Manuel de Préparation: Exploratory Data Analysis & Data Wrangling

Introduction

L'Exploratory Data Analysis (EDA) et le Data Wrangling sont les fondations de tout projet d'analyse de données. Avant de construire des modèles ou de créer des dashboards, il faut comprendre et préparer les données. Dans le contexte bancaire, cette étape est critique car les décisions basées sur des données mal comprises ou mal nettoyées peuvent avoir des conséquences financières et réglementaires importantes.

Partie 1: Exploratory Data Analysis (EDA)

1.1 Définition et Objectifs

EDA = Analyse exploratoire des données

Processus systématique d'investigation des données pour: - Comprendre la structure et les caractéristiques - Découvrir des patterns et relations - Identifier des anomalies et problèmes de qualité - Formuler des hypothèses - Préparer les analyses ultérieures

1.2 Le Framework EDA

```
                    FRAMEWORK EDA

  1. Comprendre le contexte business
  ↓
  2. Collecter et charger les données
  ↓
  3. Examiner la structure des données
  ↓
  4. Analyser les statistiques univariées
  ↓
  5. Explorer les relations bivariées/multivariées
  ↓
  6. Identifier et traiter les problèmes de qualité
  ↓
  7. Documenter les insights et décisions
```

1.3 Étape 1: Comprendre le Contexte Business

Questions à poser AVANT de toucher aux données

- Quel est l'objectif business de l'analyse?
- Qui sont les stakeholders et leurs besoins?
- Quelles décisions seront prises avec ces données?
- Quelles sont les contraintes (temps, ressources, réglementation)?

Contexte Bancaire - Exemples

```
Objectif: Réduire le taux de défaut des prêts
- Données nécessaires: Historique des prêts, caractéristiques clients
- Décision attendue: Améliorer le scoring ou les politiques de crédit
- Contraintes: Réglementation BRH, équité du traitement

Objectif: Optimiser la rétention client
- Données nécessaires: Transactions, comportement digital, contacts
- Décision attendue: Campagnes ciblées, offres personnalisées
- Contraintes: RGPD/protection des données, budget marketing
```

1.4 Étape 2: Charger et Examiner la Structure

Python - Code de Base

```python
import pandas as pd
import numpy as np

# Charger les données
df = pd.read_csv('prets_bancaires.csv')

# Premières explorations
df.head()          # Premiers enregistrements
df.tail()          # Derniers enregistrements
df.shape           # (lignes, colonnes)
df.info()          # Types de données, valeurs non-null
df.dtypes          # Types de chaque colonne
df.columns         # Noms des colonnes
df.describe()      # Statistiques descriptives
```

Questions Clés

Question	Commande Python	Pourquoi c'est important
Combien de lignes/colonnes?	`df.shape`	Taille du dataset
Quels types de données?	`df.dtypes`	Déterminer les analyses possibles
Y a-t-il des valeurs manquantes?	`df.isnull().sum()`	Qualité des données
Quelles sont les valeurs uniques?	`df['col'].nunique()`	Cardinalité des catégories

---

1.5 Étape 3: Analyse Univariée

Pour les Variables Numériques

```python
# Statistiques descriptives
df['montant_pret'].describe()
```

```python
# Distribution
df['montant_pret'].hist(bins=30)

# Box plot pour outliers
df.boxplot(column='montant_pret')

# Skewness et Kurtosis
from scipy import stats
print(f"Skewness: {stats.skew(df['montant_pret'].dropna())}")
print(f"Kurtosis: {stats.kurtosis(df['montant_pret'].dropna())}")
```

Interprétation des Statistiques

Statistique	Signification	Alerte si
Mean vs Median	Asymétrie	Grande différence
Std	Dispersion	Très grand vs moyenne
Min/Max	Étendue	Valeurs impossibles
Skewness	Asymétrie	> 1 ou < -1
Kurtosis	Queues de distribution	> 3 (leptokurtic)

Pour les Variables Catégorielles

```python
# Fréquences
df['type_compte'].value_counts()

# Proportions
df['type_compte'].value_counts(normalize=True)

# Visualisation
df['type_compte'].value_counts().plot(kind='bar')
```

Red Flags Catégorielles

- Catégorie dominante (> 90%): déséquilibre
- Trop de catégories: considérer regroupement
- Valeurs inattendues: erreurs de saisie

1.6 Étape 4: Analyse Bivariée et Multivariée

Numérique vs Numérique

```python
# Corrélation
df[['montant_pret', 'revenu', 'score_credit']].corr()

# Scatter plot
import matplotlib.pyplot as plt
plt.scatter(df['revenu'], df['montant_pret'])
plt.xlabel('Revenu')
plt.ylabel('Montant du prêt')
```

```python
# Heatmap de corrélation
import seaborn as sns
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
```

Catégorielle vs Numérique

```python
# Statistiques par groupe
df.groupby('type_compte')['solde'].describe()

# Box plot par groupe
df.boxplot(column='solde', by='type_compte')

# Test statistique (ANOVA si besoin)
from scipy import stats
groups = [group['solde'].values for name, group in df.groupby('type_compte')]
stat, p_value = stats.f_oneway(*groups)
```

Catégorielle vs Catégorielle

```python
# Table de contingence
pd.crosstab(df['type_compte'], df['statut_pret'])

# Avec pourcentages
pd.crosstab(df['type_compte'], df['statut_pret'], normalize='index')

# Test Chi-carré
from scipy.stats import chi2_contingency
chi2, p, dof, expected = chi2_contingency(pd.crosstab(df['type_compte'], df['statut_pret']))
```

1.7 Patterns et Anomalies à Rechercher

Patterns Courants dans les Données Bancaires

Pattern	Description	Exemple
Saisonnalité	Variations cycliques	Plus de retraits en fin de mois
Tendance	Direction générale	Croissance des dépôts digitaux
Clusters	Groupes naturels	Segments de clients
Corrélations	Relations entre variables	Revenu et montant accordé

Anomalies à Investiguer

Type	Comment détecter	Action
Outliers	Box plot, Z-score	Vérifier si réels ou erreurs
Valeurs impossibles	Règles métier	Solde négatif sans autorisation
Incohérences	Cross-validation	Date fin < Date début

Type	Comment détecter	Action
Doublons	`df.duplicated()`	Identifier la source

1.8 Documentation des Insights

Template de Documentation EDA

```
## Dataset: Portefeuille de Prêts
**Date d'analyse:** 2024-01-15
**Analyste:** [Nom]

### Résumé
- 10,000 prêts sur la période 2020-2023
- 15% de valeurs manquantes dans 'revenu_client'
- Distribution des montants asymétrique (skew = 1.8)

### Insights Clés
1. Forte corrélation (r=0.75) entre score_credit et approbation
2. Secteur agriculture sur-représenté dans les défauts
3. Saisonnalité marquée en décembre (pic de décaissements)

### Problèmes de Qualité
- 500 doublons identifiés
- 3 valeurs de montant négatives (erreurs)
- Catégorie 'Autre' représente 40% des secteurs

### Recommandations
- Nettoyer les doublons avant analyse
- Investiguer les valeurs négatives
- Réviser la nomenclature des secteurs
```

Partie 2: Data Wrangling

2.1 Définition

Data Wrangling = Préparation et transformation des données

Processus de conversion des données brutes en format analysable: - Nettoyage (cleaning) - Transformation (transforming) - Enrichissement (enriching) - Validation (validating)

2.2 Gestion des Valeurs Manquantes

Identifier les Valeurs Manquantes

```
# Comptage
df.isnull().sum()

# Pourcentage
(df.isnull().sum() / len(df)) * 100
```

```python
# Visualisation
import missingno as msno
msno.matrix(df)
msno.heatmap(df)  # Corrélation des manquants
```

Types de Valeurs Manquantes

Type	Description	Exemple	Traitement
MCAR	Missing Completely At Random	Erreur technique aléatoire	Suppression ou imputation
MAR	Missing At Random	Dépend d'autres variables	Imputation conditionnelle
MNAR	Missing Not At Random	Dépend de la valeur elle-même	Très complexe

Stratégies de Traitement

```python
# 1. Suppression des lignes
df_clean = df.dropna()  # Supprime toutes les lignes avec NA
df_clean = df.dropna(subset=['colonne_critique'])  # Que si colonne critique

# 2. Suppression des colonnes (si > 50% manquant)
df_clean = df.dropna(axis=1, thresh=len(df)*0.5)

# 3. Imputation par valeur constante
df['col'].fillna(0, inplace=True)
df['col'].fillna('Inconnu', inplace=True)

# 4. Imputation par statistique
df['col'].fillna(df['col'].mean(), inplace=True)
df['col'].fillna(df['col'].median(), inplace=True)
df['col'].fillna(df['col'].mode()[0], inplace=True)

# 5. Imputation par groupe
df['revenu'].fillna(df.groupby('segment')['revenu'].transform('median'), inplace=True)

# 6. Interpolation (données temporelles)
df['solde'].interpolate(method='linear', inplace=True)

# 7. Imputation avancée (KNN, MICE)
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=5)
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
```

Quand utiliser quelle stratégie?

Situation	Stratégie Recommandée
< 5% manquant	Suppression des lignes
Variable critique avec peu de manquants	Imputation médiane/mode
Pattern de manquants corrélé	Imputation conditionnelle

Situation	Stratégie Recommandée
Données temporelles	Interpolation
> 50% manquant	Supprimer la colonne

2.3 Traitement des Outliers

Détection des Outliers

```
# Méthode IQR
Q1 = df['montant'].quantile(0.25)
Q3 = df['montant'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df['montant'] < lower_bound) | (df['montant'] > upper_bound)]

# Méthode Z-score
from scipy import stats
z_scores = np.abs(stats.zscore(df['montant']))
outliers = df[z_scores > 3]

# Méthode MAD (Median Absolute Deviation) - Plus robuste
median = df['montant'].median()
mad = np.median(np.abs(df['montant'] - median))
modified_z_scores = 0.6745 * (df['montant'] - median) / mad
outliers = df[np.abs(modified_z_scores) > 3.5]
```

Traitement des Outliers

Stratégie	Quand l'utiliser	Code Python
Garder	Valeurs réelles importantes	N/A
Supprimer	Erreurs de données	`df = df[~outliers_mask]`
Capper	Réduire l'influence	`df['col'].clip(lower, upper)`
Transformer	Réduire l'asymétrie	`np.log1p(df['col'])`
Binning	Catégoriser	`pd.cut(df['col'], bins)`

Attention: Contexte Bancaire

```
Les outliers dans les données bancaires sont souvent RÉELS:
- Gros clients (high net worth individuals)
- Transactions exceptionnelles
- Événements de fraude

TOUJOURS investiguer avant de supprimer!
```

2.4 Transformation des Données

Transformation de Types

```python
# String vers numérique
df['montant'] = pd.to_numeric(df['montant'], errors='coerce')

# String vers datetime
df['date'] = pd.to_datetime(df['date'], format='%Y-%m-%d')

# Numérique vers catégorielle
df['score_cat'] = pd.cut(df['score'], bins=[0, 500, 700, 850], labels=['Low', 'Medium', 'High'])

# Catégorielle vers numérique (encoding)
df['type_encoded'] = df['type'].map({'A': 1, 'B': 2, 'C': 3})

# One-hot encoding
df_encoded = pd.get_dummies(df, columns=['type'])
```

Transformation Mathématiques

```python
# Log transform (pour distributions asymétriques)
df['montant_log'] = np.log1p(df['montant'])

# Square root
df['montant_sqrt'] = np.sqrt(df['montant'])

# Box-Cox (normalisation automatique)
from scipy import stats
df['montant_boxcox'], lambda_param = stats.boxcox(df['montant'] + 1)

# Standardisation (Z-score)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['montant_std'] = scaler.fit_transform(df[['montant']])

# Min-Max scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['montant_scaled'] = scaler.fit_transform(df[['montant']])
```

2.5 Feature Engineering

Création de Nouvelles Variables

```python
# À partir de dates
df['annee'] = df['date'].dt.year
df['mois'] = df['date'].dt.month
df['jour_semaine'] = df['date'].dt.dayofweek
df['is_weekend'] = df['jour_semaine'].isin([5, 6])
df['anciennete_jours'] = (pd.Timestamp.now() - df['date_inscription']).dt.days

# Agrégations
df['total_par_client'] = df.groupby('client_id')['montant'].transform('sum')
df['nb_transactions'] = df.groupby('client_id')['transaction_id'].transform('count')
df['moyenne_mensuelle'] = df.groupby(['client_id', 'mois'])['montant'].transform('mean')
```

```python
# Ratios
df['debt_to_income'] = df['dette_totale'] / df['revenu']
df['utilisation_credit'] = df['solde_utilise'] / df['limite_credit']

# Binning
df['tranche_age'] = pd.cut(df['age'], bins=[0, 25, 35, 50, 65, 100],
                           labels=['Jeune', 'Adulte', 'Mature', 'Senior', 'Retraité'])

# Flags
df['is_high_value'] = (df['montant'] > df['montant'].quantile(0.9)).astype(int)
df['is_nouveau_client'] = (df['anciennete_jours'] < 365).astype(int)
```

Features Bancaires Courantes

Feature	Calcul	Usage
DTI (Debt-to-Income)	dette/revenu	Capacité de remboursement
LTV (Loan-to-Value)	prêt/valeur_garantie	Risque de crédit
Utilisation	solde/limite	Comportement crédit
Récence	jours depuis dernière tx	Activité client
Fréquence	nb tx par période	Engagement
Montant moyen	total/nb_tx	Valeur client

2.6 Gestion des Doublons

```python
# Identifier les doublons
df.duplicated().sum()  # Nombre de doublons
df[df.duplicated()]    # Voir les doublons

# Doublons sur colonnes spécifiques
df[df.duplicated(subset=['client_id', 'date', 'montant'])]

# Supprimer les doublons
df_clean = df.drop_duplicates()
df_clean = df.drop_duplicates(subset=['client_id', 'date'], keep='last')

# Avant de supprimer, investiguer!
# Parfois les doublons sont légitimes (deux transactions identiques)
```

2.7 Validation et Qualité des Données

Règles de Validation

```python
# Assertions de base
assert df['montant'].min() >= 0, "Montants négatifs trouvés"
assert df['taux_interet'].between(0, 1).all(), "Taux hors limites"
assert df['date_fin'] >= df['date_debut']).all(), "Dates incohérentes"

# Validation conditionnelle
assert (df[df['statut'] == 'Actif']['solde'] != 0).all(), "Comptes actifs sans solde"
```

```python
# Intégrité référentielle
clients_prets = set(df_prets['client_id'])
clients_table = set(df_clients['client_id'])
orphelins = clients_prets - clients_table
assert len(orphelins) == 0, f"Clients orphelins: {orphelins}"
```

Profiling Automatique

```python
# Avec pandas-profiling (ydata-profiling)
from ydata_profiling import ProfileReport
profile = ProfileReport(df, title="EDA Report")
profile.to_file("eda_report.html")
```

2.8 Pipeline de Data Wrangling

Structure Recommandée

```python
def clean_data(df):
    """Pipeline complet de nettoyage"""

    # 1. Copie pour éviter les modifications in-place
    df = df.copy()

    # 2. Standardiser les noms de colonnes
    df.columns = df.columns.str.lower().str.replace(' ', '_')

    # 3. Convertir les types
    df['date'] = pd.to_datetime(df['date'])
    df['montant'] = pd.to_numeric(df['montant'], errors='coerce')

    # 4. Traiter les valeurs manquantes
    df['revenu'].fillna(df['revenu'].median(), inplace=True)
    df['secteur'].fillna('Non spécifié', inplace=True)

    # 5. Supprimer les doublons
    df = df.drop_duplicates()

    # 6. Traiter les outliers
    df['montant'] = df['montant'].clip(0, df['montant'].quantile(0.99))

    # 7. Créer les features
    df['anciennete'] = (pd.Timestamp.now() - df['date_inscription']).dt.days

    # 8. Valider
    assert df['montant'].min() >= 0

    return df

# Utilisation
df_clean = clean_data(df_raw)
```

Partie 3: Bonnes Pratiques

3.1 Principes Fondamentaux

1. **Reproductibilité:** Documenter chaque transformation
2. **Traçabilité:** Garder les données brutes intactes
3. **Validation:** Vérifier après chaque étape
4. **Itération:** L'EDA n'est pas linéaire, y revenir souvent

3.2 Erreurs Courantes à Éviter

Erreur	Conséquence	Solution
Supprimer sans investiguer	Perte d'information	Toujours analyser d'abord
Imputer aveuglément	Biais introduit	Comprendre le pattern de manquants
Ignorer le contexte métier	Mauvaises décisions	Impliquer les experts
Modifier les données brutes	Pas de retour arrière	Travailler sur des copies
Oublier de documenter	Non reproductible	Tenir un log des décisions

3.3 Checklist EDA Complète

```
Contexte business compris et documenté
Données chargées et structure vérifiée
Types de données corrects
Statistiques descriptives calculées
Distributions visualisées
Valeurs manquantes identifiées et traitées
Outliers détectés et gérés
Doublons vérifiés
Relations entre variables explorées
Features engineering réalisé
Validation finale effectuée
Insights documentés
Prochaines étapes définies
```

Partie 4: Applications Bancaires

4.1 EDA pour l'Analyse de Crédit

```python
# Variables clés à explorer
credit_vars = ['score_credit', 'revenu', 'dette', 'anciennete_emploi',
               'nb_credits', 'retards_paiement']

# Questions EDA spécifiques
# - Distribution des scores par segment
# - Corrélation score vs taux de défaut
# - Pattern des défauts par caractéristiques
# - Saisonnalité des demandes
```

4.2 EDA pour la Segmentation Client

```
# Variables RFM
df['recence'] = (date_ref - df['derniere_transaction']).dt.days
df['frequence'] = df.groupby('client_id')['transaction_id'].transform('count')
df['montant_total'] = df.groupby('client_id')['montant'].transform('sum')

# Exploration
# - Distribution de chaque métrique
# - Corrélations entre métriques
# - Identification des clusters naturels
```

4.3 EDA pour la Détection de Fraude

```
# Variables comportementales
# - Montant vs moyenne historique
# - Heure inhabituelle
# - Localisation inhabituelle
# - Fréquence de transactions

# Focus sur les outliers
# - Les fraudes sont SOUVENT des outliers
# - Ne pas supprimer sans investigation approfondie
```

Questions d'Entretien Fréquentes

1. **Quelle est votre approche pour commencer une analyse de données?** → Comprendre le contexte business, examiner la structure, explorer les distributions
2. **Comment gérez-vous les valeurs manquantes?** → Dépend du contexte: MCAR/MAR/MNAR, pourcentage, importance de la variable
3. **Quand supprimez-vous un outlier?** → Uniquement si c'est une erreur de données, jamais automatiquement
4. **Différence entre EDA et Data Wrangling?** → EDA = exploration et compréhension; Wrangling = nettoyage et préparation
5. **Comment validez-vous vos transformations?** → Assertions, comparaison avant/après, règles métier, review par pairs

---

Résumé

EDA: Les 5 Questions Essentielles

1. Quelle est la **structure** des données?
2. Quelle est la **distribution** de chaque variable?
3. Y a-t-il des **valeurs manquantes** ou **anomalies**?
4. Quelles sont les **relations** entre variables?
5. Quels sont les **insights** pour le business?

Data Wrangling: Les 5 Étapes Clés

1. **Nettoyer:** Manquants, erreurs, doublons
2. **Transformer:** Types, formats, échelles
3. **Enrichir:** Feature engineering
4. **Valider:** Règles métier, intégrité
5. **Documenter:** Chaque décision et transformation

Rappel final: L'EDA et le Data Wrangling représentent 60-80% du temps d'un projet data. C'est une étape critique qui détermine la qualité de toutes les analyses ultérieures.