Table of Contents

Read the dataset

In [1]:

```
# Import modules
import pandas as pd
import matplotlib.pyplot as plt #描画ライブラリ
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
# import pathlib
# import glob
# import math

# Show all the rows and columns up to 200
pd.set_option('display.max_columns', 200)
pd.set_option('display.max_rows', 200)
```

In [2]:

```
# Data set for seller/buyer
df_prediction_afterDA = pd.read_csv('df_prediction.csv', sep=',', header=0)
df_prediction_afterDA = df_prediction_afterDA[["DateTime", "Spot", "High", "Low", "Close", "Close_pred"]]
df_prediction_afterDA.head()
```

Out[2]:

	DateTime	Spot	High	Low	Close	Close_pred
0	2017-08-10 00:00:00	7.98	12.98	5.94	9.4	7.89
1	2017-08-10 00:30:00	7.67	12.67	5.63	9.4	7.72
2	2017-08-10 01:00:00	6.86	11.86	5.58	9.4	7.26
3	2017-08-10 01:30:00	6.58	11.58	4.91	9.4	6.79
4	2017-08-10 02:00:00	6.62	11.62	4.88	9.4	6.82

In [3]:

```
# Data set for trader
df_prediction_beforeDA = pd.read_csv('df_prediction_trader.csv', sep=',', header=0)
df_prediction_beforeDA.head()
```

Out[3]:

	Date	HH	Open	High	Low	Close	Spot	Spot_1daylag	DateTime	Close_pred
0	2017-08-10	1	10.70	12.98	5.94	9.4	8.19	8.10	2017-08-10 00:00:00	7.83
1	2017-08-10	2	6.89	12.67	5.63	9.4	8.04	8.00	2017-08-10 00:30:00	7.58
2	2017-08-10	3	6.42	11.86	5.58	9.4	7.93	8.05	2017-08-10 01:00:00	7.55
3	2017-08-10	4	10.30	11.58	4.91	9.4	7.73	7.80	2017-08-10 01:30:00	7.27
4	2017-08-10	5	10.50	11.62	4.88	9.4	7.93	7.94	2017-08-10 02:00:00	7.35

Benchmark

Benchmark 1 (Trade only on DA market)

- No strategy: Players trade on DA market as much as possible to avoid imbalance risks (100% position only on DA market)
- Calculate return and risk with "Spot price"

Requierd dataset

In [4]:

```
# Need actual spot, high, close price
df_benchmark1 = df_prediction_afterDA.copy()

# Drop Close_pred and others for intraday market since players who trade without strategy need onl
y spot price.
df_benchmark1 = df_benchmark1.drop(["Close_pred", "High", "Low", "Close"], axis=1)

df_benchmark1.head()
```

Out[4]:

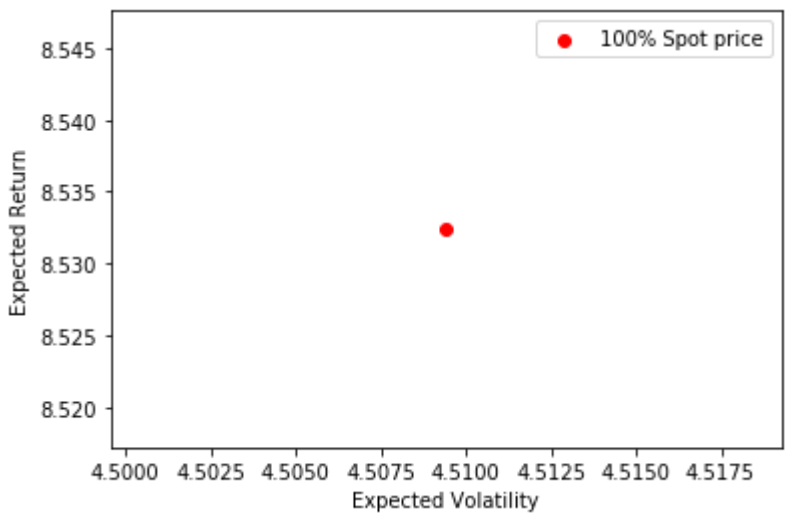
	DateTime	Spot
0	2017-08-10 00:00:00	7.98
1	2017-08-10 00:30:00	7.67
2	2017-08-10 01:00:00	6.86
3	2017-08-10 01:30:00	6.58
4	2017-08-10 02:00:00	6.62

Evaluation

In [5]:

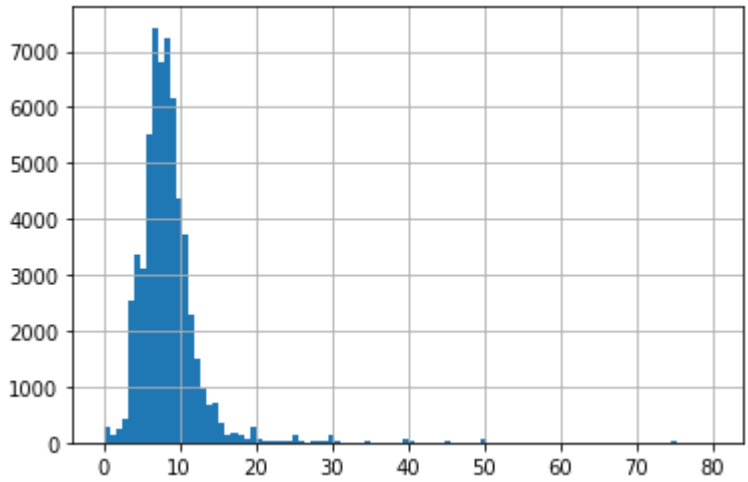
```
# fig, ax = plt.subplots(1, figsize=plt.figaspect(.25))
plt.scatter(x=df_benchmark1["Spot"].std(), y=df_benchmark1["Spot"].mean(), color="r", label="1
00% Spot price")
print("Expected return: {}".format(df_benchmark1["Spot"].mean().round(2)))
print("Standard deviation: {}".format(df_benchmark1["Spot"].std().round(2)))
print("Sharp ratio: {}".format(round(df_benchmark1["Spot"].mean()/df_benchmark1["Spot"].std
(), 2)))

plt.xlabel('Expected Volatility')
plt.ylabel('Expected Return')
plt.legend();
```

Expected return: 8.53
Standard deviation: 4.51
Sharp ratio: 1.89



In [6]:

```
df_benchmark1["Spot"].hist(bins=100);
```



Benchmark2 (Trade only on Intraday market)

- No strategy: Players trade on Intra market (100% position only on Intra market)
  --> ※ This strategy would be not realistic due to imbalence risk, but just try to look at the performance
- Calculate return and risk with "Close price"

Required dataset

In [7]:

```
# Need actual spot, high, close price
df_benchmark2 = df_prediction_afterDA.copy()

# Drop Close_pred and others for intraday market since players who trade without strategy need onl
y spot price.
df_benchmark2 = df_benchmark2.drop(["Close_pred", "High", "Low", "Spot"], axis=1)

df_benchmark2.head()
```

Out[7]:

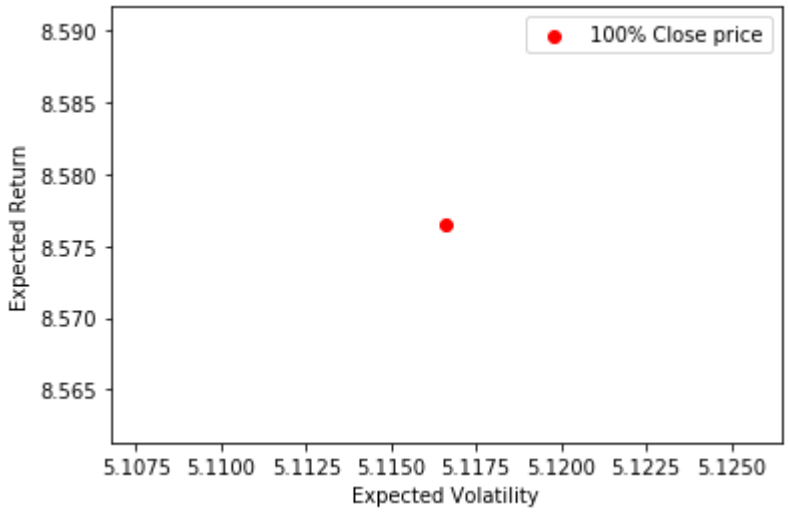
	DateTime	Close
0	2017-08-10 00:00:00	9.4
1	2017-08-10 00:30:00	9.4
2	2017-08-10 01:00:00	9.4
3	2017-08-10 01:30:00	9.4
4	2017-08-10 02:00:00	9.4

Evaluation

In [8]:

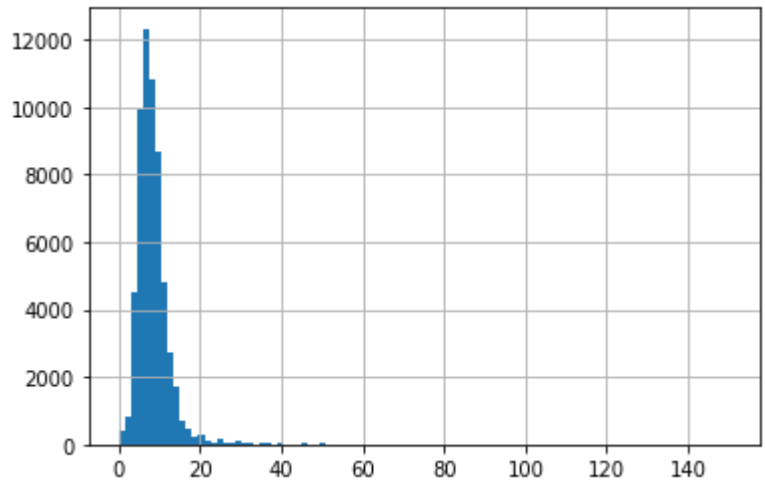
```
# fig, ax = plt.subplots(1, figsize=plt.figaspect(.25))
plt.scatter(x=df_benchmark2["Close"].std(), y=df_benchmark2["Close"].mean(), color="r", label=
"100% Close price")
print("Expected return: {}".format(df_benchmark2["Close"].mean().round(2)))
print("Standard deviation: {}".format(df_benchmark2["Close"].std().round(2)))
print("Sharp ratio: {}".format(round(df_benchmark2["Close"].mean()/df_benchmark2["Close"].std
(), 2)))

plt.xlabel('Expected Volatility')
plt.ylabel('Expected Return')
plt.legend();
```

Expected return: 8.58
Standard deviation: 5.12
Sharp ratio: 1.68



In [9]:

```
df_benchmark2["Close"].hist(bins=100);
```



As a result, both expected return and standard deviation on Intraday market are higher than that on DA market

--> Sharp ratio on Intraday market in lower than that of DA market

--> Return is high, but TOO RISKY

※ The benchmark1 will be compared with the following strategies.

- Strategies for seller aim to sell their posision for higher price than the benchmark
- Strategies for buyer aim to buy their posision for lower price than the benchmark

Trading strategies for sellers (e.g. power producers)

[Assumption]

1) All of the market participants are price-takers and their trading decisions do not affect the price on any market.

2) Auction strategy at the DA market is ignored. Realised spot price is used for evaluating the portfolios since spot price is determined based on a single price auction system.

3) Strategic decisions are made based only on the price before each prediction point, and trading performance is evaluated based on realised price after the market.

4) Trading volume is ignored. Therefore, transaction fees are also ignored. Return and risk will be defined in the next section.

5) The iceberg order is ignored since the information of the ask/bid order book is not available.

6) All the orders can be executed as far as the price of the order is within the range of the high and low price of that day. It also means there are no penalties due to imbalance.

7) For electricity producers, the influence of curtailment on their operating and trading decision making is ignored. Curtailment is restriction of generating electricity especially for renewable energy power plants.

8) For retailers, the influence of supply contracts with customers on trading decision making is ignored.

Strategy 1 (Trading with prediction before DA) without execution strategy

- Choose the higher price of 1day-lagged spot price or the predicted close price(before DA)
  --> Determinint trading position depends on the prices

In [10]:

```
Benchmark_seller = df_benchmark1["Spot"].round(2)
Benchmark_seller
```

Out[10]:

```
0       7.98
1       7.67
2       6.86
3       6.58
4       6.62
       ...
59515   35.00
59516   45.00
59517   40.00
59518   35.00
59519   25.00
Name: Spot, Length: 59520, dtype: float64
```

Required dataset

In [11]:

```
# Need predicted price before DA market and 1 dayahead spot price
df_seller_strategy1 = df_prediction_beforeDA.copy()
df_seller_strategy1["Spot_1dayahead"] = df_seller_strategy1["Spot"].shift(48)
# Drop rows that include NaN
df_seller_strategy1 = df_seller_strategy1.dropna(how='any', axis=0).reset_index(drop=True)

# Make lists for price information
SpotLag_list = list(df_seller_strategy1["Spot_1dayahead"])
Spot_list = list(df_seller_strategy1["Spot"])
High_list = list(df_seller_strategy1["High"])
Close_list = list(df_seller_strategy1["Close"])
Pred_list = list(df_seller_strategy1["Close_pred"])
# Judge_success = []

# list for executed orders
Executed = []
Judge = []
Position = []
for sl, s, h, c, p  in zip(SpotLag_list, Spot_list, High_list, Close_list, Pred_list):
    # Trade on DA market
    if sl >= p:
        Executed.append(s)
        Judge.append("True")
        Position.append("DA")
    # Trade on Intra markets
    else:
        if p <= h:
            Executed.append(p)
            Judge.append("True")
            Position.append("Intra")
        else:
            Executed.append(c)
            Judge.append("False")
            Position.append("Intra")
df_seller_strategy1["ExecutedOrder"] = pd.Series(Executed)
df_seller_strategy1["Judge"] = pd.Series(Judge)
df_seller_strategy1["Position"] = pd.Series(Position)

#     Judge_success.append((df_seller_bench["Judge_" + str(i) + "%"] == 'True').sum())
# #For graph
# Judge_success = pd.Series(Judge_success)
```

In [12]:

```
df_seller_strategy1.tail()
```

Out[12]:

	Date	HH	Open	High	Low	Close	Spot	Spot_1daylag	DateTime	Close_pred	Sp
59467	2020-12-31	44	35.0	70.0	33.00	70.00	35.0	50.00	2020-12-31 21:30:00	18.81	
59468	2020-12-31	45	42.0	70.0	41.01	45.48	40.0	40.00	2020-12-31 22:00:00	19.55	
59469	2020-12-31	46	42.0	70.0	35.00	41.33	40.0	40.00	2020-12-31 22:30:00	17.98	
59470	2020-12-31	47	37.0	70.0	33.93	36.66	35.0	33.21	2020-12-31 23:00:00	14.78	
59471	2020-12-31	48	27.0	37.5	23.93	26.46	25.0	25.00	2020-12-31 23:30:00	12.57	

Evaluation

In [13]:

```
# The percentage of DA position
DA_position = df_seller_strategy1["Position"].value_counts()[0]
Intra_position = df_seller_strategy1["Position"].value_counts()[1]
DA_ratio = round(DA_position / (DA_position+Intra_position)*100, 2)
DA_ratio
```

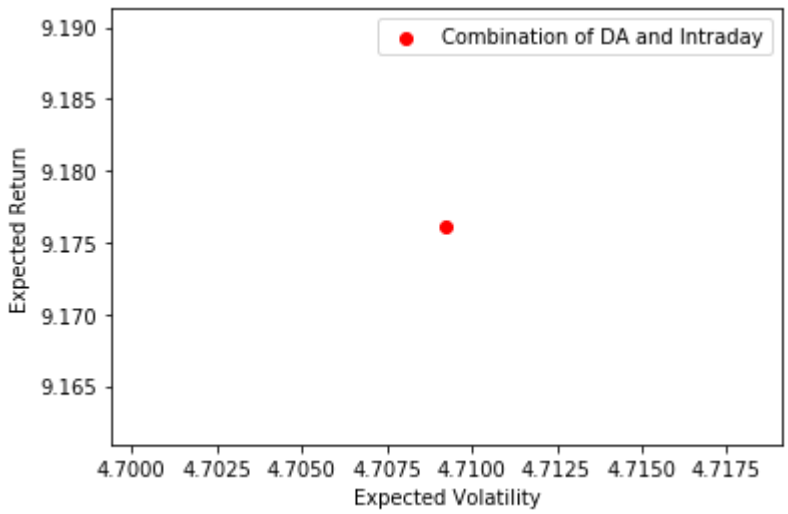
Out[13]:

64.11

In [14]:

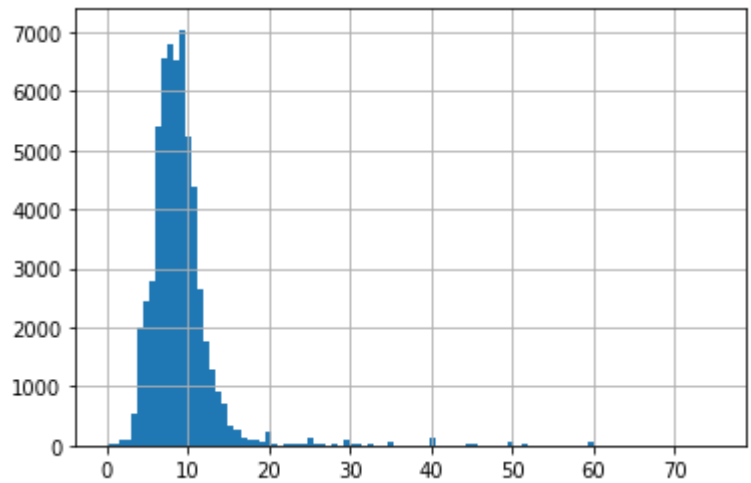
```
# fig, ax = plt.subplots(1, figsize=plt.figaspect(.25))
plt.scatter(x=df_seller_strategy1["ExecutedOrder"].std(), y=df_seller_strategy1["ExecutedOrder"]
.mean(), color="r", label="Combination of DA and Intraday")
print("Position: DA {}".format(round(DA_position / (DA_position+Intra_position)*100, 2)) + "%")
print("Expected return: {}".format(df_seller_strategy1["ExecutedOrder"].mean().round(2)))
print("Standard deviation: {}".format(df_seller_strategy1["ExecutedOrder"].std().round(2)))
print("Sharp ratio: {}".format(round(df_seller_strategy1["ExecutedOrder"].mean()/df_seller_strate
gy1["ExecutedOrder"].std(), 2)))
print("Information ratio: {}".format(round((df_seller_strategy1["ExecutedOrder"] - Benchmark_selle
r).mean()/(df_seller_strategy1["ExecutedOrder"] - Benchmark_seller).std(), 2)))

plt.xlabel('Expected Volatility')
plt.ylabel('Expected Return')
plt.legend();
```

Position: DA 64.11%
Expected return: 9.18
Standard deviation: 4.71
Sharp ratio: 1.95
Information ratio: 0.2



In [15]:

```
df_seller_strategy1["ExecutedOrder"].hist(bins=100);
```



Much higher than benchmarks

Strategy 1 with execusion strategy

- Derive strategy1 with execution strategy
  ※ 10-fold evaluation is necessary

Required dataset

In [16]:

```
# Need predicted price before DA market and 1 dayahead spot price
df_seller_strategy1_ex = df_prediction_beforeDA.copy()
df_seller_strategy1_ex["Spot_1dayahead"] = df_seller_strategy1_ex["Spot"].shift(48)
# Drop rows that include NaN
df_seller_strategy1_ex = df_seller_strategy1_ex.dropna(how='any', axis=0).reset_index(drop=Tru
e)

# Make lists for price information
SpotLag_list = list(df_seller_strategy1_ex["Spot_1dayahead"])
Spot_list = list(df_seller_strategy1["Spot"])
High_list = list(df_seller_strategy1_ex["High"])
Close_list = list(df_seller_strategy1_ex["Close"])
# Judge_success = []

# Order and execution with execusion range
for i in list(range(0, 51, 1)):
    # list for executed orders
    Executed_i = []
    # list for checking the execution results
    Judge_i = []
    Position_i = []
    # Derive the predicted price with the range from +0% ~ +20%
    Pred_list_i = list((df_seller_strategy1_ex["Close_pred"]* (1 + i/100)).round(2))
    for sl, s, h, c, p in zip(SpotLag_list, Spot_list, High_list, Close_list, Pred_list_i):
        # Trade on DA market
        if sl >= p:
            Executed_i.append(s)
            Judge.append("True")
            Position_i.append("DA")
        # Trade on Intra markets
        else:
            if p <= h:
                Executed_i.append(p)
                Judge_i.append("True")
                Position_i.append("Intra")
            else:
                Executed_i.append(c)
                Judge_i.append("False")
                Position_i.append("Intra")
    df_seller_strategy1_ex["Exec_" + str(i) + "%"] = pd.Series(Executed_i)
    df_seller_strategy1_ex["Judge_" + str(i) + "%"] = pd.Series(Judge_i)
    df_seller_strategy1_ex["Position_" + str(i) + "%"] = pd.Series(Position_i)

#     Judge_success.append((df_seller_bench["Judge_" + str(i) + "%"] == 'True').sum())
# #For graph
# Judge_success = pd.Series(Judge_success)
```

In [17]:

```
df_seller_strategy1_ex.head()
```

Out[17]:

	Date	HH	Open	High	Low	Close	Spot	Spot_1daylag	DateTime	Close_pred	Spot_1d
0	2017-08-11	1	6.86	11.50	5.39	7.41	7.74	8.19	2017-08-11 00:00:00	7.55	
1	2017-08-11	2	10.71	10.71	6.06	6.50	6.71	8.04	2017-08-11 00:30:00	7.48	
2	2017-08-11	3	10.23	10.23	5.76	6.02	6.24	7.93	2017-08-11 01:00:00	7.12	
3	2017-08-11	4	10.07	10.07	5.61	5.86	6.07	7.73	2017-08-11 01:30:00	6.80	
4	2017-08-11	5	10.09	10.09	4.89	5.88	6.09	7.93	2017-08-11 02:00:00	6.77	

Evaluation

- Seek the best execution point

In [18]:

```
n_splits=100

train_size = df_seller_strategy1_ex.index[-1]
sample_size = int(train_size/n_splits)
train_index_list = list(np.linspace(sample_size, train_size, n_splits, endpoint = True, dtype='int'))
train_index_list

# # Confirming the split logic
# for train_index in train_index_list:
#     # Divide the train/valid set  into 10 folds and pick up it.
#     X_train = df_seller_strategy1_ex.iloc[:train_index]
#     print("start:", train_index - sample_size)
#     print("end:", train_index)
```

Out[18]:

```
[594,
 1188,
 1783,
 2378,
 2972,
 3567,
 4162,
 4757,
 5351,
 5946,
 6541,
 7135,
 7730,
 8325,
 8920,
 9514,
 10109,
 10704,
 11298,
 11893,
 12488,
 13083,
 13677,
 14272,
 14867,
 15461,
 16056,
 16651,
 17246,
 17840,
 18435,
 19030,
 19624,
 20219,
 20814,
 21409,
 22003,
 22598,
 23193,
 23787,
 24382,
 24977,
 25572,
 26166,
 26761,
 27356,
 27950,
 28545,
 29140,
 29735,
 30329,
 30924,
 31519,
 32114,
 32708,
 33303,
 33898,
 34492,
 35087,
```

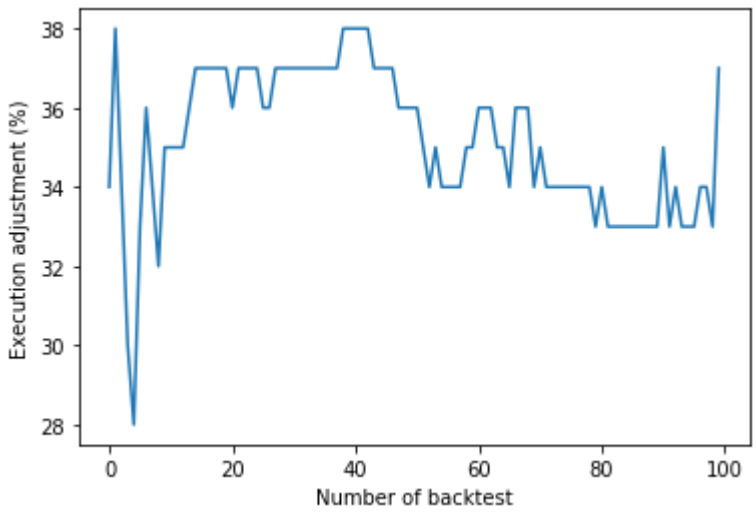
35682,
36277,
36871,
37466,
38061,
38655,
39250,
39845,
40440,
41034,
41629,
42224,
42818,
43413,
44008,
44603,
45197,
45792,
46387,
46981,
47576,
48171,
48766,
49360,
49955,
50550,
51144,
51739,
52334,
52929,
53523,
54118,
54713,
55307,
55902,
56497,
57092,
57686,
58281,
58876,
59471]

In [19]:

```
BestExec = []

for train_index in train_index_list:
    X_train = df_seller_strategy1_ex[:train_index]

    cols = []

    # List for the results for evaluation
    PortfolioReturn = []
    StandardDeviation = []
    Max = []
    Min = []
    SharpRatio = []
    InformationRatio = []

    for i in list(range(0, 51, 1)):
        Return_i = []
        Exec_list_i = list(X_train["Exec_" + str(i) + "%"])
        for e in Exec_list_i:
            Return_i.append(e)

        Return_i = pd.Series(Return_i)
        PortfolioReturn.append(Return_i.mean().round(2))
        StandardDeviation.append(Return_i.std().round(2))
        Max.append(Return_i.max().round(2))
        Min.append(Return_i.min().round(2))
        SharpRatio.append((Return_i.mean()/Return_i.std()).round(2))
        InformationRatio.append(round((Return_i - Benchmark_seller).mean()/(Return_i - Benchmark_seller).std(), 3))

        # Make columns names
        cols.append(i)

    # Make dataframe for evaluation and  switch columns and row.
    df_seller_strategy1_ex_eval = pd.DataFrame()
    # df_seller_strategy1_ex_eval = df_seller_strategy1_ex_eval.T

    # Add columns for evaluation metrics
    df_seller_strategy1_ex_eval["PortfolioReturn"] = pd.Series(PortfolioReturn)
    df_seller_strategy1_ex_eval["StandardDeviation"] = pd.Series(StandardDeviation)
    df_seller_strategy1_ex_eval["Max"] = pd.Series(Max)
    df_seller_strategy1_ex_eval["Min"] = pd.Series(Min)
    df_seller_strategy1_ex_eval["SharpRatio"] = pd.Series(SharpRatio)
    df_seller_strategy1_ex_eval["InformationRatio"] = pd.Series(InformationRatio)

    df_seller_strategy1_ex_eval.index = cols
    df_seller_strategy1_ex_eval =  df_seller_strategy1_ex_eval.reset_index()
    df_seller_strategy1_ex_eval = df_seller_strategy1_ex_eval.rename(columns={"index": "ExecBuffer(%)"})

    Exec = df_seller_strategy1_ex_eval[df_seller_strategy1_ex_eval["InformationRatio"] == df_seller_strategy1_ex_eval["InformationRatio"].max()]
    Exec = Exec["ExecBuffer(%)"][Exec["StandardDeviation"] == Exec["StandardDeviation"].min()].iloc[0]
    BestExec.append(Exec)
```

In [20]:

```
# The best execution of 100 different periods
BestExec = pd.Series(BestExec)
plt.xlabel('Number of backtest')
plt.ylabel('Execution adjustment (%)')
BestExec.plot();
```



In [21]:

```
Best_i = BestExec.iloc[-1]
Best_i
```

Out[21]:

37

- Plot all the possible portfolios

In [22]:

```
cols = []

# List for the results for evaluation
PortfolioReturn = []
StandardDeviation = []
Max = []
Min = []
SharpRatio = []
InformationRatio = []

for i in list(range(0, 51, 1)):
    Return_i = []
    Exec_list_i = list(df_seller_strategy1_ex["Exec_" + str(i) + "%"])
    for e in Exec_list_i:
        Return_i.append(e)

    Return_i = pd.Series(Return_i)
    PortfolioReturn.append(Return_i.mean().round(2))
    StandardDeviation.append(Return_i.std().round(2))
    Max.append(Return_i.max().round(2))
    Min.append(Return_i.min().round(2))
    SharpRatio.append((Return_i.mean()/Return_i.std()).round(2))
    InformationRatio.append(round((Return_i - Benchmark_seller).mean()/(Return_i - Benchmark_sell
er).std(), 3))

    # Make columns names
    cols.append("Exec+" + str(i) + "%")

# Make dataframe for evaluation and  switch columns and row.
df_seller_strategy1_ex_eval = pd.DataFrame()

#各算出結果をdf_portfolio_benchの列へ追加する
df_seller_strategy1_ex_eval["PortfolioReturn"] = pd.Series(PortfolioReturn)
df_seller_strategy1_ex_eval["StandardDeviation"] = pd.Series(StandardDeviation)
df_seller_strategy1_ex_eval["Max"] = pd.Series(Max)
df_seller_strategy1_ex_eval["Min"] = pd.Series(Min)
df_seller_strategy1_ex_eval["SharpRatio"] = pd.Series(SharpRatio)
df_seller_strategy1_ex_eval["InformationRatio"] = pd.Series(InformationRatio)

df_seller_strategy1_ex_eval.index = cols
```

In [23]:

```
df_seller_strategy1_ex_eval.head()
```

Out[23]:

	PortfolioReturn	StandardDeviation	Max	Min	SharpRatio	InformationRatio
Exec+0%	9.18	4.71	75.1	0.01	1.95	0.202
Exec+1%	9.19	4.70	75.1	0.01	1.95	0.206
Exec+2%	9.20	4.70	75.1	0.01	1.96	0.210
Exec+3%	9.21	4.68	75.1	0.01	1.97	0.215
Exec+4%	9.22	4.66	75.1	0.01	1.98	0.220

In [24]:

```
# Pick up the portfolios on the global minimum variance portfolio
std_min = df_seller_strategy1_ex_eval[df_seller_strategy1_ex_eval["StandardDeviation"] == df_s
eller_strategy1_ex_eval["StandardDeviation"].min()]
GMVP = std_min[std_min["PortfolioReturn"] == std_min["PortfolioReturn"].max()].reset_index()
GMVP
```

Out[24]:

	index	PortfolioReturn	StandardDeviation	Max	Min	SharpRatio	InformationRatio
0	Exec+17%	9.5	4.58	75.1	0.01	2.08	0.317

In [25]:

```
Best = df_seller_strategy1_ex_eval[df_seller_strategy1_ex_eval["InformationRatio"] == df_seller_
strategy1_ex_eval["InformationRatio"].max()]
Best = Best[Best["StandardDeviation"] == Best["StandardDeviation"].min()]
Best
```

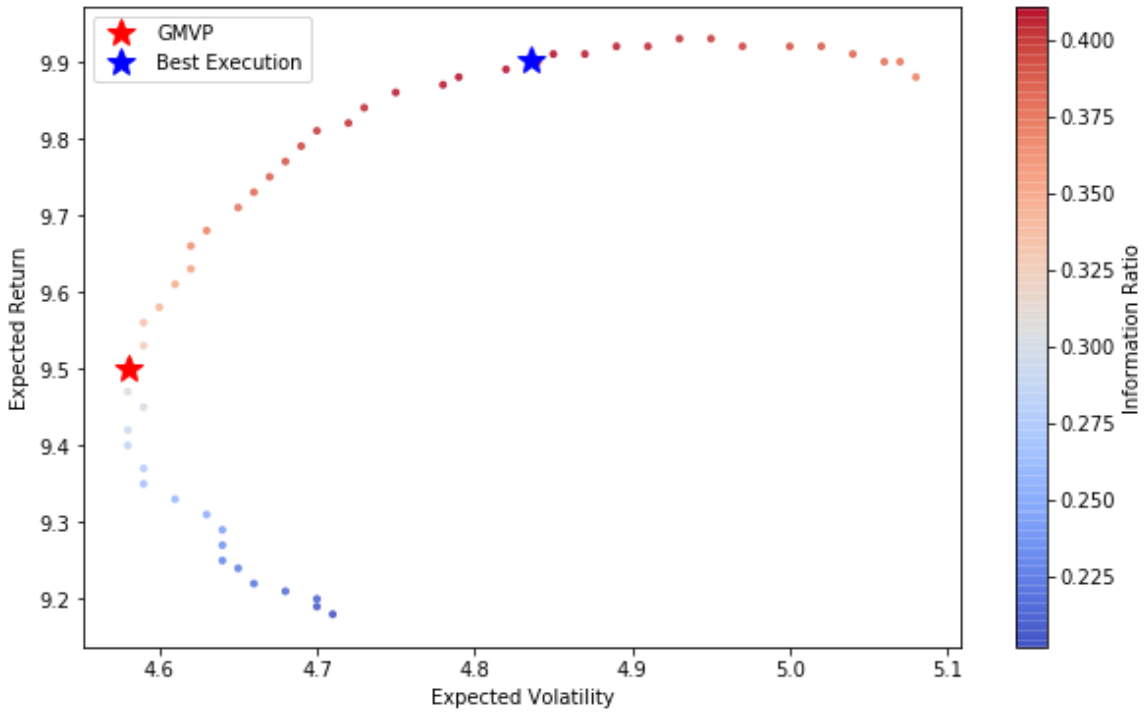
Out[25]:

	PortfolioReturn	StandardDeviation	Max	Min	SharpRatio	InformationRatio
Exec+37%	9.9	4.84	75.1	0.01	2.05	0.411

In [26]:

```
plt.figure(figsize=(10,6))

# Scatter plot all the possible portfolios
plt.scatter(df_seller_strategy1_ex_eval["StandardDeviation"], df_seller_strategy1_ex_eval["Portf
olioReturn"], c=df_seller_strategy1_ex_eval["InformationRatio"], marker='.', alpha=0.8, cmap='cool
warm')

# Global minimum variance portfolio
i = 17
DA_position_i = df_seller_strategy1_ex["Position_" + str(i) + "%"].value_counts()[0]
Intra_position_i = df_seller_strategy1_ex["Position_" + str(i) + "%"].value_counts()[1]

plt.plot(GMVP["StandardDeviation"], GMVP["PortfolioReturn"], 'r*', markersize=15.0, label="GMVP")
print("[Global Minimum Variance Portfolio]")
print("Position: DA {}".format(round(DA_position_i / (DA_position_i+Intra_position_i)*100, 2)) +
"%")
print("Expected return: {}".format(GMVP["PortfolioReturn"][0]))
print("Standard deviation: {}".format(GMVP["StandardDeviation"][0]))
print("Sharp ratio: {}".format(GMVP["SharpRatio"][0]))
print("Information ratio: {}".format(GMVP["InformationRatio"][0]))

# The Best Execusion
i = Best_i
DA_position_i = df_seller_strategy1_ex["Position_" + str(i) + "%"].value_counts()[0]
Intra_position_i = df_seller_strategy1_ex["Position_" + str(i) + "%"].value_counts()[1]

plt.plot(df_seller_strategy1_ex["Exec_" + str(i) + "%"].std(), df_seller_strategy1_ex["Exec_" + str
(i) + "%"].mean(), "b*", markersize=15.0, label="Best Execution")
print("[Best Execution Portfolio]")
print("Position: DA {}".format(round(DA_position_i / (DA_position_i+Intra_position_i)*100, 2)) +
"%")
print("Expected return: {}".format(df_seller_strategy1_ex["Exec_" + str(i) + "%"].mean().round(2
)))
print("Standard deviation: {}".format(df_seller_strategy1_ex["Exec_" + str(i) + "%"].std().round(2
)))
print("Sharp ratio: {}".format(round(df_seller_strategy1_ex["Exec_" + str(i) + "%"].mean() / df_sel
ler_strategy1_ex["Exec_" + str(i) + "%"].std(),2)))
print("Information ratio: {}".format(round((df_seller_strategy1_ex["Exec_" + str(i) + "%"] - Bench
mark_seller).mean() / (df_seller_strategy1_ex["Exec_" + str(i) + "%"] - Benchmark_seller).std(),2
)))

plt.xlabel('Expected Volatility')
plt.ylabel('Expected Return')
plt.colorbar(label='Information Ratio')
plt.legend(loc="upper left");
```

[Global Minimum Variance Portfolio]
Position: DA 73.14%
Expected return: 9.5
Standard deviation: 4.58
Sharp ratio: 2.08
Information ratio: 0.317
[Best Execution Portfolio]
Position: DA 92.51%
Expected return: 9.9
Standard deviation: 4.84
Sharp ratio: 2.05
Information ratio: 0.41



Strategy 2 (Trading with prediction after DA)

--> Players have the information of spot price and predicted price after DA market

1) Players trade with pre-determined portfolio proportion that ranges from 0% to 100%.

2) Positions are derived with execution range if players have the possition of intraday market

Required dataset

In [27]:

```
# Need predicted price after DA market and spot price
df_seller_strategy2 = df_prediction_afterDA.copy()

# Make lists for price information
Spot_list = list(df_seller_strategy2["Spot"])
High_list = list(df_seller_strategy2["High"])
Close_list = list(df_seller_strategy2["Close"])
Pred_list_i = list(df_seller_strategy2["Close_pred"])
# Judge_success = []

# list for executed orders
Executed_i = []
# list for checking the execution results
Judge_i = []

# Calculate executed price
for h, c, p in zip(High_list, Close_list, Pred_list_i):
    # Trade on Intra markets
    if p <= h:
        Executed_i.append(p)
        Judge_i.append("True")
    else:
        Executed_i.append(c)
        Judge_i.append("False")

df_seller_strategy2["ExecutedOrder"] = pd.Series(Executed_i)
df_seller_strategy2["Judge"] = pd.Series(Judge_i)

df_seller_strategy2.tail()
```

Out[27]:

	DateTime	Spot	High	Low	Close	Close_pred	ExecutedOrder	Judge
59515	2020-12-31 21:30:00	35.0	70.0	33.00	70.00	27.89	27.89	True
59516	2020-12-31 22:00:00	45.0	70.0	41.01	45.48	32.48	32.48	True
59517	2020-12-31 22:30:00	40.0	70.0	35.00	41.33	28.67	28.67	True
59518	2020-12-31 23:00:00	35.0	70.0	33.93	36.66	25.51	25.51	True
59519	2020-12-31 23:30:00	25.0	37.5	23.93	26.46	19.23	19.23	True

Evaluation

In [28]:

```
portfolio_weights= [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

cols = []

Return = []
PortfolioReturn = []
StandardDeviation = []
SharpRatio = []
InformationRatio = []

Spot_list = list(df_seller_strategy2["Spot"])
Exec_list = list(df_seller_strategy2["ExecutedOrder"])

for weight in portfolio_weights:
    Return = []
    for spot, intra in zip(Spot_list, Exec_list):
        # DA market 100%
        if weight == 1.0:
            Return.append(spot*weight)
        # Trade both on DA and Intra day
        else:
            Return.append(spot*weight + intra*(1-weight))
    # Set colmun name
    cols.append("DA" + str(weight*100) + "%")

    # Results
    Return = pd.Series(Return)
    PortfolioReturn.append(Return.mean().round(2))
    StandardDeviation.append(Return.std().round(2))
    SharpRatio.append((Return.mean()/Return.std()).round(2))
    InformationRatio.append(round((Return - Benchmark_seller).mean() / (Return - Benchmark_seller)
.std(), 3))

# Transpose columns and rows
df_seller_strategy2_eval = pd.DataFrame()
df_seller_strategy2_eval = df_seller_strategy2_eval.T

# Add columns for evaluation metrics
df_seller_strategy2_eval["PortfolioReturn"] = pd.Series(PortfolioReturn)
df_seller_strategy2_eval["StandardDeviation"] = pd.Series(StandardDeviation)
df_seller_strategy2_eval["Max"] = pd.Series(Max)
df_seller_strategy2_eval["Min"] = pd.Series(Min)
df_seller_strategy2_eval["SharpRatio"] = pd.Series(SharpRatio)
df_seller_strategy2_eval["InformationRatio"] = pd.Series(InformationRatio)
df_seller_strategy2_eval.index = cols
```

In [29]:

```
df_seller_strategy2_eval
```

Out[29]:

	PortfolioReturn	StandardDeviation	Max	Min	SharpRatio	InformationRatio
DA0%	8.30	3.56	75.1	0.01	2.33	-0.195
DA10.0%	8.33	3.65	75.1	0.01	2.28	-0.195
DA20.0%	8.35	3.74	75.1	0.01	2.23	-0.195
DA30.0%	8.37	3.83	75.1	0.01	2.18	-0.195
DA40.0%	8.40	3.93	75.1	0.01	2.14	-0.195
DA50.0%	8.42	4.02	75.1	0.01	2.09	-0.195
DA60.0%	8.44	4.12	75.1	0.01	2.05	-0.195
DA70.0%	8.46	4.21	75.1	0.01	2.01	-0.195
DA80.0%	8.49	4.31	75.1	0.01	1.97	-0.195
DA90.0%	8.51	4.41	75.1	0.01	1.93	-0.195
DA100.0%	8.53	4.51	75.1	0.01	1.89	NaN

In [30]:

```
# Pick up the portfolios on the global minimum variance portfolio
std_min = df_seller_strategy2_eval[df_seller_strategy2_eval["StandardDeviation"] == df_seller_st
rategy2_eval["StandardDeviation"].min()]
GMVP = std_min[std_min["PortfolioReturn"] == std_min["PortfolioReturn"].max()].reset_index()
GMVP
```

Out[30]:

	index	PortfolioReturn	StandardDeviation	Max	Min	SharpRatio	InformationRatio
0	DA0%	8.3	3.56	75.1	0.01	2.33	-0.195

In [31]:

```
# Pick up the best portrolio
Best = df_seller_strategy2_eval[df_seller_strategy2_eval["InformationRatio"] == df_seller_strateg
y2_eval["InformationRatio"].max()]
Best = Best[Best["StandardDeviation"] == Best["StandardDeviation"].min()]
Best
```

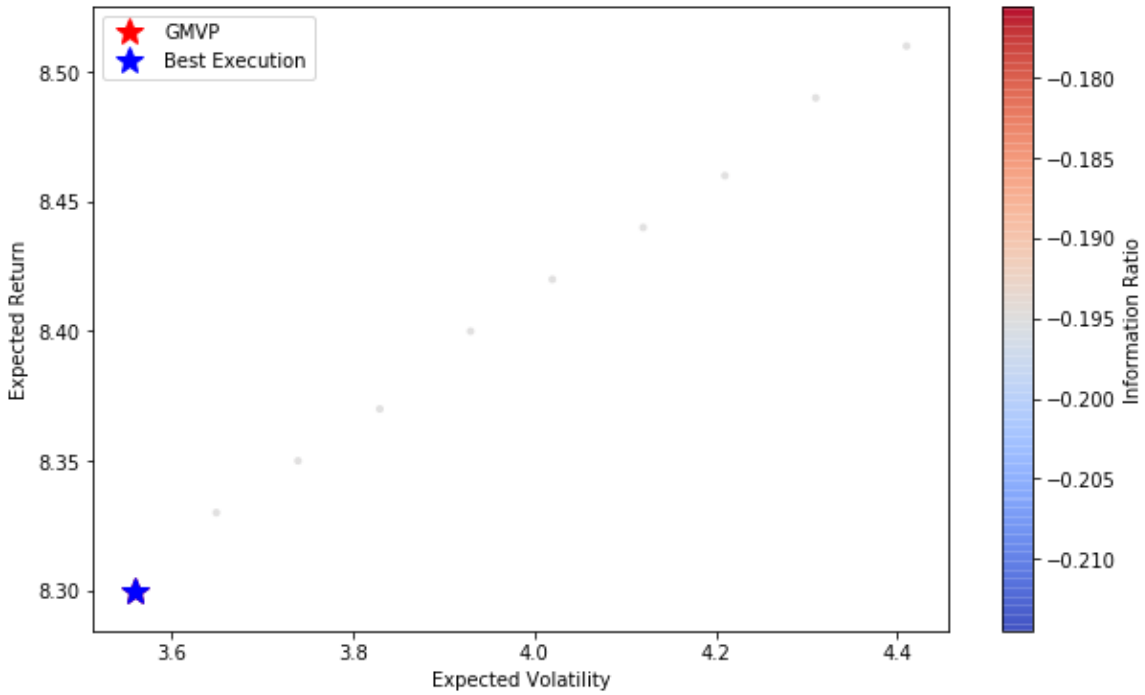
Out[31]:

	PortfolioReturn	StandardDeviation	Max	Min	SharpRatio	InformationRatio
DA0%	8.3	3.56	75.1	0.01	2.33	-0.195

In [32]:

```
plt.figure(figsize=(10,6))

# Scatter plot all the possible portfolios
plt.scatter(df_seller_strategy2_eval["StandardDeviation"], df_seller_strategy2_eval["PortfolioRetu
rn"], c=df_seller_strategy2_eval["InformationRatio"], marker='.', alpha=0.8, cmap='coolwarm')

# Global minimum variance portfolio
plt.plot(GMVP["StandardDeviation"], GMVP["PortfolioReturn"], 'r*', markersize=15.0, label="GMVP")
print("[Global Minimum Variance Portfolio]")
print("Expected return: {}".format(GMVP["PortfolioReturn"][0]))
print("Standard deviation: {}".format(GMVP["StandardDeviation"][0]))
print("Sharp ratio: {}".format(GMVP["SharpRatio"][0]))
print("Information ratio: {}".format(GMVP["InformationRatio"][0]))

# The Best Execusion
plt.plot(Best["StandardDeviation"], Best["PortfolioReturn"], "b*", markersize=15.0, label="Best Exe
cution")
print("[Best Execution Portfolio]")
print("Expected return: {}".format(Best["PortfolioReturn"][0]))
print("Standard deviation: {}".format(Best["StandardDeviation"][0]))
print("Sharp ratio: {}".format(Best["SharpRatio"][0]))
print("Information ratio: {}".format(Best["InformationRatio"][0]))

plt.xlabel('Expected Volatility')
plt.ylabel('Expected Return')
plt.colorbar(label='Information Ratio')
plt.legend(loc="upper left");
```

[Global Minimum Variance Portfolio]
Expected return: 8.3
Standard deviation: 3.56
Sharp ratio: 2.33
Information ratio: -0.195
[Best Execution Portfolio]
Expected return: 8.3
Standard deviation: 3.56
Sharp ratio: 2.33
Information ratio: -0.195



Strategy 2 with execusion strategy

--> Players have the information of spot price and predicted price after DA market

1) Players trade with pre-determined portfolio proportion that ranges from 0% to 100%.

2) Positions is derived with execution range if players have the possition of intraday market

Required dataset

In [33]:

```
# Need predicted price after DA market and spot price
df_seller_strategy2_ex = df_prediction_beforeDA.copy()

# Make lists for price information
Spot_list = list(df_seller_strategy2_ex["Spot"])
High_list = list(df_seller_strategy2_ex["High"])
Close_list = list(df_seller_strategy2_ex["Close"])

# Order and execution with execusion range
for i in list(range(0, 51, 1)):
    # list for executed orders
    Executed_i = []
    # list for checking the execution results
    Judge_i = []
    # Derive the predicted price with the range
    Pred_list_i = list((df_seller_strategy2_ex["Close_pred"]* (1 + i/100)).round(2))
    for h, c, p in zip(High_list, Close_list, Pred_list_i):
        # Trade on Intra markets
        if p <= h:
            Executed_i.append(p)
            Judge_i.append("True")
        else:
            Executed_i.append(c)
            Judge_i.append("False")

    df_seller_strategy2_ex["Exec_" + str(i) + "%"] = pd.Series(Executed_i)
    df_seller_strategy2_ex["Judge_" + str(i) + "%"] = pd.Series(Judge_i)
```

In [34]:

```
df_seller_strategy2_ex.tail()
```

Out[34]:

	Date	HH	Open	High	Low	Close	Spot	Spot_1daylag	DateTime	Close_pred	Ex
59515	2020-12-31	44	35.0	70.0	33.00	70.00	35.0	50.00	2020-12-31 21:30:00	18.81	
59516	2020-12-31	45	42.0	70.0	41.01	45.48	40.0	40.00	2020-12-31 22:00:00	19.55	
59517	2020-12-31	46	42.0	70.0	35.00	41.33	40.0	40.00	2020-12-31 22:30:00	17.98	
59518	2020-12-31	47	37.0	70.0	33.93	36.66	35.0	33.21	2020-12-31 23:00:00	14.78	
59519	2020-12-31	48	27.0	37.5	23.93	26.46	25.0	25.00	2020-12-31 23:30:00	12.57	

Evaluation

- Look for the best execution

In [35]:

```
n_splits=100

train_size = df_seller_strategy2_ex.index[-1]
sample_size = int(train_size/n_splits)
train_index_list = list(np.linspace(sample_size, train_size, n_splits, endpoint = True, dtype='int'))
train_index_list

# # Confirming the split logic
# for train_index in train_index_list:
#     # Divide the train/valid set  into 10 folds and pick up it.
#     X_train = df_seller_strategy2_ex.iloc[:train_index]
#     print("start:", train_index - sample_size)
#     print("end:", train_index)
```

Out[35]:

```
[595,
 1190,
 1785,
 2380,
 2975,
 3570,
 4166,
 4761,
 5356,
 5951,
 6546,
 7142,
 7737,
 8332,
 8927,
 9522,
 10118,
 10713,
 11308,
 11903,
 12498,
 13094,
 13689,
 14284,
 14879,
 15474,
 16069,
 16665,
 17260,
 17855,
 18450,
 19045,
 19641,
 20236,
 20831,
 21426,
 22021,
 22617,
 23212,
 23807,
 24402,
 24997,
 25593,
 26188,
 26783,
 27378,
 27973,
 28569,
 29164,
 29759,
 30354,
 30949,
 31544,
 32140,
 32735,
 33330,
 33925,
 34520,
 35116,
```

35711,
36306,
36901,
37496,
38092,
38687,
39282,
39877,
40472,
41068,
41663,
42258,
42853,
43448,
44044,
44639,
45234,
45829,
46424,
47019,
47615,
48210,
48805,
49400,
49995,
50591,
51186,
51781,
52376,
52971,
53567,
54162,
54757,
55352,
55947,
56543,
57138,
57733,
58328,
58923,
59519]

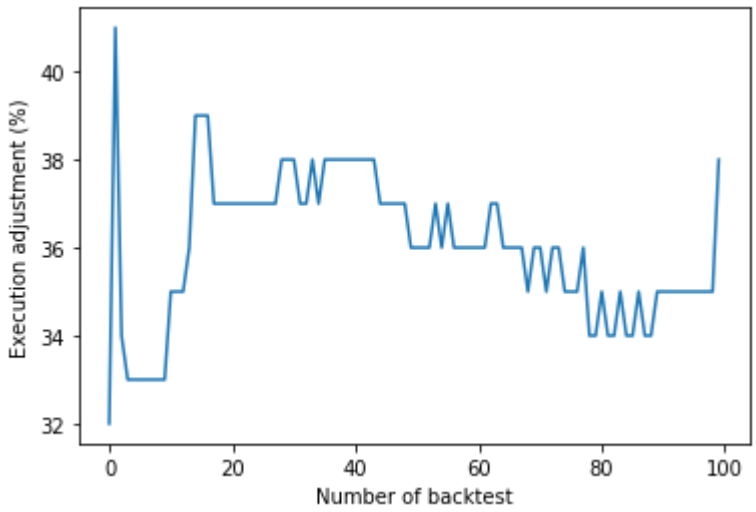
In [36]:

```
# This code is used only for confirming the best execution point
BestExec = []

for train_index in train_index_list:
    X_train = df_seller_strategy2_ex[:train_index]
    Spot_list = list(X_train["Spot"])

    cols = []
    PortfolioReturn = []
    StandardDeviation = []
    SharpRatio = []
    InformationRatio = []

    portfolio_weights= [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

    for i in list(range(0, 51, 1)):
        for weight in portfolio_weights:
            Return_iw = []
            Exec_list = list(X_train["Exec_" + str(i) + "%"])
            for spot, intra in zip(Spot_list, Exec_list):
                # DA market 100%
                if weight == 1.0:
                    Return_iw.append(spot*weight)
                # Trade both on DA and Intra day
                else:
                    Return_iw.append(spot*weight + intra*(1-weight))
            # Set colmun name
            if weight == 1.0:
                cols.append(str(0))
            else:
                cols.append(str(i))

            # Results
            Return_iw = pd.Series(Return_iw)
            PortfolioReturn.append(Return_iw.mean().round(2))
            StandardDeviation.append(Return_iw.std().round(2))
            SharpRatio.append((Return_iw.mean()/Return_iw.std()).round(2))
            InformationRatio.append(round((Return_iw - Benchmark_seller).mean() / (Return_iw - Bench
mark_seller).std(), 3))

    # Results of every portfolios derived by execution strateties
    df_seller_strategy2_ex_eval = pd.DataFrame()
    df_seller_strategy2_ex_eval = df_seller_strategy2_ex_eval.T

    # Add columns for evaluation metrics
    df_seller_strategy2_ex_eval["PortfolioReturn"] = pd.Series(PortfolioReturn)
    df_seller_strategy2_ex_eval["StandardDeviation"] = pd.Series(StandardDeviation)
    df_seller_strategy2_ex_eval["SharpRatio"] = pd.Series(SharpRatio)
    df_seller_strategy2_ex_eval["InformationRatio"] = pd.Series(InformationRatio)

    df_seller_strategy2_ex_eval.index = cols
    df_seller_strategy2_ex_eval = df_seller_strategy2_ex_eval.reset_index()
    df_seller_strategy2_ex_eval = df_seller_strategy2_ex_eval.rename(columns={"index": "ExecBuff
er(%)"})

    Exec = df_seller_strategy2_ex_eval[df_seller_strategy2_ex_eval["InformationRatio"] == df_selle
r_strategy2_ex_eval["InformationRatio"].max()]
    Exec = Exec["ExecBuffer(%)"][Exec["StandardDeviation"] == Exec["StandardDeviation"].min()].il
```

```
oc[0]
    BestExec.append(Exec)
```

In [37]:

```
BestExec = pd.Series(BestExec).astype(int)
plt.xlabel('Number of backtest')
plt.ylabel('Execution adjustment (%)')
BestExec.plot();
```



In [38]:

```
Best_i = BestExec.iloc[-1]
Best_i
```

Out[38]:

38

- Plot all the possible portfolios

In [39]:

```
Spot_list = list(df_seller_strategy2_ex["Spot"])

portfolio_weights= [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
execution_buffers = list(range(0, 51, 1))

cols = []

Return = []
PortfolioReturn = []
StandardDeviation = []
SharpRatio = []
InformationRatio = []

for i in execution_buffers:
    Exec_list = list(df_seller_strategy2_ex["Exec_" + str(i) + "%"])
    for weight in portfolio_weights:
        Return_iw = []
        for spot, intra in zip(Spot_list, Exec_list):
            # DA market 100%
            if weight == 1.0:
                Return_iw.append(spot*weight)
            # Trade both on DA and Intra day
            else:
                Return_iw.append(spot*weight + intra*(1-weight))
        # Set colmun name for execution buffer
        if weight == 1.0:
            cols.append("DA100%")
        else:
            cols.append("DA" + str(weight*100) + "%_Exec+" + str(i) + "%")

        # Results
        Return_iw = pd.Series(Return_iw)
        PortfolioReturn.append(Return_iw.mean().round(2))
        StandardDeviation.append(Return_iw.std().round(2))
        SharpRatio.append((Return_iw.mean()/Return_iw.std()).round(2))
        InformationRatio.append(round((Return_iw - Benchmark_seller).mean() / (Return_iw - Benchma
rk_seller).std(), 3))

# Transpose columns and rows
df_seller_strategy2_ex_eval = pd.DataFrame()
df_seller_strategy2_ex_eval = df_seller_strategy2_ex_eval.T

# Add columns for evaluation metrics
df_seller_strategy2_ex_eval["PortfolioReturn"] = pd.Series(PortfolioReturn)
df_seller_strategy2_ex_eval["StandardDeviation"] = pd.Series(StandardDeviation)
df_seller_strategy2_ex_eval["SharpRatio"] = pd.Series(SharpRatio)
df_seller_strategy2_ex_eval["InformationRatio"] = pd.Series(InformationRatio)

df_seller_strategy2_ex_eval.index = cols
```

In [40]:

```
# Pick up the portfolios on the global minimum variance portfolio
std_min = df_seller_strategy2_ex_eval[df_seller_strategy2_ex_eval["StandardDeviation"] == df_s
eller_strategy2_ex_eval["StandardDeviation"].min()]
GMVP = std_min[std_min["PortfolioReturn"] == std_min["PortfolioReturn"].max()].reset_index()
GMVP
```

Out[40]:

	index	PortfolioReturn	StandardDeviation	SharpRatio	InformationRatio
0	DA0%_Exec+0%	8.2	2.62	3.13	-0.117

In [41]:

```
# Pick up the best portfolio
Best = df_seller_strategy2_ex_eval[df_seller_strategy2_ex_eval["InformationRatio"] == df_seller_
strategy2_ex_eval["InformationRatio"].max()]
Best = Best[Best["StandardDeviation"] == Best["StandardDeviation"].min()]
Best
```

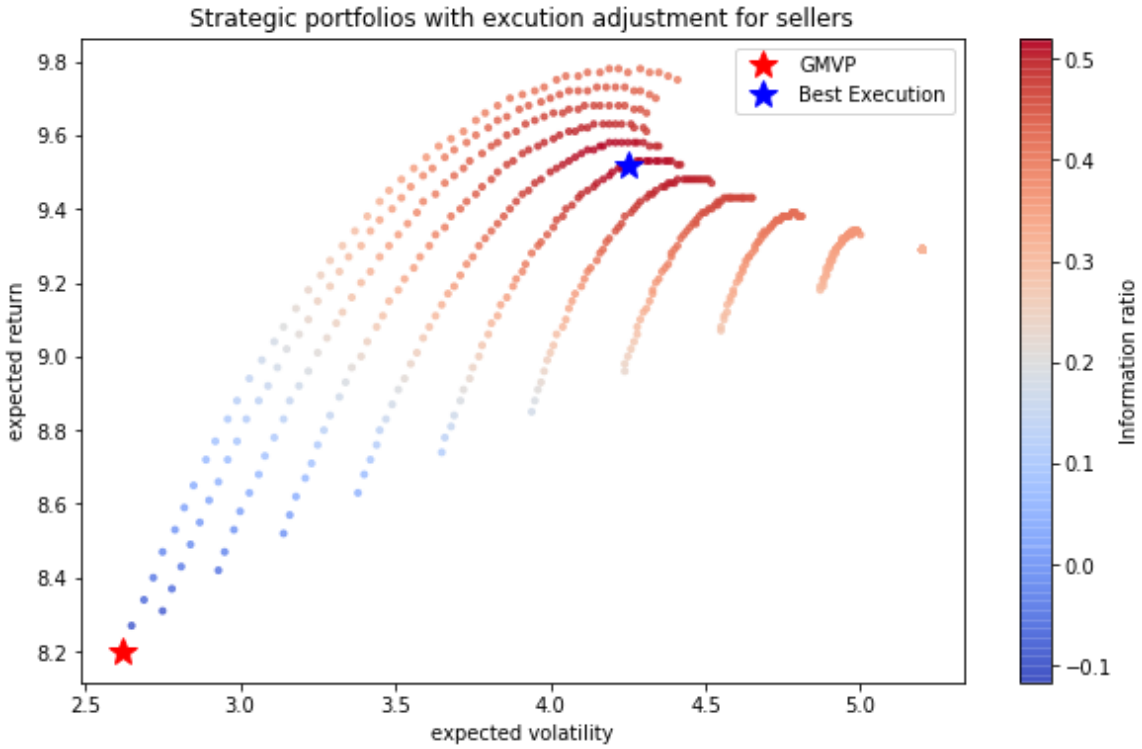
Out[41]:

	PortfolioReturn	StandardDeviation	SharpRatio	InformationRatio
DA50.0%_Exec+38%	9.52	4.25	2.24	0.521

In [42]:

```
plt.figure(figsize=(10,6))

# Scatter plot all the possible portfolios
plt.scatter(df_seller_strategy2_ex_eval["StandardDeviation"], df_seller_strategy2_ex_eval["Portf
olioReturn"], c=df_seller_strategy2_ex_eval["InformationRatio"], marker='.', alpha=0.8, cmap='cool
warm')

# Global minimum variance portfolio
plt.plot(GMVP["StandardDeviation"], GMVP["PortfolioReturn"], 'r*', markersize=15.0, label="GMVP")
print("[Global Minimum Variance Portfolio: " + str(GMVP.index[0]) + "]")
print("Expected return: {}".format(GMVP["PortfolioReturn"][0]))
print("Standard deviation: {}".format(GMVP["StandardDeviation"][0]))
print("Sharp ratio: {}".format(GMVP["SharpRatio"][0]))
print("Information ratio: {}".format(GMVP["InformationRatio"][0]))

# The Best Execusion
DA_weight = 0.5
i = Best_i
Best_Portfolio = (DA_weight * df_seller_strategy2_ex["Spot"]) + ((1 - DA_weight) * df_seller_stra
tegy2_ex["Exec_" + str(i) + "%"])
plt.plot(Best_Portfolio.std(), Best_Portfolio.mean(), "b*", markersize=15.0, label="Best Execution")
print("[Best Execution Portfolio: " + str(DA_weight) + "%_Exec+" + str(i) + "%]")
print("Expected return: {}".format(Best_Portfolio.mean().round(2)))
print("Standard deviation: {}".format(Best_Portfolio.std().round(2)))
print("Sharp ratio: {}".format(round(Best_Portfolio.mean() / Best_Portfolio.std(), 2)))
print("Information ratio: {}".format(round((Best_Portfolio - Benchmark_seller).mean() / (Best_Portf
olio - Benchmark_seller).std(), 3)))

plt.title('Strategic portfolios with excution adjustment for sellers')
plt.xlabel('expected volatility')
plt.ylabel('expected return')
plt.colorbar(label='Information ratio')
plt.legend(loc="upper right");
```

[Global Minimum Variance Portfolio: 0]
Expected return: 8.2
Standard deviation: 2.62
Sharp ratio: 3.13
Information ratio: -0.117
[Best Execution Portfolio: 0.5%_Exec+38%]
Expected return: 9.52
Standard deviation: 4.25
Sharp ratio: 2.24
Information ratio: 0.521



Trading strategy for buyers (e.g. retailers)

- Basically the same as the strategies for seller
- Buyer aims to buy for cheaper price

In [43]:

```
# Benchmark for buyer's strategy
Benchmark_buyer = -df_benchmark1["Spot"].round(2)
Benchmark_buyer
```

Out[43]:

```
0       -7.98
1       -7.67
2       -6.86
3       -6.58
4       -6.62
        ...
59515  -35.00
59516  -45.00
59517  -40.00
59518  -35.00
59519  -25.00
Name: Spot, Length: 59520, dtype: float64
```

Strategy 1 (Trading with prediction before DA) without execution strategy

Required dataset

In [44]:

```
# Need predicted price before DA market and 1 dayahead spot price
df_buyer_strategy1 = df_prediction_beforeDA.copy()
df_buyer_strategy1["Spot_1dayahead"] = df_buyer_strategy1["Spot"].shift(48)
# Drop rows that include NaN
df_buyer_strategy1 = df_buyer_strategy1.dropna(how='any', axis=0).reset_index(drop=True)

# Make lists for price information
SpotLag_list= list(df_buyer_strategy1["Spot_1dayahead"])
Spot_list = list(df_buyer_strategy1["Spot"])
Low_list = list(df_buyer_strategy1["Low"])
Close_list = list(df_buyer_strategy1["Close"])
Pred_list = list(df_buyer_strategy1["Close_pred"])
# Judge_success = []

# list for executed orders
Executed = []
Judge = []
Position = []
for sl, s, l, c, p  in zip(SpotLag_list, Spot_list, Low_list, Close_list, Pred_list):
    # Trade on DA market
    if sl <= p:
        Executed.append(-s)
        Judge.append("True")
        Position.append("DA")
    # Trade on Intra markets
    else:
        if p <= l:
            Executed.append(-p)
            Judge.append("True")
            Position.append("Intra")
        else:
            Executed.append(-c)
            Judge.append("False")
            Position.append("Intra")

df_buyer_strategy1["ExecutedOrder"] = pd.Series(Executed)
df_buyer_strategy1["Judge"] = pd.Series(Judge)
df_buyer_strategy1["Position"] = pd.Series(Position)

df_buyer_strategy1.tail()
```

Out[44]:

	Date	HH	Open	High	Low	Close	Spot	Spot_1daylag	DateTime	Close_pred	Sp
59467	2020-12-31	44	35.0	70.0	33.00	70.00	35.0	50.00	2020-12-31 21:30:00	18.81	
59468	2020-12-31	45	42.0	70.0	41.01	45.48	40.0	40.00	2020-12-31 22:00:00	19.55	
59469	2020-12-31	46	42.0	70.0	35.00	41.33	40.0	40.00	2020-12-31 22:30:00	17.98	
59470	2020-12-31	47	37.0	70.0	33.93	36.66	35.0	33.21	2020-12-31 23:00:00	14.78	
59471	2020-12-31	48	27.0	37.5	23.93	26.46	25.0	25.00	2020-12-31 23:30:00	12.57	

Evaluation

In [45]:

```
# The percentage of DA position
DA_position = df_buyer_strategy1["Position"].value_counts()[0]
Intra_position = df_buyer_strategy1["Position"].value_counts()[1]
DA_ratio = round(DA_position / (DA_position+Intra_position)*100, 2)
DA_ratio
```

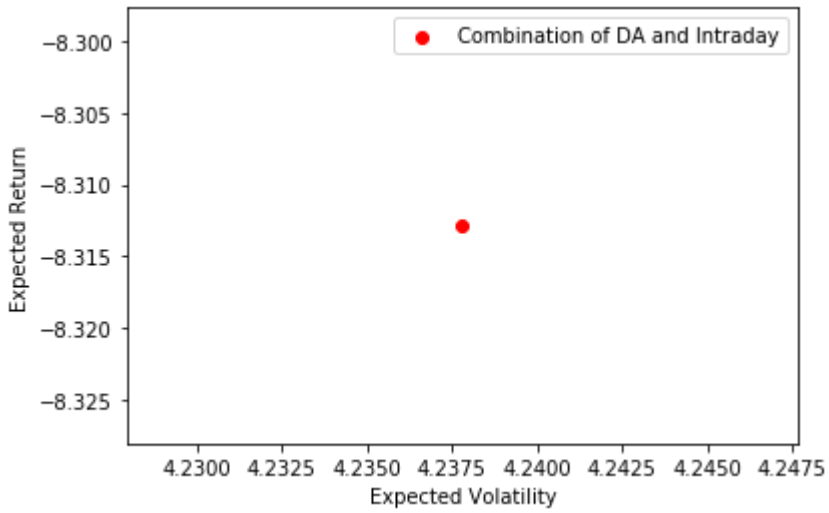
Out[45]:

63.82

In [46]:

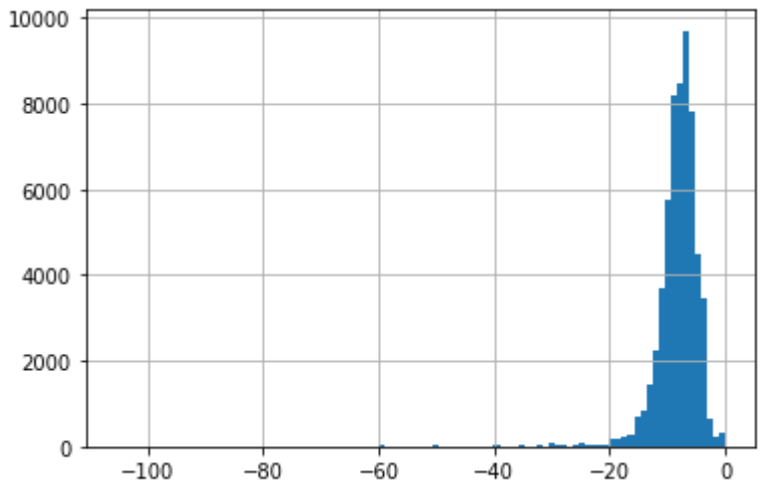
```
# fig, ax = plt.subplots(1, figsize=plt.figaspect(.25))
plt.scatter(x=df_buyer_strategy1["ExecutedOrder"].std(), y=df_buyer_strategy1["ExecutedOrder"
].mean(), color="r", label="Combination of DA and Intraday")
print("Position: DA {}".format(round(DA_position / (DA_position+Intra_position)*100, 2)) + "%")
print("Expected return: {}".format(df_buyer_strategy1["ExecutedOrder"].mean().round(2)))
print("Standard deviation: {}".format(df_buyer_strategy1["ExecutedOrder"].std().round(2)))
print("Sharp ratio: {}".format(round(df_buyer_strategy1["ExecutedOrder"].mean()/df_buyer_strat
egy1["ExecutedOrder"].std(),2)))
print("Information ratio: {}".format(round((df_buyer_strategy1["ExecutedOrder"] - Benchmark_buy
er).mean() / (df_buyer_strategy1["ExecutedOrder"] - Benchmark_buyer).std(),3)))

plt.xlabel('Expected Volatility')
plt.ylabel('Expected Return')
plt.legend();
```

Position: DA 63.82%
Expected return: -8.31
Standard deviation: 4.24
Sharp ratio: -1.96
Information ratio: 0.052



In [47]:

```
df_buyer_strategy1["ExecutedOrder"].hist(bins=100);
```



Strategy 1 with execusion strategy

Required dataset

In [48]:

```
# Need predicted price before DA market and 1 dayahead spot price
df_buyer_strategy1_ex = df_prediction_beforeDA.copy()
df_buyer_strategy1_ex["Spot_1dayahead"] = df_buyer_strategy1_ex["Spot"].shift(48)
# Drop rows that include NaN
df_buyer_strategy1_ex = df_buyer_strategy1_ex.dropna(how='any', axis=0).reset_index(drop=Tr
ue)

# Make lists for price information
SpotLag_list= list(df_buyer_strategy1_ex["Spot_1dayahead"])
Spot_list = list(df_buyer_strategy1_ex["Spot"])
Low_list = list(df_buyer_strategy1_ex["Low"])
Close_list = list(df_buyer_strategy1_ex["Close"])
# Judge_success = []

# Order and execution with execusion range
for i in list(range(0, 51, 1)):
    # list for executed orders
    Executed_i = []
    # list for checking the execution results
    Judge_i = []
    Position_i = []
    # Derive the predicted price with the range from +0% ~ +20%
    Pred_list_i = list((df_buyer_strategy1_ex["Close_pred"]* (1 - i/100)).round(2))
    for sl, s, l, c, p in zip(SpotLag_list, Spot_list, Low_list, Close_list, Pred_list_i):
        # Trade on DA market
        if sl <= p:
            Executed_i.append(-s)
            Judge_i.append("True")
            Position_i.append("DA")
        # Trade on Intra markets
        else:
            if p >= l:
                Executed_i.append(-p)
                Judge_i.append("True")
                Position_i.append("Intra")
            else:
                Executed_i.append(-c)
                Judge_i.append("False")
                Position_i.append("Intra")

    df_buyer_strategy1_ex["Exec_" + str(i) + "%"] = pd.Series(Executed_i)
    df_buyer_strategy1_ex["Judge_" + str(i) + "%"] = pd.Series(Judge_i)
    df_buyer_strategy1_ex["Position_" + str(i) + "%"] = pd.Series(Position_i)

df_buyer_strategy1_ex.head()
```

Out[48]:

	Date	HH	Open	High	Low	Close	Spot	Spot_1daylag	DateTime	Close_pred	Spot_1d
0	2017-08-11	1	6.86	11.50	5.39	7.41	7.74	8.19	2017-08-11 00:00:00	7.55	
1	2017-08-11	2	10.71	10.71	6.06	6.50	6.71	8.04	2017-08-11 00:30:00	7.48	
2	2017-08-11	3	10.23	10.23	5.76	6.02	6.24	7.93	2017-08-11 01:00:00	7.12	
3	2017-08-11	4	10.07	10.07	5.61	5.86	6.07	7.73	2017-08-11 01:30:00	6.80	
4	2017-08-11	5	10.09	10.09	4.89	5.88	6.09	7.93	2017-08-11 02:00:00	6.77	

Evaluation

In [49]:

```
n_splits=100

train_size = df_buyer_strategy1_ex.index[-1]
train_index_list = list(np.linspace(train_size/n_splits, train_size, n_splits, endpoint = True, dtype='in
t'))
train_index_list
```

Out[49]:

```
[594,
 1189,
 1784,
 2378,
 2973,
 3568,
 4162,
 4757,
 5352,
 5947,
 6541,
 7136,
 7731,
 8325,
 8920,
 9515,
 10110,
 10704,
 11299,
 11894,
 12488,
 13083,
 13678,
 14273,
 14867,
 15462,
 16057,
 16651,
 17246,
 17841,
 18436,
 19030,
 19625,
 20220,
 20814,
 21409,
 22004,
 22598,
 23193,
 23788,
 24383,
 24977,
 25572,
 26167,
 26761,
 27356,
 27951,
 28546,
 29140,
 29735,
 30330,
 30924,
 31519,
 32114,
 32709,
 33303,
 33898,
 34493,
 35087,
```

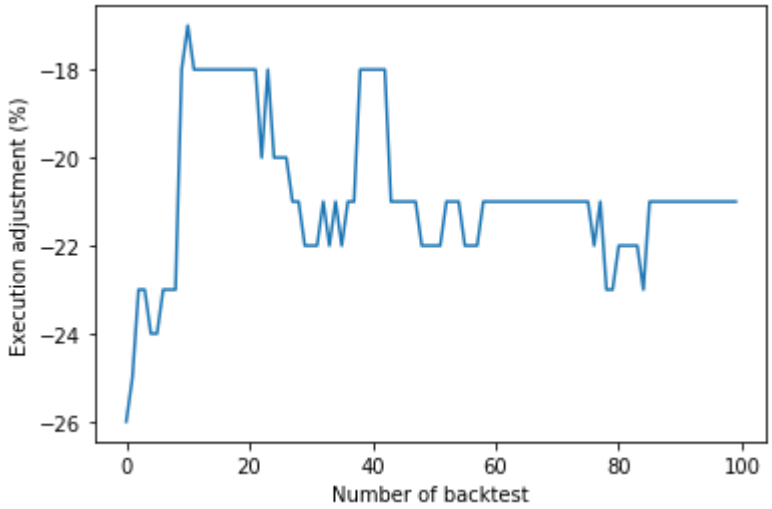
35682,
36277,
36872,
37466,
38061,
38656,
39250,
39845,
40440,
41034,
41629,
42224,
42819,
43413,
44008,
44603,
45197,
45792,
46387,
46982,
47576,
48171,
48766,
49360,
49955,
50550,
51145,
51739,
52334,
52929,
53523,
54118,
54713,
55308,
55902,
56497,
57092,
57686,
58281,
58876,
59471]

In [50]:

```
# This code is used only for confirming the best execution point
BestExec = []

for train_index in train_index_list:
    X_train = df_buyer_strategy1_ex[:train_index]

    cols = []

    # List for the results for evaluation
    PortfolioReturn = []
    StandardDeviation = []
    Max = []
    Min = []
    SharpRatio = []
    InformationRatio=[]

    for i in list(range(0, 51, 1)):
        Return_i = []
        Exec_list_i = list(X_train["Exec_" + str(i) + "%"])
        for e in Exec_list_i:
            Return_i.append(e)

        Return_i = pd.Series(Return_i)
        PortfolioReturn.append(Return_i.mean().round(2))
        StandardDeviation.append(Return_i.std().round(2))
        Max.append(Return_i.max().round(2))
        Min.append(Return_i.min().round(2))
        SharpRatio.append((Return_i.mean()/Return_i.std()).round(2))
        InformationRatio.append(round((Return_i - Benchmark_buyer).mean() / (Return_i - Benchmark
_buyer).std(), 3))

        # Make columns names
        cols.append(i)

    # Make dataframe for evaluation and  switch columns and row.
    df_buyer_strategy1_ex_eval = pd.DataFrame()

    # Add columns for evaluation metrics
    df_buyer_strategy1_ex_eval["PortfolioReturn"] = pd.Series(PortfolioReturn)
    df_buyer_strategy1_ex_eval["StandardDeviation"] = pd.Series(StandardDeviation)
    df_buyer_strategy1_ex_eval["Max"] = pd.Series(Max)
    df_buyer_strategy1_ex_eval["Min"] = pd.Series(Min)
    df_buyer_strategy1_ex_eval["SharpRatio"] = pd.Series(SharpRatio)
    df_buyer_strategy1_ex_eval["InformationRatio"] = pd.Series(InformationRatio)

    df_buyer_strategy1_ex_eval.index = cols
    df_buyer_strategy1_ex_eval = df_buyer_strategy1_ex_eval.reset_index()
    df_buyer_strategy1_ex_eval = df_buyer_strategy1_ex_eval.rename(columns={"index": "ExecBu
ffer(%)"})

    Exec = df_buyer_strategy1_ex_eval[df_buyer_strategy1_ex_eval["InformationRatio"] == df_bu
yer_strategy1_ex_eval["InformationRatio"].max()]
    Exec = -Exec["ExecBuffer(%)"][Exec["StandardDeviation"] == Exec["StandardDeviation"].min()].il
oc[0]
    BestExec.append(Exec)
```

In [51]:

```
# The best execution of 100 different periods
BestExec = pd.Series(BestExec)
plt.xlabel('Number of backtest')
plt.ylabel('Execution adjustment (%)')
BestExec.plot();
```



In [52]:

```
Best_i = BestExec.iloc[-1]
Best_i
```

Out[52]:

-21

In [53]:

```
cols = []

# List for the results for evaluation
PortfolioReturn = []
StandardDeviation = []
Max = []
Min = []
SharpRatio = []
InformationRatio = []

for i in list(range(0, 51, 1)):
    Return_i = []
    Exec_list_i = list(df_buyer_strategy1_ex["Exec_" + str(i) + "%"])
    for e in Exec_list_i:
        Return_i.append(e)

    Return_i = pd.Series(Return_i)
    PortfolioReturn.append(Return_i.mean().round(2))
    StandardDeviation.append(Return_i.std().round(2))
    Max.append(Return_i.max().round(2))
    Min.append(Return_i.min().round(2))
    SharpRatio.append((Return_i.mean()/Return_i.std()).round(2))
    InformationRatio.append(round((Return_i - Benchmark_buyer).mean() / (Return_i - Benchmark_b
uyer).std(), 3))

    # Make columns names
    cols.append("Exec+" + str(i) + "%")

# Make dataframe for evaluation and  switch columns and row.
df_buyer_strategy1_ex_eval = pd.DataFrame()

# Add columns for evaluation metrics
df_buyer_strategy1_ex_eval["PortfolioReturn"] = pd.Series(PortfolioReturn)
df_buyer_strategy1_ex_eval["StandardDeviation"] = pd.Series(StandardDeviation)
df_buyer_strategy1_ex_eval["Max"] = pd.Series(Max)
df_buyer_strategy1_ex_eval["Min"] = pd.Series(Min)
df_buyer_strategy1_ex_eval["SharpRatio"] = pd.Series(SharpRatio)
df_buyer_strategy1_ex_eval["InformationRatio"] = pd.Series(InformationRatio)

df_buyer_strategy1_ex_eval.index = cols
```

In [54]:

```
# Pick up the portfolios on the global minimum variance portfolio
std_min = df_buyer_strategy1_ex_eval[df_buyer_strategy1_ex_eval["StandardDeviation"] == df_
buyer_strategy1_ex_eval["StandardDeviation"].min()]
GMVP = std_min[std_min["PortfolioReturn"] == std_min["PortfolioReturn"].max()]
GMVP
```

Out[54]:

	PortfolioReturn	StandardDeviation	Max	Min	SharpRatio	InformationRatio
Exec+6%	-8.35	4.46	-0.01	-150.0	-1.87	0.047

In [55]:

```
# The best portfolio
Best = df_buyer_strategy1_ex_eval[df_buyer_strategy1_ex_eval["InformationRatio"] == df_buyer
_strategy1_ex_eval["InformationRatio"].max()]
Best = Best[Best["StandardDeviation"] == Best["StandardDeviation"].min()]
Best
```

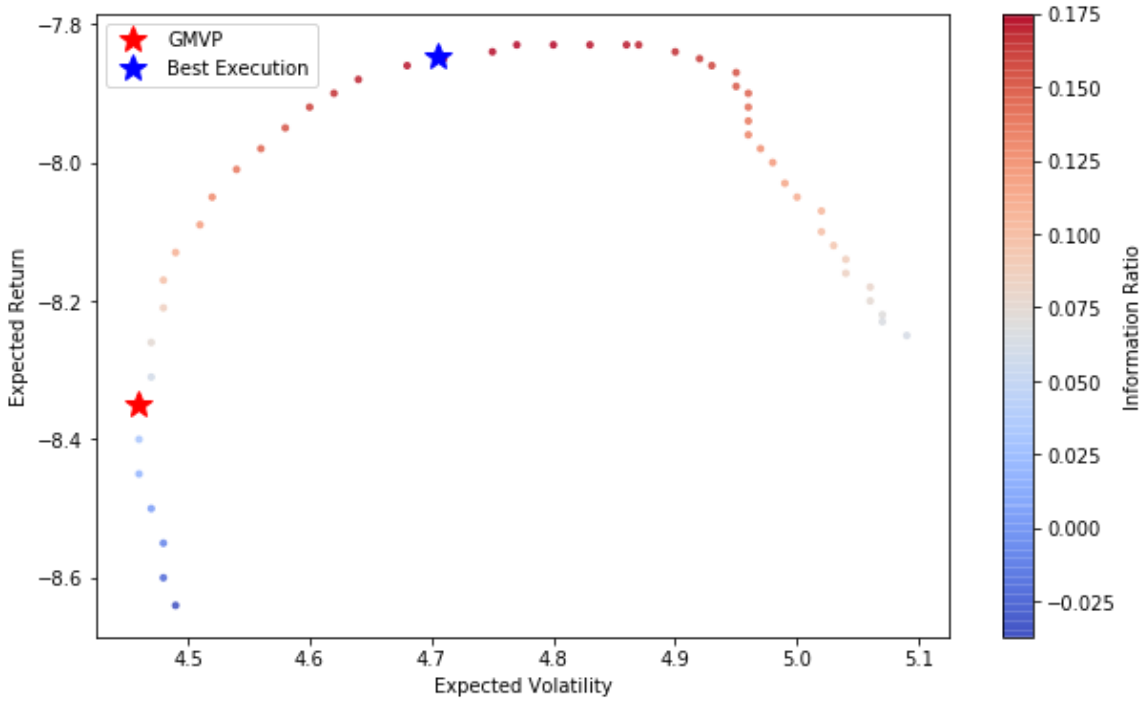
Out[55]:

	PortfolioReturn	StandardDeviation	Max	Min	SharpRatio	InformationRatio
Exec+21%	-7.85	4.71	-0.01	-150.0	-1.67	0.175

In [56]:

```
plt.figure(figsize=(10,6))

# Scatter plot all the possible portfolios
plt.scatter(df_buyer_strategy1_ex_eval["StandardDeviation"], df_buyer_strategy1_ex_eval["PortfolioReturn"], c=df_buyer_strategy1_ex_eval["InformationRatio"], marker='.', alpha=0.8, cmap='coolwarm')

# Global minimum variance portfolio
plt.plot(GMVP["StandardDeviation"], GMVP["PortfolioReturn"], 'r*', markersize=15.0, label="GMVP")
print("[Global Minimum Variance Portfolio]")
i = 6
DA_position_i = df_buyer_strategy1_ex["Position_" + str(i) + "%"].value_counts()[0]
Intra_position_i = df_buyer_strategy1_ex["Position_" + str(i) + "%"].value_counts()[1]
print("Position: DA {}".format(round(DA_position_i / (DA_position_i+Intra_position_i)*100, 2)) + "%")
print("Expected return: {}".format(GMVP["PortfolioReturn"][0]))
print("Standard deviation: {}".format(GMVP["StandardDeviation"][0]))
print("Sharp ratio: {}".format(GMVP["SharpRatio"][0]))
print("Information ratio: {}".format(GMVP["InformationRatio"][0]))

print("[Best Execution Portfolio]")
i = -Best_i
DA_position_i = df_buyer_strategy1_ex["Position_" + str(i) + "%"].value_counts()[0]
Intra_position_i = df_buyer_strategy1_ex["Position_" + str(i) + "%"].value_counts()[1]
plt.plot(df_buyer_strategy1_ex["Exec_" + str(i) + "%"].std(), df_buyer_strategy1_ex["Exec_" + str(i) + "%"].mean(), "b*", markersize=15.0, label="Best Execution")
print("Position: DA {}".format(round(DA_position_i / (DA_position_i+Intra_position_i)*100, 2)) + "%")
print("Expected return: {}".format(df_buyer_strategy1_ex["Exec_" + str(i) + "%"].mean().round(2)))
print("Standard deviation: {}".format(df_buyer_strategy1_ex["Exec_" + str(i) + "%"].std().round(2)))
print("Sharp ratio: {}".format(round(df_buyer_strategy1_ex["Exec_" + str(i) + "%"].mean() / df_buyer_strategy1_ex["Exec_" + str(i) + "%"].std(),2)))
print("Information ratio: {}".format(round((df_buyer_strategy1_ex["Exec_" + str(i) + "%"] - Benchmark_buyer).mean() / (df_buyer_strategy1_ex["Exec_" + str(i) + "%"] - Benchmark_buyer).std(), 2)))

plt.xlabel('Expected Volatility')
plt.ylabel('Expected Return')
plt.colorbar(label='Information Ratio')
plt.legend(loc="upper left");
```

[Global Minimum Variance Portfolio]
Position: DA 76.35%
Expected return: -8.35
Standard deviation: 4.46
Sharp ratio: -1.87
Information ratio: 0.047
[Best Execution Portfolio]
Position: DA 94.25%
Expected return: -7.85
Standard deviation: 4.71
Sharp ratio: -1.67
Information ratio: 0.17



Strategy 2 (Trading with prediction after DA)

Required dataset

In [57]:

```
# Need predicted price after DA market and spot price
df_buyer_strategy2 = df_prediction_afterDA.copy()

# Make lists for price information
Spot_list = list(df_buyer_strategy2["Spot"])
Low_list = list(df_buyer_strategy2["Low"])
Close_list = list(df_buyer_strategy2["Close"])
Pred_list_i = list(df_buyer_strategy2["Close_pred"])
# Judge_success = []

# list for executed orders
Executed_i = []
# list for checking the execution results
Judge_i = []

# Calculate executed price
for l, c, p in zip(Low_list, Close_list, Pred_list_i):
    # Trade on Intra markets
    if p >= l:
        Executed_i.append(p)
        Judge_i.append("True")
    else:
        Executed_i.append(c)
        Judge_i.append("False")

df_buyer_strategy2["ExecutedOrder"] = pd.Series(Executed_i)
df_buyer_strategy2["Judge"] = pd.Series(Judge_i)
```

In [58]:

```
df_buyer_strategy2.head()
```

Out[58]:

	DateTime	Spot	High	Low	Close	Close_pred	ExecutedOrder	Judge
0	2017-08-10 00:00:00	7.98	12.98	5.94	9.4	7.89	7.89	True
1	2017-08-10 00:30:00	7.67	12.67	5.63	9.4	7.72	7.72	True
2	2017-08-10 01:00:00	6.86	11.86	5.58	9.4	7.26	7.26	True
3	2017-08-10 01:30:00	6.58	11.58	4.91	9.4	6.79	6.79	True
4	2017-08-10 02:00:00	6.62	11.62	4.88	9.4	6.82	6.82	True

Evaluation

In [59]:

```
portfolio_weights= [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

cols = []

PortfolioReturn = []
StandardDeviation = []
Max = []
Min = []
SharpRatio = []
InformationRatio = []

Spot_list = list(df_buyer_strategy2["Spot"])
Exec_list = list(df_buyer_strategy2["ExecutedOrder"])

for weight in portfolio_weights:
    Return = []
    for spot, intra in zip(Spot_list, Exec_list):
        # DA market 100%
        if weight == 1.0:
            Return.append(-spot*weight)
        # Trade both on DA and Intra day
        else:
            Return.append(-spot*weight + -intra*(1-weight))
    # Set colmun name
    cols.append("DA" + str(weight*100) + "%")

    # Results
    Return = pd.Series(Return)
    PortfolioReturn.append(Return.mean().round(2))
    StandardDeviation.append(Return.std().round(2))
    Max.append(Return.max().round(2))
    Min.append(Return.min().round(2))
    SharpRatio.append((Return.mean()/Return.std()).round(2))
    InformationRatio.append(round((Return - Benchmark_buyer).mean() / (Return - Benchmark_buyer
).std(), 3))

# Transpose columns and rows
df_buyer_strategy2_eval = pd.DataFrame()
df_buyer_strategy2_eval = df_buyer_strategy2_eval.T

# Add columns for evaluation metrics
df_buyer_strategy2_eval["PortfolioReturn"] = pd.Series(PortfolioReturn)
df_buyer_strategy2_eval["StandardDeviation"] = pd.Series(StandardDeviation)
df_buyer_strategy2_eval["Max"] = pd.Series(Max)
df_buyer_strategy2_eval["Min"] = pd.Series(Min)
df_buyer_strategy2_eval["SharpRatio"] = pd.Series(SharpRatio)
df_buyer_strategy2_eval["InformationRatio"] = pd.Series(InformationRatio)

df_buyer_strategy2_eval.index = cols
```

In [60]:

```
df_buyer_strategy2_eval
```

Out[60]:

	PortfolioReturn	StandardDeviation	Max	Min	SharpRatio	InformationRatio
DA0%	-8.40	4.15	-0.16	-150.00	-2.03	0.083
DA10.0%	-8.42	4.16	-0.15	-139.70	-2.02	0.083
DA20.0%	-8.43	4.18	-0.13	-129.41	-2.02	0.083
DA30.0%	-8.44	4.20	-0.12	-119.11	-2.01	0.083
DA40.0%	-8.46	4.23	-0.10	-108.82	-2.00	0.083
DA50.0%	-8.47	4.26	-0.08	-98.52	-1.99	0.083
DA60.0%	-8.48	4.30	-0.07	-88.22	-1.97	0.083
DA70.0%	-8.49	4.35	-0.06	-78.95	-1.95	0.083
DA80.0%	-8.51	4.40	-0.04	-79.30	-1.93	0.083
DA90.0%	-8.52	4.45	-0.02	-79.65	-1.91	0.083
DA100.0%	-8.53	4.51	-0.01	-80.00	-1.89	NaN

In [61]:

```
# Pick up the portfolios on the global minimum variance portfolio
std_min = df_buyer_strategy2_eval[df_buyer_strategy2_eval["StandardDeviation"] == df_buyer_
strategy2_eval["StandardDeviation"].min()]
GMVP = std_min[std_min["PortfolioReturn"] == std_min["PortfolioReturn"].min()]
GMVP
```

Out[61]:

	PortfolioReturn	StandardDeviation	Max	Min	SharpRatio	InformationRatio
DA0%	-8.4	4.15	-0.16	-150.0	-2.03	0.083

In [62]:

```
# The Best Execusion
Best = df_buyer_strategy2_eval[df_buyer_strategy2_eval["InformationRatio"] == df_buyer_strate
gy2_eval["InformationRatio"].max()]
Best = Best[Best["StandardDeviation"] == Best["StandardDeviation"].min()]
Best
```

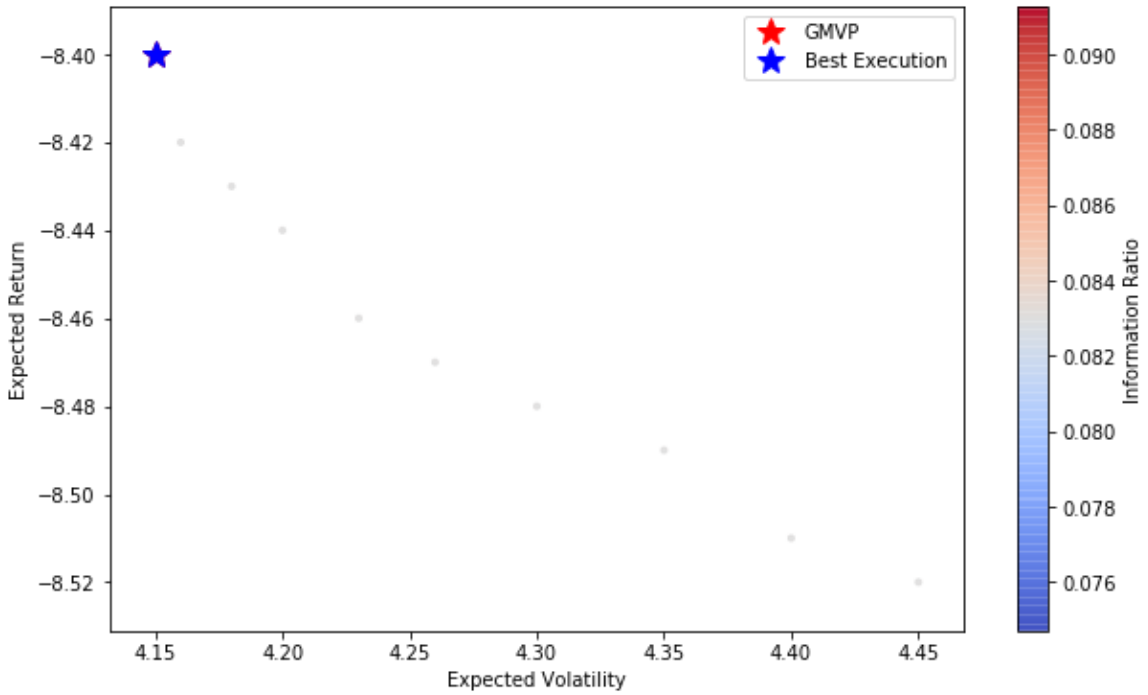
Out[62]:

	PortfolioReturn	StandardDeviation	Max	Min	SharpRatio	InformationRatio
DA0%	-8.4	4.15	-0.16	-150.0	-2.03	0.083

In [63]:

```
plt.figure(figsize=(10,6))

# Scatter plot all the possible portfolios
plt.scatter(df_buyer_strategy2_eval["StandardDeviation"], df_buyer_strategy2_eval["PortfolioReturn"], c=df_buyer_strategy2_eval["InformationRatio"], marker='.', alpha=0.8, cmap='coolwarm')

# Global minimum variance portfolio
plt.plot(GMVP["StandardDeviation"], GMVP["PortfolioReturn"], 'r*', markersize=15.0, label="GMVP")
print("[Global Minimum Variance Portfolio]")
print("Expected return: {}".format(GMVP["PortfolioReturn"][0]))
print("Standard deviation: {}".format(GMVP["StandardDeviation"][0]))
print("Sharp ratio: {}".format(GMVP["SharpRatio"][0]))
print("Information ratio: {}".format(GMVP["InformationRatio"][0]))

# Best executed portfolio
plt.plot(Best["StandardDeviation"], Best["PortfolioReturn"], "b*", markersize=15.0, label="Best Execution")
print("[Best Execution Portfolio]")
print("Expected return: {}".format(Best["PortfolioReturn"][0]))
print("Standard deviation: {}".format(Best["StandardDeviation"][0]))
print("Sharp ratio: {}".format(Best["SharpRatio"][0]))
print("Information ratio: {}".format(Best["InformationRatio"][0]))

plt.xlabel('Expected Volatility')
plt.ylabel('Expected Return')
plt.colorbar(label='Information Ratio')
plt.legend(loc="upper right");
```

[Global Minimum Variance Portfolio]
Expected return: -8.4
Standard deviation: 4.15
Sharp ratio: -2.03
Information ratio: 0.083
[Best Execution Portfolio]
Expected return: -8.4
Standard deviation: 4.15
Sharp ratio: -2.03
Information ratio: 0.083



Strategy 2 with execution strategy

Required dataset

In [64]:

```
# Need predicted price after DA market and spot price
df_buyer_strategy2_ex = df_prediction_beforeDA.copy()

# Make lists for price information
Spot_list = list(df_buyer_strategy2_ex["Spot"])
Low_list = list(df_buyer_strategy2_ex["Low"])
Close_list = list(df_buyer_strategy2_ex["Close"])
# Judge_success = []

# Order and execution with execusion range
for i in list(range(0, 51, 1)):
    # list for executed orders
    Executed_i = []
    # list for checking the execution results
    Judge_i = []
    Position_i = []
    # Derive the predicted price with the range
    Pred_list_i = list((df_buyer_strategy2_ex["Close_pred"]* (1 - i/100)).round(2))
    for l, c, p in zip(Low_list, Close_list, Pred_list_i):
        # Trade on Intra markets
        if p >= l:
            Executed_i.append(p)
            Judge_i.append("True")
            Position_i.append("DA")
        else:
            Executed_i.append(c)
            Judge_i.append("False")
            Position_i.append("Intra")

    df_buyer_strategy2_ex["Exec_" + str(i) + "%"] = pd.Series(Executed_i)
    df_buyer_strategy2_ex["Judge_" + str(i) + "%"] = pd.Series(Judge_i)
    df_buyer_strategy2_ex["Position_" + str(i) + "%"] = pd.Series(Position_i)

df_buyer_strategy2_ex.head()
```

Out[64]:

	Date	HH	Open	High	Low	Close	Spot	Spot_1daylag	DateTime	Close_pred	Exec_0%
0	2017-08-10	1	10.70	12.98	5.94	9.4	8.19	8.10	2017-08-10 00:00:00	7.83	7.8
1	2017-08-10	2	6.89	12.67	5.63	9.4	8.04	8.00	2017-08-10 00:30:00	7.58	7.5
2	2017-08-10	3	6.42	11.86	5.58	9.4	7.93	8.05	2017-08-10 01:00:00	7.55	7.5
3	2017-08-10	4	10.30	11.58	4.91	9.4	7.73	7.80	2017-08-10 01:30:00	7.27	7.2
4	2017-08-10	5	10.50	11.62	4.88	9.4	7.93	7.94	2017-08-10 02:00:00	7.35	7.3

Evaluation

- Look for the best execution

In [65]:

```
n_splits=100

train_size = df_buyer_strategy2_ex.index[-1]
train_index_list = list(np.linspace(train_size/n_splits, train_size, n_splits, endpoint = True, dtype='in
t'))
train_index_list

# # Confirming the split logic
# for train_index in train_index_list:
#     # Divide the train/valid set  into 10 folds and pick up it.
#     X_train = df_buyer_strategy2_ex.iloc[:train_index]
#     print("TRAIN:", train_index)
```

Out[65]:

```
[595,
 1190,
 1785,
 2380,
 2975,
 3571,
 4166,
 4761,
 5356,
 5951,
 6547,
 7142,
 7737,
 8332,
 8927,
 9523,
 10118,
 10713,
 11308,
 11903,
 12498,
 13094,
 13689,
 14284,
 14879,
 15474,
 16070,
 16665,
 17260,
 17855,
 18450,
 19046,
 19641,
 20236,
 20831,
 21426,
 22022,
 22617,
 23212,
 23807,
 24402,
 24997,
 25593,
 26188,
 26783,
 27378,
 27973,
 28569,
 29164,
 29759,
 30354,
 30949,
 31545,
 32140,
 32735,
 33330,
 33925,
 34521,
 35116,
```

35711,
36306,
36901,
37496,
38092,
38687,
39282,
39877,
40472,
41068,
41663,
42258,
42853,
43448,
44044,
44639,
45234,
45829,
46424,
47020,
47615,
48210,
48805,
49400,
49995,
50591,
51186,
51781,
52376,
52971,
53567,
54162,
54757,
55352,
55947,
56543,
57138,
57733,
58328,
58923,
59519]

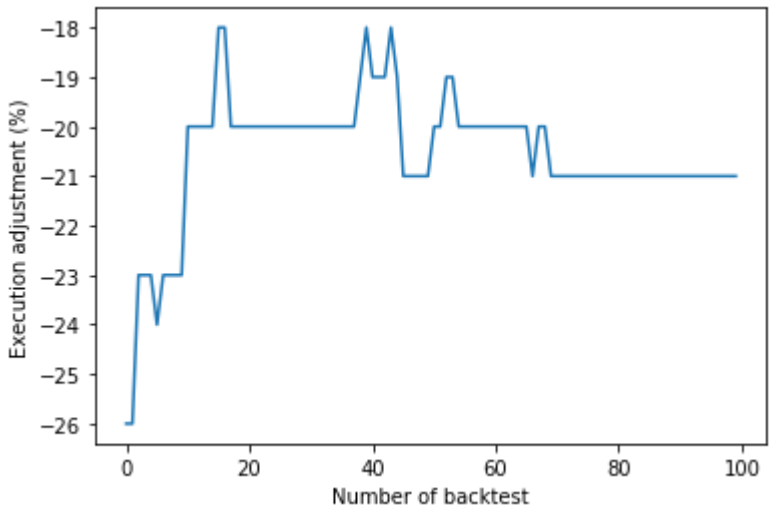
In [66]:

```
BestExec = []

for train_index in train_index_list:
    X_train = df_buyer_strategy2_ex[:train_index]

    Spot_list = list(X_train["Spot"])

    portfolio_weights= [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
    execution_buffers = list(range(0, 51, 1))

    cols = []

    PortfolioReturn = []
    StandardDeviation = []
    Max = []
    Min = []
    SharpRatio = []
    InformationRatio = []

    for i in execution_buffers:
        Exec_list = list(X_train["Exec_" + str(i) + "%"])
        for weight in portfolio_weights:
            Return = []
            for spot, intra in zip(Spot_list, Exec_list):
                # DA market 100%
                if weight == 1.0:
                    Return.append(-spot*weight)
                # Trade both on DA and Intra day
                else:
                    Return.append(-spot*weight + -intra*(1-weight))
            # Set colmun name for execution buffer
            if weight == 1.0:
                cols.append(str(0))
            else:
                cols.append(str(i))

            # Results
            Return = pd.Series(Return)
            PortfolioReturn.append(Return.mean().round(2))
            StandardDeviation.append(Return.std().round(2))
            Max.append(Return.max().round(2))
            Min.append(Return.min().round(2))
            SharpRatio.append((Return.mean()/Return.std()).round(2))
            InformationRatio.append(round((Return - Benchmark_buyer).mean() / (Return - Benchmark_b
uyer).std(), 3))

    #df_portfolio_benchの列を設定し、行列を入替えて調整する
    df_buyer_strategy2_ex_eval = pd.DataFrame()
    df_buyer_strategy2_ex_eval = df_buyer_strategy2_ex_eval.T

    #各算出結果をdf_portfolio_benchの列へ追加する
    df_buyer_strategy2_ex_eval["PortfolioReturn"] = pd.Series(PortfolioReturn)
    df_buyer_strategy2_ex_eval["StandardDeviation"] = pd.Series(StandardDeviation)
    df_buyer_strategy2_ex_eval["Max"] = pd.Series(Max)
    df_buyer_strategy2_ex_eval["Min"] = pd.Series(Min)
    df_buyer_strategy2_ex_eval["SharpRatio"] = pd.Series(SharpRatio)
    df_buyer_strategy2_ex_eval["InformationRatio"] = pd.Series(InformationRatio)

    df_buyer_strategy2_ex_eval.index = cols
```

```
    df_buyer_strategy2_ex_eval =  df_buyer_strategy2_ex_eval.reset_index()
    df_buyer_strategy2_ex_eval = df_buyer_strategy2_ex_eval.rename(columns={"index": "ExecBu
ffer(%)"})
    df_buyer_strategy2_ex_eval["ExecBuffer(%)"] = df_buyer_strategy2_ex_eval["ExecBuffer(%)"]
.astype(int)

    Exec = df_buyer_strategy2_ex_eval[df_buyer_strategy2_ex_eval["InformationRatio"] == df_bu
yer_strategy2_ex_eval["InformationRatio"].max()]
    Exec = -Exec["ExecBuffer(%)"][Exec["StandardDeviation"] == Exec["StandardDeviation"].min()].il
oc[0]
    BestExec.append(Exec)
```

In [67]:

```
# BestExec = pd.Series(BestExec)
BestExec = pd.Series(BestExec).astype(int)
plt.xlabel('Number of backtest')
plt.ylabel('Execution adjustment (%)')
BestExec.plot();
```



In [68]:

```
Best_i = BestExec.iloc[-1]
Best_i
```

Out[68]:

-21

- Plot all the possible portfolios

In [69]:

```
Spot_list = list(df_buyer_strategy2_ex["Spot"])

portfolio_weights= [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
execution_buffers = list(range(0, 51, 1))

cols = []

PortfolioReturn = []
StandardDeviation = []
Max = []
Min = []
SharpRatio = []
InformationRatio = []

for i in execution_buffers:
    Exec_list = list(df_buyer_strategy2_ex["Exec_" + str(i) + "%"])
    for weight in portfolio_weights:
        Return = []
        for spot, intra in zip(Spot_list, Exec_list):
            # DA market 100%
            if weight == 1.0:
                Return.append(-spot*weight)
            # Trade both on DA and Intra day
            else:
                Return.append(-spot*weight + -intra*(1-weight))
        # Set colmun name for execution buffer
        if weight == 1.0:
            cols.append("DA100%")
        else:
            cols.append("DA" + str(weight*100) + "%_Exec+" + str(i) + "%")

        # Results
        Return = pd.Series(Return)
        PortfolioReturn.append(Return.mean().round(2))
        StandardDeviation.append(Return.std().round(2))
        Max.append(Return.max().round(2))
        Min.append(Return.min().round(2))
        SharpRatio.append((Return.mean()/Return.std()).round(2))
        InformationRatio.append(((Return - Benchmark_buyer).mean()/(Return - Benchmark_buyer).std
()).round(3))

#df_portfolio_benchの列を設定し、行列を入替えて調整する
df_buyer_strategy2_ex_eval = pd.DataFrame()
df_buyer_strategy2_ex_eval = df_buyer_strategy2_ex_eval.T

#各算出結果をdf_portfolio_benchの列へ追加する
df_buyer_strategy2_ex_eval["PortfolioReturn"] = pd.Series(PortfolioReturn)
df_buyer_strategy2_ex_eval["StandardDeviation"] = pd.Series(StandardDeviation)
df_buyer_strategy2_ex_eval["Max"] = pd.Series(Max)
df_buyer_strategy2_ex_eval["Min"] = pd.Series(Min)
df_buyer_strategy2_ex_eval["SharpRatio"] = pd.Series(SharpRatio)
df_buyer_strategy2_ex_eval["InformationRatio"] = pd.Series(InformationRatio)

df_buyer_strategy2_ex_eval.index = cols
```

In [70]:

```
df_buyer_strategy2_ex_eval
```

Out[70]:

	PortfolioReturn	StandardDeviation	Max	Min	SharpRatio	Informatio
DA0%_Exec+0%	-8.71	4.17	-1.26	-150.00	-2.09	
DA10.0%_Exec+0%	-8.77	4.17	-1.17	-140.00	-2.10	
DA20.0%_Exec+0%	-8.83	4.19	-1.04	-130.00	-2.11	
DA30.0%_Exec+0%	-8.88	4.24	-0.91	-120.00	-2.10	
DA40.0%_Exec+0%	-8.94	4.31	-0.78	-110.00	-2.07	
...	...	...	...	...	...	
DA60.0%_Exec+50%	-8.87	4.81	-0.27	-89.99	-1.84	
DA70.0%_Exec+50%	-8.98	4.87	-0.20	-79.99	-1.84	
DA80.0%_Exec+50%	-9.08	4.95	-0.14	-77.08	-1.83	
DA90.0%_Exec+50%	-9.18	5.06	-0.07	-76.09	-1.81	
DA100%	-9.29	5.20	-0.01	-75.10	-1.79	

561 rows × 6 columns

In [71]:

```
# Pick up the portfolios on the global minimum variance portfolio
std_min = df_buyer_strategy2_ex_eval[df_buyer_strategy2_ex_eval["StandardDeviation"] == df_
buyer_strategy2_ex_eval["StandardDeviation"].min()]
GMVP = std_min[std_min["PortfolioReturn"] == std_min["PortfolioReturn"].min()]
GMVP
```

Out[71]:

	PortfolioReturn	StandardDeviation	Max	Min	SharpRatio	InformationF
DA10.0%_Exec+0%	-8.77	4.17	-1.17	-140.0	-2.1	-0

In [72]:

```
Best = df_buyer_strategy2_ex_eval[df_buyer_strategy2_ex_eval["InformationRatio"] == df_buyer
_strategy2_ex_eval["InformationRatio"].max()]
Best = Best[Best["StandardDeviation"] == Best["StandardDeviation"].min()]
Best
```

Out[72]:

	PortfolioReturn	StandardDeviation	Max	Min	SharpRatio	InformationRat
DA0%_Exec+21%	-7.82	4.56	-1.0	-150.0	-1.72	0.2

In [73]:

```
plt.figure(figsize=(10,6))

# Scatter plot all the possible portfolios
plt.scatter(df_buyer_strategy2_ex_eval["StandardDeviation"], df_buyer_strategy2_ex_eval["PortfolioReturn"], c=df_buyer_strategy2_ex_eval["InformationRatio"], marker='.', alpha=0.8, cmap='coolwarm')


# Global minimum variance portfolio
plt.plot(GMVP["StandardDeviation"], GMVP["PortfolioReturn"], 'r*', markersize=15.0, label="GMVP")
print("[Global Minimum Variance Portfolio: " + str(GMVP.index[0]) + "]")

print("Expected return: {}".format(GMVP["PortfolioReturn"][0]))
print("Standard deviation: {}".format(GMVP["StandardDeviation"][0]))
print("Information Ratio: {}".format(GMVP["InformationRatio"][0]))

# The Best Execusion
DA_weight = 0
i = -Best_i
Best_Portfolio = (DA_weight * -df_buyer_strategy2_ex["Spot"]) + ((1 - DA_weight) * -df_buyer_strategy2_ex["Exec_" + str(i) + "%"])
plt.plot(Best_Portfolio.std(), Best_Portfolio.mean(), "b*", markersize=15.0, label="Best Execution")
print("[Best Execution Portfolio: " + str(DA_weight) + "%_Exec+" + str(i) + "%]")
print("Expected return: {}".format(Best_Portfolio.mean().round(2)))
print("Standard deviation: {}".format(Best_Portfolio.std().round(2)))
print("Sharp ratio: {}".format(round(Best_Portfolio.mean() / Best_Portfolio.std(), 2)))
print("Information ratio: {}".format(round((Best_Portfolio - Benchmark_buyer).mean() / (Best_Portfolio - Benchmark_buyer).std(), 3)))

plt.title('Strategic portfolios with excution adjustment for buyers')
plt.xlabel('Expected Volatility')
plt.ylabel('Expected Return')
plt.colorbar(label='Information Ratio')
plt.legend(loc="upper left");
```

[Global Minimum Variance Portfolio: DA10.0%_Exec+0%]
Expected return: -8.77
Standard deviation: 4.17
Information Ratio: -0.108
[Best Execution Portfolio: 0%_Exec+21%]
Expected return: -7.82
Standard deviation: 4.56
Sharp ratio: -1.72
Information ratio: 0.265

In [74]:

```
Best_Portfolio
```

Out[74]:

```
0      -6.19
1      -5.99
2      -5.96
3      -5.74
4      -5.81
       ...
59515  -70.00
59516  -45.48
59517  -41.33
59518  -36.66
59519  -26.46
Length: 59520, dtype: float64
```

For Trader (Virtual bidding strategy)

[Purpose]

- To confirm the market efficiency of the JEPX
- To examine virtual bidding as a potential solution for improving liquidity of the DA and the intraday market on the JEPX

[Basic logic for virtual bidding]

1) Make long or short positions on the DA market

2) Settle all the positions by counter trading on the intraday market on the day

- Spot > Close: Entry with sell-in on the DA and buy-out on the intraday <p>
- Spot < Close: Entry with buy-in on the DA and sell-out on the intraday

Trading without execution strategy

In [75]:

```
df_trader = df_prediction_beforeDA.copy()
df_trader["Date"] = pd.to_datetime(df_trader["Date"])
df_trader["DateTime"] = pd.to_datetime(df_trader["DateTime"])
df_trader.head()
```

Out[75]:

	Date	HH	Open	High	Low	Close	Spot	Spot_1daylag	DateTime	Close_pred
0	2017-08-10	1	10.70	12.98	5.94	9.4	8.19	8.10	2017-08-10 00:00:00	7.83
1	2017-08-10	2	6.89	12.67	5.63	9.4	8.04	8.00	2017-08-10 00:30:00	7.58
2	2017-08-10	3	6.42	11.86	5.58	9.4	7.93	8.05	2017-08-10 01:00:00	7.55
3	2017-08-10	4	10.30	11.58	4.91	9.4	7.73	7.80	2017-08-10 01:30:00	7.27
4	2017-08-10	5	10.50	11.62	4.88	9.4	7.93	7.94	2017-08-10 02:00:00	7.35

Evaluation

In [76]:

```
# For execution
LaggedSpot_list = list(df_trader["Spot_1daylag"])
Pred_list = list(df_trader["Close_pred"])
Close_list = list(df_trader["Close"])
High_list = list(df_trader["High"])
Low_list = list(df_trader["Low"])

# For evaluation
Spot_list = list(df_trader["Spot"])

# Lists for results
cols = []
Return = []
PortfolioReturn = []
StandardDeviation = []
Max = []
Min = []
SharpRatio = []

# Main logic
for spot, lag_spot, pred, high, low, close in zip(Spot_list,LaggedSpot_list, Pred_list, High_list, Low_li
st, Close_list):
    # Buy-in, Sell-out ※ Decision based on lag_spot and pred
    if lag_spot < pred:
        #Return ※ Evaluation based on spot and pred/close
        # Succeed
        if pred < high:
            Return.append(round(pred - spot, 4))
        # Fail
        else:
            Return.append(round(close - spot, 4))

    # Sell-in, Buy-out ※ Decision based on lag_spot and pred
    elif lag_spot > pred:
        #Return ※ Evaluation based on spot and pred/close
        # Succeed
        if pred > low:
            Return.append(round(spot - pred, 4))
        # Fail
        else:
            Return.append(round(spot - close, 4))
    # lag_spot is the same as pred --> No trade
    else:
        Return.append(0)

df_trader_eval = pd.DataFrame()
df_trader_eval["DateTime"] = df_trader["DateTime"]
df_trader_eval["HH"] = df_trader["HH"]
df_trader_eval["Date"] = df_trader["Date"]
df_trader_eval["Return"] = pd.Series(Return)
```

In [77]:

```
df_trader_eval.head()
```

Out[77]:

	DateTime	HH	Date	Return
0	2017-08-10 00:00:00	1	2017-08-10	0.36
1	2017-08-10 00:30:00	2	2017-08-10	0.46
2	2017-08-10 01:00:00	3	2017-08-10	0.38
3	2017-08-10 01:30:00	4	2017-08-10	0.46
4	2017-08-10 02:00:00	5	2017-08-10	0.58

In [78]:

```
# Make a pivot table for results
df_trader_portfolio_table = pd.DataFrame(df_trader_eval.pivot(index='Date', columns='HH', values
='Return'))
df_trader_portfolio_table["Expected_Return"] = df_trader_portfolio_table.loc[: , 0:48].mean(axis=
1).round(2)
df_trader_portfolio_table["SharpRatio"] = (df_trader_portfolio_table.loc[: , 1:48].mean(axis=1)/df
_trader_portfolio_table.loc[: , 1:48].std(axis=1)).round(2)

# Make a new table for performance graph
risk = pd.DataFrame()
risk["Expected_Return"] = df_trader_portfolio_table.loc[: , 1:48].mean(axis=1).round(2)
risk["equity"] = risk['Expected_Return'].cumsum()
risk["cummax"] = risk["equity"].cummax()
risk["drawdown"] = -(risk["cummax"] -risk["equity"])

risk.head()
```

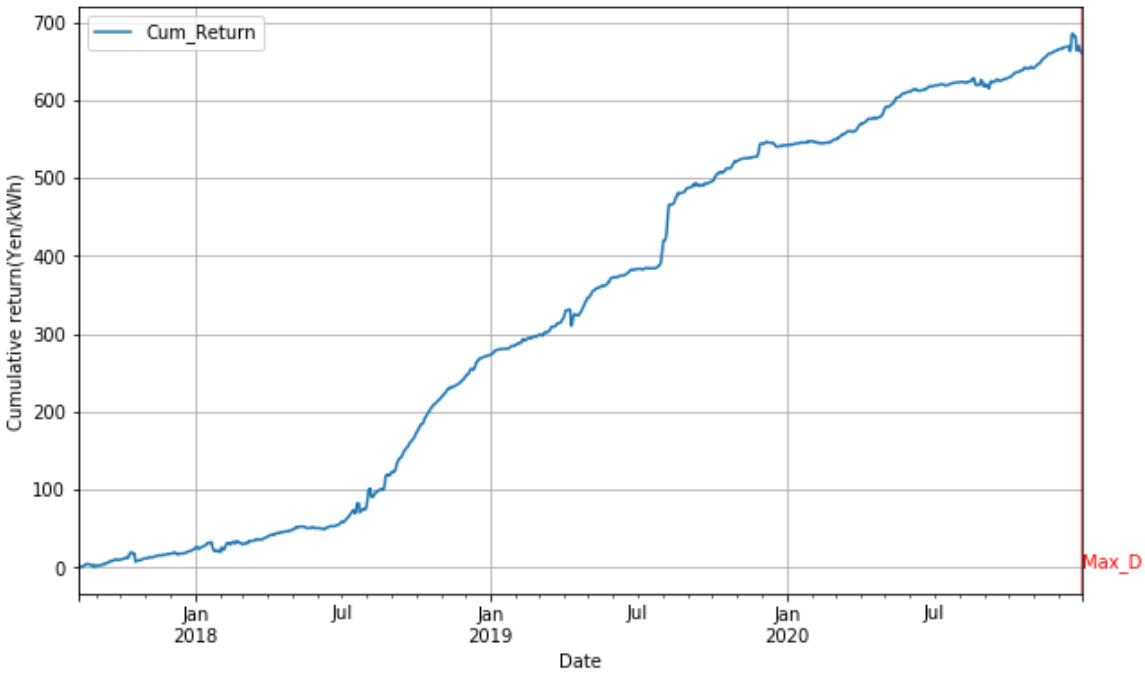
Out[78]:

Date	Expected_Return	equity	cummax	drawdown
2017-08-10	3.00	3.00	3.0	-0.00
2017-08-11	-2.65	0.35	3.0	-2.65
2017-08-12	-0.25	0.10	3.0	-2.90
2017-08-13	1.14	1.24	3.0	-1.76
2017-08-14	0.35	1.59	3.0	-1.41

In [79]:

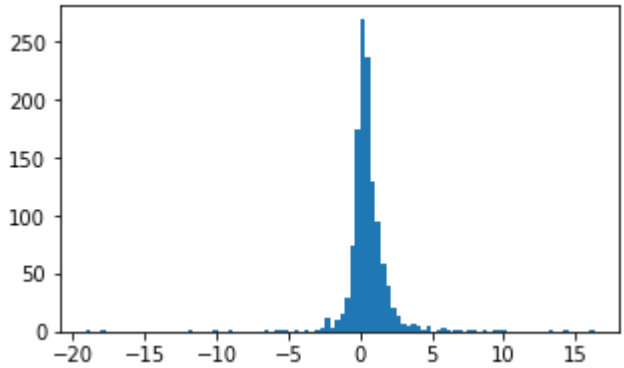
```
# Set max_drawdown
max_drawdown = risk["drawdown"].min()

# Set timestamp of max_drawdown
t_max = pd.to_datetime(risk["drawdown"].idxmin())

# Plot the performance
fig, ax = plt.subplots(1, figsize=(15,8))
# plt.title('Cumulative return through trading based on the difference of price on DA and Intraday')

risk["equity"].plot(figsize=(10, 6), label="Cum_Return")

plt.text(t_max, 0, 'Max_D', rotation=0, color="r")

print("Expected_Return: " + str(risk["Expected_Return"].mean().round(2)))
print("StandardDeviation: " + str(risk["Expected_Return"].std().round(2)))
print("SharpRatio: " + str(round(risk["Expected_Return"].mean() / risk["Expected_Return"].std(),2
)))
print("Cum_Return: " + str(risk["equity"][-1:][0].round(2)))
print("Drawdown: " + str(risk["drawdown"].min().round(2)))

ax.set(xlabel="Date", ylabel="Cumulative return(Yen/kWh)")
plt.axvline(t_max, c="r", alpha=0.5)
plt.grid()
plt.legend();
```

Expected_Return: 0.53
StandardDeviation: 1.76
SharpRatio: 0.3
Cum_Return: 659.57
Drawdown: -25.9



In [80]:

```
# Distribution of expected return
plt.figure(figsize=(5, 3))

plt.hist(risk['Expected_Return'], bins=100);
```



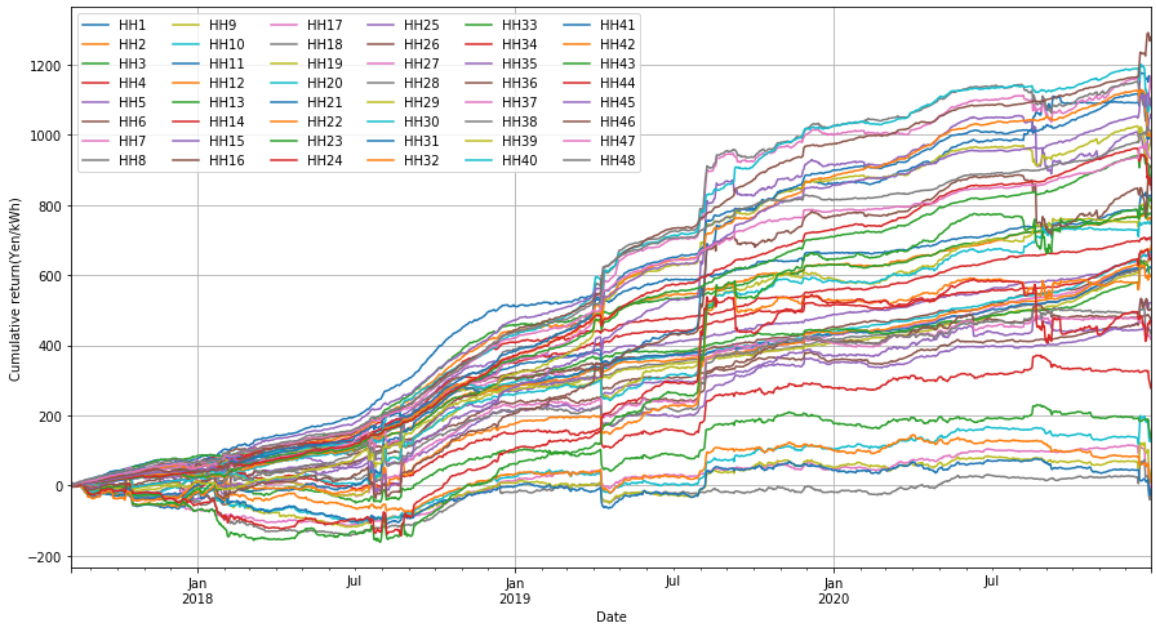
In [81]:

```
# Calculate VaR
import scipy.stats as scs

equity = 10000000

percs = np.array([1., 5.0, 10.0])
risk["returns"] = np.log(risk["equity"] / risk["equity"].shift(1))
VaR = scs.scoreatpercentile(equity * risk["returns"], percs)
def print_var():
    print('%16s %16s' % ('Confidence Level', 'Value-at-Risk'))
    print(43 * '-')
    for pair in zip(percs, VaR):
        print('%16.0f %16.0f' % (100 - pair[0], -pair[1]))

print_var()
```

```
Confidence Level    Value-at-Risk
-------------------------------------------
              99          1209799
              95           146771
              90            29673
```

In [82]:

```
# Plot each 48 item separately
fig, ax = plt.subplots(1, figsize=(15,8))
# plt.title('Cumulative return through trading based on the difference of price on DA and Intraday')

# Lists for results
Equity = []
ExpectedReturn = []
Max = []
Min = []
StandardDeviation = []
SharpRatio= []
CumMax = []
MaxDrawdown = []

for HH in list(range(1, 49, 1)):
    # makng a pivot table
    df_trader_portfolio_table = pd.DataFrame(df_trader_eval.pivot(index='Date', columns='HH', valu
es='Return'))

    # For plot and for calculating drawdown
    equity = df_trader_portfolio_table[HH].cumsum().round(2)
    # Plot the performance of each HH spot
    equity.plot(label="HH" + str(HH))

    # Keep in a list for calculating drawdown
    Equity.append(equity)
    cummax = equity.cummax()
    drawdown = -(cummax - equity)

    # Calculation for each evaluation on each HH slot
    HH_Return = df_trader_portfolio_table[HH].mean().round(2)
    HH_Std = df_trader_portfolio_table[HH].std().round(2)
    ExpectedReturn.append(HH_Return)
    StandardDeviation.append(HH_Std)
    Max.append(df_trader_portfolio_table[HH].max().round(2))
    Min.append(df_trader_portfolio_table[HH].min().round(2))
    SharpRatio.append((HH_Return / HH_Std).round(2))
    CumMax.append(cummax.max().round(2))
    MaxDrawdown.append(drawdown.min().round(2))

ax.set(xlabel="Date", ylabel="Cumulative return(Yen/kWh)")
plt.grid()
plt.legend(loc="upper left", ncol=6);
```



In [83]:

```
# Make a table for evaluation metrics for all items
df_trader_HH_Eval = pd.DataFrame()
df_trader_HH_Eval["HH"] =  pd.Series(list(range(1,49,1)))
df_trader_HH_Eval["ExpectedReturn"]  = pd.Series(ExpectedReturn)
df_trader_HH_Eval["Max"]  = pd.Series(Max)
df_trader_HH_Eval["Min"]  = pd.Series(Min)
df_trader_HH_Eval["StandardDeviation"]  = pd.Series(StandardDeviation)
df_trader_HH_Eval["SharpRatio"]  = pd.Series(SharpRatio)
df_trader_HH_Eval["CumMax"] = pd.Series(CumMax)
df_trader_HH_Eval["MaxDrawdown"]  = pd.Series(MaxDrawdown)
```

In [84]:

```
df_trader_HH_Eval[df_trader_HH_Eval["CumMax"] == df_trader_HH_Eval["CumMax"].max()]
```

Out[84]:

	HH	ExpectedReturn	Max	Min	StandardDeviation	SharpRatio	CumMax	MaxDrawdow
45	46	1.03	51.45	-16.8	2.94	0.35	1290.85	-23.4

HH46 --> 22:30

In [85]:

```
df_trader_HH_Eval[df_trader_HH_Eval["SharpRatio"] == df_trader_HH_Eval["SharpRatio"].max()]
```

Out[85]:

	HH	ExpectedReturn	Max	Min	StandardDeviation	SharpRatio	CumMax	MaxDrawdown
2	3	0.62	7.47	-6.64	1.25	0.5	778.01	-10.15

HH3 --> 1:00

In [86]:

```
df_trader_HH_Eval[df_trader_HH_Eval["CumMax"] == df_trader_HH_Eval["CumMax"].min()]
```

Out[86]:

	HH	ExpectedReturn	Max	Min	StandardDeviation	SharpRatio	CumMax	MaxDrawdo
17	18	-0.01	32.14	-58.04	3.49	-0.0	73.75	-146.

HH18 --> 8:30

In [87]:

```
df_trader_HH_Eval[df_trader_HH_Eval["MaxDrawdown"] == df_trader_HH_Eval["MaxDrawdown"].
max()]
```

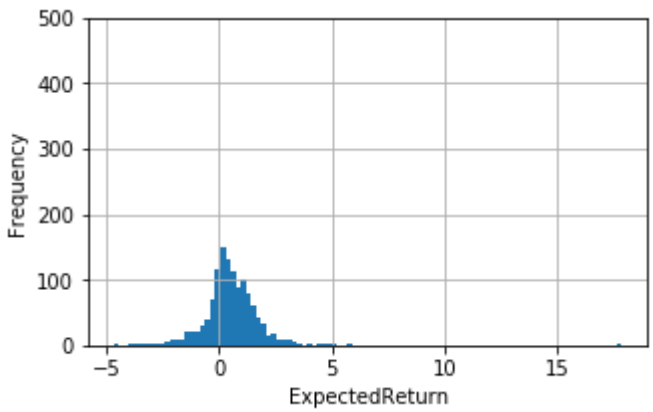
Out[87]:

	HH	ExpectedReturn	Max	Min	StandardDeviation	SharpRatio	CumMax	MaxDrawdown
5	6	0.53	17.83	-4.67	1.21	0.44	651.81	-8.87

HH6 --> 2:30

In [88]:

```
# Distribution of expected return
fig, ax = plt.subplots(1, figsize=(5,3))
plt.hist(df_trader_portfolio_table[6], bins=100)
plt.ylim(0, 500)
ax.set(xlabel="ExpectedReturn", ylabel="Frequency")
plt.grid();
```



In [89]:

```
df_trader_HH_Eval[df_trader_HH_Eval["MaxDrawdown"] == df_trader_HH_Eval["MaxDrawdown"].
min()]
```

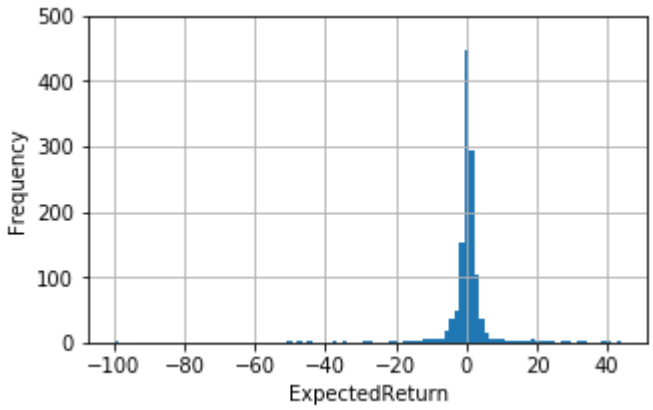
Out[89]:

	HH	ExpectedReturn	Max	Min	StandardDeviation	SharpRatio	CumMax	MaxDrawd
33	34	0.37	44.03	-100.01	6.57	0.06	589.05	-18

HH34 --> 16:30

In [90]:

```
# Distribution of expected return
fig, ax = plt.subplots(1, figsize=(5,3))
df_trader_portfolio_table[34].hist(bins=100)
plt.ylim(0, 500)
ax.set(xlabel="ExpectedReturn", ylabel="Frequency");
```



Trading with execution strategy

Required dataset

In [91]:

```
Spot_list = list(df_trader["Spot"])
LaggedSpot_list = list(df_trader["Spot_1daylag"])
Close_list = list(df_trader["Close"])
High_list = list(df_trader["High"])
Low_list = list(df_trader["Low"])

cols = []

df_trader_ex = pd.DataFrame()
df_trader_ex["DateTime"] = df_trader["DateTime"]
df_trader_ex["HH"] = df_trader["HH"]
df_trader_ex["Date"] = df_trader["Date"]

# Just for check
df_trader_ex["Spot"] = df_trader["Spot"]
df_trader_ex["Close_pred"] = df_trader["Close_pred"]
df_trader_ex["Close"] = df_trader["Close"]

for i in list(range(-30, 31, 1)):
    Return = []
    Judge = []
    Pred_list = list(round(df_trader_ex["Close_pred"]  * (1+ i/100),2))
    for spot, lag_spot, pred_ex, high, low, close in zip(Spot_list, LaggedSpot_list, Pred_list, High_list,
Low_list, Close_list):
        # Buy-in, Sell-out ※Decision based on lag_spot and pred
        if lag_spot < pred_ex:
            #※ Evaluation based on spot and pred/close
            # Success of close the position
            if pred_ex <= high:
                Return.append(pred_ex - spot)
                Judge.append("True")
            # Failure of close the position
            else:
                Return.append(close - spot)
                Judge.append("False")

        # Sell-in, Buy-out ※ Decision based on lag_spot and pred
        elif lag_spot > pred_ex:
            #※ Evaluation based on spot and pred/close
            # Success of close the position
            if pred_ex >= low:
                Return.append(spot - pred_ex)
                Judge.append("True")
            # Failure of close the position
            else:
                Return.append(spot - close)
                Judge.append("False")
        #No trade
        else:
            Return.append(0)
            Judge.append("None")

    df_trader_ex["Return_exec" + str(i) + "%"] = pd.Series(Return)
    df_trader_ex["Judge_exec" + str(i) + "%"] = pd.Series(Judge)
```

In [92]:

```
df_trader_ex
```

Out[92]:

	DateTime	HH	Date	Spot	Close_pred	Close	Return_exec-30%	Judge_exec-30%	Return_e
0	2017-08-10 00:00:00	1	2017-08-10	8.19	7.83	9.40	-1.21	False	-
1	2017-08-10 00:30:00	2	2017-08-10	8.04	7.58	9.40	-1.36	False	-
2	2017-08-10 01:00:00	3	2017-08-10	7.93	7.55	9.40	-1.47	False	-
3	2017-08-10 01:30:00	4	2017-08-10	7.73	7.27	9.40	2.64	True	
4	2017-08-10 02:00:00	5	2017-08-10	7.93	7.35	9.40	2.79	True	
...	...	...	...	...	...	...	...	...	
59515	2020-12-31 21:30:00	44	2020-12-31	35.00	18.81	70.00	-35.00	False	-3
59516	2020-12-31 22:00:00	45	2020-12-31	40.00	19.55	45.48	-5.48	False	-
59517	2020-12-31 22:30:00	46	2020-12-31	40.00	17.98	41.33	-1.33	False	-
59518	2020-12-31 23:00:00	47	2020-12-31	35.00	14.78	36.66	-1.66	False	-
59519	2020-12-31 23:30:00	48	2020-12-31	25.00	12.57	26.46	-1.46	False	-

59520 rows × 128 columns

Evaluation

In [93]:

```
n_splits=100

train_size = df_trader_ex.index[-1]
train_index_list = list(np.linspace(train_size/n_splits, train_size, n_splits, endpoint = True, dtype='in
t'))
train_index_list

# Confirming the split logic
for train_index in train_index_list:
    # Divide the train/valid set  into 10 folds and pick up it.
    X_train = df_trader_ex.iloc[:train_index]
    print("TRAIN:", train_index)
```

TRAIN: 595
TRAIN: 1190
TRAIN: 1785
TRAIN: 2380
TRAIN: 2975
TRAIN: 3571
TRAIN: 4166
TRAIN: 4761
TRAIN: 5356
TRAIN: 5951
TRAIN: 6547
TRAIN: 7142
TRAIN: 7737
TRAIN: 8332
TRAIN: 8927
TRAIN: 9523
TRAIN: 10118
TRAIN: 10713
TRAIN: 11308
TRAIN: 11903
TRAIN: 12498
TRAIN: 13094
TRAIN: 13689
TRAIN: 14284
TRAIN: 14879
TRAIN: 15474
TRAIN: 16070
TRAIN: 16665
TRAIN: 17260
TRAIN: 17855
TRAIN: 18450
TRAIN: 19046
TRAIN: 19641
TRAIN: 20236
TRAIN: 20831
TRAIN: 21426
TRAIN: 22022
TRAIN: 22617
TRAIN: 23212
TRAIN: 23807
TRAIN: 24402
TRAIN: 24997
TRAIN: 25593
TRAIN: 26188
TRAIN: 26783
TRAIN: 27378
TRAIN: 27973
TRAIN: 28569
TRAIN: 29164
TRAIN: 29759
TRAIN: 30354
TRAIN: 30949
TRAIN: 31545
TRAIN: 32140
TRAIN: 32735
TRAIN: 33330
TRAIN: 33925
TRAIN: 34521
TRAIN: 35116
TRAIN: 35711
TRAIN: 36306

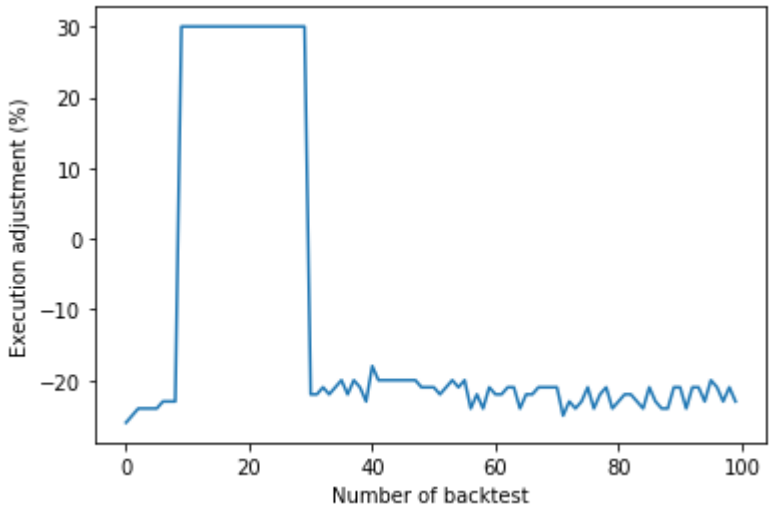
TRAIN: 36901
TRAIN: 37496
TRAIN: 38092
TRAIN: 38687
TRAIN: 39282
TRAIN: 39877
TRAIN: 40472
TRAIN: 41068
TRAIN: 41663
TRAIN: 42258
TRAIN: 42853
TRAIN: 43448
TRAIN: 44044
TRAIN: 44639
TRAIN: 45234
TRAIN: 45829
TRAIN: 46424
TRAIN: 47020
TRAIN: 47615
TRAIN: 48210
TRAIN: 48805
TRAIN: 49400
TRAIN: 49995
TRAIN: 50591
TRAIN: 51186
TRAIN: 51781
TRAIN: 52376
TRAIN: 52971
TRAIN: 53567
TRAIN: 54162
TRAIN: 54757
TRAIN: 55352
TRAIN: 55947
TRAIN: 56543
TRAIN: 57138
TRAIN: 57733
TRAIN: 58328
TRAIN: 58923
TRAIN: 59519

In [94]:

```
BestExec = []

for train_index in train_index_list:
    X_train = df_trader_ex[:train_index]

    cols = []

    # List for the results for evaluation
    PortfolioReturn = []
    StandardDeviation = []
    Max = []
    Min = []
    SharpRatio = []

    for i in list(range(-30, 31, 1)):
        Return_i = list(X_train["Return_exec" + str(i) + "%"])

        Return_i = pd.Series(Return_i)
        PortfolioReturn.append(Return_i.mean().round(2))
        StandardDeviation.append(Return_i.std().round(2))
        Max.append(Return_i.max().round(2))
        Min.append(Return_i.min().round(2))
        SharpRatio.append((Return_i.mean()/Return_i.std()).round(2))
        # Make columns names
        cols.append(i)

    # Make dataframe for evaluation and  switch columns and row.
    df_trader_ex_eval = pd.DataFrame()
    # df_trader_ex_eval = df_trader_ex_eval.T

    # Add columns for evaluation metrics
    df_trader_ex_eval["PortfolioReturn"] = pd.Series(PortfolioReturn)
    df_trader_ex_eval["StandardDeviation"] = pd.Series(StandardDeviation)
    df_trader_ex_eval["Max"] = pd.Series(Max)
    df_trader_ex_eval["Min"] = pd.Series(Min)
    df_trader_ex_eval["SharpRatio"] = pd.Series(SharpRatio)
    df_trader_ex_eval.index = cols
    df_trader_ex_eval =  df_trader_ex_eval.reset_index()
    df_trader_ex_eval = df_trader_ex_eval.rename(columns={"index": "ExecBuffer(%)"})
    df_trader_ex_eval["ExecBuffer(%)"] = df_trader_ex_eval["ExecBuffer(%)"].astype(int)

    BestExec.append(df_trader_ex_eval["ExecBuffer(%)"][df_trader_ex_eval["SharpRatio"] == df_tr
ader_ex_eval["SharpRatio"].max()].max())
```

In [95]:

```
# The best execution of 100 different periods
BestExec = pd.Series(BestExec)
plt.xlabel('Number of backtest')
plt.ylabel('Execution adjustment (%)')
BestExec.plot();
```



In [96]:

```
BestExec.tail()
```

Out[96]:

```
95  -20
96  -21
97  -23
98  -21
99  -23
dtype: int64
```

In [97]:

```
BestExec[-1:]
```

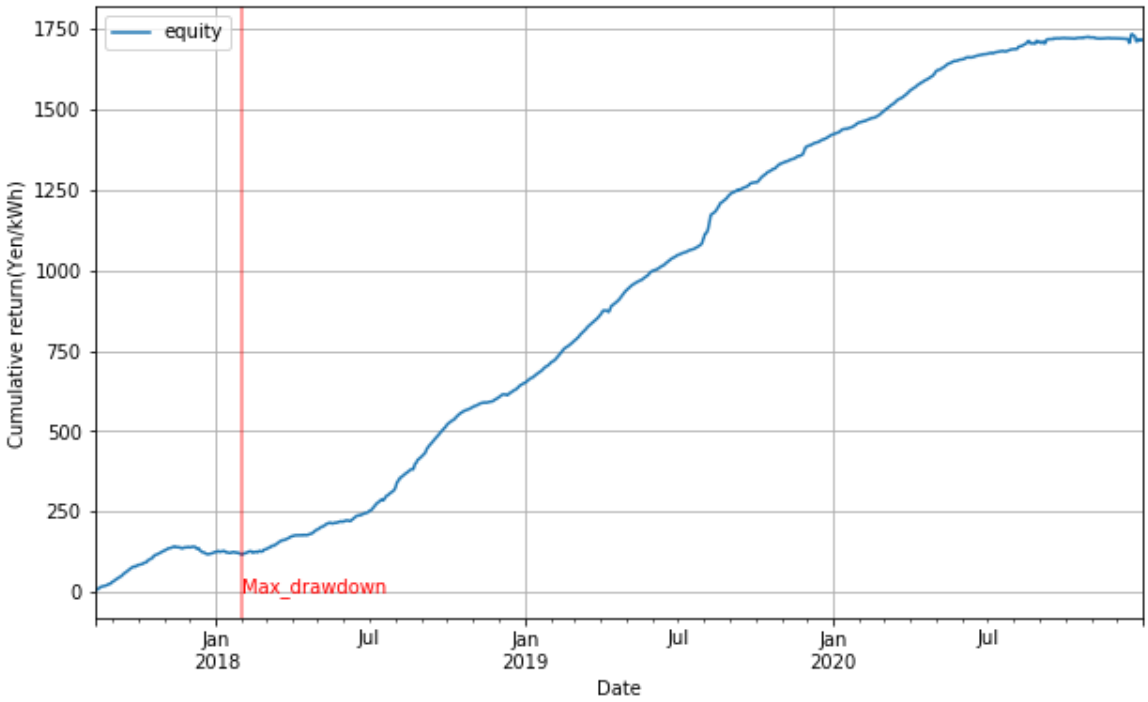
Out[97]:

```
99  -23
dtype: int64
```

In [98]:

```
i = int(BestExec[-1:])

# Pivot table for evaluation metrics
df_trader_ex_table = pd.DataFrame(df_trader_ex.pivot(index='Date', columns='HH', values='Return
_exec' + str(i)  +"%"))
df_trader_ex_table["Expected_Return"] = df_trader_ex_table.loc[: , 0:48].mean(axis=1).round(2)
df_trader_ex_table["StandardDeviation"] = df_trader_ex_table.loc[: , 1:48].std(axis=1).round(2)
df_trader_ex_table["Max"] = df_trader_ex_table.loc[: , 1:48].max(axis=1).round(2)
df_trader_ex_table["Min"] = df_trader_ex_table.loc[: , 1:48].min(axis=1).round(2)
df_trader_ex_table["SharpRatio"] = (df_trader_ex_table.loc[: , 1:48].mean(axis=1)/df_trader_ex_
table.loc[: , 1:48].std(axis=1)).round(2)

# Make a new table for performance graph
risk_ex = pd.DataFrame()
risk_ex["Expected_Return"] = df_trader_ex_table.loc[: , 1:48].mean(axis=1).round(2)
risk_ex["equity"] = risk_ex['Expected_Return'].cumsum()
risk_ex["cummax"] = risk_ex["equity"].cummax()
risk_ex["drawdown"] = -(risk_ex["cummax"] - risk_ex["equity"])

# Pick up max_drawdown
max_drawdown = risk_ex["drawdown"].min()

# Set timestamp of max_drawdown
t_max = pd.to_datetime(risk_ex["drawdown"].idxmin())

fig, ax = plt.subplots(1, figsize=(15,8))
# plt.title('Cumulative return through trading based on the difference of price on DA and Intraday')

risk_ex["equity"].plot(figsize=(10, 6))

plt.text(t_max, 0, 'Max_drawdown', rotation=0, color="r")

print("Expected_Return: " + str(risk_ex["Expected_Return"].mean().round(2)))
print("StandardDeviation: " + str(risk_ex["Expected_Return"].std().round(2)))
print("SharpRatio: " + str(round(risk_ex["Expected_Return"].mean() /  risk_ex["Expected_Return"].
std(), 3)))
print("Cumulative_Return: " + str((risk_ex["cummax"].max().round(2))))
print("Drawdown: " + str(risk_ex["drawdown"].min().round(2)))

ax.set(xlabel="Date", ylabel="Cumulative return(Yen/kWh)")
plt.axvline(t_max, c="r", alpha=0.5)
plt.grid()
plt.legend();
```

Expected_Return: 1.38
StandardDeviation: 1.81
SharpRatio: 0.763
Cumulative_Return: 1733.98
Drawdown: -25.96



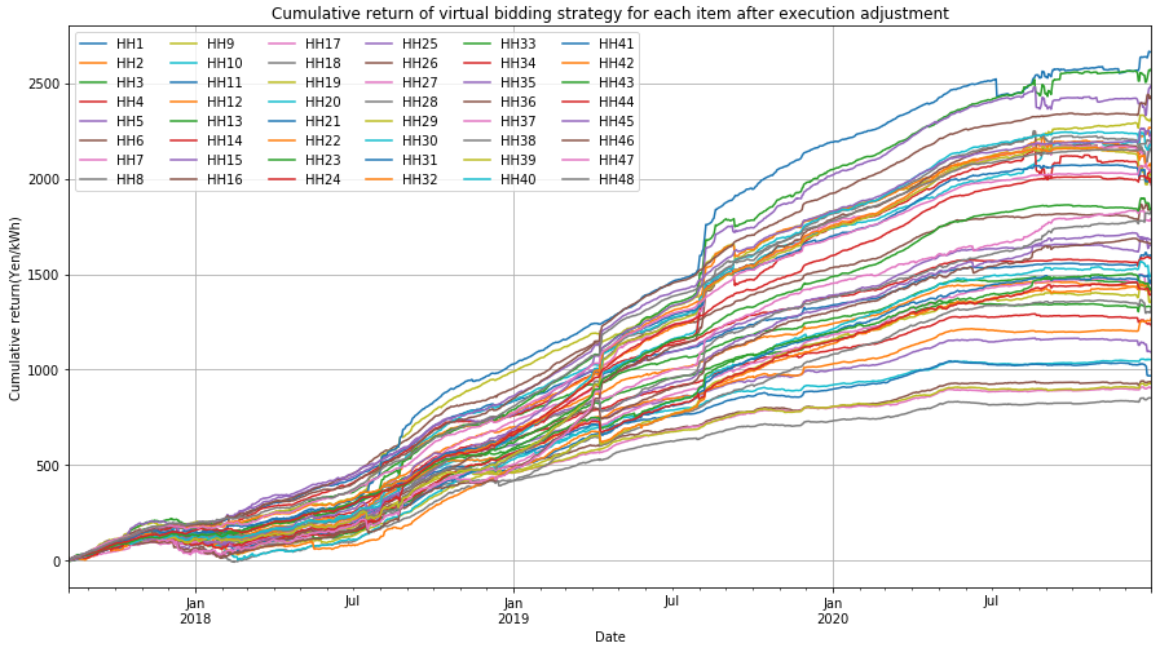
In [99]:

```
# Calculate VaR

equity = 10000000

percs = np.array([1., 5.0, 10.0])
risk_ex["returns"] = np.log(risk_ex["equity"] / risk_ex["equity"].shift(1))
VaR = scs.scoreatpercentile(equity * risk_ex["returns"], percs)
def print_var():
    print('%16s %16s' % ('Confidence Level', 'Value-at-Risk'))
    print(43 * '-')
    for pair in zip(percs, VaR):
        print('%16.0f %16.0f' % (100 - pair[0], -pair[1]))

print_var()
```

```
Confidence Level    Value-at-Risk
-------------------------------------------
              99           161567
              95            23194
              90             2281
```

In [100]:

```
# Plot each 48 item separately
fig, ax = plt.subplots(1, figsize=(15,8))
plt.title('Cumulative return of virtual bidding strategy for each item after execution adjustment')

# Lists for results
Equity = []
ExpectedReturn = []
Max = []
Min = []
StandardDeviation = []
SharpRatio= []
CumMax = []
MaxDrawdown = []

# Execution baffer (%)
i = -23

for HH in list(range(1, 49, 1)):
    # Plot table based on Return
    df_trader_portfolio_ex_table = pd.DataFrame(df_trader_ex.pivot(index='Date', columns='HH', va
lues='Return_exec' + str(i)  +"%"))

    # For plot and for calculating drawdown
    equity = df_trader_portfolio_ex_table[HH].cumsum().round(2)
    # Plot the performance of each HH spot
    equity.plot(label="HH" + str(HH))

    # Keep in a list for calculating drawdown
    Equity.append(equity)
    cummax = equity.cummax()
    drawdown = -(cummax - equity)

    # Calculation for each evaluation on each HH slot
    HH_Return = df_trader_portfolio_ex_table[HH].mean().round(2)
    HH_Std = df_trader_portfolio_ex_table[HH].std().round(2)
    ExpectedReturn.append(HH_Return)
    StandardDeviation.append(HH_Std)
    Max.append(df_trader_portfolio_ex_table[HH].max().round(2))
    Min.append(df_trader_portfolio_ex_table[HH].min().round(2))
    SharpRatio.append((HH_Return / HH_Std).round(2))
    CumMax.append(cummax.max().round(2))
    MaxDrawdown.append(drawdown.min().round(2))

ax.set(xlabel="Date", ylabel="Cumulative return(Yen/kWh)")
plt.grid()
plt.legend(loc="upper left", ncol=6);
```



In [101]:

```
df_trader_HH_ex_Eval = pd.DataFrame()
df_trader_HH_ex_Eval["HH"] =  pd.Series(list(range(1,49,1)))

df_trader_HH_ex_Eval["ExpectedReturn"]  = pd.Series(ExpectedReturn)
df_trader_HH_ex_Eval["Max"]  = pd.Series(Max)
df_trader_HH_ex_Eval["Min"]  = pd.Series(Min)
df_trader_HH_ex_Eval["StandardDeviation"]  = pd.Series(StandardDeviation)
df_trader_HH_ex_Eval["SharpRatio"]  = pd.Series(SharpRatio)
df_trader_HH_ex_Eval["CumMax"] = pd.Series(CumMax)
df_trader_HH_ex_Eval["MaxDrawdown"]  = pd.Series(MaxDrawdown)
```

In [102]:

```
df_trader_HH_ex_Eval[df_trader_HH_ex_Eval["CumMax"] == df_trader_HH_ex_Eval["CumMax"].max()]
```

Out[102]:

	HH	ExpectedReturn	Max	Min	StandardDeviation	SharpRatio	CumMax	MaxDrawdo
30	31	2.15	51.36	-92.54	5.64	0.38	2665.82	-92.

HH31 --> 15:00

In [103]:

```
df_trader_HH_ex_Eval[df_trader_HH_ex_Eval["CumMax"] == df_trader_HH_ex_Eval["CumMax"].min()]
```

Out[103]:

	HH	ExpectedReturn	Max	Min	StandardDeviation	SharpRatio	CumMax	MaxDrawdown
7	8	0.69	14.1	-7.4	1.54	0.45	852.54	-36.38

HH8 --> 3:30

In [104]:

```
df_trader_HH_ex_Eval[df_trader_HH_ex_Eval["MaxDrawdown"] == df_trader_HH_ex_Eval["MaxDrawdown"].max()]
```

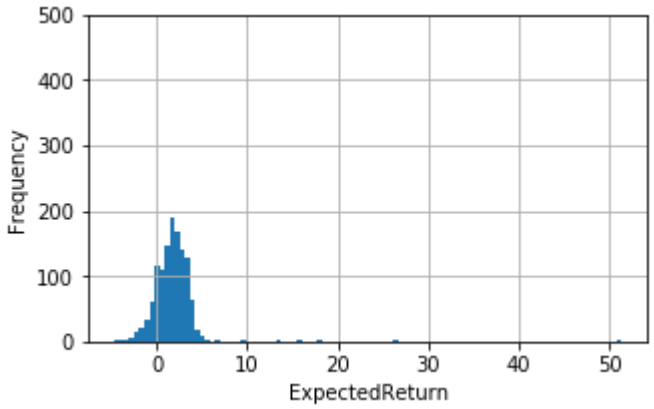
Out[104]:

	HH	ExpectedReturn	Max	Min	StandardDeviation	SharpRatio	CumMax	MaxDrawdow
47	48	1.77	51.39	-4.74	2.73	0.65	2202.79	-13.3

HH48 --> 23:30

In [105]:

```
fig, ax = plt.subplots(1, figsize=(5,3))
df_trader_portfolio_ex_table[48].hist(bins=100)
plt.ylim(0, 500)
ax.set(xlabel="ExpectedReturn", ylabel="Frequency");
```



In [106]:

```
df_trader_HH_ex_Eval[df_trader_HH_ex_Eval["MaxDrawdown"] == df_trader_HH_ex_Eval["MaxDrawdown"].min()]
```

Out[106]:

	HH	ExpectedReturn	Max	Min	StandardDeviation	SharpRatio	CumMax	MaxDrawdo
38	39	1.63	42.12	-61.31	4.78	0.34	2168.37	-200.

HH39 --> 19:00

In [107]:

```
fig, ax = plt.subplots(1, figsize=(5,3))
df_trader_portfolio_ex_table[39].hist(bins=100)
plt.ylim(0, 500)
ax.set(xlabel="ExpectedReturn", ylabel="Frequency");
```