JOUR 1_2

Manipulation avancée de données avec Pandas

Sélection et filtrage avancés

Pandas offre des fonctionnalités avancées pour sélectionner et filtrer des données à partir de DataFrames en utilisant diverses techniques :

1. Indexation Avancée

- **Utilisation de Loc et Iloc :** `loc` pour l'indexation par libellé et `iloc` pour l'indexation par position.
- **Sélection de Colonnes et de Lignes Spécifiques :**

```
# Indexation avancée avec loc et iloc
import pandas as pd

# Création d'un DataFrame
data = {
    'A': [1, 2, 3, 4],
    'B': ['a', 'b', 'c', 'd'],
    'C': [10, 20, 30, 40]
}
df = pd.DataFrame(data, index=['Row1', 'Row2', 'Row3',
'Row4'])

# Sélection de lignes spécifiques avec loc
print(df.loc[['Row2', 'Row4'], ['A', 'C']])
```

```
# Sélection de lignes et colonnes spécifiques avec iloc
print(df.iloc[[1, 3], [0, 2]])
```

2. Filtrage Conditionnel

- **Utilisation de Conditions :** Filtrer les données en fonction de conditions spécifiques.
- **Filtrage Basé sur les Valeurs :**

```
# Filtrage conditionnel des données
import pandas as pd

# Création d'un DataFrame
data = {
    'A': [1, 2, 3, 4],
    'B': ['a', 'b', 'c', 'd'],
    'C': [10, 20, 30, 40]
}
df = pd.DataFrame(data)

# Filtrage des lignes où la valeur de la colonne 'A' est supérieure à 2
filtered_df = df[df['A'] > 2]
print(filtered_df)
```

3. Utilisation de Méthodes Avancées

- **Isin() :** Vérifier si les valeurs appartiennent à une liste spécifique.
- **Between() :** Filtrer les valeurs dans une plage donnée.

```
# Utilisation de méthodes avancées pour filtrer les données
import pandas as pd

# Création d'un DataFrame
data = {
```

```
    'A': [1, 2, 3, 4],
    'B': ['a', 'b', 'c', 'd'],
    'C': [10, 20, 30, 40]
}
df = pd.DataFrame(data)

# Filtrage des lignes où la colonne 'A' est dans une liste spécifique
filtered_df = df[df['A'].isin([2, 4])]
print(filtered_df)

# Filtrage des lignes où la colonne 'C' est entre deux valeurs
filtered_df = df[df['C'].between(20, 35)]
print(filtered_df)
```

Ces techniques avancées de sélection et de filtrage permettent de manipuler et d'extraire des données spécifiques à partir de DataFrames de manière efficace, facilitant ainsi l'analyse et le traitement des données.

Jointures de données et fusion de DataFrames

Pandas propose différentes méthodes pour combiner des DataFrames en fonction de clés ou d'indices communs :

1. Merge()

- **Fusion de DataFrames :** Combine les DataFrames en fonction de colonnes ou d'index communs.
- **Paramètres :** `on`, `how`, `left_on`, `right_on`, `left_index`, `right_index`, etc.

```
# Utilisation de la méthode merge pour fusionner des DataFrames
import pandas as pd

# Création de deux DataFrames
```

```
data1 = {
    'ID': [1, 2, 3],
    'Valeur1': ['A', 'B', 'C']
}
data2 = {
    'ID': [1, 2, 4],
    'Valeur2': ['X', 'Y', 'Z']
}
df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

# Fusion des DataFrames basée sur la colonne 'ID'
merged_df = pd.merge(df1, df2, on='ID', how='inner')
print(merged_df)
```

2. Concatenate()

- **Concaténation de DataFrames :** Empile plusieurs DataFrames les uns sur les autres.
- **Paramètres :** `axis`, `ignore_index`, etc.

```
# Utilisation de la méthode concat pour concaténer des DataFrames
import pandas as pd

# Création de deux DataFrames
data1 = {
    'A': [1, 2],
    'B': ['a', 'b']
}
data2 = {
    'A': [3, 4],
    'B': ['c', 'd']
}
df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)
```

```
# Concaténation des DataFrames le long de l'axe des lignes
concatenated_df = pd.concat([df1, df2], ignore_index=True)
print(concatenated_df)
```

3. Join()

- **Jointure de DataFrames :** Effectue une jointure entre les colonnes des DataFrames.
- **Paramètres :** `on`, `how`, `lsuffix`, `rsuffix`, etc.

```
# Utilisation de la méthode join pour joindre des DataFrames
import pandas as pd

# Création de deux DataFrames
data1 = {
    'A': [1, 2],
    'B': ['a', 'b']
}
data2 = {
    'C': [3, 4],
    'D': ['c', 'd']
}
df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

# Jointure des DataFrames basée sur les index
joined_df = df1.join(df2)
print(joined_df)
```

Ces méthodes de Pandas permettent de combiner et de fusionner des DataFrames en fonction de diverses conditions et clés, facilitant ainsi l'agrégation et la consolidation de données provenant de différentes sources.

Application de Fonctions Personnalisées

- **Apply :** Appliquer une fonction personnalisée à une DataFrame ou à une série.
- **Lambda Functions :** Utilisation de fonctions lambda pour des opérations spécifiques.

```
# Exemple d'utilisation de Apply avec une fonction personnalisée
import pandas as pd

# Création d'un DataFrame
data = {
    'A': [1, 2, 3, 4],
    'B': [10, 20, 30, 40]
}
df = pd.DataFrame(data)

# Application d'une fonction pour calculer la somme de deux colonnes
df['Somme'] = df.apply(lambda row: row['A'] + row['B'], axis=1)
print(df)
```

Nettoyage Avancé des Données et Gestion des Valeurs Manquantes

1. Traitement des Valeurs Manquantes

- **Méthodes Pandas :** `isna()`, `fillna()`, `dropna()`.
- **Gestion des Valeurs Manquantes :**

```
# Gestion des valeurs manquantes avec Pandas
import pandas as pd
```

```
import numpy as np

# Création d'un DataFrame avec des valeurs manquantes
data = {
    'A': [1, np.nan, 3, np.nan],
    'B': ['a', 'b', np.nan, 'd'],
    'C': [np.nan, 5, 6, 7]
}
df = pd.DataFrame(data)

# Vérification des valeurs manquantes dans le DataFrame
print(df.isna())

# Remplacement des valeurs manquantes par une valeur spécifique
filled_df = df.fillna('Remplacé')
print(filled_df)

# Suppression des lignes contenant des valeurs manquantes
cleaned_df = df.dropna()
print(cleaned_df)
```

2. Imputation de Valeurs Manquantes

- **Imputation Statistique :** Remplacer les valeurs manquantes par la moyenne ou la médiane.
- **Utilisation de Fonctions Personnalisées :**

```
# Imputation de valeurs manquantes avec des statistiques
import pandas as pd
import numpy as np

# Création d'un DataFrame avec des valeurs manquantes
data = {
    'A': [1, 2, np.nan, 4, 5],
    'B': [10, np.nan, np.nan, 40, 50]
}
```

```
df = pd.DataFrame(data)

# Imputation des valeurs manquantes avec la moyenne
mean_imputed = df.fillna(df.mean())
print(mean_imputed)

# Imputation des valeurs manquantes avec une fonction
personnalisée
def custom_imputer(series):
    return series.fillna(series.median())

median_imputed = df.apply(custom_imputer)
print(median_imputed)
```

3. Traitement des Valeurs Duplicates

- **Détection et Suppression des Duplicatas :**

```
# Traitement des valeurs dupliquées
import pandas as pd

# Création d'un DataFrame avec des valeurs dupliquées
data = {
    'A': [1, 2, 2, 3, 4],
    'B': ['a', 'b', 'b', 'c', 'd']
}
df = pd.DataFrame(data)

# Vérification des valeurs dupliquées
print(df.duplicated())

# Suppression des lignes dupliquées
cleaned_df = df.drop_duplicates()
print(cleaned_df)
```

Ces techniques avancées de nettoyage de données en utilisant Pandas permettent de gérer efficacement les valeurs manquantes, d'effectuer des imputations de manière plus sophistiquée et de détecter/supprimer les valeurs dupliquées pour obtenir des ensembles de données propres et fiables.