

Name : Parth Nandedkar

Date : 01 Feb 2024

Topics : Python (Numpy, Pandas, Dataframe APIs, Dataframe using dynamic column list)

Batch : Data Engineering Batch-1

Numpy :

NumPy, which stands for Numerical Python, is a powerful library in Python used for numerical and scientific computing. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. Here's an overview of NumPy:

N-dimensional Arrays (ndarray):

NumPy's main object is the ndarray, which is a multidimensional array of elements of the same type.

Arrays can be one-dimensional, two-dimensional, or even higher-dimensional.

Array Creation:

Arrays in NumPy can be created using various functions like `numpy.array()`, `numpy.zeros()`, `numpy.ones()`, `numpy.arange()`, etc.

Array Operations:

NumPy provides a wide range of mathematical operations that can be performed directly on arrays, including element-wise operations, linear algebra operations, statistical operations, and more.

Universal Functions (ufunc):

NumPy functions that operate element-wise on arrays are called universal functions (ufunc).

These functions are highly optimized and efficiently handle large datasets.

Broadcasting:

NumPy supports broadcasting, which allows operations between arrays of different shapes and sizes. The smaller array is broadcasted across the larger array to perform the operation.

Indexing and Slicing:

NumPy provides powerful indexing and slicing capabilities for accessing and manipulating elements of arrays.

Linear Algebra:

NumPy includes a linear algebra module (`numpy.linalg`) that provides functions for matrix operations, eigenvalue problems, solving linear equations, and more.

Random Module:

The `numpy.random` module provides functions for generating random numbers and distributions.

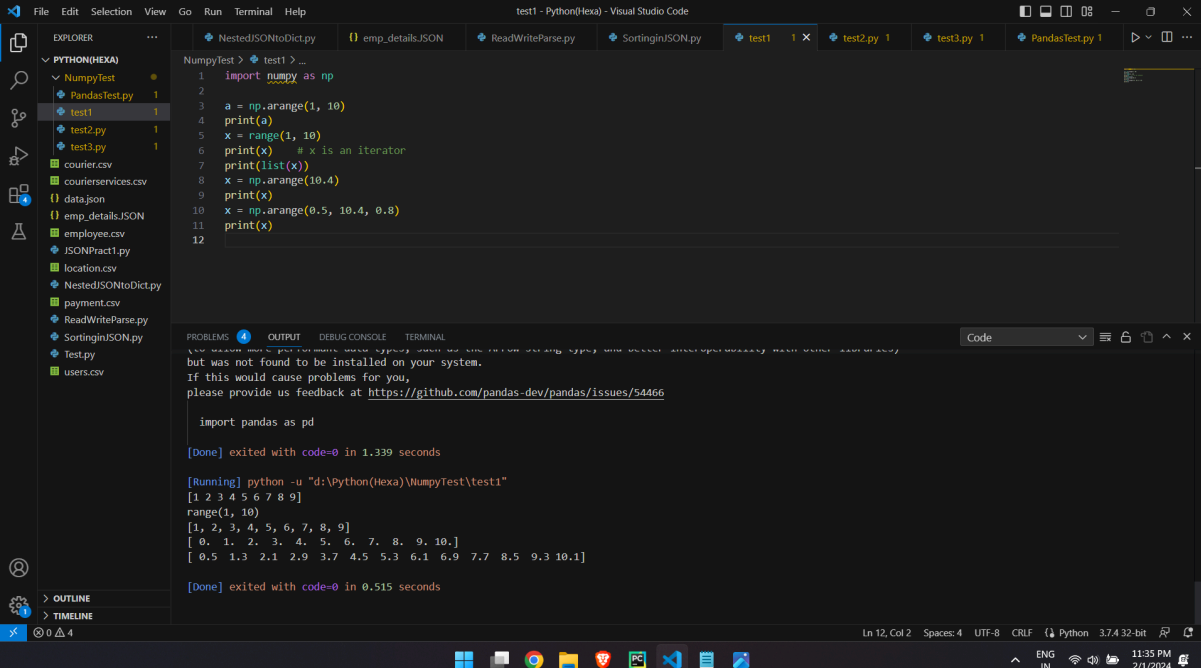
Performance:

NumPy is designed for efficiency and performance. Many of its operations are implemented in C and Fortran, making them faster than equivalent operations in pure Python.

Interoperability:

NumPy arrays can be seamlessly integrated with other libraries written in languages like C and Fortran, making it a foundational component in the scientific Python ecosystem.

Example of creating NumPy array using `arrange()` and `range()` methods:



```
1 import numpy as np
2
3 a = np.arange(1, 10)
4 print(a)
5 x = range(1, 10)
6 print(x) # x is an iterator
7 print(list(x))
8 x = np.arange(10.4)
9 print(x)
10 x = np.arange(0.5, 10.4, 0.8)
11 print(x)
12
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL

but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at <https://github.com/pandas-dev/pandas/issues/54466>

```
import pandas as pd

[Done] exited with code=0 in 1.339 seconds

[Running] python -u "d:\Python(Hexa)\NumpyTest\test1"
[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]

[Done] exited with code=0 in 0.515 seconds
```

Examples on spacing using linspace():

The screenshot shows a Visual Studio Code editor with a Python file named `test2.py`. The code in the editor is:

```
1 import numpy as np
2 print(np.linspace(1,10,7))
3 print(np.linspace(1, 10, 7, endpoint=False,retstep=False))
```

The terminal output shows the execution of the script:

```
[Running] python -u "d:\Python(Hexa)\NumpyTest\test1"
[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]

[Done] exited with code=0 in 1.339 seconds

[Done] exited with code=0 in 0.515 seconds
```

The screenshot shows a Visual Studio Code editor with a Python file named `test3.py`. The code in the editor is:

```
1 import numpy as np
2
3
4 samples, spacing = np.linspace(1, 10, retstep=True)
5 print(spacing)
6 samples, spacing = np.linspace(1, 10, 20, endpoint=True, retstep=True)
7 print(spacing)
8 samples, spacing = np.linspace(1, 10, 20, endpoint=False, retstep=True)
9 print(spacing)
```

The terminal output shows the execution of the script:

```
[Running] python -u "d:\Python(Hexa)\NumpyTest\test1"
[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]

[Done] exited with code=0 in 0.515 seconds

[Running] python -u "d:\Python(Hexa)\NumpyTest\test3.py"
0.1836734693877551
0.47368421052631576
0.45

[Done] exited with code=0 in 0.215 seconds
```

Pandas :

Pandas is a popular open-source data manipulation and analysis library for Python. It provides easy-to-use data structures such as Series and DataFrame, along with a variety of functions to perform efficient data manipulation, cleaning, and analysis. Here's a brief introduction to Pandas:

DataFrame:

The central data structure in Pandas is the DataFrame, which is a two-dimensional, labeled data structure with columns that can be of different data types.

It can be thought of as a table or spreadsheet.

Series:

A Series is a one-dimensional labeled array and is often used to represent a single column or row in a DataFrame.

Data Cleaning and Preparation:

Pandas provides functions for handling missing data, removing duplicates, and reshaping data for analysis.

Data Indexing and Selection:

Pandas offers powerful indexing and selection capabilities, allowing for easy retrieval and manipulation of data.

Data Alignment and Merging:

Data alignment is intrinsic to Pandas, and it automatically aligns data based on label indices.

The library provides functions for merging and joining datasets.

GroupBy:

Pandas supports the "split-apply-combine" paradigm with the groupby operation, enabling efficient grouping and aggregation of data.

Time Series and Date Functionality:

Pandas provides robust support for time-based data, including date range generation, frequency conversion, and resampling.

Input and Output (I/O):

Pandas supports reading and writing data in various formats, including CSV, Excel, SQL databases, and more.

Statistical Analysis:

Pandas includes a range of statistical functions for descriptive statistics, correlation, and other summary measures.

Integration with NumPy:

Pandas is built on top of NumPy, making it seamlessly integrated with other scientific computing libraries in the Python ecosystem.

Count Values in Pandas Dataframe :

```
10 'Math': [13, 10, 15, NaN, NaN, 13])
11
12 print(dataframe.count())
13 print(dataframe)
14
15 print(dataframe.count())
16
17 print("Count of students with physics marks greater than 11 is->",
18       dataframe[dataframe['Physics'] > 11]['Name'].count())
19
20 # resultant of above dataframe
21 dataframe[dataframe['Physics'] > 11]
```

Run: CountingValuesInPandasDataframe

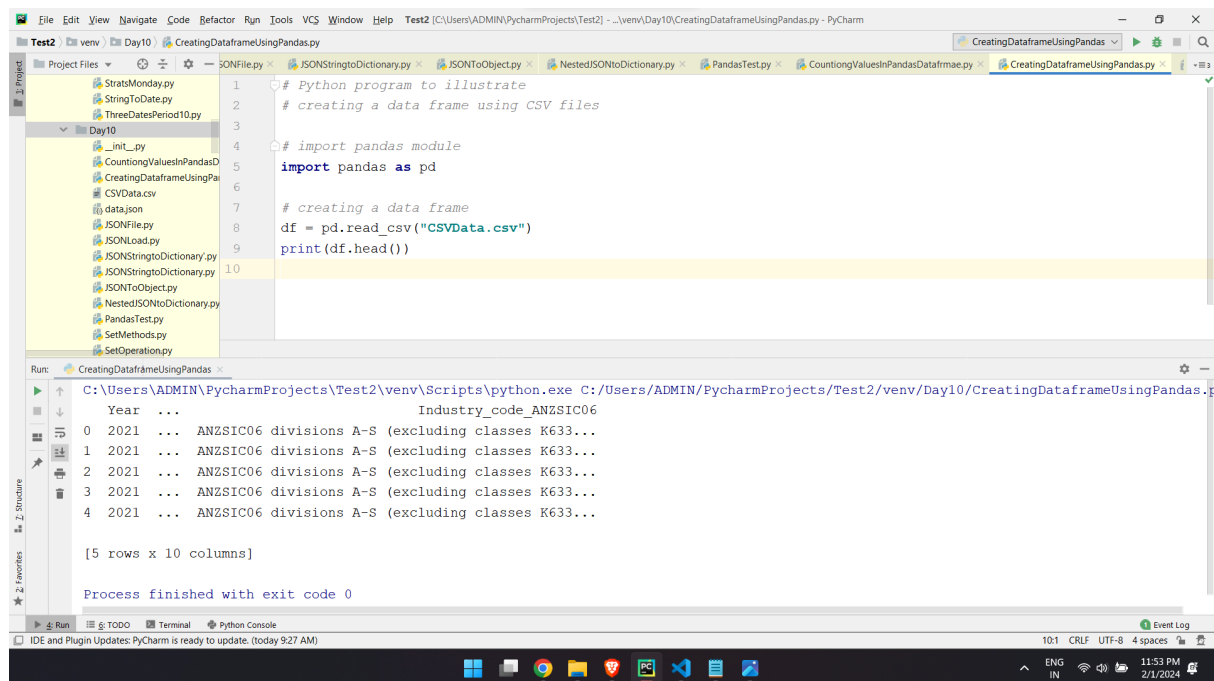
	Name	Physics	Chemistry	Math
1	Vaibhav	12.0	14.0	10.0
2	Vimal	13.0	NaN	15.0
3	Sourabh	14.0	18.0	NaN
4	Rahul	NaN	20.0	NaN
5	Shobhit	11.0	10.0	13.0

dtypes: int64

Count of students with physics marks greater than 11 is-> 3

Dataframes using dynamic column list on CSV Data :

Pandas can be used to convert different types of big data into python objects according to need so we can it to convert CSV file into Pandas object :



The screenshot shows the PyCharm IDE with a project named 'Test2'. The file explorer on the left shows a directory structure with files like 'StratsMonday.py', 'StringToDate.py', 'ThreeDatesPeriod10.py', and a 'Day10' folder containing various Python files. The main editor window shows the file 'CreatingDataframeUsingPandas.py' with the following code:

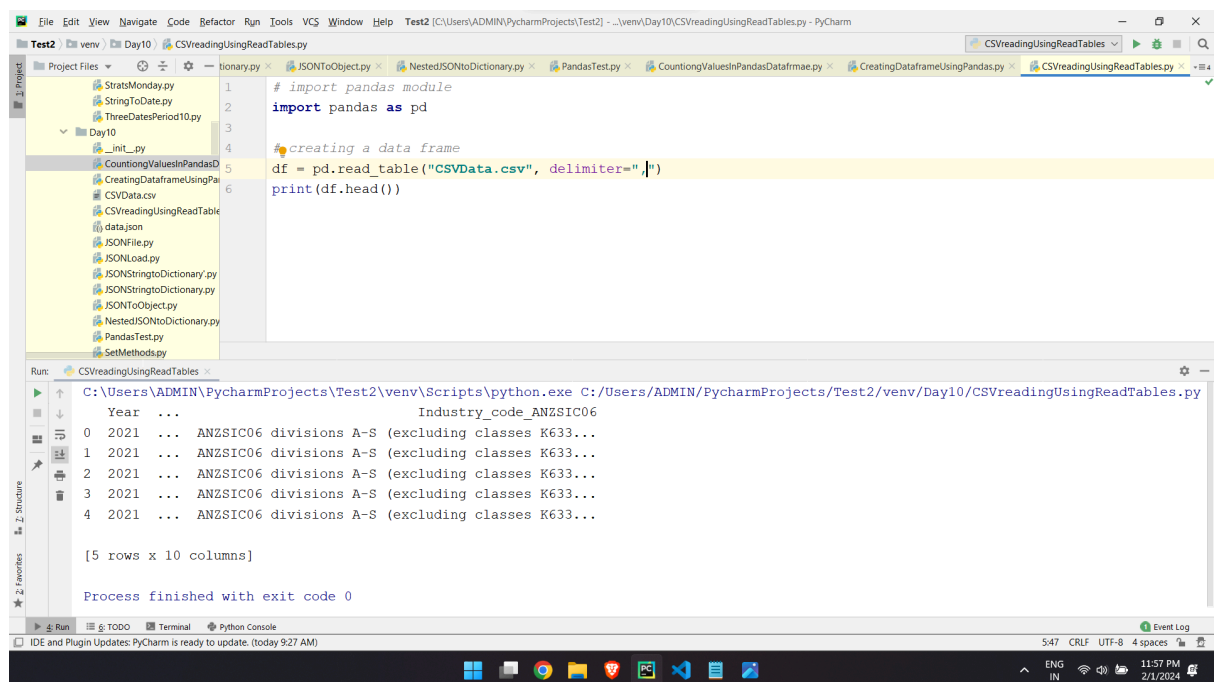
```
1 # Python program to illustrate
2 # creating a data frame using CSV files
3
4 # import pandas module
5 import pandas as pd
6
7 # creating a data frame
8 df = pd.read_csv("CSVData.csv")
9 print(df.head())
10
```

The Run window at the bottom shows the output of the program, which is a Pandas DataFrame with 5 rows and 10 columns. The data is as follows:

	Year	...	Industry_code	ANZSIC06
0	2021	...	ANZSIC06 divisions A-S (excluding classes K633...	
1	2021	...	ANZSIC06 divisions A-S (excluding classes K633...	
2	2021	...	ANZSIC06 divisions A-S (excluding classes K633...	
3	2021	...	ANZSIC06 divisions A-S (excluding classes K633...	
4	2021	...	ANZSIC06 divisions A-S (excluding classes K633...	

The Run window also indicates that the process finished with exit code 0.

Using Readtable() :



The screenshot shows the PyCharm IDE with a project named 'Test2'. The file explorer on the left shows a directory structure with files like 'StratsMonday.py', 'StringToDate.py', 'ThreeDatesPeriod10.py', and a 'Day10' folder containing various Python files. The main editor window shows the file 'CSVreadingUsingReadTables.py' with the following code:

```
1 # import pandas module
2 import pandas as pd
3
4 # creating a data frame
5 df = pd.read_table("CSVData.csv", delimiter="|")
6 print(df.head())
```

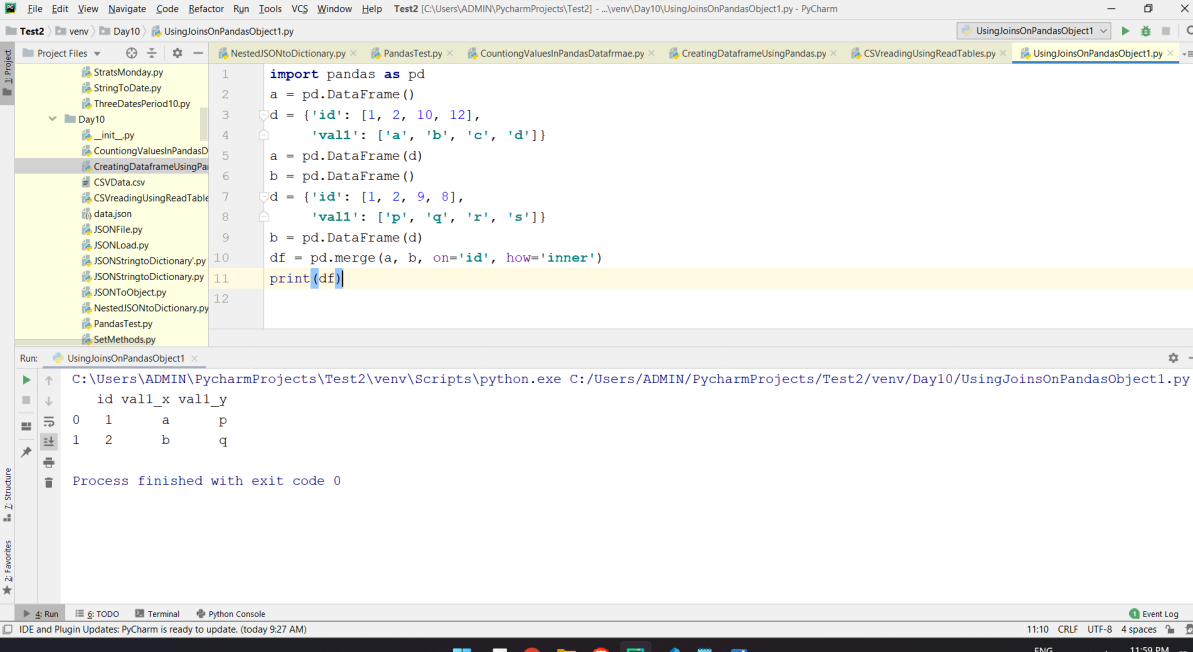
The Run window at the bottom shows the output of the program, which is a Pandas DataFrame with 5 rows and 10 columns. The data is as follows:

	Year	...	Industry_code	ANZSIC06
0	2021	...	ANZSIC06 divisions A-S (excluding classes K633...	
1	2021	...	ANZSIC06 divisions A-S (excluding classes K633...	
2	2021	...	ANZSIC06 divisions A-S (excluding classes K633...	
3	2021	...	ANZSIC06 divisions A-S (excluding classes K633...	
4	2021	...	ANZSIC06 divisions A-S (excluding classes K633...	

The Run window also indicates that the process finished with exit code 0.

Using JOINS on Dataframes :

INNER JOIN :



The screenshot shows a PyCharm IDE with a project named 'Test2'. The file explorer on the left shows a directory structure with files like 'StratsMonday.py', 'StringToDate.py', 'ThreeDatesPeriod10.py', and 'Day10'. The main editor window displays a Python script 'UsingJoinsOnPandasObject1.py' with the following code:

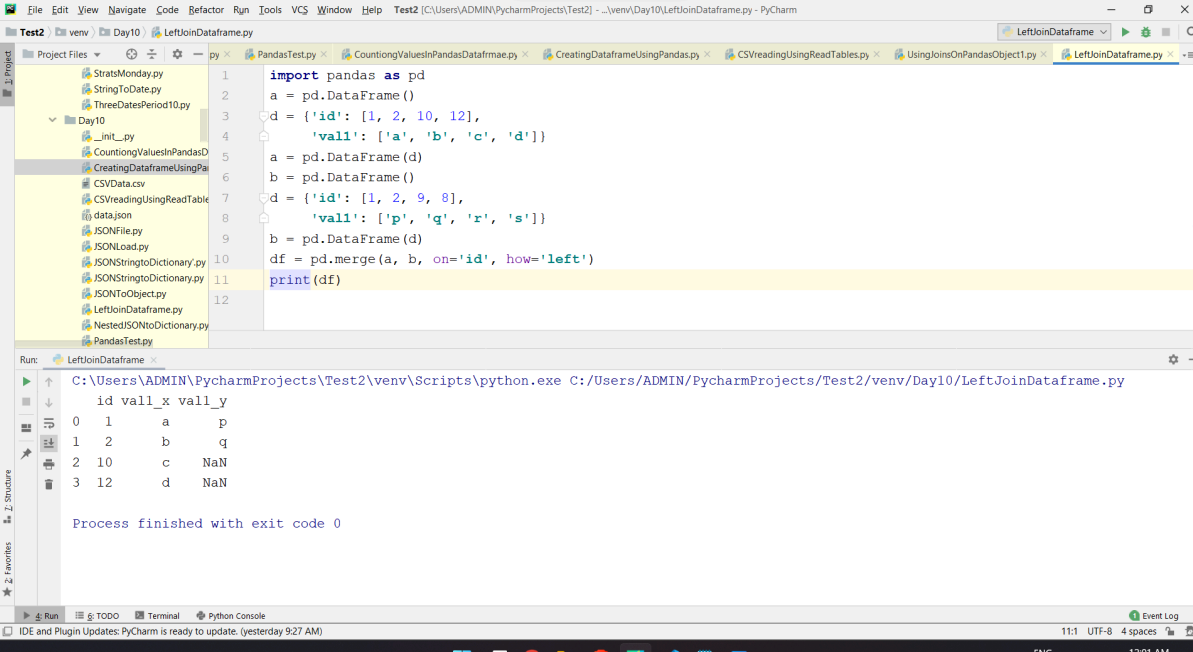
```
1 import pandas as pd
2 a = pd.DataFrame()
3 d = {'id': [1, 2, 10, 12],
4       'vall1': ['a', 'b', 'c', 'd']}
5 a = pd.DataFrame(d)
6 b = pd.DataFrame()
7 d = {'id': [1, 2, 9, 8],
8       'vall1': ['p', 'q', 'r', 's']}
9 b = pd.DataFrame(d)
10 df = pd.merge(a, b, on='id', how='inner')
11 print(df)
12
```

The Run console at the bottom shows the output of the script:

```
Run: UsingJoinsOnPandasObject1.py
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/Day10/UsingJoinsOnPandasObject1.py
id vall_x vall_y
0 1 a p
1 2 b q
Process finished with exit code 0
```

The status bar at the bottom indicates the IDE and Plugin Updates: PyCharm is ready to update. (today 9:27 AM).

LEFT JOIN :



The screenshot shows a PyCharm IDE with a project named 'Test2'. The file explorer on the left shows a directory structure with files like 'StratsMonday.py', 'StringToDate.py', 'ThreeDatesPeriod10.py', and 'Day10'. The main editor window displays a Python script 'LeftJoinDataFrame.py' with the following code:

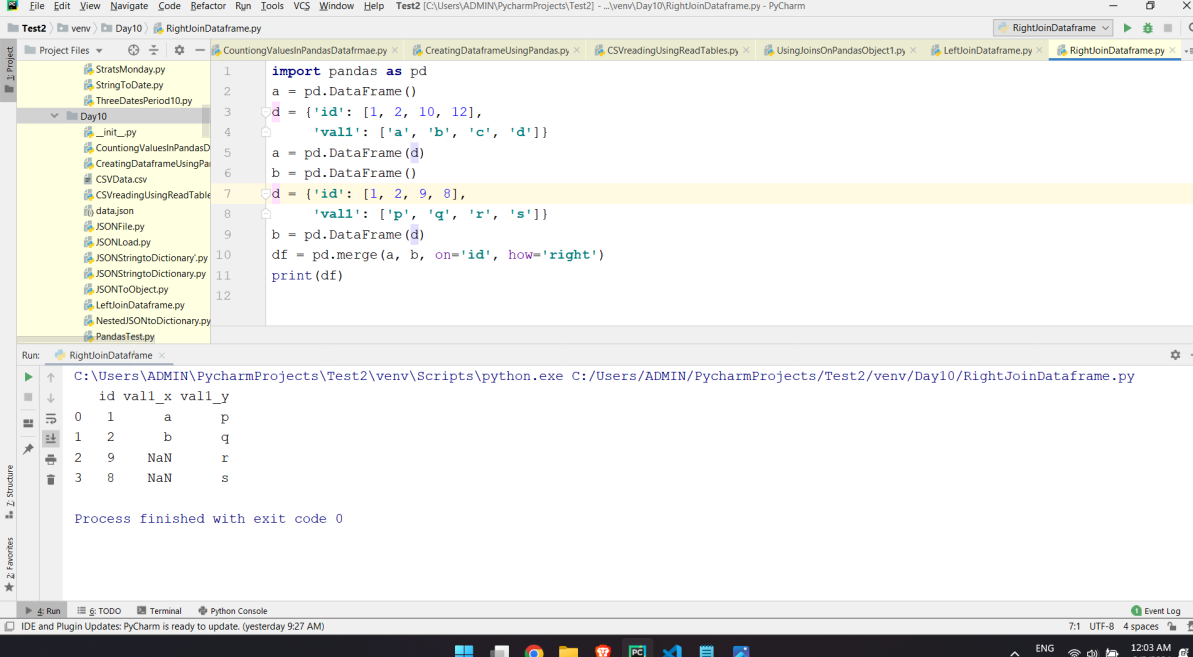
```
1 import pandas as pd
2 a = pd.DataFrame()
3 d = {'id': [1, 2, 10, 12],
4       'vall1': ['a', 'b', 'c', 'd']}
5 a = pd.DataFrame(d)
6 b = pd.DataFrame()
7 d = {'id': [1, 2, 9, 8],
8       'vall1': ['p', 'q', 'r', 's']}
9 b = pd.DataFrame(d)
10 df = pd.merge(a, b, on='id', how='left')
11 print(df)
12
```

The Run console at the bottom shows the output of the script:

```
Run: LeftJoinDataFrame.py
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/Day10/LeftJoinDataFrame.py
id vall_x vall_y
0 1 a p
1 2 b q
2 10 c NaN
3 12 d NaN
Process finished with exit code 0
```

The status bar at the bottom indicates the IDE and Plugin Updates: PyCharm is ready to update. (yesterday 9:27 AM).

RIGHT JOIN :



The screenshot shows a PyCharm IDE window with a Python script named `RightJoinDataFrame.py`. The script uses `pandas` to create two DataFrames, `a` and `b`, and performs a right join using `pd.merge(a, b, on='id', how='right')`. The output is displayed in the Run console.

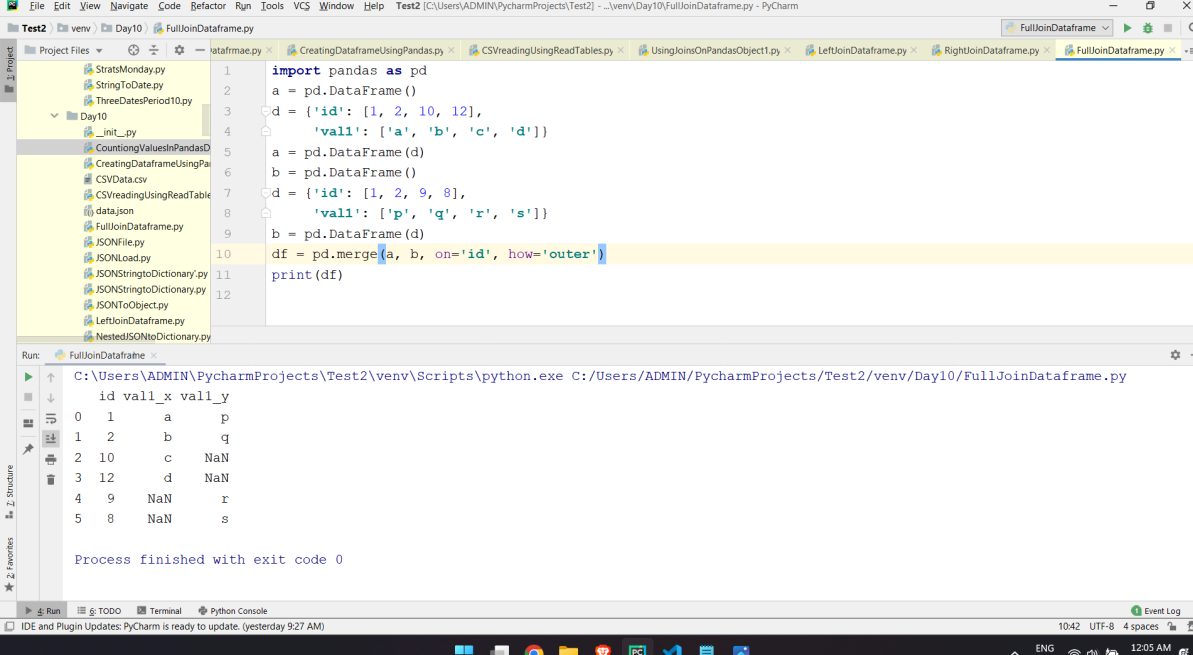
```
1 import pandas as pd
2 a = pd.DataFrame()
3 d = {'id': [1, 2, 10, 12],
4       'vall': ['a', 'b', 'c', 'd']}
5 a = pd.DataFrame(d)
6 b = pd.DataFrame()
7 d = {'id': [1, 2, 9, 8],
8       'vall': ['p', 'q', 'r', 's']}
9 b = pd.DataFrame(d)
10 df = pd.merge(a, b, on='id', how='right')
11 print(df)
12
```

Run: RightJoinDataFrame

```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/Day10/RightJoinDataFrame.py
id vall_x vall_y
0 1 a p
1 2 b q
2 9 NaN r
3 8 NaN s
```

Process finished with exit code 0

FULL JOIN :



The screenshot shows a PyCharm IDE window with a Python script named `FullJoinDataFrame.py`. The script uses `pandas` to create two DataFrames, `a` and `b`, and performs a full join using `pd.merge(a, b, on='id', how='outer')`. The output is displayed in the Run console.

```
1 import pandas as pd
2 a = pd.DataFrame()
3 d = {'id': [1, 2, 10, 12],
4       'vall': ['a', 'b', 'c', 'd']}
5 a = pd.DataFrame(d)
6 b = pd.DataFrame()
7 d = {'id': [1, 2, 9, 8],
8       'vall': ['p', 'q', 'r', 's']}
9 b = pd.DataFrame(d)
10 df = pd.merge(a, b, on='id', how='outer')
11 print(df)
12
```

Run: FullJoinDataFrame

```
C:\Users\ADMIN\PycharmProjects\Test2\venv\Scripts\python.exe C:/Users/ADMIN/PycharmProjects/Test2/venv/Day10/FullJoinDataFrame.py
id vall_x vall_y
0 1 a p
1 2 b q
2 10 c NaN
3 12 d NaN
4 9 NaN r
5 8 NaN s
```

Process finished with exit code 0

