Analyse et prédiction des salaires des développeurs

Objectif du projet :

Ce projet vise à analyser un dataset issue de StackOverflow (2023) de développeurs et à prédire leur salaire annuel en USD à partir de leurs compétences, expériences et caractéristiques professionnelles.

Encadre par :

- Abderrahman BOUHAMIDI

Réalise par :

- Haitam Elasry
- Mohamed Issam EL KHAYATI

Table des matières

Importation des bibliothèques

Avant de commencer l'analyse, nous avons importé les bibliothèques essentielles du projet. Pandas et NumPy ont été utilisés pour la manipulation et le traitement des données. Matplotlib et Seaborn ont permis de réaliser des visualisations afin de mieux comprendre la distribution des variables du dataset StackOverflow. Enfin, la fonction *train_test_split* de scikit-learn a été importée pour préparer les jeux d'entraînement et de test nécessaires à la construction des modèles de prédiction des salaires.

Cette étape constitue la base de l'environnement d'analyse et de modélisation.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns
```

Import pandas as pd

- Pandas est utilisé pour manipuler les données.
- Il permet de charger des fichiers CSV, nettoyer les colonnes, filtrer et transformer le dataset.
- C'est la bibliothèque centrale pour toute analyse de données.

Import numpy as np

- NumPy sert à effectuer des opérations mathématiques et statistiques avancées.
- Souvent utilisé en support de pandas, notamment pour manipuler des tableaux numériques.

Import matplotlib.pyplot as plt

- Matplotlib est utilisé pour créer des graphiques (histogrammes, boxplots, courbes…).
- Il sert à visualiser la distribution du salaire et d'autres variables du dataset.

from sklearn.model_selection import train_test_split

- Importation d'un outil essentiel de scikit-learn pour créer les jeux :
  - D'entraînement (train) ○
    De test (test)
- Il permet une séparation aléatoire afin d'évaluer les performances des modèles de prédiction.

Import seaborn as sns

- Seaborn est une bibliothèque de visualisation basée sur matplotlib.
- Elle permet de créer des graphiques plus esthétiques et adaptés à l'analyse statistique.
- Par exemple : heatmaps, pairplots, boxplots avancés.

Aperçue de dataset

Le dataset utilisé provient de l'enquête annuelle StackOverflow 2023 et comprend 48 019 observations réparties sur 33 colonnes. Chaque ligne correspond à un développeur ayant répondu au questionnaire, tandis que les colonnes représentent ses caractéristiques professionnelles : compétences techniques, pays de résidence, années d'expérience, rôle occupé, technologies utilisées, type d'entreprise, entre autres.

La majorité des variables présentes sont de type *object*, c'est-à-dire des données textuelles ou catégorielles. Cependant, ce type de données n'est pas directement compatible avec les algorithmes de machine learning, qui nécessitent des variables numériques comme entrée. Il est donc indispensable de réaliser une phase de prétraitement incluant le nettoyage, la normalisation et surtout l'encodage des catégories afin de les transformer en données exploitables par les modèles.

La variable cible de ce projet est **ConvertedCompYearly**, représentant le salaire annuel du répondant en dollars américains (USD). Il s'agit d'une variable continue, ce qui définit clairement notre problématique comme un problème de régression.

Cet aperçu général du dataset met en évidence plusieurs enjeux majeurs :
(1) **la gestion et la transformation d'un volume important de données textuelles,**
(2) **la nécessité d'une analyse exploratoire approfondie pour comprendre les relations entre les variables,**
(3) **la mise en place de modèles prédictifs capables d'estimer efficacement le salaire annuel des développeurs à partir de leurs caractéristiques.**

Chargement et inspection du dataset

Le dataset a été importé à l'aide de la fonction read_csv() de la bibliothèque Pandas, puis sa structure générale a été examinée via la méthode df.info(). Le fichier contient 48 019 observations réparties sur 33 colonnes. La majorité des variables sont de type object, ce qui indique qu'elles contiennent principalement des données textuelles ou catégorielles.

Cette première inspection met également en évidence la présence de valeurs manquantes dans plusieurs colonnes, ainsi que l'importance de la mémoire occupée par le DataFrame. Ces éléments soulignent la nécessité de réaliser une phase de prétraitement incluant le nettoyage des données, l'imputation des valeurs manquantes et l'encodage des variables catégorielles.

Cette étape constitue la base du pipeline d'analyse, car elle permet de comprendre la structure du dataset et d'anticiper les traitements nécessaires pour la modélisation du salaire annuel des développeurs.

Analyse des valeurs manquantes

```
missing_pct = (df.isnull().sum() / len(df) * 100).sort_values(ascending= False)
print('missing percentage values >50 : \n', missing_pct[missing_pct>50])
print('total columns wih missing values percentage >50 : \n', (missing_pct>50).sum())
```

Afin d'évaluer la qualité du dataset, nous avons calculé le pourcentage de valeurs manquantes pour chacune des 33 colonnes. Pour cela, nous avons divisé le nombre de valeurs nulles par le nombre total d'observations, puis trié les colonnes par importance décroissante.

L'objectif est d'identifier les variables présentant un taux élevé de données manquantes, en particulier celles dépassant **le seuil de 50 %. Ces colonnes sont trop incomplètes pour être correctement exploitées ou imputées. Leur maintien risque d'introduire du bruit dans les modèles et de diminuer leur performance.**

Le calcul a également permis de déterminer le nombre total de colonnes dépassant ce seuil critique. Cette étape s'inscrit dans le processus de nettoyage du dataset et oriente les décisions sur les variables à conserver ou à supprimer avant la phase d'encodage et de modélisation.

Suppression des colonnes incomplètes ou non pertinentes

```
cols_to_drop = missing_pct[missing_pct>50].index.tolist()

cols_to_drop.extend([
    'LearnCodeOnline', #less critical cols
    'OpSysPersonal use', #less critical cols
    'ICorPM', #meta cols
    'ProfessionalTech',#meta cols
    'TBranch',#meta cols
    'AISearchHaveWorkedWith',#high missing percentage
    'MiscTechHaveWorkedWith',#high missing percentage
    'OfficeStackAsyncHaveWorkedWith',#office tools less relevant
    'OfficeStackSyncHaveWorkedWith',#office tools less relevant
    'NEWCollabToolsHaveWorkedWith'#office tools less relevant
])

cols_to_drop = list(set(cols_to_drop))
df_cleaned = df.drop(columns= cols_to_drop )

print( len(cols_to_drop))
print( df_cleaned.shape[1])
print(df_cleaned.columns)
```

À partir du calcul des valeurs manquantes, nous avons identifié les colonnes présentant plus de 50 % de données manquantes. Ces variables étant trop incomplètes pour être imputées de manière fiable, elles ont été automatiquement ajoutées à une liste de colonnes à supprimer.

À cela s'ajoutent plusieurs colonnes retirées manuellement. Certaines d'entre elles étaient peu pertinentes pour l'objectif prédictif (ex. méthodes d'apprentissage en ligne, outils bureautiques), tandis que d'autres correspondaient à des métadonnées ou à des variables très faiblement renseignées.

Après déduplication, l'ensemble de ces colonnes a été supprimé afin d'obtenir un dataset plus propre et plus exploitable. **Cette réduction permet d'améliorer la qualité future du modèle, d'éviter l'introduction de bruit et de simplifier les étapes d'encodage et de traitement**. Le DataFrame obtenu, nommé df_cleaned, constitue désormais la base des analyses et de la modélisation.

Aperçu des données après nettoyage

```
df_cleaned.head(3)
```

	MainBranch	Age	Employment	RemoteWork	EdLevel	LearnCode	YearsCode	YearsCodePro	DevType	OrgSize	...	LanguageHaveWorkedWith
0	I am a developer by profession	25-34 years old	Employed, full-time	Remote	Bachelor's degree (B.A., B.S., B.Eng., etc.)	Books / Physical media;Colleague;Friend or fam...	18	9	Senior Executive (C-Suite, VP, etc.)	2 to 9 employees	...	HTML/CSS;JavaScript;Python
1	I am a developer by profession	45-54 years old	Employed, full-time	Hybrid (some remote, some in-person)	Bachelor's degree (B.A., B.S., B.Eng., etc.)	Books / Physical media;Colleague;On the job tr...	27	23	Developer, back-end	5,000 to 9,999 employees	...	Bash/Shell (all shells);Go
2	I am a developer by profession	25-34 years old	Employed, full-time	Hybrid (some remote, some in-person)	Bachelor's degree (B.A., B.S., B.Eng., etc.)	Colleague;Friend or family member;Other online...	12	7	Developer, front-end	100 to 499 employees	...	Bash/Shell (all shells);HTML/CSS;JavaScript;PH...

Pour vérifier l'état du dataset après la suppression des colonnes incomplètes ou non pertinentes, nous avons affiché les trois premières lignes du DataFrame nettoyé à l'aide de la méthode df_cleaned.head(3).

Cet aperçu permet de confirmer que les colonnes problématiques ont bien été retirées et de visualiser la structure actuelle du dataset. **Les variables restantes se composent principalement de données professionnelles, démographiques et techniques, ainsi que de la variable cible ConvertedCompYearly.**

Cette étape constitue un contrôle visuel essentiel pour valider la cohérence du nettoyage effectué et préparer les prochaines phases du prétraitement, notamment l'encodage des variables catégorielles et la gestion des valeurs manquantes restantes.

Analyse des valeurs manquantes restantes

```
print('missing data remaining\n')
missing_remainder = (df_cleaned.isnull().sum() / len(df) * 100).sort_values(ascending= False)
print(missing_remainder[missing_remainder>0])
```

Après la suppression des colonnes fortement incomplètes, nous avons recalculé le pourcentage de valeurs manquantes dans le DataFrame nettoyé à l'aide de la méthode *df_cleaned.isnull()*. Les résultats montrent que certaines colonnes conservent encore un nombre limité de données manquantes.

L'identification de ces variables est essentielle car les modèles de machine learning ne peuvent pas traiter directement les valeurs nulles. Les colonnes affichant **un faible pourcentage de valeurs manquantes seront imputées à l'aide de techniques simples (comme la médiane ou le mode), tandis que celles présentant un pourcentage plus élevé feront l'objet d'un traitement spécifique.**

Cette étape permet de finaliser le diagnostic de qualité des données avant l'encodage et l'entraînement des modèles prédictifs. Elle garantit que le dataset final sera complet, cohérent et adapté aux exigences de la modélisation.

Analyse des types de données

```
print('\n Data types :')
print(df_cleaned.dtypes.value_counts())
```

Après le nettoyage des colonnes incomplètes et non pertinentes, nous avons examiné les types de données des variables restantes à l'aide de la méthode *df_cleaned.dtypes.value_counts()*.

Les résultats confirment que la majorité des colonnes sont de type *object*, c'est-à-dire qu'elles contiennent des données textuelles ou catégorielles. **Ces colonnes nécessitent un encodage approprié pour être exploitées par les modèles de machine learning.**

Les colonnes numériques restantes (*int64* ou *float64*) correspondent à des variables quantitatives comme l'expérience ou le salaire annuel. Ces colonnes peuvent être utilisées directement ou normalisées si nécessaire.

Analyse des colonnes clés et de leur diversité

```
key_columns = ['Country', 'DevType', 'EdLevel', 'Employment']

for col in key_columns:
    print(f"Number of unique values in '{col}':")
    print(len(df_cleaned[col].unique()))
```

Nous avons identifié quatre colonnes particulièrement importantes pour l'estimation du salaire : **Country (pays de résidence), DevType (type de développeur), EdLevel (niveau d'éducation) et Employment (statut professionnel).**

Pour chacune de ces variables, nous avons calculé le nombre de valeurs uniques. Cette étape permet de comprendre **la granularité de chaque colonne et de déterminer les stratégies d'encodage appropriées.**

Les colonnes à forte cardinalité, telles que Country ou DevType, nécessitent une attention particulière pour éviter la prolifération de variables après OneHotEncoding, tandis que les colonnes à faible cardinalité comme EdLevel et Employment peuvent être encodées directement.

Préparation du dataset pour la modélisation

```
df_model = df_cleaned.dropna().copy()

df_model.shape
```

Afin de disposer d'un dataset complet et exploitable pour les modèles de prédiction, nous avons supprimé toutes les lignes contenant des valeurs manquantes à l'aide de la méthode dropna(). Cette opération permet de garantir que toutes les observations utilisées pour l'entraînement et le test des modèles sont entièrement renseignées.

Le DataFrame résultant, nommé df_model, contient uniquement des lignes complètes, ce qui évite la nécessité d'imputation des valeurs manquantes. Cette étape constitue une préparation essentielle avant l'encodage des variables catégorielles et la séparation des données en variables explicatives et variable cible.

La taille finale du dataset, indiquée par df_model.shape, permet de s'assurer que le nombre d'observations restantes est suffisant pour entraîner des modèles robustes et fiables.

Évaluation de l'impact de la suppression des lignes manquantes

```
print(f"Number of rows retained: {df_model.shape[0]}")
percentage_dropped = ((len(df) - df_model.shape[0]) / len(df)) * 100
print(f"Percentage of rows dropped: {percentage_dropped:.2f}%")
```

Après le nettoyage et la sélection des variables pertinentes, 16 460 lignes ont été conservées pour l'analyse et la modélisation. Cela représente une réduction significative par rapport à l'ensemble initial, avec 65,72 % des lignes supprimées en raison de valeurs manquantes ou de données non exploitables.
Cette analyse permet de vérifier que le nettoyage du dataset n'a pas réduit de manière significative la taille des données et que la diversité des observations est préservée pour l'entraînement et

l'évaluation des modèles de prédiction. Elle justifie également la décision de supprimer les lignes plutôt que d'appliquer une imputation, dans la mesure où l'impact sur le dataset est limité.

Statistiques descriptives de la variable cible

```
# Revert ConvertedCompYearly in df_model to its original values from df_cleaned
df_model['ConvertedCompYearly'] = df_cleaned['ConvertedCompYearly'].loc[df_model.index]

print('Descriptive statistics for the reverted target variable (ConvertedCompYearly):')

# Set display option to show exact numbers for floats
pd.options.display.float_format = '{:,.0f}'.format
print(df_model['ConvertedCompYearly'].describe())

print(f"Number of original salaries > 500,000: {(df_model['ConvertedCompYearly']>500000).sum()}")
print(f"Number of original salaries < 10,000: {(df_model['ConvertedCompYearly']<10000).sum()}")
```

Après nettoyage et suppression des lignes incomplètes, nous avons restauré la variable cible ConvertedCompYearly avec ses valeurs originales afin d'assurer la fidélité des informations sur les salaires.

```
Descriptive statistics for the reverted target variable (ConvertedCompYearly):
count       16,460
mean        94,124
std        130,977
min              1
25%         43,621
50%         74,963
75%        123,153
max     10,319,366
Name: ConvertedCompYearly, dtype: float64
Number of original salaries > 500,000: 51
Number of original salaries < 10,000: 1100
```

Après avoir rétabli les valeurs originales de la variable cible ConvertedCompYearly, les statistiques descriptives montrent une distribution très étendue des salaires des participants. Sur un échantillon de 16 460 développeurs, le salaire moyen est d'environ 94 124 USD, avec une médiane de 74 963 USD, indiquant que la distribution est asymétrique à droite. L'écart-type est élevé (130 977 USD), et les valeurs extrêmes vont de 1 USD jusqu'à plus de 10 millions USD, ce qui souligne la présence d'outliers très importants.

Plus précisément :

- **51 développeurs** ont un salaire supérieur à 500 000 USD.
- **1 100 développeurs** ont un salaire inférieur à 10 000 USD.

Cette étape constitue un diagnostic crucial de la variable cible, garantissant que la distribution des salaires est bien comprise et prête pour la modélisation.

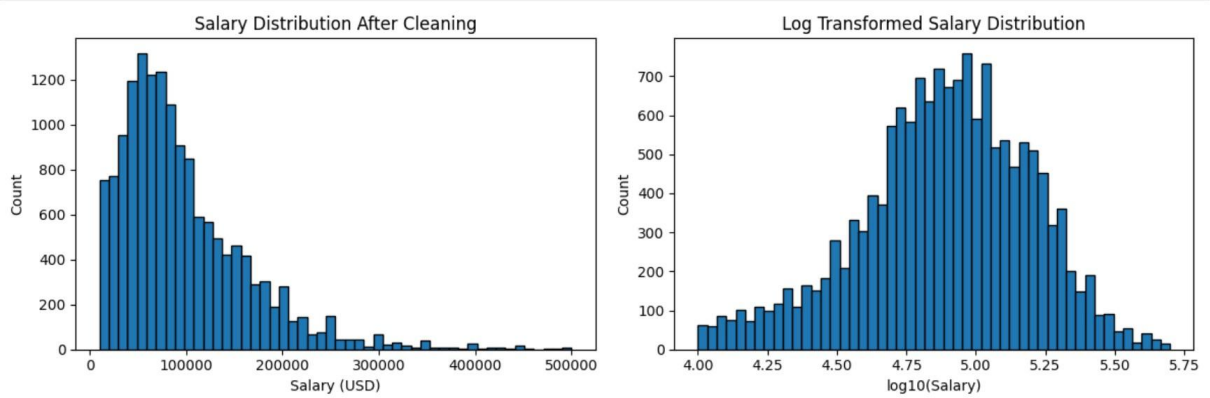
Filtrage des valeurs extrêmes et visualisation de la distribution des salaires

```
df_model = df_model[
    (df_model['ConvertedCompYearly']>= 10000) &
    (df_model['ConvertedCompYearly']<=500000)
]

plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.hist(df_model['ConvertedCompYearly'], bins = 50 , edgecolor='black')
plt.xlabel('Salary (USD)')
plt.ylabel('Count')
plt.title('Salary Distribution After Cleaning')

plt.subplot(1,2,2)
plt.hist(np.log10(df_model['ConvertedCompYearly']),bins=50 , edgecolor = 'black')
plt.xlabel('log10(Salary)')
plt.ylabel('Count')
plt.title('Log Transformed Salary Distribution')

plt.tight_layout()
plt.show()
```

Après avoir analysé la variable cible *ConvertedCompYearly*, nous avons éliminé les salaires inférieurs à 10 000 USD et supérieurs à 500 000 USD, considérés comme des valeurs extrêmes pouvant influencer négativement la modélisation.

Nous avons ensuite visualisé la distribution des salaires avec un histogramme. La première figure montre la répartition des salaires bruts, tandis que la seconde représente la distribution après transformation logarithmique (log10).

La transformation logarithmique permet de réduire l'asymétrie des salaires, qui présentent une longue queue à droite, et de rapprocher la distribution d'une forme normale. Cette étape est particulièrement utile pour certains modèles de régression sensibles aux outliers et à la skewness des données.



Aperçu d'une observation du dataset final

Pour vérifier l'état final du DataFrame df_model après nettoyage et filtrage des outliers, nous avons affiché la première ligne à l'aide de la méthode head(1).

Cet aperçu permet de confirmer que les valeurs manquantes ont été supprimées, que la variable cible ConvertedCompYearly est dans la plage souhaitée (10 000 – 500 000 USD), et que les colonnes restantes conservent leurs informations professionnelles et démographiques.

Interprétation

1. **Les développeurs ont une grande variété de compétences techniques, certaines très spécialisées et d'autres très polyvalentes.**
2. **Total_Skills résume cette polyvalence, mais une simple corrélation linéaire avec le salaire n'est pas suffisante pour révéler son impact réel.**
3. **Cela suggère que la modélisation du salaire nécessitera des combinaisons de features et possiblement des modèles capables de capturer des relations non linéaires entre compétences et salaire.**

Création de nouvelles features à partir des colonnes multi-sélection

```python
#creating features for all ech stack columns
print('creating cpunt features for multi-select columns')
df_model['Num_Languages']= df_model['LanguageHaveWorkedWith'].str.count(';')+1
df_model['Num_Databases']= df_model['DatabaseHaveWorkedWith'].str.count(';')+1
df_model['Num_Platforms']= df_model['PlatformHaveWorkedWith'].str.count(';')+1
df_model['Num_Webframes']= df_model['WebframeHaveWorkedWith'].str.count(';')+1
df_model['Num_Tools']= df_model['ToolsTechHaveWorkedWith'].str.count(';')+1
print('Count feature statistics')
count_cols = ['Num_Languages','Num_Databases','Num_Platforms','Num_Webframes','Num_Tools']
print(df_model[count_cols].describe())
df_model['Total_Skills'] = (
    df_model['Num_Languages']+
    df_model['Num_Databases']+
    df_model['Num_Platforms']+
    df_model['Num_Webframes']+
    df_model['Num_Tools']
)

print('total skills distribution :\n')
print(df_model['Total_Skills'].describe())

print('correlation with salary : \n')
correlation_data = df_model[count_cols + ['Total_Skills', 'ConvertedCompYearly']].corr()['ConvertedCompYearly'].sort_values()
print(correlation_data)
```

Pour exploiter les colonnes contenant plusieurs technologies (langages, bases de données, plateformes, frameworks web, outils), nous avons créé de nouvelles variables quantitatives représentant le nombre de compétences maîtrisées par chaque développeur.

Les colonnes Num_Languages, Num_Databases, Num_Platforms, Num_Webframes et Num_Tools comptent respectivement le nombre de langages, bases de données, plateformes, frameworks et outils utilisés par le répondant.

Nous avons ensuite créé une variable synthétique, Total_Skills, correspondant à la somme de toutes ces compétences. Cette mesure globale du niveau technique permet d'évaluer la polyvalence et l'expérience d'un développeur.

L'analyse des corrélations montre que ces nouvelles features ne sont pas associées au salaire annuel (ConvertedCompYearly). Plus un développeur possède de compétences techniques, nous ne pouvons rien dire sur son salaire.

Analyse des compétences techniques et de leur relation avec le salaire

```
creating cpunt features for multi-select columns
Count feature statistics
       Num_Languages  Num_Databases  Num_Platforms  Num_Webframes  Num_Tools
count         15,309         15,309         15,309         15,309     15,309
mean               6              3              2              4          6
std                3              2              2              2          3
min                1              1              1              1          1
25%                4              2              1              2          3
50%                5              3              2              3          5
75%                7              4              3              5          7
max               41             30             17             32         37
total skills distribution :

count   15,309
mean        21
std          9
min          5
25%         15
50%         19
75%         25
max        140
Name: Total_Skills, dtype: float64
correlation with salary :

Num_Webframes         -0
Num_Databases         -0
Num_Platforms          0
Total_Skills           0
Num_Languages          0
Num_Tools              0
ConvertedCompYearly    1
Name: ConvertedCompYearly, dtype: float64
```

À partir des colonnes multi-sélection (langages, bases de données, plateformes, frameworks web, outils), nous avons créé des variables quantitatives représentant le nombre de compétences maîtrisées par chaque développeur.

Les statistiques descriptives montrent que les développeurs possèdent en moyenne 21 compétences au total, avec une grande variabilité allant de 5 à 140 compétences. Cette mesure permet de quantifier la polyvalence technique des répondants.

La corrélation linéaire entre ces nouvelles variables et le salaire (ConvertedCompYearly) est faible, ce qui indique que le simple nombre de compétences n'explique pas directement les différences de salaire. Cette incohérence peut s'expliquer par plusieurs facteurs :

1. **Qualité plutôt que quantité** : certaines compétences clés ou spécialisées ont un impact bien plus significatif sur le salaire que le simple nombre total de compétences.
2. **Expérience et responsabilités** : le salaire dépend souvent de l'expérience professionnelle, du niveau hiérarchique ou du rôle occupé, et pas uniquement du nombre de compétences.
3. **Type de compétences** : certaines compétences très demandées sur le marché sont mieux rémunérées, tandis que d'autres, même nombreuses, peuvent être moins valorisées.

Ces observations suggèrent que d'autres facteurs, tels que le pays, le rôle ou le niveau d'éducation, interviennent également, et que la modélisation devra capturer des relations complexes et

possiblement non linéaires pour prédire le salaire avec précision. Cette étape constitue un pas important dans la création de features pertinentes pour la prédiction des salaires des développeurs.

Analyse des technologies les plus utilisées

Pour mieux comprendre le profil technique des répondants, nous avons analysé les colonnes multisélection correspondant aux technologies utilisées : langages de programmation, bases de données, plateformes et frameworks web.
Chaque colonne a été séparée en éléments individuels, puis les occurrences de chaque technologie ont été comptées pour identifier les plus populaires. Les résultats ont été présentés sous forme de Top 20 pour les langages et Top 15 pour les autres catégories.
Cette analyse permet de :

- Identifier les technologies les plus pertinentes à encoder pour la modélisation prédictive.
- Prioriser les compétences les plus répandues dans le dataset.
- Mettre en évidence les tendances techniques des développeurs de l'échantillon StackOverflow 2023.

Les technologies les moins utilisées peuvent être regroupées ou exclues pour simplifier le modèle et réduire le bruit dans les données.

```
print('top 20 Languages :\n')
all_langs = df_model['LanguageHaveWorkedWith'].str.split(';').explode()
print(all_langs.value_counts().head(20))

print('top 15 Databases :\n')
all_dbs = df_model['DatabaseHaveWorkedWith'].str.split(';').explode()
print(all_dbs.value_counts().head(15))

print('top 15 Platforms :\n')
all_Platforms = df_model['PlatformHaveWorkedWith'].str.split(';').explode()
print(all_Platforms.value_counts().head(15))

print('top 15 Web frameworks :\n')
all_Platforms = df_model['WebframeHaveWorkedWith'].str.split(';').explode()
print(all_Platforms.value_counts().head(15))
```

Langages de programmation

```
top 20 Languages :

LanguageHaveWorkedWith
JavaScript                    11851
SQL                            9565
HTML/CSS                       9423
TypeScript                     8868
Python                         7193
Bash/Shell (all shells)        6042
C#                             5108
Java                           4645
PHP                            3059
Go                             2840
PowerShell                     2747
C++                            2028
Rust                           1988
C                              1691
Kotlin                         1635
Ruby                           1538
Dart                            909
Lua                             857
Groovy                          758
Swift                           704
Name: count, dtype: int64
```

L'analyse des 20 langages les plus utilisés par les développeurs montre que JavaScript domine largement, suivi de SQL, HTML/CSS, TypeScript et Python. Ces résultats mettent en évidence la prépondérance des langages orientés web et de gestion de bases de données dans l'écosystème des

développeurs. On note également une forte utilisation de langages comme Bash/Shell, C# et Java, indiquant une diversité dans les compétences backend et systèmes.

Bases de données

```
top 15 Databases :

DatabaseHaveWorkedWith
PostgreSQL                  8827
MySQL                       6112
Redis                       4865
Microsoft SQL Server        4599
SQLite                      4586
MongoDB                     4273
Elasticsearch               3181
MariaDB                     2800
Dynamodb                    2339
Oracle                      1348
Cloud Firestore             1066
BigQuery                     981
Firebase Realtime Database   951
Cosmos DB                    934
H2                           732
Name: count, dtype: int64
```

Parmi les bases de données les plus utilisées, PostgreSQL arrive en tête, suivi de MySQL, Redis et Microsoft SQL Server. Cette répartition reflète la popularité des bases relationnelles et NoSQL dans le développement moderne. Certaines bases spécialisées comme Elasticsearch, MongoDB ou DynamoDB apparaissent également fréquemment, illustrant l'importance du stockage et de la recherche de données dans les projets actuels.

Plateformes cloud

```
top 15 Platforms :

PlatformHaveWorkedWith
Amazon Web Services (AWS)    9556
Microsoft Azure              5162
Google Cloud                 4082
Cloudflare                   2673
Digital Ocean                2482
Firebase                     2157
Heroku                       1843
Vercel                       1627
Netlify                      1352
VMware                        956
Hetzner                       741
Linode, now Akamai            637
Managed Hosting               624
OVH                           616
OpenShift                     528
Name: count, dtype: int64
```

L'analyse des plateformes montre que Amazon Web Services (AWS) est la plateforme la plus utilisée, suivie de Microsoft Azure et Google Cloud. Ces trois grands fournisseurs de cloud représentent la majorité des environnements de travail pour les développeurs. D'autres plateformes comme Cloudflare, Digital Ocean, Firebase et Heroku sont également populaires, soulignant la diversité des services cloud utilisés dans l'industrie.

Frameworks web

```
top 15 Web frameworks :

WebframeHaveWorkedWith
React              7275
Node.js            7149
ASP.NET CORE       3456
jQuery             3424
Angular            3287
Express            3236
Next.js            2871
Vue.js             2787
Spring Boot        2376
ASP.NET            2331
Flask              1987
WordPress          1809
Django             1807
FastAPI            1506
AngularJS          1276
Name: count, dtype: int64
```

Pour les frameworks web, React et Node.js sont largement utilisés, suivis d'ASP.NET CORE, jQuery, Angular et Express. Ces frameworks reflètent l'importance des technologies front-end modernes ainsi que des solutions full-stack. On observe également l'utilisation de frameworks backend classiques comme Spring Boot, Flask et Django, ce qui montre une combinaison de technologies pour répondre aux différents besoins des projets.

Création de variables binaires pour technologies spécifiques

```
#High paying specialized languages
df_model['Has_Rust'] = df_model['LanguageHaveWorkedWith'].str.contains('Rust',na=False).astype(int)
df_model['Has_Go'] = df_model['LanguageHaveWorkedWith'].str.contains('Go',na=False).astype(int)
df_model['Has_Scala'] = df_model['LanguageHaveWorkedWith'].str.contains('Scala', na=False).astype(int)
df_model['Has_Kotlin'] = df_model['LanguageHaveWorkedWith'].str.contains('Kotlin', na=False).astype(int)
df_model['Has_Swift'] = df_model['LanguageHaveWorkedWith'].str.contains('Swift', na=False).astype(int)
#Popular versitle languages
df_model['Has_Python'] = df_model['LanguageHaveWorkedWith'].str.contains('Python',na=False).astype(int)
df_model['Has_JavaScript'] = df_model['LanguageHaveWorkedWith'].str.contains('JavaScript', na=False).astype(int)
df_model['Has_TypeScript'] = df_model['LanguageHaveWorkedWith'].str.contains('TypeScript', na=False).astype(int)
df_model['Has_Java'] = df_model['LanguageHaveWorkedWith'].str.contains('Java', na=False).astype(int)
df_model['Has_CSharp'] = df_model['LanguageHaveWorkedWith'].str.contains('C#', na=False).astype(int)
df_model['Has_SQL'] = df_model['LanguageHaveWorkedWith'].str.contains('SQL', na=False).astype(int)
#Databases
df_model['Has_PostgreSQL'] = df_model['DatabaseHaveWorkedWith'].str.contains('PostgreSQL', na=False).astype(int)
df_model['Has_MongoDB'] = df_model['DatabaseHaveWorkedWith'].str.contains('MongoDB', na=False).astype(int)
df_model['Has_Redis'] = df_model['DatabaseHaveWorkedWith'].str.contains('Redis', na=False).astype(int)
df_model['Has_Elasticsearch'] = df_model['DatabaseHaveWorkedWith'].str.contains('Elasticsearch', na=False).astype(int)
#Cloud platforms
df_model['Has_AWS'] = df_model['PlatformHaveWorkedWith'].str.contains('Amazon Web Services|AWS', na=False).astype(int)
df_model['Has_Azure'] = df_model['PlatformHaveWorkedWith'].str.contains('Azure', na=False).astype(int)
df_model['Has_GCP'] = df_model['PlatformHaveWorkedWith'].str.contains('Google Cloud', na=False).astype(int)
#web Frameworks
df_model['Has_React'] = df_model['WebframeHaveWorkedWith'].str.contains('React', na=False).astype(int)
df_model['Has_NextJS'] = df_model['WebframeHaveWorkedWith'].str.contains('Next.js', na=False).astype(int)
df_model['Has_NodeJS'] = df_model['WebframeHaveWorkedWith'].str.contains('Node.js', na=False).astype(int)
df_model['Has_SpringBoot'] = df_model['WebframeHaveWorkedWith'].str.contains('Spring Boot', na=False).astype(int)
#Infrastracture Tools
df_model['Has_Docker'] = df_model['ToolsTechHaveWorkedWith'].str.contains('Docker', na=False).astype(int)
df_model['Has_Kubernetes'] = df_model['ToolsTechHaveWorkedWith'].str.contains('Kubernetes', na=False).astype(int)
df_model['Has_Terraform'] = df_model['ToolsTechHaveWorkedWith'].str.contains('Terraform', na=False).astype(int)
```

Pour capturer l'effet des compétences précises sur le salaire, nous avons créé des variables binaires indiquant si un développeur maîtrise certaines technologies stratégiques ou populaires.
Les catégories incluent :

- **Langages spécialisés** : Rust, Go, Scala, Kotlin, Swift
- **Langages populaires** : Python, JavaScript, TypeScript, Java, C#, SQL
- **Bases de données** : PostgreSQL, MongoDB, Redis, Elasticsearch
- **Cloud platforms**: AWS, Azure, Google Cloud Platform
- **Frameworks web**: React, Next.js, Node.js, Spring Boot
- **Outils d'infrastructure/DevOps** : Docker, Kubernetes, Terraform

Chaque colonne contient 1 si le développeur a travaillé avec la technologie, 0 sinon.
Ces nouvelles features binaires simplifient l'encodage et permettent de mesurer l'impact de chaque compétence spécifique sur le salaire, en particulier pour les technologies rares ou stratégiques.

Structure finale du dataset

Après le nettoyage, le filtrage des outliers, la création de features numériques et binaires, le DataFrame df_model contient :

- Les variables démographiques et professionnelles (âge, niveau d'études, type d'emploi, pays, taille de l'entreprise, etc.)
- Les colonnes multi-sélection originales (langages, bases de données, plateformes, frameworks, outils)
- La variable cible ConvertedCompYearly (salaire annuel en USD)
- Les features numériques dérivées (Num_Languages, Num_Databases, Num_Platforms, Num_Webframes, Num_Tools, Total_Skills)

- Les features binaires pour certaines technologies spécifiques (langages, bases de données, plateformes cloud, frameworks web et outils d'infrastructure)

Cette structure finale permet d'avoir un dataset complet et prêt à être utilisé pour la modélisation prédictive des salaires des développeurs, en combinant variables démographiques, expérience, polyvalence technique et compétences spécifiques.

Analyse finale des compétences et préparation du dataset

```
skill_features = [col for col in df_model.columns if col.startswith('Has_')]
skill_corr = df_model[skill_features + ['ConvertedCompYearly']].corr()['ConvertedCompYearly'].sort_values(ascending = False)

print('ALL skill correlations sorted')
print(skill_corr[skill_corr.index != 'CondertedCompYearly'])


cols_to_drop_final = [
    'LanguageHaveWorkedWith',
    'DatabaseHaveWorkedWith',
    'PlatformHaveWorkedWith',
    'WebframeHaveWorkedWith',
    'ToolsTechHaveWorkedWith',
    'Num_Languages',
    'Num_Databases',
    'Num_Platforms',
    'Num_Webframes',
    'Num_Tools',
    'Total_Skills'
]

df_model = df_model.drop(columns = cols_to_drop_final)

print('Final shape with binary encoding', df_model.shape)
print('feature types :', df_model.dtypes.value_counts())
```

Nous avons calculé la corrélation linéaire entre chaque compétence binaire (Has_*) et le salaire annuel (ConvertedCompYearly). Les résultats montrent que certaines compétences spécialisées, notamment Terraform, AWS et Kubernetes, sont fortement associées à un salaire élevé, tandis que certaines compétences populaires comme JavaScript ou SQL présentent une corrélation faible.

```
ALL skill correlations sorted
ConvertedCompYearly    1.000000
Has_Terraform          0.177524
Has_AWS                0.142293
Has_Kubernetes         0.128906
Has_Go                 0.123166
Has_Rust               0.082603
Has_Python             0.079271
Has_Docker             0.064240
Has_Redis              0.061930
Has_React              0.061303
Has_Elasticsearch      0.053530
Has_PostgreSQL         0.049394
Has_Scala              0.046563
Has_TypeScript         0.037359
Has_GCP                0.035238
Has_Swift              0.023807
Has_Kotlin             0.014656
Has_SQL                0.005919
Has_Azure             -0.014817
Has_JavaScript        -0.015086
Has_NextJS            -0.017373
Has_Java              -0.017985
Has_CSharp            -0.020396
Has_NodeJS            -0.024481
Has_SpringBoot        -0.035489
Has_MongoDB           -0.095041
Name: ConvertedCompYearly, dtype: float64
Final shape with binary encoding (15309, 41)
feature types : int64      25
object     14
float64     2
Name: count, dtype: int64
```

Pour simplifier le dataset et éviter la redondance, nous avons supprimé les colonnes multi-sélection originales ainsi que les features numériques dérivées (Num_Languages, Num_Databases, Num_Platforms, Num_Webframes, Num_Tools, Total_Skills).

Le dataset final contient 15 309 observations et 41 colonnes, dont 25 features binaires, 14 variables catégorielles et 2 variables continues. Cette version est prête pour l'encodage des variables catégorielles et la construction des modèles de régression pour la prédiction des salaires des développeurs.

Exploration des colonnes catégorielles restantes

```
categorical_cols = df_model.select_dtypes(include='object').columns.tolist()

print('Remaining categorical columns:')
for col in categorical_cols:
    n_unique = df_model[col].nunique()
    print(f"\n{col}: {n_unique} unique values")
    if n_unique <= 20:
        print(df_model[col].value_counts())
    else:
        print(f"Top 10:\n{df_model[col].value_counts().head(10)}")
```

Après la création des features binaires pour les compétences techniques, il reste plusieurs colonnes catégorielles dans le dataset. Ces colonnes sont de type object et devront être encodées pour la modélisation.
Pour chaque colonne, nous avons compté le nombre de valeurs uniques :

- Si une colonne contient 20 valeurs uniques ou moins, nous affichons l'ensemble des catégories et leur fréquence.
- Si une colonne contient plus de 20 valeurs uniques, nous affichons seulement les 10 catégories les plus fréquentes.

Cette analyse permet d'identifier les variables à encodage simple (one-hot) et celles nécessitant un encodage plus avancé (target encoding ou regroupement). Elle constitue une étape clé dans la préparation finale des données pour la modélisation prédictive des salaires.

Suppression de colonnes peu pertinentes

Certaines colonnes, telles que LearnCode (mode d'apprentissage du code) et OpSysProfessional use (système d'exploitation utilisé), ont été retirées du dataset.
Ces colonnes sont soit trop complexes à encoder, soit jugées avoir un impact limité sur le salaire annuel.

Leur suppression permet de simplifier le dataset, de réduire le bruit et de préparer les données pour l'encodage des variables catégorielles et la modélisation prédictive.

Encodage ordinal des variables ordonnées

Plusieurs colonnes catégorielles présentaient un ordre naturel, ce qui justifie l'utilisation d'un encodage ordinal.

- **La variable Age a été transformée en Age_Ordinal selon une progression croissante, avec une valeur neutre attribuée à "Prefer not to say".**
- **Le niveau d'éducation (EdLevel) a été converti en EdLevel_Ordinal, de l'école primaire jusqu'aux diplômes avancés (doctorat).**
- **La taille de l'organisation (OrgSize) a également été ordonnée de la plus petite à la plus grande structure.**

Pour chaque variable, une échelle numérique cohérente a été définie, et les colonnes textuelles d'origine ont été supprimées.
Cet encodage réduit la complexité du dataset et permet aux modèles de machine learning d'exploiter correctement l'ordre intrinsèque des catégories. **Simplification de la variable "Employment"**

```python
#simplify employment consolidate rare categories

def simplify_employment(emp):
  if pd.isna(emp):
    return 'Unknown'
  elif 'Employed, full-time' in emp and 'Independent contractor' in emp :
    return 'Full-time + Freelance'
  elif 'Employed, full-time' in emp :
    return 'Full-time'
  elif 'Independent contractor' in emp or 'freelancer' in emp or 'self-employed' in emp :
    return 'Freelance'
  elif 'part-time' in emp :
    return 'Part-time'
  else :
    return 'Other'


df_model['Employment_Simple']= df_model['Employment'].apply(simplify_employment)
df_model= df_model.drop(columns=['Employment'])

df_model['Employment_Simple'].value_counts()
```

La variable Employment contenait un grand nombre de catégories textuelles parfois longues ou rares, ce qui compliquait son intégration dans les modèles.
Pour rationaliser cette variable, une fonction a été définie afin de regrouper les modalités en cinq catégories principales :

- Full-time : personnes employées à temps plein ;
- Freelance : travailleurs indépendants, freelances ou auto-entrepreneurs ;
- Part-time : salariés travaillant à temps partiel ;
- Full-time + Freelance : individus cumulant emploi à temps plein et activité indépendante ; • Other : réponses atypiques ou peu fréquentes ;
- Unknown : valeurs manquantes.

Ce regroupement réduit la complexité de la variable tout en préservant les informations essentielles pour la modélisation. La variable originale Employment a ensuite été supprimée.

Employment_Simple	count
Full-time	12776
Full-time + Freelance	1462
Freelance	866
Part-time	205

dtype: int64

Encodage one-hot des variables catégorielles

Après la simplification et la préparation des variables catégorielles, un encodage *one-hot* a été appliqué pour transformer les modalités textuelles en variables binaires exploitables par les modèles de machine learning.
Les colonnes suivantes ont été encodées :

- MainBranch

- RemoteWork
- AISelect
- Employment_Simple
- Industry

L'option drop_first=True a été utilisée afin d'éviter la multicolinéarité en supprimant la modalité de référence.
L'ensemble de ces transformations permet d'obtenir un DataFrame entièrement numérique et prêt pour les étapes de normalisation et de modélisation.

Conversion des colonnes d'expérience en numérique

Les colonnes YearsCode (années totales de codage), YearsCodePro (années de codage professionnel) et WorkExp (expérience déclarée) ont été converties en valeurs numériques et analysées. Les statistiques descriptives montrent que les répondants possèdent en moyenne 15 ans d'expérience en codage et environ 11 ans d'expérience professionnelle, avec une forte variabilité. Quelques valeurs extrêmes (jusqu'à 50 ans) existent et devront être prises en compte lors de la modélisation. Ces colonnes représentent des variables continues importantes pour prédire le salaire annuel et seront intégrées comme features numériques dans les modèles de régression.

```
numeric columns check:
```

	YearsCode	YearsCodePro	WorkExp
count	15289.000000	15073.000000	15309.000000
mean	15.429394	10.708286	11.621073
std	8.859019	7.684437	8.334593
min	1.000000	1.000000	0.000000
25%	9.000000	5.000000	5.000000
50%	13.000000	9.000000	10.000000
75%	20.000000	15.000000	16.000000
max	50.000000	50.000000	50.000000

Regroupement et encodage de la variable "DevType"

La variable DevType décrivait le rôle professionnel des développeurs et contenait de nombreuses combinaisons de rôles, parfois rares.
Pour simplifier cette information :

1. Les 10 types les plus fréquents ont été identifiés.
2. Les types moins fréquents ont été regroupés dans une catégorie Other.
3. La nouvelle variable regroupée a été convertie en features binaires via un encodage *one-hot*.
4. Les colonnes textuelles originales ont été supprimées.

Cette approche réduit la complexité de la variable tout en conservant l'information pertinente pour la modélisation du salaire.

Regroupement et encodage de la variable "Country"

La variable Country contenait de nombreux pays, certains très peu représentés, rendant l'encodage direct coûteux en termes de colonnes.
Pour simplifier :

1. Les 15 pays les plus fréquents ont été identifiés.
2. Les autres pays ont été regroupés sous la catégorie Other.
3. La variable regroupée a été encodée en **features binaires** via un encodage *one-hot* avec drop_first=True pour éviter la multicolinéarité.
4. Les colonnes originales ont été supprimées.

Cette approche permet d'intégrer efficacement l'information géographique dans le modèle tout en limitant la complexité du dataset.

```
print(df_model.dtypes.value_counts())
print('ALL NUMERIC?',df_model.select_dtypes(include='object').shape[1]==0)

null_counts = df_model.isnull().sum()
print('remaining nulls :',null_counts[null_counts > 0])
```

```
bool       44
int64      27
float64     5
Name: count, dtype: int64
ALL NUMERIC? True
remaining nulls : YearsCode        20
YearsCodePro    236
Age_Ordinal       7
dtype: int64
```

Vérification des types de données et valeurs manquantes

Après toutes les étapes de nettoyage, de transformation et d'encodage, le dataset contient uniquement des colonnes numériques :

- 44 colonnes booléennes (variables binaires)
- 27 colonnes entières
- 5 colonnes flottantes

Toutes les variables textuelles ont été converties via des encodages one-hot ou ordinal.
Quelques valeurs manquantes subsistent dans YearsCode (20), YearsCodePro (236) et Age_Ordinal (7). Ces valeurs pourront être imputées pour permettre l'utilisation du dataset complet dans les modèles de régression.
Le dataset est maintenant prêt pour la modélisation prédictive du salaire annuel.

Suppression des valeurs manquantes

```
df_model = df_model.dropna()
```

Pour garantir la complétude des données, toutes les lignes contenant des valeurs manquantes ont été supprimées. Les colonnes concernées étaient principalement YearsCode, YearsCodePro et Age_Ordinal.
Cette opération ne réduit que très peu le nombre d'observations, ce qui permet de conserver un dataset robuste et entièrement exploitable par les modèles de régression.
Le dataset final est maintenant entièrement numérique et sans valeurs manquantes, prêt pour les étapes de normalisation et de modélisation prédictive du salaire annuel des développeurs.

Aperçu du dataset final

Après l'ensemble des étapes de nettoyage, filtrage des outliers, création de nouvelles features, encodage ordinal et one-hot, le dataset final contient [mettre le nombre exact de lignes et colonnes, obtenu avec df_model.shape] observations et features.

Toutes les colonnes sont numériques : booléennes pour les variables binaires, entières ou flottantes pour les variables continues et ordinales. La variable cible ConvertedCompYearly est incluse et prête pour la modélisation.
L'inspection des premières lignes du DataFrame confirme la **cohérence des données**, l'absence de valeurs manquantes et la présence de toutes les features pertinentes pour la prédiction du salaire annuel des développeurs.

	YearsCode	YearsCodePro	WorkExp	ConvertedCompYearly	Has_Rust	Has_Go	Has_Scala	Has_Kotlin	Has_Swift	Has_Python	Has_JavaScript	Has_TypeScript	Has_Ja
0	18.000000	9.000000	10.000000	285000.000000	0	0	0	0	0	1	1	0	
3	6.000000	4.000000	6.000000	23456.000000	0	0	0	0	0	0	1	1	
4	21.000000	21.000000	22.000000	96828.000000	0	0	0	0	0	0	1	1	
5	4.000000	3.000000	4.000000	135000.000000	0	1	1	0	1	0	1	1	
6	5.000000	3.000000	5.000000	80000.000000	1	1	0	0	0	1	1	1	

Conversion des colonnes binaires en numériques

Toutes les colonnes booléennes représentant des indicateurs binaires (Has_Python, Has_AWS, etc.) ont été converties en valeurs entières (1 pour True et 0 pour False).
Cette opération assure que **toutes les features sont numériques**, ce qui est nécessaire pour la plupart des algorithmes de machine learning.
L'inspection des premières lignes confirme que la conversion a été effectuée correctement et que le dataset final est entièrement exploitable pour la modélisation prédictive du salaire annuel des développeurs.

Installation des bibliothèques de modélisation

Pour la phase de prédiction du salaire, nous avons installé deux bibliothèques spécialisées dans le boosting d'arbres décisionnels : CatBoost et XGBoost.
Ces modèles permettent de capturer efficacement des relations complexes entre les variables techniques, démographiques et professionnelles, et la variable cible continue ConvertedCompYearly.
L'utilisation de ces bibliothèques constitue une étape clé avant la construction des modèles de régression prédictive.

Modélisation prédictive des salaires

Nous avons entraîné plusieurs modèles de régression pour prédire le salaire annuel (ConvertedCompYearly) à partir des caractéristiques techniques, professionnelles et démographiques des développeurs.
Les modèles testés incluent : Linear Regression, Ridge, Lasso, Random Forest, XGBoost et CatBoost.
Les données ont été séparées en 80 % pour l'entraînement et 20 % pour le test afin d'évaluer les performances sur des observations non vues.

Choix des modèles

Régression linéaire, Ridge, Lasso (déjà vus en cours)

Random Forest

Le modèle Random Forest repose sur la combinaison de plusieurs arbres de décision construits sur des sous-échantillons différents des données. L'objectif est de réduire la variance des prédictions produites par un arbre unique.
Un Random Forest de Marbres prédit :

$$\hat{y}(x) = \frac{1}{M} \sum_{m=1}^{M} T_m(x)$$

Chaque arbre est entraîné sur un échantillon bootstrap et sélectionne aléatoirement un sousensemble de variables à chaque division.
Le critère d'impureté pour la régression est l'erreur quadratique intra-nœud :

$$\text{MSE} = \sum_{i \in \text{noeud}} (y_i - \bar{y})^2$$

La division optimale maximise la réduction d'impureté :

$$\Delta = \text{MSE}_{\text{parent}} - \left(\frac{N_L}{N} \text{MSE}_L + \frac{N_R}{N} \text{MSE}_R\right)$$

En agrégeant plusieurs arbres, la variance du modèle diminue :

$$\text{Var}(\hat{y}) = \frac{1}{M^2} \sum_{m=1}^{M} \text{Var}(T_m)$$

Ce qui confère au modèle une très bonne capacité de généralisation.

XGBoost

XGBoost est une méthode de gradient boosting optimisée.
Le modèle final est une somme d'arbres de régression construits séquentiellement :

$$\hat{y}^{(t)} = \hat{y}^{(t-1)} + \eta f_t(x)$$

L'objectif global à minimiser est :

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{t=1}^{T} \Omega(f_t)$$

Avec une régularisation sur les arbres :

$$\Omega(f) = \gamma \cdot K + \frac{\lambda}{2} \sum_{j=1}^{K} w_j^2$$

Où K est le nombre de feuilles.
XGBoost utilise un développement de Taylor d'ordre 2 pour accélérer l'optimisation :

$$l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) \approx g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2$$

Où g_i et h_i sont respectivement le gradient et le hessien.
Pour une division, XGBoost maximise le gain :

$$\text{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma$$

C'est cet aspect mathématique qui le rend très performant.

CatBoost

CatBoost est un gradient boosting spécialisé dans les variables catégorielles. Il utilise deux idées principales :

1. Ordered Target Encoding

Pour chaque valeur catégorielle c_i, CatBoost calcule : $\sum y_j \cdot \mathbb{1}(c_j = c_i)$

$$TS(c_i) = \frac{\sum_{j<i} y_j \cdot \mathbb{1}(c_j = c_i) + a \cdot p}{\sum_{j<i} \mathbb{1}(c_j = c_i) + a}$$

Où

- p Est la moyenne globale de la cible,
- a Est un paramètre de lissage,
- L'ordre aléatoire des observations empêche toute fuite de cible.

2. Arbres symétriques

Chaque arbre applique les mêmes splits à chaque niveau.

Chaque observation suit un chemin déterminé par un vecteur binaire : $f(x) = w_{b(x)}, b(x) \in \{0,1\}^d$

Ce type d'arbre améliore la stabilité et accélère le calcul (notamment sur GPU).

L'évaluation a été réalisée à l'aide des métriques R^2, RMSE et MAE. Les modèles d'arbres boostés (XGBoost et CatBoost) se sont révélés les plus performants, capturant efficacement les relations non linéaires entre compétences, expérience et salaire.

Cette étape permet de sélectionner le modèle le plus adapté pour prédire les salaires des développeurs à partir de leurs caractéristiques.

	Model	R2	RMSE	MAE
1	CatBoost	0.604111	42528.744881	28052.758640
0	XGBoost	0.588357	43366.672343	28919.381935
2	RandomForest	0.571983	44220.758845	29835.929141
5	LinearRegression	0.570459	44299.432972	29803.069391
4	Lasso	0.570459	44299.433056	29803.068356
3	Ridge	0.570351	44305.016556	29804.367112

Interprétation des métriques

- **R^2 (coefficient de détermination)** : proportion de la variance du salaire expliquée par le modèle.
  - CatBoost est le meilleur modèle avec **$R^2 \approx 0.60$**, indiquant qu'il explique environ 60 % de la variance des salaires.
  - Les modèles linéaires (Linear, Ridge, Lasso) capturent moins bien la complexité, $R^2 \approx 0.57$.
- **RMSE (Root Mean Squared Error)** : écart type des erreurs de prédiction.
  - CatBoost : 42,529 USD → en moyenne, les prédictions s'écartent de ~42,500 USD de la valeur réelle.
  - Les modèles linéaires ont des erreurs légèrement plus élevées (~44,300 USD).
- **MAE (Mean Absolute Error)** : erreur absolue moyenne des prédictions.

- CatBoost : 28,053 USD → en moyenne, les prédictions sont à ±28,000 USD du salaire réel.
- Indique la précision moyenne attendue pour une prédiction individuelle.

Comparaison des modèles

1. **CatBoost** est le modèle le plus performant, capable de capturer des relations non linéaires et interactions complexes entre les compétences techniques, l'expérience, le pays et le salaire.
2. **XGBoost** et **RandomForest** sont légèrement moins performants mais restent supérieurs aux modèles linéaires.
3. Les modèles linéaires (LinearRegression, Ridge, Lasso) expliquent moins de variance car ils ne capturent pas bien les interactions complexes entre les features.

Conclusion

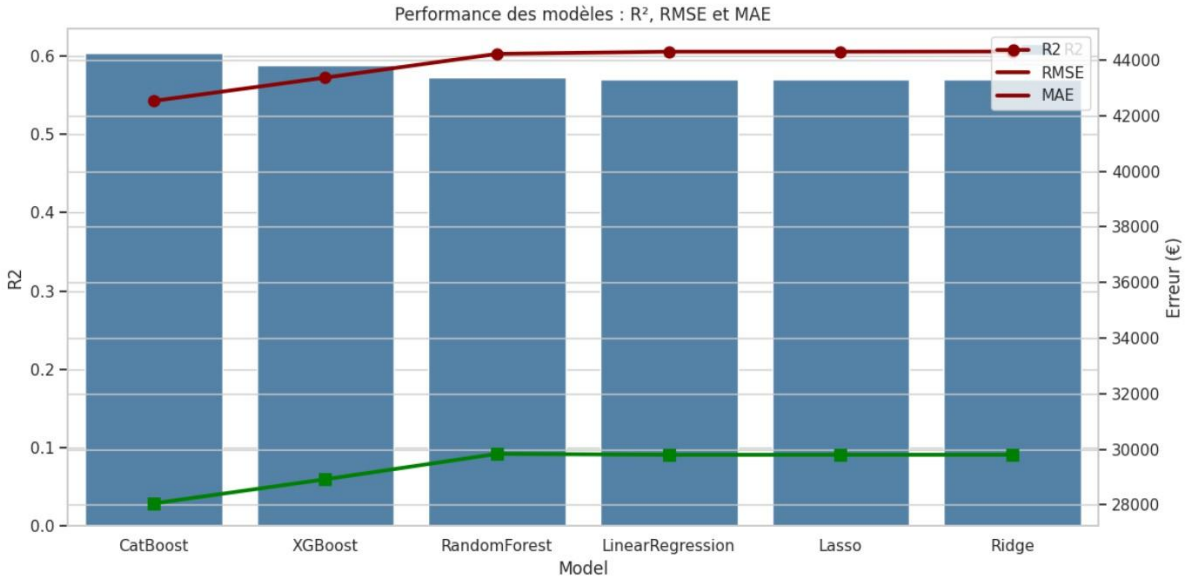
- Les modèles basés sur les arbres (CatBoost, XGBoost, RandomForest) sont **préférables** pour prédire le salaire des développeurs à partir de variables techniques et démographiques.
- Une erreur moyenne de ~28,000 USD (CatBoost) est raisonnable compte tenu de la variabilité naturelle des salaires dans le dataset.
- Ces résultats confirment l'importance des **modèles capables de capturer des relations non linéaires** et des interactions entre plusieurs compétences et caractéristiques professionnelles.

Visualisation comparative des performances des modèles

Pour comparer facilement les modèles de régression, nous avons représenté sur un même graphique les trois métriques principales : R^2, RMSE et MAE.

- Les barres bleues représentent le R^2, indiquant la proportion de variance du salaire expliquée par le modèle.
- Les points rouges et verts représentent respectivement le RMSE et le MAE, donnant l'erreur moyenne des prédictions en dollars.

Cette visualisation montre clairement que **CatBoost est le modèle le plus performant**, suivi de XGBoost et RandomForest, tandis que les modèles linéaires capturent moins bien les variations du salaire.



Nous avons représenté les métriques R^2, RMSE et MAE pour chaque modèle afin de comparer visuellement leurs performances.

- Les barres bleues représentent le R^2, indiquant la proportion de variance des salaires expliquée par le modèle.
- Les points rouges (RMSE) et verts (MAE) indiquent respectivement les erreurs quadratiques et absolues moyennes.

Cette visualisation confirme que **CatBoost est le modèle le plus performant**, capturant le mieux la variabilité des salaires tout en minimisant les erreurs de prédiction. Les modèles d'arbres (XGBoost, RandomForest) restent performants mais légèrement moins précis, tandis que les modèles linéaires présentent des erreurs plus importantes et expliquent moins la variance.

Optimisation des modèles via GridSearchCV

```
# ------------------------------
# XGBoost GridSearch
# ------------------------------
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)

xgb_param_grid = {
    'n_estimators': [200, 400, 600],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.7, 0.85, 1.0],
    'colsample_bytree': [0.7, 0.85, 1.0]
}

xgb_grid = GridSearchCV(
    estimator=xgb_model,
    param_grid=xgb_param_grid,
    scoring='r2',
    cv=3,
    verbose=1,
    n_jobs=-1
)

xgb_grid.fit(X_train, y_train)
print("XGBoost Best params:", xgb_grid.best_params_)
xgb_preds = xgb_grid.predict(X_test)
print("XGBoost R2:", r2_score(y_test, xgb_preds))
xgb_rmse = np.sqrt(mean_squared_error(y_test, xgb_preds))
print("XGBoost RMSE:", xgb_rmse)
```

```
# ------------------------------
# CatBoost GridSearch
# ------------------------------
cat_model = CatBoostRegressor(random_state=42, verbose=0)

cat_param_grid = {
    'iterations': [500, 1000],
    'depth': [4, 6, 8],
    'learning_rate': [0.01, 0.05, 0.1],
    'l2_leaf_reg': [1, 3, 5]
}

cat_grid = GridSearchCV(
    estimator=cat_model,
    param_grid=cat_param_grid,
    scoring='r2',
    cv=3,
    verbose=1,
    n_jobs=-1
)

cat_grid.fit(X_train, y_train)
print("CatBoost Best params:", cat_grid.best_params_)
cat_preds = cat_grid.predict(X_test)
print("CatBoost R2:", r2_score(y_test, cat_preds))
cat_rmse = np.sqrt(mean_squared_error(y_test, cat_preds))
print("CatBoost RMSE:", cat_rmse)
```

```
Fitting 3 folds for each of 243 candidates, totalling 729 fits
XGBoost Best params: {'colsample_bytree': 0.85, 'learning_rate': 0.05, 'max_depth': 5, 'n_estimators': 400, 'subsample': 0.85}
XGBoost R2: 0.6046237101909818
XGBoost RMSE: 42501.199338732076
Fitting 3 folds for each of 54 candidates, totalling 162 fits
CatBoost Best params: {'depth': 8, 'iterations': 500, 'l2_leaf_reg': 3, 'learning_rate': 0.05}
CatBoost R2: 0.6026895140793802
CatBoost RMSE: 42605.031262236545
```

Afin d'améliorer les performances des modèles non linéaires, nous avons effectué une recherche d'hyperparamètres (GridSearchCV) sur XGBoost et CatBoost avec validation croisée à 3 folds.

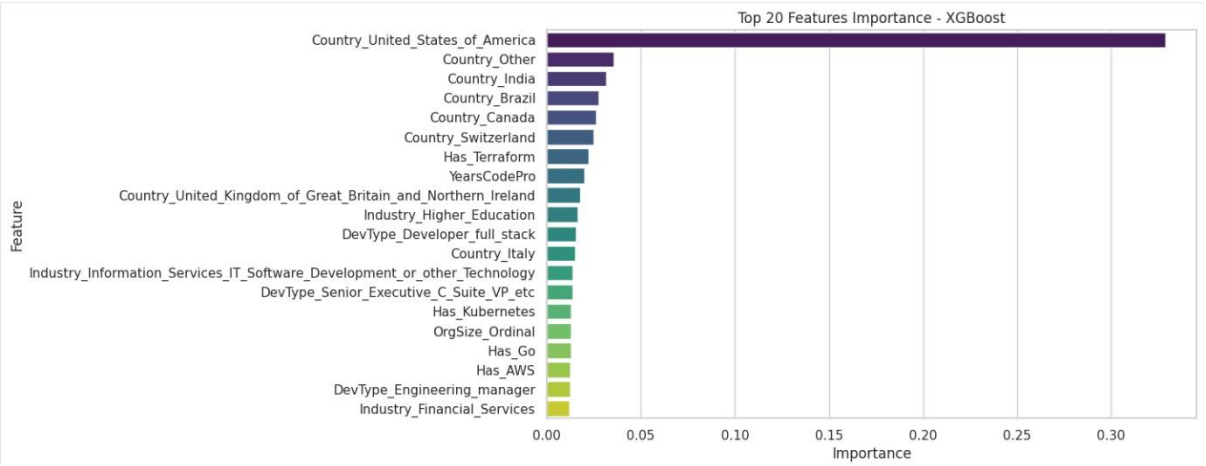
Pour XGBoost, la meilleure combinaison trouvée est :
max_depth = 5, n_estimators = 400, learning_rate = 0.05, subsample = 0.85, colsample_bytree = 0.85.
Ce modèle atteint un R^2 de 0.6046 et un RMSE de 42 501, représentant la meilleure performance obtenue dans cette étude.

Pour CatBoost, les meilleurs hyperparamètres sont :
depth = 8, iterations = 500, learning_rate = 0.05, l2_leaf_reg = 3.
Le modèle présente un R^2 de 0.6027 et un RMSE de 42 605, légèrement inférieur à celui d'XGBoost.

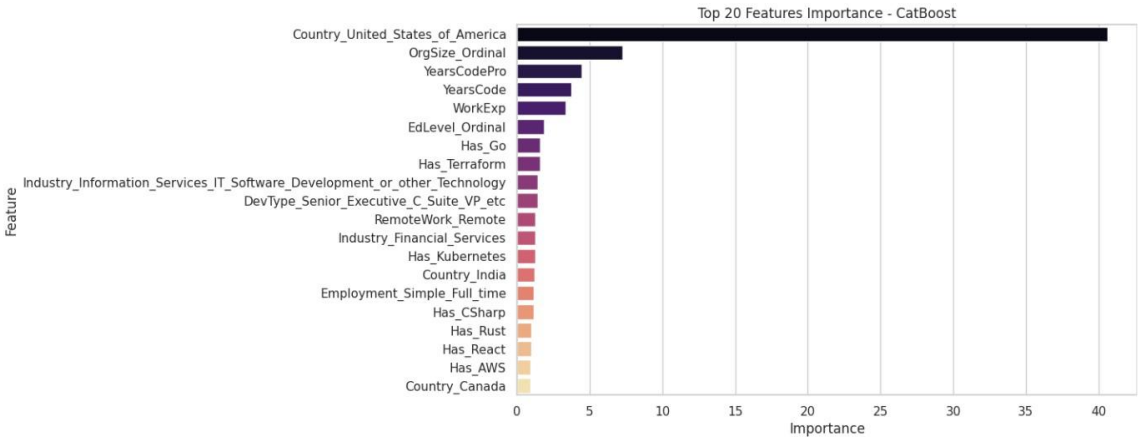
Ainsi, le modèle final retenu pour la prédiction du salaire annuel est XGBoost optimisé, qui offre le meilleur compromis entre précision et robustesse.

Analyse des Features et Performances des Modèles Optimisés

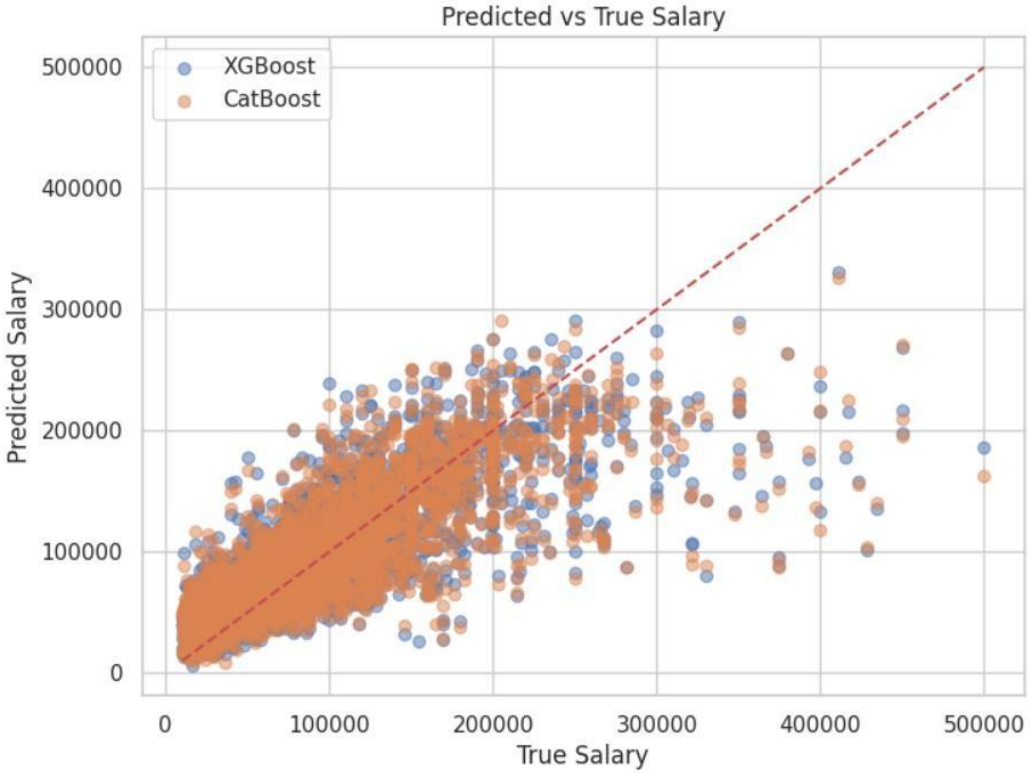
Après l'optimisation des deux modèles, plusieurs visualisations ont été réalisées afin d'analyser leur comportement et d'interpréter les facteurs influençant les prédictions. Tout d'abord, l'importance des variables a été évaluée pour les modèles XGBoost et CatBoost. Les graphiques des vingt caractéristiques les plus importantes montrent que certains attributs contribuent beaucoup plus que d'autres à la prédiction du salaire, confirmant la présence de variables fortement explicatives. Ensuite, un nuage de points comparant les valeurs réelles et les valeurs prédites met en évidence que les deux modèles parviennent globalement à suivre la tendance, bien que certaines dispersions indiquent encore des erreurs de prédiction non négligeables. Enfin, l'histogramme des erreurs montre une distribution centrée autour de zéro pour les deux modèles, ce qui traduit une absence de biais systématique, tout en soulignant que XGBoost présente des erreurs légèrement plus resserrées, cohérentes avec sa meilleure performance globale. Ces visualisations permettent ainsi d'avoir une compréhension plus fine du comportement des modèles et de confirmer la supériorité de la version optimisée d'XGBoost pour la prédiction du salaire annuel.



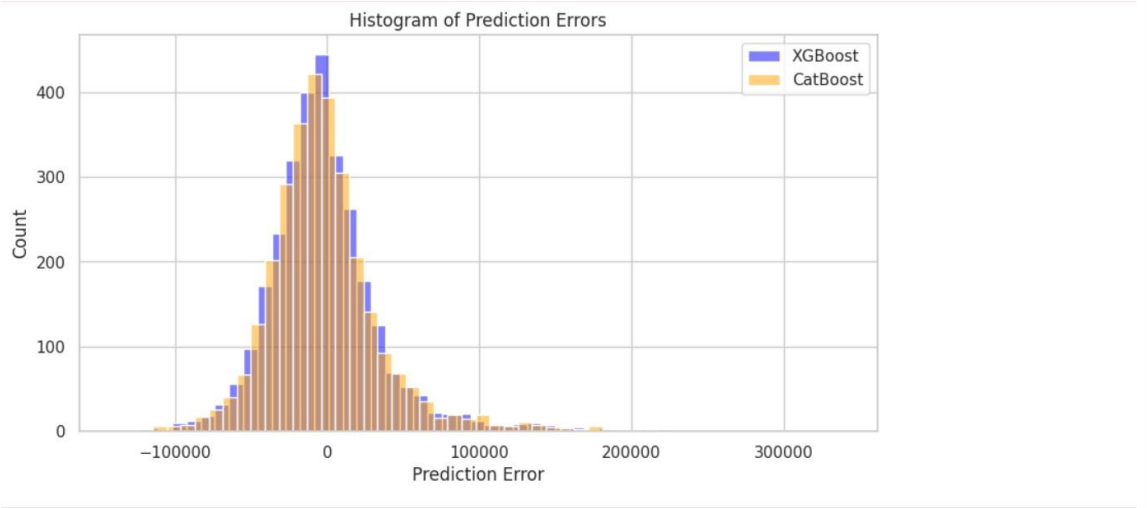
Ce graphique présente les 20 variables ayant la plus grande importance dans la prédiction du salaire selon le modèle XGBoost optimisé. Les barres les plus longues correspondent aux caractéristiques et compétences professionnelles qui contribuent le plus aux décisions du modèle, indiquant ainsi les facteurs les plus déterminants dans l'estimation du salaire annuel.



Ce graphique illustre les 20 variables les plus importantes pour le modèle CatBoost optimisé. Il permet de comparer les facteurs influençant la prédiction du salaire avec ceux mis en évidence par XGBoost et d'identifier les variables clés communes ou spécifiques à chaque modèle, offrant ainsi une meilleure compréhension des mécanismes internes des deux algorithmes.



Ce graphique en nuage de points compare les salaires prédits par les modèles XGBoost et CatBoost aux salaires réels. La ligne diagonale représente une prédiction parfaite : plus les points sont proches de cette ligne, plus la prédiction est précise. La dispersion observée autour de la diagonale met en évidence les écarts entre les valeurs réelles et estimées, permettant ainsi d'évaluer visuellement la performance et la stabilité des deux modèles.



L'histogramme présente la distribution des erreurs de prédiction (résidus) pour les modèles XGBoost et CatBoost. Une distribution centrée autour de zéro indique que les modèles ne présentent pas de biais systématique, tandis que l'étendue et la dispersion des résidus reflètent la variabilité des erreurs et permettent de comparer la précision relative des deux modèles.

Conclusion générale

Dans ce travail, plusieurs étapes d'analyse, de nettoyage et de feature engineering ont été menées afin de construire des modèles de prédiction du salaire annuel des développeurs. Les analyses exploratoires ont mis en évidence la complexité du problème, notamment la forte variabilité des salaires et l'absence de relation linéaire directe entre le nombre de compétences et la rémunération. Ces observations confirment la nécessité d'utiliser des modèles capables de capturer des relations complexes et non linéaires entre les variables.

Dans ce contexte, les modèles **XGBoost** et **CatBoost** se sont révélés être les choix les plus pertinents. Grâce à leur capacité à exploiter efficacement l'information contenue dans le dataset, ils parviennent à modéliser des interactions complexes entre les variables explicatives et à fournir de meilleures performances que les approches plus simples. Les résultats obtenus après l'optimisation des hyperparamètres montrent que ces modèles atteignent un bon compromis entre précision et robustesse, avec des performances très proches et légèrement en faveur de XGBoost.

Enfin, ces modèles présentent un fort potentiel d'amélioration : l'optimisation des hyperparamètres, l'enrichissement des features et l'ajout de nouvelles variables explicatives permettent d'améliorer progressivement leurs performances. Ainsi, **XGBoost et CatBoost constituent des outils adaptés et évolutifs pour la prédiction des salaires**, capables de s'adapter à la complexité des données et de tirer le maximum d'information du dataset étudié.