RISIS

RESEARCH INFRASTRUCTURE FOR SCIENCE AND INNOVATION POLICY STUDIES

**On-line Course**

# *Laboratory Session: Developing Imputation Techniques in Python*

**D. Aureli**

**Davide.Aureli@uniroma1.it**

*Università di Roma "Sapienza"*

*Dip. di Ingegneria dell'Informazione, Elettronica e Telecomunicazioni (DIET)*

# Outline

- Part 1: Machine Learning Tools
- Part 2: The Imputation with ETER Data
  - Trend Smoothing Code
  - Donor Code
- Part 3: Conclusions & Future Works

# Machine Learning (ML) Tools

Platform: **Anaconda** (https://docs.anaconda.com/anaconda/install/)

Programming Language: **Python**

Software: Integrated Development Environment (IDE)

**Jupyter** / **Spyder**

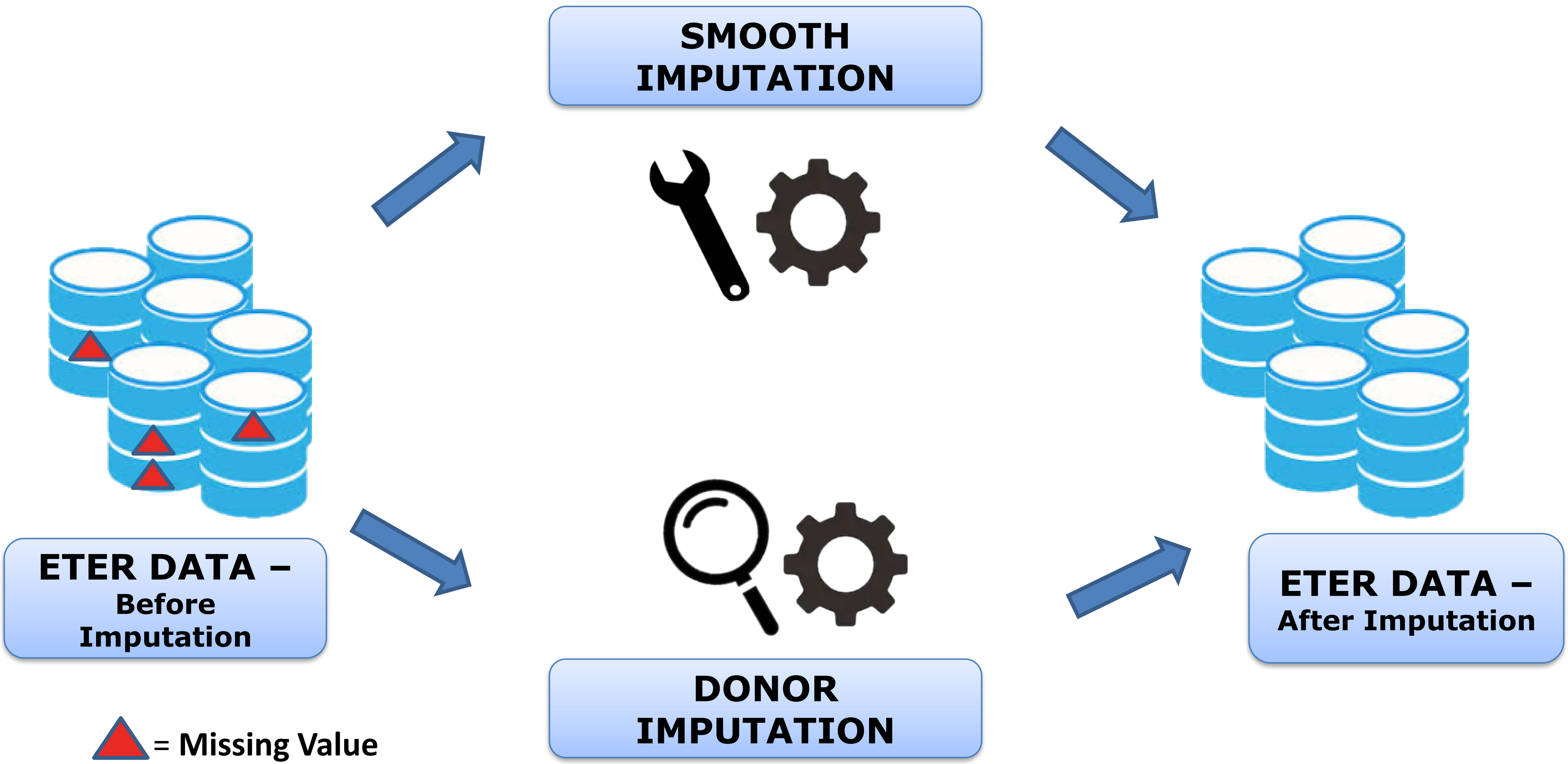
Library: Pandas (https://pandas.pydata.org/)

Scikit-Learn (https://scikit-learn.org/stable/)

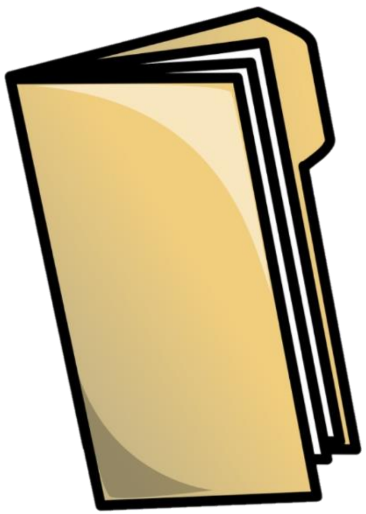
Recommended Lecture*: Prof. Renato Bruni, ‘’*Optimization and Machine Learning for the Imputation of Missing Interconnected Data*”

**(strongly suggested before this lecture)*

# Workflow Overview



# Directory Structure



*original_dataset.xlsx*
*Columns_Final_Name.xlsx*

*1_smooth_imputation.py*
*2_merge_smoothfiles.py*
*3_donor_imputation.py*
*4_donor_imputation_relaxed.py*
*5_smooth_after_donor.py*
*6_merge_smoothfiles.py*
*7_add_ratios_and_trends.py*

*columns_ordered.pkl*
*columns_ordered_bibliometric.pkl*

# Python Code Pipeline

1_smooth_imputation.py

2_merge_smoothfiles.py

→ Trending Smooth Imputation

3_donor_imputation.py

4_donor_imputation_relaxed.py

5_smooth_after_donor.py

6_merge_smoothfiles.py

→ Donor & Donor Relaxed Imputation

7_add_ratios_and_trends.py

→ Statistical Analysis

# Imputation Procedure

- ## Objective:

In our work we want to reconstruct as much as possible missing values through the Machine Learning procedures. We consider 2 different types of missing value: **single** or **small sequence** and **almost complete** or **full sequence**. These different situations are tackled with 2 different methods : **Smooth Imputation** or **Donor Imputation**.

- **Smooth Imputation:** *(Impute from the same Institution)* Combination of the weighted average and the linear regression considering the time series.

- **Donor Imputation**: *(Impute from the Donor Institution)* For each Institution under imputation, find a similar complete Institution (Donor) at least for the variable we need,and use the values of the latter to impute the missing values of the former.

# Smooth Imputation

**Main Steps:**

- 1. Data Preparation & Cleaning Part.
- 2. ETER ID check about **Missing Sequence** (single, consecutive or full).
- 3. Extraction of Training and Test Data, computing **Linear Regression** Model and **Weighted Average.**
- 4. Combination between LR and WA :

$$v_i = (a^2 / a^2 + 1)\, v_i^{WA} + (1 / a^2 + 1)\, v_i^{LR}$$

Coefficient $a = \dfrac{2|m|}{\min\limits_{h \neq i} \{v_h\}}$

m = *slope* of LR model ,

$\min \{v_h\}$ minimum available value

- 5. If the result is **Negative**, really unfeasible, we use the **Exponetiation Operation** applied with an *exponent* in *(0,1) to obtain a positive value in line with the Trend.*
- 6. Add Imputed values in the final Dataset.

# Different types of Missing Values

**Check Missing Sequence**

```
def checkSequenceMissing(dictionary):

    #List of values
    lista_val = list(dictionary.values())
    #Index first missing
    start_index = lista_val.index("m")
    #Number of Missing
    tot_missing = Counter(dictionary.values())["m"]
    #Boolean val for the missing sequence
    sequence = "OK"

    for i in range(tot_missing):
        if lista_val[start_index] != "m":
            #Here we detect not a consecutive missing values
            sequence = "Not consecutive"
        else:
            start_index += 1

    return sequence
```

# Institution Data Extraction

```
# SMOOTH Function

#Imputed value using linear regression (We exploit the implementation of Linear Regression model from Scikit-learn)
from sklearn import  linear_model


#The year are selected according to the data available. The user can modify the list according to the available data.
def imputation_missing_LR_WA(dataset, variable, year=[2017, 2016, 2015,2014,2013,2012,2011]):

    not_consecutive_missing = 0
    tutti_coef = []

    # Working considering the ETER ID
    for institute in tqdm(sorted(list(set(dataset["ETER ID"])))):

        print("Working with this Institute: " + str(institute))
        #print()

        #Extract a small dataset about the info of that ETER ID
        a = DataFrame(dataset[dataset["ETER ID"] == institute].copy())

        #Extract just info about the imputed variable and Reference year
        d = DataFrame(a[["Reference year",variable]].copy())

        #Dictionary Creation {Year:value: for instance 2016:31, 2015:20, 2014:m......2011:11}
        valori = dict(d.values.tolist())
```

Select ETER ID

# Model Preparation

```
#Function transforms missing value "m", we need of at least 2 real values to make our prediction

#Check about it and the presence of at least 1 element in the dictionary.

if len(valori.keys()) >= 1 and len(valori.keys()) <=7 and Counter(list(valori.values()))["m"] >=1 and Counter(list(valori

    print("We work with: " + str(institute))
    #print(valori)

    #Check about the missing typology
    if any(i in valori.values() for i in replace) == False:
        print("This Institue "+ institute + " does not have missing different from m")
        seq = checkSequenceMissing(valori)

        #Here Institution with missing in different years not consecutive
        if seq == "Not consecutive":
            #Used in the weighted average
            fattore_peso = 2
            not_consecutive_missing += 1
            #print(institute)
            #print(valori)
        #Consecutive Missing
        else:
            #Used in the weighted average
            fattore_peso = 10

        #Small dataset composed by English Institution Name, Year and Imputed Variable; sorted by the year.
        pp = a[["English Institution Name","Reference year",variable]]
        pp = pp.sort_values(by=["Reference year"])

        #Linear Regression Part
        Test = []
        Train = []

        #Indexes of the small dataframe
        indici = list(pp.index.values)
        valori_cambiati_indici = []

        for i in range(len(pp)):
            #2 is the column imputed
            if pp.iloc[i,2] == "m":
                Test.append(i)
                valori_cambiati_indici.append(indici[i])
            else:
                Train.append(i)
```

Type of Missing

Train & Test Data

# Linear Combination: LR - WA

```python
#Training --> Rows with data
#Test --> Rows with missing("m")

#In position 1 there is our X independent variable (Reference Year).

X_train = pp.iloc[Train,1]
X_test = pp.iloc[Test,1]

y_train = pp.iloc[Train,2]
y_test = pp.iloc[Test,2]


# Create linear regression object
regr = linear_model.LinearRegression()

#Make the Reshape, cause X must be a Matrix
# Train the model using the training set
regr.fit(np.array(X_train).reshape(-1,1), y_train)

# Make predictions using the testing set
pred = regr.predict(np.array(X_test).reshape(-1,1))

#From an array to list our prediction
pred = [int(elem) for elem in list(pred) ]
```

**Linear Regression Model**

**Weighted Average**

```python
#WEIGHTED AVERAGE PART according to the Temporal r.

if Test[0] > Train[-1]:

    numeratore = 0
    peso = 1
    tot_peso = 0

    for elem in y_train:

        peso *= fattore_peso
        numeratore += peso*elem
        tot_peso += peso

    #WA Computation
    media_pesata = numeratore/tot_peso

elif Test[0] < Train[-1]:

    numeratore = 0
    peso = len(y_train)*1000
    tot_peso = 0

    for elem in y_train:

        peso = peso/fattore_peso
        numeratore += peso*elem
        tot_peso += peso

    #WA Computation
    media_pesata = int(numeratore/tot_peso)

print("Values from the analysis: LR & WA \n")
print("Starting with these values: \n")
print(valori)
print()
print("Coeff (Slope) for LR Model")
print(regr.coef_)
print("Linear Regression Prediction")
print(pred)
print("Weighted Average Prediction")
print(media_pesata)
print()
```

# Smooth Value Computation

```python
#We need to normalize the angular coefficient for the growth, going from 1 to 1000 is different
#respect to go from 1000 to 2000

normalizzatore = min(y_train)

#Here we make a check for the infinite values, in case where the min is 0
if normalizzatore == 0:
    normalizzatore = 1


#Our Coefficient "a" (Coefficient "a" in Linear Combination between LR and
#Nome Variabile = coeff_a

#All Variables except PhD
coeff_a = 2* abs(regr.coef_) / normalizzatore
#PhD Students
#coeff = 4* abs(regr.coef_) / normalizzatore


tutti_coef.append(coeff_a)

#coeff = abs(regr.coef_)
print("Our 'coeff_a' is:  " + str(coeff_a[0]))

#Linear Combination between Linear Regression(LR) and Weighted Average(WA)
#Smooth =  [a^2 / (a^2 + 1)](WA) +  [1 / (a^2 + 1)](LR)
# if a --> 0 LR will have a coefficient equals to 1, otherwise WA increases its weight.

final_val_imputed = []

for i in range(len(pred)):
    vvv = pred[i]
    if vvv < 0 :
        final_val_imputed.append(vvv)
    else:
        linear_comb = (coeff_a[0]**2/(coeff_a[0]**2 +1 ))*media_pesata + (1/(coeff_a[0]**2 +1 ))*pred[i]
        final_val_imputed.append(linear_comb)

print()
print("Value Imputed")
print(final_val_imputed)
print()
print()

for i in range(len(valori_cambiati_indici)):
    dataset[variable][valori_cambiati_indici[i]] = round(final_val_imputed[i],2)
```

«a» Coefficient

# Exponentiation Operation

```
# =============================================================================
# ## Sweeten Negative Prediction
# =============================================================================

#SmoothExponent
SmoothExponent = 0.6

def changeNegativeValues(prova):

    indici_negativi = []

    for ind, vv in enumerate(prova):
        if vv not in missing_completi and vv <0:
            indici_negativi.append(ind)

    if len(prova)-1 in indici_negativi:
        for i in range(len(indici_negativi)):

            #prova[indici_negativi[i]] = int(prova[indici_negativi[i]-1]*(SmoothExponent**(i+1)))
            prova[indici_negativi[i]] = int(prova[indici_negativi[i]-1]*(SmoothExponent))

    else:

        #multiple = 1
        for i in range(len(indici_negativi)-1,-1,-1):

            #prova[indici_negativi[i]] = int(prova[indici_negativi[i]+1]*(SmoothExponent**(multiple)))
            prova[indici_negativi[i]] = int(prova[indici_negativi[i]+1]*(SmoothExponent))


            #multiple += 1

    return prova
```

**Avoid Negative Values**

# Smooth Example - Students

**Eter ID: FR0026**

```
Working with this ETER ID Institution: FR0026
This Institution does not have missing different from 'm'

Starting Values:
Year   -   Studets
2017   -   18728
2016   -   'm'
2015   -   'm'
2014   -   17203
2013   -   16808
2012   -   16618
2011   -   16143

Values from the analysis LR and WA
Coeff (Slope) for LR Model: 425.05
Linear Regression Prediction: [17780 , 18205]
Weighted Average Prediction: 16192

Our 'coeff_a': 0.053
Value Imputed: [2015:17776, 2016:18199]
```

# Smooth Example - Students

**Eter ID: SI0022**

```
Working with this ETER ID Institution: SI0022
This Institution does not have missing different from 'm'

Starting Values:
Year   -   Studets
2016   -   19
2015   -   24
2014   -   26
2013   -   'm'
2012   -   'm'
2011   -   'm'

Values from the analysis LR and WA
Coeff (Slope) for LR Model: -3.5
Linear Regression Prediction: [37, 33, 30]
Weighted Average Prediction: 25

Our 'coeff_a': 0.368
Value Imputed: [2011:36, 2012:32, 2013:29]
```

# Donor Imputation

**Main Steps:**

- 1. Data Preparation (add Country Similarity) & Cleaning Part.
- 2. ETER ID check about **Missing Sequence** (*almost complete* or *full*).
- 3. Creation of the container for all possible **Donor Institutions.**
- 4. **“Window” filter** on categorical variables and Size, Trend and Ratios.
- 5. Computation of the **Distance Function** based on multiple features, with the extraction of the nearest Donor Institution.
- 6. Value imputation, if possible with the application of a **normalization method**.
- 7. Add Imputed values in the final Dataset.

# Initial set of Donors

```
# =============================================================================
# Selection of DONOR Institutions
# =============================================================================
#DONATORI with valid values in all variables considered.

values_good_donor = 5

replace = ["a", "x", "xc", "xr", "nc", "c", "s", "Null"]

#Create a dictionary with all the variables that we can donate
donatori_diz_poss = {"completi":[], "non_completi":[]}

for i in set(imputation_test["ETER ID"]):

    #The Flag "buono" counts the number of variables a donor can donate.
    buono = 0

    for var in variabili_imputazione:
        valori = list(imputation_test[imputation_test["ETER ID"] == i][var])
        if any(i in valori for i in replace) == False:

            #Add Check for the number of 0 missing values
            if Counter(valori)["m"] == 0 and (len(valori) == 6 or len(valori) == 7):
                buono += 1

    if buono == len(variabili_imputazione):
        donatori_diz_poss["completi"].append(i)
     #Specify number of variables to consider an Institution as "good" Donor in the list "non_completi"
    if buono > values_good_donor:
        #At least 5 variables completed
        donatori_diz_poss["non_completi"].append(i)

#Check to maintain the number of times an Institution will be selected as Donor
#Choose the typology of Institution that can be Donor

#donatore_scelto = { nn:0 for nn in donatori_diz_poss["completi"]}
donatore_scelto = { nn:0 for nn in donatori_diz_poss["non_completi"]}

#For the moment we work with Institutions complete in all the variables.

#donatori_completi = donatori_diz_poss["completi"]
donatori_completi = donatori_diz_poss["non_completi"]

print("Total Number of Donor selected: " + str(len(donatori_completi)))
```

**Donor Selection**

# Applying filters on the initial set of Donors

Categorical Window

```
# ==============================================================================
# WINDOW Analysis - Check used during the first part of DONOR Imputation
# ==============================================================================

#Categorical Window Analysis

def finestraCategorica(dataset, lista_var_categoriche, donatori, imputato):

    data_work = dataset[dataset["ETER ID"] == imputato].copy()
    diz_finestra = {}

    for var in lista_var_categoriche:
        #We take the first value for the categorical feature cause should be the
        #same for each year.
        diz_finestra[var] = list(data_work[var])[0]
    #print(diz_finestra)

    #"donatori" has to be a list.
    data_donatori = dataset[dataset["ETER ID"].isin(donatori)]

    for var_controllo in lista_var_categoriche:
        data_donatori = data_donatori[data_donatori[var_controllo] == diz_finestra[var_controllo]]

    donatori_finali = list(set(data_donatori["ETER ID"]))

    return donatori_finali

var_categoriche_check = ["Institution Category standardized","Country Code"]
```

# Filters on the set of Donors

```
#Main part Window Filter Analysis

diz_final_id_size = {}
diz_final_id_trend = {}

print()
print("Identification of feasible Donors")
print()

for univ_imputata in tqdm(diz_eterid_da_imputare):

    #print("Working with " + str(univ_imputata))
    #print(len(donatori))

    # 1 - Window for the Categorical filter

    donatori_prima_finestra = finestraCategorica(imputation_test, var_categoriche_check, donatori, univ_imputata )
    #print(len(donatori_prima_finestra))


    #Check about the number of donor Institutions we find out

    if len(donatori_prima_finestra) == 0:
        #print("No Donor respects the CATEGORICAL Window")
        #print()
        #print()
        pass
    else:
        #print("Go On - SIZE Window")

        # 2 - Window for the size filter
        donatori_seconda_finestra, diz_size_id_val = finestraSize(imputation_test, variabili_correlazione, donatori_prima_finestr
                                                                  univ_imputata )

        #print(len(donatori_seconda_finestra))
        #print()

        diz_final_id_size[univ_imputata] = diz_size_id_val
        #print(diz_size_id_val)
        #print()

        #Check about the number of donor Institutions we find out.
        if len(donatori_seconda_finestra) == 0:
            #print("No Donor respects the SIZE Window")
```

**Initial Set of Selected Donors**

**Set of Selected Donors is reduced**

# Filters on the set of Donors

```
# 3 - Window for the Trend filter

donatori_terza_finestra, diz_trend_id_val = finestraTrend(imputation_test, variabili_correlazione, donatori_seconda_f
                                             univ_imputata )

#print(len(donatori_terza_finestra))

diz_final_id_trend[univ_imputata] = diz_trend_id_val
#print(diz_trend_id_val)
#print()

if len(donatori_terza_finestra) == 0:

    #print("No Donor respects the TREND Window")
    #print()
    #print()
    pass

else:


    # 4 - Window for the Ratios filter

    #print("Go On - RATIOS Window")

    donatori_quarta_finestra = finestraRatio(imputation_test, colonne_rapporto, donatori_terza_finestra,
                                             univ_imputata )

    #print(len(donatori_quarta_finestra))
    #print()
    #print()


    diz_accoppiamento_imputato_donatori[univ_imputata] = donatori_quarta_finestra
```

Reduced Again

# Distance Function: using Country

```python
# ============================================================================
# SIMILARITY NATIONS - Add Feature
# ============================================================================

#This part is useful for the computation on the distance to choose a possible Donor
#considering also the feature about Geographical region.

similarity_country = {}

#All possible Country Code
country = set(list(starting_dataset["Country Code"]))

country = ["NL", "BE", "LU", "CH","LI", "DE", "AT", "IT", "ES", "GR", "PT", "HU", "CZ", "LT", "LV", "PL", "EE","SK",
           "AL","BG","HR","ME","SI","MK","RS","RO", "NO", "SE", "DK", "FI","IS", "UK", "IE", "MT", "FR", "CY","TR"]


for naz in country:
    if naz in ["NL", "BE", "LU", "CH","LI"] :
        similarity_country[naz] = 1
    elif naz in ["DE", "AT"]:
        similarity_country[naz] = 2
    elif naz in ["IT", "ES", "GR", "PT"]:
        similarity_country[naz] = 3
    #EST del Nord
    elif naz in ["HU", "CZ", "LT", "LV", "PL", "EE","SK"]:
        similarity_country[naz] = 4
    #EST del SUD
    elif naz in ["AL","BG","HR","ME","SI","MK","RS","RO"]:
        similarity_country[naz] = 5
    elif naz in ["NO", "SE", "DK", "FI","IS"]:
        similarity_country[naz] = 6
    elif naz in ["UK", "IE", "MT"]:
        similarity_country[naz] = 7
    elif naz in ["FR"]:
        similarity_country[naz] = 8
    elif naz in ["CY","TR"]:
        similarity_country[naz] = 9
    else:
        #Check for the Nation without a specific region.
        print(naz)
```

Country Similarity

# Distance Funct. : using Size, Trend, …

```
def creareVettore_Knn(name_imputato, lista_donatori, diz_valori_imputato, diz_size, diz_trend):

    #Big vector for all the distances
    distanze = []
    missing = ["a", "x", "xc", "xr", "nc", "c", "s", "Null", "m"]

    for nome in lista_donatori:
        #print("Working with " + nome)

        indice_donatore = imputation_test[imputation_test["ETER ID"] == nome].index[0]

        distanza_singola_univ = []

        for var in diz_valori_imputato:

            #print(var)
            #print()

            if var == "Size":

                distanza_singola_univ.append(calcoloMaxVariazione(diz_size[nome]))

                #print("SIZE")
                #print(calcoloMaxVariazione(diz_size[nome]))


            if var == "Trend":

                distanza_singola_univ.append(calcoloMaxVariazione(diz_trend[nome]))


                #print("TREND")
                #print(calcoloMaxVariazione(diz_trend[nome]))

            if var == "Num_Val_Selected":

                #This function compute the number of variables the Donor can give to the
                #Institution imputed
                distanza_singola_univ.append(calcoloNumeroMissing(name_imputato,nome))
```

KNN Vector

# Distance Funct.: other variables

```python
if var == "Institution Category standardized" :

    valore_donat = imputation_test.loc[indice_donatore]["Institution Category standardized"]
    #print(valore_donat)
    #print(diz_valori_imputato[var])

    if  valore_donat == diz_valori_imputato[var]:
        distanza_singola_univ.append(0)
    elif valore_donat in missing:
        #distanza_singola_univ.append(np.NaN)
        distanza_singola_univ.append(3)
    else:
        distanza_singola_univ.append(3)


if var == "Institution Category - English" :

    valore_donat = imputation_test.loc[indice_donatore]["Institution Category - English"]

    #print(valore_donat)
    #print(diz_valori_imputato[var])

    if  valore_donat == diz_valori_imputato[var]:
        distanza_singola_univ.append(0)
    elif valore_donat in missing:
        distanza_singola_univ.append(3)
        #distanza_singola_univ.append(np.NaN)
    else:
        distanza_singola_univ.append(3)

if  var == "Distance education institution":

    valore_donat = imputation_test.loc[indice_donatore]["Distance education institution"]

    #print(valore_donat)
    #print(diz_valori_imputato[var])

    if valore_donat == diz_valori_imputato[var]:
        distanza_singola_univ.append(0)
    elif valore_donat in missing:
        distanza_singola_univ.append(3)
        #distanza_singola_univ.append(np.NaN)
    else:
        distanza_singola_univ.append(3)
```

**KNN Vector**

# Donor Example – All Variables

**Imputed: DE0256 → Donor: DE0245**

```
Working with ETER ID DE0256

Possible DONOR Institutions: 508

Categorical Filter
Available Categorical Variables: {'Institution Category standardized': 2, 'Country Code': 'DE'}
DONOR Institutions selected: 89

Size Filter
Available Size Variables: {}
DONOR Institutions selected: 89

Trend Filter
Available Trend Variables: {}
DONOR Institutions selected: 89

Ratio Filter
Available Ratio Variables: {}
DONOR Institutions selected: 89
```

**Donors after Filtering**

```
[ DE0337 DE0341 DE0211 DE0189 DE0312 DE0315 DE0366 DE0338 DE0232
DE0316 DE0302 DE0334 DE0234 DE0309 DE0247 DE0271 DE0226 DE0307
 ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
 DE0224 DE0369 DE0372 DE0265 DE0227 DE0248 DE0313 DE0205 DE0295]
```

```
Working with: DE0256
Possible DONOR Institutions: 89

Variables Available for Distance Computation:
{'Institution Category standardized': 2, 'Institution Category- English': 'university of applied sciences',
'Country Code': 'DE', 'Distance education institution': 0, 'Legal status': 1, 'Num_Val_Selected': 0}

Best DONOR available: DE0245 (Katholische Stiftungsfachhochschule München) , Distance: 2.0
```

# Donor Example – Rescaled Values

**Imp.: FI0024 → Donor: FI0021**

```
Working with this ETER ID Institution: FI0024

(After Donor Imputation - No Available Institutions)
Start Relaxed Donor Imputation

Best DONOR available: FI0021, Distance: 6.2

Start Normalization
Available Variable:{Students_afterSmooth, Graduates_afterSmooth}
Selected Variable Normalization: Students_afterSmooth

FI0024 - Students{Year:Value}
{2016:824, 2015:840, 2014:861, 2013:788, 2012:774, 2011:659}

FI0021 - Students{Year:Value}
{2016:4020, 2015:4032, 2014:4002, 2013:4240, 2012:4338, 2011:4517}

FI0021 - Academic Staff FTE{Year:Value}
{2016:305.2959, 2015:297.3551, 2014:295.7779, 2013:304.2254, 2012:290, 2011:284.445}

Final Imputation for Academic Staff FTE - FI0024
{2016:63, 2015:62, 2014:64, 2013:57, 2012:52, 2011:41
```

**Normalization**

**Imputed Value**

# Conclusions

- **Fast** imputation of several types of missing values.
- Different imputation techniques for **short** or **full sequences**.
- Really **flexible** according to the request of the user.
- Applicable to **every ETER variable**.
- Could be easily adapted to other educational data, or other **interconnected** data with different origin.
- Code avialble in Python, can take advantage of several libraries for instance **Pandas** to handle with *Excel files* and **Scikit-learn** with many <u>Machine Learning Tools</u>.

# Further Developments

- For many incomplete Institutions, we have **scarcity of Donors**: missing values are often too much and they are concentrated on some types of Institutions, so those types are very difficult to impute. Possible other sources for *data integration* ?

- **Smooth Imputation** works initially with a univariate model, and later Ratios are used to connect the different variables. Maybe a native *multivariate extension* is possible ?

- If a Donor contains errors, its **errors are propagated** when it is used for imputation. We plan to study further filtering techniques for Donors.

- Full **Parallelization** ?

THANK YOU

for your attention!

Any Questions ?