Pandas Datetime Cheat Sheet

Pandas is a powerful library for working with datetime data in Python. Pandas offer variaty of attributes, methods, classes to work with date and time.

• import

Datetime related libraries.

from datetime import datetime timedelta	manipulating dates and time
import pandas as pd	Working with datetime Series
import time	various time-related functions

• now & today

Get current date and time in Pandas.

Pd.to_datetime('today')	get current date and time in local timezone
Pd.to_datetime ('now')	current timestamp in UTC
pd.to_datetime ('today').norma-lize()	today's date midnight
datetime.today().strftime('%d/%m/%y')	get current date in a given format
time.strftime ('%d/%m/%y')	use module time for current date & time
datetime.datetime.now().isoformat()	get local timestamp ISO format
datetime.datetime.utcnow().isoformat()	get UTC time in ISO format

parse date & time

Parse strings to datetime.

pd.to_datetime('2023-01-11 16:11:26.862697')	Parse string to datetime
pd.to_datetime(df['date'])	Parse column to datetime
pd.to_datetime(df['date'], dayfirst = True)	Specify parse order - True 10/11/12 is parsed as 2012-11-10
pd.to_datetime(df['date'], yearfirst = True)	True - 10/11/12 is parsed as 2010-11-12
pd.to_datetime(df['date'], infer_datetime_format = True)	infer the date format based on the first non-NaN item
pd.to_datetime(df['date'], format = '%y-%m-%d %H:%M:%S')	specify parsing format %y-%m-%d %H:%M:%S
pd.to_datetime(df[['year','month','day']])	parse multiple columns

attributes

Datetime components and attributes.

dt.year	get year from datetime
dt.month	get month from datetime
dt.day	get day from datetime
dt.hour	get hour from datetime

Attributes

dt.minute	get minute from datetime
dt.second	get second from datetime
dt.day_of_year	get day of the year from datetime
dt.dayofweek	return the day of the week
dt.is_leap_year	boolean indicator if the date belongs to a leap year
dt.daysinmonth	the number of days in the month

methods

Popular datetime methods.

dto = pd.to_datetime('2023-01-11 16:26.862697') dto.day_name()	return the day names with specified locale
dto.month_name()	return the month names with specified locale
dto.tz_localize('UTC')	localize tz-naive Datetime to tz-aware Datetime
dto.tz_convert('US/central')	Convert tz-aware Datetime Array/ Index from one time zone to another
idx= pd.date_range('2023-01-01', periods=3) idx.to_period()	Converts Datetime to period
dto.round('H')	round operation on the data to the specified freq

• methods

dto.floor ('1min')	rounds down to the nearest value
dto.ceil ('1min')	rounds up to the nearest value

• calculation

Datetime calculation- extraction and addition of dates and time.

t1= pd.to_datetime('1/1/2023 01:00') t2=pd.to_datetime ('today') (t2-t1).components	get time difference and return components(day, hour, minute, second)
(t2 - t1). seconds	get only seconds component
(t2-t1).total_seconds()	get total seconds between two dates
df['date1'].dt.year-df['date2'].dt.year	Calculate difference of the years.
df['date']+ pd.Timedelta(days=1)	add 1 day to datetime column
df['date']+ pd.DateOffset(hours=16)	add 16 hours to datetime column
pd.Timedelta (5, 'H')	get timedelta - 5 hours
pd.Timedelta(1,'d').total_seconds()	get 1 day delta as seconds

• select date & time

Filter rows by date, year, period or time.

df.set_index(['date']) df.sort_index(inplace=True, ascending=True)	Set date column as index and sort for consistence
df.index = df['date']	Set date as index and keep the column
df.loc['2023']	Select rows by index dates in 2023 year
df.loc['2022-7']	Select rows by index dates in July 2022
df.loc['2022-1-1']	Select rows by index dates by a given day
df.loc['2019':'2022']	Select all rows between two years (inclusive)
df.loc[df['date'] > '2022-01-01']	Select all rows for datetime column
df.between_time('11:10','12:15')	Select between start and end time
df[df['date'].between('2022', '2023')]	Select rows by date column between two dates
df.loc['2022-7-1 11:10' : 2023-1-1 12:15']	locate by timestamps
pd.Interval(t1,t2).length	get interval length between two timestamps
pd.Interval(t1,t2).overlaps (pd.Interval(t3,t4))	check if two periods overlaps

time zone

Timezone aware timestamps and conversion.

pd.Timestamp.now()	naive local time
pd.Timestamp.utcnow()	timezone aware (UTC)
pd.Timestamp.now(tz='Europe/Rome').tz_localize(None)	naive local time
pd.Timestamp.now(tz='Europe/Rome')	timezone aware local time
pd.Timestamp.utcnow().tz_loca-lize(None)	remove the timezone information but converting to UTC
pd.Timestamp.utcnow().tz_con-vert(None)	removes the timezone information resulting in naive local
dto.dt.tz_localize('+0100')	localize using offset

read_csv

read_csv - parsing dates.

pd.read_csv('test.csv', parse_dates=['end_date'])	try parsing columns each as a separate date column
pd.read_csv('test_csv', parse_dates=[['yy', 'mm', 'dd']])	combine columns yy, mm, dd and parse as a single date column
pd.read_csv('test.csv', parse_dat-es={'date1':['yy','mm','dd']})	parse columns yy, mm, dd as date and call result 'date1'

• read_csv

pd.read_csv('test.csv', day-first = True)	DD/MM format dates, international and European format
pd.read_csv('test.csv', keep_date_col = True)	if parse_dates then keep the original columns

• date & time format

Date and time directives for formatting

from datetime import datetime now = datetime.now() now.strftime('%x')	14:56:48 || get current time and format it
%a	Thu || Abbreviated weekday (Sun)
%A	Thursday || Weekday (Sunday)
%b	Jan || Abbreviated month name (Jan)
%B	January || Month name (January)
%C	Thu Jan 5 14:50:48 2023 || Date and time
%d	5 || Day (leading zeros) (01 to 31)
%H	14 || 24 hour (leading zeros) (00 to 23)
%I	2 || 12 hour (leading zeros) (01 to 12)
%j	5 || Day of year (001 to 366)
%m	1 || Month (01 to 12)
%M	53 || Minute (00 to 59)

• date & time format

%p	PM || AM or PM
%S	48 || Second (00 to 29)
%U	1 || Week number (00 to 53)
%w	4 || Weekday (0-Sun to 6-Sat)
%W	1 || Week number (00 to 53)
%x	01/05/23 || Date
%X	14:56:48 || Time
%y	23 || Year without century (00 to 99)
%Y	2023 || year (2008)
%z	GMT || Time zone (GMT)
%%	% || A literal % character