Single and multi factor screening models enhanced by deep learning

Financial Markets Analytics

Full Name	Student ID
Davide Ghilardi	857384
Fabio Salerno	861419

Milan, June 23, 2023

Contents

1 | Introduction

Stock screening models are used by investors and traders to filter and identify potential investment opportunities based on specific criteria. These models analyze large amounts of financial data to narrow down the universe of stocks and highlight those that meet the desired parameters, with the goal of creating portfolios with maximum or minimum exposure to those factors. The aim of the project is to deploy in python different kinds of screening models and exploiting them to create portfolios that will be evaluated in terms of returns, and calculating appropriate absolute and relative performance measures to the benchmark.
In particular will be deployed:

- Univariate screening model.
- Simultaneous multivariate screening model.
- Sequential multivariate screening model.
- Momentum screening enhanced by deep learning techniques.

1.1 | Data extraction

The data used come from the "euro.xls" dataset, which includes 797 securities and refers to a time period of 111 months. The dataset contains for each stock several key indicators for companies such as PE ratio, PB ratio, EBITDA and many others. First of all some preliminary operations where done to extract the data from the Excel file.

```
import pandas as pd
import numpy as np

#Data loading
data = pd.read_excel('Euro.xlsx', header=None)
```

The following functions where designed to retrieve for a single factor its value for each stock along the time period available:

```
# Function to match a given string
def match_string(x, string):
    if pd.isna(x):
        return False
    else:
        return string in x

# Function to get the factor df for all equities
def get_factor_df(factor, perfect_match=False):
    if perfect_match:
        price_book = data[0].apply(lambda x: x == factor)
    else:
        price_book = data[0].apply(match_string, args={factor})
    df = data[price_book].set_index(equities)
    df.drop(0, axis=1, inplace=True)
    df.columns = dates[1:]

    return df
```

1.2 | Benchmark calculation

To better evaluate the screening models performances, it was necessary to have a **benchmark** in particular it was used the average monthly log returns of all the stocks equally weighted. To obtain the bcenchmark first of all through the function *get_factor_df*() the prices for each stock were extracted from the dataset. Then those prices were used to compute the log returns for each stock.
Another usefull function deployed was **get_returns**() that computes the aggregate log returns for a df containing the returns of multiple stocks.

```
from datetime import datetime

def get_returns(df):
    returns = []
```

```
    for mth in df.columns:
        mth = mth.strftime('%Y-%m-%d')
        zero_mask = df[mth].apply(lambda x: (x != 0) and (not pd.isna(x)) and (np.abs(x) != np.inf))
        returns.append(df[mth].loc[zero_mask].mean())
    return pd.Series(returns, index=dates[1:])
```

This function was used in this case to compute the returns of the benchmark portfolio.

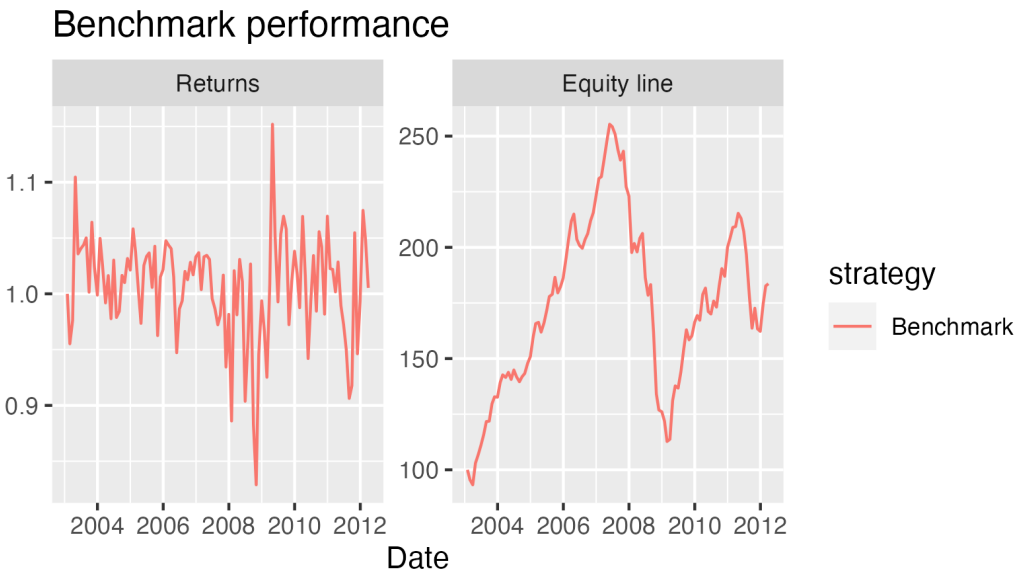

Figure 1.1: Benchmark performances over time.

2 | Univariate screening

In this section the focus will be on the deployment of a single factor screening model, in particular it was chosen to use the factor PB (price to book value). The main goal was for each month screening the stocks according to PB factor and classifying them from the lowest PB value to the highest. PB ratio uses the book value which is a calculation (difference between assets and liabilities) that comes from balance sheets, this calculation is not updated instantly but is produced with a delay, for this project it was considered as a proxy for the correction of those numbers a three period (three months) time lag. Morover the strategy was: each month creating a portfolio composed of the top 25 stocks with a long-position investment and the bottom 25 with a short position-investment. The choice on the number of stocks composing the portfolio was based on trying to find a good tradeoff between a too much contracted and not-diversified portfolio (picking for example 6 stocks) and avoiding to include stocks not so much exposed to the specific factor (picking for example 100 stocks the 50th stock will be much less exposed than the first one).
Another important element to consider are the transaction costs, in this project were assumed to be around 20 basis points. To take into account the transaction costs it was designed a specific function that given two portfolios (corresponding to the previous period and corresponding to the current period) outputs the turnover rate.

```
# Fuction to compute the share of stocks that changes between periods
def get_turnover(prec, actual, ptf_size):
    k = 0
    for i in prec: # prec is the previous period portfolio
        if i not in actual: # actual is the current period portfolio
            k += 1

    return k / ptf_size # ptf_size is the size of the long/short portfolio
```

The univariate screening was deployed by designing the **univariate_screening**() function, that takes as inputs:

- **df** is the dataframe with the specific factor computed on each stock for the 111 months (obtainable by using the "get_factor_df" function).
- **sign**, set 'sign' 1 to go long on the higher values and short on lower values while set 'sign' -1 to go short on the higher values and long on lower values.
- **how**, depending on the strategy you want to apply on your portfolio, you can select: 'long', 'short' or 'both'.
- **ptf_size** is the size of the long/short portfolio, 50 setted as default.
- **trans_cost**, the percentage of transaction costs to be considered, 20 basis points setted as default.
- **holding_period**, indicates for how long the position is maintained until the next allocation.

The function outputs for each time period the returns obtained by the strategy.

```
# Function to compute univariate screening returns for a factor df
def univariate_screening(df, sign, how='both', ptf_size = 50, trans_cost = 0.002, holding_period=1):
    # Check available methods
    if how not in ('long', 'short', 'both'):
        print('Mehtod not allowed!')
        return None

    long_returns = []
    short_returns = []

    to_long = []
    to_short = []

    long_temp = []
    short_temp = []

    hist_long = []
    hist_short = []
    hist_turnover = []

    if how == 'both':
```

```
        ptf_size = ptf_size // 2 # with 'both' method, ptf size is split in long and short positions

    # Compute retuns for every month
    for i, mth in enumerate(df.columns[:-1]):

        if i % holding_period == 0:
            zero_mask = df[mth] != 0 # filter non-zero stocks for the factor

            ordered_pb = (df.loc[zero_mask, mth].dropna() * sign).sort_values(ascending=False) # order
     stock depending on the factor
            # set 'sign' 1 to go long on the higher values and short on lower values
            # set 'sign' -1 to go short on the higher values and long on lower values

            if how == 'both':
                to_long = ordered_pb[:ptf_size].index # stock to go long
                to_short = ordered_pb[-ptf_size:].index # stock to go long
            elif how == 'long':
                to_long = ordered_pb[:ptf_size].index # stock to go long
            else:
                to_short = ordered_pb[-ptf_size:].index # stock to go long

        hist_long.append(to_long)
        hist_short.append(to_short)

        # Compute returns
        long_return = get_returns(log_returns.T.loc[to_long])[df.columns[i+1]]
        short_return = - get_returns(log_returns.T.loc[to_short])[df.columns[i+1]] # computed by
     inverting the sign of regular returns

        # Transaction costs
        long_turnover = get_turnover(to_long, long_temp, ptf_size)
        short_turnover = get_turnover(to_short, short_temp, ptf_size)

        long_fee = trans_cost * long_turnover
        short_fee = trans_cost * short_turnover

        long_temp = to_long
        short_temp = to_short

        long_returns.append(long_return - long_fee)
        short_returns.append(short_return - short_fee)

    long_returns = pd.Series(long_returns, index=df.columns[1:])
    short_returns = pd.Series(short_returns, index=df.columns[1:])

    # Compute overall returns
    univ_returns = pd.concat([long_returns, short_returns], axis=1)
    univ_returns.columns = ['Long', 'Short']
    univ_returns['Overall'] = (univ_returns['Long'] + univ_returns['Short']) / 2

    if how == 'long':
        final_returns = long_returns
    elif how == 'short':
        final_returns = short_returns
    else:
        final_returns = univ_returns['Overall']

    return final_returns, hist_long, hist_short
```

Applying this function to the PB_ratio dataframe previously computed, it was possible to perform the screening for each month and to produce the returns. For the screening it was decided to go only long, avoiding underexposed equities.

2.1 | Performances

An overview of the results can be noticed comparing the results of this strategy and the benchmark:

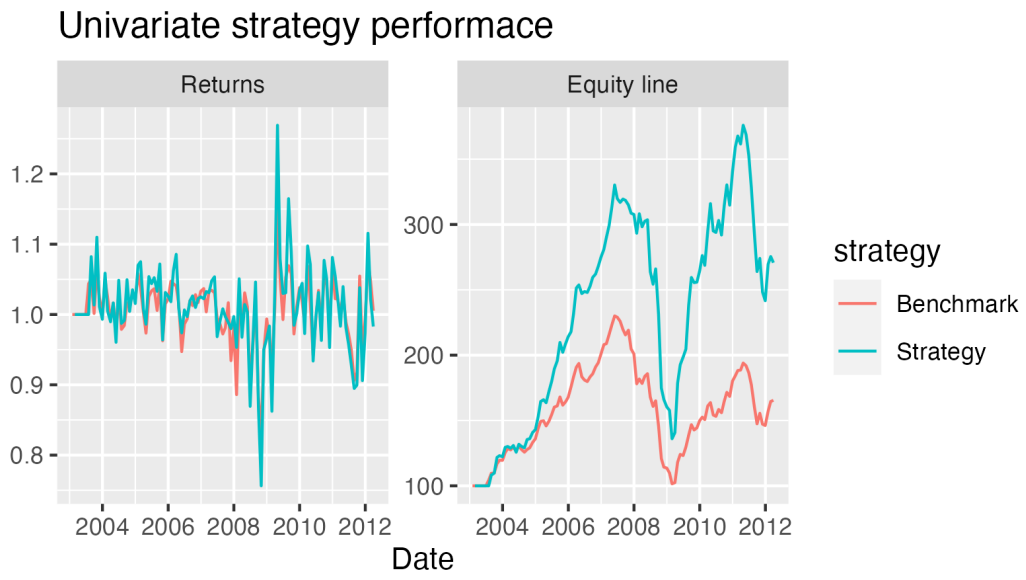

Figure 2.1: Results obtained by the strategy versus the benchmark (Univariate screen PB_ratio).

As is possible to notice, the strategy sligthly outperformed the benchmark, with an equity line constantly above the benchmark's. As expected, having a lower nuber of assets, the strategy shows higher variance both in the return and in the equity line than the benchmark.
To better evaluate the results and account for the volatility, absolute and relative indicators were computed. In particular, we chose Information Ratio (IR) as relative metric, while Sahrpe, Traynor, and Sortino ratios as absolute metrics.
All the values for each metric represent the annual average, and to compute them, the following functions were defined:

```
# Absolute metrics
def get_metrics(strat, bchmk, rf=0.03):

    rf = np.log((1 + rf) ** (1 / 12))

    # Sharpe
    sharpe = strat - rf
    sharpe = np.mean(sharpe) / np.std(sharpe) * np.sqrt(12)

    # Traynor
    traynor = strat - rf
    traynor = np.mean(traynor) / (np.cov(strat, bchmk)[0][1] / np.var(bchmk)) * 12

    # Sorbino
    Sortino = strat - rf
    under_mean = Sortino[Sortino <= np.mean(Sortino)]
    Sortino = np.mean(Sortino) / np.std(under_mean) * np.sqrt(12)

    return {
        'Sharpe': np.round(sharpe, 3),
        'Traynor': np.round(traynor, 3),
        'Sortino': np.round(Sortino, 3)}
```

Considering the screening model developed using the PB_ratio, the results are the following:

	Information ratio	Sharpe ratio	Traynor ratio	Sortino ratio
PB_ratio screening	0.803	0.517	0.090	0.631

Table 2.1: Performance metrics(PB_ratio screening).

What was noticed from the graph can be also resumed by the performance metrics; in fact, all of them are positive and moderate, indicating that excess of return is scaled by the greater risk taken.

To better understand the assets allocation of the strategy through time, a pixel map was designed where the stocks available are the rows, and the time period is in the columns. White pixels represent stocks included in the portfolio (each of them equally weighted), while the black ones are the stocks not included.

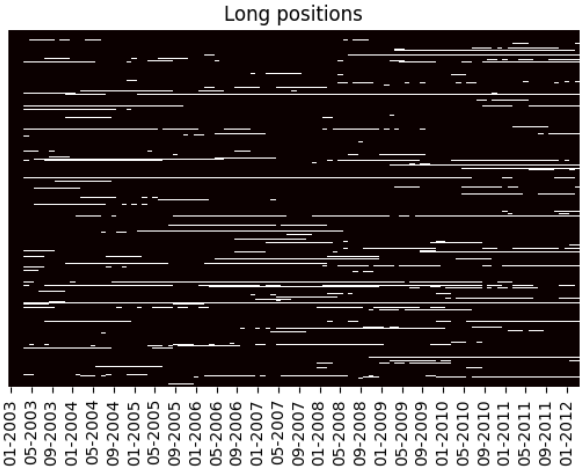

Figure 2.2: Pixel map, asset allocation of the strategy (univariate screening.)

In this case, it is possible to notice that the stocks allocation doesn't change so much through time, so the portfolio remains stable.

3 | Multivariate screening

The basic idea behind multivariate screening models is to identify a set of factors relevant on the basis of the assumptions of the asset manager, and implement a classification of securities according to those factors, and thus select portfolios with maximum and minimum exposure. In **simultaneous screening models**: are applied simultaneously more than one screen through a model with multiple factors, to produce an aggregated score, which will be used to perform a screening of the stocks. In **sequential screening models**: the universe of titles is selected on the basis of the first criterion, then the second and so on, progressively narrowing the final set of titles. For this project it was tried to deploy and perform both types of screening procedures, being inspired by the **Lakonishok strategy**, which is focused on picking the most undervalued top 30% capitalisation companies according to P/E and P/B. Finally, through Delta consensus rising and RSI reversing, stocks whose trend of depression has reversed are identified. Considering the information available, the factors used to perform the screening were:

- CUR_MKT_CAP, current market cap;
- P/B price to book value;
- P/E price earnings;
- RSI_30D, 30 days relative strenght index;
- 3 months delta consensus.

3.1 | Simultaneous screening

The simultaneous screeining was performed by combining each single factor (normalized) with the goal of obtaining a general score used to perform the screenining. Instead of equal weights, it was decided to scale each factor by its screening performance, and for this reason the level of relevance of each factor is set to be equal to its univariate information ratio.
First of all, it was designed the function: **lakonishock_screening**() that takes as inputs:

- **df** is the dataframe with the specific factor computed on each stock for the 111 months (obtainable by using the "get_factor_df" function).
- **equities** list of the equities to be used.
- **sign**, set 'sign' 1 to go long on the higher values and short on lower values while set 'sign' -1 to go short on the higher values and long on lower values.
- **ptf_size** is the size of the long/short portfolio, 50 setted as default.
- **trans_cost**, the percentage of transaction costs to be considered, 20 basis points setted as default.
- **holding_period**, indicates for how long the position is maintained until the next allocation.

```
# Function to compute Lakonishock screening
def lakonishock_screening(df, equities, sign, ptf_size = 50, trans_cost = 0.002, holding_period=1):

    returns = []
    temp = []

    for (i, mth), eqy in zip(enumerate(df.columns[:-1]), equities):

        if i % holding_period == 0:
            mth_df = df.loc[eqy, mth] # monthly factor series filtered for selected stocks

            zero_mask = mth_df != 0 # filter non-zero stocks

            ordered_pb = (mth_df.loc[zero_mask].dropna() * sign).sort_values(ascending=False)

            to_long = ordered_pb[-ptf_size:].index

        # Compute returns
        ret = get_returns(log_returns.T.loc[to_long])[df.columns[i+1]]
```

```
        # Transaction costs
        long_fee = trans_cost * get_turnover(to_long, temp, ptf_size)

        temp = to_long

        returns.append(ret - long_fee)

    returns = pd.Series(returns, index=df.columns[1:])

    return returns
```

The function introduced was used to perform a screening for each selected factor (PB, PE, RSI, dCONS), considering just the top 30% market cap stocks for each month. Then, information ratios are computed and used to obtain the weights of each factor in the final score. The weights were computed scaling each single factor's IR with the sum of them ($IR_{tot} = IR_{PB} + IR_{PE} + IR_{RSI} + IR_{dCON}$).
The following code describe how for each period of time the weights were computed, taking into account just the information available every month.

```
weights = {
    'PB': [],
    'PE': [],
    'RSI': [],
    'dCON': []
}

Zscores = []

for i, mth in enumerate(PB.columns):
    # Get the IR for each univariate screening
    PB_IR = get_IR(PB_screen.loc[:mth], bch.loc[:mth])
    PE_IR = get_IR(PE_screen.loc[:mth], bch.loc[:mth])
    RSI_IR = get_IR(RSI_screen.loc[:mth], bch.loc[:mth])
    dCON_IR = get_IR(dCON_screen.loc[:mth], bch.loc[:mth])

    TOT_IR = np.sum(np.abs((PB_IR, PE_IR, RSI_IR, dCON_IR)))
    sign = np.sign(np.sum((PB_IR, PE_IR, RSI_IR, dCON_IR)))

    weights['PB'].append(PB_IR / TOT_IR * sign)
    weights['PE'].append(PE_IR / TOT_IR * sign)
    weights['RSI'].append(RSI_IR / TOT_IR * sign)
    weights['dCON'].append(dCON_IR / TOT_IR * sign)

    Zsc = (PBz.loc[:, mth] * PB_IR + PEz.loc[:, mth] * PE_IR + RSIz.loc[:, mth] * RSI_IR - dCONz.loc[:,
     mth] * dCON_IR) / TOT_IR * sign
    Zscores.append(Zsc)

Zscores = pd.concat(Zscores, axis=1)
weights = pd.DataFrame(weights)
weights.index = Zscores.columns
```

Then, after obtaining each weight, the total score is computed:

$$\bar{Z} = w_{PB} \cdot z_{PB} + w_{PE} \cdot z_{PE} + w_{RSI} \cdot z_{RSI} - w_{dCON} \cdot z_{dCON}$$

From the following graph it is possible to appriciate the weights distribution through time for each factor. What mainly stands out from the graph is that dCON factor's weight, which starts very high (75%), reduces through time, while the RSI's starts with a very low value and then obtains more and more relevance. P/E and P/B factors' weight remain roughly constant and low through time.

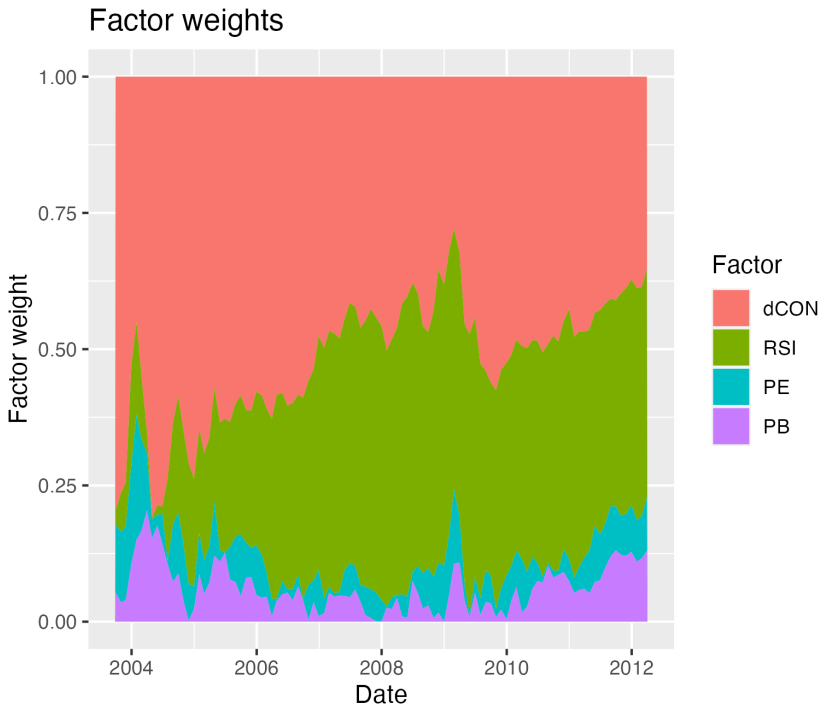

Figure 3.1: Weights distribution for each factor over time.

The overall score was used to perform a univariate screening of the stocks taking only long positions. Considering the screening model developed the results were the following:

	Information ratio	Sharpe ratio	Traynor ratio	Sortino ratio
Simultaneous screening	0.127	0.186	0.034	0.227

Table 3.1: Performance metrics (simultaneous screening).

The information ratio and the absolute metrics have a slightly positive value, so this strategy is performing a little above the benchmark.
This can be also noticed graphically, with the equity line of the strategy ending roughly at the same level of the benchmark's (slightly above).

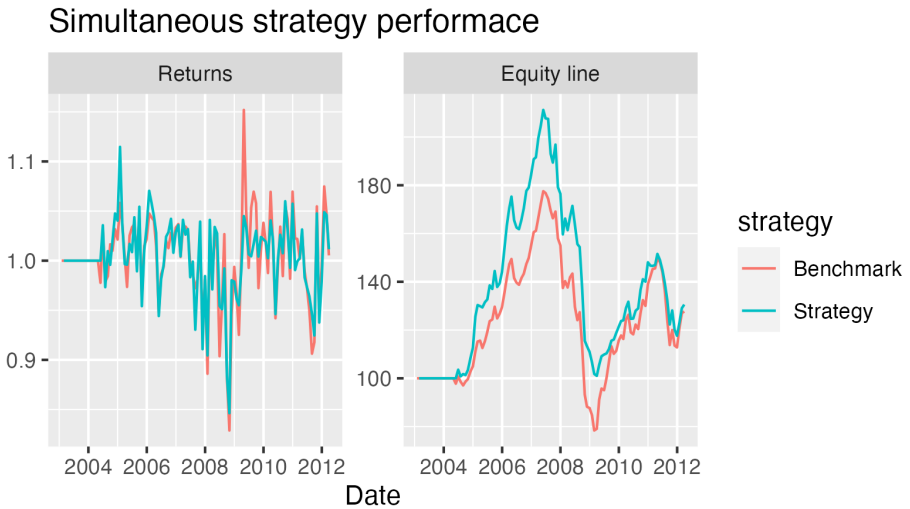

Figure 3.2: Results obtained by the strategy versus the benchmark (Simultaneous screening)

Looking at the asset allocation it is possible to notice that assets inside the portfolio tends to change a lot at the beginning of the strategy so the portfolio is very dynamic. On the other hand, at the beginning of 2008, positions starts to become more stable.

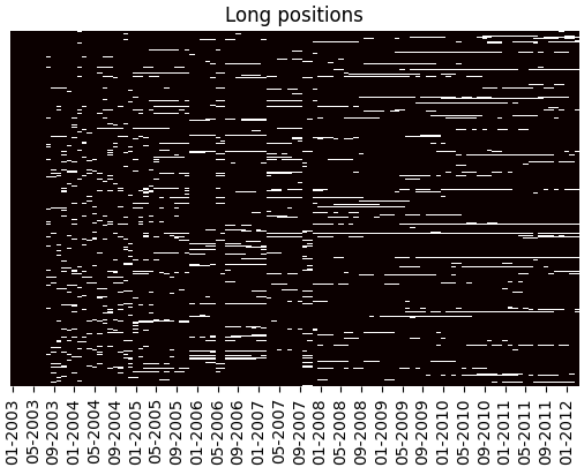

Figure 3.3: Pixel map, asset allocation of the strategy (simultaneous screening).

3.2 | Sequential screening

The sequential screening consists in deploying iteratively a univariate screening where each step a single factor from the ones chosen (Market cap, RSI, dCON, PB, PE) is used to sort the securities and to select top and bottom ones, thus reducing the universe of stocks. Following the Lakonishok's strategy, the screening is performed starting from the top 30% cap stocks. To deploy sequential strategy, the function **sequential_screening**() was designed, the function takes as inputs:

- **dfs**, a list of dataframes for each factor used for the sequential screening (obtainable by using the "get_factor_df" function).
- **signs**, a list of signs for each factor; 'sign' 1 to go long on the higher values and short on lower values while set 'sign' -1 to go short on the higher values and long on lower values.
- **how**, depending on the strategy you want to apply on your portfolio, you can select: 'long', 'short' or 'both'.
- **eqty_start**, the number of equities setted for starting the first step of the iterative selection (default setted as 500).
- **eqty_end**, the target number of stocks at the end of the sequential screening (default setted as 50).
- **trans_cost**, the percentage of transaction costs to be considered, 20 basis points setted as default.
- **holding_period**, indicates for how long the position is maintained until the next allocation.

```
# Function to compute sequential screening
def sequential_screening(dfs, signs, how='both', eqty_start=500, eqty_end=50, trans_cost = 0.002,
     holding_period=1):
    if how not in ('long', 'short', 'both'):
        print('Mehtod not allowed!')
        return None

    long_returns = []
    short_returns = []

    # Compute the number of stocks to take at each sequential step
    step = (eqty_start - eqty_end) // len(dfs)
    sizes = [eqty_start - (i+1) * step for i in range(len(dfs) - 1)]
    sizes.append(eqty_end)

    to_long = []
```

```
to_short = []

long_temp = []
short_temp = []

hist_long = []
hist_short = []

for i, mth in enumerate(dfs[0].columns[:-1]):
    # at the beginnning to_long and to_short contain all the stocks
    if how == 'both':
        to_long = list(dfs[0].index)
        to_short = list(dfs[0].index)
    elif how == 'long':
        to_long = list(dfs[0].index)
    else:
        to_short = list(dfs[0].index)

    if i % holding_period == 0:
        # sequential screening
        for df, ptf_size, sign in zip(dfs, sizes, signs):

            all_eqty = list(to_short) + [i for i in to_long if i not in to_short] # equities to be
considered at each step

            if how == 'both':
                ptf_size = ptf_size // 2

            zero_mask = df.loc[all_eqty, mth] != 0 # filter non-zero stocks

            ordered_pb = (df.loc[zero_mask.index].loc[zero_mask, mth].dropna() * sign).sort_values(
ascending=False)

            if how == 'both':
              to_long = ordered_pb[:ptf_size].index # stock to go long
              to_short = ordered_pb[-ptf_size:].index # stock to go long
            elif how == 'long':
              to_long = ordered_pb[:ptf_size].index # stock to go long
            else:
              to_short = ordered_pb[-ptf_size:].index # stock to go long

    hist_long.append(to_long)
    hist_short.append(to_short)

    # Compute returns
    long_return = get_returns(log_returns.T.loc[to_long])[df.columns[i+1]]
    short_return = - get_returns(log_returns.T.loc[to_short])[df.columns[i+1]] # i ritorni short
vanno invertiti di segno!

    # Transaction costs
    long_fee = trans_cost * get_turnover(to_long, long_temp, ptf_size)
    short_fee = trans_cost * get_turnover(to_short, short_temp, ptf_size)

    long_temp = to_long
    short_temp = to_short

    long_returns.append(long_return - long_fee)
    short_returns.append(short_return - short_fee)

long_returns = pd.Series(long_returns, index=dfs[0].columns[1:])
short_returns = pd.Series(short_returns, index=dfs[0].columns[1:])

univ_returns = pd.concat([long_returns, short_returns], axis=1)
univ_returns.columns = ['Long', 'Short']
univ_returns['Overall'] = (univ_returns['Long'] + univ_returns['Short']) / 2

if how == 'long':
    final_returns = long_returns
elif how == 'short':
    final_returns = short_returns
else:
    final_returns = univ_returns['Overall']
```

```
    return final_returns, hist_long, hist_short
```

Appying this function to the selected factors, it's possible to perform the screening for each month and to produce the returns. Considering the screening model developed, which follows a only-long strategy, the results were the following:

	Information ratio	Sharpe ratio	Traynor ratio	Sortino ratio
Sequential screening	-0.727	-0.109	-0.018	-0.127

Table 3.2: Performance metrics (sequential screening).

The information ratio shows a negative value so this strategy is not performing well considering the benchmark; also Sharpe and Sortino ratios are very low. This can be noticed graphically looking at the equity lines, with the strategy being under the benchmark.

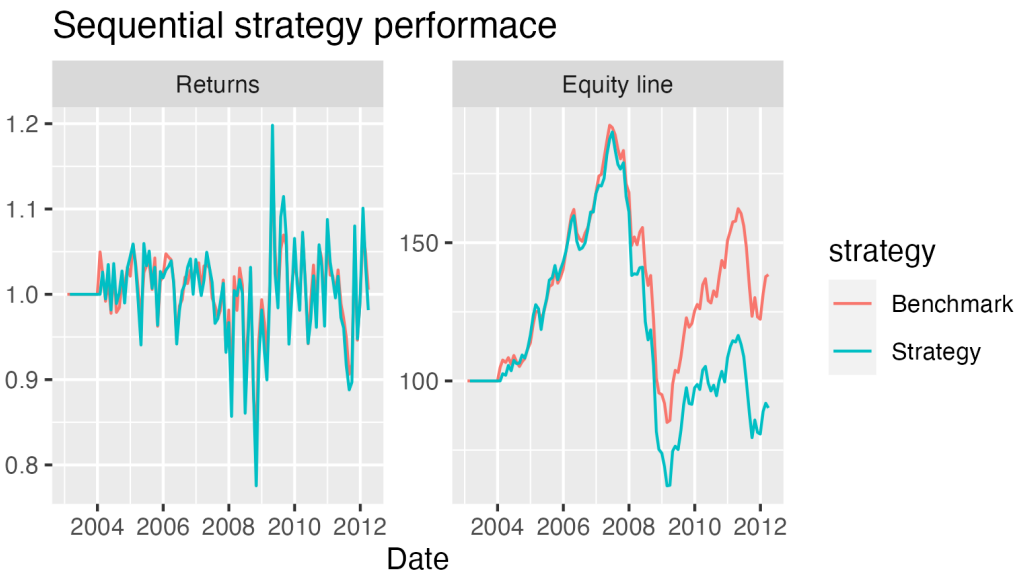

Figure 3.4: Results obtained by the strategy versus the benchmark (sequential screening).

Looking deeper to the portfolio allocation, it is possible to notice that some assets don't change much through time while others are traded more.

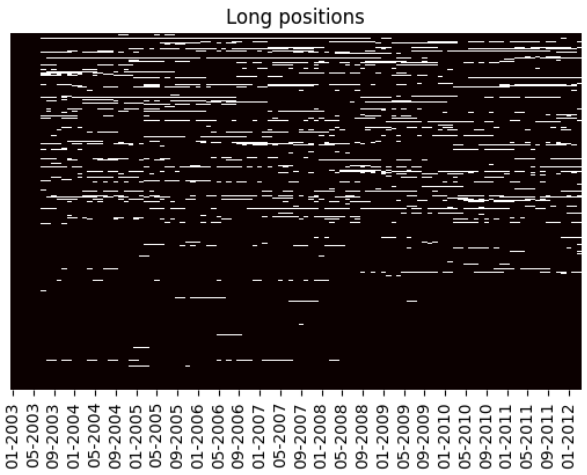

Figure 3.5: Pixel map, asset allocation of the strategy (sequential screening).

4 | Collinearity solutions

One of the main issues of using a z-score as a compound measure for all the chosen factors, is the possibly high correlation between them, that can lead to information redundancy, and can affect screening results. In general, collinearity is a linear association between two or more covariates, so it is possible to retrieve a variable through a linear combination of the others.
Addressing collinearity is important to ensure the validity and reliability of the models, allowing for accurate interpretation and meaningful conclusions. In many occasions collinearity is a problem to deal with, and many solutions can be considered such as exploiting residualized factors, directly removing high correlated ones, and twisting the facotrs' weights.
The idea proposed in this project for trying to solve collinearity issues is through the deployment of a PCA analysis. Principal Component Analysis is a dimensionality reduction technique used to transform a high-dimensional dataset into a lower-dimensional representation while preserving the most important information or patterns present in the original data. The main objective of PCA is to identify the principal components, which are new variables that are linear combinations of the original ones. This technique is widely used as a preprocessing step before applying other algorithms to improve computational efficiency and remove redundant information. So the basic idea is combining the screening factors through a PCA, to obtain a single score which retains just the significant information and avoids redundancy; then this new score will be then used to perform the screening of the stocks.

```
#Performing the PCA
from sklearn.decomposition import PCA

pca_df = []

pca = PCA(n_components=1)

for i in dates[1:]:
    dfs = []
    for df in (PBz, PEz, RSIz, dCONz):
        dfs.append(df.loc[:, i])

    dfs = pd.concat(dfs, axis=1)
    scores = pca.fit_transform(dfs.fillna(0)) # compute PCA score
    pca_df.append(pd.Series(scores.ravel()))

pca_df = pd.concat(pca_df, axis=1)
pca_df.index = df.index
pca_df.columns = dates[1:]

# Univariate screening on PCA scores
pca_returns, hlong, hshort = univariate_screening(pca_df, 1, how='long', ptf_size=50)
```

Now, the obtained PCA score was used to perform a univariate screening of the stocks taking only long positions.

Table 4.1: Performance metrics (PCA):

	Information ratio	Sharpe ratio	Traynor ratio	Sortino ratio
Screening PCA score	0.051	0.204	0.039	0.289

The IR is equal to 0.051 which indicates that the strategy performed slightly better than benchmark. This can be also noticed graphically, with the equity lines of the strategy and the benchamrk ending almost at the same level.

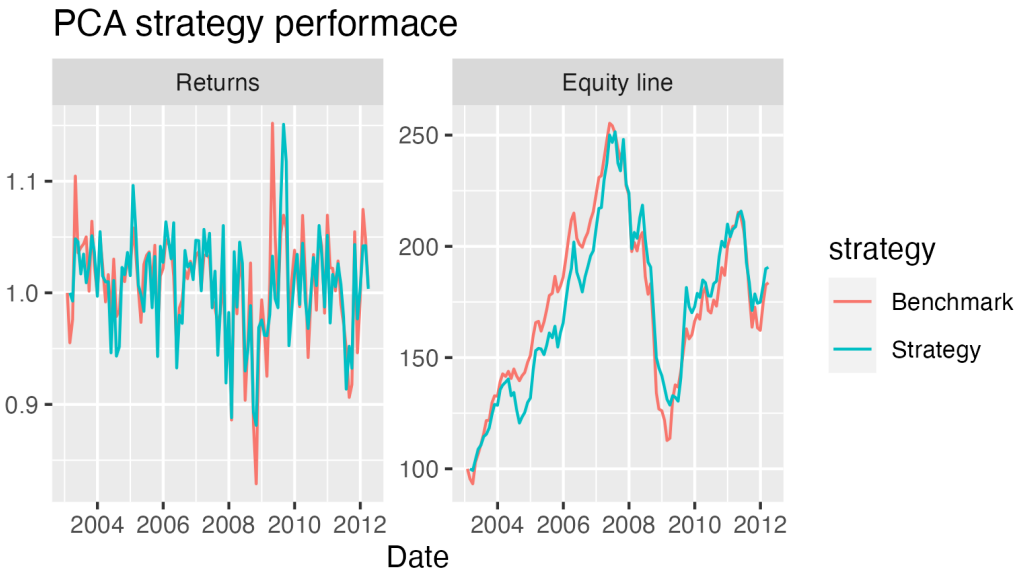

Figure 4.1: Results obtained by the strategy versus the benchmark (PCA screening).

While looking at the asset allocation is possible to notice that assets inside the portfolio tends to moderately change through time. In particular, they change all together in precise periods of time; this is a quite interesting phenomenon that should be due to cross-sectional nature of the PCA we computed.

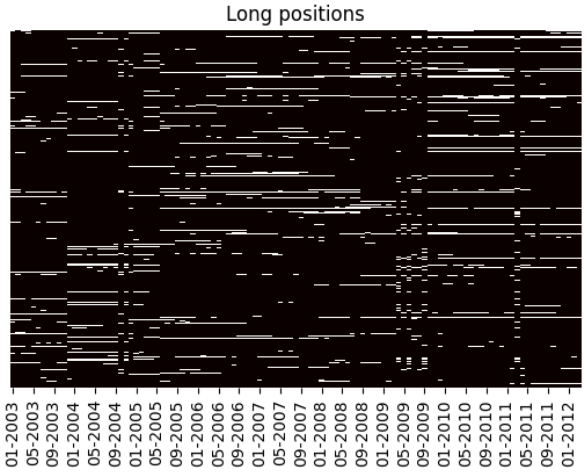

Figure 4.2: Pixel map, asset allocation of the strategy (PCA screening).

5 | Deep Learning to enhance Momentum Strategies

Inspired by [3], it was decided to apply deep learning strategies to enhance the performance of screening models. Deep learning is a branch of machine learning that focuses on building and training neural networks with multiple layers. These networks are designed to automatically learn hierarchical representations of data, allowing them to handle complex problems.
In the context of finance, deep learning techniques have been applied to a wide range of tasks [2] such as:

- *Stock Market Prediction*: DL models have been used for stock market prediction by leveraging historical price and trading volume data.
- *Portfolio Optimization*: DL can automatically learn to allocate assets in a portfolio to maximize returns while considering risk and transaction costs.
- *Fraud Detection*: by learning complex patterns and anomalies in transaction data, DL models enable the detection of fraudulent activities with high accuracy.
- *Credit Risk Assessment*: by analyzing large volumes of customer financial data, DL networks can extract relevant features and relationships to predict creditworthiness and assess default risks.
- *Algorithmic Trading*: by training on price data, DL networks can identify signals to make buy or sell decisions in an automated trading system.

In the project, deep learning has been employed to build a simple network to forecast log returns of each stock.

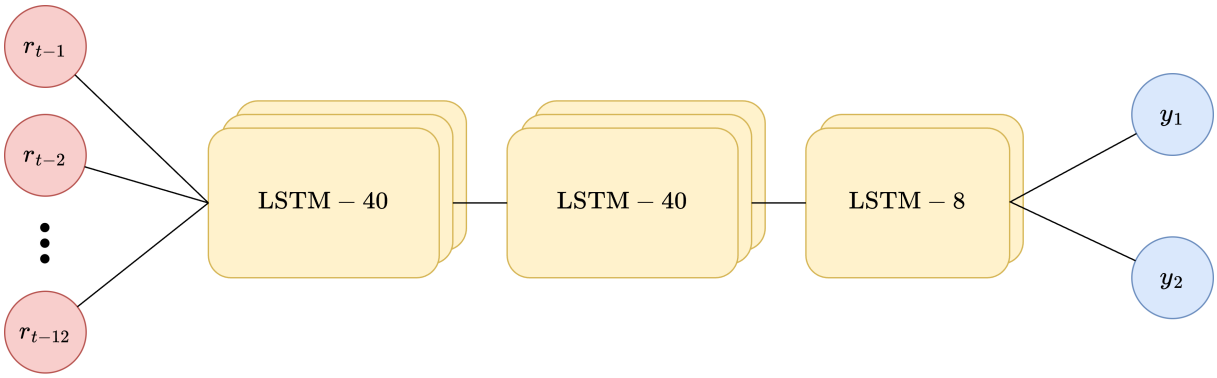

Figure 5.1: Deep learning model architecture.

The red circles represents the inputs and are the returns of 12 consecutive months, the blue circles represents the output of the model, which returns both the probabilities of having a positive or a negative return for the following month.
The yellow blocks are layers of Long Short-Term Memory (LSTM) modules [1]. They are a type of recurrent neural network (RNN) specifically designed to handle temporal data.
The model was trained on all stocks from January 2004 to May 2010, validated unitl April 2011, and tested on subsequent months. The following table shows the results

	Training	Validation	Test
Baseline	0.562	**0.657**	0.507
Deep Learning MOM	**0.596**	0.618	**0.515**

Table 5.1: Deep Learning performances.

The table shows the accuracy of the model on predicting whether the stock will have positive or negative returns. Even though the results aren't impressive, the model can beat a random guess on the test set by 0.8%.
Finally, for each stock, a series of predictions for the validation and test months is computed and aggregated into a single data frame. The latter was then used for univariate screening in both directions with 200 stocks and a holding peiod of one month.

	Information ratio	Sharpe ratio	Traynor ratio	Sortino ratio
Screening DL (TRAIN)	0.902	0.284	0.049	0.317
Screening DL (TEST)	-0.338	-0.035	-0.006	-0.063

Table 5.2: Performance metrics (DL screening).

While all metrics are positive for the training set, the ones relative to the test set are all negative. This is probably due to overfitting and reflects the difficulty of predicting prices only with their historical data. Since the strategy has to be evaluated on the test set, it is not performing well with respect to the benchmark. This can be noticed graphically (in the bottom-right chart), with the strategy's equity line ending slightly under the market's.

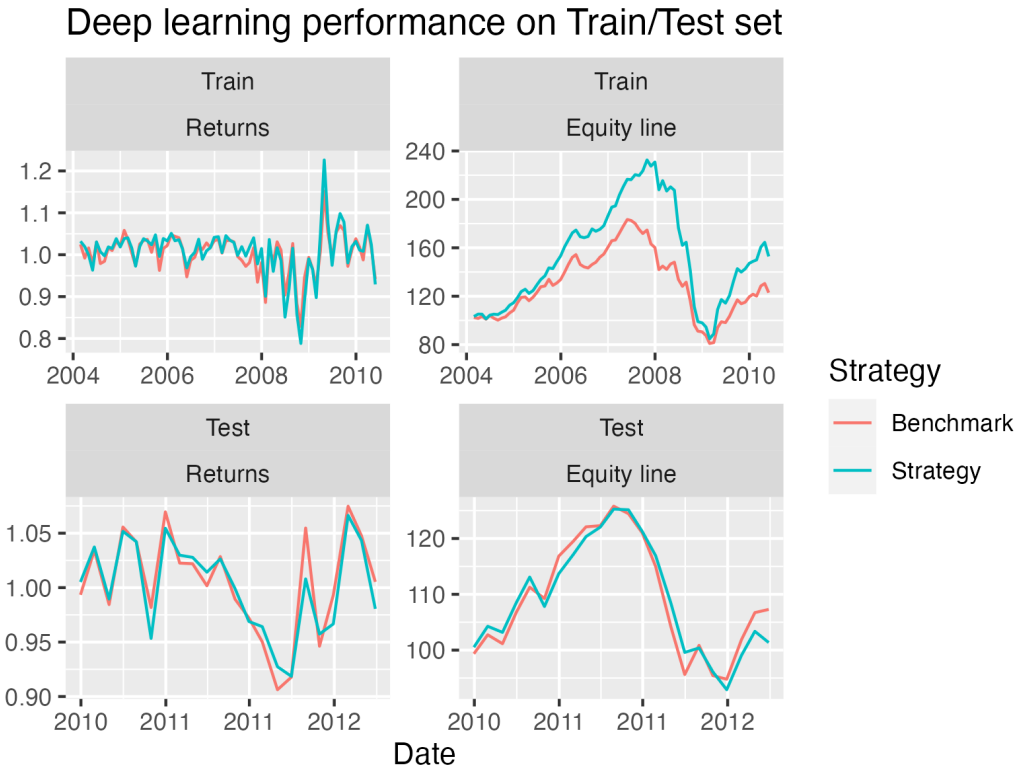

Figure 5.2: Results obtained by the strategy versus the benchmark (Deep Learning screening TRAIN + TEST).

Finally, on the training set the, equities, which are more (200), tend to change a lot through time. In the test set, position seems more stable but it has to be considered the fact that the time window is just of two years.

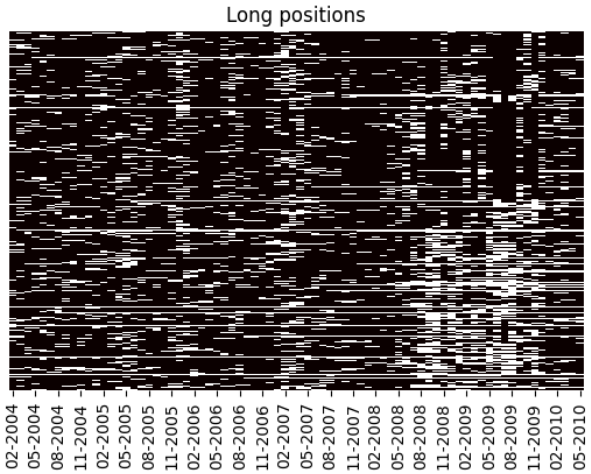

Figure 5.3: Pixel map, asset allocation of the strategy (Deep Learning screening TRAIN).

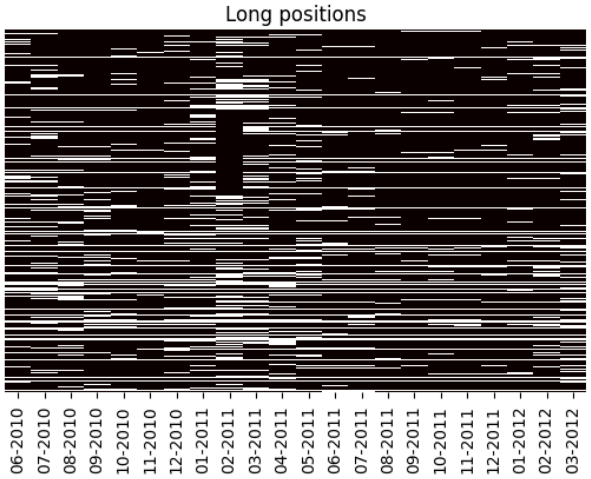

Figure 5.4: Pixel map, asset allocation of the strategy (Deep Learning screening TEST).

5.1 | Conclusions

Summing up, the results obtained by the different screening strategies deployed, it is possible to notice that the PB univariate screening strategy is the best one relatively to the benchmark with an IR of 0.803 and is also the best in terms of absolute metrics. The second best performing strategy is the simultaneous screening with an IR of 0.127, and also the PCA strategy has performed slightly above the benchmark with an IR of 0.051. The results obtained by the DL momentum strategy on the test set are all negative, so this strategy is not performing well with respect to the benchmark. Finally, sequential screening is the the strategy that has the worst performance.

	Information ratio	Sharpe ratio	Traynor ratio	Sortino ratio
PB_ratio screening	**0.803**	**0.517**	**0.090**	**0.631**
Simultaneous screening	0.127	0.186	0.034	0.227
Sequential screening	-0.727	-0.109	-0.018	-0.127
Screening PCA score	0.051	0.204	0.039	0.289
Screening DL (TEST)	-0.338	-0.035	-0.006	-0.063

Table 5.3: Summary performance metrics:

All the models were set to have a holding period of one month, so each month the screening is renewed with the new available factors and the portfolio composition is updated. The holding period impacts on the transaction costs, in fact, the more frequently the portfolio is updated, the more it will be affected by transaction costs. Due to this consideration, it was tried to see how the performances of all the strategies will change considering different holding periods.

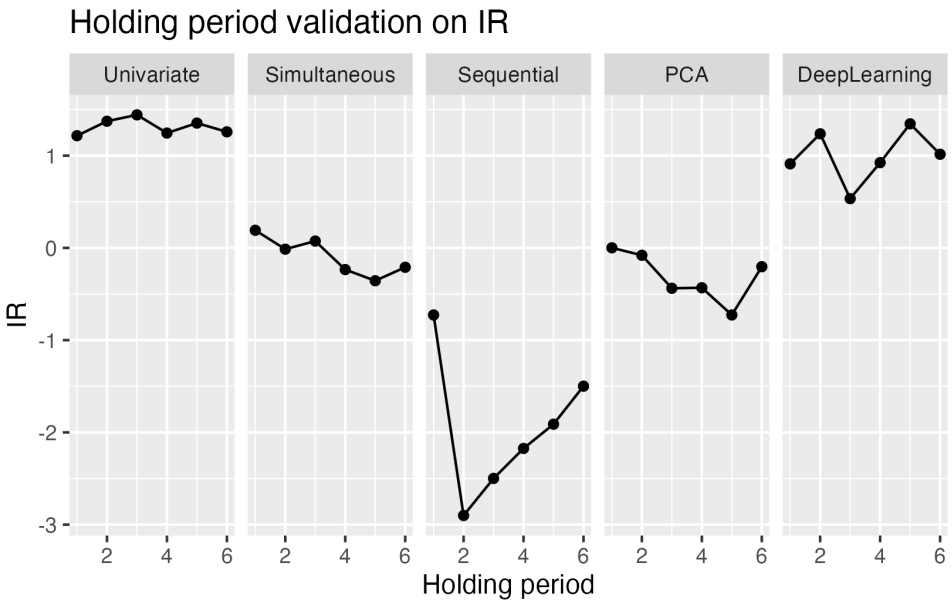

Figure 5.5: IR ratios applying the same strategies but with different holding periods.

In above chart, it's possible to notice how the IR changes by setting the strategy with different holding periods. In particular,

- For univariate screening the IR remains stable by varying the holding period. For the simultaneous screening the IR decreases.
- For the DL and PCA screenings, performance initially decreases by raising the holding period; but with an holding period of six month the IR re-increased to a value near to the one obtained initially.
- Sequential screening's IR decreases by changing the holding period from 1 to 2 months, but then re-increases for higher values.

6 | References

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[2] Jian Huang, Junyi Chai, and Stella Cho. Deep learning in finance and banking: A literature review and classification. *Frontiers of Business Research in China*, 14, 12 2020.

[3] Lawrence Takeuchi. Applying deep learning to enhance momentum trading strategies in stocks. 2013.