Date Functionality

Extending the Time series, Date functionalities play major role in financial data analysis. While working with Date data, we will frequently come across the following −

- Generating sequence of dates
- Convert the date series to different frequencies

Create a Range of Dates

Using the **date.range()** function by specifying the periods and the frequency, we can create the date series. By default, the frequency of range is Days.

Live Demo

```
import pandas as pd

print pd.date_range('1/1/2011', periods=5)
```

Its **output** is as follows −

```
DatetimeIndex(['2011-01-01', '2011-01-02', '2011-01-03', '2011-01-
04', '2011-01-05'],
   dtype='datetime64[ns]', freq='D')
```

Change the Date Frequency

Live Demo

```
import pandas as pd

print pd.date_range('1/1/2011', periods=5,freq='M')
```

Its **output** is as follows −

```
DatetimeIndex(['2011-01-31', '2011-02-28', '2011-03-31', '2011-04-
30', '2011-05-31'],
   dtype='datetime64[ns]', freq='M')
```

bdate_range

bdate_range() stands for business date ranges. Unlike date_range(), it excludes Saturday and Sunday.

Live Demo

```
import pandas as pd

print pd.date_range('1/1/2011', periods=5)
```

Its **output** is as follows −

```
DatetimeIndex(['2011-01-01', '2011-01-02', '2011-01-03', '2011-01-
04', '2011-01-05'],
   dtype='datetime64[ns]', freq='D')
```

Observe, after 3rd March, the date jumps to 6th march excluding 4th and 5th. Just check your calendar for the days.

Convenience functions like **date_range** and **bdate_range** utilize a variety of frequency aliases. The default frequency for date_range is a calendar day while the default for bdate_range is a business day.

Live Demo

```
import pandas as pd
start = pd.datetime(2011, 1, 1)
end = pd.datetime(2011, 1, 5)

print pd.date_range(start, end)
```

Its **output** is as follows −

```
DatetimeIndex(['2011-01-01', '2011-01-02', '2011-01-03', '2011-01-
04', '2011-01-05'],
   dtype='datetime64[ns]', freq='D')
```

Offset Aliases

A number of string aliases are given to useful common time series frequencies. We will refer to these aliases as offset aliases.

Alias	Description	Alias	Description
B	business day frequency	BQS	business quarter start frequency
D	calendar day frequency	A	annual(Year) end frequency
W	weekly frequency	BA	business year end frequency
M	month end frequency	BAS	business year start frequency
SM	semi-month end frequency	BH	business hour frequency
BM	business month end frequency	H	hourly frequency

MS	month start frequency	T, min	minutely frequency
SMS	SMS semi month start frequency	S	secondly frequency
BMS	business month start frequency	L, ms	milliseconds
Q	quarter end frequency	U, us	microseconds
BQ	business quarter end frequency	N	nanoseconds
QS	quarter start frequency		

Categorical Data

Often in real-time, data includes the text columns, which are repetitive. Features like gender, country, and codes are always repetitive. These are the examples for categorical data.

Categorical variables can take on only a limited, and usually fixed number of possible values. Besides the fixed length, categorical data might have an order but cannot perform numerical operation. Categorical are a Pandas data type.

The categorical data type is useful in the following cases −

- A string variable consisting of only a few different values. Converting such a string variable to a categorical variable will save some memory.
- The lexical order of a variable is not the same as the logical order (“one”, “two”, “three”). By converting to a categorical and specifying an order on the categories, sorting and min/max will use the logical order instead of the lexical order.
- As a signal to other python libraries that this column should be treated as a categorical variable (e.g. to use suitable statistical methods or plot types).

Object Creation

Categorical object can be created in multiple ways. The different ways have been described below −

category

By specifying the dtype as "category" in pandas object creation.

Live Demo

```
import pandas as pd

s = pd.Series(["a","b","c","a"], dtype="category")
print s
```

Its **output** is as follows −

```
0  a
1  b
2  c
3  a
dtype: category
Categories (3, object): [a, b, c]
```

The number of elements passed to the series object is four, but the categories are only three. Observe the same in the output Categories.

pd.Categorical

Using the standard pandas Categorical constructor, we can create a category object.

```
pandas.Categorical(values, categories, ordered)
```

Let's take an example −

Live Demo

```
import pandas as pd

cat = pd.Categorical(['a', 'b', 'c', 'a', 'b', 'c'])
print cat
```

Its **output** is as follows −

```
[a, b, c, a, b, c]
Categories (3, object): [a, b, c]
```

Let's have another example −

Live Demo

```
import pandas as pd

cat = cat=pd.Categorical(['a','b','c','a','b','c','d'], ['c', 'b',
'a'])
print cat
```

Its **output** is as follows −

```
[a, b, c, a, b, c, NaN]
Categories (3, object): [c, b, a]
```

Here, the second argument signifies the categories. Thus, any value which is not present in the categories will be treated as **NaN**.

Now, take a look at the following example −

Live Demo

```
import pandas as pd

cat = cat=pd.Categorical(['a','b','c','a','b','c','d'], ['c', 'b',
'a'],ordered=True)
print cat
```

Its **output** is as follows −

```
[a, b, c, a, b, c, NaN]
Categories (3, object): [c < b < a]
```

Logically, the order means that, **a** is greater than **b** and **b** is greater than **c**.

Description

Using the **.describe()** command on the categorical data, we get similar output to a **Series** or **DataFrame** of the **type** string.

Live Demo

```
import pandas as pd
import numpy as np

cat = pd.Categorical(["a", "c", "c", np.nan], categories=["b", "a",
"c"])
df = pd.DataFrame({"cat":cat, "s":["a", "c", "c", np.nan]})

print df.describe()
print df["cat"].describe()
```

Its **output** is as follows −

```
        cat s
count     3 3
unique    2 2
top       c c
freq      2 2
count     3
unique    2
top       c
freq      2
Name: cat, dtype: object
```

Get the Properties of the Category

obj.cat.categories command is used to get the **categories of the object**.

Live Demo

```
import pandas as pd
import numpy as np

s = pd.Categorical(["a", "c", "c", np.nan], categories=["b", "a",
"c"])
print s.categories
```

Its **output** is as follows −

```
Index([u'b', u'a', u'c'], dtype='object')
```

obj.ordered command is used to get the order of the object.

Live Demo

```
import pandas as pd
import numpy as np

cat = pd.Categorical(["a", "c", "c", np.nan], categories=["b", "a",
"c"])
print cat.ordered
```

Its **output** is as follows −

```
False
```

The function returned **false** because we haven't specified any order.

Renaming Categories

Renaming categories is done by assigning new values to the **series.cat.categories**series.cat.categories property.

Live Demo

```
import pandas as pd

s = pd.Series(["a","b","c","a"], dtype="category")
s.cat.categories = ["Group %s" % g for g in s.cat.categories]
print s.cat.categories
```

Its **output** is as follows −

```
Index([u'Group a', u'Group b', u'Group c'], dtype='object')
```

Initial categories **[a,b,c]** are updated by the **s.cat.categories** property of the object.

Appending New Categories

Using the Categorical.add.categories() method, new categories can be appended.

Live Demo

```
import pandas as pd

s = pd.Series(["a","b","c","a"], dtype="category")
s = s.cat.add_categories([4])
print s.cat.categories
```

Its **output** is as follows −

```
Index([u'a', u'b', u'c', 4], dtype='object')
```

Removing Categories

Using the **Categorical.remove_categories()** method, unwanted categories can be removed.

Live Demo

```
import pandas as pd

s = pd.Series(["a","b","c","a"], dtype="category")
print ("Original object:")
print s

print ("After removal:")
print s.cat.remove_categories("a")
```

Its **output** is as follows −

```
Original object:
0  a
1  b
2  c
3  a
dtype: category
Categories (3, object): [a, b, c]

After removal:
0  NaN
1  b
2  c
3  NaN
dtype: category
Categories (2, object): [b, c]
```

Comparison of Categorical Data

Comparing categorical data with other objects is possible in three cases −

- comparing equality (== and !=) to a list-like object (list, Series, array, ...) of the same length as the categorical data.
- all comparisons (==, !=, >, >=, <, and <=) of categorical data to another categorical Series, when ordered==True and the categories are the same.
- all comparisons of a categorical data to a scalar.

Take a look at the following example −

Live Demo

```
import pandas as pd

cat = pd.Series([1,2,3]).astype("category", categories=[1,2,3],
ordered=True)
cat1 = pd.Series([2,2,2]).astype("category", categories=[1,2,3],
ordered=True)

print cat>cat1
```

Its **output** is as follows −

```
0  False
1  False
2  True
dtype: bool
```