Impacts of climate change on food Security

Mohamed Fadlelseed
Fulchany Arvelin Nanitelamio
Bacoro dit Elhadj lansar
Madalena Makiesse
Zeinab Mohmmed
Israa Abdulrahman Mohammed kheir

Date: 13/10/2024

Outline

- Concept note and implementation plan
    - Background
    - Objectives
    - SDG Relation
- Data
    - Data Collection
    - Exploratory Data Analysis et Feature Engineering
- Model
    - Model Selection and Training
    - Model Evaluation and Hyperparameter Tuning
    - Model Refinement and Testing
- Results
    - Evaluation Results
    - Deployment
    - Future Work

Concept note and Implementation Plan

Background

- Climate change poses a significant threat to food security by affecting crop yields, water availability, and livestock productivity.
- Addressing climate change is essential for ensuring food security and combating hunger. Conversely, improving food security can also mitigate the impacts of climate change by reducing greenhouse gas emissions associated with agriculture and food production.
- Projects those address both SDG 13 and SDG 2 are critical for creating a more sustainable and equitable world.
- Traditional climate models project future climate scenarios but often lack specific predictions on their effects on food security.
- Systems exist for predicting weather patterns and natural disasters but may not link these predictions directly to food security outcomes.

frontier tech leaders

Objectives

1. Develop predictive models to estimate the impact of climate change on crop yields using historical climate and agricultural data.
2. Identify and analyze key climate variables that significantly affect food security.
3. Enhance understanding of regional vulnerabilities to climate change and suggest targeted interventions.

frontier tech leaders

SDG Relation

- Our project tackles the 2nd and 13th Sustainable Development Goals, concentrating on zero hunger and climate action.
- By integrating these objectives, we seek to examine the impact of climate change on food security by predicting essential factors like crop yield, cereal production, and undernourishment, utilizing climatological data as inputs.

the source of the image

the source of the image

frontier tech leaders

Data

Data Collection

❑ Sources of the dataset:

1. The Humanitarian Data Exchange (humdata.org)where we sourced data concerning natural resources and climate data, among others.

2.The FAOSTAT database , from which we have obtained data pertaining to food security.

Preprocessing steps during data collection:

➢ For The Humanitarian Data Exchange (humdata.org):-

- Preprocessing during the data collection phase included cleaning, normalizing, and aggregating the data from different sources.
- Datasets were downloaded in CSV format and prepared for feature engineering.

```
In [8]: directory = r'D:\Capstone\Countries'

        features = [
            'CO2 emissions (kt)',
            'Other greenhouse gas emissions, HFC, PFC and SF6 (thousand metric tons of CO2 equivale
            'Methane emissions (kt of CO2 equivalent)',
            'Average precipitation in depth (mm per year)',
            'Droughts, floods, extreme temperatures (% of population affected)',
            'Annual freshwater withdrawals, total (% of internal resources)',
            'Cereal yield (kg per hectare)'
        ]

        all_dataframes = []

        files = [f for f in os.listdir(directory) if f.endswith('.csv')]

        for file in files:
            file_path = os.path.join(directory, file)

            df = pd.read_csv(file_path)

            if 'Indicator Name' in df.columns:
                filtered_df = df[df['Indicator Name'].isin(features)]

                all_dataframes.append(filtered_df)

                print(f'Processed {file}')
```

```
        print(f'Processed {file}')
    else:
        print(f'Column "Indicator Name" not found in {file}')

if all_dataframes:
    merged_df = pd.concat(all_dataframes, ignore_index=True)

    output_file = r'D:\Capstone\agriculture\cereal.csv'
    merged_df.to_csv(output_file, index=False)

    print(f'Merged data saved to {output_file}')
else:
    print('No valid data to merge.')
```

```
Processed Burkina Faso.csv
Processed Burundi.csv
Processed Cabo Verde.csv
Processed Cambodia.csv
Processed Cameroon.csv
```

➢ Setting Directory and Defining Features:

- The script starts by defining a directory path (D:\Capstone\Countries) where the dataset files are stored.
- A list of relevant features (indicators) is also provided, which include metrics like CO2 emissions, methane emissions, cereal yield, and more. These features are later used to filter the dataset.

➢ Reading and Filtering Data:

- Files Listing: All CSV files in the specified directory are listed and iterated over. Only files with the .csv extension are processed.
- Reading CSV Files: For each CSV file, pd.read_csv() is used to load the data into a DataFrame.
- Filtering by Indicator: The script checks if the file contains the Indicator Name column. If present, the DataFrame is filtered to retain only rows where Indicator Name matches one of the specified features. These filtered DataFrames are collected in a list (all_dataframes).

➢ Concatenating Data:

- Once all the files are processed, the script checks if any filtered DataFrames were collected. If so, the individual DataFrames are concatenated using pd.concat(), combining all the filtered data into a single DataFrame (merged_df).
- Saving the Merged Data: The combined dataset is saved as a new CSV file (cereal.csv), allowing the processed data to be stored for future use.

- Removing unnecessary columns, pivoted the data, and renamed columns for clarity. Finally, we saved the cleaned, reshaped, and renamed data to a new CSV file (cerealmixed2.csv), providing a dataset that is ready for analysis with clear column names and a structured format.

In [9]:
```
df = pd.read_csv("cereal.csv")
df.head()
```

Out[9]:

	Country Name	Country ISO3	Year	Indicator Name	Indicator Code	Value
0	Afghanistan	AFG	2020	Average precipitation in depth (mm per year)	AG.LND.PRCP.MM	327.0
1	Afghanistan	AFG	2019	Average precipitation in depth (mm per year)	AG.LND.PRCP.MM	327.0
2	Afghanistan	AFG	2018	Average precipitation in depth (mm per year)	AG.LND.PRCP.MM	327.0
3	Afghanistan	AFG	2017	Average precipitation in depth (mm per year)	AG.LND.PRCP.MM	327.0
4	Afghanistan	AFG	2016	Average precipitation in depth (mm per year)	AG.LND.PRCP.MM	327.0

In [10]:
```
df.drop(columns=["Country ISO3", "Indicator Code"], inplace=True)
df.head()
```

Out[10]:

	Country Name	Year	Indicator Name	Value
0	Afghanistan	2020	Average precipitation in depth (mm per year)	327.0
1	Afghanistan	2019	Average precipitation in depth (mm per year)	327.0
2	Afghanistan	2018	Average precipitation in depth (mm per year)	327.0

➢ Handling missing values

- In order to fill missing values, fillna with different methods, such as mean, ffill, etc. were being used, and all that after making a copy of the original dataset with a variable we called df2 that contains just data for year >= 2000.

```
In [1]: # Importing Libraries
import geopandas as gpd
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.offline as py
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, BaggingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, mean_absolute_percentage_error
from sklearn.model_selection import cross_val_score, KFold, train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.tree import DecisionTreeRegressor
from xgboost import XGBRegressor
```

```
In [2]: df = pd.read_csv("cerealmixed.csv")
df.head()
```

Out[2]:

	Country Name	Year	Freshwater withdrawals (% internal)	Avg precipitation (mm/year)	CO2 emissions (kt)	Cereal yield (kg/hectare)	Methane emissions (kt CO2 eq.)	Other GHG emissions (thousand tons)
0	Afghanistan	1961	NaN	327.0	NaN	1115.1	NaN	NaN
1	Afghanistan	1962	NaN	327.0	NaN	1079.0	NaN	NaN
2	Afghanistan	1963	NaN	327.0	NaN	985.8	NaN	NaN

```
In [10]: df['Year'].unique()

Out[10]: array([1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971,
         1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982,
         1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993,
         1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004,
         2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015,
         2016, 2017, 2018, 2019, 2020, 2021, 2022], dtype=int64)
```

```
In [11]: df2 = df[df['Year'] >= 2000]
         df2['Year'].unique()

Out[11]: array([2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010,
         2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021,
         2022], dtype=int64)
```

```
mean_cereal_yield_by_country = df2.groupby('Country')['Cereal yield (kg/hectare)'].mean()

# Remplacez les valeurs nulles par la moyenne pour chaque pays
df2['Cereal yield (kg/hectare)'] = df2.apply(
    lambda row: mean_cereal_yield_by_country[row['Country']]
    if pd.isnull(row['Cereal yield (kg/hectare)'])
    else row['Cereal yield (kg/hectare)'],
    axis=1
)

# Vérifiez que les valeurs nulles ont été remplacées
null_values_after = df2['Cereal yield (kg/hectare)'].isnull().sum()
print(f"Nombre de valeurs nulles après remplacement par la moyenne : {null_values_after}")
```

```
Nombre de valeurs nulles après remplacement par la moyenne : 21
```

```
countries_with_null_cereal_yield = df2[df2['Cereal yield (kg/hectare)'].isnull()]['Country'].unique()

print(f"Pays avec des valeurs nulles pour 'Cereal yield (kg/hectare)':\n{countries_with_null_cereal_yield}")
```

```
Pays avec des valeurs nulles pour 'Cereal yield (kg/hectare)':
['St. Lucia']
```

```
df2 = df2[df2['Country'] != 'St. Lucia']
```

```
df2.isna().sum()
```

```
Country                                   0
```


```
In [29]: countries_with_null_cereal_yield = df2[df2['Cereal yield (kg/hectare)'].isnull()]['Country'].unique()

         print(f"Pays avec des valeurs nulles pour 'Cereal yield (kg/hectare)':\n{countries_with_null_cereal_yield}")

         Pays avec des valeurs nulles pour 'Cereal yield (kg/hectare)':
         ['St. Lucia']

In [30]: df2 = df2[df2['Country'] != 'St. Lucia']

In [31]: df2.isna().sum()
```


```
Out[31]: Country                                   0
         Year                                      0
         Freshwater withdrawals (% internal)       0
         Avg precipitation (mm/year)               0
         CO2 emissions (kt)                        0
         Cereal yield (kg/hectare)                 0
         Methane emissions (kt CO2 eq.)            0
         Other GHG emissions (thousand tons)       0
         dtype: int64
```

Exploratory Data Analysis and Feature Engineering

- Process of creating or transforming features

```
df = pd.read_csv("yieldmix.csv")
df.head()
```

	Unnamed: 0	Area	Item	Year	hg/ha_yield	average_rain_fall_mm_per_year	pesticides_tonnes	avg_temp
0	0	Albania	Maize	1990	36613	1485.0	121.0	16.37
1	1	Albania	Potatoes	1990	66667	1485.0	121.0	16.37
2	2	Albania	Rice, paddy	1990	23333	1485.0	121.0	16.37
3	3	Albania	Sorghum	1990	12500	1485.0	121.0	16.37
4	4	Albania	Soybeans	1990	7000	1485.0	121.0	16.37

```
df.tail()
```

Exploratory Data Analysis and Feature Engineering

The dataset contains categorical variables that are not in a numerical format, which machine learning algorithms cannot process directly. To transform these categorical variables, we applied one-hot encoding using pd.get_dummies().

This method converts each category into a new binary column (dummy variables), where a value of 1 indicates the presence of the category, and 0 indicates its absence.

The rationale behind this transformation is to represent the categorical data in a format suitable for models while avoiding assigning any ordinal relationship between the categories, which could lead to biased predictions.

In [41]:
```
X = df.drop(labels = 'hg/ha_yield', axis = 1)
y = df['hg/ha_yield']

# Convert data into dummy variables
X = pd.get_dummies(X)
```

In [42]:
```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

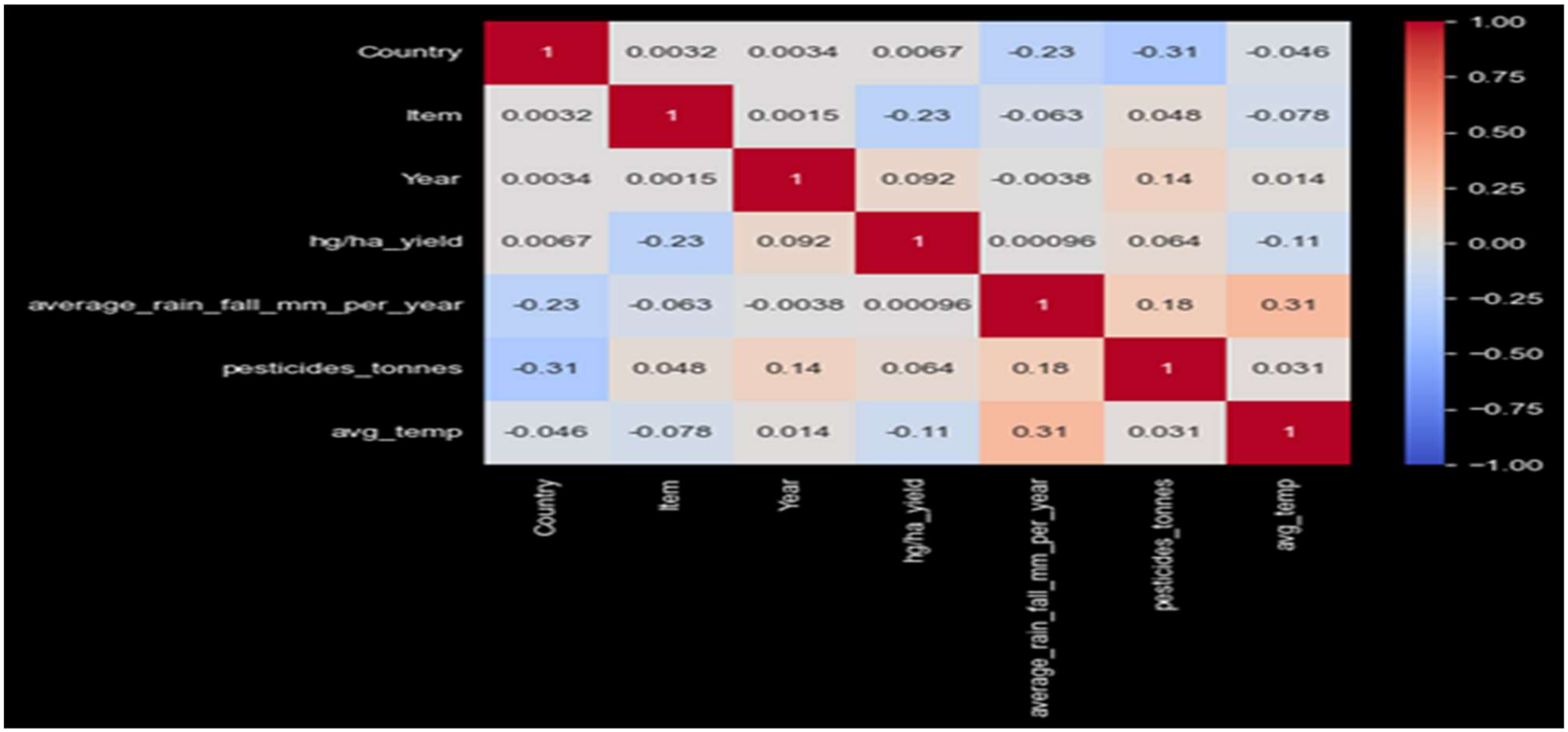
In [43]:
```
X_train.head()
```

Out[43]:

	Year	average_rain_fall_mm_per_year	pesticides_tonnes	avg_temp	Country_Albania	Country_Algeria	Country_Angola	Country_Argentina	Country_Arme
5493	2005	1604.0	829.59	25.36	False	False	False	False	Fa
10969	1992	1083.0	70791.00	25.91	False	False	False	False	Fa
2001	1997	1292.0	484.59	25.81	False	False	False	False	Fa
22157	1997	494.0	16936.00	23.76	False	False	False	False	Fa
311	2005	1010.0	40.00	24.41	False	False	True	False	Fa

5 rows × 115 columns

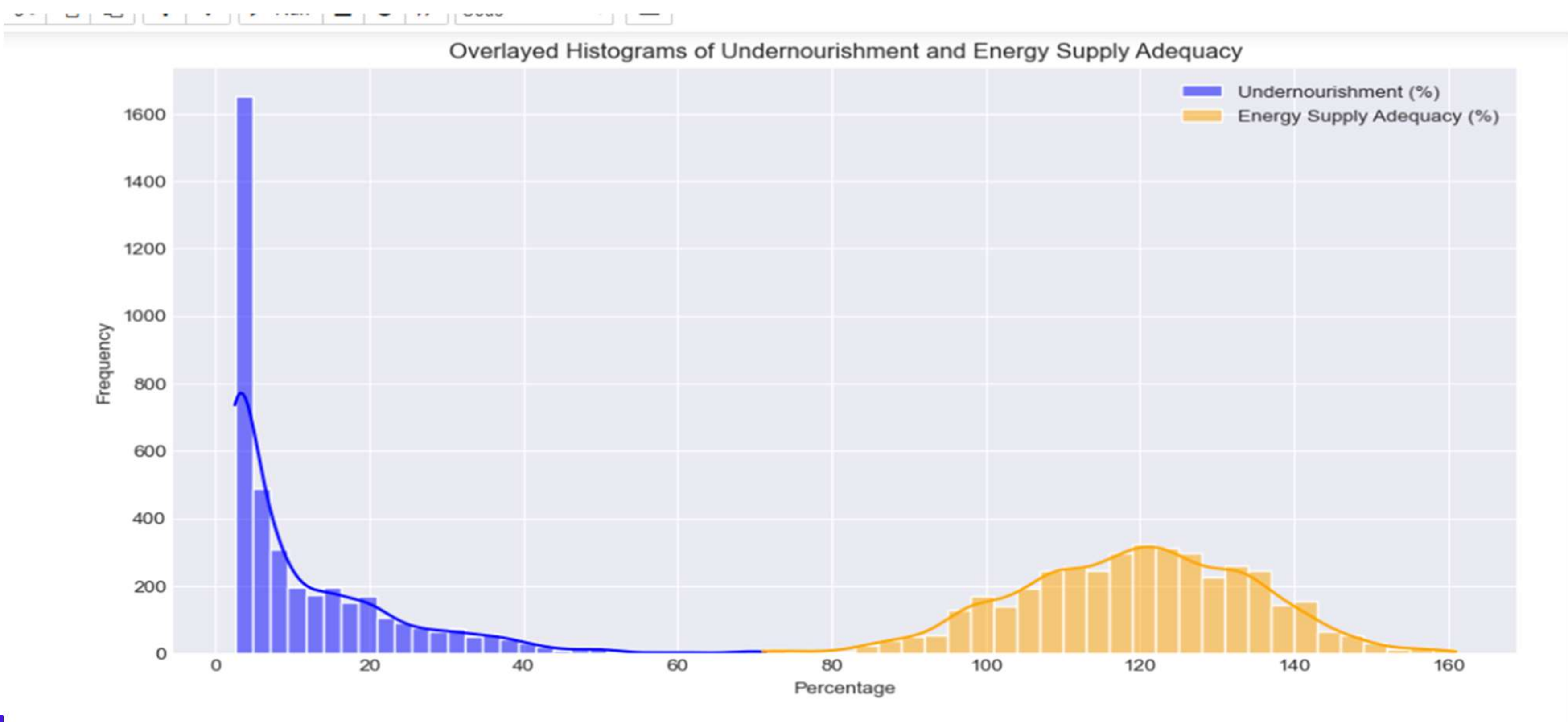
Exploratory Data Analysis and Feature Engineering



frontier tech leaders

Exploratory Data Analysis and Feature Engineering

Exploratory Data Analysis and Feature Engineering



Model

Model Selection and Training

Predictions

We have made 3 different predictions, as follow:

- We firstly predicted the cereal yield.
- Secondly, we predicted the crop yield.
- Lastly, we predicted the undernourishment.

```
]: # Dataframe consisting of metrics of all the models
result_df = pd.DataFrame(results, columns = ['Model', 'Accuracy', 'MSE'
# Add red and green highlights in the dataframe to display best and wor
result_format_df = result_df.style.highlight_max(subset = ['Accuracy','
display(result_format_df)
```

	Model	Accuracy	MSE	MAE	MAPE	R2_score
0	Linear Regression	0.913128	642501.979524	435.664811	0.221335	0.913128
1	Decision Tree	0.875673	919518.205001	383.257054	0.132909	0.875673
2	Random Forest	0.927724	534548.435991	314.961583	0.113387	0.927724
3	Gradient Boost	0.822000	1316479.623035	898.048053	0.534209	0.822000
4	XGBoost	0.934655	483288.263094	370.004805	0.168408	0.934655
5	Bagging Regressor	0.928170	531249.323113	315.874675	0.112139	0.928170
6	KNN	0.733158	1973552.240273	783.503357	0.325711	0.733158

```
]: from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 6, 9],
    'learning_rate': [0.01, 0.1, 0.2],
```

Model Selection and Training

All of the models except for the Linear Regression and Random Forest models have the highest CV score within the 1 - 5 folds.

In [49]:
```
# Dataframe consisting of metrics of all the models
result_df = pd.DataFrame(results, columns = ['Model', 'Accuracy', 'MSE', 'MAE', 'MAPE', 'R2_score'])
# Add red and green highlights in the dataframe to display best and worst performing models
result_format_df = result_df.style.highlight_max(subset = ['Accuracy','R2_score'], color = 'green').highlight_min(subset = ['MSE'
display(result_format_df)
```

	Model	Accuracy	MSE	MAE	MAPE	R2_score
0	Linear Regression	0.755142	1776116996.515923	29582.494556	0.875027	0.755142
1	Decision Tree	0.979028	152126879.690211	3674.954505	0.073275	0.979028
2	Random Forest	0.987491	90736304.678697	3472.209294	0.075039	0.987491
3	Gradient Boost	0.873241	919466056.192313	19396.278598	0.529985	0.873241
4	XGBoost	0.975929	174599934.341565	7720.576922	0.213953	0.975929
5	Bagging Regressor	0.987528	90464333.251987	3469.872735	0.074980	0.987528
6	KNN	0.332020	4845307069.320509	48062.047654	1.533120	0.332020

Utilizing K-Fold cross-validation, the Bagging Regressor model still remains the best model. The KNN still remains the worst model. There is no significant change in any of the models' accuracy when using K-Fold cross-validation

In [50]:
```
df.columns
```

frontier tech leaders

Model Selection and Training

We have made 3 different predictions, as follo

- We firstly predicted the cereal yield.
- Secondly, we predicted the crop yield.
- Lastly, we predicted the undernourishment.

```
# Add red and green highlights in the dataframe to display
results_format_df = results_df.style.highlight_max(subset =
display(results_format_df)
```

	Model	Accuracy	MSE	R2_score
0	Linear Regression	0.940822	7.972303	0.940822
1	Decision Tree	0.976387	3.181061	0.976387
2	Random Forest	0.986245	1.853085	0.986245
3	Gradient Boost	0.972070	3.762675	0.972070
4	XGBoost	0.990710	1.251478	0.990710
5	Bagging Regressor	0.986120	1.869931	0.986120
6	KNN	0.950578	6.657921	0.950578
7	Support Vector Regressor	0.788836	28.447365	0.788836
8	Elastic Net	0.756269	32.834661	0.756269
9	Ridge Regression	0.940055	8.075538	0.940055
10	Lasso Regression	0.769577	31.041818	0.769577
11	CatBoost	0.991754	1.110823	0.991754
12	Stacking Regressor	0.987569	1.674715	0.987569
13	Neural Network	0.881535	15.959217	0.881535
14	Hist Gradient Boosting	0.980250	2.660659	0.980250

```
In [54]: # Define the parameter grid for CatBoost
```

frontier tech leaders

Model Evaluation and Hyperparameter Tuning

For the 3 predictions we used the GridsearchCV method to find the best parameters for each one of our 3 best models, as follow:

4	XGBoost	0.934655	483288.263094	370.004805	0.168408	0.934655
5	Bagging Regressor	0.928170	531249.323113	315.874675	0.112139	0.928170
6	KNN	0.733158	1973552.240273	783.503357	0.325711	0.733158

In [65]:

```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 6, 9],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0]
}

# Initialize the XGBoost model
xgboost_model = XGBRegressor(random_state=1)

# Setup GridSearchCV
grid_search = GridSearchCV(estimator=xgboost_model, param_grid=param_grid, cv=5)

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Print the best parameters
print(f"Best Parameters: {grid_search.best_params_}")
```

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 200, 'subsample': 0.8}
```

Model Evaluation and Hyperparameter Tuning

0	Linear Regression	0.755142	1776116996.515923	29582.494556	0.875027	0.755142
1	Decision Tree	0.979028	152126879.690211	3674.954505	0.073275	0.979028
2	Random Forest	0.987491	90736304.678697	3472.209294	0.075039	0.987491
3	Gradient Boost	0.873241	919466056.192313	19396.278598	0.529985	0.873241
4	XGBoost	0.975929	174599934.341565	7720.576922	0.213953	0.975929
5	Bagging Regressor	0.987528	90464333.251987	3469.872735	0.074980	0.987528
6	KNN	0.332020	4845307069.320509	48062.047654	1.533120	0.332020

Utilizing K-Fold cross-validation, the Bagging Regressor model still remains the best model. The KNN still remains the
change in any of the models' accuracy when using K-Fold cross-validation

In [50]:
```
df.columns
```

Out[50]:
```
Index(['Country', 'Item', 'Year', 'hg/ha_yield',
       'average_rain_fall_mm_per_year', 'pesticides_tonnes', 'avg_temp'],
      dtype='object')
```

In [51]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {'n_estimators': [50, 100, 200], 'max_features': [0.5, 1.0]}
bagging_grid = GridSearchCV(BaggingRegressor(random_state=1), param_grid, cv=5)
bagging_grid.fit(X_train, y_train)
print(f"Best Parameters: {bagging_grid.best_params_}")
```

```
Best Parameters: {'max_features': 1.0, 'n_estimators': 200}
```

Model Evaluation and Hyperparameter Tuning

10	Lasso Regression	0.769577	31.041818	0.769577
11	CatBoost	0.991754	1.110823	0.991754
12	Stacking Regressor	0.987569	1.674715	0.987569
13	Neural Network	0.881535	15.959217	0.881535
14	Hist Gradient Boosting	0.980250	2.660659	0.980250

54]:

```
# Define the parameter grid for CatBoost
param_grid = {
    'iterations': [100, 200, 500],
    'depth': [3, 6, 9],
    'learning_rate': [0.01, 0.1, 0.2],
    'l2_leaf_reg': [1, 3, 5],  # L2 regularization term
    'subsample': [0.8, 1.0]
}

# Initialize the CatBoost model
catboost_model = CatBoostRegressor(random_state=1, verbose=0)

# Setup GridSearchCV
grid_search = GridSearchCV(estimator=catboost_model, param_grid=param_grid, cv=5, n_jobs=-1)

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Print the best parameters
print(f"Best Parameters: {grid_search.best_params_}")
```

```
Best Parameters: {'depth': 6, 'iterations': 500, 'l2_leaf_reg': 1, 'learning_rate': 0.2, 'subsample': 1.0}
```

Model Evaluation and Hyperparameter Tuning

```
Run  Code
        y.append(data[i + time_steps, -1])     # The target (Undernourishment)
    return np.array(x), np.array(y)

# Create training data for the country
X, y = create_sequences(df_scaled, time_steps)

# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X.shape[1], X.shape[2])))
model.add(LSTM(units=50))
model.add(Dense(1))  # Output layer to predict the undernourishment percentage

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model on the country's data
model.fit(X, y, epochs=50, batch_size=32)

# Predict the future undernourishment percentages for 2023, 2024, and 2025
future_predictions = []
current_input = df_scaled[-time_steps:, :-1].reshape(1, time_steps, len(features))  # Reshape for LSTM input

for _ in range(3):  # Predict for 3 years
    predicted_undernourishment = model.predict(current_input)

    # Inverse transform the prediction (only for the target column)
    predicted_undernourishment = scaler.inverse_transform(
        np.hstack([np.zeros((1, len(features))), predicted_undernourishment])
    )[:, -1]

    future_predictions.append(predicted_undernourishment[0])
```

```
Epoch 25/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 0.0333
Epoch 26/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 0.0327
Epoch 27/50
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - loss: 0.0320
```

```
In [73]: # Display the metrics for each year
print(f"Metrics for 2023: MAE: {mae_2023:.4f}, MSE: {mse_2023:.4f}, RMSE: {rmse_2023:.4f}, R²: {r2_2023:.4f}")
print(f"Metrics for 2024: MAE: {mae_2024:.4f}, MSE: {mse_2024:.4f}, RMSE: {rmse_2024:.4f}, R²: {r2_2024:.4f}")
print(f"Metrics for 2025: MAE: {mae_2025:.4f}, MSE: {mse_2025:.4f}, RMSE: {rmse_2025:.4f}, R²: {r2_2025:.4f}")

Metrics for 2023: MAE: 1.1421, MSE: 4.1198, RMSE: 2.0297, R²: 0.9617
Metrics for 2024: MAE: 2.9079, MSE: 28.3223, RMSE: 5.3219, R²: 0.7494
Metrics for 2025: MAE: 4.7200, MSE: 74.7737, RMSE: 8.6472, R²: 0.3694
```

Model Refinement and Testing

We used the following metrics to evaluate model performance on the test dataset:

- Accuracy
- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- Mean Absolute Percentage Error (MAPE)
- R^2 Score

The test results were consistent with the cross-validation scores from the training phase, indicating that the models generalized well to unseen data. For example, the Bagging Regressor had an accuracy of 98.88% on the test set, closely matching its cross-validation performance.

Model Refinement and Testing

```python
# Loops through the list of machine learning models above
for name, model in models :
    # Train Model
    model.fit(X_train, y_train)
    # Make Predictions
    y_pred = model.predict(X_test)

    accuracy = model.score(X_test, y_test)
    MSE = mean_squared_error(y_test, y_pred)
    R2_score = r2_score(y_test, y_pred)
    # Add all metrics of model to a list
    results.append((name, accuracy, MSE, R2_score))

    acc = (model.score(X_train , y_train) * 100)
    print(f'Accuracy of {name} Model Train is {acc:.2f}')
    acc = (model.score(X_test , y_test) * 100)
    print(f'Accuracy of the {name} Model Test is {acc:.2f}')

    data = {'y_test' : [y_test],
            'y_pred' : [y_pred]}
    data_df = pd.DataFrame(data)

    fig = px.scatter(data_df, x = y_test, y = y_pred,
                     labels = {'x' : 'Actual Values', 'y' : 'Predicted Values'},
                     trendline = 'ols', trendline_color_override = 'red',
                     template = 'plotly_dark')
    fig.show()
```

Results

Evaluation Results

1- Cereal Yield Model

```
# Dataframe consisting of metrics of all the models
result_df = pd.DataFrame(results, columns = ['Model', 'Accuracy', 'MSE', 'MAE', 'MAPE', 'R2_score'])
# Add red and green highlights in the dataframe to display best and worst performing models
result_format_df = result_df.style.highlight_max(subset = ['Accuracy','R2_score'], color = 'green').highlight_
display(result_format_df)
```

✓ 0.0s Python

	Model	Accuracy	MSE	MAE	MAPE	R2_score
0	Linear Regression	0.913128	642501.979524	435.664811	0.221335	0.913128
1	Decision Tree	0.875673	919518.205001	383.257054	0.132909	0.875673
2	Random Forest	0.927724	534548.435991	314.961583	0.113387	0.927724
3	Gradient Boost	0.822000	1316479.623035	898.048053	0.534209	0.822000
4	XGBoost	0.934655	483288.263094	370.004805	0.168408	0.934655
5	Bagging Regressor	0.928170	531249.323113	315.874675	0.112139	0.928170
6	KNN	0.733158	1973552.240273	783.503357	0.325711	0.733158

Evaluation Results

2- Crop Yield Model

```
# Dataframe consisting of metrics of all the models
result_df = pd.DataFrame(results, columns = ['Model', 'Accuracy', 'MSE', 'MAE', 'MAPE', 'R2_score'])
# Add red and green highlights in the dataframe to display best and worst performing models
result_format_df = result_df.style.highlight_max(subset = ['Accuracy','R2_score'], color = 'green').highlight_
display(result_format_df)
```

✓ 0.0s Python

	Model	Accuracy	MSE	MAE	MAPE	R2_score
0	Linear Regression	0.755142	1776116996.515923	29582.494556	0.875027	0.755142
1	Decision Tree	0.979028	152126879.690211	3674.954505	0.073275	0.979028
2	Random Forest	0.987491	90736304.678697	3472.209294	0.075039	0.987491
3	Gradient Boost	0.873241	919466056.192313	19396.278598	0.529985	0.873241
4	XGBoost	0.975929	174599934.341565	7720.576922	0.213953	0.975929
5	Bagging Regressor	0.987528	90464333.251987	3469.872735	0.074980	0.987528
6	KNN	0.332020	4845307069.320509	48062.047654	1.533120	0.332020

Utilizing K-Fold cross-validation, the Bagging Regressor model still remains the best model. The KNN still remains the worst model. There is no significant change in any of the models' accuracy when using K-Fold cross-validation

Evaluation Results

3- Undernourishment Model

```
results_format_df = results_df.style.highlight_max(subse
display(results_format_df)
```

0.0s

	Model	Accuracy	MSE	R2_score
0	Linear Regression	0.940822	7.972303	0.940822
1	Decision Tree	0.976387	3.181061	0.976387
2	Random Forest	0.986245	1.853085	0.986245
3	Gradient Boost	0.972070	3.762675	0.972070
4	XGBoost	0.990710	1.251478	0.990710
5	Bagging Regressor	0.986120	1.869931	0.986120
6	KNN	0.950578	6.657921	0.950578
7	Support Vector Regressor	0.788836	28.447365	0.788836
8	Elastic Net	0.756269	32.834661	0.756269
9	Ridge Regression	0.940055	8.075538	0.940055
10	Lasso Regression	0.769577	31.041818	0.769577
11	CatBoost	0.991754	1.110823	0.991754
12	Stacking Regressor	0.987569	1.674715	0.987569
13	Neural Network	0.881535	15.959217	0.881535
14	Hist Gradient Boosting	0.980250	2.660659	0.980250

Evaluation Results

```
...
Overall Metrics: MAE: 2.1991, MSE: 15.7607, RMSE: 3.9700, R²: 0.8606
Metrics for 2023: MAE: 1.7594, MSE: 10.5307, RMSE: 3.2451, R²: 0.9021
Metrics for 2024: MAE: 2.2170, MSE: 15.0295, RMSE: 3.8768, R²: 0.8670
Metrics for 2025: MAE: 2.6208, MSE: 21.7218, RMSE: 4.6607, R²: 0.8168
```

Deployment

The link of video : https://youtu.be/A2WP0CEhDWc

Future Work

- Develop models predicting crop yield impacts under different climate scenarios.
- Create decision support systems for farmers based on ML analysis of local conditions.
- Study the Global Hunger Index (GHI): Examine the relationship between key climate variables and the Global Hunger Index to understand their combined effect on global hunger levels.

References

- Climate Change — As You Sow. (n.d.). Retrieved September 25, 2024, from https://www.asyousow.org/our-work/climate-and-energy/climate-change
- Parry ML, Rosenzweig C, Iglesias A, Livermore M, Fischer G. Effects of climate change on global food production under SRES emissions and socio-economic scenarios. Global Environ Chang. 2004;14(1):53–67.
- Khan F, Khan F, Liou YA, Khan F, Spöck G, Wang X, et al. Assessing the impacts of temperature extremes on agriculture yield and projecting future extremes using machine learning and deep learning approaches with CMIP6 data. Int J Appl Earth Obs Geoinf. 2024;132(300).
- Xiong R, Peng H, Chen X, Shuai C. Machine learning-enhanced evaluation of food security across 169 economies. Environ Dev Sustain [Internet]. 2024;(0123456789). Available from: https://doi.org/10.1007/s10668-024-05212-1

- Tubiello FN. System of Environmental-Economic Accounting for Agriculture, Forestry and Fisheries
- Food Security Indicators [Presentation]. 2016;(June):24–6. Available from: https://seea.un.org/sites/seea.un.org/files/food_security_indicators.pdf
- WAITRO SDG2 Workgroup Webinar- Harvesting Global Solutions: Uniting for Zero Hunger Research Collaborations and Funding Opportunities - WAITRO. (n.d.). Retrieved October 1, 2024, from https://waitro.org/event/waitro-sdg2-workgroup-webinar-harvesting-global-solutions-uniting-for-zero-hunger-research-collaborations-and-funding-opportunities/
- Artificial Intelligence and Global Challenges — Zero Hunger | by DAIA | DAIA | Medium. (n.d.). Retrieved October 1, 2024, from https://medium.com/daia/artificial-intelligence-and-global-challenges-a-plan-for-progress-603efece1905
- Climate Change • Integrated Water Resource Management - from traditional knowledge to modern techniques • Department of Earth Sciences. (n.d.). Retrieved October 1, 2024, from https://www.geo.fu-berlin.de/en/v/iwrm/Implementation/water_and_the_physical_environment/Climate-and-Climate-Change/climate_change/index.html

Thank you!

frontier tech leaders