Semaine 12

Initiation à l'algorithmique et programmation

Revekka Kyriakoglou

Installing Python and Pandas

Installer Pandas :

- Ouvrez votre invite de commande et entrez :
  `pip install pandas`

Installing Python and Pandas

Installer Pandas :

- Ouvrez votre invite de commande et entrez :
  `pip install pandas`

Lien intéressant pour consulter :
`https://prog-learn.mds.ubc.ca/en/module1`

Exemple : Utilisation de Pandas

```
import pandas as pd

# Definition of dataset
mydataset = {
  'voitures': ["BMW", "Volvo", "Ford"],
  'passages': [3, 7, 2]
}
myvar = pd.DataFrame(mydataset) # Creation
print(myvar)
```

Sortie attendue

```
   voitures  passages
0  BMW              3
1  Volvo            7
2  Ford             2
```

Création d'un dictionnaire et chargement dans un DataFrame

```
# Definition d'un dictionnaire
data = {
    'Nom': ['Alice', 'Bob', 'Charles', 'Diane'],
    'Age': [25, 30, 35, 40],
    'Ville': ['Paris', 'Lyon', 'Marseille', 'Nantes']
}

# Charger le dictionnaire dans un DataFrame
df = pd.DataFrame(data)

# Afficher le DataFrame
print(df)
```

Sortie attendue

```
      Nom  Âge      Ville
0   Alice   25      Paris
1     Bob   30       Lyon
2  Charles   35  Marseille
3   Diane   40     Nantes
```

Points Clés

- Les clés du dictionnaire deviennent les noms des colonnes dans le DataFrame.
- Les valeurs associées à chaque clé deviennent les données des colonnes.
- Utilisez cette méthode pour une conversion rapide de données structurées en DataFrame pour une analyse plus approfondie.

Visualiser les Données dans un DataFrame

Afficher les premières et dernières lignes

```
# Afficher les premieres lignes
print(df.head())

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

# Afficher les dernieres lignes
print(df.tail())
```

Afficher un échantillon aléatoire de données

```
# Afficher 3 lignes aleatoires
print(df.sample(3))
```

Vérifier la structure du DataFrame

```
# Afficher le nombre de lignes et de colonnes
print(df.shape)

# Afficher les types de donnees des colonnes
print(df.dtypes)
```

Résumé statistique pour les données numériques

```
# Afficher le resume statistique
print(df.describe())
```

Utilisez `df.info()` pour obtenir un résumé complet, y compris l'utilisation de la mémoire.

L'inspection régulière des données est cruciale pour identifier les erreurs.

Qu'est-ce que `describe()` ?

`describe()` génère un résumé statistique des colonnes numériques dans un DataFrame, fournissant des informations clés telles que la moyenne, l'écart-type, les quantiles et les valeurs extrêmes.

Exemple de code

```
description = df.describe()
print(description)

# Calculer le resume statistique pour la colonne 'Age'
description_age = df['Age'].describe()
print(description_age)
```

```
count     4.0
mean     32.5
std       6.0
min      25.0
25%      28.75
50%      32.5
75%      36.25
max      40.0
Name: Age, dtype: float64
```

Par défaut, `describe()` ne considère que les colonnes numériques.

Pour inclure les objets ou les catégories, utilisez `describe(include='all')`.

Très utile pour une première analyse des données et pour détecter des anomalies ou des erreurs.

Localiser une ligne par position avec `iloc`

```
# Localiser la troisieme ligne
row = df.iloc[2]
```

Utiliser un argument d'index pour nommer vos propres indexes

```
# Nommer les indexes lors de la creation du DF
df = pd.DataFrame(data, index=['a', 'b', 'c'])
# Acceder a une ligne par son index nomme
row = df.loc['a']
```

Localiser les indexes nommés avec `loc`

```
# Localiser la ligne avec l'index 'b'
specific_row = df.loc['b']
# Localiser plusieurs lignes specifiques
rows = df.loc[['a', 'c']]
```

Utilisez `iloc` pour accéder aux lignes par leur position numérique.

Utilisez `loc` pour un accès plus intuitif par des labels.

Renommer les indexes peut faciliter la référence directe à des ensembles de données spécifiques.

Manipulation de Données

Supprimer des colonnes ou des lignes avec `drop()`

```
# Supprimer une colonne
df = df.drop('Ville', axis=1)

# Supprimer une ligne par index
df = df.drop([0], axis=0)
```

Renommer des colonnes avec `rename()`

```
# Renommer une colonne
df = df.rename(columns={'Age': 'age'})
```

Regrouper des données avec `groupby()`

```
# Grouper par 'Ville' et
# calculer la moyenne d'age
grouped = df.groupby('Ville')['Age'].mean()
print(grouped)
```

Conseils Pratiques

- Utilisez `drop()` pour nettoyer les DataFrames en supprimant les données inutiles.
- `rename()` est utile pour clarifier les noms des colonnes avant l'analyse.
- `groupby()` est essentiel pour l'analyse segmentée, permettant des comparaisons et des résumés statistiques par groupe.

Gestion des Données Manquantes avec Pandas (1/2)

```
import pandas as pd
import numpy as np

data = {
  'Noms': ["Alice", "Bob", None, "Diane"],
  ' ge ': [28, None, 35, 29],
  'Ville': ["Paris", "Lyon", "Marseille", None]
}
df = pd.DataFrame(data)
# Remplissage des valeurs manquantes
df_filled = df.fillna('Inconnu')
# Affichage du DataFrame original et modifie
print("DataFrame Original:")
print(df)
print("\nDataFrame Modifie:")
print(df_filled)
```

Gestion des Données Manquantes avec Pandas (2/2)

Sortie attendue

```
DataFrame Original:
    Noms   Âge       Ville
0  Alice  28.0       Paris
1    Bob   NaN        Lyon
2   None  35.0   Marseille
3  Diane  29.0        None

DataFrame Modifié:
    Noms      Âge       Ville
0  Alice     28.0       Paris
1    Bob  Inconnu        Lyon
2  Inconnu   35.0   Marseille
3  Diane     29.0     Inconnu
```

Importer des Données avec Pandas

Importer des données

```
# Importer des donn es depuis un fichier CSV
df = pd.read_csv('chemin/vers/le/fichier.csv')

# Importer des donn es depuis un fichier Excel
df = pd.read_excel('chemin/vers/le/fichier.xlsx')

# Importer des donn es depuis un fichier JSON
df = pd.read_json('chemin/vers/le/fichier.json')
```

Exporter des données

```
# Exporter des donn es vers un fichier CSV
df.to_csv('chemin/vers/le/nouveau_fichier.csv')

# Exporter des donn es vers un fichier Excel
df.to_excel('chemin/vers/le/nouveau_fichier.xlsx')

# Exporter des donn es vers un fichier JSON
df.to_json('chemin/vers/le/nouveau_fichier.json')
```

Utilisez les paramètres additionnels pour spécifier des options comme l'encodage, l'index, ou les feuilles spécifiques lors de l'importation ou de l'exportation.

Utilisation de Paramètres pour Importer et Exporter (1/2)

Paramètres Importants lors de l'Importation

```
# Lire un fichier avec un encodage specifique
df = pd.read_csv('fichier.csv', encoding='utf-8')

# Importer sans les lignes d'en-tete
df = pd.read_csv('fichier.csv', header=None)

# Lire un Excel et selectionner une feuille
df = pd.read_excel('fichier.xlsx', sheet_name='Feuille2')
```

Utilisation de Paramètres pour Importer et Exporter (2/2)

Paramètres Importants lors de l'Exportation

```
# Exporter un DataFrame en CSV sans l'index
df.to_csv('nouveau_fichier.csv', index=False)

# Exporter un DataFrame en Excel
# avec seulement certaines colonnes
df.to_excel('chemin/vers/nouveau_fichier.xlsx',
    columns=['Nom', 'Age'])

# Exporter un DF en JSON en format de tableau
df.to_json('chemin/vers/nouveau_fichier.json',
    orient='table')
```

Exercise 1

- Créez un fichier `data.csv` avec le contenu suivant :

  Name, Age, City, Proficiency
  Alice, 28, Paris, Advanced
  Bob, , Lyon, Intermediate
  Charlie, 32, Paris, Beginner
  Diane, 25, Marseille,

- Importer les données :
  Lire les données de `data.csv` dans un DataFrame pandas.

Solution

```
import pandas as pd
import numpy as np

# Import the data
df = pd.read_csv('data.csv')
print(df)
```

Détecter les valeurs manquantes

```
# Verifier les valeurs manquantes dans chaque colonne
print(df.isnull().sum())
```

Supprimer les valeurs manquantes

```
# Supprimer les lignes contenant des valeurs manquantes
df_cleaned = df.dropna()
# Supprimer les lignes ou toutes les colonnes sont manquantes
df_cleaned = df.dropna(how='all')
```

Remplacer les valeurs manquantes

```
# Remplacer les valeurs manquantes par une valeur specifique
df_filled = df.fillna('Inconnu')
# Remplacer les valeurs manquantes dans une colonne specifique
df['Age'].fillna(value=df['Age'].mean(), inplace=True)
```

Manipulation des Jeux de Données

Changer les Types de Données

```
# Changer le type d'une colonne
df['Age'] = df['Age'].astype(float)

# Convertir une colonne en type categoriel
df['Ville'] = df['Ville'].astype('category')
```

Trier les Données

```
# Trier les donnees par 'Age' dans un ordre croissant
df.sort_values(by='Age', inplace=True)

# Trier les donnees par 'Ville' et 'Age' dans un ordre decroissant
df.sort_values(by=['Ville', 'Age'], ascending=[True, False], inplace=T
```

Modifier les types de données peut aider à optimiser la mémoire et améliorer les performances des opérations.

Le tri est crucial pour l'analyse de données, permettant un accès rapide aux enregistrements basé sur des critères spécifiques.

Exercice 1 (suite)

Remplacer les valeurs manquantes

```
# Remplacer les valeurs manquantes dans 'Age' par la moyenne
df['Age'].fillna(df['Age'].mean(), inplace=True)

# Remplacer les valeurs manquantes dans 'City' par 'Inconnu'
df['City'].fillna('Unknown', inplace=True)
```

Utiliser `fillna` permet de remplacer les valeurs manquantes (NaN) par une valeur spécifiée. Ceci est utile pour :

- Maintenir l'intégrité des données lors des analyses.
- Eviter les erreurs lors des opérations de calcul ou de visualisation.

Conseils pratiques

- Pour les données numériques, considérez la moyenne, la médiane ou un calcul spécifique adapté au contexte.
- Pour les données catégorielles, comme les villes ou les noms, utilisez une catégorie ’Inconnu’, ’Unknown’ ou la plus fréquente.

Ajout de Colonnes Supplémentaires dans un DataFrame

Ajouter une colonne calculée

```
# Ajouter une colonne bas e sur les donn es existantes
df['Doubled␣Age'] = df['Age'] * 2

# Ajouter une colonne conditionnelle
df['Senior'] = df['Age'].apply(lambda x: 'Yes' if x >= 65
                                          else 'No')
```

Ajout de Colonnes Supplémentaires dans un DataFrame

Ajouter une colonne calculée

```
# Ajouter une colonne bas e sur les donn es existantes
df['Doubled Age'] = df['Age'] * 2

# Ajouter une colonne conditionnelle
df['Senior'] = df['Age'].apply(lambda x: 'Yes' if x >= 65
                                          else 'No')
```

Ajouter une colonne constante

```
# Ajouter une colonne avec une valeur constante
df['Study Status'] = 'Active'
```

Utiliser des opérations vectorisées pour des calculs efficaces sur des colonnes.

L'utilisation de la fonction `apply()` permet d'intégrer des conditions ou des fonctions personnalisées.

Ajouter des colonnes constantes peut être utile pour initialiser des indicateurs ou des étiquettes.

Exercise 1 (suite)

Exercise 1 (suite)

1. Ajoutez une nouvelle colonne intitulée "Years_of_Study" (années d'études) et définissez une règle simple pour la remplir :
   - 2 ans pour "Beginner" (débutant),
   - 4 ans pour "Intermediate" (intermédiaire),
   - 6 ans pour "Advanced" (avancé).
2. Filtrer le DataFrame pour n'inclure que les individus âgés de plus de 25 ans.
3. Exporter les données

Solution 1, question 2 (apply())

```
df['Years_of_Study'] = df['Language_Proficiency'].apply(lambda x: 6 if x == 'Advanced' else
                        4 if x == 'Intermediate' else
                        2 if x == 'Beginner' else
                        0)
```

Solution 2, question 2 (apply())

```
# Define a function
def calculate_years_of_study(proficiency):
    if proficiency == 'Beginner':
        return 2
    elif proficiency == 'Intermediate':
        return 4
    elif proficiency == 'Advanced':
        return 6
    else:
        return 0  # In case there is an unexpected proficiency level

# Apply the function to the 'Language Proficiency' column to create a new 'Years_of_Study' co
df['Years_of_Study'] = df['Language␣Proficiency'].apply(calculate_years_of_study)
```

Solution 3, question 2 (map())

```
import pandas as pd
import numpy as np

# Import the data
df = pd.read_csv('data.csv')

# Handling missing data
df['Age'].fillna(df['Age'].mean(), inplace=True)
df['City'].fillna('Unknown', inplace=True)

# Add 'Years of Study' based on 'Proficiency'
level_mapping = {'Beginner': 2, 'Intermediate': 4,  'Advanced': 6}
df['Years of Study'] = df['Level'].map(level_mapping)

# Filter rows where Age is greater than 25
df_filtered = df[df['Age'] > 25]

# Export the data
df_filtered.to_csv('processed_linguistic_data.csv', index=False)
```

Techniques de Combinaison de DataFrames

Utiliser `concat()`

```
# Concat ner deux DataFrames
df_concat = pd.concat([DATAFRAME1, DATAFRAME2])
```

- `concat()` regroupe une liste de DataFrames le long d'un axe.
- Utile lorsque les DataFrames partagent les mêmes colonnes.

Si vous ne précisez pas comment vous souhaitez concaténer les données, Python empilera les jeux de données horizontalement par défaut.

Si vous souhaitez que les données soient regroupées côte à côte, il suffit de spécifier `axis = 1` dans la fonction de concaténation pour placer simplement le second jeu de données à droite du premier.

```
import pandas as pd
df1 = pd.DataFrame({
    'A': ['A0', 'A1', 'A2'],
    'B': ['B0', 'B1', 'B2']})
df2 = pd.DataFrame({
    'A': ['A3', 'A4', 'A5'],
    'B': ['B3', 'B4', 'B5']})

# Concatenating dataframes vertically
vertical_concat = pd.concat([df1, df2], axis=0)
# Concatenating dataframes horizontally
horizontal_concat = pd.concat([df1, df2], axis=1)
print(vertical_concat)
print(horizontal_concat)
```

Utiliser join()

```
# Joindre deux DataFrames sur les index
df_joined = DATAFRAME1.join(DATAFRAME2, lsuffix='_left',
                                        rsuffix='_right')
```

- join() combine des DataFrames qui ont un index en commun.
- Pratique lorsque deux DataFrames partagent une colonne mais diffèrent dans les autres.

```
import pandas as pd
# Creating two dataframes without overlapping column names
df1 = pd.DataFrame({
    'A': ['A0', 'A1', 'A2'],
    'B': ['B0', 'B1', 'B2']
}, index=['K0', 'K1', 'K2'])

df2 = pd.DataFrame({
    'C': ['C0', 'C1', 'C2'],
    'D': ['D0', 'D1', 'D2']
}, index=['K0', 'K2', 'K3'])

joined_df = df1.join(df2, how='outer')
# other methods: 'inner', 'left', 'right'
print("Joined␣DataFrame:\n", joined_df)
```

```python
import pandas as pd

# Creating two dataframes with overlapping column names
df1 = pd.DataFrame({
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35]
})

df2 = pd.DataFrame({
    'Name': ['David', 'Eve', 'Frank'],
    'Age': [28, 22, 45]
})

joined_df = df1.join(df2, lsuffix='_left', rsuffix='_right')

print(joined_df)
```

Utiliser `merge()` (Optionnel)

- `merge()` est similaire à `join()` mais offre des options plus complexes.
- Pour les cas avancés, référez-vous à la documentation officielle.

Choisir la méthode de combinaison en fonction de la structure des données et des besoins de l'analyse.

`concat()` est la plus simple, suivie par `join()`, avec `merge()` pour des besoins plus complexes.

Exercice

1. Importer le jeu de données : `https://docs.google.com/spreadsheets/d/1b9Pdu75T9UxzBzIGEhcOmNX43dHSZjrVB0jrHmR3XNQ/edit?usp=sharing`
2. Charger le fichier CSV dans un DataFrame et utiliser des fonctions telles que `head()`, `info()`, et `describe()` pour comprendre sa structure et son contenu.
3. Traiter les valeurs manquantes dans la colonne ‘Score‘ en les remplaçant par la moyenne des scores de la langue correspondante.
4. Renommer les colonnes si nécessaire et convertir ‘EnrollmentDate‘ en format de date.
5. Calculer la moyenne des scores pour chaque langue et déterminer quel semestre a obtenu la meilleure moyenne de scores.
6. Fusionner ce jeu de données avec un autre DataFrame qui inclut des détails supplémentaires sur les cours (créé manuellement pour cet exercice).
7. Trier le DataFrame résultant par ‘Score‘ en ordre décroissant puis par ‘Name‘ en ordre croissant.
8. Sauvegarder le DataFrame final dans ‘processed_university_language_courses.csv‘.

Pourquoi visualiser les données ?

La visualisation aide à comprendre les tendances, les motifs et les valeurs aberrantes dans les données. Elle est cruciale pour une analyse et une communication efficaces des données.

- **Amélioration de la compréhension :** Les visualisations transforment les données complexes en graphiques et diagrammes plus simples à comprendre.
- **Communication efficace :** Les graphiques et autres formes de visualisation peuvent communiquer des informations rapidement et efficacement.
- **Analyse :** Les visualisations permettent aux analystes de voir de grandes quantités de données d'un coup d'œil. Cela est crucial pour l'analyse exploratoire des données, où l'identification rapide des patterns, des corrélations et des anomalies peut orienter les étapes suivantes de l'analyse.
- **Détection d'erreurs :** Les erreurs dans les données sont souvent plus visibles quand elles sont représentées graphiquement.
- **Accessibilité :** Les visualisations permettent à nous de comprendre les implications des données sans nécessiter une expertise en statistiques.

Types de visualisations

- **Diagrammes de dispersion :** Utilise des points pour représenter les valeurs obtenues pour deux variables différentes, l'une le long de l'axe des abscisses et l'autre le long de l'axe des ordonnées. Ce graphique est utilisé pour analyser la relation entre les deux variables afin de voir si elles sont corrélées.

Types de visualisations

- **Diagrammes de dispersion :** Utilise des points pour représenter les valeurs obtenues pour deux variables différentes, l'une le long de l'axe des abscisses et l'autre le long de l'axe des ordonnées. Ce graphique est utilisé pour analyser la relation entre les deux variables afin de voir si elles sont corrélées.
- **Graphiques linéaires :** Suivi des changements sur des périodes courtes ou longues. Les graphiques linéaires. C'est utilisé pour montrer les tendances au fil du temps, comme l'évolution des revenus mensuels, le trafic quotidien sur un site web, ou les changements annuels dans les chiffres de vente.


Creating a meaningful line graph requires a sequential or time-based variable.

Types de visualisations

- **Diagrammes de dispersion :** Utilise des points pour représenter les valeurs obtenues pour deux variables différentes, l'une le long de l'axe des abscisses et l'autre le long de l'axe des ordonnées. Ce graphique est utilisé pour analyser la relation entre les deux variables afin de voir si elles sont corrélées.
- **Graphiques linéaires :** Suivi des changements sur des périodes courtes ou longues. Les graphiques linéaires. C'est utilisé pour montrer les tendances au fil du temps, comme l'évolution des revenus mensuels, le trafic quotidien sur un site web, ou les changements annuels dans les chiffres de vente.

   Creating a meaningful line graph requires a sequential or time-based variable.
- **Histogrammes :** Les histogrammes regroupent les données numériques en intervalles, affichant la fréquence des points de données dans chaque intervalle. Idéal pour montrer la distribution des données, comme la distribution des scores d'examen ou la distribution des âges d'une population.

Types de visualisations

- **Diagrammes de dispersion :** Utilise des points pour représenter les valeurs obtenues pour deux variables différentes, l'une le long de l'axe des abscisses et l'autre le long de l'axe des ordonnées. Ce graphique est utilisé pour analyser la relation entre les deux variables afin de voir si elles sont corrélées.
- **Graphiques linéaires :** Suivi des changements sur des périodes courtes ou longues. Les graphiques linéaires. C'est utilisé pour montrer les tendances au fil du temps, comme l'évolution des revenus mensuels, le trafic quotidien sur un site web, ou les changements annuels dans les chiffres de vente.

   Creating a meaningful line graph requires a sequential or time-based variable.
- **Histogrammes :** Les histogrammes regroupent les données numériques en intervalles, affichant la fréquence des points de données dans chaque intervalle. Idéal pour montrer la distribution des données, comme la distribution des scores d'examen ou la distribution des âges d'une population.
- **Diagrammes en barres :** Utilisés pour comparer les montants ou les fréquences à travers différentes catégories. Les barres peuvent être affichées horizontalement ou verticalement.

Types de visualisations

- **Diagrammes de dispersion :** Utilise des points pour représenter les valeurs obtenues pour deux variables différentes, l'une le long de l'axe des abscisses et l'autre le long de l'axe des ordonnées. Ce graphique est utilisé pour analyser la relation entre les deux variables afin de voir si elles sont corrélées.
- **Graphiques linéaires :** Suivi des changements sur des périodes courtes ou longues. Les graphiques linéaires. C'est utilisé pour montrer les tendances au fil du temps, comme l'évolution des revenus mensuels, le trafic quotidien sur un site web, ou les changements annuels dans les chiffres de vente.

   Creating a meaningful line graph requires a sequential or time-based variable.
- **Histogrammes :** Les histogrammes regroupent les données numériques en intervalles, affichant la fréquence des points de données dans chaque intervalle. Idéal pour montrer la distribution des données, comme la distribution des scores d'examen ou la distribution des âges d'une population.
- **Diagrammes en barres :** Utilisés pour comparer les montants ou les fréquences à travers différentes catégories. Les barres peuvent être affichées horizontalement ou verticalement.
- **Cartes de chaleur :** Représente les données par des variations de couleur. Les couleurs fournissent un résumé visuel de l'information qui facilite l'identification des motifs, y compris les concentrations ou les régions particulières de valeurs hautes et basses.

Exemple : Visualiser des données avec Pandas

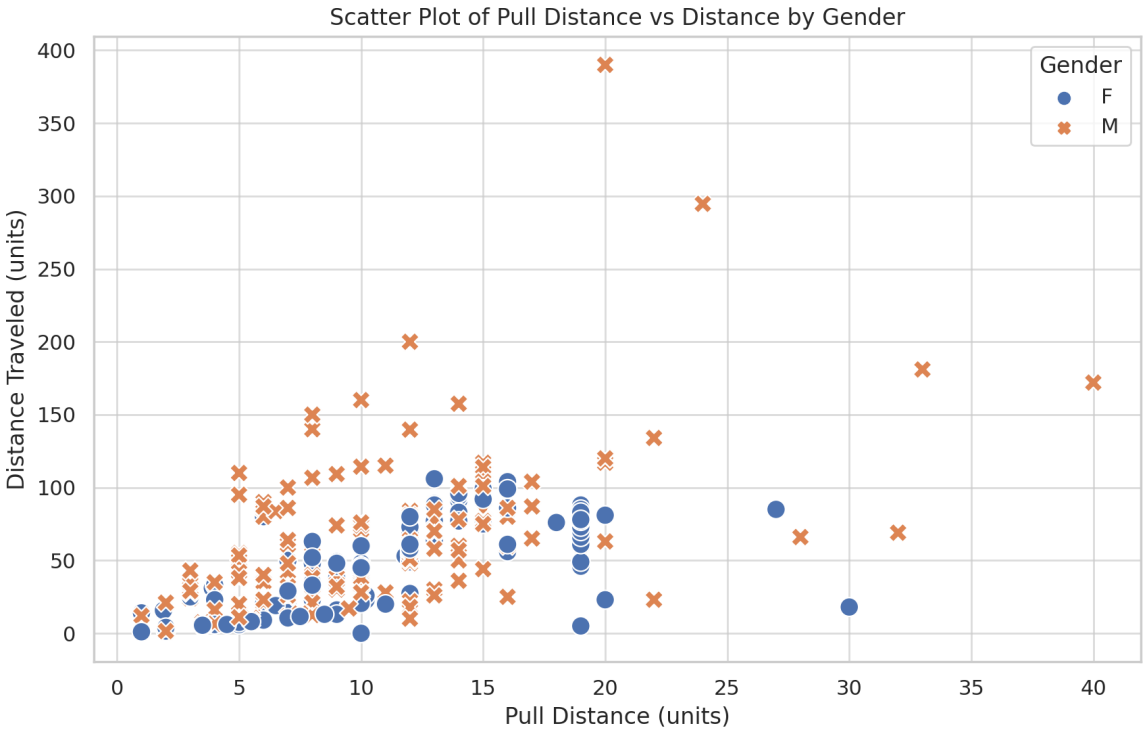
Pour la visualisation des données, nous utiliserons les données et l'exemple fournis par l'université Miamy.

Données :
https://miamioh.instructure.com//files//3082832//download?download_frd=1

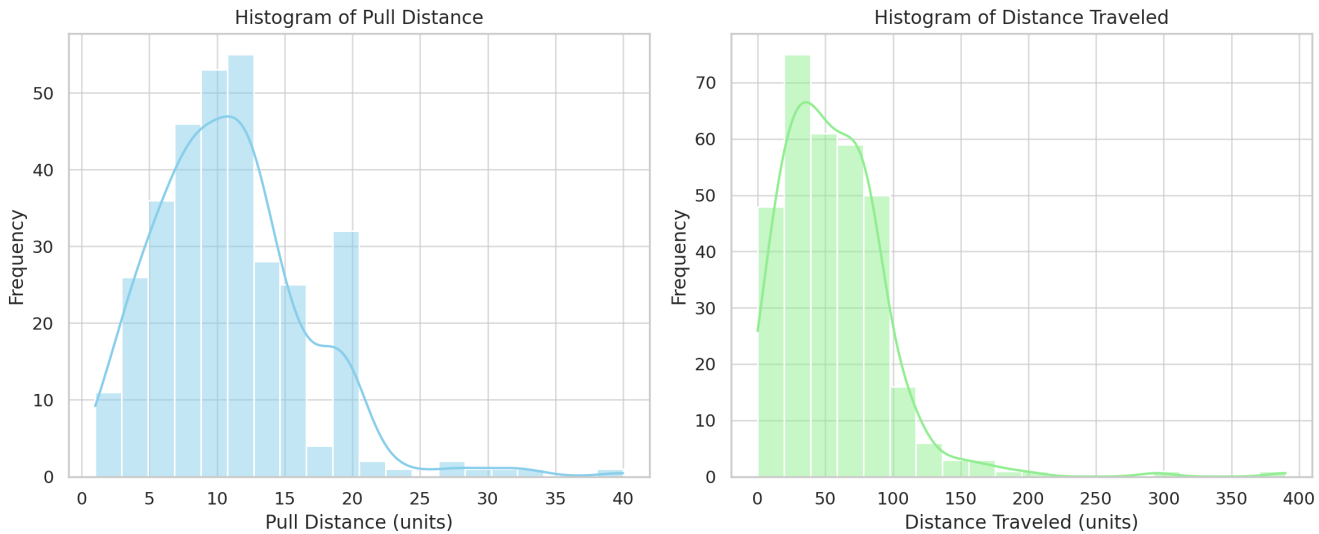
```
from urllib.request inport urlretrieve

url = 'https://miamioh.instructure.com//files//...'

urlretrieve(url, 'exemple_miami.csv')

df = pd.read_csv('exemle_miami.csv', sep = ';')
print(df.head())
```

https://miamioh.edu/centers-institutes/center-for-analytics-data-science/students/coding-tutorials/python/visualization.html

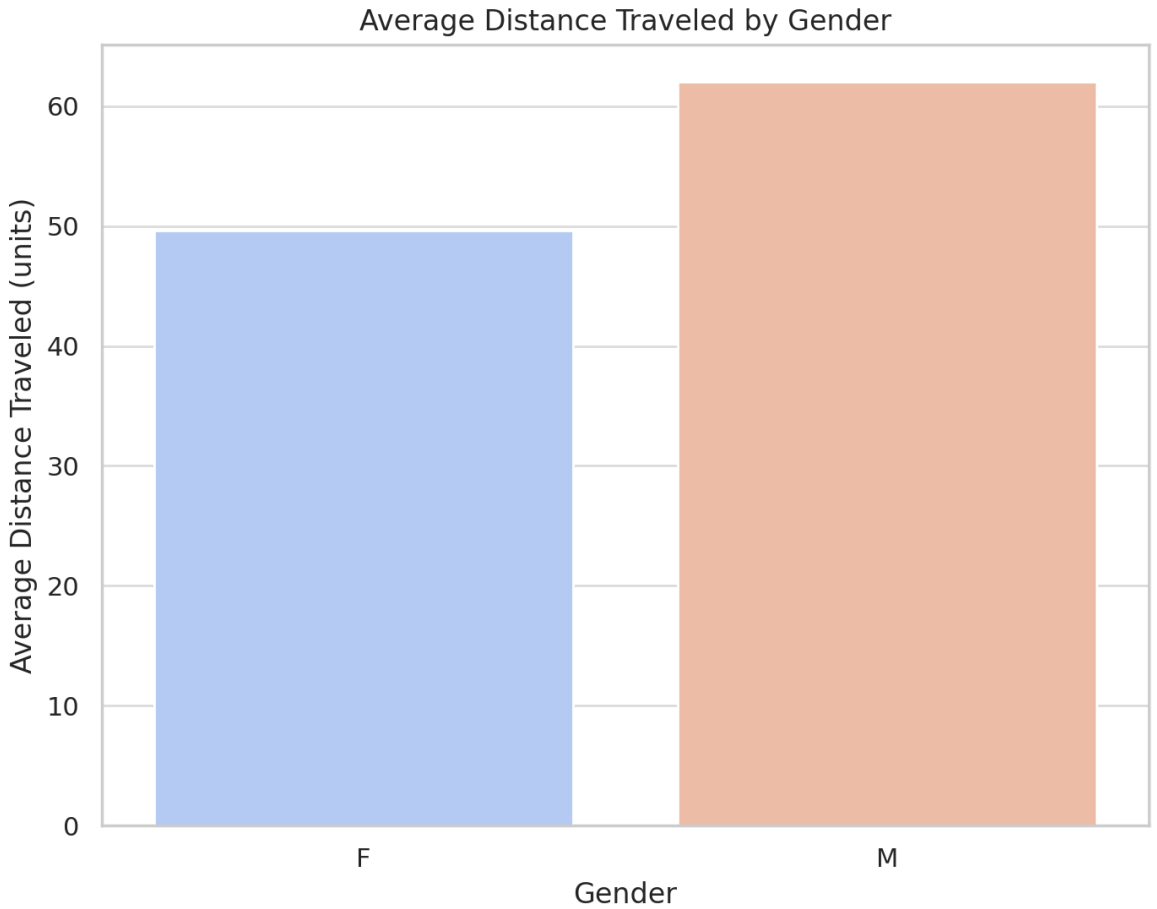
Diagramme de dispersion entre la Distance de Tirage et la Distance Parcourue



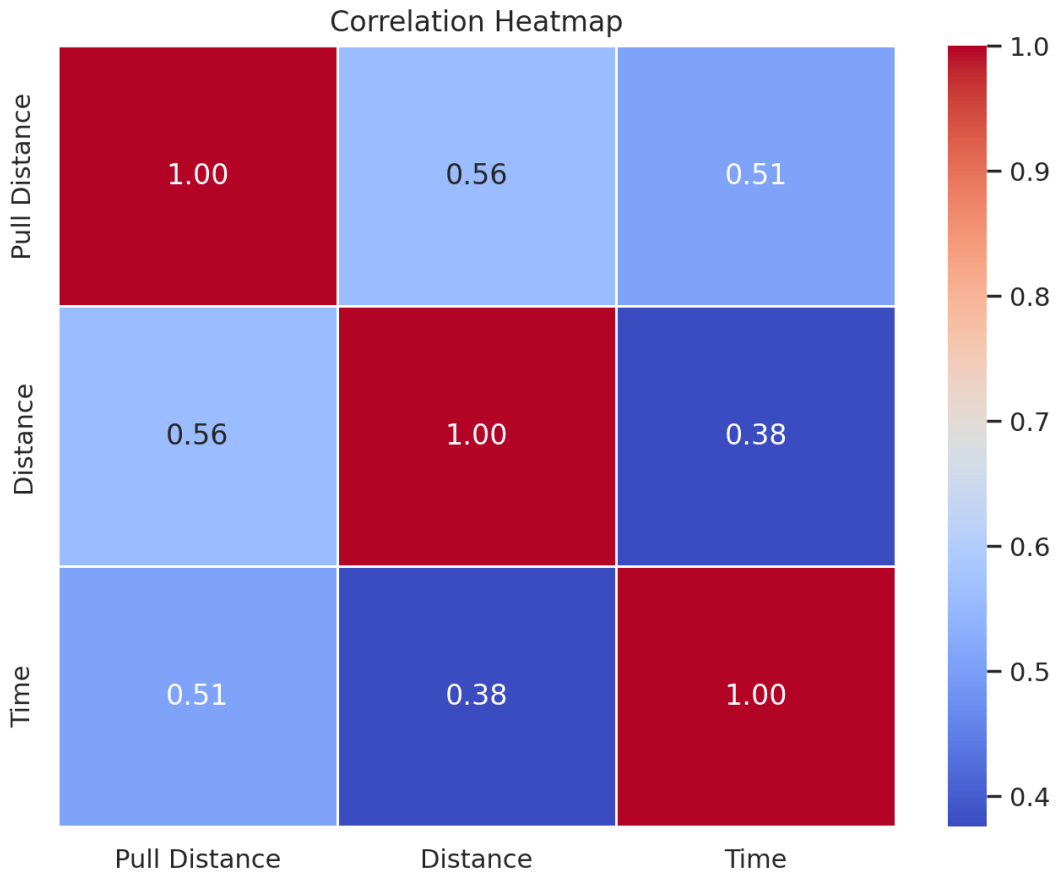
Histogrammes des Distances de Tirage et des Distances Parcourues



Distance Moyenne Parcourue par Genre



Carte de Chaleur de Corrélation



Outils de visualisation

Bibliothèques Python populaires pour la visualisation de données :

- Matplotlib
- Seaborn
- Pandas plotting

pip install pandas matplotlib seaborn

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
data = pd.read_csv('path_to_your_dataset.csv')

# Set visualization style
sns.set(style="whitegrid")

# Scatter Plot for Pull Distance vs Distance
plt.figure(figsize=(10, 6))
sns.scatterplot(data=data, x='Pull Distance', y='Distance',
        hue='Gender', style='Gender',
        markers={'F': 'o', 'M': 'X'},
        s=100)
plt.title('Scatter Plot: Pull Distance vs Distance by Gender')
plt.xlabel('Pull Distance')
plt.ylabel('Distance Traveled')
plt.legend(title='Gender')
plt.savefig('scatter_plot.png')
plt.show()
```

```
# Histograms for Pull Distance and Distance
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
sns.histplot(data['Pull Distance'], bins=20,
                kde=True,
                color='skyblue')
plt.title('Histogram of Pull Distance')
plt.xlabel('Pull Distance (units)')
plt.ylabel('Frequency')
plt.subplot(1, 2, 2)
sns.histplot(data['Distance'], bins=20,
                kde=True,
                color='lightgreen')
plt.title('Histogram of Distance Traveled')
plt.xlabel('Distance Traveled')
plt.ylabel('Frequency')
plt.savefig('histograms.png')
plt.show()
```

```
# Bar Chart for Average Distance Traveled by Gender
plt.figure(figsize=(8, 6))
sns.barplot(data=data, x='Gender', y='Distance',
                ci=None,
                palette='coolwarm')
plt.title('Average Distance Traveled by Gender')
plt.xlabel('Gender')
plt.ylabel('Average Distance Traveled')
plt.savefig('bar_chart.png')
plt.show()

# Heatmap for Correlation: Pull Distance, Distance, Time
correlation_matrix = data[['Pull Dist', 'Dist', 'Time']].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True,
                cmap='coolwarm',
                fmt=".2f",
                linewidths=.5)
plt.title('Correlation Heatmap')
plt.savefig('heatmap.png')
plt.show()
```

Exercice (suite)

Créez des graphes et expliquez les données de l'exercice précédent.