Prediction of CO2 Emissions from Country-Specific Data

A Machine Learning Project by Anshuman Mandal

Stage 1: Data Cleaning and Preparation

Notebook Contents

0. Introduction

Project Summary

The aim of this project is to analyze country-specific data and develop machine learning models to predict CO2 emissions from various country parameters. The project uses the publicly available dataset, *Climate Change Data* from the World Bank Group, which provides data on a vast majority of countries over a range of years, covering parameters such as:

- **Country**: The vast majority of countries worldwide
- **Year**: Ranging from 1990 to 2011
- **Greenhouse Gas Emissions**: CO2, CH4, N2O, others
- **Population-Specific Parameters**: Population count, urban population, population growth, etc.
- **Country Economic Indicators**: GDP, GNI, Foreign Direct Investment, etc.
- **Land-Related Parameters**: Cereal yield, agricultural land, nationally terrestrial protected areas, etc.
- **Climate Data**: Precipitation, national disasters, etc.
- **Energy Use**
- **Counts of Certain Types of Medical Personnel**

Project Stages

The project is divided into three stages:

1. **Data Cleaning and Preparation**
2. **Data Exploration and Visualization**
3. **Predictive Analysis with the Random Forest Machine Learning Algorithm**

Each stage is described in a separate Jupyter Notebook (`.ipynb` file) and a derived PDF file.

Notebook Summary - Stage 1: Data Cleaning and Preparation
The purpose of this notebook is to explain the first stage of the project: the cleaning and transformation of the available data to prepare it for visualization and analysis (described in subsequent notebooks).

- **Input**: Excel data file from the original online source.
- **Output**: Comma-separated values (CSV) file containing the cleaned data, ready for visualization and analysis.
- **Programming Language**: Python 3.7.
- **Libraries Used**: `pandas`, `numpy`.

Data Source

The data used in this project comes from the *Climate Change Data* of the World Bank Group, which provides country-specific data on various parameters such as CO2 emissions, energy use, population count, urban population, cereal yield, nationally terrestrial protected areas, GDP, GNI, etc.

- **Dataset**: Publicly available at World Bank Climate Change Data (https://datacatalog.worldbank.org/dataset/climate-change-data).
- **License**: Creative Commons Attribution 4.0 International license.

This analysis provides a global overview of the imported Climate Change dataset. Here's a breakdown of the key findings and the code used to analyze the data:

1. Notebook Setup

We start by importing the necessary libraries and loading the dataset:

```
import pandas as pd
import numpy as np

# Define the file name and the data sheet
orig_data_file = r"climate_change_download_0.xls"
data_sheet = "Data"

# Read the data from the Excel file into a pandas DataFrame
data_orig = pd.read_excel(io=orig_data_file, sheet_name=data_sheet)
```

2. Global Data Overview

2.1 Shape and Structure of the Dataset

The dataset consists of 13,512 rows and 28 columns:

```
print("Shape of the original dataset:")
print(data_orig.shape)
```

2.2 Available Columns

The dataset includes various columns representing country codes, names, series codes, and values across different years:

```
print("Available columns:")
print(data_orig.columns)
```

2.3 Data Types of Columns

Most of the columns have `object` data types, indicating they are stored as strings:

```
print("Column data types:")
print(data_orig.dtypes)
```

2.4 Preview of the Data

A preview of the first 5 rows gives an idea of the data structure:

```
print("Overview of the first 5 rows:")
print(data_orig.head())
```

2.5 Descriptive Statistics

Descriptive statistics provide insights into the number of unique values, most frequent values, and more:

```
print("Descriptive statistics of the columns:")
print(data_orig.describe())
```

3. Detailed Column Analysis

3.1 Series Name

The 'Series name' column provides descriptions of various indicators included in the dataset:

```
print("Contents of the column 'Series name':")
print(data_orig['Series name'].unique())
```

3.2 Series Code

The 'Series code' column includes unique codes corresponding to each indicator:

```
print("Contents of the column 'Series code':")
print(data_orig['Series code'].unique())
```

3.3 Scale and Decimals

The 'SCALE' and 'Decimals' columns seem to contain some values marked as 'Text'. These need to be further examined:

```
print("Contents of the column 'SCALE':")
print(data_orig['SCALE'].unique())

print("Contents of the column 'Decimals':")
print(data_orig['Decimals'].unique())
```

3.4 Identifying Rows with 'Text' in 'SCALE' and 'Decimals'

We identify the rows where 'SCALE' and 'Decimals' columns are marked as 'Text':

```
# Rows where 'SCALE' is 'Text'
print(data_orig[data_orig['SCALE']=='Text'])

# Rows where 'Decimals' is 'Text'
print(data_orig[data_orig['Decimals']=='Text'])
```

Findings from the Global Overview

This global overview provides key insights about the available data:

- **Shape**: The dataset contains 28 columns and 13,512 rows.
- **Data Types**: All columns are of type "object," meaning they are neither purely numeric nor text values.
- **Missing Values**: The dataset has a significant amount of missing values, denoted as both `NaN` (Not a Number) and the string `".."`.
- **'Text' Rows**: Rows marked as 'Text' in the columns 'SCALE' and 'Decimals' do not contain any useful information and are almost entirely composed of `NaN` values.
- **Columns**:
  - The columns represent key values such as country, corresponding years, and series codes/names.
  - The columns 'Country name', 'Series code', 'SCALE', and 'Decimals' do not provide meaningful information and are therefore considered obsolete.
  - The column 'Series name' contains the country-specific features necessary for analysis.
  - The names of the features in the 'Series name' column are clear but too lengthy for practical use in code.

3. Define the Initial Project Goals

The first overview of the raw data allows us to define the initial goals and objectives of the machine learning project. These goals will be refined as more insights are gained from the data. However, this initial definition will help develop a strategy for data cleaning, transformation, and visualization.

Summary of Available Data: The dataset can be categorized into the following country-specific parameters/features:

- Various emissions of greenhouse gases such as CO2, CH4, N2O, etc.

- Population-specific parameters: population count, urban population, population growth, etc.
- Economic indicators: GDP, GNI, Foreign Direct Investment, etc.
- Land-related parameters: cereal yield, agricultural land, nationally protected areas, etc.
- Climate data: precipitation, natural disasters, etc.
- Energy use
- Counts of certain types of medical personnel
- And more...

Initial Goal: The primary objective of this machine learning project is to analyze the relationships among these variable categories and evaluate the influence of factors such as the country's economy, energy use, land use, etc., on greenhouse gas emissions, precipitation, and other climate-related data. The final aim is to develop a machine learning model capable of predicting climate-related data or emissions based on other country-specific parameters.

As the project progresses and more insights are gained, these goals will be refined in more detail.

4. Data Cleaning

Organization of the Data Cleaning and Transformation

The main goal of data cleaning and transformation is to represent the features (country parameters from the 'Series name' column) as separate columns and to make each row identifiable by a country and a year. Additionally, it is necessary to transform the years into a single column.

Key tasks for data cleaning:

1. **Remove rows marked as "Text"** in the "SCALE" and "Decimals" columns.
2. **Remove unnecessary columns**: "Country name", "Series code", "SCALE", "Decimals".
3. **Transform missing values**: Replace `".."` and empty cells (`""`) with `NaN` values for easier recognition as missing data.
4. **Convert data columns to numeric**: Ensure that all data columns are transformed into a numerical data type.
5. **Rename features in 'Series name'**: Shorten the feature names in the 'Series name' column for practicality.

4.1 Removing Rows Marked as "Text" in the "SCALE" and "Decimals" Columns

```
# Assign the data to a new DataFrame, which will be modified
data_clean = data_orig

print("Original number of rows:")
print(data_clean.shape[0])

# Remove rows characterized as "Text" in the SCALE column
data_clean = data_clean[data_clean['SCALE'] != 'Text']

print("Current number of rows:")
print(data_clean.shape[0])
```

Output:

- Original number of rows: 13,512
- Current number of rows: 10,017

4.2 Removing Unnecessary Columns: "Country name", "Series code", "SCALE", "Decimals"

```
print("Original number of columns:")
print(data_clean.shape[1])

data_clean = data_clean.drop(['Country name', 'Series code', 'SCALE', 'Decimals'], axis='columns')

print("Current number of columns:")
print(data_clean.shape[1])
```

Output:

- Original number of columns: 28
- Current number of columns: 24

4.3 Transform Missing Values

```
data_clean.iloc[:, 2:] = data_clean.iloc[:, 2:].replace({'': np.nan, '..': np.nan})
```

4.4 Convert All Data Columns to Numeric Type

```
data_clean2 = data_clean.applymap(lambda x: pd.to_numeric(x, errors='ignore'))
# Errors are ignored to avoid issues with the first two columns, which don't need numeric transformation

print("Print the column data types after transformation:")
data_clean2.dtypes
```

Output:

```
Country code     object
Series name      object
1990            float64
1991            float64
1992            float64
...
2010            float64
2011            float64
dtype: object
```

4.5 Rename Features in 'Series name'

The variable names in the 'Series name' column are clear but too long and not practical to use in code. To improve usability, the most relevant feature names are renamed with shorter labels as indicated in the table below:

```
chosen_vars = {
    'Cereal yield (kg per hectare)': 'cereal_yield',
    'Foreign direct investment, net inflows (% of GDP)': 'fdi_perc_gdp',
    'Access to electricity (% of total population)': 'elec_access_perc',
    'Energy use per units of GDP (kg oil eq./$1,000 of 2005 PPP $)': 'en_per_gdp',
    'Energy use per capita (kilograms of oil equivalent)': 'en_per_cap',
    'CO2 emissions, total (KtCO2)': 'co2_ttl',
    'CO2 emissions per capita (metric tons)': 'co2_per_cap',
    'CO2 emissions per units of GDP (kg/$1,000 of 2005 PPP $)': 'co2_per_gdp',
    'Other GHG emissions, total (KtCO2e)': 'other_ghg_ttl',
    'Methane (CH4) emissions, total (KtCO2e)': 'ch4_ttl',
    'Nitrous oxide (N2O) emissions, total (KtCO2e)': 'n2o_ttl',
    'Droughts, floods, extreme temps (% pop. avg. 1990-2009)': 'nat_emerg',
    'Population in urban agglomerations >1million (%)': 'pop_urb_aggl_perc',
    'Nationally terrestrial protected areas (% of total land area)': 'prot_area_perc',
    'GDP ($)': 'gdp',
    'GNI per capita (Atlas $)': 'gni_per_cap',
    'Under-five mortality rate (per 1,000)': 'under_5_mort_rate',
    'Population growth (annual %)': 'pop_growth_perc',
    'Population': 'pop',
    'Urban population growth (annual %)': 'urb_pop_growth_perc',
    'Urban population': 'urb_pop'
}

# Rename all variables in the column "Series name" with shorter versions
data_clean2['Series name'] = data_clean2['Series name'].replace(to_replace=chosen_vars)
```

5. Data Frame Transformation

Current Layout of the Data Frame:

```
data_clean2.head()
```

Output:

```
Country code  Series name  1990  1991  1992  1993  1994  1995  1996  1997  ...  2002  2003  2004  2005  2006  2007  2008  2009  2010
0          ABW  Land area below 5m (% of land area)  29.574810    NaN    NaN    NaN    NaN    NaN    NaN    NaN ...    NaN    NaN
1          ADO  Land area below 5m (% of land area)   0.000000    NaN    NaN    NaN    NaN    NaN    NaN    NaN ...    NaN    NaN
2          AFG  Land area below 5m (% of land area)   0.000000    NaN    NaN    NaN    NaN    NaN    NaN    NaN ...    NaN    NaN
3          AGO  Land area below 5m (% of land area)   0.208235    NaN    NaN    NaN    NaN    NaN    NaN    NaN ...    NaN    NaN
4          ALB

 Land area below 5m (% of land area)   0.479237    NaN    NaN    NaN    NaN    NaN    NaN    NaN ...    NaN    NaN    NaN    NaN
```

This data frame layout is highly impractical, given that each row represents a feature and each column represents a year. To prepare this data for modeling, it is crucial to transpose the data so that each row represents a country and a year, and each column represents one of the features.

This task involves transforming the data from "wide" format into a "long" format by gathering all year values into a single column.

6. Data Cleaning: Handling Missing Values

6.1 Filtering Countries with Excessive Missing Values

To ensure the dataset remains manageable and meaningful, countries with more than 90 missing values were removed. This step significantly reduces the volume of missing data.

```
print("number of missing values in the whole dataset before filtering the countries:")
print(all_vars_clean.isnull().sum().sum())
print("number of rows before filtering the countries:")
print(all_vars_clean.shape[0])

# Filter countries with fewer than 90 missing values
countries_filter = []
for key, val in countries_missing_sorted.items():
    if val < 90:
        countries_filter.append(key)

all_vars_clean = all_vars_clean[all_vars_clean['country'].isin(countries_filter)]

print("number of missing values in the whole dataset after filtering the countries:")
print(all_vars_clean.isnull().sum().sum())
print("number of rows after filtering the countries:")
print(all_vars_clean.shape[0])
```

Output:

- Number of missing values in the whole dataset before filtering the countries: 29,818
- Number of rows before filtering the countries: 4,194
- Number of missing values in the whole dataset after filtering the countries: 7,854
- Number of rows after filtering the countries: 1,728

6.2 Checking Features (Columns) for Missing Values

After filtering out countries, the remaining dataset still had some columns with a significant number of missing values. Columns with more than 20 missing values were removed to avoid excessive data loss.

```
# Count of NaN values per column
print("NaN values count in each column:")
print(all_vars_clean.isnull().sum())
```

Output:

```
country                  0
year                     0
cereal_yield            10
fdi_perc_gdp            17
elec_access_perc      1728
en_per_gdp               0
en_per_cap               0
co2_ttl                  9
co2_per_cap              9
co2_per_gdp              9
other_ghg_ttl         1446
ch4_ttl               1440
n2o_ttl               1440
nat_emerg             1728
pop_urb_aggl_perc        0
prot_area_perc           0
gdp                      2
gni_per_cap             16
under_5_mort_rate        0
pop_growth_perc          0
pop                      0
urb_pop_growth_perc      0
urb_pop                  0
dtype: int64
```

Decision: Columns with more than 20 missing values were removed to ensure that the dataset is usable without significant loss of observations.

```
from itertools import compress

# Boolean mapping for features with more than 20 missing values
vars_bad = all_vars_clean.isnull().sum() > 20

# Remove columns with excessive missing values
all_vars_clean2 = all_vars_clean.drop(compress(data=all_vars_clean.columns, selectors=vars_bad), axis='columns')

print("Remaining missing values per column:")
print(all_vars_clean2.isnull().sum())
```

Output:

```
Remaining missing values per column:
country                0
year                   0
cereal_yield          10
fdi_perc_gdp          17
en_per_gdp             0
en_per_cap             0
co2_ttl                9
co2_per_cap            9
co2_per_gdp            9
pop_urb_aggl_perc      0
prot_area_perc         0
gdp                    2
gni_per_cap           16
under_5_mort_rate      0
pop_growth_perc        0
pop                    0
urb_pop_growth_perc    0
urb_pop                0
dtype: int64
```

6.3 Removing Remaining Missing Values

Since removing rows with remaining missing values does not drastically impact the dataset's size, these rows were removed.

```
# Delete rows with any number of missing values
all_vars_clean3 = all_vars_clean2.dropna(axis='rows', how='any')

print("Remaining missing values per column:")
print(all_vars_clean3.isnull().sum())

print("Final shape of the cleaned dataset:")
print(all_vars_clean3.shape)
```

Output:

```
Remaining missing values per column:
country               0
year                  0
cereal_yield          0
fdi_perc_gdp          0
en_per_gdp            0
en_per_cap            0
co2_ttl               0
co2_per_cap           0
co2_per_gdp           0
pop_urb_aggl_perc     0
prot_area_perc        0
gdp                   0
gni_per_cap           0
under_5_mort_rate     0
pop_growth_perc       0
pop                   0
urb_pop_growth_perc   0
urb_pop               0
dtype: int64

Final shape of the cleaned dataset:
(1,700, 18)
```

7. Export of the Cleaned Data Frame

The cleaned dataset is now ready for further analysis and can be exported to a CSV file for use in subsequent stages of the project.

```
# Export the cleaned DataFrame to a CSV file
all_vars_clean3.to_csv('data_cleaned.csv', index=False)
```

Next Steps: The subsequent stages of the project include Data Visualization and Predictive Analysis, which will be detailed in the corresponding notebooks.