01.b.GFS_timeSeries

October 29, 2022

1 Time series

La libreria `siphon` viene con un servicio de consulta especializado para series temporales.

[ ]:
```
from siphon.catalog import TDSCatalog
```

[ ]:
```
# si el catalogo que se desea es de 0.25 grados
best_gfs = TDSCatalog('http://thredds.ucar.edu/thredds/catalog/grib/NCEP/GFS/
 ↪Global_0p25deg/catalog.xml')

# lista de colecciones
print(best_gfs.datasets)

# tupla (),
# lista [],
# diccionario {},
```

```
['Full Collection (Reference / Forecast Time) Dataset', 'Best GFS Quarter Degree
Forecast Time Series', 'Latest Collection for GFS Quarter Degree Forecast']
```

[ ]:
```
best_ds = list(best_gfs.datasets.values())[1]

# diccionario de urls disponibles con la información
best_ds.access_urls
```

[ ]:
```
{'OPENDAP':
'https://thredds.ucar.edu/thredds/dodsC/grib/NCEP/GFS/Global_0p25deg/Best',
 'CdmRemote':
'https://thredds.ucar.edu/thredds/cdmremote/grib/NCEP/GFS/Global_0p25deg/Best',
 'JupyterNotebook':
'https://thredds.ucar.edu/thredds/notebook/grib/NCEP/GFS/Global_0p25deg/Best',
 'NetcdfSubset':
'https://thredds.ucar.edu/thredds/ncss/grid/grib/NCEP/GFS/Global_0p25deg/Best',
 'WMS':
'https://thredds.ucar.edu/thredds/wms/grib/NCEP/GFS/Global_0p25deg/Best',
 'WCS':
'https://thredds.ucar.edu/thredds/wcs/grib/NCEP/GFS/Global_0p25deg/Best',
 'ISO':
```

```
'https://thredds.ucar.edu/thredds/iso/grib/NCEP/GFS/Global_0p25deg/Best',
 'NCML':
'https://thredds.ucar.edu/thredds/ncml/grib/NCEP/GFS/Global_0p25deg/Best',
 'UDDC':
'https://thredds.ucar.edu/thredds/uddc/grib/NCEP/GFS/Global_0p25deg/Best'}
```

[ ]:
```
best_ds = best_gfs.datasets[0]
ncss = best_ds.subset()
query = ncss.query()
```

Requerimiento de datos, para eso recurrimos a la función `query.lonlat_pint()`

[ ]:
```
from datetime import datetime, timedelta, timezone

# definicion de tiempo actual
now = datetime.now(timezone.utc)

# buscando la posición de lima
query.lonlat_point(-77.05 + 360, -12.08).vertical_level(100_000).
 ↪time_range(now, now + timedelta(days=30))
query.accept('netcdf4')
query.variables('Temperature_isobaric', 'Relative_humidity_isobaric')

# haciendo el requerimiento de los datos
data = ncss.get_data(query)
```

[ ]:
```
now + timedelta(hours=3)
```

[ ]:
```
datetime.datetime(2022, 10, 27, 1, 14, 9, 694102, tzinfo=datetime.timezone.utc)
```

[ ]:
```
data.variables.keys()
```

[ ]:
```
dict_keys(['latitude', 'longitude', 'stationAltitude', 'station_id',
'station_description', 'time', 'stationIndex', 'Temperature_isobaric',
'Relative_humidity_isobaric'])
```

[ ]:
```
# extracción de variables especificas
temp = data.variables['Temperature_isobaric']
relh = data.variables['Relative_humidity_isobaric']
tiempo = data.variables['time']
```

[ ]:
```
temp.shape
```

[ ]:
```
(90,)
```

[ ]:
```
import numpy as np
from netCDF4 import num2date
```

```
# tenemos que ayudarnos con el numpy.array para trabajar con estos datos
time_vals = np.array(num2date(tiempo[:].squeeze(), tiempo.units),␣
 ↪dtype='datetime64[s]')
time_vals[:5]
```

[ ]:
```
array(['2022-10-26T21:00:00', '2022-10-27T00:00:00',
       '2022-10-27T03:00:00', '2022-10-27T06:00:00',
       '2022-10-27T09:00:00'], dtype='datetime64[s]')
```

Vamos a dejar nuestros datos mas fáciles de manipular, utilizaremos el `xarray.backends`

recuerda que los datos que vamos a utlizar son los que vienen del `data = ncss.get_data(query)`

[ ]:
```
# utilizemos la función backend del xarray
from xarray.backends import NetCDF4DataStore
import xarray as xr

# creamos una nueva variable para poder trabajar sin muchas sorpresas
data_2 = xr.open_dataset(NetCDF4DataStore(data))

data # data en netcdf4
```

[ ]:
```
<class 'netCDF4._netCDF4.Dataset'>
root group (NETCDF4_CLASSIC data model, file format HDF5):
    Conventions: CF-1.6
    history: Written by CFPointWriter
    title: Extracted data from TDS Feature Collection null
    featureType: timeSeries
    DSG_representation: Timeseries of station data in the indexed ragged array
representation, H.2.5
    time_coverage_start: 2022-10-26T21:00:00Z
    time_coverage_end: 2022-11-11T12:00:00Z
    geospatial_lat_min: -12.0005
    geospatial_lat_max: -11.9995
    geospatial_lon_min: -77.0005
    geospatial_lon_max: -76.9995
    dimensions(sizes): obs(90), station(1), station_id_strlen(37),
station_description_strlen(37)
    variables(dimensions): float64 latitude(station), float64
longitude(station), float64 stationAltitude(station), |S1 station_id(station,
station_id_strlen), |S1 station_description(station,
station_description_strlen), float64 time(obs), int32 stationIndex(obs), float32
Temperature_isobaric(obs), float32 Relative_humidity_isobaric(obs)
    groups:
```

[ ]:
```
# guardando los datos en netcdf
data_2.to_netcdf('data.nc')
```

[ ]:
```
#abriendo datos en netcdf
nc = xr.open_dataset('data.nc')
nc
```

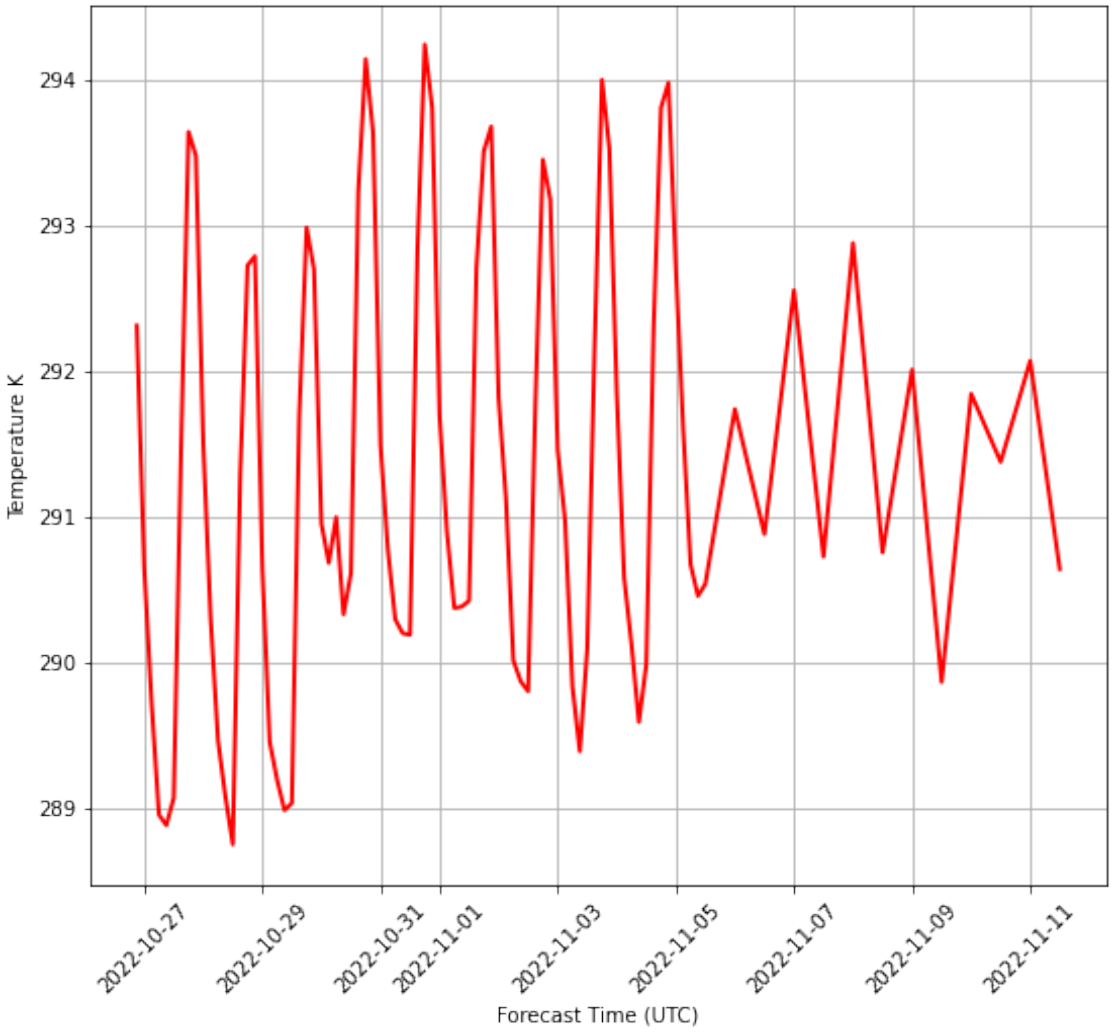
[ ]:
```
<xarray.Dataset>
Dimensions:                      (station: 1, obs: 90)
Coordinates:
    latitude                     (station) float64 …
    longitude                    (station) float64 …
    stationAltitude              (station) float64 …
    time                         (obs) datetime64[ns] …
Dimensions without coordinates: station, obs
Data variables:
    station_id                   (station) |S37 …
    station_description          (station) |S37 …
    stationIndex                 (obs) int32 …
    Temperature_isobaric         (obs) float32 …
    Relative_humidity_isobaric   (obs) float32 …
Attributes:
    Conventions:          CF-1.6
    history:              Written by CFPointWriter
    title:                Extracted data from TDS Feature Collection null
    featureType:          timeSeries
    DSG_representation:   Timeseries of station data in the indexed ragged ar…
    time_coverage_start:  2022-10-26T21:00:00Z
    time_coverage_end:    2022-11-11T12:00:00Z
    geospatial_lat_min:   -12.0005
    geospatial_lat_max:   -11.9995
    geospatial_lon_min:   -77.0005
    geospatial_lon_max:   -76.9995
```

1.1 Plot normal

[ ]:
```
import matplotlib.pyplot as plt

# crear subplot
fig, ax = plt.subplots(1, 1, figsize=(9, 8))

# plot de la serie
ax.plot(time_vals, temp[:].squeeze(), 'r', linewidth=2)

# colocar nombre a los ejes
ax.set_ylabel(f'{temp.long_name.split()[0]} {temp.units}')
# ax.set_ylabel(str(temp.long_name.split()[0]) + ' ' + str(temp.units))
```

```
ax.set_xlabel('Forecast Time (UTC)')

# modificar los nombres de los "ticks"
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
ax.grid(True)
```

```
/tmp/ipykernel_56919/4165639485.py:15: UserWarning: FixedFormatter should only
be used together with FixedLocator
  ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
```



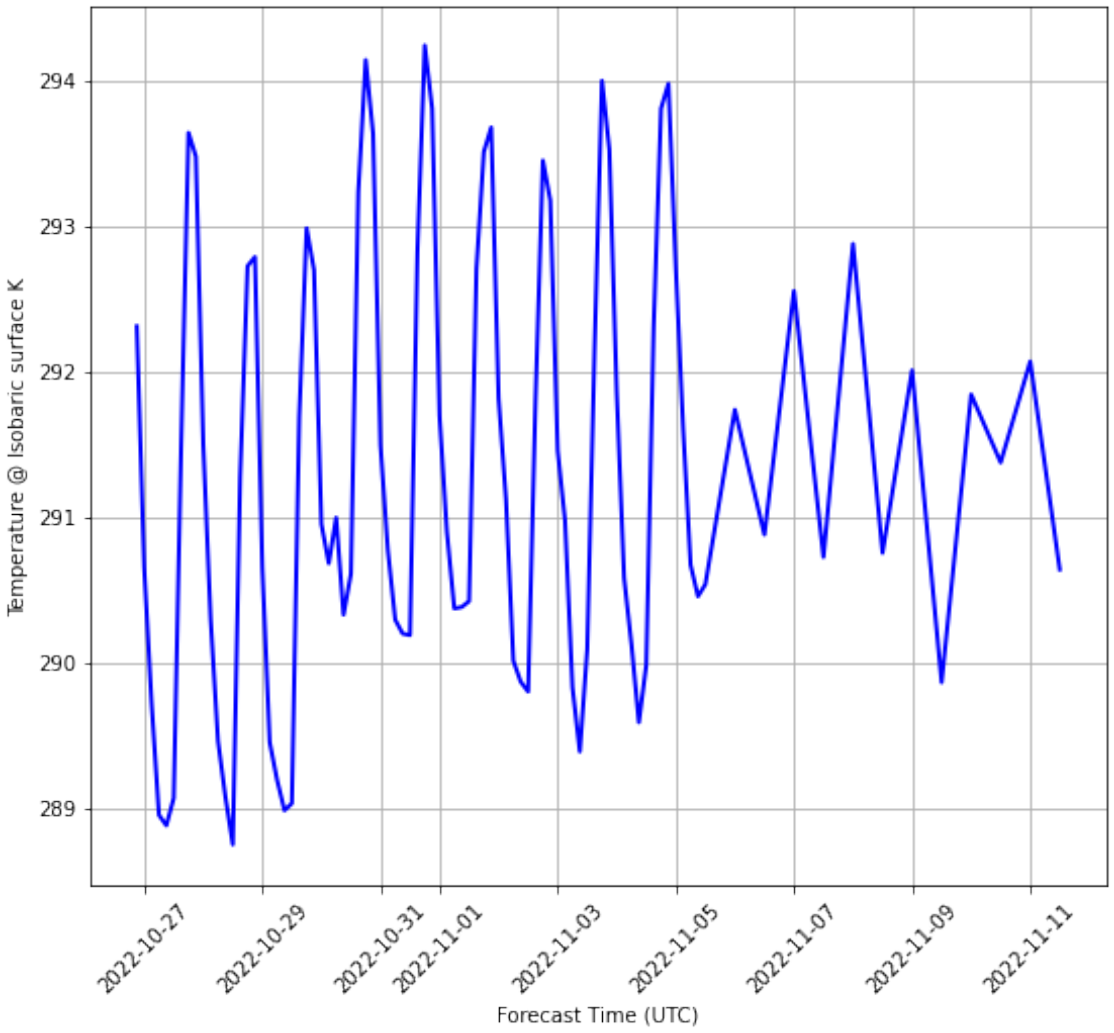
[ ]:
```
data_2
```

[ ]:
```
<xarray.Dataset>
Dimensions:                    (station: 1, obs: 90)
Coordinates:
```

```
    latitude                    (station) float64 -12.0
    longitude                   (station) float64 -77.0
    stationAltitude             (station) float64 1e+05
    time                        (obs) datetime64[ns] 2022-10-26T21:00:00 … …
Dimensions without coordinates: station, obs
Data variables:
    station_id                  (station) |S37 b'GridPointRequestedAt[12.080S…
    station_description         (station) |S37 b'GridPointRequestedAt[12.080S…
    stationIndex                (obs) int32 0 0 0 0 0 0 0 0 … 0 0 0 0 0 0 0 0
    Temperature_isobaric        (obs) float32 292.3 290.7 289.7 … 292.1 290.6
    Relative_humidity_isobaric  (obs) float32 66.8 74.8 79.4 … 71.0 71.6 78.5
Attributes:
    Conventions:          CF-1.6
    history:              Written by CFPointWriter
    title:                Extracted data from TDS Feature Collection null
    featureType:          timeSeries
    DSG_representation:   Timeseries of station data in the indexed ragged ar…
    time_coverage_start:  2022-10-26T21:00:00Z
    time_coverage_end:    2022-11-11T12:00:00Z
    geospatial_lat_min:   -12.0005
    geospatial_lat_max:   -11.9995
    geospatial_lon_min:   -77.0005
    geospatial_lon_max:   -76.9995
```

[ ]:
```python
import matplotlib.pyplot as plt

# crear subplot
fig, ax = plt.subplots(1, 1, figsize=(9, 8))

# plot de la serie temporal
ax.plot(data_2.time, data_2.Temperature_isobaric, 'b', linewidth=2)

# edición de los nombres de los ejes
ax.set_ylabel(f'{data_2.Temperature_isobaric.long_name} {data_2.
 ↪Temperature_isobaric.units}')
ax.set_xlabel('Forecast Time (UTC)')

# edición de los ticks
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
ax.grid(True)
```

```
/tmp/ipykernel_56919/3795037060.py:14: UserWarning: FixedFormatter should only
be used together with FixedLocator
  ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
```



1.2 Plot mejorado

Vamos a utilizar la base disponibilizada por el `geocat-viz` para manipular los ejes

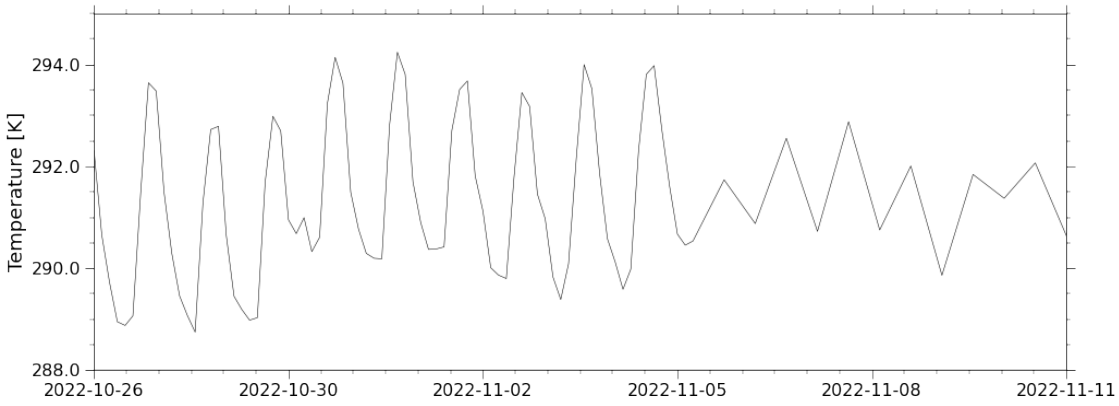
[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator, FormatStrFormatter

import geocat.viz as gv
```

Explorando algunas variables para ver cual es su fucionamiento

[ ]:
```
tiempo
```

[ ]:
```
<class 'netCDF4._netCDF4.Variable'>
float64 time(obs)
    units: Hour since 2022-10-26T12:00:00Z
    long_name: time of measurement
    calendar: proleptic_gregorian
unlimited dimensions: obs
current shape = (90,)
filling off
```

[ ]:
```
# vamos a utilizar el pandas para tener mas control del tiempo
from matplotlib.pyplot import ylabel
import pandas as pd

fechas = pd.to_datetime(data_2.time)

# Confiración de las caracteristicas de la figura
plt.figure(1, figsize=(16, 6))
ax = plt.gca()

# Formato de los "ticks"
ax.xaxis.set_major_formatter(FormatStrFormatter('%d')) # acepta enteros
ax.yaxis.set_major_formatter(FormatStrFormatter('%.1f')) # acepta decimales

plt.plot(fechas, data_2.Temperature_isobaric, color='black', linewidth=0.5)

# agregandole configuraciones a los ejes
gv.add_major_minor_ticks(ax,
                         x_minor_per_major=6, # número de divisiones entre␣
 ↪marcadores principales
                         y_minor_per_major=4,
                         labelsize=16)

# agregandole caracteristicas al los ejes
gv.set_axes_limits_and_ticks(ax,
                             xlim=(fechas.min(), fechas.max()),
                             ylim=(288, 295),
                             xticks=pd.date_range(fechas.min(), fechas.max(),␣
 ↪6),
                             xticklabels=pd.date_range(fechas.min(), fechas.
 ↪max(), 6).date,
                             yticks=np.arange(284, 296, 2))
ax.set_ylabel('Temperature [K]', size=18)

plt.show()
```



[ ]:

1.3 Retornar al indice