

# **IE416: Robot Programming**

## **Lab Session I**

Basics of Python



**Group Name - Techno Titans**

Rutvik Vegad - 202201143  
Avinash Baraiya - 202201211

- Objective :-

- The main goal of this lab session was to introduce us to the fundamentals of Python programming by solving ten different problems. The session aimed to help us grasp essential Python concepts, including the use of the NumPy library, and apply them to implement solutions of varying complexity. Through hands-on practice in a coding environment, students gained experience working with Python commands and enhancing their problem-solving skills.

- Items Used by us :-

- Google Colaboratory
- Python 3.10 Key
- Libraries:
  - NumPy (np) - for numerical computations and array operations
  - Pandas (pd) - for data manipulation and analysis
  - Matplotlib (plt) - for basic data visualization
  - Seaborn (sns) - for statistical data visualization
  - Plotly Express (px) - for interactive data visualization

Google Collab Link :-

**Results and Observations**

 [IE416\\_Lab1.ipynb](#)

**NumPy**


 [NumPy.ipynb](#)

**Visualization**

 [Visualization\\_.ipynb](#)


## → Questions solved :-


### 1. Write a function that gives number of days of given year ?

```
 def number_of_days(year):  
    # Check if the year is a leap year  
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):  
        return 366  
    else:  
        return 365  
  
    # Test the function  
print(number_of_days(1990)) # Output: 365  
print(number_of_days(2044)) # Output: 366
```

- To determine the number of days in a given year, we need to check whether the year is a leap year or not.
- A leap year has 366 days, whereas a non-leap year has 365 days.
- A year is considered a leap year if:
  - It is divisible by 4 but not by 100, OR It is divisible by 400.



### 2. Count the frequency of each character in a string and store it in a dictionary. An example is given below ?

```
 # Q2. Count the frequency of each character in a string and store it in a dictionary.  
def char_frequency(s):  
    freq = {}  
    for char in s:  
        if char in freq:  
            freq[char] += 1  
        else:  
            freq[char] = 1  
    return freq  
  
    # Test the function  
print(char_frequency('adcbddaacd')) # Output: {'a': 3, 'd': 3, 'c': 2, 'b': 2}
```

```
 {'a': 3, 'd': 3, 'c': 2, 'b': 2}
```

- To count the frequency of each character in a string, iterate through the string and store character counts in a dictionary. If a character is already present, increment its count; otherwise, add it with an initial count of 1.

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

```
 def remove_duplicates(lst):  
    seen = set()  
    result = []  
    for item in lst:  
        if item not in seen:  
            seen.add(item)  
            result.append(item)  
    return result  
  
# Test the function  
input_list = [1, 2, 3, 4, 2, 3, 5, 6, 1, 4]  
print(remove_duplicates(input_list)) # Output: [1, 2, 3, 4, 5, 6]  
  
 [1, 2, 3, 4, 5, 6]
```

- To remove duplicates from a list while maintaining the order of first occurrences, we can use a set to track seen elements and a list to store unique values. As we iterate through the list, we add elements to the new list only if they haven't been seen before.

4. Write a program to sort a stack using only another stack (no other data structures like arrays or linked lists) ?

```
def sort_stack(input_stack):
    temp_stack = []

    while input_stack:
        # Pop an element from the input stack
        temp = input_stack.pop()

        # While temporary stack is not empty and top of temp_stack is greater than temp
        while temp_stack and temp_stack[-1] > temp:
            # Pop from temp_stack and push it to the input stack
            input_stack.append(temp_stack.pop())

        # Push temp in temp_stack
        temp_stack.append(temp)

    # Transfer sorted elements back to input_stack
    while temp_stack:
        input_stack.append(temp_stack.pop())

    return input_stack

# Test the function
stack = [9, 5, 1, 3]
sorted_stack = sort_stack(stack)
print(sorted_stack) # Output: [1, 3, 5, 9]
```

[9, 5, 3, 1]

- To sort a stack using only another stack, we follow a **sorting insertion approach**:
  1. Initialize an empty auxiliary stack (**sorted\_stack**).
  2. Pop elements from the original stack (**input\_stack**) one by one.
  3. Place each popped element in its correct position in **sorted\_stack**:
    - If the top of **sorted\_stack** is greater than the popped element, move elements back to **input\_stack** until the right position is found.
    - Push the popped element onto **sorted\_stack**.
  4. Repeat until **input\_stack** is empty.
  5. The **sorted\_stack** now contains elements in sorted order (smallest at the bottom, largest at the top).

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

```
def pascalTriangle(numOfRows):  
    result = []  
    for i in range(numOfRows):  
        row = [1] * (i + 1)  
        for j in range(1, i):  
            row[j] = result[i - 1][j - 1] + result[i - 1][j]  
        result.append(row)  
    return result  
  
pascalPrint = pascalTriangle(7)  
for row in pascalPrint:  
    print(' ' * (len(pascalPrint) - len(row)), end=' ')  
    print(' '.join(map(str, row)))
```

```
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
```

- To generate Pascal's Triangle, we define a function `pascalTriangle(numOfRows)` in a separate module `pascal.py`. This function constructs the triangle row by row, where each row is formed based on the values of the previous row. We then import and use this function in `main.py`.

6. 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

# Create a 6x6 matrix with random values
matrix = np.random.rand(6, 6)
print("Original Matrix:\n", matrix)

# Replace all values greater than 0.5 with 1, and all others with 0
matrix[matrix > 0.5] = 1
matrix[matrix <= 0.5] = 0
print("Modified Matrix:\n", matrix)

# Extract a 3x3 submatrix starting from index (2, 2)
submatrix = matrix[2:5, 2:5]
print("3x3 Submatrix:\n", submatrix)

# Calculate the mean of the submatrix
mean_value = np.mean(submatrix)
print("Mean of the 3x3 Submatrix:", mean_value)
```

Original Matrix:

```
[[0.7881975  0.63668192 0.61340385 0.98275369 0.28059997 0.24656002]
 [0.06706819 0.91066652 0.3583643  0.46155881 0.47676433 0.52862127]
 [0.11405751 0.59594391 0.36236733 0.94005207 0.48492507 0.49799924]
 [0.64960609 0.17765795 0.20510469 0.97721074 0.81525051 0.52144702]
 [0.50364409 0.58912334 0.74269166 0.66877679 0.95057508 0.93287522]
 [0.42687909 0.68035739 0.278724   0.54460455 0.80435973 0.79118823]]
```

Modified Matrix:

```
[[1. 1. 1. 1. 0. 0.]
 [0. 1. 0. 0. 0. 1.]
 [0. 1. 0. 1. 0. 0.]
 [1. 0. 0. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1.]
 [0. 1. 0. 1. 1. 1.]]
```

3x3 Submatrix:

```
[[0. 1. 0.]
 [0. 1. 1.]
 [1. 1. 1.]]
```

Mean of the 3x3 Submatrix: 0.6666666666666666

- Generate a 6×6 matrix with random values between 0 and 1 using NumPy.
- Replace values: Convert all values greater than 0.5 to 1, and the rest to 0.

- Extract a 3×3 submatrix starting from index (2,2).
- Compute the mean of the extracted submatrix.

## 7. Array Reshaping:

- a. 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

# Create a 1D array with 16 elements
array_1d = np.arange(16)
print("\n\n1D Array:\n", array_1d)

# Reshape it into a 4x4 matrix
matrix_4x4 = array_1d.reshape(4, 4)
print("\n\n4x4 Matrix:\n", matrix_4x4)

# Create a 3x3x3 array
array_3x3x3 = np.arange(27).reshape(3, 3, 3)
print("\n\n3x3x3 Array:\n", array_3x3x3)

# Flatten the 3x3x3 array into a 1D array
flattened_array = array_3x3x3.flatten()
print("\n\nFlattened Array:\n", flattened_array)

# Reshape a matrix into a new shape without changing its data
reshaped_matrix = matrix_4x4.reshape(2, 8)
print("\n\nReshaped Matrix (2x8):\n", reshaped_matrix)
```



```
1D Array:
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]

4x4 Matrix:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

3x3x3 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]
 [24 25 26]]]

Flattened 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
 24 25 26]

Reshaped Matrix (2x8):
[[ 0  1  2  3  4  5  6  7]
 [ 8  9 10 11 12 13 14 15]]
```

- Create a 1D array with 16 elements and reshape it into a 4×4 matrix.
- Flatten a 3×3×3 array into a 1D array.
- Reshape a matrix into a new shape while keeping its data unchanged.

8. 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 ?

```
def Fibonacci_sum(n):  
    # Base cases  
    if n <= 0:  
        return 0  
    elif n == 1:  
        return 0  
    elif n == 2:  
        return 1  
    else:  
        # Recursive case  
        return Fibonacci_sum(n-1) + Fibonacci(n-1)  
  
def Fibonacci(n):  
    if n <= 0:  
        return 0  
    elif n == 1:  
        return 1  
    else:  
        return Fibonacci(n-1) + Fibonacci(n-2)  
  
# Test the function  
print(Fibonacci_sum(1)) # Output: 0  
print(Fibonacci_sum(4)) # Output: 4  
print(Fibonacci_sum(7)) # Output: 20
```

```
0  
4  
20
```

- To calculate the sum of the first  $n$  Fibonacci numbers recursively, we follow these steps:
  - Define the Fibonacci sequence where:  $F(0) = 0$ ,  $F(1) = 1$   $F(n) = F(n-1) + F(n-2)$  for  $n \geq 2$

- Calculate the sum recursively by adding the n-th Fibonacci number to the sum of the previous n-1 terms.
- Base cases:
  - If n = 0, return 0 (sum of the empty sequence).
  - If n = 1, return 0 (since the sum of [0] is 0).
  - If n = 2, return 1 (since the sum of [0, 1] is 1).
- Recursive call to compute Fibonacci\_sum(n-1) + Fibonacci(n-1), where Fibonacci(n-1) gives the last term.

9. 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(d, key):
    if key not in d:
        raise KeyError(f"Key '{key}' not found in the dictionary.")
    return d[key]

def main():
    sample_dict = {
        'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10,
        'k': 11, 'l': 12
    }
    user_key = 'a' # Static key

    try:
        value = get_value_from_dict(sample_dict, user_key)
        print(f"The value for key '{user_key}' is {value}.")
    except KeyError as e:
        print(e)

# Call the main function
main()
```

The value for key 'a' is 1.

- Define `get_value_from_dict`: This function takes a dictionary and a key as parameters. It tries to return the value corresponding to

the key. If the key is not found, it raises a `KeyError` with a custom error message.

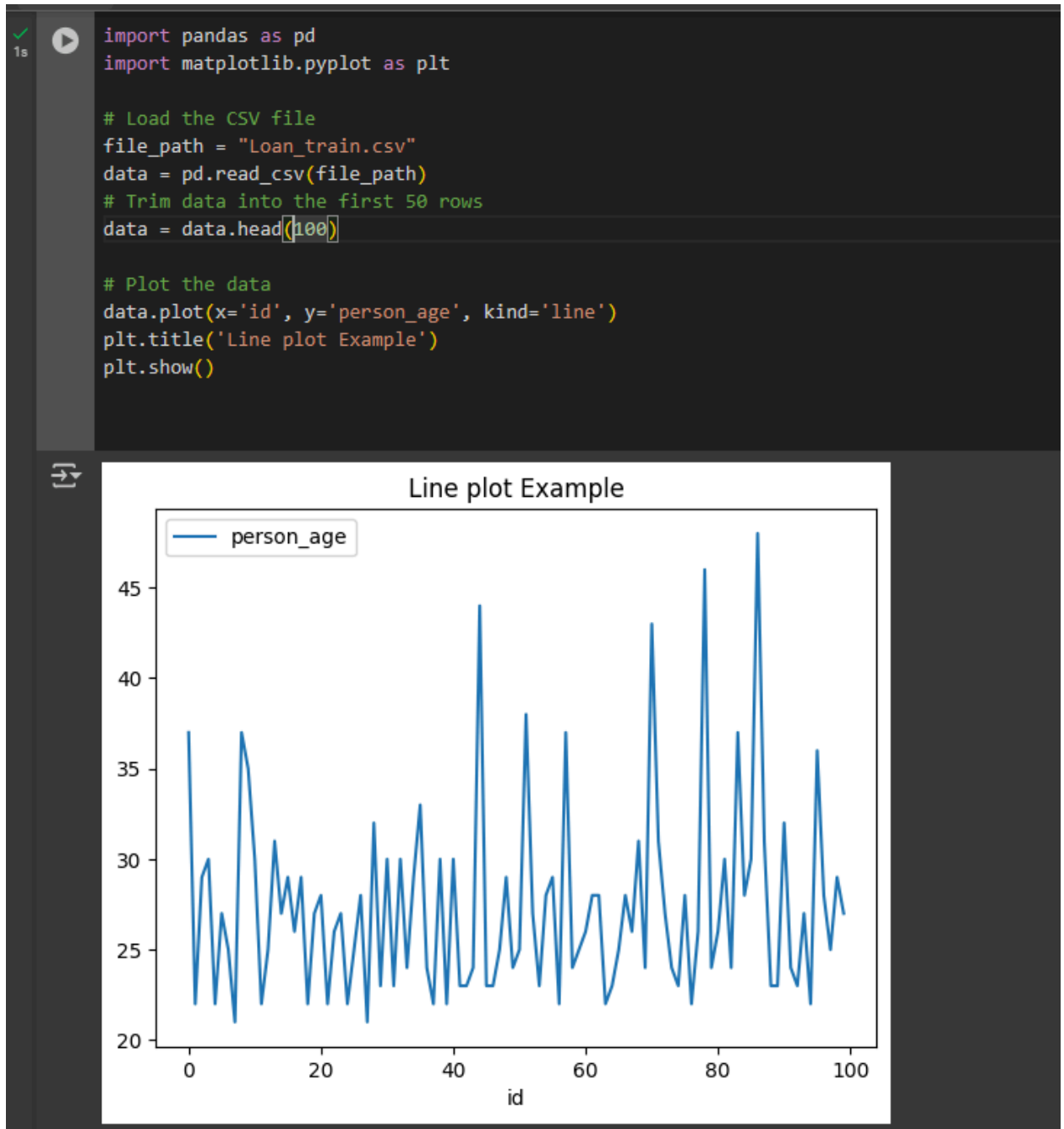
- **Define a `main` function:** This function prompts the user to provide a key, calls `get_value_from_dict`, and handles the `KeyError` to display a user-friendly message if the key is not found in the dictionary.

## 10. Visualization of Data :=

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.

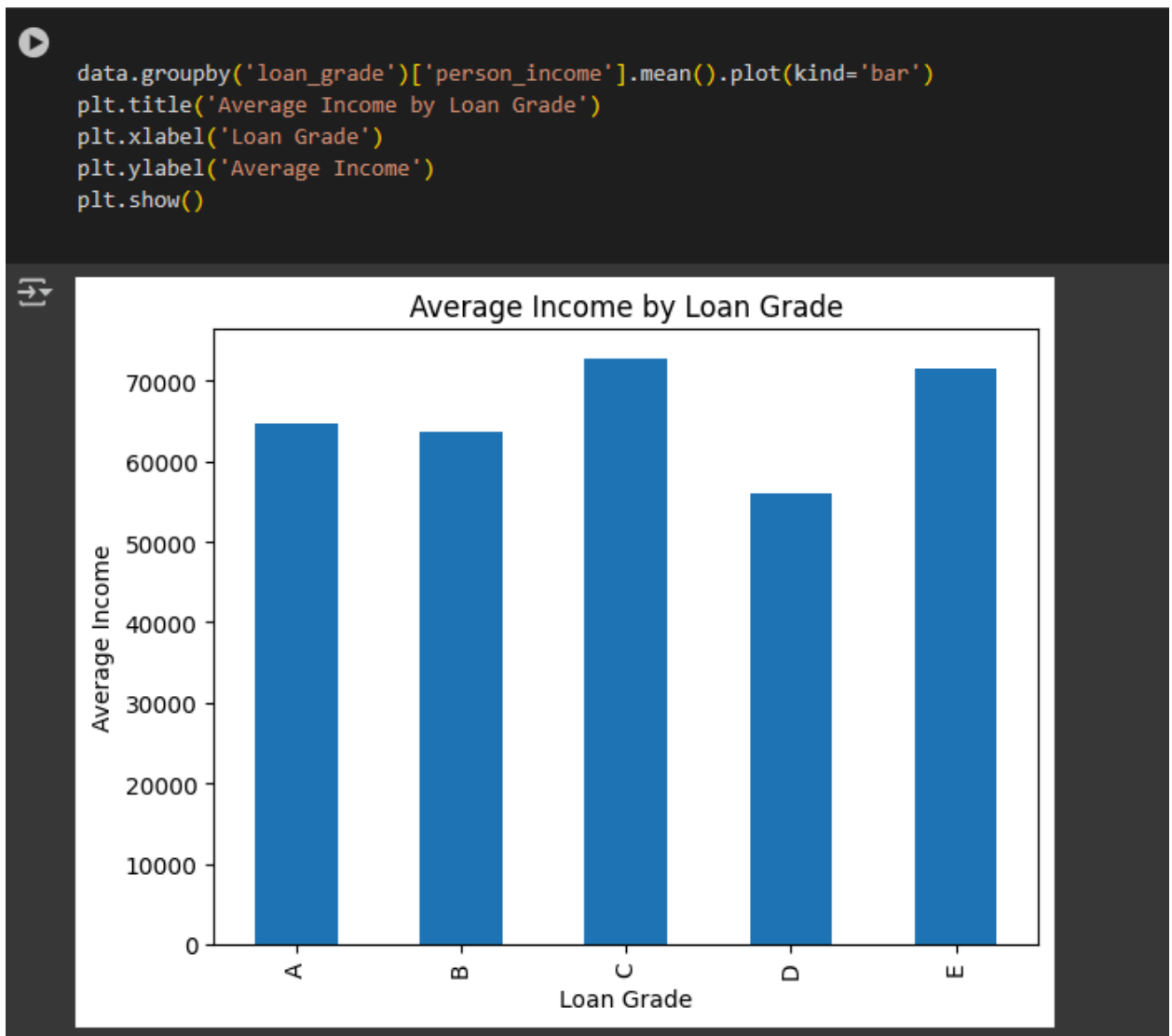
- Code Implementation
- Importing `Loan_train.csv` file from Github

## 1. Line Plot Graph



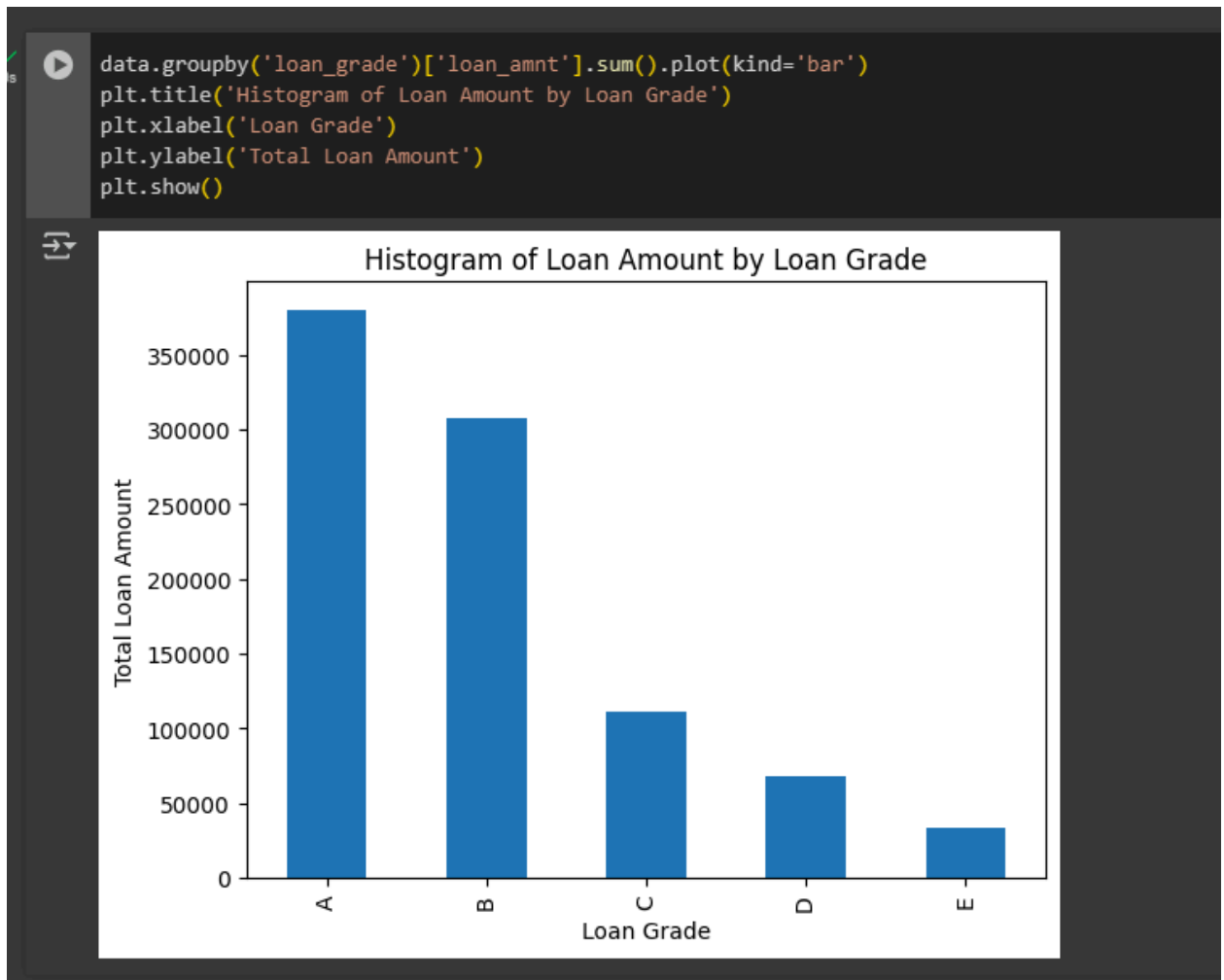
- **Load CSV:** Reads `Loan_train.csv` into a Pandas DataFrame.
- **Trim Data:** Selects the first 100 rows for analysis.
- **Plot Line Chart:** Plots `id` (x-axis) vs. `person_age` (y-axis).
- **Show Plot:** Displays the graph using Matplotlib.

## 2. Bar Chart



- **Group Data:** Groups by `loan_grade` and calculates the mean `person_income`.
- **Plot Bar Chart:** Displays average income for each loan grade.
- **Set Labels:** Adds title, x-axis, and y-axis labels.
- **Show Plot:** Displays the bar chart using Matplotlib.

### 3. Histogram



- **Group Data:** Sums `loan_amnt` for each `loan_grade`.
- **Plot Bar Chart:** Visualizes total loan amounts by grade.
- **Set Labels:** Adds title, x-axis, and y-axis labels.
- **Show Plot:** Displays the bar chart using Matplotlib.

#### 4. Multiple Bar Plots with Legends

```
import matplotlib.pyplot as plt
import numpy as np

# Sample data
categories = data['loan_grade'].head(5).tolist()
values1 = data['person_income'].head(5).tolist()
values2 = data['person_income'].iloc[5:10].tolist()

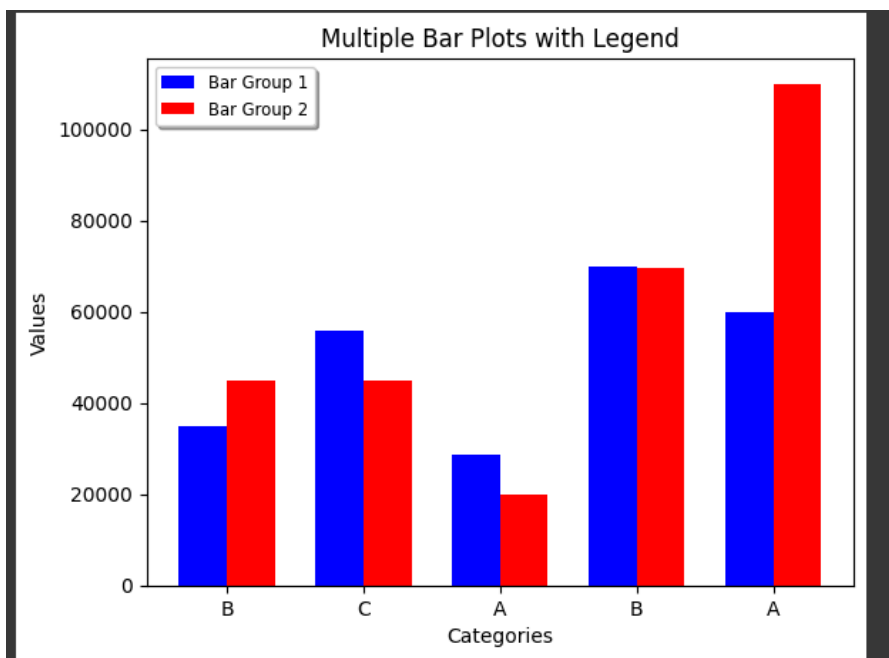
bar_width = 0.35
index = np.arange(len(categories))

# Plotting the bars
plt.bar(index, values1, bar_width, label='Bar Group 1', color='blue')
plt.bar(index + bar_width, values2[:len(index)], bar_width, label='Bar Group 2', color='red')

# Adding a title and labels
plt.title('Multiple Bar Plots with Legend')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.xticks(index + bar_width / 2, categories)

# Adding the legend
plt.legend(loc='best', fontsize='small', frameon=True, shadow=True)

# Display the plot
plt.show()
```



- **Prepare Data:** Extracts the first 5 `loan_grade` values and two income groups.
- **Set Bar Width:** Defines the width and position of bars using `numpy`.
- **Plot Bars:** Creates two grouped bar charts for comparison.
- **Customize Plot:** Adds title, labels, x-ticks, and a legend.
- **Show Plot:** Displays the grouped bar chart using Matplotlib.
-



## 5. Scatter Plots

```
# Sample data
x1 = data['person_age'].head(5).tolist()
y1 = data['person_income'].head(5).tolist()
x2 = data['person_age'].iloc[5:10].tolist()
y2 = data['person_income'].iloc[5:10].tolist()

# Scatter plot
plt.scatter(x1, y1, label='Group 1', color='blue', marker='o')
plt.scatter(x2, y2, label='Group 2', color='red', marker='x')

# Adding a title and labels
plt.title('Scatter Plot of Age vs Income')
plt.xlabel('Age')
plt.ylabel('Income')

# Adding the legend
plt.legend()

# Display the plot
plt.show()
```



- **Prepare Data:** Extracts `person_age` and `person_income` for two groups.
- **Plot Scatter Points:** Uses blue circles for Group 1 and red crosses for Group 2.
- **Customize Plot:** Adds title, x-axis, and y-axis labels.
- **Add Legend:** Differentiates the two groups.

## 6. Multiple Line Plots

```
import matplotlib.pyplot as plt

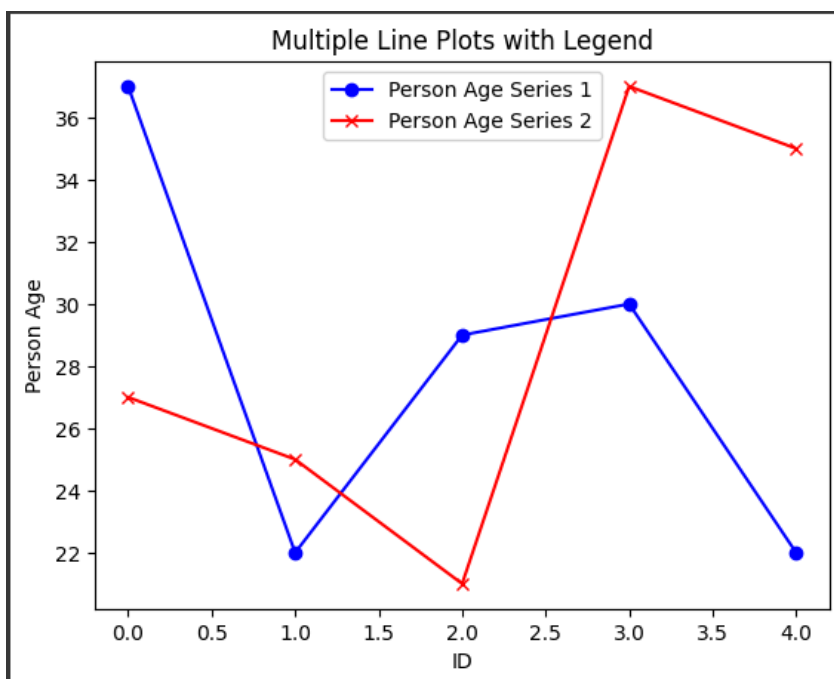
# Sample data
x = data['id'].head(5).tolist()
y1 = data['person_age'].head(5).tolist()
y2 = data['person_age'].iloc[5:10].tolist()

# Plotting the data
plt.plot(x, y1, label='Person Age Series 1', color='blue', marker='o')
plt.plot(x, y2, label='Person Age Series 2', color='red', marker='x')

# Adding a title and labels
plt.title('Multiple Line Plots with Legend')
plt.xlabel('ID')
plt.ylabel('Person Age')

# Adding the legend
plt.legend()

# Display the plot
plt.show()
```



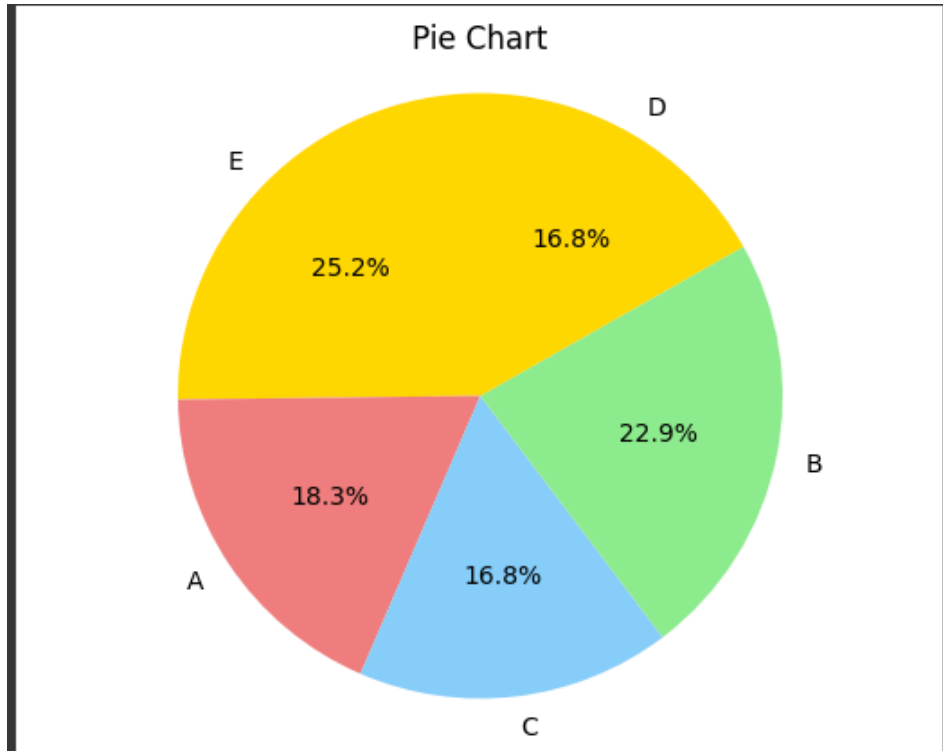
- **Prepare Data:** Extracts `id` and `person_age` for two series.
- **Plot Line Charts:** Plots two line graphs with different markers and colors.
- **Customize Plot:** Adds title, x-axis, and y-axis labels.
- **Add Legend:** Differentiates the two series.

## 7. Pie Chart

```
import matplotlib.pyplot as plt

# Data for the pie chart
sizes = data['person_age'].iloc[35:40].tolist() # Sizes of the pie slices
labels = data['loan_grade'].iloc[35:40].tolist() # Labels for the slices
colors = ['gold', 'lightcoral', 'lightskyblue', 'lightgreen'] # Colors for each slice

# Create a pie chart
plt.figure()
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90)
plt.axis('equal') # Equal aspect ratio ensures that pie chart is circular.
plt.title('Pie Chart')
plt.show()
```



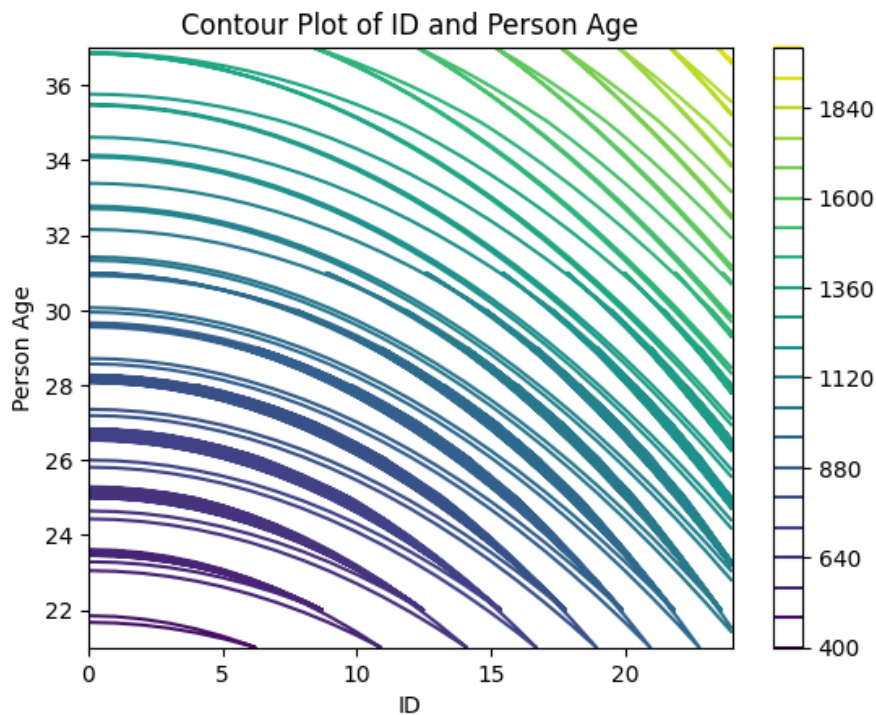
- **Prepare Data:** Extracts `person_age` and `loan_grade` for the pie chart slices.
- **Set Colors:** Defines custom colors for each slice.
- **Plot Pie Chart:** Creates a pie chart with percentage labels and a starting angle of 90°.
- **Customize Plot:** Ensures the pie chart is circular and adds a title.
- **Show Plot:** Displays the pie chart using Matplotlib.

## 8. Contour Plots

```
x = data['id'].head(25).tolist()
y = data['person_age'].head(25).tolist()
X, Y = np.meshgrid(x, y)

# Define a function for Z
Z = X**2 + Y**2

# Create a contour plot
plt.figure()
contour = plt.contour(X, Y, Z, levels=20, cmap='viridis')
plt.colorbar(contour)
plt.title('Contour Plot of ID and Person Age')
plt.xlabel('ID')
plt.ylabel('Person Age')
plt.show()
```



- **Prepare Data:** Extracts `id` and `person_age` for creating a mesh grid.
- **Define Function:** Calculates  $Z$  as the sum of squares of  $X$  and  $Y$ .
- **Plot Contour:** Creates a contour plot with 20 levels and applies the `viridis` colormap.
- **Customize Plot:** Adds color bar, title, and axis labels.

## 9. 3D Scatter Plots

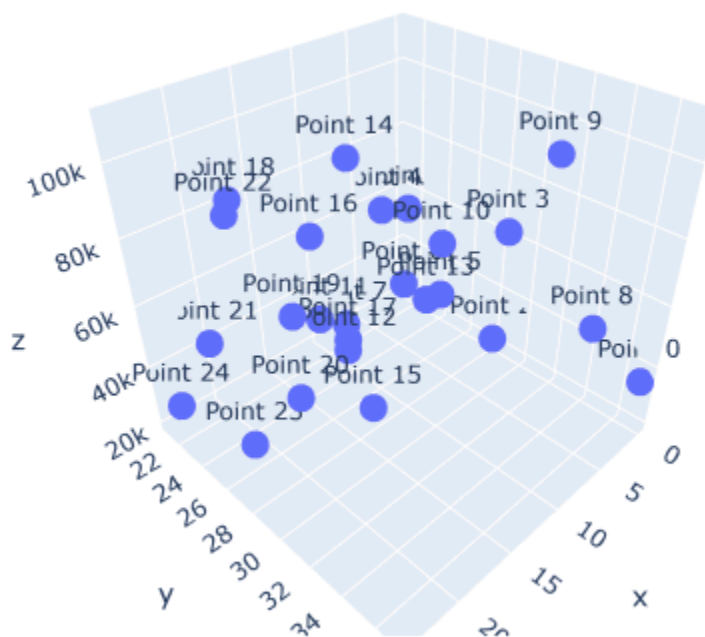
```
import plotly.express as px
import pandas as pd

# Load the CSV file (Assuming you have a CSV file named 'Loan_train.csv')
file_path = "Loan_train.csv"
loan_data = pd.read_csv(file_path) # Read the data into a DataFrame named loan_data

# Trim data into the first 100 rows
loan_data = loan_data.head(100)

df = pd.DataFrame({
    "x": loan_data['id'].head(25).tolist(),
    "y": loan_data['person_age'].head(25).tolist(),
    "z": loan_data['person_income'].head(25).tolist(),
    "label": [f'Point {i}' for i in range(25)]
})

# Create 3D Scatter Plot using plotly
fig = px.scatter_3d(df, x='x', y='y', z='z', text='label', title='3D Scatter Plot')
fig.show()
```



- **Prepare Data:** Loads and trims the first 100 rows from `Loan_train.csv`, then selects specific columns (`id`, `person_age`, `person_income`) for the 3D plot.
- **Create DataFrame:** Organizes data into `x`, `y`, `z` coordinates and labels each point.
- **Plot 3D Scatter:** Uses `plotly` to create an interactive 3D scatter plot with point labels.
- **Show Plot:** Displays the 3D plot with `fig.show()` from Plotly.