02.Procesamiento_series_tiempo

October 29, 2022

0.1 Datos

Los datos que vamos a trabajar fueron descargados desde el NOAA para la estación del Pisco con **lat:-13.752725, lon: -76.201172 y Ele:11.9 m**.

0.2 Importar librerias

[ ]:
```
import pandas as pd
```

0.3 Leer archivo

[ ]:
```
data = pd.read_csv('../../data/3117033.csv')
data
```

[ ]:

	STATION	NAME	DATE	PRCP	TAVG	TMAX	TMIN
0	PEM00084691	PISCO INTERNATIONAL, PE	1963-03-01	NaN	23.3	27.2	NaN
1	PEM00084691	PISCO INTERNATIONAL, PE	1963-03-02	NaN	22.5	27.8	17.2
2	PEM00084691	PISCO INTERNATIONAL, PE	1963-03-03	NaN	21.3	26.1	16.1
3	PEM00084691	PISCO INTERNATIONAL, PE	1963-03-04	NaN	21.1	26.1	17.2
4	PEM00084691	PISCO INTERNATIONAL, PE	1963-03-05	NaN	22.7	27.2	17.2
…	…	…	…	…	…	…	…
20186	PEM00084691	PISCO INTERNATIONAL, PE	2022-10-11	NaN	16.8	NaN	13.8
20187	PEM00084691	PISCO INTERNATIONAL, PE	2022-10-12	NaN	16.1	21.5	11.6
20188	PEM00084691	PISCO INTERNATIONAL, PE	2022-10-13	NaN	15.3	NaN	12.3
20189	PEM00084691	PISCO INTERNATIONAL, PE	2022-10-14	NaN	16.9	NaN	15.0
20190	PEM00084691	PISCO INTERNATIONAL, PE	2022-10-15	NaN	15.7	NaN	…

```
20190  11.5

[20191 rows x 7 columns]
```

Podemos ver el formato de cada columna.

[ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20191 entries, 0 to 20190
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   STATION  20191 non-null  object
 1   NAME     20191 non-null  object
 2   DATE     20191 non-null  object
 3   PRCP     9306 non-null   float64
 4   TAVG     20191 non-null  float64
 5   TMAX     12710 non-null  float64
 6   TMIN     14201 non-null  float64
dtypes: float64(4), object(3)
memory usage: 1.1+ MB
```

0.4 Procesar archivo

Vamos a convertir la columna **DATE** en formato fecha.

[ ]:
```
data['DATE'] = pd.to_datetime(data['DATE'])
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20191 entries, 0 to 20190
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   STATION  20191 non-null  object
 1   NAME     20191 non-null  object
 2   DATE     20191 non-null  datetime64[ns]
 3   PRCP     9306 non-null   float64
 4   TAVG     20191 non-null  float64
 5   TMAX     12710 non-null  float64
 6   TMIN     14201 non-null  float64
dtypes: datetime64[ns](1), float64(4), object(2)
memory usage: 1.1+ MB
```

Ahora vamos a declarar el **index** con la columna **DATE**

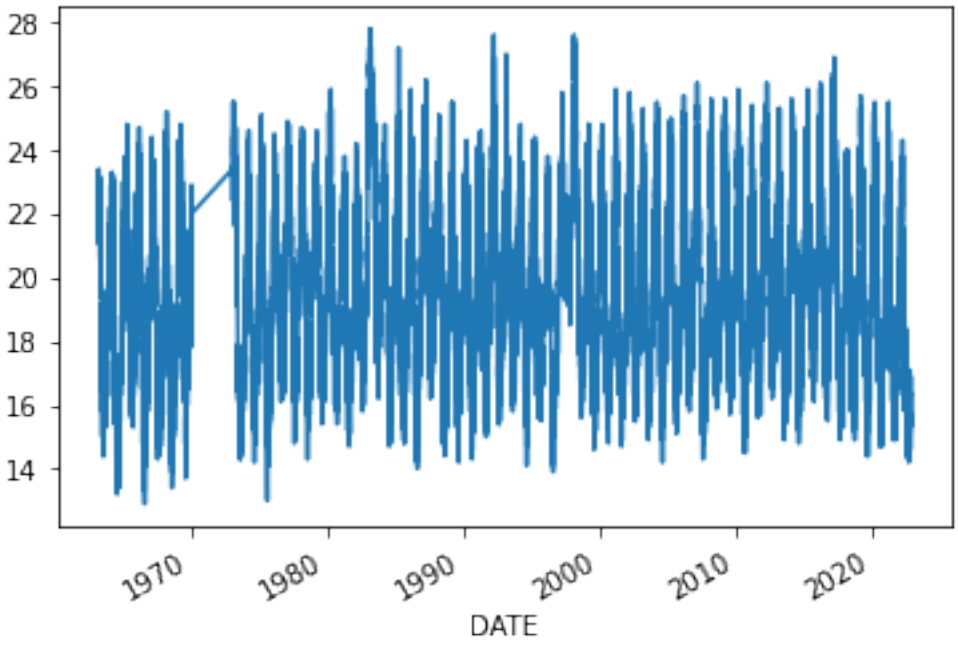
[ ]:
```
data.index = data['DATE']
data
```

[ ]:
```
                STATION                     NAME       DATE  PRCP  TAVG  TMAX  \
DATE
1963-03-01  PEM00084691  PISCO INTERNATIONAL, PE 1963-03-01   NaN  23.3  27.2
1963-03-02  PEM00084691  PISCO INTERNATIONAL, PE 1963-03-02   NaN  22.5  27.8
1963-03-03  PEM00084691  PISCO INTERNATIONAL, PE 1963-03-03   NaN  21.3  26.1
1963-03-04  PEM00084691  PISCO INTERNATIONAL, PE 1963-03-04   NaN  21.1  26.1
1963-03-05  PEM00084691  PISCO INTERNATIONAL, PE 1963-03-05   NaN  22.7  27.2
…                  …                       …        …    …    …    …
2022-10-11  PEM00084691  PISCO INTERNATIONAL, PE 2022-10-11   NaN  16.8   NaN
2022-10-12  PEM00084691  PISCO INTERNATIONAL, PE 2022-10-12   NaN  16.1  21.5
2022-10-13  PEM00084691  PISCO INTERNATIONAL, PE 2022-10-13   NaN  15.3   NaN
2022-10-14  PEM00084691  PISCO INTERNATIONAL, PE 2022-10-14   NaN  16.9   NaN
2022-10-15  PEM00084691  PISCO INTERNATIONAL, PE 2022-10-15   NaN  15.7   NaN

             TMIN
DATE
1963-03-01   NaN
1963-03-02  17.2
1963-03-03  16.1
1963-03-04  17.2
1963-03-05  17.2
…            …
2022-10-11  13.8
2022-10-12  11.6
2022-10-13  12.3
2022-10-14  15.0
2022-10-15  11.5

[20191 rows x 7 columns]
```

Ahora podemos hacer una figura para visualizar nuestros datos.

[ ]:
```
data['TAVG'].plot()
```

[ ]:
```
<AxesSubplot:xlabel='DATE'>
```



0.4.1 Selección en un determinado periodo

Aveces queremos seleccionar un determinado periodo de tiempo. Entonces podemos usar `loc` para hacer esto.

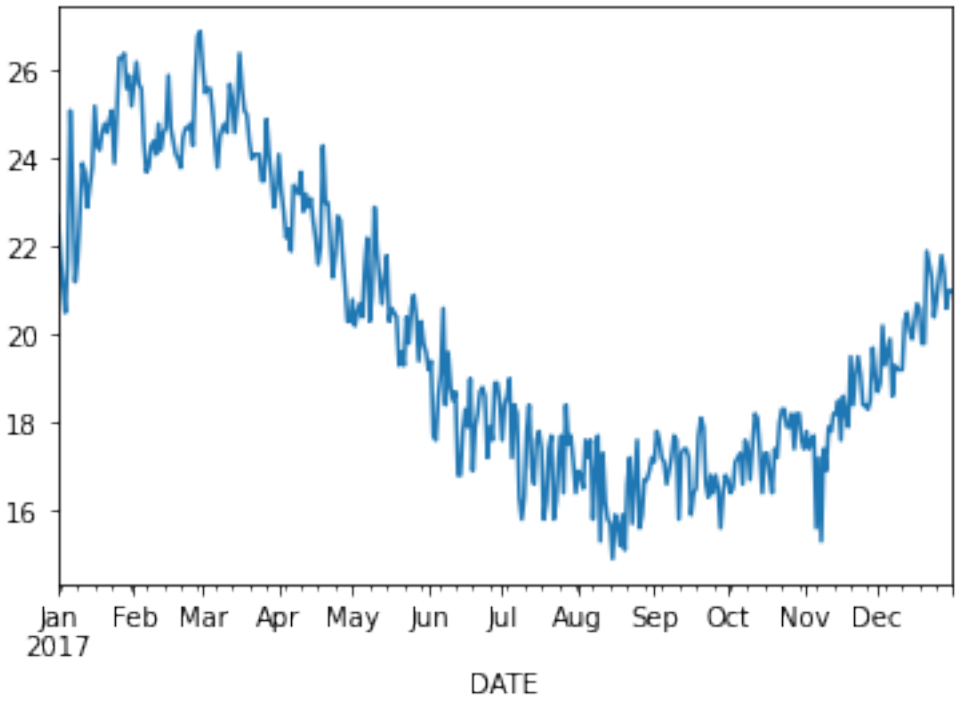
[ ]:
```
data2017 = data.loc['2017']
data2017
```

[ ]:
```
                STATION                     NAME       DATE  PRCP  TAVG  TMAX  \
DATE
2017-01-01  PEM00084691  PISCO INTERNATIONAL, PE 2017-01-01   NaN  22.7  26.3
2017-01-02  PEM00084691  PISCO INTERNATIONAL, PE 2017-01-02   NaN  21.8   NaN
2017-01-03  PEM00084691  PISCO INTERNATIONAL, PE 2017-01-03   NaN  21.1  26.2
2017-01-04  PEM00084691  PISCO INTERNATIONAL, PE 2017-01-04   NaN  20.5  26.4
2017-01-05  PEM00084691  PISCO INTERNATIONAL, PE 2017-01-05   NaN  21.7   NaN
…                  …                     …        …   …   …   …
2017-12-27  PEM00084691  PISCO INTERNATIONAL, PE 2017-12-27   NaN  21.8   NaN
2017-12-28  PEM00084691  PISCO INTERNATIONAL, PE 2017-12-28   NaN  21.4  25.8
2017-12-29  PEM00084691  PISCO INTERNATIONAL, PE 2017-12-29   NaN  20.6  26.1
2017-12-30  PEM00084691  PISCO INTERNATIONAL, PE 2017-12-30   NaN  21.0  23.3
2017-12-31  PEM00084691  PISCO INTERNATIONAL, PE 2017-12-31   NaN  21.0   NaN

            TMIN
DATE
2017-01-01  19.6
2017-01-02   NaN
```

```
2017-01-03  16.7
2017-01-04   NaN
2017-01-05  19.8
…            …
2017-12-27   NaN
2017-12-28  17.4
2017-12-29  17.2
2017-12-30   NaN
2017-12-31  17.5

[365 rows x 7 columns]
```

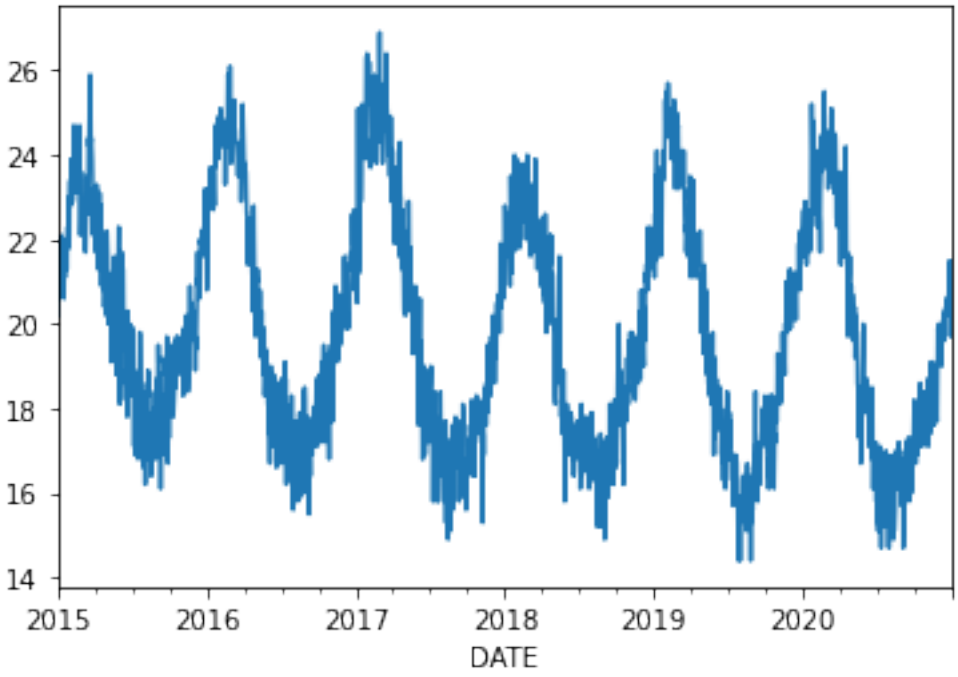
[ ]:
```
data2017['TAVG'].plot()
```

[ ]:
```
<AxesSubplot:xlabel='DATE'>
```



[ ]:
```
data['TAVG'].loc['2015':'2020'].plot()
```

[ ]:
```
<AxesSubplot:xlabel='DATE'>
```



0.4.2 Promedios

Podemos promediar los datos desde diarias para mensuales. Usamos **`resample`** para esto.

[ ]:
```
data_month = data.resample('1M').mean()
data_month
```

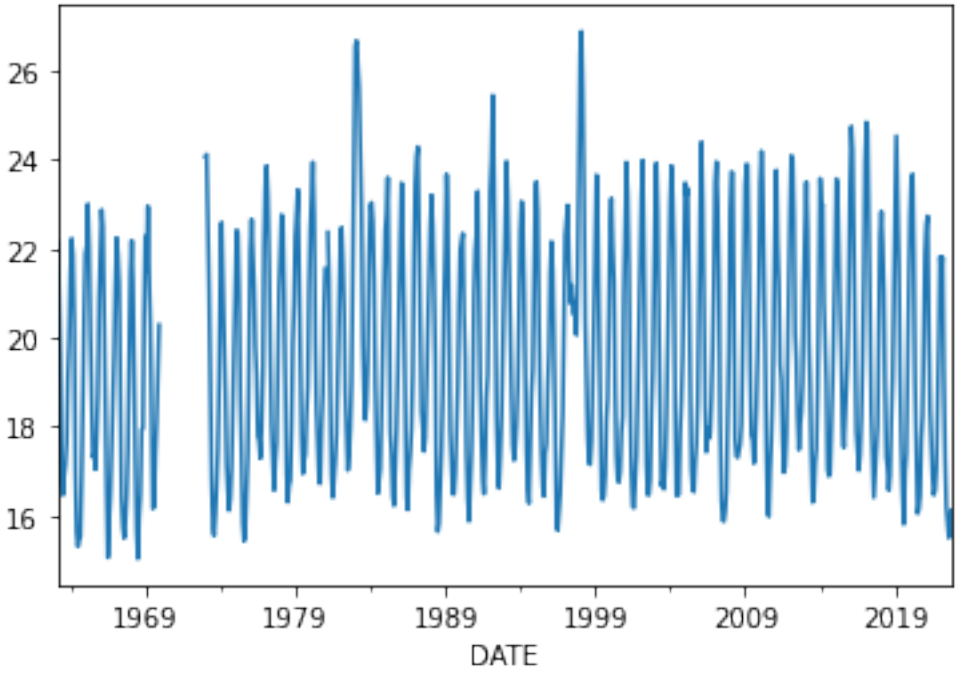
[ ]:

DATE	PRCP	TAVG	TMAX	TMIN
1963-03-31	NaN	22.150000	27.108000	17.151852
1963-04-30	NaN	20.832000	25.991304	16.333333
1963-05-31	NaN	19.395833	24.495238	15.052174
1963-06-30	NaN	16.800000	21.764286	12.508000
1963-07-31	NaN	16.461290	21.042308	12.086667
…	…	…	…	…
2022-06-30	0.0	16.206667	20.022222	12.958333
2022-07-31	0.0	15.861290	19.060000	12.766667
2022-08-31	0.0	15.500000	18.692308	12.843478
2022-09-30	0.0	15.896667	19.760000	13.190000
2022-10-31	NaN	16.120000	20.050000	12.821429

[716 rows x 4 columns]

[ ]:
```
data_month['TAVG'].plot()
```

[ ]: <AxesSubplot:xlabel='DATE'>



También podemos promediar los datos desde diarias para anuales. Usamos **resample** para esto.

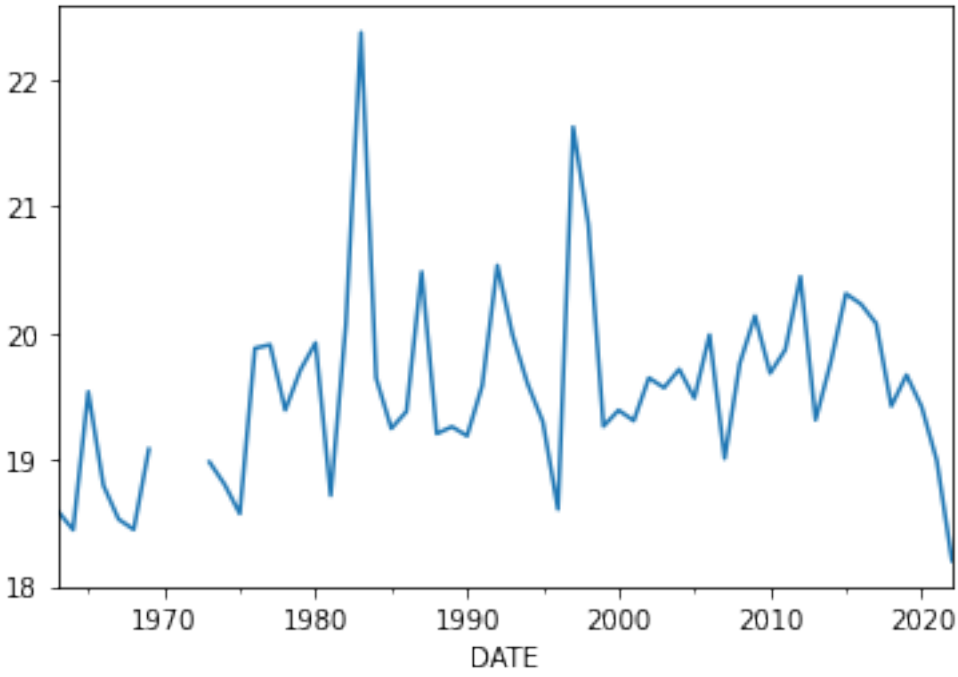
[ ]:
```
data_year = data.resample('1Y').mean()
data_year.head()
```

[ ]:

DATE	PRCP	TAVG	TMAX	TMIN
1963-12-31	NaN	18.593262	23.351751	14.315970
1964-12-31	NaN	18.446429	23.178652	14.630903
1965-12-31	NaN	19.536585	23.927178	15.745946
1966-12-31	NaN	18.797806	23.068269	14.951408
1967-12-31	NaN	18.530164	23.003524	15.002222

[ ]:
```
data_year['TAVG'].plot()
```

[ ]: <AxesSubplot:xlabel='DATE'>



0.4.3 Promedios y sumas más avanzados

Aveces queremo promediar, sumar, sacar desviación estándar, etc. Entonces podemos adicionar la función **agg** para hacer esto.

[ ]:
```
# Importamos la libreria numpy
import numpy as np
```

[ ]:
```
# La funcions `columns` no inprime el nombre de las columnas
data.columns
```

[ ]:
```
Index(['STATION', 'NAME', 'DATE', 'PRCP', 'TAVG', 'TMAX', 'TMIN'],
dtype='object')
```

[ ]:
```
data.resample('1M').agg({'PRCP':np.sum, 'TAVG':np.mean,'TMIN':np.mean, 'TMAX':
 ↪np.mean})
```

[ ]:

DATE	PRCP	TAVG	TMIN	TMAX
1963-03-31	0.0	22.150000	17.151852	27.108000
1963-04-30	0.0	20.832000	16.333333	25.991304
1963-05-31	0.0	19.395833	15.052174	24.495238
1963-06-30	0.0	16.800000	12.508000	21.764286
1963-07-31	0.0	16.461290	12.086667	21.042308
…	…	…	…	…

```
2022-06-30   0.0  16.206667  12.958333  20.022222
2022-07-31   0.0  15.861290  12.766667  19.060000
2022-08-31   0.0  15.500000  12.843478  18.692308
2022-09-30   0.0  15.896667  13.190000  19.760000
2022-10-31   0.0  16.120000  12.821429  20.050000

[716 rows x 4 columns]
```

Si vemos en el **dataframe** que imprimimos , durante los años que no hubo observaciones se escribió como cero. Esto es porque en los pandas automáticamente resetea los valores a cero donde no hay datos nan. Para mejorar este resultado podemos crear una función que solo considere las líneas donde hay observaciones.

[ ]:
```
# Función mejorada para sumar columnas donde contienen nan's
def nansumwrapper(a, **kwargs):
    if np.isnan(a).all():
        return np.nan
    else:
        return np.nansum(a, **kwargs)
```

[ ]:
```
data_month1 = data.resample('1M').agg({'PRCP':nansumwrapper, 'TAVG':np.mean,
'TMIN':np.mean, 'TMAX':np.mean})
data_month1
```

[ ]:
```
             PRCP       TAVG       TMIN       TMAX
DATE
1963-03-31   NaN  22.150000  17.151852  27.108000
1963-04-30   NaN  20.832000  16.333333  25.991304
1963-05-31   NaN  19.395833  15.052174  24.495238
1963-06-30   NaN  16.800000  12.508000  21.764286
1963-07-31   NaN  16.461290  12.086667  21.042308
…            …          …          …          …
2022-06-30   0.0  16.206667  12.958333  20.022222
2022-07-31   0.0  15.861290  12.766667  19.060000
2022-08-31   0.0  15.500000  12.843478  18.692308
2022-09-30   0.0  15.896667  13.190000  19.760000
2022-10-31   NaN  16.120000  12.821429  20.050000

[716 rows x 4 columns]
```

0.4.4 Algunos indicadores estadísticos

Usamos la función **describe** para obtener algunos indicadores estadísticos de nuestros datos.

[ ]:
```
data_month1.describe()
```

[ ]:

	PRCP	TAVG	TMIN	TMAX
count	464.000000	679.000000	671.000000	661.000000
mean	2.462069	19.602924	16.021075	24.462461
std	18.704374	2.591665	2.662457	3.071263
min	0.000000	15.032258	10.539130	18.692308
25%	0.000000	17.370185	13.789474	21.844444
50%	0.000000	19.120000	15.500000	23.982609
75%	0.000000	21.822957	18.170844	27.094444
max	271.800000	26.889286	24.293750	32.284000