IE416: Robot Programming Lab-01

Group Syntax

Yash Tarpara - 202201422

Kaushik Prajapati - 202201472

Links:

1. **GitHub Repository:** Access the course lab files – [Click Here].
2. **Notebook for Lab Question :** Direct link to the `.ipynb` file – [Click Here].
3. **Learning NumPy:** Direct link to the `.ipynb` file – [Click Here].
4. **Exploring Data Visualization with Python Libraries:** Direct link to the `.ipynb` file – [Click Here].

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

Code:

```
def findNumberOfDays(year):
    if year % 100 == 0:
        if year % 400 == 0:
            return 366
        else:
            return 365
    elif year % 4 == 0:
        return 366
    return 365

inputs = [1990, 2044, 1900]
for year in inputs:
    print(f"Number of days in year {year}: {findNumberOfDays(year)}")
```

Code Explanation:

The function `findNumberOfDays(year)` determines the number of days in a given year, checking whether the year is a leap year or not.

1. **Input Parameter:**
   a. `year`: An integer representing the year.
2. **Logic:**
   a. A year is a **leap year** if:
      i. It is divisible by 4, but **not divisible by 100**, or
      ii. It is divisible by 400.
   b. If the year satisfies the leap year conditions, it returns `366` (indicating the year has 366 days). Otherwise, it returns `365`.
3. **Code Flow:**
   a. If the year is divisible by 100, check if it is divisible by 400. If true, it is a leap year (`366 days`); otherwise, it is not (`365 days`).
   b. If the year is not divisible by 100, check if it is divisible by 4. If true, it is a leap year (`366 days`); otherwise, it is not (`365 days`).
4. **Sample Input:**
   a. `1990`: Regular year with 365 days.
   b. `2044`: Leap year with 366 days.
   c. `1900`: Century year not divisible by 400, so 365 days.
5. **Output:**
   a. The program iterates through the list of years `[1990, 2044, 1900]` and prints the number of days for each year.

Output of the Program:

```
Number of days in year 1990: 365
Number of days in year 2044: 366
Number of days in year 1900: 365
```

Q2. Count the frequency of each character in a string and store it in a dictionary.

Code:

```
def findFrequencies(input):

    mp = defaultdict(int)
    for char in input:
        mp[char] += 1
    return dict(mp)

input = "adcbbdaacd"
print(findFrequencies(input))
```

Code Explanation:

1. **Function Name:**
    a. `findFrequencies(input):` This function calculates the frequency of each character in a string.
2. **Logic:**
    a. A `defaultdict` of type `int` is used to store character frequencies, where the default value for each key is initialized to 0.
    b. The function iterates through each character in the input string, and for each character, it increments its count in the dictionary.
3. **Input:**
    a. input = "adcbbdaacd"

4. **Output:**
    a. A dictionary where the keys are characters and the values are their respective frequencies: {'a': 3, 'd': 3, 'c': 2, 'b': 2}

Output of the Program:

```
{'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.

Code:

```python
def remove_duplicates(nums):
    res = []
    for num in nums:
        if num not in res:
            res.append(num)
    return res

nums = [1, 2, 6, 2, 6, 3, 4, 4, 5]
print(remove_duplicates(nums))
```

Code Explanation:

1. **Function Name:**
   a. `remove_duplicates(nums)`: This function removes duplicate elements from a list while preserving the order of their first occurrence.
2. **Logic:**
   a. A new list, `res`, is created to store unique elements.
   b. The program iterates through the input list, `nums`, and adds each element to `res` only if it is not already present in `res`.
3. **Input:**
   a. `nums = [1, 2, 6, 2, 6, 3, 4, 4, 5]`
4. **Output**
   a. `Output = [1, 2, 6, 3, 4, 5]`

Output of the Program:

```
[1, 2, 6, 3, 4, 5]
```

Q4. Write a program to sort a stack using only another stack.

Code:

```
def sort_stack(stack):

    if not stack:
        return []

    top_element = stack.pop()
    srt_stack = sort_stack(stack)

    temp_stack = []
    while srt_stack and srt_stack[-1] >= top_element:
        temp_stack.append(srt_stack.pop())

    temp_stack.append(top_element)

    while temp_stack:
        srt_stack.append(temp_stack.pop())
    return srt_stack

input_stack = [9, 5, 1, 3]
output_stack = sort_stack(input_stack)
print(output_stack)
```

Code Explanation:

1. **Function Name:**
    a. `sort_stack(stack)`: This function sorts a stack in ascending order using recursion and another stack.
2. **Logic:**
    a. The algorithm follows a recursive approach:
        i. Remove the top element from the input stack (`top_element`).
        ii. Recursively sort the remaining stack (`srt_stack`).
        iii. Use a temporary stack (`temp_stack`) to hold elements from the sorted stack that are greater than `top_element`.
        iv. Insert `top_element` into its correct position in the sorted stack.
        v. Move elements back from `temp_stack` to `srt_stack`.
3. **Input:**
    a. `input_stack = [9, 5, 1, 3]`
4. **Output:**
    a. `outut_stack = [1, 3, 5, 9]`
    b.

Output of the Program:

```
[1, 3, 5, 9]
```

Q5. Create a module "pascal.py" with function `pascalTriangle(numOfRows)` and import it into "main.py".

Code :

```python
def pascalTriangle(numOfRows):

    tria = [[1] * (i + 1) for i in range(numOfRows)]

    for row in range(2, numOfRows):
        for col in range(1, row):
            tria[row][col] = tria[row - 1][col - 1] + tria[row - 1][col]

    # Maximum spacing calculation
    max_num = max(max(row) for row in tria)
    num_width = len(str(max_num)) + 2


    for row in tria:
        row_str = "".join(f"{num:>{num_width}}" for num in row)
        print(row_str.center(numOfRows * num_width))  # Center align

pascalTriangle(6)
```

Code Explanation:

1. **Logic:**
    a. A 2D list (`tria`) is created where each row represents a level in Pascal's Triangle.
    b. The first and last elements of each row are 1.
    c. Other elements are computed as the sum of the two elements directly above them.
2. **Formatting:**
    a. The largest number in the triangle is used to calculate uniform spacing for alignment.
    b. Each row is center-aligned for neat visualization.
3. **Input:**
    a. `numOfRows = 6`

4. **Output:**

```
                    1
                 1     1
              1     2     1
           1     3     3     1
        1     4     6     4     1
     1     5    10    10     5     1
```

Q6. Create a 6x6 matrix with random values and:

Code:

```
matrix = [[np.round(rd.random(), 4) for _ in range(6)] for _ in range(6)]
print("\nRandomly generated 9x9 matrix:")
for row in matrix:
    print(row)

matrix = [[1 if val > 0.5 else 0 for val in row] for row in matrix]
print("\nModified  9x9 matrix:")
for row in matrix:
    print(row)

submatrix = [row[2:5] for row in matrix[2:5]]
print("\nSubmatrix:")
for row in submatrix:
    print(row)

print(f"\nMean of submatrix: {np.mean(submatrix)}")
```

Code Explanation:

1. **Step 1**: Generate a 6x6 Matrix with Random Values
    a. A 6x6 matrix is generated using random values between `0` and `1` from the `random` module.
2. **Step 2**: Modify the Matrix
    a. Each value in the matrix is replaced:
        i. If the value is greater than `0.5`, it is replaced with `1`.
        ii. Otherwise, it is replaced with `0`.
3. **Step 3**: Extract a Submatrix
    a. A 3x3 submatrix is extracted starting from index `(2, 2)`.
4. **Step 4**: Calculate the Mean
    a. The mean of the extracted submatrix is calculated using NumPy's `mean` function.

Output:

```
Randomly generated 9x9 matrix:
[0.5731, 0.3694, 0.8643, 0.747, 0.8614, 0.5302]
[0.3749, 0.1778, 0.9005, 0.4297, 0.5222, 0.2674]
[0.7187, 0.9734, 0.572, 0.6491, 0.846, 0.8352]
[0.3682, 0.719, 0.4179, 0.5, 0.8146, 0.8464]
[0.3162, 0.7336, 0.543, 0.7824, 0.9051, 0.17]
[0.4981, 0.7566, 0.2699, 0.6434, 0.9488, 0.8958]

Modified  9x9 matrix:
[1, 0, 1, 1, 1, 1]
[0, 0, 1, 0, 1, 0]
[1, 1, 1, 1, 1, 1]
[0, 1, 0, 0, 1, 1]
[0, 1, 1, 1, 1, 0]
[0, 1, 0, 1, 1, 1]

Submatrix:
[1, 1, 1]
[0, 0, 1]
[1, 1, 1]

Mean of submatrix: 0.7777777777777778
```

Q7. Array Reshaping

Code:

```
array = np.array([int(np.floor(rd.random() * 10)) for i in range(16)])
matrix = array.reshape(4, 4)

print("1D array:")
print(array)

print("\nReshaped 4x4 matrix:")
print(matrix)

matrix_3x3x3 = np.random.randint(1, 10, size=(3, 3, 3))
flattened_array = matrix_3x3x3.flatten()

print("\n3D array:")
print(matrix_3x3x3)

print("\nFlattened 1D array:")
print(flattened_array)

reshaped_matrix = matrix_3x3x3.reshape(3, 9)

print("\nReshaped matrix:")
print(reshaped_matrix)
```

Code Explanation:

1. Create a 1D Array with 16 Elements:
   a. A 1D array with 16 random integers between `0` and 9 is generated.
   b. It is reshaped into a 4x4 matrix using the `reshape()` method.
2. Flatten a 3x3x3 Array:
   a. A 3D array with shape `(3, 3, 3)` is created with random integers between 1 and 9.
   b. The 3D array is flattened into a 1D array using the `flatten()` method.
3. Reshape the Matrix:
   a. The 3D array is reshaped into a 3x9 matrix while retaining all its original data.
4. Output:

```
1D array:
[5 9 5 0 9 5 7 6 6 9 4 8 0 6 8 2]

Reshaped 4x4 matrix:
[[5 9 5 0]
 [9 5 7 6]
 [6 9 4 8]
 [0 6 8 2]]

3D array:
[[[4 4 8]
  [7 1 9]
  [3 5 7]]

 [[3 3 9]
  [9 1 1]
  [1 1 9]]

 [[9 3 7]
  [6 4 2]
  [3 5 6]]]

Flattened 1D array:
[4 4 8 7 1 9 3 5 7 3 3 9 9 1 1 1 1 9 9 3 7 6 4 2 3 5 6]

Reshaped matrix:
[[4 4 8 7 1 9 3 5 7]
 [3 3 9 9 1 1 1 1 9]
 [9 3 7 6 4 2 3 5 6]]
```

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

Code:

```
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n - 1) + fibonacci(n - 2)

def fibonacci_sum(n):
    fib_sum = 0
    for i in range(n):
        fib_sum += fibonacci(i)
    return fib_sum

inputs = [1, 4]
for val in inputs:
    print(f"Fibbonacci_Sum({val}): {fibonacci_sum(val)}")
```

Code Explanation:

1. **Fibonacci Sequence:**
   a. The Fibonacci series starts with `0` and `1`, and each subsequent number is the sum of the two preceding numbers.
   b. The series is: `0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ....`
2. **Function 1: `fibonacci(n)`**
   a. A recursive function to compute the nth Fibonacci number.
   b. Base case: `fibonacci(0) = 0`, `fibonacci(1) = 1`.
   c. Recursive case: `fibonacci(n) = fibonacci(n - 1) + fibonacci(n - 2)`.
3. **Input and Output:**
   a. **Input:** 1 → **Output:** `0` (Sum of first 1 Fibonacci number).
   b. **Input:** 4 → **Output:** 4 (Sum of first 4 Fibonacci numbers: `0 + 1 + 1 + 2`)

Output of the Program:

```
Fibbonacci_Sum(1): 0
Fibbonacci_Sum(4): 4
```

Q9. Define a function `get_value_from_dict` to fetch a value from a dictionary using a key.

Code:

```
def get_value_from_dict(dictionary, key):
    try:
        return dictionary[key]
    except KeyError:
        raise KeyError(f"Key '{key}' not found in dictionary")

mp = { i: i ** 2 for i in range(10) }
print(f"Dictionary where value is square of key: \n{mp}\n")

keys = [4, 60, 2, 40]
for key in keys:
    try:
        print(f"Value of key {key}: {get_value_from_dict(mp, key)}")
    except KeyError as e:
        print(e)
```

Code Explanation:

1. **Function Name:**
    a. `get_value_from_dict(dictionary, key)`:
        i. Takes a dictionary and a key as parameters.
        ii. If the key exists, it returns the corresponding value.
        iii. If the key does not exist, it raises a `KeyError` with a custom error message.
2. **Main Program:**
    a. A dictionary `mp` is defined, where each key is an integer, and the value is its square. Example: `{0: 0, 1: 1, 2: 4, ..., 9: 81}`.
    b. A list of keys (`keys = [4, 60, 2, 40]`) is used to fetch values from the dictionary.
3. **Error Handling:**
    a. The `try-except` block is used to handle the `KeyError`.
    b. If a key is not found, a user-friendly error message is displayed.
4. **Input and Output:**
    a. **Input Key:** 4 → **Output Value:** `16`.
    b. **Input Key:** `60` → **Error Message:** `Key '60' not found in dictionary`.

Output of the Program:

```
Dictionary where value is square of key:
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}

Value of key 4: 16
"Key '60' not found in dictionary"
Value of key 2: 4
"Key '40' not found in dictionary"
```

Q10: Visualizing Loan Dataset Using Various Python Visualization Tools

Code Explanation:

This code loads a dataset (`Loan_train.csv`) and visualizes the data using **matplotlib, seaborn, and plotly**. The goal is to create different types of plots, including histograms, bar charts, box plots, scatter plots, pie charts, heatmaps, and interactive visualizations.

Step-by-Step Breakdown of Visualizations:

1. **Load Dataset & Set Theme**

```python
file_path = "Loan_train.csv"
df = pd.read_csv(file_path)
sns.set_theme(style="darkgrid")
```

   a. Reads the dataset using `pandas`.
   b. Sets a `seaborn` theme for consistent styling.

2. **Histogram: Loan Amount Distribution**

```python
plt.figure(figsize=(8, 6))
sns.histplot(df["loan_amnt"], bins=30, kde=True)
plt.title("Distribution of Loan Amount")
plt.show()
```

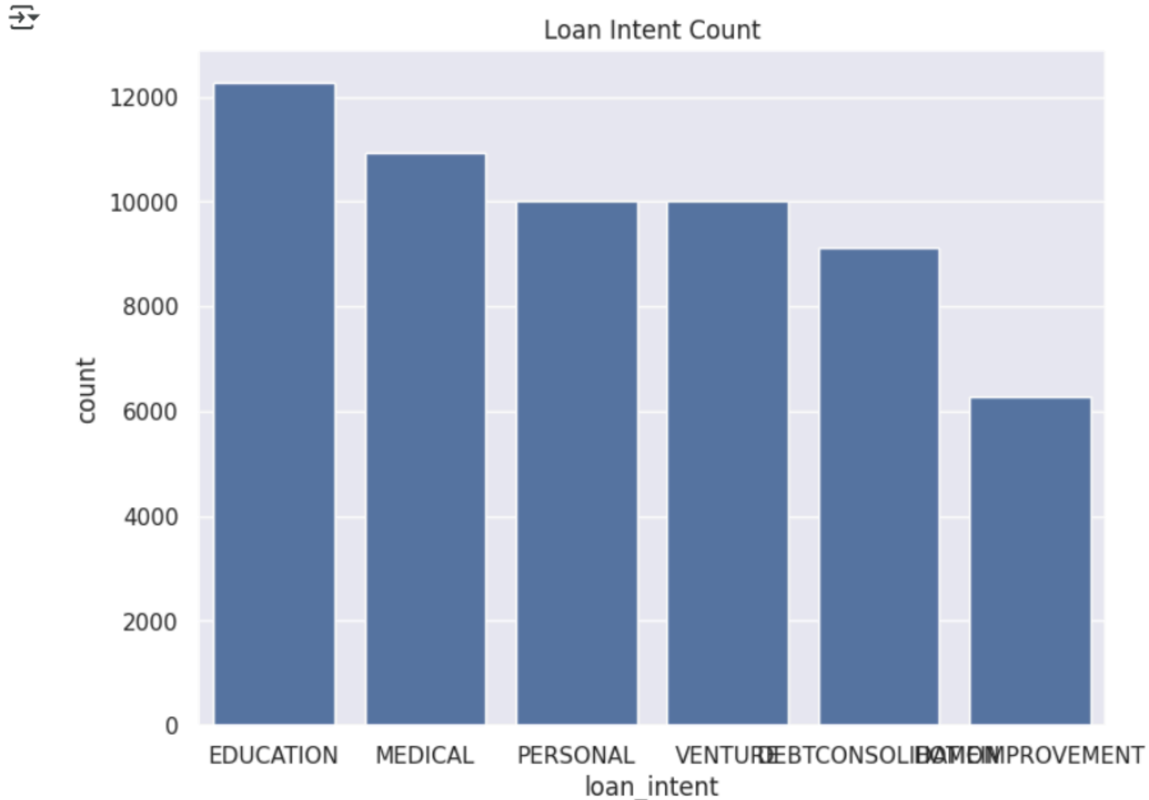
a. A histogram visualizes the distribution of `loan_amnt`.
b. `kde=True` adds a smooth density estimation curve.

3. Bar Chart: Loan Intent Count

```
[29] plt.figure(figsize=(8, 6))
     sns.countplot(x="loan_intent", data=df, order=df["loan_intent"].value_counts().index)
     plt.title("Loan Intent Count")
     plt.show()
```

a. A bar chart displays the count of each loan intent category

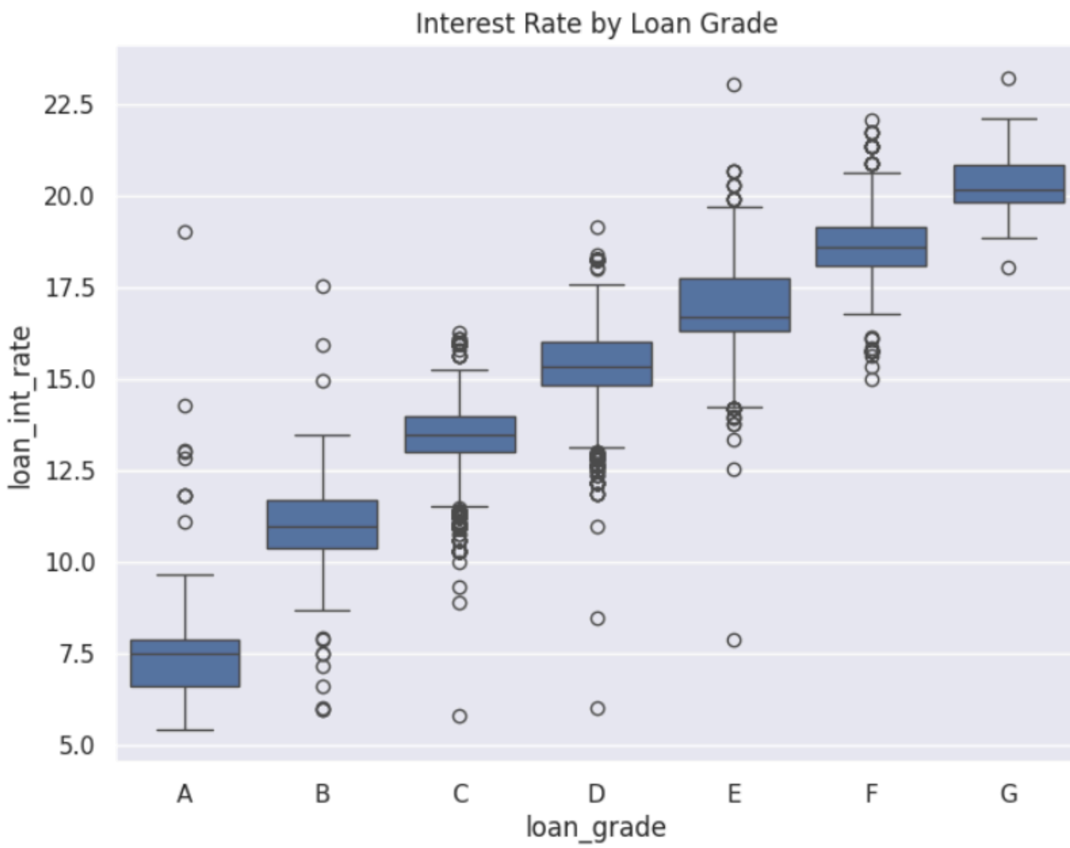
.



4. Box Plot: Interest Rate by Loan Grade

```
plt.figure(figsize=(8, 6))
sns.boxplot(x="loan_grade", y="loan_int_rate", data=df, order=sorted(df["loan_grade"].unique()))
plt.title("Interest Rate by Loan Grade")
plt.show()
```

a. A box plot compares `loan_int_rate` across different `loan_grade` categories.
b. Helps identify the distribution and presence of outliers.



5. **Scatter Plot: Income vs. Loan Amount**

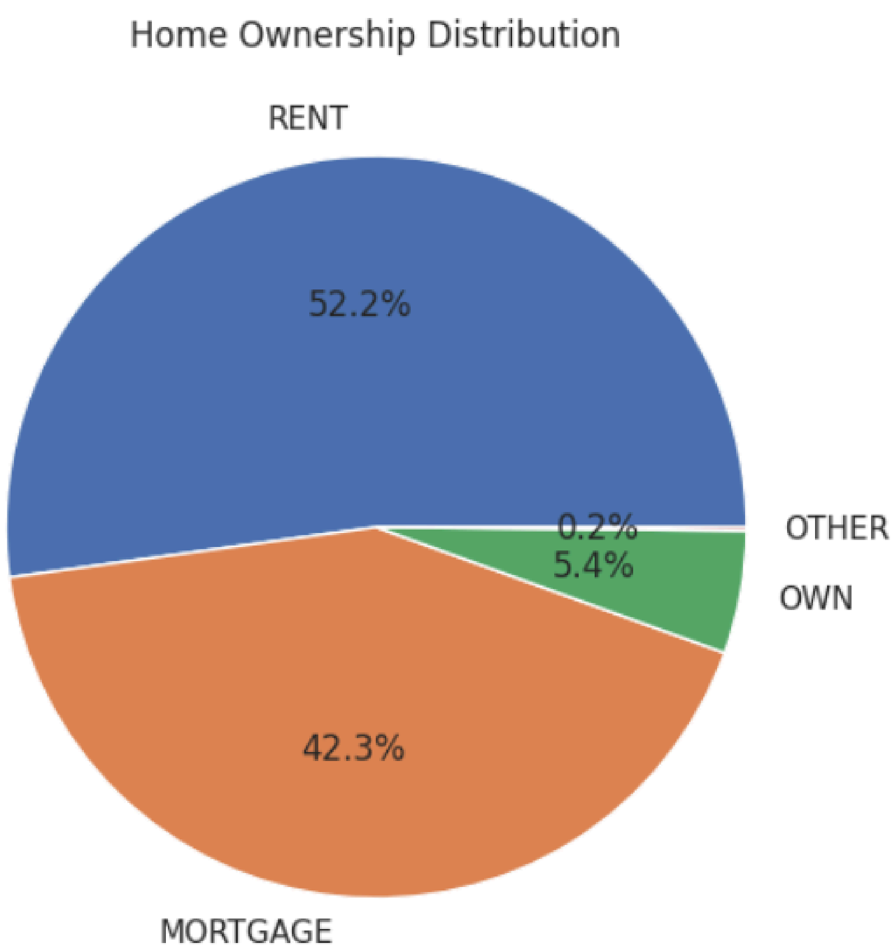
```python
plt.figure(figsize=(8, 6))
sns.scatterplot(x="person_income", y="loan_amnt", hue="loan_status", alpha=0.5, data=df)
plt.title("Income vs Loan Amount")
plt.show()
```

a. A scatter plot visualizes the relationship between `person_income` and `loan_amnt`.
b. Color (hue) represents loan status.

6. **Pie Chart: Home Ownership Distribution**

```python
plt.figure(figsize=(8, 6))
df["person_home_ownership"].value_counts().plot.pie(autopct="%1.1f%%")
plt.ylabel("")
plt.title("Home Ownership Distribution")
plt.show()
```

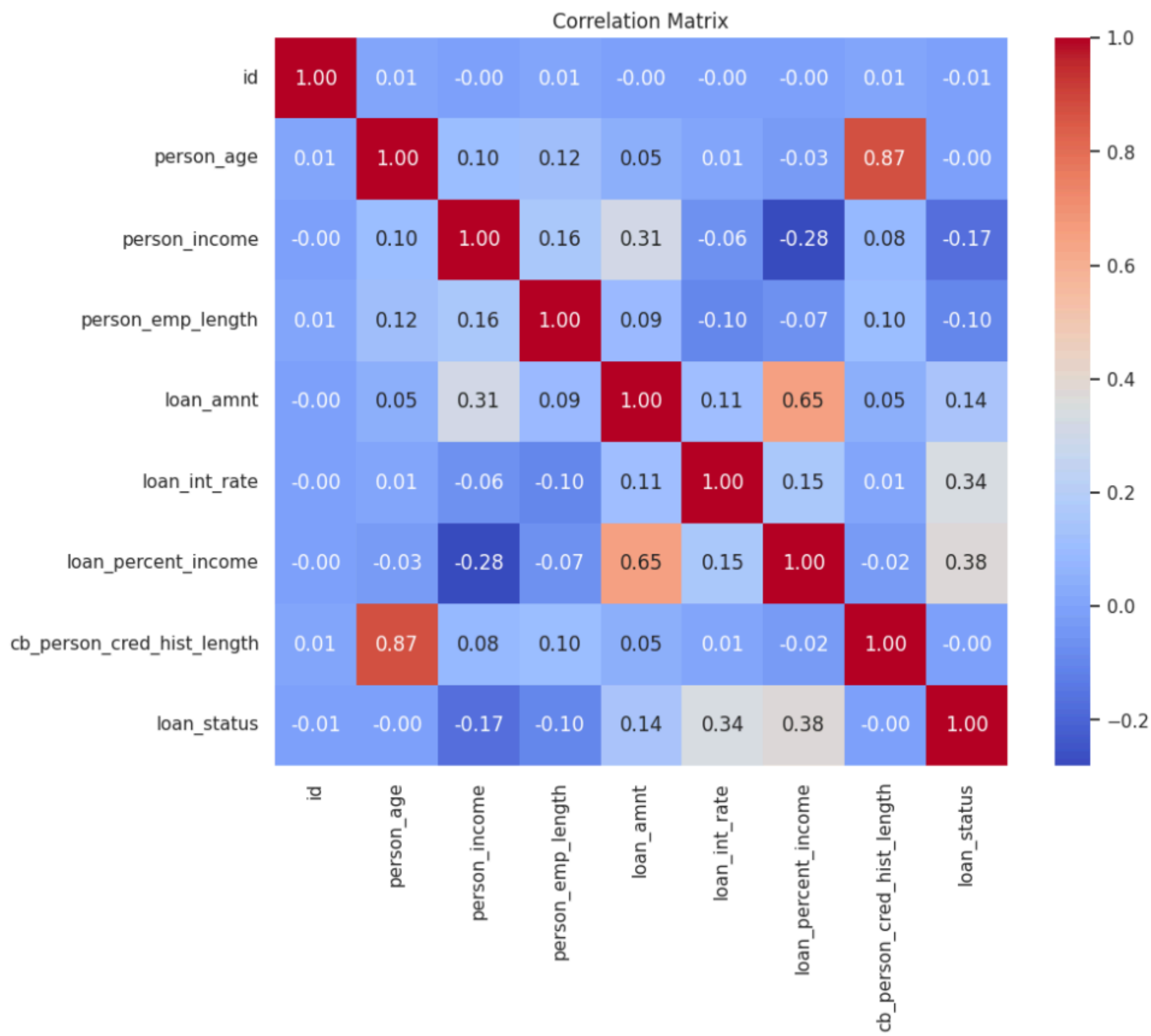
a. A pie chart **displays the proportion of home ownership types.**



7. **Heatmap: Correlation Matrix**

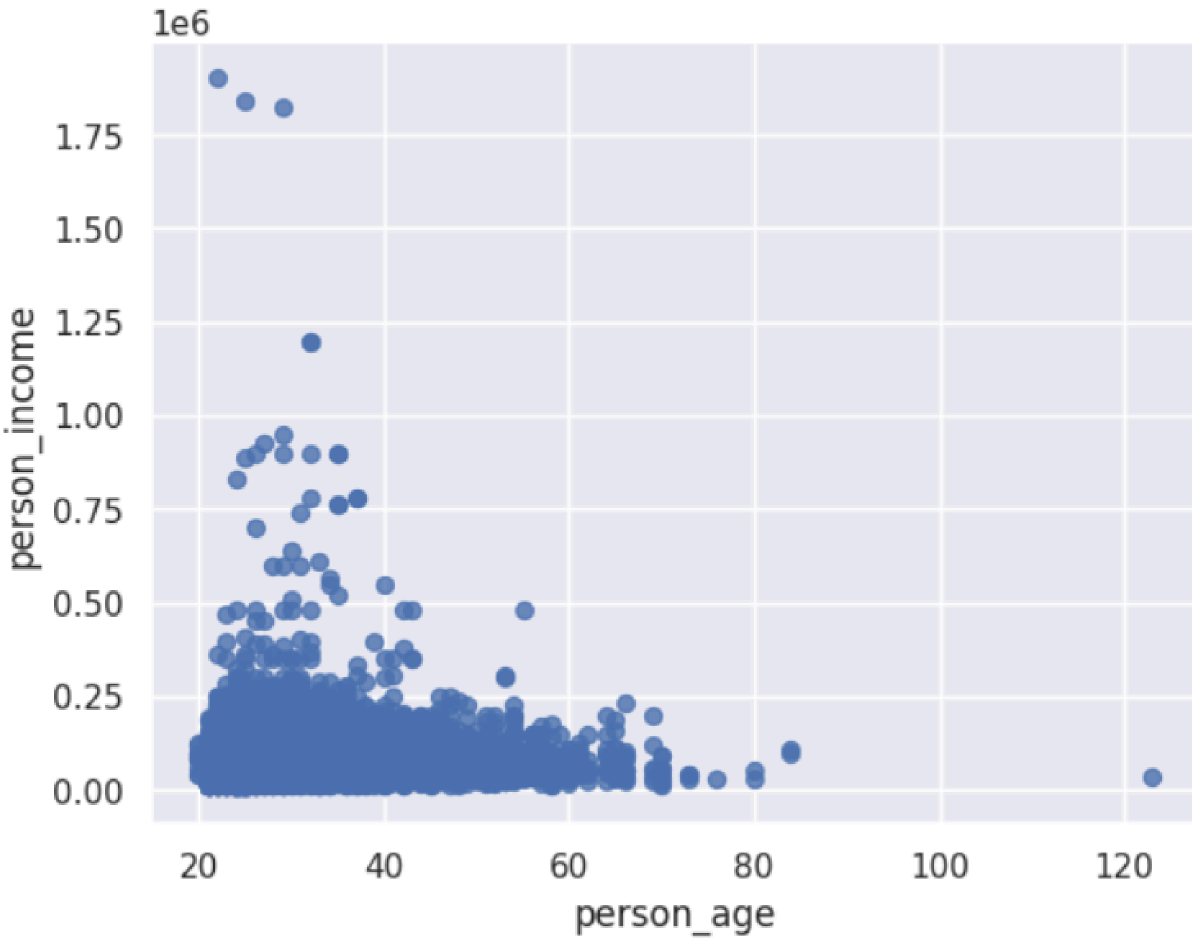
```python
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```

a. A heatmap represents correlation between numerical variables.



8. Income variation with respect to person age

```
df.plot.scatter(x='person_age', y='person_income', s=32, alpha=0.8)
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
```



9. **Interactive Scatter Plot (Plotly)**

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

a. An interactive scatter plot using `plotly`.