Time Series Forcasting Using Attention-Transformers

Import Libraries

```
import math
import numpy as np
import pandas as pd

import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.layers import Input, Dense, Dropout, LayerNormalization, M

import yfinance as yf
```

```
def calculate_bollinger_bands(data, window=10, num_of_std=2):
    """Calculate Bollinger Bands"""
    rolling_mean = data.rolling(window=window).mean()
    rolling_std = data.rolling(window=window).std()
    upper_band = rolling_mean + (rolling_std * num_of_std)
    lower_band = rolling_mean - (rolling_std * num_of_std)
    return upper_band, lower_band

def calculate_rsi(data, window=10):
    """Calculate Relative Strength Index"""
    delta = data.diff()
    gain = delta.clip(lower=0)
    loss = -delta.clip(upper=0)
    avg_gain = gain.rolling(window=window, min_periods=1).mean()
    avg_loss = loss.rolling(window=window, min_periods=1).mean()
    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))
    return rsi

def calculate_roc(data, periods=10):
    """Calculate Rate of Change."""
    roc = ((data - data.shift(periods)) / data.shift(periods)) * 100
    return roc
```

```python
# List of tickers for the big tech companies, forming the MAMAA group
tickers = ['META', 'AAPL', 'MSFT', 'AMZN', 'GOOG']
```

```python
ticker_data_frames = []
stats = {}
for ticker in tickers:

    # Download historical data for the ticker
    data = yf.download(ticker, period="60d", interval="5m")

    # Calculate the daily percentage change
    close = data['Close']
    upper, lower = calculate_bollinger_bands(close, window=14, num_of_std=2)
    width = upper - lower
    rsi = calculate_rsi(close, window=14)
    roc = calculate_roc(close, periods=14)
    volume = data['Volume']
    diff = data['Close'].diff(1)
    percent_change_close = data['Close'].pct_change() * 100

    # Create a DataFrame for the current ticker and append it to the list
    ticker_df = pd.DataFrame({
        ticker+'_close': close,
        ticker+'_width': width,
        ticker+'_rsi': rsi,
        ticker+'_roc': roc,
        ticker+'_volume': volume,
        ticker+'_diff': diff,
        ticker+'_percent_change_close': percent_change_close,
    })

    MEAN = ticker_df.mean()
    STD = ticker_df.std()

    # Keep track of mean and std
    for column in MEAN.index:
      stats[f"{column}_mean"] = MEAN[column]
      stats[f"{column}_std"] = STD[column]

    # Normalize the training features
    ticker_df = (ticker_df - MEAN) / STD

    ticker_data_frames.append(ticker_df)
```

```python
# Convert the dictionary containing feature statistics to a DataFrame for easier
stats = pd.DataFrame([stats], index=[0])

# Display the DataFrame to verify its structure
stats.head()
```

	META_close_mean	META_close_std	META_width_mean	META_width_std	META_rsi_mean	META_rsi_std	META_roc_mean	META_roc_std	META_volume_mean	META_volume_std	...
0	487.84637	23.192329	5.729313	7.677995	50.634651	16.999532	-0.003481	1.103152	187859.55596	274559.281614	...

1 rows x 70 columns

```python
df = pd.concat(ticker_data_frames, axis=1)
df.replace([np.inf, -np.inf], np.nan, inplace=True)
df.dropna(inplace=True)
df.head()
```

Datetime	META_close	META_width	META_rsi	META_roc	META_volume	META_diff	META_percent_change_close	AAPL_close	AAPL_width	AAPL_rsi
2024-02-08 10:40:00-05:00	-0.879876	-0.453225	0.276652	0.142999	-0.244456	0.603304	0.630348	2.153634	0.665285	-1.206783
2024-02-08 10:45:00-05:00	-0.876427	-0.466511	0.567228	0.255918	-0.071265	0.052056	0.052733	2.186818	0.530445	-1.393217
2024-02-08 10:50:00-05:00	-0.827704	-0.458017	1.183874	0.567584	0.393265	0.709802	0.740270	2.202234	0.427958	-1.059586
2024-02-08 10:55:00-05:00	-0.800971	-0.354933	1.075216	0.485738	0.307327	0.390325	0.405340	2.207914	0.266212	-1.099265
2024-02-08 11:00:00-05:00	-0.791053	-0.282378	1.239436	0.562648	0.095773	0.146033	0.150397	2.214404	0.161153	-0.808833
...	...	...	...	...	...	...	...	...	...	...
2024-05-03 15:35:00-04:00	-1.544859	-0.279655	0.799444	0.439925	0.047033	0.301050	0.324064	1.644018	0.334003	-1.173548
2024-05-03 15:40:00-04:00	-1.534834	-0.309009	0.741985	0.398740	-0.036340	0.147601	0.157819	1.579919	0.362761	-1.592880
2024-05-03 15:45:00-04:00	-1.557224	-0.366948	0.531697	0.309351	0.056132	-0.323338	-0.351159	1.539351	0.400166	-1.686002
2024-05-03 15:50:00-04:00	-1.567819	-0.429556	0.388045	0.235651	0.604869	-0.151976	-0.166176	1.457406	0.629032	-1.778367
2024-05-03 15:55:00-04:00	-1.538283	-0.459272	0.414547	0.252434	1.325256	0.431062	0.464833	1.441178	0.858929	-1.685598

Datetime	...	AMZN_volume	AMZN_diff	AMZN_percent_change_close	GOOG_close	GOOG_width	GOOG_rsi	GOOG_roc	GOOG_volume	GOOG_diff	GOOG_percent_change_close
2024-02-08 10:40:00-05:00	...	0.118234	0.641835	0.663281	-0.325635	-0.219043	-0.056453	-0.045230	-0.227597	0.078263	0.080923
2024-02-08 10:45:00-05:00	...	0.085416	0.171885	0.176998	-0.307664	-0.220821	-0.077832	-0.056189	-0.127030	0.439402	0.456023
2024-02-08 10:50:00-05:00	...	-0.176175	-0.134324	-0.139240	-0.299448	-0.190345	0.229790	0.097318	0.088416	0.194340	0.201217
2024-02-08 10:55:00-05:00	...	0.174715	0.167176	0.172114	-0.304069	-0.169304	0.205666	0.086381	0.074699	-0.128107	-0.133252
2024-02-08 11:00:00-05:00	...	0.075313	0.126162	0.129704	-0.307152	-0.215345	1.240011	0.442177	-0.046486	-0.089454	-0.093197
...	...	...	...	...	...	...	...	...	...	...	...
2024-05-03 15:35:00-04:00	...	-0.056003	0.875928	0.830986	1.889897	-0.379528	0.863357	0.290154	-0.089033	0.658408	0.595442
2024-05-03 15:40:00-04:00	...	-0.095673	0.137479	0.128985	1.898636	-0.360749	0.724066	0.218817	-0.169971	0.207486	0.185985
2024-05-03 15:45:00-04:00	...	-0.106711	-0.292724	-0.279124	1.890935	-0.382651	0.702740	0.213235	0.023716	-0.205491	-0.188070
2024-05-03 15:50:00-04:00	...	0.415664	0.493485	0.466883	1.926361	-0.322035	1.059723	0.417744	0.367006	0.877885	0.793297
2024-05-03 15:55:00-04:00	...	1.782702	-0.070221	-0.068045	1.911472	-0.296289	0.528694	0.187330	0.896853	-0.386041	-0.351019

4659 rows x 35 columns

```python
# Shift the dataframe up by one to align current features with the next step's o
labels = df.shift(-1)

# Remove the last row from both the features and labels to maintain consistent d
df = df.iloc[:-1]
labels = labels.iloc[:-1]
```

```python
SEQUENCE_LEN = 24  # 2 hours of data at 5-minute intervals

def create_sequences(data, labels, mean, std, sequence_length=SEQUENCE_LEN):
    sequences = []
    lab = []
    data_size = len(data)

    # Loop to create each sequence and its corresponding label
    for i in range(data_size - (sequence_length + 13)): # Ensure we have data fo
        if i == 0:
          continue
        sequences.append(data[i:i + sequence_length])  # The sequence of data
        lab.append([labels[i-1], labels[i + 12], mean[0], std[0]]) # The label a

    return np.array(sequences), np.array(lab)
```

```python
sequences_dict = {}
sequence_labels = {}
for ticker in tickers:

    # Extract close and volume data for the ticker
    close = df[ticker+'_close'].values
    width = df[ticker+'_width'].values
    rsi = df[ticker+'_rsi'].values
    roc = df[ticker+'_roc'].values
    volume = df[ticker+'_volume'].values
    diff = df[ticker+'_diff'].values
    pct_change = df[ticker+'_percent_change_close'].values

    # Combine close and volume data
    ticker_data = np.column_stack((close,
                                   width,
                                   rsi,
                                   roc,
                                   volume,
                                   diff,
                                   pct_change))

    # Generate sequences
    attribute = ticker+"_close"
    ticker_sequences, lab = create_sequences(ticker_data,
                                             labels[attribute].values[SEQUENCE_L
                                             stats[attribute+"_mean"].values,
                                             stats[attribute+"_std"].values)

    sequences_dict[ticker] = ticker_sequences
    sequence_labels[ticker] = lab
```

```python
# Combine data and labels from all tickers
all_sequences = []
all_labels = []

for ticker in tickers:
    all_sequences.extend(sequences_dict[ticker])
    all_labels.extend(sequence_labels[ticker])

# Convert to numpy arrays
all_sequences = np.array(all_sequences)
all_labels = np.array(all_labels)
```

```python
np.random.seed(42)
shuffled_indices = np.random.permutation(len(all_sequences))
all_sequences = all_sequences[shuffled_indices]
all_labels = all_labels[shuffled_indices]

train_size = int(len(all_sequences) * 0.9)

# Split sequences
train_sequences = all_sequences[:train_size]
train_labels    = all_labels[:train_size]

other_sequences = all_sequences[train_size:]
other_labels    = all_labels[train_size:]

shuffled_indices = np.random.permutation(len(other_sequences))
other_sequences = other_sequences[shuffled_indices]
other_labels = other_labels[shuffled_indices]

val_size = int(len(other_sequences) * 0.5)

validation_sequences = other_sequences[:val_size]
validation_labels = other_labels[:val_size]

test_sequences = other_sequences[val_size:]
test_labels = other_labels[val_size:]
```

```python
def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout=0):
    # Attention and Normalization
    x = LayerNormalization(epsilon=1e-6)(inputs)
    x = MultiHeadAttention(key_dim=head_size, num_heads=num_heads, dropout=dropo
    x = Add()([x, inputs])

    # Feed Forward Part
    y = LayerNormalization(epsilon=1e-6)(x)
    y = Dense(ff_dim, activation="relu")(y)
    y = Dropout(dropout)(y)
    y = Dense(inputs.shape[-1])(y)
    return Add()([y, x])
```

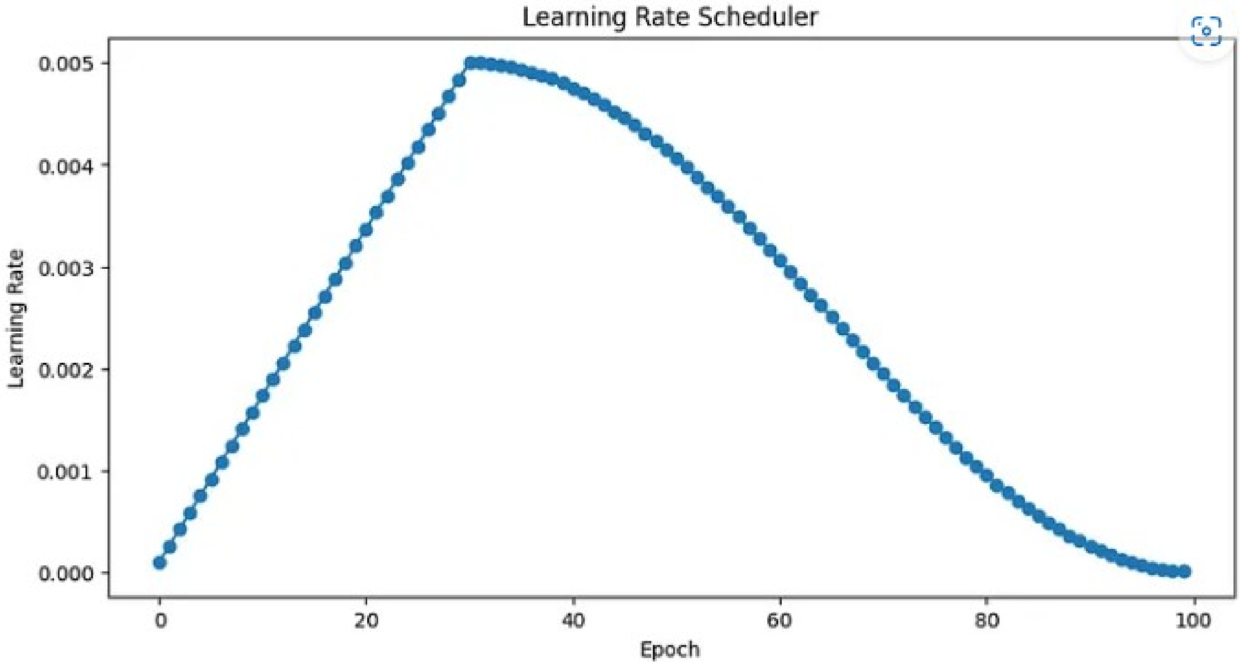
```python
def build_transformer_model(input_shape, head_size, num_heads, ff_dim, num_laye
    inputs = Input(shape=input_shape)
    x = inputs
    for _ in range(num_layers):
        x = transformer_encoder(x, head_size, num_heads, ff_dim, dropout)
    x = GlobalAveragePooling1D()(x)
    x = LayerNormalization(epsilon=1e-6)(x)
    outputs = Dense(1, activation="linear")(x)
    return Model(inputs=inputs, outputs=outputs)
```

```python
input_shape = train_sequences.shape[1:]
head_size = 256
num_heads = 16
ff_dim = 1024
num_layers = 12
dropout = 0.20

model = build_transformer_model(input_shape, head_size, num_heads, ff_dim, num_l
model.summary()
```

```
Total params: 1,708,558 (6.52 MB)
Trainable params: 1,708,558 (6.52 MB)
Non-trainable params: 0 (0.00 B)
```

```
def custom_mae_loss(y_true, y_pred):
    y_true_next = tf.cast(y_true[:, 1], tf.float64)  # Extract the true next val
    y_pred_next = tf.cast(y_pred[:, 0], tf.float64)  # Extract the predicted nex
    abs_error = tf.abs(y_true_next - y_pred_next)  # Calculate the absolute erro
    return tf.reduce_mean(abs_error)  # Return the mean of these errors

def dir_acc(y_true, y_pred):
    mean, std = tf.cast(y_true[:, 2], tf.float64), tf.cast(y_true[:, 3], tf.floa
    y_true_prev = (tf.cast(y_true[:, 0], tf.float64) * std) + mean  # Un-scale p
    y_true_next = (tf.cast(y_true[:, 1], tf.float64) * std) + mean  # Un-scale n
    y_pred_next = (tf.cast(y_pred[:, 0], tf.float64) * std) + mean  # Un-scale p

    true_change = y_true_next - y_true_prev  # Calculate true change
    pred_change = y_pred_next - y_true_prev  # Calculate predicted change

    correct_direction = tf.equal(tf.sign(true_change), tf.sign(pred_change))  #
    return tf.reduce_mean(tf.cast(correct_direction, tf.float64))  # Return the
```

```python
# Define a callback to save the best model
checkpoint_callback_train = ModelCheckpoint(
    "transformer_train_model.keras",  # Filepath to save the best model
    monitor="dir_acc",  #"loss",  # Metric to monitor
    save_best_only=True,  # Save only the best model
    mode="max",  # Minimize the monitored metric
    verbose=1,  # Display progress
)

# Define a callback to save the best model
checkpoint_callback_val = ModelCheckpoint(
    "transformer_val_model.keras",  # Filepath to save the best model
    monitor="val_dir_acc", #"val_loss",  # Metric to monitor
    save_best_only=True,  # Save only the best model
    mode="max",  # Minimize the monitored metric
    verbose=1,  # Display progress
)

def get_lr_callback(batch_size=16, mode='cos', epochs=500, plot=False):
    lr_start, lr_max, lr_min = 0.0001, 0.005, 0.00001  # Adjust learning rate bo
    lr_ramp_ep = int(0.30 * epochs)  # 30% of epochs for warm-up
    lr_sus_ep = max(0, int(0.10 * epochs) - lr_ramp_ep)  # Optional sustain phas

    def lrfn(epoch):
        if epoch < lr_ramp_ep:  # Warm-up phase
            lr = (lr_max - lr_start) / lr_ramp_ep * epoch + lr_start
        elif epoch < lr_ramp_ep + lr_sus_ep:  # Sustain phase at max learning ra
            lr = lr_max
        elif mode == 'cos':
            decay_total_epochs, decay_epoch_index = epochs - lr_ramp_ep - lr_sus
            phase = math.pi * decay_epoch_index / decay_total_epochs
            lr = (lr_max - lr_min) * 0.5 * (1 + math.cos(phase)) + lr_min
        else:
            lr = lr_min  # Default to minimum learning rate if mode is not recog

        return lr

    if plot:  # Plot learning rate curve if plot is True
        plt.figure(figsize=(10, 5))
        plt.plot(np.arange(epochs), [lrfn(epoch) for epoch in np.arange(epochs)]
        plt.xlabel('Epoch')
        plt.ylabel('Learning Rate')
        plt.title('Learning Rate Scheduler')
        plt.show()

    return tf.keras.callbacks.LearningRateScheduler(lrfn, verbose=True)
```




```
BATCH_SIZE = 64  # Number of training examples used to calculate each iteration'
EPOCHS = 100  # Total number of times the entire dataset is passed through the n

model.fit(
    train_sequences,  # Training features
    train_labels,  # Training labels
    validation_data=(validation_sequences, validation_labels),  # Validation dat
    epochs=EPOCHS,  # Number of epochs to train for
    batch_size=BATCH_SIZE,  # Size of each batch
    shuffle=True,  # Shuffle training data before each epoch
    callbacks=[checkpoint_callback_train, checkpoint_callback_val, get_lr_callba
)
```

```
Epoch 1: LearningRateScheduler setting learning rate to 0.0001.
Epoch 1/100
325/325 [==============================] - ETA: 0s - loss: 0.2581 - dir_acc: 0.5365
Epoch 1: dir_acc improved from 0.53607 to 0.53647, saving model to transformer_train_model.keras

Epoch 1: val_dir_acc improved from 0.51855 to 0.52825, saving model to transformer_val_model.keras
325/325 [==============================] - 50s 154ms/step - loss: 0.2581 - dir_acc: 0.5365 - val_loss: 0.2660 - val_dir_acc: 0.5283 - lr: 1.0000e-04

Epoch 2: LearningRateScheduler setting learning rate to 0.00026333333333333336.
Epoch 2/100
325/325 [==============================] - ETA: 0s - loss: 0.2468 - dir_acc: 0.5404
Epoch 2: dir_acc improved from 0.53647 to 0.54041, saving model to transformer_train_model.keras

Epoch 2: val_dir_acc did not improve from 0.52825
325/325 [==============================] - 50s 153ms/step - loss: 0.2468 - dir_acc: 0.5404 - val_loss: 0.2536 - val_dir_acc: 0.5146 - lr: 2.6333e-04

Epoch 3: LearningRateScheduler setting learning rate to 0.00042666666666666667.
Epoch 3/100
325/325 [==============================] - ETA: 0s - loss: 0.2336 - dir_acc: 0.5329
Epoch 3: dir_acc did not improve from 0.54041

Epoch 3: val_dir_acc did not improve from 0.52825
325/325 [==============================] - 49s 151ms/step - loss: 0.2336 - dir_acc: 0.5329 - val_loss: 0.2393 - val_dir_acc: 0.5237 - lr: 4.2667e-04

Epoch 4: LearningRateScheduler setting learning rate to 0.00059.
Epoch 4/100
325/325 [==============================] - ETA: 0s - loss: 0.2266 - dir_acc: 0.5379
Epoch 4: dir_acc did not improve from 0.54041

Epoch 4: val_dir_acc improved from 0.52825 to 0.53482, saving model to transformer_val_model.keras
```

```python
model.load_weights("transformer_val_model.keras")  # Load the best model from th
accuracy = model.evaluate(test_sequences, test_labels)[1]  # Evaluate the model
print(accuracy)

from sklearn.metrics import r2_score

predictions = model.predict(test_sequences)  # Make predictions on the test data
r2 = r2_score(test_labels[:, 1], predictions[:, 0])  # Calculate R-squared value
print(f"R-squared: {r2}")
```

Fi

```
37/37 [==============================] - 1s 39ms/step - loss: 0.1119 - dir_acc: 0.7044
0.7043918967247009
37/37 [==============================] - 1s 33ms/step
R-squared: 0.9716192227979671
```

Final R-squared Loss : 0.97