Demand Forecasting Project

1. Download electricityLoadData.csv from the shared box and upload to your own google drive
2. Instructions for coding (What you need to code):
   - Load data and preprocess data using the predefined functions.
   - Go to `build_model` section and define your own network.
   - Choose your parameters to set up the training routine.
   - Plot your result by using the `Analyze Result` section.
   - Print your notebook by clicking File > Save > PDF to upload your work.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.metrics import mean_absolute_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, LSTM
import time

plt.style.use('fivethirtyeight')
FILE_PATH = "electricityLoadData.csv"
WINDOW = 48
```

In [ ]:
```
### Data Loading and Preprocessing Functions ###
def read_data(file_path):
    data = pd.read_csv(file_path, header=1, error_bad_lines=False)
    data.drop(data.columns[[2]], axis=1, inplace=True) # drop year
    return data

def normalize_data(dataset):
    # minmax normalize
    values = dataset.values
    minima_demand = np.amin(values[:, -1])
    maxima_demand = np.amax(values[:, -1])
    scaling_parameter_demand = maxima_demand - minima_demand
    for i in range(values.shape[1]):
        values[:, i] = (values[:, i]-np.amin(values[:, i]))/(np.amax(values[:, i])-np.amin(values[:, i]))
    return minima_demand, maxima_demand, scaling_parameter_demand, pd.DataFrame(values)
```

```
def prepare_data(dataset, window_size):
    amount_of_features = len(dataset.columns)
    data = dataset.values
    sequence_length = window_size + 1
    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index: index + sequence_length])
    windowed_mat = np.array(result)

    train_split = int(round(0.8 * windowed_mat.shape[0]))
    x_train = windowed_mat[:train_split, :-1]
    y_train = windowed_mat[:train_split, -1][:,-1]
    x_test = windowed_mat[train_split:, :-1]
    y_test = windowed_mat[train_split:, -1][:,-1]
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], amount_of_features))
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], amount_of_features))
    return x_train, y_train, x_test, y_test
```

In [ ]:

```
### Load and Preprocess Data ###
dataset = read_data(FILE_PATH)
min_demand, max_demand, demand_scaling_param, dataset = normalize_data(dataset)
X_train, y_train, X_test, y_test = prepare_data(dataset[::-1], WINDOW)
print(f"X_train shape: {X_train.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
C:\Users\LAPTOP WORLD\AppData\Local\Temp\ipykernel_31652\811291537.py:3: FutureWarning: The error_bad_lines argument ha
s been deprecated and will be removed in a future version. Use on_bad_lines in the future.


  data = pd.read_csv(file_path, header=1, error_bad_lines=False)
X_train shape: (56062, 48, 5)
y_train shape: (56062,)
X_test shape: (14016, 48, 5)
y_test shape: (14016,)
```

In [ ]:

```
### Model Definition and Training ###
model = Sequential([
    LSTM(256, activation='relu', input_shape=(48,5)),
    Dense(128),
    Dense(64),
    Dense(32),
    Dense(1)
```

```
])

model.compile(optimizer='adam', loss='mse')

print("Model Summary:")
model.summary()

model.fit(
    X_train,
    y_train,
    batch_size=128,
    epochs=15,
    validation_split=0.2,
    verbose=2)
```

```
Model Summary:
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 256)               268288

 dense (Dense)               (None, 128)               32896

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 32)                2080

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 311,553
Trainable params: 311,553
Non-trainable params: 0
_________________________________________________________________
Epoch 1/15
351/351 - 144s - loss: 0.0040 - val_loss: 8.0204e-04 - 144s/epoch - 409ms/step
Epoch 2/15
351/351 - 129s - loss: 3.8311e-04 - val_loss: 3.3722e-04 - 129s/epoch - 367ms/step
Epoch 3/15
351/351 - 126s - loss: 2.4504e-04 - val_loss: 2.3838e-04 - 126s/epoch - 358ms/step
Epoch 4/15
351/351 - 120s - loss: 1.7694e-04 - val_loss: 1.8365e-04 - 120s/epoch - 341ms/step
Epoch 5/15
351/351 - 131s - loss: 1.5777e-04 - val_loss: 1.8929e-04 - 131s/epoch - 372ms/step
Epoch 6/15
351/351 - 129s - loss: 1.7274e-04 - val_loss: 1.8566e-04 - 129s/epoch - 369ms/step
Epoch 7/15
351/351 - 122s - loss: 1.4102e-04 - val_loss: 1.1944e-04 - 122s/epoch - 348ms/step
Epoch 8/15
351/351 - 85s - loss: 1.3672e-04 - val_loss: 2.2293e-04 - 85s/epoch - 243ms/step
Epoch 9/15
351/351 - 36s - loss: 1.3271e-04 - val_loss: 3.0813e-04 - 36s/epoch - 102ms/step
Epoch 10/15
351/351 - 36s - loss: 1.2310e-04 - val_loss: 1.2215e-04 - 36s/epoch - 102ms/step
Epoch 11/15
351/351 - 35s - loss: 1.1153e-04 - val_loss: 1.4057e-04 - 35s/epoch - 101ms/step
Epoch 12/15
351/351 - 35s - loss: 1.2542e-04 - val_loss: 1.0966e-04 - 35s/epoch - 100ms/step
Epoch 13/15
```

```
351/351 - 35s - loss: 1.0043e-04 - val_loss: 2.1080e-04 - 35s/epoch - 99ms/step
Epoch 14/15
351/351 - 35s - loss: 1.1307e-04 - val_loss: 1.2785e-04 - 35s/epoch - 99ms/step
Epoch 15/15
351/351 - 35s - loss: 1.3255e-04 - val_loss: 1.4750e-04 - 35s/epoch - 100ms/step
```

Out[ ]:

```
<keras.callbacks.History at 0x22f7d8fb4f0>
```

In [ ]:

```python
def denormalize_data(data, scaling_parameter, minima):
    """Denormalizes the input data."""
    return (data * scaling_parameter) + minima

def calculate_mape(y_true, y_pred):
    """Calculates the mean absolute percentage error."""
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

def plot_forecast(actual, predicted, time_window=24):
    """Plots the actual vs predicted forecast."""
    plt.plot(actual[:time_window], label='Actual', color='blue')
    plt.plot(predicted[:time_window], label='Predicted', color='red')
    plt.xlabel('Time Sequence')
    plt.ylabel('Load (MW)')
    plt.title(f'Actual vs Predicted Results in the Next {time_window * 1800} Seconds')
    plt.legend()
    plt.show()
```

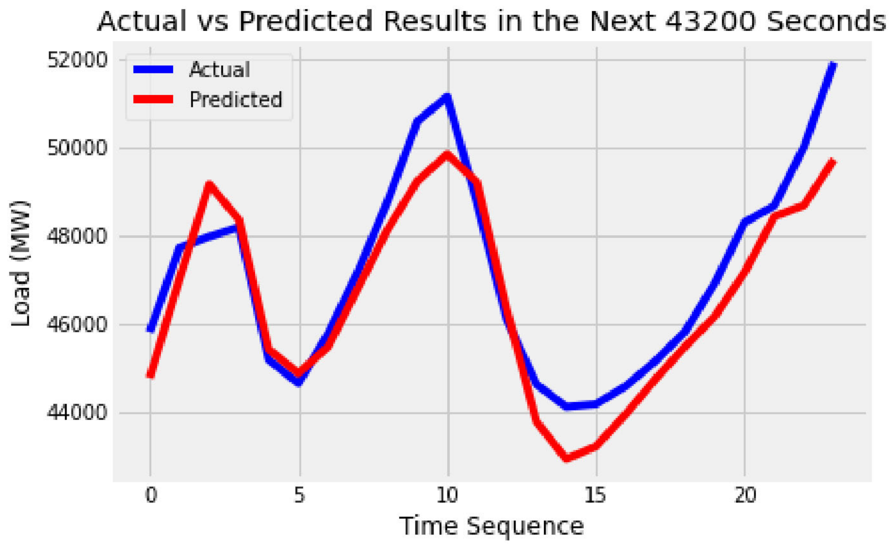
In [ ]:

```python
# Get the predicted and actual data
predicted = denormalize_data(model.predict(X_test), demand_scaling_param, min_demand)
actual = denormalize_data(y_test, demand_scaling_param, min_demand)

# Calculate and print the mean absolute percentage error and mean absolute error
mape = calculate_mape(actual, predicted)
mae = mean_absolute_error(actual, predicted)
print(f'Test MAPE: {mape:.3f}')
print(f'Test MAE: {mae:.3f}')

#Plot the forecast
plot_forecast(actual, predicted)
```

```
438/438 [==============================] - 10s 23ms/step
Test MAPE: 27.889
Test MAE: 607.329
```


Actual vs Predicted Results in the Next 43200 Seconds
Actual
Predicted
Load (MW)
52000
50000
48000
46000
44000
0
5
10
15
20
Time Sequence