

Introduction to Merging and Joining Datasets

Merging and joining datasets are crucial techniques in data analysis that allow you to integrate information from multiple sources. These operations enable analysts and data scientists to create richer datasets that facilitate deeper insights and more comprehensive analyses.

We We can join, merge, and concat dataframe using different methods.

In Dataframe `df.merge()`, `df.join()`, and `df.concat()` methods help in joining, merging and concat different dataframe.

Concatenating DataFrame

- In order to concat dataframe, we use `concat()` function which helps in concatenating a dataframe. We can concat a dataframe in many different ways, they are:
- Concatenating DataFrame using `.concat()`
- Concatenating DataFrame by setting logic on axes
- Concatenating DataFrame using `.append()`
- Concatenating DataFrame by ignoring indexes
- Concatenating DataFrame with group keys

1. Concatenating DataFrame using `.concat()` :

- In order to concat a dataframe, we use `.concat()` function this function concat a dataframe and returns a new dataframe.

```
# importing pandas module
import pandas as pd

# Define a dictionary containing employee data
data1 = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age': [27, 24, 22, 32],
        'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
        'Qualification': ['Msc', 'MA', 'MCA', 'Phd']}

# Define a dictionary containing employee data
data2 = {'Name': ['Abhi', 'Ayushi', 'Dhiraj', 'Hitesh'],
        'Age': [17, 14, 12, 52],
        'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
        'Qualification': ['Btech', 'B.A', 'Bcom', 'B.hons']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data1, index=[0, 1, 2, 3])

# Convert the dictionary into DataFrame
df1 = pd.DataFrame(data2, index=[4, 5, 6, 7])

print(df, "\n\n", df1)
```

Now we apply `.concat` function in order to concat two dataframe.

```
# using a .concat() method  
frames = [df, df1]  
  
res1 = pd.concat(frames)  
res1
```

Output :

As shown in the output image, we have created two dataframe after concatenating we get one dataframe.

	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannuaaj	Phd

	Name	Age	Address	Qualification
4	Abhi	17	Nagpur	Btech
5	Ayushi	14	Kanpur	B.A
6	Dhiraj	12	Allahabad	Bcom
7	Hitesh	52	Kannuaaj	B.hons

	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannuaj	Phd
4	Abhi	17	Nagpur	Btech
5	Ayushi	14	Kanpur	B.A
6	Dhiraj	12	Allahabad	Bcom
7	Hitesh	52	Kannuaj	B.hons

2. Concatenating DataFrame by setting logic on axes :

In order to concat dataframe, we have to set different logic on axes. We can set axes in the following three ways:

- Taking the union of them all, `join='outer'`. This is the default option as it results in zero information loss.
- Taking the intersection, `join='inner'`.
- Use a specific index, as passed to the `join_axes` argument.

```
# importing pandas module
import pandas as pd

# Define a dictionary containing employee data
data1 = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age': [27, 24, 22, 32],
        'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
        'Qualification': ['Msc', 'MA', 'MCA', 'Phd'],
        'Mobile No': [97, 91, 58, 76]}

# Define a dictionary containing employee data
data2 = {'Name': ['Gaurav', 'Anuj', 'Dhiraj', 'Hitesh'],
        'Age': [22, 32, 12, 52],
        'Address': ['Allahabad', 'Kannuaj', 'Allahabad', 'Kannuaj'],
        'Qualification': ['MCA', 'Phd', 'Bcom', 'B.hons'],
        'Salary': [1000, 2000, 3000, 4000]}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data1, index=[0, 1, 2, 3])

# Convert the dictionary into DataFrame
df1 = pd.DataFrame(data2, index=[2, 3, 6, 7])

print(df, "\n\n", df1)
```

- Now we set `axes join = inner` for intersection of dataframe.

```
# applying concat with axes  
# join = 'inner'  
res2 = pd.concat([df, df1], axis=1, join='inner')  
  
res2
```

Output :

As shown in the output image, we get the intersection of dataframe.

	Name	Age	Address	Qualification	Mobile No
0	Jai	27	Nagpur	Msc	97
1	Princi	24	Kanpur	MA	91
2	Gaurav	22	Allahabad	MCA	58
3	Anuj	32	Kannuaj	Phd	76

	Name	Age	Address	Qualification	Salary
2	Gaurav	22	Allahabad	MCA	1000
3	Anuj	32	Kannuaj	Phd	2000
6	Dhiraj	12	Allahabad	Bcom	3000
7	Hitesh	52	Kannuaj	B.hons	4000

	Name	Age	Address	Qualification	Mobile No	Name	Age	Address	Qualification	Salary
2	Gaurav	22	Allahabad	MCA	58	Gaurav	22	Allahabad	MCA	1000
3	Anuj	32	Kannuaj	Phd	76	Anuj	32	Kannuaj	Phd	2000

- Now we set `axes join = outer` for union of dataframe.

```
# using a .concat for  
# union of dataframe  
res2 = pd.concat([df, df1], axis=1, sort=False)
```

```
res2
```


Output :

As shown in the output image, we get the union of dataframe

	Name	Age	Address	Qualification	Mobile No	Name	Age	Address	Qualification	Salary
0	Jai	27.0	Nagpur	Msc	97.0	NaN	NaN	NaN	NaN	NaN
1	Princi	24.0	Kanpur	MA	91.0	NaN	NaN	NaN	NaN	NaN
2	Gaurav	22.0	Allahabad	MCA	58.0	Gaurav	22.0	Allahabad	MCA	1000.0
3	Anuj	32.0	Kannuaj	Phd	76.0	Anuj	32.0	Kannuaj	Phd	2000.0
6	NaN	NaN	NaN	NaN	NaN	Dhiraj	12.0	Allahabad	Bcom	3000.0
7	NaN	NaN	NaN	NaN	NaN	Hitesh	52.0	Kannuaj	B.hons	4000.0

- Now we used a specific index, as passed to the `join_axes` argument

```
# using join_axes
```

```
res3 = pd.concat([df, df1], axis=1, join_axes=[df.index])
```

```
res3
```


	Name	Age	Address	Qualification	Mobile No	Name	Age	Address	Qualification	Salary
0	Jai	27	Nagpur	Msc	97	NaN	NaN	NaN	NaN	NaN
1	Princi	24	Kanpur	MA	91	NaN	NaN	NaN	NaN	NaN
2	Gaurav	22	Allahabad	MCA	58	Gaurav	22.0	Allahabad	MCA	1000.0
3	Anuj	32	Kannuaj	Phd	76	Anuj	32.0	Kannuaj	Phd	2000.0

3. Concatenating DataFrame using .append()

In order to concat a dataframe, we use .append() function this function concatenate along axis=0, namely the index. This function exist before .concat.

```
# importing pandas module
import pandas as pd

# Define a dictionary containing employee data
data1 = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
         'Age': [27, 24, 22, 32],
         'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
         'Qualification': ['Msc', 'MA', 'MCA', 'Phd']}

# Define a dictionary containing employee data
data2 = {'Name': ['Abhi', 'Ayushi', 'Dhiraj', 'Hitesh'],
         'Age': [17, 14, 12, 52],
         'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
         'Qualification': ['Btech', 'B.A', 'Bcom', 'B.hons']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data1, index=[0, 1, 2, 3])

# Convert the dictionary into DataFrame
df1 = pd.DataFrame(data2, index=[4, 5, 6, 7])

print(df, "\n\n", df1)
```

- Now we apply `.append()` function in order to concat to dataframe

```
# using append function
```

```
res = df.append(df1)
```

```
res
```

	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannuaj	Phd

	Name	Age	Address	Qualification
4	Abhi	17	Nagpur	Btech
5	Ayushi	14	Kanpur	B.A
6	Dhiraj	12	Allahabad	Bcom
7	Hitesh	52	Kannuaj	B.hons

	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannuaj	Phd
4	Abhi	17	Nagpur	Btech
5	Ayushi	14	Kanpur	B.A
6	Dhiraj	12	Allahabad	Bcom
7	Hitesh	52	Kannuaj	B.hons

4. Concatenating DataFrame by ignoring indexes :

In order to concat a dataframe by ignoring indexes, we ignore index which don't have a meaningful meaning, you may wish to append them and ignore the fact that they may have overlapping indexes. In order to do that we use `ignore_index` as an argument.


```
# importing pandas module
import pandas as pd

# Define a dictionary containing employee data
data1 = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age': [27, 24, 22, 32],
        'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
        'Qualification': ['Msc', 'MA', 'MCA', 'Phd'],
        'Mobile No': [97, 91, 58, 76]}

# Define a dictionary containing employee data
data2 = {'Name': ['Gaurav', 'Anuj', 'Dhiraj', 'Hitesh'],
        'Age': [22, 32, 12, 52],
        'Address': ['Allahabad', 'Kannuaj', 'Allahabad', 'Kannuaj'],
        'Qualification': ['MCA', 'Phd', 'Bcom', 'B.hons'],
        'Salary': [1000, 2000, 3000, 4000]}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data1, index=[0, 1, 2, 3])

# Convert the dictionary into DataFrame
df1 = pd.DataFrame(data2, index=[2, 3, 6, 7])

print(df, "\n\n", df1)
```


- Now we are going to apply `ignore_index` as an argument.

```
# using ignore_index
```

```
res = pd.concat([df, df1], ignore_index=True)
```

```
res
```

	Name	Age	Address	Qualification	Mobile No
0	Jai	27	Nagpur	Msc	97
1	Princi	24	Kanpur	MA	91
2	Gaurav	22	Allahabad	MCA	58
3	Anuj	32	Kannuaj	Phd	76

	Name	Age	Address	Qualification	Salary
2	Gaurav	22	Allahabad	MCA	1000
3	Anuj	32	Kannuaj	Phd	2000
6	Dhiraj	12	Allahabad	Bcom	3000
7	Hitesh	52	Kannuaj	B.hons	4000

	Address	Age	Mobile No	Name	Qualification	Salary
0	Nagpur	27	97.0	Jai	Msc	NaN
1	Kanpur	24	91.0	Princi	MA	NaN
2	Allahabad	22	58.0	Gaurav	MCA	NaN
3	Kannuaj	32	76.0	Anuj	Phd	NaN
4	Allahabad	22	NaN	Gaurav	MCA	1000.0
5	Kannuaj	32	NaN	Anuj	Phd	2000.0
6	Allahabad	12	NaN	Dhiraj	Bcom	3000.0
7	Kannuaj	52	NaN	Hitesh	B.hons	4000.0

5. Concatenating DataFrame with group keys :

In order to concat dataframe with group keys, we override the column names with the use of the keys argument. Keys argument is to override the column names when creating a new DataFrame based on existing Series.

```
# importing pandas module
import pandas as pd

# Define a dictionary containing employee data
data1 = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age': [27, 24, 22, 32],
        'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
        'Qualification': ['Msc', 'MA', 'MCA', 'Phd']}

# Define a dictionary containing employee data
data2 = {'Name': ['Abhi', 'Ayushi', 'Dhiraj', 'Hitesh'],
        'Age': [17, 14, 12, 52],
        'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
        'Qualification': ['Btech', 'B.A', 'Bcom', 'B.hons']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data1, index=[0, 1, 2, 3])

# Convert the dictionary into DataFrame
df1 = pd.DataFrame(data2, index=[4, 5, 6, 7])

print(df, "\n\n", df1)
```

- Now we use keys as an argument.

```
# using keys  
frames = [df, df1 ]  
  
res = pd.concat(frames, keys=['x', 'y'])  
res
```


Output :

	Name	Age	Address	Qualification
0	Jai	27	Nagpur	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannuaj	Phd


	Name	Age	Address	Qualification
4	Abhi	17	Nagpur	Btech
5	Ayushi	14	Kanpur	B.A
6	Dhiraj	12	Allahabad	Bcom
7	Hitesh	52	Kannuaj	B.hons

		Name	Age	Address	Qualification
x	0	Jai	27	Nagpur	Msc
	1	Princi	24	Kanpur	MA
	2	Gaurav	22	Allahabad	MCA
	3	Anuj	32	Kannuaj	Phd
y	4	Abhi	17	Nagpur	Btech
	5	Ayushi	14	Kanpur	B.A
	6	Dhiraj	12	Allahabad	Bcom
	7	Hitesh	52	Kannuaj	B.hons

Concatenating Vertically

Dataset 1: Scores in Math

python

 Copy code


```
import pandas as pd

data1 = {
    'Student': ['Alice', 'Bob', 'Charlie'],
    'Math_Score': [85, 90, 78]
}

df1 = pd.DataFrame(data1)
print("Math Scores:")
print(df1)
```

Output:


javascript

 Copy code

	Student	Math_Score
0	Alice	85
1	Bob	90
2	Charlie	78


Dataset 2: Scores in Science

python

 Copy code

```
data2 = {  
    'Student': ['Alice', 'Bob', 'Charlie'],  
    'Science_Score': [88, 92, 80]  
}  
  
df2 = pd.DataFrame(data2)  
print("\nScience Scores:")  
print(df2)
```


Output:

 Copy code

	Student	Science_Score
0	Alice	88
1	Bob	92
2	Charlie	80

Concatenating Vertically : If you want to combine these two datasets vertically (adding rows), you can use **pd.concat()** :

python

 Copy code

```
# Combining dataframes vertically


vertical_concat = pd.concat([df1, df2], axis=0)

print("\nVertical Concatenation:")

print(vertical_concat)
```

Output:


r

 Copy code

	Student	Math_Score	Science_Score
0	Alice	85	NaN
1	Bob	90	NaN
2	Charlie	78	NaN
0	Alice	NaN	88.0
1	Bob	NaN	92.0
2	Charlie	NaN	80.0

Concatenating Horizontally To combine them horizontally (adding columns), use **axis=1**:


python

 Copy code

```
# Combining dataframes horizontally  
horizontal_concat = pd.concat([df1, df2], axis=1)  
print("\nHorizontal Concatenation:")  
print(horizontal_concat)
```

Output:

javascript

 Copy code

	Student	Math_Score	Student	Science_Score
0	Alice	85	Alice	88
1	Bob	90	Bob	92
2	Charlie	78	Charlie	80

Merging DataFrame

1. Merging a dataframe with **one unique key** combination

```
# importing pandas module
import pandas as pd

# Define a dictionary containing employee data
data1 = {'key': ['K0', 'K1', 'K2', 'K3'],
        'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age': [27, 24, 22, 32],}

# Define a dictionary containing employee data
data2 = {'key': ['K0', 'K1', 'K2', 'K3'],
        'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
        'Qualification': ['Btech', 'B.A', 'Bcom', 'B.hons']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data1)

# Convert the dictionary into DataFrame
df1 = pd.DataFrame(data2)

print(df, "\n\n", df1)
```

Now we are using **.merge()** with one unique key combination.

```
# using .merge() function  
res = pd.merge(df, df1, on='key')
```

```
res
```

Output :

	key	Name	Age
0	K0	Jai	27
1	K1	Princi	24
2	K2	Gaurav	22
3	K3	Anuj	32

	key	Address	Qualification
0	K0	Nagpur	Btech
1	K1	Kanpur	B.A
2	K2	Allahabad	Bcom
3	K3	Kannauj	B.hons

	key	Name	Age	Address	Qualification
--	-----	------	-----	---------	---------------

0	K0	Jai	27	Nagpur	Btech
---	----	-----	----	--------	-------

1	K1	Princi	24	Kanpur	B.A
---	----	--------	----	--------	-----

2	K2	Gaurav	22	Allahabad	Bcom
---	----	--------	----	-----------	------

3	K3	Anuj	32	Kannuaj	B.hons
---	----	------	----	---------	--------

2. Merging dataframe using **multiple join keys**.

```
# importing pandas module
import pandas as pd

# Define a dictionary containing employee data
data1 = {'key': ['K0', 'K1', 'K2', 'K3'],
         'key1': ['K0', 'K1', 'K0', 'K1'],
         'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
         'Age': [27, 24, 22, 32],}

# Define a dictionary containing employee data
data2 = {'key': ['K0', 'K1', 'K2', 'K3'],
         'key1': ['K0', 'K0', 'K0', 'K0'],
         'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],
         'Qualification': ['Btech', 'B.A', 'Bcom', 'B.hons']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data1)

# Convert the dictionary into DataFrame
df1 = pd.DataFrame(data2)

print(df, "\n\n", df1)
```

Now we merge dataframe using multiple keys.

```
# merging dataframe using multiple keys  
res1 = pd.merge(df, df1, on=['key', 'key1'])
```

```
res1
```


Output :

	key	key1	Name	Age
0	K0	K0	Jai	27
1	K1	K1	Princi	24
2	K2	K0	Gaurav	22
3	K3	K1	Anuj	32

	key	key1	Address	Qualification
0	K0	K0	Nagpur	Btech
1	K1	K0	Kanpur	B.A
2	K2	K0	Allahabad	Bcom
3	K3	K0	Kannauj	B.hons

	key	key1	Name	Age	Address	Qualification
0	K0	K0	Jai	27	Nagpur	Btech
1	K2	K0	Gaurav	22	Allahabad	Bcom

3. Merging dataframe using **how** in an argument:

We use how argument to merge specifies how to determine which keys are to be included in the resulting table. If a key combination does not appear in either the left or right tables, the values in the joined table will be NA. Here is a summary of the how options and their SQL equivalent names:

MERGE METHOD	JOIN NAME	DESCRIPTION
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames

```
# importing pandas module
```

```
import pandas as pd
```

```
# Define a dictionary containing employee data
```

```
data1 = {'key': ['K0', 'K1', 'K2', 'K3'],  
         'key1': ['K0', 'K1', 'K0', 'K1'],  
         'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],  
         'Age': [27, 24, 22, 32],}
```

```
# Define a dictionary containing employee data
```

```
data2 = {'key': ['K0', 'K1', 'K2', 'K3'],  
         'key1': ['K0', 'K0', 'K0', 'K0'],  
         'Address': ['Nagpur', 'Kanpur', 'Allahabad', 'Kannuaj'],  
         'Qualification': ['Btech', 'B.A', 'Bcom', 'B.hons']}
```

```
# Convert the dictionary into DataFrame
```

```
df = pd.DataFrame(data1)
```

Now we set `how = 'left'` in order to use keys from **left frame** only.

```
# using keys from left frame  
res = pd.merge(df, df1, how='left', on=['key', 'key1'])
```

```
res
```

Left

	key	key1	Name	Age
0	K0	K0	Jai	27
1	K1	K1	Princi	24
2	K2	K0	Gaurav	22
3	K3	K1	Anuj	32

Right

	key	key1	Address	Qualification
0	K0	K0	Nagpur	Btech
1	K1	K0	Kanpur	B.A
2	K2	K0	Allahabad	Bcom
3	K3	K0	Kannauj	B.hons

	key	key1	Name	Age	Address	Qualification
0	K0	K0	Jai	27	Nagpur	Btech
1	K1	K1	Princi	24	NaN	NaN
2	K2	K0	Gaurav	22	Allahabad	Bcom
3	K3	K1	Anuj	32	NaN	NaN

Now we set `how = 'right'` in order to use keys from **right frame** only.

```
# using keys from right frame
```

```
res1 = pd.merge(df, df1, how='right', on=['key', 'key1'])
```

```
res1
```


Output :

Left

	key	key1	Name	Age
0	K0	K0	Jai	27
1	K1	K1	Princi	24
2	K2	K0	Gaurav	22
3	K3	K1	Anuj	32

Right

	key	key1	Address	Qualification
0	K0	K0	Nagpur	Btech
1	K1	K0	Kanpur	B.A
2	K2	K0	Allahabad	Bcom
3	K3	K0	Kannuaaj	B.hons

	key	key1	Name	Age	Address	Qualification
0	K0	K0	Jai	27.0	Nagpur	Btech
1	K2	K0	Gaurav	22.0	Allahabad	Bcom
2	K1	K0	NaN	NaN	Kanpur	B.A
3	K3	K0	NaN	NaN	Kannuaj	B.hons

Now we set `how = 'outer'` in order to get union of keys from dataframes.

```
# getting union of keys  
res2 = pd.merge(df, df1, how='outer', on=['key', 'key1'])  
  
res2
```

Output :

Left

	key	key1	Name	Age
0	K0	K0	Jai	27
1	K1	K1	Princi	24
2	K2	K0	Gaurav	22
3	K3	K1	Anuj	32

Right

	key	key1	Address	Qualification
0	K0	K0	Nagpur	Btech
1	K1	K0	Kanpur	B.A
2	K2	K0	Allahabad	Bcom
3	K3	K0	Kannauj	B.hons

	key	key1	Name	Age	Address	Qualification
0	K0	K0	Jai	27.0	Nagpur	Btech
1	K1	K1	Princi	24.0	NaN	NaN
2	K2	K0	Gaurav	22.0	Allahabad	Bcom
3	K3	K1	Anuj	32.0	NaN	NaN
4	K1	K0	NaN	NaN	Kanpur	B.A
5	K3	K0	NaN	NaN	Kannuaaj	B.hons

Now we set `how = 'inner'` in order to get intersection of keys from dataframes.

```
# getting intersection of keys  
res3 = pd.merge(df, df1, how='inner', on=['key', 'key1'])
```

```
res3
```

Output :

Left

	key	key1	Name	Age
0	K0	K0	Jai	27
1	K1	K1	Princi	24
2	K2	K0	Gaurav	22
3	K3	K1	Anuj	32

Right

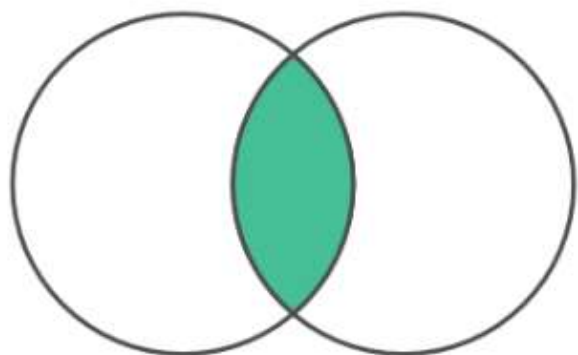
	key	key1	Address	Qualification
0	K0	K0	Nagpur	Btech
1	K1	K0	Kanpur	B.A
2	K2	K0	Allahabad	Bcom
3	K3	K0	Kannuaj	B.hons

	key	key1	Name	Age	Address	Qualification
0	K0	K0	Jai	27	Nagpur	Btech
1	K2	K0	Gaurav	22	Allahabad	Bcom

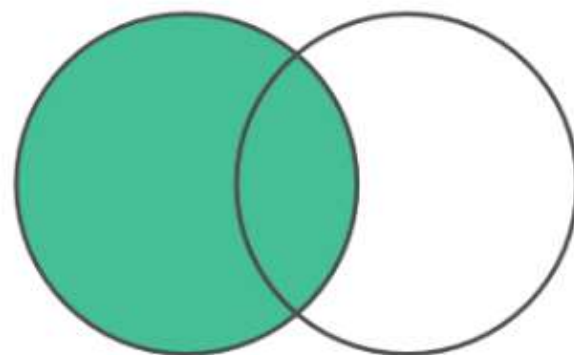
Joining DataFrame

When we need to combine very large DataFrames, joins serve as a powerful way to perform these operations swiftly. Joins can only be done on two DataFrames at a time, denoted as left and right tables. The key is the common column that the two DataFrames will be joined on.

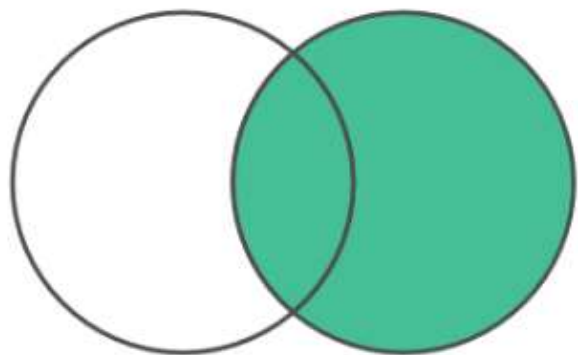
There are four basic ways to handle the join (inner, left, right, and outer), depending on which rows must retain their data.



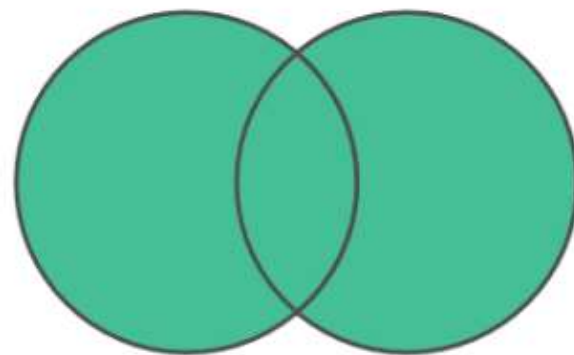
INNER JOIN



LEFT OUTER JOIN



RIGHT OUTER JOIN



FULL OUTER JOIN

In order to join dataframe, we use `.join()` function this function is used for combining the **columns of two potentially differently-indexed** DataFrames **into a single result DataFrame.**

```
# importing pandas module
import pandas as pd

# Define a dictionary containing employee data
data1 = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
         'Age': [27, 24, 22, 32]}

# Define a dictionary containing employee data
data2 = {'Address': ['Allahabad', 'Kannuaj', 'Allahabad', 'Kannuaj'],
         'Qualification': ['MCA', 'Phd', 'Bcom', 'B.hons']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data1, index=['K0', 'K1', 'K2', 'K3'])

# Convert the dictionary into DataFrame
df1 = pd.DataFrame(data2, index=['K0', 'K2', 'K3', 'K4'])

print(df, "\n\n", df1)
```

Now we are use **.join()** method in order to join dataframes

```
# joining dataframe  
res = df.join(df1)
```

```
res
```

Output :

Left

	Name	Age
K0	Jai	27
K1	Princi	24
K2	Gaurav	22
K3	Anuj	32

Right

	Address	Qualification
K0	Allahabad	MCA
K2	Kannuaj	Phd
K3	Allahabad	Bcom
K4	Kannuaj	B.hons

	Name	Age	Address	Qualification
K0	Jai	27	Allahabad	MCA
K1	Princi	24	NaN	NaN
K2	Gaurav	22	Kannuaj	Phd
K3	Anuj	32	Allahabad	Bcom

Now we use `how = 'outer'` in order to get union

```
# getting union  
res1 = df.join(df1, how='outer')
```

```
res1
```

Output :

	Name	Age	Address	Qualification	Mobile No
0	Jai	27	Nagpur	Msc	97
1	Princi	24	Kanpur	MA	91
2	Gaurav	22	Allahabad	MCA	58
3	Anuj	32	Kannuaj	Phd	76

	Name	Age	Address	Qualification	Salary
2	Gaurav	22	Allahabad	MCA	1000
3	Anuj	32	Kannuaj	Phd	2000
6	Dhiraj	12	Allahabad	Bcom	3000
7	Hitesh	52	Kannuaj	B.hons	4000

	Name	Age	Address	Qualification	Mobile No	Name	Age	Address	Qualification	Salary
2	Gaurav	22	Allahabad	MCA	58	Gaurav	22	Allahabad	MCA	1000
3	Anuj	32	Kannuaj	Phd	76	Anuj	32	Kannuaj	Phd	2000

Joining dataframe using **on** in an argument :

In order to join dataframes we use **on** in an argument. `join()` takes an optional `on` argument which may be a column or multiple column names, which specifies that the passed DataFrame is to be aligned on that column in the DataFrame.

```
# importing pandas module
import pandas as pd

# Define a dictionary containing employee data
data1 = {'Name': ['Jai', 'Princi', 'Gaurav', 'Anuj'],
         'Age': [27, 24, 22, 32],
         'Key': ['K0', 'K1', 'K2', 'K3']}

# Define a dictionary containing employee data
data2 = {'Address': ['Allahabad', 'Kannauj', 'Allahabad', 'Kannauj'],
         'Qualification': ['MCA', 'Phd', 'Bcom', 'B.hons']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data1)

# Convert the dictionary into DataFrame
df1 = pd.DataFrame(data2, index=['K0', 'K2', 'K3', 'K4'])

print(df, "\n\n", df1)
```

Now we are using **.join** with “**on**” argument.

```
# using on argument in join  
res2 = df.join(df1, on='Key')
```

```
res2
```

Output :

Left

	Name	Age	Key
0	Jai	27	K0
1	Princi	24	K1
2	Gaurav	22	K2
3	Anuj	32	K3

Right

	Address	Qualification
K0	Allahabad	MCA
K2	Kannauj	Phd
K3	Allahabad	Bcom
K4	Kannauj	B.hons

	Name	Age	Address	Qualification
K0	Jai	27.0	Allahabad	MCA
K1	Princi	24.0	NaN	NaN
K2	Gaurav	22.0	Kannuaj	Phd
K3	Anuj	32.0	Allahabad	Bcom
K4	NaN	NaN	Kannuaj	B.hons

Joining singly-indexed DataFrame with multi-indexed DataFrame :

In order to join singly indexed dataframe with multi-indexed dataframe, the level will match on the name of the index of the singly-indexed frame against a level name of the multi-indexed frame.


```
# importing pandas module
import pandas as pd

# Define a dictionary containing employee data
data1 = {'Name': ['Jai', 'Princi', 'Gaurav'],
         'Age': [27, 24, 22]}

# Define a dictionary containing employee data
data2 = {'Address': ['Allahabad', 'Kannauj', 'Allahabad', 'Kanpur'],
         'Qualification': ['MCA', 'Phd', 'Bcom', 'B.hons']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data1, index=pd.Index(['K0', 'K1', 'K2'], name='key'))

index = pd.MultiIndex.from_tuples([('K0', 'Y0'), ('K1', 'Y1'),
                                   ('K2', 'Y2'), ('K2', 'Y3')],
                                names=['key', 'Y'])

# Convert the dictionary into DataFrame
df1 = pd.DataFrame(data2, index=index)

print(df, "\n\n", df1)
```


Now we join singly indexed dataframe with multi-indexed dataframe.

```
# joining singly indexed with  
# multi indexed  
result = df.join(df1, how='inner')  
  
result
```

Output :

	Name	Age
key		
K0	Jai	27
K1	Princi	24
K2	Gaurav	22

		Address	Qualification
key	Y		
K0	Y0	Allahabad	MCA
K1	Y1	Kannuaj	Phd
K2	Y2	Allahabad	Bcom
	Y3	Kanpur	B.hons

		Name	Age	Address	Qualification
--	--	------	-----	---------	---------------

key	Y
-----	---

K0	Y0	Jai	27	Allahabad	MCA
----	----	-----	----	-----------	-----

K1	Y1	Princi	24	Kannuaaj	Phd
----	----	--------	----	----------	-----

K2	Y2	Gaurav	22	Allahabad	Bcom
----	----	--------	----	-----------	------

	Y3	Gaurav	22	Kanpur	B.hons
--	----	--------	----	--------	--------

Dealing with duplicate key during data merging

When you merge two datasets using a key, duplicate keys can lead to multiple rows in the resulting DataFrame. This may not always be desirable, as it can complicate analysis and data interpretation.

Overview of the Process:

1. **Create DataFrames** with duplicate keys.
2. **Merge** the DataFrames.
3. **Handle duplicates** by either aggregating or removing them

Step 1: Create Sample DataFramesLet's create two DataFrames: one with duplicate keys and one with unique keys.

```
import pandas as pd

# Create DataFrame 1 with duplicates
data1 = {
    'key': ['A', 'B', 'C', 'A'], # Duplicate key 'A'
    'value1': [10, 20, 30, 40] # Associated values
}

# Create DataFrame 2 with unique keys
data2 = {
    'key': ['A', 'B', 'D'], # 'D' is unmatched
    'value2': [100, 200, 300] # Associated values
}
```

```
# Create pandas DataFrames
```

```
df1 = pd.DataFrame(data1)
```

```
df2 = pd.DataFrame(data2)
```

```
print("DataFrame 1:")
```

```
print(df1)
```

```
print("\nDataFrame 2:")
```

```
print(df2)
```

Step 2: Merge the DataFramesNow, let's merge the two DataFrames on the key column using an outer join to include all keys.

python

```
# Merge DataFrames
```

```
merged_df = pd.merge(df1, df2, on='key', how='outer')
```


```
print("\nMerged DataFrame:")
```

```
print(merged_df)
```


Output :

The merged DataFrame will look like this:

CSS


 Copy code

	key	value1	value2
0	A	10.0	100.0
1	A	40.0	100.0
2	B	20.0	200.0
3	C	30.0	NaN
4	D	NaN	300.0

Step 3: Handle Duplicates

Option 1: Aggregating Duplicate Values You can summarize the data for duplicate keys, for example by taking the mean of values.

python

 Copy code

```
# Aggregate by taking the mean for duplicate keys in df1
df1_agg = df1.groupby('key', as_index=False).mean()

# Merge with the aggregated DataFrame
merged_agg_df = pd.merge(df1_agg, df2, on='key', how='outer')
print("\nMerged DataFrame after aggregation:")
print(merged_agg_df)
```

Output :

After AggregationThis will result in:

CSS

	key	value1	value2
0	A	25.0	100.0
1	B	20.0	200.0
2	C	30.0	NaN
3	D	NaN	300.0

Option 2: Removing Duplicates

If you want to keep only one occurrence of each duplicate key:

python

```
# Remove duplicates from df1
df1_unique = df1.drop_duplicates(subset='key')

# Merge with unique keys
merged_unique_df = pd.merge(df1_unique, df2, on='key', how='outer')
print("\nMerged DataFrame after removing duplicates:")
print(merged_unique_df)
```

Output :

This will give you:

CSS

	key	value1	value2
0	A	10	100.0
1	B	20	200.0
2	C	30	NaN
3	D	NaN	300.0

Complete Example Code:

Here's the complete code for all the steps:

```
import pandas as pd

# Create DataFrame 1 with duplicates
data1 = {
    'key': ['A', 'B', 'C', 'A'], # Duplicate key 'A'
    'value1': [10, 20, 30, 40] # Associated values
}

# Create DataFrame 2 with unique keys
data2 = {
    'key': ['A', 'B', 'D'], # 'D' is unmatched
    'value2': [100, 200, 300] # Associated values
}
```

```
# Create pandas DataFrames
```

```
df1 = pd.DataFrame(data1)
```

```
df2 = pd.DataFrame(data2)
```

```
print("DataFrame 1:")
```

```
print(df1)
```

```
print("\nDataFrame 2:")
```

```
print(df2)
```

```
# Merge DataFrames
```

```
merged_df = pd.merge(df1, df2, on='key', how='outer')
```

```
print("\nMerged DataFrame:")
```

```
print(merged_df)
```



```
# Aggregate values in df1 by taking the mean for duplicate keys
```

```
df1_agg = df1.groupby('key', as_index=False).mean()
```

```
merged_agg_df = pd.merge(df1_agg, df2, on='key', how='outer')
```

```
print("\nMerged DataFrame after aggregation:")
```

```
print(merged_agg_df)
```

```
# Remove duplicates from df1
```

```
df1_unique = df1.drop_duplicates(subset='key')
```

```
merged_unique_df = pd.merge(df1_unique, df2, on='key', how='outer')
```

```
print("\nMerged DataFrame after removing duplicates:")
```

```
print(merged_unique_df)
```

Advantages

- 1.Data Integrity:** Handling duplicates helps maintain the accuracy and integrity of the merged dataset, preventing misleading results.
- 2.Enhanced Analysis:** Aggregating duplicates can provide more meaningful insights, such as average scores instead of individual entries.
- 3.Cleaner Data:** Removing or addressing duplicates results in a cleaner, more manageable dataset that is easier to work

Disadvantages

- 1.Data Loss Risk:** Removing duplicates without careful consideration may lead to loss of important data or context.
- 2.Increased Complexity:** The process of identifying and handling duplicates adds complexity to the data merging workflow, requiring additional processing steps.
- 3.Potential Misinterpretation:** Aggregating data may hide nuances and variations within the duplicates that could be relevant for specific analyses.

Dealing with mismatched keys during data merging

Dealing with mismatched keys during data merging involves addressing discrepancies that occur when combining two or more datasets based on a common identifier (key). Mismatched keys can result from differences in naming conventions, missing records, or data from different sources that do not align perfectly. Effectively managing these mismatches is crucial for ensuring data integrity and accuracy in analyses.

Example:

Consider two datasets:

Dataset A (Users)

UserID	Name
1	Alice
2	Bob
3	Charlie

Dataset B (Scores)

UserID	Score
1	90
2	85
4	70

Merging Datasets:

```
import pandas as pd

# Create the two datasets

data_a = {
    'UserID': [1, 2, 3],
    'Name': ['Alice', 'Bob', 'Charlie']
}

data_b = {
    'UserID': [1, 2, 4],
    'Score': [90, 85, 70]
}
```



```
# Convert them into DataFrames
df_a = pd.DataFrame(data_a)
df_b = pd.DataFrame(data_b)

# Merge the datasets on 'UserID' using an outer join
merged_df = pd.merge(df_a, df_b, on='UserID', how='outer')

# Display the result
print("Merged DataFrame:")
print(merged_df)

# Fill missing values
merged_df.fillna({'Name': 'Unknown', 'Score': 0}, inplace=True)
print("\nDataFrame after filling NaN values:")
print(merged_df)
```

Output

Merged DataFrame:

	UserID	Name	Score
0	1	Alice	90.0
1	2	Bob	85.0
2	3	Charlie	NaN
3	4	NaN	70.0

DataFrame after filling NaN values:

	UserID	Name	Score
0	1	Alice	90.0
1	2	Bob	85.0
2	3	Charlie	0.0
3	4	Unknown	70.0

How to Deal with Mismatched Keys

1. Use Different Join Types:

- **Inner Join:** Only includes rows with matching keys. This can lead to data loss if there are many mismatches.
- **Outer Join:** Includes all rows from both datasets, filling in missing values with NaN.
- **Left Join:** Includes all rows from the left dataset and only matching rows from the right dataset.
- **Right Join:** Includes all rows from the right dataset and only matching rows from

2. Fill Missing Values:

After merging, you can fill NaN values with default values, like 0, "Unknown", or the mean/median of the column.

3. Identify and Clean Data:

Before merging, identify keys that are present in one dataset but not the other. You can clean or standardize the data to ensure consistency.

4. Standardize Keys:

Ensure that keys are in the same format across datasets (e.g., case sensitivity, leading/trailing spaces).

Advantages

1. Comprehensive Data Capture:

Outer joins ensure no data is lost during the merge, providing a full view of available data.

2. Flexibility in Analysis:

Handling mismatches allows for deeper analysis, enabling analysts to explore data gaps and their implications.

3. Improved Decision-Making:

A complete dataset enhances the quality of insights drawn from data analysis, leading to better business decisions.

Disadvantages

1. Increased Complexity:

Managing mismatched keys can complicate the data merging process, requiring additional steps and checks.

2. Data Quality Issues:

Filling missing values might lead to inaccurate conclusions if the default values do not reflect the true nature of the data.

3. Performance Overhead:

Merging large datasets, especially with many mismatches, can be resource-intensive and slow down processing times.

Advanced Merging Techniques:

1. Fuzzy Matching

Fuzzy matching is a technique used to identify strings that are similar but not exactly the same. This is particularly useful when merging datasets that may have inconsistencies due to typos, different formats, or variations in spelling.

Or

String matching or fuzzy matching is a method to find strings which match a given pattern or string approximately. It identifies the likelihood or probability that two records are true match based on some parameters. In the example shown below, the algorithms try to match the 5 different variations to the given string 'Microsoft Corporation' and score each of

ID	Name
01	Microsoft
02	Microsoft Co.
03	Microsoft Corporation
04	Mcrosoft Corp
05	Microosoftt



Microsoft Corporation

How Fuzzy Matching Works

Fuzzy matching typically employs algorithms that calculate a similarity score between two strings. The higher the score, the more similar the strings are. This enables matches even when the strings contain errors or differences.

Concept and Importance

Fuzzy matching enhances data quality by allowing flexible comparisons based on similarity rather than exact matches. This technique is widely used in applications like customer relationship management, record linkage, and natural language processing.

Common Algorithms for Fuzzy Matching

Several algorithms are employed in fuzzy matching

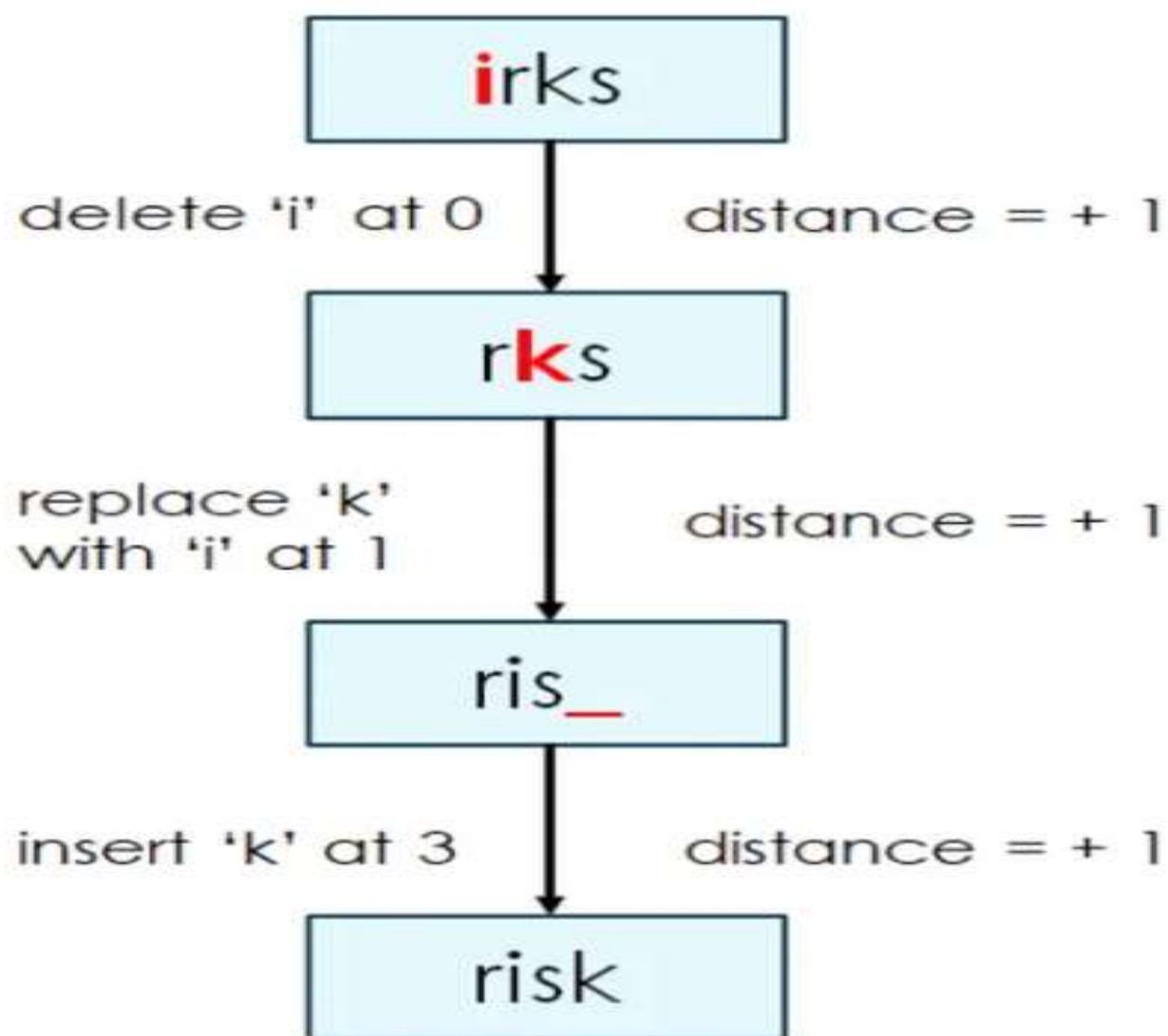
1. Levenshtein Distance:

- **Definition:** Measures the minimum number of single-character edits (insertions, deletions, substitutions) required to transform one string into another.
- **Use Case:** Ideal for detecting small typographical errors. For example, it can match "Jon" to "John".

2. Jaccard Similarity:

- **Definition:** Compares the size of the intersection of two sets to the size of their union.
- **Use Case:** Effective for comparing text

Levenshtein Distance



3. Cosine Similarity:

- **Definition:** Measures the cosine of the angle between two vectors in a multi-dimensional space.
- **Use Case:** Commonly used in natural language processing to determine the similarity of documents.

4. Soundex and Metaphone:

- **Definition:** Phonetic algorithms that convert words into codes based on how they sound.
- **Use Case:** Useful for matching names that may be spelled differently but sound similar.

5.N-grams :

- **Definition:** Breaks down strings into contiguous sequences of n characters or words.
- **Use Case:** Helpful in text comparison where the sequence of characters is more significant than the overall structure.

Implementation in Python

The fuzzywuzzy library is a popular choice for implementing fuzzy matching in Python. Below is a complete example demonstrating

```
import pandas as pd
from fuzzywuzzy import process

# Sample datasets
data1 = {'Name': ['John Smith', 'Jane Doe', 'Jake White']}
data2 = {'Name': ['Jon Smith', 'J. Doe', 'Jacque White', 'Jane D.']}

df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)
```

```
# Function to find the best match
```

```
def get_best_match(name, choices):
```

```
    match, score = process.extractOne(name, choices)
```

```
    return match, score
```

```
# Apply fuzzy matching to the first dataset
```

```
df1['Best Match'], df1['Score'] = zip(*df1['Name'].apply(get_best_match, choices=df2['Name
```

```
# Display results
```

```
print(df1)
```

Output :

	Name	Best Match	Score
0	John Smith	Jon Smith	90
1	Jane Doe	J. Doe	80
2	Jake White	Jacque White	80

Conclusion

Fuzzy matching is a powerful technique for improving data quality by allowing flexible string comparisons. By utilizing algorithms like Levenshtein distance and Jaccard similarity, it effectively addresses the challenges posed by imperfect data. Understanding and implementing these algorithms can significantly enhance data management practices and improve decision-making based on accurate datasets.

