COS30049 - Intelligent System

MACHINE LEARNING

TRAN DUC ANH DANG

Student ID: 103995439

Requirements

Virtual Environment

- Google Colab
- Jupyter Notebook

Decencies

- numpy
- matplotlib
- mplfinance
- pandas
- scikit-learn
- pandas-datareader
- yfinance
- pandas_ta
- joblib

Installation

*Note: Anaconda is required unless Google Collab is being used

Anaconda/Virtual Environment

1. Download Anaconda: Go to the Anaconda website (https://www.anaconda.com/products/distribution) and download the appropriate version for your operating system.
2. Install Anaconda: Follow the installation instructions for the OS from the Anaconda website.
3. Open Anaconda Navigator: Launch Anaconda Navigator from your installed applications.
4. Create a New Environment (Required): Create a new environment to isolate Jupyter installation on each project. Click on "Environments" in Navigator and then "Create" to make a new environment.
5. Install Jupyter Notebook: In the selected environment, click on the environment name and select "Open Terminal". In the terminal, type: conda install jupyter.

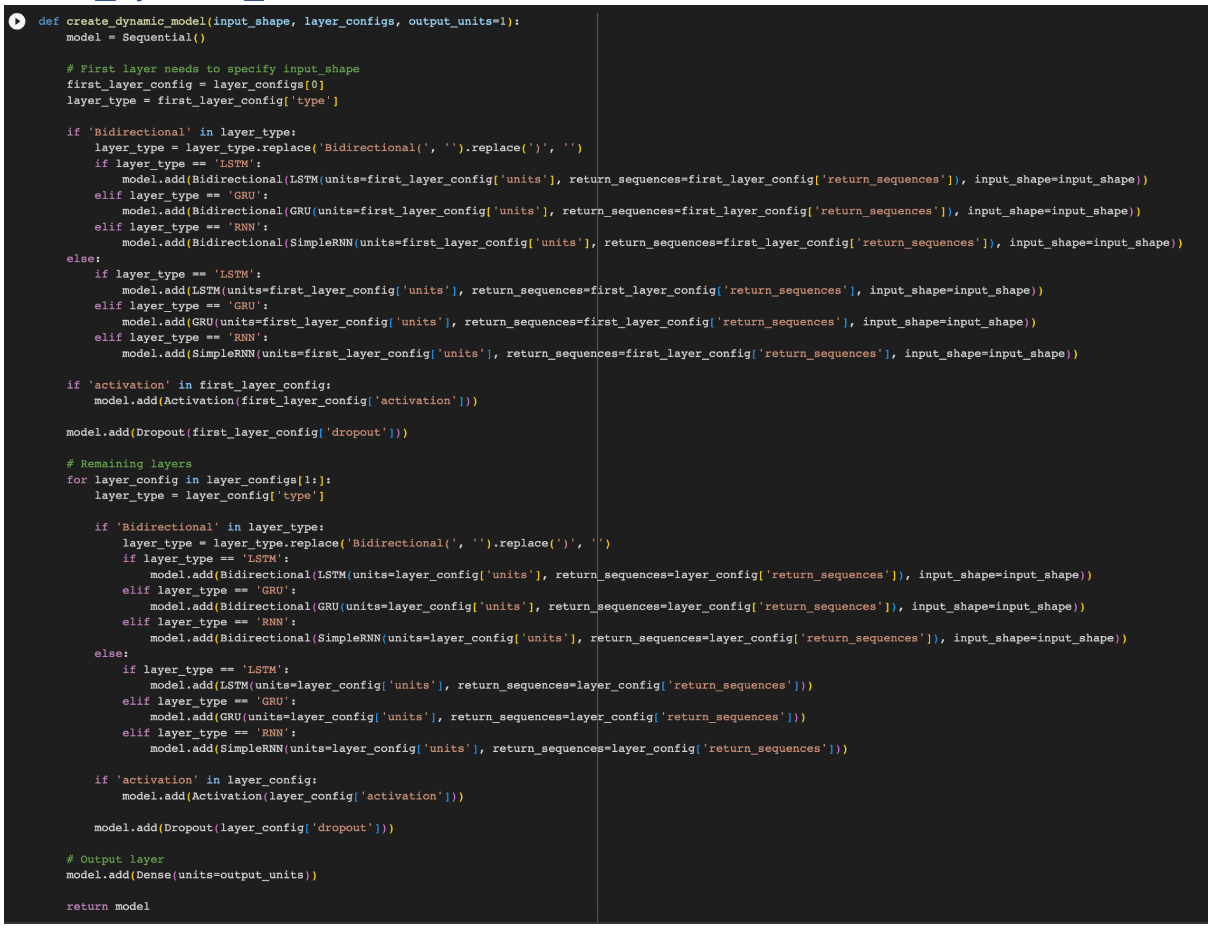
Dependencies

In Google Colab or Jupyter Notebook, it can directly install the required dependencies using the !pip command in code cells. Here's an example of how to install the dependencies:

!pip install <package> or !pip install -r <text file>

Machine Learning 1

create_dynamic_model



```python
def create_dynamic_model(input_shape, layer_configs, output_units=1):
    model = Sequential()

    # First layer needs to specify input_shape
    first_layer_config = layer_configs[0]
    layer_type = first_layer_config['type']

    if 'Bidirectional' in layer_type:
        layer_type = layer_type.replace('Bidirectional(', '').replace(')', '')
        if layer_type == 'LSTM':
            model.add(Bidirectional(LSTM(units=first_layer_config['units'], return_sequences=first_layer_config['return_sequences']), input_shape=input_shape))
        elif layer_type == 'GRU':
            model.add(Bidirectional(GRU(units=first_layer_config['units'], return_sequences=first_layer_config['return_sequences']), input_shape=input_shape))
        elif layer_type == 'RNN':
            model.add(Bidirectional(SimpleRNN(units=first_layer_config['units'], return_sequences=first_layer_config['return_sequences']), input_shape=input_shape))
    else:
        if layer_type == 'LSTM':
            model.add(LSTM(units=first_layer_config['units'], return_sequences=first_layer_config['return_sequences'], input_shape=input_shape))
        elif layer_type == 'GRU':
            model.add(GRU(units=first_layer_config['units'], return_sequences=first_layer_config['return_sequences'], input_shape=input_shape))
        elif layer_type == 'RNN':
            model.add(SimpleRNN(units=first_layer_config['units'], return_sequences=first_layer_config['return_sequences'], input_shape=input_shape))

    if 'activation' in first_layer_config:
        model.add(Activation(first_layer_config['activation']))

    model.add(Dropout(first_layer_config['dropout']))

    # Remaining layers
    for layer_config in layer_configs[1:]:
        layer_type = layer_config['type']

        if 'Bidirectional' in layer_type:
            layer_type = layer_type.replace('Bidirectional(', '').replace(')', '')
            if layer_type == 'LSTM':
                model.add(Bidirectional(LSTM(units=layer_config['units'], return_sequences=layer_config['return_sequences']), input_shape=input_shape))
            elif layer_type == 'GRU':
                model.add(Bidirectional(GRU(units=layer_config['units'], return_sequences=layer_config['return_sequences']), input_shape=input_shape))
            elif layer_type == 'RNN':
                model.add(Bidirectional(SimpleRNN(units=layer_config['units'], return_sequences=layer_config['return_sequences']), input_shape=input_shape))
        else:
            if layer_type == 'LSTM':
                model.add(LSTM(units=layer_config['units'], return_sequences=layer_config['return_sequences']))
            elif layer_type == 'GRU':
                model.add(GRU(units=layer_config['units'], return_sequences=layer_config['return_sequences']))
            elif layer_type == 'RNN':
                model.add(SimpleRNN(units=layer_config['units'], return_sequences=layer_config['return_sequences']))

        if 'activation' in layer_config:
            model.add(Activation(layer_config['activation']))

        model.add(Dropout(layer_config['dropout']))

    # Output layer
    model.add(Dense(units=output_units))

    return model
```

The function `create_dynamic_model` takes the following parameters:

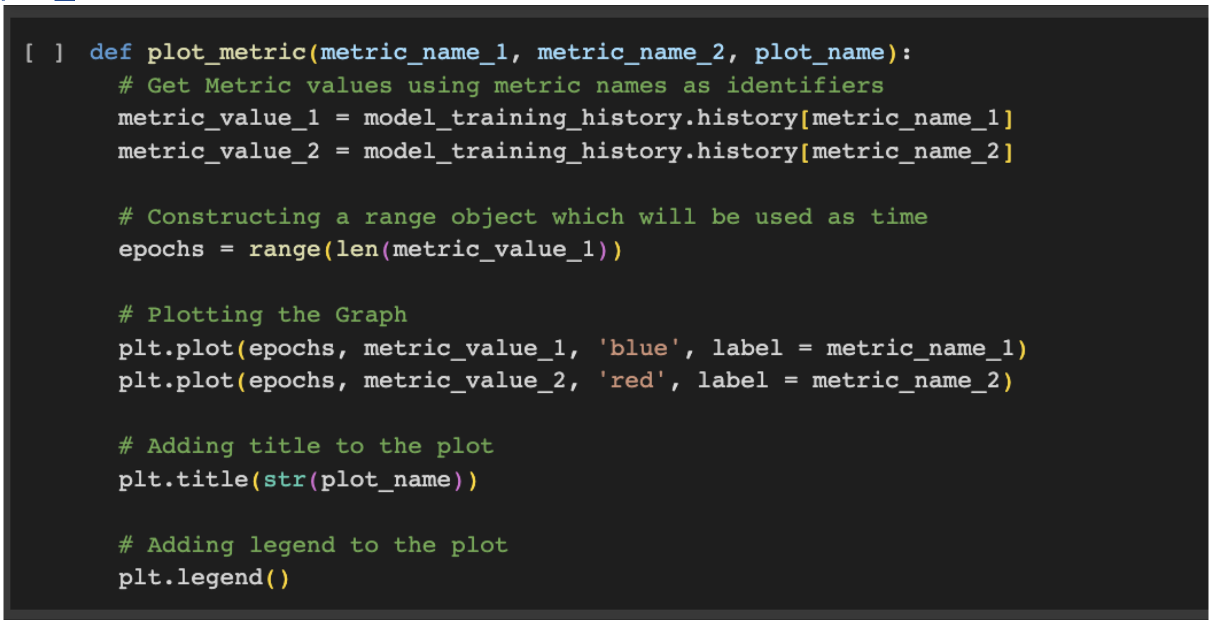
1. `input_shape`: This parameter, a tuple, denotes the shape of the input data that will be fed into the neural network.
2. `layer_configs`: A list of dictionaries representing the configuration for each layer in the neural network. The configurations include details like the type of the layer (LSTM, GRU, RNN, etc.), the number of units, whether the layer should return sequences, activation function, and the dropout rate.
3. `output_units`: An optional parameter with a default value of 1. It indicates the number of units in the output layer of the neural network.

This function exhibits the following characteristics:

1. Initialization of Sequential Model: The function initiates a Sequential model which serves as the framework to which various layers will be added based on the configurations specified in `layer_configs`.
2. First Layer Configuration: The function configures the first layer based on the details provided in the first dictionary of `layer_configs`. It includes handling different types of layers (LSTM, GRU, and SimpleRNN) and potentially wrapping them with a Bidirectional layer. The input shape for this layer is specified by the `input_shape` parameter.

3. Adding Activation and Dropout to First Layer: If an activation function is specified in the configuration of the first layer, it is added following the layer. A dropout layer is then added with the specified dropout rate to prevent overfitting.
4. Configuration of Subsequent Layers: Following the first layer, the function iterates through the remaining configurations in `layer_configs` to add subsequent layers to the model. Similar to the first layer, these can be of different types (LSTM, GRU, SimpleRNN) and can be wrapped with a Bidirectional layer if specified. Activation functions and dropout layers are added as specified in each configuration.
5. Output Layer Addition: Finally, an output layer with the specified number of units (`output_units`) is added to the model to obtain the final output.
6. Returning the Model: After configuring and assembling all layers, the function returns the constructed Sequential model which can then be compiled and used for training with data.

plot_metric



```
[ ] def plot_metric(metric_name_1, metric_name_2, plot_name):
      # Get Metric values using metric names as identifiers
      metric_value_1 = model_training_history.history[metric_name_1]
      metric_value_2 = model_training_history.history[metric_name_2]

      # Constructing a range object which will be used as time
      epochs = range(len(metric_value_1))

      # Plotting the Graph
      plt.plot(epochs, metric_value_1, 'blue', label = metric_name_1)
      plt.plot(epochs, metric_value_2, 'red', label = metric_name_2)

      # Adding title to the plot
      plt.title(str(plot_name))

      # Adding legend to the plot
      plt.legend()
```

The function `plot_metric` takes the following parameters:

1. `metric_name_1`: A string representing the name of the first metric to be plotted. This name is used to retrieve the metric values from the `model_training_history.history` dictionary.
2. `metric_name_2`: Similarly, this is a string representing the name of the second metric to be plotted. The metric values are retrieved from the `model_training_history.history` dictionary.
3. `plot_name`: A string that will be used as the title of the plot, helping in identifying the graph.

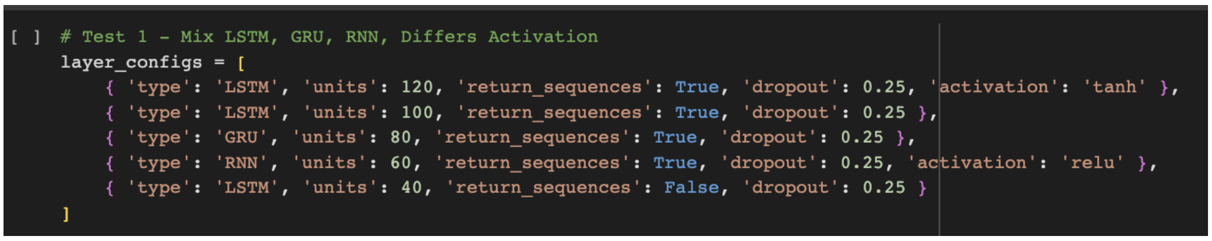
This function performs the following operations:

1. Fetching Metric Values: Using the metric names specified by `metric_name_1` and `metric_name_2` as identifiers, the function fetches the metric values from the `model_training_history.history` dictionary.

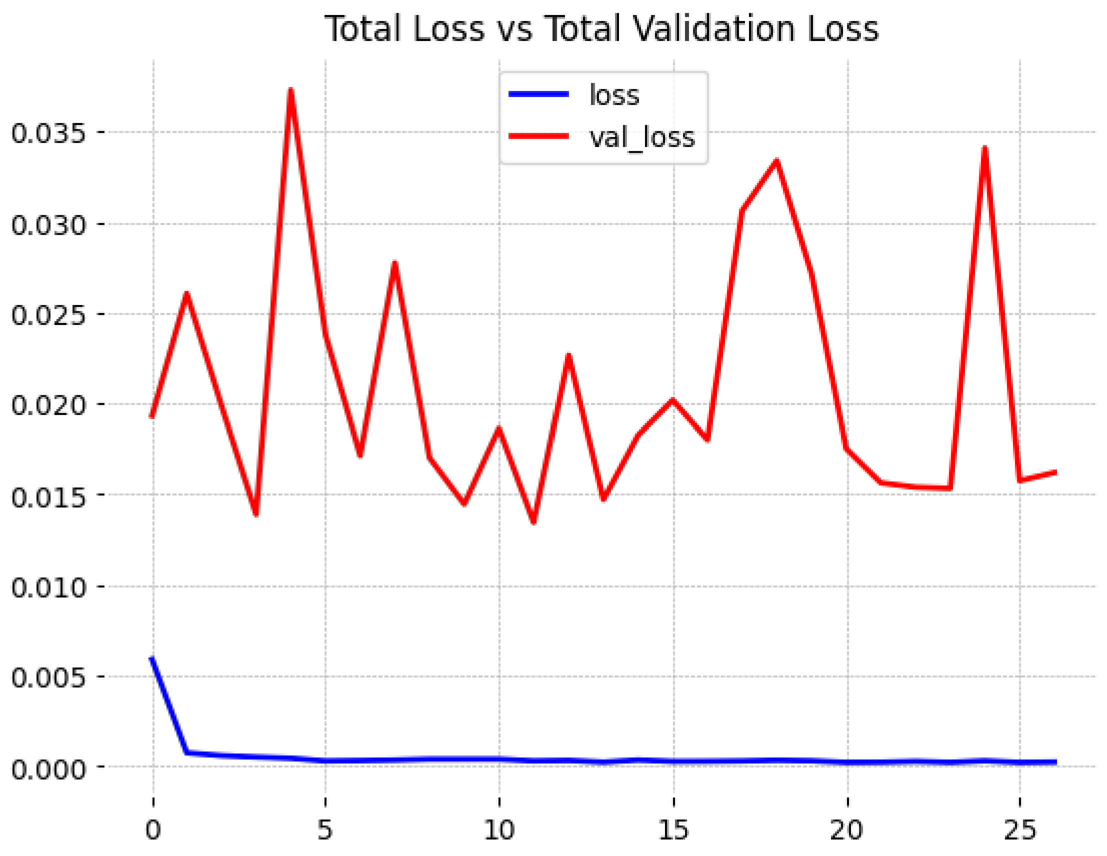
2. Epoch Range Calculation: The function calculates the number of epochs by determining the length of the metric values list (obtained using `metric_name_1`). This range is used as the time or x-axis in the plot.
3. Plotting the Graph: Using matplotlib's plot function, it plots the metric values against the epoch range. The plot for `metric_name_1` is displayed in blue, and the plot for `metric_name_2` is displayed in red.
4. Adding Title to the Plot: It sets the title of the plot to the value specified by the `plot_name` parameter, assisting in identifying the content of the graph.
5. Adding Legend to the Plot: Finally, a legend is added to the plot, which helps in distinguishing between the two metric plots, with labels indicating which line corresponds to `metric_name_1` and which corresponds to `metric_name_2`.

Model's Performance

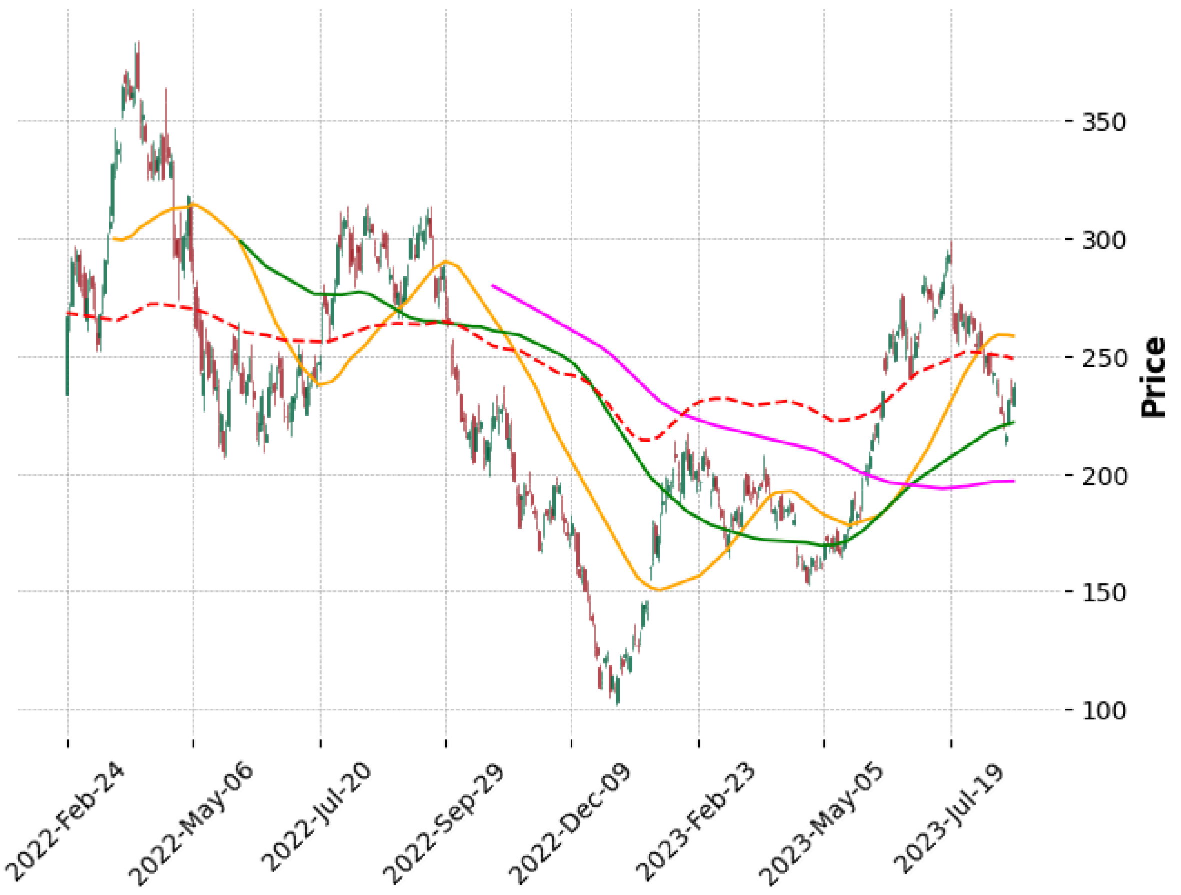
Layer Configs 1 - Mix LSTM, GRU, RNN, Differents Activations



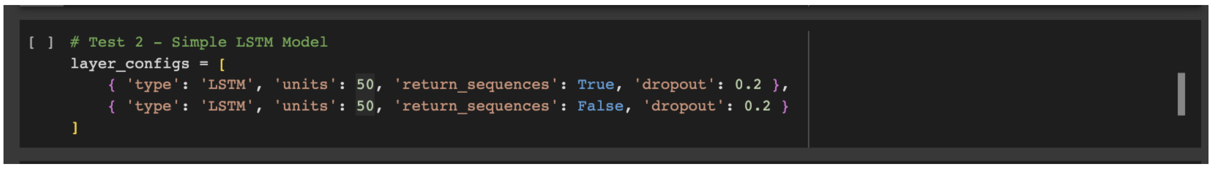
```
[ ]  # Test 1 - Mix LSTM, GRU, RNN, Differs Activation
     layer_configs = [
         { 'type': 'LSTM', 'units': 120, 'return_sequences': True, 'dropout': 0.25, 'activation': 'tanh' },
         { 'type': 'LSTM', 'units': 100, 'return_sequences': True, 'dropout': 0.25 },
         { 'type': 'GRU', 'units': 80, 'return_sequences': True, 'dropout': 0.25 },
         { 'type': 'RNN', 'units': 60, 'return_sequences': True, 'dropout': 0.25, 'activation': 'relu' },
         { 'type': 'LSTM', 'units': 40, 'return_sequences': False, 'dropout': 0.25 }
     ]
```



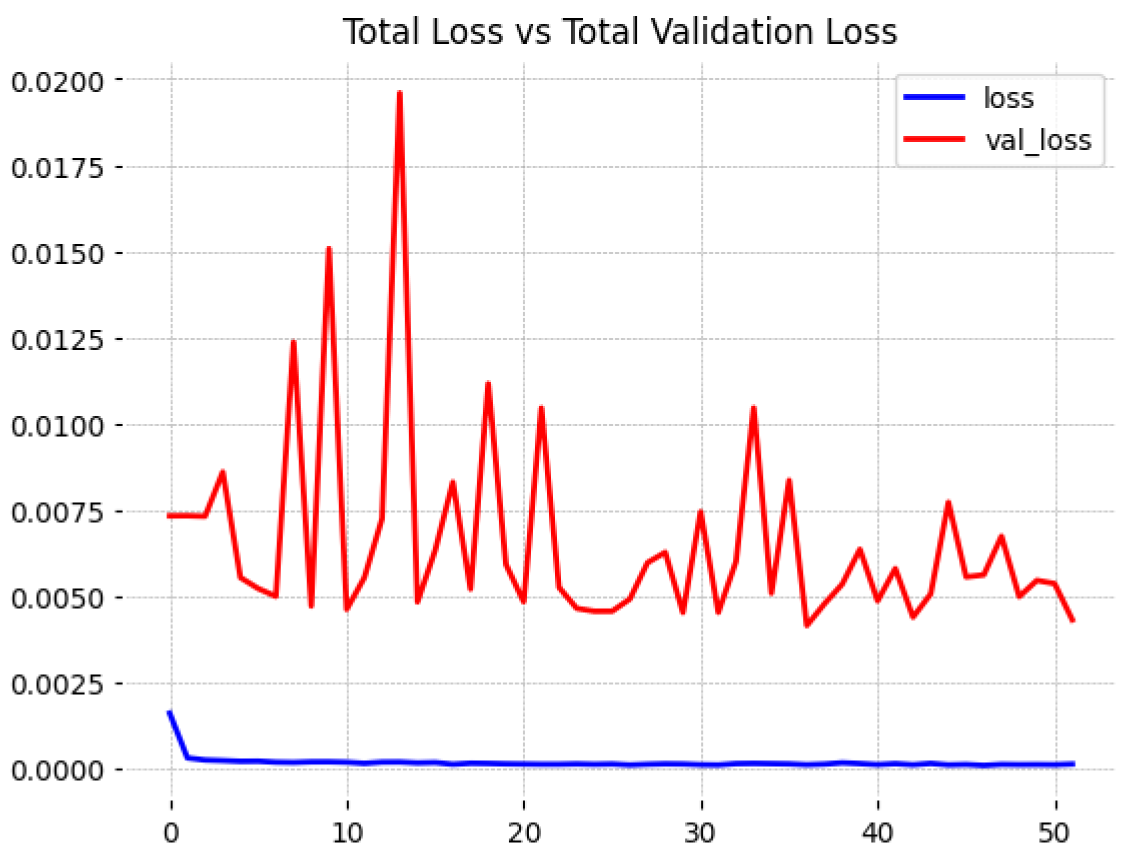
TSLACandlestick Chart



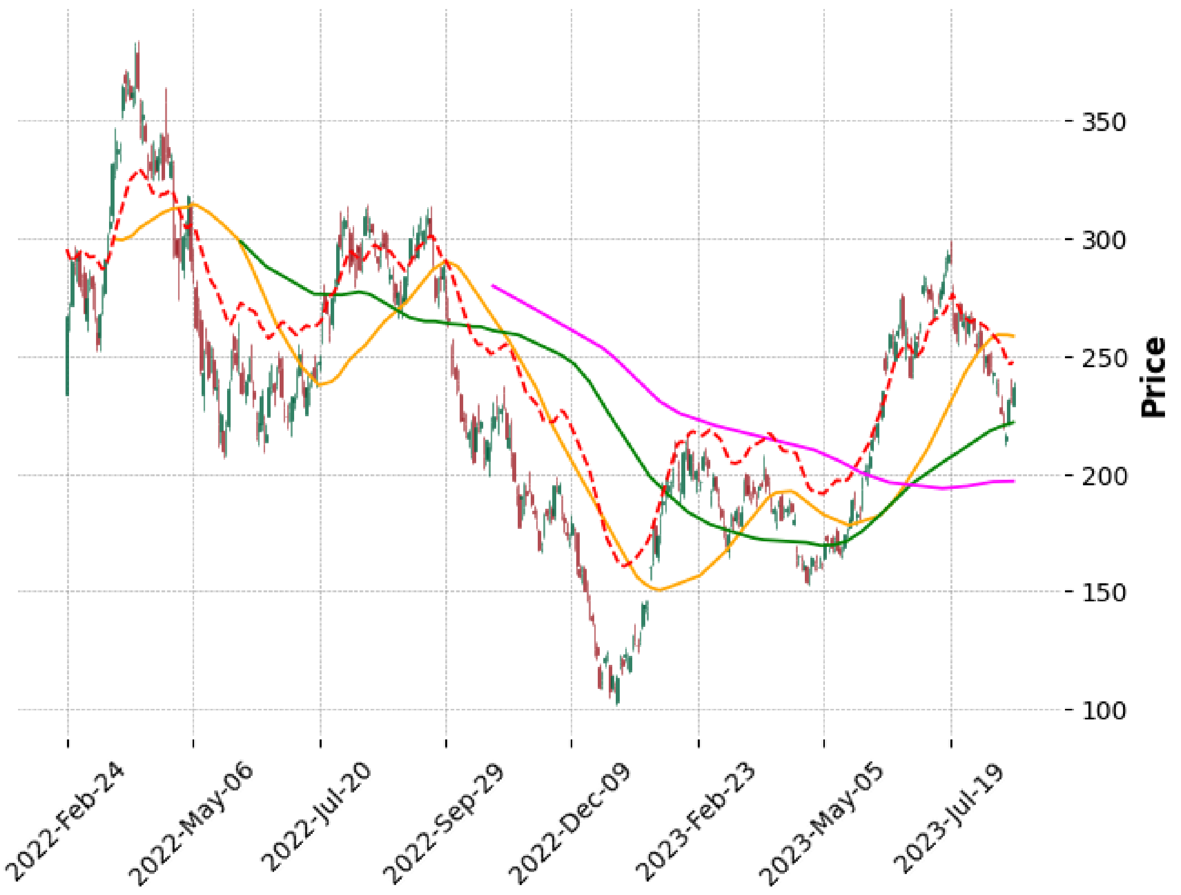
Layer Configs 2 - Simple LSTM Model



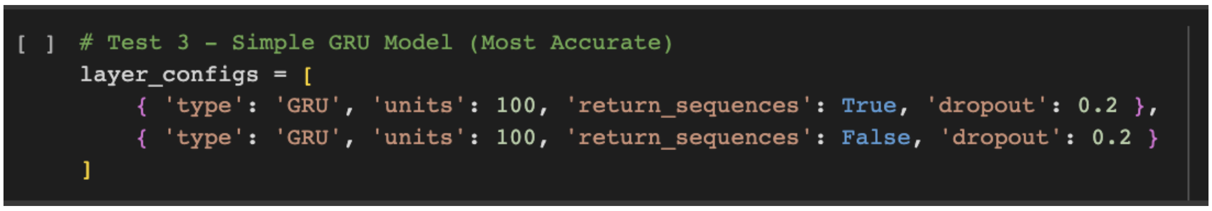
```
[ ]  # Test 2 - Simple LSTM Model
     layer_configs = [
         { 'type': 'LSTM', 'units': 50, 'return_sequences': True, 'dropout': 0.2 },
         { 'type': 'LSTM', 'units': 50, 'return_sequences': False, 'dropout': 0.2 }
     ]
```



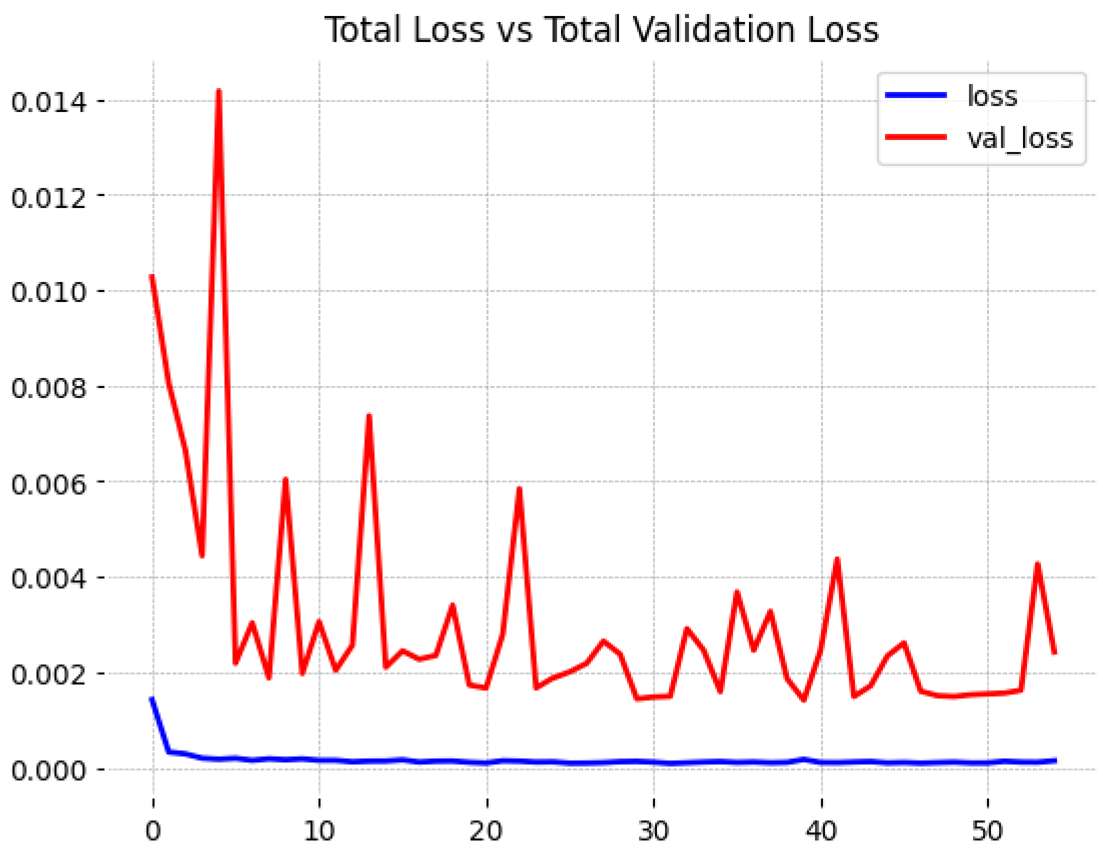
TSLACandlestick Chart



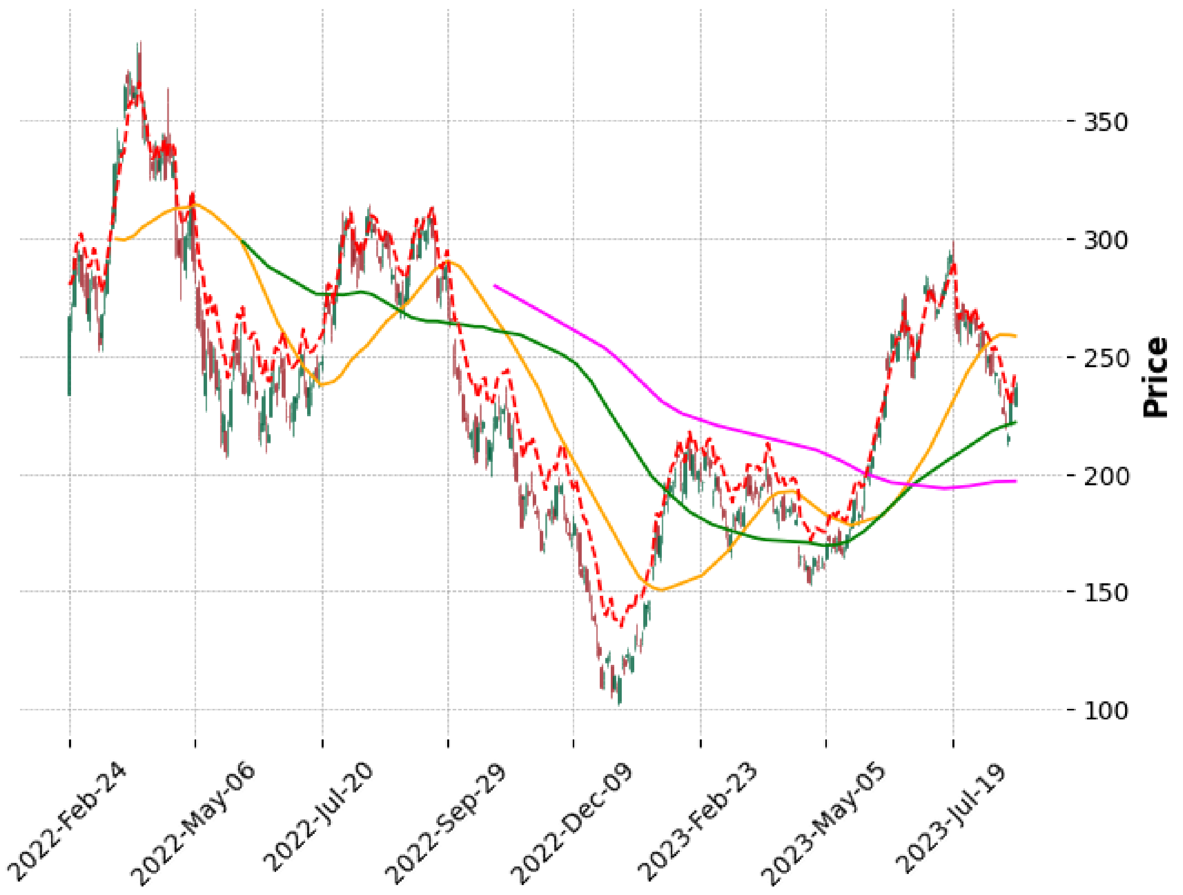
Layer Configs 3 - Simple GRU Model (Most Accurate)



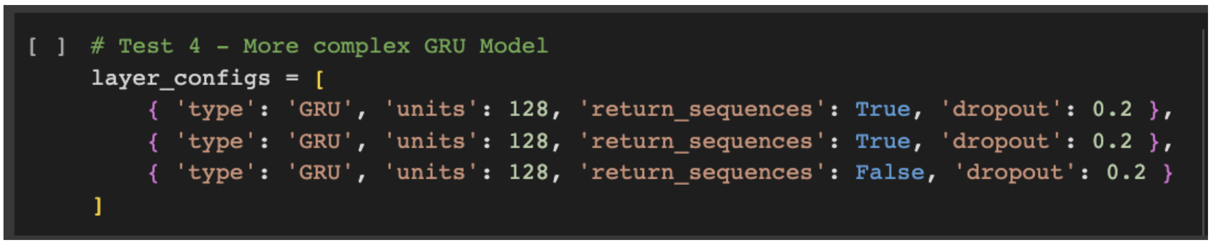
```
[ ]  # Test 3 - Simple GRU Model (Most Accurate)
     layer_configs = [
         { 'type': 'GRU', 'units': 100, 'return_sequences': True, 'dropout': 0.2 },
         { 'type': 'GRU', 'units': 100, 'return_sequences': False, 'dropout': 0.2 }
     ]
```



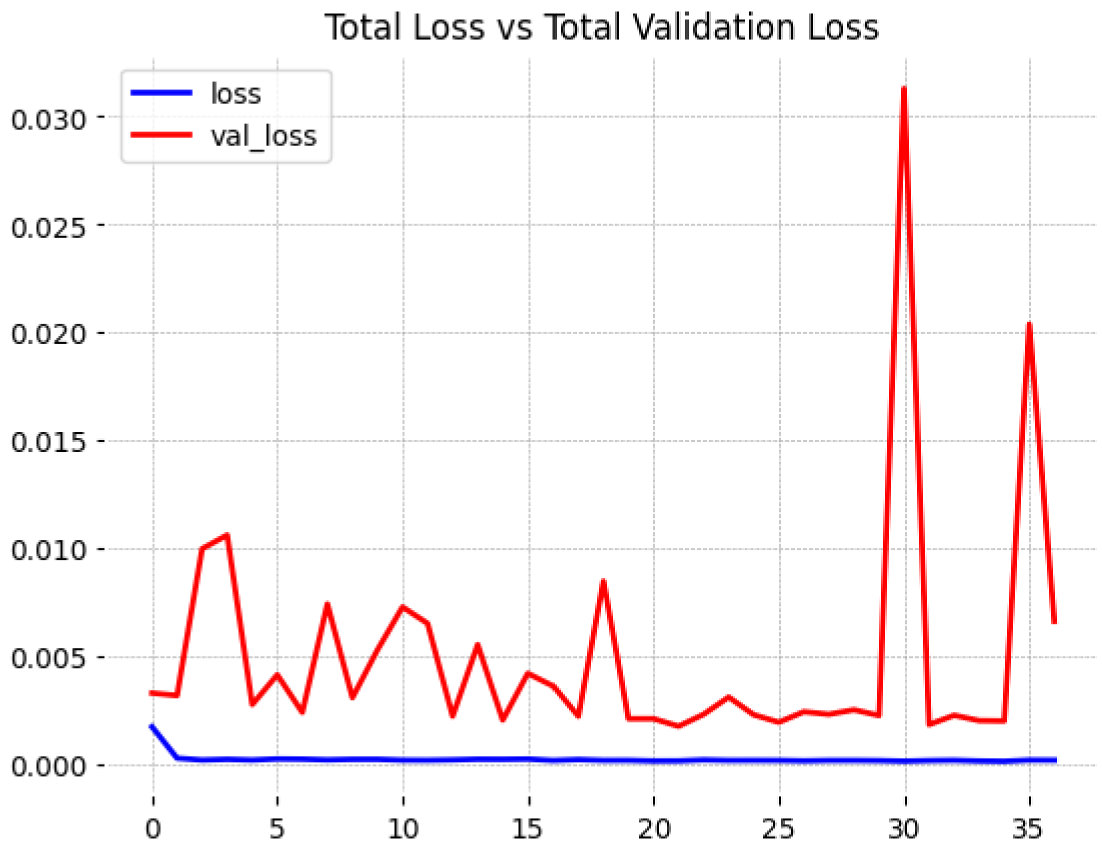
TSLACandlestick Chart



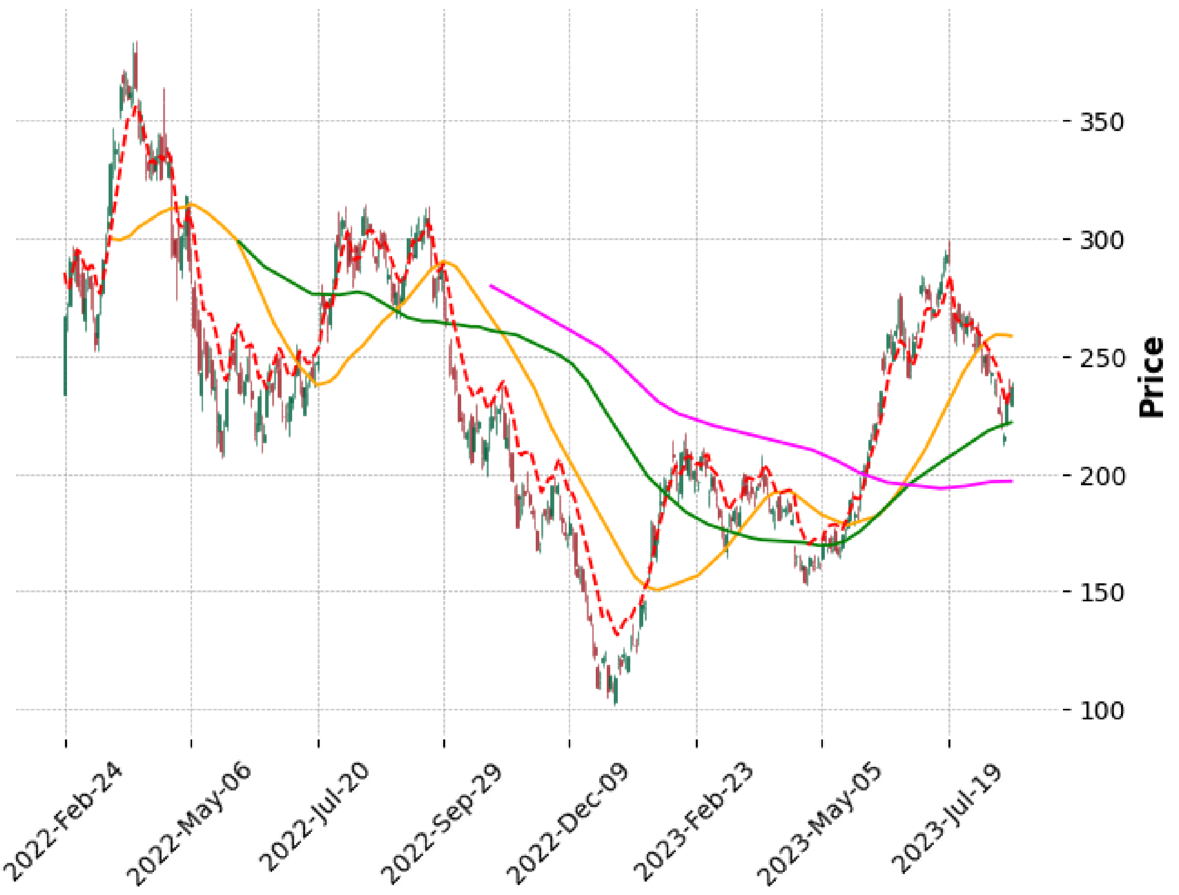
Layer Configs 4 - More complex GRU Model



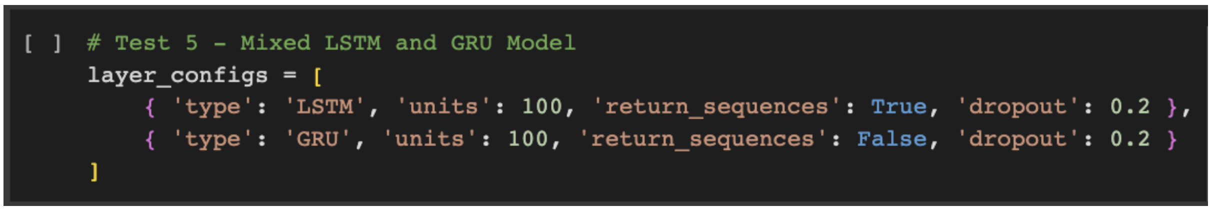
```
[ ] # Test 4 - More complex GRU Model
    layer_configs = [
        { 'type': 'GRU', 'units': 128, 'return_sequences': True, 'dropout': 0.2 },
        { 'type': 'GRU', 'units': 128, 'return_sequences': True, 'dropout': 0.2 },
        { 'type': 'GRU', 'units': 128, 'return_sequences': False, 'dropout': 0.2 }
    ]
```



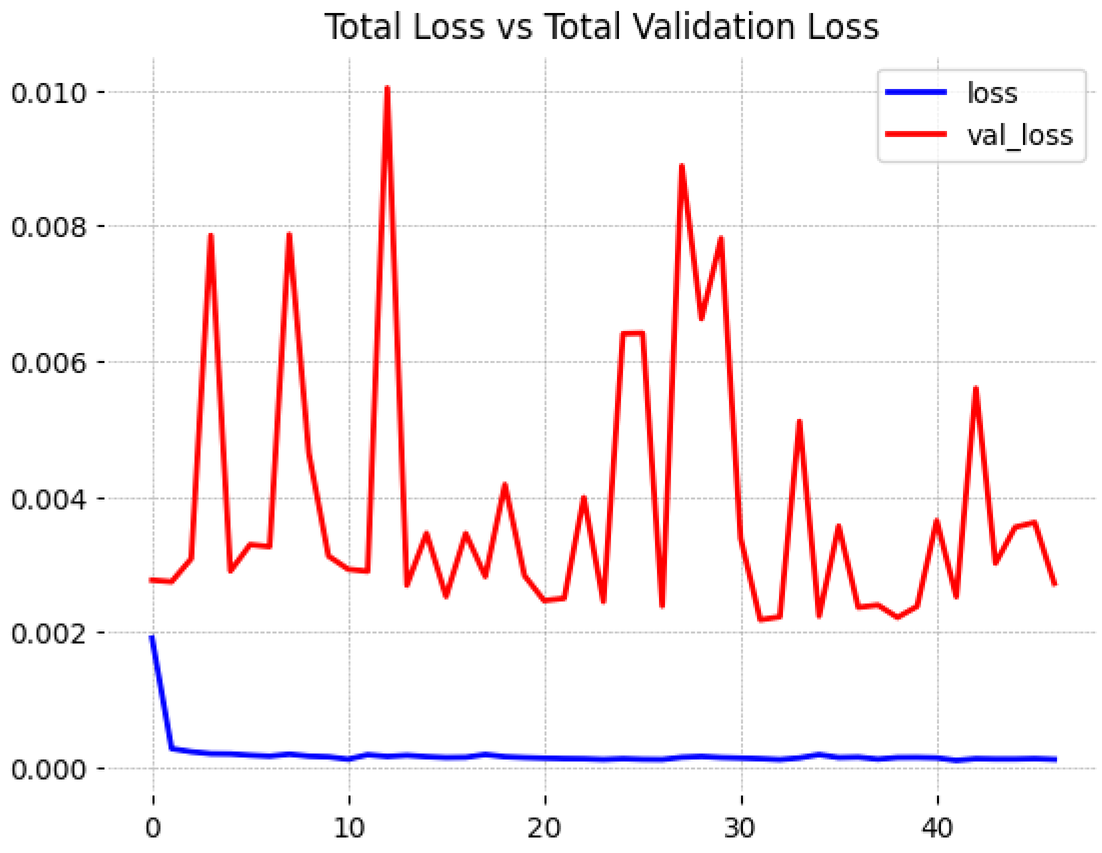
TSLACandlestick Chart



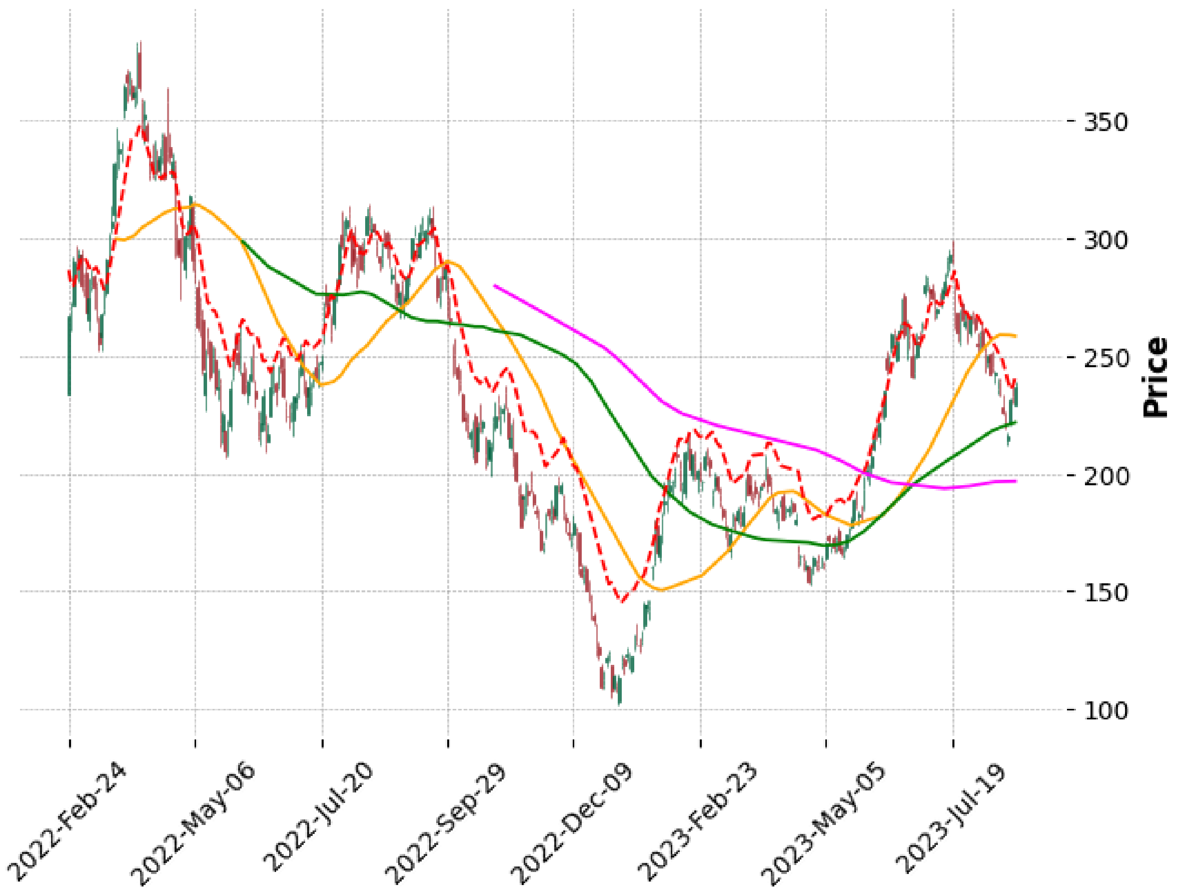
Layer Configs 5 - Mixed LSTM and GRU Model



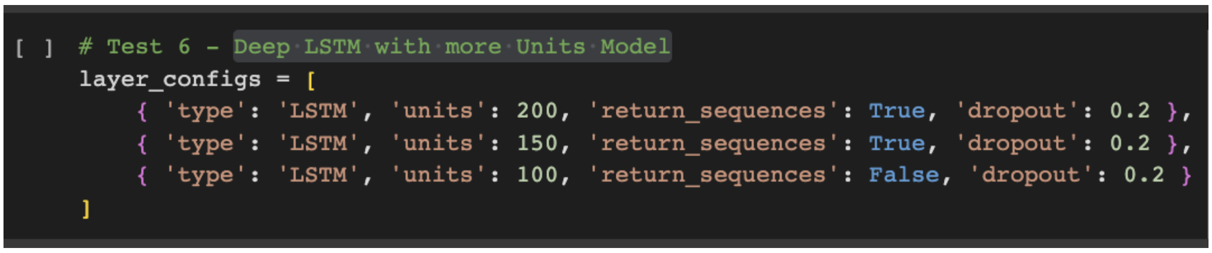
```
[ ]  # Test 5 - Mixed LSTM and GRU Model
     layer_configs = [
         { 'type': 'LSTM', 'units': 100, 'return_sequences': True, 'dropout': 0.2 },
         { 'type': 'GRU', 'units': 100, 'return_sequences': False, 'dropout': 0.2 }
     ]
```



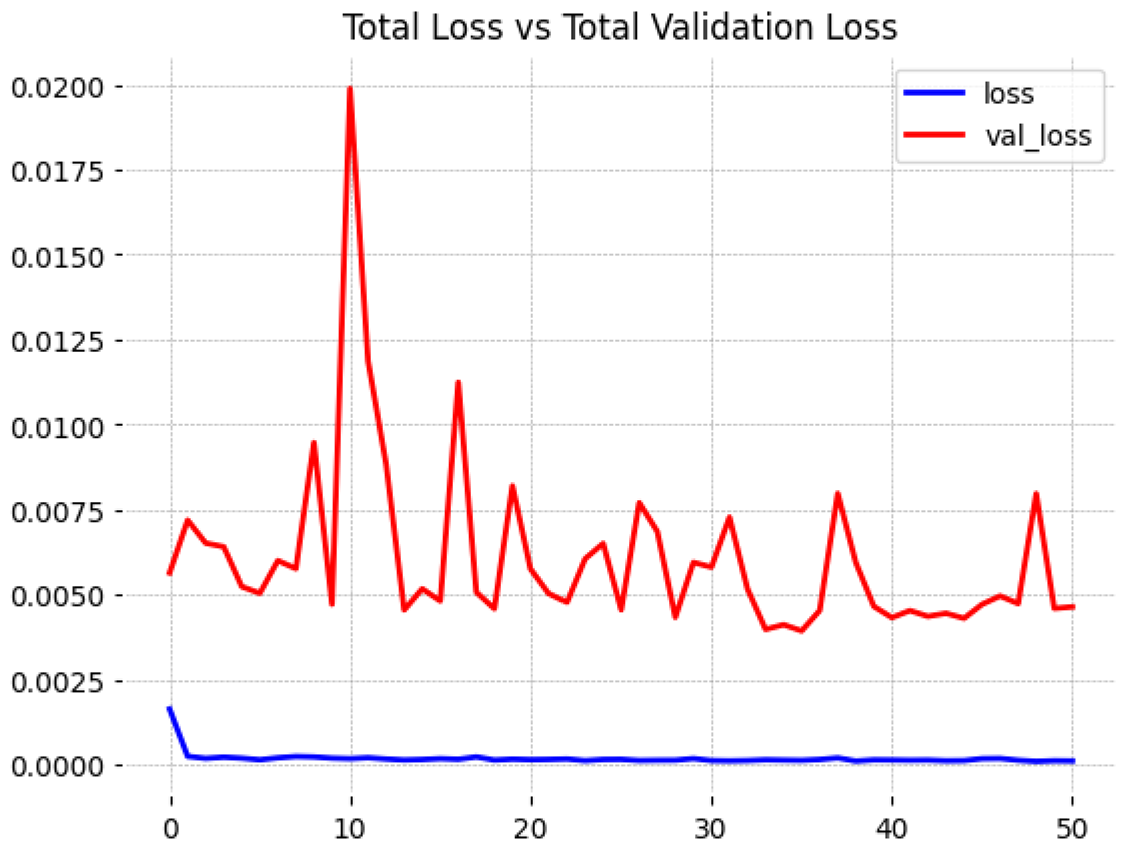
TSLACandlestick Chart



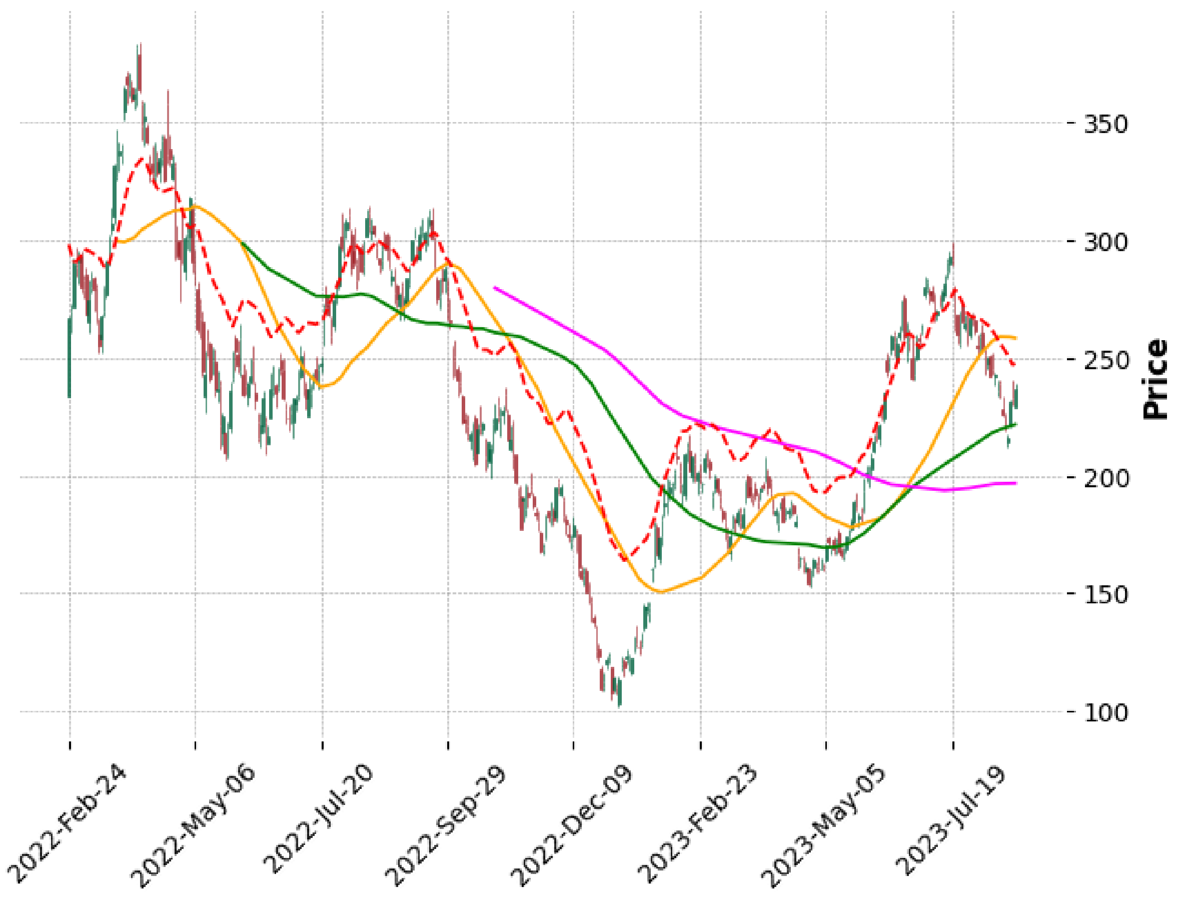
Layer Configs 6 - Deep LSTM with more Units Model



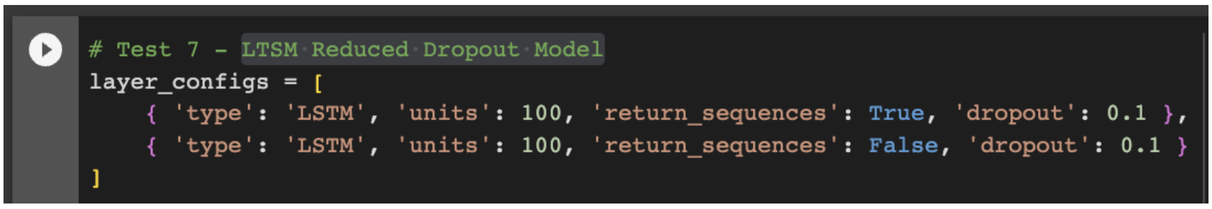
```
[ ] # Test 6 - Deep LSTM with more Units Model
    layer_configs = [
        { 'type': 'LSTM', 'units': 200, 'return_sequences': True, 'dropout': 0.2 },
        { 'type': 'LSTM', 'units': 150, 'return_sequences': True, 'dropout': 0.2 },
        { 'type': 'LSTM', 'units': 100, 'return_sequences': False, 'dropout': 0.2 }
    ]
```



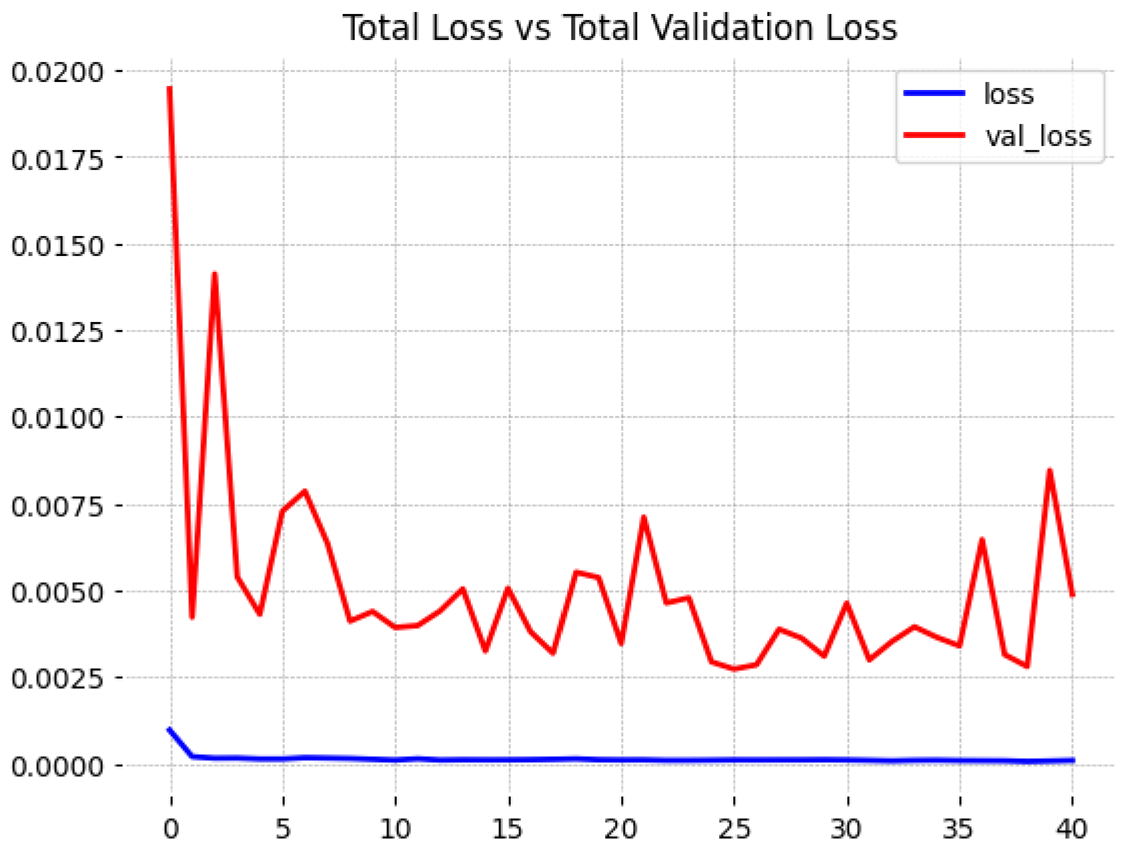
TSLACandlestick Chart



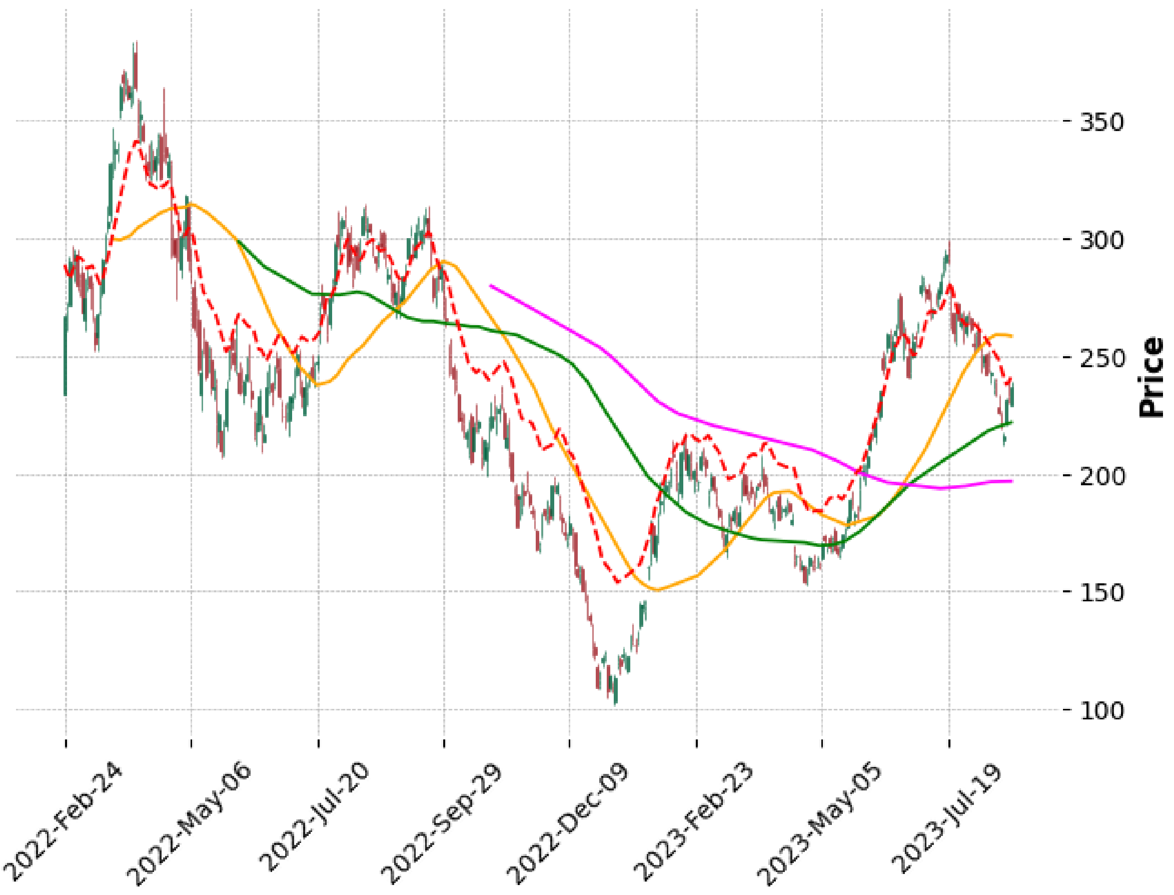
Layer Configs 7 - LTSM Reduced Dropout Model



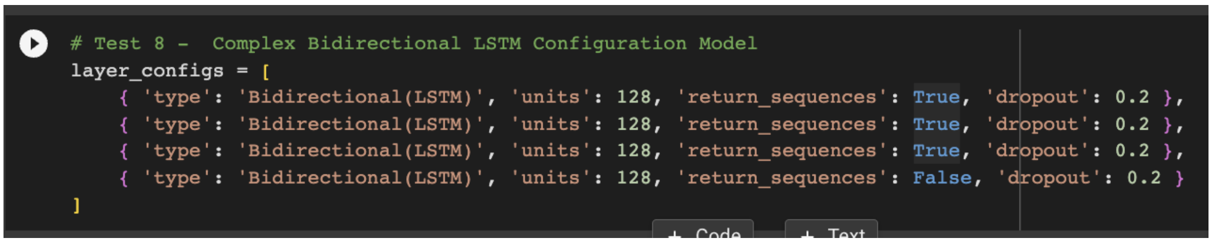
```
# Test 7 - LTSM Reduced Dropout Model
layer_configs = [
    { 'type': 'LSTM', 'units': 100, 'return_sequences': True, 'dropout': 0.1 },
    { 'type': 'LSTM', 'units': 100, 'return_sequences': False, 'dropout': 0.1 }
]
```



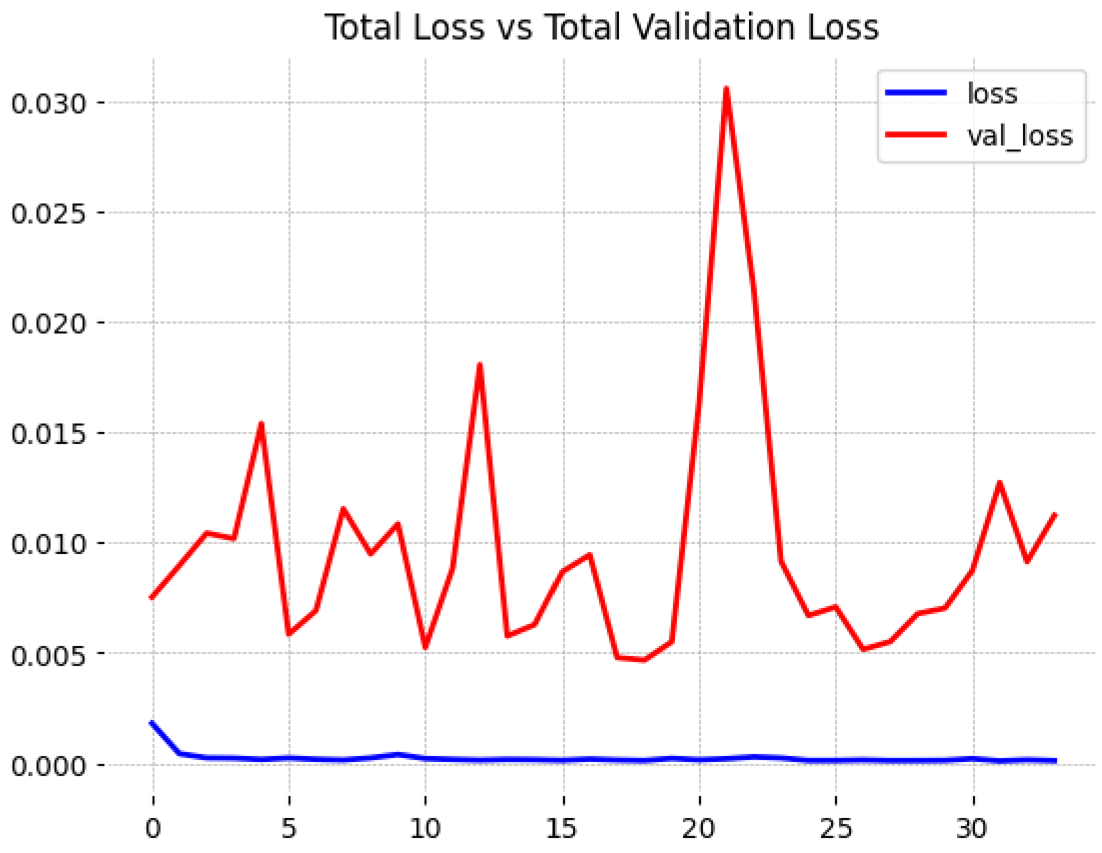
TSLACandlestick Chart

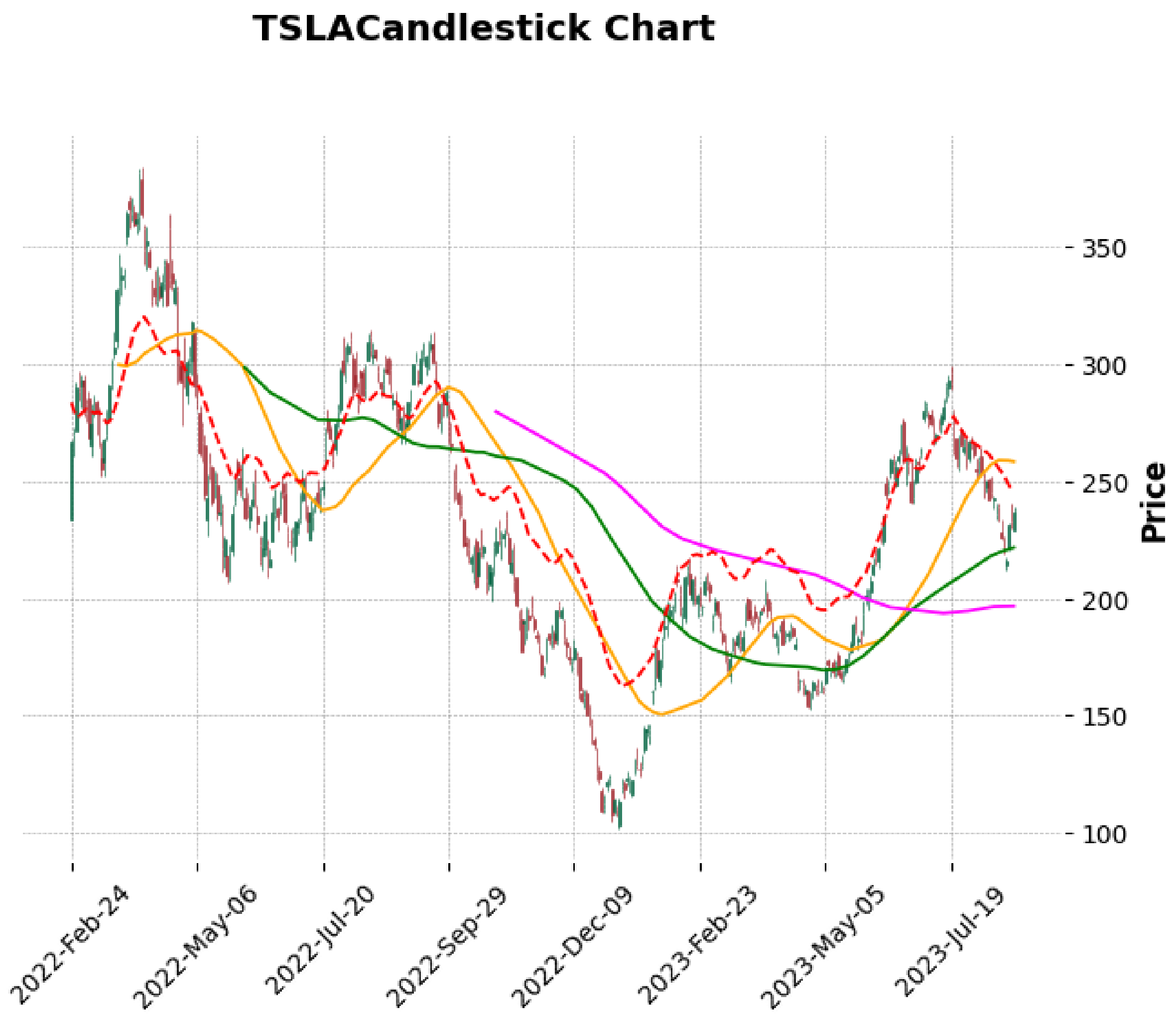


Layer Configs 8 - Complex Bidirectional LSTM Configuration Model



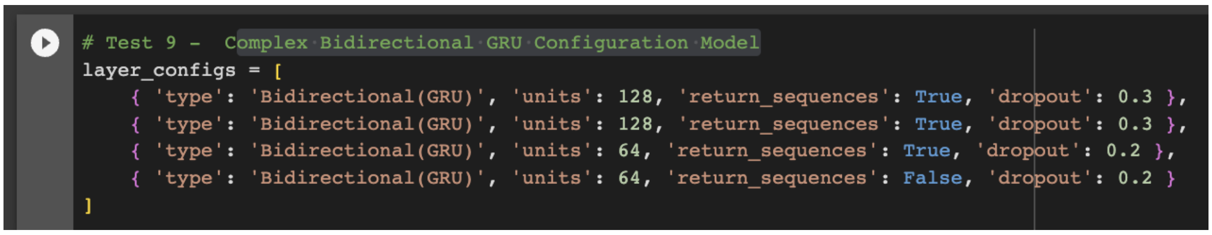
```
# Test 8 -  Complex Bidirectional LSTM Configuration Model
layer_configs = [
    { 'type': 'Bidirectional(LSTM)', 'units': 128, 'return_sequences': True, 'dropout': 0.2 },
    { 'type': 'Bidirectional(LSTM)', 'units': 128, 'return_sequences': True, 'dropout': 0.2 },
    { 'type': 'Bidirectional(LSTM)', 'units': 128, 'return_sequences': True, 'dropout': 0.2 },
    { 'type': 'Bidirectional(LSTM)', 'units': 128, 'return_sequences': False, 'dropout': 0.2 }
]
```



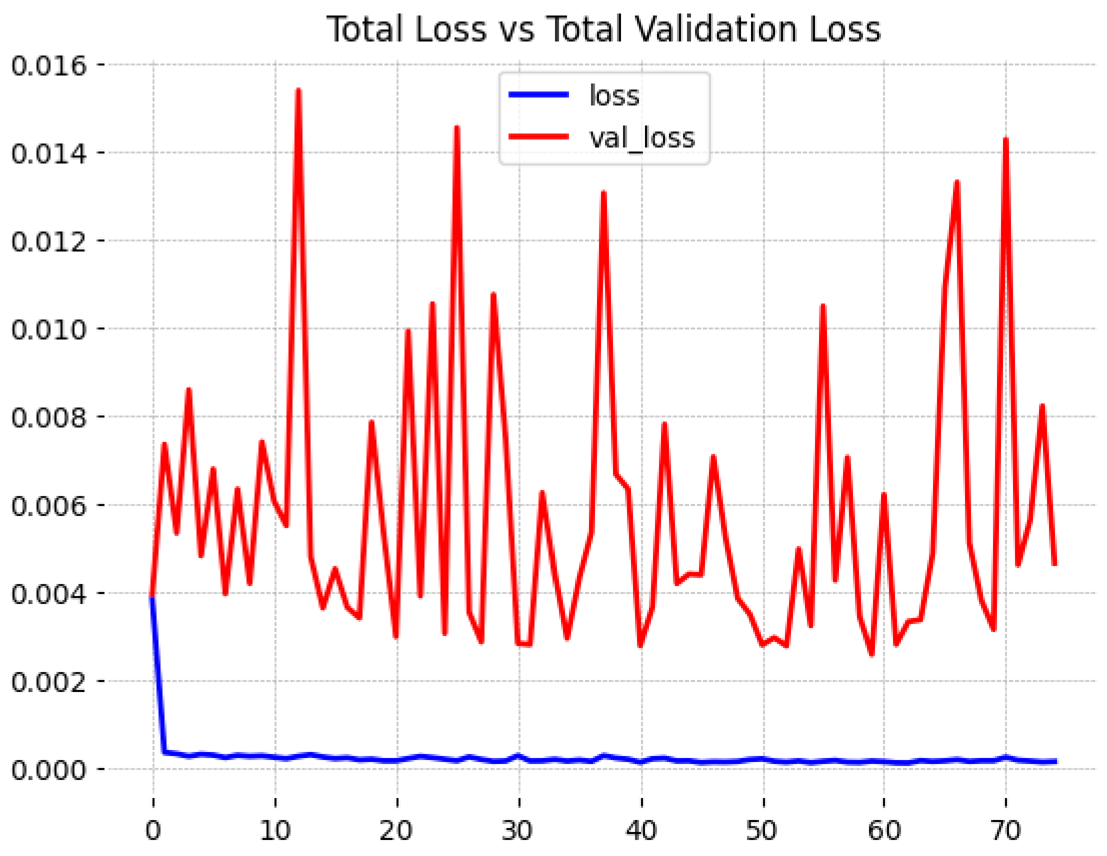


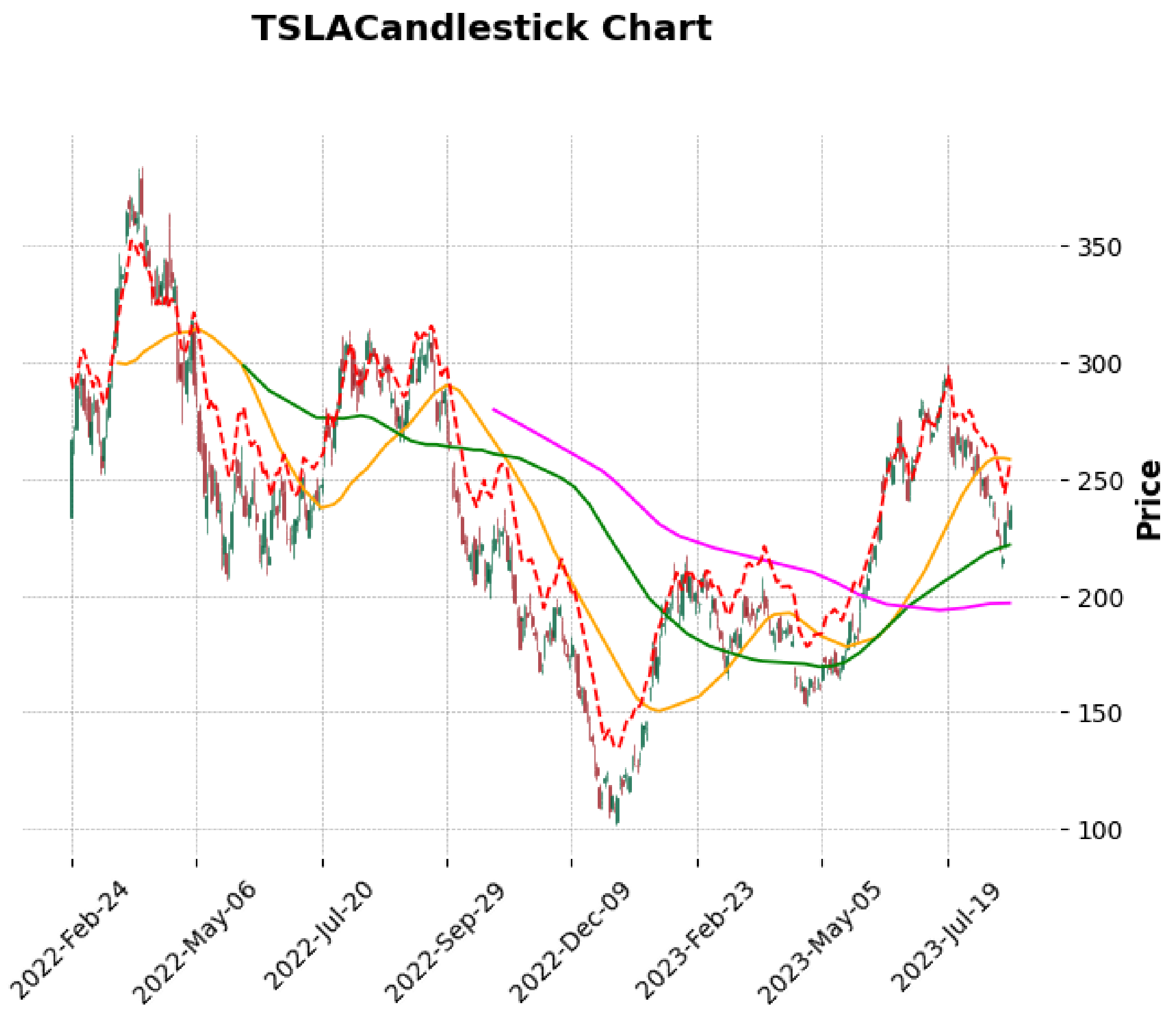
TSLACandlestick Chart
Price
350
300
250
200
150
100
2022-Feb-24
2022-May-06
2022-Jul-20
2022-Sep-29
2022-Dec-09
2023-Feb-23
2023-May-05
2023-Jul-19

Layer Configs 9 - Complex Bidirectional GRU Configuration Model

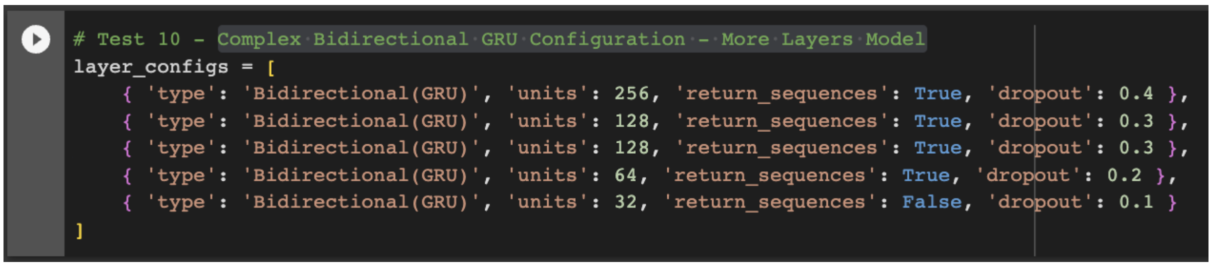


```
# Test 9 -  Complex Bidirectional GRU Configuration Model
layer_configs = [
    { 'type': 'Bidirectional(GRU)', 'units': 128, 'return_sequences': True, 'dropout': 0.3 },
    { 'type': 'Bidirectional(GRU)', 'units': 128, 'return_sequences': True, 'dropout': 0.3 },
    { 'type': 'Bidirectional(GRU)', 'units': 64, 'return_sequences': True, 'dropout': 0.2 },
    { 'type': 'Bidirectional(GRU)', 'units': 64, 'return_sequences': False, 'dropout': 0.2 }
]
```

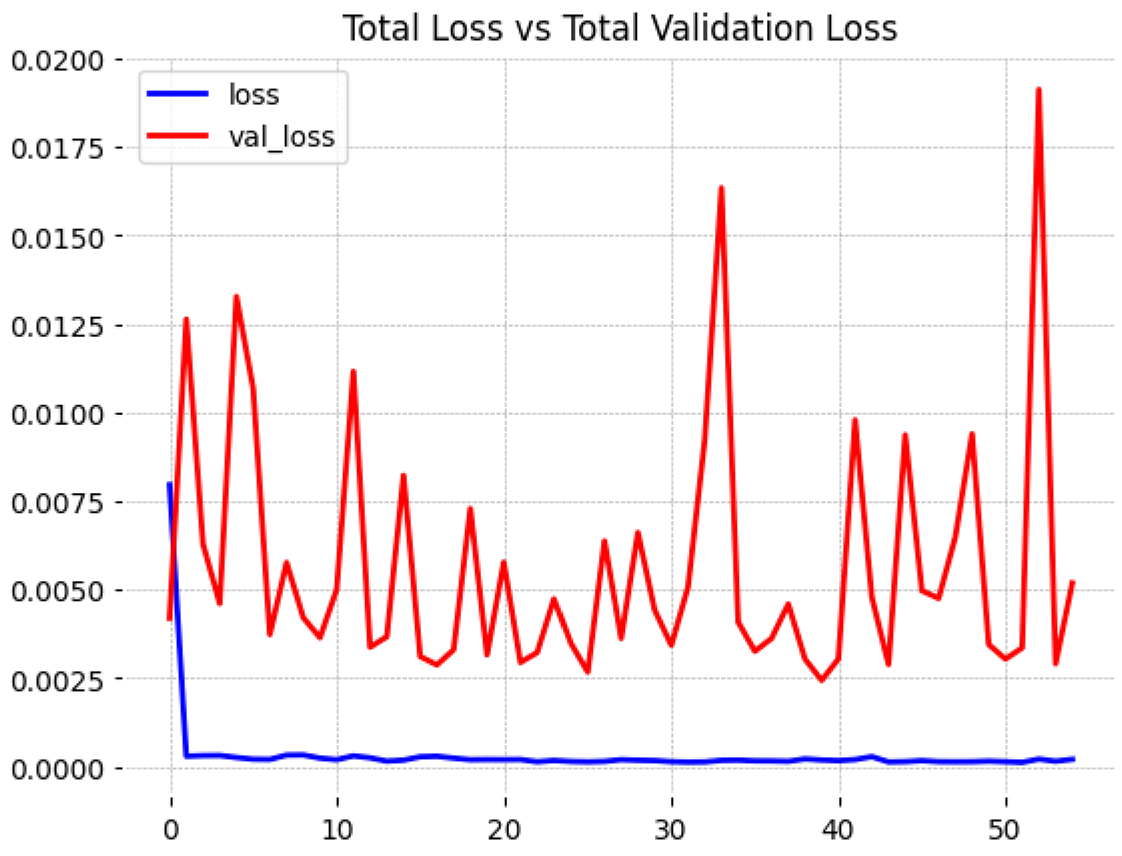


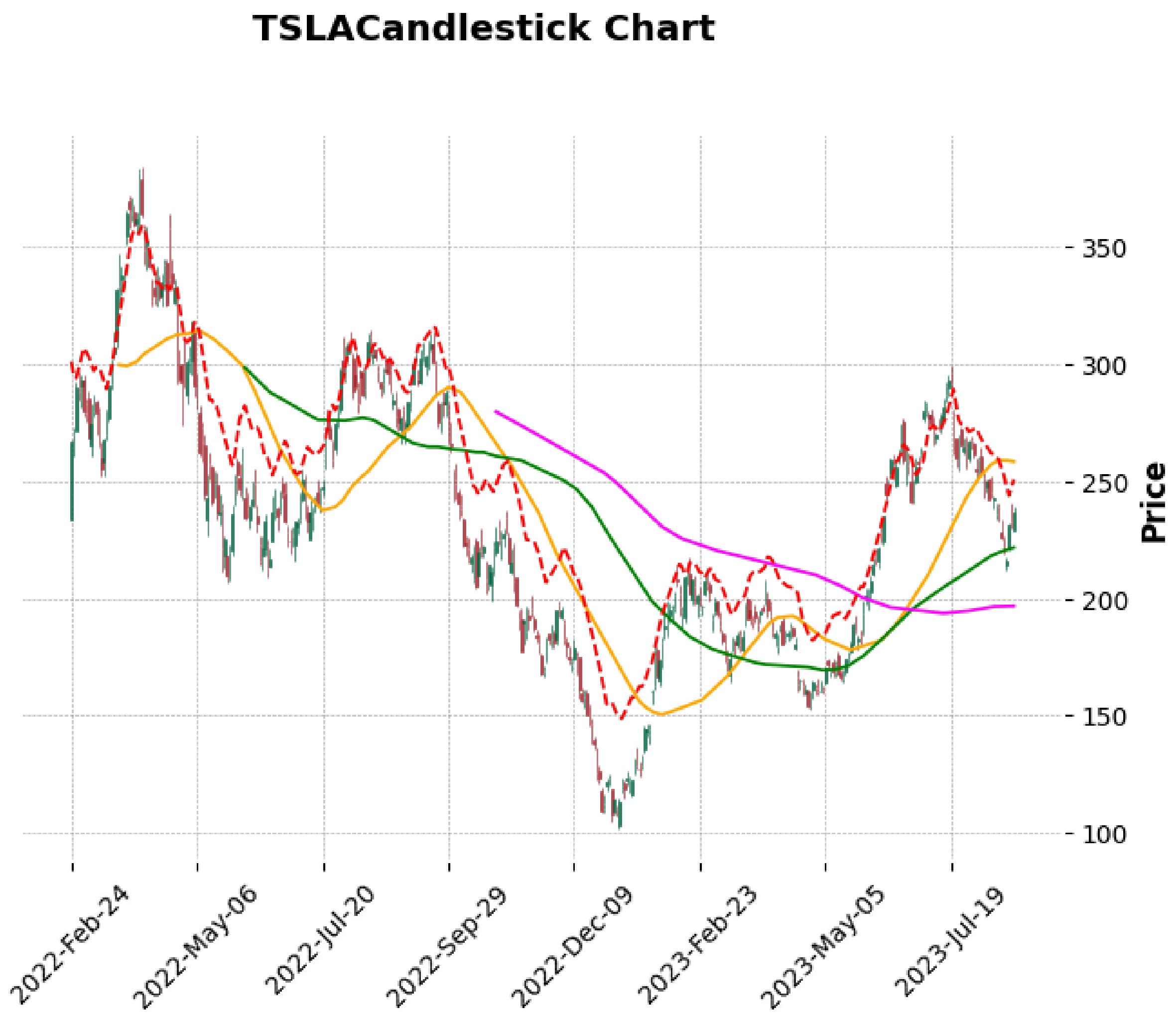

TSLACandlestick Chart
350
300
250
200
150
100
Price
2022-Feb-24
2022-May-06
2022-Jul-20
2022-Sep-29
2022-Dec-09
2023-Feb-23
2023-May-05
2023-Jul-19

Layer Configs 10 - Complex Bidirectional GRU Configuration - More Layers Model



```
# Test 10 – Complex Bidirectional GRU Configuration – More Layers Model
layer_configs = [
    { 'type': 'Bidirectional(GRU)', 'units': 256, 'return_sequences': True, 'dropout': 0.4 },
    { 'type': 'Bidirectional(GRU)', 'units': 128, 'return_sequences': True, 'dropout': 0.3 },
    { 'type': 'Bidirectional(GRU)', 'units': 128, 'return_sequences': True, 'dropout': 0.3 },
    { 'type': 'Bidirectional(GRU)', 'units': 64, 'return_sequences': True, 'dropout': 0.2 },
    { 'type': 'Bidirectional(GRU)', 'units': 32, 'return_sequences': False, 'dropout': 0.1 }
]
```





Conclusion

Based on a series of experiments, it has been observed that the simple LSTM and GRU models yield the highest accuracy. However, as this project progresses, various other models will be explored and tested to potentially enhance the accuracy further.