# 项目背景

数据统计分析对数据进行收集、整理、及分析的过程。信息化过程中，各个行业将建立起各自的数据中心，未来将基于大数据进行行业分析。

从行业角度，在证券行业通过对大数据建立数学模型发掘股票价格走势；从企业管理角度，产品或项目市场销售情况和各部门成本投入制定相应报表，从中发掘各个企业管理中的短板；从岗位角度，产品经理通过对用户行为建立起数据中心，通过对该数据中心进行分析以判断用户对产品功能的粘粘度。

大数据分析未来将在各行各业崭露头角。为对一个行业或一组数据进行数据分析，不仅仅需要行业专业及管理知识，同时需要相关的工具来帮助收集数据、整理数据。最终将数据以图形化方式展现。

Python作为广泛使用的大数据分析工具，其高效、快捷、易学深受大众的欢迎。

本项目主旨，将以股票数据为基础，学习数据分析过程，通过Python工具实现数据模型，实现机器分析将结果展示。

本项目涉及范围：

1. Python通过tushare获取股票相关数据；
2. 实现数据清理、分析、建模和图形展示。

# 需求描述

本项目最终目标：

1. 使用tushare获取相关股票数据；
2. 存储相关股票数据；
3. 通过图形化方式展现数据。

## 阶段一：实现Python获取tushare数据

1、系统通过股票代码查询获取A股行情数据

2、获取信息有：

股票代码、股票名称、日期、开票价、最高价、收盘价、最低价、成交量、涨跌幅、换手率

3、系统自动每隔3秒获取新数据

4、在终端输入命令显示交易数据信息

## 阶段二：实现程序可配置化

配置化内容包括：股票代码、刷新频率等

1. 系统读取股票代码list文件(stock.txt)

## 阶段三：数据保存至本地

1、将数据保存至mysql或excel等数据库

## 阶段四：数据图形化展现

1. 趋势图

# 功能描述

# 详细设计

## 获取交易数据信息

### 历史行情数据

获取个股历史交易数据（包括均线数据），可以通过参数设置获取日k线、周k线、月k线，以及5分钟、15分钟、30分钟和60分钟k线数据。本接口只能获取近3年的日线数据，适合搭配均线数据进行选股和分析，如果需要全部历史数据，请调用下一个接get\_h\_data()

import tushare as ts

ts.get\_hist\_data('600848') #一次性获取全部日k线数据

* **设定历史数据的时间**

ts.get\_hist\_data('600848',start='2015-01-05',end='2015-01-09')

* **其它**

ts.get\_hist\_data('600848'，ktype='W') #获取周k线数据

ts.get\_hist\_data('600848'，ktype='M') #获取月k线数据

ts.get\_hist\_data('600848'，ktype='5') #获取5分钟k线数据

ts.get\_hist\_data('600848'，ktype='15') #获取15分钟k线数据

ts.get\_hist\_data('600848'，ktype='30') #获取30分钟k线数据

ts.get\_hist\_data('600848'，ktype='60') #获取60分钟k线数据

ts.get\_hist\_data('sh'）#获取上证指数k线数据，其它参数与个股一致，下同

ts.get\_hist\_data('sz'）#获取深圳成指k线数据

ts.get\_hist\_data('hs300'）#获取沪深300指数k线数据

ts.get\_hist\_data('sz50'）#获取上证50指数k线数据

ts.get\_hist\_data('zxb'）#获取中小板指数k线数据

ts.get\_hist\_data('cyb'）#获取创业板指数k线数据

### 复权历史数据

df = ts.get\_stock\_basics()

date = df.ix['600848']['timeToMarket'] #上市日期YYYYMMDD

### 实时行情数据（所有股票）

import tushare as ts

ts.get\_today\_all()

### 获取大盘指数行情

### 获取实时分笔

* **获取单子股票实时数据**

import tushare as ts

df = ts.get\_realtime\_quotes('002410') #Single stock symbol

df[['code','name','price','bid','ask','volume','amount','time']]

* **获取多个股票代码数据**

#symbols from a list

ts.get\_realtime\_quotes(['600848','000980','000981'])

#from a Series

ts.get\_realtime\_quotes(df['code'].tail(10)) #一次获取10个股票的实时分笔数据

### 个股历史分笔数据

import tushare as ts

df = ts.get\_tick\_data('600848',date='2014-01-09')

df.head(10)

### 当日历史分笔

## 图形化显示cmd

import os

os.system('cls') #清除屏幕数据

## 系统每隔3秒自动刷新数据

import time

time.sleep(3)

# 系统功能清单（checklist)

1）系统读取配置文件

配置文件包括内容：股票代码、刷新数据间隔

2）

# 知识点

## 数据分析

### 《深入浅出-数据分析》

#### 分解数据

**确定->分解->评估->决策**

1. ***确定问题及目标***

心智模型？大家对问题及事物的认知。

你对外界的假设和你确信的观点就是你的心智模型；

心智模型应当包括你不了解的因素；

1. ***对所收集的问题进行分类、分组***

比较因子、基准假设

对大问题进行分解及量化

1. ***评估数据***

通过“比较”对数据及假设进行评估总结，回答之前提出的问题。

1. ***决策***

将评估结果形成方案。

## Python

### 环境配置

使用NotePad++：

* “运行”增加Python运行快捷

cmd /k C:\Users\Liang.Lu\Desktop\Python\Anaconda "$(FULL\_CURRENT\_PATH)" & PAUSE & EXIT

* Python 字体颜色

### 基础语法

#### 基本概念

#### 运算符与表达式

#### 控制流

#### 函数

#### 模块

模块就是一个包含了所有你定义的函数和变量的文件，模块必须以.py为扩展名。模块可以从其他程序中‘输入’(import)以便利用它的功能。

在python程序中导入其他模块使用'import',所导入的模块必须在sys.path所列的目录中，因为sys.path第一个字符串是空串''即当前目录，所以程序中可导入当前目录的模块。

***1. 字节编译的.pyc文件***

导入模块比较费时，python做了优化，以便导入模块更快些。一种方法是创建字节编译的文件，这些文件以.pyc为扩展名。

pyc是一种二进制文件，是py文件经编译后产生的一种byte code，而且是跨平台的（平台无关）字节码，是有python虚拟机执行的，类似于java或.net虚拟机的概念。pyc的内容，是跟python的版本相关的，不同版本编译后的pyc文件是不同的。

***2. from .. import***

如果想直接使用其他模块的变量或其他，而不加'模块名+.'前缀，可以使用from .. import。例如想直接使用sys的argv，from sys import argv 或 from sys import \*

***3. 模块的\_\_name\_\_***

每个模块都有一个名称，py文件对应模块名默认为py文件名，也可在py文件中为\_\_name\_\_赋值；如果是\_\_name\_\_，说明这个模块被用户单独运行。

***4. dir()函数***

dir(sys)返回sys模块的名称列表；如果不提供参数，即dir()，则返回当前模块中定义名称列表。

del -> 删除一个变量/名称，del之后，该变量就不能再使用。

#### 数据结构

##### 列表

list是处理**一组有序**项目的数据结构，列表是可变的数据结构。列表的项目包含在方括号[]中，eg: [1, 2, 3]， 空列表[]。

判断列表中是否包含某项可以使用in， 比如 l = [1, 2, 3]; print 1 in l; #True；

支持索引和切片操作；索引时若超出范围，则IndexError；

使用函数len()查看长度；

使用del可以删除列表中的项，eg: del l[0] # 如果超出范围，则IndexError。

##### 元组

tuple和list十分相似，但是***tuple是不可变的，即不能修改tuple***，元组通过***圆括号中用逗号分割***的项定义；支持索引和切片操作；可以使用 in

查看一个元素是否在tuple中。空元组()；只含有一个元素的元组("a",) #需要加个逗号

**优点**：tuple比list速度快；对不需要修改的数据进行‘写保护’，可以是代码更安全

　 tuple与list可以相互转换，使用内置的函数list()和tuple()。

##### 字典

字典由***键值对组成，键必须是唯一***的；eg: d = {key1:value1, key2:value2}；***空字典用{}***表示；

字典中的键值对是没有顺序的，如果想要一个特定的顺序，那么使用前需要对它们排序；

d[key] = value，如果字典中已有key，则为其赋值为value，否则添加新的键值对key/value；

使用del d[key] 可以删除键值对；判断字典中是否有某键，可以使用in 或 not in.

##### 序列

**序列类型**是指容器内的元素从0开始的索引顺序访问，一次可以访问一个或者多个元素；列表、元组和字符串都是序列；序列的两个主要特点是索引操作符和切片操作符；***索引可以得到特定元素；切片可以得到部分序列***；

切片操作符中的第一个数（冒号之前）表示切片开始的位置，第二个数（冒号之后）表示切片到哪里结束。 如果不指定第一个数，Python就从序列首开始。如果没有指定第二个数，则Python会停止在序列尾。 注意，返回的序列从开始位置 开始 ，刚好在结束位置之前 结束。即开始位置是包含在序列切片中的，而结束位置被排斥在切片外。 可以用负数做切片。负数用在从序列尾开始计算的位置。

##### 绑定

当创建一个对象并给它赋一个变量的时候，这个变量仅仅“指向”那个对象，而不是表示这个对象本身！ 也就是说，***变量名指向计算机中存储那个对象的内存***，这被称作名称到***对象的绑定***。

如果要复制一个列表或者类似的序列或者其他复杂的对象（不是如整数那样的简单对象），那么必须使用切片操作符来取得拷贝。

#### 面向对象编程

　　python支持面向对象编程；类和对象是面向对象编程的两个主要方面，类创建一个新的类型，对象是这个类的实例。

　　对象可以使用普通的属于对象的变量存储数据，属于对象或类的变量被称为***域***；对象也可以使用属于类的函数，这样的函数称为类的方法；域和方法可以合称为类的属性。

　　域有两种类型--属于实例的或属于类本身；它们分别被称为实例变量和类变量。

　　类使用关键字class创建，类的域和方法被列在一个缩进块中。

　　类的方法必须有一个额外的第一个参数，但是在调用时不为这个参数赋值，这个特殊变量指对象本身，按照惯例它的名称是self，类似C#中的this。

class Animal:

pass #empty block

　　\_\_init\_\_方法　　在类的一个对象被创建时调用该方法；相当于c++中的构造函数。

　　\_\_del\_\_方法　　在类的对象被销毁时调用该方法；相当于c++中的析构函数。在使用del删除一个对象时也就调用\_\_del\_\_方法。

　　Python中所有的类成员(包括数据成员)都是public的；只有一个例外，如果使用的数据成员以双下划线为前缀，则为私有变量。

class Person:

Count = 0

def \_\_init\_\_(self, name, age):

Person.Count += 1

self.name = name

self.\_\_age = age

p = Person("peter", 25)

p1 = Person("john", 20)

print Person.Count #2

print p.name #peter

print p.\_\_age #AttributeError: Person instance has no attribute '\_\_age'

**继承**：为了使用继承，基类的名称作为一个元组跟在类名称的后面；python支持多重继承。下面是一个关于继承的例子：

class SchoolMember:

'''Represent any school member.'''

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

print "Initializing a school member."

def tell(self):

'''Tell my details'''

print "Name: %s, Age: %s, " % (self.name, self.age),

class Teacher(SchoolMember):

'''Represent a teacher.'''

def \_\_init\_\_(self, name, age, salary):

SchoolMember.\_\_init\_\_(self, name, age)

self.salary = salary

print "Initializing a teacher"

def tell(self):

SchoolMember.tell(self)

print "Salary: %d" % self.salary

class Student(SchoolMember):

'''Represent a student.'''

def \_\_init\_\_(self, name, age, marks):

SchoolMember.\_\_init\_\_(self, name, age)

self.marks = marks

print "Initializing a student"

def tell(self):

SchoolMember.tell(self)

print "Marks: %d" % self.marks

print SchoolMember.\_\_doc\_\_

print Teacher.\_\_doc\_\_

print Student.\_\_doc\_\_

t = Teacher("Mr. Li", 30, 9000)

s = Student("Peter", 25, 90)

members = [t, s]

for m in members:

m.tell()

程序输出如下：

Represent any school member.

Represent a teacher.

Represent a student.

Initializing a school member.

Initializing a teacher

Initializing a school member.

Initializing a student

Name: Mr. Li, Age: 30, Salary: 9000

Name: Peter, Age: 25, Marks: 90

#### 输入/输出

程序与用户的交互需要使用输入/输出，主要包括控制台和文件；对于控制台可以使用raw\_input和print，也可使用str类。raw\_input(xxx)输入xxx然后读取用户的输入并返回。

**1. 文件输入/输出**

　　　　可以使用file类打开一个文件，使用file的read、readline和write来恰当的读写文件。对文件读写能力取决于打开文件时使用的模式，常用模式

　　有读模式("r")、写模式("w")、追加模式("a")，文件操作之后需要调用close方法来关闭文件。

test = '''\

This is a program about file I/O.

Author: Peter Zhange

Date: 2011/12/25

'''

f = file("test.txt", "w") # open for writing, the file will be created if the file doesn't exist

f.write(test) # write text to file

f.close() # close the file

f = file("test.txt") # if no mode is specified, the default mode is readonly.

while True:

line = f.readline()

if len(line) == 0: # zero length indicates the EOF of the file

break

print line,

f.close()

**2. 存储器**

　　　　python提供一个标准的模块，成为pickle，使用它可以在一个文件中存储任何python对象，之后可以完整的取出来，这被称为持久地存储对象；还有另外一个模块成为cPickle，它的功能和pickle完全一样，只不过它是用c写的，要比pickle速度快(大约快1000倍)。

复制代码

import cPickle

datafile = "data.data"

namelist = ["peter", "john", "king"]

f = file(datafile, "w")

cPickle.dump(namelist, f)

f.close()

del namelist

f = file(datafile)

storednamelist = cPickle.load(f)

print storednamelist

#['peter', 'john', 'king']

#### 异常

* 当程序中出现某些异常的状况时，异常就发生了。python中可以使用try ... except 处理。
* 可以让try ... except 关联上一个else，当没有异常时则执行else。
* 我们可以定义自己的异常类，需要继承Error或Exception。
* try...finally

try:

f = file("test.txt")

while True:

line = f.readline()

if len(line) == 0:

break

time.sleep(2)

print line,

finally:

f.close()

print "Cleaning up..."

#### Python标准库

* **sys模块　　sys模块包含系统对应的功能**

sys.argv　　---包含命令行参数，第一个参数是py的文件名

sys.platform　　---返回平台类型

sys.exit([status])　　---退出程序，可选的status(范围：0-127)：0表示正常退出，其他表示不正常，可抛异常事件供捕获

sys.path　　　　---程序中导入模块对应的文件必须放在sys.path包含的目录中，使用sys.path.append添加自己的模块路径

sys.modules　　---This is a dictionary that maps module names to modules which have already been loaded

sys.stdin,sys.stdout,sys.stderr　　---包含与标准I/O 流对应的流对象

* **os模块　　该模块包含普遍的操作系统功能**

os.name字符串指示你正在使用的平台。比如对于Windows，它是'nt'，而对于Linux/Unix用户，它是'posix'

os.getcwd()函数得到当前工作目录，即当前Python脚本工作的目录路径

os.getenv()和os.putenv()函数分别用来读取和设置环境变量

os.listdir()返回指定目录下的所有文件和目录名

os.remove()函数用来删除一个文件

os.system()函数用来运行shell命令

os.linesep字符串给出当前平台使用的行终止符。例如，Windows使用'\r\n'，Linux使用'\n'而Mac使用'\r'

os.sep 操作系统特定的路径分割符

os.path.split()函数返回一个路径的目录名和文件名

os.path.isfile()和os.path.isdir()函数分别检验给出的路径是一个文件还是目录

os.path.existe()函数用来检验给出的路径是否真地存在

#### 其它

**1. 一些特殊的方法**

\_\_init\_\_(self,...) 这个方法在新建对象恰好要被返回使用之前被调用。

\_\_del\_\_(self) 恰好在对象要被删除之前调用。

\_\_str\_\_(self) 在我们对对象使用print语句或是使用str()的时候调用。

\_\_lt\_\_(self,other) 当使用 小于 运算符（<）的时候调用。类似地，对于所有的运算符（+，>等等）都有特殊的方法。

\_\_getitem\_\_(self,key) 使用x[key]索引操作符的时候调用。

\_\_len\_\_(self) 对序列对象使用内建的len()函数的时候调用。

**2. 综合列表**

通过列表综合，可以从一个已有的列表导出一个新的列表

**3. 函数接收元组/列表/字典**

当函数接收元组或字典形式的参数的时候，有一种特殊的方法，使用\*和\*\*前缀。该方法在函数需要获取可变数量的参数的时候特别有用。

由于在args变量前有\*前缀，所有多余的函数参数都会作为一个元组存储在args中。如果使用的是\*\*前缀，多余的参数则会被认为是一个字典的键/值对。

def powersum(power, \*args):

total = 0

for i in args:

total += pow(i, power)

return total

print powersum(2, 1, 2, 3) #14

def displaydic(\*\*args):

for key,value in args.items():

print "key:%s;value:%s" % (key, value)

displaydic(a="one", b="two", c="three")

#key:a;value:one

#key:c;value:three

#key:b;value:two

**4. lambda**

lambda语句被用来创建新的函数对象，并在运行时返回它们。lambda需要一个参数，后面仅跟单个表达式作为函数体，而表达式的值被这个

新建的函数返回。 注意，即便是print语句也不能用在lambda形式中，只能使用表达式。

**5. exec/eval**

exec语句用来执行储存在字符串或文件中的Python语句；eval语句用来计算存储在字符串中的有效Python表达式。

**6. assert**

assert语句用来断言某个条件是真的，并且在它非真的时候引发一个错误--AssertionError。

**7. repr函数**

　repr函数用来取得对象的规范字符串表示。反引号（也称转换符）可以完成相同的功能。

　注意，在大多数时候有eval(repr(object)) == object。

可以通过定义类的\_\_repr\_\_方法来控制对象在被repr函数调用的时候返回的内容。

## Pandas

***#Pandas其能做什么；#在数据分析中运用场景***

## NumPy

***#NumPy其能做什么；#在数据分析中运用场景***

## TuShare

# 程序化交易平台分析

## 竞品

<http://vnpy.org/>