

Test d'homogénéité de 9 moyennes - étude du prix moyen au mètre carré des 9 arrondissements de la ville de Lyon.

En analyse de données il est courant de comparer différents groupes notamment d'après un paramètre statistique. Généralement, on s'intéresse à comparer ces groupes d'après leur moyenne. On se pose la question de savoir si les différences de moyennes observées entre nos groupes sont le fait de simples fluctuations d'échantillonnages ou bien si elles reflètent une tendance réelle. En d'autres termes, on se demande si les différences observées sont généralisable aux populations parentes. Par exemple, lorsqu'on s'intéresse à comparer les prix des loyers entre différents arrondissements on va sélectionner un échantillon représentatif du marché immobilier pour chaque arrondissement. On observe alors que le prix moyen au mètre carré est plus élevé dans l'un des arrondissements. On veut donc savoir si cette différence n'est pas simplement le fait du hasard - i.e. la conséquence de notre sélection - mais qu'elle reflète une tendance réelle du marché. On veut donc répondre à la question suivante : les populations parentes de nos sous-échantillons sont-elles différentes d'après un paramètre de tendance centrale tel que la moyenne ?

Afin d'illustrer la mise en place d'un test d'homogénéité de moyenne pour plus de 2 sous-échantillons, nous utiliserons les données issues d'un célèbre site d'annonces immobilières. J'ai extrait les données de 207 annonces immobilières à Lyon. On va donc étudier les différences de prix moyen au mètre carré pour chaque arrondissement de Lyon - 9 arrondissements. Puisque le nombre de groupes (sous-échantillons) est supérieur à 2 on doit utiliser une analyse de la variance (ANOVA). Toutefois, cette technique statistique repose sur des conditions de validité des résultats : normalité, homoscedasticité et indépendance des résidus. Si ces conditions ne sont pas rencontrées, alors il faudra employer soit une Anova de Welch, soit un test de Kruskal-Wallis.

Avant d'étudier la significativité statistique des différences de moyenne de nos 9 sous-échantillons, nous allons inspecter et nettoyer la base de données. Puis, nous vérifierons les conditions d'application de l'anova. Le cas échéant, nous choisirons d'adopter l'une ou l'autre des méthodes. Enfin, nous présenterons la réalisation de test post-hoc : ces tests permettent de savoir quels groupes présentent des différences significatives de moyenne.

```
In [1]: import pandas as pd  
import numpy as np
```

```
In [2]: df = pd.read_csv('loyer_lyon.csv')
```

```
In [3]: display(df.head(), df.shape)
```

	housing_type	price	nbr_rooms	area	location	description
0	Studio	456	1	22.0	69008	Secteur Moulin à vent. A louer, chambre indivi...
1	Studio	426	1	21.0	69008	Secteur Moulin à vent. A louer, chambre indivi...
2	Appartement	1115	3	84.0	69003	Exclusivité ORPI - Lyon 3ème - 219 Avenue Féli...
3	Appartement	899	2	60.0	69008	LYON 8 Secteur Grand Trou, proche des commerce...
4	Appartement	1252	3	78.0	69007	LYON 7EME / résidence « LE 372 » rue Garibaldi...

(207, 6)

```
In [4]: df.drop_duplicates(keep='first', inplace=True, ignore_index=True)
```

Notre ensemble de données comporte 207 observations et 6 variables. Voyons comment se présentent ces variables. On va utiliser **skimpy** une bibliothèque plutôt sympa pour jeter un rapide coup d'oeil aux caractéristiques statistiques de nos variables.

```
In [5]: from skimpy import skim  
skim(df)
```





Data Summary

dataframe	Values
Number of rows	176
Number of columns	6

Data Types

Column Type	Count
int64	3
string	2
float64	1

number

column_name	NA	NA %	mean	sd	p0	p25	p50	p75	p100	hist
price	0	0	1113	588.1	280	707.8	960	1345	4000	
nbr_rooms	0	0	2.619	1.264	1	2	3	3	6	
area	9	5.11	63.77	32.62	12	42.5	60	82.5	178	
location	0	0	69000	2.721	69000	69000	69000	69010	69010	

string

column_name	NA	NA %	words per row	total words
housing_type	0	0	1	
description	0	0	86	1!

End

Puisque notre étude concerne le prix moyen au mètre carré par arrondissement on va se concentrer sur l'analyse des variables : *prix* (price), *surface* (area) et *arrondissement* (location). Les deux premières vont nous permettre de construire la variable prix au mètre carré.

Concernant la variable *prix*, la distribution est asymétrique à droite (la moyenne est supérieure à la médiane) et très dispersée (écart-type de 595.6). Néanmoins, elle ne présente pas de valeurs manquantes.

La variable *surface* présente 9 valeurs manquantes et sa distribution est également asymétrique à droite.

Enfin, la variable *arrondissement* est encodé comme une variable numérique alors qu'il s'agit d'une variable catégorielle.

Au sujet des 9 valeurs manquantes : il n'est pas possible de remplacer ces valeurs manquantes par zéro (ou toute autre valeur) puisqu'une surface ne peut pas être nulle. Nous allons donc supprimer ces observations de notre base de données. Puis nous pourrions créer la

variable *prix au mètre carré*. Pour ce qui est de la variable *arrondissement* nous devons la retyper pour poursuivre notre analyse.

```
In [6]: df.dropna(inplace=True, ignore_index=True)
```

On se retrouve finalement avec 198 observations. On va changer type de la variable *arrondissement*.

```
In [7]: df['location'] = df['location'].astype(str).astype('category')
```

Voyons comment sont répartis les individus au sein de chaque catégorie.

```
In [8]: df.location.value_counts()
```

```
Out[8]: location
69003      31
69008      29
69001      20
69002      19
69007      17
69006      16
69009      15
69005      11
69004       8
69000       1
Name: count, dtype: int64
```

On a seulement une observation pour la catégorie *69000*, il faut donc la retirer de notre base de données : cela n'apportera pas d'information et on ne pourra pas faire d'analyse sur un seul individu statistique. Notez que ce code postal est attribué aux annonces immobilières des appartements situés "autour de lyon".

```
In [9]: df.drop(index=df[df['location']=="69000"].index, inplace=True)
```

On se retrouve finalement avec 197 observations. On peut maintenant passer à la définition de notre variable dépendante *prix au mètre carré*. Formellement, elle se définit comme le rapport du prix et de la surface.

```
In [10]: df['prix_m2'] = df['price']/df['area']
df.prix_m2.describe()
```

```
Out[10]: count    166.000000
         mean     18.822601
         std      6.602210
         min      5.515789
         25%     14.726891
         50%     17.423086
         75%     21.452492
         max     41.250000
         Name: prix_m2, dtype: float64
```

La variable *prix au mètre carré* est asymétrique à droite (moyenne supérieure à la médiane). Analysons de plus près la distribution de la variable avec un histogramme.

```
In [11]: import plotly.express as px
         fig = px.histogram(df, x="prix_m2", log_y=False)
         fig.show()
```

Il y a effectivement quelques observations qui étire la queue supérieure de la distribution. Voyons si le critère de Tukey permet de révéler l'existence de points extrêmes (outliers).

```
In [12]: fig = px.box(df, y="prix_m2",)
         fig.show()
```

On peut effectivement suspecter la présence de points extrêmes relativement au reste de la distribution : ceux ayant un prix au mètre carré supérieur à 35€.

```
In [15]: # Test de Kolmogorov-Smirnov.
         # Je préfère utiliser un test d'adéquation non-paramétrique.
         from scipy.stats import kstest

         statistic, pvalue, *_ = kstest(df["prix_m2"], cdf="norm", alternative='two-sided')
         print(f"La statistique de test est {statistic}. La p-value est {pvalue}")
```

La statistique de test est 0.9999999826391145. La p-value est 0.0

D'après le test d'adéquation de Kolmogorov-Smirnov, avec un risque de première espèce de 5%, il y a peu de chance pour que nos données soient normalement distribuées. Je fais le choix de ne pas utiliser un test d'adéquation paramétrique (Shapiro-Wilk) car il serait peu

robuste à la présence d'outliers.

Nous ne pouvons donc pas appliquer le test de Grubbs pour détecter les outliers. Nous pouvons toutefois utiliser une version robuste du score Z.

```
In [16]: # Score Z modifié
from outliers_detection import robust_z_score

#help(robust_z_score)
scores = robust_z_score(variable="prix_m2", df=df)
for j, score in enumerate(scores):
    if np.abs(score) > 4 :
        print(df.loc[j, "prix_m2"])
```

```
39.75
39.588235294117645
41.25
23.88888888888889
```

On a 3 outliers : les valeurs 38.125, 39.75, 39.59 et 41.25 ont un score Z robuste supérieur à 4. Les valeurs dont le score est supérieur à 4 ont peu de chances d'être observées. Voyons à quel type de bien correspondent ces valeurs.

```
In [17]: df[df["prix_m2"] >= 38.125]
```

```
Out[17]:
```

	housing_type	price	nbr_rooms	area	location	description	prix_m2
51	Studio	636	1	16.0	69008	Monplaisir, rue Henri Pensier, dans un immeubl...	39.750000
59	Studio	673	1	17.0	69002	Appartement 1 pièce Lyon 2ème - A LOUER, LYON ...	39.588235
74	Studio	495	1	12.0	69003	Avant toute chose : oui le logement est dispon...	41.250000
148	Studio	610	1	16.0	69001	Location Lyon 1 libre le 13/09/24. Entre parti...	38.125000

Ces observations sont toutes des studios situés dans divers arrondissements de Lyon. La demande pour ces logements étant forte lors de la récolte de données (été 2024) il se peut que le tarif de ces logements ne soit pas représentatif du prix pour ce type de logement. Vérifions cela en graphique.

```
In [18]: fig = px.box(df[df["housing_type"]=="Studio"], y="prix_m2")
fig.show()
```

```
In [20]: from scipy.stats import shapiro
statistic, pvalue = shapiro(df[df["housing_type"]=="Studio"]["prix_m2"],)
print(statistic, pvalue)
```

0.9617986083030701 0.19289450347423553

La distribution du prix au mètre carré des studios présente une très légère asymétrie à droite mais pas d'outliers d'après le critère de Tukey. D'après le test de Shapiro-Wilk, on a peu d'évidence pour rejeter l'hypothèse nulle.

On peut donc conclure que ces prix ne sont pas aberrants (pas d'erreur de saisie). Toutefois, ce sont des points atypiques relativement aux prix des autres types de biens. Il ne faut donc pas les retirer de notre base de données. Ceci étant, leur présence peut biaiser notre analyse. De plus, puisque l'objectif est de savoir si le **prix moyen des biens au mètre carré diffère significativement d'un arrondissement à l'autre**, nous pouvons mener notre analyse avec ces points tout en restant attentif à leur impact sur les distributions des prix par arrondissement.

Analyse de la variance du prix au mètre carré par arrondissement.

L'analyse de la variance permet d'éprouver la relation entre un facteur qualitatif et un facteur quantitatif. Particulièrement, il permet de savoir si, pour un facteur qualitatif données, le facteur quantitatif présente en moyenne une différence statistiquement significative.

L'analyse de la variance repose sur l'étude du rapport des variances intra- et inter-classes. Elle repose donc sur un test F de Fisher.

Toutefois, il faut avant tout réaliser une régression linéaire simple. En effet, l'Anova est un modèle linéaire. Aussi, comme tout modèle linéaire, des hypothèses doivent être satisfaites pour s'assurer de la validité des résultats. Entre autres, on doit vérifier que les résidus sont à variance homogène et suivent une distribution Normale ; également, les résidus doivent être indépendants ce qui est bien souvent assuré par le design de l'enquête.

Vérifier les conditions d'applications de l'Anova : normalité, homoscedasticité, indépendance.

L'indépendance des données est évidente mais on peut s'en assurer avec un test de Durbin-Watson. Concernant l'homoscedasticité et la

normalité des données, une première inspection peut être faite via un boxplot du prix au mètre carré pour chaque arrondissement.

```
In [21]: import plotly.express as px

fig = px.box(df, y="prix_m2", x="location")
fig.show()
```

D'après ce graphique on peut faire trois remarques :

1. Certaines distributions présentent un outlier au niveau de la queue supérieure de la distribution ;
2. Les distributions ne sont pas normalement distribuées ;
3. Les variances ne sont pas homogènes.

Afin de vérifier les conclusions de notre analyse graphique on peut :

- réaliser un test de Grubbs - vérifier la significativité statistique des outliers ;
- réaliser un test de Shapiro-Wilk / Kolmogorov-Smirnov - tester l'adéquation des données à une distribution normale ;
- réaliser un test de Leven/ Bartlett - tester l'homogénéité des variances pour plus de 2 groupes (si l'hypothèse de normalité n'est pas satisfaite, je préfère utiliser le test de Levene).

Le test de Grubbs repose sur la normalité des données. On va donc utiliser un test d'adéquation avant de détecter les outliers. Vu l'asymétrie et l'existence de potentiels outliers (d'après le critère de Tukey), je privilégie un test d'adéquation non-paramétrique : le test de Kolmogorov-Smirnov.

```
In [28]: # Test de Kolmogorov-Smirnov.
# Je préfère utiliser un test d'adéquation non-paramétrique.
from scipy.stats import kstest

groups = df.groupby("location", observed=True)["prix_m2"].apply(list)

for group in groups:
    statistic, pvalue, *_ = kstest(group, cdf="norm", alternative='two-sided',)
    print(f"La statistique de test est {statistic}. La p-value est {pvalue}")
```



```

La statistique de test est 1.0. La p-value est 0.0
La statistique de test est 1.0. La p-value est 0.0
La statistique de test est 0.9999999999999991. La p-value est 0.0
La statistique de test est 1.0. La p-value est 0.0
La statistique de test est 1.0. La p-value est 0.0
La statistique de test est 1.0. La p-value est 0.0
La statistique de test est 1.0. La p-value est 0.0
La statistique de test est 0.9999999826391145. La p-value est 1.772614558370926e-225
La statistique de test est 0.9999999964435136. La p-value est 3.6842352285620746e-127

```

Le test ne permet pas d'accepter l'hypothèse nulle : d'après la distribution de nos données, ces dernières ont peu de chance d'être normalement distribuées en chaque point du facteur qualitatif. Si vous préférez employer un test d'adéquation paramétrique et que votre échantillon compte moins de 5 000 observations, vous pouvez utiliser le test de Shapiro-Wilk. Voici comment :

```

In [30]: from scipy.stats import shapiro
        for group in groups:
            statistic, pvalue = shapiro(group,)
            print(statistic, pvalue)

```

```

0.8958697319030762 0.03452790528535843
0.9055184125900269 0.061288632452487946
0.8486941456794739 0.00047526619164273143
0.9523464441299438 0.7348506450653076
0.7696629762649536 0.003757531987503171
0.9053825736045837 0.09805720299482346
0.9088010787963867 0.09548340737819672
0.9023045301437378 0.011129096150398254
0.8471646308898926 0.015836745500564575

```

D'après le test de Shapiro-Wilk, l'échantillon '69005' présente suffisamment d'évidence pour ne pas rejeter l'hypothèse nulle au seuil de significativité de 5%. Pour les autres arrondissements, nous n'avons pas suffisamment d'évidence pour affirmer que les données sont normalement distribuées en chaque point du facteur qualitatif.

Avec de telles conclusions, il n'est pas possible d'appliquer le test de Grubbs ou le score Z. Nous allons donc utiliser le **score Z modifié** - c'est une version robuste du score Z. J'ai décrit la mise en place de ce test dans une note dédiée à la **détection et au traitement des outliers**. Au cours de cette note j'ai défini une fonction permettant de calculer le score Z modifié. Nous allons donc importer depuis ce projet cette procédure.

Après avoir recherché d'éventuels outliers, et pour la démonstration, nous allons vérifier l'hypothèse d'homoscédasticité. Cependant, puisque nos sous-échantillons ne sont pas normalement distribués, cette étape n'est pas nécessaire : vous pouvez passer directement à la mise en place du test de Kruskal-Wallis. Il s'agit d'un test (non-paramétrique) d'homogénéité de médianes (au moins 2). C'est une alternative à l'Anova à un facteur lorsque les hypothèses de normalité et/ou d'homoscédasticité ne sont pas respectées.

Notez que pour vérifier l'homogénéité des variances il est préférable d'employer le test de Levene à celui de Bartlett lorsque les données ne sont pas normalement distribuées.

```
In [31]: # création des sous-échantillons.
groups = df.groupby('location', observed=True)['prix_m2'].apply(list)
display(groups, len(groups))
```

```
location
69001    [25.3, 17.737704918032787, 11.64655172413793, ...
69002    [23.466666666666665, 17.162921348314608, 39.58...
69003    [13.273809523809524, 17.941176470588236, 33.0,...
69004    [19.444444444444443, 23.076923076923077, 16.38...
69005    [22.75, 12.323529411764707, 36.363636363637,...
69006    [20.992, 17.896551724137932, 18.22619047619047...
69007    [16.05128205128205, 21.442622950819672, 31.875...
69008    [20.727272727272727, 20.285714285714285, 14.98...
69009    [15.742857142857142, 36.6875, 17.7692307692307...
Name: prix_m2, dtype: object
9
```

```
In [32]: # Score Z modifié
from outliers_detection import robust_z_score

#help(robust_z_score)
for i in range(0, len(groups)):
    scores = robust_z_score(variable=groups[i])
    for j, score in enumerate(scores):
        if np.abs(score) > 4 :
            print(i+1, groups[i][j])
```

```
1 38.125
3 41.25
5 36.36363636363637
9 36.6875
```

```
/tmp/ipykernel_8541/1017329994.py:6: FutureWarning:
```

```
Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
```

```
/tmp/ipykernel_8541/1017329994.py:9: FutureWarning:
```

```
Series.__getitem__ treating keys as positions is deprecated. In a future version, integer keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
```

Les observations 38.125 (1er arrondissement) et 41.125 (3ème arrondissement) sont les mêmes outliers que ceux rencontrés dans la distribution globale du prix au mètre carré. Les observations 36.36 (appartement 2 chambres) et 36.69 (studio) sont quant à elles des outliers pour le 5ème et le 9ème arrondissement, respectivement.

Ces outliers ne sont pas des erreurs de saisie, simplement des points atypiques. Ce type de points peut largement affecter nos résultats en introduisant un biais. Dans ce cas, deux solutions sont possibles : soit ils ont un intérêt pour notre analyse, dans quel cas, on les conserve et on évalue l'impact qu'ils ont sur notre analyse ; soit ils n'ont pas d'intérêt particulier, dans quel cas on les supprime de notre échantillon.

Pour simplifier l'analyse, nous allons les retirer.

```
In [33]: _ = df[(df["prix_m2"] == 36.6875) | (df["prix_m2"] == 36.36363636363637) | (df["prix_m2"] == 38.125) | (df["prix_m2"] == 41.25)]
df.drop(index=_, inplace = True)
```

Si vous souhaitez réaliser un test de Grubbs malgré le possible défaut de normalité de nos sous-échantillons, voici comment : utilisez la fonction `smirnov_grubbs` du module `outliers`. La fonction `outliers_gesd` vous permet de rechercher plusieurs outliers à la fois et le paramètre `report` permet d'obtenir un rendu résultat très sympa.

```
# test de Grubbs. from outliers import smirnov_grubbs as grubbs groups_outlier_free = [] for i, group in zip(groups.index, groups.values): Gtest = grubbs.test(group, alpha=.05) groups_outlier_free.append(Gtest) print(f"{i} : La valeur maximale est-elle un outlier ? {len(group) - len(Gtest)}") from scikit_posthocs import outliers_gesd as gesd for group in groups: outliers = gesd(x = group, outliers = 5, report = True, alpha = 0.05) outliers
```

Nous allons maintenant réaliser le test d'homogénéité des variances. Cet étape n'a d'intérêt que de démontrer comment effectuer ce test avec python. Le test de Bartlett est utilisé lorsque vos sous-échantillons sont normalement distribués, sinon il faut utiliser le test de

Levene.

```
In [35]: # Test de Levene - test d'homogénéité des variances pour distributions Anormales.
```

```
from scipy.stats import levene
```

```
statistic, pvalue = levene(*groups, center="median", proportiontocut=0.05)  
display(statistic, pvalue)
```

```
0.397139395684097
```

```
0.9208356695839017
```

```
# Test de Bartlett - test d'homogénéité des variances pour distributions Normales. from scipy.stats import bartlett  
statistic, pvalue = bartlett(*groups, axis=0, nan_policy="omit", keepdims=False) display(statistic, pvalue)
```

D'après le test de Levene, nous avons suffisamment d'évidence pour affirmer que les variances de nos sous-échantillons sont homogènes.

Notez que si vos sous-échantillons sont normalement distribués et que vos variances ne sont pas homogènes vous pouvez appliquer l'Anova de Welch.

Passons maintenant à la réalisation d'un test de Kruskal-Wallis. Ce test repose sur une somme des rangs, c'est pourquoi il est souvent présenté comme un test de comparaisons de médianes. C'est une généralisation du test de Mann-Whitney-Wilcoxon.

Avec python vous pouvez utiliser le module *scipy.stats* qui propose une fonction *kruskal*.

```
In [36]: # Test de Kruskal-Wallis.
```

```
from scipy.stats import kruskal  
statistic, pvalue = kruskal(*groups, nan_policy="omit", axis=0, keepdims=False)  
print(f"La statistique de test est : {statistic}. La pvalue vaut : {pvalue}")
```

```
La statistique de test est : 17.117054318966783. La pvalue vaut : 0.028913021830870668
```

```
from scipy.stats import kruskal  
statistic, pvalue = kruskal(*groups_outlier_free, nan_policy="omit", axis=0, keepdims=False)  
print(f"La statistique de test est : {statistic}. La pvalue vaut : {pvalue}")
```

D'après le résultat du test de Kruskal-Wallis, nous avons suffisamment d'évidence pour rejeter l'hypothèse nulle : au moins deux des populations parentes diffèrent d'après leur médiane. Pour l'exprimer autrement, il y a de faible chance pour que toutes les médianes des sous-échantillons soient identiques.

Pour savoir quelles médianes diffèrent les unes des autres (ou les unes par rapport à un groupe de référence) on doit réaliser un test post-hoc. Dans ce qui suit nous allons réaliser une procédure de Tukey : on compare tous les échantillons deux-à-deux avec un test de Wilcoxon-

Mann-Withney ou un test de Dunn. (Attention, en python, le test de Wilcoxon fait référence au test des rangs signés et non pas à celui de la somme des rangs : vous devez utiliser le test de Mann-Whitney).

```
In [37]: # test de Tukey-Wilcoxon
from scikit_posthocs import posthoc_mannwhitney as MWW

results = MWW(a=df, val_col = "prix_m2", group_col = "location", use_continuity = True, alternative = 'two-sided',
```

/home/jordan-43000/.local/lib/python3.10/site-packages/scikit_posthocs/_posthocs.py:1898: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.

```
from scikit_posthocs import posthoc_mannwhitney as MWW results = MWW(groups_outlier_free, use_continuity = True, alternative = 'two-sided',
p_adjust = "holm", sort = True) results
```

Avant d'en venir au résultat, observez le paramètre "**p_adjust='holm'**" : il permet d'ajuster les p-values des tests en fonction du nombre de tests d'hypothèse effectués, c'est-à-dire que les p-values sont diminuées afin de nous assurer que le risque de première espèce global α est toujours celui qu'on s'est fixé avant de réaliser les n tests d'hypothèse indépendants, à savoir 5%.

En effet, lorsqu'on réalise un grand nombre de test - comme lors de comparaisons multiples - le risque de rejeter l'hypothèse nulle à tort diminue à mesure que le nombre de test augmente. Ainsi, pour n tests réalisés, le risque de première espèce global sera $1 - (1 - \alpha)^n$: soit la **probabilité de faire au moins une erreur avec n tests d'hypothèses indépendants**. On doit donc corriger les p-values pour nous assurer que globalement, le risque de première espèce est toujours à 5%. Pour cela il existe différentes méthodes dont la plus connue, Bonferroni, n'est pas une solution adaptée. Il est courant d'utiliser la **méthode de Holm**.

Voyons les différences significatives que notre test post-hoc a permis de mettre en évidence.

```
In [38]: results
```

Out[38]:

	69001	69002	69003	69004	69005	69006	69007	69008	69009
69001	1.0	1.00000	1.0	1.0	1.000000	1.000000	1.000000	1.000000	1.000000
69002	1.0	1.00000	1.0	1.0	1.000000	1.000000	1.000000	0.365260	1.000000
69003	1.0	1.00000	1.0	1.0	1.000000	1.000000	1.000000	1.000000	1.000000
69004	1.0	1.00000	1.0	1.0	1.000000	1.000000	1.000000	1.000000	1.000000
69005	1.0	1.00000	1.0	1.0	1.000000	0.607632	0.897557	1.000000	1.000000
69006	1.0	1.00000	1.0	1.0	0.607632	1.000000	1.000000	0.142710	0.441254
69007	1.0	1.00000	1.0	1.0	0.897557	1.000000	1.000000	0.236841	1.000000
69008	1.0	0.36526	1.0	1.0	1.000000	0.142710	0.236841	1.000000	1.000000
69009	1.0	1.00000	1.0	1.0	1.000000	0.441254	1.000000	1.000000	1.000000

```
In [39]: for i in results.index:
          for j in results.columns:
              if i < j:
                  pvalue = results.loc[i, j]
                  if pvalue < 0.05:
                      print(f"{i} and {j} have significantly different medians : {pvalue}")
```

D'après les résultats des tests de Mann-Whitney-Wilcoxon deux-à-deux, aucun de nos sous-échantillons ne présente une différence de médiane statistiquement significative vis-à-vis d'un autre.

Notez que ce type de test n'est pas idéale après réalisation d'un test de Kruskal-Wallis puisqu'il compare les sommes de rangs deux à deux et non pas la moyenne des rangs de tous les sous-échantillons. On peut donc utiliser le test de Dunn qui permet de conserver l'esprit du test de Kruskal-Wallis : il utilise les mêmes classements partagés que ceux calculés par le test de Kruskal-Wallis, et la même variance mise en commun que celle impliquée par l'hypothèse nulle du test de Kruskal-Wallis. Il utilise donc les mêmes données que le test de Kruskal-Wallis pour tester les différences entre deux groupes. **Le test de Dunn est donc plus adapté si vous souhaitez réaliser des comparaisons deux-à-deux après un test de Kruskal-Wallis.** Pour en savoir plus, [lisez cet article](#).

Nous allons maintenant utiliser un test post-hoc avec le test de Dunn.

In [40]: *# Test de Dunn*

```
from scikit_posthocs import posthoc_dunn as dunn

results = dunn(a=df, val_col = "prix_m2", group_col = "location", p_adjust = "holm")

for i in results.index:
    for j in results.columns:
        if i < j:
            pvalue = results.loc[i, j]
            if pvalue < 0.05:
                print(f"{i} and {j} have significantly different medians : {pvalue}")
```

/home/jordan-43000/.local/lib/python3.10/site-packages/scikit_posthocs/_posthocs.py:357: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.

/home/jordan-43000/.local/lib/python3.10/site-packages/scikit_posthocs/_posthocs.py:360: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.

69006 and 69008 have significantly different medians : 0.0399519766249451

D'après le test de Dunn on peut bien observer une **différence de prix médian au mètre carré entre le huitième et le sixième arrondissement** de Lyon.

Nous allons maintenant passer aux comparaisons multiples à un groupe de référence : le huitième arrondissement. Pour cela, on va réaliser une procédure à la Dunnett, c'est-à-dire qu'on va comparer deux-à-deux tous les sous-échantillons sans correction des p-values, puis, on corrigera les p-values relative aux comparaisons avec le huitième arrondissement.

In [41]: *# Procédure de Dunnett avec test de Dunn.*

```
results = dunn(a=df, val_col = "prix_m2", group_col = "location", p_adjust = None)
results = results.loc["69008",:].drop("69008")
results
```

/home/jordan-43000/.local/lib/python3.10/site-packages/scikit_posthocs/_posthocs.py:357: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.

/home/jordan-43000/.local/lib/python3.10/site-packages/scikit_posthocs/_posthocs.py:360: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.

```
Out[41]: 69001    0.035133
        69002    0.005223
        69003    0.060912
        69004    0.134794
        69005    0.938485
        69006    0.001110
        69007    0.004270
        69009    0.614696
        Name: 69008, dtype: float64
```

```
In [42]: # on corrige les pvalues avec une procédure de Holm.

from statsmodels.stats.multitest import multipletests as adjusted_pvalues

reject, pvalues_corrected, *_ = adjusted_pvalues(pvals=results, alpha=0.05, method="holm", maxiter=1, is_sorted=False)

display(reject, pvalues_corrected)

array([False,  True, False, False, False,  True,  True, False])
array([0.17566696, 0.03133621, 0.2436493 , 0.40438271, 1.          ,
        0.00887822, 0.0298885 , 1.          ])
```

```
In [43]: for i, decision, p, pc in zip(results.index, reject, results, pvalues_corrected):
        print("--"*35)
        print("Location | Decision | Pvalues | Pvalues_corrected_holm")
        print(i, decision, p, pc)
```



```
-----  
Location | Decision | Pvalues | Pvalues_corrected_holm  
69001 False 0.035133391165558685 0.17566695582779343  
-----
```

```
Location | Decision | Pvalues | Pvalues_corrected_holm  
69002 True 0.00522270094181683 0.03133620565090098  
-----
```

```
Location | Decision | Pvalues | Pvalues_corrected_holm  
69003 False 0.06091232513567451 0.24364930054269804  
-----
```

```
Location | Decision | Pvalues | Pvalues_corrected_holm  
69004 False 0.13479423510606464 0.4043827053181939  
-----
```

```
Location | Decision | Pvalues | Pvalues_corrected_holm  
69005 False 0.938485359205977 1.0  
-----
```

```
Location | Decision | Pvalues | Pvalues_corrected_holm  
69006 True 0.0011097771284706973 0.008878217027765579  
-----
```

```
Location | Decision | Pvalues | Pvalues_corrected_holm  
69007 True 0.004269786399141684 0.029888504793991792  
-----
```

```
Location | Decision | Pvalues | Pvalues_corrected_holm  
69009 False 0.6146960552337402 1.0  
-----
```

D'après la procédure de Dunnett, en utilisant les pvalues non corrigées du test de Dunn, le prix au mètre carré médian du 8ème arrondissement est significativement différent de celui des arrondissements suivants :

- 2ème,
- 6ème, et
- 7ème.

Voyons le résultat avec le test de Mann-Whitney-Wilcoxon.

```
In [45]: results = MWW(a=df, val_col = "prix_m2", group_col = "location", use_continuity = True, alternative = 'two-sided',  
results = results.loc["69008",].drop("69008")  
display(results)  
  
reject, pvalues_corrected, *_ = adjusted_pvalues(pvals=results, alpha=0.05, method="holm", maxiter=1, is_sorted=False)
```

```
display(reject, pvalues_corrected)

for i, decision, p, pc in zip(results.index, reject, results, pvalues_corrected):
    print("--"*35)
    print("Location | Decision | Pvalues | Pvalues_corrected_holm")
    print(i, decision, p, pc)
```

/home/jordan-43000/.local/lib/python3.10/site-packages/scikit_posthocs/_posthocs.py:1898: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.

```
69001    1.000000
69002    0.365260
69003    1.000000
69004    1.000000
69005    1.000000
69006    0.142710
69007    0.236841
69009    1.000000
Name: 69008, dtype: float64
array([False, False, False, False, False, False, False, False])
array([1., 1., 1., 1., 1., 1., 1., 1.])
```

```

-----
Location | Decision | Pvalues | Pvalues_corrected_holm
69001 False 1.0 1.0
-----
Location | Decision | Pvalues | Pvalues_corrected_holm
69002 False 0.3652598270607638 1.0
-----
Location | Decision | Pvalues | Pvalues_corrected_holm
69003 False 1.0 1.0
-----
Location | Decision | Pvalues | Pvalues_corrected_holm
69004 False 1.0 1.0
-----
Location | Decision | Pvalues | Pvalues_corrected_holm
69005 False 1.0 1.0
-----
Location | Decision | Pvalues | Pvalues_corrected_holm
69006 False 0.14270950486359135 1.0
-----
Location | Decision | Pvalues | Pvalues_corrected_holm
69007 False 0.236841483624664 1.0
-----
Location | Decision | Pvalues | Pvalues_corrected_holm
69009 False 1.0 1.0

```

D'après la procédure de Dunnett réalisée avec un test de Mann-Whitney-Wilcoxon, aucun des arrondissements ne présente un prix moyen du mètre carré significativement différent du huitième arrondissement. Ce résultat est concordant avec le test post-hoc deux-à-deux réalisé avec le test de Mann-Withney-Wilcoxon.

Voilà, nous sommes arrivés au bout de notre analyse ! Vous savez comment analyser une différence de sous-échantillons d'après leur tendance centrale. J'espère que ce post vous a plu.