# How the preprocessing and feature engineering of dsire rps summary works

## Step 1: Import Libraries

## Step 2: Load the CSV file

## Step 3: Define a Cleaning Function

def extract\_percentage(val):

if isinstance(val, str) and '%' in val:

try:

return float(val.replace('%', '').strip())

except:

return None

return None

* Purpose: Cleans the Target and Clean\_Energy\_Target columns.
* If the string includes a %, it removes the symbol and converts it to a float.
* If the value is something like “105 MW” or not a string, it returns None.

## Step 4: Clean the Percentage Columns

df['Target'] = df['Target'].apply(extract\_percentage)

df['Clean\_Energy\_Target'] = df['Clean\_Energy\_Target'].apply(extract\_percentage)

Applies the cleaning function to extract numeric percentage values from:

* Target (RPS target)
* Clean\_Energy\_Target (long-term clean energy goal)

## Step 5: Filter Only by RPS Year

df = df[(df['Year'] >= 2000) & (df['Year'] <= 2019)].copy()

Keeps only rows where:

* The RPS target year (Year) is between 2000 and 2019.
* Rows with future Clean\_Energy\_Year values like 2040 or 2050 are not removed.

## Step 6: Add Binary Flags

df['has\_rps'] = df['Target'].notna().astype(int)

df['has\_clean\_energy\_target'] = df['Clean\_Energy\_Target'].notna().astype(int)

Creates two new columns:

* has\_rps: 1 if the state has an RPS target, 0 otherwise.
* has\_clean\_energy\_target: 1 if the state has a clean energy goal, 0 otherwise

## Step 7: Feature Engineering - Duration Calculation

df['rps\_target\_duration'] = df['Year'] - 2010

df['clean\_energy\_duration'] = df['Clean\_Energy\_Year'] - 2010

Computes:

* How many years into the future (from a 2010 baseline) the RPS or clean energy target is set.
* Useful for comparing how aggressive or long-term the commitments are.

## Step 8: Fill Missing Duration Values

df['rps\_target\_duration'] = df['rps\_target\_duration'].fillna(0)

df['clean\_energy\_duration'] = df['clean\_energy\_duration'].fillna(0)

Fills any missing NaN values with 0 to prevent issues in downstream models

## 

## What Happened

This kept only RPS policies set between 2000 and 2019, and then we counted states.

However, several rows in the original dataset shared the same Year, meaning some states were excluded, and others may be duplicated or missing due to:

* Missing or invalid Target values (like “105 MW”)
* NaN in Year
* Filtering out Year values beyond 2019 (many were in 2040-2050)

Only 5 rows/states met the following conditions:

* Had an RPS target year between 2000-2019
* Had a valid numerical target (a percentage like 15%, not “105 MW”)

## Why This Matters

* Most RPS or clean energy policies in the DSIRE file are set for future years (2030-2050).
* Filtering by 2000-2019 excludes the majority of modern renewable targets.

# How the preprocessing and feature engineering of use\_all\_btu.csv works

## Step 1: Load the Raw Data

* Loads the original dataset, where each year is a separate column, and rows are combinations of MSN codes and states.

## Step 2: Reshape from Wide to Long Format

value\_vars = [col for col in energy\_df.columns if col.isdigit()]

melted = pd.melt(

energy\_df,

id\_vars=["MSN", "State"],

value\_vars=value\_vars,

var\_name="Year",

value\_name="Data"

)

* Transforms the dataset from wide format (years as columns) to long format:
  + Each row becomes a unique (State, Year, MSN) entry.
  + “Data” contains the energy consumption for that year and MSN code.

## Step 3: Clean and Convert Columns

melted["Year"] = melted["Year"].astype(int)

melted["Data"] = pd.to\_numeric(melted["Data"], errors="coerce")

* Ensures the “Year” is treated as an integer and “Data” is numeric.
* Invalid values are converted to NaN using errors=’coerce’.

## Step 4: Identify Fossil vs Renewable Energy

fossil\_prefixes = {"NG", "CL", "CO", "PA", "PC", "DF", "JF", "FF"}

renewable\_prefixes = {"HY", "WD", "WS", "SO", "GE", "WY"}

* Defines prefixes for classifying MSN codes into fossil and renewable energy.

melted = melted[melted["MSN"].str[:2].isin(fossil\_prefixes.union(renewable\_prefixes))]

melted["energy\_type"] = melted["MSN"].str[:2].map(

lambda x: "Fossil" if x in fossil\_prefixes else "Renewable"

)

* Filters the dataset to only include fossil and renewable sources
* Adds a new column “energy\_type” based on the MSN prefix

## Step 5: Aggregate Energy Use by Type

agg\_df = (

melted.groupby(["State", "Year", "energy\_type"])["Data"]

.sum()

.reset\_index()

.pivot(index=["State", "Year"], columns="energy\_type", values="Data")

.fillna(0)

.reset\_index()

)

* Aggregates energy consumption by State, Year, and energy\_type
* Pivots the result so that we have one row per state-year, with columns:
  + “Fossil” (total fossil energy)
  + “Renewable” (total renewable energy)

## Step 6: Compute Fossil Share

agg\_df["Total"] = agg\_df["Fossil"] + agg\_df["Renewable"]

agg\_df["fossil\_share"] = agg\_df["Fossil"] / agg\_df["Total"]

* Calculates the total energy consumption
* Computes the fossil energy share as a percentage of total.

## Step 7: Extract Fossil Share for 2000 and 2019

df\_2000 = agg\_df[agg\_df["Year"] == 2000][["State", "fossil\_share"]].rename(columns={"fossil\_share": "fossil\_share\_2000"})

df\_2019 = agg\_df[agg\_df["Year"] == 2019][["State", "fossil\_share"]].rename(columns={"fossil\_share": "fossil\_share\_2019"})

* Selects fossil share values for the years 2000 and 2019, per state.

merged = pd.merge(df\_2000, df\_2019, on="State")

* Merges both years’ fossil share values by state.

## Step 8: Compute Fossil Fuel Reduction and Label Success

merged["delta\_fossil\_share"] = merged["fossil\_share\_2000"] - merged["fossil\_share\_2019"]

* Computes the change in fossil fuel share from 2000 to 2019

median\_delta = merged["delta\_fossil\_share"].median()

merged["Y\_OUTCOME"] = (merged["delta\_fossil\_share"] > median\_delta).astype(int)

* Calculates the median reduction across all states.
* Labels each state as:
  + 1 (successful): reduced more than the median
  + 0 (less successful): reduced less or increased fossil use