```
import numpy as np
import pandas as pd
```

Timestamp Object

Time stamps reference particular moments in time (e.g., Oct 24th, 2022 at 7:00pm)

Creating Timestamp objects

```
# creating a timestamp
type(pd.Timestamp('2023/1/5'))
```

```
# variations
pd.Timestamp('2023-1-5')
pd.Timestamp('2023, 1, 5')
```

```
# only year
pd.Timestamp('2023')
```

```
# using text
pd.Timestamp('5th January 2023')
```

```
# providing time also
pd.Timestamp('5th January 2023 9:21AM')
# pd.Timestamp('2023/1/5/9/21')
```

```
# AM and PM
```

```
# using datetime.datetime object
import datetime as dt

x = pd.Timestamp(dt.datetime(2023,1,5,9,21,56))
x
```

```
# fetching attributes
x.year
x.month
x.day
x.hour
x.minute
x.second
```

```
# why separate objects to handle data and time when python already has datetime functionality?
```

- syntax wise datetime is very convenient
- But the performance takes a hit while working with huge data. List vs Numpy Array
- The weaknesses of Python's datetime format inspired the NumPy team to add a set of native time series data type to NumPy.
- The datetime64 dtype encodes dates as 64-bit integers, and thus allows arrays of dates to be represented very compactly.

```
import numpy as np
date = np.array('2015-07-04', dtype=np.datetime64)
date
```

```
date + np.arange(12)
```

- Because of the uniform type in NumPy datetime64 arrays, this type of operation can be accomplished much more quickly than if we were working directly with Python's datetime objects, especially as arrays get large
- Pandas Timestamp object combines the ease-of-use of python datetime with the efficient storage and vectorized interface of numpy.datetime64
- From a group of these Timestamp objects, Pandas can construct a DatetimeIndex that can be used to index data in a Series or DataFrame

DatetimeIndex Object

A collection of pandas timestamp

```
# from strings
type(pd.DatetimeIndex(['2023/1/1','2022/1/1','2021/1/1']))
```

```
# using python datetime object
pd.DatetimeIndex([dt.datetime(2023,1,1),dt.datetime(2022,1,1),dt.datetime(2021,1,1)])
```

```
# using pd.timestamps
dt_index = pd.DatetimeIndex([pd.Timestamp(2023,1,1),pd.Timestamp(2022,1,1),pd.Timestamp(2021,1,1)])
```

```
# using datatimeindex as series index

pd.Series([1,2,3],index=dt_index)
```

date_range function

```
# generate daily dates in a given range
pd.date_range(start='2023/1/5',end='2023/2/28',freq='3D')
```

```
# alternate days in a given range
pd.date_range(start='2023/1/5',end='2023/2/28',freq='3D')
```

```
# B -> business days
pd.date_range(start='2023/1/5',end='2023/2/28',freq='B')
```

```
# W -> one week per day
pd.date_range(start='2023/1/5',end='2023/2/28',freq='W-THU')
```

```
# H -> Hourly data(factor)
pd.date_range(start='2023/1/5',end='2023/2/28',freq='6H')
```

```
# M -> Month end
pd.date_range(start='2023/1/5',end='2023/2/28',freq='M')
```

```
# MS -> Month start
pd.date_range(start='2023/1/5',end='2023/2/28',freq='MS')
```

```
# A -> Year end
pd.date_range(start='2023/1/5',end='2030/2/28',freq='A')
```

```
# using periods(number of results)
pd.date_range(start='2023/1/5',periods=25,freq='M')
```

to_datetime function

converts an existing objects to pandas timestamp/datetimeindex object

```
# simple series example

s = pd.Series(['2023/1/1','2022/1/1','2021/1/1'])
pd.to_datetime(s).dt.day_name()
```

```
# with errors
s = pd.Series(['2023/1/1','2022/1/1','2021/130/1'])
pd.to_datetime(s,errors='coerce').dt.month_name()
```

```
df = pd.read_csv('/content/expense_data.csv')
df.shape
```

```
df.head()
```

```
df['Date'] = pd.to_datetime(df['Date'])
```

```
df.info()
```

dt accessor

Accessor object for datetimelike properties of the Series values.

```
df['Date'].dt.is_quarter_start
```

```
# plot graph
import matplotlib.pyplot as plt
plt.plot(df['Date'],df['INR'])
```

```
# day name wise bar chart/month wise bar chart

df['day_name'] = df['Date'].dt.day_name()
```

```
df.head()
```

```
df.groupby('day_name')['INR'].mean().plot(kind='bar')
```

```
df['month_name'] = df['Date'].dt.month_name()
```

```
df.head()
```

```
df.groupby('month_name')['INR'].sum().plot(kind='bar')
```

```
df[df['Date'].dt.is_month_end]
```