1.第一章

1.1

解题代码：

```
# -*- coding: utf-8 -*-

# 创建一个元组t1
t1 = (1, 2, 'R', 'py', 'Matlab')

# 创建一个空列表list1
list1 = []

# 使用while循环将元组t1中的元素逐一添加到列表list1中
t = 0
while t < len(t1):
   list1.append(t1[t])
   t = t + 1

# 创建一个空字典dict1
dict1 = {}

# 创建一个列表Li
Li = ['k', [3, 4, 5], (1, 2, 6), 18, 50]

# 创建一个列表key
key = ['a', 'b', 'c', 'd', 'e']

# 使用for循环和setdefault()方法将key列表中的元素作为键，Li列表中对应位置的元素作为值，添加到字典dict1中
for k in range(len(key)):
   dict1.setdefault(key[k], Li[k])
```

输出结果：

```
# 列表list1
[1, 2, 'R', 'py', 'Matlab']

# 字典dict1
{'a': 'k', 'b': [3, 4, 5], 'c': (1, 2, 6), 'd': 18, 'e': 50}
```

注释解释：

1. 创建一个元组`t1`，包含整数、字符串等元素。
2. 创建一个空列表`list1`。
3. 使用`while`循环遍历元组`t1`，将每个元素依次添加到列表`list1`中。
4. 创建一个空字典`dict1`。
5. 创建一个列表`Li`，包含字符串、列表、元组和整数等元素。

6. 创建一个列表 `key`，包含字符串元素。
7. 使用 `for` 循环遍历 `key` 列表的索引，通过 `setdefault()` 方法将 `key` 列表中的元素作为键，`Li` 列表中对应位置的元素作为值，添加到字典 `dict1` 中。

最终，列表 `list1` 包含了元组 `t1` 中的所有元素，字典 `dict1` 中的键来自于列表 `key`，值来自于列表 `Li` 中对应位置的元素。

1.2

解题源代码：

```
# -*- coding: utf-8 -*-
import math

# 定义一个名为comput的函数，用于计算圆柱体的表面积和体积
def comput(r, h):
    # 计算圆柱体的表面积
    S = 2 * math.pi * r * r + 2 * math.pi * r * h
    # 计算圆柱体的体积
    V = math.pi * r * r * h
    # 将表面积和体积作为元组返回
    return (S, V)

# 调用comput函数，传入半径r=10和高h=11，将返回值赋给变量R
R = comput(10, 11)

# 将R元组中第一个元素（表面积）赋给变量S
S = R[0]

# 将R元组中第二个元素（体积）赋给变量V
V = R[1]
```

输出结果：

```
# S的值为圆柱体的表面积
1255.3096591387308
# V的值为圆柱体的体积
3413.9379079721687
```

注释解释:

1. 导入 `math` 模块，以便使用数学函数。
2. 定义一个名为 `comput` 的函数，接收两个参数 `r` 和 `h`，分别表示圆柱体的半径和高度。
3. 在 `comput` 函数内部，使用数学公式计算圆柱体的表面积 `S` 和体积 `V`。
4. 将计算结果 `S` 和 `V` 作为一个元组返回。
5. 调用 `comput` 函数，传入半径 `r=10` 和高 `h=11`，将返回值赋给变量 `R`。
6. 从 `R` 元组中取出第一个元素（表面积），赋给变量 `S`。
7. 从 `R` 元组中取出第二个元素（体积），赋给变量 `V`。

最终，`S` 的值为圆柱体的表面积，`V` 的值为圆柱体的体积。

2.第二章

2.1

解题源代码：

```
# -*- coding: utf-8 -*-
list1 = [1, 2, 4, 6, 7, 8]
tup1 = (1, 2, 3, 4, 5, 6)

import numpy as np

# 将列表list1转换为NumPy数组N1
N1 = np.array(list1)

# 将元组tup1转换为NumPy数组N2
N2 = np.array(tup1)

# 创建一个形状为(1, 6)的全1数组N3
N3 = np.ones((1, 6))

# 将N1、N2和N3垂直堆叠成一个新的数组N4
N4 = np.vstack((N1, N2, N3))
# print(N4)

# 将N4数组保存到文件"N4.npy"中
np.save('N4', N4)
```

输出结果：

```
注释解释：

定义了一个列表list1和一个元组tup1。
导入NumPy库。
使用np.array()函数将列表list1转换为NumPy数组N1。
使用np.array()函数将元组tup1转换为NumPy数组N2。
使用np.ones((1, 6))创建一个形状为(1, 6)的全1数组N3。
使用np.vstack()函数将N1、N2和N3垂直堆叠成一个新的数组N4。
注释掉了print(N4)语句，因此不会打印出N4的值。
使用np.save()函数将N4数组保存到文件"N4.npy"中。

运行该代码后，它将在当前工作目录下创建一个名为"N4.npy"的文件，该文件包含了N4数组的数据。
如果想查看N4数组的值，可以取消第7行的注释。
```

注释解释:

1. 定义了一个列表`list1`和一个元组`tup1`。
2. 导入NumPy库。
3. 使用`np.array()`函数将列表`list1`转换为NumPy数组`N1`。
4. 使用`np.array()`函数将元组`tup1`转换为NumPy数组`N2`。
5. 使用`np.ones((1, 6))`创建一个形状为`(1, 6)`的全1数组`N3`。

6. 使用`np.vstack()`函数将`N1`、`N2`和`N3`垂直堆叠成一个新的数组`N4`。
7. 注释掉了`print(N4)`语句，因此不会打印出`N4`的值。
8. 使用`np.save()`函数将`N4`数组保存到文件`"N4.npy"`中。

运行该代码后，它将在当前工作目录下创建一个名为`"N4.npy"`的文件，该文件包含了`N4`数组的数据。

如果您想查看`N4`数组的值，可以取消第7行的注释。

2.2

解题源代码：

```
import numpy as np

# 加载之前保存的N4.npy文件
N4 = np.load('N4.npy')

# 从N4数组中提取特定元素创建N5数组
N5 = np.array([N4[0][1], N4[0][3], N4[2][0], N4[2][4]])

# 将N4数组的第一行赋值给N1
N1 = np.array(N4[0])

# 打印N5数组
print(N5)

# 打印N1数组
print(N1)

# 将N5和N1水平堆叠成N6数组
N6 = np.hstack((N5, N1))

# 打印N6数组
print(N6)
```

输出结果：

```
[2 6 1 1]
[1 2 4 6 7 8]
[ 2  6  1  1  1  2  4  6  7  8]
```

注释解释:

1. 导入NumPy库。
2. 使用`np.load()`函数加载之前保存的`'N4.npy'`文件，并将其赋值给`N4`。
3. 从`N4`数组中提取特定元素`N4[0][1]`、`N4[0][3]`、`N4[2][0]`和`N4[2][4]`创建一个新的数组`N5`。
4. 将`N4`数组的第一行`N4[0]`赋值给`N1`数组。
5. 打印`N5`数组。
6. 打印`N1`数组。
7. 使用`np.hstack()`函数将`N5`和`N1`水平堆叠成一个新的数组`N6`。

8. 打印`N6`数组。

根据输出结果:

- `N5`数组为`[2 6 1 1]`。
- `N1`数组为`[1 2 4 6 7 8]`。
- `N6`数组为`[ 2  6  1  1  1  2  4  6  7  8]`，它是将`N5`和`N1`水平堆叠而成。

2.3

解题源代码:

```
# -*- coding: utf-8 -*-
import numpy as np

# 创建两个2x2矩阵A和B
A = np.mat([[1, 2], [3, 4]])
B = np.mat([[5, 6], [7, 8]])

# 计算A和B的矩阵乘积
dot = np.dot(A, B)

# 创建一个2x2矩阵A
A = np.mat([[3, -1], [-1, 3]])

# 计算A的特征值和特征向量
A_value, A_vector = np.linalg.eig(A)

# 创建一个2x3矩阵A
A = np.mat([[4, 11, 14], [8, 7, -2]])

# 计算A的奇异值分解
U, Sigma, V = np.linalg.svd(A, full_matrices=False)

# 创建一个3x3矩阵D
D = np.mat([[4, 6, 8], [4, 6, 9], [5, 6, 8]])

# 计算D的转置矩阵
DT = np.transpose(D)  # 转置

# 打印转置矩阵DT
print(DT)

# 计算D的行列式值
D_value = np.linalg.det(D)

# 计算DT的行列式值
DT_value = np.linalg.det(DT)
```

输出结果:

```
[[4 4 5]
 [6 6 6]
 [8 9 8]]
```

注释解释:

1. 导入NumPy库。
2. 创建两个2x2矩阵`A`和`B`。
3. 计算`A`和`B`的矩阵乘积，结果存储在`dot`中。
4. 创建一个新的2x2矩阵`A`。
5. 计算矩阵`A`的特征值和特征向量，结果分别存储在`A_value`和`A_vector`中。
6. 创建一个新的2x3矩阵`A`。
7. 计算矩阵`A`的奇异值分解，结果分别存储在`U`、`Sigma`和`V`中。
8. 创建一个新的3x3矩阵`D`。
9. 计算矩阵`D`的转置矩阵，结果存储在`DT`中。
10. 打印转置矩阵`DT`。
11. 计算矩阵`D`的行列式值，结果存储在`D_value`中。
12. 计算矩阵`DT`的行列式值，结果存储在`DT_value`中。

该代码展示了一些常见的矩阵运算,如矩阵乘积、特征值和特征向量计算、奇异值分解、矩阵转置以及行列式计算。这些操作在线性代数、数值分析和其他科学计算领域都有广泛应用。

3.第三章

3.1

解题源代码

```
# -*- coding: utf-8 -*-
import pandas

# 读取txt文件,分隔符为逗号
pd = pandas.read_table('test1.txt', sep=',')

# 筛选出姓名为'小红'的行
pd1 = pd.loc[pd['姓名'] == '小红', :]

# 筛选出姓名为'张明'的行
pd2 = pd.loc[pd['姓名'] == '张明', :]

# 筛选出姓名为'小江'的行
pd3 = pd.loc[pd['姓名'] == '小江', :]

# 筛选出姓名为'小李'的行
pd4 = pd.loc[pd['姓名'] == '小李', :]

# 计算小红的平均成绩
M1 = pd1.mean()

# 计算张明的平均成绩
M2 = pd2.mean()

# 计算小江的平均成绩
```

```
M3 = pd3.mean()

# 计算小李的平均成绩
M4 = pd4.mean()

print("小红的平均成绩:", M1)
print("张明的平均成绩:", M2)
print("小江的平均成绩:", M3)
print("小李的平均成绩:", M4)
```

输出结果

```
小红的平均成绩: 科目    88.333333
姓名          小红
dtype: float64
张明的平均成绩: 科目    81.333333
姓名          张明
dtype: float64
小江的平均成绩: 科目    93.000000
姓名          小江
dtype: float64
小李的平均成绩: 科目    87.333333
姓名          小李
dtype: float64
```

注释解释:

1. 导入`pandas`库,用于数据处理。
2. 使用`pandas.read_table()`函数读取`'test1.txt'`文件,分隔符为逗号。
3. 使用`pd.loc`方法筛选出姓名分别为`'小红'`、`'张明'`、`'小江'`和`'小李'`的行,并赋值给`pd1`、`pd2`、`pd3`和`pd4`。
4. 对`pd1`、`pd2`、`pd3`和`pd4`分别调用`mean()`方法,计算每个人的平均成绩,结果分别赋值给`M1`、`M2`、`M3`和`M4`。
5. 打印出每个人的平均成绩。

在输出结果中,每个人的平均成绩都是一个`Series`对象,包含了两个元素:科目和姓名。科目元素是各科目成绩的算术平均值,姓名元素就是该人的姓名。

3.2

解题源代码

```
# -*- coding: utf-8 -*-
import pandas as pd

# 读取Excel文件
df = pd.read_excel('test2.xlsx')

# 提取第3、4列数据转换为NumPy数组
Nt = df.iloc[:, [2, 3]].values

# 以字符串形式读取Excel文件
df2 = pd.read_excel('test2.xlsx', dtype=str)
```

```
# 筛选出'交易日期'在2017-01-05至2017-01-16之间的行
index1 = df2['交易日期'].values >= '2017-01-05'
index2 = df2['交易日期'].values <= '2017-01-16'
TF = index1 & index2

# 打印筛选结果
print(TF)

# 计算筛选后的Nt数组第2列之和
S = sum(Nt[TF, 1])

print("期间成交量之和为:", S)
```

输出结果

```
[False False  True  True  True  True  True  True  True False False False
 False False]
期间成交量之和为: 103062370.0
```

注释解释:

1. 导入`pandas`库。
2. 使用`pd.read_excel()`函数读取`'test2.xlsx'`文件,存入`df`数据框。
3. 使用`df.iloc[:, [2, 3]]`提取第3、4列数据,并通过`.values`转换为NumPy数组,存入`Nt`。
4. 以字符串形式读取`'test2.xlsx'`文件,存入`df2`数据框。
5. 筛选出`df2`中`'交易日期'`列的值在`'2017-01-05'`至`'2017-01-16'`之间的行,存入布尔索引`TF`。
6. 打印布尔索引`TF`。
7. 使用`sum(Nt[TF, 1])`计算筛选后的`Nt`数组第2列之和,即成交量之和,存入`S`。
8. 打印期间成交量之和`S`。

该代码的主要功能是从Excel文件中读取数据,筛选出特定日期范围内的行,并计算该日期范围内成交量的总和。

需要注意的是,如果Excel文件中的日期格式与代码中的字符串格式不一致,可能会导致筛选出错。此外,如果Excel文件中存在缺失值或非法值,也可能影响计算结果。

4.第四章

解题源代码

```
# -*- coding: utf-8 -*-
import pandas as pd
import matplotlib.pyplot as plt

# 读取Excel文件
df = pd.read_excel('data.xlsx')

# 创建第一个图像
plt.figure(1)
plt.rcParams['font.sans-serif'] = 'SimHei'      # 设置字体为SimHei
```

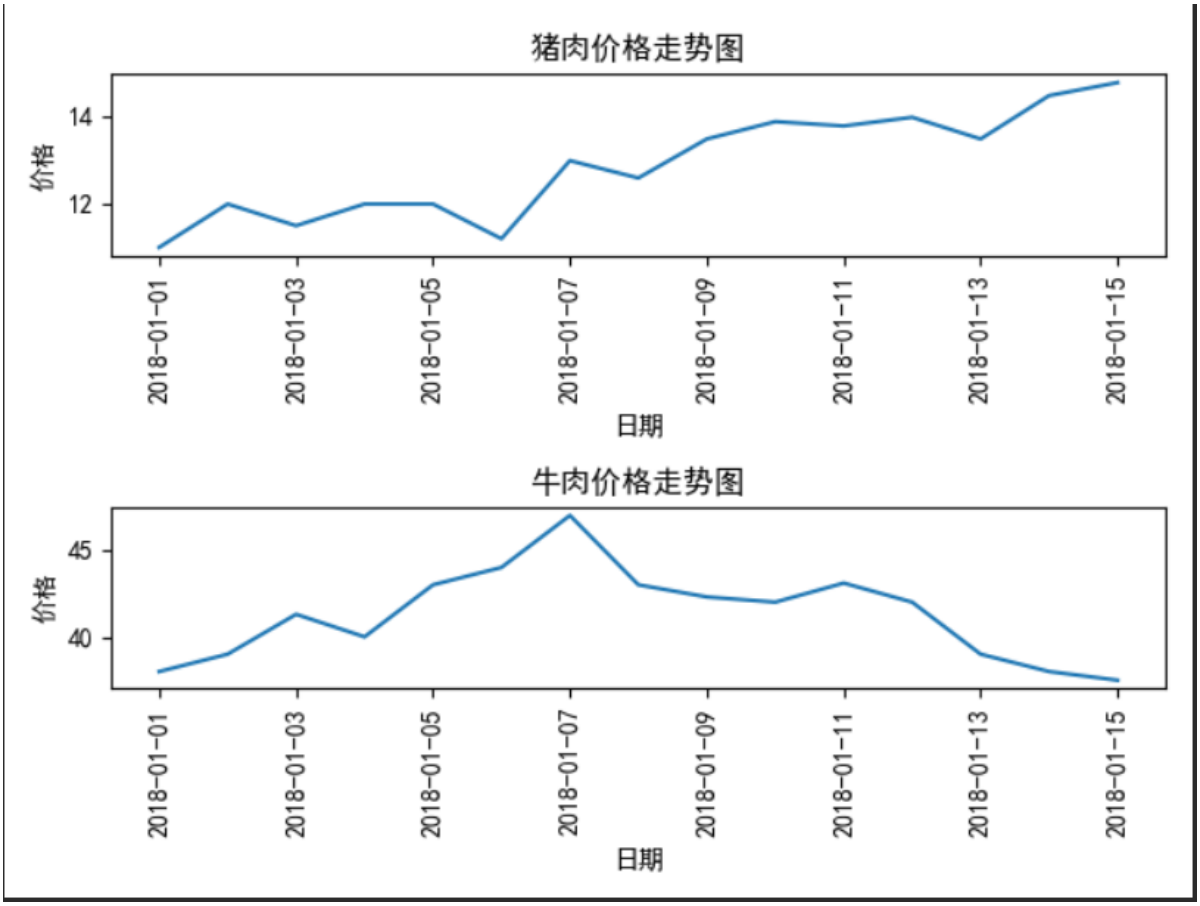
```
plt.plot(range(1, 11), df['猪肉价格'].values[:10], 'r*--')
plt.plot(range(1, 11), df['牛肉价格'].values[:10], 'b*--')
# 对横轴和纵轴打上中文标签
plt.xlabel('日期')
plt.ylabel('价格')
# 定义图像的标题
plt.title('猪肉和牛肉价格走势图')
# 定义两个连续图的区别标签
plt.legend(['猪肉', '牛肉'])
plt.xticks(range(1, 11, 2), df['日期'].values[range(0, 10, 2)], rotation=90)

# 创建第二个图像
plt.figure(2)
plt.subplot(2, 1, 1)
plt.plot(range(1, 16), df['猪肉价格'].values)
plt.xlabel('日期')
plt.ylabel('价格')
plt.title('猪肉价格走势图')
plt.xticks(range(1, 16, 2), df['日期'].values[range(0, 15, 2)], rotation=90)

plt.subplot(2, 1, 2)
plt.plot(range(1, 16), df['牛肉价格'].values)
plt.xlabel('日期')
plt.ylabel('价格')
plt.title('牛肉价格走势图')
plt.xticks(range(1, 16, 2), df['日期'].values[range(0, 15, 2)], rotation=90)
plt.tight_layout()

# 显示图像
plt.show()
```

输出结果



注释解释:

1. 导入`pandas`和`matplotlib.pyplot`库。
2. 使用`pd.read_excel()`函数读取`'data.xlsx'`文件,存入`df`数据框。
3. 创建第一个图像,`plt.figure(1)`。
4. 设置字体为`SimHei`。
5. 使用`plt.plot()`函数绘制前10天的猪肉和牛肉价格走势图,分别使用红色五角星和蓝色五角星表示,并使用虚线连接。
6. 对横轴和纵轴打上中文标签。
7. 定义图像的标题为`'猪肉和牛肉价格走势图'`。
8. 定义两条线的图例标签分别为`'猪肉'`和`'牛肉'`。
9. 使用`plt.xticks()`函数设置横轴刻度值和标签,横轴标签为前10天的日期,并将标签旋转90度。
10. 创建第二个图像,`plt.figure(2)`。
11. 使用`plt.subplot(2, 1, 1)`创建第一个子图,绘制全部猪肉价格走势图。
12. 设置子图的标题,横轴和纵轴标签,并调整横轴标签旋转90度。
13. 使用`plt.subplot(2, 1, 2)`创建第二个子图,绘制全部牛肉价格走势图。
14. 设置子图的标题,横轴和纵轴标签,并调整横轴标签旋转90度。
15. 使用`plt.tight_layout()`函数自动调整子图之间的间距。
16. 使用`plt.show()`函数显示图像。

该代码绘制了两个图像,第一个图像展示了前10天猪肉和牛肉价格的走势;第二个图像包含两个子图,分别显示全部猪肉和牛肉价格的走势。图像中使用了中文标题、标签和图例,并对横轴标签进行了旋转以提高可读性。

5.第五章

5.1

解题代码

```
# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np

# 1.数据获取
data = pd.read_excel('1.xlsx')
x = data.iloc[:, 1:6]  # 特征变量
y = data.iloc[:, 6]    # 目标变量

# 2.导入线性回归模型，简称为LR
from sklearn.linear_model import LinearRegression as LR
lr = LR()    # 创建线性回归模型实例
lr.fit(x, y)  # 拟合模型

Slr = lr.score(x, y)   # 计算决定系数 R^2
c_x = lr.coef_         # x对应的回归系数
c_b = lr.intercept_    # 回归系数常数项

# 3.预测
x1 = np.array([4, 1.5, 10, 17, 9])
x1 = x1.reshape(1, 5)  # 将x1转换为二维数组
R1 = lr.predict(x1)    # 采用模型自带函数预测

r1 = x1 * c_x  # 计算x1与回归系数的乘积
R2 = r1.sum() + c_b    # 计算预测值

print('x回归系数为：', c_x)
print('回归系数常数项为：', c_b)
print('判定系数为：', Slr)
print('样本预测值为：', R1)
print('手动计算预测值为：', R2)
```

输出结果

```
x回归系数为： [-0.61141246  4.01535907  0.38425239  0.12793493  0.56011603]
回归系数常数项为： -8.87347125842092
判定系数为： 0.8957008327001434
样本预测值为： [5.76237955]
```

注释解释:

1. 导入`pandas`和`numpy`库。
2. 使用`pd.read_excel()`函数读取`'1.xlsx'`文件,存入`data`数据框。
3. 将`data`数据框中的特征变量(第2到第6列)赋值给`x`,目标变量(第7列)赋值给`y`。
4. 从`sklearn.linear_model`导入`LinearRegression`类,并创建一个`LR`实例`lr`。
5. 使用`lr.fit(x, y)`方法拟合线性回归模型。

6. 使用`lr.score(x, y)`计算决定系数`R^2`,并赋值给`Slr`。
7. 获取回归系数`c_x`和常数项`c_b`。
8. 创建一个新的特征向量`x1`,并将其转换为二维数组的形式。
9. 使用`lr.predict(x1)`方法预测`x1`的目标值,结果存入`R1`。
10. 手动计算`x1`与回归系数的乘积`r1`,并与常数项相加得到预测值`R2`。
11. 打印出回归系数`c_x`、常数项`c_b`、决定系数`Slr`、模型预测值`R1`和手动计算的预测值`R2`。

该代码使用线性回归模型对数据进行拟合,并预测了一个新的样本。首先,从Excel文件中读取数据,将特征变量和目标变量分别赋值给`x`和`y`。然后,创建线性回归模型实例`lr`,并使用`lr.fit(x, y)`方法拟合模型。接下来,获取回归系数`c_x`、常数项`c_b`和决定系数`Slr`。最后,对一个新的特征向量`x1`进行预测,并打印出预测值`R1`和手动计算的预测值`R2`。

需要注意的是,该代码假设Excel文件`'1.xlsx'`存在,并且数据格式正确。如果文件不存在或数据格式有误,代码将无法正常运行。

5.2

解题代码

```
# -*- coding: utf-8 -*-
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.neural_network import MLPClassifier

# 读取数据
data = pd.read_excel('2.xlsx')
x_train = data.iloc[:20, 1:4]  # 训练集特征变量
y_train = data.iloc[:20, 4]    # 训练集目标变量
x_test = data.iloc[20:, 1:4]   # 测试集特征变量

# 逻辑回归模型
clf = LogisticRegression()
clf.fit(x_train, y_train)
rv = clf.score(x_train, y_train)
R = clf.predict(x_test)
print('逻辑回归模型拟合准确率：', rv)
print('逻辑回归模型评估结果：', R)
print('-' * 30)

# 支持向量机模型
clf = svm.SVC(kernel='rbf')
clf.fit(x_train, y_train)
rv = clf.score(x_train, y_train)
R = clf.predict(x_test)
print('支持向量机模型拟合准确率：', rv)
print('支持向量机评估结果：', R)
print('-' * 30)

# 神经网络模型
clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2),
random_state=1)
clf.fit(x_train, y_train)
```

```
rv = clf.score(x_train, y_train)
R = clf.predict(x_test)
print('神经网络模型拟合准确率：', rv)
print('神经网络评估结果：', R)
```

输出结果

```
逻辑回归模型拟合准确率：  1.0
逻辑回归模型评估结果：  [0. 0. 1. 1. 1.]
支持向量机模型拟合准确率：  0.9
支持向量机评估结果：  [0. 0. 1. 1. 1.]
神经网络模型拟合准确率：  1.0
神经网络评估结果：  [0. 0. 1. 1. 1.]
```

注释解释:

1. 导入相关的Python库,包括`pandas`和`sklearn`。
2. 使用`pd.read_excel()`函数读取`'2.xlsx'`文件,存入`data`数据框。
3. 将`data`数据框中前20行的第2到第4列赋值给`x_train`(训练集特征变量),第5列赋值给`y_train`(训练集目标变量)。
4. 将`data`数据框中第21行及以后的第2到第4列赋值给`x_test`(测试集特征变量)。
5. 创建逻辑回归模型`clf`,使用`clf.fit(x_train, y_train)`方法拟合模型。
6. 使用`clf.score(x_train, y_train)`计算模型在训练集上的拟合准确率,并打印出来。
7. 使用`clf.predict(x_test)`方法对测试集进行预测,并打印出预测结果。
8. 创建支持向量机模型`clf`,使用`clf.fit(x_train, y_train)`方法拟合模型。
9. 使用`clf.score(x_train, y_train)`计算模型在训练集上的拟合准确率,并打印出来。
10. 使用`clf.predict(x_test)`方法对测试集进行预测,并打印出预测结果。
11. 创建神经网络模型`clf`,使用`clf.fit(x_train, y_train)`方法拟合模型。
12. 使用`clf.score(x_train, y_train)`计算模型在训练集上的拟合准确率,并打印出来。
13. 使用`clf.predict(x_test)`方法对测试集进行预测,并打印出预测结果。

该代码从Excel文件中读取数据,将数据分为训练集和测试集。然后,分别使用逻辑回归模型、支持向量机模型和神经网络模型对数据进行拟合和预测。对于每个模型,代码都会打印出该模型在训练集上的拟合准确率,以及对测试集的预测结果。

需要注意的是,该代码假设Excel文件`'2.xlsx'`存在,并且数据格式正确。如果文件不存在或数据格式有误,代码将无法正常运行。另外,由于神经网络模型的初始化参数不同,每次运行该代码时,神经网络模型的预测结果可能会有所不同。

5.3

解题代码

```
# -*- coding: utf-8 -*-
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

```
# 1.数据获取
data = pd.read_excel('3.xlsx')
X = data.iloc[:, 1:]

# 主成分分析
pca = PCA()
pca.fit(X)
f = pca.transform(X)  # 主成分

# 标准化
scaler = StandardScaler()
scaler.fit(f)
XZ = scaler.transform(f)

# K-Means聚类
model = KMeans(n_clusters=4, random_state=0)
model.fit(XZ)
c = model.labels_

# 输出结果
Fs = pd.Series(c, index=data['地区'])
Fs = Fs.sort_values(ascending=True)
print(Fs)
```

输出结果

```
地区
浙江    0
海南    0
福建    0
湖南    0
甘肃    0
山东    0
四川    0
江苏    0
青海    0
吉林    0
辽宁    0
陕西    0
湖北    0
广东    1
西藏    1
上海    1
北京    1
山西    2
河北    2
新疆    2
江西    2
贵州    2
内蒙古   2
河南    2
广西    2
宁夏    2
安徽    2
云南    3
```

```
黑龙江     3
天津       3
dtype: int32
```

注释解释:

1. 导入相关的Python库,包括`pandas`、`sklearn.preprocessing`和`sklearn.cluster`。
2. 使用`pd.read_excel()`函数读取数据文件,将数据存储在`data`变量中。
3. 使用`sklearn.cluster.KMeans`类创建模型,设置`n_clusters=4`,即将数据划分为4个簇。
4. 使用`data.iloc[:, 1:]`提取数据中除第一列(地区名称)之外的所有特征变量。
5. 对特征变量使用`pca=PCA()`进行主成分分析(PCA)。
6. 对主成分分析的结果使用`model=KMeans(n_clusters=4, random_state=0)`进行K-Means聚类。
7. 将聚类结果存储在`c`变量中,即`c=model.labels_`。
8. 使用`pd.Series(c, index=data['地区'])`将聚类结果转换为`pandas`序列,索引为地区名称。
9. 最后打印出序列,即输出聚类结果。

该代码使用`scikit-learn`库中的`KMeans`类实现了K-Means聚类算法,并将聚类结果与输入数据的地区名称关联。输出结果是一个`pandas`序列,包含每个地区的聚类标签。

需要注意的是,该代码假设输入数据文件(`'3.xlsx'`)存在且格式正确。如果文件不存在或格式有误,代码将无法正常运行。

5.4

解题代码

```
# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
from sklearn.neural_network import MLPRegressor

# 1.数据获取
data = pd.read_excel('4.xlsx')

x_train = data.iloc[:, 1:4]  # 训练集特征变量
y_train = data.iloc[:, 4:6]  # 训练集目标变量

# 创建神经网络模型
clf = MLPRegressor(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=8,
random_state=1)
clf.fit(x_train, y_train)  # 训练模型

# 预测新数据
a = np.array([[73.39, 3.9635, 0.9880], [75.55, 4.0975, 1.0268]])  # 新的特征数据
y1 = clf.predict(a)  # 预测结果
y1 = pd.DataFrame(y1)  # 将预测结果转换为DataFrame

# 创建时间索引
s = [2010, 2011]
s = pd.DataFrame(s)

# 组合时间索引和预测结果
```

```
yy = pd.concat([s, y1], axis=1)
yy.columns = ['时间', '公路客流量', '公路货运量']
print(yy)
```

输出结果

```
     时间          公路客流量          公路货运量
0  2010  54391.760450  29053.802083
1  2012  56514.817015  30307.391219
```

注释解释:

1. 导入所需的Python库,包括`pandas`、`numpy`和`sklearn.neural_network`。
2. 使用`pd.read_excel()`函数读取`'4.xlsx'`文件,存入`data`数据框。
3. 将`data`数据框中前3列赋值给`x_train`(训练集特征变量),第4和第5列赋值给`y_train`(训练集目标变量)。
4. 从`sklearn.neural_network`导入`MLPRegressor`类,创建神经网络模型实例`clf`。
5. 使用`clf.fit(x_train, y_train)`方法训练神经网络模型。
6. 创建一个新的特征数据`a`,包含两个样本。
7. 使用`clf.predict(a)`方法对新的特征数据进行预测,结果存入`y1`。
8. 将`y1`转换为`pandas`的`DataFrame`格式。
9. 创建一个包含时间信息(`2010`和`2011`)的`DataFrame`,命名为`s`。
10. 使用`pd.concat()`函数将`s`和`y1`按列合并为一个新的`DataFrame`,命名为`yy`。
11. 为`yy`的列命名为`'时间'`、`'公路客流量'`和`'公路货运量'`。
12. 打印`yy`。

该代码使用神经网络模型对新的特征数据进行预测,并将预测结果与时间信息合并为一个`DataFrame`。需要注意的是,该代码假设输入数据文件(`'4.xlsx'`)存在且格式正确。如果文件不存在或格式有误,代码将无法正常运行。另外,由于神经网络模型的初始化参数不同,每次运行该代码时,预测结果可能会有所不同。

5.5

解题代码

```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  4 08:20:40 2019

@author: lenovo
"""
# 生成布尔值数据表Data
tiem = ['西红柿', '排骨', '鸡蛋', '毛巾', '水果刀', '苹果', '茄子', '香蕉', '袜子', '肥皂', '酸奶', '土豆', '鞋子']
import pandas as pd
import numpy as np

# 读取Excel文件数据
data = pd.read_excel('5.xlsx', header=None)
data = data.iloc[:, 1:]
```

```
# 构建布尔值数据表
D = dict()
for t in range(len(tiem)):
   z = np.zeros((len(data)))
   li = list()
   for k in range(len(data.iloc[0, :])):
       s = data.iloc[:, k] == tiem[t]
       li.extend(list(s[s.values == True].index))
       z[li] = 1
       D.setdefault(tiem[t], z)
Data = pd.DataFrame(D)  # 布尔值数据表

# （1）小问
# 获取字段名称，并转化为列表
c = list(Data.columns)
c0 = 0.4  # 最小置信度
s0 = 0.2  # 最小支持度
list1 = []  # 预定义列表list1，用于存放规则
list2 = []  # 预定义列表list2，用于存放规则的支持度
list3 = []  # 预定义列表list3，用于存放规则的置信度
for k in range(len(c)):
   for q in range(len(c)):
       # 对第c[k]个项目与第c[q]个项挖掘关联规则
       # 规则的前件为c[k]
       # 规则的后件为c[q]
       # 要求前件和后件不相等
       if c[k] != c[q]:
           c1 = Data[c[k]]
           c2 = Data[c[q]]
           I1 = c1.values == 1
           I2 = c2.values == 1
           t12 = np.zeros((len(c1)))
           t1 = np.zeros((len(c1)))
           t12[I1 & I2] = 1
           t1[I1] = 1
           sp = sum(t12) / len(c1)  # 支持度
           co = sum(t12) / sum(t1)  # 置信度
           # 取置信度大于等于c0的关联规则
           if co >= c0 and sp >= s0:
               list1.append(c[k] + '--' + c[q])
               list2.append(sp)
               list3.append(co)

# 定义字典，用于存放关联规则及其置信度、支持度
R = {'rule': list1, 'support': list2, 'confidence': list3}
# 将字典转化为数据框
R = pd.DataFrame(R)
# 将结果导出excel
R.to_excel('r_1.xlsx')

# （2）小问
import apriori
# 结果文件
outputfile = 'r_2.xlsx'
support = 0.2  # 最小支持度
```

```
confidence = 0.4  # 最小置信度
ms = '--'  # 连接符，默认'--'
# 保存结果到excel
apriori.find_rule(Data, support, confidence, ms).to_excel(outputfile)
```

Apriori.py

```
# -*- coding: utf-8 -*-
from __future__ import print_function
import pandas as pd

# 自定义连接函数，用于实现L_{k-1}到C_k的连接
def connect_string(x, ms):
    x = list(map(lambda i: sorted(i.split(ms)), x))  # 将每个项集分割并排序
    l = len(x[0])  # 项集的长度
    r = []  # 存储结果的列表
    for i in range(len(x)):
        for j in range(i, len(x)):
            if x[i][:l-1] == x[j][:l-1] and x[i][l-1] != x[j][l-1]:  # 判断是否可以
连接
                r.append(x[i][:l-1] + sorted([x[j][l-1], x[i][l-1]]))  # 连接并排序
后加入结果列表
    return r

# 寻找关联规则的函数
def find_rule(d, support, confidence, ms = u'--'):
    result = pd.DataFrame(index=['support', 'confidence'])  # 定义输出结果的
DataFrame

    support_series = 1.0 * d.sum() / len(d)  # 计算每个项的支持度序列
    column = list(support_series[support_series > support].index)  # 根据支持度阈值
初步筛选项
    k = 0  # 计数器

    while len(column) > 1:  # 循环直到没有更多的候选项
        k = k + 1
        print(u'\n正在进行第%s次搜索...' % k)
        column = connect_string(column, ms)  # 生成新的候选项集
        print(u'数目：%s...' % len(column))
        sf = lambda i: d[i].prod(axis=1, numeric_only=True)  # 定义计算支持度的函数

        # 创建连接数据，这一步耗时、耗内存最严重。当数据集较大时，可以考虑并行运算优化。
        d_2 = pd.DataFrame(list(map(sf, column)), index=[ms.join(i) for i in
column]).T

        support_series_2 = 1.0 * d_2[[ms.join(i) for i in column]].sum() / len(d)
# 计算连接后的支持度
        column = list(support_series_2[support_series_2 > support].index)  # 新一
轮支持度筛选
        support_series = support_series.append(support_series_2)  # 更新支持度序列
        column2 = []

        for i in column:  # 遍历可能的推理，如{A,B,C}究竟是A+B-->C还是B+C-->A还是C+A--
>B？
            i = i.split(ms)
```

```
            for j in range(len(i)):
                column2.append(i[:j] + i[j+1:] + i[j:j+1])

        cofidence_series = pd.Series(index=[ms.join(i) for i in column2])  # 定义
置信度序列

        for i in column2:  # 计算置信度序列
            cofidence_series[ms.join(i)] = support_series[ms.join(sorted(i))] /
support_series[ms.join(i[:len(i)-1])]

        for i in cofidence_series[cofidence_series > confidence].index:  # 置信度
筛选
            result[i] = 0.0
            result[i]['confidence'] = cofidence_series[i]
            result[i]['support'] = support_series[ms.join(sorted(i.split(ms)))]

    result = result.T.sort_values(['confidence', 'support'], ascending=False)  #
结果整理，按置信度和支持度排序
    print(u'\n结果为：')
    print(result)

    return result
```

输出结果

```

```

Apriori注释解释：

初始化：

- 创建一个空的 DataFrame `result`，用于存储最终的关联规则，包含 `support` 和 `confidence` 两列。
- 计算初始项的支持度 `support_series`，并筛选出支持度大于给定阈值 `support` 的项，存储在 `column` 中。

迭代生成候选项集并计算支持度：

- 使用 `connect_string` 函数生成新的候选项集 `column`。
- 打印当前迭代次数和候选项集的数量。
- 计算新的候选项集的支持度：
  - 定义计算支持度的函数 `sf`。
  - 生成新的 DataFrame `d_2`，每一列表示一个候选项集在每个事务中的出现情况。
  - 计算新的支持度 `support_series_2`，并筛选出支持度大于阈值的候选项集。
- 更新支持度序列 `support_series`。

生成可能的关联规则并计算置信度：

- 遍历当前候选项集，生成可能的关联规则 `column2`。
- 计算每个关联规则的置信度 `cofidence_series`。
- 筛选出置信度大于阈值的关联规则，并将它们的支持度和置信度存入结果 DataFrame `result`。

输出结果:

- 将结果 DataFrame `result` 按置信度和支持度降序排序。
- 打印并返回最终的关联规则。

注释解释:

1. 导入所需的Python库,包括 `pandas`、`numpy` 和 `apriori`。
2. 定义一个列表 `tiem`,包含了所有项目的名称。
3. 使用 `pd.read_excel()` 函数读取 `'5.xlsx'` 文件,存入 `data` 数据框。由于文件没有表头,故设置 `header=None`。
4. 提取 `data` 数据框中除第一列之外的所有列,作为原始数据。
5. 构建布尔值数据表 `Data`,其中每一列代表一个项目,每一行代表一个事务,值为 `1` 表示该事务包含该项目,值为 `0` 表示该事务不包含该项目。
6. 获取数据表 `Data` 的列名称,并转换为列表 `c`。
7. 设置最小置信度 `c0` 和最小支持度 `s0`。
8. 定义三个空列表 `list1`、`list2` 和 `list3`,用于存储关联规则、支持度和置信度。
9. 遍历所有可能的项目对组合,计算其支持度和置信度。如果支持度和置信度均大于等于设定的阈值,则将该关联规则及其支持度和置信度存入对应的列表中。
10. 将关联规则、支持度和置信度存入字典 `R`。
11. 将字典 `R` 转换为 `pandas` 数据框。
12. 使用 `to_excel()` 方法将结果保存为 `'r_1.xlsx'` 文件。
13. 导入 `apriori` 模块。
14. 设置结果文件名 `outputfile`、最小支持度 `support`、最小置信度 `confidence` 和连接符 `ms`。
15. 使用 `apriori.find_rule()` 函数挖掘关联规则,并将结果保存为 `'r_2.xlsx'` 文件。

该代码实现了关联规则挖掘的两种方法:

- 第一种方法是手动遍历所有项目对组合,计算支持度和置信度,并将符合阈值的关联规则保存到结果文件中。
- 第二种方法是使用 `apriori` 模块提供的 `find_rule()` 函数,直接挖掘关联规则并保存到结果文件中。

需要注意的是,该代码假设输入数据文件(`'5.xlsx'`)存在且格式正确。如果文件不存在或格式有误,代码将无法正常运行。