IE 416 - Robot Programming

LAB 1

RoboPy Hunters

Meet Joshi - 202201065
Smit Shah - 202201192

Useful Links :

Github Link

Colab Link

NumPy Link

Visualisation Link

IE 416 - Robot Programming

LAB 1

Group name : RoboPy Hunters

Group member-

Meet Joshi - 202201065

Smit shah - 202201192

GitHub Link

Colab Link

Numpy link

Visulisation Link

+ Code + Text

Q1. Write a function that gives number of days of given year.

Input : 1990 Input : 2044

Output : 365 Output : 366

```
def days_in_year(year):
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
        return 366
    else:
        return 365

year = int(input("Enter a year: "))
print(f"Days in {year}: {days_in_year(year)}")
```

```
Enter a year: 2044
Days in 2044: 366
```

A function was created to determine whether a given year is a leap year. Based on this, it returns 366 for leap years and 365 for common years.

Steps:

- Take a year as input from the user.
- Use conditional checks to determine if the year is divisible by 4 and 100, with an exception for years divisible by 400.
- Return 365 for common years and 366 for leap years.

Example: For input years 1990 and 2044, the function returns 365 and 366, respectively.

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}

```
def char_frequency(string):
    freq = {}
    for char in string:
        freq[char] = freq.get(char, 0) + 1
    return freq


string = input("Enter string: ")
print("Character's frequency:", char_frequency(string))
```

```
Enter string: Meet Smit
Character's frequency: {'M': 1, 'e': 2, 't': 2, ' ': 1, 'S': 1, 'm': 1, 'i': 1}
```

This program calculates the frequency of each character in a given string and stores the counts in a dictionary.

Steps:

1) Traverse the string, processing each character one by one.

2) Update the dictionary to record the number of times each character appears.

Example Output:

For the input "adcbbdaacd", the output is {'a': 3, 'd': 3, 'c': 2, 'b': 2}.

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]

```
def remove_duplicates(lst):
    seen = set()
    result = []
    for item in lst:
        if item not in seen:
            seen.add(item)
            result.append(item)
    return result


lst = list(map(int, input("Enter numbers separated by spaces: ").split()))
print("List without duplicates is:", remove_duplicates(lst))
```

```
Enter numbers separated by spaces: 1 2 3 4 5 4 3 2 1 6
List without duplicates is: [1, 2, 3, 4, 5, 6]
```

This function removes duplicate elements from a list while maintaining the order of their first appearance.

Steps:

1. Use a set to track elements that have already been encountered.
2. Add elements to a new list only if they are not already in the set.

Example Output:

For the input [1, 2, 3, 4, 2, 3, 5, 6, 1, 4], the output is [1, 2, 3, 4, 5, 6].

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):
    helper_stack = []
    while stack:
        temp = stack.pop()
        while helper_stack and helper_stack[-1] > temp:
            stack.append(helper_stack.pop())
        helper_stack.append(temp)
    return helper_stack

stack = list(map(int, input("Enter a stack element separated by spaces: ").split()))
sorted_stack = sort_stack(stack)
print("Sorted stack given by:", sorted_stack)
```

```
Enter a stack element separated by spaces: 9 5 1 3
Sorted stack given by: [1, 3, 5, 9]
```

A stack-sorting program was implemented using an additional stack.

Steps:

1. Pop elements from the input stack and insert them into a temporary stack while maintaining sorted order.
2. Transfer the sorted elements back to the original stack.

Example Output:
For the input [9, 5, 1, 3], the output is [1, 3, 5, 9].

Q5. Make a module “pascal.py” with function “pascalTriangle(numOfRows)” and import into “main.py”.

```
def pascal_triangle(num_of_rows):
    triangle = []
    for i in range(num_of_rows):
        row = [1] * (i + 1)
        for j in range(1, i):
            row[j] = triangle[i - 1][j - 1] + triangle[i - 1][j]
        triangle.append(row)
    return triangle


number_rows = int(input("Enter number of rows of Pascal's Triangle: "))

triangle = pascal_triangle(number_rows)

print("\nPascal's Triangle:")
max_width = len(" ".join(map(str, triangle[-1])))
for row in triangle:
    row_str = " ".join(map(str, row)).center(max_width)
    print(row_str)
```

```
Enter number of rows of Pascal's Triangle: 17

 Pascal's Triangle:
                                     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
                           1 7 21 35 35 21 7 1
                          1 8 28 56 70 56 28 8 1
                       1 9 36 84 126 126 84 36 9 1
                     1 10 45 120 210 252 210 120 45 10 1
                   1 11 55 165 330 462 462 330 165 55 11 1
                 1 12 66 220 495 792 924 792 495 220 66 12 1
              1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
           1 14 91 364 1001 2002 3003 3432 3003 2002 1001 364 91 14 1
        1 15 105 455 1365 3003 5005 6435 6435 5005 3003 1365 455 105 15 1
     1 16 120 560 1820 4368 8008 11440 12870 11440 8008 4368 1820 560 120 16 1
```

A module named **pascal.py** was created, featuring a function to generate Pascal's triangle up to a specified number of rows.

Steps:

1. Construct the triangle row by row using loops or recursion.
2. Import and use this module in the main program.

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.random((6, 6))
matrix[matrix > 0.5] = 1
matrix[matrix <= 0.5] = 0

sub_matrix = matrix[2:5, 2:5]
mean_value = np.mean(sub_matrix)

print("Matrix:\n", matrix)
print("Sub-matrix:\n", sub_matrix)
print("Mean of sub-matrix:", mean_value)
```

```
Matrix:
 [[0. 1. 1. 0. 1. 1.]
 [1. 0. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 0.]
 [1. 0. 0. 0. 1. 0.]
 [1. 0. 0. 1. 0. 1.]
 [1. 1. 0. 1. 1. 0.]]
Sub-matrix:
 [[1. 1. 1.]
 [0. 0. 1.]
 [0. 1. 0.]]
Mean of sub-matrix: 0.5555555555555556
```

A 6x6 matrix was generated with random values between 0 and 1. Two operations were performed:

1. Thresholding: Values greater than 0.5 were replaced with 1, while the rest were set to 0.
2. Submatrix Extraction: A 3x3 submatrix starting at index (2,2) was extracted, and its mean was calculated.

Output:
The modified 6x6 matrix and the mean of the submatrix are displayed.

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


array = np.arange(1, 17)
reshaped_array = array.reshape(4, 4)

array_3d = np.random.randint(1, 10, (3, 3, 3))
flattened_array = array_3d.flatten()


reshaped_array_2 = array.reshape(8, 2)

print("Original array:", array)
print("Reshaped 4x4 matrix:\n", reshaped_array)
print("3d array:", array_3d)
print("Flattened array:", flattened_array)
print("Reshaped 8x2 matrix:\n", reshaped_array_2)
```

```
Original array: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
Reshaped 4x4 matrix:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
3d array: [[[2 5 4]
  [7 1 6]
  [9 8 6]]

 [[1 7 8]
  [9 8 1]
  [4 7 2]]

 [[2 9 7]
  [4 7 1]
  [6 8 9]]]
Flattened array: [2 5 4 7 1 6 9 8 6 1 7 8 9 8 1 4 7 2 2 9 7 4 7 1 6 8 9]
Reshaped 8x2 matrix:
 [[ 1  2]
 [ 3  4]
 [ 5  6]
 [ 7  8]
 [ 9 10]
 [11 12]
 [13 14]
 [15 16]]
```

Three array manipulations were performed:

1. Reshaping: A 1D array with 16 elements was converted into a 4×4 matrix.
2. Flattening: A 3×3×3 array was transformed into a 1D array.
3. Reshaping: A matrix was restructured into a new shape while preserving the data order.

Output:
The results of each operation are displayed.

Q8. Write a recursive function Fibonacci_sum(n) to calaculate 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
    elif n == 2:
        return 1
    else:
        return fibonacci_sum(n - 1) + fibonacci_sum(n - 2) + 1


n = int(input("Enter n: "))
print("Fibonacci sum:", fibonacci_sum(n))
```

```
Enter n: 4
Fibonacci sum: 4
```

A recursive function was created to calculate the sum of the first n Fibonacci numbers.

Steps:

1. Compute the Fibonacci numbers recursively.
2. Sum the first n Fibonacci numbers.

Output:
For input n = 4, the result is a sum of 4.

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.

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


def main():
    sample_dict = {'m': 1, 'n': 2, 'o': 3, 'p': 4, 'q':17}
    key = input("Enter a key: ")
    try:
        value = get_value_from_dict(sample_dict, key)
        print(f"The value for '{key}' is: {value}")
    except KeyError as e:
        print(e)

main()
```

```
Enter a key: q
The value for 'q' is: 17
```

A function was implemented to fetch a value for a given key from a dictionary. If the key is absent, a KeyError is raised with a custom error message.

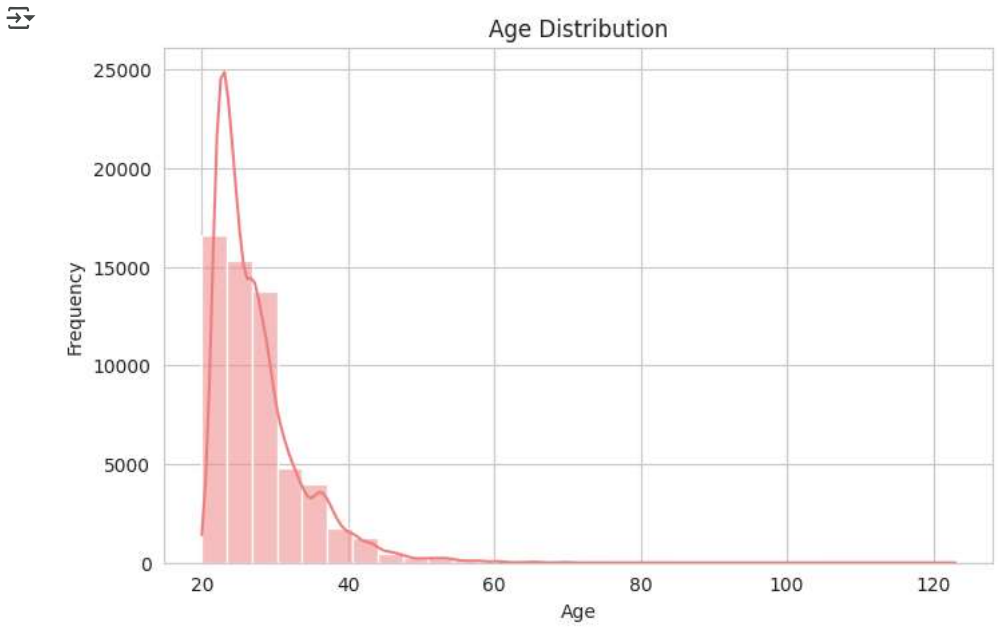
Steps:

1. Try to access the value associated with the key in the dictionary.
2. If the key is not found, catch the KeyError and display a user-friendly error message.

Example Output:
The function returns the value for an existing key or a custom error message if the key is missing.

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.

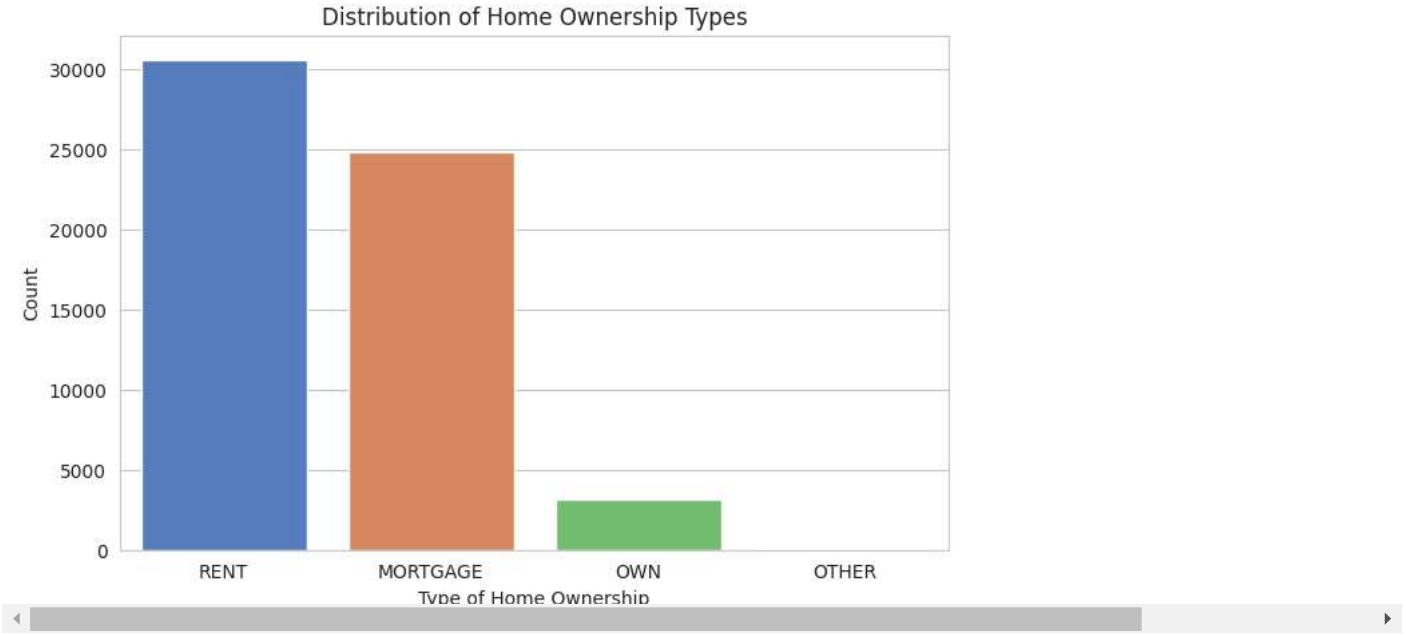
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# URL for the raw CSV file on GitHub
csv_file_url = "https://raw.githubusercontent.com/spidermanMJ17/IE416-RoboProgramming-Lab/refs/heads/main/LAB1/Loan_train.csv"

# Load the dataset into a DataFrame
loan_dataset = pd.read_csv(csv_file_url)

# Set the style for seaborn plots
sns.set_style("whitegrid")

# Plot the distribution of ages
plt.figure(figsize=(8, 5))
sns.histplot(loan_dataset['person_age'], kde=True, bins=30, color='lightcoral')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



Double-click (or enter) to edit

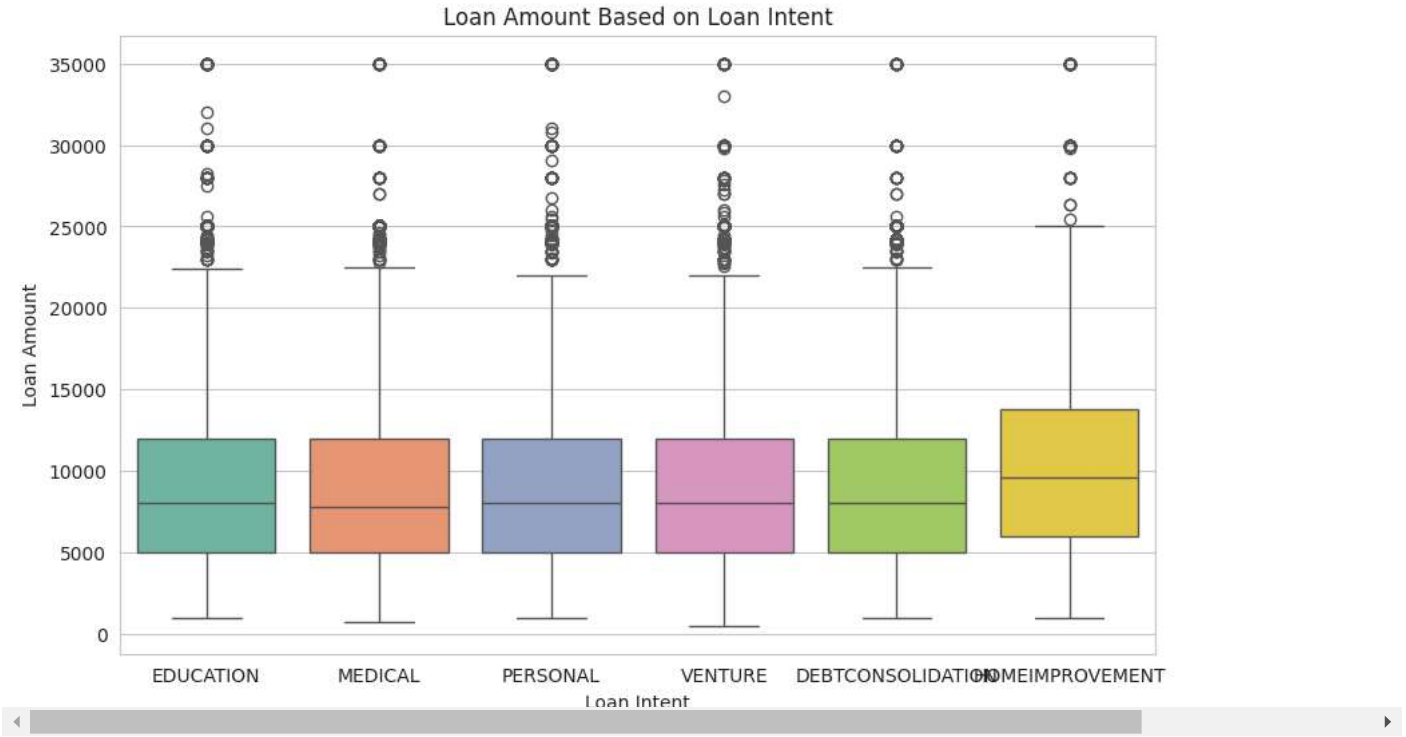
```
# Plot the distribution of home ownership types
plt.figure(figsize=(8, 5))
sns.countplot(x='person_home_ownership', data=loan_dataset, palette='muted',
              order=loan_dataset['person_home_ownership'].value_counts().index)
plt.title('Distribution of Home Ownership Types')
plt.xlabel('Type of Home Ownership')
plt.ylabel('Count')
plt.show()
```

```
<ipython-input-18-bbdaa97015df>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legenc
```

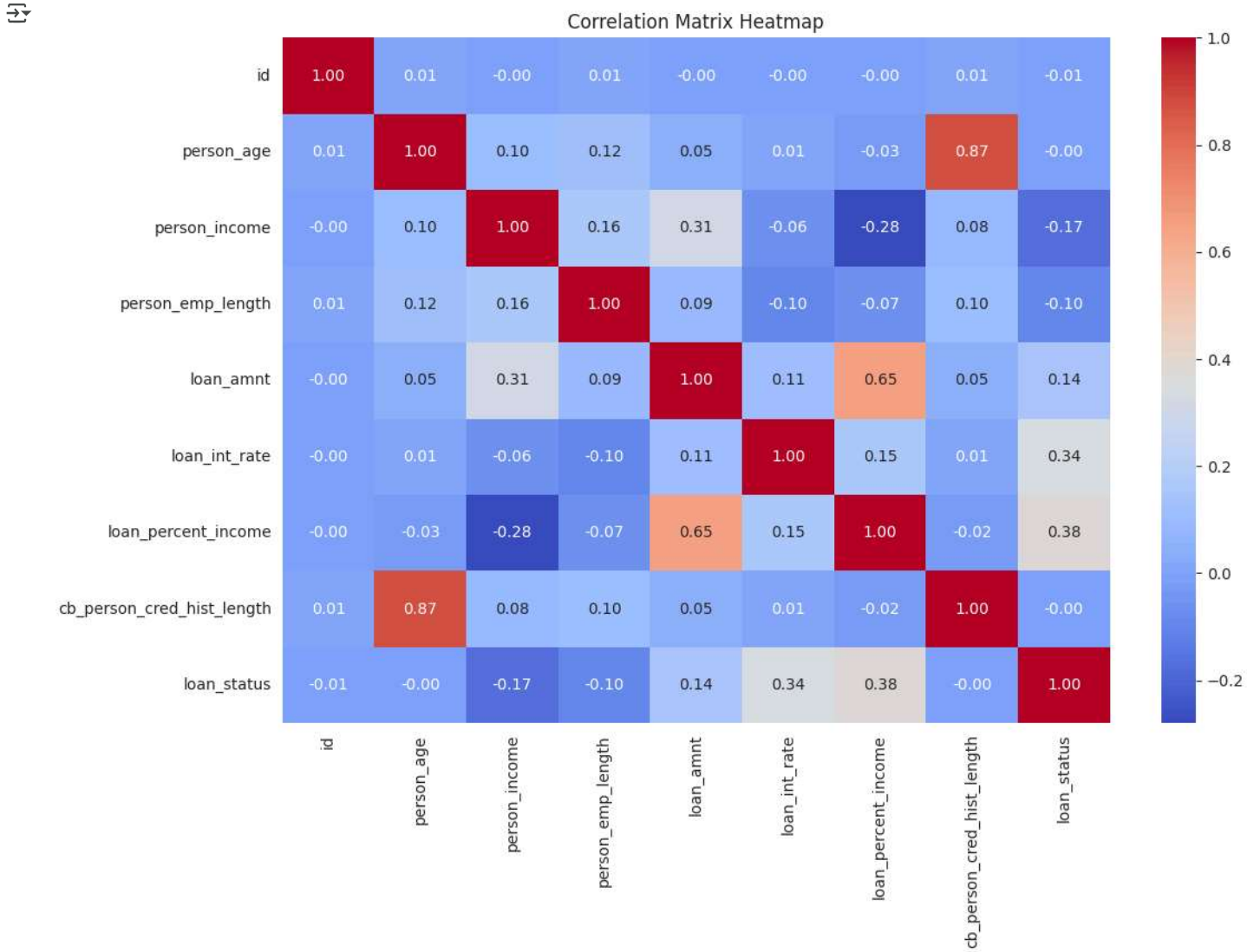



```
# Create a boxplot showing loan amount by loan intent
plt.figure(figsize=(10, 6))
sns.boxplot(x='loan_intent', y='loan_amnt', data=loan_dataset, palette='Set2')
plt.title('Loan Amount Based on Loan Intent')
plt.xlabel('Loan Intent')
plt.ylabel('Loan Amount')
plt.show()
```

```
<ipython-input-19-f25709083f44>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legenc
```



```
# Generate a heatmap for correlations between numerical features
plt.figure(figsize=(12, 8))
numeric_columns = loan_dataset.select_dtypes(include=['float64', 'int64'])
corr_matrix = numeric_columns.corr()
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm', cbar=True)
plt.title('Correlation Matrix Heatmap')
plt.show()
```




```
# Scatter plot for Income vs. Loan Amount with loan intent as the color
scatter_fig = px.scatter(loan_dataset, x='person_income', y='loan_amnt',
                         color='loan_intent',
                         title='Income vs Loan Amount by Loan Intent',
                         labels={'person_income': 'Income', 'loan_amnt': 'Loan Amount'})
scatter_fig.show()
```


```
# Pie chart showing the distribution of loan grades
loan_grade_counts = loan_dataset['loan_grade'].value_counts()
pie_fig = px.pie(values=loan_grade_counts, names=loan_grade_counts.index,
                 title='Distribution of Loan Grades',
                 color_discrete_sequence=px.colors.sequential.Plasma)
pie_fig.show()
```