**Lire des fichiers**

|  |  |  |
| --- | --- | --- |
| **Modèle** | **Exemple** | **Description** |
| pd.read\_csv(filepath, sep=..., encoding=...) | pd.read\_csv("data.csv", sep=";", encoding="utf-8") | Lire un fichier CSV |
| pd.read\_excel(filepath, sheet\_name=..., usecols=..., engine=...) | pd.read\_excel("data.xlsx", sheet\_name="Feuil1", usecols="A:C",  engine=’calamine’) | Lire un fichier Excel |
| pd.read\_pickle(filepath) | pd.read\_pickle("data.pkl") | Lire un fichier pickle (.pkl) |

**Explorer un DataFrame**

|  |  |  |
| --- | --- | --- |
| **Modèle** | **Exemple** | **Description** |
| df.shape | df.shape | Nombre de lignes et colonnes |
| df.index | df.index | Accès à l’index du DataFrame |
| df.columns | df.columns | Liste des noms de colonnes |
| df.dtypes | df.dtypes | Type de chaque colonne |
| df.info() | df.info() | Résumé des colonnes, types, non-nuls et mémoire |
| df.describe() | df.describe() | Statistiques descriptives sur colonnes numériques |
| df.isna().sum() | df.isna().sum() | Nombre de valeurs manquantes par colonne |
| df.head(n) | df.head(10) | Afficher les n premières lignes (défaut n=5) |
| df.tail(n) | df.tail(3) | Afficher les n dernières lignes (défaut n=5) |

**Explorer une Series**

|  |  |  |
| --- | --- | --- |
| Modèle | Exemple | Description |
| series.unique() | df["country"].unique() | Valeurs uniques dans la Series |
| series.nunique() | df["gender"].nunique() | Nombre de valeurs uniques |
| series.sort\_values() | df["score"].sort\_values() | Trier les valeurs (ordre croissant par défaut) |
| series.value\_counts( normalize=...) | df["region"] .value\_counts(normalize=True) | Compter les occurrences (avec % si normalize=True) |

**Slicer un DataFrame**

|  |  |  |
| --- | --- | --- |
| Modèle | Exemple | Description |
| df.loc[ligne, colonne] | df.loc[0, "age"] | Accès par nom de ligne et colonne |
| df.iloc[ligne, colonne] | df.iloc[0, 2] | Accès par index numérique de ligne et colonne |
| df.at[ligne, colonne] | df.at[0, "name"] | Accès rapide à une cellule (par nom) |
| df.iat[ligne, colonne] | df.iat[0, 1] | Accès rapide à une cellule (par index numérique) |

**Filtrer un DataFrame avec .query()**

|  |  |
| --- | --- |
| **Nom** | **Utilité** |
| df.query(condition) | Filtre le DataFrame selon la condition (nom de colonne, valeur) |

| **Modèle** | **Exemple** | **Description** |
| --- | --- | --- |
| col == valeur | df.query("year == 2020") | Égal à |
| col != valeur | df.query("continent != 'Asia'") | Différent de |
| col > valeur | df.query("age > 18") | Supérieur à |
| col >= valeur | df.query("score >= 90") | Supérieur ou égal |
| col < valeur | df.query("price < 100") | Inférieur à |
| col <= valeur | df.query("weight <= 70") | Inférieur ou égal |
| col in @liste | df.query("country in @european\_countries") | Appartient à une liste (utilisation de @ pour une variable externe) |
| col not in @liste | df.query("brand not in @excluded\_brands") | N'appartient pas à une liste |
| col.str.contains('texte') | df.query("name.str.contains('John', case=False, na=False)") | Contient une sous-chaîne |
| col.isna() ou col.notnal() | df.query("score.isna()") | Valeurs manquantes |
| Conditions multiples avec and | df.query("year > 2010 and year < 2020") | Condition composée (ET logique) |
| Conditions multiples avec or | df.query("continent == 'Europe' or continent == 'Asia'") | Condition composée (OU logique) |
| col.between(val1, val2) | df.query("age.between(20, 30)") | Valeur comprise entre deux bornes |
| Négation avec ~ | df.query("~(status == 'inactive')") | Négation logique |
| Filtrage direct de booléens | df.query("is\_active") OU df.query("~is\_deleted") | Colonnes booléennes (True/False) |

**Filtrer un DataFrame avec .loc**

| **Modèle** | **Exemple** | **Description** |
| --- | --- | --- |
| df.loc[df[col] == valeur] | df.loc[df[year] == 2020] | Égal à |
| df.loc[df[col] != valeur] | df.loc[df[continent] != 'Asia'] | Différent de |
| df.loc[df[col] > valeur] | df.loc[df[age] > 18] | Supérieur à |
| df.loc[df[col] >= valeur] | df.loc[df[score] >= 90] | Supérieur ou égal |
| df.loc[df[col] < valeur] | df.loc[df[price] < 100] | Inférieur à |
| df.loc[df[col] <= valeur] | df.loc[df[weight] <= 70] | Inférieur ou égal |
| df.loc[df[col].isin(liste)] | df.loc[df[country] .isin(european\_countries)] | Appartient à une liste |
| df.loc[~df[col].isin(liste)] | df.loc[~df[brand] .isin(excluded\_brands)] | N'appartient pas à une liste |
| df.loc[df[col].str.contains('texte', case=..., na=False)] | df.loc[df[name] .str.contains('John', case=False, na=False)] | Contient une sous-chaîne |
| df.loc[df[col].isna()] | df.loc[df[score].isna()] | Valeurs manquantes |
| df.loc[(cond1) & (cond2)] | df.loc[ (df[year] > 2010)  & (df[year] < 2020)] | Condition composée (ET logique) |
| df.loc[(cond1) | (cond2)] | `df.loc[(df[continent] == 'Europe') | (df[continent] == 'Asia')]` | Condition composée (OU logique) |
| df.loc[df[col].between(val1, val2)] | df.loc[df[age]between(20, 30)] | Valeur comprise entre deux bornes |
| df.loc[~(condition)] | df.loc[~(df[status] == 'inactive')] | Négation logique |

**Ajouter une colonne**

| **Modèle** | **Exemple** | **Description** |
| --- | --- | --- |
| df[nouvelle\_col] = valeur | df[discounted] = False | Ajouter une colonne avec une valeur fixe |
| df[col2] = df[col1] \* facteur | df[prix\_ttc] = df[prix] \* 1.2 | Colonne calculée à partir d'une autre |
| df[col3] = df[col1] + df[col2] | df[total] = df[net] + df[taxe] | Combinaison de colonnes existantes |
| df[col] = df[col].apply(fonction) | df[upper\_name] = df[name].apply(str.upper) | Transformation via une fonction (lent) |
| df[col] = df[col].apply(function lambda) | df[cat\_age\_2] = df[age].apply(lambda x : “-25” if age < 25 else “25+” ) | Transformation via une fonction lambda (lent) |
| df[col] = pd.cut(df[val], bins=...) | df[age\_group] = pd.cut(df[age], bins=[0,18,65,100]) | Création de n tranches (rapide) |
| df[col] = np.where(condition, A, B) | df[statut] = np.where(df[score] >= 10, "valid", "retry") | Conditionnelle avec NumPy |
| df[col] = df[col].map(dictionnaire) | df[continent\_name] = df[continent].map(code\_to\_label) | Remplacement via un dictionnaire |
| df.loc[condition, col] = valeur | df.loc[df[age] < 18, categorie] = "minor" | Ajout conditionnel à certaines lignes |
| df = df.assign(col=...) | df = df.assign(next\_year = df[year] + 1) | Ajout sans modifier l’original, méthode chainable |

**Supprimer une ligne ou une colonne**

| **Modèle** | **Exemple** | **Description** |
| --- | --- | --- |
| df.drop(nom\_colonne, axis=1) | df.drop(age, axis=1) | Supprimer une colonne |
| df.drop([col1, col2], axis=1) | df.drop(["age", "gender"], axis=1) | Supprimer plusieurs colonnes |
| df.drop(index, axis=0) | df.drop(0, axis=0) | Supprimer une ligne par son index |
| df.drop([index1, index2]) | df.drop([0, 5]) | Supprimer plusieurs lignes |
| df.drop(df[df.colonne == valeur].index) | df.drop(df[df[status] == inactive].index) | Supprimer des lignes selon une condition |
| df = df[df.colonne != valeur] | df = df[df[status] != inactive] | Filtrage inversé (équivaut à suppression) |
| df = df[~df.colonne.isin(liste)] | df = df[~df[country].isin(banned\_countries)] | Supprimer lignes selon une liste de valeurs |
| df = df.dropna(axis=0) | df = df.dropna(axis=0, how=’all’) Supprimée si toutes les valeurs sont na | Supprimer les lignes avec valeurs manquantes |
| df = df.dropna(axis=0, how=’any’) Supprimée si au – 1 valeur est na. Valeur par défaut |
| df = df.dropna(axis=1) | df = df.dropna(axis=1) | Supprimer les colonnes avec valeurs manquantes |
| df = df.loc[df[col].notna()] | df = df.loc[df[score].notna()] | Conserver seulement les lignes non nulles |

**Concaténer deux DataFrames**

| **Modèle** | **Exemple** | **Description** |
| --- | --- | --- |
| pd.concat([df1, df2]) | pd.concat([df1, df2]) | Concaténation verticale (lignes) par défaut |
| pd.concat([df1, df2], axis=1) | pd.concat([df1, df2], axis=1) | Concaténation horizontale (colonnes) |
| pd.concat([df1, df2], ignore\_index=True) | pd.concat([df1, df2], ignore\_index=True) | Réindexe les lignes après concat |
| pd.concat([df1, df2], keys=["A", "B"]) | pd.concat([df1, df2], keys=["2023", "2024"]) | Ajoute un niveau hiérarchique d’index |
| pd.concat([df1, df2], join="inner") | pd.concat([df1, df2], join="inner") | Conserve uniquement les colonnes communes |
| pd.concat([df1, df2], axis=1, join="inner") | pd.concat([df1, df2], axis=1, join="inner") | Jointure uniquement sur les index communs |

**Merge (= Jointure SQL sur valeurs de colonne) deux DataFrames**

| **Modèle** | **Exemple** | **Description** |
| --- | --- | --- |
| df1.merge(df2, how=’inner’, on="col") | df1.merge(df2,  how=’inner’, #défaut  on="id") | Jointure interne sur une clé commune |
| df1.merge(df2, how="left", on="col") | df1.merge(df2, how="left",  on="id") | Jointure à gauche |
| df1.merge(df2, how="right", on="col") | df1.merge(df2, how="right",  on="id") | Jointure à droite |
| df1.merge(df2, how="outer", on="col") | df1.merge(df2, how="outer",  on="id") | Jointure complète |
| df1.merge(df2, left\_on="col1" ,right\_on="col2") | df1.merge(df2, left\_on="user\_id", right\_on="id") | Noms de clés différents |
| df1.merge(df2, on="col", suffixes=("\_A", "\_B")) | df1.merge(df2, on="id",  suffixes=("\_left", "\_right")) | Gérer les colonnes en double |
| df1.merge(df2, on="col”, how="outer", sort=False) | df1.merge(df2, on="id", how="outer", sort=False) | Jointure sans trier, gain de temps |
| df1.merge(df2, on="col", validate="1:1") | df1.merge(df2, on="id",  validate="1:1") | Valider le type de jointure (ex : 1:1, m:1...) |

**Join (=Jointure sur index ou nom de colonne) deux DataFrames**

| **Modèle** | **Exemple** | **Description** |
| --- | --- | --- |
| df1.join(df2) | df1.join(df2) | Jointure à gauche sur les index |
| df1.join(df2, how="inner") | df1.join(df2, how="inner") | Jointure interne sur les index communs |
| df1.join(df2, how="outer") | df1.join(df2, how="outer") | Jointure complète sur les index |
| df1.join(df2, how="left", sort=False) | df1.join(df2,  how="left", sort=False) | Jointure gauche sans trier, gain de temps |
| df1.join(df2, lsuffix="\_left", rsuffix="\_right") | df1.join(df2, lsuffix="\_A",  rsuffix="\_B") | Gérer les doublons de colonnes |
| df1.join([df2, df3]) | df1.join([df2, df3]) | Joindre plusieurs df en même temps |

**Agréger un DataFrame : simple opération**

| **Modèle** | **Exemple** | **Description** |
| --- | --- | --- |
| df.agg("fonction") | df.agg("mean") | Agrégation sur tout un dataFrame |
| df["col"].agg("fonction") | df["price"].agg("sum") | Agréger une seule colonne |
| df.agg( ["fonction1", "fonction2"] ) | df.agg(["min", "max"]) | Appliquer plusieurs fonctions |
| df.agg( {"col1": "fonction",  "col2": "fonction"}) | df.agg( {"price": "mean", "qty": "sum"} ) | Agréger colonne par colonne |

Exemples de fonctions : median, min, max, sum, prod, first, last, cumsum, quantile, idxmin, idxmax, fonctions lambda…

**Agréger un DataFrame avec .groupby()**

| **Modèle** | **Exemple** | **Description** |
| --- | --- | --- |
| df.groupby(« col ») .agg(« fonction ») | df.groupby(“category”) .agg(“mean”) | Agrégation par groupe |
| df.groupby(« col »)[[« c1 », « c2 »]].agg(« fonction ») | df.groupby(“brand”) [[“price”, “qty”]] .agg(“sum”) | Agrégation multi-colonnes par groupe |
| df.groupby(« col »).agg({ « c1 » : « fonction1 »,  « c2 » : « fonction2 »}) | df.groupby(“type”).agg( {“price”: “mean”, “qty”: “sum”} ) | Fonctions différentes selon les colonnes |
| df.groupby([« col1 », « col2 »]).agg(« fonction ») | df.groupby([“region”, “year”]).agg(“sum”) | Agrégation sur plusieurs colonnes |
| df.groupby(« col »).size() | df.groupby(« gender »).size() | Taille de chaque groupe |
| df.groupby(« col »).count() | df.groupby(« team »).count() | Compte non-nul par colonne dans chaque groupe |

Les exemples de fonctions possibles avec .agg() sont généralement possibles avec .groupby()

**Agréger un DataFrame avec .piovt\_table()**

| **Modèle** | **Exemple** | **Description** |
| --- | --- | --- |
| df.pivot\_table(values="val", index="row", columns="col", aggfunc="fonction") | df.pivot\_table(values="sales", index="product",  columns="month",  aggfunc="sum") | Tableau croisé dynamique |