```python

def basic_clean(df):
    # 1. normalize column names
    df.columns = [c.strip() for c in df.columns]

    # 2. ensure numeric types for flux and error columns
    for c in FLUX_COLS + ERR_COLS:
        if c in df.columns:
            df[c] = pd.to_numeric(df[c], errors='coerce')

    # 3. replace negative/zero flux values with NaN (physical flux >= 0)
    for c in FLUX_COLS:
        if c in df.columns:
            df.loc[df[c] <= 0, c] = np.nan

    # 4. replace non-positive errors with NaN
    for c in ERR_COLS:
        if c in df.columns:
            df.loc[df[c] <= 0, c] = np.nan

    # 5. drop rows with all fluxes missing (useless rows)
    if set(FLUX_COLS).issubset(df.columns):
        df = df.dropna(subset=FLUX_COLS, how='all').reset_index(drop=True)

    # 6. drop exact duplicate rows
    df = df.drop_duplicates().reset_index(drop=True)

    return df
```

```python

def derive_features(df):
    # a) total_flux (sum of available fluxes)
    available_flux = [c for c in FLUX_COLS if c in df.columns]
    df['total_flux'] = df[available_flux].sum(axis=1, skipna=True)

    # b) mean_flux_err
    available_err = [c for c in ERR_COLS if c in df.columns]
    if available_err:
        df['mean_flux_err'] = df[available_err].mean(axis=1, skipna=True)
    else:
        df['mean_flux_err'] = np.nan

    # c) flux_count (# of bands with detection)
    df['flux_count'] = df[available_flux].notna().sum(axis=1)

    # d) brightness class (quantiles)
    # add 1e-12 to avoid zero problems
    df['brightness_class'] = pd.qcut(df['total_flux'].fillna(0)+1e-12, q=3, labels=['Low','Med

    # e) spectral_index_proxy: log(flux1/flux9) / log(freq1/freq9)
    # approximate freq1=30 GHz, freq9=857 GHz for Planck typical bands
    if available_flux:
        f1, f9 = available_flux[0], available_flux[-1]
        df['spectral_index_proxy'] = np.log(df[f1].replace(0, np.nan) / df[f9].replace(0, np.n

    return df
```

```python

def derive_features(df):
    # a) total_flux (sum of available fluxes)
    available_flux = [c for c in FLUX_COLS if c in df.columns]
    df['total_flux'] = df[available_flux].sum(axis=1, skipna=True)

    # b) mean_flux_err
    available_err = [c for c in ERR_COLS if c in df.columns]
    if available_err:
        df['mean_flux_err'] = df[available_err].mean(axis=1, skipna=True)
    else:
        df['mean_flux_err'] = np.nan

    # c) flux_count (# of bands with detection)
    df['flux_count'] = df[available_flux].notna().sum(axis=1)

    # d) brightness class (quantiles)
    # add 1e-12 to avoid zero problems
    df['brightness_class'] = pd.qcut(df['total_flux'].fillna(0)+1e-12, q=3, labels=['Low','Medium','High'])

    # e) spectral_index_proxy: log(flux1/flux9) / log(freq1/freq9)
    # approximate freq1=30 GHz, freq9=857 GHz for Planck typical bands
    if available_flux:
        f1, f9 = available_flux[0], available_flux[-1]
        df['spectral_index_proxy'] = np.log(df[f1].replace(0, np.nan) / df[f9].replace(0, np.nan)) / np.log(30/857)

    return df
```

```python
def normalize_fluxes(df):
    # create normalized (0-1) versions for each flux band
    for c in FLUX_COLS:
        if c in df.columns:
            col = df[c]
            minv, maxv = col.min(skipna=True), col.max(skipna=True)
            if pd.notna(minv) and pd.notna(maxv) and maxv > minv:
                df[f"{c}_norm"] = (col - minv) / (maxv - minv)
            else:
                df[f"{c}_norm"] = np.nan

    # create z-score columns as well
    present = [c for c in FLUX_COLS if c in df.columns]
    if present:
        scaler = StandardScaler()
        df_z = df[present].fillna(0)
        zvals = scaler.fit_transform(df_z)
        for i, c in enumerate(present):
            df[f"{c}_z"] = zvals[:, i]

    return df

def filter_reliable(df, rel_err_threshold=0.5):
    # mark reliable if mean_flux_err <= rel_err_threshold * total_flux
    df['reliable'] = True
    mask = (df['mean_flux_err'] > rel_err_threshold * df['total_flux'])
    df.loc[mask, 'reliable'] = False
    return df
```

```python

def iqr_filter(df, col='total_flux', k=1.5):
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)
    iqr = q3 - q1
    lower, upper = q1 - k*iqr, q3 + k*iqr
    return df[(df[col] >= lower) & (df[col] <= upper)].reset_index(drop=True)

def full_pipeline(path="planck.csv"):
    raw = load_raw(path)
    cleaned = basic_clean(raw.copy())
    derived = derive_features(cleaned)
    normed = normalize_fluxes(derived)
    normed = filter_reliable(normed)
    return raw, normed
```