****

**IE416: Robot Programming Lab-01**

**Group Syntax**

**Yash Tarpara - 202201422**

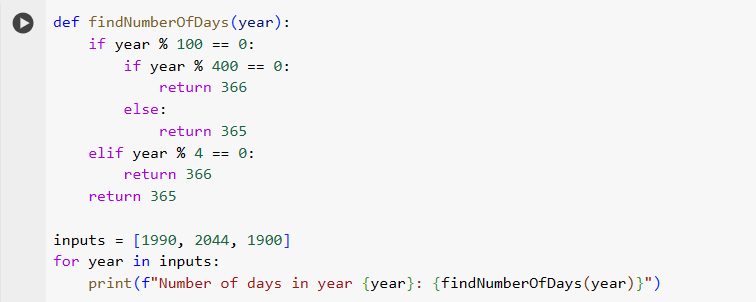
**Kaushik Prajapati - 202201472**

**Links:**

1. **GitHub Repository:** Access the course lab files – [[Click Here](https://github.com/Kaushik-Prajapati-05/IE416-Robot-Programming)].
2. **Notebook for Lab Question :** Direct link to the .ipynb file – [[Click Here](https://colab.research.google.com/drive/1N8ZZcJyWJPOlzYnXMrBVHG1UlpfGqTqO)].
3. **Learning NumPy:** Direct link to the .ipynb file – [[Click Here](https://colab.research.google.com/drive/1wSaWvL-2bl2cI3O7MPWbHIb0N08vJsQh)].
4. **Exploring Data Visualization with Python Libraries:** Direct link to the .ipynb file – [[Click Here](https://colab.research.google.com/drive/1j99zNUIBPN6t8HlPt3pSBUqDc3gotHKB)].

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

#### **Code:**

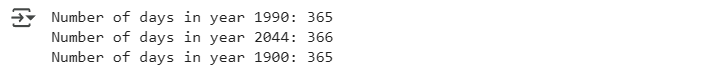


#### **Code Explanation:**

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

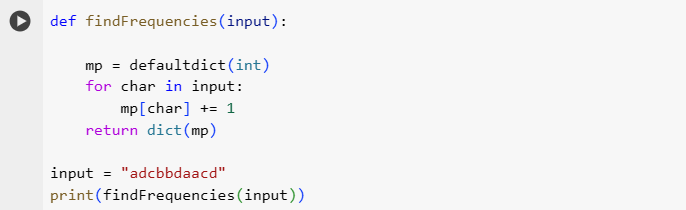
1. **Input Parameter:**
   1. year: An integer representing the year.
2. **Logic:**
   1. A year is a **leap year** if:
      1. It is divisible by 4, but **not divisible by 100**, or
      2. It is divisible by 400.
   2. If the year satisfies the leap year conditions, it returns 366 (indicating the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

#### **Output of the Program:**



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

#### **Code:**



#### **Code Explanation:**

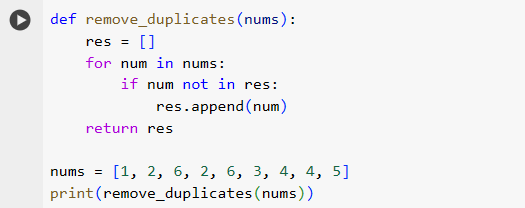
1. **Function Name:**
   1. findFrequencies(input): This function calculates the frequency of each character in a string.
2. **Logic:**
   1. A defaultdict of type int is used to store character frequencies, where the default value for each key is initialized to 0.
   2. The function iterates through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

****

### **Q3. Write a program to remove duplicates from a list but keep the first occurrence of each element.**

#### **Code:**



#### **Code Explanation:**

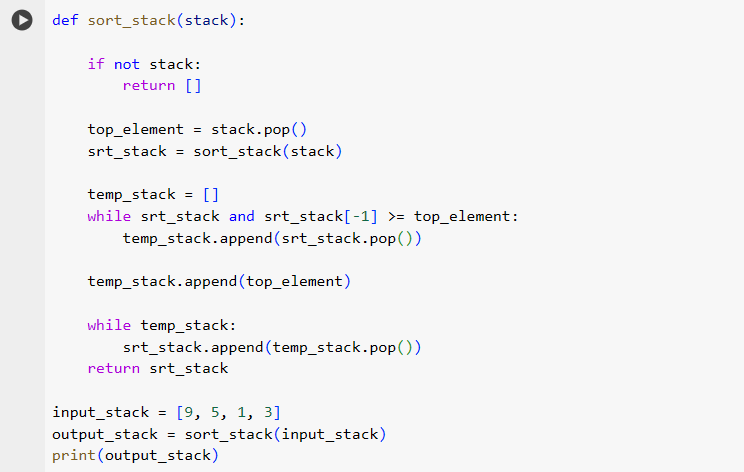
1. **Function Name:**
   1. remove\_duplicates(nums): This function removes duplicate elements from a list while preserving the order of their first occurrence.
2. **Logic:**
   1. A new list, res, is created to store unique elements.
   2. The program iterates through the input list, nums, and adds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

#### **Output of the Program:**

### 

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

#### **Code:**



#### **Code Explanation:**

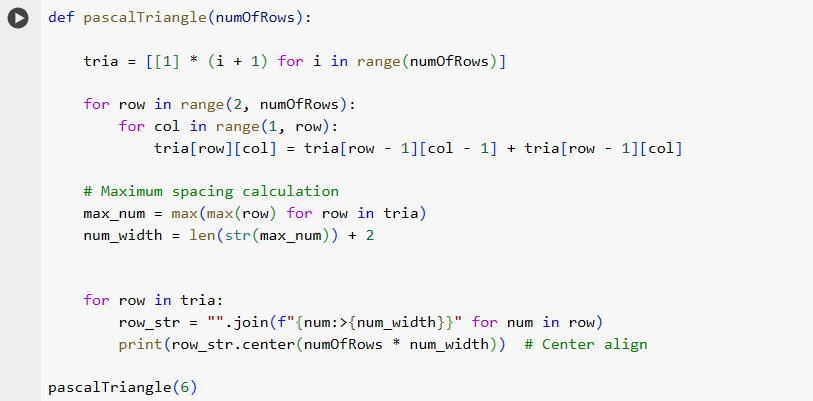
1. **Function Name:**
   1. sort\_stack(stack): This function sorts a stack in ascending order using recursion and another stack.
2. **Logic:**
   1. The algorithm follows a recursive approach:
      1. Remove the top element from the input stack (top\_element).
      2. Recursively sort the remaining stack (srt\_stack).
      3. Use a temporary stack (temp\_stack) to hold elements from the sorted stack that are greater than top\_element.
      4. Insert top\_element into its correct position in the sorted stack.
      5. Move elements back from temp\_stack to srt\_stack.
3. **Input:**
   1. input\_stack = [9, 5, 1, 3]
4. **Output:**
   1. outut\_stack = [1, 3, 5, 9]

#### **Output of the Program:**

****

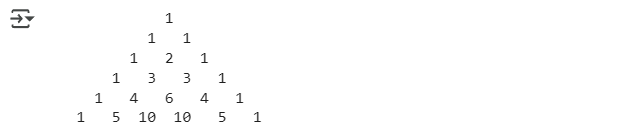
### **Q5. Create a module “pascal.py” with function pascalTriangle(numOfRows) and import it into “main.py”.**

#### **Code :**



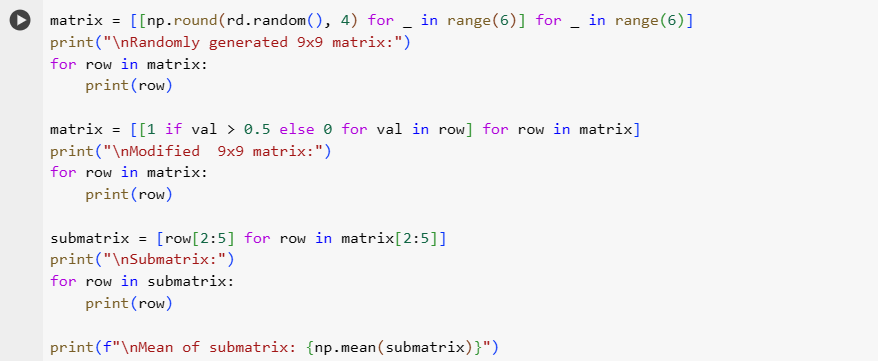
#### **Code Explanation:**

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

****

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

#### **Code:**

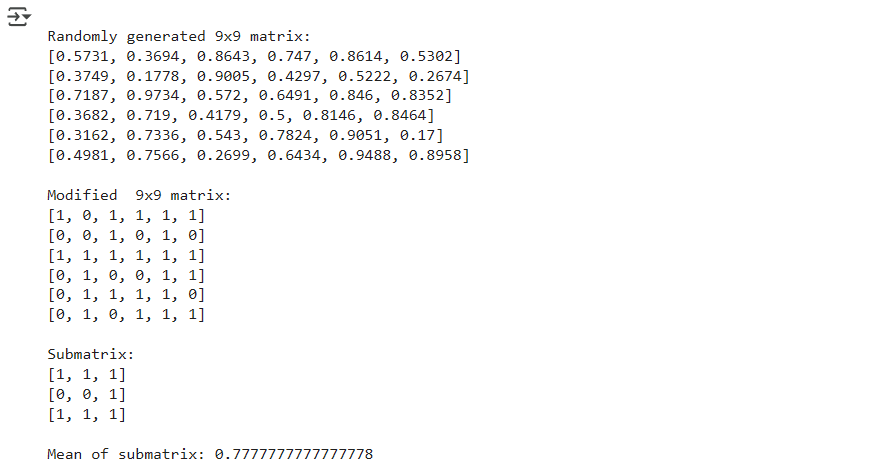


#### **Code Explanation:**

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

#### 

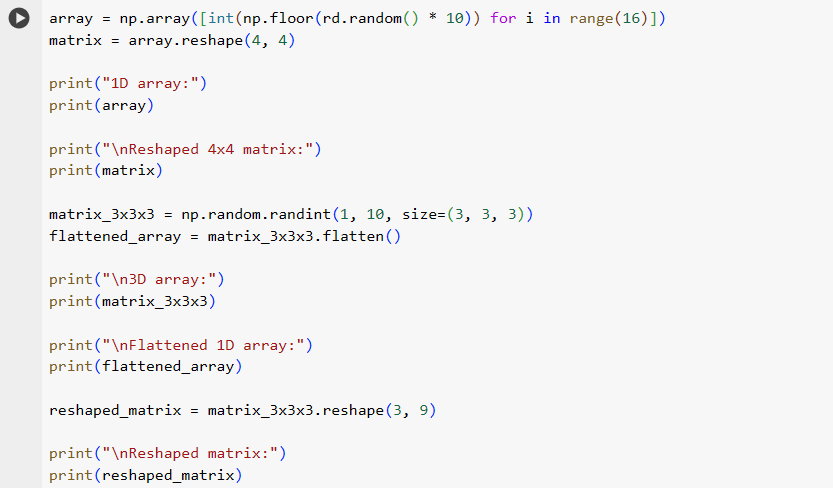
#### **Output:**

****

### 

### **Q7. Array Reshaping**

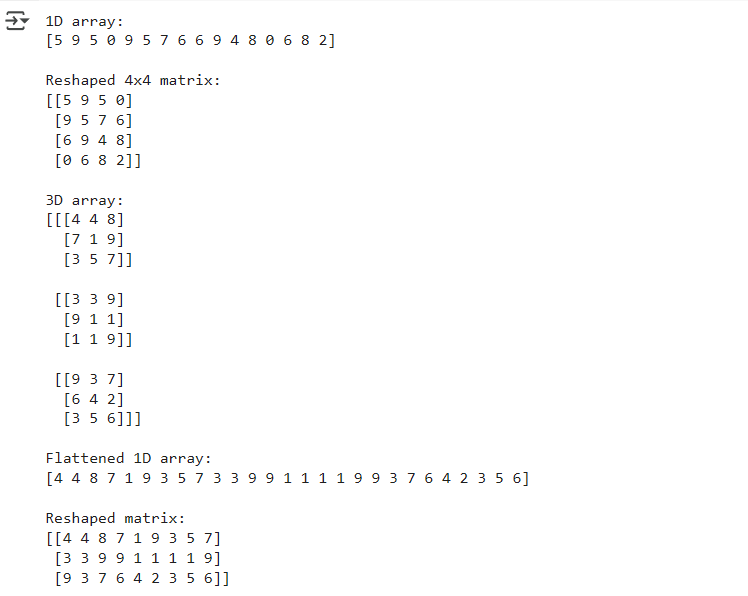
#### **Code:**



#### 

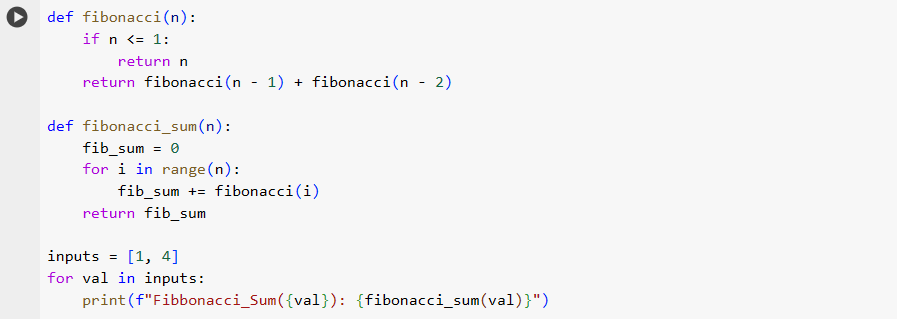
#### **Code Explanation:**

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

****

### **Q8. Write a recursive function fibonacci\_sum(n) to calculate the sum of the first n numbers in the Fibonacci series.**

#### **Code:**



#### **Code Explanation:**

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

#### **Output of the Program:**



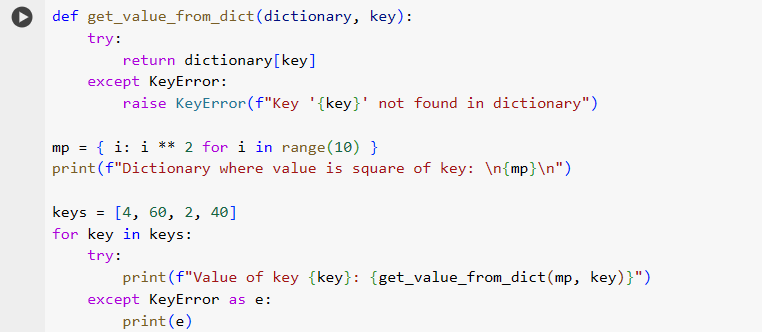
### 

### 

### 

### **Q9. Define a function get\_value\_from\_dict to fetch a value from a dictionary using a key.**

#### **Code:**

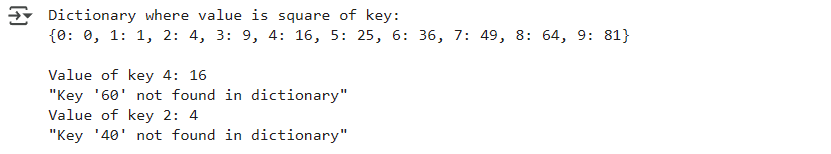


#### **Code Explanation:**

1. **Function Name:**
   1. get\_value\_from\_dict(dictionary, key):
      1. Takes a dictionary and a key as parameters.
      2. If the key exists, it returns the corresponding value.
      3. If the key does not exist, it raises a KeyError with a custom error message.
2. **Main Program:**
   1. 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}.
   2. A list of keys (keys = [4, 60, 2, 40]) is used to fetch values from the dictionary.
3. **Error Handling:**
   1. The try-except block is used to handle the KeyError.
   2. If a key is not found, a user-friendly error message is displayed.
4. **Input and Output:**
   1. **Input Key:** 4 → **Output Value:** 16.
   2. **Input Key:** 60 → **Error Message:** Key '60' not found in dictionary.

#### 

#### **Output of the Program:**



### 

### 

### 

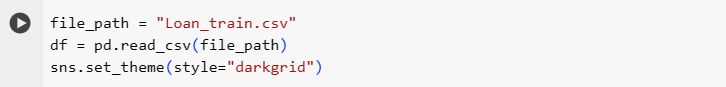
### **Q10: Visualizing Loan Dataset Using Various Python Visualization Tools**

### **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.

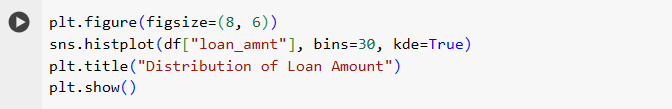
**Step-by-Step Breakdown of Visualizations:**

1. **Load Dataset & Set Theme**

****

* 1. Reads the dataset using pandas.
  2. Sets a seaborn theme for consistent styling.

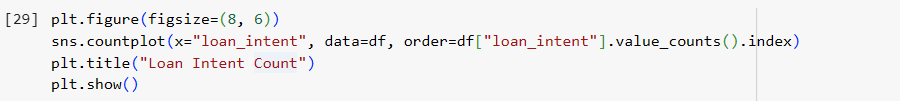
1. **Histogram: Loan Amount Distribution**

****

* 1. A histogram visualizes the distribution of loan\_amnt.
  2. kde=True adds a smooth density estimation curve.

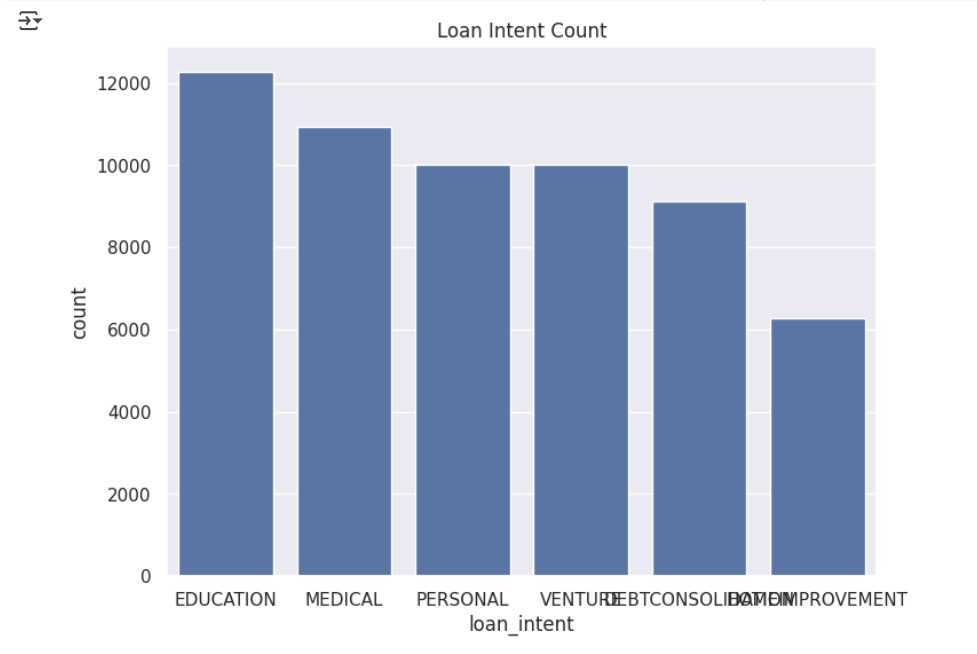


1. **Bar Chart: Loan Intent Count**

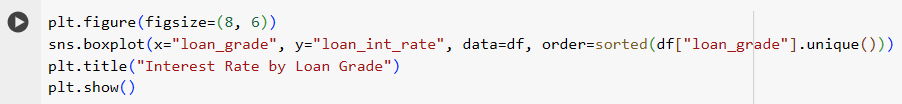
****

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

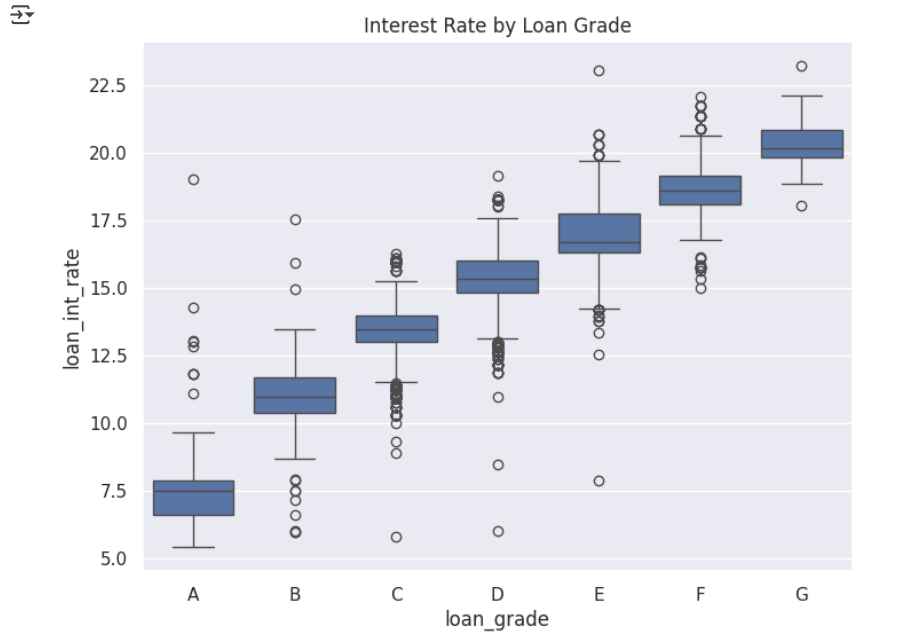
.



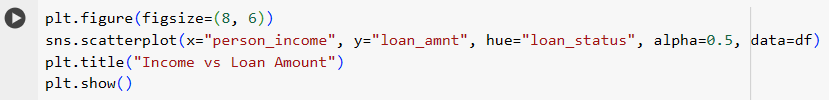
1. **Box Plot: Interest Rate by Loan Grade**

****

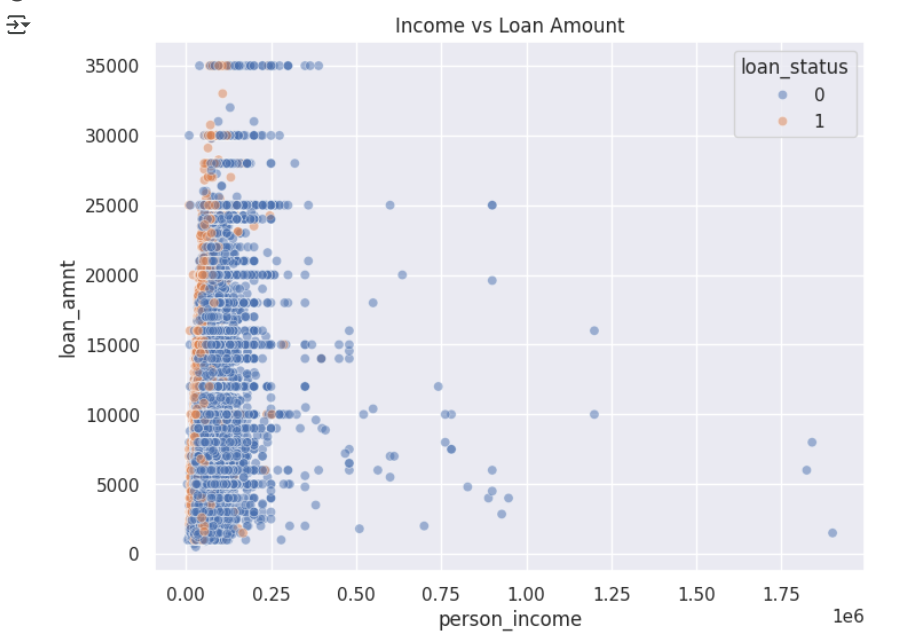
* 1. A box plot compares loan\_int\_rate across different loan\_grade categories.
  2. Helps identify the distribution and presence of outliers.



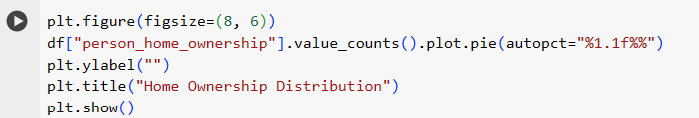
1. **Scatter Plot: Income vs. Loan Amount**

****

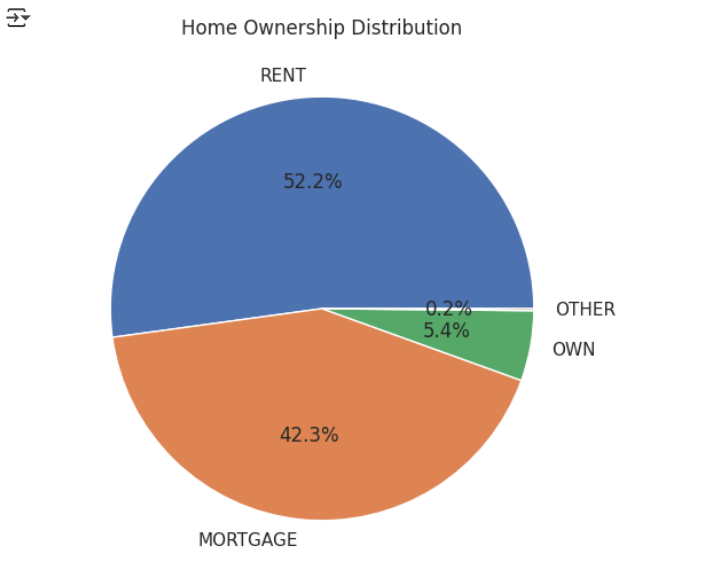
* 1. A scatter plot visualizes the relationship between person\_income and loan\_amnt.
  2. Color (hue) represents loan status.



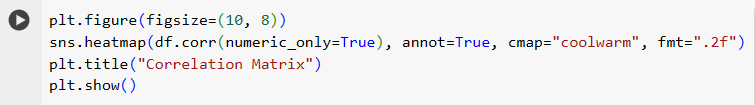
1. **Pie Chart: Home Ownership Distribution**

****

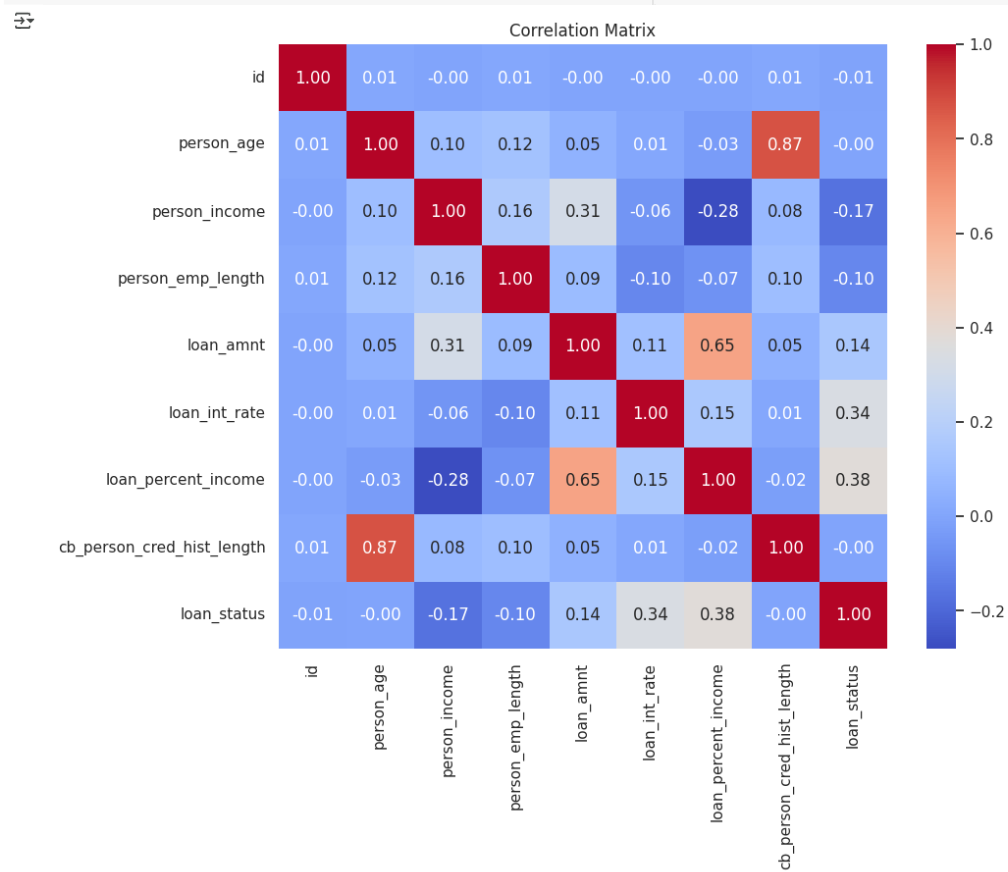
* 1. A pie cha**rt displays the proportion of home ownership types.**

****

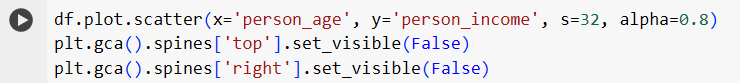
1. **Heatmap: Correlation Matrix**

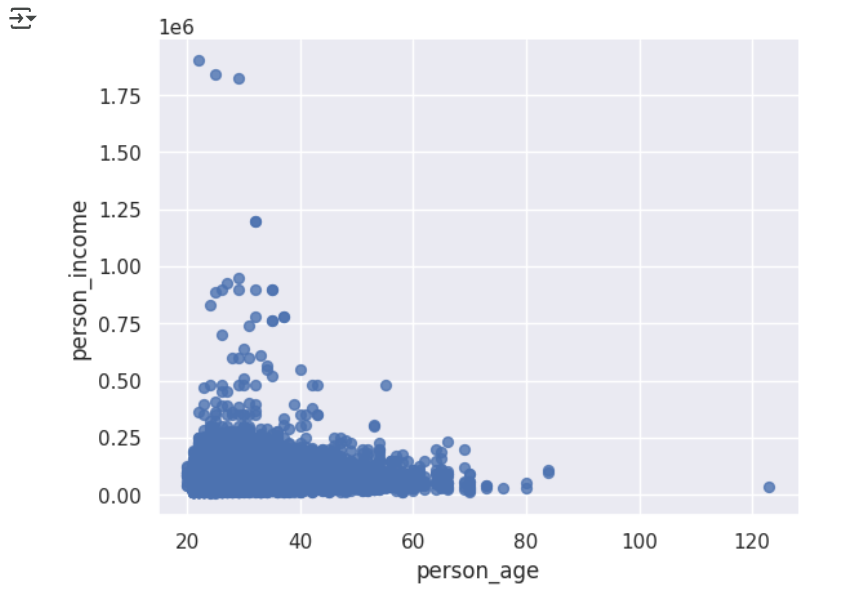
****

* 1. A heatmap represents correlation between numerical variables.

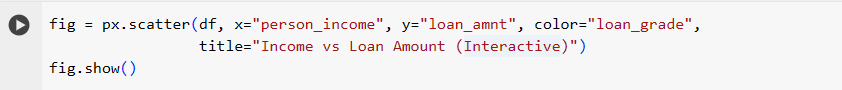


1. **Income variation with respect to person age**

****

****

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

****

* 1. An interactive scatter plot using plotly.

