In [134…

```python
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import pandas as pd
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBRegressor
import scipy.stats as stats
```

In [111…

```python
df = pd.read_excel("./datasets/datasets.xlsx", sheet_name="Sheet1")

features = [
    'Tempmax_C', 'Tempmin_C', 'windspeedmax', 'windspeedmean',
    'solarradiation', 'uvindex', 'cloudcover', 'humidity', 'precip'
]

X = df[features]
y_wind = df['Wind_GWh']
y_solar = df['(Combined) Solar_GWh']  # 已合并 Rooftop + Utility 的值


X_train_wind, X_test_wind, y_train_wind, y_test_wind = train_test_split(X, y
X_train_solar, X_test_solar, y_train_solar, y_test_solar = train_test_split(
print(np.mean(y_train_wind), np.mean(y_test_wind))
model_wind = xgb.XGBRegressor(
    n_estimators=200,
    eta=0.31,
    max_depth=4,
    random_state=42,
    gamma=49,
    reg_lambda=1.02,
    )
model_solar = xgb.XGBRegressor(
    n_estimators=200,
    eta=0.31,
    max_depth=4,
    random_state=42
)


model_wind.fit(X_train_wind, y_train_wind)
model_solar.fit(X_train_solar, y_train_solar)

pred_wind = model_wind.predict(X_test_wind)
pred_solar = model_solar.predict(X_test_solar)

pred_combined = pred_wind + pred_solar
y_combined_true = y_test_wind + y_test_solar

rmse_wind = np.sqrt(mean_squared_error(y_test_wind, pred_wind))
r2_wind = r2_score(y_test_wind, pred_wind)
rmse_sloar = np.sqrt(mean_squared_error(y_test_solar, pred_solar))
r2_sloar = r2_score(y_test_solar, pred_solar)
```

```python
rmse = np.sqrt(mean_squared_error(y_combined_true, pred_combined))
r2 = r2_score(y_combined_true, pred_combined)

print(f"RMSE (wind): {rmse_wind:.3f}")
print(f"R² (wind): {r2_wind:.3f}")

print(f"RMSE (solar): {rmse_sloar:.3f}")
print(f"R² (solar): {r2_sloar:.3f}")

print(f"RMSE (Combined): {rmse:.3f}")
print(f"R² (Combined): {r2:.3f}")
```

```
18.601061643835617 18.162297297297297
RMSE (wind): 3.990
R² (wind): 0.798
RMSE (solar): 1.022
R² (solar): 0.945
RMSE (Combined): 4.059
R² (Combined): 0.808
```

In [109…

```python
values = [i/100 for i in range(1, 101)]
best_rmse = float('inf')
best_r2 = 0
best_value = None
print(np.mean(y_train_wind), np.mean(y_test_wind))
for value in values:
    print(f"Trying value = {value}")
    model = xgb.XGBRegressor(
        n_estimators=200,
        eta=value,
        max_depth=4,
        random_state=42
    )
    model.fit(X_train_solar, y_train_solar)
    y_pred = model.predict(X_test_solar)

    rmse = np.sqrt(mean_squared_error(y_test_solar, y_pred))
    r2 = r2_score(y_test_solar, y_pred)

    # 如果 rmse 无效则跳过
    if np.isnan(rmse):
        print(f" Skipping eta = {eta} due to NaN RMSE")
        continue

    if rmse < best_rmse:
        best_rmse = rmse
        best_r2 = r2
        best_value = value


    print(f"value: {value:.2f}, RMSE: {rmse:.3f}, R²: {r2:.3f}")

print(f"\nBest value = {best_value:.2f}, RMSE = {best_rmse:.3f}, R² = {best_
```

```
18.601061643835617 18.162297297297297
Trying value = 0.01
value: 0.01, RMSE: 1.159, R²: 0.929
Trying value = 0.02
value: 0.02, RMSE: 1.010, R²: 0.946
Trying value = 0.03
value: 0.03, RMSE: 1.003, R²: 0.947
Trying value = 0.04
value: 0.04, RMSE: 1.007, R²: 0.946
Trying value = 0.05
value: 0.05, RMSE: 1.021, R²: 0.945
Trying value = 0.06
value: 0.06, RMSE: 1.022, R²: 0.945
Trying value = 0.07
value: 0.07, RMSE: 1.020, R²: 0.945
Trying value = 0.08
value: 0.08, RMSE: 1.037, R²: 0.943
Trying value = 0.09
value: 0.09, RMSE: 1.052, R²: 0.941
Trying value = 0.1
value: 0.10, RMSE: 1.034, R²: 0.943
Trying value = 0.11
value: 0.11, RMSE: 1.006, R²: 0.946
Trying value = 0.12
value: 0.12, RMSE: 1.039, R²: 0.943
Trying value = 0.13
value: 0.13, RMSE: 1.015, R²: 0.945
Trying value = 0.14
value: 0.14, RMSE: 1.058, R²: 0.941
Trying value = 0.15
value: 0.15, RMSE: 1.022, R²: 0.945
Trying value = 0.16
value: 0.16, RMSE: 1.033, R²: 0.943
Trying value = 0.17
value: 0.17, RMSE: 1.040, R²: 0.943
Trying value = 0.18
value: 0.18, RMSE: 1.013, R²: 0.945
Trying value = 0.19
value: 0.19, RMSE: 1.007, R²: 0.946
Trying value = 0.2
value: 0.20, RMSE: 1.044, R²: 0.942
Trying value = 0.21
value: 0.21, RMSE: 1.042, R²: 0.942
Trying value = 0.22
value: 0.22, RMSE: 1.029, R²: 0.944
Trying value = 0.23
value: 0.23, RMSE: 1.064, R²: 0.940
Trying value = 0.24
value: 0.24, RMSE: 1.075, R²: 0.939
Trying value = 0.25
value: 0.25, RMSE: 1.031, R²: 0.944
Trying value = 0.26
value: 0.26, RMSE: 1.038, R²: 0.943
Trying value = 0.27
value: 0.27, RMSE: 1.012, R²: 0.946
Trying value = 0.28
```

```
value: 0.28, RMSE: 1.047, R²: 0.942
Trying value = 0.29
value: 0.29, RMSE: 1.055, R²: 0.941
Trying value = 0.3
value: 0.30, RMSE: 1.093, R²: 0.937
Trying value = 0.31
value: 0.31, RMSE: 1.022, R²: 0.945
Trying value = 0.32
value: 0.32, RMSE: 1.059, R²: 0.940
Trying value = 0.33
value: 0.33, RMSE: 1.145, R²: 0.930
Trying value = 0.34
value: 0.34, RMSE: 1.063, R²: 0.940
Trying value = 0.35
value: 0.35, RMSE: 1.096, R²: 0.936
Trying value = 0.36
value: 0.36, RMSE: 1.031, R²: 0.944
Trying value = 0.37
value: 0.37, RMSE: 1.091, R²: 0.937
Trying value = 0.38
value: 0.38, RMSE: 1.033, R²: 0.943
Trying value = 0.39
value: 0.39, RMSE: 1.078, R²: 0.938
Trying value = 0.4
value: 0.40, RMSE: 1.061, R²: 0.940
Trying value = 0.41
value: 0.41, RMSE: 1.023, R²: 0.944
Trying value = 0.42
value: 0.42, RMSE: 1.123, R²: 0.933
Trying value = 0.43
value: 0.43, RMSE: 1.143, R²: 0.931
Trying value = 0.44
value: 0.44, RMSE: 1.109, R²: 0.935
Trying value = 0.45
value: 0.45, RMSE: 1.130, R²: 0.932
Trying value = 0.46
value: 0.46, RMSE: 1.113, R²: 0.934
Trying value = 0.47
value: 0.47, RMSE: 1.106, R²: 0.935
Trying value = 0.48
value: 0.48, RMSE: 1.057, R²: 0.941
Trying value = 0.49
value: 0.49, RMSE: 1.059, R²: 0.940
Trying value = 0.5
value: 0.50, RMSE: 1.112, R²: 0.934
Trying value = 0.51
value: 0.51, RMSE: 1.047, R²: 0.942
Trying value = 0.52
value: 0.52, RMSE: 1.083, R²: 0.938
Trying value = 0.53
value: 0.53, RMSE: 1.125, R²: 0.933
Trying value = 0.54
value: 0.54, RMSE: 1.184, R²: 0.926
Trying value = 0.55
value: 0.55, RMSE: 1.121, R²: 0.933
Trying value = 0.56
```

```
value: 0.56, RMSE: 1.067, R²: 0.940
Trying value = 0.57
value: 0.57, RMSE: 1.103, R²: 0.935
Trying value = 0.58
value: 0.58, RMSE: 1.176, R²: 0.926
Trying value = 0.59
value: 0.59, RMSE: 1.160, R²: 0.929
Trying value = 0.6
value: 0.60, RMSE: 1.151, R²: 0.930
Trying value = 0.61
value: 0.61, RMSE: 1.238, R²: 0.919
Trying value = 0.62
value: 0.62, RMSE: 1.139, R²: 0.931
Trying value = 0.63
value: 0.63, RMSE: 1.057, R²: 0.941
Trying value = 0.64
value: 0.64, RMSE: 1.060, R²: 0.940
Trying value = 0.65
value: 0.65, RMSE: 1.004, R²: 0.946
Trying value = 0.66
value: 0.66, RMSE: 1.068, R²: 0.939
Trying value = 0.67
value: 0.67, RMSE: 1.074, R²: 0.939
Trying value = 0.68
value: 0.68, RMSE: 1.049, R²: 0.942
Trying value = 0.69
value: 0.69, RMSE: 1.092, R²: 0.937
Trying value = 0.7
value: 0.70, RMSE: 1.042, R²: 0.942
Trying value = 0.71
value: 0.71, RMSE: 1.059, R²: 0.940
Trying value = 0.72
value: 0.72, RMSE: 1.075, R²: 0.939
Trying value = 0.73
value: 0.73, RMSE: 0.983, R²: 0.949
Trying value = 0.74
value: 0.74, RMSE: 1.169, R²: 0.927
Trying value = 0.75
value: 0.75, RMSE: 1.169, R²: 0.927
Trying value = 0.76
value: 0.76, RMSE: 1.149, R²: 0.930
Trying value = 0.77
value: 0.77, RMSE: 1.288, R²: 0.912
Trying value = 0.78
value: 0.78, RMSE: 1.286, R²: 0.912
Trying value = 0.79
value: 0.79, RMSE: 1.278, R²: 0.913
Trying value = 0.8
value: 0.80, RMSE: 1.174, R²: 0.927
Trying value = 0.81
value: 0.81, RMSE: 1.174, R²: 0.927
Trying value = 0.82
value: 0.82, RMSE: 1.240, R²: 0.918
Trying value = 0.83
value: 0.83, RMSE: 1.173, R²: 0.927
Trying value = 0.84
```

```
value: 0.84, RMSE: 1.334, R²: 0.905
Trying value = 0.85
value: 0.85, RMSE: 1.325, R²: 0.907
Trying value = 0.86
value: 0.86, RMSE: 1.345, R²: 0.904
Trying value = 0.87
value: 0.87, RMSE: 1.442, R²: 0.890
Trying value = 0.88
value: 0.88, RMSE: 1.231, R²: 0.920
Trying value = 0.89
value: 0.89, RMSE: 1.248, R²: 0.917
Trying value = 0.9
value: 0.90, RMSE: 1.351, R²: 0.903
Trying value = 0.91
value: 0.91, RMSE: 1.314, R²: 0.908
Trying value = 0.92
value: 0.92, RMSE: 1.416, R²: 0.894
Trying value = 0.93
value: 0.93, RMSE: 1.321, R²: 0.907
Trying value = 0.94
value: 0.94, RMSE: 1.363, R²: 0.901
Trying value = 0.95
value: 0.95, RMSE: 1.415, R²: 0.894
Trying value = 0.96
value: 0.96, RMSE: 1.303, R²: 0.910
Trying value = 0.97
value: 0.97, RMSE: 1.224, R²: 0.920
Trying value = 0.98
value: 0.98, RMSE: 1.396, R²: 0.897
Trying value = 0.99
value: 0.99, RMSE: 1.245, R²: 0.918
Trying value = 1.0
value: 1.00, RMSE: 1.258, R²: 0.916

Best value = 0.73, RMSE = 0.983, R² = 0.949
```

In [136…

```
# Features and target
features = [
    'Tempmax_C', 'Tempmin_C', 'windspeedmax', 'windspeedmean',
    'solarradiation', 'uvindex', 'cloudcover', 'humidity', 'precip'
]
X = df[features]
y_wind = df['Wind_GWh']
y_solar = df['(Combined) Solar_GWh']

# Split data once to ensure alignment across targets
X_train, X_test, y_train_wind, y_test_wind, y_train_solar, y_test_solar = tr
    X, y_wind, y_solar, test_size=0.2, random_state=42
)

# Define models for wind and solar
model_wind = xgb.XGBRegressor(
    n_estimators=200,
    eta=0.31,
    max_depth=4,
    gamma=49,
```

```python
    reg_lambda=1.02,
    random_state=42
)

model_solar = xgb.XGBRegressor(
    n_estimators=200,
    eta=0.31,
    max_depth=4,
    random_state=42
)

# Train the models
model_wind.fit(X_train, y_train_wind)
model_solar.fit(X_train, y_train_solar)

# Make predictions
pred_wind = model_wind.predict(X_test)
pred_solar = model_solar.predict(X_test)

# Combine predictions and true values (aligned by row)
pred_combined = pred_wind + pred_solar
y_combined_true = y_test_wind + y_test_solar

# Evaluate model performance
rmse_wind = np.sqrt(mean_squared_error(y_test_wind, pred_wind))
r2_wind = r2_score(y_test_wind, pred_wind)

rmse_solar = np.sqrt(mean_squared_error(y_test_solar, pred_solar))
r2_solar = r2_score(y_test_solar, pred_solar)

rmse_combined = np.sqrt(mean_squared_error(y_combined_true, pred_combined))
r2_combined = r2_score(y_combined_true, pred_combined)

# Print evaluation results
print(f"RMSE (Wind):     {rmse_wind:.3f}, R²: {r2_wind:.3f}")
print(f"RMSE (Solar):    {rmse_solar:.3f}, R²: {r2_solar:.3f}")
print(f"RMSE (Combined): {rmse_combined:.3f}, R²: {r2_combined:.3f}")
```

```
RMSE (Wind):     3.990, R²: 0.798
RMSE (Solar):    1.022, R²: 0.945
RMSE (Combined): 4.059, R²: 0.808
```

In [137…

```python
# Define features and targets
features = [
    'Tempmax_C', 'Tempmin_C', 'windspeedmax', 'windspeedmean',
    'solarradiation', 'uvindex', 'cloudcover', 'humidity', 'precip'
]
X = df[features]
y_wind = df['Wind_GWh']
y_solar = df['(Combined) Solar_GWh']
y_combined = y_wind + y_solar

# Split all data in one step to ensure row alignment
X_train, X_test, y_train_wind, y_test_wind, y_train_solar, y_test_solar, y_t
    X, y_wind, y_solar, y_combined, test_size=0.2, random_state=42
)
```

```python
# Train base models (Wind and Solar)
model_wind = xgb.XGBRegressor(
    eta=0.31, max_depth=4, n_estimators=200, gamma=49, reg_lambda=1.02, ranc
)
model_solar = xgb.XGBRegressor(
    eta=0.31, max_depth=4, n_estimators=200, random_state=42
)

model_wind.fit(X_train, y_train_wind)
model_solar.fit(X_train, y_train_solar)

# Predict using base models
pred_wind = model_wind.predict(X_test)
pred_solar = model_solar.predict(X_test)

# Construct stacking input features from base model predictions
stacked_X = np.column_stack([pred_wind, pred_solar])

# Train meta-model using combined target
meta_model = xgb.XGBRegressor(
    eta=0.1, max_depth=2, n_estimators=100, random_state=42
)
meta_model.fit(stacked_X, y_test_combined)

# Predict combined output using meta-model
y_pred_combined = meta_model.predict(stacked_X)

# Evaluate the stacked model
rmse_stacked = np.sqrt(mean_squared_error(y_test_combined, y_pred_combined))
r2_stacked = r2_score(y_test_combined, y_pred_combined)

print(f"RMSE (XGB Stacked Combined): {rmse_stacked:.3f}")
print(f"R²   (XGB Stacked Combined): {r2_stacked:.3f}")
```

```
RMSE (XGB Stacked Combined): 2.202
R²   (XGB Stacked Combined): 0.944
```

In [133…

```python
# Load the dataset
df = pd.read_excel("./datasets/datasets.xlsx", sheet_name="Sheet1")

# Define features and targets
features = [
    'Tempmax_C', 'Tempmin_C', 'windspeedmax', 'windspeedmean',
    'solarradiation', 'uvindex', 'cloudcover', 'humidity', 'precip'
]
X = df[features]
y_wind = df['Wind_GWh']
y_solar = df['(Combined) Solar_GWh']
y_combined = y_wind + y_solar

# Split all data in one step to ensure alignment across targets
X_train, X_test, y_train_wind, y_test_wind, y_train_solar, y_test_solar, y_t
    X, y_wind, y_solar, y_combined, test_size=0.2, random_state=42
)
```

```python
# Train base models for wind and solar generation
model_wind = XGBRegressor(
    eta=0.31, max_depth=4, n_estimators=200, gamma=49, reg_lambda=1.02, ranc
)
model_solar = XGBRegressor(
    eta=0.31, max_depth=4, n_estimators=200, random_state=42
)

model_wind.fit(X_train, y_train_wind)
model_solar.fit(X_train, y_train_solar)

# Generate predictions from base models for stacking input
pred_wind = model_wind.predict(X_test)
pred_solar = model_solar.predict(X_test)
stacked_X = np.column_stack([pred_wind, pred_solar])  # shape: (n_samples, 2

# Define parameter grid for GridSearchCV
param_grid = {
    'eta': [0.01, 0.02, 0.03, 0.05, 0.1, 0.2],
    'max_depth': [2, 3, 4, 5],
    'n_estimators': [50, 100, 150, 200, 250],
    'gamma': [0, 1, 2, 5, 10],
    'reg_lambda': [0, 0.5, 1, 2],
    'colsample_bytree': [0.7, 0.8, 0.9, 1.0],
    'subsample': [0.6, 0.7, 0.8, 0.9, 1.0]
}  # Total combinations: 6×4×5×5×4×4×5 = 96,000

# Initialize GridSearchCV for the meta-model
grid_search = GridSearchCV(
    estimator=XGBRegressor(random_state=42),
    param_grid=param_grid,
    cv=5,
    scoring='neg_root_mean_squared_error',
    verbose=1,
    n_jobs=-1
)

# Fit meta-model using combined generation as the target
grid_search.fit(stacked_X, y_test_combined)

# Evaluate the best meta-model
best_meta_model = grid_search.best_estimator_
y_pred_combined = best_meta_model.predict(stacked_X)

rmse = np.sqrt(mean_squared_error(y_test_combined, y_pred_combined))
r2 = r2_score(y_test_combined, y_pred_combined)

print("\nBest Meta-Model Parameters:")
print(grid_search.best_params_)
print(f"\nFinal RMSE (Stacked): {rmse:.3f}")
print(f"Final R²   (Stacked): {r2:.3f}")
```

```
Fitting 5 folds for each of 48000 candidates, totalling 240000 fits

Best Meta-Model Parameters:
{'colsample_bytree': 1.0, 'eta': 0.1, 'gamma': 1, 'max_depth': 2, 'n_estimat
ors': 50, 'reg_lambda': 0, 'subsample': 0.6}

inal RMSE (Stacked): 2.548
Final R²   (Stacked): 0.924
```

In [135…

```
# Define features and targets
features = [
    'Tempmax_C', 'Tempmin_C', 'windspeedmax', 'windspeedmean',
    'solarradiation', 'uvindex', 'cloudcover', 'humidity', 'precip'
]
X = df[features]
y_wind = df['Wind_GWh']
y_solar = df['(Combined) Solar_GWh']
y_combined = y_wind + y_solar

# Split all data at once to ensure row alignment
X_train, X_test, y_train_wind, y_test_wind, y_train_solar, y_test_solar, y_t
    X, y_wind, y_solar, y_combined, test_size=0.2, random_state=42
)

# Train base models for wind and solar generation
model_wind = XGBRegressor(
    eta=0.31, max_depth=4, n_estimators=200, gamma=49, reg_lambda=1.02,
    random_state=42
)
model_solar = XGBRegressor(
    eta=0.31, max_depth=4, n_estimators=200, random_state=42
)

model_wind.fit(X_train, y_train_wind)
model_solar.fit(X_train, y_train_solar)

# Use base model predictions as stacking features
pred_wind = model_wind.predict(X_test)
pred_solar = model_solar.predict(X_test)
stacked_X = np.column_stack([pred_wind, pred_solar])

# Define parameter distributions for randomized search
param_dist = {
    'eta': stats.uniform(0.01, 0.3),              # 0.01 to 0.31
    'max_depth': [2, 3, 4, 5, 6],
    'n_estimators': [50, 100, 150, 200, 250],
    'gamma': [0, 1, 2, 5, 10],
    'reg_lambda': stats.uniform(0, 3),            # 0 to 3
    'colsample_bytree': stats.uniform(0.7, 0.3),  # 0.7 to 1.0
    'subsample': stats.uniform(0.6, 0.4)          # 0.6 to 1.0
}

# Set up RandomizedSearchCV for the meta-model
random_search = RandomizedSearchCV(
    estimator=XGBRegressor(tree_method='hist', device='cuda', random_state=4
    param_distributions=param_dist,
```

```python
    n_iter=2400,
    scoring='neg_root_mean_squared_error',
    cv=5,
    verbose=1,
    n_jobs=-1,
    random_state=42
)

# Fit the randomized search (may take a long time)
random_search.fit(stacked_X, y_test_combined)

# Evaluate the best meta-model
best_model = random_search.best_estimator_
y_pred_combined = best_model.predict(stacked_X)

rmse = np.sqrt(mean_squared_error(y_test_combined, y_pred_combined))
r2 = r2_score(y_test_combined, y_pred_combined)

print("\nBest Parameters (RandomizedSearchCV):")
print(random_search.best_params_)
print(f"Final RMSE: {rmse:.3f}")
print(f"Final R²: {r2:.3f}")
```

```
Fitting 5 folds for each of 2400 candidates, totalling 12000 fits

Best Parameters (RandomizedSearchCV):
{'colsample_bytree': 0.8532241907732696, 'eta': 0.1352233009446337, 'gamma':
10, 'max_depth': 2, 'n_estimators': 150, 'reg_lambda': 0.8082370013955644,
'subsample': 0.6976502088991097}
Final RMSE: 2.134
Final R²: 0.947
```