

YouCode / Yousseoufia

Développeur Data

2023-2024

Projet : Job Analysis

Introduction

En tant que développeur Data, dans le cadre de ce projet nous souhaitons réaliser une analyse approfondie du marché du travail dans ces domaines émergents afin de mieux cibler nos initiatives de recrutement, d'acquisition de talents et de développement de compétences.

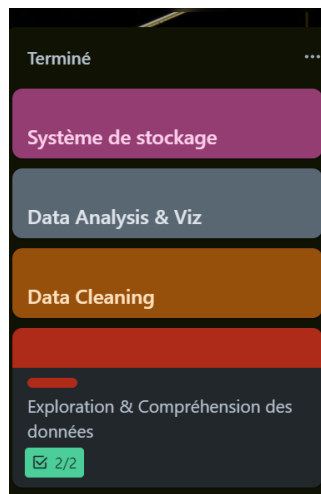
Nous souhaitons également visualiser les résultats de manière claire et informative, permettant ainsi aux parties prenantes de mieux comprendre les conclusions tirées de l'analyse.

De plus, nous cherchons à mettre en place un système de stockage des données pour faciliter la gestion et l'accès aux informations collectées tout au long du projet.

Planification du travail

Pour une bonne gestion et du projet, nous avons divisé le travail en plusieurs tâches avec l'outil Trello.

Le projet était réparti sur quatre (04) axes comme présentés sur la figure ci-dessous :



Exploration & Compréhension des données

Nous allons exploiter un dataset de 3198 lignes et 8 colonnes, de types objet

```
data.shape
```

```
(3198, 8)
```

Nous avons un dataset de 3198 entrées et 8 champs

```
data.dtypes
```

```
Company          object
Job Title         object
Location          object
Job Type          object
Experience level  object
Salary            object
Requirement of the company object
Facilities        object
dtype: object
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3198 entries, 0 to 3197
Data columns (total 8 columns):
#   Column              Non-Null Count  Dtype
---  ---
0   Company              3197 non-null  object
1   Job Title            3197 non-null  object
2   Location              3197 non-null  object
3   Job Type              3197 non-null  object
4   Experience level      2962 non-null  object
5   Salary                3009 non-null  object
6   Requirement of the company 3198 non-null  object
7   Facilities            3198 non-null  object
dtypes: object(8)
memory usage: 200.0+ KB
```

Nous avons 3198 entrées.

- Seules les colonnes "**Facilities**" et "Requirment of company " semblent toutes bien remplies
- Les colonnes "**Company**", "**Job Title**", "**Location**" ont seulement une valeur manquante
- La colonne "**Experience level**" a **2962** entrées **sur 3198**. Donc près **de 93%** de valeurs remplies et **7%** de valeurs manquantes
- La colonne "**Salary**" a **3009** entrées sur **3198**. Donc **94%** de valeurs remplies et **6%** de valeurs manquantes.

Déjà, nous nous rendons compte que le dataset a quelques impuretés. Pour nous assurer de réaliser une analyse correcte, nous allons procéder à un nettoyage du dataset (data cleaning).

```
# Afficher un résumé statistique du DataFrame
summary = data.describe(include='all')
summary
```

	Company	Job Title	Location	Job Type	Experience level	Salary	Requirement of the company	Facilities
count	3197	3197	3197	3197	2962	3009	3198	3198
unique	1106	2138	1117	3	4	218	2600	777
top	Publicis Groupe	Data Engineer	Bengaluru, India	Full Time	Senior-level	115K+ *	Big Data,Business Intelligence,Data analysis,E...	****
freq	126	105	90	3116	1876	253	12	542

Des statistiques résumées sur les différents champs

Data Cleaning (Nettoyage)

Cette étape consiste à nettoyer les données en corrigeant les erreurs, en supprimant les impuretés pour ne conserver que les valeurs propres, correctes, utiles et pertinentes. Nous serons amenés à dans ce projet à :

- Gérer les doublons
- Gérer les valeurs manquantes (Salary)
- Extraction d'information (Location => Country)
- Uniformiser les unités (monnaie)

Gestion des doublons

Gestion des doublons

```
: data.duplicated().sum()
```

```
: 202
```

Nous avons 202 doublons. Nous allons les supprimer pour éviter les conflits de données.

```
: data.drop_duplicates(inplace = True)
data.duplicated().sum()
```

```
: 0
```

Les doublons ont bien été supprimés

Nous allons maintenant procéder au cas par cas avec les colonnes notamment la colonne Salary et Location.

```
data["Salary"].isna().sum()
```

172

La colonne **Salary** a 172 valeurs manquantes. Après une exploration plus approfondie, nous avons trouvé que les devises ne sont pas uniformes.

```
distinct_salary = data["Salary"].unique().tolist()
distinct_salary
```

```
' 59K+',
'EUR 81K',
' 190K+',
' 72K+',
' 136K+',
' 102K+ *',
' 31K+ *',
' 89K+ *',
' 156K+',
' 107K+',
' 199K+',
' 224K+',
' 142K+',
'GBP 35K',
' 183K+',
' 164K+',
'EUR 130K+',
' 174K+',
```

Après avoir vérifié les types de job de valeurs manquantes de la colonne Salary, on se rend compte que la plupart sont des stagiaires. Les stagiaires n'ont généralement pas de salaire. Ce qui semble logique, c'est de remplacer ces valeurs manquantes par 0.

	Company	Job Title	Location	Job Type	Experience level	Salary	Requirment of the company
7	NielsenIQ	Intern (Business Intelligence Service Support)	Bangkok, Thailand	Internship	Entry-level	NaN	Business Intelligence,Excel,Genetics,,
8	Western Digital	Summer 2023 Data Engineering Intern	San Jose, CA, United States	Internship	Entry-level	NaN	Big Data,Computer Science,Engineering,Machine ... Career dev
39	Angi	Senior Data Scientist	Indianapolis, IN - Hybrid	Full Time	Senior-level	NaN	Big Data,Data Mining,Machine Learning,Mathemat... 40 devel
		Junior Data					

Il n'y a pas que les stagiaires, certains professionnels même sénior ont des valeurs manquantes. Une explication plausible 'est qu'ils sont en négociation se salaire. Cependant, pour ne pas biaiser l'analyse avec des valeurs approximatives, nous avons décidé de les mettre dans la même catégorie que les stagiaires (Salaire=0).

```
data["Salary"] = data["Salary"].fillna(0)
data["Salary"].isna().sum()
```

0

Les valeurs manquantes dans la colonne Salary sont désormais gérées. Il reste à uniformiser les devises

```
def convert_to_usd(amount, currency):
    if currency == 'EUR':
        return amount * 1.18
    elif currency == 'GBP':
        return amount * 1.38
    else:
        return amount

def extract_numeric_salary(salary_str):
    try:
        return float(salary_str)
    except ValueError:
        currency_code = salary_str[:3] # Extract the first three characters as currency code
        numeric_salary = salary_str[3:].strip().replace(',', '') # Extract the numeric part and remove commas
        return convert_to_usd(float(numeric_salary), currency_code)
```

En faisant la conversion, nous avons transformé la colonne en valeur numérique décimale comme présenté dans la figure ci-après :

97000.0
92000.0
48000.0
129000.0
92000.0
115000.0
39000.0
44000.0
73000.0
40000.0
59000.0

Ensuite, nous nous sommes attaqués à la colonne Company :

Company

```
missing_values_count = data["Company"].isna().sum()
print("Number of missing values in 'Company' column:", missing_values_count)
```

Number of missing values in 'Company' column: 1

```
data.loc[data["Company"].isna()]
```

	Company	Job Title	Location	Job Type	Experience level	Salary	Requirment of the company	Facilities
797	NaN	NaN	NaN	NaN	NaN	0.0

La ligne n'a aucune information qui soit utile. Nous allons donc la supprimer

En supprimant la ligne, on se rend compte qu'elle contenait les seules valeurs manquantes des colonnes **Job Title**, **Job Type**, **Location**.

Nous nous attaquons à la colonne « **Experience level** » où nous avons remplacé les 227 valeurs manquantes par « **Not Specified** ».

Experience level

```
data["Experience level"].isna().sum()
```

227

Vérifions si tout s'est passé comme souhaité

```
data["Experience level"] = data["Experience level"].fillna("Not Specified")
data.loc[data["Experience level"] == "Not Specified"]
```

	Company	Job Title	Location	Job Type	Experience level	Salary	Requirement of the company	Facilities
2	Cricut	Machine Learning Engineer	South Jordan, UT, United States	Full Time	Not Specified	90000.0	Agile,Architecture,AWS,Computer Science,Comput...	Career development,,,
11	NielsenIQ	Data Analyst - Revenue Optimizer	Toronto, ON, Canada	Full Time	Not Specified	80000.0	Business Analytics,Business Intelligence,Data ...	Career development,Startup environment,,
18	Issuu	BI Analyst	Braga	Full Time	Not Specified	48000.0	Business Analytics,Business Intelligence,Data ...	Competitive pay,Equity,Health care,Insurance,

```
data["Experience level"].isna().sum()
```

0

```
data["Experience level"].unique()
```

```
array(['Entry-level', 'Not Specified', 'Mid-level', 'Senior-level',  
      'Executive-level'], dtype=object)
```

Plus de valeurs manquantes !

Pour la colonne **Location**, plus tard dans l'étape de l'analyse, nous avons extrait les noms des pays pour les stocker dans une autre colonne **Country**.

Data Analysis

Cette étape va consister à trouver des informations cachées dans le dataset. Et pour y arriver, nous essayerons de trouver réponse à des questions jugées pertinentes et intéressantes.

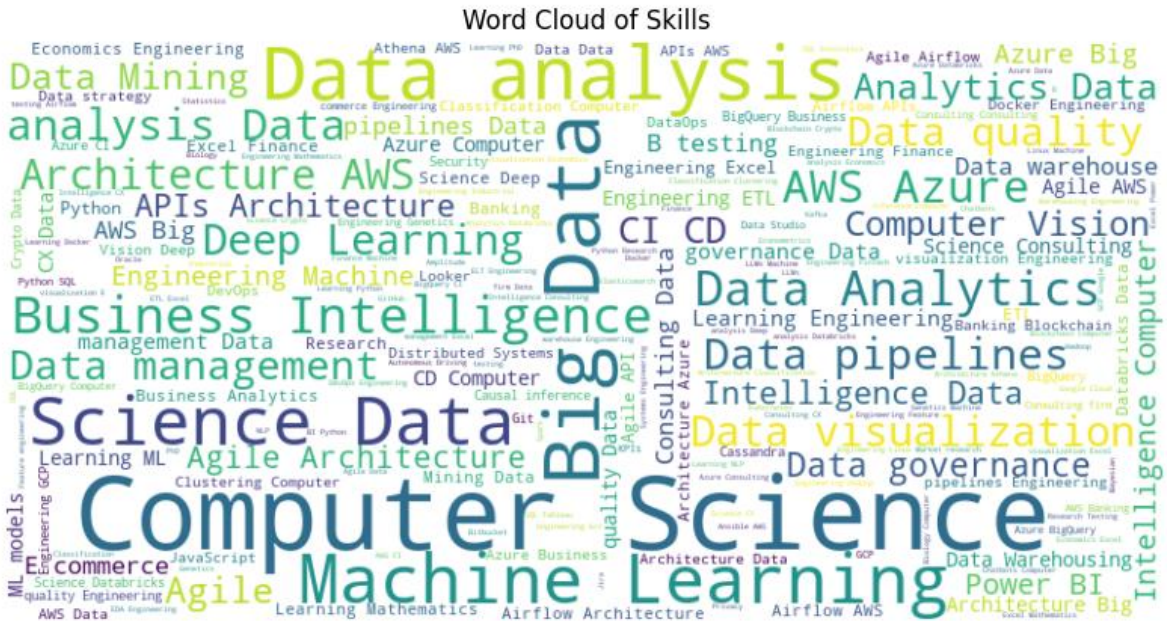
L'objectif est de :

- ✓ Identifier les tendances clés du marché de l'emploi en IA, DS et Big Data.
- ✓ Mettre en évidence les compétences les plus demandées et les titres de poste les plus courants dans ces domaines.
- ✓ Analyser les différences et les similitudes entre les opportunités d'emploi en IA, DS et Big Data.

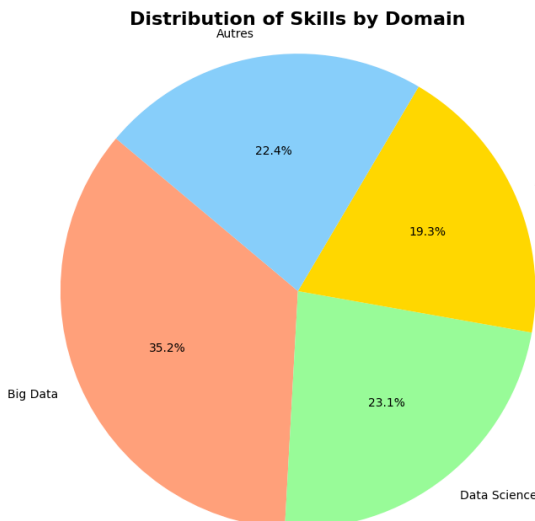
Pour cela, nous nous sommes posés un certain nombre de questions :

En affichant le word Cloud de la colonne **Requirment of the company**, on peut facilement se rendre compte que les compétences les plus demandées de façon générales sont :

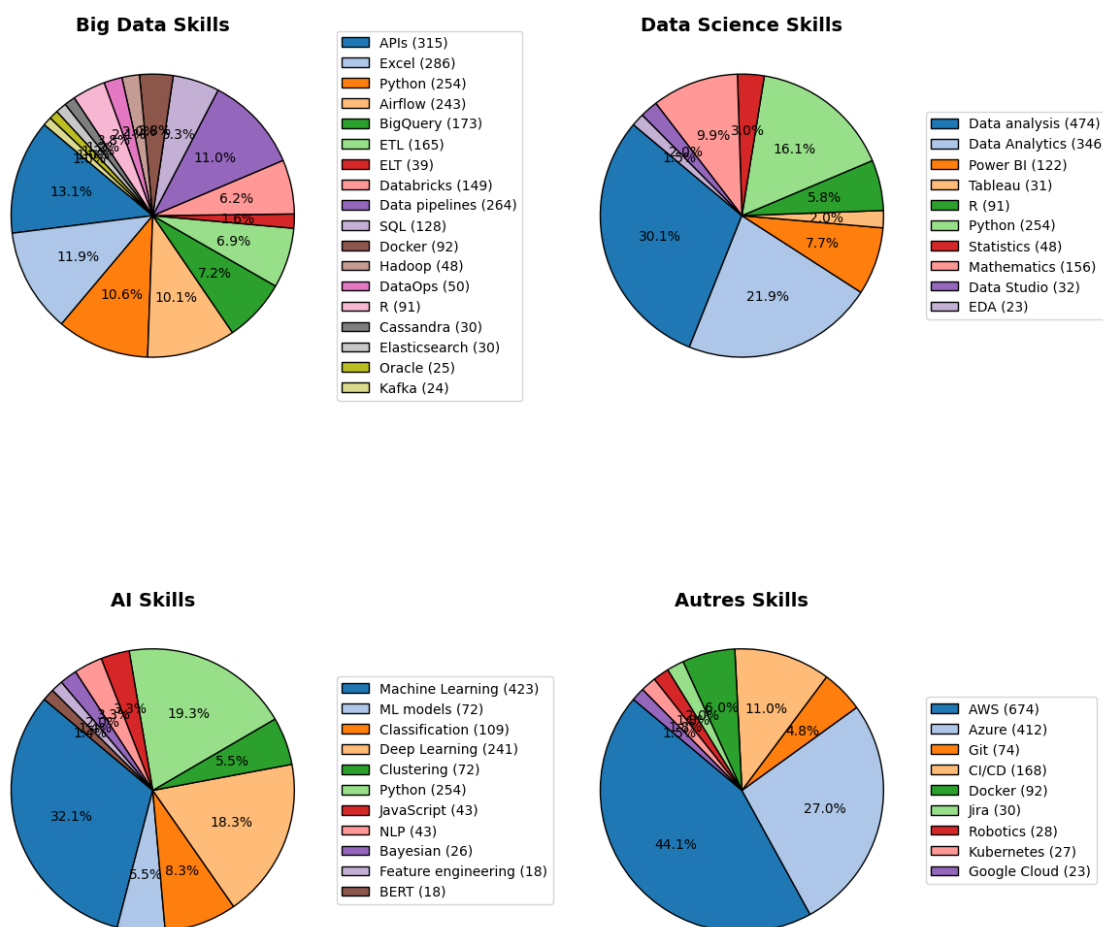
- ✓ Computer Science
- ✓ Data Analysis
- ✓ Big Data
- ✓ Machine Learning
- ✓ Business Intelligence entre autres



Mais lorsqu'on analyse les 100 compétences les plus demandées, on s'aperçoit du pourcentage de chaque domaine.



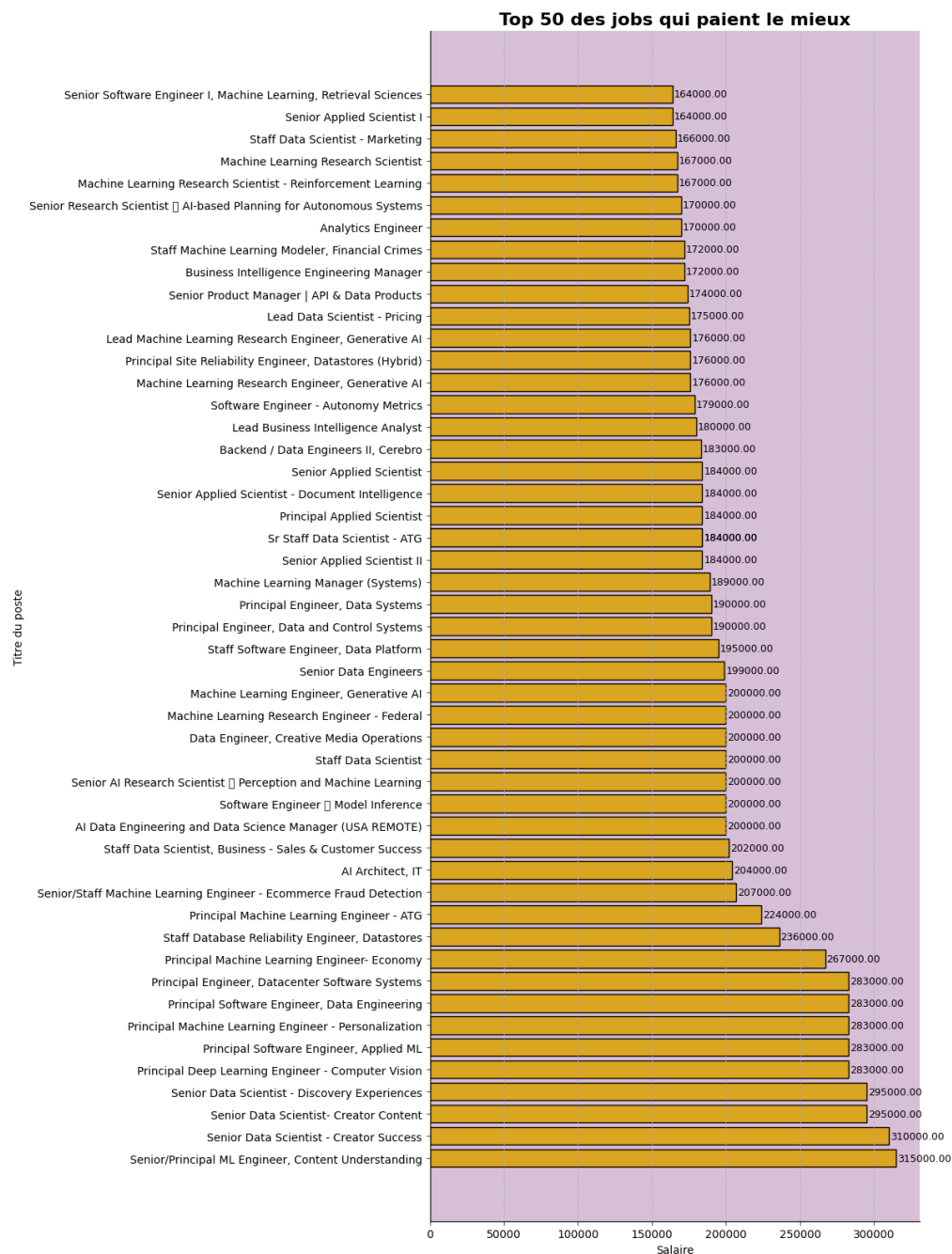
Avec un zoom sur chaque domaine, on détecte les compétences les plus demandées dans chaque domaine :



Quels sont les jobs les mieux les payés ?

Nous allons pour cela vérifier les jobs associés aux salaires les plus élevés

Les jobs associés au top 50 des salaires les plus élevés sont répertoriés dans ce graphique



Dans quels pays se situent les compagnies qui paient le mieux ?

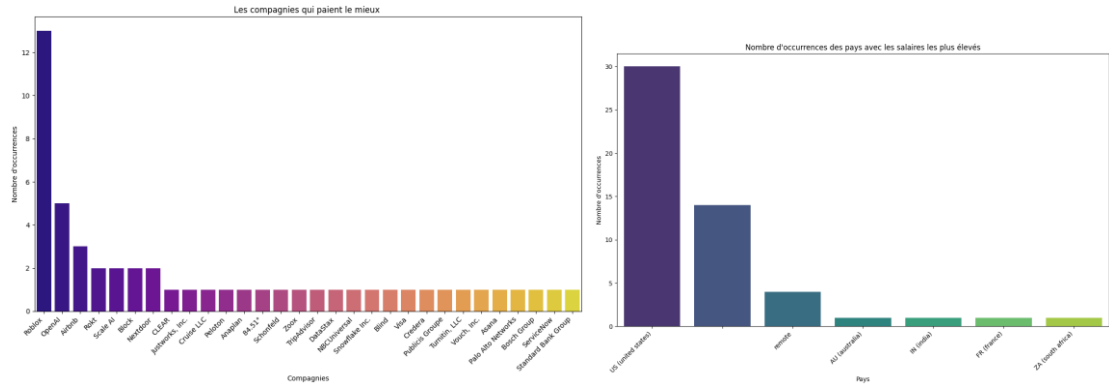
Quelles sont les compétences recherchées par ces compagnies ?

Ces compagnies sont entre autres

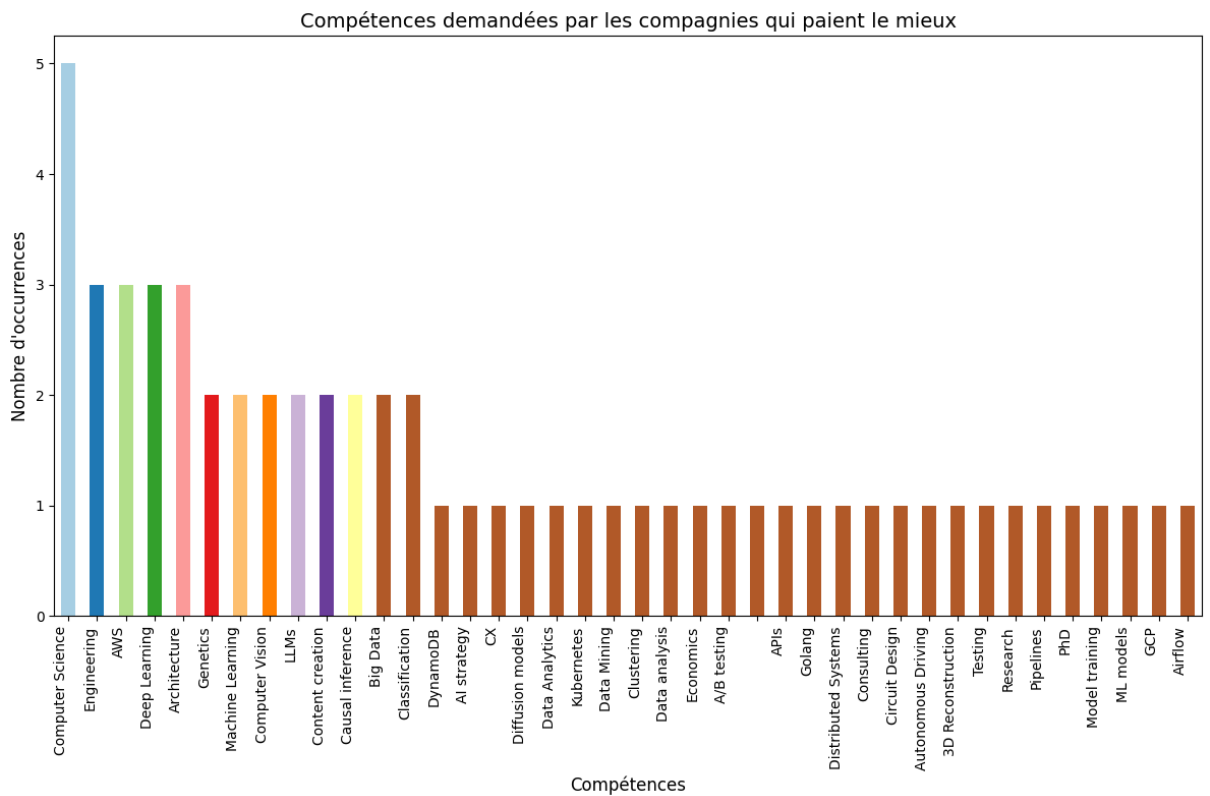
- ✓ Roblox
- ✓ OpenAI
- ✓ Airbnb

Et les pays associés sont entre autres :

- ✓ Les Etats-Unis
- ✓ L'Australie
- ✓ L'Inde
- ✓ La France



Les compétences demandées sont les suivantes :



Quels sont les pays qui paient le mieux ?

On fait une comparaison des pays selon le salaire moyen, on obtient :

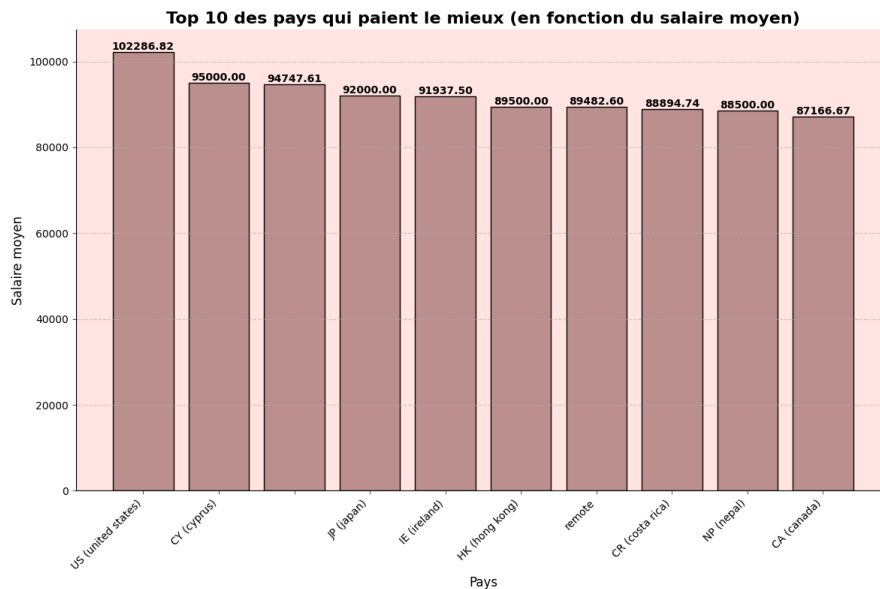
Pays selon la valeur du salaire



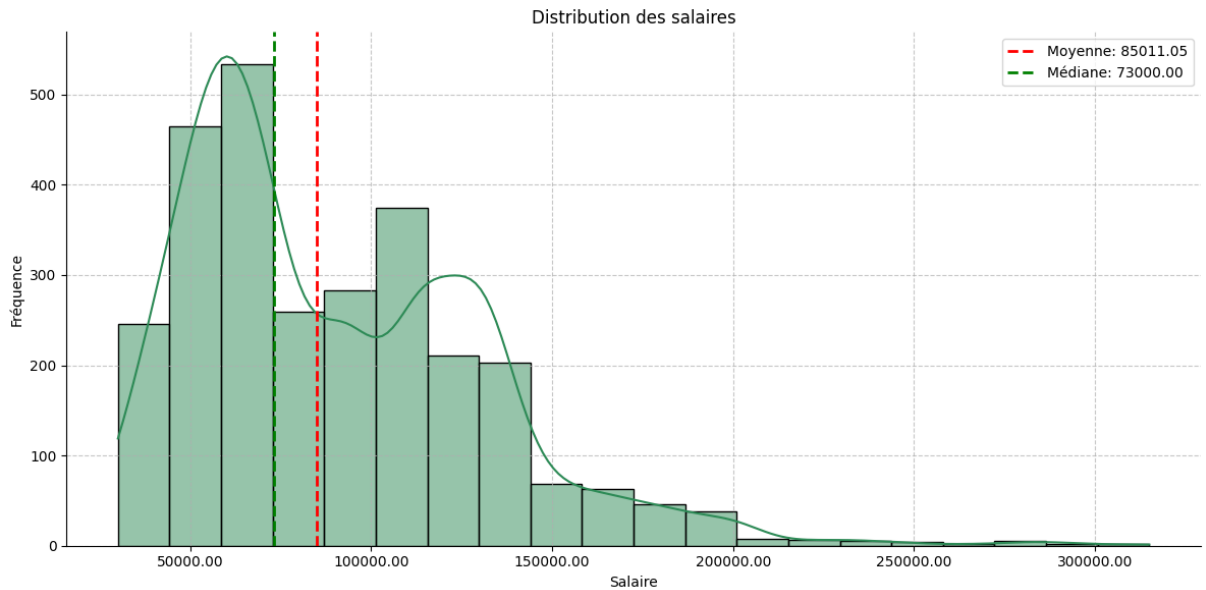
Parmi ces pays, on voit clairement :

- ✓ Les Etats-Unis
- ✓ Le Japon
- ✓ Costa Rica etc

Le top 10 de ces pays sont les suivants :



La distribution des salaires

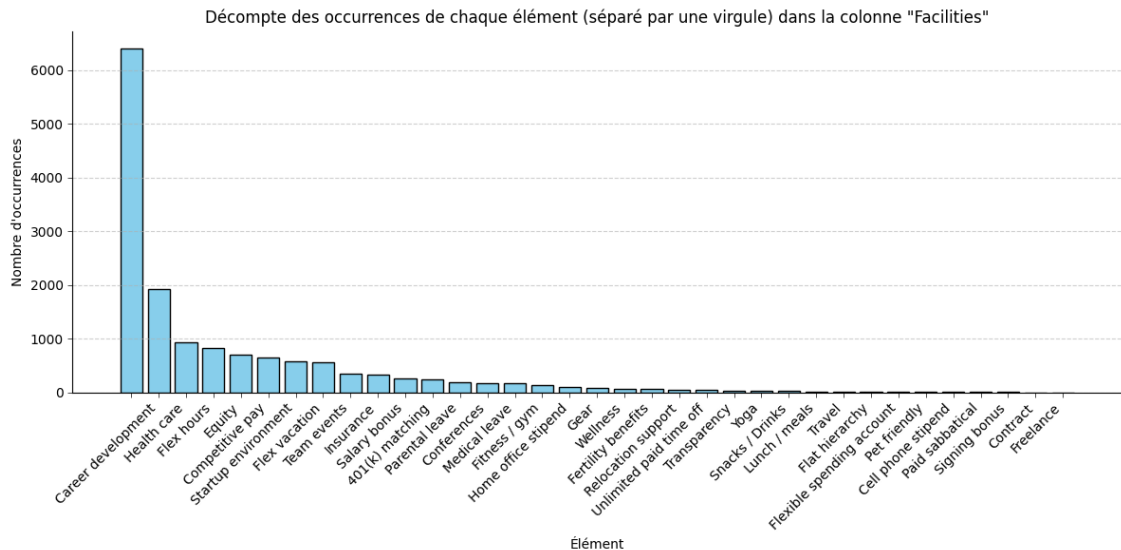


On est ici en présence d'une distribution bimodale. Elle indique que la distribution des salaires n'est pas normale. Deux pics se présentent pour révéler qu'il y a en fait deux catégories de salaires. Les très bons salaires et les salaires moyens. On remarque également que les salariés moyens et ceux qui ont les plus gros salaires sont presque à nombre égal.

Quelles sont les promesses des entreprises ?

Les promesses les plus fréquentes des entreprises sont exprimées à travers ce graphe parmi lesquelles nous avons :

- career development
- Health care
- Flex hours
- Equity
- Competitive pay
- Startup environment
- Flex vacation



Tout ceci nous permet de conclure qu'en plus du salaire, pour attirer des employés, il faut aussi proposer des possibilités de développement personnel et de carrière.

Nous allons à présent stocker nos données dans une base de données. Pour cela, nous allons procéder en trois étapes (MCD-MLD-MPD)

Data Modeling

❖ Modèle Conceptuel de Données

Les entités qui ont été détectées sont les suivantes :

- ✓ **Company** (company_id, company_name)
- ✓ **Job** (job_id, job_title, experience_id, company_id, type_id, salary_id, requirement_id, isremote, facility_id, country_id)
- ✓ **Experience** (experience_id, level)
- ✓ **Type** (type_id, type)
- ✓ **Salary** (salary_id, salary)
- ✓ **Country** (country_id, country_name)
- ✓ **Requirement** (requirement_id, requirement)
- ✓ **Facility** (facility_id, facility)

❖ Modèle Logique des Données

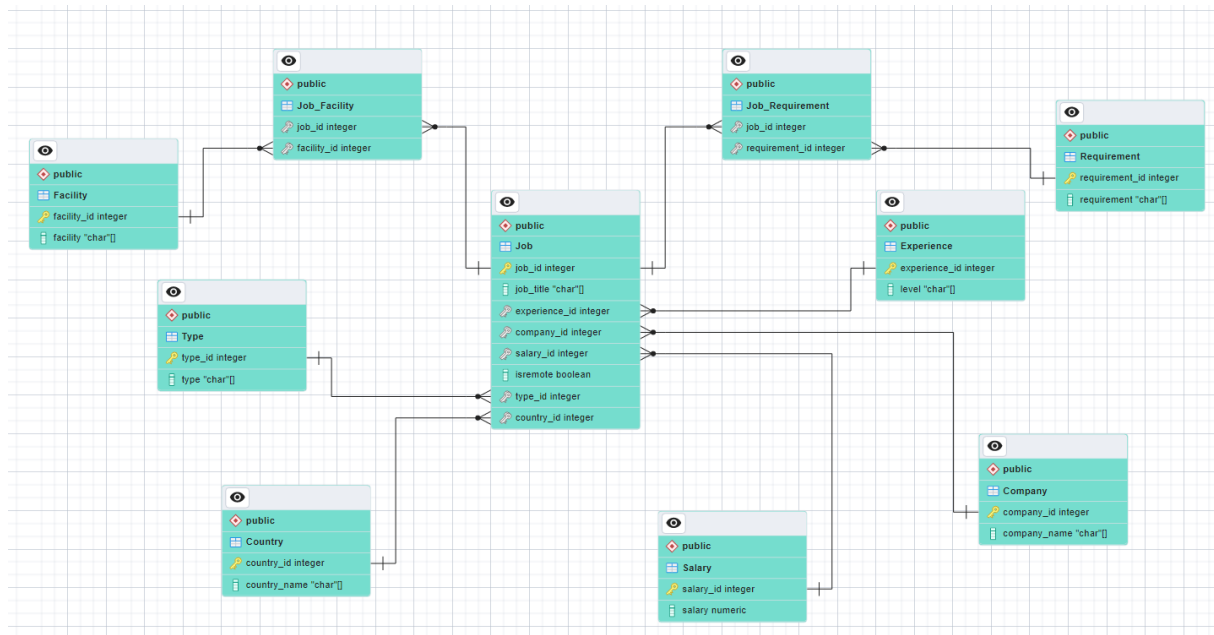
- Une compagnie peut avoir plusieurs offres jobs d'emploi (**one-to-many**) (**Company_Job**)
- Un niveau d'expérience peut être associé à plusieurs **jobs** (**one-to-many**) (**Job_Experience**)
- Un pays peut être relié à plusieurs **jobs** (**one-to-many**) (**Job_Country**)
- Un type de job peut être associé à plusieurs jobs (**one-to-many**) (**Job_Type**)
- Un salaire peut être associé à plusieurs jobs (**one-to-many**) (**Job_Salary**)

- Une compétence peut être associée à plusieurs jobs et plusieurs jobs peuvent être associés à une compétence (**many-to-many**) (**Job_Requirement**)
- Un avantage peut associer à plusieurs jobs et inversement (**many-to-many**) (**Job_Facility**)

Pour implémenter les relations **many-to-many**, nous allons créer des tables-relations (**Job_Requirement**, **Job_Facility**)

- ✓ **Job_Requirement** (job_id, job_requirement)
- ✓ **Job_Facility** (job_id, facility_id)

Ce qui nous permet de réaliser ce diagramme d'entité-relation



❖ Modèle Physique des Données

Dans cette dernière étape, nous avons créé la base de données et toute sa structure (tables) puis avons inséré les données correspondantes.

Le SGBD choisi est SQL Server. Pour effectuer la connexion avec la base de Données, nous avons utilisé **pyodbc**.

La connexion s'est faite grâce au code suivant :

```

: connexion = pyodbc.connect(
    Trusted_Connection='Yes',
    Driver='{ODBC Driver 17 for SQL Server}',
    Server='LAPTOP-L0N961TQ\SQLEXPRESS',
    Database='Jobs'
)
print("Connexion to database succeeded!")
cursor = connexion.cursor()

```

Connexion to database succeeded!

Creation de la table Company

```

]: try:
    # Create the Company table
    cursor.execute('''
        CREATE TABLE Company (
            company_id INT PRIMARY KEY IDENTITY(1,1),
            company_name NVARCHAR(100)
        )
    ''')

    # Commit the changes to the database
    connexion.commit()

    print("The Company table has been created successfully!")

except Exception as e:
    # Print the error message if an exception occurs
    print("Error:", e)

```

The Company table has been created successfully!

Insertion dans la table Company

```
# Insertion dans la table Company
companies = data["Company"].dropna().unique()
for company in companies:
    # Check if the company already exists in the table
    cursor.execute("SELECT company_name FROM Company WHERE company_name = ?", (company,))
    if not cursor.fetchone():
        # Insert the company into the table
        cursor.execute("INSERT INTO Company (company_name) VALUES (?)", (company,))

# Commit the changes to the database
connexion.commit()

print("Data has been inserted into the Company table.")
```

Data has been inserted into the Company table.

Creation de la table Experience

```
# Create the Experience table
cursor.execute('''
    CREATE TABLE Experience (
        experience_id INT PRIMARY KEY IDENTITY(1,1),
        level VARCHAR(100)
    )
''')
connexion.commit()
print("The Experience table has been created successfully!")
```

The Experience table has been created successfully!

Insertion dans la table Experience

```
# Insertion dans la table Company
levels = data["Experience level"].dropna().unique()
for level in levels:
    cursor.execute("SELECT level FROM Experience WHERE level = ?", (level,))
    if not cursor.fetchone():
        cursor.execute("INSERT INTO Experience (level) VALUES (?)", (level,))
connexion.commit()
print("Data has been inserted into the Experience table.")
```

Data has been inserted into the Experience table.

Création de la table Country

```
# Create the Country table
cursor.execute('''
    CREATE TABLE Country (
        country_id INT PRIMARY KEY IDENTITY(1,1),
        country_name NVARCHAR(50)
    )
''')
connexion.commit()
print("The Country table has been created successfully!")
```

The Country table has been created successfully!

Insertion dans la table Country

```
: # Insertion dans la table Country
country_names = data["Country"].dropna().unique()
for country_name in country_names:
    cursor.execute("SELECT country_name FROM Country WHERE country_name = ?", country_name)
    if not cursor.fetchone():
        cursor.execute("INSERT INTO Country (country_name) VALUES (?)", country_name)
connexion.commit()
print("Data has been inserted into the Country table.")
```

Data has been inserted into the Country table.

Création de la table Type

```
# Create the Type table
cursor.execute('''
    CREATE TABLE Type (
        type_id INT PRIMARY KEY IDENTITY(1,1),
        type NVARCHAR(50)
    )
''')
connexion.commit()
print("The Type table has been created successfully!")
```

The Type table has been created successfully!

Insertion dans la table Type

```
# Insertion dans la table Type
job_types = data["Job Type"].dropna().unique()
for job_type in job_types:
    cursor.execute("SELECT type FROM Type WHERE type = ?", job_type)
    if not cursor.fetchone():
        cursor.execute("INSERT INTO Type (type) VALUES (?)", job_type)
connexion.commit()
print("Data has been inserted into the Type table.")
```

Data has been inserted into the Type table.

Création de la table Salary

```
# Create the Salary table
cursor.execute('''
    CREATE TABLE Salary (
        salary_id INT PRIMARY KEY IDENTITY(1,1),
        salary FLOAT
    )
''')
connexion.commit()
print("The Salary table has been created successfully!")
```

The Salary table has been created successfully!

Insertion dans la table Salary

```
# Insertion dans la table Salary
salaries = data["Salary"].dropna().unique()
for salary in salaries:
    cursor.execute("SELECT salary FROM Salary WHERE salary = ?", salary)
    if not cursor.fetchone():
        cursor.execute("INSERT INTO Salary (salary) VALUES (?)", salary)
connexion.commit()
print("Data has been inserted into the Salary table.")
```

Data has been inserted into the Salary table.

Création de la table Requirement

```
# Create the Requirement table
cursor.execute('''
    CREATE TABLE Requirement (
        requirement_id INT PRIMARY KEY IDENTITY(1,1),
        requirement NVARCHAR(500)
    )
''')
connexion.commit()
print("The Requirement table has been created successfully!")
```

The Requirement table has been created successfully!

```
# Modify the column "requirement" to increase its length to 500 characters
cursor.execute("ALTER TABLE Requirement ALTER COLUMN requirement NVARCHAR(500)")
connexion.commit()
```

Insertion dans la table Requirement

```
: # Créez un dictionnaire vide pour stocker Les valeurs uniques de chaque colonne
unique_values_dict = {}

# Parcourez chaque colonne du dataset
for column in data.columns:
    # Sélectionnez les valeurs uniques de la colonne en utilisant la méthode unique()
    unique_values = data[column].dropna().unique()

    # Utilisez un ensemble (set) pour stocker les valeurs uniques
    unique_values_set = set(unique_values)

    # Ajoutez les valeurs uniques dans le dictionnaire avec le nom de la colonne comme clé
    unique_values_dict[column] = unique_values_set

# Affichez les éléments uniques pour chaque colonne
for column, unique_values in unique_values_dict.items():
    print(f"Colonnes '{column}':")
    for value in unique_values:
        print(f"    - {value}")
    print() # Ajoutez une ligne vide pour séparer les résultats de chaque colonne
```

```

# Loop through unique_values and insert them into the Requirement table
for column_unique_values in unique_values_dict.values():
    for unique_value in column_unique_values:
        # Convert unique_value to a string before insertion
        unique_value_str = str(unique_value)

        # Replace "schema_name" with the actual schema name if applicable
        cursor.execute("SELECT requirement FROM Requirement WHERE requirement = ?", unique_value_str)

        if not cursor.fetchone():
            cursor.execute("INSERT INTO Requirement (requirement) VALUES (?)", unique_value_str)

# Commit the changes to the database
connexion.commit()

print("Data has been inserted into the Requirement table!")

```

Data has been inserted into the Requirement table!

Création de la table Facility


















```

# Create the Facility table
cursor.execute('''
    CREATE TABLE Facility (
        facility_id INT PRIMARY KEY IDENTITY(1,1),
        facility NVARCHAR(500)
    )
''')
connexion.commit()
print("The Requirement table has been created successfully!")

```

The Requirement table has been created successfully!

Ainsi de suite pour toutes les autres tables

- [-]  Jobs
 - [+]  Database Diagrams
 - [-]  Tables
 - [+]  System Tables
 - [+]  FileTables
 - [+]  External Tables
 - [+]  Graph Tables
 - [+]  dbo.Company
 - [+]  dbo.Country
 - [+]  dbo.Experience
 - [+]  dbo.Facility
 - [+]  dbo.Job
 - [+]  dbo.Job_Facility
 - [+]  dbo.Job_Requirement
 - [+]  dbo.Requirement
 - [+]  dbo.Salary
 - [+]  dbo.Type