Week 4 Attempt Summary

This week we were tasked with moving from our v2 code base to the v3 code base

In this Week we take our code base to a more optimal level where we don't construct the deep learning model manually by adding layers of the network, instead we create a function that would take some inputs as 'number_of_layers' , "layer_name (LSTM, RNN and GRU)" and etc..., to create a deep learning model for us.

```
def create_model(n_steps, n_features, loss='huber', units=256,
cell=GRU, n_layers=2, dropout=0.4, optimizer='adam',
bidirectional=False):
    model = Sequential()
    for i in range(n_layers):
        if i == 0:
            # First layer
            if bidirectional:
                model.add(tf.keras.layers.Bidirectional(cell(units,
return_sequences=True), input_shape=(n_steps, n_features)))
            else:
                model.add(cell(units, return_sequences=True,
input_shape=(n_steps, n_features)))
        elif i == n_layers - 1:
            # Last layer
            if bidirectional:
                model.add(tf.keras.layers.Bidirectional(cell(units,
return_sequences=False)))
            else:
                model.add(cell(units, return_sequences=False))
        else:
            # Hidden layers
            if bidirectional:
                model.add(tf.keras.layers.Bidirectional(cell(units,
return_sequences=True)))
            else:
                model.add(cell(units, return_sequences=True))
        model.add(Dropout(dropout))

    model.add(Dense(1, activation='linear'))
    model.compile(loss=loss, optimizer=optimizer)
    print("Model created and compiled.")
    return model
```

The cell name could change from GRU to LSTM to SimpleRNN

And now the algorithm uses a iterative prediction method for more optimization

```
def iterative_predict(model, data, n_steps, future_steps):
    # Retrieve the last 'n_steps' from the test set (ensure the
correct column is used)
    last_sequence = data['test']['Adj Close'].values[-n_steps:]  # Use
the correct column name

    # Reshape to match the model's expected input shape: (1, N_STEPS,
1)
    last_sequence = last_sequence.reshape((1, n_steps, 1))

    predictions = []

    for step in range(future_steps):
        # Predict the next step
        prediction = model.predict(last_sequence)

        # Reshape the prediction to match the shape needed for
appending (1, 1)
        prediction = prediction.reshape((1, 1, 1))

        # If scaling, apply inverse transform using the 'Adj Close'
scaler only
        if SCALE:
            prediction = data['column_scaler']['Adj
Close'].inverse_transform(prediction.reshape(-1, 1)).reshape(1, 1, 1)
```

```
        # Store the predicted price
        predictions.append(prediction[0][0][0])

        # Update the sequence with the new prediction for the next step
        # Remove the first element and append the prediction at the end
        last_sequence = np.append(last_sequence[:, 1:, :], prediction, axis=1)

    return predictions
```