Chapter 6: Pandas

Pandas is a fast, powerful, flexible and easy to use open-source data analysis and manipulation tool, built on top of the Python programming language.
Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series.

Advantages

- Fast and efficient for manipulating and analyzing data.
- Data from different file objects can be loaded.
- Easy handling of missing data (represented as NaN)
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Data set merging and joining.
- Flexible reshaping and pivoting of data sets
- Provides time-series functionality.
- Powerful group by functionality for performing split-apply-combine operations on data sets.

There are two important data types defined by python:

- Series
- DataFrame

Series

We can say 'Series' is the single column or collection of observations in a single variable.
Pandas.Series(data,name= 'Name')

Data Frame

A data frame is a two-dimensional data structure where in the data are aligned in the tabulated form of columns and rows or it stores related columns of data.

We can create a data frame in python with the help of DataFrame keyword using the following constructor:
pandas.DataFrame(data, index, columns, dtype, copy)

The ***parameters*** are:

- Data: The data can be included with list, dictionary, series, map, ndarray, constants and also another data frame.
- Index: For the giving a label to the row, we use index. This is optional and hence if we are passing the index, it will take default indexing values that are generated from np.arange(n).

- Columns: For the column labels, we will be passing it in columns = 'label1','label2' and so on. If we are not passing it then by default it will take the values of np.range(n).
- Dtype: It is the data type for each column.
- Copy: it used to copy the data.

Reading the data from files

We will use read_csv to read the files. For the csv files, we can read it directly and use the parameter as the location of the file with the file name and the extension.
For text files, we use the following syntax:

pandas.read_csv('filename.txt', sep= ' ', header = None, names= ['col1', 'col2',…])

Parameters:

- *Filename.txt*: The name of filename and if required we have to mention the location of the file, if it is not in the same folder.
- *Sep*: It is separator between the elements to identify to which column it belongs to.
- *Header*: If the file already has the first row as the column names, we don't have to mention this parameter. By default, it will read the first line as the column names. We will use 'header = None' to create the column names which will be 0,1,2, and so on.
- *Names*: We will use 'names =' to mention the columns names which we want to create with.

Attributes of DataFrame

- Shape: It is used to display the number of rows and columns.
- Size: It is used to display the number of total elements in the data frame.
- Dtype: It is used to display the data type for each column.
- T: It is used to display the transposed data frame. That is the rows become columns and columns becomes row.
- ndim: It is used to display the number of dimensions.
- Values: It is used to display the values in numpy array form.
- Value_counts(): It is used to display the count of sample for each unique elements of the variable.
- Head(): It is used to display the first 5 rows by default. If we want more than 5, we need to mention the number within the parenthesis.
- Tail(): It is will display the last 5 rows by default and if we want to a specific number of rows, we will mention that number within the parenthesis.

Basic operations with dataframe

- Deleting a column in the data frame, we can use three methods:
- By using del function and
- By using pop function.
- By using drop function.
- Slicing the row of the data frame
- Dataframe[start:end:step]

- Adding columns
- Dataframe['New_column_name'] = values
- Deleting a row in the data frame
- Dataframe.drop(index_number,axis = 0)
- Renaming column names
- Dataframe.rename(columns = {"old_name" : "New_name" , "old_name" : "new_name",…}
- Re-ordering the columns
- We will use DataFrame function to re-arrange the column roder as per our requirement.
- Pd.DataFrame(data_frame_name, columns = ['col2','col3','col4',….]

Indexing and Slicing with dataframe

Subsetting Rows

Slicing is used with the help [] operator. It will select a set of rows and the columns from a data frame.
To slice a set of rows, we will use:

DataFrame[start:stop]

When we slice the data frame, the start bound will be included in the output, but the stop bound which we mention should be one beyond the row. So, let's say we have put the number 5, it will read up to 4.

Subsetting rows and columns

We can select a specific range of row and column from the data frame with the help of either label or integer-based indexing:

- *loc:* It is primarily a label-based indexing, but we can also use integer which will be considered as label.
- *iloc:* It primarily a integer-based indexing.

To slice, we will use as follows:

DataFrame.iloc[start:stop, start:stop] – To slice as we use start and stop
DataFrame.iloc[Row , Column] – To bring a specific element

Subsetting data using criteria

We can use the following operators to query the data according to the criteria:

- == Equals
- != Not Equals
- >or< Greater than or lesser than
- <= Lesser than or equal to
- >= Greater than or equal to

Dealing with Null values

Dropna() function:

Syntax:
Dataframename.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)
Parameters:

- *axis:* The values of axis can be given in integer or string form. It indicates the row/column. The values can be 0 or 1; or 'index' or 'column, respectively.
- *how:* It can include two values that are 'any' or 'all'. 'any' drop the row/column if ANY value is a null value and 'all' drops ALL the values if they are null.
- *thresh:* It specifies the minimum value which is to be drop.
- *subset:* It limits the dropping process to specific rows/columns.
- *inplace:* It helps in changing the dataframe, if the value is True.

Dropna function helps the user to analyze and drop the null values in different ways.

Fillna() function:

Syntax:
Dataframe.fillna(value=None, method=None, axis=None, inplace=False, limit=None, downcast=None)
Paramters:

- value : Value to use to fill holes
- method : Method to use for filling holes in re-indexed Series pad / ffill
- axis : {0 or 'index'}
- inplace : If True, fill in place.
- limit : If method is specified, this is the maximum number of consecutive NaN values to forward/backward fill
- downcast : dict, default is None
- Returns : filled : Series

Data Manipulation operators

Concat method:

This function helps us out to perform concatenation along with an axis of Pandas objects with optional set logic along another axis.

Syntax:
pandas.concat(objects, axis = 0, join = 'outer', ignore_index = False, sort = False)

- Objects: We will mention the data frames which we need to include.
- Axis: It consists of value 0 or 1.
- Join: It consists of the values inner or outer. The default is inner.

- Ignore_index: It consists of the values True or False. The default is False. If it is True, it does not uses the index values along the concatenation axis.
- Sort: It consists of the values True or False. The default is False.

Merge method:

It helps us to update the content of two data frame by merging them together based on the key columns or indices.

Syntax:

pandas.merge(left, right, how= 'inner', on = None)

- Left: dataframe
- Right:dataframe
- How: It includes left, right, outer, inner and cross. Left will consist of the values from the left frame. Right will consist of the values from the right frame. Outer will consist of all the values from both the frame. Inner will consist of the values from only in the both the frame. And cross creates a Cartesian product.