Merci de m'avoir invité à participer à l'institut d'été en sciences sociales computationnelles de cette année. J'ai moi-même suivi cet institut organisé par Vissého il y a plusieurs années, et je ne peux pas assez insister sur l'amélioration considérable de mes compétences en programmation grâce à ce cours. C'est une expérience incroyable où l'on apprend beaucoup de choses nouvelles et où l'on se fait de nombreux nouveaux amis et collaborateurs.

Je suis actuellement Candidat au doctorat à l'Université de Toronto en criminologie et je prévois de défendre ma thèse cet automne. En ce moment, je travaille en tant que data scientist pour différentes agences gouvernementales, principalement des agences de police. Par le passé, j'ai également travaillé avec des commissions des droits de l'homme et des organisations internationales telles que l'OCDE. Ma recherche doctorale utilise diverses méthodes de traitement du langage naturel et autres méthodes computationnelles pour améliorer les enquêtes criminelles sur les crimes financiers complexes. Cependant, en tant que data scientist travaillant en dehors du milieu universitaire, je m'occupe de divers travaux, allant du web scraping à la modélisation statistique, en passant par la visualisation de données et la conception de tableaux de bord. Je travaille principalement avec R et Python.

Aujourd'hui, le sujet de notre session de formation, le web scraping, est l'un de mes préférés en tant que chercheur. Le web scraping, comme nous allons l'apprendre, est une technique de collecte de données extrêmement puissante à ajouter à votre kit méthodologique, et R propose de nombreux outils qui rendent le web scraping beaucoup plus facile que dans d'autres langages de programmation.

Le plan pour aujourd'hui est de faire une brève présentation d'environ 20 minutes pour vous donner un aperçu des concepts et des idées de base que nous devons comprendre pour faire du web scraping en R. Je vais faire cette présentation en français. Ensuite, nous ouvrirons le débat pour les questions, nous prendrons une courte pause de 5 minutes, puis nous passerons au tutoriel où nous ouvrirons RStudio et effectuerons du web scraping réel. Je suis anglophone, je viens de la Colombie-Britannique, mais je vis maintenant au Québec, au nord de Montréal, et j'apprends encore à parler français. Donc, je ferai la présentation en français, puis passerai à l'anglais pour le tutoriel. Veuillez poser vos questions en français ou en anglais, selon ce que vous préférez. Si j’ai besoin d'aide pour comprendre, je suis sûr qu'il y a quelqu'un ici qui peut m’aider.

Donc, qui a déjà fait du web scraping ? Vous pouvez écrire dans le chat si tu veux. **SLIDE**

Alors, qu'est-ce que c’est le web scraping ? En termes simples, le web scraping consiste à collecter automatiquement des informations sur Internet.

Il existe de nombreux outils disponibles dans R que nous pouvons utiliser pour faire du web scraping. Aujourd'hui, je vais vous parler d'un paquet appelé << rvest >>, mais je mentionnerai également plusieurs autres outils vers la fin de la session. Si vous n'avez jamais fait de web scraping auparavant, rvest est l'endroit où commencer, car c'est l'outil que vous utiliserez le plus souvent pour faire du web scraping avec R. D'autres paquets comme RSelenium sont importants, mais seulement nécessaires lorsque nous traitons des sites web plus complexes qui utilisent beaucoup de JavaScript et de contenu dynamique. Si cela vous semble confus, ne vous inquiétez pas, je vais expliquer brièvement ce que cela signifie.

Dans le contexte de la recherche en sciences sociales, le web scraping offre de nombreuses opportunités, mais présente également de nombreux défis. **SLIDE**

En termes d'opportunités, le web scraping permet d'accéder à de nouvelles données accessibles qui pourraient ne pas être disponibles par d'autres moyens. Il peut s'agir d'un ensemble de données textuelles telles que des documents PDF, un corpus de tweets ou de commentaires Reddit, une collection d'articles de presse, ou encore un tableau de données quantitatives collectées à partir de sources diverses. Le web scraping nous permet de collecter des informations en temps réel, ce qui est essentiel pour certains projets de recherche. Comme je le mentionnerai à la fin, il existe maintenant des moyens d'automatiser nos scripts R pour les exécuter toutes les 10 minutes, toutes les heures ou toutes les semaines, afin de collecter des informations sur Internet au fil du temps. Le web scraping nous permet de mener des recherches mixtes, en combinant des données provenant d'Internet avec d'autres sources de données plus traditionnelles telles que les données du recensement ou les données institutionnelles. Enfin, le web scraping ouvre des possibilités pour la recherche comparative, en facilitant la collecte de données provenant de sources multiples qui peuvent représenter différentes périodes, pays, langues, etc.

Il y a aussi des défis. Le web scraping soulève des considérations éthiques et légales, et il n'y a généralement pas de réponse facile à ces questions. La plupart des comités d'éthique universitaires n'ont pas encore défini leur position concernant le web scraping et les meilleures pratiques pour l'évaluer. Le web scraping peut être difficile techniquement. Enfin, il y aura des défis liés aux données elles-mêmes - leur qualité, leur représentativité et leur biais.

Il n'y a pas de réponse facile aux défis juridiques et éthiques auxquels nous serons confrontés lors du web scraping, mais, il existe certaines meilleures pratiques que nous pouvons suivre. Par exemple, on peut respecter les directives fournies par les sites web dans leurs fichiers robots.txt, qui spécifient les autorisations et les restrictions pour le scraping. Nous pouvons personnaliser les user-agents et les en-têtes pour nous conformer aux politiques du site, essentiellement pour informer le webmaster qu'il s'agit d'une recherche et fournir des informations de contact ainsi que d'autres détails sur la raison pour laquelle nous le faisons. Nous devons être conscients des conditions d'utilisation des sites que nous raclons et respecter les restrictions d'utilisation des données. **SLIDE** Et nous devons réfléchir attentivement à la confidentialité et aux autres implications de la collecte d'informations personnelles sur Internet.

Je veux en dire un peu plus sur le fichier robots.txt, pour ceux qui ne l'ont peut-être pas vu auparavant ou qui ne savent pas ce que c'est. Mon conseil est de toujours *lire* les consignes indiquées dans le fichier robots.txt d'un site web. Je ne vais pas dire que vous devez les suivre à la lettre, car la plupart du temps, ces consignes n'ont pas été rédigées en pensant aux chercheurs universitaires, mais vous devriez toujours commencer par les inspecter. Le fichier robots.txt spécifie ce qui peut être accédé ou non par les robots d'exploration web, y compris les robots de scraping.

Maintenant, voyons comment trouver ce fichier robots.txt. Il s'agit d'un fichier texte situé à la racine d'un site web. **Demonstrate** then **SLIDE**

Pour comprendre le fichier de robots.txt, il y a quelques termes techniques que nous devons comprendre. Le premier c’est << User-agent >>. Cela fait référence au robot spécifique auquel s'appliquent les directives suivantes. Différents robots peuvent avoir des règles et des autorisations différentes qui leur sont spécifiées. Ce que nous voulons rechercher, c'est l'astérisque or le << wildcard>, car cela s'applique à tout le monde, y compris à nous. **SLIDE**

En dessous de << User-agent >>, on trouve généralement les directives << Allow >> et << Disallow >>. Elles indiquent quelles parties du site web sont autorisées ou non à être accessibles par le user-agent spécifié. La directive << Allow >> enumère les URL ou répertoires que notre scraper est autorisé à extraire, tandis que la directive "Disallow" enumère ceux qui ne peuvent pas être récupérés.

Enfin, vous trouverez parfois un << Crawl-delay >> dans certains fichiers robots.txt. Il s'agit d'une directive facultative qui spécifie le délai d'attente, en secondes, que notre algorithme ou notre bot de scraping doit respecter entre chaque demande successive au site web. Cela est utilisé pour gérer la charge sur le serveur. Trop de demandes, provenant de trop d'ordinateurs, trop rapidement, peuvent provoquer un plantage du serveur, ce qui est… mauvais. Nous voulons éviter cela à tout prix. **SLIDE**

D'accord, supposons que nous avons identifié un site web que nous souhaitons scraper, nous avons inspecté le fichier robots.txt et peut-être aussi les conditions d'utilisation. Maintenant, nous voulons effectuer un scrape. Comment faisons-nous ?

Avant d'arriver là, nous devons comprendre un peu l'Internet et les éléments constitutifs des sites web. La plupart des sites web sont construits à partir de trois langages de programmation principaux : HTML, CSS et JavaScript. Nous n'avons pas besoin d'être des experts dans ces langages de programmation pour pouvoir faire du web scraping en R, mais nous devons connaître les bases. **SLIDE**

Commençons par HTML. Qui a déjà travaillé en html ? HTML signifie HyperText Markup Language. C'est un langage utilisé pour structurer et présenter du contenu sur le web. Il utilise des balises pour définir des éléments HTML tels que des titres, des paragraphes, des listes, etc. Ces éléments peuvent être imbriqués les uns dans les autres pour créer une structure hiérarchique. Même si vous n'avez jamais écrit de code HTML auparavant, je suis sûr que vous le reconnaîtrez. Le web scraping consiste à extraire du contenu du << source code >> d'un site web. Lorsque nous concevons notre scraper, nous voulons inspecter ce source code pour voir comment il est structuré et élaborer un plan pour localiser et extraire le contenu souhaité. **SLIDE**

Pour accéder au << source code >> d'un site web, nous pouvons utiliser les outils de développement du navigateur. Chaque navigateur dispose d'un mode développeur. En faisant un clic droit sur une page web et en sélectionnant << Inspect >>. **SLIDE**

Un document HTML est composé de plusieurs parties essentielles qui contribuent à sa structure. L'élément <html> sert de racine du document HTML, encapsulant tous les autres éléments à l'intérieur. L'élément <head> contient des métadonnées sur le document, telles que le titre de la page, les liens vers des feuilles de style ou des scripts. L'élément <body> représente le contenu principal de la page HTML qui est visible par les utilisateurs. Il comprend du texte, des images, des vidéos, des liens et d'autres éléments qui constituent le contenu réel affiché dans le navigateur. Ces trois éléments travaillent ensemble pour définir la structure et la présentation d'un document HTML. **SLIDE**

À l'intérieur de ces trois niveaux du document HTML, qui représente la structure hiérarchique d'un document HTML, nous trouvons un large éventail d'éléments imbriqués qui contribuent au contenu et à la mise en page de la page web. Par exemple, les balises <p> sont utilisées pour définir des paragraphes de texte, permettant une présentation organisée et structurée des informations textuelles.

Ces balises servent de blocs de construction pour structurer le contenu d'une page web. Cependant, ces balises seules ne fournissent pas une spécificité suffisante pour cibler et extraire les informations précises dont nous avons besoin lors du web scraping. C'est là que les sélecteurs CSS entrent en jeu.

En utilisant les sélecteurs CSS, nous pouvons spécifier des critères au-delà des noms de balises génériques pour identifier et extraire précisément les éléments désirés. Les sélecteurs CSS nous permettent de cibler des éléments en fonction de divers attributs tels que les noms de classe, les identifiants (IDs) ou même leurs relations hiérarchiques au sein de la structure HTML. En utilisant des sélecteurs CSS, nous pouvons extraire des informations pertinentes à partir de pages web lors du scraping. Par exemple, les sélecteurs de classe (par exemple, **.container**) nous permettent de cibler des éléments avec des attributs de classe spécifiques, tandis que les sélecteurs d'identifiant (par exemple, **#titre-presentation**) nous aident à extraire des éléments avec des IDs uniques.

Une autre approche pour extraire des informations à partir de HTML consiste à utiliser << XPath >>. XPath est un langage utilisé pour naviguer dans les documents HTML. Avec XPath, nous pouvons définir des chemins et des modèles pour sélectionner des éléments en fonction de leur emplacement dans l'arborescence du document. **SLIDE**

Je veux prendre un moment pour mentionner un outil vraiment utile appelé SelectorGadget. SelectorGadget est un outil pratique qui nous aide à identifier et à générer automatiquement des sélecteurs CSS et des XPath pour des éléments HTML spécifiques. Il simplifie le processus de sélection et d'extraction de données à partir de pages web. **Demonstrate.**

Maintenant, une fois que nous avons l'URL de la page que nous voulons scraper, que nous avons inspecté le HTML en utilisant le mode développeur ou SelectorGadget, et que nous avons identifié les sélecteurs CSS ou les XPath qui nous donneront le contenu souhaité, il est temps d'extraire les informations. Nous commençons par envoyer ce qu'on appelle une << GET request >> au serveur. Si la requête réussit, elle extrait le document HTML d'une page web et l'envoie dans notre environnement R, où nous pouvons le parser et extraire les parties qui nous intéressent. **SLIDE**

Nous approchons du tutoriel, et c'est là que tout deviendra beaucoup plus clair. Mais d'abord, quelques petites choses supplémentaires que je pense que vous devriez savoir. En premier, le contenu dynamique. **SLIDE**

Tout d'abord, je tiens à souligner que rvest ne sera pas toujours suffisant. Bien que rvest soit un paquet puissant et populaire pour le web scraping en R, il cible principalement le contenu web statique, c'est-à-dire qu'il est conçu pour extraire des données à partir de pages web qui ne changent pas dynamiquement. Le contenu web statique fait référence aux pages web où la structure HTML et les données restent constantes dans le temps. **EXAMPLE and then SLIDE**

Pour gérer les scénarios de web scraping dynamique, rvest seul peut ne pas suffire. Dans de tels cas, une approche alternative consiste à utiliser des outils comme RSelenium ou à utiliser des API qui signifie Application Programming Interface.

Aujourd'hui, nous n'allons pas nous plonger dans les détails des APIs ou de Rselenium, mais je veux que vous sachiez qu'ils existent et que parfois vous devrez utiliser ces outils plutôt que rvest.

RSelenium est un paquet R qui permet d'automatiser les navigateurs web et d'interagir avec du contenu web dynamique. Il permet le scraping de pages web qui dépendent fortement de JavaScript en contrôlant un navigateur réel de manière programmatique, généralement appelé < remote browser >>. En automatisant les actions du navigateur, RSelenium vous permet d'extraire des données à partir de pages web qui sont rendues dynamiquement.

Alternativement, de nombreux sites web fournissent des API qui permettent aux développeurs d'accéder et de récupérer des données spécifiques de manière structurée. Parfois, ces API sont très coûteuses, par exemple l'API de Twitter coûte environ 45 000 dollars américains par mois pour les chercheurs, tandis que d'autres peuvent être gratuites. Les API sont un ensemble de règles et de protocoles qui permettent à différentes applications logicielles de communiquer entre elles, dans ce cas notre ordinateur avec le serveur backend d'une page web, nous permettant de demander et d'obtenir les informations souhaitées. **SLIDE**

Enfin, je tiens à mentionner qu'il est possible, et maintenant beaucoup plus facile qu'auparavant, d'exécuter nos scripts de web scraping sur un serveur distant programmé. Aujourd'hui, nous n'aborderons pas la façon de faire cela. Mais, il est important que vous sachiez que c'est possible. Par exemple, nous pouvons écrire des scénarios de scraping qui s'exécutent toutes les 10 minutes, toutes les heures, tous les jours, toutes les semaines, et regrouper les données dans un tableau hébergé quelque part dans le cloud, comme sur Google Sheets. Des plateformes telles que GitHub Actions, AWS et Heroku proposent des solutions de web scraping basées sur le cloud. Cela relève plutôt de l'avancé dans le monde du web scraping, mais avec de la pratique, vous pouvez certainement y arriver en quelques jours ou semaines, et il existe de nombreux tutoriels en ligne très bons qui vous guident pas à pas sur la marche à suivre.

D'accord, avez-vous des questions ? Je sais que c'était beaucoup d'informations. Mai, ne vous inquiétez pas, tout deviendra beaucoup plus clair pendant le tutoriel lorsque nous ferons réellement le web scraping avec rvest.