How to create new columns *derived* from existing columns in Pandas?

Answer

- We create a new column by assigning the output to the DataFrame with a new column name in between the `[]`.
- Let's say we want to create a new column `'C'` whose values are the multiplication of column `'B'` with column `'A'`. The operation will be easy to implement and will be element-wise, so there's no need to loop over rows.

```
import pandas as pd

# Create example data
df = pd.DataFrame({
  "A": [420, 380, 390],
  "B": [50, 40, 45]
})


df["C"] = df["A"] * df["B"]
```

- Also other mathematical operators (`+`, `-`, `\*`, `/`) or logical operators (`<`, `>`, `=`,...) work element-wise. But if we need more advanced logic, we can use arbitrary Python code via `apply()`.
- Depending on the case, we can use `rename` with a dictionary or function to rename row labels or column names according to the problem.

Define the different ways a DataFrame can be created in pandas?

We can create a DataFrame using following ways:

- Lists
- Dict of ndarrays

Example-1: Create a DataFrame using List:

1. **import** pandas as pd
2. # a list of strings
3. a = ['Python', 'Pandas']

```
# Calling DataFrame constructor on list
info = pd.DataFrame(a)
print(info)
```

Example-2: Create a DataFrame from dict of ndarrays:

```
import pandas as pd
info = {'ID' :[101, 102, 103],'Department' :['B.Sc','B.Tech','M.Tech',]}
info = pd.DataFrame(info)
print (info)
```

How will you create a series from dict in Pandas?

A Series is defined as a one-dimensional array that is capable of storing various data types.

We can create a Pandas Series from Dictionary:

Create a Series from dict:

We can also create a Series from dict. If the dictionary object is being passed as an input and the index is not specified, then the dictionary keys are taken in a sorted order to construct the index.

If index is passed, then values correspond to a particular label in the index will be extracted from the dictionary.

```
import pandas as pd
import numpy as np
info = {'x' : 0., 'y' : 1., 'z' : 2.}
a = pd.Series(info)
print (a)
```

How can we create a copy of the series in Pandas?

We can create the copy of series by using the following syntax:

pandas.Series.copy

Series.copy(deep=True)

The above statements make a deep copy that includes a copy of the data and the indices. If we set the value of deep to **False**, it will neither copy the indices nor the data.

How will you create an empty DataFrame in Pandas?

A DataFrame is a widely used data structure of pandas and works with a two-dimensional array with labeled axes (rows and columns) It is defined as a standard way to store data and has two different indexes, i.e., row index and column index.

Create an empty DataFrame:

The below code shows how to create an empty DataFrame in Pandas:

```
# importing the pandas library
import pandas as pd
info = pd.DataFrame()
print (info)
```

How will you add a column to a pandas DataFrame?

We can add any new column to an existing DataFrame. The below code demonstrates how to add any new column to an existing DataFrame:

```
# importing the pandas library
import pandas as pd
info = {'one' : pd.Series([1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e']),
        'two' : pd.Series([1, 2, 3, 4, 5, 6], index=['a', 'b', 'c', 'd', 'e', 'f'])}

info = pd.DataFrame(info)

# Add a new column to an existing DataFrame object

print ("Add new column by passing series")
info['three']=pd.Series([20,40,60],index=['a','b','c'])
```

```
print (info)
print ("Add new column using existing DataFrame columns")
info['four']=info['one']+info['three']
print (info)
```

How to get the items of series A not present in series B?

We can remove items present in **p2** from **p1** using isin() method.

```
import pandas as pd
p1 = pd.Series([2, 4, 6, 8, 10])
p2 = pd.Series([8, 10, 12, 14, 16])
p1[~p1.isin(p2)]
```

How to get the items not common to both series A and series B?

We get all the items of p1 and p2 not common to both using below example:

```
import pandas as pd
import numpy as np
p1 = pd.Series([2, 4, 6, 8, 10])
p2 = pd.Series([8, 10, 12, 14, 16])
p1[~p1.isin(p2)]
p_u = pd.Series(np.union1d(p1, p2))  # union
p_i = pd.Series(np.intersect1d(p1, p2))  # intersect
p_u[~p_u.isin(p_i)]
```

How to get the minimum, 25th percentile, median, 75th, and max of a numeric series?

We can compute the minimum, 25th percentile, median, 75th, and maximum of **p** as below example:

```
import pandas as pd
import numpy as np
```

```
3. p = pd.Series(np.random.normal(14, 6, 22))
4. state = np.random.RandomState(120)
5. p = pd.Series(state.normal(14, 6, 22))
6. np.percentile(p, q=[0, 25, 50, 75, 100])
```

How to get frequency counts of unique items of a series?

We can calculate the frequency counts of each unique value p as below example:

```
1. import pandas as pd
2. import numpy as np
3. p= pd.Series(np.take(list('pqrstu'), np.random.randint(6, size=17)))
4. p = pd.Series(np.take(list('pqrstu'), np.random.randint(6, size=17)))
5. p.value_counts()
```

How to convert a numpy array to a dataframe of given shape?

We can reshape the series **p** into a dataframe with 6 rows and 2 columns as below example:

```
1. import pandas as pd
2. import numpy as np
3. p = pd.Series(np.random.randint(1, 7, 35))
4. # Input
5. p = pd.Series(np.random.randint(1, 7, 35))
6. info = pd.DataFrame(p.values.reshape(7,5))
7. print(info)
```

How can we convert a Series to DataFrame?

The Pandas **Series.to_frame()** function is used to convert the series object to the DataFrame.

```
1. Series.to_frame(name=None)
```

name: Refers to the object. Its Default value is None. If it has one value, the passed name will be substituted for the series name.

```
s = pd.Series(["a", "b", "c"],
name="vals")
s.to_frame()
```

How can we sort the DataFrame?

We can efficiently perform sorting in the DataFrame through different kinds:

- By label
- By Actual value

By label

The DataFrame can be sorted by using the sort_index() method. It can be done by passing the axis arguments and the order of sorting. The sorting is done on row labels in ascending order by default.

By Actual Value

It is another kind through which sorting can be performed in the DataFrame. Like index sorting, **sort_values()** is a method for sorting the values.

It also provides a feature in which we can specify the column name of the DataFrame with which values are to be sorted. It is done by passing the '**by**' argument.

What is Time Series in Pandas?

The Time series data is defined as an essential source for information that provides a strategy that is used in various businesses. From a conventional finance industry to the education industry, it consists of a lot of details about the time.

Time series forecasting is the machine learning modeling that deals with the Time Series data for predicting future values through Time Series modeling.

How to convert String to date?

The below code demonstrates how to convert the string to date:

```
fromdatetime import datetime

# Define dates as the strings
```

```
dmy_str1 = 'Wednesday, July 14, 2018'
dmy_str2 = '14/7/17'
dmy_str3 = '14-07-2017'

# Define dates as the datetime objects
dmy_dt1 = datetime.strptime(date_str1, '%A, %B %d, %Y')
dmy_dt2 = datetime.strptime(date_str2, '%m/%d/%y')
dmy_dt3 = datetime.strptime(date_str3, '%m-%d-%Y')

#Print the converted dates
print(dmy_dt1)
print(dmy_dt2)
print(dmy_dt3)
```

What is Data Aggregation?

The main task of Data Aggregation is to apply some aggregation to one or more columns. It uses the following:

- **sum:** It is used to return the sum of the values for the requested axis.
- **min:** It is used to return a minimum of the values for the requested axis.
- **max:** It is used to return a maximum values for the requested axis.

Describe Data Operations in Pandas?

In Pandas, there are different useful data operations for DataFrame, which are as follows:

- **Row and column selection**

We can select any row and column of the DataFrame by passing the name of the rows and columns. When you select it from the DataFrame, it becomes one-dimensional and considered as Series.

- **Filter Data**

We can filter the data by providing some of the boolean expressions in DataFrame.

- **Null values**

A Null value occurs when no data is provided to the items. The various columns may contain no values, which are usually represented as NaN.

Define GroupBy in Pandas?

In Pandas, **groupby()** function allows us to rearrange the data by utilizing them on real-world data sets. Its primary task is to split the data into various groups. These groups are categorized based on some criteria. The objects can be divided from any of their axes.

DataFrame.groupby(by=None, axis=0, level=None, as_index=True, sort=True, group_keys=True, squeeze=False, **kwargs)

Explain Categorical Data In Pandas?

Categorical data refers to real-time data that can be repetitive; for instance, data values under categories such as country, gender, codes will always be repetitive. Categorical values in pandas can also take only a limited and fixed number of possible values.

Numerical operations cannot be performed on such data. All values of categorical data in pandas are either in categories or np.nan.