

A decorative border composed of small ice cream cones with yellow and pink scoops and brown cones, arranged in a rectangular frame around the text.

Atma Ram Sanatan Dharma College, Delhi University

PRACTICAL FILE

By : KESHAV PAL, 38046

Subject = Data Analysis and Visualization

1. Write programs in Python using NumPy library to do the following:

- a. Create a two-dimensional array, ARR1 having random values from 0 to 1. Compute the mean, standard deviation, and variance of ARR1 along the second axis.

```
import numpy as np
```

```
data = np.random.random((3, 3))  
print(data.mean())  
print(data.var())  
print(data.std())
```

```
0.508700530143497  
0.10468072164871692  
0.32354400264680677
```

- b. Create a 2-dimensional array of size m x n integer elements, also print the shape, type and data type of the array and then reshape it into an n x m array, where n and m are user inputs given at the run time.

```
1 import numpy as np  
2  
3 data = np.random.random((3, 3))  
4 print(data.shape)  
5 print(data.dtype)  
6 print(type(data))
```

```
(3, 3)  
float64  
<class 'numpy.ndarray'>
```

- c. Test whether the elements of a given 1D array are zero, non-zero and NaN. Record the indices of these elements in three separate arrays.



```
1 import numpy as np
2
3
4 arr = np.array([np.nan, 0, 34, 23, 9, 0, np.nan])
5
6 null = []
7 non_null = []
8 zeroes = []
9
10 print(list(np.where(arr==0)[0]))
11 print(list(np.where(arr!=0)[0]))
12 print(list(np.where(np.isnan(arr))[0]))
```

```
[1, 5]
[0, 2, 3, 4, 6]
[0, 6]
```

- d. Create three random arrays of the same size: Array1, Array2 and Array3. Subtract Array 2 from Array3 and store in Array4. Create another array Array5 having two times the values in Array1. Find Covariance and Correlation of Array1 with Array4 and Array5 respectively.



```
1 import numpy as np
2
3 arr1 = np.random.random((4))
4 arr2 = np.random.random((4))
5 arr3 = np.random.random((4))
6
7 arr4 = arr3 - arr2
8 arr5 = 2*arr1
9
10 print(np.cov(arr1, arr4))
11 print(np.corrcoef(arr1, arr5))
```

```
[[0.0743075  0.02299131]
 [0.02299131 0.06583582]]
[[1.  1.]
 [1.  1.]]
```

- e. Create two random arrays of the same size 10: Array1, and Array2. Find the sum of the first half of both the arrays and product of the second half of both the arrays.

```

1 import numpy as np
2
3 arr1 = np.random.random(10)
4 arr2 = np.random.random(10)
5 print(arr1[:10//2]+arr2[:10//2])
6 print(arr1[10//2:]*arr2[10//2:])

```

```

[0.31664414 0.61745974 0.58128857 1.61628902 1.14172332]
[0.13147597 0.44284835 0.16847581 0.01150051 0.02681162]

```

- f. Create an array with random values. Determine the size of the memory occupied by the array.

```

1 import numpy as np
2
3 arr = np.random.random(5)
4 print(5*arr.itemsize)

```

```

40

```

- g. Create a 2-dimensional array of size m x n having integer elements in the range (10,100). Write statements to swap any two rows, reverse a specified column and store updated array in another variable.

```

1 import numpy as np
2
3 def swap_row(arr, row_num1:int, row_num2:int) :
4     arr[[row_num1, row_num2]] = arr[[row_num2, row_num1]]
5     return arr
6
7 def rev_column(arr, column_num:int) :
8     arr[:, column_num] = np.array(list(reversed(arr[:,column_num])))
9     return arr
10
11 m = 9
12 n = 10
13 arr = np.array([x for x in range(10, 100)]).reshape(m, n)
14 print(swap_row(arr, 1, 7))
15 print(rev_column(arr, 5))

```

```

[[10 11 12 13 14 15 16 17 18 19]
 [80 81 82 83 84 85 86 87 88 89]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [20 21 22 23 24 25 26 27 28 29]
 [90 91 92 93 94 95 96 97 98 99]]
[[10 11 12 13 14 95 16 17 18 19]
 [80 81 82 83 84 25 86 87 88 89]
 [30 31 32 33 34 75 36 37 38 39]
 [40 41 42 43 44 65 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 45 66 67 68 69]
 [70 71 72 73 74 35 76 77 78 79]
 [20 21 22 23 24 85 26 27 28 29]
 [90 91 92 93 94 15 96 97 98 99]]

```

2a. Create a series with 5 elements. Display the series sorted on index and also sorted on values separately.

```

1 import pandas as pd
2
3 arr = [5, 6, 2, 4, 3]
4 ind = ["Del", "Bom", "Kol", "Mum", "Ben"]
5
6 df = pd.Series(arr, index=ind)
7 print(df.sort_index())
8 print(df.sort_values())

```

Ben	3
Bom	6
Del	5
Kol	2
Mum	4
dtype:	int64
Kol	2
Ben	3
Mum	4
Del	5
Bom	6
dtype:	int64

2b. Create a series with N elements with some duplicate values. Find the minimum and maximum ranks assigned to the values using 'first' and 'max' methods.

```

1 import pandas as pd, numpy as np
2
3
4 n = 10
5 arr = [np.random.randint(1, 5) for _ in range(n)]
6 arr = pd.Series(arr)
7 print(arr.rank(method="first"))
8 print(arr.rank(method="max"))

```

```

0      3.0
1      1.0
2      4.0
3      6.0
4      2.0
5      7.0
6      5.0
7      9.0
8     10.0
9      8.0
dtype: float64

```

```

0      5.0
1      2.0
2      5.0
3      8.0
4      2.0
5      8.0
6      5.0
7     10.0
8     10.0
9      8.0
dtype: float64

```

2c. Display the index value of the minimum and maximum element of a Series.

```

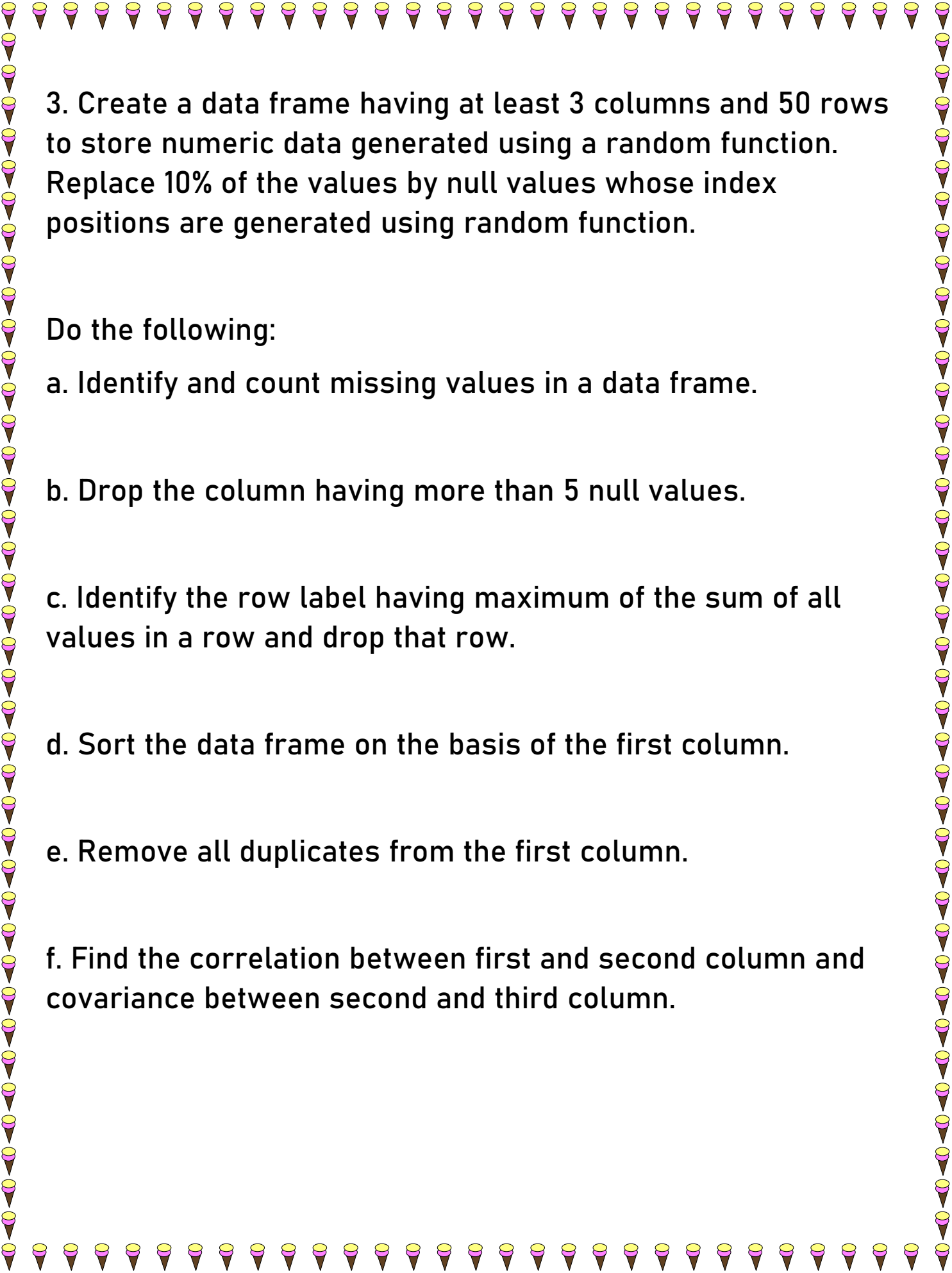
1 import pandas as pd, numpy as np
2
3 n = 10
4 arr = [np.random.randint(1, 5000) for _ in range(10)]
5 arr = pd.Series(arr)
6 print(arr)
7 print(arr.idxmax())
8 print(arr.idxmin())

```

```

0      4475
1      2573
2         90
3      4846
4      4415
5      3459
6      4359
7      3323
8      2560
9      4610
dtype: int64
3
2

```



3. Create a data frame having at least 3 columns and 50 rows to store numeric data generated using a random function. Replace 10% of the values by null values whose index positions are generated using random function.

Do the following:

- a. Identify and count missing values in a data frame.
- b. Drop the column having more than 5 null values.
- c. Identify the row label having maximum of the sum of all values in a row and drop that row.
- d. Sort the data frame on the basis of the first column.
- e. Remove all duplicates from the first column.
- f. Find the correlation between first and second column and covariance between second and third column.

g. Discretize the second column and create 5 bins.

```
1 import pandas as pd
2 import numpy as np
3
4 # Step 1: Create a DataFrame with 50 rows and 3 columns of random numeric data
5 df = pd.DataFrame(np.random.rand(50, 3))
6
7 # Step 2: Randomly assign NaN values in 5 locations
8 for i in range(5):
9     x, y = np.random.randint(0, 50), np.random.randint(0, 3)
10    df.iloc[x, y] = pd.NA
11    print(f"NaN inserted at position ({x}, {y})")
12
13 # a. Identify and count missing values in a data frame
14 print("\n=== Step a ===")
15 print("Total missing values in the DataFrame:", df.isnull().sum().sum())
16 print("Missing values per column:\n", df.isnull().sum())
17
18 # b. Drop the column having more than 5 null values
19 print("\n=== Step b ===")
20 df_cleaned = df.dropna(thresh=len(df)-5, axis=1)
21 print("DataFrame after dropping columns with more than 5 NaN values:\n", df_cleaned)
22
23 # c. Identify the row label having maximum sum and drop that row
24 print("\n=== Step c ===")
25 max_sum_row = df_cleaned.sum(axis=1).idxmax() # Find row with the maximum sum
26 df_cleaned = df_cleaned.drop(max_sum_row)
27 print(f"DataFrame after dropping the row with the highest sum (Row {max_sum_row}):\n", df_cleaned)
28
29 # d. Sort the DataFrame on the basis of the first column (column 0)
30 print("\n=== Step d ===")
31 df_sorted = df_cleaned.sort_values(by=0)
32 print("DataFrame sorted by the first column (0):\n", df_sorted)
33
34 # e. Remove all duplicates from the first column
35 print("\n=== Step e ===")
36 df_unique = df_sorted.drop_duplicates(subset=0)
37 print("DataFrame after removing duplicates from the first column:\n", df_unique)
38
39 # f. Find the correlation between the first and second column and the covariance between second and third column
40 print("\n=== Step f ===")
41 correlation_0_1 = df_unique[0].corr(df_unique[1])
42 covariance_1_2 = df_unique[1].cov(df_unique[2])
43 print(f"Correlation between column 0 and 1: {correlation_0_1}")
44 print(f"Covariance between column 1 and 2: {covariance_1_2}")
45
46 # g. Discretize the second column and create 5 bins
47 print("\n=== Step g ===")
48 df_unique['1_bins'] = pd.cut(df_unique[1], bins=5)
49 print("DataFrame with second column discretized into 5 bins:\n", df_unique[['1', '1_bins']])
```

```
=== Step a ===
Total missing values in the DataFrame: 5
Missing values per column:
   0    1
1   1
2   3
dtype: int64
```


=== Step b ===

DataFrame after dropping columns with more than 5 NaN values:

	0	1	2
0	0.606343	0.437721	0.506426
1	0.724283	0.272135	0.851721
2	0.507298	0.197020	0.714557
3	0.741114	0.766793	0.009092
4	NaN	NaN	0.779128
5	0.841222	0.862341	0.421909
6	0.488868	0.162804	0.093809
7	0.916533	0.117800	0.333735
8	0.003147	0.300955	0.159704
9	0.881547	0.315183	0.537133
10	0.150085	0.766748	0.560416
11	0.218139	0.878493	NaN
12	0.229250	0.370456	0.694175
13	0.034148	0.408599	0.363868
14	0.824857	0.541242	0.777410
15	0.672641	0.508239	0.546477
16	0.583560	0.518940	0.444462
17	0.929532	0.713665	0.759410
18	0.747207	0.971862	0.162871
19	0.828837	0.287827	0.368200
20	0.778087	0.406916	0.993291
21	0.420774	0.875328	0.016342
22	0.578103	0.465460	0.099945
23	0.666834	0.531853	0.228951
24	0.370956	0.668974	0.649178
25	0.180062	0.628518	0.986265
26	0.407937	0.792560	0.587410
27	0.887565	0.026310	0.552025
28	0.841912	0.075434	0.851402
29	0.305488	0.445185	0.516906
30	0.809684	0.922027	NaN

=== Step c ===

DataFrame after dropping the row with the highest sum (Row 43):

	0	1	2
0	0.606343	0.437721	0.506426
1	0.724283	0.272135	0.851721
2	0.507298	0.197020	0.714557
3	0.741114	0.766793	0.009092
4	NaN	NaN	0.779128
5	0.841222	0.862341	0.421909
6	0.488868	0.162804	0.093809
7	0.916533	0.117800	0.333735
8	0.003147	0.300955	0.159704
9	0.881547	0.315183	0.537133
10	0.150085	0.766748	0.560416

=== Step d ===

DataFrame sorted by the first column (0):

	0	1	2
8	0.003147	0.300955	0.159704
13	0.034148	0.408599	0.363868
34	0.135532	0.420598	0.844575
10	0.150085	0.766748	0.560416
36	0.159859	0.277840	NaN
25	0.180062	0.628518	0.986265
49	0.192455	0.689707	0.941994
11	0.218139	0.878493	NaN
12	0.229250	0.370456	0.694175
47	0.237345	0.900883	0.983760
46	0.267144	0.391111	0.789550
29	0.305488	0.445185	0.516906
39	0.364996	0.807026	0.828575
24	0.370956	0.668974	0.649178
42	0.405441	0.907892	0.325024
26	0.407937	0.792560	0.587410
21	0.420774	0.875328	0.016342
32	0.483426	0.511725	0.512629
6	0.488868	0.162804	0.093809
33	0.493456	0.621875	0.495075
2	0.507298	0.197020	0.714557
22	0.578103	0.465460	0.099945
16	0.583560	0.518940	0.444462

=== Step e ===

DataFrame after removing duplicates from the first column:

	0	1	2
8	0.003147	0.300955	0.159704
13	0.034148	0.408599	0.363868
34	0.135532	0.420598	0.844575
10	0.150085	0.766748	0.560416
36	0.159859	0.277840	NaN
25	0.180062	0.628518	0.986265
49	0.192455	0.689707	0.941994
11	0.218139	0.878493	NaN
12	0.229250	0.370456	0.694175
47	0.237345	0.900883	0.983760
46	0.267144	0.391111	0.789550
29	0.305488	0.445185	0.516906
39	0.364996	0.807026	0.828575
24	0.370956	0.668974	0.649178

```

=== Step f ===
Correlation between column 0 and 1: -0.21247300696045984
Covariance between column 1 and 2: 0.0009468562897036913

=== Step g ===
DataFrame with second column discretized into 5 bins:

```

	1	1_bins
8	0.300955	(0.202, 0.395]
13	0.408599	(0.395, 0.587]
34	0.420598	(0.395, 0.587]
10	0.766748	(0.587, 0.779]
36	0.277840	(0.202, 0.395]
25	0.628518	(0.587, 0.779]
49	0.689707	(0.587, 0.779]
11	0.878493	(0.779, 0.972]
12	0.370456	(0.202, 0.395]
47	0.900883	(0.779, 0.972]
46	0.391111	(0.202, 0.395]
29	0.445185	(0.395, 0.587]
39	0.807026	(0.779, 0.972]
24	0.668974	(0.587, 0.779]
42	0.907892	(0.779, 0.972]
26	0.792560	(0.779, 0.972]
21	0.875328	(0.779, 0.972]
32	0.511725	(0.395, 0.587]
6	0.162804	(0.0089, 0.202]
33	0.621875	(0.587, 0.779]

4. Consider two excel files having attendance of two workshops, each of duration 5 days. Each file has three fields 'Name', 'Date, duration (in minutes) where names may be repetitive within a file. Note that duration may take one of three values (30, 40, 50) only. Import the data into two data frames and do the following:

- Perform merging of the two data frames to find the names of students who had attended both workshops.
- Find names of all students who have attended a single workshop only.

c. Merge two data frames row-wise and find the total number of records in the data frame.

d. Merge two data frames row-wise and use two columns viz. names and dates as multi-row indexes. Generate descriptive statistics for this hierarchical data frame.

```
1 import pandas as pd
2
3 # Step 1: Import data from two Excel files into two DataFrames
4 df1 = pd.read_excel('workshop1.xlsx')
5 df2 = pd.read_excel('workshop2.xlsx')
6
7
8 # a. Merging of the two data frames to find the names of students who had attended both workshops
9 print("\n=== Step a ===")
10 common_students = pd.merge(df1[['Name']], df2[['Name']], on='Name', how='inner').drop_duplicates()
11 print("Students who attended both workshops:\n", common_students)
12
13 # b. Find names of all students who have attended a single workshop only
14 print("\n=== Step b ===")
15 students_only1 = pd.concat([df1[['Name']], df2[['Name']]]).drop_duplicates(keep=False)
16 print("Students who attended only one workshop:\n", students_only1)
17
18 # c. Merge two data frames row-wise and find the total number of records in the data frame
19 print("\n=== Step c ===")
20 df_rowwise = pd.concat([df1, df2], axis=0)
21 total_records = len(df_rowwise)
22 print("Total number of records after row-wise merge:", total_records)
23
24 # d. Merge two data frames row-wise and use two columns viz. names and dates as multi-row indexes.
25 # Generate descriptive statistics for this hierarchical data frame.
26 print("\n=== Step d ===")
27 df_hierarchical = df_rowwise.set_index(['Name', 'Date'])
28 print("Hierarchical DataFrame (Name, Date as multi-index):\n", df_hierarchical.head())
29
30 # Generating descriptive statistics for the hierarchical data frame
31 statistics = df_hierarchical.describe()
32 print("\nDescriptive statistics for the hierarchical data frame:\n", statistics)
33
```



5. Using Iris data, plot the following with proper legend and axis labels:

- a. Load data into pandas' data frame. Use `pandas.info ()` method to look at the info on datatypes in the dataset.
- b. Find the number of missing values in each column (Check number of null values in a column using `df.isnull().sum()`).
- c. Plot bar chart to show the frequency of each class label in the data.
- d. Draw a scatter plot for Petal Length vs Sepal Length and fit a regression line.
- e. Plot density distribution for feature Petal width.
- f. Use a pair plot to show pairwise bivariate distribution in the Iris Dataset.
- g. Draw heatmap for any two numeric attributes.
- h. Compute mean, mode, median, standard deviation, confidence interval and standard error for each numeric feature.
- i. Compute correlation coefficients between each pair of features and plot heatmap.

```

1 import pandas as pd
2 import numpy as np
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5 from sklearn import datasets
6
7 # a. Load the Iris data into a pandas DataFrame
8 iris = datasets.load_iris()
9 df = pd.DataFrame(data=iris['data'], columns=iris['feature_names'])
10 df['target'] = iris['target']
11 df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)
12
13 # Check the data
14 print(df.head())
15
16 # Use pandas.info() to check data types
17 print("\n=== Step a ===")
18 df.info()
19
20 # b. Find the number of missing values in each column
21 print("\n=== Step b ===")
22 missing_values = df.isnull().sum()
23 print("Missing values in each column:\n", missing_values)
24
25 # c. Plot bar chart to show the frequency of each class label in the data
26 print("\n=== Step c ===")
27 plt.figure(figsize=(8, 6))
28 sns.countplot(x='species', data=df)
29 plt.title('Frequency of Each Class Label (Species)')
30 plt.xlabel('Species')
31 plt.ylabel('Frequency')
32 plt.show()
33
34 # d. Scatter plot for Petal Length vs Sepal Length and fit a regression line
35 print("\n=== Step d ===")
36 plt.figure(figsize=(8, 6))
37 sns.regplot(x=df['sepal length (cm)'], y=df['petal length (cm)'])
38 plt.title('Scatter Plot: Petal Length vs Sepal Length')
39 plt.xlabel('Sepal Length (cm)')
40 plt.ylabel('Petal Length (cm)')
41 plt.show()
42
43 # e. Plot density distribution for Petal width
44 print("\n=== Step e ===")
45 plt.figure(figsize=(8, 6))
46 sns.kdeplot(df['petal width (cm)'], shade=True)
47 plt.title('Density Distribution for Petal Width')
48 plt.xlabel('Petal Width (cm)')
49 plt.show()
50
51 # f. Use a pair plot to show pairwise bivariate distribution in the Iris Dataset
52 print("\n=== Step f ===")
53 sns.pairplot(df, hue='species', markers=["o", "s", "D"])
54 plt.show()
55
56 # g. Draw heatmap for any two numeric attributes
57 print("\n=== Step g ===")
58 plt.figure(figsize=(8, 6))
59 corr_matrix = df[['sepal length (cm)', 'petal length (cm)']].corr()
60 sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
61 plt.title('Heatmap for Sepal Length and Petal Length')
62 plt.show()
63
64 # h. Compute mean, mode, median, standard deviation, standard error for each numeric feature
65 print("\n=== Step h ===")
66
67 # Compute statistics for each column
68 numeric_cols = df.select_dtypes(include=[np.number]).columns[:-1]
69
70 for col in numeric_cols:
71     mean = df[col].mean()
72     mode = df[col].mode()[0]
73     median = df[col].median()
74     std_dev = df[col].std()
75     sem = df[col].sem() # Standard error of the mean
76
77     print(f"Statistics for {col}:")
78     print(f"Mean: {mean}")
79     print(f"Mode: {mode}")
80     print(f"Median: {median}")
81     print(f"Standard Deviation: {std_dev}")
82     print(f"Standard Error: {sem}")
83     print()
84
85 # i. Compute correlation coefficients between each pair of features and plot heatmap
86 print("\n=== Step i ===")
87 plt.figure(figsize=(10, 8))
88 correlation_matrix = df.iloc[:, :4].corr()
89 sns.heatmap(correlation_matrix, annot=True, cmap='viridis')
90 plt.title('Correlation Heatmap of Features')
91 plt.show()

```




6. Using Titanic dataset, to do the following:

a. Clean the data by dropping the column which has the largest number of missing values.

b. Find total number of passengers with age more than 30

c. Find total fare paid by passengers of second class

d. Compare number of survivors of each passenger class

e. Compute descriptive statistics for age attribute gender wise

f. Draw a scatter plot for passenger fare paid by Female and Male passengers separately.

g. Compare density distribution for features age and passenger fare

h. Draw the pie chart for three groups labelled as class 1, class 2, class 3 respectively displayed in different colors. The occurrence of each group converted into percentage should be displayed in the pie chart. Appropriately label the chart.

i. Find % of survived passengers for each class and answer the question "Did class play a role in survival?".

```

1 import pandas as pd
2 import numpy as np
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5
6 # Load Titanic dataset
7 df = sns.load_dataset('titanic')
8
9 # a. Clean the data by dropping the column which has the largest number of missing values
10 print("\n=== Step a ===")
11 missing_vals = df.isnull().sum()
12 print("Missing values in each column:\n", missing_vals)
13
14 # Drop the column with the most missing values
15 column_to_drop = missing_vals.idxmax()
16 df_cleaned = df.drop(columns=[column_to_drop])
17 print(f"Column dropped: {column_to_drop}")
18 print(df_cleaned.head())
19
20 # b. Find total number of passengers with age more than 30
21 print("\n=== Step b ===")
22 passengers_above_30 = df[df['age'] > 30].shape[0]
23 print(f"Total passengers with age more than 30: {passengers_above_30}")
24
25 # c. Find total fare paid by passengers of second class
26 print("\n=== Step c ===")
27 total_fare_second_class = df[df['pclass'] == 2]['fare'].sum()
28 print(f"Total fare paid by passengers of second class: {total_fare_second_class}")
29
30 # d. Compare number of survivors of each passenger class
31 print("\n=== Step d ===")
32 survivors_by_class = df.groupby('pclass')['survived'].sum()
33 print("Number of survivors by passenger class:\n", survivors_by_class)
34
35 # e. Compute descriptive statistics for age attribute gender wise
36 print("\n=== Step e ===")
37 age_gender_stats = df.groupby('sex')['age'].describe()
38 print("Descriptive statistics for age gender wise:\n", age_gender_stats)
39
40 # f. Draw a scatter plot for passenger fare paid by Female and Male passengers separately
41 print("\n=== Step f ===")
42 plt.figure(figsize=(8, 6))
43 sns.scatterplot(x='fare', y='age', hue='sex', data=df)
44 plt.title('Scatter Plot: Fare Paid by Female and Male Passengers')
45 plt.xlabel('Fare Paid')
46 plt.ylabel('Age')
47 plt.show()
48
49 # g. Compare density distribution for features age and passenger fare
50 print("\n=== Step g ===")
51 plt.figure(figsize=(8, 6))
52 sns.kdeplot(df['age'].dropna(), label='Age', shade=True)
53 sns.kdeplot(df['fare'].dropna(), label='Fare', shade=True)
54 plt.title('Density Distribution for Age and Fare')
55 plt.legend()
56 plt.show()
57
58 # h. Draw a pie chart for three groups labeled as class 1, class 2, class 3 respectively
59 print("\n=== Step h ===")
60 pclass_counts = df['pclass'].value_counts()
61 pclass_percentage = pclass_counts / pclass_counts.sum() * 100
62
63 plt.figure(figsize=(8, 6))
64 plt.pie(pclass_percentage, labels=['Class 1', 'Class 2', 'Class 3'], autopct='%1.1f%%', colors=['#ff9999', '#66b3ff', '#99ff99'])
65 plt.title('Passenger Class Distribution')
66 plt.show()
67
68 # i. Find % of survived passengers for each class and answer the question "Did class play a role in survival?"
69 print("\n=== Step i ===")
70 survival_rate_by_class = df.groupby('pclass')['survived'].mean() * 100
71 print(f"Percentage of survived passengers by class:\n{survival_rate_by_class}")
72
73 # Did class play a role in survival?
74 if survival_rate_by_class[1] > survival_rate_by_class[3]:
75     print("\nYes, class played a role in survival. Higher class passengers had a higher survival rate.")
76 else:
77     print("\nNo, class did not play a significant role in survival.")

```


7. Consider the following data frame containing a family name, gender of the family member and her/his monthly income in each record.
- Calculate and display familywise gross monthly income.
 - Display the highest and lowest monthly income for each family name.
 - Calculate and display monthly income of all members earning income less than Rs. 80000.00.
 - Display total number of females along with their average monthly income.
 - Delete rows with Monthly income less than the average income of all members.

```
1 import pandas as pd
2
3 # Create the data frame
4 data = {
5     'FamilyName': ['Shah', 'Vats', 'Vats', 'Kumar', 'Vats', 'Kumar', 'Shah', 'Shah', 'Kumar', 'Vats'],
6     'Gender': ['Male', 'Male', 'Female', 'Female', 'Female', 'Male', 'Male', 'Female', 'Female', 'Male'],
7     'MonthlyIncome': [44000.00, 65000.00, 43150.00, 66500.00, 255000.00, 103000.00, 55000.00, 112400.00, 81030.00, 71900.00]
8 }
9
10 df = pd.DataFrame(data)
11
12 # a. Calculate and display familywise gross monthly income
13 print("\n=== Step a: Familywise Gross Monthly Income ===")
14 familywise_income = df.groupby('FamilyName')['MonthlyIncome'].sum()
15 print(familywise_income)
16
17 # b. Display the highest and lowest monthly income for each family name
18 print("\n=== Step b: Highest and Lowest Monthly Income for Each Family ===")
19 income_stats = df.groupby('FamilyName')['MonthlyIncome'].agg(['max', 'min'])
20 print(income_stats)
21
22 # c. Calculate and display monthly income of all members earning income less than Rs. 80000.00
23 print("\n=== Step c: Members Earning Less than Rs. 80000.00 ===")
24 low_income_members = df[df['MonthlyIncome'] < 80000]
25 print(low_income_members)
26
27 # d. Display total number of females along with their average monthly income
28 print("\n=== Step d: Total Number of Females and Their Average Monthly Income ===")
29 females = df[df['Gender'] == 'Female']
30 total_females = females.shape[0]
31 avg_income_females = females['MonthlyIncome'].mean()
32 print(f"Total number of females: {total_females}")
33 print(f"Average monthly income of females: Rs. {avg_income_females:.2f}")
34
35 # e. Delete rows with Monthly income less than the average income of all members
36 print("\n=== Step e: Rows with Monthly Income Less Than the Average Income ===")
37 average_income = df['MonthlyIncome'].mean()
38 df_above_avg = df[df['MonthlyIncome'] >= average_income]
39 print(f"Average income of all members: Rs. {average_income:.2f}")
40 print(df_above_avg)
41
```

A decorative border of ice cream cones surrounds the text. The border is composed of 100 cones, each with a yellow scoop, a pink swirl, and a brown cone, arranged in a rectangular frame.

THE END