Analyse De Données De La Consommation De Carburant

```
pip install jyquickhelper

#Cette commande est utilisée pour installer le package Python appelé jyquickhelper. Ce package est un ensemble d'outils et d'utilitaires
#notamment pour simplifier certaines tâches liées à l'analyse de données, à la visualisation ou à l'interaction avec les notebooks Jupyt
```

```
Requirement already satisfied: ipykernel in /usr/local/lib/python3.10/dist-packages (from jupyter->jyquickhelper) (5.5.6)
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.10/dist-packages (from jupyter->jyquickhelper) (7.7.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from notebook->jyquickhelper) (3.1.3)
Requirement already satisfied: tornado>=6.1 in /usr/local/lib/python3.10/dist-packages (from notebook->jyquickhelper) (6.3.3)
Requirement already satisfied: pyzmq<25,>=17 in /usr/local/lib/python3.10/dist-packages (from notebook->jyquickhelper) (23.2.1)
Requirement already satisfied: argon2-cffi in /usr/local/lib/python3.10/dist-packages (from notebook->jyquickhelper) (23.1.0)
Requirement already satisfied: jupyter-core>=4.6.1 in /usr/local/lib/python3.10/dist-packages (from notebook->jyquickhelper) (5.7
Requirement already satisfied: jupyter-client<8,>=5.3.4 in /usr/local/lib/python3.10/dist-packages (from notebook->jyquickhelper)
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.10/dist-packages (from notebook->jyquickhelper) (0.2.0)
Requirement already satisfied: nbformat in /usr/local/lib/python3.10/dist-packages (from notebook->jyquickhelper) (5.10.3)
Requirement already satisfied: nest-asyncio>=1.5 in /usr/local/lib/python3.10/dist-packages (from notebook->jyquickhelper) (1.6.0
Requirement already satisfied: Send2Trash>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from notebook->jyquickhelper) (1.8.2
Requirement already satisfied: terminado>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from notebook->jyquickhelper) (0.18.1
Requirement already satisfied: prometheus-client in /usr/local/lib/python3.10/dist-packages (from notebook->jyquickhelper) (0.20.
Requirement already satisfied: nbclassic>=0.4.7 in /usr/local/lib/python3.10/dist-packages (from notebook->jyquickhelper) (1.0.0)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from jedi>=0.16->ipython->jyquickh
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.10/dist-packages (from jupyter-client<8,>=5.3.4->no
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.6.1->notebook->
Requirement already satisfied: jupyter-server>=1.8 in /usr/local/lib/python3.10/dist-packages (from nbclassic>=0.4.7->notebook->j
Requirement already satisfied: notebook-shim>=0.2.3 in /usr/local/lib/python3.10/dist-packages (from nbclassic>=0.4.7->notebook->
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->jyquickhelper) (4.9.4)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->jyquickhelper)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->jyquickhelper) (6.1.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->jyquickhelper) (0.
Requirement already satisfied: entrypoints>=0.2.2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->jyquickhel
Requirement already satisfied: jupyterlab-pygments in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->jyquickhe
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->jyquickhelper
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->jyquickhelp
Requirement already satisfied: nbclient>=0.5.0 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->jyquickhelper
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->jyquickhelper) (24.
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->jyquickh
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-packages (from nbconvert->jupyter->jyquickhelper) (1.2.
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.10/dist-packages (from nbformat->notebook->jyquickhelper)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-packages (from nbformat->notebook->jyquickhelper
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packages (from pexpect>4.3->ipython->jyquickhelp
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.
Requirement already satisfied: argon2-cffi-bindings in /usr/local/lib/python3.10/dist-packages (from argon2-cffi->notebook->jyqui
Requirement already satisfied: widgetsnbextension~=3.6.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter->jy
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ipywidgets->jupyter->jy
Collecting qtpy>=2.4.0 (from qtconsole->jupyter->jyquickhelper)
  Downloading QtPy-2.4.1-py3-none-any.whl (93 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 93.5/93.5 kB 10.5 MB/s eta 0:00:00
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->notebook
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->no
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->noteboo
Requirement already satisfied: anyio<4,>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from jupyter-server>=1.8->nbclassic>=0
Requirement already satisfied: websocket-client in /usr/local/lib/python3.10/dist-packages (from jupyter-server>=1.8->nbclassic>=
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.1->jupyter-client<8,>
Requirement already satisfied: cffi>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from argon2-cffi-bindings->argon2-cffi->no
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->nbconvert->jupyter-
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->nbconvert->jupyter->jyquickh
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server>=1.8->n
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server>=1.8
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server>=1
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.0.1->argon2-cffi-bindings->argo
Installing collected packages: qtpy, jedi, qtconsole, jupyter, jyquickhelper
Successfully installed jedi-0.19.1 jupyter-1.0.0 jyquickhelper-0.4.220 qtconsole-5.5.1 qtpy-2.4.1
```

```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
#Cettecommande add_notebook_menu() provenant de la bibliothèque jyquickhelper est utilisée pour ajouter un menu interactif à un noteboo
#Ce menu offre des fonctionnalités supplémentaires pour naviguer et manipuler le notebook, telles que l'exportation vers différents form
```

```
# Importation de mes bibliothèques
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import scipy
```

```
from google.colab import files
uploaded = files.upload()
```

Sélect. fichiers payement_…rant (3).csv

- **payement_carburant (3).csv**(text/csv) - 2515 bytes, last modified: 25/03/2024 - 100% done

```
Saving payement_carburant (3).csv to payement_carburant (3).csv
```

```
df = pd.read_csv('payement_carburant (3).csv')
df
```

	ID_PAYEMENT	NOM_CARBURANT	DATE_PAYEMENT	HEURE_PAYEMENT	VOLUME_CARBURANT	PRIX_UNITAIRE_LITRE	TOTAL_TTC	TOTAL_NET	M
0	1	Diesel	2023-12-11	18:40:09	20.22l	1.731	35.000	29.17	DECEME
1	2	Diesel	2023-09-23	12:09:00	33.66l	1.487	50.050	42.78	SEPTEME
2	3	Diesel	2023-12-08	15:31:34	13.69l	1.462	20.010	17.25	DECEME
3	4	Diesel	2023-10-19	19:12:27	8.10l	1.859	15.060	12.55	OCTOE
4	5	Diesel	2024-02-15	17:16:30	13.89l	1.462	20.310	17.51	FEVR
5	6	Diesel	2024-01-11	19:02:32	41.47l	1.504	62.370	53.77	JANV
6	7	Diesel	2024-01-23	17:47:46	50.65l	1.602	81.140	69.95	JANV
7	8	Diesel	2023-10-04	18:20:10	5.3l	1.929	10.240	8.53	OCTOE
8	9	Diesel	2024-02-24	14:35:16	23.02l	1.657	38.140	31.78	FEVR
9	10	Diesel	2023-08-03	07:23:18	24.83l	1.654	41.070	35.41	AC
10	11	Diesel	2023-11-07	18:00:48	13.39l	1.867	25.000	20.83	NOVEME
11	12	Diesel	2023-10-24	18:00:08	39.57l	1.462	57.850	49.87	OCTOE
12	13	Diesel	2023-12-29	12:52:26	37.31l	1.695	56.000	48.67	DECEME
13	14	Diesel	2023-09-04	18:06:21	21.30l	1.504	32.040	27.62	SEPTEME
14	15	Diesel	2024-01-26	17:41:26	6.31l	1.542	12.430	10.72	JANV
15	16	Diesel	2023-11-17	18:41:09	49.81l	1.527	76.210	65.14	NOVEME
16	17	Diesel	2023-07-13	18:52:41	16.97l	1.781	30.220	25.18	JUIL
17	18	Diesel	2023-09-23	12:09:00	43.74l	1.604	70.160	60.48	SEPTEME
18	19	Diesel	2023-11-05	05:42:19	18.32l	1.638	30.010	25.87	NOVEME
19	20	Diesel	2023-08-31	18:33:52	42.12l	1.614	67.980	58.60	AC
20	21	Diesel	2023-06-28	12:33:36	29.93l	1.720	51.480	44.38	J
21	22	Diesel	2023-08-16	07:29:34	35.69l	1.626	58.030	49.60	AC
22	23	Diesel	2023-10-30	17:05:19	33.36l	54.640	1.638	47.10	OCTOE
23	24	Diesel	2023-12-14	18:13:14	26.51l	1.509	40.000	34.19	DECEME
24	25	Diesel	2023-07-16	13:25:17	43.92l	1.573	69.090	59.56	JUIL
25	26	Diesel	2023-07-21	15:40:09	11.78l	1.783	21.000	17.50	JUIL
26	27	Diesel	2023-12-10	16:12:54	35.66l	1.767	63.010	53.67	DECEME

Next steps: View recommended plots

Description de notre Dataset

```
!pip install --upgrade pydantic
```

```
Requirement already satisfied: pydantic in /usr/local/lib/python3.10/dist-packages (2.6.4)
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic) (0.6.0)
Requirement already satisfied: pydantic-core==2.16.3 in /usr/local/lib/python3.10/dist-packages (from pydantic) (2.16.3)
Requirement already satisfied: typing-extensions>=4.6.1 in /usr/local/lib/python3.10/dist-packages (from pydantic) (4.10.0)
```

```
!pip install ydata-profiling
```

```
  Downloading dacite-1.8.1-py3-none-any.whl (14 kB)
Requirement already satisfied: numba<1,>=0.56.0 in /usr/local/lib/python3.10/dist-packages (from ydata-profiling) (0.58.1)
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.10/dist-packages (from imagehash==4.3.1->ydata-profiling) (1.
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from imagehash==4.3.1->ydata-profiling) (9.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2<3.2,>=2.11.1->ydata-profil
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.9,>=3.2->ydata-prof
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.9,>=3.2->ydata-profilin
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.9,>=3.2->ydata-pro
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.9,>=3.2->ydata-pro
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.9,>=3.2->ydata-profi
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.9,>=3.2->ydata-prof
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib<3.9,>=3.2->ydata-
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba<1,>=0.56.0->ydat
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas!=1.4.0,<3,>1.1->ydata-profili
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.10/dist-packages (from phik<0.13,>=0.11.1->ydata-profilin
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-profili
Requirement already satisfied: pydantic-core==2.16.3 in /usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-profilin
Requirement already satisfied: typing-extensions>=4.6.1 in /usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-profi
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->yda
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-profiling
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-pro
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-pro
Requirement already satisfied: patsy>=0.5.4 in /usr/local/lib/python3.10/dist-packages (from statsmodels<1,>=0.13.2->ydata-profil
Collecting pandas!=1.4.0,<3,>1.1 (from ydata-profiling)
  Downloading pandas-2.2.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (13.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.0/13.0 MB 44.0 MB/s eta 0:00:00
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.10/dist-packages (from visions[type_image_path]<0.7.7,>=0.
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.10/dist-packages (from visions[type_image_path]<0.7.7,>=0.
Collecting tzdata>=2022.7 (from pandas!=1.4.0,<3,>1.1->ydata-profiling)
  Downloading tzdata-2024.1-py2.py3-none-any.whl (345 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 345.4/345.4 kB 29.4 MB/s eta 0:00:00
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.4->statsmodels<1,>=0.13.2->ydata-p
Building wheels for collected packages: htmlmin
  Building wheel for htmlmin (setup.py) ... done
  Created wheel for htmlmin: filename=htmlmin-0.1.12-py3-none-any.whl size=27080 sha256=81c3a96addac70f534e18549bc7af077dfafd1ea1
  Stored in directory: /root/.cache/pip/wheels/dd/91/29/a79cecb328d01739e64017b6fb9a1ab9d8cb1853098ec5966d
Successfully built htmlmin
Installing collected packages: htmlmin, tzdata, typeguard, multimethod, dacite, pandas, imagehash, visions, seaborn, phik, ydata-
  Attempting uninstall: pandas
    Found existing installation: pandas 1.5.3
    Uninstalling pandas-1.5.3:
      Successfully uninstalled pandas-1.5.3
  Attempting uninstall: seaborn
    Found existing installation: seaborn 0.13.1
    Uninstalling seaborn-0.13.1:
      Successfully uninstalled seaborn-0.13.1
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the
bigframes 0.26.0 requires pandas<2.1.4,>=1.5.0, but you have pandas 2.2.1 which is incompatible.
google-colab 1.0.0 requires pandas==1.5.3, but you have pandas 2.2.1 which is incompatible.
```

```
def reporting(dataset):
#https://github.com/pandas-profiling/pandas-profiling
#Exploratory Data Analysis
#https://www.datacorner.fr/pandas-profiling/

    from ydata_profiling import ProfileReport
    #from pandas_profiling import ProfileReport
    resultats = df.profile_report(title='Pandas Profiling Report', correlations={"cramers": {"calculate": False}})
    resultats.to_file("ProfilingWorldHapiness_ESTHER_AGNES2.html")
    return None
reporting(df.sample(frac=0.3))
```

```
/usr/local/lib/python3.10/dist-packages/ydata_profiling/profile_report.py:354: UserWarning: Try running command: 'pip install --upgr
  warnings.warn(
Summarize dataset: 100%                                                    35/35 [00:05<00:00,  3.48it/s, Completed]
Generate report structure: 100%                                                1/1 [00:08<00:00,  8.29s/it]
Render HTML: 100%                                                          1/1 [00:00<00:00,  1.57it/s]
Export report to file: 100%                                                1/1 [00:00<00:00, 38.17it/s]
```

```
df.shape
#Mon Dataset compte 27 lignes et 8 Colonnes
```

```
(27, 10)
```

```
df_quanti = df.select_dtypes(exclude='object')
df_quali =df.select_dtypes(include='object')
print(
    f"La base de données contient {df_quanti.shape[1]} variables quantitatives",
    f" et {df_quali.shape[1]} variables qualitatives")
```

La base de données contient 5 variables quantitatives et 5 variables qualitatives

```
df.info()
# Informations sur le Dataset pour savoir les colonnes quantitatives(int, float) et Qualitatives(object, categorie)
```

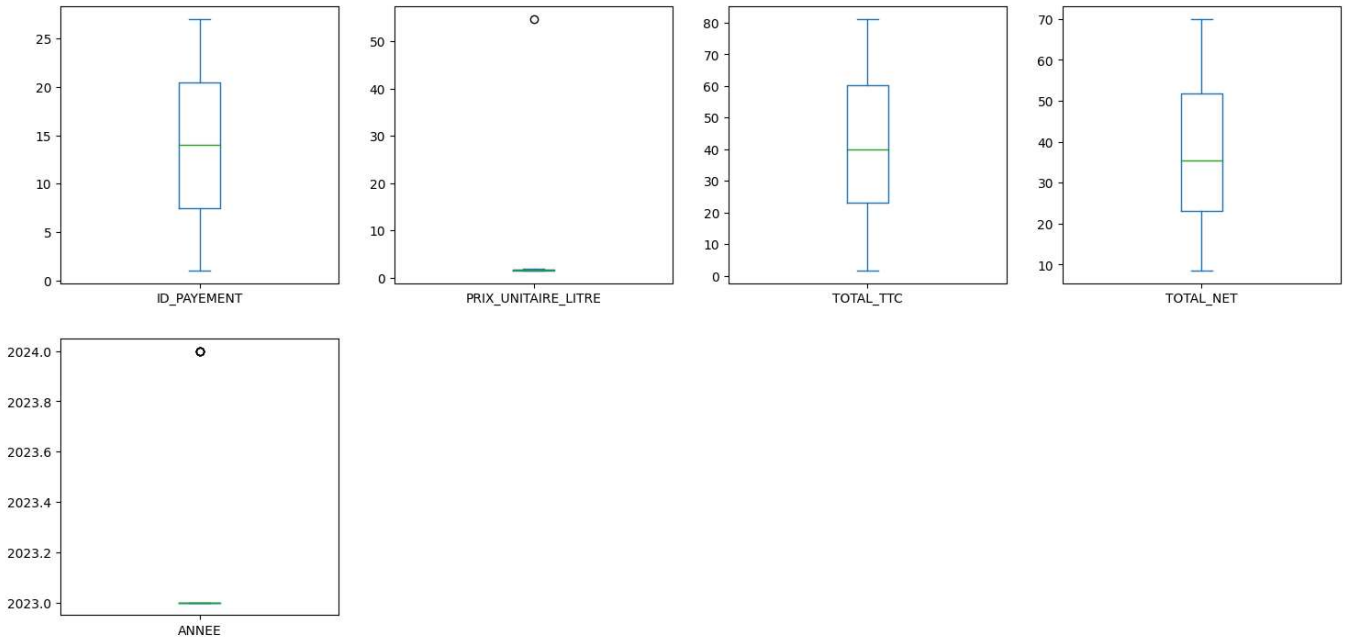
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27 entries, 0 to 26
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID_PAYEMENT          27 non-null     int64
 1   NOM_CARBURANT        27 non-null     object
 2   DATE_PAYEMENT        27 non-null     object
 3   HEURE_PAYEMENT       27 non-null     object
 4   VOLUME_CARBURANT     27 non-null     object
 5   PRIX_UNITAIRE_LITRE  27 non-null     float64
 6   TOTAL_TTC            27 non-null     float64
 7   TOTAL_NET            27 non-null     float64
 8   MOIS                 27 non-null     object
 9   ANNEE                27 non-null     int64
dtypes: float64(3), int64(2), object(5)
memory usage: 2.2+ KB
```

```
df.describe() # Quantitative
#Les variables  comptent 27 observations dans les colonnes
# Ona La moyenne des valeurs de chaque variable
# std (écart-type): C'est une mesure de la dispersion des valeurs par rapport à la moyenne. Un écart-type élevé indique une dispersion i
# min:On a le Min qui est  la valeur la plus basse de chaque colonne
# 25% (premier quartile)
# 50% (médiane): C'est la valeur médiane de la distribution des données
# 75% (troisième quartile)
# max : C'est la valeur la plus élevée dans la colonne.
```

	ID_PAYEMENT	PRIX_UNITAIRE_LITRE	TOTAL_TTC	TOTAL_NET	ANNEE
count	27.000000	27.000000	27.000000	27.000000	27.000000
mean	14.000000	3.599963	42.056963	37.691852	2023.185185
std	7.937254	10.201322	22.353499	18.147082	0.395847
min	1.000000	1.462000	1.638000	8.530000	2023.000000
25%	7.500000	1.518000	23.000000	23.005000	2023.000000
50%	14.000000	1.626000	40.000000	35.410000	2023.000000
75%	20.500000	1.749000	60.200000	51.770000	2023.000000
max	27.000000	54.640000	81.140000	69.950000	2024.000000

Verification des outlears et des données manquantes

```
#Vérification de la présence de Outlears
df.plot(kind='box', subplots=True, layout=(4,4), sharex=False, sharey=False, figsize=(18,18))
plt.show()

#On constate que la variable suivante a une  donnée abbérante: Prx_Unitaire_Litre et ANNEE
#Et les  Variables suivantes n'ont pas de données abbérantes: ID_payement,Total_ttc et Total_Net
```




```
def outliers_rate(dataset):
    df_num=df.select_dtypes(exclude=['object',"category"])   #object=variables qualitatives
    Q1 = df_num.quantile(0.25)
    Q3 = df_num.quantile(0.75)
    IQR = Q3 - Q1
    is_outliers = (df_num < (Q1 - 1.5 * IQR)) |(df_num > (Q3 + 1.5 * IQR))

    outliers_missing=pd.DataFrame({ 'outliers_rate':is_outliers.sum(axis=0)/len(df_num)*100})
    return outliers_missing
outliers_rate(df)

#Pour déterminer le taux de valeur abbérante de chaque colonne
#On voit que Prix_Unitaire_Litre a un taux de valeur abbérante de 3.70%
#On voit que ANNEE a un taux de valeur abbérante de 18.51%
```

	outliers_rate
ID_PAYEMENT	0.000000
PRIX_UNITAIRE_LITRE	3.703704
TOTAL_TTC	0.000000
TOTAL_NET	0.000000
ANNEE	18.518519

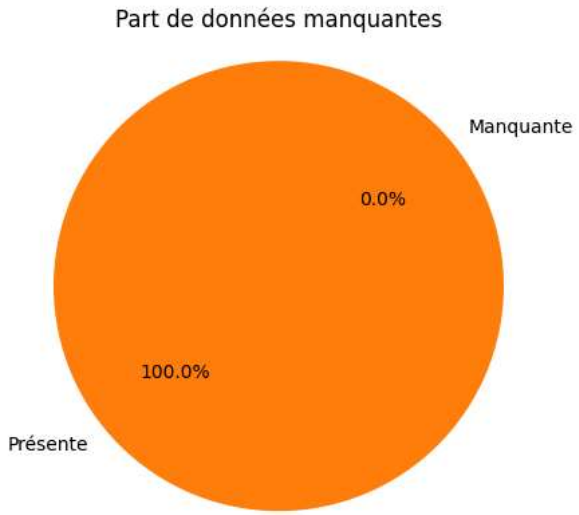
```
df.isna().sum()
# On a le nombre de données manquantes pour chaque colonne.
```

```
ID_PAYEMENT           0
NOM_CARBURANT         0
DATE_PAYEMENT         0
HEURE_PAYEMENT        0
VOLUME_CARBURANT      0
PRIX_UNITAIRE_LITRE   0
TOTAL_TTC             0
TOTAL_NET             0
MOIS                  0
ANNEE                 0
dtype: int64
```

```
df.isna().sum(axis=0) # Pour determiner le nombre de donnée manquante de chaque colonne
#Nombre de donnéés manquante pour chaque colonne
```

```
ID_PAYEMENT           0
NOM_CARBURANT         0
DATE_PAYEMENT         0
HEURE_PAYEMENT        0
VOLUME_CARBURANT      0
PRIX_UNITAIRE_LITRE   0
TOTAL_TTC             0
TOTAL_NET             0
MOIS                  0
ANNEE                 0
dtype: int64
```

```
labels = 'Manquante', 'Présente'

fig, ax = plt.subplots()
sizes = df.isnull().sum().sum(), df.notnull().sum().sum()
ax.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=False, startangle=40)
ax.axis('equal')
ax.set_title('Part de données manquantes')

plt.show()

#Le graphique nous montre belle et bien qu'on a pas de donné manquante dans le dataset
```



Statistique univariée

```
#Variable prix_Unitaire_Litre
```

```
df. PRIX_UNITAIRE_LITRE.describe()
#Le Prix_Unitaire_Litre compte 27 observations dans la colonne
# La moyenne des valeurs dans la colonne est d'environ 3.60
# std (écart-type): C'est une mesure de la dispersion des valeurs par rapport à la moyenne. Un écart-type élevé indique une dispersion i
#Ici, l'écart-type est d'environ 10.20, ce qui indique une variation assez importante dans les prix unitaires par litre.
# *min: C'est la valeur la plus basse dans la colonne. Dans ce cas, le prix unitaire le plus bas est d'environ 1.46.
# 25% (premier quartile): Cela signifie que 25% des données sont inférieures à cette valeur. Le premier quartile est à environ 1.52.
# 50% (médiane): C'est la valeur médiane de la distribution des données, c'est-à-dire que la moitié des données sont inférieures à cette
# 75% (troisième quartile): Cela signifie que 75% des données sont inférieures à cette valeur. Le troisième quartile est à environ 1.75.
# max : C'est la valeur la plus élevée dans la colonne. Dans ce cas, le prix unitaire le plus élevé est d'environ 54.64.
```

```
count    27.000000
mean      3.599963
std      10.201322
min       1.462000
25%       1.518000
50%       1.626000
75%       1.749000
max      54.640000
Name: PRIX_UNITAIRE_LITRE, dtype: float64
```

```
df['PRIX_UNITAIRE_LITRE'].value_counts()
#pour voir le nombre de Prix_Unitaire_Litre qui ont les memes Prix_Unitaire_Litre
```

```
1.462     3
1.504     2
```

```
1.731     1
1.781     1
1.783     1
1.573     1
1.509     1
54.640    1
1.626     1
1.720     1
1.614     1
1.638     1
1.604     1
1.527     1
1.487     1
1.542     1
1.695     1
1.867     1
1.654     1
1.657     1
1.929     1
1.602     1
1.859     1
1.767     1
Name: PRIX_UNITAIRE_LITRE, dtype: int64
```

```
#On va regrouper les Volume_Carburant selon leur Prix_Unitaire_Litre
df.groupby(['VOLUME_CARBURANT']).mean()
#On voit que les Prix_Unitaire_Litre  varient en fonction du  Volume_Carburant
```

```
<ipython-input-40-f585a682fa48>:2: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a fu
  df.groupby(['VOLUME_CARBURANT']).mean()
```

VOLUME_CARBURANT	ID_PAYEMENT	PRIX_UNITAIRE_LITRE	TOTAL_TTC	TOTAL_NET	ANNEE
11.78l	26.0	1.783	21.000	17.50	2023.0
13.39l	11.0	1.867	25.000	20.83	2023.0
13.69l	3.0	1.462	20.010	17.25	2023.0
13.89l	5.0	1.462	20.310	17.51	2024.0
16.97l	17.0	1.781	30.220	25.18	2023.0
18.32l	19.0	1.638	30.010	25.87	2023.0
20.22l	1.0	1.731	35.000	29.17	2023.0
21.30l	14.0	1.504	32.040	27.62	2023.0
23.02l	9.0	1.657	38.140	31.78	2024.0
24.83l	10.0	1.654	41.070	35.41	2023.0
26.51l	24.0	1.509	40.000	34.19	2023.0
29.93l	21.0	1.720	51.480	44.38	2023.0
33.36l	23.0	54.640	1.638	47.10	2023.0
33.66l	2.0	1.487	50.050	42.78	2023.0
35.66l	27.0	1.767	63.010	53.67	2023.0
35.69l	22.0	1.626	58.030	49.60	2023.0
37.31l	13.0	1.695	56.000	48.67	2023.0
39.57l	12.0	1.462	57.850	49.87	2023.0
41.47l	6.0	1.504	62.370	53.77	2024.0
42.12l	20.0	1.614	67.980	58.60	2023.0
43.74l	18.0	1.604	70.160	60.48	2023.0
43.92l	25.0	1.573	69.090	59.56	2023.0
49.81l	16.0	1.527	76.210	65.14	2023.0
5.3l	8.0	1.929	10.240	8.53	2023.0
50.65l	7.0	1.602	81.140	69.95	2024.0
6.31l	15.0	1.542	12.430	10.72	2024.0
8.10l	4.0	1.859	15.060	12.55	2023.0

```
df. PRIX_UNITAIRE_LITRE.var()
# Renvoie la variance du Prix_Unitaire_Litre,  cette variance peut être interprétée comme une mesure de dispersion ou de variation dans
# On constate que  la variance est élevée, cela signifie que les prix unitaires par litre varient considérablement autour de la moyenne
```

```
104.06697972934474
```

```
df.PRIX_UNITAIRE_LITRE.median()
# Renvoie le mediane des statistiques du Prix_Unitaire_Litre, c'est-à-dire que la moitié des données sont inférieures à cette valeur et
```

```
1.626
```

```
df.PRIX_UNITAIRE_LITRE.skew()-0
# on constact que le coefficient de skewness est positif
# conclusion: La distribution est asymétrique vers la droite
```

```
5.194720981047423
```

```
df.PRIX_UNITAIRE_LITRE.kurtosis()-3
#  on constact que le coefficient de kurtosis est positif donc > à 3
# conclusion: La distribution est pointue
```

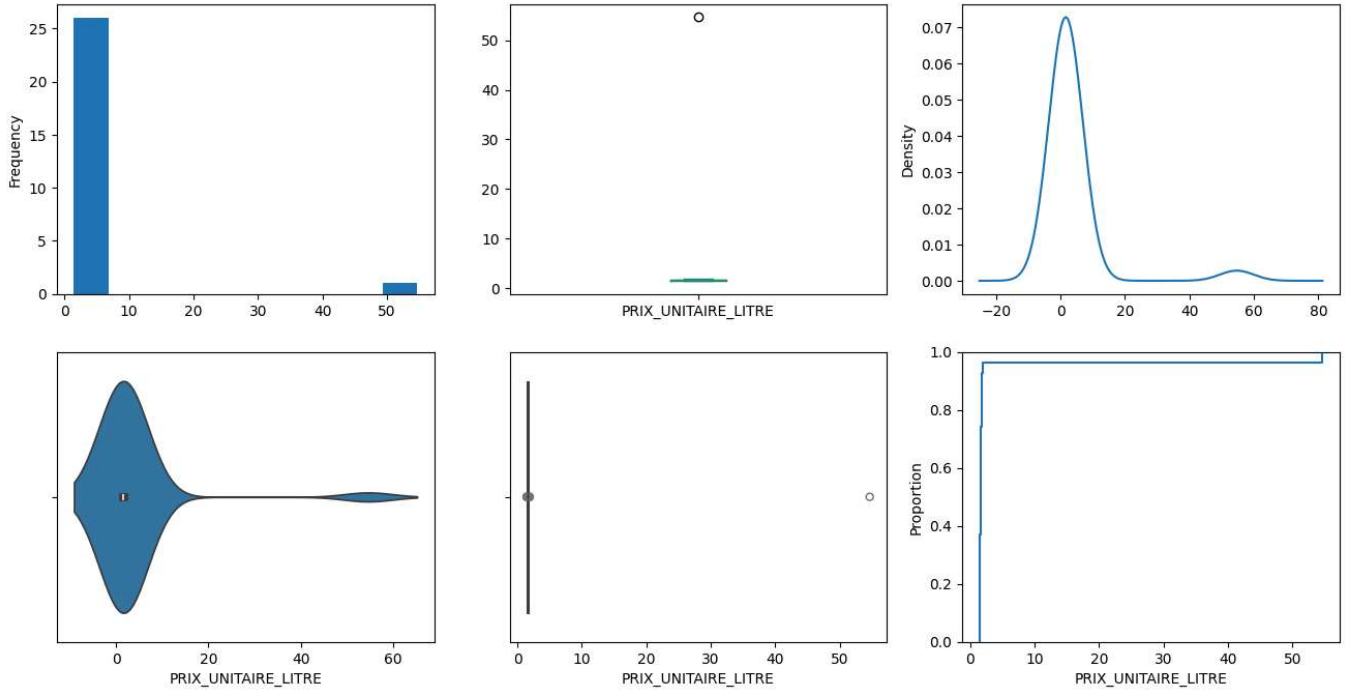
```
23.989706598961696
```

```
taux_de_dispersion=df.PRIX_UNITAIRE_LITRE.std()/df.Prix_Unitaire_Litre.mean()*100
taux_de_dispersion
# On se rend compte que le taux de dispersion > 0.15
# conclusion: la distribution est hétérogène c'est à dire que la variable Prix_Unitaire_Litre  est dispersée
```

```
283.3729834161082
```

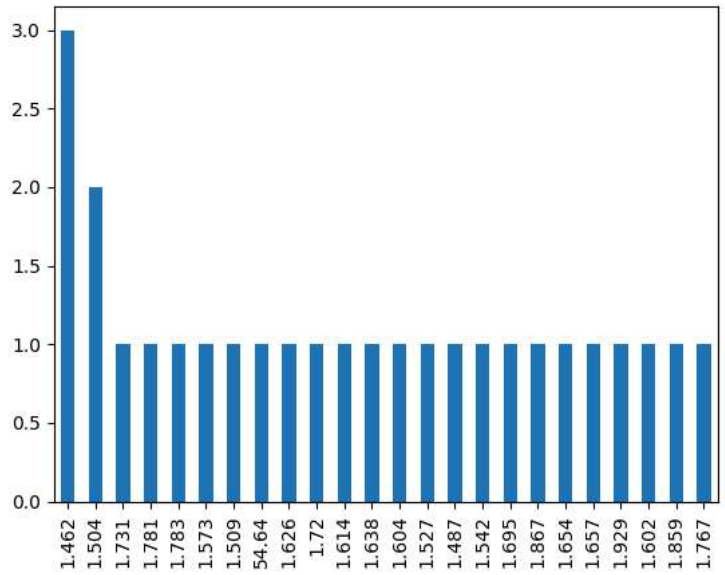
```
from os import stat
def monanalyse(data):
    e=1e-10
    stat=pd.DataFrame()
    dfnum = data.select_dtypes(exclude='object')
    stat=dfnum.describe().T
    stat["var"]=dfnum.var()
    stat["erreur_type"]=stat["std"]/np.sqrt(dfnum.shape[0])
    stat["skewness"]=dfnum.skew()-0
    stat["kurtosis"]=dfnum.kurtosis()-3
    stat["taux_dispersion_avec_moyen"]=stat["std"]/stat["mean"]*100
    stat["taux_dispersion_avec_median"]=(stat["75%"]-stat["25%"])/(stat["50%"]+e)*100

    stat["outmax"]=stat["75%"]+1.5*(stat["75%"]-stat["25%"])  #Q3+1.5*IQR
    stat["outmin"]=stat["25%"]-1.5*(stat["75%"]-stat["25%"])  #Q1-1.5*IQR

    stat["taux_outliers"]=((dfnum>stat["75%"]+1.5*(stat["75%"]-stat["25%"]))
                        | (dfnum<stat["25%"]-1.5*(stat["75%"]-stat["25%"]))).sum()/dfnum.shape[0]*100
    stat["taux_manquante"]=dfnum.isna().sum()/dfnum.shape[0]*100

    stat["unique"]=dfnum.nunique()#/dfnum.shape[0]*100
    return stat




def plot_univarie_quantitative(df, col):
    fig, st_axis = plt.subplots(2, 3, figsize=(16,8))
    #plt.title("Eboulis des valeurs propres")

    df.PRIX_UNITAIRE_LITRE.plot(kind="hist",ax = st_axis[0,0])
    df.PRIX_UNITAIRE_LITRE.plot(kind="box",ax = st_axis[0,1])
    df.PRIX_UNITAIRE_LITRE.plot(kind="density",ax = st_axis[0,2])

    sns.violinplot(x=col,data=df,ax = st_axis[1,0])
    sns.boxenplot(x=col,data=df,ax = st_axis[1,1])
    sns.ecdfplot(x=col, data=df, complementary=False, ax = st_axis[1,2])
    print(monanalyse(df).T[col])
    #ts_ax = plt.subplot2grid(layout, (0, 0), colspan=2)
    #ts_ax.set_title('essai')

    return None
plot_univarie_quantitative(df, "PRIX_UNITAIRE_LITRE")
```

```
count                          27.000000
mean                            3.599963
std                            10.201322
min                             1.462000
25%                             1.518000
50%                             1.626000
75%                             1.749000
max                            54.640000
var                           104.066980
erreur_type                     1.963245
skewness                        5.194721
kurtosis                       23.989707
taux_dispersion_avec_moyen    283.372983
taux_dispersion_avec_median    14.206642
outmax                          2.095500
outmin                          1.171500
taux_outliers                   3.703704
taux_manquante                  0.000000
unique                         24.000000
Name: PRIX_UNITAIRE_LITRE, dtype: float64
```




```
#L'histogramme nous montre que les PRIX_Unitaire_Litre


df['PRIX_UNITAIRE_LITRE'].value_counts().plot.bar()
#Chaque barre dans le diagramme représente une valeur unique de Prix_Unitaire_Litre, et la hauteur de chaque barre indique combien de fo
#Cela vous donne une idée de la distribution des prix unitaires par litre dans vos données.
```

```
<Axes: >
```



ETUDE DE LA VARIABLE CATEGORIELLE Volume_Carburant

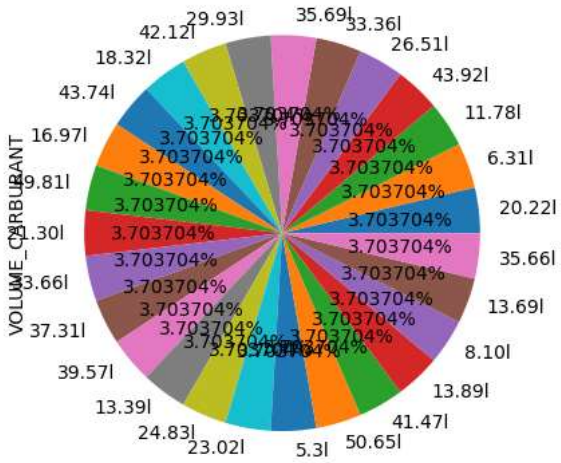
```
df.VOLUME_CARBURANT.value_counts()
# Pour regarder les valeurs uniques de la colonne Volume_Carburant et de voir combien de fois chaque valeur apparaît. Cela peut vous dor
```

```
20.22l    1
6.31l     1
11.78l    1
43.92l    1
26.51l    1
33.36l    1
35.69l    1
29.93l    1
42.12l    1
18.32l    1
43.74l    1
16.97l    1
49.81l    1
21.30l    1
33.66l    1
37.31l    1
39.57l    1
13.39l    1
24.83l    1
23.02l    1
5.3l      1
50.65l    1
41.47l    1
13.89l    1
8.10l     1
13.69l    1
35.66l    1
Name: VOLUME_CARBURANT, dtype: int64
```

```
df.VOLUME_CARBURANT.describe()
#On a 27 entrées dans la colonne Volume_Carburant  , avec 27 valeurs uniques
#Dont le meilleur est celui de 20.22l
```

```
count         27
unique        27
top       20.22l
freq           1
Name: VOLUME_CARBURANT, dtype: object
```

```
def pie(df, var):
    df[var].value_counts().plot(kind="pie", autopct='%4.6f%%')
pie(df,"VOLUME_CARBURANT")
# Ce diagrame en camembert nous montre la répartition des différentes valeurs de la variable Volume_Carburant.
# Chaque secteur représente une valeur unique de cette variable, et sa taille est proportionnelle à la fréquence de cette valeur par rap
# Cela nous  permet de voir la contribution de chaque valeur à l'ensemble des données, exprimée en pourcentage.
```



STATTISTIQUE GENERALE ET UNFERENTIELLE

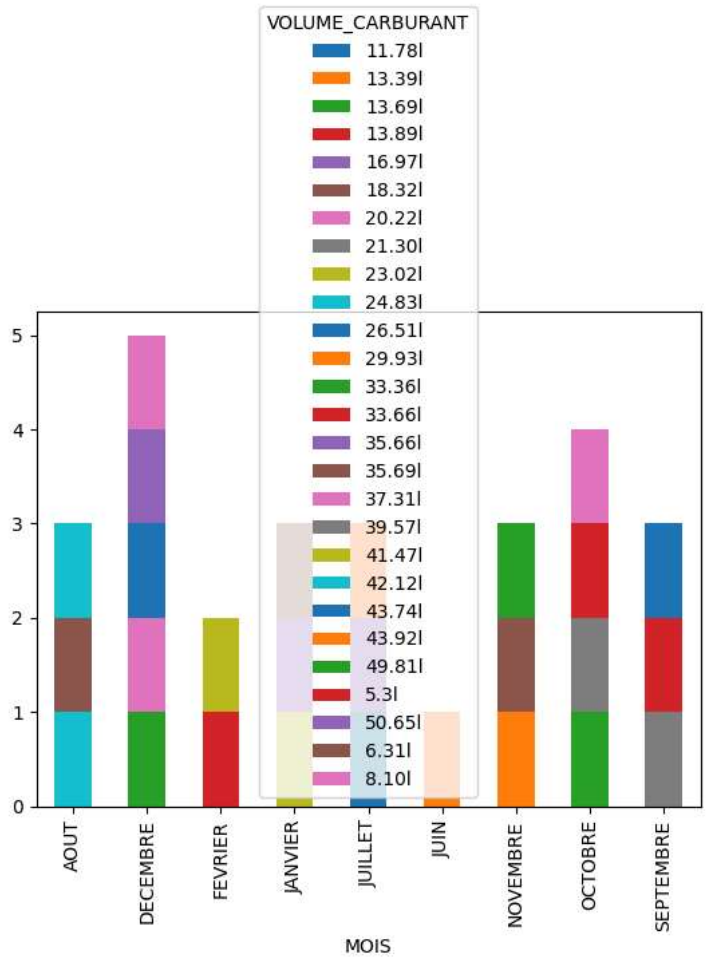
```
#Etude des variables catégorielles VOLUME8_CARBURANT_LITRE  et MOIS

# Tableau de contergence

t=pd.crosstab(df.VOLUME_CARBURANT , df.MOIS)
t

#Ce tableau de contergence nous montre combien le client a consommé chaque mois, on peut bien voir qu'au mois de JANVIER il n'a pas  cor
```

MOIS VOLUME_CARBURANT	AOUT	DECEMBRE	FEVRIER	JANVIER	JUILLET	JUIN	NOVEMBRE	OCTOBRE	SEPTEMBRE
11.78l	0	0	0	0	1	0	0	0	0
13.39l	0	0	0	0	0	0	1	0	0
13.69l	0	1	0	0	0	0	0	0	0
13.89l	0	0	1	0	0	0	0	0	0
16.97l	0	0	0	0	1	0	0	0	0
18.32l	0	0	0	0	0	0	1	0	0
20.22l	0	1	0	0	0	0	0	0	0
21.30l	0	0	0	0	0	0	0	0	1
23.02l	0	0	1	0	0	0	0	0	0
24.83l	1	0	0	0	0	0	0	0	0
26.51l	0	1	0	0	0	0	0	0	0
29.93l	0	0	0	0	0	1	0	0	0
33.36l	0	0	0	0	0	0	0	1	0
33.66l	0	0	0	0	0	0	0	0	1
35.66l	0	1	0	0	0	0	0	0	0
35.69l	1	0	0	0	0	0	0	0	0
37.31l	0	1	0	0	0	0	0	0	0
39.57l	0	0	0	0	0	0	0	1	0
41.47l	0	0	0	1	0	0	0	0	0
42.12l	1	0	0	0	0	0	0	0	0
43.74l	0	0	0	0	0	0	0	0	1
43.92l	0	0	0	0	1	0	0	0	0
49.81l	0	0	0	0	0	0	1	0	0
5.3l	0	0	0	0	0	0	0	1	0
50.65l	0	0	0	1	0	0	0	0	0
6.31l	0	0	0	1	0	0	0	0	0
8.10l	0	0	0	0	0	0	0	1	0

Next steps: View recommended plots

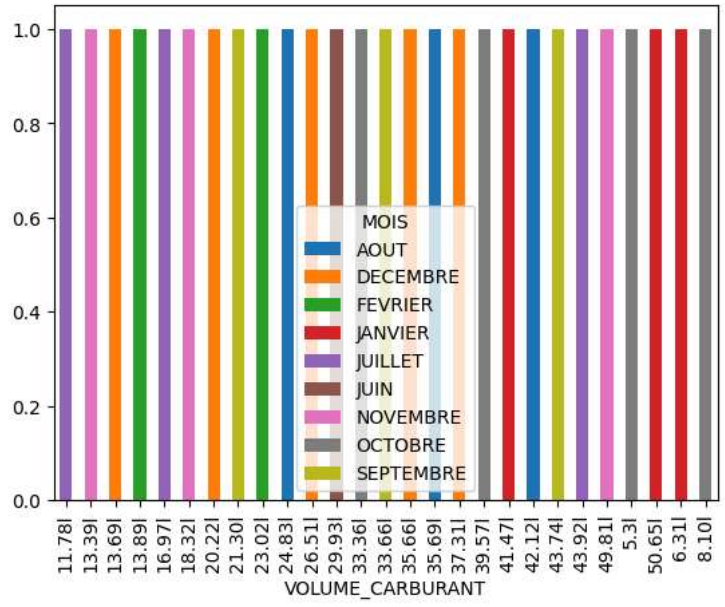
```
t=pd.crosstab(df.MOIS, df.VOLUME_CARBURANT)
t.plot.bar(stacked=True)
#INTERPRETATION: Ce graphe en bar nous montre très clairement que le client a plus consommé le carburant au mois Decembre, suivi du mois
#Janvier,Juillet, Fevrier enfin Juin
```

```
<Axes: xlabel='MOIS'>
```




```
t=pd.crosstab(df.VOLUME_CARBURANT, df.MOIS)
t.plot.bar(stacked=True)
```

```
<Axes: xlabel='VOLUME_CARBURANT'>
```




```
#TEST de KI2 pour tester la dependance entre nos 2 variables qualitatives


#On pose les hypothèses de départ :

#H0 : Variables indépendantes si p-value > 5%
#H1 : Variables non indépendantes si p-value < 5%


from scipy.stats import chi2_contingency
Khi2_obs, p_value, ddl, effectif_theorique = chi2_contingency(t)
```

```
p_value
# P_Value = 0.24 > 0.05 donc on accepte l'hypothèse que la dépendance entre le Volume_Carburant  et les  MOIS  est le fruit du hazard
#Donc il y a  une dépendance entre Volume_Carburant et MOIS
```

```
0.2409926945034041
```

```
#Hypothèses H0: Les pays sont independants des régions
import numpy as np
from scipy import stats
tableau_contingency=pd.crosstab(df.VOLUME_CARBURANT, df.MOIS)
tableau_contingency
stats.chi2_contingency(tableau_contingency, correction = True)
# P_Value = 0.33 > 0.05 donc on accepte l'hypothèse que la dépendance entre le Volume_Carburant  et les  MOIS  est le fruit du hazard
#Donc il y a  une dépendance entre Volume_Carburant et MOIS
```

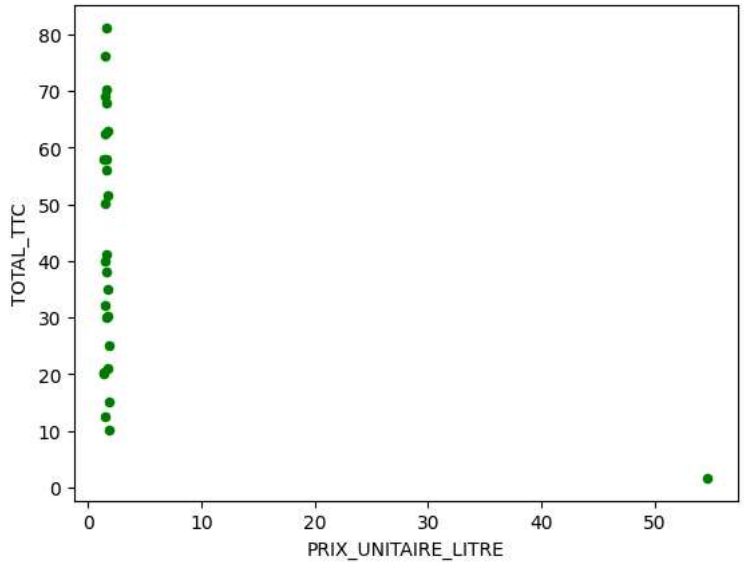
```
Chi2ContingencyResult(statistic=216.0, pvalue=0.33727791207689345, dof=208, expected_freq=array([[0.11111111, 0.18518519,
0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111],
       [0.11111111, 0.18518519, 0.07407407, 0.11111111, 0.11111111,
        0.03703704, 0.11111111, 0.14814815, 0.11111111]]))
```

Étudiez les associations entre variables quantitatives (visualisation et test)

```
#NOus allons étudier l'association entre les PRIX_UNITAIRE_LITRE  et TOTAL_TTC
```

```
df.plot.scatter("PRIX_UNITAIRE_LITRE","TOTAL_TTC", color = "green")
#On voit que  les points forment une tendance ascendante, cela suggère une relation positive entre le prix unitaire par litre et le tota
#Cela signifie que lorsque le prix par litre augmente, le total TTC a tendance à augmenter également.
```

```
<Axes: xlabel='PRIX_UNITAIRE_LITRE', ylabel='TOTAL_TTC'>
```



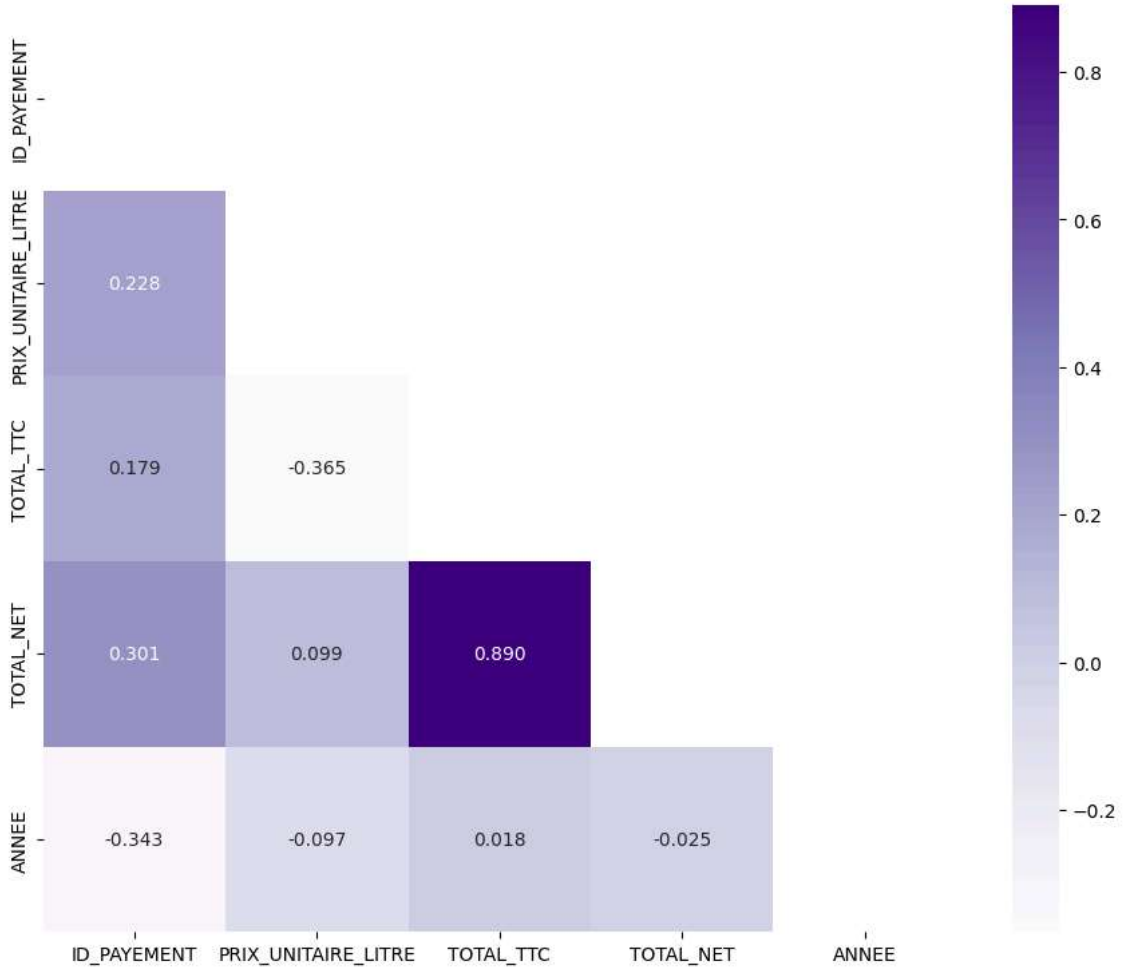

```
df.corr()

#Entre le PRIX_UNITAIRE_LITRE et TOTAL_TTC on voit que la correlation est de -0,36 ce qui indique une correlation négative  cela signifi
#ont tendance à évoluer dans des directions opposées.
#Entre PRIX_UNITAIRE_LITRE et  TOTAL_NET : La corrélation est de 0.09, suggérant une relation positive.cela signifie que les variables c
#Cela signifie que les logements avec des charges plus élevées ont tendance à avoir des prix plus élevés.
#Entre TOTAL_TTC et TOTAL_NET : La corrélation est de 0.88, indiquant une relation positive .cela signifie que les variables ont tendanc
```

```
<ipython-input-73-2f6f6606aa2c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
  df.corr()
```

	ID_PAYEMENT	PRIX_UNITAIRE_LITRE	TOTAL_TTC	TOTAL_NET	ANNEE
ID_PAYEMENT	1.000000	0.227851	0.178923	0.301122	-0.342757
PRIX_UNITAIRE_LITRE	0.227851	1.000000	-0.365313	0.099257	-0.097462
TOTAL_TTC	0.178923	-0.365313	1.000000	0.889776	0.017844
TOTAL_NET	0.301122	0.099257	0.889776	1.000000	-0.025321
ANNEE	-0.342757	-0.097462	0.017844	-0.025321	1.000000

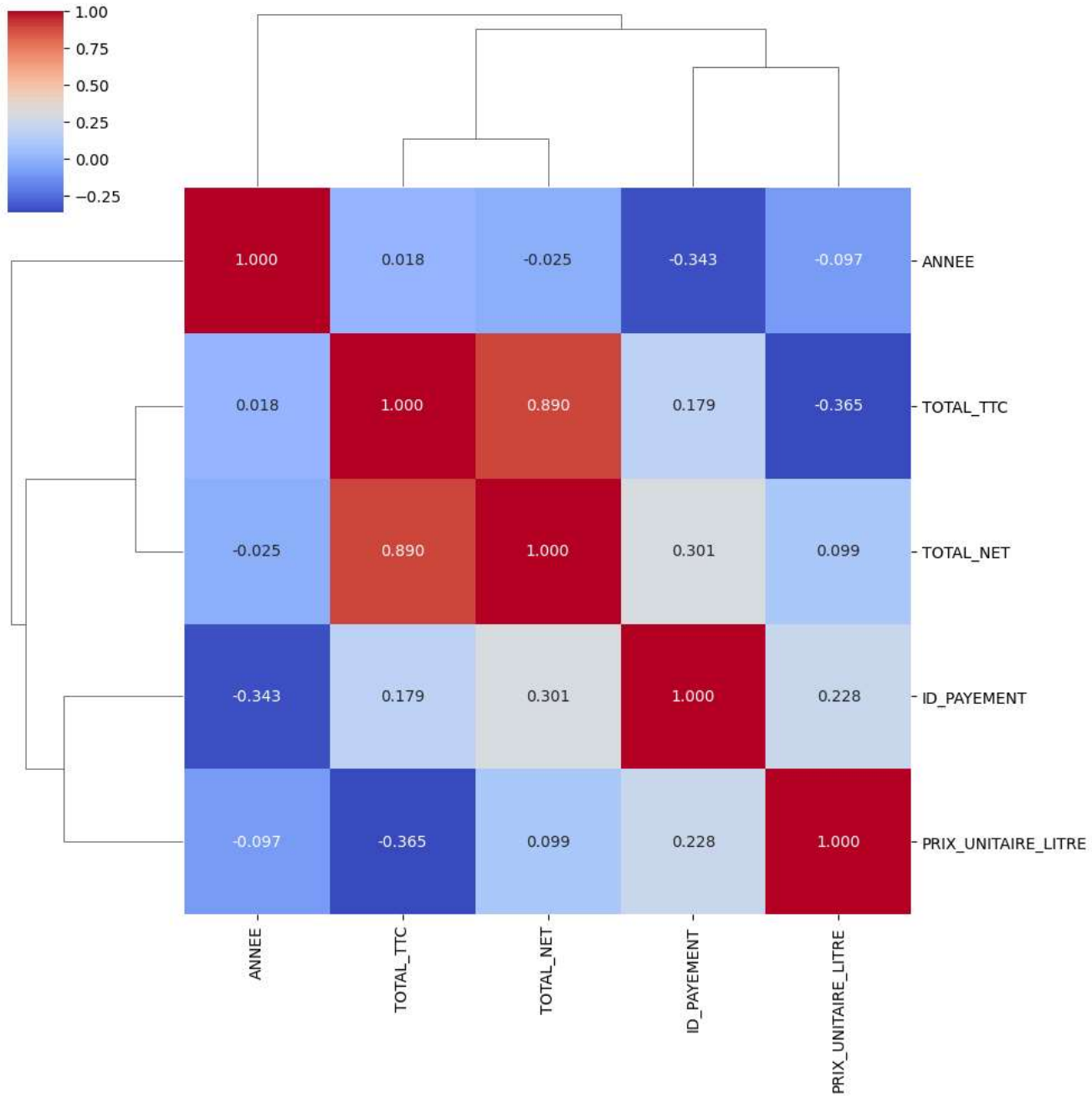
```
mask = np.triu(np.ones_like(df.corr(), dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(df.corr(), mask=mask,annot=True,fmt="1.3f",cmap='Purples')
# on comprend bien de quoi il s’agît. Les couleurs sont bien choisies du plus foncé pour une forte corrélation au plus clair pour une fa
```

```
<ipython-input-74-bf42ac5c1570>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
  mask = np.triu(np.ones_like(df.corr(), dtype=bool))
<ipython-input-74-bf42ac5c1570>:7: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
  sns.heatmap(df.corr(), mask=mask,annot=True,fmt="1.3f",cmap='Purples')
<Axes: >
```



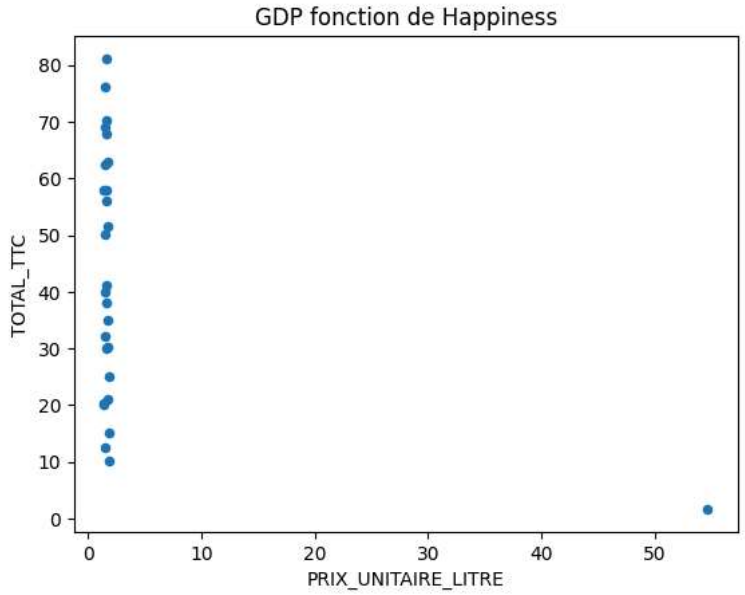

```
#H clustermap
def clustermap(df):
    df_num=df.select_dtypes(exclude=['object'])
    plt.figure(figsize=(8, 8))
    sns.clustermap(df_num.corr(),annot=True,fmt="1.3f",cmap='coolwarm')
    return None
clustermap(df)
# INTERPRETATION:
```

```
<Figure size 800x800 with 0 Axes>
```




```
df.plot.scatter(x="PRIX_UNITAIRE_LITRE", y="TOTAL_TTC",title="PRIX_UNITAIRE_LITRE fonction de TOTAL_TTC")
#INTERPRETATION: La courbe sugerre une faible correlation entre PRIX_UNITAIRE_LITRE et TOTAL_TTC
```

```
<Axes: title={'center': 'GDP fonction de Happiness'}, xlabel='PRIX_UNITAIRE_LITRE', ylabel='TOTAL_TTC'>
```




```
 #Test statistique de pearson pour prendre une decision sur la dépendance
scipy.stats.pearsonr(df.PRIX_UNITAIRE_LITRE, df.TOTAL_TTC)
#P_Value = 0.06 > 0.05 donc accepte l'hypothèse que la dépendance entre PRIX_UNITAIRE_LITRE et TOTAL_TTC est le fruit du hazard
#Donc qu'il n'y a pas une dépendance entre ces deux variables
```

```
PearsonRResult(statistic=-0.365313369970005, pvalue=0.06096404558831618)
```

ETUDE DES ASSOCIATIONS ENTRE VARIABLE QUANTITATIVE ET VARIABLE CATEGORIELLES

```
# Nous  allons choisir la variable TOTAL_TTC et et la variable  MOIS


H_columns = ['TOTAL_TTC']
RH_columns = ['MOIS', 'TOTAL_TTC']
df_h = pd.DataFrame(df, columns=RH_columns)
df_h

#ce code extrait uniquement les colonnes 'MOIS' et 'TOTAL_TTC' du DataFrame original df pour les stocker dans un nouveau DataFrame appel
```

	MOIS	TOTAL_TTC
0	DECEMBRE	35.000
1	SEPTEMBRE	50.050
2	DECEMBRE	20.010
3	OCTOBRE	15.060
4	FEVRIER	20.310
5	JANVIER	62.370
6	JANVIER	81.140
7	OCTOBRE	10.240
8	FEVRIER	38.140
9	AOUT	41.070
10	NOVEMBRE	25.000
11	OCTOBRE	57.850
12	DECEMBRE	56.000
13	SEPTEMBRE	32.040
14	JANVIER	12.430
15	NOVEMBRE	76.210
16	JUILLET	30.220
17	SEPTEMBRE	70.160
18	NOVEMBRE	30.010
19	AOUT	67.980
20	JUIN	51.480
21	AOUT	58.030
22	OCTOBRE	1.638
23	DECEMBRE	40.000
24	JUILLET	69.090
25	JUILLET	21.000
26	DECEMBRE	63.010

Next steps: View recommended plots

```
df.groupby("MOIS").mean()["TOTAL_TTC"].sort_values(ascending=False)
#On voit bien les mois ou le client a plus consommé le carburant par ordre croisssant
```

```
<ipython-input-80-245b70734aa0>:1: FutureWarning: The default value of numeric_only in DataFrameGroupBy.mean is deprecated. In a fut
  df.groupby("MOIS").mean()["TOTAL_TTC"].sort_values(ascending=False)
MOIS
AOUT         55.693333
JANVIER      51.980000
JUIN         51.480000
SEPTEMBRE    50.750000
NOVEMBRE     43.740000
DECEMBRE     42.804000
JUILLET      40.103333
FEVRIER      29.225000
OCTOBRE      21.197000
Name: TOTAL_TTC, dtype: float64
```

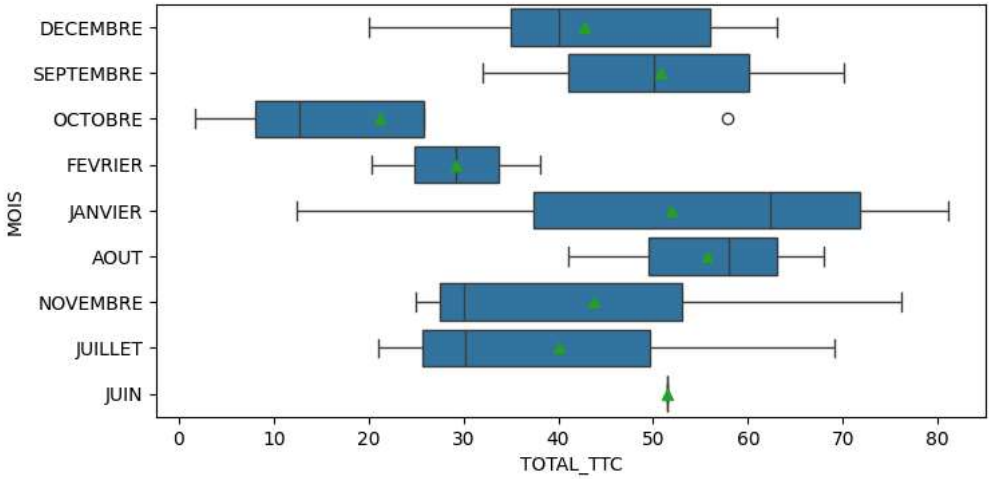
```
#regardons les pourcentage des prix selon le Garage
df.groupby("MOIS").sum()["TOTAL_TTC"]/df.TOTAL_TTC.sum()*100 #pourcentage
```

```
<ipython-input-82-ebd2281458ac>:2: FutureWarning: The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a futu
  df.groupby("MOIS").sum()["TOTAL_TTC"]/df.TOTAL_TTC.sum()*100 #pourcentage
MOIS
AOUT         14.713730
DECEMBRE     18.847454
FEVRIER       5.147340
JANVIER      13.732698
JUILLET      10.594978
JUIN          4.533534
NOVEMBRE     11.555756
OCTOBRE       7.466769
```

```
SEPTEMBRE    13.407742
Name: TOTAL_TTC, dtype: float64
```

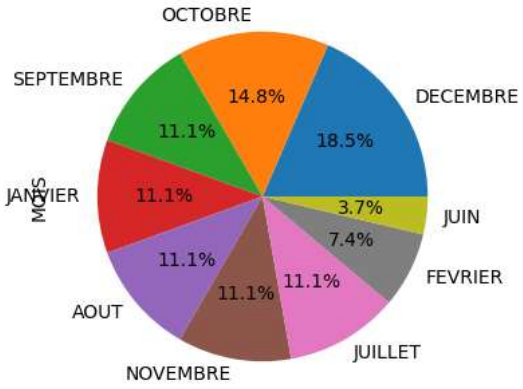
```
fig, ax = plt.subplots(figsize=(8,4))  #1 inch = 2,54 centimètres

sns.boxplot(x='TOTAL_TTC',y="MOIS",showmeans=True,showfliers=True,data=df)
#A cette etape, on ne peut pas affirmer qu'il y a pas une dependance entre TOTAL_TTC et MOIS, il faudra analyser et faire des tests!
```

```
<Axes: xlabel='TOTAL_TTC', ylabel='MOIS'>
```



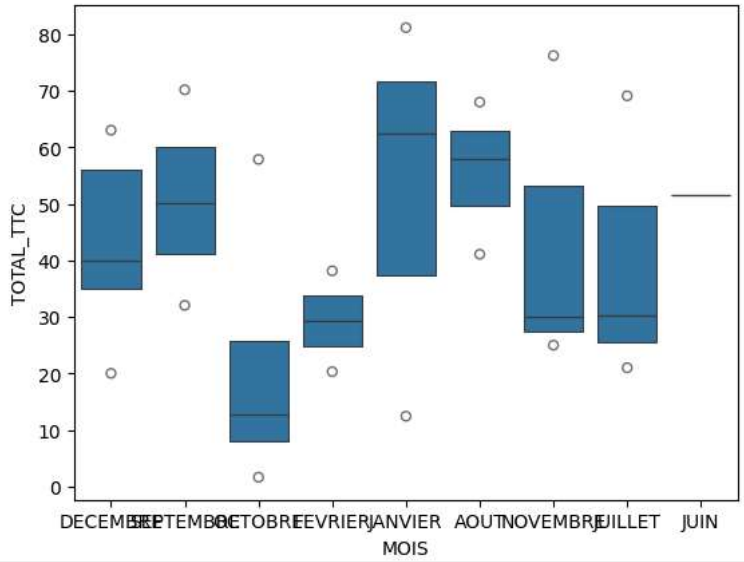

```
fig, ax = plt.subplots(figsize=(4,4))

df.MOIS.value_counts().plot(kind='pie', autopct='%1.1f%%')
#On voit que Decembre est le mois le plus consommé
```

```
<Axes: ylabel='MOIS'>
```



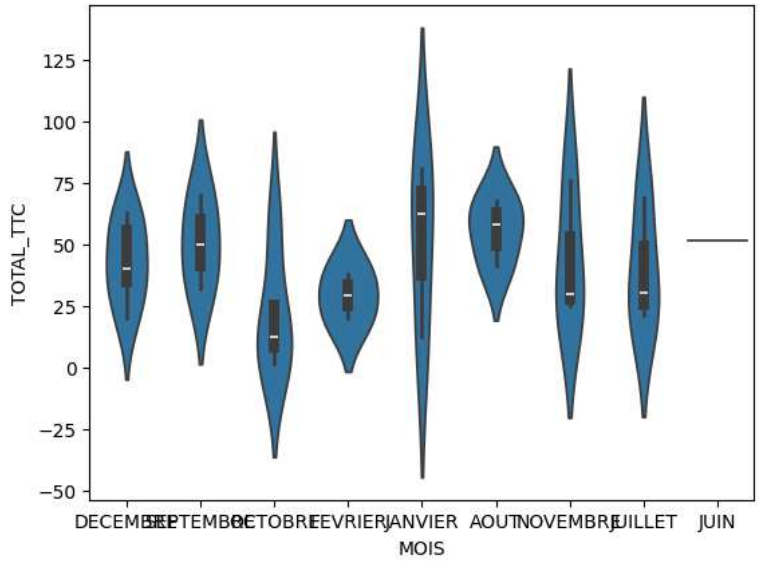

```
sns.boxenplot(x="MOIS", y="TOTAL_TTC",
              scale="linear", data=df)
```

```
<ipython-input-86-84090afa2a79>:1: FutureWarning:

The `scale` parameter has been renamed to `width_method` and will be removed in v0.15. Pass `width_method='linear' for the same effe
  sns.boxenplot(x="MOIS", y="TOTAL_TTC",
<Axes: xlabel='MOIS', ylabel='TOTAL_TTC'>
```




```
sns.violinplot(x="MOIS", y="TOTAL_TTC", data=df)
```

```
<Axes: xlabel='MOIS', ylabel='TOTAL_TTC'>
```




```
#Vérification de la dépendance entre les Prix et Garage
#H0: "Garage et Prix sont INDEPENDANTS"
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Ordinary Least Squares (OLS) model
model = ols('TOTAL_TTC ~MOIS', data=df).fit()
anova_table = sm.stats.anova_lm(model)
anova_table
#Vérification: "TOTAL_TTC et MOIS sont INDEPENDANTS" car p-value(PR(>F))= 5.880809e-22 >0.05


#si p-value(PR(>F))>0.05 (seuil de moralite) alors il y a pas dependance entre les deux variables
```

	df	sum_sq	mean_sq	F	PR(>F)
MOIS	8.0	3261.361024	407.670128	0.754146	0.64557
Residual	18.0	9730.290411	540.571690	NaN	NaN

Next steps: View recommended plots

```
from scipy.stats import f_oneway, kruskal
label=list(df.MOIS.unique())
data_a = [df[df["MOIS"] == a].TOTAL_TTC for a in label]

# Anova Test
stat1, p1 = f_oneway(*data_a)
print('---ANOVA---')
print('stat={:.3f}, pvalue={}'.format(stat1, p1))
print('Proba. same distr') if p1 > 0.05 else print('Prob. different distr')
print("")
stat2, p2 = kruskal(*data_a)
print('---Kruskal-Wallis---')
print('stat={:.3f}, pvalue={}'.format(stat2, p2))
print('Prob. same distr') if p2 > 0.05 else print('Prob. different distr')

#p-value(PR(>F))>0.05 (seuil de moralite) alors il y a pas dependance entre les deux variables
```

```
---ANOVA---
stat=0.754, pvalue=0.6455703621872977
Proba. same distr

---Kruskal-Wallis---
stat=6.683, pvalue=0.5711807840235592
Prob. same distr
```

STANDARDISATION OU METTRE LES DONNEES AU MEME ECHELLES

```
import datetime
import sklearn


import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import scipy
from scipy import stats


df = pd.read_csv('payement_carburant (3).csv')
df
```

	ID_PAYEMENT	NOM_CARBURANT	DATE_PAYEMENT	HEURE_PAYEMENT	VOLUME_CARBURANT	PRIX_UNITAIRE_LITRE	TOTAL_TTC	TOTAL_NET	M
0	1	Diesel	2023-12-11	18:40:09	20.22l	1.731	35.000	29.17	DECEME
1	2	Diesel	2023-09-23	12:09:00	33.66l	1.487	50.050	42.78	SEPTEME
2	3	Diesel	2023-12-08	15:31:34	13.69l	1.462	20.010	17.25	DECEME
3	4	Diesel	2023-10-19	19:12:27	8.10l	1.859	15.060	12.55	OCTOE
4	5	Diesel	2024-02-15	17:16:30	13.89l	1.462	20.310	17.51	FEVR
5	6	Diesel	2024-01-11	19:02:32	41.47l	1.504	62.370	53.77	JANV
6	7	Diesel	2024-01-23	17:47:46	50.65l	1.602	81.140	69.95	JANV
7	8	Diesel	2023-10-04	18:20:10	5.3l	1.929	10.240	8.53	OCTOE
8	9	Diesel	2024-02-24	14:35:16	23.02l	1.657	38.140	31.78	FEVR
9	10	Diesel	2023-08-03	07:23:18	24.83l	1.654	41.070	35.41	AO
10	11	Diesel	2023-11-07	18:00:48	13.39l	1.867	25.000	20.83	NOVEME
11	12	Diesel	2023-10-24	18:00:08	39.57l	1.462	57.850	49.87	OCTOE
12	13	Diesel	2023-12-29	12:52:26	37.31l	1.695	56.000	48.67	DECEME
13	14	Diesel	2023-09-04	18:06:21	21.30l	1.504	32.040	27.62	SEPTEME
14	15	Diesel	2024-01-26	17:41:26	6.31l	1.542	12.430	10.72	JANV
15	16	Diesel	2023-11-17	18:41:09	49.81l	1.527	76.210	65.14	NOVEME
16	17	Diesel	2023-07-13	18:52:41	16.97l	1.781	30.220	25.18	JUIL
17	18	Diesel	2023-09-23	12:09:00	43.74l	1.604	70.160	60.48	SEPTEME
18	19	Diesel	2023-11-05	05:42:19	18.32l	1.638	30.010	25.87	NOVEME
19	20	Diesel	2023-08-31	18:33:52	42.12l	1.614	67.980	58.60	AO
20	21	Diesel	2023-06-28	12:33:36	29.93l	1.720	51.480	44.38	J
21	22	Diesel	2023-08-16	07:29:34	35.69l	1.626	58.030	49.60	AO
22	23	Diesel	2023-10-30	17:05:19	33.36l	54.640	1.638	47.10	OCTOE
23	24	Diesel	2023-12-14	18:13:14	26.51l	1.509	40.000	34.19	DECEME
24	25	Diesel	2023-07-16	13:25:17	43.92l	1.573	69.090	59.56	JUIL
25	26	Diesel	2023-07-21	15:40:09	11.78l	1.783	21.000	17.50	JUIL
26	27	Diesel	2023-12-10	16:12:54	35.66l	1.767	63.010	53.67	DECEME

Next steps: View recommended plots

```
df= df.dropna().drop_duplicates()
```

```
df.isna().sum().sum() # on a aucune donnéé manquante parfait
```

```
0
```

```
df.duplicated().sum()  # 0 donnée manquante
```

```
0
```

```
df.columns
```

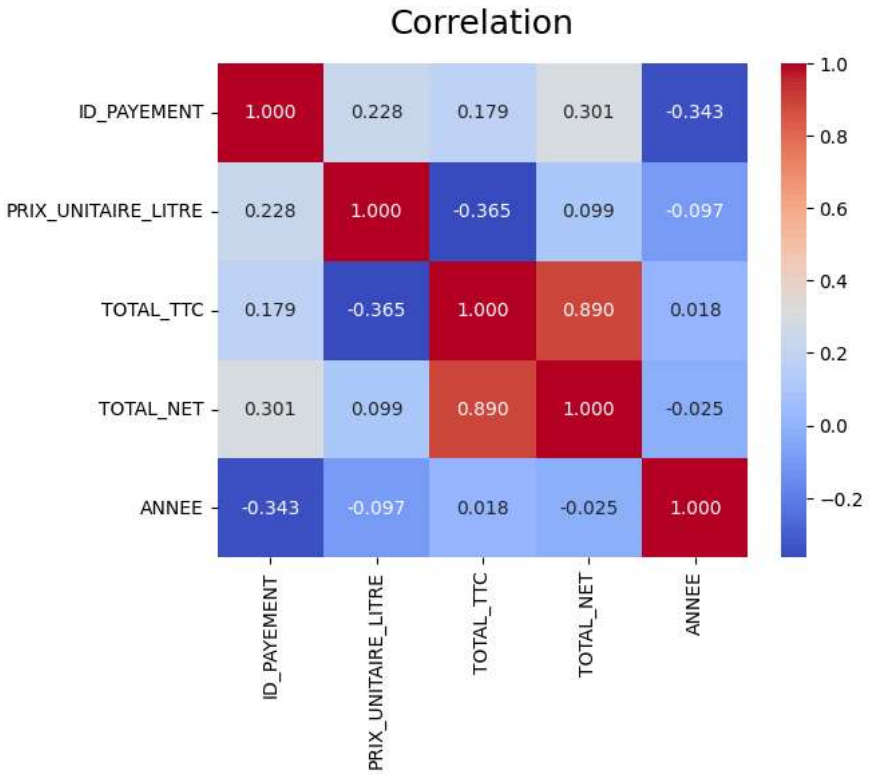
```
Index(['ID_PAYEMENT', 'NOM_CARBURANT', 'DATE_PAYEMENT', 'HEURE_PAYEMENT',
       'VOLUME_CARBURANT', 'PRIX_UNITAIRE_LITRE', 'TOTAL_TTC', 'TOTAL_NET',
       'MOIS', 'ANNEE'],
      dtype='object')
```

```
X = df.select_dtypes(include=['float64','int64'])
#LE PCA, comme tout modele IA, prend en entrees que des variables numeriques
X.head()
```

	ID_PAYEMENT	PRIX_UNITAIRE_LITRE	TOTAL_TTC	TOTAL_NET	ANNEE
0	1	1.731	35.00	29.17	2023
1	2	1.487	50.05	42.78	2023
2	3	1.462	20.01	17.25	2023
3	4	1.859	15.06	12.55	2023
4	5	1.462	20.31	17.51	2024

Next steps: View recommended plots

```
import seaborn as sns
heatmap=sns.heatmap(X.corr(),annot=True, fmt=".3f", cmap='coolwarm')
heatmap.set_title('Correlation', fontdict={'fontsize':18}, pad=16);
# On remarqu'il y a des FEATURES(variables explicatives) qui sont corrélées: les modeles detestent
```



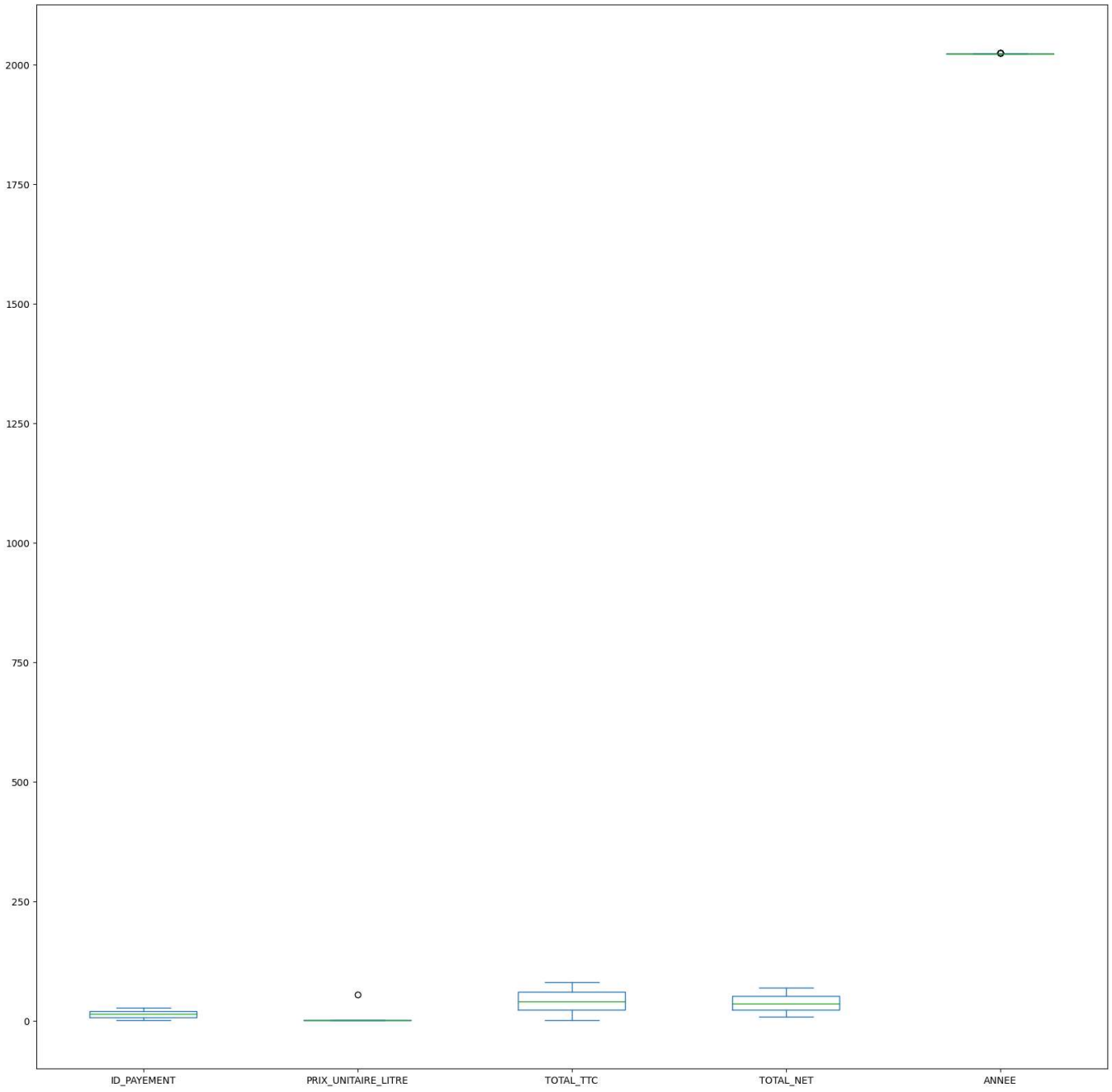

```
X.shape # apres nettoyage on 27 lignes et 5 colonnes
```

```
(27, 5)
```

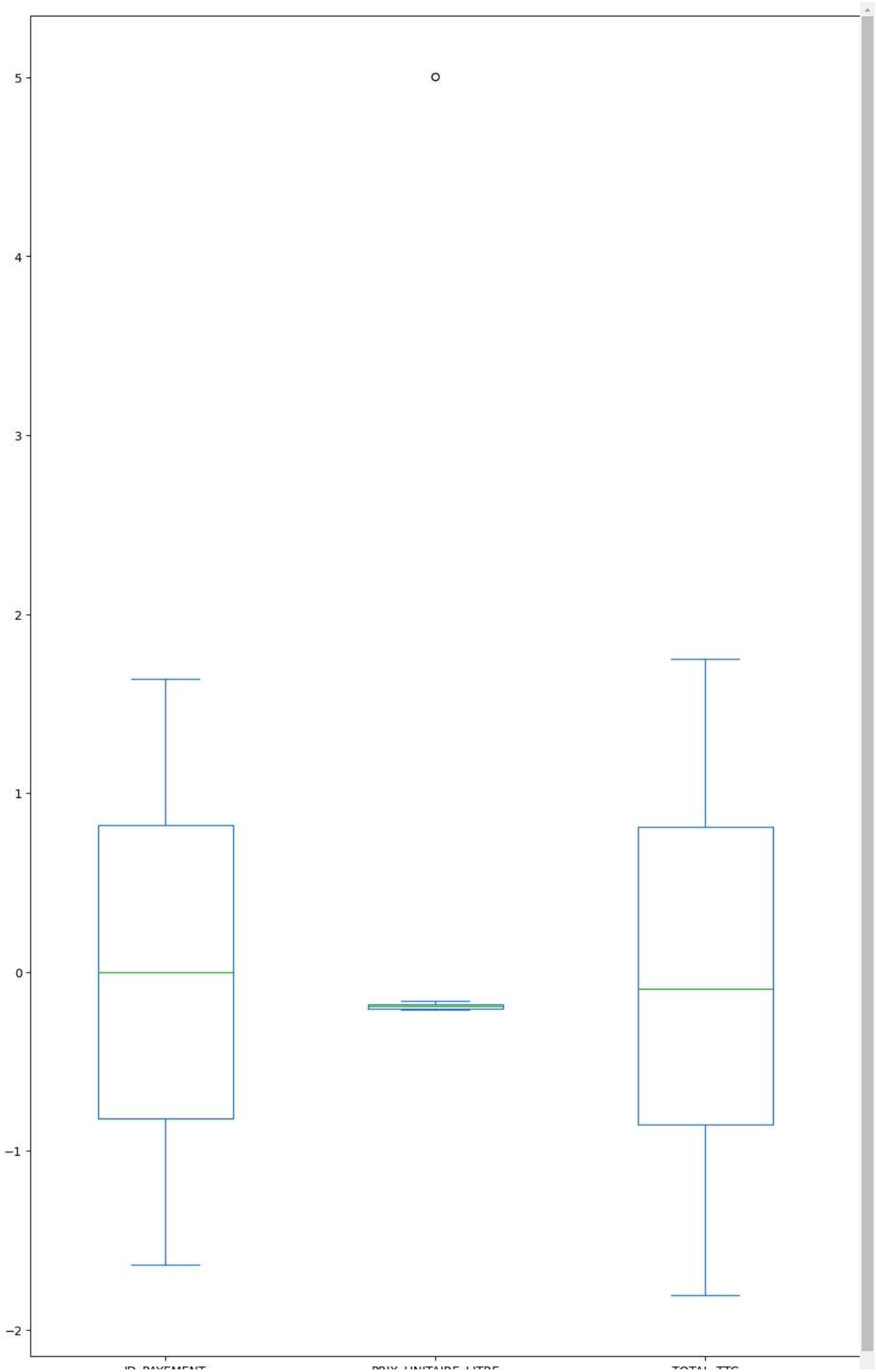
```
df.sem() /df.mean()*100  # pourcentage de l'écart-type par rapport à la moyenne

#pour évaluer la variabilité des données par rapport à leur moyenne, exprimée en pourcentage.
```

```
<ipython-input-22-5d254eef7ba8>:1: FutureWarning: The default value of numeric_only in DataFrame.sem is deprecated. In a future vers
  df.sem() /df.mean()*100
<ipython-input-22-5d254eef7ba8>:1: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future ver
  df.sem() /df.mean()*100
ID_PAYEMENT           10.910895
PRIX_UNITAIRE_LITRE    54.535156
TOTAL_TTC             10.228824
TOTAL_NET              9.265683
ANNEE                  0.003765
dtype: float64
```

```
X.plot(kind = 'box', subplots=False , layout=(2,5), sharex=True, sharey=True, figsize=(20,20))
plt.show()
# Le graphe montre belle et bien que nos données ne sont pas au meme echelle et la variable qui a la plus grande echelle est la variable
```


2000
1750
1500
1250
1000
750
500
250
0
ID_PAYEMENT
PRIX_UNITAIRE_LITRE
TOTAL_TTC
TOTAL_NET
ANNEE

```
# Nous allons standardiser nos varaibles pour qu'elles ont les memes echelles car on peut developper un modèle sans mettre au meme echell
XX=(X-X.mean())/X.std()
XX.plot(kind='box', subplots=False, layout=(5,7), sharex=True, sharey=True, figsize=(20,20))
plt.show()
# là nous remarquons que nos données on presque les memes echelles on est bon
```

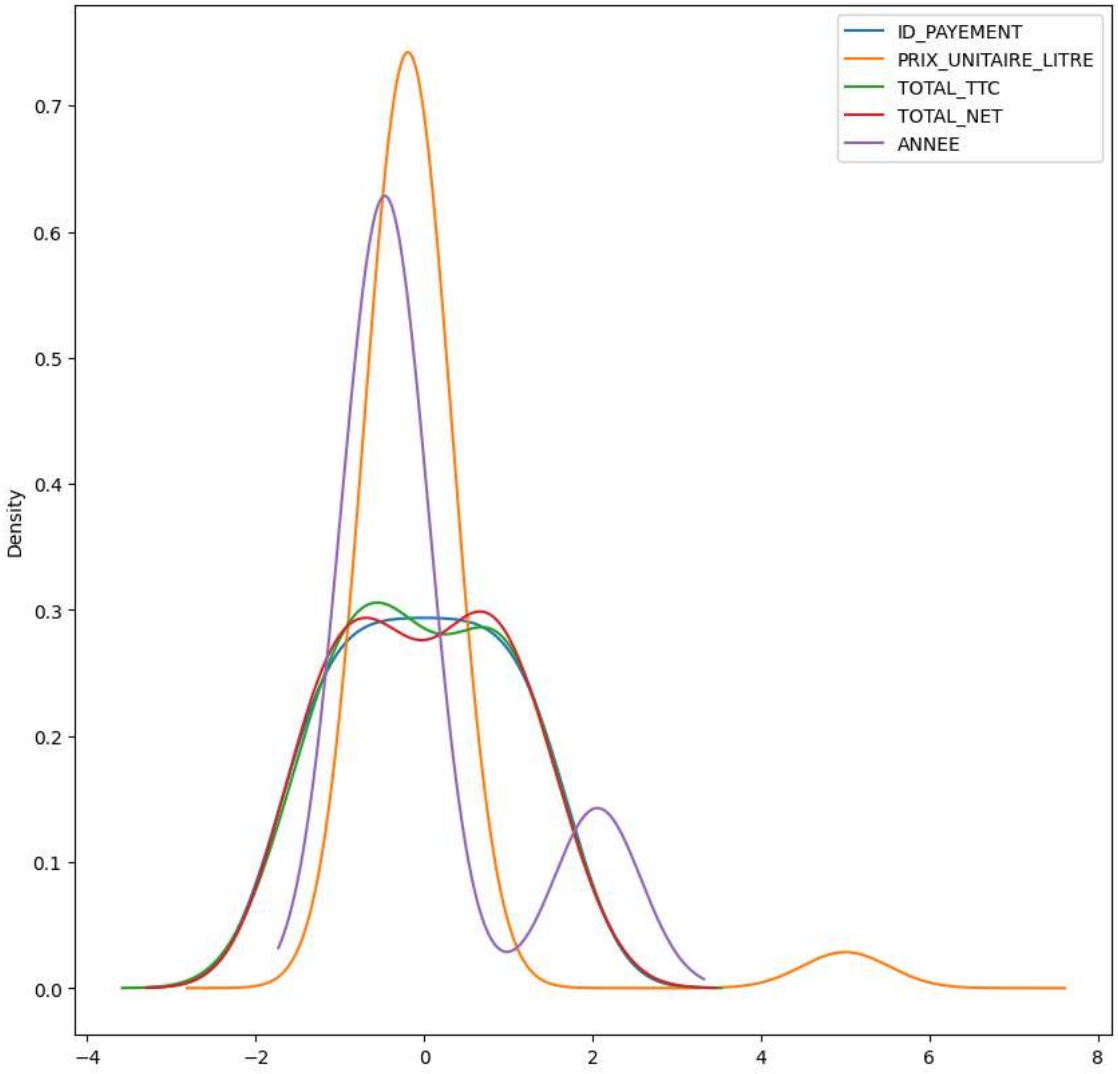

5
4
3
2
1
0
−1
−2

ID_PAYEMENT PRIX_UNITAIRE_LITRE TOTAL_TTC

```
XX.head()
# Toutes les informations obéissent aux meme regle dc elles sont aux memes echelles
```

	ID_PAYEMENT	PRIX_UNITAIRE_LITRE	TOTAL_TTC	TOTAL_NET	ANNEE
0	-1.637846	-0.183208	-0.315698	-0.469599	-0.467820
1	-1.511858	-0.207126	0.357574	0.280384	-0.467820
2	-1.385870	-0.209577	-0.986287	-1.126454	-0.467820
3	-1.259882	-0.170661	-1.207729	-1.385449	-0.467820
4	-1.133893	-0.209577	-0.972866	-1.112127	2.058406

```
XX=(X-X.mean())/X.std()
XX.plot(kind='density', subplots=False, layout=(5,7), sharex=True, sharey=True, figsize=(10,10))
plt.show()
# on voit aussi que avec le diagramme de densité ils sont presque aussi au meme echelle
```




```
XXX=(X-X.min())/(X.max()-X.min())
XXX.describe()
# Avec le min/max scaler tout est compris entre 0 et 1 les max=1 et les min= 0
# On voit que tous les minimum c'est 0 et tous les maximum c'est 1 ça prouve toujours que tous est au meme echelle ce qui est bon pour r
```

	ID_PAYEMENT	PRIX_UNITAIRE_LITRE	TOTAL_TTC	TOTAL_NET	ANNEE
count	27.000000	27.000000	27.000000	27.000000	27.000000
mean	0.500000	0.040204	0.508402	0.474794	0.185185
std	0.305279	0.191834	0.281169	0.295459	0.395847
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.250000	0.001053	0.268698	0.235672	0.000000
50%	0.500000	0.003084	0.482529	0.437642	0.000000
75%	0.750000	0.005397	0.736610	0.704005	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000

PCA

```
n_comp=5
from sklearn.decomposition import PCA


#Centrage et réduction , on a prit notre X de depart
X_scaled = (X-X.mean())/X.std() #scaler.fit_transform(X)
X_scaled
# On crée de nouvelle variable à partir des variables de départ
```

	ID_PAYEMENT	PRIX_UNITAIRE_LITRE	TOTAL_TTC	TOTAL_NET	ANNEE
0	-1.637846	-0.183208	-0.315698	-0.469599	-0.467820
1	-1.511858	-0.207126	0.357574	0.280384	-0.467820
2	-1.385870	-0.209577	-0.986287	-1.126454	-0.467820
3	-1.259882	-0.170661	-1.207729	-1.385449	-0.467820
4	-1.133893	-0.209577	-0.972866	-1.112127	2.058406
5	-1.007905	-0.205460	0.908718	0.885991	2.058406
6	-0.881917	-0.195853	1.748408	1.777594	2.058406
7	-0.755929	-0.163799	-1.423355	-1.606972	-0.467820
8	-0.629941	-0.190462	-0.175228	-0.325774	2.058406
9	-0.503953	-0.190756	-0.044153	-0.125742	-0.467820
10	-0.377964	-0.169876	-0.763056	-0.929177	-0.467820
11	-0.251976	-0.209577	0.706513	0.671080	-0.467820
12	-0.125988	-0.186737	0.623752	0.604954	-0.467820
13	0.000000	-0.205460	-0.448116	-0.555012	-0.467820
14	0.125988	-0.201735	-1.325384	-1.486291	2.058406
15	0.251976	-0.203205	1.527861	1.512538	-0.467820
16	0.377964	-0.178307	-0.529535	-0.689469	-0.467820
17	0.503953	-0.195657	1.257210	1.255747	-0.467820
18	0.629941	-0.192324	-0.538930	-0.651446	-0.467820
19	0.755929	-0.194677	1.159686	1.152149	-0.467820
20	0.881917	-0.184286	0.421546	0.368552	-0.467820
21	1.007905	-0.193501	0.714565	0.656202	-0.467820
22	1.133893	5.003277	-1.808172	0.518439	-0.467820
23	1.259882	-0.204970	-0.092020	-0.192971	-0.467820
24	1.385870	-0.198696	1.209343	1.205050	-0.467820
25	1.511858	-0.178111	-0.941999	-1.112678	-0.467820
26	1.637846	-0.179679	0.937349	0.880480	-0.467820

Next steps: View recommended plots

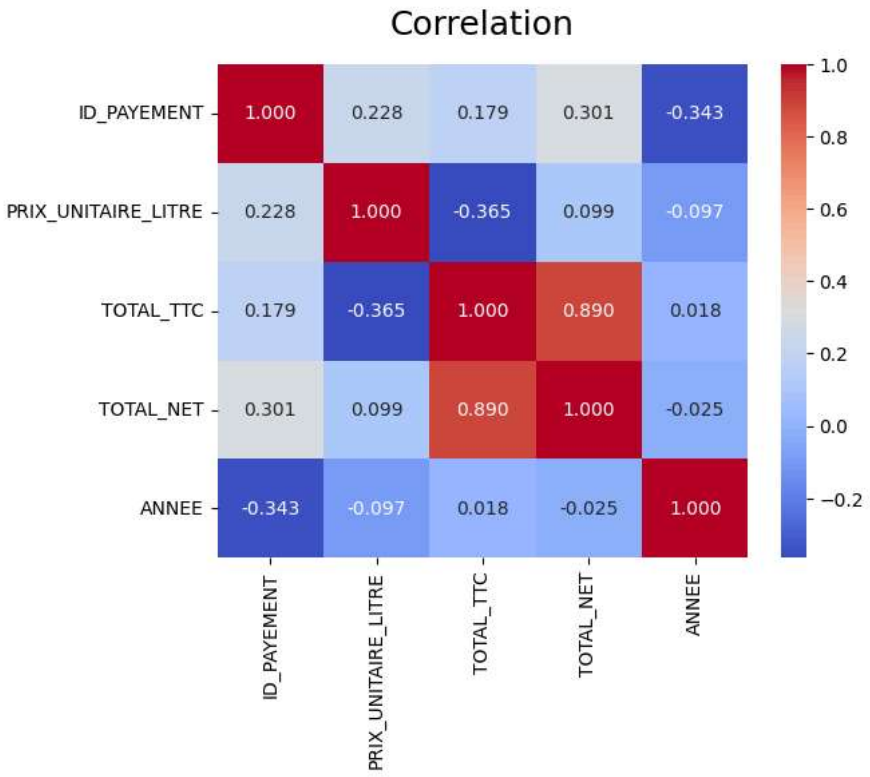
```
#Instanciation de l'ACP #ON passe de x1, x2, x3,x4, x5,... à PCA1, PCA2, PCA3, PCA4, PCA5
pca = PCA(n_comp).fit(X_scaled)
pca    # où sont calculés PCA1 , PCA2, PCA3, PCA4, PCA5
```

```
PCA
PCA(n_components=5)
```

```
X_projected = pca.transform(X_scaled) # c'est mon X après transformation


## on crée également 5 variables à partir des variables qu'on avait avant
pca.components_    # STOCKE LES NOUVELLES VARIABLES dans  PCA1 PCA2, PCA3, PCA4, PCA5 mais on ne sait pas qui PCA1,PCA2... parce que on
```

```
array([[-3.24863370e-01,  8.91151533e-02, -6.60895697e-01,
        -6.62370671e-01,  1.04901154e-01],
       [ 5.54566263e-01,  5.81857726e-01, -2.71306167e-01,
        -6.84435028e-03, -5.29380737e-01],
       [-9.69425631e-02,  6.80363257e-01, -4.42354340e-02,
         2.88566794e-01,  6.65192010e-01],
       [ 7.59948583e-01, -2.99476443e-01, -8.96472884e-02,
        -2.41842969e-01,  5.16011001e-01],
       [-5.84111608e-04,  3.17678729e-01,  6.92543451e-01,
        -6.47656878e-01,  2.00493856e-03]])
```

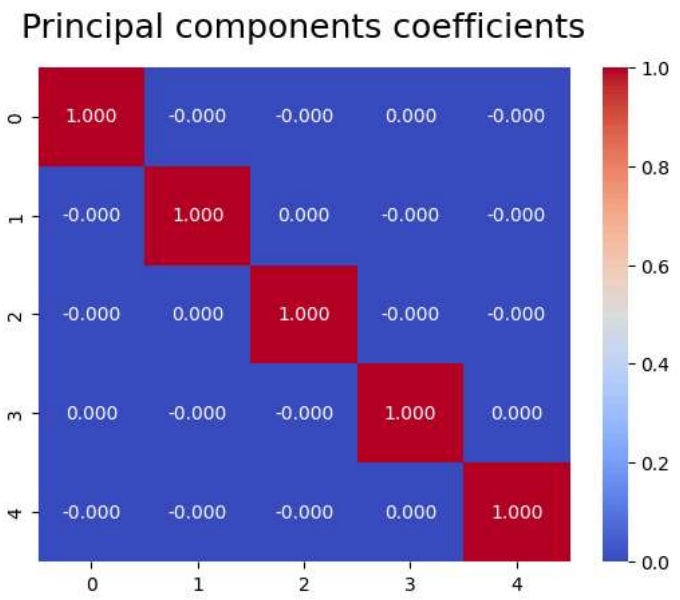
```
import seaborn as sns
heatmap=sns.heatmap(X.corr(),annot=True, fmt=".3f", cmap='coolwarm')
heatmap.set_title('Correlation', fontdict={'fontsize':18}, pad=16);
#INCONVIENTS: il y a des FEATURES(variables explicatives) qui sont corrélées: les modeles detestent les variables corrélées.
```




```
# -- Principal components coefficients (depending on original features)
df_pca = pd.DataFrame(pca.components_,
                      index=['PC'+str(i+1) for i in range(n_comp)],
                      columns=X.columns).T
df_pca.head()
# on crée un objet pandas qui va nous afficher PCA1, PCA2, PCA3, PCA4, PCA5 avec des index ou on a mis nos nouvelles variables sous form
# on crée également 5 variables à partir des variables qu'on avait avant
# On avait nos X de départ maintenant on a nos X darrivé pour le modèle (PCA)
```

	PC1	PC2	PC3	PC4	PC5
ID_PAYEMENT	-0.324863	0.554566	-0.096943	0.759949	-0.000584
PRIX_UNITAIRE_LITRE	0.089115	0.581858	0.680363	-0.299476	0.317679
TOTAL_TTC	-0.660896	-0.271306	-0.044235	-0.089647	0.692543
TOTAL_NET	-0.662371	-0.006844	0.288567	-0.241843	-0.647657
ANNEE	0.104901	-0.529381	0.665192	0.516011	0.002005

```
# Plotting correlation coefficient X_projected
import seaborn as sns
heatmap=sns.heatmap(pd.DataFrame(X_projected).corr(),annot=True, fmt=".3f", cmap='coolwarm')
heatmap.set_title('Principal components coefficients', fontdict={'fontsize':18}, pad=16);
# là on fait la projection de  la correlation des variables aprés tranformation(resultat ici du PCA) pour voir après transformation pour
#Aucune variable dans ma nouvelle base de données est correlée avec les autres: ABSENCE DE CORRELATION ENTRE VARIABLES EXPLICATIVES
```




```
(pca.explained_variance_ratio_*100).cumsum()    # Le ratio nous dit que la 2e variable est intéressantes les derniers info (souvent) rep
#Le ratio nous dit que la première variable est intéressantes à 40,52% , la deuxième variable est intéressante à 70,02%
# la troisième variable est intéressante à intéressante à 88,78%, la quatrième est intéressante à 99,99
```

```
array([ 40.52276401,  70.02287206,  88.78478347,  99.99558998,
       100.        ])
```

```
infopca=pd.DataFrame()
infopca["variables"]=['PC'+str(i+1) for i in range(n_comp)]
infopca["importance"]=pca.explained_variance_ratio_*100
infopca
# On veut voir le PCA le plus important
# On constate que PCA1 est le plus important à 40,52% il comprend la plus de la moitié des infos de notre dataset
# Ensuite on a le PCA2 qui est important à 29.50%
# Ensuite on a le PCA3 qui est important à 18.76%
#Ensuite on a le PCA4 qui est important à 11.21%
#Ensuite on a le PCA5 qui n'est vraiment pas important
```

	variables	importance
0	PC1	40.522764
1	PC2	29.500108
2	PC3	18.761911
3	PC4	11.210807
4	PC5	0.004410

Next steps: View recommended plots

```
infopca.plot(kind="bar")
# le diagramme en bar nous montre bien le dégré d'importance effectivement c'est le PCA1 qui est le plus impotant, ensuite c'est le PCA2
```