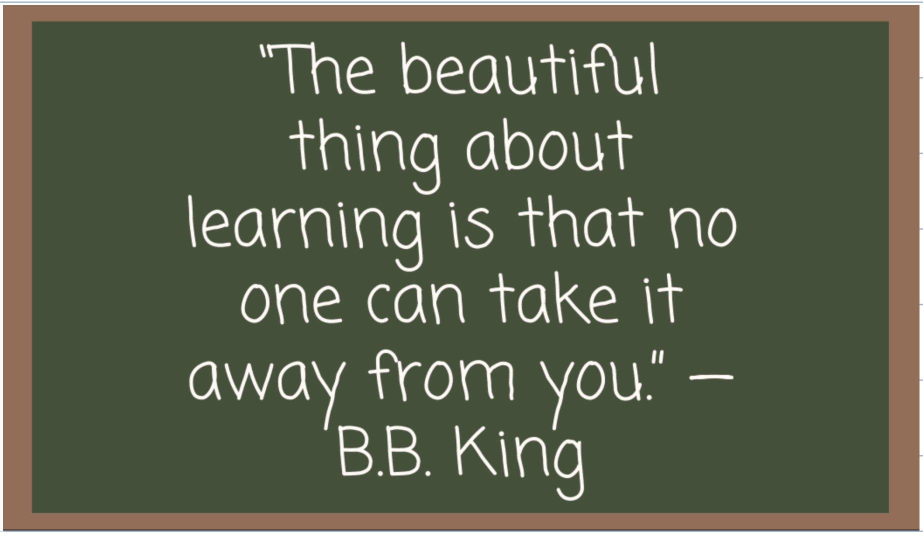
Content

- Working with both rows & columns
- Handling duplicate records
- Pandas built-in operations
    - Aggregate functions
    - Sorting values
- Concatenating DataFrames
- Merging DataFrames


"The beautiful thing about learning is that no one can take it away from you." — B.B. King

How can we do add a row using the `concat()` method?

Code:

```
new_row = {'country': 'India', 'year': 2000,'population':13500000, 'continent

df = pd.concat([df, pd.DataFrame([new_row])], ignore_index=True)
```

Why are we using `ignore_index=True` ?

- This parameter tells Pandas to ignore the existing index and create a new one based on the length of the resulting DataFrame.

Perfect! Our row is now added at the bottom of the dataframe.

Note:

- `concat()` doesn't mutate the the dataframe.
- It does not change the DataFrame, but returns a new DataFrame with the appended row.

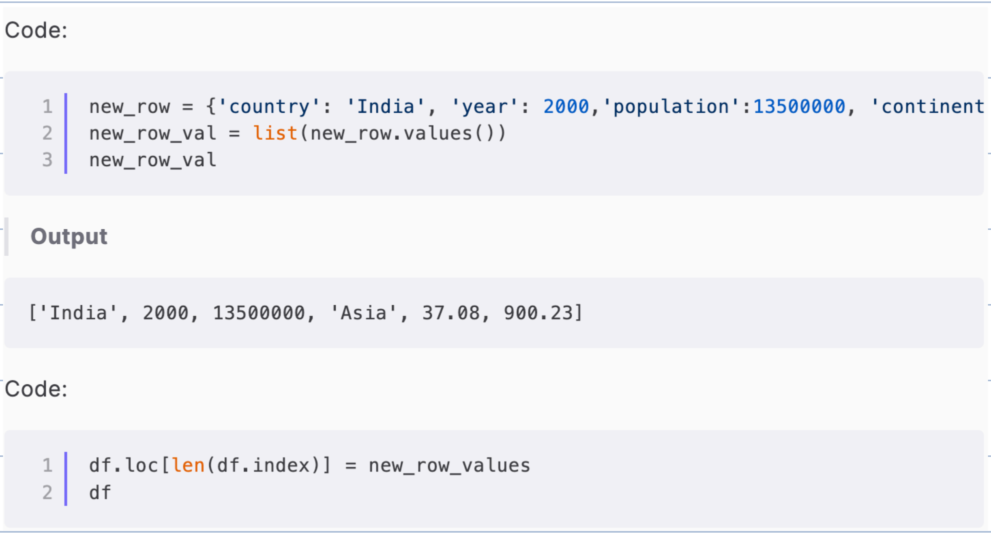
Another method would be by using `loc`.

We will need to provide the position at which we want to add the new row.

What do you think this positional value would be?

- `len(df.index)` since we will add the new row at the end.

For this method we only need to insert the values of columns in the respective manner.



Code:

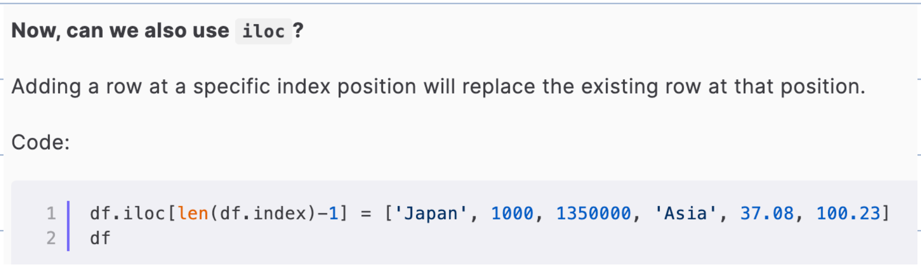
```
new_row = {'country': 'India', 'year': 2000,'population':13500000, 'continent
new_row_val = list(new_row.values())
new_row_val
```

Output

```
['India', 2000, 13500000, 'Asia', 37.08, 900.23]
```

Code:

```
df.loc[len(df.index)] = new_row_values
df
```

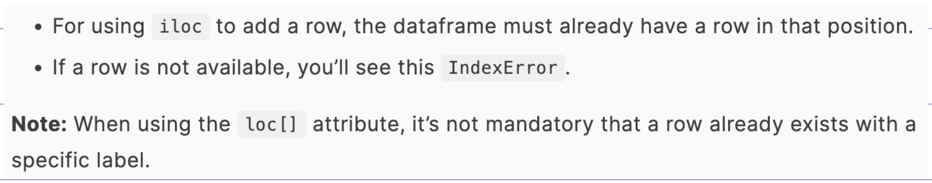


Now, can we also use `iloc`?

Adding a row at a specific index position will replace the existing row at that position.

Code:

```
df.iloc[len(df.index)-1] = ['Japan', 1000, 1350000, 'Asia', 37.08, 100.23]
df
```



- For using `iloc` to add a row, the dataframe must already have a row in that position.
- If a row is not available, you'll see this `IndexError`.

Note: When using the `loc[]` attribute, it's not mandatory that a row already exists with a specific label.

What if we want to delete a row?

- use `df.drop()`

If you remember we specified axis=1 for columns.

We can modify this - `axis=0` for rows.

Does `drop()` method uses positional indices or labels?

- We had to specify column title.
- So **`drop()` uses labels**, NOT positional indices.

df = df.drop (index, axis = 0)

How can we drop multiple rows?

Code:

```
df.drop([1, 2, 4], axis=0) # drops rows with labels 1, 2, 4
```

How to check for duplicate rows?

- We use `duplicated()` method on the DataFrame.

```
# Extracting duplicate rows

df.loc[df.duplicated()]
```

How do we get rid of these duplicate rows?

- We can use the `drop_duplicates()` function.

But how do we decide among all duplicate rows which ones to keep?

Here we can use the `keep` argument.

It has only three distinct values -

- `first`
- `last`
- `False`

The default is 'first'.

If `first`, this considers first value as unique and rest of the identical values as duplicate.

If `last`, this considers last value as unique and rest of the identical values as duplicate.

Code:

```
df.drop_duplicates(keep='last')
```

If `False`, this considers all the identical values as duplicates.

Code:

```
df.drop_duplicates(keep=False)
```

What if you want to look for duplicacy only for a few columns?

We can use the `subset` argument to mention the list of columns which we want to use.

Code:

```
df.drop_duplicates(subset=['country'],keep='first')
```

Slicing the DataFrame

How can we slice the dataframe into, say first 4 rows and first 3 columns?

- We can use `iloc`

Code:

```
df.iloc[0:4, 0:3]
```

Recall, we need to work with explicit labels while using `loc` .

Code:

```
df.loc[1:5, ['country','life_exp']]
```

How can we get specific rows and columns?

Code:

```
df.iloc[[0,10,100], [0,2,3]]
```

Pandas built-in operations

Aggregate functions

le = df['life_exp']

le.mean()

What other operations can we do?

- `sum()`
- `count()`
- `min()`
- `max()`

... and so on

Sorting Values

If you notice, the `life_exp` column is not sorted.

How can we perform sorting in Pandas?

Code:

```
df.sort_values(['life_exp'])
```

Rows get sorted **based on values in `life_exp` column**.

By default, values are sorted in **ascending order**.

How can we sort the rows in descending order?

Code:

```
df.sort_values(['life_exp'], ascending=False)
```

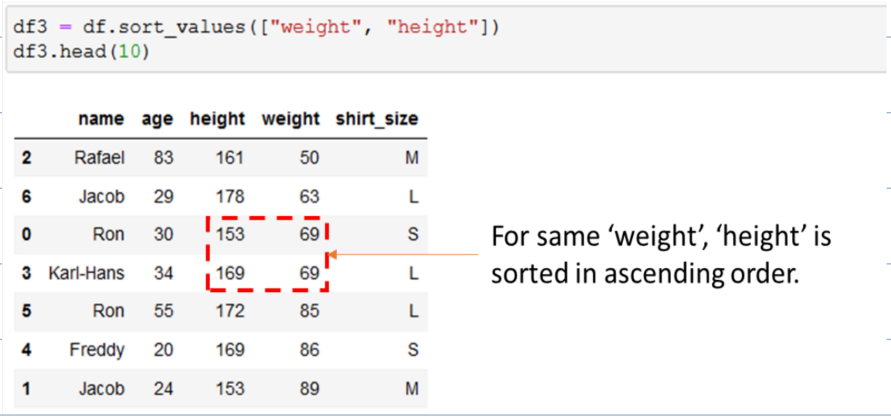
Can we perform sorting on multiple columns? Yes!

Code:

```
df.sort_values(['year', 'life_exp'])
```

What exactly happened here?

- Rows were **first sorted** based on `'year'`
- Then, **rows with same values of `'year'`** were sorted based on `'lifeExp'`



```
df3 = df.sort_values(["weight", "height"])
df3.head(10)
```

	name	age	height	weight	shirt_size
2	Rafael	83	161	50	M
6	Jacob	29	178	63	L
0	Ron	30	153	69	S
3	Karl-Hans	34	169	69	L
5	Ron	55	172	85	L
4	Freddy	20	169	86	S
1	Jacob	24	153	89	M

For same 'weight', 'height' is sorted in ascending order.

This way, we can do multi-level sorting of our data.

How can we have different sorting orders for different columns in multi-level sorting?

Code:

```
df.sort_values(['year', 'life_exp'], ascending=[False, True])
```

Concatenating DataFrames

Often times our data is separated into multiple tables, and we would require to work with them.

Let's see a mini use-case of `users` and `messages` .

```
users = pd.DataFrame({"userid":[1, 2, 3], "name":["sharadh", "shahid", "khusalli"]})
users
```

```
msgs = pd.DataFrame({"userid":[1, 1, 2, 4], "msg":['hmm', "acha", "theek hai",
msgs
```

Can we combine these 2 DataFrames to form a single DataFrame?

Code:

```
pd.concat([users, msgs])
```

How exactly did `concat()` work?

- **By default**, `axis=0` (row-wise) for concatenation.
- `userid` , being same in both DataFrames, was **combined into a single column**.
  - First values of `users` dataframe were placed, with values of column `msg` as NaN
  - Then values of `msgs` dataframe were placed, with values of column `msg` as NaN
- The original indices of the rows were preserved.

How can we make the indices unique for each row?

Code:

```
pd.concat([users, msgs], ignore_index=True)
```

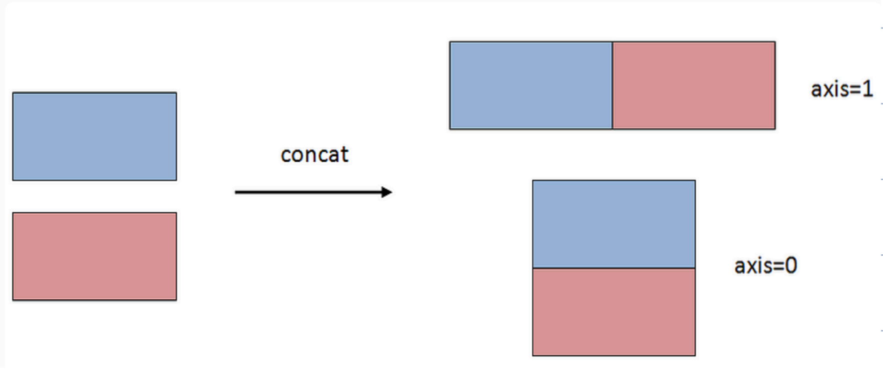
Merging DataFrames

So far we have only concatenated but not merged data.

But what is the difference between `concat` and `merge`?

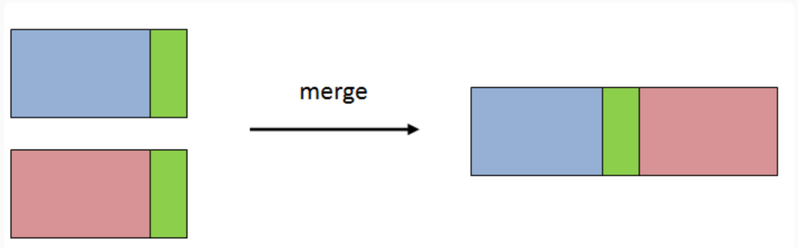
`concat`

- simply stacks multiple dataframes together along an axis.



`merge`

- combines dataframes in a **smart** way based on values in shared column(s).



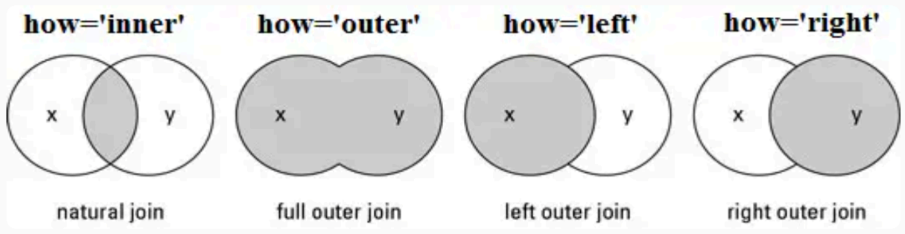
How can we join the dataframes?

Code:

```
users.merge(msgs, on="userid")
```

What type of join is this? Inner Join

Remember joins from SQL?



The `on` parameter specifies the `key`, similar to `primary key` in SQL.