**INDEX**

|  |  |  |
| --- | --- | --- |
| S. NO. | PROLOGUE | PAGE |
| 1. | **Q1:** Given Below Is A Dictionary Having Two Keys ‘Boys’  And ‘Girls’ And Having Two Lists Of Heights Of Five Boys And Five Girls Respectively As Values Associated With These Keys  Original Dictionary Of Lists: {'Boys': [72, 68, 70, 69, 74],  'Girls': [63, 65, 69, 62, 61]}  From The Given Dictionary Of Lists Create The Following List Of Dictionaries: [{'Boys': 72, 'Girls': 63}, {'Boys': 68,  'Girls': 65}, {'Boys': 70, 'Girls': 69}, {'Boys': 69, 'Girls': 62},  {‘Boys’:74,  ‘Girls’:61]. | 5 |
| 2. | **Q2:** WRITE PROGRAMS IN PYTHON USING NUMPY  LIBRARY TO DO THE FOLLOWING:-   * Compute the mean, standard deviation, and variance of a two dimensional random integer array along the second axis * Get the indices of the sorted array * Create a 2-dimensional array of size m x n integer elements * Test whether the elements of a given array are   zero, non-zero and NaN | 6 |
| 3. | **Q3:** CREATE A DATAFRAME HAVING AT LEAST 3 COLUMNS  AND 50 ROWS TO STORE NUMERIC DATA GENERATED USING A RANDOM FUNCTION   * Identify and count missing values * Drop the column having more than 5 null values * Identify the row label having maximum of the sum * Sort the dataframe on the basis of the first column * Remove all duplicates from the first column * Find correaltion and covariance | 8 |

|  |  |  |
| --- | --- | --- |
| 4. | **Q4:** CONSIDER TWO EXCEL FILES HAVING ATTENDANCE OF A WORKSHOP’S PARTICIPANTS FOR TWO DAYS   * Perform merging of the two dataframes * Find names of all students who have attended workshop on either of the days * Merge two data frames row-wise * Merge two data frames and use two columns names and duration as multi-row indexes | 18 |
| 5. | **Q5:** TAKING IRIS DATA, PLOT THE FOLLOWING WITH PROPER LEGEND AND AXIS LABELS   * Plot bar chart to show the frequency of each class label in the data * Draw a scatter plot for Petal width vs sepal width * Plot density distribution for feature petal length * Use a pair plot to show pairwise bivariate distribution | 21 |
| 6. | **Q6:** CONSIDER ANY SALES TRAINING/ WEATHER  FORECASTING DATASET   * Compute mean of a series grouped by another series * Fill an intermittent time series to replace all missing dates * Perform appropriate year-month string to dates conversion. * Split a dataset to group by two columns and then sort * Split a given dataframe into groups with bin counts. | 24 |

|  |  |  |
| --- | --- | --- |
| 7. | **Q7:** CONSIDER A DATA FRAME CONTAINING DATA ABOUT STUDENTS   * Perform one hot encoding of the last two columns of categorical data using the get\_dummies() function. * Sort this data frame on the “Birth Month” column | 31 |
| 8. | **Q8:** CONSIDER THE FOLLOWING DATA FRAME   * Calculate and display familywise gross monthly income * Calculate and display the member with the highest monthly income in a family * Calculate and display monthly income of all members with income greater than Rs. 60000.00 * Calculate and display the average monthly income of the female members in the Shah family. | 34 |

Q1 http://localhost:8889/nbconvert/html/Q1.ipynb?download=false

# Q1. Given below is a dictionary having two keys ‘Boys’ and ‘Girls’ and having two lists of heights of five Boys and Five Girls respectively as values associated with these keys

Original dictionary of lists:

# {'Boys': [72, 68, 70, 69, 74], 'Girls': [63, 65,

69, 62, 61]}

# From the given dictionary of lists create the following list of dictionaries:

[{'Boys': 72, 'Girls': 63}, {'Boys': 68, 'Girls':

# 65}, {'Boys': 70, 'Girls': 69}, {'Boys': 69,

'Girls': 62}, {‘Boys’:74, ‘Girls’:61]

In [1]:

Dict **=** {'Boys':[72,68,70,69,74], 'Girls':[63,65,69,62,61]}

keys **=** list(Dict**.**keys())

data1 **=** Dict[keys[0]] data2 **=** Dict[keys[1]]

j **=** len(data1) res **=** []

**for** i **in** range(j):

t1 **=** {}

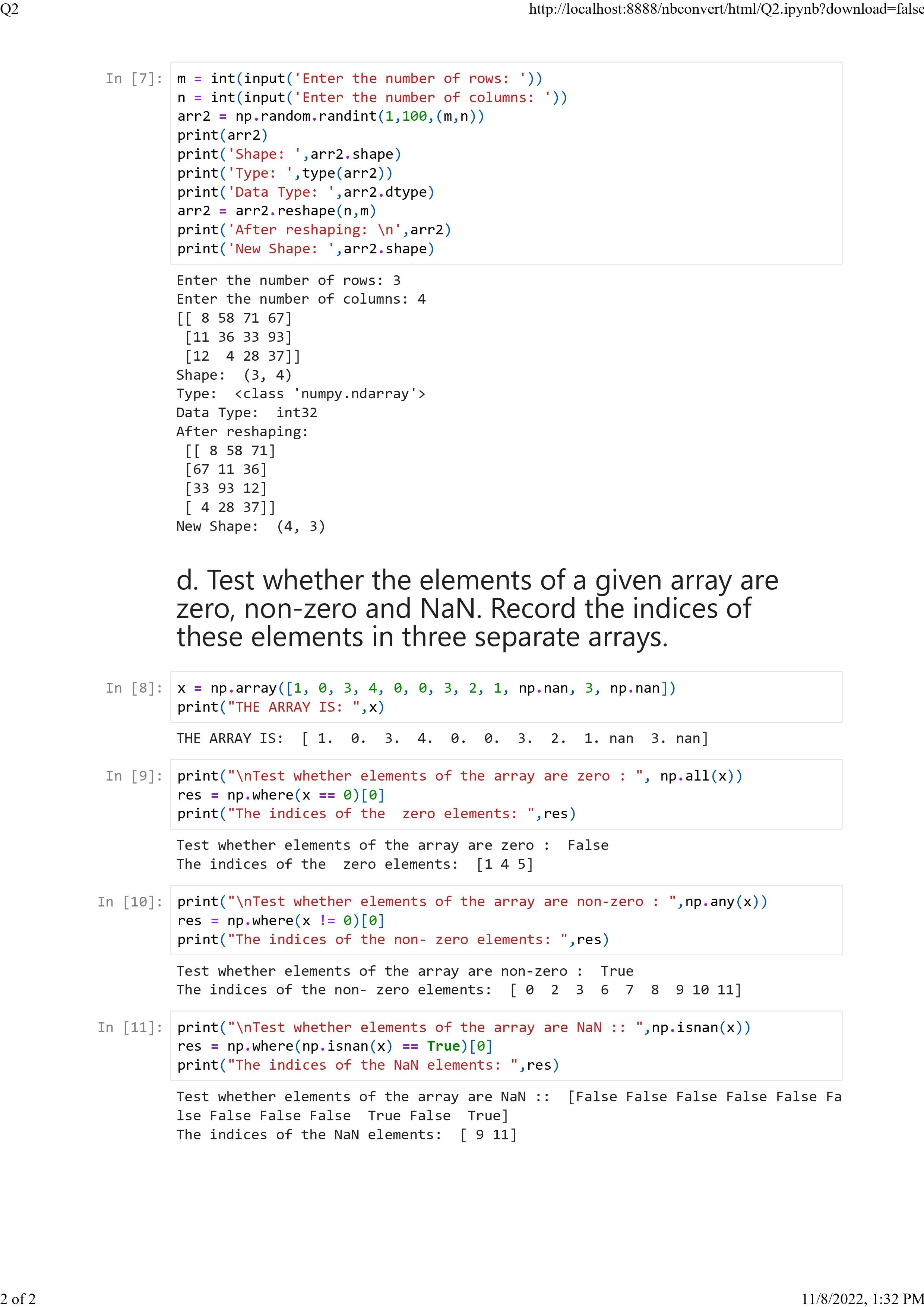
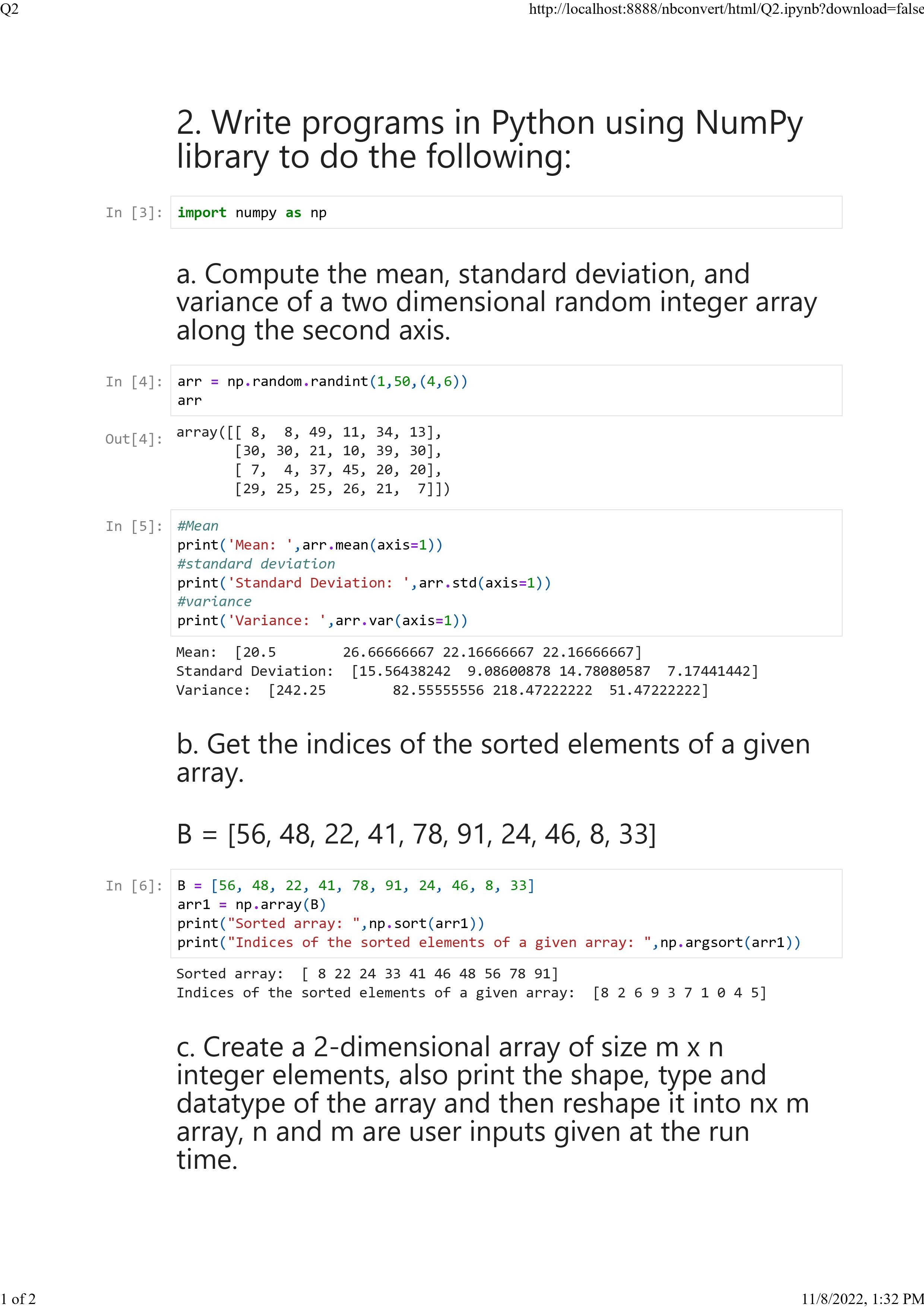
t1[keys[0]] **=** data1[i] t1[keys[1]] **=** data2[i] res**.**append(t1)

print(res)

[{'Boys': 72, 'Girls': 63}, {'Boys': 68, 'Girls': 65}, {'Boys': 70, 'Girls': 6

9}, {'Boys': 69, 'Girls': 62}, {'Boys': 74, 'Girls': 61}]

1 of 1 11/9/2022, 2:26 PM



# 3. Create a dataframe 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:

In [6]:

**import** pandas **as** pd

**import** numpy **as** np

df **=** pd**.**DataFrame(np**.**random**.**randint(0,100,size**=**(50,3)), columns**=**list('123'))

**for** c **in** df**.**sample(int(df**.**shape[0]**\***df**.**shape[1]**\***0.10))**.**index: df**.**loc[c,str(np**.**random**.**randint(1,4))]**=**np**.**nan

df

Out[6]:

**1 2 3**

**0** 25.0 83.0 92.0

**1** 87.0 59.0 55.0

**2** 35.0 10.0 76.0

**3** 16.0 72.0 87.0

**4** 93.0 NaN 7.0

**5** 39.0 NaN 95.0

**6** 85.0 45.0 86.0

**7** 24.0 45.0 12.0

**8** 50.0 23.0 37.0

**9** 59.0 26.0 1.0

**10** 98.0 39.0 84.0

**11** 89.0 10.0 NaN

**12** 17.0 0.0 NaN

**13** 46.0 65.0 NaN

**14** 1.0 93.0 4.0

**15** 87.0 0.0 30.0

**16** 5.0 62.0 56.0

**17** 97.0 48.0 70.0

**18** 86.0 91.0 46.0

**19** 92.0 63.0 57.0

**20** 67.0 NaN 16.0

**21** 34.0 0.0 50.0

**22** 17.0 80.0 NaN

**23** 72.0 NaN 81.0

**24** 3.0 90.0 NaN

**25** 63.0 70.0 55.0

**26** 33.0 NaN 68.0

**27** 80.0 87.0 30.0

**28** 77.0 10.0 15.0

**29** 16.0 99.0 3.0

**30** 16.0 43.0 60.0

**31** 7.0 14.0 90.0

**32** 97.0 24.0 54.0

**33** 87.0 NaN 52.0

**34** NaN 31.0 0.0

**35** 28.0 NaN 72.0

**36** 28.0 82.0 66.0

**37** 25.0 75.0 4.0

**38** NaN 94.0 58.0

**1 2 3**

**39** 32.0 30.0 77.0

**40** 8.0 2.0 30.0

**41** 10.0 63.0 15.0

**42** 76.0 51.0 88.0

**43** 4.0 21.0 26.0

**44** 1.0 0.0 15.0

**45** 31.0 10.0 56.0

**46** 67.0 93.0 56.0

**47** 40.0 13.0 46.0

**48** 58.0 68.0 77.0

**49** NaN 53.0 56.0

## Identify and count missing values in a dataframe.

In [7]:

print(df**.**isnull()**.**sum()**.**sum())

15

## Drop the column having more than 5 null values.

In [8]:

**for** col **in** df**.**columns: print(col,df[col]**.**isnull()**.**sum())

df**.**dropna(axis **=** 1,thresh**=**(df**.**shape[0]**-**5))**.**head()

1 3

2 7

3 5

Out[8]: **1 3**

**0** 25.0 92.0

**1** 87.0 55.0

**2** 35.0 76.0

**3** 16.0 87.0

**4** 93.0 7.0

## Identify the row label having maximum of the sum of all values in a row and drop that row.

In [10]:

sum**=**df**.**sum(axis**=**1) print("SUM IS :\n",sum)

print("\nMAXIMUM SUM IS :",sum**.**max()) max\_sum\_row **=** df**.**sum(axis**=**1)**.**idxmax()

print("\nRow Label having maximum sum is :" ,max\_sum\_row)

df **=** df**.**drop(max\_sum\_row ,axis **=**0)

print("\nDATA Frame AFTER REMOVING THE ROW HAVING MAXIMUM SUM VALUE")

df

|  |  |
| --- | --- |
| SUM | IS : |
| 0 | 200.0 |
| 1 | 201.0 |
| 2 | 121.0 |
| 3 | 175.0 |
| 4 | 100.0 |
| 5 | 134.0 |
| 6 | 216.0 |
| 7 | 81.0 |
| 8 | 110.0 |
| 9 | 86.0 |
| 10 | 221.0 |
| 11 | 99.0 |
| 12 | 17.0 |
| 13 | 111.0 |
| 14 | 98.0 |
| 15 | 117.0 |
| 16 | 123.0 |
| 17 | 215.0 |
| 19 | 212.0 |
| 20 | 83.0 |
| 21 | 84.0 |
| 22 | 97.0 |
| 23 | 153.0 |
| 24 | 93.0 |
| 25 | 188.0 |
| 26 | 101.0 |
| 27 | 197.0 |
| 28 | 102.0 |
| 29 | 118.0 |
| 30 | 119.0 |
| 31 | 111.0 |
| 32 | 175.0 |
| 33 | 139.0 |
| 34 | 31.0 |
| 35 | 100.0 |
| 36 | 176.0 |
| 37 | 104.0 |
| 38 | 152.0 |
| 39 | 139.0 |
| 40 | 40.0 |
| 41 | 88.0 |
| 42 | 215.0 |
| 43 | 51.0 |
| 44 | 16.0 |
| 45 | 97.0 |
| 46 | 216.0 |
| 47 | 99.0 |
| 48 | 203.0 |
| 49 | 109.0 |

dtype: float64

MAXIMUM SUM IS : 221.0

Row Label having maximum sum is : 10

DATA Frame AFTER REMOVING THE ROW HAVING MAXIMUM SUM VALUE

Out[10]:

**1 2 3**

**0** 25.0 83.0 92.0

**1** 87.0 59.0 55.0

**2** 35.0 10.0 76.0

**3** 16.0 72.0 87.0

**4** 93.0 NaN 7.0

**5** 39.0 NaN 95.0

**6** 85.0 45.0 86.0

**7** 24.0 45.0 12.0

**8** 50.0 23.0 37.0

**9** 59.0 26.0 1.0

**11** 89.0 10.0 NaN

**12** 17.0 0.0 NaN

**13** 46.0 65.0 NaN

**14** 1.0 93.0 4.0

**15** 87.0 0.0 30.0

**16** 5.0 62.0 56.0

**17** 97.0 48.0 70.0

**19** 92.0 63.0 57.0

**20** 67.0 NaN 16.0

**21** 34.0 0.0 50.0

**22** 17.0 80.0 NaN

**23** 72.0 NaN 81.0

**24** 3.0 90.0 NaN

**25** 63.0 70.0 55.0

**26** 33.0 NaN 68.0

**27** 80.0 87.0 30.0

**28** 77.0 10.0 15.0

**29** 16.0 99.0 3.0

**30** 16.0 43.0 60.0

**31** 7.0 14.0 90.0

**32** 97.0 24.0 54.0

**33** 87.0 NaN 52.0

**34** NaN 31.0 0.0

**35** 28.0 NaN 72.0

**36** 28.0 82.0 66.0

**37** 25.0 75.0 4.0

**38** NaN 94.0 58.0

**39** 32.0 30.0 77.0

**40** 8.0 2.0 30.0

**1 2 3**

**41** 10.0 63.0 15.0

**42** 76.0 51.0 88.0

**43** 4.0 21.0 26.0

**44** 1.0 0.0 15.0

**45** 31.0 10.0 56.0

**46** 67.0 93.0 56.0

**47** 40.0 13.0 46.0

**48** 58.0 68.0 77.0

**49** NaN 53.0 56.0

## Sort the dataframe on the basis of the first colum

In [13]:

sortdf**=**df**.**sort\_values('1') sortdf

Out[13]:

**1 2 3**

**44** 1.0 0.0 15.0

**14** 1.0 93.0 4.0

**24** 3.0 90.0 NaN

**43** 4.0 21.0 26.0

**16** 5.0 62.0 56.0

**31** 7.0 14.0 90.0

**40** 8.0 2.0 30.0

**41** 10.0 63.0 15.0

**3** 16.0 72.0 87.0

**30** 16.0 43.0 60.0

**29** 16.0 99.0 3.0

**12** 17.0 0.0 NaN

**22** 17.0 80.0 NaN

**7** 24.0 45.0 12.0

**37** 25.0 75.0 4.0

**0** 25.0 83.0 92.0

**36** 28.0 82.0 66.0

**35** 28.0 NaN 72.0

**45** 31.0 10.0 56.0

**39** 32.0 30.0 77.0

**26** 33.0 NaN 68.0

**21** 34.0 0.0 50.0

**2** 35.0 10.0 76.0

**5** 39.0 NaN 95.0

**47** 40.0 13.0 46.0

**13** 46.0 65.0 NaN

**8** 50.0 23.0 37.0

**48** 58.0 68.0 77.0

**9** 59.0 26.0 1.0

**25** 63.0 70.0 55.0

**20** 67.0 NaN 16.0

**46** 67.0 93.0 56.0

**23** 72.0 NaN 81.0

**42** 76.0 51.0 88.0

**28** 77.0 10.0 15.0

**27** 80.0 87.0 30.0

**6** 85.0 45.0 86.0

**15** 87.0 0.0 30.0

**33** 87.0 NaN 52.0

**1 2 3**

**1** 87.0 59.0 55.0

**11** 89.0 10.0 NaN

**19** 92.0 63.0 57.0

**4** 93.0 NaN 7.0

**17** 97.0 48.0 70.0

**32** 97.0 24.0 54.0

**34** NaN 31.0 0.0

**38** NaN 94.0 58.0

**49** NaN 53.0 56.0

## Remove all duplicates from the first column.

In [15]:

df **=**df**.**drop\_duplicates(subset**=**'1',keep **=** "first") df

Out[15]:

**1 2 3**

**0** 25.0 83.0 92.0

**1** 87.0 59.0 55.0

**2** 35.0 10.0 76.0

**3** 16.0 72.0 87.0

**4** 93.0 NaN 7.0

**5** 39.0 NaN 95.0

**6** 85.0 45.0 86.0

**7** 24.0 45.0 12.0

**8** 50.0 23.0 37.0

**9** 59.0 26.0 1.0

**11** 89.0 10.0 NaN

**12** 17.0 0.0 NaN

**13** 46.0 65.0 NaN

**14** 1.0 93.0 4.0

**16** 5.0 62.0 56.0

**17** 97.0 48.0 70.0

**19** 92.0 63.0 57.0

**20** 67.0 NaN 16.0

**21** 34.0 0.0 50.0

**23** 72.0 NaN 81.0

**24** 3.0 90.0 NaN

**25** 63.0 70.0 55.0

**26** 33.0 NaN 68.0

**27** 80.0 87.0 30.0

**28** 77.0 10.0 15.0

**31** 7.0 14.0 90.0

**34** NaN 31.0 0.0

**35** 28.0 NaN 72.0

**39** 32.0 30.0 77.0

**40** 8.0 2.0 30.0

**41** 10.0 63.0 15.0

**42** 76.0 51.0 88.0

**43** 4.0 21.0 26.0

**45** 31.0 10.0 56.0

**47** 40.0 13.0 46.0

**48** 58.0 68.0 77.0

## Find the correlation between first and second column and covariance between second and third column

In [17]:

correlation **=** df['1']**.**corr(df['2'])

print("CORRELATION between column 1 and 2 : ", correlation) covariance **=** df['2']**.**cov(df['3'])

print("COVARIANCE between column 2 and 3 :",covariance)

CORRELATION between column 1 and 2 : 0.025998863412796235

COVARIANCE between column 2 and 3 : 86.28923076923074

## Detect the outliers and remove the rows having outliers.

In [19]:

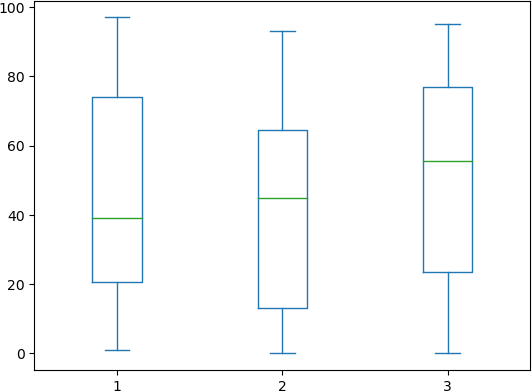
df**.**plot**.**box()

Out[19]:

In [20]:

df1 **=** pd**.**cut(df['2'],bins**=**5)**.**head() df1

<AxesSubplot:>



## Discretize second column and create 5 bins

Out[20]:

0 (74.4, 93.0]

|  |  |  |
| --- | --- | --- |
| 1 | (55.8, | 74.4] |
| 2 | (-0.093, | 18.6] |
| 3 | (55.8, | 74.4] |
| 4 |  | NaN |

Name: 2, dtype: category

Categories (5, interval[float64, right]): [(-0.093, 18.6] < (18.6, 37.2] < (37.2, 55.8] < (5

5.8, 74.4] < (74.4, 93.0]]

# Q4. Consider two excel files having attendance of a workshop’s participants for two days. Each file has three fields ‘Name’, ‘Time of joining’, duration (in minutes) where names are unique within a file. Note that duration may take one of three values (30, 40, 50) only. Import the data into two dataframes and do the following:

In [16]:

**import** numpy **as** np

**import** pandas **as** pd

dfDay1 **=** pd**.**read\_excel('F:\CS\sem 5\Data Analysis and Visualisation\Practicals\Day1\_register. dfDay2 **=** pd**.**read\_excel('F:\CS\sem 5\Data Analysis and Visualisation\Practicals\Day2\_register.

dfDay1, dfDay2

Out[16]:

In [6]:

merged**=**pd**.**merge(dfDay1,dfDay2,how**=**'inner',on**=**'Name') merged

( Name Time of Joining Duration 0 Christine 11:00:00 40

1 Susan 11:04:00 30

2 Margaret 11:08:00 30

3 Judith 11:01:00 40

4 Jennifer 11:12:00 50

5 Mary 11:10:00 30

6 Elizabeth 11:11:00 30

7 Patricia 11:07:00 40

8 Linda 11:08:00 50

9 Barbara 11:15:00 30

10 Lynette 11:02:00 40

11 Robyn 11:03:00 30

12 Anne 11:19:00 30

13 Karen 11:13:00 40

14 Helen 11:05:00 50,

Name Time of Joining Duration 0 Christine 11:00:00 40

1 Susan 11:06:00 30

2 Lynette 11:10:00 40

3 Judith 11:09:00 40

4 Mary 11:10:00 50

5 Sandra 11:12:00 30

6 Elizabeth 11:08:00 30

7 Patricia 11:08:00 40

8 Wendy 11:13:00 40

9 Barbara 11:14:00 30

10 Janet 11:02:00 50

11 Robyn 11:00:00 30

12 Heather 11:08:00 30

13 Karen 11:09:00 40

14 Diane 11:06:00 30)

## Perform merging of the two dataframes to find the names of students who had attended the workshop on both days.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Out[6]: | **Name** | **Time of Joining\_x** | **Duration\_x** | **Time of Joining\_y** | **Duration\_y** |
|  | **0** Christine | 11:00:00 | 40 | 11:00:00 | 40 |
|  | **1** Susan | 11:04:00 | 30 | 11:06:00 | 30 |
|  | **2** Judith | 11:01:00 | 40 | 11:09:00 | 40 |
|  | **3** Mary | 11:10:00 | 30 | 11:10:00 | 50 |
|  | **4** Elizabeth | 11:11:00 | 30 | 11:08:00 | 30 |
|  | **5** Patricia | 11:07:00 | 40 | 11:08:00 | 40 |
|  | **6** Barbara | 11:15:00 | 30 | 11:14:00 | 30 |
|  | **7** Lynette | 11:02:00 | 40 | 11:10:00 | 40 |
|  | **8** Robyn | 11:03:00 | 30 | 11:00:00 | 30 |
|  | **9** Karen | 11:13:00 | 40 | 11:09:00 | 40 |

1. Find names of all students who have attended workshop on either of the days.

In [7]:

either\_day **=** pd**.**merge(dfDay1,dfDay2,how**=**'outer',on**=**'Name') either\_day

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Out[7]: | **Name** | **Time of Joining\_x** | **Duration\_x** | **Time of Joining\_y** | **Duration\_y** |
|  | **0** Christine | 11:00:00 | 40.0 | 11:00:00 | 40.0 |
|  | **1** Susan | 11:04:00 | 30.0 | 11:06:00 | 30.0 |
|  | **2** Margaret | 11:08:00 | 30.0 | NaN | NaN |
|  | **3** Judith | 11:01:00 | 40.0 | 11:09:00 | 40.0 |
|  | **4** Jennifer | 11:12:00 | 50.0 | NaN | NaN |
|  | **5** Mary | 11:10:00 | 30.0 | 11:10:00 | 50.0 |
|  | **6** Elizabeth | 11:11:00 | 30.0 | 11:08:00 | 30.0 |
|  | **7** Patricia | 11:07:00 | 40.0 | 11:08:00 | 40.0 |
|  | **8** Linda | 11:08:00 | 50.0 | NaN | NaN |
|  | **9** Barbara | 11:15:00 | 30.0 | 11:14:00 | 30.0 |
|  | **10** Lynette | 11:02:00 | 40.0 | 11:10:00 | 40.0 |
|  | **11** Robyn | 11:03:00 | 30.0 | 11:00:00 | 30.0 |
|  | **12** Anne | 11:19:00 | 30.0 | NaN | NaN |
|  | **13** Karen | 11:13:00 | 40.0 | 11:09:00 | 40.0 |
|  | **14** Helen | 11:05:00 | 50.0 | NaN | NaN |
|  | **15** Sandra | NaN | NaN | 11:12:00 | 30.0 |
|  | **16** Wendy | NaN | NaN | 11:13:00 | 40.0 |
|  | **17** Janet | NaN | NaN | 11:02:00 | 50.0 |
|  | **18** Heather | NaN | NaN | 11:08:00 | 30.0 |
|  | **19** Diane | NaN | NaN | 11:06:00 | 30.0 |

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

In [8]:

either\_day['Name']**.**count()

Out[8]:

In [9]:

both\_days **=** pd**.**merge(dfDay1,dfDay2,how**=**'outer',on**=**['Name','Duration'])**.**copy() *# creates a co*

both\_days**.**fillna(value**=**'-',inplace**=True**) *# to fill out the missing values in the given series*

both\_days**.**set\_index(['Name','Duration']) *# a method to set a List as index of a Data Frame*

20

1. Merge two data frames and use two columns names and duration as multi-row indexes. Generate descriptive statistics for this multi-index.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Out[9]: | **Name** | **Duration** | **Time of Joining\_x** | **Time of Joining\_y** |
|  | **Christine** | **40** | 11:00:00 | 11:00:00 |
|  | **Susan** | **30** | 11:04:00 | 11:06:00 |
|  | **Margaret** | **30** | 11:08:00 | - |
|  | **Judith** | **40** | 11:01:00 | 11:09:00 |
|  | **Jennifer** | **50** | 11:12:00 | - |
|  | **Mary** | **30** | 11:10:00 | - |
|  | **Elizabeth** | **30** | 11:11:00 | 11:08:00 |
|  | **Patricia** | **40** | 11:07:00 | 11:08:00 |
|  | **Linda** | **50** | 11:08:00 | - |
|  | **Barbara** | **30** | 11:15:00 | 11:14:00 |
|  | **Lynette** | **40** | 11:02:00 | 11:10:00 |
|  | **Robyn** | **30** | 11:03:00 | 11:00:00 |
|  | **Anne** | **30** | 11:19:00 | - |
|  | **Karen** | **40** | 11:13:00 | 11:09:00 |
|  | **Helen** | **50** | 11:05:00 | - |
|  | **Mary** | **50** | - | 11:10:00 |
|  | **Sandra** | **30** | - | 11:12:00 |
|  | **Wendy** | **40** | - | 11:13:00 |
|  | **Janet** | **50** | - | 11:02:00 |
|  | **Heather** | **30** | - | 11:08:00 |
|  | **Diane** | **30** | - | 11:06:00 |

Q5. Taking Iris data, plot the following with proper legend and axis labels: (Download IRIS data from: <https://archive.ics.uci.edu/ml/datasets/iris>or import it from sklearn.datasets)

In [10]:

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

iris **=** sns**.**load\_dataset('iris')

## Plot bar chart to show the frequency of each class label in the data.

In [11]:

sns**.**countplot(x**=**'species',data**=**iris,palette**=**'Set2') plt**.**xlabel('Species')

plt**.**ylabel('Frequency') plt**.**title('Frequency of Each class label')

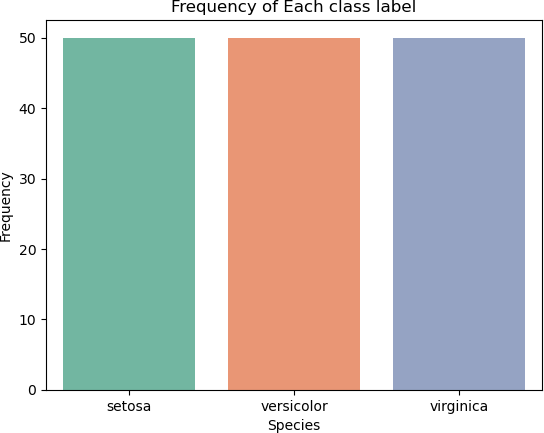
Out[11]:

In [12]:

plt**.**scatter(x**=**'petal\_width',y**=**'sepal\_width',data**=**iris) plt**.**xlabel('Petal Width')

plt**.**ylabel('Sepal Width') plt**.**title("Scatter plot")

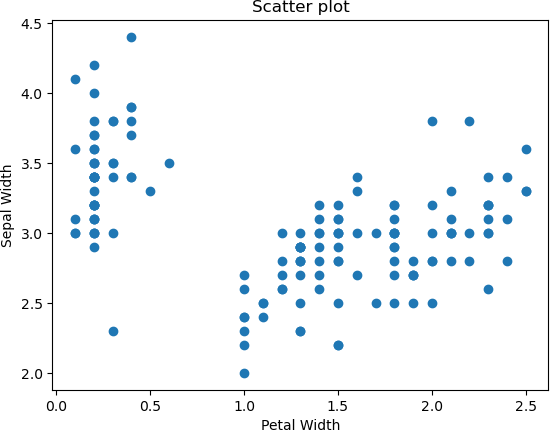
Text(0.5, 1.0, 'Frequency of Each class label')



## Draw a scatter plot for Petal width vs sepal width.

Out[12]:

Text(0.5, 1.0, 'Scatter plot')



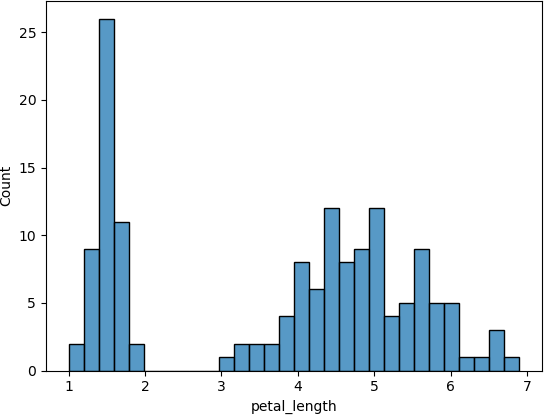
## Plot density distribution for feature petal length

In [13]:

sns**.**histplot(iris['petal\_length'],kde**=False**,bins**=**30)

Out[13]:

<AxesSubplot:xlabel='petal\_length', ylabel='Count'>



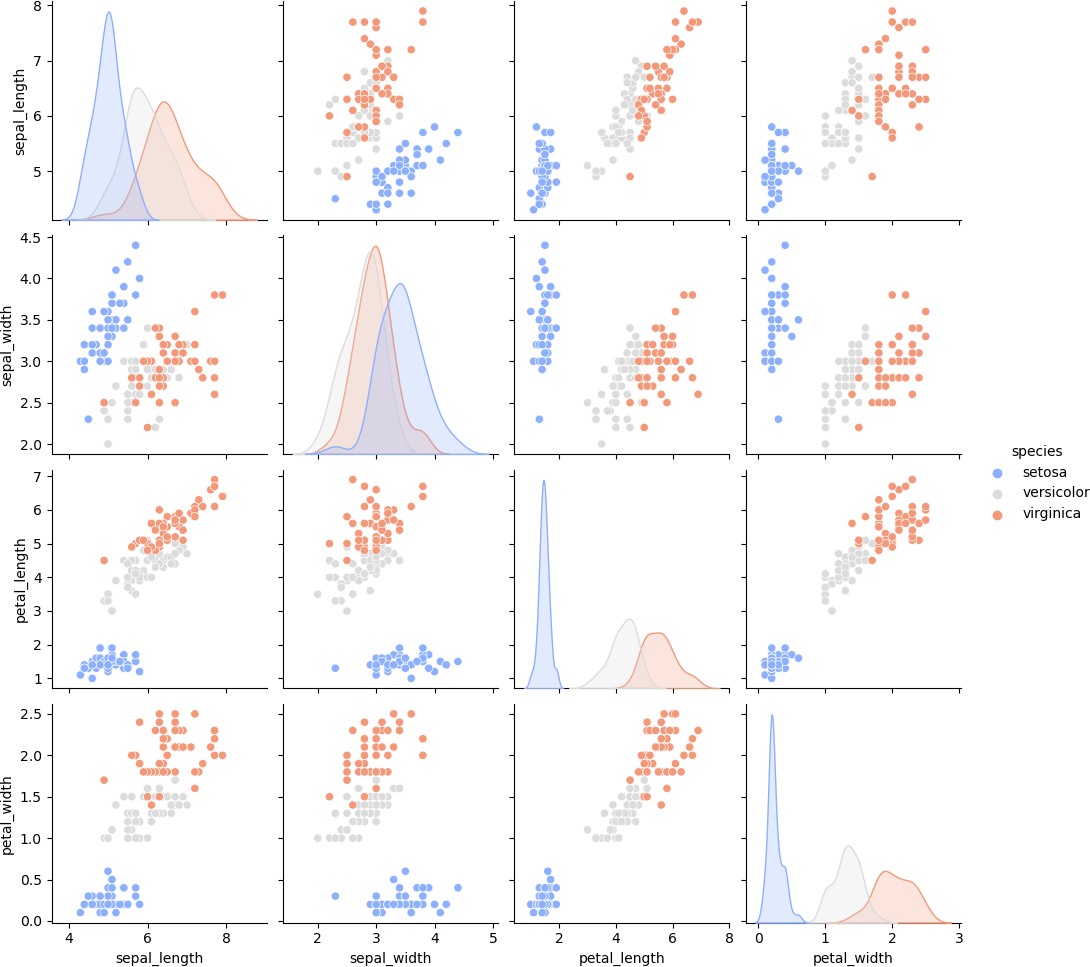
## Use a pair plot to show pairwise bivariate distribution in the Iris Dataset.

In [15]:

Out[15]:

sns**.**pairplot(iris,hue**=**'species',palette**=**'coolwarm')

<seaborn.axisgrid.PairGrid at 0x225675c35b0>



# Q6. Consider any sales training/ weather forecasting dataset

In [1]:

**import** pandas **as** pd

**import** numpy **as** np

**from** random **import** sample

**import** os

**from** dateutil.parser **import** parse

df **=** pd**.**read\_csv("F:\CS\sem 5\Data Analysis and Visualisation\Practicals\weather df

### 1 of 7 11/9/2022, 12:59 PM

Out[1]:

**Formatted Summary Date**

**Precip Type**

**Temperature**

**(C)**

**Apparent Temperature**

**(C)**

**Humidity**

**Wind Speed (km/h)**

**W**

**Bea (degr**

**0** 2006-04-01

00:00:00+02:00

**1** 2006-04-01

01:00:00+02:00

Partly Cloudy

rain

9.355556

7.227778

0.86 14.2646

2

Partly Cloudy

rain 9.472222 7.388889 0.89 14.1197 2

**2** 2006-04-01

02:00:00+02:00

**3** 2006-04-01

03:00:00+02:00

Partly Cloudy

rain

8.288889

5.944444

0.83 14.1036

2

Mostly Cloudy

rain 9.377778 9.377778 0.89 3.9284 2

**4** 2006-04-01

04:00:00+02:00

**...** ... ... ... ... ... ... ...

Mostly Cloudy

rain 8.755556 6.977778 0.83 11.0446 2

**96448** 2016-09-09

19:00:00+02:00

Partly Cloudy

rain 26.016667 26.016667 0.43 10.9963

**96449** 2016-09-09

20:00:00+02:00

Partly Cloudy

rain

24.583333

24.583333

0.48 10.0947

**96450** 2016-09-09

21:00:00+02:00

Partly Cloudy

rain 22.038889 22.038889 0.56 8.9838

**96451** 2016-09-09

22:00:00+02:00

Partly Cloudy

rain

21.522222

21.522222

0.60 10.5294

**96452** 2016-09-09

23:00:00+02:00

Partly Cloudy

rain 20.438889 20.438889 0.61 5.8765

96453 rows × 12 columns

2 of 7 11/9/2022, 12:59 PM

* + 1. Compute mean of a series grouped by another series

In [2]:

df[['Humidity']]**.**groupby(df['Summary'])**.**mean()

Out[2]:

**Summary**

**Humidity**

**Breezy** 0.637778

**Breezy and Dry** 0.260000

**Breezy and Foggy** 0.938571

**Breezy and Mostly Cloudy** 0.637054

**Breezy and Overcast** 0.763144

**Breezy and Partly Cloudy** 0.545803

**Clear** 0.729708

**Dangerously Windy and Partly Cloudy** 0.490000

**Drizzle** 0.867949

**Dry** 0.230294

**Dry and Mostly Cloudy** 0.242143

**Dry and Partly Cloudy** 0.240814

**Foggy** 0.950765

**Humid and Mostly Cloudy** 0.874250

**Humid and Overcast** 0.881429

**Humid and Partly Cloudy** 0.848824

**Light Rain** 0.888095

**Mostly Cloudy** 0.725069

**Overcast** 0.837232

**Partly Cloudy** 0.648571

**Rain** 0.947000

**Windy** 0.572500

**Windy and Dry** 0.240000

**Windy and Foggy** 0.900000

**Windy and Mostly Cloudy** 0.600000

**Windy and Overcast** 0.708667

**Windy and Partly Cloudy** 0.528806

3 of 7 11/9/2022, 12:59 PM

## Fill an intermittent time series to replace all missing dates with values of previous non-missing date.

In [3]:

df['Formatted Date']**.**isnull()**.**sum() data **=** df**.**copy()

random\_indices **=** data['Formatted Date']**.**sample(1000)**.**index data**.**loc[random\_indices,'Formatted Date'] **=** np**.**nan

data['Formatted Date']**.**isna()**.**value\_counts()

data**.**fillna(method **=** 'ffill')**.**loc[random\_indices, 'Formatted Date']**.**head(5)

Out[3]:

In [5]:

temp **=** df**.**copy()

dummy\_yearMonths **=** ['Jan 2005', 'Feb 2016', 'Mar 2017', 'Apr 2018', 'May 2019', temp\_size **=** len(temp)

temp['Dummy\_Date'] **=** [sample(dummy\_yearMonths,1)[0]

**for** i **in** range(temp\_size)] temp

69306 2013-10-04 17:00:00+02:00

|  |  |  |
| --- | --- | --- |
| 71530 | 2014-08-06 | 09:00:00+02:00 |
| 52037 | 2011-09-15 | 04:00:00+02:00 |
| 51522 | 2011-10-23 | 18:00:00+02:00 |
| 28474 | 2009-12-07 | 09:00:00+01:00 |

Name: Formatted Date, dtype: object

## Perform appropriate year-month string to dates conversion.

### 4 of 7 11/9/2022, 12:59 PM

Out[5]:

**Formatted Summary Date**

**Precip Type**

**Temperature**

**(C)**

**Apparent Temperature**

**(C)**

**Humidity**

**Wind Speed (km/h)**

**W**

**Bea (degr**

**0** 2006-04-01

00:00:00+02:00

**1** 2006-04-01

01:00:00+02:00

Partly Cloudy

rain

9.355556

7.227778

0.86 14.2646

2

Partly Cloudy

rain 9.472222 7.388889 0.89 14.1197 2

**2** 2006-04-01

02:00:00+02:00

**3** 2006-04-01

03:00:00+02:00

Partly Cloudy

rain

8.288889

5.944444

0.83 14.1036

2

Mostly Cloudy

rain 9.377778 9.377778 0.89 3.9284 2

**4** 2006-04-01

04:00:00+02:00

**...** ... ... ... ... ... ... ...

Mostly Cloudy

rain 8.755556 6.977778 0.83 11.0446 2

**96448** 2016-09-09

19:00:00+02:00

Partly Cloudy

rain 26.016667 26.016667 0.43 10.9963

**96449** 2016-09-09

20:00:00+02:00

Partly Cloudy

rain

24.583333

24.583333

0.48 10.0947

**96450** 2016-09-09

21:00:00+02:00

Partly Cloudy

rain 22.038889 22.038889 0.56 8.9838

**96451** 2016-09-09

22:00:00+02:00

Partly Cloudy

rain

21.522222

21.522222

0.60 10.5294

**96452** 2016-09-09

23:00:00+02:00

Partly Cloudy

rain 20.438889 20.438889 0.61 5.8765

96453 rows × 13 columns

In [6]:

temp['Dummy\_Date']**.**map(**lambda** d: parse(d))

### 5 of 7 11/9/2022, 12:59 PM

Out[6]:

In [9]:

**def** sort\_values(df, column**=**'Summary', ascending **= True**): **return** df**.**sort\_values(by**=**column, ascending **=** ascending)

grouped **=** df**.**groupby(['Summary', 'Precip Type'])

temp **=** grouped**.**apply(sort\_values, column **=** 'Temperature (C)', ascending **= False**) temp['Temperature (C)']

0 2019-05-09

|  |  |
| --- | --- |
| 1 | 2016-02-09 |
| 2 | 2021-06-09 |
| 3 | 2019-05-09 |
| 4 | 2019-05-09 |
| 96448 | 2017-03-09 |
| 96449 | 2017-03-09 |
| 96450 | 2005-01-09 |
| 96451 | 2005-01-09 |
| 96452 | 2021-06-09 |
| Name: | Dummy\_Date, Length: 96453, dtype: datetime64[ns] |

## Split a dataset to group by two columns and then sort the aggregated results within the groups.

Out[9]:

In [10]:

df**.**groupby(['Summary'])**.**count()

df**.**groupby(['Summary'])**.**size()

Summary Precip Type

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Breezy |  | rain | 12805 | 37.588889 |
|  |  |  | 30183 | 33.888889 |
|  |  |  | 53776 | 30.900000 |
|  |  |  | 53777 | 29.938889 |
|  |  |  | 45035 | 29.738889 |
| Windy and | Partly Cloudy | rain | 63542 | 3.983333 |
|  | | | 63540 | 3.911111 |
| 63539 | 2.872222 |
| 31833 | 2.222222 |
| 31832 | 1.111111 |

Name: Temperature (C), Length: 95936, dtype: float64

## Split a given dataframe into groups with bin counts.

### 6 of 7 11/9/2022, 12:59 PM

Out[10]:

Summary

Breezy 54

Breezy and Dry 1

Breezy and Foggy 35

Breezy and Mostly Cloudy 516

Breezy and Overcast 528

Breezy and Partly Cloudy 386

Clear 10890

|  |  |  |
| --- | --- | --- |
| Dangerously Windy and Partly | Cloudy | 1 |
| Drizzle |  | 39 |
| Dry |  | 34 |
| Dry and Mostly Cloudy |  | 14 |
| Dry and Partly Cloudy |  | 86 |
| Foggy |  | 7148 |
| Humid and Mostly Cloudy |  | 40 |
| Humid and Overcast |  | 7 |
| Humid and Partly Cloudy |  | 17 |
| Light Rain |  | 63 |
| Mostly Cloudy |  | 28094 |
| Overcast |  | 16597 |
| Partly Cloudy |  | 31733 |
| Rain |  | 10 |
| Windy |  | 8 |
| Windy and Dry |  | 1 |
| Windy and Foggy |  | 4 |
| Windy and Mostly Cloudy |  | 35 |
| Windy and Overcast |  | 45 |
| Windy and Partly Cloudy |  | 67 |
| dtype: int64 |  |  |

7 of 7 11/9/2022, 12:59 PM

# Q7: Consider a data frame containing data about students i.e. name, gender and passing division:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| In [2]: | **import** numpy **as** np  **import** pandas **as** pd | | | | |
|  | df df | **=** pd**.**read\_csv('F:\CS\sem | 5\Data | Analysis and | Visualisation\Practicals\Student |
| Out[2]: |  | **Name Birth\_Month** | **Gender** | **Pass\_Division** |  |
|  | **0** | Mudit Chauhan December | M | III |  |
|  | **1** | Seema Chopra January | F | II |  |
|  | **2** | Rani Gupta March | F | I |  |
|  | **3** | Aditya Narayan October | M | I |  |
|  | **4** | Sanjeev Sahni February | M | II |  |
|  | **5** | Prakash Kumar December | M | III |  |
|  | **6** | Ritu Agarwal September | F | I |  |
|  | **7** | Akshay Goel August | M | I |  |
|  | **8** | Meeta Kulkarni July | F | II |  |
|  | **9** | Preeti Ahuja November | F | II |  |
|  | **10** | Sunil Das Gupta April | M | III |  |
|  | **11** | Sunali Sapre January | F | I |  |
|  | **12** | Rashmi Talwar June | F | III |  |
|  | **13** | Ashish Dubey May | M | II |  |
|  | **14** | Kiran Sharma February | F | II |  |
|  | **15** | Sameer Bansal October | M | I |  |

## Perform one hot encoding of the last two columns of categorical data using the get\_dummies() function.

In [3]:

df1 **=** pd**.**get\_dummies(df, columns**=**['Gender', 'Pass\_Division']) df1

### 1 of 3 11/9/2022, 1:19 PM

**0** Mudit Chauhan

December 0 1 0 0

**1**

Seema Chopra

January

1

0

0

1

**2** Rani

Gupta

March 1 0 1 0

**3**

Aditya Narayan

October

0

1

1

0

**4** Sanjeev Sahni

**5**

Prakash Kumar

December

0

1

0

0

February 0 1 0 1

**6** Ritu

Agarwal

September 1 0 1 0

**7**

Akshay Goel

August

0

1

1

0

**8** Meeta Kulkarni

**9**

Preeti Ahuja

November

1

0

0

1

July 1 0 0 1

Sunil

**10** Das

Gupta

April 0 1 0 0

**11**

Sunali Sapre

January

1

0

1

0

**12** Rashmi Talwar

June 1 0 0 0

**13**

Ashish Dubey

May

0

1

0

1

**14** Kiran Sharma

February 1 0 0 1

**15**

Sameer Bansal

October

0

1

1

0

## Sort this data frame on the “Birth Month” column (i.e. January to December). Hint: Convert Month to Categorical.

In [4]:

months **=** ["January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December"]

df1['Birth\_Month'] **=** pd**.**Categorical(df1['Birth\_Month'], categories**=**months, order df1**.**sort\_values(by**=**'Birth\_Month', inplace**=True**)

df1

### 2 of 3 11/9/2022, 1:19 PM

**1** Seema Chopra

**11**

Sunali Sapre

January

1

0

1

0

January 1 0 0 1

**4** Sanjeev Sahni

**14**

Kiran Sharma

February

1

0

0

1

February 0 1 0 1

**2** Rani

Gupta

**10**

Sunil Das Gupta

April

0

1

0

0

March 1 0 1 0

**13** Ashish Dubey

**12**

Rashmi Talwar

June

1

0

0

0

May 0 1 0 1

**8** Meeta Kulkarni

**7**

Akshay Goel

August

0

1

1

0

July 1 0 0 1

**6** Ritu

Agarwal

**3**

Aditya Narayan

October

0

1

1

0

September 1 0 1 0

**15** Sameer Bansal

**9**

Preeti Ahuja

November

1

0

0

1

October 0 1 1 0

**0** Mudit Chauhan

December 0 1 0 0

**5**

Prakash Kumar

December

0

1

0

0

3 of 3 11/9/2022, 1:19 PM

# Q8. Consider the following data frame containing a family name, gender of the family member and her/his monthly income in each record

In [10]:

**import** numpy **as** np

**import** pandas **as** pd

df **=** pd**.**read\_csv('F:\CS\sem 5\Data Analysis and Visualisation\Practicals/family\_ df

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Out[10]: |  | **Name** | **Gender** | **Monthly Income (Rs.)** |
|  | **0** | Shah | Male | 114000 |
|  | **1** | Vats | Male | 65000 |
|  | **2** | Vats | Female | 43150 |
|  | **3** | Kumar | Female | 69500 |
|  | **4** | Vats | Female | 155000 |
|  | **5** | Kumar | Male | 103000 |
|  | **6** | Shah | Male | 55000 |
|  | **7** | Shah | Female | 112400 |
|  | **8** | Kumar | Female | 81030 |
|  | **9** | Vats | Male | 71900 |

## Calculate and display familywise gross monthly income.

In [13]:

df**.**groupby('Name')['Monthly Income (Rs.)']**.**sum()

Out[13]:

In [12]:

df1 **=** pd**.**DataFrame(df**.**groupby('Name')['Monthly Income (Rs.)']**.**max()) df1

Name

|  |  |
| --- | --- |
| Kumar | 253530 |
| Shah | 281400 |
| Vats | 335050 |
| Name: | Monthly Income (Rs.), dtype: int64 |

## Calculate and display the member with the highest monthly income in a family.

### 1 of 2 11/9/2022, 2:21 PM

Out[12]:

**Name**

**Monthly Income (Rs.)**

**Kumar** 103000

**Shah** 114000

**Vats** 155000

## Calculate and display monthly income of all members with income greater than Rs. 60000.00.

In [14]:

df[df['Monthly Income (Rs.)'] **>** 60000]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Out[14]: |  | **Name** | **Gender** | **Monthly Income (Rs.)** |
|  | **0** | Shah | Male | 114000 |
|  | **1** | Vats | Male | 65000 |
|  | **3** | Kumar | Female | 69500 |
|  | **4** | Vats | Female | 155000 |
|  | **5** | Kumar | Male | 103000 |
|  | **7** | Shah | Female | 112400 |
|  | **8** | Kumar | Female | 81030 |
|  | **9** | Vats | Male | 71900 |

## Calculate and display the average monthly income of the female members in the Shah family.

In [15]:

df[(df['Name'] **==** 'Shah') **&** (df['Gender'] **==** 'Female')]['Monthly Income (Rs.)']

Out[15]:

112400.0

### 2 of 2 11/9/2022, 2:21 PM