AlgoÉT$

Python 101

Science des données

Le but du Workshop

Introduction
Découvrez le monde de la science des données et les APIs.

Public Cible
Du débutant complet aux programmeurs expérimentés.

Objectif
Acquérir des compétences pratiques et applicables.

Langage
Tout sera enseigné en Python.

Table de matière

Les Fondamentaux de Python: Nous commencerons par les bases pour ceux qui sont nouveaux dans ce langage de programmation puissant.

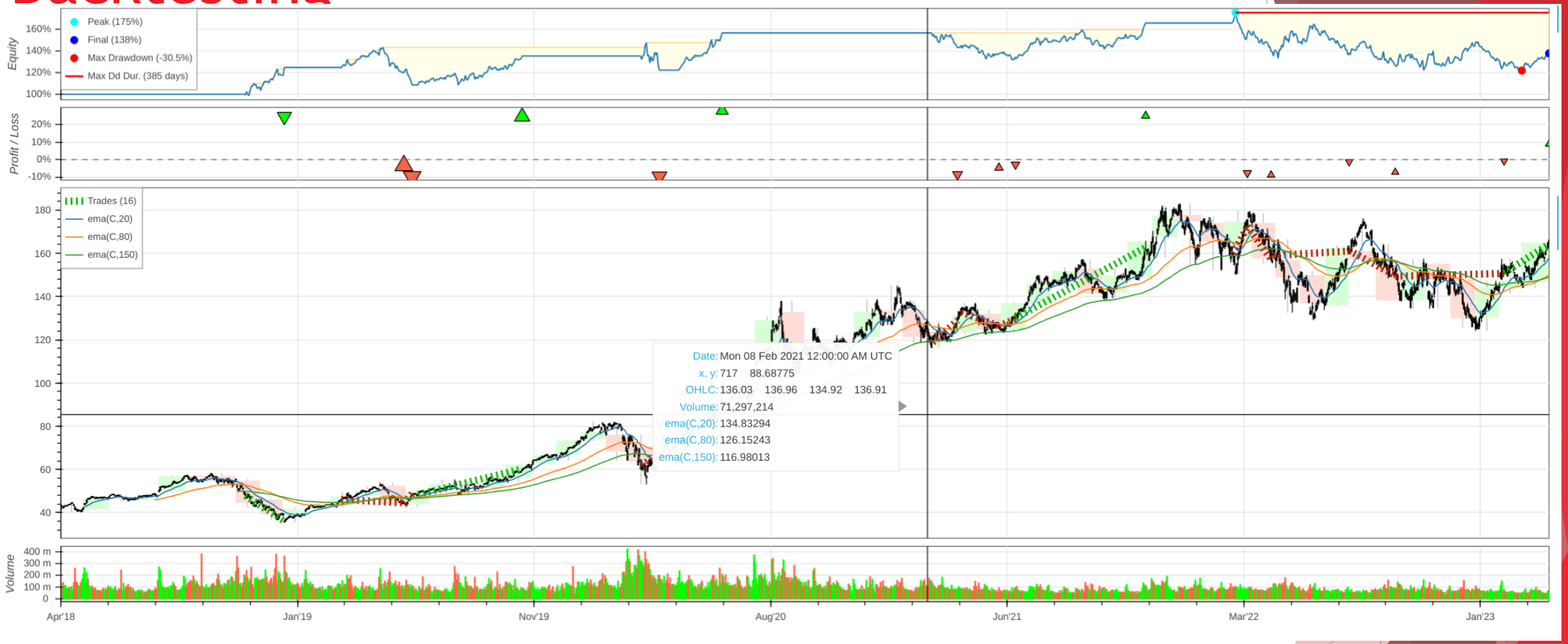
APIs et Sources de Données: Apprenez à interagir avec des APIs pour récupérer des données et les utiliser dans vos projets.

Science des Données avec Pandas: Découvrez comment nettoyer, manipuler et analyser des données avec Pandas.

Visualisation de Données: Utilisez des outils comme Matplotlib, Seaborn et Plotly pour créer des visualisations de données époustouflantes.

Project Indicateurs Techniques en Finance: Découvrez comment utiliser des indicateurs techniques comme la Moyenne Mobile Exponentielle dans l'analyse financière.

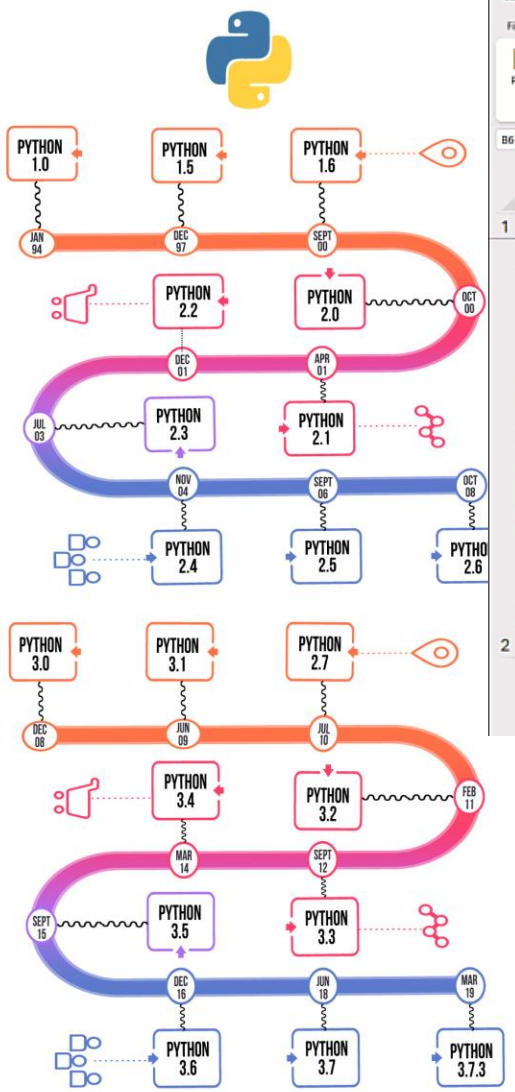
Backtesting

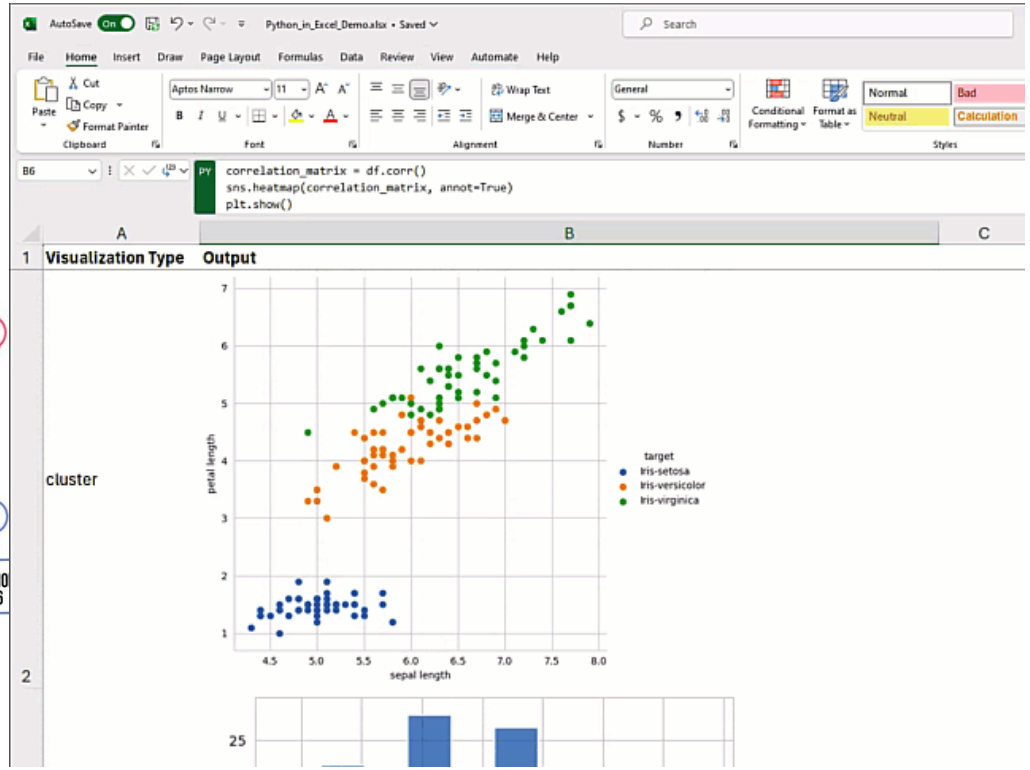


- https://github.com/AlgoETS/CookiecutterStrategie
- https://github.com/AlgoETS/Senators
- https://github.com/AlgoETS/BatchBacktesting
- https://github.com/AlgoETS/MovingAverage

Environment

- Local
- Cloud





Mojo🔥

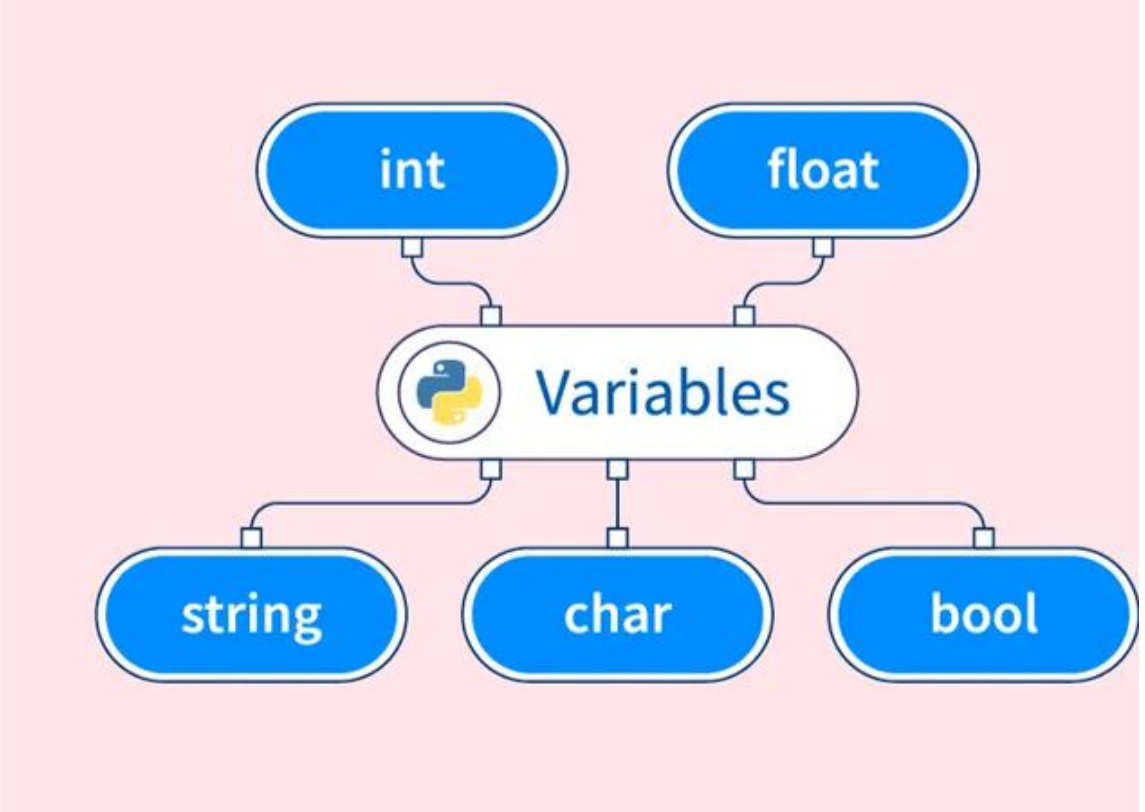
Montréal Python

Histoire

Indentation en Python

- 4 space
- L'indentation en Python remplace les accolades {} utilisées dans d'autres langages pour définir les blocs de code. Elle rend le code plus lisible et facile à comprendre.
- Une mauvaise indentation entraîne une erreur de syntaxe, ce qui empêche le code de s'exécuter.

Type



- Bien sûr, voici une explication des types de données courants en Python :
- 1. str (String) : Utilisé pour représenter du texte. Entouré de guillemets simples ou doubles.

Exemple : "Bonjour", 'Python'

- 2. int (Integer) : Utilisé pour représenter des nombres entiers.

Exemple : 5, 3, 0

- 3. float : Utilisé pour représenter des nombres réels, c'estàdire des nombres avec des décimales.
- Exemple : 3.14, 0.001, 2.0

```python
# Entier (Integer)
age = 25

# Flottant (Float)
prix = 15.99

# Chaîne de caractères (String)
prenom = "Jean"
message = "Bonjour tout le monde!"

# Booléen (Boolean)
est_actif = False
est_etudiant = True

# Liste (List)
nombres = [1, 2, 3, 4, 5]
amis = ["Alice", "Bob", "Charlie"]

# Tuple (Immutable List)
coordonnees = (45.4215, -75.6972)

# Dictionnaire (Dictionary)
personne = {
    "nom": "Dupont",
    "age": 30,
    "email": "dupont@email.com"
}

# Ensemble (Set)
fruits = {"pomme", "banane", "orange

# None (Aucune valeur)
```

Variable

- Les variables sont des conteneurs pour stocker des valeurs de données.
- Python n'a pas de commande pour déclarer une variable.

List

List : Une collection ordonnée et modifiable d'éléments. Les listes sont définies en entourant les éléments de crochets.

[1, 2, 3], ['a', 'b', 'c']

Tuple : Une collection ordonnée mais non modifiable d'éléments. Les tuples sont définis en entourant les éléments de parenthèses.

(1, 2, 3), ('x', 'y', 'z')

Dictionaire

dictionary (Dictionnaire) : Une collection non ordonnée, modifiable et indexée d'éléments. Les dictionnaires sont définis avec des clés et des valeurs.

{'nom': 'Jean', 'âge': 30}

set (Ensemble) : Une collection non ordonnée et non indexée d'éléments uniques. Les ensembles sont définis en entourant les éléments avec des accolades, mais sans clés.

{1, 2, 3}, {'a', 'b', 'c'}

Condtion Booléens

Égal à :

(a == b)

Différent de :

(a != b)

Inférieur à :

(a < b)

Inférieur ou égal à :

(a <= b)

Supérieur ou égal à :

(a >= b)

Combinaision boolean

Opérateurs Booléens

- and:
  - Retourne True si les deux expressions sont vraies.
- if a > 10 and b < 20:
- or :
  - Retourne True si au moins une des expressions est vraie.
- if a > 10 or b < 20:

Opérateur Ternaire

- Permet de réaliser une affectation conditionnelle de manière concise.
- y = "Pair" if x % 2 == 0 else "Impair"

```python
# Vérifier si une variable est None
if valeur is None:
    print("La variable 'valeur' est None.")

# Vérifier si une variable n'est pas None
if valeur is not None:
    print("La variable 'valeur' n'est pas None.")

# Utilisation de l'opérateur de non-égalité !=
if x != y:
    print(f"x ({x}) et y ({y}) sont différents.")

# Utilisation de l'opérateur >= (supérieur ou égal)
if x >= y:
    print(f"x ({x}) est supérieur ou égal à y ({y}).")

# Utilisation de l'opérateur <= (inférieur ou égal)
if x <= y:
    print(f"x ({x}) est inférieur ou égal à y ({y}).")

# Combiner plusieurs conditions avec 'and' et 'or'
if x > 10 and y < 30:
    print("x est supérieur à 10 et y est inférieur à 30.")

if x > 10 or y < 30:
    print("Soit x est supérieur à 10, soit y est inférieur à 30, soit les deux.")

# Utilisation de 'elif' et 'else' pour des conditions multiples
if x > y:
    print("x est supérieur à y.")
elif x < y:
    print("x est inférieur à y.")
else:
    print("x est égal à y.")

# Utilisation de l'opérateur Walrus (Python 3.8+)
if (n := x * 2) > y:
    print(f"n ({n}) est supérieur à y ({y})")
```

Fonctions

```python
def saluer(nom):
    """Salut toi !"""
    return f"Bonjour, {nom}"
```

```python
# 0,1,2,3,4
for i in range(5):
    print(i)

noms = ["Alice", "Bob", "Charlie"]

for indice, nom in enumerate(noms):
    print(f"Indice: {indice}, Nom:
{nom}")
# basic
i = 0
while i < 3:
    print(i)
    i += 1

# break
i = 0
while i < 5:
    if i == 2:
        break
    print(i)
    i += 1

# continue
i = 0
while i < 5:
    i += 1
    if i == 3:
        continue
    print(i)
```

Boucles

- **break** : Termine la boucle immédiatement et passe à la ligne suivant la boucle.
- **continue** : Passe à la prochaine itération de la boucle, en ignorant les lignes qui suivent **continue** dans le bloc de code.

For

Range

- Génère une séquence de nombres que l'on peut utiliser dans une boucle for.
  - for i in range(5): # 0, 1, 2, 3, 4
    - print(i)

Enumerate

- Utilisée pour obtenir l'indice et la valeur d'éléments dans une séquence.
  - for indice, valeur in enumerate(['a', 'b', 'c']):
    - print(indice, valeur)

Iterateur

- for lettre in "Python":
  - print(lettre)

```python
class Animal:
    def parler(self):
        pass

class Chien(Animal):
    def parler(self):
        return "Ouaf"

class Chat(Animal):
    def parler(self):
        return "Miaou"

class Vache(Animal):
    def parler(self):
        return "Meuh"

# Utilisation du polymorphisme
def faire_parler(animal):
    print(animal.parler())

# Test du polymorphisme
chien = Chien()
chat = Chat()
vache = Vache()

faire_parler(chien)  # Affiche "Ouaf
faire_parler(chat)   # Affiche "Miao
faire_parler(vache)  # Affiche "Meuh
```

Classes

- **Classe** : Un modèle pour créer des objets.
- **Objet** : Une instance d'une classe.
- **Héritage** : La capacité d'une classe à hériter des propriétés et des méthodes d'une autre classe.
- **Polymorphisme** : La capacité d'objets de différentes classes à être traités comme des objets d'une classe commune.
- **Redéfinition de Méthode: Method Overriding**

Gestion des exceptions

```python
# Définition de l'exception personnalisée
class DivisionParZeroError(Exception):
    def __init__(self, message="Impossible de diviser par zéro."):
        self.message = message
        super().__init__(self.message)

# Utilisation de l'exception personnalisée
try:
    x = 10
    y = 0
    if y == 0:
        raise DivisionParZeroError()
    print(x / y)
except DivisionParZeroError as e:
    print(e)
```

API

- Interface de Programmation d'Application
- Pour obtenir des données météorologiques: API météo en envoyant une requête HTTP.
- L'API retourne ensuite les données en format JSON ou XML que l'application peut utiliser.

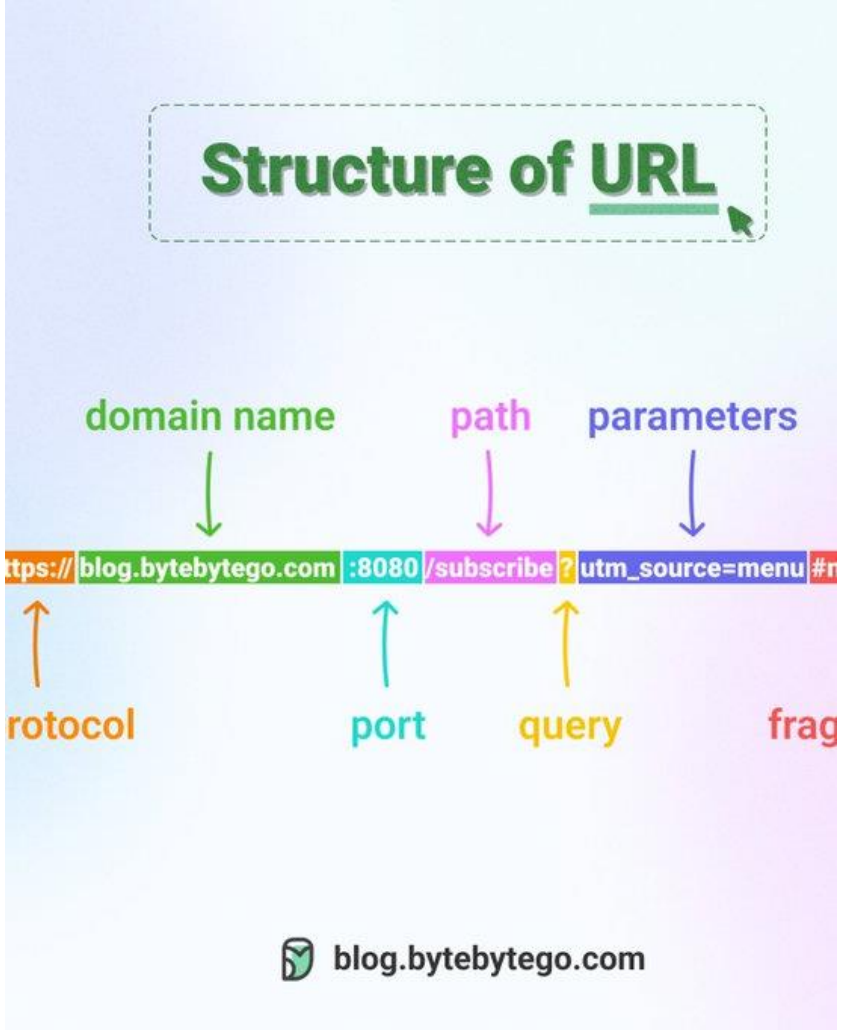
JSON (JavaScript Object Notation)

- Il est couramment utilisé pour échanger des données entre un serveur et une application web.
- Les données sont représentées en paires clé-valeur.
  - Les objets sont entre accolades **{}**.
- Les tableaux sont entre crochets **[]**.

```
{
  "nom": "Jean",
  "age": 30,
  "email": "jean@email.com",
  "estActif": true,
  "compétences": ["Python", "Java"]
}
```

URL (Uniform Resource Locator)

- **Protocole**
  - Mode de communication (**http**, **https**)
- **Domaine**
  - L'adresse du site web (**www.exemple.com**)
- **Port**
  - Numéro de port pour la connexion (optionnel, **:8080**).
- **Chemin (Path)**
  - Le chemin vers une ressource spécifique sur le serveur (**/page1**).
- **Paramètres (Query)**
  - Paramètres supplémentaires envoyés au serveur (**?id=123**).
- **Fragment**
  - Un identificateur pour une partie spécifique d'une page (**#section1**).

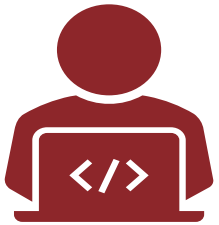


Type de serveur

Top 6 Most Popular API Architecture Styles ByteByteGo.com

Style	Illustration	Use Cases
SOAP	XML	XML-based for enterprise applications
RESTful	Resource	Resource-based for web servers
GraphQL		Query language reduce network load
gRPC	abc abc	High performance for microservices
WebSocket	push	Bi-directional for low-latency data exchange
Webhook	async	Asynchronous for event-driven application

HTTP



HTTP Verbs

GET: Retrieve data from the server.

POST: Submit data to the server.

PUT: Update data on the server.

DELETE: Remove data from the server.

HTTP Status Codes

200 OK: Successful GET or PUT.

201 Created: Successful POST.

204 No Content: Successful DELETE.

400 Bad Request: Client error.

401 Unauthorized: Authentication failed.

404 Not Found: Resource not found.

500 Internal Server Error: Server error.

API

```python
import requests
import json

url = "https://jsonplaceholder.typicode.com/todos/1"
response = requests.get(url)
data = json.loads(response.text)


import httpx

async def fetch_data():
    async with httpx.AsyncClient() as client:
        response = await
client.get('https://jsonplaceholder.typicode.com/todos/1')
    return response.json()
```

```python
import requests
import json

# Replace 'YOUR_API_KEY' with your actual API key
url = "https://financialmodelingprep.com/api/v3/income-statement/AAPL?limit=120&apikey=YOUR_API_KEY"
response = requests.get(url)
data = json.loads(response.text)

# Print revenue and net income for the most recent fiscal year
print(f"Revenue: {data[0]['revenue']}")
print(f"Net Income: {data[0]['netIncome']}")
```

FMP

Search for endpoint ⌘+F

Changelog
Status
Google Sheets Integration
Excel Add-on

STOCK FUNDAMENTALS

Financial Statements List
Financial Statements Annual/Quarter
Sales Revenue By Segments New
Revenue Geographic by Segments New
Financial Statements As Reported
International Filings Annual/Quarter
List of dates and links
Annual Reports on Form 10-K
Quarterly Earnings Reports
Shares Float
SEC RSS Feeds Real-time
Earning Call Transcript
SEC Filings
Rss feed 8K (Important Events)
Company Notes due New

STOCK FUNDAMENTALS ANALYSIS

Financial Ratios Annual/Quarter
Stock Financial scores New

Home Documentation Pricing Extras Blog Contact Sign

Financial Modeling Prep API Documentation

Getting Started

- We are the most accurate **financial data API** out there.
- To make it simple we are a **free stock API** | **historical data API** | **financial statements API**
- We update our financial statements in real time, every statements is audited, standardized, and up to date.
- We cover NYSE, NASDAQ, AMEX, EURONEXT, TSX, INDEXES, ETFs, MUTUAL FUNDS, FOREX and CRYPTO.
- We have **real time stock price**, we cover the fundamental data part of the stocks via providing income statement, balance sheet statement and cashfl **and annually**.
- We have **1 minute, 15 minutes, 30 minutes, 1 hour** and daily historical stock prices.
- Attribution is required for all users. It is as simple as putting "**Data provided by Financial Modeling Prep**" somewhere on your site or app and linking https://financialmodelingprep.com/developer/docs/. In case of limited screen space, or design constraints, the attribution link can be included in your te

Get your Free API Key Today!

Get my API KEY here

Panda

- DataFrame
- Simple
- Analyse Rapide
- Import & Export
- Norme

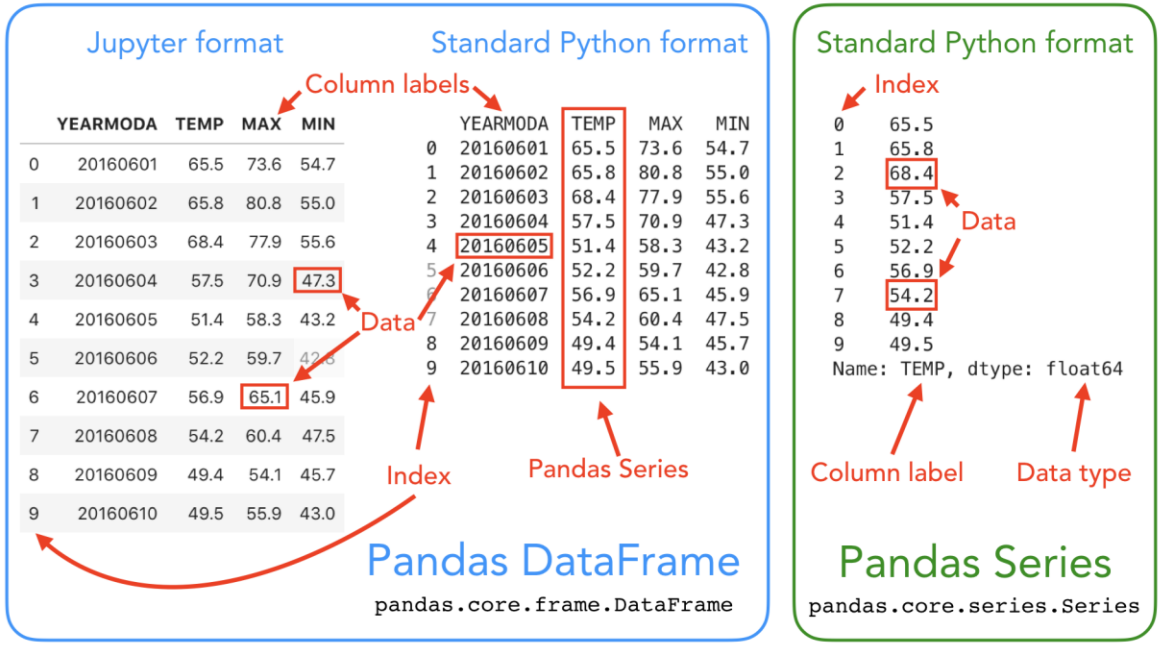
DataFrame

- Structure Tabulaire
- Trier, Filtrer, Agréger
- Intégration avec Matplotlib
- Column, Row

```python
# Load our necessary libraries
import pandas as pd
import numpy as np
```

```python
# Create data into CSV (that we'll import later on)
# Let's say the data is for a veterinarian keeping tabs on his clients.

raw_data = {'pet_name': ['Woof', 'Chester', 'Rex', 'Mystery', 'Pumpkin'],
        'pet_last_name': ['Smith', 'Kim', "", 'Taylor', ""],
        'good_pet_score': [96, 34, 89, 92, 79],
        'type': ['dog', 'cat', 'mini-dinosaur', "unknown", "bird"],
        'amount_owed': ["5000", "9,000", 570, 622, 190]}
df = pd.DataFrame(raw_data, columns = ['pet_name', 'pet_last_name', 'good_pet_score', 'type', 'amount_owed'])
df
```

	pet_name	pet_last_name	good_pet_score	type	amount_owed
0	Woof	Smith	96	dog	5000
1	Chester	Kim	34	cat	9,000
2	Rex		89	mini-dinosaur	570
3	Mystery	Taylor	92	unknown	622
4	Pumpkin		79	bird	190



Panda

pandas

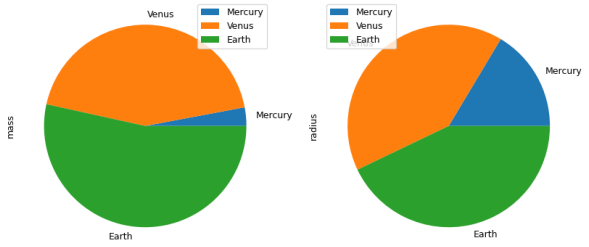
$$y_i t = \beta' x_{it} + \mu_i + \epsilon_{it}$$

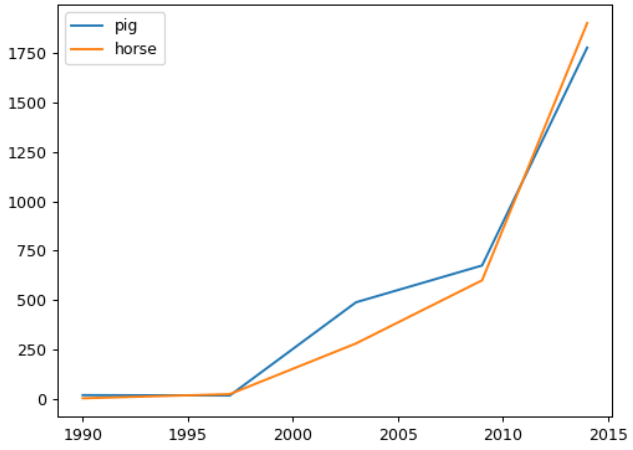
	BandName	WavelengthMax	WavelengthMin
0	CoastalAerosol	450	430
1	Blue	510	450
2	Green	590	530
3	Red	670	640
4	NearInfrared	880	850
5	ShortWaveInfrared_1	1650	1570
6	ShortWaveInfrared_2	2290	2110
7	Cirrus	1380	1360

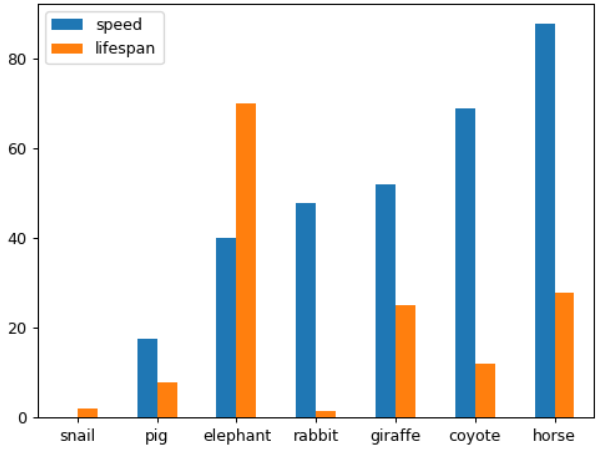
```python
# Lire un fichier CSV dans un DataFrame
df = pd.read_csv('donnees.csv')

# Afficher les 10 premières lignes
print(df.head(10))

# Afficher les 15 dernières lignes
print(df.tail(15))

# Regrouper par une colonne et trouver la moyenne
print(df.groupby('nom_colonne').mean())

# Filtrer les lignes où 'nom_colonne' est supérieur à une valeur
filtered_df = df[df['nom_colonne'] > valeur]

# Supprimer les lignes avec des valeurs NaN
df.dropna(inplace=True)

# Appliquer une fonction à 'nom_colonne'
df['nom_colonne'] = df['nom_colonne'].apply(lambda x: fonction(x))

# Obtenir un résumé statistique
print(df.describe())

# Fusionner deux DataFrames sur une colonne commune
merged_df = pd.merge(df1, df2, on='colonne_commune')

# Sauvegarder le DataFrame dans un fichier CSV
df.to_csv('nouvelles_donnees.csv', index=False)
```

Visualisation

- Interprétation Rapide
- Gestion des Risques
- Analyse des Tendances
- Support à la Décision
- https://datavizcatalogue.com





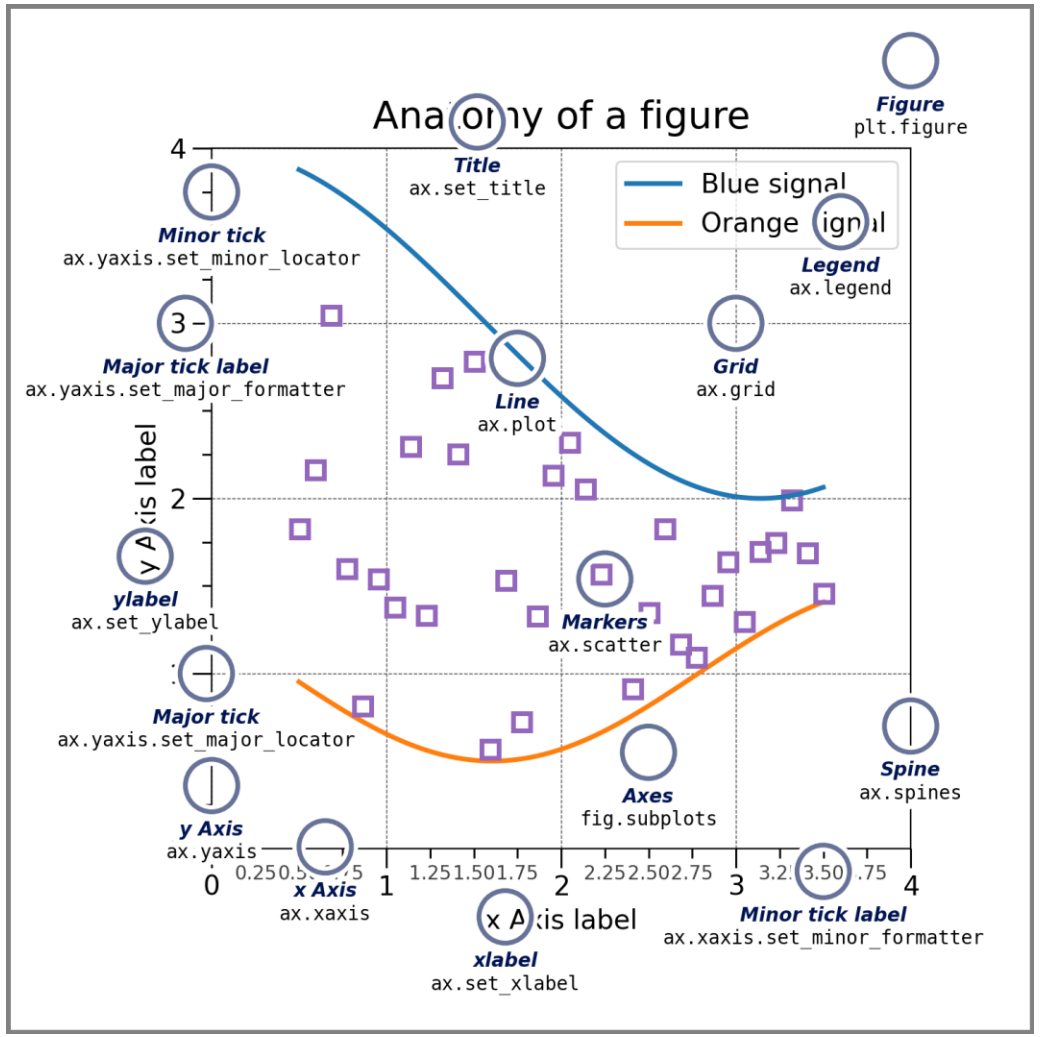


matplotlib

- Interactive
- Popular
- Custom

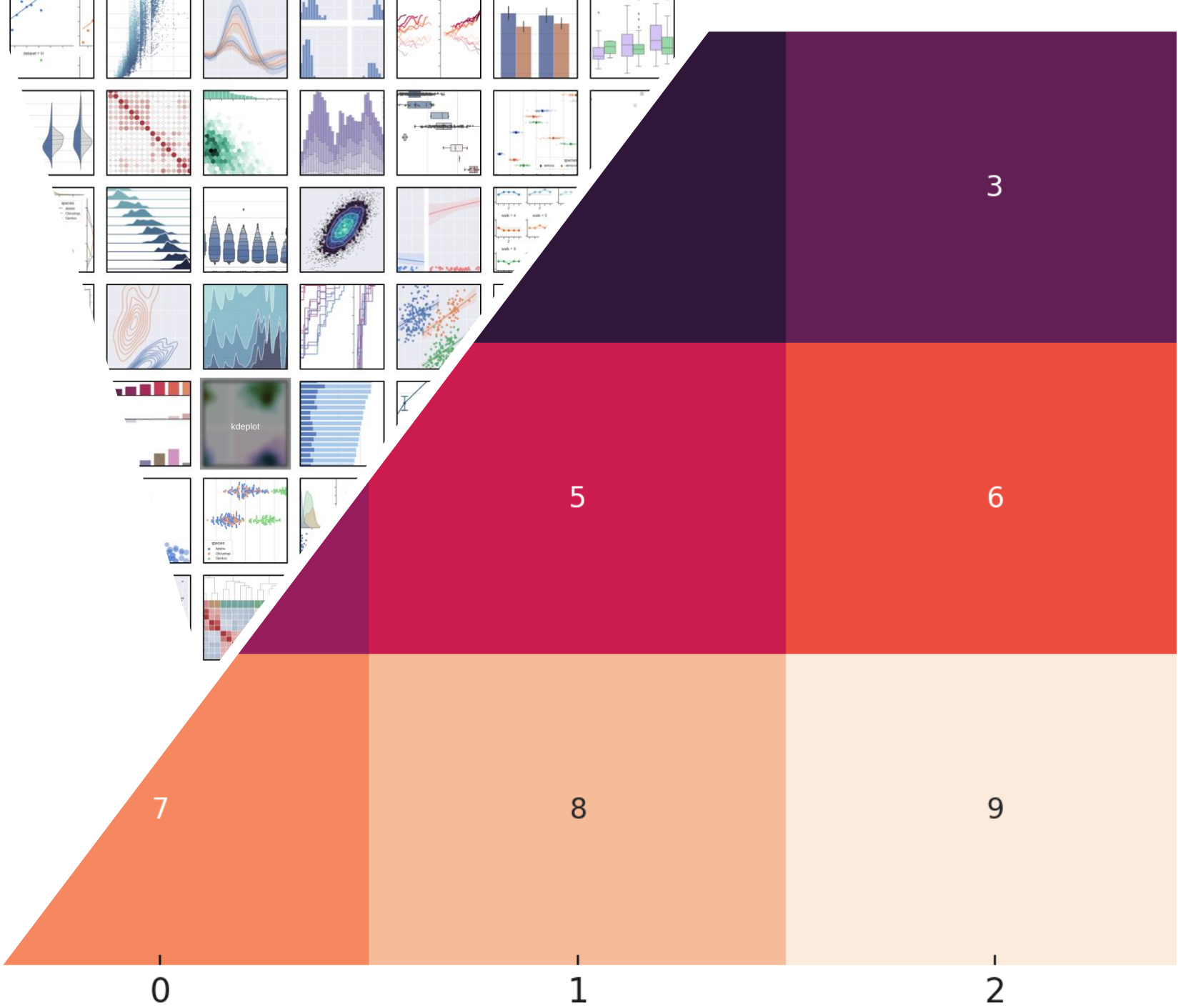
Plot

- Figure
- Axes
- Legends
- Line
- Grid
- Label
- Interval



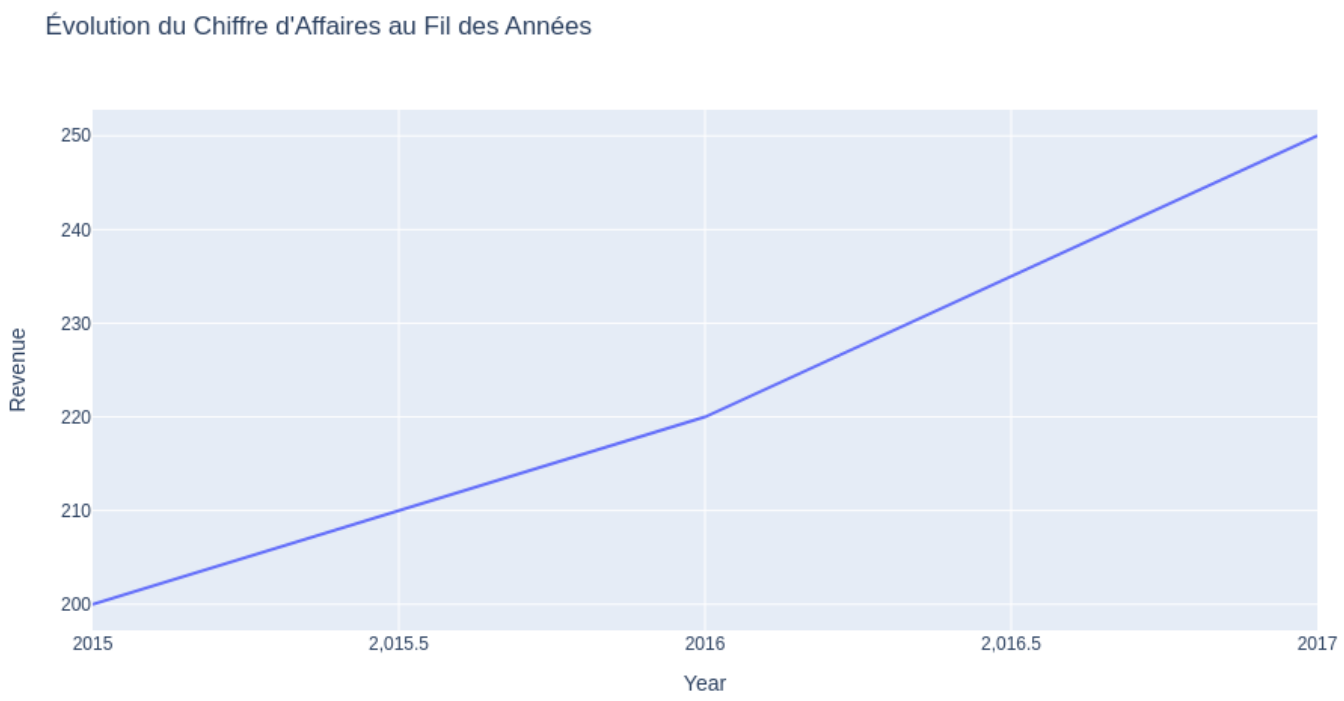
Seaborn

- Plus complexe
- Alternative pour science de donnée



Plotly

- Simple et facile à apprendre
- Integration Web (React)
- Multi language



Calculer une Moyenne Mobile Exponentielle (EMA)

1. RÉCUPÉRATION DES DONNÉES FINANCIÈRES

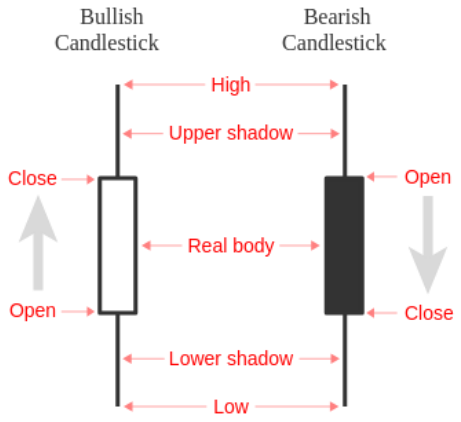
2. NETTOYAGE DES DONNÉES AVEC PANDAS

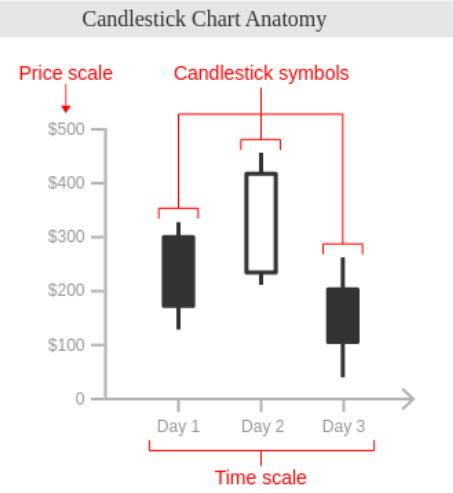
3. CALCUL DE L'EMA

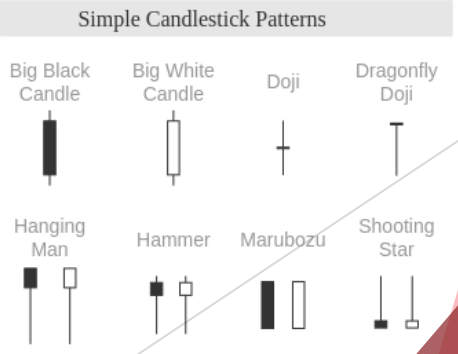
4. VISUALISATION AVEC MATPLOTLIB

Candle

$90
$80
$70
$60
$50
$40
$30
$20
$10
$0
March
April


Bullish Candlestick
Bearish Candlestick
High
Upper shadow
Close
Open
Real body
Open
Close
Lower shadow
Low


Candlestick Chart Anatomy
Price scale
Candlestick symbols
$500
$400
$300
$200
$100
0
Day 1
Day 2
Day 3
Time scale


Simple Candlestick Patterns
Big Black Candle
Big White Candle
Doji
Dragonfly Doji
Hanging Man
Hammer
Marubozu
Shooting Star

Étape 1: Récupération des Données Financières

```python
import requests
import json

# Replace 'YOUR_API_KEY' with your Financial Modeling Prep API key
url = "<https://financialmodelingprep.com/api/v3/historical-price-full/AAPL?apikey=YOUR_API_KEY>"

response = requests.get(url)
data = json.loads(response.text)

# Extract historical data
historical_data = data['historical']
```

Étape 2: Nettoyage des Données avec Pandas

```python
import pandas as pd

# Create DataFrame
df = pd.DataFrame(historical_data)

# Convert 'date' to datetime type
df['date'] = pd.to_datetime(df['date'])

# Sort by date
df.sort_values('date', inplace=True)

# Reset index
df.reset_index(drop=True, inplace=True)
```

Représentation graphique et définition

- Lorsqu'une moyenne mobile rapide est plus élevée qu'une moyenne mobile lente (ex. $MME_{20} > MME_{80}$) c'est un indicateur haussier (bullish).
- Lorsqu'une moyenne mobile rapide est plus basse qu'une moyenne mobile lente (ex. $MME_{80} > MME_{20}$) c'est un indicateur baissier (bearish).

Méthode de calcul des MME

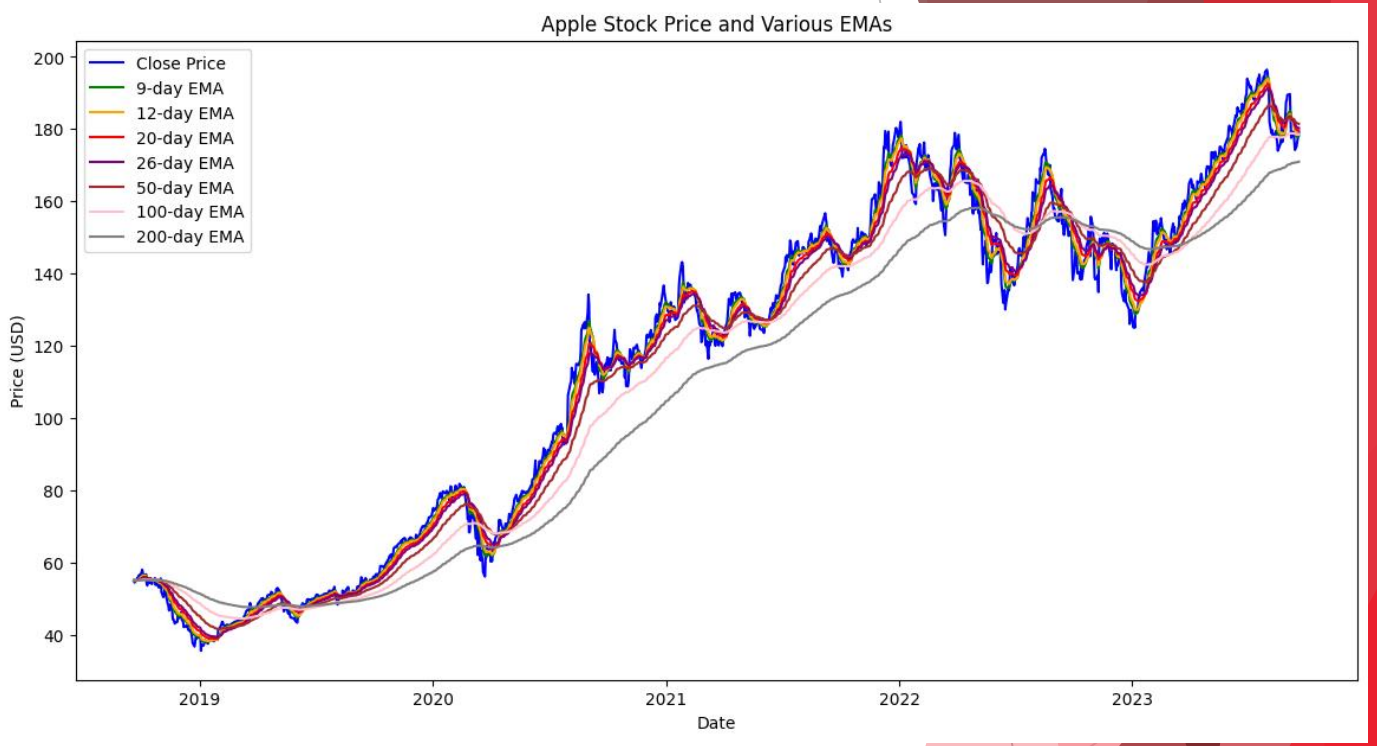
- Configurer le nombre de jours (k) de décalage pour les calculs.
  - Nous utiliserons 20 jours, 80 jours et 150 jours
- Calculer le α
  - $\alpha = \frac{2}{k+1}$
- La première moyenne, à n = k+1, est une moyenne régulière des k derniers jours
- Les suivantes s'obtiennent par l'équation suivante
  - $MME_n = \alpha\,(close_n - MME_{n-1}) + MME_{n-1}$

Étape 3: Calcul de l'EMA

```python
# Calculate 9-day EMA
df['EMA_9'] = df['close'].ewm(span=9, adjust=False).mean()

# Calculate 12-day EMA
df['EMA_12'] = df['close'].ewm(span=12, adjust=False).mean()

# Calculate 20-day EMA (Already provided)
df['EMA_20'] = df['close'].ewm(span=20, adjust=False).mean()

# Calculate 26-day EMA
df['EMA_26'] = df['close'].ewm(span=26, adjust=False).mean()

# Calculate 50-day EMA
df['EMA_50'] = df['close'].ewm(span=50, adjust=False).mean()

# Calculate 100-day EMA
df['EMA_100'] = df['close'].ewm(span=100, adjust=False).mean()

# Calculate 200-day EMA
df['EMA_200'] = df['close'].ewm(span=200, adjust=False).mean()
```

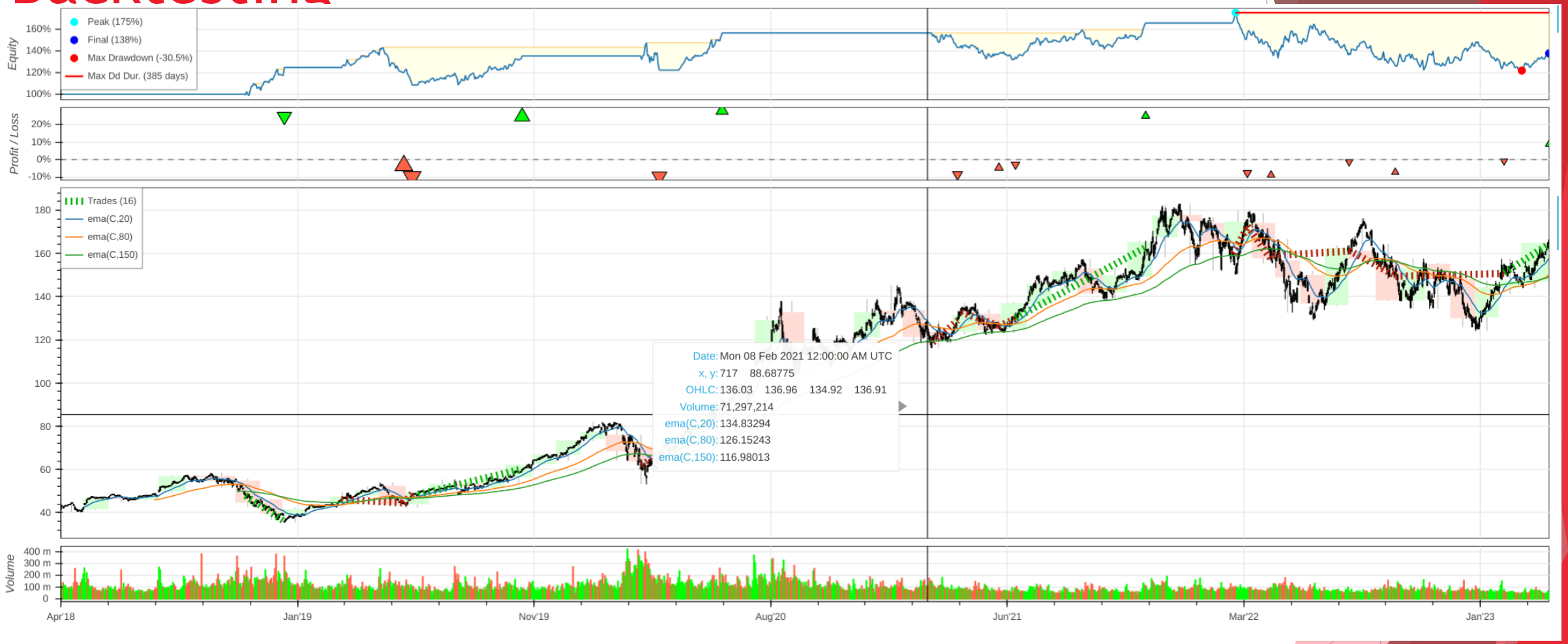
Étape 4: Visualisation avec Matplotlib

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(14, 7))

# Plot close price
plt.plot(df['date'], df['close'], label='Close Price', color='blue')

# Plot 9-day EMA
plt.plot(df['date'], df['EMA_9'], label='9-day EMA', color='green')

# Plot 12-day EMA
plt.plot(df['date'], df['EMA_12'], label='12-day EMA',
color='orange')
# Plot 20-day EMA
plt.plot(df['date'], df['EMA_20'], label='20-day EMA', color='red')

# Plot 26-day EMA
plt.plot(df['date'], df['EMA_26'], label='26-day EMA',
color='purple')
# Plot 50-day EMA
plt.plot(df['date'], df['EMA_50'], label='50-day EMA', color='brown')

# Plot 100-day EMA
plt.plot(df['date'], df['EMA_100'], label='100-day EMA', color='pink')

# Plot 200-day EMA
plt.plot(df['date'], df['EMA_200'], label='200-day EMA', color='gray')

plt.title('Apple Stock Price and Various EMAs')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()

plt.show()
```



TradingView

Backtesting



- https://github.com/AlgoETS/CookiecutterStrategie
- https://github.com/AlgoETS/Senators
- https://github.com/AlgoETS/BatchBacktesting
- https://github.com/AlgoETS/MovingAverage