In [7]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [8]:
```python
data_path = 'https://gist.githubusercontent.com/netj/8836201/raw/6f9306ad21398ea43cba4f7d537619d0e07d5ae3/iris.csv'
df = pd.read_csv(data_path)
df.head()
```

Out[8]:

	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 [9]:
```python
df.size, df.shape
```

Out[9]:
```
(750, (150, 5))
```

In [10]:
```python
df.describe()
```

Out[10]:

	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 [11]:

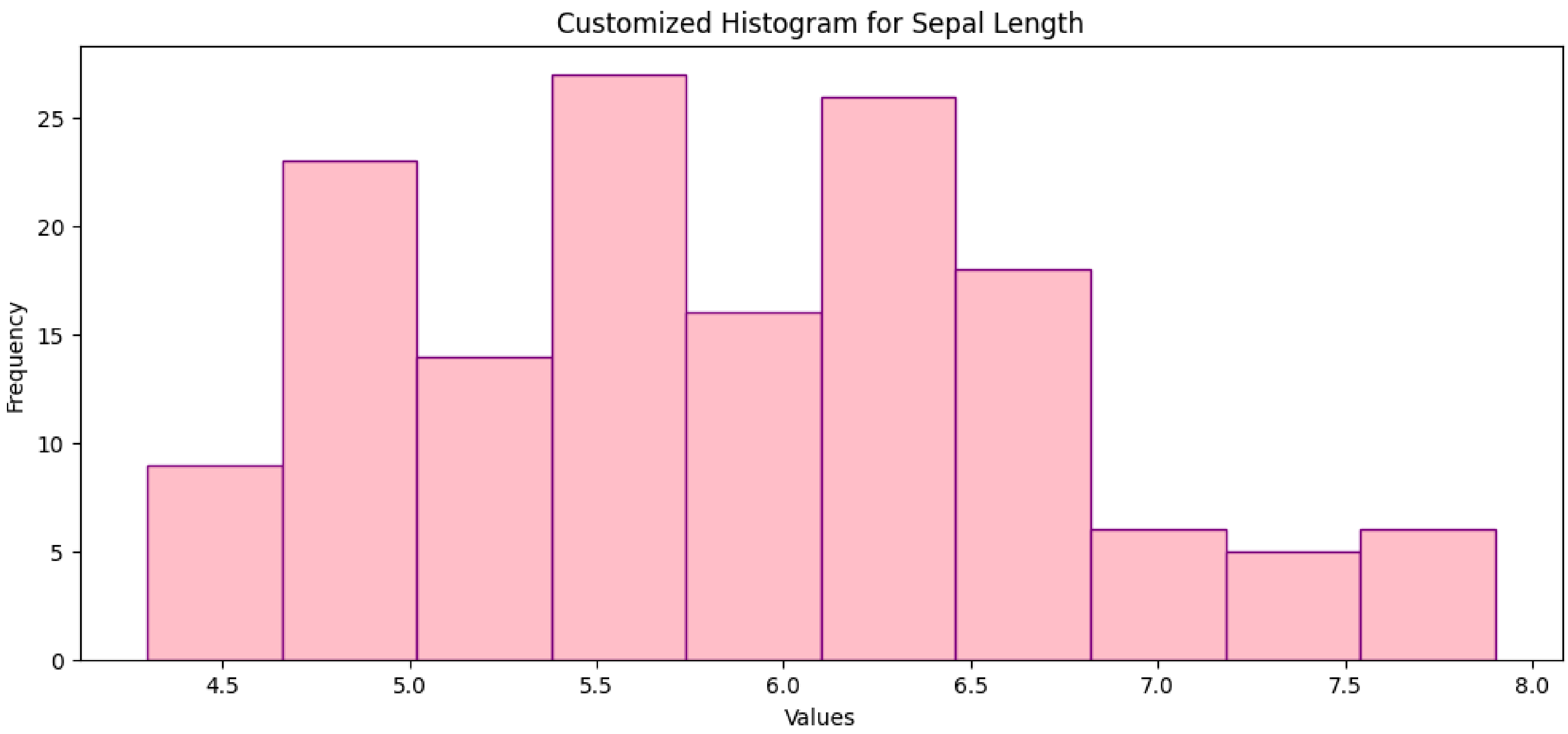
```
df.info()
```

```
<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
```

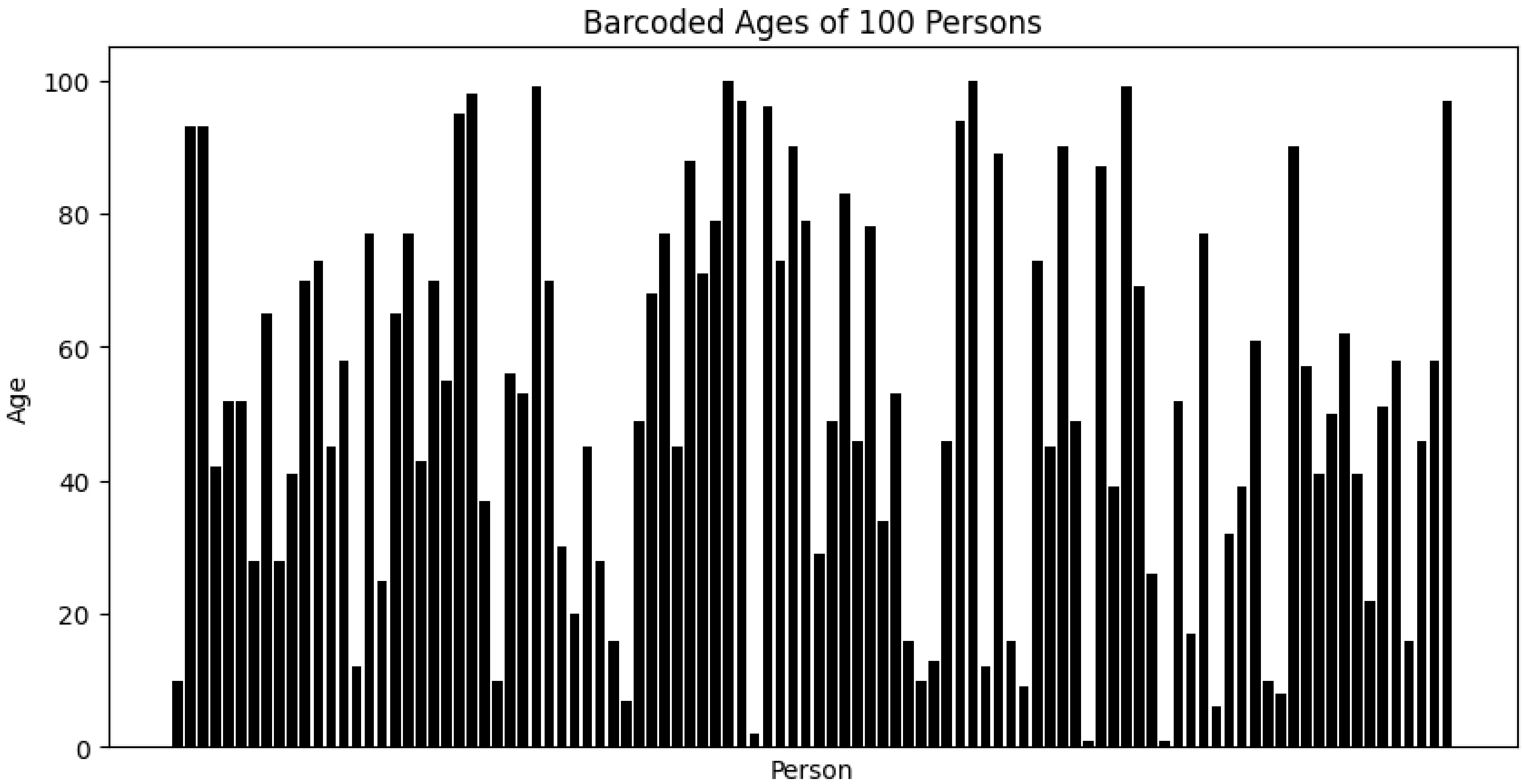
In [12]:

```
numeric_column = df['sepal.length']

x_values = list(map(float, input("Enter the x-axis values (edges of bars) separated by spaces: ").split()))
n_bins = int(input("Enter the number of bins to display: "))
bar_color = input("Enter the bar color (e.g., 'blue', 'green'): ")
edge_color = input("Enter the edge color for the bars (e.g., 'black'): ")
transparency = float(input("Enter the transparency value for the bars (0 to 1): "))
individual_colors = list(map(str, input("Enter the colors for each bar separated by spaces: ").split()))
```

```
plt.figure(figsize=(12, 5))
plt.hist(numeric_column, bins=x_values, color=bar_color, edgecolor=edge_color, alpha=transparency)


bars = plt.hist(numeric_column, bins=n_bins, color=bar_color, edgecolor=edge_color, alpha=transparency)[2]
for i in range(len(bars)):
    # Cycle through the individual colors
    bars[i].set_facecolor(individual_colors[i % len(individual_colors)])


plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Customized Histogram for Sepal Length')
plt.show()
```

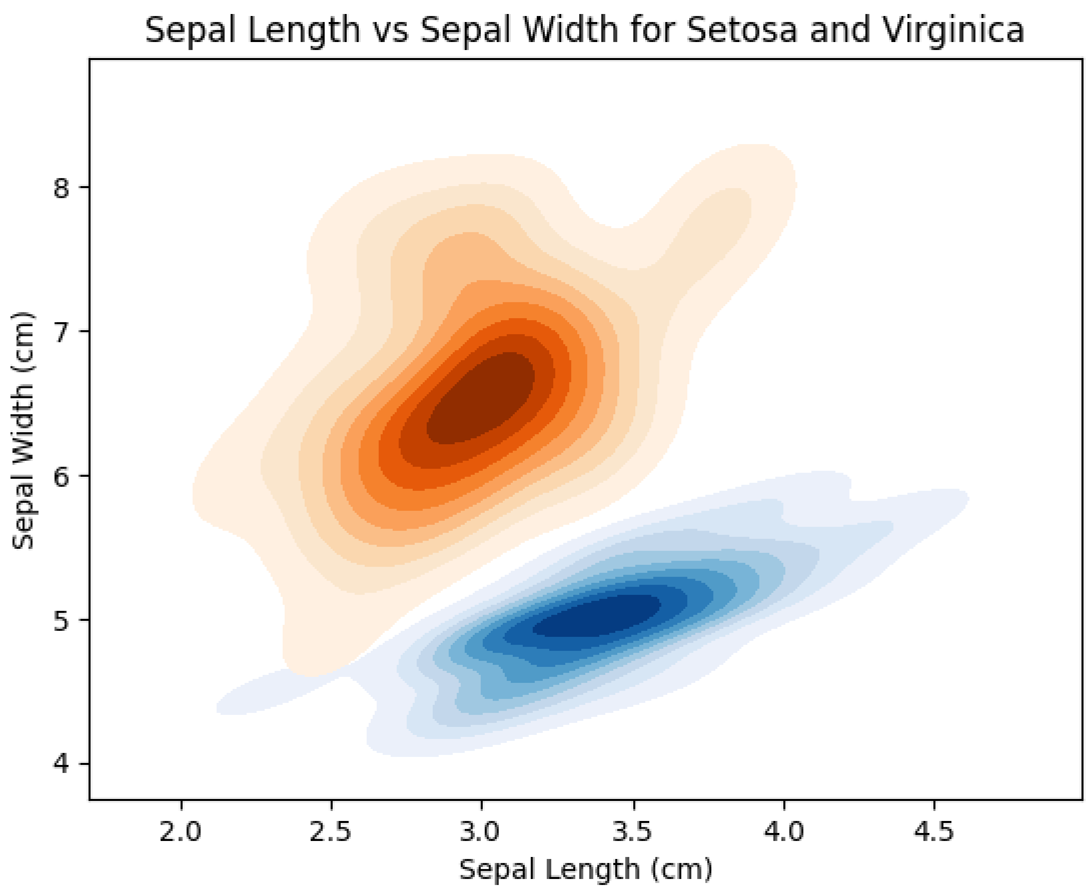


In [13]:
```
ages = np.random.randint(1, 101, size=100)


plt.figure(figsize=(10, 5))
plt.bar(range(100), ages, color='black')


plt.title('Barcoded Ages of 100 Persons')
plt.xlabel('Person')
plt.ylabel('Age')
plt.xticks([])


plt.show()
```



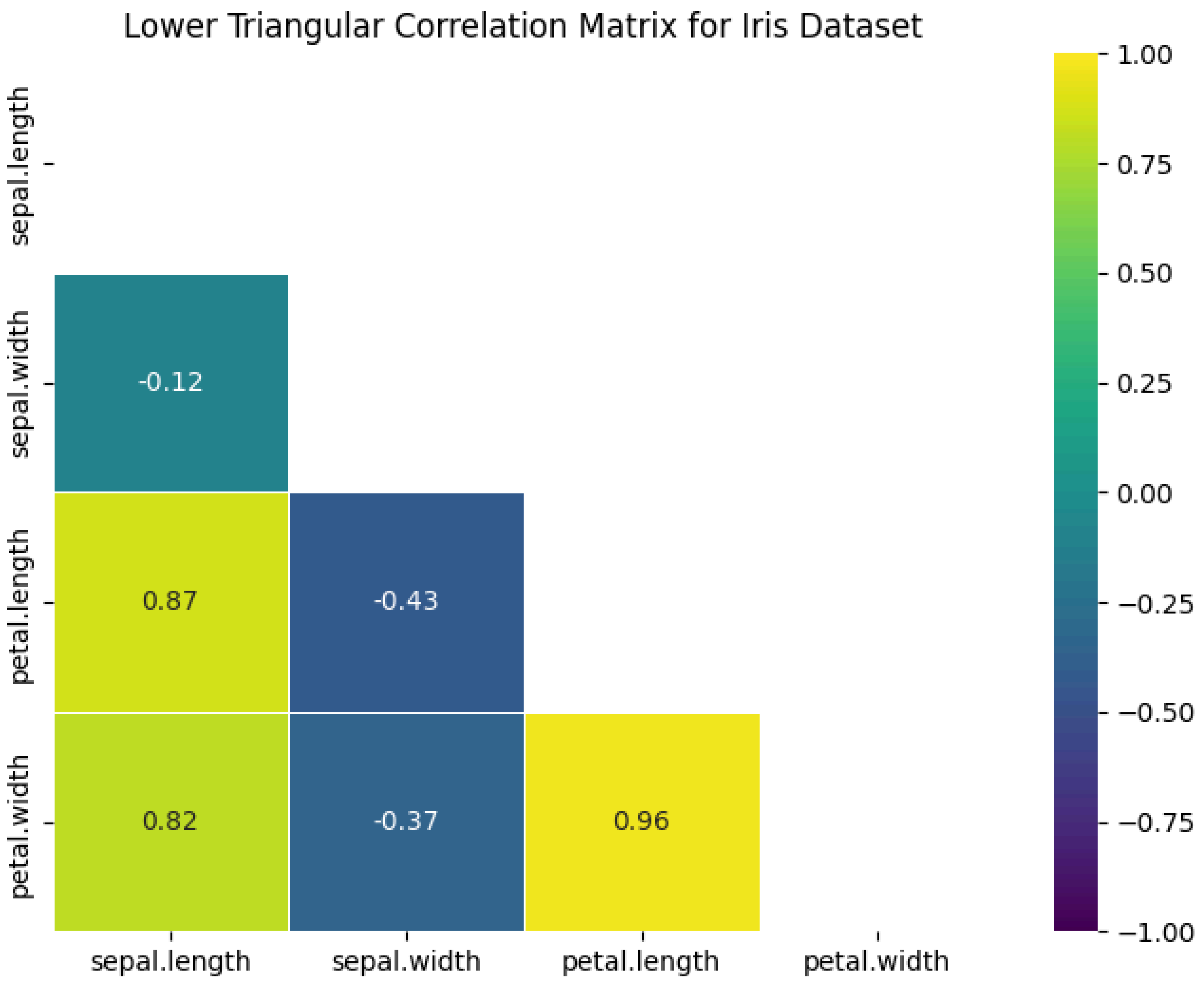
In [14]:
```
iris_setosa = df.loc[df["variety"] == "Setosa"]
iris_virginica = df.loc[df["variety"] == "Virginica"]


sns.kdeplot(
    x=iris_setosa["sepal.width"],
    y=iris_setosa["sepal.length"],
    fill=True,
    cmap="Blues",
    label="Setosa"
)
```

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


plt.xlabel("Sepal Length (cm)")
plt.ylabel("Sepal Width (cm)")


plt.title("Sepal Length vs Sepal Width for Setosa and Virginica")

plt.grid(False)


plt.show()
```



In [15]:
```
df_numeric = df.drop(columns=['variety'])


correlation_matrix = df_numeric.corr()


mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))


plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, mask=mask, annot=True, cmap='viridis', linewidths=0.5, vmin=-1, vmax=1)
plt.grid(False)
```

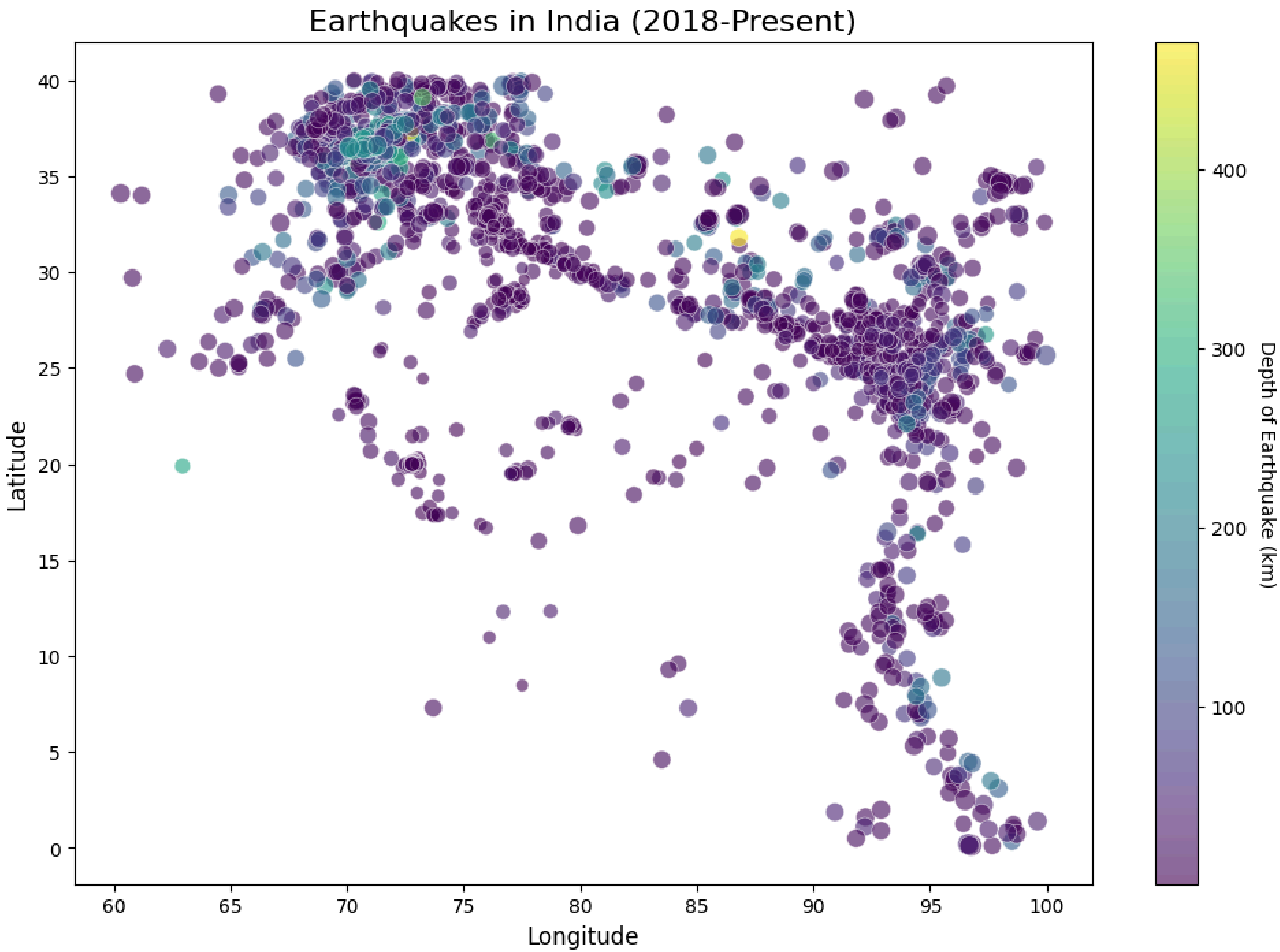
```python
plt.title('Lower Triangular Correlation Matrix for Iris Dataset')
plt.show()
```



In [16]:
```python
df = pd.read_csv(r'..\dataset\Indian_earthquake_data.csv')

df['Origin Time'] = pd.to_datetime(df['Origin Time'])


df_filtered = df[df['Origin Time'] >= '2018-01-01']
```

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


scatter = plt.scatter(
    x=df_filtered['Longitude'],
    y=df_filtered['Latitude'],
    s=df_filtered['Magnitude'] * 20,
    c=df_filtered['Depth'],
    cmap='viridis',
    alpha=0.6,
    edgecolors="w",
    linewidth=0.5
)


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


plt.title('Earthquakes in India (2018-Present)', fontsize=16)
plt.xlabel('Longitude', fontsize=12)
plt.ylabel('Latitude', fontsize=12)


plt.grid(True)

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

```
C:\Users\Ayush\AppData\Local\Temp\ipykernel_9560\2892608194.py:3: FutureWarning: Parsed string "2021-07-31 09:43:23 IST" includ
ed an un-recognized timezone "IST". Dropping unrecognized timezones is deprecated; in a future version this will raise. Instead
pass the string without the timezone, then use .tz_localize to convert to a recognized timezone.
  df['Origin Time'] = pd.to_datetime(df['Origin Time'])
```



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

In [17]:
```python
pima_df = pd.read_csv(r"..\dataset\diabetes.csv")

print("\nPima Indians Diabetes Dataset:")
pima_df.head()
```

```
Pima Indians Diabetes Dataset:
```

Out[17]:

	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 [18]:
```python
pima_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 [19]:
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```
X = pima_df[['Glucose']]
y = pima_df['Outcome']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

y_pred = linear_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (Linear Regression): {mse}")
```

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

In [20]:

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train, y_train)

y_pred_log = logistic_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_log)
print(f"Accuracy (Logistic Regression): {accuracy}")
```

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

In [21]:

```
X_multi = pima_df[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', '
y_multi = pima_df['Outcome']

X_train, X_test, y_train, y_test = train_test_split(X_multi, y_multi, test_size=0.3, random_state=42)
```

```python
lm_multi = LinearRegression()
lm_multi.fit(X_train, y_train)


y_pred_multi = lm_multi.predict(X_test)
mse_multi = mean_squared_error(y_test, y_pred_multi)
print(f"Mean Squared Error (Multiple Regression): {mse_multi}")
```

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

Linear Regression (Using Glucose):

- Mean Squared Error (MSE): 0.1831271615072512

Logistic Regression (Using Glucose):

- Accuracy: 72.29 %

Multiple Regression (Using all predictors):

- Mean Squared Error (MSE): 0.17603335005142035

Conclusion:

From the results, the logistic regression model using `Glucose` alone provides an accuracy of 72.29%, showing moderate predictive power for diagnosing diabetes. The multiple regression model, which uses more predictors, achieves an MSE of 0.17603335005142035, suggesting it might perform better in predicting diabetes outcomes compared to a simple linear regression model.