Chapter 6 Concatenation and Merging

Concatenation

Often the data we need exists in two or more separate sources, fortunately, Pandas makes it easy to combine these together. The simplest combination is if all the sources are already in the same format, then a concatenation through the **Pandas.concat()** call is all that is needed.

Ex: pd.concat([df1, df2, ..., dfN])

Concatenation of Series

Concatenation of Series by rows (One below another) - Default

In [1]:
```
import pandas as pd

# Create series s1
s1 = pd.Series(['1', '2', '3'])

# Create series s2
s2 = pd.Series(['4', '5', '6'])
```

In [2]:
```
# Print series s1
s1
```

Out[2]:
```
0    1
1    2
2    3
dtype: object
```

In [3]:
```
# Print series s2
s2
```

Out[3]:
```
0    4
1    5
2    6
dtype: object
```

In [4]:
```
# Concat both the series by rows (s2 after s1)
pd.concat([s1, s2], keys=["s1","s2"])

# Returns a new series with records of s2 after records of s1
# The indexes are preserved for both the series s1 (0, 1, 2) and s2 (0, 1, 2)
# Can create group labels using keys
```

Out[4]:
```
s1  0    1
    1    2
    2    3
s2  0    4
    1    5
    2    6
dtype: object
```

In [5]:
```
# Concat both the series by rows (s2 after s1)
pd.concat([s1, s2], ignore_index=True)

# Returns a new series with records of s2 after records of s1
# The indexes are preserved for the series s1 (0, 1, 2) and new indexes created for
# Since indexes are ignored, we cannot create group labels
```

Out[5]:
```
0    1
1    2
2    3
3    4
4    5
5    6
dtype: object
```

In [6]:
```
# Concat both the series by rows (s1 after s2)
pd.concat([s2, s1], keys=["s2","s1"])

# Returns a new series with records of s1 after records of s2
# The indexes are preserved for both the series s2 (0, 1, 2) and s1 (0, 1, 2)
# Can create group labels using keys
```

Out[6]:
```
s2  0    4
    1    5
    2    6
s1  0    1
    1    2
    2    3
dtype: object
```

In [7]:
```
# Concat both the series by rows (s1 after s2)
pd.concat([s2, s1], ignore_index=True)

# Returns a new series with records of s1 after records of s2
# The indexes are preserved for the series s2 (0, 1, 2) and new indexes created for
# Since indexes are ignored, we cannot create group labels
```

Out[7]:
```
0    4
1    5
2    6
3    1
4    2
5    3
dtype: object
```

Concatenation of Series by columns/indexes (One adjacent to another)

In [8]:
```
# Create series c1
c1 = pd.Series(['red', 'orange', 'yellow'])

# Create series c2
c2 = pd.Series(['green', 'blue', 'purple'])
```

In [9]:
```
# Print series c1
c1
```

Out[9]:
```
0       red
1    orange
2    yellow
dtype: object
```

In [10]:
```
# Print series c2
c2
```

Out[10]:
```
0     green
1      blue
2    purple
dtype: object
```

In [11]:
```
# Concat both the series by columns
pd.concat([c1,c2], axis=1)

# Returns a dataframe with records of c1 then c2, adjacent to each other
```

Out[11]:

	0	1
0	red	green
1	orange	blue
2	yellow	purple

In [12]:
```
# Concat both the series by columns
pd.concat([c2,c1], axis=1)

# Returns a dataframe with records of c2 then c1, adjacent to each other
```

Out[12]:

	0	1
0	green	red
1	blue	orange
2	purple	yellow

Outer vs. Inner Concatenation of Series

In [13]:
```
# Create series with different indexes to understand the working of Concatenation b
animals = pd.Series(
    data=["badger", "cougar", "anaconda", "elk", "pika"],
    index=["b", "c", "a", "e", "p"]
```

```
)

fruits = pd.Series(
    data=["apple", "banana", "cherry", "durian"],
    index=["a", "b", "c", "d"]
)
```

In [14]:
```
# Print animals series
animals
```

Out[14]:
```
b      badger
c      cougar
a    anaconda
e         elk
p        pika
dtype: object
```

In [15]:
```
# Print fruits series
fruits
```

Out[15]:
```
a     apple
b    banana
c    cherry
d    durian
dtype: object
```

In [16]:
```
# Concat both the series by columns - Outer Concatenation (default)
pd.concat([animals,fruits], axis=1, keys=["animals", "fruits"])

# Returns a dataframe with records of animals then fruits, adjacent to each other
# Concatenated by index, and the indexes of animals (first dataframe) are preserved
# If any record with corresponding index is missing, it is replaced by NaN
# We can set the column names using keys
```

Out[16]:

	animals	fruits
b	badger	banana
c	cougar	cherry
a	anaconda	apple
e	elk	NaN
p	pika	NaN
d	NaN	durian

In [17]:
```
# Concat both the series by columns - Inner Concatenation
pd.concat([animals,fruits], axis=1, keys=["animals", "fruits"], join="inner")

# Returns a dataframe with records of animals then fruits, adjacent to each other
# Concatenated by index, and the indexes of animals (first dataframe) are preserved
# If any record with corresponding index is missing, it is omitted
# We can set the column names using keys
```

Out[17]:

	animals	fruits
b	badger	banana
c	cougar	cherry
a	anaconda	apple

In [18]:
```
# Concat both the series by columns - Outer Concatenation (default)
pd.concat([fruits,animals], axis=1, keys=["fruits","animals"])

# Returns a dataframe with records of fruits then animals, adjacent to each other
# Concatenated by index, and the indexes of fruits (first dataframe) are preserved
# If any record with corresponding index is missing, it is replaced by NaN
# We can set the column names using keys
```

Out[18]:

	fruits	animals
a	apple	anaconda
b	banana	badger
c	cherry	cougar
d	durian	NaN
e	NaN	elk
p	NaN	pika

In [19]:
```
# Concat both the series by columns - Inner Concatenation
pd.concat([fruits,animals], axis=1, keys=["fruits","animals"], join="inner")

# Returns a dataframe with records of fruits then animals, adjacent to each other
# Concatenated by index, and the indexes of fruits (first dataframe) are preserved
# If any record with corresponding index is missing, it is omitted
# We can set the column names using keys
```

Out[19]:

	fruits	animals
a	apple	anaconda
b	banana	badger
c	cherry	cougar

Concatenation of Dataframes

Concatenation of Dataframes by rows (One below another) - Default

In [20]:
```
# Create a dataframe harvest_21
harvest_21 = pd.DataFrame([['potatoes', 900], ['garlic', 1350], ['onions', 875]], c
```

```
# Print dataframe
harvest_21
```

Out[20]:

	crop	qty
0	potatoes	900
1	garlic	1350
2	onions	875

In [21]:
```
# Create a dataframe harvest_22
harvest_22 = pd.DataFrame([['garlic', 1600], ['spinach', 560], ['turnips', 999], ['

# Print dataframe
harvest_22
```

Out[21]:

	crop	qty
0	garlic	1600
1	spinach	560
2	turnips	999
3	onions	1000

In [22]:
```
# Concat both the dataframes, providing keys to group
pd.concat([harvest_21, harvest_22], keys=["2021", "2022"])
```

Out[22]:

		crop	qty
2021	**0**	potatoes	900
	1	garlic	1350
	2	onions	875
2022	**0**	garlic	1600
	1	spinach	560
	2	turnips	999
	3	onions	1000

In [23]:
```
# Concat both the dataframes, ignoring index
pd.concat([harvest_21, harvest_22], ignore_index=True)
```

Out[23]:

	crop	qty
0	potatoes	900
1	garlic	1350
2	onions	875
3	garlic	1600
4	spinach	560
5	turnips	999
6	onions	1000

Concatenation of Dataframes by columns/indexes (One adjacent to another)

In [24]:
```
# Create a dataframe livestock
livestock = pd.DataFrame(
    [["pasture", 9], ["stable", 3], ["coop", 34]],
    columns=["location", "qty"],
    index=["alpaca", "horse", "chicken"]
)

# Print the dataframe
livestock
```

Out[24]:

	location	qty
alpaca	pasture	9
horse	stable	3
chicken	coop	34

In [25]:
```
# Create a dataframe weights
weights = pd.DataFrame(
    [[4, 10], [900, 2000], [1.2, 4], [110, 150]],
    columns=["min_weight", "max_weight"],
    index=["chicken", "horse", "duck", "alpaca"]
)

# Print the dataframe
weights
```

Out[25]:

	min_weight	max_weight
chicken	4.0	10
horse	900.0	2000
duck	1.2	4
alpaca	110.0	150

In [26]:
```
# Concat both the dataframes by columns, providing keys to group - Outer Concatenat
pd.concat([livestock, weights], axis=1, keys=["Source1", "Source2"])
```

Out[26]:

	Source1		Source2	
	location	**qty**	**min_weight**	**max_weight**
alpaca	pasture	9.0	110.0	150
horse	stable	3.0	900.0	2000
chicken	coop	34.0	4.0	10
duck	NaN	NaN	1.2	4

In [27]:
```
# Concat both the dataframes by columns - Inner Concatenation
pd.concat([livestock, weights], axis=1, keys=["Source1", "Source2"], join="inner")
```

Out[27]:

	Source1		Source2	
	location	**qty**	**min_weight**	**max_weight**
alpaca	pasture	9	110.0	150
horse	stable	3	900.0	2000
chicken	coop	34	4.0	10

Merging

1. Often, DataFrames are not in the exact same order or format, which means we cannot simply concatenate them together.

In such cases, we need to use the **DataFrame.merge()** method to merge the DataFrames based on a common column.

This operation is analogous to a JOIN command in SQL, where the column passed for merging must be unique and non-null.

2. The 'on' parameter in the DataFrame.merge() method specifies the column or index level names to join on.

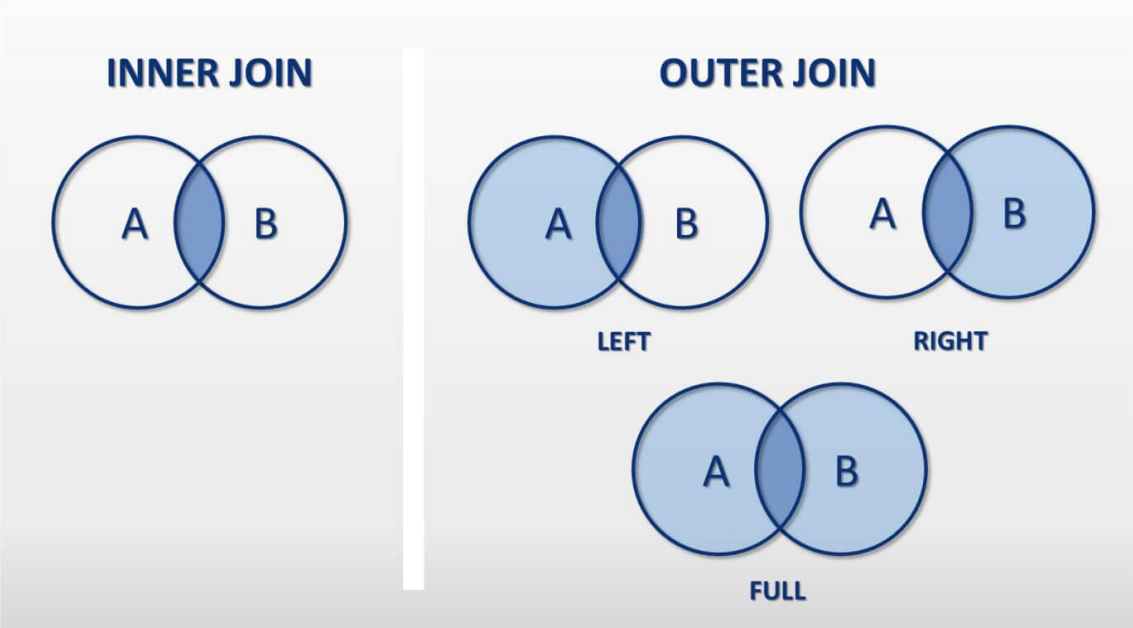
It allows you to explicitly define the column(s) that should serve as the key(s) for the merge operation.

When using the 'on' parameter, both DataFrames should have a column with the same name, and the values in that column are used to align the rows for merging.

3. The **DataFrame.merge()** method also accepts a key argument, labeled **how**, to specify the type of join to be performed.

There are 3 main ways to merge tables together using the **how** parameter:

```
1) Inner
2) Outer and
3) Left or Right
```



In [28]:
```python
# Create a dataframe teams
teams = pd.DataFrame(
    [
        ["Suns", "Phoenix", 20, 4],
        ["Mavericks", "Dallas", 11, 12],
        ["Rockets", "Houston", 7, 16],
        ["Nuggets", "Denver", 11, 12]
    ],
    columns=["team", "city", "wins", "losses"]
)

# Print the dataframe
teams
```

Out[28]:

	team	city	wins	losses
0	Suns	Phoenix	20	4
1	Mavericks	Dallas	11	12
2	Rockets	Houston	7	16
3	Nuggets	Denver	11	12

In [29]:
```
# Create a dataframe cities
cities = pd.DataFrame(
    [
        ["Houston", "Texas", 2310000],
        ["Phoenix", "Arizona", 1630000],
        ["San Diego", "California", 1410000],
        ["Dallas", "Texas", 1310000],
    ],
    columns=["city", "state", "population"]
)

# Print the dataframe
cities
```

Out[29]:

	city	state	population
0	Houston	Texas	2310000
1	Phoenix	Arizona	1630000
2	San Diego	California	1410000
3	Dallas	Texas	1310000

Inner Join - Default

Merges the datasets by including only the rows with matching keys in both datasets, resulting in a dataset containing only the intersection of the two datasets.

In [30]:
```
# Merge the dataframes on the column city - Inner Join (default)
teams.merge(cities, on="city", how="inner")

# Prints records available in both the dataframes
```

Out[30]:

	team	city	wins	losses	state	population
0	Suns	Phoenix	20	4	Arizona	1630000
1	Mavericks	Dallas	11	12	Texas	1310000
2	Rockets	Houston	7	16	Texas	2310000

Outer Join

Combines the datasets by including all rows from both datasets, filling in missing values with NaN where data is unavailable in atleast one of the datasets.

In [31]:
```
# Merge the dataframes on the column city using Outer Join
teams.merge(cities, on="city", how="outer")

# Prints all the records regardless of whether it being available in both the dataf
```

Out[31]:

	team	city	wins	losses	state	population
0	Suns	Phoenix	20.0	4.0	Arizona	1630000.0
1	Mavericks	Dallas	11.0	12.0	Texas	1310000.0
2	Rockets	Houston	7.0	16.0	Texas	2310000.0
3	Nuggets	Denver	11.0	12.0	NaN	NaN
4	NaN	San Diego	NaN	NaN	California	1410000.0

Left Join

Merges the datasets by including all rows from the left dataset and matching rows from the right dataset, filling in missing values with NaN where data is unavailable in the right dataset.

In [32]:
```
# Merge the dataframes on the column city using Left Join
teams.merge(cities, on="city", how="left")

# Prints all the records of teams dataframe regardless of whether it being availabl
```

Out[32]:

	team	city	wins	losses	state	population
0	Suns	Phoenix	20	4	Arizona	1630000.0
1	Mavericks	Dallas	11	12	Texas	1310000.0
2	Rockets	Houston	7	16	Texas	2310000.0
3	Nuggets	Denver	11	12	NaN	NaN

Right Join

Merges the datasets by including all rows from the right dataset and matching rows from the left dataset, filling in missing values with NaN where data is unavailable in the left dataset.

In [33]:
```
# Merge the dataframes on the column city using Right Join
teams.merge(cities, on="city", how="right")

# Prints all the records of cities dataframe regardless of whether it being availab
```

Out[33]:

	team	city	wins	losses	state	population
0	Rockets	Houston	7.0	16.0	Texas	2310000
1	Suns	Phoenix	20.0	4.0	Arizona	1630000
2	NaN	San Diego	NaN	NaN	California	1410000
3	Mavericks	Dallas	11.0	12.0	Texas	1310000