CPSC393 Final Project

A comprehensive stock market prediction system utilizing an ensemble of machine learning techniques, including LSTM, GRU, ARIMA, Transformer, and Prophet models. Designed to analyze and predict stock prices, this project showcases a comparative study of different algorithms and their effectiveness in predicting financial market trends, while leveraging the power of ensemble methods for improved accuracy and robustness.

Prerequisites

Python 3.8+
Pandas
NumPy
Scikit-learn
Scikit-optimize
TensorFlow
Keras
Statsmodels
Matplotlib
Docker

Setup and Usage

This section combines the instructions for installing Docker and using the Jupyter notebook 'Testing.ipynb' to train and evaluate the models.

Docker Installation

Install Docker on your machine if you haven't already.
Docker Desktop Download

Building the Docker Image

To build the Docker image, navigate to the project directory and run the following command:

```
docker build -t cpsc393_stock_pred .
```

Running the Docker Container

After building the Docker image, you can run the project using the following command:

```
docker run -it --rm -p 8000:8000 cpsc393_stock_pred
```

A link will be given in the terminal. Copy and paste the link into your browser to access the Jupyter notebook. The testing notebook is called 'Testing.ipynb'.

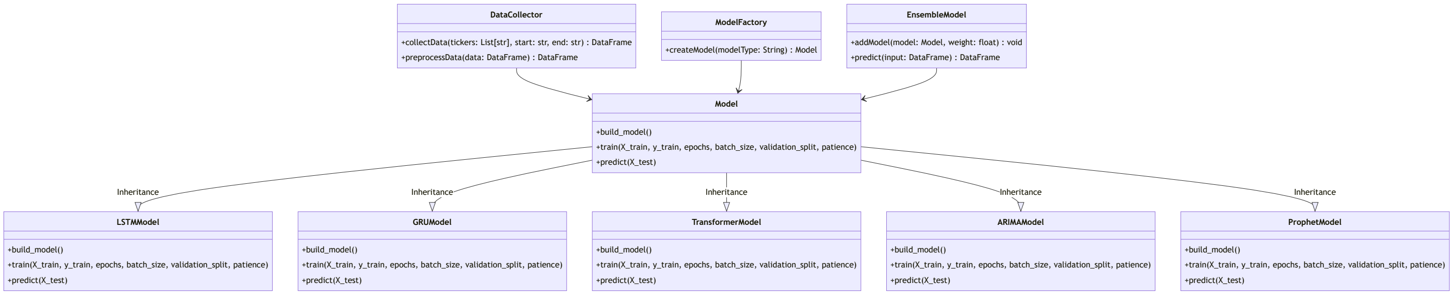
Shutting Down the Docker Container

To shut down the Docker container, press Ctrl+C in the terminal.

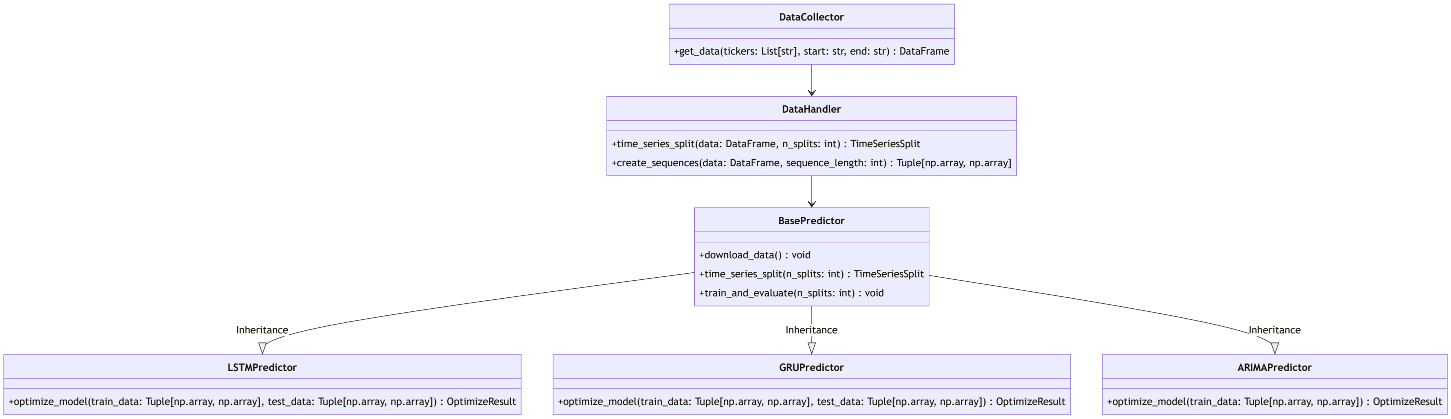
- **Always remember to shut down the Docker container** when you are done using it. If you do not shut down the container, it will continue to run in the background and use up your computer's resources.

Class Diagram

Originally, we had a class diagram that looked like this:


DataCollector
+collectData(tickers: List[str], start: str, end: str) : DataFrame
+preprocessData(data: DataFrame) : DataFrame
ModelFactory
+createModel(modelType: String) : Model
EnsembleModel
+addModel(model: Model, weight: float) : void
+predict(input: DataFrame) : DataFrame
Model
+build_model()
+train(X_train, y_train, epochs, batch_size, validation_split, patience)
+predict(X_test)
Inheritance
Inheritance
Inheritance
Inheritance
Inheritance
LSTMModel
+build_model()
+train(X_train, y_train, epochs, batch_size, validation_split, patience)
+predict(X_test)
GRUModel
+build_model()
+train(X_train, y_train, epochs, batch_size, validation_split, patience)
+predict(X_test)
TransformerModel
+build_model()
+train(X_train, y_train, epochs, batch_size, validation_split, patience)
+predict(X_test)
ARIMAModel
+build_model()
+train(X_train, y_train, epochs, batch_size, validation_split, patience)
+predict(X_test)
ProphetModel
+build_model()
+train(X_train, y_train, epochs, batch_size, validation_split, patience)
+predict(X_test)

current class diagram


DataCollector
+get_data(tickers: List[str], start: str, end: str) : DataFrame
DataHandler
+time_series_split(data: DataFrame, n_splits: int) : TimeSeriesSplit
+create_sequences(data: DataFrame, sequence_length: int) : Tuple[np.array, np.array]
BasePredictor
+download_data() : void
+time_series_split(n_splits: int) : TimeSeriesSplit
+train_and_evaluate(n_splits: int) : void
Inheritance
Inheritance
Inheritance
LSTMPredictor
+optimize_model(train_data: Tuple[np.array, np.array], test_data: Tuple[np.array, np.array]) : OptimizeResult
GRUPredictor
+optimize_model(train_data: Tuple[np.array, np.array], test_data: Tuple[np.array, np.array]) : OptimizeResult
ARIMAPredictor
+optimize_model(train_data: Tuple[np.array, np.array]) : OptimizeResult

Syntax error in graph
mermaid version 9.4.0

Here's a high-level description of how the current code works:

1. **Data Collection**: The **'DataCollector'** class in the **'data_collector.py'** module is responsible for downloading stock data from online sources (e.g., Yahoo Finance) based on the provided tickers, start, and end dates. This data is then stored in a Pandas DataFrame.
2. **Data Processing**: The **'DataHandler'** class in the **'data_handler.py'** module processes and prepares the data for use in the models. This includes creating sequences of data for time series modeling, splitting the data into train and test sets using time series cross-validation, and scaling the data when necessary.
3. **Model Creation**: The **'BasePredictor'** class in the **'predictor.py'** module serves as a base class for the specific predictor classes for LSTM, GRU, ARIMA, Transformer, and Prophet models (i.e., LSTMPredictor, GRUPredictor, ARIMAPredictor, TransformerPredictor, and ProphetPredictor). These predictor classes use a factory design pattern to create instances of the corresponding models, which are implemented in the Models directory.
4. **Hyperparameter Optimization**: Each predictor class defines a search space for hyperparameter optimization using Scikit-optimize's **'gp_minimize'** function. This function searches for the best hyperparameters for each model by minimizing the mean squared error (MSE) on the validation set.
5. **Model Training and Evaluation**: After finding the best hyperparameters for each model, the models are trained on the entire training set, and their performance is evaluated on the test set. The **'train_and_evaluate'** method of each predictor class handles this process.
6. **Visualization**: The **'StockVisualizer'** class in the **'stock_visualizer.py'** module provides visualization tools for comparing the predictions of the models with the actual data. This helps in understanding the accuracy and effectiveness of the models.

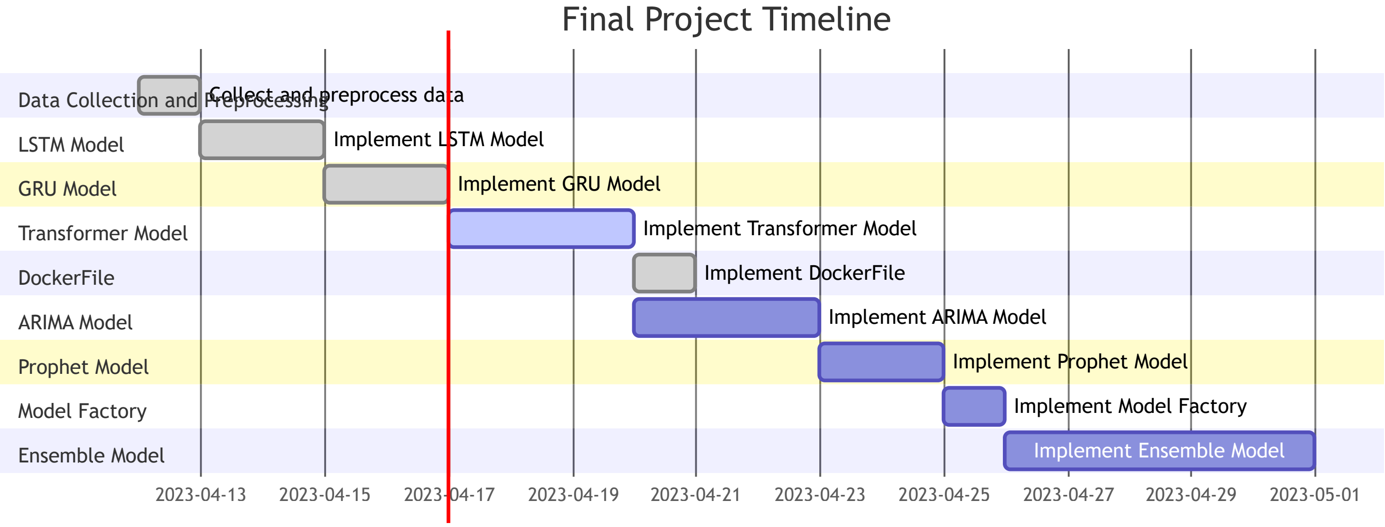
Functionality for the Transformer and Prophet models will be added in the future. The **'EnsembleModel'** class will be used to combine the predictions of the different models into a single prediction. This will be done by taking a weighted average of the predictions of the individual models, where the weights are determined by the performance of each model on the test set.

The use of a factory design pattern allows for easy addition and modification of different model types in the project. By extending the BasePredictor class and implementing the necessary methods, new models can be added to the project with minimal changes to the existing codebase. This design pattern helps maintain the modularity and flexibility of the project.

Timeline

We will be finished with the project by May 1st.

The following is a timeline of the project.
Gray boxes are completed tasks, Glowing Blue boxes are tasks in progress, and Blue boxes are tasks that are yet to be completed.


Final Project Timeline
Data Collection and Preprocessing
Collect and preprocess data
LSTM Model
Implement LSTM Model
GRU Model
Implement GRU Model
Transformer Model
Implement Transformer Model
DockerFile
Implement DockerFile
ARIMA Model
Implement ARIMA Model
Prophet Model
Implement Prophet Model
Model Factory
Implement Model Factory
Ensemble Model
Implement Ensemble Model