Pandas Basics

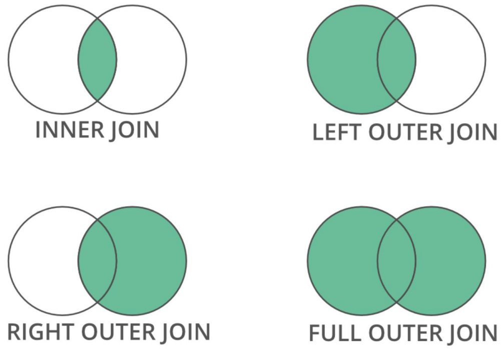
Relevel
by Unacademy

Join and Merging Dataframe

Pandas provide various facilities for efficiently combining Series or DataFrame with various kinds of set logic for the indexes.

Join can combine two data frames, but the join method combines two data frames based on their indexes. In contrast, the merge method is more versatile and allows us to specify columns beside the index to join for both data frames.

There are four ways (inner, left, right, and outer) to perform merge or join operations depending on the rows we wish to retain.



JOIN

DataFrame.join(other, on=None, how='left', lsuffix='', rsuffix='', sort=False)

By default, .join() will attempt to do a left join on indices.

- Other is the only required parameter. It defines the other DataFrame to join. You can also specify a list of DataFrames here, allowing you to combine several datasets in a single .join() call.
- On specifies an optional column or index name for the left DataFrame to join the other DataFrame's index. If it's set to None, which is the default, you'll get an index-on-index join.
- With the same options as from merge(). The difference is that it's index-based unless you also specify columns with on.
- lsuffix and rsuffix are similar to suffixes in merge(). They specify a suffix to add to any overlapping columns but have no effect when passing a list of other DataFrames.
- Sort can be enabled to sort the resulting DataFrame by the join key.

Merge

DataFrame.merge(right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'),indicator=False, validate=None)

- How defines what kind of merge to make. It defaults to 'inner', but other possible options include 'outer', 'left', and 'right'.
- On tells merge() which columns or indices, also called key columns or key indices, you want to join. This is optional. If it isn't specified, and left_index and right_index (covered below) are False, then columns from the two DataFrames that share names will be used as join keys. The column or index you specify must be present in both objects if you use one.
- left_on and right_on specify a column or index present only in the left or right object that you're merging. Both default to None.
- left_index and right_index both default to False, but if you want to use the index of the left or right object to be merged, then you can set the relevant argument to True.
- Suffixes are a tuple of strings to append to identical column names that don't merge keys. This allows you to keep track of the origins of columns with the same name.
- Sort can be enabled to sort the resulting DataFrame by the join key. If False, the order of the join keys depends on the join type (how keyword).
- An indicator, if True, add a column to the output DataFrame called "_merge" with information on the source of each row. The column can be given a different name by providing a string argument. The column will have a Categorical type with the value of "left_only" for observations whose merge key only appears in the left DataFrame, "right_only" for observations whose merge key only appears in the right DataFrame, and "both" if the observation's merge key is found in both DataFrames.

Pandas Join vs Merge Key Points

The join () method performs join on row indices and doesn't support joining on columns unless the column is set as an index.

Join () by default performs left join.

The merge () method performs join on indices, columns, and combinations of these two.

Merge () by default performs inner join.

Both these methods support inner, left, right, and outer join types. Merge additionally supports the cross join.

Notebook link - https://colab.research.google.com/drive/1zJcfq9eix2wjD7QYkC2-C8Fvr4sY8Q2h?usp=sharing

Grouping Data

Syntax: DataFrame.groupby(by=None, axis=0, level=None, as_index=True)

Parameters :

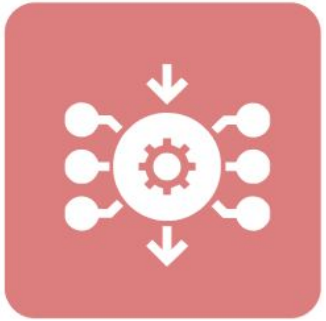
- by: mapping, function, str, or iterable
- axis: int, default 0
- level: If the axis is a MultiIndex (hierarchical), group by a particular level or levels
- as_index: For aggregated output, return an object with group labels as the index. Only relevant for DataFrame input. as_index=False is effectively "SQL-style" grouped output

Using group by, we can select a particular data category and apply functions to those categories. Groupby is a widely used technique in data science. Groupby helps to aggregate data efficiently.

Grouping Data

In the groupby below mentioned, one or more steps are involved :

1. Splitting: We can split data in the group by applying conditions to datasets
2. Applying: In this step, we apply an independent function to each group
3. Combining: In this step, we combine different datasets after applying groupby and results into a data structure

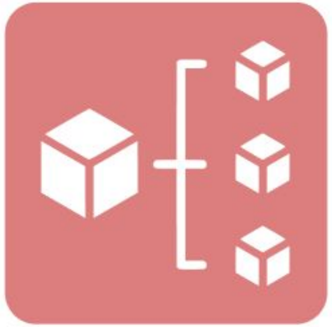


Splitting Data into Groups

To split the data, we use the groupby() function; this function is used to split the data into groups based on some criteria.

There are various ways to split data like:
obj.groupby(key)
obj.groupby(key, axis=1)
obj.groupby([key1, key2])

Note: We refer to the grouping objects as the keys in this.



Splitting Data into Groups

Grouping with one key:

One key will be passed as an argument to group the data with one key in the groupby function.
Move to jupyter notebook, for example.

Grouping data with multiple keys :

We pass various keys in the groupby function to group data with multiple keys.
Move to jupyter notebook, for example.

Splitting Data into Groups

Grouping by sorting keys :

Group keys are sorted by default using the groupby operation. Users can pass sort=False for potential speedups.
Move to jupyter notebook, for example.

Selecting groups:

To select a group, we can select a group using GroupBy.get_group(). We can select a group by applying the function GroupBy.get_group to select a single group.

Move to jupyter notebook, for example.

Applying a function to a group

After splitting data into a group, we apply a function to each group. Some operations are mentioned below that we perform :

- Aggregation
- Filtration

Aggregation: It is a process of performing statistics computations on each group. The aggregated function returns a single aggregated value for each group. [20 min]
Example: Computing group average or group sums.
Move to jupyter notebooks under section Applying aggregation functions to groups.

Filtration: It's a process in which we discard some groups based on conditional computation that gives true or false. [10 min]
For Example, Filtering data based on the group sum is above a certain threshold.
Move to jupyter notebook under the section Filtering groups for examples.

Data Handling and Manipulation

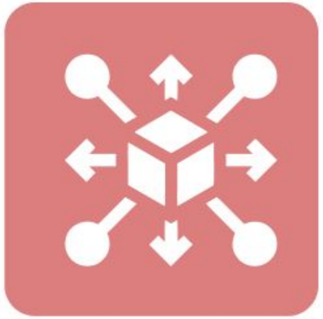
Drop Duplicates

drop_duplicates() function in pandas is used to analyze and remove duplicate data from the data frame

Syntax:
DataFrame.drop_duplicates(subset=None, keep='first', inplace=False)
Params :
Subset: list of columns is passed; only those columns will be considered for duplicates.
Keep:
It has only three distinct values :

- If 'first', it considers the first value as unique and the rest of the same values as duplicate.
- If 'last', it considers the last value unique and the rest duplicate.
- If False, it considers all of the same values as duplicates.

Move to jupyter notebook, for example, on drop duplicates.



Data Handling and Manipulation

Sorting Values

sort_values() function sorts a data frame in Descending or Ascending order of passed Column (Default is Ascending). The sorted function from Python is different from sort_values(); hence it can not work on Dataframe.

Data Handling and Manipulation

Pivot function in Pandas

The pivot function uses unique values from index/columns and fills them with values. The pivot() function is used to reshape a given DataFrame organized by given index/column values.

Syntax
DataFrame.pivot(self, index=None, columns=None, values=None)

index: Column to use to make new frame's index
columns: Column to use to make new frame columns
values: Columns to use for populating new frame's values

Move to jupyter notebook, for example, on Pivot function in pandas.

Data Handling and Manipulation

Handling Missing (NULL) values

Missing data is a common problem we face in many datasets. Some items or all the items in rows could be missing. Missing data can also be referred to as NA(Not available). Many times items in the datasets are missing because the information couldn't be collected fully while collecting data.

Example: While surveying to collect data, some people might not be comfortable sharing age but could share other information asked in the survey.

In pandas, missing values are represented by NaN or None. There are several functions for detecting, removing, and replacing null values in pandas.

Data Handling and Manipulation

Checking for missing values

isnull() function :

To check for null values in Python, we use the isnull() function. This function returns a boolean dataframe with True for NaN values.

notnull() function:

notnull() function is also used to check for null values. This function returns a boolean dataframe with False for NaN values.

Move to jupyter notebook for examples.

Data Handling and Manipulation

Removing null values

dropna() function removes rows with null values in pandas dataframe.

Move to jupyter notebook for syntax and example.
Please put the URL of nba.csv in the Removing null values example.

Filling Missing values

fillna() function
fillna() function fills the null values with the specifications given value. This method returns a new dataframe unless we mention the inplace argument to be true.

Move to jupyter notebook for syntax and example on fillna() function.

Questions

Q.1 Given below code :

```
import pandas as pd

df = pd.DataFrame({'id':[1,2,4],'features':[["A","B","C"],["A","D","E"],["C","D","F"]]})

df['features_t'] = df["features"].apply(lambda x: " ".join(["_".join(i.split(" ")) for i in x]))
```

What will be the output of the following print command?

print df['features_t']

Questions

Answer 1:

0 A B C
1 A D E
2 C D F

Questions

Q2: Given a list, output the corresponding pandas series with odd indexes only

Input :

```
given_list = [2, 4, 5, 6, 9]
index = [1, 3, 5, 7, 9]
```

Output:

```
1    2
3    4
5    5
7    6
9    9
dtype: int64
```

Questions

Answer 2:

```
given_list = [2, 4, 5, 6, 9]
series = pd.Series(given_list, index = [1, 3, 5, 7, 9])

print(series)
```

Questions

Q3. Given a dictionary, convert it into a corresponding dataframe and display it.

Input:

```
dictionary = {'name': ['Vinay', 'Kushal', 'Aman'],
              'age' : [22, 25, 24],
              'occ' : ['engineer', 'doctor', 'accountant']}
```

Output:

Dataframe

```
    name   age      occ
0   Vinay   22    engineer
1  Kushal   25     doctor
2   Aman    24   accountant
```

Questions

Answer 3:

```
dictionary = {'name': ['Vinay', 'Kushal', 'Aman'],
              'age' : [22, 25, 24],
              'occ' : ['engineer', 'doctor', 'accountant']}

dataframe = pd.DataFrame(dictionary)

print(dataframe)
```

THANK YOU