In [1]:
```
from __future__ import print_function
from __future__ import division

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set_context(rc={'figure.figsize': (14, 7) } )
figzize_me = figsize =(14, 7)

import os
import sys
# 使用insert 0即只使用github，避免交叉使用了pip安装的abupy，导致的版本不一致问题
sys.path.insert(0, os.path.abspath('../'))
import abupy
```

- 因为需要全市场回测所以本章无法使用沙盒数据，《量化交易之路》中的原始示例使用的是美股市场，这里的示例改为使用A股市场。
- 本节建议对照阅读abu量化文档第20-23节内容
- 本节的基础是在abu量化文档中第20节内容完成运行后有A股训练集交易和A股测试集交易数据之后

abu量化系统github地址 (https://github.com/bbfamily/abu) (您的star是我的动力！)

abu量化文档教程ipython notebook (https://github.com/bbfamily/abu/tree/master/abupy_lecture)

第11章 量化系统-机器学习•ABU

In [2]:
```
from abupy import AbuFactorAtrNStop, AbuFactorPreAtrNStop, AbuFactorCl
from abupy import abu, EMarketTargetType, AbuMetricsBase, ABuMarketDra
from abupy import EMarketTargetType, EDataCacheType, EMarketSourceType
from abupy import AbuUmpMainDeg, AbuUmpMainJump, AbuUmpMainPrice, AbuU
from abupy import AbuUmpEdgeDeg, AbuUmpEdgePrice, AbuUmpEdgeWave, AbuU
from abupy import AbuUmpMainDegExtend, ump, Parallel, delayed, AbuMulP
,
# 关闭沙盒数据
abupy.env.disable_example_env_ipython()
```

```
disable example env
```

In [3]:
```
abupy.env.g_market_target = EMarketTargetType.E_MARKET_TARGET_CN
abupy.env.g_data_fetch_mode = EMarketDataFetchMode.E_DATA_FETCH_FORCE_
abu_result_tuple_train = abu.load_abu_result_tuple(n_folds=5, store_ty
                                                   custom_name='train_cn')
abu_result_tuple_test = abu.load_abu_result_tuple(n_folds=5, store_typ
```

```
                                                    custom_name='test_cn')
ABuProgress.clear_output()
print('训练集结果：')
metrics_train = AbuMetricsBase.show_general(*abu_result_tuple_train, o
print('测试集结果：')
metrics_test  = AbuMetricsBase.show_general(*abu_result_tuple_test, on
```

训练集结果：

买入后卖出的交易数量:25346
买入后尚未卖出的交易数量:341
胜率:36.4555%
平均获利期望:10.8069%
平均亏损期望:-7.3617%
盈亏比:0.8648
策略收益：-1.3295%
基准收益：-25.7195%
策略年化收益：-0.5310%
基准年化收益：-10.2715%
策略买入成交比例:22.8987%
策略资金利用率比例:70.5727%
策略共执行631个交易日

测试集结果：

买入后卖出的交易数量:2866
买入后尚未卖出的交易数量:36
胜率:35.2059%
平均获利期望:11.0019%
平均亏损期望:-7.1648%

盈亏比:0.8531
策略收益：23.0973%
基准收益：-25.7195%
策略年化收益：9.2243%
基准年化收益：-10.2715%
策略买入成交比例:26.7057%
策略资金利用率比例:67.1742%
策略共执行631个交易日

11.1 搜索引擎与量化交易

- 本节建议对照abu量化文档第15节内容进行阅读

In [4]:
```
orders_pd_train = abu_result_tuple_train.orders_pd
```

In [5]:
```
# 选择失败的前20笔交易绘制交易快照
# 这里只是示例，实战中根据需要挑选，rank或者其他方式
plot_simple = orders_pd_train[orders_pd_train.profit_cg < 0][:20]
# save=True保存在本地， 文件保存在~/abu/data/save_png/中
ABuMarketDrawing.plot_candle_from_order(plot_simple, save=True)
```

11.2 主裁

11.2.1 角度主裁

请对照阅读ABU量化系统使用文档： 第15节 中相关内容

In [6]:
```
from abupy import AbuUmpMainDeg
# 参数为orders_pd
ump_deg = AbuUmpMainDeg(orders_pd_train)
# df即由之前ump_main_make_xy生成的类df，表11-1所示
ump_deg.fiter.df.head()
```

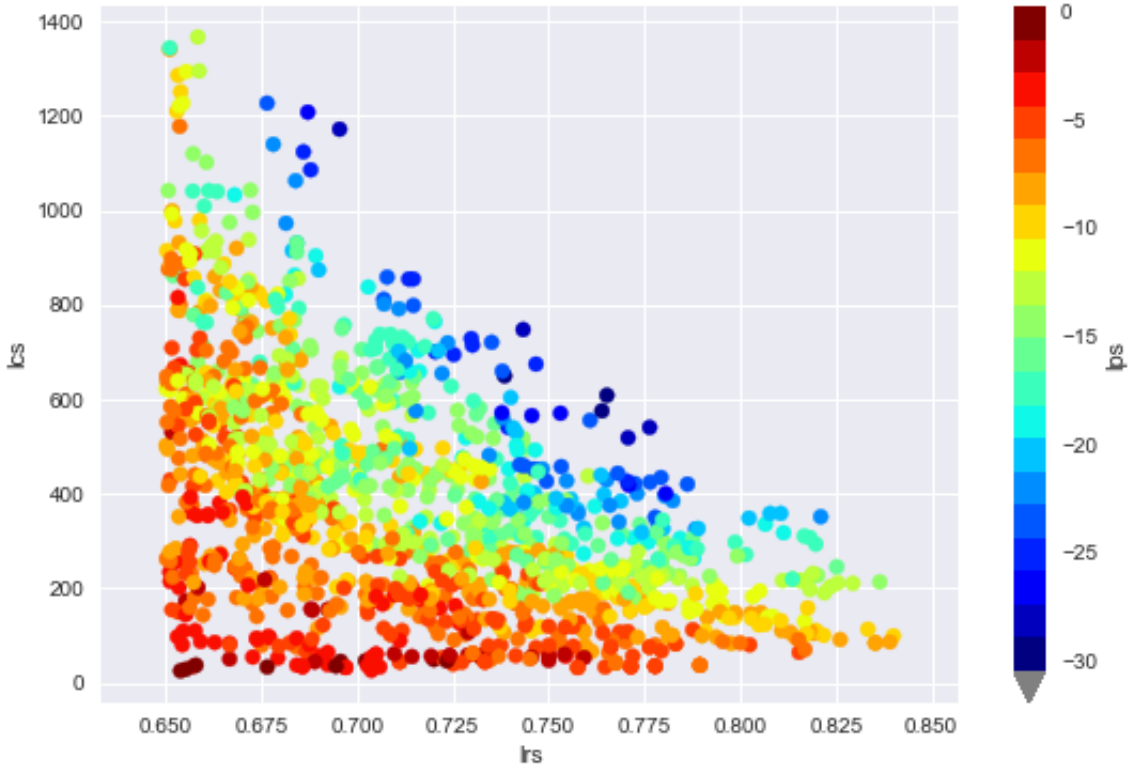
Out[6]:

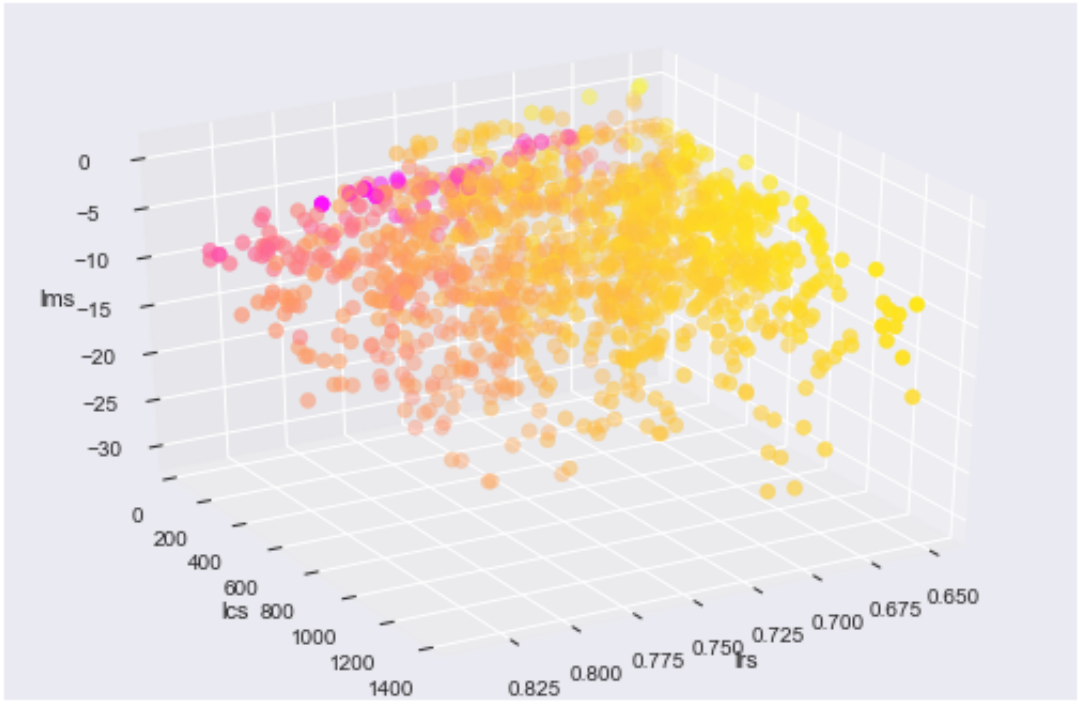
	result	buy_deg_ang252	buy_deg_ang42	buy_deg_ang60	buy_deg_ang21
2015-07-02	0	1.856	3.637	8.057	6.702
2015-07-02	0	3.670	3.670	3.670	6.761
2015-07-02	1	0.720	0.720	0.720	1.374
2015-07-02	1	0.730	0.730	0.730	1.391
2015-07-02	1	-0.000	-0.000	-0.000	0.000

耗时操作，快的电脑大概几分钟，具体根据电脑性能，cpu数量，启动多进程进行训练：

In [7]:
```
_ = ump_deg.fit(brust_min=False)
```

```
pid:2414 gmm fit:100.0%
```





In [8]:
```
ump_deg.cprs
```

Out[8]:

	lcs	lms	lps	lrs
83_37	38	-0.1622	-6.1620	0.7895
80_78	261	-0.0127	-3.3248	0.6552
75_29	548	-0.0081	-4.4382	0.6642
62_37	49	-0.1216	-5.9592	0.7347
79_50	227	-0.0618	-14.0296	0.7841
79_53	496	-0.0293	-14.5182	0.7440
79_52	95	-0.0344	-3.2677	0.6737
62_33	171	-0.0481	-8.2281	0.7076
84_48	211	-0.0597	-12.5978	0.8294
79_57	443	-0.0219	-9.7193	0.7178
...	...	...	...	...
78_31	309	-0.0321	-9.9110	0.7023
61_7	415	-0.0344	-14.2612	0.7084
78_33	180	-0.0616	-11.0934	0.7778
71_32	132	-0.0748	-9.8782	0.8106
71_31	343	-0.0359	-12.3261	0.7318
71_30	115	-0.0406	-4.6648	0.7304
64_49	327	-0.0639	-20.8981	0.7890
72_7	391	-0.0504	-19.6970	0.7519
72_4	160	-0.0431	-6.8947	0.6562
83_67	159	-0.0609	-9.6896	0.8176

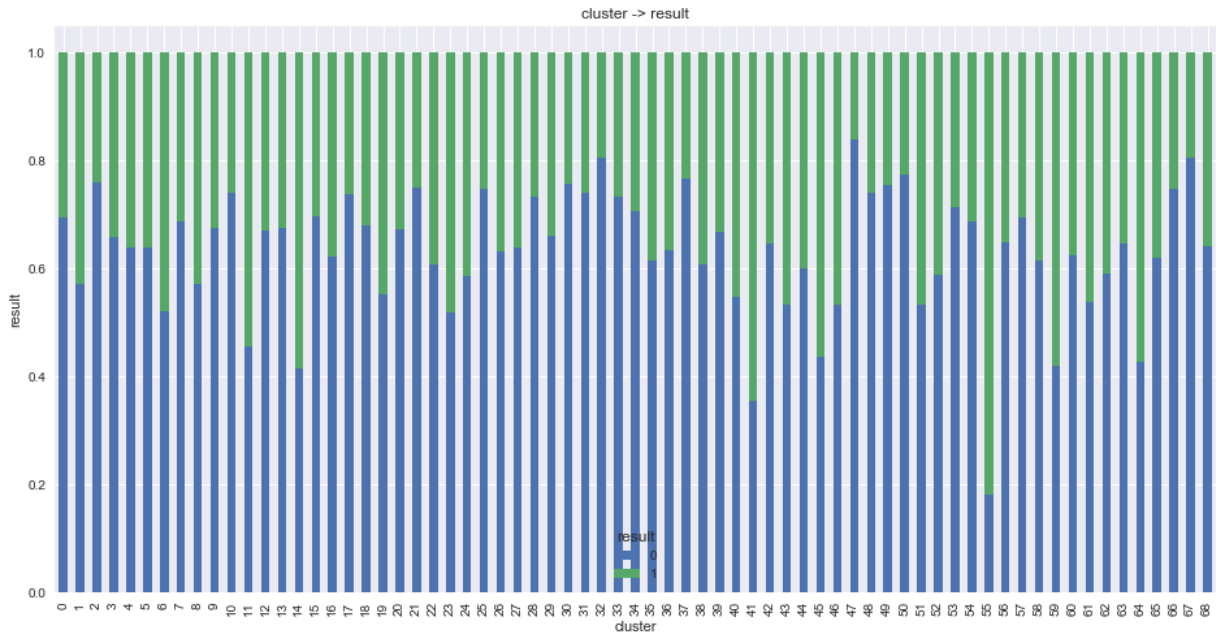
1363 rows × 4 columns

In [9]:
```
max_failed_cluster = ump_deg.cprs.loc[ump_deg.cprs.lrs.argmax()]
print('失败概率最大的分类簇{0}, 失败率为{1:.2f}%, 簇交易总数{2}, ' \
      '簇平均交易获利{3:.2f}%'.format(ump_deg.cprs.lrs.argmax(),
                                  max_failed_cluster.lrs * 100,
                                  max_failed_cluster.lcs,
                                  max_failed_cluster.lms * 100))
```

```
失败概率最大的分类簇69_47, 失败率为84.00%, 簇交易总数100.0, 簇平均交易获利-8
.95%
```

In [10]:
```
cpt = int(ump_deg.cprs.lrs.argmax().split('_')[0])
print(cpt)
ump_deg.show_parse_rt(ump_deg.rts[cpt])
```

```
69
```



In [11]:
```
max_failed_cluster_orders = ump_deg.nts[ump_deg.cprs.lrs.argmax()]
# 表11-3所示
max_failed_cluster_orders
```

Out[11]:

	result	buy_deg_ang252	buy_deg_ang42	buy_deg_ang60	buy_deg_ang21	ind	cluste
2015-07-02	1	51.515	-0.000	26.250	0.000	19	4
2015-07-02	0	50.557	-0.000	17.802	0.000	43	4
2015-07-02	0	25.290	5.739	22.604	9.599	45	4
2015-07-13	1	49.913	16.889	40.568	1.380	56	4
2015-07-23	0	36.896	1.278	25.467	5.727	67	4
2015-07-24	0	32.255	2.258	19.986	6.265	70	4
2015-07-30	1	28.554	6.943	13.750	8.763	94	4
2015-08-18	0	36.322	6.205	23.845	7.671	183	4
2015-08-18	0	36.322	6.205	23.845	7.671	189	4

2015-08-18	0	37.760	13.014	24.538	10.620	190	4
...	...	...	...	...	...	...	
2017-04-27	0	38.307	5.616	30.424	5.241	19150	4
2017-05-05	0	38.703	10.534	20.332	6.891	19215	4
2017-05-10	1	38.699	7.598	26.609	3.718	19238	4
2017-05-10	1	38.699	7.598	26.609	3.718	19239	4
2017-05-18	0	37.624	8.784	17.653	6.887	19298	4
2017-07-05	0	45.527	10.283	24.937	2.107	20308	4
2017-08-01	0	44.037	27.494	24.903	9.737	21460	4
2017-09-08	0	31.546	16.339	30.421	7.934	23230	4
2017-11-07	0	47.720	23.196	25.368	6.949	25147	4
2017-11-14	0	36.652	15.882	15.108	8.277	25271	4

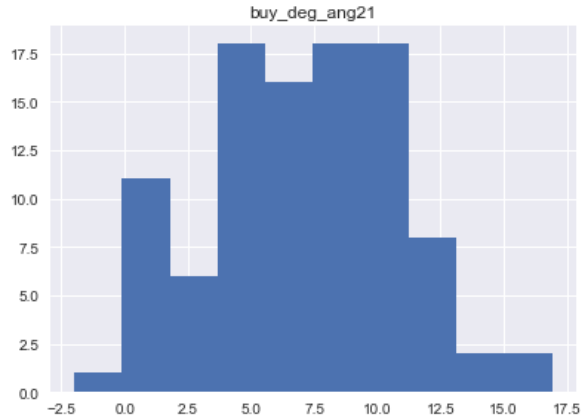
100 rows × 7 columns

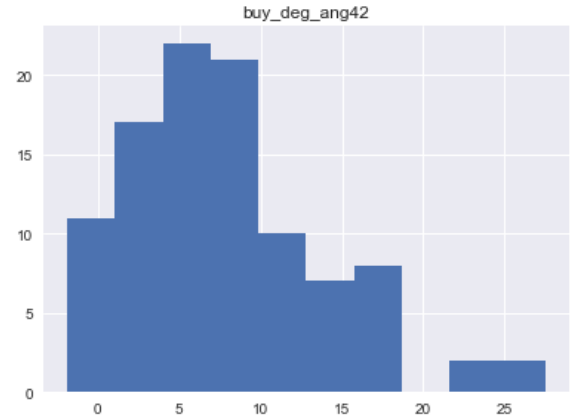
由于不是同一份沙盒数据，所以下面结果内容与书中分析内容不符，需要按照实际情况分析:

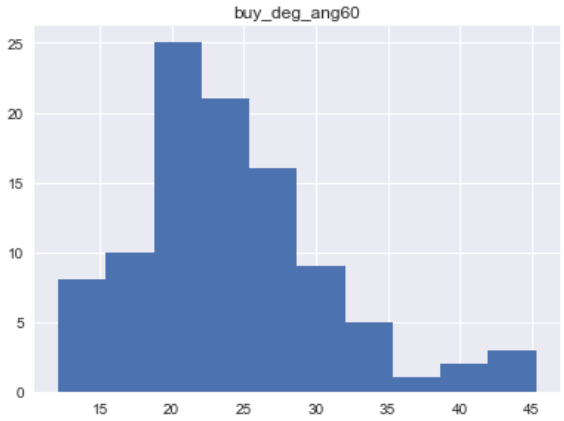
比如下面的特征即是42日和60日的deg格外大，21和252相对训练集平均值也很大:

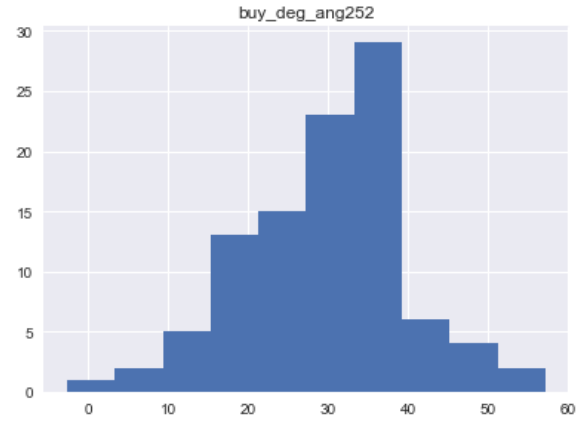
In [12]:
```
from abupy import ml

ml.show_orders_hist(max_failed_cluster_orders, ['buy_deg_ang21', 'buy_
print('分类簇中deg_ang60平均值为{0:.2f}'.format(
    max_failed_cluster_orders.buy_deg_ang60.mean()))

print('分类簇中deg_ang21平均值为{0:.2f}'.format(
    max_failed_cluster_orders.buy_deg_ang21.mean()))

print('分类簇中deg_ang42平均值为{0:.2f}'.format(
    max_failed_cluster_orders.buy_deg_ang42.mean()))

print('分类簇中deg_ang252平均值为{0:.2f}'.format(
    max_failed_cluster_orders.buy_deg_ang252.mean()))
```

```
分类簇中deg_ang60平均值为24.14
分类簇中deg_ang21平均值为7.02
分类簇中deg_ang42平均值为7.95
分类簇中deg_ang252平均值为30.13
```



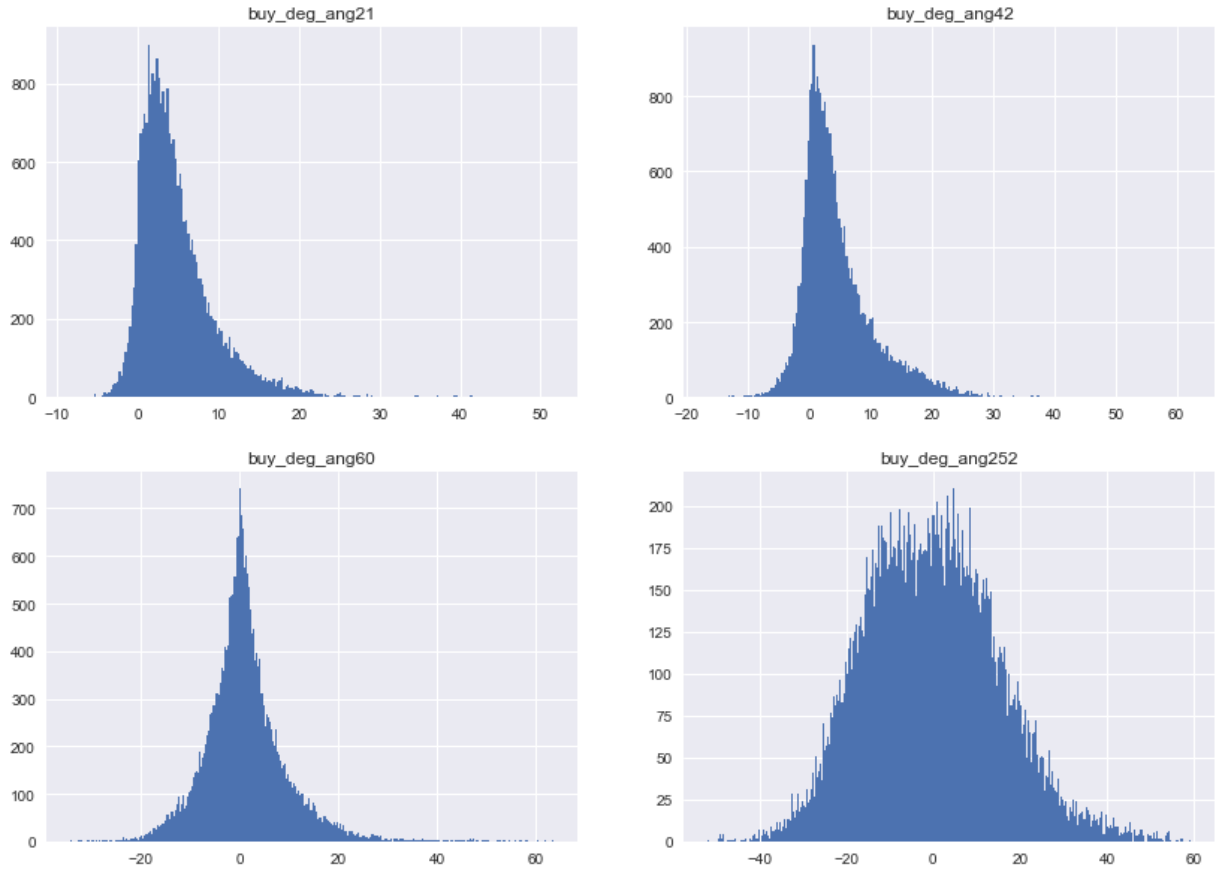






In [13]:
```
ml.show_orders_hist(orders_pd_train, ['buy_deg_ang21', 'buy_deg_ang42'
print('训练数据集中deg_ang60平均值为{0:.2f}'.format(
    orders_pd_train.buy_deg_ang60.mean()))

print('训练数据集中deg_ang21平均值为{0:.2f}'.format(
    orders_pd_train.buy_deg_ang21.mean()))

print('训练数据集中deg_ang42平均值为{0:.2f}'.format(
    orders_pd_train.buy_deg_ang42.mean()))

print('训练数据集中deg_ang252平均值为{0:.2f}'.format(
    orders_pd_train.buy_deg_ang252.mean()))
```

```
训练数据集中deg_ang60平均值为1.10
训练数据集中deg_ang21平均值为4.87
训练数据集中deg_ang42平均值为4.70
训练数据集中deg_ang252平均值为-0.32
```



In [14]:
```
progress = AbuProgress(len(max_failed_cluster_orders), 0, label='plot s
for ind in np.arange(0, len(max_failed_cluster_orders)):
    progress.show(ind)
    order_ind = int(max_failed_cluster_orders.iloc[ind].ind)
    # 交易快照文件保存在~/abu/data/save_png/中
    ABuMarketDrawing.plot_candle_from_order(ump_deg.fiter.order_has_re
```

```
plot snap::0.0%

plot snap::99.0%
```

交易快照文件保存在~/abu/data/save_png/中, 下面打开对应目录：save_png

In [*]:
```
if abupy.env.g_is_mac_os:
    !open $abupy.env.g_project_data_dir
else:
    !echo $abupy.env.g_project_data_dir
```

11.2.2 使用全局最优对分类簇集合进行筛选

In [*]:
```
brust_min = ump_deg.brust_min()
brust_min
```

```
AbuUmpMainDeg: brute min progress::0.0%

AbuUmpMainDeg: brute min progress::69.94%
```

In [*]:
```
llps = ump_deg.cprs[(ump_deg.cprs['lps'] <= brust_min[0]) & (ump_deg.cp
llps
```

In [*]:
```
ump_deg.choose_cprs_component(llps)
```

In [*]:
```
ump_deg.dump_clf(llps)
```

11.2.3 跳空主裁

请对照阅读ABU量化系统使用文档： 第16节 UMP主裁交易决策 中相关内容

In [*]:
```
from abupy import AbuUmpMainJump
# 耗时操作，大概需要10几分钟，具体根据电脑性能，cpu情况
ump_jump = AbuUmpMainJump.ump_main_clf_dump(orders_pd_train, save_orde
```

In [*]:
```
ump_jump.fiter.df.head()
```

下面这个的这个拦截特征比较明显，两天前才发生向上跳空的交易：

In [*]:
```
print('失败概率最大的分类簇{0}'.format(ump_jump.cprs.lrs.argmax()))
# 拿出跳空失败概率最大的分类簇
max_failed_cluster_orders = ump_jump.nts[ump_jump.cprs.lrs.argmax()]
# 显示失败概率最大的分类簇，表11-6所示
max_failed_cluster_orders
```

In [*]:
```
ml.show_orders_hist(max_failed_cluster_orders, feature_columns=['buy_d
                                                                'buy_d

print('分类簇中jump_up_power平均值为{0:.2f}， 向上跳空平均天数{1:.2f}'.forma
    max_failed_cluster_orders.buy_jump_up_power.mean(), max_failed_clu

print('分类簇中jump_down_power平均值为{0:.2f}, 向下跳空平均天数{1:.2f}'.form
    max_failed_cluster_orders.buy_jump_down_power.mean(), max_failed_c

print('训练数据集中jump_up_power平均值为{0:.2f}，向上跳空平均天数{1:.2f}'.for
    orders_pd_train.buy_jump_up_power.mean(), orders_pd_train.buy_diff_

print('训练数据集中jump_down_power平均值为{0:.2f}, 向下跳空平均天数{1:.2f}'.
    orders_pd_train.buy_jump_down_power.mean(), orders_pd_train.buy_di
```

11.2.4 价格主裁

请对照阅读ABU量化系统使用文档： 第16节 UMP主裁交易决策 中相关内容

In [*]:
```
from abupy import AbuUmpMainPrice
ump_price = AbuUmpMainPrice.ump_main_clf_dump(orders_pd_train, save_or
```

In [*]:
```
ump_price.fiter.df.head()
```

In [*]:
```
print('失败概率最大的分类簇{0}'.format(ump_price.cprs.lrs.argmax()))

# 拿出价格失败概率最大的分类簇
max_failed_cluster_orders = ump_price.nts[ump_price.cprs.lrs.argmax()]
# 表11-8所示
max_failed_cluster_orders
```

11.2.5 波动主裁

请对照阅读ABU量化系统使用文档： 第16节 UMP主裁交易决策 中相关内容

In [*]:
```
from abupy import AbuUmpMainWave
ump_wave = AbuUmpMainWave.ump_main_clf_dump(orders_pd_train, save_orde
```

In [*]:
```
ump_wave.fiter.df.head()
```

In [*]:
```
print('失败概率最大的分类簇{0}'.format(ump_wave.cprs.lrs.argmax()))
# 拿出波动特征失败概率最大的分类簇
max_failed_cluster_orders = ump_wave.nts[ump_wave.cprs.lrs.argmax()]
# 表11-10所示
max_failed_cluster_orders
```

In [*]:
```
ml.show_orders_hist(max_failed_cluster_orders, feature_columns=['buy_wa

print('分类簇中wave_score1平均值为{0:.2f}'.format(
    max_failed_cluster_orders.buy_wave_score1.mean()))

print('分类簇中wave_score3平均值为{0:.2f}'.format(
    max_failed_cluster_orders.buy_wave_score3.mean()))

ml.show_orders_hist(orders_pd_train, feature_columns=['buy_wave_score1

print('训练数据集中wave_score1平均值为{0:.2f}'.format(
    orders_pd_train.buy_wave_score1.mean()))

print('训练数据集中wave_score3平均值为{0:.2f}'.format(
    orders_pd_train.buy_wave_score1.mean()))
```

11.2.6 验证主裁是否称职

请对照阅读ABU量化系统使用文档： 第21节 A股UMP决策 中相关内容

In [*]:
```
# 选取有交易结果的数据order_has_result
order_has_result = abu_result_tuple_test.orders_pd[abu_result_tuple_te
```

In [ ]:
```
ump_wave.best_hit_cnt_info(ump_wave.llps)
```

In [ ]:
```
from abupy import AbuUmpMainDeg, AbuUmpMainJump, AbuUmpMainPrice, AbuU
ump_deg = AbuUmpMainDeg(predict=True)
ump_jump = AbuUmpMainJump(predict=True)
ump_price = AbuUmpMainPrice(predict=True)
ump_wave = AbuUmpMainWave(predict=True)
```

In [ ]:
```
def apply_ml_features_ump(order, predicter, progress, need_hit_cnt):
    if not isinstance(order.ml_features, dict):
        import ast
        # 低版本pandas dict对象取出来会成为str
        ml_features = ast.literal_eval(order.ml_features)
    else:
        ml_features = order.ml_features
    progress.show()
    # 将交易单中的买入时刻特征传递给ump主裁决策器，让每一个主裁来决策是否进行拦截
    return predicter.predict_kwargs(need_hit_cnt=need_hit_cnt, **ml_fe

def pararllel_func(ump, ump_name):
    with AbuMulPidProgress(len(order_has_result), '{} complete'.format
        # 启动多进程进度条，对order_has_result进行apply
        ump_result = order_has_result.apply(apply_ml_features_ump, axi
    return ump_name, ump_result

# 并行处理4个主裁，每一个主裁启动一个进程进行拦截决策
parallel = Parallel(
    n_jobs=4, verbose=0, pre_dispatch='2*n_jobs')
out = parallel(delayed(pararllel_func)(ump, ump_name)
                               for ump, ump_name in zip([ump_deg, ump_j
                                                         ['ump_deg', 'um

# 将每一个进程中的裁判的拦截决策进行汇总
for sub_out in out:
    order_has_result[sub_out[0]] = sub_out[1]
```

In [ ]:
```
block_pd = order_has_result.filter(regex='^ump_*')
# 把所有主裁的决策进行相加
block_pd['sum_bk'] = block_pd.sum(axis=1)
block_pd['result'] = order_has_result['result']
# 有投票1的即会进行拦截
block_pd = block_pd[block_pd.sum_bk > 0]
print('四个裁判整体拦截正确率{:.2f}%'.format(block_pd[block_pd.result == -
block_pd.tail()
```

In [ ]:
```
print('角度裁判拦截正确率{:.2f}%, 拦截交易数量{}'.format(*sub_ump_show('ump_
print('角度扩展裁判拦拦截正确率{:.2f}%, 拦截交易数量{}'.format(*sub_ump_show
print('单混裁判拦截正确率{:.2f}%, 拦截交易数量{}'.format(*sub_ump_show('ump_
print('价格裁判拦截正确率{:.2f}%, 拦截交易数量{}'.format(*sub_ump_show('ump_
```

11.2.7 在abu系统中开启主裁拦截模式

请对照阅读ABU量化系统使用文档： 第21节 A股UMP决策 中相关内容

11.3 边裁

11.3.1 角度边裁

请对照阅读ABU量化系统使用文档： 第17节 UMP边裁交易决策，第21节 A股UMP决策 中相关内容

11.3.2 价格边裁

请对照阅读ABU量化系统使用文档： 第17节 UMP边裁交易决策，第21节 A股UMP决策 中相关内容

11.3.3 波动边裁

请对照阅读ABU量化系统使用文档： 第17节 UMP边裁交易决策，第21节 A股UMP决策 中相关内容

11.3.4 综合边裁

请对照阅读ABU量化系统使用文档： 第17节 UMP边裁交易决策，第21节 A股UMP决策 中相关内容

11.3.5 验证边裁是否称职

请对照阅读ABU量化系统使用文档： 第21节 A股UMP决策 中相关内容

11.3.6 在abu系统中开启边裁拦截模式

请对照阅读ABU量化系统使用文档： 第21节 A股UMP决策 中相关内容

In [ ]: