USTHB : 2023/2024

M1-Réseaux et Télécommunications : TP Programmation orientée objets en Python

Compte-rendu du TP1 : Initiation et prise en main du langage Python (Modules : NumPy, matplotlib, Pandas, fichiers CSV, etc)

Nom : CHIKER **Prénom :** Imad **Matricule :** 202031066756

Nom : BENNOUI **Prénom :** Romeissa **Matricule :** 202031062680

Section : RT-B- **Sous-groupe : 2**

But du TP :

Ce TP opte pour la familiarisation avec le langage de programmation Python. Nous abordons l'installation de Python, l'intégration des librairies, l'importation de modules, et la manipulation des fichiers.

1- Phase de prétraitement :

Nous avons d'abord importé notre fichier csv à l'aide de Pandas, avec la commande « pd.read_csv » qui lit le fichier csv et crée notre Dataframe Df.

```
import pandas as pd
Df=pd.read_csv('C:/Users/admin/Documents/Telecom/M1 RT/TP POO/titanic-passengers.csv', delimiter=';' )
Df
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	343	No	2	Collander, Mr. Erik Gustaf	male	28.0	0	0	248740	13.0000	NaN	S
1	76	No	3	Moen, Mr. Sigurd Hansen	male	25.0	0	0	348123	7.6500	F G73	S
2	641	No	3	Jensen, Mr. Hans Peder	male	20.0	0	0	350050	7.8542	NaN	S
3	568	No	3	Palsson, Mrs. Nils (Alma Cornelia Berglund)	female	29.0	0	4	349909	21.0750	NaN	S
4	672	No	1	Davidson, Mr. Thornton	male	31.0	1	0	F.C. 12750	52.0000	B71	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	10	Yes	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C
887	61	No	3	Sirayanian, Mr. Orsen	male	22.0	0	0	2669	7.2292	NaN	C
888	535	No	3	Cacic, Miss. Marija	female	30.0	0	0	315084	8.6625	NaN	S
889	102	No	3	Petroff, Mr. Pastcho ("Pentcho")	male	NaN	0	0	349215	7.8958	NaN	S
890	428	Yes	2	Phillips, Miss. Kate Florence ("Mrs Kate Louis...	female	19.0	0	0	250655	26.0000	NaN	S

891 rows × 12 columns

Nous en avons déduit que notre Dataframe comporte 891 lignes et 12 colonnes.
Après cela, nous avons exécuté la commande « Df.isnull().sum() » pour calculer le nombre de NaN (Not a Number/ information manquante) dans chaque catégorie.

```
Df.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Nous avons donc déduit qu'il y avait 177 NaN dans la catégorie « Age », 687 NaN dans la catégorie « Cabin » et 2 NaN dans la catégorie « Embarked ».

Nous avons également testé la commande « Df.drop » afin de supprimer la colonne « Embarked ».

```
Df.drop('Embarked', axis=1)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	343	No	2	Collander, Mr. Erik Gustaf	male	28.0	0	0	248740	13.0000	NaN
1	76	No	3	Moen, Mr. Sigurd Hansen	male	25.0	0	0	348123	7.6500	F G73
2	641	No	3	Jensen, Mr. Hans Peder	male	20.0	0	0	350050	7.8542	NaN
3	568	No	3	Palsson, Mrs. Nils (Alma Cornelia Berglund)	female	29.0	0	4	349909	21.0750	NaN
4	672	No	1	Davidson, Mr. Thornton	male	31.0	1	0	F.C. 12750	52.0000	B71
...	...	...	...	...	...	...	...	...	...	...	...
886	10	Yes	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN
887	61	No	3	Sirayanian, Mr. Orsen	male	22.0	0	0	2669	7.2292	NaN
888	535	No	3	Cacic, Miss. Marija	female	30.0	0	0	315084	8.6625	NaN
889	102	No	3	Petroff, Mr. Pastcho ("Pentcho")	male	NaN	0	0	349215	7.8958	NaN
890	428	Yes	2	Phillips, Miss. Kate Florence ("Mrs Kate Louis...	female	19.0	0	0	250655	26.0000	NaN

891 rows × 11 columns

Nous obtenons ainsi 891 lignes et 11 colonnes au lieu de 12 comme avant.
L'argument « axis » indique l'axe sur lequel l'opération de suppression doit être effectuée. Lorsque « axis= 1 », cela signifie que nous supprimons une colonne. Si « axis=0 », cela signifierait que nous supprimons une ligne.
Par défaut, « drop () » ne modifie pas le DataFrame d'origine, mais retourne plutôt un nouveau DataFrame avec la colonne supprimée. Pour la modifier nous devons ajouter l'argument « inplace=True » comme suit : « Df.drop('Embarked', axis=1, inplace=True) ».

Avant de passer à la phase de visualisation, nous avons dû remplacer les informations manquantes (NaN) par des valeurs appropriées.
Nous avons donc commencé par la catégorie « Cabin » en exécutant ce code :

```
number_of_elements=len(Df["Cabin"])
print("Number of elements:",number_of_elements)
print(Df["Cabin"].value_counts())
Df["Cabin"].fillna('G6',inplace=True)
Df.tail()
```

```
Number of elements: 891
Cabin
G6             4
B96 B98        4
C23 C25 C27    4
F33            3
D              3
              ..
C91            1
D45            1
F G63          1
A34            1
E63            1
Name: count, Length: 147, dtype: int64
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
886	10	Yes	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	G6	C
887	61	No	3	Sirayanian, Mr. Orsen	male	22.0	0	0	2669	7.2292	G6	C
888	535	No	3	Cacic, Miss. Marija	female	30.0	0	0	315084	8.6625	G6	S
889	102	No	3	Petroff, Mr. Pastcho ("Pentcho")	male	NaN	0	0	349215	7.8958	G6	S
890	428	Yes	2	Phillips, Miss. Kate Florence ("Mrs Kate Louis...	female	19.0	0	0	250655	26.0000	G6	S

Ce code commence donc par afficher le nombre total d'éléments non manquants dans la colonne "Cabin", puis affiche le nombre des valeurs uniques dans cette colonne. Ensuite, il remplace les valeurs manquantes par 'G6' dans la colonne "Cabin" et affiche les cinq dernières lignes du DataFrame pour vérifier les modifications. Nous avons donc traité les données manquantes dans la colonne "Cabin" en leur attribuant la valeur 'G6'.

Nous sommes passés ensuite à la catégorie « Age » en exécutant ce code :
« Df['Age'].fillna(Df['Age'].mean(), inplace=True) ».

Ce code remplace les valeurs manquantes dans la colonne "Age" par la moyenne d'âge de l'ensemble de données. Cela permet de gérer les données manquantes de manière simple en utilisant la moyenne pour les remplacer. Cela peut être utile pour préparer les données en vue d'une analyse statistique ou de la création de modèles prédictifs.

Enfin nous avons traité la catégorie « Embarked » en éxecutant ce code :

```
Df['Age'].fillna(Df['Age'].mean(), inplace=True)
```

```
print(Df["Embarked"].value_counts())
```

```
Embarked
S    644
C    168
Q     77
Name: count, dtype: int64
```

```
Df["Embarked"].fillna('S',inplace=True)
```

Ce code remplace les valeurs manquantes dans la colonne "Embarked" par la lettre 'S'.

Pour vérifier que nous n'avons plus d'informations manquantes « NaN » nous avons ré-exécuté le code « Df.isnull().sum() ».

```
Df.isnull().sum()
```

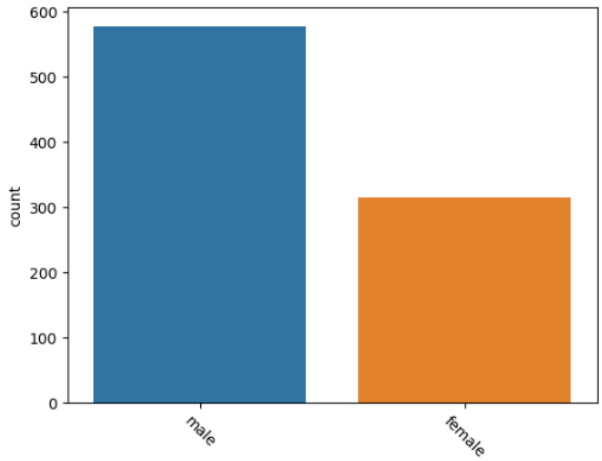
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

Nous en concluons qu'effectivement, il n'y a plus d'informations manquantes et pouvons donc passer à la 2ème phase.

2- Phase de visualisation :

Grace aux bibliothèques Seaborn et Matplotlib nous avons créé un graphique qui montre la répartition des sexes ('Male' et 'Female') dans le DataFrame Df en exécutant ce code :

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(x='Sex', data=Df)
plt.xticks(rotation=-45)
plt.show()
```



- sns.countplot(x='Sex', data=Df): crée un countplot à partir des données dans le DataFrame Df. Le graphique compte les occurrences de chaque valeur unique (male/female) dans la colonne 'Sex' et affiche le nombre d'occurrences sur l'axe y. L'argument x='Sex' spécifie que la colonne 'Sex' sera utilisée comme variable à compter, et data=Df indique que les données proviennent du DataFrame Df.
- plt.xticks(rotation=-45): incline les étiquettes de l'axe des x de 45 degrés pour une meilleure lisibilité.
- plt.show(): affiche le graphique à l'écran.

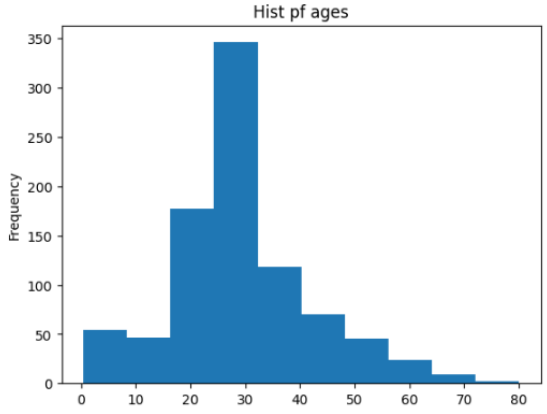
Nous pouvons voir qu'il y avait plus d'hommes que de femmes à bord du Titanic.

Nous avons par la suite visualisé la distribution des âges grâce à un histogramme en exécutant le code suivant :

- plt.title("Hist pf ages"): définit le titre du graphique comme "Hist of ages »
- plt.xlabel("Age"): définit l'étiquette de l'axe des x en utilisant la fonction comme "Age".
- Df['Age'].plot.hist(): génère l'histogramme lui-même. Elle accède à la colonne "Age" du DataFrame Df et elle utilise .plot.hist() pour créer un histogramme de cette colonne.

```
plt.title("Hist pf ages")
plt.xlabel("Age")
Df['Age'].plot.hist()
```

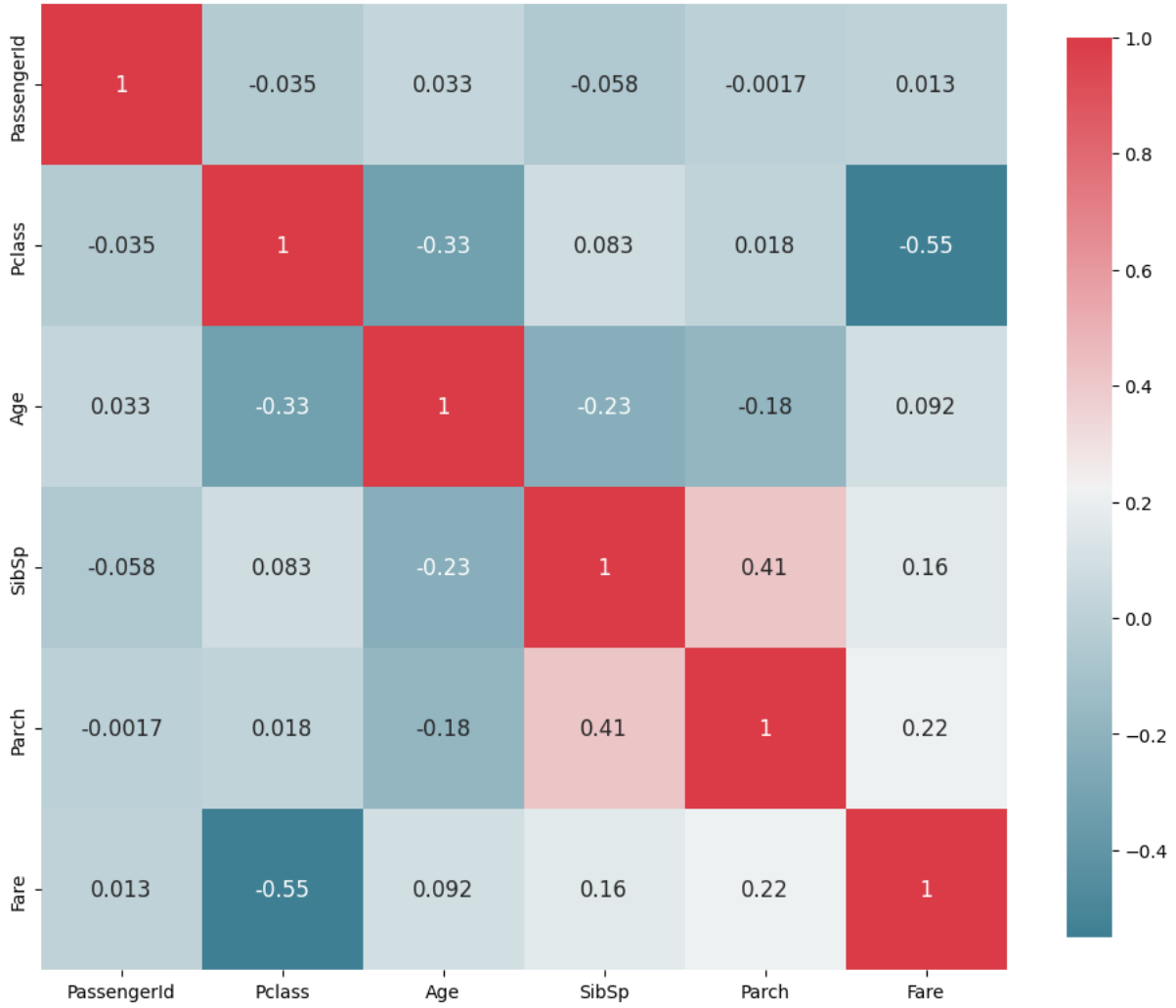
```
<AxesSubplot: title={'center': 'Hist pf ages'}, ylabel='Frequency'>
```



Enfin nous avons exécuté cette fonction :

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
def plot_correlation_map( Df ):
 corr = Df.corr()
 s , ax = plt.subplots( figsize =( 12 , 10 ) )
 cmap = sns.diverging_palette( 220 , 10 , as_cmap = True )
 s = sns.heatmap(
 corr,
 cmap = cmap,
 square=True,
 cbar_kws={ 'shrink' : .9 },
 ax=ax,
 annot = True,
 annot_kws = { 'fontsize' : 12 }
 )
plot_correlation_map(Df)
```

Et avons obtenu cette figure :



Cette fonction commence par calculer la matrice de corrélation en appelant la méthode .corr() sur le DataFrame d'entrée Df. La matrice de corrélation est une matrice carrée qui montre les coefficients de corrélation entre toutes les paires de variables numériques dans le DataFrame. Une valeur proche de 1 indique une corrélation positive forte, -1 une corrélation négative forte, et 0 une absence de corrélation.

Ensuite, la fonction crée une figure (subplot) avec une taille de 12x10 pouces pour afficher le graphique. Elle utilise la bibliothèque Seaborn pour définir une palette de couleurs divergentes (diverging_palette) qui permet de distinguer visuellement les valeurs de corrélation positives et négatives de la corrélation.
La fonction utilise également Seaborn pour créer un heatmap (carte thermique) en utilisant la matrice de corrélation calculée précédemment. Le paramètre square=True crée un heatmap carré. Le paramètre cbar_kws={'shrink': .9} réduit la taille de la barre de couleur sur le côté du graphique. Les annotations sont ajoutées aux cellules du heatmap pour afficher les valeurs de corrélation avec une taille de police spécifiée (12).
L'utilité principale de cette fonction est de visualiser les relations de corrélation entre les variables numériques dans un DataFrame. Cela peut être utile pour l'analyse des données, car il permet de repérer facilement les variables qui sont fortement corrélées entre elles.

- Nous pouvons voir que Fare a une bonne corrélation avec PClass, et que Parch a également une bonne corrélation avec SibSp.

Paragraphe en anglais :

This function starts by computing the correlation matrix by calling the .corr() method on the input DataFrame Df. The correlation matrix is a square matrix that displays the correlation coefficients between all pairs of numeric variables in the DataFrame. A value close to 1 indicates a strong positive correlation, -1 indicates a strong negative correlation, and 0 indicates no correlation.

Next, the function creates a figure (subplot) with a size of 12x10 inches to display the graph. It uses the Seaborn library to define a diverging color palette (diverging_palette) that visually distinguishes positive and negative correlation values.

The function also uses Seaborn to create a heatmap using the previously calculated correlation matrix. The parameter square=True creates a square heatmap. The parameter cbar_kws={'shrink': .9} reduces the size of the color bar on the side of the graph. Annotations are added to the cells of the heatmap to display the correlation values with a specified font size (12).

The primary utility of this function is to visualize the correlation relationships between numeric variables in a DataFrame. This can be valuable for data analysis as it allows easy identification of variables that are strongly correlated with each other.