```
import warnings ; warnings.filterwarnings(’ignore’)
import numpy as np
import pandas as pd ; pd.set_option(’display.max_columns’, None) ; pd.set_option(’display.max_rows’, 4)
import matplotlib.pyplot as plt ; import matplotlib.dates as mdates
import seaborn as sns ; color_pal = sns.color_palette("husl", 9) ; plt.style.use(’fivethirtyeight’)
import plotly.graph_objects as go
import plotly.express as px
import plotly.io as pio
from colorama import Fore
from sklearn.dummy import DummyRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import TimeSeriesSplit
import xgboost as xgb
from datetime import datetime, date
import math
import os
import re
import missingno as msno
from tqdm import tqdm
weather_and_consumption_df = pd.read_csv(’../data/processed/weather_and_consumption.csv’,
index_col=0, parse_dates=True)
weather_and_consumption_df.head(1)
weather_and_consumption_df.columns
df = weather_and_consumption_df.copy()
def create_features(df, column_names, lags, window_sizes):
    """
    Create time series features based on time series index and add lag and rolling features for specified
columns.
    """
    created_features = []
    basic_features = [’dayofweek’, ’quarter’, ’month’, ’year’, ’dayofyear’]
    for feature in basic_features:
        df[feature] = getattr(df.index, feature)
        created_features.append(feature)

    for column_name in column_names:
        for lag in lags:
            lag_feature_name = f’{column_name}_lag_{lag}’
            df[lag_feature_name] = df[column_name].shift(lag)
            created_features.append(lag_feature_name)
        for window in window_sizes:
            rolling_mean_name = f’{column_name}_rolling_mean_{window}’
            df[rolling_mean_name] = df[column_name].shift(1).rolling(window=window).mean()
            created_features.append(rolling_mean_name)

    return df, created_features
df, created_features = create_features(df,
                                       column_names=[’total_consumption’, ’Global_intensity’, ’Sub_metering_3’,
’Sub_metering_1’,
                                                     ’temp’, ’day_length’, ’tempmax’, ’feelslike’, ’feelslikemax’, ’feelslikemin’,
’tempmin’],
                                       lags=[1, 2, 3, 4, 5, 6, 7, 30, 90, 365],
                                       window_sizes=[2, 3, 4, 5, 6, 7, 30, 90, 365])
EXTERNAL_FEATURES = [’tempmax’, ’tempmin’, ’temp’, ’feelslikemax’, ’feelslikemin’, ’feelslike’, ’dew’,
                    ’humidity’, ’precip’, ’precipprob’, ’precipcover’, ’snow’, ’snowdepth’, ’windgust’, ’windspeed’,
                    ’winddir’, ’sealevelpressure’, ’cloudcover’, ’visibility’, ’moonphase’, ’conditions_clear’,
                    ’conditions_overcast’, ’conditions_partiallycloudy’, ’conditions_rain’,
’conditions_rainovercast’,
                    ’conditions_rainpartiallycloudy’, ’conditions_snowovercast’, ’conditions_snowpartiallycloudy’,

                    ’conditions_snowrain’, ’conditions_snowrainovercast’, ’conditions_snowrainpartiallycloudy’,
                    ’day_length’, ’is_holiday’]
FEATURES = created_features
TARGET = ’total_consumption’
df.tail(2)
threshold = ’2010-05-17’
train_df = df.loc[df.index < threshold].copy()
test_df = df.loc[df.index >= threshold].copy()
X_train = train_df[FEATURES+EXTERNAL_FEATURES]
y_train = train_df[TARGET]
X_test = test_df[FEATURES+EXTERNAL_FEATURES]
y_test = test_df[TARGET]
trace1 = go.Scatter(x=train_df.index, y=train_df.total_consumption, mode=’lines’, name=’Training Set’)
trace2 = go.Scatter(x=test_df.index, y=test_df.total_consumption, mode=’lines’, name=’Test Set’)
vline = go.layout.Shape(type="line", x0=threshold, y0=0, x1=threshold, y1=max(df.total_consumption),
                       line=dict(color="Black", width=2, dash="dash"))

layout = go.Layout(title=’Data Train/Test Split’,
                  xaxis=dict(title=’Date’),
                  yaxis=dict(title=’Total Consumption’),
                  shapes=[vline])

fig = go.Figure(data=[trace1, trace2], layout=layout)
fig.show()
X_train.columns
rfr = RandomForestRegressor(n_estimators=600, max_depth=3)
import xgboost as xgb
xgb = xgb.XGBRegressor(base_score=0.5, booster=’gbtree’,
                      n_estimators=1000,
                      early_stopping_rounds=50,
                      objective=’reg:linear’,
                      max_depth=3,
                      learning_rate=0.01)
rfr.fit(X_train, y_train)
xgb.fit(X_train, y_train, verbose=100,
        eval_set=[(X_train, y_train), (X_train, y_train)])
feature_data_rfr = pd.DataFrame({
    ’Feature’: X_train.columns,
    ’Importance’: rfr.feature_importances_,
    ’Model’: ’Random Forest’
})
feature_data_xgb = pd.DataFrame({
    ’Feature’: X_train.columns,
    ’Importance’: xgb.feature_importances_,
    ’Model’: ’XGBoost’
})
feature_data_combined = pd.concat([feature_data_rfr, feature_data_xgb])
top_features_rfr = feature_data_rfr.sort_values(by=’Importance’, ascending=False).head(10)
top_features_xgb = feature_data_xgb.sort_values(by=’Importance’, ascending=False).head(10)
```

```
fig, axs = plt.subplots(1, 2, figsize=(20, 6))
sns.barplot(data=top_features_rfr, x=’Importance’, y=’Feature’, palette=’viridis’, ax=axs[0])
axs[0].set_title(’Random Forest: Top 10 Features’, fontsize=16)
axs[0].set_xlabel(’Feature Importance’, fontsize=12)
axs[0].set_ylabel(’Feature’, fontsize=12)
sns.barplot(data=top_features_xgb, x=’Importance’, y=’Feature’, palette=’viridis’, ax=axs[1])
axs[1].set_title(’XGBoost: Top 10 Features’, fontsize=16)
axs[1].set_xlabel(’Feature Importance’, fontsize=12)
axs[1].set_ylabel(’Feature’, fontsize=12)
plt.suptitle("1-Day Future Prediction", fontsize=20)
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.savefig(’../results/top_features_comparison_1-Day_Future_Prediction.png’)
plt.show()
test_df[’RandomForest_Prediction’] = rfr.predict(X_test)
test_df[’XGBoost_Prediction’] = xgb.predict(X_test)
df_final = df.merge(test_df[[’RandomForest_Prediction’, ’XGBoost_Prediction’]], how=’left’,
left_index=True, right_index=True)
train_data = go.Scatter(x=train_df.index, y=train_df[’total_consumption’], mode=’lines’, name=’Train Data’,
line=dict(color=’Blue’))
test_data = go.Scatter(x=test_df.index, y=test_df[’total_consumption’], mode=’lines’, name=’Test Data’,
line=dict(color=’ForestGreen’))
random_forest_predictions = go.Scatter(x=df_final.index, y=df_final[’RandomForest_Prediction’],
mode=’markers’, name=’Random Forest Predictions’, marker=dict(color=’Red’))
xgboost_predictions = go.Scatter(x=df_final.index, y=df_final[’XGBoost_Prediction’], mode=’markers’,
name=’XGBoost Prediction Predictions’, marker=dict(color=’Orange’))
vline = dict(
    type="line", x0=threshold, y0=0, x1=threshold, y1=1, line=dict(color="Black", width=2, dash="dash"),
xref=’x’, yref=’paper’
)
layout = go.Layout(
    title="Real Data and Predictions Comparison for ’1-Day’ Future Prediction",
    xaxis=dict(title=’Index/Date’),
    yaxis=dict(title=’Total Consumption/Predictions’),
    legend_title=’Legend’,
    shapes=[vline]
)
fig = go.Figure(data=[train_data, test_data, random_forest_predictions, xgboost_predictions],
layout=layout)
fig.show()
y_pred_rfr = rfr.predict(X_test)
rmse_rfr = np.sqrt(mean_squared_error(y_test, y_pred_rfr))
print(f"Random Forest RMSE: {rmse_rfr}")
y_pred_xgb = xgb.predict(X_test)
rmse_xgb = np.sqrt(mean_squared_error(y_test, y_pred_xgb))
print(f"XGBoost RMSE: {rmse_xgb}")
pio.write_html(fig, file=’../results/Real Data and Predictions Comparison for 1-Day Future Prediction.html’)
df = weather_and_consumption_df.copy()
df, created_features = create_features(df,
                        column_names=[’total_consumption’, ’Global_intensity’, ’Sub_metering_3’,
’Sub_metering_1’,
                                              ’temp’, ’day_length’, ’tempmax’, ’feelslike’, ’feelslikemax’, ’feelslikemin’,
’tempmin’],
                        lags=[30, 40, 50, 60, 90, 365],
                        window_sizes=[])
FEATURES = created_features
```

```
EXTERNAL_FEATURES = [’tempmax’, ’tempmin’, ’temp’, ’feelslikemax’, ’feelslikemin’,
                     ’feelslike’, ’dew’, ’humidity’, ’precip’, ’precipprob’, ’precipcover’,
                     ’snow’, ’snowdepth’, ’windgust’, ’windspeed’, ’winddir’,
                     ’sealevelpressure’, ’cloudcover’, ’visibility’, ’moonphase’,
                     ’conditions_clear’, ’conditions_overcast’, ’conditions_partiallycloudy’,
                     ’conditions_rain’, ’conditions_rainovercast’,
                     ’conditions_rainpartiallycloudy’, ’conditions_snowovercast’,
                     ’conditions_snowpartiallycloudy’, ’conditions_snowrain’,
                     ’conditions_snowrainovercast’, ’conditions_snowrainpartiallycloudy’,
                     ’day_length’, ’is_holiday’]
TARGET = ’total_consumption’
threshold = ’2010-05-17’
train_df_cv = df.loc[df.index < threshold].copy()
test_df_cv = df.loc[df.index >= threshold].copy()
X_train_cv = train_df_cv[FEATURES+EXTERNAL_FEATURES]
y_train_cv = train_df_cv[TARGET]
X_test_cv = test_df_cv[FEATURES+EXTERNAL_FEATURES]
y_test_cv = test_df_cv[TARGET]
trace1 = go.Scatter(x=train_df_cv.index, y=train_df_cv.total_consumption, mode=’lines’, name=’Training
Set’)
trace2 = go.Scatter(x=test_df_cv.index, y=test_df_cv.total_consumption, mode=’lines’, name=’Test Set’)
vline = go.layout.Shape(type="line", x0=threshold, y0=0, x1=threshold, y1=max(df.total_consumption),
                        line=dict(color="Black", width=2, dash="dash"))
layout = go.Layout(title=’Data Train/Test Split’,
                   xaxis=dict(title=’Date’),
                   yaxis=dict(title=’Total Consumption’),
                   shapes=[vline])

fig = go.Figure(data=[trace1, trace2], layout=layout)
fig.show()
tss = TimeSeriesSplit(n_splits=7, test_size=30)
fig, axs = plt.subplots(7, 1, figsize=(20, 15), sharex=True)
color_palette = plt.get_cmap(’Set1’)
for fold, (train_idx, val_idx) in enumerate(tss.split(train_df_cv)):
    train_cv = df.iloc[train_idx]
    test_cv = df.iloc[val_idx]
    axs[fold].plot(train_cv.index, train_cv[TARGET], label=’Training Set’, linewidth=2,
color=color_palette(0))
    axs[fold].plot(test_cv.index, test_cv[TARGET], label=’Validation Set’, color=color_palette(1),
linewidth=2)
    axs[fold].axvline(test_cv.index.min(), color=’gray’, ls=’--’, lw=2)
    axs[fold].set_title(f’Fold {fold+1}’, fontsize=14, fontweight=’bold’)
    axs[fold].xaxis.set_major_locator(mdates.AutoDateLocator())
    axs[fold].xaxis.set_major_formatter(mdates.DateFormatter(’%Y-%m-%d’))
    plt.setp(axs[fold].xaxis.get_majorticklabels(), rotation=45, ha="right", fontsize=12)

    # Improve readability
    axs[fold].tick_params(axis=’y’, labelsize=12)
    axs[fold].grid(True, which=’major’, linestyle=’--’, linewidth=’0.5’, color=’gray’)
    axs[fold].legend(fontsize=12, loc=’upper left’)
fig.tight_layout(rect=[0.03, 0.03, 0.97, 0.95])
fig.subplots_adjust(hspace=0.4)
fig.text(0.5, 0.02, ’Date’, ha=’center’, va=’center’, fontsize=16, fontweight=’bold’)
fig.text(0.01, 0.5, ’Total Consumption’, ha=’center’, va=’center’, rotation=’vertical’, fontsize=16,
fontweight=’bold’)
```

```
fig.suptitle("7-Fold Time Series Split on Train Data", fontsize=24, fontweight=’bold’, y=0.98)
plt.savefig(’../results/7-Fold Time Series Split on Train Data.png’)

plt.show()
X_train_cv.columns
rfr_cv = RandomForestRegressor(n_estimators=600, max_depth=3)
import xgboost as xgb
xgb_cv = xgb.XGBRegressor(base_score=0.5, booster=’gbtree’,
                          n_estimators=2000,
                          early_stopping_rounds=50,
                          objective=’reg:squarederror’,
                          max_depth=3,
                          learning_rate=0.05)
preds = []
scores = []

for fold, (train_idx, val_idx) in tqdm(enumerate(tss.split(X_train_cv))):
    X_train_fold = X_train_cv.iloc[train_idx]
    y_train_fold = y_train_cv.iloc[train_idx]
    X_val_fold = X_train_cv.iloc[val_idx]
    y_val_fold = y_train_cv.iloc[val_idx]

    # Fit the Random Forest model
    rfr_cv.fit(X_train_fold, y_train_fold)
    # Predict on the validation set
    y_pred_rfr = rfr_cv.predict(X_val_fold)
    # Calculate and store the score for Random Forest
    score_rfr = np.sqrt(mean_squared_error(y_val_fold, y_pred_rfr))
    print(f"Fold {fold}: Random Forest Regressor RMSE = {score_rfr}")

    # Fit the XGBoost model with early stopping
    xgb_cv.fit(X_train_fold, y_train_fold, verbose=100,
               eval_set=[(X_val_fold, y_val_fold)])
    # Predict on the validation set using the best iteration
    y_pred_xgb = xgb_cv.predict(X_val_fold)
    # Calculate and store the score for XGBoost
    score_xgb = np.sqrt(mean_squared_error(y_val_fold, y_pred_xgb))
    print(f"Fold {fold}: XGBoost Regressor RMSE = {score_xgb}")
    preds.append({’RF’: y_pred_rfr, ’XGB’: y_pred_xgb})
    scores.append({’RF’: score_rfr, ’XGB’: score_xgb})
avg_score_rfr = np.mean([score[’RF’] for score in scores])
avg_score_xgb = np.mean([score[’XGB’] for score in scores])
print(f"Random Forest Regressor Average RMSE across all folds: {avg_score_rfr}")
print(f"XGBoost Regressor Average RMSE across all folds: {avg_score_xgb}")
rf_rmse_scores = [score[’RF’] for score in scores]
xgb_rmse_scores = [score[’XGB’] for score in scores]
folds = list(range(1, len(rf_rmse_scores) + 1))

# Plotting
plt.figure(figsize=(20, 4))
plt.plot(folds, rf_rmse_scores, marker=’o’, label=’Random Forest RMSE’)
plt.plot(folds, xgb_rmse_scores, marker=’s’, label=’XGBoost RMSE’)
plt.xlabel(’Fold’)
plt.ylabel(’RMSE’)
plt.title(’Evolution of RMSE across folds’)
plt.legend()
plt.grid(True)
plt.xticks(folds)
plt.savefig(’../results/Evolution of RMSE across folds.png’)
plt.show()
feature_data_rfr = pd.DataFrame({
    ’Feature’: X_train_cv.columns,
    ’Importance’: rfr_cv.feature_importances_
}).sort_values(by=’Importance’, ascending=False).head(10)

# Create DataFrame for XGBoost feature importances
feature_data_xgb = pd.DataFrame({
    ’Feature’: X_train_cv.columns,
    ’Importance’: xgb_cv.feature_importances_
}).sort_values(by=’Importance’, ascending=False).head(10)
fig, axs = plt.subplots(1, 2, figsize=(20, 6))
sns.barplot(data=feature_data_rfr, x=’Importance’, y=’Feature’, palette=’viridis’, ax=axs[0])
axs[0].set_title(’Random Forest: Top 10 Features after TimeSeries Cross val’, fontsize=16)
axs[0].set_xlabel(’Feature Importance’, fontsize=12)
axs[0].set_ylabel(’Feature’, fontsize=12)
sns.barplot(data=feature_data_xgb, x=’Importance’, y=’Feature’, palette=’viridis’, ax=axs[1])
axs[1].set_title(’XGBoost: Top 10 Features after TimeSeries Cross val’, fontsize=16)
axs[1].set_xlabel(’Feature Importance’, fontsize=12)
axs[1].set_ylabel(’Feature’, fontsize=12)
plt.suptitle("30-Day Future Prediction", fontsize=20)
plt.tight_layout()
plt.savefig(’../results/top_features_comparison_30-Day_Future_Prediction.png’)
plt.show()
test_df_cv[’RandomForest_Prediction_cv’] = rfr_cv.predict(X_test_cv)
test_df_cv[’XGBoost_Prediction_cv’] = xgb_cv.predict(X_test_cv)
df_final = df.merge(test_df_cv[[’RandomForest_Prediction_cv’, ’XGBoost_Prediction_cv’]], how=’left’,
left_index=True, right_index=True)
train_data = go.Scatter(x=train_df_cv.index, y=train_df_cv[’total_consumption’], mode=’lines’,
name=’Train Data’, line=dict(color=’Blue’))
test_data = go.Scatter(x=test_df_cv.index, y=test_df_cv[’total_consumption’], mode=’lines’, name=’Test
Data’, line=dict(color=’ForestGreen’))
random_forest_predictions = go.Scatter(x=df_final.index, y=df_final[’RandomForest_Prediction_cv’],
mode=’markers’, name=’Random Forest CV Predictions’, marker=dict(color=’Red’))
xgboost_predictions = go.Scatter(x=df_final.index, y=df_final[’XGBoost_Prediction_cv’], mode=’markers’,
name=’XGBoost Prediction CV Predictions’, marker=dict(color=’Orange’))
vline = dict(
    type="line", x0=threshold, y0=0, x1=threshold, y1=1, line=dict(color="Black", width=2, dash="dash"),
xref=’x’, yref=’paper’
)
layout = go.Layout(
    title="Real Data and Predictions Comparison for ’30-Day’ Future Prediction",
    xaxis=dict(title=’Index/Date’),
    yaxis=dict(title=’Total Consumption/Predictions’),
    legend_title=’Legend’,
    shapes=[vline]  # Adding the vertical line to the layout
)
fig = go.Figure(data=[train_data, test_data, random_forest_predictions, xgboost_predictions],
layout=layout)
fig.show()
y_pred_rfr_cv = rfr_cv.predict(X_test_cv)
```

```
rmse_rfr_cv = np.sqrt(mean_squared_error(y_test_cv, y_pred_rfr_cv))
print(f"Random Forest RMSE: {rmse_rfr_cv}")
y_pred_xgb_cv = xgb_cv.predict(X_test_cv)
rmse_xgb_cv = np.sqrt(mean_squared_error(y_test_cv, y_pred_xgb_cv))
print(f"XGBoost RMSE: {rmse_xgb_cv}")
pio.write_html(fig, file=’../results/Real Data and Predictions Comparison for 30-Day Future Prediction.html’)
```