Week 3

Normalize features

In [59]:
```
X.describe().T
```

Out[59]:

	count	mean	std	min	25%	50%	75%	r
Substance	22092.0	1.500000	1.118059	0.0	0.75	1.500	2.250	3.
Unit	22092.0	0.250000	0.433023	0.0	0.00	0.000	0.250	1.
Supply Chain Emission Factors without Margins	22092.0	0.084807	0.267039	0.0	0.00	0.002	0.044	7.
Margins of Supply Chain Emission Factors	22092.0	0.012857	0.078720	0.0	0.00	0.000	0.000	3.
DQ ReliabilityScore of Factors without Margins	22092.0	3.308030	0.499643	2.0	3.00	3.000	4.000	4.
DQ TemporalCorrelation of Factors without Margins	22092.0	2.571429	0.494883	2.0	2.00	3.000	3.000	3.
DQ GeographicalCorrelation of Factors without Margins	22092.0	1.000000	0.000000	1.0	1.00	1.000	1.000	1.
DQ TechnologicalCorrelation of Factors without Margins	22092.0	2.632129	1.135661	1.0	1.00	3.000	3.000	5.
DQ DataCollection of Factors without Margins	22092.0	1.000000	0.000000	1.0	1.00	1.000	1.000	1.
Source	22092.0	0.500634	0.500011	0.0	0.00	1.000	1.000	1.

In [60]:
```
# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [61]:
```
X_scaled[0].min(),X_scaled[0].max()
```

Out[61]:
```
(np.float64(-1.3416407864998738), np.float64(1.3849614361966767))
```

In [62]:
```
np.round(X_scaled.mean()),np.round(X_scaled.std())
```

Out[62]:
```
(np.float64(-0.0), np.float64(1.0))
```

Divide the data into train and test

In [63]:
```
X.shape
```

Out[63]:
```
(22092, 10)
```

In [65]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2
```

In [66]:
```
X_train.shape
```

Out[66]:
```
(17673, 10)
```

In [67]:
```
X_test.shape
```

Out[67]:
```
(4419, 10)
```

Select the model for training

In [68]:
```
RF_model = RandomForestRegressor(random_state=42) # Initializing Random Forest
```

Step 4: Training

In [70]:
```
RF_model.fit(X_train, y_train) # Fitting the model on training data
```

Out[70]:
```
▾ RandomForestRegressor
RandomForestRegressor(random_state=42)
```

Step 5: Prediction and Evaluation

In [71]:
```
RF_y_pred = RF_model.predict(X_test) # Making predictions on the test set
```

In [72]:
```
RF_y_pred[:20]
```

Out[72]:
```
array([2.93080000e-01, 1.00000000e-03, 1.21122793e-03, 1.16130018e-03,
       0.00000000e+00, 4.00000000e-03, 1.24555977e-04, 2.20009044e-03,
       2.00000000e-03, 3.94080000e-01, 0.00000000e+00, 1.40000000e-02,
       4.08395607e-03, 7.00000000e-03, 2.15970231e-03, 2.89160331e-04,
       1.02821706e-03, 3.15790000e-01, 9.00000000e-03, 0.00000000e+00])
```

In [73]:
```
RF_mse = mean_squared_error(y_test, RF_y_pred) # Calculating Mean Squared Erro
RF_rmse = np.sqrt(RF_mse) # Calculating Root Mean Squared Error (RMSE)
# Calculating R² score
RF_r2 = r2_score(y_test, RF_y_pred)
```

```
print(f'RMSE: {RF_rmse}')
print(f'R² Score: {RF_r2}')
```

```
RMSE: 0.006143789217304181
R² Score: 0.9993280085696331
```

In [74]:
```
from sklearn.linear_model import LinearRegression # Importing Linear Regressic
LR_model = LinearRegression() # Initializing Linear Regression model
# Fitting the Linear Regression model on training data

LR_model.fit(X_train, y_train)

LR_y_pred = LR_model.predict(X_test) # Making predictions on the test set usin


LR_mse = mean_squared_error(y_test, LR_y_pred) # Calculating Mean Squared Erro
LR_rmse = np.sqrt(LR_mse) # Calculating Root Mean Squared Error (RMSE) for Lin
LR_r2 = r2_score(y_test, LR_y_pred) # Calculating R² score for Linear Regressi

print(f'RMSE: {LR_rmse}')
print(f'R² Score: {LR_r2}')
```

```
RMSE: 0.00028073792916293835
R² Score: 0.9999985968848819
```

Step 6: Hyperparameter Tuning

In [75]:
```
# Hyperparameter tuning for Random Forest Regressor using GridSearchCV
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}

# Perform grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(RandomForestRegressor(random_state=42), param_grid,

# Fit the grid search model on the training data
grid_search.fit(X_train, y_train)

# Best model from grid search
best_model = grid_search.best_estimator_
print("Best Parameters:", grid_search.best_params_)
```

```
Best Parameters: {'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 100}
```

Use best parameters for prediction

In [76]:
```
# Use the best model to make predictions on the test set
```

```
y_pred_best = best_model.predict(X_test)


HP_mse = mean_squared_error(y_test, y_pred_best)
HP_rmse = np.sqrt(HP_mse)
HP_r2 = r2_score(y_test, y_pred_best)

print(f'RMSE: {HP_rmse}')
print(f'R² Score: {HP_r2}')
```

```
RMSE: 0.005948528382514106
R² Score: 0.9993700440298772
```

Step 7: Comapartive Study and Slecting the Best model

In [77]:
```
# Create a comparative DataFrame for all models
results = {
    'Model': ['Random Forest (Default)', 'Linear Regression', 'Random Forest (
    'MSE': [RF_mse, LR_mse, HP_mse],
    'RMSE': [RF_rmse, LR_rmse, HP_rmse],
    'R2': [RF_r2, LR_r2, HP_r2]
}

# Create a DataFrame to compare the results of different models
comparison_df = pd.DataFrame(results)
print(comparison_df)
```

```
                     Model           MSE      RMSE        R2
0  Random Forest (Default)  3.774615e-05  0.006144  0.999328
1        Linear Regression  7.881378e-08  0.000281  0.999999
2    Random Forest (Tuned)  3.538499e-05  0.005949  0.999370
```

Save model and encoders

In [78]:
```
# Create a directory to save the models if it doesn't exist
!mkdir models
```

In [79]:
```
# Save model and encoders
joblib.dump(best_model, 'models/LR_model.pkl')    # Save the best model
joblib.dump(scaler, 'models/scaler.pkl') # Save the scaler used for normalizat
```

Out[79]:
```
['models/scaler.pkl']
```