In [1]:
```
import pandas as pd
import numpy as np
import csv
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.metrics import classification_report, accuracy_score, precision_score, reca
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error,mean_absol
from sklearn.preprocessing import MinMaxScaler
```

In [2]:
```
filename = "predicted_series_{0}_block{1}_w{2}.csv"
models = ['rnn','lstm']
blocks = ['1','10','20','30']
win = ['50','20']
```

In [3]:
```
test_data_file = pd.read_csv("test_data3.csv")
names = test_data_file.columns.tolist()
```

In [4]:
```
def mean_std(test_list):
    mean = sum(test_list) / len(test_list)
    variance = sum([((x - mean) ** 2) for x in test_list]) / len(test_list)
    res = variance ** 0.5
    return [mean, res]
```

In [5]:
```
from statistics import mean
```

In [6]:
```
R2score_tot = []
RMSE_tot = []
MAPE_tot = []
MAE_tot = []
MAPE_labels = []
files = []
windows = []
b = []
m = []
c=0
high_acc = []
good = []
mid = []
bad = []
for model in models:
    for block in blocks:
        for w in win:
            c+=1
            file = filename.format(model,block,w)
            testfile = "test_data{0}.csv".format(blocks.index(block)+1)
            windows.append(w)
            m.append(model)
            b.append(block)
            predicted_series_rnn = pd.read_csv(file)
            test_data_file = pd.read_csv(testfile)
            files.append(file)
            print(file, testfile)
            R2score = []
            RMSE = []
            MAPE = []
            MAE = []
            for name in names:
                test = test_data_file[name][int(w):].tolist()
                pred = predicted_series_rnn[name].tolist()
                #plot(pred,test)
                R2score.append(r2_score(test, pred))
                RMSE.append(np.sqrt(mean_squared_error(test, pred)))
```

```
                MAPE.append(mean_absolute_percentage_error(test, pred))
                MAE.append(mean_absolute_error(test, pred))

            high = 0
            good_ = 0
            mid_ = 0
            bad_ = 0
            for i in range(len(MAPE)):
                if MAPE[i]<=0.10:
                    high+=1
                elif MAPE[i]<=0.20 and MAPE[i]>0.10:
                    good_+=1
                elif MAPE[i]<=0.50 and MAPE[i]>0.20:
                    mid_+=1
                elif MAPE[i]>0.50:
                    bad_+=1

            high_acc.append(high)
            good.append(good_)
            mid.append(mid_)
            bad.append(bad_)
            R2score_tot.append(mean_std(R2score))
            RMSE_tot.append(mean_std(RMSE))
            MAPE_tot.append(mean_std(MAPE))
            MAE_tot.append(mean_std(MAE))
```

```
predicted_series_rnn_block1_w50.csv test_data1.csv
predicted_series_rnn_block1_w20.csv test_data1.csv
predicted_series_rnn_block10_w50.csv test_data2.csv
predicted_series_rnn_block10_w20.csv test_data2.csv
predicted_series_rnn_block20_w50.csv test_data3.csv
predicted_series_rnn_block20_w20.csv test_data3.csv
predicted_series_rnn_block30_w50.csv test_data4.csv
predicted_series_rnn_block30_w20.csv test_data4.csv
predicted_series_lstm_block1_w50.csv test_data1.csv
predicted_series_lstm_block1_w20.csv test_data1.csv
predicted_series_lstm_block10_w50.csv test_data2.csv
predicted_series_lstm_block10_w20.csv test_data2.csv
predicted_series_lstm_block20_w50.csv test_data3.csv
predicted_series_lstm_block20_w20.csv test_data3.csv
predicted_series_lstm_block30_w50.csv test_data4.csv
predicted_series_lstm_block30_w20.csv test_data4.csv
```

In [7]:
```
df = pd.DataFrame({
    'Files': files,
    'Block':b,
    'Window':windows,
    'Model': m,
    'High': high_acc,
    'Good': good,
    'Mid': mid,
    'Bad': bad
})
```

In [8]:
```
df_round = df.round(2)
df_round.to_csv("mape_labels.csv")
```

In [9]:
```
R2_mean = []
R2_std = []
for i in R2score_tot:
    print(i, i[1]/i[0])
    R2_mean.append(i[0])
    R2_std.append(i[1])
```

```
[0.7905424959699804, 0.1504867927385568] 0.1903588908954381
```

```
[0.7897788660808004, 0.12580194098077263] 0.1592875504570695
[0.9702673819021994, 0.04010606969724329] 0.04133506953373589
[0.9644707545270543, 0.028277614551443286] 0.029319307421933935
[0.9819790827441726, 0.024656725571361164] 0.025109216687647908
[0.9778311480320238, 0.027103317326623456] 0.02771778888530132
[0.9824152815270502, 0.026395411708256322] 0.026867875739094497
[0.9788110570555104, 0.028352615955096937] 0.02896638299161449
[0.7872163549790705, 0.1311912057821846] 0.16665203276381693
[0.7922328482554688, 0.09987584608149362] 0.12606880199606035
[0.9761188517741332, 0.0324048214051848] 0.03319761865708034
[0.9674927054050065, 0.04610109096541769] 0.04765006568821529
[0.9820570574218511, 0.03720732624419269] 0.03788713289416336
[0.9767062163791912, 0.022117303792646207] 0.022644786550697557
[0.981517241647321, 0.038326014365134865] 0.03904772401227586
[0.9814877501808776, 0.016806256734246416] 0.017123246552133944
```

In [10]:
```
MAPE_mean = []
MAPE_std = []
for i in MAPE_tot:
    print(i, i[1]/i[0])
    MAPE_mean.append(i[0])
    MAPE_std.append(i[1])
```

```
[0.19519403842956715, 0.14810321983657285] 0.7587486842740522
[0.19987011487793538, 0.1495318382929695] 0.7481450560244665
[0.06660278937384334, 0.042158541765406204] 0.6329846266463274
[0.0748010166107654, 0.04860910951978934] 0.6498455732591407
[0.04771831837837801, 0.03170268838521743] 0.6643714502643173
[0.051272615779275256, 0.02965868743556692] 0.5784508354956052
[0.04261971466867888, 0.02508246809574713] 0.5885179732134662
[0.0470851870733642, 0.02769338957816019] 0.5881550292029353
[0.17808714305986226, 0.11588936418714697] 0.6507452598540024
[0.17916123818062862, 0.1085848141360262] 0.6060731396963893
[0.0548325290685142, 0.031458211018119596] 0.5737143909377591
[0.06699232661279844, 0.04100126846219039] 0.6120293253758613
[0.04070058669170898, 0.0288365203492443] 0.7085038003916175
[0.051050026889971914, 0.024611010400292437] 0.482095934118439
[0.03819108922130367, 0.030001220359156557] 0.7855555044610077
[0.04631577095675475, 0.023755000853610615] 0.5128922689377398
```

In [11]:
```
MAE_mean = []
MAE_std =[]
for i in MAE_tot:
    print(i, i[1]/i[0])
    MAE_mean.append(i[0])
    MAE_std.append(i[1])
```

```
[1.024076463758599, 0.7291192462689874] 0.7119773494187631
[1.0409497443151305, 0.6453183761085421] 0.6199323066581998
[0.37429304952389275, 0.25496840105101076] 0.6811999351185789
[0.42368820361761395, 0.28180947789365757] 0.665134113924955
[0.26748321095537647, 0.19213548067068698] 0.71830856218763
[0.3051781248677179, 0.2209770181181784] 0.7240919322574736
[0.25078611939452194, 0.18348659332631734] 0.7316457297130828
[0.28266759962212656, 0.2226890986700329] 0.7878126073441963
[0.9895244069165563, 0.614067229379179] 0.6205680477277621
[0.9689504061684131, 0.5229167962871334] 0.5396734373175394
[0.3111157591399913, 0.20009169567264223] 0.6431422703425573
[0.3695158519876495, 0.2497426703612161] 0.6758645644505756
[0.24136652827431715, 0.2123147644170293] 0.8796363188177027
[0.29232491422445334, 0.17079888149335284] 0.5842775390749274
[0.23671356991761472, 0.2720678065320696] 1.1493544988855495
[0.27056490375016395, 0.18794472072640026] 0.6946382110960921
```

In [12]:
```
RMSE_mean = []
RMSE_std = []
```

```
for i in RMSE_tot:
    print(i, i[1]/i[0])
    RMSE_mean.append(i[0])
    RMSE_std.append(i[1])
```

```
[1.6670870879858384, 0.8905010965821514] 0.5341659131065839
[1.6870298628003126, 0.8323310950036094] 0.49337069447129833
[0.5130898523867832, 0.2921713742306395] 0.5694351055112888
[0.5956324083183224, 0.32841681448849114] 0.5513749921964892
[0.34883546340476174, 0.21981874360059012] 0.6301502188311898
[0.4141666006756527, 0.25998523567703485] 0.6277310513520565
[0.32040170596153306, 0.211825344879328] 0.6611242728675091
[0.379493494471496, 0.27218366077381606] 0.7172287924273231
[1.6801143484754053, 0.812830682415111] 0.4837948578635151
[1.665136502829104, 0.7599101133618877] 0.4563650560003841
[0.46143047852028113, 0.2494602198646781] 0.5406236290776654
[0.5383204655275987, 0.29532293688632355] 0.5486006120850014
[0.3302517363990494, 0.2581530940615623] 0.7816858039154441
[0.40872024771583254, 0.20553341815838705] 0.5028706537222656
[0.31846367971555395, 0.3236163269711548] 1.0161797014347227
[0.36731240855698954, 0.22624789608396179] 0.6159549495558596
```

In [13]:
```
df = pd.DataFrame({
    'Files': files,
    'Block':b,
    'Window':windows,
    'Model': m,
    'R2': R2_mean,
    'MAPE': MAPE_mean,
    'RMSE': RMSE_mean,
    'MAE': MAE_mean
})
```

In [14]:
```
df_round = df.round(2)
df_round.to_csv("results_mean.csv")
```

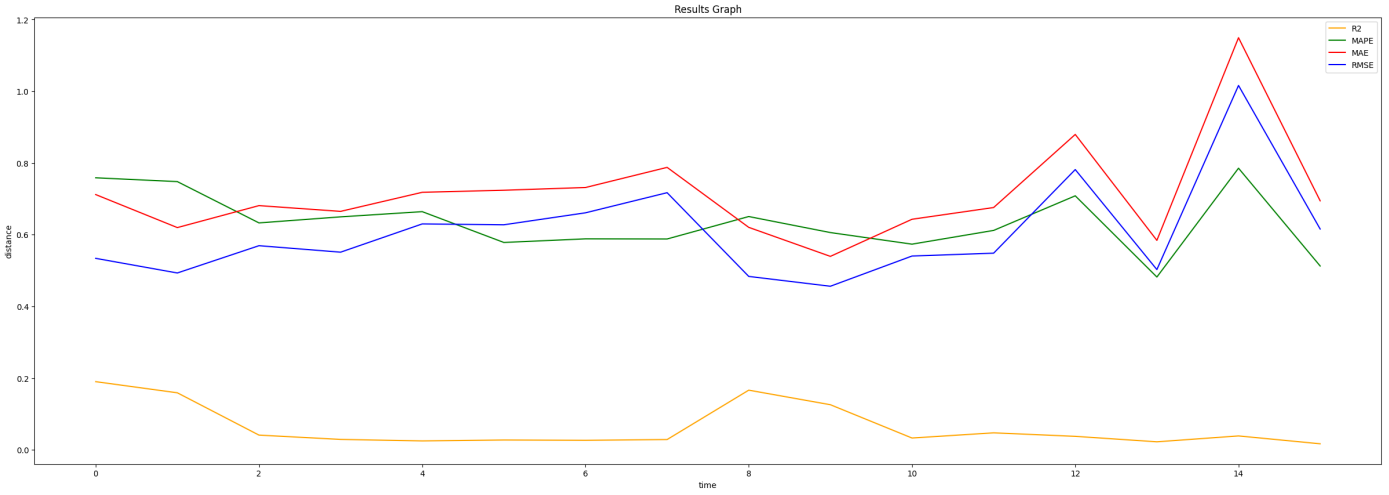
In [15]:
```
df = pd.DataFrame({
    'Files': files,
    'Block':b,
    'Window':windows,
    'Model': m,
    'R2': R2_std,
    'MAPE': MAPE_std,
    'RMSE': RMSE_std,
    'MAE': MAE_std
})

df_round = df.round(2)
df_round.to_csv("results_std.csv")
```

In [16]:
```
R2_cv = []
MAPE_cv = []
MAE_cv = []
RMSE_cv = []
for i in range(0,len(files)):
    R2_cv.append(R2score_tot[i][1]/R2score_tot[i][0])
    MAPE_cv.append(MAPE_tot[i][1]/MAPE_tot[i][0])
    MAE_cv.append(MAE_tot[i][1]/MAE_tot[i][0])
    RMSE_cv.append(RMSE_tot[i][1]/RMSE_tot[i][0])
    print(files[i],R2score_tot[i][1]/R2score_tot[i][0], MAPE_tot[i][1]/MAPE_tot[i][0], M
```

```
predicted_series_rnn_block1_w50.csv 0.1903588908954381 0.7587486842740522 0.711977349418
7631 0.5341659131065839
predicted_series_rnn_block1_w20.csv 0.1592875504570695 0.7481450560244665 0.619932306658
1998 0.49337069447129833
```

```
predicted_series_rnn_block10_w50.csv 0.04133506953373589 0.6329846266463274 0.6811999351
185789 0.5694351055112888
predicted_series_rnn_block10_w20.csv 0.029319307421933935 0.6498455732591407 0.665134113
924955 0.5513749921964892
predicted_series_rnn_block20_w50.csv 0.025109216687647908 0.6643714502643173 0.718308562
18763 0.6301502188311898
predicted_series_rnn_block20_w20.csv 0.02771778888530132 0.5784508354956052 0.7240919322
574736 0.6277310513520565
predicted_series_rnn_block30_w50.csv 0.026867875739094497 0.5885179732134662 0.731645729
7130828 0.6611242728675091
predicted_series_rnn_block30_w20.csv 0.02896638299161449 0.5881550292029353 0.7878126073
441963 0.7172287924273231
predicted_series_lstm_block1_w50.csv 0.16665203276381693 0.6507452598540024 0.6205680477
277621 0.4837948578635151
predicted_series_lstm_block1_w20.csv 0.12606880199606035 0.6060731396963893 0.5396734373
175394 0.4563650560003841
predicted_series_lstm_block10_w50.csv 0.03319761865708034 0.5737143909377591 0.643142270
3425573 0.5406236290776654
predicted_series_lstm_block10_w20.csv 0.04765006568821529 0.6120293253758613 0.675864564
4505756 0.5486006120850014
predicted_series_lstm_block20_w50.csv 0.03788713289416336 0.7085038003916175 0.879636318
8177027 0.7816858039154441
predicted_series_lstm_block20_w20.csv 0.022644786550697557 0.482095934118439 0.584277539
0749274 0.5028706537222656
predicted_series_lstm_block30_w50.csv 0.03904772401227586 0.7855555044610077 1.149354498
8855495 1.0161797014347227
predicted_series_lstm_block30_w20.csv 0.017123246552133944 0.5128922689377398 0.69463821
10960921 0.6159549495558596
```

In [17]:
```
plt.figure(figsize = (30,10))
plt.plot(R2_cv, label = "R2", c = "orange")
plt.plot(MAPE_cv, label = "MAPE", c = "green")
plt.plot(MAE_cv, label = "MAE", c = "red")
plt.plot(RMSE_cv, label = "RMSE", c = "blue")
plt.xlabel("time")
plt.ylabel("distance")
plt.title("Results Graph")
plt.legend()
plt.show()
```



In [18]:
```
print('R2:',files[R2_cv.index(min(R2_cv))],min(R2_cv))
print('MAPE:',files[MAPE_cv.index(min(MAPE_cv))],min(MAPE_cv))
print('MAE:',files[MAE_cv.index(min(MAE_cv))],min(MAE_cv))
print('RMSE:',files[RMSE_cv.index(min(RMSE_cv))],min(RMSE_cv))
```

```
R2: predicted_series_lstm_block30_w20.csv 0.017123246552133944
MAPE: predicted_series_lstm_block20_w20.csv 0.482095934118439
MAE: predicted_series_lstm_block1_w20.csv 0.5396734373175394
RMSE: predicted_series_lstm_block1_w20.csv 0.4563650560003841
```

In [19]:
```
df = pd.DataFrame({
    'Files': files,
    'Block':b,
    'Window':windows,
    'Model': m,
    'R2': R2_cv,
    'MAPE': MAPE_cv,
    'MAE': MAE_cv,
    'RMSE': RMSE_cv
})

df_round = df.round(2)
df_round.to_csv("results_cv.csv")
```

In [20]:
```
df['R2_rank'] = df['R2'].rank(ascending=True)
df['MAPE_rank'] = df['MAPE'].rank(ascending=True)
df['MAE_rank'] = df['MAE'].rank(ascending=True)
df['RMSE_rank'] = df['RMSE'].rank(ascending=True)

df['average_rank'] = df[['R2_rank', 'MAPE_rank', 'MAE_rank', 'RMSE_rank']].mean(axis=1)

df_round = df.round(2)
df_sorted = df_round.sort_values('average_rank')
df_sorted[['Block','Window','Model', 'R2', 'MAPE', 'MAE', 'RMSE', 'average_rank']].to_cs
```

In [21]:
```
df_sorted[['Block','Window','Model', 'R2', 'MAPE', 'MAE', 'RMSE', 'average_rank']]
```

Out[21]:

	Block	Window	Model	R2	MAPE	MAE	RMSE	average_rank
13	20	20	lstm	0.02	0.48	0.58	0.50	2.25
9	1	20	lstm	0.13	0.61	0.54	0.46	5.50
10	10	50	lstm	0.03	0.57	0.64	0.54	5.50
15	30	20	lstm	0.02	0.51	0.69	0.62	5.50
3	10	20	rnn	0.03	0.65	0.67	0.55	7.75
5	20	20	rnn	0.03	0.58	0.72	0.63	8.00
8	1	50	lstm	0.17	0.65	0.62	0.48	8.00
1	1	20	rnn	0.16	0.75	0.62	0.49	8.50
11	10	20	lstm	0.05	0.61	0.68	0.55	8.50
6	30	50	rnn	0.03	0.59	0.73	0.66	9.00
2	10	50	rnn	0.04	0.63	0.68	0.57	9.25
4	20	50	rnn	0.03	0.66	0.72	0.63	9.50
7	30	20	rnn	0.03	0.59	0.79	0.72	9.75
0	1	50	rnn	0.19	0.76	0.71	0.53	11.50
12	20	50	lstm	0.04	0.71	0.88	0.78	13.00
14	30	50	lstm	0.04	0.79	1.15	1.02	14.50

In [22]:
```
df
```

Out[22]:

	Files	Block	Window	Model	R2	MAPE	MAE	RMSE	R2_rank
0	predicted_series_rnn_block1_w50.csv	1	50	rnn	0.190359	0.758749	0.711977	0.534166	16.0
1	predicted_series_rnn_block1_w20.csv	1	20	rnn	0.159288	0.748145	0.619932	0.493371	14.0

2	predicted_series_rnn_block10_w50.csv	10	50	rnn	0.041335	0.632985	0.681200	0.569435	11.0
3	predicted_series_rnn_block10_w20.csv	10	20	rnn	0.029319	0.649846	0.665134	0.551375	7.0
4	predicted_series_rnn_block20_w50.csv	20	50	rnn	0.025109	0.664371	0.718309	0.630150	3.0
5	predicted_series_rnn_block20_w20.csv	20	20	rnn	0.027718	0.578451	0.724092	0.627731	5.0
6	predicted_series_rnn_block30_w50.csv	30	50	rnn	0.026868	0.588518	0.731646	0.661124	4.0
7	predicted_series_rnn_block30_w20.csv	30	20	rnn	0.028966	0.588155	0.787813	0.717229	6.0
8	predicted_series_lstm_block1_w50.csv	1	50	lstm	0.166652	0.650745	0.620568	0.483795	15.0
9	predicted_series_lstm_block1_w20.csv	1	20	lstm	0.126069	0.606073	0.539673	0.456365	13.0
10	predicted_series_lstm_block10_w50.csv	10	50	lstm	0.033198	0.573714	0.643142	0.540624	8.0
11	predicted_series_lstm_block10_w20.csv	10	20	lstm	0.047650	0.612029	0.675865	0.548601	12.0
12	predicted_series_lstm_block20_w50.csv	20	50	lstm	0.037887	0.708504	0.879636	0.781686	9.0
13	predicted_series_lstm_block20_w20.csv	20	20	lstm	0.022645	0.482096	0.584278	0.502871	2.0
14	predicted_series_lstm_block30_w50.csv	30	50	lstm	0.039048	0.785556	1.149354	1.016180	10.0
15	predicted_series_lstm_block30_w20.csv	30	20	lstm	0.017123	0.512892	0.694638	0.615955	1.0

In [23]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from math import pi
```

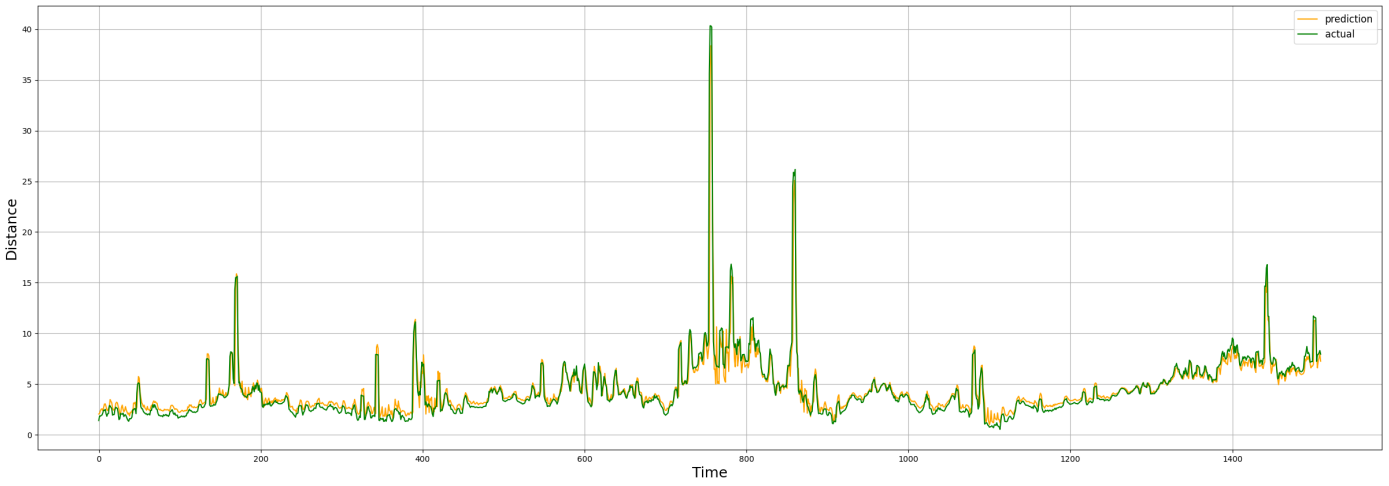
In [24]:
```
import matplotlib.pyplot as plt
import yfinance as yf

def plot(pred,test,stockname,model,block,w):
    plt.figure(figsize = (30,10))
    plt.plot(pred, label = "prediction", c = "orange")
    plt.plot(test, label = "actual", c = "green")
    plt.xlabel("Time", fontsize=18)
    plt.ylabel("Distance", fontsize=18)
    plt.legend(fontsize=12)
    plt.grid(True)
    stockname = stock[1:-1]
    img = "results_graphs/{0}_{1}_block{2}_window{3}".format(stockname,model,block,w)
    plt.savefig(img+'.jpeg', dpi=1200, bbox_inches='tight')
    plt.show()


def plot_actual(stockname, n,save):
    stockname = stockname[1:-1]
    s = stockname.split('_')
    print(s)
    s1 = s[0]+'.NS'
    s2 = s[1]+'.NS'
    stock1 = yf.download(s1, start='2003-01-01', end='2023-12-31')['Adj Close']
    stock2 = yf.download(s2, start='2003-01-01', end='2023-12-31')['Adj Close']

    # Take the last 3701 prices for each stock
    stock1_1 = stock1[-n:]
    stock2_1 = stock2[-n:]

    # Normalize time series individually using standard deviation
    normalized_stock1 = (stock1_1 - stock1_1.mean()) / (stock1_1.std())
    normalized_stock2 = (stock2_1 - stock2_1.mean()) / (stock2_1.std())

    # Plot the normalized time series
```

```
    plt.figure(figsize=(30, 10))
    plt.plot(normalized_stock1, label=s1)
    plt.plot(normalized_stock2, label=s2)
    plt.xlabel('Date', fontsize=18)
    plt.ylabel('Normalized Price', fontsize=18)
    plt.legend(fontsize=12)
    plt.grid(True)
    if save==1:
        img = "results_graphs/{0}_actual".format(stockname)
        plt.savefig(img+'.jpeg',dpi=1200, bbox_inches='tight')
    plt.show()
```

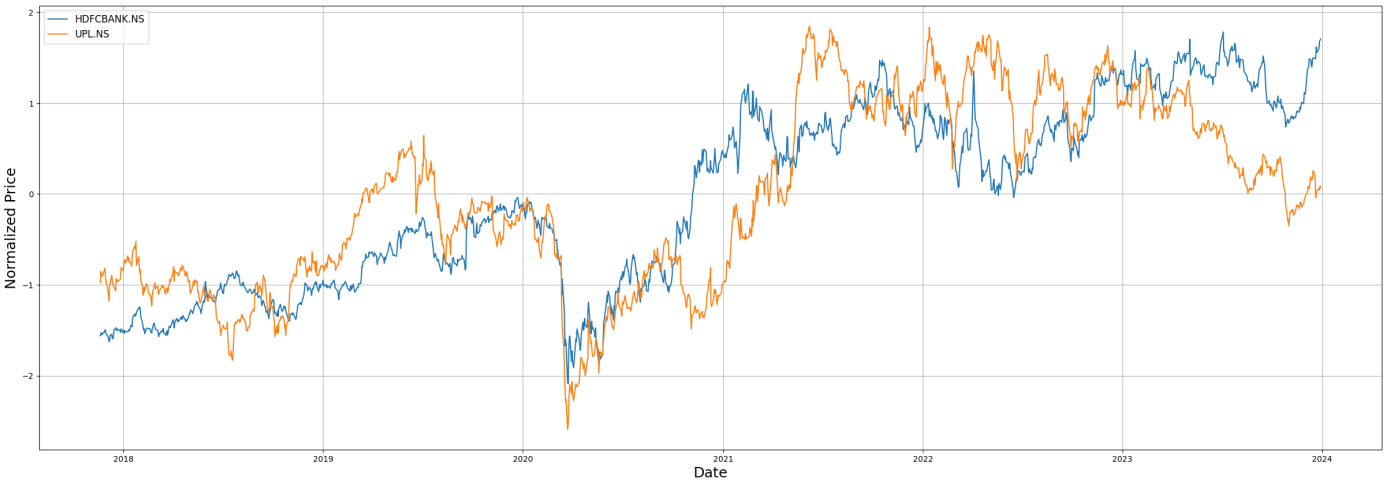
In [25]:

```
stock = '"HDFCBANK_UPL"'
c = 0
for model in models:
    for block in blocks:
        for w in win:
            c+=1
            file = filename.format(model,block,w)
            testfile = "test_data{0}.csv".format(blocks.index(block)+1)
            predicted_series = pd.read_csv(file)
            test_data_file = pd.read_csv(testfile)
            print(file, testfile, w)
            test = test_data_file[stock][int(w):].tolist()
            pred = predicted_series[stock].tolist()
            plot(pred,test, stock, model, block, w)
            if c==1:
                plot_actual(stock, len(test),1)
            else:
                plot_actual(stock, len(test),0)
```

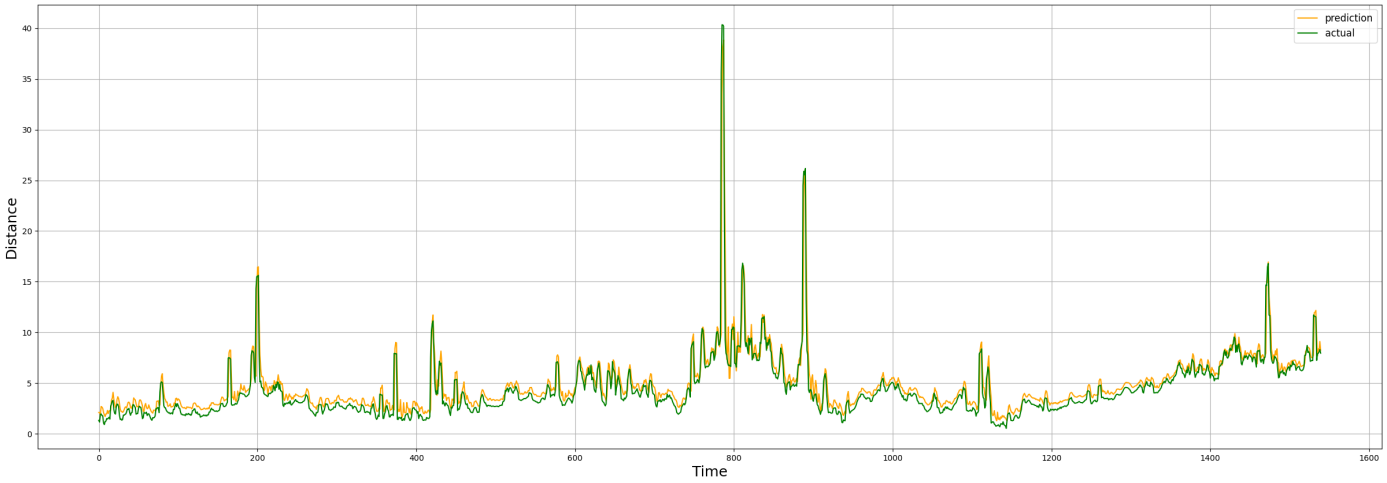
```
predicted_series_rnn_block1_w50.csv test_data1.csv 50
```




```
['HDFCBANK', 'UPL']
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
```

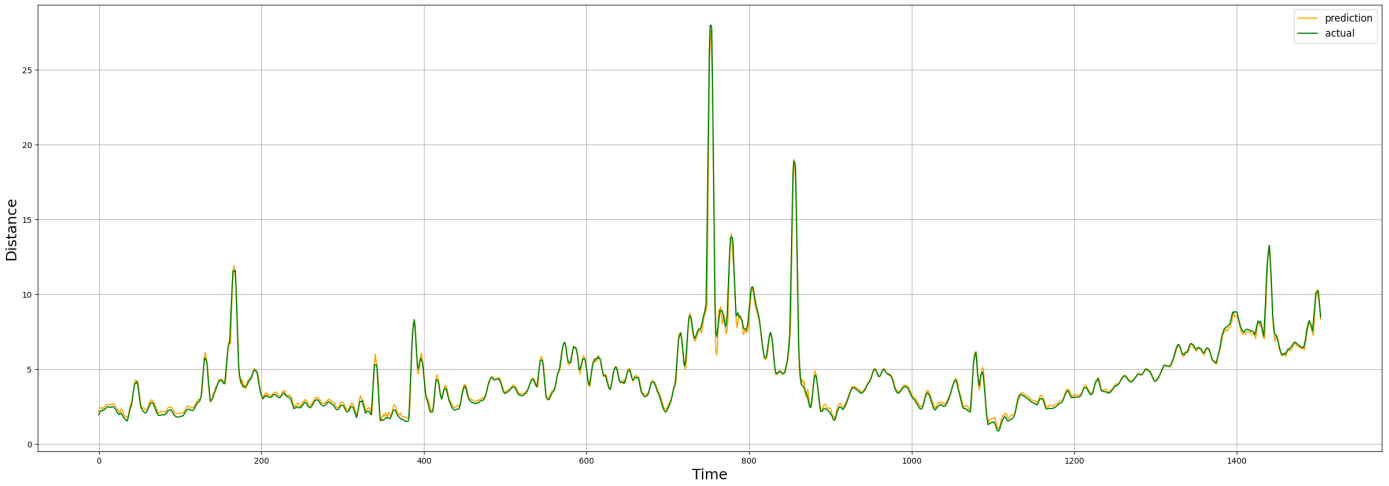


predicted_series_rnn_block1_w20.csv test_data1.csv 20

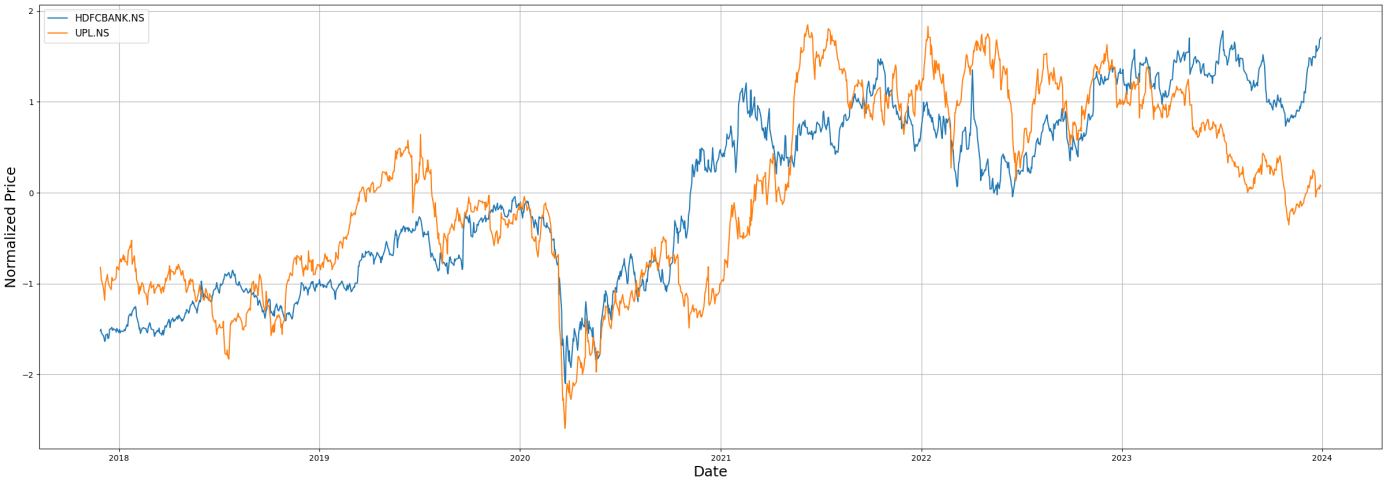


['HDFCBANK', 'UPL']
[**********100%%**********] 1 of 1 completed
[**********100%%**********] 1 of 1 completed

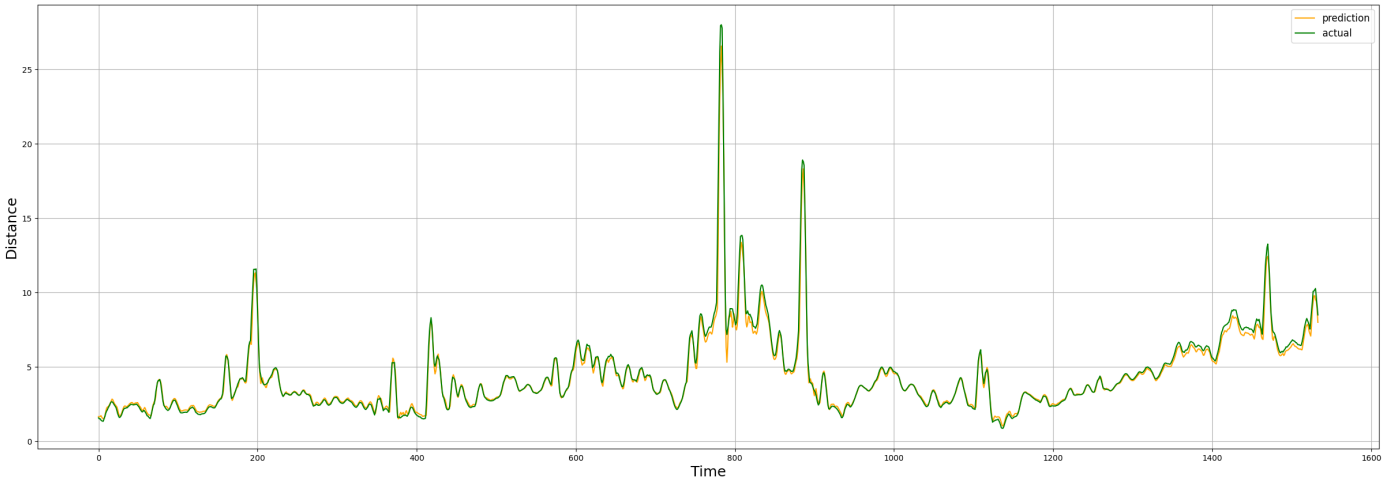
predicted_series_rnn_block10_w50.csv test_data2.csv 50



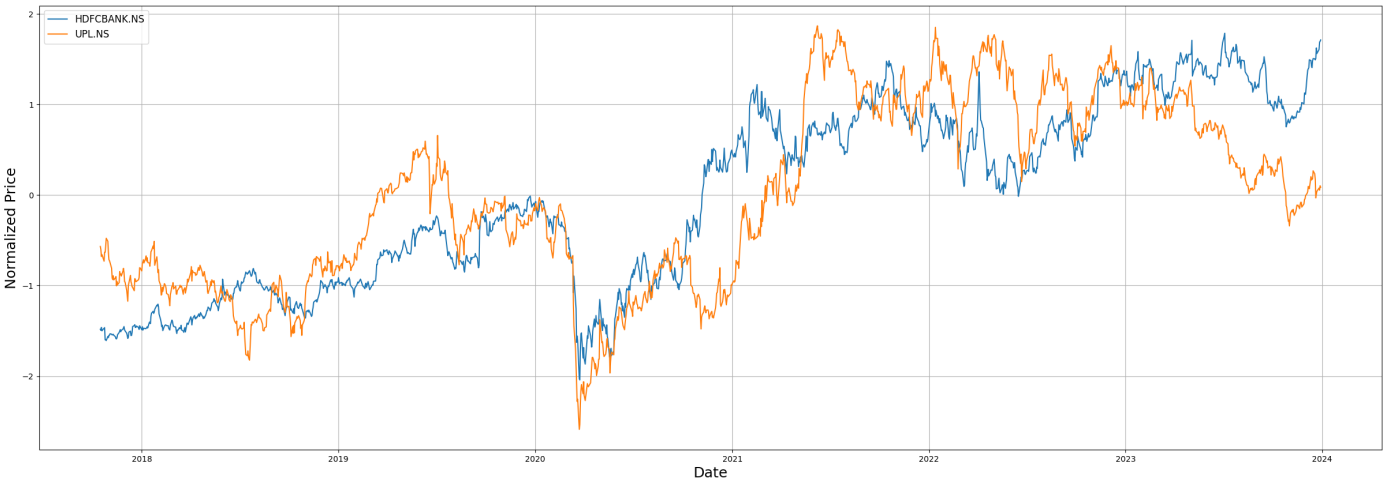

```
['HDFCBANK', 'UPL']
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
```



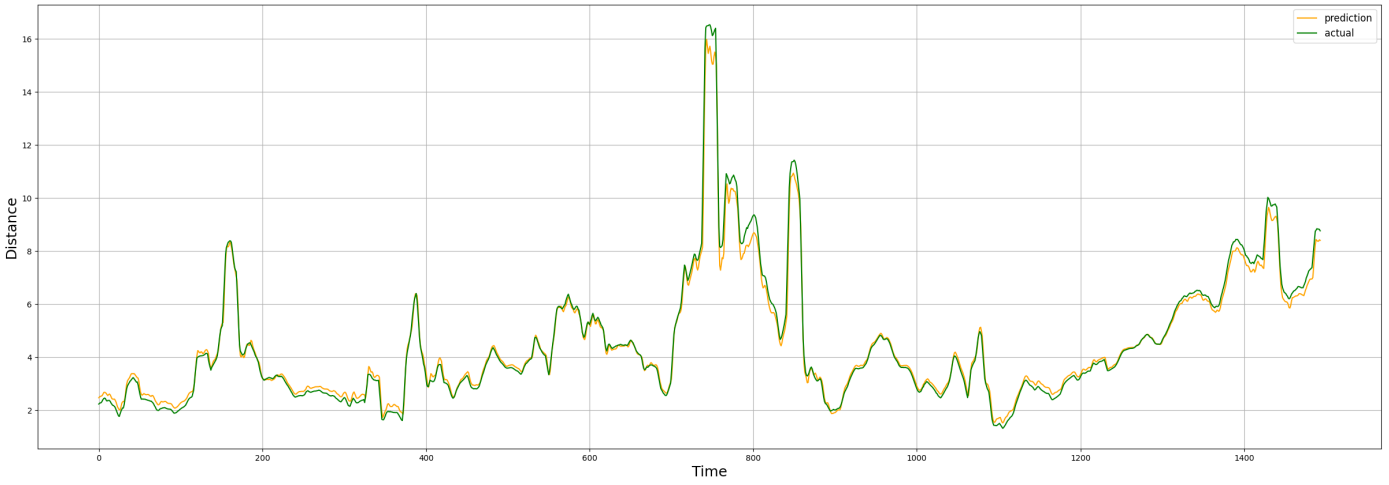

```
predicted_series_rnn_block10_w20.csv test_data2.csv 20
```



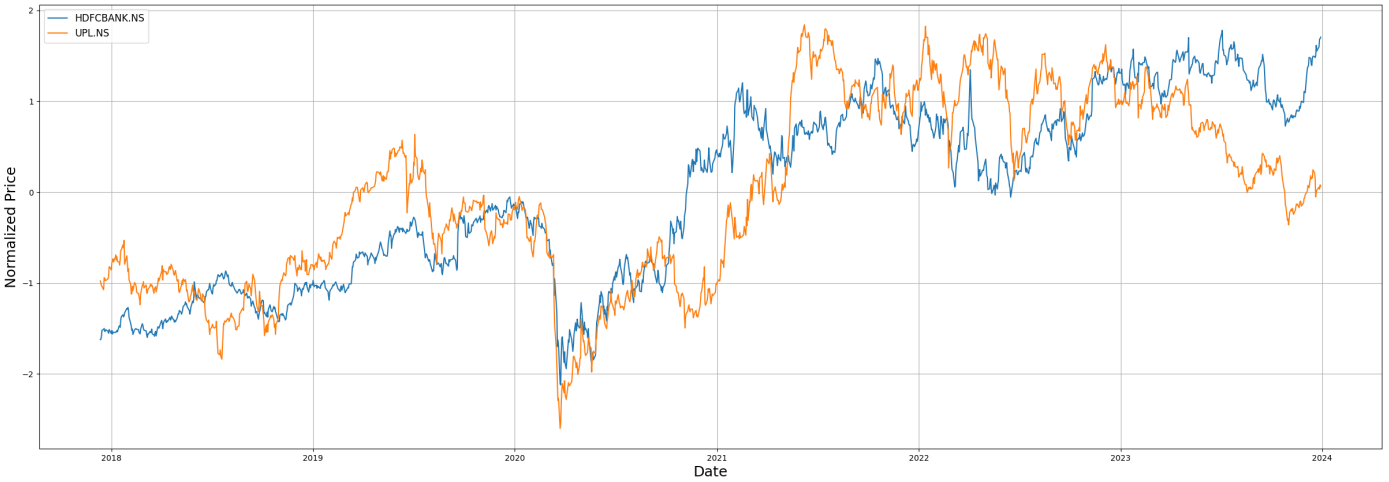

```
['HDFCBANK', 'UPL']
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
```



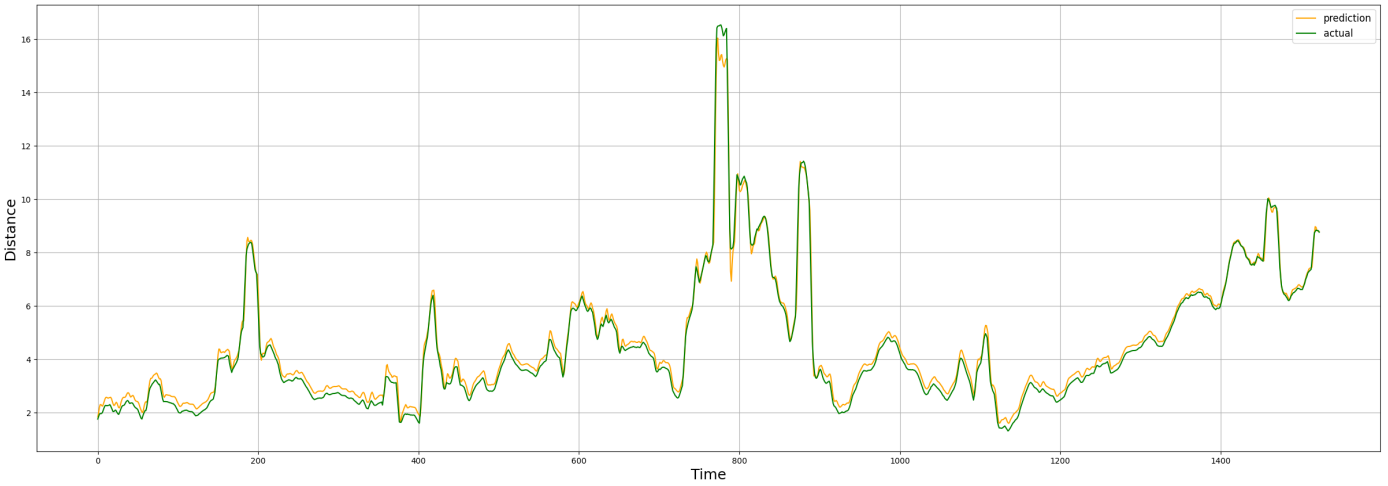
predicted_series_rnn_block20_w50.csv test_data3.csv 50



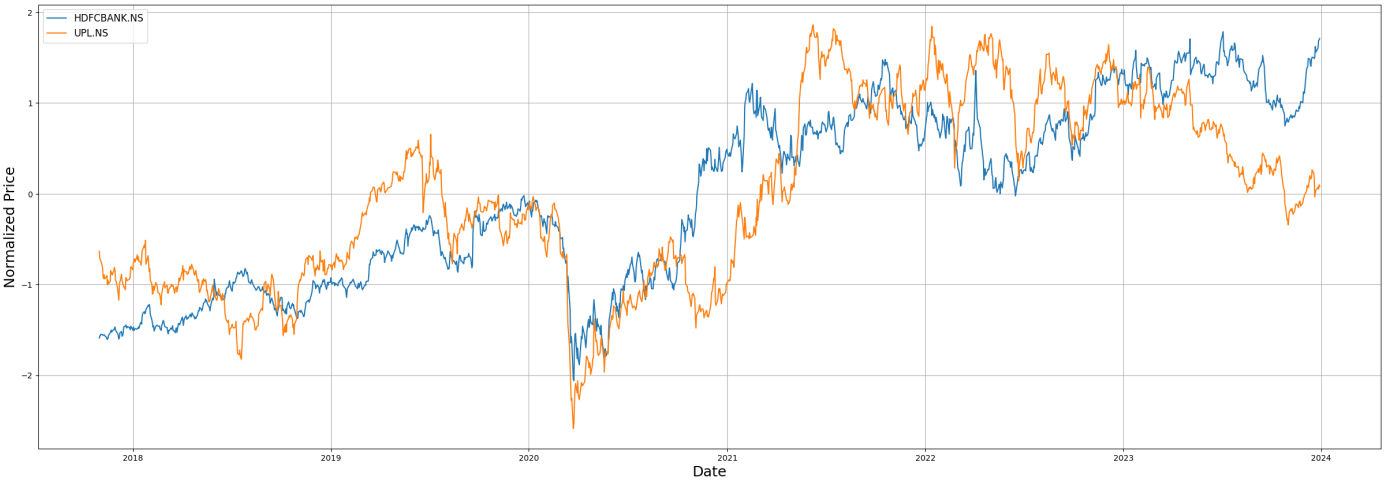
['HDFCBANK', 'UPL']
[**********100%%**********] 1 of 1 completed
[**********100%%**********] 1 of 1 completed



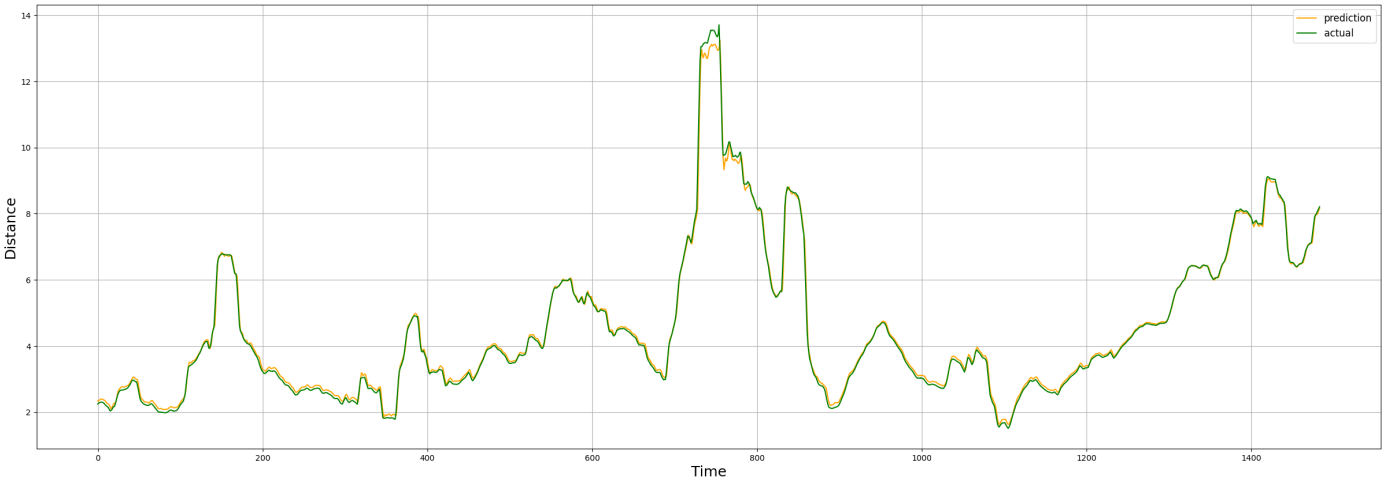
predicted_series_rnn_block20_w20.csv test_data3.csv 20



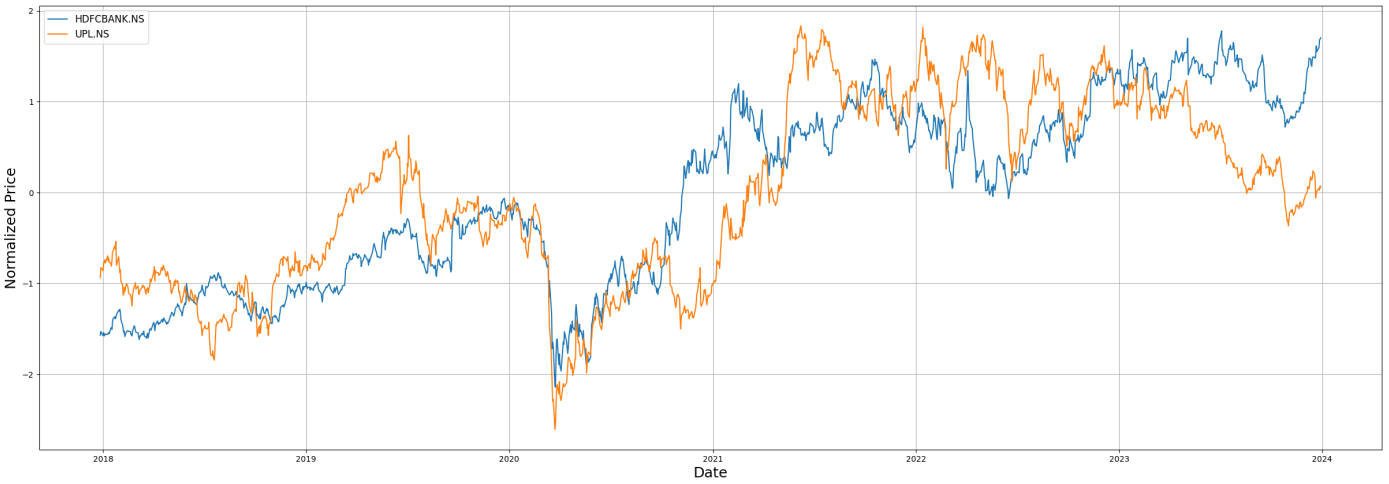

```
['HDFCBANK', 'UPL']
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
```



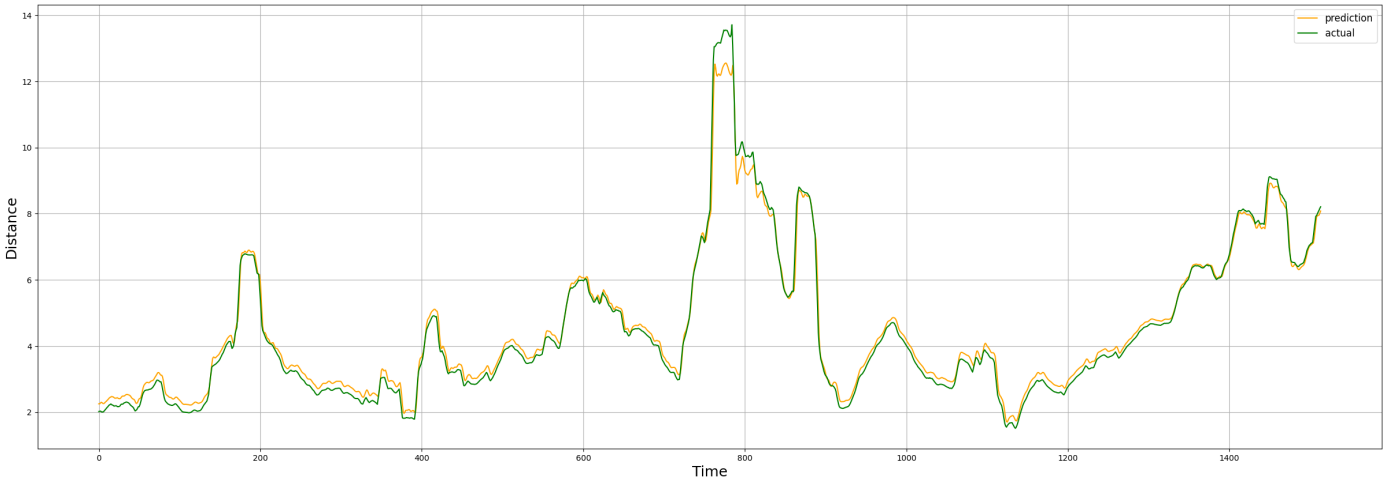

```
predicted_series_rnn_block30_w50.csv test_data4.csv 50
```



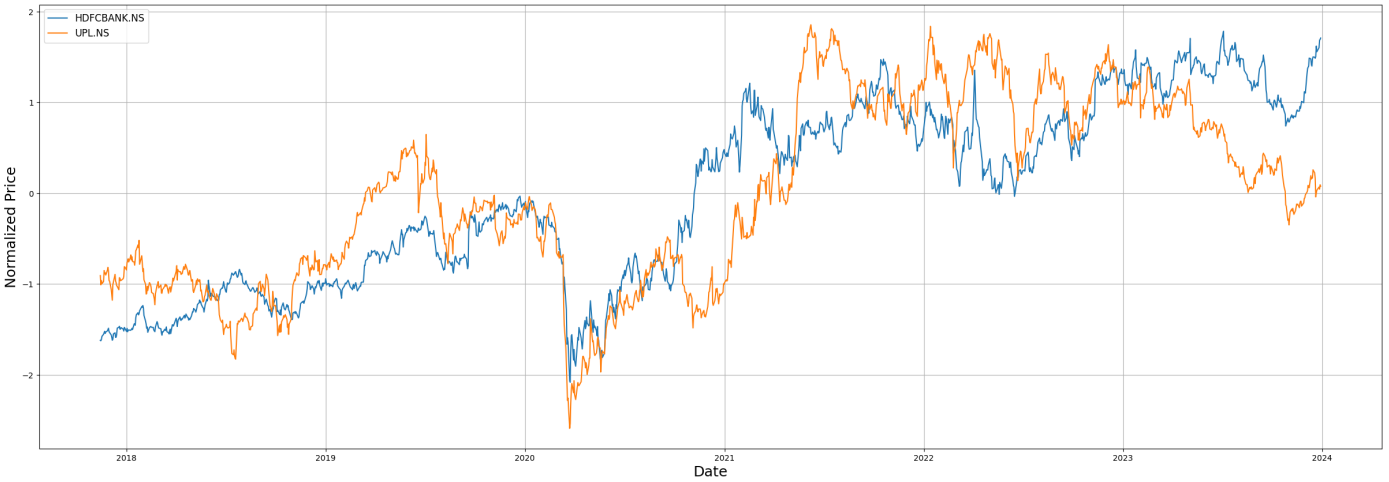

```
['HDFCBANK', 'UPL']
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
```



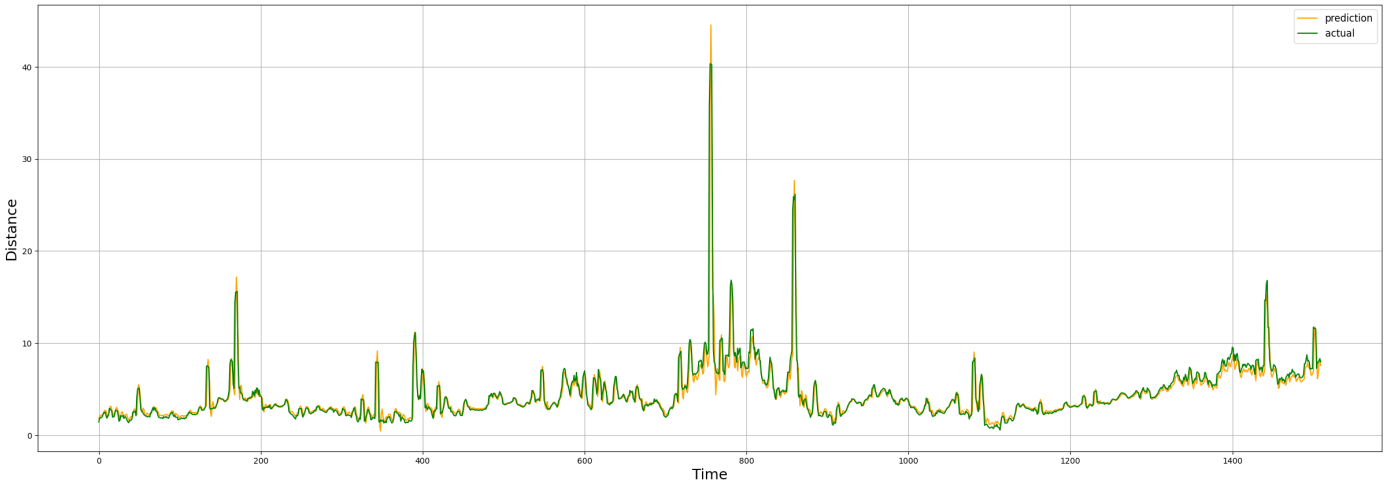
predicted_series_rnn_block30_w20.csv test_data4.csv 20



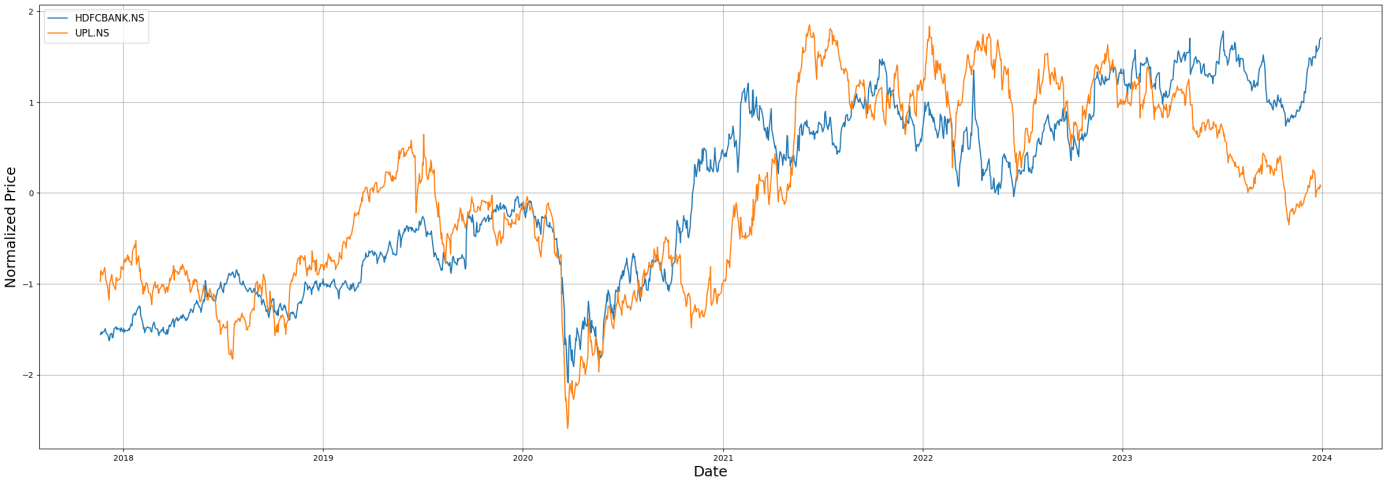
['HDFCBANK', 'UPL']
[**********100%**********] 1 of 1 completed
[**********100%**********] 1 of 1 completed



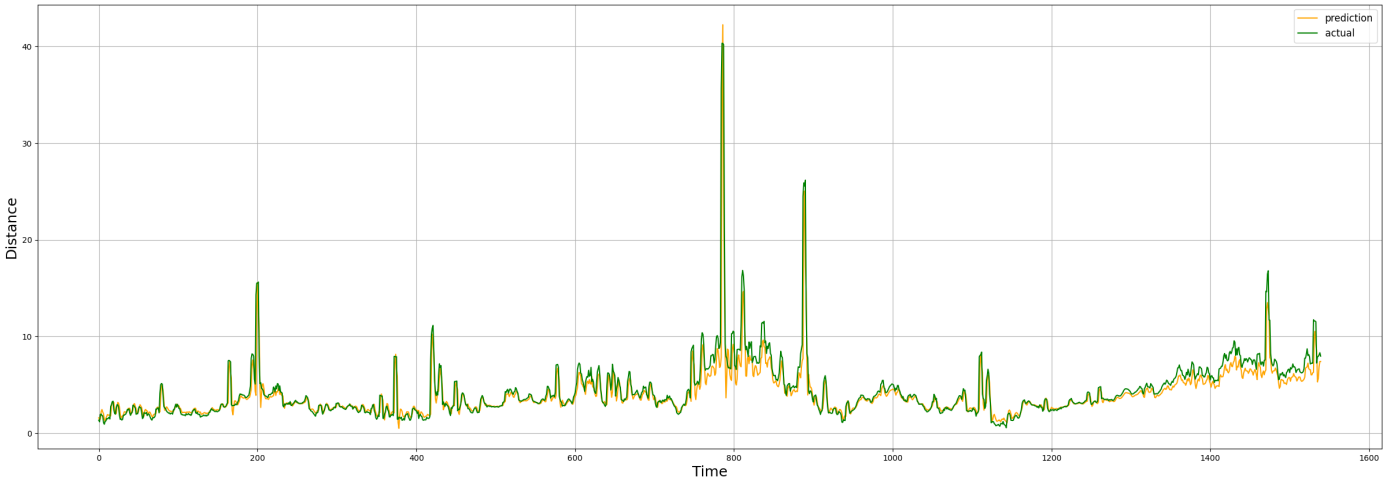
predicted_series_lstm_block1_w50.csv test_data1.csv 50



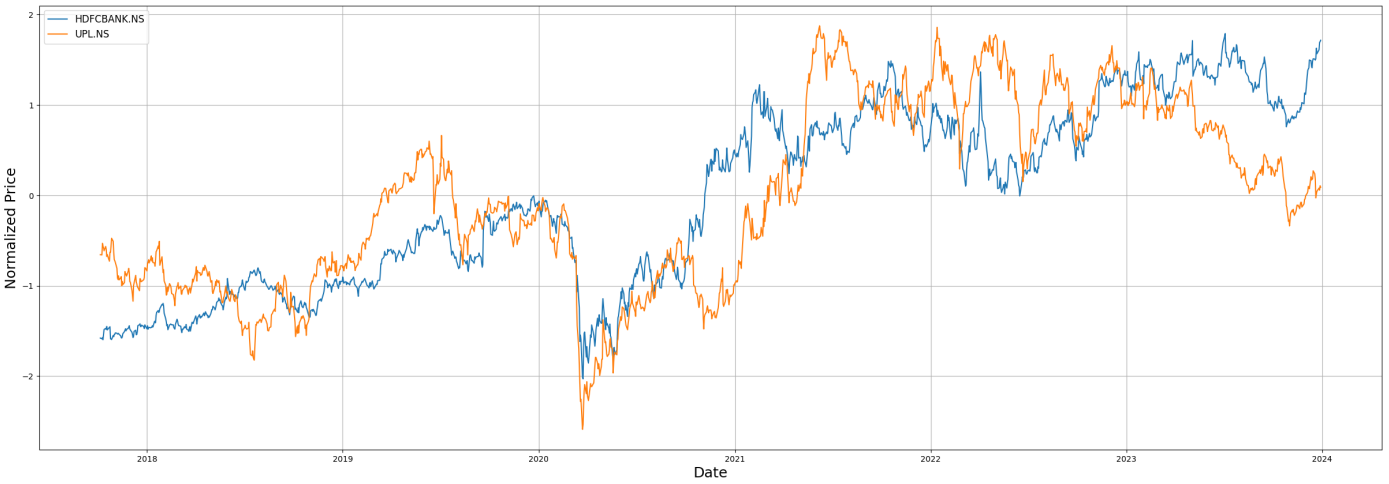

```
['HDFCBANK', 'UPL']
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
```



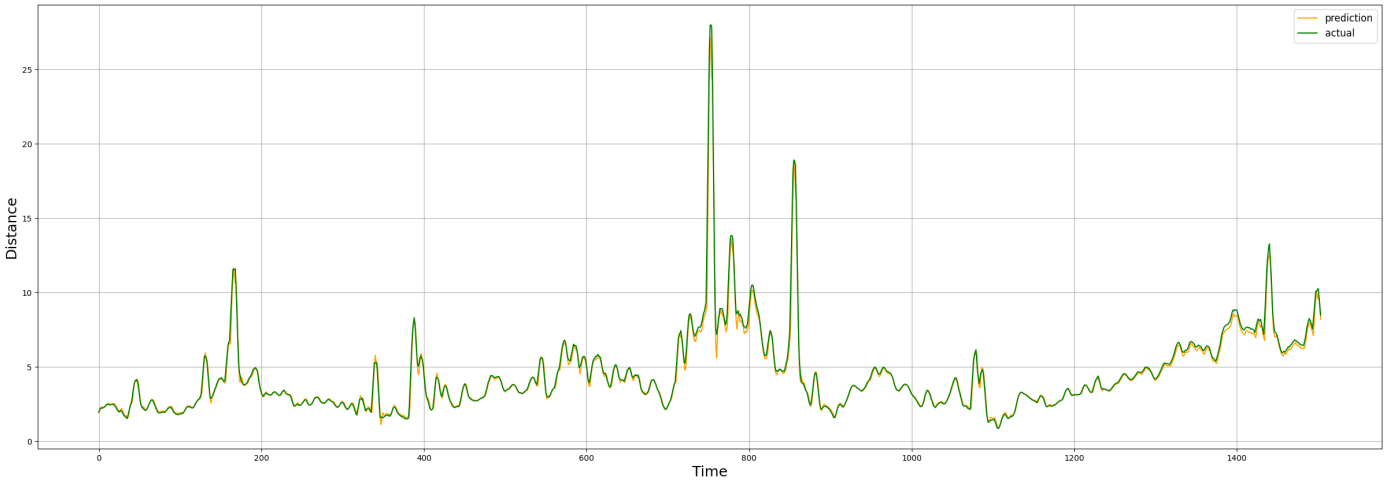

```
predicted_series_lstm_block1_w20.csv test_data1.csv 20
```



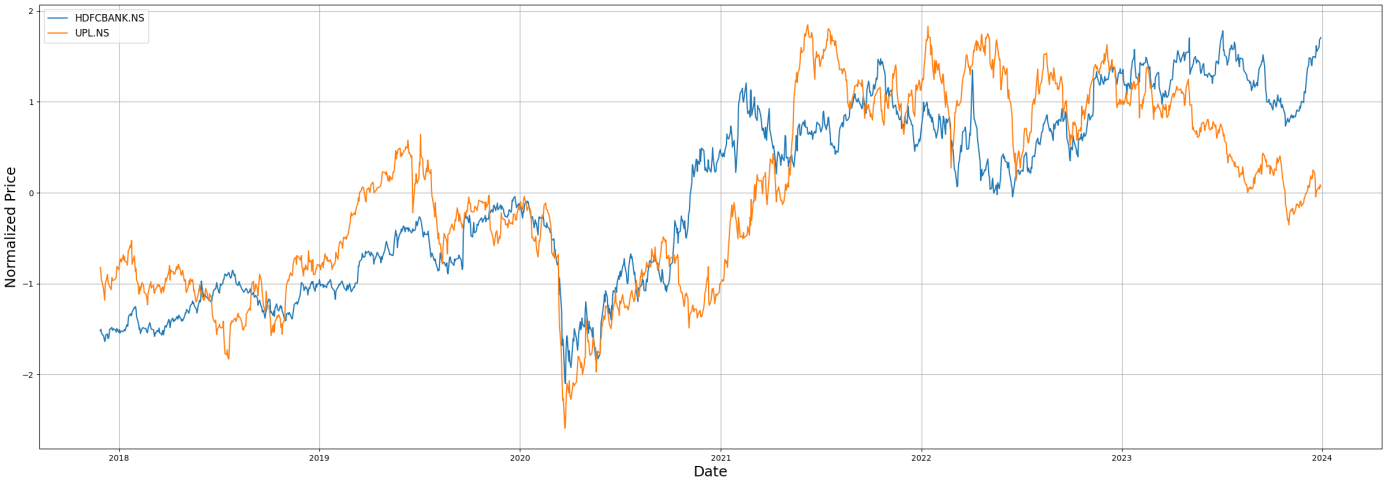

```
['HDFCBANK', 'UPL']
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
```



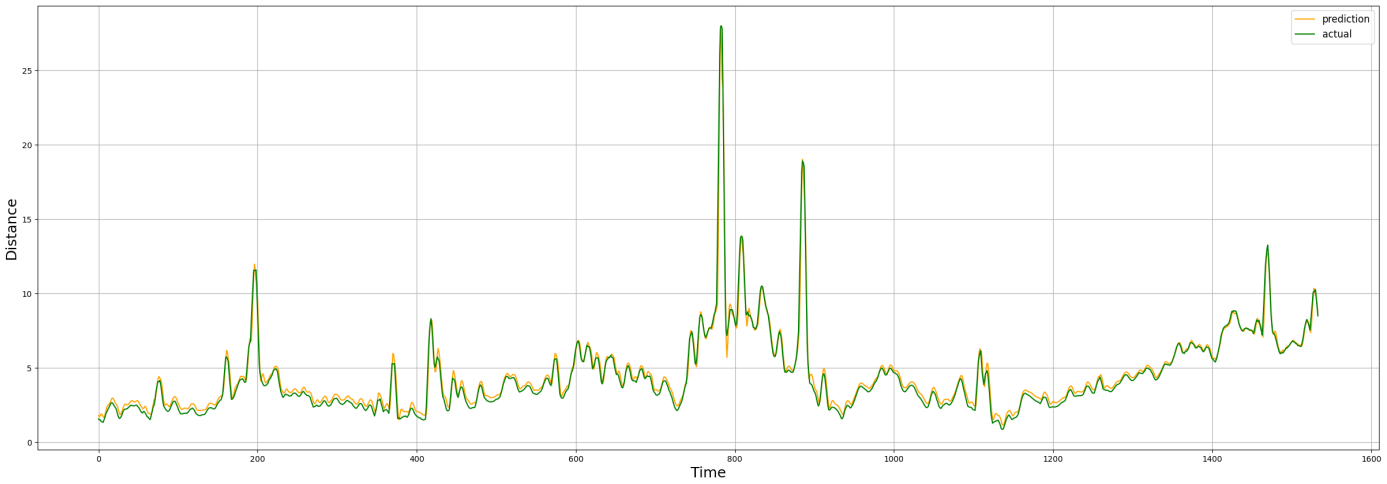
predicted_series_lstm_block10_w50.csv test_data2.csv 50



['HDFCBANK', 'UPL']
[**********100%%**********] 1 of 1 completed
[**********100%%**********] 1 of 1 completed

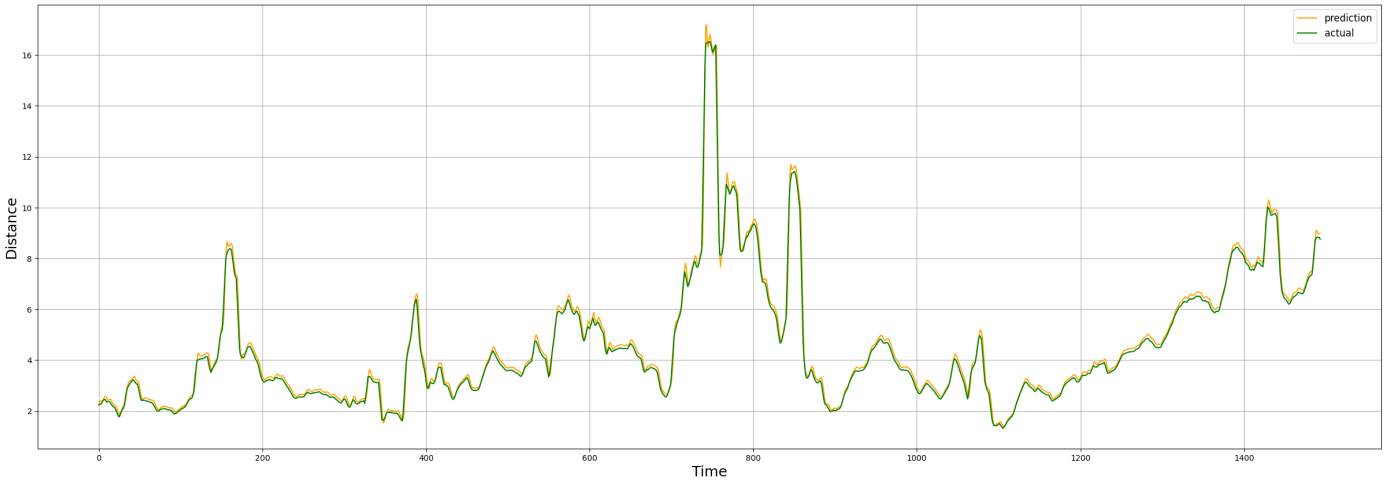


predicted_series_lstm_block10_w20.csv test_data2.csv 20

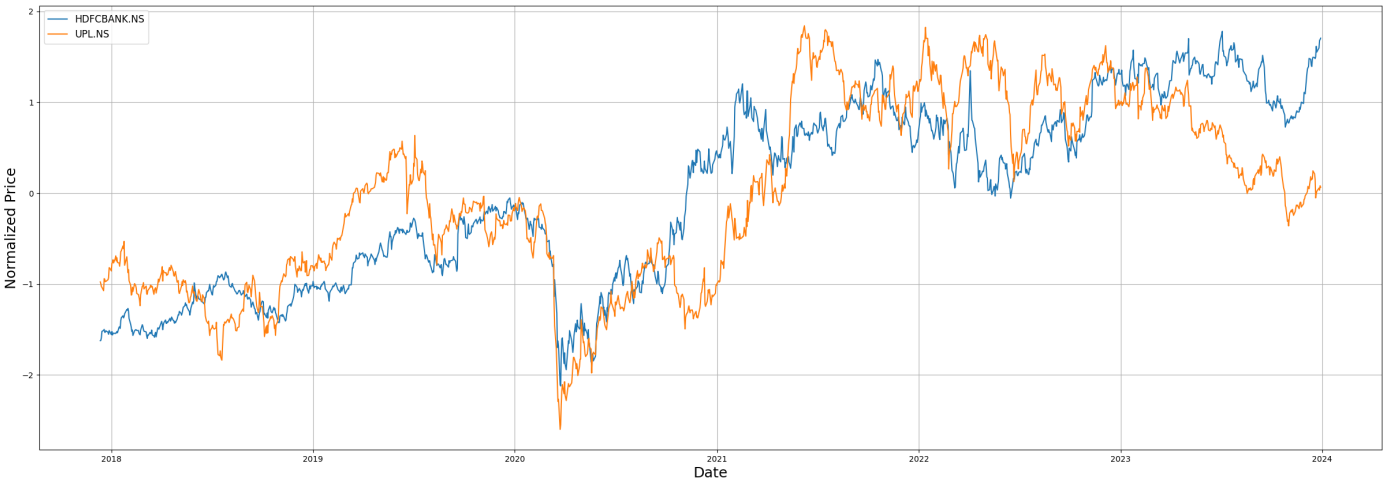



```
['HDFCBANK', 'UPL']
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
```

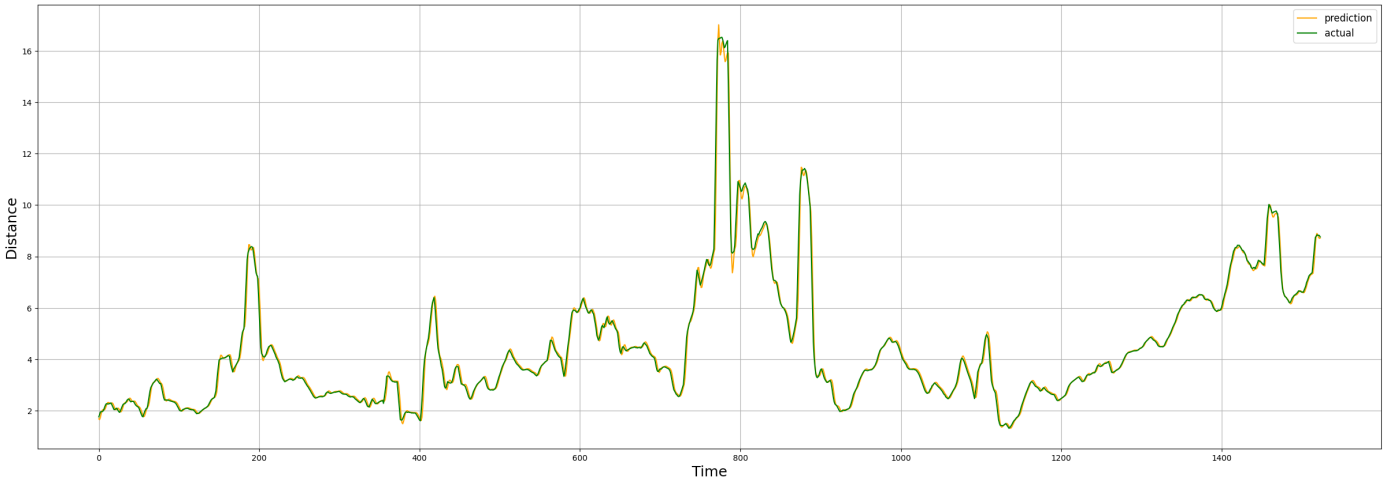

```
predicted_series_lstm_block20_w50.csv test_data3.csv 50
```



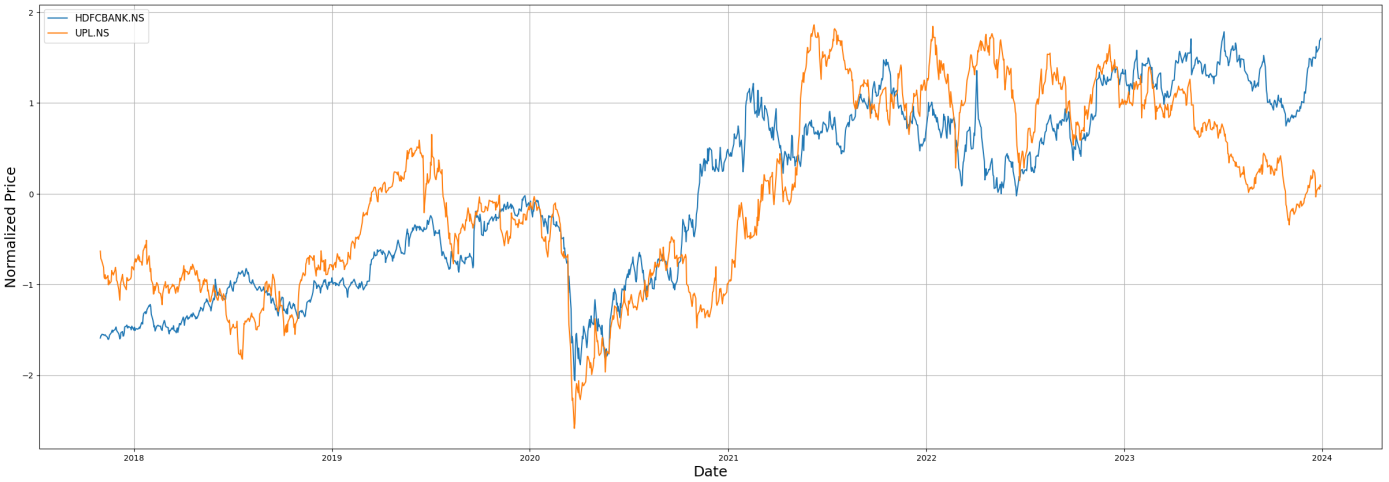

```
['HDFCBANK', 'UPL']
[*********************100%%**********************]  1 of 1 completed
[*********************100%%**********************]  1 of 1 completed
```



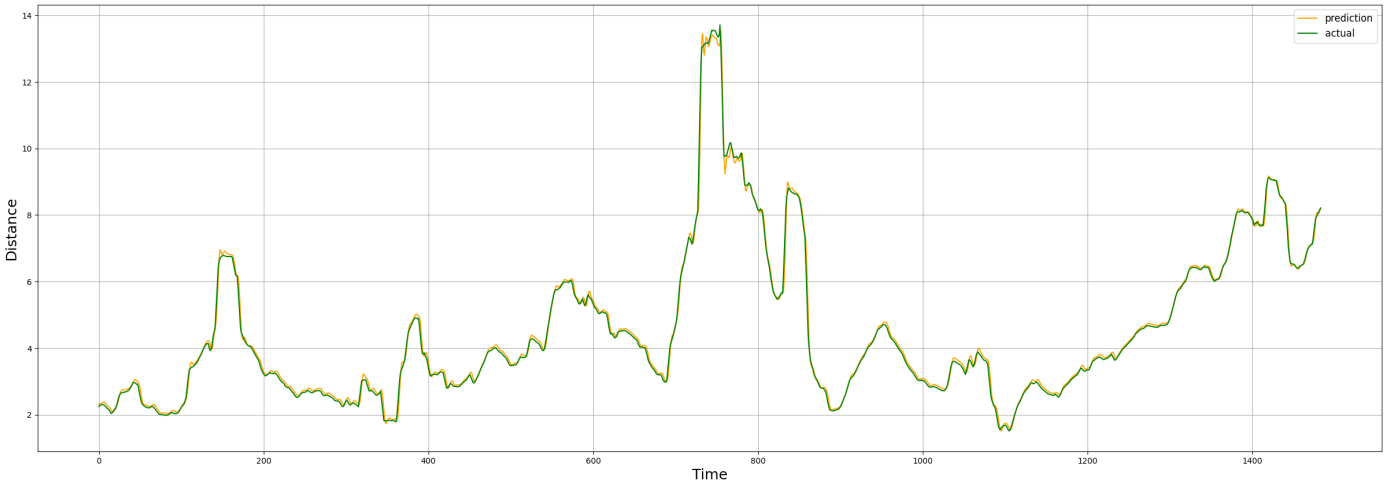
predicted_series_lstm_block20_w20.csv test_data3.csv 20



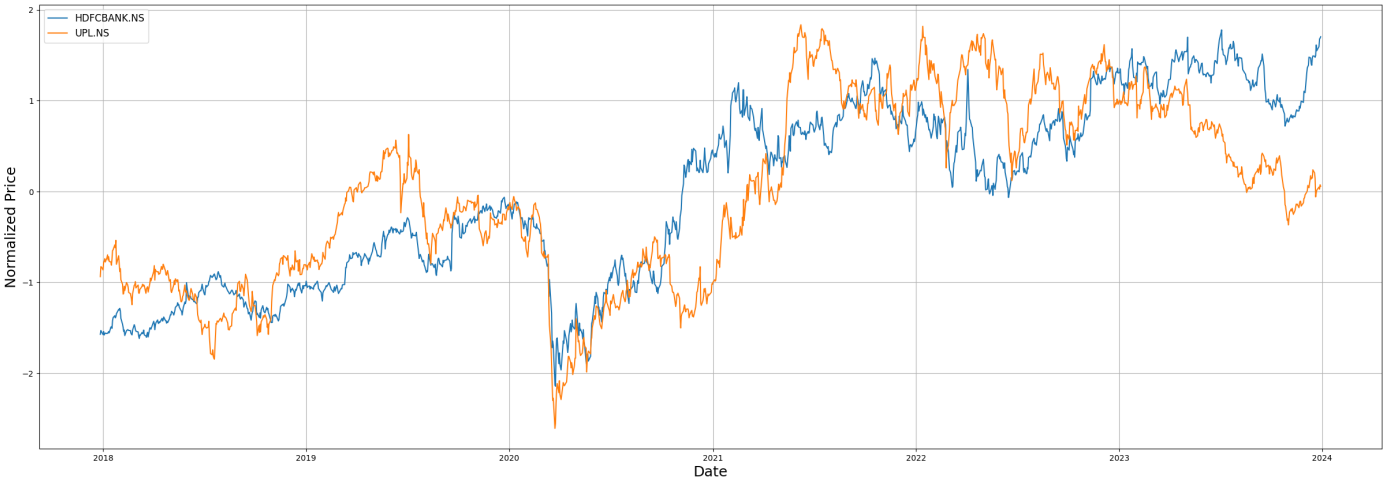
['HDFCBANK', 'UPL']
[**********100%%**********] 1 of 1 completed
[**********100%%**********] 1 of 1 completed



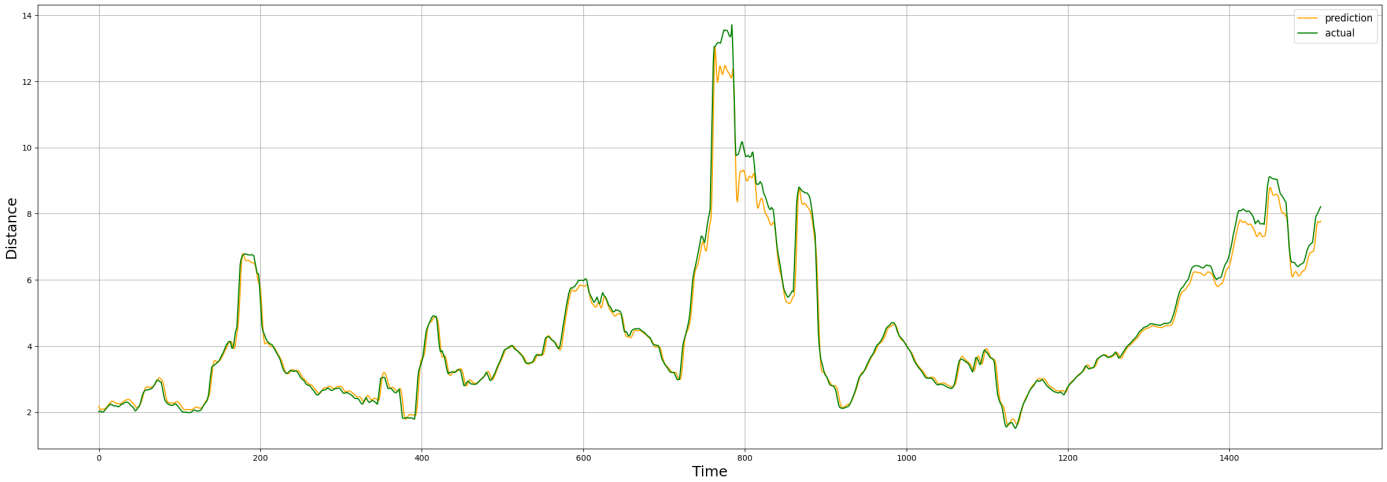
predicted_series_lstm_block30_w50.csv test_data4.csv 50



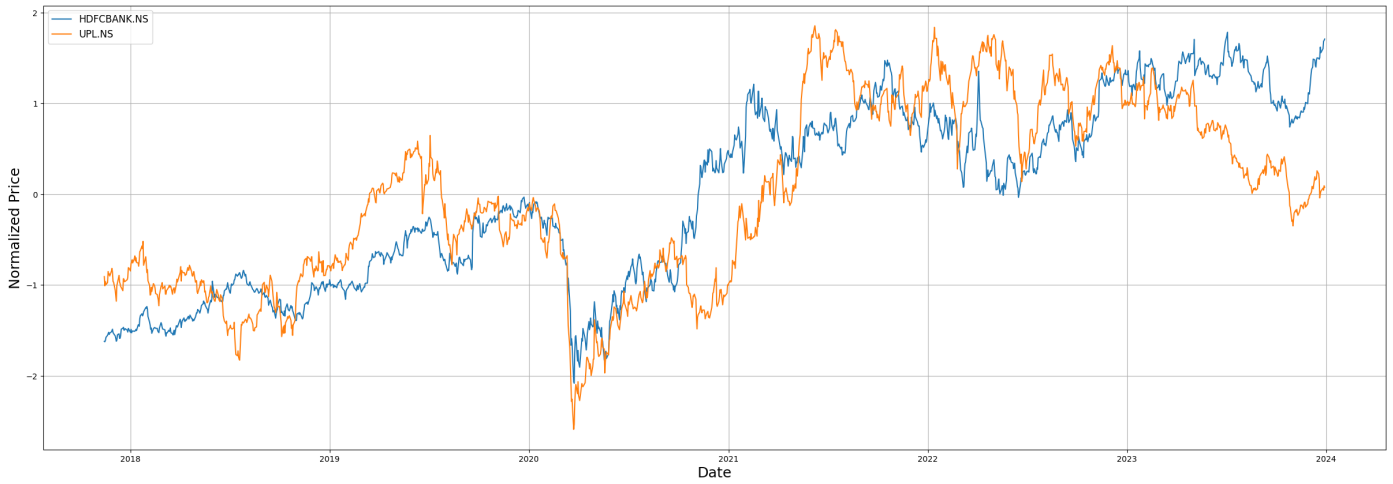

```
['HDFCBANK', 'UPL']
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
```




```
predicted_series_lstm_block30_w20.csv test_data4.csv 20
```




```
['HDFCBANK', 'UPL']
[**********100%%**********]  1 of 1 completed
[**********100%%**********]  1 of 1 completed
```



In [26]:
```
df = pd.DataFrame({
    'Files': files,
    'R2': R2_cv,
    'MAPE': MAPE_cv,
    'MAE': MAE_cv,
    'RMSE': RMSE_cv
})
df['R2_normalized'] = (df['R2'] - df['R2'].min()) / (df['R2'].max() - df['R2'].min())
df['MAPE_normalized'] = (df['MAPE'].max() - df['MAPE']) / (df['MAPE'].max() - df['MAPE']
df['MAE_normalized'] = (df['MAE'].max() - df['MAE']) / (df['MAE'].max() - df['MAE'].min(
df['RMSE_normalized'] = (df['RMSE'].max() - df['RMSE']) / (df['RMSE'].max() - df['RMSE']
```

In [27]:
```
#Classification
```

In [28]:
```
epsilons = pd.read_csv("epsilon_test1.csv")
```

In [29]:
```
test_data_file = pd.read_csv("test_data3.csv")
print(test_data_file.columns)
```

```
Index(['"HDFCBANK_M&M"', '"HDFCBANK_ULTRACEMCO"', '"HDFCBANK_GRASIM"',
       '"HDFCBANK_PIDILITIND"', '"HDFCBANK_LT"', '"HDFCBANK_HINDUNILVR"',
       '"HDFCBANK_ITC"', '"HDFCBANK_RELIANCE"', '"HDFCBANK_ONGC"',
       '"HDFCBANK_UPL"',
       ...
       '"INFY_TATASTEEL"', '"INFY_SUNPHARMA"', '"INFY_CONCOR"',
       '"INFY_BHARTIARTL"', '"TATASTEEL_SUNPHARMA"', '"TATASTEEL_CONCOR"',
       '"TATASTEEL_BHARTIARTL"', '"SUNPHARMA_CONCOR"',
       '"SUNPHARMA_BHARTIARTL"', '"CONCOR_BHARTIARTL"'],
      dtype='object', length=190)
```

In [30]:
```
names = test_data_file.columns.tolist()
```

In [31]:
```
filename = "predicted_series_lstm_block20_w20.csv"
predicted_series_lstm = pd.read_csv(filename)
```

In [32]:
```
#for LSTM
c = 0
for epsilon in epsilons:
    accuracies = []
    precisions_0 = []
    precisions_1 = []
    recalls_0 = []
    recalls_1 = []
    f1_scores_0 = []
    f1_scores_1 = []
    supports_0 = []
    supports_1 = []
```

```
    for name in names:
        epsilon_list = epsilons[epsilon].tolist()
        threshold = epsilon_list[names.index(name)]
        y_pred = (predicted_series_lstm[name] < threshold).astype(int)
        y_true = (test_data_file[name] < threshold).astype(int)
        y_true = y_true[20:]

        # Calculate metrics
        accuracy = accuracy_score(y_true, y_pred)
        precision = precision_score(y_true, y_pred, average=None, zero_division=1)
        recall = recall_score(y_true, y_pred, average=None, zero_division=1)
        f1 = f1_score(y_true, y_pred, average=None, zero_division=1)
        report = classification_report(y_true, y_pred, output_dict=True, zero_division=1

        # Extracting support from classification report
        support_0 = report['0']['support']
        support_1 = report['1']['support'] if '1' in report else 0

        # Append results to lists
        accuracies.append(accuracy)
        precisions_0.append(precision[0])
        recalls_0.append(recall[0])
        f1_scores_0.append(f1[0])
        supports_0.append(support_0)
        if support_1 > 0:
            precisions_1.append(precision[1])
            recalls_1.append(recall[1])
            f1_scores_1.append(f1[1])
            supports_1.append(support_1)
        else:
            precisions_1.append(None)
            recalls_1.append(None)
            f1_scores_1.append(None)
            supports_1.append(0)

    per_class1 = []
    for i in range(len(supports_1)):
        per1 = (supports_1[i]/(supports_1[i] + supports_0[i]))
        per_class1.append(per1)

    df_results = pd.DataFrame({
        "stock_pair": names,
        "accuracy": accuracies,
        "precision": precisions_1,
        "recall": recalls_1,
        "f1": f1_scores_1,
        "%class 1": per_class1
    })

    # Save dataframe to CSV
    df_results.to_csv('lstm_block20_w20_classification_results_{0}.csv'.format(epsilon),
```

In [33]:
```
from prettytable import PrettyTable
dfs = []

for epsilon in epsilons:
    df = pd.read_csv(f'lstm_block20_w20_classification_results_{epsilon}.csv')
    df['epsilon'] = epsilon
    dfs.append(df)

df_results = pd.concat(dfs, ignore_index=True)
```

In [34]:
```
summary = df_results.groupby('epsilon').agg(['mean', 'std', 'min', 'max', 'median'])
summary.columns = ['_'.join(col).strip() for col in summary.columns.values]
```

```
C:\Users\sanja\AppData\Local\Temp\ipykernel_48892\3977975528.py:1: FutureWarning: ['stoc
k_pair'] did not aggregate successfully. If any error is raised this will raise in a fut
ure version of pandas. Drop these columns/ops to avoid this warning.
  summary = df_results.groupby('epsilon').agg(['mean', 'std', 'min', 'max', 'median'])
```

In [35]:
```python
def generate_summary_table(df, epsilon):
    # Filter for the specific epsilon value
    epsilon_df = df[df['epsilon'] == epsilon]
    # Select only numeric columns for aggregation
    numeric_cols = epsilon_df.select_dtypes(include=['number']).columns
    # Calculate summary statistics
    summary = epsilon_df[numeric_cols].agg(['mean', 'std', 'min', 'max', 'median']).T
    summary.columns = ['Mean', 'Std Dev', 'Min', 'Max', 'Median']

    return summary
```

In [36]:
```python
pd.set_option('display.float_format', '{:.2f}'.format)

for epsilon in epsilons:
    summary_table = generate_summary_table(df_results, epsilon)
    print(f"\nSummary Statistics for Epsilon = {epsilon}")
    display(summary_table)
    summary_table_rounded = summary_table.round(2)
    print(summary_table['Mean']['accuracy'])
    summary_table_rounded.to_csv('lstm_block20_w20_summary_results_{0}.csv'.format(epsil
```

```
Summary Statistics for Epsilon = RR 1% (Testing)
```

	Mean	Std Dev	Min	Max	Median
accuracy	1.00	0.00	0.97	1.00	1.00
precision	0.91	0.16	0.57	1.00	1.00
recall	0.45	0.41	0.00	1.00	0.54
f1	0.48	0.41	0.00	0.90	0.63
%class 1	0.00	0.00	0.00	0.03	0.00

```
0.9996822765575356

Summary Statistics for Epsilon = RR 5%(Testing)
```

	Mean	Std Dev	Min	Max	Median
accuracy	1.00	0.01	0.94	1.00	1.00
precision	0.81	0.31	0.00	1.00	0.97
recall	0.40	0.36	0.00	1.00	0.37
f1	0.55	0.36	0.00	1.00	0.62
%class 1	0.01	0.01	0.00	0.09	0.00

```
0.9971025003453515

Summary Statistics for Epsilon = RR 10%(Testing)
```

	Mean	Std Dev	Min	Max	Median
accuracy	0.99	0.01	0.96	1.00	1.00
precision	0.92	0.13	0.25	1.00	0.99
recall	0.52	0.37	0.00	1.00	0.65
f1	0.56	0.37	0.00	1.00	0.74

%class 1	0.02	0.03	0.00	0.15	0.00

```
0.9942602569415665
```

```
Summary Statistics for Epsilon = RR 20%(Testing)
```

	Mean	Std Dev	Min	Max	Median
accuracy	0.99	0.02	0.86	1.00	0.99
precision	0.91	0.16	0.00	1.00	0.97
recall	0.70	0.32	0.00	1.00	0.84
f1	0.74	0.29	0.00	1.00	0.88
%class 1	0.05	0.06	0.00	0.27	0.02

```
0.9882200580190635
```

```
Summary Statistics for Epsilon = RR 25%_Testing
```

	Mean	Std Dev	Min	Max	Median
accuracy	0.98	0.02	0.86	1.00	0.99
precision	0.94	0.09	0.50	1.00	0.97
recall	0.71	0.31	0.00	1.00	0.85
f1	0.75	0.29	0.00	1.00	0.89
%class 1	0.07	0.08	0.00	0.33	0.05

```
0.9845351567896118
```

```
Summary Statistics for Epsilon = RR 30%_Testing
```

	Mean	Std Dev	Min	Max	Median
accuracy	0.98	0.02	0.88	1.00	0.99
precision	0.94	0.10	0.19	1.00	0.97
recall	0.79	0.24	0.00	1.00	0.90
f1	0.83	0.20	0.00	1.00	0.91
%class 1	0.10	0.10	0.00	0.41	0.06

```
0.982836027075563
```

```
Summary Statistics for Epsilon = RR 35%_Testing
```

	Mean	Std Dev	Min	Max	Median
accuracy	0.98	0.02	0.88	1.00	0.99
precision	0.94	0.12	0.00	1.00	0.97
recall	0.80	0.27	0.00	1.00	0.93
f1	0.83	0.24	0.00	1.00	0.93
%class 1	0.13	0.12	0.00	0.51	0.10

```
0.9808606161071973
```

```
Summary Statistics for Epsilon = RR 40%_Testing
```

	Mean	Std Dev	Min	Max	Median
accuracy	0.98	0.02	0.90	1.00	0.99

precision	0.94	0.12	0.00	1.00	0.98
recall	0.83	0.25	0.00	1.00	0.94
f1	0.86	0.22	0.00	1.00	0.94
%class 1	0.16	0.14	0.00	0.57	0.13

```
0.9811576184555879
```

```
Summary Statistics for Epsilon = RR 45%_Testing
```

	Mean	Std Dev	Min	Max	Median
accuracy	0.98	0.02	0.90	1.00	0.98
precision	0.95	0.09	0.25	1.00	0.97
recall	0.86	0.22	0.00	1.00	0.94
f1	0.88	0.20	0.00	1.00	0.95
%class 1	0.19	0.16	0.00	0.63	0.17

```
0.9790406133443845
```

In [37]:
```
def calculate_cv(summary_table):
    summary_table['CV'] = summary_table['Std Dev'] / summary_table['Mean']
    return summary_table

pd.set_option('display.float_format', '{:.4f}'.format)

# Initialize an empty DataFrame to store CV values for each epsilon
cv_df = pd.DataFrame()

for epsilon in epsilons:
    summary_table = generate_summary_table(df_results, epsilon)
    summary_table = calculate_cv(summary_table)
    print(f"\nSummary Statistics for Epsilon = {epsilon}")
    display(summary_table)

    # Extract the CV values and store them in a DataFrame
    cv_values = summary_table['CV'].to_frame().T
    cv_values['epsilon'] = epsilon
    cv_df = pd.concat([cv_df, cv_values], ignore_index=True)

# Set 'epsilon' as the index
cv_df.set_index('epsilon', inplace=True)

# Drop the first 3 rows
cv_df = cv_df.iloc[3:]

# Display the CV DataFrame
print("\nCoefficient of Variation (CV) for Each Epsilon")
display(cv_df)
```

```
Summary Statistics for Epsilon = RR 1% (Testing)
```

	Mean	Std Dev	Min	Max	Median	CV
accuracy	0.9997	0.0025	0.9665	1.0000	1.0000	0.0025
precision	0.9062	0.1570	0.5714	1.0000	1.0000	0.1733
recall	0.4550	0.4058	0.0000	1.0000	0.5357	0.8919
f1	0.4817	0.4144	0.0000	0.9000	0.6303	0.8602
%class 1	0.0004	0.0027	0.0000	0.0335	0.0000	6.8160

Summary Statistics for Epsilon = RR 5%(Testing)

	Mean	Std Dev	Min	Max	Median	CV
accuracy	0.9971	0.0077	0.9403	1.0000	1.0000	0.0077
precision	0.8088	0.3101	0.0000	1.0000	0.9713	0.3833
recall	0.4042	0.3621	0.0000	1.0000	0.3670	0.8959
f1	0.5498	0.3600	0.0000	1.0000	0.6216	0.6548
%class 1	0.0056	0.0149	0.0000	0.0879	0.0000	2.6749

Summary Statistics for Epsilon = RR 10%(Testing)

	Mean	Std Dev	Min	Max	Median	CV
accuracy	0.9943	0.0086	0.9560	1.0000	1.0000	0.0086
precision	0.9243	0.1312	0.2500	1.0000	0.9917	0.1420
recall	0.5232	0.3726	0.0000	1.0000	0.6548	0.7122
f1	0.5596	0.3678	0.0000	1.0000	0.7429	0.6574
%class 1	0.0170	0.0305	0.0000	0.1483	0.0000	1.7981

Summary Statistics for Epsilon = RR 20%(Testing)

	Mean	Std Dev	Min	Max	Median	CV
accuracy	0.9882	0.0168	0.8648	1.0000	0.9938	0.0170
precision	0.9129	0.1626	0.0000	1.0000	0.9674	0.1781
recall	0.6976	0.3230	0.0000	1.0000	0.8443	0.4631
f1	0.7446	0.2916	0.0000	1.0000	0.8832	0.3916
%class 1	0.0503	0.0648	0.0000	0.2677	0.0233	1.2894

Summary Statistics for Epsilon = RR 25%_Testing

	Mean	Std Dev	Min	Max	Median	CV
accuracy	0.9845	0.0206	0.8615	1.0000	0.9915	0.0210
precision	0.9411	0.0897	0.5000	1.0000	0.9684	0.0954
recall	0.7135	0.3134	0.0000	1.0000	0.8542	0.4392
f1	0.7519	0.2902	0.0000	0.9951	0.8872	0.3860
%class 1	0.0721	0.0829	0.0000	0.3307	0.0456	1.1495

Summary Statistics for Epsilon = RR 30%_Testing

	Mean	Std Dev	Min	Max	Median	CV
accuracy	0.9828	0.0208	0.8753	1.0000	0.9895	0.0211
precision	0.9400	0.0974	0.1875	1.0000	0.9686	0.1036
recall	0.7943	0.2385	0.0000	1.0000	0.8994	0.3003
f1	0.8310	0.2007	0.0000	0.9951	0.9139	0.2416
%class 1	0.0979	0.1029	0.0000	0.4147	0.0650	1.0507

Summary Statistics for Epsilon = RR 35%_Testing

	Mean	Std Dev	Min	Max	Median	CV
accuracy	0.9809	0.0206	0.8825	1.0000	0.9875	0.0210

precision	0.9421	0.1198	0.0000	1.0000	0.9714	0.1272
recall	0.7982	0.2688	0.0000	1.0000	0.9281	0.3367
f1	0.8318	0.2355	0.0000	1.0000	0.9282	0.2831
%class 1	0.1269	0.1220	0.0000	0.5138	0.1004	0.9617

Summary Statistics for Epsilon = RR 40%_Testing

	Mean	Std Dev	Min	Max	Median	CV
accuracy	0.9812	0.0193	0.8990	1.0000	0.9869	0.0196
precision	0.9436	0.1245	0.0000	1.0000	0.9752	0.1320
recall	0.8304	0.2544	0.0000	1.0000	0.9372	0.3063
f1	0.8648	0.2169	0.0000	1.0000	0.9434	0.2508
%class 1	0.1564	0.1405	0.0000	0.5728	0.1306	0.8985

Summary Statistics for Epsilon = RR 45%_Testing

	Mean	Std Dev	Min	Max	Median	CV
accuracy	0.9790	0.0194	0.8963	1.0000	0.9833	0.0198
precision	0.9488	0.0868	0.2500	1.0000	0.9747	0.0915
recall	0.8572	0.2238	0.0000	1.0000	0.9431	0.2611
f1	0.8762	0.1989	0.0000	0.9976	0.9451	0.2270
%class 1	0.1864	0.1585	0.0000	0.6293	0.1703	0.8502

Coefficient of Variation (CV) for Each Epsilon

epsilon	accuracy	precision	recall	f1	%class 1
RR 20%(Testing)	0.0170	0.1781	0.4631	0.3916	1.2894
RR 25%_Testing	0.0210	0.0954	0.4392	0.3860	1.1495
RR 30%_Testing	0.0211	0.1036	0.3003	0.2416	1.0507
RR 35%_Testing	0.0210	0.1272	0.3367	0.2831	0.9617
RR 40%_Testing	0.0196	0.1320	0.3063	0.2508	0.8985
RR 45%_Testing	0.0198	0.0915	0.2611	0.2270	0.8502

In [38]:
```
rank_df = cv_df.rank(ascending=True)

# Calculate average rank for each epsilon
cv_df['average_rank'] = rank_df.mean(axis=1)

# Sort epsilons by average rank
ranked_epsilons = cv_df.sort_values(by='average_rank')

# Display the ranked epsilons
print("Ranked Epsilons Based on Average Rank")
display(ranked_epsilons)
ranked_epsilons_rounded = ranked_epsilons.round(2)
ranked_epsilons_rounded.to_csv('ranked_epsilons.csv')
```

Ranked Epsilons Based on Average Rank

	accuracy	precision	recall	f1	%class 1	average_rank

epsilon						
RR 45%_Testing	0.0198	0.0915	0.2611	0.2270	0.8502	1.4000
RR 40%_Testing	0.0196	0.1320	0.3063	0.2508	0.8985	3.0000
RR 30%_Testing	0.0211	0.1036	0.3003	0.2416	1.0507	3.4000
RR 35%_Testing	0.0210	0.1272	0.3367	0.2831	0.9617	4.0000
RR 25%_Testing	0.0210	0.0954	0.4392	0.3860	1.1495	4.2000
RR 20%(Testing)	0.0170	0.1781	0.4631	0.3916	1.2894	5.0000

In [39]:
```
df2 = df_results[df_results['epsilon'] == 'RR 45%_Testing']
df2
```

Out[39]:

	stock_pair	accuracy	precision	recall	f1	%class 1	epsilon
1520	"HDFCBANK_M&M"	0.9882	0.9203	0.9478	0.9338	0.0879	RR 45%_Testing
1521	"HDFCBANK_ULTRACEMCO"	0.9731	0.9178	0.9877	0.9515	0.2671	RR 45%_Testing
1522	"HDFCBANK_GRASIM"	0.9941	0.9986	0.9889	0.9937	0.4711	RR 45%_Testing
1523	"HDFCBANK_PIDILITIND"	0.9711	0.9632	0.7706	0.8562	0.1115	RR 45%_Testing
1524	"HDFCBANK_LT"	0.9928	0.9924	0.9286	0.9594	0.0919	RR 45%_Testing
...	...	...	...	...	...	...	...
1705	"TATASTEEL_CONCOR"	0.9770	0.9655	0.8924	0.9275	0.1647	RR 45%_Testing
1706	"TATASTEEL_BHARTIARTL"	0.9731	0.8953	0.9636	0.9282	0.1804	RR 45%_Testing
1707	"SUNPHARMA_CONCOR"	0.9921	0.8913	0.8542	0.8723	0.0315	RR 45%_Testing
1708	"SUNPHARMA_BHARTIARTL"	0.9777	0.9289	0.9015	0.9150	0.1332	RR 45%_Testing
1709	"CONCOR_BHARTIARTL"	0.9856	0.9778	0.6769	0.8000	0.0427	RR 45%_Testing

190 rows × 7 columns

In [40]:
```
len(df2[df2['f1_1'] == 0.0])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\pandas\core\indexes\base.py:3802, in Index.get_loc(self, key, method, tolerance)
   3801 try:
-> 3802     return self._engine.get_loc(casted_key)
   3803 except KeyError as err:

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\pandas\_libs\index.pyx:138, in pandas._libs.index.IndexEngine.get_loc()

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\pandas\_libs\index.pyx:165, in pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:5745, in pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:5753, in pandas._libs.hashtable.PyObjectHashTable.get_item()
```

```
KeyError: 'f1_1'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[40], line 1
----> 1 len(df2[df2['f1_1'] == 0.0])

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\pandas\core\frame.py:3807, in DataFrame.__getitem__(self, key)
   3805 if self.columns.nlevels > 1:
   3806     return self._getitem_multilevel(key)
-> 3807 indexer = self.columns.get_loc(key)
   3808 if is_integer(indexer):
   3809     indexer = [indexer]

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\pandas\core\indexes\base.py:3804, in Index.get_loc(self, key, method, tolerance)
   3802     return self._engine.get_loc(casted_key)
   3803 except KeyError as err:
-> 3804     raise KeyError(key) from err
   3805 except TypeError:
   3806     # If we have a listlike key, _check_indexing_error will raise
   3807     #  InvalidIndexError. Otherwise we fall through and re-raise
   3808     #  the TypeError.
   3809     self._check_indexing_error(key)

KeyError: 'f1_1'
```

In [ ]:
```
df2[df2['stock_pair'] == '"HDFCBANK_UPL"']
```

In [ ]:
```
df2[df2['f1_1'] <= 0.4]
```

In [ ]:
```
len(df2[df2['f1_1'] <= 0.4])
```

In [ ]:
```
filename = "predicted_series_lstm_block20_w20.csv"
predicted_series_lstm = pd.read_csv(filename)

test_data_file = pd.read_csv('test_data3.csv')
```

In [ ]:
```
def plot2(pred,test,stockname,block,w):
    plt.figure(figsize = (30,10))
    plt.plot(pred, label = "prediction", c = "orange")
    plt.plot(test, label = "actual", c = "green")
    plt.xlabel("Time", fontsize=18)
    plt.ylabel("Distance", fontsize=18)
    plt.legend(fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.axvspan(215, 375, color='grey', alpha=0.2)
    plt.axvspan(125, 215, color='red', alpha=0.2)
    plt.axvspan(50, 125, color='green', alpha=0.2)
    plt.grid(True)
    plt.savefig('ComparisonGraph_Prediction.jpeg', dpi=1200, bbox_inches='tight')
    plt.show()


def plot_actual2(stockname, n):
    stockname = stockname[1:-1]
    s = stockname.split('_')
    print(s)
```

```
    s1 = s[0]+'.NS'
    s2 = s[1]+'.NS'
    stock1 = yf.download(s1, start='2003-01-01', end='2023-12-31')['Adj Close']
    stock2 = yf.download(s2, start='2003-01-01', end='2023-12-31')['Adj Close']

    # Take the last 3701 prices for each stock
    stock1_1 = stock1[-n:]
    stock2_1 = stock2[-n:]

    # Normalize time series individually using standard deviation
    normalized_stock1 = (stock1_1 - stock1_1.mean()) / (stock1_1.std())
    normalized_stock2 = (stock2_1 - stock2_1.mean()) / (stock2_1.std())

    # Plot the normalized time series
    plt.figure(figsize=(30, 10))
    plt.plot(normalized_stock1, label=s[0])
    plt.plot(normalized_stock2, label=s[1])
    plt.xlabel('Date', fontsize=18)
    plt.ylabel('Normalized Price', fontsize=18)
    plt.legend(fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.axvspan('2018-11-01', '2019-08-01', color='grey', alpha=0.2)
    plt.axvspan('2018-06-01', '2018-11-01', color='red', alpha=0.2)
    plt.axvspan('2018-02-01', '2018-06-01', color='green', alpha=0.2)
    plt.grid(True)
    plt.savefig('ComparisonGraph_Actual.jpeg', dpi=1200, bbox_inches='tight')
    plt.show()
```

In [ ]:
```
stock = '"HDFCBANK_UPL"'
test = test_data_file[stock][int(20):].tolist()
pred = predicted_series_lstm[stock].tolist()
plot2(pred,test, stock, 20, 20)
plot_actual2(stock, len(test))
```

In [ ]:
```
import matplotlib.pyplot as plt
import yfinance as yf

def plot2(ax, pred, test):
    ax.plot(pred, label="prediction", c="orange")
    ax.plot(test, label="actual", c="green")
    ax.set_xlabel("Time", fontsize=18)
    ax.set_ylabel("Distance", fontsize=18)
    ax.legend(fontsize=14)
    ax.tick_params(axis='x', labelsize=12)
    ax.tick_params(axis='y', labelsize=12)
    ax.axvspan(215, 375, color='grey', alpha=0.2)
    ax.axvspan(125, 215, color='red', alpha=0.2)
    ax.axvspan(50, 125, color='green', alpha=0.2)
    ax.grid(True)
    ax.text(0.5, -0.15, '(a)', transform=ax.transAxes, fontsize=20, va='top', ha='center

def plot_actual2(ax, stockname, n):
    stockname = stockname[1:-1]
    s = stockname.split('_')
    s1 = s[0]+'.NS'
    s2 = s[1]+'.NS'
    stock1 = yf.download(s1, start='2003-01-01', end='2023-12-31')['Adj Close']
    stock2 = yf.download(s2, start='2003-01-01', end='2023-12-31')['Adj Close']

    stock1_1 = stock1[-n:]
    stock2_1 = stock2[-n:]

    normalized_stock1 = (stock1_1 - stock1_1.mean()) / stock1_1.std()
    normalized_stock2 = (stock2_1 - stock2_1.mean()) / stock2_1.std()
```

```
    ax.plot(normalized_stock1, label=s[0])
    ax.plot(normalized_stock2, label=s[1])
    ax.set_xlabel('Date', fontsize=18)
    ax.set_ylabel('Normalized Price', fontsize=18)
    ax.legend(fontsize=14)
    ax.tick_params(axis='x', labelsize=12)
    ax.tick_params(axis='y', labelsize=12)
    ax.axvspan('2018-11-01', '2019-08-01', color='grey', alpha=0.2)
    ax.axvspan('2018-06-01', '2018-11-01', color='red', alpha=0.2)
    ax.axvspan('2018-02-01', '2018-06-01', color='green', alpha=0.2)
    ax.grid(True)
    ax.text(0.5, -0.15, '(b)', transform=ax.transAxes, fontsize=20, va='top', ha='center

# Create subplots
fig, axes = plt.subplots(2, 1, figsize=(30, 20))

# Plot prediction vs actual
plot2(axes[0], pred, test)

# Plot actual stock prices
plot_actual2(axes[1], stock, len(test))

# Adjust layout to add spacing between plots
plt.tight_layout(pad=6.0)

# Save and show the figure
plt.savefig('ComparisonGraph_Combined.jpeg', dpi=1200, bbox_inches='tight')
plt.show()
```

In [ ]:

In [ ]:

In [ ]: