Pandas

Lire un fichier csv avec pandas

```
pd.read_csv('data.csv')

# pour choisir le séparateur
pd.read_csv('data.csv', sep=';')
```

Séléctionner des colonnes

1. Séléctionner des colonnes

```
df[['col1', 'col2']]
```

2. Séléctionner des colonnes avec des conditions

```
df[df['col1'] == 'condition']
```

3. Séléctionner des colonnes avec plusieurs conditions

```
df[(df['col1'] == 'condition1') & (df['col2'] == 'condition2')]
```

4. Séléctionner des colonnes par des types

```
df.select_dtypes(include=['int64'])

df.select_dtypes(include=[np.number])
```

Séléctionner des lignes

1. `.loc[]` pour sélécionner plusieurs lignes

```
data.loc[[1,2], "nom_colonne"]

# data.loc permet de récupérer des ligne
# data.loc[[1,2]] -> récupère les lignes 1 et 2

# data.loc[[1,2], "nom_colonne"] -> récupère la colonne sex de la ligne 1 et 2
```

2. `.at[]` pour séléctionner une seule ligne

```
data.at[1, "nom_colonne"]
```

Récupérer des informations sur le dataframe

1. Récupérer les informations sur le dataframe

```
df.info()
```

2. Récupérer les statistiques sur le dataframe

```
df.describe()
```

3. Récupérer les colonnes du dataframe

```
df.columns
```

4. Récupérer les valeurs uniques d'une colonne

```
df['colonne'].unique()
```

6. Avoir la taille du dataframe

```
df.shape
```

7. Récupérer la moyenne d'une colonne

```
df['colonne'].mean()
```

8. Récupérer la somme d'une colonne

```
df['colonne'].sum()
```

9. Récupère le type des colonnes

```
data.dtypes
```

Récupérer les NaN

```
df.isna()
```

Récupérer les lignes avec des NaN

```
df.isna().any(axis=1)
# axis=1 pour récupérer les lignes
# axis=0 pour récupérer les colonnes
```

Supprimer des éléments avec un filtre

1. Supprimer les lignes avec des filtres

```
data.drop(data[filtre].index, inplace=True)

# le premier paramètre est une liste d'index

# inplace=True pour modifier le dataframe
# inplace=False pour ne pas modifier le dataframe
```

2. Supprimer des colonnes

```
data.drop(columns=['col1', 'col2'], inplace=True)
```

Compter des valeurs dans une colonne

```
df['colonne'].value_counts()
```

Remplacer des valeurs

```
data["colonne"].replace({1: "Homme", 0: "Femme"}, inplace=True)
```

Définir une colonne comme index

```
data.set_index('colonne', inplace=True)
```

Regarder comment les valeurs se répartissent

```
pd.cut(data['colonne'], bins=3)
```

Regarder les corrélations

```
data.corr()
```

Trier les données

```
data.sort_values('colonne', ascending=False)
```

Sklearn

Séparer les données

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Créer un modèle

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.cluster import KMeans
...
model = LinearRegression()
```

Entraîner un modèle

```
model.fit(X_train, y_train)
```

Faire des prédictions

```
model.predict(X_test)
```

Evaluer un modèle

```
model.score(X_test, y_test)
```

Sauvegarder un modèle

```
import joblib
joblib.dump(model, 'model.pkl')
```

Charger un modèle

```
model = joblib.load('model.pkl')
```

Créer un pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('model', LinearRegression())
])
```

Le PCA (Principal Component Analysis)

c'est une méthode de réduction de dimensionnalité qui permet de réduire le nombre de colonnes d'un dataframe.
permet de savoir quelles colonnes sont les plus importantes / permet de lier les colonnes entre elles

```
from sklearn.decomposition import PCA

pca = PCA(svd_solver='full')
data = pca.fit_transform(data)
# transforme en DataFrame
data = pd.DataFrame(data)
```

Pour afficher les colonnes les plus importantes

```
pca.explained_variance_ratio_

# transforme en DataFrame
pd.DataFrame(pca.explained_variance_ratio_)
```

CAH (Classification Ascendante Hiérarchique)

C'est une méthode de clustering qui permet de regrouper des individus en fonction de leurs caractéristiques.
ça permet de savoir si des individus se ressemblent ou non pour les regrouper ensemble et les différencier des autres. Pour que entrainement soit efficace, il faut que les individus soient proches les uns des autres.

```
from sklearn.cluster import AgglomerativeClustering
model = AgglomerativeClustering(n_clusters=3)
model.fit(data)

# pour récupérer les labels
model.labels_
```

CAH avec PCA

```
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering
# model est AgglomerativeClustering et on fit sur les données transformées par PCA
model.fit(dataPCA)
PCA_CAH = pd.DataFrame(model.labels_, index=data.index)
```

Matrice de confusion

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

Matplotlib et Seaborn

Créer un graphique

```
import matplotlib.pyplot as plt
plt.plot(x, y)
```

dendrogramme

```
from scipy.cluster.hierarchy import dendrogram, linkage
Z = linkage(data, 'ward')
dendrogram(Z)
```

nuage de points

```
import seaborn as sns
sns.scatterplot(x='col1', y='col2', data=data)
```

histogramme

```
sns.histplot(data['col1'])
```

pairplot

```
sns.pairplot(data)
```

scatterplot

```
sns.scatterplot(x='col1', y='col2', data=data)
```

Matrice de confusion

```
import seaborn as sns
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True)
```