==========

Université Paris 1 Panthéon-Sorbonne

Devoir d'Apprentissage Statistique Avancé

Présenté par :

Berthony Sully
Agui Tchabou
Landy Clément
Jerrold Nemba

Au Professeur : **Alain Célisse**

1er Avril 2022

==========

Partie 1

1. Prédire la survie des passagers en fonction des autres variables qualitatives descriptives de ces passagers. Pour cela, vous disposer du fichier "train" pour apprendre la règle de classification, puis du fichier "test" pour évaluer sa performance de prédiction.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
# loading data
train_data = pd.read_csv(r'https://raw.githubusercontent.com/htsull/Adv-Stat-Learn-HW/main/data/train-data.csv')
test_data = pd.read_csv(r'https://raw.githubusercontent.com/htsull/Adv-Stat-Learn-HW/main/data/test-data.csv')
print('train_data.shape:', train_data.shape)
print('test_data.shape:', test_data.shape)
```

```
train_data.shape: (712, 25)
test_data.shape: (179, 25)
```

In [ ]:
```
# data concatenation
data = pd.concat([train_data, test_data], axis=0)
# data shape
print('Full data shape: ', data.shape)
```

```
Full data shape:  (891, 25)
```

In [ ]:
```
# explore data
print('\nData info:')
print(data.info())
```

```
Data info:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 0 to 178
Data columns (total 25 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Survived    891 non-null    int64
 1   Age         891 non-null    float64
 2   Fare        891 non-null    float64
 3   Pclass_1    891 non-null    int64
 4   Pclass_2    891 non-null    int64
 5   Pclass_3    891 non-null    int64
 6   Sex_female  891 non-null    int64
 7   Sex_male    891 non-null    int64
 8   SibSp_0     891 non-null    int64
 9   SibSp_1     891 non-null    int64
 10  SibSp_2     891 non-null    int64
 11  SibSp_3     891 non-null    int64
 12  SibSp_4     891 non-null    int64
 13  SibSp_5     891 non-null    int64
 14  SibSp_8     891 non-null    int64
 15  Parch_0     891 non-null    int64
 16  Parch_1     891 non-null    int64
 17  Parch_2     891 non-null    int64
 18  Parch_3     891 non-null    int64
 19  Parch_4     891 non-null    int64
 20  Parch_5     891 non-null    int64
 21  Parch_6     891 non-null    int64
 22  Embarked_C  891 non-null    int64
 23  Embarked_Q  891 non-null    int64
 24  Embarked_S  891 non-null    int64
dtypes: float64(2), int64(23)
memory usage: 181.0 KB
None
```

In [ ]:

```
data.columns
```

Out[ ]:

```
Index(['Survived', 'Age', 'Fare', 'Pclass_1', 'Pclass_2', 'Pclass_3',
       'Sex_female', 'Sex_male', 'SibSp_0', 'SibSp_1', 'SibSp_2', 'SibSp_3',
       'SibSp_4', 'SibSp_5', 'SibSp_8', 'Parch_0', 'Parch_1', 'Parch_2',
       'Parch_3', 'Parch_4', 'Parch_5', 'Parch_6', 'Embarked_C', 'Embarked_Q',
       'Embarked_S'],
      dtype='object')
```

In [ ]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 0 to 178
Data columns (total 25 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Survived    891 non-null    int64
 1   Age         891 non-null    float64
 2   Fare        891 non-null    float64
 3   Pclass_1    891 non-null    int64
 4   Pclass_2    891 non-null    int64
 5   Pclass_3    891 non-null    int64
 6   Sex_female  891 non-null    int64
 7   Sex_male    891 non-null    int64
 8   SibSp_0     891 non-null    int64
 9   SibSp_1     891 non-null    int64
 10  SibSp_2     891 non-null    int64
 11  SibSp_3     891 non-null    int64
 12  SibSp_4     891 non-null    int64
 13  SibSp_5     891 non-null    int64
 14  SibSp_8     891 non-null    int64
 15  Parch_0     891 non-null    int64
```

```
 16  Parch_1     891 non-null    int64
 17  Parch_2     891 non-null    int64
 18  Parch_3     891 non-null    int64
 19  Parch_4     891 non-null    int64
 20  Parch_5     891 non-null    int64
 21  Parch_6     891 non-null    int64
 22  Embarked_C  891 non-null    int64
 23  Embarked_Q  891 non-null    int64
 24  Embarked_S  891 non-null    int64
dtypes: float64(2), int64(23)
memory usage: 181.0 KB
```

In [ ]:
```
print('\nData description:')
data.describe().T
```

Data description:

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
Survived	891.0	0.383838	0.486592	0.00	0.0000	0.000000	1.0	1.0000
Age	891.0	29.699118	13.002015	0.42	22.0000	29.699118	35.0	80.0000
Fare	891.0	32.204208	49.693429	0.00	7.9104	14.454200	31.0	512.3292
Pclass_1	891.0	0.242424	0.428790	0.00	0.0000	0.000000	0.0	1.0000
Pclass_2	891.0	0.206510	0.405028	0.00	0.0000	0.000000	0.0	1.0000
Pclass_3	891.0	0.551066	0.497665	0.00	0.0000	1.000000	1.0	1.0000
Sex_female	891.0	0.352413	0.477990	0.00	0.0000	0.000000	1.0	1.0000
Sex_male	891.0	0.647587	0.477990	0.00	0.0000	1.000000	1.0	1.0000
SibSp_0	891.0	0.682379	0.465813	0.00	0.0000	1.000000	1.0	1.0000
SibSp_1	891.0	0.234568	0.423966	0.00	0.0000	0.000000	0.0	1.0000
SibSp_2	891.0	0.031425	0.174562	0.00	0.0000	0.000000	0.0	1.0000
SibSp_3	891.0	0.017957	0.132871	0.00	0.0000	0.000000	0.0	1.0000
SibSp_4	891.0	0.020202	0.140770	0.00	0.0000	0.000000	0.0	1.0000
SibSp_5	891.0	0.005612	0.074743	0.00	0.0000	0.000000	0.0	1.0000
SibSp_8	891.0	0.007856	0.088337	0.00	0.0000	0.000000	0.0	1.0000
Parch_0	891.0	0.760943	0.426747	0.00	1.0000	1.000000	1.0	1.0000
Parch_1	891.0	0.132435	0.339154	0.00	0.0000	0.000000	0.0	1.0000
Parch_2	891.0	0.089787	0.286037	0.00	0.0000	0.000000	0.0	1.0000
Parch_3	891.0	0.005612	0.074743	0.00	0.0000	0.000000	0.0	1.0000
Parch_4	891.0	0.004489	0.066890	0.00	0.0000	0.000000	0.0	1.0000
Parch_5	891.0	0.005612	0.074743	0.00	0.0000	0.000000	0.0	1.0000
Parch_6	891.0	0.001122	0.033501	0.00	0.0000	0.000000	0.0	1.0000
Embarked_C	891.0	0.188552	0.391372	0.00	0.0000	0.000000	0.0	1.0000
Embarked_Q	891.0	0.086420	0.281141	0.00	0.0000	0.000000	0.0	1.0000
Embarked_S	891.0	0.725028	0.446751	0.00	0.0000	1.000000	1.0	1.0000

In [ ]:
```
# drop columns age and fare in train and test data
```

```
train_data = train_data.drop(['Age', 'Fare'], axis=1)
test_data = test_data.drop(['Age', 'Fare'], axis=1)
```

In [ ]:

```
# import gradient boosting classifier
from sklearn.ensemble import GradientBoostingClassifier

X_train = train_data.drop(['Survived'], axis=1)
X_test = test_data.drop(['Survived'], axis=1)
# define target variable
target = 'Survived'
```

In [ ]:

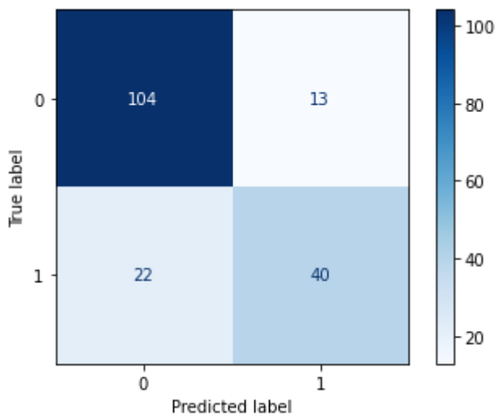
```
xgb = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=0)
model_xgb = xgb.fit(X_train, train_data[target])
score_train = model_xgb.score(X_train, train_data[target])
score_test = model_xgb.score(X_test, test_data[target])
print('Score on train data :{:.2f}'.format(score_train))
print('Score on test data :{:.2f}'.format(score_test))
```

```
Score on train data :0.82
Score on test data :0.80
```

In [ ]:

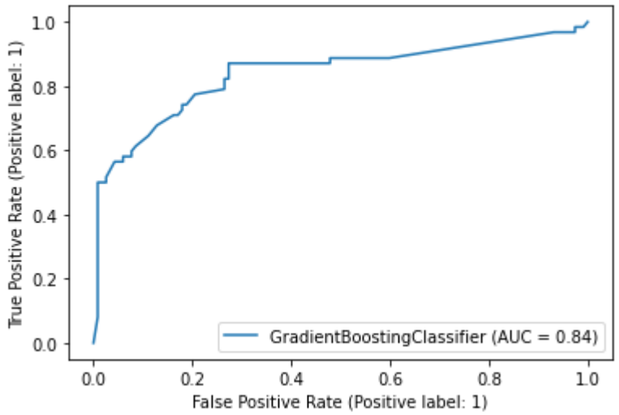
```
# import plot confusion matrix and classification report
from sklearn.metrics import plot_confusion_matrix, classification_report
print('Classification report:')
print(classification_report(test_data[target], model_xgb.predict(X_test)))
print('Confusion matrix:')
plot_confusion_matrix(model_xgb, X_test, test_data[target], cmap=plt.cm.Blues);
```

```
Classification report:
              precision    recall  f1-score   support

           0       0.83      0.89      0.86       117
           1       0.75      0.65      0.70        62

    accuracy                           0.80       179
   macro avg       0.79      0.77      0.78       179
weighted avg       0.80      0.80      0.80       179

Confusion matrix:
```



In [ ]:

```
#import plot roc curve
from sklearn.metrics import plot_roc_curve
plot_roc_curve(model_xgb, X_test, test_data[target]);
```



Partie 2

1. Réaliser une réduction de dimension non-supervisée (de type ACM) à partir des données qualitatives en concaténant les deux fichiers fournis (train et test).

In [ ]:
```
from fanalysis.mca import MCA
```

In [ ]:
```
# On supprime les variables quantitatives
col_to_delete = ["Age", "Fare", "Survived"]

data2 = data.copy()

data.drop(columns=col_to_delete, inplace=True)

data.head()
```

Out[ ]:

	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	SibSp_0	SibSp_1	SibSp_2	SibSp_3	SibSp_4	...	Parch_0	Parch_1	Parch_2	Parch_3	Parch_4	Parch_5	Parch_6	Embarked_C	Embarked_Q	Embarked_S
0	0	0	1	0	1	1	0	0	0	0	...	1	0	0	0	0	0	0	1	0	0
1	0	1	0	1	0	1	0	0	0	0	...	1	0	0	0	0	0	0	0	0	1
2	0	0	1	1	0	0	1	0	0	0	...	1	0	0	0	0	0	0	0	0	1
3	1	0	0	0	1	1	0	0	0	0	...	1	0	0	0	0	0	0	0	0	1
4	0	0	1	0	1	1	0	0	0	0	...	1	0	0	0	0	0	0	0	0	1

5 rows × 22 columns

In [ ]:
```
data = data.astype(int)
data.head()
```

Out[ ]:

	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	SibSp_0	SibSp_1	SibSp_2	SibSp_3	SibSp_4	...	Parch_0	Parch_1	Parch_2	Parch_3	Parch_4	Parch_5	Parch_6	Embarked_C	Embarked_Q	Embarked_S
0	0	0	1	0	1	1	0	0	0	0	...	1	0	0	0	0	0	0	1	0	0
1	0	1	0	1	0	1	0	0	0	0	...	1	0	0	0	0	0	0	0	0	1
2	0	0	1	1	0	0	1	0	0	0	...	1	0	0	0	0	0	0	0	0	1
3	1	0	0	0	1	1	0	0	0	0	...	1	0	0	0	0	0	0	0	0	1

	Pclass_1	Pclass_2	Pclass_3	Sex_female	Sex_male	SibSp_0	SibSp_1	SibSp_2	SibSp_3	SibSp_4	...	Parch_0	Parch_1	Parch_2	Parch_3	Parch_4	Parch_5	Parch_6	Embarked_C	Embarked_Q	Embarked_S
4	0	0	1	0	1	1	0	0	0	0	...	1	0	0	0	0	0	0	0	0	1

5 rows × 22 columns

Notre analyse va porter sur les variables catégorielles suivantes :

In [ ]:
```
var_labels=data.columns
var_labels
```

Out[ ]:
```
Index(['Pclass_1', 'Pclass_2', 'Pclass_3', 'Sex_female', 'Sex_male', 'SibSp_0',
       'SibSp_1', 'SibSp_2', 'SibSp_3', 'SibSp_4', 'SibSp_5', 'SibSp_8',
       'Parch_0', 'Parch_1', 'Parch_2', 'Parch_3', 'Parch_4', 'Parch_5',
       'Parch_6', 'Embarked_C', 'Embarked_Q', 'Embarked_S'],
      dtype='object')
```

Instanciation de la classe MCA en lui passant des étiquettes pour les variables :

In [ ]:
```
acm = MCA(row_labels=data.index, var_labels=data.columns)
```

Estimons le modèle en appliquant la méthode fit de la classe MCA sur le jeu de données:

In [ ]:
```
acm.fit(data.values)
```

Out[ ]:
```
MCA(row_labels=Int64Index([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
            ...
            169, 170, 171, 172, 173, 174, 175, 176, 177, 178],
           dtype='int64', length=891),
    var_labels=Index(['Pclass_1', 'Pclass_2', 'Pclass_3', 'Sex_female', 'Sex_male', 'SibSp_0',
       'SibSp_1', 'SibSp_2', 'SibSp_3', 'SibSp_4', 'SibSp_5', 'SibSp_8',
       'Parch_0', 'Parch_1', 'Parch_2', 'Parch_3', 'Parch_4', 'Parch_5',
       'Parch_6', 'Embarked_C', 'Embarked_Q', 'Embarked_S'],
      dtype='object'))
```

Analyse des valeurs propres

L'attribut acm.eig_ contient successivement:

- en 1ère colonne : les valeurs propres en valeur absolue
- en 2ème colonne : les valeurs propres en pourcentage de la variance totale
- en 3ème colonne : les valeurs propres en pourcentage cumulé de la variance totale

Nombre de facteurs

In [ ]:
```
#récupération des infos - nombre de variables
p = 5
#nombre d'observations
n = data.shape[0]

#nombre total de modalités
M = data.shape[1]
```

In [ ]:
```
#nombre max de facteurs
Hmax = M-p
print("le nombre maximum de facteurs est:", Hmax)
```

```
le nombre maximum de facteurs est: 17
```

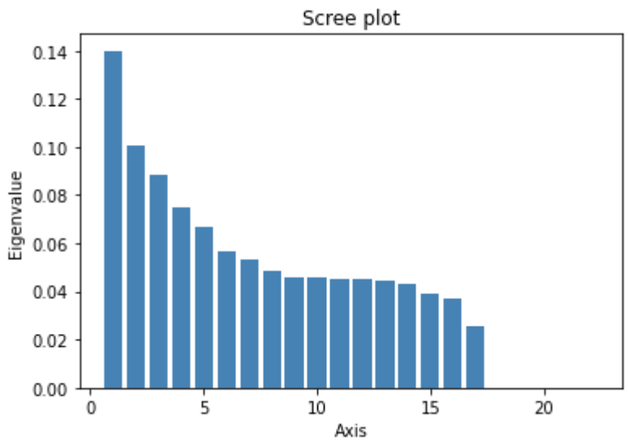
Affichons les valeurs propres ainsi que les pourcentages d'inertie associées aux facteurs, individuelles et cumulées:

In [ ]:
```
#valeurs propres
print(pd.DataFrame(np.transpose(acm.eig_),columns=['Val. Abs','Val. %','Cumul %']))
```

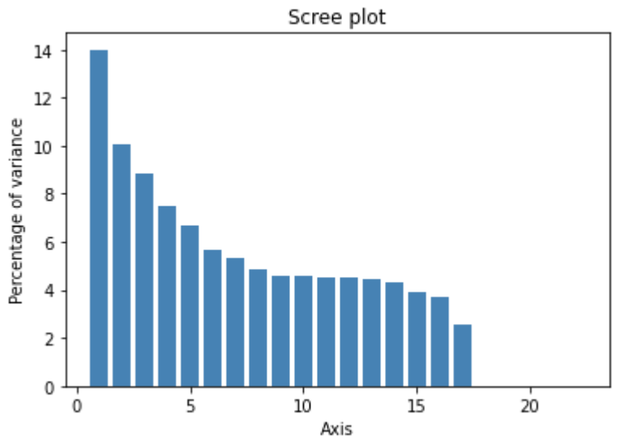
```
        Val. Abs        Val. %     Cumul %
0   1.399841e-01  1.399841e+01   13.998405
1   1.004588e-01  1.004588e+01   24.044282
2   8.826307e-02  8.826307e+00   32.870589
3   7.468726e-02  7.468726e+00   40.339315
4   6.702596e-02  6.702596e+00   47.041910
5   5.669152e-02  5.669152e+00   52.711062
6   5.341955e-02  5.341955e+00   58.053017
7   4.874291e-02  4.874291e+00   62.927308
8   4.582061e-02  4.582061e+00   67.509369
9   4.576222e-02  4.576222e+00   72.085591
10  4.542077e-02  4.542077e+00   76.627669
11  4.518858e-02  4.518858e+00   81.146527
12  4.419738e-02  4.419738e+00   85.566266
13  4.305616e-02  4.305616e+00   89.871881
14  3.898444e-02  3.898444e+00   93.770325
15  3.696772e-02  3.696772e+00   97.467097
16  2.532903e-02  2.532903e+00  100.000000
17  1.378636e-31  1.378636e-29  100.000000
18  9.453854e-32  9.453854e-30  100.000000
19  1.452962e-32  1.452962e-30  100.000000
20  1.047041e-32  1.047041e-30  100.000000
21  7.727144e-33  7.727144e-31  100.000000
```

Nous voyons que le pourcentage d'inertie totale est expliquée à 100% à partir du 16e axes. Donc, le nombre de facteurs maximum est 16. Représentons graphiquement les valeurs propres (Par défaut : représentation en valeur absolue) pour confirmer le nombre de facteurs

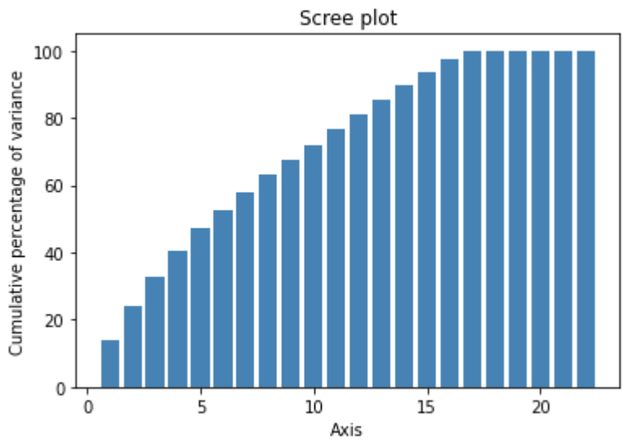
In [ ]:
```
acm.plot_eigenvalues()
```



In [ ]:
```
acm.plot_eigenvalues(type="percentage")
```



In [ ]:
```
acm.plot_eigenvalues(type="cumulative")
```



Extraction des statistiques sur les points lignes

Export de la totalité des données lignes vers une DataFrame pandas

On peut simplement envoyer vers une Dataframe : les coordonnées, les contributions et les cos2 de chacun des points lignes, pour tous les axes factoriels (identifiés par les suffixes dim1, dim2, etc.).

In [ ]:
```
df_rows = acm.row_topandas()
print(df_rows.head())
```

```
   row_coord_dim1  row_coord_dim2  row_coord_dim3  row_coord_dim4  \
0       -0.306426       -0.318816       -0.294252       -0.242257
1        0.049504       -0.038725        0.333656        0.565314
2        0.296587        0.067278       -0.051408        0.216418
3       -0.258446       -0.121505        0.299475       -0.108548
4       -0.391743        0.179505       -0.039579       -0.017802

   row_coord_dim5  row_coord_dim6  row_coord_dim7  row_coord_dim8  \
0        0.069985        0.062901       -0.079568       -0.038858
1       -0.005832        0.163735       -0.081911        0.123279
2       -0.304380       -0.403303        0.036499       -0.005613
3        0.212133       -0.062091        0.164697       -0.053202
4       -0.010479       -0.048901        0.072983       -0.060007

   row_coord_dim9  row_coord_dim10  ...  row_cos2_dim13  row_cos2_dim14  \
0       -0.048859        -0.011086  ...        0.085100        0.002679
```

```
1       -0.003629        -0.002794  ...        0.004221        0.001385
2        0.103982        -0.028562  ...        0.001349        0.042461
3        0.001885         0.022635  ...        0.120993        0.002768
4       -0.005692         0.002525  ...        0.000209        0.001775

   row_cos2_dim15  row_cos2_dim16  row_cos2_dim17  row_cos2_dim18  \
0        0.000371        0.138130        0.001259    3.294677e-31
1        0.000269        0.016092        0.024803    1.401712e-29
2        0.083824        0.081810        0.000009    1.410865e-30
3        0.007612        0.126226        0.001328    2.906012e-30
4        0.026614        0.011708        0.006366    1.827092e-33

   row_cos2_dim19  row_cos2_dim20  row_cos2_dim21  row_cos2_dim22
0    8.822002e-29    3.438223e-30    3.085627e-32    6.977185e-33
1    4.896017e-31    2.976700e-32    5.300047e-30    3.172812e-31
2    3.125797e-32    1.450678e-29    4.191194e-32    3.984300e-34
3    7.258589e-30    3.190658e-33    4.495327e-32    1.181629e-30
4    4.616603e-30    5.986737e-31    7.119405e-31    4.758777e-35

[5 rows x 66 columns]
```

Statistiques pour les points lignes

In [ ]:
```
# Coordonnées des points lignes
print(acm.row_coord_[5:])
```

```
[[-2.84113791e-01  1.00633290e-01  4.19959961e-01 ...  7.56565657e-16
  -2.70493293e-17  7.75816771e-16]
 [-2.58446479e-01 -1.21505226e-01  2.99474858e-01 ...  4.95806650e-16
  -7.67922589e-17 -5.81813482e-16]
 [-3.91742624e-01  1.79504779e-01 -3.95790177e-02 ...  2.42987705e-16
  -7.48683271e-16 -1.12104212e-15]
 ...
 [-3.91742624e-01  1.79504779e-01 -3.95790177e-02 ... -2.48846475e-17
  -8.98427768e-18  3.73343756e-17]
 [-3.91742624e-01  1.79504779e-01 -3.95790177e-02 ... -2.48846475e-17
  -8.98427768e-18  3.73343756e-17]
 [-1.73130163e-01 -6.19825784e-01  4.48022400e-02 ... -6.42593239e-17
  -1.90668927e-17  3.72005314e-18]]
```

In [ ]:
```
# Contributions des points lignes
print(acm.row_contrib_[5:])
```

```
[[6.47184888e-02 1.13140424e-02 2.24263723e-01 ... 4.42141541e+00
  7.84280735e-03 8.74221755e+00]
 [5.35531446e-02 1.64939388e-02 1.14041781e-01 ... 1.89885922e+00
  6.32111118e-02 4.91666755e+00]
 [1.23039737e-01 3.59986731e-02 1.99192636e-03 ... 4.56075425e-01
  6.00834416e+00 1.82535418e+01]
 ...
 [1.23039737e-01 3.59986731e-02 1.99192636e-03 ... 4.78333789e-03
  8.65216667e-04 2.02451532e-02]
 [1.23039737e-01 3.59986731e-02 1.99192636e-03 ... 4.78333789e-03
  8.65216667e-04 2.02451532e-02]
 [2.40319611e-02 4.29213861e-01 2.55236459e-03 ... 3.18963051e-02
  3.89689215e-03 2.01002551e-04]]
```

In [ ]:
```
# Cos2 des points lignes
print(acm.row_cos2_[5:])
```

```
[[2.05927316e-01 2.58352499e-02 4.49930155e-01 ... 1.46023440e-30
  1.86656162e-33 1.53549234e-30]
 [1.87277134e-01 4.13936223e-02 2.51457302e-01 ... 6.89236405e-31
  1.65340178e-32 9.49098037e-31]
 [7.29395840e-01 1.53148821e-01 7.44546197e-03 ... 2.80627498e-31
  2.66414516e-30 5.97317843e-30]
```

```
...
[7.29395840e-01 1.53148821e-01 7.44546197e-03 ... 2.94323279e-33
 3.83643601e-34 6.62490127e-33]
[7.29395840e-01 1.53148821e-01 7.44546197e-03 ... 2.94323279e-33
 3.83643601e-34 6.62490127e-33]
[4.65197836e-02 5.96254251e-01 3.11524113e-03 ... 6.40861991e-33
 5.64224648e-34 2.14778348e-35]]
```

Extraction des statistiques sur les points colonnes

Export de la totalité des données colonnes vers une DataFrame pandas

On peut envoyer vers une Dataframe : les coordonnées, les contributions et les cos2 de chacun des points colonnes, pour tous les axes factoriels (identifiés par les suffixes dim1, dim2, etc.).

In [ ]:
```
df_cols = acm.col_topandas()
print('dimension df_cols:', df_cols.shape)
display(df_cols.head())
```

```
dimension df_cols: (44, 66)
```

	col_coord_dim1	col_coord_dim2	col_coord_dim3	col_coord_dim4	col_coord_dim5	col_coord_dim6	col_coord_dim7	col_coord_dim8	col_coord_dim9	col_coord_dim10	...	col_cos2_dim13	col_cos2_dim14	col_cos2_dim15
Pclass_1_0	-0.124716	0.315026	-0.170821	0.137933	-0.216012	0.096838	-0.156606	0.059930	-0.007955	-0.018404	...	0.112695	0.002735	0.003138
Pclass_1_1	0.389739	-0.984458	0.533814	-0.431041	0.675036	-0.302620	0.489393	-0.187282	0.024861	0.057513	...	0.112695	0.002735	0.003138
Pclass_2_0	-0.062610	-0.051519	-0.308138	-0.138979	0.088734	-0.195456	0.205529	-0.144599	0.006702	0.011733	...	0.052087	0.009252	0.039118
Pclass_2_1	0.240573	0.197958	1.183987	0.534011	-0.340950	0.751018	-0.789723	0.555604	-0.025753	-0.045084	...	0.052087	0.009252	0.039118
Pclass_3_0	0.321122	-0.440547	0.832894	0.012883	0.207682	0.182054	-0.099000	0.154446	0.001578	0.010319	...	0.010712	0.001104	0.043777

5 rows × 66 columns

Statistiques pour les points colonnes

Plus la coordonnée d'une modalité est élevée (en valeur absolue) sur cet axe, plus sa contribution à l'inertie de cet axe est grande :

In [ ]:
```
# Coordonnées des points colonnes
acm.col_coord_[:]
```

In [ ]:
```
# Contributions des points colonnes
print(acm.col_contrib_[5:])
```

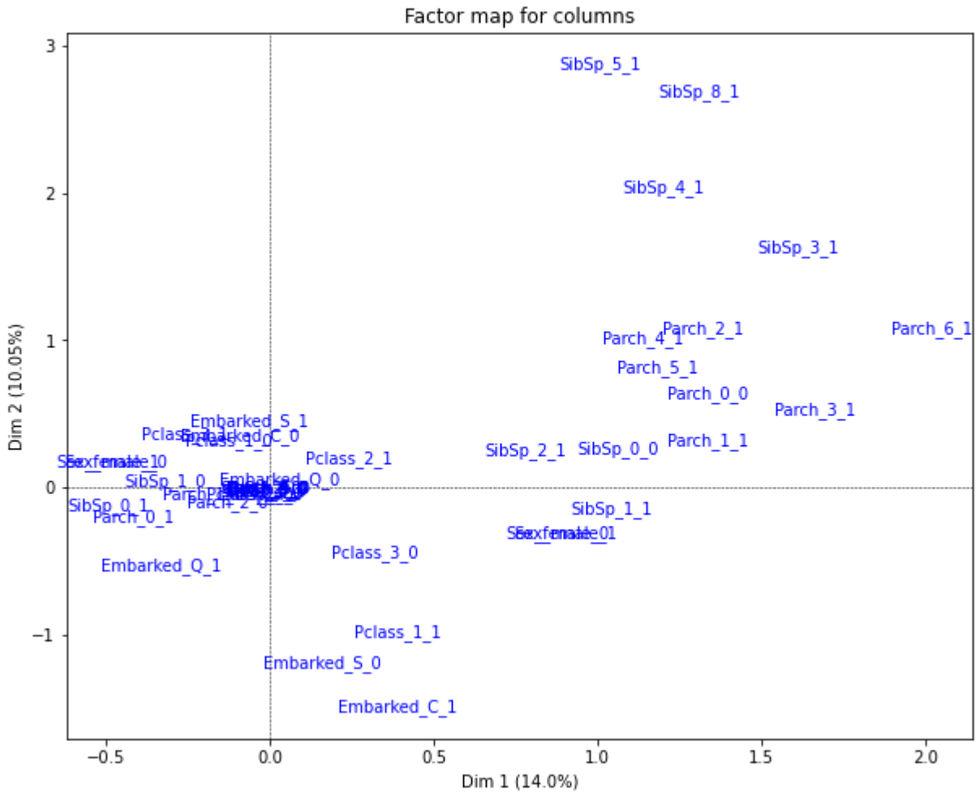
Qualité de la représentation des colonnes

Plus les cos2 sont élevés, plus l'angle entre les points et l'axe est près de zéro et mieux les points sont représentés sur l'axe.

In [ ]:
```
# Cos2 des points colonnes
print(acm.col_cos2_)
```

Graphiques factoriels

In [ ]:
```
acm.mapping_col(1, 2, short_labels=False, figsize=(10, 8))
```



Dans le plan factoriel, le centre correspond au profil moyen des deux variables. En général, on s'intéresse davantage aux modalités qui s'éloignent du centre de gravité qu'à celles au centre. Il s'agit aussi des modalités qui contribuent le plus aux axes.

Partie 3

1. Réaliser une CCA à partir des deux fichiers concaténés en utilisant d'une part comme table X la variable PtClass qui présente 3 modalités (et qui représente le Pont d'embarquement sur le navire), et comme table Y toutes les autres variables qualitatives. L'idée est de trouver de nouveaux axes de représentation qui sont le plus en lien avec le pont d'embarquement (indicateur de classe sociale des passagers)

In [ ]:
```
data2.columns
```

Out[ ]:
```
Index(['Survived', 'Age', 'Fare', 'Pclass_1', 'Pclass_2', 'Pclass_3',
       'Sex_female', 'Sex_male', 'SibSp_0', 'SibSp_1', 'SibSp_2', 'SibSp_3',
       'SibSp_4', 'SibSp_5', 'SibSp_8', 'Parch_0', 'Parch_1', 'Parch_2',
       'Parch_3', 'Parch_4', 'Parch_5', 'Parch_6', 'Embarked_C', 'Embarked_Q',
       'Embarked_S'],
      dtype='object')
```

In [ ]:
```
X = ['Pclass_1', 'Pclass_2', 'Pclass_3']
Y = ['Survived', 'Sex_female', 'Sex_male', 'SibSp_0',
       'SibSp_1', 'SibSp_2', 'SibSp_3', 'SibSp_4', 'SibSp_5', 'SibSp_8',
       'Parch_0', 'Parch_1', 'Parch_2', 'Parch_3', 'Parch_4', 'Parch_5',
       'Parch_6', 'Embarked_C', 'Embarked_Q', 'Embarked_S']
table_X = data2[X]
table_Y = data2[Y]
print('Table X :')
display(table_X.head())
print('Table Y :')
display(table_Y.head())
```

Table X :

	Pclass_1	Pclass_2	Pclass_3
0	0	0	1
1	0	1	0
2	0	0	1
3	1	0	0
4	0	0	1

Table Y :

	Survived	Sex_female	Sex_male	SibSp_0	SibSp_1	SibSp_2	SibSp_3	SibSp_4	SibSp_5	SibSp_8	Parch_0	Parch_1	Parch_2	Parch_3	Parch_4	Parch_5	Parch_6	Embarked_C	Embarked_Q	Embarked_S
0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
1	1	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
2	1	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
3	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
4	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1

In [ ]:
```
from sklearn.cross_decomposition import CCA

CCA_ = CCA()
CCA_.fit(table_X, table_Y)
X_c, Y_c = CCA_.transform(table_X, table_Y)
```

In [ ]:
```
print(X_c.shape)
print(Y_c.shape)
```

```
(891, 2)
(891, 2)
```

In [ ]:
```
results = pd.DataFrame({"CCX_1":X_c[:, 0],
                        "CCY_1":Y_c[:, 0],
                        "CCX_2":X_c[:, 1],
                        "CCY_2":Y_c[:, 1],
                        'Survived': data2.Survived.tolist()})
results.head()
```

Out[ ]:

	CCX_1	CCY_1	CCX_2	CCY_2	Survived
0	-5.430812e-16	-3.029384e-15	1.623973e-16	-1.243975e-15	0
1	2.545692e-15	-1.182295e-16	-1.447448e-16	6.280735e-16	1
2	-5.430812e-16	1.269549e-15	1.623973e-16	-7.041941e-16	1
3	-7.294656e-16	-1.182295e-16	1.187523e-15	6.280735e-16	0
4	-5.430812e-16	-1.182295e-16	1.623973e-16	6.280735e-16	0

In [ ]:
```
import numpy as np
np.corrcoef(X_c[:, 0], Y_c[:, 0])
```

Out[ ]:
```
array([[1.        , 0.22313519],
```

```
       [0.22313519, 1.        ]])
```

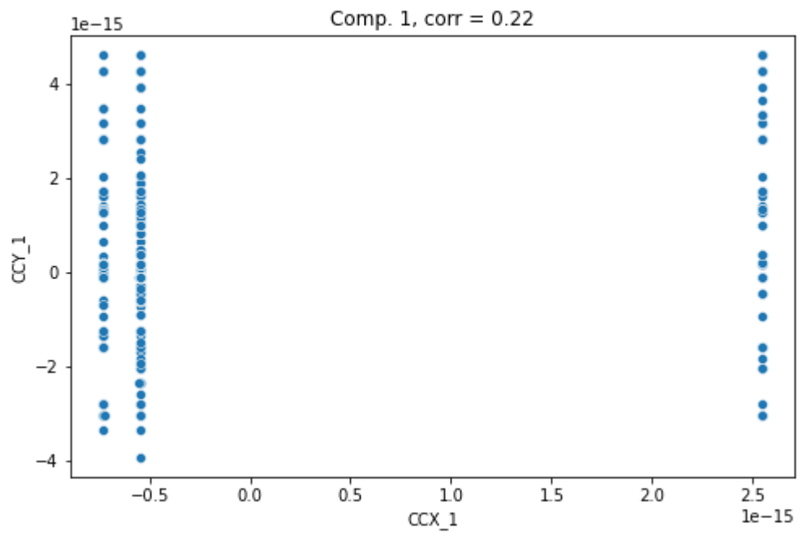
In [ ]:
```
np.corrcoef(X_c[:, 1], Y_c[:, 1])
```

Out[ ]:
```
array([[ 1.        , -0.22825849],
       [-0.22825849,  1.        ]])
```

On voit que les deux premieres axes canoniques ne sont pas tres correlees, de meme aussi pour les deux secondes

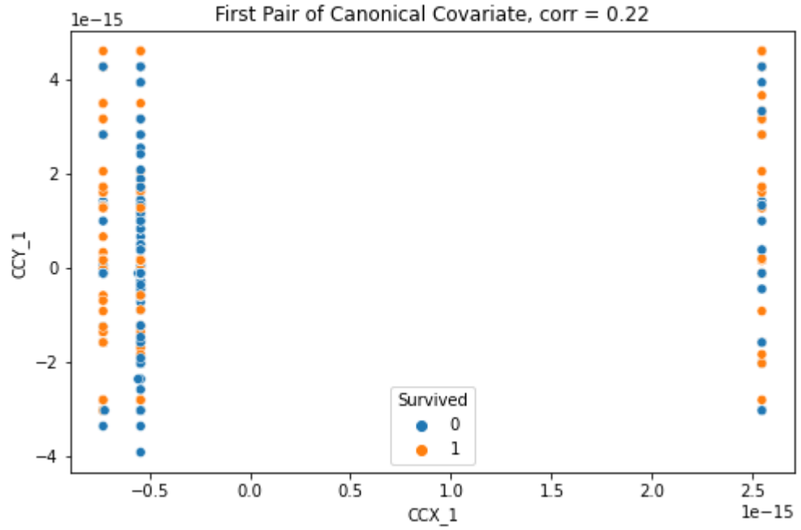
In [ ]:
```
import seaborn as sns

# sns.set_context("talk", font_scale=1.2)
plt.figure(figsize=(8,5))
sns.scatterplot(x="CCX_1", y="CCY_1", data=results)
plt.title('Comp. 1, corr = %.2f' % np.corrcoef(X_c[:, 0], Y_c[:, 0])[0, 1])
```

Out[ ]:
```
Text(0.5, 1.0, 'Comp. 1, corr = 0.22')
```



In [ ]:
```
plt.figure(figsize=(8,5))
sns.scatterplot(x="CCX_1",
                y="CCY_1",
                hue="Survived", data=results)
plt.title('First Pair of Canonical Covariate, corr = %.2f' %
          np.corrcoef(X_c[:, 0], Y_c[:, 0])[0, 1])
```

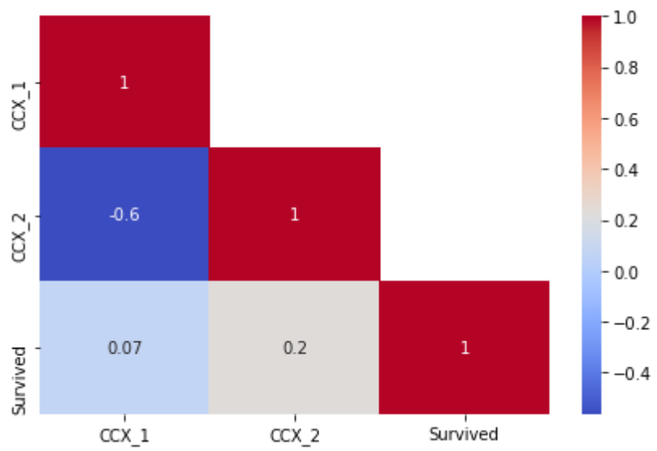
Out[ ]:
```
Text(0.5, 1.0, 'First Pair of Canonical Covariate, corr = 0.22')
```



In [ ]:

```
ccX_df = pd.DataFrame({"CCX_1":X_c[:, 0],
                       "CCX_2":X_c[:, 1],
                       "Survived":data2.Survived.astype('category').cat.codes})


corr_X_df= ccX_df.corr(method='pearson')
display(corr_X_df.head())
X_df_lt = corr_X_df.where(np.tril(np.ones(corr_X_df.shape)).astype(bool))
sns.heatmap(X_df_lt,cmap="coolwarm",annot=True,fmt='.1g')
plt.tight_layout()
```

	CCX_1	CCX_2	Survived
CCX_1	1.000000	-0.563158	0.073603
CCX_2	-0.563158	1.000000	0.232101
Survived	0.073603	0.232101	1.000000



In [ ]:

```
ccY_df = pd.DataFrame({"CCY_1":Y_c[:, 0],
                       "CCY_2":Y_c[:, 1],
                       "Survived":data2.Survived.astype('category').cat.codes})
```

```
corr_Y_df= ccY_df.corr(method='pearson')
display(corr_Y_df.head())
X_df_lt = corr_Y_df.where(np.tril(np.ones(corr_Y_df.shape)).astype(bool))
sns.heatmap(X_df_lt,cmap="coolwarm",annot=True,fmt='.1g')
plt.tight_layout()
```

	CCY_1	CCY_2	Survived
CCY_1	1.000000	-0.132399	0.122320
CCY_2	-0.132399	1.000000	-0.398458
Survived	0.122320	-0.398458	1.000000