oversampling-undersampling-smote

May 15, 2024

1 Importation des packages

[25]:
```
# Importer les modules nécessaires
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

2 Chargement des données

Les données concernent des campagnes de marketing direct (appels téléphoniques) d'une institution bancaire portugaise.

[2]:
```
bankdata=pd.read_csv("bank_cleaned.csv",sep=",",index_col=0)
```

[3]:
```
bankdata.head()
```

[3]:

	age	job	marital	education	default	balance	housing	loan	day	month	duration	campaign	pdays	previous	poutcome	response	response_binary
0	58	management	married	tertiary	no	2143	yes	no	5	may	4.35	1	-1	0	unknown	no	0
1	44	technician	single	secondary	no	29	yes	no	5	may	2.52	1	-1	0	unknown	no	0
2	33	entrepreneur	married	secondary	no	2	yes	yes	5	may	1.27	1	-1	0	unknown	no	0
5	35	management	married	tertiary	no	231	yes	no	5	may	2.32	1	-1	0	unknown	no	0
6	28	management	single	tertiary	no	447	yes	yes	5	may	3.62	1	-1	0	unknown	no	0

```
5                0
6                0
```

[ ]:

2.1 Description de la base de données “bank_cleaned.csv”

La base de données “bank_cleaned.csv” contient des données sur les campagnes de marketing direct (appels téléphoniques) d’une institution bancaire portugaise. Les données ont été nettoyées et préparées pour l’analyse.

Les caractéristiques (variables explicatives) enregistrées pour chaque client sont les suivantes :

-age : l’âge du client (variable numérique)

-job : la profession du client (variable catégorielle)

-marital : l’état matrimonial du client (variable catégorielle)

-education : le niveau d’éducation du client (variable catégorielle)

-default : indique si le client a un crédit en défaut ou non (variable catégorielle)

-balance : le solde du compte du client (variable numérique)

-housing : indique si le client a un prêt immobilier ou non (variable catégorielle)

-loan : indique si le client a un prêt personnel ou non (variable catégorielle)

-day : le jour du mois de la dernière communication avec le client (variable numérique)

-month : le mois de la dernière communication avec le client (variable catégorielle)

-duration : la durée de la dernière communication avec le client, en secondes (variable numérique)

-campaign : le nombre de contacts effectués au cours de cette campagne pour ce client (variable numérique)

-pdays : le nombre de jours écoulés depuis le dernier contact avec le client lors d’une campagne précédente (variable numérique ; 999 signifie que le client n’a pas été contacté précédemment)

-previous : le nombre de contacts effectués avant cette campagne pour ce client (variable numérique)

-poutcome : le résultat de la précédente campagne marketing (variable catégorielle)

-response : la réponse du client à la dernière campagne marketing (variable catégorielle)

-response_binary : la réponse du client à la dernière campagne marketing, encodée en binaire (0 = pas intéressé, 1 = intéressé) (variable numérique)

La variable cible est :

- response_binary : a-t-il souscrit un dépôt à terme ? (variable catégorielle)

Les données ont été préparées pour l’analyse en remplaçant les valeurs manquantes par des valeurs médianes ou moyennes, en convertissant les variables catégorielles en variables binaires, et en supprimant les variables inutiles ou redondantes.

La base de données contient 4521 entrées (lignes) et 9 caractéristiques (colonnes).

[4]:
```
# Renommer les colonnes
bankdata.rename(columns={
    'age': 'age',
    'job': 'profession',
    'marital': 'situation_familiale',
    'education': 'niveau_etudes',
    'default': 'defaut_credit',
    'balance': 'solde_bancaire',
    'housing': 'pret_immobilier',
    'loan': 'pret_personnel',
    'day': 'jour_du_mois',
    'month': 'mois',
    'duration': 'duree_appel',
    'campaign': 'nb_appels',
    'pdays': 'nb_jours_depuis_dernier_appel',
    'previous': 'nb_appels_precedents',
    'poutcome': 'resultat_campagne_precedente',
    'response': 'reponse_campagne_actuelle',
    'response_binary': 'reponse_campagne_actuelle_binaire'
}, inplace=True)

# Afficher les noms de colonnes mis à jour
print(bankdata.columns)
```

```
Index(['age', 'profession', 'situation_familiale', 'niveau_etudes',
       'defaut_credit', 'solde_bancaire', 'pret_immobilier', 'pret_personnel',
       'jour_du_mois', 'mois', 'duree_appel', 'nb_appels',
       'nb_jours_depuis_dernier_appel', 'nb_appels_precedents',
       'resultat_campagne_precedente', 'reponse_campagne_actuelle',
       'reponse_campagne_actuelle_binaire'],
      dtype='object')
```

2.2 Qualité des données

[5]:
```
bankdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 40841 entries, 0 to 45209
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   age                             40841 non-null  int64
 1   profession                      40841 non-null  object
 2   situation_familiale             40841 non-null  object
 3   niveau_etudes                   40841 non-null  object
 4   defaut_credit                   40841 non-null  object
```

```
 5   solde_bancaire                   40841 non-null  int64
 6   pret_immobilier                  40841 non-null  object
 7   pret_personnel                   40841 non-null  object
 8   jour_du_mois                     40841 non-null  int64
 9   mois                             40841 non-null  object
 10  duree_appel                      40841 non-null  float64
 11  nb_appels                        40841 non-null  int64
 12  nb_jours_depuis_dernier_appel    40841 non-null  int64
 13  nb_appels_precedents             40841 non-null  int64
 14  resultat_campagne_precedente     40841 non-null  object
 15  reponse_campagne_actuelle        40841 non-null  object
 16  reponse_campagne_actuelle_binaire  40841 non-null  int64
dtypes: float64(1), int64(7), object(9)
memory usage: 5.6+ MB
```

[6]:
```
bankdata.describe(include="all")
```

[6]:

	age	profession	situation_familiale	niveau_etudes	\
count	40841.000000	40841	40841	40841	
unique	NaN	12	3	3	
top	NaN	blue-collar	married	secondary	
freq	NaN	8805	24641	21933	
mean	40.790676	NaN	NaN	NaN	
std	10.475473	NaN	NaN	NaN	
min	18.000000	NaN	NaN	NaN	
25%	33.000000	NaN	NaN	NaN	
50%	39.000000	NaN	NaN	NaN	
75%	48.000000	NaN	NaN	NaN	
max	95.000000	NaN	NaN	NaN	

	defaut_credit	solde_bancaire	pret_immobilier	pret_personnel	\
count	40841	40841.000000	40841	40841	
unique	2	NaN	2	2	
top	no	NaN	yes	no	
freq	40078	NaN	22820	34042	
mean	NaN	1073.981807	NaN	NaN	
std	NaN	1712.556186	NaN	NaN	
min	NaN	-6847.000000	NaN	NaN	
25%	NaN	64.000000	NaN	NaN	
50%	NaN	421.000000	NaN	NaN	
75%	NaN	1333.000000	NaN	NaN	
max	NaN	10443.000000	NaN	NaN	

	jour_du_mois	mois	duree_appel	nb_appels	\
count	40841.000000	40841	40841.000000	40841.000000	
unique	NaN	12	NaN	NaN	
top	NaN	may	NaN	NaN	

freq	NaN	12496	NaN	NaN
mean	15.863666	NaN	4.308949	2.774149
std	8.313608	NaN	4.305823	3.104177
min	1.000000	NaN	0.100000	1.000000
25%	8.000000	NaN	1.730000	1.000000
50%	16.000000	NaN	3.000000	2.000000
75%	21.000000	NaN	5.300000	3.000000
max	31.000000	NaN	81.970000	58.000000

	nb_jours_depuis_dernier_appel	nb_appels_precedents	\
count	40841.000000	40841.000000	
unique	NaN	NaN	
top	NaN	NaN	
freq	NaN	NaN	
mean	32.248304	0.436791	
std	90.738402	1.572342	
min	-1.000000	0.000000	
25%	-1.000000	0.000000	
50%	-1.000000	0.000000	
75%	-1.000000	0.000000	
max	871.000000	55.000000	

	resultat_campagne_precedente	reponse_campagne_actuelle	\
count	40841	40841	
unique	3	2	
top	unknown	no	
freq	34802	36202	
mean	NaN	NaN	
std	NaN	NaN	
min	NaN	NaN	
25%	NaN	NaN	
50%	NaN	NaN	
75%	NaN	NaN	
max	NaN	NaN	

	reponse_campagne_actuelle_binaire
count	40841.000000
unique	NaN
top	NaN
freq	NaN
mean	0.113587
std	0.317313
min	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
max	1.000000

[7]:
```
bankdata.isnull().sum()
```

[7]:
```
age                                0
profession                         0
situation_familiale                0
niveau_etudes                      0
defaut_credit                      0
solde_bancaire                     0
pret_immobilier                    0
pret_personnel                     0
jour_du_mois                       0
mois                               0
duree_appel                        0
nb_appels                          0
nb_jours_depuis_dernier_appel      0
nb_appels_precedents               0
resultat_campagne_precedente       0
reponse_campagne_actuelle          0
reponse_campagne_actuelle_binaire  0
dtype: int64
```

3 Descripition de la base de données

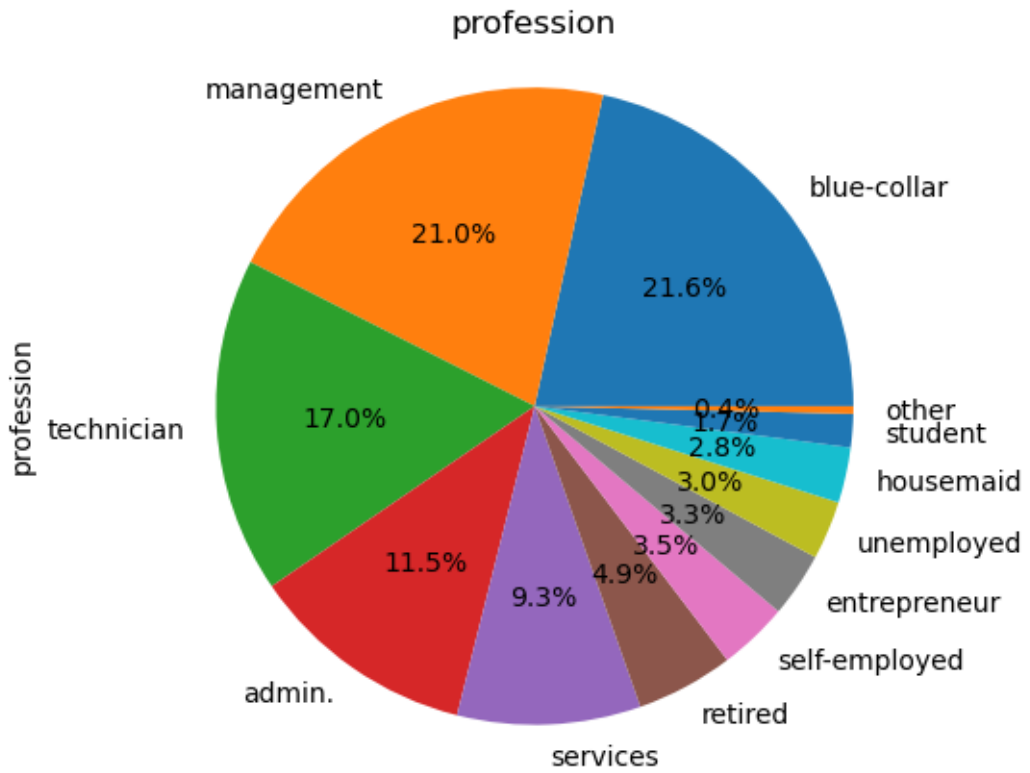
[8]:
```
# Créer une nouvelle version de la base de données sans la variable "response"
bankdata_new = bankdata.drop(columns=['reponse_campagne_actuelle_binaire'])
bankdata_new.head()
```

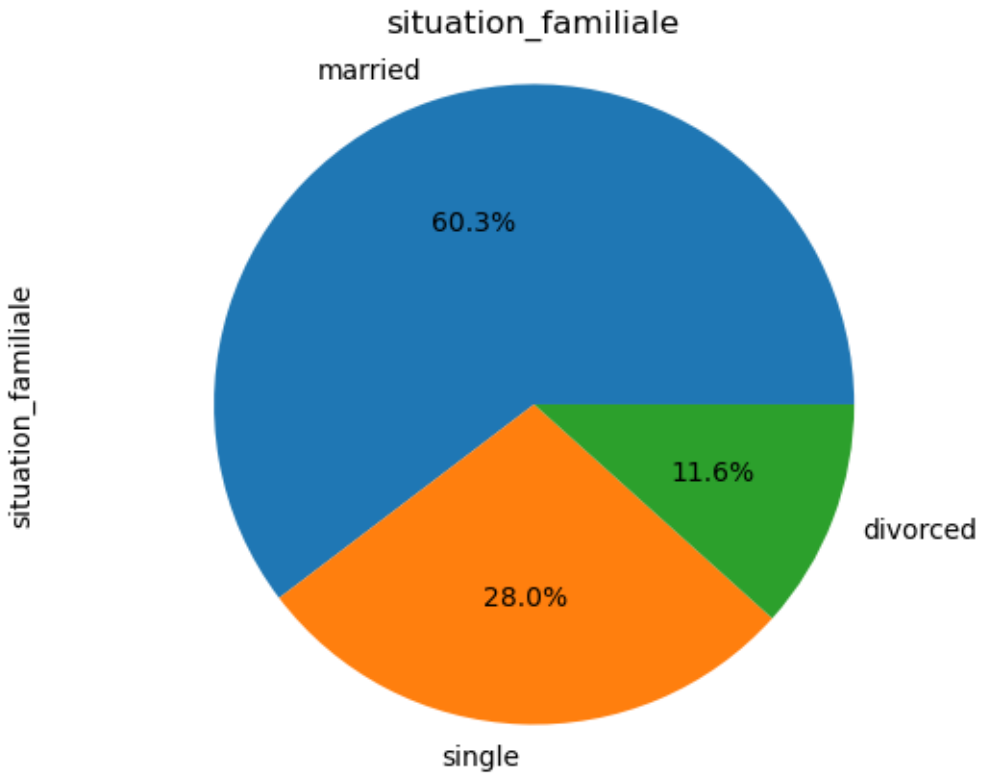
[8]:

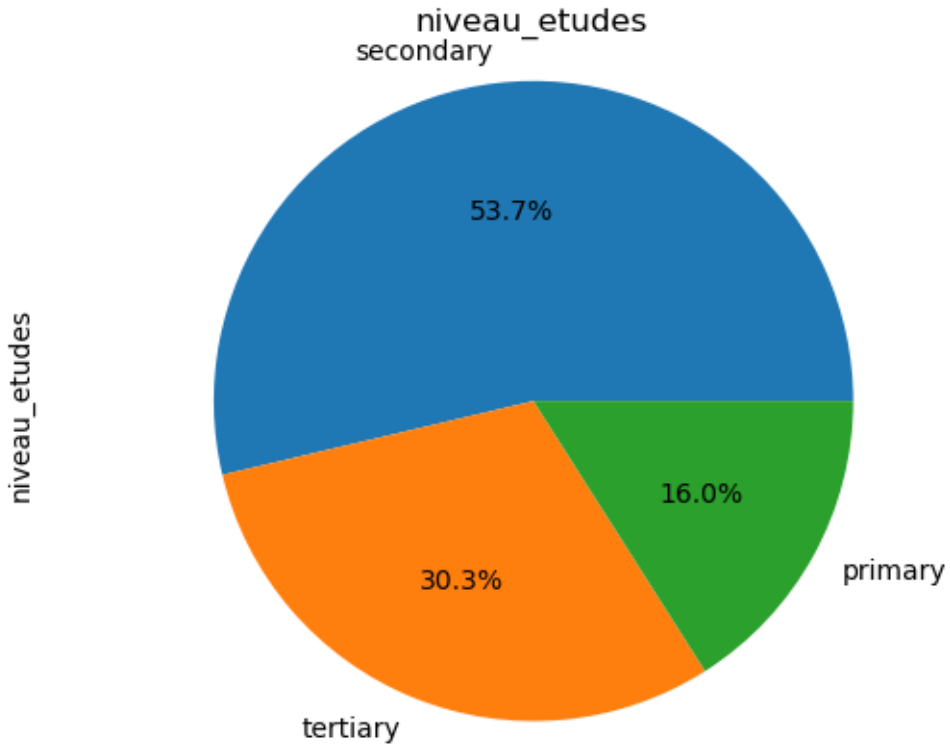
	age	profession	situation_familiale	niveau_etudes	defaut_credit	solde_bancaire	pret_immobilier	pret_personnel	jour_du_mois	mois	duree_appel	nb_appels	nb_jours_depuis_dernier_appel	nb_appels_precedents	resultat_campagne_precedente	reponse_campagne_actuelle
0	58	management	married	tertiary	no	2143	yes	no	5	may	4.35	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	5	may	2.52	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	5	may	1.27	1	-1	0	unknown	no
5	35	management	married	tertiary	no	231	yes	no	5	may	2.32	1	-1	0	unknown	no
6	28	management	single	tertiary	no	447	yes	yes	5	may						

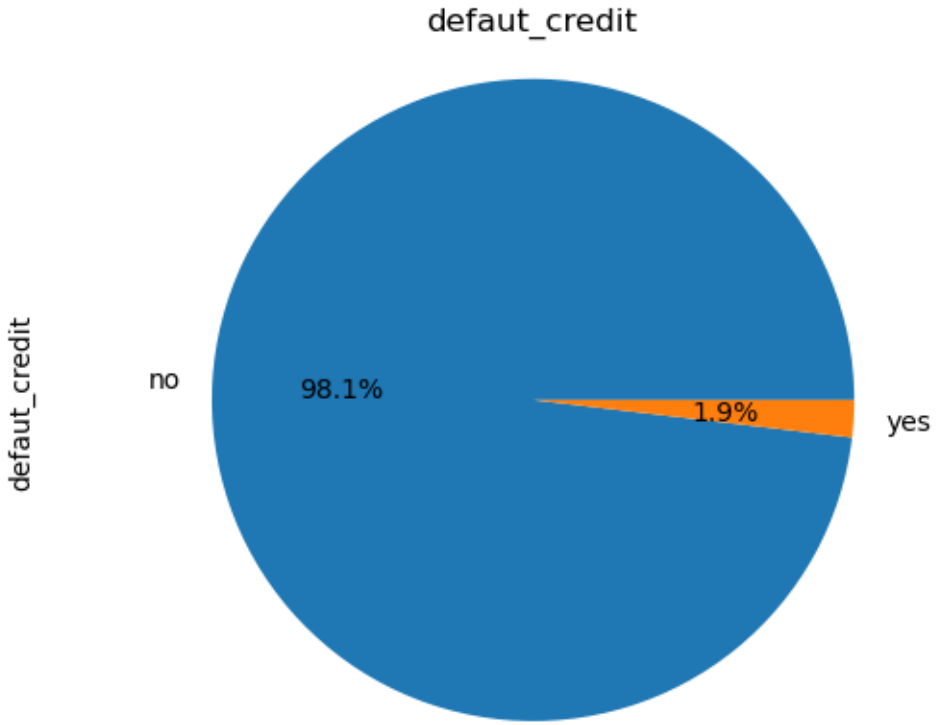
```
6         3.62          1                                -1

   nb_appels_precedents resultat_campagne_precedente reponse_campagne_actuelle
0                     0                      unknown                        no
1                     0                      unknown                        no
2                     0                      unknown                        no
5                     0                      unknown                        no
6                     0                      unknown                        no
```

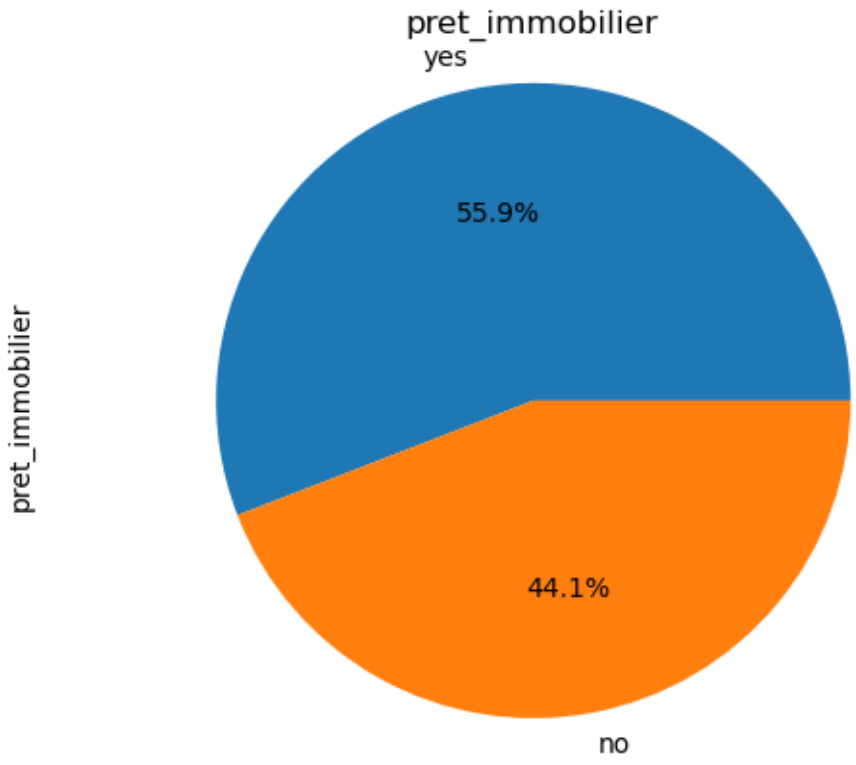
[9]:
```
# Sélectionner les variables catégorielles
cat_vars = ['profession', 'situation_familiale', 'niveau_etudes',␣
 ↪'defaut_credit', 'pret_immobilier', 'pret_personnel',
             'mois', 'resultat_campagne_precedente',␣
 ↪'reponse_campagne_actuelle', 'reponse_campagne_actuelle_binaire']


# Générer un pie plot pour chaque variable catégorielle
for var in cat_vars:
    bankdata[var].value_counts().plot(kind='pie', autopct='%1.1f%%')
    plt.title(var)
    plt.axis('equal')
    plt.show()
```

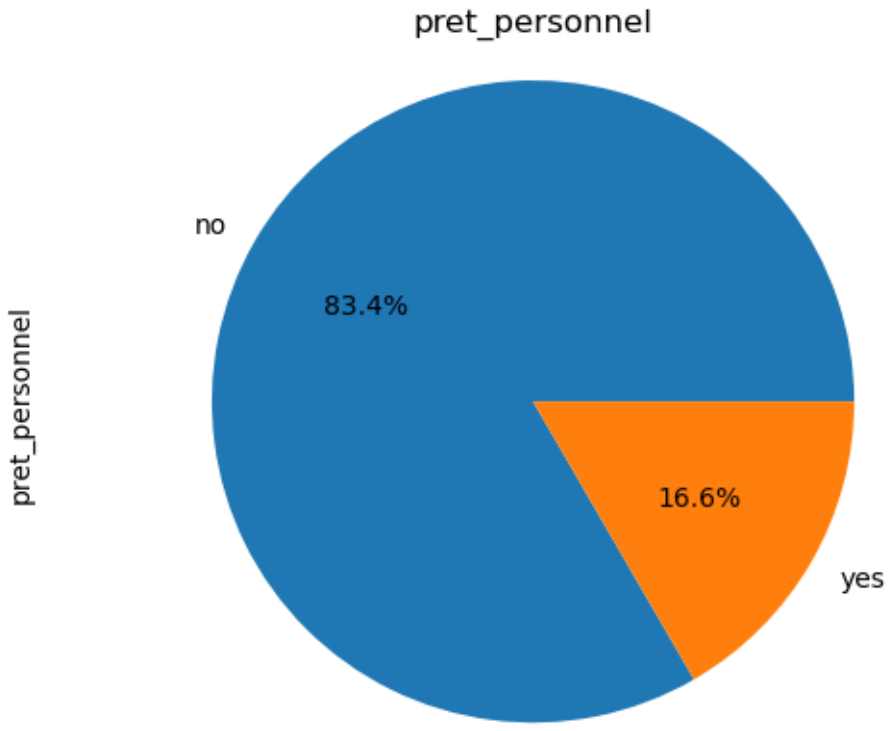


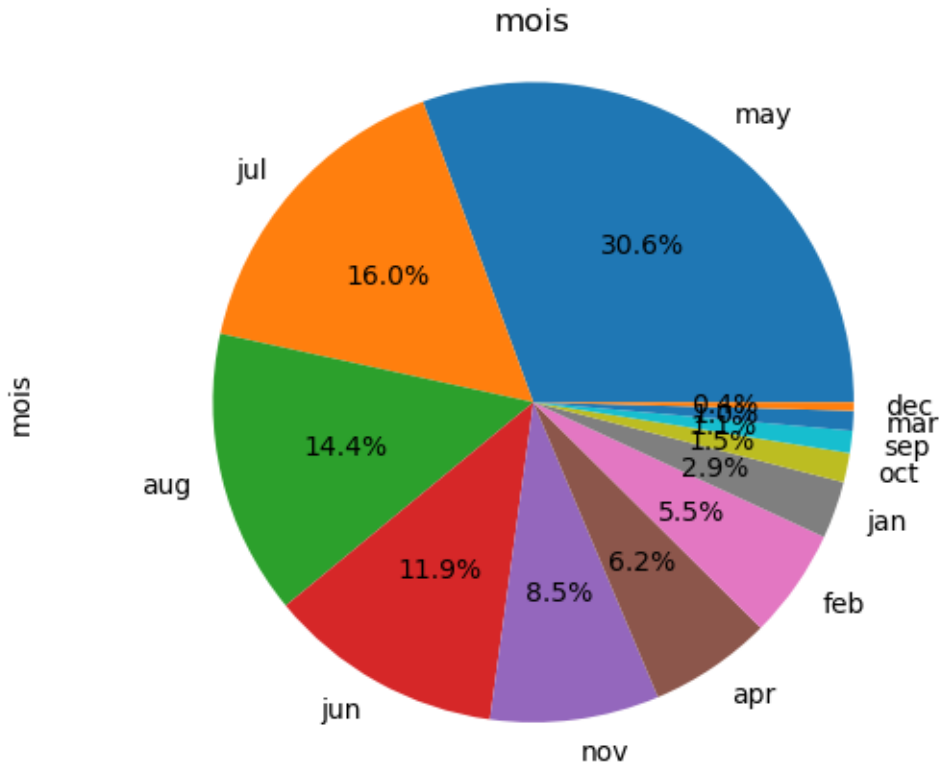

situation_familiale
married
60.3%
situation_familiale
11.6%
divorced
28.0%
single

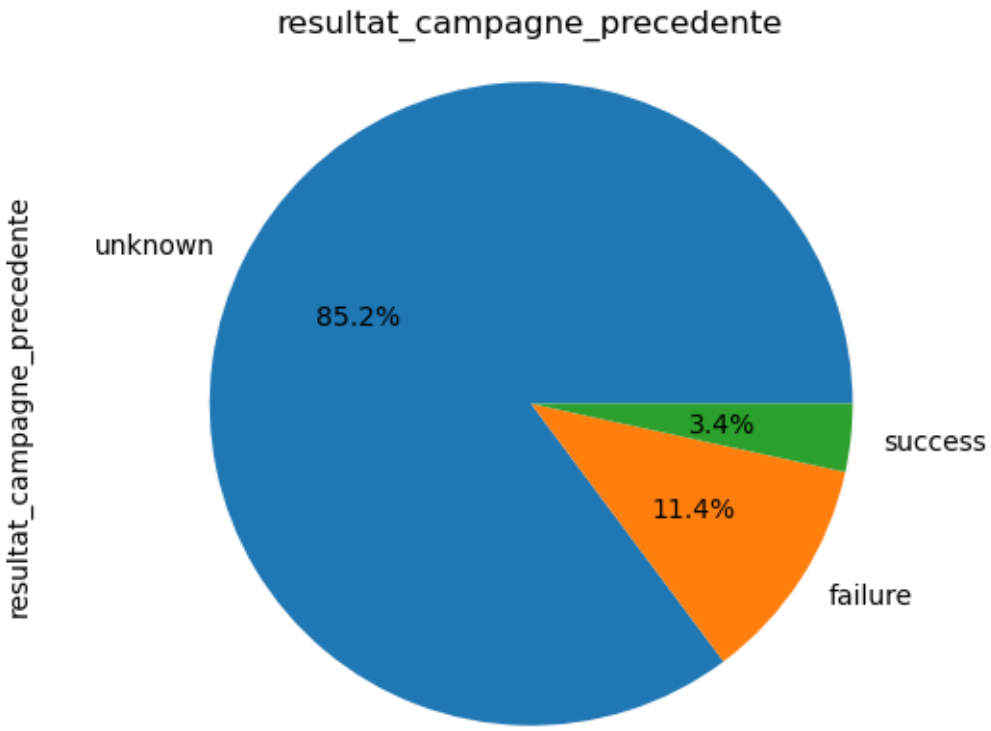

niveau_etudes
secondary
53.7%
niveau_etudes
16.0%
primary
30.3%
tertiary

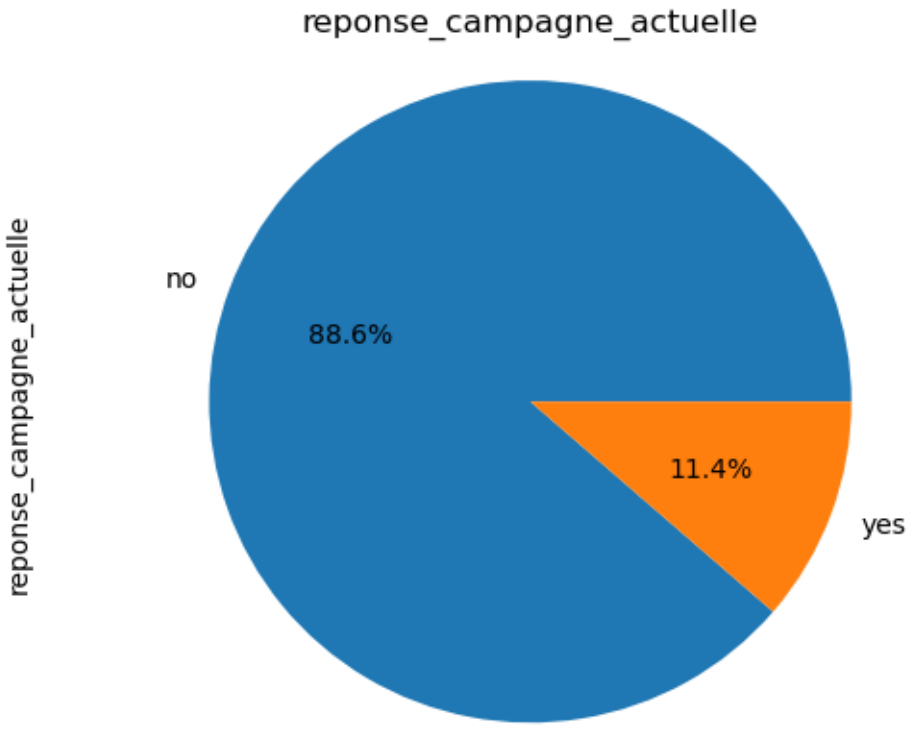

defaut_credit
default_credit
no
98.1%
1.9%
yes

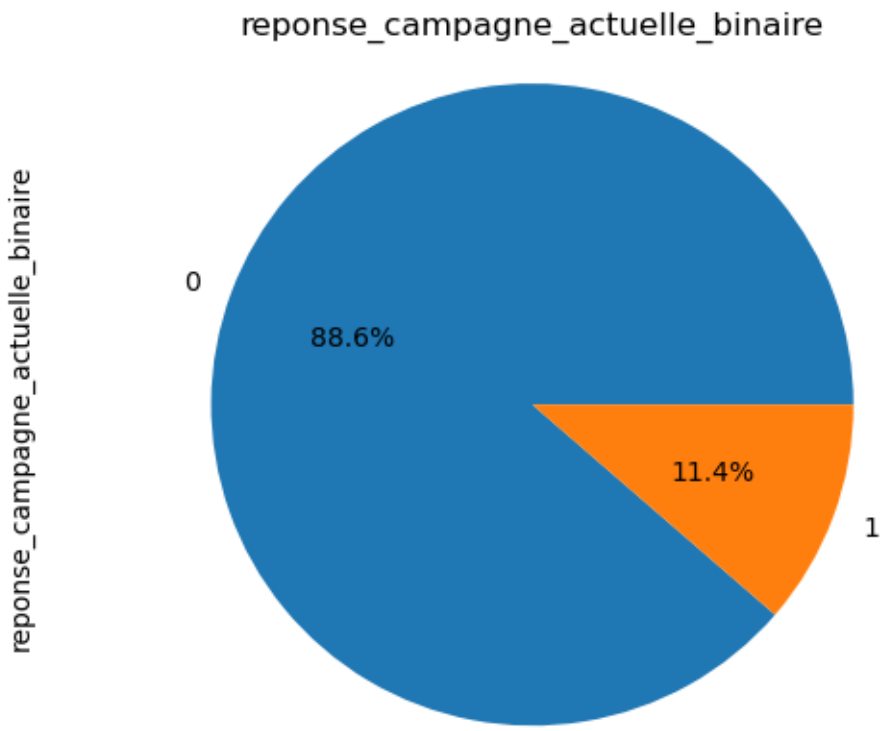

pret_immobilier
yes
55.9%
pret_immobilier
44.1%
no


pret_personnel
no
83.4%
pret_personnel
16.6%
yes

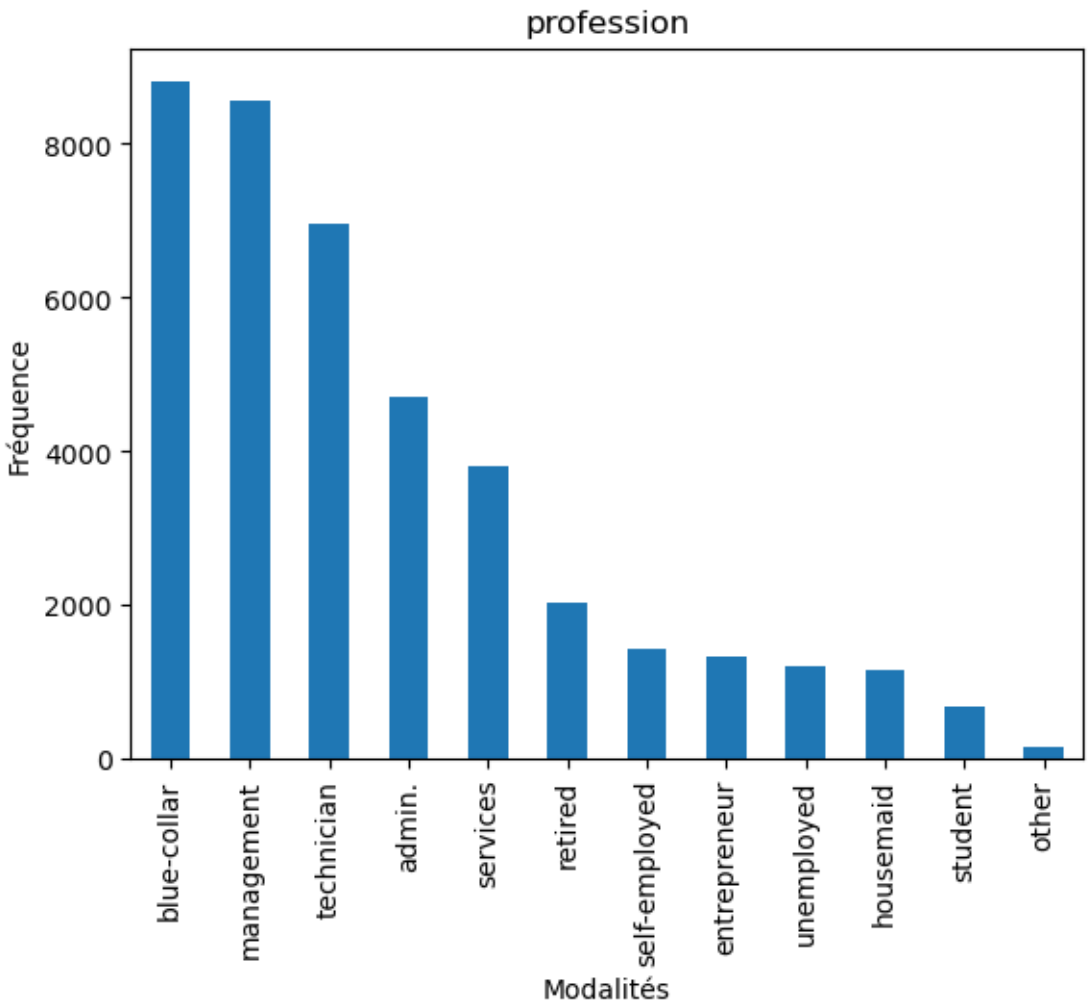

mois
may
jul
aug
jun
nov
apr
feb
jan
oct
sep
mar
dec
30.6%
16.0%
14.4%
11.9%
8.5%
6.2%
5.5%
2.9%
1.5%
mois

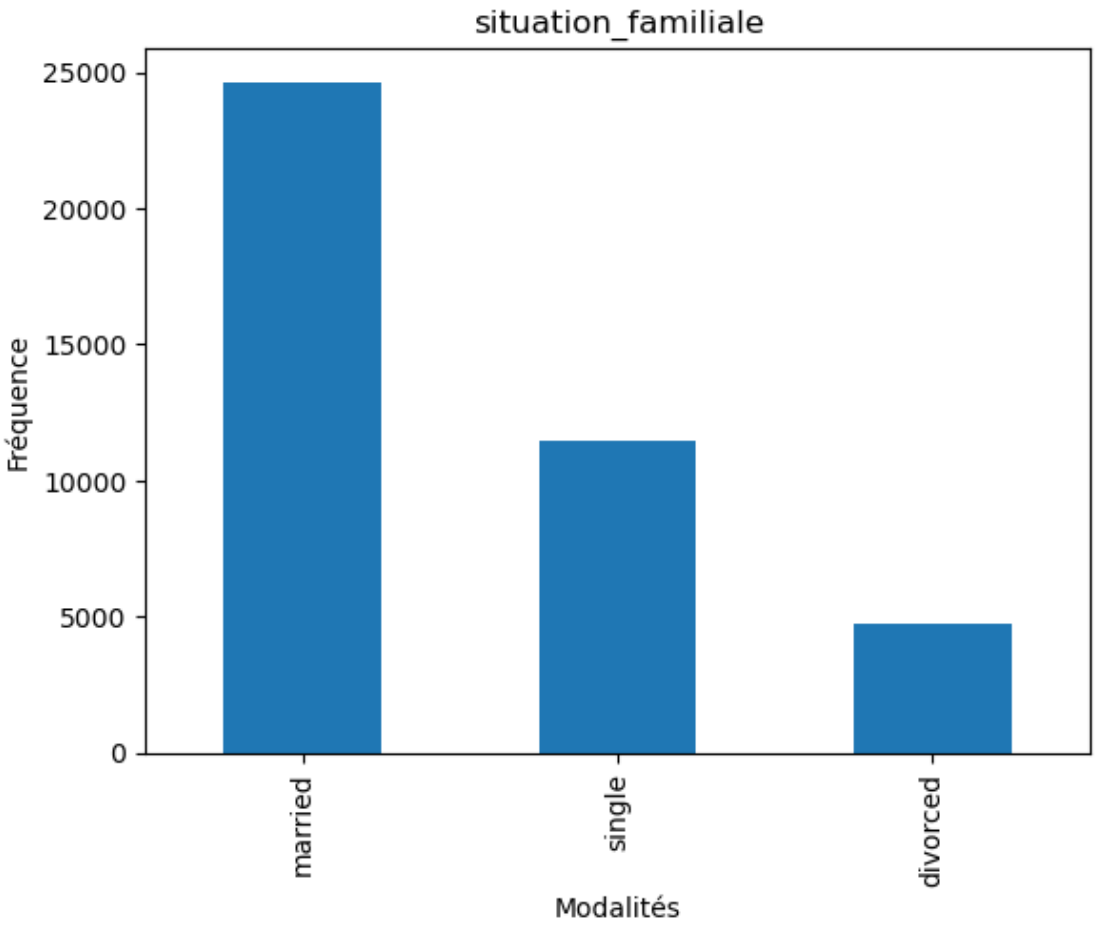

resultat_campagne_precedente
resultat_campagne_precedente
unknown
85.2%
3.4%
success
11.4%
failure

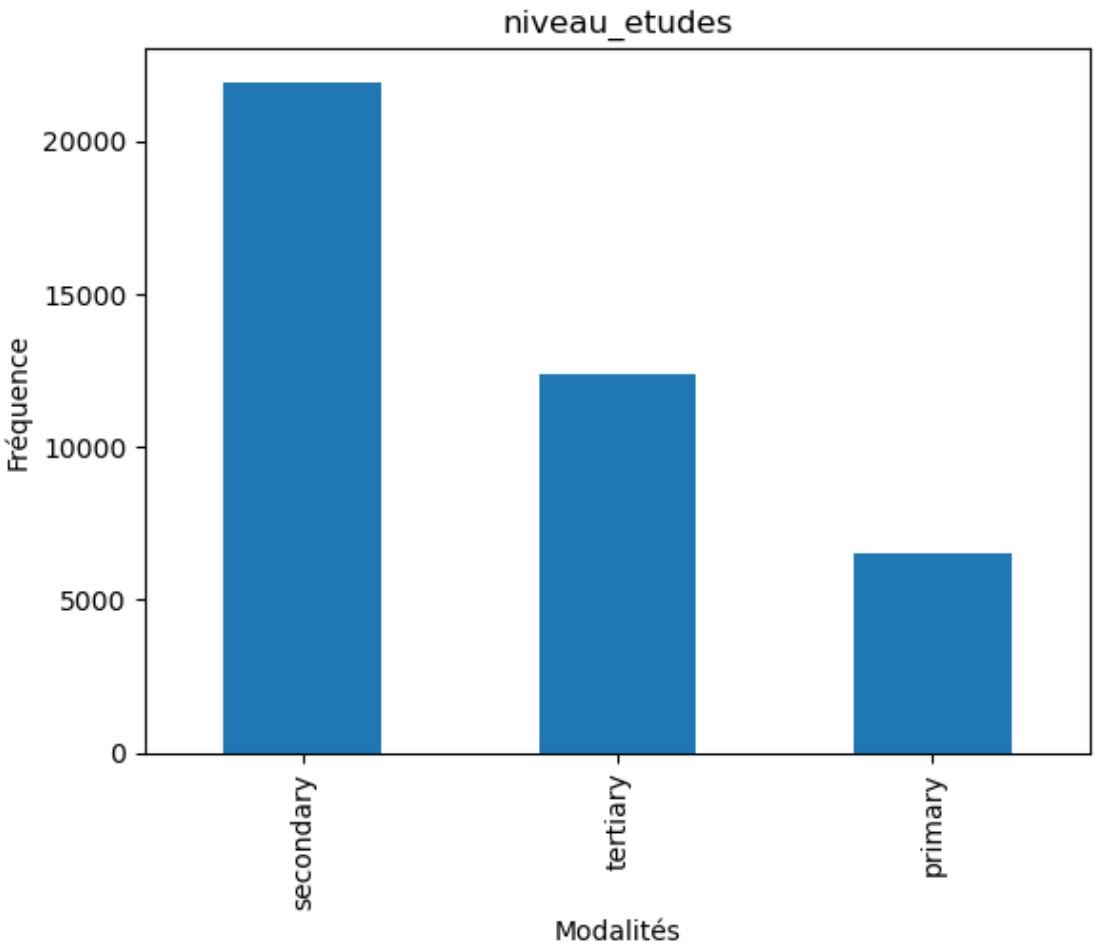

reponse_campagne_actuelle
reponse_campagne_actuelle
no
88.6%
11.4%
yes

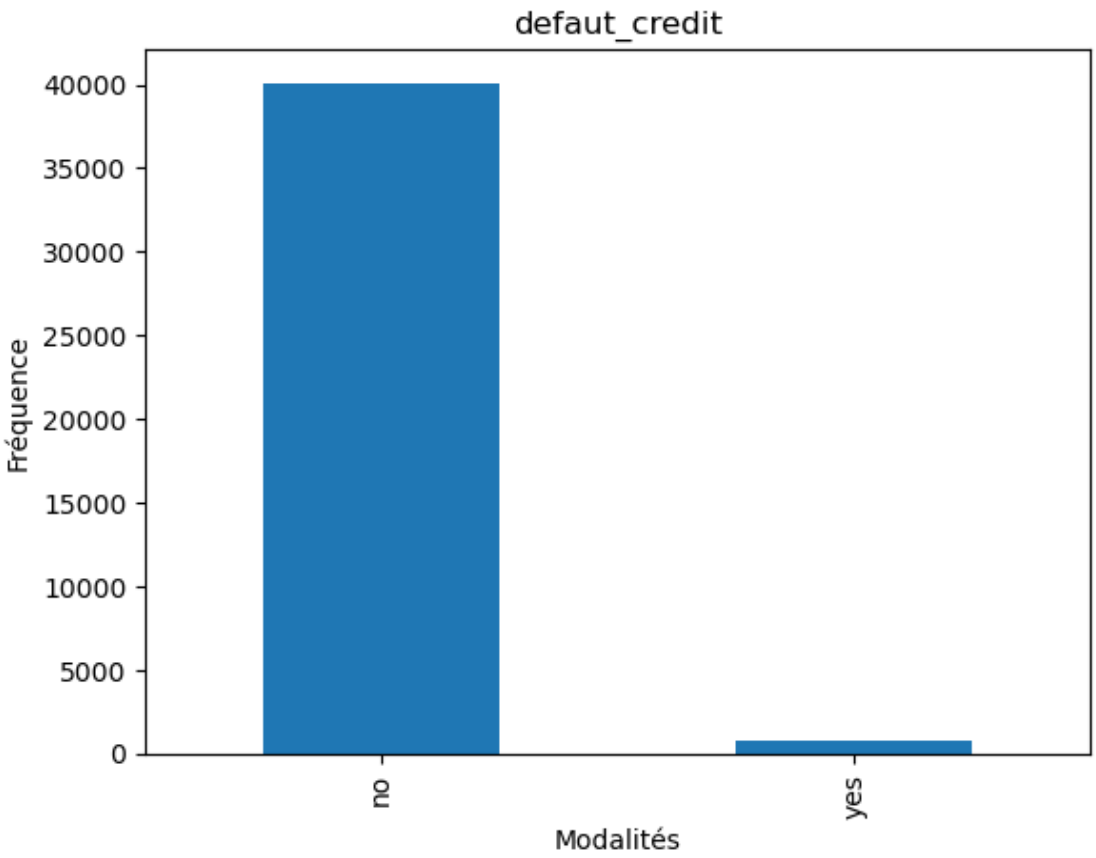


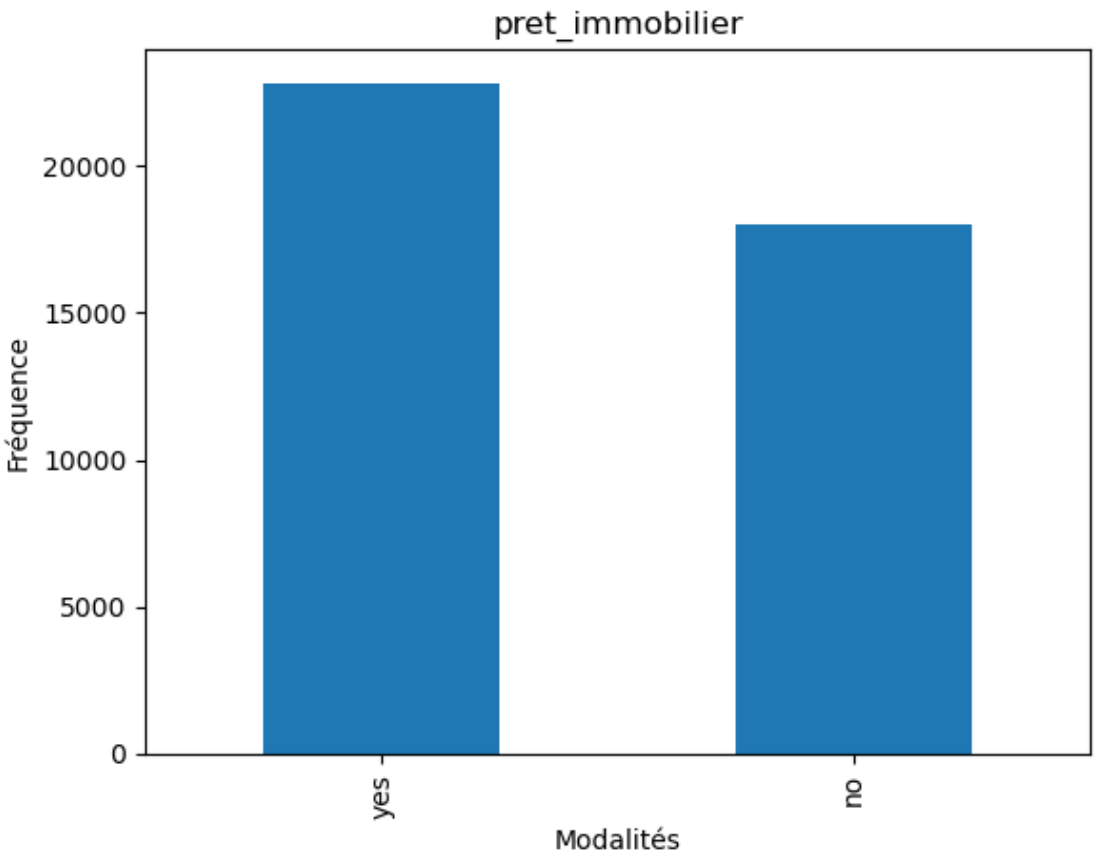
[10]:
```
# Générer un bar plot pour chaque variable catégorielle
for var in cat_vars:
    bankdata[var].value_counts().plot(kind='bar')
    plt.title(var)
    plt.xlabel('Modalités')
    plt.ylabel('Fréquence')
    plt.show()
```

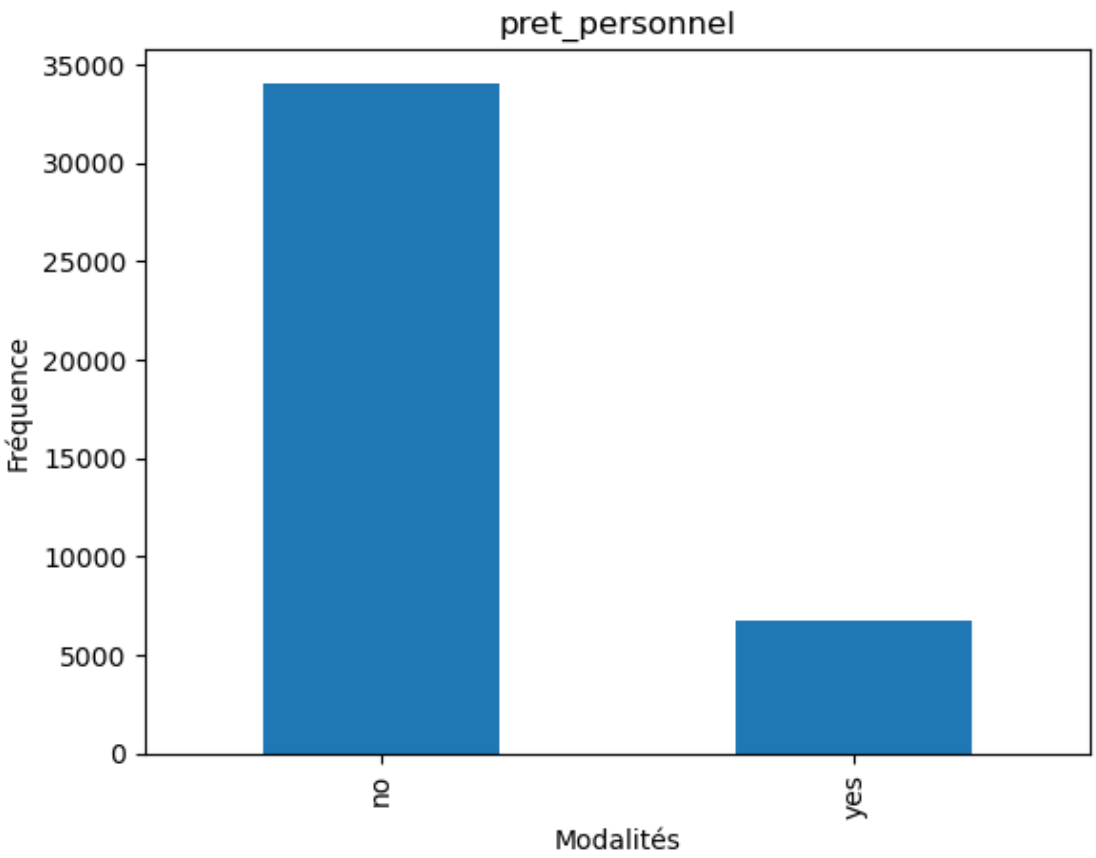

profession
Fréquence
8000
6000
4000
2000
0
blue-collar
management
technician
admin.
services
retired
self-employed
entrepreneur
unemployed
housemaid
student
other
Modalités

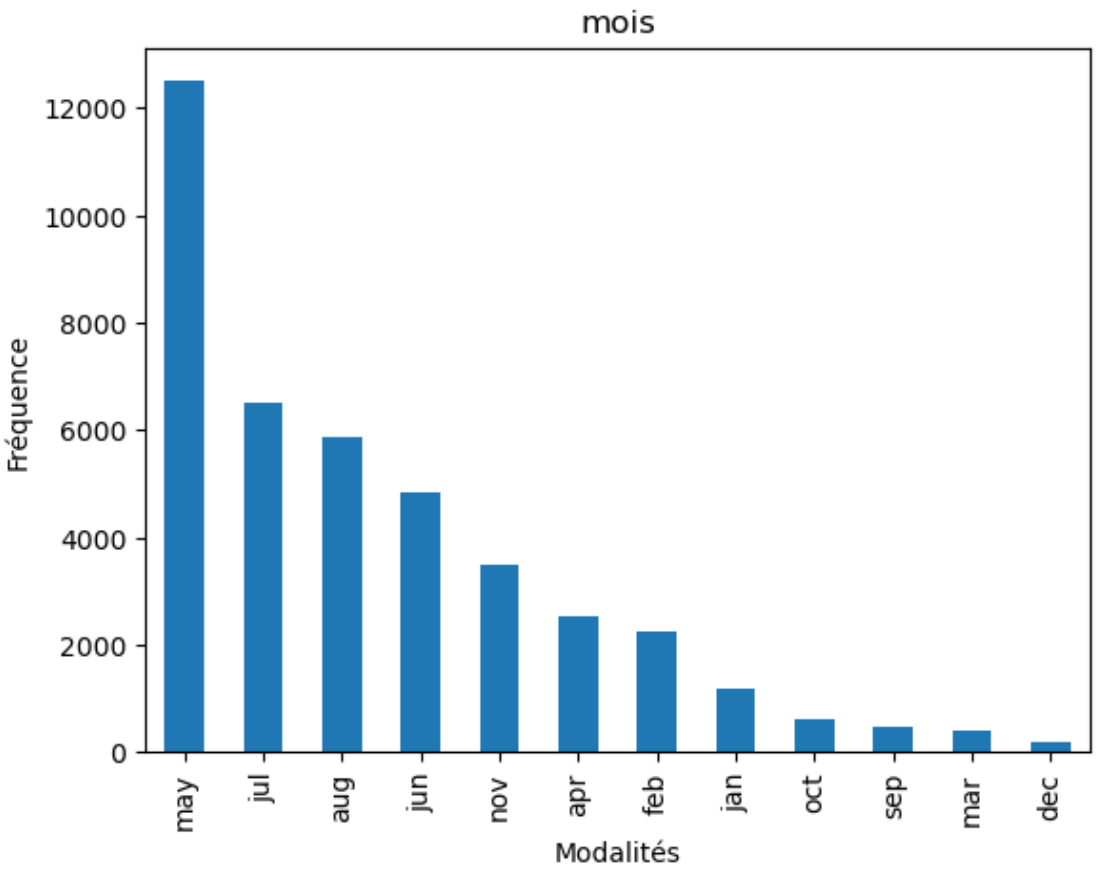

situation_familiale
25000
20000
15000
10000
5000
0
Fréquence
married
single
divorced
Modalités

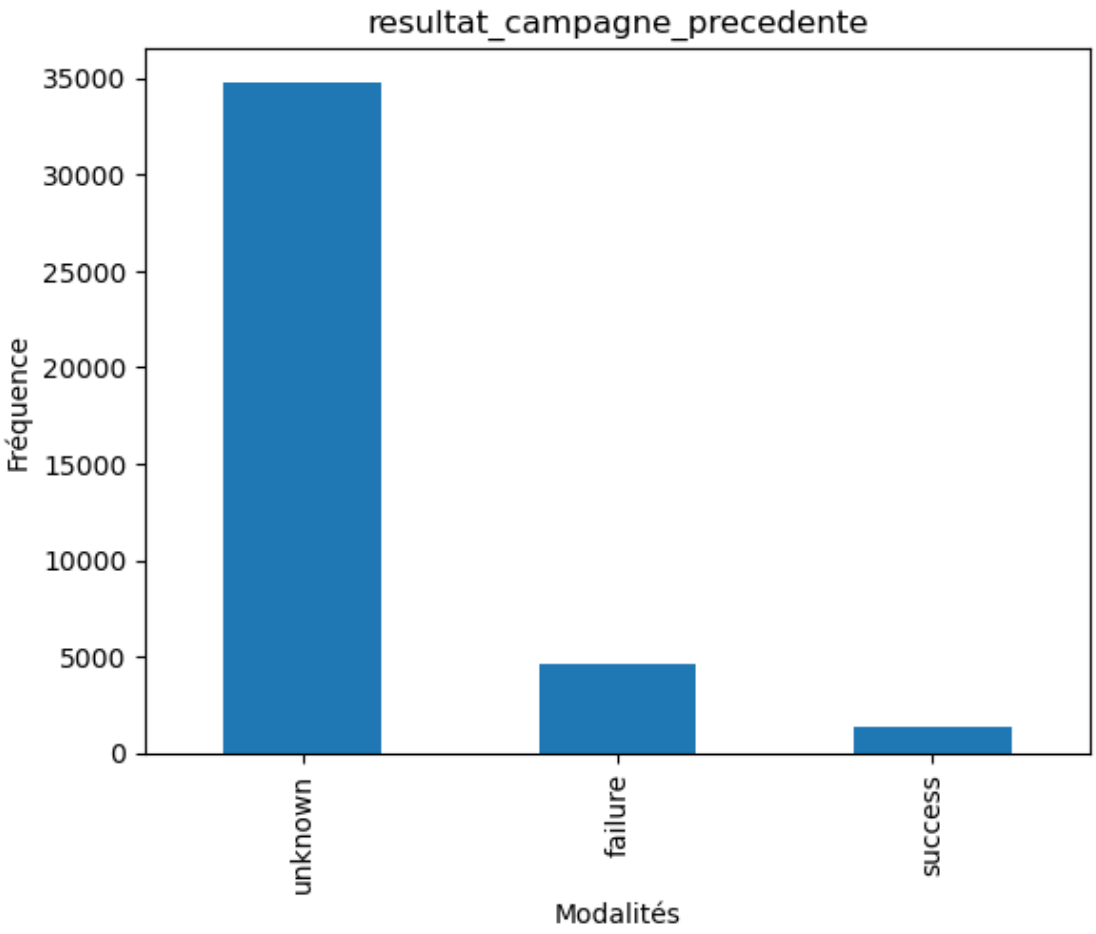

niveau_etudes
Fréquence
20000
15000
10000
5000
0
secondary
tertiary
primary
Modalités

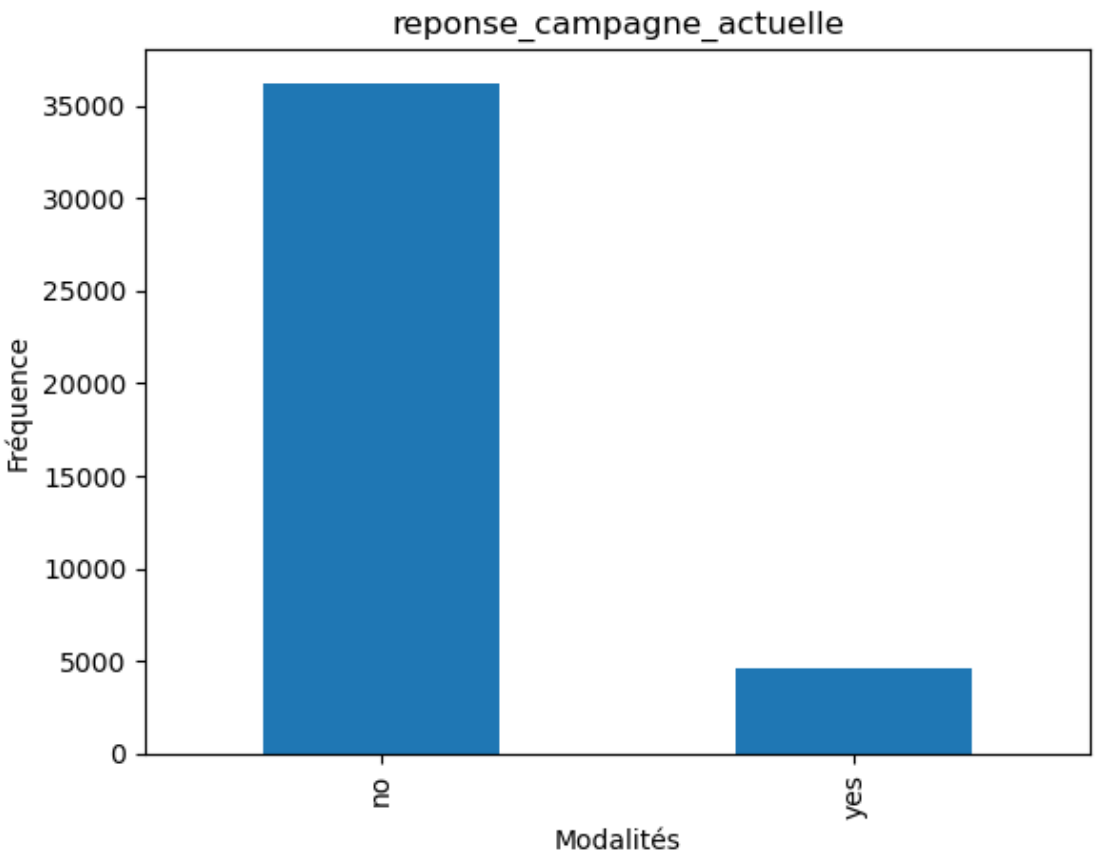

defaut_credit
40000
35000
30000
25000
20000
15000
10000
5000
0
Fréquence
no
yes
Modalités

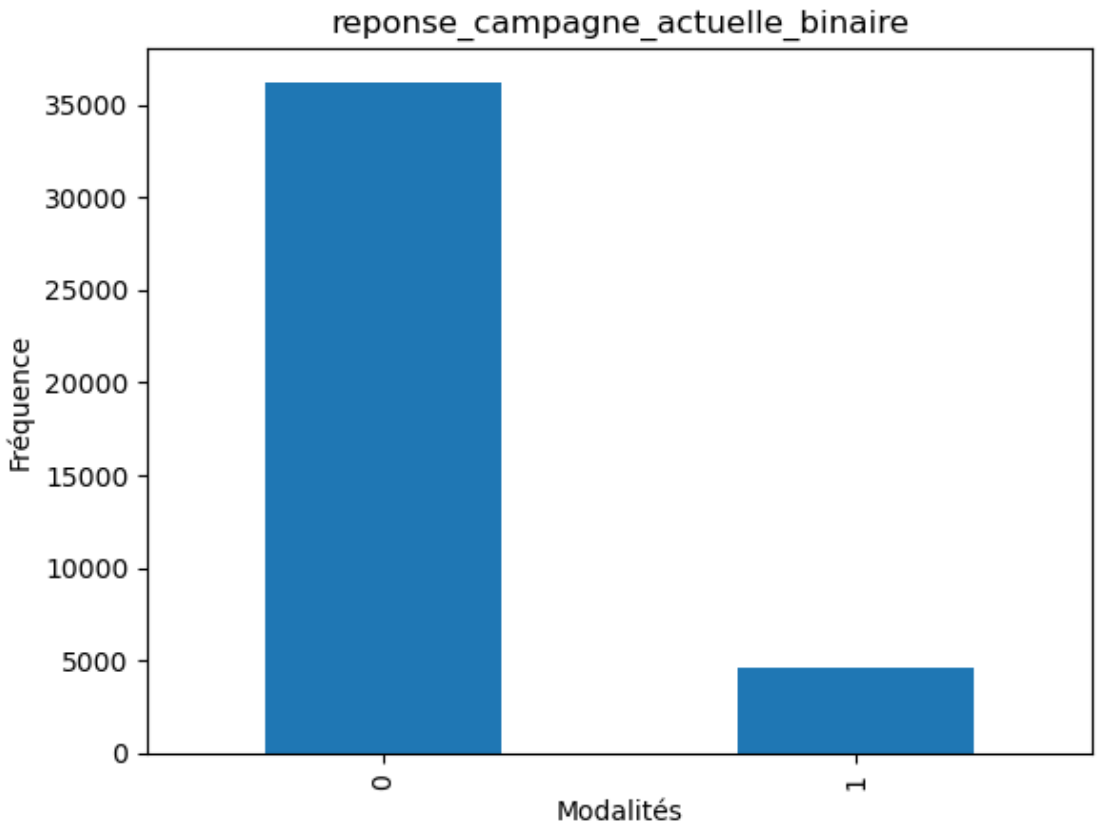

pret_immobilier
20000
15000
10000
5000
0
Fréquence
yes
no
Modalités


pret_personnel
35000
30000
25000
20000
15000
10000
5000
0
Fréquence
no
yes
Modalités


mois
12000
10000
8000
6000
4000
2000
0
Fréquence
may
jul
aug
jun
nov
apr
feb
jan
oct
sep
mar
dec
Modalités

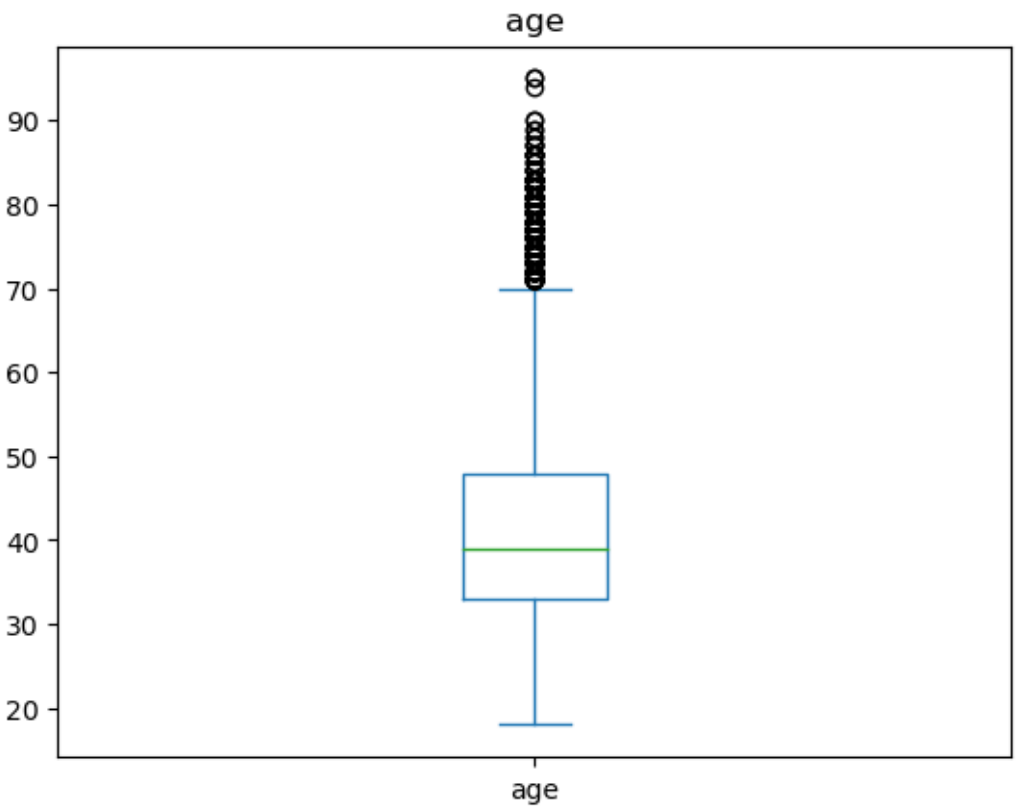

resultat_campagne_precedente
35000
30000
25000
20000
15000
10000
5000
0
Fréquence
unknown
failure
success
Modalités

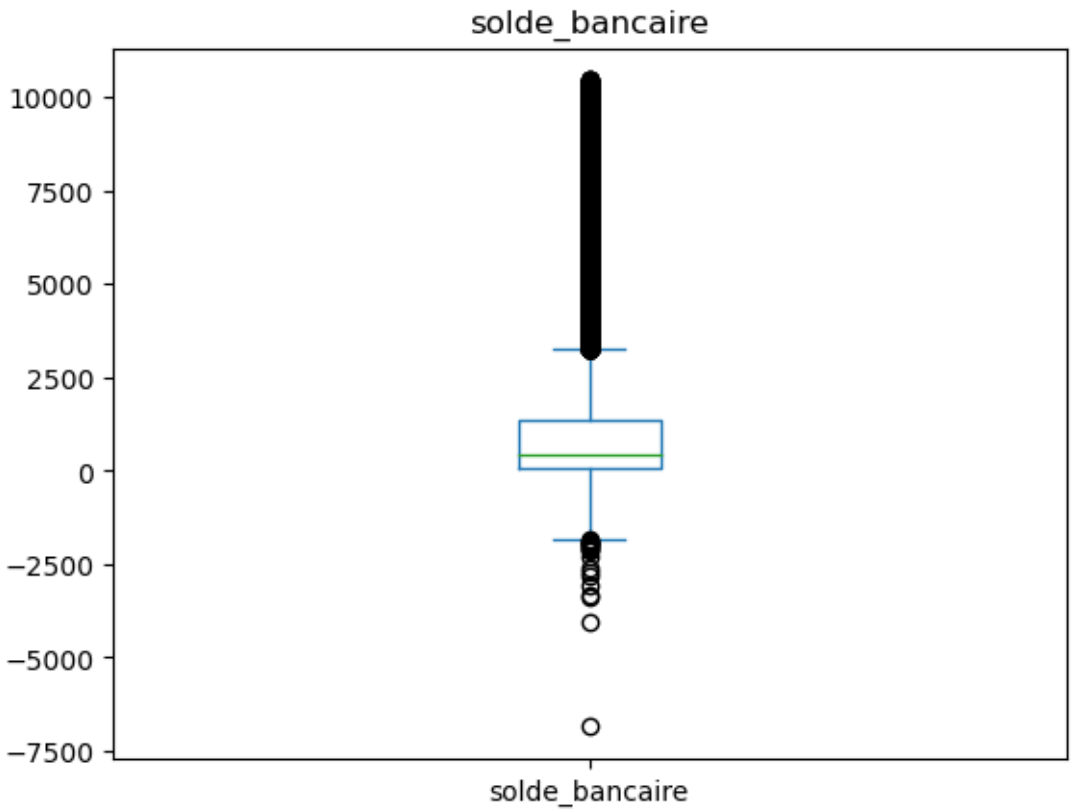

reponse_campagne_actuelle
35000
30000
25000
20000
15000
10000
5000
0
Fréquence
no
yes
Modalités

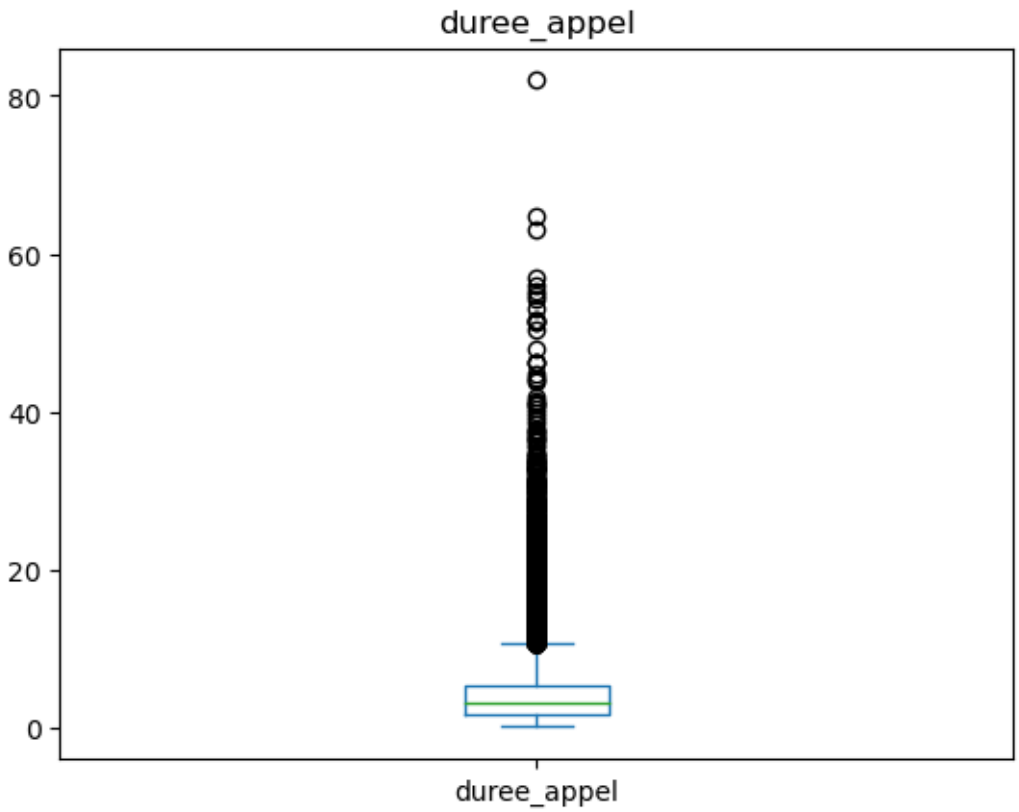


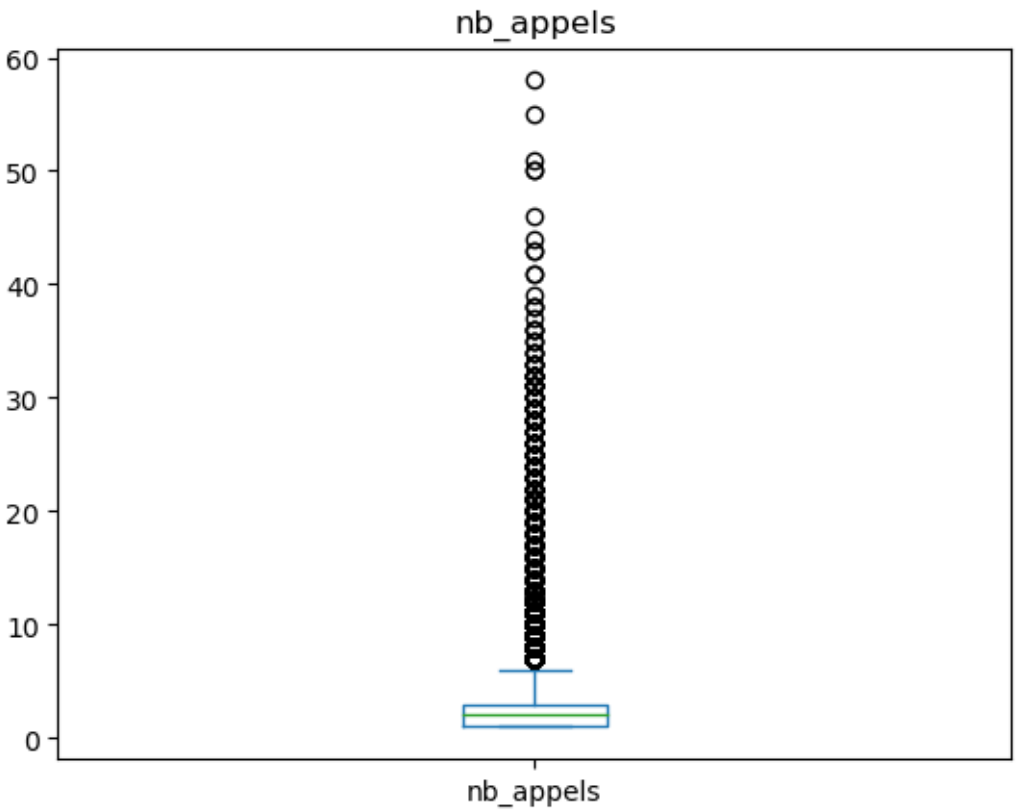
[ ]:

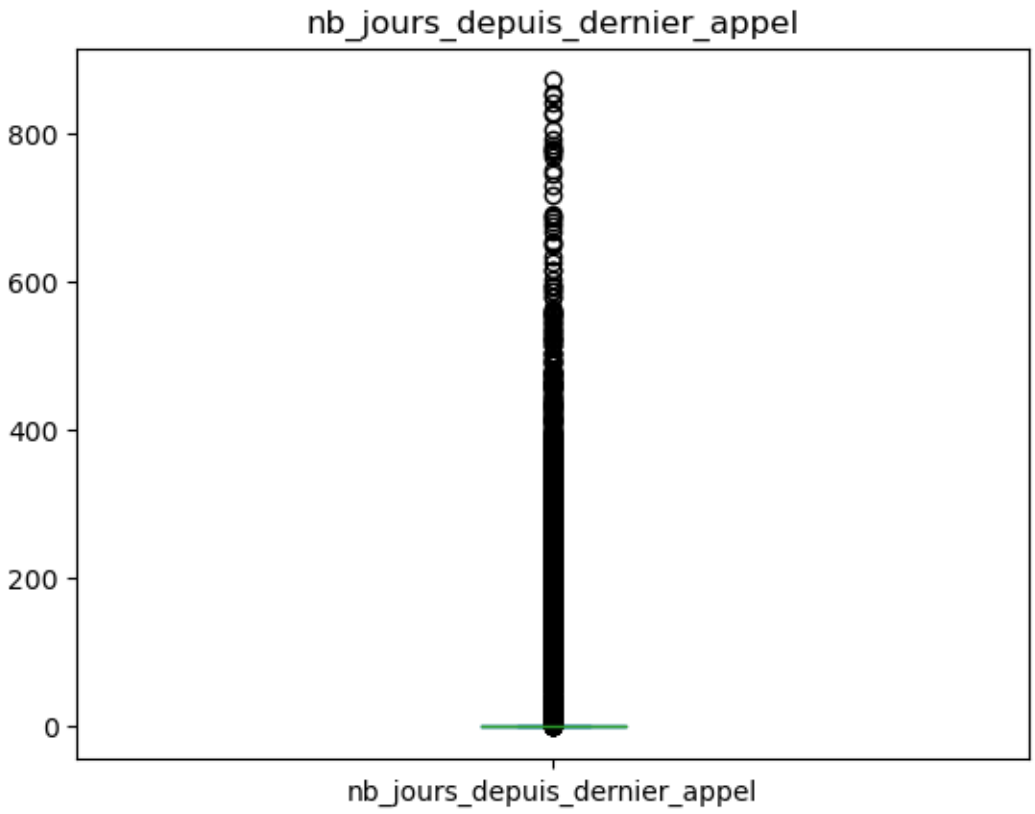
[11]:
```
# Sélectionner les variables numériques
num_vars = ['age', 'solde_bancaire', 'duree_appel', 'nb_appels',␣
 ↪'nb_jours_depuis_dernier_appel', 'nb_appels_precedents']

# Générer un box plot pour chaque variable numérique
for var in num_vars:
    bankdata[var].plot(kind='box')
    plt.title(var)
    plt.show()
```

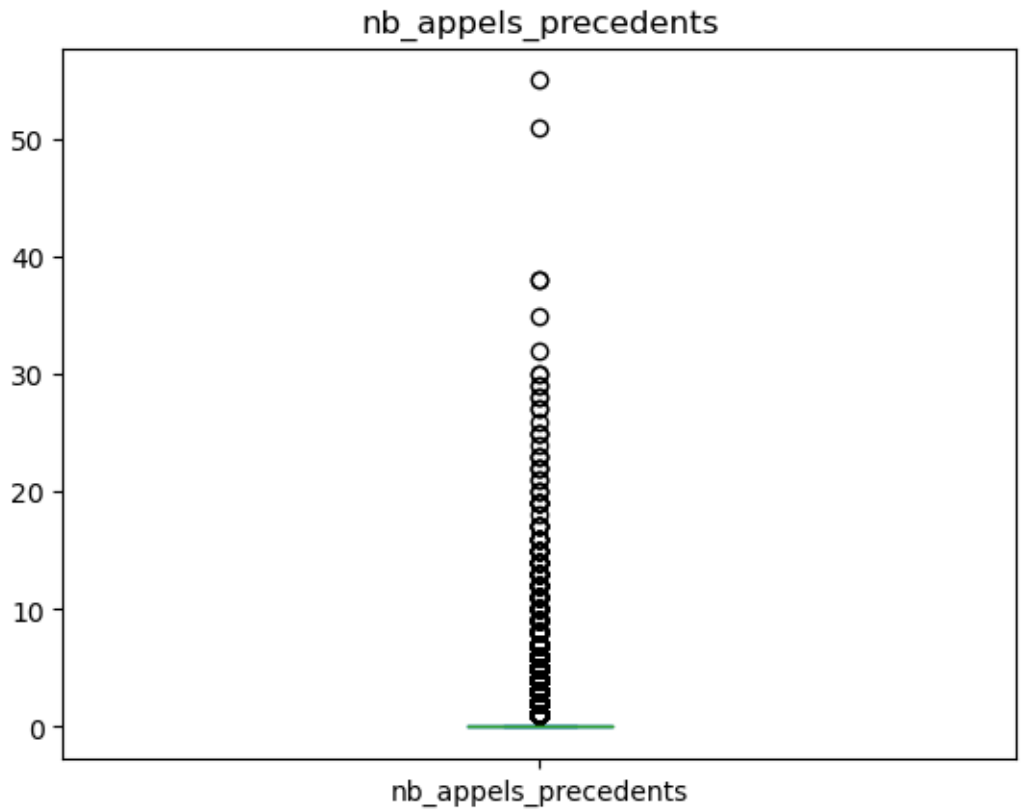

age
90
80
70
60
50
40
30
20
age


solde_bancaire
10000
7500
5000
2500
0
−2500
−5000
−7500
solde_bancaire


duree_appel
80
60
40
20
0
duree_appel


nb_appels
60
50
40
30
20
10
0
nb_appels

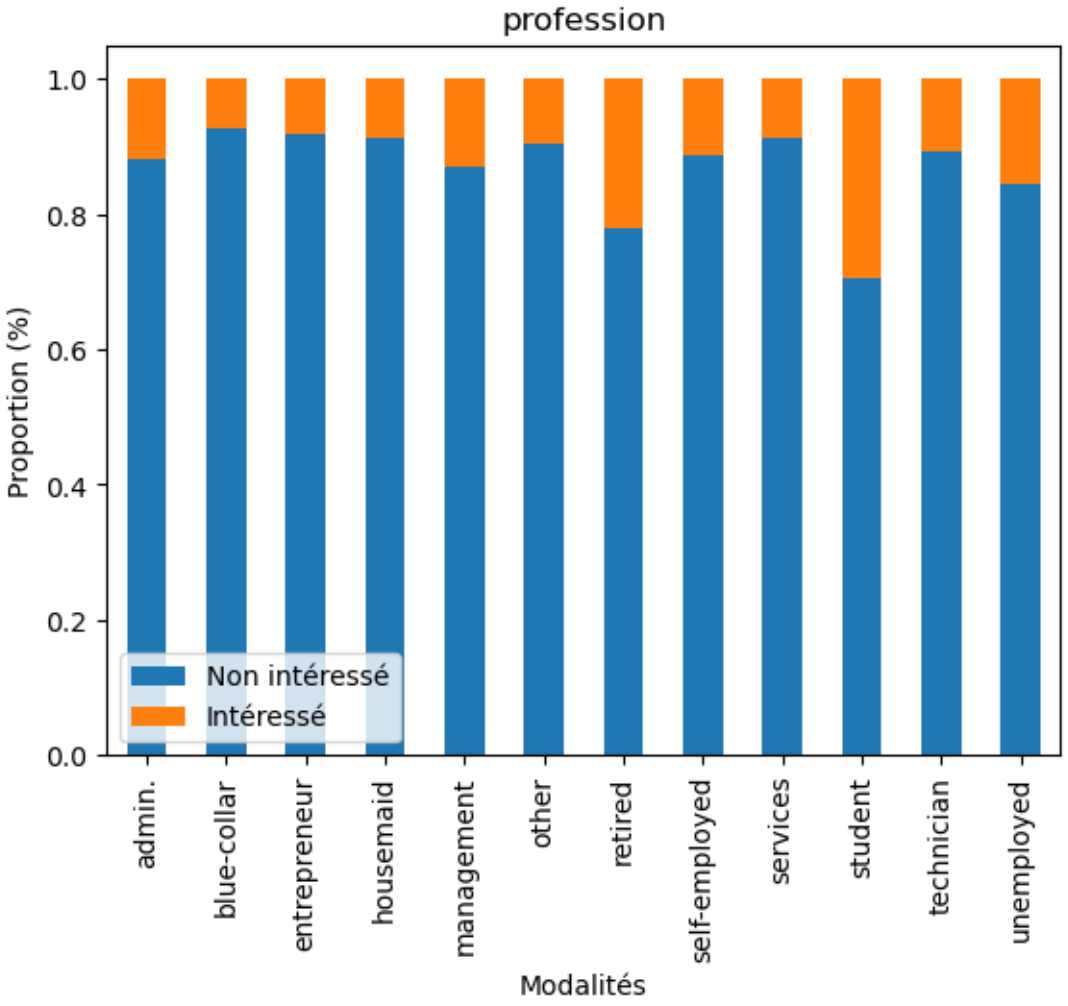

nb_jours_depuis_dernier_appel
800
600
400
200
0
nb_jours_depuis_dernier_appel

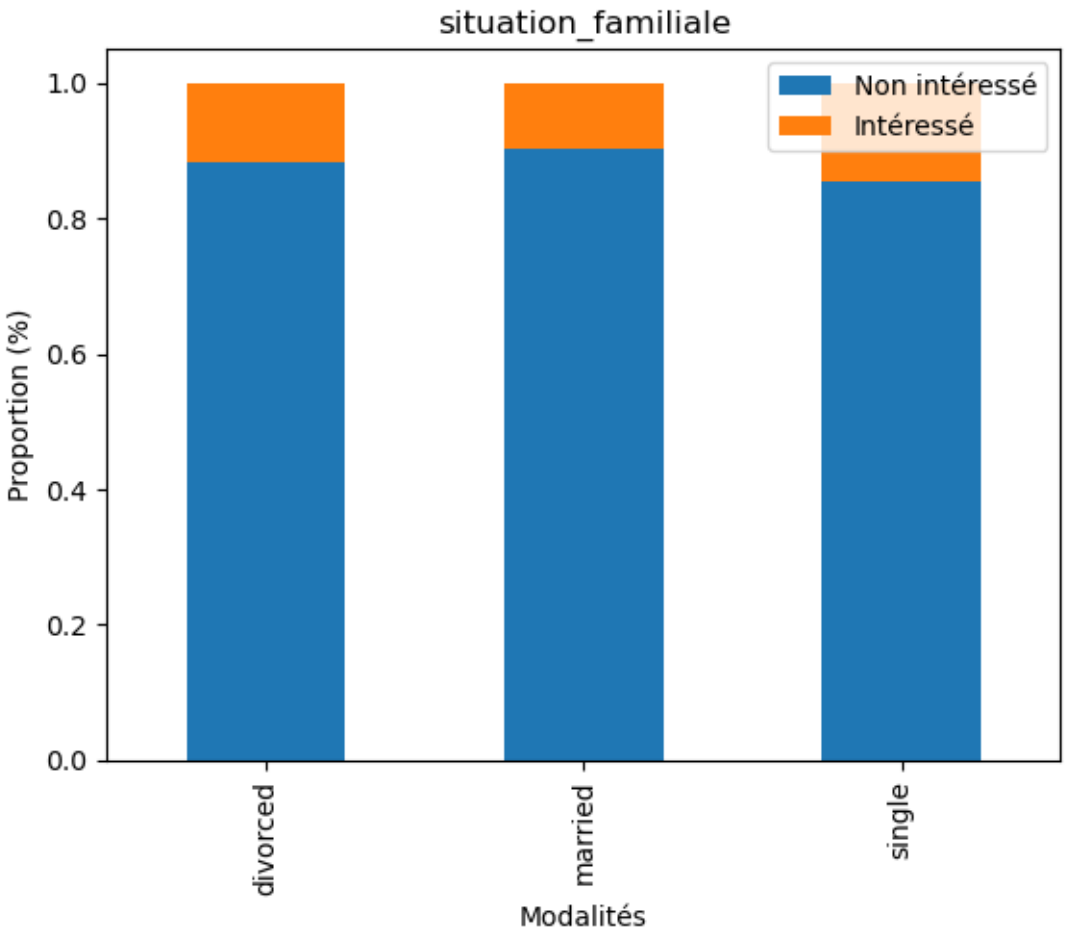


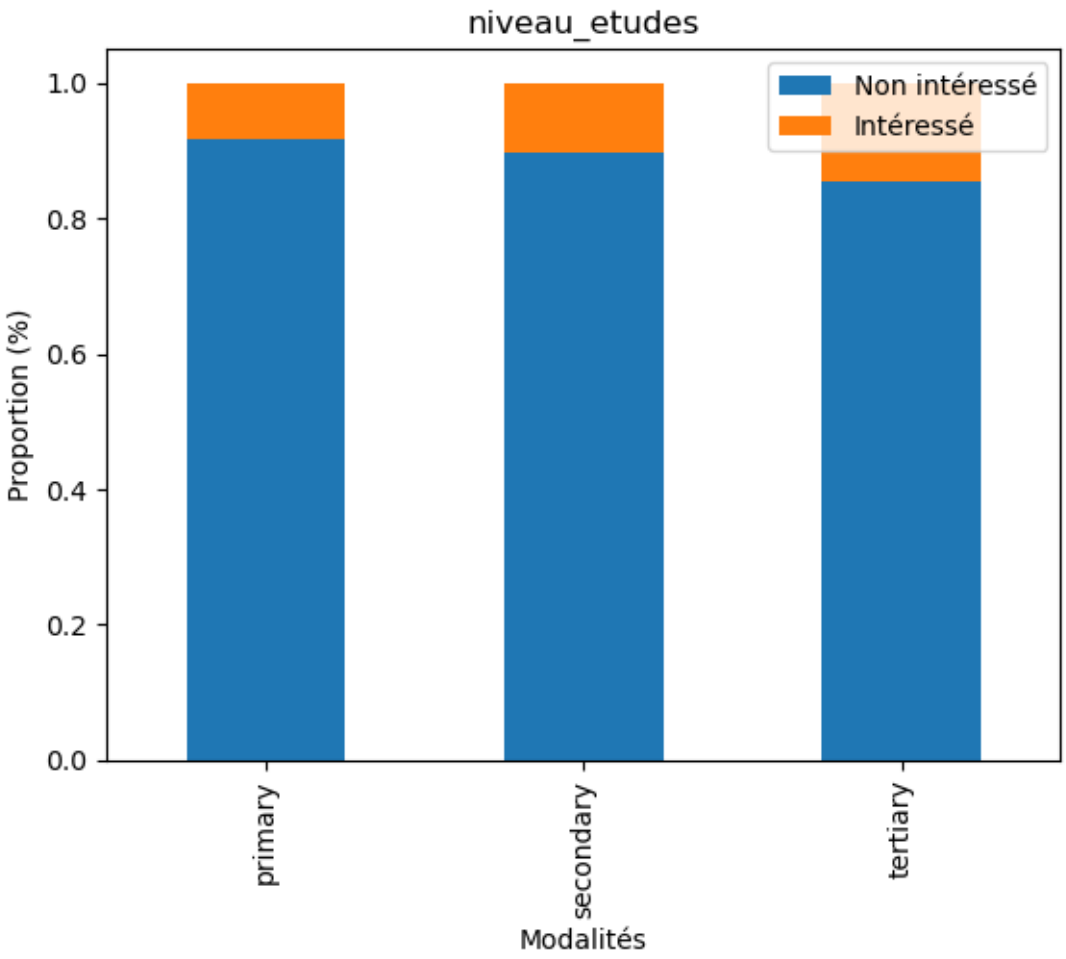
4 Analyses bivariée

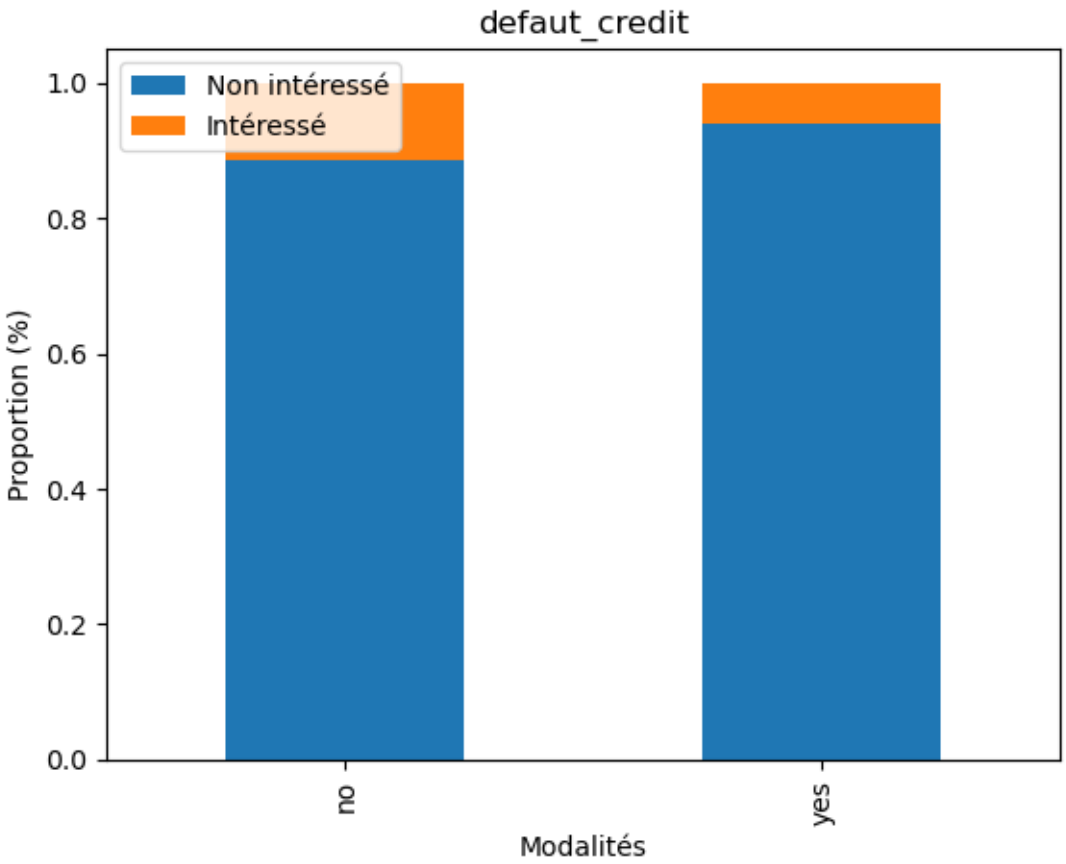
[12]:
```
import seaborn as sns
# Sélectionner les variables catégorielles
# Sélectionner les variables catégorielles
cat_vars = ['profession', 'situation_familiale', 'niveau_etudes',␣
 ↪'defaut_credit', 'pret_immobilier', 'pret_personnel',␣
 ↪'resultat_campagne_precedente']


# Générer un count plot pour chaque variable catégorielle
# Définir la taille des figures




# Générer un bar plot pour chaque variable catégorielle
for var in cat_vars:
    figsize = (20, 20)
```

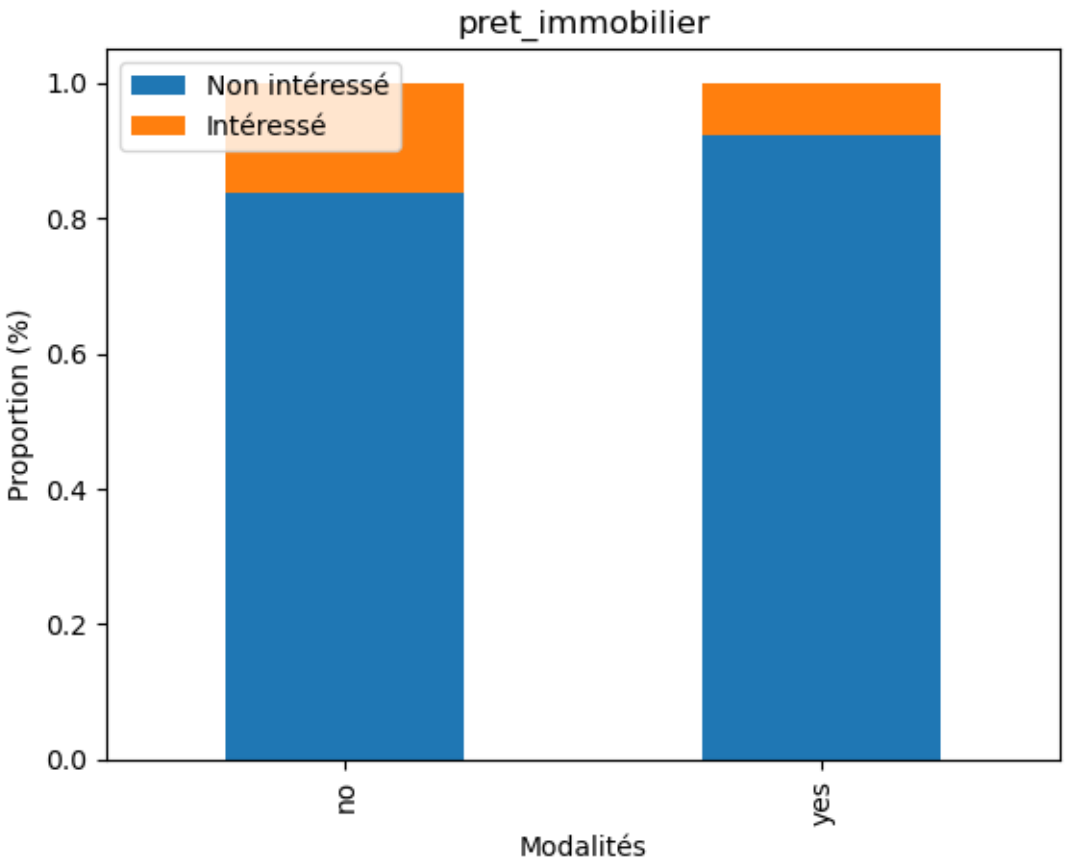
```
    (bankdata.groupby([var,␣
↪'reponse_campagne_actuelle_binaire'])['reponse_campagne_actuelle_binaire'].
↪count()/bankdata.groupby([var])[var].count()).unstack(level=1).
↪plot(kind='bar', stacked=True)
    plt.title(var)
    plt.xlabel('Modalités')
    plt.ylabel('Proportion (%)')
    plt.legend(['Non intéressé', 'Intéressé'])
    plt.show()
```

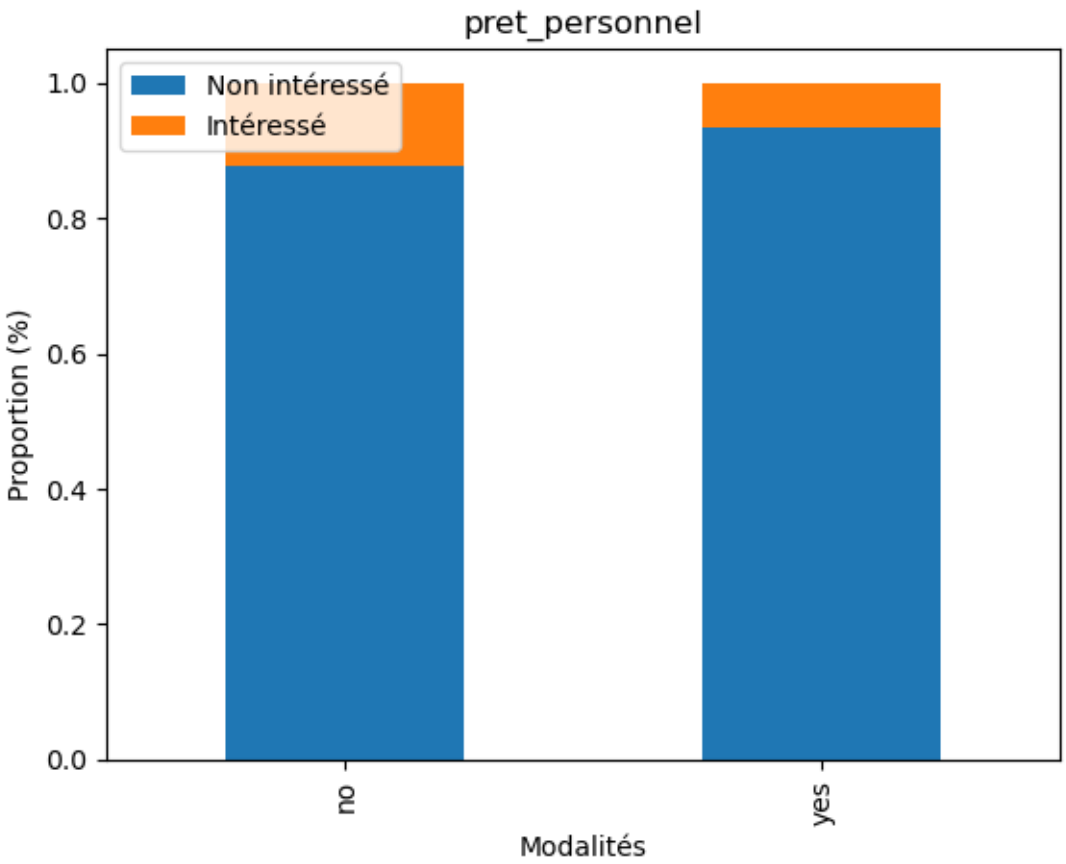


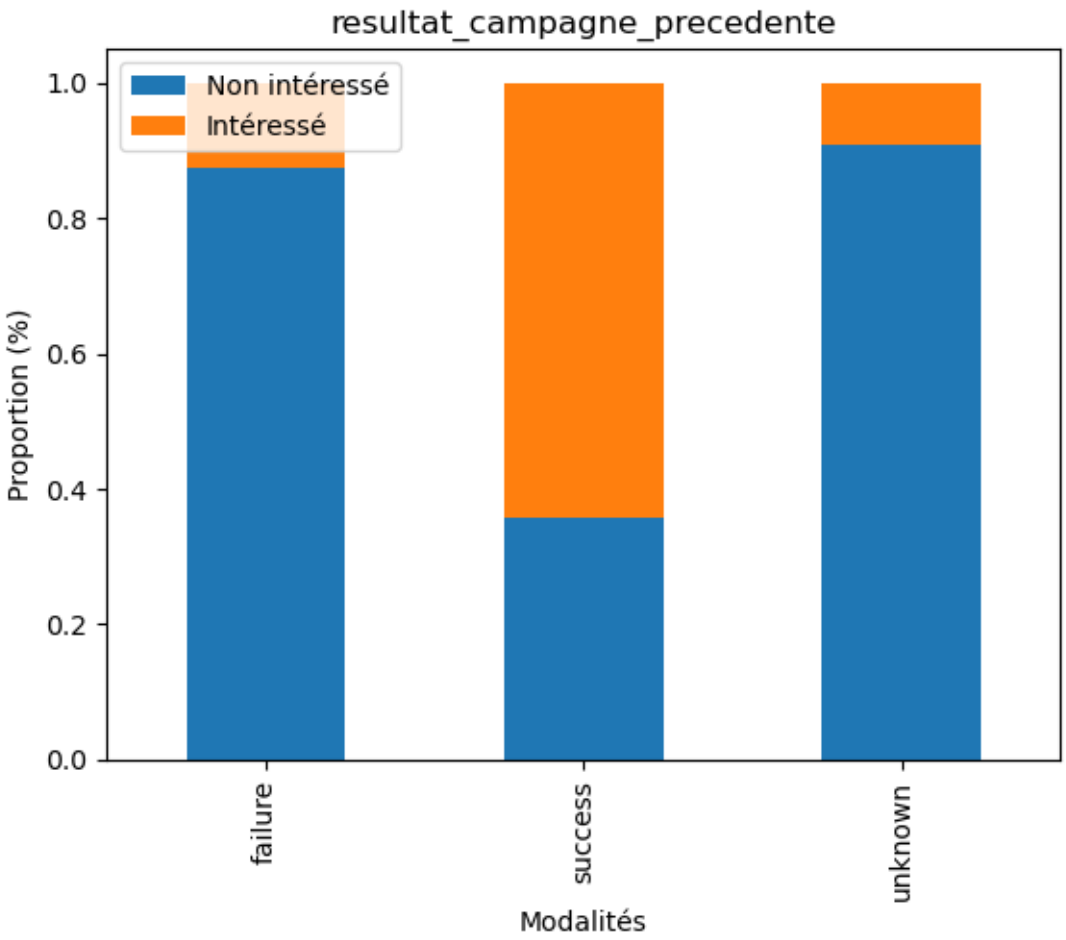

situation_familiale
Non intéressé
Intéressé
Proportion (%)
1.0
0.8
0.6
0.4
0.2
0.0
divorced
married
single
Modalités


niveau_etudes
Non intéressé
Intéressé
1.0
0.8
0.6
0.4
0.2
0.0
Proportion (%)
primary
secondary
tertiary
Modalités


defaut_credit
Non intéressé
Intéressé
1.0
0.8
0.6
0.4
0.2
0.0
Proportion (%)
no
yes
Modalités


pret_immobilier
Non intéressé
Intéressé
1.0
0.8
0.6
0.4
0.2
0.0
Proportion (%)
no
yes
Modalités


pret_personnel
Non intéressé
Intéressé
1.0
0.8
0.6
0.4
0.2
0.0
Proportion (%)
no
yes
Modalités



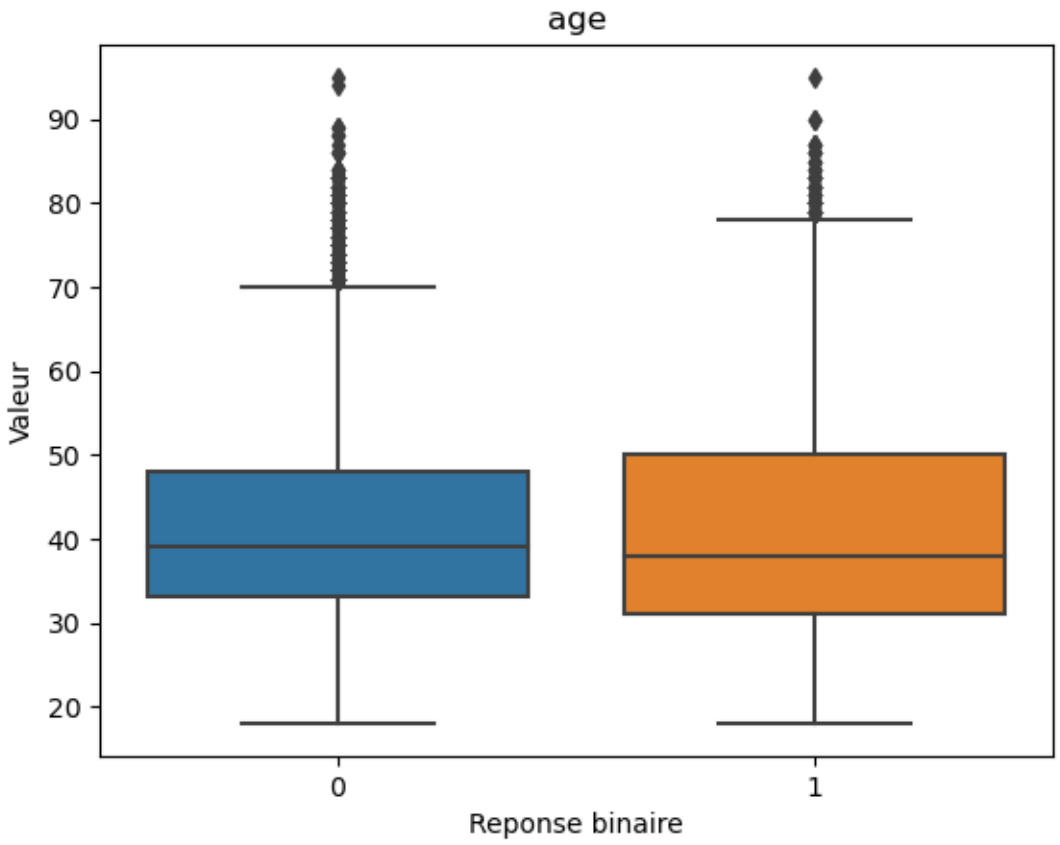
[13]:
```
from scipy.stats import chi2_contingency
# Sélectionner les variables catégorielles
cat_vars = ['profession', 'situation_familiale', 'niveau_etudes',␣
 ↪'defaut_credit', 'pret_immobilier', 'pret_personnel',␣
 ↪'resultat_campagne_precedente']

# Initialiser les listes pour stocker les résultats
var_names = []
chi2_stats = []
p_values = []
cramer_vs = []

# Parcourir toutes les variables catégorielles
for var in cat_vars:
    # Calculer le tableau de contingence
    contingency_table = pd.
 ↪crosstab(bankdata['reponse_campagne_actuelle_binaire'], bankdata[var])
```

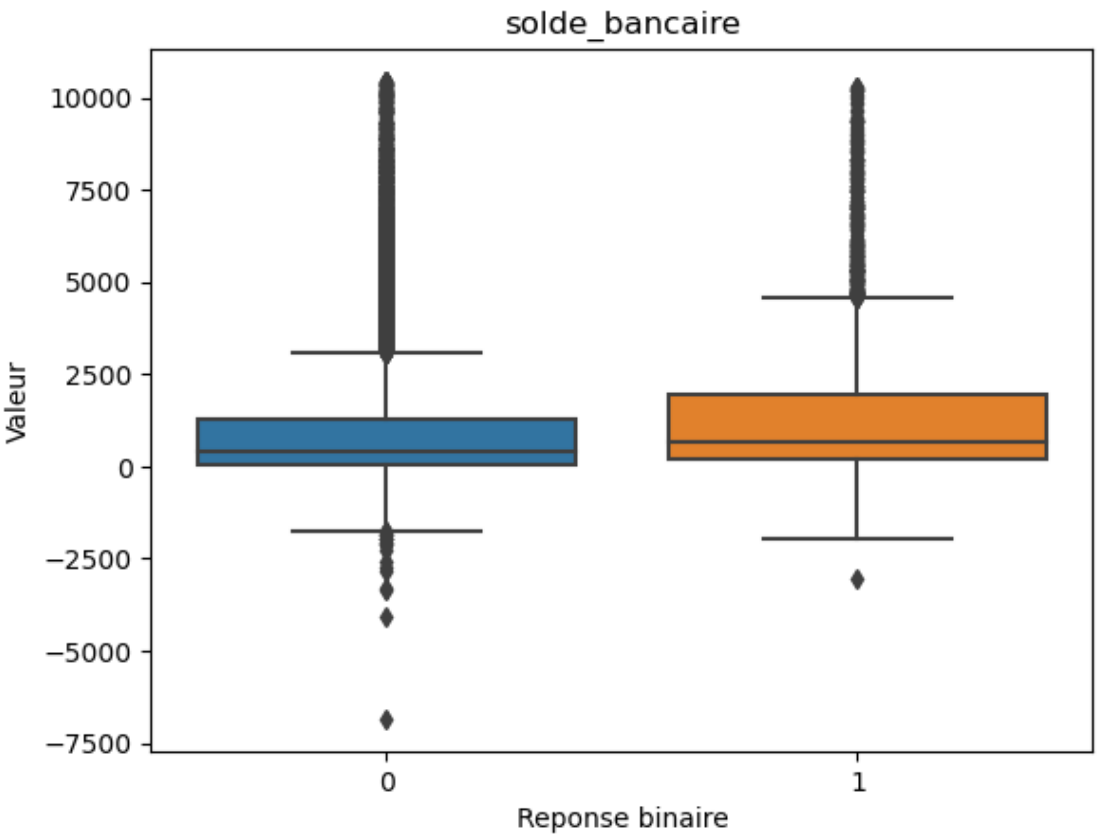
```
    # Calculer la statistique de test du Chi-deux et la p-valeur
    chi2, p, dof, expected = chi2_contingency(contingency_table)
    # Calculer le coefficient V de Cramer
    n = contingency_table.sum().sum()
    phi2 = chi2/n
    r,k = contingency_table.shape
    phi2corr = max(0, phi2-((k-1)*(r-1))/(n-1))
    rc = r-((r-1)**2)/(n-1)
    kc = k-((k-1)**2)/(n-1)
    cramer_v = np.sqrt(phi2corr/min(rc-1,kc-1))
    # Ajouter les résultats aux listes correspondantes
    var_names.append(var)
    chi2_stats.append(chi2)
    p_values.append(p)
    cramer_vs.append(cramer_v)

# Créer un DataFrame avec les résultats
results_df = pd.DataFrame({
    'Variable': var_names,
    'Chi2': chi2_stats,
    'P-valeur': p_values,
    'V de Cramer': cramer_vs
})

# Trier le DataFrame par ordre croissant de V de Cramer
results_df.sort_values(by='V de Cramer', inplace=True,ascending=False)

# Afficher le tableau des résultats
print(results_df)
```

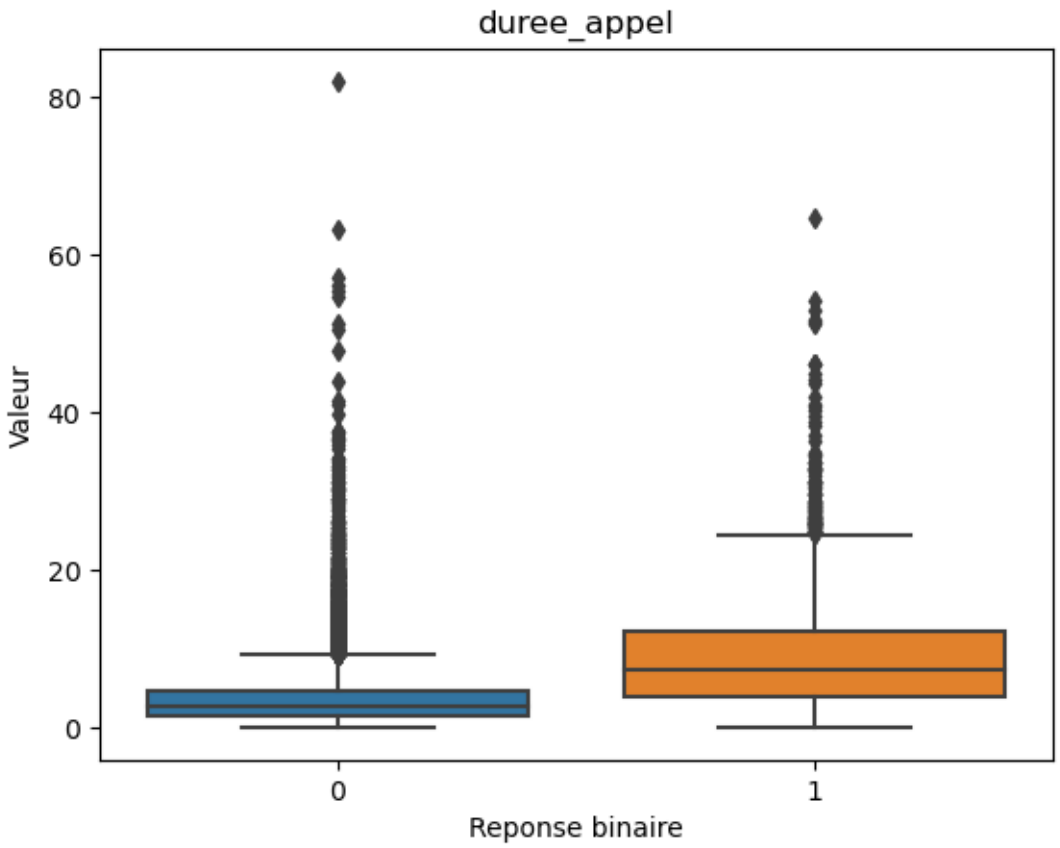
```
                      Variable         Chi2       P-valeur  V de Cramer
6  resultat_campagne_precedente  4072.108164   0.000000e+00     0.315689
4               pret_immobilier   720.301364  1.151140e-158     0.132713
0                    profession   708.668621  7.474840e-145     0.130702
2                 niveau_etudes   210.723005   1.746147e-46     0.071489
1           situation_familiale   181.858771   3.234965e-40     0.066363
5                pret_personnel   176.976953   2.215667e-40     0.065642
3                 defaut_credit    21.401052   3.725668e-06     0.022350
```

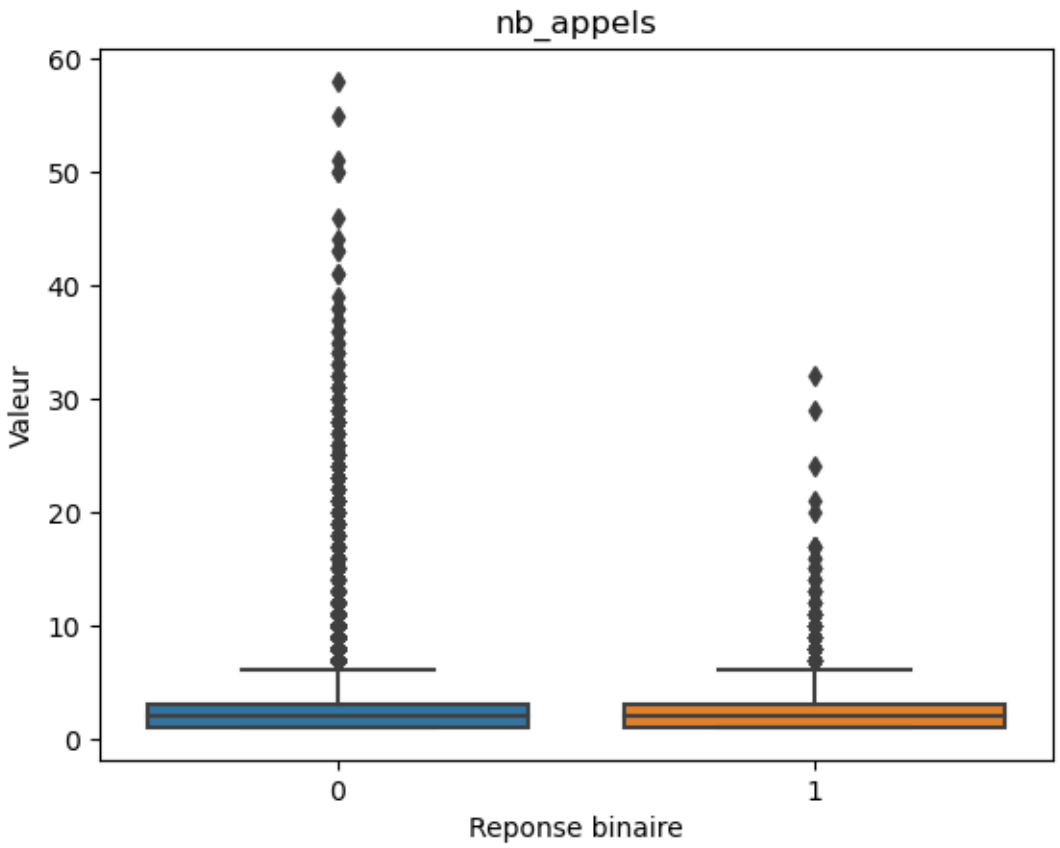
[14]:
```
# Sélectionner les variables numériques

# Sélectionner les variables numériques
num_vars = ['age', 'solde_bancaire', 'duree_appel', 'nb_appels',␣
 ↪'nb_jours_depuis_dernier_appel', 'nb_appels_precedents']


# Générer un box plot pour chaque variable numérique
```

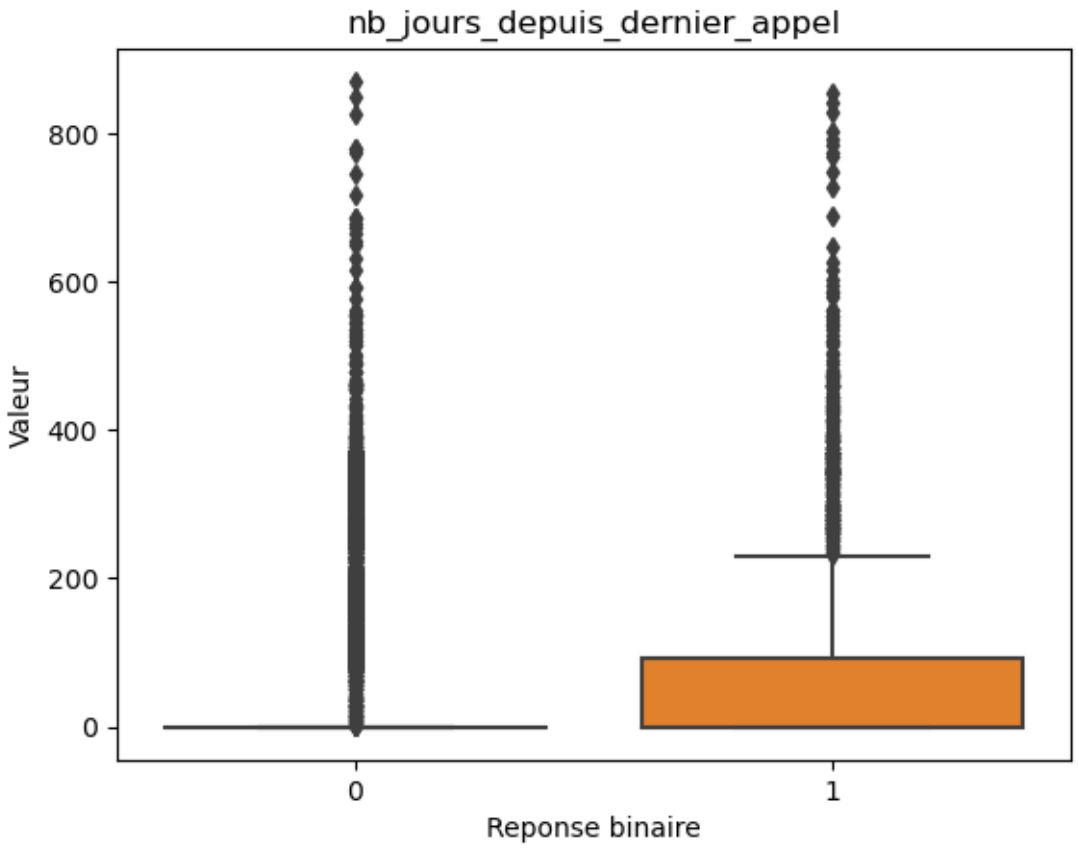
```
for var in num_vars:
    sns.boxplot(x='reponse_campagne_actuelle_binaire', y=var, data=bankdata)
    plt.title(var)
    plt.xlabel('Reponse binaire')
    plt.ylabel('Valeur')
    plt.show()
```

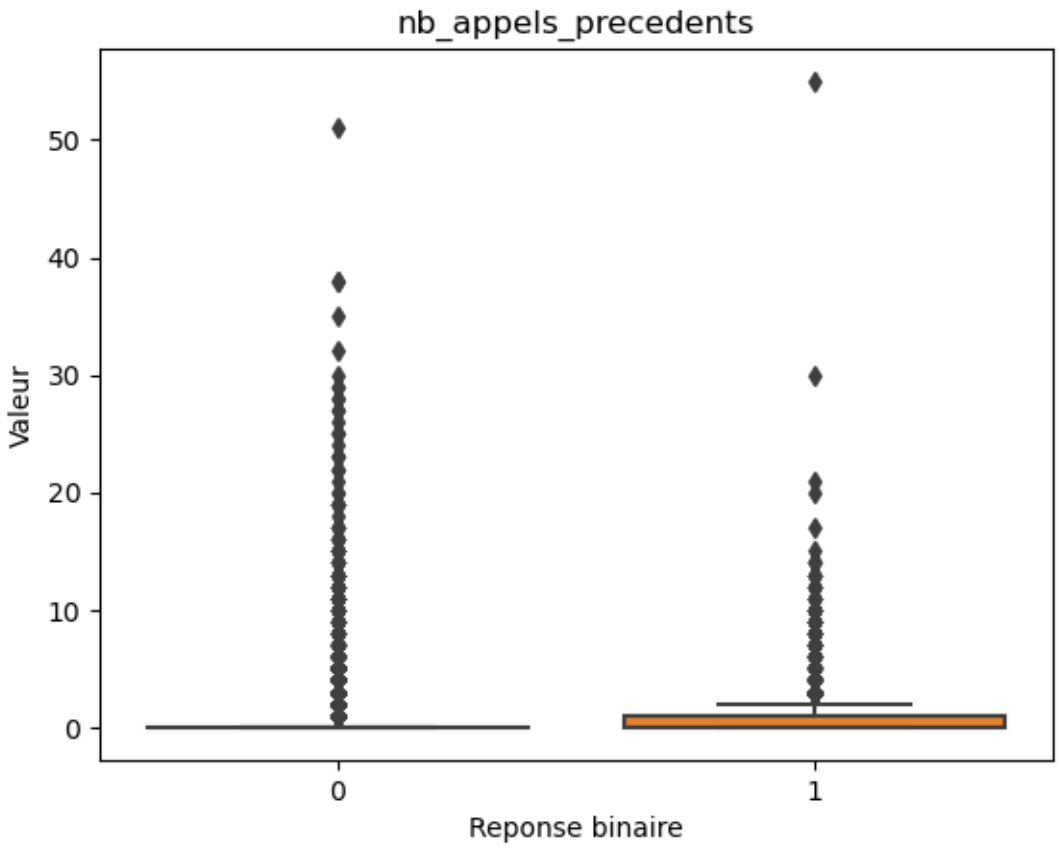



solde_bancaire
10000
7500
5000
2500
0
−2500
−5000
−7500
Valeur
0
1
Reponse binaire


duree_appel
Valeur
80
60
40
20
0
0
1
Reponse binaire


nb_appels
60
50
40
30
20
10
0
Valeur
0
1
Reponse binaire


nb_jours_depuis_dernier_appel
800
600
400
200
0
Valeur
0
1
Reponse binaire



[15]:
```
import scipy.stats as stats
# Initialiser les listes pour stocker les résultats
var_names = []
kw_stats = []
p_values = []

# Parcourir toutes les variables numériques
for var in num_vars:
    # Calculer les groupes de valeurs
    groups = [bankdata[bankdata['reponse_campagne_actuelle_binaire'] ==␣
 ↪0][var], bankdata[bankdata['reponse_campagne_actuelle_binaire'] == 1][var]]
    # Appliquer le test de Kruskal-Wallis
    kw_stat, p = stats.kruskal(*groups)
    # Ajouter les résultats aux listes correspondantes
    var_names.append(var)
    kw_stats.append(kw_stat)
    p_values.append(p)

# Créer un DataFrame avec les résultats
results_df = pd.DataFrame({
```

```
    'Variable': var_names,
    'Kruskal-Wallis': kw_stats,
    'P-valeur': p_values
})

# Trier le DataFrame par ordre croissant de p-valeur
results_df.sort_values(by='P-valeur', inplace=True)

# Afficher le tableau des résultats
print(results_df)
```

```
                       Variable  Kruskal-Wallis       P-valeur
2                   duree_appel     4767.047442   0.000000e+00
5           nb_appels_precedents     1241.504630  5.825212e-272
4  nb_jours_depuis_dernier_appel     1060.285642  1.414875e-232
1                 solde_bancaire      367.404987   6.874073e-82
3                    nb_appels      290.919176   3.135286e-65
0                          age        4.517712   3.354567e-02
```

5 Modélisation

[16]:
```
import statsmodels.api as sm
# Sélectionner les variables explicatives et la variable d'intérêt
X = bankdata[['age', 'profession', 'situation_familiale', 'niveau_etudes',␣
 ↪'defaut_credit', 'solde_bancaire',
              'pret_immobilier', 'pret_personnel', 'duree_appel', 'nb_appels',␣
 ↪'nb_jours_depuis_dernier_appel',
              'nb_appels_precedents', 'resultat_campagne_precedente']]
y = bankdata['reponse_campagne_actuelle_binaire']
```

[17]:
```
# Convertir les variables catégorielles en variables indicatrices (dummies)
X = pd.get_dummies(X, columns=['profession', 'situation_familiale',␣
 ↪'niveau_etudes', 'defaut_credit', 'pret_immobilier',
                                'pret_personnel', ␣
 ↪'resultat_campagne_precedente'], drop_first=True)
```

[18]:
```
# Ajouter une constante pour l'interception
X = sm.add_constant(X)
```

[19]:
```
# Diviser les données en ensembles d'apprentissage et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[20]:
```
# Créer le modèle de régression logistique
logit_model = sm.Logit(y_train, X_train)
```

[21]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor


# Calculer le VIF pour chaque variable explicative
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(X_train.values, i) for i in␣
 ↪range(X_train.shape[1])]
vif["features"] = X_train.columns

# Afficher les résultats
print(vif)
```

```
    VIF Factor                              features
0   113.280676                                 const
1     1.649159                                   age
2     1.047648                         solde_bancaire
3     1.013305                            duree_appel
4     1.025593                              nb_appels
5     4.805838          nb_jours_depuis_dernier_appel
6     1.812787                   nb_appels_precedents
7     2.483125                 profession_blue-collar
8     1.297411                profession_entrepreneur
9     1.289581                   profession_housemaid
10    3.023262                  profession_management
11    1.038067                       profession_other
12    1.691235                     profession_retired
13    1.322147                profession_self-employed
14    1.638126                     profession_services
15    1.190628                      profession_student
16    2.078461                   profession_technician
17    1.231003                   profession_unemployed
18    2.514295             situation_familiale_married
19    2.843501              situation_familiale_single
20    2.491433                 niveau_etudes_secondary
21    3.359809                  niveau_etudes_tertiary
22    1.018237                       defaut_credit_yes
23    1.136293                     pret_immobilier_yes
24    1.031292                      pret_personnel_yes
25    1.375903  resultat_campagne_precedente_success
26    6.472694  resultat_campagne_precedente_unknown
```

[22]:
```
# Ajuster le modèle aux données d'apprentissage
result = logit_model.fit()
```

```
Optimization terminated successfully.
         Current function value: 0.245375
         Iterations 8
```

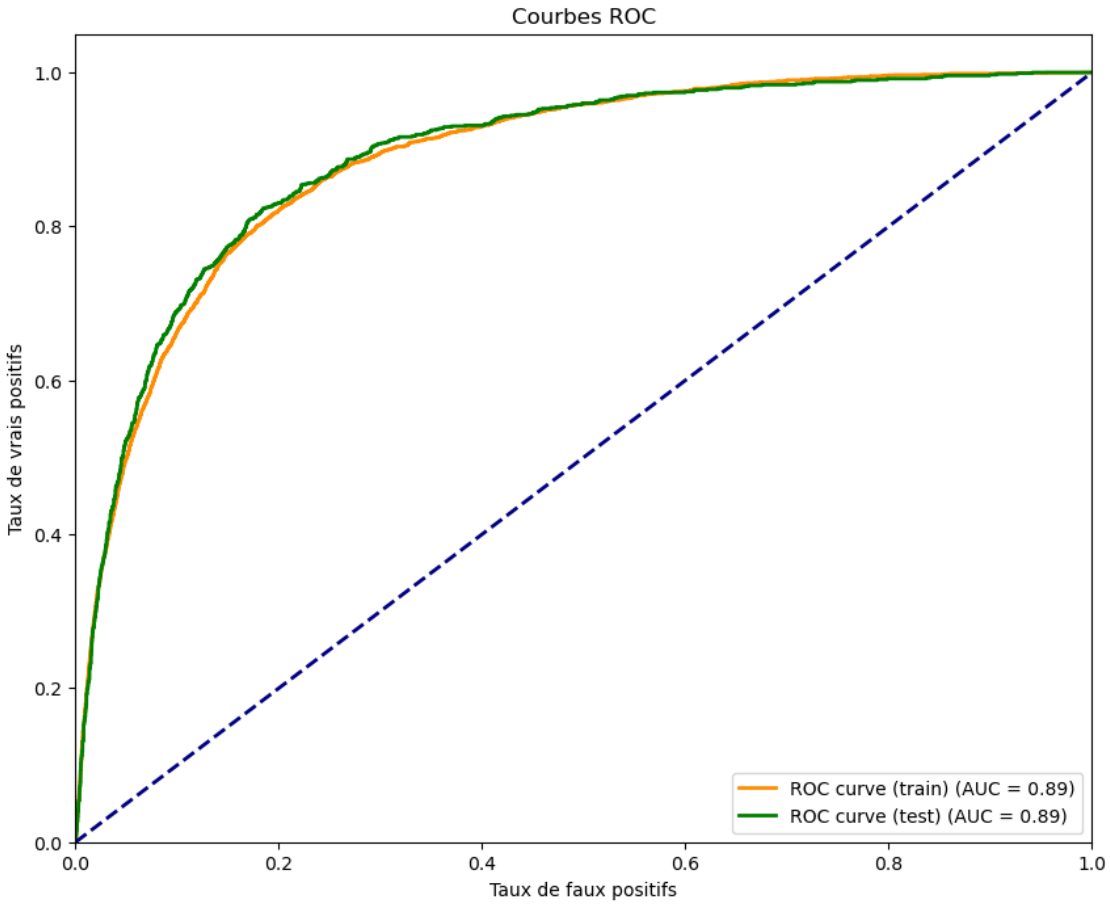
[23]:
```
# Afficher le résumé des résultats de la régression
print(result.summary())
```

```
                                 Logit Regression Results
==============================================================================
Dep. Variable:     reponse_campagne_actuelle_binaire   No. Observations:
32672
Model:                                        Logit   Df Residuals:
32645
Method:                                         MLE   Df Model:
26
Date:                              Sun, 30 Apr 2023   Pseudo R-squ.:
0.2973
Time:                                     23:43:37   Log-Likelihood:
-8016.9
converged:                                    True   LL-Null:
-11409.
Covariance Type:                          nonrobust   LLR p-value:
0.000
==============================================================================
                                 coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
const                         -2.6909      0.203    -13.233      0.000
-3.089      -2.292
age                           -0.0005      0.003     -0.199      0.842
-0.006       0.005
solde_bancaire              5.814e-05   1.13e-05      5.123      0.000
3.59e-05    8.04e-05
duree_appel                    0.2402      0.004     54.725      0.000
0.232       0.249
nb_appels                     -0.1101      0.012     -9.349      0.000
-0.133      -0.087
nb_jours_depuis_dernier_appel  0.0006      0.000      1.571      0.116
-0.000       0.001
nb_appels_precedents           0.0298      0.014      2.139      0.032
0.002       0.057
profession_blue-collar        -0.4196      0.084     -5.022      0.000
-0.583      -0.256
profession_entrepreneur       -0.4631      0.146     -3.174      0.002
-0.749      -0.177
profession_housemaid          -0.6168      0.161     -3.837      0.000
-0.932      -0.302
profession_management         -0.2604      0.085     -3.055      0.002
```

```
-0.427      -0.093
profession_other                        -0.3614      0.335     -1.079      0.280
-1.018       0.295
profession_retired                       0.3468      0.112      3.104      0.002
0.128       0.566
profession_self-employed                -0.5289      0.133     -3.982      0.000
-0.789      -0.269
profession_services                     -0.3290      0.096     -3.416      0.001
-0.518      -0.140
profession_student                       0.5150      0.134      3.844      0.000
0.252       0.778
profession_technician                   -0.2901      0.079     -3.656      0.000
-0.446      -0.135
profession_unemployed                   -0.2190      0.131     -1.674      0.094
-0.475       0.037
situation_familiale_married             -0.1097      0.069     -1.589      0.112
-0.245       0.026
situation_familiale_single               0.2409      0.078      3.091      0.002
0.088       0.394
niveau_etudes_secondary                  0.1844      0.074      2.505      0.012
0.040       0.329
niveau_etudes_tertiary                   0.4604      0.086      5.348      0.000
0.292       0.629
defaut_credit_yes                       -0.1693      0.190     -0.891      0.373
-0.542       0.203
pret_immobilier_yes                     -0.9259      0.046    -19.975      0.000
-1.017      -0.835
pret_personnel_yes                      -0.5071      0.068     -7.448      0.000
-0.641      -0.374
resultat_campagne_precedente_success     2.3192      0.093     24.940      0.000
2.137       2.501
resultat_campagne_precedente_unknown    -0.3594      0.118     -3.034      0.002
-0.592      -0.127
==============================================================================
========================
```

[24]:
```python
from sklearn.metrics import roc_curve, auc
# Obtenir les prédictions du modèle sur l'ensemble d'entraînement et de test
y_train_pred = result.predict(X_train)
y_test_pred = result.predict(X_test)

# Calculer les courbes ROC et les aires sous la courbe (AUC)
fpr_train, tpr_train, thresholds_train = roc_curve(y_train, y_train_pred)
roc_auc_train = auc(fpr_train, tpr_train)

fpr_test, tpr_test, thresholds_test = roc_curve(y_test, y_test_pred)
roc_auc_test = auc(fpr_test, tpr_test)
```

```
# Tracer les courbes ROC
plt.figure(figsize=(10, 8))
plt.plot(fpr_train, tpr_train, color='darkorange', lw=2, label='ROC curve␣
 ↪(train) (AUC = %0.2f)' % roc_auc_train)
plt.plot(fpr_test, tpr_test, color='green', lw=2, label='ROC curve (test) (AUC␣
 ↪= %0.2f)' % roc_auc_test)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Taux de faux positifs')
plt.ylabel('Taux de vrais positifs')
plt.title('Courbes ROC')
plt.legend(loc="lower right")
plt.show()
```



6 Oversampling, Undersampling et Smote

6.1 Oversampling ou Suréchantillonnage

L'**oversampling** est une technique de rééchantillonnage utilisée pour gérer les ensembles de données déséquilibrés en augmentant le nombre d'échantillons de la classe minoritaire. Cela peut être fait en dupliquant les échantillons existants ou en générant de nouveaux échantillons synthétiques à partir des données existantes, par exemple en utilisant des méthodes telles que SMOTE (Synthetic Minority Over-sampling Technique) ou ADASYN (Adaptive Synthetic Sampling).

6.1.1 Avantages

- Améliore la performance du modèle sur la classe minoritaire en augmentant la quantité d'informations disponibles pour l'apprentissage.
- Réduit le biais envers la classe majoritaire, ce qui peut améliorer la précision globale du modèle.
- Facilite la découverte de modèles significatifs dans les données en permettant aux algorithmes d'apprentissage d'explorer plus en profondeur la structure de la classe minoritaire.

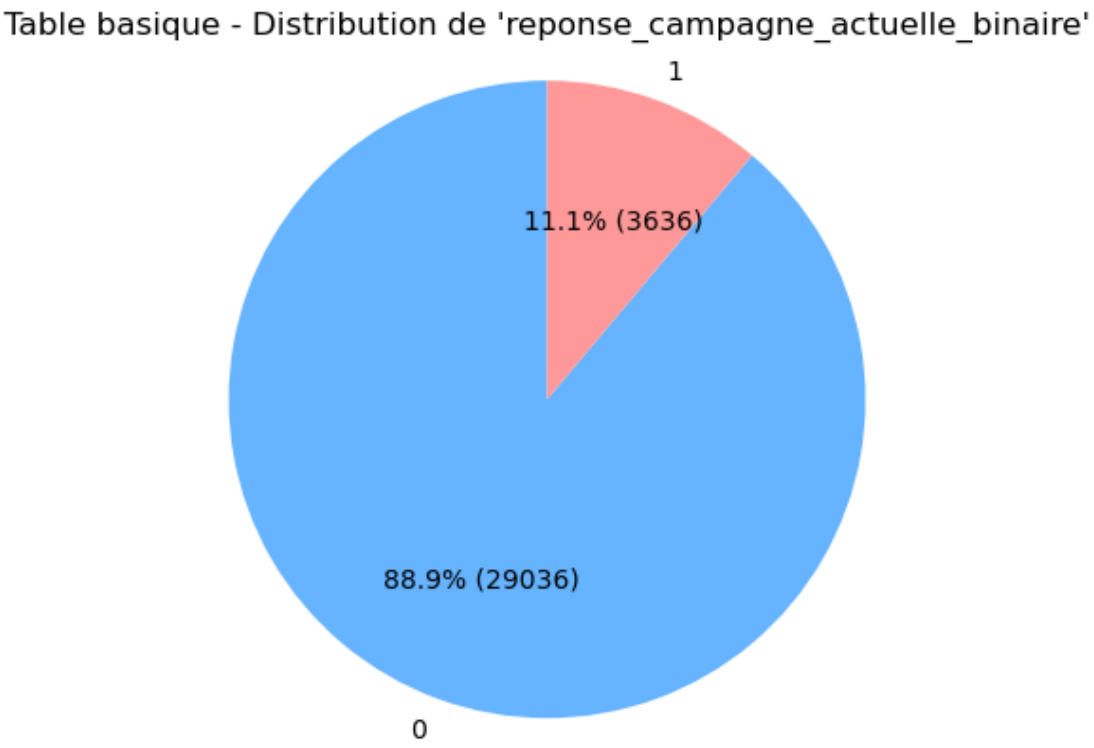
6.1.2 Inconvénients

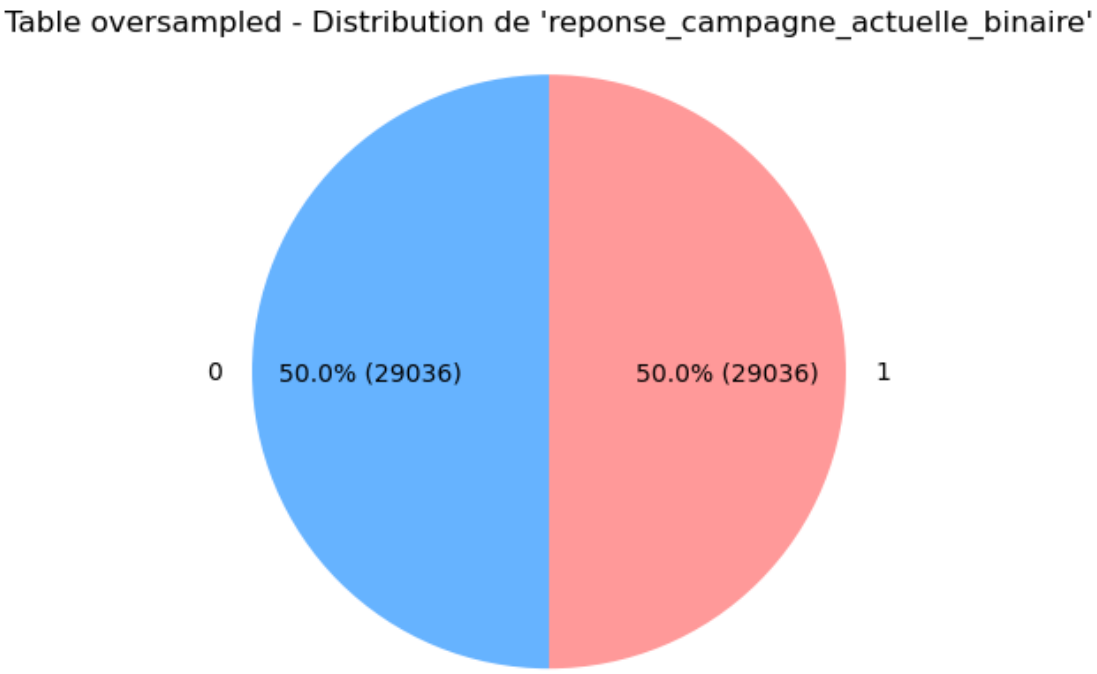
- Peut entraîner un surapprentissage, car les échantillons dupliqués ou synthétiques peuvent augmenter la complexité du modèle sans apporter d'informations nouvelles.
- Augmente la taille de l'ensemble de données, ce qui peut augmenter les temps d'apprentissage et de prédiction.

6.1.3 Analyses

[26]:
```
# Importer les bibliothèques nécessaires
from imblearn.over_sampling import RandomOverSampler
from sklearn.model_selection import train_test_split


# Diviser les données en ensembles d'apprentissage et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)


# Initialiser l'objet RandomOverSampler
ros = RandomOverSampler(sampling_strategy=1, random_state=42)

# Appliquer l'oversampling sur les données d'apprentissage
X_train_oversampled, y_train_oversampled = ros.fit_resample(X_train, y_train)

# Créer un nouveau DataFrame avec les données oversampled
bankdata_oversampled = pd.concat([X_train_oversampled, y_train_oversampled],␣
 ↪axis=1)
```

[27]:
```
# Fonction pour créer un pie chart avec les proportions et les nombres
def plot_pie_chart(y, title):
    labels = ['0', '1']
    sizes = y.value_counts().values
    colors = ['#66b3ff', '#ff9999']

    # Modifier le format des labels pour inclure les proportions et les nombres
    def autopct_format(pct, all_values):
        absolute = int(round(pct / 100 * sum(all_values)))
        return f"{pct:.1f}% ({absolute})"

    fig, ax = plt.subplots()
    ax.pie(sizes, labels=labels, colors=colors, autopct=lambda pct:␣
 ↪autopct_format(pct, sizes), startangle=90)
    ax.axis('equal')  # Pour assurer que le diagramme est bien rond
    plt.title(title)
    plt.show()

# Créer un pie chart pour la table basique
plot_pie_chart(y_train, "Table basique - Distribution de␣
 ↪'reponse_campagne_actuelle_binaire'")

# Créer un pie chart pour la table oversampled
plot_pie_chart(y_train_oversampled, "Table oversampled - Distribution de␣
 ↪'reponse_campagne_actuelle_binaire'")
```





6.1.4 Impact sur le modèle

[29]:
```
import statsmodels.api as sm
from sklearn.metrics import roc_auc_score

# Entraîner et évaluer la régression logistique sur la base basique
logreg_basique = sm.Logit(y_train, X_train).fit()
y_pred_basique = logreg_basique.predict(X_test)
y_pred_basique_train = logreg_basique.predict(X_train)
auc_basique = roc_auc_score(y_test, y_pred_basique)
auc_train_basique = roc_auc_score(y_train, y_pred_basique_train)
# Entraîner et évaluer la régression logistique sur la base oversampled
logreg_oversampled = sm.Logit(y_train_oversampled, X_train_oversampled).
 ↪fit(disp=0)
y_pred_oversampled = logreg_oversampled.predict(X_test)
auc_oversampled = roc_auc_score(y_test, y_pred_oversampled)
y_pred_oversampled_train = logreg_oversampled.predict(X_train_oversampled)
auc_oversampled_train = roc_auc_score(y_train_oversampled,␣
 ↪y_pred_oversampled_train)
```

```
Optimization terminated successfully.
         Current function value: 0.245375
```

```
          Iterations 8
```

[30]:
```
# Créer un DataFrame avec les performances
performances = pd.DataFrame({
    'Modèle': ['Base basique', 'Base oversampled'],
    'AUC - Entraînement': [auc_train_basique, auc_oversampled_train],
    'AUC - Test': [auc_basique, auc_oversampled]
})

# Afficher les performances
print(performances)
```

```
             Modèle  AUC - Entraînement  AUC - Test
0      Base basique            0.887054    0.890839
1  Base oversampled            0.888582    0.893050
```

Undersampling ou sous échantillonage

L'**undersampling** est une technique de rééchantillonnage utilisée pour gérer les ensembles de données déséquilibrés en réduisant le nombre d'échantillons de la classe majoritaire. Cela peut être fait en supprimant aléatoirement des échantillons de la classe majoritaire ou en utilisant des méthodes plus sophistiquées telles que Tomek Links ou ENN (Edited Nearest Neighbors).

6.1.5 Avantages

- Réduit la taille de l'ensemble de données, ce qui peut diminuer les temps d'apprentissage et de prédiction.
- Peut améliorer la performance du modèle sur la classe minoritaire en réduisant le biais envers la classe majoritaire.
- Élimine les échantillons bruyants ou redondants de la classe majoritaire, ce qui peut simplifier le modèle et éviter le surapprentissage.

6.1.6 Inconvénients

- Peut entraîner une perte d'informations importantes en supprimant des échantillons de la classe majoritaire, ce qui peut nuire à la performance globale du modèle.
- Ne résout pas toujours le problème du déséquilibre des classes ; dans certains cas, l'oversampling ou d'autres techniques de rééchantillonnage peuvent être plus appropriées.
- Peut ne pas être efficace si la classe majoritaire contient de nombreuses sous-classes ou groupes distincts, car l'undersampling peut éliminer certains de ces groupes et réduire la capacité du modèle à les distinguer.

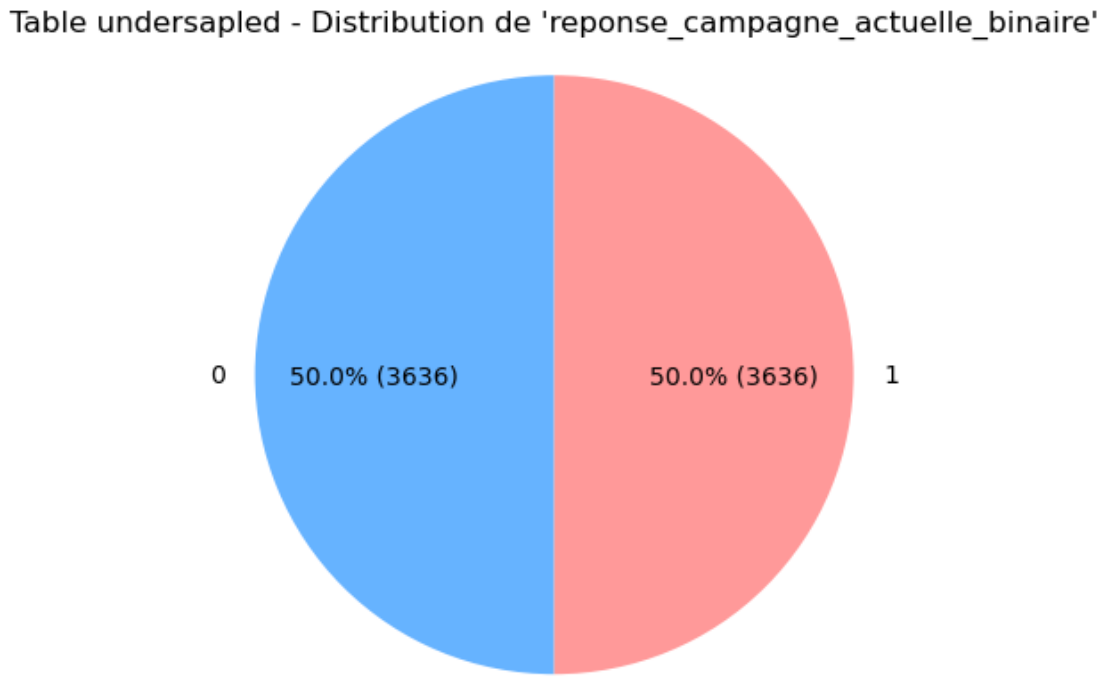
6.1.7 Analyse

[31]:
```
# Importer les bibliothèques nécessaires
from imblearn.under_sampling import RandomUnderSampler

# Initialiser l'objet RandomUnderSampler
rus = RandomUnderSampler(sampling_strategy='auto', random_state=42)
```

```
# Appliquer l'undersampling sur les données d'apprentissage
X_train_undersampled, y_train_undersampled = rus.fit_resample(X_train, y_train)

# Créer un nouveau DataFrame avec les données undersampled
bankdata_undersampled = pd.concat([X_train_undersampled, y_train_undersampled],␣
 ↪axis=1)

# Afficher la nouvelle distribution des données
print(bankdata_undersampled['reponse_campagne_actuelle_binaire'].value_counts())
```

```
0    3636
1    3636
Name: reponse_campagne_actuelle_binaire, dtype: int64
```

[32]:
```
# Créer un pie chart pour la table oversampled
plot_pie_chart(y_train_undersampled, "Table undersapled - Distribution de␣
 ↪'reponse_campagne_actuelle_binaire'")
```



6.1.8 Impact sur le modèle

[ ]:
```
# Entraîner et évaluer la régression logistique sur la base undersampled

# Entraîner et évaluer la régression logistique sur la base oversampled
lr_undersampled = sm.Logit(y_train_undersampled, X_train_undersampled).fit()


y_pred_undersampled = lr_undersampled.predict(X_test)
auc_undersampled = roc_auc_score(y_test, y_pred_undersampled)

# Ajouter les performances de la base undersampled au DataFrame
performances.loc[2] = ['Base undersampled', auc_undersampled, auc_undersampled]

# Afficher les performances
print(performances)
```

6.2 Rééchantillonage SMOTE (Synthetic Minority Over-sampling Technique)

6.3 SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE est une technique de rééchantillonnage spécifique pour gérer les ensembles de données déséquilibrés.

SMOTE génère des échantillons synthétiques de la classe minoritaire en utilisant l'interpolation entre les échantillons existants. Pour chaque échantillon de la classe minoritaire, SMOTE sélectionne un certain nombre de ses voisins les plus proches appartenant à la même classe, puis génère de nouveaux échantillons en interpolant les attributs de l'échantillon original et de ses voisins.

6.3.1 Avantages

- Améliore la performance du modèle sur la classe minoritaire en augmentant la quantité d'informations disponibles pour l'apprentissage.
- Réduit le biais envers la classe majoritaire, ce qui peut améliorer la précision globale du modèle.
- Génère des échantillons synthétiques plutôt que de dupliquer les échantillons existants, ce qui peut aider à éviter le surapprentissage et permettre une meilleure généralisation.

6.3.2 Inconvénients

- Peut créer des échantillons synthétiques qui ne représentent pas la réalité, ce qui peut entraîner un modèle moins robuste ou moins généralisable.
- Augmente la taille de l'ensemble de données, ce qui peut augmenter les temps d'apprentissage et de prédiction.

[ ]:
```
## Rééchantillonge SMOTE sur la base d'apprentissage
from imblearn.over_sampling import SMOTE
```

```
# Initialiser l'objet SMOTE
smote = SMOTE(sampling_strategy='auto', random_state=42)

# Appliquer le rééchantillonnage SMOTE sur les données d'apprentissage
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Créer un nouveau DataFrame avec les données rééchantillonnées
bankdata_smote = pd.concat([X_train_smote, y_train_smote], axis=1)

# Afficher la nouvelle distribution des données

plot_pie_chart(y_train_smote, "Table SMOTE - Distribution de␣
 ↪'reponse_campagne_actuelle_binaire'")
```

[ ]:
```
# Entraîner et évaluer la régression logistique sur la base SMOTE




lr_smote = sm.Logit(y_train_smote, X_train_smote).fit()
y_pred_smote = lr_smote.predict(X_test)
auc_smote = roc_auc_score(y_test, y_pred_smote)

# Ajouter les performances de la base SMOTE au DataFrame
performances.loc[3] = ['Base SMOTE', auc_smote, auc_smote]

# Afficher les performances
print(performances)
```

6.4 ADASYN (Adaptive Synthetic Sampling)

ADASYN est une technique de rééchantillonnage pour gérer les ensembles de données déséquilibrés.

ADASYN est similaire à SMOTE, **mais génère des échantillons synthétiques en adaptant la densité des échantillons minoritaires selon leurs voisins**. ADASYN accorde plus d'importance aux échantillons de la classe minoritaire qui sont difficiles à apprendre, en créant plus d'échantillons synthétiques pour ces échantillons.

6.4.1 Avantages

- Améliore la performance du modèle sur la classe minoritaire en augmentant la quantité d'informations disponibles pour l'apprentissage.
- Réduit le biais envers la classe majoritaire, ce qui peut améliorer la précision globale du modèle.
- Génère des échantillons synthétiques adaptés aux régions où la classe minoritaire est difficile à apprendre, ce qui peut aider à éviter le surapprentissage et permettre une meilleure généralisation.

- Peut être combiné avec d'autres techniques de rééchantillonnage, telles que l'undersampling, pour créer un ensemble de données équilibré.

6.4.2 Inconvénients

- Peut créer des échantillons synthétiques qui ne représentent pas la réalité, ce qui peut entraîner un modèle moins robuste ou moins généralisable.
- Augmente la taille de l'ensemble de données, ce qui peut augmenter les temps d'apprentissage et de prédiction.

6.4.3 Implémentation

[ ]:
```
from imblearn.over_sampling import ADASYN

adasyn = ADASYN(sampling_strategy='auto', random_state=42)
X_train_adasyn, y_train_adasyn = adasyn.fit_resample(X_train, y_train)



# Créer un pie chart pour la table oversampled
plot_pie_chart(y_train_adasyn, "Table ADASYN - Distribution de␣
 ↪'reponse_campagne_actuelle_binaire'")
```

[33]:
```
# Fonction pour l'oversampling stratifié basé sur une colonne spécifique
def stratified_oversampling(data, target_col, stratify_col):
    unique_strata = data[stratify_col].unique()
    oversampled_data = pd.DataFrame(columns=data.columns)

    for stratum in unique_strata:
        stratum_data = data[data[stratify_col] == stratum]
        X = stratum_data.drop(target_col, axis=1)
        y = stratum_data[target_col]

        ros = RandomOverSampler(random_state=42)
        X_resampled, y_resampled = ros.fit_resample(X, y)

        resampled_data = pd.concat([X_resampled, y_resampled], axis=1)
        oversampled_data = pd.concat([oversampled_data, resampled_data])

    return oversampled_data.sample(frac=1, random_state=42).
 ↪reset_index(drop=True)

# Application de l'oversampling stratifié
oversampled_bankdata = stratified_oversampling(bankdata,␣
 ↪'reponse_campagne_actuelle_binaire', 'niveau_etudes')

# Affichage des proportions de niveaux d'études avant et après l'oversampling␣
 ↪stratifié
```

```
print("Proportions avant l'oversampling stratifié :")
print(bankdata['niveau_etudes'].value_counts(normalize=True))

print("\nProportions après l'oversampling stratifié :")
print(oversampled_bankdata['niveau_etudes'].value_counts(normalize=True))
```

```
Proportions avant l'oversampling stratifié :
secondary    0.537034
tertiary     0.303127
primary      0.159839
Name: niveau_etudes, dtype: float64

Proportions après l'oversampling stratifié :
secondary    0.542898
tertiary     0.291890
primary      0.165212
Name: niveau_etudes, dtype: float64
```

[ ]: