CQF - Exam 3

Imports

In [ ]:
```
import pandas as pd
import numpy as np

# Visualization
import seaborn as sns
plt.style.use('seaborn')
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, TimeSeriesSplit
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score, ConfusionMatrixDispla
```

A. Maths and Feature Engineering

1. Consider MSE(β) wrt to the true value β in context of regression methods,

$$E\left[(\hat{\beta}-\beta)^2\right] = Var[\hat{\beta}] + \left(E[\hat{\beta}]-\beta\right)^2$$

(a) can there exist an estimator with the smaller MSE than minimal least squares?

The answer is: Yes, may exist a biased estimator with smaller MSE. In this case the estimator would increase a little the bias, $\left(E[\hat{\beta}]-\beta\right)^2$ with a larger decrease in variance term $Var[\hat{\beta}]$.

It is important to note that least square has the smallest mean squared error of all linear estimators **unbiased**.

(b) for a prediction, does the MSE measure an irreducible error or model error?

The MSE measures the model error, thus the quality of a predictor and is computed as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)$$

where $\hat{Y}_i$ is the predicted values by the model.

2. What does entropy say about the partitions in a classification problem?

(a) high entropy means the partitions are pure

(b) high entropy means the partitions are not pure

Answer: (b) Not pure

Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information. It is a measure of disorder or purity or unpredictability or uncertainty. Low entropy means less uncertain and high entropy means more uncertain.

3. Perform subset selection using any of all of a) filter, b) wrapper and, c) embedded methods

Load dataset

Let´s load open, high, low, close prices (dividend adjusted) for PETR4 BZ Equity

In [ ]:
```
df = pd.read_csv('data.csv', index_col='date', parse_dates=['date'])
```

Describe dataset

In [ ]:
```
df.describe()
```

Out[ ]:

	open	high	low	close
count	1240.000000	1240.000000	1240.000000	1240.000000
mean	13.421986	13.636153	13.196804	13.414884
std	5.237729	5.339208	5.150517	5.252697
min	4.707236	5.179236	4.613687	4.800786
25%	9.945838	10.127989	9.820758	9.973879
50%	11.684253	11.839912	11.536278	11.706667
75%	16.924746	17.212493	16.624367	16.926054
max	28.526247	30.003359	28.440277	29.479727

In [ ]:
```
df.head()
```

Out[ ]:

	open	high	low	close
date				
2018-05-21	10.296750	10.371707	9.839116	9.882513
2018-05-22	9.807405	9.997150	9.546507	9.767875
2018-05-23	9.578131	9.676956	9.198643	9.198643
2018-05-24	7.925775	8.202485	7.767655	7.937634
2018-05-25	8.162955	8.408042	7.771608	7.826950

Cleaning & Imputation

Check if data is already cleaned, if so no further manipulation required

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
open     0
high     0
low      0
close    0
dtype: int64
```

Feature Specification

In [ ]:
```
#create features
def create_features(df_orig):
    df = df_orig.copy()
    df['oc'] = df.open - df.close
    df['hl'] = df.high - df.low
    # returns
    for i in [1,5]:
        df[f'rt{i}'] = df.close / df.close.shift(i)
    # momentums
    for i in range(0,10):
        df[f'm{i+1}'] = df.close - df.close.shift(i+1)
    df['ma5'] = df.close.rolling(window=5).mean()
    df['ma10'] = df.close.rolling(window=10).mean()
    df['ewma'] = df.close.ewm(span=len(df.index),adjust=False).mean()
    df.dropna(inplace=True)
    return df
```

Let´s define the independent variables to be used in the evaluation

In [ ]:
```
df_featured = create_features(df)
df_featured
```

Out[ ]:

date	open	high	low	close	oc	hl	rt1	rt5	r
2018-06-05	6.862417	7.036349	6.558035	6.558035	0.304381	0.478314	0.946378	0.981076	-0.3715
2018-06-06	6.510599	6.593612	6.273419	6.455257	0.055342	0.320193	0.984328	0.846114	-0.1027
2018-06-07	6.324808	6.388056	5.901837	6.229936	0.094872	0.486220	0.965095	0.830348	-0.2253
2018-06-08	6.249701	6.399915	5.953226	6.028333	0.221368	0.446690	0.967640	0.943688	-0.2016
2018-06-11	6.127158	6.245748	5.980897	6.091581	0.035577	0.264851	1.010492	0.879064	0.0632
...	...	...	...	...	...	...	...	...	
2023-05-15	26.060000	26.150000	25.400000	25.660000	0.400000	0.750000	0.977524	1.046920	-0.5900
2023-05-16	26.110000	27.030000	26.080000	26.300000	-0.190000	0.950000	1.024942	1.069540	0.6400
2023-05-17	26.600000	26.760000	25.510000	25.660000	0.940000	1.250000	0.975665	1.046066	-0.6400
2023-05-18	25.590000	25.850000	25.350000	25.810000	-0.220000	0.500000	1.005846	1.014943	0.1500
2023-05-19	26.080000	26.180000	25.640000	25.920000	0.160000	0.540000	1.004262	0.987429	0.1100

1230 rows × 21 columns

Feature Selection

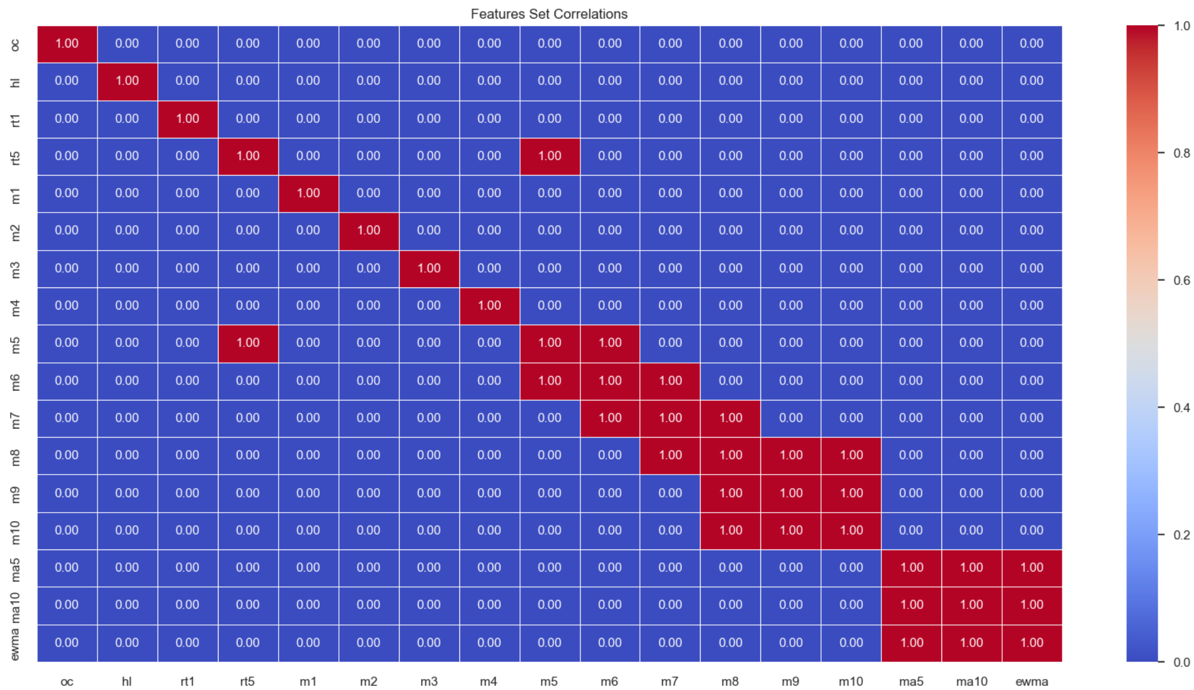
Let´s use correlation measure in order to check if we can reduce the number of input variables.

In [ ]:

```
sns.set(rc={'figure.figsize': (20, 10)})

sns.heatmap(df_featured.drop(['open', 'high', 'low', 'close'], axis=1).corr()>0.9,
            annot=True,
            annot_kws={"size": 11},
            fmt=".2f",
            linewidth=.5,
            cmap="coolwarm",
            cbar=True);

plt.title('Features Set Correlations');
```



As we can see, some of the features are high correlated with others and to address multicollinearity among features we will drop some of the them.

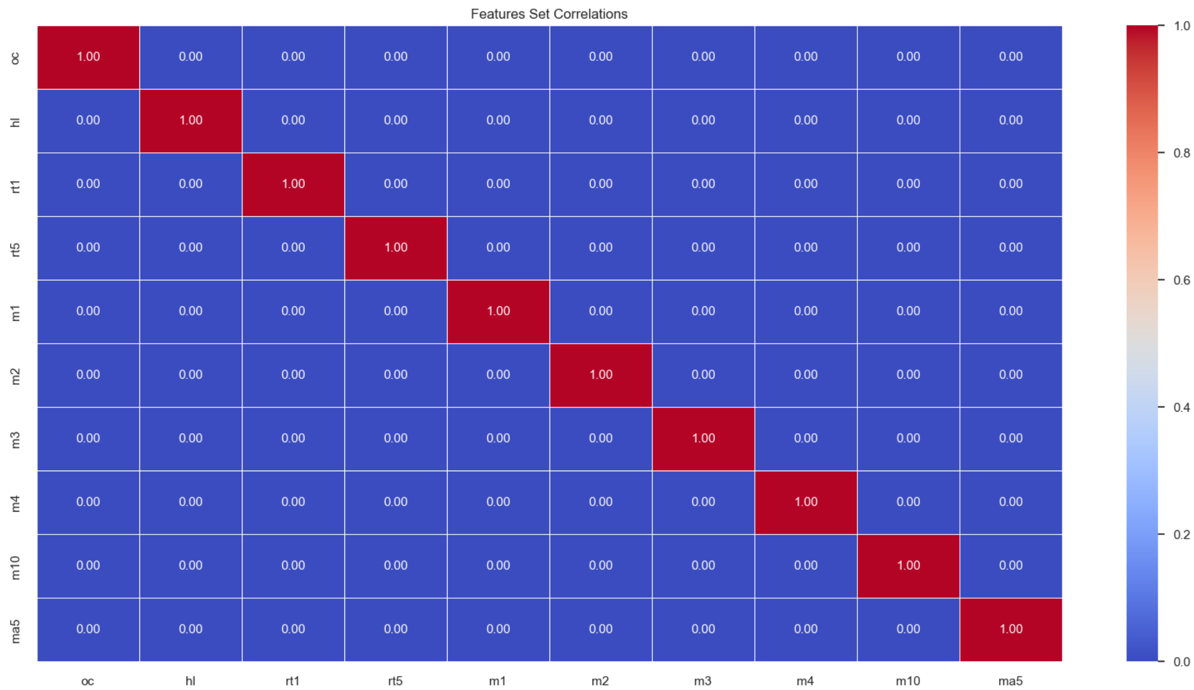
In [ ]:
```
to_drop = ['m5','m6', 'm7', 'm8', 'm9', 'ma10', 'ewma']
df_featured.drop(to_drop, axis=1, inplace=True)
```

Let´s check correlation measure once more

In [ ]:
```
sns.set(rc={'figure.figsize': (20, 10)})

sns.heatmap(df_featured.drop(['open', 'high', 'low', 'close'], axis=1).corr()>0.9,
            annot=True,
            annot_kws={"size": 11},
            fmt=".2f",
            linewidth=.5,
            cmap="coolwarm",
            cbar=True);

plt.title('Features Set Correlations');
```



In [ ]:
```
df_featured.describe()
```

Out[ ]:

	open	high	low	close	oc	hl	rt1
count	1230.000000	1230.000000	1230.000000	1230.000000	1230.000000	1230.000000	1230.000000
mean	13.464187	13.678231	13.241037	13.459112	0.005075	0.437194	1.001521
std	5.236895	5.339447	5.146613	5.249874	0.311505	0.299531	0.029511
min	4.707236	5.179236	4.613687	4.800786	-2.039822	0.087497	0.703022
25%	9.996309	10.172188	9.859260	10.012680	-0.140572	0.232959	0.987353
50%	11.708163	11.885520	11.554111	11.720969	0.004252	0.344439	1.001107
75%	16.963984	17.235802	16.674380	16.957445	0.157335	0.544250	1.015944
max	28.526247	30.003359	28.440277	29.479727	1.953853	2.149238	1.222222

Target or Label Definition

Now, we will define dependent variable, and for that we will impose a threshold considering positive returns only the ones above 0.25%, thus:

$$y_t = \begin{cases} 1, & \text{if } p_{t+1} > 1.0025 * p_t \\ 0, & \text{if } p_{t+1} \text{ otherwise} \end{cases}$$

In [ ]:
```
y = np.where(df_featured.close.shift(-1) / df_featured.close > 1.0001, 1, 0)
```

Generate array X, with the feature set

In [ ]:
```
X = df_featured[['oc', 'hl', 'rt1', 'm1','ma5']].values
```

Split Data

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_
```

Feature scaling

In [ ]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

After performing feature scaling, all values will be normalized and looks like this

In [ ]:
```
pd.DataFrame(data=X_train)
```

Out[ ]:

	0	1	2	3	4
0	0.180279	-0.770954	-0.226072	-0.198210	-0.455644
1	-0.298236	-1.059884	0.038846	0.030330	-0.449564
2	1.616267	1.821762	-0.790036	-1.275920	1.418526
3	-0.793814	1.135772	0.653336	0.870673	0.798816
4	0.382914	-0.700072	-0.082861	-0.089247	0.310346
...	...	...	...	...	...
917	1.434156	0.225632	-1.393774	-1.393878	-0.134913
918	1.820085	0.850007	-1.659232	-1.792237	0.049893
919	3.064085	2.281022	-1.380663	-2.550681	1.967923
920	0.088771	-0.814578	0.022280	0.010189	-0.720325
921	-0.047862	-0.713169	-0.206837	-0.168145	-0.751004

922 rows × 5 columns

In [ ]:
```
model = SVC(kernel = 'rbf', random_state=0)
model.fit(X_train, y_train)
```

Out[ ]:

▾ SVC
```
SVC(random_state=0)
```

In [ ]:
```
y_pred = model.predict(X_test)
```

In [ ]:
```
pd.DataFrame(data=y_pred)
```

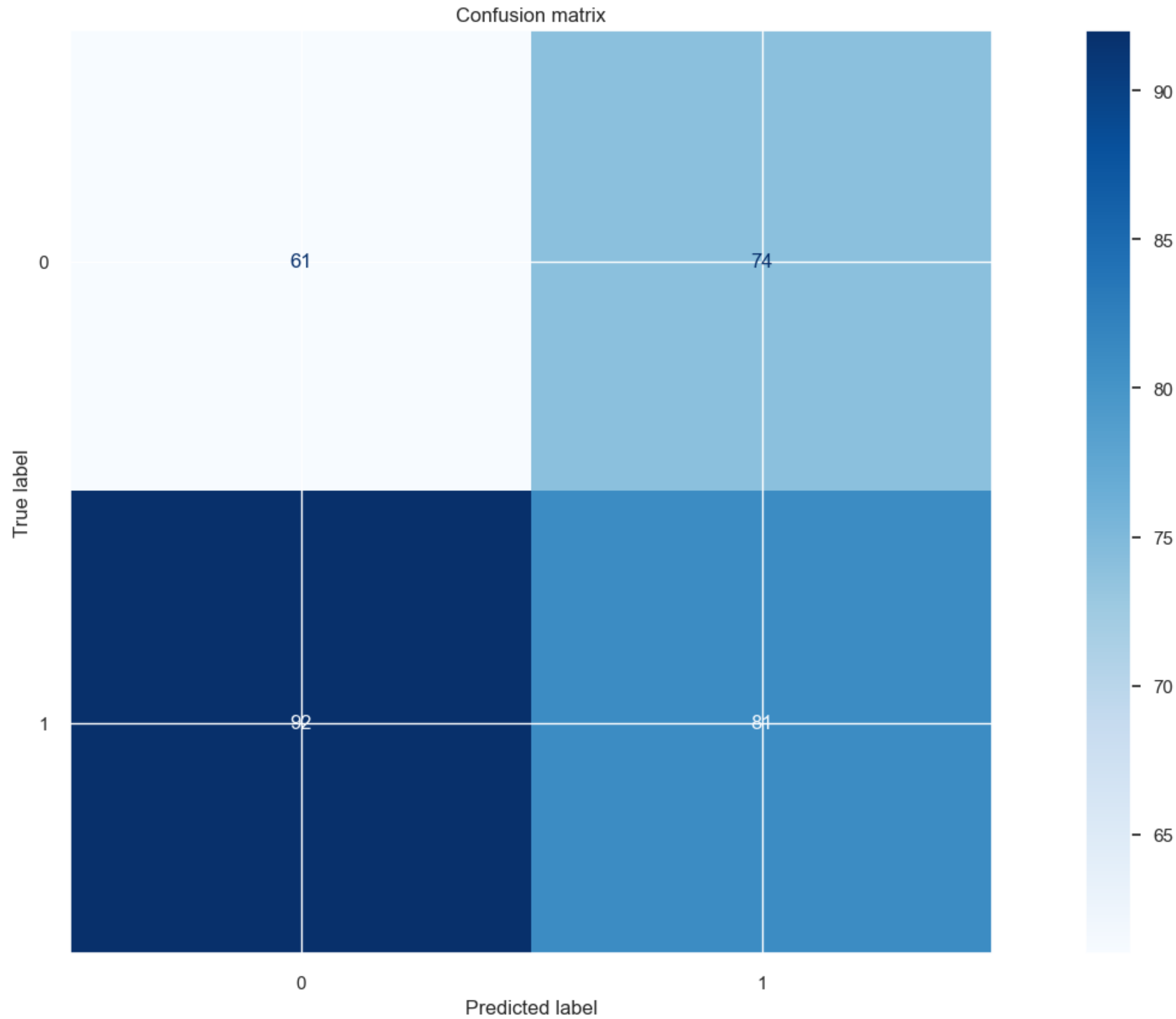
Out[ ]:

	0
0	0
1	0
2	0
3	1
4	0
...	...
303	0
304	1
305	0
306	1
307	0

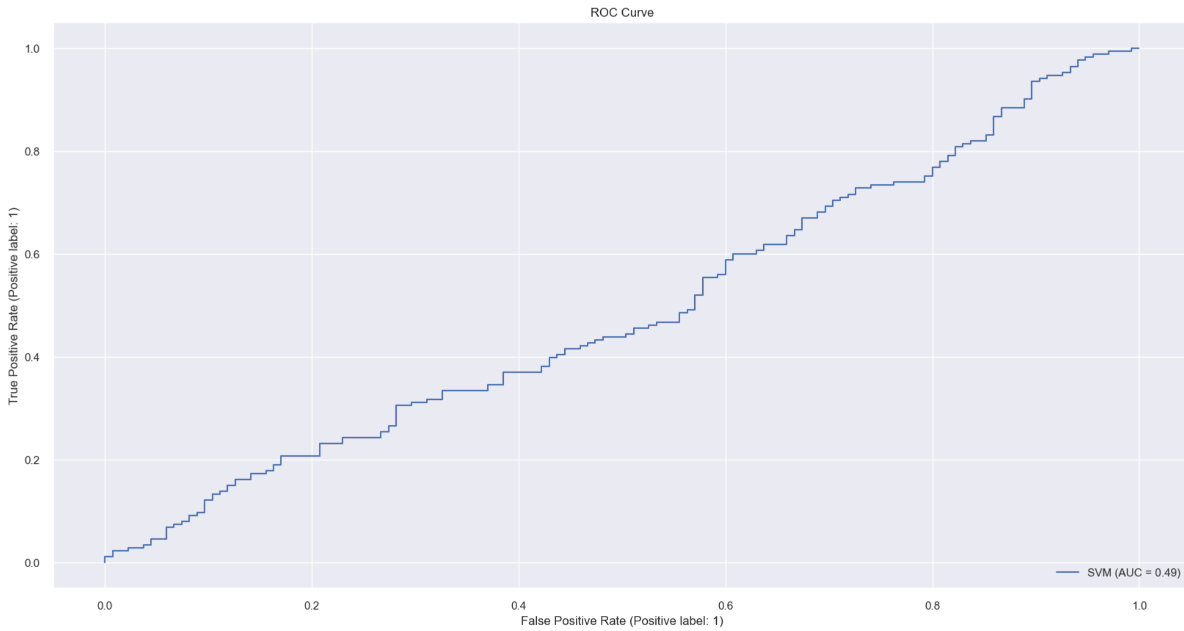
308 rows × 1 columns

In [ ]:

```
# Display confussion matrix
disp = ConfusionMatrixDisplay.from_estimator(
        model,
        X_test,
        y_test,
        display_labels=model.classes_,
        cmap=plt.cm.Blues
    )
disp.ax_.set_title('Confusion matrix')
plt.show()
```



In [ ]:
```
# Display ROCCurve
disp_roc = RocCurveDisplay.from_estimator(
            model  ,
            X_test,
            y_test,
            name='SVM')
disp_roc.ax_.set_title('ROC Curve')
plt.show()
```



In [ ]:
```
accuracy_score(y_test, y_pred)
```

Out[ ]:
```
0.461038961038961
```

In [ ]:
```
# Classification Report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.40      0.45      0.42       135
           1       0.52      0.47      0.49       173

    accuracy                           0.46       308
   macro avg       0.46      0.46      0.46       308
weighted avg       0.47      0.46      0.46       308
```

Hyper-parameter Tuning

GridSearch

First we check the parameters available in the model

In [ ]:
```
model.get_params()
```

Out[ ]:
```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'rbf',
 'max_iter': -1,
 'probability': False,
 'random_state': 0,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

As we are using Gaussian radial basis function (RBF), let´s now search the best C and gamma parameter to be used in the model

In [ ]:
```python
param_grid = {'gamma':np.arange(0.0002,99), 'C': np.arange(0.2,99)}
tscv = TimeSeriesSplit(n_splits=2, gap=1)
grid_search = GridSearchCV(model, param_grid, scoring='roc_auc', n_jobs=-1, cv=tscv
grid_search.fit(X_train, y_train)
```

```
Fitting 2 folds for each of 9801 candidates, totalling 19602 fits
```

Out[ ]:
```
▸ GridSearchCV
▸ estimator: SVC
  ▸ SVC
```

Tunned model

Now, let´s train the model using the best parameter searched

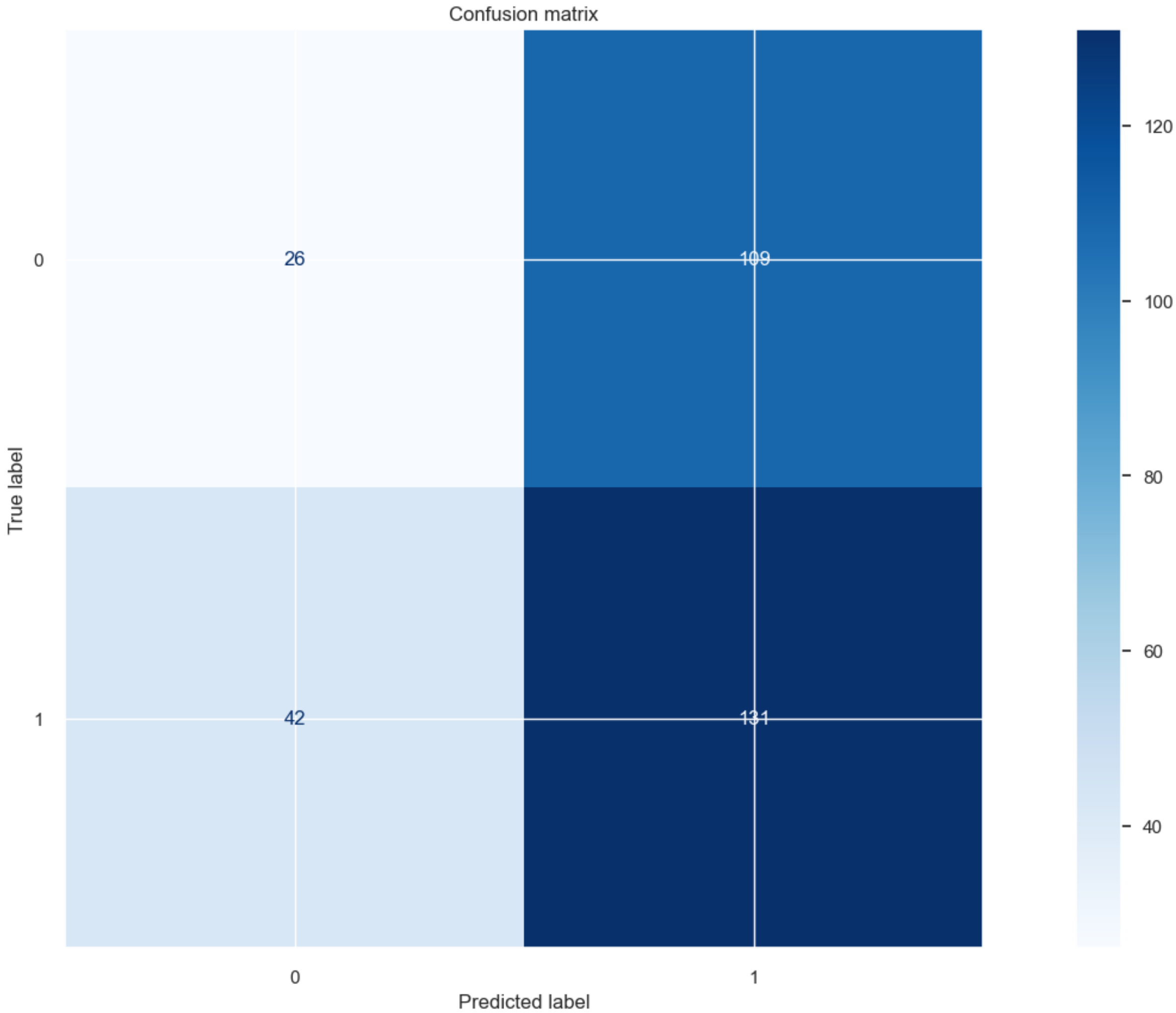
In [ ]:
```python
model = SVC(kernel = 'rbf', random_state=0, gamma=grid_search.best_params_['gamma']
model.fit(X_train, y_train)
```

Out[ ]:
```
▾ SVC
SVC(C=12.2, gamma=0.0002, random_state=0)
```

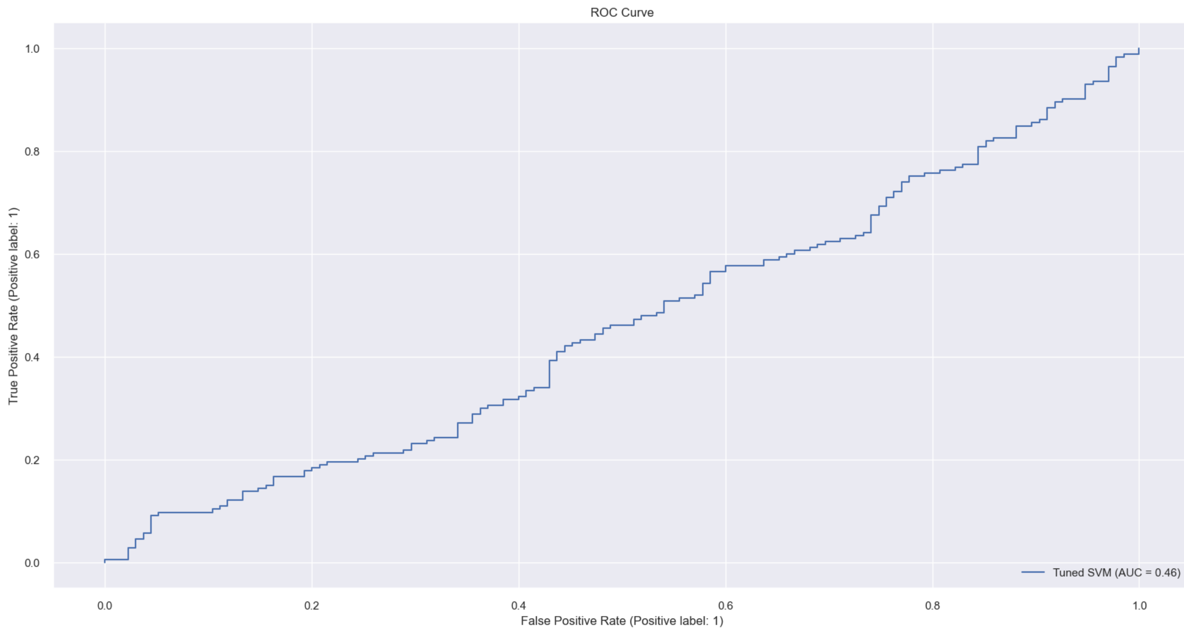
In [ ]:
```python
y_pred = model.predict(X_test)
```

In [ ]:
```python
# Display confussion matrix
disp = ConfusionMatrixDisplay.from_estimator(
        model,
        X_test,
        y_test,
        display_labels=model.classes_,
        cmap=plt.cm.Blues
    )
```

```
disp.ax_.set_title('Confusion matrix')
plt.show()
```



In [ ]:
```
# Display ROCCurve
disp_roc = RocCurveDisplay.from_estimator(
            model  ,
            X_test,
            y_test,
            name='Tuned SVM')
disp_roc.ax_.set_title('ROC Curve')
plt.show()
```



In [ ]:
```
accuracy_score(y_test, y_pred)
```

Out[ ]:
```
0.5097402597402597
```

In [ ]:
```
# Classification Report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.38      0.19      0.26       135
           1       0.55      0.76      0.63       173

    accuracy                           0.51       308
   macro avg       0.46      0.47      0.45       308
weighted avg       0.47      0.51      0.47       308
```

Observations

1. After hyperparametrization, model accuracy improved around 10.4%
2. Model improved predictor for the uptrend when compared to the downtrend