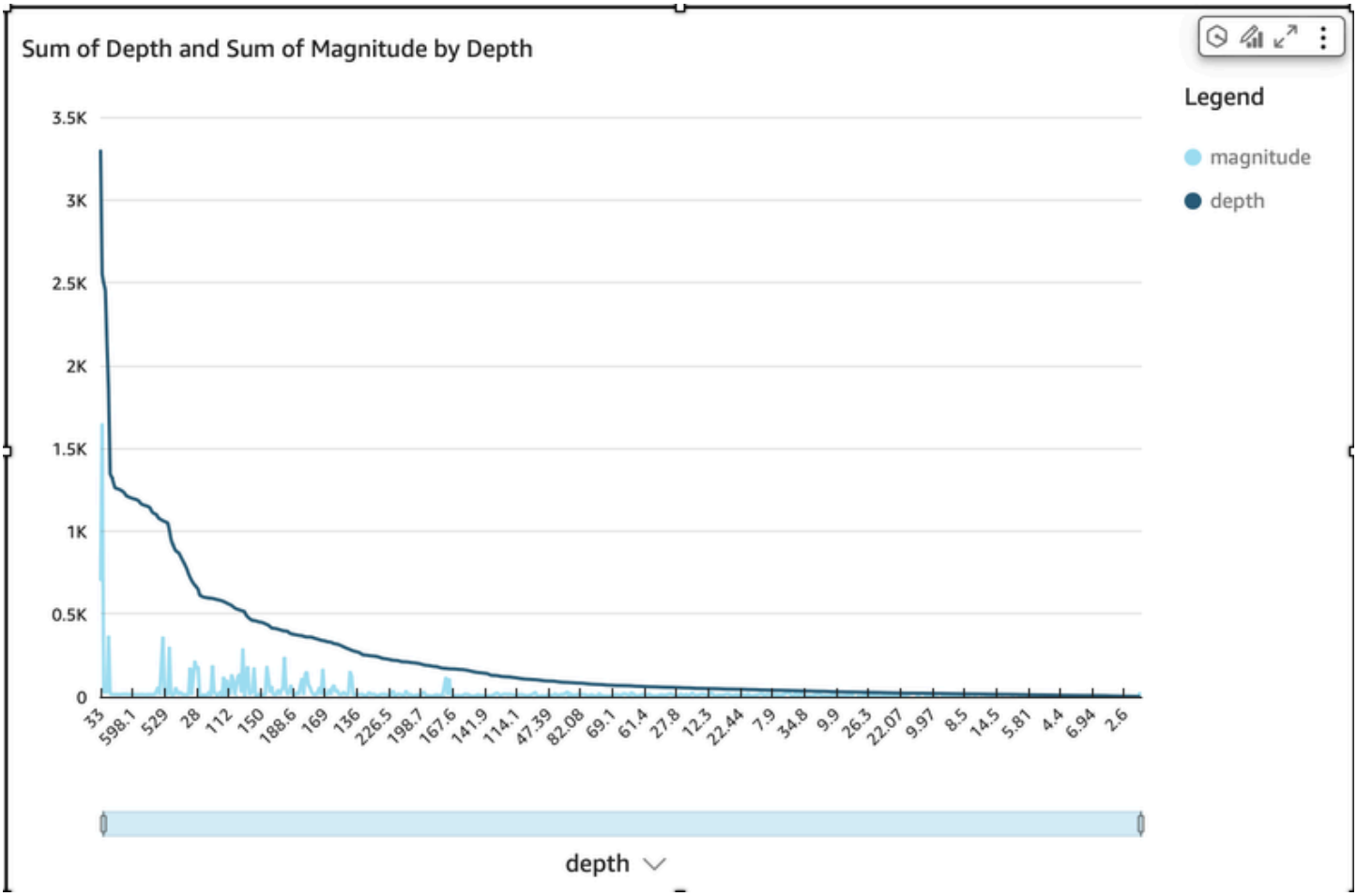


4. Line Chart

- **X-Axis**: Depth.
- **Value (Y-Axis)**: Sum of Depth and Sum of Magnitude.

Analysis:

The line chart visualizes the relationship between earthquake depth and magnitude. By observing the trends, we can determine whether deeper earthquakes tend to have higher or lower magnitudes. This is useful for understanding how depth influences seismic energy release.


Sum of Depth and Sum of Magnitude by Depth
Legend
magnitude
depth
3.5K
3K
2.5K
2K
1.5K
1K
0.5K
0
33
598.1
529
28
112
150
188.6
169
136
226.5
198.7
167.6
141.9
114.1
47.39
82.08
69.1
61.4
27.8
12.3
22.44
7.9
34.8
9.9
26.3
22.07
9.97
8.5
14.5
5.81
4.4
6.94
2.6
depth