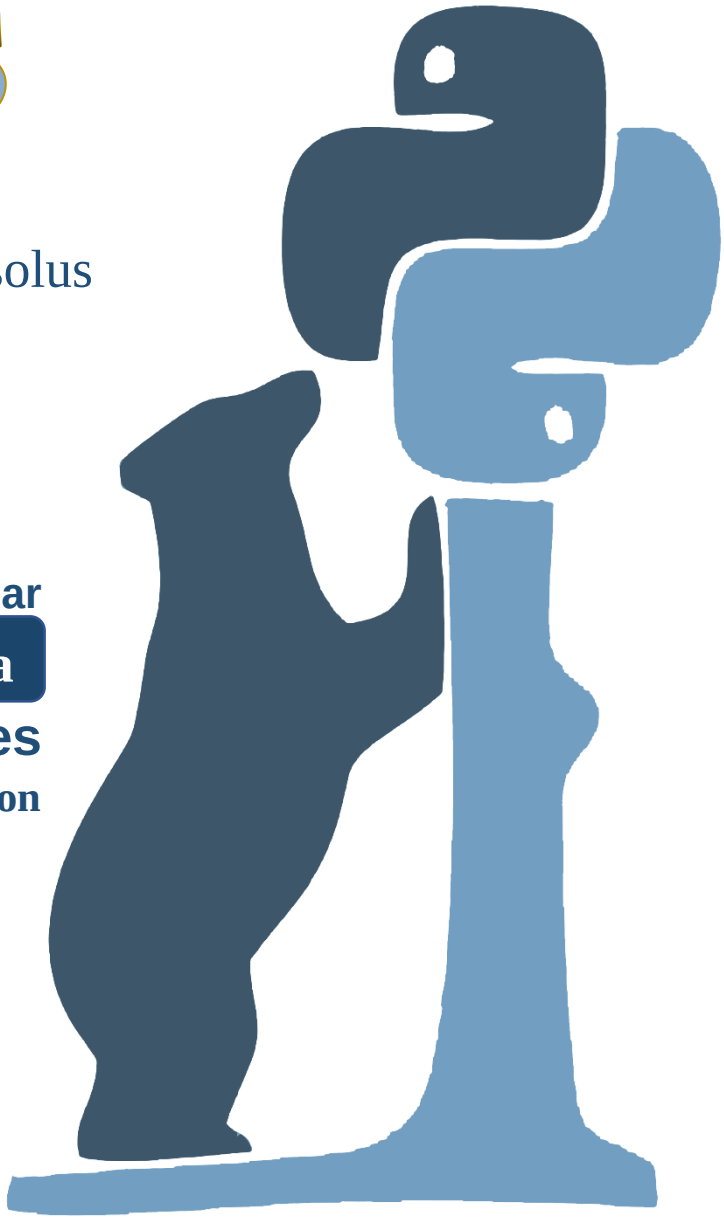
Pandas

Pour l'analyse de données :
Guide complet et exercices résolus



Par

Bernard MUTUALA Nkota

Analyste de données

Excel | SQL | Power BI | Python

Kinshasa - 2025

CHAPITRE I : MANIPULER ET EXPLORER LES DONNEES

1.1. Objectif

Apprendre à charger, lire et comprendre la structure d'un jeu de données (dataset) à l'aide de pandas.

1.2. Contexte professionnel

Cas réel :

Vous travaillez comme data analyst dans une entreprise de télécommunications appelée AfriTel RDC.
Le service marketing vous donne un fichier clients.csv contenant des informations sur les clients :

- ✓ Leur nom,
- ✓ Leur âge,
- ✓ Leur ville,
- ✓ Leur type d'abonnement,
- ✓ Et le montant de leur dernière facture.

Votre rôle est d'ouvrir le fichier, explorer son contenu, comprendre les colonnes, et vérifier s'il est bien structuré avant d'en faire une analyse.

Jeu de données simulé (exemple)

Fichier : clients.csv

Nom	Age	Ville	Abonnement	Facture ($)
Bernard	28	Kinshasa	Gold	45.5
Christelle	24	Lubumbashi	Silver	30.2
David	31	Goma	Gold	50.0
Sarah	22	Kinshasa	Bronze	25.1
Junior	27	Kisangani	Silver	35.3

1.3. Charger le fichier avec pandas

```
import pandas as pd
# Charger le fichier depuis le bureau
fichier = r"C:\Users\VotreNom\Desktop\clients.csv"
df = pd.read_csv(fichier)
# Afficher les 5 premières lignes
print(df.head())
```

Explication :

import pandas as pd : on importe la bibliothèque pandas et on lui donne le raccourci pd.
pd.read_csv() : permet de lire un fichier CSV.
fichier = r"C:\Users\VotreNom\Desktop\clients.csv" : **r** signifie "**raw string**", pour éviter les erreurs avec les barres obliques.
df.head() → affiche les 5 premières lignes du tableau.

Résultat attendu :

	Nom	Age	Ville	Abonnement	Facture ($)
0	Bernard	28	Kinshasa	Gold	45.5
1	Christelle	24	Lubumbashi	Silver	30.2
2	David	31	Goma	Gold	50.0
3	Sarah	22	Kinshasa	Bronze	25.1
4	Junior	27	Kisangani	Silver	35.3

1.4. Explorer la structure du tableau

```
# Dimensions du tableau
print("Dimensions :", df.shape)
# Nom des colonnes
print("Colonnes :", df.columns.tolist())
# Types de données
print("\nTypes de données :")
print(df.dtypes)
# Informations générales
print("\nInfos générales :")
print(df.info())
```

Explication :

df.shape : donne (nombre_lignes, nombre_colonnes)
df.columns.tolist() : liste les noms de colonnes
df.dtypes : affiche le type de chaque colonne (int64, float64, object, etc.)
df.info() : affiche les types, le nombre de valeurs non nulles et la mémoire utilisée.

Résultat attendu :

```
Dimensions : (5, 5)
Colonnes : ['Nom', 'Age', 'Ville', 'Abonnement', 'Facture ($)']
Types de données :
Nom           object
Age           int64
Ville         object
Abonnement    object
Facture ($)   float64
dtype: object
```

1.5. Voir un résumé statistique

```
print(df.describe())
```

Explication :

df.describe() : résume seulement les colonnes numériques (âge, facture, etc.).

Résultat attendu :

```
              Age        Facture ($)
count    5.000000       5.000000
mean    26.400000      37.220000
std      3.201562       9.339905
min     22.000000      25.100000
25%     24.000000      30.200000
50%     27.000000      35.300000
75%     28.000000      45.500000
max     31.000000      50.000000
Abonnement     object
Facture ($)   float64
dtype: object
```

Exemple de lecture :

- Moyenne d'âge = 26,4 ans
- Moyenne facture = 37,22 $
- Facture minimale = 25,1 $
- Age minimale = 22 ans

1.6. Voir toutes les données sous forme de tableau

```
print(df.to_string(index=False))
```

Ce code affiche tout le DataFrame proprement, sans masquer les lignes ni les indices.

1.7. Vérifier les valeurs manquantes

```
print(df.isna().sum())
```

Explication :

df.isna() : renvoie True pour chaque cellule vide.
.sum() : compte combien de valeurs manquent par colonne.

1.8. Interprétation professionnelle

Après avoir exécuté les codes précédents, vous pouvez déjà tirer quelques conclusions :

- ✓ *Les données sont complètes (aucune valeur manquante).*
- ✓ *Les types sont corrects (nombres, texte, etc.).*
- ✓ *Le fichier est prêt pour l'analyse.*

Cette étape (exploration) est toujours la première dans un projet data :

« On ne nettoie ni n'analyse sans d'abord comprendre la structure des données. »

1.9. Résumé des notions vues :

Fonction	Utilité
pd.read_csv()	Charger un fichier CSV
df.head()	Voir les premières lignes
df.shape	Dimensions du tableau
df.columns	Noms des colonnes
df.dtypes	Types de données
df.info()	Structure générale
df.describe()	Statistiques de base
df.isna().sum()	Valeurs manquantes

CHAPITRE II : SELECTIONNER ET FILTRER LES DONNEES

2.1. Objectif

Apprendre à extraire, filtrer et sélectionner les lignes et colonnes précises d'un DataFrame.

C'est une compétence centrale pour isoler les données pertinentes dans une analyse.

2.2. Contexte professionnel

Vous continuez votre travail chez AfriTel RDC.
Le département de marketing veut :

- ✓ Voir seulement certaines colonnes (ex. Nom, Ville, Facture),
- ✓ Analyser les clients d'une seule ville,
- ✓ Isoler les clients dont la facture dépasse un certain montant et
- ✓ Accéder à une valeur précise (par exemple, la facture du premier client).

2.3. Jeu de données (même que dans le niveau 1)

Nom	Age	Ville	Abonnement	Facture ($)
Bernard	28	Kinshasa	Gold	45.5
Christelle	24	Lubumbashi	Silver	30.2
David	31	Goma	Gold	50.0
Sarah	22	Kinshasa	Bronze	25.1
Junior	27	Kisangani	Silver	35.3

2.4. Charger le fichier

```
import pandas as pd
df = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients.csv")
print(df.head())
```

2.5. Sélectionner des colonnes

2.5.1. Une seule colonne

```
print(df['Nom'])
```

Explication :

Les crochets [ ] indiquent une sélection.
Cela renvoie une série pandas (Series).

Résultat attendu :

```
0     Bernard
1     Christelle
2     David
3     Sarah
4     Junior
```

1.5.2. Plusieurs colonnes

```
print(df[['Nom', 'Facture ($)']])
```

Explication :

Double crochets [[ ]] = plusieurs colonnes.
Résultat : un DataFrame.

Résultat attendu :

Nom	Facture ($)
Bernard	45.5
Christelle	30.2
David	50.0
Sarah	25.1
Junior	35.3

2.5. Sélectionner des lignes

2.5.1. Par position (*iloc*)

```
print(df.iloc[0])  # première ligne
```

Explication :

iloc = indexation par position numérique (0 = première ligne, 1 = deuxième…).

Résultat attendu :

```
Nom             Bernard
Age                  28
Ville          Kinshasa
Abonnement         Gold
Facture ($)        45.5
Name: 0, dtype: object
```

2.5.2. Par étiquette (**loc**)

```
print(df.loc[3, 'Ville'])
```

Explication :

loc = indexation par étiquette (nom de colonne).
Ici, on prend la ligne 3 (Sarah) et la colonne "Ville".

Résultat attendu :

Kinshasa

2.6. Filtrer selon une condition

2.6.1. Exemples simples

```
# Clients dont la facture dépasse 35$
filtre = df[df['Facture ($)'] > 35]
print(filtre)
```

Résultat attendu :

Nom	Age	Ville	Abonnement	Facture ($)
Bernard	28	Kinshasa	Gold	45.5
David	31	Goma	Gold	50.0
Junior	27	Kisangani	Silver	35.3

2.6.2. Filtrer selon du texte

```
# Clients de Kinshasa
print(df[df['Ville'] == 'Kinshasa'])
```

Résultat attendu :

Nom	Age	Ville	Abonnement	Facture ($)
Bernard	28	Kinshasa	Gold	45.5
Sarah	22	Kinshasa	Bronze	25.1

2.6.3. Filtre combiné (avec ET et OU)

```
# Clients de Kinshasa ayant une facture > 30$
print(df[(df['Ville'] == 'Kinshasa') & (df['Facture ($)'] > 30)])
```

Explication :

& = ET
| = OU
Chaque condition est mise entre parenthèses ( ).

Résultat attendu :

Nom	Age	Ville	Abonnement	Facture ($)
Bernard	28	Kinshasa	Gold	45.5

2.7. Trier les données

Trier par montant de facture

```
df_sorted = df.sort_values(by='Facture ($)', ascending=False)
print(df_sorted)
```

Explication :

ascending=False : tri du plus grand au plus petit.

Vous pouvez trier par plusieurs colonnes :

by=['Ville', 'Facture ($)'].

Résultat attendu :

Nom	Age	Ville	Abonnement	Facture ($)
David	31	Goma	Gold	50.0
Bernard	28	Kinshasa	Gold	45.5
Junior	27	Kisangani	Silver	35.3
Christelle	24	Lubumbashi	Silver	30.2
Sarah	22	Kinshasa	Bronze	25.1

2.8. Cas professionnel résumé

Vous voulez extraire :

- Les clients de Kinshasa,
- Avec une facture > 30$ et
- Afficher seulement Nom, Ville, Facture.

Code complet :

```
import pandas as pd
df = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients.csv")
resultat = df[(df['Ville'] == 'Kinshasa') & (df['Facture ($)'] > 30)][['Nom', 'Ville', 'Facture ($)']]
print(resultat)
```

Résultat final attendu :

Nom	Ville	Facture ($)
Bernard	Kinshasa	45.5

2.9. Résumé des notions clés

Syntaxe	Description
df['colonne']	Sélectionner une colonne
df[['col1','col2']]	Sélectionner plusieurs colonnes
df.iloc[i]	Sélectionner une ligne par position
df.loc[i, 'col']	Sélectionner une valeur précise
df[df['col'] > x]	Filtrer selon une condition
df.sort_values(by='col')	Trier les lignes

CHAPITRE III : NETTOYER LES DONNEES AVEC PANDAS

3.1. Objectif

Apprendre à identifier, corriger, et nettoyer les problèmes dans un DataFrame : valeurs manquantes, doublons, fautes de frappe, types de données incorrects, formatage incohérent, etc.

3.2. Contexte professionnel

Vous êtes toujours analyste chez AfriTel RDC. Vous recevez cette fois un fichier ***clients_bruts.csv*** contenant des erreurs typiques dans les données collectées via un formulaire en ligne. Votre chef vous demande de le nettoyer avant de l'envoyer à Power BI pour analyse.

3.2.1. Jeu de données brut (avant nettoyage)

Nom	Age	Ville	Abonnement	Facture ($)
Bernard	28	kinshasa	Gold	45.5
Christelle	NaN	Lubumbashi	Silver	30.2
David	31	Goma	gold	NaN
Sarah	22	Kinshasa	Bronze	25.1
Junior	27	Kisangani	Silver	35.3
Bernard	28	kinshasa	Gold	45.5

3.2.2. Problèmes identifiés

1. Valeurs manquantes (NaN dans Age et Facture)
2. Doublons (Bernard apparaît deux fois)
3. Incohérence de casse (kinshasa vs Kinshasa, gold vs Gold)
4. Type de données à vérifier

3.3. Identifier les valeurs manquantes

```
print("Valeurs manquantes par colonne :")
print(df.isna().sum())
```

Résultat attendu :

```
Nom           0
Age           1
Ville         0
Abonnement    0
Facture ($)   1
dtype: int64
```

Explication :

df.isna() : renvoie True pour chaque cellule vide.
.sum() : compte combien de valeurs manquent dans chaque colonne.

3.4. Traiter les valeurs manquantes

Option 1 : Supprimer les lignes incomplètes

```
df_clean = df.dropna()
```

Option 2 : Remplacer par une valeur (ex. moyenne)

```
df['Age'].fillna(df['Age'].mean(), inplace=True)
df['Facture ($)'].fillna(df['Facture ($)'].mean(), inplace=True)
```

Explication :

fillna() : remplace les valeurs manquantes.
inplace=True : applique directement la modification dans le DataFrame.

3.5. Supprimer les doublons

```
print("Nombre de doublons :", df.duplicated().sum())
df.drop_duplicates(inplace=True)
```

Résultat :

```
Nombre de doublons : 1
```

Explication :

df.duplicated() : renvoie True pour les lignes répétées.
drop_duplicates() : supprime les doublons.

3.6. Corriger les incohérences de casse (majuscules/minuscules)

```
df['Ville'] = df['Ville'].str.title()          # Première lettre en majuscule
df['Abonnement'] = df['Abonnement'].str.capitalize()  # Gold, Silver, Bronze
```

Avant:

kinshasa, gold

Après:

Kinshasa, Gold

3.7. Vérifier les types de données

```
print(df.dtypes)
```

Si vous voyez par exemple :

Age	float64
Facture ($)	object

et que “Facture” est texte au lieu de nombre, vous corrigez :

```
df['Facture ($)'] = df['Facture ($)'].astype(float)
```

Explication :

astype() : convertit le type d’une colonne.

3.8. Renommer les colonnes

```
df.rename(columns={'Facture ($)': 'Facture_USD'}, inplace=True)
```

Pourquoi ?

Donner des noms plus simples ou plus uniformes aide à éviter les erreurs dans le code et à mieux comprendre les analyses.

3.9. Nettoyer les espaces ou caractères invisibles

```
df['Nom'] = df['Nom'].str.strip()  # supprime les espaces au début/fin
```

3.10. Vérification finale

```
print(df)
print("\nValeurs manquantes après nettoyage :")
print(df.isna().sum())
```

Résultat final :

Nom	Age	Ville	Abonnement	Facture_USD
Bernard	28	Kinshasa	Gold	45.5
Christelle	26.4	Lubumbashi	Silver	30.2
David	31	Goma	Gold	39.525
Sarah	22	Kinshasa	Bronze	25.1
Junior	27	Kisangani	Silver	35.3

- ✓ Les valeurs manquantes ont été remplacées par la moyenne.
- ✓ Les doublons ont été supprimés.
- ✓ Les majuscules sont uniformisées.

3.11. Cas professionnel complet

Code global résumé :

```
import pandas as pd

# 1 Charger les données
df = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients_bruts.csv")
# 2 Remplacer les valeurs manquantes par la moyenne
df['Age'].fillna(df['Age'].mean(), inplace=True)
df['Facture ($)'].fillna(df['Facture ($)'].mean(), inplace=True)
# 3 Supprimer les doublons
df.drop_duplicates(inplace=True)
# 4 Uniformiser la casse
df['Ville'] = df['Ville'].str.title()
df['Abonnement'] = df['Abonnement'].str.capitalize()
# 5 Renommer la colonne
df.rename(columns={'Facture ($)': 'Facture_USD'}, inplace=True)
# 6 Supprimer les espaces
df['Nom'] = df['Nom'].str.strip()
# 7 Vérification finale
print(df)
```

3.12. Résumé des fonctions vues

Fonction	Rôle
df.isna()	Identifier les valeurs manquantes
df.fillna()	Remplacer les valeurs manquantes
df.dropna()	Supprimer les lignes incomplètes
df.duplicated()	Détecter les doublons
df.drop_duplicates()	Supprimer les doublons
df.str.title()	Mettre la première lettre en majuscule
df.astype()	Modifier le type de donnée
df.rename()	Renommer les colonnes
df.str.strip()	Supprimer les espaces inutiles

3.13. Interprétation professionnelle

Vous pouvez maintenant :

- ✓ Nettoyer les données reçues des formulaires clients
- ✓ Uniformiser les textes pour éviter des erreurs dans les graphiques
- ✓ Corriger les types et supprimer les doublons
- ✓ Préparer un dataset fiable et propre pour l'analyse Power BI, Excel ou SQL

CHAPITRE IV : TRANSFORMER LES DONNEES

4.1. Objectif

Apprendre à modifier, enrichir ou combiner des colonnes dans un DataFrame pour créer de nouvelles informations utiles à l'analyse.

4.2. Contexte professionnel

Toujours chez AfriTel RDC, le service marketing souhaite analyser les revenus annuels des clients et catégoriser leur niveau de dépense en fonction de leur facture mensuelle.

Le dataset utilisé : clients_clean.csv

Nom	Age	Ville	Abonnement	Facture_USD
Bernard	28	Kinshasa	Gold	45.5
Christelle	26	Lubumbashi	Silver	30.2
David	31	Goma	Gold	39.5
Sarah	22	Kinshasa	Bronze	25.1
Junior	27	Kisangani	Silver	35.3

4.3. Charger les données

```
import pandas as pd
df = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients_clean.csv")
print(df.head())
```

4.4. Créer de nouvelles colonnes

4.4.1. Calculer le revenu annuel

```
df['Revenu_annuel'] = df['Facture_USD'] * 12
print(df[['Nom', 'Facture_USD', 'Revenu_annuel']])
```

Explication :

On multiplie la facture mensuelle par 12 pour obtenir le revenu annuel.
Résultat : nouvelle colonne Revenu_annuel.

Résultat attendu :

```
Nom         Facture_USD  Revenu_annuel
Bernard     45.5         546.0
Christelle  30.2         362.4
David       39.5         474.0
Sarah       25.1         301.2
Junior      35.3         423.6
```

4.4.2. Catégoriser les clients selon le revenu

```
import numpy as np
df['Categorie_depense'] = np.where(df['Revenu_annuel'] > 450, 'Haut', 'Bas')
print(df[['Nom', 'Revenu_annuel', 'Categorie_depense']])
```

Explication :

np.where(condition, valeur_si_vrai, valeur_si_faux) : permet de créer une nouvelle colonne basée sur une condition.

Ici, si le revenu annuel > 450 : catégorie "Haut", sinon "Bas".

Résultat attendu :

Nom	Revenu_annuel	Categorie_depense
Bernard	546.0	Haut
Christelle	362.4	Bas
David	474.0	Haut
Sarah	301.2	Bas
Junior	423.6	Bas

4.4.3. Combiner deux colonnes pour créer une nouvelle info

```
df['Nom_Complet'] = df['Nom'] + " (" + df['Ville'] + ")"
print(df[['Nom', 'Ville', 'Nom_Complet']])
```

Explication :

On concatène le nom et la ville pour créer un identifiant unique. Utile pour les graphiques ou rapports.

Résultat attendu :

Nom	Ville	Nom_Complet
Bernard	Kinshasa	Bernard (Kinshasa)
Christelle	Lubumbashi	Christelle (Lubumbashi)
David	Goma	David (Goma)
Sarah	Kinshasa	Sarah (Kinshasa)
Junior	Kisangani	Junior (Kisangani)

4.4.4. Transformer le type d'une colonne

```
df['Age'] = df['Age'].astype(float)
print(df.dtypes)
```

Explication :

astype(float) : permet de convertir les entiers en nombres décimaux, si nécessaire pour calculs ou analyses.

4.5. Cas pratique professionnel

Objectifs complets :

- ✓ Calculer le revenu annuel.
- ✓ Catégoriser les clients selon leur dépense.
- ✓ Créer un identifiant combiné Nom + Ville.
- ✓ Vérifier les types pour l'analyse.

Code complet

```
import pandas as pd
import numpy as np
df = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients_clean.csv")
# Revenu annuel
df['Revenu_annuel'] = df['Facture_USD'] * 12
# Catégorie de dépense
df['Categorie_depense'] = np.where(df['Revenu_annuel'] > 450, 'Haut', 'Bas')
# Identifiant Nom + Ville
df['Nom_Complet'] = df['Nom'] + " (" + df['Ville'] + ")"
# Conversion type
df['Age'] = df['Age'].astype(float)
print(df)
```

4.6. Résumé des fonctions vues

Fonction / méthode	Utilité
df['Nouvelle_col'] = …	Créer une colonne
np.where(condition, valeur_si_vrai, valeur_si_faux)	Catégoriser selon condition
+ (concaténation de colonnes)	Créer des identifiants uniques
astype()	Convertir type de données

4.7. Interprétation professionnelle

- ✓ Après cette transformation, votre dataset est prêt pour l'analyse.
- ✓ Vous pouvez maintenant créer des graphiques, résumer les données, ou l'exporter pour Power BI.
- ✓ La transformation est clé pour enrichir vos analyses avec des indicateurs pertinents.

CHAPITRE V : REGROUPER ET RESUMER LES DONNEES

5.1. Objectif

Apprendre à regrouper les données selon une ou plusieurs colonnes et calculer des agrégations utiles comme la somme, la moyenne, le compte, le maximum, etc. C'est l'équivalent des GROUP BY en SQL.

5.2. Contexte professionnel

Chez AfriTel RDC, le service marketing souhaite :

- ✓ Connaitre le revenu moyen par ville.
- ✓ Compter combien de clients ont chaque type d'abonnement.
- ✓ Identifier le client le plus dépensier par ville.

Dataset : clients_transforme.csv

Nom	Age	Ville	Abon-nement	Facture USD	Revenu_ annuel	Categorie_ depense
Bernard	28	Kinshasa	Gold	45.5	546.0	Haut
Christelle	26	Lubumbashi	Silver	30.2	362.4	Bas
David	31	Goma	Gold	39.5	474.0	Haut
Sarah	22	Kinshasa	Bronze	25.1	301.2	Bas
Junior	27	Kisangani	Silver	35.3	423.6	Bas

5.3. Charger le fichier

```
import pandas as pd
df =pd.read_csv(r"C:\Users\VotreNom\Desktop\clients_transforme.csv")
print(df.head())
```

5.4. Regrouper par ville et calculer la moyenne du revenu annuel

```
moyenne_par_ville = df.groupby('Ville')['Revenu_annuel'].mean()
print(moyenne_par_ville)
```

Explication :

groupby('Ville') : regroupe les données par ville.
['Revenu_annuel'].mean() : calcule la moyenne pour chaque groupe.

Résultat attendu :

```
Ville
Goma          474.0
Kinshasa      423.6
Kisangani     423.6
Lubumbashi    362.4
Name: Revenu_annuel, dtype: float64
```

5.5. Compter le nombre de clients par abonnement

```
compte_abonnement = df.groupby('Abonnement').size()
print(compte_abonnement)
```

Explication :

size() : compte le nombre de lignes pour chaque groupe.

Résultat attendu :

```
Abonnement
Bronze    1
Gold      2
Silver    2
dtype: int64
```

5.6. Somme, maximum, minimum par groupe

```
somme_max_min = df.groupby('Ville')['Revenu_annuel'].agg(['sum', 'max', 'min'])
print(somme_max_min)
```

Explication :

.agg(['sum', 'max', 'min']) : calcule plusieurs agrégations à la fois.

Résultat attendu :

Ville	sum	max	min
Goma	474.0	474.0	474.0
Kinshasa	847.2	546.0	301.2
Kisangani	423.6	423.6	423.6
Lubumbashi	362.4	362.4	362.4

5.7. Regrouper sur plusieurs colonnes

Exemple : moyenne par ville et catégorie de dépense

```
moyenne_ville_categorie = df.groupby(['Ville',
'Categorie_depense'])['Revenu_annuel'].mean()
print(moyenne_ville_categorie)
```

Résultat attendu :

Ville	Categorie_depense	
Goma	Haut	474.0
Kinshasa	Bas	301.2
	Haut	546.0
Kisangani	Bas	423.6
Lubumbashi	Bas	362.4

Name: Revenu_annuel, dtype: float64

5.8. Réinitialiser l'index pour un DataFrame propre

```
df_grouped = df.groupby('Ville')['Revenu_annuel'].mean().reset_index()
print(df_grouped)
```

Résultat attendu :

Ville	Revenu_annuel
Goma	474.0
Kinshasa	423.6
Kisangani	423.6
Lubumbashi	362.4

5.9. Cas pratique complet

Calculer ou compter :

1. Moyenne du revenu annuel par ville
2. Nombre de clients par type d'abonnement
3. Somme, max, min par ville

Code global :

```
import pandas as pd
df =pd.read_csv(r"C:\Users\VotreNom\Desktop\clients_transforme.csv")
#1 Moyenne du revenu annuel par ville
moyenne_par_ville = df.groupby('Ville')['Revenu_annuel'].mean()
print("Moyenne par ville :\n", moyenne_par_ville)
#2 Nombre de clients par abonnement
compte_abonnement = df.groupby('Abonnement').size()
print("\nNombre de clients par abonnement :\n", compte_abonnement)
#3 Somme, max, min par ville
somme_max_min = df.groupby('Ville')['Revenu_annuel'].agg(['sum', 'max', 'min'])
print("\nSomme, max et min par ville :\n", somme_max_min)
```

5.10. Résumé des fonctions clés

Fonction	Rôle
groupby('colonne')	Regrouper les données par colonne
['col'].mean()	Moyenne par groupe
.size()	Compter les lignes par groupe
.agg(['sum','max','min'])	Calculer plusieurs agrégations
reset_index()	Transformer un Series groupé en DataFrame

5.11. Interprétation professionnelle

Vous pouvez maintenant résumer rapidement les données selon des catégories ou groupes. Vous pouvez créer des rapports utiles pour le marketing :

- ✓ Revenu moyen par ville,
- ✓ Clients par abonnement, ou
- ✓ Identifier les clients les plus dépensiers.

Ces techniques sont la base pour Power BI, Excel, ou SQL.

Chapitre VI : Fusionner et Joindre les Tables

6.1. Objectif

Apprendre à combiner plusieurs DataFrames pour enrichir l'analyse. C'est essentiel lorsque vous avez des données réparties sur différents fichiers ou sources, par exemple : clients, abonnements, paiements.

6.2. Contexte professionnel

Toujours chez AfriTel RDC, vous avez deux fichiers :

1. clients.csv : informations générales sur les clients

ClientID	Nom	Ville	Abonnement
1	Bernard	Kinshasa	Gold
2	Christelle	Lubumbashi	Silver
3	David	Goma	Gold
4	Sarah	Kinshasa	Bronze
5	Junior	Kisangani	Silver

2. factures.csv : informations sur les factures

ClientID	Facture_USD	Date
1	45.5	01/11/2025
2	30.2	01/11/2025
3	39.5	01/11/2025
4	25.1	01/11/2025
5	35.3	01/11/2025

Objectif : avoir un seul DataFrame avec toutes les infos des clients et leurs factures.

6.3. Charger les fichiers

```
import pandas as pd
clients = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients.csv")
factures = pd.read_csv(r"C:\Users\VotreNom\Desktop\factures.csv")
print(clients)
print(factures)
```

6.4. Fusionner les DataFrames (merge)

```
df_complet = pd.merge(clients, factures, on='ClientID', how='inner')
print(df_complet)
```

Explication :

on='ClientID' : fusion sur la colonne commune ClientID.
how='inner' : ne garde que les clients présents dans les deux tables.

Résultat attendu :

ClientID	Nom	Ville	Abonnement	Facture_USD	Date
1	Bernard	Kinshasa	Gold	45.5	2025-11-01
2	Christelle	Lubumbashi	Silver	30.2	2025-11-01
3	David	Goma	Gold	39.5	2025-11-01
4	Sarah	Kinshasa	Bronze	25.1	2025-11-01
5	Junior	Kisangani	Silver	35.3	2025-11-01

6.5. Types de jointures

Type	Description
inner	Garde seulement les valeurs présentes dans les deux tables
left	Garde tous les clients de la table de gauche, même si pas de facture
right	Garde tous les clients de la table de droite, même si pas de client correspondant
outer	Garde tous les clients et toutes les factures, remplissant NaN si aucune correspondance

***Exemple** : jointure gauche*

```
df_left = pd.merge(clients, factures, on='ClientID', how='left')
print(df_left)
```

Comme résultat, Tous les clients apparaissent même s'ils n'ont pas de facture (NaN dans Facture_USD si pas de facture).

6.6. Fusion sur plusieurs colonnes

Si les tables ont plusieurs colonnes clés, vous pouvez fusionner sur toutes :

```
# Par exemple, ClientID et Date
df_multi = pd.merge(clients, factures, on=['ClientID', 'Date'], how='inner')
```

6.7. Ajouter une colonne depuis une autre table (map)

Si vous voulez ajouter juste une colonne sans faire une fusion complète :

```
# Créer un dictionnaire ClientID -> Ville
ville_dict = clients.set_index('ClientID')['Ville'].to_dict()
# Ajouter la colonne Ville dans factures
factures['Ville'] = factures['ClientID'].map(ville_dict)
print(factures)
```

Explication :
map() : permet de mapper une clé vers une valeur depuis une autre table.

6.8. Cas pratique complet

Objectifs :

- ✓ Fusionner clients et factures pour obtenir toutes les infos dans un seul DataFrame.
- ✓ Ajouter une colonne supplémentaire Categorie_depense basée sur Facture_USD.

Code

```
import pandas as pd
import numpy as np
clients = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients.csv")
factures = pd.read_csv(r"C:\Users\VotreNom\Desktop\factures.csv")
# Fusionner
df = pd.merge(clients, factures, on='ClientID', how='inner')
# Ajouter catégorie de dépense
df['Categorie_depense'] = np.where(df['Facture_USD'] > 40, 'Haut', 'Bas')
print(df)
```

Résultat attendu :

ClientID	Nom	Ville	Abonnement	Facture_USD	Date	Categorie_depense
1	Bernard	Kinshasa	Gold	45.5	2025-11-01	Haut
2	Christelle	Lubumbashi	Silver	30.2	2025-11-01	Bas
3	David	Goma	Gold	39.5	2025-11-01	Bas
4	Sarah	Kinshasa	Bronze	25.1	2025-11-01	Bas
5	Junior	Kisangani	Silver	35.3	2025-11-01	Bas

6.9. Résumé des fonctions clés

Fonction / méthode	Utilité
pd.merge(df1, df2, on='col')	Fusionner deux DataFrames
how='inner' / 'left' / 'right' / 'outer'	Type de jointure
map()	Ajouter une colonne basée sur une clé
set_index()	Préparer le mapping pour map()

6.10. Interprétation professionnelle

- ✓ Vous pouvez regrouper plusieurs sources dans un seul dataset pour des analyses plus riches.
- ✓ C'est la base pour rapports financiers, analyses marketing, ou tableaux de bord Power BI.
- ✓ Les jointures correctes garantissent que toutes les données sont alignées sans perte d'information.

CHAPITRE VII : PIVOT TABLES ET TABLEAUX CROISES

7.1. Objectif

Apprendre à résumer et analyser rapidement les données groupées à la manière des tableaux croisés dynamiques Excel, mais directement dans pandas.

7.2. Contexte professionnel

Toujours chez AfriTel RDC, le service marketing veut analyser :

- ✓ Le revenu total et moyen par ville et par type d'abonnement.
- ✓ Le nombre de clients par catégorie de dépense et par ville.

Dataset : clients_factures.csv

ClientID	Nom	Ville	Abon-nement	Facture _USD	Revenu_ annuel
1	Bernard	Kinshasa	Gold	45.5	546.0
2	Christelle	Lubumbashi	Silver	30.2	362.4
3	David	Goma	Gold	39.5	474.0
4	Sarah	Kinshasa	Bronze	25.1	301.2
5	Junior	Kisangani	Silver	35.3	423.6

7.3. Charger le fichier

```
import pandas as pd
df = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients_factures.csv")
print(df.head())
```

7.4. Créer un tableau croisé simple (pivot_table)

Exemple : revenu moyen par ville

```
pivot_ville = df.pivot_table(values='Revenu_annuel', index='Ville', aggfunc='mean')
print(pivot_ville)
```

Résultat attendu :

```
Ville          Revenu_annuel
Goma           474.0
Kinshasa       423.0
Kisangani      423.6
Lubumbashi     362.4
```

Explication :

values : colonne à résumer
index : ligne du pivot
aggfunc='mean' : calcul de la moyenne (on peut utiliser sum, max, min, etc.)

7.5. Pivot avec plusieurs colonnes et plusieurs fonctions

Exemple : revenu total et moyen par ville et abonnement

```
pivot_multi = df.pivot_table(
    values='Revenu_annuel',
    index='Ville',
    columns='Abonnement',
    aggfunc=['sum', 'mean'],
    fill_value=0
)
print(pivot_multi)
```

Explication :

columns='Abonnement' : crée des colonnes pour chaque type d'abonnement
aggfunc=['sum','mean'] : deux fonctions d'agrégation
fill_value=0 : remplace les valeurs manquantes par 0

Résultat attendu :

sum				mean		
Abonnement	Bronze	Gold	Silver	Bronze	Gold	Silver
Ville						
Goma	0	474	0	0	474	0
Kinshasa	301	546	0	301	546	0
Kisangani	0	0	423	0	0	423
Lubumbashi	0	0	362	0	0	362

7.6. Utiliser aggfunc personnalisée

```
pivot_custom = df.pivot_table(
    values='Revenu_annuel',
    index='Ville',
    aggfunc=lambda x: x.max() - x.min()  # étendue des revenus
)
print(pivot_custom)
```

Explication:

lambda x: x.max() - x.min() : calcule l'écart entre le revenu le plus élevé et le plus faible par ville.

7.7. Tableaux croisés simples avec crosstab

Exemple : compter les clients par ville et catégorie de dépense

```
crosstab = pd.crosstab(df['Ville'], df['Categorie_depense'])
print(crosstab)
```

Résultat attendu :

Categorie_depense	Bas	Haut
Goma	0	1
Kinshasa	1	1
Kisangani	1	0
Lubumbashi	1	0

Explication :

pd.crosstab(index, columns) : crée un tableau croisé avec les comptes par défaut. C'est très pratique pour analyser la répartition des clients.

7.8. Cas pratique complet

Objectif :

Calculer le revenu total et moyen par ville et abonnement et compter les clients par catégorie de dépense et ville

```
import pandas as pd
df = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients_factures.csv")
# Pivot Table : revenu total et moyen par ville et abonnement
pivot = df.pivot_table(
    values='Revenu_annuel',
    index='Ville',
    columns='Abonnement',
    aggfunc=['sum', 'mean'],
    fill_value=0
)
print("Pivot Table :\n", pivot)
# Crosstab : nombre de clients par ville et catégorie de dépense
crosstab = pd.crosstab(df['Ville'], df['Categorie_depense'])
print("\nCrosstab :\n", crosstab)
```

7.9. Résumé des fonctions clés

Fonction / méthode	Utilité
pivot_table()	Résumer les données avec lignes et colonnes
aggfunc	Fonction d'agrégation (sum, mean, min, max, lambda)
fill_value	Remplacer les valeurs manquantes
pd.crosstab()	Compter la fréquence des valeurs par catégorie

7.10. Interprétation professionnelle

Vous pouvez :

- ✓ Résumer les revenus par ville et type d'abonnement.
- ✓ Identifier rapidement les catégories de clients et
- ✓ Préparer des indicateurs clés pour le reporting ou Power BI.

Chapitre VIII : Manipuler les Dates et Heures

8.1. Objectif :

Apprendre à travailler avec les dates et heures pour analyser des séries temporelles, calculer des durées, extraire des composants (année, mois, jour) et filtrer par périodes.

8.2. Contexte professionnel

Chez AfriTel RDC, le service marketing souhaite :
1. Analyser le revenu mensuel des clients.
2. Identifier les clients ayant plusieurs factures sur un même mois.
3. Calculer le nombre de jours depuis la dernière facture.

Dataset : factures_dates.csv

ClientID	Nom	Facture_USD	Date
1	Bernard	45.5	15/01/2025
2	Christelle	30.2	10/02/2025
3	David	39.5	20/01/2025
4	Sarah	25.1	05/03/2025
5	Junior	35.3	28/02/2025

8.3. Charger le fichier et convertir les dates

```
import pandas as pd
df = pd.read_csv(r"C:\Users\VotreNom\Desktop\factures_dates.csv")
# Convertir la colonne Date en datetime
df['Date'] = pd.to_datetime(df['Date'])
print(df.dtypes)
print(df.head())
```

Explication :

pd.to_datetime() : transforme une colonne texte en type datetime, ce qui permet toutes les manipulations temporelles.

8.4. Extraire les composants de la date

```
# Extraire année, mois, jour, jour de la semaine
df['Annee'] = df['Date'].dt.year
df['Mois'] = df['Date'].dt.month
df['Jour'] = df['Date'].dt.day
df['Jour_semaine'] = df['Date'].dt.day_name()
print(df)
```

Résultat attendu :

ClientID	Nom	Facture_USD	Date	Annee	Mois	Jour	Jour_semaine
1	Bernard	45.5	2025-01-15	2025	1	15	Wednesday
2	Christelle	30.2	2025-02-10	2025	2	10	Monday
3	David	39.5	2025-01-20	2025	1	20	Monday
4	Sarah	25.1	2025-03-05	2025	3	5	Wednesday
5	Junior	35.3	2025-02-28	2025	2	28	Friday

8.5. Filtrer les données par période

Exemple : toutes les factures de février 2025

```
df_fevrier = df[(df['Date'] >= '2025-02-01') & (df['Date'] <= '2025-02-28')]
print(df_fevrier)
```

8.6. Calculer la durée entre deux dates

```
# Supposons une colonne Date_facture_precedente
df['Date_precedente'] = pd.to_datetime(['2024-12-20','2025-01-10','2025-01-05','2025-02-25','2025-01-30'])
# Durée en jours entre deux factures
df['Jours_depuis_derniere'] = (df['Date'] - df['Date_precedente']).dt.days
print(df[['Nom','Date','Date_precedente','Jours_depuis_derniere']])
```

Résultat attendu :

Nom	Date	Date_precedente	Jours_depuis_derniere
Bernard	2025-01-15	2024-12-20	26
Christelle	2025-02-10	2025-01-10	31
David	2025-01-20	2025-01-05	15
Sarah	2025-03-05	2025-02-25	8
Junior	2025-02-28	2025-01-30	29

8.7. Grouper par mois et calculer le revenu total

```
revenu_mensuel = df.groupby(df['Date'].dt.to_period('M'))['Facture_USD'].sum()
print(revenu_mensuel)
```

Résultat attendu :

```
Date
2025-01    85.0
2025-02    65.5
2025-03    25.1
Freq: M, Name: Facture_USD, dtype: float64
```

Explication :

dt.to_period('M') : regroupe par mois, ignorants les jours précis.
On peut faire la même chose pour Y (année), W (semaine), etc.

8.8. Cas pratique complet

Objectifs :

- ✓ Convertir les dates
- ✓ Extraire année et mois
- ✓ Filtrer février 2025
- ✓ Calculer le nombre de jours depuis la dernière facture
- ✓ Revenu mensuel total

Code :

```
import pandas as pd
df = pd.read_csv(r"C:\Users\VotreNom\Desktop\factures_dates.csv")
df['Date'] = pd.to_datetime(df['Date'])
# Ajouter composants date
df['Annee'] = df['Date'].dt.year
df['Mois'] = df['Date'].dt.month
# Ajouter une date précédente fictive
df['Date_precedente'] = pd.to_datetime(['2024-12-20','2025-01-10','2025-01-05','2025-02-25','2025-01-30'])
df['Jours_depuis_derniere'] = (df['Date'] - df['Date_precedente']).dt.days
# Filtrer février 2025
df_fevrier = df[(df['Mois']==2) & (df['Annee']==2025)]
# Revenu mensuel total
revenu_mensuel = df.groupby(df['Date'].dt.to_period('M'))['Facture_USD'].sum()
print("Factures février 2025 :\n", df_fevrier)
print("\nRevenu mensuel total :\n", revenu_mensuel)
```

8.9. Résumé des fonctions clés

Fonction / méthode	Utilité
pd.to_datetime()	Convertir en type datetime
dt.year, dt.month, dt.day	Extraire composants de la date
dt.day_name()	Obtenir le jour de la semaine
dt.to_period('M')	Grouper par période (mois)
Soustraction de dates	Calculer durée en jours

8.10. Interprétation professionnelle

Vous pouvez maintenant :

- ✓ Analyser les revenus par mois, trimestre ou année.
- ✓ Calculer la fréquence des achats ou factures pour chaque client.
- ✓ Préparer des données pour Power BI ou Excel avec série temporelle, ou détecter les clients inactifs.

CHAPITRE IX : APPLIQUER DES FONCTIONS SUR LES COLONNES ET LIGNES

9.1. Objectif

Apprendre à appliquer des transformations personnalisées sur les colonnes ou lignes de votre DataFrame. C'est essentiel pour créer des indicateurs complexes ou des modifications avancées.

9.2. Contexte professionnel

Chez AfriTel RDC, le service marketing souhaite :

- ✓ Catégoriser les clients selon le revenu annuel de façon personnalisée : Bas, Moyen, Haut.
- ✓ Calculer des bonus fictifs sur facture selon des règles complexes.
- ✓ Appliquer des fonctions spécifiques sur des colonnes texte, par exemple mettre les noms en majuscules.

Dataset : clients_factures.csv

ClientID	Nom	Facture_USD	Revenu_annuel
1	Bernard	45.5	546.0
2	Christelle	30.2	362.4
3	David	39.5	474.0
4	Sarah	25.1	301.2
5	Junior	35.3	423.6

9.3. Appliquer une fonction avec apply sur une colonne

Exemple : catégoriser le revenu annuel

```
import pandas as pd
df = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients_factures.csv")
# Fonction personnalisée
def categorie_revenu(x):
    if x < 400:
        return 'Bas'
    elif x < 500:
        return 'Moyen'
    else:
        return 'Haut'
# Appliquer la fonction
df['Categorie_revenu'] = df['Revenu_annuel'].apply(categorie_revenu)
print(df)
```

Résultat attendu :

Nom	Revenu_annuel	Categorie_revenu
Bernard	546.0	Haut
Christelle	362.4	Bas
David	474.0	Moyen
Sarah	301.2	Bas
Junior	423.6	Moyen

Explication :

apply() : applique une fonction ligne par ligne sur une colonne.
Utile pour des règles métiers complexes.

9.4. Appliquer une fonction avec lambda et apply

Exemple : calculer un bonus de 10% sur facture si facture > 40 USD

Code:

```
df['Bonus'] = df['Facture_USD'].apply(lambda x: x*0.1 if x>40 else 0)
print(df[['Nom','Facture_USD','Bonus']])
```

Résultat attendu :

Nom	Facture_USD	Bonus
Bernard	45.5	4.55
Christelle	30.2	0.0
David	39.5	0.0
Sarah	25.1	0.0
Junior	35.3	0.0

9.5. Appliquer une fonction sur plusieurs colonnes (axis=1)

Exemple : créer une colonne texte combinée

```
df['Nom_facture'] = df.apply(lambda row: f"{row['Nom']} ({row['Facture_USD']} USD)",
axis=1)
print(df[['Nom_facture']])
```

Résultat attendu :

Nom_facture
Bernard (45.5 USD)
Christelle (30.2 USD)
David (39.5 USD)
Sarah (25.1 USD)
Junior (35.3 USD)

Explication :

axis =1 : la fonction s'applique ligne par ligne (toutes les colonnes de la ligne sont accessibles).
Parfait pour créer des indicateurs combinés.

9.6. Utiliser map pour transformer une colonne rapidement

Exemple : mettre les noms en majuscules

```
df['Nom_upper'] = df['Nom'].map(str.upper)
print(df[['Nom','Nom_upper']])
```

Résultat attendu :

Nom	Nom_upper
Bernard	BERNARD
Christelle	CHRISTELLE
David	DAVID
Sarah	SARAH
Junior	JUNIOR

Explication :

map() applique une fonction simple sur une colonne.
Plus rapide et lisible pour modifications simples.

9.7. Cas pratique complet

Objectifs :

- ✓ Catégoriser le revenu annuel (Bas, Moyen, Haut)
- ✓ Calculer un bonus sur facture (>40 USD)
- ✓ Créer une colonne combinée Nom + Facture
- ✓ Mettre les noms en majuscules

Code :

```
import pandas as pd
df = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients_factures.csv")
#1 Categorie revenu
def categorie_revenu(x):
    if x < 400:
        return 'Bas'
    elif x < 500:
        return 'Moyen'
    else:
        return 'Haut'
df['Categorie_revenu'] = df['Revenu_annuel'].apply(categorie_revenu)
#2 Bonus sur facture
df['Bonus'] = df['Facture_USD'].apply(lambda x: x*0.1 if x>40 else 0)
#3 Colonne combinée Nom + Facture
df['Nom_facture'] = df.apply(lambda row: f"{row['Nom']} ({row['Facture_USD']} USD)",
axis=1)
#4 Noms en majuscules
df['Nom_upper'] = df['Nom'].map(str.upper)
print(df)
```

9.8. Résumé des fonctions clés

Fonction / méthode	Utilité
apply(func)	Appliquer une fonction sur une colonne
apply(func, axis=1)	Appliquer une fonction ligne par ligne
lambda	Créer des fonctions anonymes simples
map(func)	Transformer rapidement une colonne

9.9. Interprétation professionnelle

Vous pouvez :

- ✓ Appliquer des règles métiers complexes sur tes datasets,
- ✓ Créer des indicateurs avancés pour les tableaux de bord et
- ✓ Préparer des données enrichies pour Power BI ou Excel.

SERIE DE 30 EXERCICES
ET LEURS RESOLUTIONS

NIVEAU FACILE : 10 EXERCICES PANDAS

Exercice 1 : Charger un fichier CSV et afficher les 5 premières lignes

Contexte professionnel :

Vous travaillez chez AfriTel RDC et vous recevez le fichier clients.csv. Vous voulez vérifier que les données se chargent correctement.

Solution :

```
import pandas as pd
# Charger le fichier CSV
clients = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients.csv")
# Afficher les 5 premières lignes
print(clients.head())
```

Résultat attendu :

ClientID	Nom	Ville	Abonnement
1	Bernard	Kinshasa	Gold
2	Christelle	Lubumbashi	Silver
3	David	Goma	Gold
4	Sarah	Kinshasa	Bronze
5	Junior	Kisangani	Silver

Explication :

pd.read_csv() : charge le fichier CSV dans un DataFrame.
head() : permet de voir les 5 premières lignes rapidement pour vérifier la structure.

Exercice 2 : Afficher les colonnes et types de données

Contexte professionnel :

Vous voulez préparer un rapport rapide pour votre manager et savoir quelles colonnes sont numériques ou texte.

Solution :

```
print(clients.columns)
print(clients.dtypes)
```

Résultat attendu :

```
Index(['ClientID', 'Nom', 'Ville', 'Abonnement'],
dtype='object')
ClientID      int64
Nom           object
Ville         object
Abonnement    object
dtype: object
```

Explication :

Columns : liste toutes les colonnes du DataFrame.
dtypes : montre le type de données pour chaque colonne (entier, texte, etc.).

Exercice 3 : Afficher les informations globales du DataFrame

Contexte professionnel :

Vous voulez un aperçu rapide du dataset, du nombre de lignes et des valeurs manquantes.

Solution :

```
print(clients.info())
```

Résultat attendu :

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
Column        Non-Null        Count        Dtype
0             ClientID        5 non-null   int64
 1            Nom             5 non-null   object
 2            Ville           5 non-null   object
 3            Abonnement      5 non-null   object
dtypes: int64(1), object(3)
memory usage: 288.0+ bytes
```

Explication :

info() : donne le nombre de lignes, colonnes, types de données et les valeurs manquantes.

Exercice 4 : Afficher des statistiques descriptives

Contexte professionnel :

Vous voulez connaître le montant moyen des factures, le minimum et le maximum dans *factures.csv*.

Solution :

```
factures = pd.read_csv(r"C:\Users\VotreNom\Desktop\factures.csv")
print(factures.describe())
```

Résultat attendu :

	ClientID	Facture_USD
count	5.000000	5.000000
mean	3.000000	35.12
std	1.581139	7.21
min	1.000000	25.10
25%	2.000000	30.20
50%	3.000000	35.30
75%	4.000000	39.50
max	5.000000	45.50

Explication :

describe() : calcule des statistiques de base (moyenne, écart type, min, max, quartiles) pour les colonnes numériques.

Exercice 5 : Sélectionner une seule colonne

Contexte professionnel :

Vous pouvez extraire uniquement les noms des clients pour un email marketing.

Solution :

```
noms = clients['Nom']
print(noms)
```

Résultat attendu :

```
0      Bernard
1    Christelle
2        David
3        Sarah
4       Junior
Name: Nom, dtype: object
```

Explication:

df['Nom'] : sélectionne une colonne.
Le résultat est un Series pandas.

Exercice 6 : Sélectionner plusieurs colonnes

Contexte professionnel :

Vous voulez créer un tableau avec seulement le nom et la ville pour la livraison des factures.

Solution :

```
nom_ville = clients[['Nom','Ville']]
print(nom_ville)
```

Résultat attendu :

Nom	Ville
Bernard	Kinshasa
Christelle	Lubumbashi
David	Goma
Sarah	Kinshasa
Junior	Kisangani

Explication :

df[['col1','col2']] : sélectionne plusieurs colonnes.
Résultat : un DataFrame.

Exercice 7 : Filtrer les lignes selon une condition

Contexte professionnel :

Vous voulez lister tous les clients de Kinshasa pour envoyer une promotion locale.

Solution :

```
kinshasa_clients = clients[clients['Ville']=='Kinshasa']
print(kinshasa_clients)
```

Résultat attendu :

ClientID	Nom	Ville	Abonnement
1	Bernard	Kinshasa	Gold
4	Sarah	Kinshasa	Bronze

Explication :

df[df['Ville']=='Kinshasa'] : filtre les lignes où la condition est vraie.

Exercice 8 : Filtrer avec plusieurs conditions

Contexte professionnel :

Vous voulez les clients de Kinshasa avec abonnement Gold pour un suivi VIP.

Solution :

```
vip_clients = clients[(clients['Ville']=='Kinshasa') & (clients['Abonnement']=='Gold')]
print(vip_clients)
```

Résultat attendu :

ClientID	Nom	Ville	Abonnement
1	Bernard	Kinshasa	Gold

Explication :

& : combine plusieurs conditions.
Parenthèses obligatoires pour chaque condition.

Exercice 9 : Trier les données

Contexte professionnel :

Vous voulez trier les clients selon le montant de leur facture pour identifier les plus gros payeurs.

Solution:

```
sorted_factures = factures.sort_values(by='Facture_USD', ascending=False)
print(sorted_factures)
```

Résultat attendu :

ClientID	Facture_USD	Date
1	45.5	2025-11-01
3	39.5	2025-11-01
5	35.3	2025-11-01
2	30.2	2025-11-01
4	25.1	2025-11-01

Explication :

sort_values(by='col', ascending=False) : trie par ordre décroissant.
Permet de mettre en évidence les valeurs les plus élevées.

Exercice 10 : Renommer les colonnes

Contexte professionnel :

Vous voulez standardiser les colonnes avant d'exporter le fichier vers Excel.

Solution :

```
clients_renamed = clients.rename(columns={'Nom':'Nom_Client','Ville':'Ville_Client'})
print(clients_renamed)
```

Résultat attendu :

ClientID	Nom_Client	Ville_Client	Abonnement
1	Bernard	Kinshasa	Gold
2	Christelle	Lubumbashi	Silver
3	David	Goma	Gold
4	Sarah	Kinshasa	Bronze
5	Junior	Kisangani	Silver

Explication :

rename(columns={'ancienne':'nouvelle'}) : modifie le nom des colonnes.
Utile pour harmoniser les fichiers avant analyse ou partage.

NIVEAU MOYEN : 10 EXERCICES PANDAS

Exercice 1 : Créer une nouvelle colonne à partir d'une opération

Contexte professionnel :

Chez AfriTel RDC, vous voulez calculer le Montant TTC des factures en ajoutant 16% de TVA.

Solution :

```
import pandas as pd
factures = pd.read_csv(r"C:\Users\VotreNom\Desktop\factures.csv")
factures['Montant_TTC'] = factures['Facture_USD'] * 1.16
print(factures)
```

Résultat attendu :

ClientID	Facture_USD	Montant_TTC
1	45.5	52.78
2	30.2	35.03
3	39.5	45.82
4	25.1	29.12
5	35.3	40.95

Explication :

Multiplier la colonne par 1.16 pour ajouter la TVA.
Le résultat est une nouvelle colonne du DataFrame.

Exercice 2 : Supprimer une colonne inutile

Contexte professionnel :

La direction ne veut plus la colonne ClientID dans le rapport final.

Solution :

```
factures.drop(columns=['ClientID'], inplace=True)
print(factures)
```

Résultat attendu :

Facture_USD	Montant_TTC
45.5	52.78
30.2	35.03
39.5	45.82
25.1	29.12
35.3	40.95

Explication :

drop(columns=[...]) : supprime les colonnes spécifiées.
inplace=True : modifie directement le DataFrame.

Exercice 3 : Renommer plusieurs colonnes

Contexte professionnel :

Avant export Excel, vous voulez renommer les colonnes pour plus de clarté.

Solution :

```
factures.rename(columns={'Facture_USD':'Montant_HT','Montant_TTC':'Montant_TTC
_USD'}, inplace=True)
print(factures)
```

Résultat attendu :

Montant_HT	Montant_TTC_USD
45.5	52.78
30.2	35.03
39.5	45.82
25.1	29.12
35.3	40.95

Explication :

rename(columns={ancienne: nouvelle}) : permet de renommer plusieurs colonnes.

Facilite la lecture et la compréhension du rapport.

Exercice 4 : Trier par plusieurs colonnes

Contexte professionnel :

Vous voulez trier les factures par Montant_TTC décroissant, puis par Montant_HT croissant.

Solution:

```
factures_sorted = factures.sort_values(by=['Montant_TTC_USD','Montant_HT'],
ascending=[False, True])
print(factures_sorted)
```

Résultat attendu :

Montant_HT	Montant_TTC_USD
45.5	52.78
39.5	45.82
35.3	40.95
30.2	35.03
25.1	29.12

Explication :

sort_values(by=[col1,col2], ascending=[True,False]) : permet un tri hiérarchique.

Exercice 5 : Filtrer avec condition sur plusieurs colonnes

Contexte professionnel :

Identifier les factures supérieures à 35 USD et Montant_TTC supérieur à 40 USD.

Solution :

```
factures_filtrees = factures[(factures['Montant_HT']>35) &
(factures['Montant_TTC_USD']>40)]
print(factures_filtrees)
```

Résultat attendu :

Montant_HT	Montant_TTC_USD
45.5	52.78
39.5	45.82
35.3	40.95

Explication :

& : pour combiner les conditions.
Utile pour rapports ciblés.

Exercice 6 : Compter les valeurs uniques

Contexte professionnel :

Tu veux savoir combien de types d'abonnements différents existent.

Solution :

```
clients = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients.csv")
print(clients['Abonnement'].value_counts())
```

Résultat attendu :

```
Gold     2
Silver   2
Bronze   1
Name: Abonnement, dtype: int64
```

Explication :

value_counts() : compte les occurrences de chaque valeur unique.
Utile pour analyse de segmentation client.

Exercice 7 : Grouper et calculer une statistique

Contexte professionnel :

Analyser le revenu moyen par ville.

Solution :

```
clients_factures = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients_factures.csv")
pivot_ville = clients_factures.groupby('Ville')['Revenu_annuel'].mean()
print(pivot_ville)
```

Résultat attendu :

```
Goma          474.0
Kinshasa      423.0
Kisangani     423.6
Lubumbashi    362.4
Name: Revenu_annuel, dtype: float64
```

Explication :

groupby('Ville')['Revenu_annuel'].mean() : calcule la moyenne par ville. Permet de résumer rapidement les données.

Exercice 8 : Créer une colonne avec fonction personnalisée

Contexte professionnel :

Catégoriser les clients selon le revenu annuel : Bas <400, Moyen 400-500, Haut >500.

Solution:

```
def categorie_revenu(x):
    if x<400:
        return 'Bas'
    elif x<500:
        return 'Moyen'
    else:
        return 'Haut'
clients_factures['Categorie'] = clients_factures['Revenu_annuel'].apply(categorie_revenu)
print(clients_factures[['Nom','Revenu_annuel','Categorie']])
```

Résultat attendu :

Nom	Revenu_annuel	Categorie
Bernard	546.0	Haut
Christelle	362.4	Bas
David	474.0	Moyen
Sarah	301.2	Bas
Junior	423.6	Moyen

Explication :

apply(func) : permet d'appliquer une fonction sur chaque valeur d'une colonne. C'est l'idéal pour des règles métiers.

Exercice 9 : Créer une colonne combinée avec ***apply*** *et* ***lambda***

Contexte professionnel :

Créer un identifiant client combinant nom et ville.

Solution :

```
clients_factures['ID_Client'] = clients_factures.apply(lambda row:
f"{row['Nom']}_{row['Ville']}", axis=1)
print(clients_factures[['Nom','Ville','ID_Client']])
```

Résultat attendu :

Nom	Ville	ID_Client
Bernard	Kinshasa	Bernard_Kinshasa
Christelle	Lubumbashi	Christelle_Lubumbashi
David	Goma	David_Goma
Sarah	Kinshasa	Sarah_Kinshasa
Junior	Kisangani	Junior_Kisangani

Explication :

apply(lambda row: ...) : applique une fonction ligne par ligne (axis=1). Permet de combiner plusieurs colonnes.

Exercice 10 : Trier et filtrer avec query

Contexte professionnel :

Identifier les clients de Kinshasa avec revenu > 400 USD pour une promotion VIP.

Solution :

```
vip_clients = clients_factures.query("Ville=='Kinshasa' & Revenu_annuel>400")
print(vip_clients[['Nom','Ville','Revenu_annuel']])
```

Résultat attendu :

Nom	Ville	Revenu_annuel
Bernard	Kinshasa	546.0

Explication :

query() : permet de filtrer avec une syntaxe simple.
Pratique pour des conditions complexes sans parenthèses.

NIVEAU DIFFICILE : 10 EXERCICES PANDAS

Exercice 1 : Manipulation avancée des dates

Contexte professionnel :

Chez AfriTel RDC, vous voulez analyser le délai entre deux factures pour chaque client afin d'identifier les clients inactifs.

Solution :

```
import pandas as pd
factures = pd.read_csv(r"C:\Users\VotreNom\Desktop\factures_dates.csv")
factures['Date'] = pd.to_datetime(factures['Date'])
# Trier par client et date
factures = factures.sort_values(by=['ClientID','Date'])
# Calculer la différence entre factures consécutives
factures['Delai_jours'] = factures.groupby('ClientID')['Date'].diff().dt.days
print(factures)
```

Résultat attendu :

ClientID	Nom	Date	Facture_USD	Delai_jours
1	Bernard	2025-01-15	45.5	NaN
1	Bernard	2025-02-15	50.0	31
2	Christelle	2025-02-10	30.2	NaN

Explication :

groupby('ClientID')['Date'].diff() : calcule la différence entre dates consécutives par client.
dt.days : convertit la différence en jours. C'est utile pour analyser la rétention client.

Exercice 2 : Détection des valeurs manquantes et remplissage

Contexte professionnel :

Le dataset clients_factures.csv contient des valeurs manquantes dans Revenu_annuel. Vous voulez remplacer les valeurs manquantes par la moyenne.

Solution :

```
clients_factures = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients_factures.csv")
# Remplacer NaN par la moyenne
clients_factures['Revenu_annuel'].fillna(clients_factures['Revenu_annuel'].mean(),
inplace=True)
print(clients_factures)
```

Explication :

fillna() : permet de remplacer les valeurs manquantes.
Remplacer par la moyenne est courant en analyse de données.

Exercice 3 : Fusionner deux DataFrames

Contexte professionnel :

Vous avez deux fichiers : clients.csv et factures.csv. Vous voulez combiner les informations pour créer un rapport complet.

Solution :

```
clients = pd.read_csv(r"C:\Users\VotreNom\Desktop\clients.csv")
factures = pd.read_csv(r"C:\Users\VotreNom\Desktop\factures.csv")
rapport = pd.merge(factures, clients, on='ClientID', how='left')
print(rapport.head())
```

Résultat attendu :

ClientID	Facture_USD	Date	Nom	Ville	Abonnement
1	45.5	2025-01-15	Bernard	Kinshasa	Gold
2	30.2	2025-02-10	Christelle	Lubumbashi	Silver

Explication :

merge(..., on='ClientID', how='left') : combine les datasets par la colonne commune.
left join : conserve toutes les lignes du DataFrame de gauche.

Exercice 4 : Grouper et calculer plusieurs agrégations

Contexte professionnel :

Analyser le revenu total et moyen par ville.

Solution :

```
agg_ville = clients_factures.groupby('Ville')['Revenu_annuel'].agg(['sum','mean'])
print(agg_ville)
```

Résultat attendu :

Ville	sum	mean
Goma	474.0	474.0
Kinshasa	969.0	484.5
Kisangani	423.6	423.6
Lubumbashi	362.4	362.4

Explication :

agg(['sum','mean']) : permet de calculer plusieurs statistiques en un seul regroupement. C'est très utile pour les rapports consolidés.

*Exercice 5 : Transformer des colonnes avec **apply** et **lambda** complexes*

Contexte professionnel :

Vous voulez appliquer un bonus de fidélité : 5% si le revenu > 400, sinon 2%.

Solution :

```
clients_factures['Bonus'] = clients_factures['Revenu_annuel'].apply(lambda x: x*0.05 if x>400 else x*0.02)
print(clients_factures[['Nom','Revenu_annuel','Bonus']])
```

Résultat attendu :

Nom	Revenu_annuel	Bonus
Bernard	546.0	27.3
Christelle	362.4	7.248
David	474.0	23.7
Sarah	301.2	6.024
Junior	423.6	21.18

Explication :

lambda : permet des conditions directement dans l'assignation.
Combine logique métier et calcul en une seule ligne.

Exercice 6 : Pivot table pour résumé

Contexte professionnel :

Créer un tableau croisé avec revenu moyen par ville et type d'abonnement.

Solution :

```
pivot = pd.pivot_table(clients_factures, values='Revenu_annuel', index='Ville', columns='Abonnement', aggfunc='mean')
print(pivot)
```

Résultat attendu :

Ville	Bronze	Gold	Silver
Goma	NaN	474.0	NaN
Kinshasa	301.2	546.0	NaN
Kisangani	NaN	NaN	423.6
Lubumbashi	NaN	NaN	362.4

Explication :

pivot_table : permet de résumer plusieurs dimensions.
Utile pour rapports avancés et tableaux de bord.

Exercice 7 : Détecter et supprimer les doublons

Contexte professionnel :

Votre fichier factures.csv contient des doublons accidentels.

Solution :

```
factures_no_dup = factures.drop_duplicates()
print(factures_no_dup)
```

Explication :

drop_duplicates() : supprime les lignes identiques. C'est essentiel pour le nettoyage des données avant analyse.

Exercice 8 : Combiner apply et datetime

Contexte professionnel :

Calculer l'âge des factures en jours par rapport à aujourd'hui.

Solution:

```
import datetime
factures['Date'] = pd.to_datetime(factures['Date'])
factures['Age_jours'] = factures['Date'].apply(lambda x: (datetime.datetime.today() -
x).days)
print(factures[['ClientID','Date','Age_jours']])
```

Explication :

datetime.datetime.today() - x : calcule le délai.
apply(lambda x: ...) : applique la transformation ligne par ligne.

Exercice 9 : Chaîner plusieurs opérations

Contexte professionnel :

Filtrer les clients avec revenu > 400, trier par revenu décroissant, et afficher uniquement les noms et revenus.

Solution :

```
top_clients =
clients_factures[clients_factures['Revenu_annuel']>400].sort_values('Revenu_annuel',
ascending=False)[['Nom','Revenu_annuel']]
print(top_clients)
```

Résultat attendu :

Nom	Revenu_annuel
Bernard	546.0
David	474.0
Junior	423.6

Explication :

Chaîner plusieurs opérations permet de filtrer, trier et sélectionner en une seule ligne.

Exercice 10 : Exporter le DataFrame nettoyé

Contexte professionnel :

Après analyse, vous voulez exporter le rapport final vers Excel pour votre manager.

Solution :

```
clients_factures.to_excel(r"C:\Users\VotreNom\Desktop\rapport_clients.xlsx",
index=False)
```

Explication :

to_excel() : exporte le DataFrame.
index=False : supprime la colonne d'index automatique.

TABLE DES MATIERES