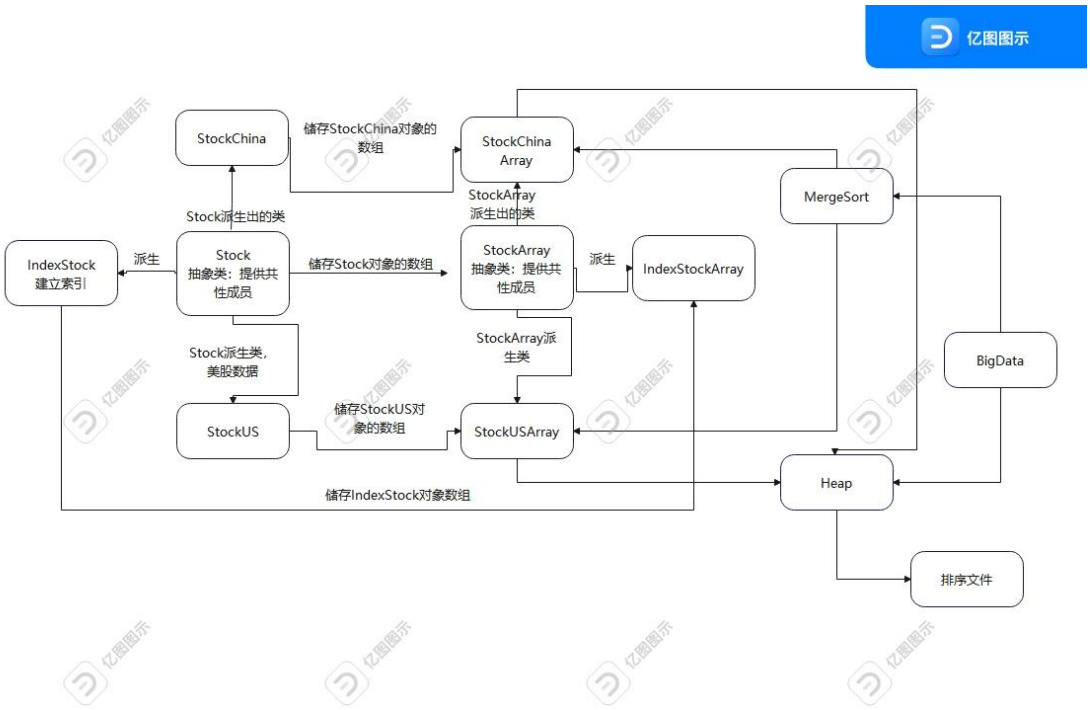
大作业报告

0 前言

根据数据处理要求，我把 6 个任务分割成了三个 Qt 项目。

1. StockDataExternalSort: 外排序和数据访问优化（建立索引）,这两个任务都要对所有股票和所有月份进行处理
2. MyStockPlotWindow 股票价格可视化和股票价格预测，对一支用户指定的股票在指定的月份的处理
3. Try: 指标计算和 K-means 聚类，对用户指定的月份内的多支股票进行处理

1 外排序和数据访问优化（建立索引）




```
#ifndef STOCK_H
#define STOCK_H

#include <iostream>
#include <string>
#include <fstream>
#include <sstream>

class Stock
{
public:
    Stock(int sid = 0) : SrcID(sid) {}
    virtual ~Stock();

    // 虚函数，用于派生类的赋值操作
```

```
16.         virtual const Stock& operator=(const Stock& st) = 0;
17.
18.         // 字符串形式
19.         virtual std::string toString() const = 0;
20.
21.         // 获得节特征值，主要用于关系比较
22.         virtual std::string sym() const = 0;
23.         virtual std::string dt() const = 0;
24.
25.         // 重载比较运算符，比较方法都一样：代码、日期，故不需要设计成虚函数在每个类里重
    载
26.         bool operator>(const Stock& st) const;
27.         bool operator>=(const Stock& st) const;
28.         bool operator==(const Stock& st) const;
29.         bool operator<=(const Stock& st) const;
30.         bool operator<(const Stock& st) const;
31.
32.         // 获取和设置 SrcID 属性
33.         int getSrcID() const;
34.         void setSrcID(int newSrcID);
35.
36.         // 友元函数，文件流输入输出
37.         friend std::ifstream& operator>>(std::ifstream& in, Stock& st);
38.         friend std::ofstream& operator<<(std::ofstream& out, const Stock& st);
39.
40.     private:
41.         // 从文件流中读入当前行数据到节点中
42.         virtual void readline(std::ifstream& in) = 0;
43.         // 将节点数据写入文件
44.         virtual void write(std::ofstream& out) const = 0;
45.
46.     protected:
47.         int SrcID; // 节点的源 ID，在 heap 中标注来自哪个小文件
48.     };
49.
50.     inline int Stock::getSrcID() const
51.     {
52.         return SrcID;
53.     }
54.
55.     inline void Stock::setSrcID(int newSrcID)
56.     {
57.         SrcID = newSrcID;
58.     }
```

```
59.
60.     #endif // STOCK_H
```

Stock 是抽象类，提供 srcID 是共有属性。其他的像股票的 symbol, datetime, open, high, low, close 等虽然也是都具有的，但是如果抽象在 Stock 类里，会对派生类调用不方便：如果设计成 public 对象，丧失封装特性；设计成 private 对象，就不能在派生类里更改。所以没有进行提炼。比较运算符的重载是按照字典序一次对 symbol 和 datetime 进行重载，比较时调用派生类要重载的纯虚函数 sym(),dt()，所以可以实现派生类之间的统一，不用设计成纯虚函数。

三个派生类 StockChina, StockUS, IndexStock 重载了纯虚函数，同时提供了私有成员变量的外部接口，不再赘述。

```
1.      #ifndef STOCKARRAY_H
2.      #define STOCKARRAY_H
3.
4.      #include "stock.h"
5.
6.      class StockArray
7.      {
8.      protected:
9.          int size; // 动态数组的大小，类族共享成员
10.
11.     public:
12.         StockArray(int size = 0) : size(size) { ; }
13.         virtual ~StockArray() { ; }
14.
15.         virtual Stock& operator[](int i) = 0; // 可以修改
16.         virtual const Stock& operator[](int i) const = 0; // 只读访问，不能修改
17.
18.         int sizeOf() const { return size; } // 获取数组大小
19.
20.         void setSize(int newSize) { size = newSize; } // 设置数组大小
21.     };
22.
23.     #endif // STOCKARRAY_H
24.
```

StockArray 也是抽象类，提供 size 作为类族共享，重载 [] 运算符使其对象可以像正常数组使用更为轻便。

1.1 外排

构建类模板 BigData<class T1, class T2>, Heap<class T1, class T2>, MergeSort<class T1>

T1 是 Array，T2 是元素，这样在进行美国和中国股市排序的时候，就只用对类模板进行实例化。

使用 BigData 可以读取部分数据进入内存进行归并排序，按需对小文件的个数或者

大小进行设定。在这个项目中，对中国股票进行排序采用了指定小文件个数的方法，对美国股票的程序采用了指定小文件大小的方法。

可以指定 ASC 进行升序排序，DES 降序，设定之后程序会自己相应地判断使用 minHeap 或者 maxHeap。

1.2 数据访问优化

创建 IndexStock 类。

```
#ifndef INDEXSTOCK_H
#define INDEXSTOCK_H

#include "stock.h"

class IndexStock : public Stock
{
public:
    IndexStock();
    IndexStock(std::string symbol, std::string month);
    IndexStock(const IndexStock& ins);
    ~IndexStock();

    const Stock& operator = (const Stock& st);
    const IndexStock& operator=(const IndexStock& stc);

    std::string toString() const;

    std::string sym() const;
    std::string dt() const;

    friend std::ifstream& operator >> (std::ifstream& in, Stock& st);
    friend std::ofstream& operator << (std::ofstream& out, Stock& st);

private:
    std::string symbol;
    std::string month;
    int position;

    void readline(std::ifstream& in);
    void write(std::ofstream& out) const;

    std::string dt2mon(char* datetime);

    void copyfrom(const IndexStock& ins);
};

```

```
38.     #endif // INDEXSTOCK_H
```

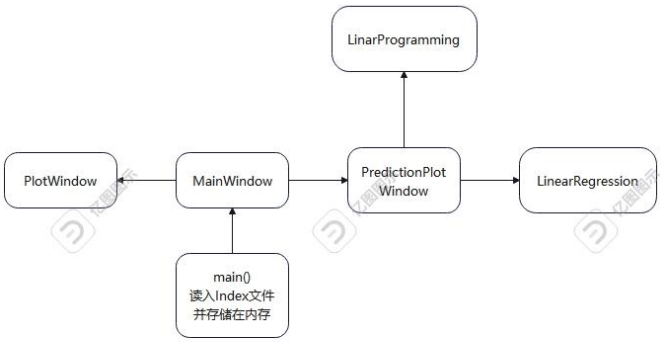
重载输入>>，获取 symbol 和 datetime，用 `std::string dt2mon(char* datetime);` 把 datetime 转化成 month，这时运用 tellg(),返回的就是该行 open 的位置。用 getline() 函数，这时读取了整行数据，不处理，程序继续。

创建 IndexStockArray 的指针 pIndexStockArray，开辟两个空间，先读入 pIndexStockArray[0]，然后继续读入下一行数据并存储在 pIndexStockArray[1]，与 pIndedxStockArray[0]进行比较，如果相同就继续，否则把 pIndexStockArray[0]写入文件，并更新 pIndexStockArray[0]的值。（运算符==是 Stock 类里的，但是重载 sym()函数返回月份信息使得可以延续使用。）

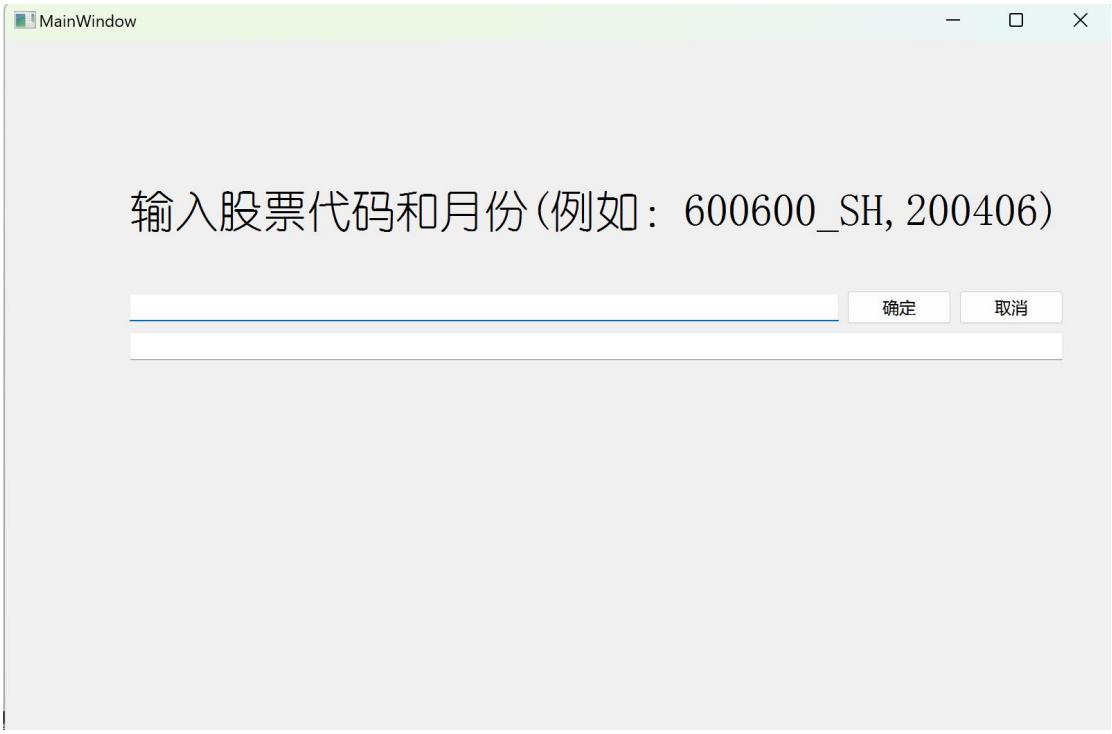
```
1.      for (int i = 0; i < capacity - 1; i++)
2.          {
3.              in >> pIndexStock[1];
4.              if (pIndexStock[0] == pIndexStock[1]) continue;
5.
6.              out << pIndexStock[1].toString();
7.              pIndexStock[0] = pIndexStock[1];
8.          }
```

这部分封装在 IndexStockArray 的类成员函数 run()，调用时简便，数据安全。

2 股票价格预测和可视化



在 main 函数里，读入所有的 index 信息存储在 indexArr 里。在 MainWindow 里封装一个 private IndexStockArray 变量，把 indexArr 传入 MainWindow，这样就把数据预处理和交互窗口分开。



交互界面如图，在第一栏输入指定股票和月份后，点击确定，若查询到相关数据，则在第二栏打印“查询成功！”，否则打印“未找到！查询失败”。点击确定后会绘制 K 形图和价格预测折线图。点击取消推出交互界面。

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    MainWindow(IndexStockArray& arr,QWidget *parent = nullptr);
    ~MainWindow();

    int CommitClick(std::string input);

    void setIndexArr(const IndexStockArray& arr);

private slots:
    void on_commitButton_clicked();

    void on_cancelButton_clicked();

private:
    Ui::MainWindow *ui;

    PlotWindow *pw;

    PredictionPlotWindow *ppw;

    IndexStockArray indexArr;
    StockChinaArray aStArr;//存储一支股票在这个月的数据
    StockChinaArray preStArr;//存储该股票前一个月的数据进行预测

    std::string filepath;//"D:\\stock_data\\bigdataCh\\ASCSorted_output_China.csv"
};
```

MainWindow 类封装两个 StockChinaArray，为后面的可视化做准备，第一个 aStArr 存储用户指定的股票在指定月份的数据，而 preStArr 存储指定月份前一个月的数据。在用户输入代码和月份信息后，在内存的 IndexStockArray 中检索到相关位置后，在大文件里定位到该位置读入 aStArr，同时读入前一个月的位置，读入 preStArr。

发现其实 K 形图和价格预测都要用到同样部分的数据，虽然 K 形图要用 open, high, low, close 四个数据，价格预测只要 close，但考虑到总体不算大，分开读两次显得笨拙，所以设计在 MainWindow 里读入，这样在后续画图时可以调用内存里的数据，不用从文件里重新读取，提高效率。

2.1 K 形图绘制

K 形图调用 aStArr 之后，运用 qcustomplot 中的 QCPFinancialDataContainer 绘制。

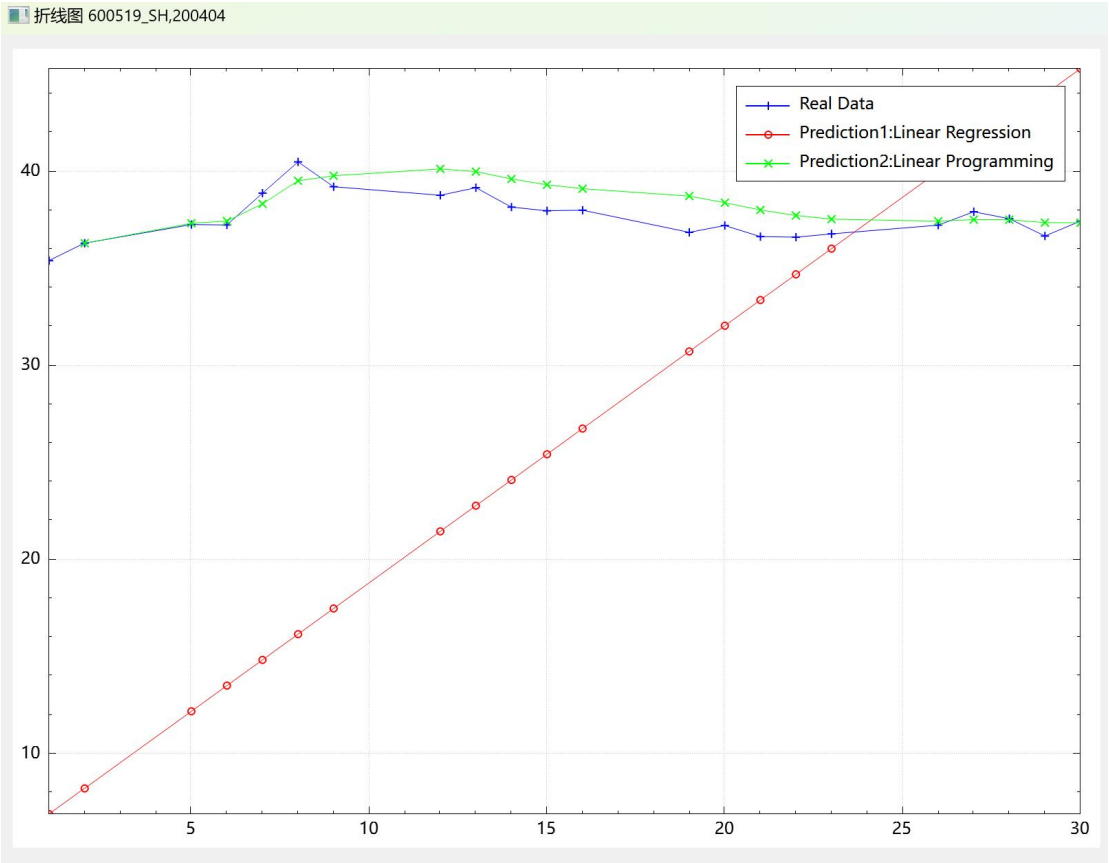
左上角显示股票代码和指定月份。

2.2 价格预测

我提供了两种预测方案，分别封装在 LinearRegression 和 LinearProgramming 里。

```
class PredictionPlotWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit PredictionPlotWindow(QWidget *parent = nullptr);
    ~PredictionPlotWindow();

    //从
    int loadFrom(StockChinaArray& Arr);
    //在窗体显示图形
    void showFigure(QString input);

    void LRprepare(StockChinaArray& Arr);//对前一个月进行线性回归得到w 和b

    void LRpredict(StockChinaArray& Arr);//对这个月进行线性回归

    void LPrun();//线性规划

private:
    Ui::PredictionPlotWindow *ui;

```

```
        LinearRegression LR;//线性回归

        LinearProgramming LP;//线性规划
        //绘制
        void draw(QCustomPlot *customPlot);

        QVector<double> x;//这个月的日期
        QVector<double> y_pred1;//线性回归预测的价格
        QVector<double> y_pred2;//线性规划预测的价格
        QVector<double> y;//真实价格
        QVector<double> xx;//前一个月的日期
        QVector<double> yy;//前一个月的价格

    };
```



2.2.1 线性回归

红线是线性回归预测(LinearRegression)，显然预测很失败。我尝试根据 Python 与人工智能课上学到的手搓了线性回归模型，但是效果很差。

```
double* LinearRegression::__calc_gradient(QVector<double> x, QVector<double> y)//梯度下降
{
    double* dw_and_db;
    dw_and_db = new double[2];
    double sum_w = 0, sum_b = 0;
```

```
        for (int i = 0; i < x.size(); i++)
        {
            double temp = 2 * (x[i] * w + b - y[i]);
            sum_w += temp * x[i];
            sum_b += temp;
        }
        dw_and_db[0] = (sqrt(sum_w / x.size()) > 0.000001) ? sqrt(sum_w / x.size()) : 0;
        dw_and_db[1] = (sqrt(sum_b / x.size()) > 0.000001) ? sqrt(sum_b / x.size()) : 0;
        return dw_and_db;
    }
```

运用了梯度下降法。个人怀疑问题出在这部分。当梯度小于 0.000001 时认为梯度没有下降，导致的误差。但是不做此限制，梯度下降可能因为过小返回 nan，不能绘图。从打印出来的 loss 来看，误差确实是在下降，但是可能收敛在一个比较大的值上。不排除有其它 bug 的可能，只是实在没找到了。大概画出来有三种样子：y=x, y=-x, y=k（某个常数）。

2.2.2 线性规划

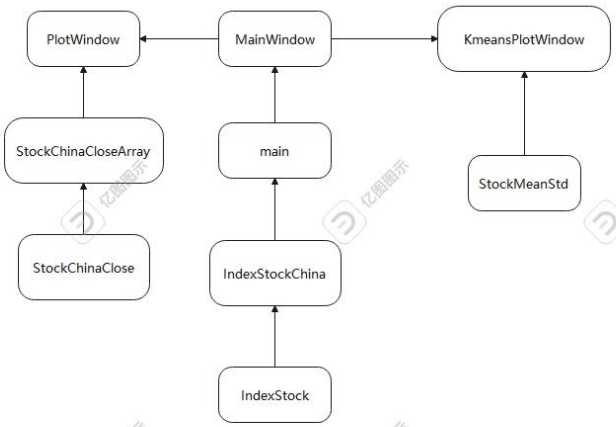
绿色那条线就是线性规划的结果。

线性回归效果不理想，我突然想起高中数学的线性规划模型，觉得可以试一试。

$$\hat{b}=\frac{\sum_{i=1}^{n}(t_i-\bar{t})(y_i-\bar{y})}{\sum_{i=1}^{n}(t_i-\bar{t})^2},\ \hat{a}=\bar{y}-\hat{b}\bar{t}.$$

y=bx+a。其实这也是线性回归方程，简单地区分叫做线性规划。先利用前一个的数据计算出初始的 b 和 a，再一天一天地计算这个月的数据，这样成功让直线拐弯。预测效果还是很不错的。

3 指标计算和 K-means 聚类



启动程序，相似地，在 main 函数里读入 Index 信息建立索引。

如图，先在第一栏输入月份，随后遍历 IndexStockArray 数组，找出所有的这个月的股票并返回在第一个框里，用户从中挑选 10 支股票后输入第二个框，点击确认后计算收益率

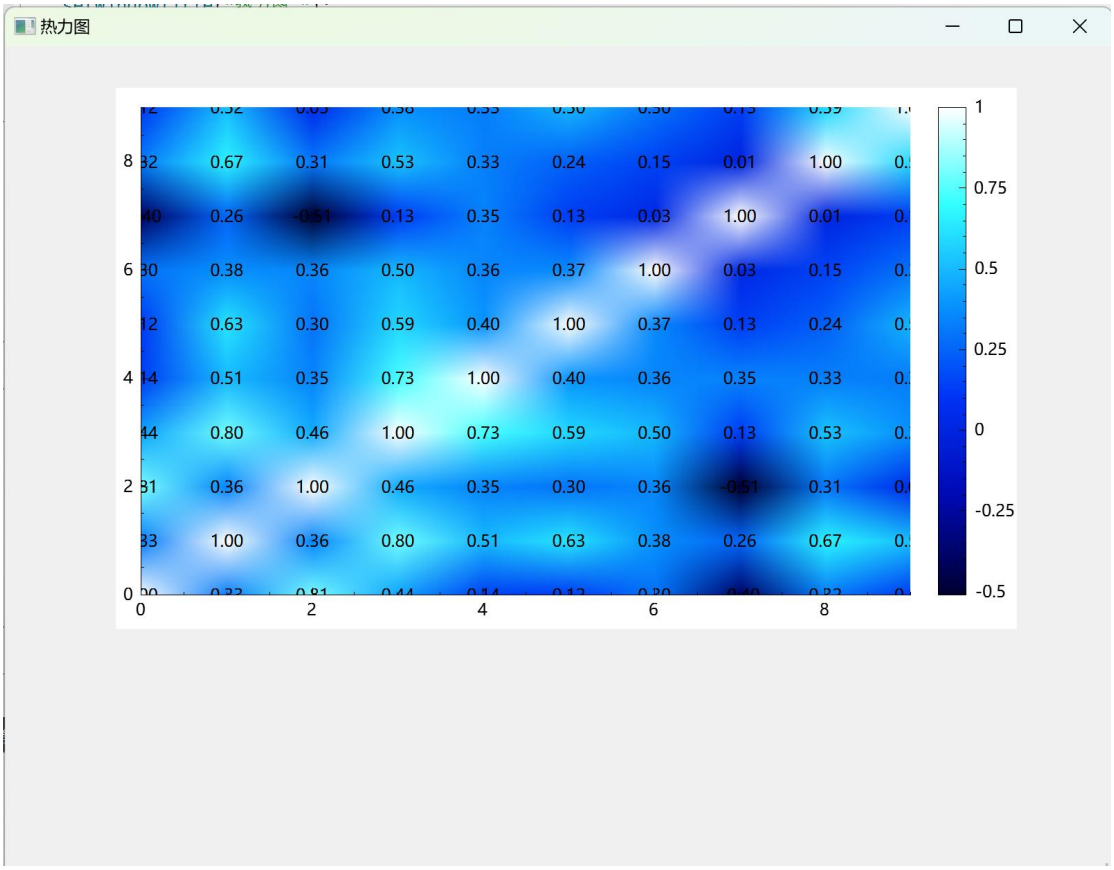
和两两之间的皮尔森系数，绘制热力图。点击确认推出界面。

进行 K-means 聚类，点击确认键后绘制 K-means 聚类这个月所有股票的散点图。股票数据以月收盘价的平均值为 x 坐标，收盘价的标准差为 y 坐标。

3.1 指标计算

新创建一个类 StockChinaClose 仅保留 symbol，datetime，close 三个成员变量。其余操作与之前相似，存储在 StockChinaCloseArray 中，在 PlotWindow 中通过 loadfrom(const StockChinaCloseArray& arr);传入数据后分别计算两两之间的皮尔森系数，返回一个 double** 指针，指向 10*10 的数组，及皮尔森系数矩阵并绘制出来。

```
1.  double** PlotWindow::loadfrom(const StockChinaCloseArray* pSCCA)
2.  {
3.      double** pdouble = new double* [10];
4.      for (int i = 0; i < 10; i++)
5.      {
6.          pdouble[i] = new double [10];
7.      }
8.      for (int i = 0; i < 10; i++)
9.      {
10.         for (int j = 0; j < 10; j++)
11.         {
12.            pdouble[i][j] = Pierce(pSCCA[i], pSCCA[j]);
13.         }
14.     }
15.     return pdouble;
16. }
```

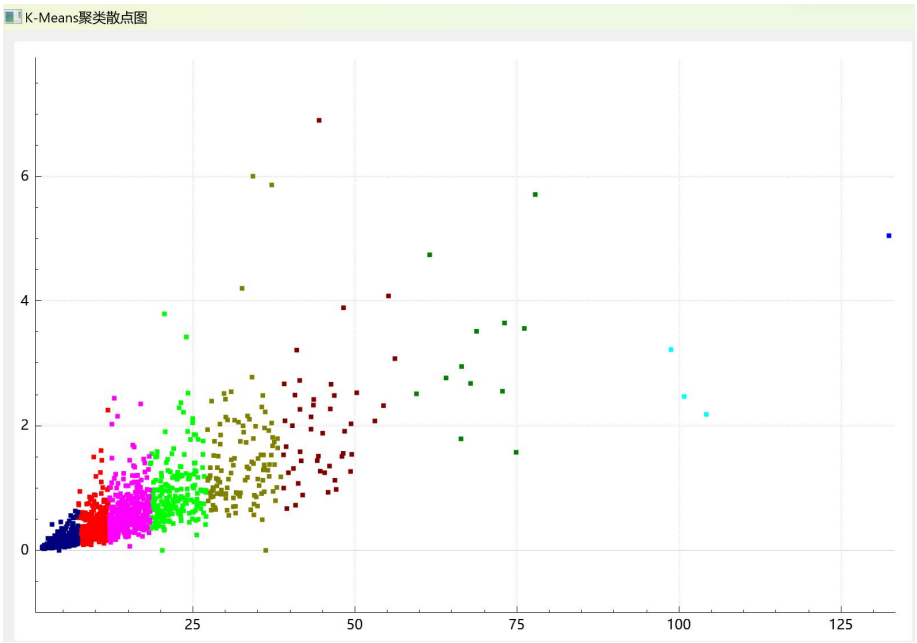


可见在 y=x 线上的值均为 1.00，及自身与自身的比值。所有值在-1 到 1 之间，符合基本要求。

3.2K-Means 聚类

因 K-Means 聚类设计无监督学习，尝试用 double 指针手搓模型，最终失败，还是采用了 vector。

新建一个 StockMeanStd 类，保存股票收盘价平均值和标准差。在 MainWindow 类创建一个 StockMeanStd 指针 pSMS，通过索引在大文件里读取一支股票在一个月的所以数据后就调用 StockMeanStd 的 loadfrom 方法，保存进 pSMS 指针指向的数组。



```
void MainWindow::prepareKMeans()
{
    std::string mon;//用户输入
    mon = wordInput.toStdString();

    std::ifstream fileIN;
    fileIN.open("D:\\stock_data\\bigdataCh\\ASCSorted_output_China.csv",
                std::ios::in | std::ios::binary);

    int size = isArr.getSize();
    pSMS = new StockMeanStD[size];

    for (int i = 0; i < isArr.getSize(); i++)
    {
        StockChinaCloseArray stArr;
        stArr.resize(0, false);
        int pos = isArr[i].pos();
        pos -= 21; //退回这一行的开头
        fileIN.seekg(pos, std::ios::beg);

        while (!fileIN.eof())
        {
            StockChinaClose st;
            fileIN >> st;
            std::string stmon;
            stmon = st.dt2mon();
            if ((st.sym() == isArr[i].sym()) && (stmon == mon))
            {
                stArr.push_back(st);
            }
            else break;
        }
        pSMS[i].loadfrom(stArr);
        //qDebug() << QString::fromStdString(pSMS[i].toString()) << '\n';
    }
    fileIN.close();
    number = isArr.getSize();
    qDebug() << isArr.getSize() << '\n';
}
```

```
void StockMeanStD::loadfrom(const StockChinaCloseArray& arr)
{
    if (!arr.getSize()) return;
```

```
        symbol = arr[0].sym();
        mean = cal_mean(arr);
        standardDeviation = cal_standard_deviation(arr);
    }
```

在 KmeansPlotWindow 调用 loadfrom 函数后读入，创建 vector<vector<double>>进行 K-Means 聚类，并绘制。

4 反思总结代后记

这个大作业写了总共有十天左右，每天都一门心思扑在上面。其间参考了很多老师提供的源码和网络上的知识。后面 QCustomPlot 也是一点一点的学习，收获好多好多。

4.1 可能的改进方向

除去失败的线性回归，我认为后两个项目中对继承、多态的使用不是很多，这是遗憾的地方。绘制热力图也不是很美观，界面不够大导致边缘数据不能很好地表现出来。

4.2 一点点总结

4.2.1 类模板

类模板定义和实现要在同一个 .h 文件里。我之前按照习惯分开在.h 和.cpp 文件里。导致一直报错 LNK2019，在这个地方卡顿了整整两天。后来在某天半夜在小喇叭里请教才得以解决……

4.2.2vector 和指针

上学期学习 C 语言，自认为对指针掌握不是很好，所以一直偏向于使用 vector。但是在这次编写中，有一次使用 vector 结果报错 bad alloc，上网查资料后发现 vector 很吃内存，后采用指针成功解决，加深了对指针的理解。但是指针也不是万能的。在写 K-Means 聚类时，尝试通过以下语句：

```
double* pd;
int n = sizeof(pd) / sizeof(pd[0]);
```

获取数组大小，但是以上只对数组名有用而对指针无效。

最后考虑，最好的方法还是自己创建一个 Array 类进行存储，重载[], resize(), =, push_back, size()等实现。最后没有创建一个 double array 实现 K-means，也是一个遗憾……

最后的最后，还是觉得此次程序达到预期，还学习了监督学习和无监督学习的一点入门，对大数据的处理也有了初步了解，就这样结束吧！谢谢老师！