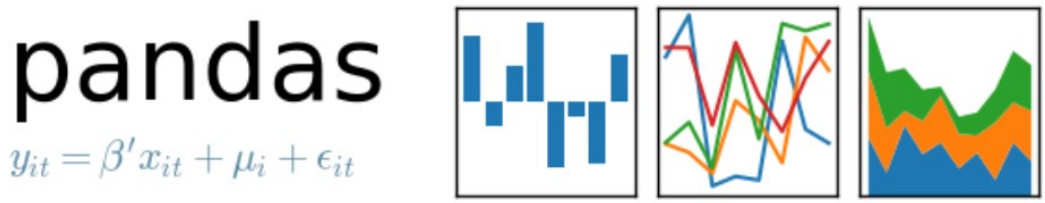
# 21 Pandas operations for absolute beginners

Prerequisites: Python and NumPy basics.

Parijat Bhatt Follow
Sep 5, 2019 · 7 min read ★




Source: https://pandas.pydata.org/

## Introduction

Pandas is an easy to use and a very powerful library for data analysis. Like NumPy, it vectorises most of the basic operations that can be parallely computed even on a CPU, resulting in faster computation. The operations specified here are very basic but too important if you are just getting started with Pandas. You will be required to import pandas as 'pd' and then use 'pd' object to perform other basic pandas operations.

1. **How to read data from a CSV file or a text file?**

A CSV file is comma-separated so in order to read a CSV file, do:

```
df = pd.read_csv(file_path, sep=',', header = 0,
index_col=False,names=None)
```

**Explanation:**

'read_csv' function has a plethora of parameters and I have specified only a few, ones that you may use most often. A few key points:

a) **header=0** means you have the names of columns in the first row in the file and if you don't you will have to specify header=None

b) **index_col = False** means to not use the first column of the data as an index in the data frame, you might want to set it to true if the first column is really an index.

c) **names = None** implies you are not specifying the column names and want it to be inferred from csv file, which means that your header = some_number contains column names. Otherwise, you can specify the names in here in the same order as you have the data in the csv file.
**If you are reading a text file separated by space or tab, you could simply change the sep to be:**

```
sep = " " or sep='\t'
```

## 2. How to create a data frame using a dictionary of pre-existing columns or NumPy 2D arrays?

Using the dictionary

```
# c1, c2, c3, c4 are column names.
d_dic =
{'first_col_name':c1,'second_col_names':c2,'3rd_col_name':c3
} df = pd.DataFrame(data = d_dic)
```

Using NumPy arrays

```
np_data = np.zeros((no_of_samples,no_of_features))
#any_numpy_array

df = pd.DataFrame(data=np_data, columns = list_of_Col_names)
```

## 3. How to visualize the top and bottom x values in a data frame?

```
df.head(num_of_rows_to_view) #top_values
df.tail(num_of_rows_to_view) #bottom_values

col = list_of_columns_to_view

df[col].head(num_of_rows_to_view)
df[col].tail(num_of_rows_to_view)
```

### 4. How to rename one or more columns?

```
df = pd.DataFrame(data={'a':[1,2,3,4,5],'b':[0,1,5,10,15]})

new_df = df.rename({'a':'new_a','b':'new_b'})
```

It is important to store the return data frame to a new data frame #as the renaming is not in-place.

### 5. How to get column names in a list?

```
df.columns.tolist()
```

Not using tolist() function also does the job if you only want to iterate over the names but it returns everything as an index object.

### 6. How to get the frequency of values in a series?

```
df[col].value_counts() #returns a mapper of key,frequency
pair

df[col].value_counts()[key] to get frequency of a key value
```

### 7. How to reset an index to an existing column or another list or array?

```
new_df = df.reset_index(drop=True,inplace=False)
```

If you do **inplace=True**, there is no need to store it to a new_df. Also, when you are resetting the index to pandas RangeIndex(), you have the option to either keep the old index or drop it with 'drop' parameter. You may want to keep it, especially when it was one of the columns originally and you temporarily set it as the newindex.

**8. How to remove a column?**

```
df.drop(columns = list_of_cols_to_drop)
```

**9. How to change the index in a data frame?**

```
df.set_index(col_name,inplace=True)
```

This will set col_name col as the index. You could pass more than one column to set them as index. inplace keyword serves the same purpose like before.

**10. How to remove rows or columns if they have nan values?**

```
df.dropna(axis=0,inplace=True)
```

axis= 0 will drop any column that has nan values, which you might not want most times. axis = 1 will drop only the rows that have nan values in any of the columns. inplace is same like above.

**11. How to slice a data frame given a condition?**

You always need to specify a mask in the form of logical conditions.
For eg, if you have column age and you would want to select the data frame

where age column has a particular value or lies in a list. Then you can achieve the slicing as follows:

```
mask = df['age'] == age_value
or
mask = df['age].isin(list_of_age_values)

result = df[mask]
```

with multiple conditions: Eg. Choosing rows where both height and age correspond to particular values.

```
mask = (df['age']==age_value) & (df['height'] ==
height_value)

result = df[mask]
```

**12. How to slice a data frame given names of columns or index values of rows?**

There are 4 options here: at, iat, loc and iloc. Among these ‘iat’ and ‘iloc’ are similar in the sense they provide integer-based indexing while ‘loc’ and ‘at’ provide name-based indexing.

Another thing to note here is that ‘iat’, at ‘provide’ indexing for single element while using ‘loc’ and ‘iloc’ one can slice more than one element.

```
Examples:

a)
df.iat[1,2] provides the element at 1th row and 2nd column.
Here it's important to note that number 1 doesn't correspond
to 1 in index column of dataframe. It's totally possible
that index in df does not have 1 at all. It's like python
array indexing.
```

```
b)
df.at[first,col_name] provides the value in the row where
index value is first and column name is col_name

c)
df.loc[list_of_indices,list_of_cols]
eg df.loc[[4,5],['age','height']]
Slices dataframe for matching indices and column names

d)
df.iloc[[0,1],[5,6]] used for interger based indexing will
return 0 and 1st row for 5th and 6th column.
```

## 13. How to iterate over rows?

```
iterrows() and itertuples()

for i,row in df.iterrows():
    sum+=row['hieght']

iterrows() passess an iterators over rows which are returned
as series. If a change is made to any of the data element of
a row, it may reflect upon the dataframe as it does not
return a copy of rows.

itertuples() returns named tuples

for row in df.itertuples():
    print(row.age)
```

## 14. How to sort by a column?

```
df.sort_values(by = list_of_cols,ascending=True)
```

## 15. How to apply a function to each element to a series?

```
df['series_name'].apply(f)
```

```
where f is the function you want to apply to each element of
the series. If you also want to pass arguments to the custom
function, you could modify it like this.

def f(x,**kwargs):
    #do_somthing
    return value_to_store

df['series_name'].apply(f, a= 1, b=2,c =3)

If you want to apply a function to more than a series, then:

def f(row):
    age = row['age']
    height = row['height']

df[['age','height']].apply(f,axis=1)
If you don't use axis=1, f will be applied to each element
of both the series. axis=1 helps to pass age and height of
each row for any manipulation you want.
```

### 16. How to apply a function to all elements in a data frame?

```
new_df = df.applymap(f)
```

### 17. How to slice a data frame if values of a series lie in a list?

Use masking and isin. To choose data samples where age lies in the list:

```
df[df['age'].isin(age_list)]
```

To chose the opposite, data samples where age does not lie in the list use:

```
df[~df['age'].isin(age_list)]
```

### 18. How to group-by column values and aggregate over another column or apply a function to it?

```
df.groupby(['age']).agg({'height':'mean'})
```

This will group the data frame by series ‘age’ and for the height column, it will apply the mean of the grouped values. Sometimes it may happen, that you would want to group-by a certain column and convert all the corresponding grouped elements for other columns into a list. You may achieve this by:

```
df.groupby(['age']).agg(list)
```

### 19. How to create duplicates for other columns for each element in a list of a particular column?

The question may be a little confusing. What I actually mean is, suppose you have the following data frame df:

```
Age Height(in cm)

20  180
20  175
18  165
18  163
16  170
```

After applying group-by with a list aggregator, you may get something like:

```
Age Height(in cm)
20  [180,175]
18  [165,163]
16  [170]
```

Now, what if you want to go back to the original data frame by undoing the last operation? You could achieve that using the newly introduced operation called explode in pandas version 0.25.

```
df['height'].explode() will give the desired outcome.
```

### 20. How to concatenate two data frames?

Suppose you have two data-frames df1 and df2 with the given columns name, age, and height and you would want to achieve the concatenation of the two columns. axis=0 is the vertical axis. Here, the result data-frame will have the columns appended from the data-frames:

```
df1 --> name,age,height
df2---> name,age,height

result = pd.concat([df1,df2],axis=0)
```

For horizontal concatenation,

```
df1--> name,age

df2--->height,salary

result = pd.concat([df1,df2], axis=1)
```

### 21. How to merge two data frames?

```
For the previous example, assume you have an employee
database forming two dataframes like

df1--> name, age, height

df2---> name, salary, pincode, sick_leaves_taken

You may want to combine these two dataframe such that each
row has all details of an employee. In order to acheive
this, you would have to perform a merge operation.

df1.merge(df2, on=['name'],how='inner')
```

```
This operation will provide a dataframe where each row will
comprise of name, age, height, salary, pincode,
sick_leaves_taken.

how = 'inner' means include the row in result if there is a
matching name in both the data frames. For more read:
https://pandas.pydata.org/pandas-
docs/stable/reference/api/pandas.DataFrame.merge.html?
highlight=merge#pandas.DataFrame.merge
```

## Wrap-up

For any data analysis project as a beginner, you may require to know these operations very well. I have always found Pandas to be a very useful library and now you can integrate with various other data analytics tools and languages. Knowing pandas operations may even help while learning languages that support distributed algorithms.

## Contact

If you liked this post, please clap and share it with others who might find it useful. I really love data science and if you are interested in it too, let's connect on LinkedIn or follow me here on towards data science platform.

Thanks to Amber Teng.