《量化投资基础》教程

一、量化投资简介

1.0 量化投资基础概念

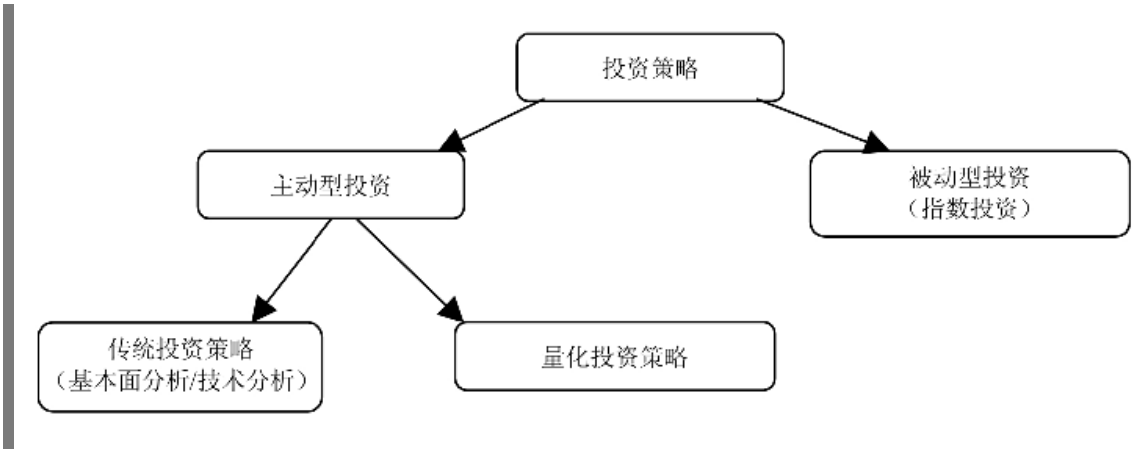
> *什么是量化投资? 简单来讲，量化投资就是利用计算机科技并采用一定的数学模型去实现投资理念、实现投资策略的过程。*

- 量化投资的主要内容包括：量化选股、量化择时、股指期货套利、商品期货套利、统计套利、期权套利、算法交易、ETF/LOF套利、高频交易等。

1. 量化选股：量化选股就是采用数量的方法判断某个公司是否值得买入的行为。根据b某个方法，如果该公司满足了该方法的条件，则放入股票池；如果不满足，则从股票池中剔除。
2. 量化择时: 是指利用某种方法来判断大势的走势情况，是上涨还是下跌或者是盘整。如果判断是上涨，则买入持有；如果判断是下跌，则卖出清仓；如果判断是震荡，则进行高抛低吸;
3. 股指期货套利: 股指期货套利是指利用股指期货市场存在的不合理价格，同时参与股指期货与股票现货市场交易，或者同时进行不同期限，不同（但相近）类别股票指数合约交易，以赚取差价的行为，股指期货套利主要分为期现套利和跨期套利两种。
4. 商品期货套利:商品期货套利盈利的逻辑原理是基于：相关商品在不同地点、不同时间对应都有一个合理的价格差价；由于价格的波动性，价格差价经常出现不合理；不合理必然要回到合理；不合理回到合理的这部分价格区间就是盈利区间。
5. 统计套利：统计套利是利用证券价格的历史统计规律进行的套利，是一种风险套利，其风险在于这种历史统计规律在未来一段时间内是否继续存在。
6. 期权套利：期权套利交易是指同时买进卖出同一相关期货，但不同敲定价格或不同到期月份的看涨或看跌期权合约，希望在日后对冲交易部位或履约时获利的交易。
7. 算法交易： 又被称为自动交易或者机器交易，它指的是通过使用计算机程序来发出交易指令。在交易中，程序可以决定的范围包括交易时间的选择、交易的价格，甚至可以包括最后需要成交的证券数量。

- 量化投资的基础理论知识包括：人工智能、数据挖掘、小波分析、支持向量机、分形理论和随机过程。量化投资需要的IT技术包括：数据库、数据仓库、面向对象编程等。

- 投资策略分类



- 传统投资的不足：

1. 传统主动型投资策略受到人类思维可以处理的信息量的限制。人类思维在任何时候都只能考虑有限数目的变量。例如，对于600只的股票样本，被一个传统主动型基金经理紧密跟踪的也许只包括200只，这样就会明显排除从其他股票获益的机会。
2. 传统主动型投资策略容易受到认知偏差的影响。例如，大多数人都只愿意记住自己成功的喜悦而不愿记住失败的教训。
3. 传统主动型投资策略更强调收益率而不是风险控制，更加偏重个股挖掘而不是投资组合构造。

- 量化投资的优势：

1. 纪律性。严格执行量化投资模型所给出的投资建议，而不是随着投资者情绪的变化而随意更改。
2. 系统性。量化投资的系统性特征主要包括多层次的量化模型、多角度的观察及海量数据的观察等。多层次模型主要包括大类资产配置模型、行业选择模型、精选个股模型等。多角度观察主要包括对宏观周期、市场结构、估值、成长、盈利质量、分析师盈利预测、市场情绪等多个角度的分析。
3. 及时性。及时快速地跟踪市场变化，不断发现能够提供超额收益的新的统计模型，寻找新的交易机会。
4. 准确性。准确客观评价交易机会，克服主观情绪偏差，妥善运用套利的思想。量化投资正是在找估值洼地，通过全面、系统性的扫描，捕捉错误定价、错误估值带来的机会。定性投资经理大部分时间在琢磨哪一个企业是伟大的企业，哪个股票是可以翻倍的股票；而量化投资经理大部分精力花在分析哪里是估值洼地，哪一个品种被低估了，买入低估的，卖出高估的。
5. 分散化：在控制风险的条件下，充当准确实现分散化投资目标的工具。分散化，也可以说量化投资是靠概率取胜。这表现为两个方面：一是量化投资不断地从历史中挖掘有望在未来重复的历史规律并且加以利用，这些历史规律都是有较大概率获胜的策略；二是依靠筛选出股票组合来取胜，而不是一只或几只股票取胜，从投资组合理念来看也是捕获大概率获胜的股票，而不是押宝到单个股票上。

- 量化投资和传统投资收益比较

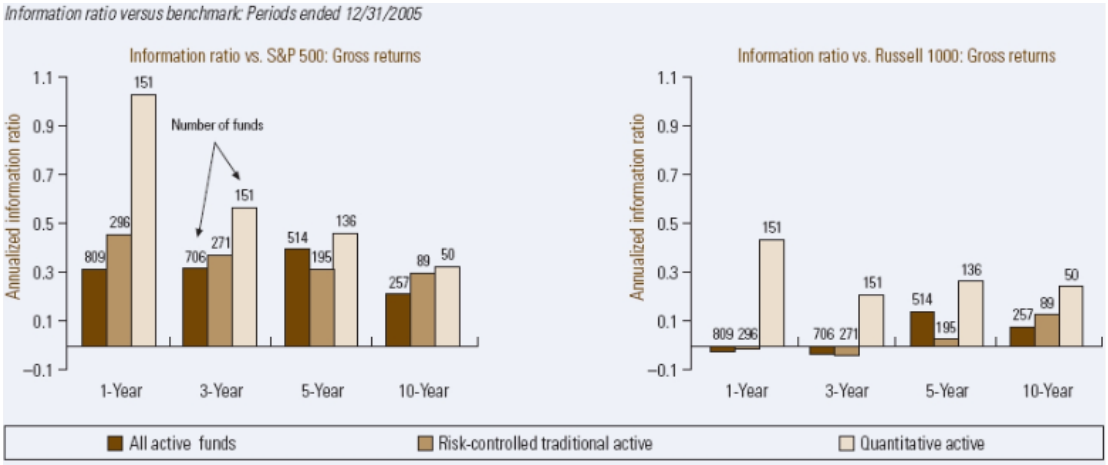


图1-2 量化投资与传统投资策略业绩比较（1996—2005年）

- 量化投资的历史

> *量化投资早在五十多年之前已经应用于证券市场，经过几十年的发展，目前量化投资在海外市场已经非常普及。*
> *量化投资的历史可以追溯到上个世纪60 年代，时任MIT 教授的爱德华索普利用21点原理发明了科学股票市场系统，他是最早一批采用纯数学技术从股市中赚钱的人，可谓是量化投资的鼻祖。*
> *1970 年，数学家和物理学家就开始尝试着把量化交易技术应用于证券市场。*
> *1971 年，巴克莱投资管理公司发行世界上第一只指数基金。*
> *1973 年，美国伯克利大学教授罗森伯格成立了巴尔罗森伯格管理公司，其后发展为量化投资界众所周知的BARRA。*
> *1977 年，巴克莱投资管理公司发行世界上第一只量化投资基金。到了1988 年，全球量化基金数量达到了21 只，管理资金规模突破了80 亿美元。*
> *伴随着电子计算机的发展，量化投资在1995 年之后进入了高速发展阶段。*
> *2007 年，西蒙斯管理的大奖章量化基金，近20 年平均年收益率高达35%，超过了股神巴菲特的年收益率。*
> *2008年，全球量化基金数量更是达到了1184 只，管理的总资金高达148 亿美元，金融风暴中，西蒙斯年回报率更是高达80%。*
> *2009 年，量化投资基金在全球基金业份额的30%以上，其中指数类投资几乎部使用量化交易方式。*

2.0 量化投资中的简单策略

1. **SMA策略(Simple Moving Average)**

- 对收盘价格取平均
- 示例方法1:

> 1. *当股价上升并上穿X天的移动平均线时，则买入；*
> 2. *当股价下降并下穿X天的移动平均线时，则卖出。*

- 示例方法2:

> 1. *当X天的移动平均线上穿Y天的移动平均线时，则买入；*
> 2. *当X天的移动平均线下穿Y天的移动平均线时，则卖出。*

- 示例代码

2. 动量策略(Momentum Strategy)

- 基于股价动量效应的投资策略，如果某只股票在前一段时期表现较好，那么下一段时期该股票仍将有良好表现。
- 示例方法

> 1. *前1日收益率大于0时,则买入；（优化：前5日平均收益率大于0时,则买入）*
> 2. *前1日收益率小于0时，则卖出。（优化：前5日平均收益率小于0时,则卖出）*

- 示例代码

3. 均值反转策略（Mean-Revertion Strategy）

- 基于股价反转效应的投资策略，如果某只股票在前一段时期表现不好，那么下一段时期该股票将会反转，即表现良好。
- 示例方法

> 1. *前1日收益率大于0时,则卖出；（优化：计算股价与移动均价(如20天)的离差,如离差大于某一阈值（如100）,则卖出）*
> 2. *前1日收益率小于0时，则买入。（优化：计算股价与移动均价(如20天)的离差,如离差小于某一阈值（如-100）,则买入）*

- 示例代码

二、python基础教程

1.0 python 语言简介

Python是一种跨平台的计算机程序设计语言。 结合了解释性、编译性、互动性和面向对象的脚本语言。最初被设计用于编写自动化脚本(shell)，随着版本的不断更新和语言新功能的添加，越多被用于独立的、大型项目的开发。

Python是一种解释型脚本语言，可以应用于以下领域:

1. Web 和 Internet开发: Django,Flask
2. 科学计算和统计:numpy,pandas,scipy
3. 人工智能:TensorFlow
4. 桌面界面开发:PyQT,Tkinter
5. 网络爬虫:spyder,requests
6. 游戏开发：Pygame
7. 操作文件/数据库：openpyxl,Python-docx，PyMysql

2.0 python 安装和环境配置

> *安装Python*

1. Python官网：https://www.python.org/
2. Anaconda官网：http://www.anaconda.com/

> *下载IDE*

1. pycharm官网：https://www.jetbrains.com/pycharm/download/
2. jupyter Notebook：下载Anaconda后自带
3. spyder：下载Anaconda后自带

3.0 python 基础语法

1. 第一个python语句
   print('我是CSMAR')

---

2. 变量 与 数据类型

- QT = "CSMAR"　　#字符串 str
- QT2 = 123　　#整数数值 int
- QT2 = 123.01　　#浮点数值 float
- QT3 = [1,2,3]　　#列表 list
- QT4 = {'name':'我是CSMAR'}　#字典 dic
- QT5 = ("平台","CSMAR")　　#元祖 tuple
- QT6 = {'C','S','M','A','R'}　#集合 set

---

3. 运算符

- 算数运算符： 加 +; 减 -; 乘 *; 除 /;取余 % ;幂 ** ;取整数 //
- 比较运算符： 等于 == ;不等于 ！ =; 大于 >; 小于 <; 大于等于>=; 小于等于<=;
- 赋值运算符： 等于 = ； 加 += ； 减 -= ； 乘 *= ； 除 /= ;
- 逻辑运算符： and ； or ； not
- 成员运算符： in ； not in
- 身份运算符： is ； is not
- 运算符优先级：

运算符	描述
**	指数 (最高优先级)
~ + -	按位翻转, 一元加号和减号 (最后两个的方法名为 +@ 和 -@)
* / % //	乘，除，求余数和取整除
+ -	加法减法
>> <<	右移，左移运算符
&	位 'AND'
^ |	位运算符
<= < > >=	比较运算符
== !=	等于运算符
= %= /= //= -= += *= **=	赋值运算符
is is not	身份运算符
in not in	成员运算符
not and or	逻辑运算符

---

4. 条件控制

- if，elif，else
- 示例：

```
a = 10
if a <10:
 print(a)
elif a == 10:
 print ("I am CSMAR")
else:
 pass
```

---

5. 循环结构

- for循环和while循环
- 示例(for)：

```
a = 0
for a in range(0,11):
 print('我已经跑了'+str(a)+'公里了')
```

- 示例(while):

```
a = 0
while a < 10:
 a += 1
 print('我已经跑了'+str(a)+'公里了')

else:
 print('终于跑完10公里了')
```

---

6. break/continue/pass

- break语句用来终止循环语句;
- continue语句用来告诉Python跳过当前循环的剩余语句，然后继续进行下一轮循环;
- pass语句不做任何事情，一般用做占位语句。
- 示例：

```
a = 0
while a < 10:
 a += 1
 if a == 5:
    #continue
    print('我已经跑了'+str(a)+'公里了')
 else:
    print('我已经跑了'+str(a)+'公里了')
```

---

7. 函数

- 内置函数

内置函数				
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reverse()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	exec 内置表达式

- 自定义函数（示例）

```
import time
def run(m):
  t = time.strftime('%H时%M分%S秒',time.localtime(time.time()))
  #print('我在' + str(t) + '跑完'+str(m)+'公里')
  return m*2

def work(m):
  #print('每天跑步'+str(run(m))+'公里，健康工作'+str(run(m)10)+'年')
  print(('每天跑步{}公里，健康工作{}年').format(str(run(m)),str(run(m)10)))
work(1)
```

---

8. 注释符

- # ： 一般用于单行注释
- ''' ''' : 可用于多行注释
- """ """ : 可用于多行注释

三、 希施玛量化数据接口介绍

---

1. 登录登出及查询

```
# 引入所需模块
import QtAPI.QtDataAPI as api

# 登录初始化
ret, errMsg = api.QtLogin("账号", "密码")
if ret == 0:
    print('登录成功')
else:
    print('登录失败:{}'.format(errMsg))
```

```
#通过QtLogout(qtUser)登出账号
ret,errMsg = api.QtLogout("账号")
print(ret)
```

```
#获取月度计费数据
ret, errMsg, dataCols = api.GetMonthBill( "2017-01-01", "2017-07-30")
print(ret,errMsg,dataCols)
```

```
#GetDetailBill 查询数据计费明细
ret, errMsg, dataCols = api.GetDetailBill( "2017-01-01", "2017-07-30")
print(dataCols)
```

2.提取基本信息类

```
#获取证券市场上全部交易所信息
ret, errMsg, dataCols = api.GetExchanges("CHN")
print(ret,errMsg)
print(dataCols)
```

```
#通过GetSecurityInfo取证券基本信息 可以获取到 SecurityID
#GetSecurityInfo(securities, securityIDS, fields, sortDefs = [],
updateTime = "", reserve = "")
securities = ["000001.SZSE"]
fields = ["Symbol","Market", "ShortName","SecurityID"]
data_securityInfo=api.GetSecurityInfo(securities,[],fields)
print(data_securityInfo)
```

```
#通过GetFactor获取因子数据    需要权限
securities = ["000001.SZSE"]
SecurityID = []
fields = ["Symbol","QF_CCI"]
data =
api.GetFactor(securities=securities,securityIDS=SecurityID,fields=fiel
ds,dateBegin ="2018-01-01",dateEnd ="2019-01-01")
data
```

```
#通过GetTradeTypes获取交易品种信息
#data=api.GetTradeTypes(securityType, fields, date = "")
data=api.GetTradeTypes("S0107",
["TradingType","Market","VarietyID","TradingUnit","TradingMargin"],
date = "2020-07-01")
print(data))
```

```
#通过GetTradeCalendar函数获取交易所交易日历
#GetTradeCalendar(exchangeCode, dateBegin, dateEnd, fields)
exchangecode=["SZSE"]
fields=["CalendarDate", "Market", "IsOpen", "TradingType",
"IsNightTrading" ]
data_TCalendar=api.GetTradeCalendar(exchangecode,"2012-07-01","2014-
09-01",fields)
print(data_TCalendar)
```

```
#通过GetPlates获取板块信息
#GetPlates(includeHis = False, securityType = [], plateType = [],
nodeLevel = [], plateTreeID = "", reserve = "")
data_plates=api.GetPlates(includeHis=False,securityType =
["S0101"],plateType=["P4903"])
print(data_plates)
```

```
# 取相关板块信息
# GetRelatedPlates(securities, securityIDS = [], date = "", timezone =
8, reserve = "")
securities = ["000001.SZSE"]
ret, errMsg, dataCols = api.GetRelatedPlates(securities)
print(dataCols)
```

```
#通过GetPlateSymbols 取板块成分清单
# GetPlateSymbols(plateIDs, setOper, sortDefs = [], dateBegin = "2017-
01-01", dateEnd = "2017-02-01", timezone = 8, reserve = "")
plateIDs = [1006010001]   #plateID需事先通过GetPlates或GetRelatedPlates
函数获取
data_plateSymbols=api.GetPlateSymbols(plateIDs,api.ESetOper["k_SetUnio
n"],dateBegin = "2017-01-01", dateEnd = "2017-02-01")
print(data_plateSymbols)
```

```
#通过GetDataByTime 按时间区间取历史数据    不复权
securities = ["600276.SSE"]
fields = ["Symbol","Market", "TradingTime", "OP", "CP"]
timePeriods = [['2019-03-26 00:00:00.000', '2019-03-28 00:00:00.000']]
dataCols = api.GetDataByTime(securities,
[],fields,api.EQuoteType["k_Minute"],5, timePeriods)
print(dataCols)
```

```
#GetDataByCount 按指定数量取历史数据
#GetDataByCount(securities, securityIDS, fields, quoteType, interval,
datetime, dir, count, sortDefs = [], timezone = 8, priceAdj =
EPriceAdjust["k_AdjNone"], reserve = "")
securities = ["000001.SZSE", "000002.SZSE", "600000.SSE",
"600028.SSE"]
fields = ["Symbol","Market", "TradingTime", "OP", "HIP", "LOP", "CP"]
data_byCount = api.GetDataByCount(securities, [], fields,
api.EQuoteType["k_Minute"],5, "2017-03-24 00:00:00.000",
api.EDirection["k_Forward"], 100)
print(data_byCount)
```

```
#运用GetFinance取财务因子数据
#GetFinance(securities, securityIDS, fields, dateBegin, dateEnd,
rptType, trailType, dateType, sortDefs = [], timezone = 8, reserve =
"")
securities = ["000001.SZSE", "000002.SZSE", "600000.SSE",
"600028.SSE"]
fields = ["Symbol","Market", "BEPS", "TOTOR", "TOLPRO"]
data_finance = api.GetFinance(securities, [], fields, "2017-01-01",
"2017-04-01",api.EReportType["k_RptMergeCur"],
api.ETrailType["k_TrailSeason"], api.ERptDateType["k_RptDateIssue"])
print(data_finance)
```

```
#运用GetFactor取量化因子数据  需要权限
securities = ["000001.SZSE"]
fields = ["Symbol","Market", "TradingDate", "QF_NetAssetPS", "QF_PE"]
data_factor = api.GetFactor(securities, [], fields,dateBegin="2019-01-
01",dateEnd="2019-06-01")
print(data_factor)
```

```
#通过QueryTable获取历史财经数据
tableName = "BOND_BASICINFO"
fields = ["BondID "," FullName ", " UpdateID ", " UpdateTime "]
# conditions = [['BondID', api.EDataType['k_Double'],
api.ECalcSign['k_GE'], ['10000']]]
# sorts = [["BondID", api.ESortType["k_SortDesc"]]]
# page = [1, 100]
dataCols = api.QueryTable(tableName, fields)
print(dataCols)
```

3. 提取其它数据类型

```
#通过GetHisMarketInfo函数取证券历史变动信息
securities = ["600276.SSE"]
securityIDS="600276.SSE"
fields =
["Symbol","FwardFactor1","BwardFactor1","CumulateFwardFactor1","Cumula
teBwardFactor2"]
dataCols = api.GetHisMarketInfo(securities,
[],fields=fields,dateBegin="2019-03-26", dateEnd="2019-03-28")
print(dataCols)
```

```
#通过GetJointContracts：取期货主力或连续合约信息
contracts = ["ACC01", "AM02"]
dataCols = api.GetJointContracts(contracts, "2017-01-01", "2017-04-
30")
print(dataCols)
```

--

四、运用希施玛数据构建策略

```
# 1.导入所需模块
import QtAPI.QtDataAPI as api

# 登录初始化
ret, errMsg = api.QtLogin("账号", "密码")
if ret == 0:
    print('登录成功')
else:
    print('登录失败:{}'.format(errMsg))
```

```
# 2. 导入数据包和绘图包
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
mpl.rcParams['font.sans-serif'] = ['SimHei']    #解决中文汉字不显示
```

```
# 3.数据准备
#通过GetDataByTime 按时间区间取历史数据    不复权
securities = ["000300.SSE"]
fields = ["TradingDate","Symbol","OP", "CP"]
timePeriods = [['2014-01-01 00:00:00.000', '2014-12-31 00:00:00.000']]
data= api.GetDataByTime(securities=securities,securityIDS=
[],fields=fields,quoteType=api.EQuoteType["k_Day"],interval=1,
timePeriods=timePeriods)[2]
data.set_index('TradingDate', inplace = True)
data.index = pd.DatetimeIndex(data.index)                                    #
转换索引的数据格式
data.head(10)
```

```
# 4.策略开发
#计算股价的连续收益率
data['returns'] = np.log(data['CP'] / data['CP'].shift(1))    #计算股价
的连续收益率
#print(data)
#计算股票的交易信号
data['signal'] = np.sign(data['returns'].shift(1))
#计算策略的连续收益率
data['strategy'] = data['returns'] * data['signal']
data.head(10)
data['signal'].plot(figsize = (20,6),marker='o',linestyle='')
```

```
# 5.策略可视化，计算成基准和策略的累计收益率
data[['returns',
'strategy']].dropna().cumsum().apply(np.exp).plot(figsize=(20,
6),title = 'Cumulative rate of return')
```

```
# 6.策略的优化
#卖出操作
data['signal_2'] = np.where(np.logical_and(data['returns'].shift(2) <
0, data['returns'].shift(1) < 0), -1, np.nan)
#买入操作
data['signal_2'] = np.where(np.logical_and(data['returns'].shift(2) >
0, data['returns'].shift(1) > 0), 1, data['signal_2'])
#对空值向前填充
data['signal_2']=data['signal_2'].ffill().fillna(0)
#计算策略的连续收益率
data['strategy_2'] = data['returns'] * data['signal_2']
#data.head()
#data['signal_2'].plot(figsize = (20,6), marker='o', linestyle='')
```

```
# 7. 绘制对比图
plt.subplot(2, 1, 1)
plt.title('Signal')
plt.gca().axes.get_xaxis().set_visible(False)
data['signal'].plot(figsize = (20,6) ,marker='o', linestyle='')
plt.legend(loc = 'center left')
plt.subplot(2, 1, 2)
data['signal_2'].plot(figsize = (20,6),marker='o', linestyle='')
plt.legend(loc = 'center left')
```

```
# 8. 策略优化后的可视化
data[['returns',
'strategy','strategy_2']].dropna().cumsum().apply(np.exp).plot(figsize
=(20, 6))
```

五、希施玛量化平台实操

希施玛量化平台简介

> *希施玛量化平台是一款集数据分析、策略回验、投资研究、模拟交易为一体的策略研究在线平台。平台提供高速便利的云端研究环境，帮助用户快速进行策略研究和数据分析，让您的量化投资如虎添翼。*

- 量化平台以数据科学常用的编程语言Python为基础，基于交互式Notebook开发形成。平台封装了策略回测框架和相关数据API等，并提供对常用Python三方库的支持，最大程度保留了Notebook原生功能和快捷键。
- 常用功能

1. 策略回测：支持股票日频和分钟频的策略回测；
2. 投资研究：提供数据API，支持获取国内六大交易所证券行情数据、证券基础数据、证券财务数据以及我司特有的量化因子数据；
3. 数据分析：内置大量Python数据分析与图形展示模块，助您进行有效的数据研究；
4. 其它功能：模拟交易、因子研究、实盘交易等多项功能将会陆续上线中....

> *Notebook*

- 量化平台中Notebook的使用体验与Jupyter NoteBook基本一致。每个Notebook由一个或多个Cell单元格的输入与输出构成。单元格一共分为三种类型：

> 1. *代码模式：代码模式单元格可输入任意Python代码，运行单元格将执行单元格内代码，效果与Python交互式终端环境相同。一般来说，代码模式的单元格适用于数据获取、统计分析、数据研究等相关工作。*
> 2. *策略模式：策略模式单元格需要您输入符合云平台策略规范的代码。输入完毕后执行单元格，平台会自动加载策略框架，并按照设定的回测参数进行策略回测。回测过程中界面将动态生成并更新资金收益曲线。回测完成后，可以查看和下载详细的策略报告。*
> 3. *文档模式：文档模式单元格中可输入文字内容，运行单元格后，平台将会按照Markdown的语法规范对内容进行渲染和展示，有关Markdown语法的相关内*

> *容请自行查阅相关资料。文档模式单元格常用于注释说明和作为内容的辅助展示。*

- 回测初始化部分
  默认的策略模版中带有一个名为initialize的函数，该函数会在正式回测进行之前执行。作为整个回测生命周期的初始化函数，常用作基准资产、交易手续费、用户自定义变量等的设置。
- 策略主体部分
  默认的策略模版中，最重要的部分便是handle_data函数。一般来说，您可以在该函数内编写您的策略，该函数默认每个调仓周期都会调用执行一次。

双均线策略

```
import talib as ta
 # 引入python技术分析模块talib
import numpy as np
 # 引入python数据分析模块numpy

start = '2019-06-01'
 # 回测起始时间
end = '2019-08-01'
 # 回测结束时间
capital_base = 1000000
 # 回测账户资金
bench_mark = '000300.SSE'
  # 回测基准指标
freq = 'd'
 # 策略类型，'d'表示日间策略使用日线回测，'m'表示日内策略使用分钟线回测
securities = static_securities_pool(['000001.SZSE'])
 # 证券池，支持动态，静态和指定代码三种方式

# 回测初始化函数，该函数会在正式回测进行之前调用一次
def initialize(context):
    # 回测参数设置
    set_commission(PerValue(cost=0.0003))
  # 设置策略手续费模型，示例为根据成交金额收取万飞之三的费用
    slippage_model = VolumeShareSlippage(0.5,0.1)
  # 构造策略滑点模型，示例为成交比例模式，限制成交比例为0.5，价格影响因子为
0.1
    set_slippage(equities = slippage_model)
  # 设置策略滑点模型
    set_cancel_policy(EODCancel())
 # 设置自动撤单策略，示例为未结订单每日盘后自动撤销

    # 用户变量设置
    context.my_asset = '000001.SZSE'
 # 设置目标证券，可用在handle_data函数中使用

# 策略运行逻辑函数，每个单位时间(如果按天回测,则每天调用一次,如果按分钟,则每
分钟调用一次)调用一次
def handle_data(context,data):
    position = get_position(context.my_asset)
  # 获取目标证券持仓
```

```
    volume = 0 if position == None else position['volume']
# 获取目标证券持仓
    cp = data.history(context.my_asset,['close'],30,'d')['close']
 # 获取目标证券历史收盘价信息，用于计算策略信号
    ma5 = ta.MA(np.array(cp),5)
 # 计算目标证券5日均线数据
    ma13 = ta.MA(np.array(cp),13)
 # 计算目标证券13日均线数据

    can_trade = data.can_trade(context.my_asset)
# 判断标的是否可以进行交易
    if ma5[-2] < ma13[-2] and ma5[-1] > ma13[-1] and volume == 0:
 # 5日均线上穿13日均线且持仓为0，证券可交易则买入8成仓
        if can_trade:order_target_percent(context.my_asset,0.8,0)
    elif ma5[-2] > ma13[-2] and ma5[-1] < ma13[-1] and volume != 0:
 # 5日均线下穿13日均线且持仓不为0，证券可交易则卖出持仓
        if can_trade:order_target_percent(context.my_asset,0,0)
```