Efficient Python Tricks and Tools for Data Scientists

Work with Datetime in pandas - By Khuyen Tran

parse_dates: Convert Columns into Datetime When Using pandas to Read CSV Files

If there are datetime columns in your CSV file, use the `parse_dates` parameter when reading CSV file with pandas. This reduces one extra step to convert these columns from string to datetime after reading the file.

```
import pandas as pd

df = pd.read_csv("data1.csv", parse_dates=
["date_column_1", "date_column_2"])
```

```
df
```

	date_column_1	date_column_2	value
0	2021-02-10	2021-02-11	3
1	2021-02-12	2021-02-13	3

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 3 columns):
 #   Column          Non-Null Count  Dtype

---  ------          --------------  -----

 0   date_column_1  2 non-null
datetime64[ns]
 1   date_column_2  2 non-null
datetime64[ns]
 2   value           2 non-null      int64

dtypes: datetime64[ns](2), int64(1)
memory usage: 176.0 bytes
```

pandas' DateOffset: Add a Time Interval to a pandas Timestamp

If you want to add days, months, or other time intervals to a pandas `Timestamp`, use `pd.DateOffset`.

```
import pandas as pd
from pandas.tseries.offsets import DateOffset,
BDay

ts = pd.Timestamp('2021-10-10 9:00:00')

# Increase the timestamp by 3 months
ts + DateOffset(months=3)
```

```
Timestamp('2022-01-10 09:00:00')
```

```
# Increase the timestamp by 3 years and 3
hours
ts + DateOffset(years=3, hours=3)
```

```
Timestamp('2024-10-10 12:00:00')
```

You can also increase the timestamp by `n` business days using `BDay`.

```
# Increase the timestamp by 6 business days
ts + BDay(n=6)
```

```
Timestamp('2021-10-18 09:00:00')
```

Link to pandas DateOffset.

DataFrame rolling: Find The Average of The Previous n Datapoints Using pandas

If you want to find the average of the previous n data points (simple moving average) with pandas, use df.rolling(time_period).mean().

The code below shows how to find the simple moving average of the previous 3 data-points.

```
from datetime import date
import pandas as pd

df = pd.DataFrame(
    {
        "date": [
            date(2021, 1, 20),
            date(2021, 1, 21),
            date(2021, 1, 22),
            date(2021, 1, 23),
            date(2021, 1, 24),
        ],
        "value": [1, 2, 3, 4, 5],
    }
).set_index("date")

df
```

	value
date	
2021-01-20	1
2021-01-21	2
2021-01-22	3
2021-01-23	4
2021-01-24	5

```
df.rolling(3).mean()
```

	value
date	
2021-01-20	NaN
2021-01-21	NaN
2021-01-22	2.0
2021-01-23	3.0
2021-01-24	4.0

pandas.Series.dt: Access Datetime Properties of a pandas Series

The easiest way to access datetime properties of pandas Series values is to use `pandas.Series.dt`.

```
import pandas as pd

df = pd.DataFrame({"date": ["2021/05/13 15:00", "2022-6-20 14:00"], "values": [1, 3]})

df["date"] = pd.to_datetime(df["date"])

df["date"].dt.year
```

```
0    2021
1    2022
Name: date, dtype: int64
```

```
df["date"].dt.time
```

```
0    15:00:00
1    14:00:00
Name: date, dtype: object
```

Get Rows within a Year Range

If you want to get all data starting in a particular year and exclude the previous years, simply use `df.loc['year':]` like below. This works when the index of your `pd.Dataframe` is `DatetimeIndex`.

```
from datetime import datetime
import pandas as pd

df = pd.DataFrame(
    {
        "date": [datetime(2018, 10, 1),
datetime(2019, 10, 1), datetime(2020, 10, 1)],
        "val": [1, 2, 3],
    }
).set_index("date")

df
```

	val
date	
2018-10-01	1
2019-10-01	2
2020-10-01	3

```
df.loc["2019":]
```

	val
date	
2019-10-01	2
2020-10-01	3

pandas.reindex: Replace the Values of the Missing Dates with 0

Have you ever got a time series with missing dates? This can cause a problem since many time series methods require a fixed frequency index.

To fix this issue, you can replace the values of the missing dates with 0 using `pd.date_range` and `pd.reindex`.

```
import pandas as pd

s = pd.Series([1, 2, 3], index=["2021-07-20",
"2021-07-23", "2021-07-25"])
s.index = pd.to_datetime(s.index)
s
```

```
2021-07-20    1
2021-07-23    2
2021-07-25    3
dtype: int64
```

```
# Get dates ranging from 2021/7/20 to
2021/7/25
new_index = pd.date_range("2021-07-20", "2021-
07-25")

# Conform Series to new index
new_s = s.reindex(new_index, fill_value=0)
new_s
```

```
2021-07-20    1
2021-07-21    0
2021-07-22    0
2021-07-23    2
2021-07-24    0
2021-07-25    3
Freq: D, dtype: int64
```

Select DataFrame Rows Before or After a Specific Date

If you want to get the rows whose dates are before or after a specific date, use the comparison operator and a date string.

```
import pandas as pd

df = pd.DataFrame(
    {"date": pd.date_range(start="2021-7-19",
end="2021-7-23"), "value": list(range(5))}
)
df
```

	date	value
0	2021-07-19	0
1	2021-07-20	1
2	2021-07-21	2
3	2021-07-22	3
4	2021-07-23	4

```
filtered_df = df[df.date <= "2021-07-21"]
filtered_df
```

	date	value
0	2021-07-19	0
1	2021-07-20	1
2	2021-07-21	2

resample: Resample Time-Series Data

If you want to change the frequency of time-series data, use `resample`. In the code below, I use `resample` to show the records every two days instead of every day.

```
import pandas as pd
from numpy.random import randint

index = pd.date_range("2022-02-01", "2022-02-6")
s = pd.Series(index=index, data=randint(0, 10, 6))
s
```

```
2022-02-01    7
2022-02-02    0
2022-02-03    6
2022-02-04    2
2022-02-05    9
2022-02-06    2
Freq: D, dtype: int64
```

```
s.resample('2D').sum()
```

```
2022-02-01     7
2022-02-03     8
2022-02-05    11
Freq: 2D, dtype: int64
```