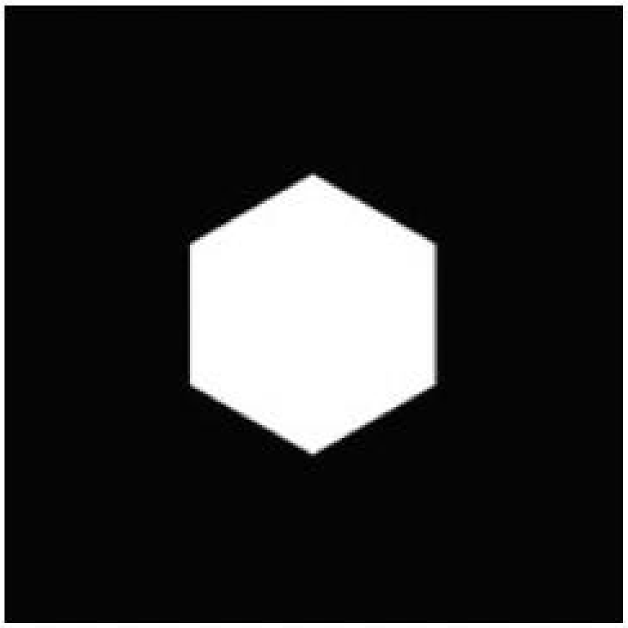
REPORT



Jahnavi Shah
Masters in Statistics (Data Science)

INTRODUCTION :

The objective of this project was to implement a transformer-based model to process trade and market data and generate trade recommendations (Buy, Sell, Hold). The model's performance was then evaluated against a simple trading blotter strategy.

Why was this model chosen :

The transformer model was chosen for its ability to capture long-range dependencies and complex patterns in sequential data, making it well-suited for financial time series analysis.

DATASET & PREPROCESSING :

Calculating Liquidity**:** Liquidity was computed as the sum of the bid and ask sizes multiplied by their respective prices. This metric helps to understand the market depth and the ease with which assets can be traded.

Calculating RSI**:** The Relative Strength Index (RSI) was calculated using the rolling averages of gains and losses over a specified window. RSI is a momentum oscillator that measures the speed and change of price movements, providing insights into potential overbought or oversold conditions.

Calculating Volatility and Liquidity: Rolling statistics for volatility and liquidity were computed using the percentage change in price and rolling mean and standard deviation of volume and liquidity. These metrics help in understanding the market dynamics and the variability of price movements.

Calculating Moving Averages: Short and long-term Simple Moving Averages (SMA) and Exponential Moving Averages (EMA) were computed for the price. Moving averages smooth out price data to identify the direction of the trend over different periods. They are crucial for:

> **Simple Moving Average (SMA):**
>
> **Short-Term SMA (`SMA_short`): Calculated over a short window (default is 40 periods).**
>
> **Long-Term SMA (`SMA_long`): Calculated over a longer window (default is 100 periods).**

Purpose and Benefits:

- **SMAs help identify the overall direction of the trend. For example, a short-term SMA crossing above a long-term SMA may signal an upward trend. In this project, SMAs were used to smooth out price data, reducing noise and providing clearer trend signals. These trends assist in generating Buy and Sell signals based on the relative positions of short-term and long-term SMAs.**
- **The SMA is easy to compute and understand. It provides a straightforward measure of the average price over a specific number of periods.**
- **SMA helps identify the overall direction of the price movement. A rising SMA indicates an uptrend, while a falling SMA suggests a downtrend.**
- **Crossovers between short-term and long-term SMAs can generate trading signals. For example, a short-term SMA crossing above a long-term SMA may signal a buying opportunity.**

Exponential Moving Average (EMA):

Short-Term EMA (`EMA_short`): Calculated using an exponentially weighted moving average over a short window.

Long-Term EMA (`EMA_long`): Calculated using an exponentially weighted moving average over a longer window.

Purpose and Benefits:

- **EMAs give more weight to recent prices, making them more responsive to new information. This sensitivity helps identify short-term trends and reversals more effectively than SMAs. EMAs were used in conjunction with RSI to generate more accurate Buy and Sell signals by capturing recent price movements and trend reversals.**
- **Unlike SMA, the EMA gives more weight to recent prices, making it more responsive to new information. This sensitivity makes EMA particularly useful for identifying short-term trends and reversals.**
- **Lag Reduction: The EMA reduces the lag compared to the SMA, allowing traders to react more quickly to price changes.**
- **Complementary Use: Using both SMA and EMA together provides a more comprehensive view of the market. While SMA smooths out price data, EMA captures recent price movements more effectively, helping to confirm signals generated by SMA.**

Generating Actions: Buy, Sell, and Hold actions were generated based on conditions involving RSI and moving averages. These conditions help in creating a basic rule-based strategy to guide the model's learning.

Normalization: The features were normalized using StandardScaler to ensure the data was on a similar scale, which is crucial for the convergence of the learning algorithms. Normalization ensured that all input features contributed equally to the model's learning process, preventing any single feature from disproportionately influencing the model's predictions.

DATASET BALANCING

Random Over-Sampling

- **Purpose: To address class imbalance in the target labels, ensuring that the model does not become biased towards the majority class.**
- **Steps:**
  - **Reshape the data for oversampling.**
  - **Apply `RandomOverSampler` to balance the dataset.**
  - **Reshape the data back to its original sequence shape.**
- **Benefits: Improves the model's ability to learn from minority classes (Buy and Sell actions), leading to more balanced and accurate predictions.**

MODEL IMPLEMENTATION

Transformer Model

A transformer model was implemented using PyTorch. The transformer architecture was chosen due to its superior performance in capturing long-range dependencies and handling sequential data without the limitations of recurrent neural networks (RNNs), such as vanishing gradients.

Model Components and Their Purposes

1. **Embedding Layer:**
   - **Definition:** `self.embedding = nn.Linear(input_dim, 256)`
   - **Purpose:** This layer projects the input features into a higher-dimensional space (256 dimensions). This allows the model to learn more abstract and informative representations of the input data.
   - **Shape Transformation:** Transforms the input data from shape `(batch_size, sequence_length, input_dim)` to `(batch_size, sequence_length, 256)`.
2. **Batch Normalization Layer:**
   - **Definition:** `self.batch_norm = nn.BatchNorm1d(256)`
   - **Purpose:** This layer normalizes the input to the transformer encoder, ensuring that the data has a mean of zero and a standard deviation of one. This helps stabilize and speed up the training process.

- **Shape Transformation:** Applies normalization across the feature dimension.

3. **Transformer Encoder Layers:**

```
encoder_layers = nn.TransformerEncoderLayer(d_model=256,
nhead=num_heads, batch_first=True)

self.transformer_encoder = nn.TransformerEncoder(encoder_layers,
num_layers)
```

   - **Purpose:** The encoder layers capture the sequential dependencies and complex patterns in the data through self-attention mechanisms. The `d_model` parameter specifies the dimension of the input, and `nhead`specifies the number of attention heads. Multiple encoder layers (num_layers) allow the model to learn hierarchical representations.
   - **Shape Transformation:** Processes the data while maintaining the shape `(batch_size, sequence_length, 256)`.

4. **Adaptive Average Pooling Layer:**
   - **Definition:** `self.pooling = nn.AdaptiveAvgPool1d(1)`
   - **Purpose:** This layer aggregates the sequence information to produce a fixed-size vector representing the entire sequence. It reduces the dimensionality of the sequence, focusing on the most informative parts.
   - **Shape Transformation:** Changes the shape from `(batch_size, sequence_length, 256)` to `(batch_size, 256)` by averaging over the sequence length.

5. **Fully Connected Layers:**
   - **First Fully Connected Layer:**
     i. **Definition:** `self.fc1 = nn.Linear(256, 128)`
     ii. **Purpose:** This layer reduces the dimensionality from 256 to 128 and applies a ReLU activation function to introduce non-linearity.
   - **Dropout Layer:**
     i. **Definition:** `self.dropout = nn.Dropout(0.3)`
     ii. **Purpose:** The dropout layer randomly sets a fraction (0.3) of input units to zero during training, which helps prevent overfitting.
   - **Second Fully Connected Layer:**
     i. **Definition:** `self.fc2 = nn.Linear(128, output_dim)`
     ii. **Purpose:** This layer reduces the dimensionality from 128 to the output dimension (3), which corresponds to the three possible actions: Buy, Sell, Hold.

6. **Forward Pass:**
     i. The input features are first embedded into a higher-dimensional space.
     ii. The embedded features are batch normalized.
     iii. The transformer encoder layers process the normalized features to capture sequential patterns.

iv. The sequence information is aggregated using adaptive average pooling.
v. The pooled features are passed through fully connected layers with dropout to produce the final output.

Model Initialization

- **Input Dimension:** `input_dim = len(features)` (the number of selected features)
- **Number of Attention Heads:** `num_heads = 8`
- **Number of Encoder Layers:** `num_layers = 4`
- **Output Dimension:** `output_dim = 3` (representing Buy, Sell, Hold actions)

Purpose and Benefits in this Project

- **Embedding Layer:** Allows the model to learn abstract representations of the input features, enhancing its ability to capture complex relationships in the data.
- **Batch Normalization:** Stabilizes the training process and speeds up convergence.
- **Transformer Encoder Layers:** Capture long-range dependencies and intricate patterns in the sequential data, which are crucial for accurate time series predictions.
- **Adaptive Average Pooling:** Aggregates sequence information into a fixed-size vector, focusing on the most informative parts of the sequence.
- **Fully Connected Layers:** Introduce non-linearity and prevent overfitting, ultimately producing the final action probabilities.
- **Overall Architecture:** The combination of these components enables the model to process and learn from sequential trade and market data effectively, leading to robust trade recommendations.

Model Training and Optimization

Loss Function and Optimizer

- **Loss Function:**
  - **Criterion:** `criterion = nn.CrossEntropyLoss()`
  - **Purpose:** The CrossEntropyLoss function is used for multi-class classification tasks. It computes the difference between the predicted probabilities and the true labels, guiding the model to minimize this difference.
  - **Benefits:** CrossEntropyLoss is suitable for predicting categorical outcomes (Buy, Sell, Hold), and it helps the model learn to distinguish between these classes effectively.

- **Optimizer:**

```
optimizer = optim.Adam(model.parameters(), lr=0.0001)
```

  - **Purpose:** The Adam optimizer is used to update the model's parameters during training. It combines the advantages of two other extensions of stochastic gradient descent (SGD), namely Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp).
  - **Benefits:** Adam provides adaptive learning rates for each parameter, improving convergence speed and performance. It is particularly effective for models with a large number of parameters, like the transformer model.

Learning Rate Scheduler

```
scheduler = ReduceLROnPlateau(optimizer, 'min', patience=3,
factor=0.5)
```

- **Purpose:** The ReduceLROnPlateau scheduler adjusts the learning rate based on the validation loss. If the validation loss does not improve for a specified number of epochs (patience), the learning rate is reduced by a factor.
- **Benefits:** Adjusting the learning rate helps prevent the model from getting stuck in local minima and promotes better convergence. By reducing the learning rate when progress slows, the scheduler helps fine-tune the model towards the end of training.

Early Stopping Mechanism

Early Stopping Class

- **Purpose:** Early stopping monitors the validation loss and stops training if the model's performance does not improve for a specified number of epochs (patience). This prevents overfitting by stopping the training process once the model starts to generalize poorly on unseen data.
- **Benefits:** Early stopping ensures that the model does not overfit the training data, improving its generalization ability on new data. It also saves computational resources by halting training early when further improvements are unlikely.

Training Loop

Training Process

- **Number of Epochs:** `num_epochs = 50`
- **Loss Tracking:** The training loop tracks the loss for each epoch to monitor the model's performance.
- **Steps:**

    - **Model Training:** The model is set to training mode.
    - **Batch Processing:** For each batch of inputs and targets from the dataloader, the optimizer's gradients are zeroed, the forward pass is performed, the loss is calculated, and backpropagation is used to update the model's parameters.
    - **Loss Calculation:** The running loss is accumulated for each batch, and the average loss for the epoch is calculated.
    - **Learning Rate Adjustment:** The learning rate scheduler adjusts the learning rate based on the epoch loss.
    - **Early Stopping:** The early stopping mechanism monitors the epoch loss and determines whether to stop training based on validation performance.

Plotting the Loss Curve

- **Purpose:** Visualizing the loss over epochs helps in understanding the model's learning progress and the effectiveness of the training process.

Purpose and Benefits in this Project

- **Loss Function and Optimizer:** The combination of CrossEntropyLoss and the Adam optimizer ensures effective learning of the model, guiding it to make accurate predictions for Buy, Sell, and Hold actions.
- **Learning Rate Scheduler:** Adjusting the learning rate dynamically helps in fine-tuning the model, leading to better convergence and performance.
- **Early Stopping:** Prevents overfitting and saves computational resources by stopping training early when improvements plateau.
- **Training Loop:** The structured training process, with loss tracking and learning rate adjustments, ensures that the model learns effectively from the data while monitoring for overfitting.

RESULTS & COMPARISON

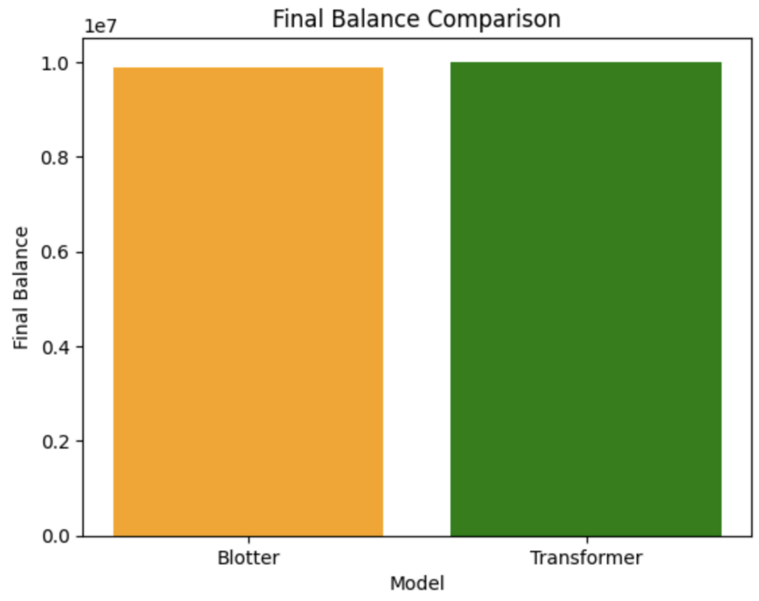
- **Purpose:** To consolidate the performance metrics of the Blotter and Transformer models into a single DataFrame for easy comparison and visualization.

Visualization of Results

Plotting Final Balance

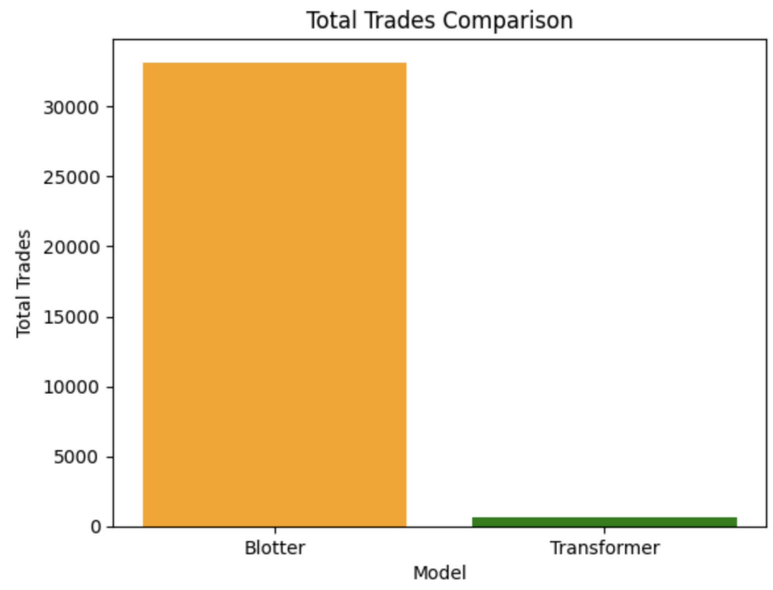
- **Purpose:** To compare the final portfolio balance achieved by the Blotter and Transformer models.
- **Steps:**
    - Create a bar plot to visualize the final balance for each model.
    - Use different colors to distinguish between the models.
    - Label the axes and add a title for clarity.

- **Interpretation:** This plot helps in understanding which model performed better in terms of generating a higher final portfolio balance. A higher final balance indicates a more profitable trading strategy.



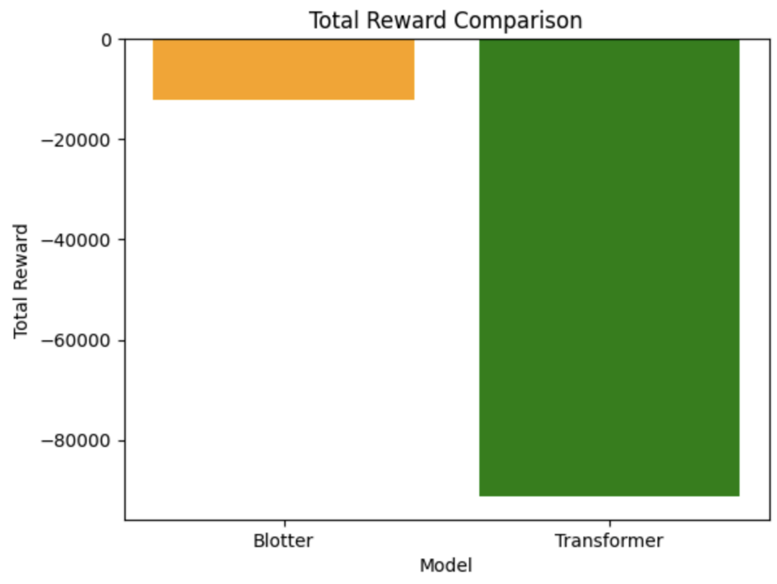
Plotting Total Trades

- **Purpose:** To compare the total number of trades executed by the Blotter and Transformer models.
- **Steps:**
    - Create a bar plot to visualize the total trades for each model.
    - Use different colors to distinguish between the models.
    - Label the axes and add a title for clarity.
- **Interpretation:** This plot helps in understanding the trading frequency of each model. A higher number of trades might indicate a more active trading strategy, while a lower number might indicate a more conservative approach.



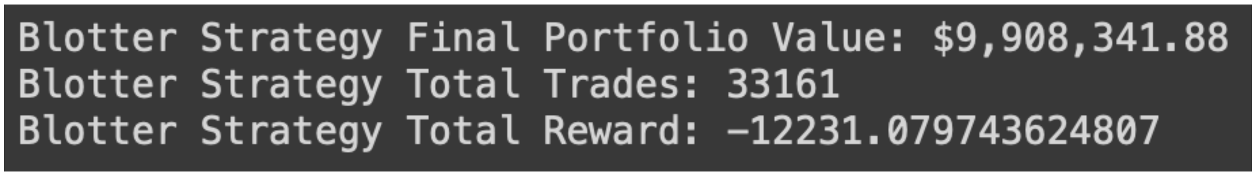
Plotting Total Reward

- **Purpose:** To compare the total reward achieved by the Blotter and Transformer models.
- **Steps:**
    - Create a bar plot to visualize the total reward for each model.
    - Use different colors to distinguish between the models.
    - Label the axes and add a title for clarity.
- **Interpretation:** This plot helps in understanding the overall performance of each model in terms of cumulative reward. A higher total reward indicates better overall performance, considering factors such as transaction costs, slippage, and time penalties.

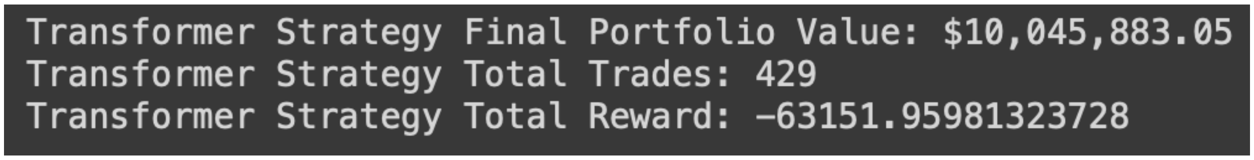


Results DataFrame

- **Purpose:** To display the consolidated performance metrics in tabular form for easy
- reference.
- **Interpretation:** The DataFrame provides a clear comparison of the key performance metrics (Final Balance, Total Trades, Total Reward) for both models. This helps in making an informed decision about which model performed better overall.



```
Blotter Strategy Final Portfolio Value: $9,908,341.88
Blotter Strategy Total Trades: 33161
Blotter Strategy Total Reward: -12231.079743624807
```



```
Transformer Strategy Final Portfolio Value: $10,045,883.05
Transformer Strategy Total Trades: 429
Transformer Strategy Total Reward: -63151.95981323728
```

Purpose and Benefits in this Project

- **Visualization:** Visualizing the results using bar plots provides a clear and intuitive understanding of the performance differences between the Blotter and Transformer models.
- **Comparison:** By comparing the Final Balance, Total Trades, and Total Reward, we can evaluate the effectiveness of each trading strategy and identify areas for improvement.
- **Decision-Making:** The visualizations and consolidated metrics help in making data-driven decisions about model selection and further enhancements.