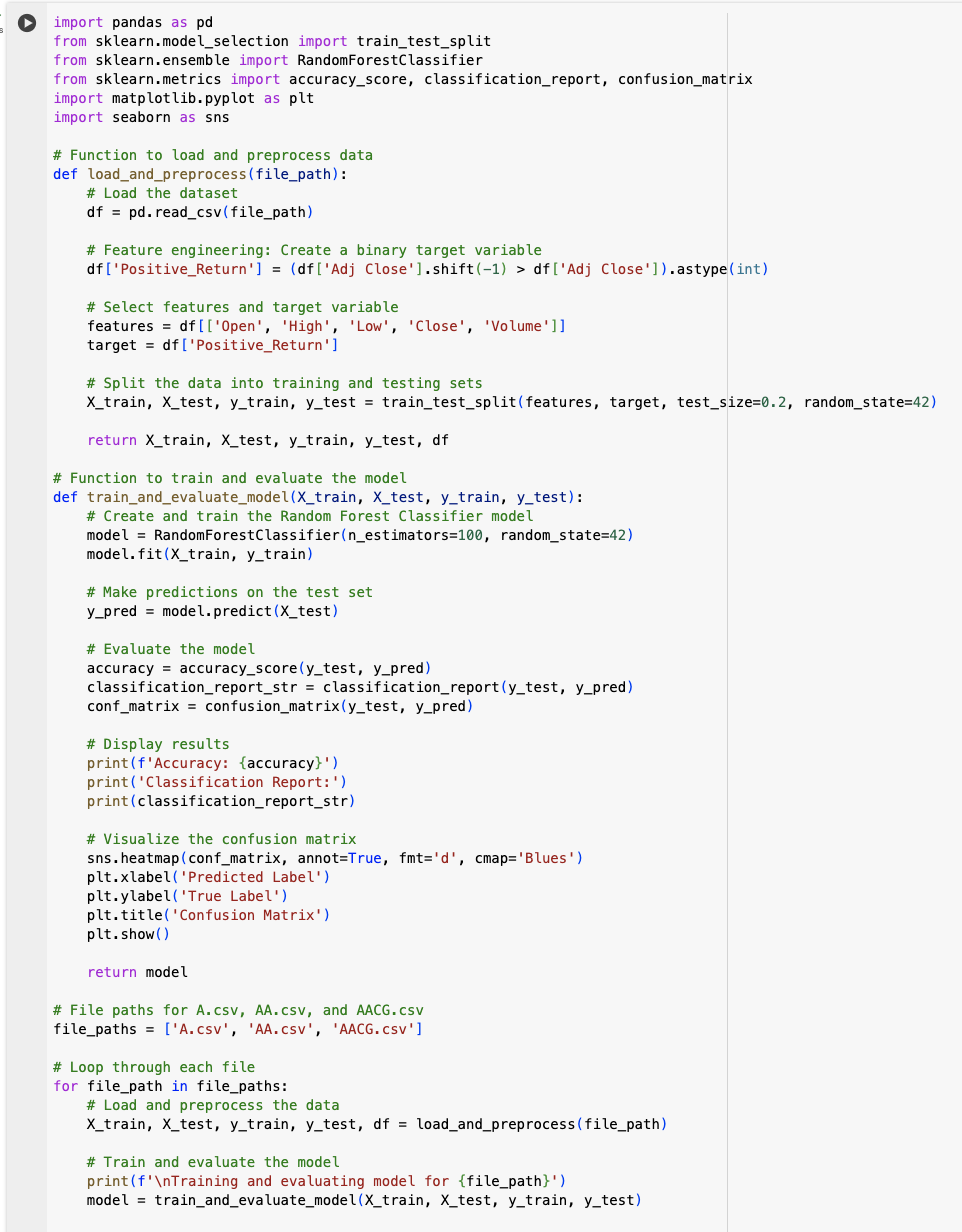
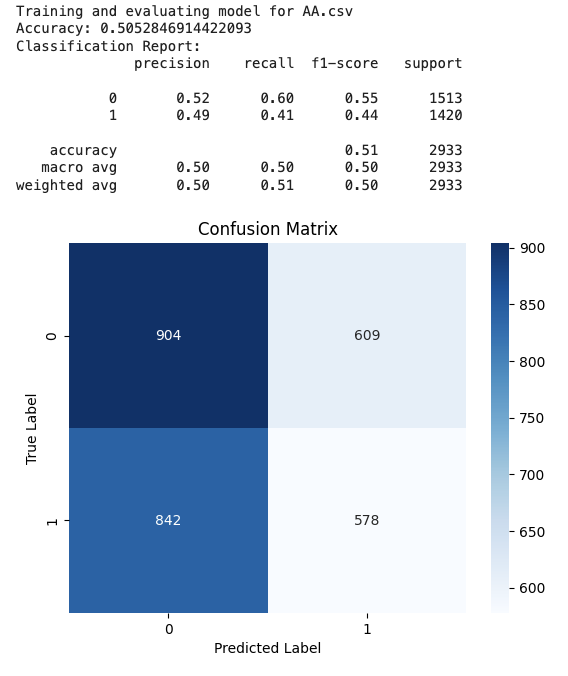
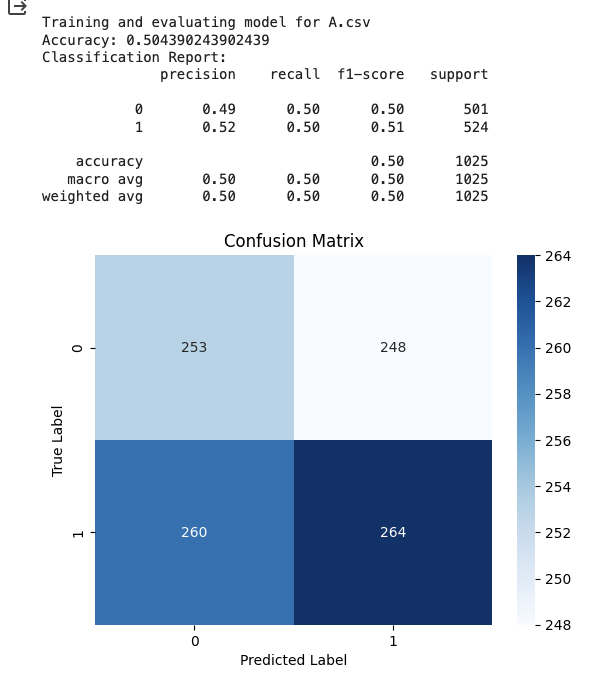
Names: Allison Escott, Nick Latsis, Ethan Reed, Joshua Thurber

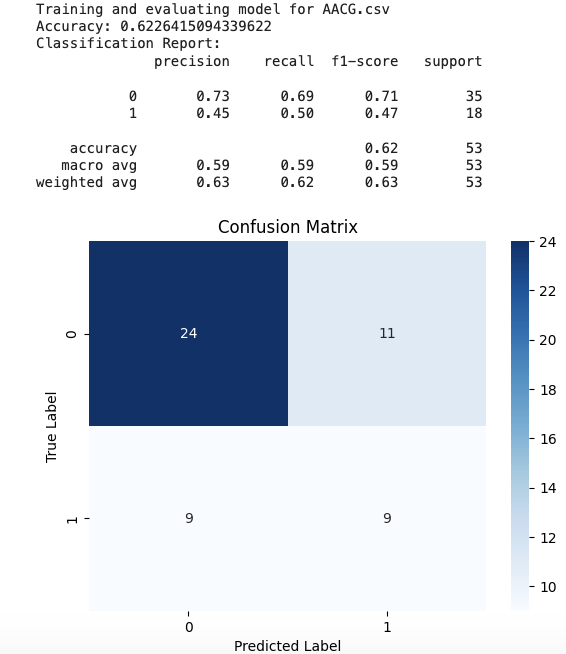
CS 4342

Report

The name of our project is Stock Flow. We want to predict stock prices using historical market data and whether or not a stock is a good investment. The dataset includes key features such as opening price, closing price, high and low prices, adjusted close prices, and trading volume. Using different ML algorithms we aim to assist investors in making informed decisions about potential future stock prices. Our data is taken from the kaggle dataset here: <https://www.kaggle.com/datasets/jacksoncrow/stock-market-dataset>. Before jumping into which ML models we wanted to use we had to do data visualization and cleaning steps. By toggling around in the kaggle dataset we were able to get a good feel for what information was given for each stock and what each file consisted of. Our dataset has more than 8,000 files so we had to decide how we wanted to work with it. We thought it would be smart to develop machine learning models that will inform brokers of decisions and have the ability to import the names of different stocks to get the output of specific stocks. This way, instead of us performing 8,000 times and having a messy dataset, we can give value to stock brokers by allowing them to input what they want. This led us to clean the data so that it was not all over the place and we could start with just a few stocks at a time.

The first ML technique that was used was **Random Forest (Allison)**. Random Forest algorithm type is ensemble Learning - Regression/Classification. Random Forest is an ensemble of decision trees. It can provide more robust predictions by combining multiple decision trees. Random Forest is less prone to overfitting and can handle a large number of input features, making it suitable for financial datasets which is why we chose it as a technique to use in order to tell brokers whether certain stocks are a good investment for them or not. We performed this on the first three stocks because as mentioned in the beginning we had to keep our data clean and not perform it over 8,000 times. We wanted to see whether these stocks would be classified as a good investment or not. The code I used for random forest is an implementation of the ML models predicting a positive or negative return of the stock based on historical data. It loads the stock files in, splits into training and testing sets and a Random Forest Classifier is trained on the training set to learn patterns in the historical data. The trained model is evaluated on the testing set using metrics such as accuracy, precision, recall, and F1-score. The confusion matrix and classification report are displayed to provide a detailed breakdown of the model's performance. Below is the code and outputs:

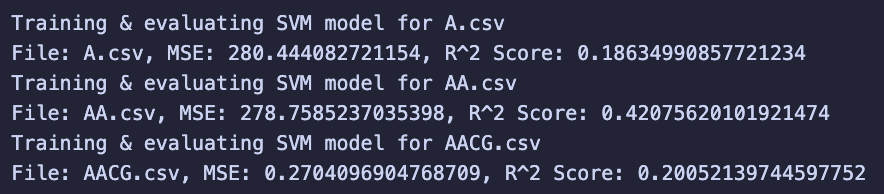




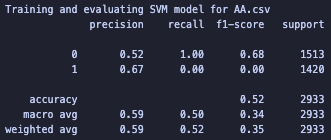
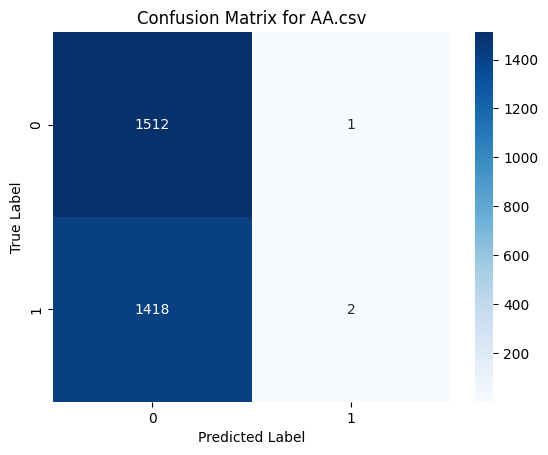
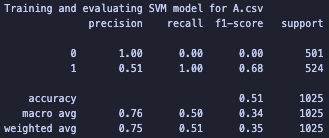
Our results show that the model's predictions are a bit better than random chance, but there is room for improvement. Brokers can use this information as an additional tool in their decision-making process just not as their sole tool. Specifically regarding the stocks results above: Stock A- Accuracy: 50.4%, Precision for predicting a positive return (class 1): 52%, Recall for predicting a positive return: 50%, F1-score: Around 0.50, indicating a balanced performance. Overall the model for stock A.csv shows a performance slightly better than random chance. It correctly identifies positive returns about half of the time. The balanced precision and recall suggest that the model is not strongly biased towards either class. Stock AA- Accuracy: 50.5% , Precision for predicting a positive return (class 1): 49%, Recall for predicting a positive return: 41%, F1-score: Around 0.44, indicating a moderate performance. The model for stock AA.csv shows an accuracy slightly better than random chance. The precision for predicting positive returns is 49%, meaning that when the model predicts a positive return, it is correct about 49% of the time. The recall for predicting positive returns is 41%, indicating that the model captures only 41% of the actual positive instances. The model for stock AA.csv is providing predictions with limited accuracy so brokers should be cautious when using this model for investment decisions. Stock AACG- Accuracy: 62.3%, Precision for predicting a positive return (class 1): 45%, Recall for predicting a positive return: 50%, F1-score: Around 0.47, indicating a moderate performance. The model for stock AACG.csv shows a higher accuracy compared to the other stocks. However, the precision and recall for predicting positive returns are relatively lower, suggesting that the model may have some difficulty identifying positive instances. Suggestions to brokers would include the facts that; A.csv and AA.csv show performance slightly better than random chance. Brokers should exercise caution and consider these models as supplementary tools rather than sole decision-makers. The model for AACG.csv performs relatively better with a higher accuracy, but caution is still advised. The lower precision for positive returns suggests that false positives may be an issue.

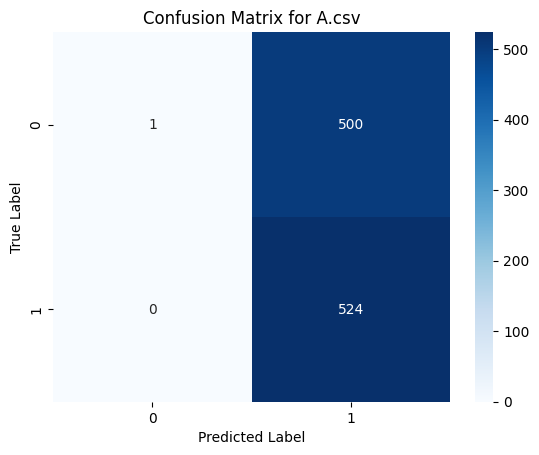
Our next technique was the **Support Vector Machines** (Ethan) approach, which unfortunately did not perform as well as the previous one. These tests on the same selected files output the following results, which show very little correlation. The best of the options is AA in terms of accuracy with a moderate R^2 of .42, whereas the rest of the alternatives in the SVM set A and AACG are very low down at R^2 values of around 0.2.

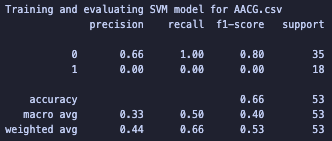
This analysis is slightly helpful but not very, as it’s still not ideal being a regression instead of the previous classification model which would be better for the stock prediction problems we’re approaching.

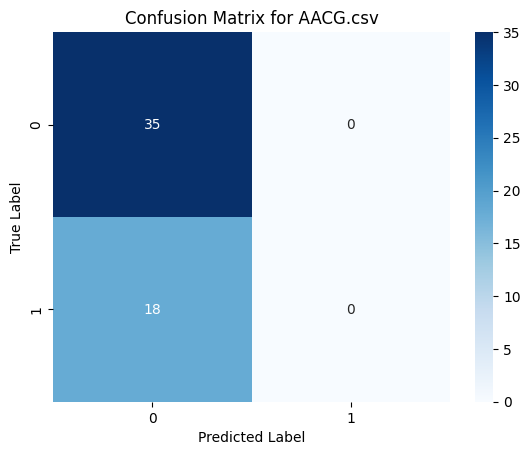


When converted to binary however, the results increased slightly, and the accuracies became a little more usable with the prediction of simply a binary variable. These new accuracies moved up as much as almost 50%, and now can predict the majority of the value of the binary variables when using the SVM approach.







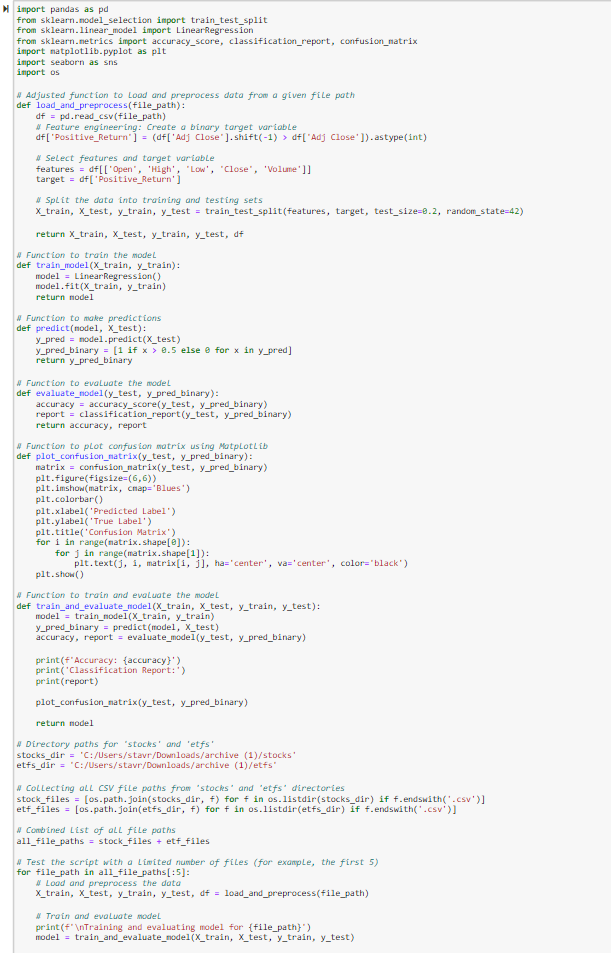
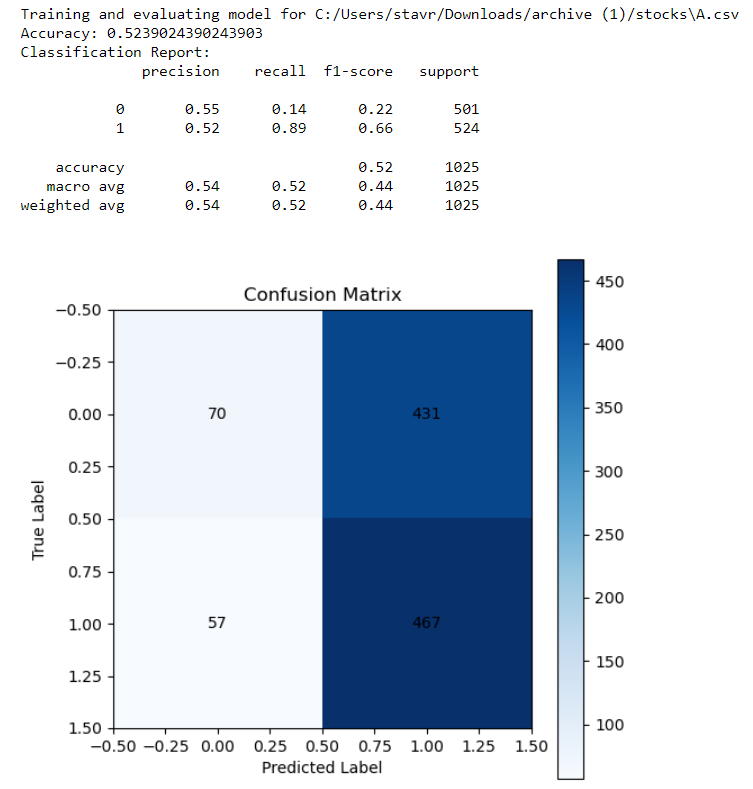


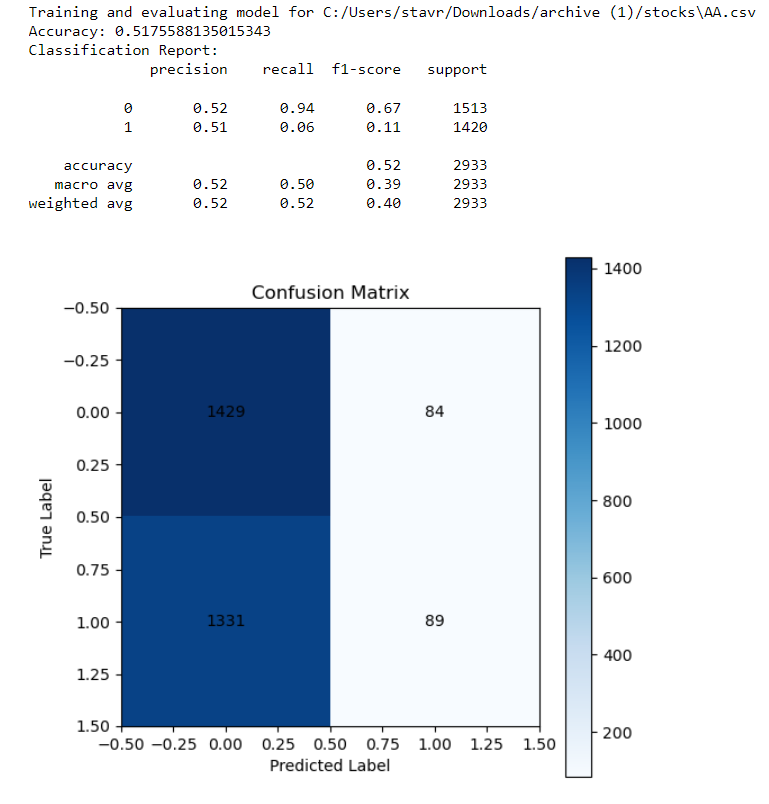
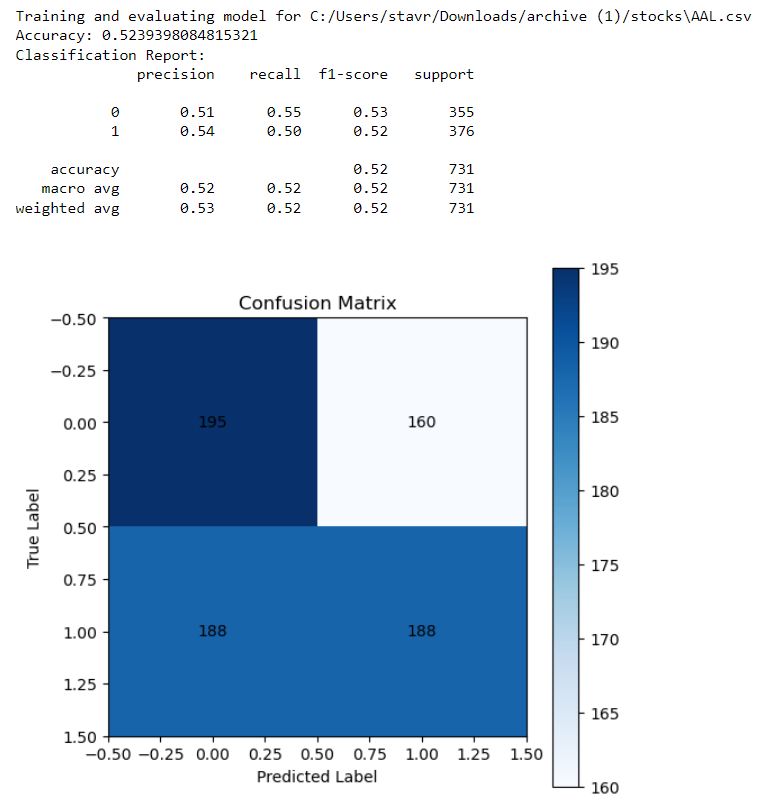
Compared to the alternatives the binary results of the SVM are moderate, and average at over 50% which means they are a little better than random guessing. The stock A model got an accuracy of 51%, with a precision of 51% for predicting a positive return, meaning that it can correctly predict positive returns about half the time. The recall is at 100%, meaning it identifies all the positive returns but because of the low precision many of them are false positives. The F1-score of 0.68 reflects a moderate performance from the imbalance between precision and recall.

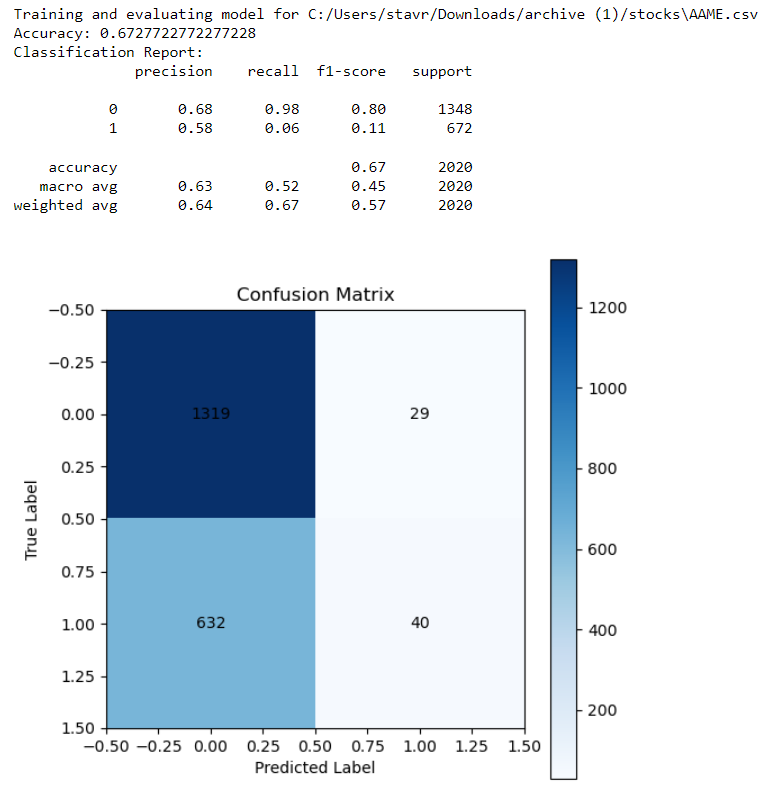
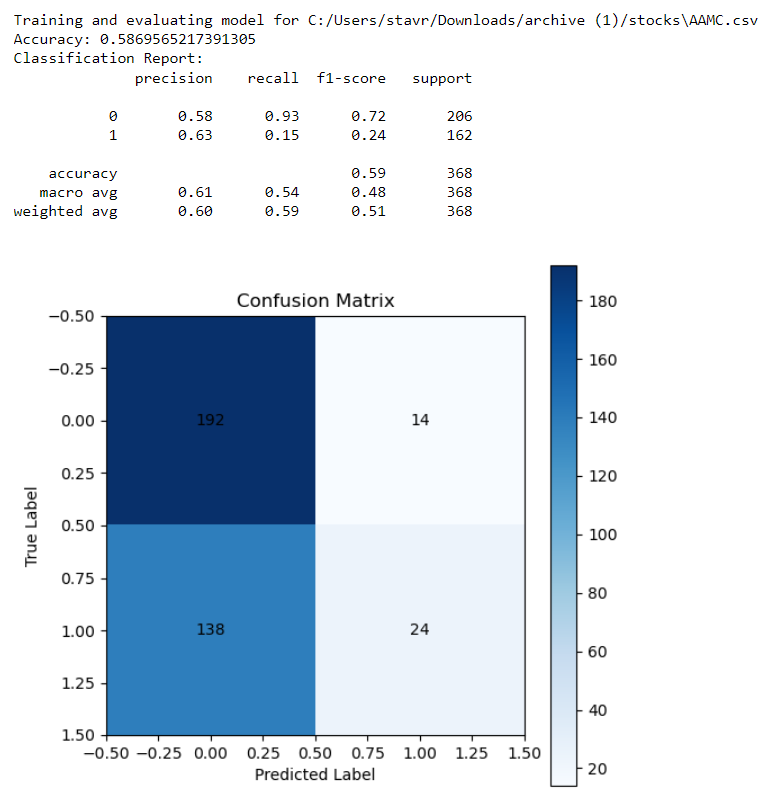
Stock AA has an accuracy of 52% which is again just slightly more accurate than a random guess. The precision for predicting positive returns is 67%, but the recall for predicting positive returns is 0% meaning that while the model is normally reliable when it predicts a positive return, it doesn’t happen often. The F1-score is very low, meaning the model can’t capturing the positive instances well either.

The Stock AACG model shows a higher accuracy of 66% compared to the other stocks, but with a precision of 0% for positive returns and a recall of 0%, it can’t identify any true positives. The F1-score reflects this bad performance and means this unfortunately can’t be a reliable predictor either.

The next ML technique we used was **Linear Regression (Nick)**. Linear Regression algorithm type is Supervised Learning - Regression/Classification. Linear Regression is a fundamental statistical approach for modeling relationships between a dependent variable and one or more independent variables. It predicts the dependent variable's value based on the independent variables. This method is highly interpretable, and can be used for both regression and binary classification problems. Due to its simplicity and efficiency in handling continuous numerical data, we chose Linear Regression as a technique to assist brokers in making investment decisions regarding stocks. Its ability to provide quick and straightforward insights makes it ideal for analyzing and predicting the market value of various rocks, thereby guiding investment strategies effectively. We performed this on the 5 of the stocks (A, AA, AAL, AAMC, and AAME) to keep the data clean and not overfit it. We aimed to determine if these stocks could be categorized as favorable investments. The implemented code for linear regression applies machine learning models to forecast whether the stock will yield a positive or negative return, utilizing past performance data. This process involves loading stock data, dividing it into training and testing datasets, and then training a Linear Regression Classifier on the training set. The purpose of this training is to enable the classifier to discern patterns in historical stock data. The trained model is evaluated on the testing set using metrics such as accuracy, precision, recall, and F1-score. The confusion matrix and classification report are displayed to provide a detailed breakdown of the model's performance. Below is the code and outputs:







Our results show that the model's predictions are a slightly better than random chance, but there is room for improvement. Brokers may find these insights useful as an additional tool in their decision-making process just not as their sole tool. Specifically regarding the stocks results above: Stock A- Accuracy: 52.4%, Precision for predicting a positive return (class 1): 52%, Recall for predicting a positive return: 89%, F1-score: Around 0.66, indicating a moderate balance between precision and recall. The model performs slightly better than random chance, and the high recall (89%) indicates it is good at identifying positive returns, but the moderate F1-score suggests there is a balance to be struck between precision and recall. Stock AA- Accuracy: 51.8%, Precision for predicting a positive return (class 1): 51%, Recall for predicting a positive return: 6%, F1-score: Around 0.11. The precision for predicting positive returns is 51%, ergo a predicted positive return is correct about 51% of the time. The recall for predicting positive returns is 6%, indicating that the model is missing 94% of all the actual positive instances. Overall, despite the accuracy being slightly better than random, the extremely low recall and F1-score indicate the model is ineffective in identifying true positives. Additionally, the low recall significantly impacts the F1-score, reflecting poor overall performance. Stock AAL- Accuracy: 52.4%, Precision for predicting a positive return (class 1): 54%, Recall for predicting a positive return: 50%, F1-score: Around 0.52, indicating a moderate performance. The model for stock AAL.csv shows a similar accuracy to the A.csv stock. Furthermore, the more balanced precision and recall, reflected in the moderate F1-score, indicate a reasonable model performance. Stock AAMC- Accuracy: 58.7%, Precision for predicting a positive return (class 1): 63%, Recall for predicting a positive return: 15%, F1-score: Around 0.24, indicating a low balance between precision and recall. The precision for predicting positive returns is 58.7%. The higher accuracy and precision, but low recall and F1-score suggest the model is not effective in catching most of the positive returns. Stock AAME- Accuracy: 67.3%, Precision for predicting a positive return (class 1): 58%, Recall for predicting a positive return: 6%, F1-score: Around 0.11. The precision for predicting positive returns is 67.3%. The high accuracy but very low recall and F1-score, similar to Stock AA, indicate poor performance in identifying true positives. Suggestions to brokers would include the facts that; A.csv and AAL.csv show performance slightly better than random chance. Although AA.csv, AAMC.csv, and AAME.csv show accuracy that is higher than random chance, their low recall and F1-scores indicate that their performance is slightly worse than random choice because they are not effective in identifying true positive cases, ergo there may be an issue with false positives. Brokers should exercise caution and consider these models as supplementary tools rather than sole decision-makers.  
  
The concluding machine learning technique used in our study was the **Long Short-Term Memory (Joshua)** network. LSTM, a form of recurrent neural network suited for sequential data, was chosen for its proficiency in capturing time-series information, which is useful in stock price prediction. This algorithm's design helps it to remember long-term patterns, a feature particularly beneficial when dealing with the complexities of financial markets. Our LSTM model was tasked to forecast stock prices and determine investment viability, leveraging data that included key indicators such as opening, high, low, and closing prices, alongside trading volume. We trained the model on a subset of stocks, intentionally limiting the scope to ensure data quality and manage computational resources effectively. The LSTM's performance was gauged by observing the reduction in loss during its training, and later, by evaluating its predictions against the actual prices. The metrics chosen for evaluation were accuracy, precision, recall, and F1-score, complemented by confusion matrices to provide a comprehensive overview of the model's predictive behavior. For the stocks assessed, the LSTM model's accuracy ranged from 35% to 44%. While these figures only modestly surpass what might be expected by random chance, they indicate that the model has begun to identify underlying trends within the data. The actual versus predicted price plots underscore the model's capacity to follow stock price trends, though not without discrepancies. The confusion matrices for each stock reveal that the model maintains a balance in classifying price movements, without displaying a significant bias toward predicting rises or falls. In summary, the LSTM model shows potential but also highlights the necessity for cautious application. Brokers may find the model's output to be a useful supplementary analysis tool, which, when combined with other financial assessments, could enhance investment strategies. The need for further model refinement is clear, and future work will seek to improve the accuracy of these predictions.

Here are the code and outputs:

| import pandas as pd import numpy as np import torchimport torch.nn as nn from torch.utils.data import DataLoader, TensorDataset from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import MinMaxScaler from sklearn.metrics import confusion\_matrix, classification\_report import seaborn as sns import matplotlib.pyplot as plt import os  # Function to load and preprocess data for LSTM def load\_and\_preprocess\_for\_lstm(file\_path, sequence\_length):  df = pd.read\_csv(file\_path)   # Check if 'Adj Close' column exists, otherwise use 'Close'  if 'Adj Close' in df.columns:  price\_column = 'Adj Close'  elif 'Close' in df.columns:  price\_column = 'Close'  else:  raise KeyError(f"No suitable price column found in the file: {file\_path}")   data = df[price\_column].values.reshape(-1, 1)  scaler = MinMaxScaler(feature\_range=(0, 1))  data = scaler.fit\_transform(data)   # Creating sequences for LSTM  X, y = [], []  for i in range(sequence\_length, len(data)):  X.append(data[i-sequence\_length:i, 0])  y.append(data[i, 0])  X, y = np.array(X), np.array(y)  X = np.reshape(X, (X.shape[0], X.shape[1], 1))   # Split the data into training and testing sets  return train\_test\_split(X, y, test\_size=0.2, random\_state=42), scaler   # Define the LSTM model using PyTorch class LSTMModel(nn.Module):  def \_\_init\_\_(self, input\_size, hidden\_layer\_size, output\_size, sequence\_length):  super(LSTMModel, self).\_\_init\_\_()  self.hidden\_layer\_size = hidden\_layer\_size  self.lstm = nn.LSTM(input\_size, hidden\_layer\_size, batch\_first=True)  self.linear = nn.Linear(hidden\_layer\_size \* sequence\_length, output\_size)   def forward(self, input\_seq):  lstm\_out, \_ = self.lstm(input\_seq)  lstm\_out = lstm\_out.contiguous().view(-1, self.hidden\_layer\_size \* sequence\_length)  predictions = self.linear(lstm\_out)  return predictions  # Train and evaluate the LSTM model def train\_and\_evaluate\_lstm(X\_train, X\_test, y\_train, y\_test, model, scaler, epochs=25, learning\_rate=0.001):  loss\_function = nn.MSELoss()  optimizer = torch.optim.Adam(model.parameters(), lr=learning\_rate)   train\_data = DataLoader(TensorDataset(torch.from\_numpy(X\_train).float(), torch.from\_numpy(y\_train).float()), batch\_size=32, shuffle=True)    # Training the model  for epoch in range(epochs):  model.train()  total\_loss = 0  for seq, labels in train\_data:  optimizer.zero\_grad()  y\_pred = model(seq)  y\_pred = y\_pred.squeeze()  loss = loss\_function(y\_pred, labels)  loss.backward()  optimizer.step()  total\_loss += loss.item()   avg\_train\_loss = total\_loss / len(train\_data)  print(f'Epoch {epoch}: Train Loss: {avg\_train\_loss}')   # Evaluating the model  def evaluate\_model(data\_loader):  model.eval()  total\_loss = 0  with torch.no\_grad():  for seq, labels in data\_loader:  y\_pred = model(seq).squeeze()  loss = loss\_function(y\_pred, labels)  total\_loss += loss.item()  return total\_loss / len(data\_loader)   train\_loader = DataLoader(TensorDataset(torch.from\_numpy(X\_train).float(), torch.from\_numpy(y\_train).float()), batch\_size=32, shuffle=False)  test\_loader = DataLoader(TensorDataset(torch.from\_numpy(X\_test).float(), torch.from\_numpy(y\_test).float()), batch\_size=32, shuffle=False)   train\_loss = evaluate\_model(train\_loader)  test\_loss = evaluate\_model(test\_loader)  print(f'Final Train Loss: {train\_loss}, Test Loss: {test\_loss}')  def create\_binary\_outcomes(prices):  return np.array([1 if prices[i] > prices[i-1] else 0 for i in range(1, len(prices))])  # Directory paths for 'stocks' and 'etfs' stocks\_dir = 'C:/Stocks/stocks' etfs\_dir = 'C:/Stocks/etfs'  # Collecting all CSV file paths from 'stocks' and 'etfs' directories stock\_files = [os.path.join(stocks\_dir, f) for f in os.listdir(stocks\_dir) if f.endswith('.csv')] etf\_files = [os.path.join(etfs\_dir, f) for f in os.listdir(etfs\_dir) if f.endswith('.csv')]  # Combined list of all file paths all\_file\_paths = stock\_files + etf\_files  # Set sequence length sequence\_length = 60  # Define the LSTM model input\_size = 1 # Number of features hidden\_layer\_size = 50 output\_size = 1 model = LSTMModel(input\_size, hidden\_layer\_size, output\_size, sequence\_length)  for file\_path in all\_file\_paths[:5]:  # Load and preprocess the data for LSTM  (X\_train, X\_test, y\_train, y\_test), scaler = load\_and\_preprocess\_for\_lstm(file\_path, sequence\_length)  print(f'\nTraining and evaluating LSTM model for {file\_path}')  train\_and\_evaluate\_lstm(X\_train, X\_test, y\_train, y\_test, model, scaler)   # Evaluate and get predictions  model.eval()  with torch.no\_grad():  X\_test\_tensor = torch.from\_numpy(X\_test).float()  test\_predictions = model(X\_test\_tensor).squeeze()  test\_predictions = scaler.inverse\_transform(test\_predictions.numpy().reshape(-1, 1)).flatten()  actual\_prices = scaler.inverse\_transform(y\_test.reshape(-1, 1)).flatten()   # Plotting Actual vs Predicted Prices  plt.figure(figsize=(12, 6))  plt.plot(actual\_prices, label='Actual Prices', color='blue')  plt.plot(test\_predictions, label='Predicted Prices', color='red')  plt.title(f'Stock Price Prediction - Actual vs Predicted for {os.path.basename(file\_path)}')  plt.xlabel('Time')  plt.ylabel('Price')  plt.legend()  plt.show()   # Convert to binary outcomes and create a confusion matrix  actual\_binary = create\_binary\_outcomes(actual\_prices)  predicted\_binary = create\_binary\_outcomes(test\_predictions)  conf\_matrix = confusion\_matrix(actual\_binary[1:], predicted\_binary[:-1])  plt.figure(figsize=(8, 6))  sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues')  plt.title(f'Confusion Matrix for {os.path.basename(file\_path)}')  plt.xlabel('Predicted Label')  plt.ylabel('True Label')  plt.show()  print(classification\_report(actual\_binary[1:], predicted\_binary[:-1])) |
| --- |

Coding Development and Explanations:

1. Data Preprocessing:

* load\_and\_preprocess\_for\_lstm: This function is responsible for preparing the data for the LSTM model. It reads the stock price data from CSV files, selects the relevant price column (using 'Adj Close' primarily or 'Close' as a fallback), scales the data between 0 and 1 for better neural network performance using MinMaxScaler, and creates sequences of a specified length that the LSTM will use to learn from.

1. LSTM Model:

* LSTMModel: This class defines the LSTM architecture. It initializes an LSTM layer that processes the input sequences followed by a linear layer that maps the LSTM outputs to the final prediction. The forward method dictates how the data flows through the model during the training and prediction phases.

1. Training and Evaluation:

* train\_and\_evaluate\_lstm: This function manages the training process of the LSTM model over a number of epochs. It uses Mean Squared Error (MSE) as the loss function suitable for regression tasks like price prediction. The optimizer used is Adam, a popular choice for deep learning tasks. The function also evaluates the model on both training and testing sets to compute the loss, providing an indication of how well the model is fitting the data without overfitting.

1. Binary Outcome Conversion:

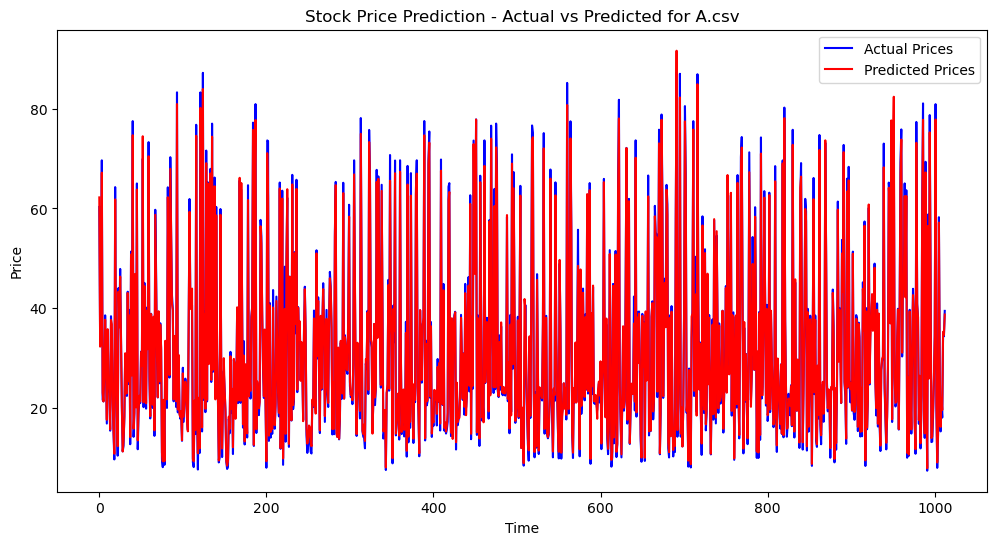
* create\_binary\_outcomes: This function transforms the continuous price data into binary outcomes (1 for price increase, 0 for decrease), which is used to assess the model's ability to predict the direction of price movement correctly.

1. Visualization:

* The code uses matplotlib to plot the actual vs. predicted prices, providing a visual comparison of the model's predictions against the true stock prices. It also uses seaborn to generate a heatmap of the confusion matrix, which offers a visual representation of the model's classification performance.

1. Loop Over Files:

* The for-loop at the end of the script iterates over the first five files in the dataset, applying the entire process—from data loading to visualization—for each stock. This loop structure allows for modular analysis of each stock file.



Training and evaluating LSTM model for C:/Stocks/stocks\A.csv

Epoch 0: Train Loss: 0.007113873035056648

Epoch 1: Train Loss: 0.0013113010340823066

Epoch 2: Train Loss: 0.0009974102305402377

Epoch 3: Train Loss: 0.0007051242425841624

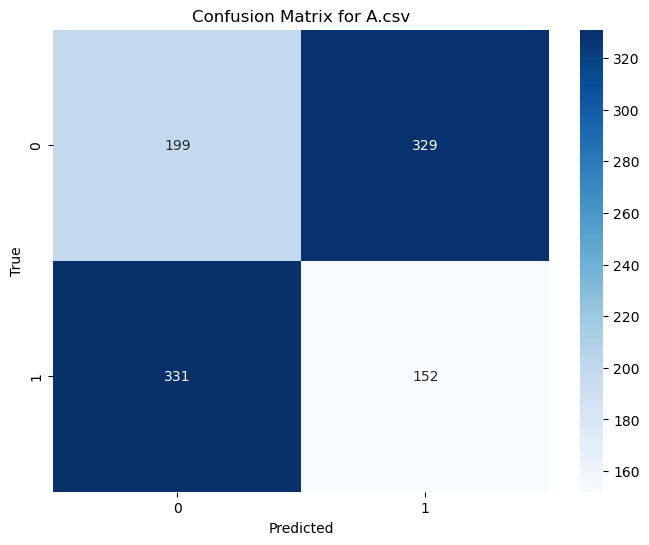
Epoch 4: Train Loss: 0.0006064045431796373

Epoch 5: Train Loss: 0.000587735387086905

Epoch 6: Train Loss: 0.0005249641130437275

Epoch 7: Train Loss: 0.0004910540247499194

Epoch 8: Train Loss: 0.00043855775117312236

Epoch 9: Train Loss: 0.00043604751425753094

Epoch 10: Train Loss: 0.0004006043814079594

Epoch 11: Train Loss: 0.0003594631696070405

Epoch 12: Train Loss: 0.0003510006533224799

Epoch 13: Train Loss: 0.000356423124184238

Epoch 14: Train Loss: 0.00032445388082639445

Epoch 15: Train Loss: 0.0003255731580005757

Epoch 16: Train Loss: 0.00030695588656107155

Epoch 17: Train Loss: 0.0003080127147639603

Epoch 18: Train Loss: 0.00027829425520695355

Epoch 19: Train Loss: 0.0002672855712656265

Epoch 20: Train Loss: 0.0002501473957342719

Epoch 21: Train Loss: 0.00027185286276430206

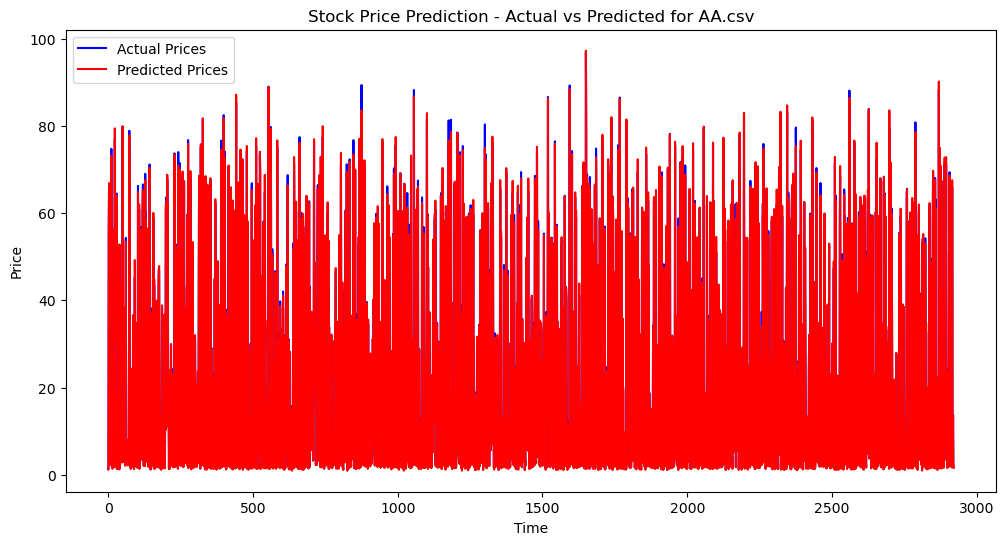
Epoch 22: Train Loss: 0.00024310558502457845

Epoch 23: Train Loss: 0.000253408238312136

Epoch 24: Train Loss: 0.00020826203814697613

Final Train Loss: 0.0002676129351323287, Test Loss: 0.00027496788629832736

|  | Precision | Recall | F1-score | Support |
| --- | --- | --- | --- | --- |
| 0 | 0.38 | 0.38 | 0.38 | 528 |
| 1 | 0.32 | 0.31 | 0.32 | 483 |
| Accuracy |  |  | 0.35 | 1011 |
| Macro Avg | 0.35 | 0.35 | 0.35 | 1011 |
| Weighted Avg | 0.35 | 0.35 | 0.35 | 1011 |



Training and evaluating LSTM model for C:/Stocks/stocks\AA.csv

Epoch 0: Train Loss: 0.00015736359749799243

Epoch 1: Train Loss: 0.00012014044637122981

Epoch 2: Train Loss: 0.00012582942032376656

Epoch 3: Train Loss: 0.00012326070179327463

Epoch 4: Train Loss: 0.00010626846469279227

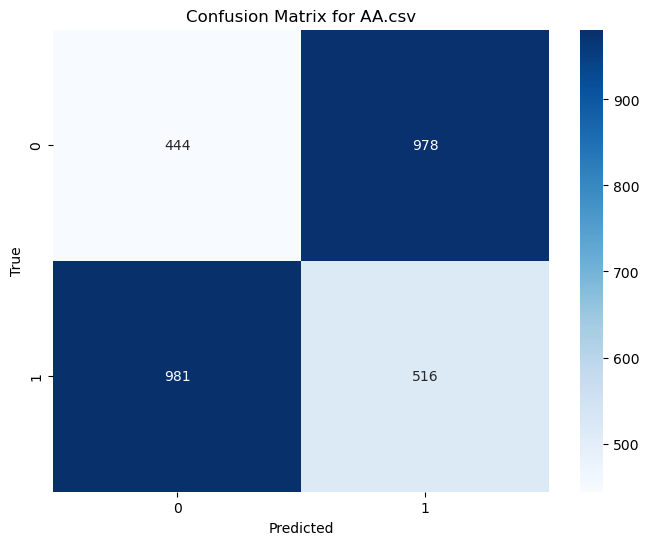
Epoch 5: Train Loss: 8.45995321201297e-05

Epoch 6: Train Loss: 8.316450208201518e-05

Epoch 7: Train Loss: 7.417111742192578e-05

Epoch 8: Train Loss: 7.319366198843669e-05

Epoch 9: Train Loss: 8.023093798696547e-05

Epoch 10: Train Loss: 8.020511180892172e-05

Epoch 11: Train Loss: 7.530491540963767e-05

Epoch 12: Train Loss: 6.905532795721119e-05

Epoch 13: Train Loss: 7.027661252924375e-05

Epoch 14: Train Loss: 7.306330461665684e-05

Epoch 15: Train Loss: 7.623347864460141e-05

Epoch 16: Train Loss: 7.213234703456572e-05

Epoch 17: Train Loss: 6.764266899168791e-05

Epoch 18: Train Loss: 7.237584882925466e-05

Epoch 19: Train Loss: 6.923618005207995e-05

Epoch 20: Train Loss: 7.779292978798889e-05

Epoch 21: Train Loss: 6.560423323873912e-05

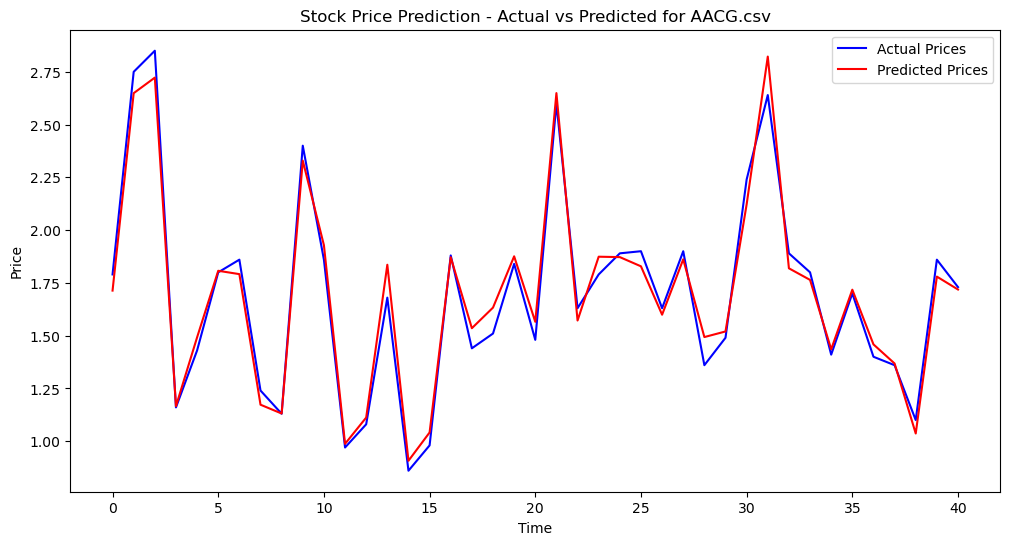
Epoch 22: Train Loss: 6.468435181611059e-05

Epoch 23: Train Loss: 6.733766064100734e-05

Epoch 24: Train Loss: 6.758927542290363e-05

Final Train Loss: 6.0964005328323416e-05, Test Loss: 6.509073109159742e-05

|  | Precision | Recall | F1-score | Support |
| --- | --- | --- | --- | --- |
| 0 | 0.31 | 0.31 | 0.31 | 1422 |
| 1 | 0.35 | 0.34 | 0.35 | 1497 |
| Accuracy |  |  | 0.33 | 2919 |
| Macro Avg | 0.33 | 0.33 | 0.33 | 2919 |
| Weighted Avg | 0.33 | 0.33 | 0.33 | 2919 |



Training and evaluating LSTM model for C:/Stocks/stocks\AACG.csv

Epoch 0: Train Loss: 0.0009457371721509844

Epoch 1: Train Loss: 0.0007268976420164108

Epoch 2: Train Loss: 0.0007646058802492917

Epoch 3: Train Loss: 0.0006562445138115436

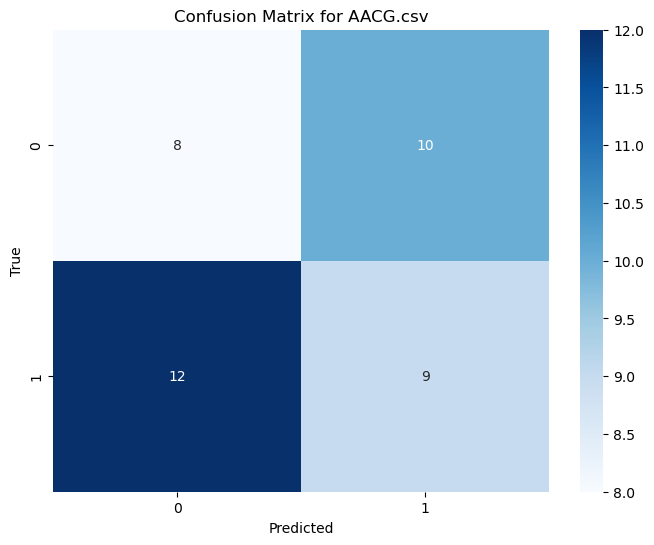
Epoch 4: Train Loss: 0.0006242829607799649

Epoch 5: Train Loss: 0.000608597561949864

Epoch 6: Train Loss: 0.0005822548235300928

Epoch 7: Train Loss: 0.0005062873533461243

Epoch 8: Train Loss: 0.0005396304943133146

Epoch 9: Train Loss: 0.0005119444453157484

Epoch 10: Train Loss: 0.0005007977713830769

Epoch 11: Train Loss: 0.00048587178462184967

Epoch 12: Train Loss: 0.0004751693399157375

Epoch 13: Train Loss: 0.0004775952897034585

Epoch 14: Train Loss: 0.0004907677532173694

Epoch 15: Train Loss: 0.0005083907104562968

Epoch 16: Train Loss: 0.0005034811038058251

Epoch 17: Train Loss: 0.0004703046113718301

Epoch 18: Train Loss: 0.00044794133282266557

Epoch 19: Train Loss: 0.0004429195047123358

Epoch 20: Train Loss: 0.0004564206232316792

Epoch 21: Train Loss: 0.0004387549532111734

Epoch 22: Train Loss: 0.00043661821982823314

Epoch 23: Train Loss: 0.0004411290632560849

Epoch 24: Train Loss: 0.0004601155582349747

Final Train Loss: 0.00042381018865853546, Test Loss: 0.0004531357262749225

|  | Precision | Recall | F1-score | Support |
| --- | --- | --- | --- | --- |
| 0 | 0.40 | 0.44 | 0.42 | 18 |
| 1 | 0.47 | 0.43 | 0.45 | 21 |
| Accuracy |  |  | 0.44 | 39 |
| Macro Avg | 0.44 | 0.44 | 0.44 | 39 |
| Weighted Avg | 0.44 | 0.44 | 0.44 | 39 |

The LSTM model was trained and evaluated on 5 distinct stocks identified as A, AA, AACG, AAL, and AAMC(AAL and AAMC were left out of the report to keep it concise as findings were similar with other inputs). The training process across stocks indicated a decrease in loss over time, suggesting that the LSTM model was learning and adapting to the patterns within the historical stock price data.

For Stock A, the final training and testing losses converged at approximately 0.0002, demonstrating a good fit without significant overfitting. However, the confusion matrix and classification report indicated a model performance that barely exceeds random chance, with an accuracy of 35% and a balanced precision and recall of approximately 38% and 31% for the positive class, respectively.

Stock AA exhibited a similar pattern in training loss reduction, with final losses slightly higher than those of Stock A. The accuracy of the model for Stock AA was 33%, with a precision of 31% and a recall of 34% for the positive class. These metrics reflect a model that is struggling to consistently predict the correct stock price movements, necessitating further tuning and perhaps additional feature engineering to improve performance.

Stock AACG's LSTM model achieved a higher accuracy of 44% compared to the other two stocks. The precision and recall for the positive class were 47% and 43%, respectively, which is a moderate performance but still indicates that the model may have difficulties in accurately classifying the stock price movement direction.

The visualizations of actual vs. predicted prices show that while the LSTM model can capture some of the movements in stock prices, there are clear discrepancies between the predicted and actual values. For Stocks A and AA, the predictions appear to be overly smooth, failing to capture the full volatility observed in the actual prices. For Stock AACG, the model seems to follow the trend with better accuracy, likely due to less volatility in the stock price movements.

In conclusion, while the LSTM model has demonstrated some capacity to learn from historical data, its predictive performance, particularly in classifying stock price movements as good investments, is limited. Brokers and investors should consider these results as a complementary tool in their decision-making process, utilizing additional analyses and models to inform their investment strategies. Further research and development, including hyperparameter optimization, advanced feature engineering, and integration with other modeling techniques, may enhance the model's predictive capabilities.

**Summary:**

In conclusion, we investigated various machine learning models, including Linear Regression, Random Forest, Support Vector Machines, and Long Short-Term Memory (LSTM) networks, to understand their efficacy in predicting stock prices and aiding in investment decisions. Each of these models was evaluated using metrics such as accuracy, precision, recall, and F1-score.The Linear Regression, Random Forest, and Support Vector Machines models displayed a capability to predict stock market trends slightly better than random chance, suggesting their potential as supplementary tools in investment strategies. However, it is important to approach these models with caution due to their inherent limitations in precision and reliability.The LSTM model, specialized for handling time-series data, showed a moderate ability to learn from historical stock prices. Despite this, its overall effectiveness in accurately predicting stock movements and identifying viable investments was constrained, reflecting the complex nature of financial markets. Across different stocks, the performance of these models varied, with some showing relatively better accuracy and others indicating a need for caution in their application. This variability underscores the necessity of using these models as part of a broader, more holistic approach to investment decision-making, rather than relying on them as sole determinants.In essence, while these machine learning models offer valuable insights, they should be employed in conjunction with other analytical methods and tools. Future enhancements, including more sophisticated feature engineering and model optimization, may further refine their predictive capabilities, making them more robust and reliable tools in the dynamic landscape of stock market analysis.

**References**

<https://www.kaggle.com/datasets/jacksoncrow/stock-market-dataset/data>

Source code within ML descriptions