1. Finance Problem Summary & Preparation

Short-term asset return is a challenging quantity to predict. Effcient markets produce near-Normal daily returns with no significant correlation between r_t, r_{t-1}. In this project, I will do a limited exercise in supervised learning with an objective to produce a model to predict positive moves (up trend) using machine learning model as specified later in the below section.

1.1 Data Acquisition & Importing Packages

For this experiment, I choose `Hang Seng Index` (Ticker: HSI Index) daily closing price as the target form and download data from `Bloomberg` for period from 12/10/2018 till 13/10/2023. The total row count is 1233 and no non-null data.

In [ ]:
```
#Basic
import pandas as pd
import numpy as np
import talib #"C:\Users\Lenovo\ta_lib-0.4.25-cp311-cp311-win_amd64.whl"

# Visualization
import matplotlib
import matplotlib.pyplot as plt
!pip install seaborn
import seaborn as sns ## correlation matrix
from pylab import plt
plt.style.use('seaborn')
%matplotlib inline

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

```
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Requirement already satisfied: seaborn in c:\python311\lib\site-packages (0.13.0)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\python311\lib\site-packages (from seaborn) (1.25.0)
Requirement already satisfied: pandas>=1.2 in c:\python311\lib\site-packages (from seaborn) (2.0.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.3 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from seaborn) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib!=3.6.1,>=3.3->seaborn) (1.1.0)
Requirement already satisfied: cycler>=0.10 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib!=3.6.1,>=3.3->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib!=3.6.1,>=3.3->seaborn) (4.40.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib!=3.6.1,>=3.3->seaborn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib!=3.6.1,>=3.3->seaborn) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib!=3.6.1,>=3.3->seaborn) (9.5.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib!=3.6.1,>=3.3->seaborn) (3.1.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib!=3.6.1,>=3.3->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\python311\lib\site-packages (from pandas>=1.2->seaborn) (2023.3)
Requirement already satisfied: tzdata>=2022.1 in c:\python311\lib\site-packages (from pandas>=1.2->seaborn) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.3->seaborn) (1.16.0)
```

```
[notice] A new release of pip is available: 23.2.1 -> 23.3
[notice] To update, run: python.exe -m pip install --upgrade pip
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_8592\1792456682.py:12: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer co
rrespond to the styles shipped by seaborn. However, they will remain available as 'seaborn-v0_8-<style>'. Alternatively, directly use the seaborn API instead.
 plt.style.use('seaborn')
```

In [ ]:
```
# Preprocessing & Cross validation
!pip3 install scikit-learn
from sklearn.preprocessing import MinMaxScaler, RobustScaler, StandardScaler, Normalizer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV, TimeSeriesSplit, cross_val_score
from sklearn.model_selection import RandomizedSearchCV, cross_val_score
import datetime, pickle

# SVM
from sklearn import svm
from sklearn.svm import SVR
from sklearn.svm import SVC

# Metrics
from src.plot_confusion_matrix import plot_confusion_matrix
from src.plot_roc_curve import plot_roc_curve
from src.Features_Library import pastReturns
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error, accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import auc, roc_curve
from sklearn.metrics import auc, roc_auc_score


# Classifier
from sklearn.ensemble import RandomForestClassifier #分类决策树模型
from matplotlib.colors import ListedColormap

# Confusion Matrix
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import classification_report

# SHAP
!pip install shap
import shap
```

```
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Requirement already satisfied: scikit-learn in c:\python311\lib\site-packages (1.3.1)
Requirement already satisfied: numpy<2.0,>=1.17.3 in c:\python311\lib\site-packages (from scikit-learn) (1.25.0)
Requirement already satisfied: scipy>=1.5.0 in c:\python311\lib\site-packages (from scikit-learn) (1.11.0)
Requirement already satisfied: joblib>=1.1.1 in c:\python311\lib\site-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\python311\lib\site-packages (from scikit-learn) (3.2.0)
```

```
[notice] A new release of pip is available: 23.2.1 -> 23.3
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Requirement already satisfied: shap in c:\python311\lib\site-packages (0.43.0)
Requirement already satisfied: numpy in c:\python311\lib\site-packages (from shap) (1.25.0)
Requirement already satisfied: scipy in c:\python311\lib\site-packages (from shap) (1.11.0)
Requirement already satisfied: scikit-learn in c:\python311\lib\site-packages (from shap) (1.3.1)
Requirement already satisfied: pandas in c:\python311\lib\site-packages (from shap) (2.0.2)
Requirement already satisfied: tqdm>=4.27.0 in c:\python311\lib\site-packages (from shap) (4.66.1)
Requirement already satisfied: packaging>20.9 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from shap) (23.1)
Requirement already satisfied: slicer==0.0.7 in c:\python311\lib\site-packages (from shap) (0.0.7)
Requirement already satisfied: numba in c:\python311\lib\site-packages (from shap) (0.58.1)
Requirement already satisfied: cloudpickle in c:\python311\lib\site-packages (from shap) (3.0.0)
Requirement already satisfied: colorama in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from tqdm>=4.27.0->shap) (0.4.6)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in c:\python311\lib\site-packages (from numba->shap) (0.41.1)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\python311\lib\site-packages (from pandas->shap) (2023.3)
Requirement already satisfied: tzdata>=2022.1 in c:\python311\lib\site-packages (from pandas->shap) (2023.3)
Requirement already satisfied: joblib>=1.1.1 in c:\python311\lib\site-packages (from scikit-learn->shap) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\python311\lib\site-packages (from scikit-learn->shap) (3.2.0)
Requirement already satisfied: six>=1.5 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from python-dateutil>=2.8.2->pandas->shap) (1.16.0)
```

```
[notice] A new release of pip is available: 23.2.1 -> 23.3
[notice] To update, run: python.exe -m pip install --upgrade pip
```

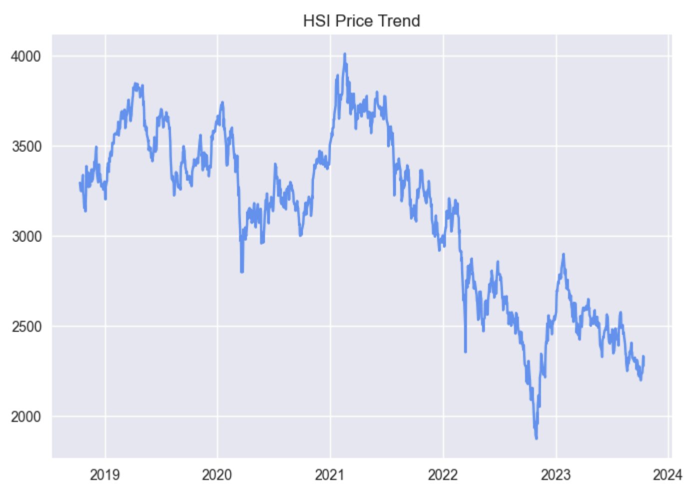
1.2 Description of Dataset

In [ ]:
```python
# Load file
hsi_px = pd.read_csv('HSI_Index_2018_2023.csv', index_col=0, parse_dates=True)
hsi_px.head()
```

Out[ ]:

Date	Open	High	Low	Close	Volume
2023-10-13	2294.01535	2302.56673	2272.30839	2276.97394	2139651209
2023-10-12	2333.71167	2337.88489	2323.11438	2331.14894	3861574899
2023-10-11	2296.38728	2304.77902	2287.80372	2288.23725	2029133726
2023-10-10	2267.00642	2291.30368	2255.67307	2259.09021	1596847248
2023-10-09	2234.92670	2251.88879	2228.20801	2237.10155	1221298004

In [ ]:
```python
# Visualize raw price series
plt.title('HSI Price Trend')
plt.plot(hsi_px['Close'], color='cornflowerblue');
```



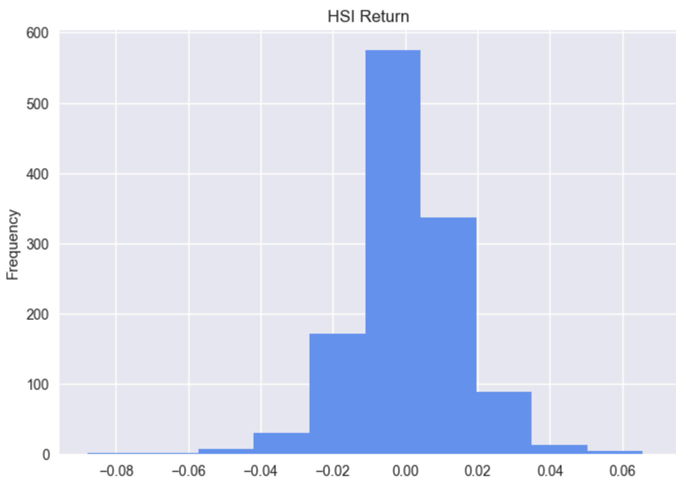
In [ ]:
```python
# Calculate returns
hsi_px['Returns'] = np.log(hsi_px['Close']).diff()
hsi_px = hsi_px.dropna()
hsi_px
```

Out[ ]:

Date	Open	High	Low	Close	Volume	Returns
2023-10-12	2333.71167	2337.88489	2323.11438	2331.14894	3861574899	0.023514
2023-10-11	2296.38728	2304.77902	2287.80372	2288.23725	2029133726	-0.018579
2023-10-10	2267.00642	2291.30368	2255.67307	2259.09021	1596847248	-0.012820
2023-10-09	2234.92670	2251.88879	2228.20801	2237.10155	1221298004	-0.009781
2023-10-06	2218.05470	2247.90581	2218.05470	2232.86086	1141209956	-0.001897
...	...	...	...	...	...	...
2018-10-19	3210.43147	3283.31016	3200.00765	3260.09157	2100898688	-0.023027
2018-10-18	3275.62292	3275.62292	3222.93923	3247.54086	1819304134	-0.003857
2018-10-16	3264.79010	3280.86385	3224.00664	3248.98048	1512630563	0.000443
2018-10-15	3277.55672	3277.55672	3241.30829	3247.11723	1664833532	-0.000574
2018-10-12	3241.83651	3297.27012	3227.21715	3292.89643	2164240597	0.014000

1231 rows × 6 columns

In [ ]:
```python
# Visualize return series
hsi_px['Close'].pct_change().plot.hist(bins=50)
plt.hist(hsi_px['Returns'], color='cornflowerblue')
plt.title('HSI Return');
```



2. Feature Engineering

Here, we are going to engineer some useful features from our stock price data for machine learning. In this context, our desired predictor variables are the moving average (MA), the relative strength index (RSI), and the daily volume change. Our target variable is the 5-days future close price change percentage.

We will use some in-built dataframe methods and ta-lib libraries to generate the predictor variables and target variables for machine learning.

2.1 Define Label - 'Sign'

Label or the target variable is also known as the dependent variable. In this project, the target variable *'Sign'* is designed as the label. Since the aim of this project is to predict positive moves, I will calculate sign value based on the **1-day future close price change percentage**:

$$y_i = \begin{cases} 1, & if * Returns* > 0.002 \\ 0, & \text{Otherwise} \end{cases}$$

Here I use the dataframe `shift method` to tweak the adjusted close price data to obtain the price percentage change for every next 1 days. With parameter -1, the price values will be shifted forward to the next 1 indexes. Then I set the parameter of 1 to the `pct_change method` to obtain the 1-day future close price change percentage.

In [ ]:
```
hsi_px['Returns'] = np.log(hsi_px['Close']/hsi_px['Close'].shift(1))
hsi_px = hsi_px.dropna()
hsi_px
```

Out[ ]:

	Open	High	Low	Close	Volume	Returns
Date						
2023-10-11	2296.38728	2304.77902	2287.80372	2288.23725	2029133726	-0.018579
2023-10-10	2267.00642	2291.30368	2255.67307	2259.09021	1596847248	-0.012820
2023-10-09	2234.92670	2251.88879	2228.20801	2237.10155	1221298004	-0.009781
2023-10-06	2218.05470	2247.90581	2218.05470	2232.86086	1141209956	-0.001897
2023-10-05	2202.72372	2213.76417	2196.03764	2198.11395	1088259898	-0.015684
...	...	...	...	...	...	...
2018-10-19	3210.43147	3283.31016	3200.00765	3260.09157	2100898688	-0.023027
2018-10-18	3275.62292	3275.62292	3222.93923	3247.54086	1819304134	-0.003857
2018-10-16	3264.79010	3280.86385	3224.00664	3248.98048	1512630563	0.000443
2018-10-15	3277.55672	3277.55672	3241.30829	3247.11723	1664833532	-0.000574
2018-10-12	3241.83651	3297.27012	3227.21715	3292.89643	2164240597	0.014000

1230 rows × 6 columns

In [ ]:
```
def num_config(x):
    if x > 0.002 :
        return 1
    else:
        return 0
```

In [ ]:
```
hsi_px['Sign'] = hsi_px['Returns'].map(num_config)
hsi_px
```

Out[ ]:

	Open	High	Low	Close	Volume	Returns	Sign
Date							
2023-10-11	2296.38728	2304.77902	2287.80372	2288.23725	2029133726	-0.018579	0
2023-10-10	2267.00642	2291.30368	2255.67307	2259.09021	1596847248	-0.012820	0
2023-10-09	2234.92670	2251.88879	2228.20801	2237.10155	1221298004	-0.009781	0
2023-10-06	2218.05470	2247.90581	2218.05470	2232.86086	1141209956	-0.001897	0
2023-10-05	2202.72372	2213.76417	2196.03764	2198.11395	1088259898	-0.015684	0
...	...	...	...	...	...	...	...
2018-10-19	3210.43147	3283.31016	3200.00765	3260.09157	2100898688	-0.023027	0
2018-10-18	3275.62292	3275.62292	3222.93923	3247.54086	1819304134	-0.003857	0
2018-10-16	3264.79010	3280.86385	3224.00664	3248.98048	1512630563	0.000443	0
2018-10-15	3277.55672	3277.55672	3241.30829	3247.11723	1664833532	-0.000574	0
2018-10-12	3241.83651	3297.27012	3227.21715	3292.89643	2164240597	0.014000	1

1230 rows × 7 columns

2.2 Define Features

Here, I engineer some useful features from our index price data for machine learning. In this context, my desired predictor variables include:

- `Open-Close:` O-C
- `High-Low:` H-L
- `Moving average:` MA_i
- `Relative strength index (RSI):` RSI_i
- `Momentum:` Momentum_i
- `Volume:` Volume_1d_change
- `Past returns:` Pass_Returns_i

In [ ]:
```
feature_names = []
```

O-C, H-L

In [ ]:
```
hsi_px['O-C'] = hsi_px['Open'] - hsi_px['Close']
hsi_px['H-L'] = hsi_px['High'] - hsi_px['Low']

#Append List
price_features = ['H-L','O-C']
feature_names.extend(price_features)
```

Moving Average and RSI

In [ ]:
```
for n in [14, 30, 50, 200]:
    hsi_px['MA' + '_' + str(n)] = talib.SMA(hsi_px['Close'].values, timeperiod=n)
    hsi_px['RSI' + '_' + str(n)] = talib.RSI(hsi_px['Close'].values, timeperiod=n)

    #Append List
    feature_names = feature_names + ['MA' + '_' + str(n), 'RSI' + '_' + str(n)]
```

Momentum

In [ ]:
```
hsi_px['Momentum_3'] = hsi_px['Close'].diff(3)
hsi_px['Momentum_5'] = hsi_px['Close'].diff(5)
hsi_px['Momentum_10'] = hsi_px['Close'].diff(10)

#Append List
Momentum_features = ['Momentum_3','Momentum_5','Momentum_10']
feature_names.extend(Momentum_features)
```

Volume

In [ ]:
```
hsi_px['Volume_1d_change'] = hsi_px['Volume'].pct_change()

#Append List
volume_features = ['Volume_1d_change']
feature_names.extend(volume_features)
```

Past returns

In [ ]:
```
hsi_px['Pass_Returns_3'] = hsi_px['Returns'].diff(3)
hsi_px['Pass_Returns_5'] = hsi_px['Returns'].diff(5)
hsi_px['Pass_Returns_10'] = hsi_px['Returns'].diff(10)

#Append List
returns_features = ['Pass_Returns_3','Pass_Returns_5','Pass_Returns_10']
feature_names.extend(returns_features)
feature_names
```

Out[ ]:
```
['H-L',
 'O-C',
 'MA_14',
 'RSI_14',
 'MA_30',
 'RSI_30',
 'MA_50',
 'RSI_50',
 'MA_200',
 'RSI_200',
 'Momentum_3',
 'Momentum_5',
 'Momentum_10',
 'Volume_1d_change',
 'Pass_Returns_3',
 'Pass_Returns_5',
 'Pass_Returns_10']
```

Summary of Factors & Charts

In [ ]:
```
hsi_px
```

Out[ ]:

	Open	High	Low	Close	Volume	Returns	Sign	O-C	H-L	MA_14	...	RSI_50	MA_200	RSI_200	Momentum_3	Momentum_5	Momentum_1
Date																	
2023-10-11	2296.38728	2304.77902	2287.80372	2288.23725	2029133726	-0.018579	0	8.15003	16.97530	NaN	...	NaN	NaN	NaN	NaN	NaN	Nal
2023-10-10	2267.00642	2291.30368	2255.67307	2259.09021	1596847248	-0.012820	0	7.91621	35.63061	NaN	...	NaN	NaN	NaN	NaN	NaN	Nal
2023-10-09	2234.92670	2251.88879	2228.20801	2237.10155	1221298004	-0.009781	0	-2.17485	23.68078	NaN	...	NaN	NaN	NaN	NaN	NaN	Nal
2023-10-06	2218.05470	2247.90581	2218.05470	2232.86086	1141209956	-0.001897	0	-14.80616	29.85111	NaN	...	NaN	NaN	NaN	-55.37639	NaN	Nal
2023-10-05	2202.72372	2213.76417	2196.03764	2198.11395	1088259898	-0.015684	0	4.60977	17.72653	NaN	...	NaN	NaN	NaN	-60.97626	NaN	Nal
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
2018-10-19	3210.43147	3283.31016	3200.00765	3260.09157	2100898688	-0.023027	0	-49.66010	83.30251	3249.191409	...	46.083348	3522.331560	48.614817	40.07244	107.13156	-125.0915
2018-10-18	3275.62292	3275.62292	3222.93923	3247.54086	1819304134	-0.003857	0	28.08206	52.68369	3242.582622	...	45.738483	3522.457700	48.515248	13.98512	59.91923	4.3390
2018-10-16	3264.79010	3280.86385	3224.00664	3248.98048	1512630563	0.000443	0	15.80962	56.85721	3236.442269	...	45.785969	3522.245770	48.527401	-87.05070	28.96135	63.4862
2018-10-15	3277.55672	3277.55672	3241.30829	3247.11723	1664833532	-0.000574	0	30.43949	36.24843	3231.809847	...	45.733112	3521.944130	48.512505	-12.97434	13.56149	113.1265
2018-10-12	3241.83651	3297.27012	3227.21715	3292.89643	2164240597	0.014000	1	-51.05992	70.05297	3225.217939	...	47.259584	3521.750596	48.899828	45.35557	-43.13475	129.3463

1230 rows × 24 columns

In [ ]:
```
hsi_px.describe().T
```

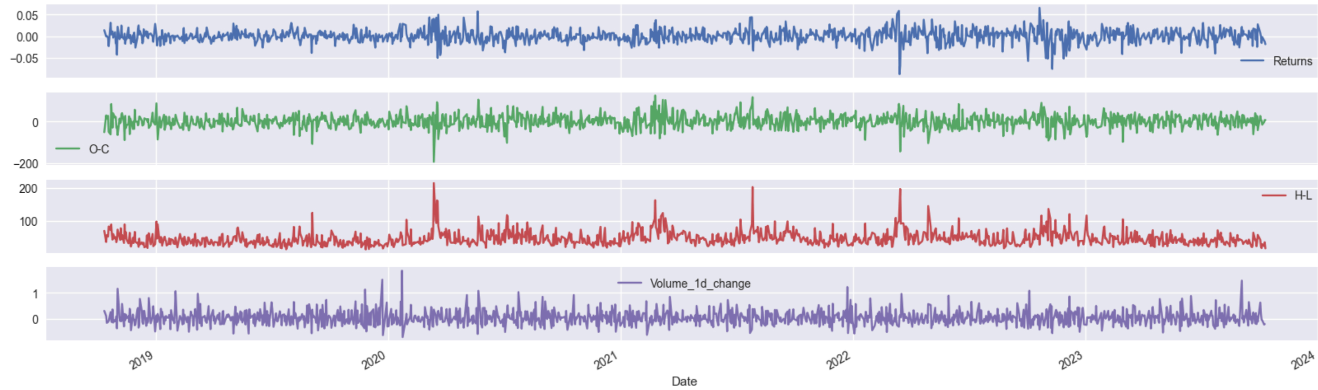
Out[ ]:

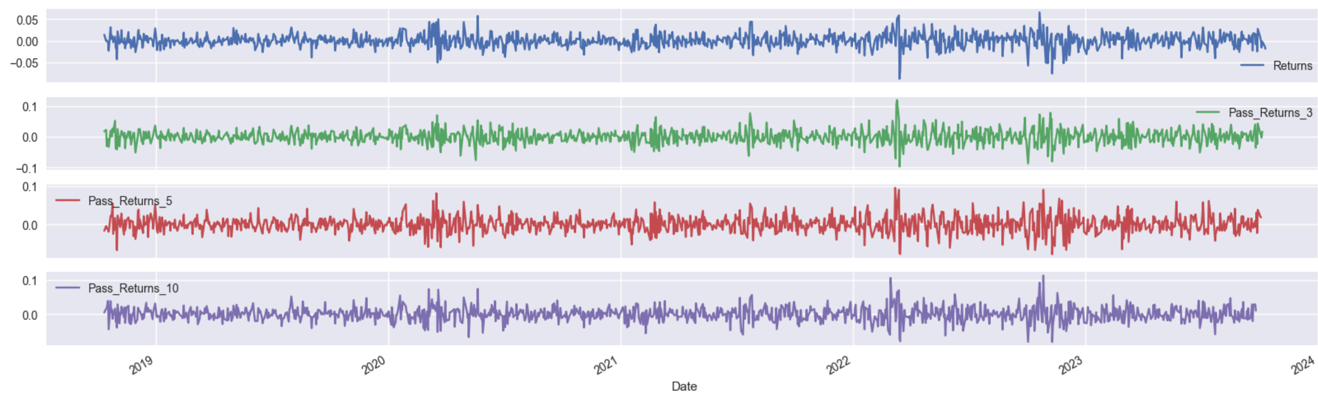
	count	mean	std	min	25%	50%	75%	max
Open	1230.0	3.112104e+03	4.773808e+02	1.889404e+03	2.663751e+03	3.217849e+03	3.472746e+03	4.022206e+03
High	1230.0	3.132995e+03	4.758159e+02	1.925389e+03	2.689686e+03	3.239672e+03	3.486965e+03	4.022206e+03
Low	1230.0	3.086283e+03	4.770671e+02	1.859672e+03	2.639201e+03	3.184912e+03	3.445927e+03	3.943162e+03
Close	1230.0	3.110484e+03	4.770717e+02	1.871101e+03	2.659102e+03	3.215347e+03	3.468698e+03	4.009667e+03
Volume	1230.0	2.164604e+09	7.818019e+08	4.839841e+08	1.677201e+09	2.003807e+09	2.487771e+09	6.012760e+09
Returns	1230.0	2.808181e-04	1.488117e-02	-8.791196e-02	-7.533236e-03	-2.308477e-04	8.179509e-03	6.569868e-02
Sign	1230.0	4.325203e-01	4.956271e-01	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00
O-C	1230.0	1.619635e+00	3.194051e+01	-1.947015e+02	-1.807519e+01	1.940420e+00	2.128644e+01	1.269453e+02
H-L	1230.0	4.671132e+01	2.166354e+01	1.372950e+01	3.208756e+01	4.183362e+01	5.626029e+01	2.141988e+02
MA_14	1217.0	3.114399e+03	4.707299e+02	1.992976e+03	2.669456e+03	3.228003e+03	3.470461e+03	3.882145e+03
RSI_14	1216.0	5.090725e+01	1.160776e+01	1.828110e+01	4.271427e+01	5.125549e+01	5.977896e+01	7.861674e+01
MA_30	1201.0	3.118942e+03	4.638491e+02	2.083219e+03	2.682394e+03	3.227870e+03	3.492301e+03	3.829622e+03
RSI_30	1200.0	5.111538e+01	7.491970e+00	2.909624e+01	4.611700e+01	5.151712e+01	5.697914e+01	6.675369e+01
MA_50	1181.0	3.123893e+03	4.558545e+02	2.162033e+03	2.669264e+03	3.224053e+03	3.510061e+03	3.781313e+03
RSI_50	1180.0	5.126890e+01	5.496695e+00	3.552140e+01	4.727274e+01	5.212838e+01	5.558756e+01	6.147958e+01
MA_200	1031.0	3.142997e+03	4.026844e+02	2.455204e+03	2.762657e+03	3.299724e+03	3.495795e+03	3.605651e+03
RSI_200	1030.0	5.162551e+01	2.109025e+00	4.664770e+01	4.983675e+01	5.165931e+01	5.324203e+01	5.612859e+01
Momentum_3	1227.0	2.448708e+00	7.526556e+01	-3.845007e+02	-4.513164e+01	2.527740e+00	5.130719e+01	3.452931e+02
Momentum_5	1225.0	4.147937e+00	9.621711e+01	-4.443831e+02	-5.949185e+01	1.336070e+00	6.595941e+01	3.775376e+02
Momentum_10	1220.0	8.243288e+00	1.339183e+02	-4.488178e+02	-8.148202e+01	9.362575e+00	8.965967e+01	6.474942e+02
Volume_1d_change	1229.0	3.768441e-02	2.862187e-01	-7.222440e-01	-1.457584e-01	1.114717e-02	1.806948e-01	1.864206e+00
Pass_Returns_3	1227.0	4.486526e-05	2.144821e-02	-9.714326e-02	-1.228785e-02	-2.021170e-04	1.219613e-02	1.196219e-01
Pass_Returns_5	1225.0	3.734472e-05	2.103773e-02	-7.819600e-02	-1.266869e-02	1.212152e-04	1.159414e-02	9.467694e-02
Pass_Returns_10	1220.0	6.124952e-05	2.084112e-02	-8.195740e-02	-1.203431e-02	-3.350284e-04	1.198325e-02	1.130859e-01

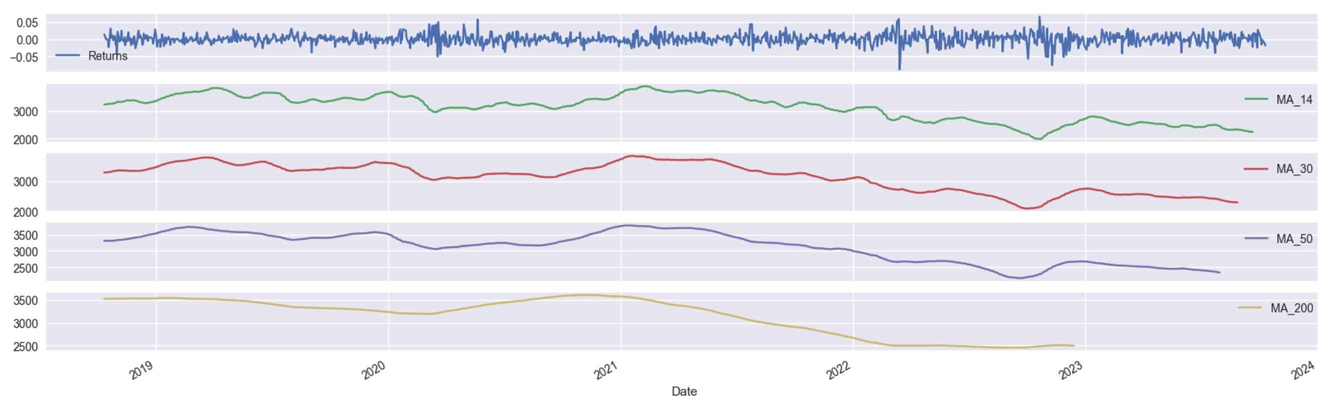
In [ ]:
```
hsi_px.plot(y=['Returns','O-C', 'H-L','Volume_1d_change'], subplots=True, figsize=(20, 6))
hsi_px.plot(y=['Returns', 'Pass_Returns_3', 'Pass_Returns_5', 'Pass_Returns_10'], subplots=True, figsize=(20, 6))
hsi_px.plot(y=['Returns', 'MA_14', 'MA_30','MA_50','MA_200'], subplots=True, figsize=(20, 6))
```

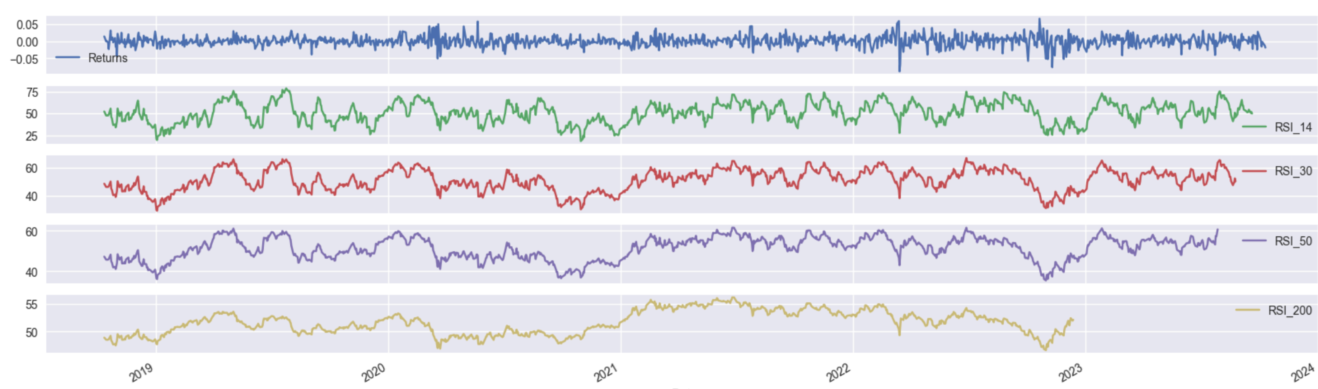
```
hsi_px.plot(y=['Returns', 'RSI_14', 'RSI_30','RSI_50','RSI_200'], subplots=True, figsize=(20, 6))
hsi_px.plot(y=['Returns', 'Momentum_3', 'Momentum_5', 'Momentum_10'], subplots=True, figsize=(20, 6))
```

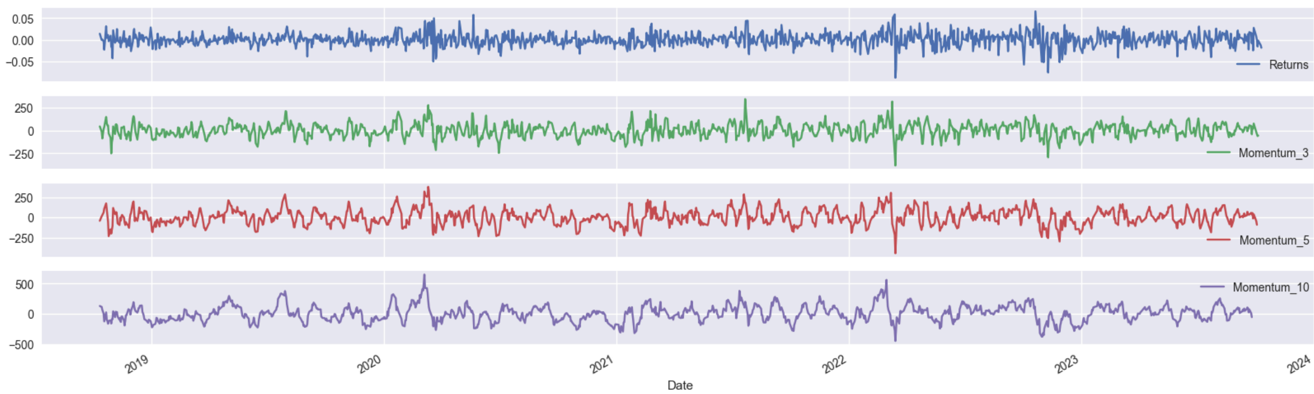
```
Out[ ]: array([<Axes: xlabel='Date'>, <Axes: xlabel='Date'>,
               <Axes: xlabel='Date'>, <Axes: xlabel='Date'>], dtype=object)
```











3. Question - What are voting classifiers in ensemble learning?

A Voting Classifier is an ensemble learning method that combines several base models to produce the final optimum solution. The base model can independently use different algorithms such as KNN, Random forests, Regression, etc., to predict individual outputs. This brings diversity in the output, thus called Heterogeneous ensembling. In contrast, if base models use the same algorithm to predict separate outcomes, this is called Homogeneous ensembling.

Voting Classifier supports two types of votings:

- In `Hard voting (majority voting)`, we predict the final class label as the class label that has been predicted most frequently by the classification models. The base model's classifiers are fed with the training data individually. The models predict the output class independent of each other. Suppose three classifiers predicted the output class(A, A, B), so here the majority predicted A as output. Hence A will be the final prediction.
- In `Soft voting`, we predict the class labels by averaging the class-probabilities (only recommended if the classifiers are well-calibrated). Classifiers or base models are fed with training data to predict the classes out of m possible courses. Each base model classifier independently assigns the probability of occurrence of each type. In the end, the average of the possibilities of each class is calculated, and the final output is the class having the highest probability. Suppose given some input to three models, the prediction probability for class A = (0.30, 0.47, 0.53) and B = (0.20, 0.32, 0.40). So the average for class A is 0.4333 and B is 0.3067, the winner is clearly class A because it had the highest probability averaged by each classifier.

Below I will show a sample python code to implement the Voting Classifier:

In [ ]:
```
# importing libraries
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn import model_selection
! pip install mlxtend
from mlxtend.classifier import EnsembleVoteClassifier


# visualization
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
import matplotlib.gridspec as gridspec
import itertools
```

```
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Requirement already satisfied: mlxtend in c:\python311\lib\site-packages (0.23.0)
Requirement already satisfied: scipy>=1.2.1 in c:\python311\lib\site-packages (from mlxtend) (1.11.0)
Requirement already satisfied: numpy>=1.16.2 in c:\python311\lib\site-packages (from mlxtend) (1.25.0)
Requirement already satisfied: pandas>=0.24.2 in c:\python311\lib\site-packages (from mlxtend) (2.0.2)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\python311\lib\site-packages (from mlxtend) (1.3.1)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from mlxtend) (3.7.1)
Requirement already satisfied: joblib>=0.13.2 in c:\python311\lib\site-packages (from mlxtend) (1.3.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (1.1.0)
Requirement already satisfied: cycler>=0.10 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (4.40.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (23.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (9.5.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (3.1.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\python311\lib\site-packages (from pandas>=0.24.2->mlxtend) (2023.3)
Requirement already satisfied: tzdata>=2022.1 in c:\python311\lib\site-packages (from pandas>=0.24.2->mlxtend) (2023.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\python311\lib\site-packages (from scikit-learn>=1.0.2->mlxtend) (3.2.0)
Requirement already satisfied: six>=1.5 in c:\users\lenovo\appdata\roaming\python\python311\site-packages (from python-dateutil>=2.7->matplotlib>=3.0.0->mlxtend) (1.16.0)
[notice] A new release of pip is available: 23.2.1 -> 23.3
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [ ]:
```
# Loading iris dataset
iris = datasets.load_iris()

X = iris.data[:, 1:3]
Y = iris.target

clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()

print('5-fold cross validation:\n')

labels = ['Logistic Regression', 'Random Forest', 'Naive Bayes']

for clf, label in zip([clf1, clf2, clf3], labels):

    scores = model_selection.cross_val_score(clf, X, Y,
                                              cv=5,
                                              scoring='accuracy')
```

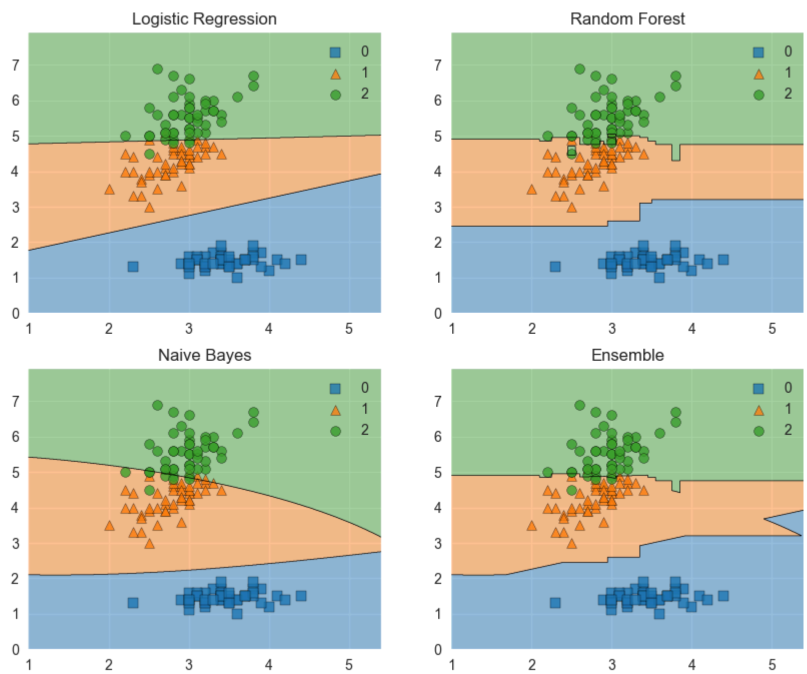
```
    print("Accuracy: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```

```
5-fold cross validation:

Accuracy: 0.95 (+/- 0.04) [Logistic Regression]
Accuracy: 0.94 (+/- 0.04) [Random Forest]
Accuracy: 0.91 (+/- 0.04) [Naive Bayes]
```

In [ ]:
```
eclf = EnsembleVoteClassifier(clfs=[clf1, clf2, clf3], weights=[1,1,1])

labels = ['Logistic Regression', 'Random Forest', 'Naive Bayes', 'Ensemble']
for clf, label in zip([clf1, clf2, clf3, eclf], labels):

    scores = model_selection.cross_val_score(clf, X, Y,
                                              cv=5,
                                              scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```

```
Accuracy: 0.95 (+/- 0.04) [Logistic Regression]
Accuracy: 0.94 (+/- 0.04) [Random Forest]
Accuracy: 0.91 (+/- 0.04) [Naive Bayes]
Accuracy: 0.95 (+/- 0.04) [Ensemble]
```

In [ ]:
```
gs = gridspec.GridSpec(2, 2)

fig = plt.figure(figsize=(10,8))

labels = ['Logistic Regression', 'Random Forest', 'Naive Bayes', 'Ensemble']
for clf, lab, grd in zip([clf1, clf2, clf3, eclf],
                         labels,
                         itertools.product([0, 1], repeat=2)):

    clf.fit(X, Y)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X, y=Y, clf=clf)
    plt.title(lab)
```



4. Question - Explain the role of the regularization parameter C in a Support Vector Machine (SVM) model. How does varying C affect the model's bias and variance trade-off?

The 'C' parameter controls the amount of regularization/penalty applied to the misclassified data and it determines the balance between achieving a low training error and allowing for misclassifications, affecting the generalization performance and the potential for overfitting or underfitting.

- `Larger` values of C mean `low regularizatio` and puts more emphasis on minimizing the training error, potentially leading to a narrower margin. The SVM algorithm seeks to fit the training data as accurately as possible, even if it means sacrificing a wider margin.
- This can be beneficial when the data points are not well-separated or when there is a significant presence of noise or outliers.
- However, setting C too large can increase the risk of `overfitting`, where the model becomes too specific to the training data and performs poorly on new, unseen data.

====================================

- `Smaller` values of C mean `higher regularization` and allows for a larger margin, potentially leading to more misclassifications on the training data.
- This can be useful in scenarios where the data points are well-separated, and there is a low presence of noise or outliers.
- However, it is important to be cautious as setting C too small can lead to `underfitting`, where the model fails to capture the underlying patterns in the data (may lead to `lower accuracy`).

4.1 Sample code using RBF Kernel to select the optimal C

There are several approaches to select the optimal C value: Grid Search, Randomized Search, Bayesian Optimization, Mataheuristic Algorithms and etc. Below, I use `RBF Kernel` to illustrate different performance of a trained SVM model under different C scenarios, from which we could see that in this case *lower C value allow the classifier to learn better under noisy data*.

In [ ]:
```
import math
from sklearn.model_selection import ParameterGrid, train_test_split
from sklearn.pipeline import Pipeline
from sklearn import metrics
```

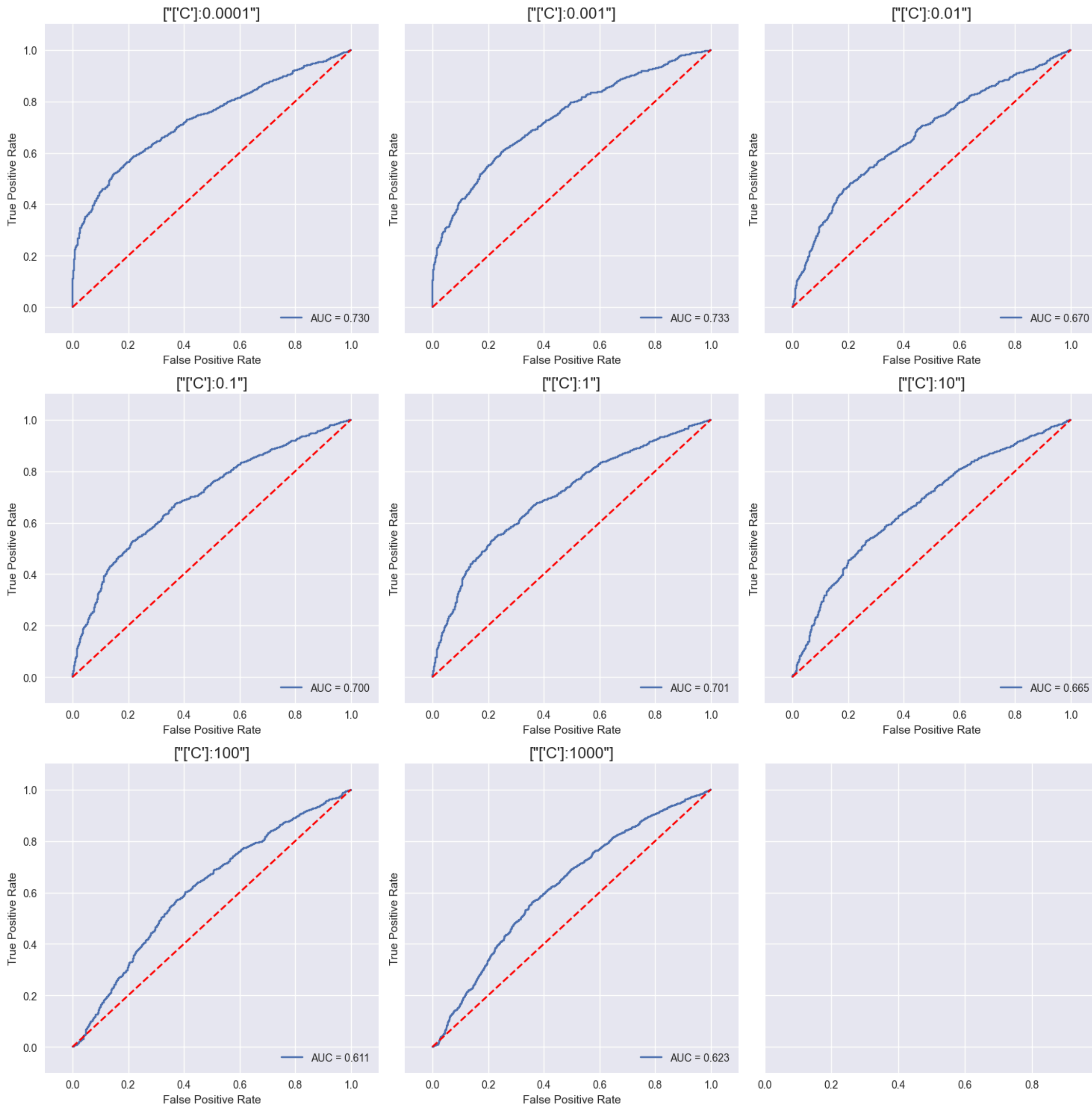
```python
from sklearn.datasets import make_classification

np.random.seed(222)
X, y = make_classification(
    n_samples=10000,
    n_features=10,
    n_informative=10,
    n_redundant=0,
    weights=[0.3,0.7],
    class_sep=0.7,
    flip_y=0.35) # the default value for flip_y is 0.01, or 1%
X_train, _ , y_train, _ = train_test_split(X, y, test_size=0.25)
```

In [ ]:
```python
np.random.seed(222)
X, y = make_classification(
    n_samples=10000,
    n_features=10,
    n_informative=10,
    n_redundant=0,
    weights=[0.3,0.7],
    class_sep=0.7,
    flip_y=0.0)
_, X_test , _ , y_test = train_test_split(X, y, test_size=0.25)
```

In [ ]:
```python
pipeline = Pipeline([
    ('prep',MinMaxScaler()),
    ('clf',SVC())
])


param_grid6 = [
    {
        'clf__kernel': ['rbf'],
        'clf__C':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
        'clf__gamma':['auto']
    }
]
```

In [ ]:
```python
%%time
num_cols = 3
num_rows = math.ceil(len(ParameterGrid(param_grid6)) / num_cols)

# create a single figure
plt.clf()
fig,axes = plt.subplots(num_rows,num_cols,sharey=True)
fig.set_size_inches(num_cols*5,num_rows*5)

for i,g in enumerate(ParameterGrid(param_grid6)):

    pipeline.set_params(**g)
    pipeline.fit(X_train,y_train)

    y_preds = pipeline.decision_function(X_test)

    # fpr means false-positive-rate
    # tpr means true-positive-rate
    fpr, tpr, _ = metrics.roc_curve(y_test, y_preds,pos_label=1)

    auc_score = metrics.auc(fpr, tpr)

    ax = axes[i // num_cols, i % num_cols]

    ax.set_title(str([r"{}:{}".format(
        k.split('__')[1:],v) for k,v in g.items() if "gamma" not in k and "kernel" not in k]),fontsize=15)
    ax.plot(fpr, tpr, label='AUC = {:.3f}'.format(auc_score))
    ax.legend(loc='lower right')

    # it's helpful to add a diagonal to indicate where chance
    # scores lie (i.e. just flipping a coin)
    ax.plot([0,1],[0,1],'r--')

    ax.set_xlim([-0.1,1.1])
    ax.set_ylim([-0.1,1.1])
    ax.set_ylabel('True Positive Rate')
    ax.set_xlabel('False Positive Rate')

plt.gcf().tight_layout()
plt.show()
```

```
<Figure size 800x550 with 0 Axes>
```




```
CPU times: total: 32.6 s
Wall time: 32.8 s
```

5. Pre-processing

5.1 Categorise extremely small near-zero returns

In my case, I set the return threshold as 0.20%, meaning that the returns below the threshold are labeled as negative. The sign of return is labeled as 1 if the return is positive, otherwise 0.

In [ ]:

```
print("The label 1 accounts for",
        round(hsi_px.query("Sign == 1").shape[0]/hsi_px.shape[0],4)*100,"%,",
"and the rest is label 0 about",
        round(hsi_px.query("Sign == 0").shape[0]/hsi_px.shape[0],4)*100,"%.")
```

```
The label 1 accounts for 43.25 %, and the rest is label 0 about 56.75 %.
```

5.2 MinMaxScaler

I use `MinMaxScaler` as the model's scaler. It uses the minimum and maximum values of a feature to rescale values to within a range and is commonly referred as normalization.

In [ ]:

```
hsi_px = hsi_px.dropna()
scaler = MinMaxScaler()
hsi_px[hsi_px.columns] = scaler.fit_transform(hsi_px[hsi_px.columns])
hsi_px
```

Out[ ]:

Date	Open	High	Low	Close	Volume	Returns	Sign	O-C	H-L	MA_14	...	RSI_50	MA_200	RSI_200	Momentum_3	Momentum_5	Momentum_10	Volume_1d_change
2022-12-13	0.287450	0.290539	0.303850	0.303544	0.379855	0.547529	0.0	0.550076	0.140302	0.280848	...	0.459019	0.038932	0.574579	0.556031	0.623494	0.398949	0.254779
2022-12-12	0.295696	0.291188	0.302872	0.295586	0.475913	0.528197	0.0	0.657664	0.157248	0.274937	...	0.436872	0.039993	0.557658	0.544182	0.559328	0.364464	0.358701
2022-12-09	0.287793	0.302667	0.303958	0.320605	0.640159	0.709983	1.0	0.438911	0.266032	0.274120	...	0.513485	0.041382	0.606515	0.563691	0.610956	0.451433	0.391944
2022-12-08	0.256548	0.277891	0.276891	0.293563	0.429061	0.423363	0.0	0.411529	0.288195	0.272735	...	0.438884	0.042539	0.549257	0.497615	0.550778	0.386458	0.167084
2022-12-07	0.281203	0.290704	0.266314	0.255013	0.872685	0.353907	0.0	0.831324	0.532138	0.268379	...	0.340350	0.043488	0.469258	0.407966	0.402699	0.371859	0.611256
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2018-10-19	0.619386	0.647611	0.643313	0.649496	0.292454	0.422402	0.0	0.450934	0.347051	0.664957	...	0.406883	0.927576	0.207482	0.581771	0.671007	0.295287	0.247073
2018-10-18	0.649952	0.643944	0.654319	0.643628	0.241522	0.547193	0.0	0.692634	0.194315	0.661458	...	0.393598	0.927686	0.196980	0.546025	0.613566	0.413347	0.227419
2018-10-16	0.644873	0.646444	0.654831	0.644301	0.186053	0.575189	0.0	0.654479	0.215134	0.658208	...	0.395427	0.927502	0.198262	0.407581	0.575900	0.467298	0.214069
2018-10-15	0.650859	0.644867	0.663135	0.643430	0.213582	0.568569	0.0	0.699963	0.112331	0.655756	...	0.393391	0.927240	0.196691	0.509084	0.557164	0.512577	0.318145
2018-10-12	0.634111	0.654268	0.656372	0.664836	0.303911	0.663443	1.0	0.446582	0.280958	0.652267	...	0.452196	0.927071	0.237544	0.589011	0.488184	0.527372	0.395220

1030 rows × 24 columns

5.3 Splitting data into training and testing set

Since index prices are typical time series data hence I don't split the dataset randomly. If I do random split here then the training dataset may contain future function - simply speaking X contains Y. This reflects in the experiment would be that the accuracy score becomes very high and leads to misleading conclusion. So in my project, I will just split the data based on the year.

In [ ]:
```
# Split dataset
hsi_px['Date'] = hsi_px.index
train_data = hsi_px[hsi_px['Date']<'20220101']
test_data = hsi_px[hsi_px['Date']>='20220101']

#Set up Training and Testing Dataset
X_train = train_data[['Open', 'High', 'Low', 'Close', 'Volume', 'O-C','H-L', 'MA_14', 'RSI_14', 'MA_30', 'RSI_30', 'MA_50', 'RSI_50','MA_200', 'RSI_200', 'Momentum_3', 'Momentum_5', 'M
Y_train = train_data['Sign'].values

X_test = test_data[['Open', 'High', 'Low', 'Close', 'Volume', 'O-C','H-L', 'MA_14', 'RSI_14', 'MA_30', 'RSI_30', 'MA_50', 'RSI_50','MA_200', 'RSI_200', 'Momentum_3', 'Momentum_5', 'Mom
Y_test = test_data['Sign'].values
```

6. Model Building

Next, I will use `Random Forest Classifier` to produce a model to predict positive moves.

6.1 Fit Model - Random Forest Classifier (Default Parameters)

In [ ]:
```
# Fitting the classifier into the Training set
clf = RandomForestClassifier(random_state=1)
clf.fit(X_train, Y_train)
```

Out[ ]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=1)
```

6.2 Predict Model

In [ ]:
```
# Predicting the test set results
Y_pred = clf.predict(X_test)
a = pd.DataFrame()
a['Y_pred'] = list(Y_pred)
a['Y_test'] = list(Y_test)
a
```

Out[ ]:

	Y_pred	Y_test
0	0.0	0.0
1	0.0	0.0
2	1.0	1.0
3	0.0	0.0
4	0.0	0.0
...	...	...
230	0.0	0.0
231	0.0	0.0
232	0.0	0.0
233	1.0	1.0
234	0.0	0.0

235 rows × 2 columns

In [ ]:
```
print(f'Train Accuracy: {accuracy_score(Y_train,clf.predict(X_train))}, Test Accuracy: {accuracy_score(Y_test,clf.predict(X_test))}')
```

```
Train Accuracy: 1.0, Test Accuracy: 0.8212765957446808
```

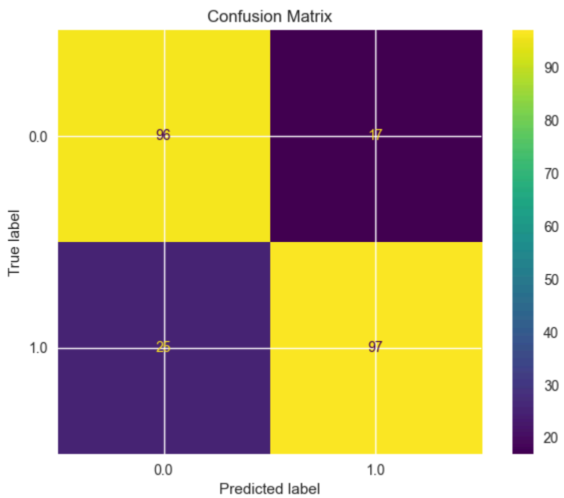
6.3 Prediction Quality

6.3.1 Confusion Matrix

In [ ]:
```
# Confusion Matrix for binary classification
tn, fp, fn, tp = confusion_matrix(Y_test, Y_pred).ravel()
print(tn, fp, fn, tp)
```

```
96 17 25 97
```

In [ ]:
```
# import matplotlib.pyplot as plt
# from sklearn.metrics import ConfusionMatrixDisplay
# from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred, labels=clf.classes_)
color = 'white'
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()
plt.title('Confusion Matrix')
plt.show()
```



6.3.2 Classification Report

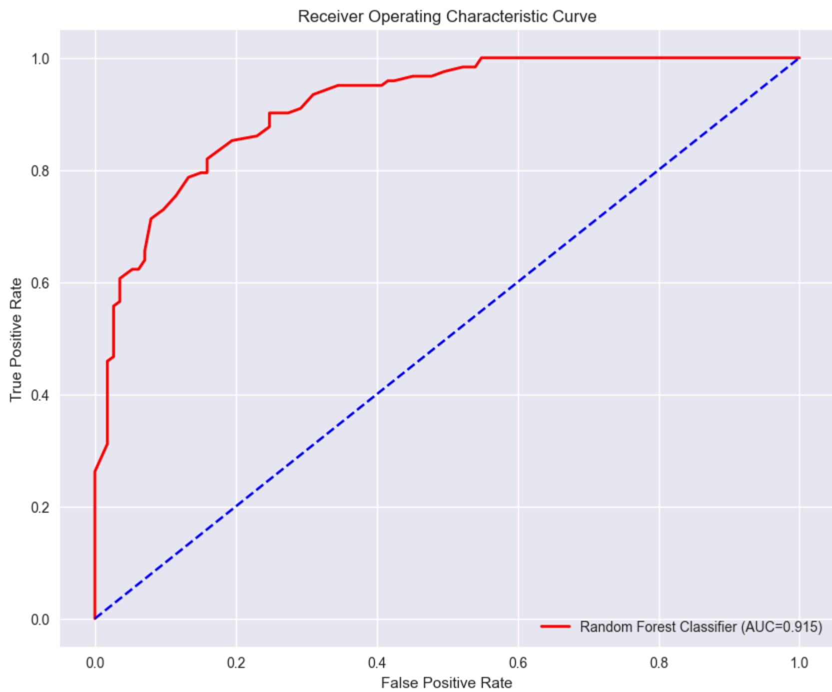
In [ ]:
```
# from sklearn.metrics import classification_report
print("Classification Report is:")
print(classification_report(Y_test, Y_pred))
```

```
Classification Report is:
              precision    recall  f1-score   support

         0.0       0.79      0.85      0.82       113
         1.0       0.85      0.80      0.82       122

    accuracy                           0.82       235
   macro avg       0.82      0.82      0.82       235
weighted avg       0.82      0.82      0.82       235
```

6.3.3 ROC Curve

In [ ]:
```
#首先我们使用建立好的模型对测试集数据进行预测预测的概率
score = clf.predict_proba(X_test)[:,1]

#使用roc_curve方法得到三个模型的真正率TP,假正率FP和阈值threshold
fpr,tpr,thres = roc_curve(Y_test,score,)

print("AUC is:",auc(fpr,tpr))

#创建画布
fig,ax = plt.subplots(figsize=(10,8))

#自定义标签名称label=''
ax.plot(fpr,tpr,linewidth=2,
        label='Random Forest Classifier (AUC={})'.format(str(round(auc(fpr,tpr),3))),color='red')

#绘制对角线
ax.plot([0,1],[0,1],linestyle='--',color='blue')

#调整字体大小
plt.legend(fontsize=12)
plt.legend(loc="lower right")

#调整标题
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve')

plt.show()
```

```
AUC is: 0.9154939793993906
```



7. Hyper-parameter Tuning & Best Model

7.1 Correlation Matrix

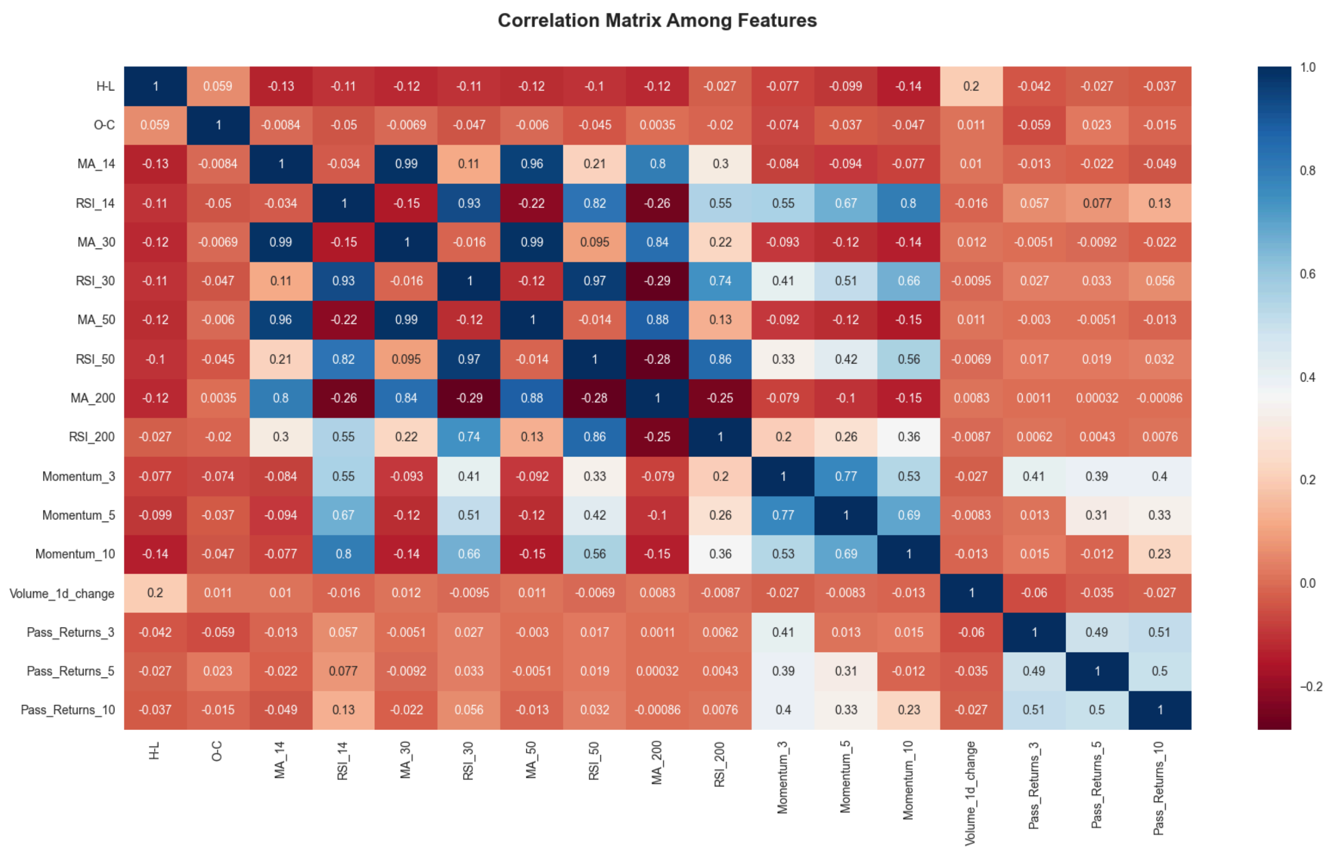
This step is to analyze the correlation among features since correlation bias affect the predictions in Machine Learning. The interplay between variables with high correlation can reduce the significance of them hence may cause the misleading result.

In [ ]:
```
pd.options.display.float_format = '{:,.2f}'.format ## 指定小数位数
hsi_px = hsi_px[feature_names]
hsi_corr_matrix = hsi_px.corr()
hsi_corr_matrix
```

Out[ ]:

	H-L	O-C	MA_14	RSI_14	MA_30	RSI_30	MA_50	RSI_50	MA_200	RSI_200	Momentum_3	Momentum_5	Momentum_10	Volume_1d_change	Pass_Returns_3	Pass_Returns_5	Pa
H-L	1.00	0.06	-0.13	-0.11	-0.12	-0.11	-0.12	-0.10	-0.12	-0.03	-0.08	-0.10	-0.14	0.20	-0.04	-0.03	
O-C	0.06	1.00	-0.01	-0.05	-0.01	-0.05	-0.01	-0.04	0.00	-0.02	-0.07	-0.04	-0.05	0.01	-0.06	0.02	
MA_14	-0.13	-0.01	1.00	-0.03	0.99	0.11	0.96	0.21	0.80	0.30	-0.08	-0.09	-0.08	0.01	-0.01	-0.02	
RSI_14	-0.11	-0.05	-0.03	1.00	-0.15	0.93	-0.22	0.82	-0.26	0.55	0.55	0.67	0.80	-0.02	0.06	0.08	
MA_30	-0.12	-0.01	0.99	-0.15	1.00	-0.02	0.99	0.10	0.84	0.22	-0.09	-0.12	-0.14	0.01	-0.01	-0.01	
RSI_30	-0.11	-0.05	0.11	0.93	-0.02	1.00	-0.12	0.97	-0.29	0.74	0.41	0.51	0.66	-0.01	0.03	0.03	
MA_50	-0.12	-0.01	0.96	-0.22	0.99	-0.12	1.00	-0.01	0.88	0.13	-0.09	-0.12	-0.15	0.01	-0.00	-0.01	
RSI_50	-0.10	-0.04	0.21	0.82	0.10	0.97	-0.01	1.00	-0.28	0.86	0.33	0.42	0.56	-0.01	0.02	0.02	
MA_200	-0.12	0.00	0.80	-0.26	0.84	-0.29	0.88	-0.28	1.00	-0.25	-0.08	-0.10	-0.15	0.01	0.00	0.00	
RSI_200	-0.03	-0.02	0.30	0.55	0.22	0.74	0.13	0.86	-0.25	1.00	0.20	0.26	0.36	-0.01	0.01	0.00	
Momentum_3	-0.08	-0.07	-0.08	0.55	-0.09	0.41	-0.09	0.33	-0.08	0.20	1.00	0.77	0.53	-0.03	0.41	0.39	
Momentum_5	-0.10	-0.04	-0.09	0.67	-0.12	0.51	-0.12	0.42	-0.10	0.26	0.77	1.00	0.69	-0.01	0.01	0.31	
Momentum_10	-0.14	-0.05	-0.08	0.80	-0.14	0.66	-0.15	0.56	-0.15	0.36	0.53	0.69	1.00	-0.01	0.02	-0.01	
Volume_1d_change	0.20	0.01	0.01	-0.02	0.01	-0.01	0.01	-0.01	0.01	-0.01	-0.03	-0.01	-0.01	1.00	-0.06	-0.03	
Pass_Returns_3	-0.04	-0.06	-0.01	0.06	-0.01	0.03	-0.00	0.02	0.00	0.01	0.41	0.01	0.02	-0.06	1.00	0.49	
Pass_Returns_5	-0.03	0.02	-0.02	0.08	-0.01	0.03	-0.01	0.02	0.00	0.00	0.39	0.31	-0.01	-0.03	0.49	1.00	
Pass_Returns_10	-0.04	-0.01	-0.05	0.13	-0.02	0.06	-0.01	0.03	-0.00	0.01	0.40	0.33	0.23	-0.03	0.51	0.50	

In [ ]:
```
plt.figure(figsize=(20, 10))
sns.heatmap(hsi_corr_matrix, annot=True, cmap='RdBu', xticklabels=1, yticklabels=1)
plt.title(
    'Correlation Matrix Among Features',
    fontsize=16, y=1.05, weight='bold'
)
plt.show()
```



In [ ]:
```python
# setting correlation maximum threshold
corr_max = 0.80

# detecting pairwise features with abs(corr) > rho_max
Pairwise_item = np.where(abs(hsi_corr_matrix) >= corr_max)
Pairwise_list = [
    [hsi_corr_matrix.index[x], hsi_corr_matrix.columns[y], round(hsi_corr_matrix.iloc[x, y], 3)]
    for x, y in zip(*Pairwise_item) if x != y and x < y
]
print("List Pairwise Features surpass the threshold 0.80 :")
Pairwise_list
```

```
List Pairwise Features surpass the threshold 0.80 :
```

Out[ ]:
```
[['MA_14', 'MA_30', 0.985],
 ['MA_14', 'MA_50', 0.958],
 ['MA_14', 'MA_200', 0.804],
 ['RSI_14', 'RSI_30', 0.926],
 ['RSI_14', 'RSI_50', 0.819],
 ['RSI_14', 'Momentum_10', 0.803],
 ['MA_30', 'MA_50', 0.987],
 ['MA_30', 'MA_200', 0.84],
 ['RSI_30', 'RSI_50', 0.969],
 ['MA_50', 'MA_200', 0.878],
 ['RSI_50', 'RSI_200', 0.855]]
```

We can clearly distinguish two groups of highly corerlated features, and two pairwise. These subsets represent the `informational clusters` of the linear relationships among features.

- `['MA_14', 'MA_30', 'MA_50', 'MA_200']`
- `['RSI_14', 'RSI_30', 'RSI_50']`
- `['RSI_50', 'RSI_200']`
- `['RSI_14', 'Momentum_10']`

Before we decide to eliminate any factor, we need to check the join effects and the hierarchical importance of the features - influence of the features as a whole set of informative characteristics to the model. In my project, I will use SHapley Additive exPlanations (SHAP) for the join effects to decide which factors to keep/remove to improve model performance.

7.2 Feature Importance - RFC

In [ ]:
```python
# Adjust Test Dataset to include only features
Y_test_RFC = test_data['Sign'].values
X_test_RFC = test_data[feature_names].values

Y_train_RFC = train_data['Sign'].values
X_train_RFC = train_data[feature_names].values
```

In [ ]:
```python
# Fitting the classifier into the Training set
clf_RFC = RandomForestClassifier(random_state=1)
clf_RFC.fit(X_train_RFC, Y_train_RFC)
```

Out[ ]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=1)
```

In [ ]:
```python
#分析特征变量的特征重要性
c = pd.DataFrame()
importances = clf_RFC.feature_importances_
features = test_data[feature_names].columns
```

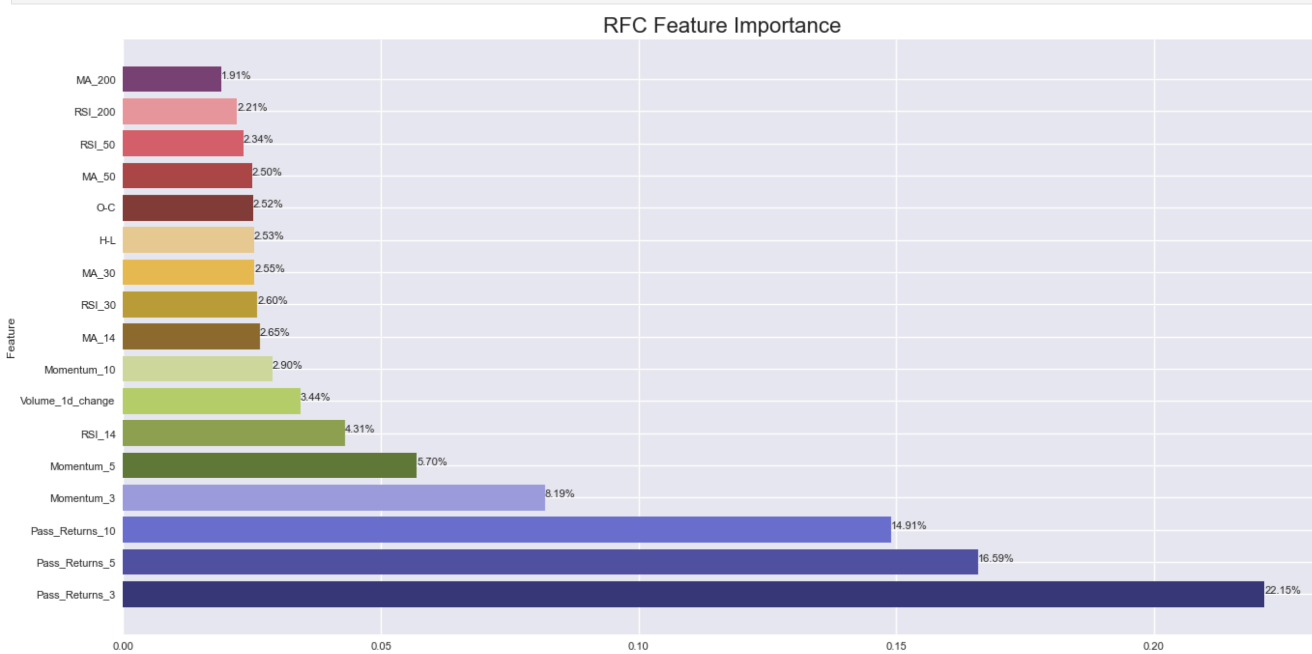
```
c['Feature Importance'] = importances
c['Feature'] = features

c.index = features
c = c.sort_values('Feature Importance', ascending=False)
c
```

Out[ ]:

	Feature Importance	Feature
Pass_Returns_3	0.22	Pass_Returns_3
Pass_Returns_5	0.17	Pass_Returns_5
Pass_Returns_10	0.15	Pass_Returns_10
Momentum_3	0.08	Momentum_3
Momentum_5	0.06	Momentum_5
RSI_14	0.04	RSI_14
Volume_1d_change	0.03	Volume_1d_change
Momentum_10	0.03	Momentum_10
MA_14	0.03	MA_14
RSI_30	0.03	RSI_30
MA_30	0.03	MA_30
H-L	0.03	H-L
O-C	0.03	O-C
MA_50	0.03	MA_50
RSI_50	0.02	RSI_50
RSI_200	0.02	RSI_200
MA_200	0.02	MA_200

In [ ]:

```
# Plot the chart
fig,ax=plt.subplots(figsize=(20, 10), dpi=80)
b=ax.barh(range(len(c['Feature'])),c['Feature Importance'],color=plt.get_cmap('tab20b')(range(20)),)

#设置y轴标签
plt.ylabel('Feature')
plt.xlabel('Feature Importance')

#设置Y轴刻度线标签
ax.set_yticks(range(len(c['Feature'])))
ax.set_yticklabels(c['Feature'], fontsize = 10)
plt.rc('font', size=30)
plt.rc('axes', titlesize=20)

#添加数据标签
lis = list(c['Feature Importance'])
lens = []
for i in range(len(lis)):
    lens.append(i)
for a,b in zip(lis,lens):
    plt.text(a,b+0.1,'{:.2f}%'.format(a*100),ha = 'left',va = 'center',fontsize=10)

plt.title('RFC Feature Importance')
plt.show()
```



7.3 Feature Importance - Shapley Additive exPlanations (SHAP) Method

In [ ]:

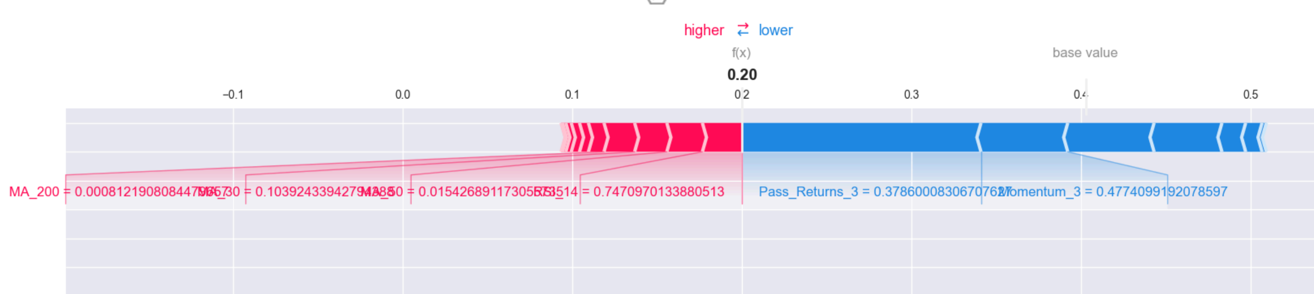
```
# Create Tree Explainer object that can calculate shap values
explainer = shap.TreeExplainer(clf_RFC)
```

In [ ]:
```python
#Let's choose some instances from the test dataset to understand to the classifier makes predictions for them.
choosen_instance = test_data[feature_names].loc[['2022-09-07']]
choosen_instance
```

Out[ ]:

Date	H-L	O-C	MA_14	RSI_14	MA_30	RSI_30	MA_50	RSI_50	MA_200	RSI_200	Momentum_3	Momentum_5	Momentum_10	Volume_1d_change	Pass_Returns_3	Pass_Returns_5	Pass_Returns_10
2022-09-07	0.09	0.55	0.20	0.75	0.10	0.73	0.02	0.66	0.00	0.48	0.48	0.56	0.51	0.31	0.38	0.46	0.41

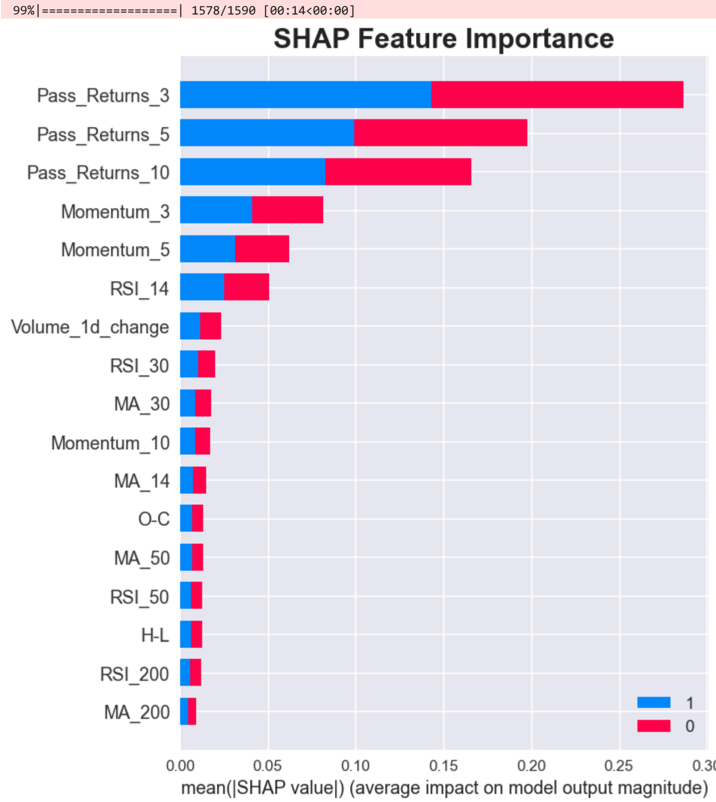
In [ ]:
```python
# Calculate Shap values
shap_values = explainer.shap_values(choosen_instance)
shap.initjs()
shap.force_plot(explainer.expected_value[1], shap_values[1], choosen_instance,matplotlib=True)
```



- Feature values in pink cause to increase the prediction. Size of the bar shows the magnitude of the feature's effect.
- Feature values in blue cause to decrease the prediction.
- Sum of all feature SHAP values explain why model prediction was different from the baseline.
- Model predicted 0.17, whereas the base_value is 0.40. Biggest effect is Pass_Returns_3; This has decreased the chances of being marked as Positive Sign. Next, Pass_Returns_10 also decreases the positive sign probability while RSI_14 increases the chance of being marked as positive sign.

In [ ]:
```python
# defining a sample of 100 datapoints to serve as background dist
backDist = shap.utils.sample(X_train_RFC, 100, random_state=42)
# defining the SHAP tree explainer
explainer = shap.TreeExplainer(clf_RFC, backDist)
# computing the SHAP values for each feature
shap_values = explainer.shap_values(X_train_RFC)
# plotting the SHAP summary bar plot
shap.summary_plot(
    shap_values, hsi_px.values,
        plot_type="bar", class_names= ["0", "1"], feature_names = feature_names, show=False
)
plt.title("SHAP Feature Importance", weight='bold')
plt.show()
```

```
99%|==================| 1578/1590 [00:14<00:00]
```



In general, the feature importance ranking results based on SHAP technology and the feature importance ranking results of random forests are consistent. Because SHAP technology calculates the importance of features based on the concept of Shapley values, feature importance in a random forest is achieved by calculating the impurity that each feature can reduce when it splits on the tree. Both methods determine the importance of features from different angles, but both can provide similar results.

However, sometimes SHAP techniques and random forests may differ in the ranking of feature importance. This may be because `SHAP technology is able to calculate the impact of each feature on the prediction outcome at a more granular level`, whereas the feature importance of a random forest simply estimates the importance of each feature. In addition, `SHAP technology can detect interactions between features, providing a more comprehensive ranking of feature importance`. Therefore, while SHAP techniques and random forests typically agree on feature importance ranking results, SHAP techniques can provide more granular and comprehensive feature importance ranking results.

In this case, if we compare the chart with earlier generated "RFC Feature Importance" can observe that:

1. the ranking becomes different since the 7th feature.
2. they share same 5 most important features: `Pass_Returns_3`, `Pass_Returns_5`, `Pass_Returns_10`, `Momentum_3` and `Momentum_5`. Since none of them are listed as highly correlated feature in "Correlation Matrix Among Features" chart, I will include all of them in the reduced features dataset.
3. Since RSI_14 and Momentum_10 are highly correlated so I will only include one of them.
4. `Volume_1d_change` has low correlation with other factors and has high importance itself so will be included.
5. Most "MA_" features are highly correlated so I will select one of them. Considering 3, I will select `Momentum_10`, `RSI_30` and `MA_30`
6. Last but not the least, I will consider these insignificant but low correlated features: `H-L`, `O-C` and `RSI_200`

To conclude, our reduced features dataset will include:

In [ ]:
```
reduced_feature_names_SHAP = ['Pass_Returns_3','Pass_Returns_5','Pass_Returns_10','Momentum_3','Momentum_5','Volume_1d_change','Momentum_10','RSI_30','MA_30','H-L','O-C','RSI_200']
reduced_feature_names_SHAP
```

Out[ ]:
```
['Pass_Returns_3',
 'Pass_Returns_5',
 'Pass_Returns_10',
 'Momentum_3',
 'Momentum_5',
 'Volume_1d_change',
 'Momentum_10',
 'RSI_30',
 'MA_30',
 'H-L',
 'O-C',
 'RSI_200']
```

7.4 Predict Model & Prediction Quality - Modified model (SHAP)

In [ ]:
```
#Set up Training and Testing Dataset
X_train_SHAP = train_data[reduced_feature_names_SHAP]
Y_train_SHAP = train_data['Sign']

X_test_SHAP = test_data[reduced_feature_names_SHAP]
Y_test_SHAP = test_data['Sign']
```

In [ ]:
```
clf_SHAP = RandomForestClassifier(random_state=1)
clf_SHAP.fit(X_train_SHAP, Y_train_SHAP)
```

Out[ ]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=1)
```

In [ ]:
```
# Predicting the results
Y_pred_SHAP = clf_SHAP.predict(X_test_SHAP)
a_SHAP = pd.DataFrame()
a_SHAP['Y_pred_SHAP'] = list(Y_pred_SHAP)
a_SHAP['Y_test_SHAP'] = list(Y_test_SHAP)
a_SHAP
```

Out[ ]:

	Y_pred_SHAP	Y_test_SHAP
0	0.00	0.00
1	0.00	0.00
2	1.00	1.00
3	0.00	0.00
4	0.00	0.00
...	...	...
230	0.00	0.00
231	0.00	0.00
232	0.00	0.00
233	1.00	1.00
234	0.00	0.00

235 rows × 2 columns

In [ ]:
```
print(f'Train Accuracy (SHAP): {accuracy_score(Y_train_SHAP,clf_SHAP.predict(X_train_SHAP))}, Test Accuracy: {accuracy_score(Y_test_SHAP,clf_SHAP.predict(X_test_SHAP))}')
```

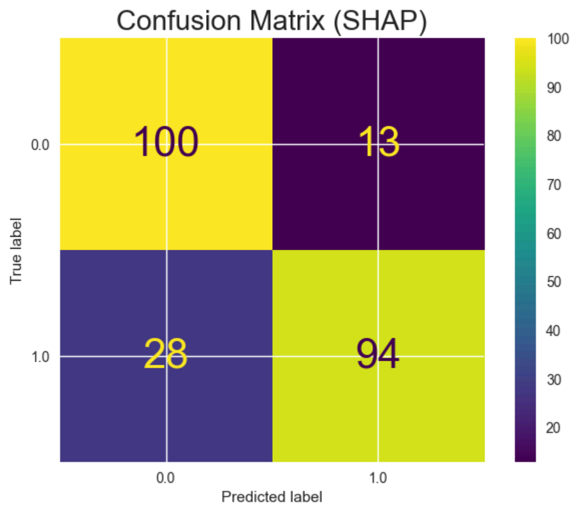
```
Train Accuracy (SHAP): 1.0, Test Accuracy: 0.825531914893617
```

7.4.1 Confusion Matrix

In [ ]:
```
tn_SHAP, fp_SHAP, fn_SHAP, tp_SHAP = confusion_matrix(Y_test_SHAP, Y_pred_SHAP).ravel()
print(tn_SHAP, fp_SHAP, fn_SHAP, tp_SHAP)
```

```
100 13 28 94
```

In [ ]:
```
cm_SHAP = confusion_matrix(Y_test_SHAP, Y_pred_SHAP, labels=clf_SHAP.classes_)
color = 'white'
disp = ConfusionMatrixDisplay(confusion_matrix=cm_SHAP, display_labels=clf_SHAP.classes_)
disp.plot()
plt.title('Confusion Matrix (SHAP)')
plt.show()
```



7.4.2 Classification Report

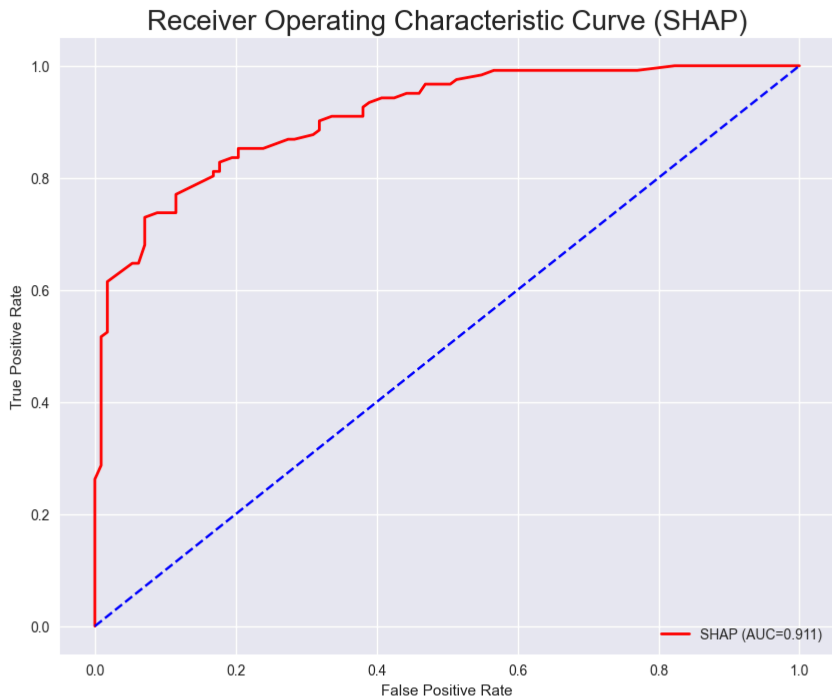
In [ ]:
```
print("Classification Report (SHAP) is:")
print(classification_report(Y_test_SHAP, Y_pred_SHAP))
```

```
Classification Report (SHAP) is:
              precision    recall  f1-score   support

         0.0       0.78      0.88      0.83       113
         1.0       0.88      0.77      0.82       122

    accuracy                           0.83       235
   macro avg       0.83      0.83      0.83       235
weighted avg       0.83      0.83      0.83       235
```

7.4.3 ROC Curve

In [ ]:
```
#首先我们使用建立好的模型对测试集数据进行预测预测的概率
score_SHAP = clf_SHAP.predict_proba(X_test_SHAP)[:,1]

#使用roc_curve方法得到三个模型的真正率TP,假正率FP和阈值threshold
fpr_SHAP,tpr_SHAP,thres_SHAP = roc_curve(Y_test_SHAP,score_SHAP,)

print("AUC (SHAP) is:",auc(fpr_SHAP,tpr_SHAP))

#创建画布
fig_SHAP,ax_SHAP = plt.subplots(figsize=(10,8))

#自定义标签名称label=''
ax_SHAP.plot(fpr_SHAP,tpr_SHAP,linewidth=2,
        label='SHAP (AUC={})'.format(str(round(auc(fpr_SHAP,tpr_SHAP),3))),color='red')

#绘制对角线
ax_SHAP.plot([0,1],[0,1],linestyle='--',color='blue')

#调整字体大小
plt.legend(fontsize=12)
plt.legend(loc="lower right")

#调整标题
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve (SHAP)')

plt.show()
```

```
AUC (SHAP) is: 0.9105614391411577
```



7.5 Predict Model & Prediction Quality - Dropped RSI_200

Now, I will drop the least important feature from the previous reduced feature set from the model, rebuild the model and check its effect on accuracy.

In [ ]:
```
# New Feature Set
reduced_feature_names_NoRSI200 = ['Pass_Returns_3','Pass_Returns_5','Pass_Returns_10','Momentum_3','Momentum_5','Volume_1d_change','Momentum_10','RSI_30','MA_30','H-L','O-C']

# Set up Training and Testing Dataset
X_train_NoRSI200 = train_data[reduced_feature_names_NoRSI200]
Y_train_NoRSI200 = train_data['Sign']

X_test_NoRSI200 = test_data[reduced_feature_names_NoRSI200]
Y_test_NoRSI200 = test_data['Sign']

# Classfier
clf_NoRSI200 = RandomForestClassifier(random_state=1)
clf_NoRSI200.fit(X_train_NoRSI200, Y_train_NoRSI200)

# Predicting the results
Y_pred_NoRSI200 = clf_NoRSI200.predict(X_test_NoRSI200)
a_NoRSI200 = pd.DataFrame()
a_NoRSI200['Y_pred_NoRSI200'] = list(Y_pred_NoRSI200)
a_NoRSI200['Y_test_NoRSI200'] = list(Y_test_NoRSI200)
a_NoRSI200
```

Out[ ]:

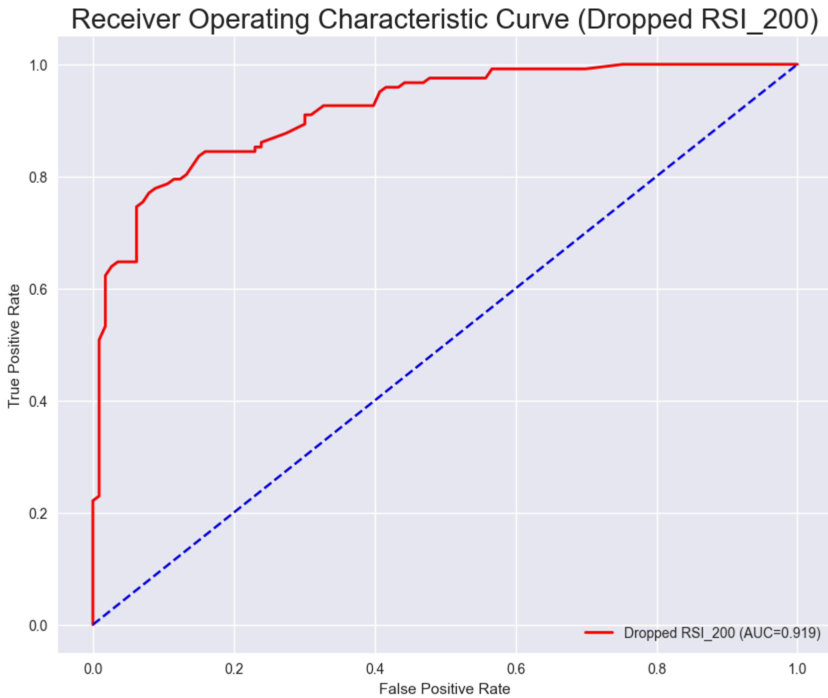
	Y_pred_NoRSI200	Y_test_NoRSI200
0	0.00	0.00
1	0.00	0.00
2	1.00	1.00
3	0.00	0.00
4	0.00	0.00
...	...	...
230	0.00	0.00
231	0.00	0.00
232	0.00	0.00
233	1.00	1.00
234	0.00	0.00

235 rows × 2 columns

In [ ]:
```
#首先我们使用建立好的模型对测试集数据进行预测预测的概率
score_NoRSI200 = clf_NoRSI200.predict_proba(X_test_NoRSI200)[:,1]

#使用roc_curve方法得到三个模型的真正率TP,假正率FP和阈值threshold
fpr_NoRSI200,tpr_NoRSI200,thres_NoRSI200 = roc_curve(Y_test_NoRSI200,score_NoRSI200,)

print("AUC (Dropped RSI_200) is:",auc(fpr_NoRSI200,tpr_NoRSI200))

#创建画布
fig_NoRSI200,ax_NoRSI200 = plt.subplots(figsize=(10,8))

#自定义标签名称label=''
ax_NoRSI200.plot(fpr_NoRSI200,tpr_NoRSI200,linewidth=2,
        label='Dropped RSI_200 (AUC={})'.format(str(round(auc(fpr_NoRSI200,tpr_NoRSI200),3))),color='red')

#绘制对角线
ax_NoRSI200.plot([0,1],[0,1],linestyle='--',color='blue')
```

```
#调整字体大小
plt.legend(fontsize=12)
plt.legend(loc="lower right")

#调整标题
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve (Dropped RSI_200)')

plt.show()
```

AUC (Dropped RSI_200) is: 0.9189032351661105



7.6 Predict Model & Prediction Quality - RFC model with different n_estimators [GridSearchCV]

In [ ]:
```
parameters = {'n_estimators':range(10,200,10)}   #需要调优的参数
new_model = RandomForestClassifier(random_state=1) #原模型
grid_search = GridSearchCV(new_model, parameters, n_jobs=-1 , scoring='f1_macro')
grid_search.fit(X_train_NoRSI200, Y_train_NoRSI200)
grid_search.best_params_
print('Best n_estimators parameter:  ' + str(grid_search.best_params_))
```

Best n_estimators parameter: {'n_estimators': 130}

In [ ]:
```
# Classfier
clf_best = RandomForestClassifier(n_estimators=130 , random_state=1)
clf_best.fit(X_train_NoRSI200, Y_train_NoRSI200)

# Predicting the results
Y_pred_best = clf_best.predict(X_test_NoRSI200)
a_best = pd.DataFrame()
a_best['Y_pred_best'] = list(Y_pred_best)
a_best['Y_test_best'] = list(Y_test_NoRSI200)
a_best
```

Out[ ]:

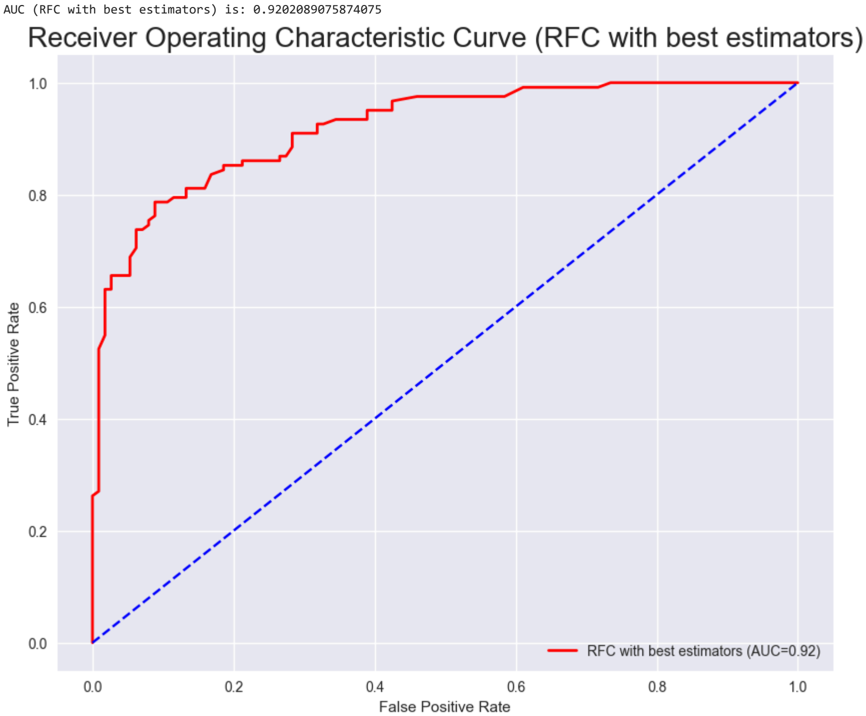
	Y_pred_best	Y_test_best
0	0.00	0.00
1	0.00	0.00
2	1.00	1.00
3	0.00	0.00
4	0.00	0.00
...	...	...
230	0.00	0.00
231	0.00	0.00
232	0.00	0.00
233	1.00	1.00
234	0.00	0.00

235 rows × 2 columns

In [ ]:
```
#首先我们使用建立好的模型对测试集数据进行预测预测的概率
score_best = clf_best.predict_proba(X_test_NoRSI200)[:,1]

#使用roc_curve方法得到三个模型的真正率TP,假正率FP和阈值threshold
fpr_best,tpr_best,thres_best = roc_curve(Y_test_NoRSI200,score_best,)

print("AUC (RFC with best estimators) is:",auc(fpr_best,tpr_best))

#创建画布
fig_best,ax_best = plt.subplots(figsize=(10,8))
```

```
#自定义标签名称label=''
ax_best.plot(fpr_best,tpr_best,linewidth=2,
        label='RFC with best estimators (AUC={})'.format(str(round(auc(fpr_best,tpr_best),3))),color='red')

#绘制对角线
ax_best.plot([0,1],[0,1],linestyle='--',color='blue')

#调整字体大小
plt.legend(fontsize=12)
plt.legend(loc="lower right")

#调整标题
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve (RFC with best estimators)')

plt.show()
```

```
AUC (RFC with best estimators) is: 0.9202089075874075
```



8. Comparison & Conclusion

In [ ]:
```
print(f'Train Accuracy (default RFC): {accuracy_score(Y_train,clf.predict(X_train))}, Test Accuracy: {accuracy_score(Y_test,clf.predict(X_test))}')
print(f'Train Accuracy (SHAP): {accuracy_score(Y_train_SHAP,clf_SHAP.predict(X_train_SHAP))}, Test Accuracy: {accuracy_score(Y_test_SHAP,clf_SHAP.predict(X_test_SHAP))}')
print(f'Train Accuracy (Dropped RSI_200): {accuracy_score(Y_train_NoRSI200,clf_NoRSI200.predict(X_train_NoRSI200))}, Test Accuracy: {accuracy_score(Y_test_NoRSI200,clf_NoRSI200.predict
print(f'Train Accuracy (RFC with best estimators): {accuracy_score(Y_train_NoRSI200,clf_best.predict(X_train_NoRSI200))}, Test Accuracy: {accuracy_score(Y_test_NoRSI200,clf_best.predic
```

```
Train Accuracy (default RFC): 1.0, Test Accuracy: 0.8212765957446808
Train Accuracy (SHAP): 1.0, Test Accuracy: 0.825531914893617
Train Accuracy (Dropped RSI_200): 1.0, Test Accuracy: 0.8425531914893617
Train Accuracy (RFC with best estimators): 1.0, Test Accuracy: 0.8425531914893617
```

In [ ]:
```
print("Classification Report (default RFC) is:")
print(classification_report(Y_test, Y_pred))

print("Classification Report (SHAP) is:")
print(classification_report(Y_test_SHAP, Y_pred_SHAP))
```

```
Classification Report (default RFC) is:
              precision    recall  f1-score   support

         0.0       0.79      0.85      0.82       113
         1.0       0.85      0.80      0.82       122

    accuracy                           0.82       235
   macro avg       0.82      0.82      0.82       235
weighted avg       0.82      0.82      0.82       235

Classification Report (SHAP) is:
              precision    recall  f1-score   support

         0.0       0.78      0.88      0.83       113
         1.0       0.88      0.77      0.82       122

    accuracy                           0.83       235
   macro avg       0.83      0.83      0.83       235
weighted avg       0.83      0.83      0.83       235
```

In [ ]:
```
print("Classification Report (Dropped RSI_200) is:")
print(classification_report(Y_test_NoRSI200, Y_pred_NoRSI200))

print("Classification Report (RFC with best estimators) is:")
print(classification_report(Y_test_NoRSI200, Y_pred_best))
```

```
Classification Report (Dropped RSI_200) is:
              precision    recall  f1-score   support

         0.0       0.79      0.91      0.85       113
         1.0       0.90      0.78      0.84       122

    accuracy                           0.84       235
   macro avg       0.85      0.85      0.84       235
weighted avg       0.85      0.84      0.84       235

Classification Report (RFC with best estimators) is:
              precision    recall  f1-score   support

         0.0       0.79      0.91      0.85       113
         1.0       0.90      0.78      0.84       122

    accuracy                           0.84       235
   macro avg       0.85      0.85      0.84       235
weighted avg       0.85      0.84      0.84       235
```

In [ ]:

```
print("AUC (default RFC) is:",auc(fpr,tpr))
print("AUC (SHAP) is:",auc(fpr_SHAP,tpr_SHAP))
print("AUC (Dropped RSI_200) is:",auc(fpr_NoRSI200,tpr_NoRSI200))
print("AUC (RFC with best estimators) is:",auc(fpr_best,tpr_best))
```

```
AUC (default RFC) is: 0.9154939793993906
AUC (SHAP) is: 0.9105614391411577
AUC (Dropped RSI_200) is: 0.9189032351661105
AUC (RFC with best estimators) is: 0.9202089075874075
```

- The model accuracy and f1 score have been improved with RSI_200 variable removed from the model.
- Applying best estimator hyperpamater or not has no impact on the accuracy result or f1 score.
- The model with best estimator is more accurate than that with default setting in ROC.

To conclude, I believe in mycase `the BEST Model is RFC with 130 the best n_estimators as well as removing the least significant feature RSI_200` .

9. Reference

[1] alokesh985. “Introduction to Support Vector Machines (SVM).” GeeksforGeeks, 9 June 2020, www.geeksforgeeks.org/introduction-to-support-vector-machines-svm/.

[2] “EnsembleVoteClassifier: A Majority Voting Classifier - Mlxtend.” Rasbt.github.io, rasbt.github.io/mlxtend/user_guide/classifier/EnsembleVoteClassifier/.

[3] “Jupyter Notebook Viewer.” Nbviewer.org, nbviewer.org/github/queirozfcom/python-sandbox/blob/master/python3/notebooks/svm-c/svm-c.ipynb?flush_cache=true#rbf-kernel. Accessed 18 Oct. 2023.

[4] Koehrsen, Will. “Hyperparameter Tuning the Random Forest in Python.” Medium, Towards Data Science, 10 Jan. 2018, towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74.

[5] Mandot, Pushkar. “What Is the Significance of c Value in Support Vector Machine?” Medium, 28 Aug. 2018, medium.com/@pushkarmandot/what-is-the-significance-of-c-value-in-support-vector-machine-28224e852c5a.

[6] “Python机器学习笔记：Grid SearchCV（网格搜索） - 战争热诚 - 博客园.” Www.cnblogs.com, 4 May 2019, www.cnblogs.com/wj-1314/p/10422159.html.

[7] Rajkeshav. “Coding Ninjas Studio.” Www.codingninjas.com, www.codingninjas.com/studio/library/the-voting-classifier.

[8] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.