MI
MI
ML

Superhero

GROUPE 11

Loïc Amiand, François Bélières, Cassandra Nguyen, Riswane Maricar

TABLE OF CONTENTS

01 **Data exploration et cleaning**
Prise de connaissance du dataset

02 **Visualisations et constats**
Data visualization pour explorer le contenu

03 **Entraîner le modèle**
Appliquer les modèles

04 **API**
Notre solution

Besoin métier

En tant que grande maison d’édition, on veut innover dans la création de nos nouveaux super-héros

⇒ identifier les caractéristiques propres qui font la force de notre maison
⇒ identifier les points forts de nos concurrents pour s’en inspirer un peu

Besoin métier

On souhaite également anticiper les **risques de plagiat** envers la concurrence dans la création de nos nouveaux super-héros

⇒ **éviter toute ressemblance trop proche**

Besoin métier

Solution : une API qui analyse le ‘script’ du personnage proposé pour l’équipe marketing avec toute la data visualisation sur ses attributs

⇒ voir si le héros rentre dans notre gamme et nos critères de création

01

Data exploration et cleaning

Prise de connaissance du dataset

1. Data exploration et cleaning

- **Nettoyage des données**

- vérification si existence de valeurs nulles : `df.isna().any()`
- identification du nombre de valeurs nulles : `df.isna().sum()`
- estimation en % du nombre de valeurs nulles par catégorie : ex. `skin_color => 88.07%`

```
#pourcentage missing values
def perc_missing(df):
  '''prints out columns with missing values with its %'''
  for col in df.columns :
      pct = df[col].isna().mean() * 100
      if (pct != 0):
          print('{} => {}%'.format(col, round(pct, 2)))

perc_missing(df)
```

```
name => 0.14%
real_name => 10.28%
full_name => 34.07%
history_text => 6.21%
powers_text => 25.1%
```

1. Data exploration et cleaning

- **Nettoyage des données**

- visualisation des valeurs nulles en heatmap

- corrélation avec nos résultats précédents

`skin_color => 88.07%`

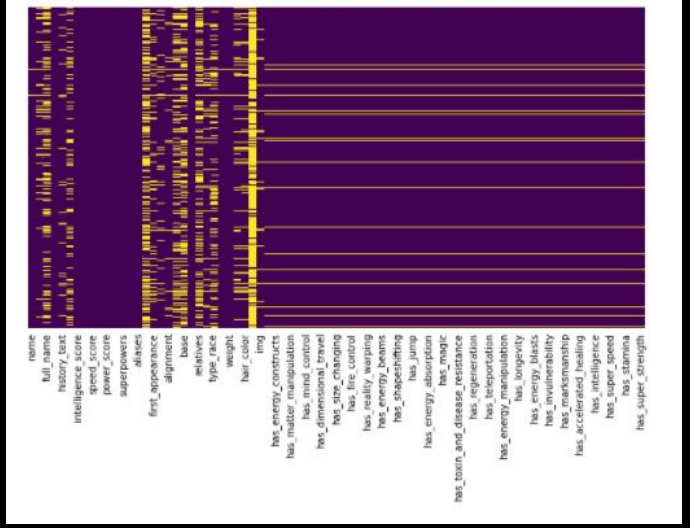
- nettoyage des caractères spéciaux

```
#replace false nan
df['overall_score'] = df['overall_score'].replace(['-'], 'NaN')
df['height'] = df['height'].replace(['-'], 'NaN')
df['weight'] = df['weight'].replace(['-'], 'NaN')
df['alter_egos'] = df['alter_egos'].replace(['[]'], 'NaN')
df['superpowers'] = df['superpowers'].replace(['[]'], 'NaN')
df['teams'] = df['teams'].replace(['[]'], 'NaN')
df['aliases'] = df['aliases'].replace(["['']"], 'NaN')
```

```
#replace infinite number
df['overall_score'] = df['overall_score'].replace(['∞'], 1000)
```

```
#heatmap missing values
plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), yticklabels=False, cmap='viridis', cbar=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd6eb0c9b50>
```



PORTFOLIO AUDIO ENGINEER PORTFOLIO AUDIO ENGINEER

1. Data exploration et cleaning

- **Nettoyage des données**

- prédiction de certaines valeurs manquantes par imputation KNN

Check missing values

```
[ ] df[['height', 'weight']].isna().any()
```

```
height    False
weight     True
dtype: bool
```

```
df[['height', 'weight']].isna().sum()
```

```
height     0
weight    39
dtype: int64
```

Imputer process

```
[ ] imputer = KNNImputer(n_neighbors = 5)
    df2 = pd.DataFrame(imputer.fit_transform(df2), columns = df2.columns)
```

```
[ ] df2.isna().any()
```

```
height    False
weight    False
dtype: bool
```

PORTFOLIO AUDIO ENGINEER PORTFOLIO AUDIO ENGINEER

02

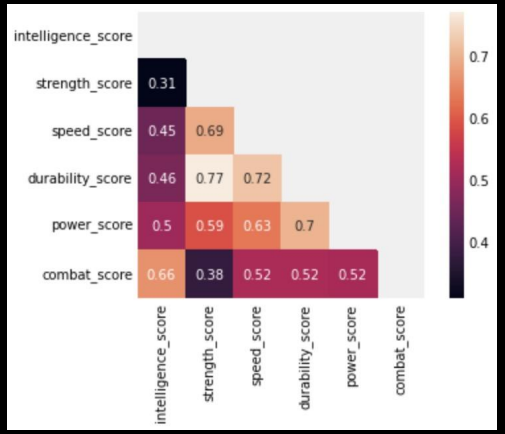
Visualisations et constats

Data visualization pour explorer le contenu

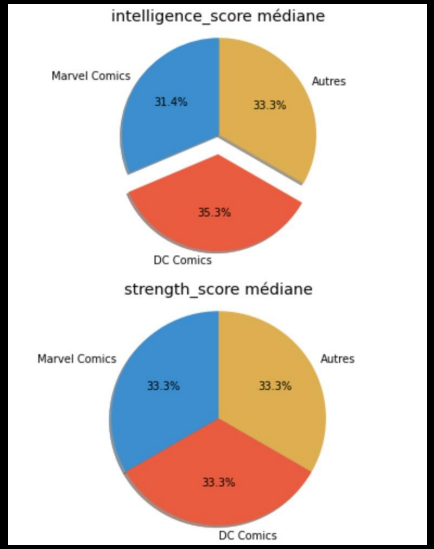
2. Visualisations et constats

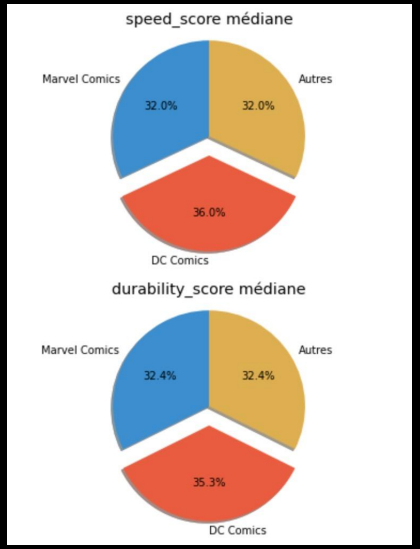
- Focus sur les stats des scores

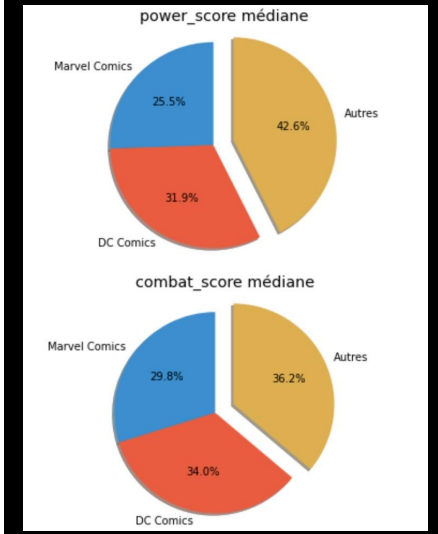
Piste intéressante : A creuser sur ces 2 métriques



Corrélation entre scores







DC domine... sauf pour power et combat

03 Entraîner le modèle

Appliquer les modèles

3. Modélisation

ACP : attributs physiques, préférences parmi les maisons ?

On remplace les valeurs nulles

	gender	type_race	eye_color	hair_color	creator
0	Male	Human	non renseigné	non renseigné	Marvel Comics
1	no gender	Autres	non renseigné	non renseigné	DC Comics
2	Male	Human	Yellow	No Hair	Marvel Comics
3	Male	Human	non renseigné	non renseigné	DC Comics
4	Male	Human	non renseigné	non renseigné	DC Comics
...	...	...	...	...	...
1445	Female	Human	Blue	Black	DC Comics
1446	Male	Autres	Blue	Blond	Capcom
1447	Male	Autres	Red	Brown	DC Comics
1448	Male	Autres	Red	Brown	DC Comics
1449	no gender	Autres	No eyes	No Hair	Marvel Comics

1450 rows × 5 columns

Puis on les dummifie

```
df_acp #dummifié, prêt pour le PCA
```

	gender	type_race	eye_color	hair_color	creator
0	1	1	0	0	Marvel Comics
1	0	0	0	0	DC Comics
2	1	1	1	1	Marvel Comics
3	1	1	0	0	DC Comics
4	1	1	0	0	DC Comics
...	...	...	...	...	...
1445	1	1	1	1	DC Comics
1446	1	0	1	1	Capcom
1447	1	0	1	1	DC Comics
1448	1	0	1	1	DC Comics
1449	0	0	0	1	Marvel Comics

3. Modélisation

On crée le train/test split

ACP sur genre, type race, eye color, hair color : quels sont les favoris de Marvel et DC ?

```
[185] from sklearn.datasets import fetch_openml
      from sklearn.decomposition import PCA
      from sklearn.preprocessing import StandardScaler
      from sklearn import metrics
      from sklearn.model_selection import train_test_split
      import pandas as pd
```

Splitting Data into Training and Test Sets

```
[187] df_acp['creator'] = df_acp['creator'].fillna('non renseigné')
```

```
[253] X = df_acp[[ 'gender', 'type_race', 'eye_color', 'hair_color']] #data
      y = df_acp['creator'] #target
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0)
```

On centre et réduit les données

```
[194] from sklearn.preprocessing import StandardScaler
      scaler = StandardScaler()

      # Fit on training set
      scaler.fit(X_train)

      # Apply transform to both the training set and the test set.
      X_train = scaler.transform(X_train)
      X_test = scaler.transform(X_test)
```

3. Modélisation

```
[214] from sklearn.decomposition import PCA

[215] pca = PCA(.95)

[217] pca.fit(X_train)

      PCA(n_components=0.95)

[218] pca.n_components_ #3 composants

      3
```

```
#mapping (transform) sur training set et test set.
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
```

```
] from sklearn.linear_model import LogisticRegression

 # instance du modèle

 logisticRegr = LogisticRegression(solver = 'lbfgs')

] # Entraînement du modèle

 logisticRegr.fit(X_train, y_train)
```

Un score de performance de 40% sur le test set

```
score = logisticRegr.score(X_test, y_test)
print(score)
```

```
0.39655172413793105
```

Variance :
comp1 : 53%, comp2 : 32%, comp3 : 11% d'infos

```
pca.explained_variance_ratio_
```

```
array([0.5341873 , 0.31540601, 0.11165336])
```

NLP – Preprocessing

1. Répartition des créateurs en 3 catégories: Marvel, DC et Autres

2. Concaténation de toutes les colonnes textuelles

3. Preprocessing avec Gensim

4. Utilisation d’un Count Vectorizer pour transformer nos données en vecteurs

NLP – Modèle & résultats

1. Bernoulli Naive Bayes
   - Accuracy: `0.664367816091954`

2. Gradient Boosting
   - Accuracy: `0.8206896551724138`

3. Multinomial Naive Bayes
   - Accuracy: `0.896551724137931`

	precision	recall	f1-score	support
Autres	0.94	0.16	0.27	107
DC Comics	0.96	0.59	0.73	137
Marvel Comics	0.57	1.00	0.73	191
accuracy			0.66	435
macro avg	0.83	0.58	0.58	435
weighted avg	0.79	0.66	0.62	435

	precision	recall	f1-score	support
Autres	0.63	0.86	0.73	107
DC Comics	0.92	0.77	0.84	137
Marvel Comics	0.91	0.83	0.87	191
accuracy			0.82	435
macro avg	0.82	0.82	0.81	435
weighted avg	0.85	0.82	0.83	435

	precision	recall	f1-score	support
Autres	0.93	0.76	0.84	107
DC Comics	0.85	0.94	0.90	137
Marvel Comics	0.91	0.94	0.93	191
accuracy			0.90	435
macro avg	0.90	0.88	0.89	435
weighted avg	0.90	0.90	0.89	435

04 API

Notre solution

NLP – API

On a gardé le Multinomial Naive Bayes

Obligation de preprocess le texte en entrée de l’API

Présentation de FastAPI

Manque de temps pour ajouter les valeurs numériques ou dummifiées

```
@app.post('/predict')
def predict_creator(data:MetricsHeroes):
    data = data.dict()

    stop_words = set(stopwords.words('english'))

    texte = data['concat']

    df_text = gensim.utils.simple_preprocess(texte, deacc=True, min_len=1)
    df_text = [wrd for wrd in df_text if wrd not in stop_words]
    df_text = [item.lower() for item in df_text]

    df_text = ' '.join(df_text)
    df_text=[df_text]

    texte_v = CV.transform(df_text)
    prediction = regressor.predict(texte_v)

    return {
        'prediction': prediction[0]
    }
```