import pandas as pd

import numpy as np

import matplotlib.pyplot as pp

import seaborn as sns

import numpy as np

import xgboost as xgb

from xgboost import XGBRegressor

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential, load\_model

from tensorflow.keras.models import load\_model

from tensorflow.keras.layers import InputLayer, LSTM, Dense

from tensorflow.keras.layers import \*

from tensorflow.keras.callbacks import ModelCheckpoint

from tensorflow.keras.losses import MeanSquaredError

from tensorflow.keras.metrics import RootMeanSquaredError

from tensorflow.keras.optimizers import Adam

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

from tensorflow.keras.callbacks import EarlyStopping

from statsmodels.tsa.seasonal import seasonal\_decompose

from sklearn.model\_selection import GridSearchCV

# Load the power generation dataset

powergen = pd.read\_csv(''/Users/user/Downloads/UKFuel.csv'')

powergen;

# Load the weather dataset

weather = pd.read\_csv('/Users/user/Downloads/UKWeather.csv')

weather;

weather.dropna(how='any',inplace=True)

weather;

#Convert the 'Date' column in weather\_data to a datetime object

weather['Date'] = pd.to\_datetime(weather['Date'], format='%b-%y')

# Get the start and end dates from the power generation dataset

start\_date = powergen['DATETIME'].min()

end\_date = powergen['DATETIME'].max()

# Generate timestamps between start and end dates with a 30-minute interval

timestamps = pd.date\_range(start=start\_date, end=end\_date, freq='30T')

# Convert the timestamps to a DataFrame

timestamp\_df = pd.DataFrame({'DATETIME': timestamps})

# Merge the timestamp DataFrame with the weather data

merged = pd.merge(timestamp\_df, weather, how='left', left\_on=timestamp\_df['DATETIME'].dt.strftime('%m-%d'),

right\_on=weather['Date'].dt.strftime('%m-%d'))

# Fill missing values with monthly averages

merged['Avg\_Temp'] = merged.groupby(merged['DATETIME'].dt.month)['Avg\_Temp'].transform(lambda x: x.fillna(x.mean()))

merged['Avg\_Heating\_Days'] = merged.groupby(merged['DATETIME'].dt.month)['Avg\_Heating\_Days'].transform(lambda x: x.fillna(x.mean()))

merged['Avg\_Windspeed'] = merged.groupby(merged['DATETIME'].dt.month)['Avg\_Windspeed'].transform(lambda x: x.fillna(x.mean()))

merged['Avg\_Daily\_Sunhrs'] = merged.groupby(merged['DATETIME'].dt.month)['Avg\_Daily\_Sunhrs'].transform(lambda x: x.fillna(x.mean()))

merged['Avg\_Monthly\_Rainfall'] = merged.groupby(merged['DATETIME'].dt.month)['Avg\_Monthly\_Rainfall'].transform(lambda x: x.fillna(x.mean()))

# Drop unnecessary columns

merged.drop(columns=['key\_0', 'Date'], inplace=True)

merged

# Rename the 'DATETIME' column to 'Date' in the powergen dataset

powergen.rename(columns={'DATETIME': 'Date'}, inplace=True)

# Convert the 'Date' column in the powergen dataset to datetime

powergen['Date'] = pd.to\_datetime(powergen['Date'])

# Remove timezone information from the 'Date' column in the weather dataset

powergen['Date'] = powergen['Date'].dt.tz\_localize(None)

powergen

# Rename the 'DATETIME' column to 'Date' in the weather dataset

merged.rename(columns={'DATETIME': 'Date'}, inplace=True)

# Remove timezone information from the 'Date' column in the weather dataset

merged['Date'] = merged['Date'].dt.tz\_localize(None)

merged

# Merge the datasets on the 'Date' column using left join

merged\_df = pd.merge(merged, powergen, on='Date', how='inner')

merged\_df.to\_csv(r'C:\Users\ENGR. HENRY\Documents\MSc Biz Analytics.csv', index=False)

merged\_df

merged\_df.info()

merged\_df.describe()

# Visualize the distribution of each feature using histograms

drop\_date = merged\_df.copy()

drop\_date.drop('Date',inplace=True,axis=1)

drop\_date.hist(figsize=(15, 10))

pp.tight\_layout()

pp.show()

DEMGEN\_series = merged\_df[['Date','GENERATION','TOTAL\_SYSTEM\_DEMAND','NET\_DEMAND']]

DEMGEN\_series.set\_index('Date', inplace=True) # Ensure to set inplace=True to modify the DataFrame in place

DEMGEN\_series.plot(style='-', figsize=(10,6), title='Demand Generation Over Time')

def generate\_trend\_features(df, y\_features):

# Hourly trend

df['hourly\_trend'] = df.index.hour

# Day of week trend

df['dayofweek\_trend'] = df.index.dayofweek

# Monthly trend

df['monthly\_trend'] = df.index.month

# Quarterly trend

df['quarterly\_trend'] = (df.index.month - 1) // 3 + 1

# Yearly trend

df['yearly\_trend'] = df.index.year

# Day of Year trend

df['dayofyear\_trend'] = df.index.dayofyear

# Generate boxplots for each y feature

trends = ['hourly\_trend', 'dayofweek\_trend', 'monthly\_trend', 'quarterly\_trend', 'yearly\_trend','dayofyear\_trend']

for trend in trends:

pp.figure(figsize=(10, 6))

for y\_feature in y\_features:

sns.boxplot(x=trend, y=df[y\_feature], data=df)

pp.title(f'Boxplot of {trend} vs {", ".join(y\_features)}')

pp.xlabel(trend)

pp.ylabel('Values')

pp.legend(y\_features)

pp.show()

# Call the function with your DataFrame and list of y features

y\_features = ['TOTAL\_SYSTEM\_DEMAND', 'NET\_DEMAND', 'GENERATION']

generate\_trend\_features(DEMGEN\_series, y\_features)

FCR\_series = merged\_df[['Date','FOSSIL','CARBON\_INTENSITY','RENEWABLE']]

FCR\_series.set\_index('Date', inplace=True)

FCR\_series.plot(style='-', figsize=(10,6), title='Fuel Sources Over Time')

DEMGEN\_series = merged\_df[['Date','GENERATION','TOTAL\_SYSTEM\_DEMAND','NET\_DEMAND']]

DEMGEN\_series.set\_index('Date', inplace=True) # Ensure to set inplace=True to modify the DataFrame in place

DEMGEN\_series.plot(style='-', figsize=(10,6), title='Demand Generation Over Time')

def generate\_trend\_features(df, y\_features):

# Hourly trend

df['hourly\_trend'] = df.index.hour

# Day of week trend

df['dayofweek\_trend'] = df.index.dayofweek

# Monthly trend

df['monthly\_trend'] = df.index.month

# Quarterly trend

df['quarterly\_trend'] = (df.index.month - 1) // 3 + 1

# Yearly trend

df['yearly\_trend'] = df.index.year

# Day of Year trend

df['dayofyear\_trend'] = df.index.dayofyear

# Generate boxplots for each y feature

trends = ['hourly\_trend', 'dayofweek\_trend', 'monthly\_trend', 'quarterly\_trend', 'yearly\_trend','dayofyear\_trend']

for trend in trends:

pp.figure(figsize=(10, 6))

for y\_feature in y\_features:

sns.boxplot(x=trend, y=df[y\_feature], data=df)

pp.title(f'Boxplot of {trend} vs {", ".join(y\_features)}')

pp.xlabel(trend)

pp.ylabel('Values')

pp.legend(y\_features)

pp.show()

# Call the function with your DataFrame and list of y features

y\_features = ['TOTAL\_SYSTEM\_DEMAND', 'NET\_DEMAND', 'GENERATION']

generate\_trend\_features(DEMGEN\_series, y\_features)

FCR\_series = merged\_df[['Date','FOSSIL','CARBON\_INTENSITY','RENEWABLE']]

FCR\_series.set\_index('Date', inplace=True)

FCR\_series.plot(style='-', figsize=(10,6), title='Fuel Sources Over Time')

def generate\_trend\_features(df, y\_features):

# Hourly trend

df['hourly\_trend'] = df.index.hour

# Day of week trend

df['dayofweek\_trend'] = df.index.dayofweek

# Monthly trend

df['monthly\_trend'] = df.index.month

# Quarterly trend

df['quarterly\_trend'] = (df.index.month - 1) // 3 + 1

# Yearly trend

df['yearly\_trend'] = df.index.year

# Day of Year trend

df['dayofyear\_trend'] = df.index.dayofyear

# Generate boxplots for each y feature

trends = ['hourly\_trend', 'dayofweek\_trend', 'monthly\_trend', 'quarterly\_trend', 'yearly\_trend','dayofyear\_trend']

for trend in trends:

pp.figure(figsize=(10, 6))

for y\_feature in y\_features:

sns.boxplot(x=trend, y=df[y\_feature], data=df)

pp.title(f'Boxplot of {trend} vs {", ".join(y\_features)}')

pp.xlabel(trend)

pp.ylabel('Values')

pp.legend(y\_features)

pp.show()

# Call the function with the DataFrame and list of y features

y\_features = ['FOSSIL','CARBON\_INTENSITY','RENEWABLE']

generate\_trend\_features(FCR\_series, y\_features)

WEATHCOND\_series = merged\_df[['Date','Avg\_Temp', 'Avg\_Heating\_Days', 'Avg\_Windspeed',

'Avg\_Daily\_Sunhrs', 'Avg\_Monthly\_Rainfall',]]

WEATHCOND\_series.set\_index('Date', inplace=True)

WEATHCOND\_series.plot(style='-', figsize=(10,6), title='Weather Conditions Over Time Time Series')

#Merge the datasets on the 'Date' column using left join

period\_df = pd.merge(merged\_df[['Date', 'Avg\_Temp', 'Avg\_Heating\_Days', 'Avg\_Windspeed',

'Avg\_Daily\_Sunhrs', 'Avg\_Monthly\_Rainfall', 'NET\_DEMAND',

'TOTAL\_SYSTEM\_DEMAND', 'ENGLAND\_WALES\_DEMAND', 'GENERATION', 'GAS',

'COAL', 'NUCLEAR', 'SCOT\_WIND', 'ENG\_WIND', 'UK\_WIND', 'WIND\_CAPACITY',

'HYDRO', 'IMPORTS', 'BIOMASS', 'OTHER', 'SOLAR', 'SOLAR\_CAP',

'CARBON\_INTENSITY', 'RENEWABLE', 'FOSSIL']], FCR\_series[['hourly\_trend','dayofweek\_trend','monthly\_trend','quarterly\_trend','yearly\_trend']], on='Date', how='inner')

period\_df

period\_df.drop(columns=['Date'],inplace=True)

period\_df

# Split the dataset into training and test sets (80% train, 20% test)

period\_df\_train, period\_df\_test = train\_test\_split(period\_df, test\_size=0.2, random\_state=42)

# Split the remaining data (20% of the original dataset) into validation set (50% for validation)

period\_df\_valid, period\_df\_test = train\_test\_split(period\_df\_test, test\_size=0.5, random\_state=42)

# Display the shapes of the resulting datasets

print("Training set shape:", period\_df\_train.shape)

print("Validation set shape:", period\_df\_valid.shape)

print("Test set shape:", period\_df\_test.shape)

# Define features and target

features = ['hourly\_trend', 'dayofweek\_trend', 'monthly\_trend', 'quarterly\_trend', 'yearly\_trend']

target = 'TOTAL\_SYSTEM\_DEMAND'

# Split features and target for train, test, and validation sets

X\_train = period\_df\_train[features]

y\_train = period\_df\_train[target]

X\_test = period\_df\_test[features]

y\_test = period\_df\_test[target]

X\_valid = period\_df\_valid[features]

y\_valid = period\_df\_valid[target]

# Train XGBoost model

xgb\_model = XGBRegressor()

xgb\_model.fit(X\_train, y\_train)

# Make predictions on test set

y\_pred\_test = xgb\_model.predict(X\_test)

# Calculate RMSE on test set

rmse\_test = mean\_squared\_error(y\_test, y\_pred\_test, squared=False)

print("RMSE on Test Set:", rmse\_test)

# Make predictions on validation set

y\_pred\_valid = xgb\_model.predict(X\_valid)

# Calculate RMSE on validation set

rmse\_valid = mean\_squared\_error(y\_valid, y\_pred\_valid, squared=False)

print("RMSE on Validation Set:", rmse\_valid)

# Define features and target

features = ['hourly\_trend', 'dayofweek\_trend', 'monthly\_trend', 'quarterly\_trend', 'yearly\_trend']

target = 'TOTAL\_SYSTEM\_DEMAND'

# Create XGBoost model

xgb\_model = XGBRegressor()

# Define parameter grid for grid search

param\_grid = {

'n\_estimators': [100, 200, 300],

'max\_depth': [3, 5, 7],

'learning\_rate': [0.05, 0.1, 0.2]

}

# Perform grid search with cross-validation

grid\_search = GridSearchCV(estimator=xgb\_model, param\_grid=param\_grid, cv=3, scoring='neg\_mean\_squared\_error')

grid\_search.fit(period\_df\_train[features], period\_df\_train[target])

# Get best parameters and best estimator

best\_params = grid\_search.best\_params\_

best\_estimator = grid\_search.best\_estimator\_

# Print best parameters

print("Best Parameters:", best\_params)

# Make predictions on test set with best estimator

y\_pred\_test = best\_estimator.predict(period\_df\_test[features])

# Calculate RMSE on test set

rmse\_test = mean\_squared\_error(period\_df\_test[target], y\_pred\_test, squared=False)

print("RMSE on Test Set:", rmse\_test)

# Make predictions on validation set with best estimator

y\_pred\_val = best\_estimator.predict(period\_df\_valid[features])

# Calculate RMSE on validation set

rmse\_val = mean\_squared\_error(period\_df\_valid[target], y\_pred\_valid, squared=False)

print("RMSE on Validation Set:", rmse\_val)

# Plot predictions vs actual values for validation set

pp.figure(figsize=(10, 6))

pp.plot(period\_df\_valid.index, period\_df\_valid[target], label='Actual')

pp.plot(period\_df\_valid.index, y\_pred\_valid, label='Predicted')

pp.xlabel('Date')

pp.ylabel('Total System Demand')

pp.title('XGBoost Model: Actual vs Predicted (Validation Set)')

pp.legend()

pp.show()

[00:37, 13/05/2024] Softwork: from sklearn.metrics import r2\_score

# Calculate R-squared on test set

r2\_test = r2\_score(period\_df\_test[target], y\_pred\_test)

print("R-squared on Test Set:", r2\_test)

# Calculate R-squared on validation set

r2\_val = r2\_score(period\_df\_valid[target], y\_pred\_valid)

print("R-squared on Validation Set:", r2\_val)

[00:37, 13/05/2024] Softwork: # Train an XGBoost model

xgb\_model = XGBRegressor()

xgb\_model.fit(period\_df\_train[features], period\_df\_train[target])

# Get feature importances from the trained XGBoost model

feature\_importances = xgb\_model.feature\_importances\_

# Create a DataFrame to store feature importances along with their corresponding names

importance\_df = pd.DataFrame({'Feature': features, 'Importance': feature\_importances})

# Sort the DataFrame by feature importance in descending order

importance\_df = importance\_df.sort\_values(by='Importance', ascending=False)

# Plot the feature importances using a bar chart

pp.figure(figsize=(10, 6))

pp.bar(importance\_df['Feature'], importance\_df['Importance'])

pp.xlabel('Feature')

pp.ylabel('Importance')

pp.title('Feature Importance')

pp.xticks(rotation=45)

pp.show()

[00:37, 13/05/2024] Softwork: weather\_related = ['Avg\_Temp', 'Avg\_Heating\_Days', 'Avg\_Windspeed','Avg\_Daily\_Sunhrs',

'Avg\_Monthly\_Rainfall', 'NET\_DEMAND','TOTAL\_SYSTEM\_DEMAND', 'GENERATION',

'SCOT\_WIND', 'ENG\_WIND', 'UK\_WIND', 'HYDRO', 'IMPORTS', 'BIOMASS', 'OTHER',

'SOLAR','CARBON\_INTENSITY', 'RENEWABLE']

# Compute correlation matrix

correlation\_matrix = merged\_df[weather\_related].corr()

# Visualize correlations using heatmaps

pp.figure(figsize=(10, 6))

sns.heatmap(correlation\_matrix, annot=True, cmap='rainbow', fmt=".2f")

pp.title('Correlation Matrix')

pp.show()