Modelling for Question 1: Solar Power Generation Forecasting (Python)

Carlos Peralta

2025-08-06

```
import pandas as pd
import plotly.express as px
from skimpy import skim
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
```

Data Loading and Preparation

We load the two datasets relevant to Question 1:

- `germany_atm_features_q1.csv`: Meteorological data.
- `germany_solar_observation_q1.csv`: Solar power generation data.

These datasets are then merged into a single dataframe for easier analysis.

```
df_solar = pd.read_csv('../data/germany_solar_observation_q2.csv', parse_dates=['DateTime'])
df_atm = pd.read_csv('../data/germany_atm_features_q2.csv', parse_dates=['DateTime'])

# Merge on DateTime
df = pd.merge(df_solar, df_atm, on='DateTime', how='inner')

# Feature enrichment: hour of day, day of year
```

```
df['hour'] = df['DateTime'].dt.hour
df['dayofyear'] = df['DateTime'].dt.dayofyear

# Prepare features and target
features = ['surface_solar_radiation_downwards', 'temperature_2m', 'total_cloud_cover',
            'total_precipitation', 'snowfall', 'snow_depth', 'wind_speed_10m',
            'wind_speed_100m', 'apparent_temperature', 'relative_humidity_2m',
            'hour', 'dayofyear']
X = df[features]
y = df['power']
```

Train/test split: last 30 days as test

```
df = df.sort_values('DateTime')
split_date = df['DateTime'].max() - pd.Timedelta(days=30)
train_idx = df['DateTime'] < split_date
test_idx = df['DateTime'] >= split_date
X_train, X_test = X[train_idx], X[test_idx]
y_train, y_test = y[train_idx], y[test_idx]
```

Modelling

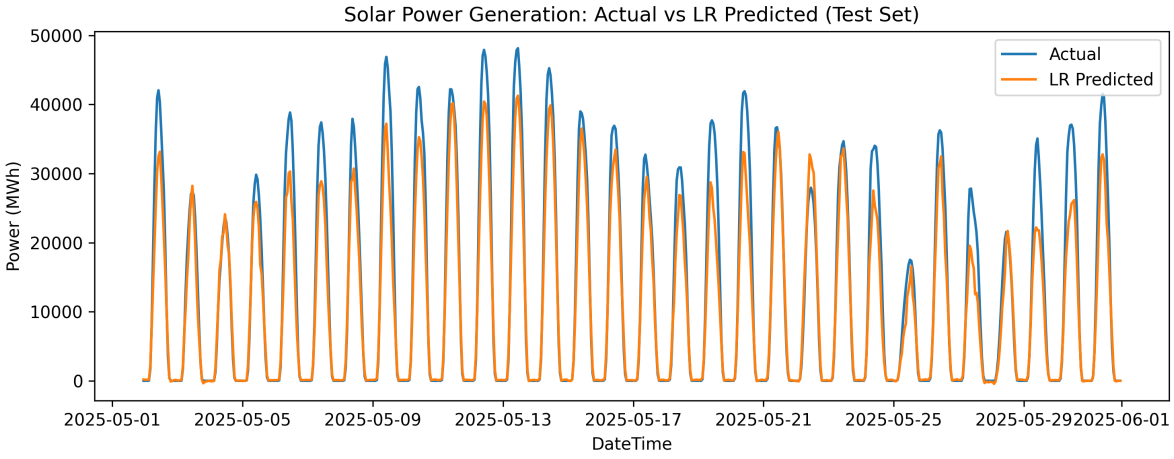
We test N simple models - linear regression - other to add - XGBoost - LSTM

We use persistence as a simple baseline. The persistence model predicts the next value is the current value.

```
# --- Linear Regression Model ---
model = LinearRegression()
model.fit(X_train, y_train)
y_pred_lr = model.predict(X_test)

rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
r2_lr = r2_score(y_test, y_pred_lr)

# --- Persistence Model ---
#For the test set, we need the last value from the training set or the previous value in the
#To align with the test set, we'll use the actual previous hour's value from the full dataset
#We need to ensure the DateTime is sorted for this.
```

```
df_test = df[test_idx].copy()
df_test['power_lag1'] = df_test['power'].shift(1)

# Handle the very first value of the test set: use the last value from the training set
if not y_train.empty and not df_test.empty:
    df_test.loc[df_test.index[0], 'power_lag1'] = y_train.iloc[-1]

y_pred_persistence = df_test['power_lag1'].values

# We need to ensure y_test and y_pred_persistence have the same length after handling the shi
# The first value of y_pred_persistence will be based on the last training value,
# so we compare from the second element of y_test onwards.
# Or, more simply, we can just drop the first row of the test set for persistence calculatio
# to avoid complexity with the train/test boundary.
# Let's simplify by comparing on the shifted data, ensuring no NaN in y_pred_persistence
# and aligning with y_test.
# For simplicity, let's just use the shifted values directly for comparison,
# dropping the first row where y_pred_persistence would be NaN if not handled.
# A more robust way is to ensure the shift correctly aligns with the test set.
# For this simple comparison, we'll use the shifted values directly.
# Let's re-align y_test and y_pred_persistence to ensure they match after the shift.
y_test_persistence = y_test.iloc[1:]
y_pred_persistence = y_pred_persistence[1:]


rmse_persistence = np.sqrt(mean_squared_error(y_test_persistence, y_pred_persistence))
r2_persistence = r2_score(y_test_persistence, y_pred_persistence)
```

```
## Plot actual vs predicted (Linear Regression)
plt.figure(figsize=(10,4))
plt.plot(df['DateTime'][test_idx], y_test.values, label='Actual')
plt.plot(df['DateTime'][test_idx], y_pred_lr, label='LR Predicted')
plt.title('Solar Power Generation: Actual vs LR Predicted (Test Set)')
plt.xlabel('DateTime')
plt.ylabel('Power (MWh)')
plt.legend()
plt.tight_layout()
plt.show()
```



Summary

```
# Output summary
print('--- Solar Power Generation Forecasting ---')
print(f'\nLinear Regression Model:')
print(f'  Test RMSE: {rmse_lr:.2f}, R2: {r2_lr:.3f}')

print(f'\nPersistence Model:')
print(f'  Test RMSE: {rmse_persistence:.2f}, R2: {r2_persistence:.3f}')

print('\nFeature importances (coefficients for Linear Regression):')
feature_importance = pd.Series(model.coef_, index=features).sort_values(ascending=False)
print(feature_importance)
```

```
--- Solar Power Generation Forecasting ---

Linear Regression Model:
  Test RMSE: 4203.94, R2: 0.918

Persistence Model:
  Test RMSE: 4391.67, R2: 0.911

Feature importances (coefficients for Linear Regression):
apparent_temperature                 158.486847
wind_speed_100m                       82.737320
snowfall                              52.922470
```

```
surface_solar_radiation_downwards     49.536016
relative_humidity_2m                   3.203976
hour                                   2.082117
dayofyear                              0.251136
total_cloud_cover                    -15.081159
wind_speed_10m                       -69.411117
snow_depth                          -159.955775
temperature_2m                      -195.413845
total_precipitation                 -787.550239
dtype: float64
```