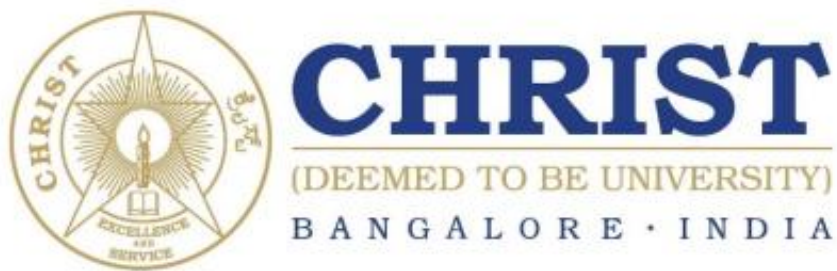


MDS581 - PROJECT II (CAPSTONE PROJECT)

SYSTEM DESIGN DOCUMENT

InvestIQ

AI-DRIVEN STOCK FORECASTING USING VISUAL PATTERNS



HIMANSHU SALVEKAR 2348052

SANJAY R 2348055

Department of Statistics and Data Science

SYSTEM DESIGN DOCUMENT

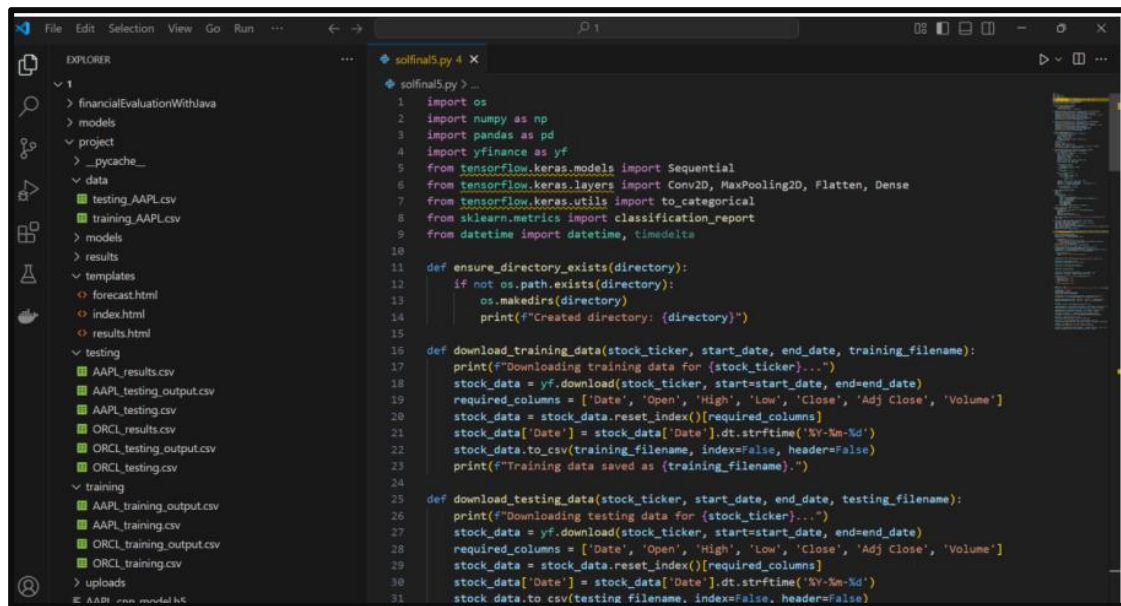
❖ INTRODUCTION:

In today's fast-paced world, financial markets, especially stock markets, play a critical role in the global economy. Stock market predictions have long been a topic of interest for both individual investors and financial institutions. Accurately predicting stock prices can significantly improve investment decisions, leading to increased profits and minimized risks. However, stock market prediction remains a complex challenge due to the volatility and unpredictability of stock prices.

The InvestIQ project is an innovative approach aimed at leveraging Convolutional Neural Networks (CNNs) to predict stock price movements based on historical data. The primary objective is to transform complex financial data into meaningful image features that capture price patterns, which are then analyzed by the CNN model to classify stock price trends. This project bridges the gap between traditional financial analysis and advanced machine learning techniques, offering investors a data-driven tool to make more informed decisions. By using historical price trends from 2008 to 2023, the model identifies patterns and predicts future stock movements with a focus on three categories: Buy, Hold, and Sell. The process involves collecting stock data, preprocessing it into histograms, and training a CNN model that learns to detect patterns indicative of future price changes. The outcomes are presented in a results file, providing users with predicted stock prices for the upcoming 15 days, helping them make strategic investment decisions.

❖ SYSTEM ARCHITECTURE:

The system architecture of InvestIQ is designed to ensure a seamless workflow from data collection, feature extraction, model training, to prediction. It is structured into three primary layers: Data Layer, Processing Layer, and Presentation Layer, each serving a distinct purpose in delivering the required functionalities efficiently. Below, each layer is elaborated further to provide a comprehensive view of the system's structure.



➤ DATA LAYER:

The Data Layer forms the foundation of the InvestIQ system, responsible for sourcing, storing, and preprocessing the stock market data required for model training and predictions. It interacts with external data sources and transforms raw data into a structured format that can be used downstream in the processing layer.

➤ DATA SOURCES:

- ✓ **Yahoo Finance API:** The primary data source for historical stock data. It provides time-series stock price data from reputable financial institutions. The dataset includes information like stock ticker symbols, daily prices, volume, and other relevant financial metrics.
- ✓ **External APIs:** Other financial APIs might be integrated for supplementary stock data, such as macroeconomic indicators that can influence stock performance.
- ✓ **User Inputs:** The user interface collects input from the user, including stock tickers, investment dates, and principal amounts, which are stored and processed later.

➤ DATA STORAGE:

```
1 2008-01-02,7.116786003112793,7.1521430015563965,6.876786231994629,6.958570957183838,5.869883060455322,1079178800
2 2008-01-03,6.978929042816162,7.049643039703369,6.881785869598389,6.961785793304443,5.872595310211182,842066400
3 2008-01-04,6.837500095367432,6.892857074737549,6.3889288902282715,6.430356979370117,5.424308776855469,1455832000
4 2008-01-07,6.473214149475098,6.557143211364746,6.0796427726745605,6.34428596496582,5.351705074310303,2072193200
5 2008-01-08,6.433570861816406,6.5164289474487305,6.099999904632568,6.1160712242126465,5.1591949462890625,1523816000
6 2008-01-09,6.117856979370117,6.410714149475098,6.010714054107666,6.4071431159973145,5.404727458953857,1813882000
7 2008-01-10,6.3421430587768555,6.464285850524902,6.26464319229126,6.3578572273254395,5.363153457641602,1482975200
8 2008-01-11,6.285714149475098,6.5164289474487305,6.0796427726745605,6.167500019073486,5.202576637268066,1232285600
9 2008-01-14,6.340000152587891, Col 3: 7.1521430015563965, 7.1090698242,6.385000228881836,5.386048793792725,1100450400
10 2008-01-15,6.347143173217773,6.400713920593262,5.880713939666748,6.037143230438232,5.092614650726318,2343278000
11 2008-01-16,5.901071071624756,6.036070823669434,5.596428871154785,5.7014288902282715,4.809423923492432,2213845200
12 2008-01-17,5.768214225769043,5.90571403503418,5.6578569412231445,5.746070861816406,4.847081661224365,1757859600
13 2008-01-18,5.775356769561768,5.919642925262451,5.700356960296631,5.762856960296631,4.861241340637207,1724343600
14 2008-01-22,5.2878570556640625,5.713571071624756,5.214285850524902,5.558570861816406,4.68891716003418,2434754000
15 2008-01-23,4.86392879486084,5.0,4.505000114440918,4.966785907745361,4.189718246459961,3372969600
16 2008-01-24,4.999642848968506,5.025000095367432,4.7146430015563965,4.8428568840026855,4.085176944732666,2005866800
17 2008-01-25,4.963929176330566,4.96750020980835,4.628929138183594,4.643214225769043,3.916771173477173,1554739200
18 2008-01-28,4.57714319229126,4.757143020629883,4.51607084274292,4.643214225769043,3.916771173477173,1474844000
19 2008-01-29,4.683928966522217,4.742499828338623,4.60892915725708,4.697856903076172,3.9628632068634033,1099982800
20 2008-01-30,4.69178581237793,4.837500095367432,4.642857074737549,4.720714092254639,3.982145071029663,1243051600
21 2008-01-31,4.623213768005371,4.880356788635254,4.621428966522217,4.834286212921143,4.07794713973999,1345674400
22 2008-02-01,4.865714073181152,4.878213882446289,4.720714092254639,4.776785850524902,4.029443740844727,1010744000
23 2008-02-04,4.793213844299316,4.853570938110352,4.693571090698242,4.701786041259766,3.9661777019500732,899234000
24 2008-02-05,4.658214092254639,4.785714149475098,4.603570938110352,4.61999885559082,3.8971872329711914,1141042000
25 2008-02-06,4.672500133514404,4.711429119110107,4.348928928375244,4.357142925262451,3.6754558086395264,1573272400
26 2008-02-07,4.284643173217773,4.4564290046691895,4.18821382522583,4.329999923706055,3.652559995651245,2083331600
```

- ✓ **Flat Files:** Historical stock data is stored in CSV files for easy access and manipulation. This approach ensures minimal storage requirements and efficient retrieval.
- ✓ **Database:** For enhanced scalability, structured databases (e.g., MySQL, MongoDB) are employed to store large amounts of stock-related data. This allows for easy querying and data normalization.
- ✓ **Feature Storage:** Preprocessed features such as Rate of Change (ROC) values and histograms are stored in separate datasets for efficient retrieval during model training.

➤ DATA PREPROCESSING:

```
def download_training_data(stock_ticker, start_date, end_date, training_filename):
    """
    Download and save training data with correctly formatted dates.
    """
    print(f"Downloading training data for {stock_ticker}...")
    stock_data = yf.download(stock_ticker, start=start_date, end=end_date)
    stock_data = stock_data.reset_index()
    stock_data['Date'] = stock_data['Date'].dt.strftime('%Y-%m-%d') # Format dates
    stock_data.to_csv(training_filename, index=False, header=True)
    print(f"Training data saved to {training_filename}.")

def download_testing_data(stock_ticker, start_date, end_date, testing_filename):
    """
    Download and save testing data with correctly formatted dates.
    """
    print(f"Downloading testing data for {stock_ticker}...")
    stock_data = yf.download(stock_ticker, start=start_date, end=end_date)
    stock_data = stock_data.reset_index()
    stock_data['Date'] = stock_data['Date'].dt.strftime('%Y-%m-%d') # Format dates
    stock_data.to_csv(testing_filename, index=False, header=True)
    print(f"Testing data saved to {testing_filename}.")
```


✓ Data Cleaning:

Data cleaning is a crucial step in the stock data preprocessing pipeline. It involves identifying and handling missing or incomplete stock data, which can arise due to factors such as data collection errors, reporting discrepancies, or unexpected market conditions. The goal is to ensure the dataset is accurate, reliable, and consistent for analysis. Missing data points are often addressed through interpolation, which estimates values based on surrounding data points, or by replacing missing values with default values like the last available observation or mean values. Anomalies, such as outliers or incorrect data points, are removed to avoid their potential to skew analysis or model results. Data cleaning not only ensures the integrity of the dataset but also enhances the performance of machine learning models by reducing noise and irrelevant information. Once cleaned, the data is transformed into a structured format, ensuring it is free from gaps and anomalies that could compromise the model's predictive accuracy. This step is particularly important when dealing with large and complex datasets, as inconsistent or missing data can lead to unreliable results. By applying proper data cleaning techniques, we improve the robustness and quality of the input data, ultimately leading to more accurate predictions and insights in stock analysis.

✓ Feature Extraction:

```
def download_training_data(stock_ticker, start_date, end_date, training_filename):  
    """  
    Download and save training data with correctly formatted dates.  
    """  
    print(f"Downloading training data for {stock_ticker}...")  
    stock_data = yf.download(stock_ticker, start=start_date, end=end_date)  
    stock_data = stock_data.reset_index()  
    stock_data['Date'] = stock_data['Date'].dt.strftime('%Y-%m-%d') # Format dates  
    stock_data.to_csv(training_filename, index=False, header=True)  
    print(f"Training data saved to {training_filename}.")  
  
def download_testing_data(stock_ticker, start_date, end_date, testing_filename):  
    """  
    Download and save testing data with correctly formatted dates.  
    """  
    print(f"Downloading testing data for {stock_ticker}...")  
    stock_data = yf.download(stock_ticker, start=start_date, end=end_date)  
    stock_data = stock_data.reset_index()  
    stock_data['Date'] = stock_data['Date'].dt.strftime('%Y-%m-%d') # Format dates  
    stock_data.to_csv(testing_filename, index=False, header=True)  
    print(f"Testing data saved to {testing_filename}.")
```

The Rate of Change (ROC) is a critical metric used to measure the percentage change in stock prices over a sliding window of 30 days. This method captures the relative price movements over a specific period, providing insights into the stock's volatility and trends. By calculating the ROC, we identify how the stock's price has shifted over time, helping to discern patterns such as upward or downward trends. The 30-day sliding window ensures that these fluctuations are captured in a meaningful timeframe, as short-term price changes can have significant implications for stock performance.

Following the computation of ROC values, histograms are generated from these values. A histogram represents the distribution of ROC values over the 30-day windows. Each bin in the histogram corresponds to a range of ROC values, effectively summarizing the intensity and frequency of price changes within that period. For example, a high concentration of ROC values within a specific range may indicate a strong trend or volatility, while a wider spread of values could suggest erratic or uncertain behavior. These histograms provide a visual way to interpret how prices have fluctuated, enabling analysts and investors to understand the stock's behavior in different market conditions.

[illegible]

Normalization is another essential step in this process, ensuring that the data is scaled appropriately. Stock prices, especially when dealing with large datasets, can vary significantly in magnitude, which can introduce biases when used in machine learning models such as Convolutional Neural Networks (CNNs). Normalization transforms the ROC values into a common scale, often by subtracting the mean and dividing by the standard deviation, to ensure the model's performance isn't influenced by large differences in feature magnitudes. This is particularly crucial in deep learning models where the input features must have similar scales to improve model convergence and

accuracy. By normalizing the data, we enhance the model's ability to learn effectively and make reliable predictions, ensuring consistent performance across varying stocks and market conditions.

➤ **PROCESSING LAYER:**

The Processing Layer is the heart of the InvestIQ system. It consists of algorithms and processes that derive insights from the preprocessed data, train machine learning models, and produce the predictions necessary for decision-making.

✓ **Feature Engineering and Extraction:**

- ◆ **Rate of Change (ROC) & 30x30 Histograms:** These features are engineered to capture stock price behavior over time, emphasizing volatility and trends that the model can learn from.
- ◆ **Convolutional Neural Network (CNN):** A core component in predicting stock price movements (buy, hold, sell).
- ◆ **Embedding Features:** Data points such as historical stock prices, volume, and other financial metrics are transformed into meaningful representations using deep learning techniques.

✓ **Model Training & Evaluation:**


- ◆ **CNN Architecture:** The CNN model consists of multiple convolutional layers followed by max-pooling layers to extract spatial features, and fully connected layers to classify the stock price movement.
- ◆ **Training Data:** The model is trained on historical stock data from the 2008-2018 period, using the ROC and histogram features extracted during data preprocessing.
- ◆ **Evaluation:** The model's performance is evaluated using metrics such as accuracy, precision, recall, and F1-score. A validation set (2019-2023) ensures robustness in predicting stock behavior in unseen scenarios.
- ◆ **Hyperparameter Tuning:** Important hyperparameters like learning rate, number of epochs, and batch size are tuned using techniques such as grid search or random search to optimize the CNN model's performance.

➤ **PREDICTION PHASE:**

- ```
def evaluate_and_save_results(model_file, test_images, test_labels, test_prices, results_file, test_dates):
 """
 Evaluate the model and save results with predictions and dates.
 """
 print(f"Evaluating model from {model_file}...")
 model = load_model(model_file)
 predictions = model.predict(test_images)
 predicted_classes = np.argmax(predictions, axis=1)
 true_classes = np.argmax(test_labels, axis=1)

 # Generate a classification report
 report = classification_report(true_classes, predicted_classes, output_dict=True)
 accuracy = report['accuracy']
 print(f"Model Accuracy: {accuracy:.2%}")

 # Save results to a CSV file
 results = pd.DataFrame({
 'Date': test_dates, # Include the date column
 'Predicted Class': predicted_classes,
 'True Class': true_classes,
 'Test Price': test_prices
 })
 results.to_csv(results_file, index=False, sep=';')
 print(f"Results saved to {results_file}.")
```

```
Epoch 1/15
81/81 2s 14ms/step - accuracy: 0.4545 - loss: 65.8529 - val_accuracy: 0.5185 - val_loss: 2.8855
Epoch 2/15
81/81  1s 13ms/step - accuracy: 0.5782 - loss: 1.4186 - val_accuracy: 0.409
```



```
Epoch 15/15
81/81 — 1s 11ms/step - accuracy: 0.8969 - loss: 0.2791 - val_accuracy: 0.7577 - val_loss: 0.5341
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Model saved to META_cnn_model.h5
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
```

```
Results saved to testing\META_results.csv
```

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.00      | 0.00   | 0.00     | 134     |
| 1            | 0.55      | 0.65   | 0.60     | 1350    |
| 2            | 0.39      | 0.31   | 0.34     | 942     |
| accuracy     |           |        | 0.48     | 2426    |
| macro avg    | 0.31      | 0.32   | 0.31     | 2426    |
| weighted avg | 0.46      | 0.48   | 0.47     | 2426    |

### ➤ PRESENTATION LAYER:

The Presentation Layer is the user-facing interface where results are displayed, enabling users to interact with the system and make informed decisions based on predictions.

```
Enter the principal amount invested: 10000
Enter the investment date (YYYY-MM-DD): 2023-11-07
Initial Investment: 10000.0
Price on 2023-11-07: 248.37217712402344
Price after 15 days: 302.9977111816406
Forecasted Principal: 12199.34
Recommendation: Buy
```

### ✓ User Interface:

- ◆ **Login and Dashboard:** Upon logging into the InvestIQ platform, users are presented with a dashboard displaying stock analysis tools.
- ◆ **Input Form:** Users input the stock ticker, investment date, and principal amount through a web-based interactive form.

- ◆ **Stock Data Visualization:** Results are displayed as a line chart showing the stock price prediction over the next 15 days.
- ◆ **Interactive Charts:** The line chart provides insights into the predicted stock behavior and helps users make buy/hold/sell decisions based on confidence levels.

✓ **Visualization:**

- ◆ **Predicted Results:** A detailed line chart shows stock price predictions for the next 15 days.
- ◆ **Historical Data:** The chart includes historical stock price trends for comparison, enhancing user confidence in predictions.
- ◆ **Confidence Metrics:** Confidence intervals and predicted accuracy are displayed, allowing users to assess the reliability of predictions.

✓ **Result Analysis and Feedback:**

- ◆ **Feedback Loop:** Users receive feedback on the accuracy of predictions based on historical performance.
- ◆ **Performance Reports:** Users can access reports summarizing prediction results, accuracy rates, and key insights derived from the predictions.
- ◆ **Recommendation:** Based on predictive accuracy, users receive buy/hold/sell recommendations to guide their investment decisions.

❖ **ALGORITHMS AND RELATED DATA STRUCTURES:**

In the InvestIQ system, several algorithms and data structures are utilized to handle stock data, compute key metrics, and make accurate predictions. These components play a vital role in processing the data efficiently and ensuring that the system delivers reliable stock movement predictions.

**1. CONVOLUTIONAL NEURAL NETWORKS (CNNS):**

The Convolutional Neural Network (CNN) algorithm is the backbone for stock price prediction in the InvestIQ system. The primary objective of the CNN is to identify spatial patterns in the histogram matrices generated from the Rate of Change (ROC) values over a 30-day window. These spatial patterns help capture underlying trends in

stock prices. The CNN architecture consists of convolutional layers that apply convolution operations to detect features in the histogram data. These features are then passed through pooling layers to reduce dimensionality and preserve significant patterns. Finally, fully connected layers combine these features to make predictions, classifying stock movements into three categories: "Hold," "Buy," or "Sell." The output from the CNN includes predicted labels along with the corresponding predicted stock prices.

## 2. RATE OF CHANGE (ROC) CALCULATION:

The ROC algorithm is essential for computing percentage changes in stock prices over a sliding 30-day window. The main purpose of the ROC calculation is to identify the momentum of stock price movements. It compares the current stock price with the price from 30 days earlier, providing a time-series representation of stock price changes. This helps in detecting trends such as upward or downward momentum. The ROC values are structured as a list, where each entry corresponds to a specific stock and window. These values serve as critical inputs to the CNN model, helping to understand and predict future stock movements.

## 3. HISTOGRAMS GENERATION:

Histograms are generated from the ROC values to visually represent the intensity of price changes over 30-day windows. The algorithm discretizes the price changes into frequency counts, producing a 30x30 matrix for each window. Each pixel in the histogram matrix reflects the intensity of stock price variations within that specific

```
def get_histogram(input_name, roc):
 df = pd.read_csv(input_name, header=None, delimiter=',')
 for r in range(len(df) - 45):
 all_y = df[5].values.tolist()
 price45 = all_y[r + 33]
 price30 = all_y[r + 29]
 dif_ratio = ((price45 - price30) / price30) * 100
 roc.append(dif_ratio)
 roc.sort()
 len_roc = len(roc)
 f_sliding = roc[2 * len_roc // 5]
 s_sliding = roc[3 * len_roc // 5]
 print("len_roc:", len_roc, "f_sliding:", f_sliding, "s_sliding:", s_sliding)
 return round(f_sliding, 6), round(s_sliding, 6)

def imagesFileCreation(input_name, output_name, f_sliding, s_sliding):
 df = pd.read_csv(input_name, header=None, delimiter=',')
 for r in range(len(df) - 45):
 img = [[255 for _ in range(30)] for _ in range(30)]
 all_y = df[5].values.tolist()
 sub_y = all_y[r:r + 30]
 current_price = round(all_y[r + 29], 2)
 price45 = all_y[r + 44]
 price30 = all_y[r + 29]
 dif_ratio = ((price45 - price30) / price30) * 100
 max_y = (all_y[r + 15] * 130) / 100
 min_y = (all_y[r + 15] * 70) / 100

 if dif_ratio >= s_sliding:
 predictLabel = 1 # Buy
 elif f_sliding < dif_ratio < s_sliding:
 predictLabel = 0 # Hold
```





not normalized. Normalization techniques, such as Min-Max scaling or Z-score normalization, are applied to ensure that all features are in a consistent range. This prevents the model from being influenced by features with larger or smaller scales, enhancing the accuracy and reliability of the CNN model. The normalized ROC values and histogram features serve as standardized inputs, improving model performance.

## 5. TRAINING AND EVALUATION OF CNN:

```
def train_and_save_model(train_images, train_labels, model_file):
 print("Training CNN model...")
 model = Sequential([
 Conv2D(32, (3, 3), activation='relu', input_shape=(30, 30, 1)),
 MaxPooling2D((2, 2)),
 Flatten(),
 Dense(64, activation='relu'),
 Dense(3, activation='softmax')
])

 model.compile(optimizer='adam',
 loss='categorical_crossentropy',
 metrics=['accuracy'])

 model.fit(train_images, train_labels, epochs=10, batch_size=32, verbose=2)
 model.save(model_file)
 print(f"Model saved to {model_file}.")

def evaluate_and_save_results(model_file, test_images, test_labels, test_prices, results_
 """
 Evaluate the model and save results with predictions and dates.
 """
 print(f"Evaluating model from {model_file}...")
 model = load_model(model_file)
 predictions = model.predict(test_images)
 predicted_classes = np.argmax(predictions, axis=1)
 true_classes = np.argmax(test_labels, axis=1)

 # Generate a classification report
 report = classification_report(true_classes, predicted_classes, output_dict=True)
 accuracy = report['accuracy']
 print(f"Model Accuracy: {accuracy:.2%}")
```

Training the CNN model involves splitting the dataset into training and testing sets. The training data, which includes the preprocessed ROC values and histograms, is fed through the CNN. The model learns to extract relevant features and make predictions on stock movement labels. During the evaluation phase, metrics such as precision, recall, and F1-score are used to measure the model's performance. The accuracy of these metrics indicates how well the model is able to classify stock movements. The

evaluation results help refine the CNN model, ensuring its predictions are more precise and reliable.

| Results saved to testing\META_results.csv |           |        |          |         |  |
|-------------------------------------------|-----------|--------|----------|---------|--|
|                                           | precision | recall | f1-score | support |  |
| 0                                         | 0.00      | 0.00   | 0.00     | 134     |  |
| 1                                         | 0.55      | 0.65   | 0.60     | 1350    |  |
| 2                                         | 0.39      | 0.31   | 0.34     | 942     |  |
| accuracy                                  |           |        | 0.48     | 2426    |  |
| macro avg                                 | 0.31      | 0.32   | 0.31     | 2426    |  |
| weighted avg                              | 0.46      | 0.48   | 0.47     | 2426    |  |

## 6. PREDICTION AND VISUALIZATION:

Once the CNN model is trained and validated, it is used to make predictions for stock movement labels. The predicted stock movement labels indicate whether the stock should be held, bought, or sold. Additionally, the model generates predictions for future stock prices based on historical data. The results are visualized through line charts, which display the predicted stock price trends over time. This provides users with clear insights into potential stock performance, helping them make informed investment decisions. The Prediction Store stores these predicted labels, actual labels, and accuracy metrics, ensuring the results are easily accessible for future analysis and reference.

### ❖ ENTITY-RELATIONSHIP DIAGRAM (ERD):

The Entity-Relationship Diagram (ERD) for the InvestIQ system illustrates the key entities and their relationships, which are central to tracking stock data, user interactions, and prediction results. The ERD provides a clear framework for understanding how data flows and how different components interact within the system.

- ✓ **Users:** The Users entity captures essential attributes such as User ID, Login Credentials, and Interaction Details. The User ID acts as a unique identifier for each user, ensuring secure access and tracking of their activities. The Login Credentials store the username and password, facilitating user authentication

when accessing the InvestIQ system. Interaction Details track user actions like stock search queries, predictions made, and visualization preferences. Each user can have multiple stock predictions, with the system storing historical data that users accessed and the predictions they generated. The relationship between Users and Predictions is many-to-one, indicating that each user can generate multiple predictions for different stocks, but a single prediction is linked to one user only.

- ✓ **Stock Data:** The Stock Data entity holds attributes such as the Ticker Symbol, Historical Data, Rate of Change (ROC), and Histograms. The Ticker Symbol serves as the unique identifier for each stock (e.g., "AAPL" for Apple, "MSFT" for Microsoft). The Historical Data includes price and volume information sourced from the Yahoo Finance API, providing the basis for further analysis. The Rate of Change (ROC) represents the percentage change in stock prices over a 30-day sliding window, capturing price momentum over time. The Histograms are 30x30 matrices that visualize the intensity of price changes, aiding in understanding the patterns of stock price fluctuations. Stock data is linked to predictions through a many-to-one relationship, where multiple prediction records can be associated with a single stock.
- ✓ **Predictions:** The Predictions entity contains attributes such as Predicted Labels, Actual Labels, and Predicted Stock Prices. The Predicted Labels are integer values indicating stock movement predictions (0 for "Hold", 1 for "Buy", and 2 for "Sell"). The Actual Labels refer to the actual market condition and are used for evaluation purposes. The Predicted Stock Prices are floating-point values representing the predicted stock price after 15 days. Each prediction is tied to a specific stock, creating a one-to-many relationship, as multiple predictions can be made for the same stock.

```
def forecast_investment(principal, investment_date, test_prices, results_df):
 """
 Forecast the investment value based on the given principal and date.
 """
 # Ensure investment_date is properly formatted
 investment_date = datetime.strptime(investment_date, '%Y-%m-%d').date()

 # Filter results for dates after the investment date
 results_df['Date'] = pd.to_datetime(results_df['Date']).dt.date # Ensure dates are in the correct format
 filtered_results = results_df[results_df['Date'] >= investment_date]

 if filtered_results.empty:
 raise ValueError("No data available after the investment date.")

 # Forecast the value based on price data
 first_price = filtered_results['Test Price'].iloc[0]
 last_price = filtered_results['Test Price'].iloc[-1]

 forecast_value = principal * (last_price / first_price)

 # Generate recommendation based on forecast logic
 if last_price > first_price:
 recommendation = "Buy"
 elif last_price < first_price:
 recommendation = "Sell"
 else:
 recommendation = "Hold"

 print(f"Forecast Value: {forecast_value:.2f}, Recommendation: {recommendation}")
 return round(forecast_value, 2), recommendation
```

```
Enter the principal amount invested: 10000
Enter the investment date (YYYY-MM-DD): 2023-11-07
Initial Investment: 10000.0
Price on 2023-11-07: 248.37217712402344
Price after 15 days: 302.9977111816406
Forecasted Principal: 12199.34
Recommendation: Buy
```

## ❖ DATA FLOW DIAGRAM (DFD):

The Data Flow Diagram (DFD) provides a high-level view of how data flows within the InvestIQ system. It illustrates the interactions between external entities, processes, and data storage components, helping to understand the data flow from input to output.

### ➤ External Entities;

- ✓ **User:** The primary external entity, responsible for providing input such as stock ticker symbols, investment dates, and principal amounts. The user initiates interactions with the system by specifying stock-related details and requesting predictions.



- ✓ **Yahoo Finance API:** Another external entity, crucial for fetching historical stock data. It retrieves data such as opening, high, low, and closing prices, volumes, and timestamps. The API also provides ROC values and histograms, which are essential for further analysis.

```
import yfinance as yf
```

```
def download_training_data(stock_ticker, start_date, end_date, training_filename):
 """
 Download and save training data with correctly formatted dates.
 """
 print(f"Downloading training data for {stock_ticker}...")
 stock_data = yf.download(stock_ticker, start=start_date, end=end_date)
 stock_data = stock_data.reset_index()
 stock_data['Date'] = stock_data['Date'].dt.strftime('%Y-%m-%d') # Format dates
 stock_data.to_csv(training_filename, index=False, header=True)
 print(f"Training data saved to {training_filename}.")

def download_testing_data(stock_ticker, start_date, end_date, testing_filename):
```

➤ **Processes:**

- ✓ **Data Collection & Preprocessing:** This process extracts ROC values from historical stock data using a 30-day sliding window and generates histograms to visualize price intensity changes. Additionally, data normalization is performed to ensure consistent scaling for machine learning models like CNNs, reducing biases caused by varying scales.
- ✓ **Model Training & Prediction:** A Convolutional Neural Network (CNN) is applied to the preprocessed data to train on histograms and predict stock movement trends (Buy, Hold, Sell). The model predicts stock prices 15 days into the future and evaluates its accuracy.
- ✓ **Result Generation:** Outputs include predicted labels, actual labels, and predicted stock prices. A line chart is generated to display the predicted stock price trends, offering a visual representation of predicted stock behavior over time.

```

1 Predicted Label,Actual Label,Test Price
2 2;1;37.70860290527344
3 2;1;33.952545166015625
4 1;1;35.40195083618164
5 1;1;35.32315826416016
6 2;1;35.99653244018555
7 2;1;36.60780715942383
8 2;1;36.72480010986328
9 2;1;36.36424255371094
10 2;1;35.81743621826172

```

### ❖ DATA STORES:

- ✓ **Stock Data Store:** Contains historical stock data, ROC values, and histograms. It serves as the foundational dataset for training the CNN model and conducting predictions.
- ✓ **Prediction Store:** Stores predicted labels, actual labels, predicted stock prices, and accuracy metrics from the CNN model. It maintains all results related to stock predictions and model performance.

### ❖ DATA DICTIONARY:

The Data Dictionary ensures clarity and consistency in the definitions of key terms and data structures used within the InvestIQ system. It provides a detailed description of each data element, ensuring proper understanding and efficient use across the system.

- i. **Stock Ticker:** A unique identifier for each stock, typically consisting of symbols like "AAPL" for Apple or "MSFT" for Microsoft.

**Example:** "AAPL", "MSFT", "GOOG"

```

2024-12-10 22:02:02.023475: I tensorflow/core/util/port.cc:11
. You may see slightly different numerical results due to flo
different computation orders. To turn them off, set the envia
)PTS=0`.
Enter the stock ticker symbol (e.g., 'AAPL', 'MSFT'): msft
Downloading training data for MSFT...

```

- ii. **Rate of Change (ROC):** The Rate of Change (ROC) represents the percentage change in stock price over a 30-day sliding window. It helps capture the trend of stock price movements, showing whether the price is rising or falling over that period.

**Example Values:** 5.3%, -2.8%, 1.2%

These values indicate changes in stock prices, providing insights into the momentum and direction of price trends over the specified timeframe.

```

Results saved to testing\MSFT_results.csv
 precision recall f1-score support

 0 0.05 0.02 0.03 182
 1 0.60 0.68 0.64 1426
 2 0.34 0.30 0.32 818

 accuracy 0.50 2426
 macro avg 0.33 0.34 0.33 2426
 weighted avg 0.47 0.50 0.49 2426

Enter the principal amount invested: 1000
Enter the date of investment (YYYY-MM-DD): 2012-11-13
Forecasted value after 15 days: 1050.26
Recommendation: BUY

```

### iii. Histograms:

A visual representation of price intensity changes over a 30-day window, where each pixel reflects the frequency of stock price variations.

**Example Values:** Each pixel value represents the intensity of stock price changes in a 30-day window.

[illegible]

#### iv. Predicted Labels:

Represents stock movement predictions, with values indicating the recommended action: 0 for "Hold," 1 for "Buy," and 2 for "Sell."

**Example Values:** 0, 1, 2

```

Results saved to testing\MSFT_results.csv

```

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0            | 0.05      | 0.02   | 0.03     | 182     |
| 1            | 0.60      | 0.68   | 0.64     | 1426    |
| 2            | 0.34      | 0.30   | 0.32     | 818     |
| accuracy     |           |        | 0.50     | 2426    |
| macro avg    | 0.33      | 0.34   | 0.33     | 2426    |
| weighted avg | 0.47      | 0.50   | 0.49     | 2426    |

```

Enter the principal amount invested: 1000
Enter the date of investment (YYYY-MM-DD): 2012-11-13
Forecasted value after 15 days: 1050.26
Recommendation: BUY

```

### v. Prediction Results:

Represents the results of stock predictions, including the actual market condition, predicted recommendation, and the forecasted stock price 15 days ahead.

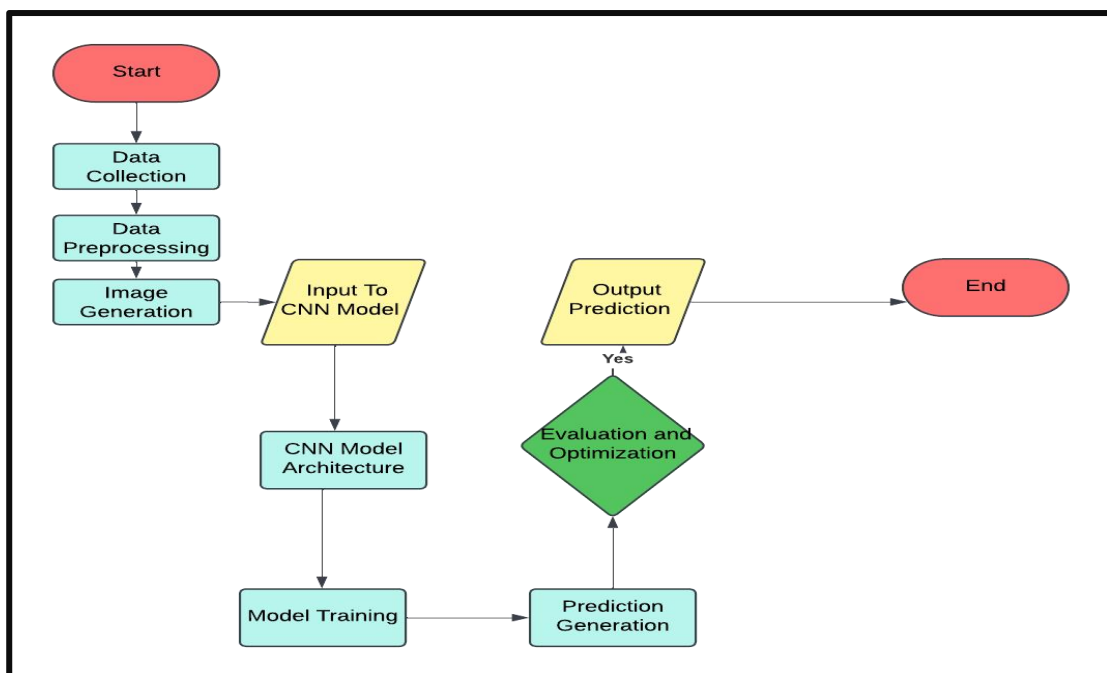
**Example Values:** Actual Label: 1   Predicted Label: 1   Predicted Stock Price: 145.75

This ERD, DFD, and Data Dictionary provide a clear structure and understanding of the InvestIQ system, ensuring smooth integration, data accuracy, and efficient functionality.



```
1 Prediction;Actual Label;Test Price
2 0;0;3.96
3 0;2;4.00
4 0;2;4.18
5 0;2;4.43
6 1;2;4.46
7 1;2;4.53
8 1;2;4.68
9 1;2;4.69
10 1;2;4.59
```

❖ BLOCK DIAGRAM:



## ❖ CONCLUSION:

The Stock Market Prediction using Time Series Data and CNN project has the potential to significantly alter the way stock market trends are predicted. In the traditional approach, stock predictions often rely on basic statistical methods or simple machine learning models that may fail to capture complex patterns in the data. However, the integration of Convolutional Neural Networks (CNNs) with time series data analysis introduces an innovative approach by leveraging deep learning techniques to recognize intricate patterns and trends within stock price data. This project aims to provide accurate and actionable insights by transforming raw stock price data into visually interpretable formats, such as graphs and charts, allowing for better decision-making for investors.

By employing CNNs, the model can take advantage of its image recognition capabilities, converting time series data into visual formats (such as time-lagged data matrices) which are easier for the neural network to process. This method allows the model to learn more complex patterns in the stock prices over time, which traditional methods might miss. As a result, the model can offer more accurate predictions of future stock prices, helping investors make informed choices.

Moreover, this project has real-world applications not just for individual investors but also for institutional investors, financial analysts, and hedge funds that rely on predictive models to drive their investment strategies. With its ability to process large amounts of data, the system can work efficiently and provide real-time predictions. This will enable stakeholders to stay ahead in the competitive stock market environment.

In essence, the project bridges the gap between traditional finance and modern machine learning techniques, presenting an innovative solution to a problem that has long been challenging for the financial industry. As the model improves over time through continued training and data processing, its relevance and accuracy will only increase, potentially revolutionizing stock market forecasting. The project also holds promise for being adapted to other markets and financial sectors, further expanding its impact and usefulness.

## ❖ REFERENCES:

- [1] H. Alom, M. Taha, and A. Asyhari, "Deep Learning: A Comprehensive Review on Techniques, Applications, and Tools," *IEEE Access*, vol. 7, pp. 143760-143786, 2019. doi: 10.1109/ACCESS.2019.2943119.
- [2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436-444, May 2015. doi: 10.1038/nature14539.
- [3] A. Karpathy, "Convolutional Neural Networks for Visual Recognition," *Stanford University Lecture Notes*, 2016. Available: <https://cs231n.stanford.edu/>
- [4] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," *Proceedings of NeurIPS*, pp. 3111-3119, 2013. Available: <https://papers.nips.cc/paper/2013>
- [5] C. Szegedy, V. Vanhoucke, and Z. Chen, "Rethinking the Inception Architecture for Computer Vision," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. doi: 10.1109/CVPR.2016.308.
- [6] F. Chollet, "Keras: The Python Deep Learning Library," *GitHub repository*, 2015. Available: <https://github.com/keras-team/keras>
- [7] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *Proceedings of NeurIPS*, pp. 448-456, 2015. Available: <https://arxiv.org/abs/1502.03167>