**GHG Emission Prediction Project**

# **Day 2 -** Understanding Step 2 Code for Week 01

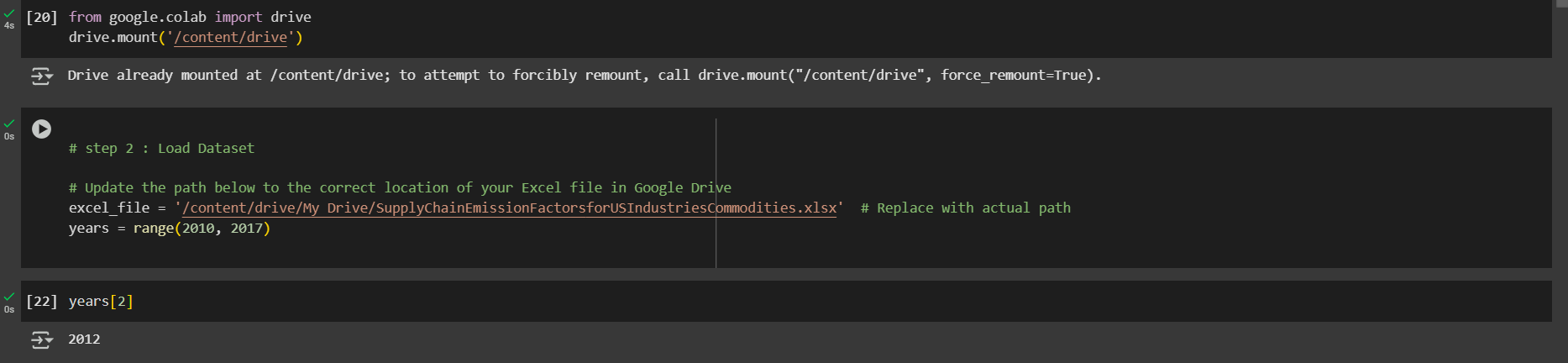
# **Date -** 19 June 2025 (Week 1)

# **Internship Name** - Edunet-Shell Skills4Future AICTE Internship

# **Intern Name:** Abhinay Singh

# **Step 2: Load Dataset**

**Code Screenshot**



✅ **Step 1: Mount Google Drive**

**from google.colab import drive**

**drive.mount('/content/drive')**

* This allows Google Colab to access files from your Google Drive.
* We must approve permission (first time only).
* Our Excel file is saved inside Drive, so mounting it lets you read it from Colab.

✅ **Step 2: Set the File Path**

**# step 2 : Load Dataset**

**# Update the path below to the correct location of your Excel file in Google Drive**

**excel\_file = '/content/drive/My Drive/SupplyChainEmissionFactorsforUSIndustriesCommodities.xlsx' # Replace with actual path**

**years = range(2010, 2017)**

* We're saving the full path to your .xlsx Excel file in a variable called excel\_file.
* This file contains multiple sheets, one for each year (2010–2016) and data type (Commodity/Industry).
* This creates a list-like object: [2010, 2011, 2012, 2013, 2014, 2015, 2016]
* We’ll use it to loop through or refer to each year dynamically when loading sheets.

✅ **Step 3: Checking the 3rd Element**

**years[2]**

* This gives the **third element** of the year's range.
* Output: 2012 — useful to double-check indexing when loading specific year sheets.

**✅ Step 4: Load 2010 Commodity Sheet**

**df\_1 = pd.read\_excel(excel\_file, sheet\_name=f'{years[0]}\_Detail\_Commodity')**

**df\_1.head()**

* years[0] = 2010, so you're loading the sheet named "2010\_Detail\_Commodity".
* pd.read\_excel() reads that sheet from the Excel file into a DataFrame df\_1.
* .head() displays the first 5 rows to preview the data.

**✅ Step 5: Load 2010 Industry Sheet**

**df\_2 = pd.read\_excel(excel\_file, sheet\_name=f'{years[0]}\_Detail\_Industry')**

**df\_2.head()**

#### **What It Does:**

* years[0] = 2010 (because of range(2010, 2017))
* So f'{years[0]}\_Detail\_Industry' becomes "2010\_Detail\_Industry**"**

**We're telling Python to:**

* 📄 Load the sheet named "2010\_Detail\_Industry" from the Excel file.
* 🧮 Store it as a DataFrame in df\_2.
* 👀 Preview the first 5 rows using .head().

**Quick Comparison**

| **Data Frame** | **Sheet Loaded** | **Data Type** |
| --- | --- | --- |
| df\_1 | “2010\_detail\_Commodity” | Commodity- Based |
| df\_2 | “2010\_detail\_Industry” | Industry-Based |

**🧠 Summary of What We’ve Achieved:**

* Mounted our Drive.
* Linked to the Excel file containing multiple years’ worth of commodity data.
* Dynamically accessed one of the year’s sheets (2010) for processing.

**We now have:**

* df\_1: GHG emission data grouped by **commodities.**
* df\_2: GHG emission data grouped by **industries.**

**🎓 In Our Project**

**df\_1 = pd.read\_excel(excel\_file, sheet\_name=f'{years[0]}\_Detail\_Commodity')**

**df\_1.head()**

* Now df\_1 holds a table (DataFrame) of commodity-wise emissions for 2010.
* We can **analyze**, **clean**, and **use** this data to train our ML model.

**✅ Step 6: Load Commodity + Industry sheets for each year (2010–2016)**

**all\_data = []**

**for year in years:**

**try:**

**df\_com = pd.read\_excel(excel\_file, sheet\_name=f'{year}\_Detail\_Commodity')**

**df\_ind = pd.read\_excel(excel\_file, sheet\_name=f'{year}\_Detail\_Industry')**

**df\_com['Source'] = 'Commodity'**

**df\_ind['Source'] = 'Industry'**

**df\_com['Year'] = df\_ind['Year'] = year**

**df\_com.columns = df\_com.columns.str.strip()**

**df\_ind.columns = df\_ind.columns.str.strip()**

**df\_com.rename(columns={**

**'Commodity Code': 'Code',**

**'Commodity Name': 'Name'**

**}, inplace=True)**

**df\_ind.rename(columns={**

**'Industry Code': 'Code',**

**'Industry Name': 'Name'**

**}, inplace=True)**

**all\_data.append(pd.concat([df\_com, df\_ind], ignore\_index=True))**

**except Exception as e:**

**print(f"Error processing year {year}: {e}")**

**🧾 Full Code Breakdown:**

**all\_data = []**

* We’re creating an empty list to store cleaned data from each year.

for year in years:

try:

* We're looping through each year from the years range (2010–2016).
* try: is used to **handle errors** if any file or sheet is missing.

**📥 Read Excel Sheets**

**df\_com = pd.read\_excel(excel\_file, sheet\_name=f'{year}\_Detail\_Commodity')**

**df\_ind = pd.read\_excel(excel\_file, sheet\_name=f'{year}\_Detail\_Industry')**

* Reads the Commodity and Industry data sheets for that specific year.
* Example: 2010\_Detail\_Commodity, 2010\_Detail\_Industry.

**🧷 Add Labels to Identify Source & Year**

**🧷 Add Labels to Identify Source & Year**

**df\_com['Source'] = 'Commodity'**

**df\_ind['Source'] = 'Industry'**

**df\_com['Year'] = df\_ind['Year'] = year**

**Adds two columns:**

* Source: to tag where the data came from
* Year: so later you can filter/compare across years

**🧹 Strip Whitespaces from Column Names**

**df\_com.columns = df\_com.columns.str.strip()**

**df\_ind.columns = df\_ind.columns.str.strip()**

* Removes leading/trailing spaces in column names, like " Commodity Code " ➝ "Commodity Code"
* This helps avoid future column name errors.

**🔠 Rename Columns for Uniformity**

**df\_com.rename(columns={**

**'Commodity Code': 'Code',**

**'Commodity Name': 'Name'**

**}, inplace=True)**

**df\_ind.rename(columns={**

**'Industry Code': 'Code',**

**'Industry Name': 'Name'**

**}, inplace=True)**

* Makes the structure of commodity and industry tables the same.
* Both will now have columns: Code, Name, Year, Source, GHG Emissions, etc.

**🔗 Concatenate & Store in List**

**all\_data.append(pd.concat([df\_com, df\_ind], ignore\_index=True))**

* pd.concat() combines both DataFrames (commodity + industry) for that year.
* ignore\_index=True resets the index.
* Appends the combined yearly data into all\_data list.

**⚠️ Handle Errors Gracefully**

**except Exception as e:**

**print(f"Error processing year {year}: {e}")**

* If any year’s sheet is missing or fails to load, it won’t crash the whole program.
* It will print a message like:  
   "Error processing year 2013: Sheet not found**"**

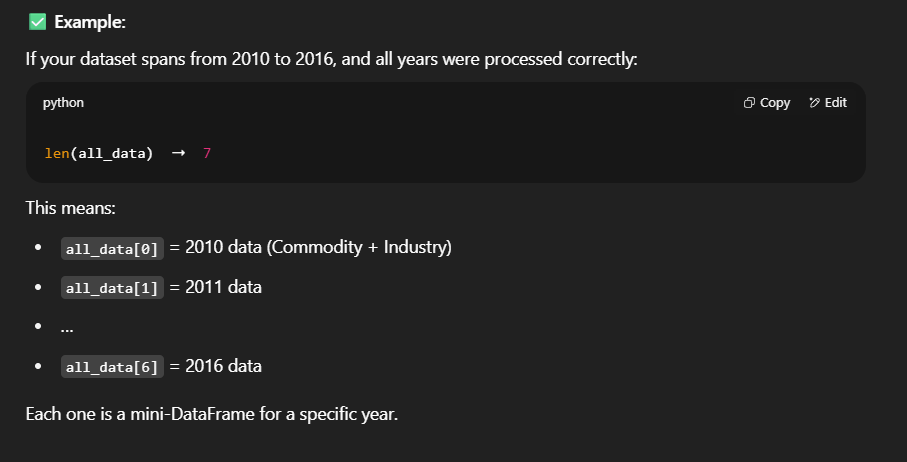
### **🔍 We Checked Output:**

all\_data[3]

* This shows the **4th element** in all\_data (i.e., data for year = 2013).

len(all\_data)

* This gives the **number of DataFrames** (i.e., year-wise combined data) stored in our all\_data list.



**df = pd.concat(all\_data, ignore\_index=True)**

**df = pd.concat(all\_data, ignore\_index=True)**

**df.head(10)**

* This combines all year-wise DataFrames into one big DataFrame called df.
* pd.concat(all\_data) = merge 7 small tables into 1
* ignore\_index=True = reset index from 0, 1, 2... up to total rows

**🎯 Result:**

Now, df contains:

* All rows from 2010–2016
* Both Industry and Commodity data
* Columns like: Code, Name, GHG\_Emissions\_kgCO2e, Source, Year, etc.
* This shows the **first 10 rows** of the combined dataset.

**Helpful to:**

* **Preview the structure**
* See if column names and values look clean
* Confirm data merge was successful

len(df)

* This gives the **total number of rows** in our final dataset

**Now your df is ready to:**

* Clean 🧹
* Analyze 📊
* Train model 🧠

**✅ Step 3: Data Preprocessing**

**#Step 3: Data Preprocessing**

**df.columns # Checking columns**

* This line displays **all the column names** of your df DataFrame.

### **Purpose:**

* To **understand** what features (inputs) you have in your dataset.
* To **identify columns** that may need:
  + Renaming for clarity.
  + Encoding if categorical.
  + Dropping if irrelevant.
  + Scaling if numeric and needed for models.

**Output :**

**Index(['Code', 'Name', 'Substance', 'Unit',**

**'Supply Chain Emission Factors without Margins',**

**'Margins of Supply Chain Emission Factors',**

**'Supply Chain Emission Factors with Margins', 'Unnamed: 7',**

**'DQ ReliabilityScore of Factors without Margins',**

**'DQ TemporalCorrelation of Factors without Margins',**

**'DQ GeographicalCorrelation of Factors without Margins',**

**'DQ TechnologicalCorrelation of Factors without Margins',**

**'DQ DataCollection of Factors without Margins', 'Source', 'Year'],**

**dtype='object'**

**df.isnull().sum()**

* This code checks for **null/missing values** in each column and shows how many are missing.

### **Purpose:**

* To **identify missing data** (NaN = Not a Number).
* Helps decide whether to:
  + Fill missing values (e.g., mean, median).
  + Drop columns or rows.
  + Treat them specially in models.

**Output:**

