1. (10 points) Create a Python script to:

- Read a data set from a CSV file (e.g., `data.csv`).
- Compute the mean, median, mode, standard deviation, and variance for the numerical columns.
- Save the computed statistics to a new CSV file (e.g., `statistics_summary.csv`).

data.csv Link - Click here

```python
import pandas as pd
import statistics
import csv

url = 'https://raw.githubusercontent.com/gagan-iitb/DataAnalyticsAndVisualization/main/Lab-W25/data.csv'
df = pd.read_csv(url)

stat_list = []
numerical_columns = df.select_dtypes(include=['number']).columns

for column in numerical_columns:
    data = df[column].dropna()
    stats = {
        "Column": column,
        "Mean": statistics.mean(data),
        "Median": statistics.median(data),
        "Mode": statistics.mode(data),
        "Standard Deviation": statistics.stdev(data),
        "Variance": statistics.variance(data),
    }
    stat_list.append(stats)

stats_df = pd.DataFrame(stat_list)
stats_df.to_csv("statistics_summary.csv", index=False)

print(f"Statistics summary saved to statistics_summary.csv")
print(stats_df)
```

```
Statistics summary saved to statistics_summary.csv
  Column    Mean  Median  Mode  Standard Deviation     Variance
0  Value  77.519    77.0    45           42.607218  1815.375014
```

1. (10 points) Given the data set [2.0, 4.5, 4.5, 5.0, 7.0, 8.0, 10.0]:

- Calculate the mean, median, mode (Traditional Way).
- Verify your results using Python's `statistics` module.

```python
import statistics

data = [2.0, 4.5, 4.5, 5.0, 7.0, 8.0, 10.0]
```

```python
# Mean
mean_manual = sum(data) / len(data)

# Median
sorted_data = sorted(data)
n = len(sorted_data)
median_manual = sorted_data[n // 2] if n%2==1 else (sorted_data[n // 2
- 1] + sorted_data[n // 2]) / 2

# Mode
frequency = {}
for value in data:
    frequency[value] = frequency.get(value, 0) + 1
mode_manual = max(frequency, key=frequency.get)

print("Manual Calculations:")
print(f"Mean: {mean_manual}")
print(f"Median: {median_manual}")
print(f"Mode: {mode_manual}")

# Using statistics module
mean_stats = statistics.mean(data)
median_stats = statistics.median(data)
mode_stats = statistics.mode(data)

print("\nUsing statistics module:")
print(f"Mean: {mean_stats}")
print(f"Median: {median_stats}")
print(f"Mode: {mode_stats}")
```

```
Manual Calculations:
Mean: 5.857142857142857
Median: 5.0
Mode: 4.5

Using statistics module:
Mean: 5.857142857142857
Median: 5.0
Mode: 4.5
```

Numpy Assignment

Part 1: Exploring Dimensions and Shapes (20 points)

1. Create an array of shape (4, 3, 2) representing a 3D matrix.

2. Print the shape and the number of dimensions (ndim) of the array.

3. Reshape the array into a 2D matrix of shape (12, 2) and verify the new shape and number of dimensions.

4. Create a scalar array with the value 7 and print its shape and dimensions.

```
import numpy as np

array_3d = np.arange(24).reshape(4, 3, 2)

print("3D Array:")
print(array_3d)
print(f"Shape: {array_3d.shape}")
print(f"Number of dimensions: {array_3d.ndim}\n")

array_2d = array_3d.reshape(12, 2)

print("Reshaped to 2D Array:")
print(array_2d)
print(f"Shape: {array_2d.shape}")
print(f"Number of dimensions: {array_2d.ndim}\n")

array_1d = np.array(7)

print("Scalar Array:")
print(array_1d)
print(f"Shape: {array_1d.shape}")
print(f"Number of dimensions: {array_1d.ndim}")
```

```
3D Array:
[[[ 0  1]
  [ 2  3]
  [ 4  5]]

 [[ 6  7]
  [ 8  9]
  [10 11]]

 [[12 13]
  [14 15]
  [16 17]]

 [[18 19]
  [20 21]
  [22 23]]]
Shape: (4, 3, 2)
Number of dimensions: 3

Reshaped to 2D Array:
[[ 0  1]
 [ 2  3]
 [ 4  5]
 [ 6  7]
 [ 8  9]
 [10 11]
 [12 13]
 [14 15]
 [16 17]
 [18 19]
 [20 21]
 [22 23]]
Shape: (12, 2)
Number of dimensions: 2

Scalar Array:
7
Shape: ()
Number of dimensions: 0
```

Part 2: Matrix Manipulations (15 points)

1. Create a random 4x4 matrix and print its values. Then, find the sum of the elements across both axes (i.e., sum along rows and sum along columns).

2. Create a 5x5 identity matrix and modify the middle element (at position [2,2]) to 99. Print the modified matrix.

3. Create a 3x3 matrix with random values, then flip the matrix left-right to get the opposite diagonal (anti-diagonal).

```python
import numpy as np

# Part 1: Create a random 4x4 matrix and find the sum across axes
random_matrix = np.random.rand(4, 4)
print("Random 4x4 Matrix:")
print(random_matrix)


sum_rows = np.sum(random_matrix, axis=1)
sum_columns = np.sum(random_matrix, axis=0)

print("\nSum along rows:", sum_rows)
print("Sum along columns:", sum_columns)

# Part 2: Create a 5x5 identity matrix and modify the middle element
identity_matrix = np.eye(5)
identity_matrix[2, 2] = 99
print("\nModified 5x5 Identity Matrix:")
print(identity_matrix)

# Part 3: Create a 3x3 random matrix and flip it left-right
random_3x3_matrix = np.random.rand(3, 3)
print("\nRandom 3x3 Matrix:")
print(random_3x3_matrix)
```

```python
# Flip the matrix left-right to get the anti-diagonal
flipped_matrix = np.fliplr(random_3x3_matrix)
print("\nFlipped Matrix (Anti-Diagonal):")
print(flipped_matrix)
```

```
Random 4x4 Matrix:
[[0.61533392 0.88292712 0.47558652 0.11647848]
 [0.98589121 0.12420045 0.58445603 0.00294031]
 [0.2916423  0.76584451 0.62439872 0.49088585]
 [0.4008244  0.91078792 0.25025465 0.02269673]]

Sum along rows: [2.09032604 1.697488   2.17277139 1.58456369]
Sum along columns: [2.29369183 2.68375999 1.93469592 0.63300138]

Modified 5x5 Identity Matrix:
[[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0. 99.  0.  0.]
 [ 0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  1.]]

Random 3x3 Matrix:
[[0.52711695 0.08386777 0.37232185]
 [0.44755518 0.80745554 0.85148271]
 [0.04908742 0.17367371 0.60144114]]

Flipped Matrix (Anti-Diagonal):
[[0.37232185 0.08386777 0.52711695]
 [0.85148271 0.80745554 0.44755518]
 [0.60144114 0.17367371 0.04908742]]
```

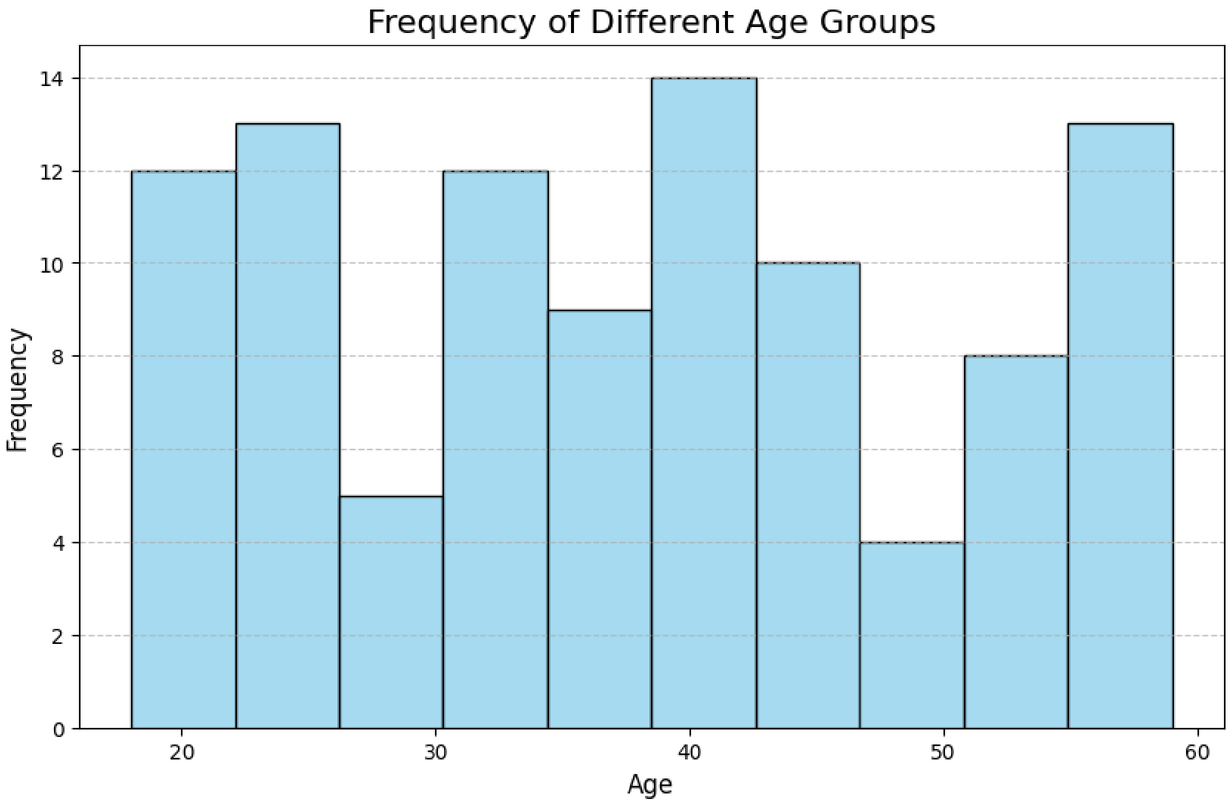
Data Visualization (15 points)

```python
import pandas as pd
import numpy as np

# Set a seed for reproducibility
np.random.seed(42)

# Generate a random dataset
num_samples = 100

data = {
    "ID": range(1, num_samples + 1),
    "Age": np.random.randint(18, 60, size=num_samples),
    "Height_cm": np.random.normal(165, 10, num_samples).round(1),
    "Weight_kg": np.random.normal(70, 15, num_samples).round(1),
    "City": np.random.choice(["New York", "Los Angeles", "Chicago",
"Houston", "Phoenix"], size=num_samples),
    "Grade": np.random.choice(["A", "B", "C", "D", "F"],
size=num_samples),
    "Monthly_Income": np.random.randint(2000, 10000, 
size=num_samples),
    "Hours_Studied": np.random.exponential(5, num_samples).round(1),
    "Passed": np.random.choice(["Yes", "No"], size=num_samples, 
p=[0.8, 0.2]),
    "Category": np.random.choice(["Category 1", "Category 2", 
"Category 3"], size=num_samples),
    "Test_Score": np.random.uniform(50, 100, 
size=num_samples).round(2),

    "Exercise_Hours": np.random.poisson(3, num_samples),
    "Favorite_Color": np.random.choice(["Red", "Blue", "Green", 
"Yellow", "Purple"], size=num_samples),
}

df = pd.DataFrame(data)

# Save to a CSV for reuse
df.to_csv("random_plotting_dataset.csv", index=False)

# Display the first few rows
print(df.head())

   ID  Age  Height_cm  Weight_kg         City Grade  Monthly_Income  \
0   1   56      163.3       67.8      Chicago     A            3852   
1   2   46      176.6       72.7      Houston     A            6910   
2   3   32      162.5       84.5  Los Angeles     A            7268   
3   4   25      157.3       54.0      Chicago     D            6175   
4   5   38      177.1       71.6      Phoenix     A            4933   

   Hours_Studied Passed    Category  Test_Score  Exercise_Hours 
Favorite_Color  
0            9.2    Yes  Category 2       64.40               6 
Blue  
1            9.1    Yes  Category 1       80.75               2 
Blue  
2            3.2    Yes  Category 3       95.59               3 
Blue  
3            2.7    Yes  Category 1       56.96               7 
Blue  
4            1.6    Yes  Category 1       55.04               0 
Blue  
```

Create a histogram to show the frequency of different age groups.

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
```

```
sns.histplot(df['Age'], bins=10, color='skyblue', edgecolor='black')
plt.title('Frequency of Different Age Groups', fontsize=16)
plt.xlabel('Age', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

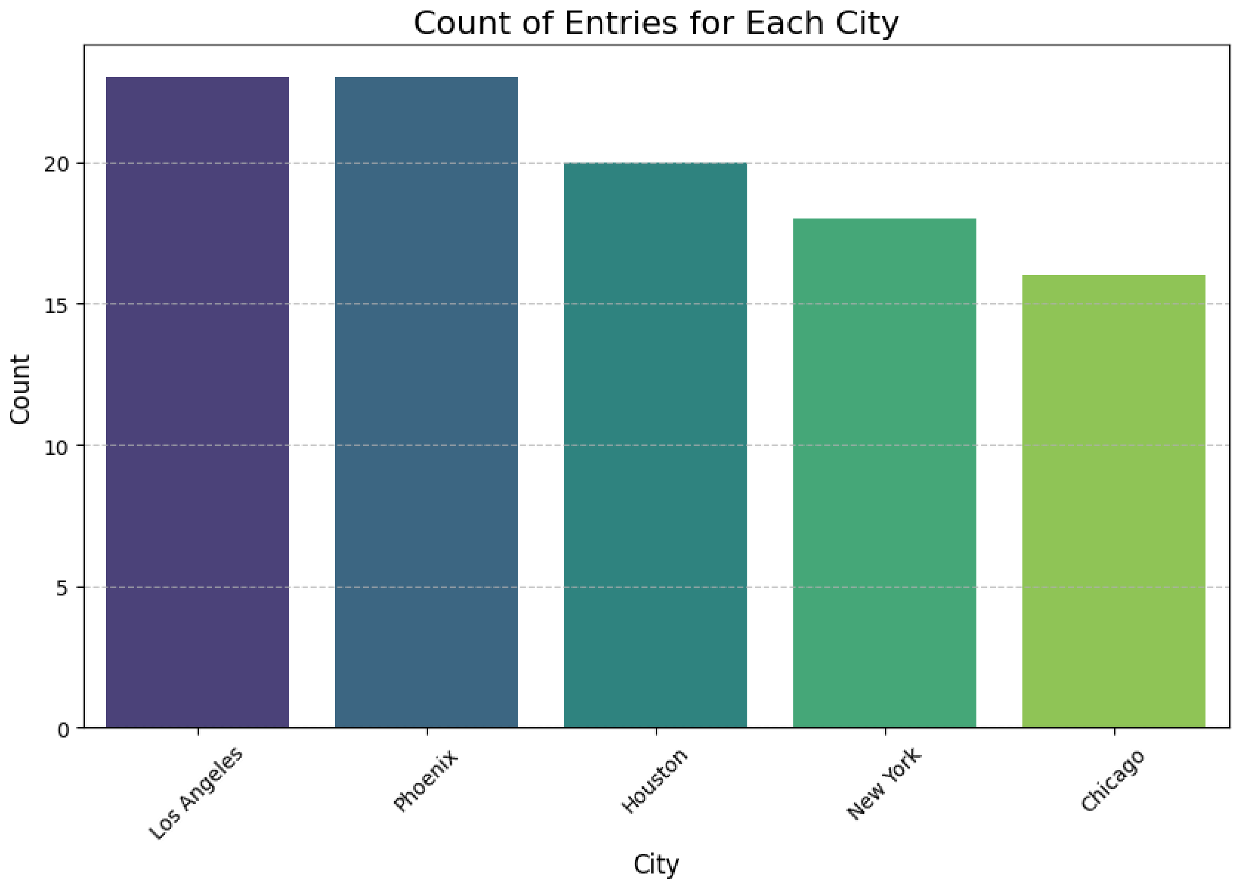


Use a bar chart to display the count of entries for each City.

```
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='City', palette='viridis',
order=df['City'].value_counts().index)
plt.title('Count of Entries for Each City', fontsize=16)
plt.xlabel('City', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=45)
plt.show()
```

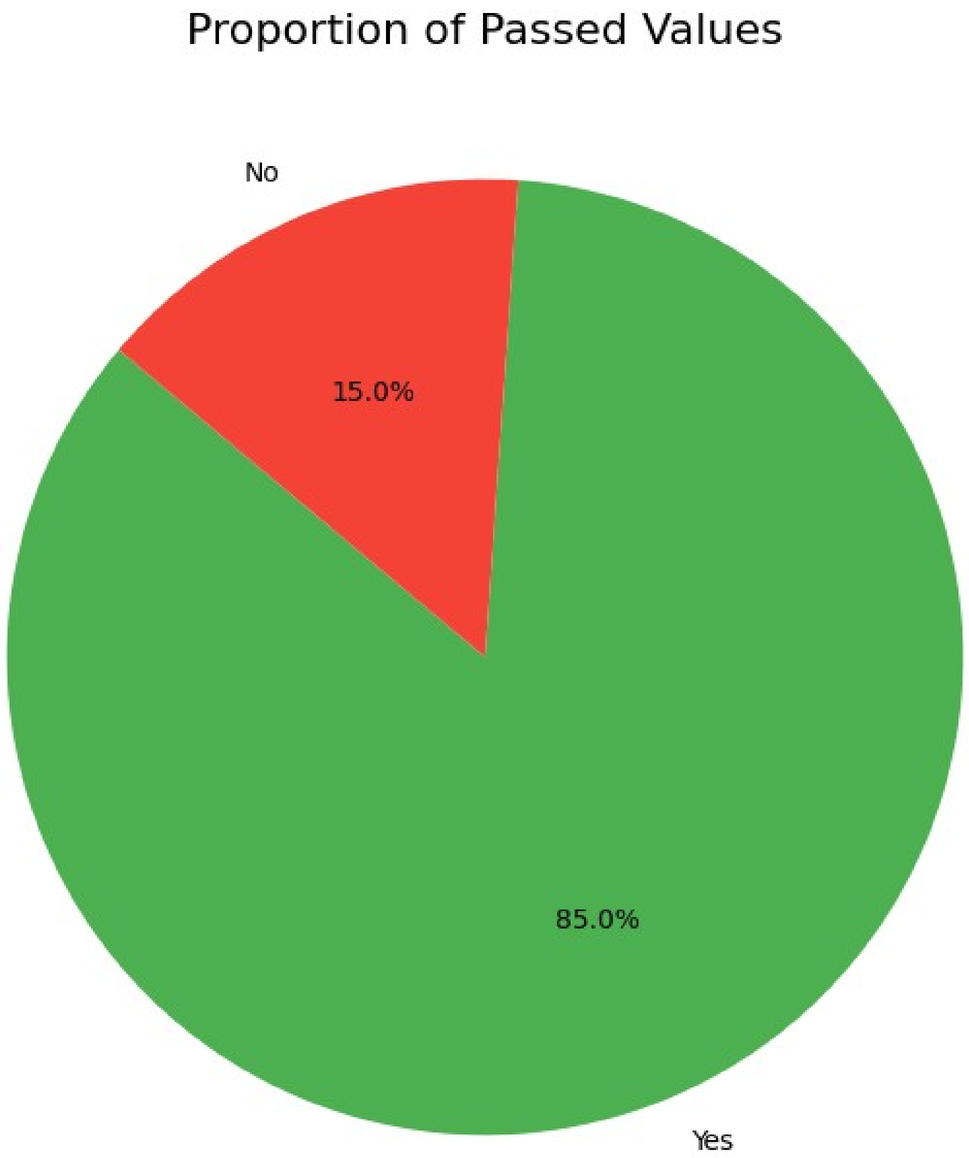
```
<ipython-input-77-52b2299ea06b>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.
```

```
  sns.countplot(data=df, x='City', palette='viridis',
order=df['City'].value_counts().index)
```



Create a pie chart to show the proportion of Passed values.

```
passed_counts = df['Passed'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(
    passed_counts,
    labels=passed_counts.index,
    autopct='%1.1f%%',
    startangle=140,
    colors=['#4caf50', '#f44336']
)
plt.title('Proportion of Passed Values', fontsize=16)
plt.show()
```



Matrix Problem: (10+40 = 50 points)

1. Calculate the rank of a 15 * 15 binary random matrix, similar to what we did in the activity. But this time, we will go to each cell and generate a 1 with probability "p".
2. Now generate a binary random matrix of size 1920 x 1080. Imagine 1s to be land and 0s to be water. Write a program to count the number of islands in your matrix. Study the following: number of islands, size of the largest island and plot them for varying values of "p".

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import svd
```

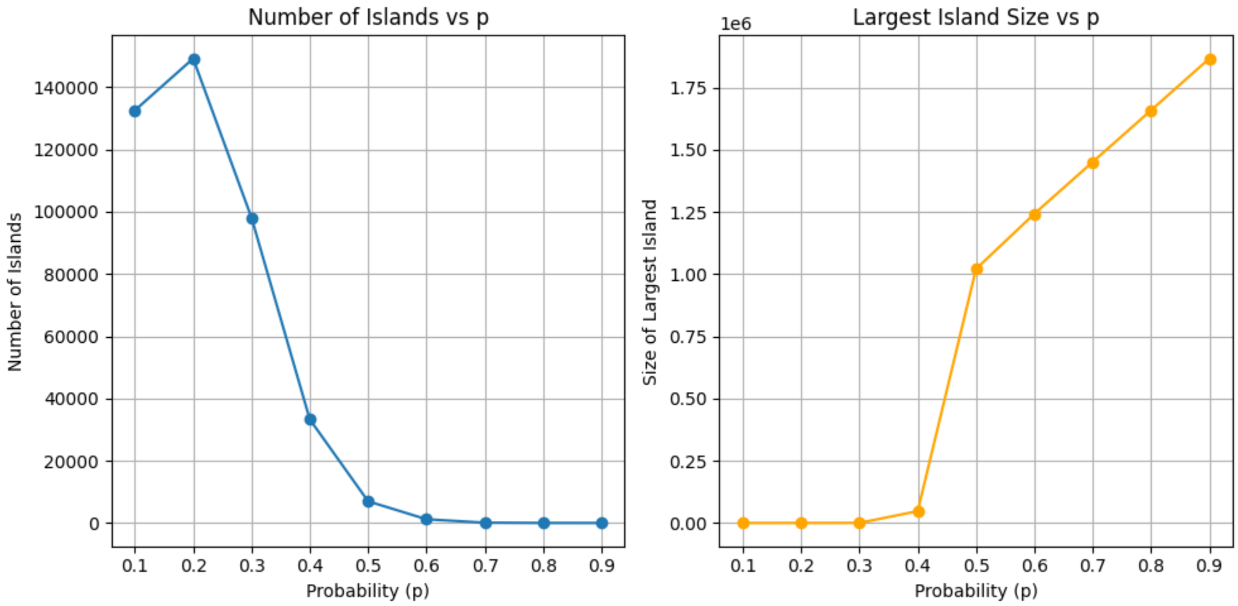
```python
def calculate_matrix_rank(p, size):
    matrix = np.random.choice([0, 1], size=size, p=[1-p, p])
    rank = np.linalg.matrix_rank(matrix)
    return matrix, rank

def count_islands(matrix):
    rows, cols = matrix.shape
    visited = np.zeros_like(matrix, dtype=bool)

    def iterative_dfs(start_x, start_y):
        stack = [(start_x, start_y)]
        size = 0
        while stack:
            x, y = stack.pop()
            if x < 0 or y < 0 or x >= rows or y >= cols or visited[x, y] or matrix[x, y] == 0:
                continue
            visited[x, y] = True
            size += 1
            for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1)]:
                stack.append((x + dx, y + dy))
        return size

    num_islands = 0
    max_size = 0

    for i in range(rows):
        for j in range(cols):
            if matrix[i, j] == 1 and not visited[i, j]:
                num_islands += 1
                max_size = max(max_size, iterative_dfs(i, j))

    return num_islands, max_size

def plot_islands(p_values, num_islands_list, max_sizes_list):
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.plot(p_values, num_islands_list, marker='o', label="Number of Islands")
    plt.xlabel("Probability (p)")
    plt.ylabel("Number of Islands")
    plt.title("Number of Islands vs p")
    plt.grid(True)

    plt.subplot(1, 2, 2)
    plt.plot(p_values, max_sizes_list, marker='o', label="Largest Island Size", color='orange')
    plt.xlabel("Probability (p)")
    plt.ylabel("Size of Largest Island")
    plt.title("Largest Island Size vs p")
    plt.grid(True)

    plt.tight_layout()
    plt.show()

p = 0.5
binary_matrix, rank = calculate_matrix_rank(p, (15, 15))
print("Binary Matrix (15x15):\n", binary_matrix)
print("Rank of the matrix:", rank)
```

```
Binary Matrix (15x15):
 [[0 1 0 0 0 0 0 1 1 1 0 1 0 0 1]
 [0 1 1 0 0 0 1 0 0 0 1 0 0 1 1]
 [0 0 0 1 0 1 1 1 0 1 1 1 0 1 0]
 [0 0 1 0 0 0 0 1 1 0 0 1 0 0 1]
 [1 0 1 0 1 0 0 0 1 1 0 0 1 0 0]
 [1 0 1 1 0 0 0 1 1 1 0 1 1 1 1]
 [1 1 1 0 0 1 0 1 1 1 0 0 0 1 1]
 [1 0 1 1 0 1 0 0 1 0 1 1 1 0 1]
 [1 1 0 0 1 0 1 1 1 1 1 1 1 1 0]
 [0 1 0 0 0 1 1 1 1 1 0 1 0 1 1]
 [0 0 0 0 0 1 1 1 1 1 1 0 1 0 1]
 [1 1 0 1 1 0 0 1 0 1 0 0 1 0 1]
 [0 0 1 1 1 1 1 0 0 1 1 1 0 0 1]
 [1 1 1 1 0 0 0 0 1 0 1 0 0 0 0]
 [1 0 1 0 0 0 1 1 0 0 1 0 0 1 1]]
Rank of the matrix: 15
```

```
p_values = np.linspace(0.1, 0.9, 9)
num_islands_list = []
largest_islands = []

for p in p_values:
  large_matrix, rank = calculate_matrix_rank(p, (1920, 1080))
  num_islands, max_size = count_islands(large_matrix)
  num_islands_list.append(num_islands)
  largest_islands.append(max_size)
  print(f"p = {p:.1f} | Number of Islands: {num_islands} | Largest
Island Size: {max_size}")

plot_islands(p_values, num_islands_list, largest_islands)
```

```
p = 0.1 | Number of Islands: 132567 | Largest Island Size: 17
p = 0.2 | Number of Islands: 149228 | Largest Island Size: 52
p = 0.3 | Number of Islands: 98141 | Largest Island Size: 272
p = 0.4 | Number of Islands: 33285 | Largest Island Size: 47517
p = 0.5 | Number of Islands: 6968 | Largest Island Size: 1021064
```

```
p = 0.6 | Number of Islands: 1196 | Largest Island Size: 1242581
p = 0.7 | Number of Islands: 96 | Largest Island Size: 1450951
p = 0.8 | Number of Islands: 7 | Largest Island Size: 1658524
p = 0.9 | Number of Islands: 1 | Largest Island Size: 1866582
```



```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import linregress, spearmanr, pearsonr

# Statistical summary
def compute_statistics(p_values, num_islands, largest_islands):
    num_islands_stats = {
        "Mean": np.mean(num_islands),
        "Standard Deviation": np.std(num_islands),
        "Variance": np.var(num_islands)
    }
    largest_islands_stats = {
        "Mean": np.mean(largest_islands),
        "Standard Deviation": np.std(largest_islands),
        "Variance": np.var(largest_islands)
    }

    print("\nStatistical Summary:")
    print("Number of Islands Statistics:", num_islands_stats)
    print("Largest Island Size Statistics:", largest_islands_stats)
    return num_islands_stats, largest_islands_stats

compute_statistics(p_values, num_islands, largest_islands)
```

```
Statistical Summary:
Number of Islands Statistics: {'Mean': 1.0, 'Standard Deviation': 0.0,
'Variance': 0.0}
Largest Island Size Statistics: {'Mean': 809728.8888888889, 'Standard
Deviation': 747438.4657030592, 'Variance': 558664260012.5432}
```

```
({'Mean': 1.0, 'Standard Deviation': 0.0, 'Variance': 0.0},
 {'Mean': 809728.8888888889,
  'Standard Deviation': 747438.4657030592,
  'Variance': 558664260012.5432})
```

```
# Optimal p value
def find_optimal_p(p_values, num_islands, largest_islands):
    max_islands_p = p_values[np.argmax(num_islands)]
    max_islands_value = np.max(num_islands)

    largest_island_p = p_values[np.argmax(largest_islands)]
    largest_island_value = np.max(largest_islands)

    print("\nOptimal Probability Analysis:")
    print(f"p with Maximum Number of Islands: {max_islands_p} (Value:
{max_islands_value})")
    print(f"p with Largest Island Size: {largest_island_p} (Value:
{largest_island_value})")
find_optimal_p(p_values, num_islands, largest_islands)
```

```
Optimal Probability Analysis:
p with Maximum Number of Islands: 0.1 (Value: 1)
p with Largest Island Size: 0.9 (Value: 1866582)
```