The Pulse of Modern Living: How IoT Data Shapes Smarter Strategies

(Demonstrating the Power of Data Collection & Analysis)

Kunal Jain

Appendix 1

```
import pandas as pd
import h5py
import hdf5plugin
from datetime import datetime

# Path to UK-DALE HDF5 file
file_path = '/users/kjain/Downloads/ukdale.h5'

# Define a manual mapping of meters to appliances
meter_mapping = {
    'meter1': 'Main Power',
    'meter2': 'stereo_speakers_bedroom',
    'meter3': 'i7_desktop',
    'meter4': 'hairdryer',
    'meter5': 'primary_tv',
    'meter6': '24_inch_lcd_bedroom',
    'meter7': 'treadmill',
    'meter8': 'network_attached_storage',
    'meter9': 'core2_server',
    'meter10': '24_inch_lcd',
    'meter11': 'PS4',
    'meter12': 'steam_iron',
```

```
    'meter13': 'nespresso_pixie',
    'meter14': 'atom_pc',
    'meter15': 'toaster',
    'meter16': 'home_theatre_amp',
    'meter17': 'sky_hd_box',
    'meter18': 'kettle',
    'meter19': 'fridge_freezer',
    'meter20': 'oven',
    'meter21': 'electric_hob',
    'meter22': 'dishwasher',
    'meter23': 'microwave',
    'meter24': 'washer_dryer',
    'meter25': 'vacuum_cleaner'
}

# Open the HDF5 file
with h5py.File(file_path, 'r') as hdf:
    print("Keys:", list(hdf.keys())) # Check available buildings

    # Navigate to House 1's electricity data
    house_path = 'building5/elec'
    house_data = hdf[house_path]

    # List to store DataFrames for each meter
    meter_dfs = []

    # Process meters (select relevant meters based on mapping)
    for meter_id in meter_mapping.keys():
        meter_path = f"{house_path}/{meter_id}"
```

```
        if "table" in house_data[meter_id]:
            print(f"Processing {meter_id} ({meter_mapping[meter_id]})...")
            table_data = house_data[meter_id]['table']
            timestamps = []
            power_values = []
            for entry in table_data[:500000]: # Read only 500,000 entries at a time
                timestamp, power = entry
                timestamps.append(datetime.utcfromtimestamp(timestamp / 1e9)) # Convert
nanoseconds
                power_values.append(power[0]) # Extract first power value

            # Create DataFrame
            df_meter = pd.DataFrame({'timestamp': timestamps, meter_mapping[meter_id]:
power_values})

            # Append to list (we will merge later)
            meter_dfs.append(df_meter)

    # Merge all meter DataFrames on timestamp
    df = meter_dfs[0]
    for meter_df in meter_dfs[1:]:
        df = df.merge(meter_df, on='timestamp', how='outer') # Outer join to include all timestamps
    # Convert timestamp to DateTimeIndex
    df['timestamp'] = pd.to_datetime(df['timestamp'])
    df.set_index('timestamp', inplace=True)
    # Filter for January 2013
    df_month = df.loc['2014-06-29':'2014-07-29']
    # Save to CSV
    df_month.to_csv('house5_appliances.csv')
```

```
print("Data saved to house5_appliances.csv")
```

Appendix 2

Time-Series Energy Usage Pattern

```
import pandas as pd
import numpy as np
import hdf5plugin
import h5py
import matplotlib.pyplot as plt

# Load the CSV file
file_path = 'house5_appliances.csv'
df = pd.read_csv(file_path, parse_dates=['timestamp'], index_col='timestamp')

# Resample data to hourly mean to reduce noise
df_resampled = df.resample('1H').mean()

# Plot Time Series Data
plt.figure(figsize=(15, 6))
for column in df_resampled.columns:
    plt.plot(df_resampled.index, df_resampled[column], label=column)

plt.xlabel('Timestamp')
plt.ylabel('Energy Usage (Watts)')
plt.title('Energy Usage Patterns')
plt.legend()
plt.xticks(rotation=45)
plt.grid()
```

```python
plt.show()
```

Appendix 3

Appliance-Level Energy Consumption

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.ensemble import IsolationForest

# Load dataset (Ensure CSV format is maintained)
df = pd.read_csv('/Users/kjain/Project/house5_appliances.csv', parse_dates=['timestamp'])
df.set_index('timestamp', inplace=True)
df.fillna(method='ffill', inplace=True)  # Handle missing values minimally

# Define colors for better visualization
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'orange', 'purple', 'brown']

# Box Plot (Power Consumption Distribution) ---
plt.figure(figsize=(14, 6))
sns.boxplot(data=df, palette='tab10', showfliers=True)
plt.xticks(rotation=90)
plt.ylabel('Power Consumption (W)')
plt.title('Power Consumption Distribution (Box Plot)')
plt.grid()
plt.show()
```

```
# Heatmap (Time vs. Appliance Usage) ---
df_hourly = df.resample('H').mean()  # Aggregate data to hourly resolution
plt.figure(figsize=(14, 6))
sns.heatmap(df_hourly.T, cmap='coolwarm', linewidths=0.5, cbar=True)
plt.xlabel('Time')
plt.ylabel('Appliances')
plt.title('Heatmap of Appliance Power Usage Over Time')
plt.xticks(rotation=45)
plt.show()
```

Appendix 4

Anomaly Detection Visualization

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.signal import find_peaks

# Load the CSV file
df = pd.read_csv('house5_appliances.csv', parse_dates=['timestamp'], index_col='timestamp')

# Drop NaN values from 'Main Power'
df = df[['Main Power']].dropna()

# Extract hour and day for heatmap analysis
df['hour'] = df.index.hour
df['day'] = pd.to_datetime(df.index.date)  # Convert day to datetime format

# Compute Rolling Mean for trend detection
df['Rolling Mean'] = df['Main Power'].rolling(window=50, min_periods=1).mean()

# Compute Autocorrelation for periodicity detection
if len(df) > 1:
```

```
    autocorr = np.correlate(df['Main Power'] - np.mean(df['Main Power']), df['Main Power'] - np.mean(df['Main Power']), mode='full')

    autocorr = autocorr[len(autocorr) // 2:]  # Keep positive lags

    peaks, _ = find_peaks(autocorr, height=0)

    dataset_period = peaks[0] if len(peaks) > 0 else None

else:

    autocorr = []

    dataset_period = None


# Time-Series Plot with Rolling Mean

plt.figure(figsize=(12, 6))

plt.plot(df.index, df['Main Power'], label="Actual Power", alpha=0.5)

plt.plot(df.index, df['Rolling Mean'], label="Rolling Mean (50 readings)", color='red')

plt.title("Main Power Consumption Over Time")

plt.xlabel("Time")

plt.ylabel("Power (W)")

plt.legend()

plt.grid(True)

plt.show()


# Autocorrelation Analysis

plt.figure(figsize=(12, 6))

if len(autocorr) > 0:

    plt.plot(autocorr, label="Autocorrelation")

    if dataset_period:

        plt.axvline(x=dataset_period, color='r', linestyle='--', label=f"Detected Period: {dataset_period}")

    plt.title("Autocorrelation of Power Usage")

    plt.xlabel("Lag")

    plt.ylabel("Autocorrelation")
```

```
    plt.legend()
    plt.grid(True)
else:
    plt.text(0.5, 0.5, "Not enough data for Autocorrelation", horizontalalignment='center',
verticalalignment='center', fontsize=12)
    plt.title("Autocorrelation of Power Usage")
    plt.axis("off")
plt.show()
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load the CSV file
file_path = 'house5_all_appliances.csv'
df = pd.read_csv(file_path, parse_dates=['timestamp'], index_col='timestamp')

# Resample data to hourly mean to reduce noise
df_resampled = df.resample('1H').mean()

# Plot Time Series Data
plt.figure(figsize=(15, 6))
for column in df_resampled.columns:
    plt.plot(df_resampled.index, df_resampled[column], label=column, alpha=0.7)

plt.xlabel('Timestamp')
plt.ylabel('Energy Usage (Watts)')
plt.title('Energy Usage Patterns')
plt.xticks(rotation=45)
plt.grid()
```

```python
# Move legend below the graph
plt.legend(ncol=4, fontsize='small', loc='upper center', bbox_to_anchor=(0.5, -0.15))
plt.show()

# --- Anomaly Detection ---
# Define threshold using standard deviation
threshold = 3  # Adjust this if needed
mean_usage = df_resampled.mean()
std_dev = df_resampled.std()

# Identify anomalies (values beyond mean ± threshold * std deviation)
anomalies = ((df_resampled - mean_usage).abs() > (threshold * std_dev))

# Plot Anomalies
plt.figure(figsize=(15, 6))
for column in df_resampled.columns:
    plt.plot(df_resampled.index, df_resampled[column], label=column, alpha=0.5)
    anomaly_points = df_resampled[column][anomalies[column]]
    plt.scatter(anomaly_points.index, anomaly_points, color='red', edgecolors='black',
                label=f'Anomaly ({column})', marker='x', s=50)

plt.xlabel('Timestamp')
plt.ylabel('Energy Usage (Watts)')
plt.title('Energy Usage with Anomalies Detected')
plt.xticks(rotation=45)
plt.grid()

# Move legend below the graph
```

```
plt.legend(ncol=4, fontsize='small', loc='upper center', bbox_to_anchor=(0.5, -0.15))
plt.show()

# Print anomaly timestamps
for column in df_resampled.columns:
    print(f"\nAnomalies in {column}:")
    print(df_resampled[column][anomalies[column]].dropna())
```