All you Should Know About Datetime Variables in Python and Pandas

Home

Aniruddha Bhandari — May 6, 2020

Beginner Libraries Python Structured Data

The Complex yet Powerful World of DateTime in Data Science

I still remember coming across my first DateTime variable when I was learning Python. It was an e-commerce project where I had to figure out the supply chain pipeline – the time it takes for an order to be shipped, the number of days it takes for an order to be delivered, etc. It was quite a fascinating problem from a data science perspective.

The issue – I wasn't familiar with how to extract and play around with the date and time components in Python.

There is an added complexity to the DateTime features, an extra layer that isn't present in numerical variables. Being able to master these DateTime features will help you go a long way towards becoming a better (and more efficient) data scientist. It's definitely helped me a lot!

And the date and time features are ubiquitous in data science projects. Think about it – they are a rich source of valuable information, and hence, can give some deep insights about any dataset at hand. Plus the amount of flexibility they offer when we're performing feature engineering – priceless!

In this article, we will first have a look at how to handle date and time features with **Python's *DateTime* module** and then we will explore **Pandas functions** for the same!

Note: I assume you're familiar with Python and the Pandas library. If not, I highly recommend taking the awesome free courses below:

- *Python for Business Analytics and Data Science*
- *Pandas for Data Analysis in Python*

All you Should Know About Datetime Variables in Python and Pandas

The Importance of the Date-Time Component

It's worth reiterating, dates and times are a treasure trove of information and that is why data scientists love them so much.

Before we dive into the crux of the article, I want you to experience this yourself. Take a look at the date and time right now. Try and imagine all kinds of information that you can extract from it to understand your reading habit. The year, month, day, hour, and minute are the usual suspects.

But if you dig a little further, you can determine whether you prefer reading on weekdays or weekends, whether you are a morning person or a night owl (we are in the same boat here!), or whether you accumulate all the interesting articles to read at the end of the month!

Clearly, the list will go on and you will gradually learn a lot about your reading habits if you repeat this exercise after collecting the data over a period of time, say a month. Now imagine how useful this feature would be in a real-world scenario where information is collected over a long period of time.

Date and time features find importance in data science problems spanning industries from sales, marketing, and finance to HR, e-commerce, retail, and many more. Predicting how the stock markets will behave tomorrow, how many products will be sold in the upcoming week, when is the best time to launch a new product, how long before a position at the company gets filled, etc. are some of the problems that we can find answers to using date and time data.

This incredible amount of insight that you can unravel from the data is what makes date and time components so fun to work with! So let's get down to the business of mastering date-time manipulation in Python.

All you Should Know About Datetime Variables in Python and Pandas

arguments: year, month, and day. Let's have a look at how it's done:

```
from datetime import date

d1 = date(2020,4,23)

print(d1)

print(type(d1))
```

datetime1.py hosted with ❤ by GitHub view raw

```
2020-04-23
<class 'datetime.date'>
```

You can see how easy it was to create a date object of **datetime** class. And it's even easier to extract features like day, month, and year from the date. This can be done using the **day**, **month**, and **year** attributes. We will see how to do that on the current local day date object that we will create using the **today()** function:

```
# present day date
d1 = date.today()
print(d1)
# day
print('Day :',d1.day)
# month
print('Month :',d1.month)
# year
print('Year :',d1.year)
```

datetime2.py hosted with ❤ by GitHub view raw

```
2020-04-23
Day : 23
Month : 4
Year : 2020
```

Working with Time in Python

time is another class of the DateTime module that accepts integer arguments for time up to microseconds and returns a DateTime object:

```
from datetime import time

t1 = time(13,20,13,40)

print(t1)

print(type(t1))
```

datetime3.py hosted with ❤ by GitHub view raw

```
13:20:13.000040
<class 'datetime.time'>
```

You can extract features like **hour**, **minute**, **second**, and **microsecond** from the time object using the respective attributes. Here is an example:

```
# hour
```

All you Should Know About Datetime Variables in Python and Pandas

```
# microsecond
print('Microsecond :',t1.microsecond)
```

datetime4.py hosted with ❤ by GitHub view raw

```
Hour : 13
Minute : 20
Second : 13
Microsecond : 40
```

This is just the tip of the iceberg. There is so much more we can do with DateTime features in Python and that's what we'll look at in the next section.

DateTime in Python

So far, we have seen how to create a date and a time object using the DateTime module. But the beauty of the DateTime module is that it lets you dovetail both the properties into a single object, **DateTime**!

datetime is a class and an object in Python's *DateTime* module, just like date and time. The arguments are a combination of date and time attributes, starting from the year and ending in microseconds.

So, let's see how you can create a **DateTime** object:

```
from datetime import datetime
d1 = datetime(2020,4,23,11,20,30,40)
print(d1)
print(type(d1))
```

datetime5.py hosted with ❤ by GitHub view raw

```
2020-04-23 11:20:30.000040
<class 'datetime.datetime'>
```

Or you could even create an object on the local date and time using the **now()** method:

```
# local date-time
d1 = datetime.now()
d1
```

datetime6.py hosted with ❤ by GitHub view raw

```
datetime.datetime(2020, 4, 23, 13, 23, 22, 684340)
```

You can go on and extract whichever value you want to from the DateTime object using the same attributes we used with the date and time objects individually.

Next, let's look at some of the methods in the DateTime class.

Updating old Dates

First, we'll see how to separate date and time from the DateTime object using the **date()** and **time()** methods. But you could also replace a value in the DateTime objects without having to change the entire date using the **replace()** method:

```
print('Datetime :',d1)
# date
```

```
datetime7.py hosted with ❤ by GitHub view raw
```

```
Datetime : 2020-04-23 13:23:22.684340
Date : 2020-04-23
Time : 13:23:22.684340
New datetime : 2020-04-24 14:23:22.684340
```

Weekday from DateTime

One really cool thing that you can do with the DateTime function is to extract the day of the week! This is especially helpful in feature engineering because the value of the target variable can be dependent on the day of the week, like sales of a product are generally higher on a weekend or traffic on StackOverflow could be higher on a weekday when people are working, etc.

The **weekday()** method returns an integer value for the day of the week, where Monday is 0 and Sunday is 6. But if you wanted it to return the weekday value between 1 and 7, like in a real-world scenario, you should use **isoweekday()**:

```
1 d1 = datetime.now()
2 # week starts from 0
3 print(d1.weekday()) # output 3 for Thurday
4 # week starts with 1
5 print(d1.isoweekday()) # output 4 in ISO format
```

```
datetime8.py hosted with ❤ by GitHub view raw
```

```
3
4
```

What Week is it?

Alright, you know the day of the week, but do you know what week of the year is it? This is another very important feature that you can generate from the given date in a dataset.

Sometimes the value of the target variable might be higher during certain times of the year. For example, the sales of products on e-commerce websites are generally higher during vacations.

You can get the week of the year by slicing the value returned by the **isocalendar()** method:

```
1 d1 = datetime.now()
2 # retuns year, week, month
3 print(d1.isocalendar())
4 print('Week :',d1.isocalendar()[1])
```

```
datetime9.py hosted with ❤ by GitHub view raw
```

```
(2020, 17, 4)
Week : 17
```

Leap Year or Not? Use Calendar!

Want to check whether it is a leap year or not? You will need to use the **isleap()** method from the [calendar] module and pass the year as an attribute:

```
1 import calendar
```

All you Should Know About Datetime Variables in Python and Pandas

Congratulations – you are living in a leap year! What did you do with the extra day? Oh, you missed it? Don't worry! Just take a day this month and do the stuff that you love! But where are you going? You got your calendar right here!

```
# Its was April when I wrote this
print(calendar.month(2020,4))
```

datetime11.py hosted with ❤ by **GitHub** **view raw**

```
    April 2020
Mo Tu We Th Fr Sa Su
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

Not free this month? You can have a look at the entire calendar for the year:

```
print(calendar.calendar(2020))
```

datetime12.py hosted with ❤ by **GitHub** **view raw**

```
                                  2020

      January               February                March
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
       1  2  3  4  5                  1  2                     1
 6  7  8  9 10 11 12   3  4  5  6  7  8  9   2  3  4  5  6  7  8
13 14 15 16 17 18 19  10 11 12 13 14 15 16   9 10 11 12 13 14 15
20 21 22 23 24 25 26  17 18 19 20 21 22 23  16 17 18 19 20 21 22
27 28 29 30 31        24 25 26 27 28 29     23 24 25 26 27 28 29
                                            30 31

       April                 May                   June
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
       1  2  3  4  5                  1  2  3   1  2  3  4  5  6  7
 6  7  8  9 10 11 12   4  5  6  7  8  9 10   8  9 10 11 12 13 14
13 14 15 16 17 18 19  11 12 13 14 15 16 17  15 16 17 18 19 20 21
20 21 22 23 24 25 26  18 19 20 21 22 23 24  22 23 24 25 26 27 28
27 28 29 30           25 26 27 28 29 30 31  29 30

       July                 August              September
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
       1  2  3  4  5                     1  2      1  2  3  4  5  6
 6  7  8  9 10 11 12   3  4  5  6  7  8  9   7  8  9 10 11 12 13
13 14 15 16 17 18 19  10 11 12 13 14 15 16  14 15 16 17 18 19 20
20 21 22 23 24 25 26  17 18 19 20 21 22 23  21 22 23 24 25 26 27
27 28 29 30 31        24 25 26 27 28 29 30  28 29 30
                      31

      October              November              December
Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su  Mo Tu We Th Fr Sa Su
          1  2  3  4                     1      1  2  3  4  5  6
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   7  8  9 10 11 12 13
12 13 14 15 16 17 18   9 10 11 12 13 14 15  14 15 16 17 18 19 20
19 20 21 22 23 24 25  16 17 18 19 20 21 22  21 22 23 24 25 26 27
26 27 28 29 30 31     23 24 25 26 27 28 29  28 29 30 31
                      30
```

Pretty cool, right? Plan your year wisely and take out some time to do the things you love!

DateTime Formats

The **Datetime** module lets you interchange the format of DateTime between a few options.

First up is the ISO format. If you wanted to create a DateTime object from the string form of the date in ISO format, use the **fromisoformat()** method. And if you intended to do the reverse, use the **isoformat()** method:

```
# ISO format
d1_datetime = date.fromisoformat('2020-04-23')
print(d1_datetime)
print(type(d1_datetime))
d1_ISO = date(2020,4,23).isoformat()
print(d1_ISO)
```

All you Should Know About Datetime Variables in Python and Pandas

```
2020-04-23
<class 'str'>
```

If you wanted to convert DateTime into a string format, you could use the **ctime()** method. This returns the date in a string format. And if you wanted to extract just the date from that, well, you would have to use slicing:

```
# date in string format
d1 = datetime.now()
# string format for date
print(d1.ctime())
# slicing to extract date
print(d1.ctime()[:10])
```

datetime14.py hosted with ❤ by GitHub view raw

```
Thu Apr 23 14:34:20 2020
Thu Apr 23
```

And if none of these functions strike your fancy, you could use the **format()** method which lets you define your own format:

```
date(2020,4,23).__format__('%Y/%m/%d')
```

datetime15.py hosted with ❤ by GitHub view raw

```
'2020/04/23'
```

Wait – what are these arguments I passed to the function? These are called formatted string codes and we will look at them in detail in the next section.

Advanced DateTime Formatting with Strptime & Strftime

These functions are very important as they let you define the format of the DateTime object explicitly. This can give you a lot of flexibility with handling DateTime features.

strptime() creates a DateTime object from a string representing date and time. It takes two arguments: the date and the format in which your date is present. Have a look below:

```
# strptime
date = '22 April, 2020 13:20:13'
d1 = datetime.strptime(date,'%d %B, %Y %H:%M:%S')
print(d1)
print(type(d1))
```

datetime16.py hosted with ❤ by GitHub view raw

```
2020-04-22 13:20:13
<class 'datetime.datetime'>
```

You define the format using the formatting codes as I did above. There are a number of formatting codes and you can have a look at them in the documentation.

The **stftime()** method, on the other hand, can be used to convert the DateTime object into a string representing date and time:

```
# strftime
d1 = datetime.now()
print('Datetime object :',d1)
new_date = d1.strftime('%d/%m/%Y %H:%M')
print('Formatted date :',new_date)
```

All you Should Know About Datetime Variables in Python and Pandas

```
<class 'str'>
```

But you can also extract some important information from the DateTime object like weekday name, month name, week number, etc. which can turn out to be very useful in terms of features as we saw in previous sections.

```
d1 = datetime.now()
print('Weekday :',d1.strftime('%A'))
print('Month :',d1.strftime('%B'))
print('Week number :',d1.strftime('%W'))
print("Locale's date and time representation :",d1.strftime('%c'))
```

datetime18.py hosted with ❤ by **GitHub** **view raw**

```
Weekday : Thursday
Month : April
Week number : 16
Locale's date and time representation : Thu Apr 23 15:28:30 2020
```

Timedelta

So far, we have seen how to create a DateTime object and how to format it. But sometimes, you might have to find the duration between two dates, which can be another very useful feature that you can derive from a dataset. This duration is, however, returned as a timedelta object.

```
# timedelta : duration between dates
d1 = datetime(2020,4,23,11,13,10)
d2 = datetime(2021,4,23,12,13,10)
duration = d2-d1
print(type(duration))
duration
```

datetime19.py hosted with ❤ by **GitHub** **view raw**

```
<class 'datetime.timedelta'>
datetime.timedelta(days=365, seconds=3600)
```

As you can see, the duration is returned as the number of days for the date and seconds for the time between the dates. So you can actually retrieve these values for your features:

```
print(duration.days) # 365
print(duration.seconds) # 3600
```

datetime20.py hosted with ❤ by **GitHub** **view raw**

But what if you actually wanted the duration in hours or minutes? Well, there is a simple solution for that.

timedelta is also a class in the DateTime module. So, you could use it to convert your duration into hours and minutes as I've done below:

```
from datetime import timedelta
# duration in hours
print('Duration in hours :',duration/timedelta(hours=1))
# duration in minutes
print('Duration in minutes :',duration/timedelta(minutes=1))
# duration in seconds
print('Duration in seconds :',duration/timedelta(seconds=1))
```

All you Should Know About Datetime Variables in Python and Pandas

```
Duration in seconds : 31539600.0
```

Now, what if you wanted to get the date 5 days from today? Do you simply add 5 to the present date?

```
1  d1 = datetime.now()
2  d1+5
```

datetime22.py hosted with ❤ by **GitHub** **view raw**

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-177-6399cec8b005> in <module>
      1 d1 = datetime.now()
----> 2 d1+5

TypeError: unsupported operand type(s) for +: 'datetime.datetime' and 'int'
```

Not quite. So how do you go about it then? You use timedelta of course!

timedelta makes it possible to add and subtract integers from a DateTime object.

```
1  d1 = datetime.now()
2  print("Today's date :",d1)
3
4  d2 = d1+timedelta(days=2)
5  print("Date 2 days from today :",d2)
6
7  d3 = d1+timedelta(weeks=2)
8  print("Date 2 weeks from today :",d3)
```

datetime23.py hosted with ❤ by **GitHub** **view raw**

```
Today's date : 2020-04-23 16:12:49.863505
Date 2 days from today : 2020-04-25 16:12:49.863505
Date 2 weeks from today : 2020-05-07 16:12:49.863505
```

DateTime in Pandas

We already know that Pandas is a great library for doing data analysis tasks. And so it goes without saying that Pandas also supports Python DateTime objects. It has some great methods for handling dates and times, such as **to_datetime()** and **to_timedelta()**.

DateTime and Timedelta objects in Pandas

The **to_datetime()** method converts the date and time in string format to a DateTime object:

```
1  # to_datetime
2  date = pd.to_datetime('24th of April, 2020')
3  print(date)
4  print(type(date))
```

datetime24.py hosted with ❤ by **GitHub** **view raw**

```
2020-04-24 00:00:00
<class 'pandas._libs.tslibs.timestamps.Timestamp'>
```

You might have noticed something strange here. The type of the object returned by to_datetime() is not DateTime but Timestamp. Well, don't worry, it is just the Pandas equivalent of Python's DateTime.

All you Should Know About Datetime Variables in Python and Pandas

```
date = datetime.now()
# present date
print(date)
# date after 1 day
print(date+pd.to_timedelta(1,unit='D'))
# date after 1 month
print(date+pd.to_timedelta(1,unit='M'))
```

datetime25.py hosted with ❤ by **GitHub** **view raw**

```
2020-04-24 17:58:22.416051
2020-04-25 17:58:22.416051
2020-05-25 04:27:28.416051
```

Here, the unit determines the unit of the argument, whether that's day, month, year, hours, etc.

Date Range in Pandas

To make the creation of date sequences a convenient task, Pandas provides the **date_range()** method. It accepts a start date, an end date, and an optional frequency code:

```
pd.date_range(start='24/4/2020', end='24/5/2020', freq='D')
```

datetime26.py hosted with ❤ by **GitHub** **view raw**

```
DatetimeIndex(['2020-04-24', '2020-04-25', '2020-04-26', '2020-04-27',
               '2020-04-28', '2020-04-29', '2020-04-30', '2020-05-01',
               '2020-05-02', '2020-05-03', '2020-05-04', '2020-05-05',
               '2020-05-06', '2020-05-07', '2020-05-08', '2020-05-09',
               '2020-05-10', '2020-05-11', '2020-05-12', '2020-05-13',
               '2020-05-14', '2020-05-15', '2020-05-16', '2020-05-17',
               '2020-05-18', '2020-05-19', '2020-05-20', '2020-05-21',
               '2020-05-22', '2020-05-23', '2020-05-24'],
              dtype='datetime64[ns]', freq='D')
```

Instead of defining the end date, you could define the period or number of time periods you want to generate:

```
from datetime import datetime
start_date = datetime.today()
dates_start = pd.date_range(start=start_date, periods=10, freq='T')
dates_start[:5]
```

datetime27.py hosted with ❤ by **GitHub** **view raw**

```
DatetimeIndex(['2020-04-24 18:06:01.312011', '2020-04-24 18:07:01.312011',
               '2020-04-24 18:08:01.312011', '2020-04-24 18:09:01.312011',
               '2020-04-24 18:10:01.312011'],
              dtype='datetime64[ns]', freq='T')
```

Making DateTime Features in Pandas

Let's also create a series of end dates and make a dummy dataset from which we can derive some new features and bring our learning about DateTime to fruition.

```
dates_end = pd.date_range(start=start_date, periods=10, freq='D')
dates_end[:5]
```

datetime28.py hosted with ❤ by **GitHub** **view raw**

```
DatetimeIndex(['2020-04-24 18:06:01.312011', '2020-04-25 18:06:01.312011',
               '2020-04-26 18:06:01.312011', '2020-04-27 18:06:01.312011',
```

All you Should Know About Datetime Variables in Python and Pandas

```
for i in range(10):
    randomList.append(random.randint(0,1))

# dataframe
df = pd.DataFrame()
df['Start_date'] = dates_start
df['End_date'] = dates_end
df['Target'] = randomList

df.head()
```

datetime29.py hosted with ❤ by GitHub view raw

Perfect! So we have a dataset containing start date, end date, and a target variable:

	Start_date	End_date	Target
0	2020-04-24 18:06:01.312011	2020-04-24 18:06:01.312011	1
1	2020-04-24 18:07:01.312011	2020-04-25 18:06:01.312011	0
2	2020-04-24 18:08:01.312011	2020-04-26 18:06:01.312011	0
3	2020-04-24 18:09:01.312011	2020-04-27 18:06:01.312011	0
4	2020-04-24 18:10:01.312011	2020-04-28 18:06:01.312011	0

We can create multiple new features from the date column, like the day, month, year, hour, minute, etc. using the **dt attribute** as shown below:

```
# day
df['Day'] = df['Start_date'].dt.day
# month
df['Month'] = df['Start_date'].dt.month
# year
df['Year'] = df['Start_date'].dt.year
# hour
df['Start_hour'] = df['Start_date'].dt.hour
# minute
df['Start_minute'] = df['Start_date'].dt.minute
# second
df['Start_second'] = df['Start_date'].dt.second
# Monday is 0 and Sunday is 6
df['Start_weekday'] = df['Start_date'].dt.weekday
# week of the year
df['Start_week_of_year'] = df['Start_date'].dt.week
# duration
df['Duration'] = df['End_date']-df['Start_date']
```

datetime30.py hosted with ❤ by GitHub view raw

	Start_date	End_date	Target	Day	Month	Year	Start_hour	Start_minute	Start_second	Start_weekday	Start_week_of_year	Duration
0	2020-04-25 16:04:23.419791	2020-04-25 16:04:23.419791	1	25	4	2020	16	4	23	5	17	0 days 00:00:00
1	2020-04-25 16:05:23.419791	2020-04-26 16:04:23.419791	0	25	4	2020	16	5	23	5	17	0 days 23:59:00
2	2020-04-25 16:06:23.419791	2020-04-27 16:04:23.419791	1	25	4	2020	16	6	23	5	17	1 days 23:58:00
3	2020-04-25 16:07:23.419791	2020-04-28 16:04:23.419791	1	25	4	2020	16	7	23	5	17	2 days 23:57:00
4	2020-04-25 16:08:23.419791	2020-04-29 16:04:23.419791	1	25	4	2020	16	8	23	5	17	3 days 23:56:00

Our duration feature is great, but what if we would like to have the duration in minutes or seconds? Remember how in the timedelta section we converted the date to seconds? We could do the same here!

All you Should Know About Datetime Variables in Python and Pandas

datetime31.py hosted with ❤ by GitHub view raw

jet	Day	Month	Year	Start_hour	Start_minute	Start_second	Start_weekday	Start_week_of_year	Duration	Duration_days	Duration_minutes	Duration_seconds
1	24	4	2020	18	6	1	4	17	0 days 00:00:00	0.000000	0.0	0.0
0	24	4	2020	18	7	1	4	17	0 days 23:59:00	0.999306	1439.0	86340.0
0	24	4	2020	18	8	1	4	17	1 days 23:58:00	1.998611	2878.0	172680.0
0	24	4	2020	18	9	1	4	17	2 days 23:57:00	2.997917	4317.0	259020.0
0	24	4	2020	18	10	1	4	17	3 days 23:56:00	3.997222	5756.0	345360.0

Great! Can you see how many new features we created from just the dates?

Now, let's make the start date the index of the DataFrame. This will help us easily analyze our dataset because we can use slicing to find data representing our desired dates:

```
df.index=df['Start_date']
df['2020-04-24':'2020-04-24'].head()
```

datetime32.py hosted with ❤ by GitHub view raw

Start_date	Start_date	End_date	Target	Day	Month	Year	Start_hour	Start_minute	Start_second	Start_weekday	Start_week_of_year	Durat
2020-04-24 18:06:01.312011	2020-04-24 18:06:01.312011	2020-04-24 18:06:01.312011	1	24	4	2020	18	6	1	4	17	0 d 00:00
2020-04-24 18:07:01.312011	2020-04-24 18:07:01.312011	2020-04-25 18:06:01.312011	0	24	4	2020	18	7	1	4	17	0 d 23:59
2020-04-24 18:08:01.312011	2020-04-24 18:08:01.312011	2020-04-26 18:06:01.312011	0	24	4	2020	18	8	1	4	17	1 d 23:58
2020-04-24 18:09:01.312011	2020-04-24 18:09:01.312011	2020-04-27 18:06:01.312011	0	24	4	2020	18	9	1	4	17	2 d 23:57
2020-04-24 18:10:01.312011	2020-04-24 18:10:01.312011	2020-04-28 18:06:01.312011	0	24	4	2020	18	10	1	4	17	3 d 23:56

Awesome! This is super useful when you want to do visualizations or any data analysis.

End Notes

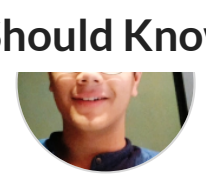
I hope you found this article on how to manipulate date and time features with Python and Pandas useful. But nothing is complete without practice. Working with time series datasets is a wonderful way to practice what we have learned in this article.

I recommend taking part in a time series hackathon on the DataHack platform. You might want to go through this and this article first in order to gear up for that hackathon.

date features datetime feature engineering pandas time features timedelta

About the Author

All you Should Know About Datetime Variables in Python and Pandas

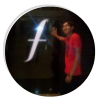


that I face, with the community to make the world a better place!

Our Top Authors






view more

Download

Analytics Vidhya App for the Latest blog/Article

Previous Post

Key Takeaways from ICLR 2020 (with a Case Study on PyTorch vs. TensorFlow)

Next Post

7 Impressive Scikit-learn Hacks, Tips and Tricks for Data Science

One thought on "All you Should Know About Datetime Variables in Python and Pandas"

Ajay Rathi says:
October 18, 2020 at 8:02 pm

i have learned more. Thanks

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name*

Email*

Website

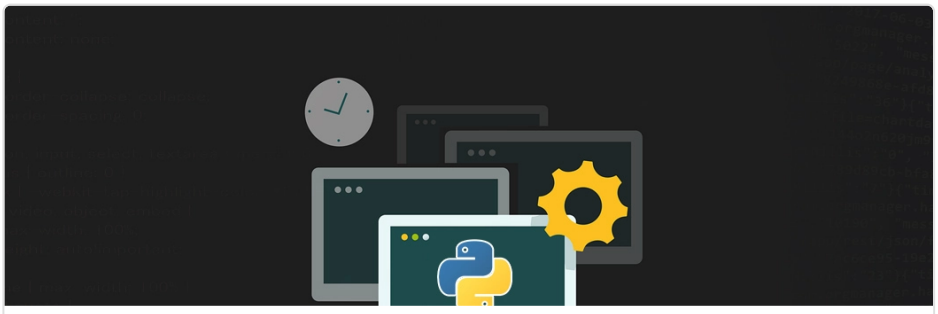
Notify me of follow-up comments by email.

Notify me of new posts by email.

Submit

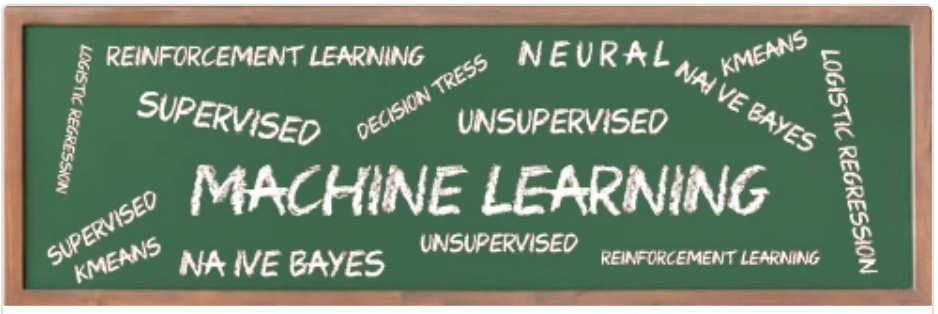
All you Should Know About Datetime Variables in Python and Pandas

Top Resources



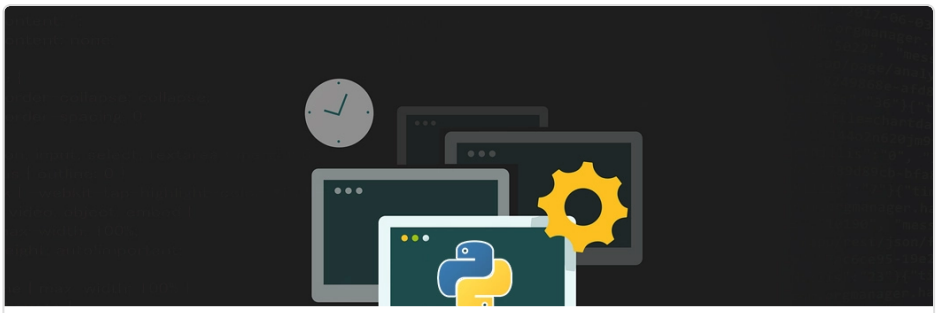
Python Tutorial: Working with CSV file for Data Science

Harika Bonthu - AUG 21, 2021



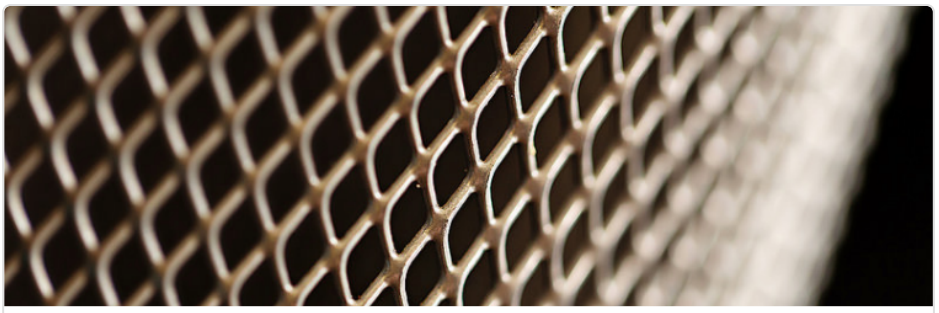
Commonly used Machine Learning Algorithms (with Python and R Codes)

Sunil Ray - SEP 09, 2017



Basic Concepts of Object-Oriented Programming in Python

Himanshi Singh - SEP 01, 2020



40 Questions to test a Data Scientist on Clustering Techniques..

Saurav Kaushik - FEB 05, 2017