**Web application to compare machine learning models.**

Client: Professor Kenneth Fletcher.

**React components:**  
**App.js :**   
Overview :  
This code defines a React component called App that serves as the main entry point for the application. It renders a set of routes using the Routes component from the react-router-dom library, allowing the user to navigate between different pages or views within the app based on the URL path. The App component also imports a number of UI components from the @material-ui/core library and icon components from the @material-ui/icons and @mui/icons-material libraries, which are used to build the layout and appearance of the app.

**Adminsignin.js :**

Overview :  
The Adminsignin component is a React component that renders a sign in form for an admin user. The form includes TextField components for the user to enter their username and password, a Checkbox component for remembering the login information, and a Button component to submit the form. The form also includes Link components for the user to sign up or go to the home page if they are not an admin.  
  
Usage  
To use the Home component in another file, import it like so:  
import Adminsignin from './Adminsignin';  
  
Then, you can render the component by including it in your JSX code  
<Adminsignin />  
  
  
**AdminActivities:**

Overview:

The Adminactivities component is a React component that provides a landing page for the admin user, where they can view and access various admin activities. The page includes a welcome message and buttons for adding or deleting deep learning models. The component imports a number of UI components from the @material-ui/core library, as well as the ApiTwoToneIcon icon component from the @mui/icons-material library. The component also uses the makeStyles hook from @material-ui/core/styles to create styles in a declarative way, and the green color from @material-ui/core/colors to style the buttons.

Usage  
To use the Home component in another file, import it like so:  
import Adminactivities from './Adminactivities';  
Then, you can render the component by including it in your JSX code

<Adminactivities />  
  
  
  
**Addmodel.js :**   
  
Overview  
The AddModel component is a React component that provides a form for the admin user to add a new deep learning model to the app. The form includes TextField components for the user to enter the model name and a description, and a Button component to submit the form. The component imports a number of UI components from the @material-ui/core library, as well as the LockOutlinedIcon icon component from the @material-ui/icons library and the FormControlLabel and Checkbox components for rendering a form control label and checkbox input.  
Usage  
To use the Home component in another file, import it like so:  
import Addmodel from './Addmodel';  
  
Then, you can render the component by including it in your JSX code  
<Addmodel />  
  
  
  
  
**Signup.js**  
Overview  
The signup component is a functional React component that provides a form for users to sign up for an account. The form includes TextField components for the user to enter their email, password, password confirmation, and telephone, and a Button component to submit the form. The component imports a number of UI components from the @material-ui/core library, as well as the LockOutlinedIcon icon component from the @material-ui/icons library and the FormControlLabel and Checkbox components for rendering a form control label and checkbox input. The component also uses the axios library to send a POST request to the server with the form data when the form is submitted.  
  
State  
The signup component uses the useState hook to manage the following state variables:  
  
formData: An object that holds the form data, including the email, password, password confirmation, and telephone entered by the user.  
Functions  
The signup component includes the following function:  
  
handleSubmit: A function that is called when the form is submitted. It prevents the default form submission behavior and sends a POST request to the /api/users endpoint with the form data.  
Usage  
To use the signup component in another file, import it like so:  
import signup from './signup';  
Then, you can render the component by including it in your JSX code:  
<signup />  
The component will render a form for the user to sign up for an account. When the form is submitted, the handleSubmit function will be called and a POST request will be sent to the server with the form data.  
  
  
  
**Home.js :**

Overview  
The Homepage component is a functional React component that displays information about machine learning models and allows users to compare them by selecting and uploading relevant datasets. It uses Material-UI components for styling and layout, and also includes custom images and icons. The component makes use of the makeStyles hook from Material-UI to apply styles to various elements, and includes buttons and form elements for interacting with the page.  
  
  
Usage  
To use the Home component in another file, import it like so:  
import Home from './Home';  
  
Then, you can render the component by including it in your JSX code  
<Home />

**Express Server**

**App.js** :

Overview

The script imports the express library, which is used to create an HTTP server and handle HTTP requests. It then creates an express app, which will be used to configure the server and handle routes.  
  
The mongoose library is imported and used to connect to a MongoDB database. The bcrypt library is imported and used to hash passwords for storing them securely in the database. The cors library is imported and used to enable cross-origin resource sharing, allowing the server to accept requests from any origin.  
  
The cors middleware is used to allow requests from any origin, and the express.json middleware is used to parse incoming request bodies as JSON.  
  
The script connects to the MongoDB database using the mongoose library and the provided connection string. If the connection is successful, it logs a message to the console. If the connection fails, it logs an error.  
  
The script requires the "userDetails" module and uses the mongoose connection to get the "userInformation" model. The model is used to create and query documents in the "userinformation" collection in the database.  
  
Finally, the script defines a route for handling user sign up. When a POST request is made to the "/Signup" route, the function inside the route handler is executed. The function gets the email and password from the request body, hashes the password using bcrypt, checks if a user with the same email already exists in the database, and creates a new user with the provided email and encrypted password if one does not exist. It then sends a response indicating the success or failure of the operation.

**UserDetails.js :**

Overview

This is a Mongoose schema for storing user information in a MongoDB database.

It defines a collection called user information with two fields: email and password.

The email field is set to be unique.

**Streamlit .py**

The code is using Streamlit to build a user interface for a machine learning model comparison. It first imports several libraries, including streamlit, torch, pandas, and numpy, and sets the device to cuda if a GPU is available, or cpu if it is not.

It then sets up containers for different sections of the user interface using the st.container() function.

The code then has a section where the user can upload a file using the st.file\_uploader() function. If a file is successfully uploaded, it is read as a CSV file using pd.read\_csv() and displayed using st.write().

The code then defines a function generate\_time\_lags() that takes in a dataframe and the number of lags and generates new columns for the specified number of time lags for the value column. It returns a copy of the dataframe with the added time lag columns.

The code then defines input\_dim as 100 and generates time lags for the energy consumption data using generate\_time\_lags(). It also generates additional features for the energy consumption data by one-hot encoding the month, day, day\_of\_week, and week\_of\_year columns and adding them to the dataframe.

The code then defines a function generate\_cyclical\_features() that takes in a dataframe, the name of a column, the period of the cyclical data, and the number of sin and cosine transformations to generate. It returns a copy of the dataframe with the added sin and cosine transformation columns.

Finally, the code defines a function generate\_lagged\_features\_and\_targets() that takes in a dataframe, the number of lagged features to include, and the number of steps to forecast. It returns two dataframes, one containing the lagged features and the other containing the targets (forecasted values). It also scales the data using StandardScaler() from sklearn.preprocessing.

**LSTM Module:**

This is a PyTorch implementation of a Long Short-Term Memory (LSTM) model. LSTMs are a type of recurrent neural network that are particularly effective at learning and remembering patterns over long periods of time, making them well-suited for tasks such as language modeling, speech recognition, and machine translation.

The **LSTMModel** class extends PyTorch's **nn.Module** class and serves as a constructor for LSTM models. It has two methods: **\_\_init\_\_** and **forward**. The **\_\_init\_\_** method initializes the model with the specified input parameters, while the **forward** method defines how the forward propagation should be calculated. PyTorch automatically handles backpropagation, so there is no need to define a separate method for it.

The **LSTMModel** class has two attributes: **hidden\_dim** and **layer\_dim**. **hidden\_dim** is the number of nodes in each layer of the LSTM, while **layer\_dim** is the number of layers in the network. It also has two sub-modules: an **nn.LSTM** module (called **lstm**) and an **nn.Linear** module (called **fc**). The **nn.LSTM** module handles the actual LSTM calculations, while the **nn.Linear** module is a fully-connected layer that converts the final state of the LSTM to the desired output shape.

In the **forward** method, the input tensor **x** is passed through the LSTM and the final state is passed through the fully-connected layer. The output is then returned. The initial hidden state and cell state of the LSTM are set to zero and passed along with the input tensor through the **self.lstm** module. The output and final state of the LSTM are returned, and the final state is reshaped and passed through the fully-connected layer before being returned as the final output of the model.

Top of Form

Bottom of Form

**GRU Module:**

This code defines a class GRUModel which is a subclass of PyTorch's nn.Module class. The GRUModel class contains a constructor method \_init\_ and a forward propagation method forward. The \_init\_ method is used to initialize the GRU model with the given input parameters. It takes in 5 arguments:

input\_dim: an integer representing the number of nodes in the input layer

hidden\_dim: an integer representing the number of nodes in each hidden layer

layer\_dim: an integer representing the number of hidden layers in the network

output\_dim: an integer representing the number of nodes in the output layer

dropout\_prob: a float representing the probability of nodes being dropped out during training

The \_init\_ method then initializes the attributes of the GRUModel instance:

layer\_dim: the number of hidden layers in the network

hidden\_dim: the number of nodes in each hidden layer

gru: a GRU model constructed with the input parameters

fc: a fully connected layer to convert the final state of the GRUs to the desired output shape

The forward method defines how the forward propagation is calculated. It takes in an input tensor x of shape (batch size, sequence length, input\_dim) and returns an output tensor of shape (batch size, output\_dim). It does this by first initializing the hidden state for the first input with zeros, then doing forward propagation by passing in the input and hidden state into the model. It reshapes the outputs in the shape of (batch\_size, seq\_length, hidden\_size) and passes it through the fully connected layer to convert the final state to the desired output shape.

The get\_model function is a helper function that takes in a string representing the type of model (either "gru" or "lstm") and a dictionary of model parameters and returns an instance of the corresponding model class initialized with the given parameters.

**Optimization Class:**

The Optimization class is a helper class that is used to train, validate, and predict with a given model. It takes a model, a loss function, and an optimizer as input, and provides a framework to train and validate the model, as well as to make predictions.

The \_init\_ method initializes the model, loss function, optimizer, and empty lists for train and validation losses.

The train\_step method takes in features (x) and target values (y) tensors and performs one step of training. It sets the model to train mode, makes predictions, computes the loss, computes gradients, and updates the model's parameters using the optimizer. It returns the loss value.

The train method takes in DataLoaders for the training and validation datasets, as well as the batch size, number of epochs, and number of features. It trains the model for a specified number of epochs by iteratively calling the train\_step method. It also has optional arguments for an early stopping condition and a learning scheduler. Finally, it saves the trained model to a file.

The validate method takes in a DataLoader for the validation data and calculates the validation loss using the loss function. It returns the validation loss.The predict method takes in a tensor of features and returns the model's predictions.

The code then defines a function plot\_dataset() that uses Plotly to plot a dataset. This function is used to plot the energy consumption data. The data is first transformed to set the index to the Datetime column and rename the PJME\_MW column to value. The data is then plotted using st.plotly\_chart().