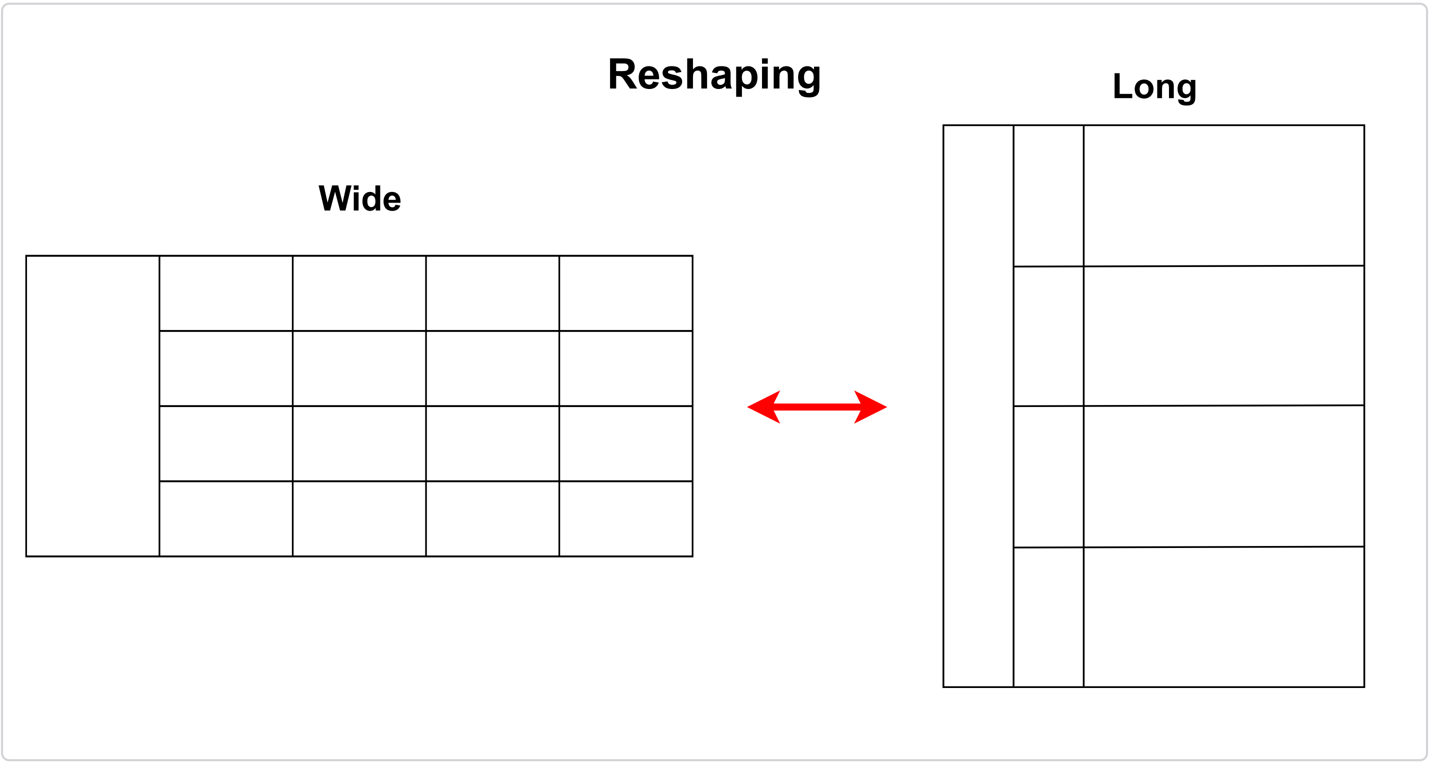
# Reshaping Data

In this lesson, some data reshaping techniques are discussed.

**We'll cover the following**

## Reshaping #

Reshaping is considered a powerful tool for getting the data ready for analysis. It transforms the initial data model into the desired shape to ease the cleaning and manipulation of data for further analysis. This technique is usually performed on a `Series` or `DatFrame` with multiple indexes.



Three techniques are mostly used to reshape data, and pandas provide built-in functions for all of them.

- **Stacking**

- **Unstacking**

- **Pivoting**

## Stack

Stacking a pandas `DataFrame` means taking the innermost level of column index from a multi-indexed `DataFrame` and adding it as another level to the innermost row index. If the `DataFrame` is not multi-indexed, it takes the first column and moves it to the innermost row level.

The following example explains this concept.

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(12).reshape(3,4), index = ['Row1', 'Row2', 'Row3'],
                                               columns = ['Col1','Col2','Col3','Col4',])

df.index.name = 'Row'
df.columns.name = 'Column'

print("The Original DataFrame")
print(df,'\n')

print("The Stacked DataFrame")
print(df.stack())
```

The `stack()` method converts the single indexed `DataFrame` into a multi-indexed `Series`. The *column* index is moved to the innermost row index along with their values. For more information on this function, refer here.

## Unstack

Unstacking a pandas `Series` or `DataFrame` takes the innermost level of row index from a multi-indexed `DataFrame` and adding it as another level to the innermost column indexes. If the `DataFrame` is not multi-indexed, it takes the first row and moves it to the innermost column level.

The following example explains this concept.

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.arange(12).reshape(3,4), index = ['Row1', 'Row2', 'Row3'],
                                            columns = ['Col1','Col2','Col3','Col4',])

df.index.name = 'Row'
df.columns.name = 'Column'

print("The Original DataFrame")
print(df,'\n')

stacked_df = df.stack()
print("The Stacked DataFrame\n", stacked_df)

print("\nThe Unstacked DataFrame")
print(stacked_df.unstack(),'\n')

print("The Unstacked DataFrame on Named index")
print(stacked_df.unstack('Row'))
```

On **line 13**, the `DataFrame` is first stacked using the `stack` function, and it is stored in a `stacked_df` variable, which is then unstacked in the following two ways:

- The function on **line 17** is the default `unstack()` function, and it moves the innermost row index level and adds it as the innermost column index level. It reverts the `DataFrame` to its original form before stacking.

- The function on **line 20** takes the row index name as a parameter. The `unstack` function is applied to the `Row` index. Now, instead of the innermost row index, the index level with the specified row name is moved as the innermost column index level.

> If unstacking is done on the `Series` object then it transforms into a `DataFrame`.

For more information on this function refer here.

## Pivot tables

This function creates a table that contains information from the original table based on the parameters defined by the user. It describes what specific information the user wants to know and how the user wants to present this information.

The `pivot` function takes three parameters as `row_index`, `column_index`, `value`.

```
import numpy as np
import pandas as pd

df = pd.DataFrame({'Company': ['Google', 'Microsoft', 'Google', 'Microsoft'],
      'Product': ['Editor', 'Editor', 'Calendar', 'Azure'],
      'Price': ['$200', '$250', '$50', '$400']})

df.index.name = 'Row'
df.columns.name = 'Column'

print("The Original DataFrame")
print(df,'\n')

print("The Pivoted DataFrame")
print(df.pivot('Company', 'Product', 'Price'))
```

A new method of declaring a `DataFrame` using the `dict` data structure of python was used in the above example. On **lines 4 - 6**, the `DataFrame` is defined using the `dict` data structure method by simply placing a *dictionary* object inside the `pd.DataFrame()` function.

It can be seen from the output that the new table has unique values of the `Product` column as the new column indexes. The unique values of the `Company` column are the new row indexes. The cells of the newly created `DataFrame` have values of the `Price` column corresponding to the row and column index values of the original `DataFrame`.

The `None` values appear in those cells of the new `DataFrame`, which have no corresponding values in the row/column *indexes*. For example, `Google` has no product named `Azure`, so there is no `Price` for it in the original `DataFrame`. Therefore, `None` is assigned to it automatically.

> Here, `None` is equivalent to `NaN`.

---

In the next lesson, data mapping along with how to find duplicates in pandas objects is discussed.