IFT 3700
Automne 2022
Devoir 2

Auteurs: Al Wahid Bio-Tchane (20137307)
&
Hanz Schepens (20189679)

Table des matières

I. Collecte de données

Dans un premier temps, nous commençons par le **Data Sourcing**, dans cette section nous allons collecter des données depuis 40 tableaux sur Wikipédia. Ensuite, nous allons traiter ces données, les nettoyer et les gérer dans une base de données où il nous faudra gérer les valeurs manquantes, relever des statistiques essentielles (les valeurs moyennes, les valeurs médianes, …) pour pouvoir finalement effectuer des prédictions, des tests ainsi que des visualisations.

a. Collecte des 40 colonnes

A partir des 40 liens présents sur le fichier des consignes, nous avons relevé les colonnes dont on aurait besoin pour effectuer notre analyse.
Les colonnes ont été dans un premier temps stocké arbitrairement dans un fichier Excel pour pouvoir travailler dessus ultérieurement.
Une collecte arbitraire nous a permis d'avoir un Dataset mal présenté comme est affiché ci-dessous :

	1	1.1	Unnamed: 2	2	2.1	Unnamed: 5	3	3.1	Unnamed: 8	4	...	40.1	Unnamed: 114	39	39.1	Unnamed: 117	6	6.1	Unnamed: 120	37	37
0	Liechtenstein*	180227.0	NaN	Albania	45.25	NaN	Afghanistan	0.2	NaN	Afghanistan*	...	28.29	NaN	Ireland	3885.0	NaN	Switzerland	0.962	NaN	China	527
1	Monaco*	173696.0	NaN	Andorra	191.23	NaN	Albania	7.5	NaN	Albania*	...	28.25	NaN	United States	3782.0	NaN	Norway	0.961	NaN	United States	1875
2	Luxembourg*	117182.0	NaN	Argentina	51.51	NaN	Algeria	0.9	NaN	Algeria	...	28.20	NaN	Belgium	3769.0	NaN	Iceland	0.959	NaN	United Kingdom	2959
3	Bermuda*	123945.0	NaN	Australia	77.88	NaN	Andorra	11.3	NaN	Andorra	...	28.00	NaN	Turkey	3711.0	NaN	Hong Kong	0.952	NaN	India	138
4	Ireland*	86251.0	NaN	Austria	117.31	NaN	Angola	6.4	NaN	Angola	...	28.00	NaN	Austria	3695.0	NaN	Australia	0.951	NaN	Germany	2097
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
407	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
408	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
409	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
410	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
411	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

412 rows × 123 columns

Il nous faut ainsi nettoyer et présenter dans une meilleure qualité ce Dataset, pour cela, nous avions effectué un certain nettoyage de données par le biais des commandes d'Excel et d'autres par des lignes de codes sous le package pandas de python, dans ce sens nous nous sommes servis du dictionnaire Anglais des pays du monde (countries_for_language) pour assurer une unique orthographe des noms des pays et les présenter dans un meilleur format.

Pour veiller à ce que les entrées du dataset soit les pays, nous les mettions en index de la base de données (chaque pays aura un index qui le représente sur le Dataset) et les données des 40 colonnes en colonnes (les features du dataset). Ceci nous permettra d'assurer que les informations d'un pays soient toutes présentes dans un format où le pays représente une entrée du Dataset, et les informations auxquelles il est associé soient toutes représentées comme des features dans le dataset, également pour éviter les redondances qui peuvent générer des valeurs manquantes.

b. Quelques informations générales sur les colonnes et traitement des valeurs manquantes

Dans un premier temps, nous avons supprimé les entrées avec plus de 12 valeurs manquantes :

```
# print the sum of nan values in each row
df.isnull().sum(axis=1).values
```

```
array([ 3,  2,  1,  3,  4,  7,  3,  8,  1,  5,  4,  2,  7,  3,  4,  4,  3,
        5,  3,  7,  8,  5,  3,  8,  6,  4,  5,  5,  6,  4,  3, 10,  6,  9,
        9,  6,  9,  8, 14,  8,  7, 12,  7,  6,  8,  7,  8,  9, 10,  9,  7,
        8,  4,  7,  8,  7,  6,  5,  7,  6, 10,  7,  8,  8,  4,  7,  7,  7,
       13,  9,  6,  8,  6,  7,  7,  7,  6, 11, 10,  8,  7, 11,  7,  8,  8,
        9, 10, 12,  9,  7,  9, 12,  9,  6, 13,  7,  9, 11, 10,  8,  9, 11,
       14, 11,  9, 11, 10, 11,  9, 10, 13, 11, 11,  9, 12,  8,  9, 12, 12,
       10, 12, 10, 11, 11, 13, 11, 11, 15, 16, 15, 13, 14, 14, 11, 15, 13,
       14, 14, 10, 11, 14, 15, 11, 13, 12, 13, 13, 12, 12, 14, 12, 15, 16,
       18, 14, 18, 18, 19, 18, 19, 17, 19, 16, 19, 17, 20, 20, 22, 20, 23,
       21, 27, 22, 28,  5,  6,  4, 10,  7, 12,  7, 11,  8,  8,  7, 14, 13,
       30, 31, 29, 31, 32, 32, 32, 35, 35, 32, 36, 36, 32, 35, 35, 33, 34,
       35, 34, 33, 34, 34, 33, 34, 33, 34, 35, 11, 12,  8,  6, 18, 33, 34,
       36, 35, 36, 35, 37, 37, 37, 38, 37, 34, 39, 39, 39, 39, 39, 39, 39,
       39, 39, 39, 39, 39, 39, 39, 39, 38, 38, 38, 39, 39, 38, 38, 38, 38,
       39, 39, 39, 38, 39, 39, 39, 39, 38, 39, 39, 39, 39, 39, 38, 38, 39,
       39, 39, 38, 39, 39, 39, 39, 38, 39, 39, 37, 38, 39, 39, 39, 39, 39,
       39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39,
       39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39,
       39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39, 39,
       39, 39, 39, 39, 39, 39], dtype=int64)
```

```
#drop the rows with more than 12 missing values
df=df[df.isnull().sum(axis=1)<=12]
```

Ensuite, Nous avons remplacé les valeurs manquantes par la médiane, pour finalement pouvoir générer un tableau décrivant notre Data en présentant le mode, la médiane, le min, le max, la moyenne et la variance de chaque colonne :

```
df1.describe()
```

Python

	40.1	1.1	2.1	3.1	4.1	5.1	6.1	7.1	8.1	9.1	...	30.1	31.1	32.1	33.1	34.1	35.1	36.1	37.1	38.1	
count	144.000000	144.000000	144.000000	144.000000	144.000000	144.000000	144.000000	144.000000	144.000000	144.000000	...	144.000000	144.000000	144.000000	144.000000	144.000000	144.000000	144.000000	144.000000	144.000000	144.00(
mean	22.163125	23882.541667	98.830000	6.161111	5.846875	2.090972	0.784035	5.566389	40.581944	0.695000	...	44.296528	158.840278	63.320139	24.099306	5.056944	174.981597	136.703403	1020.854167	16383.784722	2985.847
std	5.403630	28741.942011	45.148545	3.944308	8.495756	0.511738	0.103270	2.244361	6.875960	0.256637	...	32.559325	100.919109	25.784932	3.914175	1.678730	446.026481	83.370742	1353.893954	40539.900395	393.472
min	7.500000	1466.000000	36.020000	0.000000	0.000000	1.100000	0.512000	1.080000	11.000000	0.140000	...	3.000000	16.000000	7.000000	15.500000	2.500000	0.000000	9.080000	9.000000	7.000000	1938.00(
25%	20.137500	7087.500000	68.427500	2.700000	1.300000	2.050000	0.725250	3.585000	41.000000	0.497500	...	20.200000	91.250000	45.350000	21.275000	4.000000	23.625000	75.540000	94.250000	403.250000	2736.00(
50%	24.400000	13523.000000	90.930000	6.250000	2.500000	2.050000	0.790000	5.945000	41.000000	0.785000	...	36.350000	139.000000	66.600000	23.700000	4.800000	32.900000	121.065000	376.000000	2591.000000	2997.00(
75%	25.662500	28195.500000	109.555000	9.225000	6.125000	2.050000	0.849250	7.182500	41.000000	0.912500	...	60.575000	211.250000	83.625000	26.425000	5.500000	36.000000	184.402500	1587.250000	9376.000000	3308.75(
max	28.290000	180227.000000	245.500000	15.200000	44.700000	6.600000	0.962000	9.810000	69.000000	0.980000	...	145.900000	572.000000	100.000000	33.200000	12.900000	3753.000000	430.760000	6050.000000	275232.000000	3885.00(

8 rows × 40 columns

c. Régressions linéaires

Les valeurs manquantes ont été remplacées d'abord par la valeur médiane de la feature variable, puis nous essayons de les remplacer en utilisant une régression linéaire. Pour ce faire nous utilisons le KNN Imputer, par le biais de ces lignes de code :

```
from sklearn.impute import KNNImputer

#use the KNNImputer to fill the missing values
for i in df.columns:
    #use the 20 closest values to apply the linear regression
    imputer = KNNImputer(n_neighbors=20)
    df[i] = imputer.fit_transform(df[i].values.reshape(-1, 1))
```

d. Binarisation des valeurs du dataset

Afin d'avoir une idée sur la variabilité des colonnes de la data nous effectuons une binarisation des colonnes, il s'agirait

de créer une autre base de données (qui ressemblera à une map de 0 et 1) et qui sera concaténée par la suite à notre base de données. Dans chaque valeur de ce dataset, on affecte la valeur 1 si la valeur du dataset qui lui correspond est supérieure à la médiane de la variable feature, et la valeur 0 si c'est le contraire.

On arrive à avoir ce dataset :

	40.1	1.1	2.1	3.1	4.1	5.1	6.1	7.1	8.1	9.1	...	30.1_binary_binary	31.1_binary_binary	32.1_binary_binary	33.1_binary_binary	34.1_binary_binary	35.1_binary_binary	36.1_binary_binary	37.1_binary_binary
0	28.290000	180227.000000	45.250000	0.20000	6.700000	3.200	0.962000	9.24	48.0	0.14	...	0	0	0	1	1	1	1	1
1	28.250000	173696.000000	191.230000	7.50000	2.100000	1.700	0.961000	7.92	39.0	0.15	...	0	1	1	0	1	0	1	1
2	28.200000	117182.000000	51.510000	0.90000	1.300000	2.400	0.959000	8.16	43.0	0.16	...	0	0	0	0	1	0	1	1
3	28.000000	123945.000000	77.880000	11.30000	2.600000	2.100	0.782261	7.51	16.0	0.16	...	0	1	1	1	1	1	1	0
4	28.000000	86251.000000	117.310000	6.40000	4.800000	3.100	0.951000	7.56	59.0	0.16	...	0	0	0	0	1	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
104	21.903023	8538.000000	104.005862	2.70000	1.800000	2.345	0.717000	2.49	39.6	0.93	...	1	0	1	1	0	0	0	0
25	21.903023	25851.710744	37.620000	12.70000	15.200000	2.345	0.911000	2.68	50.0	0.34	...	0	1	0	0	1	0	1	0
38	21.903023	25851.710744	55.720000	6.15303	6.265234	2.345	0.863000	5.78	41.0	0.50	...	0	0	1	0	1	0	1	0
71	21.903023	25851.710744	56.890000	6.15303	20.000000	2.345	0.785000	6.82	39.6	0.80	...	0	1	0	0	0	1	0	0
78	21.903023	25851.710744	104.005862	0.80000	3.000000	2.345	0.768000	5.71	39.6	0.85	...	0	0	1	0	0	0	0	0

144 rows × 120 columns

II. Corrélations

a. Coefficient de corrélation entre chaque paire de colonnes

Par le biais de ces lignes de code, on calcule le coefficient de corrélation entre chaque deux colonnes dans notre Dataset :

```
#print the correlation between the columns
correlation = resulted_data.corr()
print(correlation)
```

```
Output exceeds the size limit. Open the full output data in a text editor
                        40.1       1.1       2.1       3.1       4.1  \
40.1                1.000000  0.548746 -0.022395 -0.042442  0.099882
1.1                 0.548746  1.000000 -0.104369  0.027210  0.012966
2.1                -0.022395 -0.104369  1.000000 -0.065594 -0.034444
3.1                -0.042442  0.027210 -0.065594  1.000000  0.124213
4.1                 0.099882  0.012966 -0.034444  0.124213  1.000000
...                      ...       ...       ...       ...       ...
35.1_binary_binary  0.274756  0.309080 -0.083311  0.011741  0.156975
36.1_binary_binary  0.531918  0.526751 -0.094120  0.024757  0.019955
37.1_binary_binary  0.346243  0.321218 -0.009592  0.013428  0.131564
38.1_binary_binary -0.026195  0.418107 -0.143954  0.119602 -0.068956
39.1_binary_binary  0.651439  0.577266 -0.174676  0.097828  0.070202
```

b. La plus forte corrélation …

En considérant toujours la valeur absolue, on cherche maintenant à déterminer quelle colonne est corrélée le plus avec laquelle du reste des colonnes du dataset.
On arrive à effectuer ceci par nous servir de ces lignes de code affichées ci-contre :

```
# find the most correlated feature for each variable
list_corr_max={}
for i in correlation.columns:
    correlation[i]=abs(correlation[i][correlation[i]!=1])
    list_corr_max[i]=correlation[i].idxmax()

list_corr_max
```

```
Output exceeds the size limit. Open the full output data in a text editor
{'40.1': '29.1',
 '1.1': '23.1',
 '2.1': '2.1_binary',
 '3.1': '3.1_binary',
 '4.1': '4.1_binary',
 '5.1': '5.1_binary',
 '6.1': '29.1',
 '7.1': '7.1_binary',
 '8.1': '8.1_binary',
 '9.1': '36.1',
 '10.1': '10.1_binary',
 '11.1': '11.1_binary',
 '12.1': '12.1_binary',
 '13.1': '38.1',
 '14.1': '14.1_binary',
 '15.1': '15.1_binary',
 '16.1': '16.1_binary',
 '17.1': '17.1_binary',
 '18.1': '18.1_binary',
 '19.1': '19.1_binary',
 '20.1': '20.1_binary',
 '21.1': '36.1',
 '22.1': '34.1',
 '23.1': '36.1',
 '24.1': '24.1_binary',
```

Par exemple, La colonne « 1.1 » est fortement corrélée avec la colonne « 29.1 »

c. Ordre de colonnes selon Corrélation

Si on exécute ces lignes de code :

```
list_corr={}
for i in correlation.columns:
    #if i =='40.1':
    correlation[i]=abs(correlation[i][correlation[i]!=1])
    #sort the correlated features
    list_corr[i]=correlation[i].sort_values(ascending=False).index

list_corr
```

On arrive à déterminer lesquelles du reste des colonnes du dataset qui contribuent les plus à prédire le sens de variabilité d'une colonne donnée :

Pour mieux comprendre, on prend l'exemple du premier output de cette cellule :

```
{'40.1': Index(['29.1', '39.1', '6.1', '23.1', '36.1', '40.1_binary',
       '40.1_binary_binary', '21.1', '34.1', '22.1',
       ...
       '2.1', '8.1', '32.1', '35.1', '18.1', '26.1_binary',
       '26.1_binary_binary', '28.1', '5.1', '40.1'],
```

La colonne « 40.1 » est liée très fortement aux colonnes « 29.1 », « 39.1 », « 6.1 » …. Plus on évolue dans la liste des colonnes qui suivent plus la corrélation devient faible.

III. Prédictions avec classifieur bayésien et régression linéaire

a. Analyse de la prédiction en fonction des autres colonnes

Dans un premier lien nous avons divisé notre dataset en deux parties, une pour la régression et l'autre pour la classification.

```
#devide the data into two part, the first one for regression, the second ine for naive_bayes
reg_data=resulted_data[df.columns]
cls_data=resulted_data.drop(columns=df.columns)
```

Le principe du calcul de l'importance des variables consiste à considérer chaque fois une colonne comme variable dépendante puis prendre une partie pour faire le test afin de trouver vers la fin la contribution de chaque variable indépendante dans la prédiction.
L'importance de ces variables indépendante sera ordonnée pour mieux identifier ceux les plus important.

```
{'40.1': {'mse_test': 6.33087612529354,
  'mse_train': 1.8034915653904182,
  'coef': {'9.1': 7.283616896697859,
   '6.1': 6.280544568509785,
   '24.1': 1.3040468394154563,
   '29.1': 0.7542890328125333,
   '12.1': 0.7021335877359896,
   '34.1': 0.3611043225264225,
   '5.1': 0.2631940853145729,
```

D'après cet affichage, il s'avère que l'importance diffère énormément d'une variable à l'autre, ceci se justifie par leur causalité et leur corrélation déjà existante, et le fait qu'elles sont déjà corrélées (des caractéristiques proches), voir elles appartiennent au même secteur, généralement les variables à partir de la quatrième peuvent être négligées.

b. Identification de la colonne prédisant le mieux

D'après ce dictionnaire que nous avons créé, le choix de deux est déjà judicieux vu les grandes différences entre les variables. Pour notre cas la détermination de ces deux variables pour chaque colonne était facile, via le code suivant :

```
paire[i]=[abs(coefs).sort_values(ascending=False,by='Coefficients').index[0],abs(coefs).sort_values(ascending=False,by='Coefficients').index[1]]
```

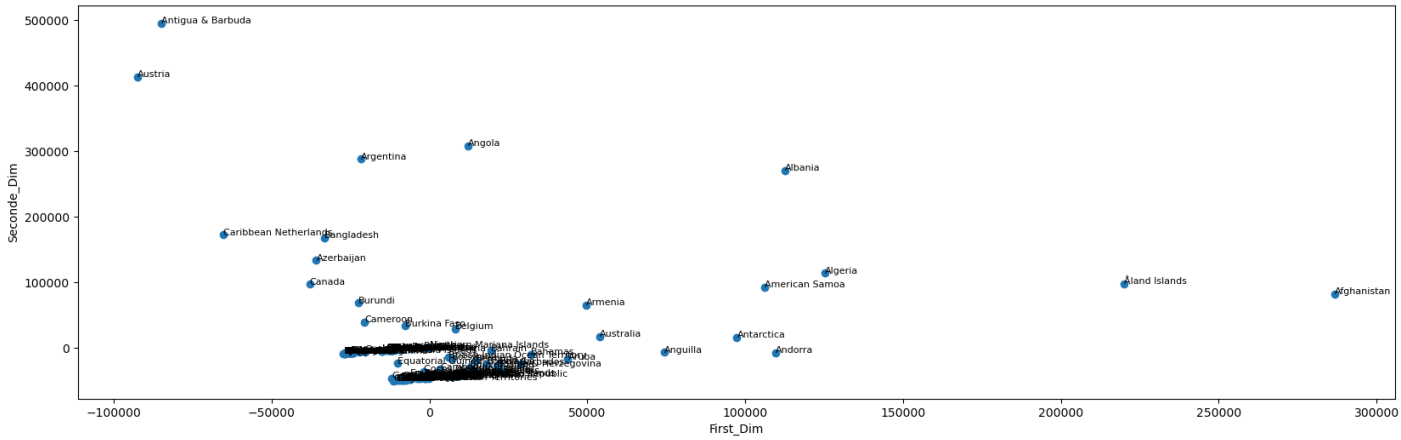
Et le résultat est :

```
{'40.1': ['9.1', '6.1'],
 '1.1': ['6.1', '24.1'],
 '2.1': ['6.1', '21.1'],
 '3.1': ['6.1', '12.1'],
 '4.1': ['6.1', '24.1'],
 '5.1': ['6.1', '24.1'],
 '6.1': ['9.1', '24.1'],
 '7.1': ['6.1', '9.1'],
 '8.1': ['9.1', '24.1'],
 '9.1': ['6.1', '12.1'],
 '10.1': ['6.1', '9.1'],
```

IV. Visualisation et représentations

a. Visualisation des 40 pays en 2 Dimensions

Le but de cette étude était de regrouper les pays ayant les mêmes caractéristiques sur plusieurs volets, après la réduction en 2D, nous avons vraiment pu voir quels sont les pays les plus différents et ceux qui sont les plus similaires.



Nous nous sommes servis de la décomposition PCA, dont on a pu réduire les 40 dimensions en juste 2.

b. Visualisation en 5 Dimensions

Les valeurs affichées sont le résultat de plusieurs valeurs en entrée. PCA est une méthode qui extrait les caractéristiques communes d'un espace de grande dimension à un espace de faible dimension. Lorsque le nombre de dimensions diminue, PCA essaye de garder le maximum d'information possible de l'ensemble de données d'origine.
C'est l'approche que nous avons suivie, en réduisant les 40 dimensions que génèrent les 40 pays en 5.