SAVIORS OF THE WORLD

DATATHON

YUPENG CHEN, ROBERTO CEDENO, SOFIA DEPOORTERE, ALESSIO MASTROPIETRO,
SRAVAN SRIDHAR, JOSEPH CLERC

CONTENT

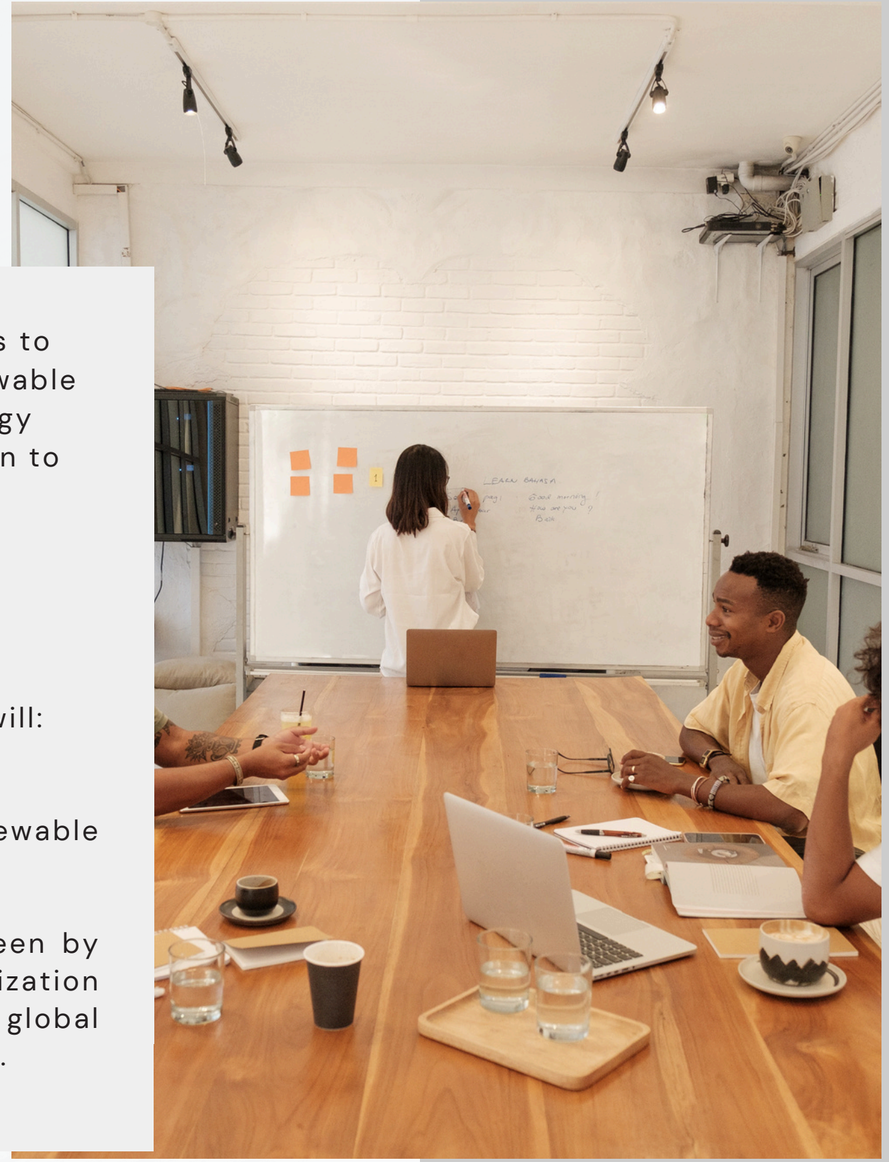
PROJECT DESCRIPTION

In response to the global energy transition, our project aims to address the challenges posed by the intermittency of renewable energy sources like wind and solar. Leveraging Battery Energy Storage Systems (BESS), we are developing a robust solution to optimize energy trading strategies in the United Kingdom's electricity market.

By producing highly accurate market forecasts, our system will:

- Maximize the profitability of renewable energy trading.
- Enhance energy security and grid stability.
- Reduce reliance on fossil fuels while preventing renewable curtailment.

This project aligns with EDP's mission to become 100% green by 2030, combining advanced forecasting models, optimization algorithms, and sustainable energy practices to lead the global energy transition and support a cleaner, more reliable future.



CONTENT

DATA CLEANSING

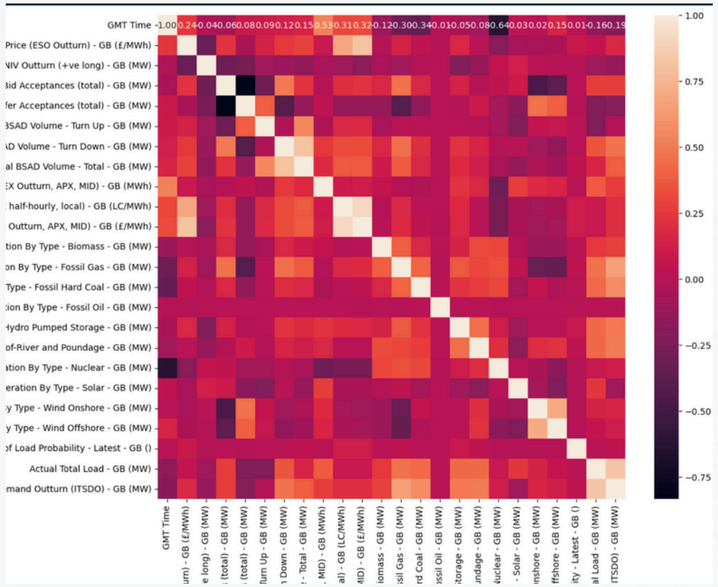
NUMERICAL VARIABLES	A KNN Algorithm to impute missing values for numerical variables
CATEGORICAL VARIABLES	Mode imputation orforward-fill/backward-fill based on context for categorical variables
NAN VALUES	We removed all the remaining rows that contained NaN

CONTENT

EDA(1)

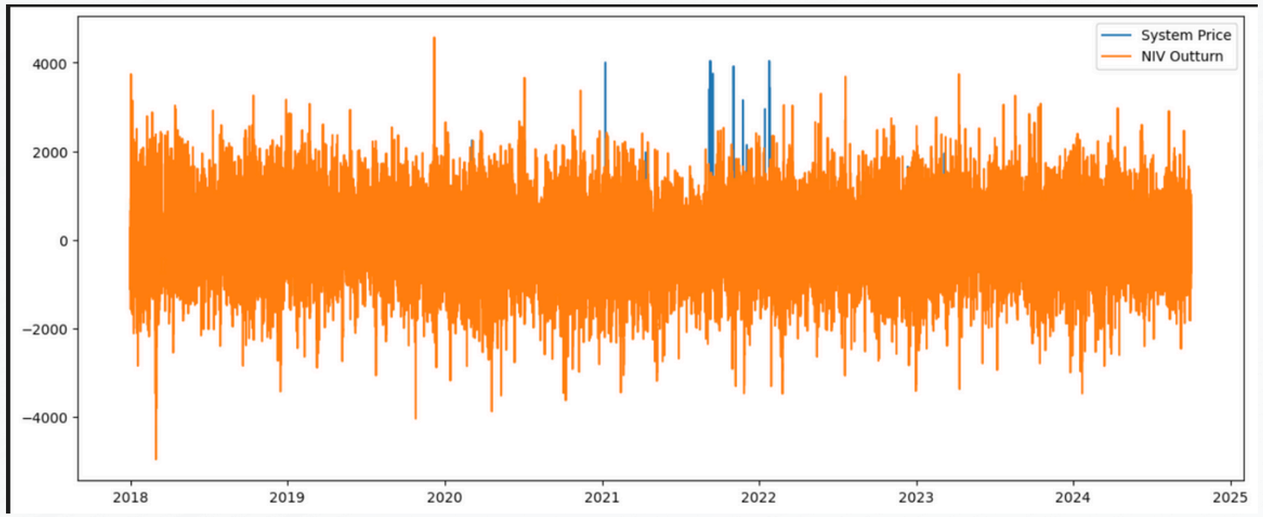
CORRELATION MATRIX

This heatmap shows the correlation matrix for various energy-related metrics, Clusters of variables, such as electricity pricing and demand, show high mutual correlations, suggesting interdependencies. Conversely, renewable energy generation (e.g., solar, wind) might exhibit negative correlations with fossil-based generation, highlighting contrasting patterns.



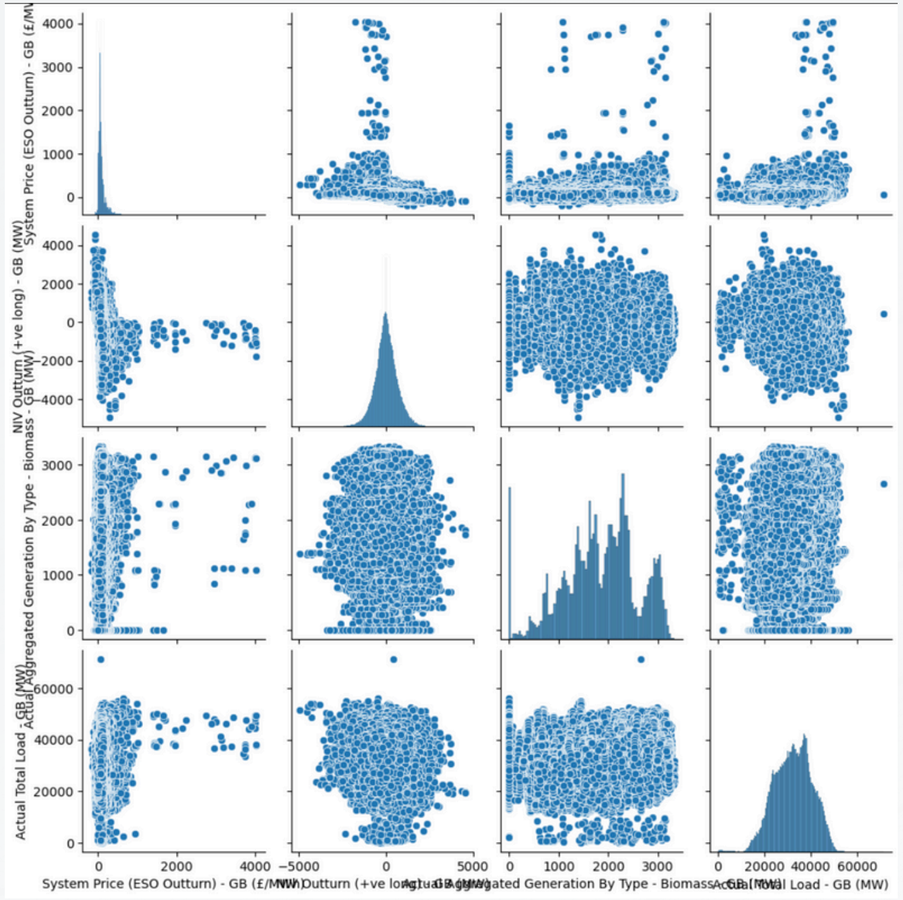
TIME SERIES PLOT

This line chart compares the trends of System Price and NIV Outturn over time from 2018 to 2025. The NIV Outturn (in orange) exhibits high variability, fluctuating significantly both positively and negatively, while the System Price (in blue) shows more sporadic appearances, indicating its data might be intermittent or only relevant in specific periods. The chart highlights the volatility of energy-related metrics over the years.

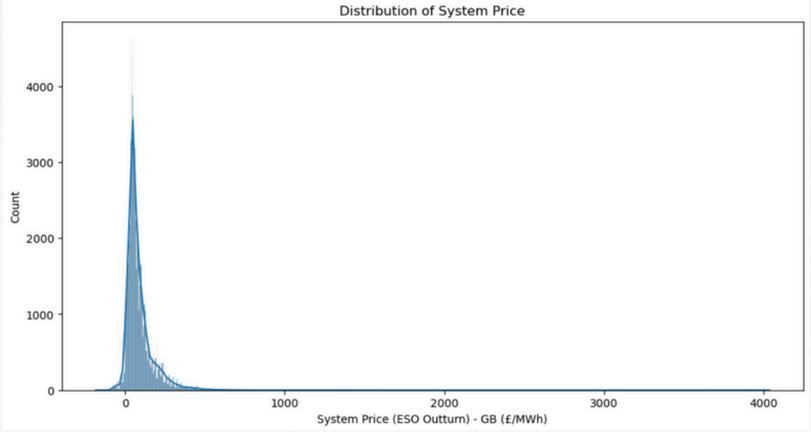


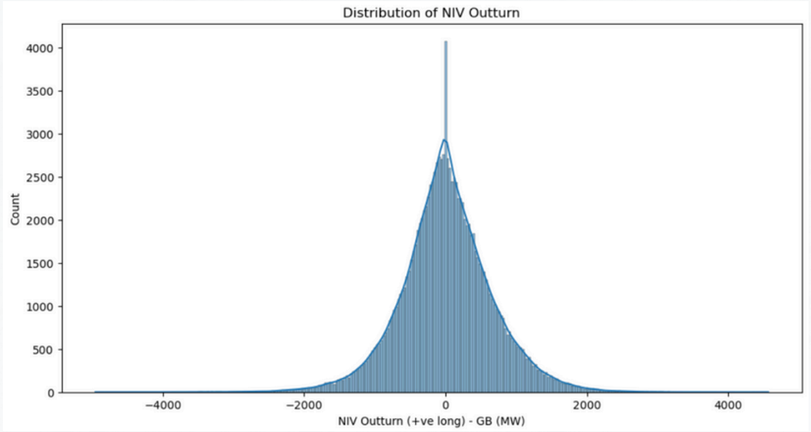
EDA(2)

PAIR PLOT



DISTRIBUTION PLOTS





This visualization combines pair plots and individual distributions of key energy variables. The pair plots on the left reveal relationships and clustering between variables, with certain metrics showing clearer patterns of association, such as between total load and specific generation types. The right-hand histograms highlight the distributions of System Price and NIV Outturn, where the System Price is positively skewed, indicating a concentration of lower values, while NIV Outturn shows a symmetric distribution centered around zero, reflecting its fluctuating nature. These insights can guide further analysis on dependencies and variability.

CONTENT

FEATURE ENGINEERING

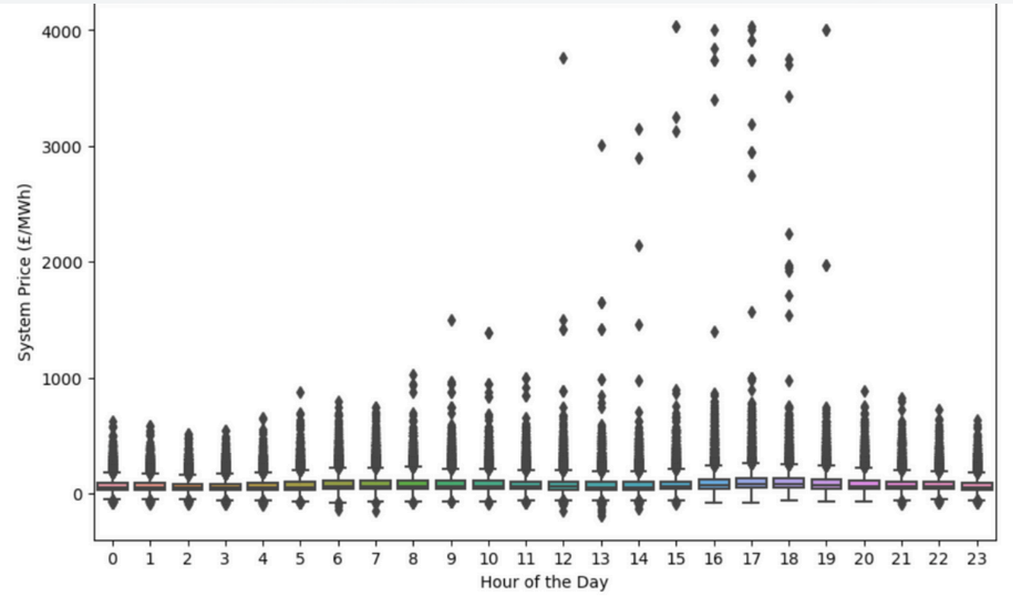
This code snippet demonstrates feature engineering for time-series data. It creates lag features (**System Price Lag1** and **NIV Outturn Lag1**) to capture the previous time step's values, and rolling statistics (**System Price Rolling Mean** and **NIV Outturn Rolling Mean**) to smooth trends over a 3-time step window. Additionally, it generates time-based features (**Hour**, **Day**, **Month**, and **Day of Week**) to capture temporal patterns. Rows with NaN values from lagging and rolling operations are dropped to ensure data integrity, and the modified dataset is previewed with data.head().

```
# Feature Engineering

# Create lag features
data['System Price Lag1'] = data['System Price (ESO Outturn) - GB (£/MWh)'].shift(1)
data['NIV Outturn Lag1'] = data['NIV Outturn (+ve long) - GB (MW)'].shift(1)

# Create rolling statistics features
data['System Price Rolling Mean'] = data['System Price (ESO Outturn) - GB (£/MWh)'].rolling(window=3).mean()
data['NIV Outturn Rolling Mean'] = data['NIV Outturn (+ve long) - GB (MW)'].rolling(window=3).mean()

# Create time-based features
data['Hour'] = data['GMT Time'].dt.hour
data['Day'] = data['GMT Time'].dt.day
data['Month'] = data['GMT Time'].dt.month
data['Day of Week'] = data['GMT Time'].dt.dayofweek

# Drop rows with NaN values created by lag and rolling features
data.dropna(inplace=True)

# Display the first few rows of the dataset with new features
data.head()
```

CONTENT

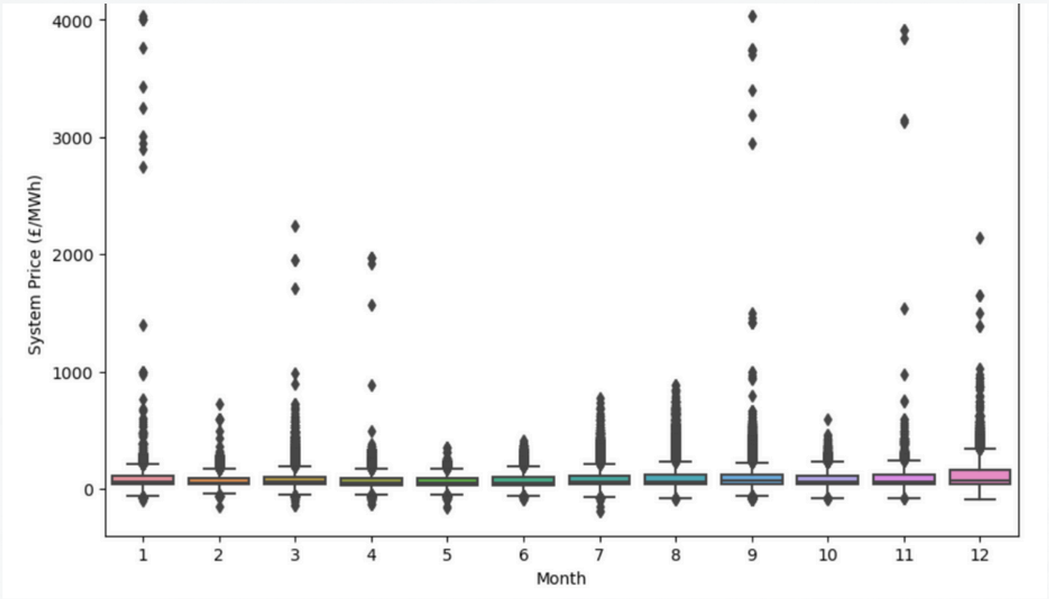
SYSTEM PRICE ANALYSIS

AVERAGE SYSTEM PRICE BY HOUR OF THE DAY



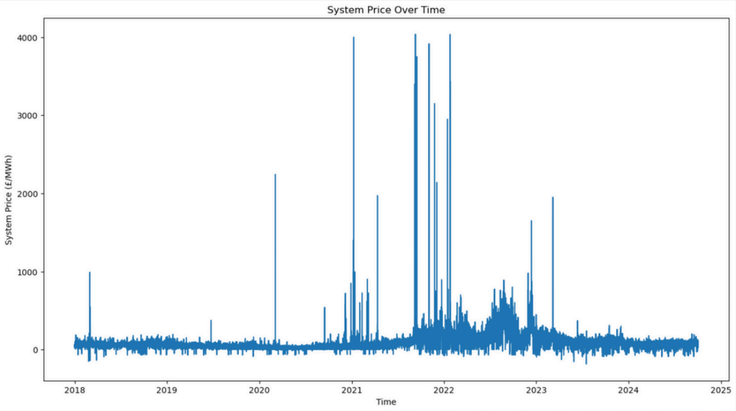
The boxplot shows that System Price tends to spike significantly **between 14:00 and 18:00**, indicating higher volatility and **extreme values during these hours**, likely due to peak demand. Prices are relatively stable during other hours, with lower averages and fewer outliers.

AVERAGE SYSTEM PRICE BY MONTH

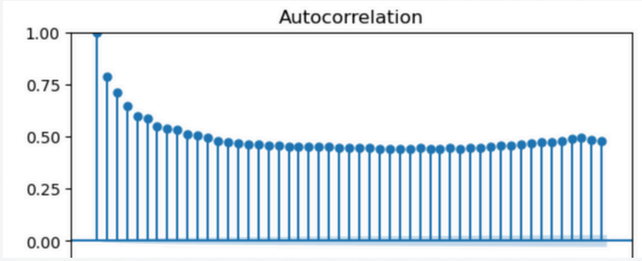


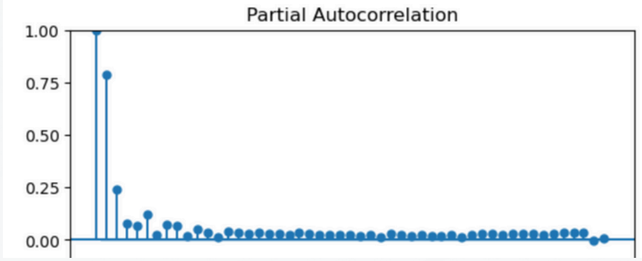
The boxplot shows that System Price experiences significant spikes **in January and December**, likely due to seasonal demand increases during winter months. Prices remain relatively stable throughout the rest of the year, with fewer outliers and lower averages.

SEASONALITY ANALYIS




```
ADF Statistic: -13.11337639227544
p-value: 1.6176204491208921e-24
```






```
                                SARIMAX Results
==========================================================================================
Dep. Variable:     System Price (ESO Outturn) - GB (£/MWh)   No. Observations:       118320
Model:                                     ARIMA(0, 0, 1)   Log Likelihood    -686306.755
Date:                                    Wed, 04 Dec 2024   AIC              1372619.511
Time:                                            01:32:09   BIC              1372648.554
Sample:                                                 0   HQIC             1372628.258
                                                   - 118320
Covariance Type:                                       opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         87.6331      0.467    187.723      0.000      86.718      88.548
ma.L1          0.5755      0.000   3119.750      0.000       0.575       0.576
sigma2      6393.1380      2.042   3131.060      0.000    6389.136    6397.140
===================================================================================
Ljung-Box (L1) (Q):                9741.50   Jarque-Bera (JB):        1752046450.28
Prob(Q):                              0.00   Prob(JB):                        0.00
Heteroskedasticity (H):               5.00   Skew:                           13.88
Prob(H) (two-sided):                  0.00   Kurtosis:                      598.49
===================================================================================
```

The System Price over time shows significant volatility with sharp spikes, particularly around 2021-2022, reflecting extreme price fluctuations. The ADF test results (ADF Statistic: -13.11, p-value: ~0) indicate that the data **is stationary**, meaning it is suitable for modeling without further differencing.
The Autocorrelation (ACF) plot reveals high positive correlations at initial lags, gradually decreasing over time, suggesting strong temporal dependencies in the data. **The Partial Autocorrelation (PACF)** plot shows a sharp drop after the first lag, indicating that most of the explanatory power comes from the immediately preceding value. That indicates that **we can use an ARIMA model**.

CONTENT

PREPROCESSING STEPS

- Adding lag features and rolling statistics data on 'system_price_eso_outturn_gb_mwh', 'niv_outturn_ve_long_gb_mw'

```
print("Adding lag features...")
lag_features = ['system_price_eso_outturn_gb_mwh', 'niv_outturn_ve_long_gb_mw']
for lag in [1, 24, 48]:  # Lag by 1 hour, 1 day, 2 days
    for col in lag_features:
        if col in merged_data.columns:
            merged_data[f'{col}_lag_{lag}'] = merged_data[col].shift(lag)
        else:
            print(f"Warning: Column {col} not found in merged_data.")
```

```
print("Adding rolling statistics...")
for window in [7, 30]:  # Weekly and monthly
    for col in lag_features:
        if col in merged_data.columns:
            merged_data[f'{col}_rolling_mean_{window}'] = merged_data[col].rolling(window).mean()
        else:
            print(f"Warning: Column {col} not found in merged_data.")
```

ML MODELS

- **Linear Regression**, **Ridge**, and **Lasso** for baseline comparisons.
- **Random Forest Regressor** and **Gradient Boosting Regressor** for capturing nonlinear relationships.
- **LightGBM** and **AdaBoost Regressor** for efficient handling of large datasets and fine-grained performance optimization.
- **Stacking Regressor** and **Voting Regressor** for combining the strengths of multiple models to improve overall accuracy.

```python
class EnhancedEnsembleModel:
    def __init__(self, target_col):
        self.target_col = target_col
        self.scaler = StandardScaler()
        self.models = {
            'rf': RandomForestRegressor(random_state=42),
            'gbm': GradientBoostingRegressor(random_state=42),
            'lgbm': LGBMRegressor(random_state=42)
        }
        self.best_models = {}

    def optimize_model(self, model_name, X_train, y_train):
        """Optimize hyperparameters using Optuna"""
        def objective(trial):
            if model_name == 'rf':
                params = {
                    'n_estimators': trial.suggest_int('n_estimators', 100, 500),
                    'max_depth': trial.suggest_int('max_depth', 5, 30),
                    'min_samples_split': trial.suggest_int('min_samples_split', 2, 10)
                }
            elif model_name == 'gbm':
                params = {
                    'n_estimators': trial.suggest_int('n_estimators', 100, 500),
                    'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.1),
                    'max_depth': trial.suggest_int('max_depth', 3, 10)
                }
            else:  # lgbm
                params = {
                    'n_estimators': trial.suggest_int('n_estimators', 100, 500),
                    'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.1),
                    'num_leaves': trial.suggest_int('num_leaves', 20, 100)
                }
```

BASE MODEL

DEFINE THE MODELS WERE GOING TO RUN BOTH TARGET VARIABLES AND TAKING THE BEST MODEL FOR EACH

```
print("Defining base models...")
base_models = [
    ('rf', RandomForestRegressor(random_state=42)),
    ('gb', GradientBoostingRegressor(random_state=42)),
    ('xgb', XGBRegressor(random_state=42, objective='reg:squarederror')),
    ('lgbm', LGBMRegressor(random_state=42))
]

grid_search_price = GridSearchCV(
    stacking_regressor_price,
    param_grid_price,
    cv=tscv,
    scoring='neg_root_mean_squared_error',
    verbose=3
)
grid_search_price.fit(X, y_price)
best_model_price = grid_search_price.best_estimator_
print(f"Best model for System Price: {grid_search_price.best_params_}")

grid_search_niv = GridSearchCV(
    stacking_regressor_niv,
    param_grid_niv,
    cv=tscv,
    scoring='neg_root_mean_squared_error',
    verbose=3
)
grid_search_niv.fit(X, y_niv)
best_model_niv = grid_search_niv.best_estimator_
print(f"Best model for NIV: {grid_search_niv.best_params_}")
```

BEST MODEL

THE STACKING REGRESSOR CONSISTENTLY OUTPERFORMED OTHER ALGORITHMS IN BOTH PREDICTIVE ACCURACY AND COMPUTATIONAL EFFICIENCY.

RMSE SCORE FOR SYSTEM PRICE

28.4731

RMSE SCORE FOR NIV

643.6186

CONTENT

STRATEGIES

STRATEGY 1

PEAK SHAVING (peak_shaving_strategy)

Use BESS to store energy during periods of low system price and discharge during peak price periods to reduce costs.

1. Identify periods with the highest predicted system prices.
2. Charge BESS during periods with low predicted system prices.
3. Discharge BESS during periods with high predicted system prices to reduce energy costs.

STRATEGY 2

ARBITRAGE (arbitrage_strategy)

Take advantage of price differences between low and high price periods to generate revenue.

1. Monitor predicted system prices to identify opportunities for buying low and selling high.
2. Charge BESS when predicted system prices are low.
3. Discharge BESS when predicted system prices are high to maximize revenue.

STRATEGY 3

GRID SUPPORT (grid_support_strategy)

Use BESS to provide grid support services such as frequency regulation and reserve capacity.

1. Use BESS to provide frequency regulation by charging and discharging to maintain grid stability.
2. Offer reserve capacity by keeping BESS charged and ready to discharge during grid emergencies.
3. Participate in grid support programs to earn additional revenue.

STRATEGIES

STRATEGY 4

Demand Response
demand_response_strategy

Use BESS to participate in demand response programs to reduce peak demand and earn incentives.

1. Enroll in demand response programs offered by grid operators.
2. Use BESS to reduce peak demand during demand response events.
3. Earn incentives and reduce energy costs by participating in demand response programs.

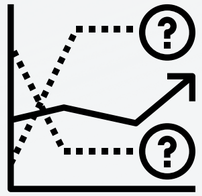
STRATEGY 5

Renewable Energy Integration
(renewable_integration_strategy)

Use BESS to store excess renewable energy and reduce reliance on fossil fuels.

1. Store excess energy generated from renewable sources such as solar and wind.
2. Discharge stored renewable energy during periods of high demand to reduce reliance on fossil fuels.
3. Enhance the use of renewable energy and contribute to environmental sustainability.

CONCLUSIONS



It is possible to produce **accurate forecasts** for the electricity prices and system imbalances, this will allow BESS to capitalize on opportunities to **generate higher profits.**

It is important to notice that **seasonal trends** provided valuable insights into the impact of weather variability and **patterns on energy market dynamics.**

The models developed were not **only scalable but also interpretable**, making them well-suited for **real-time** energy trading and storage management applications.

OUR STREAMLIT APP

HTTPS://APP-3CUEZMCPNN4EXZODQKCUCT.STREAMLIT.APP/