National College of Ireland

Configuration Manual

MSc Research Project
Programme Name

Lavanya Budhraja
Student ID: 23300540

School of Computing
National College of Ireland

Supervisor: Charlyn Rosales

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Lavanya Budhraja
Student ID:	23300540
Programme:	MSc in Artificial Intelligence for Business **Year:** 2025
Module:	Practicum
Lecturer:	Charlyn Rosales
Submission Due Date:	28 January 2026
Project Title:	A Data-Driven Framework for Estimating Hard-to-Measure Value-Chain Emissions
Word Count:	807 **Page Count:**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Lavanya Budhraja

Date: 28 January 2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Lavanya Budhraja
Student ID: 23300540

1 Introduction

The Configuration Manual's aim is to provide documentation regarding the hardware, software, and operational needs required for the "A Data-Driven Machine Learning Framework for Scope 3 Emission Estimation" research project reproduction. This writing is intended as a manual for the examiners and researchers as to the configuration of the environment, execution of the code artifacts, and the reproduction of the exact results, tables, and visualizations included in the final thesis report. It connects the theoretical methodology with practical implementation.

2 System requirements

The study was conducted and verified in a cloud environment (Google Colab) with the purpose of scalability and reproducibility. But it can still run on a local PC. The specs provided below indicate the best setup for the highest performance and productivity.

2.1 Hardware requirements

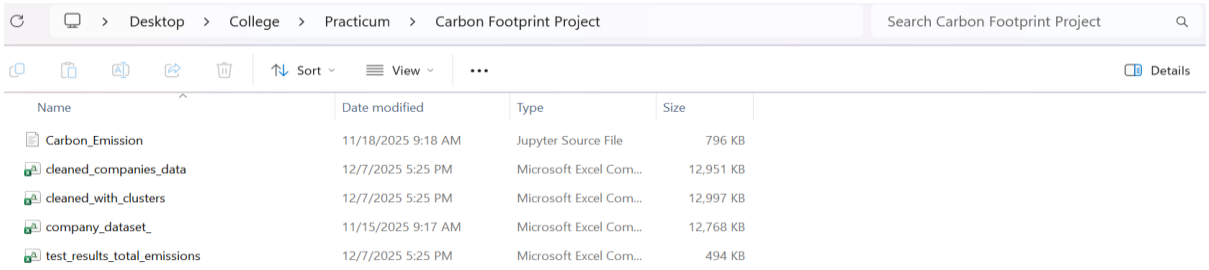
- Processor: Intel Core i5/i7 (8th Gen or newer) or Apple M1/M2 silicon.
- RAM: Minimum 8 GB disk space (16 GB is the optimal for quick model training).
- Storage: At least 500 MB of free disk space to accommodate dataset and libraries.
- Internet Connection: Needed for the installation of Python packages and usage of Google Colab.

2.2 Software requirements

- Operating System: Platform independent (Windows 10/11, macOS, Linux).
- Runtime Environment: Python 3.10.12 (or higher).
- IDE: Google Colab (Recommended) or Jupyter Notebook/Lab (Local).
- Browser: Google Chrome or Mozilla Firefox (for Colab compatibility).

3 Environmental setup

To ensure the file paths in the code function correctly, the project directory is organised as follows:



3.1 Library dependencies

The implementation is based on the Python scientific stack. There were specific versions used in this research that are critical for reproducing the exact SHAP values and model metrics.

Required Libraries:

- numpy & pandas: For data manipulation.
- matplotlib & seaborn: For data visualization.
- scikit-learn: For preprocessing pipelines, K-Means, and Random Forest.
- xgboost: For the Gradient Boosting regressor.
- shap: For model explainability and feature importance.

4 Implementation steps

This section describes the sequential execution of the Carbon_Emission.ipynb notebook.

Step 1: Initialization and Installation

The first cell of the notebook installs the non-standard libraries (xgboost and shap) which are not included in the default Anaconda or Colab runtime.

- **Action:** Run Cell 1.

Expected Output: Installation logs confirming successful download

```
# Cell 1: Install extra libraries
!pip install xgboost shap -q
```

Step 2: Data Ingestion

The system loads raw firmographic data. Ensure the file company_dataset_.csv is uploaded to the runtime session or correctly linked in your Google Drive.

- **Action:** Run Cell 2 ("Load the dataset").
- **Verification:** The output must print the dataset shape: (23808, 59).

```
# Cell 2: Load the dataset

# OPTION A: If the CSV is already in the Colab working directory:
csv_path = "company_dataset_.csv"

# OPTION B: If you want to upload manually from your local machine:
# from google.colab import files
# uploaded = files.upload()
# csv_path = list(uploaded.keys())[0]

df = pd.read_csv(csv_path)

print("Shape:", df.shape)
df.head()
```

```
Shape: (23808, 59)
```

	company_id	industry	avg_short_haul_round_trip_per_year	avg_return_flights	avg_long_haul_round_trip_per_year	avg_train_journey	size_strata	naics_code	country	revenue_usd	year	office_area_ft2
0	624c9dee6b13380001bfc5aa	management consulting	5	5	4	1	small_<50	54	ZA	1882250.84	2020	3000
1	61dc2e5083c2a70001053c1f	airlines/aviation	264	220	352	77	large_500_9999	56	IN	60890250.60	2022	206250
2	61e10a093d7b150001cbd6e8	information technology & services	13	19	12	5	medium_50_499	5415	DE	5521526.35	2019	9375

Step 3: Exploratory data analysis (EDA)

The missing values and skewness of distribution are identified using statistical profiling in the notebook. This step justifies the need for custom imputation strategy described in the thesis.

- **Action:** Run Cells 3–5.
- **Expected Output:** A histogram visualization of missing values and a statistical summary table.

```
# Cell 3: Basic inspection

print("\n--- DataFrame Info ---")
df.info()

print("\n--- First 5 rows ---")
display(df.head())

print("\n--- Basic numeric summary ---")
display(df.describe().T)
```

```
--- DataFrame Info ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23808 entries, 0 to 23807
Data columns (total 59 columns):
 #   Column                              Non-Null Count  Dtype
---  ------                              --------------  -----
 0   company_id                          23808 non-null  object
 1   industry                            23808 non-null  object
 2   avg_short_haul_round_trip_per_year  23808 non-null  int64
 3   avg_return_flights                  23808 non-null  int64
 4   avg_long_haul_round_trip_per_year   23808 non-null  int64
 5   avg_train_journey                   23808 non-null  int64
 6   size_strata                         23808 non-null  object
 7   naics_code                          23808 non-null  object
 8   country                             23808 non-null  object
 9   revenue_usd                         23808 non-null  float64
 10  year                                23808 non-null  int64
 11  office_area_ft2                     23808 non-null  int64
 12  electricity_kwh_annual              23808 non-null  float64
 13  monthly_kwh_json                    23808 non-null  object
 14  heating_source                      23808 non-null  object
 15  natural_gas_m3_annual               23808 non-null  float64
 16  fleet_cars                          21767 non-null  float64
```

```
# Cell 3b: Basic data validation and sanity checks

df_validation = df.copy()

# 1) Non-negative checks for key numeric columns
non_negative_cols = [
    'employees',
    'electricity_kwh_annual',
    'natural_gas_m3_annual',
    'fleet_cars',
    'commuters_count',
    'waste_t',
    'scope1_emissions_t',
    'scope2_emissions_t',
    'scope3_emissions_t',
    'total_emissions_t'
]

violations = {}

for col in non_negative_cols:
    if col in df_validation.columns:
        count_neg = (df_validation[col] < 0).sum()
        violations[col] = int(count_neg)

print("Negative value violations (should ideally be 0):")
for col, count_neg in violations.items():
    print(f"{col}: {count_neg}")

# 2) Check for impossible zero employees
if 'employees' in df_validation.columns:
    zero_emp = (df_validation['employees'] <= 0).sum()
    print(f"\nNumber of companies with employees <= 0: {zero_emp}")
```

```python
# 3) Basic print for extreme total emissions (99th percentile)
if 'total_emissions_t' in df_validation.columns:
    p99 = df_validation['total_emissions_t'].quantile(0.99)
    print(f"\n99th percentile of total_emissions_t: {p99:,.2f}")
    extreme_count = (df_validation['total_emissions_t'] > p99).sum()
    print(f"Number of records above 99th percentile: {extreme_count}")
```

```
Negative value violations (should ideally be 0):
electricity_kwh_annual: 0
natural_gas_m3_annual: 0
fleet_cars: 0
commuters_count: 0
waste_t: 0
scope1_emissions_t: 0
scope2_emissions_t: 0
scope3_emissions_t: 0
total_emissions_t: 0

99th percentile of total_emissions_t: 3,976,160.98
Number of records above 99th percentile: 239
```

```python
# Cell 4: Define targets, leakage columns, and feature set

all_cols = df.columns.tolist()
print("Total columns:", len(all_cols))

# 1. Emission-related columns (TARGETS + subcomponents)
emission_cols = [
    'scope1_emissions_t',
    'scope2_emissions_t',
    'scope3_emissions_t',
    'total_emissions_t',
    'emissions_per_employee_t',
    'natural_gas_emissions_t',
    'natural_gas_emissions_t',
    'fleet_emissions_t',
    'employee_commuting_emissions_t',
    'waste_emissions_t',
    'purchased_goods_emissions_t',
    'upstream_transport_emissions_t',
    'leased_assets_emissions_t',
    'investments_emissions_t',
    'refrigerants_emissions_t',
    'capex_emissions_t',
    'audited_total_emissions_t',
]

# 2. Metadata / non-feature columns
meta_cols = [
    'company_id',
    'monthly_kwh_json',
    'natural_gas_kg_per_m3',
    'measurement_uncertainty_pct',
    'measurement_method',
    'is_audited',  # treat audit status as metadata, not as an operational driver
]

# 3. Targets for supervised learning
target_cols = ['scope1_emissions_t', 'scope2_emissions_t', 'scope3_emissions_t', 'total_emissions_t']

# Ensure targets are in emission_cols
for t in target_cols:
    assert t in emission_cols, f"Target {t} should be in emission_cols"

# 4. Define usable feature columns = everything except meta + emission columns
non_feature_cols = set(emission_cols + meta_cols)
feature_cols = [c for c in all_cols if c not in non_feature_cols]
```

```python
print("\n--- Target columns ---")
print(target_cols)

print("\n--- Columns excluded from features (leakage/metadata) ---")
print(sorted(non_feature_cols))

print("\n--- Final feature columns (no emissions, no IDs, no audit metadata) ---")
print(len(feature_cols), "features:")
print(feature_cols)
```

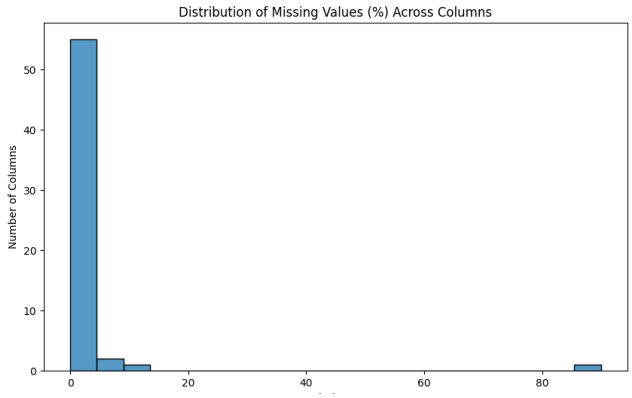
```
Total columns: 59

--- Target columns ---
['scope1_emissions_t', 'scope2_emissions_t', 'scope3_emissions_t', 'total_emissions_t']

--- Columns excluded from features (leakage/metadata) ---
['audited_total_emissions_t', 'capex_emissions_t', 'company_id', 'emissions_per_employee_t', 'employee_commuting_emissions_t', 'fleet_emissions_t', 'investments_emissions_t', 'is_audited', 'leased_assets_emissions_t', 'meas

--- Final feature columns (no emissions, no IDs, no audit metadata) ---
37 features:
['industry', 'avg_short_haul_round_trip_per_year', 'avg_return_flights', 'avg_long_haul_round_trip_per_year', 'avg_train_journey', 'size_strata', 'naics_code', 'country', 'revenue_usd', 'year', 'office_area_ft2', 'electrici
```

```python
# Cell 5: Missing values overview

missing_counts = df.isna().sum().sort_values(ascending=False)
missing_pct = (missing_counts / len(df) * 100).round(2)

missing_summary = pd.DataFrame({
    "missing_count": missing_counts,
    "missing_pct": missing_pct
})

print("=== Missing values summary (top 20) ===")
display(missing_summary.head(20))
```

```
=== Missing values summary (top 20) ===
```

	missing_count	missing_pct
audited_total_emissions_t	21415	89.95
total_procurement_spend_usd	2423	10.18
fleet_cars	2041	8.57
laptops	1383	5.81
phones	1048	4.40
avg_return_flights	0	0.00
size_strata	0	0.00
naics_code	0	0.00
avg_long_haul_round_trip_per_year	0	0.00
avg_train_journey	0	0.00
company_id	0	0.00

```
plt.figure(figsize=(10, 6))
sns.histplot(missing_pct, bins=20)
plt.title("Distribution of Missing Values (%) Across Columns")
plt.xlabel("Missing %")
plt.ylabel("Number of Columns")
plt.show()
```



Step 4: Domain-Aware Preprocessing

The present situation requires important bespoke implementation. A function impute_group_median has been defined in the code which takes care of filling in the missing data by grouping the Industry and Size strata, thus preserving the integrity of the data.

- **Action:** Run Cell 6.
- **Verification:** The console will print confirmations like Imputing total_procurement_spend_usd.

```
# Cell 6: Domain-aware pre-imputation for key numeric fields

df_clean = df.copy()

# Helper: grouped median imputation
def impute_group_median(df, col, group_cols):
    """
    Impute missing values in `col` using median within groups defined by `group_cols`.
    Falls back to global median if group median is missing.
    """
    global_median = df[col].median()
    group_medians = df.groupby(group_cols)[col].median()

    def fill_func(row):
        if pd.notna(row[col]):
            return row[col]
        key = tuple(row[g] for g in group_cols)
        if key in group_medians.index and not np.isnan(group_medians.loc[key]):
            return group_medians.loc[key]
        return global_median

    df[col] = df.apply(fill_func, axis=1)
    return df

# Columns we care about:
critical_numeric = [
    'total_procurement_spend_usd',
    'fleet_cars',
    'laptops',
    'phones'
]

group_cols = ['industry', 'size_strata']  # reasonable grouping

for col in critical_numeric:
    if col in df_clean.columns:
        print(f"Imputing {col} using group median by {group_cols}...")
        df_clean = impute_group_median(df_clean, col, group_cols)

# After this, remaining missing values will be handled by the sklearn Imputer in the pipeline.
missing_after = df_clean.isna().sum().sort_values(ascending=False).head(10)
print("\nMissing values after domain-aware imputation (top 10):")
print(missing_after)
```

```
Imputing total_procurement_spend_usd using group median by ['industry', 'size_strata']...
Imputing fleet_cars using group median by ['industry', 'size_strata']...
Imputing laptops using group median by ['industry', 'size_strata']...
Imputing phones using group median by ['industry', 'size_strata']...

Missing values after domain-aware imputation (top 10):
audited_total_emissions_t          21415
industry                               0
company_id                             0
avg_return_flights                     0
avg_long_haul_round_trip_per_year      0
avg_train_journey                      0
size_strata                            0
naics_code                             0
country                                0
revenue_usd                            0
dtype: int64
```

Step 5: Feature Engineering Pipeline

The ColumnTransformer is set up with the intention of using One-Hot Encoding for categoricals and Standard Scaling for numericals. This is done to prepare the matrix for K-Means algorithm.

- **Action:** Run Cell 9.
- **Verification:** Check that preprocessor.fit_transform executes without error.

```
# Cell 9: Preprocessing with ColumnTransformer (with Winsorization)

numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("winsor", Winsorizer(lower_quantile=0.05, upper_quantile=0.95)),
    ("scaler", StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore", sparse_output=False))
])

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
    ],
    remainder="drop"
)

# Test fit
X_train_sample = X_train.iloc[:100]
_ = preprocessor.fit_transform(X_train_sample)
print("Preprocessor test fit successful with winsorization.")
```

```
Preprocessor test fit successful with winsorization.
```

5 Model training & evaluation

5.1 Unsupervised segmentation (K-Means)

The system executes K-Means clustering (k=4) to segment the companies.

- **Action:** Run Cell 16.
- **Expected Output:** A breakdown of company counts per cluster (e.g., Cluster 2: 12,555 companies).
- **Relevance:** These cluster IDs are appended to the dataset for the final analysis.

```
# Cell 16: Final clustering and profile analysis

K_FINAL = 4  # you can tune this based on silhouette/inertia from Cell 15

# Final KMeans on all transformed features
kmeans_final = KMeans(n_clusters=K_FINAL, random_state=RANDOM_STATE, n_init=10)
cluster_labels = kmeans_final.fit_predict(X_all_transformed)

# Attach cluster labels to cleaned data
df_clusters = df_clean.copy()
df_clusters['cluster'] = cluster_labels

print("Cluster counts:")
print(df_clusters['cluster'].value_counts().sort_index())

# 1) Average emissions per cluster (targets)
cluster_profile = df_clusters.groupby('cluster')[
    ['scope1_emissions_t', 'scope2_emissions_t', 'scope3_emissions_t', 'total_emissions_t']
].mean().round(2)

print("\nAverage emissions per cluster (tCO2e):")
display(cluster_profile)

# 2) Numeric operational drivers per cluster
numeric_driver_cols = [
    'electricity_kwh_annual',
    'natural_gas_m3_annual',
    'fleet_cars',
    'total_procurement_spend_usd',
    'commuters_count',
    'waste_t',
]

print("\nCluster-wise summary of key numeric drivers (means):")
display(df_clusters.groupby('cluster')[numeric_driver_cols].mean().round(2))

# 3) Size-strata distribution per cluster (categorical, use crosstab)
print("\nCluster-wise size_strata distribution (row-normalized):")
size_strata_dist = pd.crosstab(df_clusters['cluster'], df_clusters['size_strata'], normalize='index').round(3)
display(size_strata_dist)

# 4) Country distribution per cluster (optional)
print("\nCluster-wise country distribution (row-normalized):")
country_dist = pd.crosstab(df_clusters['cluster'], df_clusters['country'], normalize='index').round(3)
display(country_dist)
```

```
Cluster counts:
cluster
0    8079
1    1481
2   12555
3    1693
Name: count, dtype: int64

Average emissions per cluster (tCO2e):
```

cluster	scope1_emissions_t	scope2_emissions_t	scope3_emissions_t	total_emissions_t
0	941.67	279.07	16030.93	17251.66
1	14633.19	6492.59	301462.40	322588.18
2	916.78	521.98	22878.35	24317.10
3	114822.37	51324.62	2298998.11	2465145.09

```
Cluster-wise summary of key numeric drivers (means):
```

cluster	electricity_kwh_annual	natural_gas_m3_annual	fleet_cars	total_procurement_spend_usd	commuters_count	waste_t
0	9.611875e+05	792.85	267.44	2.372976e+07	173.51	26.78
1	1.425015e+07	12330.28	4159.68	3.357534e+08	2657.89	346.72
2	9.413795e+05	764.98	260.06	2.285870e+07	168.29	25.99
3	1.104836e+08	95415.82	32415.52	2.500183e+09	20750.79	1914.18

```
Cluster-wise size_strata distribution (row-normalized):
```

cluster	enterprise_10000+	large_500_9999	medium_50_499	small_<50
0	0.000	0.242	0.424	0.334
1	0.131	0.869	0.000	0.000
2	0.000	0.238	0.429	0.333
3	0.999	0.001	0.000	0.000

```
Cluster-wise country distribution (row-normalized):
```

cluster	AU	BR	CN	DE	FR	IN	SG	UK	US	ZA
0	0.000	0.000	0.000	0.229	0.233	0.000	0.304	0.234	0.000	0.000
1	0.092	0.110	0.088	0.083	0.076	0.131	0.116	0.085	0.124	0.096
2	0.150	0.149	0.154	0.000	0.000	0.202	0.000	0.000	0.198	0.147
3	0.095	0.094	0.102	0.092	0.089	0.115	0.119	0.093	0.110	0.093

5.2 Supervised prediction (Ensemble Models)

The notebook trains the Random Forest and XGBoost models.

- **Action:** Run Cells 10 and 11.
- **Expected Output:** The console will print the evaluation metrics: **RMSE** and **R^2** (approx 0.52 for XGBoost).

```python
# Cell 10: Random Forest model for total_emissions_t

y_total = y_train['total_emissions_t']
y_total_test = y_test['total_emissions_t']

rf_total_pipeline = Pipeline(steps=[
    ("preprocess", preprocessor),
    ("model", RandomForestRegressor(
        n_estimators=200,
        max_depth=None,
        min_samples_split=4,
        min_samples_leaf=2,
        n_jobs=-1,
        random_state=RANDOM_STATE
    ))
])

# Fit model
rf_total_pipeline.fit(X_train, y_total)

# Predict
y_pred_total = rf_total_pipeline.predict(X_test)

# Metrics (fixed)
mse = mean_squared_error(y_total_test, y_pred_total)
rmse = np.sqrt(mse)   # Manual RMSE

mae = mean_absolute_error(y_total_test, y_pred_total)
r2 = r2_score(y_total_test, y_pred_total)

print("=== RandomForestRegressor: total_emissions_t ===")
print(f"RMSE: {rmse:,.2f}")
print(f"MAE : {mae:,.2f}")
print(f"R^2 : {r2:.4f}")
```

```
=== RandomForestRegressor: total_emissions_t ===
RMSE: 880,906.82
MAE : 95,257.72
R^2 : 0.4790
```

```python
# Cell 11: XGBoost model for total_emissions_t

y_total = y_train['total_emissions_t']
y_total_test = y_test['total_emissions_t']

xgb_total_pipeline = Pipeline(steps=[
    ("preprocess", preprocessor),
    ("model", xgb.XGBRegressor(
        n_estimators=300,
        learning_rate=0.05,
        max_depth=6,
        subsample=0.8,
        colsample_bytree=0.8,
        objective="reg:squarederror",
        random_state=RANDOM_STATE,
        n_jobs=-1
    ))
])

xgb_total_pipeline.fit(X_train, y_total)

y_pred_total_xgb = xgb_total_pipeline.predict(X_test)

# Metrics (no squared=False)
mse_xgb = mean_squared_error(y_total_test, y_pred_total_xgb)
rmse_xgb = np.sqrt(mse_xgb)
mae_xgb = mean_absolute_error(y_total_test, y_pred_total_xgb)
r2_xgb = r2_score(y_total_test, y_pred_total_xgb)

print("=== XGBoost: total_emissions_t ===")
print(f"RMSE: {rmse_xgb:,.2f}")
print(f"MAE : {mae_xgb:,.2f}")
print(f"R^2 : {r2_xgb:.4f}")
```

```
=== XGBoost: total_emissions_t ===
RMSE: 848,306.60
MAE : 97,864.13
R^2 : 0.5168
```

5.3 Explainability (SHAP)

To generate the feature importance insights cited in the thesis, the SHAP explainer is run. *Note: This step may take 1-2 minutes depending on CPU speed.*

- **Action:** Run Cell 14.
- **Expected Output:** The SHAP summary beeswarm plot.

```python
# Cell 14: Optional SHAP analysis for XGBoost total_emissions_t

# SHAP works on the model after preprocessing.
# We have taken a sample of the training data

X_train_sample = X_train.sample(n=1000, random_state=RANDOM_STATE)

# Get the preprocessed numeric + one-hot encoded array
X_train_transformed = preprocessor_fitted.transform(X_train_sample)
feature_names = get_feature_names_from_preprocessor(preprocessor_fitted, numeric_features, categorical_features)

xgb_model = xgb_total_pipeline.named_steps['model']

explainer = shap.TreeExplainer(xgb_model)
shap_values = explainer.shap_values(X_train_transformed)

shap.summary_plot(shap_values, pd.DataFrame(X_train_transformed, columns=feature_names), max_display=20)
```

```
/tmp/ipython-input-364165999.py:17: FutureWarning: The NumPy global RNG was seeded by calling `np.random.seed`. In a future version this function will no longer use the global RNG. Pass `rng` explicitly to opt-in to the new behaviour and silence this warning.
  shap.summary_plot(shap_values, pd.DataFrame(X_train_transformed, columns=feature_names), max_display=20)
```

6 Output generation

To complete the experiment, obtained processed data and predictions are exported to CSV format from the system. These files provide for external validation of the results.

- **Action:** Run Cell 20 ("Save cleaned data").
- **Location:** Files will appear in the outputs/ folder.
    - cleaned_with_clusters.csv: Contains the data with assigned Cluster IDs.

- test_results_total_emissions.csv: Contains Actual vs. Predicted values.

Issue	Potential Cause	Solution
ModuleNotFoundError: No module named 'shap'	Library not installed.	Rerun Step 1 (!pip install) and restart the kernel.
FileNotFoundError: ...csv	Incorrect file path.	Ensure the CSV is uploaded to the root directory or update the path in Cell 2.
MemoryError during SHAP	Insufficient RAM.	Reduce the sample size in the SHAP cell: X_train.sample(n=500) instead of 1000.

References

Google Colab: Google. (n.d.). Google Colab Frequently Asked Questions. Available at: https://research.google.com/colaboratory/faq.html

Python: Python Software Foundation. (n.d.). Python 3.10.12 Documentation. Available at: https://docs.python.org/3.10/

Scikit-learn: Pedregosa, F. et al. (2011). Scikit-learn: Machine Learning in Python. Available at: https://scikit-learn.org/stable/user_guide.html