PACKAGES FOR MACHINE LEARNING

Performance evaluation metrics

The following are two of the most commonly used metrics in the evaluation of regression models:

- *RMSE – Root Mean Square Error*

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - f(x_i))^2}{n}}$$

- *Coefficient of Determination (R^2)*

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - f(x_i))^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Being:

- n the number of observations used for testing
- y_i i $-$ the real value of dependent variable
- $\bar{y}$ the average of the actual values of the dependent variable
- x_i the i-th real value of the independent variable
- $f(x_i)$ the "i-th predicted value of the dependent variable (the value provided by the model for y_i)
- $y_i - f(x_i)$ the i-th mistake made by the prediction model

Evaluation of the model with the test set - using the RMSE metric

- For a better understanding, let's apply the RMSE formula in several steps:

```
print(np.ravel(notas_test))
```

```
[16.  14.8  3.4  6.8  4.5]
```

Real grades

```
print(np.ravel(notas_estimadas))
```

```
[16.8 13.6  5.   6.4  1.9]
```

Grades provided for by the model

```
erros=notas_test-notas_estimadas #erros das estimativas
print(np.ravel(erros))
```

```
[-0.8  1.2 -1.6  0.4  2.6]
```

The ravel() function converts a multidimensional array into an array with a single dimension (only to show on a single line)

```
erros_quadraticos=erros**2 #quadrado dos erros
print(np.ravel(erros_quadraticos))
```

```
[0.7 1.5 2.5 0.2 6.6]
```

```
erros_quadraticos.mean() #(Erro Quadrático Médio)
print(round(erros_quadraticos.mean(),2))
```

```
2.28
```

```
RMSE=np.sqrt(erros_quadraticos.mean()) #Raíz do Erro Quadrático Médio
print(round(RMSE,2))
```

```
1.51
```

Therefore, the RMSE error made by the prediction model was 1.51 (a fairly acceptable value, given the range of variation of the score)

Evaluation of the model with the test set - using the metric R^2

- To obtain the Coefficient of Determination, we simply invoke the score() method of the model itself:

```
R2=modelo.score(horas_test,notas_test)
print(round(R2,2))
```

```
0.92
```

To calculate R2, the score() method begins by calculating the estimates for the grades based on the hours we receive for, only then, using these estimates along with the actual grades we also receive, determine R^2

- In other words, a R^2 of 92%
  - *that reveals great performance*

  (a R^2 of 0.92 means that 92% (dependent variable) is explained by the study time (independent variable))

  - *It should be noted, however, that this is just an illustrative example, with a very small set of data that does not allow these results to be taken seriously*
    - instead of 10- and 5-student test and training datasets, respectively, more realistic examples would typically involve hundreds or thousands of instances in both sets

- The Coefficient of Determination is probably the most commonly used measure of performance among the Data Science community in the evaluation of regression models

- These and many other metrics, both regression and classification, are available in scikit-learn's metrics module

```
from sklearn.metrics import r2_score, mean_squared_error
r2_score(notas_test,notas_estimadas) #devolve 0.92
mean_squared_error(notas_test,notas_estimadas,squared=False) #devolve 1.51
```

With squared=True, calculates the MSE instead of the RMSE

Model persistence

- Once the model has been created and trained, it is necessary to preserve it for future
  - *By recording the model, it will be available so that future predictions can be made quickly and without the need for new training*
- We have two ways to serialize the trained model, making it persistent
  - *using python's Pickle module,*
  - *or through Scikit-learn's Joblib module, which optimally records objects with NumPy arrays*
- How to record the model with Pickle

```
import pickle #para gravar em disco
f = open('modelo_horas_estudo.pck','wb')
pickle.dump(modelo,f)
f.close()
```

```
import pickle #para ler do disco
f = open('modelo_horas_estudo.pck','rb')
modelo = pickle.load(f)
f.close()
```

- How to write the template with Joblib

```
import joblib #para gravar em disco
joblib.dump(modelo,'modelo_horas_estudo.jbl')
```

With Joblib it turns out to be simpler

We don't get the ones to open and close the file.

```
import joblib #para ler do disco
modelo = joblib.load('modelo_horas_estudo.jbl')
```

Preprocessing

- In this example that we have just illustrated, we start from a fictitious dataset, therefore containing complete data, without noise and in the appropriate format to be immediately analyzed
  - *But that's not the reality in most real datasets*
- As a rule, even before the application of ML techniques, datasets need to be properly prepared, involving tasks such as
  - *data selection*
  - *Cleaning data*
  - *adaptation and transformation of data*
- This set of tasks, which are intended to prepare and adapt the data for ml, constitute the pre-processing phase
  - *This is a vital step in the whole process of automatic knowledge acquisition, because it represents most of the effort spent in this type of problems (80% of the time, according to some studies), and mainly for the important impact that inevitably ends up having on the quality of the model developed*
- To illustrate the key operations that can be performed during the preprocessing phase, we will use, as an example, the dataset of AI students, but this time, intentionally including a set of imperfections that will have to be corrected

Initial Dataset

- Some imperfections have been introduced in the students dataset to make it more realistic...

```
alunos
```

	nome	genero	freq	idade	presencas	freqAnt	notaIA	aprovado
30000	Tó	M	ordin	20	28.0	False	19.2	1
31234	Ana	F	trab	20	20.0	False	12.2	0
33333	Rui	M	erasm	25	3.0	True	5.3	0
40000	Gil	M	ordin	27	NaN	False	NaN	1
44444	Zé	M	ordin	23	NaN	True	15.9	1
34567	Ivo	M	trab	21	27.0	False	14.0	1
35000	José	M	ordin	21	28.0	False	14.0	1
36000	Joel	M	ordin	22	26.0	True	12.0	1
37000	Bia	F	ordin	65	27.0	True	9.0	0
38000	Luís	M	erasm	20	25.0	False	21.5	0
39000	Rita	F	erasm	21	27.0	False	18.0	1
41000	Lara	F	trab	23	5.0	True	7.0	0
99999	Lara	F	trab	23	5.0	True	7.0	0

Note that if the idea is to build a model that will predict the grades of AI students (grade variable), this model will be regression; but if the goal is to predict whether the student approves or not (approved variable), the model will already be classification.

Cleanning Data

- One of the first tasks that need to be performed in ML is to clean up the data, eliminating or replacing invalid, atypical, or repeated-value data that may exist in the dataset
- *In Panda DataFrames and NumPy arrays, omitted data is typically represented by NaN (Not a Number),*
  - *meaning that the value is absent, undefined, or unrepresentative*
- *Invalid or omitted values, depending on the situation, may be*
  - *deleted by removing their lines,*
  - *or replaced by other specific values that are valid, usually inferred from the remaining values of your column, using location measures (average, median, fashion)*
- *The following are the different cleaning tasks that can be performed on the data*

Replacing NaNs with column mean

- Inspecting the Student DataFrame, being small, we were quickly able to locate the cells with omitted values (NaN)

presencas	freqAnt	notaIA
NaN	False	NaN
NaN	True	15.9

- But when it comes to real datasets, with hundreds or thousands of rows (registered students), the Boolean method isnull() proves to be of great use,
  - *can even be combined with the sum() aggregation method if we want to know quickly how many omitted or null values we have in each of the columns*

```
alunos.isnull().sum()
```

```
nome         0
genero       0
freq         0
idade        0
presencas    2
freqAnt      0
notaIA       1
aprovado     0
```

- One way to deal with NaN (omitted values) is to replace each of them, for example, with the average of the remaining values in the same column
  - *for this, the fillna() method of the respective column is used*
- This may not be the option indicated for the note column, since it is most likely the target variable whose value should in no way be fictionalized
  - *But it may already be a good option for the 'presences' column, especially if it is not an overexplanative variable*

```
alunos.presencas = alunos.presencas.fillna(alunos.presencas.mean())
```

After replacing the NaNs of the 'presences' column

```
alunos
```

	nome	genero	freq	idade	presencas	freqAnt	notaIA	aprovado
30000	Tó	M	ordin	20	28.000000	False	19.2	1
31234	Ana	F	trab	20	20.000000	False	12.2	0
33333	Rui	M	erasm	25	3.000000	True	5.3	0
40000	Gil	M	ordin	27	20.090909	False	NaN	1
44444	Zé	M	ordin	23	20.090909	True	15.9	1
34567	Ivo	M	trab	21	27.000000	False	14.0	1
35000	José	M	ordin	21	28.000000	False	14.0	1
36000	Joel	M	ordin	22	26.000000	True	12.0	1
37000	Bia	F	ordin	65	27.000000	True	9.0	0
38000	Luís	M	erasm	20	25.000000	False	21.5	0
39000	Rita	F	erasm	21	27.000000	False	18.0	1
41000	Lara	F	trab	23	5.000000	True	7.0	0
99999	Lara	F	trab	23	5.000000	True	7.0	0

Elimination of lines with NaNs

- In case the value of the gradeIA dependent variable is missing, the right decision will even remove the entire line (in the example, student Gil removal)

```
alunos.dropna(inplace=True)
```

deletes all lines that contain NaNs

```
alunos
```

	nome	genero	freq	idade	presencas	freqAnt	notaIA	aprovado
30000	Tó	M	ordin	20	28.000000	False	19.2	1
31234	Ana	F	trab	20	20.000000	False	12.2	0
33333	Rui	M	erasm	25	3.000000	True	5.3	0
44444	Zé	M	ordin	23	20.090909	True	15.9	1
34567	Ivo	M	trab	21	27.000000	False	14.0	1
35000	José	M	ordin	21	28.000000	False	14.0	1
36000	Joel	M	ordin	22	26.000000	True	12.0	1
37000	Bia	F	ordin	65	27.000000	True	9.0	0
38000	Luís	M	erasm	20	25.000000	False	21.5	0
39000	Rita	F	erasm	21	27.000000	False	18.0	1
41000	Lara	F	trab	23	5.000000	True	7.0	0
99999	Lara	F	trab	23	5.000000	True	7.0	0

line 40000 has been deleted

There is still a third way to eliminate NaNs, which should be adopted as a last resource: to exclude from the study the variables (columns) where they appear. This option will only make sense when the column has too many NaNs

Delete duplicate lines

- Sometimes actual datasets contain repeated rows that you may want to remove

- To find these duplicate rows, the duplicated() method can be used, which has a keep parameter that can assume one of the following values:
  - *False - all duplicate lines are flagged (classified as True)*
  - *first – only the first duplicate line is not flagged*
  - *last – only the last duplicate line is not flagged*

```
alunos.duplicated(keep=False)
```

```
30000    False
31234    False
33333    False
44444    False
34567    False
35000    False
36000    False
37000    False
38000    False
39000    False
41000     True
99999     True
```

- To select all duplicate rows, simply index the DataFrame with the Boolean series (mask) that was obtained with the previous method

```
alunos[alunos.duplicated(keep=False)]
```

	nome	genero	freq	idade	presencas	freqAnt	notaIA	aprovado
41000	Lara	F	trab	23	5.0	True	7.0	0
99999	Lara	F	trab	23	5.0	True	7.0	0

Delete duplicate lines

- Finally, to eliminate repeated rows, always maintaining the first of them, the method should be used drop_duplicates() with the parameter Keep='first'

```
alunos.drop_duplicates(keep='first', inplace=True)
```

```
alunos
```

	nome	genero	freq	idade	presencas	freqAnt	notaIA	aprovado
30000	Tó	M	ordin	20	28.000000	False	19.2	1
31234	Ana	F	trab	20	20.000000	False	12.2	0
33333	Rui	M	erasm	25	3.000000	True	5.3	0
44444	Zé	M	ordin	23	20.090909	True	15.9	1
34567	Ivo	M	trab	21	27.000000	False	14.0	1
35000	José	M	ordin	21	28.000000	False	14.0	1
36000	Joel	M	ordin	22	26.000000	True	12.0	1
37000	Bia	F	ordin	65	27.000000	True	9.0	0
38000	Luís	M	erasm	20	25.000000	False	21.5	0
39000	Rita	F	erasm	21	27.000000	False	18.0	1
41000	Lara	F	trab	23	5.000000	True	7.0	0

The last Lara disappeared

Delete duplicate lines

- Sometimes we want to eliminate rows that repeat values only in a few specific columns
  - *To eliminate, for example, rows of students who are simultaneously the same age and grade, we only have to indicate these columns using the subset parameter*

```
alunos.drop_duplicates(subset=['idade', 'notaIA'], keep='last')
```

	nome	genero	freq	idade	presencas	freqAnt	notaIA	aprovado
30000	Tó	M	ordin	20	28.000000	False	19.2	1
31234	Ana	F	trab	20	20.000000	False	12.2	0
33333	Rui	M	erasm	25	3.000000	True	5.3	0
44444	Zé	M	ordin	23	20.090909	True	15.9	1
35000	José	M	ordin	21	28.000000	False	14.0	1
36000	Joel	M	ordin	22	26.000000	True	12.0	1
37000	Bia	F	ordin	65	27.000000	True	9.0	0
38000	Luís	M	erasm	20	25.000000	False	21.5	0
39000	Rita	F	erasm	21	27.000000	False	18.0	1
41000	Lara	F	trab	23	5.000000	True	7.0	0

Ivo does not appear, for he was the same age and notes that Joseph (who was the last)

Note that, as it was not used ‘inplace=True’, The Students DataFrame has not changed. It was stuck, with this command, only exemplifying how the lines would be removed

Deletion of rows with invalid values

- We've seen how to deal with omitted values and repeated lines. What if our dataset contains inadmissible values?
  - *We can start by identifying rows with invalid values, using the index() method on the rows that are selected by the condition that verifies that the value is invalid*

```
labels_linhas=alunos[alunos.notaIA>20].index
```

```
Int64Index([38000], dtype='int64')
```

and then eliminate them

```
alunos.drop(labels_linhas, inplace=True)
```

```
alunos
```

	nome	genero	freq	idade	presencas	freqAnt	notaIA	aprovado
30000	Tó	M	ordin	20	28.000000	False	19.2	1
31234	Ana	F	trab	20	20.000000	False	12.2	0
33333	Rui	M	erasm	25	3.000000	True	5.3	0
44444	Zé	M	ordin	23	20.090909	True	15.9	1
34567	Ivo	M	trab	21	27.000000	False	14.0	1
35000	José	M	ordin	21	28.000000	False	14.0	1
36000	Joel	M	ordin	22	26.000000	True	12.0	1
37000	Bia	F	ordin	65	27.000000	True	9.0	0
39000	Rita	F	erasm	21	27.000000	False	18.0	1
41000	Lara	F	trab	23	5.000000	True	7.0	0

line 38000 has been deleted

Elimination of outliers

- An outlier is an atypical value that is quite far away from the other values observed
  - *Whether or not they result from failures or errors, these are outliers that you may usually want to remove from the dataset*
  - *its inclusion may negatively induce the model with incorrect information or related to very particular cases that should not be considered, under penalty of compromising the overall performance of the model*
- Two of the methods that can be used to identify outliers are Z-score and Tukey Fences
  - *The Z-score method accounts for how many standard deviations a given value is distanced from the mean*

$$Z_{score} = (x_i - \bar{x})/\sigma$$

With x_i the value to test, $\bar{x}$ the average and σ the standard deviation

    - If the Z-score result is greater than 3 or infers to -3, the value is considered outlier
  - *The method Tukey Fences uses as a reference measure the distance between the 1º (Q_1) e o 3º (Q_3) quartiles, to sort as outlier all the value that distances itself from these two quartiles once and a half times this distance. That is, it is outliers if*
    - Overcome $Q_3+1.5\times(Q_3-Q_1)$,
    - or if it is less than $Q_1-1.5\times(Q_3-Q_1)$.
- Let's exemplify the two methods described by trying to remove with them the outliers that may exist in the 'age' column

Elimination of outliers by Z-score

- To apply the formula $Z_{score} = (x_i - \bar{x})/\sigma$, let's start by calculating the mean and standard deviation of ages

```
media=alunos.idade.mean()
media
```

```
26.1
```

```
desvpd=alunos.idade.std()
desvpd
```

```
13.755402978870197
```

- The Z-score is then easily calculated

```
(alunos.idade-media)/desvpd
```

```
30000   -0.443462
31234   -0.443462
33333   -0.079969
44444   -0.225366
34567   -0.370763
35000   -0.370763
36000   -0.298065
37000    2.827980
39000   -0.370763
41000   -0.225366
```

- Since none of these values are greater than 3 or less than -3, it is concluded that, according to the Z-score criterion, there are no outliers in the ages
  - *not even Bia's, who is no longer exactly a young woman...*

Elimination of outliers by the *Tukey Fences*

To apply the method Tukey Fences, we start by determining the 1º e 3º quartiles with the function percentile() of the NumPy

```
import numpy as np
q1, q3 = np.percentile(alunos.idade, [25, 75])
```

```
21.0 23.0
```

We calculate their limits,

```
lim_inf=q1-1.5*(q3-q1)
```

```
18.0
```

```
lim_sup=q3+1.5*(q3-q1)
```

```
26.0
```

to easily construct the condition with the outlier criterion that will be used in indexing the respective rows (the rows that contain the outliers)

```
alunos[(alunos.idade<lim_inf)|(alunos.idade>lim_sup)]
```

	nome	genero	freq	idade	presencas	freqAnt	notaIA	aprovado
37000	Bia	F	ordin	65	27.0	True	9.0	0

With this criterion, Bia does not go unnoticed...

Then using the method index() we get to know the labels of these lines

```
labels_linhas=alunos[(alunos.idade<lim_inf)|(alunos.idade>lim_sup)].index
```

```
Int64Index([37000], dtype='int64')
```

that will ultimately be used for their removal through the method drop().

```
alunos.drop(labels_linhas, inplace=True)
```

Column normalization

- It is often necessary to normalize the values of the numeric columns of the dataset
- This normalization aims to pass the values of all columns to the same scale, without, however, losing the differentiation that exists between values of the same column
- This is a crucial technique for the performance of certain ML algorithms, such as artificial neuronal networks and support vector machines (SVm)
  - *Normalization is necessary to avoid unnecessary calculation difficulties and, mainly, to prevent attributes with wide range of values from overimposing those of reduced amplitude*
  - *Otherwise, those who assume higher amplitude values would eventually have a greater preponderance in the training of the model.*
- Min-max and Z-score normalizations are two of the most commonly used techniques
  - *min-max (passes the value to the range from 0 to 1)*

    Applies to each column the transformation $x' = (x - x_{min})/(x_{max} - x_{min})$
  - *Z-score (represents the value by its distance to the average, in number of standard deviations)*

    Applies to each column the transformation $x' = (x - \bar{x})/\sigma$

Column normalization

- To apply normalization, we must start by selecting only the numeric columns

```
colunasnum=alunos.select_dtypes(include='number')
```

```
colunasnum
```

	idade	presencas	notaIA	aprovado
30000	20	28.000000	19.2	1
31234	20	20.000000	12.2	0
33333	25	3.000000	5.3	0
44444	23	20.090909	15.9	1
34567	21	27.000000	14.0	1
35000	21	28.000000	14.0	1
36000	22	26.000000	12.0	1
39000	21	27.000000	18.0	1
41000	23	5.000000	7.0	0

- By instantiating the MinMaxScaler class of the sklearn.preprocessing package, you get the object that will be able to apply the min-max transformation (scaler object)

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

Column normalization

- Using its method fit_transform(), the 'climber' object applies the min-max transformation to the DataFrame of the numerican columns
  - *resulting in an NumPy array (scaled_values) with the values already normalized*

```
scaled_values = scaler.fit_transform(colunasnum)
```

```
array([[0.    , 1.    , 1.    , 1.    ],
       [0.    , 0.68  , 0.4964, 0.    ],
       [1.    , 0.    , 0.    , 0.    ],
       [0.6   , 0.6836, 0.7626, 1.    ],
       [0.2   , 0.96  , 0.6259, 1.    ],
       [0.2   , 1.    , 0.6259, 1.    ],
       [0.4   , 0.92  , 0.482 , 1.    ],
       [0.2   , 0.96  , 0.9137, 1.    ],
       [0.6   , 0.08  , 0.1223, 0.    ]])
```

- By assigning the columns of the initial DataFrame to the number ones, the array with the transformed values, we complete the normalization process.

Note that all numeric values in the dataset are all within the range of 0 to 1.

```
alunos[colunasnum.columns]=scaled_values
```

```
alunos
```

	nome	genero	freq	idade	presencas	freqAnt	notaIA	aprovado
30000	Tó	M	ordin	0.0	1.000000	False	1.000000	1.0
31234	Ana	F	trab	0.0	0.680000	False	0.496403	0.0
33333	Rui	M	erasm	1.0	0.000000	True	0.000000	0.0
44444	Zé	M	ordin	0.6	0.683636	True	0.762590	1.0
34567	Ivo	M	trab	0.2	0.960000	False	0.625899	1.0
35000	José	M	ordin	0.2	1.000000	False	0.625899	1.0
36000	Joel	M	ordin	0.4	0.920000	True	0.482014	1.0
39000	Rita	F	erasm	0.2	0.960000	False	0.913669	1.0
41000	Lara	F	trab	0.6	0.080000	True	0.122302	0.0

Column normalization

- Wanting to apply Z-score normalization, what would have to change was only the 'climber' object, which would become an instance of the StandardScaler class

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
scaled_values = scaler.fit_transform(colunasnum)
```

```
scaled_values
```

```
array([[-1.1487,  0.8136,  1.4082,  0.7071],
       [-1.1487, -0.049 , -0.199 , -1.4142],
       [ 2.0821, -1.882 , -1.7832, -1.4142],
       [ 0.7898, -0.0392,  0.6505,  0.7071],
       [-0.5026,  0.7057,  0.2143,  0.7071],
       [-0.5026,  0.8136,  0.2143,  0.7071],
       [ 0.1436,  0.5979, -0.2449,  0.7071],
       [-0.5026,  0.7057,  1.1326,  0.7071],
       [ 0.7898, -1.6663, -1.3928, -1.4142]])
```

```
alunos[colunasnum.columns]=scaled_values
```

We are thus left with an overview of the entire process of developing an ML model solve **exercise #16** and **#17**

from the book of exercises