

Projet Databricks : Transformation des Données en Zones Bronze, Silver et Gold

Ce document récapitule toutes les étapes effectuées, de l'importation des données jusqu'à leur analyse dans la zone Gold.

Contexte :

Nous avons récupéré une base de données regroupant des informations sur l'insertion professionnelle des jeunes ayant terminé une alternance dans différentes régions françaises. Cette base inclut des statistiques sur les taux d'embauche, les secteurs d'activité, et les localisations géographiques sur un échantillon couvrant plus de 10 000 villes. Les données proviennent de deux sources principales :

1. **DARES** : Données sur l'insertion des jeunes après la voie professionnelle, fournissant des indicateurs tels que les taux d'emploi six mois après la formation, les taux de poursuite d'études, et les disparités régionales. Ces informations sont disponibles sur le site suivant :
<https://dares.travail-emploi.gouv.fr/donnees/insertion-des-jeunes-apres-la-voie-professionnelle>
2. **BMO 2019 (Pôle emploi)** : Méthodologie et statistiques des besoins en main-d'œuvre dans les différentes régions françaises, mettant en lumière les métiers et secteurs en tension ainsi que les perspectives d'embauche. Ces données sont accessibles à l'adresse suivante :
https://statistiques.pole-emploi.org/bmo/static/methode_2019

Ingestion des données avec Airbyte

Nous avons choisi d'utiliser Airbyte en tant que condition à remplir pour notre projet car il nous nous sommes rendus compte que la plupart des autres conditions ne pouvaient pas être remplies car nous étions limités par le fait que nous utilisions la version communautaire de Databricks. Nous avons rencontré aussi beaucoup de problèmes pour trouver une source avec airbyte pour transférer les données vers le compte de stockage azure. En effet, nous avons d'abord utilisé Airtable mais nous nous sommes rendus compte que le nombre de ligne était limité à 50 000 par jeux de données importés, ce qui dans notre cas était un problème. Nous avons donc décidé de choisir un connecteur simple, le connecteur basique File qui permet de transférer des fichiers csv, json parquet etc...). Le problème que nous avons rencontré est que les fichiers que nous transférons grâce à cette méthode était sauvegardé dans le conteneur au format "append blob", un format qui ne permettait pas d'importer le fichier dans databricks, nous avons vu que lorsque le fichier avait ce format il n'était pas reconnu par notre

monture databricks. Nous avons alors tout simplement décider de choisir le compte de stockage comme source puisque c'était la seule qui fonctionnait, nous avons donc importer les fichiers csv de nos données puis nous avons utilisé airbyte pour les transférer vers un autre conteneur appelé "airbytecontainer".

Connections

8 paused

Search

All connectionsAll statusesAll sourcesAll destinations

NAME	SOURCE NAME	DESTINATION NAME	FREQUENCY	LAST SYNC	ENABLED
Azure Blob Storage → Azure Blob Storage	Azure Blob Storage	Azure Blob Storage	Manual	Il y a 2 jours	
File (CSV, JSON, Excel, Feather, Parquet) → Azure Blob Storage	File (CSV, JSON, Excel, Feather, Parquet)	Azure Blob Storage	Manual	Il y a 9 jours	
File (CSV, JSON, Excel, Feather, Parquet) → Azure Blob Storage	File (CSV, JSON, Excel, Feather, Parquet)	Azure Blob Storage	Manual	Il y a 9 jours	
File (CSV, JSON, Excel, Feather, Parquet) → Azure Storage	File (CSV, JSON, Excel, Feather, Parquet)	Azure Blob Storage	Manual	Il y a 9 jours	
File (CSV, JSON, Excel, Feather, Parquet) → Azure Blob Storage	File (CSV, JSON, Excel, Feather, Parquet)	Azure Blob Storage	Manual	Il y a 9 jours	
File (CSV, JSON, Excel, Feather, Parquet) → Azure Blob Storage	File (CSV, JSON, Excel, Feather, Parquet)	Azure Blob Storage	Manual	Il y a 9 jours	
File (CSV, JSON, Excel, Feather, Parquet) → Azure Blob Storage	File (CSV, JSON, Excel, Feather, Parquet)	Azure Blob Storage	Manual	Il y a 9 jours	
File (CSV, JSON, Excel, Feather, Parquet) → Azure Blob Storage	File (CSV, JSON, Excel, Feather, Parquet)	Azure Blob Storage	Manual	Il y a 9 jours	
File (CSV, JSON, Excel, Feather, Parquet) → Azure Blob Storage	File (CSV, JSON, Excel, Feather, Parquet)	Azure Blob Storage	Manual	Il y a 9 jours	

☐

\$logs

09/12/2024 09:58:43

Privé

Disponible

☐

airbytecontainer

29/12/2024 16:12:45

Privé

Disponible

☐

data

30/12/2024 20:26:47

Privé

Disponible

☐

ds-bronze

15/12/2024 18:21:46

Privé

Disponible

☐

ds-gold

09/12/2024 10:01:46

Privé

Disponible

☐

ds-silver

09/12/2024 10:01:39

Privé

Disponible

1. Importation des données dans la zone Bronze

Objectif : Créer un stockage intermédiaire contenant les données brutes sans transformation.

```
from azure.storage.blob import BlobServiceClient, BlobClient, ContainerClient

connection_string = f"DefaultEndpointsProtocol=https;AccountName=dlkefreiarnaud;AccountKey={access_key};EndpointSuffix=core.windows.net"
source_container_name = "data"
destination_container_name = "ds-bronze"

blob_service_client = BlobServiceClient.from_connection_string(connection_string)
source_container_client = blob_service_client.get_container_client(source_container_name)
destination_container_client = blob_service_client.get_container_client(destination_container_name)

# List all blobs in the "data" container with the prefix "BMO"
blobs_list = source_container_client.list_blobs(name_starts_with="BMO")

# Copy each BMO CSV file to the "ds-bronze/BMO/" folder
for blob in blobs_list:
    if blob.name.endswith(".csv"): # Ensure only CSV files are copied
        destination_blob_name = f"BMO/{blob.name}" # Destination path in ds-bronze/BMO/

        source_blob_url = f"https://{blob_service_client.account_name}.blob.core.windows.net/{source_container_name}/{blob.name}"

        destination_blob_client = destination_container_client.get_blob_client(destination_blob_name)
        destination_blob_client.start_copy_from_url(source_blob_url)

        print(f"File copied from {blob.name} to {destination_blob_name}")

# Verify the copy operation
print("Verifying copied files in ds-bronze/BMO:")
copied_blobs = destination_container_client.list_blobs(name_starts_with="BMO/")
for blob in copied_blobs:
    print(blob.name)
```

Étapes :

- Les fichiers CSV associés aux besoins en main-d'œuvre (BMO) sont copiés depuis le conteneur source **data** vers le conteneur de destination **ds-bronze**.
- Cette opération permet de garantir la traçabilité des données brutes avant tout traitement ou transformation.

Script utilisé :

Un script Python utilisant la bibliothèque Azure Blob Storage a été développé pour l'importation des fichiers :

- Les fichiers CSV avec le préfixe BMO sont identifiés dans le conteneur source.
- Chaque fichier est copié dans le dossier **ds-bronze/BMO/** du conteneur de destination.

2. Transformation et nettoyage des données dans la zone Silver

Objectif : Unifier, enrichir et nettoyer les données pour garantir leur qualité en vue d'une utilisation dans la zone Gold et pour la visualisation BI.

2.1 Union des données sur la main d'œuvre

- **Création du schéma :**

Nous avons défini un schéma standardisé pour toutes les tables, ce qui inclut la création et la normalisation des noms de colonnes afin d'uniformiser les données en provenance de différentes sources.

```
BMO2019 = "/mnt/ds-bronze/BMO/BMO2019.csv"
BMO2020 = "/mnt/ds-bronze/BMO/BMO2020.csv"
BMO2021 = "/mnt/ds-bronze/BMO/BMO2021.csv"
BMO2022 = "/mnt/ds-bronze/BMO/BMO2022.csv"
BMO2023 = "/mnt/ds-bronze/BMO/BMO2023.csv"
BMO2024 = "/mnt/ds-bronze/BMO/BMO2024.csv"

BMOschema = StructType([
    StructField('annee', IntegerType(), True),
    StructField('BE', IntegerType(), True),
    StructField('NOMBE', StringType(), True),
    StructField('Famille_met', StringType(), True),
    StructField('Lbl_fam_met', StringType(), True),
    StructField('Code_metier', StringType(), True),
    StructField('Nom_metier', StringType(), True),
    StructField('Dept', StringType(), True),
    StructField('NomDept', StringType(), True),
    StructField('met', IntegerType(), True),
    StructField('xmet', IntegerType(), True),
    StructField('smet', StringType(), True),
    StructField('REG', IntegerType(), True),
    StructField('NOM_REG', StringType(), True)
])
```

- **Fusion des tableaux avec la méthode "union" :**

Les tableaux ont été fusionnés en utilisant la méthode **UNION**, qui permet d'agréger les données tout en supprimant les doublons.

```
# Define the correct column order and names
columns = [
    'annee', 'BE', 'NOMBE', 'Famille_met', 'Lbl_fam_met',
    'Code_metier', 'Nom_metier', 'Dept', 'NomDept',
    'met', 'xmet', 'smet', 'REG', 'NOM_REG'
]

# Align columns in each DataFrame
df2019_aligned = df2019.select(columns)
df2020_aligned = df2020.select(columns)
df2021_aligned = df2021.select(columns)
df2022_aligned = df2022.select(columns)
df2023_aligned = df2023.select(columns)
df2024_aligned = df2024.select(columns)

# Perform the union
BMO = df2019_aligned.union(df2020_aligned) \
    .union(df2021_aligned) \
    .union(df2022_aligned) \
    .union(df2023_aligned) \
    .union(df2024_aligned)
```

2.2 Création et enrichissement des indicateurs

- **Création d'une table regroupant le nom de l'établissement et son identifiant :**
Une table intermédiaire a été générée pour organiser les établissements par leur numéro UAI (Unité Administrative Immatriculée) et leur nom.

```
catalogue_etablissements = indicateurs.select(indicateurs[0], indicateurs[1]).distinct()
display(catalogue_etablissements)
```

▶ (2) Spark Jobs

▶ catalogue_etablissements: pyspark.sql.dataframe.DataFrame = [n°UAI de l'établissement: string, Libellé de l'établissement: string]

	A ^B _C n°UAI de l'établissement	A ^B _C Libellé de l'établissement
1	0161223T	Organisme de formation-Centre de formation d'apprentis de AFTRAL Puymoyen
2	0333400S	organisme de formation-centre de formation d'apprentis - universit de bordeaux
3	0442621K	Organisme de formation-Centre de formation d'apprentis de Centre tude formation industrielle - Saint Nazaire
4	0550939X	Greta CFA Lorraine Ouest
5	0573780Z	Organisme de formation-Centre de formation d'apprentis Institut de management commercial ARTEMYS de Metz
6	0791005N	Maison Familiale Rurale SEVREUROPE - Bressuire
7	0831771Z	Organisme de formation-Centre de formation d'apprentis Tte d'affiche Frjus
8	0132976P	Greta-CFA Provence
9	0142433U	Organisme de formation-Centre de formation d'apprentis CPNLT Clecy
10	0251829C	AFTRAL CFA transport et Logistique de Franche-Comté
11	0311717Y	Organisme de formation - Centre de formation d'apprentis Commerce et Services
12	0400780F	Organisme de formation-Centre de formation d'apprentis des Industries du Bois
13	0711103N	CFA de Saône et Loire
14	0751510J	Centre de Formation d'apprentis - Alimentation et Htellerie
15	9741754N	Organisme formation-Centre de formation d'Apprentis Atouts Services Entreprises et Formation Le Port

1,764 rows | 2.60 seconds runtime

- **Ajout des géodonnées pour la visualisation :**

Nous avons utilisé un script Python pour intégrer des coordonnées géographiques aux établissements. Ce script repose sur un **endpoint API gouvernemental** pour localiser les 1764 établissements de formation sur une carte.

```
1 container_name = "ds-silver"
2 file_path = f"wasbs://{container_name}@{storage_name}.blob.core.windows.net/extracted_school_data_v2.csv"
3 spark.conf.set(
4     f"fs.azure.account.key.{storage_name}.blob.core.windows.net",
5     f"{access_key}"
6 )
7 indicateurs_geodata = spark.read.csv(file_path, header=True, inferSchema=True)
8 display(indicateurs_geodata)
```

- Une fois le script exécuté, les données géographiques ont été importées dans Azure et intégrées à la zone Silver.

2.3 Identification des formations les plus proposées

Pour mieux comprendre les formations dispensées, nous avons créé un tableau montrant les formations les plus proposées parmi tous les établissements, en nous basant sur le tableau "indicateurs".

```
from pyspark.sql.functions import col, count, dense_rank
from pyspark.sql.window import Window

indicateurs_rank = indicateurs.groupBy(indicateurs[5]) \
    .agg(count("*").alias("count")) \
    .withColumn("rank", dense_rank().over(Window.orderBy(col("count").desc())))

display(indicateurs_rank)
```

▸ indicateurs_rank: pyspark.sql.dataframe.DataFrame = [Libellé de la formation: string, count: long ... 1 more field]

	Libellé de la formation	count	rank
1	maintenance des vehicules option a voitures particulieres	367	1
2	patissier	300	2
3	coiffure	300	2
4	management commercial operationnel	298	3
5	cuisine	281	4
6	macon	276	5
7	negociation et digitalisation de la relation client	272	6
8	boulangier	256	7
9	gestion de la pme	234	8
10	commerce	217	9
11	esthetique cosmetique parfumerie	215	10
12	peintre applicateur de revetements	206	11
13	reparation des carrosseries	196	12
14	carreleur mosaiste	190	13
15	boucher	189	14

2.4 Analyse de l'employabilité et nettoyage des régions


Les contrats d'apprentissage favorisent-ils l'employabilité sur le marché du travail ? Cette question a été étudiée à l'aide d'une analyse approfondie des données BMO et **indicateurs**. Avant cela, des étapes de nettoyage et de standardisation des données régionales ont été réalisées.

- **Suppression des valeurs NULL dans la colonne région**

Pour garantir la fiabilité des données, toutes les entrées avec une valeur manquante dans la colonne **NOM_REG** ont été supprimées.

```
# Supprimer les lignes avec des valeurs nulles dans la colonne NOM_REG
BMO_reg_nullout = BMO_union.filter(col("NOM_REG").isNotNull())

# Vérifier le nombre de lignes après la suppression
print(f"Nombre de lignes après suppression des valeurs nulles : {BMO_reg_nullout.count()}")
```

►  BMO_reg_nullout: pyspark.sql.dataframe.DataFrame = [annee: string, BE: string ... 12 more fields]

Nombre de lignes après suppression des valeurs nulles : 298033

- **Standardisation des noms de régions**

Une fonction utilisateur (UDF) a été créée pour normaliser les noms de régions à partir d'un dictionnaire de correspondance. Cela garantit une uniformité dans les analyses suivantes.

```
# Define a UDF to map regions
@udf(StringType())
def standardize_region(region_name):
    return REGION_MAPPING.get(region_name, region_name)

# Apply the transformation to the "BMO" table
BMO_region_append = BMO_reg_nullout.withColumn(
    "Standardized_Region",
    standardize_region(col("NOM_REG")) # Replace "Region" with the actual column name in "BMO"
)

# Apply the transformation to the "indicateurs" table
indicateurs_region_append = indicateurs.withColumn(
    "Standardized_Region",
    standardize_region(col("Région")) # Replace "Region" with the actual column name in "indicateurs"
)
```

- **Suppression des anciennes colonnes obsolètes**

Après standardisation, les anciennes colonnes **NOM_REG** et **Région** ont été supprimées pour nettoyer les tableaux.

```
# Function to delete old region columns
def delete_old_column(df, column_to_delete="Region"):
    if column_to_delete in df.columns:
        df = df.drop(column_to_delete)
        print(f"✅ Column '{column_to_delete}' has been deleted.")
    else:
        print(f"⚠️ Column '{column_to_delete}' does not exist in the DataFrame.")
    return df

# Delete the old region columns in both BMO and indicateurs
BMO_std_region = delete_old_column(BMO_region_append, column_to_delete="NOM_REG")
indicateurs_std_region = delete_old_column(indicateurs_region_append, column_to_delete="Région")
```

2.5 Création d'une clé primaire

Une clé primaire a été créée pour chaque établissement en combinant le numéro UAI et le code formation apprentissage. Cette opération permet de garantir des jointures précises et d'éviter des produits cartésiens.

```
from pyspark.sql import functions as F

indicateurs_pk = indicateurs_std_region.withColumn(
    "unique_key",
    F.concat_ws("_", "n°UAI de l'établissement", "Code formation apprentissage")
)

# Remove duplicates
indicateurs_pk = indicateurs_pk.dropDuplicates(["unique_key"])

# Drop the old columns from the same (updated) dataframe
indicateurs_pk = indicateurs_pk.drop(
    "n°UAI de l'établissement",
    "Code formation apprentissage"
)

# Now display
display(indicateurs_pk)
```


2.6 Secteurs d'emploi les plus recruteurs

Pour identifier les secteurs d'emploi les plus dynamiques, les données ont été regroupées par `Lbl_fam_met` (famille de métiers). Des indicateurs clés comme le nombre total de projets, les projets difficiles et les projets saisonniers ont été calculés.

```
bmo_job_agg = (  
  BMO_pk  
  .groupBy("Lbl_fam_met")  
  .agg(  
    F.sum("met").alias("nombre_de_projets"),  
    F.sum("xmet").alias("nombre_de_projets_difficiles"),  
    F.sum("smet").alias("nombre_de_projets_saisonniers") # or whichever columns you have  
  )  
)
```

```
display(bmo_job_agg)
```

Table ▾ +				
	⌕ Lbl_fam_met	1.2 nombre_de_projets	1.2 nombre_de_projets_difficiles	1.2 nombre_de_projets_saisonniers
1	Autres métiers	1499667	685839	934175
2	Fonctions d'encadrement	871201	347573	158661
3	Autres techniciens et employés	192645	124137	9690
4	Ouvriers des secteurs de l'industrie	622030	352188	126956
5	Fonctions administratives	479955	188409	67488
6	Fonctions liées à la vente, au tourisme et aux servic...	2897420	1378569	1017421
7	Ouvriers de la construction et du bâtiment	509346	339460	49832
8	Fonctions sociales et médico-sociales	923155	462372	305091

3. Zone Gold : Visualisation et analyses stratégiques

Objectif : La zone Gold représente les données prêtes pour l'analyse avancée et la visualisation. Les analyses dans cette section explorent l'impact des contrats d'apprentissage sur l'employabilité, les critères influençant la rupture de contrat, et la répartition géographique des établissements et formations.

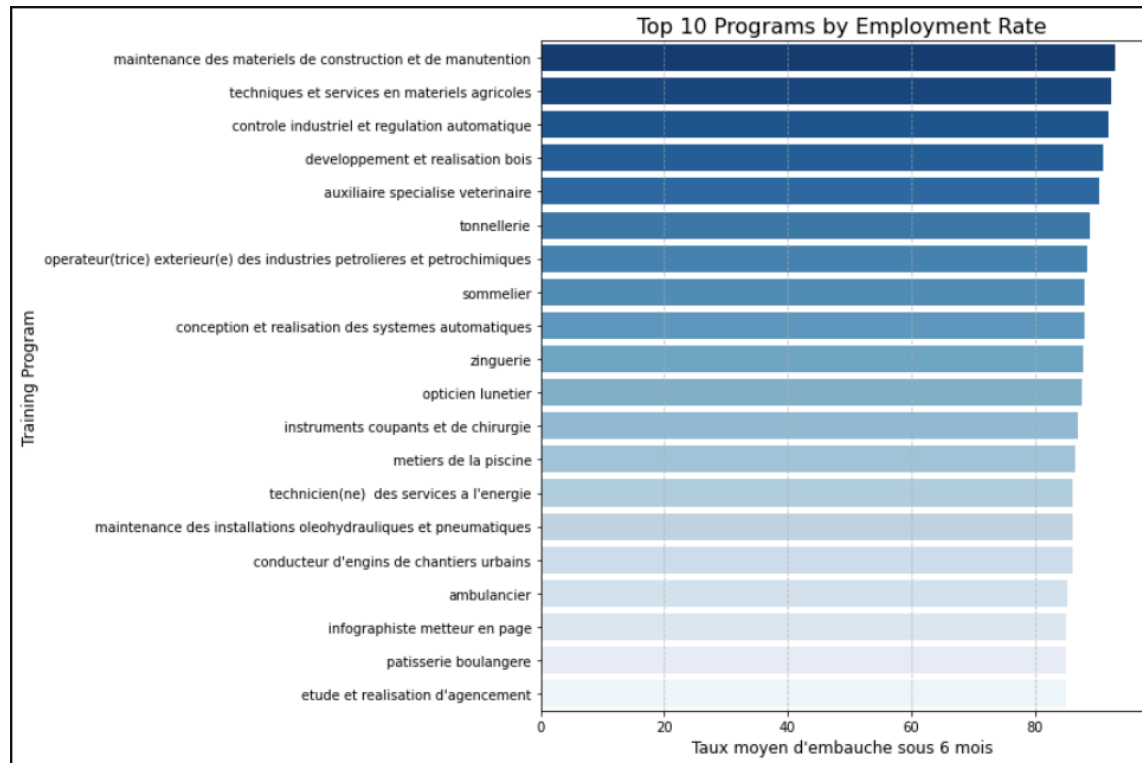
3.1 Les contrats d'apprentissage favorisent-ils l'employabilité ?

- Une analyse des taux d'emploi six mois après la sortie des programmes de formation a été réalisée pour identifier les programmes les plus efficaces.
- Il n'y a pas une grande différence dans le secteur mais nous pouvons voir que par exemple les étudiants en maintenance des matériels de construction et manutention s'en sortent globalement mieux que les pâtisseries / boulangers.

```
data["Taux d'emploi 6 mois après la sortie"] = pd.to_numeric(data["Taux d'emploi 6 mois après la sortie"], errors='coerce')

# Group by training program and calculate the average employment rate
top_10_programs = (
    data.groupby("Libellé de la formation")["Taux d'emploi 6 mois après la sortie"]
        .mean()
        .sort_values(ascending=False)
        .head(20)
        .reset_index()
)

# Plot the top 10 programs using Seaborn
plt.figure(figsize=(12, 8))
sns.barplot(
    data=top_10_programs,
    x="Taux d'emploi 6 mois après la sortie",
    y="Libellé de la formation",
    palette="Blues_r"
)
```



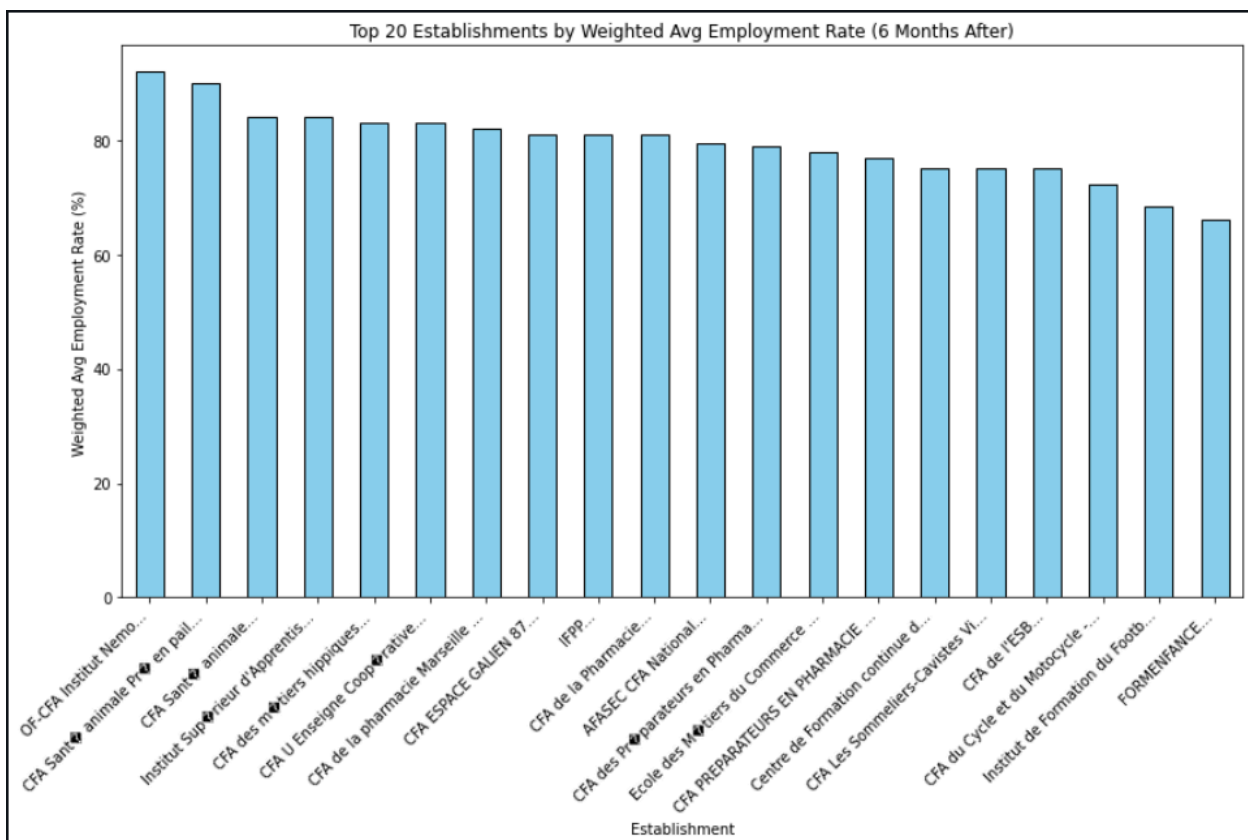
3.2 Analyse des établissements les plus performants

- Analyse des établissements ayant les taux d'emploi les plus élevés, pondérés par le nombre de programmes proposés.
- Les différences ne sont pas énormes mais entre l'Institut Nemo et Formenfance il y a quand-même environ 25% de différence.

```
# Calculate weighted average 'Taux d'emploi 6 mois après la sortie' per establishment
weighted_avg_employment_rate = (
    data.groupby("Short_Establishment_Name")
    .apply(lambda group: (group["Taux d'emploi 6 mois après la sortie"] * group["Program_Count"]).sum() / group["Program_Count"].sum())
    .sort_values(ascending=False)
    .head(20) # Top 20 establishments
)

# Visualization of weighted average employment rates
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
weighted_avg_employment_rate.plot(kind="bar", color='skyblue', edgecolor='black')
plt.title("Top 20 Establishments by Weighted Avg Employment Rate (6 Months After)")
plt.xlabel("Establishment")
plt.ylabel("Weighted Avg Employment Rate (%)")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

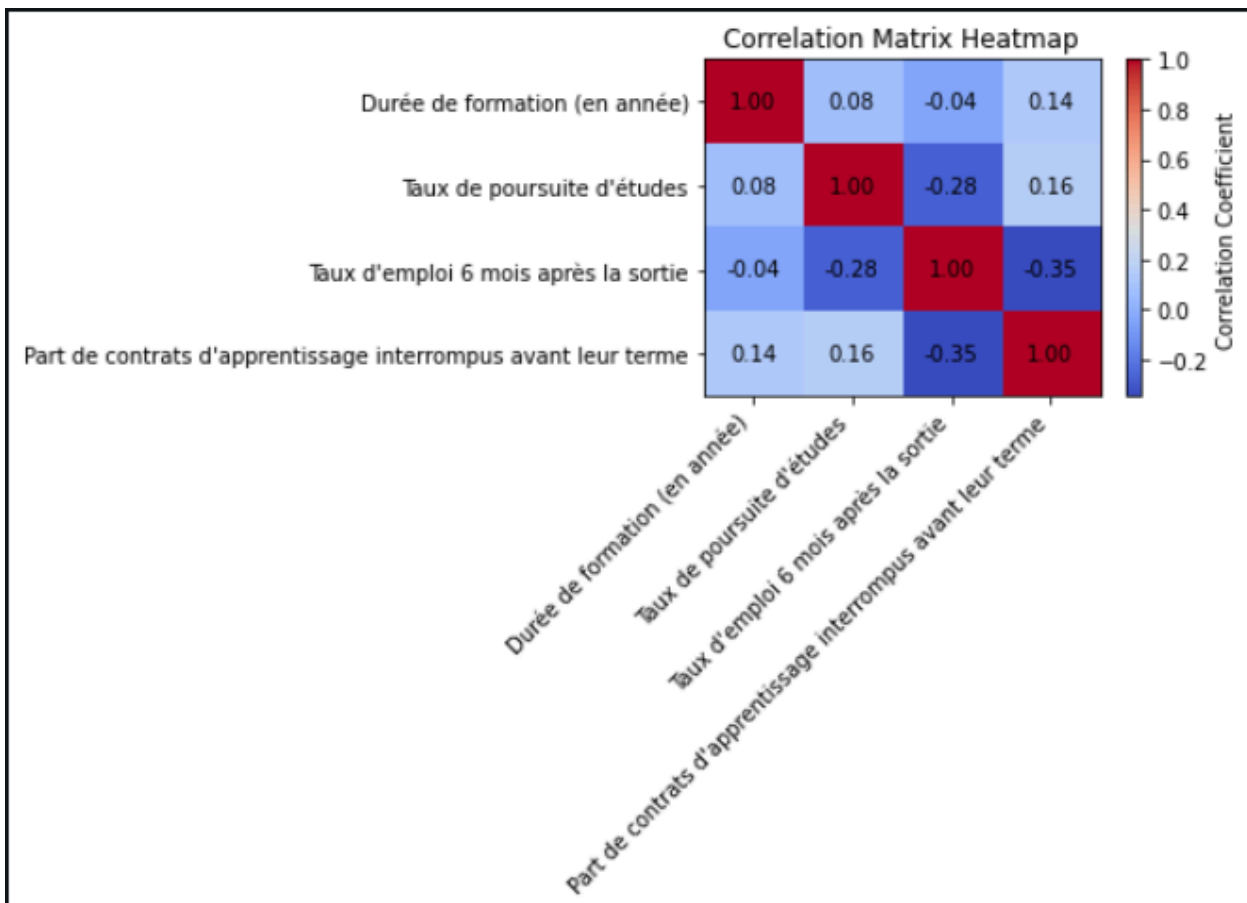


3.3 Quels critères favorisent la rupture d'un contrat d'apprentissage ?

- Une matrice de corrélation a été utilisée pour explorer les relations entre les variables clés comme la durée de formation, le taux d'emploi, et la part de contrats interrompus.
- Les corrélations ne sont pas énormes mais nous pouvons quand même conclure que l'année de formation et le taux de poursuite d'études ont une certaine influence sur le choix de rupture du contrat.

```
# Calculate the correlation matrix using pandas
correlation_matrix = correlation_data.corr()

# Plot the correlation heatmap
plt.figure(figsize=(8, 6))
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='none', aspect='auto')
plt.colorbar(label='Correlation Coefficient')
```



3.4 Répartition géographique des établissements

- Visualisation des établissements sur une carte interactive, en utilisant leurs coordonnées géographiques.
- Les établissements de formation peuvent se trouver partout en France mais en zoomant nous pouvons voir qu'il y a une beaucoup plus grande concentration en région parisienne.

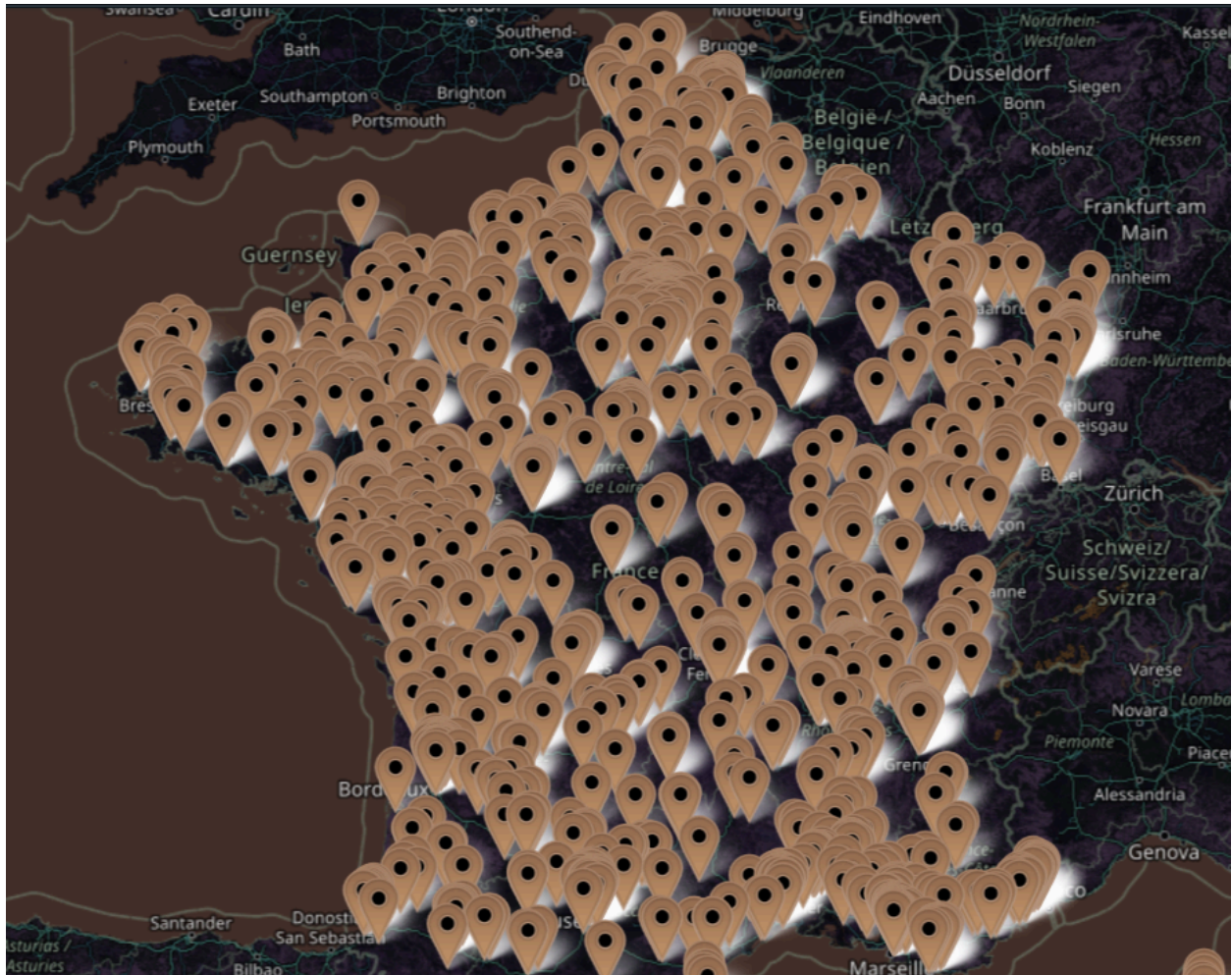
```
# Filter out invalid rows
data = data[data["Geo Coordinates"].apply(is_valid_coordinate)]

# Create a map centered at France
map_france = folium.Map(location=[46.603354, 1.888334], zoom_start=6)

# Add markers for each school using the DataFrame
for _, row in data.iterrows():
    coords = list(map(float, row["Geo Coordinates"].split(',')))
    folium.Marker(location=coords, popup=f"{row['School ID']} - {row['Locality']}").add_to(map_france)

# Display the map
map_france.save("Schools_Map.html")
display(map_france)
```

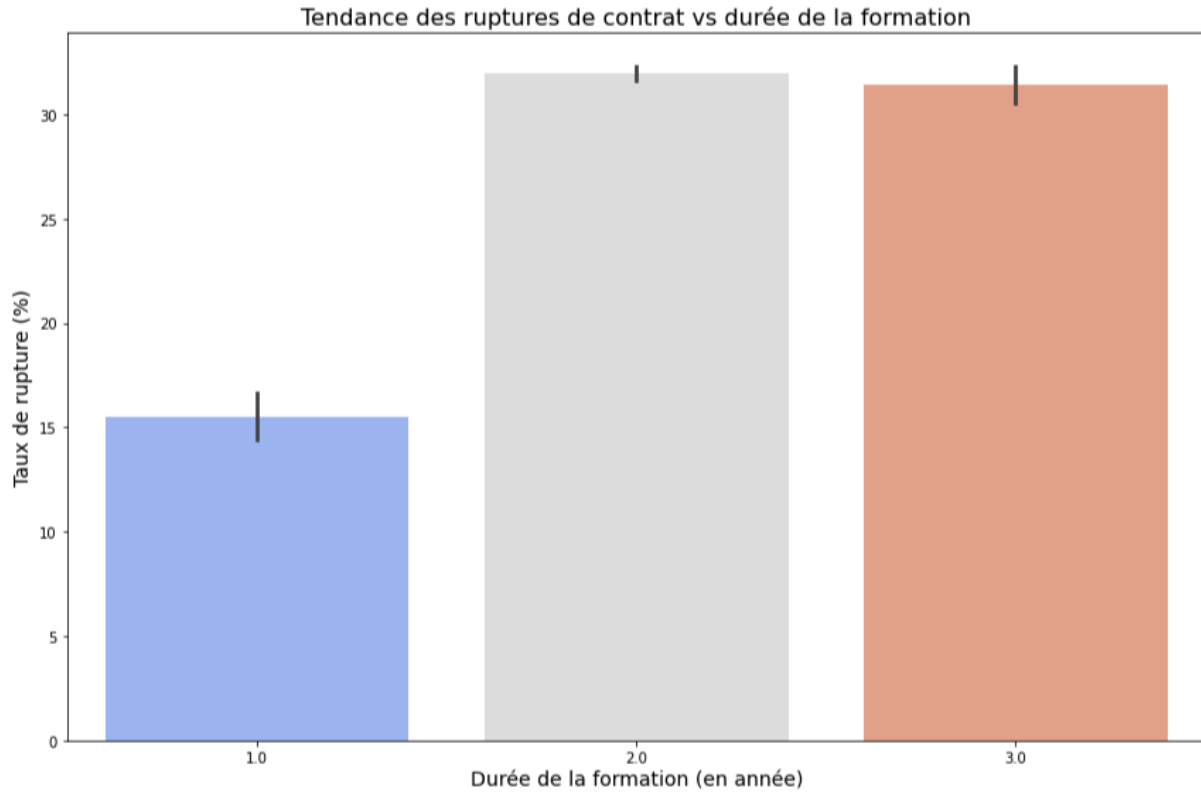
- Une carte interactive montrant les établissements avec leurs localités



3.5 Tendance des ruptures de contrat en fonction de leur durée

- Diagramme à barres qui permet de mettre en valeur la différence de ruptures de contrat en fonction de chaque année.
- Les données s'arrêtant au niveau 5 nous avons 3 années d'études supérieures et on peut voir que peu de monde ne rompt le contrat lors de la première année mais cette valeur double en deuxième et troisième année.

```
plt.figure(figsize=(12, 8))
sns.barplot(
    data=data.dropna(subset=['Durée de formation (en année)', 'Part de contrats d\'apprentissage interrompus avant leur terme']),
    x='Durée de formation (en année)',
    y='Part de contrats d\'apprentissage interrompus avant leur terme',
    palette="coolwarm"
)
plt.title('Tendance des ruptures de contrat vs durée de la formation', fontsize=16)
plt.xlabel('Durée de la formation (en année)', fontsize=14)
plt.ylabel('Taux de rupture (%)', fontsize=14)
plt.tight_layout()
plt.show()
```



3.6 Taux d'emploi par niveau de formation

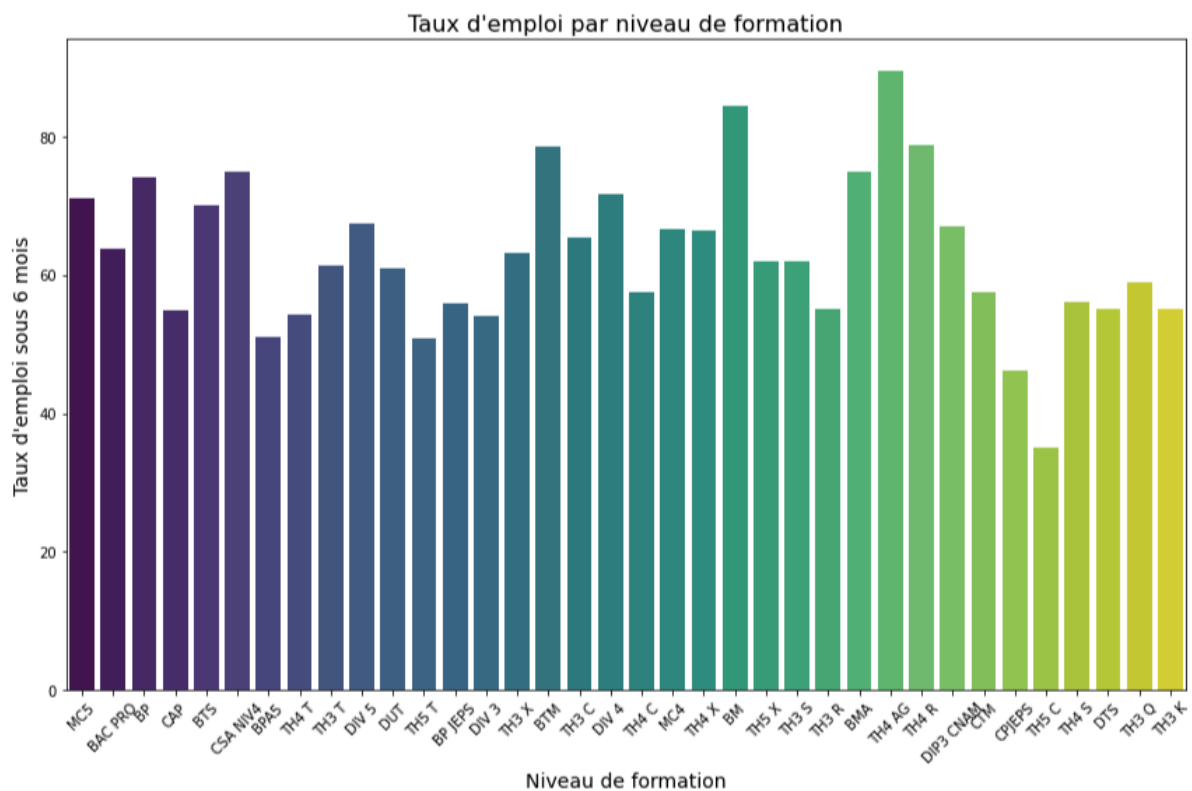
- Ce graphique permet de mettre en évidence la facilité qu'un alternant aura à trouver un emploi en fonction de sa formation.
- Ici nous voyons notamment que les formation de niveau 4 en agriculture marchent bien contrairement aux th5 en commerce


```
import seaborn as sns
import matplotlib.pyplot as plt

data = indicateurs_pk.toPandas()

data1_cleaned = data.dropna(subset=['Taux d\'emploi 6 mois après la sortie', 'Niveau de formation'])
data1_cleaned['Taux d\'emploi 6 mois après la sortie'] = data1_cleaned['Taux d\'emploi 6 mois après la sortie'].astype(float)

plt.figure(figsize=(12, 8))
sns.barplot(
    data=data1_cleaned,
    x='Niveau de formation',
    y='Taux d\'emploi 6 mois après la sortie',
    ci=None,
    palette="viridis"
)
plt.title("Taux d'emploi par niveau de formation", fontsize=16)
plt.xlabel('Niveau de formation', fontsize=14)
plt.ylabel("Taux d'emploi sous 6 mois", fontsize=14)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



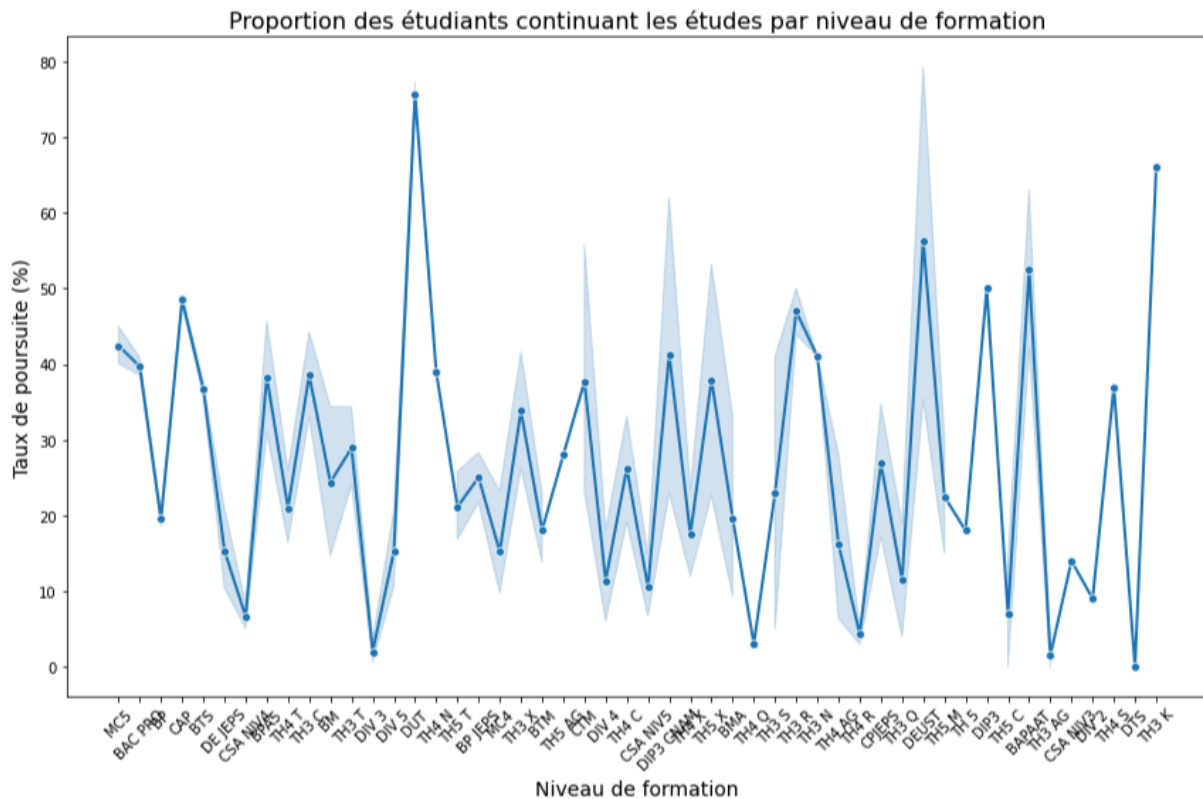
3.7 Proportion des étudiants continuant les études par niveau de formation

- En opposition au dernier, ce graphique met en évidence dans quels niveaux d'études les étudiants choisissent de le rester.

- Nous pouvons voir un gros écart entre chaque formation mais les DUT semblent être de loin les formations avec le plus de continuations.

```
study_data = data[['Niveau de formation', 'Taux de poursuite d\'études']].dropna()
study_data['Taux de poursuite d\'études'] = study_data['Taux de poursuite d\'études'].astype(float)

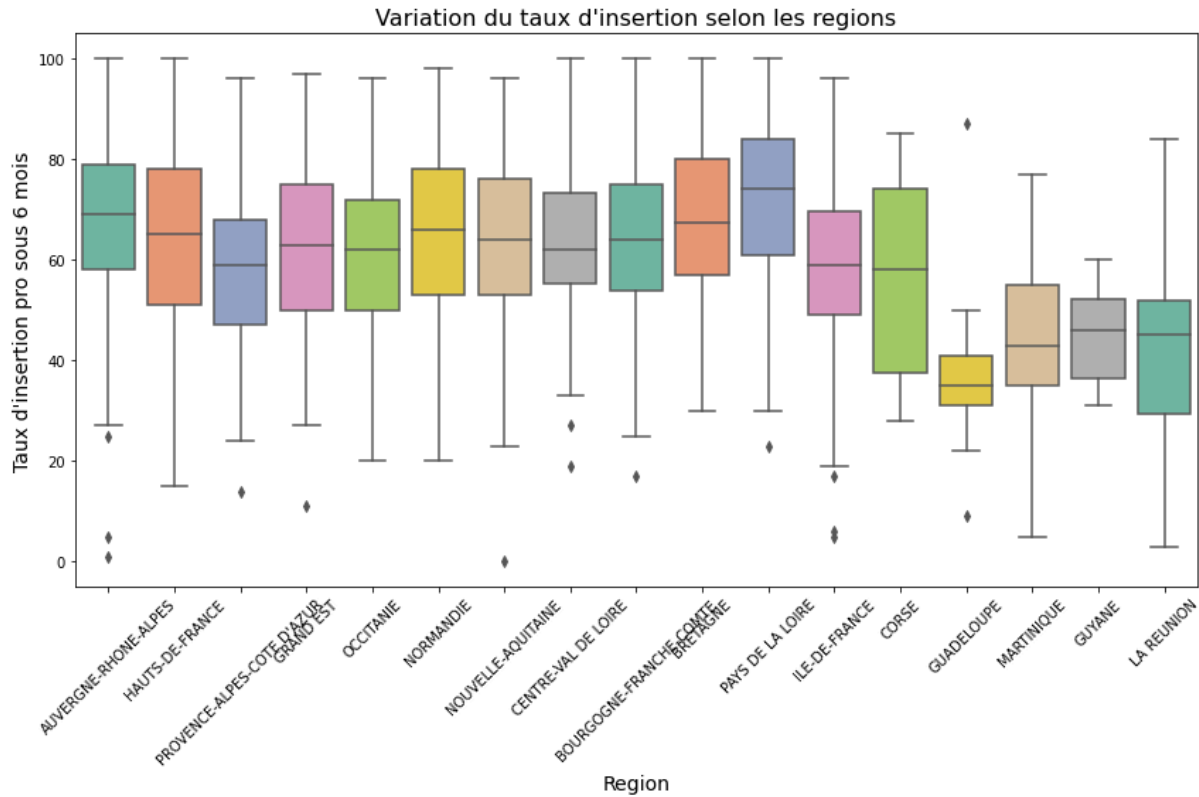
plt.figure(figsize=(12, 8))
sns.lineplot(
    data=study_data,
    x='Niveau de formation',
    y='Taux de poursuite d\'études',
    marker='o',
    linewidth=2,
    palette="viridis"
)
plt.title('Proportion des étudiants continuant les études par niveau de formation', fontsize=16)
plt.xlabel('Niveau de formation', fontsize=14)
plt.ylabel('Taux de poursuite (%)', fontsize=14)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



3.8 Variation du taux d'insertion selon les régions

- Dans ce graphique à quartiles nous pouvons voir les écarts de valeurs dans le taux d'insertion professionnelle par région.
- Ici nous pouvons voir qu'il peut y avoir de grandes différences dans les valeurs entre chaque formation mais on remarque surtout que les étudiants d'outre-mer sont ceux qui ont le plus de mal à trouver du travail et qu'en Ile de France bien qu'abritant la région parisienne n'a pas une médiane qui casse des records sûrement à cause de la demande qui ne suit pas le grand nombre de nouveaux diplômés chaque année.

```
plt.figure(figsize=(12, 8))
sns.boxplot(
    data=data.dropna(subset=['Taux d\'emploi 6 mois après la sortie', 'Standardized_Region']),
    x='Standardized_Region',
    y='Taux d\'emploi 6 mois après la sortie',
    palette="Set2"
)
plt.title("Variation du taux d'insertion selon les regions", fontsize=16)
plt.xlabel('Region', fontsize=14)
plt.ylabel("Taux d'insertion pro sous 6 mois", fontsize=14)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



3.9 Carte thermique du taux d'insertion

- Avec cette heatmap on peut très facilement voir quelles sont les parties de la France qui recrutent le plus d'alternants.
- Dans cette carte de la France métropolitaine nous remarquons directement que les endroits où il est le plus facile de trouver du travail restent les grandes villes comme Paris, Lyon et Marseille.

```
from folium.plugins import HeatMap

# Filtrer les coordonnées géographiques invalides
valid_data = merged_data[merged_data["Geo Coordinates"].str.contains(r'^~?\d+(\.\d+)?,~?\d+(\.\d+)?$', na=False)]

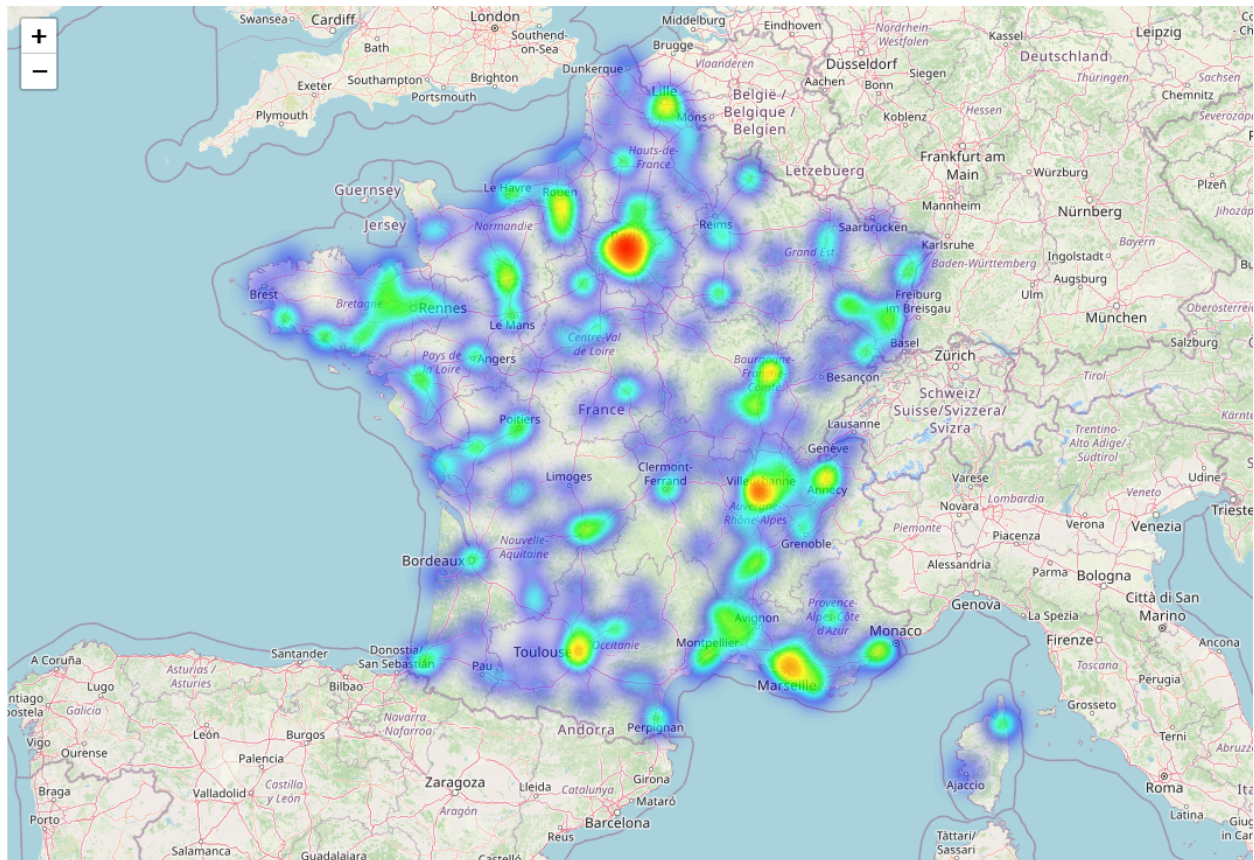
# Séparer les géocoordonnées
valid_data[['latitude', 'longitude']] = valid_data["Geo Coordinates"].str.split(',', expand=True).astype(float)

employment_heatmap = folium.Map(location=[46.603354, 1.888334], zoom_start=6)

heat_data_employment = valid_data.dropna(subset=["Taux d'emploi 6 mois après la sortie"])[
    ["latitude", "longitude", "Taux d'emploi 6 mois après la sortie"]
].values.tolist()

HeatMap(heat_data_employment, radius=15).add_to(employment_heatmap)

employment_heatmap.save("Employment_Rates_Heatmap.html")
display(employment_heatmap)
```



3.9 Carte thermique du taux de rupture

- Avec cette heatmap on peut très facilement voir quelles sont les parties de la France dont les étudiants ne finissent pas leurs contrats d'apprentissage.
- Comme la première, on voit que les plus affectées sont les grandes villes mais le reste de la France semble aussi en avoir beaucoup.


```
from folium.plugins import HeatMap
import branca.colormap as cm

valid_data = merged_data[merged_data["Geo Coordinates"].str.contains(r'^-?\d+(\.\d+)?,-?\d+(\.\d+)?$', na=False)]

valid_data[["latitude", "longitude"]] = valid_data["Geo Coordinates"].str.split(',', expand=True).astype(float)

dropout_heatmap = folium.Map(location=[46.603354, 1.888334], zoom_start=6)

heat_data_dropout = valid_data.dropna(subset=["Part de contrats d'apprentissage interrompus avant leur terme"])[
    ["latitude", "longitude", "Part de contrats d'apprentissage interrompus avant leur terme"]
].values.tolist()

HeatMap(heat_data_dropout, radius=15).add_to(dropout_heatmap)

min_dropout = valid_data["Part de contrats d'apprentissage interrompus avant leur terme"].min()
max_dropout = valid_data["Part de contrats d'apprentissage interrompus avant leur terme"].max()
colormap = cm.LinearColormap(colors=['green', 'yellow', 'red'], vmin=min_dropout, vmax=max_dropout)
colormap.caption = "Dropout Rates (%)"

colormap.add_to(dropout_heatmap)

dropout_heatmap.save("Dropout_Rates_Heatmap_With_Scale.html")
display(dropout_heatmap)
```

