Machine Learning Project 2025-26

Name:

Kamble Ritesh Dattu (EN24207143)

Kokare Tushar Pandurang (EN24207144)

Kanherkar Amol Sanjay (EN23107056)

Problem Statment :Energy Consumption Forecasting for Smart Homes

In [142…
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import IsolationForest
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

In [66]:
```
df = pd.read_csv("electricity_bill_dataset.csv")
```

In [67]:
```
df
```

Out[67]:

	Fan	Refrigerator	AirConditioner	Television	Monitor	MotorPump	Mont
0	16	23.0	2.0	6.0	1.0	0	1
1	19	22.0	2.0	3.0	1.0	0	
2	7	20.0	2.0	6.0	7.0	0	
3	7	22.0	3.0	21.0	1.0	0	
4	11	23.0	2.0	11.0	1.0	0	
...	...	...	...	...	...	...	
45340	18	22.0	3.0	22.0	1.0	0	
45341	23	23.0	2.0	6.0	12.0	0	
45342	22	22.0	2.0	20.0	1.0	0	
45343	8	21.0	2.0	22.0	7.0	0	1
45344	8	17.0	2.0	4.0	1.0	0	

45345 rows × 12 columns

Data Pre_proc

In [68]:
```
df.isnull().sum()
```

Out[68]:
```
Fan                0
Refrigerator       0
AirConditioner     0
Television         0
Monitor            0
MotorPump          0
Month              0
City               0
Company            0
MonthlyHours       0
TariffRate         0
ElectricityBill    0
dtype: int64
```

In [69]:
```
df = df.dropna()
print(df.isnull().sum())
```

```
Fan                0
Refrigerator       0
AirConditioner     0
Television         0
Monitor            0
MotorPump          0
Month              0
City               0
Company            0
MonthlyHours       0
TariffRate         0
ElectricityBill    0
dtype: int64
```

In [70]:
```
df.describe()                                    # Summary statistics for nume
```

Out[70]:

	Fan	Refrigerator	AirConditioner	Television	Monitor
count	45345.000000	45345.000000	45345.000000	45345.000000	45345.000000
mean	13.990694	21.705458	1.503959	12.502635	2.865057
std	5.470816	1.672575	1.115482	5.756007	3.894933
min	5.000000	17.000000	0.000000	3.000000	1.000000
25%	9.000000	22.000000	1.000000	7.000000	1.000000
50%	14.000000	22.000000	2.000000	13.000000	1.000000
75%	19.000000	23.000000	2.000000	17.000000	1.000000
max	23.000000	23.000000	3.000000	22.000000	12.000000

In [71]:
```
(df.dtypes)                                      # Show data types of each co
```

Out[71]:
```
Fan                 int64
Refrigerator      float64
AirConditioner    float64
Television        float64
Monitor           float64
MotorPump           int64
Month               int64
City               object
Company            object
MonthlyHours        int64
TariffRate        float64
ElectricityBill   float64
dtype: object
```

In [72]:
```
(df.shape)                                    # Output (rows, columns)
```

Out[72]:
```
(45345, 12)
```

In [73]:
```
(df.sort_index().head())
```

Out[73]:

	Fan	Refrigerator	AirConditioner	Television	Monitor	MotorPump	Month	
0	16	23.0	2.0	6.0	1.0	0	10	H
1	19	22.0	2.0	3.0	1.0	0	5	
2	7	20.0	2.0	6.0	7.0	0	7	
3	7	22.0	3.0	21.0	1.0	0	6	
4	11	23.0	2.0	11.0	1.0	0	2	

Liner Regre

In [83]:
```
# Independent Variables (X)
X = df[["Fan", "Refrigerator", "AirConditioner", "Television",
        "Monitor", "MotorPump", "Month", "MonthlyHours", "TariffRate"]]
```

In [85]:
```
# Dependent Variable
y = df["ElectricityBill"]
```

In [86]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc
```

In [87]:
```
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[87]:
```
LinearRegression()
```

In [88]:
```
y_pred = model.predict(X_test)
```

In [90]:
```
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Model Evaluation")
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R² Score:", r2)
```

```
Model Evaluation
Mean Squared Error (MSE): 4969.72818804717
Root Mean Squared Error (RMSE): 70.49629910886932
R² Score: 0.9956383663641158
```

In [97]:
```
# Create a DataFrame with Actual vs Predicted values
results = pd.DataFrame({
    "Actual": y_test.values,
    "Predicted": y_pred
})

# Show first 10 rows
print(results.head(10))
```

```
   Actual  Predicted
0  5054.7   5056.266
1  3809.2   3809.467
2  3245.3   3246.960
3  5728.8   5729.472
4  4148.0   4148.255
5  3645.6   3645.768
6  2317.7   2320.542
7  4100.8   4101.240
8  2286.9   2285.908
9  5394.4   5394.224
```

In [98]:
```
results.to_csv("linear_regression_predictions.csv", index=False)
print("Predictions saved to 'linear_regression_predictions.csv'")
```

```
Predictions saved to 'linear_regression_predictions.csv'
```

Random_fore

In [91]:
```
# Step 1 : Split Data into Training & Testing Sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

In [92]:
```
# Step 2: Build & Train Random Forest Regressor
rf_model = RandomForestRegressor(
    n_estimators=100,       # number of trees
    random_state=42,
    max_depth=None,          # grow trees fully
    min_samples_split=2      # minimum samples to split
)
rf_model.fit(X_train, y_train)
```

Out[92]:

RandomForestRegressor

RandomForestRegressor(random_state=42)

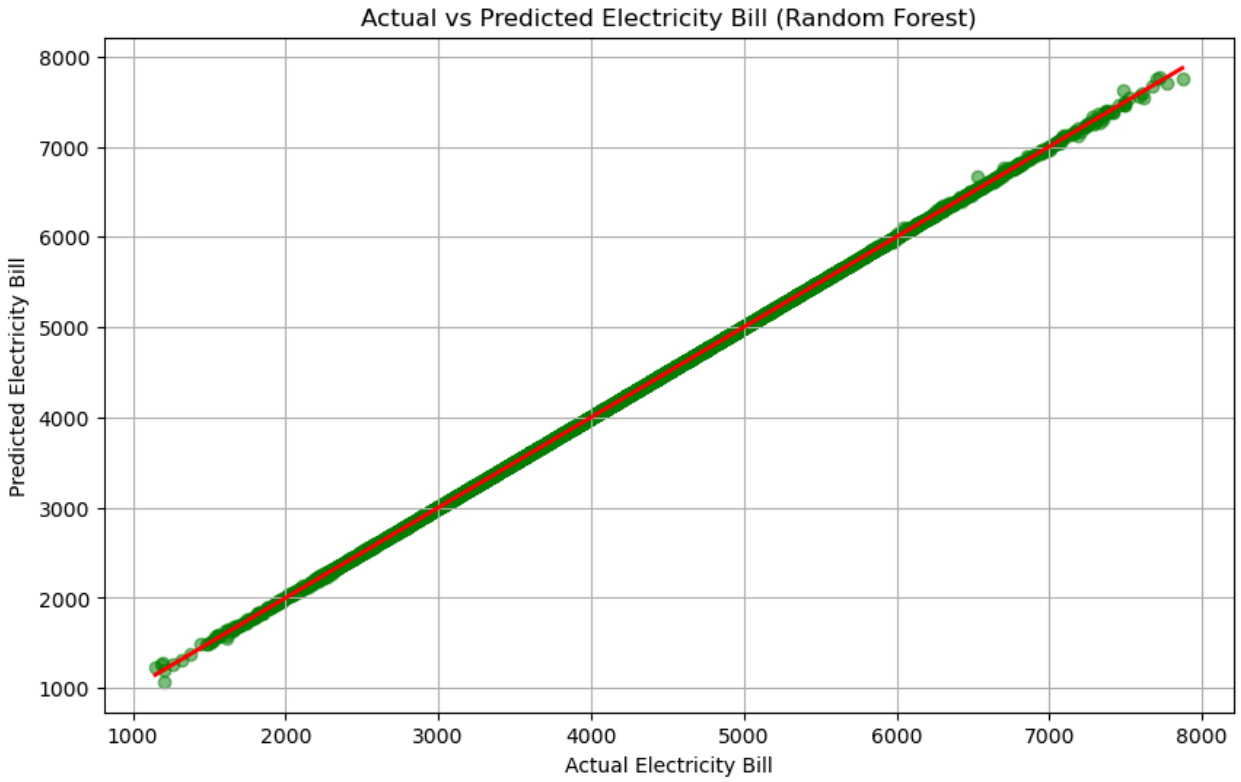
In [93]:
```
# Step 3: Make Predictions
y_pred = rf_model.predict(X_test)
```

In [94]:
```
# Step 4: Model Evaluation
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
```

In [95]:
```
print("📌 Random Forest Model Evaluation")
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("R² Score:", r2)
```

```
📌 Random Forest Model Evaluation
Mean Squared Error (MSE): 26.47928567703212
Root Mean Squared Error (RMSE): 5.145802724262962
R² Score: 0.999976760712318
```

In [96]:
```
 # Actual vs Predicted Bills
plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred, color='green', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='re
plt.xlabel("Actual Electricity Bill")
plt.ylabel("Predicted Electricity Bill")
plt.title("Actual vs Predicted Electricity Bill (Random Forest)")
plt.grid(True)
plt.show()
```



In [99]:
```
# Create DataFrame for Actual vs Predicted values
rf_results = pd.DataFrame({
    "Actual": y_test.values,
    "Predicted": y_pred
})

# Show first 10 records
print(rf_results.head(10))
```

```
   Actual  Predicted
0  5054.7   5056.266
1  3809.2   3809.467
2  3245.3   3246.960
3  5728.8   5729.472
4  4148.0   4148.255
5  3645.6   3645.768
6  2317.7   2320.542
7  4100.8   4101.240
8  2286.9   2285.908
9  5394.4   5394.224
```

In [100…
```
rf_results.to_csv("random_forest_predictions.csv", index=False)
print("Predictions saved to 'random_forest_predictions.csv'")
```

```
Predictions saved to 'random_forest_predictions.csv'
```

Isolation Forest for Anomaly Detection

In [109…
```
#  01) _Use only numeric features (exclude City, Company since they are categc
features = ["Fan", "Refrigerator", "AirConditioner", "Television",
            "Monitor", "MotorPump", "Month", "MonthlyHours", "TariffRate", "El

X = df[features]
```

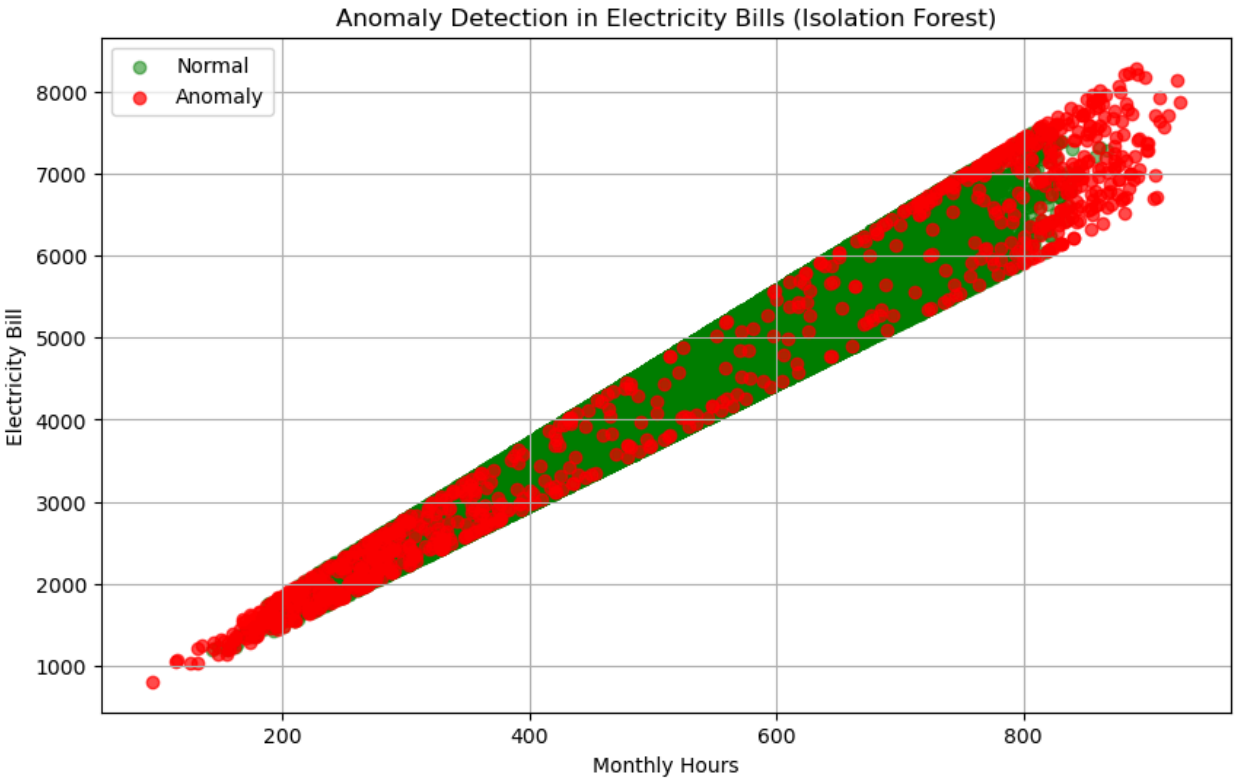
In [110…
```
# 02)
iso_forest = IsolationForest(contamination=0.02, random_state=42)  # assume 2%
df["Anomaly"] = iso_forest.fit_predict(X)   # -1 = anomaly, 1 = normal
```

In [111…
```
# 03)
normal_data = df[df["Anomaly"] == 1]
anomalies = df[df["Anomaly"] == -1]

print("Total Records:", len(df))
print("Normal Records:", len(normal_data))
print("Anomalies Detected:", len(anomalies))
```

```
Total Records: 45345
Normal Records: 44438
Anomalies Detected: 907
```

In [112…
```
#   04 ) Plot anomalies vs normal data (ElectricityBill vs MonthlyHours)
plt.figure(figsize=(10,6))
plt.scatter(normal_data["MonthlyHours"], normal_data["ElectricityBill"],
            color='green', label="Normal", alpha=0.5)
plt.scatter(anomalies["MonthlyHours"], anomalies["ElectricityBill"],
            color='red', label="Anomaly", alpha=0.7)
plt.xlabel("Monthly Hours")
plt.ylabel("Electricity Bill")
plt.title("Anomaly Detection in Electricity Bills (Isolation Forest)")
plt.legend()
plt.grid(True)
plt.show()
```



In [113…
```
# Show actual bills with anomaly labels
iso_results = df[["ElectricityBill", "Anomaly"]]
print(iso_results.head(10))
```

```
   ElectricityBill  Anomaly
0           3225.6        1
1           3806.4        1
2           3203.2        1
3           4370.0        1
4           4204.4        1
5           3485.4        1
6           6417.5        1
7           4182.0        1
8           4641.0        1
9           4392.8        1
```

In [114…
```
# Normal records
normal_data = df[df["Anomaly"] == 1]

# Anomalous records
anomalies = df[df["Anomaly"] == -1]

print("\nNormal Records Sample:")
print(normal_data.head(5))

print("\nAnomalies Detected Sample:")
print(anomalies.head(5))
```

```
Normal Records Sample:
   Fan  Refrigerator  AirConditioner  Television  Monitor  MotorPump  Month  \
0   16          23.0             2.0         6.0      1.0          0     10
1   19          22.0             2.0         3.0      1.0          0      5
2    7          20.0             2.0         6.0      7.0          0      7
3    7          22.0             3.0        21.0      1.0          0      6
4   11          23.0             2.0        11.0      1.0          0      2

        City                                    Company  MonthlyHours  \
0  Hyderabad                    Tata Power Company Ltd.           384
1   Vadodara                                       NHPC           488
2     Shimla                             Jyoti Structure           416
3     Mumbai                             Power Grid Corp           475
4     Mumbai  Ratnagiri Gas and Power Pvt. Ltd. (RGPPL)           457

   TariffRate  ElectricityBill  Anomaly
0         8.4           3225.6        1
1         7.8           3806.4        1
2         7.7           3203.2        1
3         9.2           4370.0        1
4         9.2           4204.4        1

Anomalies Detected Sample:
     Fan  Refrigerator  AirConditioner  Television  Monitor  MotorPump  Month
\
171   16          23.0             1.0        22.0     12.0          0      3
251   22          23.0             0.0        15.0     12.0          0     11
264   12          23.0             3.0        17.0     12.0          0     12
277   12          23.0             2.0        20.0     12.0          0      1
314   17          22.0             3.0        21.0     12.0          0      5

            City                          Company  MonthlyHours  TariffRate  \
171  Navi Mumbai                         SJVN Ltd.           768         9.3
251    Ratnagiri                         NLC India           735         7.4
264         Pune                         NLC India           802         9.1
277  Navi Mumbai                   Power Grid Corp           713         9.3
314      Kolkata  L&T Transmission & Distribution           888         8.7

     ElectricityBill  Anomaly
171           7142.4       -1
251           5439.0       -1
264           7298.2       -1
277           6630.9       -1
314           7725.6       -1
```

ARIMA Model__

In [119…

```python
# Take monthly average electricity bill
ts = df.groupby("Month")["ElectricityBill"].mean()

# Convert numeric months into datetime index (assuming year = 2025)
ts.index = pd.date_range(start="2025-01-01", periods=len(ts), freq="M")
```

```
print(ts)
```

```
2025-01-31    4403.964784
2025-02-28    3919.195462
2025-03-31    4409.862445
2025-04-30    4254.603439
2025-05-31    4398.011479
2025-06-30    4219.997220
2025-07-31    4432.699030
2025-08-31    4401.935753
2025-09-30    4221.012469
2025-10-31    4409.195557
2025-11-30    4258.952020
2025-12-31    4422.842174
Freq: ME, Name: ElectricityBill, dtype: float64
```

```
C:\Users\kambl\AppData\Local\Temp\ipykernel_20860\1511742778.py:5: FutureWarnin
g: 'M' is deprecated and will be removed in a future version, please use 'ME' i
nstead.
  ts.index = pd.date_range(start="2025-01-01", periods=len(ts), freq="M")
```

In [125…

```
# ARIMA order (p,d,q) = (1,1,1) as starting point
model = ARIMA(ts, order=(1,1,1))
model_fit = model.fit()
# Show model summary
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:        ElectricityBill   No. Observations:                   12
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -68.906
Date:                Fri, 03 Oct 2025   AIC                            143.811
Time:                        15:41:08   BIC                            145.005
Sample:                    01-31-2025   HQIC                           143.059
                         - 12-31-2025
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.7773      0.167     -4.658      0.000      -1.104      -0.450
ma.L1         -0.3187      0.413     -0.772      0.440      -1.128       0.491
sigma2      1.613e+04   9021.754      1.788      0.074   -1550.708    3.38e+04
===================================================================================
====
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
0.11
Prob(Q):                              1.00   Prob(JB):
0.94
Heteroskedasticity (H):               0.26   Skew:
0.05
Prob(H) (two-sided):                  0.23   Kurtosis:
2.51
===================================================================================
====

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

In [122…

```
# Forecast next 6 months
forecast = model_fit.forecast(steps=6)
# Print forecast values
print("Forecasted Electricity Bills:")
print(forecast)
```

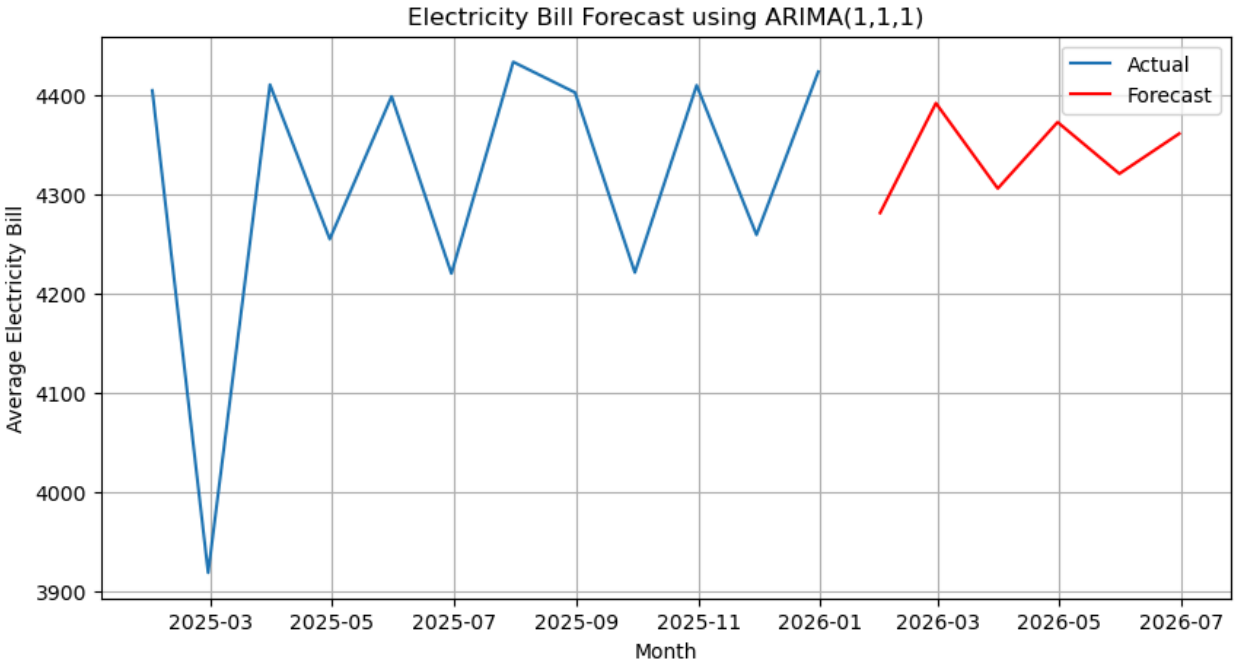
```
Forecasted Electricity Bills:
2026-01-31    4280.909265
2026-02-28    4391.229455
2026-03-31    4305.480882
2026-04-30    4372.130674
2026-05-31    4320.325791
2026-06-30    4360.592172
Freq: ME, Name: predicted_mean, dtype: float64
```

In [123…

```
# Plot actual vs forecast
plt.figure(figsize=(10,5))
plt.plot(ts, label="Actual")
plt.plot(pd.date_range(ts.index[-1] + pd.offsets.MonthEnd(1), periods=6, freq=
         forecast, label="Forecast", color="red")
plt.xlabel("Month")
plt.ylabel("Average Electricity Bill")
```

```
plt.title("Electricity Bill Forecast using ARIMA(1,1,1)")
plt.legend()
plt.grid(True)
plt.show()
```

```
C:\Users\kambl\AppData\Local\Temp\ipykernel_20860\606313886.py:4: FutureWarnin
g: 'M' is deprecated and will be removed in a future version, please use 'ME' i
nstead.
  plt.plot(pd.date_range(ts.index[-1] + pd.offsets.MonthEnd(1), periods=6, fre
q="M"),
```



In [137…
```
# Step 2: Predict values for the same period as your dataset
predicted = model_fit.predict(start=0, end=len(ts)-1, typ='levels')
```

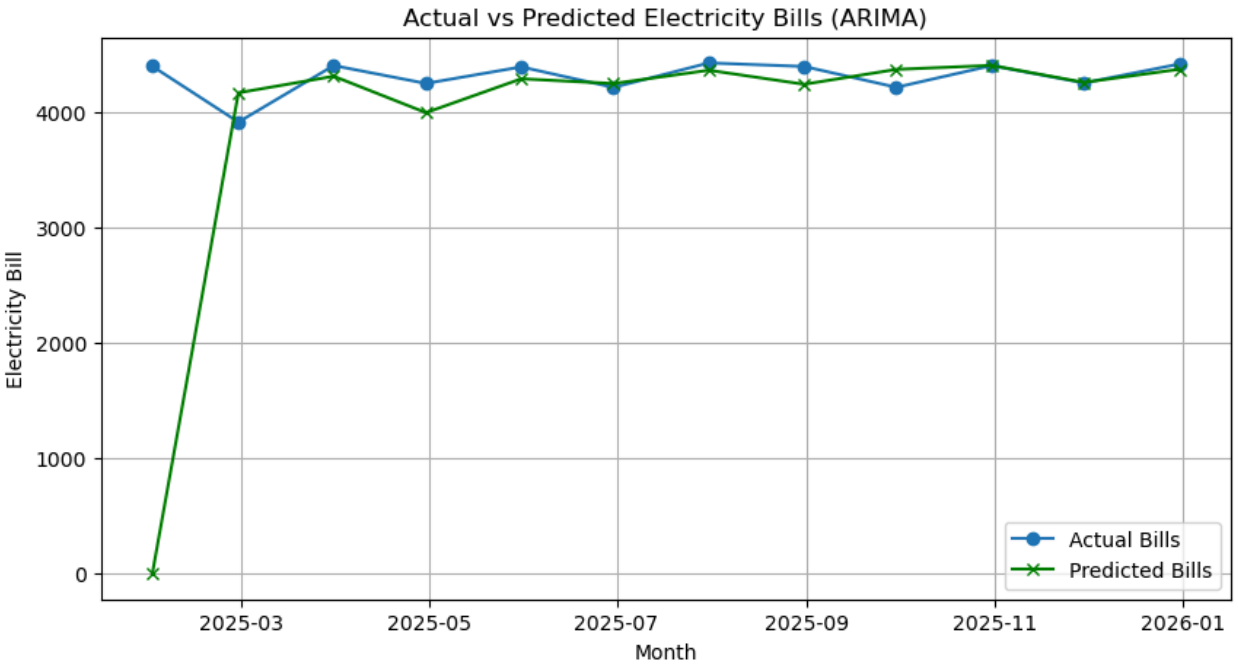
In [138…
```
# Step 3: Create a comparison DataFrame
comparison = pd.DataFrame({
    'Month': ts.index,            # Month or index
    'Actual_Bill': ts.values,     # Actual electricity bill
    'Predicted_Bill': predicted   # Predicted electricity bill
})
```

In [139…
```
# Step 4: Display the comparison
print("Comparison of Actual vs Predicted Electricity Bills:")
print(comparison.head(10))  # Show first 10 rows for example
```

```
Comparison of Actual vs Predicted Electricity Bills:
                Month  Actual_Bill  Predicted_Bill
2025-01-31 2025-01-31  4403.964784        0.000000
2025-02-28 2025-02-28  3919.195462     4173.541710
2025-03-31 2025-03-31  4409.862445     4317.027291
2025-04-30 2025-04-30  4254.603439     4000.964228
2025-05-31 2025-05-31  4398.011479     4295.015211
2025-06-30 2025-06-30  4219.997220     4253.742532
2025-07-31 2025-07-31  4432.699030     4369.116465
2025-08-31 2025-08-31  4401.935753     4247.108202
2025-09-30 2025-09-30  4221.012469     4376.502450
2025-10-31 2025-10-31  4409.195557     4411.194537
```

In [140…

```python
# Step 5: Plot actual vs predicted bills
plt.figure(figsize=(10,5))
plt.plot(ts, label="Actual Bills", marker='o')
plt.plot(predicted, label="Predicted Bills", color="green", marker='x')
plt.xlabel("Month")
plt.ylabel("Electricity Bill")
plt.title("Actual vs Predicted Electricity Bills (ARIMA)")
plt.legend()
plt.grid(True)
plt.show()
```



Recommendation Layer (Optimization)

In [152…

```python
# Combine actual and predicted bills
recommendation_df = pd.DataFrame({
    'Month': ts.index,
    'ActualBill': ts.values,
    'PredictedBill': predicted   # <-- use 'predicted' from ARIMA
```

```
})
recommendation_df.head()
```

Out[152…

	Month	ActualBill	PredictedBill
2025-01-31	2025-01-31	4403.964784	0.000000
2025-02-28	2025-02-28	3919.195462	4173.541710
2025-03-31	2025-03-31	4409.862445	4317.027291
2025-04-30	2025-04-30	4254.603439	4000.964228
2025-05-31	2025-05-31	4398.011479	4295.015211

In [156…

```
# Align lengths
n = len(ts)  # number of months in ts
subset_df = df.iloc[:n]  # take only first n rows

# Create recommendation dataframe
recommendation_df = pd.DataFrame({
    'Month': ts.index,
    'ActualBill': ts.values,
    'PredictedBill': predicted,   # ARIMA predicted values
    'Fan': subset_df['Fan'].values,
    'MotorPump': subset_df['MotorPump'].values
})
```

In [157…

```
# Generate suggestions
def generate_suggestions(df):
    suggestions = []
    for _, row in df.iterrows():
        rules = []
        if row['PredictedBill'] > row['ActualBill'] * 1.2:
            rules.append("High predicted bill: check appliances")
        if row['Fan'] > 20:
            rules.append("Shift Fan usage to off-peak hours")
        if row['MotorPump'] > 0:
            rules.append("Check MotorPump for leakage")
        suggestions.append(", ".join(rules) if rules else "No action needed")
    return suggestions

recommendation_df['Suggestions'] = generate_suggestions(recommendation_df)

# Display the recommendations
recommendation_df.head()
```

Out[157…

	Month	ActualBill	PredictedBill	Fan	MotorPump	Suggestion
2025-01-31	2025-01-31	4403.964784	0.000000	16	0	No actio needed
2025-02-28	2025-02-28	3919.195462	4173.541710	19	0	No actio needed
2025-03-31	2025-03-31	4409.862445	4317.027291	7	0	No actio needed
2025-04-30	2025-04-30	4254.603439	4000.964228	7	0	No actio needed
2025-05-31	2025-05-31	4398.011479	4295.015211	11	0	No actio needed

In [158…

```
# Show months where any action is suggested
recommendation_df[recommendation_df['Suggestions'] != "No action needed"]
```

Out[158…

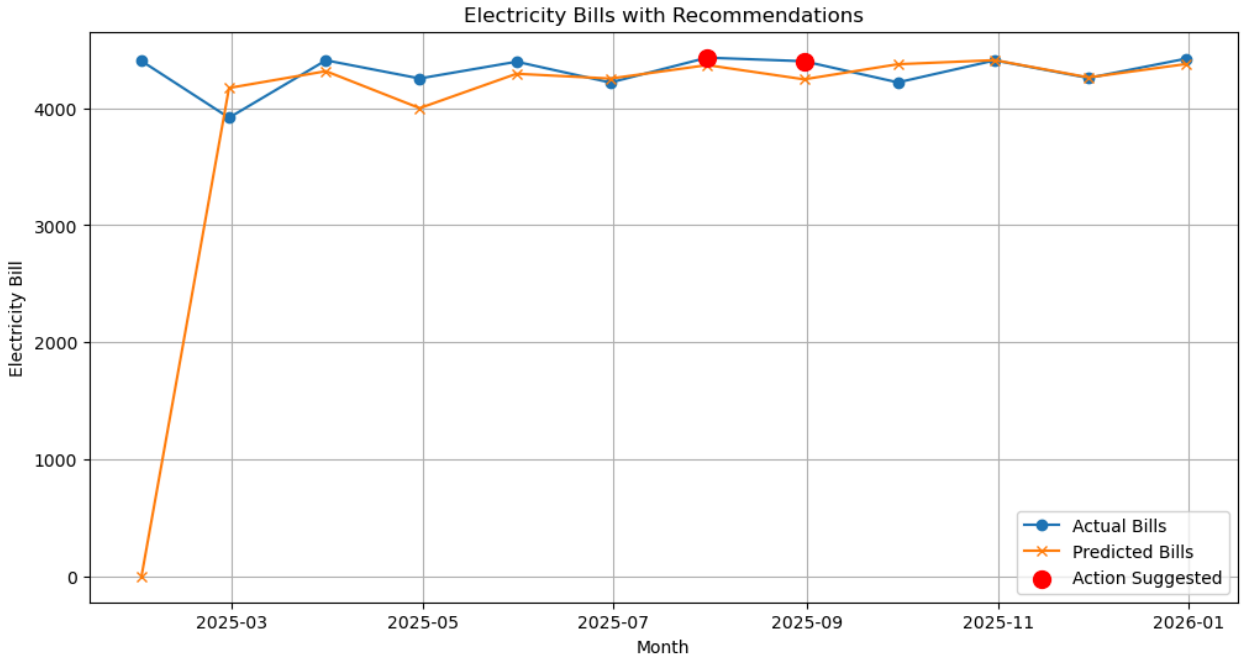
	Month	ActualBill	PredictedBill	Fan	MotorPump	Suggestion
2025-07-31	2025-07-31	4432.699030	4369.116465	23	0	Shift Fa usage to of peak hou
2025-08-31	2025-08-31	4401.935753	4247.108202	22	0	Shift Fa usage to of peak hou

In [161…

```
plt.figure(figsize=(12,6))
plt.plot(ts, label='Actual Bills', marker='o')
plt.plot(predicted, label='Predicted Bills', marker='x')  # use 'predicted'

# Highlight months needing action
action_months = recommendation_df[recommendation_df['Suggestions'] != "No acti
plt.scatter(action_months, ts[action_months], color='red', s=100, label='Actio

plt.xlabel("Month")
plt.ylabel("Electricity Bill")
plt.title("Electricity Bills with Recommendations")
plt.legend()
plt.grid(True)
plt.show()
```


Electricity Bills with Recommendations
Electricity Bill
0
1000
2000
3000
4000
2025-03
2025-05
2025-07
2025-09
2025-11
2026-01
Month
Actual Bills
Predicted Bills
Action Suggested

In [ ]: