**以下内容属于numpy+pndas项目练习.pdf中的纯代码，请参考pdf文件理解**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# 读取文件

luohu\_data = pd.read\_csv('./bj\_luohu.csv', index\_col = 'id')

# describe()展示一些基本信息

luohu\_data.describe()

# 按照company分组并计算每组个数

# groupby默认会把by的这个列作为索引列返回，可以设置下as\_index=False

company\_data = luohu\_data.groupby('company', as\_index=False).count()[['company','name']]

# 重命名列名称

company\_data.rename(columns={'name':'people\_count'}, inplace=True)

# 按照某一列排序

company\_sorted\_data = company\_data.sort\_values('people\_count', ascending=False)

company\_sorted\_data

# 按条件过滤

# 只有一人的公司

company\_sorted\_data[company\_sorted\_data['people\_count'] == 1]

# 人数前50的公司

company\_sorted\_data.head(50)

# 分数分布

# 按照步长5分桶统计下分数的分布

bins = np.arange(90, 130, 5)

bins = pd.cut(luohu\_data['score'], bins)

bin\_counts = luohu\_data['score'].groupby(bins).count()

# 处理index

bin\_counts.index = [ str(x.left) + '~' + str(x.right) for x in bin\_counts.index ]

bin\_counts.plot(kind='bar', alpha=1, rot=0)

plt.show()

# 年龄分布

# 出生日期转为年龄

luohu\_data['age'] = ((pd.to\_datetime('2019-07') - pd.to\_datetime(luohu\_data['birthday'])) / pd.Timedelta('365 days'))

luohu\_data.describe()

bins = np.arange(20, 70, 5)

bins = pd.cut(luohu\_data['age'], bins)

bin\_counts = luohu\_data['age'].groupby(bins).count()

bin\_counts.index = [ str(x.left) + '~' + str(x.right) for x in bin\_counts.index ]

bin\_counts.plot(kind='bar', alpha=1, rot=0)

plt.show()

# 阿里股票历史数据下载：https://www.nasdaq.com/symbol/baba/historical

# 也可以抓取雪球等股票app的数据

# 阿里股票走势图：https://xueqiu.com/S/BABA

# 道琼斯走势：<https://xueqiu.com/S/.DJI>

import numpy as np

from dateutil.parser import parse

# 指定打开的文件名

# 不需要的行需要skip掉

# 默认没有分隔符，所以需要指定delimiter

# 不加载全部的情况下需要指定加载哪些列usecols

# 希望把每一列加载到单独的数组中需要设置unpack=True，并指定对应的变量名，列举下有unpack和没有的区别

# stock\_info = np.loadtxt('./BABA\_stock.csv', delimiter=',', usecols=(1, 2, 3, 4, 5), skiprows=1)

stock\_info = np.loadtxt('./BABA\_stock.csv', delimiter=',', usecols=(1, 2, 3, 4, 5), skiprows=1, unpack=True)

stock\_info.shape

stock\_info = stock\_info[:, ::-1]

stock\_info

# 上涨下跌的天数

close\_info = stock\_info[0]

open\_info = stock\_info[2]

# 上涨

rise\_count = len(close\_info[open\_info - close\_info > 0])

# rise\_count = close\_info[open\_info - close\_info > 0].size

print('rise count:' + str(rise\_count))

# 下跌

fail\_count = len(close\_info[open\_info - close\_info <= 0])

print('fail count:' + str(fail\_count))

# 上张天数占比

rise\_count / close\_info.size

close\_info

# 日线转换为周线

# 什么是周线

high\_info = stock\_info[3]

low\_info = stock\_info[4]

# loadtxt方法有一个converters参数，可以利用自定义的函数把string做转换

def convert\_date(d):

return parse(d).weekday()

stock\_info = np.loadtxt('./BABA\_stock.csv', delimiter=',', usecols=(0, 1, 3, 4, 5), skiprows=1, dtype='S', converters={0: convert\_date})

#print(stock\_info)

# 倒序排列

stock\_info = stock\_info[::-1, :].astype('f8')

# 需要按照每周去分组

# 先找到星期一的数据的索引

week\_split = np.where(stock\_info[:, 0] == 0)[0]

# 按照周一去分组，split返回给定索引的分组

# 可以指定任意间隔的索引，所以split以一个list的形式返回

week\_infos\_temp = np.split(stock\_info, week\_split)

print(type(week\_infos\_temp))

# 为了简单起见，我们这里只使用一周数据有五天的，

week\_infos\_temp

week\_info = [ x for x in week\_infos\_temp if len(x) == 5 ]

# 每个星期的数据都是一样的了，我们在把他转换成ndarray

w = np.array(week\_info)

print(w.shape)

print(w[:3])

week\_open = w[:, 0, 3]

print(week\_open[:3])

week\_close = w[:, -1, 1]

print(week\_close[:3])

week\_high = w[:, :, 3].max(axis=1)

print(week\_high[:3])

week\_low = w[:, :, 4].min(axis=1)

print(week\_low[:3])

w\_info = np.array([week\_open, week\_close, week\_high, week\_close])

#print(w\_info[:3])

# 一周的数据放到一行，可以直接用转置矩阵

print('result:', w\_info.T)

# 结果保存到文件

np.savetxt('./week\_info\_baba.cvs', w\_info.T, header='open, close, high, low', delimiter=',', fmt='%.2f')

# 加载数据，把date这一列设置为索引，简单起见，只用收盘价进行分析

import numpy as np

import pandas as pd

df = pd.read\_csv('./BABA\_stock.csv', index\_col='date', usecols=[0, 1])

# 先查看以下数据

df.head()

# 将索引转换为datetime形式

df.index = pd.DatetimeIndex(df.index.str.strip("'"))

df.index

# 数据中最近的日期排在前面，按照日期重新排序

df.sort\_index(inplace=True)

print(df.head())

df.describe()

# 策略一：股价超出10日均线买入，跌破十日均线卖出

# 先计算10日均线数据

ma10 = df.rolling(10).mean().dropna()

# 买点

ma10\_model = df['close'] - ma10['close'] > 0

ma10\_model

# 第一个值False，第二个值是True，在True的时候买入，需要自定义一个移动窗口处理函数

# 因为卖的时候还需要定义类似的函数，所以这里把这两个函数放在一起

# 可以在自定义函数中print一些信息，例如w值，以方便调试---这也是调试的一种方式

def get\_deal\_date(w, is\_buy=True):

if is\_buy == True:

return True if w[0] == False and w[1] == True else False

else :

return True if w[0] == True and w[1] == False else False

# raw=False没有的话会有警告信息

# 如果删除Na值，会有缺失，所以这里用0填充，转换为bool值方便后面取值

se\_buy = ma10\_model.rolling(2).apply(get\_deal\_date, raw=False).fillna(0).astype('bool')

# apply的args接受数组或者字典给自定义函数传参

se\_sale = ma10\_model.rolling(2).apply(get\_deal\_date, raw=False, args=[False]).fillna(0).astype('bool')

# 具体的买卖点

buy\_info = df[se\_buy.values]

sale\_info = df[se\_sale.values]

# print(buy\_info.head())

# print(sale\_info.head())

# 买和卖的索引值不一样，不过数据都有63条，所以删除时间索引信息

no\_index\_buy\_info = buy\_info.reset\_index(drop=True)

no\_index\_sale\_info = sale\_info.reset\_index(drop=True)

# print(no\_index\_buy\_info.head())

# print(no\_index\_sale\_info.head())

# 每次交易的盈利情况

profit = no\_index\_sale\_info - no\_index\_buy\_info

print(profit)

# 大体数据

profit.describe()

# 总利润是36.07，注意这是每次买和卖一股的利润（买固定的股数），三年的时间交易了63次

# 最多投入210.86，平均投入143.366954，按最高投入算利润率(36.07 / 210.860000)，年化差不多5%，按平均投入算0.2515，年化将近8%，当然还有手续费没有算

profit.sum()

# 假如有1w美元，最终盈利是多少

all\_money = 10000

remain = all\_money

for i in range(len(no\_index\_buy\_info)):

buy\_count = remain / no\_index\_buy\_info.iloc[i]

remain = buy\_count \* no\_index\_sale\_info.iloc[i]

# all\_money = sale\_total - all\_money

print(remain)

# 最后剩下13799.294014，年化10%多点，还不错

# 如果加上每次交易金额的万分之三手续费

all\_money = 10000

remain = all\_money

fee = 0.0003

for i in range(len(no\_index\_buy\_info)):

buy\_count = remain / no\_index\_buy\_info.iloc[i]

remain = buy\_count \* no\_index\_sale\_info.iloc[i] \* (1 - fee)

# all\_money = sale\_total - all\_money

print(remain)

# 最终金额13540.898129，少了一点，不过也还不错

# 将10日线改为60日线试试

# 考虑一种边界情况，买了之后卖点到现在还没有出现（因为是先买的，所以不可能买点比卖点多）

# 这种情况我们把当前的股价加到卖点数据中（也可以把买点数据删除）

print(len(no\_index\_buy\_info))

print(len(no\_index\_sale\_info))

if (len(no\_index\_buy\_info) > len(no\_index\_sale\_info)):

# no\_index\_buy\_info.drop(no\_index\_buy\_info.index[-1], inplace=True)

no\_index\_sale\_info.loc[len(no\_index\_sale\_info)] = [df.iloc[-1, 0]]

# 在看下250天，也是盈利的，不是策略多牛逼，而是这个统计区间是在美股大的上升趋势中

# 从五日线一直到250日线，都回测下，然后找出最高的

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# 加载文件

# 这次只分析'App', 'Category', 'Rating', 'Reviews', 'Size', 'Installs', 'Type'

df = pd.read\_csv('./googleplaystore.csv', usecols=(0, 1, 2, 3, 4, 5, 6))

# 简单浏览下数据

df.head()

# 查看行列数量

df.shape

# 查看各个列的非空数据量

df.count()

# 有很多缺失值，需要清洗

# App处理

# 查看有没有重复值

# 也可以：df['App'].value\_counts()

pd.unique(df['App']).size

# 有重复值，先不着急删除重复值，为了不把其他列的异常值留下，先处理数值异常的列

# Category处理

df['Category'].value\_counts(dropna=False)

# 有一条异常值

df[df['Category'] == '1.9']

# Rating处理

df['Rating'].value\_counts(dropna=False)

# 用平均值填充

df['Rating'].fillna(value=df['Rating'].mean(), inplace=True)

# 有一条值是19的异常记录，和Category的异常是同一条记录

# Reviews清洗

# 用value\_counts看数据分布挺广，看起来都是数字

df['Reviews'].value\_counts(dropna=False)

df['Reviews'].str.isnumeric().sum()

# 查看有问题的那一行数据

df[~df['Reviews'].str.isnumeric()]

# 异常值和其他的一样，删除这条记录

df.drop(index=10472, inplace=True)

df['Reviews'] = df['Reviews'].astype('i8')

# Size的清洗处理

df['Size'].value\_counts()

df['Size'] = df['Size'].str.replace('M', 'e+6')

df['Size'] = df['Size'].str.replace('k', 'e+3')

# 尝试转换，此时转换报错，还有字符串

# df['Size'].astype('f8')

# 定义一个字符串判断是否可以转换

def is\_convertable(v):

try:

float(v)

return True

except ValueError:

return False

# 查看不能转换的字符串分布

temp = df['Size'].apply(is\_convertable)

df['Size'][~temp].value\_counts()

# 转换剩下的字符串

df['Size'] = df['Size'].str.replace('Varies with device', '0')

# 在看下是不是还有没转换的字符串

temp = df['Size'].apply(is\_convertable)

df['Size'][~temp].value\_counts()

# 转换类型

# e+5这种格式使用astype直接转为int有问题，如果想转成int，可以先转成f8，再转i8

# df['Size'] = df['Size'].astype('f8').astype('i8')

df['Size'] = df['Size'].astype('f8')

# 将Size为0的填充为平均数

df['Size'].replace(0, df['Size'].mean(), inplace=True)

df.describe()

# Installs数据清洗

# 先查看分布

df['Installs'].value\_counts()

# 分布比较少，直接替换

df['Installs'] = df['Installs'].str.replace('+', '')

df['Installs'] = df['Installs'].str.replace(',', '')

# 转换

df['Installs'] = df['Installs'].astype('i8')

df.describe()

# Type处理

# df.info()查看到有na值，这里需要dropna参数

df['Type'].value\_counts(dropna=False)

df[df['Type'].isnull()]

# 删除这条数据

df.drop(index=9148, inplace=True)

# 删除App重复的行

df.drop\_duplicates('App', inplace=True)

# 数据清洗完毕，可以开始分析了

# 整体情况

df.describe()

# 分Category的数据

# 分类的个数

df.Category.unique().size

# 每个分类的App数量，排序，可以得出哪些分类的app最受开发者欢迎

df.groupby('Category').count().sort\_values('App', ascending=False)

# 分类的安装量排序：娱乐社交类最被用户所需要

df.groupby('Category').mean().sort\_values('Installs', ascending=False)

# 分类的评论数据：社交游戏视频评论多

df.groupby('Category').mean().sort\_values('Reviews', ascending=False)

# 分类的打分数据，和其他数据不太一致，需要进一步分析

df.groupby('Category').mean().sort\_values('Rating', ascending=False)

# 分Type数据

# 免费占比大，付费占比小，免费仍然是主流

df.groupby('Type').count()

# 只有两个类型，且数据量差别很大，没必要继续对比了

df.groupby('Type').sum().sort\_values('Installs', ascending=False)

# Category和Type一起分析

# 可以和上面一样分析，不多说了

df.groupby(['Type', 'Category']).mean().sort\_values('Reviews', ascending=False)

# 评论安装比

# 收费的app评论比率更高

g = df.groupby(['Type', 'Category']).mean()

(g['Reviews'] / g['Installs']).sort\_values(ascending=False)

# 相关性：评论数和安装数强相关，其他的连0.1都不到，可以认为是不相关的（0.5以上可以认为是相关的，0.3以上可以认为是弱相关）

df.corr()

# 加载数据分析需要使用的库

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# 加载数据，加载之前先用文本编辑器看下数据的格式，首行是什么，分隔符是什么等

df = pd.read\_csv('./order\_info\_2016.csv', index\_col='id')

df.head()

# 加载好数据之后，第一步先分别使用describe和info方法看下数据的大概分布

# 这两个方法放到两个cell中

df.describe()

# 加载device\_type

device\_type = pd.read\_csv('./device\_type.txt')

device\_type

df.info()

# 首先要做一个数据的清洗

# order\_id

# 我们都知道order\_id在一个系统里是唯一值

# 先看下有没有重复值

# 注意：当我们对一列取size属性的时候，返回的是行数，如果对于dataframe使用size，返回的是行乘以列的结果，也就是总的元素数

df.orderId.unique().size

df.orderId.size

# 如果有重复值，我们一般最后处理，因为其他的列可能会影响到删除哪一条重复的记录

# 先处理其他的列

# userId

# userId我们只要从上面的describe和info看下值是不是在正常范围就行了

# 对于订单数据，一个用户有可能有多个订单，重复值是合理的

df.userId.unique().size

# productId

# productId最小值是0，先来看下值为0的记录数量

df.productId[(df.productId == 0)].size

# 177条记录，数量不多，可能是因为商品的上架下架引起的，处理完其他值的时候我们把这些删掉

# cityId

# cityId类似于userId，值都在正常范围，不需要处理

df.cityId.unique().size

# price

# price没有空值，且都大于0，注意单位是分，我们把它变成元

df.price = df.price / 100

# payMoney

# payMoney有负值，我们下单不可能是负值，所以这里对于负值的记录要删除掉

# 展示负值的记录

df[df.payMoney < 0]

# 删除负值的记录

df.drop(index=df[df.payMoney < 0].index, inplace=True)

# 在看下，已经没有了

df[df.payMoney < 0].index

# 变成元

df.payMoney = df.payMoney / 100

# channelId

# channelId根据info的结果，有些null的数据，可能是端的bug等原因，在下单的时候没有传channelId字段

# 数据量大的时候，删掉少量的null记录不会影响统计结果，这里我们直接删除

# 展示

df[df.channelId.isnull()]

# 删除

df.drop(index=df[df.channelId.isnull()].index, inplace=True)

# 在查看

df[df.channelId.isnull()]

# deviceType的取值可以看device\_type.txt文件，没有问题，不需要处理

# createTime和payTime都没有null，不过我们是要统计2016年的数据，所以把非2016年的删掉

# payTime类似，这里只按创建订单的时间算，就不处理了

# 先把createTime和payTime转换成datetime格式

df.createTime = pd.to\_datetime(df.createTime)

df.payTime = pd.to\_datetime(df.payTime)

df.dtypes

import datetime

startTime = datetime.datetime(2016, 1, 1)

endTime = datetime.datetime(2016, 12, 31, 23, 59, 59)

# 有16年之前的数据，需要删掉

df[df.createTime < startTime]

df.drop(index=df[df.createTime < startTime].index, inplace=True)

df[df.createTime < startTime]

# payTime早于createTime的也需要删掉

df.drop(index=df[df.createTime > df.payTime].index, inplace=True)

# 处理16年之后的数据

df[df.createTime > endTime]

# 看下支付时间有没有16年以前的，支付时间在16年之后的这里就不处理了

df[df.payTime < startTime]

# 回过头来我们把orderId重复的记录删掉

df.orderId.unique().size

df.orderId.size

df.drop(index=df[df.orderId.duplicated()].index, inplace=True)

df.orderId.unique().size

# 把productId为0的也删除掉

df.drop(index=df[df.productId==0].index, inplace=True)

# 数据清洗完毕，可以开始分析了

# 一般都是先看下数据的总体情况总体情况

# 总订单数，总下单用户，总销售额，有流水的商品数

print(df.orderId.count())

print(df.userId.unique().size)

print(df.payMoney.sum()/100)

print(df.productId.unique().size)

# 分析数据可以从两方面开始考虑，一个是维度，一个是指标，维度可以看做x轴，指标可以看成是y轴，同一个维度可以分析多个指标，同一个维度也可以做降维升维。

# 按照商品的productId

# 先看下商品销量的前十和后十个

productId\_orderCount = df.groupby('productId').count()['orderId'].sort\_values(ascending=False)

print(productId\_orderCount.head(10))

print(productId\_orderCount.tail(10))

# 销售额

productId\_turnover = df.groupby('productId').sum()['payMoney'].sort\_values(ascending=False)

print(productId\_turnover.head(10))

print(productId\_turnover.tail(10))

# 看下销量和销售额最后100个的交集，如果销量和销售额都不行，这些商品需要看看是不是要优化或者下架

problem\_productIds = productId\_turnover.tail(100).index.intersection(productId\_orderCount.tail(100).index)

# 城市的分析可以和商品维度类似

cityId\_orderCount = df.groupby('cityId').count()['orderId'].sort\_values(ascending=False)

cityId\_payMoney = df.groupby('cityId').sum()['payMoney'].sort\_values(ascending=False)

# price

# 对于价格，可以看下所有商品价格的分布，这样可以知道什么价格的商品卖的最好

# 先按照100的区间取分桶，价格是分，这里为了好看把他转成元

bins = np.arange(0, 25000, 100)

pd.cut(df.price, bins).value\_counts()

# 直方图

# 觉得尺寸小的话可以先设置下figsize，觉得后面的值没有必要展示，可以不用25000，改成10000：plt.figure(figsize=(16, 16))

plt.hist(df['price'], bins)

# 很多价格区间没有商品，如果有竞争对手的数据，可以看看是否需要补商品填充对应的价格区间

price\_cut\_count = pd.cut(df.price, bins).value\_counts()

zero\_cut\_result = (price\_cut\_count == 0)

zero\_cut\_result[zero\_cut\_result.values].index

# 按照1000分桶在看下

bins = np.arange(0, 25000, 1000)

price\_cut = pd.cut(df.price, bins).value\_counts()

# 看看1000分桶的时候5000以下的饼图

m = plt.pie(x=price\_cut.values, labels=price\_cut.index, autopct='%d%%', shadow=True)

# channelId

# 渠道的分析类似于productId，可以给出成交量最多的渠道，订单数最多的渠道等，渠道很多时候是需要花钱买流量的，所以还需要根据渠道的盈利情况和渠道成本进行综合比较，同时也可以渠道和商品等多个维度综合分析，看看不同的卖的最好的商品是否相同

# 下单时间分析

# 按小时的下单量分布，可以按时间做推广

# 中午12， 13， 14点下单比较多，应该是午休的时候，然后是晚上20点左右，晚上20点左右几乎是所有互联网产品的一个高峰，下单高峰要注意网站的稳定性、可用性

df['orderHour'] = df.createTime.dt.hour

df.groupby('orderHour').count()['orderId'].plot()

# 按照星期来看，周六下单最多，其次是周四周五

df['orderWeek'] = df.createTime.dt.dayofweek

df.groupby('orderWeek').count()['orderId']

# 下单后多久支付

def get\_seconds(x):

return x.total\_seconds()

df['payDelta'] = (df['payTime'] - df['createTime']).apply(get\_seconds)

bins = [0, 50, 100, 1000, 10000, 100000]

pd.cut(df.payDelta, bins).value\_counts()

# 饼图看下，有重合的话可以改下bins

# 绝大部分都在十几分钟之内支付完成，说明用户基本很少犹豫，购买的目的性很强

pd.cut(df.payDelta, bins).value\_counts().plot(kind='pie', autopct='%d%%', shadow=True, figsize=(10, 10))

# 月成交额

# 先把创建订单的时间设置为索引

df.set\_index('createTime', inplace=True)

turnover = df.resample('M').sum()['payMoney']

order\_count = df.resample('M').count()['orderId']

turnover.plot()