Evalutation statistique : régression logistique

Library

In [39]:
```
import numpy as np
import pandas as pd
from sas7bdat import SAS7BDAT

import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.graphics.gofplots import ProbPlot


import scipy.stats as stats
from scipy.stats import chi2
from scipy.stats import pearsonr, ttest_ind

import statsmodels.api as sm
from statsmodels.stats.stattools import durbin_watson


from sklearn.model_selection import cross_val_score
from sklearn.metrics import auc, precision_recall_curve, confusion_matrix
from sklearn.metrics import roc_curve, auc
from sklearn.calibration import calibration_curve
```

DataFrame

In [40]:
```
file_path = './data/individus_ct2013.sas7bdat'
with SAS7BDAT(file_path) as reader:
    df = reader.to_data_frame()
df.info()
```

```
[individus_ct2013.sas7bdat] column count mismatch
[individus_ct2013.sas7bdat] [individus_ct2013.sas7bdat] column count mismatch
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33673 entries, 0 to 33672
Columns: 541 entries, C1C to revmenscc_drap
dtypes: float64(76), object(465)
memory usage: 139.0+ MB
```

Périmètre de l'étude

- On s'intérresse uniquement aux individus vivant en couple dans le même logement.

In [41]:
```
df = df.query('COUPLE == "1"')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 24024 entries, 2 to 33672
Columns: 541 entries, C1C to revmenscc_drap
dtypes: float64(76), object(465)
memory usage: 99.3+ MB
```

Variable dépendante binaire

In [42]:
```
df['FPA'] = df.apply(lambda row: 1 if (row['PRACT'] == '1' and row['SEXE'

nb_FPA = df['FPA'].sum()        # Compter le nombre de lignes où 'femme_p

print("Nombre de lignes où 'femme_principal_apporteur' est égal à", nb_FP
```

```
Nombre de lignes où 'femme_principal_apporteur' est égal à 5080
```

Sélection des variables

In [43]:
```
variables_dict = {
    'etat_civil_familial': ['ANAIS', 'ETAMATRI', 'PACS', 'MER1E', 'PER1E'
    'situation_travail': ['SITUA', 'CJSITUA', 'RABS', 'STATUTEXT', 'METIE
    'revenus_conditions_vie': ['REVMENUC', 'AIDFAM'],
    'caracteristiques_logement_familial': ['TYPOLOG', 'TYPMEN15', 'NPERS'
}

# Accès aux listes individuelles
print("* Etat civil et familial:", variables_dict['etat_civil_familial'])
print("* Situation travail:", variables_dict['situation_travail'])
print("* Revenus et conditions de vie:", variables_dict['revenus_conditio
print("* Caractéristiques du logement et familial:", variables_dict['cara
```

```
* Etat civil et familial: ['ANAIS', 'ETAMATRI', 'PACS', 'MER1E', 'PER1E']
* Situation travail: ['SITUA', 'CJSITUA', 'RABS', 'STATUTEXT', 'METIER']
* Revenus et conditions de vie: ['REVMENUC', 'AIDFAM']
* Caractéristiques du logement et familial: ['TYPOLOG', 'TYPMEN15', 'NPERS',
'NACTIFS']
```

Identification des types de variables

In [44]:
```
explicatives = [var for sublist in variables_dict.values() for var in sub

numericals = [var for var in explicatives if pd.api.types.is_numeric_dtyp
categoricals = [var for var in explicatives if isinstance(df[var].dtype,
target = ['FPA']

print(f"numericals: {numericals}")
print(f"categoricals: {categoricals}")

columns_to_keep = numericals + categoricals + target
df = df[columns_to_keep]
```

```
numericals: ['ANAIS', 'REVMENUC', 'NPERS', 'NACTIFS']
categoricals: ['ETAMATRI', 'PACS', 'MER1E', 'PER1E', 'SITUA', 'CJSITUA', 'RAB
'STATUTEXT', 'METIER', 'AIDFAM', 'TYPOLOG', 'TYPMEN15']
```

Nettoyage

In [45]:
```
df.replace('', 0, inplace=True)
df.replace(' ', 0, inplace=True)
df.fillna(0, inplace=True)

nan_count = df.isna().sum()
zero_count = (df == 0).sum()

print("Nombre de NaN par colonne :")
print(nan_count)

print("\nNombre de 0 par colonne :")
print(zero_count)
```

```
df.head(20)
```

```
Nombre de NaN par colonne :
ANAIS        0
REVMENUC     0
NPERS        0
NACTIFS      0
ETAMATRI     0
PACS         0
MER1E        0
PER1E        0
SITUA        0
CJSITUA      0
RABS         0
STATUTEXT    0
METIER       0
AIDFAM       0
TYPOLOG      0
TYPMEN15     0
FPA          0
dtype: int64

Nombre de 0 par colonne :
ANAIS            0
REVMENUC      1035
NPERS          112
NACTIFS         78
ETAMATRI         0
PACS         17004
MER1E           18
PER1E           38
SITUA            4
CJSITUA          4
RABS         21471
STATUTEXT    19078
METIER           0
AIDFAM       23932
TYPOLOG          0
TYPMEN15         0
FPA          18944
dtype: int64
```

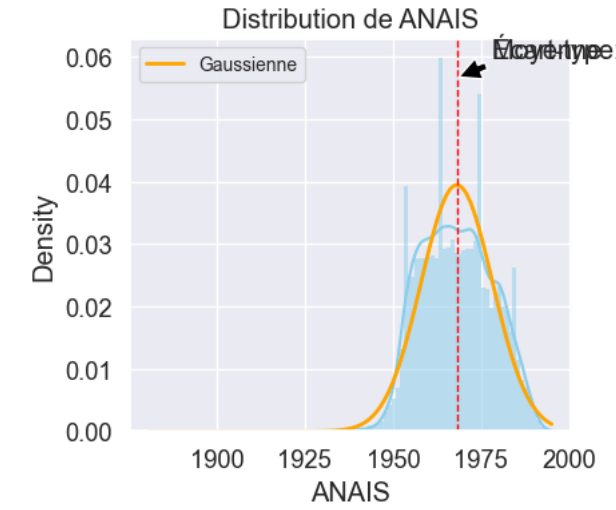
Out[45]:

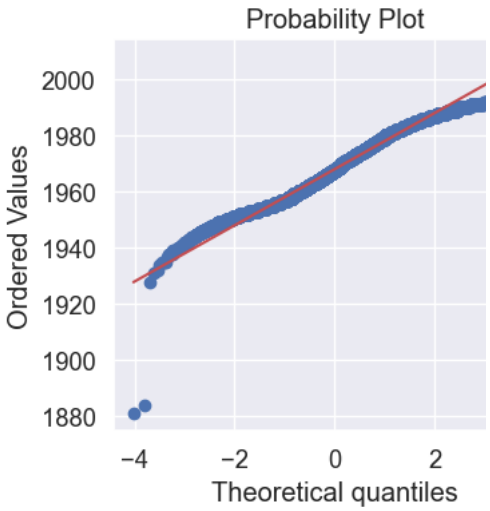
ANAIS	REVMENUC	NPERS	NACTIFS	ETAMATRI	PACS	MER	E	P
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					
.	.	.	.					

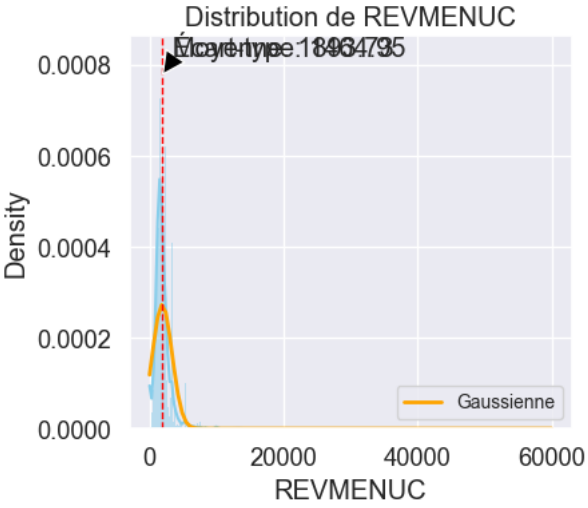
Distribution

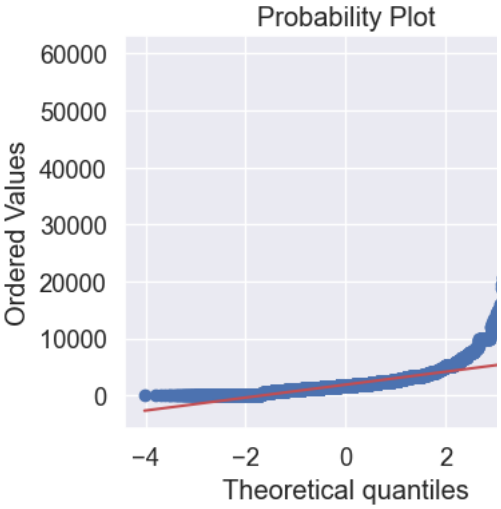
In [46]:
```
def plot_continuous_variables(df, numericals):
    # Créer une grille de sous-plots avec 2 colonnes
    num_vars = len(numericals)
    fig, axes = plt.subplots(num_vars, 2, figsize=(10, 4*num_vars))

    for i, var in enumerate(numericals):
        # Distribution de la variable (courbe de Gauss)
        ax = axes[i, 0]
        ax.set_title(f'Distribution de {var}')
        ax.grid(True)

        # Utiliser seaborn pour un tracé rapide de la distribution (peut
        sns.histplot(df[var], kde=True, ax=ax, color='skyblue', stat='den

        # Tracer la courbe de Gauss correspondante
        xmin, xmax = df[var].min(), df[var].max()
        mean, std_dev = df[var].mean(), df[var].std()
        x = np.linspace(xmin, xmax, 100)
        ax.plot(x, stats.norm.pdf(x, mean, std_dev), label='Gaussienne',
```

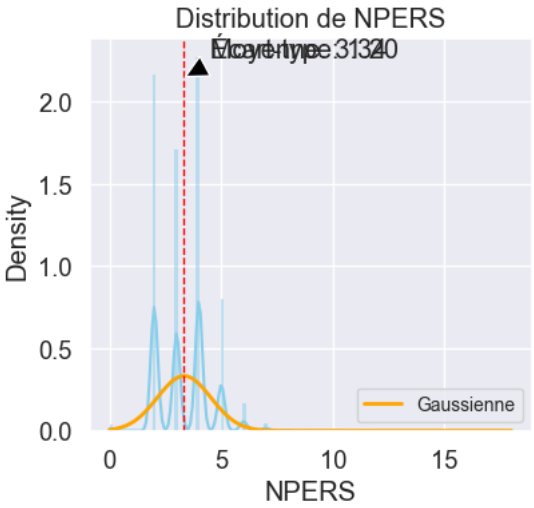
```
        ax.legend(prop={'size': 10})

        # Ajouter des annotations pour la moyenne et l'écart-type
        ax.axvline(mean, color='red', linestyle='dashed', linewidth=1)
        ax.annotate(f'Moyenne: {mean:.2f}', xy=(mean, ax.get_ylim()[1]*0.
                    arrowprops=dict(facecolor='black', shrink=0.05))
        ax.annotate(f'Écart-type: {std_dev:.2f}', xy=(mean + std_dev, ax.

        # Q-Q plot de la variable
        ax = axes[i, 1]
        ax.set_title(f'Q-Q plot de {var}')
        ax.grid(True)

        # Utiliser scipy.stats pour calculer le Q-Q plot
        stats.probplot(df[var], dist="norm", plot=ax)

    # Ajuster l'espacement entre les sous-graphiques
    plt.tight_layout()
    plt.show()

plot_continuous_variables(df, numericals)
```

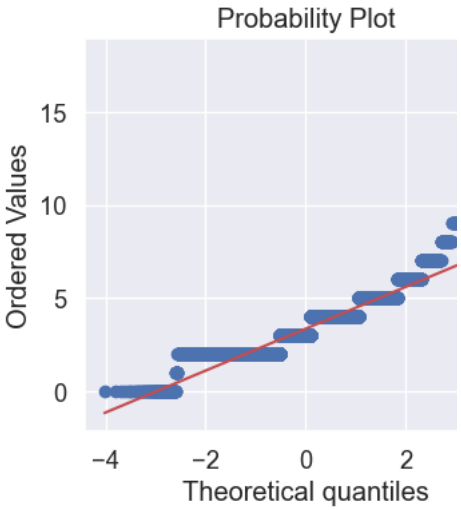

Distribution de ANAIS
Gaussienne
Density
ANAIS

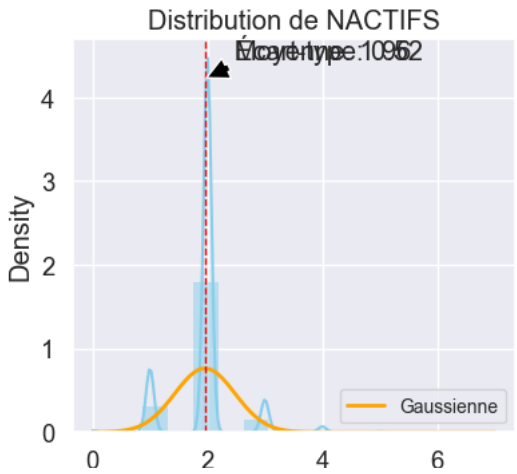

Probability Plot
Ordered Values
Theoretical quantiles

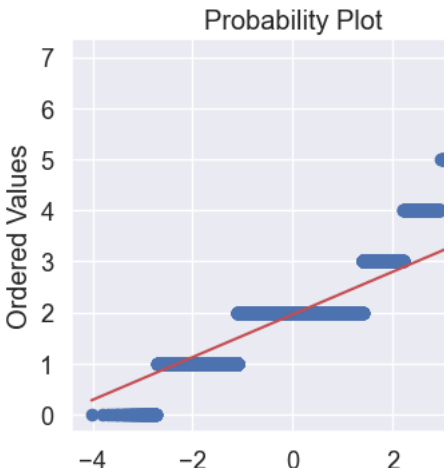

Distribution de REVMENUC
Gaussienne
Density
REVMENUC


Probability Plot
Ordered Values
Theoretical quantiles


Distribution de NPERS
Gaussienne
Density
NPERS


Probability Plot
Ordered Values
Theoretical quantiles


Distribution de NACTIFS
Gaussienne
Density

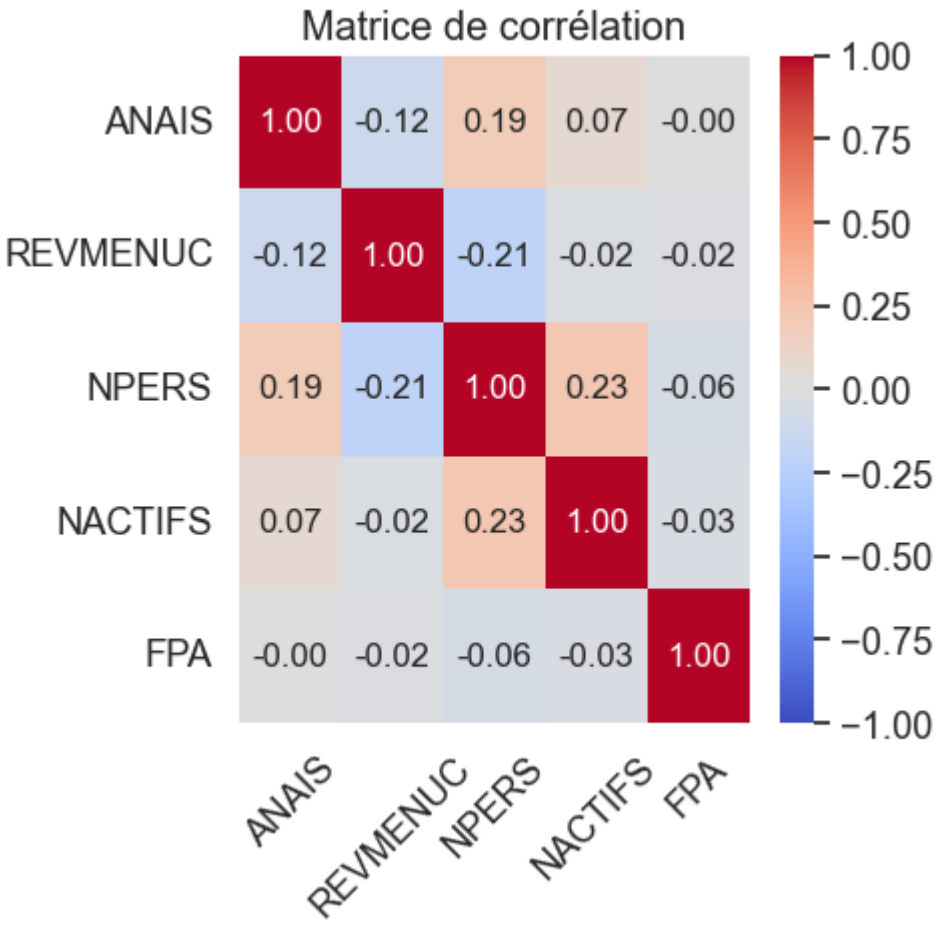

Probability Plot
Ordered Values

Analyse des dépendances

( ) Numericals_____

Analyse de corrélation

In [47]:
```
def plot_correlation_matrix(df, numericals, target):

    corr_matrix = df[numericals + target].corr()
    plt.figure(figsize=(5, 5))
    sns.set(font_scale=1.2)

    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", anno

    # Ajouter le titre et ajuster les marges
    plt.title('Matrice de corrélation')
    plt.yticks(rotation=0)
    plt.xticks(rotation=45)
    plt.tight_layout()

    # Afficher le plot
    plt.show()

plot_correlation_matrix(df, numericals, target)
```



Test de Student

- Hypothèse nulle (H0) : Il n'y a pas de différence significative entre les moyennes des variables continues pour les groupes définis

par la variable 'FPA'.
- Hypothèse alternative (H1) : Il y a une différence significative entre les moyennes des variables continues pour les groupes définis par la variable 'FPA'.

In [48]:
```
def test_association_with_fpa(df, numericals, target, alpha=0.05):
    print("___Test de Student___\n")

    new_columns = numericals + target
    groups = df[new_columns].groupby(target)
    group0 = groups.get_group((0,))
    group1 = groups.get_group((1,))


    results = []

    for var in numericals:
        t_stat, p_value = stats.ttest_ind(group0[var], group1[var], equal

        # Interpréter les résultats
        if np.isnan(p_value):
            significatif = "NaN"
        elif p_value < alpha:
            significatif = "yes"
        else:
            significatif = "no"

        # Ajouter les résultats à la liste results
        results.append({
        'Variable': var,
        'Statistique t': t_stat,
        'Valeur p': p_value,
        'Significatif': significatif,
        'Impact': abs(t_stat)
        })

    results_sorted = sorted(results, key=lambda x: x['Impact'], reverse=T
    df_results = pd.DataFrame(results_sorted)
    df_styled = df_results.style.apply(lambda row: ['background-color: li

    impactful_variables = df_results[df_results['Significatif'] == 'yes']
    print(f"Variables significatives: {impactful_variables}")

    return df_styled

test_association_with_fpa(df, numericals, target, alpha=0.05)
```

```
___Test de Student___

Variables significatives: ['NPERS', 'NACTIFS', 'REVMENUC']
```

Out[48]:

	Variable	Statistique t	Valeur p	Significatif	Impact
	NPERS	.	.	yes	.
	NACTIFS	.	.	yes	.
	REVMENUC	.	.	yes	.
	ANAIS	.	.	no	.

Selection numercials_

In [49]:
```
# Filtrage des variables continues
numericals = ['NPERS', 'NACTIFS', 'REVMENUC']

new_columns = numericals + categoricals + target
df = df[new_columns].copy()
```

( ) Categoricals____

Tableaux de contingence

In [50]:
```
def create_contingency_tables(df, categoricals, target='FPA'):

    num_categoricals = len(categoricals)
    fig, axs = plt.subplots(num_categoricals, 2, figsize=(9, num_categori

    for i, var in enumerate(categoricals):
        # Créer un tableau de contingence entre la variable catégorielle
        contingency_table = pd.crosstab(df[var], df[target])

        # Afficher le tableau de contingence à gauche
        axs[i, 0].set_title(f'Tableau de contingence pour {var} et {targe
        sns.heatmap(contingency_table, annot=True, fmt='d', cmap='Blues',

        # Créer un graphique à droite montrant la distribution de 'FPA' p
        sns.countplot(x=var, hue=target, data=df, ax=axs[i, 1])
        axs[i, 1].set_title(f'Distribution de {target} par {var}')
        axs[i, 1].legend(title=target)

        # Ajuster l'espacement entre les subplots
        plt.tight_layout()

    # Afficher les graphiques
    plt.show()

create_contingency_tables(df, categoricals, target='FPA')
```

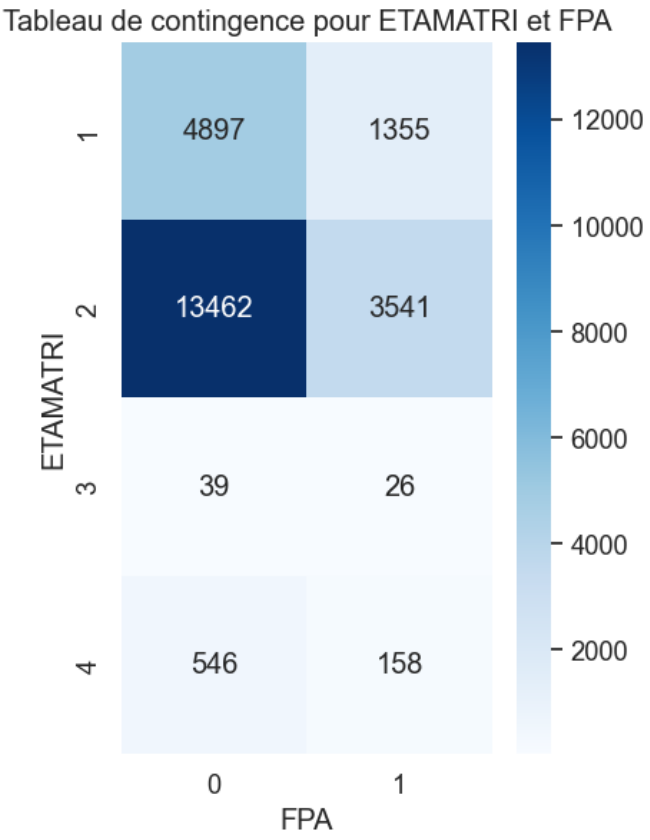
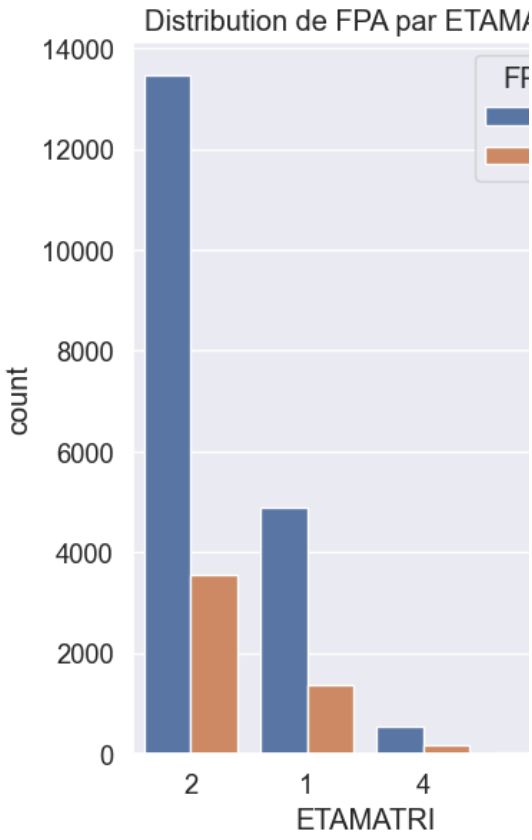


Tableau de contingence pour ETAMATRI et FPA
ETAMATRI
1
2
3
4
4897
1355
13462
3541
39
26
546
158
0
1
FPA
12000
10000
8000
6000
4000
2000


Distribution de FPA par ETAMA
14000
12000
10000
8000
6000
4000
2000
0
count
2
1
4
ETAMATRI

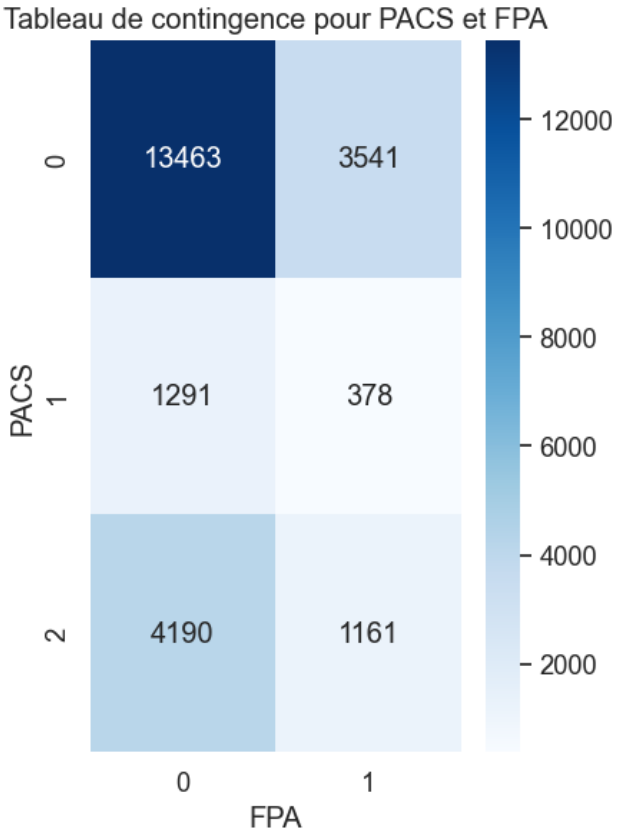
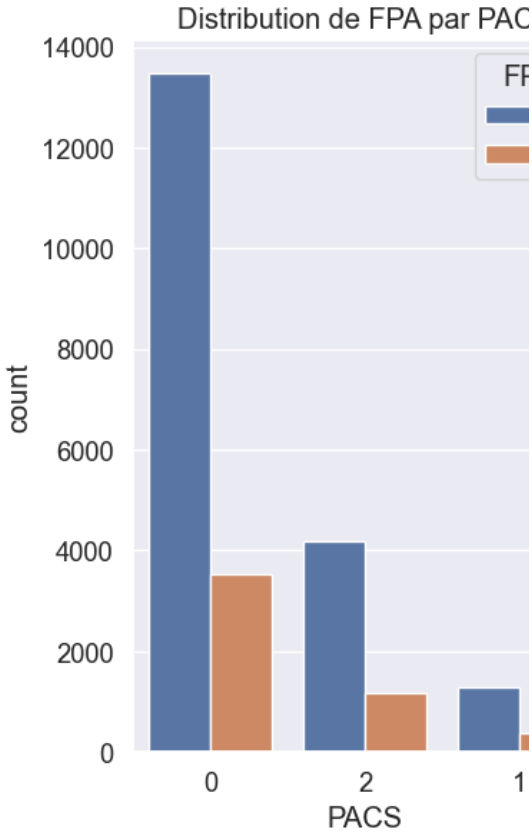


Tableau de contingence pour PACS et FPA
PACS
0
1
2
13463
3541
1291
378
4190
1161
0
1
FPA
12000
10000
8000
6000
4000
2000


Distribution de FPA par PAC
14000
12000
10000
8000
6000
4000
2000
0
count
0
2
1
PACS

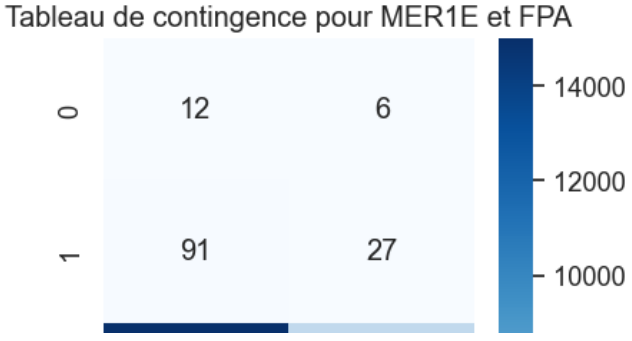
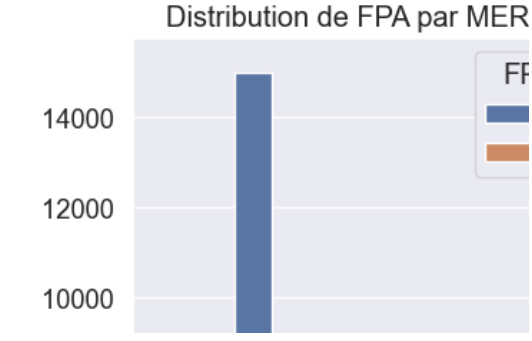


Tableau de contingence pour MER1E et FPA
0
1
12
6
91
27
14000
12000
10000


Distribution de FPA par MER
14000
12000
10000

Test du χ^2 pour évaluer leur association avec la variable dépendante.

- Hypothèse nulle (H0) : Il n'y a pas de relation entre la variable catégorielle et la variable cible 'FAP'.
- Hypothèse alternative (H1) : Il y a une relation entre la variabl catégorielle et la variable cible 'FAP'.

In [51]:
```python
def khi2_test_association(df, categoricals, target='FAP', alpha=0.05):
    print("___Test du Khi2___\n")
    results = []

    for var in categoricals:
        # Créer un tableau de contingence entre la variable catégorielle
        contingency_table = pd.crosstab(df[var], df[target])

        # Effectuer le test du chi-deux
        chi2, p_value, _, _ = stats.chi2_contingency(contingency_table)

        # Interpréter les résultats
        if p_value < alpha:
            impact = 'Yes'
        else:
            impact = 'No'

        # Ajouter les résultats à la liste
        results.append({
            'variable': var,
            'statistique': chi2,
            'p-value': p_value,
            'impact': impact
        })

    results_sorted = sorted(results, key=lambda x: x['statistique'], reve
    df_results = pd.DataFrame(results_sorted)

    # Filtrer les variables dont l'impact est 'Yes'
    impactful_variables = df_results[df_results['impact'] == 'Yes']['vari

    # Imprimer la liste des variables avec impact 'Yes'
    print("Categoricals with statistical significatif impact:", impactful

    # Appliquer le style conditionnel pour mettre en évidence les lignes
    df_styled = df_results.style.apply(lambda row: ['background-color: li

    return df_styled

khi2_test_association(df, categoricals, target='FPA', alpha=0.05)
```

```
___Test du Khi2___

Categoricals with statistical significatif impact: ['CJSITUA', 'STATUTEXT',
'TYPMEN15', 'METIER', 'RABS', 'SITUA', 'ETAMATRI']
```

Out[51]:

variable	statistique	p-value	impact
CJSITUA	.	.	Yes
STATUTEXT	.	.	Yes
TYPMEN	.	.	Yes
METIER	.	.	Yes
RABS	.	.	Yes
SITUA	.	.	Yes
ETAMATRI	.	.	Yes
AIDFAM	.	.	No
TYPOLOG	.	.	No
PACS	.	.	No
MER E	.	.	No
PER E	.	.	No

Test ANOVA (Analyse de variance)

- H0 (hypothèse nulle) : Les moyennes des groupes définis par l variable catégorielle sont égales.
- H1 (hypothèse alternative) : Au moins une paire de moyennes d groupes est différente.

In [52]:

```
def anova_test(df, categoricals, target='FPA', alpha=0.05):
    print("___Test ANOVA___\n")
    results = []

    for var in categoricals:
        # Collecter les données pour l'ANOVA
        groups = []
        for category in df[var].unique():
            group_data = df[df[var] == category][target]
            groups.append(group_data)

        # Effectuer l'ANOVA
        f_statistic, p_value = stats.f_oneway(*groups)

        # Interpréter les résultats
        if p_value < alpha:
            result = 'Yes'
        else:
            result = 'No'

        # Ajouter les résultats à la liste
        results.append({
            'variable': var,
            'statistique F': f_statistic,
            'p-value': p_value,
            'result': result
        })
```

```
    results_sorted = sorted(results, key=lambda x: x['statistique F'], re
    df_results = pd.DataFrame(results_sorted)

    # Appliquer le style conditionnel pour mettre en évidence les lignes
    def highlight_yes(s):
        return ['background-color: lightgreen' if v == 'Yes' else '' for

    styled_df = df_results.style.apply(highlight_yes, subset=['result'])

    # Afficher les variables dont la p-value est inférieure à 0,05
    significant_vars = df_results[df_results['p-value'] < alpha]['variabl
    if significant_vars:
        print(f"Categoricals with statistical significatif impact on {tar
    else:
        print(f"Aucune des variables testées n'a un effet statistiquement


    return styled_df

styled_results = anova_test(df, categoricals, target='FPA', alpha=0.05)
styled_results
```

```
___Test ANOVA___

Categoricals with statistical significatif impact on FPA (p-value < 0,05) :
CJSITUA, STATUTEXT, METIER, TYPMEN15, ETAMATRI, RABS, SITUA
```

Out[52]:

variable	statistique F	p-value	result
CJSITUA	.	.	Yes
STATUTEXT	.	.	Yes
METIER	.	.	Yes
TYPMEN	.	.	Yes
ETAMATRI	.	.	Yes
RABS	.	.	Yes
SITUA	.	.	Yes
AIDFAM	.	.	No
PACS	.	.	No
TYPOLOG	.	.	No
MER E	.	.	No
PER E	.	.	No

Selection categorcials_

In [53]:
```
# Filtrage des variables catégorielles
categoricals = ['ETAMATRI', 'SITUA', 'CJSITUA', 'RABS', 'STATUTEXT', 'MET

new_columns = numericals + categoricals + target
df = df[new_columns].copy()
```

( ) Multicolinéarité

- VIF < 15 (OK)
- VIF > 15 (NOK)

In [54]:
```
from statsmodels.stats.outliers_influence import variance_inflation_facto

for col in df.columns:
    df[col] = pd.to_numeric(df[col], errors='coerce')  # Convertit en num

X = sm.add_constant(df)  # Ajoute la constante pour estimer l'intercept
vif_data = pd.DataFrame()
vif_data['Feature'] = X.columns
vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(

# Affichage des résultats
print("Variance Inflation Factors (VIF):")
print(vif_data)
```

```
Variance Inflation Factors (VIF):
      Feature        VIF
0       const  96.784175
1       NPERS   1.632954
2     NACTIFS   1.641386
3    REVMENUC   1.074256
4    ETAMATRI   1.009607
5       SITUA   1.060303
6     CJSITUA   1.526133
7        RABS   1.014360
8   STATUTEXT   1.028806
9      METIER   1.006238
10   TYPMEN15   1.647568
11        FPA   1.028506
```

( ) Construction du modèle

- Construction initiale
  - Construisez le modèle de régression logistique en incluant les variables indépendantes sélectionnées.

In [55]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

df[categoricals] = df[categoricals].astype(str)

# Séparer les features et la target
X = df[numericals + categoricals]  # Les variables explicatives
y = df['FPA']                      # La variable cible

# Préparer les transformations pour les données
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),  # Imputer les valeurs m
    ('scaler', StandardScaler())
```

```
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),  # Imputer les
    ('onehot', OneHotEncoder(drop='first'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numericals),
        ('cat', categorical_transformer, categoricals)
    ])

# Créer un pipeline avec le préprocesseur et le modèle
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(class_weight='balanced', max_iter=1
    ])

# Séparer les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Entraîner le modèle
model.fit(X_train, y_train)

# Faire des prédictions et évaluer le modèle
y_pred = model.predict(X_test)

# Rapport de classification
print(classification_report(y_test, y_pred))

# Matrice de confusion
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.74      0.78      3756
           1       0.34      0.47      0.39      1049

    accuracy                           0.68      4805
   macro avg       0.58      0.61      0.59      4805
weighted avg       0.72      0.68      0.70      4805

[[2780  976]
 [ 556  493]]
```

- Précision (Precision) :
  - proportion d'observations positives prédites correctement parmi toutes les observations comme positives.
- Rappel (Recall) :
  - proportion d'observations positives réelles prédites correctement parmi toutes les obse positives réelles.

- F -score : mesure de la précision pondérée par le rappel. C'est la moyenne harmonique d précision et du rappel.
  - Un F -score élevé indique à la fois une bonne précision et un bon rappel.
- Support :
  - nombre réel d'occurrences de chaque classe dans l'échantillon de test.
- Accuracy (Exactitude) :
  - proportion totale de prédictions correctes parmi toutes les prédictions.
- Macro avg :
  - moyenne non pondérée des métriques de chaque classe.
- Weighted avg : moyenne pondérée des métriques de chaque classe, selon le nombre d'éch de chaque classe.
  - Cela donne une meilleure idée des performances globales du modèle compte tenu de l distribution des classes dans l'ensemble de données.
- Matrice de confusion :
  - vrais positifs (en haut à gauche)
  - faux positifs (en haut à droite)
  - faux négatifs (en bas à gauche)
  - vrais négatifs (en bas à droite)

( ) Diagnostic du modèle

- Vérification de la linéarité en logit
  - Utilisez des graphiques de régression partielle pour vérifier que la relation entre chaque indépendante et le logit de la variable dépendante est linéaire.

In [75]:
```
def check_linearity_in_logit(df, numericals, target='FPA'):

    df['REVMENUC'] = np.log(df['REVMENUC'])


    # Nettoyer les données
    df = df.replace([np.inf, -np.inf], np.nan).dropna()


    df['NPERS'] = np.sqrt(df['NPERS'])
    df['NACTIFS'] = np.sqrt(df['NACTIFS'])


    # Ajouter une constante au DataFrame
    df = sm.add_constant(df)

    # Ajuster le modèle de régression logistique
    logit_model = sm.Logit(df[target], df[numericals])
    logit_results = logit_model.fit(disp=0)

    # Calculer les probabilités prédites en logit
```

```
    predicted_logit = logit_results.predict()

    # Tracer les graphiques de dispersion
    fig, axes = plt.subplots(nrows=1, ncols=len(numericals), figsize=(15,

    for i, var in enumerate(numericals):
        # Tracer le graphique de dispersion (variable continue vs logit d
        probplot = ProbPlot(df[var], fit=True)
        axes[i].scatter(probplot.theoretical_quantiles, probplot.sample_q
        axes[i].set_title(f"{var} vs Logit des probabilités prédites")
        axes[i].set_xlabel(f"Theoretical Quantiles")
        axes[i].set_ylabel(f"{var}")
        axes[i].grid(True)

        # Ajouter la droite de régression linéaire
        fit = np.polyfit(probplot.theoretical_quantiles, probplot.sample_
        fit_values = np.polyval(fit, probplot.theoretical_quantiles)
        axes[i].plot(probplot.theoretical_quantiles, fit_values, 'r-', li

        # Calculer le R² pour la linéarité
        r2 = r2_score(probplot.sample_quantiles, fit_values)

        # Afficher le R² sur le graphique
        axes[i].text(0.05, 0.95, f'R² = {r2:.4f}', transform=axes[i].tran
                     fontsize=12, verticalalignment='top', bbox=dict(boxs

    plt.tight_layout()
    plt.show()

# Appeler la fonction pour vérifier la linéarité en logit pour les variab
check_linearity_in_logit(df, numericals, target='FPA')
```

```
/Users/mariusayrault/GitHub/Sorb-Data-Analytics/projet-statistique/.venv/lib/
python3.12/site-packages/pandas/core/arraylike.py:399: RuntimeWarning: invali
value encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[75], line 52
     49     plt.show()
     51 # Appeler la fonction pour vérifier la linéarité en logit pour les
variables continues
---> 52 check_linearity_in_logit(df, numericals, target='FPA')

Cell In[75], line 18, in check_linearity_in_logit(df, numericals, target)
     15 df = sm.add_constant(df)
     17 # Ajuster le modèle de régression logistique
---> 18 logit_model = sm.Logit(df[target], df[numericals])
     19 logit_results = logit_model.fit(disp=0)
     21 # Calculer les probabilités prédites en logit

File ~/GitHub/Sorb-Data-Analytics/projet-statistique/.venv/lib/python3.12/sit
packages/statsmodels/discrete/discrete_model.py:475, in BinaryModel.__init__(
endog, exog, offset, check_rank, **kwargs)
    472 def __init__(self, endog, exog, offset=None, check_rank=True, **kwarg
    473     # unconditional check, requires no extra kwargs added by subclass
    474     self._check_kwargs(kwargs)
--> 475     super().__init__(endog, exog, offset=offset, check_rank=check_ran
    476                      **kwargs)
    477     if not issubclass(self.__class__, MultinomialModel):
    478         if not np.all((self.endog >= 0) & (self.endog <= 1)):

File ~/GitHub/Sorb-Data-Analytics/projet-statistique/.venv/lib/python3.12/sit
packages/statsmodels/discrete/discrete_model.py:185, in DiscreteModel.__init_
, endog, exog, check_rank, **kwargs)
    183 def __init__(self, endog, exog, check_rank=True, **kwargs):
    184     self._check_rank = check_rank
--> 185     super().__init__(endog, exog, **kwargs)
    186     self.raise_on_perfect_prediction = False  # keep for backwards co
    187     self.k_extra = 0

File ~/GitHub/Sorb-Data-Analytics/projet-statistique/.venv/lib/python3.12/sit
packages/statsmodels/base/model.py:270, in LikelihoodModel.__init__(self, end
exog, **kwargs)
    269 def __init__(self, endog, exog=None, **kwargs):
--> 270     super().__init__(endog, exog, **kwargs)
    271     self.initialize()

File ~/GitHub/Sorb-Data-Analytics/projet-statistique/.venv/lib/python3.12/sit
packages/statsmodels/base/model.py:95, in Model.__init__(self, endog, exog,
**kwargs)
     93 missing = kwargs.pop('missing', 'none')
     94 hasconst = kwargs.pop('hasconst', None)
---> 95 self.data = self._handle_data(endog, exog, missing, hasconst,
     96                                **kwargs)
     97 self.k_constant = self.data.k_constant
     98 self.exog = self.data.exog

File ~/GitHub/Sorb-Data-Analytics/projet-statistique/.venv/lib/python3.12/sit
packages/statsmodels/base/model.py:135, in Model._handle_data(self, endog, ex
missing, hasconst, **kwargs)
    134 def _handle_data(self, endog, exog, missing, hasconst, **kwargs):
--> 135     data = handle_data(endog, exog, missing, hasconst, **kwargs)
    136     # kwargs arrays could have changed, easier to just attach here
    137     for key in kwargs:
```

```
File ~/GitHub/Sorb-Data-Analytics/projet-statistique/.venv/lib/python3.12/sit
packages/statsmodels/base/data.py:675, in handle_data(endog, exog, missing,
hasconst, **kwargs)
    672     exog = np.asarray(exog)
    674 klass = handle_data_class_factory(endog, exog)
--> 675 return klass(endog, exog=exog, missing=missing, hasconst=hasconst,
    676              **kwargs)

File ~/GitHub/Sorb-Data-Analytics/projet-statistique/.venv/lib/python3.12/sit
packages/statsmodels/base/data.py:88, in ModelData.__init__(self, endog, exog
missing, hasconst, **kwargs)
     86 self.const_idx = None
     87 self.k_constant = 0
---> 88 self._handle_constant(hasconst)
     89 self._check_integrity()
     90 self._cache = {}

File ~/GitHub/Sorb-Data-Analytics/projet-statistique/.venv/lib/python3.12/sit
packages/statsmodels/base/data.py:132, in ModelData._handle_constant(self,
hasconst)
    129 else:
    130     # detect where the constant is
    131     check_implicit = False
--> 132     exog_max = np.max(self.exog, axis=0)
    133     if not np.isfinite(exog_max).all():
    134         raise MissingDataError('exog contains inf or nans')

File ~/GitHub/Sorb-Data-Analytics/projet-statistique/.venv/lib/python3.12/sit
packages/numpy/_core/fromnumeric.py:2899, in max(a, axis, out, keepdims, init
where)
   2781 @array_function_dispatch(_max_dispatcher)
   2782 @set_module('numpy')
   2783 def max(a, axis=None, out=None, keepdims=np._NoValue, initial=np._NoV
   2784         where=np._NoValue):
   2785     """
   2786     Return the maximum of an array or maximum along an axis.
   2787
   (...)
   2897     5
   2898     """
-> 2899     return _wrapreduction(a, np.maximum, 'max', axis, None, out,
   2900                           keepdims=keepdims, initial=initial, where=w

File ~/GitHub/Sorb-Data-Analytics/projet-statistique/.venv/lib/python3.12/sit
packages/numpy/_core/fromnumeric.py:86, in
_wrapreduction(obj, ufunc, method, axis, dtype, out, **kwargs)
     83         else:
     84             return reduction(axis=axis, out=out, **passkwargs)
---> 86 return ufunc.reduce(obj, axis, dtype, out, **passkwargs)

ValueError: zero-size array to reduction operation maximum which has no ident
```

Le coefficient de détermination (R^2) mesure la proportion de la variance dans la variable dépend est prévisible à partir de la variable indépendante. Dans le contexte de la linéarité en logit pour l régression logistique :

R^2 proche de : Indique une forte linéarité, ce qui est souhaitable. R^2 autour de . à Indique une relation modérément linéaire. Des améliorations peuvent être nécessaires. R^2 inféri

. : Indique une faible linéarité, nécessitant souvent une transformation ou un autre type de traitement.

Transformation log

df['var_log'] = np.log(df['var'])

Transformation puissance

df['var_sqrt'] = np.sqrt(df['var']) df['var_squared'] = df['var']

Transformation inverse

df['var_inv'] = / df['var']

- Indépendance des observations Utilisez des tests comme le test de Durbin-Watson pour dé l'autocorrélation dans les résidus.

( ) Validation du modèle

- Validation croisée
  - Divisez les données en ensembles d'entraînement et de test pour valider le modèle.
  - Utilisez des mesures de performance comme l'accuracy, la précision, le rappel et l'AUC pour évaluer le modèle.

In [67]:
```
def final_model(df, numericals, categoricals, target='FPA', num_groups=10

    df = sm.add_constant(df)


    exog_numericals = df[numericals].values
    exog_categoricals = pd.get_dummies(df[categoricals], drop_first=True)

    scaler = StandardScaler()
    exog_numericals_scaled = scaler.fit_transform(exog_numericals)

    exog = np.concatenate((exog_numericals_scaled, exog_categoricals), ax

    # Ajuster le modèle de régression logistique
    model = sm.Logit(df[target], exog)
    results = model.fit(maxiter=500)

    # Calculer les résidus deviance standardisés
    residuals = results.resid_pearson


    durbin_watson_statistic = durbin_watson(residuals)
    print(f"\n----------------------------------------------------------
```

```python
if durbin_watson_statistic < 1.5:
    print("Les résidus montrent une autocorrélation positive.")
elif durbin_watson_statistic > 2.5:
    print("Les résidus montrent une autocorrélation négative.")
else:
    print("Les résidus ne montrent pas d'autocorrélation significativ



# TEST DE HOSMER-LEMESHOW

y_true = df[target].values  # Utiliser le nom de la variable cible da
y_prob = results.predict(exog)  # Utiliser le modèle pour prédire les

# Vérification des dimensions des données
if len(y_true) != len(y_prob):
    raise ValueError("Les dimensions de y_true et y_prob doivent être

# Créer les groupes en fonction des probabilités prédites
y_true = np.asarray(y_true)
y_prob = np.asarray(y_prob)
deciles = np.percentile(y_prob, np.arange(0, 100, 100/num_groups))
groups = np.digitize(y_prob, deciles)

# Initialisation des tableaux pour les fréquences observées et attend
obs_freq = np.zeros(num_groups)
exp_freq = np.zeros(num_groups)

# Calcul des fréquences observées et attendues pour chaque groupe
for i in range(num_groups):
    obs_freq[i] = np.sum((groups == (i + 1)) * y_true)
    exp_freq[i] = np.sum(groups == (i + 1)) * np.mean(y_true)

# Calcul de la statistique de test de Hosmer-Lemeshow
chi2_statistic = np.sum((obs_freq - exp_freq)**2 / exp_freq)

# Degré de liberté est (num_groups - 2) car il y a num_groups - 1 gro
df_hl = num_groups - 2

# Calcul de la valeur p
p_value_hl = 1 - chi2.cdf(chi2_statistic, df_hl)

print(f"\n----------------------------------------------------------------
print(f"Valeur p : {p_value_hl:.4f}")

# Interpréter les résultats du test de Hosmer-Lemeshow
if p_value_hl < 0.05:
    print("Le modèle ne correspond pas bien aux données observées (re
else:
    print("Le modèle correspond bien aux données observées (absence d

# VALIDATION DU MODÈLE

# Séparer les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(exog, df[target],

# Perform cross-validation with LogisticRegression from scikit-learn
```

```
    logreg = LogisticRegression()

    # Fit the model on training data
    logreg.fit(X_train, y_train)

    # Perform cross-validation
    accuracy_scores = cross_val_score(logreg, X_train, y_train, cv=5, sco
    precision_scores = cross_val_score(logreg, X_train, y_train, cv=5, sc
    recall_scores = cross_val_score(logreg, X_train, y_train, cv=5, scori
    roc_auc_scores = cross_val_score(logreg, X_train, y_train, cv=5, scor

    # Calculate mean scores
    mean_accuracy = accuracy_scores.mean()
    mean_precision = precision_scores.mean()
    mean_recall = recall_scores.mean()
    mean_roc_auc = roc_auc_scores.mean()

    # Print the results
    print("\n----------------------------------")
    print("Validation du modèle avec cross-validation :")
    print(f"Accuracy : {mean_accuracy:.4f}")
    print(f"Precision : {mean_precision:.4f}")
    print(f"Recall : {mean_recall:.4f}")
    print(f"AUC-ROC : {mean_roc_auc:.4f}")

    # Return or print results as needed
    results = {
        'Accuracy': mean_accuracy,
        'Precision': mean_precision,
        'Recall': mean_recall,
        'AUC-ROC': mean_roc_auc
    }

    return y_true, y_prob

# Appeler la fonction pour ajuster le modèle final
y_true, y_prob = final_model(df, numericals, categoricals, target='FPA',
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.480473
         Iterations: 500

----------------------------------------------------------------------------
Statistique de Durbin-Watson : 1.8541
Les résidus ne montrent pas d'autocorrélation significative.

----------------------------------------------------------------------------
Statistique du test de Hosmer-Lemeshow : 1289.7648
Valeur p : 0.0000
Le modèle ne correspond pas bien aux données observées (rejet de l'hypothèse
nulle).
```

```
/Users/mariusayrault/GitHub/Sorb-Data-Analytics/projet-statistique/.venv/lib/
python3.12/site-packages/statsmodels/base/model.py:607: ConvergenceWarning: M
Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

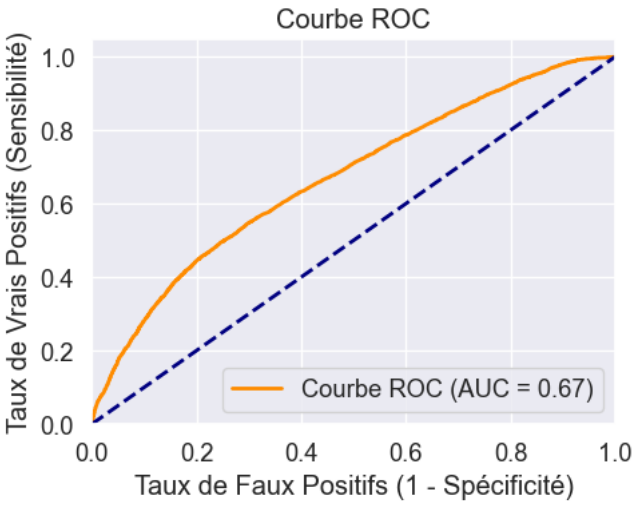
```
---------------------------------
Validation du modèle avec cross-validation :
Accuracy : 0.7937
Precision : 0.5869
Recall : 0.0563
AUC-ROC : 0.6669
```

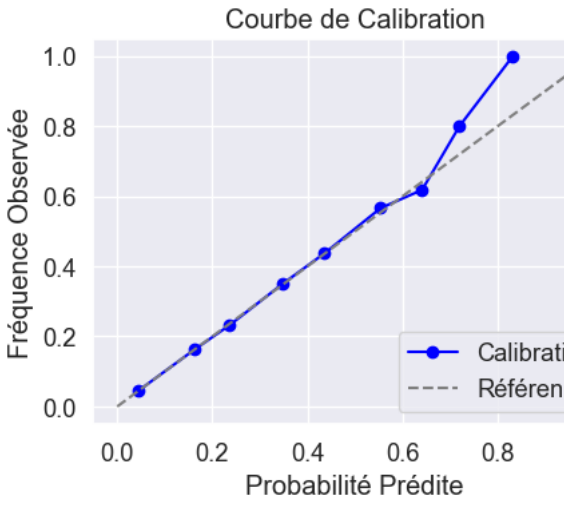
In [68]:

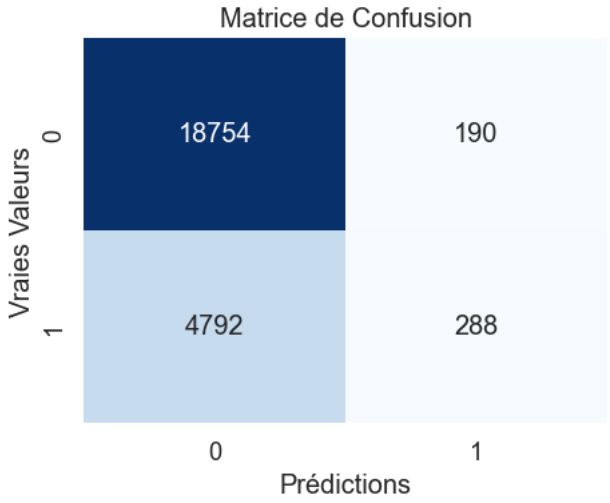
```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

def plot_roc_curve(ax, y_true, y_prob):
    fpr, tpr, thresholds = roc_curve(y_true, y_prob)
    roc_auc = auc(fpr, tpr)
    ax.plot(fpr, tpr, color='darkorange', lw=2, label=f'Courbe ROC (AUC =
    ax.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.set_xlabel('Taux de Faux Positifs (1 - Spécificité)')
    ax.set_ylabel('Taux de Vrais Positifs (Sensibilité)')
    ax.set_title('Courbe ROC')
    ax.legend(loc="lower right")



def plot_calibration_curve(ax, y_true, y_prob):
    prob_true, prob_pred = calibration_curve(y_true, y_prob, n_bins=10)
    ax.plot(prob_pred, prob_true, marker='o', color='blue', label='Calibr
    ax.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Référenc
    ax.set_xlabel('Probabilité Prédite')
    ax.set_ylabel('Fréquence Observée')
    ax.set_title('Courbe de Calibration')
    ax.legend(loc="lower right")


def plot_confusion_matrix(ax, y_true, y_prob):
    y_pred = (y_prob > 0.5).astype(int)
    cm = confusion_matrix(y_true, y_pred)

    cm_percent = cm / cm.sum() * 100

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, ax=ax)
    ax.set_xlabel('Prédictions')
    ax.set_ylabel('Vraies Valeurs')
    ax.set_title('Matrice de Confusion')

    # Interprétation spécifique à l'objectif
    print(f"\nInterprétation :")
    print(f"[%] de femmes correctement identifiées comme princip_app_ress
    print(f"[%] de femmes incorrectement identifiées comme princip_app_re
    print(f"[%] de femmes correctement identifiées comme n'étant pas les
    print(f"[%] de femmes incorrectement identifiées comme n'étant pas le

def plot_precision_recall_curve(ax, y_true, y_prob):
    precision, recall, thresholds = precision_recall_curve(y_true, y_prob
    ax.plot(recall, precision, color='blue', lw=2, label='Courbe de Préci
    ax.set_xlabel('Rappel (Sensibilité)')
    ax.set_ylabel('Précision')
    ax.set_title('Courbe de Précision-Rappel')
    ax.legend(loc="lower left")
    ax.set_xlim([0.0, 1.0])
```

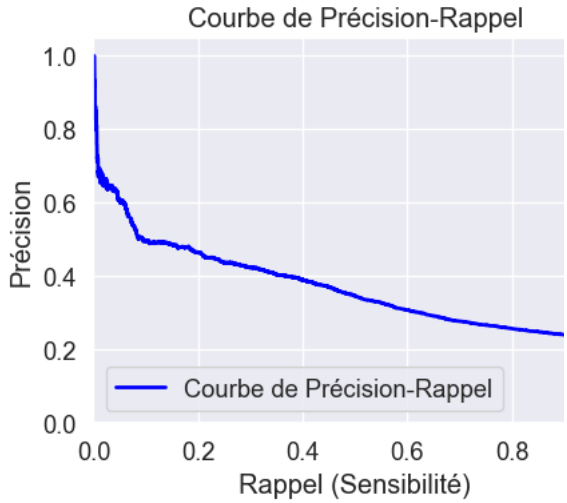
```
    ax.set_ylim([0.0, 1.05])


# Créer la figure et les sous-graphiques
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(10, 8))

# Premier sous-plot : plot_roc_curve
plot_roc_curve(axs[0, 0], y_true, y_prob)

# Deuxième sous-plot : plot_calibration_curve
plot_calibration_curve(axs[0, 1], y_true, y_prob)

# Troisième sous-plot : plot_confusion_matrix
plot_confusion_matrix(axs[1, 0], y_true, y_prob)

# Quatrième sous-plot : plot_precision_recall_curve
plot_precision_recall_curve(axs[1, 1], y_true, y_prob)

# Ajuster automatiquement les paramètres de la figure pour éviter les che
plt.tight_layout()
plt.show()
```

```
Interprétation :
[%] de femmes correctement identifiées comme princip_app_ressources (vrais
positifs) :                    1.20 %
[%] de femmes incorrectement identifiées comme princip_app_ressources (faux
positifs) :                   0.79 %
[%] de femmes correctement identifiées comme n'étant pas les princip_app_ress
(vrais négatifs) :    78.06 %
[%] de femmes incorrectement identifiées comme n'étant pas les
princip_app_ressources (faux négatifs) :   19.95 %
```









Export Notebook.pdf

In [69]:
```python
import nbformat
from nbconvert import HTMLExporter
from weasyprint import HTML
import sys

notebook_path = 'Notebook_regression_logistique.ipynb'
output_pdf_path = './data/Notebook.pdf'

def convert_notebook_to_pdf(notebook_path, output_pdf_path):
    # Lire le fichier notebook
    with open(notebook_path, 'r', encoding='utf-8') as f:
        notebook = nbformat.read(f, as_version=4)

    # Convertir le notebook en HTML
    html_exporter = HTMLExporter()
    html_data, _ = html_exporter.from_notebook_node(notebook)

    # Sauvegarder le HTML dans un fichier temporaire
    temp_html_path = notebook_path.replace('.ipynb', '.html')
    with open(temp_html_path, 'w', encoding='utf-8') as f:
        f.write(html_data)

    # Convertir le fichier HTML en PDF
    HTML(temp_html_path).write_pdf(output_pdf_path)

    print(f"Conversion terminée : {output_pdf_path}")
```

```
convert_notebook_to_pdf(notebook_path, output_pdf_path)
```

```
1 extra bytes in post.stringData array
'created' timestamp seems very low; regarding as unix timestamp
1 extra bytes in post.stringData array
Conversion terminée : ./data/Notebook.pdf
```