Execution Procedure for the Stock Price Prediction Model

This document provides the execution procedure for the stock price prediction model implemented using various machine learning models such as ARIMA, XGBoost, and a hybrid model that combines the predictions of ARIMA and XGBoost. The model is trained on Apple stock data retrieved from Yahoo Finance and utilizes Dask for parallel computation to improve performance.

# 1. Install Required Libraries

Ensure you have the following libraries installed using the following pip command:  
  
```bash  
pip install numpy pandas statsmodels joblib dask distributed xgboost dask-ml yfinance matplotlib

pip install pytorch\_lightning

pip install u8darts

pip install darts

pip install arch

pip install tensorflow

```

# 2. Setup and Initialize Dask Client

Dask is used to distribute computations across multiple processes or workers. Initialize a Dask client at the beginning of the script:  
  
```python  
client = Client()  
```

# 3. Download Data from Yahoo Finance

The model uses Apple's stock data (`AAPL`) from Yahoo Finance, retrieved from January 1, 2020, to January 1, 2023. The data is then processed to only include the 'Close' price.  
  
```python  
data = yf.download('AAPL', start='2020-01-01', end='2023-01-01')  
data = data[['Close']]  
data.index = pd.to\_datetime(data.index)  
data = data.asfreq('B').ffill()  
```

# 4. Preprocess Data

The model computes technical indicators based on the closing price, such as returns, moving averages (5-day, 20-day), and lag features. Missing values are dropped from the resulting dataset.  
  
```python  
indicators['Return'] = data['Close'].pct\_change()  
indicators['MA\_5'] = data['Close'].rolling(window=5).mean()  
indicators['MA\_20'] = data['Close'].rolling(window=20).mean()  
indicators['Lag\_1'] = data['Close'].shift(1)  
indicators = indicators.dropna()  
indicators = dd.from\_pandas(indicators, npartitions=4)  
```

# 5. Split Data into Train and Test Sets

The data is split into training and testing sets, with 80% of the data used for training and the remaining 20% for testing.  
  
```python  
train\_size = int(len(data) \* 0.8)  
train, test = data[:train\_size], data[train\_size:]  
```

# 6. Fit ARIMA Model

An ARIMA model is fit to the training data. The model is used to generate predictions for the stock prices. The predictions are added to the indicators DataFrame.  
  
```python  
model = ARIMA(train\_data, order=(1, 0, 1))  
model\_fit = model.fit()  
indicators['ARIMA\_Pred'] = model\_fit.predict(start=data.index[0], end=data.index[-1])  
```

# 7. Fit XGBoost Model

The XGBoost model is trained on the same features used for ARIMA predictions. Dask's `DaskXGBRegressor` is used for parallel training.  
  
```python  
dask\_model = xgb.dask.DaskXGBRegressor(objective='reg:squarederror', n\_estimators=100, learning\_rate=0.1)  
dask\_model.fit(X\_train\_dask, y\_train\_dask)  
```

# 8. Create Hybrid Model with Stacking

The hybrid model combines the predictions of the ARIMA and XGBoost models using stacking. A Linear Regression model is trained on the predictions of ARIMA and XGBoost.  
  
```python  
stacking\_model = LinearRegression()  
stacking\_model.fit(X\_stack, y\_stack)  
indicators\_test\_aligned['Hybrid\_Pred'] = stacking\_model.predict(X\_stack)  
```

# 9. Evaluate Performance

The models' performances are evaluated using the RMSE (Root Mean Squared Error) between the actual and predicted values for the test set.  
  
```python  
rmse\_arima = np.sqrt(mean\_squared\_error(test['Close'], indicators\_test\_aligned['ARIMA\_Pred']))  
rmse\_xgb = np.sqrt(mean\_squared\_error(test['Close'], indicators\_test\_aligned['XGBoost\_Pred']))  
rmse\_hybrid = np.sqrt(mean\_squared\_error(test['Close'], indicators\_test\_aligned['Hybrid\_Pred']))  
```

# 10. Measure Execution Time

The execution time of the models is measured for both the configurations with and without HPC (High Performance Computing) using the `measure\_execution\_time` function.  
  
```python  
time\_without\_hpc = measure\_execution\_time(without\_hpc\_execution, 'Without HPC')  
time\_with\_hpc = measure\_execution\_time(with\_hpc\_execution, 'With HPC')  
```

# 11. Display Results and Performance Summary

The performance results are summarized in a DataFrame, and visualized in bar charts comparing execution time and RMSE values.  
  
```python  
summary\_df = pd.DataFrame(results)  
summary\_df['Execution\_Time'] = [time\_without\_hpc, time\_with\_hpc]  
```