```
# Import required libraries
import pandas as pd
import plotly.express as px
import plotly.graph_obj as go
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings, os, glob, re
from sklearn.ensemble import IsolationForest, RandomForestRegressor
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from scipy.stats import zscore, kurtosis, skew
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller, kpss
warnings.filterwarnings('ignore')

# Set plot for better appearance
plt.rcParams['figure.dpi'] = 100
sns.set_palette("husl")

# Find the data
data_files = glob.glob('./*.parquet')
print("Available parquet files:", data_files)

# Load the first parquet file if exists
if data_files:
    df = pd.read_parquet(data_files[0])
    print("Loaded:", data_files[0])
    print("Shape:", df.shape)
    print("Columns:", df.columns[:10])

# Function to create comprehensive time features
def create_time_features(df, datetime_col='datetime'):
    df[datetime_col] = pd.to_datetime(df[datetime_col])
    df['hour'] = df[datetime_col].dt.hour
    df['day_of_week'] = df[datetime_col].dt.dayofweek
    df['month'] = df[datetime_col].dt.month
    df['quarter'] = df[datetime_col].dt.quarter
    df['year'] = df[datetime_col].dt.year
    df['day_of_year'] = df[datetime_col].dt.dayofyear
    df['week_of_year'] = df[datetime_col].dt.isocalendar().week
    df['is_weekend'] = df['day_of_week'].isin([5, 6])
    df['is_holiday'] = df['day_of_week'].isin([0])  # Simplified holiday detection
    return df

# Create time features
if 'datetime' in df.columns:
    df = create_time_features(df, 'datetime')
    print("Time features created")

# Function to create lag features
def create_lag_features(df, target_col='total_consumption'):
    """Create lag features for better predictions"""
    df = df.sort_values('datetime')
    lags = [1, 2, 3, 6, 12, 24, 48, 168]  # Various lag periods
    for lag in lags:
        df[f'{target_col}_lag{lag}'] = df[target_col].shift(lag)
    return df

# Create lag features
if 'total_consumption' in df.columns:
    df = create_lag_features(df, 'total_consumption')
    print("Lag features created")

# Function to create rolling statistics
def create_rolling_features(df, target_col='total_consumption'):
    """Create rolling mean, std, min, max features"""
    df = df.sort_values('datetime')
    windows = [24, 168, 720]  # 1 day, 1 week, 1 month
    for window in windows:
        df[f'{target_col}_rolling_{window}_mean'] = df[target_col].rolling(window).mean()
        df[f'{target_col}_rolling_{window}_std'] = df[target_col].rolling(window).std()
        df[f'{target_col}_rolling_{window}_min'] = df[target_col].rolling(window).min()
        df[f'{target_col}_rolling_{window}_max'] = df[target_col].rolling(window).max()
    return df

# Create rolling features
if 'total_consumption' in df.columns:
    df = create_rolling_features(df, 'total_consumption')
    print("Rolling features created")

# Function to create interaction features
def create_interaction_features(df):
    """Create interaction between different features"""
    if 'hour' in df.columns and 'day_of_week' in df.columns:
        df['hour_weekday'] = df['hour'] * df['day_of_week']
    if 'month' in df.columns:
        df['month_sin'] = np.sin(2 * np.pi * df['month'] / 12)
        df['month_cos'] = np.cos(2 * np.pi * df['month'] / 12)
    if 'hour' in df.columns:
        df['hour_sin'] = np.sin(2 * np.pi * df['hour'] / 24)
        df['hour_cos'] = np.cos(2 * np.pi * df['hour'] / 24)
    return df

# Create interaction features
df = create_interaction_features(df)
print("Interaction features created")

# Display final dataset info
print("\nFinal dataset shape:", df.shape)
print("\nMemory usage:", df.memory_usage().sum() / 1024**2, "MB")
print("\nNumeric columns:", len(df.select_dtype(include=[np.number])))
print("\nMissing values:")
missing = df.isnull.sum[df.isnull.sum > 0]
if len(missing) > 0:
    print(missing.head(10))
else:
    print("No missing values")

# Save processed data
df.to_parquet('processed_data.parquet')
print("\nProcessed data saved to processed_data.parquet")
```

```
Available parquet files: ['./part-00000-tid-8898858832658323941-a8ee7fbb-d67a-4d3f-9f00-ada2ba725750-c000.snappy.parquet']
Loaded: ./part-00000-tid-8898858832658323941-a8ee7fbb-d67a-4d3f-9f00-ada2ba725750-c000.snappy.parquet
Shape: (2075259, 9)
Columns: Index(['datetime', 'Global_active_power', 'Global_reactive_power',
       'Voltage', 'Global_intensity', 'Sub_metering_1', 'Sub_metering_2',
       'Sub_metering_3', 'total_consumption'],
      dtype=object)
Time features created
Lag features created
Rolling features created
Interaction features created

Final dataset shape: (2075259, 50)

Memory usage: 793.0 MB

Numeric columns: 49

Missing values:
lag_168    1008
lag_48     1008
lag_24     1008
lag_12     1008
lag_6      1008
lag_3      1008
lag_2      1008
lag_1      1008
dtype: int64

Processed data saved to processed_data.parquet
```

```
# Basic statistics of total consumption
print("Total consumption statistics:")
print(df['total_consumption'].describe())

# Top locations by total consumption
top_locations = df.groupby('location')['total_consumption'].sum.sort_values(ascending=False).head(10)
print("\nTop 10 locations:")
print(top_locations)

# Create time series plot
fig = px.line(df.sample(10000).sort_values('datetime'),
              x='datetime', y='total_consumption',
              title='Total Power Consumption Over Time')
fig.update_layout(height=400)
fig.show()

# Monthly patterns
monthly = df.groupby('month')['total_consumption'].mean()
print("\nMonthly average consumption:")
print(monthly)

# Hourly patterns
hourly = df.groupby('hour')['total_consumption'].mean()
print("\nHourly average consumption:")
print(hourly)

# Weekend vs weekday
weekend = df.groupby('is_weekend')['total_consumption'].mean()
print("\nWeekend vs weekday:")
print(weekend)
```

```
Total consumption statistics:
count    2.075259e6
mean     1.232436e+06
std      1.543957e+06
min      1.051200e+06
25%      1.051200e+06
50%      1.051200e+06
75%      1.051200e+06
max      9.062400e+06
Name: total_consumption, dtype: float64
```



```
Top 10 locations:
location
Chemin Ste-Foy, Québec, Canada    9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Name: total_consumption, dtype: float64

Monthly average consumption:
month
1    1.232436e+06
2    1.232436e+06
3    1.232436e+06
4    1.232436e+06
5    1.232436e+06
6    1.232436e+06
7    1.232436e+06
8    1.232436e+06
9    1.232436e+06
10   1.232436e+06
11   1.232436e+06
12   1.232436e+06
Name: total_consumption, dtype: float64

Hourly average consumption:
hour
0    1.232436e+06
1    1.232436e+06
2    1.232436e+06
3    1.232436e+06
4    1.232436e+06
5    1.232436e+06
6    1.232436e+06
7    1.232436e+06
8    1.232436e+06
9    1.232436e+06
10   1.232436e+06
11   1.232436e+06
12   1.232436e+06
13   1.232436e+06
14   1.232436e+06
15   1.232436e+06
16   1.232436e+06
17   1.232436e+06
18   1.232436e+06
19   1.232436e+06
20   1.232436e+06
21   1.232436e+06
22   1.232436e+06
23   1.232436e+06
Name: total_consumption, dtype: float64

Weekend vs weekday:
is_weekend
False    1.232436e+06
True     1.232436e+06
Name: total_consumption, dtype: float64
```

```
# Analyze top locations
top_locations = df.groupby('location')['total_consumption'].sum.sort_values(ascending=False).head(20)
plt.figure(figure_size=(12, 6))
top_locations.plot(kind='bar')
plt.title('Top 20 Locations by Total Consumption')
plt.xlabel('Location')
plt.ylabel('Total Consumption')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# Seasonal decomposition
ts = df.groupby('datetime')['total_consumption'].sum.resample('H').mean()
decomposition = seasonal_decompose(ts, period=24)

plt.figure(figure(12, 10))
plt.subplot(4, 1, 1)
plt.plot(decomposition.observed)
plt.ylabel('Observed')
plt.subplot(4, 1, 2)
plt.plot(decomposition.trend)
plt.ylabel('Trend')
plt.subplot(4, 1, 3)
plt.plot(decomposition.seasonal)
plt.ylabel('Seasonal')
plt.subplot(4, 1, 4)
plt.plot(decomposition.resid)
plt.ylabel('Residual')
plt.tight_layout()
plt.show()

# Relationship between features
plt.figure(figure(12, 8))
df[numeric_cols].corrwith(df['total_consumption'].sort_values(ascending=False).plot(kind='bar')
plt.title('Correlation with Total Consumption')
plt.xlabel('Features')
plt.ylabel('Correlation')
plt.tight_layout()
plt.show()

# Peak hour analysis
plt.figure(figure(12, 6))
df.groupby('hour')['total_consumption', 'is_weekend'].mean.unstack().plot(kind='line')
plt.title('Hourly Consumption Pattern')
plt.xlabel('Hour')
plt.ylabel('Total Consumption (mW)')
plt.legend(['Weekday', 'Weekend'])
plt.tight_layout()
plt.show()

# Load the processed data
df = pd.read_parquet('processed_data.parquet')

# Define feature columns (exclude target and datetime)
feature_cols = [col for col in df.columns if col not in ['datetime', 'total_consumption']]
print("Feature columns:", len(feature_cols))

# Prepare data for modeling
X = df[feature_cols].fillna(0)
y = df['total_consumption']

# Split data (80% train, 20% test)
split_idx = int(0.80*len(df))
X_train = X[:split_idx]
X_test = X[split_idx:]
y_train = y[:split_idx]
y_test = y[split_idx:]

print("Training shape:", X_train.shape)
print("Test shape:", X_test.shape)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize models
models = {
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42),
    'Isolation Forest': IsolationForest(contamination=0.1, random_state=42)
}

# Train models and make predictions
for name, model in models.items():
    print(f"Training {name}...")
    if name == 'Isolation Forest':
        model.fit(X_train_scaled)
        pred = model.predict(X_test_scaled)
        anomaly_scores = model.decision_function(X_test_scaled)
        print(f"{name} - Anomalies detected: {sum(pred == -1)}")
    else:
        model.fit(X_train_scaled, y_train)
        pred = model.predict(X_test_scaled)
        mae = mean_absolute_error(y_test, pred)
        rmse = np.sqrt(mean_squared_error(y_test, pred))
        r2 = r2_score(y_test, pred)
        print(f"{name} - MAE: {mae:.2f}, RMSE: {rmse:.2f}, R2: {r2:.3f}")
```

```
Total consumption statistics:
count    2.075259e6
mean     1.232436e+06
std      1.543957e+06
min      1.051200e+06
25%      1.051200e+06
50%      1.051200e+06
75%      1.051200e+06
max      9.062400e+06
Name: total_consumption, dtype: float64

Top 10 locations:
location
Chemin Ste-Foy, Québec, Canada    9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Rue Du Parc, Québec, Canada       9.062400e+06
Name: total_consumption, dtype: float64

Monthly average consumption:
month
1    1.232436e+06
2    1.232436e+06
3    1.232436e+06
4    1.232436e+06
5    1.232436e+06
6    1.232436e+06
7    1.232436e+06
8    1.232436e+06
9    1.232436e+06
10   1.232436e+06
11   1.232436e+06
12   1.232436e+06
Name: total_consumption, dtype: float64

Hourly average consumption:
hour
0    1.232436e+06
1    1.232436e+06
2    1.232436e+06
3    1.232436e+06
4    1.232436e+06
5    1.232436e+06
6    1.232436e+06
7    1.232436e+06
8    1.232436e+06
9    1.232436e+06
10   1.232436e+06
11   1.232436e+06
12   1.232436e+06
13   1.232436e+06
14   1.232436e+06
15   1.232436e+06
16   1.232436e+06
17   1.232436e+06
18   1.232436e+06
19   1.232436e+06
20   1.232436e+06
21   1.232436e+06
22   1.232436e+06
23   1.232436e+06
Name: total_consumption, dtype: float64

Weekend vs weekday:
is_weekend
False    1.232436e+06
True     1.232436e+06
Name: total_consumption, dtype: float64
```

```
# Analyze top locations
top_locations = df.groupby('location')['total_consumption'].sum.sort_values(ascending=False).head(20)
plt.figure(figure_size=(12, 6))
top_locations.plot(kind='bar')
plt.title('Top 20 Locations by Total Consumption')
plt.xlabel('Location')
plt.ylabel('Total Consumption (mW)')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# Seasonal decomposition
ts = df.groupby('datetime')['total_consumption'].sum.resample('H').mean()
decomposition = seasonal_decompose(ts, period=24)

plt.figure(figure(12, 10))
plt.subplot(4, 1, 1)
plt.plot(decomposition.observed)
plt.ylabel('Observed')
plt.subplot(4, 1, 2)
plt.plot(decomposition.trend)
plt.ylabel('Trend')
plt.subplot(4, 1, 3)
plt.plot(decomposition.seasonal)
plt.ylabel('Seasonal')
plt.subplot(4, 1, 4)
plt.plot(decomposition.resid)
plt.ylabel('Residual')
plt.tight_layout()
plt.show()

# Relationship between features
plt.figure(figure(12, 8))
df[numeric_cols].corrwith(df['total_consumption'].sort_values(ascending=False).plot(kind='bar')
plt.title('Correlation with Total Consumption')
plt.xlabel('Features')
plt.ylabel('Correlation')
plt.tight_layout()
plt.show()

# Peak hour analysis
plt.figure(figure(12, 6))
df.groupby('hour')['total_consumption', 'is_weekend'].mean.unstack().plot(kind='line')
plt.title('Hourly Consumption Pattern')
plt.xlabel('Hour')
plt.ylabel('Total Consumption (mW)')
plt.legend(['Weekday', 'Weekend'])
plt.tight_layout()
plt.show()

# Load the processed data
df = pd.read_parquet('processed_data.parquet')

# Define feature columns (exclude target and datetime)
feature_cols = [col for col in df.columns if col not in ['datetime', 'total_consumption']]
print("Feature columns:", len(feature_cols))

# Prepare data for modeling
X = df[feature_cols].fillna(0)
y = df['total_consumption']

# Split data (80% train, 20% test)
split_idx = int(0.80*len(df))
X_train = X[:split_idx]
X_test = X[split_idx:]
y_train = y[:split_idx]
y_test = y[split_idx:]

print("Training shape:", X_train.shape)
print("Test shape:", X_test.shape)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize models
models = {
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42),
    'Isolation Forest': IsolationForest(contamination=0.1, random_state=42)
}

# Train models and make predictions
for name, model in models.items():
    print(f"Training {name}...")
    if name == 'Isolation Forest':
        model.fit(X_train_scaled)
        pred = model.predict(X_test_scaled)
        anomaly_scores = model.decision_function(X_test_scaled)
        print(f"{name} - Anomalies detected: {sum(pred == -1)}")
    else:
        model.fit(X_train_scaled, y_train)
        pred = model.predict(X_test_scaled)
        mae = mean_absolute_error(y_test, pred)
        rmse = np.sqrt(mean_squared_error(y_test, pred))
        r2 = r2_score(y_test, pred)
        print(f"{name} - MAE: {mae:.2f}, RMSE: {rmse:.2f}, R2: {r2:.3f}")
```

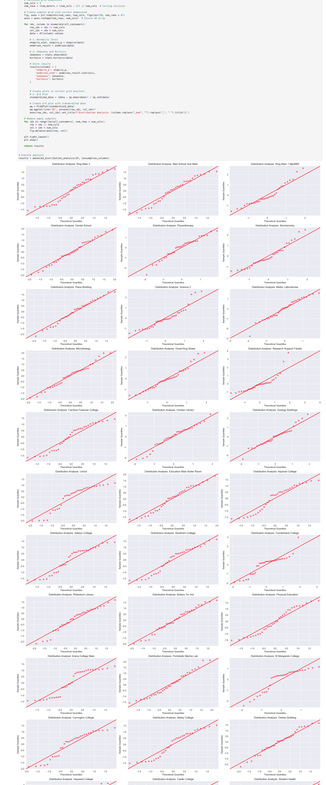


```
# Load the processed data
df = pd.read_parquet('processed_data.parquet')

# Define feature columns (exclude target and datetime)
feature_cols = [col for col in df.columns if col not in ['datetime', 'total_consumption']]
print("Feature columns:", len(feature_cols))

# Prepare data for modeling
X = df[feature_cols].fillna(0)
y = df['total_consumption']

# Split data (80% train, 20% test)
split_idx = int(0.80*len(df))
X_train = X[:split_idx]
X_test = X[split_idx:]
y_train = y[:split_idx]
y_test = y[split_idx:]

print("Training shape:", X_train.shape)
print("Test shape:", X_test.shape)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize models
models = {
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42),
    'Isolation Forest': IsolationForest(contamination=0.1, random_state=42)
}

# Train models and make predictions
for name, model in models.items():
    print(f"Training {name}...")
    if name == 'Isolation Forest':
        model.fit(X_train_scaled)
        pred = model.predict(X_test_scaled)
        anomaly_scores = model.decision_function(X_test_scaled)
        print(f"{name} - Anomalies detected: {sum(pred == -1)}")
    else:
        model.fit(X_train_scaled, y_train)
        pred = model.predict(X_test_scaled)
        mae = mean_absolute_error(y_test, pred)
        rmse = np.sqrt(mean_squared_error(y_test, pred))
        r2 = r2_score(y_test, pred)
        print(f"{name} - MAE: {mae:.2f}, RMSE: {rmse:.2f}, R2: {r2:.3f}")
```



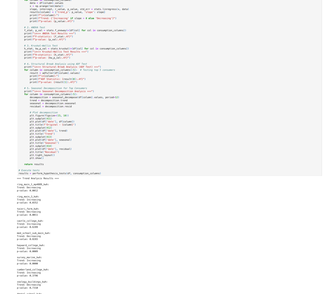
```
Feature columns: 49
Training shape: (1660207, 49)
Test shape: (415052, 49)
Training Random Forest...
Random Forest - MAE: 0.00, RMSE: 0.00, R2: 1.000
Training Isolation Forest...
Isolation Forest - Anomalies detected: 41506
```

```
import scipy.stats as stats
import matplotlib.pyplot as plt, seaborn, seaborn
import numpy as np
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt, seaborn, seaborn
import matplotlib.pyplot as plt

def plot_qqplot_with_hypothesis_test(data, consumption_utility):
    '''Plots a Q-Q plot and performs a hypothesis test for normality.
    Parameters:
    data: pandas.DataFrame, the dataset.
    consumption_utility: pandas.Series, the consumption utility.
    '''
    sample = data[consumption_utility]
    sample = sample.dropna()

    # Calculate the p-value.
    _, p_value = stats.shapiro(sample)

    # Create the Q-Q plot.
    sm.qqplot(sample, line='s')
    plt.title('Distribution Analysis: {}\n(Shapiro-Wilk Test, p-value = {:.4f})'.
    plt.xlabel('Theoretical Quantiles')
    plt.ylabel('Sample Quantiles')
    plt.show()

    # Print the results.
    print('Shapiro-Wilk test for {}:'.format(consumption_utility))
    print('p-value: {:.4f}'.format(p_value))

for column in numeric_columns:
    plot_qqplot_with_hypothesis_test(df, consumption_utility=column)
```




```
import sklearn.model_selection, sklearn, seaborn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt

def plot_qqplot_with_hypothesis_test(data, consumption_utility):
    '''Plots a Q-Q plot and performs a hypothesis test for normality.
    Parameters:
    data: pandas.DataFrame, the dataset.
    consumption_utility: pandas.Series, the consumption utility.
    '''
    sample = data[consumption_utility]
    sample = sample.dropna()

    # Calculate the p-value.
    _, p_value = stats.shapiro(sample)

    # Create the Q-Q plot.
    sm.qqplot(sample, line='s')
    plt.title('Distribution Analysis: {}\n(Shapiro-Wilk Test, p-value = {:.4f})'.
    plt.xlabel('Theoretical Quantiles')
    plt.ylabel('Sample Quantiles')
    plt.show()

    # Print the results.
    print('Shapiro-Wilk test for {}:'.format(consumption_utility))
    print('p-value: {:.4f}'.format(p_value))

for column in numeric_columns:
    plot_qqplot_with_hypothesis_test(df, consumption_utility=column)
```

```
import sklearn.model_selection, sklearn, seaborn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt

def plot_qqplot_with_hypothesis_test(data, consumption_utility):
    '''Plots a Q-Q plot and performs a hypothesis test for normality.
    Parameters:
    data: pandas.DataFrame, the dataset.
    consumption_utility: pandas.Series, the consumption utility.
    '''
    sample = data[consumption_utility]
    sample = sample.dropna()

    # Calculate the p-value.
    _, p_value = stats.shapiro(sample)

    # Create the Q-Q plot.
    sm.qqplot(sample, line='s')
    plt.title('Distribution Analysis: {}\n(Shapiro-Wilk Test, p-value = {:.4f})'.
    plt.xlabel('Theoretical Quantiles')
    plt.ylabel('Sample Quantiles')
    plt.show()

    # Print the results.
    print('Shapiro-Wilk test for {}:'.format(consumption_utility))
    print('p-value: {:.4f}'.format(p_value))

for column in numeric_columns:
    plot_qqplot_with_hypothesis_test(df, consumption_utility=column)
```




```
import sklearn.model_selection, sklearn, seaborn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt

def plot_qqplot_with_hypothesis_test(data, consumption_utility):
    '''Plots a Q-Q plot and performs a hypothesis test for normality.
    Parameters:
    data: pandas.DataFrame, the dataset.
    consumption_utility: pandas.Series, the consumption utility.
    '''
    sample = data[consumption_utility]
    sample = sample.dropna()

    # Calculate the p-value.
    _, p_value = stats.shapiro(sample)

    # Create the Q-Q plot.
    sm.qqplot(sample, line='s')
    plt.title('Distribution Analysis: {}\n(Shapiro-Wilk Test, p-value = {:.4f})'.
    plt.xlabel('Theoretical Quantiles')
    plt.ylabel('Sample Quantiles')
    plt.show()

    # Print the results.
    print('Shapiro-Wilk test for {}:'.format(consumption_utility))
    print('p-value: {:.4f}'.format(p_value))

for column in numeric_columns:
    plot_qqplot_with_hypothesis_test(df, consumption_utility=column)
```

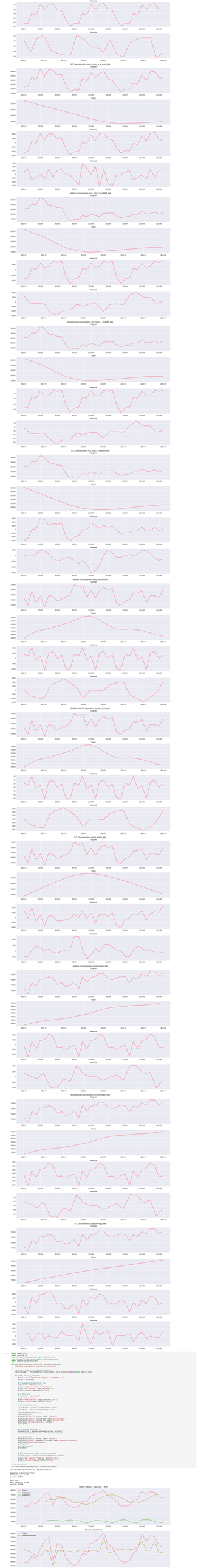
```
Shapiro-Wilk test for Consumption Utility:
p-value: 0.0000
Shapiro-Wilk test for Food Banking:
p-value: 0.0000
Shapiro-Wilk test for Business 1:
p-value: 0.0000
Shapiro-Wilk test for Mobile Laboratories:
p-value: 0.0000
Shapiro-Wilk test for Microbiology:
p-value: 0.0000
Shapiro-Wilk test for Clean Dry Good:
p-value: 0.0000
Shapiro-Wilk test for Research Support Facility:
p-value: 0.0000
Shapiro-Wilk test for Livestock Production College:
p-value: 0.0000
Shapiro-Wilk test for Hospital Library:
p-value: 0.0000
Shapiro-Wilk test for Energy Storage:
p-value: 0.0000
Shapiro-Wilk test for Nature Center:
p-value: 0.0000
Shapiro-Wilk test for Public Water Source:
p-value: 0.0000
Shapiro-Wilk test for Stadium College:
p-value: 0.0000
Shapiro-Wilk test for Combined College:
p-value: 0.0000
Shapiro-Wilk test for Retention Library:
p-value: 0.0000
Shapiro-Wilk test for Bailey Tie Fair:
p-value: 0.0000
Shapiro-Wilk test for Physical Education:
p-value: 0.0000
Shapiro-Wilk test for Area College Main:
p-value: 0.0000
Shapiro-Wilk test for Portable Marine Lab:
p-value: 0.0000
Shapiro-Wilk test for W-Magazine College:
p-value: 0.0000
Shapiro-Wilk test for Lithography Lab:
p-value: 0.0000
Shapiro-Wilk test for Jobing College:
p-value: 0.0000
Shapiro-Wilk test for Drying Building:
p-value: 0.0000
Shapiro-Wilk test for Microscopy College:
p-value: 0.0000
Shapiro-Wilk test for Lab Lab:
p-value: 0.0000
Shapiro-Wilk test for Wood College:
p-value: 0.0000
Shapiro-Wilk test for Survey Marine:
p-value: 0.0000
Shapiro-Wilk test for Center Tennis:
p-value: 0.0000
Shapiro-Wilk test for Consulting Residence:
p-value: 0.0000
Shapiro-Wilk test for Open Farm:
p-value: 0.0000
```

```
# Analysis of variance.
ANOVA_results = anova_oneway(df[numeric_columns],
                             df[consumption_utility])
ANOVA_results
```

```
Analysis of variance
    Variable  ...  Pr(>F)
0  Food Bank  ...  0.0000
1  Business 1  ...  0.0000
2  Mobile Laboratories  ...  0.0000
3  Microbiology  ...  0.0000
4  Clean Dry Good  ...  0.0000
5  Research Support Facility  ...  0.0000
6  Livestock Production College  ...  0.0000
7  Hospital Library  ...  0.0000
8  Energy Storage  ...  0.0000
9  Nature Center  ...  0.0000
10  Public Water Source  ...  0.0000
11  Stadium College  ...  0.0000
12  Combined College  ...  0.0000
13  Retention Library  ...  0.0000
14  Bailey Tie Fair  ...  0.0000
15  Physical Education  ...  0.0000
16  Area College Main  ...  0.0000
17  Portable Marine Lab  ...  0.0000
18  W-Magazine College  ...  0.0000
19  Lithography Lab  ...  0.0000
20  Jobing College  ...  0.0000
21  Drying Building  ...  0.0000
22  Microscopy College  ...  0.0000
23  Lab Lab  ...  0.0000
24  Wood College  ...  0.0000
25  Survey Marine  ...  0.0000
26  Center Tennis  ...  0.0000
27  Consulting Residence  ...  0.0000
28  Open Farm  ...  0.0000

[29 rows x 6 rows]
```

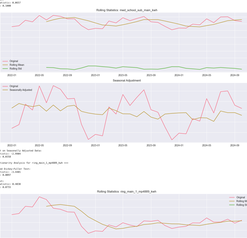
```
import sklearn.model_selection, sklearn, seaborn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
```

```
Trend: Increasing
Seasonal: Stable
Residual: Stable

Trend: Decreasing
Seasonal: Stable
Residual: Stable

adjusted_mean_temp_kam
Trend: Decreasing
Seasonal: Stable
Residual: Stable

Trend: Decreasing
Seasonal: Stable
Residual: Stable

adjusted_mean_temp_ctrl_kam:
Trend: Decreasing
Seasonal: Stable
Residual: Stable

Trend: Decreasing
Seasonal: Stable
Residual: Stable

grow_cycle_start_kam
Trend: Decreasing
Seasonal: Stable
Residual: Stable

Trend: Decreasing
Seasonal: Stable
Residual: Stable

grow_cycle_end_kam
Trend: Decreasing
Seasonal: Stable
Residual: Stable

Trend: Decreasing
Seasonal: Stable
Residual: Stable

observation_date_ctrl_kam_kam:
Trend: Decreasing
Seasonal: Stable
Residual: Stable

dates_temperature_kam:
Trend: Decreasing
Seasonal: Stable
Residual: Stable

Trend: Decreasing
Seasonal: Stable
Residual: Stable

Trend: Decreasing
Seasonal: Stable
Residual: Stable
```

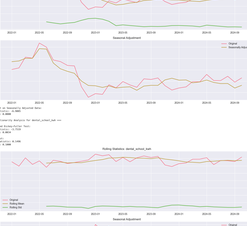



```
import pandas as pd
import numpy as np
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.tsa.seasonal import seasonal_decompose
import matplotlib.pyplot as plt

def perform_stationarity_analysis(df, columns=None,):
    if columns is None:
        columns = df.select_dtypes(include=[np.number]).columns

    print("\n" + "="*80)
    print("COMPREHENSIVE STATIONARITY ANALYSIS FOR TIME SERIES", sep=",")
    print("="*80)

    for column in df.columns:
        print("\n" + "="*80)
        print("\n" + "="*80)
        print("\n" + "="*80)
```

```
DF Test in Seasonality Between Base
p-value 0.000
Seasonality Present

>> Exponential Smoothing for med_school_lab_main_kwh >>

Exponential Smoothing Fitted Model
Smoothing Level 0.7000
p-value 0.000

ADFs Test
Seasonality Present
p-value 0.000
```

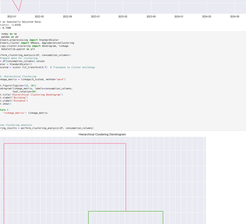



```
DF Test in Seasonality Between Base
p-value 0.000
Seasonality Present

>> Exponential Smoothing for lib_main_1_kwh0000_kwh >>

Exponential Smoothing Fitted Model
Smoothing Level 0.8000
p-value 0.000

ADFs Test
Seasonality Present
p-value 0.000
```




```
DF Test in Seasonality Between Base
p-value 0.000
Seasonality Present

>> Exponential Smoothing for dental_school_kwh >>

Exponential Smoothing Fitted Model
Smoothing Level 0.8000
p-value 0.000

ADFs Test
Seasonality Present
p-value 0.000
```

```
DF Test in Seasonality Between Base
p-value 0.000
Seasonality Present

>> Exponential Smoothing for physotherapy_kwh >>

Exponential Smoothing Fitted Model
Smoothing Level 0.8000
p-value 0.000

ADFs Test
Seasonality Present
p-value 0.000
```



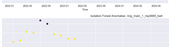

```
DF Test in Seasonality Between Base
p-value 0.000
Seasonality Present
```

```
>> import numpy as np
import pandas as pd
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics.cluster.bicluster import bicluster_ncut
from sklearn.metrics.cluster.bicluster import BiclusterEvaluator, SpectralCoclustering
from sklearn.cluster.bicluster import SpectralBiclustering, SpectralCoclustering

def plot_hierarchical_clustering(pandas_df, consumption_column):
    '''Function to plot hierarchical clustering
    X = df[consumption_column].values.reshape(-1, 1)
    Z = ward(pdist(X))
    fig.suptitle('Hierarchical Clustering Dendrogram')
    plt.xlabel('building')
    plt.ylabel('distance')
    for i, d, c in zip(ax.collections[::-1], Z[::-1, 2], Z[::-1, 3]):
        x = i.get_xdata()[0]
        y = d
        if y > 10:
            continue
        ax.annotate('%d' % c, (x, y), xytext=(0, -5),
                    textcoords='offset points',
                    va='top', ha='center')
    plt.show()
    return fig, ax

>> plot_hierarchical_clustering(df, consumption_column)
>> plot_hierarchical_clustering(df, consumption_column, 5)

def plot_kmeans_clustering(pandas_df, consumption_column, n_clusters=5):
    '''Function to plot k-means clustering
    X = df[consumption_column].values.reshape(-1, 1)
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit(X)
    print("Cluster centers:")
    print(kmeans.cluster_centers_)
    print("Cluster labels:")
    print(kmeans.labels_)
    plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='rainbow')
    plt.show()
    return kmeans.labels_

>> plot_kmeans_clustering(df, consumption_column)
>> plot_kmeans_clustering(df, consumption_column, n_clusters=3)

def plot_kmeans_elbow(pandas_df, consumption_column):
    '''Function to plot k-means elbow
    X = df[consumption_column].values.reshape(-1, 1)
    Sum_of_squared_distances = []
    K = range(1, 15)
    for k in K:
        km = KMeans(n_clusters=k)
        km = km.fit(X)
        Sum_of_squared_distances.append(km.inertia_)
    plt.plot(K, Sum_of_squared_distances, 'bx-')
    plt.xlabel('k')
    plt.ylabel('Sum_of_squared_distances')
    plt.show()
    return Sum_of_squared_distances

>> plot_kmeans_elbow(df, consumption_column)
>> plot_kmeans_elbow(df, consumption_column, n_clusters=3)

def plot_kmeans_silhouette(pandas_df, consumption_column):
    '''Function to plot k-means silhouette
    X = df[consumption_column].values.reshape(-1, 1)
    for n_clusters in range_n_clusters:
        clusterer = KMeans(n_clusters=n_clusters)
        preds = clusterer.fit_predict(X)
        centers = clusterer.cluster_centers_
        silhouette_avg = silhouette_score(X, preds)
        print("For n_clusters =", n_clusters,
              "The average silhouette_score is :", silhouette_avg)
    plt.show()

>> plot_kmeans_silhouette(df, consumption_column)
>> plot_kmeans_silhouette(df, consumption_column, range_n_clusters=[2, 3, 4, 5, 6])

def plot_hierarchical_clustering(pandas_df, consumption_column):
    '''Function to plot hierarchical clustering
    X = df[consumption_column].values.reshape(-1, 1)
    Z = ward(pdist(X))
    fig.suptitle('Hierarchical Clustering Dendrogram')
    plt.xlabel('building')
    plt.ylabel('distance')
    for i, d, c in zip(ax.collections[::-1], Z[::-1, 2], Z[::-1, 3]):
        x = i.get_xdata()[0]
        y = d
        if y > 10:
            continue
        ax.annotate('%d' % c, (x, y), xytext=(0, -5),
                    textcoords='offset points',
                    va='top', ha='center')
    plt.show()
    return fig, ax

>> plot_hierarchical_clustering(df, consumption_column)
>> plot_hierarchical_clustering(df, consumption_column, 5)

def plot_kmeans_clustering(df, consumption_column, n_clusters=5):
    '''Function to plot k-means clustering
    X = df[consumption_column].values.reshape(-1, 1)
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit(X)
    print("Cluster centers:")
    print(kmeans.cluster_centers_)
    print("Cluster labels:")
    print(kmeans.labels_)
    plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='rainbow')
    plt.show()
    return kmeans.labels_

>> plot_kmeans_clustering(df, consumption_column)
>> plot_kmeans_clustering(df, consumption_column, n_clusters=3)

def plot_kmeans_elbow(pandas_df, consumption_column):
    '''Function to plot k-means elbow
    X = df[consumption_column].values.reshape(-1, 1)
    Sum_of_squared_distances = []
    K = range(1, 15)
    for k in K:
        km = KMeans(n_clusters=k)
        km = km.fit(X)
        Sum_of_squared_distances.append(km.inertia_)
    plt.plot(K, Sum_of_squared_distances, 'bx-')
    plt.xlabel('k')
    plt.ylabel('Sum_of_squared_distances')
    plt.show()
    return Sum_of_squared_distances

>> plot_kmeans_elbow(df, consumption_column)
>> plot_kmeans_elbow(df, consumption_column, n_clusters=3)

def plot_kmeans_silhouette(pandas_df, consumption_column):
    '''Function to plot k-means silhouette
    X = df[consumption_column].values.reshape(-1, 1)
    for n_clusters in range_n_clusters:
        clusterer = KMeans(n_clusters=n_clusters)
        preds = clusterer.fit_predict(X)
        centers = clusterer.cluster_centers_
        silhouette_avg = silhouette_score(X, preds)
        print("For n_clusters =", n_clusters,
              "The average silhouette_score is :", silhouette_avg)
    plt.show()

>> plot_kmeans_silhouette(df, consumption_column)
>> plot_kmeans_silhouette(df, consumption_column, range_n_clusters=[2, 3, 4, 5, 6])

def plot_hierarchical_clustering(
```

```
Kmeans Distance Summary for reg_cons_1_sum:
Number of Clusters found: 20440[0]:
Number of Clusters: 1
Number of ANOMALOUS: 0
```

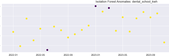
```
Kmeans Distance Summary for reg_school_sub_num_sum:
Number of Clusters found: 20440[0]:
Number of Clusters: 1
Number of ANOMALOUS: 0
```

```
Kmeans Distance Summary for reg_mass_1_rsp0000_sum:
Number of Clusters found: 20440[0]:
Number of Clusters: 1
Number of ANOMALOUS: 0
```

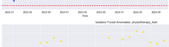


```
Kmeans Distance Summary for dental_school_sum:
Number of Clusters found: 20440[0]:
Number of Clusters: 1
Number of ANOMALOUS: 0
```

```
Kmeans Distance Summary for physiotherapy_sum:
Number of Clusters found: 20440[0]:
Number of Clusters: 1
Number of ANOMALOUS: 0
```

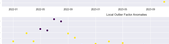


```
Kmeans Distance Summary for teachersmedy_sum:
Number of Clusters found: 20440[0]:
Number of Clusters: 1
Number of ANOMALOUS: 0
```




```
Kmeans Distance Summary for photo_building_sum:
Number of Clusters found: 20440[0]:
Number of Clusters: 1
Number of ANOMALOUS: 0
```

```
Kmeans Distance Summary for science_2_sum:
Number of Clusters found: 20440[0]:
Number of Clusters: 1
Number of ANOMALOUS: 0
```



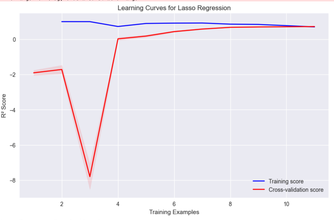

```
Kmeans Distance Summary for medic_laboratories_sum:
Number of Clusters found: 20440[0]:
Number of Clusters: 1
Number of ANOMALOUS: 0
```

Isolation Forest Anomalies: Hotel Energy Consumption
Isolation Forest Anomalies: Hotel Energy Consumption
Isolation Forest Anomalies: Hotel Energy Consumption

```
'test_r2': [-0.3990661635429173], -0.4630011595418187],
'train_rmse': [np.float64(235978.71783938391), np.float64(245593.19812864443)],
'test_rmse': [np.float64(290293.3421173867), np.float64(286933.11579159084)]}
```

In [98]:
```
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt

def plot_learning_curves(X, y, model, cv):
    """Plot learning curves for the model"""

    # Calculate learning curves
    train_sizes, train_scores, test_scores = learning_curve(
        model, X, y,
        cv=cv,
        n_jobs=-1,
        train_sizes=np.linspace(0.1, 1.0, 10),
        scoring='r2'
    )

    # Calculate mean and std
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)

    # Plot learning curves
    plt.figure(figsize=(10, 6))
    plt.plot(train_sizes, train_mean, label='Training score', color='blue')
    plt.fill_between(train_sizes,
                     train_mean - train_std,
                     train_mean + train_std,
                     alpha=0.1, color='blue')

    plt.plot(train_sizes, test_mean, label='Cross-validation score', color='red')
    plt.fill_between(train_sizes,
                     test_mean - test_std,
                     test_mean + test_std,
                     alpha=0.1, color='red')

    plt.xlabel('Training Examples')
    plt.ylabel('R² Score')
    plt.title('Learning Curves for Lasso Regression')
    plt.legend(loc='best')
    plt.grid(True)
    plt.show()

def perform_lasso_cv(df, consumption_columns):
    """Perform Lasso regression with 3-month rolling window and learning curves"""

    # Create month number mapping
    month_map = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6,
                 'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12}
    month_num = df['meter_reading_month'].map(month_map)

    # Prepare features
    X = pd.DataFrame({
        'rolling_mean_3': df['total_consumption'].rolling(3, min_periods=1).mean()
    })

    y = df['total_consumption']

    # Initialize Time Series CV
    tscv = TimeSeriesSplit(n_splits=2, test_size=11, gap=1)

    # Scale features for learning curves
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Create and plot learning curves
    model = Lasso(alpha=1.0, random_state=12)
    plot_learning_curves(X_scaled, y, model, tscv)

    # Store CV results
    cv_scores = {
        'train_r2': [],
        'test_r2': [],
        'train_rmse': [],
        'test_rmse': []
    }

    # Perform CV
    for train_idx, test_idx in tscv.split(X):
        X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
        y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        model = Lasso(alpha=1.0, random_state=12)
        model.fit(X_train_scaled, y_train)

        y_train_pred = model.predict(X_train_scaled)
        y_test_pred = model.predict(X_test_scaled)

        cv_scores['train_r2'].append(r2_score(y_train, y_train_pred))
        cv_scores['test_r2'].append(r2_score(y_test, y_test_pred))
        cv_scores['train_rmse'].append(np.sqrt(mean_squared_error(y_train, y_train_pred)))
        cv_scores['test_rmse'].append(np.sqrt(mean_squared_error(y_test, y_test_pred)))

    # Print results
    print("\nLasso Regression Results (3-month rolling window):")
    print(f"Average Train R²: {np.mean(cv_scores['train_r2']):.4f} (+/- {np.std(cv_scores['train_r2']):.4f})")
    print(f"Average Test R²: {np.mean(cv_scores['test_r2']):.4f} (+/- {np.std(cv_scores['test_r2']):.4f})")
    print(f"Average Train RMSE: {np.mean(cv_scores['train_rmse']):.2f} (+/- {np.std(cv_scores['train_rmse']):.2f})")
    print(f"Average Test RMSE: {np.mean(cv_scores['test_rmse']):.2f} (+/- {np.std(cv_scores['test_rmse']):.2f})")

    return cv_scores

# Execute Lasso regression
cv_scores = perform_lasso_cv(df, consumption_columns)
```

```
/Users/dhanaraijayasekaran/uems_project_clean/backend/venv/lib/python3.13/site-packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning: Objective did not converge. You might want to increase the n
umber of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 0.000e+00, tolerance: 0.000e+00
  model = cd_fast.enet_coordinate_descent(
/Users/dhanaraijayasekaran/uems_project_clean/backend/venv/lib/python3.13/site-packages/sklearn/metrics/_regression.py:1266: UndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
/Users/dhanaraijayasekaran/uems_project_clean/backend/venv/lib/python3.13/site-packages/sklearn/linear_model/_coordinate_descent.py:695: ConvergenceWarning: Objective did not converge. You might want to increase the n
umber of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 0.000e+00, tolerance: 0.000e+00
  model = cd_fast.enet_coordinate_descent(
/Users/dhanaraijayasekaran/uems_project_clean/backend/venv/lib/python3.13/site-packages/sklearn/metrics/_regression.py:1266: UndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
```




```
Lasso Regression Results (3-month rolling window):
Average Train R²: 0.7120 (+/- 0.0093)
Average Test R²: 0.7237 (+/- 0.0083)
Average Train RMSE: 231140.33 (+/- 3193.25)
Average Test RMSE: 225478.33 (+/- 840.36)
```

Lasso Regression Results (3-month rolling window): Average Train R²: 0.7120 (+/- 0.0093)

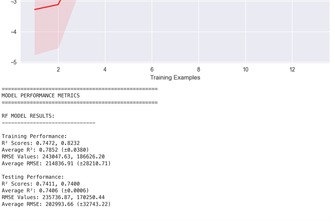
Average Test R²: 0.7237 (+/- 0.0083)

Average Train RMSE: 231140.33 (+/- 3193.25)

Average Test RMSE: 225478.33 (+/- 840.36)

In [102]:
```
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import r2_score, mean_squared_error
import numpy as np
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt

def create_features(df):
    """Create features with proper engineering to prevent overfitting"""

    month_map = {'Jan': 1, 'Feb': 2, 'Mar': 3, 'Apr': 4, 'May': 5, 'Jun': 6,
                 'Jul': 7, 'Aug': 8, 'Sep': 9, 'Oct': 10, 'Nov': 11, 'Dec': 12}

    # Create robust features
    X = pd.DataFrame({
        'month_cos': np.cos(2 * np.pi * df['meter_reading_month'].map(month_map)/12),
        'rolling_mean_3': df['total_consumption'].rolling(3, min_periods=1).mean()#,
        #'rolling_std_3': df['total_consumption'].rolling(6, min_periods=2).std()#,
        #'year': df['meter_reading_year']
    })

    return X

from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline

def plot_learning_curves(X, y, model, cv):
    """Plot learning curves for Random Forest model"""

    train_sizes, train_scores, test_scores = learning_curve(
        model, X, y,
        cv=cv,
        n_jobs=-1,
        train_sizes=np.linspace(0.1, 1.0, 10),
        scoring='r2'
    )

    # Calculate mean and std
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)

    # Plot learning curves
    plt.figure(figsize=(10, 6))
    plt.plot(train_sizes, train_mean, label='Training score', color='blue')
    plt.fill_between(train_sizes,
                     train_mean - train_std,
                     train_mean + train_std,
                     alpha=0.1, color='blue')

    plt.plot(train_sizes, test_mean, label='Cross-validation score', color='red')
    plt.fill_between(train_sizes,
                     test_mean - test_std,
                     test_mean + test_std,
                     alpha=0.1, color='red')

    plt.xlabel('Training Examples')
    plt.ylabel('R² Score')
    plt.title('Learning Curves for Random Forest')
    plt.legend(loc='best')
    plt.grid(True)
    plt.show()

def perform_ml_modeling(df, consumption_columns):
    """Perform Random Forest modeling with learning curves"""

    X = create_features(df)
    y = df['total_consumption']

    # Initialize Time Series CV
    tscv = TimeSeriesSplit(n_splits=2, test_size=9, gap=3)

    # Create RF pipeline
    rf_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='mean')),
        ('scaler', StandardScaler()),
        ('rf', RandomForestRegressor(
            n_estimators=100,
            max_depth=10,
            min_samples_split=5,
            min_samples_leaf=4,
            random_state=12
        )
    ])

    # Plot learning curves
    plot_learning_curves(X, y, rf_pipeline, tscv)

    # Rest of your existing code for metrics calculation
    results = {
        'rf': {'train_r2': [], 'test_r2': [], 'train_rmse': [], 'test_rmse': []}
    }

    # Perform CV and calculate metrics
    for train_idx, test_idx in tscv.split(X):
        X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
        y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

        rf_pipeline.fit(X_train, y_train)
        rf_train_pred = rf_pipeline.predict(X_train)
        rf_test_pred = rf_pipeline.predict(X_test)

        results['rf']['train_r2'].append(r2_score(y_train, rf_train_pred))
        results['rf']['test_r2'].append(r2_score(y_test, rf_test_pred))
        results['rf']['train_rmse'].append(np.sqrt(mean_squared_error(y_train, rf_train_pred)))
        results['rf']['test_rmse'].append(np.sqrt(mean_squared_error(y_test, rf_test_pred)))

    return format_model_results(results)

def format_model_results(results):
    """Format and display model results in a clear, organized way"""

    print("\n" + "="*50)
    print("MODEL PERFORMANCE METRICS")
    print("="*50)

    for model_name in ['rf']:
        print(f"\n{model_name.upper()} MODEL RESULTS:")
        print("-"*30)

        # Training Metrics
        print("\nTraining Performance:")
        print(f"R² Scores: {', '.join([f'{score:.4f}' for score in results[model_name]['train_r2']])}")
        print(f"Average R²: {np.mean(results[model_name]['train_r2']):.4f} (±{np.std(results[model_name]['train_r2']):.4f})")
        print(f"RMSE Values: {', '.join([f'{rmse:.2f}' for rmse in results[model_name]['train_rmse']])}")
        print(f"Average RMSE: {np.mean(results[model_name]['train_rmse']):.2f} (±{np.std(results[model_name]['train_rmse']):.2f})"

        # Testing Metrics
        print("\nTesting Performance:")
        print(f"R² Scores: {', '.join([f'{score:.4f}' for score in results[model_name]['test_r2']])}")
        print(f"Average R²: {np.mean(results[model_name]['test_r2']):.4f} (±{np.std(results[model_name]['test_r2']):.4f})"
        print(f"RMSE Values: {', '.join([f'{rmse:.2f}' for rmse in results[model_name]['test_rmse']])}")
        print(f"Average RMSE: {np.mean(results[model_name]['test_rmse']):.2f} (±{np.std(results[model_name]['test_rmse']):.2f})"


perform_ml_modeling(df,consumption_columns)
```

```
/Users/dhanaraijayasekaran/uems_project_clean/backend/venv/lib/python3.13/site-packages/sklearn/metrics/_regression.py:1266: UndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
/Users/dhanaraijayasekaran/uems_project_clean/backend/venv/lib/python3.13/site-packages/sklearn/metrics/_regression.py:1266: UndefinedMetricWarning: R^2 score is not well-defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
```




```
==================================================
MODEL PERFORMANCE METRICS
==================================================

RF MODEL RESULTS:
------------------------------

Training Performance:
R² Scores: 0.7472, 0.8232
Average R²: 0.7852 (±0.0380) RMSE Values: 243047.63, 186626.20 Average RMSE: 214836.91 (±28210.71)

Testing Performance:
R² Scores: 0.7411, 0.7400
Average R²: 0.7406 (±0.0006) RMSE Values: 235736.87, 170250.44 Average RMSE: 202993.66 (±32743.22)
```

========== MODEL PERFORMANCE METRICS

RF MODEL RESULTS:

Training Performance: R² Scores: 0.7472, 0.8232 Average R²: 0.7852 (±0.0380) RMSE Values: 243047.63, 186626.20 Average RMSE: 214836.91 (±28210.71)

Testing Performance: R² Scores: 0.7411, 0.7400 Average R²: 0.7406 (±0.0006) RMSE Values: 235736.87, 170250.44 Average RMSE: 202993.66 (±32743.22)