

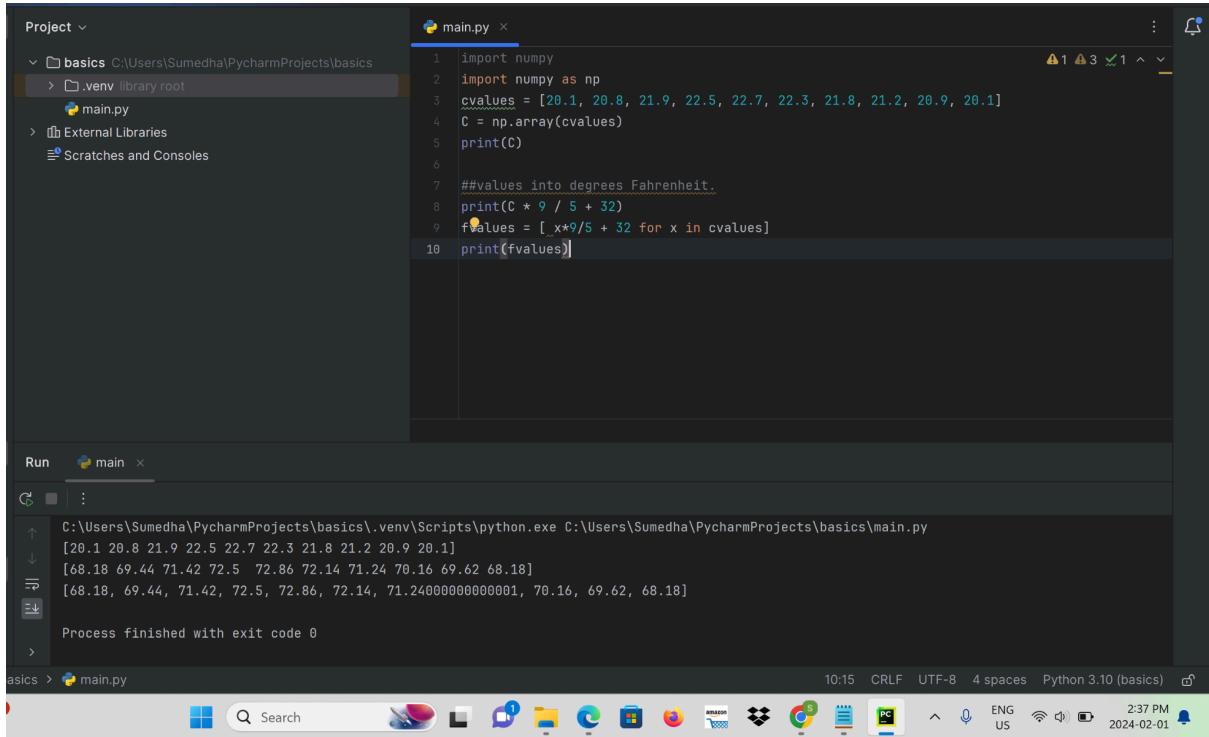
NAME : AKULA SHARATH CHANDRA

BATCH : DATA ENGINEERING

DATE : 01-02-24

**TOPIC : NUMPY,CREATE DATAFRAMES USING
DYNAMIC COLUMN LIST ON CSV DATA,PERFORMING
JOINS BETWEEN PANDA DATA FRAMES,GET UNIQUE
VALUES FROM A LIST IN PYTHON,GET COUNT BY
STATUS USING PANDAS DATAFRAME APIS**

1)NUMPY: NumPy is a powerful numerical computing library in Python. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.



```
Project basics C:\Users\Sumedha\PycharmProjects\basics
  .venv library root
    main.py
  External Libraries
  Scratches and Consoles

main.py
1 import numpy
2 import numpy as np
3 cvalues = [20.1, 20.8, 21.9, 22.5, 22.7, 22.3, 21.8, 21.2, 20.9, 20.1]
4 C = np.array(cvalues)
5 print(C)
6
7 ##values into degrees Fahrenheit
8 print(C * 9 / 5 + 32)
9 fvalues = [x*9/5 + 32 for x in cvalues]
10 print(fvalues)

Run main
C:\Users\Sumedha\PycharmProjects\basics\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\basics\main.py
[20.1 20.8 21.9 22.5 22.7 22.3 21.8 21.2 20.9 20.1]
[68.18 69.44 71.42 72.5 72.86 72.14 71.24 70.16 69.62 68.18]
[68.18, 69.44, 71.42, 72.5, 72.86, 72.14, 71.24000000000001, 70.16, 69.62, 68.18]

Process finished with exit code 0
```

2)numpy.ndarray: There are functions provided by Numpy to create arrays with evenly spaced values within a given interval. One 'arange' uses a given distance and the other one 'linspace' needs the number of elements and creates the distance automatically.

→Creation of Arrays with Evenly Spaced Values

i)Arrange :

The syntax of arange:

```
arange([start[, stop[, step]], [, dtype=None])
```

The screenshot shows the PyCharm IDE interface. The left sidebar displays a project structure with a 'basics' folder containing a 'main.py' file. The right pane shows the code for 'main.py' which imports numpy and uses np.arange to print integer arrays from 1 to 10. Below the code is a terminal window showing the execution of the script and its output. The output includes four separate examples of np.arange: one with integer steps (1 to 10), one with float steps (0 to 10), one with float start/stop (0.5 to 10.4), and one with float start/stop/steps (0.5 to 10.4 with step 0.8).

```
10 #arrange
11 import numpy as np
12 a = np.arange(1, 10)
13 print(a)
14 x = range(1, 10)
15 print(x)    # x is an iterator
16 print(list(x))
17 # further arange examples:
18 x = np.arange(10.4)
19 print(x)
20 x = np.arange(0.5, 10.4, 0.8)
21 print(x)
22 print(x)
23
```

```
Run main x
G : 
[1 2 3 4 5 6 7 8 9]
range(1, 10)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
[ 0.5  1.3  2.1  2.9  3.7  4.5  5.3  6.1  6.9  7.7  8.5  9.3 10.1]

> Process finished with exit code 0
```

if we use a float value for the step parameter, as you can see in the following example:

This screenshot is similar to the previous one, showing the PyCharm IDE with the 'main.py' code. However, the last line of code has been modified to include a float step value: np.arange(12.04, 12.84, 0.08). The terminal output remains the same as in the first screenshot, showing the four examples of np.arange.

```
10 #arrange
11 import numpy as np
12 a = np.arange(1, 10)
13 print(a)
14 x = range(1, 10)
15 print(x)    # x is an iterator
16 print(list(x))
17 # further arange examples:
18 x = np.arange(10.4)
19 print(x)
20 x = np.arange(0.5, 10.4, 0.8)
21 print(x)
22 print(x)
23 np.arange(12.04, 12.84, 0.08)
24 |
```

```
Run main x
G : 
range(1, 10)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
[ 0.5  1.3  2.1  2.9  3.7  4.5  5.3  6.1  6.9  7.7  8.5  9.3 10.1]
```

ii) linspace : The syntax of linspace:

`linspace(start, stop, num=50, endpoint=True, retstep=False)`

linspace returns an ndarray, consisting of 'num' equally spaced samples in the closed interval [start, stop] or the half-open interval [start, stop).

example:

The screenshot shows the PyCharm IDE interface. The project tree on the left shows a 'basics' folder containing a 'main.py' file. The code in 'main.py' demonstrates the use of the `np.linspace` function:

```

26 #Linspace
27 import numpy as np
28 # 50 values between 1 and 10:
29 print(np.linspace(start=1, stop=10))
30 # 7 values between 1 and 10:
31 print(np.linspace(start=1, stop=10, num=7))
32 # excluding the endpoint:
33 print(np.linspace(start=1, stop=10, num=7, endpoint=False))

```

The 'Run' tab is selected, showing the output of the script:

```

[ 0.5  1.3  2.1  2.9  3.7  4.5  5.3  6.1  6.9  7.7  8.5  9.3 10.1]
[ 1.          1.18367347  1.36734694  1.55102041  1.73469388  1.91836735
 2.10204082  2.28571429  2.46938776  2.65306122  2.83673469  3.02840816
 3.20408163  3.3877551   3.57142857  3.75510204  3.93877551  4.12244898
 4.30612245  4.48979592  4.67346939  4.85714286  5.04081633  5.2244898
 5.40816327  5.59183673  5.7755102   5.95918367  6.14285714  6.32653061
 6.51020408  6.69387755  6.87755102  7.06122449  7.24489796  7.42857143
 7.6122449   7.79591837  7.97959184  8.16326531  8.34693878  8.53061224
 8.71428571  8.89795918  9.08163265  9.26530612  9.44897959  9.63265306
 9.81632653 10.      ]
[ 1.          2.5  4.  5.5  7.  8.5 10. ]
[1.          2.28571429 3.57142857 4.85714286 6.14285714 7.42857143
 8.71428571]

Process finished with exit code 0

```

The status bar at the bottom indicates: 26:11 CRLF UTF-8 4 spaces Python 3.10 (basics) 2146 BMS.

iii) Retstep: The `retstep` parameter in the `np.linspace` function determines whether to return the step size along with the array of values. If `retstep` is set to true, the function returns a tuple containing the array of values and the step size. If `retstep` is set to false (default), only the array of values is returned.

Example:

The screenshot shows the PyCharm IDE interface. The project tree on the left shows a 'basics' folder containing a 'main.py' file. The code in 'main.py' demonstrates the use of the `np.linspace` function with the `retstep=True` parameter:

```

35 #RETSTEP
36 import numpy as np
37 samples, spacing = np.linspace(start=1, stop=10, retstep=True)
38 print(spacing)
39 samples, spacing = np.linspace(start=1, stop=10, num=20, endpoint=True, retstep=True)
40 print(spacing)
41 samples, spacing = np.linspace(start=1, stop=10, num=20, endpoint=False, retstep=True)
42 print(spacing)
43
44
45
46
47
48
49
50
51
52
53

```

The 'Run' tab is selected, showing the output of the script:

```

0.1836734693877551
0.47368421052631576
0.45
Process finished with exit code 0

```

3)CREATE DATAFRAMES USING DYNAMIC COLUMN LIST ON CSV DATA:

Creating a pandas data frame using CSV files can be achieved in multiple ways.

Method #1: Using `read_csv()` method: `read_csv()` is an important pandas function to read csv files and do operations on it.

Example:

The screenshot shows the PyCharm IDE interface. On the left, the project structure is visible with a file named 'dataframes.py' selected. The main editor window contains the following Python code:

```
1 # Python program to illustrate
2 # creating a data frame using CSV files
3 # import pandas module
4 import pandas as pd
5 # creating a data frame
6 df = pd.read_csv("C:\\\\Users\\\\sumedha\\\\OneDrive\\\\Desktop\\\\biodata.csv")
7 print(df.head())
```

Below the editor is the 'Run' tool window, which displays the output of the script. The output shows the first two rows of a DataFrame:

	Product	Age	Miles
TM195	18	Male	Single
	19	Female	Partnered

[2 rows x 7 columns]

Method #2: Using `read_table()` method: `read_table()` is another important pandas function to read csv files and create data frame from it.

Example:

The screenshot shows a Python development environment with a dark theme. The file tree on the left shows a project named 'basics' containing a '.venv' folder, a 'dataframes.py' script, and a 'main.py' script. The 'External Libraries' section lists 'pandas'. The code editor window contains the following Python code:

```
8
9
10 # Python program to illustrate
11 # creating a data frame using CSV files
12
13 # import pandas module
14 import pandas as pd
15
16 # creating a data frame
17 df = pd.read_table(filepath_or_buffer= "C:\\\\Users\\\\Sumedha\\\\OneDrive\\\\Desktop\\\\biostat.csv", delimiter=",")
18 print(df.head())
```

The run output window at the bottom shows the execution results:

```
[2 rows x 7 columns]
Product Age,Gender,Education,Marital Status,Usage Fitness,Income,Miles
0 TM195,18,Male,Single,TM195 Male Single,19,4,0...
1 TM195,19,Female,Partnered,TM195 Female Partner...
Process finished with exit code 0
```

Method #3: Using the csv module: One can directly import the csv files using the csv module and then create a data frame using that csv file.

Example:

The screenshot shows a Python development environment with a dark theme. The file tree on the left shows a project named 'basics' containing a '.venv' folder, a 'dataframes.py' script, and a 'main.py' script. The 'External Libraries' section lists 'pandas' and 'csv'. The code editor window contains the following Python code:

```
20 # Python program to illustrate
21 # creating a data frame using CSV files
22 # import pandas module
23 import pandas as pd
24 # import csv module
25 import csv
26 with open("C:\\\\Users\\\\Sumedha\\\\OneDrive\\\\Desktop\\\\biostat.csv") as csv_file:
27     # read the csv file
28     csv_reader = csv.reader(csv_file)
29
30     # now we can use this csv files into the pandas
31     df = pd.DataFrame(data=[csv_reader], index=None)
32     # iterating values of first column
33     for val in list(df[2]):
34         print(val)
```

output:

```
['TM195', '18', 'Male', 'Single', 'TM195 Male Single', '19', '4', '0', '1', '2', '3', 'Male', 'Male', '14', '15', '14', '12', '13', 'Single', 'F...
```

4)PERFORMING JOINS BETWEEN PANDA DATA FRAMES:

→ There are five types of Joins in pandas

- Inner Join
- Left Outer Join
- Right Outer Join
- Full Outer Join or simply Outer Join
- Index Join

To understand different types of joins, we will first make two DataFrames, namely **a** and **b**.

Dataframe a:

The screenshot shows the PyCharm IDE interface. The project structure on the left includes files like main.py, dataframes.py, and dataframewithjoins.py. The code editor window displays the following Python script:

```
1 # importing pandas
2 import pandas as pd
3 # Creating dataframe a
4 a = pd.DataFrame()
5 # Creating Dictionary
6 d = {'id': [1, 2, 10, 12],
7      'val1': ['a', 'b', 'c', 'd']}
8 a = pd.DataFrame(d)
9 # printing the dataframe
10 print(a)
11
12
```

The run output window at the bottom shows the resulting DataFrame:

	id	val1
0	1	a
1	2	b
2	10	c
3	12	d

Process finished with exit code 0

DataFrame b:

The screenshot shows the PyCharm IDE interface. The project structure on the left includes files like main.py, dataframes.py, and dataframewithjoins.py. The code editor window displays the following Python script:

```
11
12 # importing pandas
13 import pandas as pd
14
15 # Creating dataframe b
16 b = pd.DataFrame()
17
18 # Creating dictionary
19 d = {'id': [1, 2, 9, 8],
20      'val1': ['p', 'q', 'r', 's']}
21 b = pd.DataFrame(d)
22
23 # printing the dataframe
24 print(b)
25
26
27
```

The run output window at the bottom shows the resulting DataFrame:

	id	val1
0	1	p
1	2	q
2	9	r
3	8	s

Process finished with exit code 0

→Pandas Inner Join

Inner join is the most common type of join you'll be working with. It returns a Dataframe with only those rows that have common characteristics.

Example:

The screenshot shows a Jupyter Notebook environment with a file tree on the left and a code editor on the right. The code editor contains Python code for creating two dataframes, 'a' and 'b', and performing an inner join on them based on a common 'id' column. The output cell shows the resulting merged dataframe.

```
# importing pandas
import pandas as pd
# Creating dataframe a
a = pd.DataFrame()
# Creating Dictionary
d = {'id': [1, 2, 10, 12],
     'val1': ['a', 'b', 'c', 'd']}
a = pd.DataFrame(d)
# Creating dataframe b
b = pd.DataFrame()
# Creating dictionary
d = {'id': [1, 2, 9, 8],
     'val1': ['p', 'q', 'r', 's']}
b = pd.DataFrame(d)
# inner join
df = pd.merge(a, b, on='id', how='inner')
# display dataframe
print(df)
```

Run dataframewithjoins

	id	val1_x	val1_y
0	1	a	p
1	2	b	q

Process finished with exit code 0

→Pandas Left Join

With a left outer join, all the records from the first Dataframe will be displayed, irrespective of whether the keys in the first Dataframe can be found in the second Dataframe. Whereas, for the second Dataframe, only the records with the keys in the second Dataframe that can be found in the first Dataframe will be displayed.

Example:

The screenshot shows the PyCharm IDE interface. The top part displays the project structure with files: main.py, dataframes.py, and dataframewithjoins.py. The bottom part shows the run output for dataframewithjoins.py. The code in dataframewithjoins.py performs a left outer join between two dataframes, a and b, based on the 'id' key. The resulting DataFrame df is printed, showing four rows with columns id, val1_x, and val1_y. Rows 0, 1, and 2 have valid data, while row 3 has NaN values for both val1_x and val1_y because there is no corresponding entry in dataframe b for id 12.

```
43 # importing pandas
44 import pandas as pd
45 # Creating data frame a
46 a = pd.DataFrame()
47 # Creating Dictionary
48 d = {'id': [1, 2, 10, 12],
49       'val1': ['a', 'b', 'c', 'd']}
50 a = pd.DataFrame(d)
51 # Creating data frame b
52 b = pd.DataFrame()
53 # Creating dictionary
54 d = {'id': [1, 2, 9, 8],
55       'val1': ['p', 'q', 'r', 's']}
56 b = pd.DataFrame(d)
57 # left outer join
58 df = pd.merge(a, b, on='id', how='left')
59 # display data frame
60 print(df)
```

	id	val1_x	val1_y
0	1	a	p
1	2	b	q
2	10	c	NaN
3	12	d	NaN

→Pandas Right Outer Join

For a right join, all the records from the second Dataframe will be displayed. However, only the records with the keys in the first Dataframe that can be found in the second Dataframe will be displayed.

Example:

The screenshot shows a PyCharm IDE interface. On the left, there's a file tree with a file named 'dataframewithjoins.py'. The code in the editor is as follows:

```
# importing pandas
import pandas as pd
# Creating dataframe a
a = pd.DataFrame()
# Creating Dictionary
d = {'id': [1, 2, 10, 12],
     'val1': ['a', 'b', 'c', 'd']}
a = pd.DataFrame(d)
# Creating dataframe b
b = pd.DataFrame()
# Creating dictionary
d = {'id': [1, 2, 9, 8],
     'val1': ['p', 'q', 'r', 's']}
b = pd.DataFrame(d)
# right outer join
df = pd.merge(a, b, on='id', how='right')
# display dataframe
print(df)
```

Below the code, there's a 'Run' button and a terminal window showing the output:

```
Process finished with exit code 0
```

The terminal also displays the resulting DataFrame:

	id	val1_x	val1_y
0	1	a	p
1	2	b	q
2	9	NaN	r
3	8	NaN	s

→Pandas Full Outer Join

A full outer join returns all the rows from the left Dataframe, and all the rows from the right Dataframe, and matches up rows where possible, with NaNs elsewhere.

Example:

The screenshot shows a Jupyter Notebook interface with a code editor and a run output cell.

Code:

```
81 # importing pandas
82 import pandas as pd
83 # Creating dataframe a
84 a = pd.DataFrame()
85 # Creating Dictionary
86 d = {'id': [1, 2, 10, 12],
87       'val1': ['a', 'b', 'c', 'd']}
88 a = pd.DataFrame(d)
89 # Creating dataframe b
90 b = pd.DataFrame()
91 # Creating dictionary
92 d = {'id': [1, 2, 9, 8],
93       'val1': ['p', 'q', 'r', 's']}
94 b = pd.DataFrame(d)
95 # full outer join
96 df = pd.merge(a, b, on='id', how='outer')
97 # display dataframe
98 print(df)
```

Run Output:

```
   id  val1_x val1_y
0   1        a      p
1   2        b      q
2   8      NaN      s
3   9      NaN      r
4  10        c      NaN
5  12        d      NaN
```

→Pandas Index Join

To merge the Dataframe on indices pass

the *left_index* and *right_index* arguments as True i.e. both the Dataframes are merged on an index using default Inner Join.

Example:

The screenshot shows a Jupyter Notebook interface with a code editor and a run output cell.

Code:

```
100 # importing pandas
101 import pandas as pd
102 # Creating dataframe a
103 a = pd.DataFrame()
104 # Creating Dictionary
105 d = {'id': [1, 2, 10, 12],
106       'val1': ['a', 'b', 'c', 'd']}
107 a = pd.DataFrame(d)
108 # Creating dataframe b
109 b = pd.DataFrame()
110 # Creating dictionary
111 d = {'id': [1, 2, 9, 8],
112       'val1': ['p', 'q', 'r', 's']}
113 b = pd.DataFrame(d)
114 # index join
115 df = pd.merge(a, b, left_index=True, right_index=True)
116 # display dataframe
117 print(df)
```

Run Output:

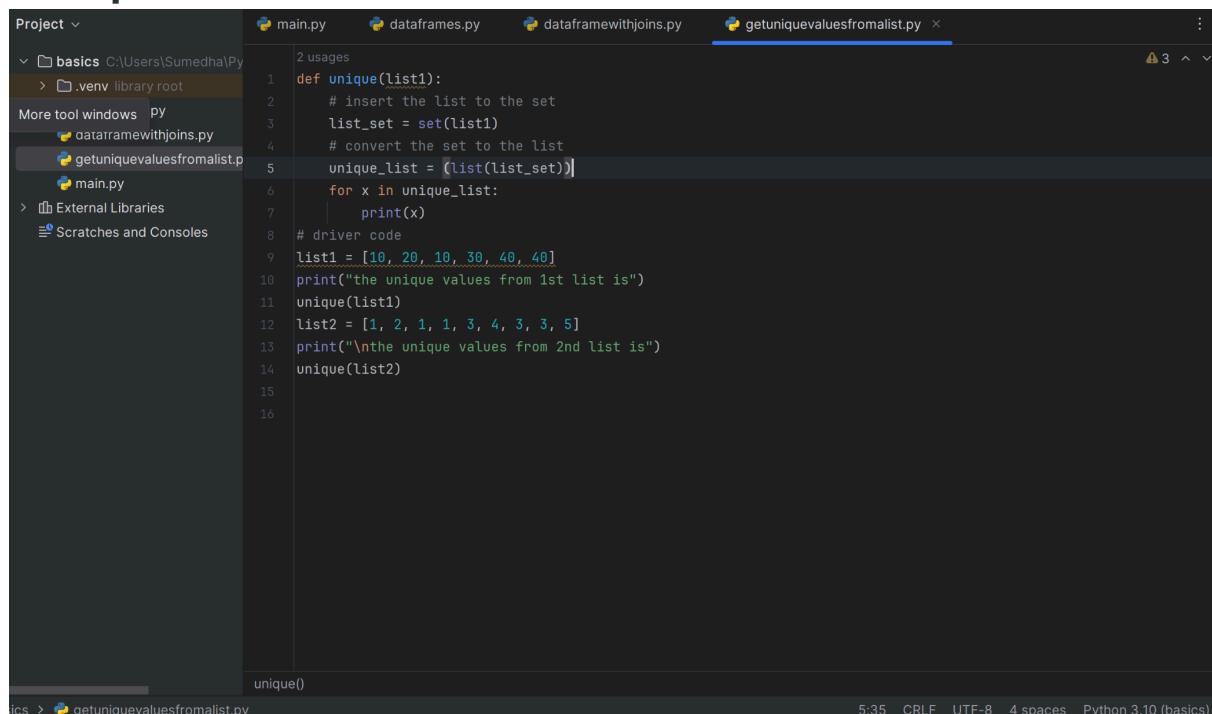
```
   id_x val1_x id_y val1_y
0     1      a     1      p
1     2      b     2      q
2    10      c     9      r
3    12      d     8      s
```

5)Get Unique Values from a List in Python:

Get Unique Values from a List Using Set Method

Using set() property of Python, we can easily check for the unique values. Insert the values of the list in a set. Set only stores a value once even if it is inserted more than once. After inserting all the values in the set by list_set=set(list1), convert this set to a list to print it.

Example:



The screenshot shows the PyCharm IDE interface. On the left, the project structure is visible with files like main.py, dataframes.py, dataframewithjoins.py, and getuniquevaluesfromalist.py. The getuniquevaluesfromalist.py file is open in the editor. The code defines a function unique() that takes a list as input, converts it to a set to remove duplicates, and then converts it back to a list before printing. It also prints the unique values of two lists, list1 and list2. The status bar at the bottom indicates the file is saved and shows the Python version as Python 3.10 (basics).

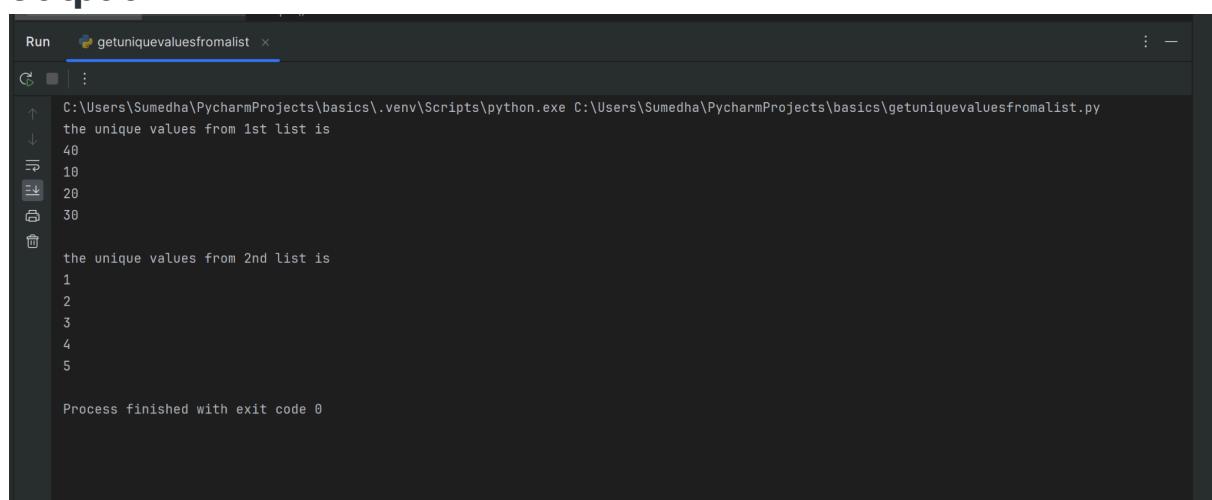
```
Project: basics C:\Users\Sumedha\PycharmProjects\basics\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\basics\getuniquevaluesfromalist.py

2 usages
1 def unique(list1):
2     # insert the list to the set
3     list_set = set(list1)
4     # convert the set to the list
5     unique_list = [list(list_set)]
6     for x in unique_list:
7         print(x)
8     # driver code
9     list1 = [10, 20, 10, 30, 40, 40]
10    print("the unique values from 1st list is")
11    unique(list1)
12    list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
13    print("\nthe unique values from 2nd list is")
14    unique(list2)

unique()

5:35 CRLF UTF-8 4 spaces Python 3.10 (basics)
```

Output:



The screenshot shows the PyCharm Run tab. The output window displays the execution of the getuniquevaluesfromalist.py script. It shows the unique values for both list1 and list2 as specified in the code. The process finished successfully with an exit code of 0.

```
Run getuniquevaluesfromalist x

C:\Users\Sumedha\PycharmProjects\basics\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\basics\getuniquevaluesfromalist.py
the unique values from 1st list is
40
10
20
30
the unique values from 2nd list is
1
2
3
4
5

Process finished with exit code 0
```

→ **Get Unique Values From a List in Python Using reduce() function:** Using Python import reduce() from functools and iterate over all element and checks if the element is a duplicate or unique value.

Example:

The screenshot shows a code editor interface with a sidebar showing project files: .venv, library root, dataframes.py, dataframewithjoins.py, getuniquevaluesfromalist.py, main.py, External Libraries, and Scratches and Consoles. The main pane contains the following Python code:

```
22
23     from functools import reduce
24
25     # Print directly by using * symbol
26     ans = reduce(lambda re, x: re + [x] if x not in re else re, list1, [])
27     print(ans)
28
29     # driver code
30     list1 = [10, 20, 10, 30, 40, 40]
31     print("the unique values from 1st list is")
32     unique(list1)
33     list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
34     print("\nthe unique values from 2nd list is")
35     unique(list2)
36
```

The Run tab shows the output of the code execution:

```
Run  getuniquevaluesfromalist ×
:
the unique values from 1st list is
[10, 20, 30, 40]

the unique values from 2nd list is
[1, 2, 3, 4, 5]

> Process finished with exit code 0
```

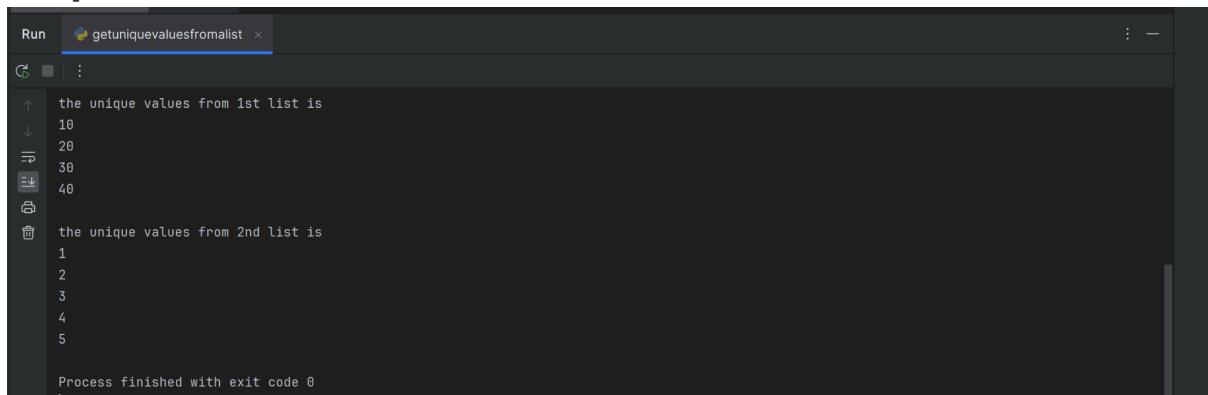
→ **Get Unique Values From a List in Python Using Operator.countOf() method:** The ‘unique’ function initializes an empty ‘unique_list’, then iterates through ‘list1’. For each element ‘x’, it employs ‘op.countof()’ to check if ‘x’ is present in ‘unique_list’. If not found (count is 0), ‘x’ is appended to ‘unique_list’. The final unique values are printed using a loop.

example:

The screenshot shows a code editor interface with a sidebar showing project files: basics, C:\Users\Sumedha\Py, .venv, library root, dataframes.py, dataframewithjoins.py, getuniquevaluesfromalist.py, main.py, External Libraries, and Scratches and Consoles. The main pane contains the following Python code:

```
35
36     import operator as op
37
38     # function to get unique values
39     def unique(list1):
40         # initialize a null list
41         unique_list = []
42
43         # traverse for all elements
44         for x in list1:
45             # check if exists in unique_list or not
46             if op.countOf(*args: unique_list, x) == 0:
47                 unique_list.append(x)
48
49         # print list
50         for x in unique_list:
51             print(x)
52
53     # driver code
54     list1 = [10, 20, 10, 30, 40, 40]
55     print("the unique values from 1st list is")
56     unique(list1)
57     list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
58     print("\nthe unique values from 2nd list is")
59     unique(list2)
```

Output:



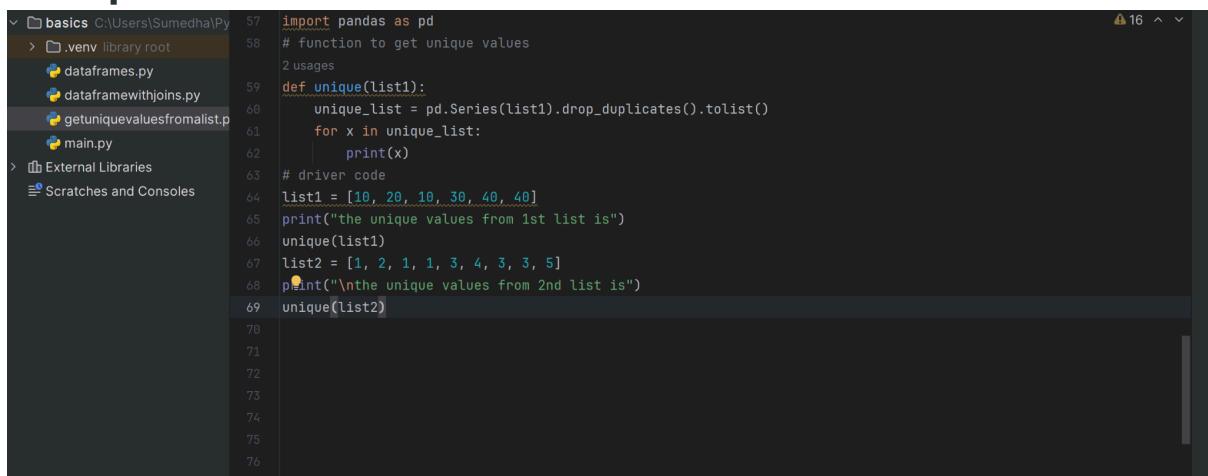
```
Run getuniquevaluesfromalist x
the unique values from 1st list is
10
20
30
40
the unique values from 2nd list is
1
2
3
4
5

Process finished with exit code 0
```

→Get Unique Values From a List in Python Using pandas

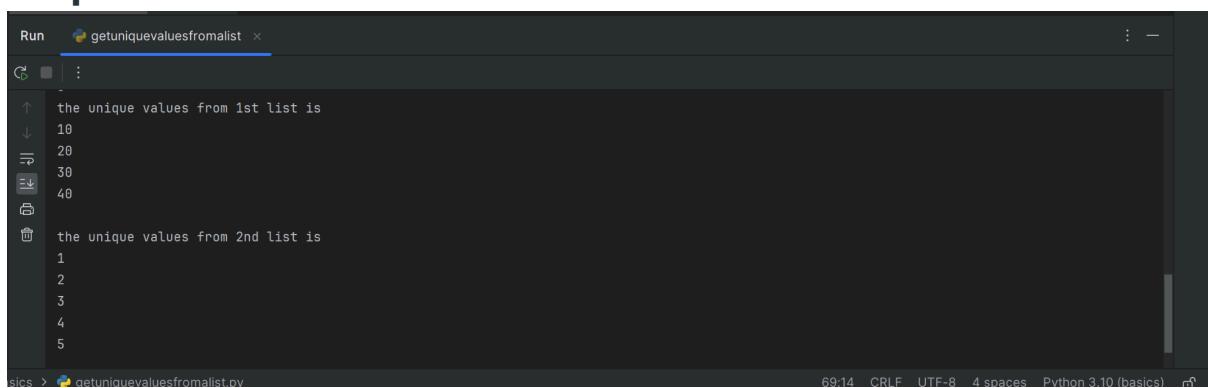
module: The ‘unique’ function utilize pandas() to create a Series from ‘list1’, then employs ‘drop_duplicates()’ to eliminate duplicates and obtain a list of unique values. Subsequently, it iterates through the unique list and prints each element. The driver code demonstrates the process for two lists, ‘list1’ and ‘list2’, providing distinct values for each list.

Example:



```
basics C:\Users\SumedhaPy 57 import pandas as pd
  .venv library root
    dataframes.py
    dataframewithjoins.py
    getuniquevaluesfromalist.py
    main.py
  External Libraries
  Scratches and Consoles
58 # function to get unique values
59 # 2 usages
60 def unique(list1):
61     unique_list = pd.Series(list1).drop_duplicates().tolist()
62     for x in unique_list:
63         print(x)
64     # driver code
65     list1 = [10, 20, 10, 30, 40, 40]
66     print("the unique values from 1st list is")
67     unique(list1)
68     list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
69     print("\nthe unique values from 2nd list is")
70     unique(list2)
```

Output:



```
Run getuniquevaluesfromalist x
the unique values from 1st list is
10
20
30
40
the unique values from 2nd list is
1
2
3
4
5
```

→Get Unique Values From a List Using numpy.unique:

Using Python's import numpy() the unique elements in the array are also obtained. In the first step convert the list to **x=numpy.array(list)** and then use numpy.unique(x) function to get the unique values from the list. **numpy.unique()** returns only the unique values in the list.

Example:

The screenshot shows a Jupyter Notebook interface. On the left, there is a file tree with files like 'dataframes.py', 'dataframewithjoins.py', 'getuniquevaluesfromalist.py' (which is the current file), and 'main.py'. The 'External Libraries' and 'Scratches and Consoles' sections are also visible. The main area contains Python code to find unique values in two lists using numpy.unique. The code is as follows:

```
71 # using numpy.unique
72 import numpy as np
73 2 usages
74 def unique(list1):
75     x = np.array(list1)
76     print(np.unique(x))
77 # driver code
78 list1 = [10, 20, 10, 30, 40, 40]
79 print("the unique values from 1st list is")
80 unique(list1)
81 list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
82 print("\nthe unique values from 2nd list is")
83 unique(list2)
```

The output section shows the results of running the code. It prints '5' followed by 'the unique values from 1st list is [10 20 30 40]' and 'the unique values from 2nd list is [1 2 3 4 5]'. At the bottom, it says 'Process finished with exit code 0'.

→Get Unique Values From a List in Python Using collections.Counter():

Using Python to import Counter() from collections() print all the keys of Counter elements or we print directly by using the “*” symbol. Below is the implementation of the above approach.

Example:

The screenshot shows a Python code editor interface. The project structure on the left includes files like main.py, dataframes.py, dataframewithjoins.py, and getuniquevaluesfromalist.py. The code in getuniquevaluesfromalist.py uses the Counter class from the collections module to find unique values in two lists. The output window at the bottom shows the execution results.

```
84     from collections import Counter
85     # Function to get unique values
86     2 usages
87     def unique(list1):
88         # Print directly by using * symbol
89         print(*Counter(list1))
90     # driver code
91     list1 = [10, 20, 10, 30, 40, 40]
92     print("the unique values from 1st list is")
93     unique(list1)
94     list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
95     print("\nthe unique values from 2nd list is")
96     unique(list2)
97
98
```

Output:

```
[1 2 3 4 5]
the unique values from 1st list is
10 20 30 40

the unique values from 2nd list is
1 2 3 4 5

Process finished with exit code 0
```

→Get Unique Values From a List Using dict.fromkeys():

Using the fromkeys() method of dictionary data structure we can fetch the unique elements. Firstly we need to define a list that consists of duplicate elements. Then we need to use a variable in which we will store the result after using the fromkeys() method. We need to convert that result into a list, as the fromkeys() method is part of the dictionary so by default it returns a dictionary with all the unique keys and None as their values.

Example:

The screenshot shows a PyCharm IDE interface. The top navigation bar includes 'File', 'Edit', 'Run', 'View', 'Code', 'Refactor', 'Tools', 'Help', and a 'Project' dropdown. Below the navigation bar is a toolbar with icons for file operations like 'New', 'Open', 'Save', 'Run', 'Stop', and 'Run Configuration'. The main area is divided into several panes: a 'Project' pane on the left showing a file structure with files like 'main.py', 'dataframes.py', 'dataframewithjoins.py', and 'getuniquevaluesfromalist.py'; a 'Code' editor pane containing the code for 'getuniquevaluesfromalist.py'; a 'Run' pane at the bottom showing the execution results; and a 'Terminal' pane on the far right.

```
# defining a list which consists duplicate values
list1 = [10, 20, 10, 30, 40, 40]
list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
# storing the result of the fromkeys()
# operation and converting it into list
unique_list_1 = list(dict.fromkeys(list1))
unique_list_2 = list(dict.fromkeys(list2))
# Printing the final result
print(unique_list_1, unique_list_2, sep="\n")
```

Run `getuniquevaluesfromalist`

[10, 20, 30, 40]
[1, 2, 3, 4, 5]

Process finished with exit code 0

6)GET COUNT BY STATUS USING PANDAS DATAFRAME APIs:

To count values in Pandas dataframe. First, we will create a data frame, and then we will count the values of different attributes.

Syntax: DataFrame.count(axis=0, level=None, numeric_only=False)

Example:

Count Values in Pandas Dataframe

Step 1: Importing libraries.

Step 2: Creating Dataframe.

The screenshot shows the PyCharm IDE interface. The top navigation bar includes 'File', 'Edit', 'Run', 'View', 'Code', 'Tools', 'Help', and a 'PyCharm' icon. Below the navigation bar is a toolbar with icons for file operations like 'New', 'Open', 'Save', 'Run', and 'Stop'. The main area is divided into two panes: the 'Project' pane on the left and the 'Run' pane on the right.

Project Pane: Shows a project structure for a 'basics' folder containing files: 'countstatusbypandas.py', 'dataframes.py', 'dataframewithjoins.py', 'getuniquevaluesfromalist.py', and 'main.py'. It also lists 'External Libraries' and 'Scratches and Consoles'.

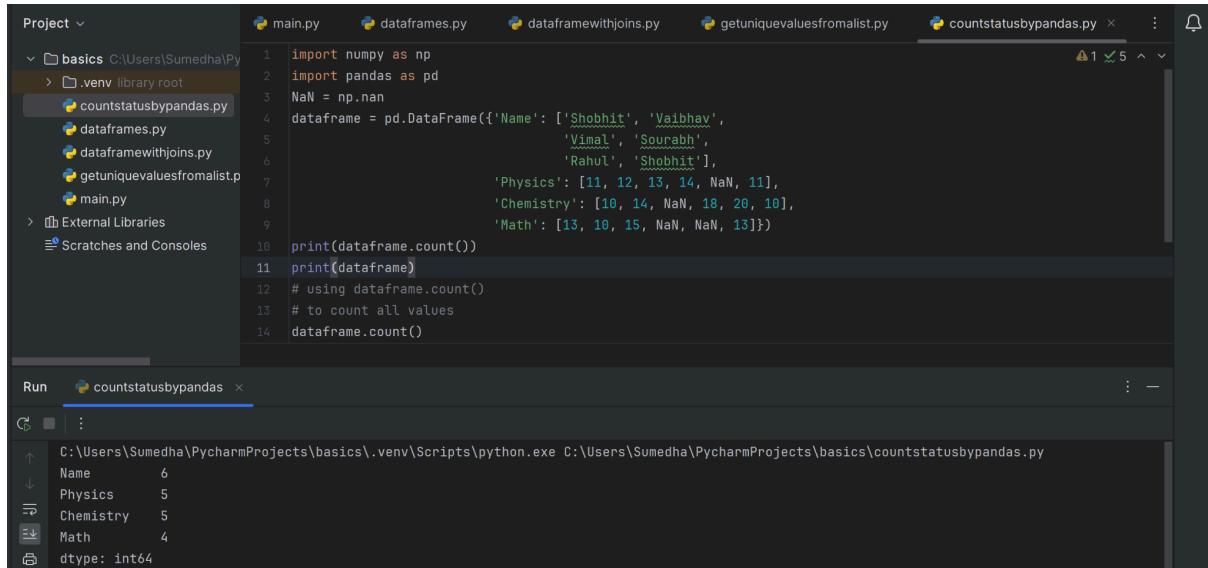
Run Pane: Displays the output of running 'countstatusbypandas.py'. The command 'C:\Users\Sumedha\PycharmProjects\basics\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\basics\countstatusbypandas.py' is shown at the top. The output shows the count of values for each column:

Column	Count
Name	6
Physics	5
Chemistry	5
Math	4

Below this, the DataFrame is printed:

```
Name    Physics    Chemistry    Math
0   Shobhit     11.0      10.0    13.0
1   Vaibhav     12.0      14.0    10.0
2   Vimal        13.0      NaN     15.0
3   Sourabh     14.0      18.0      NaN
4   Rahul         NaN      20.0      NaN
5   Shobhit     11.0      10.0    13.0
```

Step 3: In this step, we just simply use the `.count()` function to count all the values of different columns.



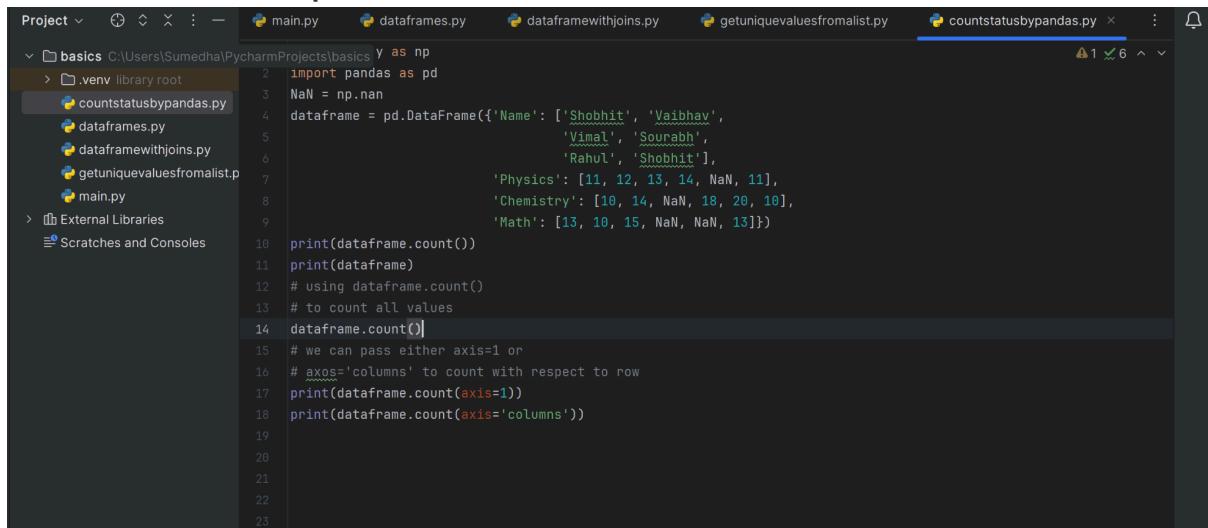
The screenshot shows a PyCharm project window. The code editor contains a Python script named `countstatusbypandas.py` with the following content:

```
1 import numpy as np
2 import pandas as pd
3 NaN = np.nan
4
5 dataFrame = pd.DataFrame({'Name': ['Shobhit', 'Vaibhav',
6                                     'Vimal', 'Sourabh',
7                                     'Rahul', 'Shobhit'],
8                           'Physics': [11, 12, 13, 14, NaN, 11],
9                           'Chemistry': [10, 14, NaN, 18, 20, 10],
10                          'Math': [13, 10, 15, NaN, NaN, 13]})
11
12 print(dataframe.count())
13 # using dataframe.count()
14 # to count all values
15
16
17
18
19
20
21
22
23
```

The run output shows the counts for each column:

```
C:\Users\Sumedha\PycharmProjects\basics\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\basics\countstatusbypandas.py
Name      6
Physics    5
Chemistry  5
Math       4
dtype: int64
```

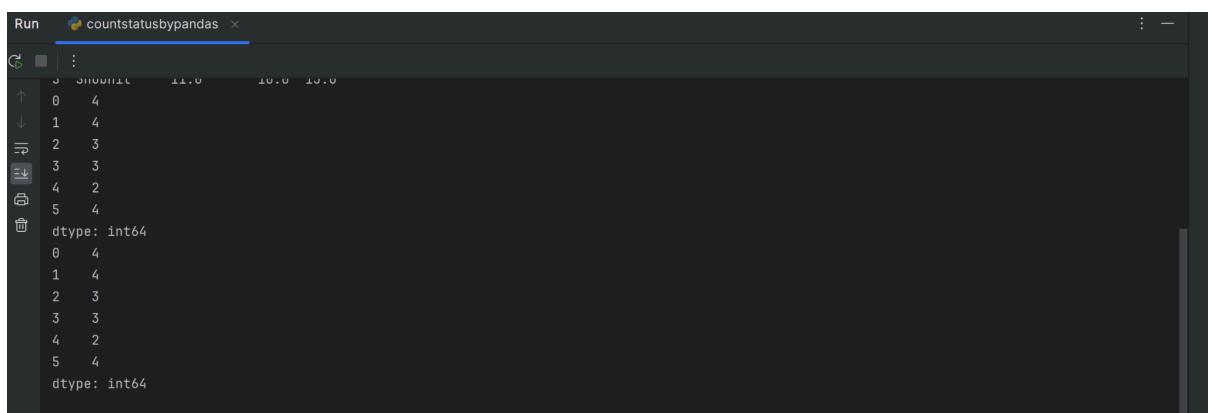
Step 4: If we want to count all the values with respect to row then we have to pass `axis=1` or 'columns'.



The screenshot shows a PyCharm project window. The code editor contains a Python script named `countstatusbypandas.py` with the following content:

```
1 import numpy as np
2 import pandas as pd
3 NaN = np.nan
4
5 dataFrame = pd.DataFrame({'Name': ['Shobhit', 'Vaibhav',
6                                     'Vimal', 'Sourabh',
7                                     'Rahul', 'Shobhit'],
8                           'Physics': [11, 12, 13, 14, NaN, 11],
9                           'Chemistry': [10, 14, NaN, 18, 20, 10],
10                          'Math': [13, 10, 15, NaN, NaN, 13]})
11
12 print(dataframe.count())
13 # using dataframe.count()
14 # to count all values
15
16 # we can pass either axis=1 or
17 # axis='columns' to count with respect to row
18 print(dataframe.count(axis=1))
19
20
21
22
23
```

OUTPUT:



The screenshot shows a PyCharm run output window. The output shows the counts for each row (index 0 to 5) across the columns:

```
Run  countstatusbypandas
C:\Users\Sumedha\PycharmProjects\basics\.venv\Scripts\python.exe C:\Users\Sumedha\PycharmProjects\basics\countstatusbypandas.py
0    4
1    4
2    3
3    3
4    2
5    4
dtype: int64
0    4
1    4
2    3
3    3
4    2
5    4
dtype: int64
```

Step 5: Now if we want to count null values in our dataframe.

The screenshot shows a PyCharm project window. The left sidebar displays a file tree with files like main.py, dataframes.py, dataframewithjoins.py, getuniquevaluesfromalist.py, and countstatusbypandas.py. The main editor area contains Python code for creating a DataFrame and counting null values:

```
1 import numpy as np
2 import pandas as pd
3 NaN = np.nan
4
5 # Create DataFrame
6 # ...
7
8 # Counting null values
9 print(dataframe.count())
10 print(dataframe)
11
12 # Using .count() to count all values
13 # To count all values
14 dataframe.count()
15
16 # We can pass either axis=1 or
17 # axis='columns' to count with respect to row
18 print(dataframe.count(axis=1))
19
20 # It will give the count
21 # Of individual columns count of null values
22 print(dataframe.isnull().sum())
23
24 # It will give the total null
25 # Values present in our DataFrame
26 print("Total Null values count: ",
27      dataframe.isnull().sum().sum())
28
```

OUTPUT:

The screenshot shows the PyCharm run console output. It displays the DataFrame structure and the total count of null values:

```
Name      0
Physics   1
Chemistry 1
Math      2
dtype: int64
Total Null values count: 4
Process finished with exit code 0
```

Step 6: Some examples to use .count()

Now we want to count no of students whose physics marks are greater than 11.

```
# Count of student with greater
# than 11 marks in physics
print("Count of students with physics marks greater than 11 is->",
      dataframe[dataframe['Physics'] > 11]['Name'].count())

# Resultant of above DataFrame
dataframe[dataframe['Physics'] > 11]
```

OUTPUT:

```
:  
Total Null values count: 4  
Count of students with physics marks greater than 11 is-> 3  
     Name  Physics  Chemistry  Math  
1  Vaibhav      12.0       14.0   10.0  
2    Vimal      13.0        NaN   15.0  
3 Sourabh      14.0       18.0    NaN  
Process finished with exit code 0
```