# Python 考试参考讲义：带函数讲解与注释代码

本讲义整合了每个函数/模块的中文解释及可直接复用的完整注释代码，适合考前快速查阅与现场套用。

【主要内容】  
1. 文件头与导入库  
2. 数据读取与清洗（pandas）  
3. 可视化（matplotlib）  
4. 数据库存储与查询（sqlite3）  
5. 神经网络实现与训练（PyTorch）

【1. 文件头与导入库】  
- 指定 UTF-8 编码，防止中文乱码。  
- 导入 pandas、matplotlib、sqlite3、torch 等模块：用于数据处理、可视化、数据库操作与神经网络。  
- 通过 os.environ 设置 KMP\_DUPLICATE\_LIB\_OK（临时解决 OpenMP 冲突，仅救急使用）。  
  
【2. 数据读取与清洗】  
- pd.read\_csv()：读取 CSV；parse\_dates 解析日期列；dayfirst=True 表示“日/月/年”。  
- fillna()：按列填充缺失值；重新计算 Total=Quantity\*Price 保证一致性。  
- dt.to\_period('M')：得到月份周期，便于按月聚合。  
  
【3. 可视化】  
- groupby() 聚合后 .plot(kind='bar'/'line') 绘制柱状/折线图；plt.show() 显示图形。  
  
【4. 数据库存储与查询】  
- sqlite3.connect() 连接数据库；cursor.execute() 执行建表/插入/查询；commit() 提交；close() 关闭。  
- strftime('%Y', Date) 用于按年份筛选（当 Date 以文本形式存储为 'YYYY-MM-DD' 时简单直观）。  
  
【5. PyTorch 神经网络】  
- 通过 nn.Module 子类化定义网络层结构与 forward() 前向传播。  
- 损失函数：nn.MSELoss()；优化器：optim.SGD()；训练流程：清零梯度→前向→计算损失→反向→更新。  
- 若做分类任务，可改为 BCEWithLogitsLoss / CrossEntropyLoss 等更合适的组合。

## 完整可复用的中文注释版代码

# -\*- coding: utf-8 -\*- # 指定源码文件编码，防止中文乱码  
  
import pandas as pd # 数据处理  
import matplotlib.pyplot as plt # 绘图  
import sqlite3 # SQLite 数据库  
import torch # PyTorch 张量运算  
import torch.nn as nn # 神经网络模块  
import torch.optim as optim # 优化器  
import os # 操作系统接口  
  
# 【临时解决方案】允许重复 OpenMP 运行时继续执行（长期不推荐，可能影响稳定与性能）  
os.environ["KMP\_DUPLICATE\_LIB\_OK"] = "TRUE"  
  
# ================= Part 1：数据读取与清洗 =================  
# 读取 CSV 文件并解析日期（dayfirst=True 表示“日/月/年”）  
df = pd.read\_csv('sales\_data.csv', parse\_dates=['Date'], dayfirst=True)  
  
# 按列填充缺失值  
df.fillna({'Product': 'Unknown', 'Quantity': 0, 'Price': 0.0, 'Total': 0.0}, inplace=True)  
  
# 备份原始日期列（既利于时间运算，也方便后续格式化为字符串入库）  
df['Date\_original'] = df['Date']  
  
# 提取月份周期，便于按月汇总  
df['Month'] = df['Date\_original'].dt.to\_period('M')  
  
# 重新计算总额，避免脏数据  
df['Total'] = df['Quantity'] \* df['Price']  
  
# 快速查看清洗结果  
print("Data Loaded and Cleaned:")  
print(df.head())  
  
# ================= Part 1-2：可视化 =================  
# 各产品销量分布（柱状图）  
product\_sales = df.groupby('Product')['Quantity'].sum()  
product\_sales.plot(kind='bar', title='Product Sales Distribution', ylabel='Total Quantity Sold', xlabel='Product')  
plt.show()  
  
# 按月总销售额（折线图）  
monthly\_sales = df.groupby('Month')['Total'].sum()  
monthly\_sales.plot(kind='line', title='Total Sales Over Time', ylabel='Total Sales', xlabel='Month')  
plt.show()  
  
# 将日期转成 'YYYY-MM-DD' 字符串，方便 SQLite 以文本字段存储并使用 strftime 按年份筛选  
df['Date'] = df['Date\_original'].dt.strftime('%Y-%m-%d')  
  
# ================= Part 2：SQLite 数据库存储与查询 =================  
# 连接/创建数据库  
conn = sqlite3.connect('SalesDB.db')  
cursor = conn.cursor()  
  
# 建表（若不存在）  
cursor.execute('''CREATE TABLE IF NOT EXISTS Sales (  
 Date TEXT,  
 Product TEXT,  
 Quantity INTEGER,  
 Price REAL,  
 Total REAL  
)''')  
  
# 将 DataFrame 数据逐行写入数据库（参数化占位符 ? 防注入与类型自动处理）  
for \_, row in df.iterrows():  
 cursor.execute('INSERT OR IGNORE INTO Sales (Date, Product, Quantity, Price, Total) VALUES (?, ?, ?, ?, ?)',  
 (row['Date'], row['Product'], row['Quantity'], row['Price'], row['Total']))  
  
# 提交事务  
conn.commit()  
print("Data inserted into SalesDB.")  
  
# 查询：2023 年总销售额  
cursor.execute("SELECT SUM(Total) FROM Sales WHERE strftime('%Y', Date)='2023'")  
print("Total Sales in 2023:", cursor.fetchone()[0])  
  
# 查询：2023 年各产品销量并按销量降序  
cursor.execute("""  
 SELECT Product, SUM(Quantity) FROM Sales  
 WHERE strftime('%Y', Date)='2023'  
 GROUP BY Product  
 ORDER BY SUM(Quantity) DESC  
""")  
for product, total\_quantity in cursor.fetchall():  
 print(f"Product: {product}, Total Quantity Sold: {total\_quantity}")  
  
# 关闭连接  
conn.close()  
  
# ================= Part 3：PyTorch 简单神经网络 =================  
class SimpleNN(nn.Module):  
 def \_\_init\_\_(self):  
 super(SimpleNN, self).\_\_init\_\_()  
 self.fc1 = nn.Linear(10, 5) # 输入层：10 -> 5  
 self.fc2 = nn.Linear(5, 3) # 隐藏层：5 -> 3  
 self.fc3 = nn.Linear(3, 1) # 输出层：3 -> 1  
  
 def forward(self, x):  
 x = torch.relu(self.fc1(x)) # ReLU 激活  
 x = torch.relu(self.fc2(x))  
 x = torch.sigmoid(self.fc3(x))# 最后用 Sigmoid（若做分类/回归可按需替换）  
 return x  
  
# 实例化模型、损失函数（MSE）与优化器（SGD）  
model = SimpleNN()  
criterion = nn.MSELoss()  
optimizer = optim.SGD(model.parameters(), lr=0.01)  
  
# 生成模拟训练数据（演示用）  
data = torch.randn(100, 10)  
target = torch.randn(100, 1)  
losses = []  
  
# 训练 20 轮  
for epoch in range(20):  
 optimizer.zero\_grad() # 清空梯度  
 output = model(data) # 前向传播  
 loss = criterion(output, target) # 计算损失  
 loss.backward() # 反向传播  
 optimizer.step() # 参数更新  
 losses.append(loss.item()) # 记录损失数值  
  
# 绘制训练损失曲线  
plt.plot(losses)  
plt.title('Training Loss Over Time')  
plt.xlabel('Epoch')  
plt.ylabel('Loss')  
plt.show()