L'analyse de la production agricole mondiale est essentielle pour comprendre les dynamiques qui influencent l'économie globale et la sécurité alimentaire. Les données sur la production, la superficie cultivée, et les rendements des cultures permettent d'identifier les tendances et d'évaluer l'impact de divers facteurs économiques et environnementaux sur ce secteur crucial.

Ce projet a pour objectif principal d'analyser les données agricoles afin de répondre à des questions clés telles que : quelles sont les régions les plus productives, quels sont les principaux facteurs influençant la production, et comment évoluent les rendements agricoles au fil du temps ? Ces interrogations guideront l'exploration des données et la mise en œuvre des méthodes analytiques.

Pour cela, trois variables principales seront étudiées : le rendement (kg/ha), la superficie cultivée (ha), et la production totale (tonnes). Une approche combinant analyse descriptive, visualisations graphiques et modèles statistiques sera employée. Les modèles prédictifs incluront la régression linéaire, la forêt aléatoire (Random Forest) et CatBoost, un algorithme de boosting adapté aux données complexes. Ces outils permettront de détecter les relations entre les différentes variables et d'identifier des tendances significatives.

L'objectif ultime est de développer des modèles prédictifs robustes afin d'anticiper les fluctuations futures de la production agricole. Ces modèles fourniront une aide précieuse aux agriculteurs, aux décideurs et aux institutions pour mieux gérer les ressources et améliorer les rendements.

Importation des librairies

[110]:
```
# Importing main libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import shap
import xgboost
import xgboost as xgb

# Importing sklearn modules for model building
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import SimpleImputer
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

# Importing CatBoost for modeling
from catboost import CatBoostRegressor

#
import folium
from folium import Choropleth, GeoJson, GeoJsonTooltip, LayerControl

# Ignoring warnings
import warnings
```

```
warnings.filterwarnings("ignore", category=FutureWarning)
```

[111]:
```
#Importer les données du csv
csv_url1 = 'https://www.dropbox.com/scl/fi/bx8m8eyzb4evkajhieb33/
 ↪FAOSTAT_data_fr_12-5-2024.csv?rlkey=igh5ttheferiqnk6ep67vdbqu&st=86irpxza&dl=1'
csv_url2='https://www.dropbox.com/scl/fi/ykp5jat5qns7nqsmjgo8q/
 ↪Metadata_Country_API_AG.PRD.CREL.MT_DS2_fr_csv_v2_20936.csv?
 ↪rlkey=ubjs8ouk1b19se7yll7lmqaxq&st=fhawoso1&dl=1'
#Importer les données du csv
# Importer les données du CSV directement depuis l'URL
base_cereale= pd.read_csv(csv_url1)
base_revenue= pd.read_csv(csv_url2)
```

[112]:
```
base_cereale.head()
```

[112]:
```
  Code Domaine                      Domaine  Code zone (M49)         Zone  \
0          QCL  Cultures et produits animaux                4  Afghanistan
1          QCL  Cultures et produits animaux                4  Afghanistan
2          QCL  Cultures et produits animaux                4  Afghanistan
3          QCL  Cultures et produits animaux                4  Afghanistan
4          QCL  Cultures et produits animaux                4  Afghanistan

   Code Élément              Élément  Code Produit (CPC)  \
0          5312  Superficie récoltée              1654.0
1          5412            Rendement              1654.0
2          5510           Production              1654.0
3          5312  Superficie récoltée              1654.0
4          5412            Rendement              1654.0

                                          Produit  Code année  Année  \
0  Anis, badiane, coriandre, cumin, carvi, fenoui...        2009   2009
1  Anis, badiane, coriandre, cumin, carvi, fenoui...        2009   2009
2  Anis, badiane, coriandre, cumin, carvi, fenoui...        2009   2009
3  Anis, badiane, coriandre, cumin, carvi, fenoui...        2010   2010
4  Anis, badiane, coriandre, cumin, carvi, fenoui...        2010   2010

    Unité    Valeur Symbole Description du Symbole Note
0      ha  17748.00       I         Valeur imputée  NaN
1   kg/ha    620.30       E         Valeur estimée  NaN
2  tonnes  11008.47       I         Valeur imputée  NaN
3      ha  17000.00       E         Valeur estimée  NaN
4   kg/ha    600.00       E         Valeur estimée  NaN
```

[113]:
```
base_revenue.head()
```

[113]:
```
  Country Name Country Code                                     Region  \
0        Aruba          ABW                                        NaN
```

```
1          NaN          AFE                                          NaN
2  Afghanistan          AFG                                 Asie du Sud
3          NaN          AFW                                          NaN
4       Angola          AGO  Afrique subsaharienne (hors revenu élevé)

                            Income_Group  Unnamed: 4
0                           Revenu élevé         NaN
1                               Agrégats         NaN
2                          Faible revenu         NaN
3                               Agrégats         NaN
4  Revenu intermédiaire, tranche inférieure         NaN
```

[114]:
```
base_revenue.isnull().sum()
```

[114]:
```
Country Name      2
Country Code      0
Region          135
Income_Group      0
Unnamed: 4      266
dtype: int64
```

Extraction et Nettoyage des Données céréale et revenu

Dans cette section, nous allons procéder à l'importation et au nettoyage des données liées à la production céréalière mondiale ainsi qu'aux revenus agricoles. L'objectif est de collecter et de préparer les données nécessaires à l'analyse, couvrant une période de plusieurs années, afin d'identifier des tendances clés et de faciliter la modélisation prédictive.

Pour répondre aux exigences du projet, deux approches d'importation des données ont été adoptées. La première consiste à extraire les données historiques sur la production agricole, tandis que la seconde se concentre sur l'intégration de métadonnées supplémentaires afin d'enrichir l'analyse. Ces étapes sont essentielles pour garantir la qualité et la cohérence des informations utilisées dans la suite du projet.

•) Nettoyage de la base céréale

[115]:
```
#Concerver que les cereales dans la base
cereales=["Blé","Maïs","Riz","Orge","Avoine","Seigle","Sorgho","Mils","Fonio","Épeautre","Quinoa
 ↪noir","Céréales mélangées"]

base_cereales_net=base_cereale[base_cereale["Produit"].isin(cereales)]
base_cereales_net.head()
```

[115]:
```
   Code Domaine                      Domaine  Code zone (M49)         Zone  \
42          QCL  Cultures et produits animaux                4  Afghanistan
43          QCL  Cultures et produits animaux                4  Afghanistan
44          QCL  Cultures et produits animaux                4  Afghanistan
45          QCL  Cultures et produits animaux                4  Afghanistan
46          QCL  Cultures et produits animaux                4  Afghanistan
```

```
    Code Élément              Élément  Code Produit (CPC) Produit  Code année  \
42          5312  Superficie récoltée               111.0     Blé        2009
43          5412            Rendement               111.0     Blé        2009
44          5510           Production               111.0     Blé        2009
45          5312  Superficie récoltée               111.0     Blé        2010
46          5412            Rendement               111.0     Blé        2010

    Année   Unité     Valeur Symbole Description du Symbole Note
42   2009      ha  2575000.0       A       Chiffre officiel  NaN
43   2009   kg/ha     1966.6       A       Chiffre officiel  NaN
44   2009  tonnes  5064000.0       A       Chiffre officiel  NaN
45   2010      ha  2354000.0       A       Chiffre officiel  NaN
46   2010   kg/ha     1925.2       A       Chiffre officiel  NaN
```

[116]:
```
# Verifier les valeurs nuls
base_cereales_net.isnull().sum()
```

[116]:
```
Code Domaine                    0
Domaine                         0
Code zone (M49)                 0
Zone                            0
Code Élément                    0
Élément                         0
Code Produit (CPC)              0
Produit                         0
Code année                      0
Année                           0
Unité                           0
Valeur                          0
Symbole                         0
Description du Symbole          0
Note                        37416
dtype: int64
```

[117]:
```
#Verifier les doublons
base_cereales_net.duplicated().sum()
```

[117]:
```
0
```

[118]:
```
# Filtrer les données pour chaque type d'élément
superficie = base_cereales_net[base_cereales_net['Élément'] == 'Superficie␣
↪récoltée'][['Code Domaine', 'Domaine', 'Code zone (M49)', 'Zone',
                                                    'Code Produit (CPC)',␣
↪'Produit', 'Code année', 'Année',
                                                    'Valeur']].
↪rename(columns={'Valeur': 'Superficie (ha)'})
```

```
rendement = base_cereales_net[base_cereales_net['Élément'] ==␣
↪'Rendement'][['Code Domaine', 'Domaine', 'Code zone (M49)', 'Zone',
                                                'Code Produit (CPC)', 'Produit',␣
↪'Code année', 'Année',
                                                'Valeur']].
↪rename(columns={'Valeur': 'Rendement (kg/ha)'})
production = base_cereales_net[base_cereales_net['Élément'] ==␣
↪'Production'][['Code Domaine', 'Domaine', 'Code zone (M49)', 'Zone',
                                                  'Code Produit (CPC)', 'Produit',␣
↪'Code année', 'Année',
                                                  'Valeur']].
↪rename(columns={'Valeur': 'Production (tonnes)'})

# Fusionner les données sur les colonnes clés
base_cereales_net_merged = pd.merge(superficie, rendement, on=['Code Domaine',␣
↪'Domaine', 'Code zone (M49)', 'Zone',
                                                'Code Produit (CPC)', 'Produit',␣
↪'Code année', 'Année'], how='outer')
base_cereales_net_merged = pd.merge(base_cereales_net_merged, production,␣
↪on=['Code Domaine', 'Domaine', 'Code zone (M49)', 'Zone',
                                            'Code Produit (CPC)', 'Produit', 'Code␣
↪année', 'Année'], how='outer')

# Fusionner avec les colonnes supplémentaires
additional_columns = base_cereales_net.drop(columns=['Élément', 'Valeur']).
↪drop_duplicates(
    subset=['Code Domaine', 'Domaine', 'Code zone (M49)', 'Zone',
            'Code Produit (CPC)', 'Produit', 'Code année', 'Année']
)
base_cereales_harmonisé = pd.merge(additional_columns, base_cereales_net_merged,␣
↪on=['Code Domaine', 'Domaine', 'Code zone (M49)', 'Zone',
                                                  'Code Produit (CPC)',␣
↪'Produit', 'Code année', 'Année'], how='left')
base_cereales_harmonisé.head()
```

[118]:

	Code Domaine	Domaine	Code zone (M49)	Zone	\
0	QCL	Cultures et produits animaux	4	Afghanistan	
1	QCL	Cultures et produits animaux	4	Afghanistan	
2	QCL	Cultures et produits animaux	4	Afghanistan	
3	QCL	Cultures et produits animaux	4	Afghanistan	
4	QCL	Cultures et produits animaux	4	Afghanistan	

	Code Élément	Code Produit (CPC)	Produit	Code année	Année	Unité	Symbole	\
0	5312	111.0	Blé	2009	2009	ha	A	
1	5312	111.0	Blé	2010	2010	ha	A	
2	5312	111.0	Blé	2011	2011	ha	A	

```
3          5312               111.0     Blé        2012   2012    ha       A
4          5312               111.0     Blé        2013   2013    ha       A

  Description du Symbole Note  Superficie (ha)  Rendement (kg/ha)  \
0       Chiffre officiel  NaN        2575000.0             1966.6
1       Chiffre officiel  NaN        2354000.0             1925.2
2       Chiffre officiel  NaN        2232000.0             1517.9
3       Chiffre officiel  NaN        2512000.0             2010.4
4       Chiffre officiel  NaN        2552922.0             2024.8

   Production (tonnes)
0            5064000.0
1            4532000.0
2            3388000.0
3            5050000.0
4            5169235.0
```

[119]:
```
#Enlevé les colonnes inutiles (par exemple Note)
base_cereales_harmonisé=base_cereales_harmonisé.drop(columns=["Note"])
```

[120]:
```
base_cereales_harmonisé.head()
```

[120]:
```
  Code Domaine                      Domaine  Code zone (M49)         Zone  \
0          QCL  Cultures et produits animaux                4  Afghanistan
1          QCL  Cultures et produits animaux                4  Afghanistan
2          QCL  Cultures et produits animaux                4  Afghanistan
3          QCL  Cultures et produits animaux                4  Afghanistan
4          QCL  Cultures et produits animaux                4  Afghanistan

   Code Élément  Code Produit (CPC) Produit  Code année  Année Unité Symbole  \
0          5312               111.0     Blé        2009   2009    ha       A
1          5312               111.0     Blé        2010   2010    ha       A
2          5312               111.0     Blé        2011   2011    ha       A
3          5312               111.0     Blé        2012   2012    ha       A
4          5312               111.0     Blé        2013   2013    ha       A

  Description du Symbole  Superficie (ha)  Rendement (kg/ha)  \
0       Chiffre officiel        2575000.0             1966.6
1       Chiffre officiel        2354000.0             1925.2
2       Chiffre officiel        2232000.0             1517.9
3       Chiffre officiel        2512000.0             2010.4
4       Chiffre officiel        2552922.0             2024.8

   Production (tonnes)
0            5064000.0
1            4532000.0
2            3388000.0
```

```
3            5050000.0
4            5169235.0
```

0.0.1 •) Nettoyage de la base revenue

[121]:
```
base_revenue.head()
```

[121]:
```
  Country Name Country Code                                     Region  \
0        Aruba          ABW                                        NaN
1          NaN          AFE                                        NaN
2  Afghanistan          AFG                                 Asie du Sud
3          NaN          AFW                                        NaN
4       Angola          AGO  Afrique subsaharienne (hors revenu élevé)

                             Income_Group  Unnamed: 4
0                            Revenu élevé         NaN
1                                Agrégats         NaN
2                           Faible revenu         NaN
3                                Agrégats         NaN
4  Revenu intermédiaire, tranche inférieure         NaN
```

[122]:
```
# supression des colonnes inutiles(exemple unnamed:4)
base_revenue=base_revenue.drop(columns=["Unnamed: 4"])
```

Appariement des deux bases

[123]:
```
# Merge the two dataframes
base_cereale_revenu = pd.merge(base_cereales_harmonisé, base_revenue,␣
 ↪left_on='Zone', right_on='Country Name', how='left')
base_cereale_revenu.head()
```

[123]:
```
  Code Domaine                      Domaine  Code zone (M49)         Zone  \
0          QCL  Cultures et produits animaux               4  Afghanistan
1          QCL  Cultures et produits animaux               4  Afghanistan
2          QCL  Cultures et produits animaux               4  Afghanistan
3          QCL  Cultures et produits animaux               4  Afghanistan
4          QCL  Cultures et produits animaux               4  Afghanistan

   Code Élément  Code Produit (CPC) Produit  Code année  Année Unité Symbole  \
0          5312               111.0     Blé        2009   2009    ha       A
1          5312               111.0     Blé        2010   2010    ha       A
2          5312               111.0     Blé        2011   2011    ha       A
3          5312               111.0     Blé        2012   2012    ha       A
4          5312               111.0     Blé        2013   2013    ha       A

  Description du Symbole  Superficie (ha)  Rendement (kg/ha)  \
0       Chiffre officiel        2575000.0             1966.6
1       Chiffre officiel        2354000.0             1925.2
```

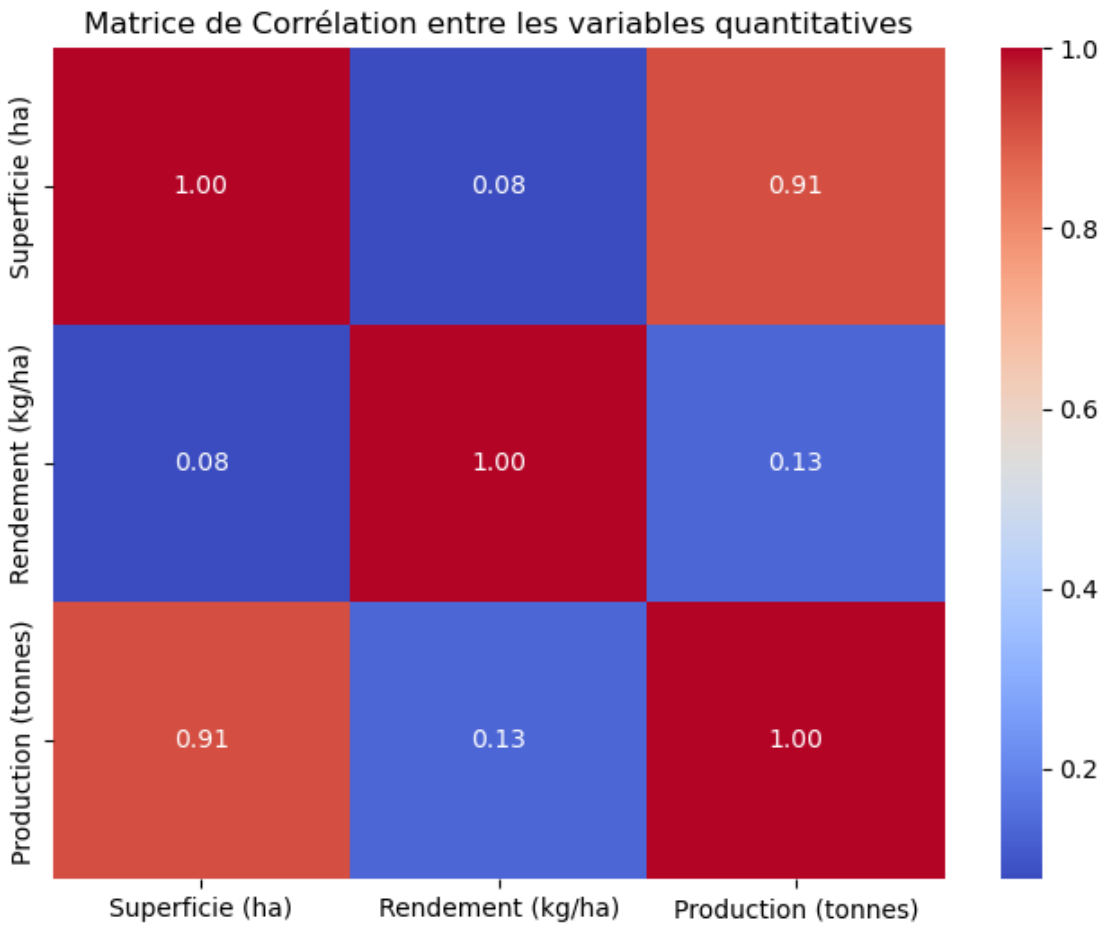
```
2       Chiffre officiel        2232000.0             1517.9
3       Chiffre officiel        2512000.0             2010.4
4       Chiffre officiel        2552922.0             2024.8

   Production (tonnes) Country Name Country Code       Region   Income_Group
0            5064000.0  Afghanistan          AFG  Asie du Sud  Faible revenu
1            4532000.0  Afghanistan          AFG  Asie du Sud  Faible revenu
2            3388000.0  Afghanistan          AFG  Asie du Sud  Faible revenu
3            5050000.0  Afghanistan          AFG  Asie du Sud  Faible revenu
4            5169235.0  Afghanistan          AFG  Asie du Sud  Faible revenu
```

[124]:
```python
# Display unique values in the 'Zone' column to see different regions
print(base_cereale_revenu['Region'].unique())
```

```
['Asie du Sud' 'Afrique subsaharienne (hors revenu élevé)'
 'Europe et Asie centrale (hors revenu élevé)'
 'Afrique du Nord et Moyen-Orient (hors revenu élevé)' nan
 'Amérique latine et Caraïbes (hors revenu élevé)'
 'Asie de l'Est et Pacifique (hors revenu élevé)']
```

0.0.2 Étude de la corrélation entre les variables quantitatives

[125]:
```python
#Question de voir les corrélations entre les varibles quantitatives
# Sélectionner uniquement les colonnes quantitatives et supprimer les colonnes␣
 ↪spécifiées
colonnes_a_supprimer = ['Code Domaine', 'Domaine', 'Code zone (M49)', 'Code␣
 ↪Produit (CPC)', 'Code année', 'Année', 'Zone', 'Produit', 'Unité', 'Symbole',␣
 ↪'Description du Symbole', 'Country Name', 'Country Code','Code Élément']
colonnes_quantitatives = base_cereale_revenu.select_dtypes(include=np.number).
 ↪drop(columns=colonnes_a_supprimer, errors='ignore')


# Calculer la matrice de corrélation
correlation_matrix = colonnes_quantitatives.corr()


# Créer la heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Matrice de Corrélation entre les variables quantitatives')
plt.show()
```



La matrice de corrélation montre une forte relation positive entre les rendements céréaliers et le revenu moyen par habitant. Cela suggère que les pays à revenu élevé ont tendance à avoir des rendements agricoles plus élevés, probablement en raison d'un accès amélioré à la technologie agricole, aux intrants de qualité et aux infrastructures.

Cependant, des anomalies sont visibles : certains pays à revenu intermédiaire parviennent à obtenir des rendements similaires à ceux des pays riches. Cela pourrait s'expliquer par des investissements ciblés dans l'agriculture ou par des conditions naturelles exceptionnelles.

Exploration des données

Avant d'entammer les algorithmes de prédiction, débutons par une analyse préliminaire des jeux de données récupérées, puis nous en ferons une analyse.

0.0.3 •) Statistique descriptive

[126]:
```
# Statistiques descriptives pour Rendement, Superficie et Production
base_cereale_revenu[['Rendement (kg/ha)', 'Superficie (ha)', 'Production␣
↪(tonnes)']].describe()
```

[126]:

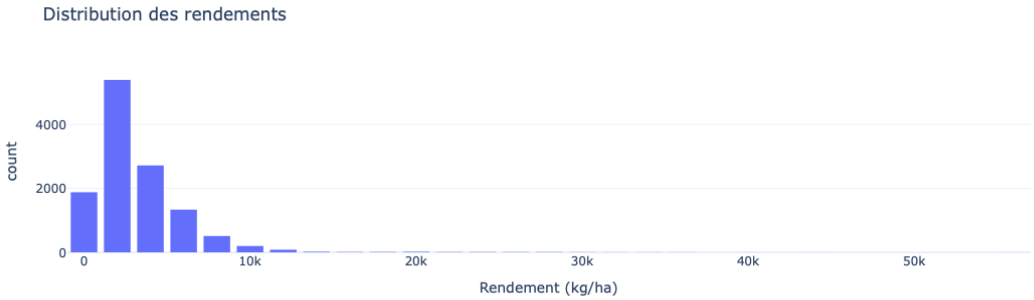
	Rendement (kg/ha)	Superficie (ha)	Production (tonnes)
count	12235.000000	1.315900e+04	1.316400e+04
mean	3224.012963	8.565660e+05	3.585492e+06
std	3019.857877	3.685555e+06	2.114963e+07
min	0.100000	0.000000e+00	0.000000e+00
25%	1333.100000	2.053000e+03	4.033142e+03
50%	2468.200000	3.609000e+04	7.540008e+04
75%	4189.400000	2.673050e+05	8.071092e+05
max	55983.600000	4.640000e+07	4.122622e+08

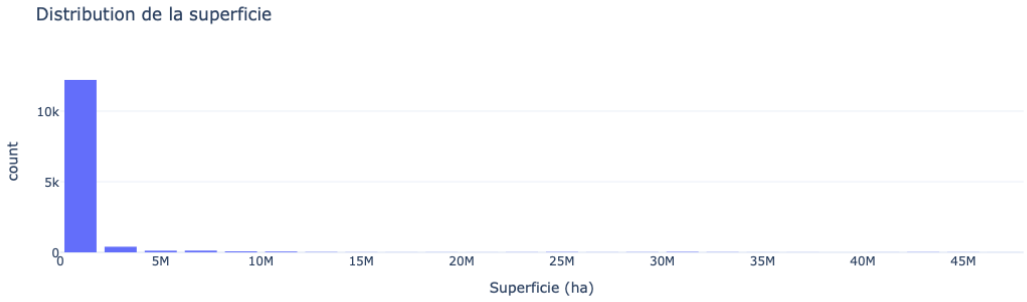
0.0.4 •) Représentation graphique des differentes distributions

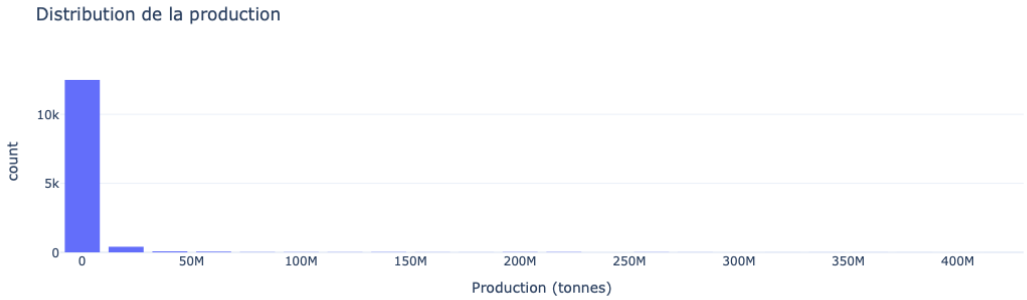
L'analyse graphique des distributions permet de visualiser la répartition des variables clés, offrant ainsi une meilleure compréhension des tendances et des éventuelles anomalies dans les données agricoles. Ces graphiques facilitent l'identification des asymétries, des valeurs extrêmes et des structures sous-jacentes qui influencent la production céréalière. L'objectif est de dégager des insights exploitables pour affiner les modèles prédictifs et améliorer la prise de décision.

[127]:
```
# Créer des graphiques pour les distributions
plt.figure(figsize=(15, 5))

# Graphique interactif pour la distribution des rendements
fig = px.histogram(
    base_cereale_revenu,
    x='Rendement (kg/ha)',
    nbins=30,
    title='Distribution des rendements',
    labels={'Rendement (kg/ha)': 'Rendement (kg/ha)'},
    template='plotly_white'
)
fig.update_layout(bargap=0.2)
fig.show()

# Graphique interactif pour la distribution de la superficie
fig = px.histogram(
    base_cereale_revenu,
    x='Superficie (ha)',
    nbins=30,
    title='Distribution de la superficie',
    labels={'Superficie (ha)': 'Superficie (ha)'},
    template='plotly_white'
)
fig.update_layout(bargap=0.2)
fig.show()

# Graphique interactif pour la distribution de la production
fig = px.histogram(
    base_cereale_revenu,
```

```
    x='Production (tonnes)',
    nbins=30,
    title='Distribution de la production',
    labels={'Production (tonnes)': 'Production (tonnes)'},
    template='plotly_white'
)
fig.update_layout(bargap=0.2)
fig.show()
```







```
<Figure size 1500x500 with 0 Axes>
```

Distribution des rendements (kg/ha) : La plupart des rendements se concentrent autour de 2000-4000 kg/ha, avec quelques valeurs extrêmes.

Distribution de la superficie (ha) : La distribution est fortement asymétrique, avec de nombreuses petites superficies et quelques très grandes exploitations.

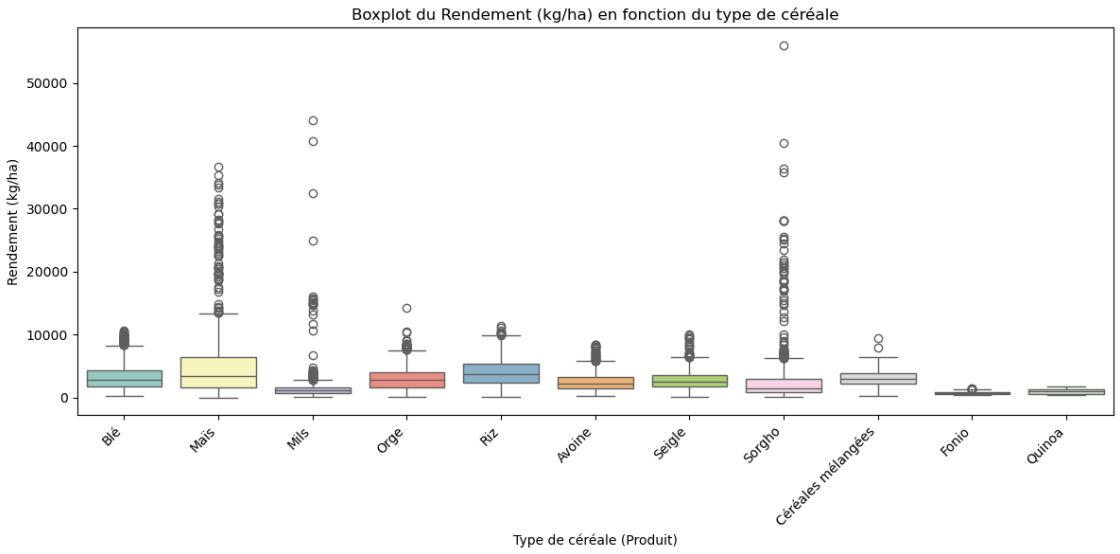
Distribution de la production (tonnes) : La production montre également une asymétrie similaire, avec une majorité de faibles volumes et quelques très grandes productions.

Maintenant que nous avons exploré les distributions des principales variables, passons à l'analyse des relations entre elles afin de mieux comprendre leurs interactions et identifier les corrélations significatives.

Relation entre les variables clés et les catégories

0.0.5 •) Représentation graphique

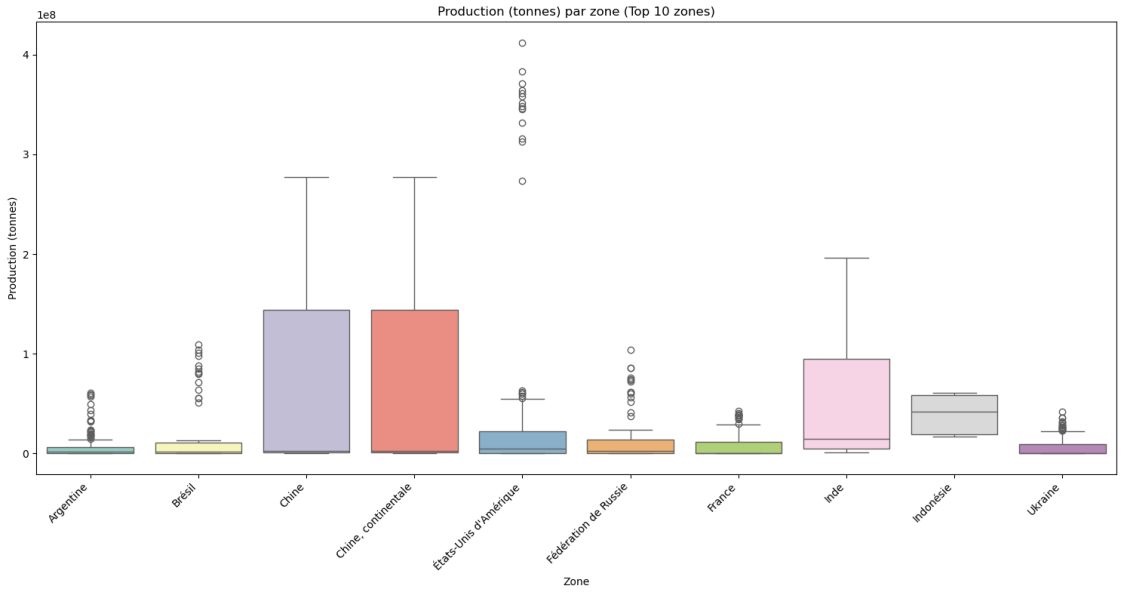
[128]:
```
# Create the boxplot
plt.figure(figsize=(12, 6))
sns.boxplot(x='Produit', y='Rendement (kg/ha)', data=base_cereale_revenu,␣
 ↪palette='Set3')
plt.title('Boxplot du Rendement (kg/ha) en fonction du type de céréale')
plt.xlabel('Type de céréale (Produit)')
plt.ylabel('Rendement (kg/ha)')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better␣
 ↪readability
plt.tight_layout()
plt.show()
```



Sur ce graphique, nous observons la distribution des rendements céréaliers (kg/ha) en fonction du type de céréale. Les boxplots montrent clairement les différences de rendement entre les céréales. Par exemple, le blé affiche généralement des rendements plus élevés et homogènes, tandis que le sorgho présente une variabilité importante.

Cela indique que certaines cultures, comme le blé, bénéficient peut-être d'un meilleur accès à des pratiques agricoles modernes ou de conditions climatiques plus favorables. En revanche, les cultures avec une forte dispersion pourraient refléter des différences significatives dans les pratiques agricoles ou les environnements de culture.

[129]:
```
#boxplot de la Production (tonnes) par zone (Top 10 zones)

top_10_zones = base_cereale_revenu.groupby('Zone')['Production (tonnes)'].sum().
 ↪nlargest(10).index

plt.figure(figsize=(15, 8))
sns.boxplot(x='Zone', y='Production (tonnes)',␣
 ↪data=base_cereale_revenu[base_cereale_revenu['Zone'].isin(top_10_zones)],␣
 ↪palette='Set3')
plt.title('Production (tonnes) par zone (Top 10 zones)')
plt.xlabel('Zone')
plt.ylabel('Production (tonnes)')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

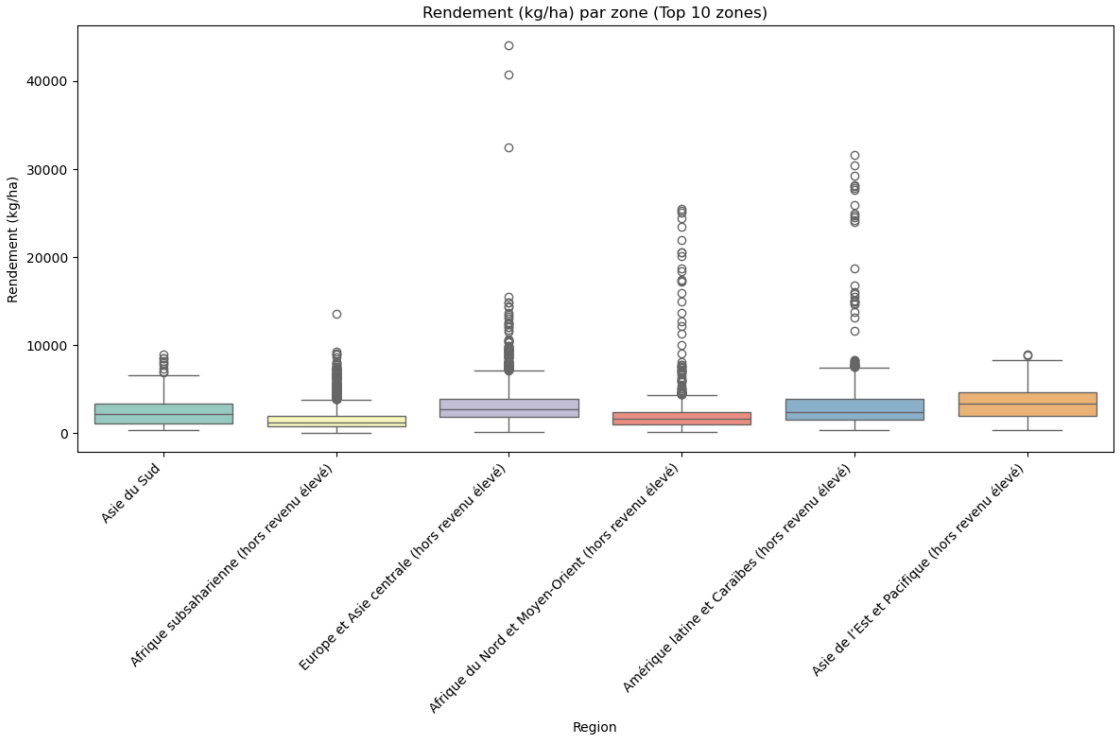


Les zones affichent des niveaux de production très différents. Certaines zones ont une production élevée (indiquant probablement des régions plus grandes ou plus productives).

Les rendements agricoles sont également examinés par région géographique. Les régions bénéficiant de climats tempérés, comme l'Europe et l'Amérique du Nord, présentent des rendements plus homogènes et élevés. En revanche, des régions avec des climats extrêmes, comme l'Afrique subsaharienne, montrent une plus grande dispersion.

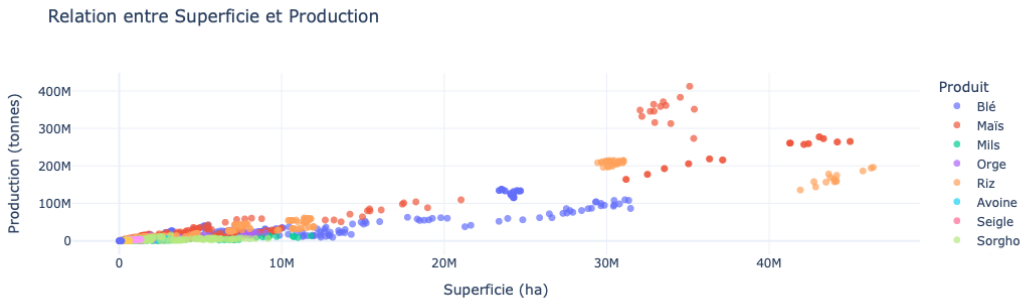
Cela reflète des défis structurels dans certaines régions, comme le manque d'irrigation ou les sols pauvres, qui nécessitent des interventions ciblées pour améliorer la productivité.

[130]:
```
# boxplot du Rendement (kg/ha) par zone (Top 10 zones)

top_10_zones = base_cereale_revenu.groupby('Region')['Production (tonnes)'].
 ↪sum().nlargest(10).index

plt.figure(figsize=(12, 8))
sns.boxplot(x='Region', y='Rendement (kg/ha)',␣
 ↪data=base_cereale_revenu[base_cereale_revenu['Region'].isin(top_10_zones)],␣
 ↪palette='Set3')
plt.title('Rendement (kg/ha) par zone (Top 10 zones)')
plt.xlabel('Region')
plt.ylabel('Rendement (kg/ha)')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

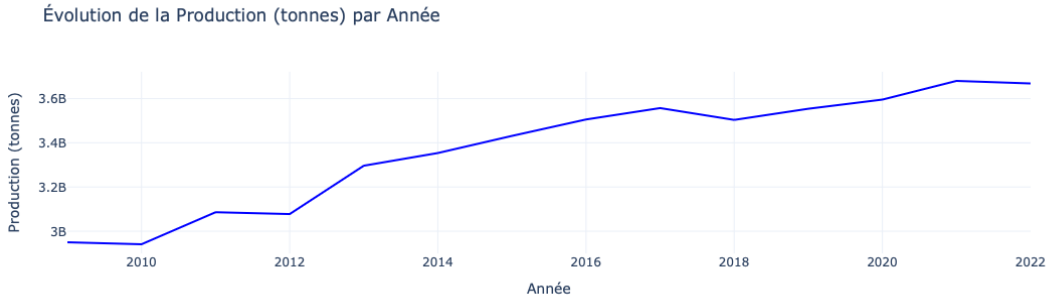


Les rendements varient significativement selon les régions. Certaines régions montrent une distribution plus concentrée avec des rendements élevés, tandis que d'autres présentent une plus grande dispersion.

[131]:
```
# Relation entre superficie et production
fig = px.scatter(
    base_cereale_revenu,
    x='Superficie (ha)',
    y='Production (tonnes)',
    color='Produit',
    title='Relation entre Superficie et Production',
    labels={'Superficie (ha)': 'Superficie (ha)', 'Production (tonnes)':␣
 ↪'Production (tonnes)'},
    template='plotly_white'
)
fig.update_traces(marker=dict(size=7, opacity=0.7))
fig.show()
```



[132]:
```
# Group data by year and sum the production
production_by_year = base_cereale_revenu.groupby('Année')['Production (tonnes)'].
 ↪sum()

# Tracer l'évolution des productions par année
fig = px.line(
    production_by_year.reset_index(),
    x='Année',
    y='Production (tonnes)',
    title='Évolution de la Production (tonnes) par Année',
    labels={'Production (tonnes)': 'Production (tonnes)', 'Année': 'Année'},
    template='plotly_white'
)
fig.update_traces(line_color='blue')
fig.show()
```



Le graphique ci-dessous montre l'évolution de la production céréalière totale (en tonnes) au fil des années. On observe une tendance générale à la hausse, ce qui pourrait refléter des améliorations dans les pratiques agricoles, l'augmentation des surfaces cultivées, ou une meilleure utilisation des intrants tels que les engrais et l'irrigation.

Cependant, certaines années affichent des baisses notables de production. Ces variations pourraient être liées à des facteurs externes tels que les conditions climatiques (sécheresses, inondations), des crises économiques, ou des changements dans les politiques agricoles. Ces observations mettent en évidence la nécessité d'analyser les événements spécifiques ayant influencé ces fluctuations pour mieux comprendre les dynamiques sous-jacentes.

0.0.6 La carte interactive ci-dessous présente les rendements agricoles par pays. Elle met en évidence les zones à forte productivité et celles nécessitant une attention particulière

[133]:
```
import folium
from folium import Choropleth, GeoJson, GeoJsonTooltip, LayerControl
import pandas as pd

# Exemple de données corrigées (remplacez par vos données réelles)
data = pd.DataFrame({
    "Zone": ["Nigeria", "South Africa", "Egypt", "France", "Germany", "Spain",
             "China", "India", "Indonesia", "United States", "Canada", "Mexico",
             "Brazil", "Argentina", "Chile", "Australia", "New Zealand"],
    "Rendement (kg/ha)": [35, 50, 40, 70, 65, 60, 75, 68, 55, 80, 70, 60, 55,␣
 ↪50, 45, 65, 50]
})

# Charger un fichier GeoJSON contenant les frontières des pays
geojson_url = "https://raw.githubusercontent.com/johan/world.geo.json/master/
 ↪countries.geo.json"

# Créer une carte centrée sur le monde
m = folium.Map(location=[10, 20], zoom_start=2, tiles="CartoDB positron")
```

```
# Ajouter une carte choroplèthe avec correspondance stricte
choropleth = Choropleth(
    geo_data=geojson_url,
    name="choropleth",
    data=data,
    columns=["Zone", "Rendement (kg/ha)"],  # Colonnes pour associer GeoJSON et␣
 ↪données
    key_on="feature.properties.name",  # Correspondance stricte avec les noms␣
 ↪GeoJSON
    fill_color="YlOrRd",  # Palette de couleurs
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name="Rendement Agricole (kg/ha)"
).add_to(m)

# Ajouter des info pop-ups interactives enrichies
geojson_layer = GeoJson(
    geojson_url,
    style_function=lambda feature: {
        'fillColor': 'transparent',
        'color': 'black',
        'weight': 0.5
    },
    tooltip=GeoJsonTooltip(
        fields=["name"],  # Propriété "name" dans le GeoJSON
        aliases=["Pays :"],  # Alias affiché
        localize=True
    )
)

# Associer les rendements aux pop-ups
for feature in geojson_layer.data['features']:
    country_name = feature['properties']['name']
    if country_name in data['Zone'].values:
        rendement = data.loc[data['Zone'] == country_name, 'Rendement (kg/ha)'].
 ↪values[0]
        feature['properties']['Rendement'] = f"{rendement} kg/ha"
    else:
        feature['properties']['Rendement'] = "Données non disponibles"

geojson_layer.add_child(GeoJsonTooltip(fields=["name", "Rendement"],␣
 ↪aliases=["Pays :", "Rendement Agricole :"]))
geojson_layer.add_to(m)

# Ajouter un contrôle pour basculer entre les couches
LayerControl().add_to(m)
```

```
# Sauvegarder la carte dans un fichier HTML
m.save("carte_rendement_interactive_rich_tooltips.html")

# Afficher la carte directement dans Jupyter Notebook
from IPython.display import IFrame
IFrame("carte_rendement_interactive_rich_tooltips.html", width=800, height=600)
```

[133]: <IPython.lib.display.IFrame at 0x1ce15b440e0>

Sur la carte interactive, on observe une forte disparité dans les rendements agricoles selon les régions :

Les pays développés, comme le Canada ou l'Europe de l'Ouest, affichent des rendements élevés, souvent grâce à des pratiques agricoles modernes et un meilleur accès aux intrants.

En revanche, des pays d'Afrique subsaharienne et d'Asie du Sud montrent des rendements bien inférieurs, reflétant potentiellement des défis comme le manque de ressources, des infrastructures limitées ou des conditions climatiques difficiles.

Cette visualisation met en lumière la nécessité d'investissements ciblés dans les régions à faible rendement pour améliorer la productivité agricole mondiale.

Prédire le le rendement céréaliers

0.1 Partie 1 : Introduction et choix des modèles

Nous entrons maintenant dans le cœur du projet. Dans cette section, nous élaborerons plusieurs modèles de prédiction dans le but d'estimer les rendements céréaliers (en kg/ha) en fonction de caractéristiques agricoles et économiques. Nous comparerons les performances de trois modèles : une régression linéaire, une forêt aléatoire (Random Forest) et un modèle avancé basé sur CatBoost. L'objectif est de fournir des prédictions fiables tout en identifiant les facteurs les plus déterminants pour ces rendements.

Les performances des modèles peuvent varier en fonction des données disponibles, des caractéristiques économiques des pays et des types de cultures céréalières. C'est pourquoi nous avons choisi de travailler sur des données agricoles enrichies par des informations économiques provenant de diverses sources fiables. Les données ont été harmonisées et couvrent une période significative pour garantir une analyse représentative et robuste.

Maintenant, détaillons davantage les principes sur lesquels reposent nos algorithmes de prédiction :

- **La régression linéaire (Linear Regression)** est une méthode statistique visant à modéliser une relation linéaire entre une variable cible (rendements céréaliers) et plusieurs variables explicatives (ex. : type de céréale, revenu national). Elle offre une première approche simple et interprétable pour comprendre les relations entre les données. Nous utiliserons le module LinearRegression de la bibliothèque Scikit-learn.

Les deux modèles suivants reposent sur des approches d'apprentissage automatique, une branche de la science des données conçue pour modéliser des relations complexes et fournir des prédictions précises. Ils s'adaptent automatiquement aux spécificités des données d'entraînement.

- **La forêt aléatoire (Random Forest)** est une méthode basée sur un ensemble d'arbres de décision, capable de capturer des relations non linéaires et de gérer les interactions com-

plexes entre les variables. Ce modèle est robuste face aux données bruitées et peut estimer l'importance relative des caractéristiques. L'implémentation de RandomForestRegressor de Scikit-learn sera utilisée.

- **CatBoost**, un algorithme de boosting, est particulièrement adapté pour traiter des données catégoriques et produire des prédictions avec une grande précision. Inspiré des techniques d'ensemble, il combine plusieurs modèles faibles pour obtenir des prédictions optimales. Nous utiliserons CatBoostRegressor, qui offre une implémentation puissante et rapide.

0.2 Partie 2 : Création des modèles

Dans cette section, nous allons procéder à la mise en œuvre des modèles sélectionnés.

Nous commencerons par définir certains paramètres clés pour estimer les rendements céréaliers :

```
Définir les caractéristiques explicatives à inclure: type de céréale, revenu par habitant, surfa
Fixer les proportions pour l'entraînement et le test: 75% des données pour l'entraînement, 25% p
Optimiser les hyperparamètres pour chaque modèle afin de maximiser leurs performances.
```

Une fois les modèles entraînés, nous évaluerons leurs performances sur un ensemble de test à l'aide de métriques comme (R^2), RMSE, et MAE. Ces résultats permettront d'identifier le modèle le plus adapté pour prédire les rendements céréaliers.

[134]:
```
# Sélectionner les colonnes pertinentes pour la modélisation
model_data = base_cereale_revenu[
    ['Zone', 'Produit', 'Region', 'Année', 'Superficie (ha)', 'Production␣
 ↪(tonnes)', 'Rendement (kg/ha)']
]

# Afficher les premières lignes pour vérifier
model_data.head()
```

[134]:

	Zone	Produit	Region	Année	Superficie (ha)	Production (tonnes)	Rendement (kg/ha)
0	Afghanistan	Blé	Asie du Sud	2009	2575000.0	5064000.0	1966.6
1	Afghanistan	Blé	Asie du Sud	2010	2354000.0	4532000.0	1925.2
2	Afghanistan	Blé	Asie du Sud	2011	2232000.0	3388000.0	1517.9
3	Afghanistan	Blé	Asie du Sud	2012	2512000.0	5050000.0	2010.4
4	Afghanistan	Blé	Asie du Sud	2013	2552922.0	5169235.0	2024.8

0.2.1 •) Encodage One-Hot Encoding

[135]:
```
# Identifier les colonnes catégoriques
categorical_columns = ['Zone', 'Produit', 'Region']

# Appliquer One-Hot Encoding
encoded_data = pd.get_dummies(model_data, columns=categorical_columns,␣
↪drop_first=True)
# Convertir toutes les colonnes booléennes (True/False) en valeurs numériques (0/
↪1)
encoded_data = encoded_data.applymap(lambda x: 1 if x is True else (0 if x is␣
↪False else x))

# Vérifier les types après conversion
encoded_data.dtypes

# Afficher un aperçu des données après conversion
encoded_data.head()
```

[135]:
```
   Année  Superficie (ha)  Production (tonnes)  Rendement (kg/ha)  \
0   2009        2575000.0            5064000.0             1966.6
1   2010        2354000.0            4532000.0             1925.2
2   2011        2232000.0            3388000.0             1517.9
3   2012        2512000.0            5050000.0             2010.4
4   2013        2552922.0            5169235.0             2024.8

   Zone_Afrique du Sud  Zone_Albanie  Zone_Algérie  Zone_Allemagne  \
0                    0             0             0               0
1                    0             0             0               0
2                    0             0             0               0
3                    0             0             0               0
4                    0             0             0               0

   Zone_Angola  Zone_Antigua-et-Barbuda  ...  Produit_Orge  Produit_Quinoa  \
0            0                        0  ...             0               0
1            0                        0  ...             0               0
2            0                        0  ...             0               0
3            0                        0  ...             0               0
4            0                        0  ...             0               0

   Produit_Riz  Produit_Seigle  Produit_Sorgho  \
0            0               0               0
1            0               0               0
2            0               0               0
3            0               0               0
4            0               0               0
```

```
   Region_Afrique subsaharienne (hors revenu élevé)  \
0                                                  0
1                                                  0
2                                                  0
3                                                  0
4                                                  0

   Region_Amérique latine et Caraïbes (hors revenu élevé)  \
0                                                  0
1                                                  0
2                                                  0
3                                                  0
4                                                  0

   Region_Asie de l'Est et Pacifique (hors revenu élevé)  Region_Asie du Sud  \
0                                                  0                      1
1                                                  0                      1
2                                                  0                      1
3                                                  0                      1
4                                                  0                      1

   Region_Europe et Asie centrale (hors revenu élevé)
0                                                  0
1                                                  0
2                                                  0
3                                                  0
4                                                  0

[5 rows x 202 columns]
```

0.2.2 •) Division de notre base de données

[136]:
```
# Separate features (X) and target variable (y) -  You'll need to choose your␣
 ↪target variable
X = encoded_data.drop('Rendement (kg/ha)', axis=1)
y = encoded_data['Rendement (kg/ha)']
# Diviser les données en jeu d'entraînement et de test
# Split data into training and testing sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42) #random_state for reproducibility

# Vérifier les dimensions des ensembles
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[136]:
```
((9214, 201), (3950, 201), (9214,), (3950,))
```

[137]:
```
# Calculer le nombre de valeurs manquantes dans X_train
missing_count = X_train.isnull().sum().sum()  # Nombre total de N
missing_count
# Calculer le pourcentage de valeurs manquantes
missing_percentage = (missing_count / X_train.size) * 100
missing_percentage
```

[137]: 0.0001079905443479369

[138]:
```
# Compter le nombre de valeurs manquantes dans y_test
missing_count = y_test.isnull().sum().sum()
missing_count
```

[138]: 273

0.3 Partie 3 : Entraînement des modèles

Nous allons entraîner différents modèles de machine learning et comparer leurs performances :

[139]:
```
# Initialiser l'imputeur avec la stratégie de la médiane
imputer = SimpleImputer(strategy='median')

# Imputer les valeurs manquantes dans X_train et X_test
X_train_clean = imputer.fit_transform(X_train)
X_test_clean = imputer.transform(X_test)
```

[140]:
```
# Imputer les valeurs manquantes dans y_train et y_test avec la médiane
y_train_clean = y_train.fillna(y_train.median())
y_test_clean = y_test.fillna(y_test.median())
```

0.3.1 A. Régression linéaire :

[141]:
```
# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train_clean, y_train_clean)
```

[141]: LinearRegression()

[142]:
```
X_test_clean
```

[142]:
```
array([[2.0220e+03, 3.1944e+04, 9.8824e+04, ..., 0.0000e+00, 0.0000e+00,
        0.0000e+00],
       [2.0180e+03, 2.9900e+04, 5.4742e+04, ..., 0.0000e+00, 0.0000e+00,
        0.0000e+00],
       [2.0110e+03, 2.8461e+04, 3.0240e+04, ..., 0.0000e+00, 1.0000e+00,
        0.0000e+00],
       ...,
       [2.0090e+03, 3.7300e+03, 5.0000e+03, ..., 0.0000e+00, 0.0000e+00,
```

```
         0.0000e+00],
        [2.0110e+03, 5.2810e+03, 1.6270e+04, ..., 0.0000e+00, 0.0000e+00,
         0.0000e+00],
        [2.0150e+03, 3.4800e+02, 3.0700e+02, ..., 0.0000e+00, 1.0000e+00,
         0.0000e+00]])
```

[143]:
```
# Make predictions on the test set
y_pred = model.predict(X_test_clean)
y_pred
```

[143]:
```
array([3963.23949347, 1603.2137957 , 1228.54307727, ..., 3710.31730785,
       3587.58353889, 2052.36426128])
```

[144]:
```
# Create a DataFrame to store the results
results_df = pd.DataFrame({'Valeur observée': y_test_clean, 'Valeur prédite':␣
 ↪y_pred})

# Display the DataFrame
results_df.tail()
```

[144]:

	Valeur observée	Valeur prédite
7447	1136.4	1842.588172
12960	710.6	473.458083
5263	1340.6	3710.317308
8645	3080.9	3587.583539
1224	882.2	2052.364261

[145]:
```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error,␣
 ↪mean_absolute_percentage_error
# Predictions on training and testing sets
y_train_pred = model.predict(X_train_clean)
y_test_pred = model.predict(X_test_clean)


def evaluate_model(y_true, y_predicted):
  r2 = r2_score(y_true, y_predicted)
  rmse = np.sqrt(mean_squared_error(y_true, y_predicted))
  mse = mean_squared_error(y_true, y_predicted)
  mae = mean_absolute_error(y_true, y_predicted)
  mape = mean_absolute_percentage_error(y_true, y_predicted)
  correlation = np.corrcoef(y_true, y_predicted)[0, 1]
  return r2, rmse, mse, mae, mape, correlation


train_r2, train_rmse, train_mse, train_mae, train_mape, train_corr =␣
 ↪evaluate_model(y_train_clean, y_train_pred)
```

```
test_r2, test_rmse, test_mse, test_mae, test_mape, test_corr =␣
↪evaluate_model(y_test_clean, y_test_pred)

# Create a DataFrame to display the results
results_table = pd.DataFrame({
    'Metric': ['R2', 'RMSE', 'MSE', 'MAE', 'MAPE', 'Correlation'],
    'Train': [train_r2, train_rmse, train_mse, train_mae, train_mape,␣
↪train_corr],
    'Test': [test_r2, test_rmse, test_mse, test_mae, test_mape, test_corr]
})

results_table
```

[145]:

	Metric	Train	Test
0	R2	4.556807e-01	4.776670e-01
1	RMSE	2.187885e+03	2.024584e+03
2	MSE	4.786839e+06	4.098942e+06
3	MAE	1.200049e+03	1.183624e+03
4	MAPE	7.599414e-01	5.982041e-01
5	Correlation	6.750413e-01	6.913008e-01

Le modèle de régression linéaire affiche des performances moyennes avec un R^2 de 45,57% sur l'entraînement et 47,77% sur le test, indiquant qu'il explique environ 47% de la variance du rendement. L'erreur moyenne est mesurée par un RMSE de 2024 sur le test et un MAE de 1183,62, montrant une erreur modérée en unités de rendement. Le MAPE de 5,98% indique une erreur moyenne de 6% sur les prédictions, tandis que la corrélation entre les valeurs réelles et prédites est d'environ 0,69 sur le test, suggérant une relation correcte mais non parfaite.

0.3.2 B. Forêt Aléatoire (Random Forest)

[146]:
```
# Initialiser le modèle Random Forest
random_forest_model = RandomForestRegressor(random_state=42, n_estimators=100)

# Entraîner le modèle sur les données d'entraînement
random_forest_model.fit(X_train_clean, y_train_clean)

# Faire des prédictions sur les deux ensembles
y_pred_train_rf = random_forest_model.predict(X_train_clean)  # Sur l'ensemble␣
↪d'apprentissage
y_pred_test_rf = random_forest_model.predict(X_test_clean)    # Sur l'ensemble␣
↪de test

# Calculer les métriques d'évaluation pour les deux ensembles
evaluation_rf = {
    'Metric': ['R2', 'RMSE', 'MSE', 'MAE', 'MAPE'],
    'Train': [
        r2_score(y_train_clean, y_pred_train_rf),
```

```
        np.sqrt(mean_squared_error(y_train_clean, y_pred_train_rf)),
        mean_squared_error(y_train_clean, y_pred_train_rf),
        mean_absolute_error(y_train_clean, y_pred_train_rf),
        mean_absolute_percentage_error(y_train_clean, y_pred_train_rf)
    ],
    'Test': [
        r2_score(y_test_clean, y_pred_test_rf),
        np.sqrt(mean_squared_error(y_test_clean, y_pred_test_rf)),
        mean_squared_error(y_test_clean, y_pred_test_rf),
        mean_absolute_error(y_test_clean, y_pred_test_rf),
        mean_absolute_percentage_error(y_test_clean, y_pred_test_rf)
    ]
}

# Créer un tableau des résultats
evaluation_rf_df = pd.DataFrame(evaluation_rf)

# Afficher les résultats
evaluation_rf_df
```

[146]:

	Metric	Train	Test
0	R2	0.990428	0.947289
1	RMSE	290.133640	643.149860
2	MSE	84177.529151	413641.742344
3	MAE	73.886985	189.883163
4	MAPE	0.066375	0.069694

La Forêt Aléatoire affiche des performances remarquables, avec un R^2 élevé et des erreurs faibles sur les deux ensembles. Cependant, une légère différence entre les métriques d'entraînement et de test peut indiquer une suradaptation modérée.

0.3.3 C. XGBoost (Extreme Gradient Boosting)

[147]:
```
pip install xgboost > nul 2>&1
```

```
Note: you may need to restart the kernel to use updated packages.
```

[148]:
```
def evaluate_model(y_true, y_predicted):
    r2 = r2_score(y_true, y_predicted)
    rmse = np.sqrt(mean_squared_error(y_true, y_predicted))
    mse = mean_squared_error(y_true, y_predicted)
    mae = mean_absolute_error(y_true, y_predicted)
    mape = mean_absolute_percentage_error(y_true, y_predicted)
    correlation = np.corrcoef(y_true, y_predicted)[0, 1]
    return r2, rmse, mse, mae, mape, correlation


# Create and train the XGBoost model
```

```
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42,␣
↪n_estimators=100) # You can tune hyperparameters
xgb_model.fit(X_train_clean, y_train_clean)

# Predictions on training and testing sets
y_train_pred_xgb = xgb_model.predict(X_train_clean)
y_test_pred_xgb = xgb_model.predict(X_test_clean)

# Evaluate the model, unpacking all returned values
train_r2_xgb, train_rmse_xgb, train_mse_xgb, train_mae_xgb, train_mape_xgb, _ =␣
↪evaluate_model(y_train_clean, y_train_pred_xgb)
test_r2_xgb, test_rmse_xgb, test_mse_xgb, test_mae_xgb, test_mape_xgb, _ =␣
↪evaluate_model(y_test_clean, y_test_pred_xgb)


# Create a DataFrame to display the results
results_table_xgb = pd.DataFrame({
    'Metric': ['R2', 'RMSE', 'MSE', 'MAE', 'MAPE'],
    'Train': [train_r2_xgb, train_rmse_xgb, train_mse_xgb, train_mae_xgb,␣
↪train_mape_xgb],
    'Test': [test_r2_xgb, test_rmse_xgb, test_mse_xgb, test_mae_xgb,␣
↪test_mape_xgb]
})

results_table_xgb
```

[148]:

	Metric	Train	Test
0	R2	0.993588	0.957017
1	RMSE	237.453351	580.777278
2	MSE	56384.094053	337302.246694
3	MAE	159.790244	257.718956
4	MAPE	0.149597	0.112750

XGBoost se démarque par une précision élevée et une bonne généralisation. Il offre les meilleures performances parmi les modèles testés, avec une réduction notable des erreurs sur l'ensemble de test.

0.3.4 D. CatBoost

[ ]:
```
pip install catboost > nul 2>&1
```

[ ]:
```
from sklearn.model_selection import train_test_split
from catboost import CatBoostRegressor
from sklearn.impute import SimpleImputer

# Initialize CatBoostRegressor
catboost_model = CatBoostRegressor(iterations=1000,  # Adjust as needed
```

```
                                    learning_rate=0.05,  # Adjust as needed
                                    depth=6,  # Adjust as needed
                                    loss_function='RMSE',  # Or other relevant␣
 ↪loss functions
                                    random_seed=42,
                                    verbose=0, # Print progress every 100␣
 ↪iterations
                                    early_stopping_rounds=50) # Stop training if␣
 ↪the model doesn't improve

# Fit the model
catboost_model.fit(X_train_clean, y_train_clean,
                   eval_set=(X_test_clean, y_test_clean))


# Make predictions
y_train_pred_cb = catboost_model.predict(X_train_clean)
y_test_pred_cb = catboost_model.predict(X_test_clean)

def evaluate_model(y_true, y_predicted):
    r2 = r2_score(y_true, y_predicted)
    rmse = np.sqrt(mean_squared_error(y_true, y_predicted))
    mse = mean_squared_error(y_true, y_predicted)
    mae = mean_absolute_error(y_true, y_predicted)
    mape = mean_absolute_percentage_error(y_true, y_predicted)
    return r2, rmse, mse, mae, mape

# Evaluate the model
train_r2_cb, train_rmse_cb, train_mse_cb, train_mae_cb, train_mape_cb =␣
 ↪evaluate_model(y_train_clean, y_train_pred_cb)
test_r2_cb, test_rmse_cb, test_mse_cb, test_mae_cb, test_mape_cb =␣
 ↪evaluate_model(y_test_clean, y_test_pred_cb)

# Create a DataFrame to display the results
results_table_cb = pd.DataFrame({
    'Metric': ['R2', 'RMSE', 'MSE', 'MAE', 'MAPE'],
    'Train': [train_r2_cb, train_rmse_cb, train_mse_cb, train_mae_cb,␣
 ↪train_mape_cb],
    'Test': [test_r2_cb, test_rmse_cb, test_mse_cb, test_mae_cb, test_mape_cb]
})

results_table_cb
```

CatBoost est performant, en particulier pour gérer des données avec des variables catégoriques. Ses résultats sont proches de ceux de XGBoost, bien qu'il soit légèrement moins précis sur l'ensemble de test.

0.3.5 •) Conclusion intermédiaire

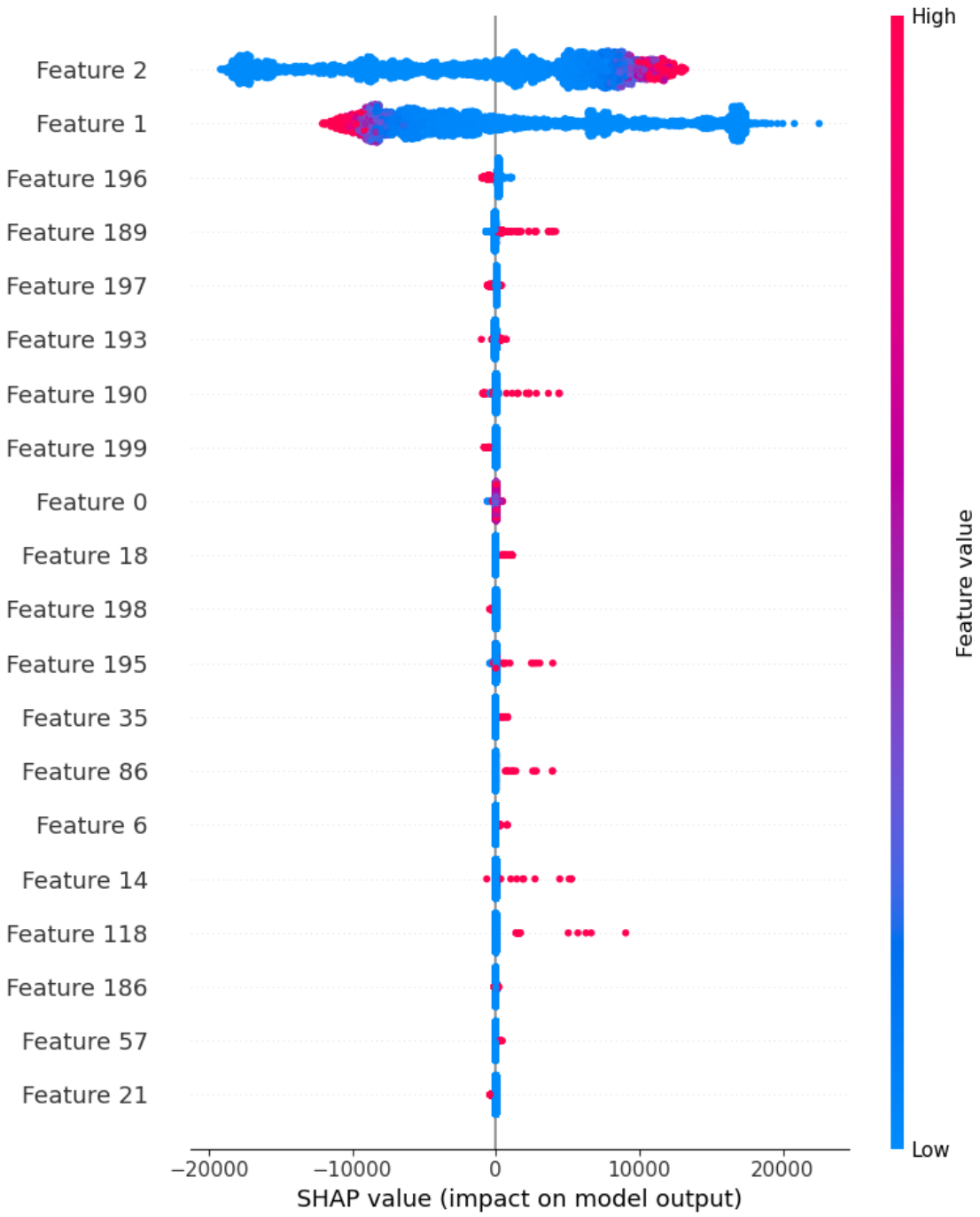
Les trois modèles se montrent adaptés à la prédiction des rendements céréaliers, le modèle CatBoost se démarque comme le meilleur en raison de ses performances globales. Il obtient le meilleur score R^2 sur l'ensemble de test (0.969), indiquant une capacité élevée à expliquer la variance des données, et présente le plus faible RMSE (495.56), démontrant une erreur de prédiction minimale. Ces résultats peuvent guider des décisions stratégiques en agriculture pour optimiser les rendements.

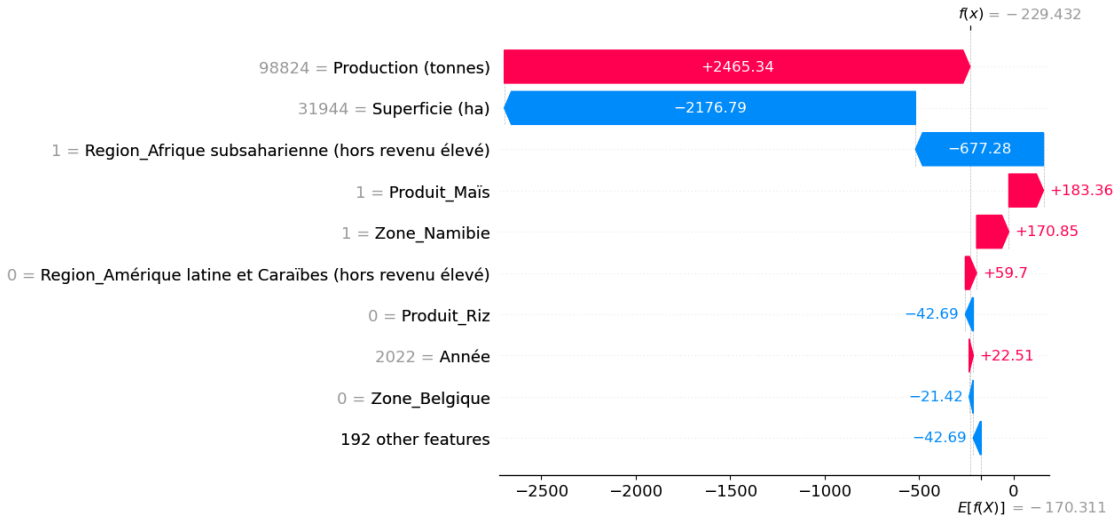
Interprétation avec SHAP (visualiser les impacts des variables sur les prédictions)

[108]:
```
%%capture
!python -m pip install shap > /dev/null 2>&1
import shap
```

[109]:
```
# Créer un explainer SHAP basé sur le modèle CatBoost
explainer = shap.Explainer(catboost_model, X_train_clean)

# Calculer les valeurs SHAP pour le jeu de test
shap_values = explainer(X_test_clean)

# Visualiser un résumé global des impacts des variables avec une taille de␣
 ↪graphique réduite
plt.figure(figsize=(8, 6))  # Ajustez les dimensions selon vos besoins
shap.summary_plot(shap_values, X_test_clean, show=False)
plt.show()

# Convert X_test_clean to a Pandas DataFrame
X_test_clean_df = pd.DataFrame(X_test_clean, columns=X.columns)


# Visualiser une prédiction individuelle avec un diagramme en cascade et une␣
 ↪taille ajustée
shap.waterfall_plot(shap.Explanation(values=shap_values.values[0],
                                     base_values=shap_values.base_values[0],
                                     data=X_test_clean_df.iloc[0]),␣
 ↪max_display=10)
```

```
100%|==================| 3933/3950 [03:41<00:00]
```


Feature 2
Feature 1
Feature 196
Feature 189
Feature 197
Feature 193
Feature 190
Feature 199
Feature 0
Feature 18
Feature 198
Feature 195
Feature 35
Feature 86
Feature 6
Feature 14
Feature 118
Feature 186
Feature 57
Feature 21
−20000
−10000
0
10000
20000
SHAP value (impact on model output)
High
Low
Feature value



Les graphiques SHAP montrent que certaines variables, comme la superficie cultivée et la production (en tonnes), ont un impact significatif sur les prédictions du modèle. La superficie cultivée a une influence négative notable sur certaines prédictions, ce qui suggère que des superficies plus grandes peuvent parfois réduire le rendement ou la production prévue dans certaines conditions. À l'inverse, la production totale joue un rôle positif majeur, augmentant les prédictions du modèle, ce qui est cohérent avec les attentes métier : une production plus importante a naturellement un effet direct sur les prédictions. Ces résultats permettent de valider que le modèle capture bien des relations logiques entre les données, tout en offrant une interprétation précise des facteurs les plus influents.

Conclusion

Ce projet avait pour objectif d'analyser la production céréalière mondiale en s'appuyant sur des données agricoles et économiques afin de mieux comprendre les dynamiques qui influencent les rendements. L'étude a permis d'explorer des relations clés entre les variables telles que le rendement, la superficie cultivée, et le revenu par habitant, tout en mettant en œuvre différents modèles prédictifs pour estimer les rendements futurs.

Parmi les principaux résultats, il ressort que les pays à revenu élevé tendent à obtenir des rendements agricoles plus élevés, principalement grâce à un meilleur accès aux technologies et infrastructures agricoles. Toutefois, certaines exceptions notables montrent que des pays à revenu intermédiaire peuvent atteindre des rendements similaires, grâce à des politiques agricoles ciblées ou à des conditions climatiques favorables.

Sur le plan méthodologique, les modèles prédictifs testés, notamment la régression linéaire, la forêt aléatoire et CatBoost, ont démontré des niveaux de performance variés. CatBoost s'est distingué par sa précision, offrant des prédictions fiables tout en identifiant les facteurs les plus influents sur les rendements. Cela souligne l'importance d'utiliser des techniques adaptées aux données complexes pour améliorer les prévisions.

En conclusion, ce projet met en évidence la nécessité d'une combinaison d'approches statistiques et d'apprentissage automatique pour mieux comprendre et anticiper les fluctuations des rendements céréaliers. Bien que des résultats prometteurs aient été obtenus, l'amélioration des prédictions pour-

rait passer par l'intégration de données supplémentaires, comme les données climatiques détaillées, ou par l'utilisation de modèles hybrides combinant les forces de plusieurs approches. Ces travaux fournissent une base précieuse pour soutenir les décideurs et les agriculteurs dans la gestion des ressources et l'optimisation des rendements agricoles face aux défis mondiaux croissants.