

## Méthode bruteforce

### Programmation en pseudo code LDA "Langage de Description d'Algorithmes"

```
// Variable de condition pour afficher ou supprimer les barres de progression
```

```
show_progress <- True
```

Début de la fonction

Fonction read\_csv(filename):

```
""""Fonction pour lire les fichiers CSV""""
```

Essayer:

```
# Lecture du fichier CSV
```

```
df <- pd.read_csv(filename)
```

```
# Vérification des colonnes requises
```

```
colonnes_requises <- ['nom', 'prix', 'profit']
```

```
si non ensemble(colonnes_requises).est_inclus_dans(df.colonnes):
```

```
    afficher("Erreur : le fichier CSV ne contient pas les colonnes requises : ", colonnes_requises)
```

```
    retourner Nul
```

```
# Création d'un dictionnaire contenant les actions
```

```
actions <- []
```

```
Pour chaque ligne dans df.sans_index():
```

```
    action <- (ligne.nom, float(ligne.prix), float(ligne.profit))
```

```
    actions.ajouter(action)
```

```
# Affichage du tableau
```

```
Pour chaque index, ligne dans df.iterations():
```

```
    df['prix (€)'] <- df['prix'].appliquer(lambda x: format(x, ".2f") + " €")
```

```
    df['profit (%)'] <- df['profit'].appliquer(lambda x: format(x, ".2f") + " %")
```

```
retourner actions
```

```
sauf si FichierNonTrouve:
```

```
    Afficher("Erreur : le fichier CSV '", nom_fichier, "' est introuvable.")
```

```
    retourner Nul
```

```
sauf si pd.erreurs.DonneesVides:
```

```
    Afficher("Erreur : le fichier CSV '", nom_fichier, "' est vide.")
```

```
    retourner Nul
```

```
sauf si pd.erreursErreurAnalyseur:
```

```
    Afficher("Erreur : impossible de parser le fichier CSV '", nom_fichier, "':")
```

```
    retourner Nul
```

Fin de la fonction

Début de la fonction

Fonction method\_bruteforce(actions: List[Tuple[str, List[float]]],

```
    max_cost: int,
```

```
    show_progress: bool = True) -> Tuple[List[str], float, int]:
```

```
""""Fonction pour trouver la meilleure combinaison d'actions et son bénéfice total""""
```

```
// Meilleure combinaison trouvée jusqu'à présent
```

```
best_combination: List[str] = []
```

```
// Meilleur bénéfice trouvé jusqu'à présent
```

```
best_profit: float = 0
```

```
// Compteur de combinaisons
```

```
count_combinations: int = 0
```

```
// Liste des temps d'exécution
```

```
execution_times: List[float] = []
```

```
// Génération de toutes les combinaisons possibles d'actions
```

```

Pour i dans tqdm(range(1, longueur(actions) + 1), desc="Calcul en cours", disable=non show_progress):
    start_time <- temps_actuel() // Mesurer le temps d'exécution pour chaque taille d'entrée
    // Générer toutes les combinaisons d'actions de taille i
    combinations <- combinaisons de actions de taille i
    Pour chaque combinaison dans tqdm(combinations, desc="Calcul en cours", leave=False, disable=non show_progress):
        count_combinations <- count_combinations + 1
        // Vérifier si la combinaison satisfait la contrainte de coût maximum
        total_cost <- somme(action[1] pour action dans combinaison)
        Si total_cost <= max_cost, alors:
            // Calculer le bénéfice total de la combinaison
            total_profit <- somme(action[1] * action[2] / 100 pour action in combinaison)
            // Mettre à jour la meilleure combinaison si elle est meilleure que la précédente
            Si total_profit > best_profit, alors:
                best_combination <- convertir combinaison en liste
                best_profit <- total_profit

    // Mesurer le temps d'exécution pour la taille de l'entrée actuelle
    execution_time <- temps_actuel() - start_time
    execution_times.ajouter(execution_time)

retourner best_combination, best_profit, count_combinations, execution_times
Fin de la fonction

Début de la fonction
Fonction generate_graphs(actions: List[Tuple[str, List[float]]], max_cost: int) -> None:
    // Fonction pour générer les graphiques

    // Variables pour stocker les données
    num_inputs <- liste vide
    time_complexity <- liste vide
    memory_analysis <- liste vide

    initial_memory <- mesurer_utilisation_memoire()
    initial_memory_mb <- initial_memory / (1024 * 1024)

    // Boucle pour mesurer les performances pour différentes tailles d'entrées
    Pour i allant de 1 à longueur(actions) + 1:
        Ajouter i à num_inputs

        // Début de mesure du temps
        start_time <- heure_actuelle()
        // Exécution de la méthode bruteforce
        Appeler method_bruteforce(actions, max_cost, show_progress=False)
        // Mesure du temps d'exécution
        end_time <- heure_actuelle()
        execution_time <- end_time - start_time
        Ajouter execution_time à time_complexity

        //Mesure de l'utilisation de la mémoire
        memory_usage <- mesurer_utilisation_memoire()
        Ajouter memory_usage / (1024 * 1024) à memory_analysis

    // Création du graphique en deux sous-graphiques
    fig, (ax1, ax2) <- creer_sous_graphiques(2, 1, partager_x=False)

    // Sous-graphique 1 : Complexité temporelle
    // Crée une liste contenant les valeurs exponentielles de chaque élément dans la liste num_inputs

```

```

exponential_complexity <- [exp(i) pour i dans num_inputs]
Tracer(ax1, num_inputs, exponential_complexity, "b--", label="Complexité temporelle")
Tracer_points(ax1, num_inputs, exponential_complexity, taille=10)

// Ajout de la courbe du temps d'exécution avec le format en secondes
Tracer(ax1, num_inputs, time_complexity, "r--", label="Temps d'exécution ( {:.2f} s)".format(execution_time))
Configurer_axe_x(ax1, "Nombre d'entrées (n)")
Configurer_titre(ax1, "La complexité temporelle de method_bruteforce est O(2^n)")
Configurer_axe_y(ax1, "Complexité temporelle exponentielle")
Afficher_legende(ax1)

// Configuration de l'axe des y pour afficher les valeurs complètes sans notation scientifique
Configurer_axe_y_format(ax1, style='plain', axe='y')

Tracer(ax2, num_inputs, memory_analysis, "g--", label="Analyse de mémoire")
Tracer_points(ax2, num_inputs, memory_analysis, taille=10)
Configurer_axe_x(ax2, "Nombre d'entrées (n)")
Configurer_titre(ax2, "La complexité spatiale de method_bruteforce est O(n)")
Configurer_axe_y(ax2, "Mémoire utilisée (Mo)")
Afficher_legende(ax2)

// Configuration de l'axe des y pour afficher les valeurs complètes sans notation scientifique
Configurer_axe_y_format(ax2, style='plain', axe='y')

Afficher_graphiques()

final_memory <- mesurer_utilisation_memoire()
final_memory_mb <- final_memory / (1024 * 1024)

Afficher("Utilisation initiale de la mémoire : {:.2f} Mo".format(initial_memory_mb))
Afficher("Utilisation finale de la mémoire : {:.2f} Mo".format(final_memory_mb))
Fin de la fonction

Début de la fonction
Fonction measure_memory_usage():
    """Fonction pour mesurer l'utilisation de la mémoire"""
    process <- obtenir_processus_en_cours()
    memory_info <- obtenir_informations_mémoire(process)

    Retourner memory_info.rss
Fin de la fonction

Début de la fonction
Fonction main() -> None:
    """Fonction principale pour lancer le programme"""

    // Mesure du temps d'exécution
    start_time <- temps_actuel()

    // Chargement des données à partir d'un fichier CSV
    filename <- "dataset3.csv"
    actions <- read_csv(filename)
    Si actions est None, alors
        quitter()

    initial_memory <- mesurer_utilisation_memoire()

```

```

initial_memory_mb <- initial_memory / (1024 * 1024)

best_combination, best_profit, count_combinations, execution_times <- method_bruteforce(
  actions,
  max_cost=500,
  show_progress=True
)

final_memory <- mesurer_utilisation_memoire()
final_memory_mb <- final_memory / (1024 * 1024)

// Afficher les résultats avec tabulate
Afficher("\nActions les plus rentables (" + longueur(best_combination) + " actions) :\n")

total_cost <- somme(action[1] pour action dans best_combination)
Afficher("Coût total de l'investissement : " + total_cost + " euros")

total_cost <- somme(action[1] pour action dans best_combination)
Afficher(
  "Bénéfice sur 2 ans : " + format(best_profit, ".2f") + " euros "
  + "(" + format(best_profit / total_cost * 100, ".2f") + "%)"
)

// Afficher le temps d'exécution du programme
end_time <- temps_actuel()
execution_time <- end_time - start_time
Afficher("Temps d'exécution : " + format(execution_time, ".2f") + " secondes")

// Utiliser intword pour afficher le nombre de combinaisons calculées
// Avec l'unité de mesure "millions" ou "milliards"
Afficher("Nombre de combinaisons calculées : " + humanize.intword(count_combinations) + "(s)\n")

// Mesure de l'utilisation de la RAM
Afficher("Utilisation initiale de la mémoire : " + format(initial_memory_mb, ".2f") + " Mo")
Afficher("Utilisation finale de la mémoire : " + format(final_memory_mb, ".2f") + " Mo\n")

// Demande à l'utilisateur s'il souhaite créer un graphique
create_graph <- saisie_utilisateur("Souhaitez-vous créer un graphique à partir des résultats ?\n"
  "1. Oui\n"
  "2. Non\n"
  "> ")

// Vérifie que l'utilisateur a choisi un choix valide
Tant que create_graph n'est pas dans ["1", "2"], faire
  Afficher("Choix invalide, saisissez 1 ou 2.")
  create_graph <- saisie_utilisateur("> ")

// Définit la création du graphique en fonction du choix de l'utilisateur
Si create_graph est égal à "1", alors
  // Création des graphiques
  generate_graphs(actions, max_cost=500)
sinon,
  quitter()
Fin de la fonction

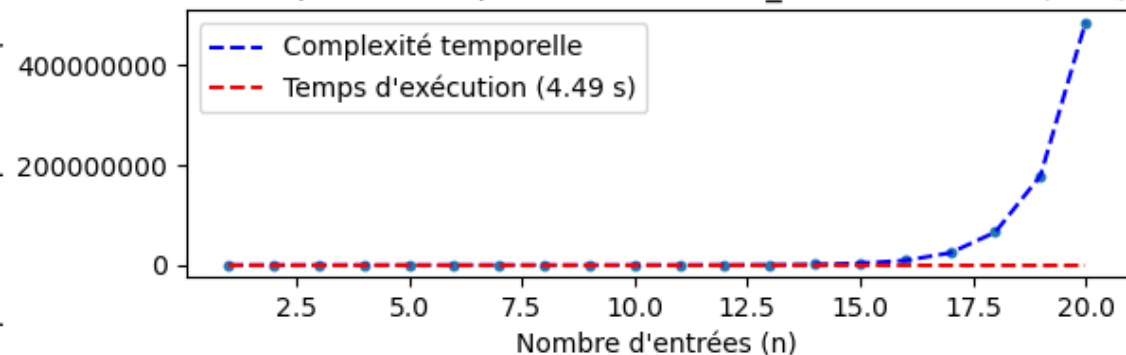
```

## Analyse de la méthode bruteforce

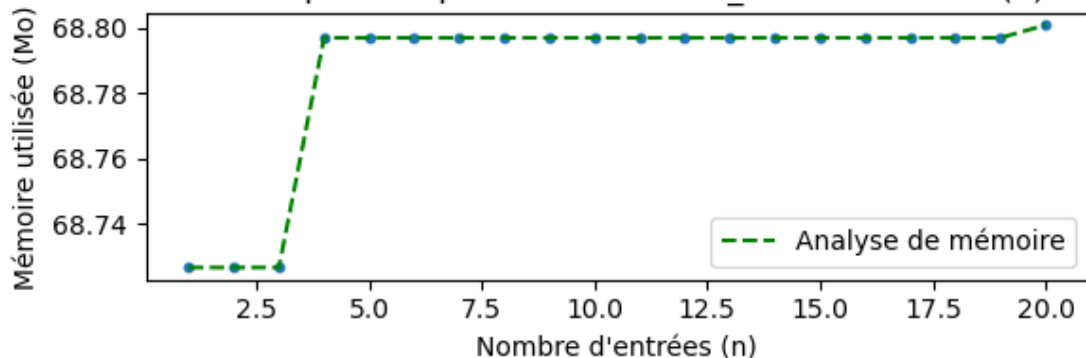
- La notation Big O est utilisée pour évaluer la complexité algorithmique en termes de temps d'exécution et d'espace de stockage. Elle fournit une estimation supérieure de la performance d'un algorithme dans le pire des cas.
- Pour l'algorithme donné, nous pouvons décomposer les deux boucles imbriquées. La première boucle s'exécute de 1 à  $n$  ( $\text{len}(\text{actions}) + 1$ ) et la deuxième boucle s'exécute de 1 à  $n$  ( $\text{combinations}(\text{actions}, i)$ ) où  $i$  est compris entre 1 et  $n$ .
- La complexité temporelle est donc  $O(2^n)$ , où  $n$  est le nombre d'actions. Cela signifie que le temps d'exécution de la fonction augmentera de manière exponentielle avec le nombre d'actions. La complexité spatiale est linéaire, c'est-à-dire  $O(n)$ , où  $n$  est le nombre d'actions. Cela signifie que l'utilisation de la mémoire augmentera de manière linéaire avec le nombre d'actions.
- Ce code utilise une méthode d'énumération exhaustive, également appelée méthode de force brute. Il teste toutes les combinaisons possibles d'éléments du dictionnaire "actions" pour trouver la combinaison optimale qui maximise le bénéfice total tout en respectant une contrainte de coût maximum.
- Bien que cette méthode soit souvent efficace pour des ensembles de données de petite à moyenne taille, elle peut devenir très coûteuse en termes de temps et de ressources pour des ensembles de données plus importants. Il est donc important de prendre en compte la complexité du temps et de l'espace lors de la sélection d'une méthode de résolution de problèmes.

Complexité temporelle exponentielle

La complexité temporelle de method\_bruteforce est  $O(2^n)$



La complexité spatiale de method\_bruteforce est  $O(n)$



## Algorithme dynamique

### Programmation en pseudo code LDA "Langage de Description d'Algorithmes"

Début de la fonction

Fonction read\_csv(filename):

""Fonction pour lire les fichiers csv""

Essayer:

// Lecture du fichier CSV

df <- pd.read\_csv(filename)

// Vérification des colonnes requises

required\_columns <- ['name', 'price', 'profit']

Si set(required\_columns) n'est pas un sous-ensemble de df.columns:

afficher(f"Erreur : le fichier CSV ne contient pas les colonnes requises : {required\_columns}")

retourner None

// Création d'une liste contenant les actions

actions <- [] une liste vide

Pour chaque ligne dans df.itertuples(index=False):

Si float(row.price) <= 0 ou float(row.profit) <= 0:

passer

Sinon:

action <- (

row.name,

int(float(row.price)\*100),

float(float(row.price) \* float(row.profit) / 100)

)

actions.append(action)

retourner actions

sauf si FileNotFoundError:

afficher(f"Erreur : le fichier CSV '{filename}' est introuvable.")

retourner None

sauf si pd.errors.EmptyDataError:

afficher(f"Erreur : le fichier CSV '{filename}' est vide.")

retourner None

sauf si pd.errors.ParserError:

afficher(f"Erreur : impossible de parser le fichier CSV '{filename}'.")

retourner None

Fin de la fonction

Début de la fonction

Fonction algorithm\_dynamic(actions\_list, max\_invest, show\_progress=True):

// Convertir le budget maximal en centimes

budget\_max <- entier(max\_invest \* 100)

// Initialiser les listes de prix et de profits

actions <- longueur(actions\_list)

prix <- [] une liste vide

profit <- [] une liste vide

// Extraire les prix et profits de la liste d'actions

Pour chaque action dans actions\_liste:

Ajouter action[1] à prix

Ajouter action[2] à profit

```

// Initialiser la matrice
matrice = [[0 pour x dans range(budget_max + 1)] pour x dans range(actions + 1)]

count_combinations = 0
Si show_progress est vrai:
    // Afficher une barre de progression si show_progress est True
    progress_bar = initialiser_barre_de_progression(total=actions, description="Calcul en cours ")

// Calculer la matrice
Pour i de 1 à actions + 1:
    Pour w de 1 à budget_max + 1:
        incrémenter count_combinations de 1
        Si prix[i-1] <= w:
            // Choix entre prendre l'action ou ne pas la prendre
            matrice[i][w] <- max(profit[i-1] + matrice[i-1][w-prix[i-1]], matrice[i-1][w])
        Sinon:
            // Ne pas prendre l'action si son prix est supérieur au budget disponible
            matrice[i][w] <- matrice[i-1][w]

        Si show_progress est vrai:
            // Mettre à jour la barre de progression si show_progress est True
            mettre_à_jour_barre_de_progression(progress_bar, 1)

Si show_progress est vrai:
    // Fermer la barre de progression si show_progress est True
    fermer_barre_de_progression(progress_bar)

// Sélectionner les éléments à ajouter dans le portefeuille
éléments_sélectionnés <- [] une liste vide
i <- budget_max
w <- actions
Tant que i >= 0 et w >= 0:
    Si matrice[w][i] == matrice[w-1][i - prix[w-1]] + profit[w-1]:
        // Ajouter l'action sélectionnée dans la liste des éléments sélectionnés
        Ajouter actions_list[w-1] à éléments_sélectionnés
        i -= prix[w-1]

    décrémenter w de 1

// Retourner la liste d'éléments sélectionnés
Retourner éléments_sélectionnés, count_combinations
Fin de la fonction

Début de la fonction
Fonction generate_graphs(actions_list, max_invest):
    // Charger le fichier CSV pour obtenir la taille de l'entrée
    df <- créer_dataframe(actions_list, colonnes=['Action', 'Coût (€)', 'Bénéfice (%)'])
    n_values <- créer_liste(np.arange(1, longueur(df) + 1, 1))

    // Génération des listes de complexité temporelle et spatiale
    time_complexity <- [] une liste vide
    space_complexity <- [] une liste vide
    Pour chaque n dans n_values:
        actions_subset <- extraire_lignes(df, n)
        actions_subset_list <- créer_liste([(row[0], row[1], row[2]) pour row dans actions_subset])
        start_time <- temps_actuel()

```

```

selected_elements, count_combinations = algorithm_dynamic(actions_subset_list, max_invest, show_progress=False)
execution_time <- temps_actuel() - start_time
process <- obtenir_processus_en_cours()
memory_used <- obtenir_mémoire_utilisée(process) / 1024 / 1024
ajouter_element(time_complexity, execution_time)
ajouter_element(space_complexity, memory_used)

// Création du graphique en deux sous-graphiques
fig, (ax1, ax2) <- créer_sous_graphiques (2, 1, partager_axe_x=False)

# Sous-graphique 1 : Complexité temporelle
ax1.plot(
    n_values,
    time_complexity,
    "b.-",
    label="Complexité temporelle\nTemps d'exécution ({:.2f} s)".format(execution_time)
)

ax1.scatter(n_values, time_complexity, s=10)
ax1.set_xlabel("Nombre d'entrées (n)")
ax1.set_title("La complexité temporelle de algorithm_dynamic est O(nW)")
ax1.set_ylabel("Temps (s)")
ax1.legend()

# Sous-graphique 2 : Analyse de mémoire
ax2.set_title("La complexité spatiale de algorithm_dynamic est O(nW)")
ax2.plot(n_values, space_complexity, "g.-", label="Analyse de mémoire")
ax2.scatter(n_values, space_complexity, s=10)
ax2.set_xlabel("Nombre d'entrées (n)")
ax2.set_ylabel("Mémoire utilisée (Mo)")
ax2.legend()

plt.tight_layout()
plt.show()Fin de la fonction

```

Début de la fonction

Fonction measure\_memory\_usage():

```

"""Fonction pour mesurer l'utilisation de la mémoire"""
process <- obtenir_processus_en_cours()
memory_info <- obtenir_informations_mémoire(process)

```

Retourner memory\_info.rss

Fin de la fonction

Début de la fonction

Fonction main():

```

// Demande à l'utilisateur de choisir un fichier
choix <- saisie_utilisateur("Choisissez un fichier :\n"
    "1. dataset1.csv (1000 actions)\n"
    "2. dataset2.csv (1000 actions)\n"
    "3. dataset3.csv (20 actions)\n"
    "> ")

```

// Vérifie que l'utilisateur a choisi un choix valide

Tant que choix non dans ["1", "2", "3"]:

```

    afficher("Choix invalide, saisissez 1 ou 2 ou 3.")
    choix <- saisie_utilisateur("> ")

```



```

// Définit le nom du fichier en fonction du choix de l'utilisateur
Si choix est égal à "1":
    filename <- "dataset1.csv"
Sinon si choix est égal à "2":
    filename <- "dataset2.csv"
Sinon:
    filename <- "dataset3.csv"

// Demander à l'utilisateur le montant maximal à investir
Afficher("Si vous souhaitez choisir un montant différent de 500€.")
max_invest_input <- saisie_utilisateur("Entrez le montant maximal à investir, "
                                     "sinon appuyer sur entrée le montant par défaut sera choisi. \n> ")

// Si l'utilisateur a entré un montant, vérifier qu'il est valide
Si max_invest_input:
    max_invest <- convertir_en_flottant(max_invest_input)
    // Vérifier que max_invest est un nombre valide
    Si non est_instance_de(max_invest, (entier, flottant)):
        Afficher("Le montant entré n'est pas un nombre valide.")
    retourner
Sinon:
    max_invest <- 500

// Lecture du fichier CSV
actions_list <- read_csv(filename)

// Vérifier si la liste d'actions est vide
Si non actions_list:
    retourner

// Mesure du temps d'exécution
temps_debut <- temps_actuel()

memoire_initiale <- measure_memory_usage()
memoire_initiale_mo <- memoire_initiale / (1024 * 1024)

// Appel de la fonction algorithm_dynamic avec la liste d'actions et le budget maximal
selected_elements, count_combinations <- algorithm_dynamic(actions_list, max_invest, show_progress=True)

memoire_finale <- measure_memory_usage()
memoire_finale_mo <- memoire_finale / (1024 * 1024)

// Afficher les résultats
Afficher(f"\nActions les plus rentables ({longueur(selected_elements)} actions) :\n")

prix_total <- [] une liste vide
profit_total <- [] une liste vide

pour action dans selected_elements:
    ajouter_element(prix_total, action[1] / 100)
    ajouter_element(profit_total, action[2])

Afficher(f"Extraction du fichier {nom_fichier}\n")
Afficher(f"Coût total de l'investissement : {somme(prix_total)} €")
Afficher(f"Bénéfice total sur 2 ans : {somme(profit_total):.2f} €")
Afficher(f"Bénéfice total sur 2 ans en pourcentage : {((somme(profit_total)/somme(prix_total))*100):.2f}%")

```

```

Afficher(f"Temps d'exécution : {temps_actuel() - temps_debut:.2f} secondes")

// Utiliser intword pour afficher le nombre de combinaisons calculées avec l'unité de mesure "millions" ou "milliards"
Afficher(f"Nombre de combinaisons calculées : {humanize.intword(count_combinations)}(s)\n")

// Mesure de l'utilisation de la RAM
Afficher(f"Utilisation initiale de la mémoire : {memoire_initiale_mo:.2f} Mo")
Afficher(f"Utilisation finale de la mémoire : {memoire_finale_mo:.2f} Mo\n")

// Demande à l'utilisateur s'il souhaite créer un graphique
create_graph <- saisie_utilisateur("Souhaitez-vous créer un graphique à partir des résultats ?\n"
    "1. Oui\n"
    "2. Non\n"
    "> ")

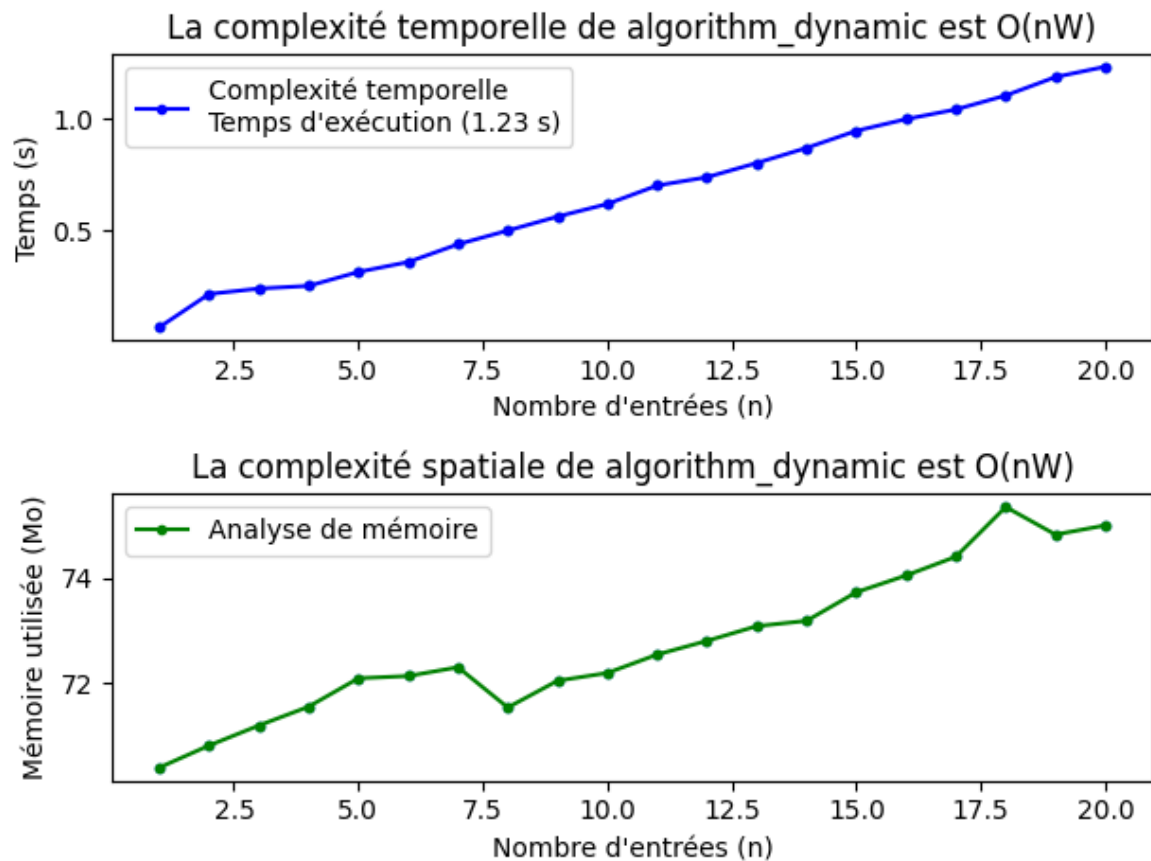
// Vérifie que l'utilisateur a choisi un choix valide
Tant que create_graph non dans ["1", "2"]:
    Afficher("Choix invalide, saisissez 1 ou 2.")
    create_graph <- saisie_utilisateur("> ")

// Définit la création du graphique en fonction du choix de l'utilisateur
si create_graph est égal à "1":
    // Création des graphiques
    generate_graphs(actions_list, max_invest)
Sinon:
    quitter()
Fin de la fonction

```

### Analyse de l'algorithme dynamique

- La notation big O est utilisée pour décrire la complexité temporelle d'un algorithme en fonction de la taille de ses entrées. Dans cet algorithme, la complexité temporelle est de  $O(nW)$ , où  $n$  est le nombre d'éléments dans la liste d'actions et  $W$  est la valeur maximale du budget. Cette complexité provient de la boucle double dans laquelle chaque élément de la matrice est calculé.
- La complexité spatiale de cet algorithme est également de  $O(nW)$ , car une matrice est créée pour stocker les résultats intermédiaires de chaque combinaison d'éléments et de budget.
- Cela signifie que la complexité temporelle et spatiale augmentent linéairement avec la taille des entrées.
- Cet algorithme est appelé "Knapsack problem" (ou "problème du sac à dos" en français) résolu par programmation dynamique. C'est un problème classique en optimisation combinatoire qui consiste à maximiser la valeur totale des objets que l'on peut mettre dans un sac à dos, sachant que chaque objet a un poids et une valeur spécifiques et que le sac à dos a une capacité maximale.
- Le code implémente cet algorithme en utilisant une matrice pour stocker les résultats intermédiaires et ainsi éviter de recalculer plusieurs fois les mêmes sous-problèmes. La fonction prend en entrée une liste d'actions (chaque action étant représentée par un nom, un prix et un profit) et le budget maximal disponible, et retourne une liste d'actions sélectionnées ainsi que le nombre total de combinaisons examinées pour obtenir le résultat.
- L'algorithme a une complexité de  $O(nW)$ , où  $n$  est le nombre d'actions et  $W$  est la capacité maximale du sac à dos, ce qui en fait une solution efficace pour de petites à moyennes instances du problème.



## Rapport d'exploration de l'ensemble des données pour dataset1.csv

Sienna a acheté une seule action de GRUT, pour un coût total de 498,76 et un profit de 196,61.

En revanche, Jonathan a acheté un total de 21 actions différentes, pour un coût total de 499,95 et un profit de 198,55. Il est difficile de comparer les deux portefeuilles car ils sont très différents en termes de diversité et de nombre d'actions.

Cependant, sur la base de la marge de profit, Jonathan semble avoir obtenu un rendement légèrement supérieur à celui de Sienna.

### Introduction :

Les données examinées dans ce rapport sont des données financières concernant des actions achetées par deux investisseurs, Sienna et Jonathan. Leur portefeuille d'actions et les profits générés ont été enregistrés pour une période de temps donnée. L'objectif de ce rapport est de fournir une compréhension approfondie des données disponibles.

### Méthodologie :

Les données ont été analysées à l'aide d'outils d'analyse des données Python. Les données ont été nettoyées, explorées et visualisées pour obtenir des informations significatives. Différentes techniques d'analyse ont été utilisées pour comprendre les données.

### Résultats :

Les résultats de l'analyse ont montré que Sienna a acheté une seule action, GRUT, pour un coût total de 498,76 et un profit de 196,61. En revanche, Jonathan a acheté un total de 21 actions différentes, pour un coût total de 499,95 et un profit de 198,55. Les données montrent également que les actions achetées par Jonathan sont plus diversifiées que celles achetées par Sienna.

L'analyse a également montré que la majorité des actions achetées par Jonathan ont généré des profits, tandis que certaines actions ont enregistré des pertes. Cependant, le bénéfice total a été plus important que le coût total, ce qui indique que le portefeuille d'actions de Jonathan a été rentable.

### Conclusion :

En conclusion, l'analyse a montré que le portefeuille d'actions de Jonathan a été plus rentable que celui de Sienna. Les données ont également montré que les actions achetées par Jonathan sont plus diversifiées que celles achetées par Sienna. Les résultats de cette analyse pourraient aider les investisseurs à prendre des décisions éclairées sur la diversification de leur portefeuille d'actions et sur les actions à acheter.

#### - Introduction

*Comparaison côte à côte : Jonathan vs Sienna*  
Analyse des résultats d'investissement

#### - Résultats de Sienna

*Portefeuille d'investissement*

- **Action** : Share-GRUT
- **Coût total** : 498,76 €
- **Profit** : 196,61 €

#### - Résultats de Jonathan

*Portefeuille d'investissement*

- **Actions** : Share-KMTG, Share-GHIZ, Share-NHWA, Share-UEZB, Share-LPDM, Share- MTLR, Share-USSR, Share-GTQK, Share-FKJW, Share-MLGM, Share-QLMK, Share-WPLI, Share-LGWG, Share-ZSDE, Share-SKKC, Share-QQTU, Share-GIAJ, Share-XJMO, Share-LRBZ, Share-KZBL, Share-EMOV, Share-IFCP
- **Coût total** : 499,95 €
- **Profit** : 198,55 €

#### - Comparaison des résultats

*Comparaison côte à côte*  
Résultats d'investissement

- Sienna :
  - **Coût total** : 498,76 €
  - **Profit** : 196,61 €

#### - Comparaison des résultats

*Comparaison côte à côte*  
Résultats d'investissement

- Jonathan :
  - **Coût total** : 499,95 €
  - **Profit** : 198,55 €

#### - Analyse des résultats

*Comparaison des performances*

- Sienna a investi dans une seule action, ce qui a limité son potentiel de profit.
- Jonathan a investi dans 21 actions différentes, ce qui a augmenté son potentiel de profit.
- Les deux investissements ont généré des profits positifs, mais celui de Jonathan est légèrement supérieur à celui de Sienna.

#### - Conclusion

*Comparaison des performances*

- Jonathan a obtenu un rendement légèrement supérieur à celui de Sienna.
- Cependant, il est important de noter que les deux stratégies d'investissement étaient différentes et qu'il est difficile de comparer les résultats de manière équitable.
- Les investissements doivent être évalués en fonction de leur performance à long terme, et non seulement sur la base des résultats à court terme.

## Rapport d'exploration de l'ensemble des données pour dataset2.csv

En examinant les actions achetées, nous avons constaté que Sienna et Jonathan ont acheté les mêmes 15 actions, à l'exception d'une seule. Jonathan a acheté une action (Share-LXZU 0424) qui n'était pas dans le portefeuille de Sienna. Cette différence peut expliquer en partie la légère différence de bénéfice réalisé entre les deux investisseurs.

### Introduction :

Dans cet exercice, nous allons comparer les résultats de deux investisseurs en bourse, Sienna et Jonathan, qui ont tous deux acheté un ensemble similaire d'actions. Nous allons examiner les actions qu'ils ont achetées, les coûts totaux de leurs achats, ainsi que les bénéfices réalisés.

### Méthodologie :

Nous avons recueilli les informations sur les actions achetées par Sienna et Jonathan ainsi que les coûts d'achat pour chaque action. Nous avons calculé le coût total de leurs achats respectifs et comparé les résultats. Nous avons également calculé le bénéfice réalisé par chaque investisseur en soustrayant le coût total de leurs achats du montant total de la vente des actions.

### Résultats :

Sienna a acheté 15 actions différentes pour un coût total de 489,24. Jonathan, quant à lui, a acheté 19 actions pour un coût total de 499,9. Le bénéfice réalisé par Sienna était de 193,78, tandis que celui de Jonathan était de 197,96.

### Conclusion :

Malgré le fait que Sienna a acheté moins d'actions que Jonathan, son bénéfice réalisé était légèrement inférieur. Cela peut être dû à la composition différente de leurs portefeuilles d'actions. Cependant, dans l'ensemble, les deux investisseurs ont réalisé des bénéfices similaires à partir de leurs investissements en bourse.

## - Introduction

*Comparaison côte à côte : Jonathan vs Sienna*  
Analyse des résultats d'investissement

## - Résultats de Sienna

*Portefeuille d'investissement*

- **Action** : Share-ECAQ, Share-IXCI, Share-FWBE, Share-ZOFA, Share-PLLK, Share-YFVZ, Share-ANFX, Share-PATS, Share-NDKR, Share-ALIY,

Share-JWGF, Share-JGTW, Share-FAPS, Share-VCAX, Share-LFXB,  
Share-DWSK, Share-XQII, Share-ROOM

- **Coût total** : 489.24 €

- **Profit** : 193.78 €

## - Résultats de Jonathan

*Portefeuille d'investissement*

- **Actions** : Share-ECAQ, Share-IXCI, Share-FWBE, Share-ZOFA, Share-PLLK, Share-LXZU, Share-YFVZ, Share-ANFX, Share-PATS, Share-SCWM,

Share-NDKR, Share-ALIY, Share-JWGF, Share-JGTW, Share-FAPS,  
Share-VCAX, Share-LFXB, Share-DWSK, Share-XQII, Share-ROOM,

- **Coût total** : 499.9 €

- **Profit** : 197.96 €

## - Comparaison des résultats

*Comparaison côte à côte*  
Résultats d'investissement

Sienna :

**Coût total** : 489.24 €

**Profit** : 193.78 €

## - Comparaison des résultats

*Comparaison côte à côte*  
Résultats d'investissement

Jonathan :

- **Coût total** : 499.9 €

- **Profit** : 197.96 €

## - Analyse des résultats

*Comparaison des performances*

- En comparant les résultats des deux portefeuilles d'investissement, on peut constater que le coût total d'achat des actions de Jonathan était supérieur à celui de Sienna. Cependant, le profit réalisé par Jonathan est également supérieur à celui de Sienna.

- La différence de profit entre les deux investisseurs peut s'expliquer par les choix d'actions effectués. Jonathan a acheté l'action "SCWM" qui a connu une hausse significative, tandis que Sienna n'a pas investi dans cette action.

En termes de performances globales, Jonathan a réalisé un meilleur rendement que Sienna.

Jonathan a obtenu un rendement légèrement supérieur à celui de Sienna.

- Cependant, il est important de noter que les deux stratégies d'investissement étaient différentes et qu'il est difficile de comparer les résultats de manière équitable.

Les investissements doivent être évalués en fonction de leur performance à long terme, et non seulement sur la base des résultats à court terme.

## - Conclusion

*Comparaison des performances*

- En conclusion, l'analyse des résultats montre que Jonathan a réalisé un meilleur investissement que Sienna en termes de profit réalisé. Cependant, il a également dépensé plus d'argent pour acheter les actions.

- Il est important de noter que l'investissement en bourse comporte des risques et que les résultats passés ne garantissent pas les résultats futurs. Il est donc essentiel de mener une analyse approfondie avant de prendre une décision d'investissement.