In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [2]:
```
dataset_url = 'https://gist.githubusercontent.com/netj/8836201/raw/6f9306ad21398ea43cba4f7d537619d0e07d5ae3/iris.csv'
iris_df = pd.read_csv(dataset_url)
iris_df.head()
```

Out[2]:

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

In [3]:
```
iris_df.size, iris_df.shape
```

Out[3]:
```
(750, (150, 5))
```

In [4]:
```
iris_df.describe()
```

Out[4]:

	sepal.length	sepal.width	petal.length	petal.width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [5]:

```
iris_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal.length  150 non-null    float64
 1   sepal.width   150 non-null    float64
 2   petal.length  150 non-null    float64
 3   petal.width   150 non-null    float64
 4   variety       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

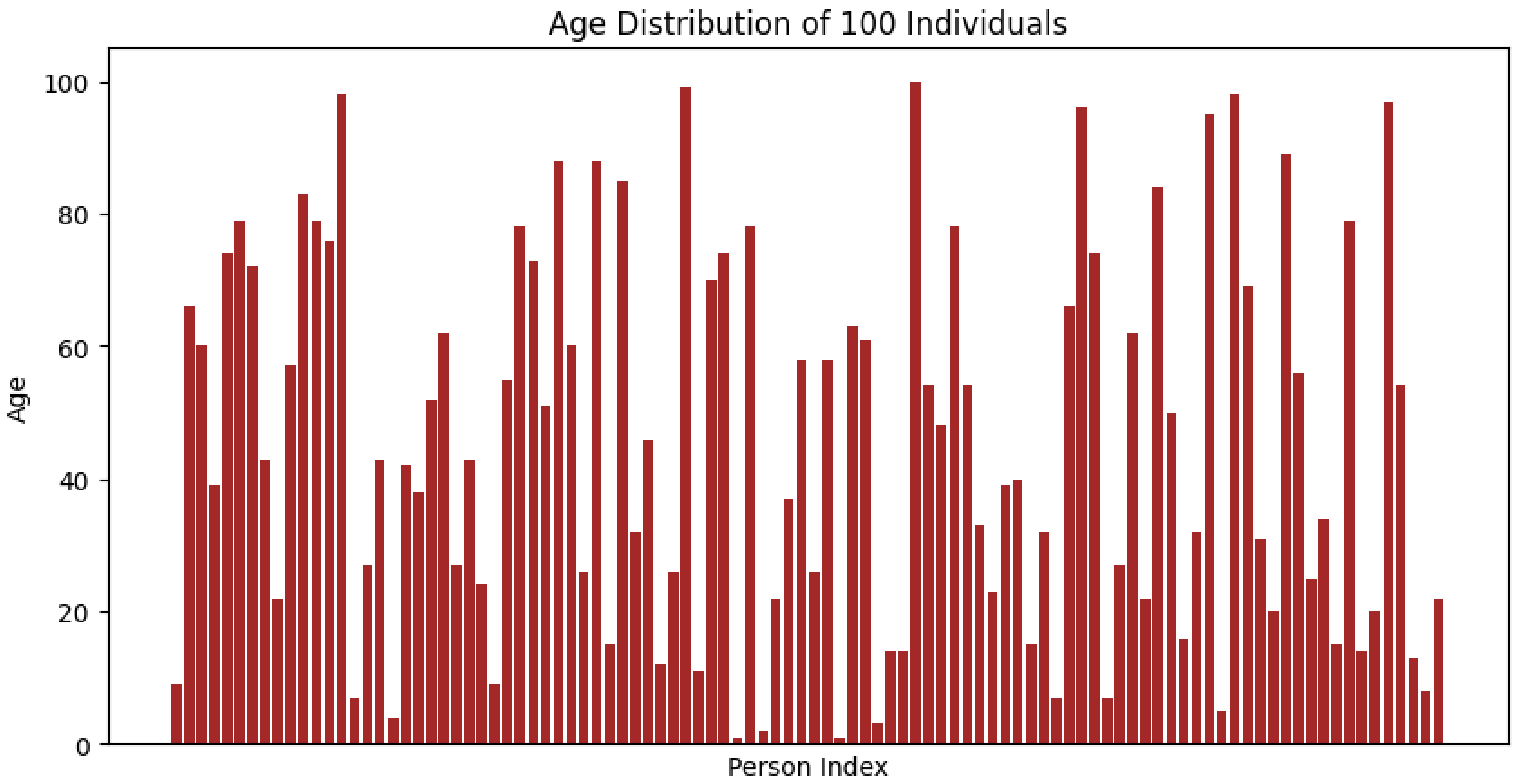
In [6]:

```
sepal_length_col = iris_df['sepal.length']
x_axis_values = list(map(float, input("Enter x-axis (bar edges) values separated by spaces: ").split()))
bins_count = int(input("Enter number of bins: "))
bar_fill = input("Enter the bar color (e.g., 'orange', 'purple'): ")
outline_color = input("Enter the outline color for bars (e.g., 'red'): ")
opacity = float(input("Enter transparency (0 to 1): "))
bar_colors = list(map(str, input("Enter colors for individual bars separated by spaces: ").split()))
```

```
plt.figure(figsize=(12, 5))
plt.hist(sepal_length_col, bins=x_axis_values, color=bar_fill, edgecolor=outline_color, alpha=opacity)

bars_custom = plt.hist(sepal_length_col, bins=bins_count, color=bar_fill, edgecolor=outline_color, alpha=opacity)[2]
for idx in range(len(bars_custom)):
    bars_custom[idx].set_facecolor(bar_colors[idx % len(bar_colors)])

plt.xlabel('Sepal Length')
plt.ylabel('Count')
plt.title('Sepal Length Distribution (Customized)')
plt.show()
```

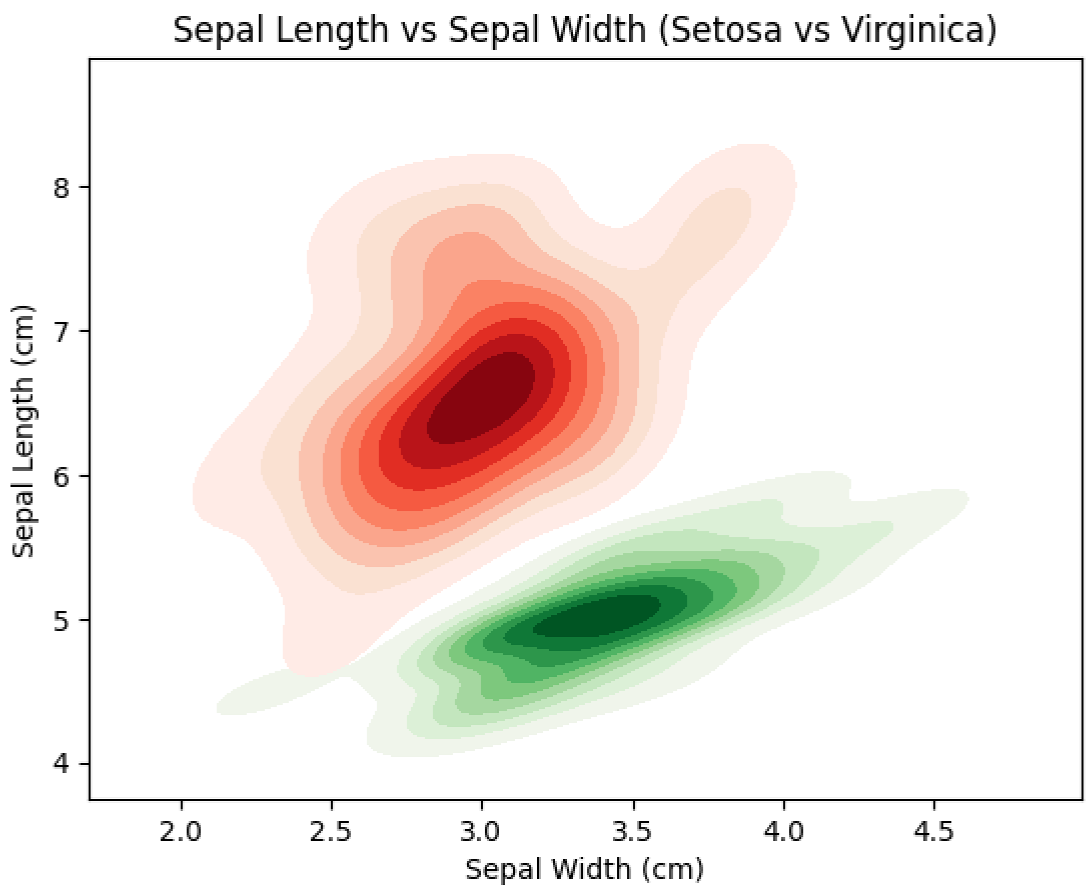
In [7]:
```
person_ages = np.random.randint(1, 101, size=100)
```

```
plt.figure(figsize=(10, 5))
plt.bar(range(100), person_ages, color='brown')

plt.title('Age Distribution of 100 Individuals')
plt.xlabel('Person Index')
plt.ylabel('Age')
plt.xticks([])

plt.show()
```



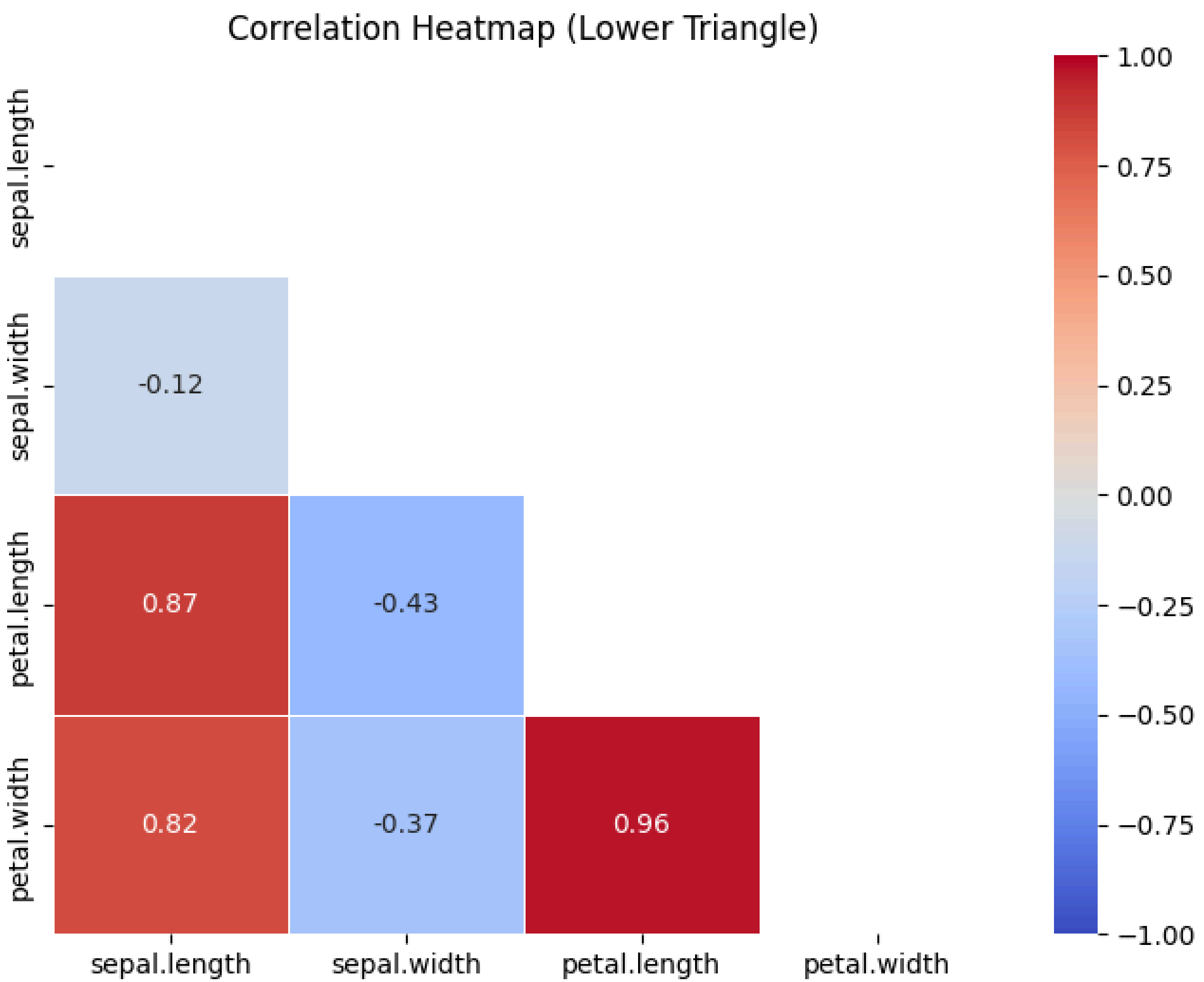
In [8]:
```
setosa_data = iris_df[iris_df["variety"] == "Setosa"]
virginica_data = iris_df[iris_df["variety"] == "Virginica"]

sns.kdeplot(
```

```
    x=setosa_data["sepal.width"],
    y=setosa_data["sepal.length"],
    fill=True,
    cmap="Greens",
    label="Setosa"
)

sns.kdeplot(
    x=virginica_data["sepal.width"],
    y=virginica_data["sepal.length"],
    fill=True,
    cmap="Reds",
    label="Virginica"
)

plt.xlabel("Sepal Width (cm)")
plt.ylabel("Sepal Length (cm)")
plt.title("Sepal Length vs Sepal Width (Setosa vs Virginica)")
plt.grid(False)
plt.show()
```



In [9]:

```
numeric_columns_only = iris_df.drop(columns=['variety'])

corr_matrix = numeric_columns_only.corr()

matrix_mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, mask=matrix_mask, annot=True, cmap='coolwarm', linewidths=0.5, vmin=-1, vmax=1)
plt.grid(False)
plt.title('Correlation Heatmap (Lower Triangle)')
plt.show()
```

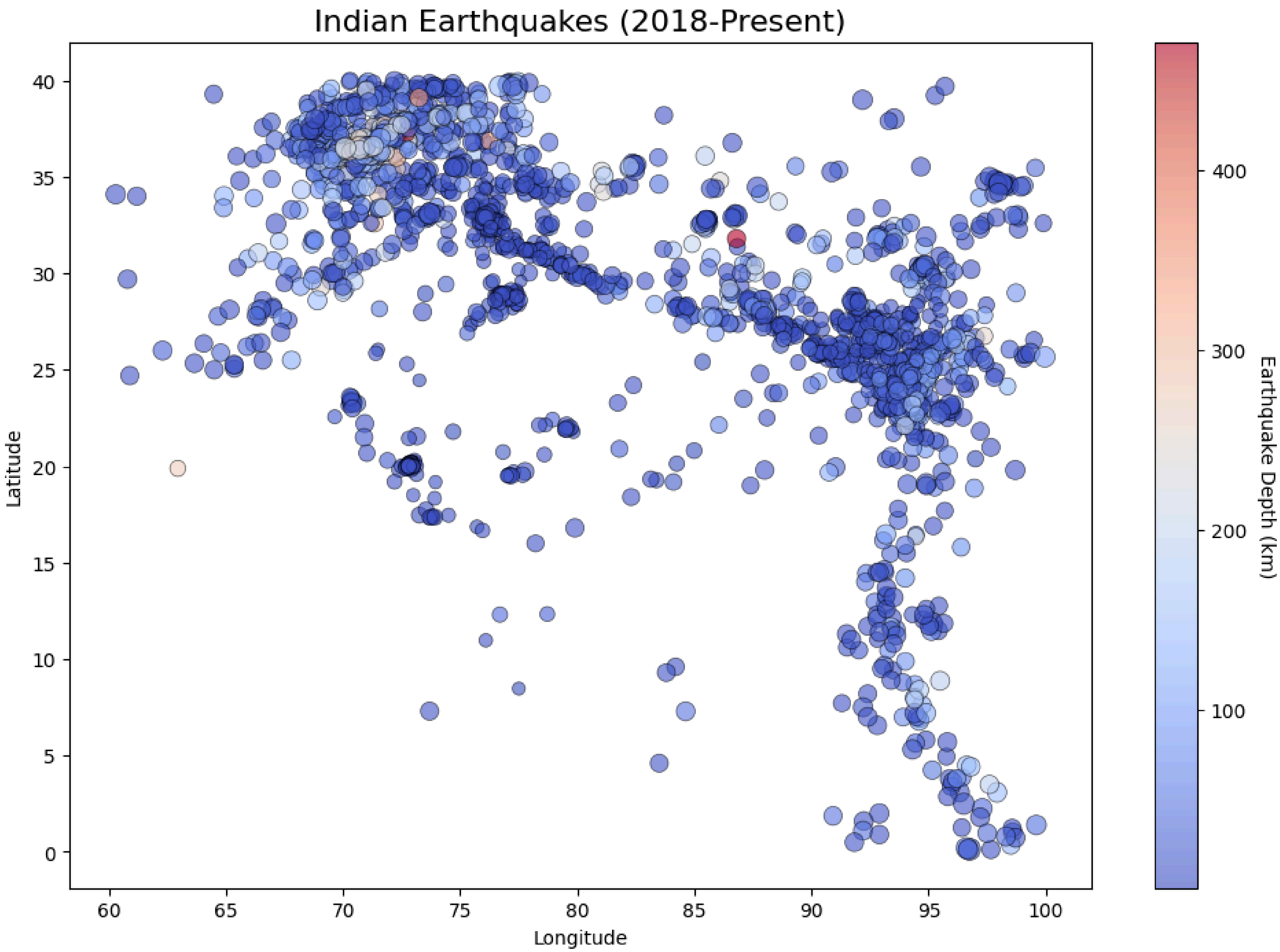


In [10]:

```
earthquake_data = pd.read_csv(r'..\dataset\Indian_earthquake_data.csv')

earthquake_data['Event Time'] = pd.to_datetime(earthquake_data['Origin Time'])

filtered_earthquakes = earthquake_data[earthquake_data['Event Time'] >= '2018-01-01']

plt.figure(figsize=(12, 8))
```

```python
scatter_plot = plt.scatter(
    x=filtered_earthquakes['Longitude'],
    y=filtered_earthquakes['Latitude'],
    s=filtered_earthquakes['Magnitude'] * 20,
    c=filtered_earthquakes['Depth'],
    cmap='coolwarm',
    alpha=0.6,
    edgecolors="black",
    linewidth=0.5
)

colorbar = plt.colorbar(scatter_plot)
colorbar.set_label('Earthquake Depth (km)', rotation=270, labelpad=15)

plt.title('Indian Earthquakes (2018-Present)', fontsize=16)
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.grid(True)

plt.grid(False)
plt.show()
```

```
C:\Users\Ayush\AppData\Local\Temp\ipykernel_12424\2146931759.py:3: FutureWarning: Parsed string "2021-07-31 09:43:23 IST" inclu
ded an un-recognized timezone "IST". Dropping unrecognized timezones is deprecated; in a future version this will raise. Instea
d pass the string without the timezone, then use .tz_localize to convert to a recognized timezone.
  earthquake_data['Event Time'] = pd.to_datetime(earthquake_data['Origin Time'])
```



Indian Earthquakes (2018-Present)
Latitude
Longitude
Earthquake Depth (km)
40
35
30
25
20
15
10
5
0
60
65
70
75
80
85
90
95
100
400
300
200
100

In [11]:
```python
diabetes_df = pd.read_csv(r"..\dataset\diabetes.csv")

print("\nDiabetes Data (Pima Indians):")
diabetes_df.head()
```

```
Diabetes Data (Pima Indians):
```

Out[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [12]:
```python
diabetes_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [13]:
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```python
X_glucose = diabetes_df[['Glucose']]
y_outcome = diabetes_df['Outcome']

X_train_glucose, X_test_glucose, y_train_glucose, y_test_glucose = train_test_split(X_glucose, y_outcome, test_size=0.3, rando

lr_model = LinearRegression()
lr_model.fit(X_train_glucose, y_train_glucose)

y_glucose_pred = lr_model.predict(X_test_glucose)
glucose_mse = mean_squared_error(y_test_glucose, y_glucose_pred)
print(f"Mean Squared Error (Linear Regression - Glucose): {glucose_mse}")
```

```
Mean Squared Error (Linear Regression - Glucose): 0.1831271615072512
```

In [14]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

X_train_log, X_test_log, y_train_log, y_test_log = train_test_split(X_glucose, y_outcome, test_size=0.3, random_state=42)

log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train_log, y_train_log)

y_log_pred = log_model.predict(X_test_log)
log_accuracy = accuracy_score(y_test_log, y_log_pred)
print(f"Accuracy (Logistic Regression - Glucose): {log_accuracy}")
```

```
Accuracy (Logistic Regression - Glucose): 0.7229437229437229
```

In [15]:

```python
X_full = diabetes_df[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction'
y_full = diabetes_df['Outcome']

X_train_full, X_test_full, y_train_full, y_test_full = train_test_split(X_full, y_full, test_size=0.3, random_state=42)

lm_full = LinearRegression()
lm_full.fit(X_train_full, y_train_full)

y_full_pred = lm_full.predict(X_test_full)
full_mse = mean_squared_error(y_test_full, y_full_pred)
print(f"Mean Squared Error (Multiple Regression): {full_mse}")
```

```
Mean Squared Error (Multiple Regression): 0.17603335005142035
```

Linear Regression (Glucose Feature):

- Mean Squared Error (MSE): 0.1831271615072512

Logistic Regression (Glucose Feature):

- Accuracy: 72.29 %

Multiple Regression (All Features):

- Mean Squared Error (MSE): 0.17603335005142035

Conclusion:

The logistic regression model with `Glucose` alone provides an accuracy of 72.29%, which indicates moderate prediction accuracy for diagnosing diabetes. The multiple regression model, utilizing all features, achieves an MSE of 0.17603335005142035, suggesting it may provide a better prediction compared to using glucose alone in a simple linear regression model.