Feature Engineering & Data Preparation (Part 3)

Objective

In this notebook, we build the final dataset used to train our machine learning model. We need to create "features" (variables) that help the model predict crop yield.

We will construct three main types of features:

1. **Historical Yields (Lag Features):** Using the yield from previous years (1, 3, and 5-year averages) to predict the future.
2. **Seasonal Weather:** Aggregating monthly weather data into seasonal averages (Winter, Spring, Summer, Fall) and shifting them to align with the crop year.
3. **Farming Inputs & Location:** Adding fertilizer/pesticide usage and GPS coordinates (Latitude/Longitude).

The final result will be saved as `x_features.parquet`.

In [11]:
```
import pandas as pd
import numpy as np
from functools import reduce
```

1. Load and Clean Data

We import the standard libraries and load the two datasets we cleaned in Part 1:

- `nasa_df.parquet` : Our weather data.
- `label_yield.parquet` : Our target crop yield data.

We also do a quick cleanup of the crop names to ensure they match perfectly.

In [12]:
```
# Load datasets
nasa_df = pd.read_parquet('Parquet/nasa_df.parquet')
label_yield = pd.read_parquet('Parquet/label_yield.parquet')

# Clean crop names for consistent column naming
label_yield['item'] = label_yield['item'].str.replace(r'[^0-9a-zA-Z ]', '', rege
label_yield['item'] = label_yield['item'].str.replace(" ", "_").str.lower()

# Generate a list of unique crops for iteration
crop_list = list(label_yield['item'].unique())
```

2. Create "Lag" Features (Past Yields)

Agricultural production often follows trends. If a farm was productive last year, it is likely to be productive this year.

We define a function `past_n_year_avg` that calculates the average yield for:

- **Lag 1:** The yield 1 year ago.
- **Lag 3:** The average yield of the last 3 years.
- **Lag 5:** The average yield of the last 5 years.

This gives the model a "memory" of recent performance.

In [13]:
```
def past_n_year_avg(df, crop_type, n):
    """
    Compute past-n-year average yield strictly for N full years.
    If less than N full past years exist, return NaN.
    """
    d = df[df['item'] == crop_type].copy()
    d['year'] = pd.to_datetime(d['year']).dt.year
    d = d.sort_values(['area', 'year'])

    def compute_avg(g):
        yrs = g['year'].values
        lbl = g['label'].values
        res = []

        for y in yrs:
            # past N years only
            mask = (yrs >= y - n) & (yrs < y)
            vals = lbl[mask]

            # strict requirement: must have exactly N rows
            if len(vals) == n:
                res.append(vals.mean())
            else:
                res.append(np.nan)

        return pd.Series(res, index=g.index)

    d[f'avg_yield_{crop_type}_{n}y'] = (
        d.groupby('area', group_keys=False)
         .apply(compute_avg, include_groups=False)
    )

    return d[['year', 'area', f'avg_yield_{crop_type}_{n}y']]
```

Generate Lags for All Crops

We run our function for every crop in our list and merge the results into a single dataframe called `features_lag_yield`.

In [14]:
```
# Iterate through all crops and generate lag features
dfs = []
for crop in crop_list:
    for n in [1, 3, 5]:
        dfs.append(past_n_year_avg(label_yield, crop, n))

# Merge all crop features into a single dataframe
features_lag_yield = reduce(
    lambda left, right: pd.merge(left, right, how='left', on=['year', 'area']),
```

```
    dfs
)
```

3. Weather Feature Engineering

Crops don't care about "January" or "February" specifically; they care about growing seasons (Spring, Summer, Autumn, Winter).

We process the weather data as follows:

1. **Group by Season:** We combine months into four seasons (e.g., Dec-Feb = Winter).
2. **Aggregate:** We calculate the **Total Rain** (Sum) and **Average Temperature/Sunlight** (Mean) for each season.
3. **Lag by 1 Year:** We align the weather from the *previous* year to the *current* crop year. This allows us to predict yields before the current season is even finished.

In [15]:
```
def prep_seasonal_weather_lag1year(nasa_df, var_list=['rain','solar','temp']):
    """
    Computes seasonal lag-1 weather features:
    - Rain -> SUM
    - Solar, Temp -> AVG
    Strict: require all months for the season/annual, else NaN
    """
    nasa_df['date'] = pd.to_datetime(nasa_df['date'])
    nasa_df['year'] = nasa_df['date'].dt.year
    nasa_df['month'] = nasa_df['date'].dt.month

    all_features = None

    for var in var_list:
        # pivot
        p = nasa_df.pivot_table(
            index=['area','year'],
            columns='month',
            values=var
        ).reset_index()

        # rename months
        month_map = {m: f"{var}_{pd.Timestamp(2000,m,1).strftime('%b').lower()}"
        p = p.rename(columns=month_map)

        # lag 1 year
        p['year'] = p['year'] + 1

        # ensure all months exist
        months = [f"{var}_{pd.Timestamp(2000,m,1).strftime('%b').lower()}" for m
        for col in months:
            if col not in p.columns:
                p[col] = pd.NA

        # define seasons
        winter = [f"{var}_jan", f"{var}_feb", f"{var}_dec"]
        spring = [f"{var}_mar", f"{var}_apr", f"{var}_may"]
        summer = [f"{var}_jun", f"{var}_jul", f"{var}_aug"]
        autumn = [f"{var}_sep", f"{var}_oct", f"{var}_nov"]

        # aggregation functions
```

```
        if var == 'rain':
            # strict sum
            agg_func = lambda df, cols: df[cols].where(df[cols].notna().all(axis
        else:
            # strict avg
            agg_func = lambda df, cols: df[cols].where(df[cols].notna().all(axis

        p[f"{'sum' if var=='rain' else 'avg'}_{var}_winter"] = agg_func(p, winte
        p[f"{'sum' if var=='rain' else 'avg'}_{var}_spring"] = agg_func(p, sprir
        p[f"{'sum' if var=='rain' else 'avg'}_{var}_summer"] = agg_func(p, summe
        p[f"{'sum' if var=='rain' else 'avg'}_{var}_autumn"] = agg_func(p, autum
        p[f"{'sum' if var=='rain' else 'avg'}_{var}_annual"] = agg_func(p, month

        # keep relevant columns
        cols_keep = ['area','year'] + [
            f"{'sum' if var=='rain' else 'avg'}_{var}_{s}" for s in ['winter','s
        ]
        p = p[cols_keep]

        # merge
        all_features = p if all_features is None else all_features.merge(p, on=[

    return all_features
```

In [16]:

```
# Process weather features
nasa_f = prep_seasonal_weather_lag1year(nasa_df, var_list=['rain','solar','temp'

# Verify
print(nasa_f.columns.tolist())
```

```
['area', 'year', 'sum_rain_winter', 'sum_rain_spring', 'sum_rain_summer', 'sum_ra
in_autumn', 'sum_rain_annual', 'avg_solar_winter', 'avg_solar_spring', 'avg_solar
_summer', 'avg_solar_autumn', 'avg_solar_annual', 'avg_temp_winter', 'avg_temp_sp
ring', 'avg_temp_summer', 'avg_temp_autumn', 'avg_temp_annual']
```

4. Add Location Data (Geospatial)

Geography plays a huge role in agriculture. We load a separate file containing the **Latitude and Longitude** for each country. This helps the model understand that "Thailand" and "Vietnam" are neighbors and might share similar traits.

In [17]:

```
# Load geospatial data (Assuming 'lat_long.csv' exists in Data folder)
latlong = pd.read_csv('Data/coordinates.csv')

# Clean and standardize formatting
latlong['area'] = latlong['Area'].str.replace(' ', '_')
latlong = latlong[['area', 'latitude', 'longitude']]

# Display sample to verify structure
latlong.head()
```

Out[17]:

	area	latitude	longitude
0	Albania	41.33	19.82
1	Algeria	28.03	1.66
2	Angola	-11.20	17.87
3	Argentina	-38.42	-63.62
4	Armenia	40.07	45.04

5. Add Farming Inputs (Fertilizers & Pesticides)

We include data on how much fertilizer and pesticide was used.

- **Logic:** We shift this data by 1 year (`Lag 1`).
- **Reason:** Farmers often plan their budget based on the previous year's usage. Using last year's data makes our prediction more practical for early forecasting.

In [18]:

```
# 1. Load the farming data
farming_df = pd.read_parquet('Parquet/farming_df.parquet')

# 2. Ensure 'year' is in datetime format for accurate date shifting
farming_df['year'] = pd.to_datetime(farming_df['year'])

# 3. Create Lag Features (Shift Year Forward by 1)
# Logic: We use 2020's pesticides for the 2021 yield row.
farming_df['year'] = farming_df['year'] + pd.DateOffset(years=1)

# === FIX START ===
# 4. Convert 'year' back to an integer to match x_features
farming_df['year'] = farming_df['year'].dt.year
# === FIX END ===

# 5. Rename columns to indicate they are lagged
farming_df = farming_df.rename(columns={
    'pesticides': 'pesticides_lag1',
    'fertilizer': 'fertilizer_lag1'
})
```

6. Final Merge and Save

We combine all our new features into one master dataset:

- **Yield Lags** + **Seasonal Weather** + **Farming Inputs** + **Location**

We filter out data before 1982 (since we don't have enough history to calculate the 5-year lag for those early years) and save the final file as `x_features_v2.parquet` .

In [19]:

```
# Merge Yield Lags with Weather Data
x_features = features_lag_yield.merge(
    nasa_f, on=['year', 'area'], how='left'
)

# 7. Merge with Farming Data
```

```python
# Now both dataframes have 'year' as an integer
x_features = x_features.merge(
    farming_df, on=['year', 'area'], how='left'
)

# Merge with Geospatial Data
x_features = x_features.merge(
    latlong, on=['area'], how='left'
)


# Prevent pandas from hiding columns
pd.set_option('display.max_columns', None)

# Show first 20 rows for Thailand
x_features[x_features['area'] == 'Thailand'].head(20)
```

Out[19]:

	year	area	avg_yield_maize_corn_1y	avg_yield_maize_corn_3y	avg_yield_maize
7309	1970	Thailand	NaN	NaN	
7310	1971	Thailand	2587.7	NaN	
7311	1972	Thailand	2421.1	NaN	
7312	1973	Thailand	1414.0	2140.933333	
7313	1974	Thailand	2227.6	2020.900000	
7314	1975	Thailand	2332.1	1991.233333	
7315	1976	Thailand	2375.2	2311.633333	
7316	1977	Thailand	2386.2	2364.500000	
7317	1978	Thailand	1717.7	2159.700000	
7318	1979	Thailand	2124.0	2075.966667	
7319	1980	Thailand	2010.3	1950.666667	
7320	1981	Thailand	2228.3	2120.866667	
7321	1982	Thailand	2353.8	2197.466667	
7322	1983	Thailand	2298.8	2293.633333	
7323	1984	Thailand	2267.4	2306.666667	
7324	1985	Thailand	2430.5	2332.233333	
7325	1986	Thailand	2571.9	2423.266667	
7326	1987	Thailand	2373.8	2458.733333	
7327	1988	Thailand	2048.6	2331.433333	
7328	1989	Thailand	2617.6	2346.666667	

In [20]:

```python
# Filter data to relevant years (1983 onwards)
x_features = x_features[x_features['year'] >= 1982]

# Save to Parquet
```

```python
x_features.to_parquet('Parquet/x_features_v2.parquet')

# Output shape for verification
print(f"Final X features shape: {x_features.shape}")

x_features.head()
```

```
Final X features shape: (6631, 66)
```

Out[20]:

	year	area	avg_yield_maize_corn_1y	avg_yield_maize_corn_3y	avg_yield_maiz
12	1982	Afghanistan	1669.0	1650.100000	
13	1983	Afghanistan	1665.8	1668.633333	
14	1984	Afghanistan	1664.1	1666.300000	
15	1985	Afghanistan	1661.2	1663.700000	
16	1986	Afghanistan	1665.2	1663.500000	