# optimizing solar energy storage and grid distribution

In [ ]:

```

```

## Load & Preprocess Data

In [1]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
C:\Users\reshm\anaconda3\Lib\site-packages\pandas\core\arrays\masked.py:60: UserWarning: Pandas requires version '1.3.6' or newer of 'bottleneck' (version '1.3.5' currently i
nstalled).
  from pandas.core import (
```

In [27]:

```python
# Load Data
file_path = "D:/Capstone_2025/code/preprocessed_data.csv"  # Update with your dataset file path
df = pd.read_csv(file_path)
df
```

Out[27]:

| | Year | Month | Day | Hour | Clearsky DHI | Clearsky DNI | Temperature | Clearsky GHI | cloud fill flag | Cloud Type | ... | Surface Albedo | Pressure | Precipitable Water | Wind Direction | Wind Speed | DayOfYear | WeekOfYear | Season | Estimated_Solar_Power_kW | Estimate |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2006 | 1 | 1 | 0 | 0 | 0 | 0.108614 | 0 | 0 | 0 | ... | 0.15 | 778 | 0.9 | 85 | 0.459459 | 1 | 52 | Winter | 0.000000 | |
| **1** | 2006 | 1 | 1 | 1 | 0 | 0 | 0.093633 | 0 | 0 | 0 | ... | 0.15 | 779 | 1.0 | 89 | 0.459459 | 1 | 52 | Winter | 0.000000 | |
| **2** | 2006 | 1 | 1 | 2 | 0 | 0 | 0.082397 | 0 | 0 | 0 | ... | 0.15 | 779 | 1.0 | 92 | 0.459459 | 1 | 52 | Winter | 0.000000 | |
| **3** | 2006 | 1 | 1 | 3 | 0 | 0 | 0.112360 | 0 | 0 | 0 | ... | 0.15 | 780 | 1.0 | 94 | 0.567568 | 1 | 52 | Winter | 0.000000 | |
| **4** | 2006 | 1 | 1 | 4 | 36 | 484 | 0.254682 | 96 | 1 | 3 | ... | 0.15 | 780 | 1.0 | 97 | 0.945946 | 1 | 52 | Winter | 0.042763 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **13431** | 2022 | 1 | 31 | 19 | 0 | 0 | 0.363296 | 0 | 0 | 1 | ... | 0.14 | 779 | 1.5 | 121 | 0.081081 | 31 | 5 | Winter | 0.000000 | |
| **13432** | 2022 | 1 | 31 | 20 | 0 | 0 | 0.359551 | 0 | 0 | 1 | ... | 0.14 | 778 | 1.5 | 185 | 0.054054 | 31 | 5 | Winter | 0.000000 | |
| **13433** | 2022 | 1 | 31 | 21 | 0 | 0 | 0.348315 | 0 | 0 | 0 | ... | 0.14 | 777 | 1.5 | 211 | 0.081081 | 31 | 5 | Winter | 0.000000 | |
| **13434** | 2022 | 1 | 31 | 22 | 0 | 0 | 0.329588 | 0 | 0 | 3 | ... | 0.14 | 777 | 1.4 | 189 | 0.135135 | 31 | 5 | Winter | 0.000000 | |
| **13435** | 2022 | 1 | 31 | 23 | 0 | 0 | 0.303371 | 0 | 0 | 0 | ... | 0.14 | 777 | 1.4 | 162 | 0.162162 | 31 | 5 | Winter | 0.000000 | |

13436 rows × 27 columns

In [3]:
```python
# Display first few rows
print(df.head())
```

```
   Year  Month  Day  Hour  Clearsky DHI  Clearsky DNI  Temperature  \
0  2006      1    1     0             0             0     0.108614
1  2006      1    1     1             0             0     0.093633
2  2006      1    1     2             0             0     0.082397
3  2006      1    1     3             0             0     0.112360
4  2006      1    1     4            36           484     0.254682

   Clearsky GHI  cloud fill flag  Cloud Type  ...  Surface Albedo  Pressure  \
0             0                0           0  ...            0.15       778
1             0                0           0  ...            0.15       779
2             0                0           0  ...            0.15       779
3             0                0           0  ...            0.15       780
4            96                1           3  ...            0.15       780

   Precipitable Water  Wind Direction  Wind Speed  DayOfYear  WeekOfYear  \
0                 0.9              85    0.459459          1          52
1                 1.0              89    0.459459          1          52
2                 1.0              92    0.459459          1          52
3                 1.0              94    0.567568          1          52
4                 1.0              97    0.945946          1          52

   Season  Estimated_Solar_Power_kW  Estimated_Wind_Power_kW
0  Winter                  0.000000                 0.150461
1  Winter                  0.000000                 0.150461
2  Winter                  0.000000                 0.150461
3  Winter                  0.000000                 0.283618
4  Winter                  0.042763                 1.313047

[5 rows x 27 columns]
```

In [4]:
```
# Check for missing values
print(df.isnull().sum())
```

```
Year                        0
Month                       0
Day                         0
Hour                        0
Clearsky DHI                0
Clearsky DNI                0
Temperature                 0
Clearsky GHI                0
cloud fill flag             0
Cloud Type                  0
Dew Point                   0
DHI                         0
DNI                         0
Fill Flag                   0
GHI                         0
Relative Humidity           0
Solar Zenith Angle          0
Surface Albedo              0
Pressure                    0
Precipitable Water          0
Wind Direction              0
Wind Speed                  0
DayOfYear                   0
WeekOfYear                  0
Season                      0
Estimated_Solar_Power_kW    0
Estimated_Wind_Power_kW     0
dtype: int64
```

In [5]:
```
# Drop unnecessary columns (e.g., Fill Flags if they are not useful)
df = df.drop(columns=["Fill Flag", "cloud fill flag"], errors='ignore')
```

In [6]:
```
# Convert categorical features (Cloud Type) into numerical
df = pd.get_dummies(df, columns=["Cloud Type"], drop_first=True)

# Normalize relevant features
scaler = StandardScaler()
df[['Temperature', 'Wind Speed', 'Dew Point', 'Pressure', 'Precipitable Water']] = scaler.fit_transform(df[['Temperature', 'Wind Speed', 'Dew Point', 'Pressure', 'Precipitable
```

In [7]:
```
# Check preprocessed data
print(df.head())
```

```
   Year  Month  Day  Hour  Clearsky DHI  Clearsky DNI  Temperature  \
0  2006      1    1     0             0             0    -1.829209
1  2006      1    1     1             0             0    -1.920452
2  2006      1    1     2             0             0    -1.988885
3  2006      1    1     3             0             0    -1.806398
4  2006      1    1     4            36           484    -0.939585

   Clearsky GHI  Dew Point       DHI  ...  Estimated_Solar_Power_kW  \
0             0  -1.656762  0.000000  ...                  0.000000
1             0  -1.551803  0.000000  ...                  0.000000
2             0  -1.446844  0.000000  ...                  0.000000
3             0  -1.368124  0.000000  ...                  0.000000
4            96  -1.289405  0.063312  ...                  0.042763

   Estimated_Wind_Power_kW  Cloud Type_1  Cloud Type_3  Cloud Type_4  \
0                 0.150461         False         False         False
1                 0.150461         False         False         False
2                 0.150461         False         False         False
3                 0.283618         False         False         False
4                 1.313047         False          True         False

   Cloud Type_5  Cloud Type_6  Cloud Type_7  Cloud Type_8  Cloud Type_9
0         False         False         False         False         False
1         False         False         False         False         False
2         False         False         False         False         False
3         False         False         False         False         False
4         False         False         False         False         False

[5 rows x 32 columns]
```

### Feature Engineering

In [8]:
```
# Define target variable (solar energy output)
df['Solar Output'] = df['GHI'] + df['DNI'] + df['DHI']  # Approximate total solar energy output

# Define features
features = ['Temperature', 'Wind Speed', 'Dew Point', 'Pressure', 'Precipitable Water', 'Surface Albedo', 'Relative Humidity', 'Cloud Type_3']
X = df[features]
y = df['Solar Output']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

### Train a Model to Predict Solar Energy Output

In [9]:
```
import numpy as np
import pandas as pd
from sklearn.ensemble import StackingRegressor
from sklearn.linear_model import Ridge
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.neural_network import MLPRegressor
```

In [10]:
```
# ---- Load & Prepare Data ----
features = ['Temperature', 'DNI', 'DHI', 'GHI', 'Relative Humidity', 'Wind Speed', 'Wind Direction', 'Surface Albedo', 'Pressure']
X = df[features]
y = df['Clearsky GHI']  # Target variable

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [11]:
```
# ---- Step 3.1: Train Individual Models ----
xgb = XGBRegressor(n_estimators=500, learning_rate=0.03, max_depth=10, subsample=0.9, colsample_bytree=0.8, random_state=42)
lgbm = LGBMRegressor(n_estimators=500, learning_rate=0.03, max_depth=10, subsample=0.9, colsample_bytree=0.8, random_state=42)
rf = RandomForestRegressor(n_estimators=500, max_depth=12, random_state=42)
```

In [12]:
```
# ---- Step 3.2: Stacking Model (Ensemble Learning) ----
stacking_model = StackingRegressor(
    estimators=[('xgb', xgb), ('lgbm', lgbm), ('rf', rf)],
    final_estimator=Ridge(alpha=1.0)  # Ridge Regression as meta-model
)

stacking_model.fit(X_train, y_train)
y_pred_stack = stacking_model.predict(X_test)
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

In [13]:
```python
# ---- Step 3.3: Deep Learning Model (MLP) ----
mlp = MLPRegressor(hidden_layer_sizes=(256, 128, 64), activation='relu', solver='adam',
                   alpha=0.0005, learning_rate_init=0.005, max_iter=1200, batch_size=32,
                   early_stopping=True, validation_fraction=0.1, random_state=42)

mlp.fit(X_train, y_train)
y_pred_mlp = mlp.predict(X_test)
```

In [14]:
```python
# ---- Step 3.4: Model Evaluation ----
def evaluate_model(y_test, y_pred, model_name):
    mae = mean_absolute_error(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)
    print(f"{model_name} -> MAE: {mae:.4f}, RMSE: {rmse:.4f}, R² Score: {r2:.4f}")
    return r2

r2_stack = evaluate_model(y_test, y_pred_stack, "Stacking Model")
r2_mlp = evaluate_model(y_test, y_pred_mlp, "Deep Learning MLP")

# ---- Step 3.5: Best Model Selection ----
best_model = max([(r2_stack, stacking_model), (r2_mlp, mlp)], key=lambda x: x[0])[1]
print("🏆 Best Model Selected:", type(best_model).__name__)
```

```
Stacking Model -> MAE: 20.1006, RMSE: 50.3195, R² Score: 0.9746
Deep Learning MLP -> MAE: 30.1546, RMSE: 68.5557, R² Score: 0.9529
🏆 Best Model Selected: StackingRegressor
```

In [ ]:

### Optimize Energy Storage & Grid Distribution

In [45]:
```python
!pip install pulp
```

```
   -------------------------------------- - 16.9/17.7 MB 1.3 MB/s eta 0:00:01
   -------------------------------------- - 17.0/17.7 MB 1.3 MB/s eta 0:00:01
   -------------------------------------- - 17.0/17.7 MB 1.3 MB/s eta 0:00:01
   -------------------------------------- - 17.1/17.7 MB 1.3 MB/s eta 0:00:01
   -------------------------------------- - 17.2/17.7 MB 1.3 MB/s eta 0:00:01
   -------------------------------------- - 17.2/17.7 MB 1.3 MB/s eta 0:00:01
   ---------------------------------------  17.3/17.7 MB 1.3 MB/s eta 0:00:01
   ---------------------------------------  17.3/17.7 MB 1.3 MB/s eta 0:00:01
   ---------------------------------------  17.4/17.7 MB 1.3 MB/s eta 0:00:01
   ---------------------------------------  17.5/17.7 MB 1.3 MB/s eta 0:00:01
   ---------------------------------------  17.5/17.7 MB 1.3 MB/s eta 0:00:01
   ---------------------------------------  17.6/17.7 MB 1.3 MB/s eta 0:00:01
   ---------------------------------------  17.6/17.7 MB 1.3 MB/s eta 0:00:01
   ---------------------------------------  17.7/17.7 MB 1.3 MB/s eta 0:00:01
   ---------------------------------------  17.7/17.7 MB 1.3 MB/s eta 0:00:01
   ---------------------------------------  17.7/17.7 MB 1.3 MB/s eta 0:00:01
   ---------------------------------------- 17.7/17.7 MB 1.3 MB/s eta 0:00:00
Installing collected packages: pulp
Successfully installed pulp-3.0.2
```

In [61]:
```python
import pandas as pd
from pulp import LpMaximize, LpProblem, LpVariable
from pulp import value
```

In [47]:
```
# Load the dataset
df = pd.read_csv("D:/Capstone_2025/code/preprocessed_data.csv")  # Replace with actual dataset path
```

In [48]:
```
# ---- Step 5.1: Define Parameters ----
storage_capacity = 500  # Maximum storage capacity in kWh
grid_demand = 300  # Example demand from the grid in kWh

# Extract relevant features
df['Total_Energy_Generated'] = df['Estimated_Solar_Power_kW'] + df['Estimated_Wind_Power_kW']
```

In [49]:
```
# ---- Step 5.2: Define Optimization Model ----
model = LpProblem("Energy_Storage_and_Distribution", LpMaximize)

# Decision Variables
solar_to_storage = LpVariable("Solar_to_Storage", lowBound=0)
solar_to_grid = LpVariable("Solar_to_Grid", lowBound=0)
wind_to_storage = LpVariable("Wind_to_Storage", lowBound=0)
wind_to_grid = LpVariable("Wind_to_Grid", lowBound=0)
```

In [50]:
```
# ---- Step 5.3: Objective Function (Maximize Stored & Grid Distributed Energy) ----
model += (solar_to_storage + wind_to_storage + solar_to_grid + wind_to_grid), "Maximize Energy Usage"
```

In [51]:
```
# ---- Step 5.4: Constraints ----
# 1️⃣ Total allocated energy should not exceed generated energy
model += solar_to_storage + solar_to_grid <= df['Estimated_Solar_Power_kW'].mean(), "Solar Energy Limit"
model += wind_to_storage + wind_to_grid <= df['Estimated_Wind_Power_kW'].mean(), "Wind Energy Limit"

# 2️⃣ Storage should not exceed capacity
model += solar_to_storage + wind_to_storage <= storage_capacity, "Storage Capacity Constraint"

# 3️⃣ Grid demand should be met
model += solar_to_grid + wind_to_grid >= grid_demand, "Grid Demand Constraint"
```

In [52]:
```
# ---- Step 5.5: Solve the Optimization Problem ----
model.solve()
```

Out[52]:
```
-1
```

In [53]:
```
# ---- Step 5.6: Print Results ----
print(f"Optimal Solar to Storage: {solar_to_storage.varValue} kWh")
print(f"Optimal Solar to Grid: {solar_to_grid.varValue} kWh")
print(f"Optimal Wind to Storage: {wind_to_storage.varValue} kWh")
print(f"Optimal Wind to Grid: {wind_to_grid.varValue} kWh")
```

```
Optimal Solar to Storage: 0.0 kWh
Optimal Solar to Grid: 0.11084681 kWh
Optimal Wind to Storage: 0.0 kWh
Optimal Wind to Grid: 299.88915 kWh
```

### Visualisation

In [58]:
```
# ---- Step 1: Define Parameters ----
storage_capacity = 1000  # Battery capacity (kWh)
grid_demand = 250  # Adjusted demand
battery_efficiency = 0.9  # 90% efficiency
solar_generated = 0.11084681  # From optimization output
wind_generated = 299.88915  # From optimization output
```

In [59]:
```
# ---- Step 2: Define Optimization Model ----
model = LpProblem("Energy_Storage_and_Distribution", LpMaximize)

# Decision Variables
solar_to_storage = LpVariable("Solar_to_Storage", lowBound=0)
solar_to_grid = LpVariable("Solar_to_Grid", lowBound=0)
wind_to_storage = LpVariable("Wind_to_Storage", lowBound=0)
wind_to_grid = LpVariable("Wind_to_Grid", lowBound=0)

# Objective Function (Maximize Energy Usage)
model += (
    solar_to_storage * battery_efficiency +
    wind_to_storage * battery_efficiency +
    solar_to_grid +
    wind_to_grid
), "Maximize Energy Usage"

# Constraints
model += solar_to_storage + solar_to_grid <= solar_generated, "Solar Limit"
model += wind_to_storage + wind_to_grid <= wind_generated, "Wind Limit"
model += solar_to_storage + wind_to_storage <= storage_capacity, "Storage Capacity"
model += solar_to_grid + wind_to_grid >= grid_demand, "Grid Demand"

# Solve the model
model.solve()
```

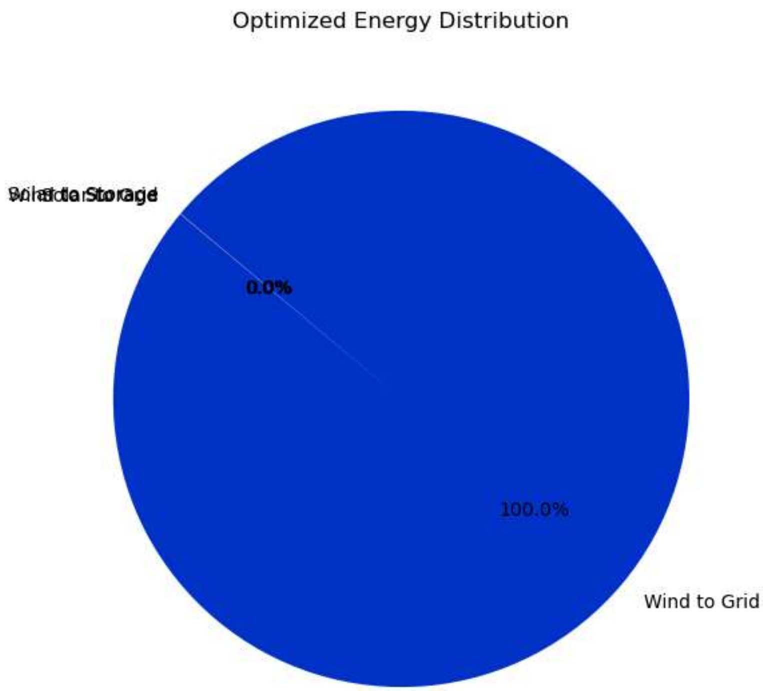
Out[59]: 1

In [62]:
```
# Extract results
solar_to_storage_value = value(solar_to_storage)
solar_to_grid_value = value(solar_to_grid)
wind_to_storage_value = value(wind_to_storage)
wind_to_grid_value = value(wind_to_grid)
```

In [63]:

```
# ---- Step 3: Visualization ----
labels = ["Solar to Storage", "Solar to Grid", "Wind to Storage", "Wind to Grid"]
values = [solar_to_storage_value, solar_to_grid_value, wind_to_storage_value, wind_to_grid_value]

# Plot a Pie Chart
plt.figure(figsize=(7, 7))
colors = ["#ffcc00", "#ff6600", "#0099ff", "#0033cc"]
plt.pie(values, labels=labels, autopct='%1.1f%%', colors=colors, startangle=140)
plt.title("Optimized Energy Distribution")
plt.show()
```




**Visualisation**

In [66]:

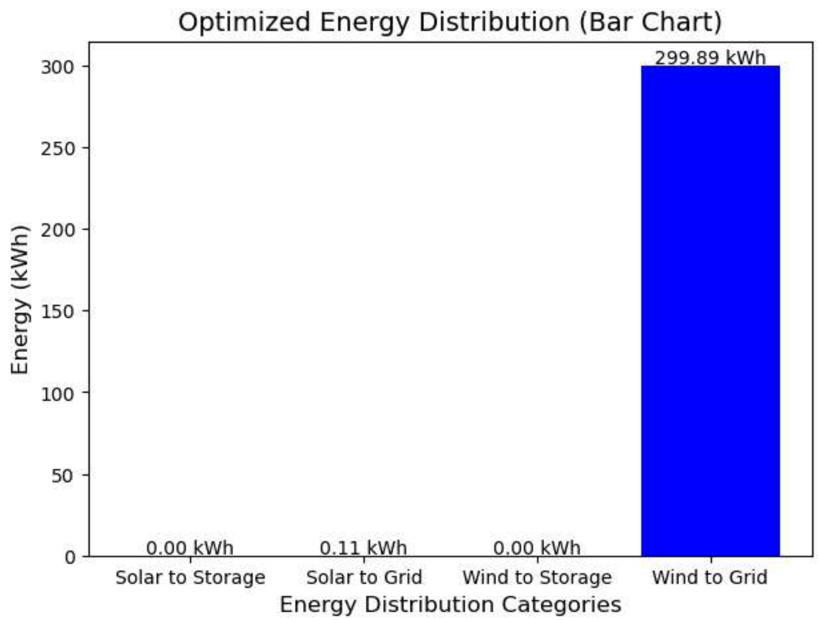
```
# Optimized energy values
labels = ['Solar to Storage', 'Solar to Grid', 'Wind to Storage', 'Wind to Grid']
values = [0.0, 0.11084681, 0.0, 299.88915]

# Define colors for better distinction
colors = ['gold', 'orange', 'skyblue', 'blue']
```

In [67]:
```python
# Create a bar chart for better readability
plt.figure(figsize=(7, 5))
plt.bar(labels, values, color=colors)
plt.xlabel("Energy Distribution Categories", fontsize=12)
plt.ylabel("Energy (kWh)", fontsize=12)
plt.title("Optimized Energy Distribution (Bar Chart)", fontsize=14)

# Show values on bars
for i, v in enumerate(values):
    plt.text(i, v + 1, f"{v:.2f} kWh", ha='center', fontsize=10)

# Show plot
plt.show()
```




In [ ]:

In [ ]:

In [ ]:

In [ ]: