Merging Data

In this lesson, an explanation on how to merge different data sets is provided.

We'll cover the following

- Merge
  - Left merge
  - Right merge
  - Outer merge
- Merge on multiple columns
- Merge on index

Merge

To merge the rows of two or more `DataFrames` based on a common *column* between them, use pandas `merge(df1, df2, ...)` function. This returns another `DataFrame` with only the common *column(s)* and their corresponding row values.

In short, two things need to be in common for the `DataFrames` to be merged:

- The column names
- The row values of those column names

DataFrame 1

DataFrame 2

Common
Column

Common
Rows

Merged DataFrame


```
import numpy as np
import pandas as pd

df1 = pd.DataFrame({'pointer':['A', 'B', 'C', 'B', 'A', 'D'],
                    'value_df1':[0,1,2,3,4,5]})
```

```
print("First DataFrame")
print(df1)


df2 = pd.DataFrame({'pointer':['B', 'C', 'B','D'],
                    'value_df2':[6,7,8,9]})

print("\nSecond DataFrame")
print(df2)

print("\nMerged DataFrame")
print('\n',pd.merge(df1, df2)) # Merging two DataFrames
```

It can be observed that the two `DataFrames` have `pointer` as the common column; in `pointer`, there are multiple common rows as displayed in the above illustration. The output shows that the two `DataFrames` are merged on the common rows of that common column, and returned a new `DataFrame` based on those.

This is the most basic type of merge. There are three more ways to merge a `DataFrame`, and this is achieved by passing specific parameters to the `merge()` function.

Left merge

The *left* merge returns a `DataFrame`, which has all rows of the `DataFrame` placed on the left side of the `merge()` function. Those rows of the left `DataFrame`, which do not have a corresponding matching value in the right `DataFrame`, are then assigned `NaN` values.

This behavior can be observed in the following example:

```
import numpy as np
import pandas as pd

df1 = pd.DataFrame({'pointer':['A', 'B', 'C', 'B', 'A', 'D'],
                    'value_df1':[0,1,2,3,4,5]})

df2 = pd.DataFrame({'pointer':['B', 'C', 'B', 'D', 'E'],
                    'value_df2':[6, 7, 8, 9, 12]})

print("Left Merged DataFrame\n")
print(pd.merge(df1, df2, how = 'left')) # Performing a left merge
```

It can be seen that those rows of left `DataFrame` that do not match or don't exist in the right `DataFrame` are assigned `NaN` values. On **line 11**, the `how` parameter is used to achieve this by assigning it the value `left`.

Right merge

The *right* merge returns a `DataFrame` that has all the rows of the `DataFrame` placed on the right side of the `merge()` function. The rows to the right `DataFrame` that do not have a corresponding matching value in the left `DataFrame` are assigned `NaN` values.

This behavior can be observed in the following example:

```
import numpy as np
import pandas as pd

df1 = pd.DataFrame({'pointer':['A', 'B', 'C', 'B', 'A', 'D'],
                    'value_df1':[0,1,2,3,4,5]})

df2 = pd.DataFrame({'pointer':['B', 'Z', 'C', 'B','D','E'],
                    'value_df2':[6,7,8,9,10,11]})

print("Right Merged DataFrame\n")
print(pd.merge(df1, df2, how = 'right')) # Performing a right merge
```

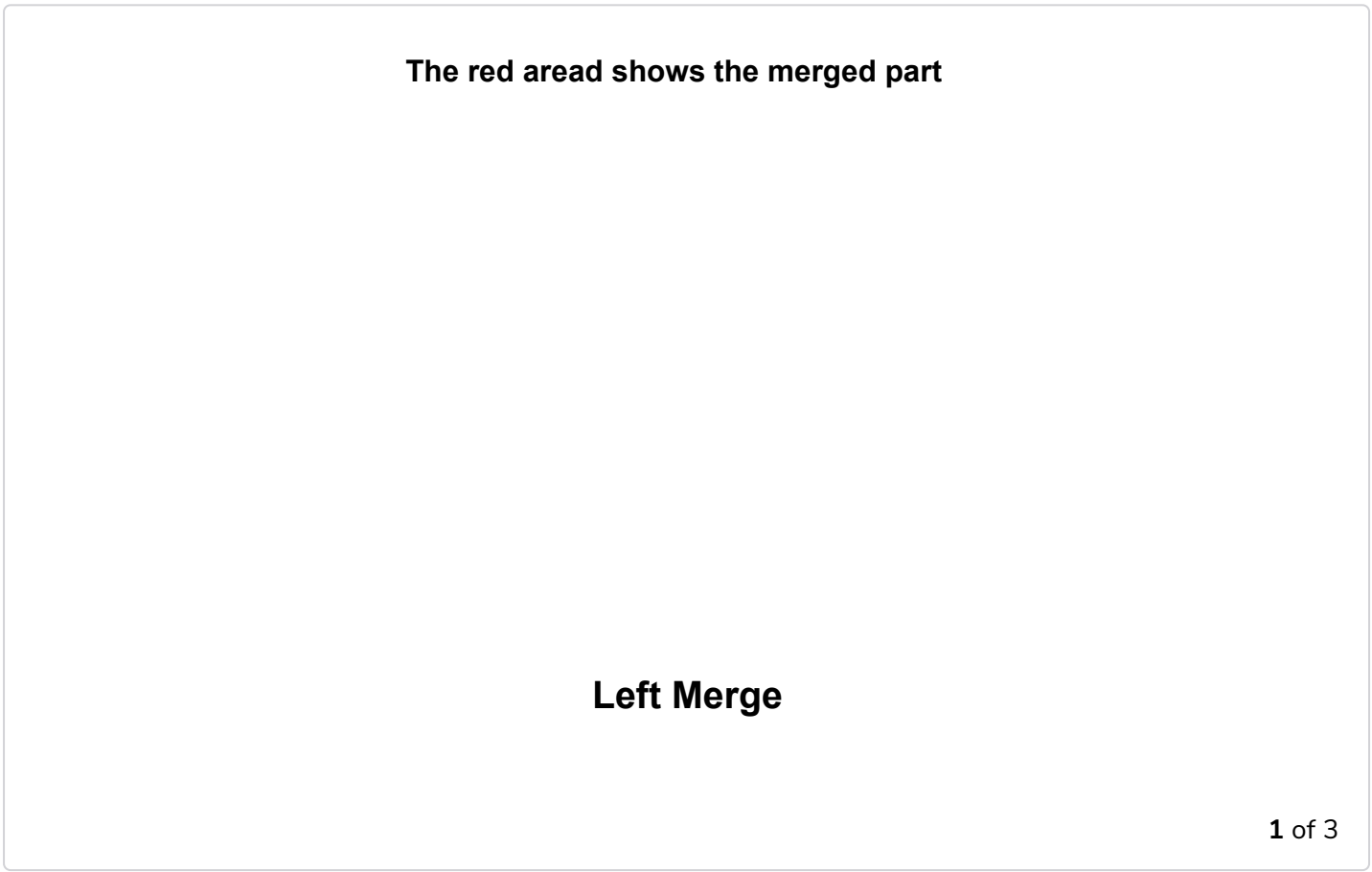
It can be seen that those rows of the right `DataFrame` that do not match or don't exist in the left `DataFrame` are assigned `NaN` values. On **line 11**, the `how` parameter is used to achieve this by assigning the value `right` to it.

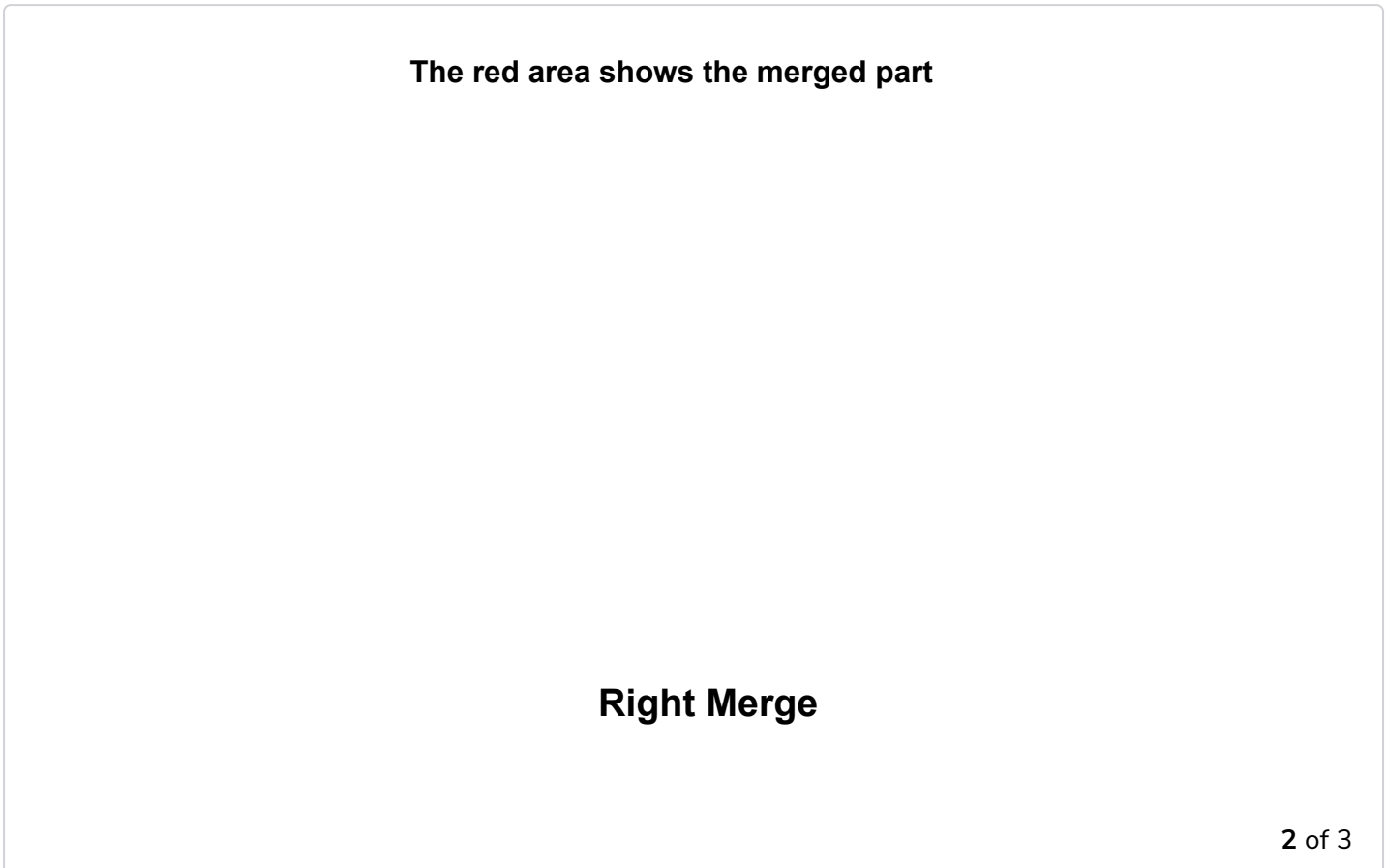
Outer merge

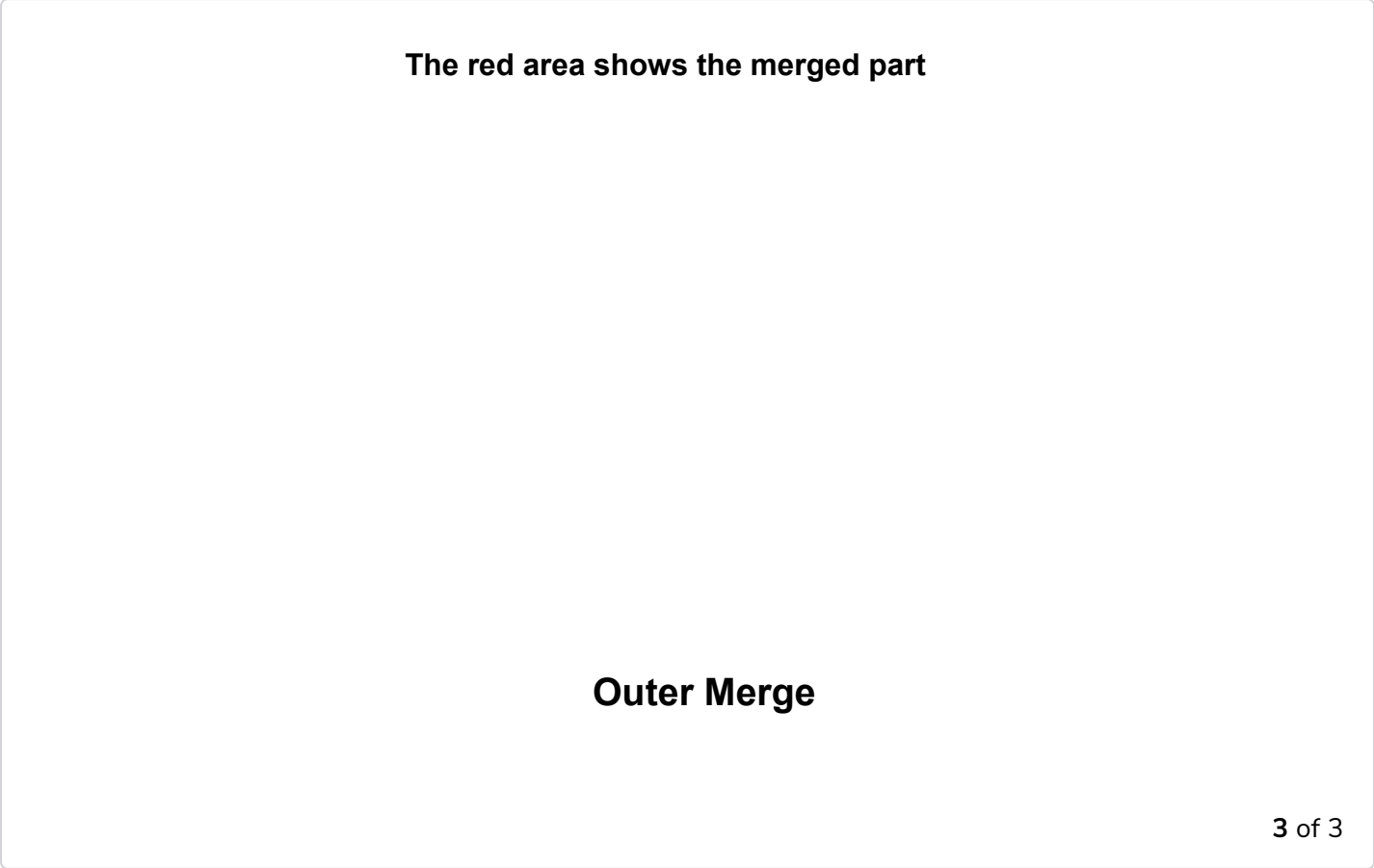
This function returns all the rows of both the `DataFrames` given in the `merge()` function. The rows that don't get matched in either case are assigned `NaN` values.

```
import numpy as np
import pandas as pd

df1 = pd.DataFrame({'pointer':['A', 'B', 'C', 'B', 'A', 'D'],
                    'value_df1':[0,1,2,3,4,5]})

df2 = pd.DataFrame({'pointer':['B', 'Z', 'C', 'B','D','E'],
                    'value_df2':[6,7,8,9,10,11]})

print("Outer Merged DataFrame\n")
print(pd.merge(df1, df2, how = 'outer')) # Performing an outer merge
```

The output is as expected. As the common column `pointer` doesn't have any corresponding matching values for the row `A` in the second `DataFrame`, they are assigned `NaN`. Similarly, there are no corresponding matching values for the rows `Z` and `E` in the first `DataFrame`, so they are also assigned `NaN`.







Merge on multiple columns

Two or more columns belonging to different `DataFrames` can also be joined if they have the same name in both the DataFrames. For example, if one DataFrame has columns `[one, two, ...]`, then the other should also have these two columns for both `DataFrames` to be merged.

```
import numpy as np
import pandas as pd

df1 = pd.DataFrame({'column1':['Pak', 'USA', 'Pak', 'UK', 'Ind','None'], #Column 1
                    'column2':['A', 'B', 'C', 'B', 'A', 'D'],            #Column 2
                    'value_df1':[0,1,2,3,4,5]})

df2 = pd.DataFrame({'column1':['USA', 'UK', 'None', 'USA', 'Pak','Ind'], #Column 1
                    'column2':['B', 'Z', 'C', 'B','D','E'],              #Column 2
                    'value_df2':[6,7,8,9,10,11]})

print("Outer Merged DataFrame on Multiple Columns\n")
print(pd.merge(df1, df2, on = ['column1', 'column2'], how = 'outer'))
```

The output shows that every row of column `value_df1` and `value_df2` is referred to by the combination of two *columns*: `column1` and `column2`. As this is an outer merge operation, every row of each `DataFrame` is part of the resultant `DataFrame`. Those rows of columns `value_df1` and `value_df2` are `NaN` for which no combination of `column1` and `column2` values exist. For example, `Pak` and `A` exist in the first `DataFrame` and not in the second one. Therefore, the corresponding value for `value_df1` is `0` and for `value_df2` is `NaN`. The same is the case for the rest.

Merge on index

So far, the two `DataFrames` have been merged based on their common rows of common columns; first, it has to find common columns and then common rows of that common column to get merged. Now, you'll see how rows can be merged on indexes. This behavior and technique is almost the same. The only difference is that now the column of one `DataFrame` is merged with the index of another `DataFrame`.

For one `DataFrame`, the common values are changed from rows to *indexes*. Two new parameters are needed for this purpose. The `left_on` parameter that takes the column name of left `DataFrame` and `right_index` parameter that has to be set to `True`.

```
import numpy as np
import pandas as pd

df1 = pd.DataFrame({'pointer':['A', 'B', 'C', 'B', 'A', 'D'],
                    'value_df1':[0,1,2,3,4,5]})

df2 = pd.DataFrame(np.arange(10,13,1), index = ['A', 'B','C'], columns = ['values'])
print(df2)
print("Merged on index\n")
print(pd.merge(df1, df2, left_on='pointer', right_index=True))
```

In the `merge()` function of the above example, two new parameters were introduced. The `left_on` parameter indicates which column to choose from the `Dataframe`, placed on the left side of the `merge()` function. The `right_index=True` indicates merging the chosen *column* on the indexes of the right `DataFrame`. Similarly, parameters like `right_on` and `left_index` can also be used depending on

the problem.

The output is the same as if two columns would be used to merge the `DataFrames`. The only difference is that the column values of the dataframe `df1` are merged with the *index* values of the dataframe `df2`.

For more information on pandas `merge()` function, refer here.

> Merging `DataFrames` is similar to joining tables in a database.

The next lesson discusses how the shape of a `DataFrame` can be changed.