# PROPHET

June 14, 2025

[8]:
```
import os
import pandas as pd
import numpy as np
from prophet import Prophet
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

[9]:
```
#    1. PREPROCESSING (base)
def preprocess_data(file_path):
    if not os.path.exists(file_path):
        raise FileNotFoundError(f"File not found: {file_path}")

    # pasing man..
    df = pd.read_csv(
        file_path,
        sep=';',
        decimal=',',
        low_memory=False
    )

    # D+T=ds
    df['ds'] = pd.to_datetime(
        df['Date'].str.strip() + ' ' + df['Time'].str.strip(),
        dayfirst=True,
        format='%d/%m/%Y %H:%M:%S'
    )

    # setting Y
    if 'Global_active_power' not in df:
        raise KeyError("'Global_active_power' column missing")
    df['y'] = pd.to_numeric(df['Global_active_power'], errors='coerce')

    df = df[['ds','y']].dropna(subset=['y']).reset_index(drop=True)
    print(f"  Preprocessed {len(df)} rows (minute-level).")
    return df
```

[11]:
```
'''DATA_PATH = r"C:
 ↪\Users\bhara\OneDrive\Pictures\Desktop\household_power_consumption.txt"
df_1= preprocess_data(DATA_PATH) #secondary'''
```

```
Preprocessed 2049280 rows (minute-level).
```

[12]:
```
PARAM_GRID = [
        {
            'seasonality_mode': 'additive',
            'daily_seasonality': False,
            'weekly_seasonality': True,
            'yearly_seasonality': True,
            'changepoint_prior_scale': cps
        }
        for cps in [0.01, 0.05, 0.1, 0.5]
    ]
TARGET_MAE = 0.5
```

[21]:
```
#    2+3. TRAIN & TUNE LOOP
def train_and_tune_prophet(train_df, val_df, param_grid, target_mae):
    #fit Prophet over each params in param_grid,
    #evaluate on val_df (dropping NaNs),
    #stop early if MAE  target_mae.

    best_model = None
    best_mae   = np.inf
    best_params= None

    for params in param_grid:
        model = Prophet(**params) #taking as arbitrary
        model.fit(train_df)

        # full forecast for val period
        future   = model.make_future_dataframe(periods=len(val_df), freq='D')
        forecast = model.predict(future)

        # merge with actuals, drop any NaNs before scoring
        df_val = val_df[['ds','y']].merge(
            forecast[['ds','yhat']], on='ds', how='left'
        ).dropna(subset=['y','yhat'])

        mae = mean_absolute_error(df_val['y'], df_val['yhat'])
        print(f" cps={params['changepoint_prior_scale']:<5} → val MAE = {mae:.
 ↪4f}")

        if mae < best_mae:
            best_mae, best_model, best_params = mae, model, params
```

```
            if mae <= target_mae:
                print("Target MAE reached....")
                break

    print(f"\n Best MAE: {best_mae:.4f} with params: {best_params}")
    return best_model
```

[17]:
```
#    4+5. FORECAST & EVALUATE
def forecast_and_evaluate(model, full_df, test_df):
    """
    1) forecast the full + test period, keeping all columns.
    2) merge & drop NaNs for error metrics.
    3) return the full forecast (for plotting).
    """
    future   = model.make_future_dataframe(periods=len(test_df), freq='D')
    forecast = model.predict(future)   # includes yhat, yhat_lower, yhat_upper,␣
 ↪etc.

    # Prepare test-only DataFrame for metrics
    df_test = test_df[['ds','y']].merge(
        forecast[['ds','yhat']], on='ds', how='left'
    ).dropna(subset=['y','yhat'])
    # Assuming `full_forecast` is your Prophet output...
    mae  = mean_absolute_error(df_test['y'], df_test['yhat'])
    rmse = mean_squared_error(df_test['y'], df_test['yhat'], squared=False)
    r2   = r2_score(df_test['y'], df_test['yhat'])

    print("\n--- Final Test Metrics ---")
    print(f"MAE:  {mae:.4f}")
    print(f"RMSE: {rmse:.4f}")
    print(f"R²:   {r2:.4f}")

    return forecast
```

[ ]:
```
if __name__ == "__main__":
    DATA_PATH = r"C:
 ↪\Users\bhara\OneDrive\Pictures\Desktop\household_power_consumption.txt"

    # 1. Preprocess minute-level data
    df_minute = preprocess_data(DATA_PATH)

    # 2. Aggregate to daily (so Stan/Prophet runs quickly)
    df = (
        df_minute
          .set_index('ds')
          .y.resample('D')    # 'D' = daily frequency
          .mean()
```

```
            .reset_index()
    )
    print(f"Now modeling on {len(df)} days of data.")

    # 3. Split: 80% train / 20% test, then hold out 10% of train for validation
    cut       = int(len(df) * 0.8)
    train_all, test = df.iloc[:cut], df.iloc[cut:]
    hold      = int(len(train_all) * 0.1)
    train, val= train_all.iloc[:-hold], train_all.iloc[-hold:]
    print(f"Train: {len(train)}, Val: {len(val)}, Test: {len(test)}")

    # 4. Build parameter grid done ...
    # 5. Train & tune
    best_model = train_and_tune_prophet(train, val, PARAM_GRID, TARGET_MAE)

    # 6. Forecast & evaluate
    full_forecast = forecast_and_evaluate(best_model, df, test)
    #july_forecast = full_forecast[(full_forecast['ds'] >= '2010-07-01') &
 ↪(full_forecast['ds'] < '2010-08-01')
#]
    #print(july_forecast[['ds','yhat','yhat_lower','yhat_upper']])
    fig1 = best_model.plot(full_forecast)
    fig2 = best_model.plot_components(full_forecast)
    plt.show()
```

[ ]: