



université PARIS-SACLAY

DATA MINING

Réalisé Par :

**BOUDJEMAI Ali*

Master 2 DataScale

Année 2022/2023

TABLE DES MATIÈRES

TABLE DES MATIÈRES

INTRODUCTION GENERALE :	1
CHAPITRE I : DEFINITION ET EXPLICATION DES CONCEPTS	
INTRODUCTION:	1
I.1.GENERALITES:	2
I.2.SELECTION D'ATTRIBUTS:	3
I.3.ROUGH SET:	4
I.4.EXPLICATION ET IMPLEMENTATION DE ROUGH SET :	5
I.5. FUZZY ROUGH SET :	6
I.6.CONCLUSION:	7
CHAPITRE II : EXPLICATION ET IMPLEMENTATION DE QUICK REDUCT	
INTRODUCTION :	1
I.1. EXPLICATION QUIK REDUCT :	2
I.2. IMPLEMENTATION QUIK REDUCT :	3
I.2.1. CHOISIR UN PETIT DATASET :	4
I.2.2. APPLIQUER QUIK REDUCT :	5
I.3. DIFFERENCE ENTRE QUIK REDUCT ET FUZZY ROUGH SET :	6
I.4. CONCLUSION :	7
CHAPITRE III : IMPLEMENTATION	
INTRODUCTION :	1
I.1.SECTION 1 :	2
I.1.1. LE CHOIX DU DATASET :	2

I.1.2. LA THEMATIQUE A RESOUDRE AVEC LE DATASET :	2
I.2. SECTION 2 (DATA UNDERSTANDING):	3
I.2.1 COMPREHENSION DES DONNEES:	3
I.2.2 LA VISUALISATION DES DONNEES :	3
I.3. SECTION 3 (DATA CLEANING):	4
I.3.1. DETECTION DES MISSING VALUES :	4
I.2.2 CHOISIR UNE METHODE ET CORRIGER LES MISSING VALEUS :	4
I.4. SECTION 4 (SELECTION D'ATTRIBUT):	5
I.4.1. SELECTION D'ATTRIBUT AVEC QuickReduct:	5
I.4.2. APPLIQUATION DE ROUGH SET THEORY:	5
I.4.3. APPLIQUATION DE FUZZY ROUGH SET :	5
I.4.3. APPLIQUATION DE Pearson Corrélation :	5
 CHAPITRE IV : EXPERIMENTATION ET DESCUSSION DES RESULTATS	
I.1.1. APPLIQUER DEUX ALGORITHMES DE MACHINE LERANINIG.....	1
I.1.2. APPLIQUER LES DIFFERENTES METRIQUES.....	2
I.1.3. INTERPRETATION DES RESULTATS :	3
I.5.CONCLUSION :	6
I.5.2. LIMITES DE QUICK REDUCT :	6
I.5.3. AMELIORATION DE QUICK REDUCT :	6
CONCLUSION GENERALE :	7

INTRODUCTION GENERALE



INTRODUCTION GÉNÉRALE :

Aujourd'hui, le monde vit une explosion quantitative des données numériques, Le volume gigantesque de données numériques, implique de mettre en œuvre de nouveaux ordres de grandeur concernant la capture, le stockage, la recherche, le partage, l'analyse et la visualisation des données. Savoir traiter toutes ces masses de données afin d'arriver à tirer de la connaissance. Ces grands volumes de données, que les data center reçoivent chaque jour, proviennent de nombreuses sources numériques : les réseaux sociaux, les médias, l'OpenData, le Web, des bases de données privées, publiques à caractère commercial ou scientifique.

Afin d'extraire de la connaissance de ces grands volumes de données, ils existent toutes une science avec beaucoup (d'algorithmes, technique, heuristique, ...etc.), que les chercheurs et ingénieurs adoptent afin d'avoir des informations très précises, prédictive dans la plupart de temps pour aider les entreprises dans leurs stratégies et leurs prises de décision.

Grâce à l'analyse de données, beaucoup de secteurs ont vécu une évolutivité exponentielle, par exemple dans la médecine, ils parviennent avec le Big Data à prédire l'accouchement prochain de femmes enceintes, de prédire des incendies, suivre des ventes en temps réels et optimiser la gestion des stocks comme le cas d'Amazon.

Dans ce contexte, nous allons voir une heuristique très utilisée pour l'extraction de la connaissance qui est la réduction de dimensionnalité, ensuite nous allons appliquer des modèles de classification (Algorithmes des machine Learning). Pour cela nous avons divisé notre travail en 4 chapitres organisé selon le plan suivant :

- ✓ Le premier chapitre consiste à définir et à expliquer les concepts abordés et l'implémentation de QUICK REDUCT.
- ✓ Le deuxième chapitre se rapporte sur l'explication et l'implémentation de QUICK REDUCT.
- ✓ Le troisième chapitre est consacré à l'implantations en python des différentes étapes du pre-processing et l'implémentation des Algorithmes de feature selection.
- ✓ Le quatrième chapitre c'est l'expérimentation et l'interprétation et discussion des résultats

CHAPITRE I :

DEFINITION ET EXPLICATION DES CONCEPTS

INTRODUCTION :

Dans ce chapitre nous allons d'abord définir c'est quoi le DATA MINING, puis on va faire un focus sur la sélection d'attributs, afin de le bien comprendre. Ainsi en va voir spécialement une technique spécifique de sélection d'attribut « **ROUGH SET THEORY** » qui signifie théorie des ensembles approximatifs et présenter la théorie de sélection d'attributs avec ce dernier. Ensuite on va le comparer avec les autres algorithmes qui existent et terminer ce chapitre avec une conclusion.

I.1.GENERALITES :

- ✓ **DATA MINING** : c'est la recherche automatique de grandes quantités de données afin de découvrir des tendances et des modèles qui vont au-delà de la simple analyse. Il est toutefois toujours couplé avec du machine Learning. Il est utilisé pour aider les entreprises à optimiser leur avenir. Il leur permet de comprendre le passé et le présent et de faire des prédictions précises sur ce qui est susceptible d'arriver. Il répond aussi à des besoins d'augmentation de revenus. Il possède 4 tâches principales qui sont :

- **La description.**
- **L'estimation.**
- **La prévision.**
- **La classification.**
- **Le clustering.**
- **L'association.**

- ✓ **Data Selection :**

La sélection des données consiste à sélectionner les données pour la découverte de connaissances. Cela peut nécessiter de sélectionner les données à partir d'un référentiel existant ou de créer une source unique de données à partir de plusieurs sources.

- ✓ **Data Cleansing/ Pre-processing :**

Cette étape consiste à transformer les données afin de les rendre appropriées pour l'analyse sous-jacente et la découverte de connaissances. Les données peuvent contenir des attributs redondants qui n'ajoutent pas grand-chose à notre information ou peuvent contenir des attributs totalement non pertinents.

Notre information ou peuvent contenir des attributs totalement non pertinents. De tels attributs sont supprimés à cette étape. Diverses techniques sont utilisées à ce stade, par exemple la sélection des caractéristiques, l'extraction des caractéristiques, la discrétisation des attributs, etc. L'objectif de base est de transformer/réduire les données pour améliorer les performances lors des étapes ultérieures.

I.2.SÉLECTION D'ATTRIBUTS :

La sélection d'attributs en Data Mining réduit l'information initiale, en identifiant les attributs redondants ou ceux dont la valeur ne contribue pas à la phase d'exploitation des données. Le but à travers cela c'est de faciliter le travail de notre algorithme d'apprentissage en essayant de gagner du temps de traitement tout en améliorant les performances et les résultats de celui-ci, sans que cela ne puisse détériorer les performances de l'algorithme.

Il existe plusieurs méthodes de sélection d'attributs comme Rough **set**, **filter**, **wrapper**, **linear**, ...etc.

I.3.ROUGH SET :

La sélection des caractéristiques vise à déterminer un sous-ensemble minimal de caractéristiques à partir d'un domaine problématique tout en conservant une précision suffisamment élevée dans la représentation des caractéristiques d'origine. Rough set theory (RST) est une méthode très utilisée dans la sélection d'attributs. RST permet la découverte des dépendances de données et la réduction du nombre d'attributs contenus dans un ensemble de données en utilisant les données seules, il n'a pas besoin d'informations supplémentaire. Rough Set n'a pas mal d'extension ou de version parmi ces versions en trouve FRST « **Fuzzy Rough Set** », ainsi y'a aussi différentes heuristiques générées par RST, l'une d'elle s'est « **QuickReduct Algorrtthm** ».

RST identifie les dépendances partielles ou totales dans les données, élimine les données redondantes, donne une approche des valeurs nulles, des données manquantes, des données dynamiques et autres.

Decision Systems :

Les systèmes de décision sont une forme spéciale de système d'information ayant un attribut de décision aussi appelé classe d'objet. Chaque objet appartient à une classe spécifique. La valeur de la classe dépend d'autres attributs appelés attributs conditionnels. Formellement :

$$\alpha = (U, C \cup \{D\})$$

Où : C = ensemble d'attributs conditionnels, D = attribut de décision (ou classe)

Indiscernability : Un système de décision représente toutes les connaissances sur un modèle. Cette table peut être inutilement grande de deux façons : il peut y avoir des objets identiques ou indiscernables ayant plus d'une occurrence et il peut y avoir des attributs superflus.

Relation d'équivalence : Une relation binaire est appelée relation d'équivalence si elle est réflexive c'est-à-dire qu'un objet est en relation avec lui-même xRx , symétrique c'est-à-dire que si xRy , alors yRx et transitive c'est-à-dire que si xRy et yRz alors xRz . La classe d'équivalence d'un élément est constituée de tous les objets tels que xRy .

Approximations :

1. **Upper approximations** : Les approximations supérieures sont un type d'approximation utilisé en analyse mathématique pour décrire un ensemble, une fonction ou un autre objet mathématique dont on sait qu'il est inférieur ou égal à l'objet faisant l'objet de l'approximation. Les approximations supérieures sont souvent utilisées conjointement avec les approximations inférieures pour fournir des limites à la valeur de l'objet approximé. Les approximations supérieures peuvent être construites de différentes manières, en fonction de la nature de l'objet à approximer et du niveau de précision souhaité pour l'approximation. Parmi les méthodes courantes de construction d'approximations supérieures, citons l'utilisation de upper bounds, envelopes et de convergent sequences.
2. **Lower approximations** : Les approximations inférieures sont un type d'approximation utilisé en analyse mathématique pour décrire un ensemble, une fonction ou un autre objet mathématique dont on sait qu'il est supérieur ou égal à l'objet faisant l'objet de l'approximation. Les approximations inférieures sont souvent utilisées conjointement

avec les approximations supérieures pour fournir des limites à la valeur de l'objet approximé. Les approximations inférieures peuvent être construites de différentes manières, en fonction de la nature de l'objet à approximer et du niveau de précision souhaité pour l'approximation.

Les approximations inférieure et supérieure sont définies comme suit respectivement :

$$X : \underline{B}X = \{x \llbracket x \rrbracket_B \subseteq X\}$$

$$X : \overline{B}X = \{x \llbracket x \rrbracket_B \cap X \neq \emptyset\}$$

Boundary region : est la suivante : $BInsurance - BInsurance$

Dans Rough set, boundary region est un ensemble d'objets qui sont indéterminés par rapport à un Rough set donné. Les Rough set sont utilisés pour décrire des informations imprécises ou incomplètes, et la boundary region est constituée d'objets pour lesquels il n'est pas possible de déterminer avec certitude s'ils appartiennent ou non au Rough set. La boundary region est l'ensemble des objets qui se trouvent à la "frontière" entre le rough set et son complément. En général, la boundary region est un sous-ensemble de la région indéterminée, qui se compose de tous les objets qui ne sont pas définitivement dans ou hors du Rough set.

Positive region : Dans RST, la région positive d'un rough set est l'ensemble des objets qui sont définitivement connus pour appartenir au rough set. La région positive est également connue comme la région certaine ou le noyau du rough set. Il s'agit de l'ensemble des objets pour lesquels il n'y a aucun doute ou incertitude quant à leur appartenance à l'ensemble. La région positive est un sous-ensemble du rough set, et elle est complémentaire de la région négative, qui est constituée d'objets dont on sait avec certitude qu'ils n'appartiennent pas au rough set. La région positive peut être utilisée pour identifier les éléments les plus importants ou les plus pertinents du rough set, et elle est souvent utilisée pour prendre des décisions ou tirer des conclusions sur la base des informations contenues dans le rough set.

Dependency : La dépendance définit comment la valeur d'un attribut détermine de façon unique la valeur d'un autre attribut. Un attribut "D" dépend d'un autre attribut "C"

par le degré "K" calculé par : $k = \gamma(C, D) = \frac{|POS_C(D)|}{|U|}$ et

$$POS_C(D) = \bigcup_{X \in U/D} \underline{C}(X)$$

Si $K = 1$, D dépend totalement de C, pour $0 < K < 1$, D dépend partiellement de C et pour $K = 0$, D ne dépend pas de C. Il est clair que si $K=1$ c'est-à-dire que D dépend totalement de C alors $IND(C) \subseteq IND(D)$.

I.4. IMPLEMENTATION DE ROUGH SET :

Choix du petit dataset :

U	COAL	SULFER	PHOSPHORUS	CRACKS
X1	HIGH	HIGH	VERY-GOOD	YES
X2	HIGH	LOW	GOOD	YES
X3	HIGH	LOW	GOOD	NO
X4	LOW	HIGH	NORMAL	NO
X5	LOW	HIGH	GOOD	YES
X6	LOW	HIGH	VERY-GOOD	YES

Idescernability :

$$IND(COAL) = \{\{X1, X2, X3\}, \{X4, X5, X6\}\}$$

$$IND(SULFER) = \{\{X1\}, \{X2, X3\}, \{X4, X5, X6\}\}$$

$$IND(PHOSPHORUS) = \{\{X1, X6\}, \{X2, X3, X5\}, \{X4\}\}$$

$$IND(COAL, SULFER) = \{\{X1\}, \{X2, X3\}, \{X4, X5, X6\}\}$$

$IND(COAL, PHOSPHORUS) = \{\{X1\}, \{X2, X3\}, \{X4\}, \{X5\}, \{X6\}\}$

$IND(SULFUR, PHOSPHORUS) = \{\{X1, X6\}, \{X2, X3\}, \{X4\}, \{X5\}\}$

$IND(COAL, SULFUR, PHOSPHORUS) = \{\{X1\}, \{X2, X3\}, \{X4\}, \{X5\}, \{X6\}\}$

Set approximations :

➤ **Lower et Upper approximations :**

$X1 = \{X1, X2, X5, X6\}; X1 = \{X: CRACKS(X) = Yes\}$

Set A = IND (COAL, SULFUR, PHOSPHORUS)

Lower: $\underline{A}X1(X=Yes) = \{\{X1\}, \{X5\}, \{X6\}\} = \{X1, X5, X6\}$

Upper: $\check{A}X1(X=Yes) = \{\{X1\}, \{X2, X3\}, \{X5\}, \{X6\}\} = \{X1, X2, X3, X5, X6\}$

$X2 = \{X3, X4\}; X2 = \{X: CRACKS(X) = NO\}$

Set A = IND (COAL, SULFUR, PHOSPHORUS)

Lower: $\underline{A}X2(X=NO) = \{\{X4\}\} = \{X4\}$

Upper: $\check{A}X2(X=NO) = \{\{X2, X3\}, \{X4\}\} = \{X2, X3, X4\}$

Boundary region :

$BNA(X1) = \{X2, X3\} \quad || \quad BNA(X2) = \{X2, X3\}$

Outside region: $U - \check{A}x_i$ (Upper Approximation)

OUT- $\check{A}X2 = U - \check{A}X1 = \{X4\}$

OUT- $\check{A}X1 = U - \check{A}X1 = \{X1, X5, X6\}$

Positive region: Lower: $\underline{A}X1 \cup$ Lower: $\underline{A}X2, POS = \{X1, X4, X5, X6\}$

Dependency: Positive region / U $K = \gamma(A, \{d\}) = 6/6$

Calculer les reducts : pour calculer tous les **reducts** possible de notre dataset, on adéjà calculer la **positive region** de (COAL, SULFER, PHOSPHORUS), dans ce qui suit en va faire des combinaisons entre les attributs conditionnelle de notre dataset donc les combinaisons possibles sont : $\{\{COAL\}, \{SULFER\}, \{PHOSPHORUS\}, \{COAL, SULFER\}, \{SULFER, PHOSPHORUS\}, \{COAL, PHOSPHORUS\}\}$ et on va calculer pour chaque combinaison sa positive region.

Si la postive région d'une combinaison égale a la positive région d'une combinaison alors cette dernière est considérée comme un reduct.

Calcule des positive région :

$$\text{POS (COAL, SULFER, PHOSPHORUS)} = \{X1, X4, X5, X6\}$$

$$\text{POS (COAL)} = \{\emptyset\}$$

$$\text{POS (SULFER)} = \{\emptyset\}$$

$$\text{POS (PHOSPHORUS)} = \{X1, X4, X6\}$$

$$\text{POS (COAL, SULFER)} = \{X1\}$$

$$\text{POS (COAL, PHOSPHORUS)} = \{X1, X4, X5, X6\} \rightarrow \text{c'est un reduct}$$

$$\text{POS (SULFER, PHOSPHORUS)} = \{X1, X4, X5, X6\} \rightarrow \text{c'est un reduct}$$

On a deux reducts car avec les deux combinaisons on a eu le même ensemble avec **POS (COAL, SULFER, PHOSPHORUS)**.

Notre ensemble de reducts **R** = $\{\{SULFER, PHOSPHORUS\}, \{COAL, PHOSPHORUS\}\}$.

On peut Constater que le reduct core de notre exemple est PHOSPHORUS car il figure dans l'ensemble de nos reducts.

I.5. PRESENTATION DES ALGORITHMES DE ROUGH SET :

QuickReduct Algorithm : Tente de calculer les réductions sans générer de manière exhaustive tous les sous-ensembles possibles.

ReverseReduct Algorithm : L'élimination en arrière est utilisée sans générer de manière exhaustive toutes les combinaisons possibles. La dépendance est calculée à l'aide de l'approche conventionnelle basée sur positive region.

Based on genetic algorithm : Il est basé sur le hasard, de sorte que la procédure peut trouver des réductions en quelques tentatives. Utilise une mesure de dépendance conventionnelle basée sur la positive region.

Fish Swarm Algorithm : Tente de trouver des réductions d'ensembles bruts en utilisant la logique d'essaimage où les essaims peuvent découvrir la meilleure combinaison de caractéristiques.

Using filter-based approach :

Les approches basées sur des filtres sont un type de méthode utilisée pour identifier des modèles ou des caractéristiques dans un ensemble de données qui sont pertinents pour un problème ou une tâche particulière. Ces approches impliquent généralement l'utilisation d'une sorte de filtre ou de critères de sélection pour sélectionner un sous-ensemble de données dans un ensemble de données plus large, qui est ensuite utilisé pour entraîner un modèle ou faire des prédictions.

I.5. FUZZY ROUGH SET:

Dans rough set theory, fuzzy rough set est une généralisation des ensembles bruts dans laquelle les valeurs d'appartenance des éléments d'un ensemble ne sont pas binaires (c'est-à-dire qu'elles ne sont pas simplement "dedans" ou "dehors"), mais plutôt des nombres réels entre 0 et 1. Cela permet une approche plus nuancée pour traiter l'incertitude et l'imprécision des données.

Utilisés dans une variété d'applications, notamment l'exploration de données, la reconnaissance des formes et la prise de décision. Ils sont particulièrement utiles dans les situations où les limites entre différentes classes ou catégories ne sont pas clairement définies, et où il peut y avoir un chevauchement entre différentes classes.

Dans Fuzzy Rough Set l'appartenance d'un élément à un ensemble est déterminée par une fonction d'appartenance, qui attribue une valeur d'appartenance à chaque élément en fonction

de ses caractéristiques ou attributs. Cette valeur d'appartenance peut être utilisée pour déterminer le degré d'appartenance d'un élément à un ensemble particulier, et peut être utilisée pour prendre des décisions ou classer des éléments en fonction de cette appartenance.

CONCLUSION :

La quantité de données à traiter augmente considérablement de jour en jour. L'augmentation de la taille des données n'est pas seulement due à l'augmentation du nombre d'enregistrements, mais aussi au nombre important d'attributs ajoutés à l'espace. Ce phénomène conduit à un dilemme appelé curse of dimensionality, c'est-à-dire à des ensembles de données comportant un nombre exponentiel d'attributs. L'approche idéale consiste à réduire le nombre de dimensions de sorte que l'ensemble réduit résultant contienne la même information que celle présente dans l'ensemble des attributs. Cette recherche aborde et tente de résoudre le dilemme de curse of dimensionality en fournissant un ensemble réduit de calcul, de sorte que les dimensions puissent être réduites avec 0% de perte d'information.

CHAPITRE II :

EXPLICATION ET IMPLEMENTATION DE QUICK REDUCT

INTRODUCTION :

Dans ce chapitre, on va présenter et expliquer à travers un exemple l'algorithme de réduction de dimensionnalité Quick Reduct. L'objectif principal de Quick Reduct est de réduire la taille des données. Le Quick Reduct peut réduire la dimensionnalité en utilisant les informations contenues dans l'ensemble de données, tout en préservant également la signification des caractéristiques (c'est-à-dire qu'il préserve la sémantique).

DEFINITION ET EXPLICATION QUIK REDUCT :

Le quick reduct est un algorithme de réduction de base de données en apprentissage automatique. Il s'agit d'une technique qui permet de réduire la taille d'une base de données en enlevant les exemples redondants ou inutiles, tout en préservant l'exactitude de la prédiction du modèle. Cela peut être utile dans les situations où la base de données est trop grande pour être traitée efficacement par un algorithme d'apprentissage automatique, ou lorsque l'on souhaite simplifier le modèle pour le rendre plus facile à comprendre et à interpréter.

Le quick reduct utilise une heuristique de sélection de sous-ensemble pour choisir les exemples à conserver dans la base de données réduite. Il sélectionne d'abord un exemple de départ au hasard, puis ajoute les exemples qui sont les plus proches de cet exemple initial, jusqu'à ce que la précision du modèle atteigne un niveau satisfaisant.

Les étapes de Quick Reduct :

1. Préparation des données : assurer que les données sont propres et prêtes à être utilisées pour l'apprentissage automatique. Cela peut inclure l'élimination de valeurs manquantes, la normalisation des données, etc.
2. Définir un modèle de prédiction : choisir un modèle de prédiction approprié en fonction du jeu de données et de l'objectif de prédiction.
3. Sélection d'un sous-ensemble de caractéristiques : utilisation de QuickReduct pour sélectionner un sous-ensemble de caractéristiques à partir du jeu de données original. QuickReduct utilise une heuristique pour évaluer l'importance relative de chaque caractéristique et sélectionne un sous-ensemble optimal en fonction de cette évaluation.

4. Entraînement et évaluation du modèle : en entraîne le modèle de prédiction en utilisant le sous-ensemble de caractéristiques sélectionné et puis évaluation de sa performance en utilisant des métriques appropriées, telles que l'erreur de prédiction ou l'aire sous la courbe ROC.

Implémentation de Quick reduct sur un petit jeu de données :

Choix du petit dataset :

U	State	Qualification	Branch	Job
X1	0	0	3	1
X2	1	1	3	0
X3	0	2	1	1
X4	1	2	2	1
X5	2	3	1	0
X6	2	3	0	0
X7	3	1	0	1

Etape 1 :

On commence par ensemble vide $R = \emptyset$

$$\gamma \{ \text{State, Qualification, Branch} \}(\text{Job}) = 7/7$$

La première étape du calcul de la mesure de dépendance basée sur la région positive consiste à calculer les classes d'équivalence en utilisant l'attribut de décision (" Job " dans notre cas). La structure des classes d'équivalence spécifie tous les objets indiscernables, c'est-à-dire les objets qui, par rapport à des attributs donnés, ne peuvent pas être distingués.

Dépendance de State :

0 : {x1, x3} ; 1 : {x2, x4} ; 2 : {x5, x6} ; 3 : {x7}

Puis on va voir la valeur des Xi dans Job

Pour notre cas on va toujours utiliser **lower approximation** c'est-à-dire si les deux éléments d'un ensemble sont à 1 ou à 0 alors on va les prendre sinon les en prend pas.

Inf 1 : { x1, x3, x7} ; Inf 0 : { x5, x6}

Puis en fait l'union entre les 2 inf c'est-à-dire (**Inf 1 U Inf 0**), ce qui va nous donner :

$$\gamma\{\text{State}\}(\text{Job}) = 5/7$$

Dépendance de Qualification :

0 : {x1} ; 1 : {x2,x7} ; 2 : {x3,x4} ; 3 : {x5, x6}

Puis on va voir la valeur des Xi dans Job

Pour notre cas on va toujours utiliser **lower approximation** c'est-à-dire si les deux éléments d'un ensemble sont à 1 ou à 0 alors on va les prendre sinon les en prend pas.

Inf 1 : { x1, x3, x4} ; Inf 0 : { x5, x6}

Puis en fait l'union entre les 2 inf c'est-à-dire (**Inf 1 U Inf 0**), ce qui va nous donner :

$$\gamma\{\text{Qualification}\}(\text{Job}) = 5/7$$

Dépendance de Branch :

0 : {x6, x7} ; 1 : {x3,x5} ; 2 : {x4} ; 3 : {x1, x2}

Puis on va voir la valeur des Xi dans Job

Pour notre cas on va toujours utiliser **lower approximation** c'est-à-dire si les deux éléments d'un ensemble sont à 1 ou à 0 alors on va les prendre sinon les en prend pas.

Inf 1 : {x4} ; Inf 0 : {}

Puis en fait l'union entre les 2 inf c'est-à-dire (**Inf 1 U Inf 0**), ce qui va nous donner :

$$\gamma\{\mathbf{Branch}\}(\mathbf{Job}) = 1/7$$

Comme {State} génère le plus haut degré de dépendance, il est ajouté au Reduct R= {State}

Etape 2 : De nouveau, nous évaluons les valeurs de dépendance suivantes :

Dépendance de (State, Qualification) :

00 : {x1} ; 11 : {x2} ; 02 : {x3} ; 12 : {x4} ; 23 : {x5, x6} ; 31{x7}

Puis on va voir la valeur des Xi dans Job

Pour notre cas on va toujours utiliser **lower approximation** c'est-à-dire si les deux éléments d'un ensemble sont à 1 ou à 0 alors on va les prendre sinon les en prend pas.

Inf 1 : {x1, x3, x4, x7} ; Inf 0 : {x2, x5, x7}

Puis en fait l'union entre les 2 inf c'est-à-dire (**Inf 1 U Inf 0**), ce qui va nous donner :

$$\gamma\{\mathbf{State, Qualification}\}(\mathbf{Job}) = 7/7$$

Arrêt des itérations puisque la dépendance générée est égale à la dépendance de l'ensemble des données.

R= {State, Qualification}

I.3. DIFFERENCE ENTRE QUIK REDUCT ET FUZZY ROUGH SET:

QuickReduct est un algorithme utilisé dans la théorie des ensembles bruts pour calculer un ensemble minimal d'attributs suffisant pour prendre des décisions fiables, tandis que l'ensemble brut flou est une généralisation des ensembles bruts qui permet une approche plus nuancée pour traiter l'incertitude et l'imprécision des données.

CONCLUSION :

QuickReduct fournit de nombreux concepts pour analyser en profondeur les ensembles de données et trouver les caractéristiques non pertinentes et redondantes. Étant donné un ensemble de données avec des valeurs d'attributs discrétisées, il est possible de trouver un sous-ensemble des attributs originaux en utilisant le QuickReduct qui sont les plus informatifs (tous les autres attributs peuvent être supprimés de l'ensemble de données avec un minimum d'information).

Cette méthode ne nécessite aucune intervention humaine. Plus important encore, elle conserve également la sémantique des données, ce qui rend les modèles résultants plus transparents à l'examen humain.

CHAPITRE III :

IMPLEMENTATION EN PYTHON

INTRODUCTION :

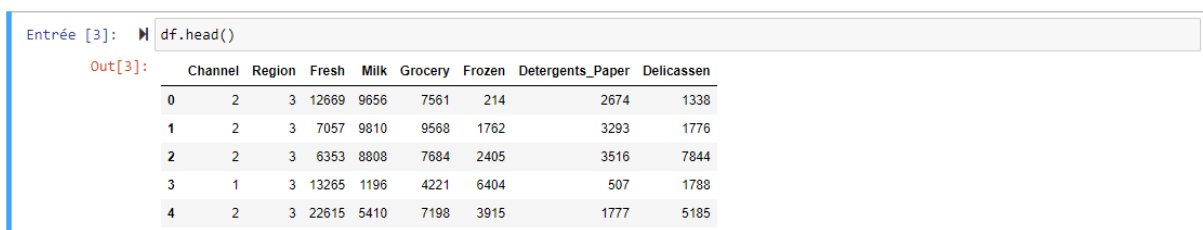
Dans les chapitres précédents, nous avons discuté de diverses techniques de sélection de caractéristiques, telles que la méthode de filtrage, la méthode de wrappe et la méthode de Rough Set Theory. Dans ce chapitre, nous allons mettre en œuvre certaines de ces techniques en utilisant Python.

Nous allons utiliser plusieurs bibliothèques Python couramment utilisées pour la science des données et l'apprentissage automatique, telles que pandas, numpy, scikit-learn et matplotlib, pour implémenter les techniques de sélection de caractéristiques que nous avons discutées. Nous allons également utiliser un ensemble de données réel pour illustrer comment ces techniques peuvent être appliquées à des problèmes réels.

Enfin, nous allons évaluer les performances des différents algorithmes de sélection de caractéristiques que nous avons implémentés en utilisant des mesures telles que la précision, le rappel et la F-mesure. Nous allons également comparer les résultats avec ceux obtenus en utilisant toutes les fonctionnalités disponibles pour voir si la sélection de caractéristiques améliore la performance du modèle

DATA UNDERSTANDING :

- Wholesale customers data, il fait référence aux clients d'un grossiste répartiteur. Il comprend les dépenses annuelles en unités monétaires (mu) sur diverses catégories de produits, il contient 8 attributs (FRAIS, LAIT, ÉPICERIE, SURGELES, DETERGENTS_PAPIER, CHARCUTERIE, CANAL, RÉGION).
- La thématique qu'on peut résoudre avec ce dataset c'est une classification des données selon par exemple la région, ou bien on peut utiliser l'attribut Channel (Canal) pour construire deux classes **Customers channel** et **Retail channel**.



Entrée [3]: `df.head()`

Out[3]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185


```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440 entries, 0 to 439
Data columns (total 8 columns):
#   Column              Non-Null Count  Dtype
---  ---
0   Channel              440 non-null    int64
1   Region               440 non-null    int64
2   Fresh                440 non-null    int64
3   Milk                 440 non-null    int64
4   Grocery              440 non-null    int64
5   Frozen               440 non-null    int64
6   Detergents_Paper     440 non-null    int64
7   Delicassen           440 non-null    int64
dtypes: int64(8)
memory usage: 27.6 KB
```

Taille du DataFrame :

DataFrame size

```
print(f'Dimensions de la dataframe est : {df.shape}')
```

```
Dimensions de la dataframe est : (440, 8)
```

- ce dataset possède 8 **features** et 440 **tuples**

Type d'attributs : ce dataset comporte que des features numérique et ne possède aucun feature catégorique.

Features type

```
pd.DataFrame(df.dtypes,
              columns = ["Type"])
```

```
8]:
```

	Type
Channel	int64
Region	int64
Fresh	int64
Milk	int64
Grocery	int64
Frozen	int64
Detergents_Paper	int64
Delicassen	int64

Quelques statistiques sur les données :

```
df.describe()
```

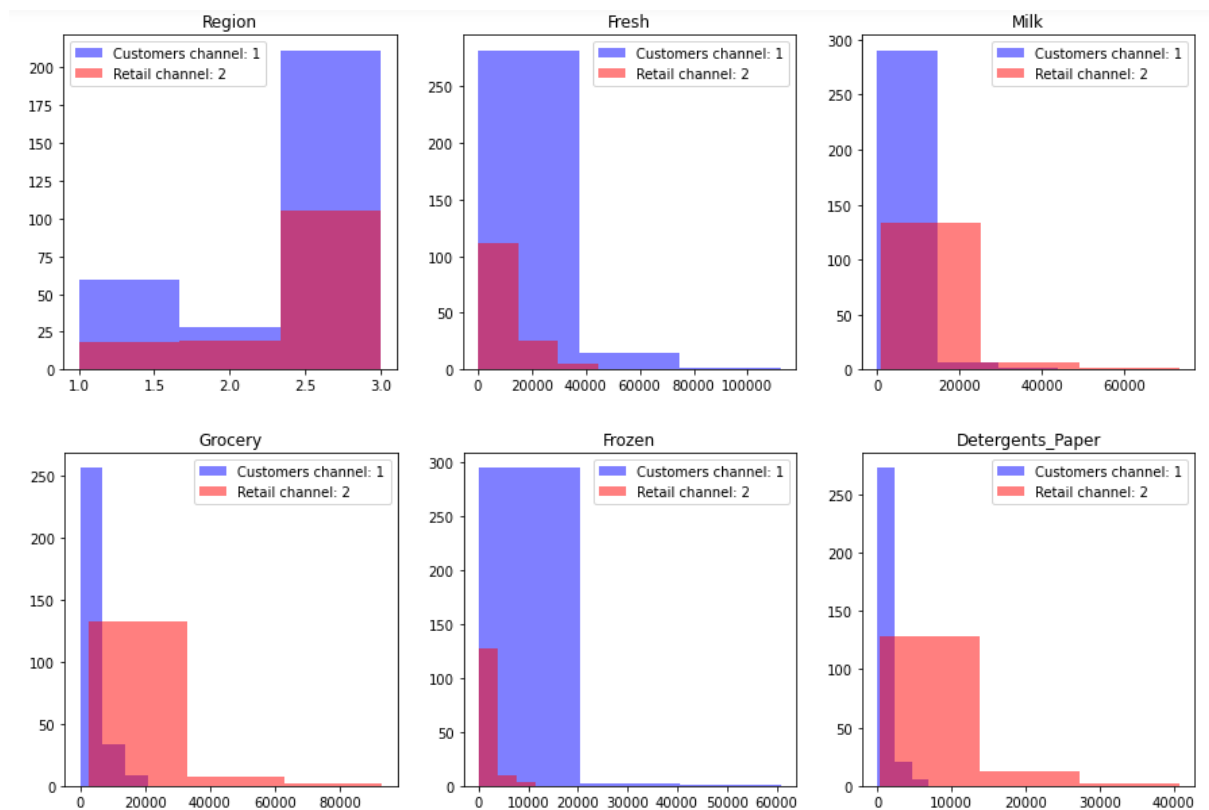
	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
count	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000
mean	1.322727	2.543182	12000.297727	5796.265909	7951.277273	3071.931818	2881.493182	1524.870455
std	0.468052	0.774272	12647.328865	7380.377175	9503.162829	4854.673333	4767.854448	2820.105937
min	1.000000	1.000000	3.000000	55.000000	3.000000	25.000000	3.000000	3.000000
25%	1.000000	2.000000	3127.750000	1533.000000	2153.000000	742.250000	256.750000	408.250000
50%	1.000000	3.000000	8504.000000	3627.000000	4755.500000	1526.000000	816.500000	965.500000
75%	2.000000	3.000000	16933.750000	7190.250000	10655.750000	3554.250000	3922.000000	1820.250000
max	2.000000	3.000000	112151.000000	73498.000000	92780.000000	60869.000000	40827.000000	47943.000000

Data Visualisation :

Data Visualisation

comprendre la distribution des données de différentes caractéristiques en utilisant les histogrammes.

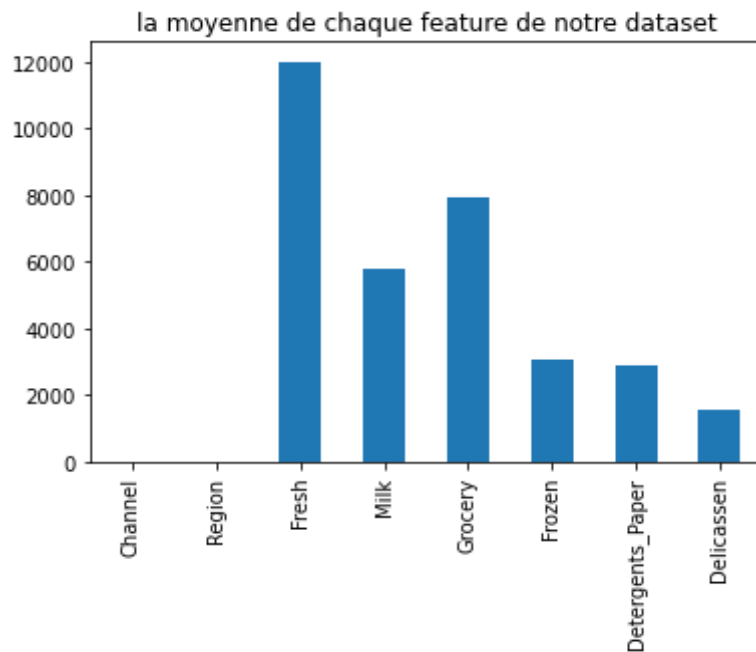
```
df1 = pd.DataFrame(df, columns=['Region', 'Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Channel'])
# separate the data into two dataframes based on the label
df_0 = df[df['Channel']==1]
df_1 = df[df['Channel']==2]
# create histograms for each feature for each label
fig, axs = plt.subplots(2, 3, figsize=(15,10))
axs = axs.ravel()
for i, feature in enumerate(['Region', 'Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper']):
    axs[i].hist(df_0[feature], bins=3, alpha=0.5, label='Customers channel: 1', color='blue')
    axs[i].hist(df_1[feature], bins=3, alpha=0.5, label='Retail channel: 2', color='red')
    axs[i].set_title(feature)
    axs[i].legend()
plt.show()
```



Moyenne de chaque attribut dans notre Dataset :

```
means = df
means.mean(skipna=True).plot(kind='bar', title="la moyenne de chaque feature de notre dataset")
means.mean()
```

```
Channel          1.322727
Region           2.543182
Fresh          12000.297727
Milk             5796.265909
Grocery          7951.277273
Frozen           3071.931818
Detergents_Paper 2881.493182
Delicassen       1524.870455
dtype: float64
```

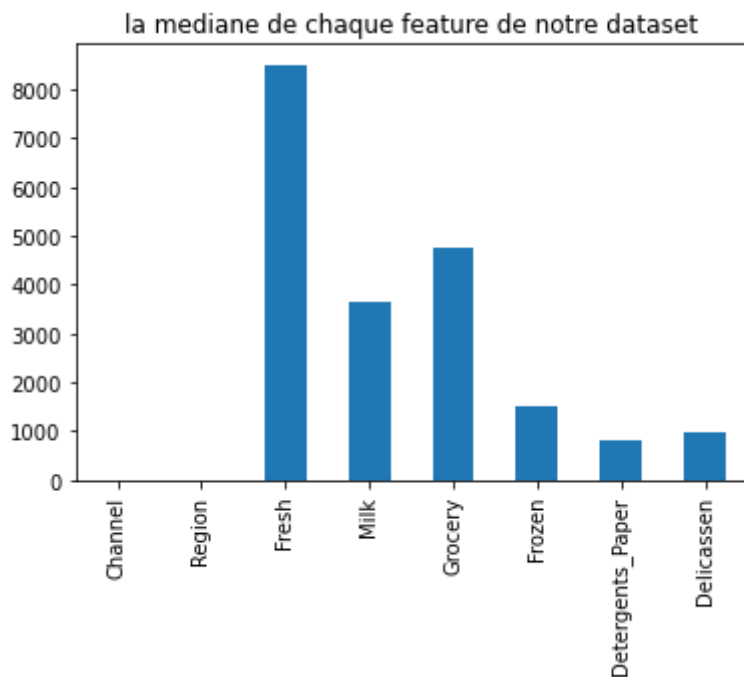


Mediane de chaque attribut dans notre Dataset :

```
means.median().plot(kind='bar',title="la mediane de chaque feature de notre dataset")

#affichage des medianes des colonnes à part
means.median()
```

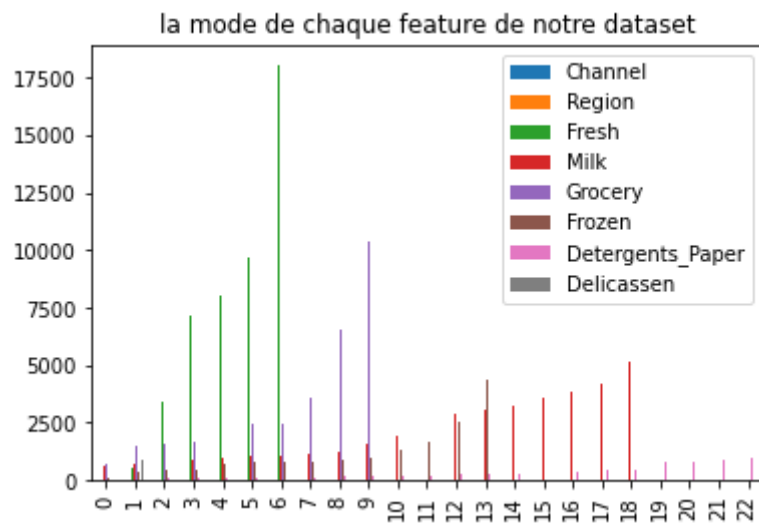
```
Channel          1.0
Region           3.0
Fresh           8504.0
Milk             3627.0
Grocery          4755.5
Frozen           1526.0
Detergents_Paper 816.5
Delicassen       965.5
dtype: float64
```



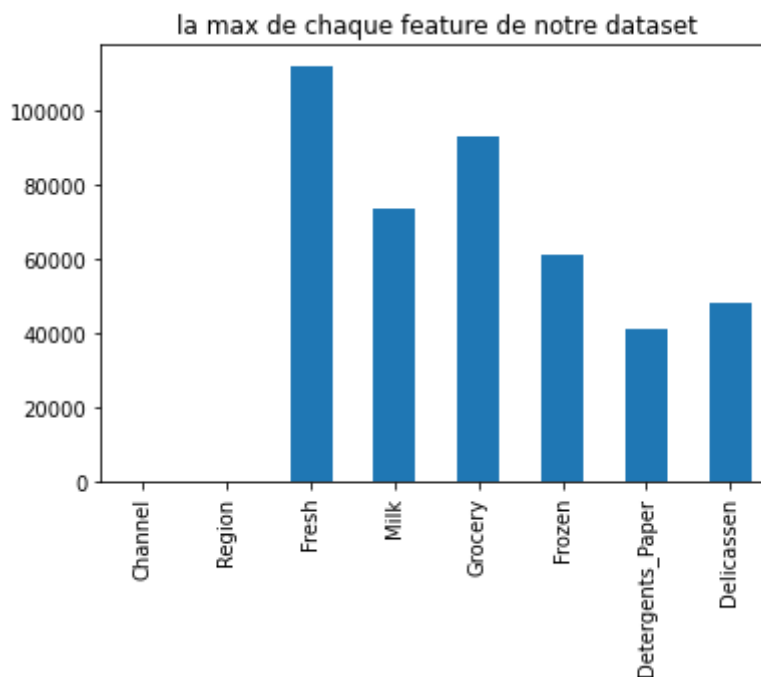
Le mode de chaque feature dans notre Dataset :

```
means.mode(dropna=True).plot(kind='bar',title="la mode de chaque feature de notre dataset")
```

```
<AxesSubplot:title={'center':'la mode de chaque feature de notre dataset'}>
```



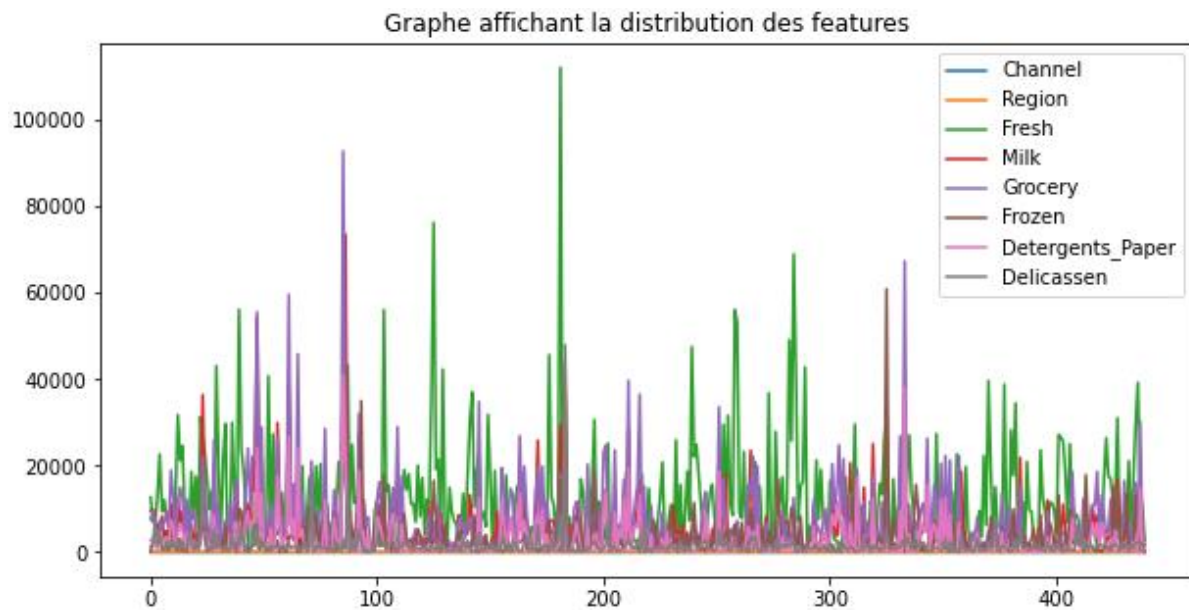
Le max de chaque feature dans notre Dataset :



Graphe affichant la distribution des features :

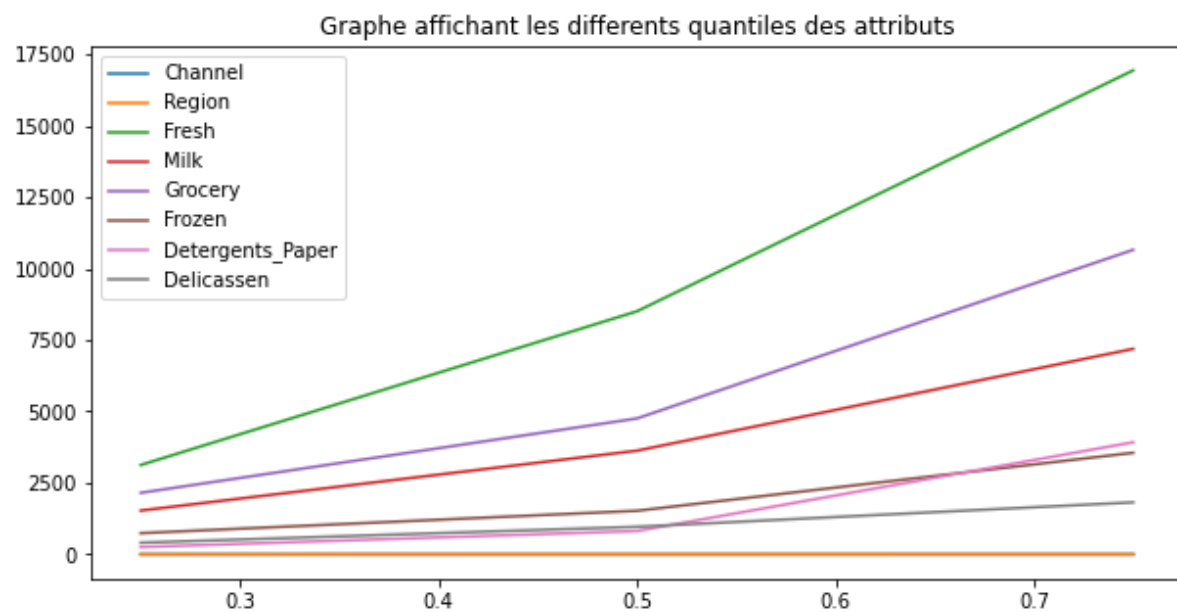
```
df.plot(figsize=(10,5),title="Graphe affichant la distribution des features")
```

```
<AxesSubplot:title={'center':'Graphe affichant la distribution des features'}>
```



Affichage d'un graphe qui représente la quantité de chaque feature :

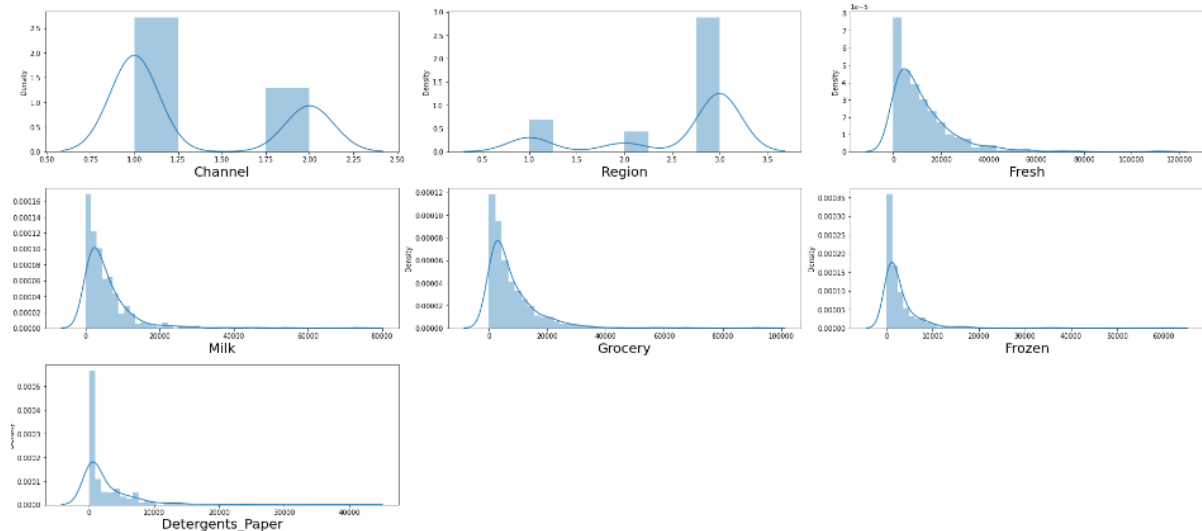
```
print("Les quantités du dataset")
x_ = means.quantile([0.25,0.5,0.75])
x_.plot(figsize=(10,5),title="Graphe affichant les differents quantiles des attributs")
```



Histogrammes des Densités :

```
plt.figure(figsize=(25,25), facecolor='white')
plotnumber=1

for column in df.columns[0:len(df.columns)-1]:
    if plotnumber<=21:
        ax=plt.subplot(7,3,plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column,fontsize=20)
        plotnumber+=1
plt.tight_layout()
plt.show()
```

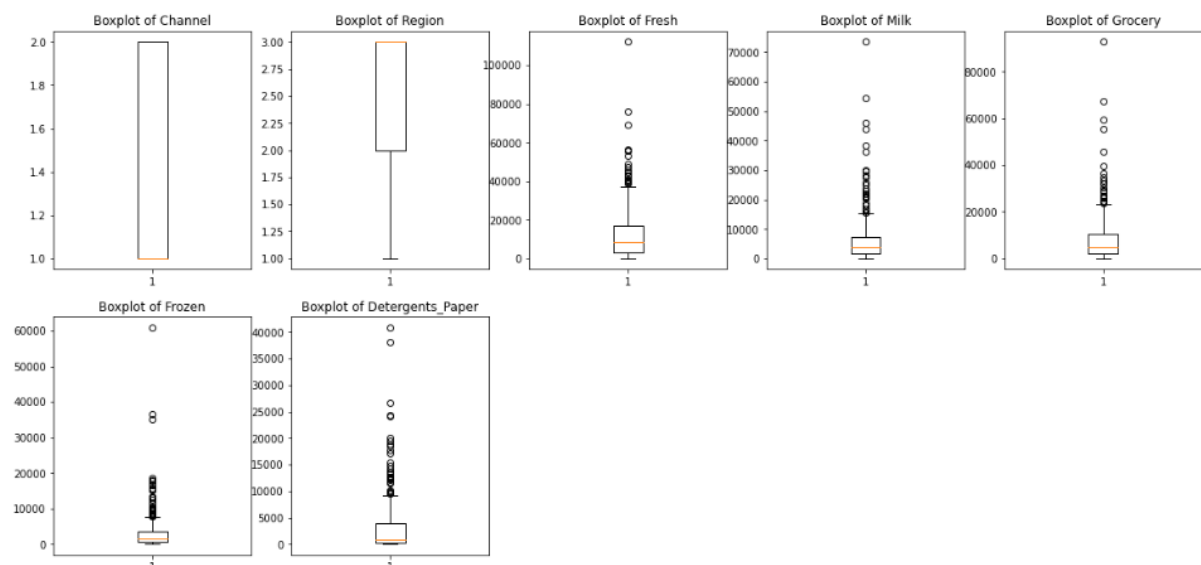


Visualisation de ces outliers :

```
plt.figure(figsize=(20,40))

for i, column in enumerate(df.columns[0:len(df.columns)-1]):
    plt.subplot(len(df.columns), 5, i+1)
    plt.boxplot(df[column])
    plt.title("Boxplot of {}".format(column))

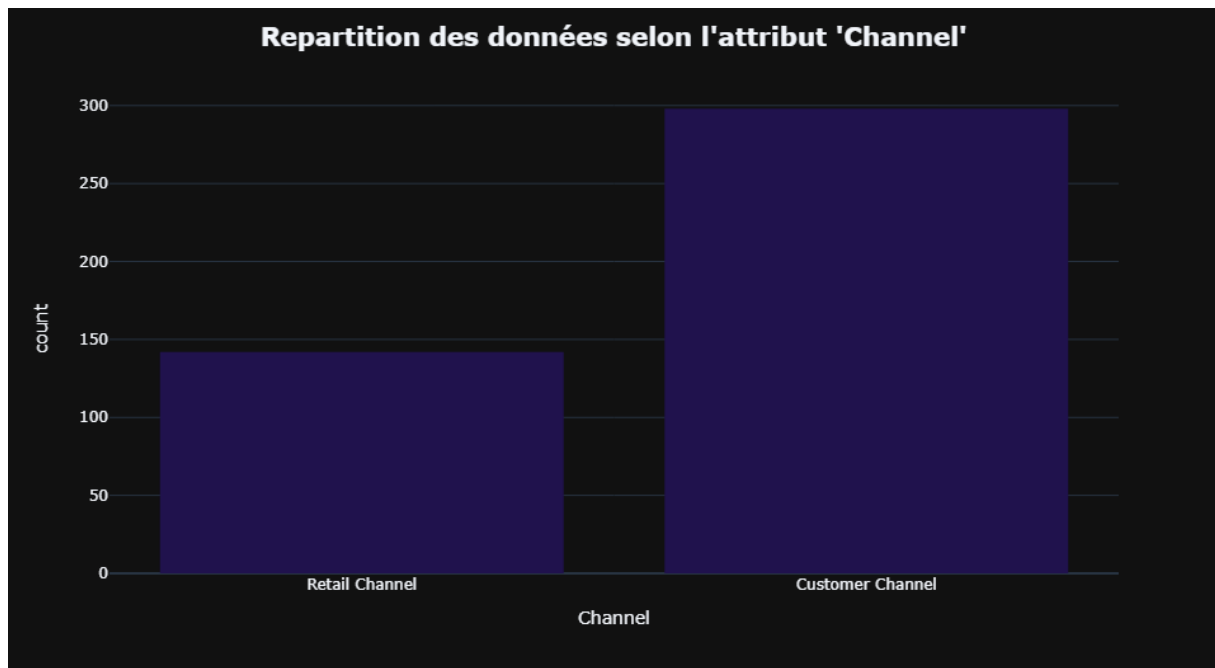
plt.show()
```



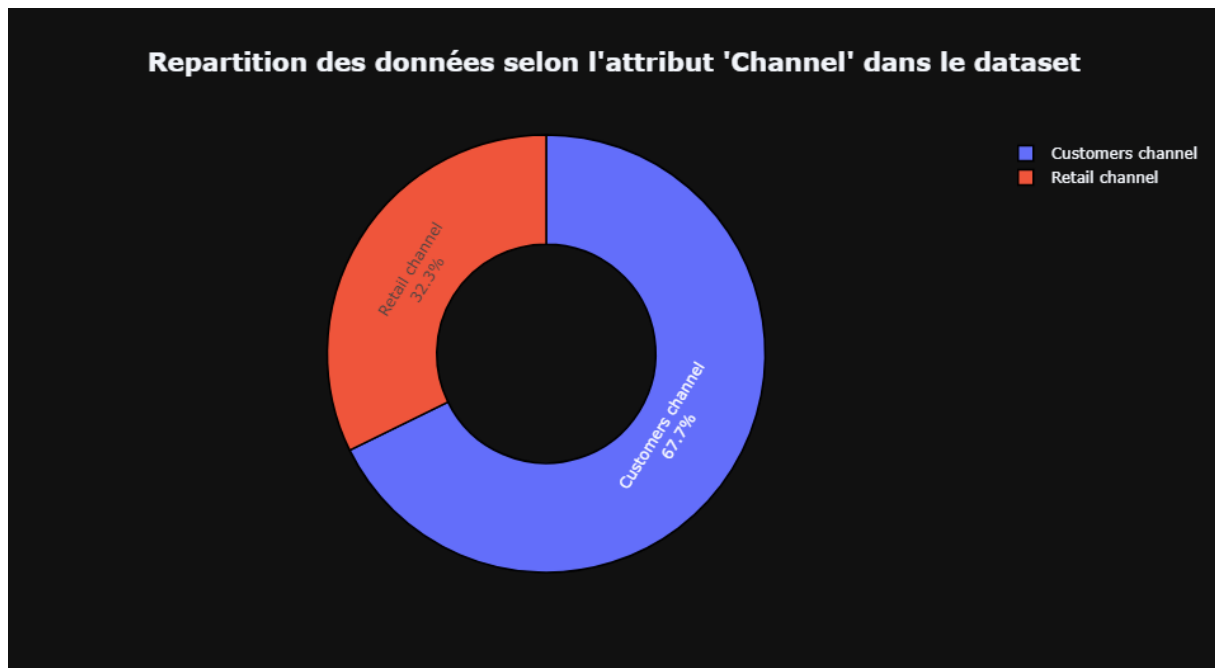
Répartition des données selon l'attribut Channel :

```
import plotly.express as px
```

```
fig = px.histogram(df, x="Channel",  
                  template='plotly_dark',  
                  color_discrete_sequence = ["#20124d"])  
  
fig.update_layout(title = "<b>Répartition des données selon l'attribut 'Channel'</b>",  
                  title_x = 0.5,  
                  title_font = dict(size = 20),  
                  uniformtext_minsize = 15)  
  
fig.show()
```



```
df["Channel"] = np.where(df["Channel"] == 1, "Customers channel", "Retail channel")  
fig = px.pie(df, names = "Channel",  
            title = "<b>Répartition des données selon l'attribut 'Channel' dans le dataset</b>",  
            hole = 0.5, template = "plotly_dark")  
  
fig.update_traces(textposition='inside',  
                  textinfo='percent+label',  
                  marker=dict(line=dict(color='#000000', width = 1.5)))  
  
fig.update_layout(title_x = 0.5,  
                  title_font = dict(size = 20),  
                  uniformtext_minsize = 15)  
  
fig.show()
```



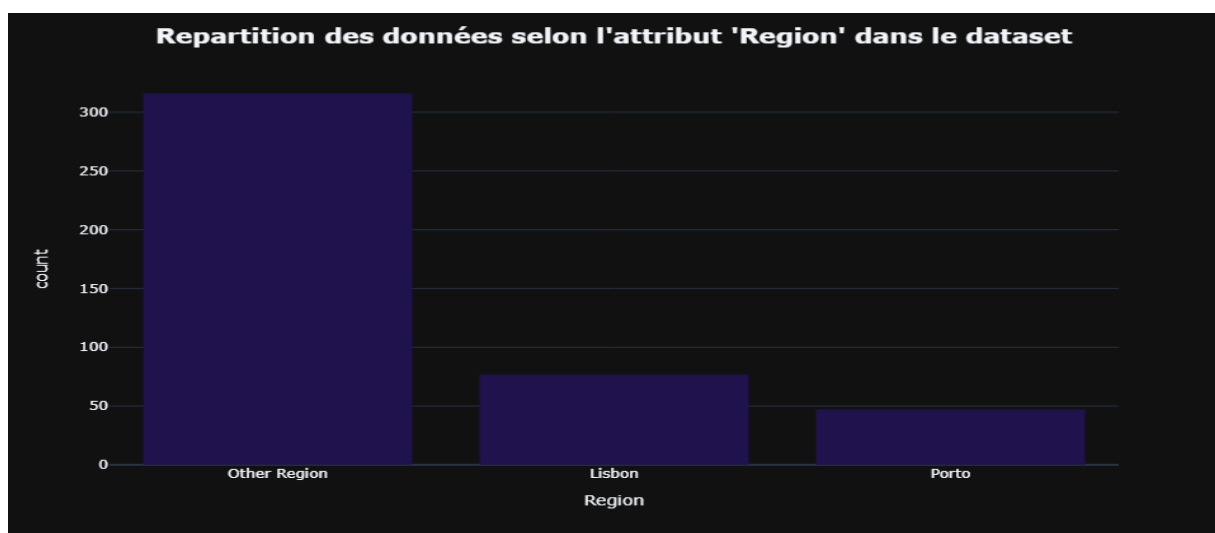
- D'après les deux figures au-dessus, on remarque que la classe majoritaire dans notre dataset c'est la classe 1 qui fait référence au produit livrés par le canal Horeca (Hôtel/Restaurant/Café).
- A noter que Horeca signifie "Hôtels, Restaurants et Cafétérias". C'est un terme utilisé pour décrire les entreprises dans le secteur de la restauration commerciale.

Répartition des données selon l'attribut 'Region' dans le dataset :

```
fig = px.histogram(df, x="Region",
                  template='plotly_dark',
                  color_discrete_sequence = ["#20124d"])

fig.update_layout(title = "<b>Repartition des données selon l'attribut 'Region' dans le dataset</b>",
                  title_x = 0.5,
                  title_font = dict(size = 20),
                  uniformtext_minsize = 15)

fig.show()
```




```

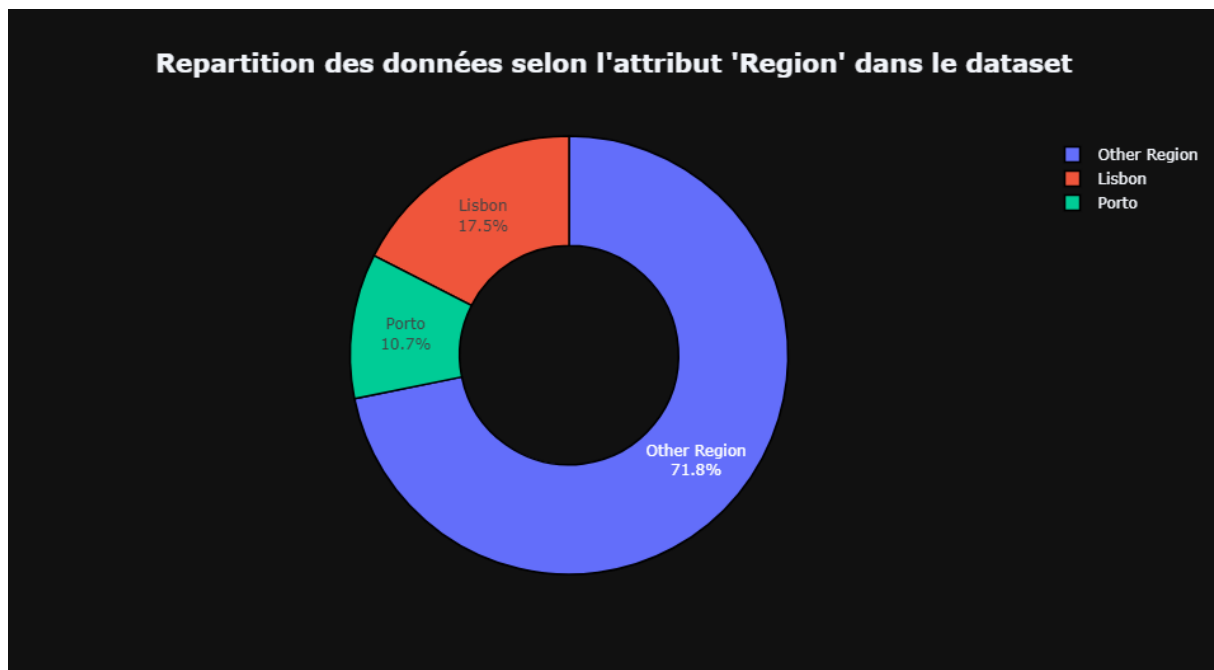
df["Region"] = np.where(df["Region"] == 1, "Lisbon", np.where(df["Region"] == 2, "Porto", "Other Region"))
fig = px.pie(df, names = "Region",
             title = "<b>Repartition des données selon l'attribut 'Region' dans le dataset</b>",
             hole = 0.5, template = "plotly_dark")

fig.update_traces(textposition='inside',
                  textinfo='percent+label',
                  marker=dict(line=dict(color='#000000', width = 1.5)))

fig.update_layout(title_x = 0.5,
                  title_font = dict(size = 20),
                  uniformtext_minsize = 15)

fig.show()

```



Dans cette figure on s'intéresse à la distribution des produits selon les régions, dans notre cas on a 3 classes qui déterminent 3 régions qui sont Lisbonne, Porto et autres Région. On voit bien que la classe majoritaire c'est la classe "Other Region". Ce qui signifie que on a plus de client dans cette classe, en deuxième position on a Lisbonne puis en dernier on a Porto.

```

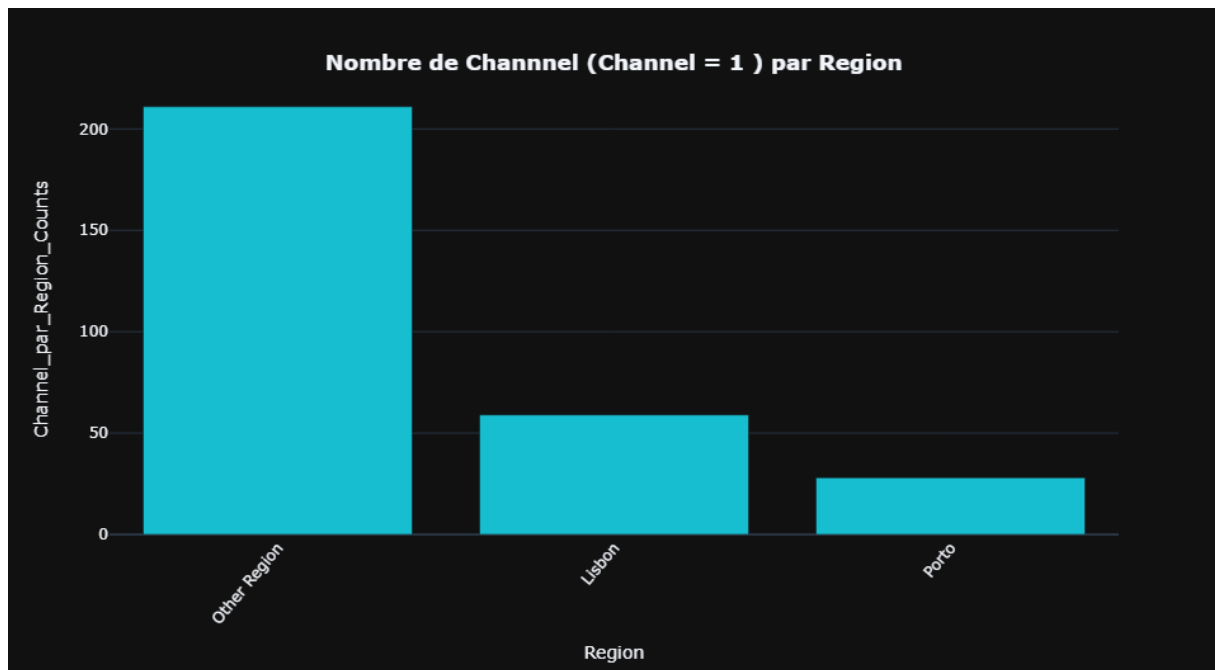
# Créer une nouvelle colonne "Region_Name" avec les noms de régions correspondants
region_names = {1: "Lisbon", 2: "Porto", 3: "Other Region"}
df["Region_Name"] = df["Region"].map(region_names)
Channel_Region_wise = Channel_Region_wise.rename(index={1: "Lisbon", 2: "Porto", 3: "Other Region"})
fig = px.bar(data_frame = Channel_Region_wise,
             x = Channel_Region_wise.index,
             y = "Channel_par_Region_Counts",
             labels = {"index": "Region"},
             color_discrete_sequence = px.colors.qualitative.D3_r,
             template='plotly_dark')

fig.update_xaxes(tickangle = 310)

fig.update_layout(title={
    'text': "<b>Nombre de Channel (Channel = 1 ) par Region</b>",
    'y': 0.93,
    'x': 0.5,
    'xanchor': 'center',
    'yanchor': 'top'})

fig.show()

```



```

label = Channel_Region_wise.index
value = Channel_Region_wise['Channel_par_Region_Counts']
Channel_Region_wise = Channel_Region_wise.rename(index={1: "Lisbon", 2: "Porto", 3: "Other Region"})

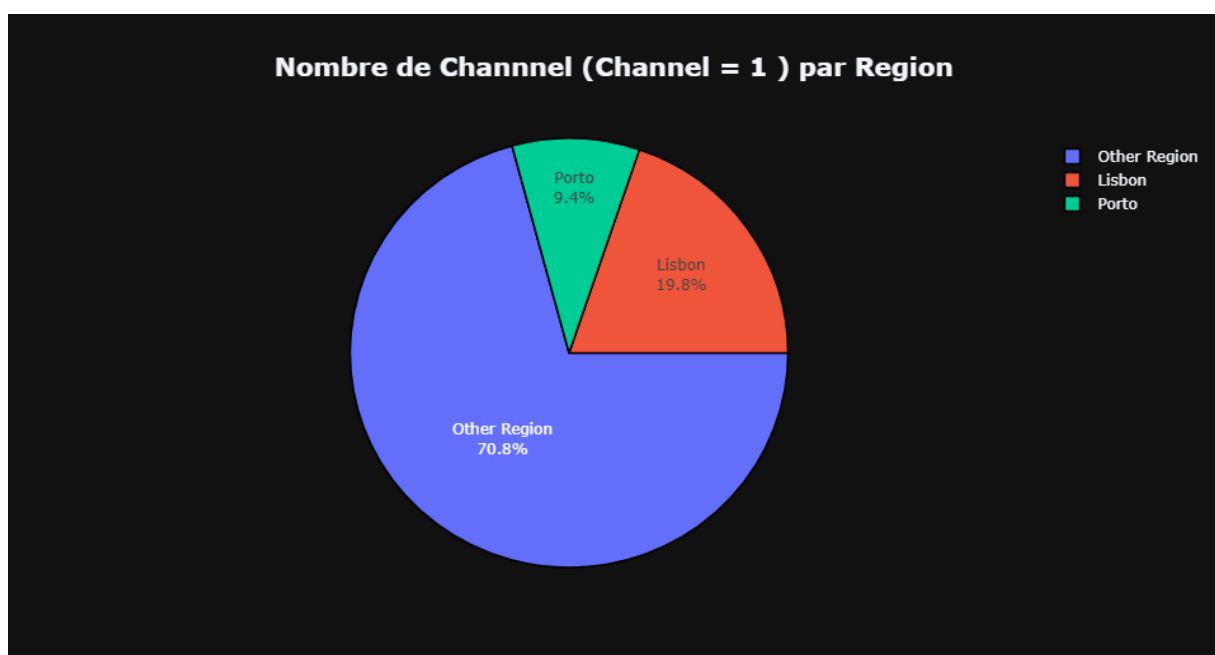
fig = go.Figure(data=[go.Pie(labels = label,
                             values = value,
                             rotation = 90)])

fig.update_traces(textposition = 'inside',
                  textinfo = 'percent+label',
                  marker = dict(line = dict(color = '#000000', width = 1.5)))

fig.update_layout(title_text="<b>Nombre de Channnel (Channel = 1 ) par Region</b>",
                  title_x = 0.5,
                  title_font = dict(size = 20),
                  uniformtext_minsize = 15,
                  template='plotly_dark')

fig.show()

```



Data Cleaninig :

Après vérification (si notre dataset possède des valeurs manquantes) on a trouvé que ce dernier est complet et il ne possède aucune valeurs manquantes (Nan). La figure montre bien que y'a pas de valeurs nulles. Dans le cas de valeurs manquantes, il existe 3 techniques pour remplacer les valeurs manquantes qui sont "Moyenne", "Médiane", et "Plus fréquent".

Label encoding :

À partir des étapes ci-dessus, nous pouvons voir qu'il n'existe pas de attributs non numériques pour chaque enregistrement dans notre dataset, en général dans le cas où on a des attributs catégorielles, **scatter_matrix** et certains algorithmes d'apprentissage s'attendent à ce que les données d'entrée soient numériques, ce qui nécessite la conversion des caractéristiques non numériques (appelées variables catégorielles). Une façon populaire de convertir les variables catégorielles est d'utiliser soit le **Label Encoding** ou le **One Hot Encoder**.

En utilisant le One Hot Encoder, chaque valeur catégorielle est convertie en une nouvelle colonne et une valeur 1 ou 0 (notation vrai/faux) est affectée à la colonne. Par exemple, si dans chaque i colonne on a X_i valeurs distinctes, le One Hot Encoder rajouteras Somme (X_i) colonnes à un dataframe ce qui est quand même beaucoup.

En utilisant le Label Encoding, chaque valeur est mappée en une unique valeur sans collision. Cette méthode présente par contre un inconvénient : les valeurs numériques peuvent être mal interprétées par les algorithmes quand celles-ci ont une sorte de hiérarchie/ordre entre elles.

Oversampling avec l'algorithme SMOTE :

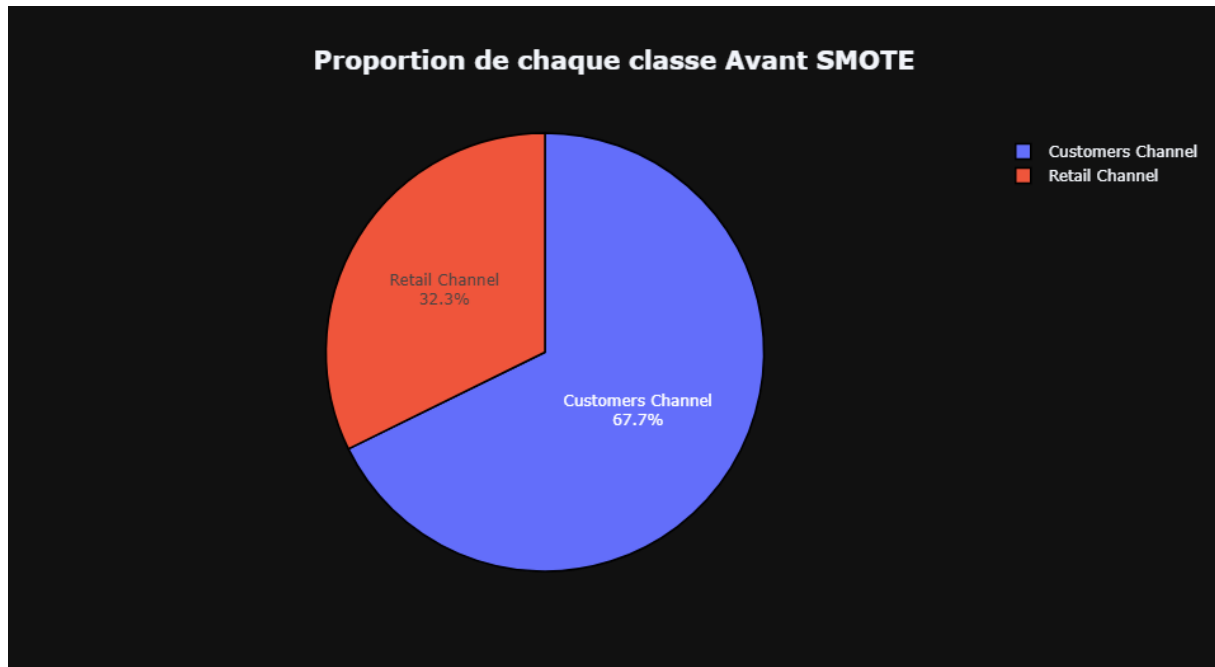
Le déséquilibre des données reflète généralement une distribution inégale des classes dans un ensemble de données.

Effectuer l'apprentissage pour un modèle sur de données déséquilibrées peut être dangereux. Car si la précision est utilisée pour mesurer la qualité d'un modèle, un modèle qui classe tous les échantillons testés dans la catégorie "1" (Channel = 1) peut être dangereux. (Channel = 1) aura une excellente précision (99,8 %), mais il est évident que ce modèle ne nous fournira aucune information précieuse. En effet, il aura appris que tout le phénomène s'explique avec les données de la classe majoritaire.

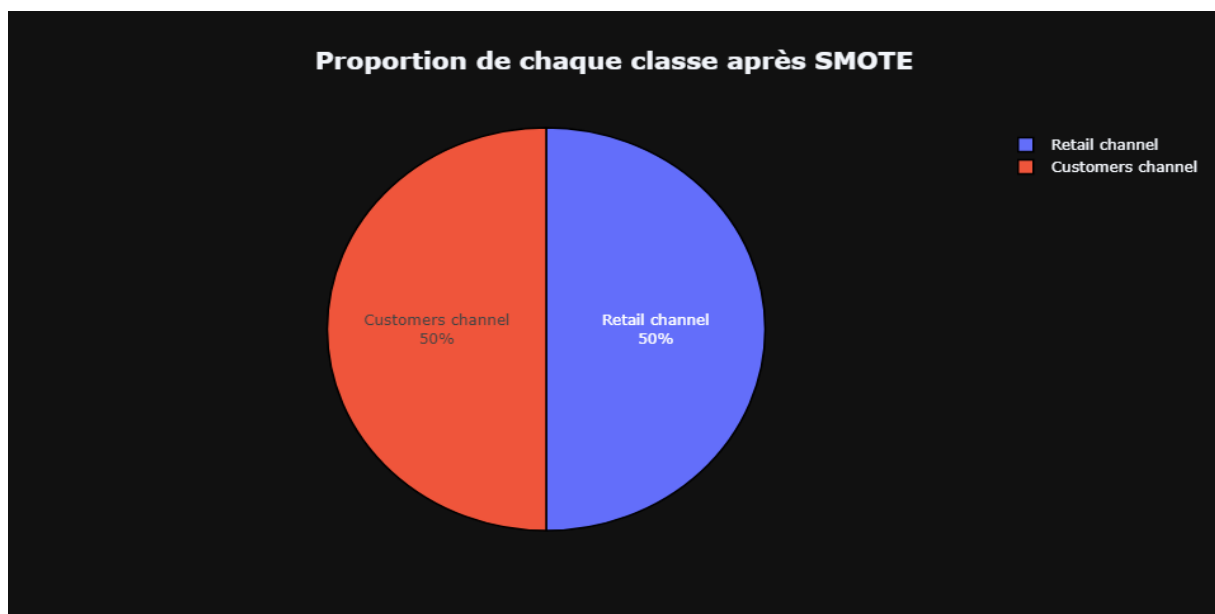
Afin de corriger ce problème, nous avons deux possibilités : soit augmenter le nombre d'instance de la classe minoritaire (oversampling) soit diminuer le nombre d'instances de la classe majoritaire (undersampling).

Dans notre cas on va utiliser la méthode smote qui est une méthode d'oversampling pour équilibrer les classes

✓ Proportion de chaque classe avant SMOTE :



✓ Proportion de chaque classe après SMOTE :

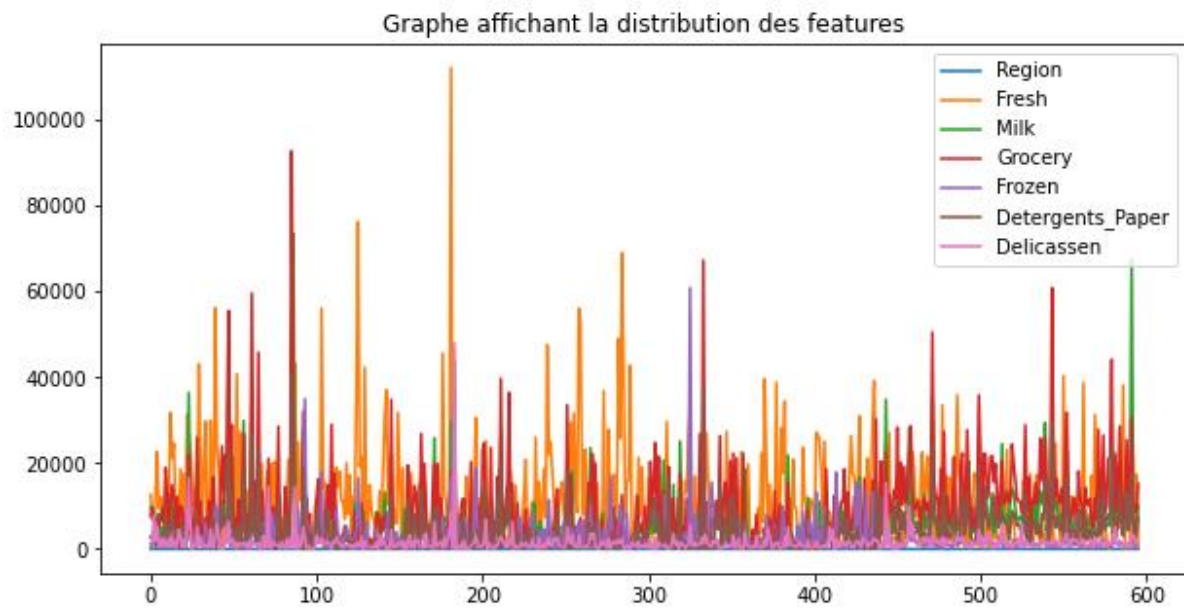


Exploratory data analysis :

✓ Extraire X et Y :

```
#EXTRAIRE X ET Y
y = pd.DataFrame({'Channel': df['Channel']})
x = df.drop('Channel', axis = 1)
```

```
x.plot(figsize=(10,5),title="Graphe affichant la distribution des features")
```



Channel	
0	2
1	2
2	2
3	1
4	2
...	...
591	2
592	2
593	2
594	2
595	2

Feature Selection Algorithms :

➤ ROUGH SET :

Le principe repose sur l'idée de définir une réduction d'un ensemble de données en utilisant une relation d'équivalence pour regrouper les objets similaires en un seul concept. Les objets qui ne peuvent pas être classés de manière définitive dans un concept sont considérés comme incertains ou flous.

1. Fonction qui calcule toutes les combinaisons possibles entre les attributs de base :

```
# Fonction qui calcule toutes les combinaisons possibles entre les attributs de base
import itertools
def recup_liste_combinaisons():

    attributs = ['Region', 'Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Delicassen']
    liste_combinaison = []

    for i in range(1, len(attributs)):
        combinaisons = itertools.combinations(attributs, i)
        liste_combinaison.append(combinaisons)

    return liste_combinaison
```

```
liste_combinaison = recup_liste_combinaisons()
```

2. Récupérer les dataset des combinaisons :

```
attributs = ['Region', 'Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Delicassen']
List_df = [] #liste qui va contenir les df des combinaisons
for i in range(0, len(liste_combinaison)):
    for j in liste_combinaison[i]:
        df_temp = x
        for k in range(0, len(attributs)):
            if(attributs[k] not in j): # supprimer les attributs qui n'appartiennent pas à j(liste_combinaison)
                df_temp = df_temp.drop(attributs[k], axis = 1)
        List_df.append(df_temp)
```

3. Calculer la Positive Region :

```
def positive_region(data):
    X = data.values.tolist()
    Y = y.values.tolist()
    Positive_Region = []
    for i in range(len(data)):
        List_lower = [] #créer une liste pour lower approxi
        List_lower.append(i) #Ajouter le premier élément
        boolean = False # créer un boolean pour savoir si on est sorti avec le break
        for j in range(0, len(data)):
            if (X[i] == X[j]):
                if (Y[i] == Y[j]):
                    List_lower.append(j)
                else:
                    boolean = True
                    break
        if (boolean == False):
            for k in range(len(List_lower)):
                Positive_Region.append(List_lower[k])
    return Positive_Region
```

4. Récupérer les reduct :

```
positive_region_all = positive_region(x)
```

```
reduct = []
for i in List_df:
    if(set(positive_region_all) == set(positive_region(i))):
        reduct.append(i)
```

5. Afficher les reduct :

```
reduct
```

```
len(reduct)
```

```
x_RST = reduct[10]
x_RST
```

- **QuickReduct** : QuickReduct est un algorithme de la famille Rough set. QuickReduct est un algorithme qui permet de déterminer les attributs minimaux qui sont nécessaires pour distinguer les objets dans un ensemble de données avec une certaine précision.

1. Transformer le dataset en une liste qui contient des listes :

```
U = df.values.tolist()
U = [[index] + value for index, value in enumerate(U)]
```

```
U
```

2. Fonction qui permet d'avoir les différentes classes :

```
def equivalence_partition( iterable, index ):
    classes = []
    dclasses = {}
    for o in iterable: # for each object
        # find the class it is in
        found = False
        for c in classes:
            indice_ele = next(iter(c))
            element = [iterable[indice_ele][ind] == o[ind] for ind in index]
            if all(element): # is it equivalent to this class?
                c.add( o[0] )
                dclasses[o[0]] = c
                found = True
                break
        if not found: # it is in a new class
            classes.append( set([o[0]]) )
            dclasses[o[0]] = classes[-1]
    return classes, dclasses
```

3. calculer la "Lower Approximation" de chaque classes :

```
def lower_appr(B):
    ind_B = equivalence_partition( U, B )[1]
    ind_d = equivalence_partition( U, D )[1]
    lower_appr_set = set()
    for x, ele in enumerate(U):
        if ind_B[x].issubset(ind_d[x]):
            lower_appr_set.add(x)
    return lower_appr_set
```

4. Calculer Gamma qui fait référence a la dependency de chaque attribut :

```
def gamma(B):
    return float(len(lower_appr(B)))/float(len(U))
```

5. Fonction permet de récupérer le bon reduct :

```
def qreduct(C):
    R = set()
    while True:
        T = R
        for x in C-R:
            if gamma(R.union(set([x]))) > gamma(T):
                T = R.union(set([x]))
        R = T
        if gamma(R) == gamma(C):
            break
    return R
```

```
decision=len(df.columns)#_____ defining the decision index
D = [decision]
B = set([ i for i in range(1,decision)]) #_____ defining condition index
Features= qreduct(B)
```

```
df_Quick.head()
```

	Fresh	Milk
0	12669	9656
1	7057	9810
2	6353	8808
3	13265	1196
4	22615	5410

- **Fuzzy Rough Set** : Fuzzy Rough est une extension de Rough Sets qui intègre la notion de flou dans la réduction d'un ensemble de données.

Consiste à utiliser des niveaux de pertinence flous pour évaluer la similitude entre les objets dans un ensemble de données.

1. **Sélectionner les meilleurs attributs** :

```
# Sélectionner Les meilleurs attributs en utilisant la méthode de sélection de caractéristiques FRFS
preprocessor = FRFS() # n_features = 2
model = preprocessor(x.values, y1.values)
X_selected = model(x.values)
```

```
df_Fuzzy.head()
```

	Fresh	Milk	Grocery	Frozen
0	12669	9656	7561	214
1	7057	9810	9568	1762
2	6353	8808	7684	2405
3	13265	1196	4221	6404
4	22615	5410	7198	3915

- **Pearson Corrélation** :

Utilise le coefficient de corrélation de Pearson pour évaluer la corrélation entre chaque paire de fonctionnalités et sélectionner celles qui ont la plus forte corrélation avec la variable cible. Les fonctionnalités hautement corrélées sont considérées comme redondantes et ne fournissent pas beaucoup d'informations supplémentaires, ce qui peut conduire à un surajustement du modèle. La sélection de fonctionnalités basée sur la corrélation de Pearson peut aider à réduire le nombre de fonctionnalités sans sacrifier la performance du modèle.

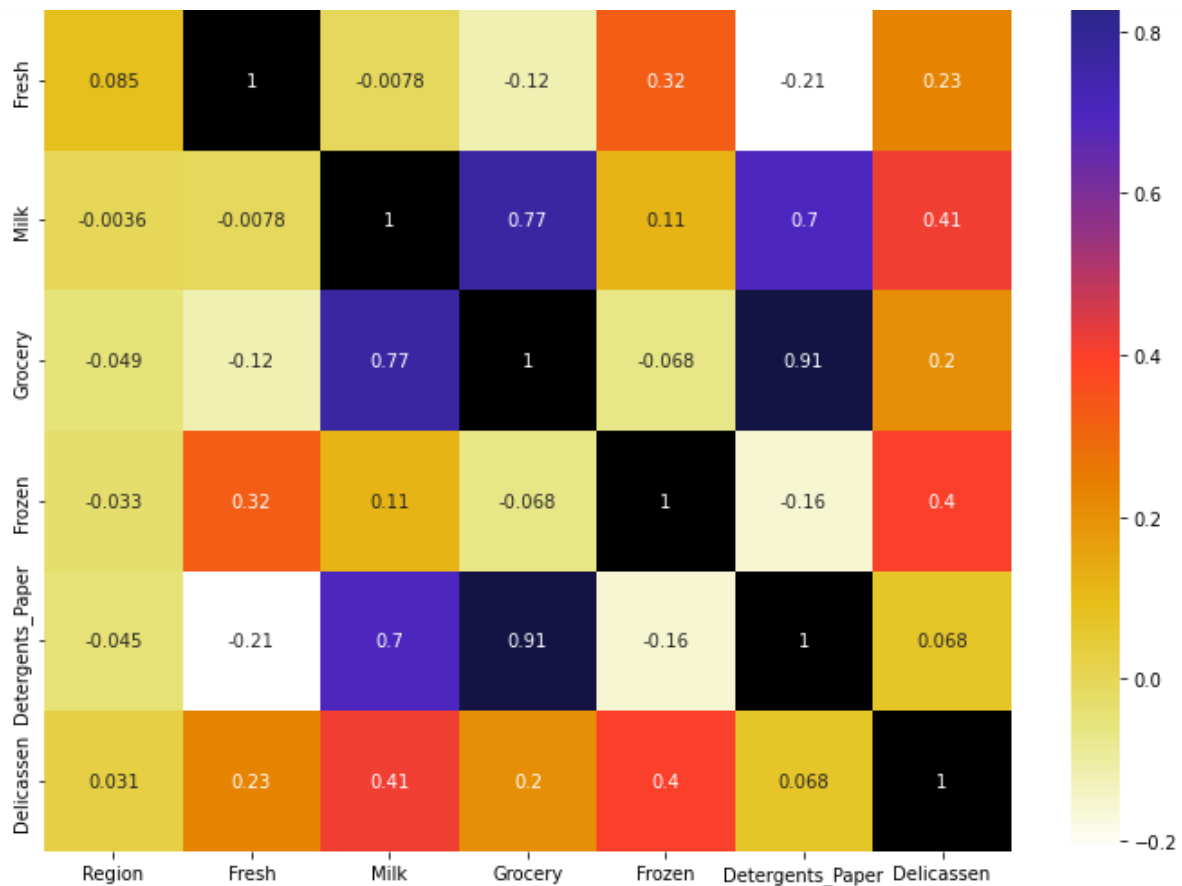
1. **Séparation en deux jeux de données** :

```
x_PR = x
y_PR = y
```

```
# SÉPARATION EN DEUX JEUX DE DONNÉES avec un ratio de 1/5
X_train4, X_test4, y_train4, y_test4 = train_test_split(x_PR, y_PR, test_size=0.2, random_state=42, stratify=y)
```


2. Générer la matrice de corrélation :

```
plt.figure(figsize=(12,10))
cor = X_train4.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.CMRmap_r)
plt.show()
```



3. Fonction qui permet de sélectionner les attributs les plus corrélés :

```
# With the following function we can select highly correlated features
# it will remove the first feature that is correlated with anything other features
def correlation(dataset, threshold):
    col_corr = set() #set of all names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i,j]) > threshold:
                colname = corr_matrix.columns[i] # getting name of column
                col_corr.add(colname)
    return col_corr
```

```
corr_features = correlation(X_train4, 0.7)
len(set(corr_features))
```

	Region	Fresh	Milk	Frozen	Delicassen
243	1	11210	3576	561	2398
332	2	22321	3216	2208	2602
408	3	8257	3880	1646	344
488	2	14911	11780	3788	2060
111	3	12579	11114	805	1519
...
286	3	7149	2247	1619	128
523	1	2361	6601	879	826
268	1	11908	8053	1069	698
333	2	8565	4980	131	1215
433	3	1982	3218	1541	1449

CONCLUSION :

Dans ce chapitre, nous avons mis en œuvre plusieurs techniques de sélection de caractéristiques en utilisant Python et des bibliothèques couramment utilisées pour la science des données et l'apprentissage automatique. Nous avons également utilisé un ensemble de données réel pour illustrer comment ces techniques peuvent être appliquées à des problèmes réels.

La mise en œuvre de ces techniques a été réalisée avec succès, et nous avons pu appliquer chaque technique pour sélectionner un sous-ensemble de caractéristiques qui maximise la performance du modèle d'apprentissage automatique. Cela démontre que ces techniques sont applicables à des problèmes réels.

Dans le prochain chapitre, nous évaluerons les performances des différents algorithmes de sélection de caractéristiques que nous avons implémentés, en utilisant des Algorithmes de machine Learning ainsi les mesures telles que la précision, le rappel et la F-mesure. Nous comparerons également les résultats avec ceux obtenus en utilisant toutes les fonctionnalités disponibles pour voir si la sélection de caractéristiques améliore la performance du modèle.

CHAPITRE IV :

EXPERIMENTATION ET DESCUSSION DES RESULTATS

INTRODUCTION :

Dans ce chapitre, nous allons évaluer les performances des différents algorithmes de sélection de caractéristiques que nous avons implémentés, en utilisant deux algorithmes de machine Learning populaires : K-Nearest Neighbors (KNN) et Random Forest.

Nous allons évaluer la performance de ces algorithmes en utilisant plusieurs mesures de performance couramment utilisées en classification, telles que la précision, le rappel, la F-mesure et l'aire sous la courbe ROC (AUC). Nous allons également tracer la courbe ROC pour chaque algorithme pour mieux comprendre leur capacité à discriminer les classes positives et négatives.

Pour effectuer cette évaluation, nous allons utiliser l'ensemble de données réel que nous avons utilisé dans le chapitre précédent pour la sélection de caractéristiques. Nous allons utiliser les sous-ensembles de caractéristiques sélectionnées par chaque algorithme de sélection de caractéristiques pour entraîner les modèles KNN et Random Forest, et comparer leurs performances avec celle des modèles entraînés avec toutes les fonctionnalités disponibles.

➤ Algorithme de classification :

Dans cette section, nous allons appliquer deux algorithmes de classification (KNN ET Random Forest) sur chaque résultat (Attributs sélectionnés) des différents algorithmes de sélection de caractéristiques utilisés.

KNN : consiste à assigner une classe à une nouvelle observation en se basant sur la classe majoritaire des k observations les plus proches dans le jeu de données d'entraînement.

La similarité entre les observations est généralement mesurée à l'aide de la distance Euclidienne ou de toute autre mesure de distance.

RANDOM FOREST : Il construit un ensemble de modèles de décision (arbres de décision), puis il fait la combinaison de ses derniers pour améliorer la performance du modèle.

I. Classification avec le dataset initial :

1. Séparation du jeu de données :

```
# SÉPARATION EN DEUX JEUX DE DONNÉES avec un ratio de 1/5
X_train0, X_test0, y_train0, y_test0 = train_test_split(df, y, test_size=0.2, random_state=42, stratify=y)
```

2. Algorithme KNN :

```
#Appliquez l'algorithme des k plus proches voisins, pour chaque valeur de k, de sorte a trouver le meilleur score.

scoreList = []

k = range(1,20)

for i in k:
    knn = KNeighborsClassifier(n_neighbors = i) # n_neighbors means k
    knn.fit(X_train0, y_train0)
    scoreList.append(knn.score(X_test0, y_test0))

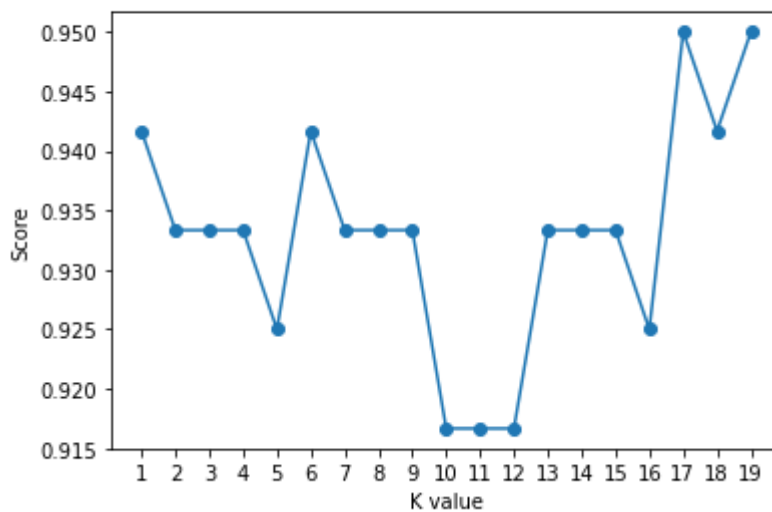
print("Maximum KNN Score is {:.2f}%".format(max(scoreList) * 100))

for i in range(len(scoreList)):
    if(max(scoreList) == scoreList[i]):
        top_k = i+1

knn = KNeighborsClassifier(n_neighbors = top_k) # n_neighbors means k
knn.fit(X_train0, y_train0)
y_pred_knn1 = knn.predict(X_test0)

plt.plot(range(1,20), scoreList, 'o-')
plt.xticks(np.arange(1,20,1))
plt.xlabel("K value")
plt.ylabel("Score")
plt.show()
```

Maximum KNN Score is 95.00%



3. Calcul du taux d'erreur :

```
#Calcul du taux d'erreur :

knn = KNeighborsClassifier(n_neighbors = 1)
knn.fit(X_train0, y_train0)
#Le taux de précision
y_test_pred0 = knn.predict(X_test0)

#taux d'erreur
score = 1 - accuracy_score(y_test0, y_test_pred0)
score

0.05833333333333335
```

4. Matrice classification report :

```
#Metric2 : CLASSIFICATION REPORT
m = classification_report(y_test0,y_test_pred0)
print("Knn")
print (m)
```

```
Knn
      precision    recall  f1-score   support

     1:   0.95      0.93      0.94         60
     2:   0.93      0.95      0.94         60

 accuracy: 0.94
 macro avg: 0.94      0.94      0.94
 weighted avg: 0.94      0.94      0.94
```

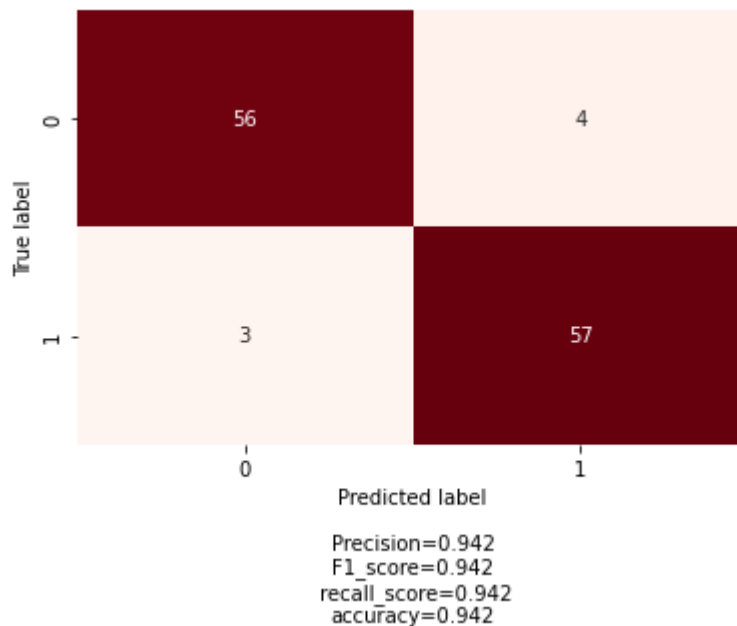
5. Les métriques :

```
# Affichage des resultats (les valeurs des métriques)

# Calcul les valeurs des métriques
precision_Score0 = precision_score(y_test0, y_test_pred0, average='micro')
F1_score0 = f1_score(y_test0, y_test_pred0, average='micro')
recall0 = recall_score(y_test0, y_test_pred0, average='micro')
accuracy0 = accuracy_score(y_test0, y_test_pred0)

# Affichage de la matrice de confusion
result = confusion_matrix(y_test0, y_test_pred0)
sns.heatmap(result, annot=True,cmap='Reds',cbar=False)

# Affichage les valeurs des métriques
stats_text = "\n\nPrecision={:0.3f}\nF1_score={:0.3f}\n recall_score={:0.3f}\naccuracy={:0.3f}".format(precision_Score0,F1_score0,recall0,accuracy0)
plt.ylabel('True label')
plt.xlabel('Predicted label' + stats_text)
```



6. RANDOM FOREST :

```
# Different RandomForestClassifier hyperparameters
rf_grid = {"n_estimators": np.arange(10, 1000, 50),
           "max_depth": [None, 3, 5, 10],
           "min_samples_split": np.arange(2, 20, 2),
           "min_samples_leaf": np.arange(1, 20, 2)}
```

```
# Setup random seed
from sklearn.model_selection import GridSearchCV, cross_validate, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
np.random.seed(42)

# Setup random hyperparameter search for RandomForestClassifier
rs_rf0 = RandomizedSearchCV(RandomForestClassifier(),
                             param_distributions=rf_grid,
                             cv=5,
                             n_iter=20,
                             verbose=True)

# Fit random hyperparameter search model
rs_rf0.fit(X_train0, y_train0);
```

Fitting 5 folds for each of 20 candidates, totalling 100 fits

7. Meilleurs paramètres :

```
rs_rf0.best_params_

{'n_estimators': 610,
 'min_samples_split': 18,
 'min_samples_leaf': 1,
 'max_depth': 5}
```

8. Score du modèle :

```
rs_rf0.score(X_test0, y_test0)
```

1.0

```
y_test_pred0 = rs_rf0.predict(X_test0)
```

9. Matrice classification report :

```
m = classification_report(y_test0, y_test_pred0)
print("Random Forest")
print(m)
```

```
Random Forest
              precision    recall  f1-score   support

     1         1.00      1.00      1.00         60
     2         1.00      1.00      1.00         60

 accuracy          1.00
 macro avg          1.00
weighted avg          1.00
```

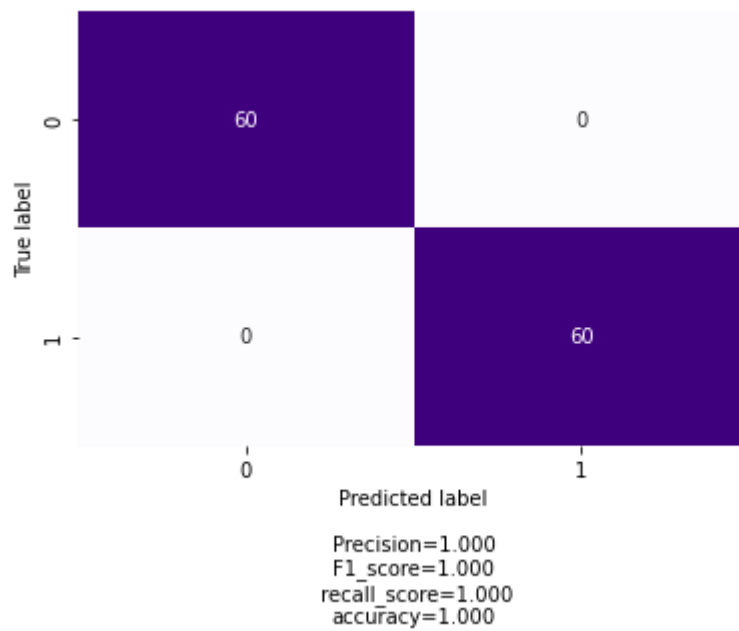
10. Les métriques :

```
# Affichage des resultats (les valeurs des métriques)

# Calcul Les valeurs des métriques
precision_Score00 = precision_score(y_test0, y_test_pred0, average='micro')
F1_score00 = f1_score(y_test0, y_test_pred0, average='micro')
recall00 = recall_score(y_test0, y_test_pred0, average='micro')
accuracy00 = accuracy_score(y_test0, y_test_pred0)

# Affichage de La matrice de confusion
result = confusion_matrix(y_test0, y_test_pred0)
sns.heatmap(result, annot=True, cmap='Purples', cbar=False)

# Affichage Les valeurs des métriques
stats_text = "\n\nPrecision={:.3f}\nF1_score={:.3f}\n recall_score={:.3f}\naccuracy={:.3f}".format(precision_Score00, F1_score00, recall00, accuracy00)
plt.ylabel('True label')
plt.xlabel('Predicted label' + stats_text)
```



11. Roc Curve :

```
import matplotlib.pyplot as plt

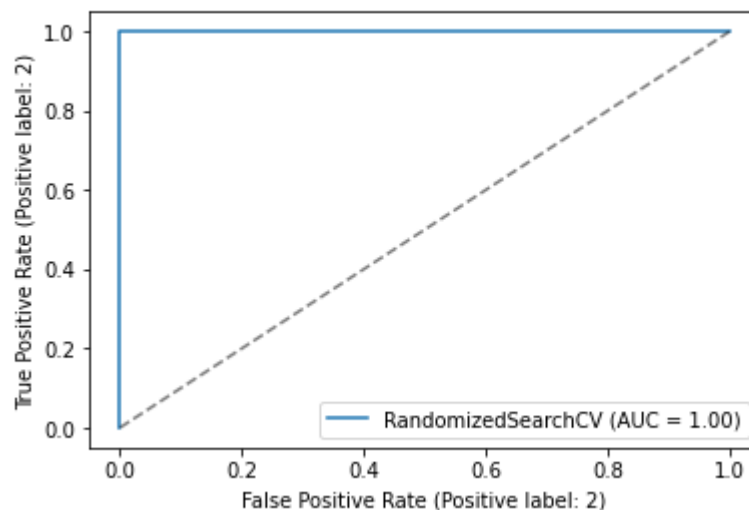
# Plot ROC curve and calculate AUC metric
rfc_disp0 = plot_roc_curve(rs_rf0, X_test0, y_test0)

# Add diagonal line to the plot
rfc_disp0.ax_.plot([0, 1], [0, 1], linestyle='--', color='black', alpha=.5)

# Set title and legend
rfc_disp0.figure_.suptitle("ROC curve for Random Forest")
plt.legend(loc='lower right')

# Show the plot
plt.show()
```

ROC curve for Random Forest



12. Comparaison en termes de performance de prédiction entre KNN et Random Forest :

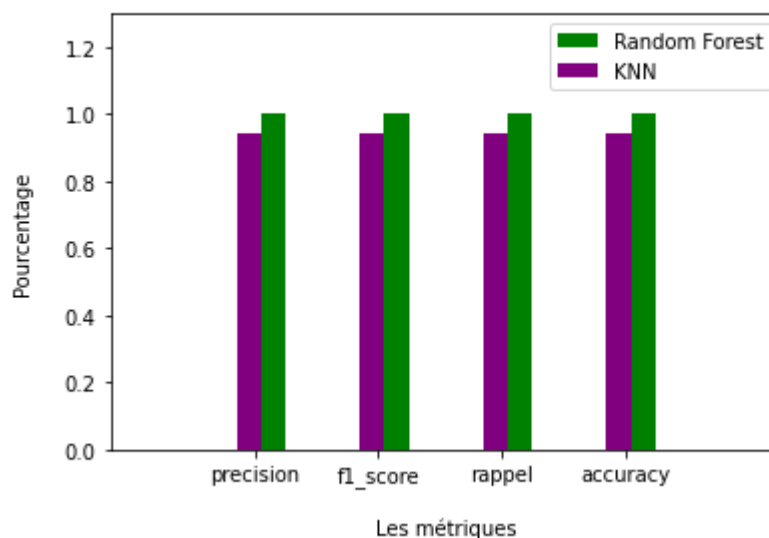
```
barWidth = 0.2
figsize = (10,6)
y1 = [precision_Score0,F1_score0,recall0,accuracy0]
y2 = [precision_Score00,F1_score00,recall00,accuracy00]
r1 = range(len(y1))
r2 = [x + barWidth for x in r1]

plt.bar(r2, y2, width = barWidth, color = ['green' for i in y1],
        linewidth = 4)
plt.bar(r1, y1, width = barWidth, color = ['purple' for i in y1],
        linewidth = 2)

plt.xticks([r + barWidth / 4 for r in range(len(y1))], ['precision', 'f1_score', 'rappel', 'accuracy'])

plt.title("Histogrammes groupés des métriques d'évaluation pour KNN, Random Forest\n")
plt.margins(0.3)
plt.xlabel('\nLes métriques')
plt.ylabel('Pourcentage\n')
plt.legend(["Random Forest", "KNN"])
```

Histogrammes groupés des métriques d'évaluation pour KNN, Random Forest



II. Classification avec Rough Set :

1. Séparation du jeu de données :

```
# SÉPARATION EN DEUX JEUX DE DONNÉES avec un ratio de 1/5
X_train1, X_test1, y_train1, y_test1 = train_test_split(x_RST, y, test_size=0.2, random_state=42,stratify=y)
```

2. Algorithme KNN :

```
#Appliquez l'algorithme des k plus proches voisins, pour chaque valeur de k, de sorte a trouver Le meilleur score.

scoreList = []

k = range(1,20)

for i in k:
    knn = KNeighborsClassifier(n_neighbors = i) # n_neighbors means k
    knn.fit(X_train1, y_train1)
    scoreList.append(knn.score(X_test1, y_test1))

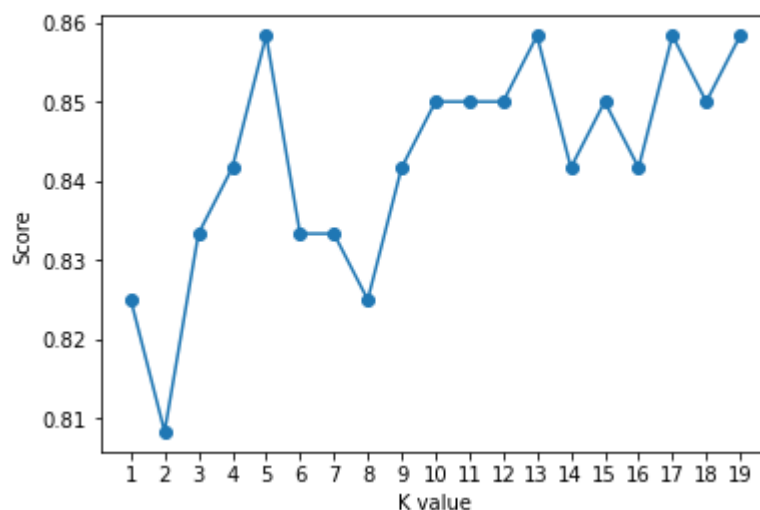
print("Maximum KNN Score is {:.2f}%".format(max(scoreList) * 100))

for i in range(len(scoreList)):
    if(max(scoreList) == scoreList[i]):
        top_k = i+1

knn = KNeighborsClassifier(n_neighbors = top_k) # n_neighbors means k
knn.fit(X_train1, y_train1)
y_pred_knn1 = knn.predict(X_test1)

plt.plot(range(1,20), scoreList,'o-')
plt.xticks(np.arange(1,20,1))
plt.xlabel("K value")
plt.ylabel("Score")
plt.show()
```

Maximum KNN Score is 85.83%



3. Calcul du taux d'erreur :

```
#Calcul du taux d'erreur :

knn = KNeighborsClassifier(n_neighbors = 1)
knn.fit(X_train1,y_train1)
#Le taux de précision
y_test_pred1= knn.predict(X_test1)

#taux d'erreur
score = 1 - accuracy_score(y_test1, y_test_pred1)
score
```

0.17500000000000004

4. Matrice classification report :

```
#Metric2 : CLASSIFICATION REPORT
m = classification_report(y_test1,y_test_pred1)
print("Knn")
print (m)
```

```
Knn
              precision    recall  f1-score   support

         1         0.84      0.80      0.82         60
         2         0.81      0.85      0.83         60

 accuracy          0.83
 macro avg         0.83      0.82      0.82
 weighted avg      0.83      0.82      0.82
```

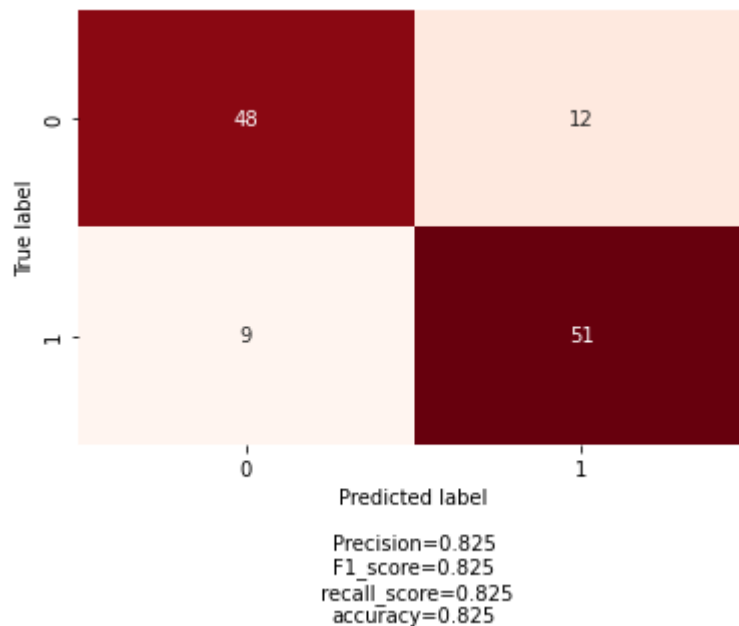
5. Les métriques :

```
# Affichage des resultats (les valeurs des métriques)

# Calcul les valeurs des métriques
precision_score1 = precision_score(y_test1, y_test_pred1, average='micro')
F1_score1 = f1_score(y_test1, y_test_pred1, average='micro')
recall1 = recall_score(y_test1, y_test_pred1, average='micro')
accuracy1 = accuracy_score(y_test1, y_test_pred1)

# Affichage de la matrice de confusion
result = confusion_matrix(y_test1, y_test_pred1)
sns.heatmap(result, annot=True,cmap='Reds',cbar=False)

# Affichage les valeurs des métriques
stats_text = "\n\nPrecision={:0.3f}\nF1_score={:0.3f}\n recall_score={:0.3f}\naccuracy={:0.3f}".format(precision_score1,F1_score1,recall1,accuracy1)
plt.ylabel('True label')
plt.xlabel('Predicted label' + stats_text)
```



6. RANDOM FOREST :

```
# Different RandomForestClassifier hyperparameters
rf_grid = {"n_estimators": np.arange(10, 1000, 50),
           "max_depth": [None, 3, 5, 10],
           "min_samples_split": np.arange(2, 20, 2),
           "min_samples_leaf": np.arange(1, 20, 2)}
```

```
# Setup random seed
from sklearn.model_selection import GridSearchCV, cross_validate, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
np.random.seed(42)

# Setup random hyperparameter search for RandomForestClassifier
rs_rf1 = RandomizedSearchCV(RandomForestClassifier(),
                             param_distributions=rf_grid,
                             cv=5,
                             n_iter=20,
                             verbose=True)

# Fit random hyperparameter search model
rs_rf1.fit(X_train1, y_train1);
```

Fitting 5 folds for each of 20 candidates, totalling 100 fits

7. Meilleurs paramètres :

```
rs_rf1.best_params_
```

```
{'n_estimators': 260,
 'min_samples_split': 8,
 'min_samples_leaf': 19,
 'max_depth': None}
```

8. Score du modèle :

```
rs_rf1.score(X_test1, y_test1)
```

0.825

```
y_test_pred1 = rs_rf1.predict(X_test1)
```

9. Matrice classification report :

```
m = classification_report(y_test1, y_test_pred1)
print("Random Forest")
print(m)
```

```
Random Forest
```

	precision	recall	f1-score	support
1	0.85	0.78	0.82	60
2	0.80	0.87	0.83	60
accuracy			0.82	120
macro avg	0.83	0.82	0.82	120
weighted avg	0.83	0.82	0.82	120

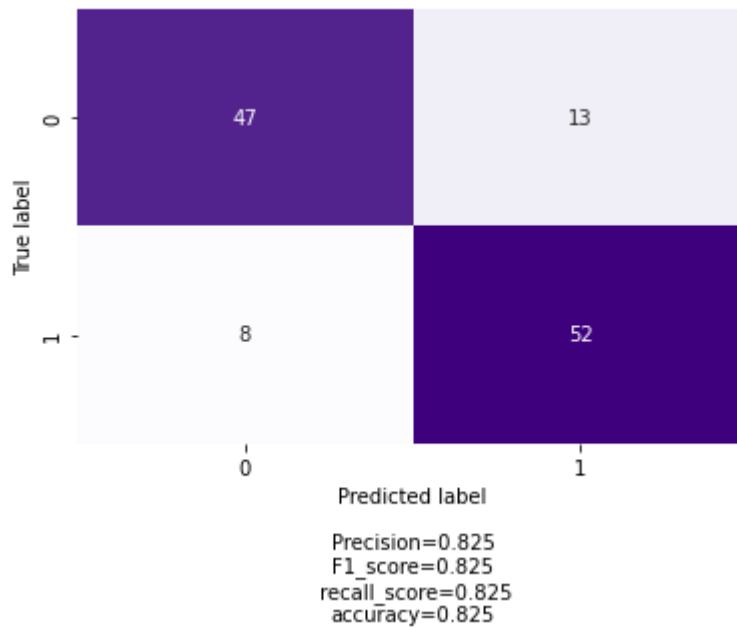
10. Les métriques :

```
# Affichage des resultats (Les valeurs des métriques)

# Calcul les valeurs des métriques
precision_Score11 = precision_score(y_test1, y_test_pred1, average='micro')
F1_score11 = f1_score(y_test1, y_test_pred1, average='micro')
recall11 = recall_score(y_test1, y_test_pred1, average='micro')
accuracy11 = accuracy_score(y_test1, y_test_pred1)

# Affichage de la matrice de confusion
result = confusion_matrix(y_test1, y_test_pred1)
sns.heatmap(result, annot=True, cmap='Purples', cbar=False)

# Affichage les valeurs des métriques
stats_text = "\n\nPrecision={:.3f}\nF1_score={:.3f}\n recall_score={:.3f}\naccuracy={:.3f}".format(precision_Score11, F1_score11, recall11, accuracy11)
plt.ylabel('True label')
plt.xlabel('Predicted label' + stats_text)
```



11. Roc Curve :

```
import matplotlib.pyplot as plt

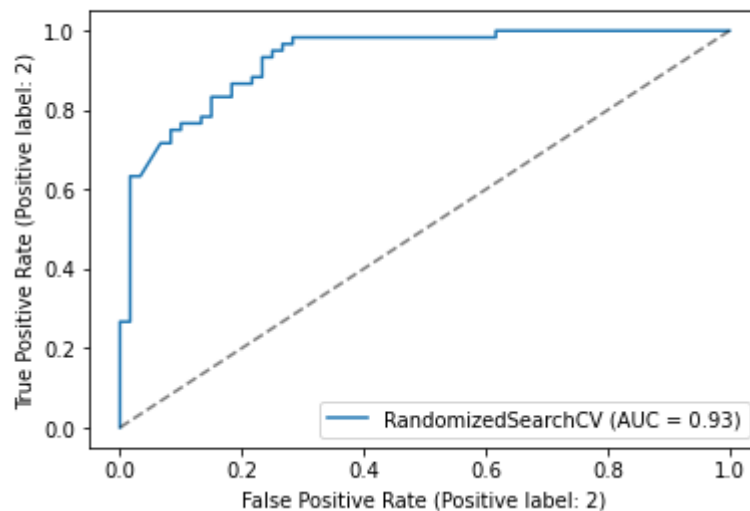
# Plot ROC curve and calculate AUC metric
rfc_disp1 = plot_roc_curve(rs_rf1, X_test1, y_test1)

# Add diagonal line to the plot
rfc_disp1.ax_.plot([0, 1], [0, 1], linestyle='--', color='black', alpha=.5)

# Set title and legend
rfc_disp1.figure_.suptitle("ROC curve for Random Forest")
plt.legend(loc='lower right')

# Show the plot
plt.show()
```

ROC curve for Random Forest



12. Comparaison en termes de performance de prédiction entre KNN et Random Forest :

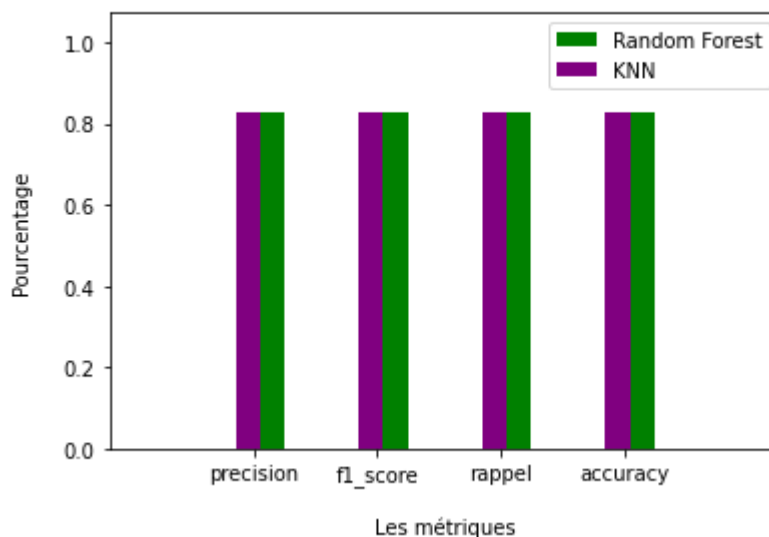
```
barWidth = 0.2
figsize = (10,6)
y1 = [precision_Score1,F1_score1,recall1,accuracy1]
y2 = [precision_Score11,F1_score11,recall11,accuracy11]
r1 = range(len(y1))
r2 = [x + barWidth for x in r1]

plt.bar(r2, y2, width = barWidth, color = ['green' for i in y1],
        linewidth = 4)
plt.bar(r1, y1, width = barWidth, color = ['purple' for i in y1],
        linewidth = 2)

plt.xticks([r + barWidth / 4 for r in range(len(y1))], ['precision', 'f1_score', 'rappel', 'accuracy'])

plt.title("Histogrammes groupés des métriques d'évaluation pour KNN, Random Forest\n")
plt.margins(0.3)
plt.xlabel('\nLes métriques')
plt.ylabel('Pourcentage\n')
plt.legend(["Random Forest", "KNN"])
```

Histogrammes groupés des métriques d'évaluation pour KNN, Random Forest



III. Classification avec QuickReduct :

1. Séparation du jeu de données :

```
# SÉPARATION EN DEUX JEUX DE DONNÉES avec un ratio de 1/5
X_train2, X_test2, y_train2, y_test2 = train_test_split(x_QR, y, test_size=0.2, random_state=42, stratify=y)
```

2. Algorithme KNN :

```
#Appliquez l'algorithme des k plus proches voisins, pour chaque valeur de k, de sorte a trouver Le meilleur score.

scoreList = []

k = range(1,20)

for i in k:
    knn = KNeighborsClassifier(n_neighbors = i) # n_neighbors means k
    knn.fit(X_train2, y_train2)
    scoreList.append(knn.score(X_test2, y_test2))

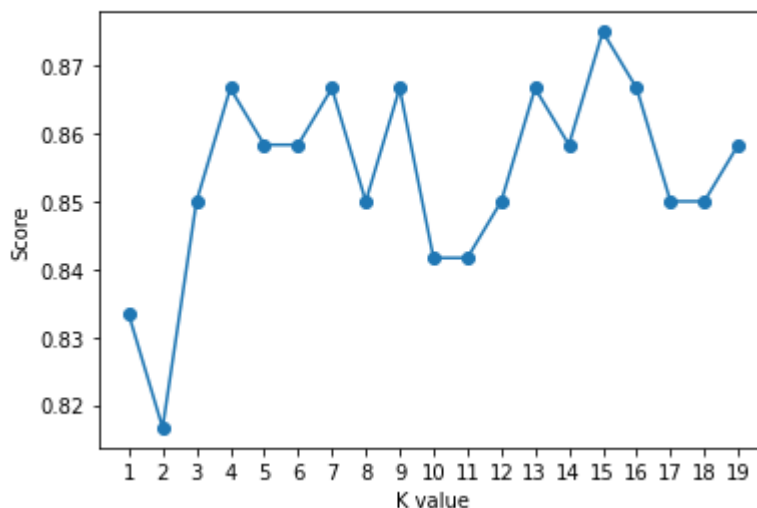
print("Maximum KNN Score is {:.2f}%".format(max(scoreList) * 100))

for i in range(len(scoreList)):
    if(max(scoreList) == scoreList[i]):
        top_k = i+1

knn = KNeighborsClassifier(n_neighbors = top_k) # n_neighbors means k
knn.fit(X_train2, y_train2)
y_pred_knn2 = knn.predict(X_test2)

plt.plot(range(1,20), scoreList,'o-')
plt.xticks(np.arange(1,20,1))
plt.xlabel("K value")
plt.ylabel("Score")
plt.show()
```

Maximum KNN Score is 87.50%



3. Calcul du taux d'erreur :

```
#Calcul du taux d'erreur :

knn = KNeighborsClassifier(n_neighbors = 1)
knn.fit(X_train2,y_train2)
#Le taux de précision
y_test_pred2= knn.predict(X_test2)

#taux d'erreur
score = 1 - accuracy_score( y_test2, y_test_pred2)
score

0.16666666666666663
```

4. Matrice classification report :

```
#Metric2 : CLASSIFICATION REPORT
m = classification_report(y_test2,y_test_pred2 )
print("Knn")
print (m)
```

```
Knn
```

	precision	recall	f1-score	support
1	0.87	0.78	0.82	60
2	0.80	0.88	0.84	60
accuracy			0.83	120
macro avg	0.84	0.83	0.83	120
weighted avg	0.84	0.83	0.83	120

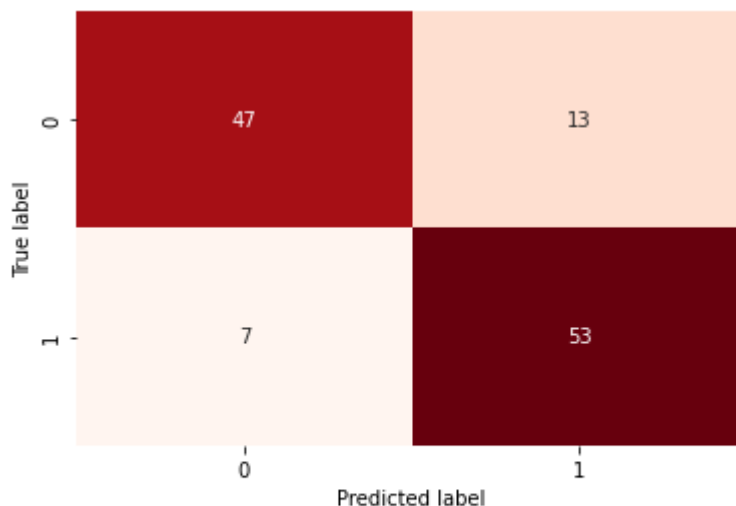
5. Les métriques :

```
# Affichage des resultats (Les valeurs des métriques)

# Calcul Les valeurs des métriques
precision_Score2 = precision_score(y_test2, y_test_pred2, average='micro')
F1_score2 = f1_score(y_test2, y_test_pred2, average='micro')
recall2 = recall_score(y_test2, y_test_pred2, average='micro')
accuracy2 = accuracy_score(y_test2, y_test_pred2)

# Affichage de La matrice de confusion
result = confusion_matrix(y_test2, y_test_pred2)
sns.heatmap(result, annot=True,cmap='Reds',cbar=False)

# Affichage Les valeurs des métriques
stats_text = "\n\nPrecision={:0.3f}\nF1_score={:0.3f}\n recall_score={:0.3f}\naccuracy={:0.3f}".format(precision_Score2,F1_s
plt.ylabel('True label')
plt.xlabel('Predicted label' + stats_text)
```



```
Precision=0.833
F1_score=0.833
recall_score=0.833
accuracy=0.833
```

6. RANDOM FOREST :

```
# Different RandomForestClassifier hyperparameters
rf_grid = {"n_estimators": np.arange(10, 1000, 50),
           "max_depth": [None, 3, 5, 10],
           "min_samples_split": np.arange(2, 20, 2),
           "min_samples_leaf": np.arange(1, 20, 2)}
```



```
# Setup random seed
from sklearn.model_selection import GridSearchCV, cross_validate, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
np.random.seed(42)

# Setup random hyperparameter search for RandomForestClassifier
rs_rf2 = RandomizedSearchCV(RandomForestClassifier(),
                             param_distributions=rf_grid,
                             cv=5,
                             n_iter=20,
                             verbose=True)

# Fit random hyperparameter search model
rs_rf2.fit(X_train2, y_train2);
```

Fitting 5 folds for each of 20 candidates, totalling 100 fits

7. Meilleurs paramètres :

```
rs_rf2.best_params_

{'n_estimators': 10,
 'min_samples_split': 16,
 'min_samples_leaf': 9,
 'max_depth': None}
```

8. Score du modèle :

```
rs_rf2.score(X_test2, y_test2)
```

0.85

```
y_test_pred2 = rs_rf2.predict(X_test2)
```

9. Matrice classification report :

```
m = classification_report(y_test2, y_test_pred2)
print("Random Forest")
print(m)
```

```
Random Forest
              precision    recall  f1-score   support

     1         0.86         0.83         0.85         60
     2         0.84         0.87         0.85         60

 accuracy          0.85
 macro avg          0.85
weighted avg          0.85
```

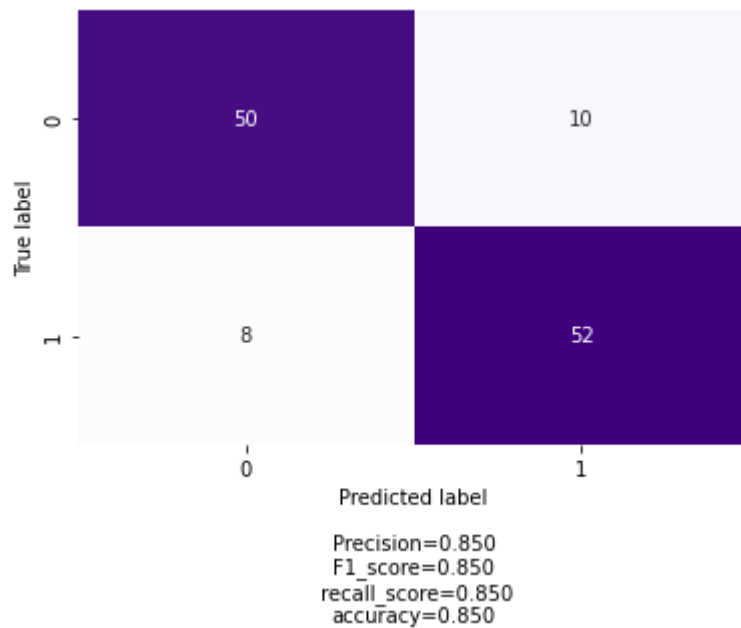
10. Les métriques :

```
# Affichage des resultats (Les valeurs des métriques)

# Calcul les valeurs des métriques
precision_Score22 = precision_score(y_test2, y_test_pred2, average='micro')
F1_score22 = f1_score(y_test2, y_test_pred2, average='micro')
recall22 = recall_score(y_test2, y_test_pred2, average='micro')
accuracy22 = accuracy_score(y_test2, y_test_pred2)

# Affichage de la matrice de confusion
result = confusion_matrix(y_test2, y_test_pred2)
sns.heatmap(result, annot=True, cmap='Purples', cbar=False)

# Affichage les valeurs des métriques
stats_text = "\n\nPrecision={:0.3f}\nF1_score={:0.3f}\n recall_score={:0.3f}\naccuracy={:0.3f}".format(precision_Score22, F1_score22, recall22, accuracy22)
plt.ylabel('True label')
plt.xlabel('Predicted label' + stats_text)
```



11. Roc Curve :

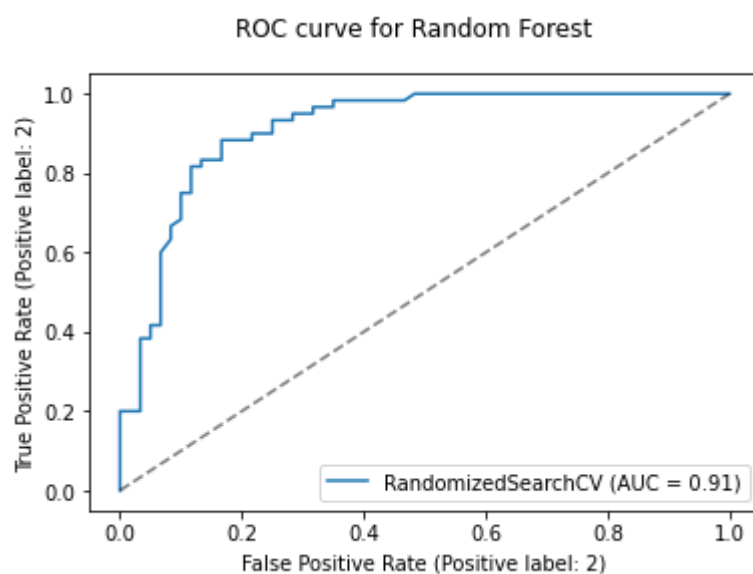
```
import matplotlib.pyplot as plt

# Plot ROC curve and calculate AUC metric
rfc_disp2 = plot_roc_curve(rs_rf2, X_test2, y_test2)

# Add diagonal line to the plot
rfc_disp2.ax_.plot([0, 1], [0, 1], linestyle='--', color='black', alpha=.5)

# Set title and legend
rfc_disp2.figure_.suptitle("ROC curve for Random Forest")
plt.legend(loc='lower right')

# Show the plot
plt.show()
```



12. Comparaison en termes de performance de prédiction entre KNN et Random Forest :

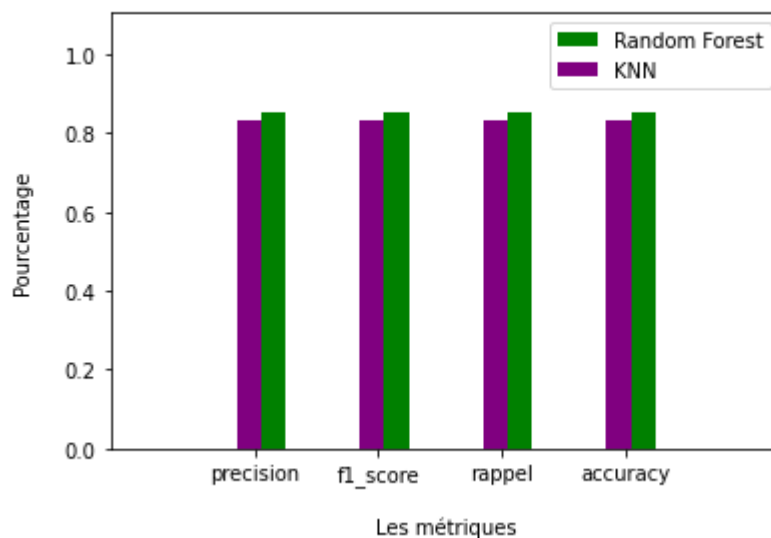
```
barWidth = 0.2
figsize = (10,6)
y1 = [precision_Score2,F1_score2,recall2,accuracy2]
y2 = [precision_Score22,F1_score22,recall22,accuracy22]
r1 = range(len(y1))
r2 = [x + barWidth for x in r1]

plt.bar(r2, y2, width = barWidth, color = ['green' for i in y1],
        linewidth = 4)
plt.bar(r1, y1, width = barWidth, color = ['purple' for i in y1],
        linewidth = 2)

plt.xticks([r + barWidth / 4 for r in range(len(y1))], ['precision', 'f1_score', 'rappel', 'accuracy'])

plt.title("Histogrammes groupés des métriques d'évaluation pour KNN, Random Forest\n")
plt.margins(0.3)
plt.xlabel('\nLes métriques')
plt.ylabel('Pourcentage\n')
plt.legend(["Random Forest", "KNN"])
```

Histogrammes groupés des métriques d'évaluation pour KNN, Random Forest



IV. Classification avec Fuzzy Rough Set:

1. Séparation du jeu de données :

```
# SÉPARATION EN DEUX JEUX DE DONNÉES avec un ratio de 1/5
X_train3, X_test3, y_train3, y_test3 = train_test_split(x_F, y, test_size=0.2, random_state=42, stratify=y)
```

2. Algorithme KNN :

```
#Appliquez l'algorithme des k plus proches voisins, pour chaque valeur de k, de sorte a trouver le meilleur score.

scoreList = []

k = range(1,20)

for i in k:
    knn = KNeighborsClassifier(n_neighbors = i) # n_neighbors means k
    knn.fit(X_train3, y_train3)
    scoreList.append(knn.score(X_test3, y_test3))

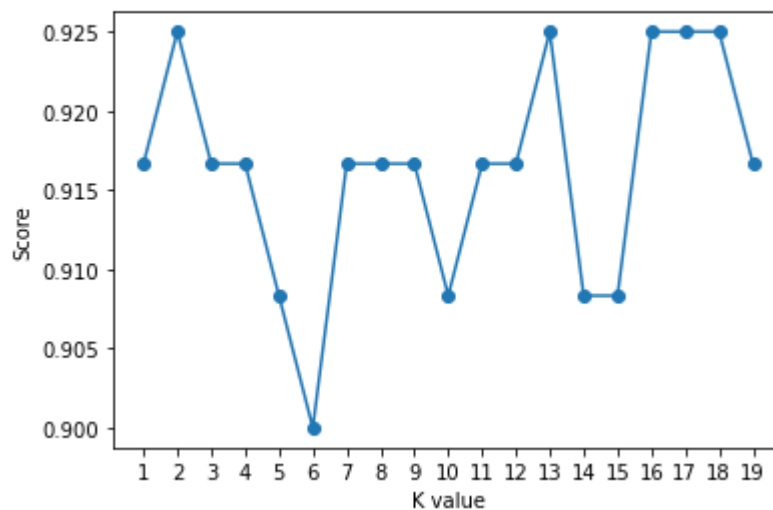
print("Maximum KNN Score is {:.2f}%".format(max(scoreList) * 100))

for i in range(len(scoreList)):
    if(max(scoreList) == scoreList[i]):
        top_k = i+1

knn = KNeighborsClassifier(n_neighbors = top_k) # n_neighbors means k
knn.fit(X_train3, y_train3)
y_pred_knn3 = knn.predict(X_test3)

plt.plot(range(1,20), scoreList, 'o-')
plt.xticks(np.arange(1,20,1))
plt.xlabel("K value")
plt.ylabel("Score")
plt.show()
```

Maximum KNN Score is 92.50%



3. Calcul du taux d'erreur :

```
#Calcul du taux d'erreur :

knn = KNeighborsClassifier(n_neighbors = 1)
knn.fit(X_train3, y_train3)
#Le taux de précision
y_test_pred3 = knn.predict(X_test3)

#taux d'erreur
score = 1 - accuracy_score(y_test3, y_test_pred3)
score

0.08333333333333337
```

4. Matrice classification report :

```
#Metric2 : CLASSIFICATION REPORT
m = classification_report(y_test3,y_test_pred3)
print("Knn")
print (m)
```

```
Knn
              precision    recall  f1-score   support

         1         0.93      0.90      0.92         60
         2         0.90      0.93      0.92         60

 accuracy          0.92
 macro avg         0.92      0.92      0.92
 weighted avg      0.92      0.92      0.92
```

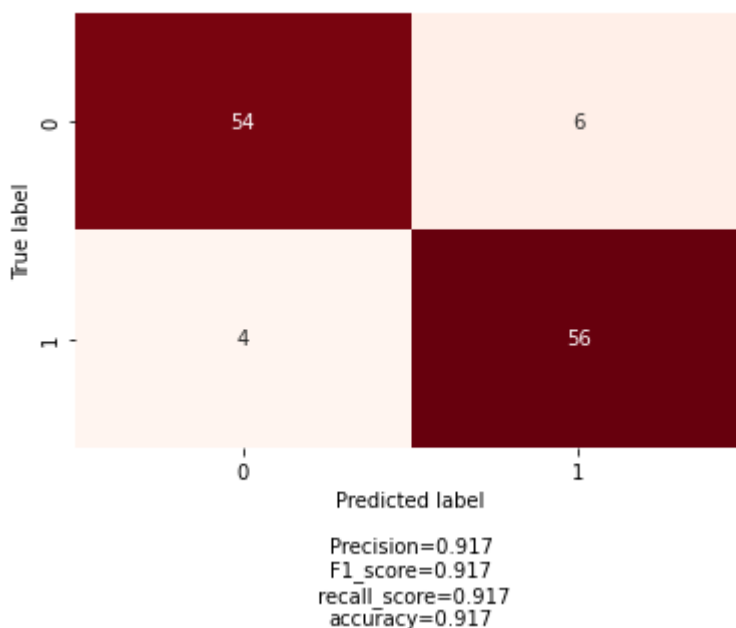
5. Les métriques :

```
# Affichage des resultats (Les valeurs des métriques)

# Calcul les valeurs des métriques
precision_Score3 = precision_score(y_test3, y_test_pred3, average='micro')
F1_score3 = f1_score(y_test3, y_test_pred3, average='micro')
recall3 = recall_score(y_test3, y_test_pred3, average='micro')
accuracy3 = accuracy_score(y_test3, y_test_pred3)

# Affichage de La matrice de confusion
result = confusion_matrix(y_test3, y_test_pred3)
sns.heatmap(result, annot=True,cmap='Reds',cbar=False)

# Affichage les valeurs des métriques
stats_text = "\n\nPrecision={:0.3f}\nF1_score={:0.3f}\n recall_score={:0.3f}\naccuracy={:0.3f}".format(precision_Score3,F1_score3,recall3,accuracy3)
plt.ylabel('True label')
plt.xlabel('Predicted label' + stats_text)
```



6. RANDOM FOREST :

```
# Different RandomForestClassifier hyperparameters
rf_grid = {"n_estimators": np.arange(10, 1000, 50),
           "max_depth": [None, 3, 5, 10],
           "min_samples_split": np.arange(2, 20, 2),
           "min_samples_leaf": np.arange(1, 20, 2)}
```

```
# Setup random seed
from sklearn.model_selection import GridSearchCV, cross_validate, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
np.random.seed(42)

# Setup random hyperparameter search for RandomForestClassifier
rs_rf3 = RandomizedSearchCV(RandomForestClassifier(),
                             param_distributions=rf_grid,
                             cv=5,
                             n_iter=20,
                             verbose=True)

# Fit random hyperparameter search model
rs_rf3.fit(X_train3, y_train3);
```

Fitting 5 folds for each of 20 candidates, totalling 100 fits

7. Meilleurs paramètres :

```
rs_rf3.best_params_
{'n_estimators': 510,
 'min_samples_split': 14,
 'min_samples_leaf': 1,
 'max_depth': None}
```

8. Score du modèle :

```
rs_rf3.score(X_test3, y_test3)
```

0.9416666666666667

```
y_test_pred3 = rs_rf3.predict(X_test3)
```

9. Matrice classification report :

```
m = classification_report(y_test3, y_test_pred3)
print("Random Forest")
print(m)
```

```
Random Forest
              precision    recall  f1-score   support

     1         0.95        0.93        0.94         60
     2         0.93        0.95        0.94         60

 accuracy          0.94
 macro avg          0.94
weighted avg          0.94
```

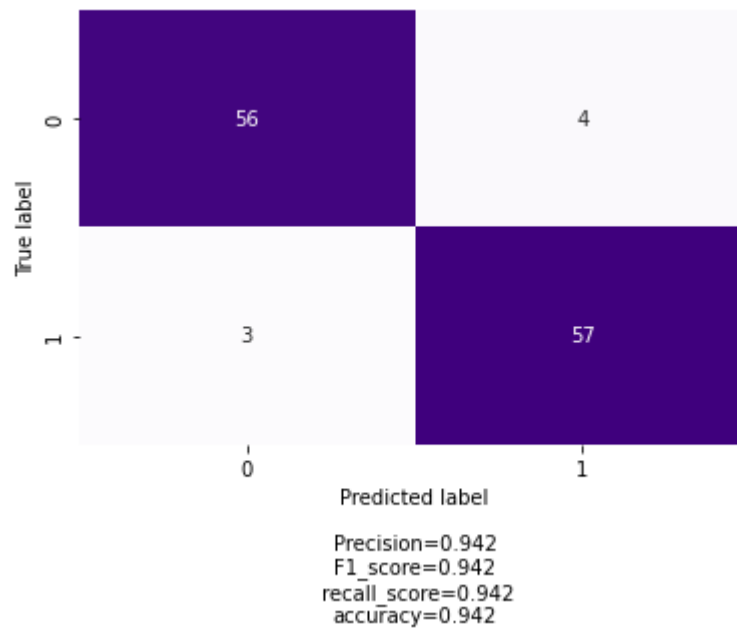
10. Les métriques :

```
# Affichage des resultats (Les valeurs des métriques)

# Calcul les valeurs des métriques
precision_Score33 = precision_score(y_test3, y_test_pred3, average='micro')
F1_score33 = f1_score(y_test3, y_test_pred3, average='micro')
recall33 = recall_score(y_test3, y_test_pred3, average='micro')
accuracy33 = accuracy_score(y_test3, y_test_pred3)

# Affichage de la matrice de confusion
result = confusion_matrix(y_test3, y_test_pred3)
sns.heatmap(result, annot=True, cmap='Purples', cbar=False)

# Affichage les valeurs des métriques
stats_text = "\n\nPrecision={:0.3f}\nF1_score={:0.3f}\n recall_score={:0.3f}\naccuracy={:0.3f}".format(precision_Score33, F1_score33, recall33, accuracy33)
plt.ylabel('True label')
plt.xlabel('Predicted label' + stats_text)
```



11. Roc Curve :

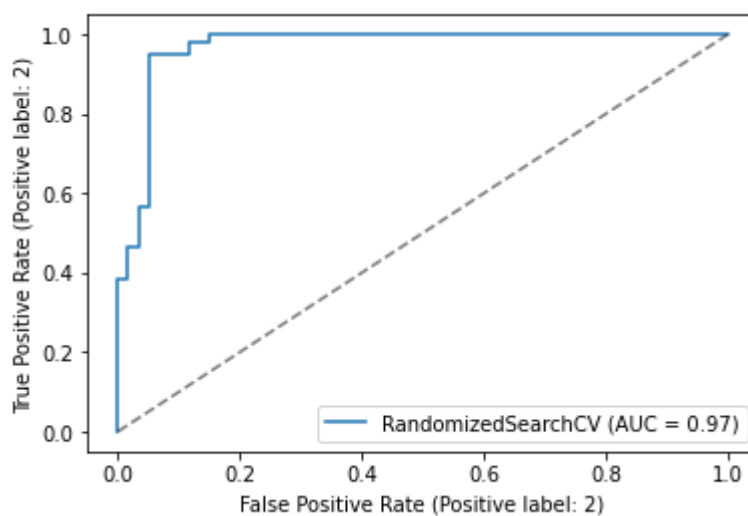
```
import matplotlib.pyplot as plt

# Plot ROC curve and calculate AUC metric
rfc_disp3 = plot_roc_curve(rs_rf3, X_test3, y_test3)

# Add diagonal line to the plot
rfc_disp3.ax_.plot([0, 1], [0, 1], linestyle='--', color='black', alpha=.5)

# Set title and legend
rfc_disp3.figure_.suptitle("ROC curve for Random Forest")
plt.legend(loc='lower right')

# Show the plot
plt.show()
```



12. Comparaison en termes de performance de prédiction entre KNN et Random Forest :

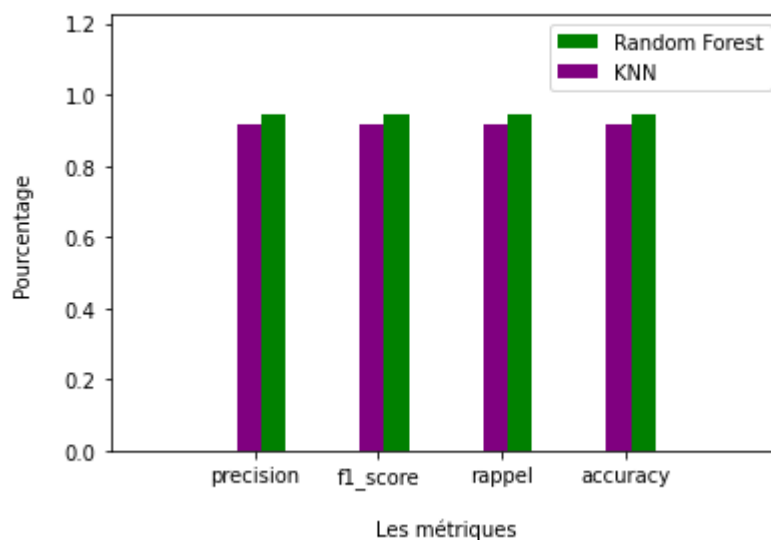
```
barWidth = 0.2
figsize = (10,6)
y1 = [precision_Score3,F1_score3,recall3,accuracy3]
y2 = [precision_Score33,F1_score33,recall33,accuracy33]
r1 = range(len(y1))
r2 = [x + barWidth for x in r1]

plt.bar(r2, y2, width = barWidth, color = ['green' for i in y1],
        linewidth = 4)
plt.bar(r1, y1, width = barWidth, color = ['purple' for i in y1],
        linewidth = 2)

plt.xticks([r + barWidth / 4 for r in range(len(y1))], ['precision', 'f1_score', 'rappel', 'accuracy'])

plt.title("Histogrammes groupés des métriques d'évaluation pour KNN, Random Forest\n")
plt.margins(0.3)
plt.xlabel('\nLes métriques')
plt.ylabel('Pourcentage\n')
plt.legend(["Random Forest", "KNN"])
```

Histogrammes groupés des métriques d'évaluation pour KNN, Random Forest



V. Classification avec Pearson Corrélation:

1. Algorithme KNN :

```
#Appliquez l'algorithme des k plus proches voisins, pour chaque valeur de k, de sorte a trouver le meilleur score.
scoreList = []
k = range(1,20)

for i in k:
    knn = KNeighborsClassifier(n_neighbors = i) # n_neighbors means k
    knn.fit(X_train4, y_train4)
    scoreList.append(knn.score(X_test4, y_test4))

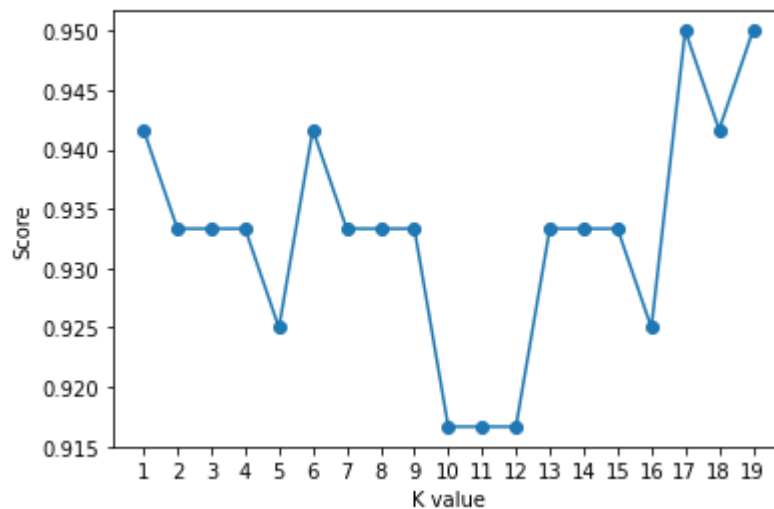
print("Maximum KNN Score is {:.2f}%".format(max(scoreList) * 100))

for i in range(len(scoreList)):
    if(max(scoreList) == scoreList[i]):
        top_k = i+1

knn = KNeighborsClassifier(n_neighbors = top_k) # n_neighbors means k
knn.fit(X_train4, y_train4)
y_pred_knn4 = knn.predict(X_test4)

plt.plot(range(1,20), scoreList, 'o-')
plt.xticks(np.arange(1,20,1))
plt.xlabel("K value")
plt.ylabel("Score")
plt.show()
```


Maximum KNN Score is 95.00%



2. Calcul du taux d'erreur :

```
#Calcul du taux d'erreur :  
  
knn = KNeighborsClassifier(n_neighbors = 1)  
knn.fit(X_train4,y_train4)  
#Le taux de précision  
y_test_pred4= knn.predict(X_test4)  
  
#taux d'erreur  
score = 1 - accuracy_score( y_test4, y_test_pred4)  
score
```

0.05833333333333335

3. Matrice classification report :

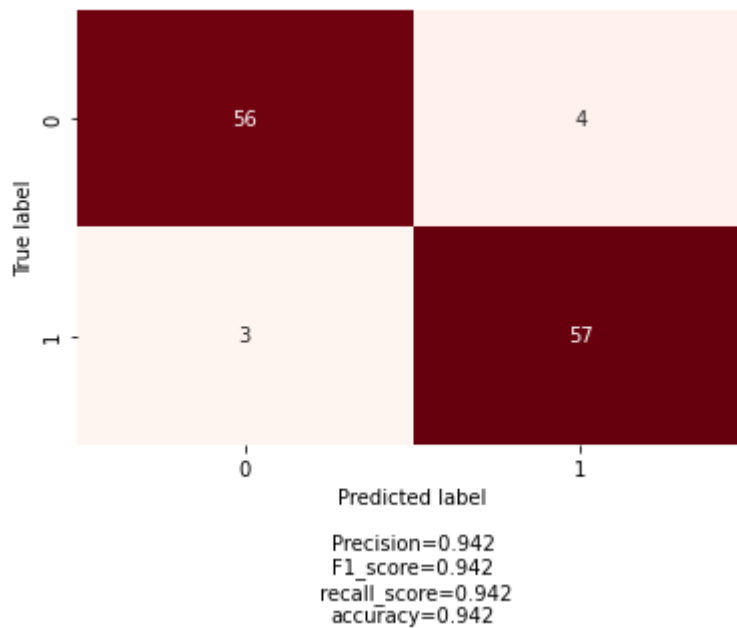
```
#Metric2 : CLASSIFICATION REPORT  
m = classification_report(y_test4,y_test_pred4 )  
print("Knn")  
print (m)
```

```
Knn
```

	precision	recall	f1-score	support
1	0.95	0.93	0.94	60
2	0.93	0.95	0.94	60
accuracy			0.94	120
macro avg	0.94	0.94	0.94	120
weighted avg	0.94	0.94	0.94	120

4. Les métriques :

```
# Affichage des resultats (Les valeurs des métriques)  
  
# Calcul Les valeurs des métriques  
precision_Score4 = precision_score(y_test4, y_test_pred4, average='micro')  
F1_score4 = f1_score(y_test4, y_test_pred4, average='micro')  
recall4 = recall_score(y_test4, y_test_pred4, average='micro')  
accuracy4 = accuracy_score(y_test4, y_test_pred4)  
  
# Affichage de La matrice de confusion  
result = confusion_matrix(y_test4, y_test_pred4)  
sns.heatmap(result, annot=True,cmap='Reds',cbar=False)  
  
# Affichage Les valeurs des métriques  
stats_text = "\n\nPrecision={:0.3f}\nF1_score={:0.3f}\n recall_score={:0.3f}\naccuracy={:0.3f}".format(precision_Score4,F1_score4,recall4,accuracy4)  
plt.ylabel('True label')  
plt.xlabel('Predicted label' + stats_text)
```



5. RANDOM FOREST :

```
# Different RandomForestClassifier hyperparameters
rf_grid = {"n_estimators": np.arange(10, 1000, 50),
           "max_depth": [None, 3, 5, 10],
           "min_samples_split": np.arange(2, 20, 2),
           "min_samples_leaf": np.arange(1, 20, 2)}
```

```
# Setup random seed
from sklearn.model_selection import GridSearchCV, cross_validate, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
np.random.seed(42)

# Setup random hyperparameter search for RandomForestClassifier
rs_rf4 = RandomizedSearchCV(RandomForestClassifier(),
                             param_distributions=rf_grid,
                             cv=5,
                             n_iter=20,
                             verbose=True)

# Fit random hyperparameter search model
rs_rf4.fit(X_train4, y_train4);
```

Fitting 5 folds for each of 20 candidates, totalling 100 fits

6. Meilleurs paramètres :

```
rs_rf4.best_params_
{'n_estimators': 310,
 'min_samples_split': 12,
 'min_samples_leaf': 5,
 'max_depth': None}
```

7. Score du modèle :

```
rs_rf4.score(X_test4, y_test4)
```

0.975

```
y_test_pred4 = rs_rf4.predict(X_test4)
```

8. Matrice classification report :

```
m = classification_report(y_test4,y_test_pred4 )
print("Random Forest")
print (m)
```

```
Random Forest
              precision    recall  f1-score   support

         1         1.00      0.95      0.97         60
         2         0.95      1.00      0.98         60

 accuracy          0.97
 macro avg          0.97
weighted avg          0.97
```

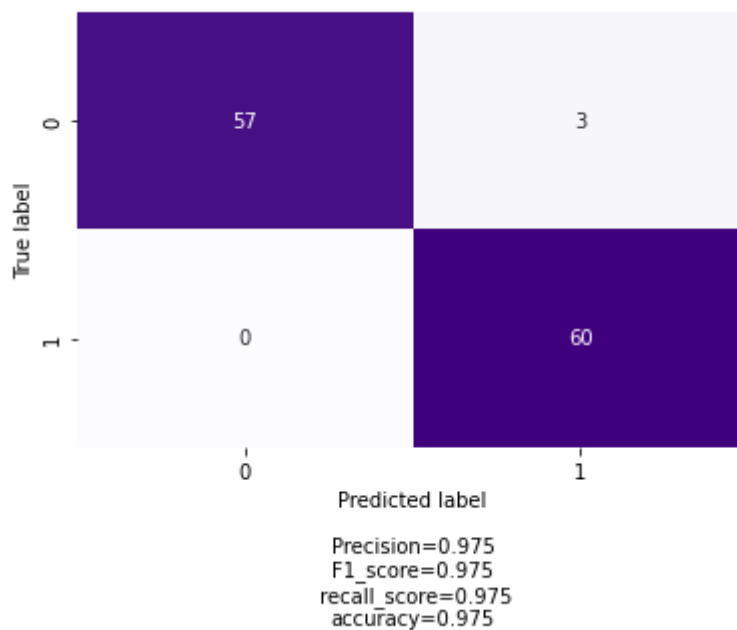
9. Les métriques :

```
# Affichage des resultats (Les valeurs des métriques)

# Calcul Les valeurs des métriques
precision_Score44 = precision_score(y_test4, y_test_pred4, average='micro')
F1_score44 = f1_score(y_test4, y_test_pred4, average='micro')
recall44 = recall_score(y_test4, y_test_pred4, average='micro')
accuracy44 = accuracy_score(y_test4, y_test_pred4)

# Affichage de La matrice de confusion
result = confusion_matrix(y_test4, y_test_pred4)
sns.heatmap(result, annot=True,cmap='Purples',cbar=False)

# Affichage Les valeurs des métriques
stats_text = "\n\nPrecision={:0.3f}\nF1_score={:0.3f}\n recall_score={:0.3f}\naccuracy={:0.3f}".format(precision_Score44,F1_score44,recall44,accuracy44)
plt.ylabel('True label')
plt.xlabel('Predicted label' + stats_text)
```



10. Roc Curve :

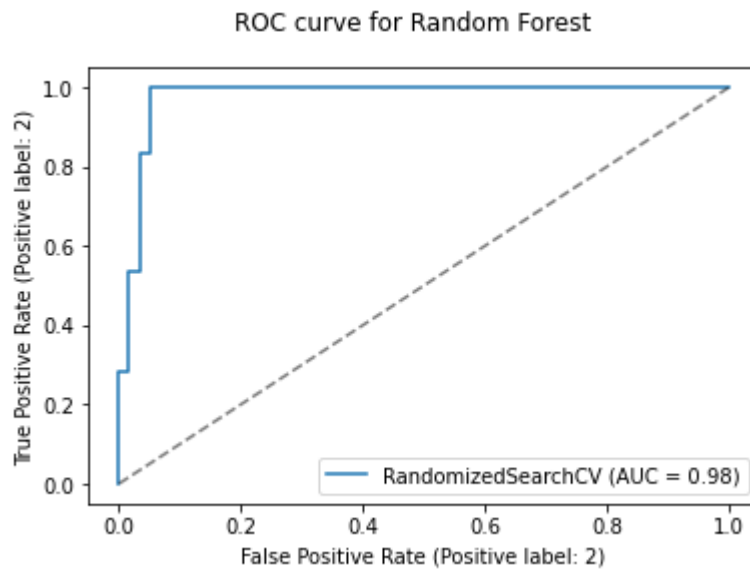
```
import matplotlib.pyplot as plt

# Plot ROC curve and calculate AUC metric
rfc_disp4 = plot_roc_curve(rs_rf4, X_test4, y_test4)

# Add diagonal Line to the plot
rfc_disp4.ax_.plot([0, 1], [0, 1], linestyle='--', color='black', alpha=.5)

# Set title and legend
rfc_disp4.figure_.suptitle("ROC curve for Random Forest")
plt.legend(loc='lower right')

# Show the plot
plt.show()
```



11. Comparaison en termes de performance de prédiction entre KNN et Random Forest :

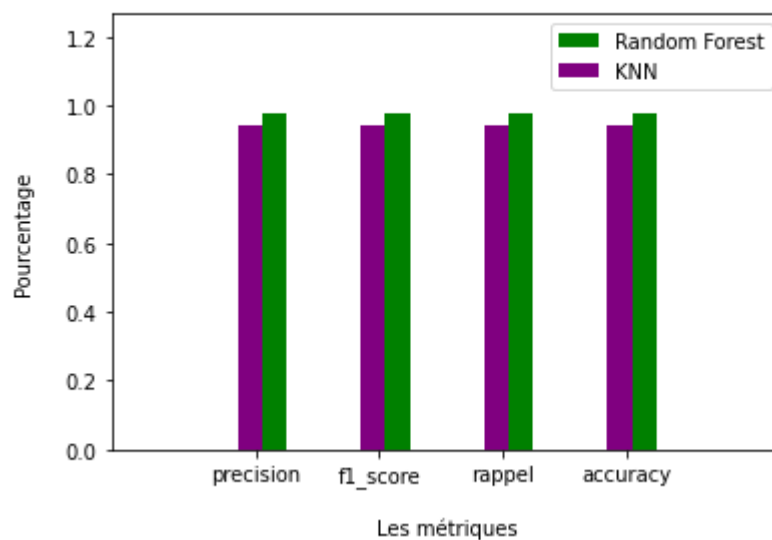
```
barWidth = 0.2
figsize = (10,6)
y1 = [precision_Score4,F1_score4,recall4,accuracy4]
y2 = [precision_Score44,F1_score44,recall44,accuracy44]
r1 = range(len(y1))
r2 = [x + barWidth for x in r1]

plt.bar(r2, y2, width = barWidth, color = ['green' for i in y1],
        linewidth = 4)
plt.bar(r1, y1, width = barWidth, color = ['purple' for i in y1],
        linewidth = 2)

plt.xticks([r + barWidth / 4 for r in range(len(y1))], ['precision', 'f1_score', 'rappel', 'accuracy'])

plt.title("Histogrammes groupés des métriques d'évaluation pour KNN, Random Forest\n")
plt.margins(0.3)
plt.xlabel('\nLes métriques')
plt.ylabel('Pourcentage\n')
plt.legend(["Random Forest", "KNN"])
```

Histogrammes groupés des métriques d'évaluation pour KNN, Random Forest



Synthèse :

Dans les différentes comparaisons qu'on a effectué sur les ensembles de données si dessus, on remarque bien que Random Forest a été plus performant sur toutes les métriques que KNN, sauf pour RST et cela revient :

Robustesse aux données bruitées : Random Forest est plus robuste aux données bruitées et peut toujours fournir des résultats cohérents malgré la présence de données aberrantes. KNN, en revanche, peut être fortement affecté par les données aberrantes. Plusieurs arbres sont utilisés pour faire des prévisions.

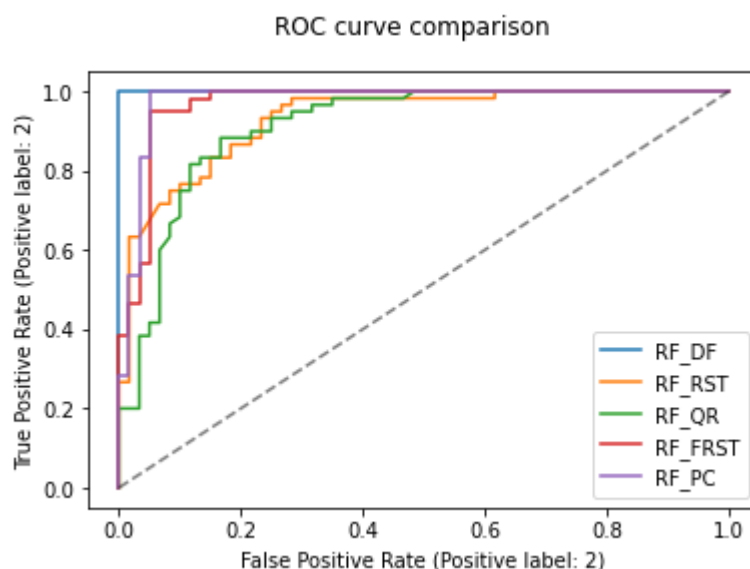
Pour le cas de RST, KNN a été de même performance de prédiction que Random Forest et cela revient à la simplicité et la rapidité du modèle. Aussi ce modelé KNN est considéré plus adapté pour les cas où les données d'entraînement sont très détaillées.

➤ **Comparaison des Algorithmes :**

1. Random Forest :

Dans cette première partie de l'étape de comparaison nous allons faire la comparaison entre les résultats obtenu après application de Random Forest sur les features sélectionnes par les différents Algorithmes de features Sélection.

```
rfc_disp0 = plot_roc_curve(rs_rf0, X_test0, y_test0)
rfc_disp1 = metrics.plot_roc_curve(rs_rf1, X_test1, y_test1, ax=rfc_disp0.ax_)
rfc_disp2 = metrics.plot_roc_curve(rs_rf2, X_test2, y_test2, ax=rfc_disp0.ax_)
rfc_disp3 = metrics.plot_roc_curve(rs_rf3, X_test3, y_test3, ax=rfc_disp0.ax_)
rfc_disp4 = metrics.plot_roc_curve(rs_rf4, X_test4, y_test4, ax=rfc_disp0.ax_)
rfc_disp0.ax_.plot([0, 1], [0, 1], linestyle='--', color='black', alpha=.5)
rfc_disp1.figure_.suptitle("ROC curve comparison")
plt.legend(["RF_DF", "RF_RST", "RF_QR", "RF_FRST", "RF_PC"])
```



D'après la figure si dessus en remarque bien que la performance parfaite soit obtenu avec le dataset initial dont on a tous les feature. En effet les résultats de prédiction de Random Forest appliqué sur "Paerson Correlation" sont meilleur que tous les autres et beaucoup proche des résultats du dataset initial, puis en a en deuxième position le RF_FRST, et enfin RF_RST avec une tous petite différence avec QuickReduct (donc en peut dire qu'ils sont au même niveau).

Cela revient à la qualité des reduct générées par chaque Algorithme de feature sélection.

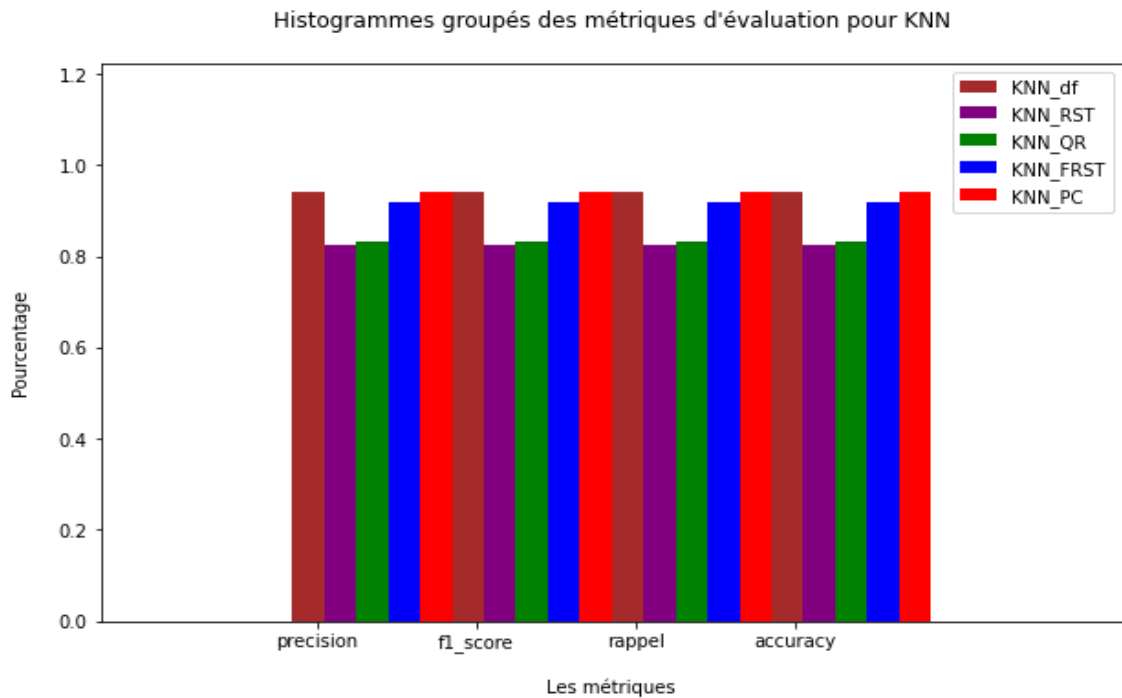
Dans cette deuxième partie de l'étape de comparaison nous allons faire la comparaison entre les résultats obtenu après application de KNN sur les features sélectionnes par les différents Algorithmes de features Selection

2. KNN :

D'après la figure qui suit en voit bien que la différence en terme de performance de prédiction entre notre classifieur KNN appliqué sur chaque reduct est très petite. Le plus performant entre eux en termes d'évaluation des métriques c'est le KNN du dataset initial, presque au même niveau on a KNN de Pearson Correlation, puis c'est KNN_FRST suivi et en dernière position on trouve KNN QuickReduct qui presque a la même performance de prédiction que KNN RST.

Cela peut être du a la qualité des reduct produit par les différents algorithmes de feature selection ainsi au comportement de l'algorithme KNN dans le processus de traitement des données de chaque Reduct.

```
barWidth = 0.2
plt.figure(figsize=(10,6))
y0 = [precision_Score0,F1_score0,recall0,accuracy0]
y1 = [precision_Score1,F1_score1,recall1,accuracy1]
y2 = [precision_Score2,F1_score2,recall2,accuracy2]
y3 = [precision_Score3,F1_score3,recall3,accuracy3]
y4 = [precision_Score4,F1_score4,recall4,accuracy4]
r0 = range(len(y0))
r1 = [x + barWidth for x in r0]
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]
r4 = [x + barWidth for x in r3]
plt.bar(r0, y0, width = barWidth, color = ['brown' for i in y0],
        linewidth = 2)
plt.bar(r1, y1, width = barWidth, color = ['purple' for i in y0],
        linewidth = 2)
plt.bar(r2, y2, width = barWidth, color = ['green' for i in y1],
        linewidth = 4)
plt.bar(r3, y3, width = barWidth, color = ['blue' for i in y2],
        linewidth = 2)
plt.bar(r4, y4, width = barWidth, color = ['red' for i in y3],
        linewidth = 2)
plt.xticks([r + barWidth / 4 for r in range(len(y1))], ['precision', 'f1_score', 'rappel', 'accuracy'])
plt.title("Histogrammes groupés des métriques d'évaluation pour KNN\n")
plt.margins(0.3)
plt.xlabel('\nLes métriques')
plt.ylabel('Pourcentage\n')
plt.legend(["KNN_df", "KNN_RST", "KNN_QR", "KNN_FRST", "KNN_PC"])
```



Conclusion :

En conclusion, nous avons exploré différentes méthodes de feature selection, à savoir rough set, quick reduct, fuzzy rough set et Pearson correlation. De plus, nous avons utilisé deux algorithmes de machine Learning, à savoir KNN et Random Forest, pour évaluer les réductions de features générées par chaque algorithme de feature selection. Nous avons utilisé des métriques d'évaluation telles que F1-score, Recall, Precision, roc curve pour évaluer les performances de chaque méthode de feature selection.

Nous avons constaté que quick reduct a montré des performances qui ne sont pas très pertinentes vis-à-vis des autres algorithmes de feature selection utilisés, mais reste toujours des bonnes performances d'un point de vue global. Parmi ses limites, on constate qu'il peut être très intensif en calcul, surtout lorsque le nombre d'attributs dans l'ensemble de données est élevé. Aussi sa sensibilité aux valeurs aberrantes dans l'ensemble de données, ce qui peut affecter les résultats de la classification.

Cet algorithme dépend de l'ensemble d'approximation initial, qui peut être influencé par des facteurs externes tels que le choix des attributs ou la méthode de classification utilisée.

Certaines méthodes peuvent mieux fonctionner avec certaines méthodes de classification que d'autres.

Pour améliorer sa capacité de sélection et sa performance, différentes méthodes existent, parmi ces dernières on a :

- L'utilisation des mesures d'évaluation plus avancées : QuickReduct utilise des mesures d'évaluation telles que l'entropie et l'importance des attributs pour identifier les attributs pertinents. Cependant, il existe d'autres mesures d'évaluation plus avancées telles que l'indice de stabilité et l'indice de redondance qui peuvent être utilisées pour améliorer la sélection des attributs pertinents.
- L'utilisation des techniques d'apprentissage non supervisées : QuickReduct utilise une approche de sélection de features supervisée, ce qui signifie qu'elle utilise des étiquettes de classe pour identifier les attributs pertinents. Cependant, les techniques d'apprentissage non supervisées telles que l'analyse en composantes principales (PCA) peuvent également être utilisées pour identifier les attributs pertinents en se basant uniquement sur la structure des données.
- L'utilisation d'une cross-validation avec QuickReduct peut améliorer la performance de l'algorithme en permettant une validation plus robuste des résultats de la sélection de caractéristiques.
- L'utilisation de lasso regression (Algorithme des Wrapper) en conjonction avec QuickReduct peut renforcer la sélection de caractéristiques en réduisant le nombre de caractéristiques redondantes ou non significatives dans le modèle final. Cette approche peut également améliorer l'interprétabilité du modèle final en fournissant un ensemble plus restreint de caractéristiques sélectionnées.

Malgré ces limites, l'algorithme QuickReduct est largement utilisé dans la théorie des ensembles approximatifs en raison de sa capacité à réduire la complexité des ensembles de données.

CONCLUSION GENERALE

CONCLUSION GÉNÉRALE :

Pour conclure ce projet, cette étude a exploré différentes méthodes de sélection de caractéristiques, à savoir Rough Set, QuickReduct, Fuzzy Rough Set et Pearson Correlation, en utilisant deux algorithmes de machine learning, KNN et Random Forest, pour évaluer les réductions de caractéristiques générées par chaque méthode. Les métriques d'évaluation telles que F1-score, Recall, Precision, ROC curve ont été utilisées pour comparer les performances de chaque algorithme.

Les résultats ont montré que chaque méthode a ses avantages et ses limites. Pearson Correlation a donné de meilleurs résultats que d'autres méthodes en termes de précision et de temps de calcul, il peut être utile pour identifier les caractéristiques linéairement corrélées, mais peut ne pas être efficace pour identifier les relations non linéaires entre les caractéristiques. Fuzzy Rough Set a été efficace pour traiter les données imprécises ou manquantes, mais peut être plus coûteux en termes de temps de calcul. Enfin, QuickReduct a donné de résultats qu'on peut considérer bon, mais peut être sensible aux caractéristiques redondantes ou non significatives.

Pour améliorer la performance de QuickReduct, une approche suggérée est d'utiliser une cross-validation et de le renforcer avec un autre algorithme tel que Lasso Regression. Cela peut améliorer la robustesse et l'interprétabilité des résultats de la sélection de caractéristiques.

Dans l'ensemble, cette étude montre qu'il n'y a pas de méthode de sélection de caractéristiques universelle qui fonctionne mieux pour toutes les tâches et que le choix de la méthode dépend des caractéristiques des données et de la tâche à accomplir. Les résultats peuvent aider les praticiens à choisir la méthode de sélection de caractéristiques la plus appropriée pour leurs données et leur tâche.

Liens/Référence :

- [Https ://scikit-learn.org/](https://scikit-learn.org/)
- [Https ://datascientest.com/i](https://datascientest.com/i)
- [Https ://medium.com/search](https://medium.com/search)
- [https ://programminghistorian.org/fr](https://programminghistorian.org/fr)
- [https ://realpython.com/](https://realpython.com/)
- <https://stackoverflow.com/>
- <https://github.com/search?q=rough+set>