INTRODUCTION

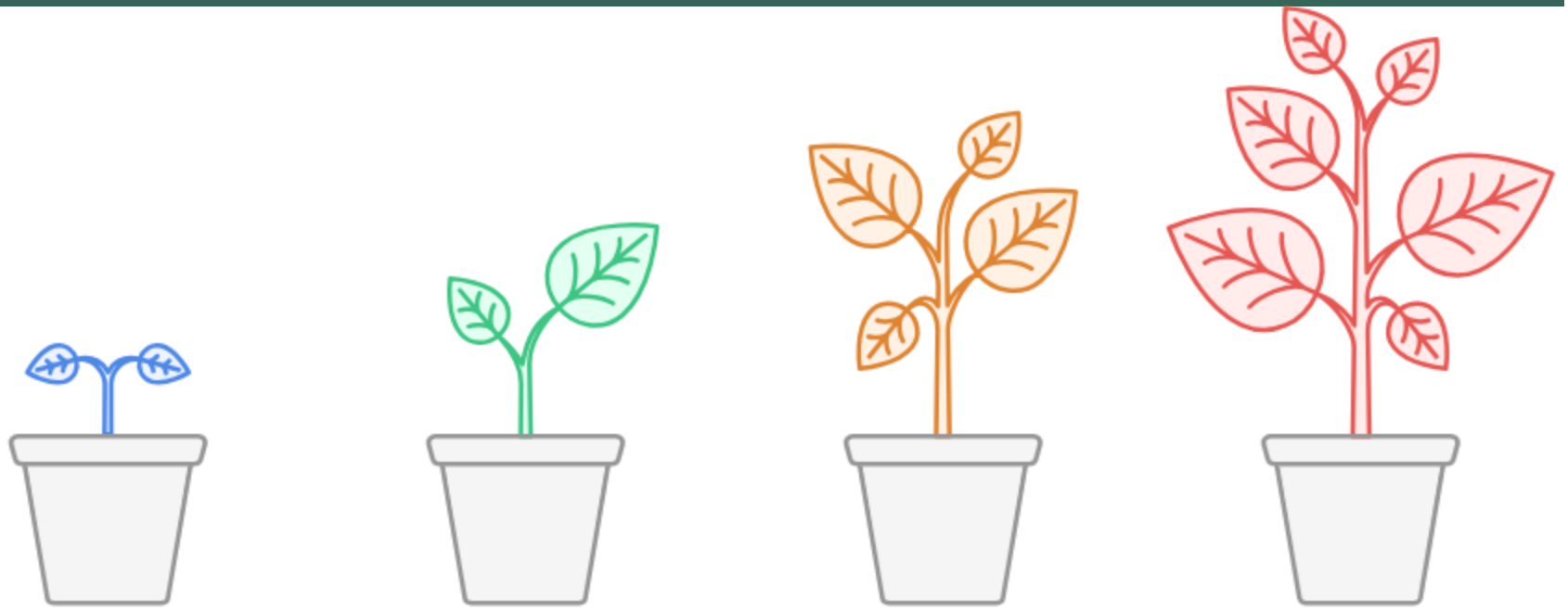
- Python pour l'analyse des données : Introduction à NumPy, Pandas, Matplotlib et Seaborn.
- Objectifs :
- Comprendre pourquoi et comment utiliser chaque outil.
- Découvrir leurs concepts fondamentaux.
- Apprendre via des cas pratiques.

Naïdine Mohamed
naidine.mohamed1@gmail.com

POURQUOI PYTHON POUR LA DATA ?

- Langage puissant, flexible et lisible.
- Large écosystème pour la manipulation de données et la visualisation.
- Bibliothèques optimisées pour la performance.

ÉTAPES CLÉS DE L'ANALYSE DE DONNÉES



Lecture des données

Transformation et nettoyage

Analyse et modélisation

Visualisation des résultats

MODULE 1 : NUMPY

- Introduction à NumPy : Manipulation efficace des tableaux numériques et calculs optimisés.

- Caractéristiques clés :
  - Prend en charge les tableaux multidimensionnels et les matrices.
  - Fournit des outils pour la manipulation des tableaux et les opérations mathématiques.
  - Sert de base à des bibliothèques telles que Pandas, Matplotlib et Scikit-learn.

https://www.w3schools.com/python/numpy/default.asp

MODULE 1 : NUMPY

- Cas pratique :
  - Manipulation et transformation des données.
  - Algèbre linéaire, transformée de Fourier et génération de nombres aléatoires.
  - Calculs à haute performance pour les grands ensembles de données.
  - Génération de datasets

NUMPY INSTALLATION ET IMPORTATION

- Installation :
  - pip install numpy
  - conda install numpy
- Importation :
  - import numpy as np

NUMPY CRÉATION DE TABLEAUX

- np.array([1, 2, 3]) : Crée un tableau à partir d'une liste.
- np.array([[1, 2], [3, 4]]) : Crée une matrice à partir de deux listes

- Remarques importantes :
  - Les tableaux sont plus efficaces que les listes Python.
  - Utilisez .shape pour obtenir les dimensions d'un tableau.

NUMPY CRÉATION DE TABLEAUX

- np.zeros((3, 3)) : Matrice de zéros de taille 3x3.
- np.ones((2, 4)) : Matrice de un de taille 2 lignes et 4 colonnes.
- np.random((3, 3)) : Matrice de valeurs aléatoires de taille 3x3.
- np.linspace(0, 1, 5) : Génère 5 valeurs entre 0 et 1.
- np.eye(4) : Matrice identité avec 1 en diagonale de taille 4x4

- Remarques importantes :
  - Personnalisation des formes de tableaux à l'aide de tuples.
  - Utile pour initialiser les poids dans l'apprentissage automatique

NUMPY OPÉRATIONS MATHÉMATIQUES

- Addition :
  - Array + 2 : ajoute 2 à tous les éléments
  - array1 + array2 : addition de tous les éléments
- Multiplication :
  - Array1 * array2
  - Np.dot(array1, array2)
  - Np.matmul(array1, array2)
- Moyenne : np.mean(array)
- Somme : np.sum(array)

- Remarques importantes :
  - Les opérations sont vectorisées, ce qui les rend plus rapides que les boucles Python.
  - Utilisez np.matmul pour la multiplication de matrices.

NUMPY INDEXATION ET DÉCOUPAGE

- Accès aux éléments :
  - Arr = np.array([1, 2, 3, 4, 5, 6])
    - Arr[0] = accès au premier élément
    - Arr[-1] = accès au dernier élément
- Découpage de tableaux :
    - Arr[1:4] = accès aux éléments 1 à 3
- Indexation multidimensionnelle :
  - Arr = np.array([[1, 2, 3], [4, 5, 6]])
    - A[0, 2] = accès au premier élément , 3ème colonne

- Remarques importantes :
  - L'index commence à 0

NUMPY MANIPULATION DES DIMENSIONS

- Reshape :
  - array.reshape((rows, cols)) : Convertit la taille d'une matrice
- Concaténation de tableaux :
  - np.concatenate((arr1, arr2))
- Découpage de tableaux :
  - Np.split(arr, 2)
- Transposer tableau :
  - array.T
  - Np.transpose(arr)

NUMPY ALGÈBRE LINÉAIRE

- Résolution de systèmes linéaires
  - A = np.array([[3, 1], [1, 2]]
  - B = np.array([9, 8])
  - X = np.linalg.solve(a, b)

NUMPY TRAVAILLER AVEC DES NOMBRES ALÉATOIRES

- Générations de nombres aléatoires :
  - Np.random.seed(42) : paramètrage de reproductibilité
  - Np.random.rand(3) : générer 3 nombres aléatoires
- Entiers aléatoires :
  - Np.random.randint(0, 10, size=5) : générer des entiers aléatoires entre 0 et 9

NUMPY – POUR LA DATA ANALYSE

- Nettoyage des données
  - Gérer les valeurs manquantes et les valeurs aberrantes.
- Transformation des données
  - Normaliser et mettre à l'échelle les données.
- Intégration
  - Utilisation avec Pandas pour l'analyse de données tabulaires.

NUMPY – APPLICATIONS RÉELLES DE NUMPY

- Science des données :
  - Traiter de grands ensembles de données pour les modèles d'apprentissage automatique.
- Traitement d'images :
  - Manipuler et analyser des données d'images sous forme de tableaux.
- Simulations physiques
  - Effectuer des calculs sur des données multidimensionnelles.
- Finance :
  - Modéliser et analyser les cours des actions et les tendances du marché.

NUMPY – ERREURS COURANTES ET SOLUTIONS

- Shape Mismatch :
  - Vérifier la forme des tableaux avant les opérations.
- TypeError :
  - Veillez à la cohérence des types de données dans les tableaux.
- MemoryError :
  - Utilisez des types de données plus petits ou découpez les grands ensembles de données.

NUMPY – BONNES PRATIQUES POUR L'UTILISATION DE NUMPY

- Utiliser des opérations vectorisées :
  - Éviter les boucles Python pour de meilleures performances.
- Documenter les transformations :
  - Garder une trace des formes et des manipulations des tableaux.
- Définir des graines aléatoires (seed) :
  - Assurer la reproductibilité des opérations aléatoires.

MODULE 2 : PANDAS

- Introduction à Pandas : Analyse et manipulation de données structurées avec DataFrame et Series.

- Caractéristiques clés :
  - Simplifie le travail avec des données structurées comme les tableaux et les séries chronologiques.
  - Prise en charge de l'importation et de l'exportation de données à partir de différents formats (CSV, Excel, SQL, etc.).
  - Offre des structures de données faciles à utiliser comme les DataFrames et les Séries.

https://www.w3schools.com/python/pandas/default.asp

MODULE 2 : PANDAS

- Cas pratique :
  - Nettoyage et prétraitement des données.
  - Analyse exploratoire des données (EDA).
  - Intégration avec des bibliothèques de visualisation et d'apprentissage automatique.

PANDAS INSTALLATION ET IMPORTATION

- Installation :
  - pip install pandas
  - Conda install pandas
- Importation : import pandas as pd

PANDAS DATAFRAME ET SERIES

- Tableau unidimensionnel étiqueté pouvant contenir n'importe quel type de données.
- Idéale pour les données à colonne unique.

- Créer une série :
  - pd.Series([0, 1, 2], index[“a”, “b”, “c”])

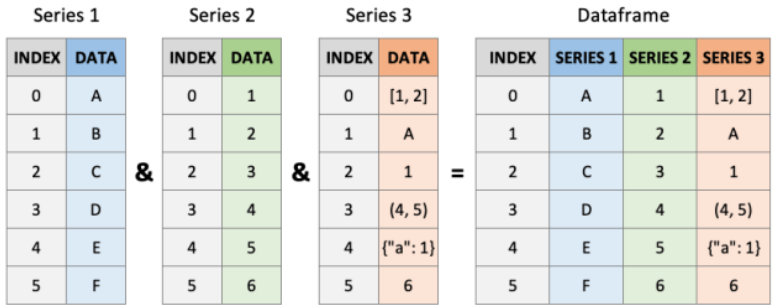
Series 1

INDEX	DATA
0	A
1	B
2	C
3	D
4	E
5	F

PANDAS DATAFRAME ET SERIES

- Structure de données bidimensionnelle et tabulaire.
- Peut contenir des types de données hétérogènes.

- Créer un DataFrame :
  - Data = {"Series1": ["A", "B"], "Series2": ["1", "2"], }
  - df = pd.DataFrame(data)



PANDAS – IMPORT ET EXPORT DES DONNÉES

- Import de la librairie
  - import pandas as pd
- Import des données
  - df = pd.read_csv('chemin/nom_du_fichier.csv')
  - df = pd.read_csv('chemin/nom_du_fichier.csv', sheet_name='Sheet_name_1')
  - Utiliser pd.read_excel() pour lire des fichiers .xls, .xlsx
- Export des données
  - df.to_csv('chemin/nom_fichier.csv')
  - df.to_excel('chemin/nom_fichier.xslx', sheet_name='Sheet_name_1')

PANDAS INDEXATION ET SÉLECTION

- Inspection des données :
  - Afficher la première ou la dernière ligne :
    - df.head(5)
    - df.tail(5)
  - Obtenir les noms des colonnes et les types de données :
    - df.info()
    - df.dtypes

- Résumé des statistiques :
  - df.describe()

PANDAS INDEXATION ET SÉLECTION

- Sélection de colonnes : df[“colonne”]
- Sélection des lignes par index :
  - Par position : df.iloc[0]
  - Par étiquette : df.loc[“index_label”]

	CustomerID	Name	Age	Region	Income Strata	Sales
	101	X1000	John	30	North	High	250
.loc[102] →	**102**	X1010	Ann	19	North	Medium	5000
	103	X1020	Joe	25	South	Medium	132
.iloc[3] →	**104**	X1030	Alice	53	East	Low	400
	105	X1040	Susan	38	South	Medium	780
	106	X1050	Bill	68	West	High	223

PANDAS INDEXATION ET SÉLECTION

- Sélectionner des colonnes :
  - df[[‘col1’, ‘col2’, ‘col3’]]
  - new_df = df[[‘col1’, ‘col2’, ‘col3’]]

- Remarque :
  - Le double [[ ]] permet de continuer à manipuler des datasets.
  - Le simple [ ] pour sortir des datasets et manipuler une colonne.

PANDAS INDEXATION ET SÉLECTION

- Sélection conditionnelle :
  - df[df['col1']<= valeur]
  - df[(df['col1']<= 500) & (df['col2']> 10)]
  - df[(df['col1']<= 500) |(df['col2']> 10)]
- Remarque :
  - Les mêmes opérateurs qu'en SQL:
    - <, <=, >, >=,
    - | (or), & (and),==,
    - isnull(), notnull(), ~(not)

PANDAS INDEXATION ET SÉLECTION

- Sélection conditionnelle par texte :
  - startswith()
    - df[df[“colonne”].astype(str).str.startswith(“texte”)]
  - endswith()
    - df[df[“colonne”].astype(str).str.endswith(“texte”)]
  - contains()
    - df[df[“colonne”].astype(str).str.contains(“texte”)]

- Remarque :
  - Ajouter astype(str) si la colonne n’est pas un string

PANDAS NETTOYAGE ET TRANSFORMATION DES DONNÉES

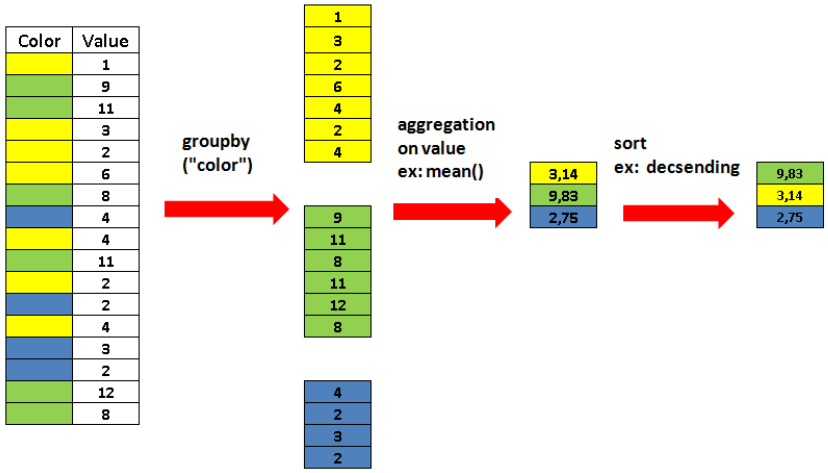
- Traitement des données manquantes
  - Supprimer les valeurs manquantes
    - df.dropna()
  - Remplir les valeurs manquantes
    - df.fillna(value)
- Renommer des colonnes
  - df.rename(columns={"ancienne_nom" : "nouveau_nom", inplace=True)
- Modification des types de données
  - df["nom_colonne"] = df["nom_colonne"].astype["int"]

PANDAS TRAVAILLER AVEC DES DATES ET DES HEURES

- Conversion en DateTime
  - df[“colonne_date”] = pd.todatetime(df[“date_column”])
- Extraction de composants de date
  - df[“year”] = df[“date_column”].dt.year
  - df[“month”] = df[“date_column”].dt.month
  - df[“day”] = df[“date_column”].dt.day

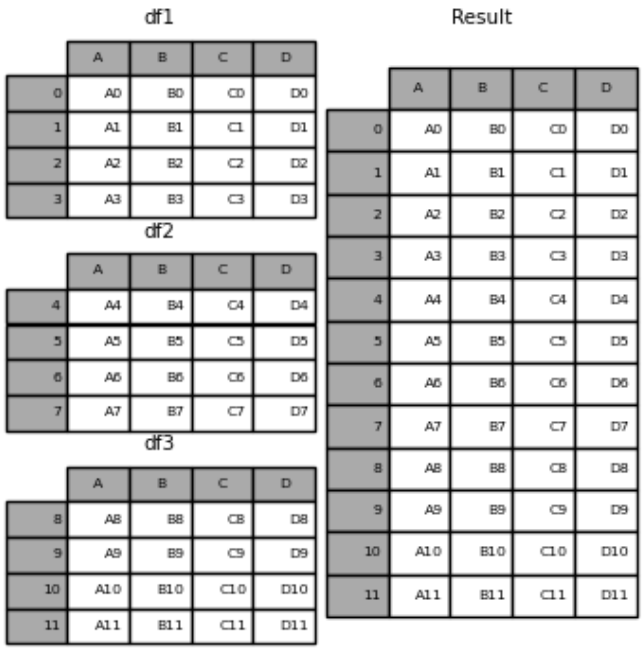
PANDAS – REGROUPEMENT ET AGGREGATION

- Regroupement des données :
  - Group = df.groupby(“color”)
- Aggregation :
  - Group_agg = Group[“value”].mean()



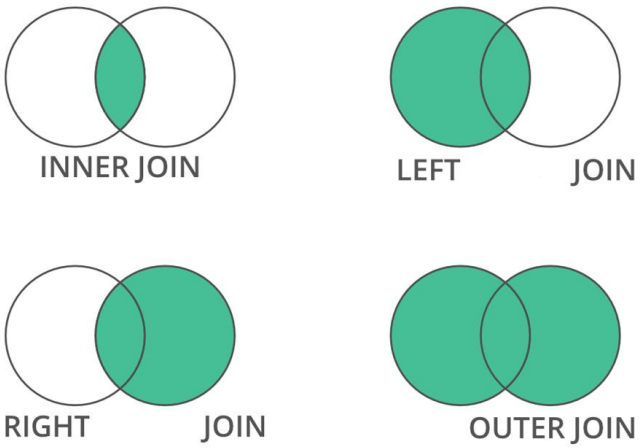
PANDAS FUSIONNER ET JOINDRE DES DONNÉES

- Concaténation de DataFrames
  - Pd.concat([df1, df2], axis=0)



PANDAS FUSIONNER ET JOINDRE DES DONNÉES

- Fusionner des DataFrames (jointure) :
  - Fusion = pd.merge(df1, df2, on="colonne_clé")
  - Fusion = pd.merge(df1, df2, on="colonne_clé", how="left")



PANDAS TRI DES DONNÉES

- Tri par colonne :
  - df_trie = df.sort_values(by="column_name", ascending=true)
- Tri par index :
  - df.sort_index(inplace=true)

PANDAS VISUALISATION AVEC PANDAS

- Visuels intégrés
  - df[“colonne”].plot(kind=“line”)
  - df.plot(kind=“bar”)
- Intégration avec Matplotlib
  - Import matplotlib.pyplot as plt
  - df[“colonne”].hist()
  - Plt.show()

PANDAS OPÉRATIONS AVANCÉES

- Table de pivot
  - df.pivot_table()

Date	City	Temperature
2023-01-01	New York	32
2023-01-01	Los Angeles	75
2023-01-02	New York	30
2023-01-02	Los Angeles	77

pivot

	Temperature	
City	Los Angeles	New York
Date		
2023-01-01	75	32
2023-01-02	77	30

PANDAS OPÉRATIONS AVANCÉES

- Appliquer des fonctions personnalisées
  - df[“nom”] = df[“nom”].apply(lambda x: x.upper())
  - df[“new_col”] = df[‘col”].apply(lambda x: np.nan if x < 90 else x)
- Remarque :
  - Cela évite de d’utiliser des boucles tout en gagnant en performance.

PANDAS STATISTIQUES DESCRIPTIVES

- Minimum : df['col'].min()
- Maximum : df['col'].max()
- Somme : df['col'].sum()
- Moyenne : df['col'].mean()
- Compte : df['col'].count()
- Nb de valeurs uniques : df['col'].unique()
- Corrélation : df.corr()

PANDAS – APPLICATIONS RÉELLES DE PANDAS

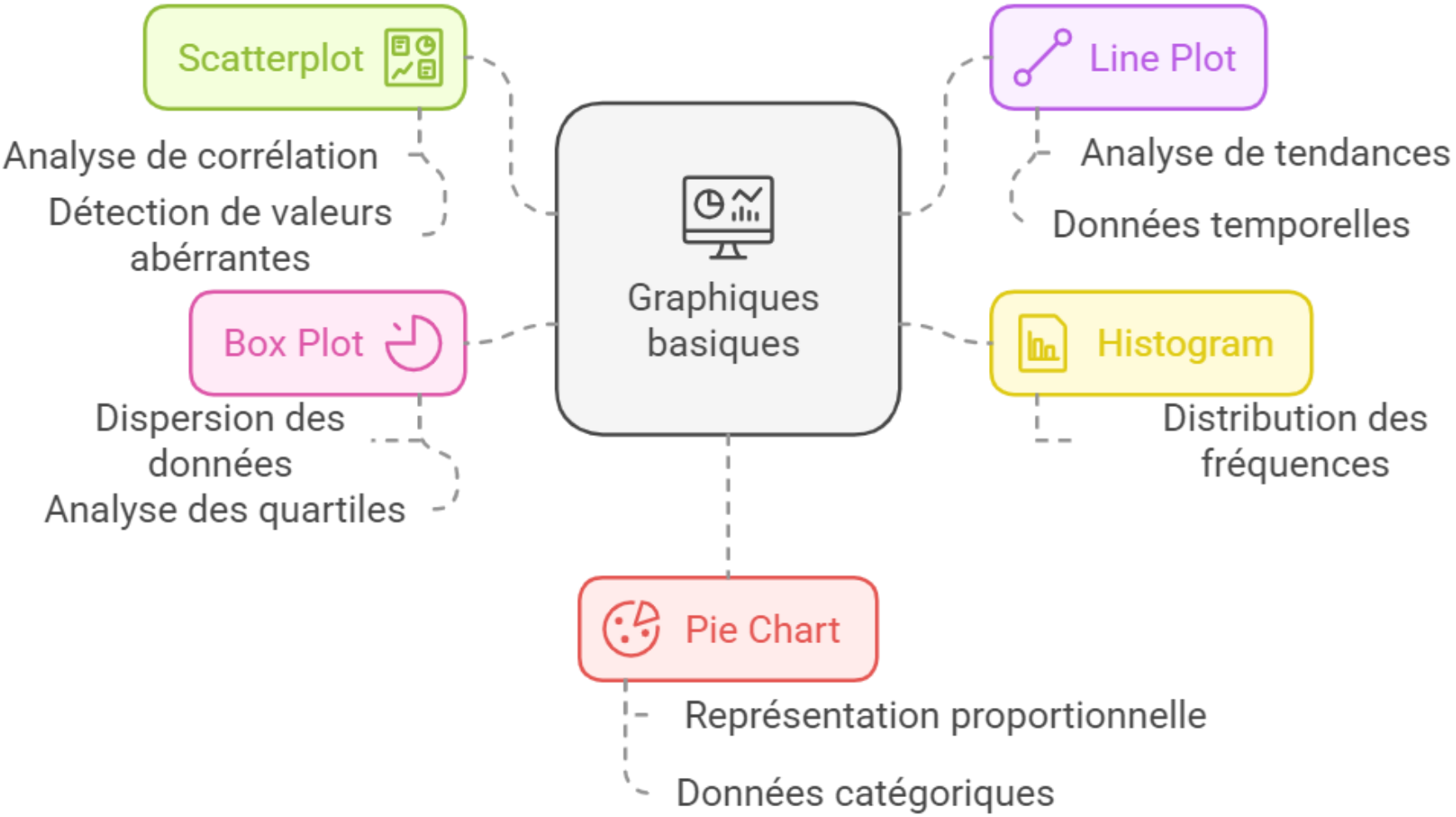
- Nettoyage de données :
  - Supprimer les doublons, gérer les valeurs manquantes et prétraiter les données.
- Analyse financière :
  - Traiter les données du marché boursier pour en tirer des tendances et des prédictions.
- WebScraping :
  - Analyser les données collectées sur les sites web à l'aide de bibliothèques comme BeautifulSoup.
- Apprentissage automatique :
  - Préparer les données pour l'entraînement des modèles à l'aide de bibliothèques comme Scikit-learn.

PANDAS – ERREURS COURANTES ET SOLUTIONS

- KeyError :
  - Assurez-vous que les étiquettes de colonne ou d'index existent.
- MemoryError (erreur de mémoire) :
  - Optimiser les opérations pour les grands ensembles de données en utilisant des chunks.
- SettingWithCopyWarning :
  - Utiliser .loc[] pour éviter l'indexation en chaîne.

PANDAS – BONNES PRATIQUES POUR L'UTILISATION DE PANDAS

- Optimiser les performances :
  - Utiliser des opérations vectorisées au lieu de boucles.
- Documenter le code :
  - Étiqueter les colonnes et décrire les transformations pour plus de clarté.
- Valider les données :
  - Vérifier les incohérences avant d'effectuer les opérations


Graphiques basiques
Scatterplot
Analyse de corrélation
Détection de valeurs abérrantes
Line Plot
Analyse de tendances
Données temporelles
Box Plot
Dispersion des données
Analyse des quartiles
Histogram
Distribution des fréquences
Pie Chart
Représentation proportionnelle
Données catégoriques

MODULE 3 : MATPLOTLIB

- Introduction à Matplotlib : Création de graphiques 2D personnalisés.

https://www.w3schools.com
/python/matplotlib_intro.asp

MATPLOTLIB INSTALLATION ET IMPORTATION

- Installation :
  - pip install matplotlib
  - Conda install matplotlib
- Importation : import matplotlib.pyplot as plt

MATPLOTLIB GRAPHIQUES SIMPLES

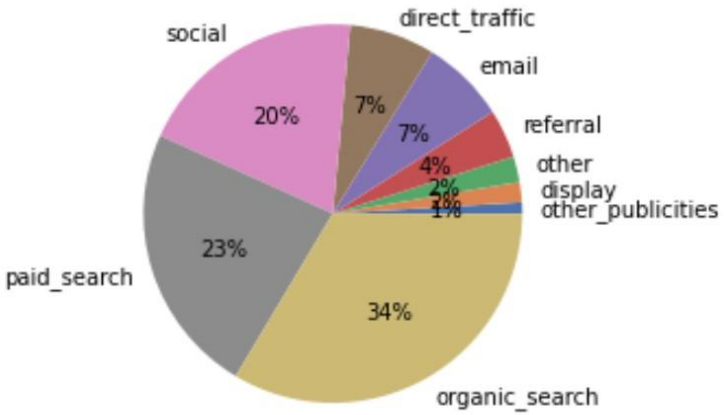
- Lignes : plt.plot(x, y)
- Barres : plt.bar(x, y)
- Nuages de points : plt.scatter(x, y)

MATPLOTLIB PERSONNALISATION

- Titre : plt.title('Titre')
- Légendes : plt.legend(['label1', 'label2'])
- Sauvegarde : plt.savefig('nom_fichier.png')

MATPLOTLIB EXEMPLE

- plt.pie(data=market, x=nb_origin2['n'], labels=nb_origin2['origin'], autopct='%.0f%%')



MODULE 4 : SEABORN

- Introduction à Seaborn : Création de graphiques élégants avec des thèmes prédéfinis.

SEABORN INSTALLATION ET IMPORTATION

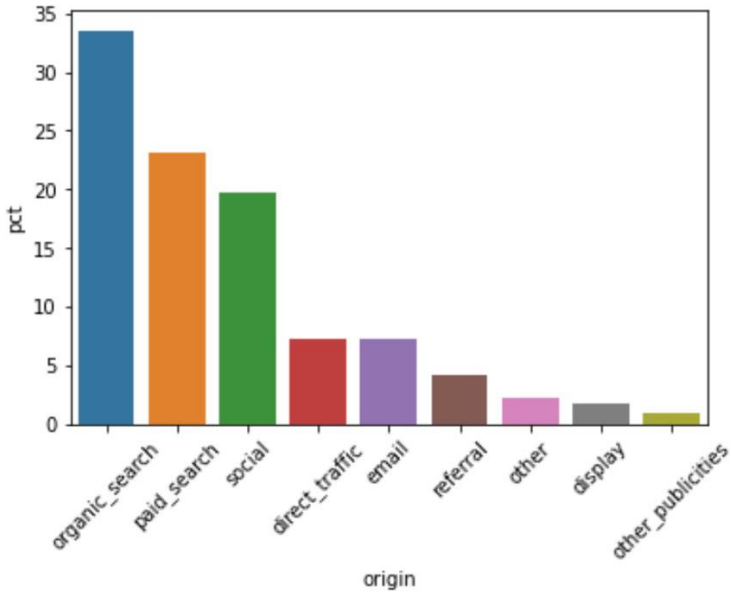
- Installation :
  - pip install seaborn
  - Conda install seaborn
- Importation : import seaborn as sns

SEABORN GRAPHIQUES DISPONIBLES

- Nuage de points : sns.scartterplot()
- Courbes : sns.lineplot()
- Graphiques à barres : sns.barplot()
- Boites à moustache : sns.boxplot()
- Distribution : sns.histplot(data), sns.kdeplot(data)

SEABORN EXEMPLE

- sb.barplot(data=nb_origin, x="origin", y="pct"). set(title="???", xlabel='date', ylabel='leads générés')



SEABORN HEATMAPS ET CORRÉLATIONS

- Exemple : sns.heatmap(df.corr(), annot=True)

	A	B	C	D
A	1	0.79552	-0.519566	-0.654733
B	0.79552	1	-0.764484	-0.0638539
C	-0.519566	-0.764484	1	-0.0656217
D	-0.654733	-0.0638539	-0.0656217	1

ÉTAPES SUIVANTES

- Pratiquer les concepts abordés avec des cas concrets.
- Approfondir les bibliothèques avec des projets avancés.
- Poser vos questions pour éclaircir les concepts difficiles.
- Parcourir de nouvelles ressources.

PYTHON ET LES BDD

- Installation :
  - pip install nom_bibliothèque (psycopg2-binary ou pymysql ou mariadb ou sqlite3)
  - conda install nom_bibliothèque (psycopg2-binary ou pymysql ou mariadb ou sqlite3)
- Importation :
  - import nom_bibliothèque (psycopg2-binary ou pymysql ou mariadb ou sqlite3)

PYTHON ET LES BDD : SQLITE

```python
import sqlite3

# Connexion ou création d'une BDD locale
conn = sqlite3.connect("ma_base_sqlite.db")
cursor = conn.cursor()

# Création de table
cursor.execute("""
CREATE TABLE IF NOT EXISTS utilisateurs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nom TEXT NOT NULL,
    age INTEGER
)
""")

# Insertion
cursor.execute("INSERT INTO utilisateurs (nom, age) VALUES (?, ?)",
               ("Alice", 30))

# Insertion multiple
utilisateurs = [
    ("Eve", 28),
    ("Frank", 32),
    ("Grace", 29)
]
cursor.executemany("INSERT INTO utilisateurs (nom, age) VALUES (?, ?)", utilisateurs)

# Mise à jour
cursor.execute("UPDATE utilisateurs SET age = ? WHERE nom = ?", (31, "Alice"))

# Suppression
cursor.execute("DELETE FROM utilisateurs WHERE nom = ?", ("Frank",))

# Requête
cursor.execute("SELECT * FROM utilisateurs")
resultats = cursor.fetchall()

print(resultats)

#
for ligne in resultats:
    print(ligne)

conn.commit()
```

Template

```
import sqlite3

# Connexion ou création d'une BDD locale
conn = sqlite3.connect("ma_base_sqlite.db")
cursor = conn.cursor()

# Création de table
cursor.execute("""
CREATE TABLE IF NOT EXISTS utilisateurs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nom TEXT NOT NULL,
    age INTEGER
)
""")

# Insertion
cursor.execute("INSERT INTO utilisateurs (nom, age) VALUES (?, ?)",
               ("Alice", 30))

# Insertion multiple
utilisateurs = [
    ("Eve", 28),
    ("Frank", 32),
    ("Grace", 29)
]
cursor.executemany("INSERT INTO utilisateurs (nom, age) VALUES (?, ?)", utilisateurs)

# Mise à jour
cursor.execute("UPDATE utilisateurs SET age = ? WHERE nom = ?", (31, "Alice"))

# Suppression
cursor.execute("DELETE FROM utilisateurs WHERE nom = ?", ("Frank",))

# Requête
cursor.execute("SELECT * FROM utilisateurs")
resultats = cursor.fetchall()

print(resultats)

conn.commit()

cursor.close()
conn.close()
```

https://www.sqlitetutorial.net/sqlite-python/

PYTHON ET LES BDD : POSTGRES

```python
import psycopg2

# Connexion à PostgreSQL
conn = psycopg2.connect(
    host="localhost",
    user="postgres",
    password="mot_de_passe",
    dbname="postgres"  # pour créer une autre BDD
)
conn.autocommit = True
cursor = conn.cursor()

# Création d'une nouvelle BDD (si elle n'existe pas déjà)
cursor.execute("SELECT 1 FROM pg_database WHERE datname = 'ma_bdd'")
exists = cursor.fetchone()
if not exists:
    cursor.execute("CREATE DATABASE ma_bdd")

# Création de table
cursor.execute("""
CREATE TABLE IF NOT EXISTS utilisateurs (
    id SERIAL PRIMARY KEY,
    nom VARCHAR(100),
    age INT
)
""")

# Insertion
cursor.execute("INSERT INTO utilisateurs (nom, age) VALUES (%s, %s)", ("Bob", 25))

# Insertion multiple
utilisateurs = [
    ("Alice", 30),
    ("Eve", 28),
    ("Frank", 32)
]
cursor.executemany("INSERT INTO utilisateurs (nom, age) VALUES (%s, %s)", utilisateurs)

# Update
cursor.execute("UPDATE utilisateurs SET age = %s WHERE nom = %s", (26, "Bob"))

# Suppression
cursor.execute("DELETE FROM utilisateurs WHERE nom = %s", ("Bob",))

# Requête
cursor.execute("SELECT * FROM utilisateurs")
print(cursor.fetchall())

for ligne in cursor.fetchall():
    print(ligne)

conn.commit()

cursor.close()
```

Template

```
import psycopg2

# Connexion à PostgreSQL
conn = psycopg2.connect(
    host="localhost",
    user="postgres",
    password="mot_de_passe",
    dbname="postgres" # pour créer une autre BDD
)
conn.autocommit = True
cursor = conn.cursor()

# Création d'une nouvelle BDD (si elle n'existe pas déjà)
cursor.execute("SELECT 1 FROM pg_database WHERE datname = 'ma_bdd'")
exists = cursor.fetchone()
if not exists:
    cursor.execute("CREATE DATABASE ma_bdd")

# Création de table
cursor.execute("""
CREATE TABLE IF NOT EXISTS utilisateurs (
    id SERIAL PRIMARY KEY,
    nom VARCHAR(100),
    age INT
)
""")

# Insertion
cursor.execute("INSERT INTO utilisateurs (nom, age) VALUES (%s, %s)", ("Bob", 25))

# Insertion multiple
utilisateurs = [
    ("Alice", 30),
    ("Eve", 28),
    ("Frank", 32)
]
cursor.executemany("INSERT INTO utilisateurs (nom, age) VALUES (%s, %s)", utilisateurs)

# Update
cursor.execute("UPDATE utilisateurs SET age = %s WHERE nom = %s", (26, "Bob"))

# Suppression
cursor.execute("DELETE FROM utilisateurs WHERE nom = %s", ("Bob",))

# Requête
cursor.execute("SELECT * FROM utilisateurs")
print(cursor.fetchall())

for ligne in cursor.fetchall():
    print(ligne)

conn.commit()

cursor.close()
conn.close()
```

https://www.psycopg.org/docs/usage.html

PYTHON ET LES BDD : MYSQL

```python
import pymysql

# Connexion à MySQL
conn = pymysql.connect(
    host="localhost",
    user="root",
    password="mot_de_passe"
)
conn.autocommit(True)
cursor = conn.cursor()

# Création de la BDD
cursor.execute("CREATE DATABASE IF NOT EXISTS ma_bdd_mysql")
cursor.execute("USE ma_bdd_mysql")

# Création de table
cursor.execute("""
CREATE TABLE IF NOT EXISTS utilisateurs (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(100),
    age INT
)
""")

# Insertion
cursor.execute("INSERT INTO utilisateurs (nom, age) VALUES (%s, %s)", ("Claire", 28))

# Insertion multiple
utilisateurs = [
    ("David", 35),
    ("Eve", 28),
    ("Frank", 32)
]
cursor.executemany("INSERT INTO utilisateurs (nom, age) VALUES (%s, %s)", utilisateurs)

# Update
cursor.execute("UPDATE utilisateurs SET age = %s WHERE nom = %s", (29, "Claire"))

# Suppression
cursor.execute("DELETE FROM utilisateurs WHERE nom = %s", ("Claire",))

# Requête
cursor.execute("SELECT * FROM utilisateurs")
print(cursor.fetchall())

for ligne in cursor.fetchall():
   print(ligne)

conn.commit()
cursor.close()
conn.close()
```

Template

```
import pymysql

# Connexion à MySQL
conn = pymysql.connect(
    host="localhost",
    user="root",
    password="mot_de_passe"
)
conn.autocommit(True)
cursor = conn.cursor()

# Création de la BDD
cursor.execute("CREATE DATABASE IF NOT EXISTS ma_bdd_mysql")
cursor.execute("USE ma_bdd_mysql")

# Création de table
cursor.execute("""
CREATE TABLE IF NOT EXISTS utilisateurs (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(100),
    age INT
)
""")

# Insertion
cursor.execute("INSERT INTO utilisateurs (nom, age) VALUES (%s, %s)", ("Claire", 28))

# Insertion multiple
utilisateurs = [
    ("David", 35),
    ("Eve", 28),
    ("Frank", 32)
]
cursor.executemany("INSERT INTO utilisateurs (nom, age) VALUES (%s, %s)", utilisateurs)

# Update
cursor.execute("UPDATE utilisateurs SET age = %s WHERE nom = %s", (29, "Claire"))

# Suppression
cursor.execute("DELETE FROM utilisateurs WHERE nom = %s", ("Claire",))

# Requête
cursor.execute("SELECT * FROM utilisateurs")
print(cursor.fetchall())

for ligne in cursor.fetchall():
   print(ligne)

conn.commit()
cursor.close()
conn.close()
```

https://www.w3schools.com/python/python_mysql_getstarted.asp

PYTHON ET LES BDD : MARIADB

```python
import mariadb

# Connexion à MariaDB
conn = mariadb.connect(
    user="root",
    password="mot_de_passe",
    host="localhost"
)
conn.autocommit = True
cursor = conn.cursor()

# Création de la BDD
cursor.execute("CREATE DATABASE IF NOT EXISTS ma_bdd_mariadb")
cursor.execute("USE ma_bdd_mariadb")

# Création de la table
cursor.execute("""
CREATE TABLE IF NOT EXISTS utilisateurs (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(100),
    age INT
)
""")

# Insertion
cursor.execute("INSERT INTO utilisateurs (nom, age) VALUES (?, ?)", ("David", 35))

# Insertion multiple
utilisateurs = [
    ("Eve", 28),
    ("Frank", 32)
]
cursor.executemany("INSERT INTO utilisateurs (nom, age) VALUES (?, ?)", utilisateurs)

# Update
cursor.execute("UPDATE utilisateurs SET age = ? WHERE nom = ?", (36, "David"))

# Suppression
cursor.execute("DELETE FROM utilisateurs WHERE nom = ?", ("David",))

# Requête
cursor.execute("SELECT * FROM utilisateurs")
print(cursor.fetchall())

for ligne in cursor.fetchall():
   print(ligne)

conn.commit()
cursor.close()
conn.close()
```

Template

```
import mariadb

# Connexion à MariaDB
conn = mariadb.connect(
    user="root",
    password="mot_de_passe",
    host="localhost"
)
conn.autocommit = True
cursor = conn.cursor()

# Création de la BDD
cursor.execute("CREATE DATABASE IF NOT EXISTS ma_bdd_mariadb")
cursor.execute("USE ma_bdd_mariadb")

# Création de la table
cursor.execute("""
CREATE TABLE IF NOT EXISTS utilisateurs (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(100),
    age INT
)
""")

# Insertion
cursor.execute("INSERT INTO utilisateurs (nom, age) VALUES (?, ?)", ("David", 35))

# Insertion multiple
utilisateurs = [
    ("Eve", 28),
    ("Frank", 32)
]
cursor.executemany("INSERT INTO utilisateurs (nom, age) VALUES (?, ?)", utilisateurs)

# Update
cursor.execute("UPDATE utilisateurs SET age = ? WHERE nom = ?", (36, "David"))

# Suppression
cursor.execute("DELETE FROM utilisateurs WHERE nom = ?", ("David",))

# Requête
cursor.execute("SELECT * FROM utilisateurs")
print(cursor.fetchall())

for ligne in cursor.fetchall():
   print(ligne)

conn.commit()
cursor.close()
conn.close()
```

https://mariadb.com/resources/blog/how-to-connect-python-programs-to-mariadb/