mooc_mertes

June 24, 2023

[4959]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import forestplot as fp
import missingno as msno
from statsmodels.graphics.mosaicplot import mosaic
from statsmodels.formula.api import ols
from statsmodels.formula.api import logit
from statsmodels.formula.api import glm
from statsmodels.api import qqplot
from scipy.stats import ttest_ind as t_student
from scipy.stats import mannwhitneyu
from scipy.stats import chi2_contingency
from scipy.stats import pearsonr
from scipy.stats import spearmanr
from scipy.stats import kstest
from scipy.stats import norm
from matplotlib.patches import Rectangle
```

[4960]:
```
# Chargement des données
effec1 = pd.read_csv("../csv/effec1.quest.compil.csv", encoding="ISO-8859-1")
effec2 = pd.read_csv("../csv/effec2.quest.compil.csv", encoding="ISO-8859-1")
effec3 = pd.read_csv("../csv/effec3.quest.compil.csv", encoding="ISO-8859-1")
#
usage1 = pd.read_csv("../csv/usages.effec1.csv", encoding="ISO-8859-1")
usage2 = pd.read_csv("../csv/usages.effec2.csv", encoding="ISO-8859-1")
usage3 = pd.read_csv("../csv/usages.effec3.csv", encoding="ISO-8859-1")
```

[4961]:
```
effec3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4233 entries, 0 to 4232
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Student_ID        3883 non-null   float64
```

```
 1   Certif.bin         3883 non-null   float64
 2   Section            1587 non-null   object
 3   Gender             1577 non-null   object
 4   birth.year         1543 non-null   float64
 5   Country            1575 non-null   object
 6   Diploma            1574 non-null   object
 7   EMLYON.et          1567 non-null   object
 8   Formation          1562 non-null   object
 9   CSP                1572 non-null   object
 10  How.heard          1571 non-null   object
 11  Exp.crea           1570 non-null   object
 12  Curiosity.MOOC     1569 non-null   object
 13  Certif.self.sat    1561 non-null   float64
 14  Rencontres         1551 non-null   object
 15  Certif.work        1562 non-null   float64
 16  Incitation         1564 non-null   float64
 17  Temps.Dispo        1561 non-null   object
 18  Exp.MOOC           1556 non-null   object
 19  Completion.proba   1559 non-null   float64
 20  EMLyon             1567 non-null   object
 21  Country_HDI        1562 non-null   object
 22  Country_HDI.fin    1562 non-null   object
 23  age                1543 non-null   float64
 24  CSP.fin            1572 non-null   object
 25  Temps.dispo.fin    1561 non-null   object
dtypes: float64(8), object(18)
memory usage: 860.0+ KB
```

[4962]:
```
# Fusion des tables effec[n] et usage[n] dans des bases intermédiaires selon
 ↪les identifiants des étudiants
bases_inter = []
for tabs in [(effec1, usage1), (effec2, usage2), (effec3, usage3)]:
    bases_inter.append(tabs[0].merge(tabs[1], on="Student_ID"))
```

[4963]:
```
# Concaténation des tables intermédiaires afin de créer une seule base commune
base = pd.concat(bases_inter, join="outer",
                 axis=0, ignore_index=False,
                 keys=[1, 2, 3])
```

[4964]:
```
# Indication de non passage de la certification pour les itération 1 et 2 où
 ↪seul l'examen existe.
base["Certif.bin"].fillna(0, inplace=True)
```

[4965]:
```
base = base.astype({"Exam.bin": bool, "Certif.bin": bool, "Student_ID": int})
```

[4966]:
```
base = base.copy()
# Map des modalités M et H de la variable Country_HDI
```

```
base["New_HDI"] = np.select([base["Country_HDI"] == "M",
                             base["Country_HDI"] == "H"],
                            ["I", "I"], default=base["Country_HDI"])
```

[4967]:
```
base = base.reset_index(level=0).rename({"level_0": "Itération"}, axis=1)
```

[4968]:
```
# Nombre de catégories (videos, quiz, exam...) par apprenant par place de␣
 ↪séquence dans la semaine pour les différentes itérations
groups = base.groupby(["Student_ID", "Itération"]).sum()

# Nombre de passages à l'examen
exam = groups[["Exam.bin"]]

# Nombre de Certifications
certif = groups[["Certif.bin"]]

# Nombre de devoirs par apprenant
devoir = groups[["Assignment.bin"]]

# Nombre de videos par apprenant par semaine pour chaque itération
week_video = [pd.DataFrame(groups.loc[:, 'S1.L1':'S1.L6'].sum(axis=1),␣
 ↪columns=["video.S1"]),
              pd.DataFrame(groups.loc[:, 'S2.L1':'S2.L6'].sum(axis=1),␣
 ↪columns=["video.S2"]),
              pd.DataFrame(groups.loc[:, 'S3.L1.1':'S3.L5'].sum(axis=1),␣
 ↪columns=["video.S3"]),
              pd.DataFrame(groups.loc[:, 'S4.L1.1':'S4.L5'].sum(axis=1),␣
 ↪columns=["video.S4"]),
              pd.DataFrame(groups.loc[:, 'S5.L1.1':'S5.L5'].sum(axis=1),␣
 ↪columns=["video.S5"])]

# Nombre de quiz par apprenant par semaine pour chaque itération
week_quiz = [groups[["Quizz.1.bin"]],
             groups[["Quizz.2.bin"]],
             groups[["Quizz.3.bin"]],
             groups[["Quizz.4.bin"]],
             groups[["Quizz.5.bin"]]]
```

[4969]:
```
# Concaténation des videos et des questionnaires de l'ensemble des semaines
compil_week_video = pd.concat(week_video, axis=1)
compil_week_quiz = pd.concat(week_quiz, axis=1)
```

[4970]:
```
compil_week_video.head()
```

[4970]:

		video.S1	video.S2	video.S3	video.S4	video.S5
Student_ID	Itération					
15	2	2	0	0	0	0

```
            3                 1         0         0         0         0
28          1                 0         0         0         0         0
34          3                 0         0         0         0         0
36          1                 0         0         0         0         0
```

[4971]:
```
compil_week_quiz.head()
```

[4971]:
```
                      Quizz.1.bin  Quizz.2.bin  Quizz.3.bin  Quizz.4.bin  \
Student_ID Itération
15         2                    0            0            0            0
           3                    0            0            0            0
28         1                    0            0            0            0
34         3                    0            0            0            0
36         1                    0            0            0            0

                      Quizz.5.bin
Student_ID Itération
15         2                    0
           3                    0
28         1                    0
34         3                    0
36         1                    0
```

[4972]:
```
# Nombre de videos et questionnaires par itération par apprenant pour␣
↪l'ensemble du MOOC.
total_video = pd.DataFrame(compil_week_video.sum(axis=1), columns=["video"])

# Nombre de questionnaires par itération par apprenant pour l'ensemble du MOOC.
total_quiz = pd.DataFrame(compil_week_quiz.sum(axis=1), columns=["quiz"])
```

[4973]:
```
total_video.head()
```

[4973]:
```
                      video
Student_ID Itération
15         2              2
           3              1
28         1              0
34         3              0
36         1              0
```

[4974]:
```
total_quiz.head()
```

[4974]:
```
                      quiz
Student_ID Itération
15         2             0
           3             0
28         1             0
```

```
34          3               0
36          1               0
```

[4975]:
```
# Création de la table regroupant toutes la variables pour mesurer l'engagement␣
 ↪de chaque apprenant pour chaque itération
total_student = pd.concat([total_video, total_quiz, devoir, exam, certif],␣
 ↪axis=1)
```

[4976]:
```
total_student.head()
```

[4976]:

Student_ID	Itération	video	quiz	Assignment.bin	Exam.bin	Certif.bin
15	2	2	0	0	0	0
	3	1	0	0	0	0
28	1	0	0	0	0	0
34	3	0	0	0	0	0
36	1	0	0	0	0	0

[4977]:
```
# Sélection des itérations
#total_student.xs(1, level=1)
```

[4978]:
```
# selection des types d'apprenant
def student_type(col):
    video, quiz, devoir, exam, certif = col
    if (exam >= 1 or certif >= 1):
        return "Completer"
    elif quiz > 0 and devoir > 0:
        return "Disengaging"
    elif video > 6:
        return "Auditing"
    else:
        return "Bystander"
```

[4979]:
```
total_student["Type"] = total_student.apply(student_type, axis=1)
```

[4980]:
```
total_student.head(10)
```

[4980]:

Student_ID	Itération	video	quiz	Assignment.bin	Exam.bin	Certif.bin	\
15	2	2	0	0	0	0	
	3	1	0	0	0	0	
28	1	0	0	0	0	0	
34	3	0	0	0	0	0	
36	1	0	0	0	0	0	
45	1	25	5	0	0	0	
83	1	22	5	1	0	0	
84	1	8	2	0	0	0	

```
87          1               1     0                0          0           0
88          3               1     0                0          0           0
```

```
                                Type
Student_ID Itération
15         2             Bystander
           3             Bystander
28         1             Bystander
34         3             Bystander
36         1             Bystander
45         1              Auditing
83         1           Disengaging
84         1              Auditing
87         1             Bystander
88         3             Bystander
```

[4981]:
```
student = total_student.reset_index()[["Student_ID", "Type", "Itération"]]
```

[4982]:
```
student["Type"].value_counts()
```

[4982]:
```
Bystander      8691
Disengaging    2643
Auditing       2120
Completer      1728
Name: Type, dtype: int64
```

[4983]:
```
# Calcul du nombre d'apprenants par type et par itération
df_type = student.groupby(["Itération", "Type"])[["Type"]].count().
 ↪rename({'Type': 'total'}, axis=1)
```

[4984]:
```
df_type
```

[4984]:
```
                       total
Itération Type
1         Auditing      1207
          Bystander     4285
          Completer       20
          Disengaging   2453
2         Auditing       538
          Bystander     2168
          Completer      876
          Disengaging    120
3         Auditing       375
          Bystander     2238
          Completer      832
          Disengaging     70
```

[4985]:
```
df_type.reset_index("Type", inplace=True)
```

[4986]:
```
# Nombre total d'apprenants par itération
df_iter = df_type.groupby("Itération").sum()
```

[4987]:
```
df_iter
```

[4987]:

Itération	total
1	7965
2	3702
3	3515

[4988]:
```
total_iter = df_type.merge(df_iter, on="Itération", suffixes=["_type", "_iter"])
```

[4989]:
```
total_iter
```

[4989]:

Itération	Type	total_type	total_iter
1	Auditing	1207	7965
1	Bystander	4285	7965
1	Completer	20	7965
1	Disengaging	2453	7965
2	Auditing	538	3702
2	Bystander	2168	3702
2	Completer	876	3702
2	Disengaging	120	3702
3	Auditing	375	3515
3	Bystander	2238	3515
3	Completer	832	3515
3	Disengaging	70	3515

[4990]:
```
# Proportion d'apprenants par types d'apprenants et par itération
total_iter["proportion/iter"] = round(total_iter["total_type"] /␣
 ↪total_iter["total_iter"] * 100, 1)
```

[4991]:
```
total_iter
```

[4991]:

Itération	Type	total_type	total_iter	proportion/iter
1	Auditing	1207	7965	15.2
1	Bystander	4285	7965	53.8
1	Completer	20	7965	0.3
1	Disengaging	2453	7965	30.8
2	Auditing	538	3702	14.5
2	Bystander	2168	3702	58.6
2	Completer	876	3702	23.7
2	Disengaging	120	3702	3.2

```
3            Auditing         375        3515            10.7
3           Bystander        2238        3515            63.7
3           Completer         832        3515            23.7
3         Disengaging          70        3515             2.0
```

[4992]:
```
base["Genre"] = base["Gender"].map({"un homme": "Homme", "une femme": "Femme"})
```

[4993]:
```
# Tableau de contingence (croisement des 2 variables catégorielles)
tab_obs = pd.crosstab(index=base["Genre"], columns=base["New_HDI"])
```

[4994]:
```
tab_obs.rename(columns={"un homme": "H", "une femme": "F"}, inplace=True)
```

[4995]:
```
tab_obs
```

[4995]:
```
New_HDI    B    I    TH
Genre
Femme    147  233  2545
Homme    883  432  4711
```

[4996]:
```
# Test d'indépendance (chi2)
chi2, p_value, degres_liberte, tableau_attendu = chi2_contingency(tab_obs)
```

[4997]:
```
tableau_attendu
```

[4997]:
```
array([[ 336.58250475,  217.308122  , 2371.10937325],
       [ 693.41749525,  447.691878  , 4884.89062675]])
```

[4998]:
```
chi2
```

[4998]:
```
179.2420322171424
```

[4999]:
```
p_value
```

[4999]:
```
1.196980957821505e-39
```

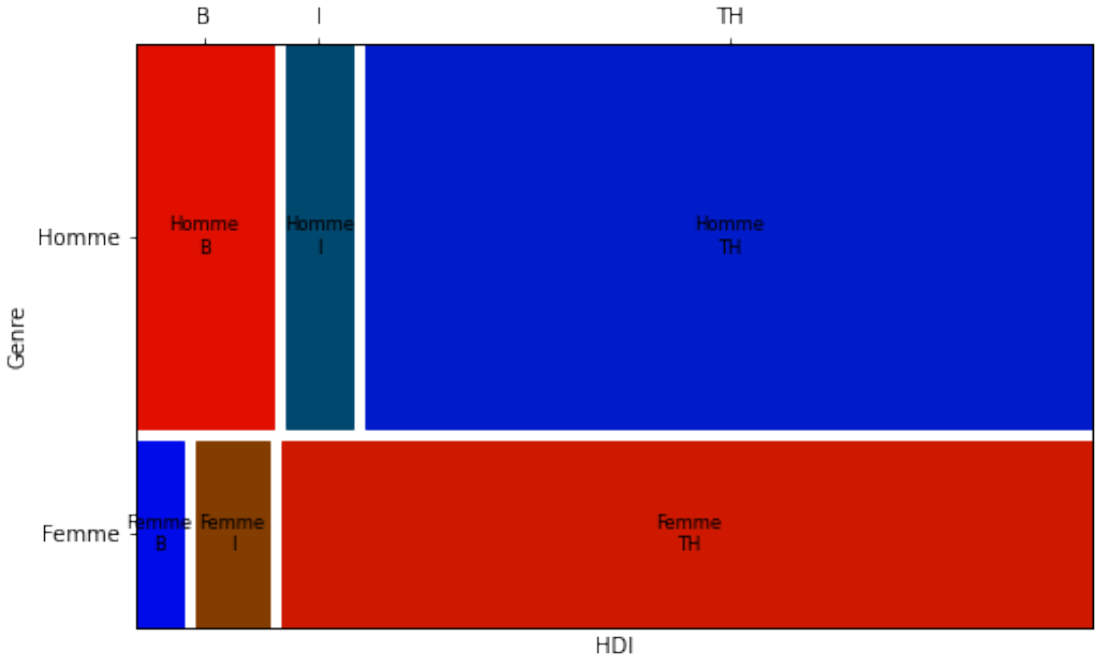
La p value est $< 5\%$ indiquerait que l'index HDI serait significativement lié au genre puisqu'il y a moins de 5 % de chance que les 2 variables soient indépendantes.

[5000]:
```
#residus = (tab_HDI_gender - tableau_attendu) / tableau_attendu
residus = tab_obs - tableau_attendu
```

[5001]:
```
residus
```

[5001]:
```
New_HDI           B          I          TH
Genre
Femme   -189.582505  15.691878  173.890627
Homme    189.582505 -15.691878 -173.890627
```

[5002]:
```
# Mosaic du tableau de contingence
#props = lambda key: {'color': 'red' if residus(key[0]) > 0 else 'blue'},
fig, ax = plt.subplots(figsize=(8,5))
mosaic(tab_obs.stack(), statistic=True, gap=0.02, horizontal=False, ax=ax)
ax.set_xlabel("HDI")
ax.set_ylabel("Genre")
plt.savefig("../graph/mosaic_contingence.png")
plt.show()
```



[5003]:
```
# valeurs attendues sous l'hypothèse nulle (H0).
x = tableau_attendu.flatten()
```

[5004]:
```
x
```

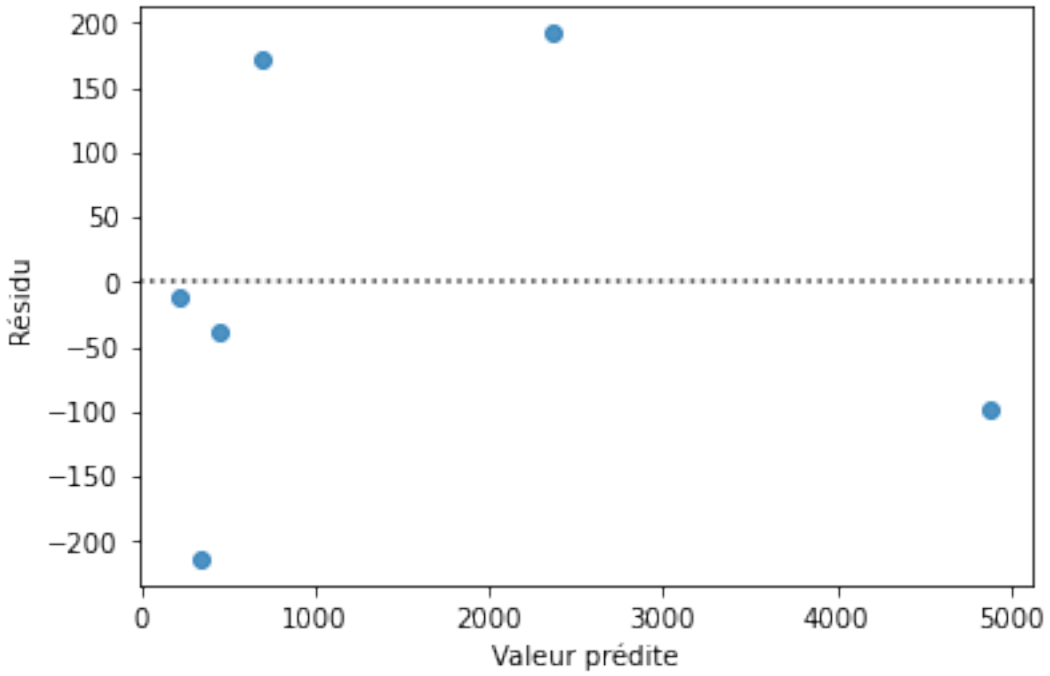
[5004]:
```
array([ 336.58250475,  217.308122  , 2371.10937325,  693.41749525,
        447.691878  , 4884.89062675])
```

[5005]:
```
# valeurs observées
y = tab_obs.stack().values
```

[5006]:
```
y
```

[5006]:
```
array([ 147,  233, 2545,  883,  432, 4711])
```

[5007]:
```
# Résidus du modèle de prédiction
fig, ax = plt.subplots()
sns.residplot(x=x, y=y, ax=ax)
#ax.set_title("Résidus du modèle observé")
ax.set_ylabel("Résidu")
ax.set_xlabel("Valeur prédite")
plt.savefig("../graph/residus_chi2.png")
plt.show()
```



fig, ax = plt.subplots() sns.heatmap(residus, annot=True, cmap='coolwarm', cbar=True, ax=ax) ax.set_ylabel("Genre") ax.set_ylabel("HDI") plt.show()

formule du V de Cramer :

V = √(² / (n * (min(r, c) - 1)))

Dans cette formule :

V représente le coefficient de Cramer. ² est la statistique du chi carré. n est la taille de l'échantillon. r est le nombre de niveaux ou de catégories de la première variable. c est le nombre de niveaux ou de catégories de la deuxième variable.

[5008]:
```
# Fonction de calcul du V de Cramer
# data = tab ndarray
def V_Cramer(data):
    # somme de chaque colonne
    n = np.sum(data)
```

```
    # taille du tableau de contingence des variables catégorielles (taille de
↪chaque échantillon pour chaque variable)
    row, col = tab_obs.shape
    # Formule du V de Cramer
    V = np.sqrt(chi2 / (n * (min([row, col]) - 1)))
    return V
```

[5009]:
```
V_Cramer(np.array(tab_obs))
```

[5009]:
```
0.14150902903141144
```

[5010]:
```
V_Cramer(tableau_attendu)
```

[5010]:
```
0.14150902903141144
```

La valeur V de Cramer étant faible il y a aurait une faible dépendance entre l'index HDI et le genre. Il y aurait donc statistiquement une association entre ses deux variables catégorielles, indiquée par la p-value du chi2, mais la valeur du V de Cramer indiquerait que cette dépendance serait faible.

[5011]:
```
# tableau du genre par étudiant
genre_etu = base[["Student_ID", "Genre"]].drop_duplicates(subset="Student_ID").
↪dropna()
```

[5012]:
```
genre_etu.head()
```

[5012]:

	Student_ID	Genre
1	19178	Femme
2	1086	Femme
3	1948	Femme
4	16209	Femme
5	6685	Homme

[5013]:
```
# Nombre total de videos par étudiant
total_video_etu = total_video.groupby("Student_ID").sum()
```

[5014]:
```
total_video_etu.head()
```

[5014]:

Student_ID	video
15	3
28	0
34	0
36	0
45	25

[5015]:
```
# tableau du genre et du nombre total de videos visionnées par étudiant
etu_genre_video = total_video_etu.merge(genre_etu, on='Student_ID')
```

[5016]:
```
etu_genre_video.head()
```

[5016]:

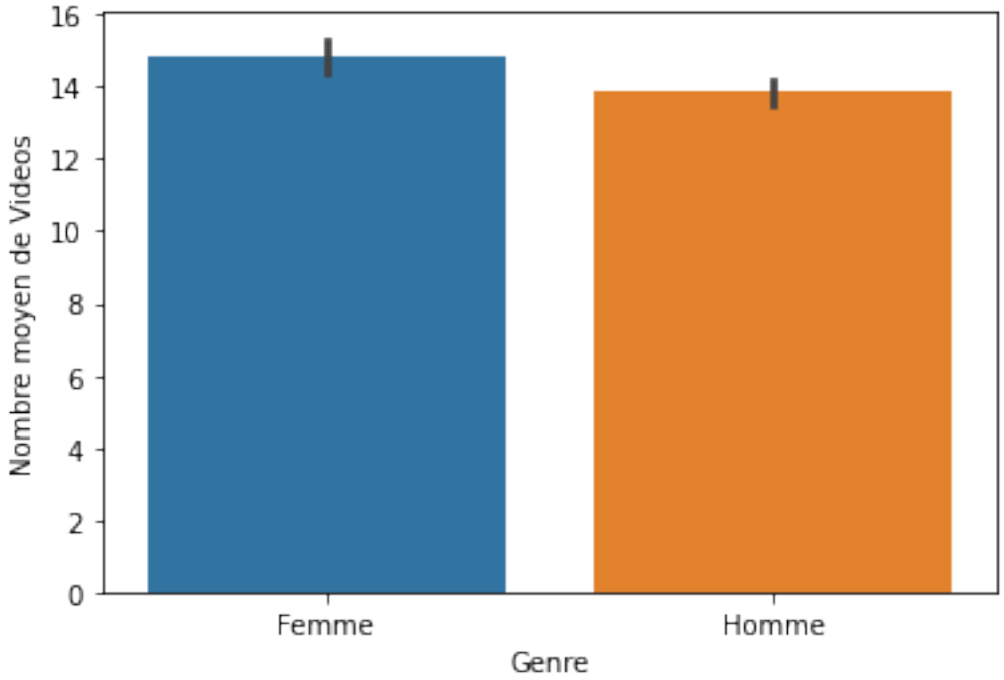
	Student_ID	video	Genre
0	45	25	Femme
1	83	22	Homme
2	84	8	Homme
3	87	1	Homme
4	94	2	Homme

[5017]:
```
# moyenne du nombre de videos visionnées par genre
total_video_etu.merge(genre_etu, on="Student_ID").groupby("Genre")[["video"]].
↪mean()
```

[5017]:

Genre	video
Femme	14.855376
Homme	13.867276

[5018]:
```
# Figure de la distribution du nombre de video par rapport au genre
fig, ax = plt.subplots()
sns.barplot(data=etu_genre_video, x="Genre", y="video", ax=ax)
ax.set_xlabel("Genre")
ax.set_ylabel("Nombre moyen de Videos")
plt.savefig("../graph/mean_video.png")
plt.show()
```



H0 (Hypothèse nulle) : il n'y a pas de différence sur le nombre de video visionnées entre les hommes et les femmes

[5019]:
```
tab_stat = etu_genre_video.pivot_table(columns="Genre", index="Student_ID",␣
↪values="video").fillna(0)
```

[5020]:
```
tab_stat.head()
```

[5020]:
```
Genre       Femme  Homme
Student_ID
45           25.0    0.0
83            0.0   22.0
84            0.0    8.0
87            0.0    1.0
94            0.0    2.0
```

[5021]:
```
# Test de Student
statistique, p_value = t_student(tab_stat["Homme"], tab_stat["Femme"])
```

[5022]:
```
statistique
```

[5022]:
```
26.98492395204523
```

[5023]:
```
p_value
```

[5023]:
```
3.4121378553908065e-157
```

Il y a moins de 5% de chance qu'il n'y ait pas de différence du nombre de visionnages entre les femmes et les hommes. Il y aurait significativement un lien entre le nombre de visionnages et le genre.

[5024]:
```
# Sélection par genre
df_hom = etu_genre_video[etu_genre_video["Genre"] == "Homme"]
df_fem = etu_genre_video[etu_genre_video["Genre"] == "Femme"]
```

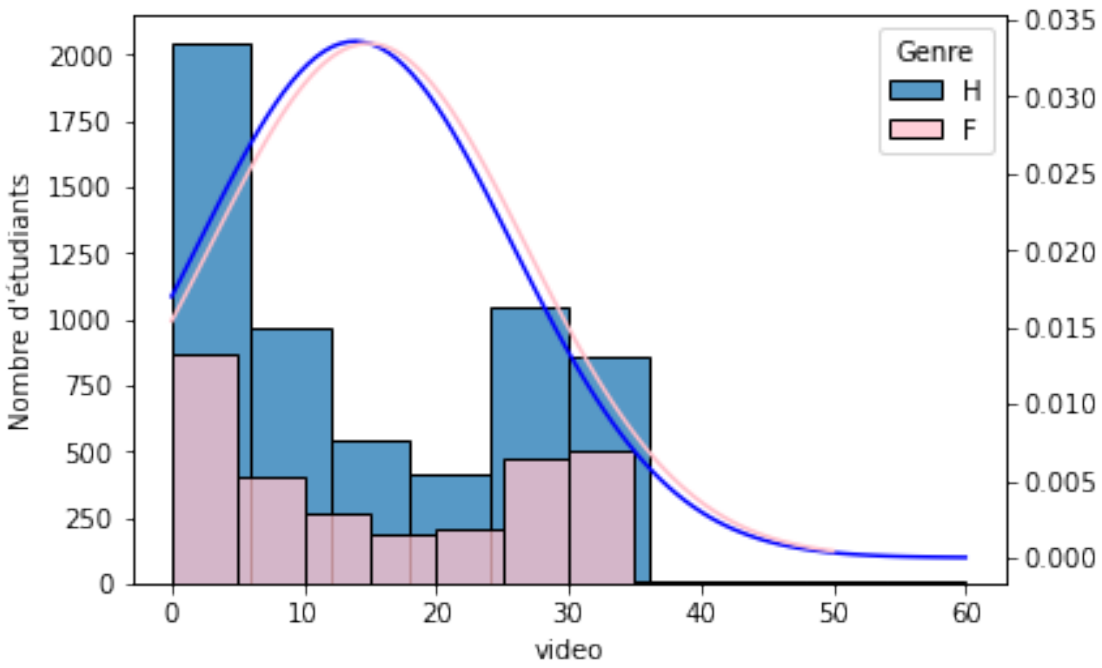
[5025]:
```
df_hom
```

[5025]:
```
      Student_ID  video  Genre
1             83     22  Homme
2             84      8  Homme
3             87      1  Homme
4             94      2  Homme
5             98     23  Homme
…            …     …     …
8807       68205     30  Homme
8808       68220     30  Homme
8809       68223      0  Homme
8811       68265     14  Homme
8813       68282      1  Homme
```

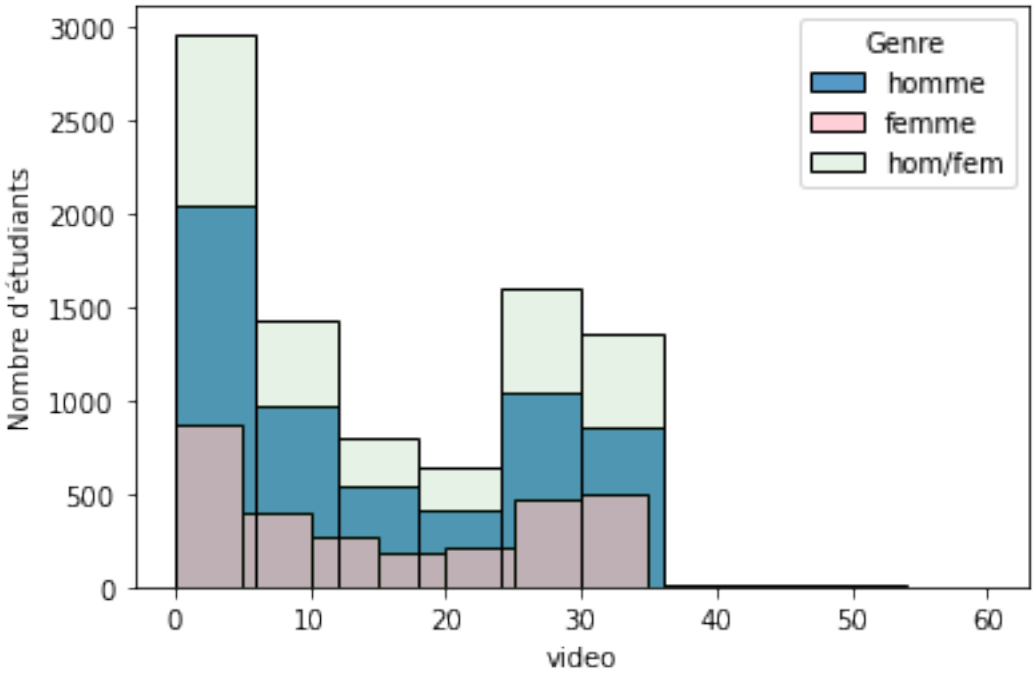
```
[5907 rows x 3 columns]
```

[5026]:
```
# Calcul des paramètres de la distribution normale pour chaque groupe
mu_hommes, std_hommes = np.mean(df_hom["video"]), np.std(df_hom["video"])
mu_femmes, std_femmes = np.mean(df_fem["video"]), np.std(df_fem["video"])

# Création des valeurs x pour tracer la courbe théorique
x_hommes = np.linspace(min(df_hom["video"]), max(df_hom["video"]), 100)
x_femmes = np.linspace(min(df_fem["video"]), max(df_fem["video"]), 100)

# Calcul des valeurs y correspondantes en utilisant la PDF de la distribution␣
 ↪normale
y_hommes = norm.pdf(x_hommes, mu_hommes, std_hommes)
y_femmes = norm.pdf(x_femmes, mu_femmes, std_femmes)
```

[5027]:
```
figure, ax1 = plt.subplots()
sns.histplot(data=df_hom, x="video", bins=10, color="tab:blue", ax=ax1,␣
 ↪label="H")
sns.histplot(data=df_fem, x="video", bins=10, color="pink", ax=ax1, label="F")
ax1.set_xlabel("video")
ax1.set_ylabel("Nombre d'étudiants")
ax1.legend(title="Genre")
ax2 = ax1.twinx()
ax2.plot(x_hommes, y_hommes, color='blue')
ax2.plot(x_femmes, y_femmes, color='pink')
plt.savefig("../graph/distribution_video.png")
plt.show()
```



[5028]:
```
# visualisation de la normalité de la distribution des données (histogramme de distribution)
data = pd.concat([df_hom["video"], df_fem["video"]])
figure, ax = plt.subplots()
sns.histplot(data=df_hom, x="video", bins=10, color="tab:blue", ax=ax, label="homme")
sns.histplot(data=df_fem, x="video", bins=10, color="pink", ax=ax, label="femme")
sns.histplot(data=data, bins=10, color="green", ax=ax, label="hom/fem", alpha=0.1)
ax.set_xlabel("video")
ax.set_ylabel("Nombre d'étudiants")
ax.legend(title="Genre")
plt.savefig("../graph/distribution_video2.png")
plt.show()
```



La distribution des données ne suit pas une loi normale. Ce qui ne permet pas de faire un test t-Student puisque la condition première est que les données doivent être normalement distribuées.

[5029]:
```
# Models de regression du nombre de video selon le genre
mdl_video_vs_genre = ols("video ~ Genre", data=etu_genre_video).fit()
```

[5030]:
```
mdl_video_vs_genre.summary()
```

[5030]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  video   R-squared:                       0.002
Model:                            OLS   Adj. R-squared:                  0.001
Method:                 Least Squares   F-statistic:                     13.45
Date:                Fri, 23 Jun 2023   Prob (F-statistic):           0.000247
Time:                        20:00:50   Log-Likelihood:                -34348.
No. Observations:                8818   AIC:                         6.870e+04
Df Residuals:                    8816   BIC:                         6.871e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
==
                 coef    std err          t      P>|t|      [0.025
0.975]
```

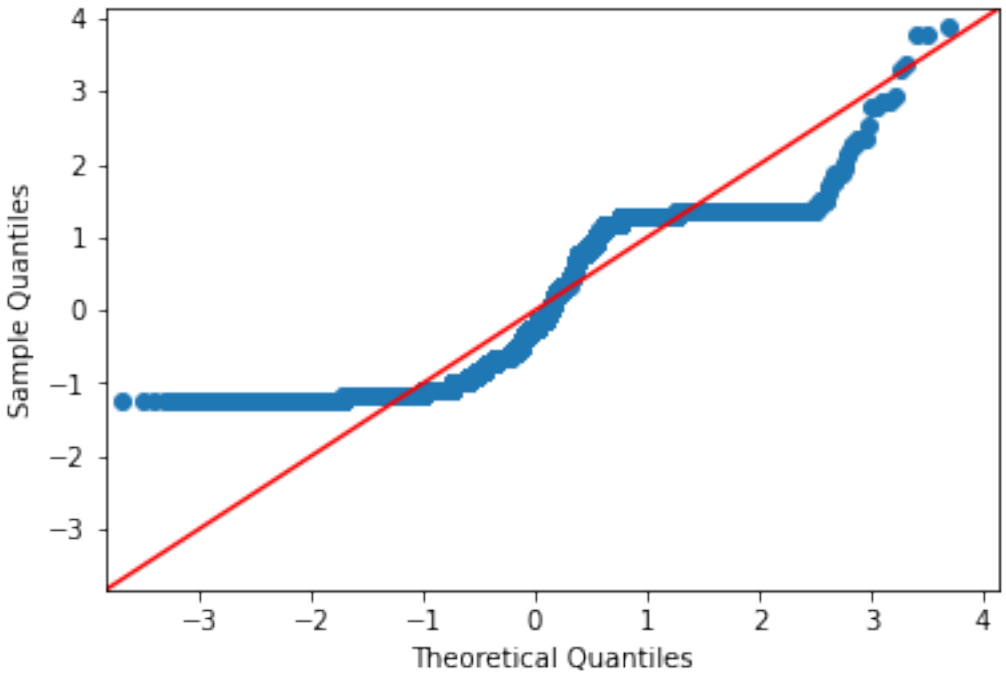
```
------------------------------------------------------------------------------
--
Intercept          14.8554      0.221     67.365      0.000      14.423
15.288
Genre[T.Homme]     -0.9881      0.269     -3.667      0.000      -1.516
-0.460
==============================================================================
Omnibus:                    59307.238   Durbin-Watson:                   1.962
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              878.417
Skew:                           0.240   Prob(JB):                    1.80e-191
Kurtosis:                       1.530   Cond. No.                         3.24
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

Le coefficient R^2 étant proche de 0, le model de régression est peu fidèle à l'observation des données.

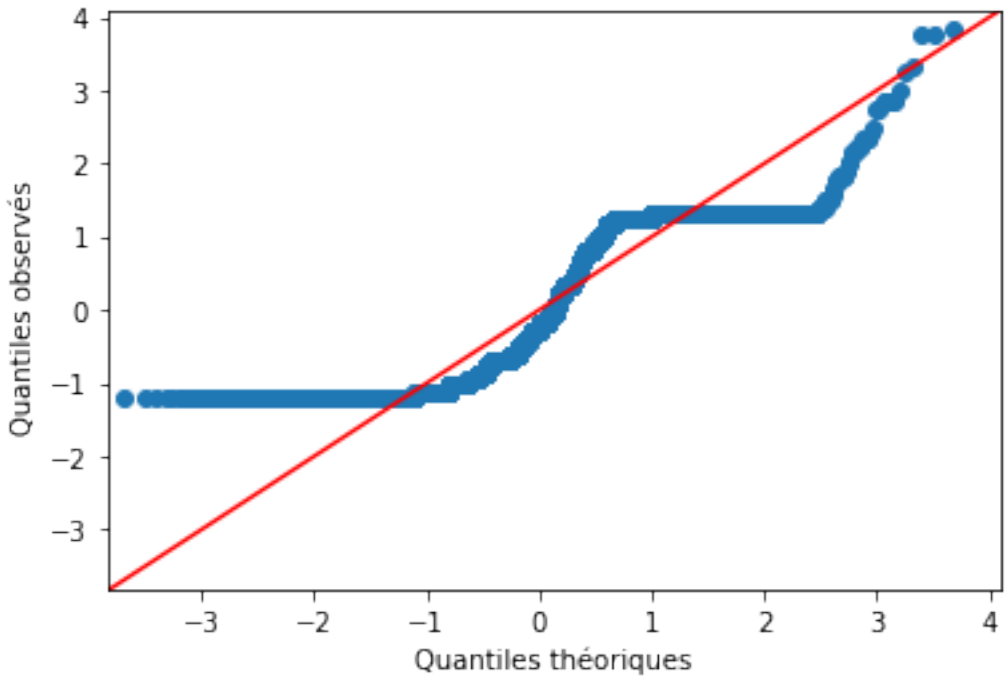
[5031]:
```
mdl_video_vs_genre.params
```

[5031]:
```
Intercept          14.855376
Genre[T.Homme]     -0.988100
dtype: float64
```

[5032]:
```
# Test de normalité de la distribution des données (Q-Q plot)
qqplot(data=mdl_video_vs_genre.resid, fit=True, line="45")
plt.show()
```



[5033]:
```
qqplot(data=data, fit=True, line="45")
plt.xlabel("Quantiles théoriques")
plt.ylabel("Quantiles observés ")
plt.savefig("../graph/distribution_video3.png")
plt.show()
```



[5034]:
```
# Test de  Kolmogrov-Smirnov
stat, p = kstest(data, 'norm')
```

[5035]:
```
stat, p
```

[5035]:
```
(0.7626887430801719, 0.0)
```

[5036]:
```
# Test non paramétrique de Mann-Whitney U
mannwhitneyu(df_hom["video"], df_fem["video"])
```

[5036]:
```
MannwhitneyuResult(statistic=8200580.5, pvalue=0.0003907509304995919)
```

[5037]:
```
# Nombre de quiz par étudiant
total_quiz_etu = total_quiz.groupby("Student_ID").sum()
```

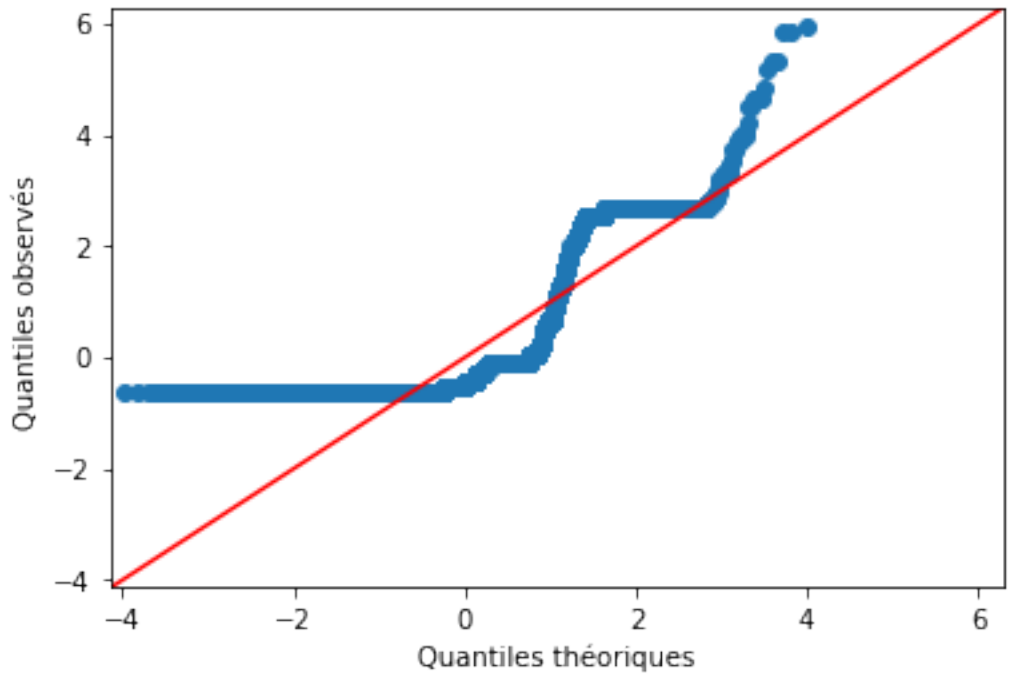
[5038]:
```
# Nombre de Quiz et vidéos par étudiants
video_quiz_etu = total_video_etu.merge(total_quiz_etu, on="Student_ID")
```

[5039]:
```
video_quiz_etu.loc[3139]
```

[5039]:
```
video    48
quiz     10
Name: 3139, dtype: int64
```

[5040]:
```
data = pd.concat([video_quiz_etu["quiz"], video_quiz_etu["video"]])
```

[5041]:
```
qqplot(data=data, fit=True, line="45")
plt.xlabel("Quantiles théoriques")
plt.ylabel("Quantiles observés ")
plt.savefig("../graph/distribution_video4.png")
plt.show()
```



[5042]:
```
# Test de Pearson
correlation, p_value = pearsonr(video_quiz_etu["quiz"], video_quiz_etu["video"])
```

[5043]:
```
correlation, p_value
```

[5043]: (0.8036026075037674, 0.0)

[5044]:
```
# Test de Spearman
correlation, p_value = spearmanr(video_quiz_etu["quiz"],␣
 ↪video_quiz_etu["video"])
```

[5045]:
```
correlation, p_value
```

[5045]: (0.804511796629543, 0.0)

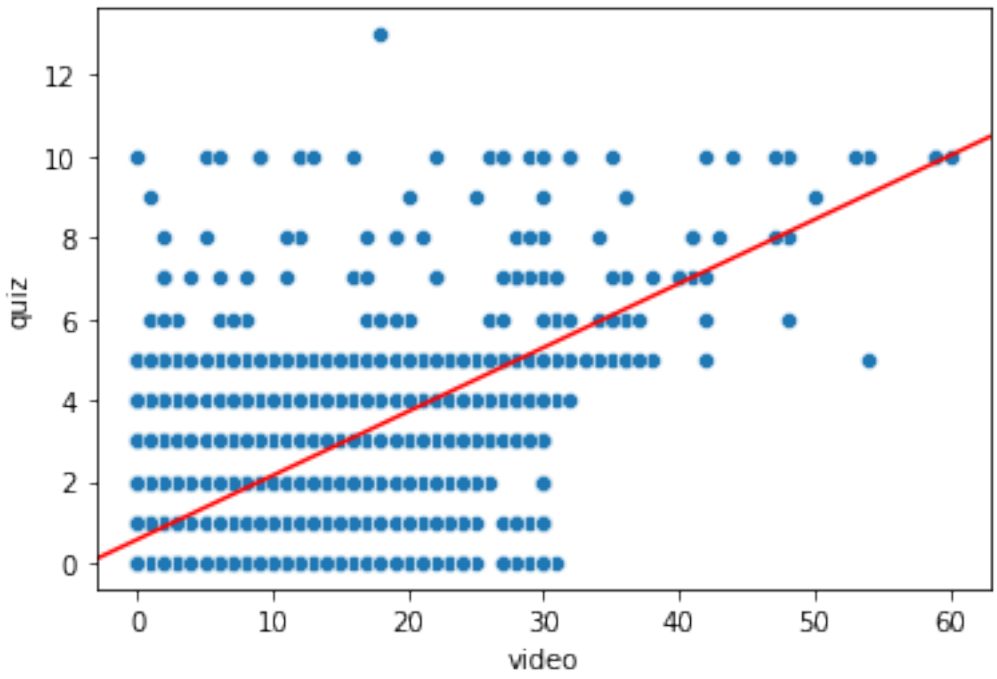
Il y a une forte corrélation (0.8) entre le nombre de videos vues et le nombre de quiz réalisés par un étudiant. La corrélation observée est statistiquement significative (p-value=0).

[5046]:
```
# Modèle de regression sur le nombre de vidéos selon le nombre de quiz
mdl_video_vs_quiz = ols("quiz ~ video", data=video_quiz_etu).fit()
```

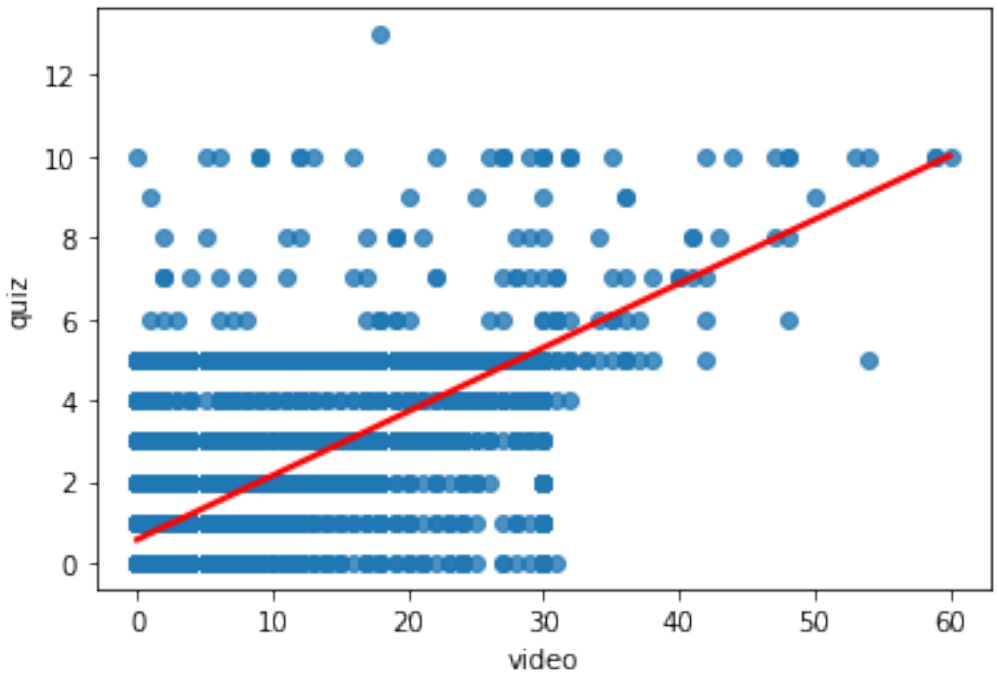
[5047]:
```
mdl_video_vs_quiz.summary()
```

[5047]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                   quiz   R-squared:                       0.646
Model:                            OLS   Adj. R-squared:                  0.646
Method:                 Least Squares   F-statistic:                 2.653e+04
Date:                Fri, 23 Jun 2023   Prob (F-statistic):               0.00
Time:                        20:00:51   Log-Likelihood:                -24981.
No. Observations:               14557   AIC:                            4.997e+04
Df Residuals:                   14555   BIC:                            4.998e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.5652      0.014     39.183      0.000       0.537       0.594
video          0.1575      0.001    162.896      0.000       0.156       0.159
==============================================================================
Omnibus:                     5315.454   Durbin-Watson:                   1.939
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            20116.783
Skew:                           1.821   Prob(JB):                         0.00
Kurtosis:                       7.461   Cond. No.                         19.3
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

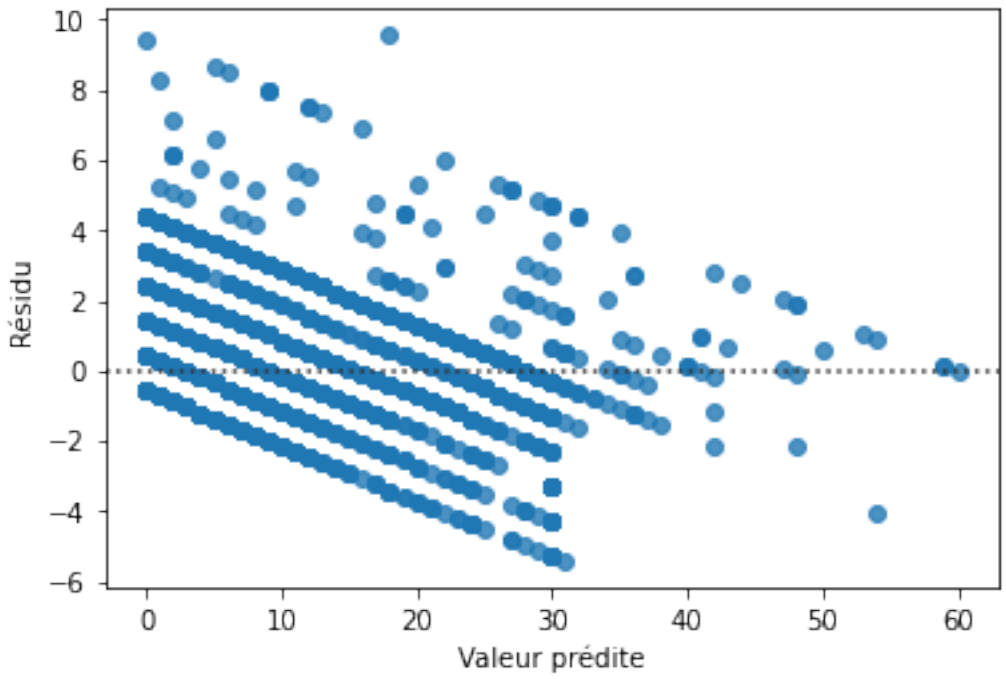
[5048]:
```
intercept, coef = mdl_video_vs_quiz.params
sns.scatterplot(data=video_quiz_etu, x="video", y="quiz")
plt.axline(xy1=(0,intercept), slope=coef, color="red")
plt.savefig("../graph/scatter2_regression.png")
plt.show()
```



[5049]:
```
fig, ax = plt.subplots()
sns.regplot(data=video_quiz_etu, x=video_quiz_etu["video"],␣
 ↪y=video_quiz_etu["quiz"], line_kws={"color": "red"}, ax=ax)
#plt.savefig("../graph/scatter_regression.png")
plt.show()
```



[5050]:
```
# Résidus du modèle de prédiction
fig, ax = plt.subplots()
sns.residplot(data=video_quiz_etu, x="video", y="quiz", ax=ax)
#ax.set_title("Résidus du modèle observé")
ax.set_ylabel("Résidu")
ax.set_xlabel("Valeur prédite")
plt.savefig("../graph/residus_regression.png")
plt.show()
```



[5051]:
```
# HDI des apprenants
student_hdi = base[["Student_ID", "New_HDI"]].dropna().set_index("Student_ID")
student_hdi.sort_values("Student_ID", inplace=True)
```

[5052]:
```
student_hdi
```

[5052]:
```
            New_HDI
Student_ID
45               TH
83                I
84                B
87               TH
94               TH
…                …
68282             B
68326            TH
68332             I
68365            TH
69565            TH

[8963 rows x 1 columns]
```

[5053]:
```
# Tableau des 3 variables dont 2 catégorielles et 1 continue
video_genre_hdi = etu_genre_video.merge(student_hdi, on="Student_ID")
```

[5054]:
```
video_genre_hdi
```

[5054]:
```
      Student_ID  video  Genre New_HDI
0             45     25  Femme      TH
1             83     22  Homme       I
2             84      8  Homme       B
3             87      1  Homme      TH
4             94      2  Homme      TH
…            …    …     …        …
8860       68282      1  Homme       B
8861       68326     30  Femme      TH
8862       68332      4  Femme       I
8863       68365      0  Femme      TH
8864       69565      9  Femme      TH

[8865 rows x 4 columns]
```

[5055]:
```
# Modèle linéaire sans interaction(genre, HDI, video)
mdl1 = ols("video ~ C(Genre) + C(New_HDI)", data=video_genre_hdi).fit()
```

[5056]:
```
mdl1.summary()
```

[5056]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  video   R-squared:                       0.057
Model:                            OLS   Adj. R-squared:                  0.056
Method:                 Least Squares   F-statistic:                     177.2
Date:                Fri, 23 Jun 2023   Prob (F-statistic):          1.41e-111
Time:                        20:00:52   Log-Likelihood:                -34351.
No. Observations:                8865   AIC:                         6.871e+04
Df Residuals:                    8861   BIC:                         6.874e+04
Df Model:                           3
Covariance Type:            nonrobust
===============================================================================
=====
                          coef    std err          t      P>|t|      [0.025
0.975]
-------------------------------------------------------------------------------
-----
Intercept               7.0011      0.432     16.211      0.000       6.155
7.848
C(Genre)[T.Homme]      -0.0959      0.267     -0.360      0.719      -0.618
0.427
C(New_HDI)[T.I]         4.6774      0.586      7.979      0.000       3.528
5.826
```

```
C(New_HDI)[T.TH]      8.6728      0.395     21.937      0.000       7.898
9.448
==============================================================================
Omnibus:                    11867.785   Durbin-Watson:                   1.891
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              639.581
Skew:                           0.218   Prob(JB):                    1.31e-139
Kurtosis:                       1.759   Cond. No.                         8.79
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

[5057]:
```
# ANOVA sans interaction
anova_table = sm.stats.anova_lm(mdl1, typ=1)
```

[5058]:
```
anova_table
```

[5058]:

	df	sum_sq	mean_sq	F	PR(>F)
C(Genre)	1.0	1.554895e+03	1554.895102	11.437356	7.229351e-04
C(New_HDI)	2.0	7.071483e+04	35357.416782	260.078874	1.738249e-110
Residual	8861.0	1.204643e+06	135.948823	NaN	NaN

[5059]:
```
# Modèle de regression avec interaction
mdl2 = ols("video ~ C(Genre)*C(New_HDI)", data=video_genre_hdi).fit()
```

[5060]:
```
mdl2.summary()
```

[5060]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  video   R-squared:                       0.057
Model:                            OLS   Adj. R-squared:                  0.057
Method:                 Least Squares   F-statistic:                     107.8
Date:                Fri, 23 Jun 2023   Prob (F-statistic):          7.64e-111
Time:                        20:00:52   Log-Likelihood:                -34347.
No. Observations:                8865   AIC:                         6.871e+04
Df Residuals:                    8859   BIC:                         6.875e+04
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
======================
                                 coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
```

```
----------------------
Intercept                                7.3310      0.968      7.573      0.000
5.434       9.229
C(Genre)[T.Homme]                       -0.4810      1.046     -0.460      0.646
-2.531       1.569
C(New_HDI)[T.I]                          2.7733      1.236      2.244      0.025
0.350       5.196
C(New_HDI)[T.TH]                         8.4673      0.995      8.506      0.000
6.516      10.419
C(Genre)[T.Homme]:C(New_HDI)[T.I]        2.8032      1.415      1.982      0.048
0.030       5.576
C(Genre)[T.Homme]:C(New_HDI)[T.TH]       0.1933      1.085      0.178      0.859
-1.933       2.320
==============================================================================
Omnibus:                    11724.976   Durbin-Watson:                   1.891
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              637.561
Skew:                           0.217   Prob(JB):                    3.59e-139
Kurtosis:                       1.760   Cond. No.                         33.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

[5061]:
```
# ANOVA avec interaction
anova_table = sm.stats.anova_lm(mdl2, typ=1)
```

[5062]:
```
anova_table
```

[5062]:

	df	sum_sq	mean_sq	F	PR(>F)
C(Genre)	1.0	1.554895e+03	1554.895102	11.443839	7.204212e-04
C(New_HDI)	2.0	7.071483e+04	35357.416782	260.226287	1.514733e-110
C(Genre):C(New_HDI)	2.0	9.541531e+02	477.076535	3.511225	2.990187e-02
Residual	8859.0	1.203688e+06	135.871810	NaN	NaN

[5063]:
```
# obtention de l'examen /certification par étudiant par session, selon le genre et l'HDI
df_exam = base.groupby(["Student_ID", "Itération", "Gender", "New_HDI"])[["Exam.bin", "Certif.bin"]].sum().reset_index(["Gender", "New_HDI"])
```

[5064]:
```
df_exam.rename(columns={"Exam.bin": "Exam", "Certif.bin": "Certif"},␣
↪inplace=True)
```

[5065]:
```
df_exam
```

[5065]:

Student_ID	Itération	Gender	New_HDI	Exam	Certif
45	1	une femme	TH	0	0
83	1	un homme	I	0	0
84	1	un homme	B	0	0
87	1	un homme	TH	0	0
94	1	un homme	TH	0	0
…		…	…	…	…
68282	3	un homme	B	0	0
68326	3	une femme	TH	1	0
68332	3	une femme	I	1	1
68365	3	une femme	TH	0	0
69565	3	une femme	TH	0	0

[8951 rows x 4 columns]

[5066]:
```
# Obtention de l'examen et/ou de la certifcation
df_exam["Exam_Certif"] = df_exam["Exam"] | df_exam["Certif"]
```

[5067]:
```
# Obtention ou non par itération
df_exam_v1 = df_exam.xs(1, level="Itération")
df_exam_v2 = df_exam.xs(2, level="Itération")
df_exam_v3 = df_exam.xs(3, level="Itération")
```

[5068]:
```
df_exam_v3["Exam_Certif"].unique()
```

[5068]:
```
array([1, 0])
```

[5069]:
```
# Generalized Linear Models (Formula du GLM)
formula = "Exam_Certif ~ Gender + New_HDI"
```

[5070]:
```
# Modèle logistic (GLM) de type binomial (variable dépendante binaire)
model = []
for n, version in enumerate([df_exam_v1, df_exam_v2, df_exam_v3]):
    model.append(glm(formula=formula, data=version, family=sm.families.
↪Binomial()))
```

[5071]:
```
# résultats de la fonction logistic par itération
result_v1 = model[0].fit()
result_v2 = model[1].fit()
result_v3 = model[2].fit()
```

[5072]:
```
result_v1.summary()
```

[5072]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:            Exam_Certif   No. Observations:                 5237
Model:                            GLM   Df Residuals:                     5233
Model Family:                Binomial   Df Model:                            3
Link Function:                  Logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -128.64
Date:                Fri, 23 Jun 2023   Deviance:                       257.28
Time:                        20:00:52   Pearson chi2:                 4.84e+03
No. Iterations:                     25   Pseudo R-squ. (CS):           0.001021
Covariance Type:            nonrobust
==============================================================================
=======
                            coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-------
Intercept              -5.9120      0.715     -8.266      0.000      -7.314
-4.510
Gender[T.une femme]      0.6235      0.456      1.367      0.172      -0.270
1.517
New_HDI[T.I]          -20.8950   1.75e+04     -0.001      0.999   -3.43e+04
3.42e+04
New_HDI[T.TH]            0.2241      0.756      0.297      0.767      -1.257
1.705
==============================================================================
=======
"""
```

[5073]:
```
mooc = ["V1", "V2", "V3"]
```

[5074]:
```
# Calcul des odds-ratio, p-value et ci pour chaque itération
full_odds_r, full_pvalues, full_ci = [], [], []
for n, version in enumerate([result_v1, result_v2, result_v3]):
    # OR
    odds = np.exp(version.params).round(3)
    odds_r = odds / odds["Intercept"]
    odds_r.name = mooc[n]
    full_odds_r.append(odds_r)
    # p-value
    p_values = version.pvalues.round(3)
    p_values.name = mooc[n]
    full_pvalues.append(p_values)
    # interval de confiance (ci)
    ci = version.conf_int(alpha=0.05) # intervalle à 95 %
```

```
    sup = np.exp(ci[1]) / np.exp(ci.loc["Intercept", 0])
    inf = np.exp(ci[0]) / np.exp(ci.loc["Intercept", 1])
    ci = "[" + inf.round(3).astype(str) + ", " + sup.round(3).astype(str) + "]"
    ci.name = mooc[n]
    full_ci.append(ci)
```

[5075]:
```
tab_coef = pd.concat([pd.DataFrame(full_odds_r).T, pd.DataFrame(full_pvalues).
↪T, pd.DataFrame(full_ci).T])
```

[5076]:
```
tab_coef
```

[5076]:

	V1	V2	V3
Intercept	1.0	1.0	1.0
Gender[T.une femme]	621.666667	2.32574	0.641167
New_HDI[T.I]	0.0	2.382688	0.837539
New_HDI[T.TH]	417.0	3.548975	0.658517
Intercept	0.0	0.0	0.157
Gender[T.une femme]	0.172	0.826	0.053
New_HDI[T.I]	0.999	0.849	0.813
New_HDI[T.TH]	0.767	0.004	0.311
Intercept	[0.061, 16.503]	[0.561, 1.783]	[0.518, 1.931]
Gender[T.une femme]	[69.397, 6844.388]	[1.451, 3.721]	[0.374, 1.099]
New_HDI[T.I]	[0.0, inf]	[1.125, 5.04]	[0.367, 1.907]
New_HDI[T.TH]	[25.872, 8259.698]	[1.959, 6.424]	[0.334, 1.297]

[5077]:
```
index = pd.MultiIndex.from_arrays([["odds-ratio", "odds-ratio", "odds-ratio",␣
↪"odds-ratio",
                                    "p-value", "p-value", "p-value", "p-value",
                                    "ci", "ci", "ci", "ci"],
                                    tab_coef.index], names=["type", "coef"])
```

[5078]:
```
tab_full = pd.DataFrame({"V1": list(tab_coef["V1"]), "V2":␣
↪list(tab_coef["V2"]), "V3": list(tab_coef["V3"])}, index=index)
```

[5079]:
```
tab_full.reset_index(inplace=True)
```

[5080]:
```
tab_full.set_index(["type"], inplace=True)
```

[5081]:
```
tab_full
```

[5081]:

type	coef	V1	V2	\
odds-ratio	Intercept	1.0	1.0	
odds-ratio	Gender[T.une femme]	621.666667	2.32574	
odds-ratio	New_HDI[T.I]	0.0	2.382688	
odds-ratio	New_HDI[T.TH]	417.0	3.548975	
p-value	Intercept	0.0	0.0	

```
p-value     Gender[T.une femme]              0.172           0.826
p-value            New_HDI[T.I]              0.999           0.849
p-value           New_HDI[T.TH]              0.767           0.004
ci                    Intercept     [0.061, 16.503]  [0.561, 1.783]
ci          Gender[T.une femme]  [69.397, 6844.388]  [1.451, 3.721]
ci                 New_HDI[T.I]          [0.0, inf]   [1.125, 5.04]
ci                New_HDI[T.TH]  [25.872, 8259.698]  [1.959, 6.424]

                         V3
type
odds-ratio              1.0
odds-ratio         0.641167
odds-ratio         0.837539
odds-ratio         0.658517
p-value               0.157
p-value               0.053
p-value               0.813
p-value               0.311
ci           [0.518, 1.931]
ci           [0.374, 1.099]
ci           [0.367, 1.907]
ci           [0.334, 1.297]
```

p > 0.05 : Non significatif (pas d'astérisque) 0.01 < p 0.05 : * (un astérisque) 0.001 < p 0.01 : ** (deux astérisques) p 0.001 : *** (trois astérisques)

[5082]:
```
ci_v1 = tab_full.loc["ci", ["coef", "V1"]]
or_v1 = tab_full.loc["odds-ratio", ["coef", "V1"]]
pval_v1 = tab_full.loc["p-value", ["coef", "V1"]]

ci_v2 = tab_full.loc["ci", ["coef", "V2"]]
or_v2 = tab_full.loc["odds-ratio", ["coef", "V2"]]
pval_v2 = tab_full.loc["p-value", ["coef", "V2"]]

ci_v3 = tab_full.loc["ci", ["coef", "V3"]]
or_v3 = tab_full.loc["odds-ratio", ["coef", "V3"]]
pval_v3 = tab_full.loc["p-value", ["coef", "V3"]]
```

[5083]:
```
ci_v1.rename(columns={'V1': 'CI'}, inplace=True)
ci_v2.rename(columns={'V2': 'CI'}, inplace=True)
ci_v3.rename(columns={'V3': 'CI'}, inplace=True)

or_v1.rename(columns={'V1': 'OR'}, inplace=True)
or_v2.rename(columns={'V2': 'OR'}, inplace=True)
or_v3.rename(columns={'V3': 'OR'}, inplace=True)

pval_v1.rename(columns={'V1': 'pval'}, inplace=True)
pval_v2.rename(columns={'V2': 'pval'}, inplace=True)
```

```
pval_v3.rename(columns={'V3': 'pval'}, inplace=True)
```

[5084]:
```
# Extraction des bornes inférieure et supérieure des CI
ci_v1[["ll", "hl"]] = ci_v1["CI"].str.strip("[]").str.split(",", expand=True).
 ↪astype(float)
ci_v2[["ll", "hl"]] = ci_v2["CI"].str.strip("[]").str.split(",", expand=True).
 ↪astype(float)
ci_v3[["ll", "hl"]] = ci_v3["CI"].str.strip("[]").str.split(",", expand=True).
 ↪astype(float)
```

[5085]:
```
# Tableaux des valeurs pour créer le forestplot
forest_v1 = ci_v1.merge(or_v1, on="coef").merge(pval_v1, on="coef")
forest_v2 = ci_v2.merge(or_v2, on="coef").merge(pval_v2, on="coef")
forest_v3 = ci_v3.merge(or_v3, on="coef").merge(pval_v3, on="coef")
```

[5086]:
```
# Renommage de l'intercept
forest_v1.iloc[0, 0] = "Réf (homme/HDI B)"
forest_v2.iloc[0, 0] = "Réf (homme/HDI B)"
forest_v3.iloc[0, 0] = "Réf (homme/HDI B)"
```

[5087]:
```
# Mise à zero des intervalle de confiance de l'intercept
forest_v1.loc[0, "ll"] = 0
forest_v1.loc[0, "hl"] = 0
forest_v2.loc[0, "ll"] = 0
forest_v2.loc[0, "hl"] = 0
forest_v3.loc[0, "ll"] = 0
forest_v3.loc[0, "hl"] = 0
```

[5088]:
```
forest_v1
```

[5088]:

	coef	CI	ll	hl	OR	pval
0	Réf (homme/HDI B)	[0.061, 16.503]	0.000	0.000	1.0	0.0
1	Gender[T.une femme]	[69.397, 6844.388]	69.397	6844.388	621.666667	0.172
2	New_HDI[T.I]	[0.0, inf]	0.000	inf	0.0	0.999
3	New_HDI[T.TH]	[25.872, 8259.698]	25.872	8259.698	417.0	0.767

[5089]:
```
forest_v2
```

[5089]:

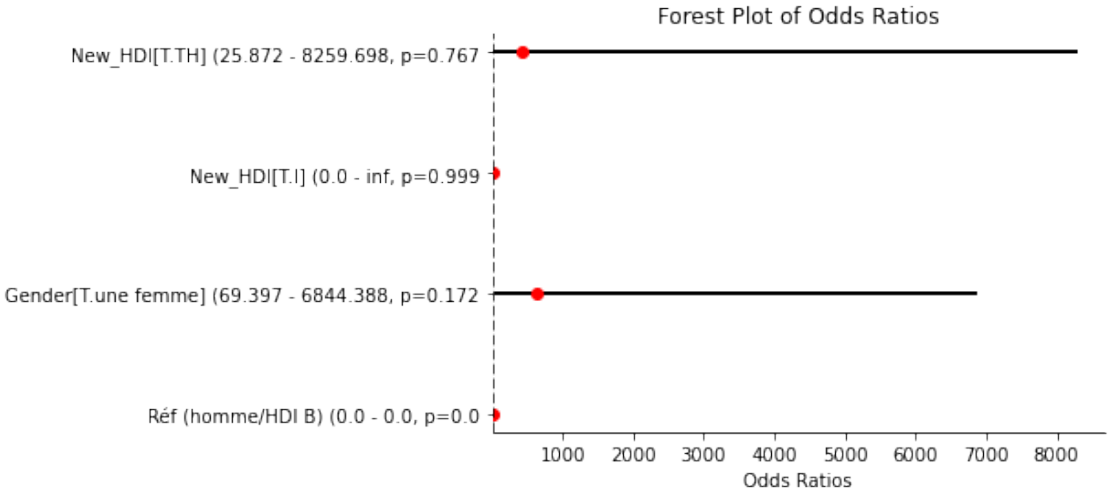
	coef	CI	ll	hl	OR	pval
0	Réf (homme/HDI B)	[0.561, 1.783]	0.000	0.000	1.0	0.0
1	Gender[T.une femme]	[1.451, 3.721]	1.451	3.721	2.32574	0.826

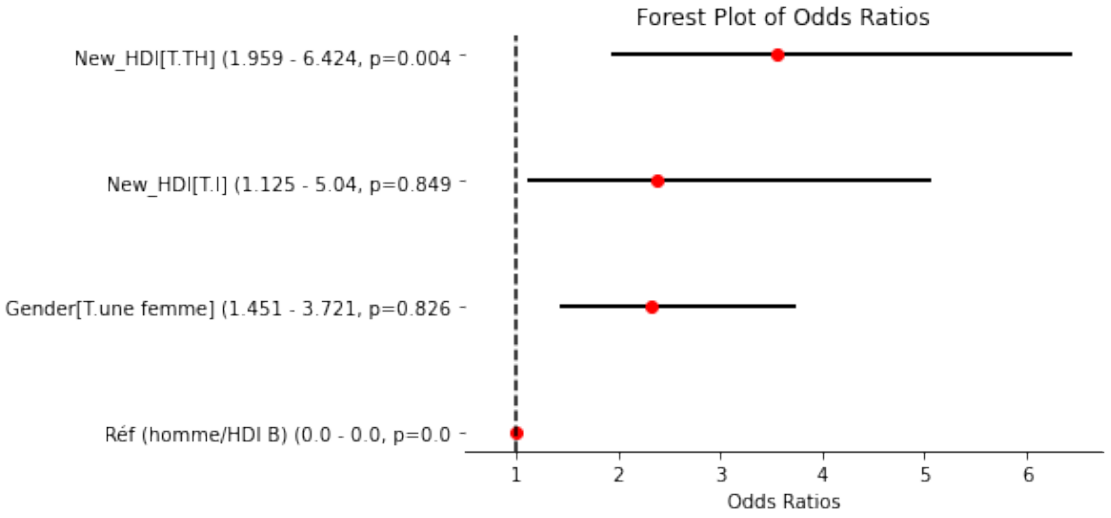
```
2          New_HDI[T.I]   [1.125, 5.04]  1.125  5.040  2.382688  0.849
3         New_HDI[T.TH]   [1.959, 6.424]  1.959  6.424  3.548975  0.004
```

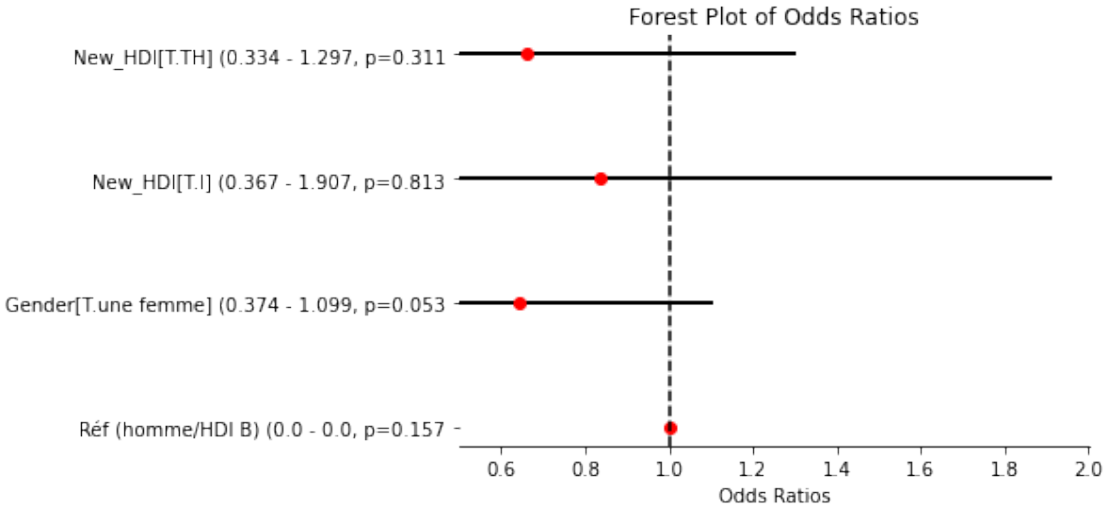
[5090]:
```
forest_v3
```

[5090]:
```
                   coef             CI     ll     hl        OR   pval
0    Réf (homme/HDI B)   [0.518, 1.931]  0.000  0.000       1.0  0.157
1  Gender[T.une femme]   [0.374, 1.099]  0.374  1.099  0.641167  0.053
2          New_HDI[T.I]   [0.367, 1.907]  0.367  1.907  0.837539  0.813
3         New_HDI[T.TH]   [0.334, 1.297]  0.334  1.297  0.658517  0.311
```

[5091]:
```
for version in [forest_v1, forest_v2, forest_v3]:
    # Données de la régression logistique
    odds_ratios = version["OR"].tolist()
    ci = list(zip(version["ll"], version["hl"]))
    ci_inf = version["ll"].astype(str)
    ci_sup = version["hl"].astype(str)
    labels = version["coef"] + " (" + ci_inf + " - " + ci_sup + ", p=" +␣
 ↪version["pval"].astype(str)

    # Création du forest plot
    fig, ax = plt.subplots()

    # OR, intervalles de confiance
    for i, (or_val, (ci_low, ci_high), label) in enumerate(zip(odds_ratios, ci,␣
 ↪labels)):
        ax.plot([ci_low, ci_high], [i, i], color="black", linewidth=2)
        ax.plot(or_val, i, 'ro')

    # Réglages des axes
    ax.set_xlim(left=0.5)
    ax.set_yticks(range(len(labels)))
    ax.set_yticklabels(labels)
    ax.set_xlabel('Odds Ratios')
    ax.axvline(1, color='black', linestyle='--')
    ax.spines["top"].set_visible(False)
    ax.spines["left"].set_visible(False)
    ax.spines["right"].set_visible(False)
    plt.title('Forest Plot of Odds Ratios')
    plt.show()
```


Forest Plot of Odds Ratios
New_HDI[T.TH] (25.872 - 8259.698, p=0.767
New_HDI[T.I] (0.0 - inf, p=0.999
Gender[T.une femme] (69.397 - 6844.388, p=0.172
Réf (homme/HDI B) (0.0 - 0.0, p=0.0
1000
2000
3000
4000
5000
6000
7000
8000
Odds Ratios


Forest Plot of Odds Ratios
New_HDI[T.TH] (1.959 - 6.424, p=0.004
New_HDI[T.I] (1.125 - 5.04, p=0.849
Gender[T.une femme] (1.451 - 3.721, p=0.826
Réf (homme/HDI B) (0.0 - 0.0, p=0.0
1
2
3
4
5
6
Odds Ratios



[5092]:
```
# Regression de Poisson
model = []
for n, version in enumerate([df_exam_v1, df_exam_v2, df_exam_v3]):
    model.append(glm(formula=formula, data=version, family=sm.families.
 ↪Poisson()))
```

[5093]:
```
# résultats de la fonction logistic par itération
result_v1 = model[0].fit()
result_v2 = model[1].fit()
result_v3 = model[2].fit()
```

[5094]:
```
# Calcul des odds-ratio, p-value et ci pour chaque itération
full_odds_r, full_pvalues, full_ci = [], [], []
for n, version in enumerate([result_v1, result_v2, result_v3]):
    # OR
    odds = np.exp(version.params).round(3)
    odds_r = odds / odds["Intercept"]
    odds_r.name = mooc[n]
    full_odds_r.append(odds_r)
    # p-value
    p_values = version.pvalues.round(3)
    p_values.name = mooc[n]
    full_pvalues.append(p_values)
    # interval de confiance (ci)
    ci = version.conf_int(alpha=0.05) # intervalle à 95 %
    sup = np.exp(ci[1]) / np.exp(ci.loc["Intercept", 0])
    inf = np.exp(ci[0]) / np.exp(ci.loc["Intercept", 1])
    ci = "[" + inf.round(3).astype(str) + ", " + sup.round(3).astype(str) + "]"
    ci.name = mooc[n]
```

```
    full_ci.append(ci)
```

[5095]:
```
tab_coef = pd.concat([pd.DataFrame(full_odds_r).T, pd.DataFrame(full_pvalues).
 ↪T, pd.DataFrame(full_ci).T])
```

[5096]:
```
index = pd.MultiIndex.from_arrays([["odds-ratio", "odds-ratio", "odds-ratio",␣
 ↪"odds-ratio",
                                    "p-value", "p-value", "p-value", "p-value",
                                    "ci", "ci", "ci", "ci"],
                                    tab_coef.index], names=["type", "coef"])
```

[5097]:
```
tab_full = pd.DataFrame({"V1": list(tab_coef["V1"]), "V2":␣
 ↪list(tab_coef["V2"]), "V3": list(tab_coef["V3"])}, index=index)
tab_full.reset_index(inplace=True)
tab_full.set_index(["type"], inplace=True)
```

[5098]:
```
ci_v1 = tab_full.loc["ci", ["coef", "V1"]]
or_v1 = tab_full.loc["odds-ratio", ["coef", "V1"]]
pval_v1 = tab_full.loc["p-value", ["coef", "V1"]]

ci_v2 = tab_full.loc["ci", ["coef", "V2"]]
or_v2 = tab_full.loc["odds-ratio", ["coef", "V2"]]
pval_v2 = tab_full.loc["p-value", ["coef", "V2"]]

ci_v3 = tab_full.loc["ci", ["coef", "V3"]]
or_v3 = tab_full.loc["odds-ratio", ["coef", "V3"]]
pval_v3 = tab_full.loc["p-value", ["coef", "V3"]]
```

[5099]:
```
ci_v1.rename(columns={'V1': 'CI'}, inplace=True)
ci_v2.rename(columns={'V2': 'CI'}, inplace=True)
ci_v3.rename(columns={'V3': 'CI'}, inplace=True)

or_v1.rename(columns={'V1': 'OR'}, inplace=True)
or_v2.rename(columns={'V2': 'OR'}, inplace=True)
or_v3.rename(columns={'V3': 'OR'}, inplace=True)

pval_v1.rename(columns={'V1': 'pval'}, inplace=True)
pval_v2.rename(columns={'V2': 'pval'}, inplace=True)
pval_v3.rename(columns={'V3': 'pval'}, inplace=True)
```

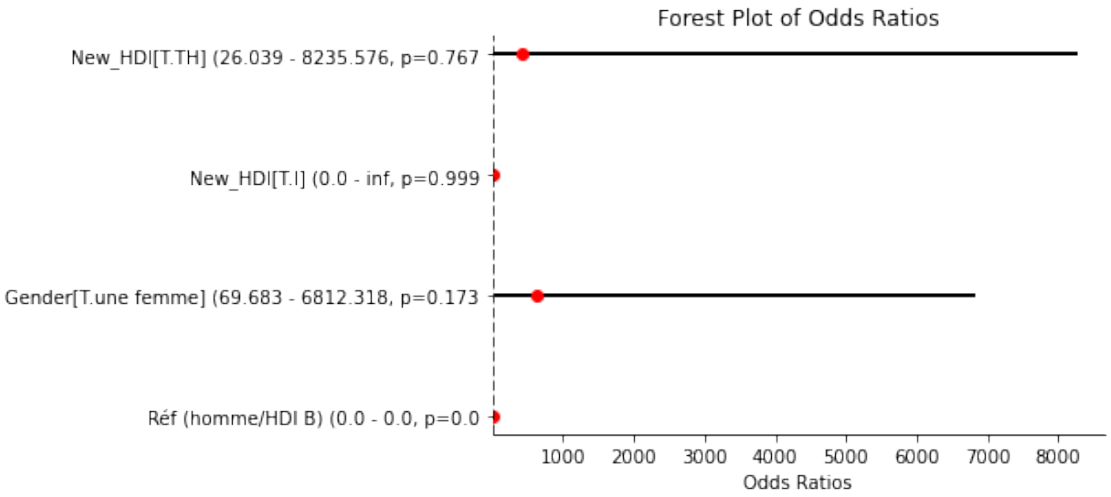
[5100]:
```
# Extraction des bornes inférieure et supérieure des CI
ci_v1[["ll", "hl"]] = ci_v1["CI"].str.strip("[]").str.split(",", expand=True).
 ↪astype(float)
ci_v2[["ll", "hl"]] = ci_v2["CI"].str.strip("[]").str.split(",", expand=True).
 ↪astype(float)
ci_v3[["ll", "hl"]] = ci_v3["CI"].str.strip("[]").str.split(",", expand=True).
 ↪astype(float)
```

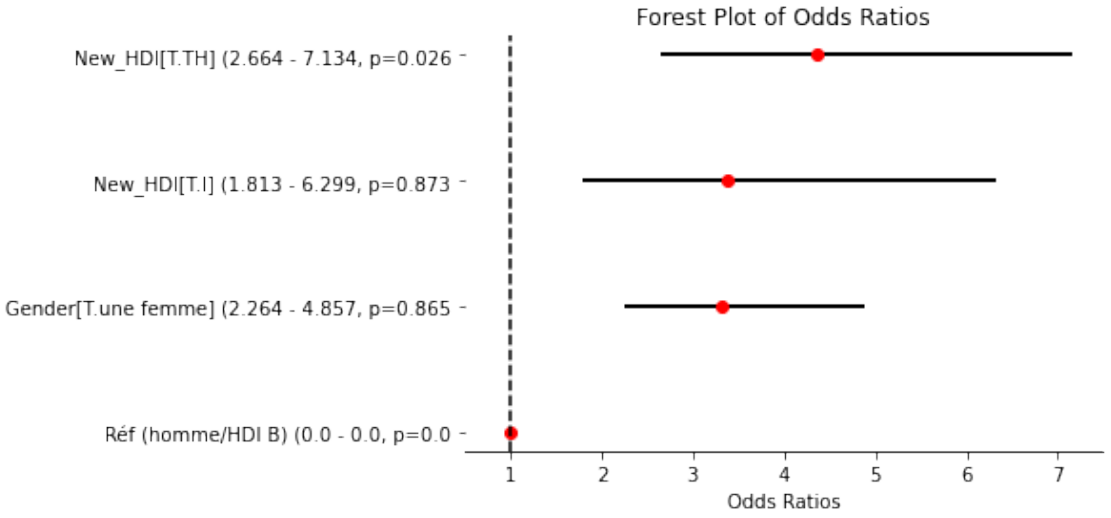
[5101]:
```
# Tableaux des valeurs pour créer le forestplot
forest_v1 = ci_v1.merge(or_v1, on="coef").merge(pval_v1, on="coef")
forest_v2 = ci_v2.merge(or_v2, on="coef").merge(pval_v2, on="coef")
forest_v3 = ci_v3.merge(or_v3, on="coef").merge(pval_v3, on="coef")
```

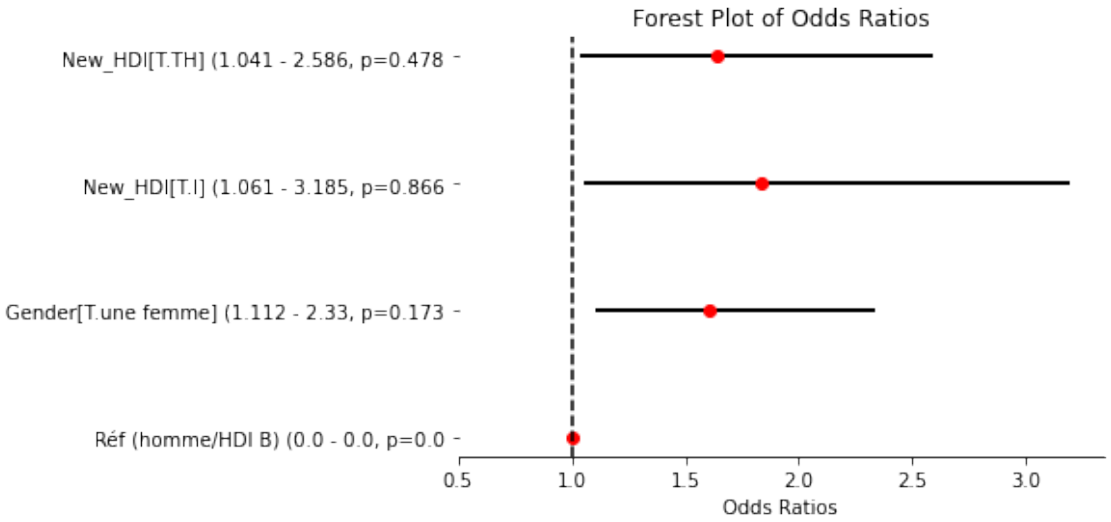
[5102]:
```
# Renommage de l'intercept
forest_v1.iloc[0, 0] = "Réf (homme/HDI B)"
forest_v2.iloc[0, 0] = "Réf (homme/HDI B)"
forest_v3.iloc[0, 0] = "Réf (homme/HDI B)"
```

[5103]:
```
# Mise à zero des intervalle de confiance de l'intercept
forest_v1.loc[0, "ll"] = 0
forest_v1.loc[0, "hl"] = 0
forest_v2.loc[0, "ll"] = 0
forest_v2.loc[0, "hl"] = 0
forest_v3.loc[0, "ll"] = 0
forest_v3.loc[0, "hl"] = 0
```

[5104]:
```
for version in [forest_v1, forest_v2, forest_v3]:
    # Données de la régression logistique
    odds_ratios = version["OR"].tolist()
    ci = list(zip(version["ll"], version["hl"]))
    ci_inf = version["ll"].astype(str)
    ci_sup = version["hl"].astype(str)
    labels = version["coef"] + " (" + ci_inf + " - " + ci_sup + ", p=" +␣
 ↪version["pval"].astype(str)

    # Création du forest plot
    fig, ax = plt.subplots()

    # OR, intervalles de confiance
    for i, (or_val, (ci_low, ci_high), label) in enumerate(zip(odds_ratios, ci,␣
 ↪labels)):
        ax.plot([ci_low, ci_high], [i, i], color="black", linewidth=2)
        ax.plot(or_val, i, 'ro')

    # Réglages des axes
    ax.set_xlim(left=0.5)
    ax.set_yticks(range(len(labels)))
    ax.set_yticklabels(labels)
    ax.set_xlabel('Odds Ratios')
    ax.axvline(1, color='black', linestyle='--')
    ax.spines["top"].set_visible(False)
    ax.spines["left"].set_visible(False)
    ax.spines["right"].set_visible(False)
    plt.title('Forest Plot of Odds Ratios')
    plt.show()
```


Forest Plot of Odds Ratios
New_HDI[T.TH] (26.039 - 8235.576, p=0.767
New_HDI[T.I] (0.0 - inf, p=0.999
Gender[T.une femme] (69.683 - 6812.318, p=0.173
Réf (homme/HDI B) (0.0 - 0.0, p=0.0
1000
2000
3000
4000
5000
6000
7000
8000
Odds Ratios


Forest Plot of Odds Ratios
New_HDI[T.TH] (2.664 - 7.134, p=0.026
New_HDI[T.I] (1.813 - 6.299, p=0.873
Gender[T.une femme] (2.264 - 4.857, p=0.865
Réf (homme/HDI B) (0.0 - 0.0, p=0.0
1
2
3
4
5
6
7
Odds Ratios


Forest Plot of Odds Ratios
New_HDI[T.TH] (1.041 - 2.586, p=0.478
New_HDI[T.I] (1.061 - 3.185, p=0.866
Gender[T.une femme] (1.112 - 2.33, p=0.173
Réf (homme/HDI B) (0.0 - 0.0, p=0.0
0.5
1.0
1.5
2.0
2.5
3.0
Odds Ratios