```
# %% importazione delle librerie

from urllib.request import urlopen
from bs4 import BeautifulSoup
from datetime import datetime
import pandas as pd
import sys
from io import StringIO
from numpy import mean
import math
import os


#%% libreria axisModifier

def componentAxisTranslationDegree(oldX,oldY,theta):
    newX=oldX*math.cos(math.radians(theta))+oldY*math.sin(math.radians(theta))
    newY=oldY*math.cos(math.radians(theta))-oldX*math.sin(math.radians(theta))
    return newX,newY

def Coord_Cart_to_Polar (x,y):
    intensity = round(math.sqrt(x*x+y*y),3)
    direction=round(math.degrees(math.atan2(y,x)),2)
    if direction <0 : direction =direction +360
    return intensity, direction

def Coord_Cart_to_Polar_North (E,N):

    Int = round(math.sqrt(E*E+N*N),3)
    if E>0 and N>0:
        Dir = round(90-math.degrees(math.atan2(N,E)),2)
       # print ("Primo Quadrante", E, N, Int, Dir)
    elif E>0 and N<0:
        alfa = math.degrees(math.atan2(N, E))
        Dir = round(90 + abs(alfa),2)
        #print ("Secondo Quadrante", E, N, Int, Dir)
    elif E<0 and N<0:
        alfa = math.degrees(math.atan2(N, E))
        Dir = round(90 + abs(alfa),2)
        #print ("Terzo Quadrante", E, N, Int, Dir) #, round(alfa,2))
    elif E<0 and N>0:
        alfa = math.degrees(math.atan2(N, E))
        Dir = round(360 - (abs(alfa) - 90),2)
        #print ("Quarto Quadrante", E, N, Int, Dir) #, round(alfa,2))
    elif E==0 and N>0:
        Dir=0.00
        #print ("Nord secco", E, N, Int, Dir) #, round(alfa,2))
    elif E==0 and N<0:
        Dir=180.00
        #print ("Sud secco", E, N, Int, Dir) #, round(alfa,2))
    elif E>0 and N==0:
        Dir=90.00
        #print ("Est secco", E, N, Int, Dir) #, round(alfa,2))
    elif E<0 and N==0:
        Dir=270.00
        #print ("Ovest secco", E, N, Int, Dir) #, round(alfa,2))
    elif E==0 and N==0:
        Dir=0.00
        #print ("Non c'é corrente", E, N, Int, Dir) #, round(alfa,2))
    return Int, Dir


#%% libreria dfModifier

def dfToOneRow (df):

    df.to_csv("temp.csv", index=False, header=False)

    tempStringObj=open("temp.csv",'r')
    tempString=tempStringObj.read()
    tempStringObj.close()

    tempStringMod = tempString.replace("\n", ",")
    tempStringMod=tempStringMod[:-1]

    tempStringObj=open("temp.csv",'w')
    tempStringObj.write(tempStringMod)
    tempStringObj.close()
    df=pd.read_csv("temp.csv",header=None)
    os.remove("temp.csv")
    return df

def noHeaderDf(df):

    df.to_csv("temp.csv",index=False,header=False)
    df=pd.read_csv("temp.csv",header=None)
    os.remove("temp.csv")
    return df # %% variabili da settare


# %% variabili da settare

startUrlString=["https://nodc.ogs.it/erddap/tabledap/CURRISO_PR.htmlTable?
                time%2Clatitude%2Clongitude%2Cdepth%2CVCSP%2CVCSP_QC%2CEWCT
                %2CEWCT_QC%2CNSCT%2CNSCT_QC&time%3E",
                "https://nodc.ogs.it/erddap/tabledap/CURRISO_TS.htmlTable?
                time%2Clatitude%2Clongitude%2CPRES%2CPRES_QC%2CTEMP%2CTEMP_QC
                %2CRVFL%2CRVFL_QC&time%3E"];
endUrlString=["&orderBy(%22time%2Cdepth%22)","&orderBy(%22time%22)"];


# %% controllo dateList per definizione datetime

tempTimeObj=open("dateList.txt",'r')
tempTime=tempTimeObj.read()
tempTimeObj.close()

if tempTime =="":
    print("\nil file dateList deve contenere la data da cui si vuole \
          far partire l'acquisizione del record\n")
    sys.exit()

newDateTime=datetime.strptime(tempTime, "%Y-%m-%dT%H-%M-%SZ")


# %% lettura dei dati web attraverso BeautifulSoup

newUrlString=[]
dfList=[]
```

```
try:
    for urlIdx in range(len(startUrlString)):

        newUrlString.append(startUrlString[urlIdx]+ \
                    newDateTime.strftime("%Y-%m-%dT")+ \
                    newDateTime.strftime("%H")+ \
                    "%3A"+ \
                    newDateTime.strftime("%M")+ \
                    "%3A"+ \
                    newDateTime.strftime("%SZ") +\
                    endUrlString[urlIdx] );

        html = urlopen(newUrlString[urlIdx]);
        soup= BeautifulSoup(html.read(),'html.parser');
        htmlTable=soup.find_all("table",{"class":"erd commonBGColor nowrap"});
        df=pd.read_html(StringIO(str(htmlTable)))[0];
        dfList.append(df);
except:
    print("\nnon ci sono nuovi record\n")
    sys.exit()


# %% creazione finalDf con data e ora

timeDf=pd.DataFrame()
finalDfAvrg=pd.DataFrame()

timeDf['tempDate'] = pd.to_datetime(dfList[1]['time','UTC'], \
                                    format='%Y-%m-%dT%H:%M:%SZ');
timeDf['dateTime'] = timeDf['tempDate'].dt.strftime("%Y-%m-%dT%H-%M-%SZ");
timeDf=timeDf.drop('tempDate',axis='columns');
timeDf['latitude'] = dfList[1]['latitude','degrees_north']
timeDf['longitude'] = dfList[1]['longitude','degrees_east']


#%% elaborazione dataframe con profili

dfList[0]=dfList[0].drop(['latitude','longitude','VCSP_QC','EWCT_QC',\
                          'NSCT_QC'], axis='columns');

dfList[0]=noHeaderDf(dfList[0])

profileColoumnName=["time","depth", "VCSP", "EWCT", "NSCT"]
dfList[0]= dfList[0].set_axis(profileColoumnName, axis=1)

gr=dfList[0].groupby('time')
keys=list(gr.groups.keys())

finalProfileDf=pd.DataFrame()
finalProfileDfAvrg=pd.DataFrame()

currAvrg=pd.Series(dtype="float64")

for keyIdx in range (len(keys)):

    #parte di riempimento del df con tutti i profili
    dfProfile=gr.get_group(keys[keyIdx])
    dfProfile=dfProfile.drop('time',axis='columns');
    dfProfile=dfProfile.iloc[::-1]
    dfProfile=dfProfile.reset_index(drop=True)
    dfProfile.insert(0,"#",list(range(1,len(dfProfile)+1)))

    dfProfile=dfToOneRow(dfProfile)
    finalProfileDf=pd.concat([finalProfileDf,dfProfile],ignore_index=True)

    #parte di riempimento del df con le medie, intensità e direzione
    dfProfileMultRow=gr.get_group(keys[keyIdx])
    dfProfileMultRow=dfProfileMultRow.drop(['time','depth','VCSP'],\
                                           axis='columns');
    dfProfileMultRow=dfProfileMultRow.iloc[::-1]
    dfProfileMultRow=dfProfileMultRow.reset_index(drop=True)

    if len(dfProfileMultRow) % 2 == 1:
        dfProfileMultRow=dfProfileMultRow.drop(dfProfileMultRow.index[0])
        dfProfileMultRow=dfProfileMultRow.reset_index(drop=True)

    EWCT_BottomAvrg= \
        round(mean(dfProfileMultRow['EWCT'].to_list()[0:int(len(dfProfileMultRow)/2)]),3)
    NSCT_BottomAvrg= \
        round(mean(dfProfileMultRow['NSCT'].to_list()[0:int(len(dfProfileMultRow)/2)]),3)
    EWCT_TopAvrg= \
        round(mean(dfProfileMultRow['EWCT'].to_list()[int(len(dfProfileMultRow)/2):]),3)
    NSCT_TopAvrg= \
        round(mean(dfProfileMultRow['NSCT'].to_list()[int(len(dfProfileMultRow)/2):]),3)

    [Int_Bottom,Dir_Bottom]=Coord_Cart_to_Polar_North(EWCT_BottomAvrg,NSCT_BottomAvrg)
    [Int_Top,Dir_Top]=Coord_Cart_to_Polar_North(EWCT_TopAvrg,NSCT_TopAvrg)

    Int_avrg=mean([Int_Top,Int_Bottom])
    currAvrg=currAvrg.append(pd.Series(Int_avrg))

    tempSrAvrg=pd.Series([EWCT_BottomAvrg, NSCT_BottomAvrg, \
                          Int_Bottom,Dir_Bottom,\
                          EWCT_TopAvrg,NSCT_TopAvrg, \
                          Int_Top,Dir_Top])
    finalProfileDfAvrg=finalProfileDfAvrg.append(tempSrAvrg,ignore_index=True)

currAvrg=currAvrg.reset_index(drop=True)


#%% elaborazione dataframe con timeseries

dfList[1]=dfList[1].drop(['time','latitude','longitude', \
                          'PRES_QC','TEMP_QC','RVFL_QC'],axis='columns');

dfList[1]=noHeaderDf(dfList[1])


#%% concatenazione diversi dataframe in quello finale

finalDf=pd.concat([timeDf,dfList[1], finalProfileDf],axis=1)
finalDfAvrg=pd.concat([timeDf,dfList[1], finalProfileDfAvrg],axis=1)


# %% set coloumn name

startColoumnName=["dateTime", "latitude", "longitude"]
profileColoumnName=[]
for rowIdx in range(int(len(finalProfileDf.columns)/5)):
    profileColoumnName=profileColoumnName+["PROFILE #","DEPTH", "VCSP", \
                                           "EWCT", "NSCT"]
tsColoumnName=["PRES","TEMP","RVFL"]

finalColoumnName=startColoumnName+tsColoumnName+profileColoumnName
finalDf = finalDf.set_axis(finalColoumnName, axis=1)

avrgColumns=['EWCT_BottomAvrg','NSCT_Bottom','Int_BottomAvrg','Dir_BottomAvrg',\
             'EWCT_TopAvrg','NSCT_TopAvrg','Int_TopAvrg','Dir_TopAvrg']
finalColoumnNameAvrg=startColoumnName+tsColoumnName+avrgColumns
finalDfAvrg = finalDfAvrg.set_axis(finalColoumnNameAvrg, axis=1)


#%% scrittura del dataframe finale su file
currentDateTime=datetime.now()
currentDateTimeString=currentDateTime.strftime("%Y-%m-%dT%H-%M-%SZ")
csvFileName='csv/ERDDAP_CurrIso_'+currentDateTimeString+'.csv'
csvFileNameAvrg='csv/ERDDAP_CurrIso_'+currentDateTimeString+'_Avrg.csv'

finalDf.to_csv(csvFileName,index=False,float_format = '%.12g')
finalDfAvrg.to_csv(csvFileNameAvrg,index=False,float_format = '%.12g')

endTime=dfList[0].time[len(dfList[0])-1]
endTime=datetime.strptime(endTime, '%Y-%m-%dT%H:%M:%SZ')
timeString=datetime.strftime(endTime, "%Y-%m-%dT%H-%M-%SZ")

tempTimeObj=open("dateList.txt",'w')
tempTimeObj.write(timeString)
tempTimeObj.close()
```