Aftershocks

Bupe Chanda

Earthquakes are caused by the sudden release of energy initiated at a rupture below the surface. After an initial earthquake, the *mainshock*, the region surrounding the initial rupture might be unstable, causing secondary earthquakes, the *aftershocks*. We will study a dataset of earthquakes, and model the probability of aftershocks based on quantities such as the distance to the mainshock rupture.

Question 1

We have several tables with information about earthquakes. `all_events.csv` contains the `date`, location (latitude `lat` and longitude `lon`), identifier `id`, intensity `mw` and seismic moment `moment` of many earthquakes. The tables in the folder `aftershocks/` contain the mechanical stresses `s1,...,s6` at different locations surrounding a mainshock, and a column indicating if an aftershock was identified at that location (0 if aftershock was not recorded, 1 otherwise). The table `selectedEvents.csv` contains a list of identifiers `id` and a list of the files with the corresponding aftershock tables.

(a) Create a new dataframe with six columns: `date`, `file`, `lat`, `lon`, `mw`, `aftershocks` with a row for each of the selected events, containing the date (from `all_events.csv`), the file containing the aftershock information (from `selectedEvents.csv`), the location of the mainshock, the intensity and the total number of aftershocks. Make sure the new dataframe is sorted by date, and display the first few rows using `head`.

(b) Implement a function `process_stress(fi, fu)` that receives the name of an aftershock file `fi` and a function `fu`. `fu` receives six arguments (the stress components `s1,...,s6`), and returns a single value. `process_stress` returns a data frame with columns `x`, `y`, `fu` and `aftershock`, with values from the corresponding aftershock file, and the outputs of the function `fu` for each row. Apply it to the event `2001BHUJIN01YAGI` with $f(s_1, \dots, s_6) = \sum_i |s_i|$, and display the first few rows of the result with `head`.

(c) Create new dataframe with four columns, `file` (from `selectedEvents.csv`), `lat`, `lon`, and `moment` (from `all_events.csv`). Sort it by the column `file` and display the first few rows with `head`.

Question 2

Note: if you are not familiar with any of the *geoms* required for this question, check the documentation of `ggplot` or `plotnine`, either with the RStudio help or searching the online documentation.

(a) Use `geom_map` (Python) or `geom_sf` (R) and the file `worldMap.shp` to plot a map of all the events in `all_events.csv`, a point for each event. Note: in R, you will need to read `worldMap.shp` first using the function `st_read` from the library `sf`; in Python, read `worldMap.shp` using `geopandas.read_file`.

(b) Use `geom_map` (Python) or `geom_sf` (R) with `worldMap.shp` to plot a map with a point for each event in `selectedEvents.csv`. Use colour to represent the intensity, and size to represent the number of associated aftershocks. Note: in R, you will need to read `worldMap.shp` first using the function `st_read` from the library `sf`; in Python, read `worldMap.shp` using `geopandas.read_file`.

(c) Plot the Euclidean norm of the stresses for `2001BHUJIN01YAGI` at the (x, y) coordinates in the corresponding file, using colour for the value of the norm, and include black points at the location of the aftershocks.

Question 3

We are going to model the probability of an aftershock with

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}, \tag{1}$$

where x will be a variable that we use to make the prediction. We are going to find the *best* parameter values β_0, β_1 to model the data of a given main event, by finding the values of β_0, β_1 that minimise

$$f(\beta_0, \beta_1) = \sum_k -y_k \log(p(x_k; \beta_0, \beta_1)) - (1 - y_k) \log(1 - p(x_k, \beta_0, \beta_1)). \tag{2}$$

This expresion corresponds to the negative log-likelihood of a model. Here $y_k \in \{0, 1\}$ is the observed outcome (no aftershock or aftershock present), and x_k is our *predictor* variable, that we will define based on information about the earthquake.

(a) Implement a function `fit(X,Y,gamma)` that receives the vectors with values x_k and y_k, and a step `gamma` for the gradient descent method, and returns β_0, β_1 obtained the gradient descent method with starting point $(0, 0)$. Test it by computing the values for `2001BHUJIN01YAGI` using the Euclidean norm of the stresses as X and the value of the column `aftershock` as Y.

(b) Implement a function `fit_file(fi,fu,gamma)` that finds the optimal values of β_0, β_1 using gradient descent as before, using the data in the aftershock file `fi`, and the function `fu` on the stresses (defined as in Question 1b). Test it by computing the values for `2001BHUJIN01YAGI` using the Euclidean norm of the stresses as X and the value of the column `aftershock` as Y.

(c) Implement a function factory `fit_file_factory(fu,gamma)` to fix the values of `fu` and `gamma` in `fit_file`. Compute the values of β_0, β_1 for all events in *selectedEvents*, using $f(s_1, \ldots, s_6) = \log(\sum_i |s_i|)$ and $gamma = 10^{-3}$. Plot the results with β_0 in the x-axis and β_1 in the y-axis, one point for each event.

Question 4

The logistic regression model from Question 3 can be extended to more variables, by defining the probability

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}}. \tag{3}$$

(a) Write a function `moment_distance(fi)` that receives the name of an aftershock file, and returns a dataframe with three columns: the mainshock seismic log-moment (log of `moment` in `all_events.csv`), the distance between the mainshock and the possible aftershock location computed (assume that the mainshock is at the centre of the grid of points in the aftershock file), and column with the presence/abscence of an aftershock. Use the column names `moment`, `distance`, `aftershock`, and note that the `moment` is the same for all the rows, since we are looking only at one mainshock event. Display the first few rows of the dataframe obtained by applying this function to `2001BHUJIN01YAGI`.

(b) Implement a function `fit2(X1,X2,Y)` that minimises the negative log-likelihood function f in Question 3 and returns the values of $\beta_0, \beta_1, \beta_2$. Use `optim` (in R) or `scipy.optimize.minimize` in Python, and **do not** use the derivative of f. Obtain the values of $\beta_0, \beta_1, \beta_2$ for `2001BHUJIN01YAGI` using `moment` for x_1, `distance` for x_2 and `aftershock` for y.

(c) Implement a function `fit2_file(fi)` that returns the values of $\beta_0, \beta_1, \beta_2$ for the aftershock file `fi` using `moment` for x_1, `distance` for x_2 and `aftershock` for y. Plot the values of β_1 vs β_0 and β_2 vs β_0 in two separate plots, one point for each event in `selectedEvents.csv`.

Required Modules

```python
import numpy as np
import pandas as pd
import os
import geopandas as gpd
from geopandas import read_file
from plotnine import *
from glob import glob
from scipy.optimize import minimize
```

Answer 1

(a)

```python
# Load selectedEvents.csv
selected_events_df = pd.read_csv("selectedEvents.csv", usecols=['id', 'file'])

# Merge selectedEvents.csv with all_events.csv on 'id'
first_df = pd.merge(selected_events_df,
pd.read_csv("all_events.csv"), on='id',
how='inner')

# Function to calculate total aftershocks
def calculate_total_aftershocks(file):
    return pd.read_csv(os.path.join("aftershocks", file))['aftershock'].sum()

# Create a new column in the dataframe for the number of aftershocks
# Apply the function to each file in 'file' column
first_df['aftershocks'] = first_df['file'].apply(calculate_total_aftershocks)

# Convert 'date' column to datetime format and sort by date
first_df['date'] = pd.to_datetime(first_df['date'])
first_df.sort_values(by='date', inplace=True, ascending=False)

# Select desired columns and display the first few rows
first_df[['date', 'file', 'lat', 'lon', 'mw', 'aftershocks']].head()
```

	date	file	lat	lon	mw	aftershocks
8	2012-09-05	2012COSTAR01HAYE_grid.csv	10.10170	-85.34460	7.57	25.0
0	2011-03-09	2011OFFSHO01HAYE_grid.csv	38.44870	142.85741	7.30	621.0
6	2007-11-14	2007TOCOPI01SLAD_grid.csv	-22.19723	-69.85295	7.70	70.0
3	2006-12-26	2006PINGTU01YENx_grid.csv	21.69000	120.56000	6.90	20.0
4	2001-01-26	2001BHUJIN01YAGI_grid.csv	23.63000	70.24000	7.66	145.0

(b)

```
def process_stress(fi, fu):
    # Read the aftershock file and create a dataframe
    aftershock_df = pd.read_csv(fi)

    # Apply the function fu to the stress components s1, s2, ..., s6
    aftershock_df['fu'] = aftershock_df.apply(lambda row:
      fu(row['s1'], row['s2'], row['s3'], row['s4'], row['s5'], row['s6']),
      axis=1)

    # Select desired columns and output them
    return aftershock_df[['x', 'y', 'fu', 'aftershock']]

# Define the function fu
def sum_of_absolute_stresses(s1, s2, s3, s4, s5, s6):
    return abs(s1) + abs(s2) + abs(s3) + abs(s4) + abs(s5) + abs(s6)

# Apply process_stress
# Name the data frame: Sum of Absolute Stress Data Frame
SAS_DF = process_stress("aftershocks/2001BHUJIN01YAGI_grid.csv",
sum_of_absolute_stresses)

SAS_DF.head()
```

	x	y	fu	aftershock
0	547594.439578	2.503102e+06	86315.950044	0.0
1	552594.439578	2.503102e+06	93082.647564	0.0
2	557594.439578	2.503102e+06	99307.713303	0.0
3	562594.439578	2.503102e+06	104760.596201	0.0
4	567594.439578	2.503102e+06	109217.784340	0.0

(c)

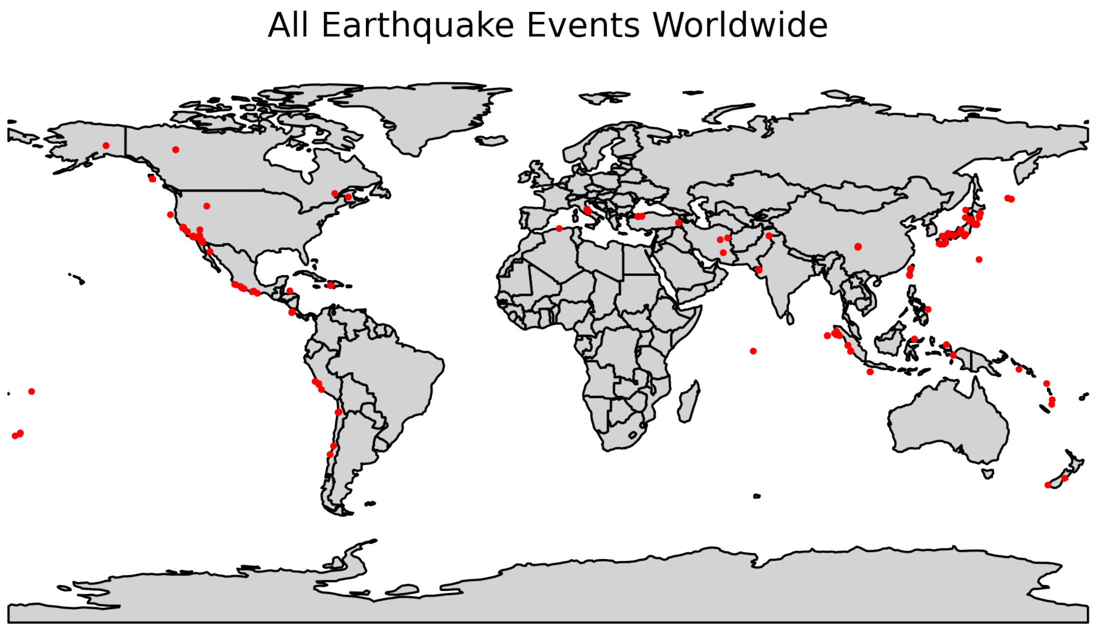
```
# Select the desired columns and create a new dataframe
new_result_df = first_df.loc[:,['file', 'lat', 'lon', 'moment']]

# Sort by the column 'file'
new_result_df.sort_values(by='file', inplace=True)

new_result_df.head()
```

	file	lat	lon	moment
2	1989LOMAPR01WALD_grid.csv	37.0410	-121.8830	2.890000e+19
5	1994NORTHR01WALD_grid.csv	34.2130	-118.5370	1.750000e+19
9	1997ZIRKUH01SUDH_grid.csv	33.8200	59.8000	7.640000e+19
7	1998HIDASW09IDEx_grid.csv	36.3222	137.6327	5.660000e+16
1	2000TOTTOR01IWAT_grid.csv	35.2690	133.3570	2.160000e+19

Answer 2

(a)

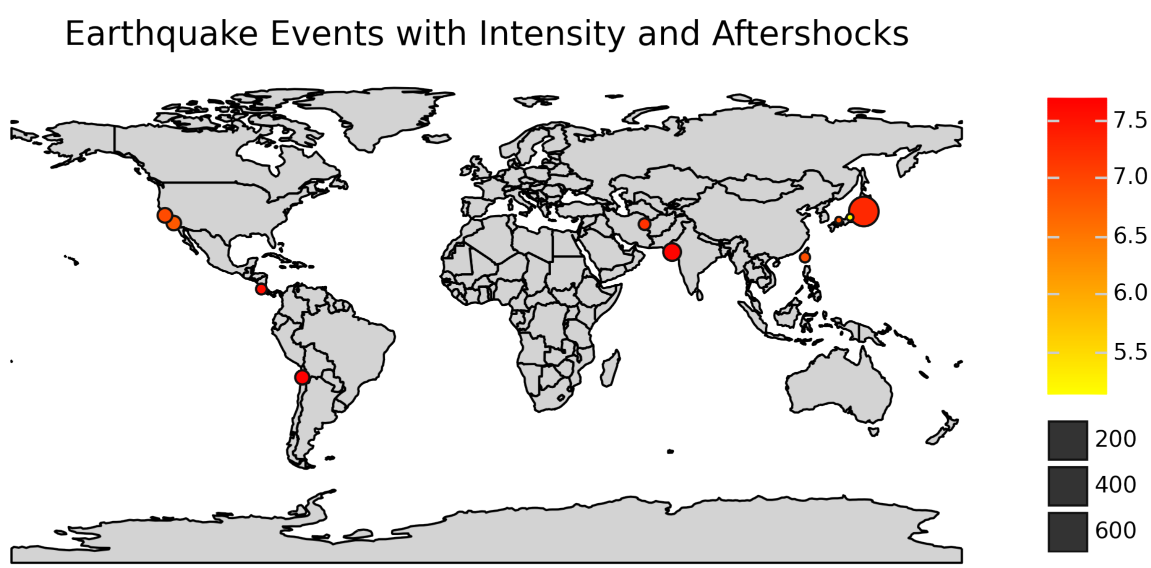
```
# Read earthquake data
all_events_df = pd.read_csv('all_events.csv')

# Read shapefile
world_map = read_file('worldMap.shp')

# Plot world map
base_plot = (
    ggplot() + geom_map(world_map, fill='lightgrey', color='black') +
    theme_void() + labs(title='All Earthquake Events Worldwide'))

# Plot earthquake events
earthquake_plot = (
    base_plot + geom_point(data=all_events_df, mapping=aes(x='lon', y='lat'),
      color='red', size=0.5) + labs(color='Earthquake Events') +
    theme(legend_title=element_blank()) + theme(aspect_ratio=0.5)
)
# Show plot
earthquake_plot.show()
```



(b)

```
# Read earthquake data from the first aftershock dataframe
earthquake_data = gpd.GeoDataFrame(
  first_df,
  geometry=gpd.points_from_xy(first_df['lon'], first_df['lat']))

# Plot world map
base_plot = (
    ggplot() +
    geom_map(world_map, fill='lightgrey', color='black') +
    theme_void() + labs(title='All Earthquake Events Worldwide'))

# Plot earthquake events with colour for intensity and size for aftershocks
earthquake_plot = (
    base_plot +
    geom_map(earthquake_data, aes(fill='mw', size='aftershocks')) +
```

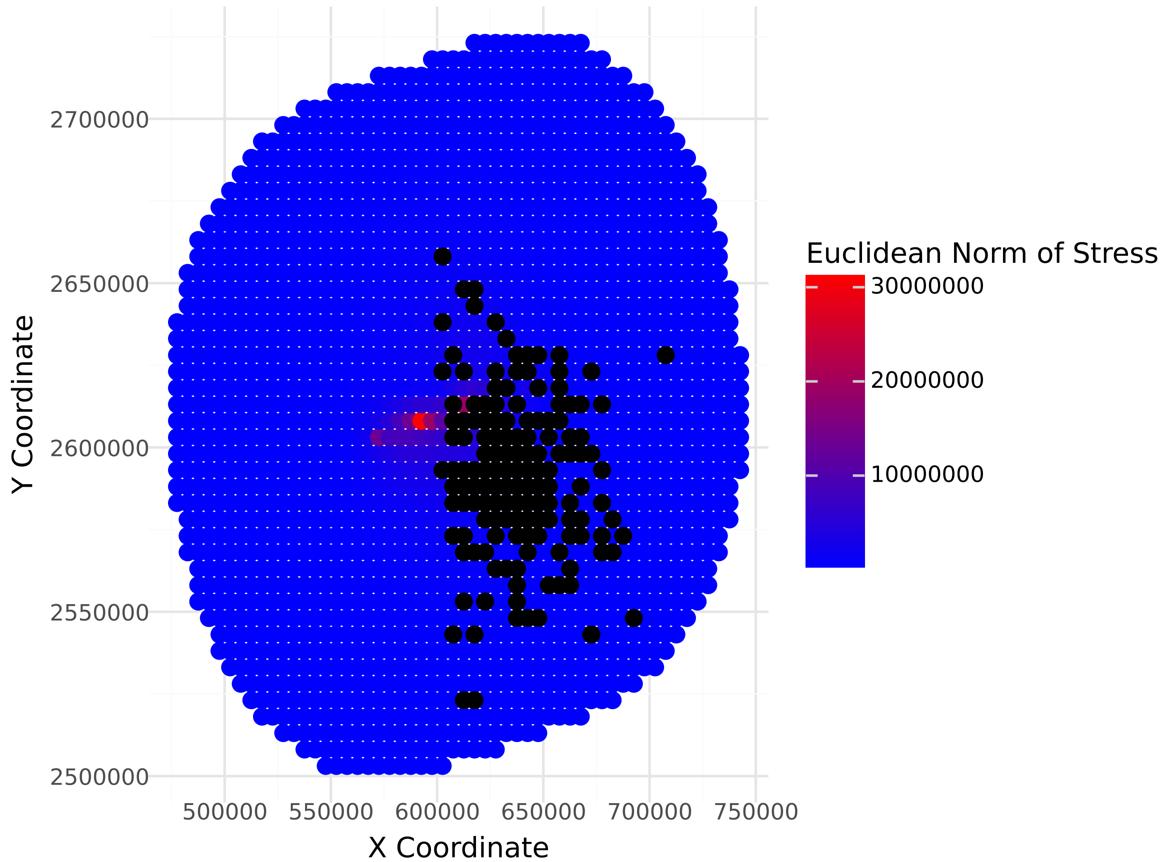
```
    scale_fill_gradient(low="yellow", high="red") +
    labs(title='Earthquake Events with Intensity and Aftershocks') +
    theme_void() + theme(legend_title=element_blank())
    + theme(aspect_ratio=0.5))

# Show plot
earthquake_plot.show()
```



(c)

```
# Define the function fu
def euclidean_norm(s1, s2, s3, s4, s5, s6):
    return (s1**2 + s2**2 + s3**2 + s4**2 + s5**2 + s6**2) ** 0.5

#Create a data frame with name Euclidean Norm Data Frame
EN_DF = process_stress("aftershocks/2001BHUJIN01YAGI_grid.csv",
```

```
euclidean_norm)

# Create a ggplot object with EN_DF DataFrame
plot = ggplot(EN_DF, aes(x='x', y='y')) + \
       geom_point(aes(color='fu'), size=3) + \
       geom_point(data=EN_DF[EN_DF['aftershock'] == 1],
                  color='black', size=3, shape='o') + \
       scale_color_gradient(low="blue", high="red") + \
       labs(x='X Coordinate', y='Y Coordinate',
            color='Euclidean Norm of Stress') + \
       theme_minimal()
plot.draw()
```



Answer 3

(a)

```python
# Define likelihood functions
def logistic_function(x, beta0, beta1):
    z = beta0 + beta1 * x
    clipped_z = np.clip(z, -500, 500)  # Clip values to prevent overflow
    return 1 / (1 + np.exp(-clipped_z))

def negative_log_likelihood(beta0, beta1, X, Y):
    p = logistic_function(X, beta0, beta1)
    return -np.sum(Y * np.log(p) + (1 - Y) * np.log(1 - p))

# Find the fradient of the negative log likelihood
def grad_negative_log_likelihood(beta0, beta1, X, Y):
    p = logistic_function(X, beta0, beta1)

    # Function Gradients
    grad_beta0 = np.sum(Y - p)
    grad_beta1 = np.sum((Y - p) * X)

    return np.array([grad_beta0, grad_beta1])

def fit(X, Y, gamma):
    beta = np.zeros(2)  # Starting point (0,0)
    for i in range(1000):
        grad = grad_negative_log_likelihood(beta[0], beta[1], X[i], Y[i])
        beta -= gamma * grad # Update beta using the gradient descent method
    return beta

# Testing the function fit(X,Y,gamma) with 2001BHUJIN01YAGI
# Use the Euclidean norm Data Frame
fit(EN_DF['fu'], EN_DF['aftershock'], 1e-3)
```

```
array([8.78500000e-01, 4.01779083e+05])
```

(b)

```python
def fit_file(fi, fu, gamma):
    # Make a dataframe from the aftershock file
    aftershock_df = process_stress(fi, fu)

    # Extract X (predictor) and Y (outcome) from the aftershock dataframe
    X = aftershock_df['fu']
    Y = aftershock_df['aftershock']
```

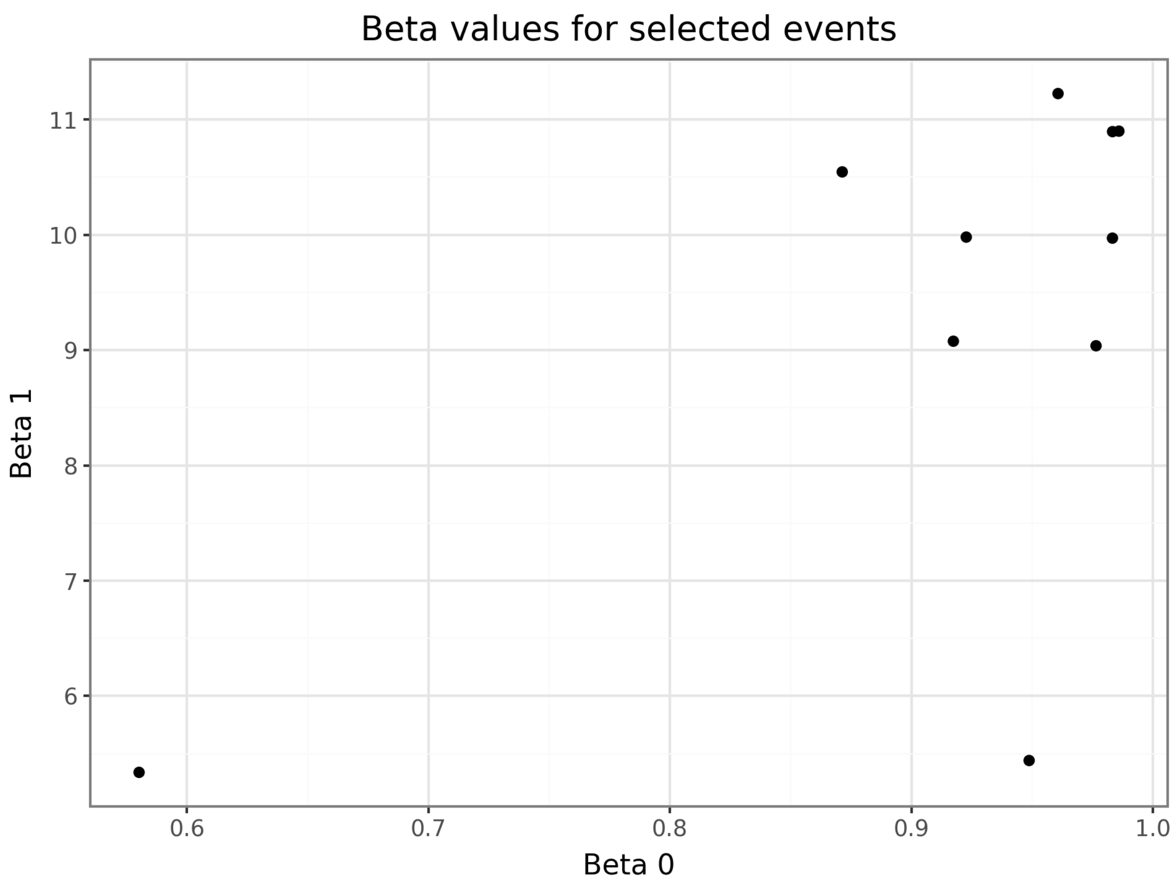
```
    # Find optimal beta values using gradient descent
    beta = fit(X, Y, gamma)

    return beta

fit_file("aftershocks/2001BHUJIN01YAGI_grid.csv", euclidean_norm, 1e-3)
```

```
array([8.78500000e-01, 4.01779083e+05])
```

(c)

```
# Define the log sum function
def abs_sum(s1, s2, s3, s4, s5, s6):
    return np.log(
    np.abs(s1) + np.abs(s2) + np.abs(s3) +
    np.abs(s4) + np.abs(s5) + np.abs(s6))

def fit_file_factory(fu, gamma):
    def fit_file(fi):
        return fit_file_fixed(fi, fu, gamma)
    return fit_file

def fit_file_fixed(fi, fu, gamma):
    # Process the aftershock file
    aftershock_df = process_stress(fi, fu)

    # Extract X (predictor) and Y (outcome) from the processed dataframe
    X = aftershock_df['fu']
    Y = aftershock_df['aftershock']

    # Find optimal beta values using gradient descent
    beta = fit(X, Y, gamma)

    return beta

# Create fit_file function using fit_file_factory
fit_file = fit_file_factory(abs_sum, 1e-3)

# Compute beta values for each event
beta_values = []
beta_values = selected_events_df.apply(
```

```
  lambda row: fit_file("aftershocks/" + row['file']), axis=1).tolist()

# Convert beta values to pandas DataFrame
beta_df = pd.DataFrame(beta_values, columns=['Beta0', 'Beta1'])

# Plot results using plotnine
(ggplot(beta_df, aes(x='Beta0', y='Beta1')) +
 geom_point() +
 labs(x='Beta 0', y='Beta 1', title='Beta values for selected events') +
 theme_bw()
)
```



Answer 4

(a)

```python
def moment_distance(fi):
    #Find the file in the aftershocks folder
    df = pd.read_csv(os.path.join("aftershocks", fi),
    usecols=['x', 'y', 'aftershock'])

    # Assuming that the mainshock is at the centre of the grid of points
    # Find the location of the mainshock
    center_x = df['x'].agg(['min', 'max']).sum() / 2
    center_y = df['y'].agg(['min', 'max']).sum() / 2

    # Find the distance to the centre
    df['distance'] = np.sqrt(
      (df['x'] - center_x)**2 + (df['y'] - center_y)**2)

    # Find the corresponding seismic moment from first_df
    # Extract mainshock ID from filename
    mainshock_id = fi.split('_')[0]
    # Get mainshock row and calculate log moment
    mainshock_row = first_df[first_df['id'] == mainshock_id].iloc[0]
    log_moment = np.log(mainshock_row['moment'])

    # Add moment and aftershock columns to the dataframe
    df['moment'] = log_moment
    df['aftershock'] = df['aftershock'].astype(int)

    # Remove unnecessary columns
    df.drop(columns=['x', 'y'], inplace=True)

    return df

# Test the function with '2001BHUJIN01YAGI_grid.csv'
moment_distance('2001BHUJIN01YAGI_grid.csv').head()
```

	aftershock	distance	moment
0	0	126515.809289	47.290076
1	0	124121.915873	47.290076
2	0	121886.217432	47.290076
3	0	119817.569663	47.290076
4	0	117924.764151	47.290076

(b)

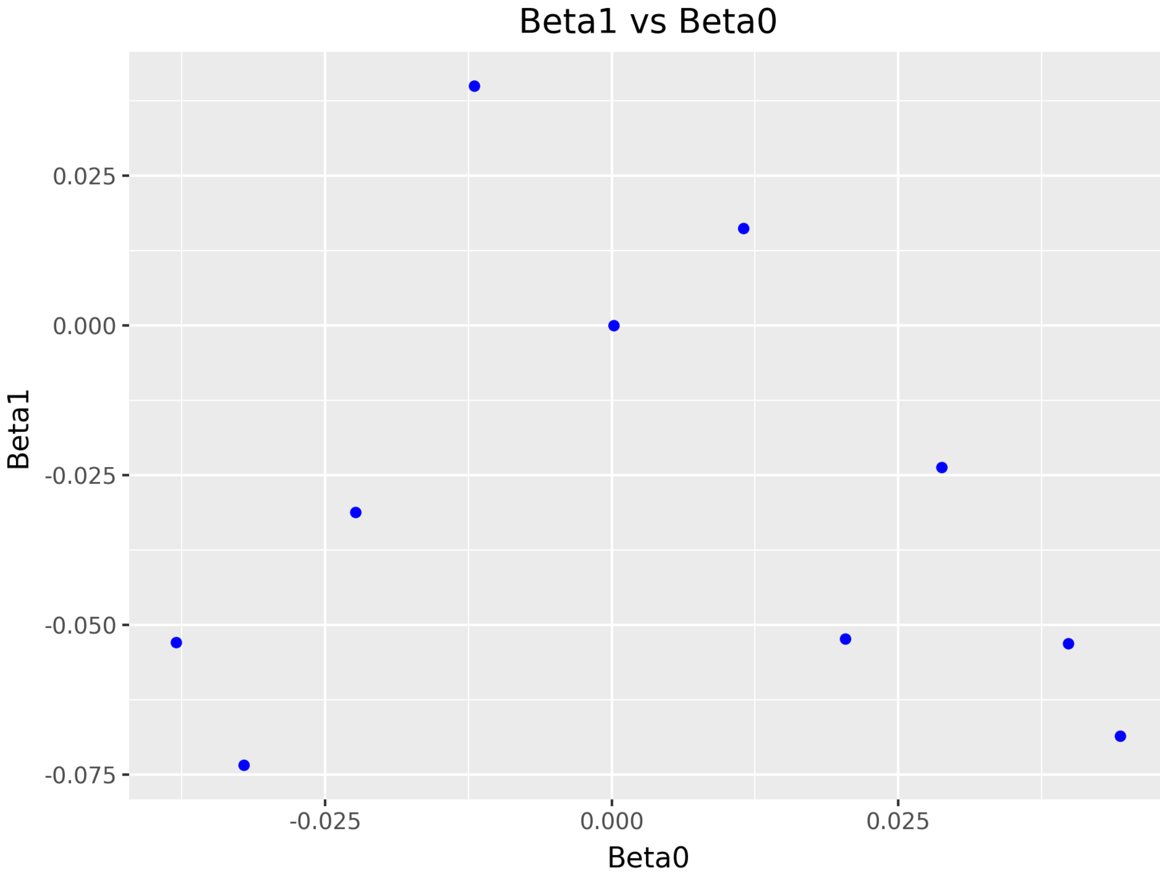
```python
#Define the new likelihood functions

def negative_log_likelihood(beta, X, Y):

    # Calculate the logistic function
    predicted_probabilities = 1 / (1 + np.exp(-
    (beta[0] + beta[1]*X[0] + beta[2]*X[1])))

    # Clip probabilities to avoid division by zero or log of zero
    predicted_probabilities = np.clip(predicted_probabilities,
    1e-15, 1 - 1e-15)

    # Calculate the negative log-likelihood
    log_likelihood = -np.sum(
      Y*np.log(predicted_probabilities)
      + (1-Y)*np.log(1-predicted_probabilities))
    return log_likelihood

def fit2(X1, X2, Y):
    initial_beta = np.zeros(3) # Initial values for beta (0,0,0)

    # Minimize the negative log-likelihood function
    result = minimize(negative_log_likelihood,
    initial_beta, args=(np.array([X1, X2]), Y),
    method='Nelder-Mead')

    # Retrieve the optimized beta values
    return result.x

# Testing the function fit2(X1, X2, Y) with 2001BHUJIN01YAGI data
df = moment_distance('2001BHUJIN01YAGI_grid.csv')

fit2(df['moment'], df['distance'], df['aftershock'])
```

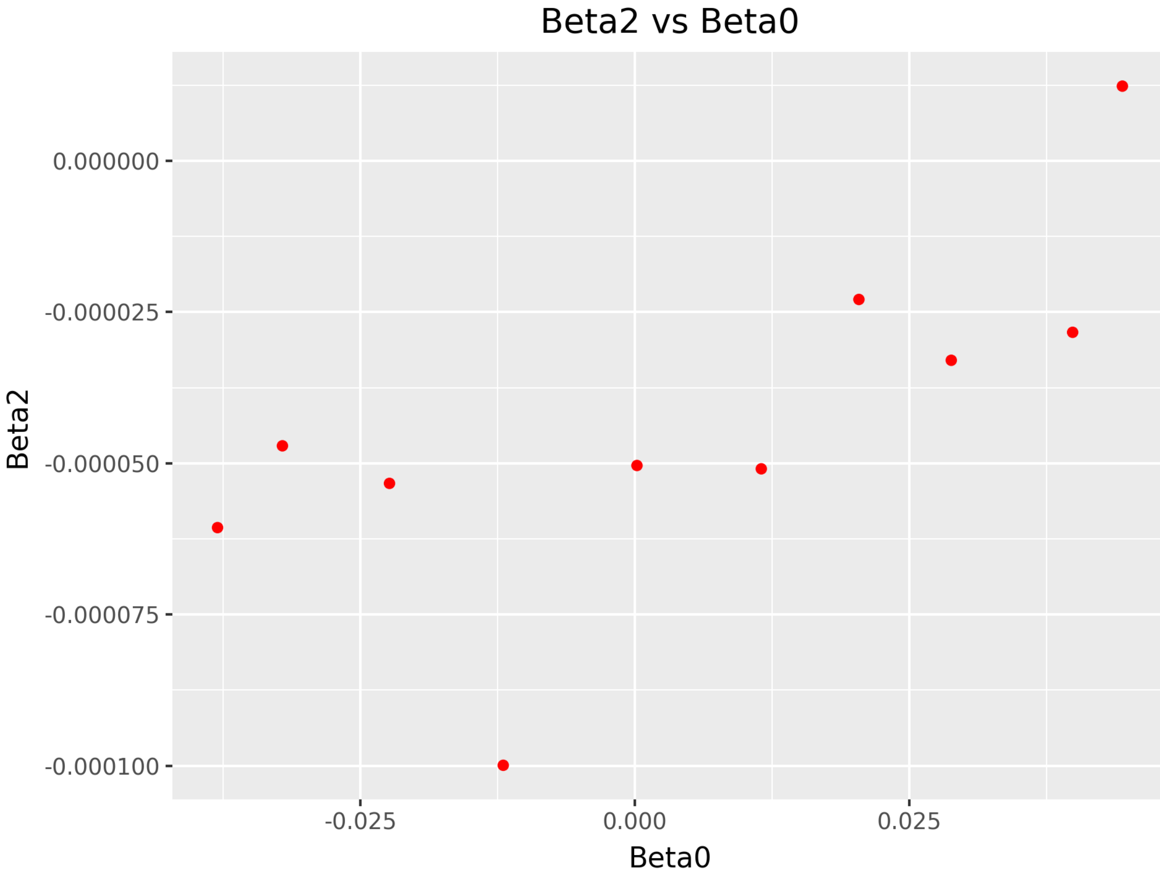
```
array([ 1.15217420e-02,  1.61335242e-02, -5.09192971e-05])
```

(c)

```python
def fit2_file(fi):
    # Extract data from the aftershock file
    df = moment_distance(fi)
```

```python
    # Fit logistic regression model
    beta_values = fit2(df['moment'], df['distance'], df['aftershock'])

    return beta_values

# Plot values of beta1 vs beta0 and beta2 vs beta0
def plot_beta_values(selected_events_df):
    beta_values = selected_events_df['file'].apply(fit2_file).tolist()
    beta0_values, beta1_values, beta2_values = zip(*beta_values)

    df_plot = pd.DataFrame({
        'Beta0': beta0_values,
        'Beta1': beta1_values,
        'Beta2': beta2_values
    })

    # Plot for Beta1 vs Beta 0
    plot1 = (
        ggplot(df_plot, aes(x='Beta0', y='Beta1')) +
        geom_point(color='blue') +
        labs(title='Beta1 vs Beta0', x='Beta0', y='Beta1')
    )

    # Plot for Beta2 vs Beta 0
    plot2 = (
        ggplot(df_plot, aes(x='Beta0', y='Beta2')) +
        geom_point(color='red') +
        labs(title='Beta2 vs Beta0', x='Beta0', y='Beta2')
    )

    return plot1, plot2

# Load selectedEvents.csv
selected_events_df = pd.read_csv("selectedEvents.csv", usecols=['id', 'file'])

# Plot beta values for each of the selected events
plot1, plot2 = plot_beta_values(selected_events_df)

plot1.show()
plot2.show()
```


Beta1 vs Beta0
Beta1
0.025
0.000
-0.025
-0.050
-0.075
-0.025
0.000
0.025
Beta0


Beta2 vs Beta0
0.000000
-0.000025
-0.000050
-0.000075
-0.000100
Beta2
-0.025
0.000
0.025
Beta0