Robot Programming

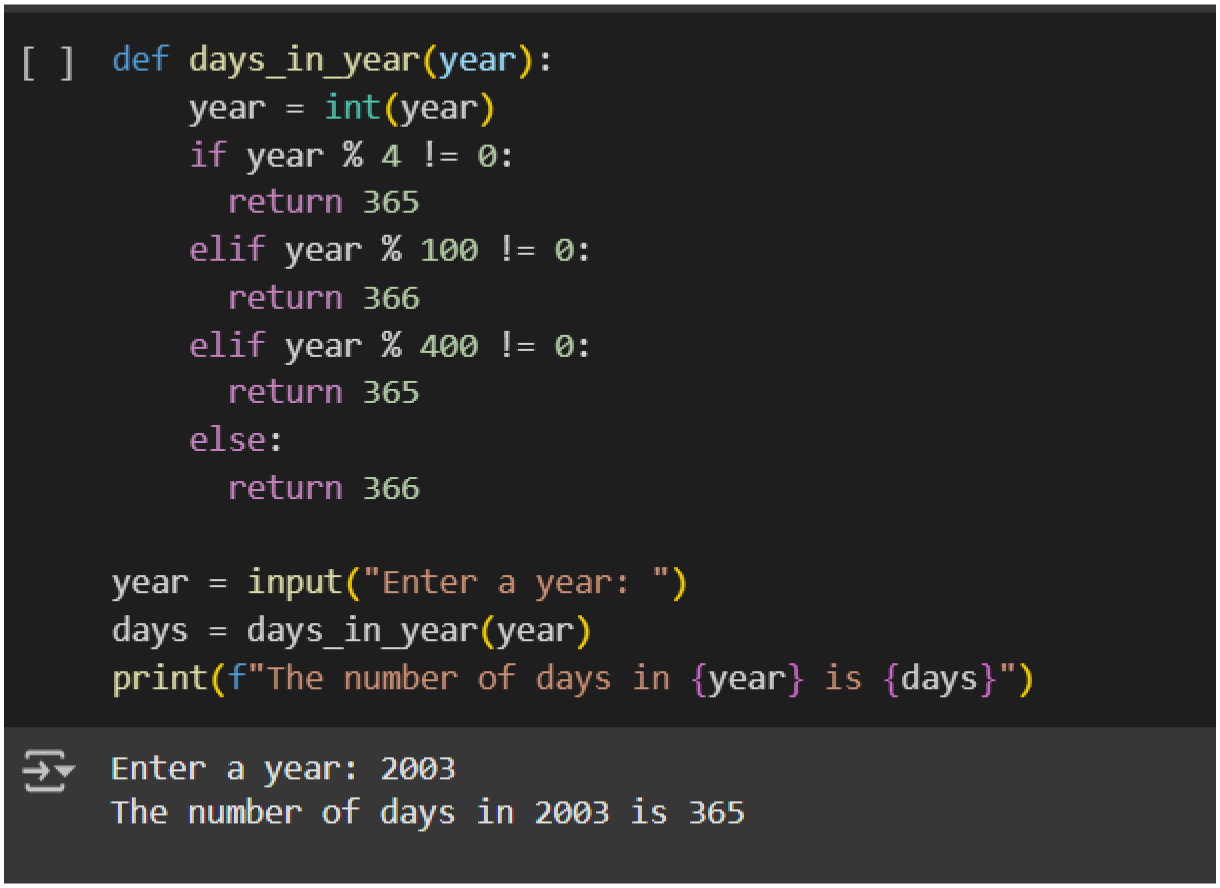
Lab-1

Team Members:

Prachi Chaudhari - 202201265
Ansh Pandya - 202201303

Google Colab Link:
https://colab.research.google.com/drive/1v3swZe2NFLrpN9gSkjQhq151xmhiLQA0#scrollTo=98pa3JFJ9QFc

Q1. Write a function that gives the number of days of a given year.
Input : 1990 Input : 2044
Output : 365 Output : 366



```
[ ]  def days_in_year(year):
         year = int(year)
         if year % 4 != 0:
           return 365
         elif year % 100 != 0:
           return 366
         elif year % 400 != 0:
           return 365
         else:
           return 366

     year = input("Enter a year: ")
     days = days_in_year(year)
     print(f"The number of days in {year} is {days}")
```

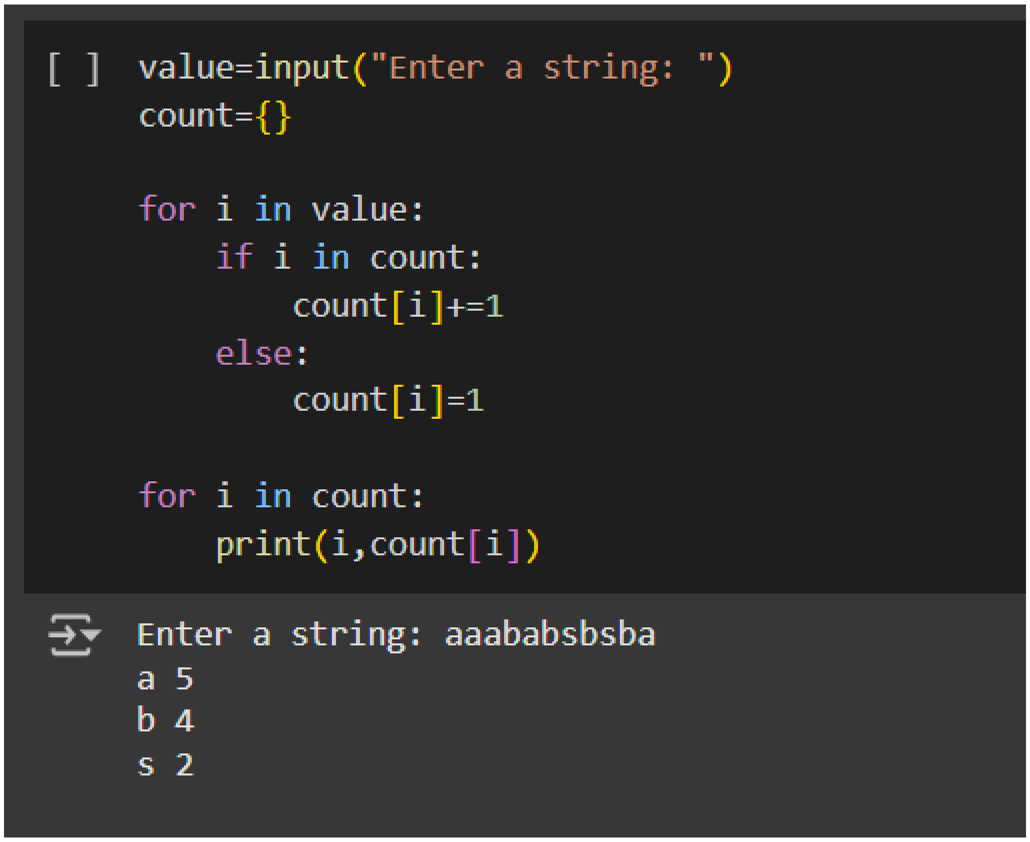
```
Enter a year: 2003
The number of days in 2003 is 365
```

Explanation:

This code checks if a year is a leap year:

1. Leap year: If divisible by 4, but not by 100, unless divisible by 400.
2. Not a leap year: If divisible by 100 but not by 400, or not divisible by 4.

Q2. Count the frequency of each character in a string and store it in a dictionary. An example is given below. Input: ‘adcbbdaacd’ Output: {‘a’: 3, ‘b’: 2, ‘c’: 2, ‘d’: 3}



```
value=input("Enter a string: ")
count={}

for i in value:
    if i in count:
        count[i]+=1
    else:
        count[i]=1

for i in count:
    print(i,count[i])
```

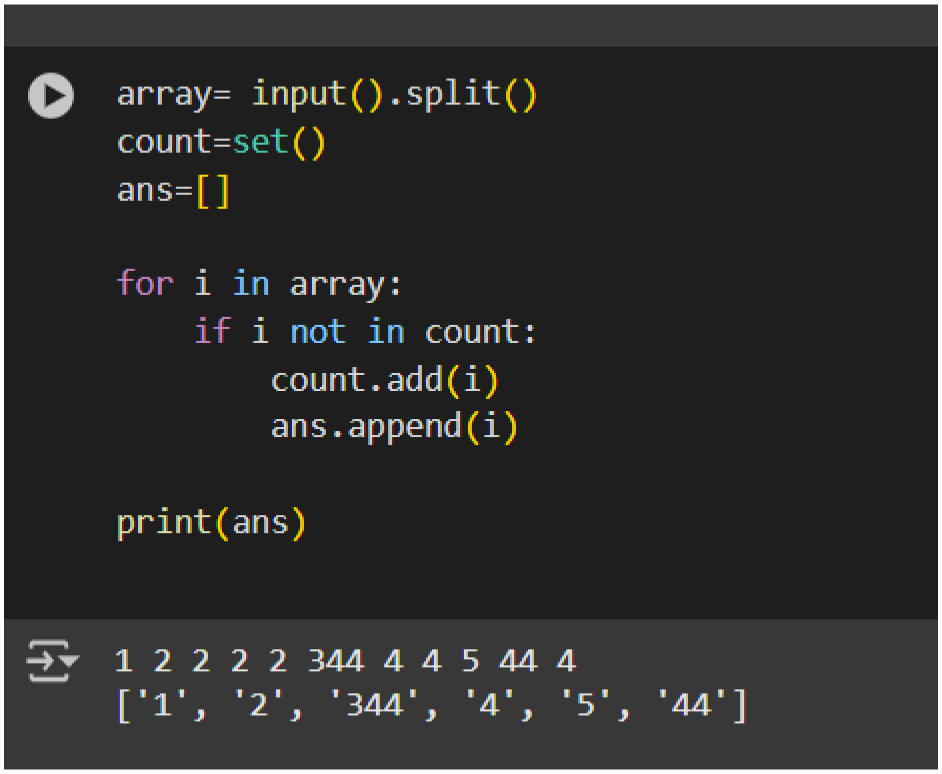
```
Enter a string: aaababsbsba
a 5
b 4
s 2
```

Explanation:

Here, count[i] will track the number of occurrences of a letter in the string using the for loop. If the letter is encountered for the first time it will set value 1 and else if it has been encountered before it will increase the count by 1.

Q3. Write a program to remove duplicates from a list but keep the first occurrence of each element.
Input: [1, 2, 3, 4, 2, 3, 5, 6, 1, 4]

Output: [1, 2, 3, 4, 5, 6]



```
array= input().split()
count=set()
ans=[]

for i in array:
    if i not in count:
        count.add(i)
        ans.append(i)

print(ans)
```

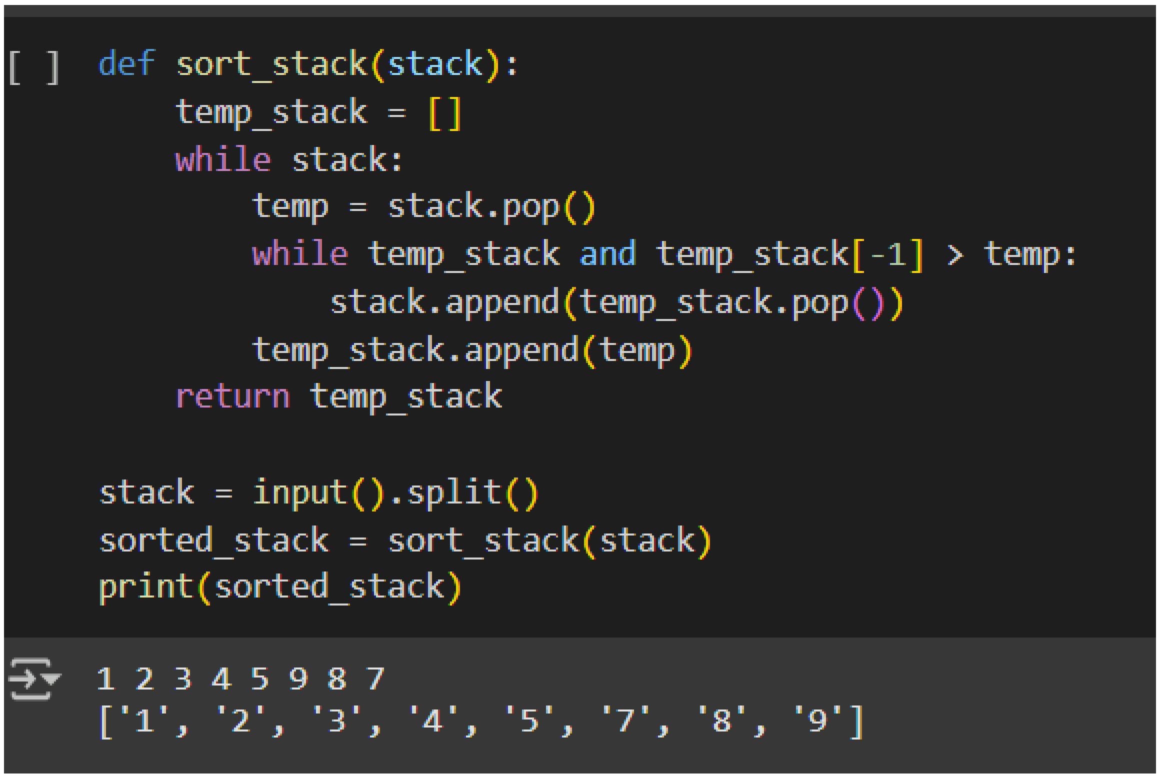
```
1 2 2 2 2 344 4 4 5 44 4
['1', '2', '344', '4', '5', '44']
```

Explanation:

Here the count has a set data type where the occurrence of the number from the array is tracked. “ans” is an empty array where the number is added if it is not present in the existing set and added to the set too. If the element is present in the set it won’t add to the array and it will continue checking for the next element. This way the “ans” will have only the unique occurrences of each element.

Q4. Write a program to sort a stack using only another stack (no other data structures like arrays or linked lists).
Input: stack = [9, 5, 1, 3]

Output: stack = [1, 3, 5, 9]



```
[ ]  def sort_stack(stack):
         temp_stack = []
         while stack:
             temp = stack.pop()
             while temp_stack and temp_stack[-1] > temp:
                 stack.append(temp_stack.pop())
             temp_stack.append(temp)
         return temp_stack

     stack = input().split()
     sorted_stack = sort_stack(stack)
     print(sorted_stack)
```

```
1 2 3 4 5 9 8 7
['1', '2', '3', '4', '5', '7', '8', '9']
```

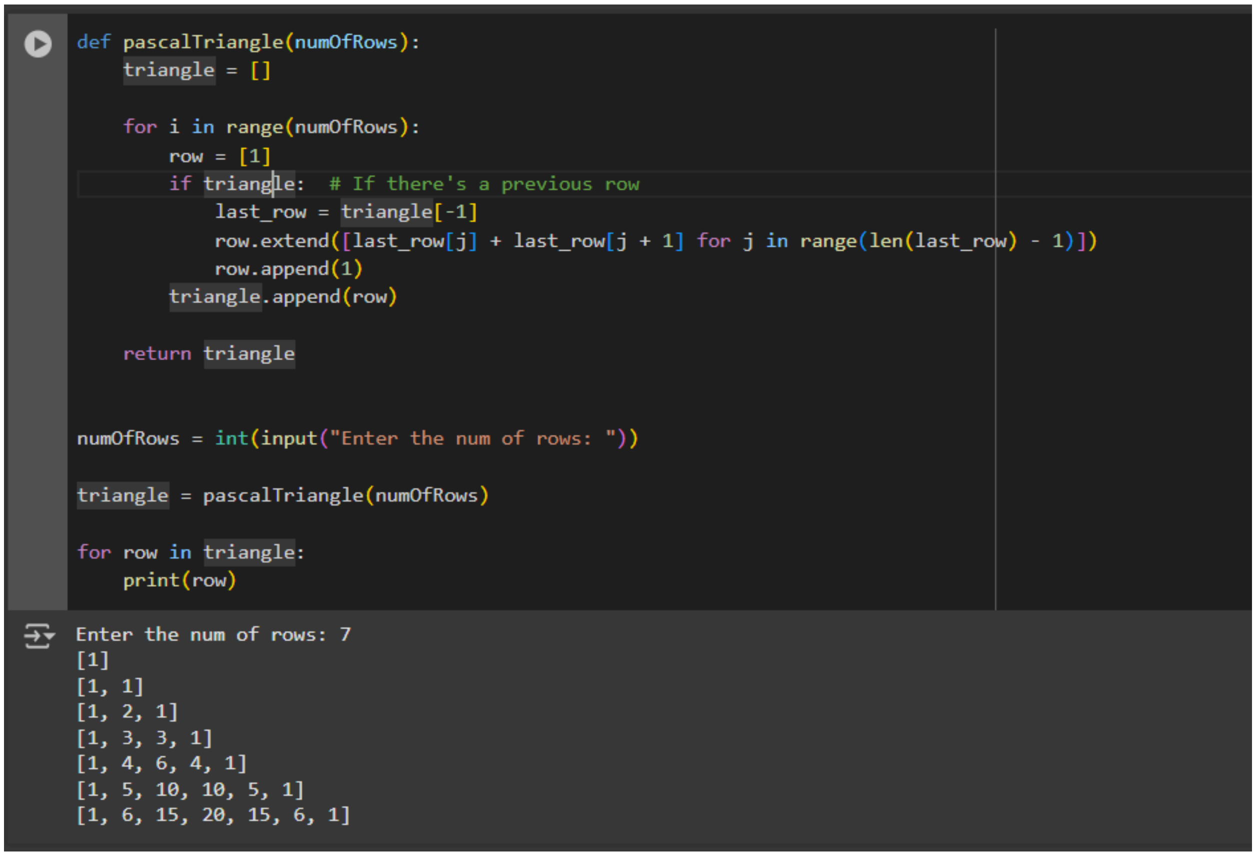
Explanation:

This code sorts a stack (list) using an auxiliary stack (`temp_stack`) instead of built-in sorting functions. Here's a short breakdown of the logic:

1. Initialize an empty `temp_stack` to store sorted elements.
2. Pop elements from `stack` one by one and store them in `temp_stack` in sorted order.
3. If the top of `temp_stack` is greater than the current element, move elements back to `stack` until the correct position is found.

4. Push the current element into `temp_stack`. This ensures `temp_stack` always remains sorted.
5. Repeat until the stack is empty.
6. Return `temp_stack` as the sorted stack.

Q5. Make a module “pascal.py” with function “pascalTriangle(numOfRows)” and import into “main.py”. Input : Enter the num of rows : 7



```python
def pascalTriangle(numOfRows):
    triangle = []

    for i in range(numOfRows):
        row = [1]
        if triangle:  # If there's a previous row
            last_row = triangle[-1]
            row.extend([last_row[j] + last_row[j + 1] for j in range(len(last_row) - 1)])
            row.append(1)
        triangle.append(row)

    return triangle


numOfRows = int(input("Enter the num of rows: "))

triangle = pascalTriangle(numOfRows)

for row in triangle:
    print(row)
```

```
Enter the num of rows: 7
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, 6, 15, 20, 15, 6, 1]
```

Explanation:

Initialize an empty list `triangle` to store the rows.

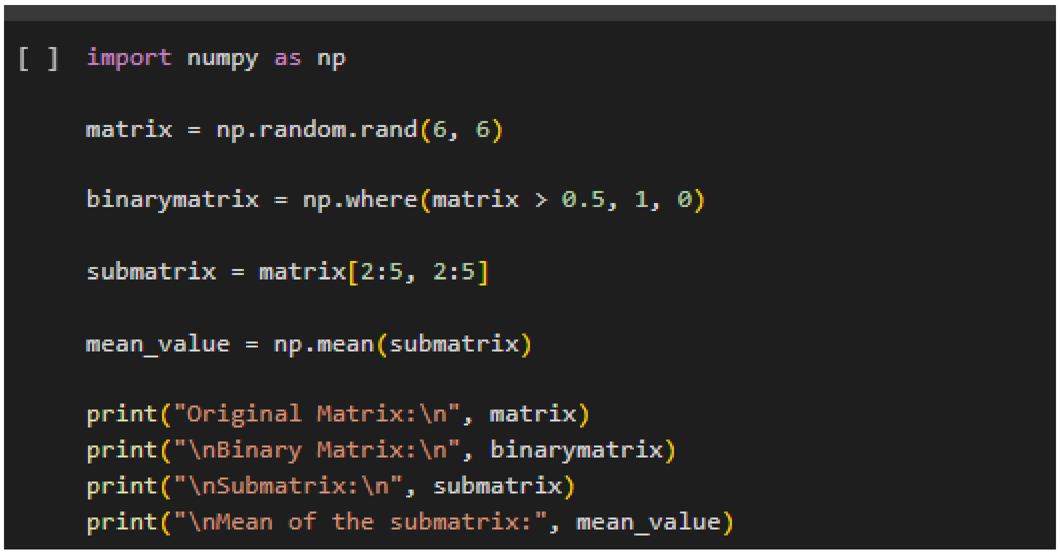
Loop through `numOfRows` to generate each row.

Start each row with `[1]`.
If there's a previous row (`triangle` is not empty):

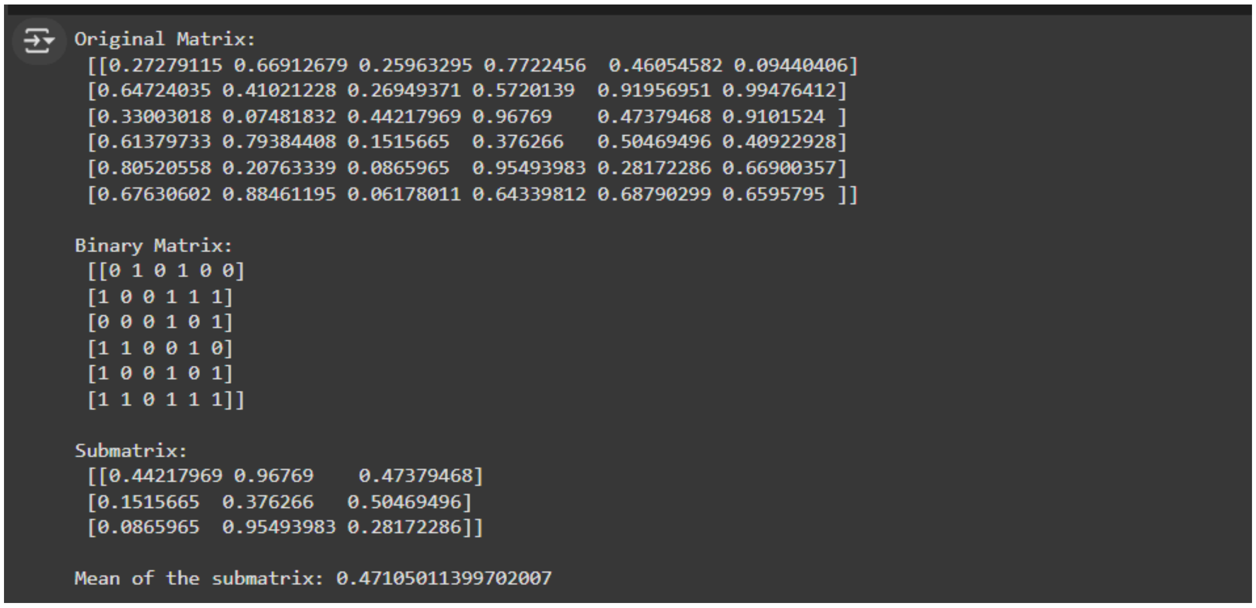
- Retrieve the last row.
- Compute new values by summing adjacent pairs from the last row.
- Append 1 at the end of the row.

Store the generated row in a triangle.
After generating all rows, print the triangle.

Q6. Create a 6x6 matrix with random values and: Replace all values greater than 0.5 with 1, and all others with 0. Extract a 3x3 submatrix starting from index (2, 2) and calculate its mean.



```
[ ]  import numpy as np

     matrix = np.random.rand(6, 6)

     binarymatrix = np.where(matrix > 0.5, 1, 0)

     submatrix = matrix[2:5, 2:5]

     mean_value = np.mean(submatrix)

     print("Original Matrix:\n", matrix)
     print("\nBinary Matrix:\n", binarymatrix)
     print("\nSubmatrix:\n", submatrix)
     print("\nMean of the submatrix:", mean_value)
```

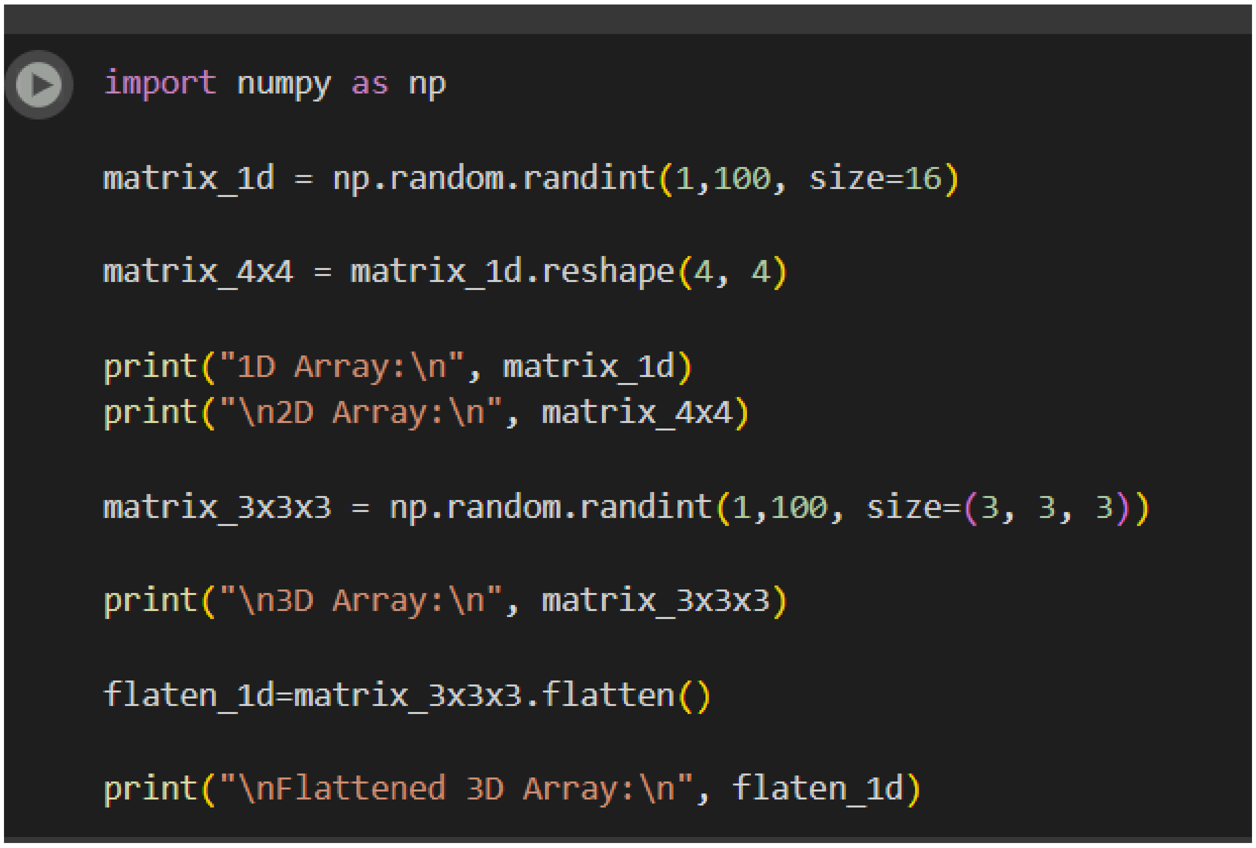


```
Original Matrix:
 [[0.27279115 0.66912679 0.25963295 0.7722456  0.46054582 0.09440406]
 [0.64724035 0.41021228 0.26949371 0.5720139  0.91956951 0.99476412]
 [0.33003018 0.07481832 0.44217969 0.96769    0.47379468 0.9101524 ]
 [0.61379733 0.79384408 0.1515665  0.376266   0.50469496 0.40922928]
 [0.80520558 0.20763339 0.0865965  0.95493983 0.28172286 0.66900357]
 [0.67630602 0.88461195 0.06178011 0.64339812 0.68790299 0.6595795 ]]

Binary Matrix:
 [[0 1 0 1 0 0]
 [1 0 0 1 1 1]
 [0 0 0 1 0 1]
 [1 1 0 0 1 0]
 [1 0 0 1 0 1]
 [1 1 0 1 1 1]]

Submatrix:
 [[0.44217969 0.96769    0.47379468]
 [0.1515665  0.376266   0.50469496]
 [0.0865965  0.95493983 0.28172286]]

Mean of the submatrix: 0.47105011399702007
```

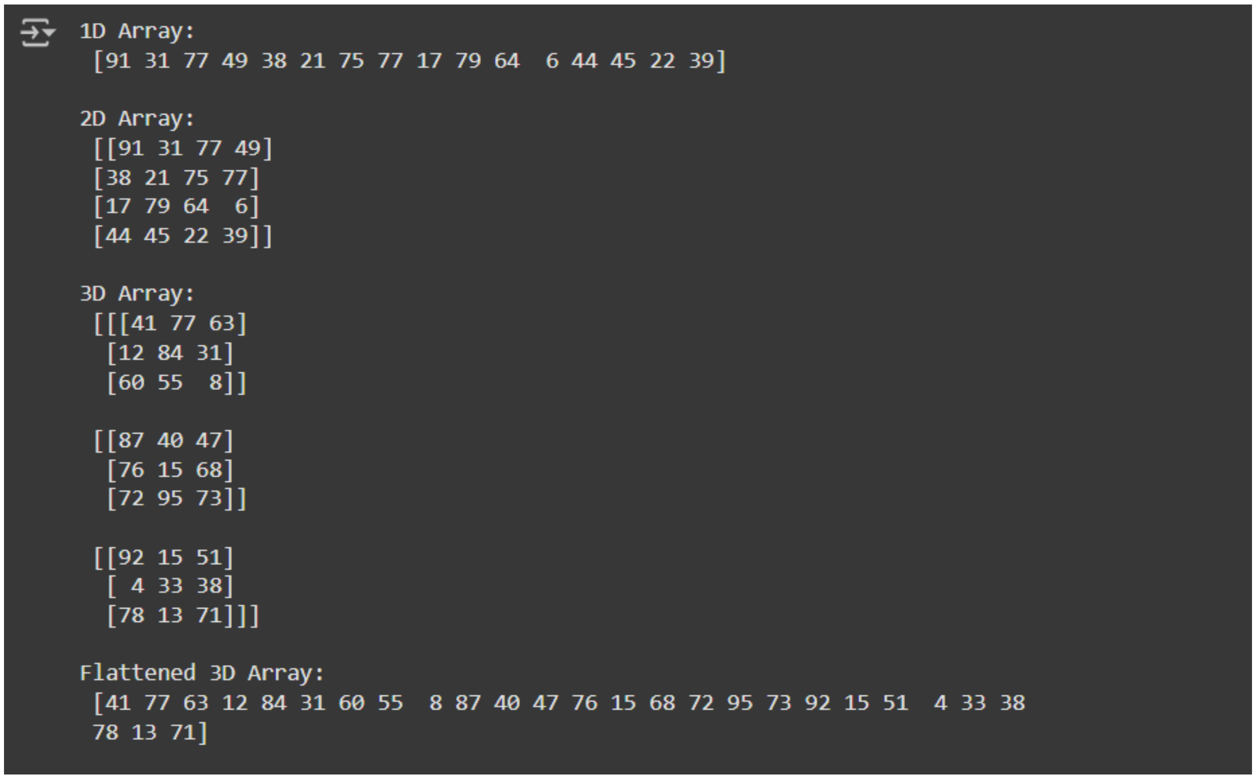
Explanation:

1. **Generate a 6×6 matrix (`matrix`)** with random values between `0` and 1 using `np.random.rand(6, 6)`.
2. **Convert it into a binary matrix (`binarymatrix`)**:
   - If an element is greater than `0.5`, replace it with 1; otherwise, replace it with `0`.
3. **Extract a 3×3 `submatrix` (`submatrix`)** from rows 2 to 4 and columns 2 to 4 (indexing is zero-based).
4. **Calculate the mean value (`mean_value`)** of the extracted submatrix using `np.mean(submatrix)`.
5. **Print the original matrix, binary matrix, submatrix, and its mean value.**

Q7. Array Reshaping: Create a 1D array with 16 elements. Reshape it into a 4x4 matrix. Flatten a 3x3x3 array into a 1D array. Reshape a matrix into a new shape without changing its data.



```
import numpy as np

matrix_1d = np.random.randint(1,100, size=16)

matrix_4x4 = matrix_1d.reshape(4, 4)

print("1D Array:\n", matrix_1d)
print("\n2D Array:\n", matrix_4x4)

matrix_3x3x3 = np.random.randint(1,100, size=(3, 3, 3))

print("\n3D Array:\n", matrix_3x3x3)

flaten_1d=matrix_3x3x3.flatten()

print("\nFlattened 3D Array:\n", flaten_1d)
```



```
1D Array:
 [91 31 77 49 38 21 75 77 17 79 64  6 44 45 22 39]

2D Array:
 [[91 31 77 49]
 [38 21 75 77]
 [17 79 64  6]
 [44 45 22 39]]

3D Array:
 [[[41 77 63]
  [12 84 31]
  [60 55  8]]

 [[87 40 47]
  [76 15 68]
  [72 95 73]]

 [[92 15 51]
  [ 4 33 38]
  [78 13 71]]]

Flattened 3D Array:
 [41 77 63 12 84 31 60 55  8 87 40 47 76 15 68 72 95 73 92 15 51  4 33 38
 78 13 71]
```

Explanation:

1. Generate a 1D array (`matrix_1d`)

   - Create a 1D array of size 16 with random integers between 1 and 99 using `np.random.randint(1, 100, size=16)`.

2. Convert it into a 4×4 matrix (`matrix_4x4`)

   - Reshape the 1D array into a 4×4 matrix using `.reshape(4, 4)`.

3. Generate a 3×3×3 3D array (`matrix_3x3x3`)

   - Create a 3D array with dimensions 3×3×3, filled with random integers between 1 and 99.

4. Flatten the 3D array (`flaten_1d`)

   - Convert the 3D array into a 1D array using `.flatten()`.

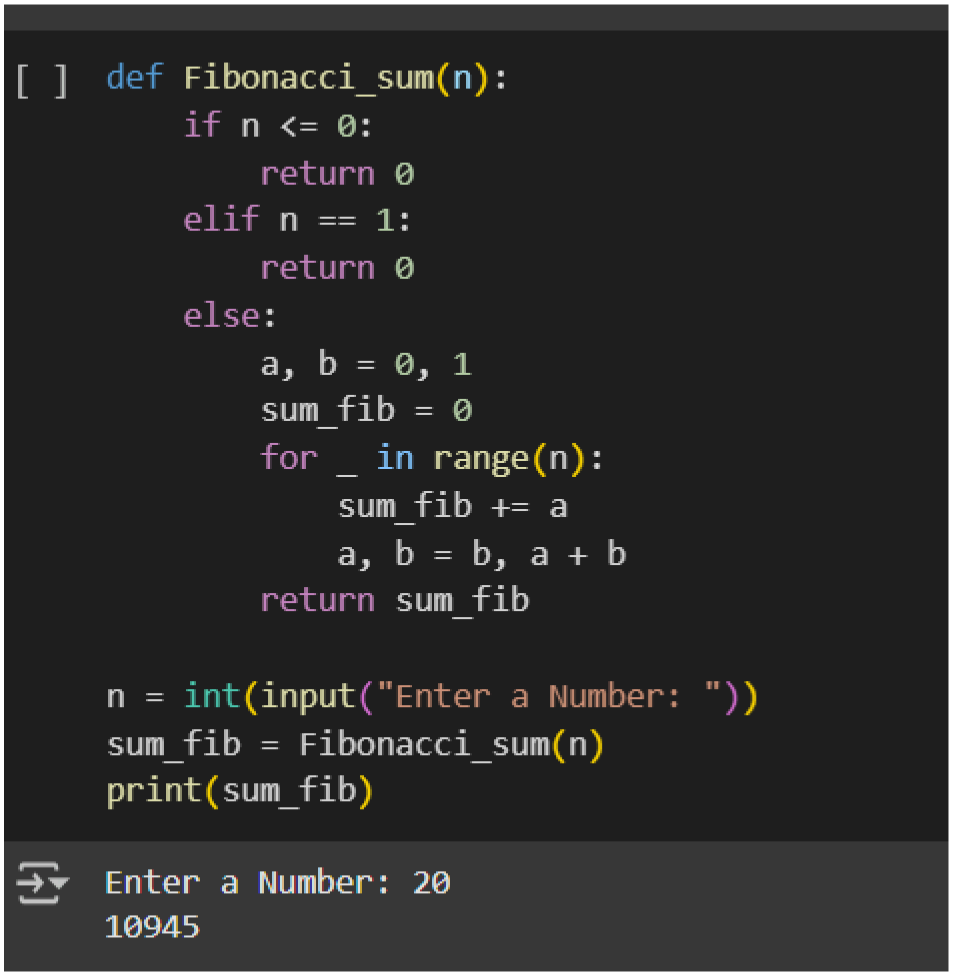
5. Print all arrays
   - Displays the original 1D array, reshaped 4×4 matrix, 3D array, and flattened version of the 3D array.

Q8. Write a recursive function Fibonacci_sum(n) to calculate the sum of first n numbers in Fibonacci series

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89.

Input: 1 Output: 0

Input: 4: Output: 4



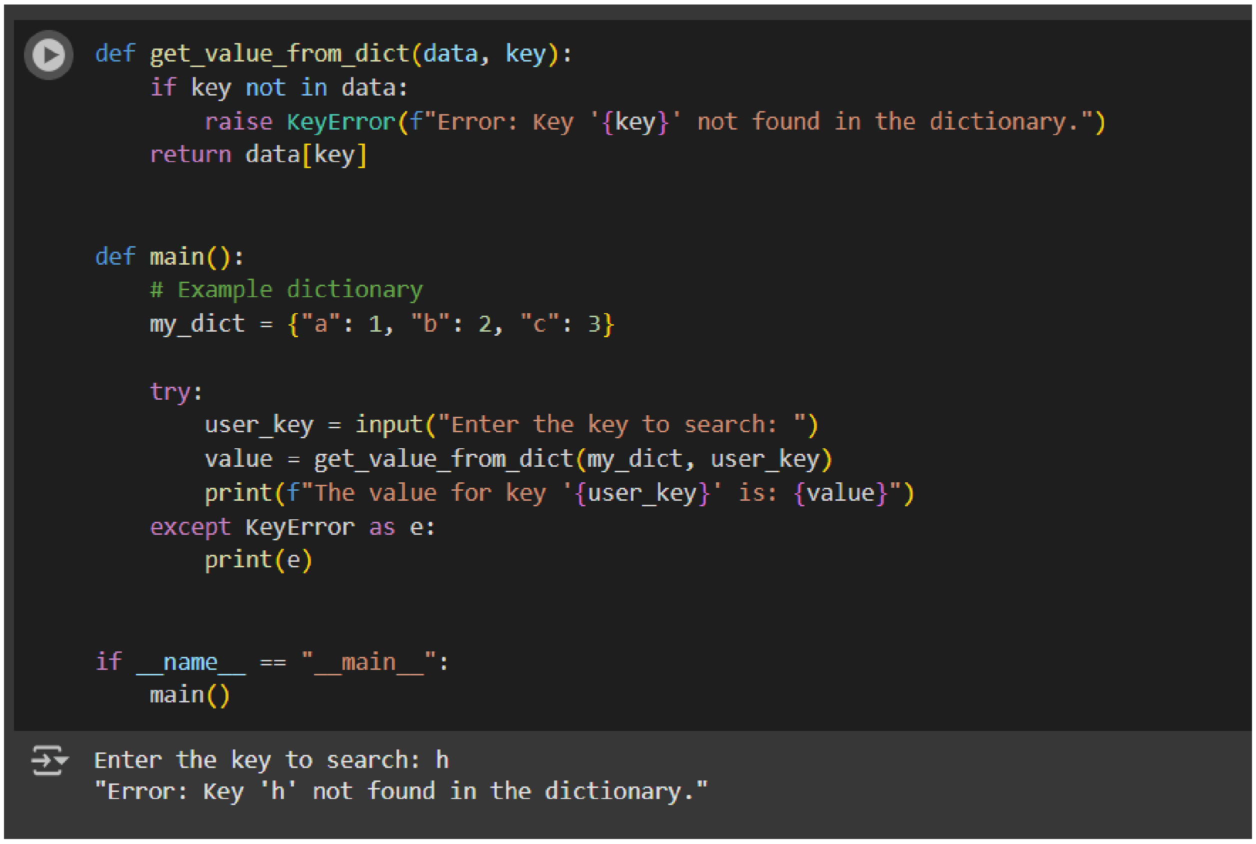
```
[ ]  def Fibonacci_sum(n):
         if n <= 0:
             return 0
         elif n == 1:
             return 0
         else:
             a, b = 0, 1
             sum_fib = 0
             for _ in range(n):
                 sum_fib += a
                 a, b = b, a + b
             return sum_fib

     n = int(input("Enter a Number: "))
     sum_fib = Fibonacci_sum(n)
     print(sum_fib)
```

```
Enter a Number: 20
10945
```

Explanation:

1. Handle base cases:
    - If `n <= 0`, return `0` (invalid input).
    - If `n == 1`, return `0` (since the first Fibonacci number is `0`).
2. Initialize Fibonacci sequence:
    - Start with `a = 0` (first Fibonacci number) and `b = 1` (second Fibonacci number).
    - Use `sum_fib = 0` to keep track of the sum.
3. Loop through n Fibonacci numbers:
    - Add a (current Fibonacci number) to `sum_fib`.
    - Update `a, b` to the next Fibonacci numbers using `a, b = b, a + b`.
4. Return the sum of the first n Fibonacci numbers.

Q9. Define a function get_value_from_dict that takes a dictionary and a key as parameters. If the key is not present in the dictionary, the function should raise a KeyError with a custom error message. Write a main function that calls get_value_from_dict with a dictionary and user-provided key. Handle KeyError and display a user-friendly message if the key is not found.



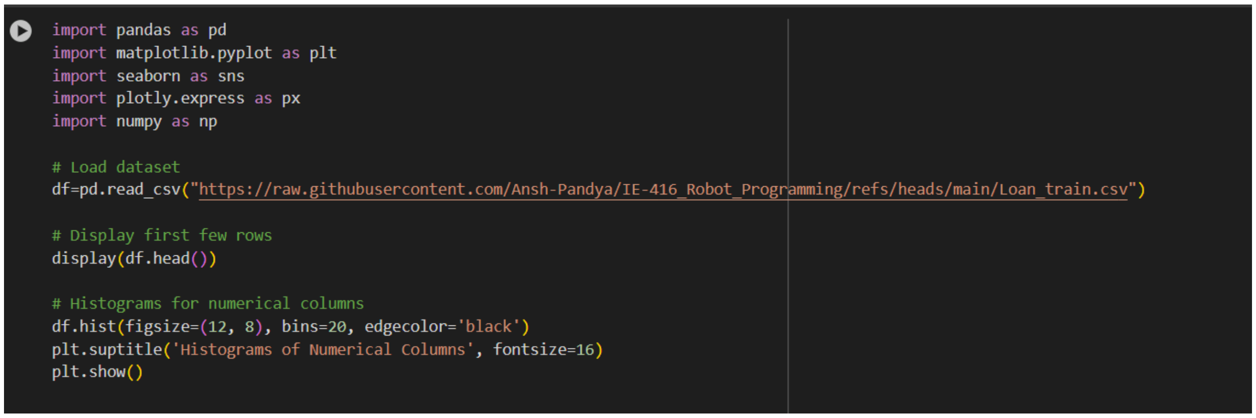
```python
def get_value_from_dict(data, key):
    if key not in data:
        raise KeyError(f"Error: Key '{key}' not found in the dictionary.")
    return data[key]


def main():
    # Example dictionary
    my_dict = {"a": 1, "b": 2, "c": 3}

    try:
        user_key = input("Enter the key to search: ")
        value = get_value_from_dict(my_dict, user_key)
        print(f"The value for key '{user_key}' is: {value}")
    except KeyError as e:
        print(e)


if __name__ == "__main__":
    main()
```

```
Enter the key to search: h
"Error: Key 'h' not found in the dictionary."
```

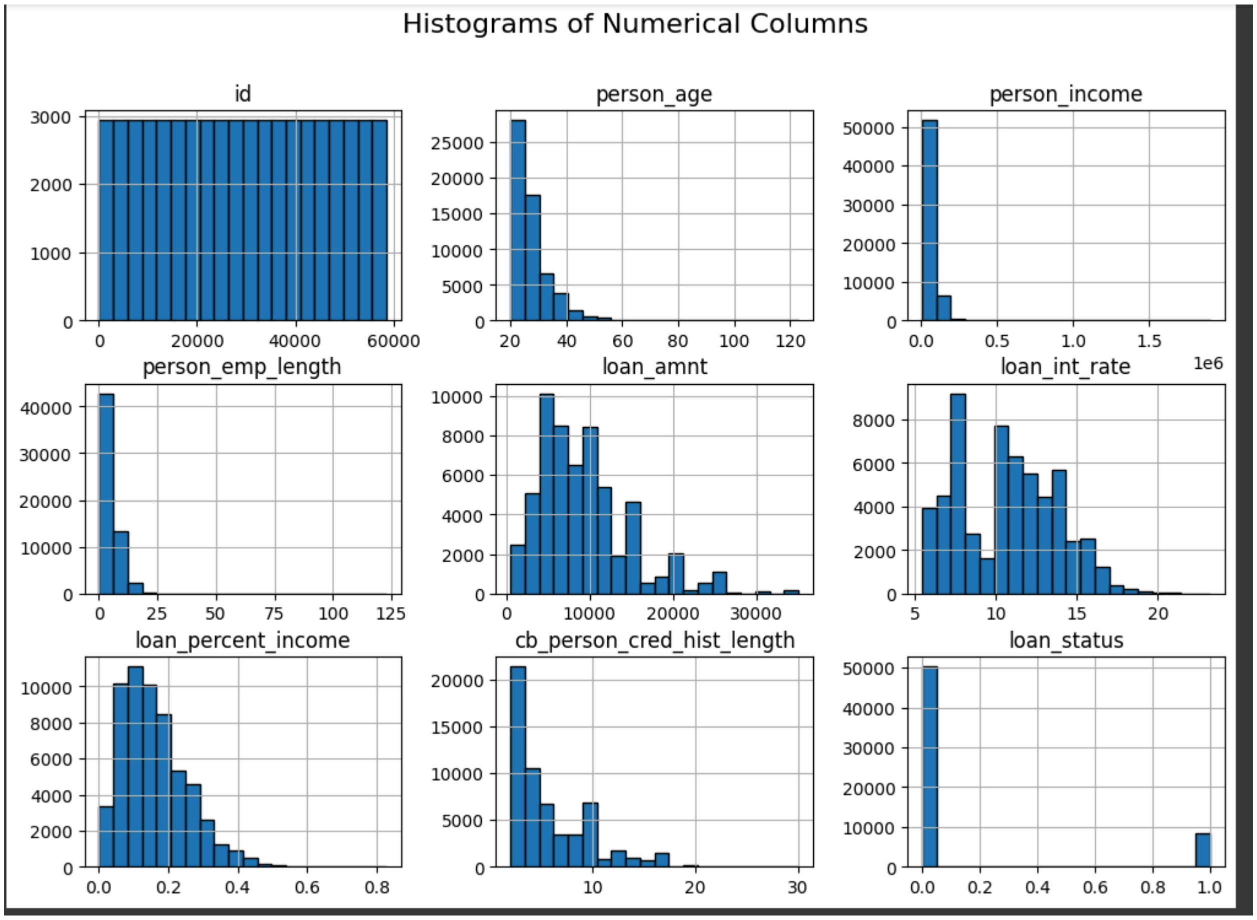
Explanation:

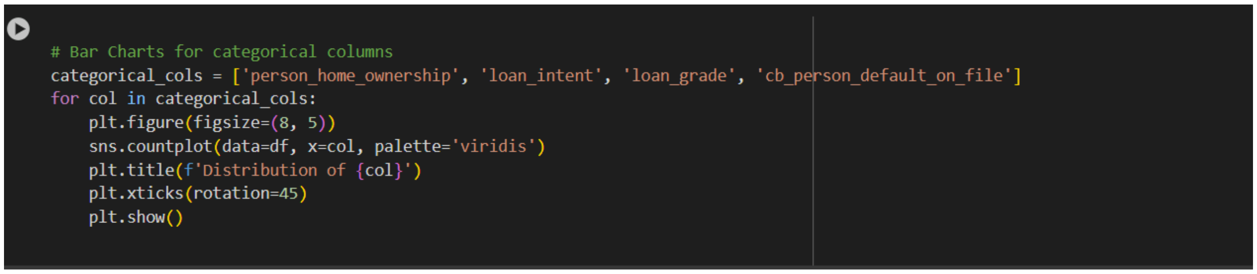
1. Function `get_value_from_dict(data, key)`
   - Checks if the `key` exists in the dictionary.
   - If not found, raises a `KeyError` with a custom error message.
   - If found, returns the corresponding value.
2. Function `main()`
   - Defines a sample dictionary (`my_dict = {"a": 1, "b": 2, "c": 3}`).
   - Takes user input for the key.
   - Calls `get_value_from_dict()` to retrieve the value.
   - Handles `KeyError` using `try-except` and prints an error message if the key is missing.
3. `if __name__ == "__main__": main()`
   - Ensures that `main()` runs only when the script is executed directly.

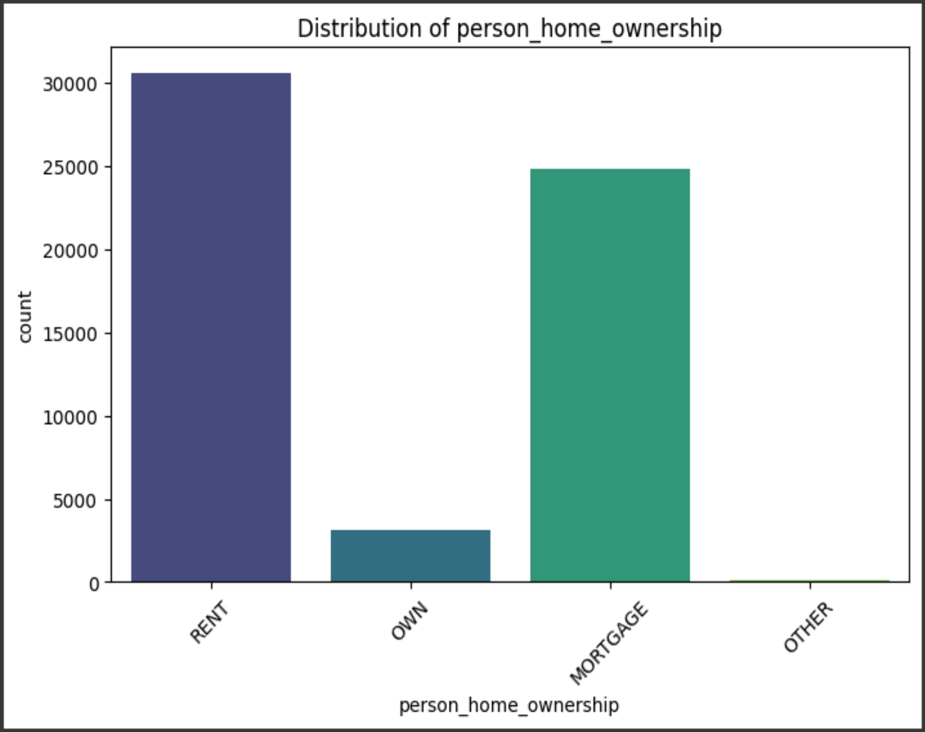
Q10. Using the following dataset, visualize the data with the maximum number of visualization tools available in Python. Create a variety of plots and charts, including but not limited to bar charts, pie charts, line graphs, scatter plots, histograms, and heatmaps. Use libraries such as matplotlib,seaborn, and plotly to explore different ways of presenting the data. Provide clear titles, labels, and legends to enhance the readability of your visualizations.



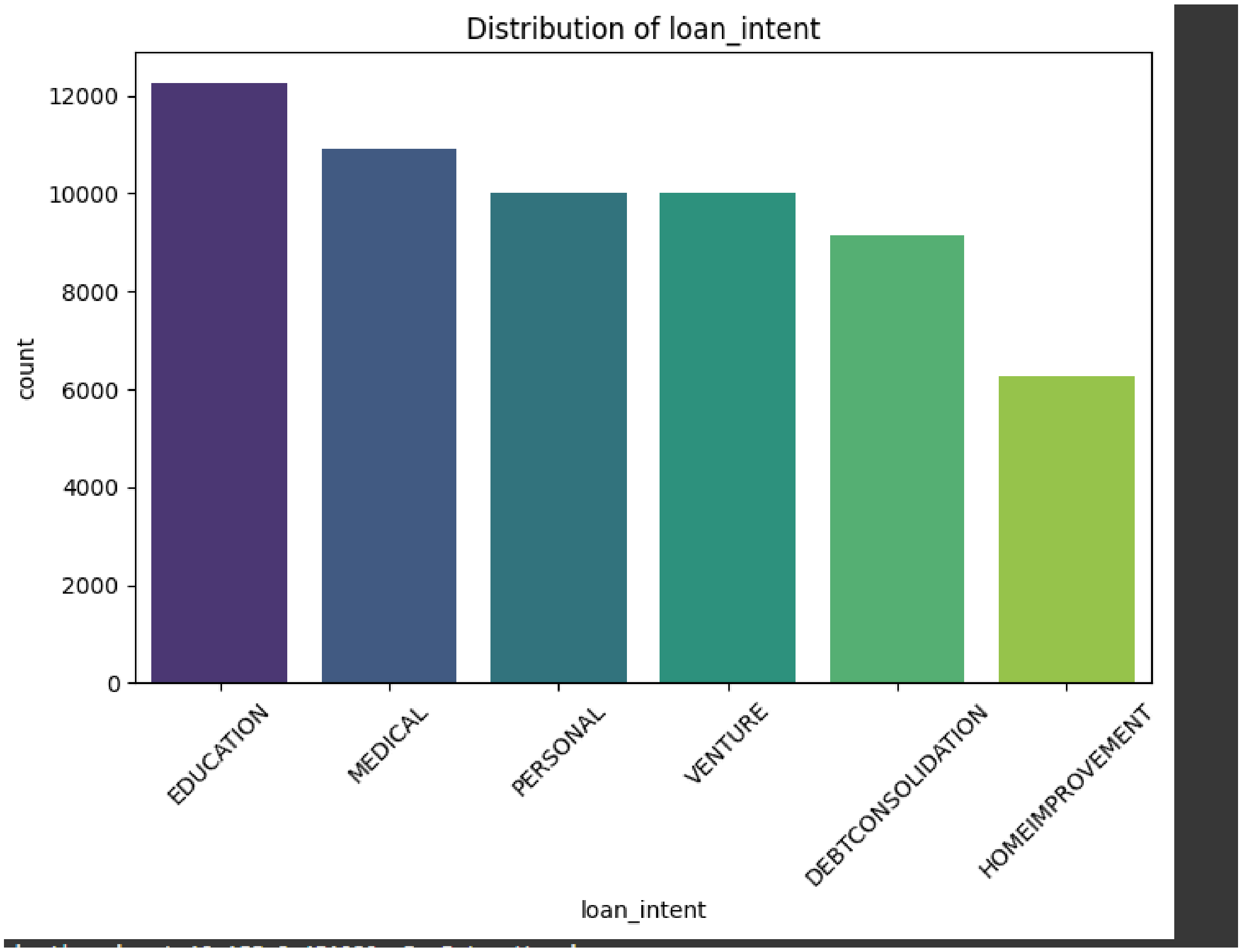
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import numpy as np

# Load dataset
df=pd.read_csv("https://raw.githubusercontent.com/Ansh-Pandya/IE-416_Robot_Programming/refs/heads/main/Loan_train.csv")

# Display first few rows
display(df.head())

# Histograms for numerical columns
df.hist(figsize=(12, 8), bins=20, edgecolor='black')
plt.suptitle('Histograms of Numerical Columns', fontsize=16)
plt.show()
```

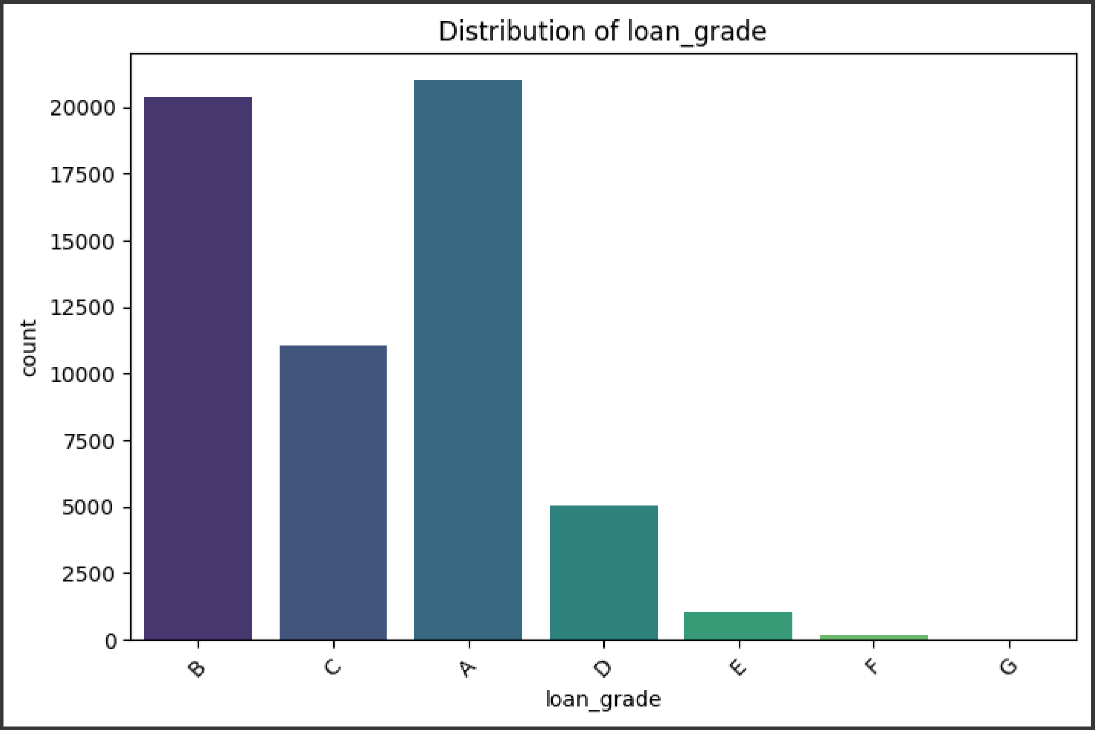



```
# Bar Charts for categorical columns
categorical_cols = ['person_home_ownership', 'loan_intent', 'loan_grade', 'cb_person_default_on_file']
for col in categorical_cols:
    plt.figure(figsize=(8, 5))
    sns.countplot(data=df, x=col, palette='viridis')
    plt.title(f'Distribution of {col}')
    plt.xticks(rotation=45)
    plt.show()
```

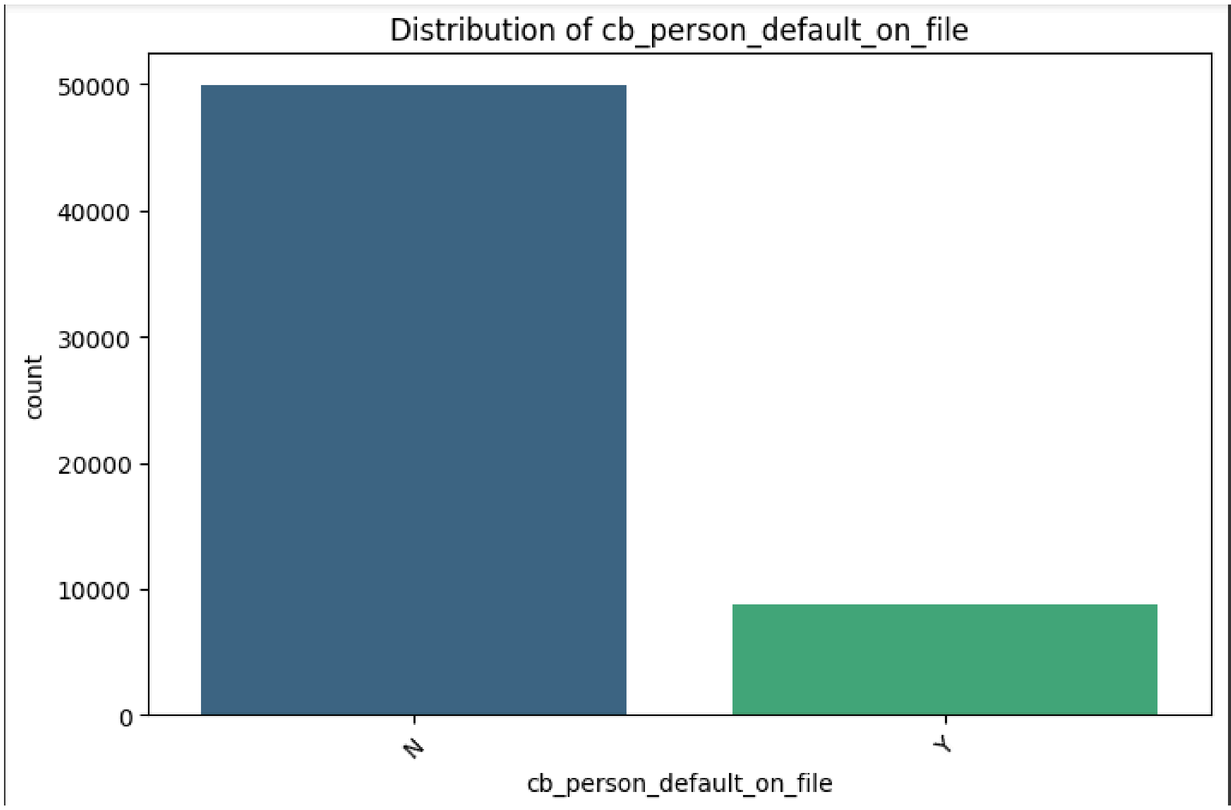


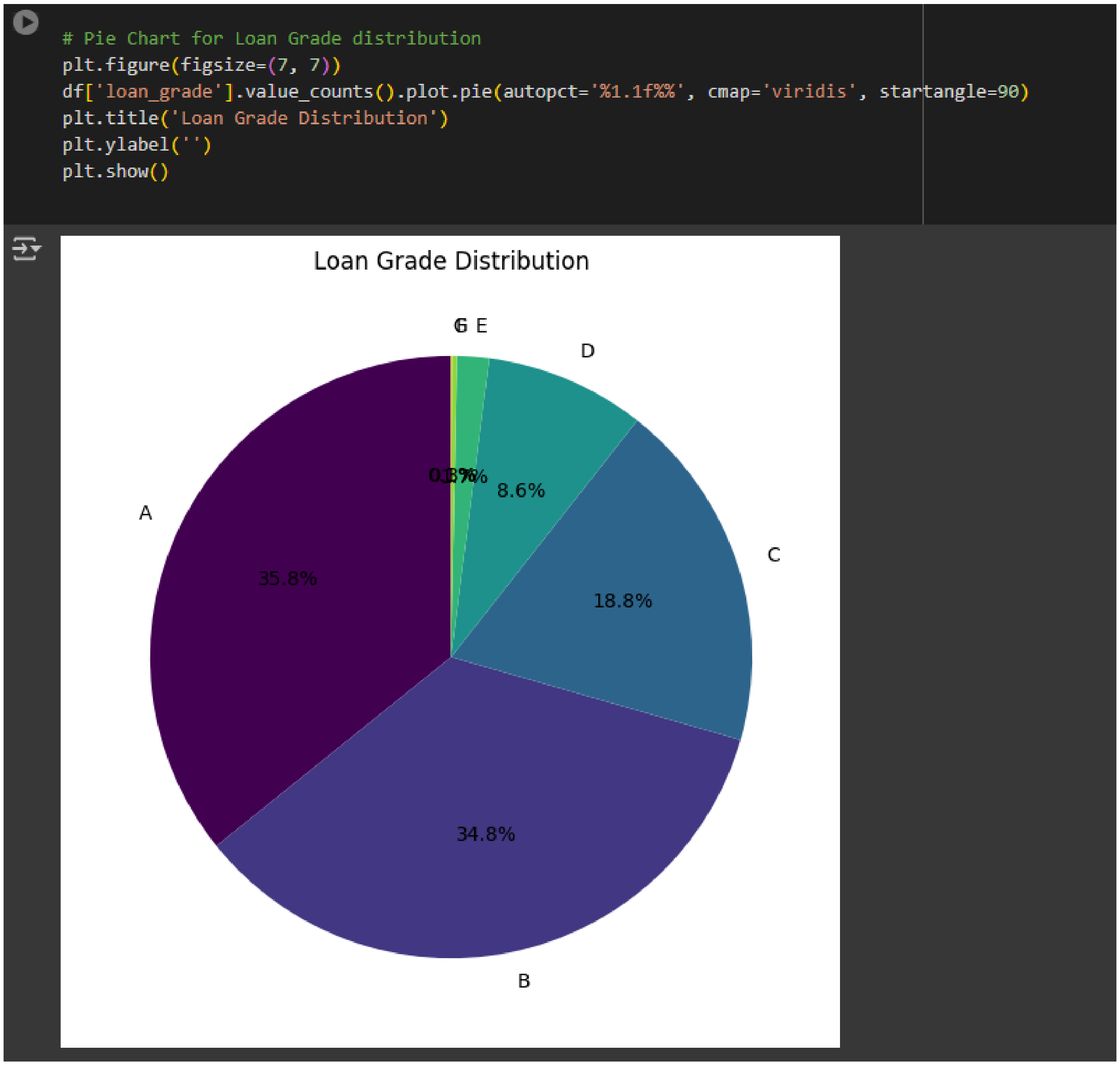
```
<ipython-input-12-155e8a451080>:5: FutureWarning:
```


Distribution of loan_intent
count
0
2000
4000
6000
8000
10000
12000
EDUCATION
MEDICAL
PERSONAL
VENTURE
DEBTCONSOLIDATION
HOMEIMPROVEMENT
loan_intent

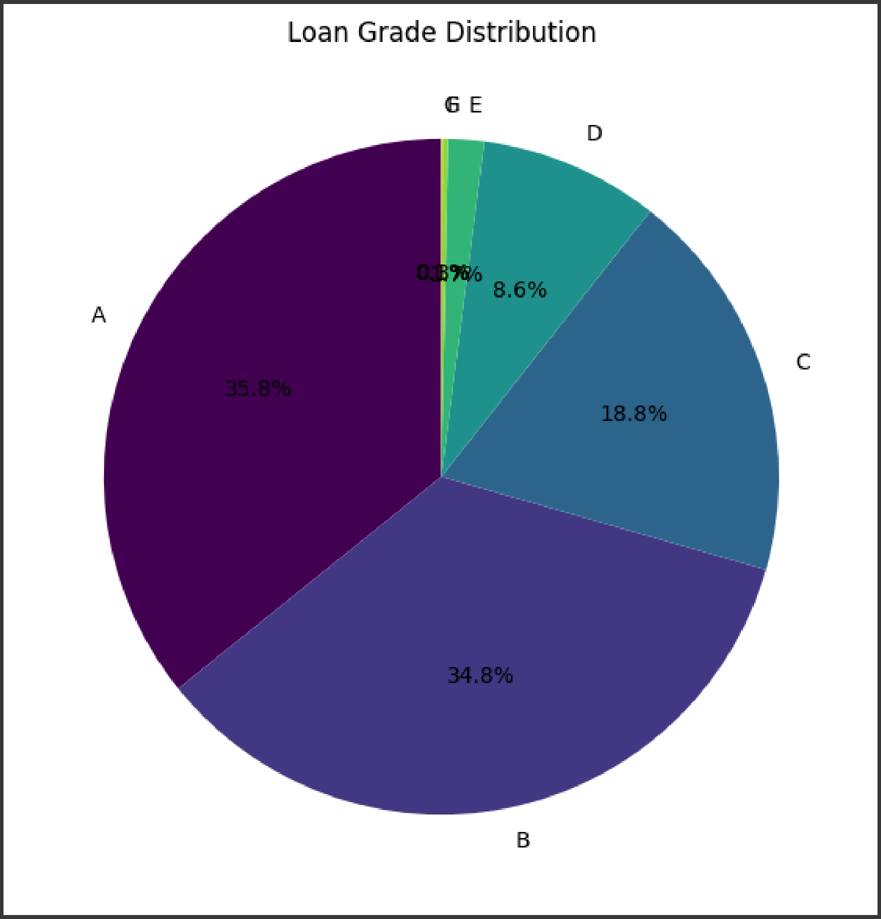


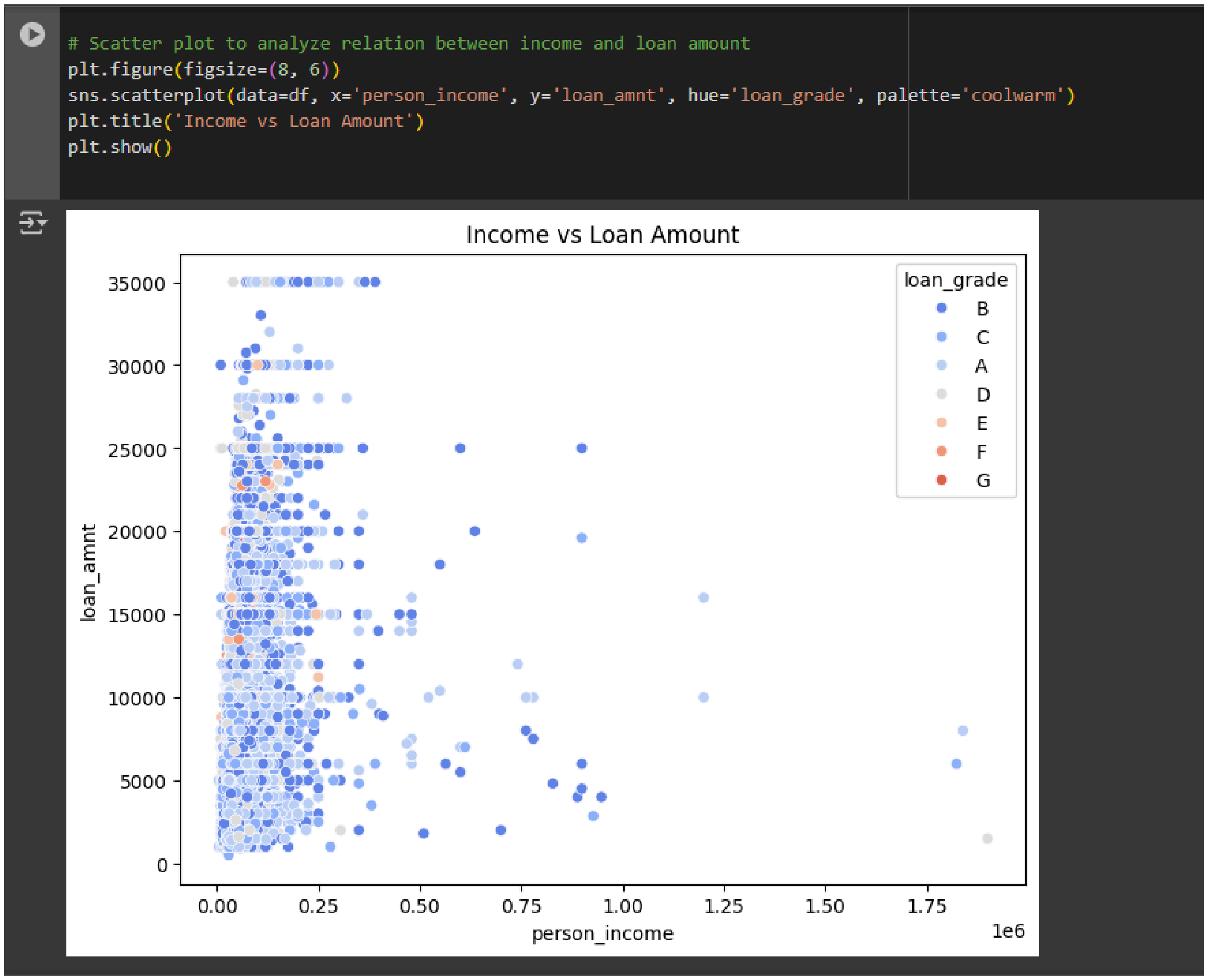
```
<ipython-input-12-155e8a451080>:5: FutureWarning:
```



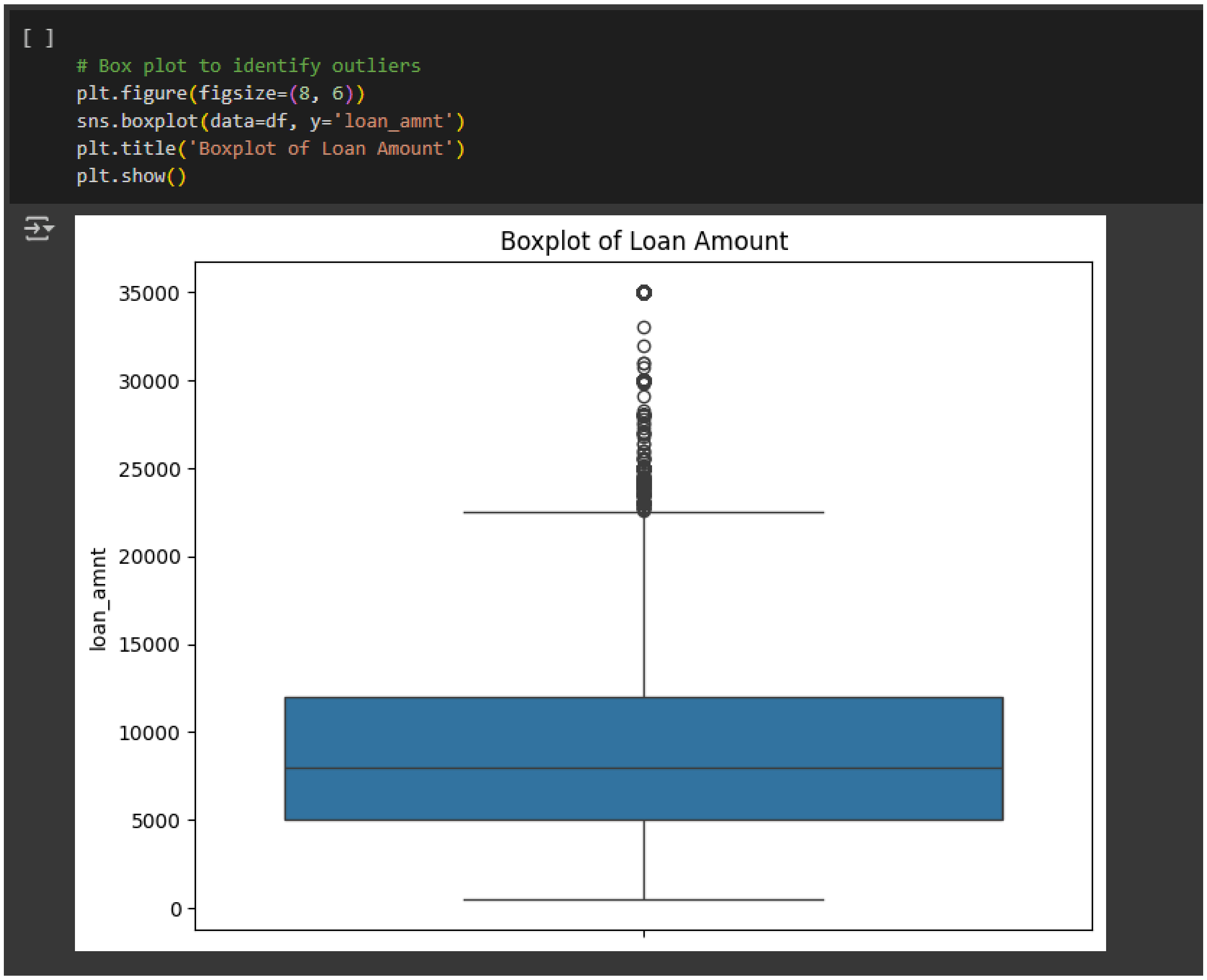


```
# Pie Chart for Loan Grade distribution
plt.figure(figsize=(7, 7))
df['loan_grade'].value_counts().plot.pie(autopct='%1.1f%%', cmap='viridis', startangle=90)
plt.title('Loan Grade Distribution')
plt.ylabel('')
plt.show()
```

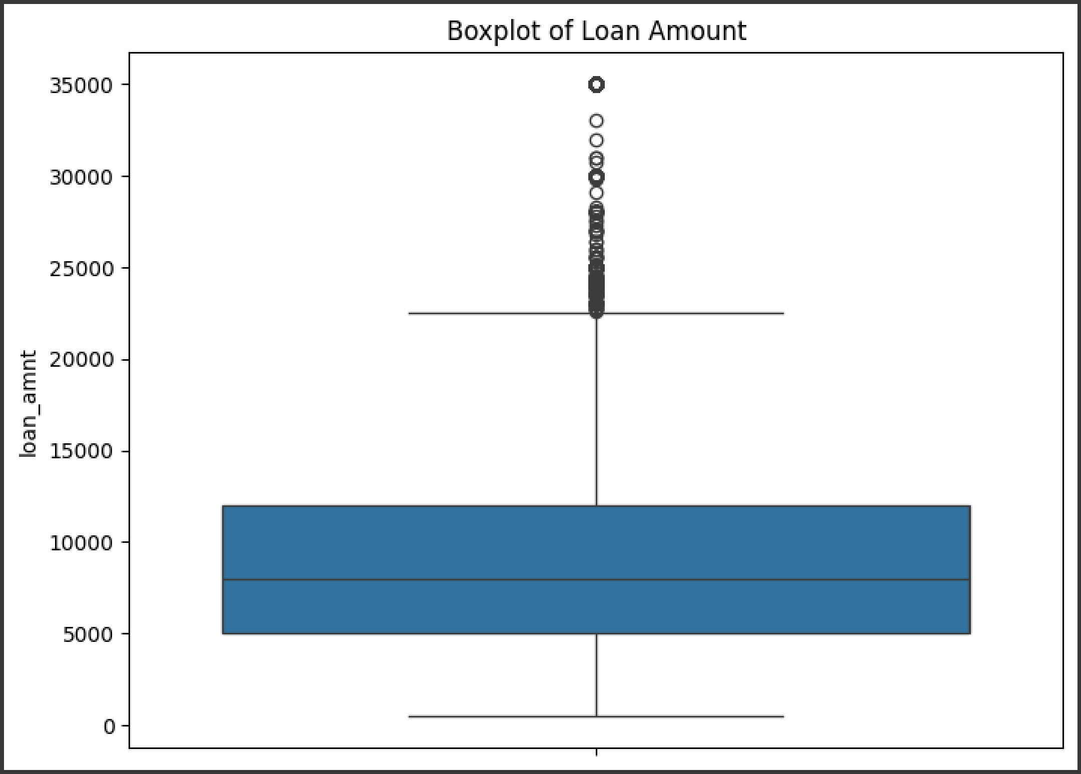




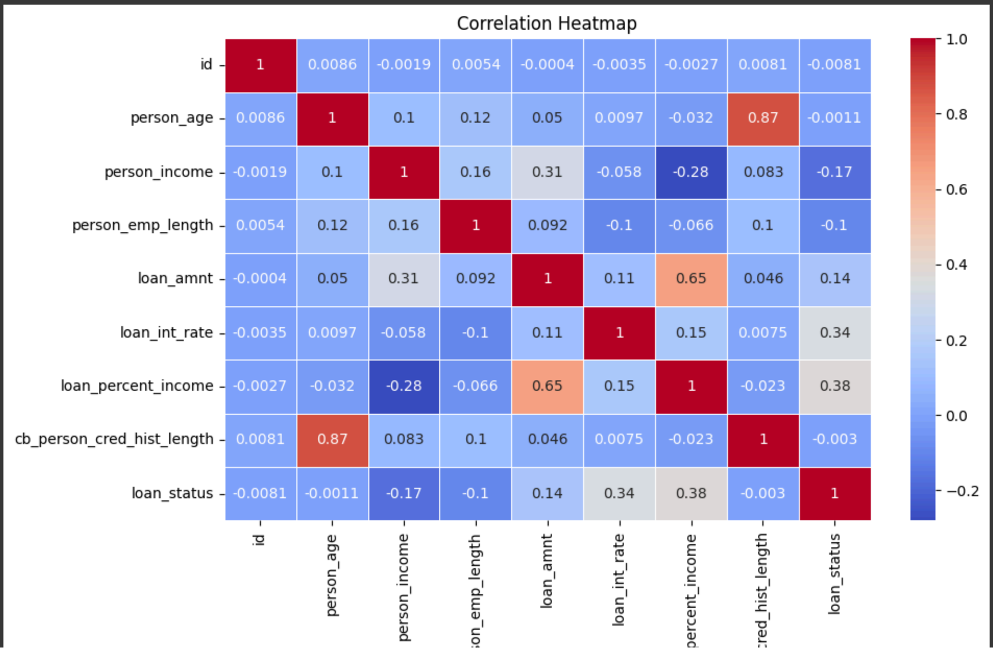
```
# Scatter plot to analyze relation between income and loan amount
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='person_income', y='loan_amnt', hue='loan_grade', palette='coolwarm')
plt.title('Income vs Loan Amount')
plt.show()
```



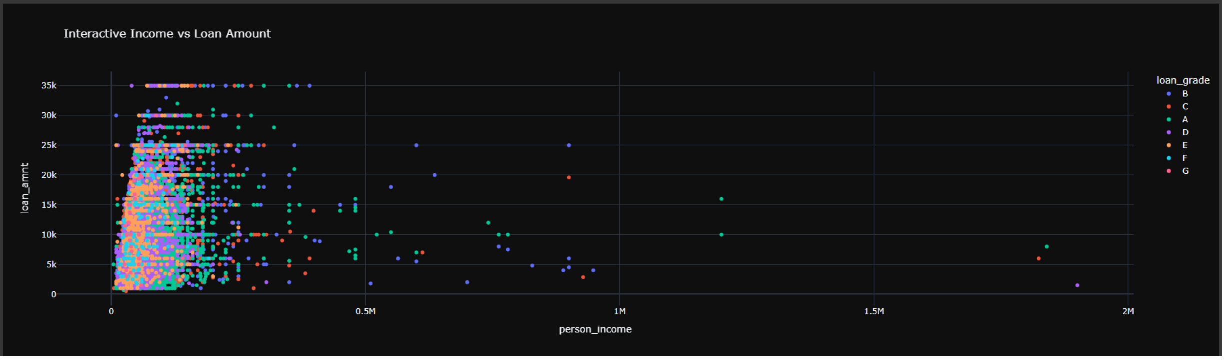
```
[ ]
    # Box plot to identify outliers
    plt.figure(figsize=(8, 6))
    sns.boxplot(data=df, y='loan_amnt')
    plt.title('Boxplot of Loan Amount')
    plt.show()
```



```python
# Correlation heatmap
plt.figure(figsize=(10, 6))
numeric_df = df.select_dtypes(include=['number'])  # Select only numerical columns
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



```python
# Interactive scatter plot with Plotly
fig = px.scatter(df, x='person_income', y='loan_amnt', color='loan_grade', title='Interactive Income vs Loan Amount')
fig.show()
```



Explanation:

1. Here the data set is imported from my github repository.
2. By importing different set of libraries various observations and graphs are generated for visualization of the data.