Projet Final: Programmation Avancée en Python et Introduction à R

Introduction

Ce document décrit les spécifications du projet final pour le cours de Programmation Avancée en Python et Introduction à R. Le projet final est composé de trois tâches principales, qui seront réalisées en équipes de 7 à 10 étudiants. L'objectif de ce projet est de permettre aux étudiants d'appliquer les concepts appris en classe de manière pratique.

Tâche 1: Implémentation d'un sous-ensemble de NumPy en Python Pur

Dans cette tâche, vous devez implémenter une classe `**Array**` en Python pur qui réplique certaines des fonctionnalités de base de la bibliothèque NumPy. La classe doit permettre la création de tableaux (à une ou deux dimensions) et supporter les opérations suivantes :

- Création d'un array a partir d'une liste Python. (e.g. Array([1,2,3,4,5]) créé un array 1d de cinq éléments ; Array([[1,2],[3,4]]) créé un array 2d de shape(2,2)). Comme avec Numpy.
- Les opérations éléments par éléments sur les matrices: Addition (+), Soustraction (-), Multiplication (*), Division (/) de arrays (1d ou 2d).
- Le produit scalaire de deux arrays avec l'opérateur '@' (e.g. x@y, comme avec Numpy). On va supporter cette opération uniquement sur les array 1d et le résultat devrait être un scalaire.
- Recherche d'éléments (avec l'opérateur 'in')
- indexage et Slicing (e.g. indexage x[0, 1] pour un array 2d, slicing x[1:5]. Les deux x[0:2, 0] pour un array 2d) un peu comme avec Numpy.

Chaque fonction doit inclure des annotations de type pour pratiquer l'utilisation des type hints en Python. Vous devrez également implémenter des méthodes spéciales (__add__, __sub__, __mul__, etc.) pour rendre votre classe compatible avec les opérations précitées.

Détails supplémentaires :

- Support de la création de tableaux 1D et 2D comme mentionne plus haut.
- Les opérations éléments par éléments (+, -, *, /) fonctionnent seulement sur les tableaux de même dimension et forme (shape) sans diffusion (broadcasting). Sauf si

l'un des termes est un scalaire (e.g., x * 2. Si x est un Array ([1,2,3]) alors on aura [1*2, 2*2, 3*2], et donc un Array([2,4,6])).

- Le produit scalaire doit être possible sur les arrays 1D uniquement. Plusieurs autres combinaisons sont possibles, mais pour réduire la portée du travail, nous nous limiterons aux array 1d.
- Support de la fonction **len** pour montrer la longueur du tableau (nombre d'éléments en 1D, nombre de lignes en 2D) comme avec Numpy.
- Support de l'attribut **shape** qui renvoie un tuple comme en NumPy.
- Recherche avec l'opérateur 'in', comme en NumPy (par exemple, **5 in x** pour vérifier la présence d'un élément).
- Pour les opérations non supportées, lever une erreur avec un message explicite.
- Ajouter des annotations de type basiques à toutes les fonctions.
- Tout le code sera dans un seul fichier **numpy.py**

Tâche 2: Tracé de Graphiques et Analyse de Données

Dans cette tâche, vous allez réaliser une analyse de données basique en utilisant à la fois Python et R. Vous recevrez un jeu de données simple et vous devrez créer les graphiques suivants :

➔ Un histogramme
➔ Un graphique nuage de point (scatter plot)

Pour chaque graphique, vous devez fournir une courte analyse expliquant les tendances ou les corrélations possibles observées (1-3 phrases au plus, dans le rapport). Les graphiques doivent être créés dans les deux langages, Python et R, afin de comparer les outils de visualisation dans chaque environnement.

Vous devez utiliser matplotlib pour Python. Le jeu de données fourni sera un fichier CSV sur les prix de l'immobilier.

Pour l'histogramme, utilisez la colonne 'bedrooms'. Pour le graphique de dispersion, utilisez la colonne 'area' en abscisse et 'price' en ordonnées. Le dataset peut être téléchargé Housing.csv .

Tâche 3: Petit Programme GUI en Python avec Tkinter

Dans cette tâche, vous devez créer une petite application GUI en utilisant Tkinter en Python. L'application doit permettre à l'utilisateur de saisir une description textuelle et de générer une image correspondante à l'aide d'un modèle pré-entraîné de Hugging Face. Voici les spécifications détaillées :

- Une zone de texte pour saisir la description
- Un bouton pour générer l'image
- Une zone d'affichage pour montrer l'image générée avec la légende
- Un spinner (indicateur de chargement) qui apparaît pendant la génération de l'image

Vous devez utiliser ce modèle léger de Hugging Face pour la génération d'images : tiny-stable-diffusion-pipe. Notez que ce modèle génère principalement une image avec des pixels aléatoires parce que c'est un tout petit modèle de test. Il a et a été choisi car il est très léger et peut fonctionner sur un ordinateur personnel sans être extrêmement lent. Ceci vous permettra de coder et tester votre application plus facilement plutôt que d'utiliser un modele lourd qui prendra des dizaines de minutes pour générer une image sur un ordinateur personnel. Pour le fun (ceci est optionnel, vous serez noté sur le petit modèle de test, car plus rapide a exécuter), vous pouvez tester des modèles plus grands après avoir terminé, par exemple tiny-sd. Tout ce que vous aurez à faire c'est de changer le nom du modèle dans et ça devrait marcher normalement.

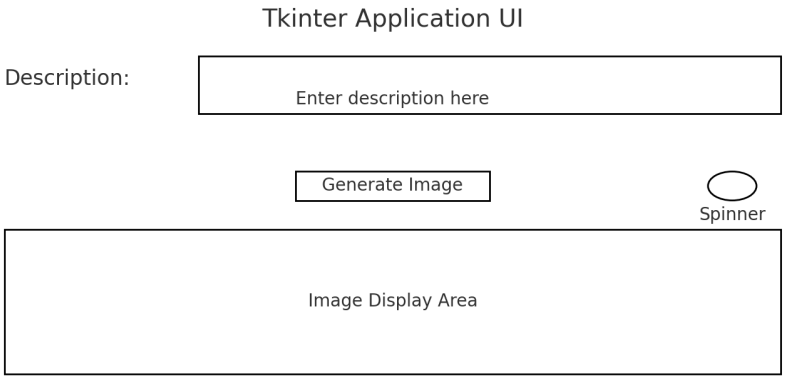
Les modèles peuvent également être testés sur Google Colab (c'est optionnel aussi, juste une recommandation pour tester le modele) avec un GPU (n'oubliez pas d'installer pip install 'transformers diffusers accelerate').

Voici un extrait de code pour utiliser le modèle :

```python
from diffusers import StableDiffusionPipeline
pipe = StableDiffusionPipeline.from_pretrained("hf-internal-testing/tiny-stable-diffusion-torch")
prompt = "a photo of an astronaut riding a horse on mars"
image = pipe(prompt).images[0]
image.save("astronaut_rides_horse.png")
```

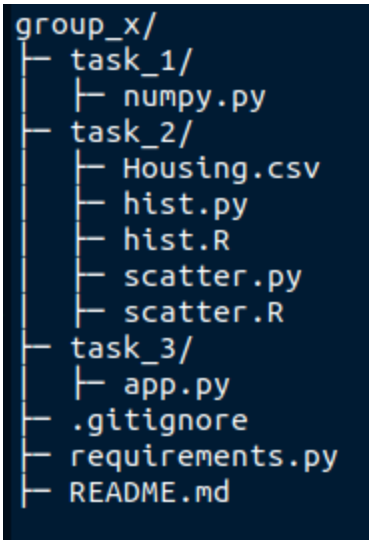
Esquisse de l'interface utilisateur

Ci-dessous une illustration de l'interface utilisateur souhaitée (le minimum à avoir dans le GUI de votre application. Vous pouvez réarranger les éléments comme bon vous semble et modifier les textes comme le nom de l'application, etc…):



Logistique et autres

- Le projet débute le **Lundi 17 Juin** et fini le **vendredi 28 Juin.**
- Vous devrez créer un repo GitHub avec pour nom le **'groupe_x'**. x représente le numéro du groupe.
- Le rapport doit être court et concis (1-3 pages A4 au plus).
- Le rapport doit inclure la contribution de chaque membre du groupe
- Chaque groupe présentera son projet le **Samedi 28 Juin**



Vous utiliserez cette structure pour le projet. le rapport sera dans le fichier **README.md** . Vous pourrez ajouter des fichiers supplémentaires si nécessaire.

Conclusion

Ce projet final a pour but de vous permettre de mettre en pratique les concepts avancés de Python et les bases de R que vous avez appris durant ce cours. Nous vous encourageons à travailler en équipe et à exploiter les compétences de chacun pour réaliser ces tâches. Une fois le projet terminé, chaque groupe doit soumettre un rapport minimaliste et concis, comprenant quatre sections : Tâche 1, Tâche 2, Tâche 3 et Contributions de chaque membre de l'équipe. Bonne chance !