```
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from datetime import datetime, timedelta
from tqdm import tqdm
warnings.filterwarnings('ignore')
color_pal = sns.color_palette("husl", 9)
plt.style.use('fivethirtyeight')
weather_and_consumption_df = pd.read_csv('../data/processed/weather_and_consumption.csv',
index_col=0, parse_dates=True)
print("Loaded dataset preview:")
print(weather_and_consumption_df.head(1))
class BaseEnergyModel:
    def __init__(self, df, column_names, external_features, lags, window_sizes, n_estimators=600,
max_depth=3):
        self.df = df
        self.column_names = column_names
        self.external_features = external_features
        self.lags = lags
        self.window_sizes = window_sizes
        self.model = RandomForestRegressor(n_estimators=n_estimators, max_depth=max_depth)
        self.created_features = []
        self._create_features()
        self._train()
    def plot_feature_importance(self, top_n=10):
        if not hasattr(self, 'model') or not hasattr(self.model, 'feature_importances_'):
            print("Model must be trained before plotting feature importances.")
            return
        features = self.created_features + self.external_features
        importances = self.model.feature_importances_
        feature_data = pd.DataFrame({'Feature': features, 'Importance':
importances}).sort_values(by='Importance', ascending=False).head(top_n)
        plt.figure(figsize=(20, 5))
        sns.barplot(data=feature_data, x='Importance', y='Feature', palette='viridis')
        model_type = "Short-Term" if isinstance(self, ShortTermEnergyModel) else "Long-Term"
        plt.title(f'{model_type} Model: Top {top_n} Features', fontsize=16)
        plt.xlabel('Feature Importance', fontsize=12)
        plt.ylabel('Feature', fontsize=12)
        plt.tight_layout()
        plt.show()
    def _create_features(self):
        self.df['dayofweek'] = self.df.index.dayofweek
        self.created_features.append('dayofweek')

        for column_name in self.column_names:
            for lag in self.lags:
                feature_name = f"{column_name}_lag_{lag}"
                self.df[feature_name] = self.df[column_name].shift(lag)
                self.created_features.append(feature_name)

            for window in self.window_sizes:
                feature_name = f"{column_name}_rolling_mean_{window}"
```

```
                self.df[feature_name] = self.df[column_name].shift(1).rolling(window=window).mean()
                self.created_features.append(feature_name)
    def _train(self):
        features = self.created_features + self.external_features
        X_train = self.df[features].dropna()
        y_train = self.df[self.column_names[0]].loc[X_train.index]
        self.used_features = list(X_train.columns)
        self.model.fit(X_train, y_train)
    def predict_for_date(self, date):
        date = pd.to_datetime(date)
        if date not in self.df.index:
            print(f"No direct data available for {date}, prediction requires feature presence.")
            return None
        features_order = self.created_features + self.external_features
        X_test = self.df.loc[[date], features_order]

        if not X_test.empty:
            prediction = self.model.predict(X_test)
            return prediction[0]
        else:
            print("Features not available for prediction.")
            return None
class ShortTermEnergyModel(BaseEnergyModel):
    def __init__(self, df):
        super().__init__(df,
                         column_names=['total_consumption', 'day_length', 'dayofweek'],
                         external_features=['day_length'],
                         lags=[1, 2, 3, 4, 5, 6, 7, 30],
                         window_sizes=[2, 3, 4, 5, 6, 7, 30])
class LongTermEnergyModel(BaseEnergyModel):
    def __init__(self, df):
        super().__init__(df,
                         column_names=['total_consumption', 'day_length'],
                         external_features=['feelslike', 'temp', 'day_length', 'tempmax'],
                         lags=[30, 40, 365],
                         window_sizes=[])
short_term_model = ShortTermEnergyModel(weather_and_consumption_df)
short_term_model.plot_feature_importance(top_n=10)

date = '2010-11-25'
short_term_pred = short_term_model.predict_for_date(date)
print(f"Short-term prediction for {date}: {short_term_pred}")

long_term_model = LongTermEnergyModel(weather_and_consumption_df)
long_term_model.plot_feature_importance(top_n=10)
long_term_pred = long_term_model.predict_for_date(date)
print(f"Long-term prediction for {date}: {long_term_pred}")
# Comparison
date_range = pd.date_range(start='2009-05-17', end='2010-05-17', freq='D')
long_term_errors = np.zeros(len(date_range))
short_term_errors = np.zeros(len(date_range))
predictions = []
for i, date in enumerate(tqdm(date_range, desc='Testing Models')):
    date_str = date.strftime('%Y-%m-%d')
    long_term_prediction = long_term_model.predict_for_date(date_str)
```

```
    short_term_prediction = short_term_model.predict_for_date(date_str)
    real_value = weather_and_consumption_df.loc[date_str, 'total_consumption']

    long_term_errors[i] = long_term_prediction - real_value
    short_term_errors[i] = short_term_prediction - real_value

    predictions.append({
        'date': date,
        'long_term_prediction': long_term_prediction,
        'short_term_prediction': short_term_prediction,
        'real_value': real_value
    })
predictions_df = pd.DataFrame(predictions).set_index('date')
predictions_df[['real_value', 'long_term_prediction', 'short_term_prediction']].plot(figsize=(20, 6))
plt.xlabel('Date')
plt.ylabel('Consumption')
plt.title('Comparison of Long-Term and Short-Term Predictions with Real Values')
plt.grid(True)
plt.legend(['Real', 'Long-Term', 'Short-Term'])
plt.show()
long_term_mae = np.mean(np.abs(long_term_errors))
short_term_mae = np.mean(np.abs(short_term_errors))
print(f"Long-term MAE: {long_term_mae}")
print(f"Short-term MAE: {short_term_mae}")
```