

Exploration des Données (EDA) pour un Projet de Machine Learning

Georf MIGUIAMA BAMBA

Janvier 2026

Document méthodologique : bonnes pratiques EDA (de la compréhension métier jusqu'aux prochaines étapes).

Table des matières

1	Introduction	3
2	Comprendre le Problème	3
3	Chargement des Données	3
4	Vérification de la Qualité des Données	3
5	Analyse Univariée	4
6	Analyse Bivariée	5
7	Opportunités d'Ingénierie des Caractéristiques	5
8	Détection des Valeurs Aberrantes et Anomalies	6
9	Gestion des Valeurs Manquantes	6
10	Notions Avancées Souvent Oubliées en EDA	7
10.1	Détection de fuite de cible (target leakage)	7
10.2	Déséquilibre de classes (classification)	7
10.3	Split train/test avant certaines transformations	7
10.4	Scaling et pipelines (éviter les fuites)	7
11	Vérification du Dataset Final	8
12	Insights et Prochaines Étapes	8
13	Conclusion	9

1 Introduction

L'**Exploration des Données (EDA)** est une étape indispensable avant l'entraînement d'un modèle. Elle permet de :

- comprendre la structure et la qualité des données ;
- détecter les incohérences (valeurs manquantes, doublons, outliers) ;
- identifier des relations utiles entre variables ;
- réduire les risques d'erreurs (fuite de cible, biais, variables non pertinentes) ;
- guider les choix de transformation, de modélisation et d'évaluation.

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
```

Imports de base EDA

2 Comprendre le Problème

Avant de coder, définir le cadre métier est crucial.

- Quel est l'objectif métier ?
- Quelles sont les variables clés (cible et caractéristiques) ?
- Y a-t-il des contraintes spécifiques ?

```
1 problem = {
2     "type": "classification",      # ou regression / forecasting
3     "target": "y",
4     "metric": "f1",                # ex: rmse, mae, auc, etc.
5     "constraints": ["latency<200ms", "interpretability"]
6 }
7 print(problem)
```

Checklist rapide métier

3 Chargement des Données

- Lecture du fichier (CSV, Excel, SQL, etc.)
- Aperçu des premières lignes (`df.head()`)
- Vérification de la dimension du dataset (`df.shape`)
- Informations générales sur les colonnes (`df.info()`)

```
1 df = pd.read_csv("data.csv")           # ou pd.read_excel(...)
2 print(df.head())
3 print("Shape:", df.shape)
4 print(df.info())
```

Chargement et aperçu

4 Vérification de la Qualité des Données

- Valeurs manquantes (`df.isnull().sum()`)
- Doublons (`df.duplicated().sum()`)

- Types de données (df.dtypes)
- Détection des valeurs aberrantes (Boxplots, IQR, Z-score)

```

1 missing = df.isnull().sum().sort_values(ascending=False)
2 missing_ratio = (df.isnull().mean() * 100).sort_values(ascending=False)
3
4 duplicates = df.duplicated().sum()
5 types = df.dtypes
6
7 print("Duplicates:", duplicates)
8 print(missing.head(10))
9 print(missing_ratio.head(10))
10 print(types)

```

Qualité des données: manquants/doublons/types

```

1 plt.figure()
2 sns.heatmap(df.isnull(), cbar=False)
3 plt.title("Heatmap des valeurs manquantes")
4 plt.show()

```

Carte thermique des valeurs manquantes

5 Analyse Univariée

- Analyse des variables numériques (df.describe(), histogrammes)
- Analyse des variables catégorielles (df['colonne'].value_counts())

```

1 num_cols = df.select_dtypes(include="number").columns
2
3 desc = df[num_cols].describe().T
4 desc["missing_%"] = df[num_cols].isnull().mean().values * 100
5 desc["skew"] = df[num_cols].skew(numeric_only=True)
6 desc["kurtosis"] = df[num_cols].kurt(numeric_only=True)
7
8 print(desc.sort_values("missing_%", ascending=False).head(10))

```

Résumé univarié robuste

```

1 for c in num_cols[:10]: # limite pour éviter trop de figures
2     plt.figure()
3     df[c].hist(bins=30)
4     plt.title(f"Distribution: {c}")
5     plt.show()

```

Histogrammes numériques

```

1 cat_cols = df.select_dtypes(exclude="number").columns
2
3 for c in cat_cols[:10]:
4     vc = df[c].value_counts(dropna=False)
5     print("\nColonne:", c)
6     print("Cardinalité:", df[c].nunique(dropna=False))
7     print(vc.head(10))

```

Analyse catégorielle (cardinalité)

6 Analyse Bivariée

- Relation Numérique vs Numérique (Corrélation, Scatter plot)
- Relation Catégorique vs Numérique (Boxplot)
- Relation Catégorique vs Catégorique (Crosstab, graphiques empilés)

```
1 corr = df[num_cols].corr()  
2 plt.figure()  
3 sns.heatmap(corr, annot=False)  
4 plt.title("Matrice de corrélation (numérique)")  
5 plt.show()
```

Correlation et heatmap

```
1 # Remplace x/y par tes variables  
2 # sns.scatterplot(data=df, x="feature_num_1", y="feature_num_2")  
3 # plt.show()
```

Scatter plot (exemple)

```
1 # sns.boxplot(data=df, x="feature_cat", y="target")  
2 # plt.xticks(rotation=45)  
3 # plt.show()
```

Catégorique vs numérique (boxplot - exemple)

```
1 # Exemple:  
2 # ct = pd.crosstab(df["cat1"], df["cat2"], normalize="index")  
3 # print(ct)
```

Catégorique vs catégorique (crosstab)

7 Opportunités d'Ingénierie des Caractéristiques

- Création de nouvelles variables (date, agrégations, etc.)
- Encodage des variables catégorielles (One-hot encoding, Label encoding)
- Transformation des données biaisées (Log, Box-Cox)

```
1 # Exemple: ratio  
2 # df["ratio"] = df["a"] / (df["b"] + 1e-9)  
3  
4 # Exemple dates  
5 # df["date"] = pd.to_datetime(df["date"])  
6 # df["month"] = df["date"].dt.month  
7 # df["dayofweek"] = df["date"].dt.dayofweek
```

Creation de features (exemples)

```
1 # One-hot encoding  
2 # df = pd.get_dummies(df, columns=["feature_cat"], drop_first=True)  
3  
4 # Label encoding (attention: impose un ordre arbitraire)  
5 # from sklearn.preprocessing import LabelEncoder  
6 # le = LabelEncoder()  
7 # df["feature_cat_le"] = le.fit_transform(df["feature_cat"].astype(str))
```

Encodage simple

```

1 # Log (si forte asymetrie)
2 # df["x_log"] = np.log1p(df["x"])
3
4 # Box-Cox (necessite x>0)
5 # from scipy.stats import boxcox
6 # df["x_boxcox"], lam = boxcox(df["x"] + 1e-9)

```

Transformations de distributions

8 Détection des Valeurs Aberrantes et Anomalies

- Boxplots, Z-score, IQR
- Détection avancée : Isolation Forest, DBSCAN

```

1 col = "feature_num" # a remplacer
2 Q1 = df[col].quantile(0.25)
3 Q3 = df[col].quantile(0.75)
4 IQR = Q3 - Q1
5
6 mask_out = (df[col] < Q1 - 1.5*IQR) | (df[col] > Q3 + 1.5*IQR)
7 outliers = df[mask_out]
8 print("Nb outliers:", outliers.shape[0])

```

Outliers via IQR (exemple)

```

1 # from sklearn.ensemble import IsolationForest
2 # num_data = df.select_dtypes(include="number").fillna(0)
3 # iso = IsolationForest(contamination=0.01, random_state=42)
4 # df["anomaly_flag"] = iso.fit_predict(num_data) # -1 anomalie, 1
      normal
5 # print(df["anomaly_flag"].value_counts())

```

Anomalies avancees (IsolationForest - optionnel)

9 Gestion des Valeurs Manquantes

- Suppression des valeurs manquantes
- Imputation (moyenne, médiane, mode, interpolation)
- Imputation prédictive avec modèles ML

```

1 # 1) Drop (si faible proportion)
2 df_drop = df.dropna()
3
4 # 2) Imputation simple
5 # Num: mediane / moyenne
6 # Cat: mode
7 # (a adapter colonne par colonne)
8 # df["num_col"] = df["num_col"].fillna(df["num_col"].median())
9 # df["cat_col"] = df["cat_col"].fillna(df["cat_col"].mode()[0])
10
11 # 3) Imputation predictive
12 # from sklearn.impute import KNNImputer
13 # imputer = KNNImputer(n_neighbors=5)

```

```

14 # df[num_cols] = imputer.fit_transform(df[num_cols])
      Stratégies d'imputation

```

10 Notions Avancées Souvent Oubliées en EDA

Cette section regroupe des contrôles très pratiques pour éviter des erreurs fréquentes.

10.1 Détection de fuite de cible (target leakage)

```

1 target = "y" # a remplacer
2 if target in df.columns:
3     tmp = df.select_dtypes(include="number").drop(columns=[target],
4           errors="ignore")
4     corr_to_target = df[tmp.columns].corrwith(df[target]).sort_values(
5       key=lambda s: s.abs(), ascending=False)
5     print(corr_to_target.head(10))

```

Recherche de variables trop explicatives

10.2 Déséquilibre de classes (classification)

```

1 # if target in df.columns:
2 #     print(df[target].value_counts(normalize=True))

```

Verification imbalance

10.3 Split train/test avant certaines transformations

```

1 # from sklearn.model_selection import train_test_split
2 # X = df.drop(columns=[target])
3 # y = df[target]
4 # X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
5   =0.2, random_state=42, stratify=y if y.unique().size < 20 else None)

```

Bon reflexe: split avant fit preprocessing

10.4 Scaling et pipelines (éviter les fuites)

```

1 # from sklearn.compose import ColumnTransformer
2 # from sklearn.preprocessing import OneHotEncoder, StandardScaler
3 # from sklearn.pipeline import Pipeline
4 # from sklearn.impute import SimpleImputer
5 # from sklearn.linear_model import LogisticRegression
6
7 # num_features = X_train.select_dtypes(include="number").columns
8 # cat_features = X_train.select_dtypes(exclude="number").columns
9
10 # numeric_tf = Pipeline(steps=[
11 #     ("imputer", SimpleImputer(strategy="median")),
12 #     ("scaler", StandardScaler())

```

```

13 # ])
14
15 # categorical_tf = Pipeline(steps=[
16 #     ("imputer", SimpleImputer(strategy="most_frequent")),
17 #     ("onehot", OneHotEncoder(handle_unknown="ignore"))
18 # ])
19
20 # preprocessor = ColumnTransformer(
21 #     transformers=[
22 #         ("num", numeric_tf, num_features),
23 #         ("cat", categorical_tf, cat_features)
24 #     ]
25 # )
26
27 # model = Pipeline(steps=[
28 #     ("preprocess", preprocessor),
29 #     ("clf", LogisticRegression(max_iter=1000))
30 # ])
31
32 # model.fit(X_train, y_train)

```

Pipeline sklearn (exemple)

11 Vérification du Dataset Final

- Résumé des données après nettoyage
- Sauvegarde du dataset propre (df.to_csv('cleaned_data.csv'))

```

1 print("Shape:", df.shape)
2 print("Missing top:", df.isnull().sum().sort_values(ascending=False).
      head(10))
3 print("Duplicates:", df.duplicated().sum())
4
5 # Sauvegarde
6 df.to_csv("cleaned_data.csv", index=False)

```

Checks finaux

12 Insights et Prochaines Étapes

- Synthèse des découvertes principales
- Identification des problèmes potentiels
- Planification des prochaines étapes (sélection de variables, choix du modèle)

```

1 summary = {
2     "n_rows": df.shape[0],
3     "n_cols": df.shape[1],
4     "duplicates": int(df.duplicated().sum()),
5     "missing_cells_%": float(df.isnull().mean().mean() * 100)
6 }
7 print(summary)

```

Exemple de mini synthèse automatique

13 Conclusion

Une EDA bien menée permet de gagner du temps, réduire les bugs, et améliorer la qualité du modèle. Elle doit être :

- **structurée** (checklist + étapes reproductibles),
- **traçable** (résultats sauvegardés, décisions documentées),
- **orientée métier** (métriques, contraintes, interprétabilité),
- **sans fuite de données** (split/pipelines).