Name - Debangan Ghosh
Roll - AP22110010984

LAB-1

Code 1: Processing Employee Data

```
import pandas as pd
import numpy as np

# New sample input data for the Employee.txt file
data = """EmpName,EmpId,EmpDesig,EmpSalary
Alice,201,Director,120000
Bob,202,Developer,
Charlie,,Analyst,75000
,203,Intern,30000
,204,,45000"""

# Simulate reading from a file by using a StringIO object
from io import StringIO
employee_data = pd.read_csv(StringIO(data))

# Calculate the average salary (ignoring NaN values)
average_salary = employee_data['EmpSalary'].astype(float).mean()

# Fill missing values in the dataset
employee_data['EmpName'] = employee_data['EmpName'].fillna('Unknown')
employee_data['EmpId'] = employee_data['EmpId'].fillna('NotAssigned')
employee_data['EmpDesig'] = employee_data['EmpDesig'].fillna('NotAvailable')
employee_data['EmpSalary'] = employee_data['EmpSalary'].fillna(average_salary)

# Round the salary to the nearest integer
employee_data['EmpSalary'] = employee_data['EmpSalary'].astype(float).round(0)

# Display the processed data
print("\nUpdated Employee Data:")
print(employee_data)
```

Expected Output:

Updated Employee Data:

```
   EmpName        EmpId       EmpDesig  EmpSalary
0    Alice          201       Director   120000.0
1      Bob          202      Developer    67500.0
2  Charlie  NotAssigned       Analyst    75000.0
3  Unknown          203        Intern    30000.0
4  Unknown          204  NotAvailable    45000.0
```

Code 2: Equal-Frequency Binning and Smoothing

```
import numpy as np

# Function to divide data into bins of equal frequency
def perform_equal_frequency_binning(values, bin_count):
    sorted_values = np.sort(values)
    bin_size = len(values) // bin_count
    bins = []
    for start in range(0, len(sorted_values), bin_size):
        if len(bins) < bin_count:
            bin_content = sorted_values[start:start + bin_size]
            bins.append(bin_content)
    # Handle leftover values
    if len(bins) > 0 and len(sorted_values) % bin_count != 0:
        remaining = sorted_values[bin_count * bin_size:]
        bins[-1] = np.concatenate([bins[-1], remaining])
    return bins

# Function to smooth data in each bin using the mean
def apply_bin_mean_smoothing(bins):
    smoothed_bins = []
    for bin_data in bins:
        mean_value = np.mean(bin_data)
        smoothed_bins.append(np.full_like(bin_data, mean_value))
    return smoothed_bins

def process_data():
    # Input from the user
    product_count = int(input("Enter the number of items: "))
    print("Enter the prices of the items:")
    item_prices = list(map(float, input().split()))
    bin_count = int(input("Enter the desired number of bins: "))

    # Perform binning and smoothing
    binned_data = perform_equal_frequency_binning(item_prices, bin_count)
    smoothed_data = apply_bin_mean_smoothing(binned_data)
```

```
    # Output the results
    print("\nBinned Data:")
    for idx, group in enumerate(smoothed_data, 1):
        bin_mean = round(group[0], 2)
        bin_output = f"{bin_mean:.2f}, " * (len(group) - 1) + f"{bin_mean:.2f}"
        print(f"Bin {idx}: {bin_output}")

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

Sample Input:

Enter the number of items: 10
Enter the prices of the items:
80 120 160 200 240 280 320 360 400 440
Enter the desired number of bins: 5

Expected Output:

Binned Data:
Bin 1: 100.00, 100.00
Bin 2: 180.00, 180.00
Bin 3: 260.00, 260.00
Bin 4: 340.00, 340.00
Bin 5: 420.00, 420.00