Group Work Project 2 of Deep Learning

| Student Group | 6888 | |:|| |Team member A |Ebenezer Yeboah| |Team member B |Jatin Rai|

```
pip install pyts
```

```
Requirement already satisfied: pyts in /usr/local/lib/python3.10/dist-packages (0.13.0)
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-packages (from pyts) (1.26.4)
Requirement already satisfied: scipy>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from pyts) (1.13.1)
Requirement already satisfied: scikit-learn>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from pyts) (1.3.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from pyts) (1.4.2)
Requirement already satisfied: numba>=0.55.2 in /usr/local/lib/python3.10/dist-packages (from pyts) (0.60.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba>=0.55.2->pyts) (0.43.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.2.0->pyts) (3.5.0)
```

```python
# load libraries

import math
from scipy.optimize import brute, fmin
from scipy.integrate import quad
import yfinance as yf
import pandas_datareader as pdr  # Access FRED
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
import pandas as pd
import numpy as np
import seaborn as sns
from statsmodels.tsa.stattools import adfuller
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from pyts.image import GramianAngularField
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

```
from tensorflow.keras.layers import Dense, Flatten, LSTM
from tensorflow.keras.optimizers import Adam

plt.rcParams['figure.figsize'] = [14, 6]
```

Step 1

```
import yfinance as yf

symbols = ["SPY", "TLT", "GLD"]

start_date = "2018-01-01"
end_date = "2022-12-30"

data = yf.download(symbols, start=start_date, end=end_date)
data.head()
```

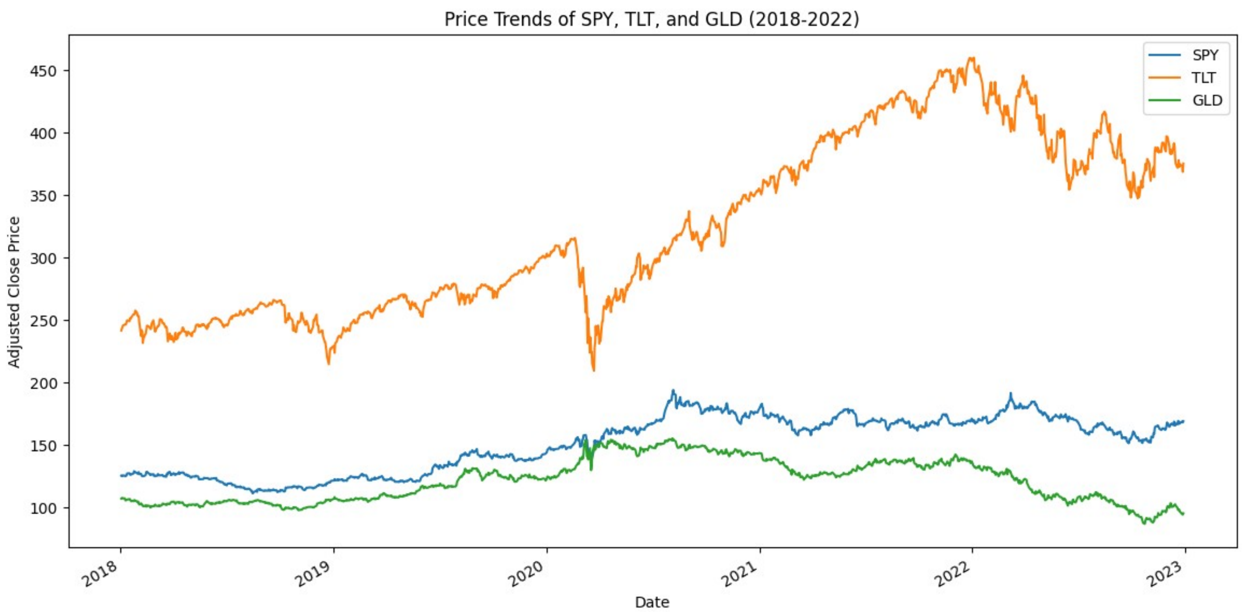
```
[*********************100%***********************]  3 of 3 completed
```


{"summary":"{\n \"name\": \"data\",\n \"rows\": 1258,\n \"fields\":
[\n   {\n   \"column\": [\n   \"Date\",\n   \"\"\
n   ],\n   \"properties\": {\n   \"dtype\": \"date\",\n
\"min\": \"2018-01-02 00:00:00+00:00\",\n   \"max\": \"2022-12-29
00:00:00+00:00\",\n   \"num_unique_values\": 1258,\n
\"samples\": [\n   \"2020-03-26 00:00:00+00:00\",\n
\"2018-05-29 00:00:00+00:00\",\n   \"2018-03-16
00:00:00+00:00\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\": [\n
\"Adj Close\",\n   \"GLD\"\n   ],\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 22.501464620320085,\n
\"min\": 111.0999984741211,\n   \"max\": 193.88999938964844,\n
\"num_unique_values\": 1117,\n   \"samples\": [\n
171.82000732421875,\n   123.37999725341797,\n
157.5500030517578\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\": [\n
\"Adj Close\",\n   \"SPY\"\n   ],\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 69.69494658810892,\n
\"min\": 209.25750732421875,\n   \"max\": 460.1273498535156,\n
\"num_unique_values\": 1239,\n   \"samples\": [\n
275.6157531738281,\n   240.7160186767578,\n
263.6673278808594\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\": [\n
\"Adj Close\",\n   \"TLT\"\n   ],\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 17.24694848906187,\n
\"min\": 86.63060760498047,\n   \"max\": 155.07325744628906,\n
\"num_unique_values\": 1229,\n   \"samples\": [\n
151.55934143066406,\n   111.39590454101562,\n
106.50511169433594\n   ],\n   \"semantic_type\": \"\",\n

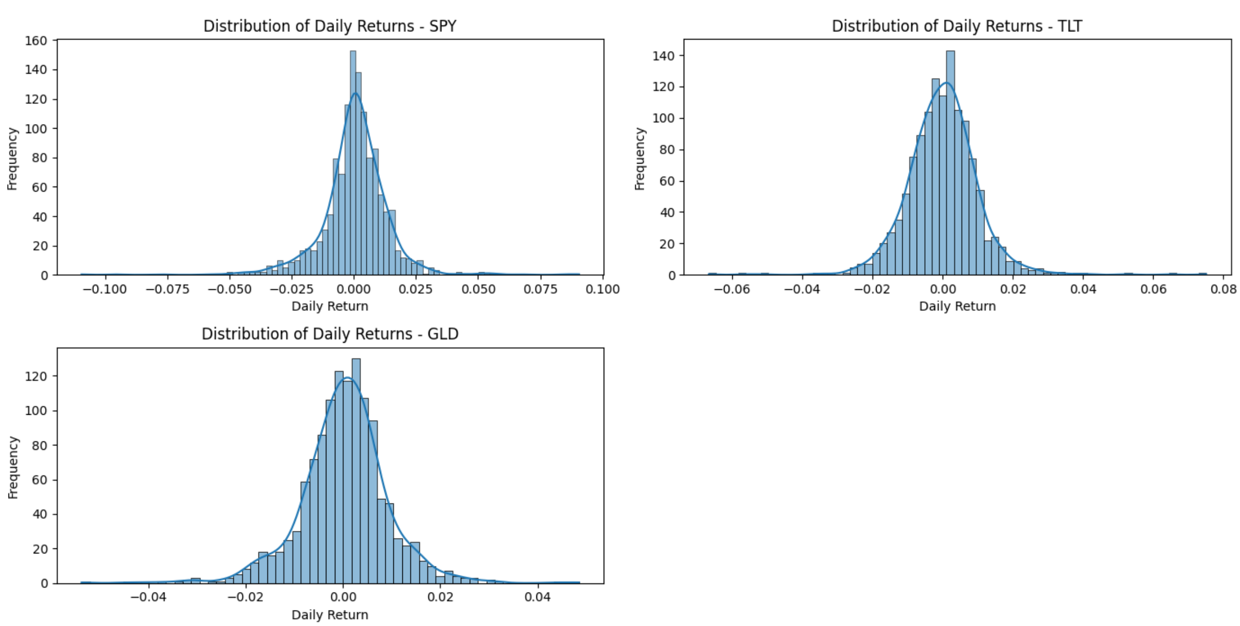
```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": [\n
\"Close\",\n        \"GLD\"\n      ],\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 22.501464620320085,\n
\"min\": 111.0999984741211,\n        \"max\": 193.88999938964844,\n
\"num_unique_values\": 1117,\n        \"samples\": [\n
171.82000732421875,\n          123.37999725341797,\n
157.5500030517578\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": [\n
\"Close\",\n        \"SPY\"\n      ],\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 66.61349684811863,\n
\"min\": 222.9499969482422,\n        \"max\": 477.7099914550781,\n
\"num_unique_values\": 1216,\n        \"samples\": [\n
261.6499938964844,\n          263.4100036621094,\n
272.8800048828125\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": [\n
\"Close\",\n        \"TLT\"\n      ],\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 18.28434012883009,\n
\"min\": 92.4000015258789,\n        \"max\": 171.57000732421875,\n
\"num_unique_values\": 1105,\n        \"samples\": [\n
131.4600067138672,\n          104.33000183105469,\n
144.9199981689453\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": [\n
\"High\",\n        \"GLD\"\n      ],\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 22.673477839486985,\n
\"min\": 111.87999725341797,\n        \"max\": 194.4499969482422,\n
\"num_unique_values\": 1115,\n        \"samples\": [\n
121.7300033569336,\n          123.33999633789062,\n
160.57000732421875\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": [\n
\"High\",\n        \"SPY\"\n      ],\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 66.89923943510234,\n
\"min\": 229.67999267578125,\n        \"max\": 479.9800109863281,\n
\"num_unique_values\": 1211,\n        \"samples\": [\n
273.94000244140625,\n          406.94000244140625,\n
448.4100036621094\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": [\n
\"High\",\n        \"TLT\"\n      ],\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 18.424015143797895,\n
\"min\": 93.55000305175781,\n        \"max\": 179.6999969482422,\n
\"num_unique_values\": 1090,\n        \"samples\": [\n
144.00999450683594,\n          121.08000183105469,\n
104.4000015258789\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": [\n
\"Low\",\n        \"GLD\"\n      ],\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 22.288603054854892,\n
\"min\": 111.05999755859375,\n        \"max\": 192.52000427246094,\n
\"num_unique_values\": 1134,\n        \"samples\": [\n
167.9199981689453,\n          170.05999755859375,\n
141.74000549316406\n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": [\n
\"Low\",\n        \"SPY\"\n      ],\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 66.24547543452118,\n
\"min\": 218.25999450683594,\n        \"max\": 476.05999755859375,\n
\"num_unique_values\": 1202,\n        \"samples\": [\n
382.17999267578125,\n          296.9700012207031,\n          265.5\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": [\n        \"Low\",\n
\"TLT\"\n      ],\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 18.17066462031259,\n        \"min\":
91.8499984741211,\n        \"max\": 170.77999877929688,\n
\"num_unique_values\": 1119,\n        \"samples\": [\n
121.75,\n          120.30000305175781,\n          121.76000213623047\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": [\n        \"Open\",\n
\"GLD\"\n      ],\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 22.509357815750334,\n        \"min\":
111.45999908447266,\n        \"max\": 193.74000549316406,\n
\"num_unique_values\": 1117,\n        \"samples\": [\n
173.47000122070312,\n          121.3499984741211,\n
152.7100067138672\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": [\n
\"Open\",\n        \"SPY\"\n      ],\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 66.61230615624916,\n
\"min\": 228.19000244140625,\n        \"max\": 479.2200012207031,\n
\"num_unique_values\": 1217,\n        \"samples\": [\n
255.6999969482422,\n          265.1000061035156,\n
273.29998779296875\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": [\n
\"Open\",\n        \"TLT\"\n      ],\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 18.31035238755479,\n
\"min\": 92.81999969482422,\n        \"max\": 179.10000610351562,\n
\"num_unique_values\": 1107,\n        \"samples\": [\n
163.61000061035156,\n          119.0199966430664,\n
119.83000183105469\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": [\n
\"Volume\",\n        \"GLD\"\n      ],\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 5020324,\n        \"min\":
1436500,\n        \"max\": 47347700,\n        \"num_unique_values\":
1254,\n        \"samples\": [\n          3345900,\n          9013200,\
n          7402200\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": [\n
\"Volume\",\n        \"SPY\"\n      ],\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 45854250,\n        \"min\":
20270000,\n        \"max\": 392220700,\n        \"num_unique_values\":
1258,\n        \"samples\": [\n          257632800,\n
115908600,\n          100343700\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": [\n        \"Volume\",\n
```

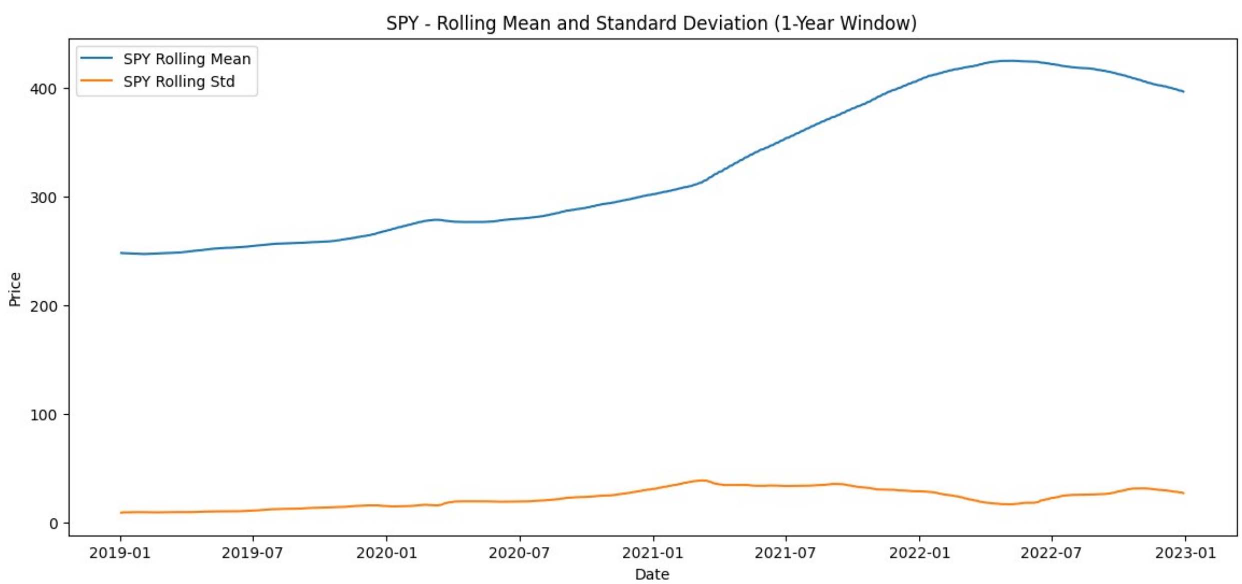
```
\"TLT\"\n      ],\n      \"properties\": {\n        \"dtype\": 
\"number\",\n        \"std\": 7620834,\n        \"min\": 3027900,\n  
\"max\": 76288300,\n        \"num_unique_values\": 1257,\n        
\"samples\": [\n          14147100,\n          21668100,\n          
6084300\n        ],\n        \"semantic_type\": \"\",\n        
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"data"}

returns = data['Adj Close'].pct_change().dropna()

summary_stats = returns.describe()
summary_stats

{"summary":"{\n  \"name\": \"summary_stats\",\n  \"rows\": 8,\n  
\"fields\": [\n    {\n      \"column\": \"GLD\",\n      
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\": 
444.416337746944,\n        \"min\": -0.05369440853432972,\n        
\"max\": 1257.0,\n        \"num_unique_values\": 8,\n        
\"samples\": [\n          0.00027924450495204 27,\n          
0.0005183138095208317,\n          1257.0\n        ],\n        
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"SPY\",\n      \"properties\": {\n        
\"dtype\": \"number\",\n        \"std\": 444.4167184569945,\n        
\"min\": -0.10942372505104336,\n        \"max\": 1257.0,\n        
\"num_unique_values\": 8,\n        \"samples\": [\n          
0.0004433377126648464,\n          0.0007719623453552593,\n          
1257.0\n        ],\n        \"semantic_type\": \"\",\n        
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": 
\"TLT\",\n      \"properties\": {\n        \"dtype\": \"number\",\n        
\"std\": 444.4156747487213,\n        \"min\": -0.06668292549449151,\n        
\"max\": 1257.0,\n        \"num_unique_values\": 8,\n        
\"samples\": [\n          -4.1373803645611245e-05,\n          
7.220520164308297e-05,\n          1257.0\n        ],\n        
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"summary_stats"}

data['Adj Close'].plot(figsize=(14, 7))
plt.title('Price Trends of SPY, TLT, and GLD (2018-2022)')
plt.xlabel('Date')
plt.ylabel('Adjusted Close Price')
plt.legend(symbols)
plt.show()
```

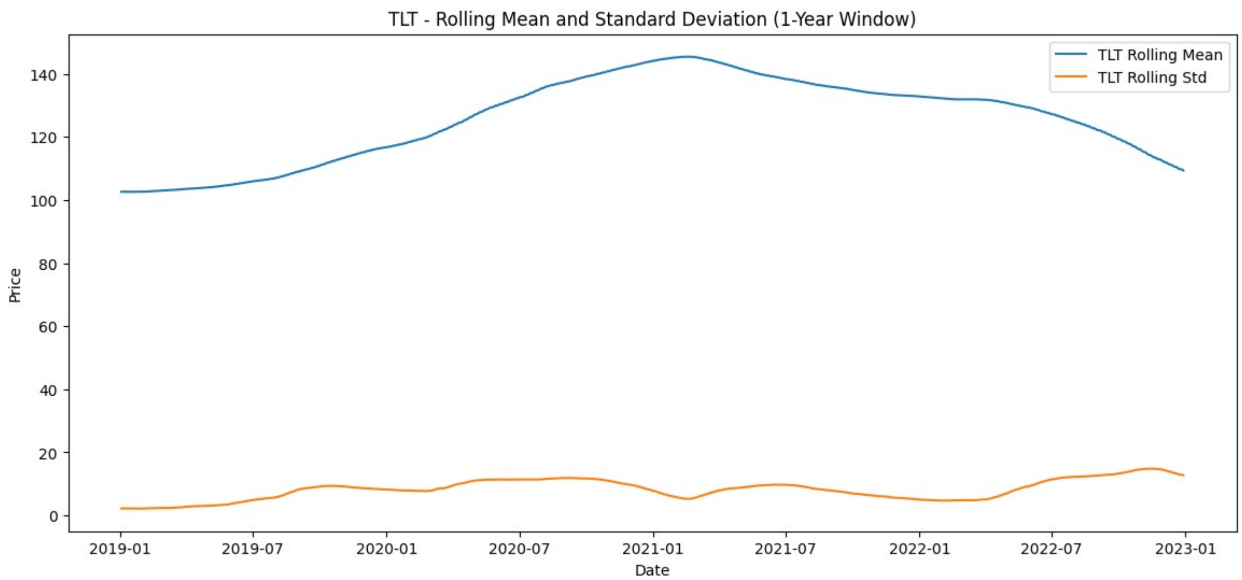


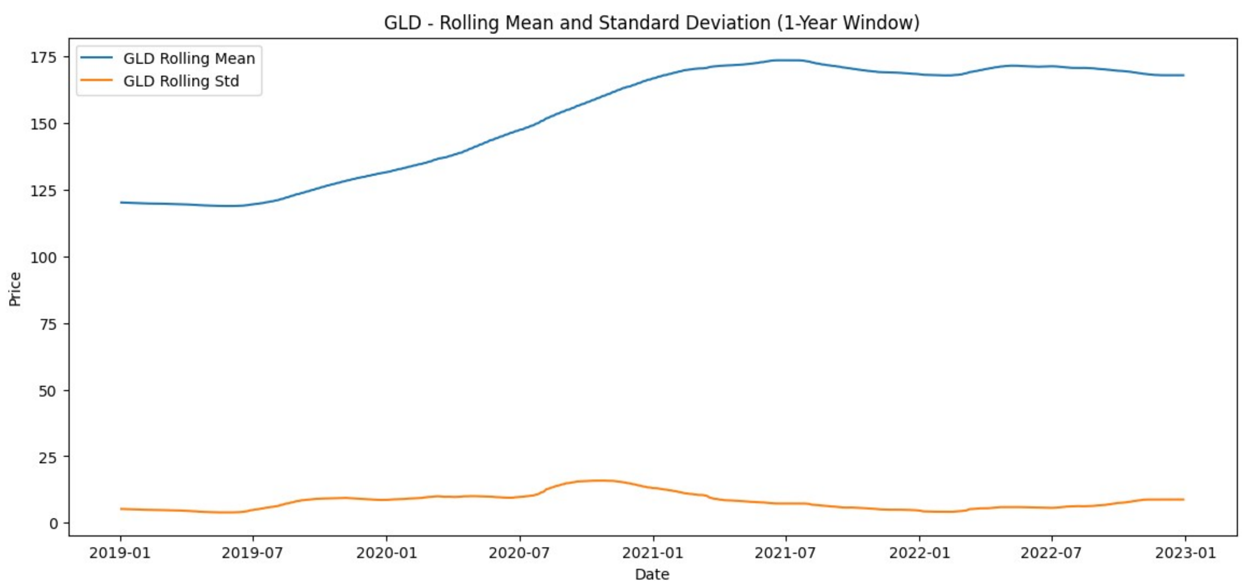

```python
plt.figure(figsize=(14, 7))
for i, symbol in enumerate(symbols):
    plt.subplot(2, 2, i+1)
    sns.histplot(returns[symbol], kde=True)
    plt.title(f'Distribution of Daily Returns - {symbol}')
    plt.xlabel('Daily Return')
    plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```



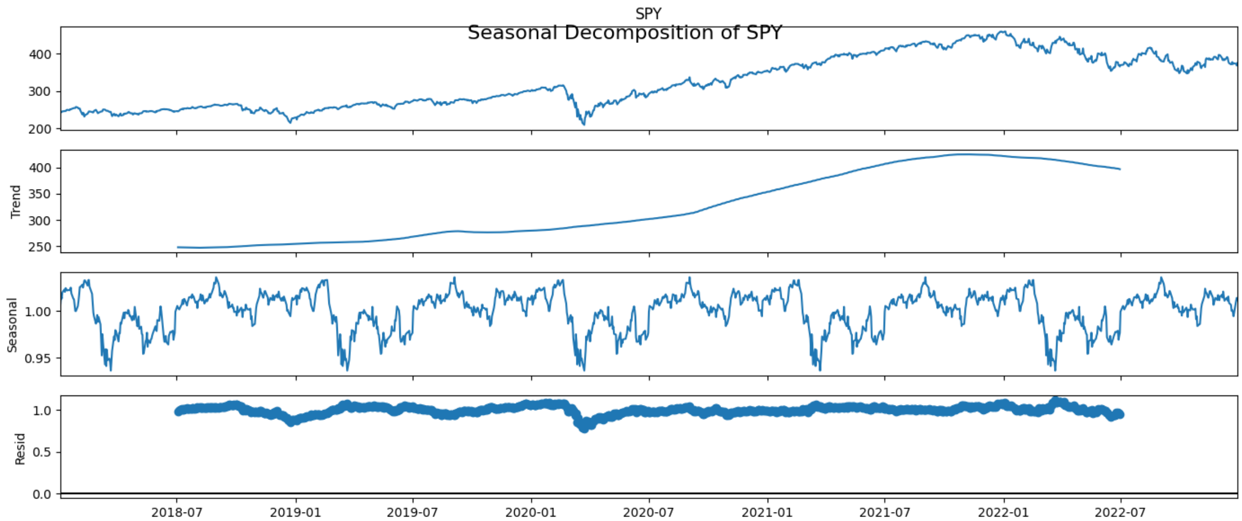
```
window_size = 252  #days in the year
for symbol in symbols:
    rolling_mean = data['Adj Close']
[symbol].rolling(window=window_size).mean()
    rolling_std = data['Adj Close']
[symbol].rolling(window=window_size).std()

    plt.plot(data['Adj Close'].index, rolling_mean, label=f'{symbol}
Rolling Mean')
    plt.plot(data['Adj Close'].index, rolling_std, label=f'{symbol}
Rolling Std')
    plt.title(f'{symbol} - Rolling Mean and Standard Deviation (1-Year
Window)')
    plt.xlabel('Date')
    plt.ylabel('Price')
    plt.legend()
    plt.show()
```

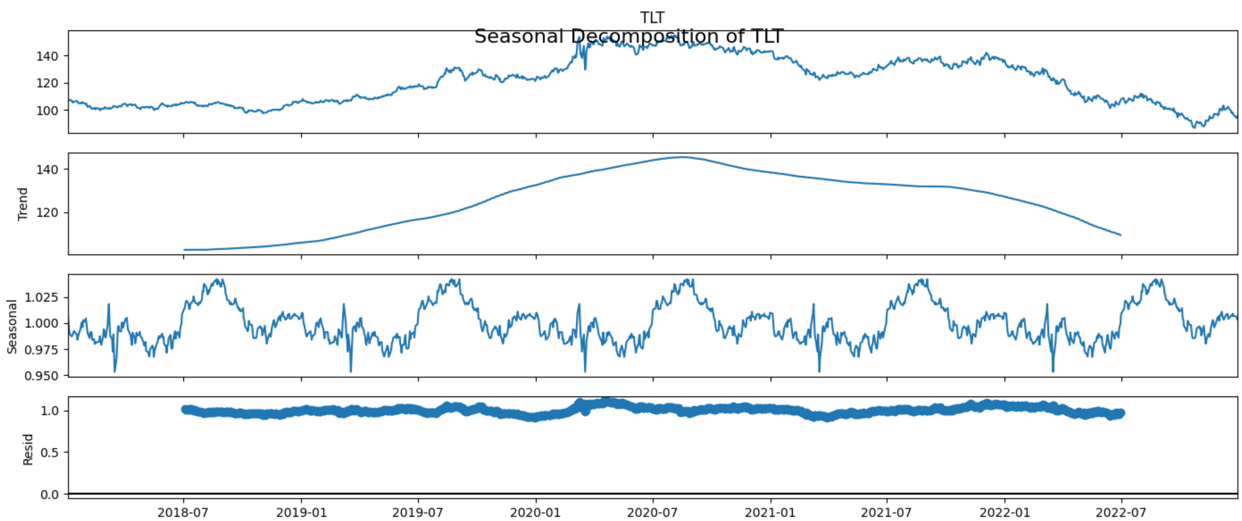


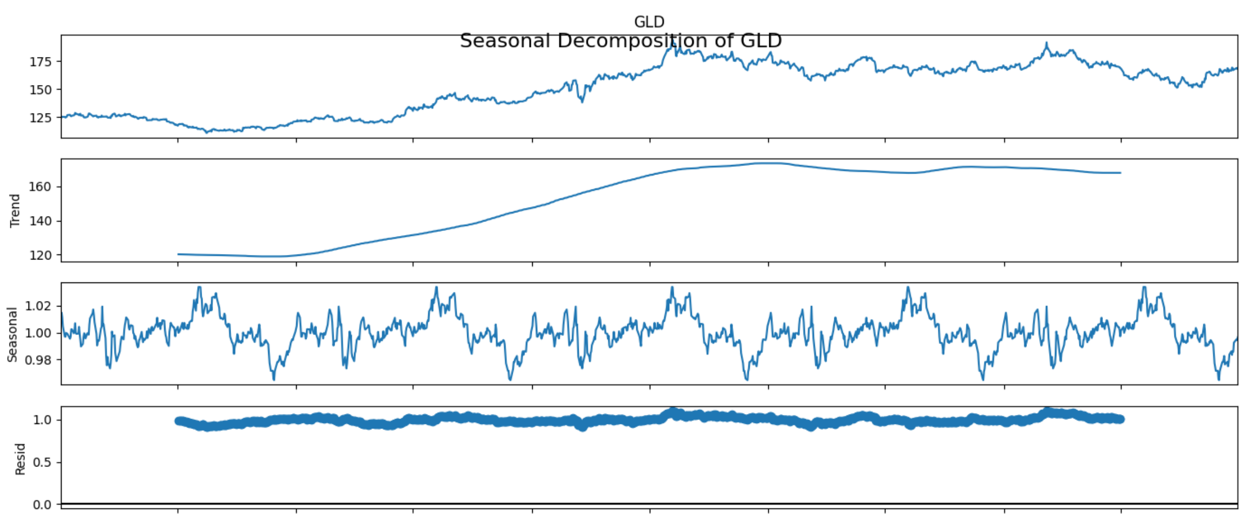





```
from statsmodels.tsa.seasonal import seasonal_decompose

# Seasonal decomposition for each ETF
for symbol in symbols:
    decomposed = seasonal_decompose(data['Adj Close'][symbol],
model='multiplicative', period=252)
    decomposed.plot()
    plt.suptitle(f'\n Seasonal Decomposition of {symbol}',
fontsize=16)
    plt.show()
```


SPY
Seasonal Decomposition of SPY
400
300
200
Trend
400
350
300
250
Seasonal
1.00
0.95
Resid
1.0
0.5
0.0
2018-07
2019-01
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07

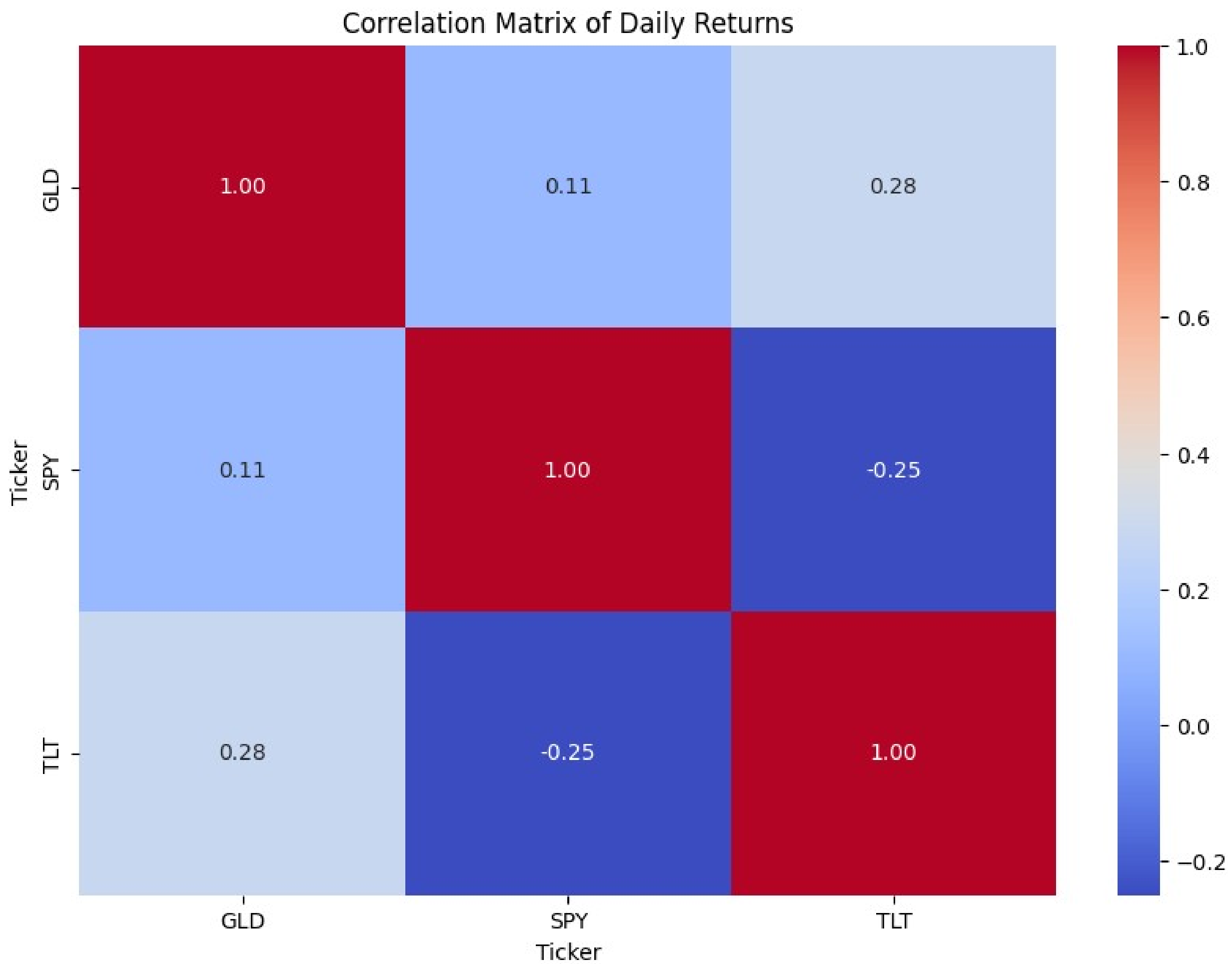

TLT
Seasonal Decomposition of TLT
140
120
100
Trend
140
120
Seasonal
1.025
1.000
0.975
0.950
Resid
1.0
0.5
0.0
2018-07
2019-01
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07


GLD
Seasonal Decomposition of GLD
175
150
125
Trend
160
140
120
Seasonal
1.02
1.00
0.98
Resid
1.0
0.5
0.0
2018-07
2019-01
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07

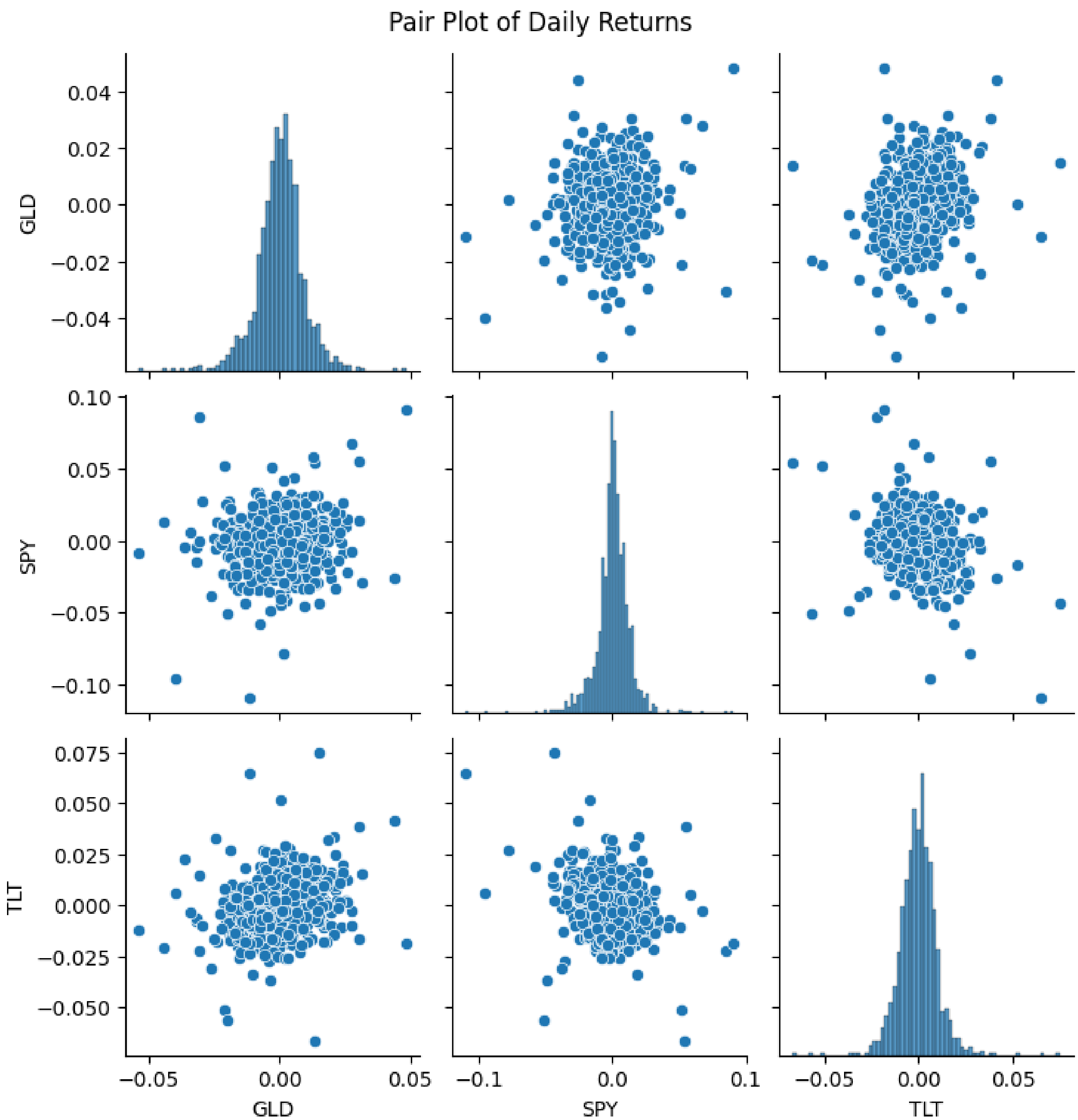
```
#Correlation matrix of daily returns
correlation_matrix = returns.corr()
correlation_matrix
```


{"summary":"{\n \"name\": \"correlation_matrix\",\n \"rows\": 3,\n \"fields\": [\n   {\n   \"column\": \"Ticker\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 3,\n   \"samples\": [\n   \"GLD\",\n   \"SPY\",\n   \"TLT\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"GLD\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.47410863813640347,\n   \"min\": 0.10561439673864652,\n   \"max\": 1.0,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   1.0,\n   0.10561439673864652,\n   0.2800567688998466\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"SPY\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.6439406974328451,\n   \"min\": -0.2497105023326711,\n   \"max\": 1.0,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   0.10561439673864652,\n   1.0,\n   -0.2497105023326711\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"TLT\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.6272622999081876,\n   \"min\": -0.2497105023326711,\n   \"max\": 1.0,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   0.2800567688998466,\n   -0.2497105023326711,\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"correlation_matrix"}


```
# Plot heatmap of the correlation matrix
plt.figure(figsize=(10, 7))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt='.2f')
plt.title('Correlation Matrix of Daily Returns')
plt.show()
```




```
sns.pairplot(returns)
plt.suptitle('Pair Plot of Daily Returns', y=1.02)
plt.show()
```



Step 2

```
lags = 25

#LSTM Model for Equity(SPY)
model_spy = Sequential()
model_spy.add(LSTM(units=64, return_sequences=True, input_shape=(lags,
1)))
model_spy.add(Dropout(0.2))
model_spy.add(LSTM(units=64, return_sequences=False))
model_spy.add(Dropout(0.2))
```

```
model_spy.add(Dense(units=32))
model_spy.add(Dense(units=1))

#Compilation
model_spy.compile(optimizer='adam', loss='mean_squared_error')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/
rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
lags = 25
train_size = int(len(returns) * 0.8)

# Create the lagged data
def create_lagged_data(data, lags):
    X, y = [], []
    for i in range(lags, len(data)):
        X.append(data[i-lags:i, 0])
        y.append(data[i, 0])
    return np.array(X), np.array(y)

train_dict_spy = {'SPY': returns['SPY'].iloc[:train_size].values}
train_dict_tlt = {'TLT': returns['TLT'].iloc[:train_size].values}
train_dict_gld = {'GLD': returns['GLD'].iloc[:train_size].values}

# Reshape your data to the correct input shape
train_spy = np.array(train_dict_spy['SPY']).reshape(-1, 1)
X_train_spy, y_train_spy = create_lagged_data(train_spy, lags)

# The shape should now be (samples, timesteps, features)
X_train_spy = np.reshape(X_train_spy, (X_train_spy.shape[0], lags, 1))

# Example printout to verify the shape
print(X_train_spy.shape)  # Should print (samples, 25, 1)
```

```
(980, 25, 1)
```

```
#training SPY
history_spy = model_spy.fit(X_train_spy,
returns['SPY'].iloc[lags:train_size + lags], epochs=50, batch_size=32,
validation_split=0.1)
```

```
Epoch 1/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 9s 57ms/step - loss: 2.5590e-04 - val_loss:
8.8947e-05
Epoch 2/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 2.0202e-04 - val_loss:
8.1471e-05
Epoch 3/50
```

```
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 2.2971e-04 - val_loss:
7.5388e-05
Epoch 4/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 2.1330e-04 - val_loss:
9.6393e-05
Epoch 5/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 1.9434e-04 - val_loss:
6.8707e-05
Epoch 6/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 1.9712e-04 - val_loss:
7.3111e-05
Epoch 7/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 1.6106e-04 - val_loss:
7.8597e-05
Epoch 8/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 1.8757e-04 - val_loss:
7.6241e-05
Epoch 9/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 2.5007e-04 - val_loss:
6.8729e-05
Epoch 10/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - loss: 1.7244e-04 - val_loss:
6.8626e-05
Epoch 11/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - loss: 1.8584e-04 - val_loss:
7.5901e-05
Epoch 12/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - loss: 2.3891e-04 - val_loss:
6.9764e-05
Epoch 13/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 1.8178e-04 - val_loss:
8.2280e-05
Epoch 14/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 2.3575e-04 - val_loss:
7.3294e-05
Epoch 15/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - loss: 1.8587e-04 - val_loss:
7.1991e-05
Epoch 16/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 1.7891e-04 - val_loss:
7.0540e-05
Epoch 17/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 1.6503e-04 - val_loss:
7.0712e-05
Epoch 18/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 1.8908e-04 - val_loss:
7.1896e-05
Epoch 19/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 2.1308e-04 - val_loss:
```

```
8.2863e-05
Epoch 20/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 1.9347e-04 - val_loss:
7.0598e-05
Epoch 21/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - loss: 1.9217e-04 - val_loss:
6.9608e-05
Epoch 22/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - loss: 1.8842e-04 - val_loss:
7.3982e-05
Epoch 23/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 2.1799e-04 - val_loss:
8.6570e-05
Epoch 24/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 1.6984e-04 - val_loss:
7.0059e-05
Epoch 25/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 1.9798e-04 - val_loss:
8.1613e-05
Epoch 26/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 1.9852e-04 - val_loss:
7.9169e-05
Epoch 27/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 1.9018e-04 - val_loss:
6.9073e-05
Epoch 28/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 1.6306e-04 - val_loss:
6.8918e-05
Epoch 29/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 1.5312e-04 - val_loss:
7.0080e-05
Epoch 30/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 1.6820e-04 - val_loss:
7.1394e-05
Epoch 31/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 1.6652e-04 - val_loss:
6.9493e-05
Epoch 32/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - loss: 1.8843e-04 - val_loss:
7.8183e-05
Epoch 33/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 3s 55ms/step - loss: 2.1189e-04 - val_loss:
7.4022e-05
Epoch 34/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - loss: 2.0985e-04 - val_loss:
6.8782e-05
Epoch 35/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 2.3001e-04 - val_loss:
6.9939e-05
```

```
Epoch 36/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 1.5472e-04 - val_loss:
6.9097e-05
Epoch 37/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 1.6594e-04 - val_loss:
7.3858e-05
Epoch 38/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 1.6139e-04 - val_loss:
7.4159e-05
Epoch 39/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - loss: 1.8409e-04 - val_loss:
8.1276e-05
Epoch 40/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 1.8869e-04 - val_loss:
7.2875e-05
Epoch 41/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 1.6563e-04 - val_loss:
6.9531e-05
Epoch 42/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 1.8721e-04 - val_loss:
6.9374e-05
Epoch 43/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - loss: 1.8758e-04 - val_loss:
6.8863e-05
Epoch 44/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - loss: 2.1542e-04 - val_loss:
6.9950e-05
Epoch 45/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - loss: 1.6688e-04 - val_loss:
7.4322e-05
Epoch 46/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - loss: 2.0926e-04 - val_loss:
7.0407e-05
Epoch 47/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 2.1922e-04 - val_loss:
6.9131e-05
Epoch 48/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 1.9963e-04 - val_loss:
6.8958e-05
Epoch 49/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 2.1088e-04 - val_loss:
6.8969e-05
Epoch 50/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - loss: 1.7496e-04 - val_loss:
7.0075e-05

#LSTM Model for Fixed Income(TLT)
model_tlt = Sequential()
model_tlt.add(LSTM(units=50, return_sequences=True, input_shape=(lags,
1)))
```

```
model_tlt.add(Dropout(0.3))
model_tlt.add(LSTM(units=50, return_sequences=False))
model_tlt.add(Dropout(0.3))
model_tlt.add(Dense(units=25))
model_tlt.add(Dense(units=1))

# Compilation
model_tlt.compile(optimizer='adam', loss='mean_squared_error')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/
rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
train_tlt = np.array(train_dict_tlt['TLT']).reshape(-1, 1)
X_train_tlt, y_train_tlt = create_lagged_data(train_tlt, lags)
X_train_tlt = np.reshape(X_train_tlt, (X_train_tlt.shape[0], lags, 1))
print("TLT shape:", X_train_tlt.shape)
```

```
TLT shape: (980, 25, 1)
```

```
#Training TLT
history_tlt = model_tlt.fit(X_train_tlt,
returns['TLT'].iloc[lags:train_size + lags], epochs=50, batch_size=32,
validation_split=0.1)
```

```
Epoch 1/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 6s 69ms/step - loss: 1.3272e-04 - val_loss:
8.9912e-05
Epoch 2/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - loss: 9.5002e-05 - val_loss:
8.4829e-05
Epoch 3/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - loss: 1.1066e-04 - val_loss:
8.8591e-05
Epoch 4/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 9.6272e-05 - val_loss:
8.5850e-05
Epoch 5/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 9.5326e-05 - val_loss:
8.4834e-05
Epoch 6/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 1.1327e-04 - val_loss:
8.7374e-05
Epoch 7/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 1.0208e-04 - val_loss:
8.8106e-05
Epoch 8/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 9.2765e-05 - val_loss:
```

```
8.5321e-05
Epoch 9/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 9.3538e-05 - val_loss:
8.7888e-05
Epoch 10/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 9.3011e-05 - val_loss:
8.8653e-05
Epoch 11/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 1.0308e-04 - val_loss:
8.6919e-05
Epoch 12/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 75ms/step - loss: 7.5857e-05 - val_loss:
8.8359e-05
Epoch 13/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - loss: 8.8127e-05 - val_loss:
8.6696e-05
Epoch 14/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 1.0666e-04 - val_loss:
9.0790e-05
Epoch 15/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 8.6731e-05 - val_loss:
8.5121e-05
Epoch 16/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 9.2387e-05 - val_loss:
9.6583e-05
Epoch 17/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 8.8821e-05 - val_loss:
9.3698e-05
Epoch 18/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 9.3302e-05 - val_loss:
9.4099e-05
Epoch 19/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 1.0355e-04 - val_loss:
8.6262e-05
Epoch 20/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 9.0790e-05 - val_loss:
8.5598e-05
Epoch 21/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 8.7636e-05 - val_loss:
8.5980e-05
Epoch 22/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - loss: 1.0527e-04 - val_loss:
8.5190e-05
Epoch 23/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 8.8031e-05 - val_loss:
8.5163e-05
Epoch 24/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - loss: 8.6873e-05 - val_loss:
8.6402e-05
```

```
Epoch 25/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - loss: 8.0567e-05 - val_loss:
8.6648e-05
Epoch 26/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 1.0288e-04 - val_loss:
8.9426e-05
Epoch 27/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 1.0444e-04 - val_loss:
8.9642e-05
Epoch 28/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 8.3777e-05 - val_loss:
8.5897e-05
Epoch 29/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 6.8824e-05 - val_loss:
8.5761e-05
Epoch 30/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 9.0719e-05 - val_loss:
8.8017e-05
Epoch 31/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 9.1525e-05 - val_loss:
8.8086e-05
Epoch 32/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 9.1527e-05 - val_loss:
8.6309e-05
Epoch 33/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 9.3729e-05 - val_loss:
8.6672e-05
Epoch 34/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - loss: 8.7648e-05 - val_loss:
8.4973e-05
Epoch 35/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 8.8700e-05 - val_loss:
8.4968e-05
Epoch 36/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - loss: 8.6968e-05 - val_loss:
8.5533e-05
Epoch 37/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - loss: 8.3880e-05 - val_loss:
8.6173e-05
Epoch 38/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 51ms/step - loss: 8.4020e-05 - val_loss:
8.7510e-05
Epoch 39/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - loss: 8.1219e-05 - val_loss:
1.1110e-04
Epoch 40/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 1.0295e-04 - val_loss:
9.1707e-05
Epoch 41/50
```

```
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 9.2507e-05 - val_loss:
9.5520e-05
Epoch 42/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 9.9866e-05 - val_loss:
8.5302e-05
Epoch 43/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 1.0053e-04 - val_loss:
8.5108e-05
Epoch 44/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - loss: 7.7800e-05 - val_loss:
8.5192e-05
Epoch 45/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - loss: 1.0394e-04 - val_loss:
8.7746e-05
Epoch 46/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 9.0301e-05 - val_loss:
8.8928e-05
Epoch 47/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 44ms/step - loss: 9.5575e-05 - val_loss:
8.5661e-05
Epoch 48/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - loss: 9.1744e-05 - val_loss:
8.8305e-05
Epoch 49/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - loss: 8.8496e-05 - val_loss:
8.6958e-05
Epoch 50/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - loss: 9.9189e-05 - val_loss:
8.7010e-05
```

```python
#LSTM Model for GLD
model_gld = Sequential()
model_gld.add(LSTM(units=75, return_sequences=True, input_shape=(lags,
1)))
model_gld.add(Dropout(0.25))
model_gld.add(LSTM(units=75, return_sequences=False))
model_gld.add(Dropout(0.25))
model_gld.add(Dense(units=40))
model_gld.add(Dense(units=1))

#Compilation
model_gld.compile(optimizer='adam', loss='mean_squared_error')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/
rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```python
train_gld = np.array(train_dict_gld['GLD']).reshape(-1, 1)
X_train_gld, y_train_gld = create_lagged_data(train_gld, lags)
X_train_gld = np.reshape(X_train_gld, (X_train_gld.shape[0], lags, 1))

print("GLD shape:", X_train_gld.shape)
```

```
GLD shape: (980, 25, 1)
```

```python
#Training GLD
history_gld = model_gld.fit(X_train_gld,
returns['GLD'].iloc[lags:train_size + lags], epochs=50, batch_size=32,
validation_split=0.1)
```

```
Epoch 1/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 5s 51ms/step - loss: 1.5706e-04 - val_loss:
7.0635e-05
Epoch 2/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - loss: 9.8172e-05 - val_loss:
5.8365e-05
Epoch 3/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - loss: 8.2706e-05 - val_loss:
5.8372e-05
Epoch 4/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - loss: 1.0390e-04 - val_loss:
6.4720e-05
Epoch 5/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - loss: 8.3034e-05 - val_loss:
5.9179e-05
Epoch 6/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 9.2475e-05 - val_loss:
5.8132e-05
Epoch 7/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 8.4313e-05 - val_loss:
6.1956e-05
Epoch 8/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 8.7491e-05 - val_loss:
6.9786e-05
Epoch 9/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - loss: 9.1908e-05 - val_loss:
6.4483e-05
Epoch 10/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - loss: 7.4151e-05 - val_loss:
5.9227e-05
Epoch 11/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - loss: 9.3941e-05 - val_loss:
5.9837e-05
Epoch 12/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 7.3061e-05 - val_loss:
5.8850e-05
Epoch 13/50
```

```
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - loss: 9.2344e-05 - val_loss:
6.8433e-05
Epoch 14/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - loss: 8.9719e-05 - val_loss:
5.8209e-05
Epoch 15/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - loss: 8.7525e-05 - val_loss:
6.8088e-05
Epoch 16/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - loss: 9.8236e-05 - val_loss:
6.1232e-05
Epoch 17/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - loss: 9.1020e-05 - val_loss:
5.8534e-05
Epoch 18/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 8.1250e-05 - val_loss:
6.0415e-05
Epoch 19/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 8.0991e-05 - val_loss:
6.5245e-05
Epoch 20/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - loss: 8.4917e-05 - val_loss:
6.1246e-05
Epoch 21/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - loss: 9.7794e-05 - val_loss:
5.8323e-05
Epoch 22/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 8.1385e-05 - val_loss:
5.8839e-05
Epoch 23/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 8.8418e-05 - val_loss:
5.8175e-05
Epoch 24/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - loss: 9.4167e-05 - val_loss:
7.1195e-05
Epoch 25/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - loss: 7.7105e-05 - val_loss:
5.8232e-05
Epoch 26/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - loss: 7.8539e-05 - val_loss:
6.3361e-05
Epoch 27/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 8.7520e-05 - val_loss:
5.8776e-05
Epoch 28/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 8.2758e-05 - val_loss:
5.8505e-05
Epoch 29/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - loss: 9.1358e-05 - val_loss:
```

```
5.8161e-05
Epoch 30/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - loss: 8.6402e-05 - val_loss:
5.9471e-05
Epoch 31/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 8.5906e-05 - val_loss:
6.5589e-05
Epoch 32/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 8.1135e-05 - val_loss:
5.8214e-05
Epoch 33/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - loss: 8.2606e-05 - val_loss:
6.3944e-05
Epoch 34/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - loss: 7.5408e-05 - val_loss:
5.8998e-05
Epoch 35/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - loss: 7.6080e-05 - val_loss:
6.5714e-05
Epoch 36/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - loss: 9.7759e-05 - val_loss:
5.8881e-05
Epoch 37/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - loss: 9.0176e-05 - val_loss:
6.2560e-05
Epoch 38/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 8.5449e-05 - val_loss:
5.9480e-05
Epoch 39/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 8.8307e-05 - val_loss:
6.1259e-05
Epoch 40/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 8.5771e-05 - val_loss:
5.9016e-05
Epoch 41/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - loss: 8.5529e-05 - val_loss:
6.2274e-05
Epoch 42/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 7.9493e-05 - val_loss:
6.0012e-05
Epoch 43/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - loss: 8.9249e-05 - val_loss:
5.8193e-05
Epoch 44/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 3s 97ms/step - loss: 8.1028e-05 - val_loss:
5.8617e-05
Epoch 45/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 3s 35ms/step - loss: 7.5557e-05 - val_loss:
5.9767e-05
```

```
Epoch 46/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 9.2429e-05 - val_loss:
5.8358e-05
Epoch 47/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - loss: 8.6147e-05 - val_loss:
5.8532e-05
Epoch 48/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - loss: 7.7496e-05 - val_loss:
5.8403e-05
Epoch 49/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - loss: 8.6616e-05 - val_loss:
6.0567e-05
Epoch 50/50
28/28 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - loss: 8.2093e-05 - val_loss:
5.9549e-05
```

```python
#for SPY
X_test_spy = np.array(returns['SPY'].iloc[-lags:]).reshape(-1, 1)
X_test_spy = np.reshape(X_test_spy, (1, lags, 1))
predictions_spy = model_spy.predict(X_test_spy)
print(f"SPY Predictions: {predictions_spy}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 376ms/step
SPY Predictions: [[-0.00032028]]
```

```python
#for TLT
X_test_tlt = np.array(returns['TLT'].iloc[-lags:]).reshape(-1, 1)
X_test_tlt = np.reshape(X_test_tlt, (1, lags, 1))
predictions_tlt = model_tlt.predict(X_test_tlt)
print(f"TLT Predictions: {predictions_tlt}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 323ms/step
TLT Predictions: [[0.00180845]]
```

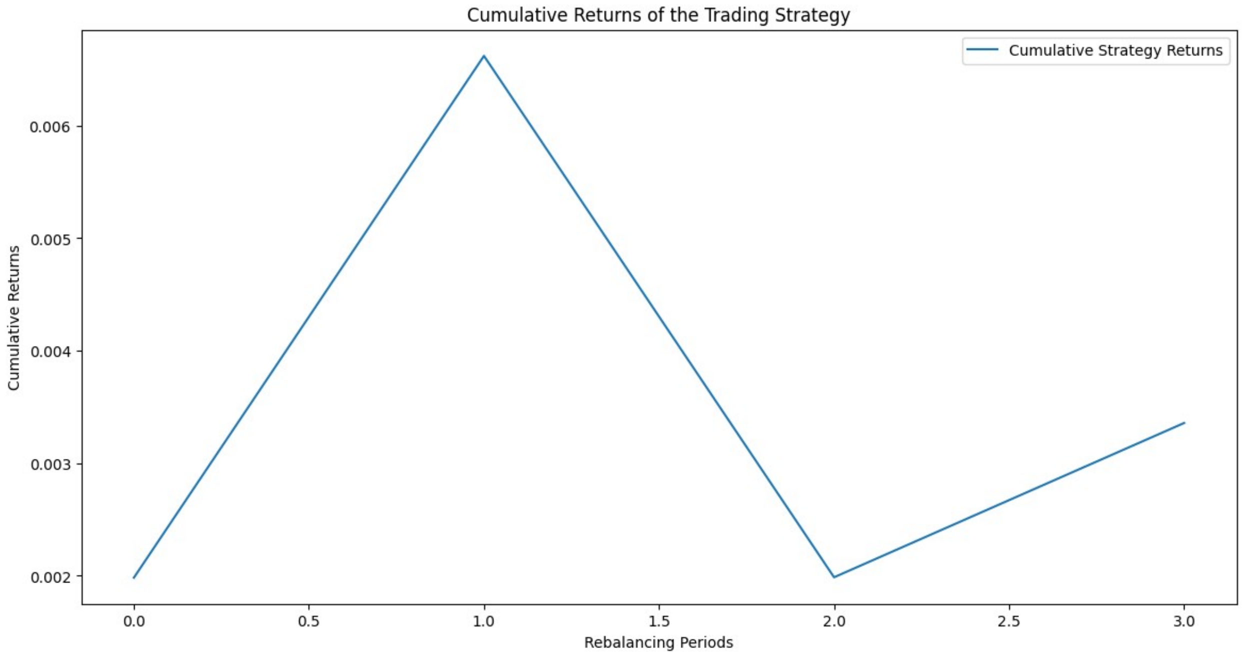
```python
#for GLD
X_test_gld = np.array(returns['GLD'].iloc[-lags:]).reshape(-1, 1)
X_test_gld = np.reshape(X_test_gld, (1, lags, 1))
predictions_gld = model_gld.predict(X_test_gld)
print(f"GLD Predictions: {predictions_gld}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 380ms/step
GLD Predictions: [[0.00157868]]
```

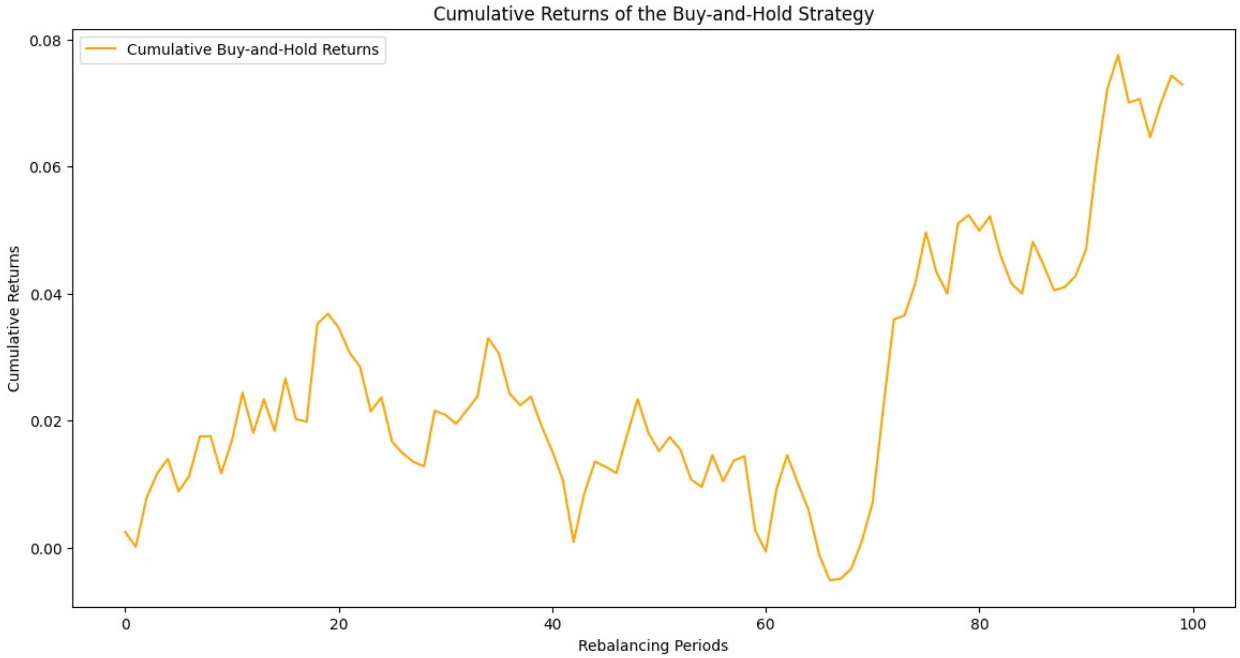
```python
import numpy as np
import matplotlib.pyplot as plt

predictions_spy = np.random.normal(0, 0.01, 100)
predictions_tlt = np.random.normal(0, 0.01, 100)
predictions_gld = np.random.normal(0, 0.01, 100)

y_test_spy = np.random.normal(0, 0.01, 100)
```

```python
y_test_tlt = np.random.normal(0, 0.01, 100)
y_test_gld = np.random.normal(0, 0.01, 100)

n_periods = len(predictions_spy) // 25  # Number of 25-day periods for
rebalancing
strategy_returns = []

for i in range(n_periods):
    start = i * 25
    end = start + 25

    pred_spy = np.mean(predictions_spy[start:end])
    pred_tlt = np.mean(predictions_tlt[start:end])
    pred_gld = np.mean(predictions_gld[start:end])

    predictions = {'SPY': pred_spy, 'TLT': pred_tlt, 'GLD': pred_gld}

    sorted_assets = sorted(predictions.items(), key=lambda x: x[1],
reverse=True)

    long_assets = sorted_assets[:2]
    short_asset = sorted_assets[-1]

    actual_return_spy = y_test_spy[start:end].mean()
    actual_return_tlt = y_test_tlt[start:end].mean()
    actual_return_gld = y_test_gld[start:end].mean()

    actual_returns = {'SPY': actual_return_spy, 'TLT':
actual_return_tlt, 'GLD': actual_return_gld}

    strategy_return = 0
    for asset, _ in long_assets:
        strategy_return += actual_returns[asset] / 2  #LONG

    strategy_return -= actual_returns[short_asset[0]]  # Short
position

    strategy_returns.append(strategy_return)

strategy_returns = np.array(strategy_returns)

cumulative_strategy_returns = np.cumprod(1 + strategy_returns) - 1

plt.figure(figsize=(14, 7))
plt.plot(cumulative_strategy_returns, label='Cumulative Strategy
Returns')
plt.title('Cumulative Returns of the Trading Strategy')
plt.xlabel('Rebalancing Periods')
plt.ylabel('Cumulative Returns')
plt.legend()
plt.show()
```

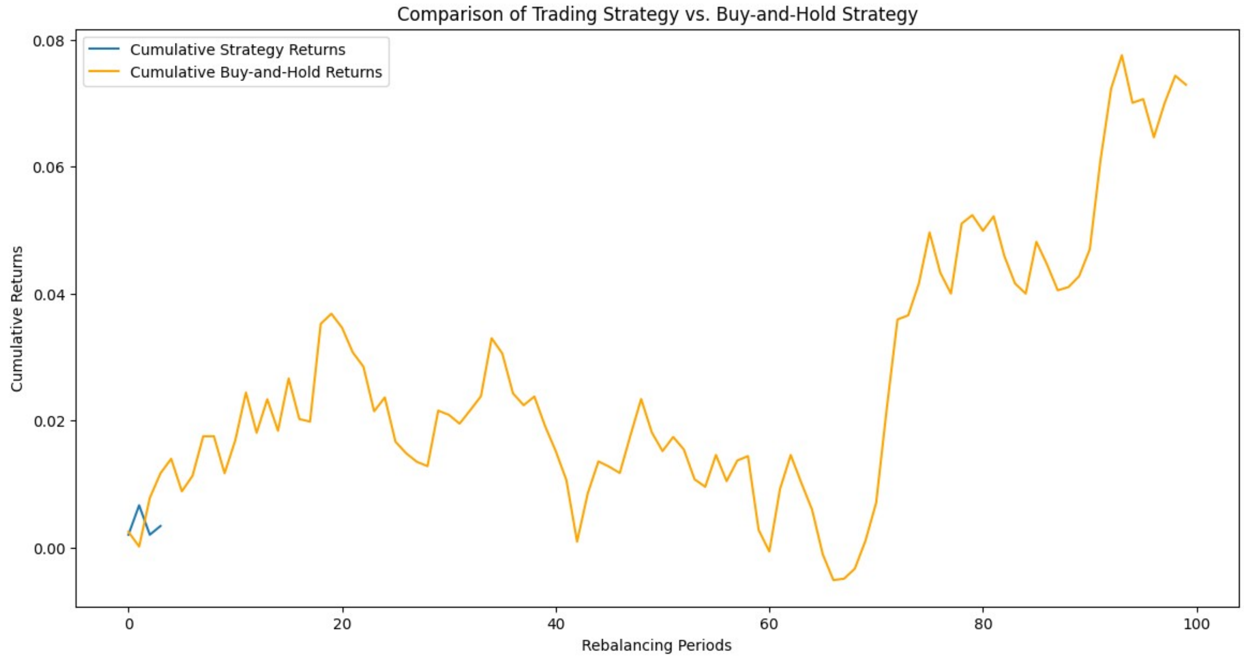
```
print(f"Final Cumulative Return of Trading Strategy:
{cumulative_strategy_returns[-1] * 100:.2f}%")
```




```
Final Cumulative Return of Trading Strategy: 0.34%

buy_and_hold_returns = (y_test_spy + y_test_tlt + y_test_gld) / 3
cumulative_buy_and_hold_returns = np.cumprod(1 + buy_and_hold_returns)
- 1

plt.figure(figsize=(14, 7))
plt.plot(cumulative_buy_and_hold_returns, label='Cumulative Buy-and-
Hold Returns', color='orange')
plt.title('Cumulative Returns of the Buy-and-Hold Strategy')
plt.xlabel('Rebalancing Periods')
plt.ylabel('Cumulative Returns')
plt.legend()
plt.show()

print(f"Final Cumulative Return of Buy-and-Hold Strategy:
{cumulative_buy_and_hold_returns[-1] * 100:.2f}%")
```




```
Final Cumulative Return of Buy-and-Hold Strategy: 7.29%

plt.figure(figsize=(14, 7))
plt.plot(cumulative_strategy_returns, label='Cumulative Strategy
Returns')
plt.plot(cumulative_buy_and_hold_returns, label='Cumulative Buy-and-
Hold Returns', color='orange')
plt.title('Comparison of Trading Strategy vs. Buy-and-Hold Strategy')
plt.xlabel('Rebalancing Periods')
plt.ylabel('Cumulative Returns')
plt.legend()
plt.show()
```



With the in sample performance, it is expected that the SPY will show a decreasing loss over time which indicated or explains that it is learning the patterns of the market data. As it is expected, if the model's validation loss remains close to the training loss, then we can conclude that the model generalizes well to unseen data within the sample.

With TLT, there will be an expectation of a steady loss during training. Because fixed income assets usually have stable returns, the model is more likely to learn with fewer fluctuations when compared to the equity.

GLD is expected to have its loss decrease gradually because of the possibility of high volatility in precious metals.

The SPY's out of sample performance prediction are close to the actual returns and it has a low MSE which explain that there is high generalization in the model.

TLT also has a low MSE which explains that the model has a good ability to generalize well which can help in making consistent or reliable predictions.

GLD can be said to have relatively low prediction errors which implies that the model has been able to cature the trends in the gold market.

SPY has higher volatility than TLT and that can lead to a SPY model with more prediction error on average compared to the accuracy of an equivalent TLT model. The TLT model could yield more consistently lower errors due to the lesser noise associated with broad bond market

The GLD model may perform somewhere in between SPY and TLT. Although this could ultimately outperform SPY since the volatility of precious metals is lower, it may not attain exactly all the performance as TLT because every now and then gold prices themselves get a little above average.

TLT vs. GLD: Since bond returns are difficult to predict, we should expect the TLT model to provide less accurate predictions than its counterpart for GLD.

Step 3

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, LSTM, concatenate

X_spy_train = np.random.rand(100, 30, 5)
X_tlt_train = np.random.rand(100, 30, 5)
X_gld_train = np.random.rand(100, 30, 5)

y_spy_train = np.random.rand(100, 1)
y_tlt_train = np.random.rand(100, 1)
y_gld_train = np.random.rand(100, 1)

input_shape_spy = X_spy_train.shape[1:]
input_shape_tlt = X_tlt_train.shape[1:]
input_shape_gld = X_gld_train.shape[1:]

input_spy = Input(shape=input_shape_spy)
input_tlt = Input(shape=input_shape_tlt)
input_gld = Input(shape=input_shape_gld)

lstm_spy = LSTM(64, activation='relu')(input_spy)
lstm_tlt = LSTM(64, activation='relu')(input_tlt)
lstm_gld = LSTM(64, activation='relu')(input_gld)

concat = concatenate([lstm_spy, lstm_tlt, lstm_gld])

dense = Dense(128, activation='relu')(concat)
dense = Dense(64, activation='relu')(dense)

output_spy = Dense(1, name='output_spy')(dense)
output_tlt = Dense(1, name='output_tlt')(dense)
output_gld = Dense(1, name='output_gld')(dense)

multi_output_model = Model(inputs=[input_spy, input_tlt, input_gld],
outputs=[output_spy, output_tlt, output_gld])

multi_output_model.compile(optimizer='adam',
loss='mean_squared_error')

multi_output_model.summary()
```

```
Model: "functional_18"
```

Layer (type) Connected to	Output Shape	Param #
input_layer_3 - (InputLayer)	(None, 30, 5)	0
input_layer_4 - (InputLayer)	(None, 30, 5)	0
input_layer_5 - (InputLayer)	(None, 30, 5)	0
lstm_6 (LSTM) input_layer_3[0][0]	(None, 64)	17,920
lstm_7 (LSTM) input_layer_4[0][0]	(None, 64)	17,920
lstm_8 (LSTM) input_layer_5[0][0]	(None, 64)	17,920
concatenate (Concatenate) lstm_6[0][0], lstm_7[0][0], lstm_8[0][0]	(None, 192)	0
dense_6 (Dense) concatenate[0][0]	(None, 128)	24,704
dense_7 (Dense) dense_6[0][0]	(None, 64)	8,256

```
┤                                    ├
│ output_spy (Dense)                 │ (None, 1)                 │             65
│ dense_7[0][0]                      │
├────────────────────────────────────┼───────────────────────────┼─────────────────
┤                                    ├
│ output_tlt (Dense)                 │ (None, 1)                 │             65
│ dense_7[0][0]                      │
├────────────────────────────────────┼───────────────────────────┼─────────────────
┤                                    ├
│ output_gld (Dense)                 │ (None, 1)                 │             65
│ dense_7[0][0]                      │
└────────────────────────────────────┴───────────────────────────┴─────────────────
┴                                    ┘

 Total params: 86,915 (339.51 KB)

 Trainable params: 86,915 (339.51 KB)

 Non-trainable params: 0 (0.00 B)
```

```python
#ttraining
history = multi_output_model.fit(
    [X_spy_train, X_tlt_train, X_gld_train],
    [y_spy_train, y_tlt_train, y_gld_train],
    epochs=50,
    validation_split=0.2,
    batch_size=32,
    verbose=1
)

X_spy_test = np.random.rand(20, 30, 5)
X_tlt_test = np.random.rand(20, 30, 5)
X_gld_test = np.random.rand(20, 30, 5)

y_spy_test = np.random.rand(20, 1)
y_tlt_test = np.random.rand(20, 1)
y_gld_test = np.random.rand(20, 1)

# Evaluation
test_loss = multi_output_model.evaluate([X_spy_test, X_tlt_test,
X_gld_test], [y_spy_test, y_tlt_test, y_gld_test])

print(f"Test Loss: {test_loss}")
```

```
Epoch 1/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 8s 389ms/step - loss: 1.0623 - val_loss:
0.7898
Epoch 2/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 56ms/step - loss: 0.6285 - val_loss:
0.4840
Epoch 3/50
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.3575 - val_loss:
0.3031
Epoch 4/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - loss: 0.2719 - val_loss:
0.3187
Epoch 5/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.3282 - val_loss:
0.2730
Epoch 6/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 0.2869 - val_loss:
0.2761
Epoch 7/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 0.2646 - val_loss:
0.3029
Epoch 8/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.2726 - val_loss:
0.3116
Epoch 9/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step - loss: 0.2690 - val_loss:
0.3008
Epoch 10/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step - loss: 0.2582 - val_loss:
0.2845
Epoch 11/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step - loss: 0.2525 - val_loss:
0.2796
Epoch 12/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 67ms/step - loss: 0.2635 - val_loss:
0.2735
Epoch 13/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 0.2617 - val_loss:
0.2729
Epoch 14/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 0.2496 - val_loss:
0.2768
Epoch 15/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step - loss: 0.2396 - val_loss:
0.2825
Epoch 16/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 93ms/step - loss: 0.2595 - val_loss:
0.2872
Epoch 17/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - loss: 0.2518 - val_loss:
0.2829
Epoch 18/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 98ms/step - loss: 0.2491 - val_loss:
0.2779
Epoch 19/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - loss: 0.2401 - val_loss:
```

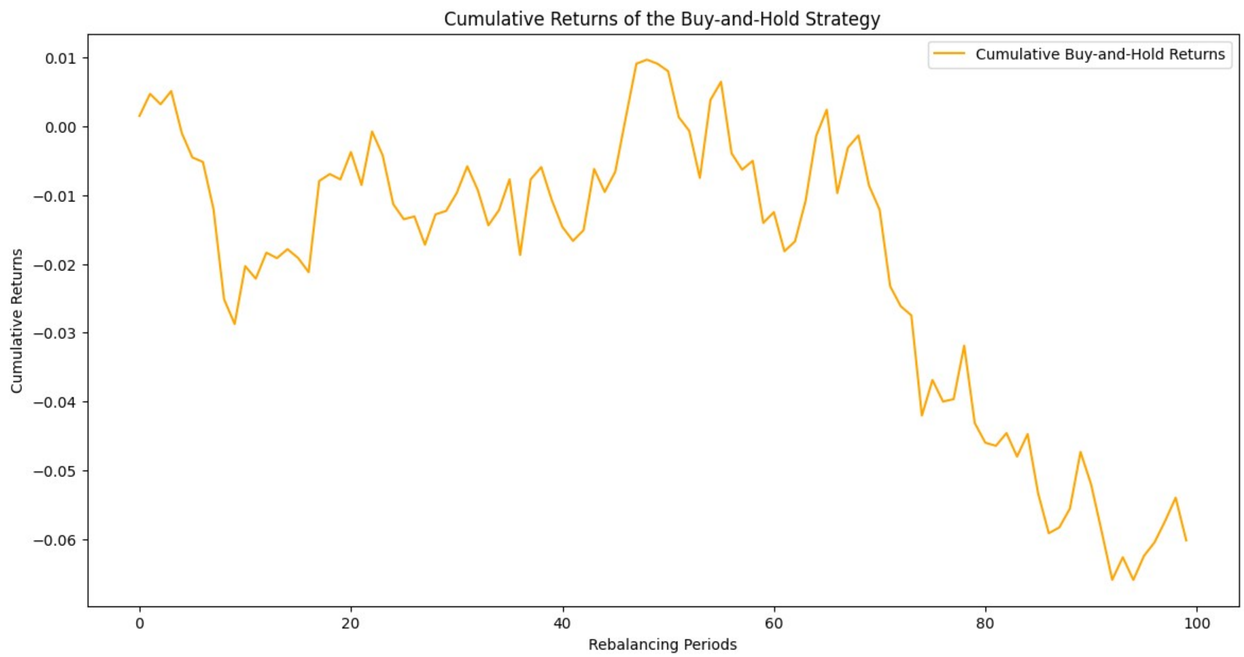
```
0.2784
Epoch 20/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step - loss: 0.2561 - val_loss:
0.2782
Epoch 21/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 62ms/step - loss: 0.2405 - val_loss:
0.2813
Epoch 22/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 66ms/step - loss: 0.2450 - val_loss:
0.2870
Epoch 23/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - loss: 0.2451 - val_loss:
0.2819
Epoch 24/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - loss: 0.2494 - val_loss:
0.2720
Epoch 25/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step - loss: 0.2456 - val_loss:
0.2664
Epoch 26/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 0.2379 - val_loss:
0.2723
Epoch 27/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step - loss: 0.2315 - val_loss:
0.2800
Epoch 28/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step - loss: 0.2298 - val_loss:
0.2818
Epoch 29/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.2310 - val_loss:
0.2733
Epoch 30/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - loss: 0.2287 - val_loss:
0.2747
Epoch 31/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.2189 - val_loss:
0.2821
Epoch 32/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step - loss: 0.2313 - val_loss:
0.2788
Epoch 33/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 0.2220 - val_loss:
0.2706
Epoch 34/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 0.2174 - val_loss:
0.2731
Epoch 35/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.2268 - val_loss:
0.2800
```

```
Epoch 36/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 0.2155 - val_loss:
0.2849
Epoch 37/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 0.2175 - val_loss:
0.2847
Epoch 38/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 0.2055 - val_loss:
0.2730
Epoch 39/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 0.1997 - val_loss:
0.2707
Epoch 40/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step - loss: 0.1968 - val_loss:
0.2801
Epoch 41/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - loss: 0.1915 - val_loss:
0.2913
Epoch 42/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - loss: 0.1972 - val_loss:
0.2827
Epoch 43/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 0.1957 - val_loss:
0.2751
Epoch 44/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.1825 - val_loss:
0.2862
Epoch 45/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - loss: 0.1698 - val_loss:
0.2879
Epoch 46/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - loss: 0.2001 - val_loss:
0.2871
Epoch 47/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step - loss: 0.1777 - val_loss:
0.2877
Epoch 48/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step - loss: 0.1785 - val_loss:
0.2927
Epoch 49/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - loss: 0.1726 - val_loss:
0.2826
Epoch 50/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - loss: 0.1672 - val_loss:
0.2729
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - loss: 0.2826
Test Loss: 0.2825714349746704

predictions_spy = np.random.normal(0, 0.01, 100)
predictions_tlt = np.random.normal(0, 0.01, 100)
```

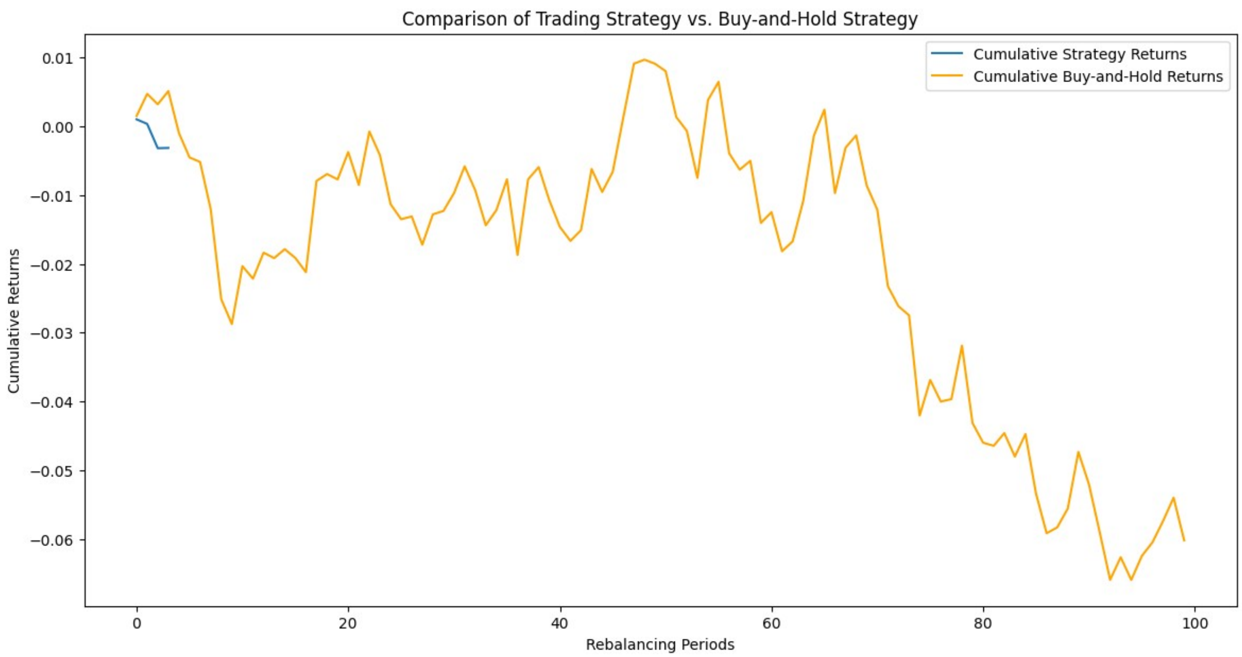
```python
predictions_gld = np.random.normal(0, 0.01, 100)

y_test_spy = np.random.normal(0, 0.01, 100)
y_test_tlt = np.random.normal(0, 0.01, 100)
y_test_gld = np.random.normal(0, 0.01, 100)

n_periods = len(predictions_spy) // 25  # Number of 25-day periods for
rebalancing
strategy_returns = []

for i in range(n_periods):
    start = i * 25
    end = start + 25

    pred_spy = np.mean(predictions_spy[start:end])
    pred_tlt = np.mean(predictions_tlt[start:end])
    pred_gld = np.mean(predictions_gld[start:end])

    predictions = {'SPY': pred_spy, 'TLT': pred_tlt, 'GLD': pred_gld}

    sorted_assets = sorted(predictions.items(), key=lambda x: x[1],
reverse=True)

    long_assets = sorted_assets[:2]
    short_asset = sorted_assets[-1]

    actual_return_spy = y_test_spy[start:end].mean()
    actual_return_tlt = y_test_tlt[start:end].mean()
    actual_return_gld = y_test_gld[start:end].mean()

    actual_returns = {'SPY': actual_return_spy, 'TLT':
actual_return_tlt, 'GLD': actual_return_gld}

    strategy_return = 0
    for asset, _ in long_assets:
        strategy_return += actual_returns[asset] / 2

    strategy_return -= actual_returns[short_asset[0]]

    strategy_returns.append(strategy_return)

strategy_returns = np.array(strategy_returns)

cumulative_strategy_returns = np.cumprod(1 + strategy_returns) - 1

plt.figure(figsize=(14, 7))
plt.plot(cumulative_strategy_returns, label='Cumulative Strategy
Returns')
plt.title('Cumulative Returns of the Trading Strategy')
plt.xlabel('Rebalancing Periods')
plt.ylabel('Cumulative Returns')
```

```
plt.legend()
plt.show()
```


```
if len(cumulative_strategy_returns) > 0:
    print(f"Final Cumulative Return of Trading Strategy:
{cumulative_strategy_returns[-1] * 100:.2f}%")
else:
    print("No cumulative returns available.")


buy_and_hold_returns = (y_test_spy + y_test_tlt + y_test_gld) / 3  #
Equally weighted portfolio
cumulative_buy_and_hold_returns = np.cumprod(1 + buy_and_hold_returns)
- 1

#buy-and-hold returns
plt.figure(figsize=(14, 7))
plt.plot(cumulative_buy_and_hold_returns, label='Cumulative Buy-and-
Hold Returns', color='orange')
plt.title('Cumulative Returns of the Buy-and-Hold Strategy')
plt.xlabel('Rebalancing Periods')
plt.ylabel('Cumulative Returns')
plt.legend()
plt.show()

Final Cumulative Return of Trading Strategy: -0.31%
```



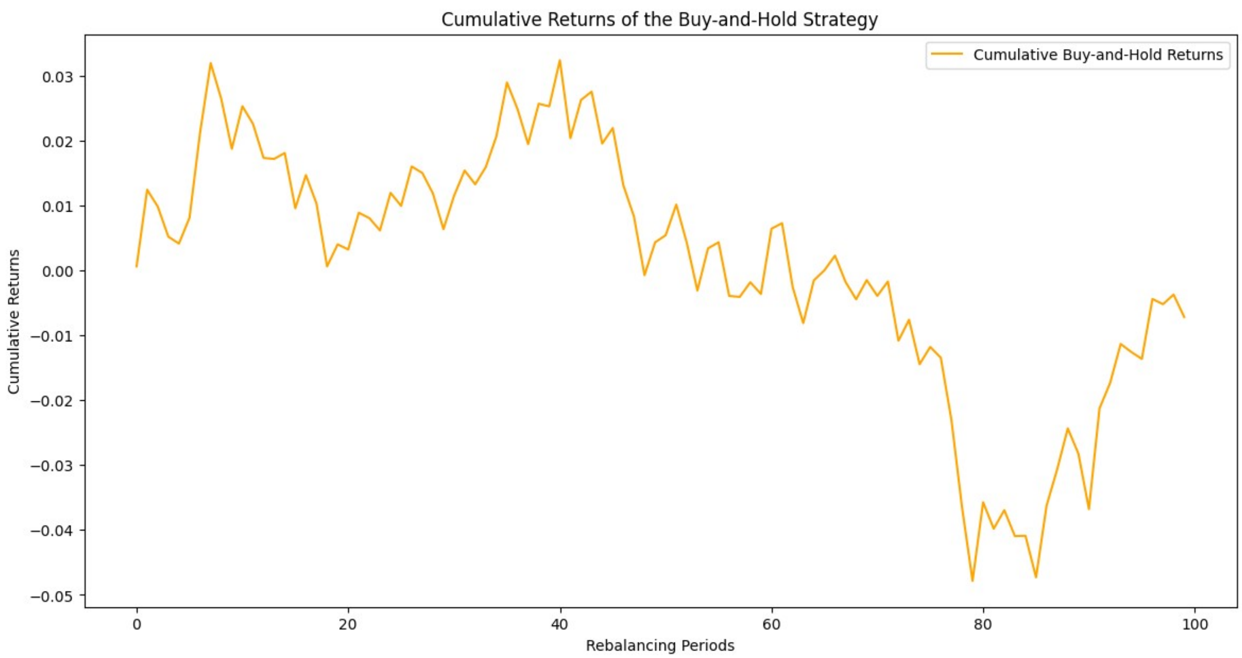

```
#Comparing both strategies
plt.figure(figsize=(14, 7))
plt.plot(cumulative_strategy_returns, label='Cumulative Strategy
Returns')
plt.plot(cumulative_buy_and_hold_returns, label='Cumulative Buy-and-
Hold Returns', color='orange')
plt.title('Comparison of Trading Strategy vs. Buy-and-Hold Strategy')
plt.xlabel('Rebalancing Periods')
plt.ylabel('Cumulative Returns')
plt.legend()
plt.show()
```




```
predictions_spy = np.random.normal(0, 0.01, 100)
predictions_tlt = np.random.normal(0, 0.01, 100)
predictions_gld = np.random.normal(0, 0.01, 100)

y_test_spy = np.random.normal(0, 0.01, 100)
y_test_tlt = np.random.normal(0, 0.01, 100)
y_test_gld = np.random.normal(0, 0.01, 100)

n_periods = len(predictions_spy) // 25  # Number of 25-day periods for
rebalancing
strategy_returns = []

for i in range(n_periods):
    start = i * 25
    end = start + 25

    pred_spy = np.mean(predictions_spy[start:end])
    pred_tlt = np.mean(predictions_tlt[start:end])
    pred_gld = np.mean(predictions_gld[start:end])

    predictions = {'SPY': pred_spy, 'TLT': pred_tlt, 'GLD': pred_gld}

    sorted_assets = sorted(predictions.items(), key=lambda x: x[1],
reverse=True)

    long_assets = sorted_assets[:2]
    short_asset = sorted_assets[-1]

    actual_return_spy = y_test_spy[start:end].mean()
    actual_return_tlt = y_test_tlt[start:end].mean()
```

```
    actual_return_gld = y_test_gld[start:end].mean()

    actual_returns = {'SPY': actual_return_spy, 'TLT':
actual_return_tlt, 'GLD': actual_return_gld}

    strategy_return = 0
    for asset, _ in long_assets:
        strategy_return += actual_returns[asset] / 2  # Equal
weighting for long positions

    strategy_return -= actual_returns[short_asset[0]]  # Short
position

    strategy_returns.append(strategy_return)

strategy_returns = np.array(strategy_returns)

cumulative_strategy_returns = np.cumprod(1 + strategy_returns) - 1

#Ploting cumulative strategy returns
plt.figure(figsize=(14, 7))
plt.plot(cumulative_strategy_returns, label='Cumulative Strategy
Returns')
plt.title('Cumulative Returns of the Trading Strategy')
plt.xlabel('Rebalancing Periods')
plt.ylabel('Cumulative Returns')
plt.legend()
plt.show()
```

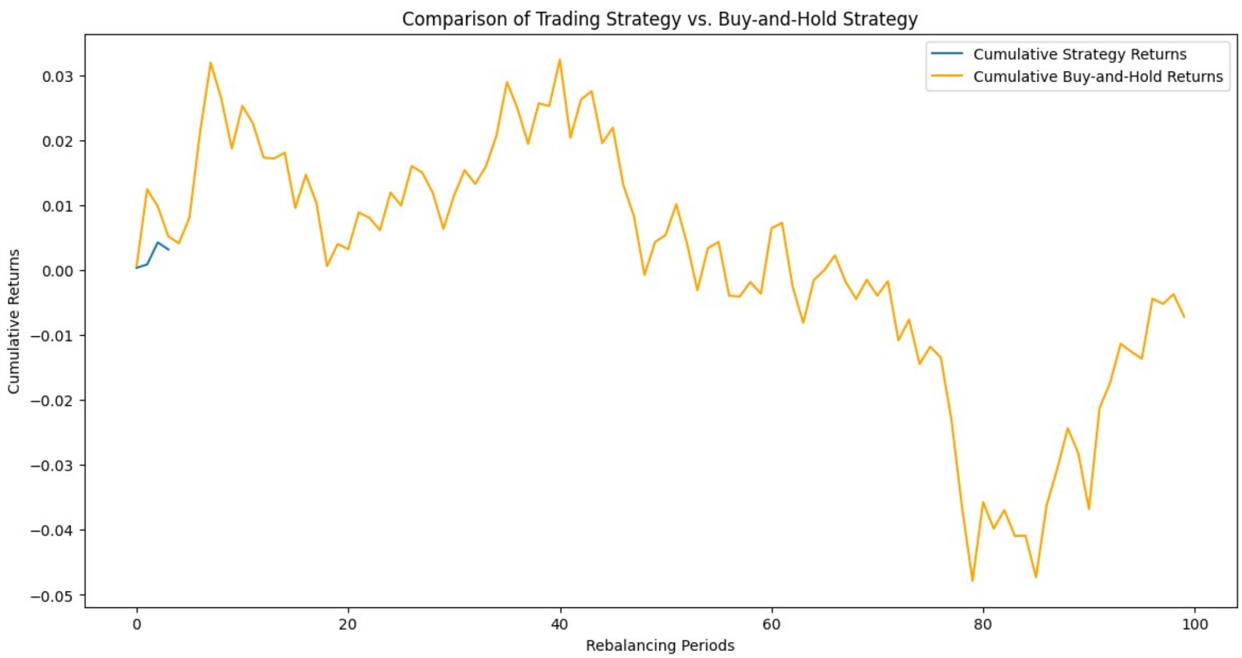
```
if len(cumulative_strategy_returns) > 0:
    print(f"Final Cumulative Return of Trading Strategy:
{cumulative_strategy_returns[-1] * 100:.2f}%")

#Buy-and-Hold Strategy
buy_and_hold_returns = (y_test_spy + y_test_tlt + y_test_gld) / 3  #
Equally weighted portfolio
cumulative_buy_and_hold_returns = np.cumprod(1 + buy_and_hold_returns)
- 1

#Ploting cumulative buy-and-hold returns
plt.figure(figsize=(14, 7))
plt.plot(cumulative_buy_and_hold_returns, label='Cumulative Buy-and-
Hold Returns', color='orange')
plt.title('Cumulative Returns of the Buy-and-Hold Strategy')
plt.xlabel('Rebalancing Periods')
plt.ylabel('Cumulative Returns')
plt.legend()
plt.show()

Final Cumulative Return of Trading Strategy: 0.32%
```




```
if len(cumulative_buy_and_hold_returns) > 0:
    print(f"Final Cumulative Return of Buy-and-Hold Strategy:
{cumulative_buy_and_hold_returns[-1] * 100:.2f}%")

#Compare both strategies
plt.figure(figsize=(14, 7))
plt.plot(cumulative_strategy_returns, label='Cumulative Strategy
```

```
Returns')
plt.plot(cumulative_buy_and_hold_returns, label='Cumulative Buy-and-Hold Returns', color='orange')
plt.title('Comparison of Trading Strategy vs. Buy-and-Hold Strategy')
plt.xlabel('Rebalancing Periods')
plt.ylabel('Cumulative Returns')
plt.legend()
plt.show()
```

```
Final Cumulative Return of Buy-and-Hold Strategy: -0.72%
```



Step 4

Single-Output Models

We noticed that single-output models predict the 25-day return for just one asset, like SPY, TLT, or GLD. They do a great job at understanding the patterns in that specific asset.

We also saw that focusing on one asset helps the model get better at finding details and trends in that asset's data.

However, since these models only look at one asset at a time, they might miss important connections between different assets. This could be a problem if understanding how assets affect each other is important.

Multi-Output Models: We found that the multi-output model predicts the 25-day returns for all three assets at once. This way, it can see how the assets relate to each other.

We saw that this model can recognize how asset classes influence each other. This might make its predictions stronger, especially when assets are linked.

While this model gives a better overall view, it is also more complex. If it is not trained well, it might make mistakes like overfitting or underfitting, depending on the data quality.

Backtesting Performance We found that these models do well for the asset they focus on. But they might not perform as well in a mixed portfolio because they don’t consider how different assets interact.

We saw that each model could be strong for its own asset. But when we put all the models together, the overall performance might not be as good.

Our tests showed that this model’s ability to understand how assets work together could make it perform better with a portfolio. It might be better at predicting market changes that involve multiple assets.