Module 4

Trade Call Prediction using Classification

In this module, we'd be covering the concept of classification and utilize our skills to solve the following queries – (Stock Price = Close Price)

Problem Statements

`

Problem 4.1

Import the csv file of the stock which contained the Bollinger columns as well.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
df = pd.read_csv("RELIANCE.csv", parse_dates=True)
df.head()
```

Out[2]:

	Symbol	Series	Date	Prev Close	Open Price	High Price	Low Price	Last Price	Close Price	Average Price
0	RELIANCE	EQ	15-May-2017	1350.45	1356.40	1356.40	1333.50	1343.60	1344.10	1344.22
1	RELIANCE	EQ	16-May-2017	1344.10	1346.05	1376.90	1341.00	1356.20	1356.30	1360.59
2	RELIANCE	EQ	17-May-2017	1356.30	1353.00	1365.95	1347.75	1350.00	1353.10	1354.16
3	RELIANCE	EQ	18-May-2017	1353.10	1340.25	1350.00	1324.10	1327.45	1327.35	1336.14
4	RELIANCE	EQ	19-May-2017	1327.35	1333.00	1335.70	1310.00	1318.20	1318.85	1321.99

In [3]:

```
df.shape
```

Out[3]:

```
(495, 15)
```

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 495 entries, 0 to 494
Data columns (total 15 columns):
Symbol                    495 non-null object
Series                    495 non-null object
Date                      495 non-null object
Prev Close                495 non-null float64
Open Price                495 non-null float64
High Price                495 non-null float64
Low Price                 495 non-null float64
Last Price                495 non-null float64
Close Price               495 non-null float64
Average Price             495 non-null float64
Total Traded Quantity     495 non-null int64
Turnover                  495 non-null float64
No. of Trades             495 non-null int64
Deliverable Qty           495 non-null int64
% Dly Qt to Traded Qty    495 non-null float64
dtypes: float64(9), int64(3), object(3)
memory usage: 58.1+ KB
```

In [5]:

```
df.duplicated().sum()
```

Out[5]:

```
0
```

In [6]:

```
df.isnull().sum()
```

Out[6]:

```
Symbol                    0
Series                    0
Date                      0
Prev Close                0
Open Price                0
High Price                0
Low Price                 0
Last Price                0
Close Price               0
Average Price             0
Total Traded Quantity     0
Turnover                  0
No. of Trades             0
Deliverable Qty           0
% Dly Qt to Traded Qty    0
dtype: int64
```

Source: For Calculating Bollinger Bands (https://towardsdatascience.com/trading-technical-analysis-with-pandas-43e737a17861)

Bollinger Bands (https://www.bollingerbands.com/bollinger-bands) is used to define the prevailing high and low prices in a market to characterize the trading band of a financial instrument or commodity. Bollinger Bands are a volatility indicator. Bands are consists of Moving Average (MA) line, a upper band and lower band. The upper and lower bands are simply MA adding and subtracting standard deviation. Standard deviation is a measurement of volatility. That's why it's a volatility indictor.

Create a new column 'Call', whose entries are -

- 'Buy' if the stock price is below the lower Bollinger band
- 'Hold Buy/ Liquidate Short' if the stock price is between the lower and middle Bollinger band
- 'Hold Short/ Liquidate Buy' if the stock price is between the middle and upper Bollinger band
- 'Short' if the stock price is above the upper Bollinger band

In [7]:

```
# calculating Simple Moving Average with 20 days window
df['sma'] = df['Close Price'].rolling(window=20).mean()

# calculating the standar deviation
df['rstd'] = df['Close Price'].rolling(window=20).std()
```

In [8]:

```
df['upper_band'] = df['sma'] + 2 * df['rstd']
df['lower_band'] = df['sma'] - 2 * df['rstd']
df['mid_band'] = (df['upper_band'] + df['lower_band']) / 2
df.dropna(inplace=True)
df.head()
```

Out[8]:

	Symbol	Series	Date	Prev Close	Open Price	High Price	Low Price	Last Price	Close Price	Average Price
19	RELIANCE	EQ	09-Jun-2017	1331.70	1345.10	1352.95	1331.0	1332.95	1335.70	1340.93
20	RELIANCE	EQ	12-Jun-2017	1335.70	1326.55	1329.75	1317.0	1319.00	1319.45	1321.07
21	RELIANCE	EQ	13-Jun-2017	1319.45	1320.60	1327.00	1311.0	1312.00	1314.35	1318.87
22	RELIANCE	EQ	14-Jun-2017	1314.35	1315.90	1360.00	1315.9	1360.00	1357.50	1348.06
23	RELIANCE	EQ	15-Jun-2017	1357.50	1360.00	1395.00	1359.1	1377.35	1383.95	1379.51

In [9]:

```
def call(df):
    if df['Close Price'] < df['lower_band']:
        return "Buy"
    elif (df['Close Price'] > df['lower_band']) and (df['Close Price'] < df['mid_band'
]):
        return "Hold Buy/Liquidate Short"
    elif (df['Close Price'] > df['mid_band']) and (df['Close Price'] < df['upper_band'
]):
        return "Hold Short/Liquidate Buy"
    else:
        return "Short"
```

In [10]:

```
df['Call'] = df.apply(call, axis = 1)
df.head()
```

Out[10]:

	Symbol	Series	Date	Prev Close	Open Price	High Price	Low Price	Last Price	Close Price	Average Price
19	RELIANCE	EQ	09-Jun-2017	1331.70	1345.10	1352.95	1331.0	1332.95	1335.70	1340.93
20	RELIANCE	EQ	12-Jun-2017	1335.70	1326.55	1329.75	1317.0	1319.00	1319.45	1321.07
21	RELIANCE	EQ	13-Jun-2017	1319.45	1320.60	1327.00	1311.0	1312.00	1314.35	1318.87
22	RELIANCE	EQ	14-Jun-2017	1314.35	1315.90	1360.00	1315.9	1360.00	1357.50	1348.06
23	RELIANCE	EQ	15-Jun-2017	1357.50	1360.00	1395.00	1359.1	1377.35	1383.95	1379.51

5 rows × 21 columns

Now train a classification model with the 3 bollinger columns and the stock price as inputs and 'Calls' as output. Check the accuracy on a test set. (There are many classifier models to choose from, try each one out and compare the accuracy for each)

Import another stock data and create the bollinger columns. Using the already defined model, predict the daily calls for this new stock.

In [11]:

```
from sklearn.preprocessing import StandardScaler,LabelEncoder
scr = StandardScaler()
lbc = LabelEncoder()
```

In [12]:

```
x = df[['Close Price', 'lower_band', 'mid_band', 'upper_band']]
x = scr.fit_transform(x)
y = df['Call']
y = lbc.fit_transform(y).reshape(-1, 1)
```

In [13]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size = 0.3, random_state
= 7)
```

In [14]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
```

In [15]:

```
max_features = y.max()
```

In [16]:

```
from collections import OrderedDict
```

In [17]:

```
classifier_models = OrderedDict([
    ("Nearest Neighbors", KNeighborsClassifier(max_features)),
    ("Linear SVM",        SVC(kernel="linear", C=0.025)),
    ("RBF SVM",           SVC(gamma=2, C=1)),
    ("Decision Tree",     DecisionTreeClassifier(max_depth=5)),
    ("Random Forest",     RandomForestClassifier(max_depth=5, n_estimators=10, max_feat
ures=max_features)),
    ("AdaBoost",          AdaBoostClassifier()),
    ("Naive Bayes",       GaussianNB())
])

classifier_models
```

Out[17]:

```
OrderedDict([('Nearest Neighbors',
              KNeighborsClassifier(algorithm='auto', leaf_size=30, metric
='minkowski',
                                   metric_params=None, n_jobs=None, n_neighbors=3, p
=2,
                                   weights='uniform')),
             ('Linear SVM',
              SVC(C=0.025, cache_size=200, class_weight=None, coef0=0.0,
                decision_function_shape='ovr', degree=3, gamma='auto_depre
cated',
                kernel='linear', max_iter=-1, probability=False, random_st
ate=None,
                shrinking=True, tol=0.001, verbose=False)),
             ('RBF SVM', SVC(C=1, cache_size=200, class_weight=None, coef0
=0.0,
                decision_function_shape='ovr', degree=3, gamma=2, kernel
='rbf',
                max_iter=-1, probability=False, random_state=None, shrinki
ng=True,
                tol=0.001, verbose=False)),
             ('Decision Tree',
              DecisionTreeClassifier(class_weight=None, criterion='gini',
max_depth=5,
                                    max_features=None, max_leaf_nodes=None,
                                    min_impurity_decrease=0.0, min_impurity_split=No
ne,
                                    min_samples_leaf=1, min_samples_split=2,
                                    min_weight_fraction_leaf=0.0, presort=False, ran
dom_state=None,
                                    splitter='best')),
             ('Random Forest',
              RandomForestClassifier(bootstrap=True, class_weight=None, cr
iterion='gini',
                                    max_depth=5, max_features=3, max_leaf_nodes=Non
e,
                                    min_impurity_decrease=0.0, min_impurity_split=No
ne,
                                    min_samples_leaf=1, min_samples_split=2,
                                    min_weight_fraction_leaf=0.0, n_estimators=10, n
_jobs=None,
                                    oob_score=False, random_state=None, verbose=0,
                                    warm_start=False)),
             ('AdaBoost',
              AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
                                learning_rate=1.0, n_estimators=50, random_state=N
one)),
             ('Naive Bayes', GaussianNB(priors=None, var_smoothing=1e-0
9))])
```

In [18]:

```
accuracy_scores = {}
for model_name, classifier in classifier_models.items():
    classifier.fit(X_train, Y_train)
    y_pred = classifier.predict(X_test)
    accuracy_scores[model_name] = classifier.score(X_test, Y_test)

accuracy_scores = OrderedDict(sorted(accuracy_scores.items(), key=lambda x: x[1]))
accuracy_scores
```

```
C:\Users\Jesus\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel
().
  This is separate from the ipykernel package so we can avoid doing import
s until
C:\Users\Jesus\Anaconda3\lib\site-packages\sklearn\utils\validation.py:76
1: DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Jesus\Anaconda3\lib\site-packages\sklearn\utils\validation.py:76
1: DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Jesus\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples,), for example using ravel
().
  This is separate from the ipykernel package so we can avoid doing import
s until
C:\Users\Jesus\Anaconda3\lib\site-packages\sklearn\utils\validation.py:76
1: DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
C:\Users\Jesus\Anaconda3\lib\site-packages\sklearn\utils\validation.py:76
1: DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

Out[18]:

```
OrderedDict([('Naive Bayes', 0.5174825174825175),
             ('Linear SVM', 0.5454545454545454),
             ('AdaBoost', 0.6713286713286714),
             ('Decision Tree', 0.7272727272727273),
             ('RBF SVM', 0.7692307692307693),
             ('Random Forest', 0.7832167832167832),
             ('Nearest Neighbors', 0.8041958041958042)])
```

In [19]:

```
hero_df = pd.read_csv('HEROMOTOCO.csv', parse_dates=['Date'])
hero_df.set_index('Date', inplace=True)

hero_df["sma"] = hero_df["Close Price"].rolling(20).mean()
hero_df["std"] = hero_df["Close Price"].rolling(20).std()
hero_df["upper_band"] = hero_df["sma"] + hero_df["std"] * 2
hero_df["lower_band"] = hero_df["sma"] - hero_df["std"] * 2
```

In [20]:

```
hero_df.isnull().sum()
```

Out[20]:

```
Symbol                     0
Series                     0
Prev Close                 0
Open Price                 0
High Price                 0
Low Price                  0
Last Price                 0
Close Price                0
Average Price              0
Total Traded Quantity      0
Turnover                   0
No. of Trades              0
Deliverable Qty            0
% Dly Qt to Traded Qty     0
sma                       19
std                       19
upper_band                19
lower_band                19
dtype: int64
```

In [21]:

```
hero_df   = hero_df.dropna()
hero_X = scr.fit_transform(hero_df[['Close Price', 'std', 'upper_band', 'lower_band']])

hero_df['Call'] = classifier_models["Nearest Neighbors"].predict(hero_X)


hero_df.to_csv('hero_trained.csv')
```

In [22]:

```
print("hero_df['Call'].unique() =", hero_df['Call'].unique())
hero_df.head()
```

```
hero_df['Call'].unique() = [2 1 3]
```

Out[22]:

	Symbol	Series	Prev Close	Open Price	High Price	Low Price	Last Price	Close Price	Average Price
Date									
2017-06-09	HEROMOTOCO	EQ	3784.45	3796.3	3800.50	3755.60	3786.00	3780.10	3779.46
2017-06-12	HEROMOTOCO	EQ	3780.10	3777.0	3790.00	3760.35	3771.05	3773.25	3779.97
2017-06-13	HEROMOTOCO	EQ	3773.25	3780.0	3785.35	3742.25	3751.00	3752.50	3765.91
2017-06-14	HEROMOTOCO	EQ	3752.50	3764.0	3808.00	3751.05	3781.30	3789.05	3788.77
2017-06-15	HEROMOTOCO	EQ	3789.05	3786.0	3820.00	3760.00	3774.00	3777.70	3785.08

In [23]:

```
hero_df.reset_index(inplace=True)
```

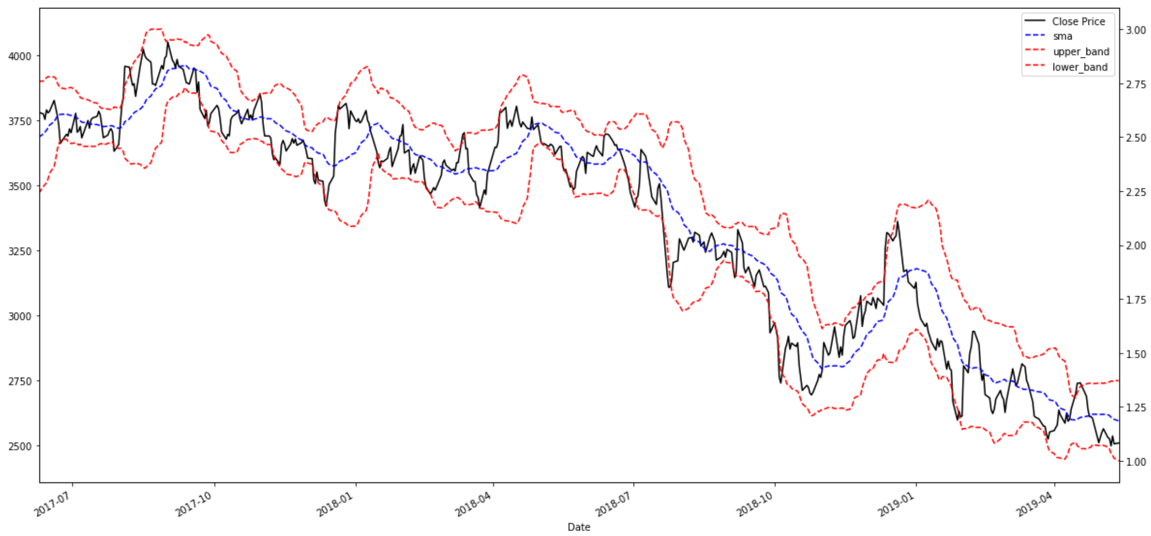
In [24]:

```
fig = plt.figure(figsize=(20,10))
ax1 = plt.gca()
ax2 = ax1.twinx()


hero_df.plot(kind='line',x='Date', y='Close Price', ax=ax1, color='black')
hero_df.plot(kind='line',x='Date', y='sma', ax=ax1, color='blue', linestyle='--')
hero_df.plot(kind='line',x='Date', y='upper_band', ax=ax1, color='red',  linestyle='--'
)
hero_df.plot(kind='line',x='Date', y='lower_band', ax=ax1, color='red',  linestyle='--'
)
ax2.plot(hero_df['Call'])
```

Out[24]:

```
[<matplotlib.lines.Line2D at 0x250d4fd53c8>]
```



Problem 4.2

Now, we'll again utilize classification to make a trade call, and measure the efficiency of our trading algorithm over the past two years. For this assignment , we will use RandomForest classifier.

Import the stock data file of your choice

In [25]:

```
maruthi_df = pd.read_csv('MARUTI.csv').set_index('Date')

maruthi_df.head()
```

Out[25]:

Date	Symbol	Series	Prev Close	Open Price	High Price	Low Price	Last Price	Close Price	Average Price	Total Traded Quantity
15-May-2017	MARUTI	EQ	6730.20	6759.4	6839.10	6733.45	6824.0	6823.90	6796.84	336356
16-May-2017	MARUTI	EQ	6823.90	6850.0	6977.55	6830.00	6968.4	6953.95	6902.22	707624
17-May-2017	MARUTI	EQ	6953.95	6950.0	6979.00	6885.85	6945.0	6958.20	6931.60	445461
18-May-2017	MARUTI	EQ	6958.20	6918.3	6948.00	6814.45	6822.0	6831.05	6869.68	406814
19-May-2017	MARUTI	EQ	6831.05	6854.8	6893.90	6691.55	6756.3	6790.55	6791.60	552223

In [26]:

```
maruthi_df.shape
```

Out[26]:

(496, 14)

In [27]:

```
maruthi_df.isnull().sum()
```

Out[27]:

```
Symbol                    0
Series                    0
Prev Close                0
Open Price                0
High Price                0
Low Price                 0
Last Price                0
Close Price               0
Average Price             0
Total Traded Quantity     0
Turnover                  0
No. of Trades             0
Deliverable Qty           0
% Dly Qt to Traded Qty    0
dtype: int64
```

Define 4 new columns , whose values are:

- % change between Open and Close price for the day
- % change between Low and High price for the day
- 5 day rolling mean of the day to day % change in Close Price
- 5 day rolling std of the day to day % change in Close Price

In [28]:

```
maruthi_df['price_Open_Close'] = (maruthi_df['Close Price'] - maruthi_df['Open Price'])
/ maruthi_df['Open Price']
maruthi_df['price_High_Low'] = (maruthi_df['High Price']  - maruthi_df['Low Price']) /
maruthi_df['Low Price']

maruthi_df['Day_Perc_Change'] = maruthi_df['Close Price'].pct_change().fillna(0)

maruthi_df['5day_mean'] = maruthi_df['Day_Perc_Change'].rolling(5).mean()
maruthi_df['5day_std'] = maruthi_df['Day_Perc_Change'].rolling(5).std()

maruthi_df.dropna(inplace=True)

maruthi_df.head()
```

Out[28]:

	Symbol	Series	Prev Close	Open Price	High Price	Low Price	Last Price	Close Price	Average Price	To Trad Quant
Date										
19-May-2017	MARUTI	EQ	6831.05	6854.80	6893.90	6691.55	6756.30	6790.55	6791.60	5522
22-May-2017	MARUTI	EQ	6790.55	6803.95	6843.95	6689.25	6694.30	6701.70	6732.22	3274
23-May-2017	MARUTI	EQ	6701.70	6765.00	6910.00	6743.65	6867.95	6878.85	6855.36	9564
24-May-2017	MARUTI	EQ	6878.85	6903.00	6912.55	6813.00	6865.00	6869.65	6863.75	4336
25-May-2017	MARUTI	EQ	6869.65	6879.95	7018.00	6835.00	6970.00	6985.70	6916.65	4357

Create a new column 'Action' whose values are:

- 1 if next day's price(Close) is greater than present day's.
- (-1) if next day's price(Close) is less than present day's.
- i.e. Action [i] = 1 if Close[i+1] > Close[i]
- i.e. Action [i] = (-1) if Close[i+1] < Close[i]

In [29]:

```
maruthi_df['Action'] = np.where(maruthi_df['Close Price'].shift(-1) > maruthi_df['Close Price'], 1, -1 )

maruthi_df.head()
```

Out[29]:

Date	Symbol	Series	Prev Close	Open Price	High Price	Low Price	Last Price	Close Price	Average Price	To Trad Quant
19-May-2017	MARUTI	EQ	6831.05	6854.80	6893.90	6691.55	6756.30	6790.55	6791.60	5522
22-May-2017	MARUTI	EQ	6790.55	6803.95	6843.95	6689.25	6694.30	6701.70	6732.22	3274
23-May-2017	MARUTI	EQ	6701.70	6765.00	6910.00	6743.65	6867.95	6878.85	6855.36	9564
24-May-2017	MARUTI	EQ	6878.85	6903.00	6912.55	6813.00	6865.00	6869.65	6863.75	4336
25-May-2017	MARUTI	EQ	6869.65	6879.95	7018.00	6835.00	6970.00	6985.70	6916.65	4357

In [30]:

```
maruthi_df.to_csv('maruthi_trained.csv')
```

Construct a classification model with the 4 new inputs and 'Action' as target

In [31]:

```
maruthi_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 492 entries, 19-May-2017 to 13-May-2019
Data columns (total 20 columns):
Symbol                    492 non-null object
Series                    492 non-null object
Prev Close                492 non-null float64
Open Price                492 non-null float64
High Price                492 non-null float64
Low Price                 492 non-null float64
Last Price                492 non-null float64
Close Price               492 non-null float64
Average Price             492 non-null float64
Total Traded Quantity     492 non-null int64
Turnover                  492 non-null float64
No. of Trades             492 non-null int64
Deliverable Qty           492 non-null int64
% Dly Qt to Traded Qty    492 non-null float64
price_Open_Close          492 non-null float64
price_High_Low            492 non-null float64
Day_Perc_Change           492 non-null float64
5day_mean                 492 non-null float64
5day_std                  492 non-null float64
Action                    492 non-null int32
dtypes: float64(14), int32(1), int64(3), object(2)
memory usage: 78.8+ KB
```

In [32]:

```
maruthi_df.isnull().sum()
```

Out[32]:

```
Symbol                    0
Series                    0
Prev Close                0
Open Price                0
High Price                0
Low Price                 0
Last Price                0
Close Price               0
Average Price             0
Total Traded Quantity     0
Turnover                  0
No. of Trades             0
Deliverable Qty           0
% Dly Qt to Traded Qty    0
price_Open_Close          0
price_High_Low            0
Day_Perc_Change           0
5day_mean                 0
5day_std                  0
Action                    0
dtype: int64
```

In [33]:

```
maruthi_df.dropna(inplace=True)
X = maruthi_df[['price_Open_Close', 'price_High_Low', '5day_mean', '5day_std']]
Y = maruthi_df['Action']


X = StandardScaler().fit_transform(X)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=
42)


rfc = RandomForestClassifier(n_estimators=100, max_features=2)
rfc
```

Out[33]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
            max_depth=None, max_features=2, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [34]:

```
rfc.fit(X_train,  Y_train)
rfc.score(X_test, Y_test)
```

Out[34]:

```
0.5153374233128835
```

Check the accuracy of this model , also , plot the net cumulative returns (in %) if we were to follow this algorithmic model

image

In [35]:

```
# Cumulative Product of PCT change in Close_Price with predicted actions
plt.figure(figsize=(25, 7))

cumulative_returns = ( 1 + (maruthi_df['Close Price'].pct_change() * maruthi_df['Actio
n']) ).dropna().cumprod()
cumulative_returns.plot()
```

Out[35]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x250d57c0a20>
```