2.0.gapminder_analysis

December 9, 2024

1 Analyse du jeu de données “Gapminder” avec pandas

[5]:
```
# Importation des bibliothèques nécessaires
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

2 Chargement

[34]:
```
# Charger le jeu de données gapminder, édité avec des valeurs manquantes et des␣
↪duplications
df = pd.read_csv('data/gapminder_edited.csv')
```

[8]:
```
df.head()
```

[8]:

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_num
0	Congo, Rep.	Africa	1987.0	57.470	2064095.0	4201.194937	COG	178.0
1	Bangladesh	Asia	1982.0	50.009	93074406.0	676.981866	BGD	50.0
2	Algeria	Africa	1977.0	58.014	17152804.0	4910.416756	DZA	12.0
3	Malawi	Africa	1972.0	41.766	4730997.0	584.621971	MWI	454.0
4	Tanzania	Africa	1952.0	41.215	8322925.0	716.650072	TZA	834.0

[9]:
```
df.shape
```

[9]:
```
(1712, 8)
```

[10]:
```
df.sample(5)
```

[10]:
```
            country continent    year  lifeExp          pop     gdpPercap  \
1401       Botswana    Africa  2007.0   50.728    1639131.0  12569.851770
934            Peru  Americas  1977.0   58.447   15990099.0   6281.290855
1166        Myanmar      Asia  1972.0   53.070   28466390.0    357.000000
418            Cuba  Americas  1967.0   68.290    8139332.0   5690.268015
606   United States  Americas  1962.0   70.210  186538000.0  16173.145860

     iso_alpha  iso_num
1401       BWA     72.0
934        PER    604.0
1166       MMR    104.0
418        CUB    192.0
606        USA    840.0
```

[41]:
```python
import json

# Load the raw JSON data
with open('data/parking.json') as f:
    data = json.load(f)

# Extract the 'features' and convert to DataFrame
df_parkings = pd.json_normalize(data['features'])

# Show the first few rows
df_parkings.head()
```

[41]:
```
      type         id geometry_name geometry.type  \
0  Feature  parking.1          geom         Point
1  Feature  parking.2          geom         Point
2  Feature  parking.3          geom         Point
3  Feature  parking.4          geom         Point
4  Feature  parking.5          geom         Point

              geometry.coordinates  properties.objectid properties.id  \
0  [704445.415162, 7059340.511054]                    1       LIL0001
1  [704133.963794, 7059796.512165]                    2       LIL0002
2  [704764.808825, 7059700.637336]                    3       LIL0003
3  [705570.611236, 7059392.022501]                    4       LIL0004
4  [705133.104323, 7059765.537221]                    5       LIL0005

   properties.nom          properties.adresse properties.ville  \
0     République       Place de la République           LILLE
1           Plaza               Rue Nationale           LILLE
2        Tanneurs             Rue du Molinel           LILLE
3    Grand Palais  Boulevard des Citées Unies           LILLE
4  Lille Flandres             Rue de Tournai           LILLE
```

```
  properties.code_insee properties.etat properties.txt_aff  \
0                 59350          OUVERT                145
1                 59350          OUVERT                 50
2                 59350          OUVERT                270
3                 59350          OUVERT               1020
4                 59350          OUVERT                 40

   properties.nbr_total  properties.nbr_libre     properties.dtdate  \
0                   310                   146  2024-12-09T13:54:00Z
1                   323                    50  2024-12-09T13:54:00Z
2                   563                   270  2024-12-09T13:54:00Z
3                  1182                  1024  2024-12-09T13:54:00Z
4                   374                    42  2024-12-09T13:54:00Z

   properties.longitude  properties.latitude
0              3.062724            50.631027
1              3.058334            50.635122
2              3.067235            50.634257
3              3.078601            50.631481
4              3.072433            50.634836
```

3 Nettoyage

[11]:
```
# display duplicate rows
df[df.duplicated()]
```

[11]:
```
            country continent    year  lifeExp         pop     gdpPercap  \
270         Hungary    Europe  1957.0   66.410   9839000.0   6040.180011
845        Colombia  Americas  1967.0   59.963  19764027.0   2678.729839
850             NaN       NaN     NaN      NaN         NaN           NaN
895         Belgium    Europe  1992.0   76.460  10045622.0  25575.570690
942             NaN       NaN     NaN      NaN         NaN           NaN
1183  Saudi Arabia      Asia  1967.0   49.901   5618198.0  16903.048860
1320      Honduras  Americas  1967.0   50.924   2500689.0   2538.269358

     iso_alpha  iso_num
270        HUN    348.0
845        COL    170.0
850        NaN      NaN
895        BEL     56.0
942        NaN      NaN
1183       SAU    682.0
1320       HND    340.0
```

[12]:
```
# Supprimer les doublons
df = df.drop_duplicates()
```

[17]:
```
# Afficher les lignes avec des valeurs manquantes
df[df.isnull().any(axis=1)]
```

[17]:

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_num
128	NaN	Africa	1982.0	58.161	31140029.0	8568.266228	ZAF	710.0
569	Hong Kong, China	Asia	NaN	73.600	4583700.0	11186.141250	HKG	344.0
1348	Japan	Asia	1967.0	71.430	100825279.0	9847.788607	JPN	NaN
1638	Cameroon	Africa	1957.0	40.428	5359923.0	1313.048099	NaN	120.0

[18]:
```
print(f'Nombre de lignes du dataframe avant suppression des valeurs manquantes :
 ↪ {df.shape[0]}')
df.dropna(inplace=True)
print(f'Nombre de lignes du dataframe après suppression des valeurs manquantes :
 ↪ {df.shape[0]}')
```

```
Nombre de lignes du dataframe avant suppression des valeurs manquantes : 1703
Nombre de lignes du dataframe après suppression des valeurs manquantes : 1699
```

[14]:
```
# renvoyer les lignes où une colonne spécifique a une valeur manquante
df_pop_missing = df[df['pop'].isnull()]
df_pop_missing
```

[14]:

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_num
349	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
451	West Bank and Gaza	Asia	1977.0	60.765	NaN	3682.831494	PSE	275.0

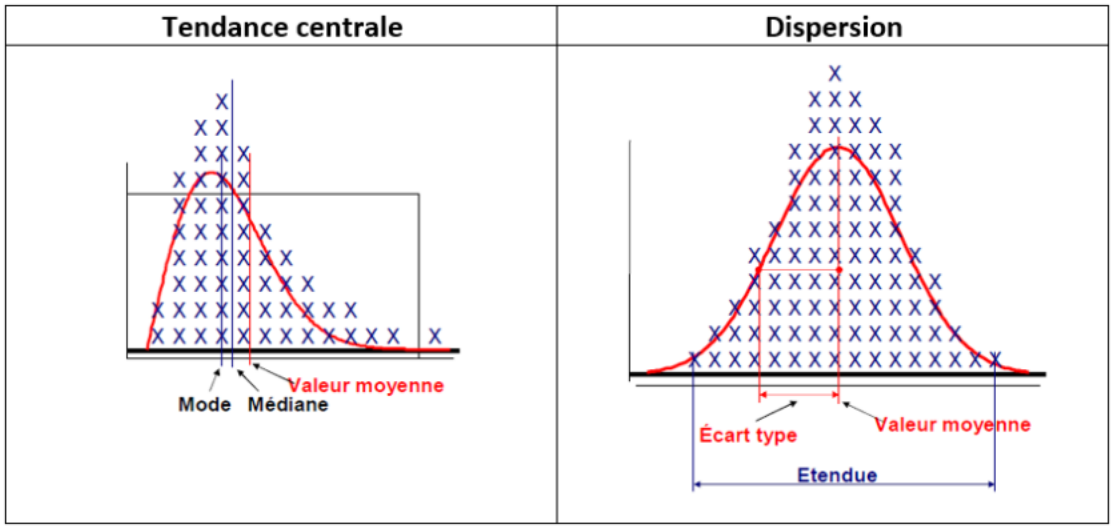
[15]:
```
# Supprimer les lignes avec des valeurs manquantes dans la colonne 'pop'

print(f'Nombre de lignes avant suppression des valeurs manquantes: {df.
 ↪shape[0]}')
df = df.dropna(subset=['pop'], axis=0)
print(f'Nombre de lignes après suppression des valeurs manquantes: {df.
 ↪shape[0]}')
```

```
Nombre de lignes avant suppression des valeurs manquantes: 1705
Nombre de lignes après suppression des valeurs manquantes: 1703
```

[16]:
```
# .all() pour renvoyer les lignes où toutes les colonnes ont des valeurs␣
 ↪manquantes
df_all_missing = df[df.isnull().all(axis=1)]

# .sum() pour renvoyer les lignes avec au moins une valeur manquante
df_at_least_one_missing = df[df.isnull().sum(axis=1) > 0]

# .sum() pour renvoyer les lignes avec exact
df_exactly_two_missing = df[df.isnull().sum(axis=1) == 2]

# .notnull().all() pour renvoyer les lignes où toutes les colonnes ont des␣
 ↪valeurs non manquantes
df_all_non_missing = df[df.notnull().all(axis=1)]

# .notnull().any() pour renvoyer les lignes où au moins une colonne a une␣
 ↪valeur non manquante
df_at_least_one_non_missing = df[df.notnull().any(axis=1)]
```

4 Statistiques descriptives

[35]:
```
# Charger le jeu de données original:
df = pd.read_csv('data/gapminder.csv')
```



[115]:
```
# Calculer des mesures de tendance centrale et de dispersion
print("Statistiques descriptives :")
df.describe()
```

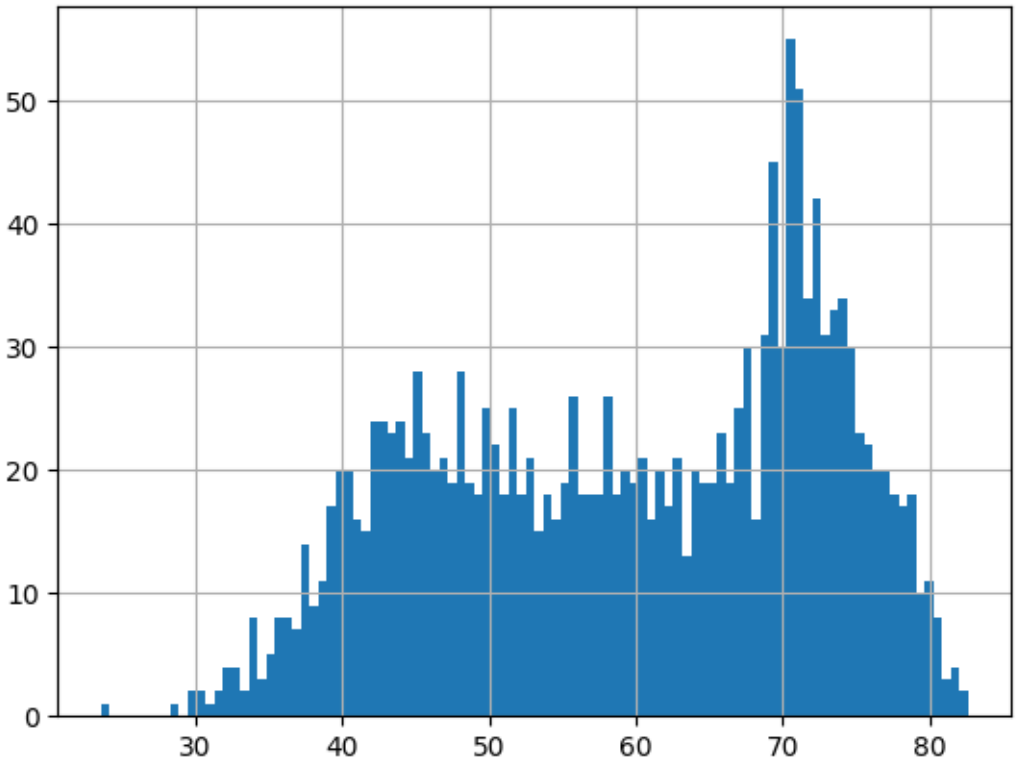
```
Statistiques descriptives :
```

[115]:

	year	lifeExp	pop	gdpPercap	iso_num
count	1702.000000	1703.000000	1.703000e+03	1703.000000	1702.000000
mean	1979.502938	59.473682	2.961785e+07	7217.401359	425.988837
std	17.275264	12.920864	1.061869e+08	9859.977960	248.423325
min	1952.000000	23.599000	6.001100e+04	241.165876	4.000000
25%	1963.250000	48.185000	2.797182e+06	1201.919257	208.000000
50%	1982.000000	60.660000	7.026113e+06	3530.690067	410.000000
75%	1995.750000	70.846000	1.959366e+07	9325.856454	638.000000
max	2007.000000	82.603000	1.318683e+09	113523.132900	894.000000

[116]:
```
df['lifeExp'].hist(bins=100)
```

[116]: <Axes: >



[117]:
```
df['lifeExp'].mean()
```

[117]: np.float64(59.47368155020552)

[118]:
```
df['lifeExp'].std()
```

[118]: np.float64(12.920863652449352)

5 Sélection, filtrage et tri

[119]:
```
# Exemple : Filtrage des données pour l'année 2007
filtered_df = df[df['year'] == 2007]
print("Données pour l'année 2007 :")
filtered_df
```

```
Données pour l'année 2007 :
```

[119]:

	country	continent	year	lifeExp	pop	gdpPercap	\
11	Singapore	Asia	2007.0	79.972	4553009.0	47143.179640	
13	Philippines	Asia	2007.0	71.688	91077287.0	3190.481016	
27	Thailand	Asia	2007.0	70.616	65068149.0	7458.396327	
30	Iran	Asia	2007.0	70.964	69453570.0	11605.714490	
54	Zimbabwe	Africa	2007.0	43.487	12311143.0	469.709298	
…	…	…	…	…	…	…	
1642	Burundi	Africa	2007.0	49.580	8390505.0	430.070692	
1683	Poland	Europe	2007.0	75.563	38518241.0	15389.924680	
1685	Mauritius	Africa	2007.0	72.801	1250882.0	10956.991120	
1705	Lebanon	Asia	2007.0	71.993	3921278.0	10461.058680	
1706	Egypt	Africa	2007.0	71.338	80264543.0	5581.180998	

	iso_alpha	iso_num
11	SGP	702.0
13	PHL	608.0
27	THA	764.0
30	IRN	364.0
54	ZWE	716.0
…	…	…
1642	BDI	108.0
1683	POL	616.0
1685	MUS	480.0
1705	LBN	422.0
1706	EGY	818.0

```
[142 rows x 8 columns]
```

[120]:
```
# Ajout d'une condition: espérance de vie > 70
filtered_df = df[(df['year'] == 2007) & (df['lifeExp'] > 70)]
print("Pays avec une espérance de vie > 70 en 2007 :")
filtered_df
```

```
Pays avec une espérance de vie > 70 en 2007 :
```

[120]:

	country	continent	year	lifeExp	pop	gdpPercap	\
11	Singapore	Asia	2007.0	79.972	4553009.0	47143.179640	
13	Philippines	Asia	2007.0	71.688	91077287.0	3190.481016	

```
27       Thailand      Asia  2007.0   70.616  65068149.0   7458.396327
30           Iran      Asia  2007.0   70.964  69453570.0  11605.714490
56          Chile  Americas  2007.0   78.553  16284741.0  13171.638850
…             …       …     …        …          …             …
1631  Puerto Rico  Americas  2007.0   78.746   3942491.0  19328.709010
1683       Poland    Europe  2007.0   75.563  38518241.0  15389.924680
1685    Mauritius    Africa  2007.0   72.801   1250882.0  10956.991120
1705      Lebanon      Asia  2007.0   71.993   3921278.0  10461.058680
1706        Egypt    Africa  2007.0   71.338  80264543.0   5581.180998

     iso_alpha  iso_num
11         SGP    702.0
13         PHL    608.0
27         THA    764.0
30         IRN    364.0
56         CHL    152.0
…          …      …
1631       PRI    630.0
1683       POL    616.0
1685       MUS    480.0
1705       LBN    422.0
1706       EGY    818.0

[83 rows x 8 columns]
```

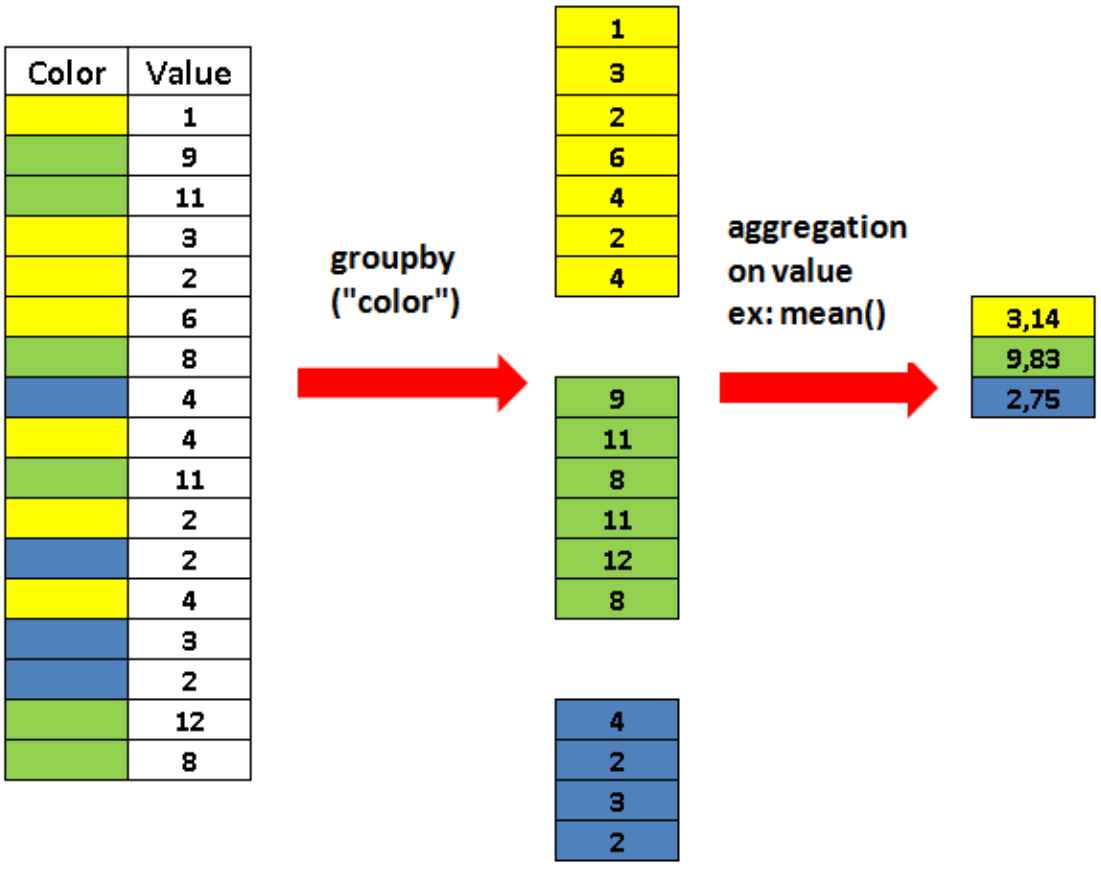
[121]:
```
# Triés par ordre décroissant de l'espérance de vie:
filtered_and_sorted_df = df[(df['lifeExp'] > 70) & (df['year'] == 2007)].
 ↪sort_values(by='lifeExp', ascending=False)
print("Pays avec une espérance de vie > 70 en 2007, triés par ordre décroissant␣
 ↪de l'espérance de vie :")
filtered_and_sorted_df
```

```
Pays avec une espérance de vie > 70 en 2007, triés par ordre décroissant de
l'espérance de vie :
```

[121]:
```
              country continent    year  lifeExp          pop     gdpPercap  \
548             Japan      Asia  2007.0   82.603  127467972.0  31656.068060
216   Hong Kong, China      Asia  2007.0   82.208    6980412.0  39724.978670
1598           Iceland    Europe  2007.0   81.757     301931.0  36180.789190
1409       Switzerland    Europe  2007.0   81.701    7554661.0  37506.419070
919          Australia   Oceania  2007.0   81.235   20434176.0  34435.367440
…                …       …     …        …          …             …
30               Iran      Asia  2007.0   70.964   69453570.0  11605.714490
652         Indonesia      Asia  2007.0   70.650  223547000.0   3540.651564
27           Thailand      Asia  2007.0   70.616   65068149.0   7458.396327
1504         Guatemala  Americas  2007.0   70.259   12572928.0   5186.050003
1057          Honduras  Americas  2007.0   70.198    7483763.0   3548.330846
```

```
     iso_alpha  iso_num
548         JPN    392.0
216         HKG    344.0
1598        ISL    352.0
1409        CHE    756.0
919         AUS     36.0
…           …       …
30          IRN    364.0
652         IDN    360.0
27          THA    764.0
1504        GTM    320.0
1057        HND    340.0

[83 rows x 8 columns]
```

6 Groupement / Aggrégation



[122]:
```
# Espérance de vie moyenne par continent en 2007
grouped_data = df[df['year'] == 2007].groupby('continent')['lifeExp'].mean()
print("Espérance de vie moyenne par continent en 2007 :")
grouped_data
```

```
Espérance de vie moyenne par continent en 2007 :
```

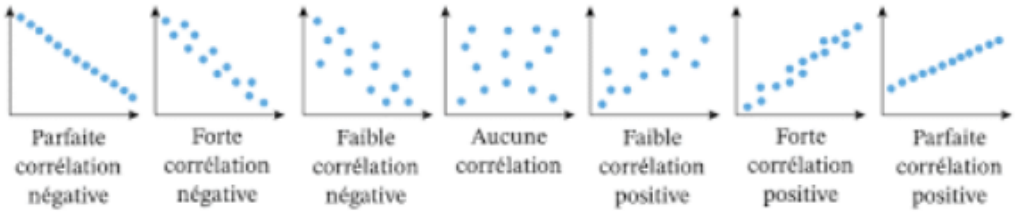
[122]:
```
continent
Africa      54.806038
Americas    73.608120
Asia        70.728485
Europe      77.648600
Oceania     80.719500
Name: lifeExp, dtype: float64
```

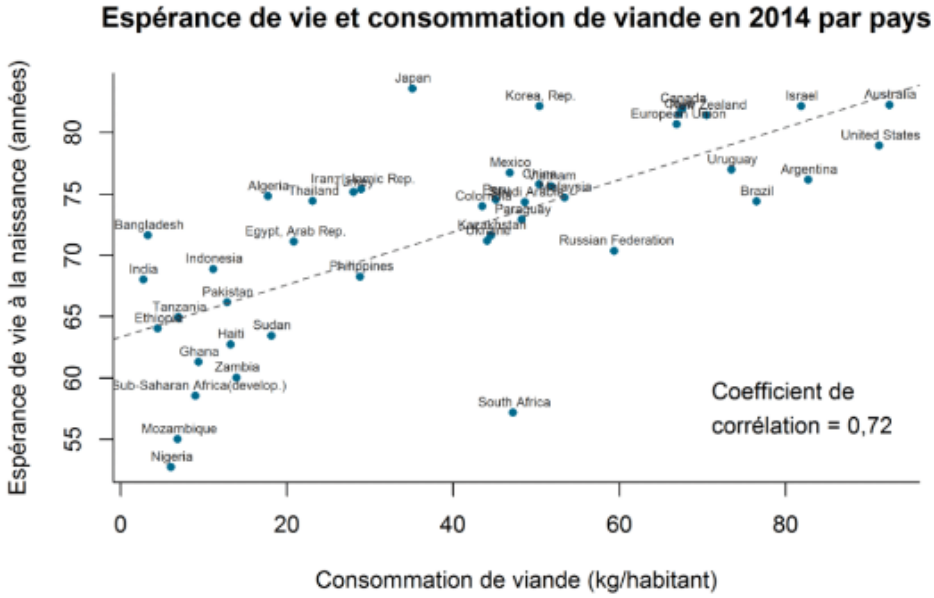
7 Exploration et relations entre variables

Le coefficient de corrélation de Pearson et celui de Spearman sont deux mesures différentes de la corrélation entre deux variables, et ils sont adaptés à des types d'analyses distinctes.

7.1 Coefficient de corrélation de Pearson

- Nature : Mesure linéaire.
- Utilisation : Évalue la force et la direction d'une relation linéaire entre deux variables quantitatives.
- Hypothèse : Suppose que les données sont normalement distribuées et que la relation entre les variables est linéaire.
- Valeur : Entre -1 (corrélation négative parfaite) et 1 (corrélation positive parfaite).
- Sensibilité : Très sensible aux valeurs aberrantes.





[123]:
```
pearson_corr = df['gdpPercap'].corr(df['lifeExp'], method='pearson')
print(f"Coefficient de corrélation de Pearson entre PIB et espérance de vie :␣
↪{pearson_corr:.2f}")
```

```
Coefficient de corrélation de Pearson entre PIB et espérance de vie : 0.58
```

- La relation entre le PIB et l'espérance de vie est modérément positive. Cela signifie qu'une augmentation du PIB est associée à une augmentation de l'espérance de vie, mais la relation n'est pas strictement linéaire.
- Limitation : Cette valeur relativement basse (comparée à Spearman) indique que des variations ou des anomalies dans les données (par exemple, des pays avec un PIB élevé mais une faible espérance de vie) peuvent affaiblir la corrélation linéaire.

7.2 Coefficient de corrélation de Spearman

- Nature : Basé sur les rangs.
- Utilisation : Mesure la force et la direction d'une relation monotone (croissante ou décroissante), qu'elle soit linéaire ou non.
- Hypothèse : Aucune hypothèse stricte sur la distribution ou la linéarité des données.
- Valeur : Entre -1 et 1, tout comme Pearson.

- Sensibilité : Moins sensible aux valeurs aberrantes car il utilise les rangs des données plutôt que leurs valeurs absolues.

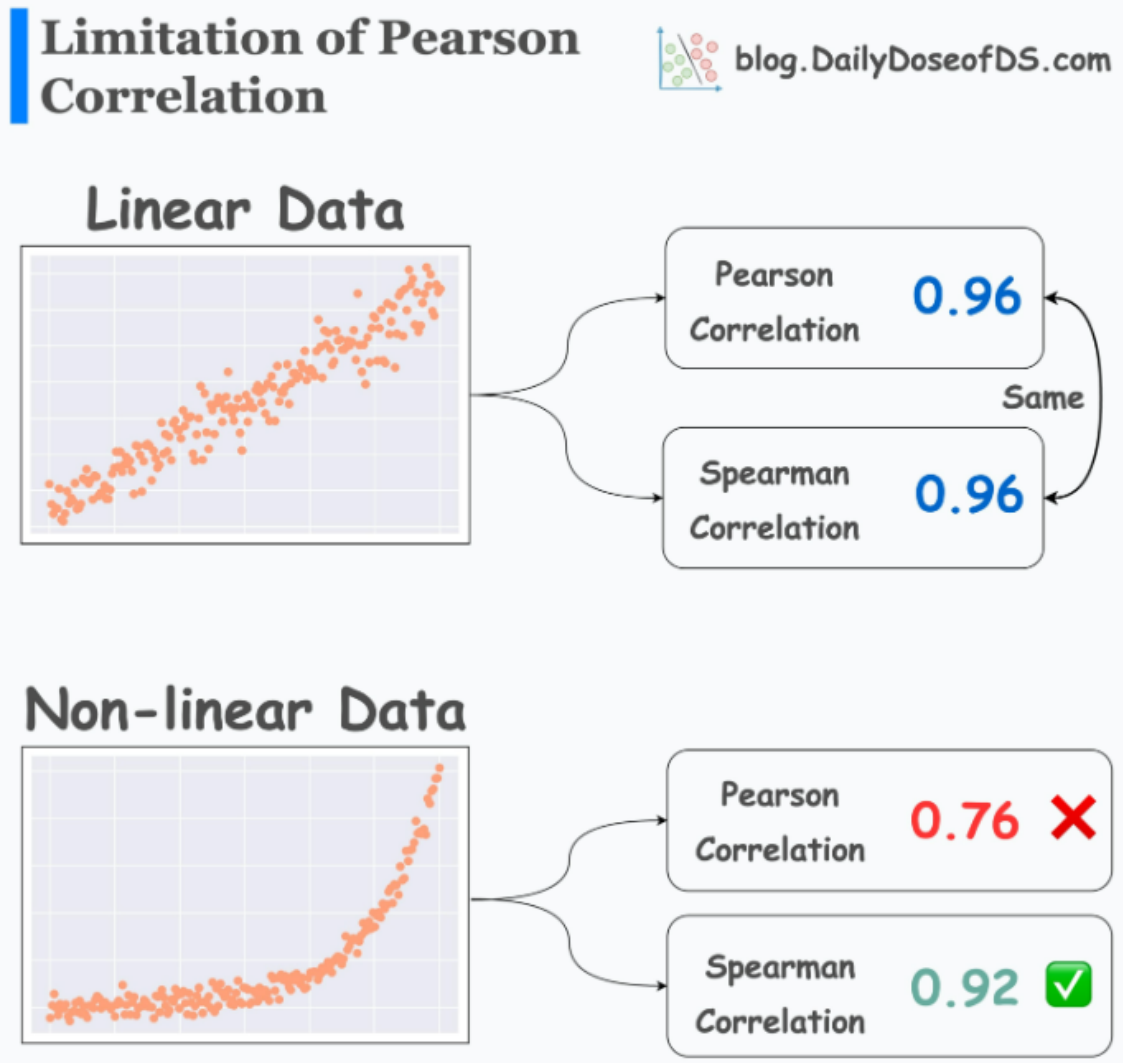
[124]:
```
# b) Corrélation de Spearman
spearman_corr = df['gdpPercap'].corr(df['lifeExp'], method='spearman')
print(f"Coefficient de corrélation de Spearman entre PIB et espérance de vie :␣
↪{spearman_corr:.2f}")
```

```
Coefficient de corrélation de Spearman entre PIB et espérance de vie : 0.83
```

- La relation entre le PIB et l'espérance de vie est fortement monotone. Cela signifie que lorsque le PIB augmente, l'espérance de vie tend à augmenter également, même si la relation n'est pas nécessairement linéaire.
- Avantage de Spearman : En se basant sur les rangs (et non sur les valeurs exactes), ce coefficient est moins influencé par les valeurs aberrantes ou des cas particuliers.

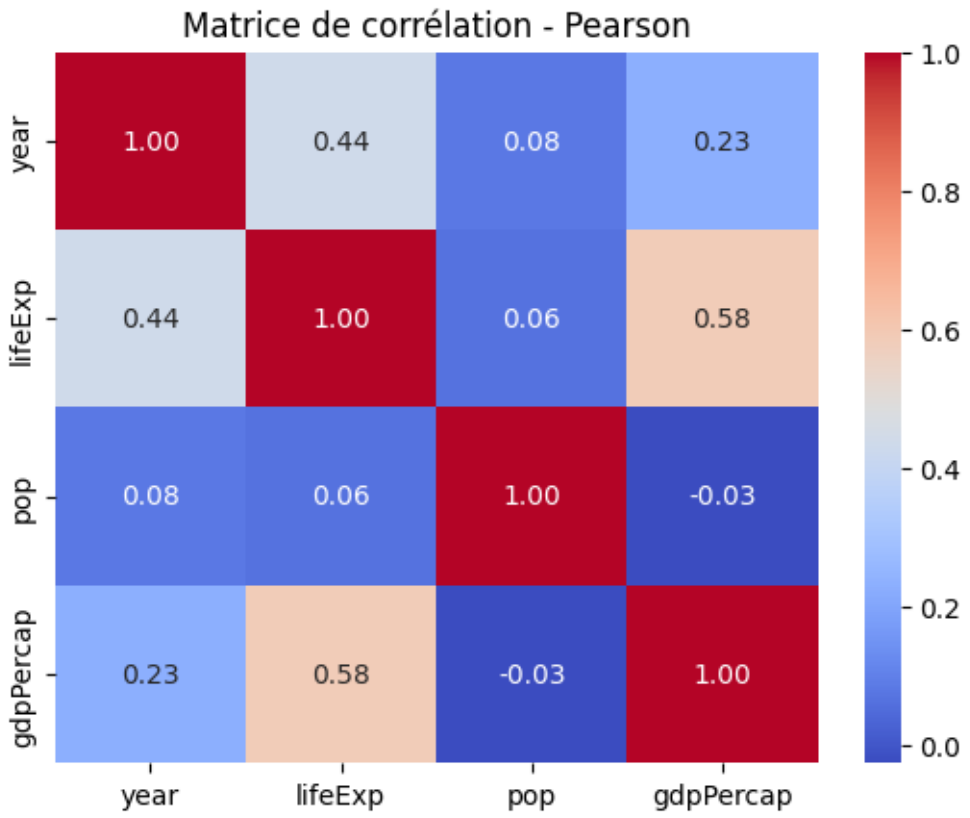
7.3 Interprétation globale

- **Non-linéarité :** Une différence importante entre les deux coefficients (comme ici) suggère que la relation entre le PIB et l'espérance de vie est probablement non-linéaire. Par exemple, une augmentation du PIB pourrait avoir un impact important sur l'espérance de vie dans les pays à faible PIB, mais un effet moins marqué dans les pays riches.
- **Impact des valeurs aberrantes :** Le coefficient de Pearson est probablement affecté par des valeurs extrêmes ou des outliers dans les données (comme des pays riches avec une faible espérance de vie ou vice-versa), tandis que Spearman capture mieux la tendance générale.

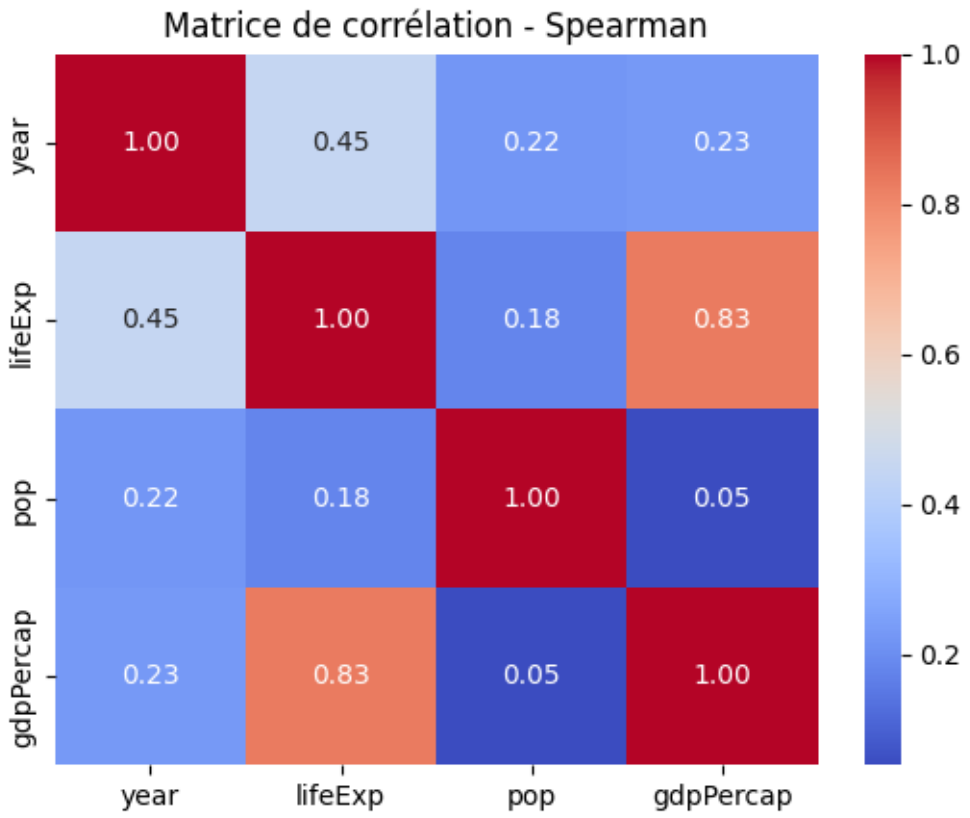


7.4 Matrice de corrélation

[125]:
```
corr_matrix = df[['year', 'lifeExp', 'pop', 'gdpPercap']].corr(method='pearson')
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Matrice de corrélation - Pearson")
plt.show()
```



[128]:
```
corr_matrix = df[['year', 'lifeExp', 'pop', 'gdpPercap']].
 ↪corr(method='spearman')
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Matrice de corrélation - Spearman")
plt.show()
```



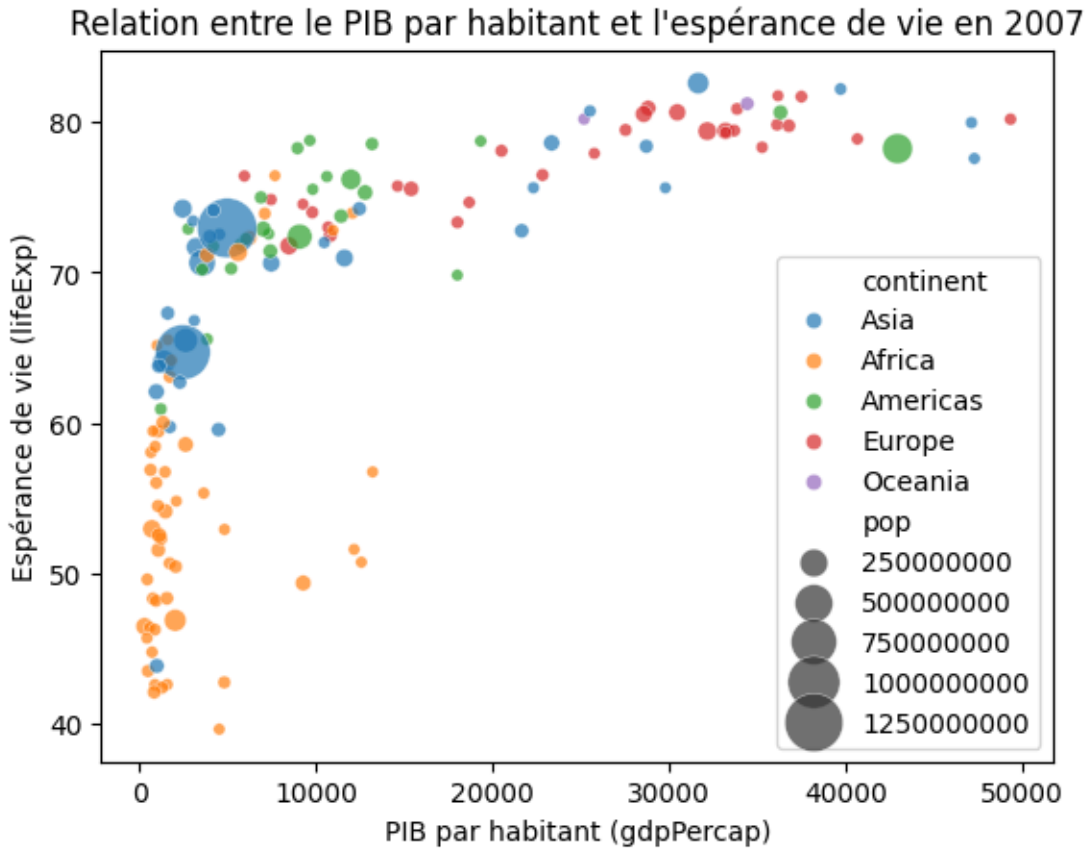
7.5 Scatter matrix

[129]:
```
import plotly.express as px

# filtered_df = df[df['continent'] == 'Europe']
# filtered_df = df[df['country'] == 'Honduras']
# filtered_df = filtered_df[['year', 'lifeExp', 'pop', 'gdpPercap']]

filtered_df = df[['year', 'lifeExp', 'pop', 'gdpPercap']]
fig = px.scatter_matrix(filtered_df)
# fig = px.scatter_matrix(df[df['year'] == 2007][['continent', 'year',␣
 ↪'lifeExp', 'pop', 'gdpPercap']], color='continent')
fig.show()
```

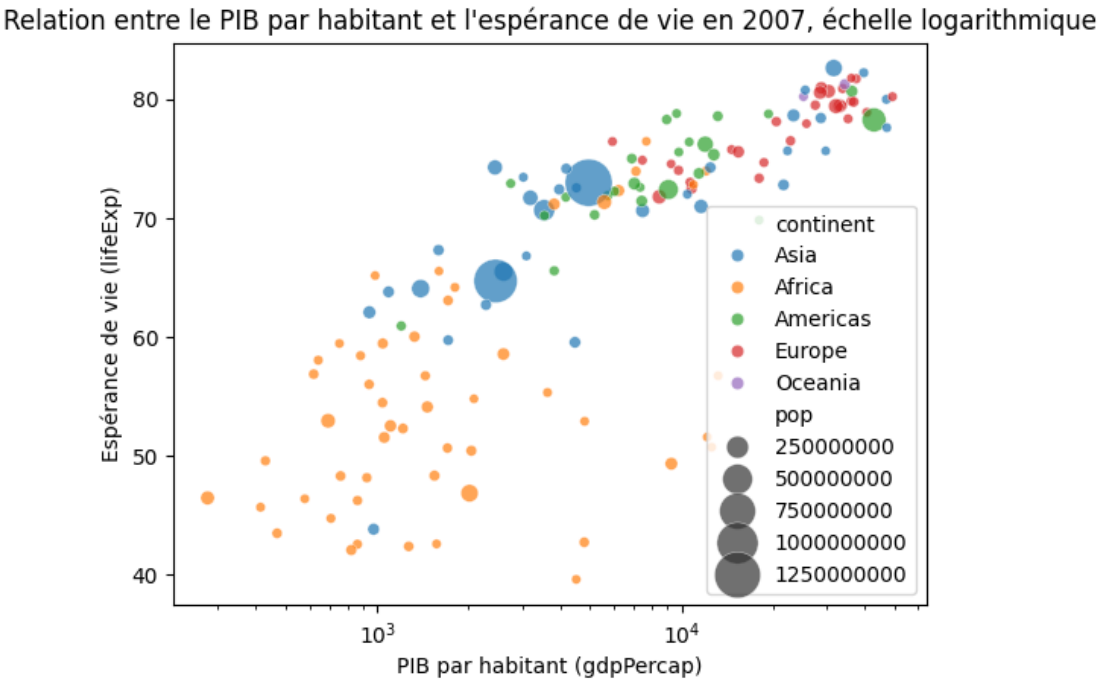
7.6 Analyse bi-variée

[130]:
```
# Visualisation de la relation entre le PIB par habitant et l'espérance de vie␣
 ↪en 2007
sns.scatterplot(data=df[df['year'] == 2007], x='gdpPercap', y='lifeExp',␣
 ↪hue='continent', alpha=0.7, size='pop', sizes=(20, 500))
```

```
plt.title("Relation entre le PIB par habitant et l'espérance de vie en 2007")
plt.xlabel("PIB par habitant (gdpPercap)")
plt.ylabel("Espérance de vie (lifeExp)")
plt.show()
```



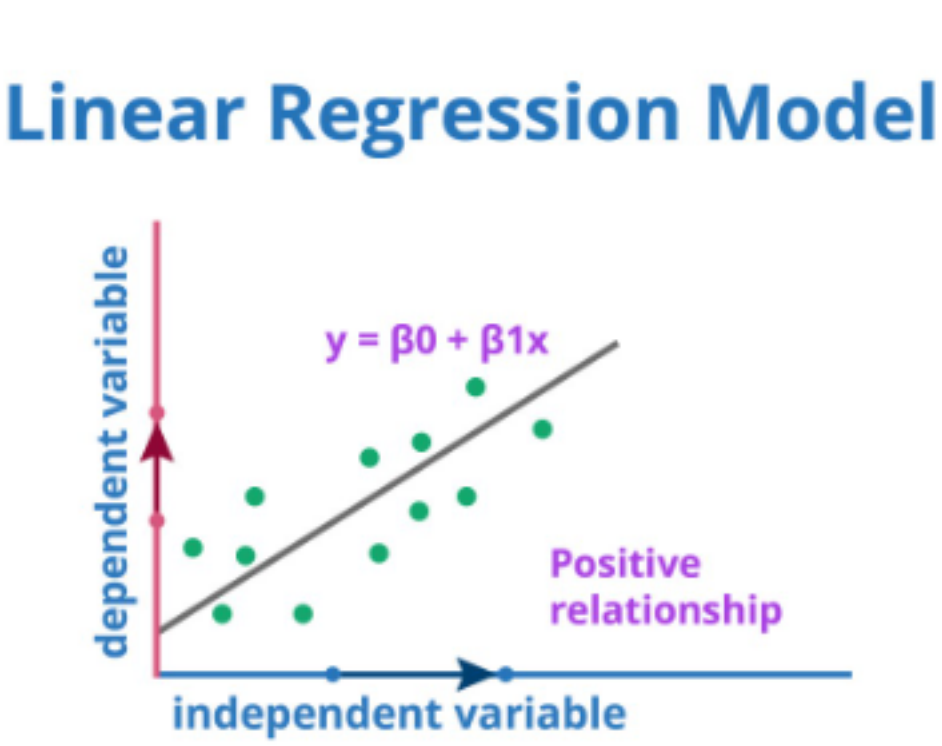
[131]:
```
# Application d'une échelle logarithmique
sns.scatterplot(data=df[df['year'] == 2007], x='gdpPercap', y='lifeExp',␣
 ↪hue='continent', alpha=0.7, size='pop', sizes=(20, 500))
plt.xscale('log')
plt.title("Relation entre le PIB par habitant et l'espérance de vie en 2007,␣
 ↪échelle logarithmique")
plt.xlabel("PIB par habitant (gdpPercap)")
plt.ylabel("Espérance de vie (lifeExp)")
plt.show()
```

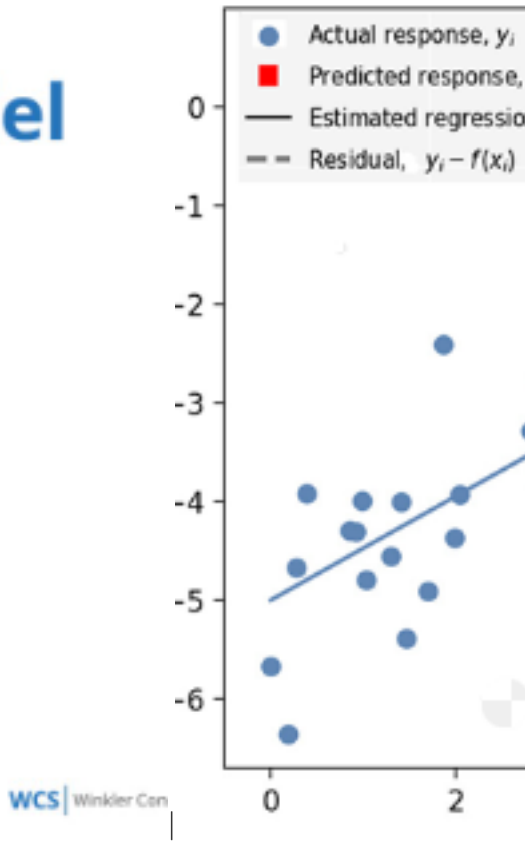


[132]:
```
import plotly.express as px

fig = px.scatter(df[df['year'] == 2007], y="lifeExp", x="gdpPercap",␣
 ↪color="continent", log_x=True, size="pop", size_max=60,
                 hover_name="country", height=600, width=1000,␣
 ↪template="simple_white",
                 color_discrete_sequence=px.colors.qualitative.G10,
                 title="Health vs Wealth 2007",
                 labels=dict(
                     continent="Continent", pop="Population",
                     gdpPercap="GDP per Capita (US$, price-adjusted)",
                     lifeExp="Life Expectancy (years)"))

fig.update_layout(font_family="Rockwell",
                  legend=dict(orientation="h", title="", y=1.1, x=1,␣
 ↪xanchor="right", yanchor="bottom"))
fig.update_xaxes(tickprefix="$", range=[2,5], dtick=1)
fig.update_yaxes(range=[30,90])
fig.add_hline((df["lifeExp"]*df["pop"]).sum()/df["pop"].sum(), line_width=1,␣
 ↪line_dash="dot")
fig.add_vline((df["gdpPercap"]*df["pop"]).sum()/df["pop"].sum(), line_width=1,␣
 ↪line_dash="dot")
fig.show()
```

[38]:
```
import plotly.express as px

# Création du graphique animé
fig = px.scatter(
    df,
    x="gdpPercap",
    y="lifeExp",
    animation_frame="year",
    animation_group="country",
    size="pop",
    color="continent",
    hover_name="country",
    log_x=True,  # Échelle logarithmique pour le PIB par habitant
    size_max=60,
    title="Relation entre le PIB par habitant et l'espérance de vie (Animation␣
 ↪par année)"
)

# Mise en page
fig.update_layout(
    xaxis_title="PIB par habitant (gdpPercap, échelle logarithmique)",
    yaxis_title="Espérance de vie (lifeExp)",
    legend_title="Continent",
    # yaxis_range=[20, 90],
    # xaxis_range=[-10e3, 60e3]
)

# Afficher le graphique
fig.show()
```

8 Régression linéaire simple



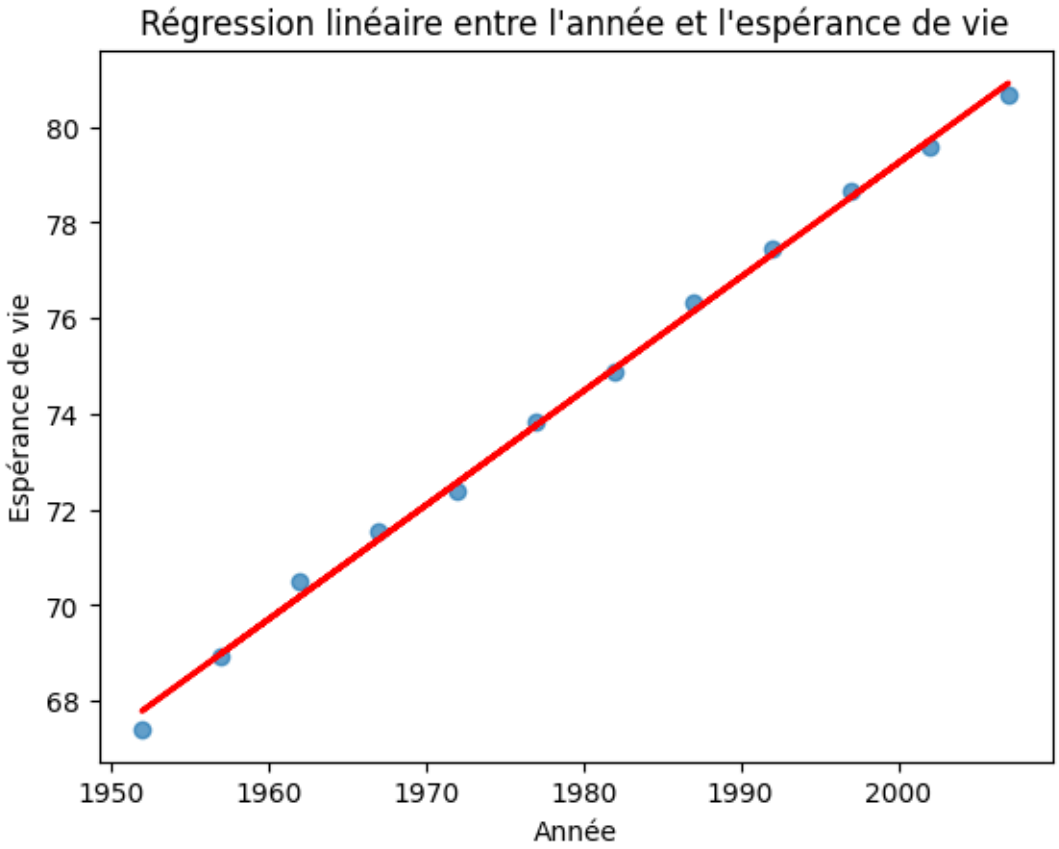


[134]:
```
# Modèle de régression entre l'année et l'espérance de vie

# Filtrer les données pour la France
france_df = df[df['country'] == 'France']

# Sélectionner les features et la target
X = france_df[['year']] # Feature
y = france_df['lifeExp'] # Target

# Créer un modèle de régression linéaire
model = LinearRegression()
model.fit(X, y)

# Afficher les paramètres du modèle
slope, intercept = model.coef_[0], model.intercept_
print(f"Modèle de régression : y = {slope:.2f} * x + {intercept:.2f}")
```

```
Modèle de régression : y = 0.24 * x + -397.76
```

[135]:
```
# Visualisation de la droite de régression
y_pred = model.predict(X)
```

```
plt.scatter(X, y, alpha=0.7)
plt.plot(X, y_pred, color='red', linewidth=2)
plt.title("Régression linéaire entre l'année et l'espérance de vie")
plt.xlabel("Année")
plt.ylabel("Espérance de vie")
plt.show()
```

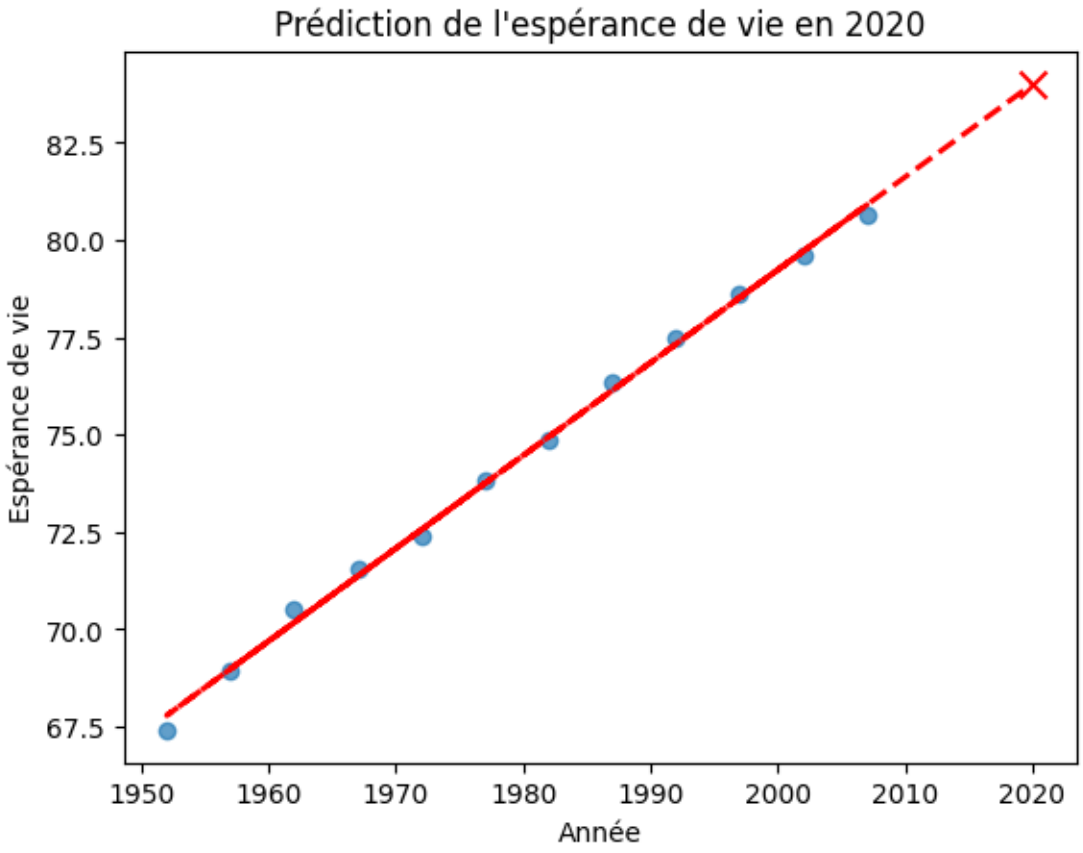


[136]:
```
# Prédiction de l'espérance de vie en 2020
X_pred_2020 = pd.DataFrame([[2020]], columns=['year'])
y_pred_2020 = model.predict(X_pred_2020)
print(f"Prédiction de l'espérance de vie en 2020 : {y_pred_2020[0]:.2f}")
```

```
Prédiction de l'espérance de vie en 2020 : 84.01
```

[137]:
```
# Visualisation de la prédiction de l'espérance de vie en 2020
plt.scatter(X, y, alpha=0.7)

# Ajouter la prédiction
plt.scatter(X_pred_2020, y_pred_2020, color='red', marker='x', s=100)
```

```
# Ajouter la droite de régression
plt.plot(np.append(X, X_pred_2020), np.append(y_pred, y_pred_2020),␣
 ↪color='red', linestyle='--', linewidth=2)

plt.title("Prédiction de l'espérance de vie en 2020")
plt.xlabel("Année")
plt.ylabel("Espérance de vie")
plt.show()
```



9 Appliquer ces compétences sur des cas pratiques (données de santé, données financières, données de vente, etc.)