第 5 章

会计决策分析方法

会计决策可以帮助会计人员解决企业资金运动过程中所出现的问题，以及把握机会而制定和选择活动方案。通过了解企业资金运动的方向、状态和效益，可以帮助企业做出更有利的选择。而且由于资金运动具有可控性，人们就可以通过决策和控制，促使企业的资金运动朝着有利的方向发展。本章主要围绕着成本分析、本量利分析、边际值分析、预测分析和财会机器人几方面来说明。

5.1 成本分析

成本分析方法利用成本计划、成本核算和其他有关信息实现对实际成本支出的有效控制，从而了解成本计划完成情况，查明成本变化的原因，为寻求降低成本提供途径和方法。

5.1.1 不同性质的成本对比分析

单位固定成本具有反比例变动性，固定成本具有总额不变性的特征，导致单位产品负担的固定成本随着业务量的变动呈反比例变动；而变动成本则具有总额的正比例变动性，会随着业务量的变动呈正比例变动。

常见的可视化展示可以有助于更好地观察相关变动趋势，如表 5-1 所示的数据。

表 5-1 一家企业的固定成本和产品材料成本情况

产量(件)	固定成本(元)	单位产品所负担的固定成本(元)	单位产品材料成本(元)	产品材料成本总成本(元)
1000	30 000 000	30 000	5	5000
2000	30 000 000	15 000	5	10 000
3000	30 000 000	10 000	5	15 000
4000	30 000 000	7500	5	20 000
5000	30 000 000	6000	5	25 000
6000	30 000 000	5000	5	30 000

此类问题其实反映了一种基本的数据对比分析功能，基本的方法就是直接将相关数据绘制在同一张坐标图上。

【例 5-1】 可视化对比“单位产品所负担的固定成本”和“产品材料成本总成本”。

```
1  import matplotlib.pyplot as plt
2  import pandas as pd
3  data = pd.read_csv('企业成本变动数据.csv', encoding = 'GBK')
4  plt.plot(data['产量(件)'], data['单位产品所负担的固定成本(元)'])
5  plt.plot(data['产量(件)'], data['产品材料成本总成本(元)'])
6  plt.show()
```

输出如图 5-1 所示。

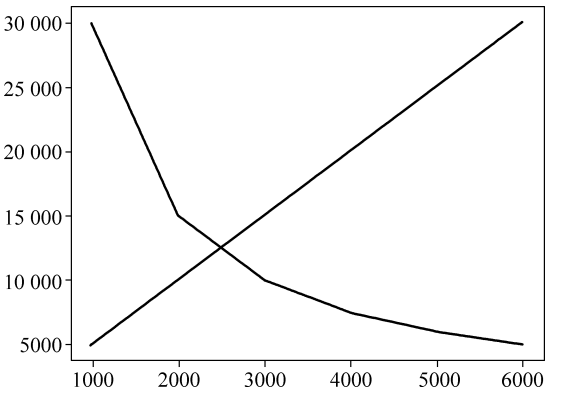

图 5-1 不同性质的成本变动对比

如果数据取值范围不一致，就不能使用同一坐标系来绘制，否则会导致对变动差异无法有效区分，更合适的做法是分别绘制。代码如下：

```
1  import matplotlib.pyplot as plt
2  import pandas as pd
3  data = pd.read_csv('企业成本变动数据.csv', encoding = 'GBK')
4  plt.subplot(211)
5  plt.plot(data['产量(件)'], data['单位产品所负担的固定成本(元)'])
6  plt.subplot(212)
7  plt.plot(data['产量(件)'], data['产品材料成本总成本(元)'])
8  plt.show()
```

输出如图 5-2 所示。

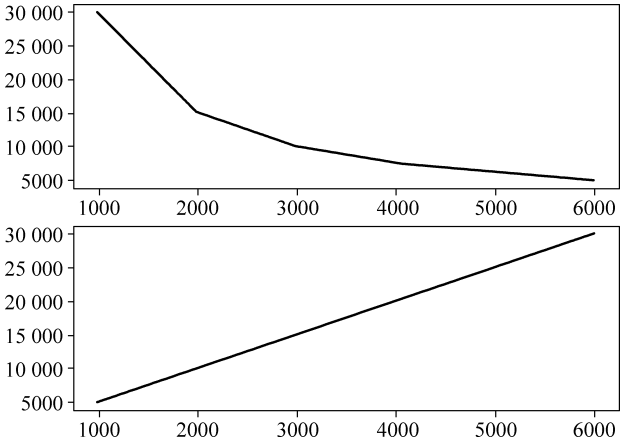

图 5-2 使用不同子图来绘制不同性质的成本变动

5.1.2　历史成本法的拟合

历史成本法可以根据以往若干时期的数据分析实际成本与业务量之间的依存关系，进而描述成本的性态并以此来确定决策所需要的未来成本数据。该方法假设在既定的生产流程和工艺设计条件下，历史数据可以比较准确地表达成本与业务量之间的依存关系，并可以应用到现在或将来的决策中。传统的方法包括高低点法、散布图法和回归法等。利用 Python 的 Numpy 等库可以实现基于回归方法的历史成本法。

例如，某企业的产量和混合成本的历史资料如表 5-2 所示。

表 5-2　某企业的产量和混合成本的历史资料

月　份	产量(件)	成本(元)
1	50	350
2	55	410
3	60	410
4	75	500
5	75	510
6	85	530

【例 5-2】 使用历史成本法拟合成本变化趋势。

```
import pandas as pd
from numpy import polyfit
import matplotlib.pyplot as plt
data = pd.read_csv('混合成本和产量的历史资料.csv', encoding = 'GBK')
x = data['产量(件)']
y = data['成本(元)']
coeff = polyfit(x, y, 1)
plt.plot(x, y, 'rx')
plt.plot(x, coeff[0] * x + coeff[1], 'k-')
plt.show()
```

输出如图 5-3 所示。

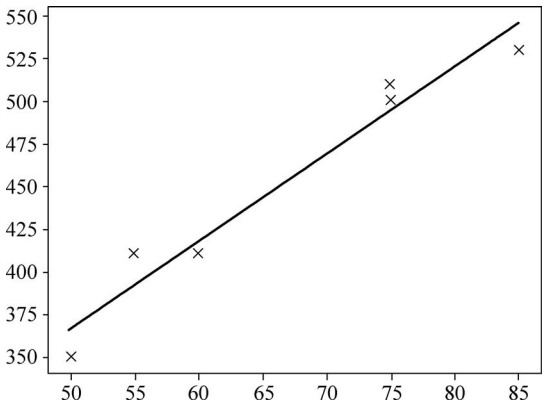

图 5-3　使用回归方法实现历史成本法

扫码看彩图

代码说明如下：

（1）第七行使用了 Numpy 库的 polyfit()方法来拟合回归，该方法可以直接对前两个参数进行拟合，第三个参数值为 1 表示一阶多项式，即线性拟合。同时，该方法通过列表返回拟合后的直线斜率和截距。

（2）第八、九行使用原始数据和 polyfit()返回的直线斜率和截距计算数据，绘制相应的拟合直线，其中样式“rx”和“k-”分别表示红色的叉和黑色的实线。

为了简单使用拟合后的参数，还可以直接利用 poly1d 自动生成一个以传入数值为参数的多项式函数。代码如下：

```
1  import pandas as pd
2  from numpy import polyfit, poly1d
3  import matplotlib.pyplot as plt
4  data = pd.read_csv('混合成本和产量的历史资料.csv', encoding = 'GBK')
5  x = data['产量(件)']
6  y = data['成本(元)']
7  coeff = polyfit(x, y, 1)
8  plt.plot(x, y, 'rx')
9  func = poly1d(coeff)
10 plt.plot(x, func(x), 'k-')
11 plt.show()
```

输出内容同上。

其中 poly1d()方法可以根据参数来生成一个函数，然后即可使用该函数进一步得到拟合函数的输出结果。借助此方法，还可以实现二阶多项式拟合，只需要将 polyfit()的第三个参数改为 2 即可，如下：

```
coeff = polyfit(x, y, 2)
```

输出如图 5-4 所示。

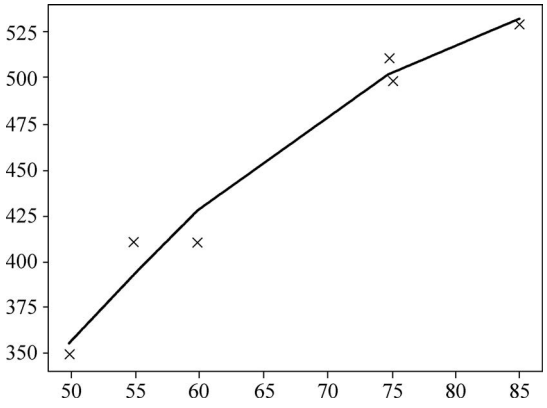

图 5-4　使用二阶多项式拟合实现成本拟合

这个拟合的多项式函数具有符号计算功能，不仅可以表达符号形式，而且可以进行符号运算，这对于进一步的算法设计有着很大的意义。如：

```
1  import pandas as pd
2  from numpy import polyfit, poly1d
```

```
data = pd.read_csv('混合成本和产量的历史资料.csv', encoding = 'GBK')
x = data['产量(件)']
y = data['成本(元)']
coeff = polyfit(x, y, 1)
func = poly1d(coeff)
print(func)
print(func + 2 * func ** 2)
```

输出如下：

```
5.125 x + 110
      2
52.53   + 2260 x + 2.431e+04
```

5.1.3　账户成本法

账户成本法需要从会计系统中寻找成本性态信息，根据成本账户的内容结合与产量的关系，判断其属于哪一类成本。一般而言，管理费用与产量关系不明显，可以看成固定成本，而间接材料费虽然与产量并不构成正比关系，但是也应该看成变动成本。

例如，某企业只生产一种产品，生产月成本如表 5-3 所示。

表 5-3　某企业的月成本

账　　户	月成本(元)
原材料费	20 000
工人工资	24 000
燃料动力费	8000
修理费	4000
管理人员工资	4000
折旧费	16 000
办公费	4000

成本性态判断只能在一定的假设条件下进行。一般而言，原材料费和工人工资通常为变动成本，燃料动力费、修理费、管理人员工资虽然与产量的变动不构成正比关系，但由于人们不了解其他产量水平下的实际成本，无法对其进行成本性态分析，因而只能先将其视为变动成本。折旧费和办公费与产量变动没有明显关系，因而确定为固定成本。

【例 5-3】　使用账户成本法分析企业成本。

```
import pandas as pd

def getCostClass(account):
    costclass = {'原材料费': '变动成本',
                 '工人工资': '变动成本',
                 '燃料动力费': '变动成本',
                 '修理费': '变动成本',
                 '管理人员工资': '变动成本',
                 '折旧费': '固定成本',
                 '办公费': '固定成本'}
```

```
        return costclass[account]

2  data = pd.read_csv('企业成本数据.csv', encoding = 'GBK')
3  data['类别'] = data['账户'].apply(getCostClass)
4  result = data.groupby('类别').sum()
5  a = result[result.index == '固定成本']['总成本(元)'][0]
6  b = result[result.index == '变动成本']['总成本(元)'][0]
7  print('固定成本为: {},5000 件产品的单位变动成本为: {}'.format(a, b / 5000))
```

输出如下：

```
固定成本为: 20000,5000 件产品的单位变动成本为: 12.0
```

代码说明如下：

(1) 代码开头定义的 getCostClass()函数采取了一种字典的表达方式，用来映射现有各种具体成本名称和标准成本类型之间的关系，可以将参数指定的成本名称作为键直接获取所对应的成本类型值。

(2) 主代码第三行通过获取每种成本的类型后，在现有每行成本后追加了一个新列，表达成本类型，并以此进行分组汇总计算。

(3) 第五、六行通过条件选择和列选择分别获取了两种成本的数值。此时每个结果只有一列一行，但是仍然为 Series 类型，因此需要通过切片获取第一个元素值。

(4) 输出采取了格式化字符输出方法，同样的功能也可以写为

```
print('固定成本为: ' + str(a) + ',5000 件产品的单位变动成本为: ' + str(b / 5000))
```

5.1.4 作业成本法

作业成本法更适合现代制造业中常见的产品成本计算，因为现代产品的制造费用构成更加复杂，直接人工成本所占比例相对不断下降。作业成本法以作业为中心，根据作业对资源耗费的情况将资源成本分配到作业中，然后根据产品和服务所耗用的作业量，最终将成本分配到产品或服务中。该方法的基本成本对象是作业，产品间接成本的分配趋于合理，不再区分直接费用和间接费用，所有成本均是变动成本。因此，该方法更有利于企业分析成本产生的动因分析，以便更好地降低成本。

某企业生产甲、乙两种产品，本月有关两种产品的成本资料如表 5-4 所示。

表 5-4 某企业两种产品的基本成本资料

成本项目	甲产品	乙产品
本月产量(件)	100	400
单位产品工时(小时)	4	4
直接材料单位成本(元)	20	50
直接人工单位成本(元)	40	20

总相关制造费用为 2 万元，两种产品的复杂程度不同，耗用作业量也不同。已知归纳出 5 项关键性作业，并将每项关键性作业作为一个作业中心来建立成本库，具体数据如表 5-5 所示。

表 5-5　作业成本及产品作业情况

作 业 名 称	作业成本(元)	甲产品作业量(件)	乙产品作业量(件)
设备维护	4000	16	4
订单处理	2000	140	60
机器维修	1600	60	20
机器运行	10 000	400	1600
质量检验	2400	120	80

【例 5-4】　使用作业成本法分析企业成本。

```
1  import pandas as pd
2  item = pd.read_excel('作业成本数据.xlsx', engine = 'openpyxl', sheet_name = '产品成本')
3  activity = pd.read_excel('作业成本数据.xlsx', engine = 'openpyxl', sheet_name = '作业
   成本')
4  activity['作业成本动因分配率'] = activity['作业成本'] / (activity['甲产品作业量'] +
   activity['乙产品作业量'])
5  activity['甲产品制造费用作业成本'] = activity['甲产品作业量'] * activity['作业成本
   动因分配率']
6  activity['乙产品制造费用作业成本'] = activity['乙产品作业量'] * activity['作业成本
   动因分配率']
7  cost1OfItem1 = item[2:]['甲产品'].sum()
8  cost1OfItem2 = item[2:]['乙产品'].sum()
9  cost2OfItem1 = activity['甲产品制造费用作业成本'].sum()
10 cost2OfItem2 = activity['乙产品制造费用作业成本'].sum()
11 quanlityOfItem1 = item['甲产品'][0:1][0]
12 quanlityOfItem2 = item['乙产品'][0:1][0]
13 costOfItem1 = cost1OfItem1 + cost2OfItem1 / quanlityOfItem1
14 costOfItem2 = cost1OfItem2 + cost2OfItem2 / quanlityOfItem2
15 print('甲产品的最终单位成本: {},乙产品的最终单位成本: {}'.format(costOfItem1,
   costOfItem2))
```

输出如下：

```
甲产品的最终单位成本: 152.4,乙产品的最终单位成本: 96.9
```

代码说明如下：

(1) 第二行和第三行代码分别读取了位于扩展名为 xlsx 的 Excel 文档中两张工作表的数据内容，此时需要加载 openpyxl 库。

(2) 第四行代码计算所有产品的作业成本动因分配率，用总作业成本除以所有产品的作业量。这里只涉及制造费用相关成本。

(3) 第五、六行在已知作业成本动因分配率的情况下，根据两种产品各自的作业量，分别计算各自的制造费用作业成本。同第四行代码一样，这里所计算的数据都以新数据列的方式存储到现有的 DataFrame 中。

(4) 第七、八行计算了两种产品(甲产品对应 item1，乙产品对应 item2)各自的直接成本(cost1)，这些信息位于“产品成本”工作表中，表中从第三行开始为产品的成本信息，因此行选择从第三行(行号为 2)开始。

(5) 第九、十行汇总了两种产品的制造费用作业成本(cost2)，该信息位于“作业成本”工作表中。

(6) 第十一、十二行分别得到了两种产品的生产数量,这里的查询对于列直接使用名称,对于行使用序号,0:1 表示第一行,即从 0 开始不到 1,所以为第一行。但是,这里的 item 查询结果是 Series 类型,因此相关结果即使只有一个值,也会返回一个序列,所以需要通过再次切片取得第一个元素来获得最终的结果。

(7) 第十三、十四行分别计算了两种产品的最终单位成本,最终单位成本为单位直接成本和单位作业成本之和。可以看出每种产品的成本都是由直接材料单位成本和直接人工单位成本构成的基本成本与按照作业量分摊的制造成本之和。

5.2 本量利分析

本量利分析是指在成本性态分析基础上,通过数学方法来建立成本、业务量和利润三者之间的关系模型,揭示成本、销售量和利润等诸多变量之间的内在联系,最终为企业经营决策提供定量分析基础。

5.2.1 本量利基本指标分析

假设一家企业只生产一种产品,其单价为 10 元,单位变动成本为 6 元,全年固定成本为 3 万元,当年产量为 1.2 万件。相关的本量利基本指标(如单位贡献毛益、贡献毛益、贡献毛益率、变动成本率、营业利润等)的计算方法如下:

$$\text{单位贡献毛益} = \text{单价} - \text{单位变动成本}$$

$$\text{贡献毛益} = \text{单位贡献毛益} \times \text{产量}$$

$$\text{贡献毛益率} = \frac{\text{单位贡献毛益}}{\text{单价}}$$

$$\text{变动成本率} = \frac{\text{单位变动成本}}{\text{单价}}$$

$$\text{营业利润} = \text{单位贡献毛益} \times \text{产量} - \text{固定成本}$$

【例 5-5】 对企业生产状况进行本量利指标分析。

```
price = 10                                  # 单价
vCostsPerUnit = 6                           # 单位变动成本
fixedCost = 30000                           # 固定成本
quantity = 12000                            # 产量
cm = price - vCostsPerUnit                  # 单位贡献毛益
Tcm = cm * quantity                         # 贡献毛益
cmR = cm / price                            # 贡献毛益率
bR = vCostsPerUnit / price                  # 变动成本率
profit = cm * quantity - fixedCost          # 营业利润
print('单位贡献毛益:{}'.format(cm))
print('贡献毛益:{}'.format(Tcm))
print('贡献毛益率:{}'.format(cmR))
print('变动成本率:{}'.format(bR))
```

```
14  print('营业利润:{}'.format(profit))
```

输出如下：

```
单位贡献毛益:4
贡献毛益:48000
贡献毛益率:0.4
变动成本率:0.6
营业利润:18000
```

相关计算都比较直观。在这些指标中，贡献毛益是衡量企业经济效益的重要指标，反映了贡献边际或边际贡献。贡献毛益的大小将直接影响企业产品销售盈亏水平的高低，只有当贡献毛益大于固定成本时才能为企业提供利润。

贡献毛益率是反映产品盈利能力的相对数指标，它表明每增加一元销售额能够为企业做出的贡献。它与变动成本率具有互补关系，因此变动成本率也可以直接使用贡献毛益率求得如下：

```
bR = 1 - cmR  # 变动成本率
```

5.2.2　盈亏平衡分析

1. 盈亏平衡临界点计算及企业经营安全评价

盈亏平衡临界点也称为保本点、损益平衡点、够本点等。此时，产品总收入等于总成本，贡献毛益正好抵偿全部固定成本，利润为零，企业处于不盈利也不亏损的状态。显然，盈亏平衡临界点对于企业的经营决策具有重要意义，只要销售业务量超过保本点，企业就会有盈利，反之就会亏损。因此，在本量利分析基础上，可以通过确定产品的盈亏平衡点，确定企业的产品生产销售的安全程度，结合保利分析和多因素变动分析，提供生产经营决策依据。

具体计算公式为

$$\text{保本量} = \frac{\text{固定成本}}{\text{单位贡献毛益}}$$

$$\text{保本额} = \text{保本量} \times \text{单价}$$

本节沿用 5.2.1 节的例子。

【例 5-6】 计算盈亏平衡点的保本量和保本额。

```
price = 10                              # 单价
vCostsPerUnit = 6                       # 单位变动成本
fixedCost = 30000                       # 固定成本
quantity = 12000                        # 产量
cm = price - vCostsPerUnit              # 单位贡献毛益
Tcm = cm * quantity                     # 贡献毛益
cmR = cm / price                        # 贡献毛益率
bR = vCostsPerUnit / price              # 变动成本率
profit = cm * quantity - fixedCost      # 营业利润
basicQuantity = fixedCost / cm          # 保本量
```

```
basicValue = basicQuantity * price      # 保本额
print('保本量为{},保本额为{}'.format(basicQuantity, basicValue))
```

输出如下：

```
保本量为 7500.0,保本额为 75000.0
```

有了保本量等指标后，就可以进一步对企业经营安全程度做出评价。常见的方法是安全边际指标分析，即根据销售业务量与保本业务量的差量计算，具体包括以绝对量表示的安全边际量、安全边际额（两者分别以数量和金额测度）和以相对量表示的安全边际率。与安全边际指标相反的指标有保本作业率，也称为危险率，指保本点业务量占实际或预计销售业务量的百分比，因此，保本作业率和安全边际率之和为 1，保本作业率数值越小，说明企业经营的安全程度越高。

具体计算公式如下：

$$\text{安全边际量} = \text{产量} - \text{保本量}$$

$$\text{安全边际额} = \text{单价} \times \text{产量} - \text{保本额}$$

$$\text{安全边际率} = \frac{\text{安全边际量}}{\text{产量}}$$

$$\text{保本作业率} = \frac{\text{保本量}}{\text{产量}}$$

【例 5-7】 对企业生产状况进行安全边际指标分析。

```
price = 10                                  # 单价
vCostsPerUnit = 6                           # 单位变动成本
fixedCost = 30000                           # 固定成本
quantity = 12000                            # 产量
cm = price - vCostsPerUnit                  # 单位贡献毛益
Tcm = cm * quantity                         # 贡献毛益
cmR = cm / price                            # 贡献毛益率
bR = vCostsPerUnit / price                  # 变动成本率
profit = cm * quantity - fixedCost          # 营业利润
basicQuantity = fixedCost / cm              # 保本量
basicValue = basicQuantity * price          # 保本额
MSq = quantity - basicQuantity              # 安全边际量
MSv = price * quantity - basicValue         # 安全边际额
MSR = MSq / quantity                        # 安全边际率
dR = basicQuantity / quantity               # 保本作业率
print('安全边际量为{:,},安全边际额为{:,}'.format(MSq, MSv))
print('安全边际率为{:.1%},保本作业率为{:.1%}'.format(MSR, dR))
```

输出如下：

```
安全边际量为 4,500.0,安全边际额为 45,000.0
安全边际率为 37.5%,保本作业率为 62.5%
```

相关指标计算都比较直观，其中由于保本作业率和安全边际率存在反相关关系，因

此可以计算：

```
dR = 1 - MSR  # 保本作业率
```

此时的安全边际率为37.5%，一般该指标位于30%和40%之间时表示安全，高于40%表示非常安全，而低于10%则表示危险。

最后的输出采取了特定格式。其中，安全边际量和安全边际额增加了千分位，保本作业率和安全边际率都采取了保留两位小数的百分数输出格式，这种格式就是通过花括号中的格式化字符来表示。此时必须以冒号开头，有“,”表示增加千分位，有“.1%”表示保留一位小数并且以百分数显示。更多字符串格式可参见附录C。

根据上述本量利分析结果，可以进一步实现目标利润模型，即保利分析。此时，保利量(额)是指实现目标利润所需的销售量(额)。由于现实世界中的成本、业务量和利润诸因素往往关系复杂，因此保利分析一般是通过逐一描述业务量、成本、单价、利润等因素相对于其他因素而存在的定量关系。

具体计算公式如下：

$$\text{保利量} = \frac{\text{固定成本} + \text{目标利润}}{\text{单位贡献毛益}}$$

$$\text{保利额} = \text{单价} \times \text{保利量}$$

假设该企业目标利润为2万元，其他条件不变。

【例5-8】 对企业生产状况进行保利分析。

```
price = 10                                          # 单价
vCostsPerUnit = 6                                   # 单位变动成本
fixedCost = 30000                                   # 固定成本
quantity = 12000                                    # 产量
cm = price - vCostsPerUnit                          # 单位贡献毛益
Tcm = cm * quantity                                 # 贡献毛益
cmR = cm / price                                    # 贡献毛益率
bR = vCostsPerUnit / price                          # 变动成本率
profit = cm * quantity - fixedCost                  # 营业利润
basicQuantity = fixedCost / cm                      # 保本量
basicValue = basicQuantity * price                  # 保本额
MSq = quantity - basicQuantity                      # 安全边际量
MSv = price * quantity - basicValue                 # 安全边际额
MSR = MSq / quantity                                # 安全边际率
dR = basicQuantity / quantity                       # 保本作业率
targetProfit = 20000                                # 目标利润
profitQuantity = (fixedCost + targetProfit) / cm    # 保利量
profitValue = price * profitQuantity                # 保利额
print('保利量为{},保利额为{}'.format(profitQuantity, profitValue))
```

输出如下：

```
保利量为12500.0,保利额为125000.0
```

由于目标利润是指缴纳所得税前的利润，因此从税后利润入手进行目标利润的规划和分析具有更现实的意义，此时就需要考虑所得税率变动对实现目标利润的影响。这个分析通常是通过净保利点来进行，具体包括实现目标净利润销售量的净保利量和实现目标净利润销售额的净保利额两种形式。

具体计算公式如下：

$$净保利量 = \frac{固定成本 + \frac{目标净利润}{1 - 所得税率}}{单位贡献毛益}$$

$$净保利额 = \frac{固定成本 + \frac{目标净利润}{1 - 所得税率}}{贡献毛益率}$$

假设企业目标净利润为 0.75 万元，所得税为 25%，其他条件不变。

【例 5-9】 对企业生产状况进行净保利点分析。

```
price = 10                                          # 单价
vCostsPerUnit = 6                                   # 单位变动成本
fixedCost = 30000                                   # 固定成本
quantity = 12000                                    # 产量
cm = price - vCostsPerUnit                          # 单位贡献毛益
Tcm = cm * quantity                                 # 贡献毛益
cmR = cm / price                                    # 贡献毛益率
bR = vCostsPerUnit / price                          # 变动成本率
profit = cm * quantity - fixedCost                  # 营业利润
basicQuantity = fixedCost / cm                      # 保本量
basicValue = basicQuantity * price                  # 保本额
MSq = quantity - basicQuantity                      # 安全边际量
MSv = price * quantity - basicValue                 # 安全边际额
MSR = MSq / quantity                                # 安全边际率
dR = basicQuantity / quantity                       # 保本作业率
targetProfit = 20000                                # 目标利润
profitQuantity = (fixedCost + targetProfit) / cm    # 保利量
profitValue = price * profitQuantity                # 保利额
targetNetProfit = 7500                              # 目标净利润
incomeTaxRate = 0.25                                # 所得税率
netProfitQuantity = (fixedCost + targetNetProfit / (1 - incomeTaxRate)) / cm
                                                    # 净保利量
netProfitValue = (fixedCost + targetNetProfit / (1 - incomeTaxRate)) / cmR # 净保利额
print('净保利量为{:.0f}万件,净保利额为{:.0f}万元'.format(netProfitQuantity /
10000, netProfitValue / 10000))
```

输出如下：

```
净保利量为 1 万件,净保利额为 10 万元
```

输出格式采取了取整的格式化写法。

2. 多品种盈亏临界点计算

现实中企业往往会产销多种产品，此时盈亏临界点就不能简单地使用销售量等实物

单位来计算，而需要计算盈亏临界点的销售额。具体方法包括加权平均模型、联合单位法等，此处以加权平均模型为例来说明。

加权平均模型是计算多品种盈亏临界点的常用方法。由于企业生产的各种产品盈利能力（即贡献毛益率）不同，因此需要加权平均各种产品的贡献毛益率来得到反映全部产品盈利能力的最终指标，权值使用各种产品的销售百分比，所以该模型的关键在于求出各种产品的贡献毛益率和各自的销售百分比。

假设某企业生产销售 A、B、C 三种产品，产销平衡，固定成本为 8.6 万元，每种产品的基本生产信息如表 5-6 所示。

表 5-6　企业产品的基本生产信息

产　品	销售量（件）	销售单价（元）	单位变动成本（元）
A	5000	40	25
B	10 000	10	6
C	12 500	16	8

【例 5-10】 对企业生产状况进行多品种盈亏临界点分析。

```
import pandas as pd
data = pd.read_csv('加权平均模型.csv', encoding = 'GBK')
data['销售百分比'] = data['销售量(件)'] * data['销售单价(元)']
data['销售百分比'] = data['销售百分比'] / data['销售百分比'].sum()
data['单位贡献毛益'] = data['销售单价(元)'] - data['单位变动成本(元)']
data['贡献毛益率'] = data['单位贡献毛益'] / data['销售单价(元)']
weightedCMR = (data['销售百分比'] * data['贡献毛益率']).sum()
fixedCost = 86000                                    # 固定成本
criticalPoint = round(fixedCost / weightedCMR, 4)    # 综合盈亏临界点销售额
data['产品盈亏临界点销售额'] = criticalPoint * data['销售百分比']
data['产品盈亏临界点销售量'] = data['产品盈亏临界点销售额'] / data['销售单价(元)']
print(data)
```

输出如下：

```
  产品  销售量(件)  销售单价(元)  单位变动成本(元)  销售百分比  单位贡献毛益  贡献毛益率  产品盈亏临界点销售额  产品盈亏临界点销售量
0  A    5000       40          25     0.4       15  0.375   80000.0    2000.0
1  B   10000       10           6     0.2        4  0.400   40000.0    4000.0
2  C   12500       16           8     0.4        8  0.500   80000.0    5000.0
```

代码说明如下：

（1）第三、四行计算了每种产品的销售百分比，即该产品的销售金额占总销售金额的比例。

（2）第五、六行计算了贡献毛益率，即每种产品的单价减去成本后的差值在单价中的比例。

（3）第七行使用销售百分比和贡献毛益率计算了加权贡献毛益率。

（4）第九、十行以加权贡献毛益率分摊了固定成本，得到综合盈亏临界点销售额。由于综合盈亏临界点销售额可能会因为金额计算产生一定的精度误差，因此此处进行了必要的四舍五入。

(5) 第十一、十二行以每种产品的销售百分比和综合盈亏临界点销售额得到每种产品的盈亏临界点销售额,进而结合每种产品的单价算出每种产品的盈亏临界点销售量。

5.3 边际值分析

边际值可以揭示两个具有因果或相关关系的经济变量之间的动态关系。例如当某个自变量发生一个微小单位的数量变化时,另一个因变量因此而发生的数量变化值称为该因变量的边际值。通过边际值的分析,可以有效地找到在成本、收益和利润等方面的关键指标数值,从而为生产计划和经营决策提供帮助。

5.3.1 边际成本计算

边际成本是总成本函数对产量等自变量的导数。同样,边际收益、边际利润、边际需求分别都是收益函数、利润函数、需求函数对相应自变量的导数。

如果某种商品的成本函数为(x 为产量)

$$\text{cost}(x) = 1000 + \frac{x^2}{1500}$$

可以直接利用 cost()函数得到总成本,平均成本等于总成本和商品件数的商。

【例 5-11】 计算生产 1000 件商品的总成本与平均成本。

```
1  def cost(c):
2      return 1000 + c ** 2 / 1500

3  quantity = 1000
4  print('总成本为: %f' % cost(quantity))
5  print('平均成本为: %f' % (cost(quantity) / quantity))
```

输出如下:

```
总成本为: 1666.666667
平均成本为: 1.666667
```

总成本的平均变化率等于总成本的变化值与商品件数变化值的商。

【例 5-12】 计算生产 1000～1200 件商品的总成本平均变化率。

```
1  def cost(c):
2      return 1000 + c ** 2 / 1500

3  quantity1 = 1200
4  quantity2 = 1000
5  print('总成本平均变化率为: %f' % ((cost(quantity1) - cost(quantity2)) / (quantity1 -
   quantity2)))
```

输出如下:

```
总成本平均变化率为: 1.466667
```

直接进行边际成本的求导计算比较复杂,可以考虑使用 scipy 的求导方法,如例 5-13 所示。

【例 5-13】 使用 scipy 库计算生产 1000 件产品的边际成本。

```
1  from scipy.misc import derivative

2  def cost(c):
3        return 1000 + c ** 2 / 1500

4  print(derivative(cost, 1000))
```

输出如下：

```
1.333333333333485
```

边际成本的计算需要使用求导的数学方法，可以利用 SciPy 库的 derivative()方法来实现。该方法有两个参数，第一个参数就是需要求导的函数，第二个参数就是自变量为该值时相应的导数值，即边际值。

同样的功能也可以使用 Sympy 库来实现。

【例 5-14】 使用 Sympy 库计算生产 1000 件产品的边际成本。

```
1  from sympy import *
2  x = symbols('x', positive = True)
3  cost = 1000 + x ** 2 / 1500
4  cost_d = diff(cost, x)
5  print(cost_d.subs(x, 1000))
```

输出如下：

```
4/3
```

Sympy 为一种符号计算库，它采取了更类似于人类书面表达数学运算的计算方式。代码说明如下：

(1) 第一行导入 Sympy 库，该库需要单独安装。

(2) 第二行定义了一个符号变量，名称为 x，并且 positive 属性值为 True 表示只考虑正数。很多函数计算结果会产生多个数值，但是在诸如经济管理实际应用中，只有正数数值解才有意义。

(3) 第三行定义了基于符号变量 x 的函数，此时，代码并不会计算相关结果存储在 cost 中，同时 cost 也不是字符串，而是一种采取符号变量的函数表达式。可以如下观察输出如下：

```
1  from sympy import *
2  x = symbols('x', positive = True)
3  cost = 1000 + x ** 2 / 1500
4  print(cost)
```

输出如下：

```
x**2/1500 + 1000
```

(4) 第四行利用 diff()方法进行函数表达式求导，两个参数分别表示要求导的函数和符号变量。可以如下观察输出：

```
1  from sympy import *
```

```
2  x = symbols('x', positive = True)
3  cost = 1000 + x ** 2 / 1500
4  cost_d = diff(cost, x)
5  print(cost_d)
```

输出如下：

```
x/750
```

(5) 第五行表示利用函数表达式的 subs()方法计算该函数表达式的结果，两个参数分别表示符号变量和此时的自变量取值。可以理解为此时将 1000 代入 x 自变量去求相应函数导数的因变量结果值。

5.3.2 边际收益计算

收益等于销售的商品数量和价格的乘积。假设有商品价格函数为(x 为销售量)

$$\text{price}(x) = 200 + \frac{x}{4}$$

【例 5-15】 计算生产 100 件商品的总收益与边际收益。

```
1  from scipy.misc import derivative

2  def price(q):
3      return 200 + q / 4

4  def totalRevenue(q):
5      return price(q) * q

6  quantity = 100
7  print('总收益为 % f' % (totalRevenue(quantity)))
8  print('边际收益为 % f' % (derivative(totalRevenue, quantity)))
```

输出如下：

```
总收益为 22500.000000
边际收益为 250.000000
```

这个边际收益值表明，在销售商品数量为 100 时，再多(少)销售一件，总收益会增加(减少)250 元。

【例 5-16】 计算销售 100～200 件商品的总收益平均变化率。

```
1  def price(q):
2      return 200 + q / 4

3  def totalRevenue(q):
4      return price(q) * q

5  quantity1 = 200
6  quantity2 = 100
7  print('总收益平均变化率为 % f' % ((totalRevenue(quantity1) - totalRevenue(quantity2)) /
   (quantity1 - quantity2)))
```

输出如下：

```
总收益平均变化率为 275.000000
```

本例的计算方式类似于前面的总成本平均变化率。

5.3.3　边际利润计算

边际利润一般为以销售量为自变量的利润函数的导数，其经济含义表现为当商品销售量为多少单位时，如果再多(少)销售一个单位商品，利润会增加(减少)多少单位。

假设有商品利润函数为(x 为销售量)

$$\text{profit}(x)=\frac{25\,000x-5x^3}{\sqrt{x}}$$

【例 5-17】 利用散点图展示销售量从 10 到 50 的边际利润。

```
from scipy.misc import derivative
import matplotlib.pyplot as plt
import math

def profit(q):
    return (25000 * q - 5 * q ** 3) / math.sqrt(q)

for i in range(10, 50):
    plt.scatter(i, derivative(profit, i))
plt.show()
```

输出如图 5-5 所示。

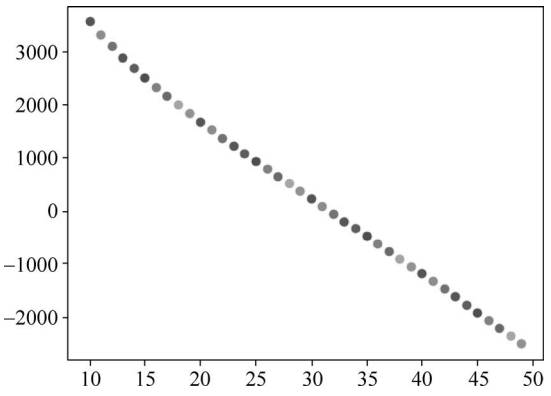

图 5-5　边际利润的散点图分布

本例对图中的颜色没有进行设置，系统将会使用随机颜色绘制。从图中可以看出，销售量并非越大越好，当销售量超过 30 后，边际利润反而为负，意味着此时销售量如果继续增大，反而会导致利润下降。

5.3.4　需求价格弹性计算

传统的边际效率分析方法是利用各种经济函数的自变量及其函数值的绝对改变量比率来计算，而弹性分析的意义就在于利用这些数值的相对变化率。这对于价格来说非常重要。例如 10 元的商品价格变化 1 元与 1000 元的商品价格变化 1 元相比，利用相对

变化幅度才能更好地反映这种价格变化的意义。

商品的需求量是指消费者有支付能力并且愿意购买的商品数量，如果不考虑其他因素，一般而言，商品的需求量与商品价格呈现减函数的特征。为了更好地测度需求和价格的关系，可以使用需求价格弹性指标来表达商品需求量对价格变化的敏感程度。

设函数 $y=f(x)$，弹性函数 F 的计算公式为

$$F=f'\times\frac{x}{f(x)}$$

假设价格需求函数为(x 为价格)

$$\text{demand}(x)=\text{e}^{\frac{-x}{100}}$$

【例 5-18】 计算商品价格分别为 50、100 和 150 时的需求价格弹性。

```
from scipy.misc import derivative
import math

def demand(p):
    return math.e ** (-p / 100)

price1 = 50
price2 = 100
price3 = 150
print((derivative(demand, price1) * price1 / demand(price1)))
print((derivative(demand, price2) * price2 / demand(price2)))
print((derivative(demand, price3) * price3 / demand(price3)))
```

输出如下：

```
-0.5000083333749993
-1.0000166667500023
-1.5000250001249986
```

这些结果数值可以反映价格变化对需求的影响程度。例如价格为 50 元时，需求价格弹性为 -0.5，表示此时商品价格如果在 50 元基础上增加 1 元的话，需求量就会减少 0.5%。一般而言，如果需求价格弹性的绝对值小于 1，说明需求量的变化幅度小于价格的变化幅度，可以称为缺乏弹性，因此适当提高价格不影响销售量，同时增加相应的收入。如果需求价格弹性的绝对值等于 1 时，说明需求量变化幅度和价格变化幅度一致，这个价格通常也是最优价格，此时的弹性数值可以称为单位弹性。如果需求价格弹性的绝对值大于 1，说明需求量的变化幅度大于价格的变化幅度，可以认为富有弹性，此时适当降低价格可以增加销售量，从而实现总收入的增加。

5.4 预测分析

5.4.1 销售预测

销售预测是指可以根据某种产品的历史销售数据等对其在未来一定期间内的销售

变动趋势做出预计和推断。因此，销售预测也是一种常见的时间序列预测方法，常见的方法有移动加权平均法、指数平滑法等。考虑到前文已经介绍部分相关内容，这里重点围绕具体案例和一些实现方法来说明。

1. 移动加权平均法

移动加权平均法是对过去若干期的销售量（额）按其时间远近分别进行加权，一般近期权数大而远期权数小（这也是移动概念的来源），然后计算加权平均数，并以此作为计划期的销售预测值。

例如，某企业产品月销售额如表 5-7 所示。

表 5-7　企业产品的月销售额

月　　份	销售额（万元）
7	520
8	480
9	500
10	560
11	600
12	620

所采取的移动加权平均法步骤如下：

（1）以三个月的季度为时间单位，计算月份销售额时序数据中首尾季度的月平均销售额；

（2）利用初始季度和最后季度来得到平均每月销售变动趋势值；

（3）再以 0.2、0.3 和 0.5 作为系数分别以最后一季度三个月销售额进行加权，并结合平均每月销售变动趋势值，得到未来一个月的销售额预测值。

【例 5-19】 对企业产品销售额进行移动加权平均法预测。

```
sales = [520, 480, 500, 560, 600, 620]

# 初始季度的月平均销售额
avg1 = 0
for i in range(3):
      avg1 += sales[i]
avg1 = avg1 / 3

# 最后季度的月平均销售额
avg2 = 0
for i in range(3):
      avg2 += sales[-i - 1]
avg2 = avg2 / 3

# 利用初始季度和最后季度来得到平均每月销售变动趋势值
increment = (avg2 - avg1) / 3

# 结合权值和平均每月销售变动趋势值，预测未来一个月的销售额
print(sales[-3] * 0.2 + sales[-2] * 0.3 + sales[-1] * 0.5 + increment)
```

输出如下：

```
633.1111111111111
```

代码说明如下：

（1）第一行定义了初始的两个季度六个月的月销售额，以列表存储。

（2）在计算初始季度的月平均销售额部分，通过循环三次获得列表初始三个元素的总和并得到平均值。

（3）在计算最后季度的月平均销售额部分，也是通过循环三次，获得列表最后三个元素的总和并得到平均值。这里的列表序号切片使用了负数写法，表示从后往前进行切片。这种写法更为灵活（即最后三个列表元素），可以确保即使有更多中间季度的月份销售额出现，计算结果也不受影响。

（4）最后的预测部分就是按照算法要求，通过加权系数、最后季度三个月销售额和平均每月销售变动趋势值计算销售额预测值。

这个实现方法本身并没有问题，但是写法较死板，最明显的就是初始季度和最后季度的月平均销售额计算代码非常雷同，而且最后的权值计算代码也非常固化。

【例 5-20】 对企业产品销售额进行移动加权平均法预测（合并计算不同季度的月平均销售额）。

```
sales = [520, 480, 500, 560, 600, 620]
weights = [0.2, 0.3, 0.5]

# 所有季度的月平均销售额
avg = [0] * 2
for i in range(2):
        sum = 0
        for j in range(3):
                sum += sales[i * 3 + j]
        avg[i] = sum / 3

# 平均每月销售变动趋势值
increment = (avg[-1] - avg[0]) / 3

# 结合权值和平均每月销售变动趋势值，预测未来一个月的销售额
result = 0
for i in range(3):
        result += sales[i - 3] * weights[i]
print(result + increment)
```

输出内容同上。

这里有三处较大的代码修改，代码说明如下：

（1）在计算所有季度的月平均销售额部分，将初始季度和最后季度的月平均销售额合并成所有季度的月平均销售额计算，虽然只利用了头尾季度，但是算法更简洁。

（2）在计算平均每月销售变动趋势值部分，只利用头尾时间单位的计算差值。

（3）将权值单独提取出来，通过循环方式计算移动加权平均结果，这有助于后期写出更为灵活的代码。

但是，已有的这些代码仍然不能适应更灵活的数据要求。如果有更多季度的月份销售额，长度为 2 的 avg 列表显然将会失效。另外，如果不再用季度作为时间单位，而是用四个月为时间单位并且权值共有 4 个小数，这些都会导致代码修改较大。为此，既要能保证代码的正确性，还要能保证代码的可修改性。

【例 5-21】 对企业产品销售额进行移动加权平均法预测（灵活考虑不同长度的时间单位）。

```
sales = [520, 480, 500, 560, 600, 620]  # 各月的销售量
weights = [0.2, 0.3, 0.5]               # 移动权值
size = len(weights)                     # 表示一个时间单位是一个季度(三个月)

# 所有季度的月平均销售额
avg = [0] * (len(sales) // size)        # 得到几个时间单位
for i in range(len(avg)):               # 计算各时间单位的平均销售额
      sum = 0
      for j in range(size):
            sum += sales[i * size + j]
      avg[i] = sum / size

# 平均每月销售变动趋势值
increment = (avg[-1] - avg[0]) / size

# 结合权值和平均每月销售变动趋势值，预测未来一个月的销售额
result = 0
for i in range(size):                   # 加权计算结果
      result += sales[i - size] * weights[i]
print(result + increment)
```

输出内容同上。

本例代码的优势在于如果改变了初始的销售额和权值，代码并不需要改动即可直接运行得到最终结果，例如将头两行代码替换为

```
sales = [520, 480, 500, 560, 570, 580, 590, 595, 600, 560, 600, 620]  # 各月的销售量
weights = [0.2, 0.3, 0.2, 0.3]                                        # 移动权值
```

则输出为

```
614.0
```

此时就变成了以四个月为时间单位并且共有三个时间单位的预测结果。

甚至可以进一步通过函数封装，实现更复杂的应用场景。例如将移动加权平均预测法封装成 predict()函数，数据输入封装成 inputSeries()函数。该代码可以不断允许用户输入原始月销售额序列和权值序列，直至在询问时不输入 Y 而退出。

【例 5-22】 对企业产品销售额进行移动加权平均法预测（使用函数封装功能）。

```
# 使用移动加权平均法预测数值的函数
def predict(sales, weights):
    size = len(weights)

    avg = [0] * (len(sales) // size)
    for i in range(len(avg)):
```

```
            sum = 0
            for j in range(size):
                  sum += sales[i * size + j]
            avg[i] = sum / size
      increment = (avg[-1] - avg[0]) / size

      result = 0
      for i in range(size):
            result += sales[i - size] * weights[i]
      return result + increment

# 允许不断输入数值的函数,以 0 结束,并以列表返回所有数值
def inputSeries(title):
      arrays = []
      while True:
          num = float(input(title))
          if num == 0:
              break
          arrays.append(num)
      return arrays

# 主代码
while True:
      strs = input('开始计算,确认吗(Y/N)?')
      if strs.upper() == 'Y':
            arrays1 = inputSeries('请输入各月销售额: ')
            arrays2 = inputSeries('请输入各权值系数: ')
            print(predict(arrays1, arrays2))
      else:
            break
```

本例代码允许用户不断输入各月销售额数据和权值系数,直至在提示用户“开始计算,确认吗”时输入除“Y”以外的任何字符结束。

部分输入和输出展示如下:

```
开始计算,确认吗(Y/N)?Y
请输入各月销售额: 520
请输入各月销售额: 480
请输入各月销售额: 500
请输入各月销售额: 560
请输入各月销售额: 600
请输入各月销售额: 620
请输入各月销售额: 0
请输入各个权值系数: 0.2
请输入各个权值系数: 0.3
请输入各个权值系数: 0.5
请输入各个权值系数: 0
633.1111111111111
开始计算,确认吗(Y/N)?
……
```

2. 指数平滑法

指数平滑也称为指数移动平均(Exponential Moving Average,EMA),是以指数式递

减加权的移动平均。各数值的权重随时间指数式递减，越近期的数据权重越高。常用的指数平滑方法有一次指数平滑、二次指数平滑和三次指数平滑等。

一次指数平滑其实就是对历史数据的加权平均，因此只能预测一个未来值，它可以用于任何一种没有明显函数规律但确实存在某种前后关联的时间序列的短期预测。因此，该方法局限性较大，预测值不能反映趋势变动、季节波动等有规律的变动，只适合短期预测，而不适合中长期的预测，而且由于预测值是历史数据的均值，因此与实际序列的变化相比有滞后现象。

【例 5-23】 对企业产品销售额进行一次指数平滑法预测。

```
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
sales = [520, 480, 500, 560, 570, 580, 590, 595, 600, 560, 600, 620]
model = SimpleExpSmoothing(sales)
r1 = model.fit()
pred1 = r1.predict()
print(pred1)
```

输出如下：

```
[619.9999997]
```

这里使用了 statsmodels 库的 SimpleExpSmoothing()方法，采用了类似于常见机器学习的训练和预测方法。由于它只能预测未来的一个值，因此即使增加预测的数量，数值也相等，例如修改 predict()方法，使其指定预测未来四个连续数值，利用 start 和 end 参数可以指定预测范围的上下数量边界。代码如下：

```
pred1 = r1.predict(start = len(sales), end = len(sales) + 3)
```

输出如下：

```
[619.9999997 619.9999997 619.9999997 619.9999997]
```

二次指数平滑是对一次指数平滑的再平滑，适用于具有线性趋势的时间序列。它不能单独地进行预测，必须与一次指数平滑法配合，建立预测的数学模型，然后运用数学模型确定预测值。二次指数平滑一方面考虑了所有的历史数据，另一方面也兼顾了时间序列的变化趋势。具体方法如霍尔特(Holt's)平滑法等。

【例 5-24】 对企业产品销售额进行二次指数平滑法预测。

```
from statsmodels.tsa.holtwinters import Holt
sales = [520, 480, 500, 560, 570, 580, 590, 595, 600, 560, 600, 620]
model = Holt(sales)
r1 = model.fit()
pred1 = r1.predict(start = len(sales), end = len(sales) + 5)
print(pred1)
```

输出如下：

```
[630.15151139 640.23892314 650.32633489 660.41374665 670.5011584 680.58857015]
```

本例使用了 Statsmodels 库的 Holt()方法。输出通过设置 predict 函数的 start 和 end 参数，实现对未来 6 期结果的预测。

三次指数平滑是在二次指数平滑的基础上再进行一次指数平滑，可以进一步消除二

次指数平滑可能还具有的诸如季节效应等时间波动特征。

【例 5-25】 对企业产品销售额进行三次指数平滑法预测。

```
from statsmodels.tsa.holtwinters import ExponentialSmoothing
sales = [520, 480, 500, 560, 570, 580, 590, 595, 600, 560, 600, 620]
model = ExponentialSmoothing(sales, trend='add', seasonal='add', seasonal_periods=4)
r1 = model.fit()
pred1 = r1.predict(start=len(sales), end=len(sales) + 5)
print(pred1)
```

输出如下：

```
[629.40714887 606.07272147 629.40514569 657.73836198 666.44558167 643.11115427]
```

其中 ExponentialSmoothing()方法的参数需要根据数据情况来灵活设定，trend 和 seasonal 参数都可以设置为加法趋势和乘法趋势等不同的趋势计算方法，seasonal_periods 参数设定季节性周期所涵盖的数值个数。

3. 回归分析法

回归分析法是一种相关性预测方法，它假设某种因素与销售量呈现某种函数关系，因此通过拟合计算找出这种函数关系，并利用这些函数关系实现销售量的预测。

这里仍然以前面的销售额预测为例。

【例 5-26】 对企业产品销售额进行回归分析法预测。

```
from sklearn import linear_model
import numpy as np
sales = [520, 480, 500, 560, 570, 580, 590, 595, 600, 560, 600, 620]
model = linear_model.LinearRegression()
x = np.arange(len(sales))
x.resize((len(sales), 1))
model.fit(x, sales)
newX = np.arange(len(sales), len(sales) + 6)
newX.resize((len(newX), 1))
predict_outcome = model.predict(newX)
print(predict_outcome)
```

输出如下：

```
[630.15151515 640.23892774 650.32634033 660.41375291 670.5011655 680.58857809]
```

代码说明如下：

(1) 第四行使用了 Numpy 库提供的 LinearRegression()方法进行线性回归预测。

(2) 第五行代码得到了一个整数序列，每个值对应原始销售额的序号。Numpy 的 arange()方法可以生成一个数组，参数指定数组长度，如此时的 x 数组内容为

```
[ 0  1  2  3  4  5  6  7  8  9  10  11]
```

这里之所以使用这个从 0 开始的整数序列作为 x，即特征列，是因为这个例子中不存在其他特征列。在真实的线性回归预测中，自变量特征列可能有很多。为了实现这种与时间相关的线性预测，将每个数据的序号作为自变量，因此后续即可根据更多的整数来得到时序的更多预测值。

（3）第六行代码将 x 由一维序列转换为二维序列，即

```
[[ 0]
 [ 1]
 [ 2]
  ...
 [10]
 [11]]
```

按照 Numpy 库的计算要求，参与计算需要得到一个多行多列的数据表示，其中的行数代表着样本记录个数，列数代表着特征的数量。因为这里只有一个特征，所以只有一列，但是即使如此，也必须保持多行多列的二维格式。

除了使用 resize()方法外，还可以使用

```
x = x.reshape(-1, 1)
```

得到类似的效果。注意，reshape()方法本身并不改变当前变量，因此需要更新原有变量。同时，这里的 -1 参数表示具体行数数值由计算自动确定。

（4）第八行代码表示准备预测的数值序号，代表了预测值的相关特征值，即

```
[12 13 14 15 16 17]
```

（5）还可以通过进一步查看相关拟合系数来了解回归计算的细节。如：

```
1  from sklearn import datasets, linear_model
2  import numpy as np
3  sales = [520, 480, 500, 560, 570, 580, 590, 595, 600, 560, 600, 620]
4  model = linear_model.LinearRegression()
5  x = np.arange(len(sales))
6  x.resize((len(sales), 1))
7  model.fit(x, sales)
8  print('y = {}x + {}'.format(model.coef_[0], model.intercept_))
```

输出如下：

```
y = 10.087412587412583x + 509.10256410256414
```

输出结果以直观的线性方程来表示。其中，coef_属性表示系数，因为可以有多个自变量，因此即使这里只有一个结果值，也是通过列表结果来返回；intercept_属性表示截距。

5.4.2　成本预测

成本预测和销售预测一样，都可以采取一些类似的时序数值预测方法来完成，很多适合销售预测的方法也一样适合成本预测。这里只讨论一些比较特殊的成本预测方法，如高低点法。该方法利用一元线性关系表示成本费用的发展趋势，选用一定时期的最高业务量和最低业务量的总成本之差与两者业务量之差进行对比，得到一元线性关系的系数和截距，据此实现对未来成本的预测。

例如，某企业产品成本和产量数据如表 5-8 所示。

表 5-8 某企业产品成本和产量数据

月份	产量(件)	固定成本总额(元)	单位变动成本(元)
8	30	20 000	30
9	20	22 000	32
10	25	20 000	30
11	45	23 000	34
12	55	23 400	28

【例 5-27】 利用高低点法预测企业成本(假设 1 月份产量为 60)。

```
import pandas as pd
data = pd.read_csv('产品产量与成本.csv', encoding = 'GBK')
data['总成本'] = data['固定成本总额(元)'] + data['产量(件)'] * data['单位变动成本(元)']
data.sort_values(by = '产量(件)', ascending = False, inplace = True)
max_x = data.head(1)['产量(件)'].values[0]
min_x = data.tail(1)['产量(件)'].values[0]
max_y = data.head(1)['总成本'].values[0]
min_y = data.tail(1)['总成本'].values[0]
a = (max_y - min_y) / (max_x - min_x)
b = min_y - a * min_x
volume = 60
totalCost = a * volume + b
print('总成本为: {},单位成本为: {}'.format(totalCost, totalCost / volume))
```

输出如下:

```
总成本为: 25268.571428571428,单位成本为: 421.1428571428571
```

代码说明如下:

(1) 第三行计算总成本,新增一个数据列,总成本为固定成本和变动成本之和。

(2) 第四行按照产量倒序排序记录,因此最高产量的记录排在第一条,最低产量的记录排在最后一条。

(3) 第五、六行利用 head()和 tail()方法分别获取第一条记录和最后一条记录,即最高产量和最低产量。

(4) 第七、八行再次获取了最高产量和最低产量各自的总成本。

(5) 第九行计算了最高成本和最低成本之差与两者产量之差的比值,以此作为一元线性关系的系数。

(6) 第十行使用上述一元线性关系系数计算了一元线性关系的截距。

(7) 高低点法计算非常直观,最后利用当前月的产量 60,直接计算得到预测总成本。

5.4.3 利润预测

利润指标是衡量企业经营效益的最常见指标,综合反映了企业销售增长和成本控制的能力。不同于销售额预测和成本预测,对利润的影响因素很多,而且不同的因素对利润还会产生不同方向的影响关系。而且,在企业有多种产品在生产的情况下,即使各种因素保持不变,仅仅改变不同产品的生产规模和比例,也会导致利润的变化。

本量利方法也称为边际贡献法，根据利润、销售量、成本之间的数量关系来确定目标利润。例如，某企业生产两种产品，全年固定成本为85万元，具体数据如表5-9所示。

表5-9　某企业产品销售资料

产品	预测销售量(件)	预测销售收入(元)	销售比重	单位边际贡献	边际贡献率
A	14 500	1 595 000	0.5587	50	0.4545
B	7000	1 260 000	0.4413	80	0.4444

在这组数据中，销售比重即为两种产品的预测销售收入占比，可以直接利用预测销售收入计算得出，如A产品的销售比重为1 595 000 / (1 595 000 + 1 260 000) = 0.5587。

单位边际贡献为单位销售收入减去单位变动成本，B产品高于A产品，说明B产品对利润贡献较大。边际贡献率为边际贡献在销售收入中所占的百分比，也可以直接利用预测销售量、预测销售收入和单位边际贡献计算得出，即

$$\text{边际贡献率}=\frac{\text{预测销售量}\times\text{单位边际贡献}}{\text{预测销售收入}}$$

如A产品的边际贡献率为14 500 * 50 / 1 595 000 = 0.4545。

本量利方法的关键就是利用销售收入和边际贡献率，即全部边际贡献减去固定成本即为预测的目标利润。具体计算时可以先求得销售比重和边际贡献率的乘积之和，再与销售收入总和相乘后减去固定成本来得到。

【例5-28】 利用本量利方法预测企业利润。

```
1  import pandas as pd
2  data = pd.read_csv('产品销售资料.csv', encoding = 'GBK')
3  data['销售比重 * 边际贡献率'] = data['销售比重'] * data['边际贡献率']
4  totalRevenue = data['预测销售收入(元)'].sum()
5  fixedCost = 850000
6  print('预测利润为: {}'.format((totalRevenue * data['销售比重 * 边际贡献率'].sum()) -
   fixedCost))
```

输出如下：

```
预测利润为: 434872.3938499999
```

由于该数据的很多列都是计算得到，因此也可以直接使用原始数据得到结果。代码如下：

```
1  import pandas as pd
2  data = pd.read_csv('产品销售资料.csv', encoding = 'GBK')
3  data['边际贡献'] = data['预测销售收入(元)'] * data['边际贡献率']
4  fixedCost = 850000
5  print('预测利润为: {}'.format((data['边际贡献'].sum()) - fixedCost))
```

输出如下：

```
预测利润为: 434871.5
```

上述两个结果数值的微小差异源于表格数据中的销售比重精度较低。

5.4.4 资金预测

这里主要结合销售百分比法来说明。该方法一般按照以下两个重要步骤来进行：首先需要确定随着销售额变动而变动的资产和负债项目，这些项目一般都会随着销售额的增长而相应增长。其中部分项目并不存在这样的关系，如固定资产、长期投资、无形资产、长期负债和股东权益等；其次考虑计划期所提取的折旧准备(应扣除计划期用于更新改造的金额)和留存收益两个项目，通常可以作为计划期需要追加资金的内部资金来源。

具体计算方法如下：

预测资金额＝未来销售收入的增长率×(资产项目基期数额－负债项目基期数额)－
折旧摊销额与同期用于更新改造的资金的差额－
销售收入及基期销售净利润率计算的净利润与预计发放股利之差额＋
零星资金开支数额

例如，某公司去年销售收入 1.2 亿元，净利润 480 万元，股利发放率 50%，厂房设备利用已达到饱和状态。该企业去年年底的简化资产负债表如表 5-10 所示。

表 5-10 某企业的简化资产负债表

资产(万元)		负债及所有者权益(万元)	
货币资金	120	应付账款	600
应收账款	400	应交税费	300
存货	2600	长期负债	1310
固定资产净额	4800	股本	5400
无形资产	40	留存收益	350
资产总计	7960	负债及所有者权益总计	7960

如果该企业在今年销售收入增至 1.5 亿元，销售净利率与去年相同，该企业仍按照去年股利发放率支付股利，按照折旧计划提取 60 万元折旧，其中 50%用于设备改造，零星资金需求量增加 30 万元。

【例 5-29】 预测该企业今年需要追加的资金金额。

```
salesRevenue1 = 12000                # 去年销售收入
netProfit1 = 480                     # 去年净利润
payoutRatio = 0.5                    # 去年股利发放率
salesRevenue2 = 15000                # 今年销售收入
netProfit2 = 480                     # 今年净利润
payoutRatio = 0.5                    # 今年股利发放率
depreciation = 60                    # 折旧
equipmentTransformation = 0.5        # 设备改造在折旧中的比例
capitalDemand = 30                   # 零星资金需求量

monetaryFund = 120                   # 货币资金
accountsReceivable = 400             # 应收账款
```

```
stock = 2600                          # 存货
netFixedAssets = 4800                 # 固定资产净额

payableAccounts = 600                 # 应付账款
payableTaxes = 300                    # 应交税费

K = (salesRevenue2 - salesRevenue1) / salesRevenue1          # 未来销售收入的增长率
A = monetaryFund + accountsReceivable + stock + netFixedAssets  # 随着销售额变动的
                                                                # 资产项目基期数额
L = payableAccounts + payableTaxes     # 随着销售额变动的负债项目基期数额
D = depreciation * (1 - equipmentTransformation)  # 计划期提取的折旧摊销额与同期
                                                  # 用于更新改造的资金差额
R = salesRevenue2 * (netProfit1 / salesRevenue1) * (1 - payoutRatio)  # 按计划期
#销售收入及基期销售净利润率计算的净利润与预计发放股利之差额
M = capitalDemand                      # 计划期新增的零星资金开支数额
print('预测追加资金数额为: {}万元'.format(K * (A - L) - D - R + M))
```

输出如下：

```
预测追加资金数额为: 1455.0 万元
```

5.5　财会机器人

所谓财会机器人，就是指利用计算机程序自动实现过去很多需要人工处理的财务工作，如记账凭证会计科目的填写和计算、会计信息系统的自动录入等。常用的方法有以下两种。

第一种是通过程序来模拟人工的键盘和鼠标操作，从而让程序按照既定操作位置和流程快速、精确地进行自动输入，对于要处理的财会文件和财会系统而言，它并不知道这些操作是由人工发出还是由程序发出。由于程序模拟键盘、鼠标操作速度更快，而且适合大批量的重复劳动，因此比人工操作更方便快捷。

第二种是通过程序直接访问财务数据文件，完成过去由人工来完成的数据填写工作。这种方法依赖于按照正确的格式来访问相应的数据文件，比如常见的 Excel 文档就可以很方便地实现此类操作。

两种方法各有特点。第一种方法的好处在于完全屏蔽被操作对象的差异，只是模拟人工操作，适用面更广，甚至可以直接模拟操作财务软件系统。但是该方法需要准确理解按键盘的先后次序和单击鼠标的准确位置，这些操作极大地依赖于当前计算机的软硬件环境，而且一旦发生意外的中断，如弹窗等，可能就会导致异常的发生。第二种方法相对简单快速，直接读取文件并完成计算和再次写入，但是只能处理一些常见的文件格式，而且受限于数据文件的保护等措施，这种方法应用面较窄。

下面的操作以完成一个记账凭证的会计科目填写为例。该记账凭证保存在“记账凭证.xlsx”文件中，内容如图 5-6 所示。

其中有三个工作表，每个工作表都有部分会计科目和借贷方金额。现在的任务是自动将借贷方金额(C11 单元格)以大写汉字金额的方式填写到合计栏(B11 单元格)中。

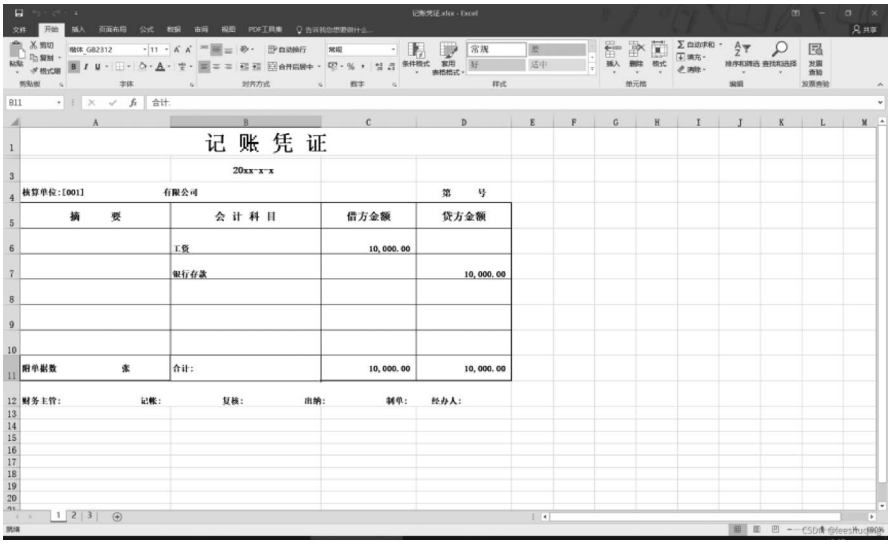

图 5-6 保存记账凭证的 Excel 文件

5.5.1 基于键盘鼠标模拟的实现方法

【例 5-30】 使用键盘鼠标模拟来实现记账凭证的自动填写。

```
import pyautogui
import pyperclip
import time

# 转换为大写汉字金额的函数
def daxie(number):
    numchar = ['零', '壹', '贰', '叁', '肆', '伍', '陆', '柒', '捌', '玖']
    pr = ['圆整', '拾', '佰', '仟', '萬', '拾', '佰', '仟', '亿']
    length = len(str(number))
    result = ''
    for i in str(number):
        length -= 1
        result += ('%s%s' % (numchar[int(i)], pr[length]))
    result = result.replace('零萬', '萬')
    result = result.replace('零仟', '')
    result = result.replace('零佰', '')
    result = result.replace('零拾', '')
    result = result.replace('零圆', '圆')
    return result

# 基本参数设置
time.sleep(5)
pyautogui.PAUSE = 0.5
pyautogui.FAILSAFE = True

# 操作记账凭证的三张工作表
for i in range(3):
```

```
        pyautogui.click(800, 780)
        pyautogui.hotkey('ctrl', 'c')
        pyperclip.copy(daxie(int(float(pyperclip.paste().replace(',', '')))))
        pyautogui.doubleClick(490, 780)
        pyautogui.hotkey('ctrl', 'v')
        pyautogui.hotkey('ctrl', 'pagedown')
        pyautogui.hotkey('ctrl', 'home')
```

为了该程序的正常运行，可以首先使用 Excel 打开该记账凭证文件，最大化显示，然后运行该程序后，再次切换到 Excel 显示界面，即可等待其自动完成。本例目前只在 Excel 中测试通过，建议使用 Excel 打开。可以观察到各工作表的合计栏被不断地填充为正确的大写汉字金额，直至全部填写完毕后程序结束。

代码说明如下：

（1）开头两行导入了两个重要库：第一个库是 Pyautogui 库，它实现了对键盘鼠标的模拟。除了该库外，还有很多类似的库，如 Selenium 等，这些库一般被称为自动化测试库，因为主要的应用场景就是模拟键盘鼠标实现自动化测试软件；第二个库是 Pyperclip，可以实现对操作系统剪切板内容的读写访问。

（2）转换大写汉字金额的函数是一个简单的实现版本，result 返回值保存最终的汉字金额。其中的循环语句可以遍历原始金额的每个数字，并以该数字作为序号到 numchar 列表找到对应的大写汉字，相应的数位汉字是由 pr 列表完成，该列表内容为对应的不同数位应该补充的汉字信息，直接拼接在对应的 numchar 大写汉字后。为了得到更好的输出效果，在输出前还对一些诸如“零萬”“零佰”等特殊表示专门进行了简化处理。为了简化问题，该函数只考虑整数金额。测试代码如下：

```
print(daxie(1230001))
```

输出如下：

```
壹佰贰拾叁萬壹圆整
```

（3）在基本参数设置部分中，使用 Time 库的 sleep()方法设置了暂停 5 秒，主要是为了让用户在运行该程序后，有时间去打开 Excel 文件并使之最大化显示在屏幕上。Pyautogui 库的 PAUSE 属性用于设置延迟，数值越小操作越快，FAILSAFE 值为 False 可以自动防止一般故障，使程序更为稳健。

（4）代码的关键在操作记账凭证的三张工作表部分。这里使用了三次循环，每次处理一个工作表。为了说明清楚，首先需要解释 Pyautogui 库的工作原理。比如运行下面的代码：

```
1  import pyautogui
2  pyautogui.press('a')
```

运行后，可以发现在当前编辑器光标处自动出现了一个字符 a，就像刚才按下了 a 键一样。注意两个问题：一是此时如果大写模式打开，会输入 A；二是如果打开了中文输入法，就会显示似乎以 a 来输入汉字的界面，如图 5-7 所示。

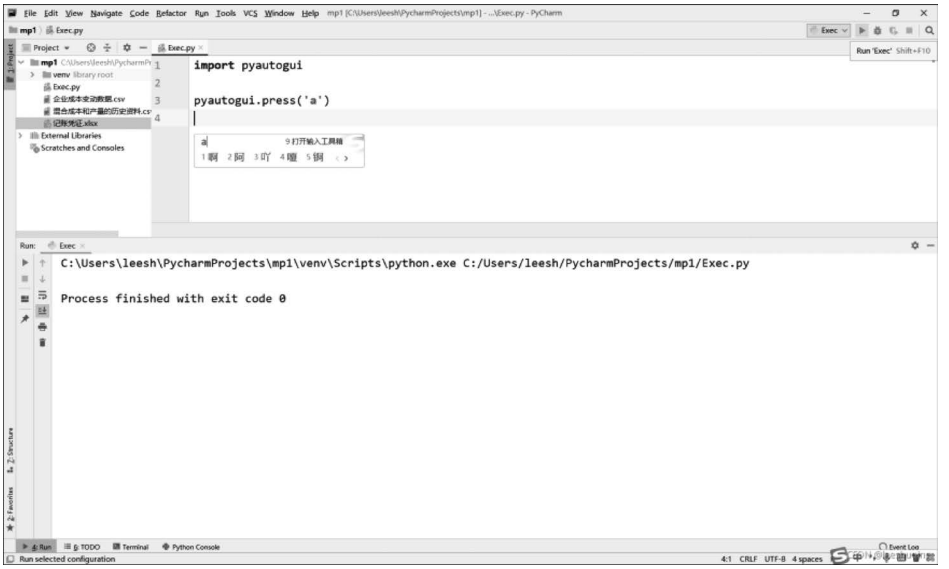

图 5-7　使用 pyautogui 模拟按 a 键

下面进一步了解单击鼠标的操作，代码如下：

```
1  import pyautogui
2  import time
3  time.sleep(5)
4  pyautogui.click(800, 780)
```

运行上述代码后，可以快速打开 Excel 文档界面并最大化，会发现大约 5 秒后(sleep()方法的作用)，C11 单元格被单击了一次。这个过程其实就是 click()函数模拟的，而后面两个整数代表了单击的坐标，这个坐标就是在运行程序的计算机上 Excel 中 C11 单元格中的一个点，位置是从左向右水平 800 像素、从上往下垂直 780 像素处。因此大家如果需要定位到 C11，就需要灵活根据自己计算机的屏幕来确定坐标。

方法很简单，按 PrtSc 键截取当前最大化 Excel 文件的屏幕，然后在开始菜单中打开“画笔”(注意不是画图 3D 等)，直接按 Ctrl＋V 组合键粘贴，再按 Esc 键取消选择。此时在图片上移动鼠标，会发现左下角有两个数字在变化，那就是坐标，如图 5-8 所示。此时即可选中位于 C11 单元格的任意一个点来查看坐标即可。

有一种可能性，那就是屏幕相对较小，不能直接看到 C11，还需要滚动鼠标才能看到。为了简化练习，可以考虑将当前 Excel 表格内容缩小显示，例如单击 Excel 表格右下角的滑钮缩小，或者折叠工具栏，使得 C11 直接显示在屏幕即可。后续实际操作中，也可以通过模拟不断的翻页达到同样的效果。

有了这些基础知识后，就可以详细了解如何操作记账凭证的三张工作表。

在主循环第一行代码处，由于默认打开的是第一张工作表，在(800，780)像素处单击即可选中当前单元格。

第二行通过 hotkey()方法模拟了按 Ctrl＋C 复制快捷键，此时 C11 单元格的数字金额已被复制到剪切板上。

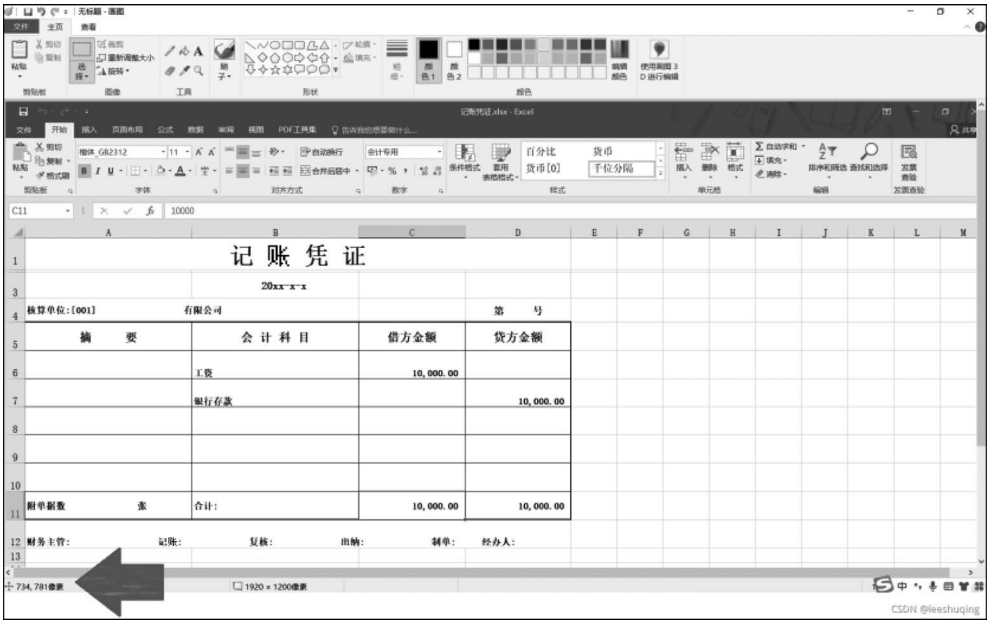

图 5-8　使用画笔软件来获取单击位置的坐标

第三行层次比较多，最里层的代码是

```
pyperclip.paste()
```

即获取当前剪切板的内容，然后去除千分位的逗号分隔符。

之所以先通过 float()转换为浮点数再通过 int()转换为整数，是因为原始数值为浮点数，直接将含有小数点的字符串转换为整数会产生错误。

然后进一步将这个整数金额传入 daxie()函数得到对应的大写汉字金额，最后再次将这个结果通过 Pyperclip 库的 copy()方法复制到剪贴板上。

第四行模拟了双击操作，该位置位于 B11 单元格中。之所以双击，是因为在 Excel 表格中双击单元格可以进入编辑模式，会在已有文本内容最后显示光标并允许用户添加更多内容。

第五行模拟按 Ctrl＋V 键将剪切板内容粘贴到当前 B11 单元格文本后，即大写汉字金额。

第六行模拟按 Ctrl＋PageDown 键，实现翻页到下一工作表。

第七行模拟按 Ctrl＋Home 键，实现将当前工作表定位到左上角的 A1 单元格，以便于对齐整个工作表，防止后面的工作表单元格显示错位。

填写好的 Excel 文档如图 5-9 所示。

5.5.2　基于文件读写的实现方法

【例 5-31】　使用文件读写来实现记账凭证的自动填写。

```
1  import openpyxl
```

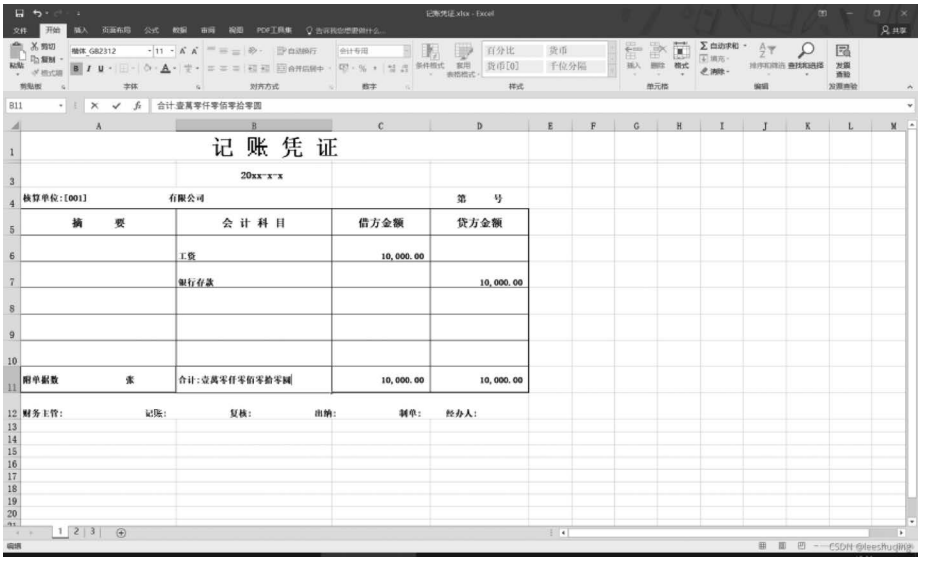

图 5-9 已经自动填写大写汉字金额的记账凭证文档

```
# 转换为大写汉字金额的函数
def daxie(number):
    numchar = ['零', '壹', '贰', '叁', '肆', '伍', '陆', '柒', '捌', '玖']
    pr = ['圆整', '拾', '佰', '仟', '萬', '拾', '佰', '仟', '亿']
    length = len(str(number))
    result = ''
    for i in str(number):
        length -= 1
        result += ('%s%s' % (numchar[int(i)], pr[length]))
    result = result.replace('零萬', '萬')
    result = result.replace('零仟', '')
    result = result.replace('零佰', '')
    result = result.replace('零拾', '')
    result = result.replace('零圆', '圆')
    return result

# 操作记账凭证的三张工作表
wb = openpyxl.load_workbook('记账凭证.xlsx', data_only=True)
for i in range(1, 4):
    sheet = wb[str(i)]
    value1 = sheet.cell(11, 2).value
    value2 = sheet.cell(11, 3).value
    sheet.cell(11, 2).value = value1 + daxie(value2)
wb.save('记账凭证_New.xlsx')
```

本例代码运行相对于例 5-30 更简单，只需要将原始记账凭证文件放在项目目录中，运行后即可看到一个新的“记账凭证_New.xlsx”文件，其中的内容已经更新。

下面主要介绍操作记账凭证的三张工作表的部分，代码说明如下：

(1) 第一行使用 Openpyxl 库的 load_workbook()方法加载记账凭证文件，data_only 参数值为 True 表示只读取数据而忽略文件的计算公式。

（2）第二行进入主循环，循环遍历每张工作表，由于工作表序号从 1 开始，因此 i 的循环数值为 1、2、3。

（3）第三行获取第 i 个工作表。

（4）第四、五行分别获取了 B11 单元格和 C11 单元格的内容，B11 中有“合计”原始内容。

（5）第六行将 C11 单元格的金额转换为大写汉字后再追加到 B11 单元格内容后面。

（6）第七行在结束全部循环处理后将更新内容保存到新的 Excel 文件中。

虽然对于这个例子，直接读写 Excel 文件的方法似乎更简单，但是在更多应用场景下，尤其是一些面向多种软件界面的处理流程，基于键盘、鼠标模拟的方式更为灵活和方便。

习题

1. 观察使用多种非线性回归模型来拟合 5.1.2 节的案例数据。

2. 结合具体企业财务数据，使用多种成本分析方法观察比较企业的成本控制情况。

3. 结合具体企业财务数据，使用多种本量利方法观察比较企业的经营管理情况。

4. 查阅资料，总结函数求导的常用 Python 实现方法。

5. 解决销售预测问题可以使用多种回归预测模型，结合具体企业财务数据（要有一定规模的时序记录数量），使用主流机器学习方法进行预测实验，并评估实验效果。

6. 查阅资料，了解适用于小规模数据集合的其他企业财务指标预测方法，并给出实现代码。

7. 使用 Pyautogui 自动化操作模拟库，完成一些财务管理软件的自动化批量操作，例如批量、依次打开文件，获取数据，并填报到软件系统中。