18 Janvier, 2025

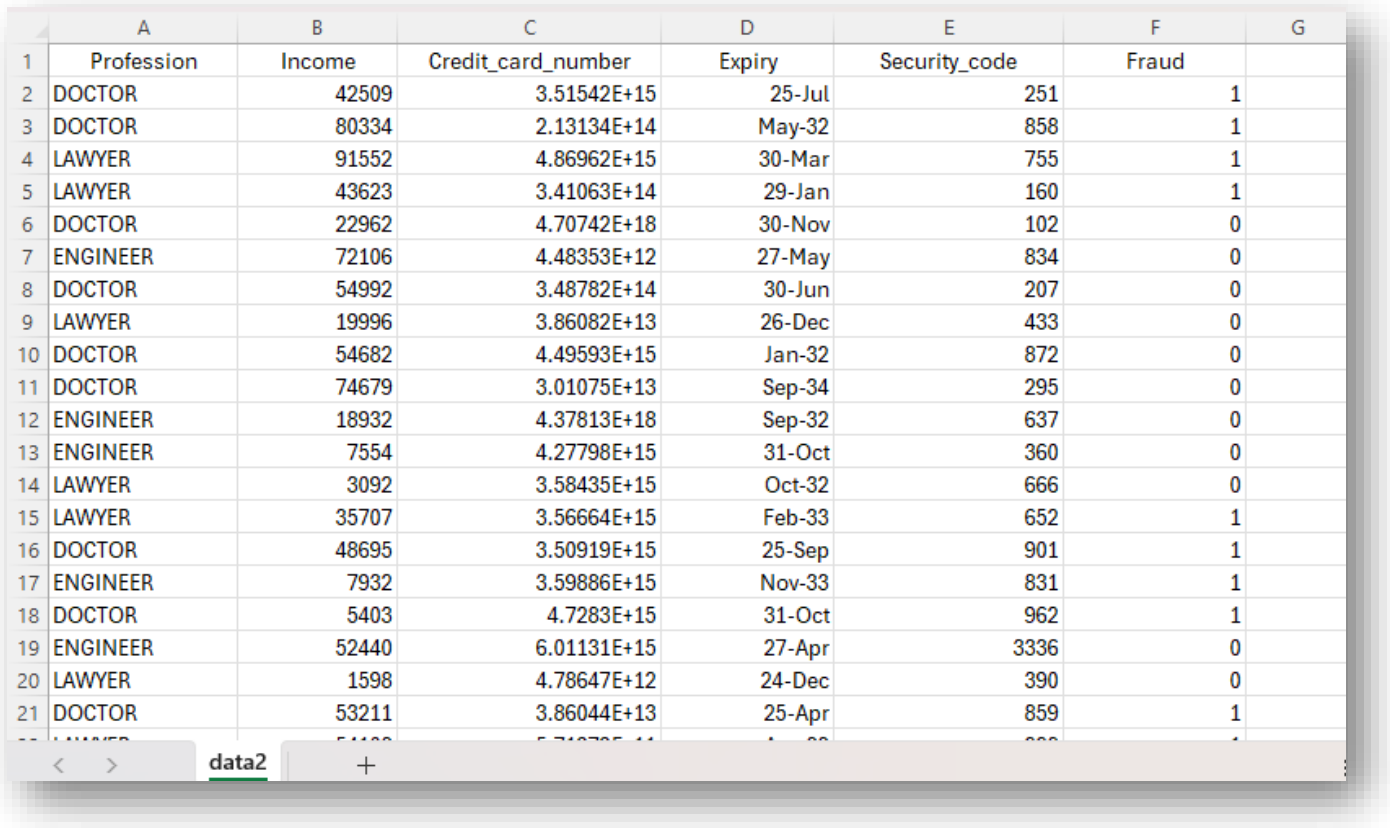
Data Mining

Mlle. Chaima Rouita
Mlle. Latifa Didi Alaoui
Mlle. Nouhayla Elassouli
Mlle. Nihal Tizgha

Introduction du projet :

Ce projet a pour objectif de détecter les transactions frauduleuses à l'aide de techniques avancées de data mining et d'apprentissage automatique. Le travail s'est articulé autour de l'exploration des données, du prétraitement, et de l'extraction de caractéristiques textuelles avec TF-IDF. Ensuite, des approches de sélection de caractéristiques basées sur des méthodes embarquées comme Lasso et l'importance des caractéristiques de Random Forest ont été utilisées pour identifier les attributs pertinents et améliorer la précision des prédictions, ce qui est essentiel pour une détection efficace de la fraude. Enfin, la classification a été effectuée à l'aide de modèles tels que SVM et KNN.

Description du Dataset :



	A	B	C	D	E	F	G
1	Profession	Income	Credit_card_number	Expiry	Security_code	Fraud	
2	DOCTOR	42509	3.51542E+15	25-Jul	251	1	
3	DOCTOR	80334	2.13134E+14	May-32	858	1	
4	LAWYER	91552	4.86962E+15	30-Mar	755	1	
5	LAWYER	43623	3.41063E+14	29-Jan	160	1	
6	DOCTOR	22962	4.70742E+18	30-Nov	102	0	
7	ENGINEER	72106	4.48353E+12	27-May	834	0	
8	DOCTOR	54992	3.48782E+14	30-Jun	207	0	
9	LAWYER	19996	3.86082E+13	26-Dec	433	0	
10	DOCTOR	54682	4.49593E+15	Jan-32	872	0	
11	DOCTOR	74679	3.01075E+13	Sep-34	295	0	
12	ENGINEER	18932	4.37813E+18	Sep-32	637	0	
13	ENGINEER	7554	4.27798E+15	31-Oct	360	0	
14	LAWYER	3092	3.58435E+15	Oct-32	666	0	
15	LAWYER	35707	3.56664E+15	Feb-33	652	1	
16	DOCTOR	48695	3.50919E+15	25-Sep	901	1	
17	ENGINEER	7932	3.59886E+15	Nov-33	831	1	
18	DOCTOR	5403	4.7283E+15	31-Oct	962	1	
19	ENGINEER	52440	6.01131E+15	27-Apr	3336	0	
20	LAWYER	1598	4.78647E+12	24-Dec	390	0	
21	DOCTOR	53211	3.86044E+13	25-Apr	859	1	

data2

1. Structure des Données

Le dataset contient les colonnes suivantes :

- **Profession** : Colonne catégorielle indiquant la profession de l'individu (DOCTOR, ENGINEER, LAWYER).
- **Income** : Revenu annuel de l'individu (valeurs numériques).
- **Credit_card_number** : Numéro de carte de crédit (valeur sensible non utilisée dans l'analyse).
- **Expiry** : Date d'expiration de la carte de crédit.
- **Security_code** : Code de sécurité de la carte (numérique).
- **Fraud** : Colonne cible indiquant si une transaction est frauduleuse (1) ou légitime (0).

2. Statistiques Descriptives

- Taille : 10 000 lignes et 6 colonnes.
- Données présentes : Aucune valeur manquante.
- Types : Combinaison de colonnes catégoriques (Profession), numériques (Income, Security_code) et sensibles.

Etapes du Projet :

1. Prétraitement des Données

1.1 Nettoyage des Données

- **Colonnes supprimées** : Les colonnes Credit_card_number et Expiry ont été supprimées car elles ne contribuent pas directement à l'analyse ou à la modélisation.
    - Exemple avant suppression :

Credit_card_number	Expiry	Income
3515418493460774	07/25	42509
213134223583196	05/32	80334

- Exemple après suppression:

Income
42509
80334

1.2 Encodage des Données Catégoriques

- La colonne Profession a été encodée en valeurs numériques à l'aide de **LabelEncoder** :
  - DOCTOR → 0
  - ENGINEER → 1
  - LAWYER → 2

1.3 Normalisation des Données Numériques

- Les colonnes Income et Security_code ont été normalisées à l'aide de **MinMaxScaler** pour les ramener dans une plage de valeurs entre 0 et 1.
- La normalisation a été nécessaire pour éviter que les échelles différentes des colonnes numériques n'influencent de manière disproportionnée les modèles d'apprentissage automatique, garantissant ainsi une meilleure convergence et une performance optimale.

2. Extraction de Caractéristiques Textuelles (TF-IDF)

2.1 Ajout d'une Colonne Textuelle

Pour appliquer l'extraction de caractéristiques TF-IDF, une colonne textuelle intitulée Description a été ajoutée au dataset. Voici quelques exemples de valeurs :

- "Doctor with high income"
- "Engineer specialized in software"
- "Lawyer focused on corporate law"

Cette colonne a été générée pour correspondre à la longueur totale du dataset.

[15]:
```
# Generate a description column programmatically
descriptions = [
    "Doctor with high income",
    "Engineer specialized in software",
    "Lawyer focused on corporate law",
    "Engineer in mechanical systems",
    "Doctor working in cardiology",
    "Engineer in AI research",
    "Lawyer specializing in criminal law",
    "Doctor with expertise in pediatrics",
    "Engineer handling robotics projects",
    "Lawyer working in public defense"
]

# Repeat descriptions to match the dataset length
data['Description'] = descriptions * (len(data) // len(descriptions)) + descriptions[:len(data) % len(descriptions)]

# Check the updated dataset
print(data.head())
```

```
   Profession    Income  Security_code  Fraud  \
0           0  0.425144       0.025125      1
1           0  0.803451       0.085886      1
2           2  0.915647       0.075576      1
3           2  0.436285       0.016016      1
```

2.2 Application de TF-IDF

- Extraction des 100 principales caractéristiques textuelles avec **TfidfVectorizer**.
- Les valeurs TF-IDF ont été converties en colonnes numériques et ajoutées au dataset.
- colonne Description a été supprimée après l'extraction TF-IDF.

```python
# Apply TF-IDF to the new Description column
vectorizer = TfidfVectorizer(max_features=100)  # Extract top 100 features
tfidf_matrix = vectorizer.fit_transform(data['Description'])

# Convert TF-IDF matrix to a DataFrame
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=vectorizer.get_feature_names_out())

# Concatenate the TF-IDF DataFrame with the original dataset
data = pd.concat([data, tfidf_df], axis=1)

# Drop the original Description column if no longer needed
data = data.drop('Description', axis=1)

# Check the updated dataset
print(data.head())
```

```
   Profession    Income  Security_code  Fraud   ai  cardiology  corporate  \
0           0  0.425144       0.025125      1  0.0    0.000000   0.000000
1           0  0.803451       0.085886      1  0.0    0.000000   0.000000
2           2  0.915647       0.075576      1  0.0    0.000000   0.495685
3           2  0.436285       0.016016      1  0.0    0.000000   0.000000
4           0  0.229644       0.010210      0  0.0    0.668337   0.000000

   criminal  defense    doctor  ...  projects  public  research  robotics  \
0       0.0      0.0  0.380943  ...       0.0     0.0       0.0       0.0
1       0.0      0.0  0.000000  ...       0.0     0.0       0.0       0.0
2       0.0      0.0  0.000000  ...       0.0     0.0       0.0       0.0
3       0.0      0.0  0.000000  ...       0.0     0.0       0.0       0.0
```

2.3 Sélection des Caractéristiques

- Utilisation de **SelectKBest** avec le test du **Chi2** pour sélectionner les caractéristiques les plus pertinentes.

- Les caractéristiques finales incluent :
    - Profession, Income, Security_code (originales).
    - Plusieurs caractéristiques textuelles extraites par TF-IDF.

3. Sélection de Caractéristiques avec Approches Intégrées

3.1 Sélection de Caractéristiques avec Lasso

Lasso, ou Least Absolute Shrinkage and Selection Operator, est une méthode de régression qui permet à la fois la régularisation et la sélection de caractéristiques en pénalisant les coefficients des variables.

Méthodologie :

1. **Prétraitement des Données** :
    - Les données ont été préparées en séparant la variable cible Fraud et les caractéristiques (features).
2. **Séparation du Jeu de Données** :
    - Les données ont été divisées en ensembles d'entraînement et de test, avec 20 % des données utilisées pour les tests.

```
# Séparer le jeu de données en ensembles d'entraînement et de test
from sklearn.model_selection import train_test_split

#la variable cible est 'Fraud'
X = data.drop('Fraud', axis=1) # Features : Toutes les colonnes sauf 'Fraud'
y = data['Fraud']              # target

# Diviser les données
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

3. **Application de Lasso**:

    Plusieurs valeurs d'alpha ont été testées pour évaluer leur impact sur les

```
# 1. Sélection de caractéristiques avec Lasso
for alpha in [0.1, 0.01, 0.001]:
    lasso = Lasso(alpha=alpha)
    lasso.fit(X_train, y_train)
    print(f"Alpha: {alpha}, Coefficients: {lasso.coef_}")
lasso.fit(X_train, y_train)
```

```
Alpha: 0.1, Coefficients: [-0. -0.  0. -0. -0.  0.  0.  0. -0. -0.  0.  0. -0. -0.  0. -0.  0.  0.
  0.  0.  0. -0.  0. -0. -0. -0. -0.  0.  0.  0. -0.]
Alpha: 0.01, Coefficients: [-0. -0.  0. -0. -0.  0.  0.  0. -0. -0.  0.  0. -0. -0.  0. -0.  0.  0.
  0.  0.  0. -0.  0. -0. -0. -0. -0.  0.  0.  0. -0.]
Alpha: 0.001, Coefficients: [-7.06673753e-03 -0.00000000e+00  0.00000000e+00 -0.00000000e+00
 -2.90975924e-02  0.00000000e+00  0.00000000e+00  4.26375451e-03
 -0.00000000e+00 -0.00000000e+00  2.43425569e-02  0.00000000e+00
 -0.00000000e+00 -0.00000000e+00  0.00000000e+00 -0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  2.32084474e-05 -0.00000000e+00  2.72839631e-03 -0.00000000e+00
 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00]
```

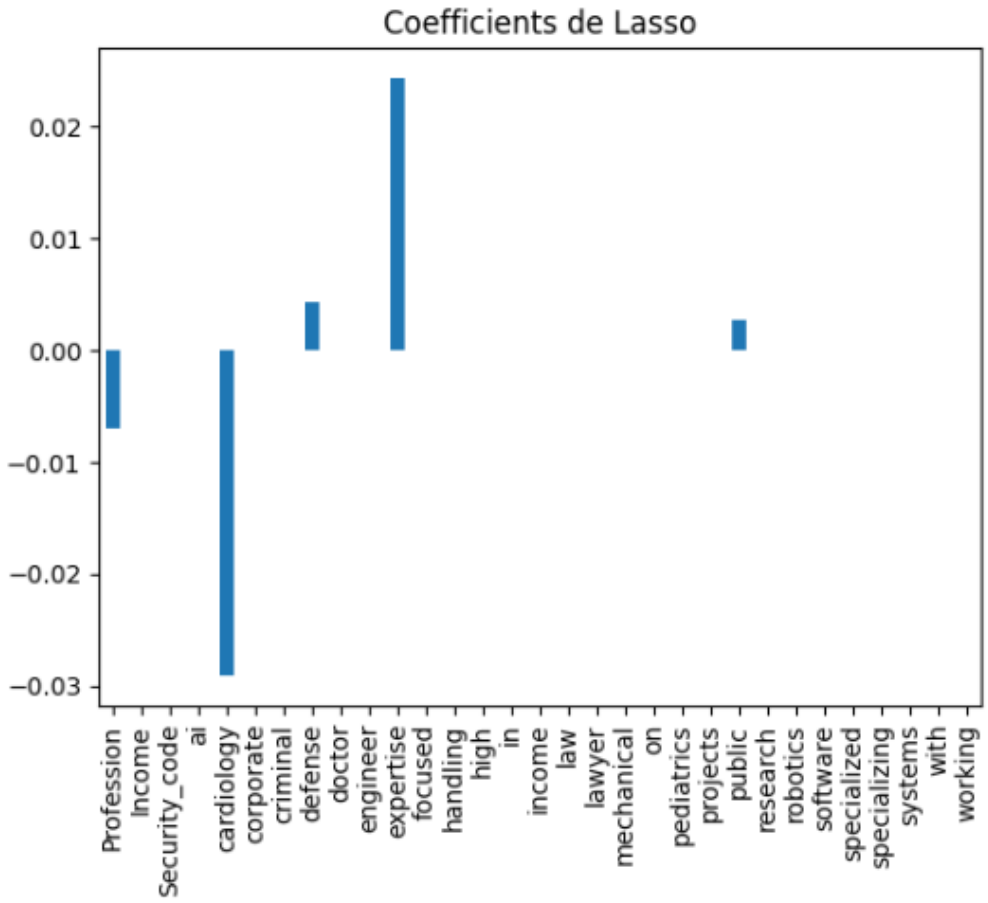
```
Lasso
Lasso(alpha=0.001)
```

    coefficients.

4. **Affichage des Coefficients et Sélection des Caractéristiques** :

Après avoir ajusté le modèle avec la meilleure valeur d'alpha, les coefficients ont été visualisés.

```python
# Afficher les coefficients de Lasso
lasso_coef = pd.Series(lasso.coef_, index=X.columns)
lasso_coef.plot(kind='bar', title='Coefficients de Lasso')
plt.show()
```




```python
selected_features = X.columns[lasso.coef_ != 0]
print("Caractéristiques sélectionnées :", selected_features)
```

```
Caractéristiques sélectionnées : Index(['Profession', 'cardiology', 'defense', 'expertise', 'pediatrics',
       'public'],
      dtype='object')
```

Interprétation du Résultat

1. **Coefficients de Lasso** :
   - Les coefficients de Lasso indiquent l'importance ou la contribution de chaque caractéristique à la prédiction de la variable cible (la fraude).

- Un coefficient positif signifie que lorsque la valeur de cette caractéristique augmente, la probabilité de fraude augmente également.
- Un coefficient négatif indique une relation inverse : lorsque la valeur de la caractéristique augmente, la probabilité de fraude diminue.

2. **Coefficients Nuls** :
   - Les caractéristiques dont les coefficients sont réduits à zéro par Lasso ne sont pas considérées comme pertinentes pour la prédiction. Cela signifie qu'elles n'apportent pas d'information utile dans le modèle.
   - Dans votre résultat, seules certaines caractéristiques ont des coefficients non nuls, ce qui indique qu'elles sont les seules jugées significatives par le modèle.
3. **Sélection des Caractéristiques** :
   - En affichant les caractéristiques avec des coefficients non nuls, vous obtenez une liste des caractéristiques qui contribuent le plus à la prédiction de la fraude.
   - Dans votre cas, les caractéristiques sélectionnées sont Profession, cardiology, defense, expertise, et pediatrics. Cela signifie que ces caractéristiques sont jugées importantes.

Visualisation

- Le graphique montre les coefficients de manière visuelle, ce qui facilite la comparaison entre les différentes caractéristiques.
- Les barres représentant les coefficients non nuls permettent d'identifier rapidement quelles caractéristiques ont une influence positive ou négative sur la variable cible.

3.2 Sélection de Caractéristiques avec Random Forest

Random Forest est un algorithme d'apprentissage supervisé qui utilise une approche d'ensemble pour améliorer la précision et la robustesse des prédictions. En plus de ses performances, il fournit une évaluation de l'importance des caractéristiques, ce qui est particulièrement utile pour la sélection de caractéristiques.

Méthodologie :

1. **Création et Ajustement du Modèle** :
    - Un modèle RandomForestClassifier a été créé avec 100 arbres pour capturer la complexité des données.

```
# 2. Sélection de caractéristiques avec Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
```

2. **Importance des Caractéristiques** :

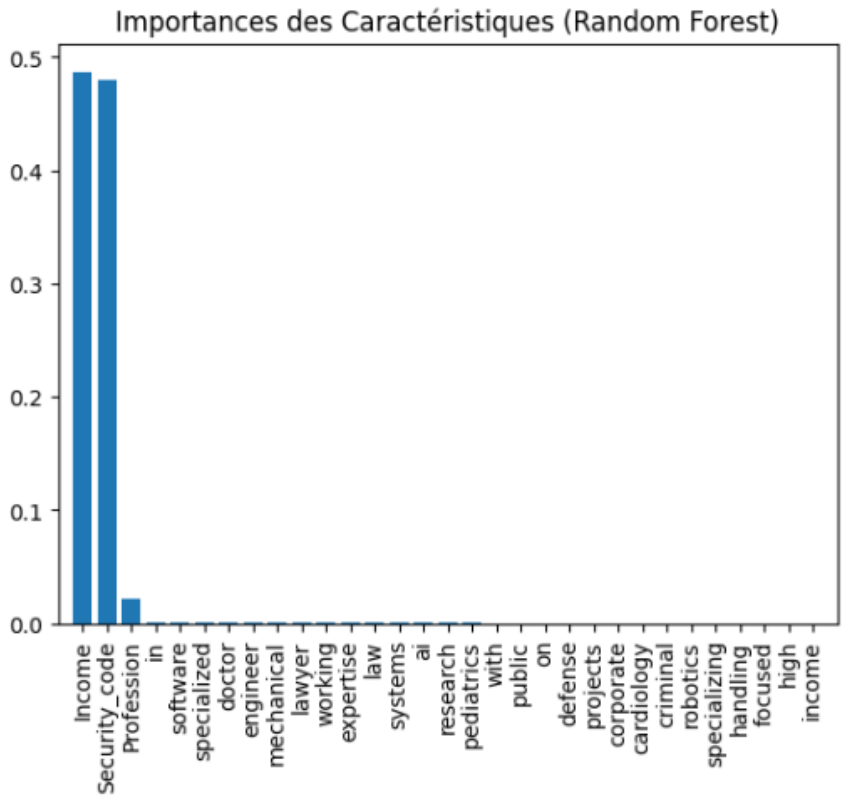
- Les importances des caractéristiques ont été extraites à l'aide de l'attribut feature_importances_.

```
# Importance des caractéristiques
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]
```

3. **Visualisation des Importances** :

- Un graphique a été créé pour visualiser l'importance de chaque caractéristique.

```
[44]: # Tracer l'importance des caractéristiques
      plt.figure()
      plt.title('Importances des Caractéristiques (Random Forest)')
      plt.bar(range(X.shape[1]), importances[indices], align='center')
      plt.xticks(range(X.shape[1]), X.columns[indices], rotation=90)
      plt.xlim([-1, X.shape[1]])
      plt.show()
```



4. **Affichage des Importances** :

- Les importances de chaque caractéristique ont été imprimées pour une analyse plus détaillée.

```python
#imprimer l'importance de chaque caractéristique
for i in range(X.shape[1]):
    print(f"Caractéristique : {X.columns[indices[i]]}, Importance : {importances[indices[i]]}")
```

```
Caractéristique : Income, Importance : 0.48700293154946833
Caractéristique : Security_code, Importance : 0.4792432944093339
Caractéristique : Profession, Importance : 0.02160745158733809
Caractéristique : in, Importance : 0.0009958226890070196
Caractéristique : software, Importance : 0.000818089317556792
Caractéristique : specialized, Importance : 0.0007269721741546642
Caractéristique : doctor, Importance : 0.0006831708532976759
Caractéristique : engineer, Importance : 0.0006585838011644094
Caractéristique : mechanical, Importance : 0.0006022373878096835
Caractéristique : lawyer, Importance : 0.0005707227974022451
Caractéristique : working, Importance : 0.0005671728833324043
Caractéristique : expertise, Importance : 0.0004665895931827831
Caractéristique : law, Importance : 0.00046524100112338705
Caractéristique : systems, Importance : 0.0004645144409247893
Caractéristique : ai, Importance : 0.00044278365449605683
Caractéristique : research, Importance : 0.00043030087350939855
Caractéristique : pediatrics, Importance : 0.0003937572674746261
Caractéristique : with, Importance : 0.0003828572155860046
Caractéristique : public, Importance : 0.00035332263753117845
Caractéristique : on, Importance : 0.0003168862229632247
Caractéristique : defense, Importance : 0.0003101219996742835
Caractéristique : projects, Importance : 0.0002957404753025154
Caractéristique : corporate, Importance : 0.00029547237847383373
Caractéristique : cardiology, Importance : 0.00029276152495937496
Caractéristique : criminal, Importance : 0.00027632088905091116
Caractéristique : robotics, Importance : 0.0002662988030606263
Caractéristique : specializing, Importance : 0.0002555446954546547
Caractéristique : handling, Importance : 0.00024366853666085372
Caractéristique : focused, Importance : 0.00021724645341544046
Caractéristique : high, Importance : 0.0001824055817127503
Caractéristique : income, Importance : 0.00017188132911979655
```

Interprétation des Résultats

1. **Importance des Caractéristiques** :
   - L'importance des caractéristiques dans une forêt aléatoire est mesurée par la façon dont chaque caractéristique contribue à réduire l'erreur lors de la prédiction. Plus la valeur d'importance est élevée, plus la caractéristique est considérée comme influente.
   - Les valeurs d'importance varient entre 0 et 1. Une valeur près de 1 signifie que la caractéristique est très importante, tandis qu'une valeur près de 0 indique qu'elle n'a que peu ou pas d'impact.
2. **Graphique d'Importance** :

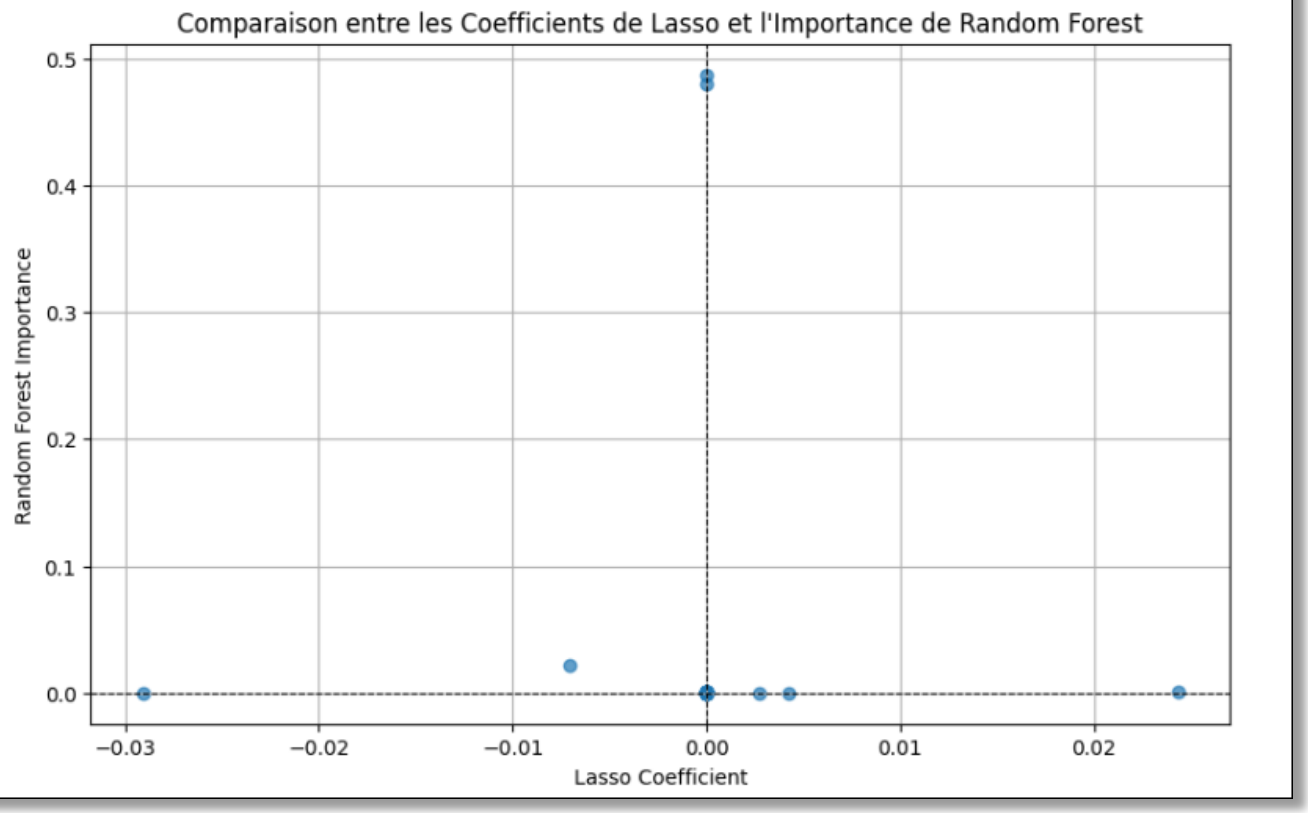
- Le graphique montre visuellement l'importance relative de chaque caractéristique. Les barres plus longues indiquent une plus grande importance.
- Dans votre cas, la majorité des caractéristiques semblent avoir une importance très faible ou nulle, ce qui pourrait indiquer que seules quelques caractéristiques sont réellement significatives pour le modèle.

3. **Liste des Caractéristiques** :
   - La liste montre chaque caractéristique avec son importance respective.
   - Les caractéristiques avec des valeurs d'importance élevées sont celles que vous devriez envisager de conserver pour vos modèles, tandis que celles avec des importances très faibles peuvent être considérées comme non pertinentes et potentiellement supprimées.

3.3 Comparaison des Résultats

1. **Graphique de Comparaison entre les Coefficients de Lasso et l'Importance de Random Forest**

```
# 3. Nuage de Points pour comparer Lasso et Random Forest
plt.figure(figsize=(10, 6))
plt.scatter(lasso_coef, importances, alpha=0.7)
plt.title('Comparaison entre les Coefficients de Lasso et l\'Importance de Random Forest')
plt.xlabel('Lasso Coefficient')
plt.ylabel('Random Forest Importance')
plt.axhline(0, color='black', linewidth=0.8, linestyle='--')
plt.axvline(0, color='black', linewidth=0.8, linestyle='--')
plt.grid()
plt.show()
```



Description du Graphique :

- Ce graphique scatterplot montre la relation entre les coefficients de Lasso (axe horizontal) et l'importance des caractéristiques selon la méthode Random Forest (axe vertical).
- Chaque point représente une caractéristique du modèle.

Interprétation :

- **Coefficients de Lasso** :
  - Les coefficients de Lasso peuvent être positifs ou négatifs. Un coefficient positif indique que la caractéristique augmente la probabilité de prédiction de la fraude, tandis qu'un coefficient négatif indique l'inverse.
- **Importance de Random Forest** :
  - Les valeurs d'importance proches de 0 signifient que la caractéristique n'a pas beaucoup d'impact sur la prédiction.

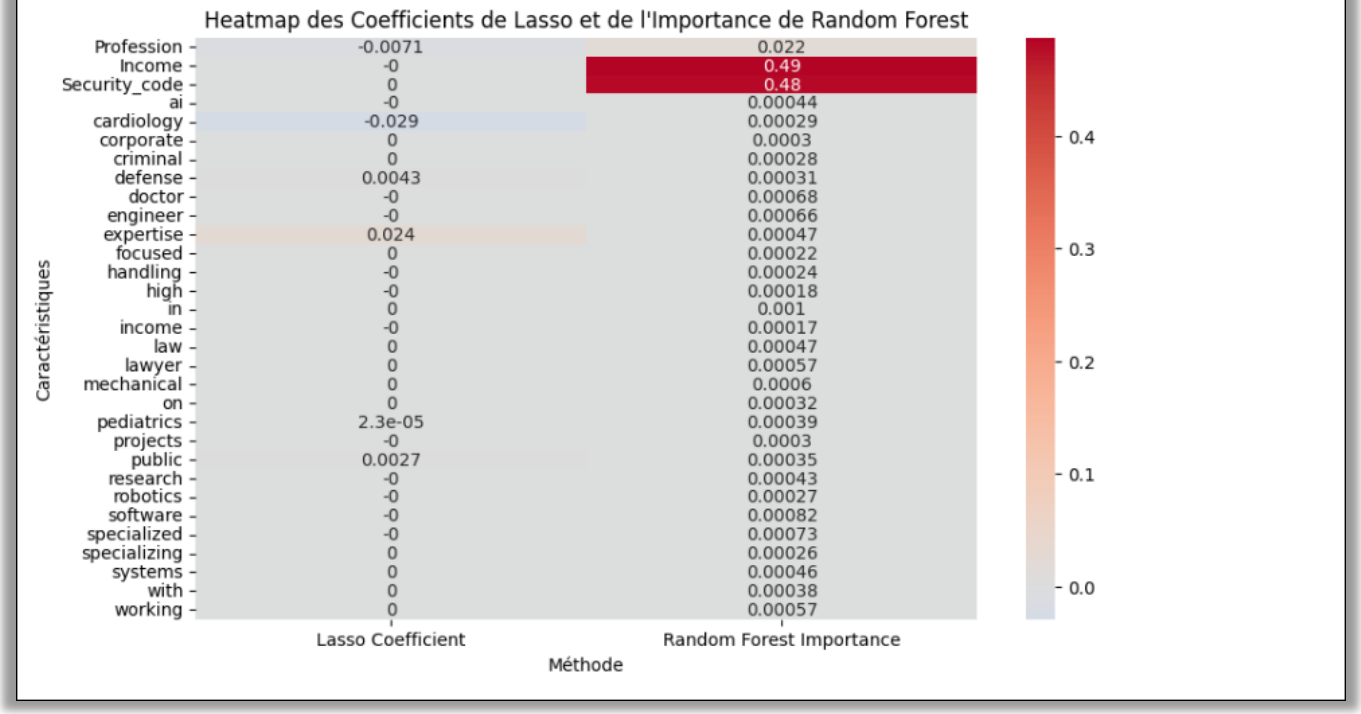
Observations :

- Les points situés dans la partie supérieure du graphique (valeurs d'importance élevées) correspondent aux caractéristiques jugées importantes par Random Forest. Si ces points ont également des coefficients de Lasso significatifs, cela indique une cohérence entre les deux méthodes.
- Les points en bas indiquent des caractéristiques que Random Forest considère comme peu importantes, même si elles ont des coefficients de Lasso, ce qui peut suggérer qu'elles ne sont pas de bons prédicteurs pour la fraude, malgré leur présence dans le modèle Lasso.

2. Heatmap des Coefficients de Lasso et de l'Importance de Random Forest

```python
# 4. Heatmap des Coefficients et Importances
comparison_df = pd.DataFrame({
    'Lasso Coefficient': lasso_coef,
    'Random Forest Importance': importances
})

plt.figure(figsize=(10, 6))
sns.heatmap(comparison_df, annot=True, cmap='coolwarm', center=0)
plt.title('Heatmap des Coefficients de Lasso et de l\'Importance de Random Forest')
plt.xlabel('Méthode')
plt.ylabel('Caractéristiques')
plt.show()
```



Description du Graphique :

- Cette heatmap représente les coefficients de Lasso et l'importance des caractéristiques selon Random Forest en utilisant un code couleur pour visualiser les valeurs.
- Les caractéristiques sont listées sur l'axe vertical, tandis que les méthodes (Lasso et Random Forest) sont affichées sur l'axe horizontal.

Interprétation :

- **Couleurs** :
  - Les couleurs plus foncées indiquent des valeurs plus élevées, tandis que les couleurs plus claires représentent des valeurs plus faibles.

- **Coefficients de Lasso** :
  - Les valeurs positives et négatives des coefficients sont facilement visualisées, ce qui permet de voir rapidement quelles caractéristiques ont un impact positif ou négatif sur la prédiction.
- **Importance de Random Forest** :
  - Les valeurs d'importance sont également affichées, permettant une comparaison directe avec les coefficients de Lasso.

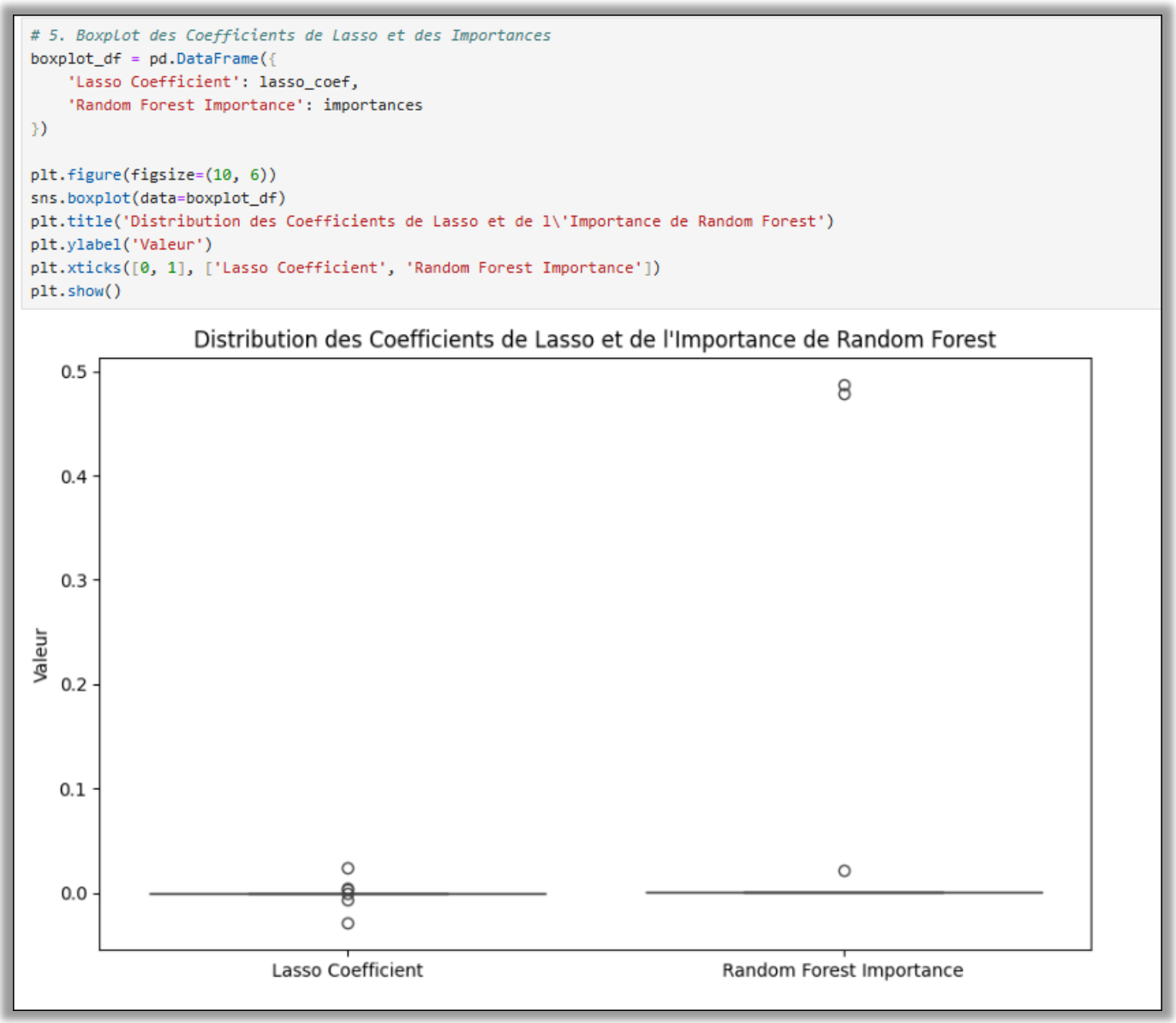
Observations :

- Les caractéristiques avec des coefficients de Lasso significatifs (en rouge) peuvent être facilement identifiées, tout comme celles qui ont une importance élevée dans Random Forest (en bleu).
- Si une caractéristique a un coefficient de Lasso élevé mais une faible importance dans Random Forest, cela peut indiquer que, bien qu'elle soit statistiquement significative, elle ne contribue pas réellement à la performance prédictive lorsque l'on considère des interactions complexes dans les données (ce qui est le cas avec Random Forest).
- En revanche, les caractéristiques avec des importances élevées dans Random Forest, mais des coefficients de Lasso faibles, peuvent indiquer que leur contribution dépend fortement des interactions avec d'autres caractéristiques.

3. Graphique de Distribution des Coefficients de Lasso et de l'Importance de Random Forest

```python
# 5. Boxplot des Coefficients de Lasso et des Importances
boxplot_df = pd.DataFrame({
    'Lasso Coefficient': lasso_coef,
    'Random Forest Importance': importances
})

plt.figure(figsize=(10, 6))
sns.boxplot(data=boxplot_df)
plt.title('Distribution des Coefficients de Lasso et de l\'Importance de Random Forest')
plt.ylabel('Valeur')
plt.xticks([0, 1], ['Lasso Coefficient', 'Random Forest Importance'])
plt.show()
```



Description du Graphique :

- Ce graphique représente la distribution des coefficients de Lasso et l'importance des caractéristiques selon Random Forest.
- Les valeurs sont placées sur l'axe vertical, avec des boîtes pour chaque méthode.

Interprétation :

- Distribution des Coefficients de Lasso :

    - Les boîtes montrent la médiane et l'étendue interquartile des coefficients, ce qui permet d'apprécier la variabilité des coefficients estimés par Lasso.
    - Des points au-delà des moustaches peuvent indiquer des valeurs aberrantes.
- Importance de Random Forest :
    - La boîte pour l'importance de Random Forest montre également la médiane et l'étendue, ce qui permet de voir comment les valeurs d'importance se répartissent.

Observations :

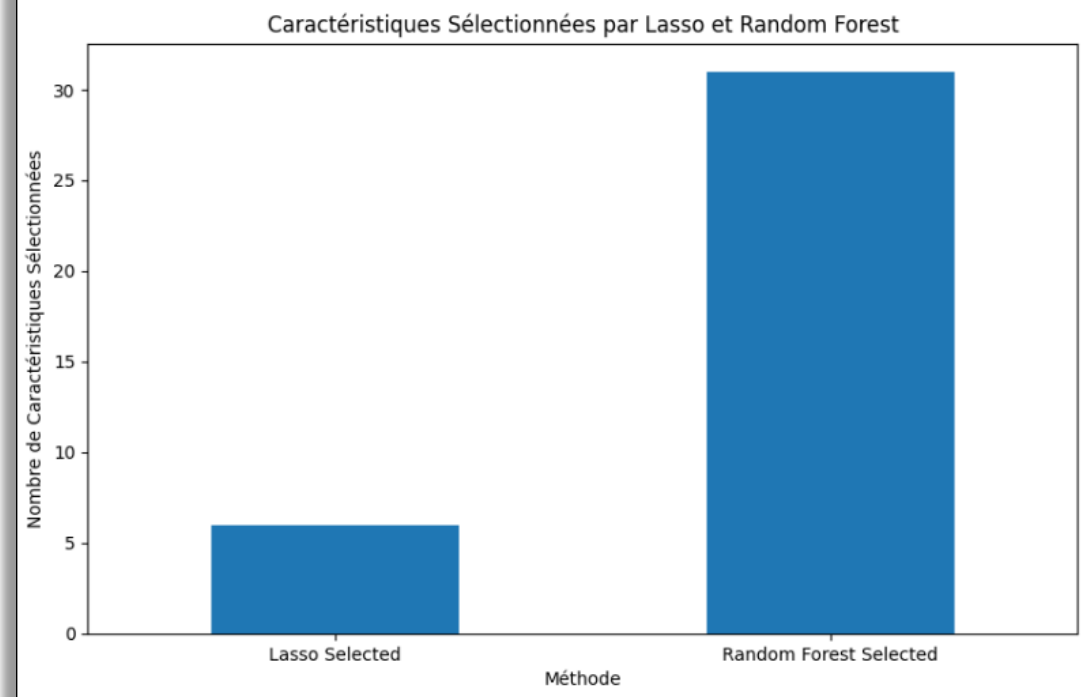
- Si la médiane des coefficients de Lasso est proche de zéro, cela peut indiquer que très peu de caractéristiques ont un impact significatif sur le modèle.
- En revanche, si les valeurs d'importance de Random Forest sont plus élevées et étendues, cela peut suggérer que plusieurs caractéristiques sont jugées importantes, même si elles n'apparaissent pas comme significatives dans le modèle Lasso.
- La comparaison entre les deux distributions peut aider à identifier des caractéristiques qui sont importantes dans un modèle tout en étant peut-être sous-estimées par Lasso.

4. Graphique des Caractéristiques Sélectionnées par Lasso et Random Forest

```python
# 6. Graphique de Barres Empilées pour les Caractéristiques Sélectionnées
lasso_selected = lasso_coef != 0
rf_selected = importances > 0

selection_df = pd.DataFrame({
    'Lasso Selected': lasso_selected,
    'Random Forest Selected': rf_selected
})

plt.figure(figsize=(10, 6))
selection_df.sum().plot(kind='bar', stacked=True)
plt.title('Caractéristiques Sélectionnées par Lasso et Random Forest')
plt.ylabel('Nombre de Caractéristiques Sélectionnées')
plt.xlabel('Méthode')
plt.xticks(rotation=0)
plt.show()
```



Description du Graphique :

- Ce graphique à barres montre le nombre de caractéristiques sélectionnées par chaque méthode : Lasso et Random Forest.

Interprétation :

- **Lasso Selected** :
  - Le nombre de caractéristiques sélectionnées par Lasso est généralement plus faible, ce qui est attendu car Lasso tend à réduire certains coefficients à zéro, favorisant ainsi la parcimonie.

- **Random Forest Selected** :
  - Le nombre de caractéristiques sélectionnées par Random Forest est nettement plus élevé. Cela est dû à la nature de l'algorithme, qui peut prendre en compte des interactions complexes et sélectionner plusieurs caractéristiques pertinentes.

Observations :

- La différence significative entre les deux barres souligne que Lasso favorise un modèle plus simple et plus interprétable, tandis que Random Forest peut capturer des relations plus complexes au prix de la simplicité.
- Cette information est cruciale pour la prise de décisions : si l'interprétabilité est une priorité, Lasso peut être le choix préféré. En revanche, si la précision est plus importante, Random Forest pourrait être plus approprié.

Explication de l'Accuracy

```python
from sklearn.metrics import accuracy_score

# 1. Précision du modèle Lasso
# Prédictions avec Lasso (doit être binaire, donc seuil de 0.5)
lasso_predictions = (lasso.predict(X_test) > 0.5).astype(int)
lasso_accuracy = accuracy_score(y_test, lasso_predictions)
print(f'Accuracy du modèle Lasso : {lasso_accuracy:.2f}')

# 2. Précision du modèle Random Forest
# Prédictions avec Random Forest
rf_predictions = rf.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print(f'Accuracy du modèle Random Forest : {rf_accuracy:.2f}')
```

```
Accuracy du modèle Lasso : 0.50
Accuracy du modèle Random Forest : 0.51
```

Accuracy (ou précision) est une métrique utilisée pour évaluer la performance d'un modèle de classification. Elle est définie comme la proportion de prédictions correctes parmi l'ensemble des prédictions effectuées. En d'autres termes, l'accuracy mesure combien de fois le modèle a correctement classé les instances par rapport au nombre total d'instances.
La formule pour calculer l'accuracy est :

Accuracy=Nombre de pre´dictions correctesNombre total de pre´dictions\text{Accuracy} = \frac{\text{Nombre de prédictions correctes}}{\text{Nombre total de prédictions}}Accuracy=Nombre total de pre´dictionsNombre de pre´dictions correctes

Résumé des Résultats

- **Accuracy du modèle Lasso : 0.50**
  - Cela signifie que le modèle Lasso a correctement classé 50% des instances de test. Autrement dit, il a eu autant de prédictions correctes que de prédictions incorrectes, ce qui suggère qu'il n'est pas meilleur qu'un tirage aléatoire pour ce problème.
- **Accuracy du modèle Random Forest : 0.51**
  - Le modèle Random Forest a un léger avantage avec une accuracy de 51%. Cela indique qu'il a correctement classé 51% des instances, ce qui est légèrement supérieur à la chance, mais reste très proche de l'indécision.

Conclusion

Les deux modèles, Lasso et Random Forest, montrent une performance d'accuracy relativement faible, proche de 50%. Cela implique que les modèles ne parviennent pas à capturer efficacement les relations dans les données pour prédire la fraude

4- le Modèle KNN

Étape 1 : Importer la bibliothèque nécessaire

- Importation de la classe KNeighborsClassifier de la bibliothèque scikit-learn, utilisée pour implémenter l'algorithme KNN.

Étape 2 : Initialisation et test de différentes valeurs de k

- Définition de plusieurs valeurs de k à tester : [3, 5, 7].
- Initialisation des variables best_k et best_accuracy pour stocker la meilleure valeur de k et sa précision associée

```
[ ]: #partie knn

[44]: # Importer la bibliothèque nécessaire pour KNN
      from sklearn.neighbors import KNeighborsClassifier

      # 1. Initialisation du modèle KNN
      k_values = [3, 5, 7]  # Différents nombres de voisins à tester
      best_k = None
      best_accuracy = 0
```

Étape 3 : Boucle pour tester chaque valeur de k

- Pour chaque valeur de k :
    1. Création d'un modèle KNN avec n_neighbors=k.
    2. Entraînement du modèle avec les données d'entraînement X_train et y_train.
    3. Prédiction des classes sur les données de test X_test.
    4. Calcul de la précision du modèle en comparant les prédictions avec les classes réelles y_test.
    5. Comparaison de la précision avec la meilleure précision obtenue jusqu'à présent et mise à jour des variables best_k et best_accuracy si nécessaire.

```
# Tester plusieurs valeurs de k pour trouver la meilleure
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)  # Entraînement du modèle
    knn_predictions = knn.predict(X_test)  # Prédictions
    knn_accuracy = accuracy_score(y_test, knn_predictions)  # Précision du modèle

    print(f'Accuracy pour k={k}: {knn_accuracy:.2f}')
    if knn_accuracy > best_accuracy:
        best_k = k
        best_accuracy = knn_accuracy
```

Étape 4 : Afficher la meilleure valeur de k

Étape 5 : Entraîner le modèle avec la meilleure valeur de k

- Création et entraînement du modèle final avec la valeur optimale de k trouvée à l'étape précédente.

```
# Afficher le meilleur k
print(f'Le meilleur k est {best_k} avec une précision de {best_accuracy:.2f}')

# 2. Entraîner le modèle avec le meilleur k
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
```

Étape 6 : Prédictions finales

- Utilisation du modèle entraîné pour effectuer des prédictions sur les données de test.

Étape 7 : Évaluation du modèle

```python
# 3. Prédictions finales
knn_predictions = knn.predict(X_test)

# 4. Évaluation du modèle
print("Rapport de classification (KNN) :")
print(classification_report(y_test, knn_predictions))
```

Étape 8 : Visualisation de la matrice de confusion

- Calcul de la matrice de confusion, qui montre le nombre de prédictions correctes et incorrectes pour chaque classe.
- Visualisation de cette matrice sous forme de heatmap pour une meilleure compréhension.

Étape 9 : Comparer avec d'autres modèles

- Calcul et affichage de la précision finale du modèle KNN pour évaluer ses performances globales.

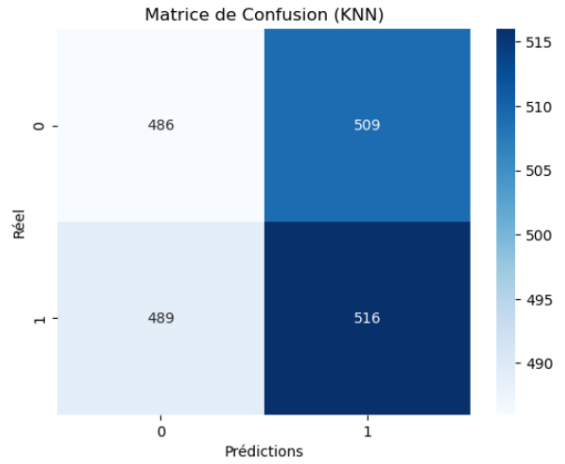
```python
# 5. Matrice de confusion
conf_matrix = confusion_matrix(y_test, knn_predictions)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Matrice de Confusion (KNN)')
plt.xlabel('Prédictions')
plt.ylabel('Réel')
plt.show()

# Comparer avec d'autres modèles
knn_accuracy = accuracy_score(y_test, knn_predictions)
print(f'Accuracy du modèle KNN : {knn_accuracy:.2f}')
```

```
Accuracy pour k=3: 0.50
Accuracy pour k=5: 0.50
Accuracy pour k=7: 0.49
Le meilleur k est 3 avec une précision de 0.50
Rapport de classification (KNN) :
              precision    recall  f1-score   support

           0       0.50      0.49      0.49       995
           1       0.50      0.51      0.51      1005

    accuracy                           0.50      2000
   macro avg       0.50      0.50      0.50      2000
weighted avg       0.50      0.50      0.50      2000
```




```
Accuracy du modèle KNN : 0.50
```

1. Analyse des précisions pour différentes valeurs de k

- Accuracy pour k=3 : 0.50

  Le modèle a une précision de 50 %, ce qui signifie qu'il prédit correctement 50 % des exemples de test.
- Accuracy pour k=5 : 0.50

  Avec 5 voisins, la précision reste inchangée à 50 %.
- Accuracy pour k=7 : 0.49

  En augmentant k à 7, la précision diminue légèrement à 49 %. Cela peut indiquer que l'inclusion de plus de voisins a introduit du bruit, réduisant la capacité du modèle à bien généraliser.
- Meilleur k

  La meilleure précision (50 %) a été obtenue pour k=3. Cette valeur a donc été retenue pour entraîner le modèle final.

2. Rapport de classification

Le rapport de classification fournit des métriques clés pour évaluer les performances du modèle :

- Classes :
  - Classe 0 (support : 995) :
    - *Precision (0.50)* : 50 % des prédictions pour la classe 0 sont correctes.
    - *Recall (0.49)* : Le modèle détecte 49 % des exemples appartenant réellement à la classe 0.
    - *F1-score (0.49)* : Moyenne harmonique entre précision et rappel.
  - Classe 1 (support : 1005) :
    - *Precision (0.50)* : 50 % des prédictions pour la classe 1 sont correctes.
    - *Recall (0.51)* : Le modèle détecte 51 % des exemples appartenant réellement à la classe 1.
    - *F1-score (0.51)* : Moyenne harmonique entre précision et rappel.
- Moyennes globales :
  - *Accuracy (0.50)* : Le modèle est correct dans 50 % des cas.
  - *Macro avg (0.50)* : Moyenne des métriques (précision, rappel, F1-score) calculées indépendamment pour chaque classe.
  - *Weighted avg (0.50)* : Moyenne pondérée par le nombre d'exemples dans chaque classe.

5- le Modèle SVM

1-Importation des bibliothèques nécessaires

- SVC : Importé depuis sklearn.svm, c'est le modèle SVM utilisé pour la classification.
- classification_report, confusion_matrix, accuracy_score : Fournissent des métriques pour évaluer les performances du modèle (précision, rappel, F1-score, etc.).

- matplotlib.pyplot et seaborn : Utilisés pour visualiser la matrice de confusion.

2. Initialisation et entraînement du modèle SVM

- **Initialisation :**
  - **kernel='linear'** : Spécifie l'utilisation d'un noyau linéaire pour séparer les données.
  - **C=1.0** : Paramètre de régularisation ; une valeur plus grande réduit les marges pour mieux séparer les données mais peut conduire à un sur-apprentissage.
  - **random_state=42** : Fixe un état aléatoire pour assurer la reproductibilité.
- **Entraînement :**
  - **fit(X_train, y_train)** : Entraîne le modèle SVM sur les données d'entraînement (X_train et y_train).

```
[ ]: #partie svm

[45]: from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns

# Initialiser et entraîner le modèle SVM
svm_model = SVC(kernel='linear', C=1.0, random_state=42)
svm_model.fit(X_train, y_train)
```

3. Prédictions sur le jeu de test

- Prédictions :
  - Le modèle prédit les classes des données de test (X_test) en fonction des connaissances acquises lors de l'entraînement.

4. Évaluation des performances

- Précision globale (accuracy) :
  - Calculée avec accuracy_score(y_test, svm_predictions), cette métrique mesure la proportion d'exemples correctement classés par rapport au total.

- Résultat affiché sous forme d'un pourcentage.

```
# Prédire les résultats sur le jeu de test
svm_predictions = svm_model.predict(X_test)

# Évaluer les performances
svm_accuracy = accuracy_score(y_test, svm_predictions)
print(f'Accuracy du modèle SVM : {svm_accuracy:.2f}')
```

5. Rapport de classification

- Rapport de classification :
  - Fournit des métriques pour chaque classe (par exemple, Non Fraud et Fraud):
    - Précision (Precision) : Pourcentage des prédictions correctes par classe.
    - Rappel (Recall) : Pourcentage des exemples d'une classe bien détectés par le modèle.
    - F1-score : Moyenne harmonique entre précision et rappel, donnant une mesure équilibrée.
    - Support : Nombre d'exemples réels dans chaque classe.

6. Matrice de confusion

- Montre le nombre d'exemples correctement ou incorrectement classés pour chaque classe.
- Structure :
  - Ligne = Classes réelles (Vérité terrain).
  - Colonne = Classes prédites.

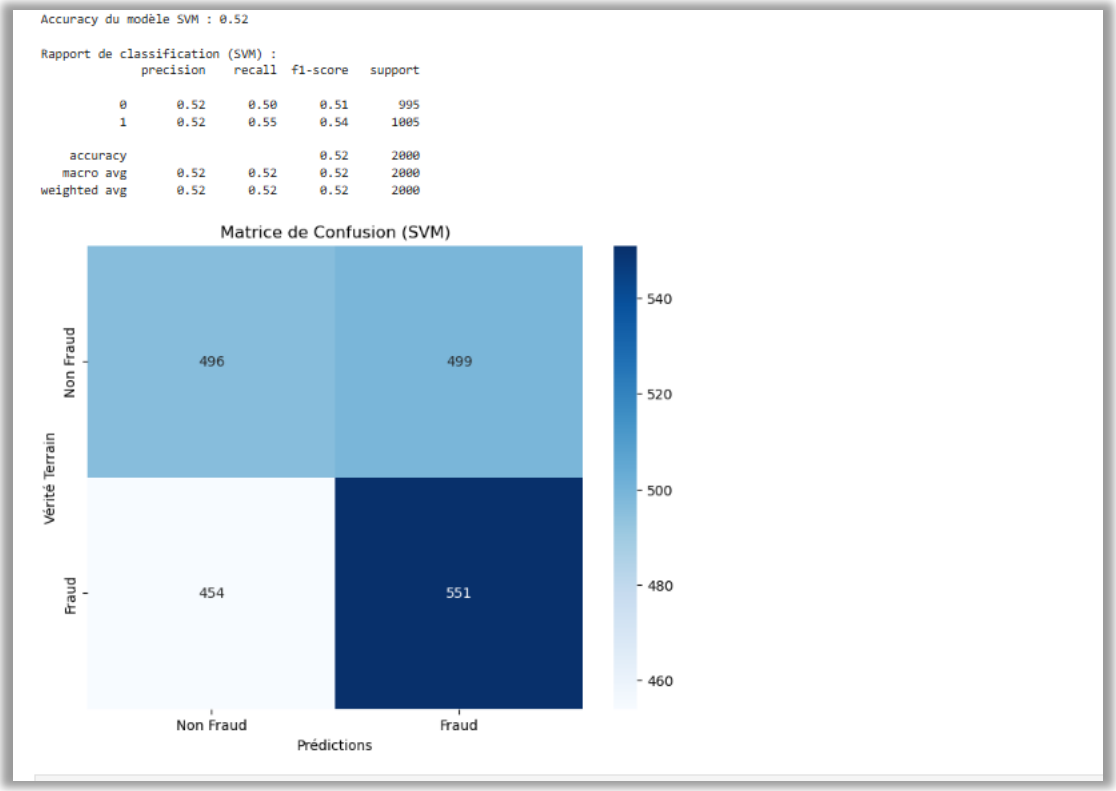
```python
# Rapport de classification
print("\nRapport de classification (SVM) :")
print(classification_report(y_test, svm_predictions))

# Matrice de confusion
conf_matrix = confusion_matrix(y_test, svm_predictions)
```

7. Visualisation de la matrice de confusion

- Visualisation graphique :
  - Utilise seaborn.heatmap pour afficher la matrice sous forme de tableau coloré.
  - annot=True : Affiche les valeurs numériques sur la matrice.
  - xticklabels et yticklabels : Étiquettes pour indiquer les classes (par exemple, Non Fraud et Fraud).
  - cmap='Blues' : Palette de couleurs pour la matrice.

```python
# Visualisation de la matrice de confusion
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Non Fraud', 'Fraud'], yticklabels=['Non Fraud', 'Fraud'])
plt.title('Matrice de Confusion (SVM)')
plt.xlabel('Prédictions')
plt.ylabel('Vérité Terrain')
plt.show()
```



1. Accuracy du modèle

- Accuracy globale : 0.52
    - Cela signifie que 52 % des exemples dans le jeu de test ont été correctement classés.
    - Bien que la précision dépasse le seuil aléatoire (50 % pour un problème binaire), elle reste faible. Cela peut indiquer que :
        - Les données ne sont pas entièrement linéairement séparables (le modèle utilise un noyau linéaire).
        - Les caractéristiques (features) peuvent manquer de pertinence ou nécessiter un prétraitement supplémentaire.

2. Rapport de classification

Classe 0 (Non Fraud) :

- Precision : 0.52
    - Sur toutes les prédictions faites pour la classe 0, 52 % étaient correctes.
- Recall : 0.50
    - Sur toutes les instances réelles de la classe 0, seulement 50 % ont été correctement identifiées.
- F1-score : 0.51
    - L'équilibre entre précision et rappel est modéré.

Classe 1 (Fraud) :

- Precision : 0.52
    - Sur toutes les prédictions faites pour la classe 1, 52 % étaient correctes.
- Recall : 0.55
    - Sur toutes les instances réelles de la classe 1, 55 % ont été correctement identifiées.
- F1-score : 0.54
    - Indique une performance légèrement meilleure pour la classe 1 par rapport à la classe 0.

3. Moyennes globales (Macro avg et Weighted avg)

- Macro avg :
    - Moyenne simple des métriques (précision, rappel, F1-score) pour les deux classes.
    - Toutes les classes sont traitées de manière égale, indépendamment du nombre d'exemples par classe.
    - Valeur : 0.52 (indiquant une performance modérée).
- Weighted avg :
    - Moyenne pondérée par le nombre d'exemples dans chaque classe.

- Utilise le support de chaque classe pour accorder plus de poids à celle contenant davantage d'exemples.
- Valeur : 0.52 (similaire à la macro moyenne, car les classes sont équilibrées en termes de support).