Imputing Missing Dates(not Data) in Python

BEGINNER DATA ANALYSIS DATA EXPLORATION PYTHON TECHNIQUE TIME SERIES

Introduction

Imputing missing values is a crucial step when dealing with data. It is one of the steps performed in the Data Analysis. And coming to time-series data, the missing dates play a major role in the overall analysis or when we are trying to visualize the time-series data. If the missing dates are untouched, the performance of many time-series Machine Learning models will be affected. So one must carefully handle the missing dates and ensure the data is rearranged with the appropriate date imputations. This article explains how to impute missing dates in our python project.

Imputing missing dates depends on the type of data we get. The time-series data can be monthly, weekly, or even daily data. In this article, we will walk through all these scenarios where we will identify the missing dates in the data and write respective codes to impute these missing dates. We will be doing this using the Python Pandas library.

Learning Objectives

1. To identify missing dates
2. Learning how to impute dates
3. Dealing with missing dates in daily, weekly, or monthly data

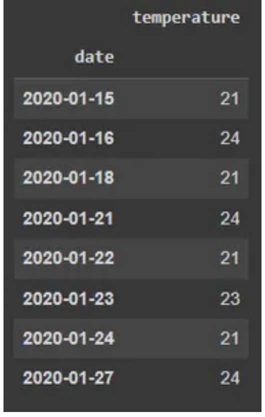
This article was published as a part of the Data Science Blogathon.

Table of Contents

Imputing Missing Dates in Daily Data

This section will discuss how to identify and impute data containing daily dates. The dataset I'm using contains daily temperature data. The first step in this will be to convert the date column to datetime datatype. We have the to_datetime() method from the Pandas library for this. Then we will make the date column the index. So let's change the data type of the date column and view the dataset.

```
import pandas as pd df_daily = pd.read_excel("/content/daily date.xlsx") df_daily.index= pd.to_datetime(df_daily.index)
```



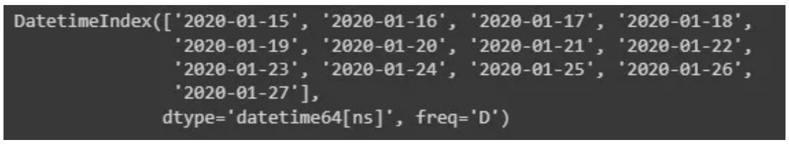
```
            temperature
      date
2020-01-15           21
2020-01-16           24
2020-01-18           21
2020-01-21           24
2020-01-22           21
2020-01-23           23
2020-01-24           21
2020-01-27           24
```

In the above image, we can see the dataset we have taken. The dates here range from 15th Jan 2020 to 27th Jan 2020. You can view that the data contains missing dates like the 14th, 19th, 20th, etc. dates.

The next step is to store the dataset's start and end date. With this, we can generate daily dates from the start date to the end date using the **date_range()** function in pandas. Let's see its implementation below.

```
start = df_daily.index[0].date() end = df_daily.index[len(df_daily)-1].date() new_dates = pd.date_range(start=start,end=end,freq='D')
```

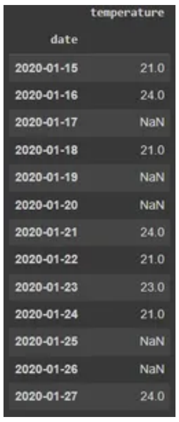
The **start** variable contains the date 2020-01-15, and the **end** variable contains the date 2020-01-27. These start and end dates are passed like parameters to the **date_range()** method. As the data we deal with is daily, we specify the frequency as **'D.'** The resulting dates are stored in the variable **new_dates**. The output shown is the contents of the variable **new_dates**.



```
DatetimeIndex(['2020-01-15', '2020-01-16', '2020-01-17', '2020-01-18',
               '2020-01-19', '2020-01-20', '2020-01-21', '2020-01-22',
               '2020-01-23', '2020-01-24', '2020-01-25', '2020-01-26',
               '2020-01-27'],
              dtype='datetime64[ns]', freq='D')
```

Finally, we have generated the new dates. The next step will be reindexing these new dates with the old dates in the daily temperature data. To achieve this, we use the **reindex()** method. To do this

```
df_daily = df_daily.reindex(new_dates) df_daily = df_daily.rename_axis('date') print(df_daily)
```



```
            temperature
      date
2020-01-15         21.0
2020-01-16         24.0
2020-01-17          NaN
2020-01-18         21.0
2020-01-19          NaN
2020-01-20          NaN
2020-01-21         24.0
2020-01-22         21.0
2020-01-23         23.0
2020-01-24         21.0
2020-01-25          NaN
2020-01-26          NaN
2020-01-27         24.0
```

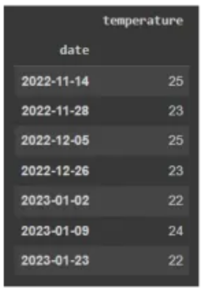
Finally, the dates have been imputed. From the above pic, it is visible that the missing dates have been added. When the missing dates are added, the corresponding values for other columns will be NaN. These can be imputed in different ways. We are not going to talk about that here. We will be only talking about the missing dates.

Imputing Missing Dates in Weekly Data

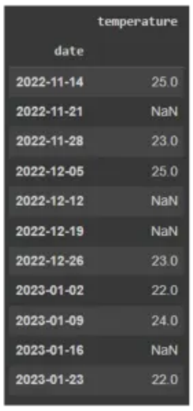
This section will deal with the weekly data containing missing dates. It is very much similar to the missing daily dates problem. First, we convert the date to datetime type using the pandas **to_datetime()**. Then we store the first and last dates from our data in the **start** and **end** variables. Then we use the **data_range()** to get the new dates and reindex them to our data frame.

```
df_weekly    =    pd.read_excel("/content/weekly.xlsx")    df_weekly.index=    pd.to_datetime(df_weekly.index,
infer_datetime_format=True) start = df_weekly.index[0].date() end = df_weekly.index[len(df_weekly)-1].date()
new_dates = pd.date_range(start=start,end=end,freq='7D') df_weekly = df_weekly.reindex(new_dates) df_weekly =
df_weekly.rename_axis('date')
```

In the above code, for the **to_datetime()** function, we have used **infer_datetime_format=True**. This function determines which part in 2020-11-14 belongs to the Month, which part belongs to the Day, and which Year. From the output images, we can see that the missing dates are imputed, and at the same time, **NaN** values are placed in the missing columns.



	temperature
date	
2022-11-14	25
2022-11-28	23
2022-12-05	25
2022-12-26	23
2023-01-02	22
2023-01-09	24
2023-01-23	22



	temperature
date	
2022-11-14	25.0
2022-11-21	NaN
2022-11-28	23.0
2022-12-05	25.0
2022-12-12	NaN
2022-12-19	NaN
2022-12-26	23.0
2023-01-02	22.0
2023-01-09	24.0
2023-01-16	NaN
2023-01-23	22.0

One thing to note here is that, in the **date_range()** function, we have used **'7D'** as frequency instead of **'w.' 'w'** is the frequency you can use to generate weekly dates. **'7D'** also is the same; it generates the dates with having 7 days gap between two dates.

But the main difference is that when we use **'w**,' the dates are generated in a different format. For example, in this data, our first day is 14th Nov 2022, which is Monday, and the next day generated must be 21st Nov. But using **'w'**, the first day will be created is 20th Nov. Because **'w'** generates dates such that they must start from the start of the week that is Sunday. As the 20th Nov is Sunday, the **'w'** will start generating days from Sunday, so the dates generated will be 20th Nov, 27th Nov, etc. So it is better to go with **'7D'** instead of **'w,'** which will generate the dates correctly.

Imputing Missing Dates in Monthly Data

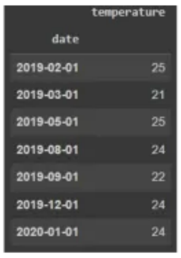
In this section, we will finally work with monthly data containing missing dates. By now, you must have found a way to impute missing dates for monthly data. If not, do not worry.

As usual, first, we convert the date to datetime type using the pandas **to_datetime()**. Then we store the first and last dates from our data in the **start** and **end** variables. Then we use the **data_range()** to get the new dates and then reindex them to our DataFrame.

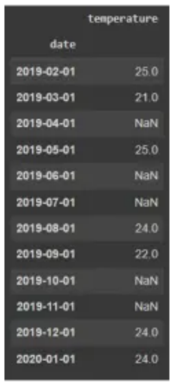
```
df_monthly    =    pd.read_excel("/content/monthly.xlsx")    df_monthly.index=    pd.to_datetime(df_monthly.index,
format="%d-%m-%Y")   start   =   df_monthly.index[0].date()   end   =   df_monthly.index[len(df_monthly)-1].date()
```

```
new_dates    =    pd.date_range(start=start,end=end,freq='MS')    df_monthly    =    df_monthly.reindex(new_dates)
df_monthly = df_monthly.rename_axis('date')
```

Here instead of **infer_date_format**, we have used **format** in **to_datetime()**. It functions a bit differently. We pass the format of our date to the **format** variable. The frequency used here is **'MS,'** which refers to the month's start. The dates are generated with the starting day for all months. If you want the day to be the end date, then you can use just **'M'** as freq. Below is the output; we can see that the missing monthly dates were indeed added to the data.



	temperature
date	
2019-02-01	25
2019-03-01	21
2019-05-01	25
2019-08-01	24
2019-09-01	22
2019-12-01	24
2020-01-01	24



	temperature
date	
2019-02-01	25.0
2019-03-01	21.0
2019-04-01	NaN
2019-05-01	25.0
2019-06-01	NaN
2019-07-01	NaN
2019-08-01	24.0
2019-09-01	22.0
2019-10-01	NaN
2019-11-01	NaN
2019-12-01	24.0
2020-01-01	24.0

Conclusion

While analyzing time-series data, it's crucial to fill in missing dates. Many methods are available in Python to find the missing dates, and we have used the Pandas package to solve this. The use of Pandas and its functions to fill in missing dates in Python was covered in this article. We learned this by applying these functions to weekly, daily, and monthly missing data. This article's ideas will show you how to efficiently manage missing dates in your data.

The key takeaways from this article include the following:

1. Using the **to_datetime()** method to convert the date column type to pandas datetime type and then further used to impute missing dates.
2. **date_range()** method can be used to generate dates for a given start and end dates, and the frequency of generated dates can be modified
3. **reindex()** method is used to reindex the old date column with the new date column, which has missing dates.

The media shown in this article is not owned by Analytics Vidhya and is used at the Author's discretion.

Article Url - https://www.analyticsvidhya.com/blog/2023/02/impute-missing-dates-not-data-in-python/

Ajay Kumar Reddy