	题目	答案

1.1.1	#读取数据集 1 分
	data = _____
		data = **pd.read_csv("patient_data.csv")**

	#创建新列'RiskLevel'，根据住院天数判断风险等级 3 分
	_____ = _____(_____, '高风险患者', '低风险患者')
		data['RiskLevel'] = **np.where(data['DaysInHospital']>7**, '高风险患者', '低风险患者')

	#统计不同风险等级的患者数量 2 分
	risk_counts = data_____._____
		risk_counts = data**['RiskLevel'].value_counts()**

	#计算高风险患者占比 1 分
	high_risk_ratio = risk_counts['高风险患者'] / _____
		high_risk_ratio = risk_counts['高风险患者'] / **len(data)**

	#计算低风险患者占比 1 分
	low_risk_ratio = risk_counts['低风险患者'] / _____
		low_risk_ratio = risk_counts['低风险患者'] / **len(data)**

	#根据 BMI 值划分指定区间 4 分 data['BMIRange'] = _____(_____, _____, _____, right=False) # 使用左闭右开区间	data['BMIRange'] = **pd.cut(data['BMI'], bins=bmi_bins, labels=bmi_labels**, right=False)

	#计算每个 BMI 区间中高风险患者的比例 2 分
	bmi_risk_rate = _____(_____)['RiskLevel'].apply(lambda x: (x == '高风险患者').mean())
		bmi_risk_rate = **data.groupby('BMIRange')**['RiskLevel'].apply(lambda x: (x == '高风险患者').mean())

	#统计每个 BMI 区间的患者数量 1 分
	bmi_patient_count = data_____
		bmi_patient_count = data**['BMIRange'].value_counts()**

	#根据年龄值划分指定区间 4 分 data['AgeRange'] = _____(_____, _____, _____, right=False) # 使用左闭右开区间	data['AgeRange'] = **pd.cut(data['Age'], bins=age_bins, labels=age_labels**, right=False)

	#计算每个年龄区间中高风险患者的比例 2 分
	age_risk_rate = _____(_____)['RiskLevel'].apply(lambda x: (x == '高风险患者').mean())
		age_risk_rate = **data.groupby('AgeRange')**['RiskLevel'].apply(lambda x: (x == '高风险患者').mean())

	#统计每个年龄区间的患者数量 1 分
	age_patient_count = data_____
		age_patient_count = data**['AgeRange'].value_counts()**

1.1.2	#读取数据集 2 分
	data = _____
		data = **pd.read_csv("sensor_data.csv")**

	#对传感器类型进行分组，并计算每个组的数据数量和平均值 3 分
	sensor_stats = _____(_____)['Value']._____
		sensor_stats = **data.groupby('SensorType')**['Value'].**agg(['count','mean'])**

	#筛选出温度和湿度数据，然后按位置和传感器类型分组，计算每个组的平均值 2 分 location_stats =	location_stats = data[data['SensorType'].**isin(['Temperature','Humidity'])].groupby(['Locatio**

	data[data['SensorType']._____._____['Value'].mean().unstack()	**n','SensorType'])**['Value'].mean().unstack()
	#标记异常值 3 分 data['is_abnormal'] = _____(((_____) & ((data['Value'] < -10) | (data['Value'] > 50))) | ((_____) & ((data['Value'] < 0) | (data['Value'] > 100))), True, False)	data['is_abnormal'] = **np.where**(((**data['SensorType']=='Temperature'**) & ((data['Value'] < -10) | (data['Value'] > 50))) | ((**data['SensorType']=='Humidity'**) & ((data['Value'] < 0) | (data['Value'] > 100))), True, False)

	#输出异常值数量 2 分
	print("异常值数量:", data['is_abnormal']._____)
		print("异常值数量:", data['is_abnormal'].**sum()**)

	#填补缺失值 #使用前向填充和后向填充的方法填补缺失值 4 分 data['Value']._____(_____, inplace=True) data['Value']._____(_____, inplace=True)	data['Value'].**fillna(method = 'ffill'**, inplace=True) data['Value'].**fillna(method = 'bfill'**, inplace=True)

	#保存清洗后的数据
	#删除用于标记异常值的列，并将清洗后的数据保存到新的 CSV 文件中 4 分
	cleaned_data = _____(_____=['is_abnormal'])
	_____('cleaned_sensor_data.csv', _____)
		cleaned_data = **data.drop(columns**=['is_abnormal'])
		cleaned_data.to_csv('cleaned_sensor_data.csv', **index = False**)

1.1.3	missing_values = data._____ #数据缺失值统计 2 分
	duplicate_values = data._____ #数据重复值统计 2 分
		missing_values = data.**isnull().sum()**
		duplicate_values = data.**duplicated().sum()**

	data['is_age_valid'] = _____._____(18, 70) #Age 合理性审核 2 分 data['is_income_valid'] = _____ > _____ #Income 合理性审核 2 分 data['is_loan_amount_valid'] = _____ < (____ * 5) #LoanAmount 合理性审核 2 分 data['is_credit_score_valid'] = _____.____(300, 850) #CreditScor 合理性审核 2 分	data['is_age_valid'] = **data['Age'].between**(18, 70) data['is_income_valid'] = **data['Income'] > 2000** data['is_loan_amount_valid'] = **data['LoanAmount'] < (data['Income']** * 5) data['is_credit_score_valid'] = **data['CreditScore'].between**(300, 850)

	# 保存清洗后的数据
	_____._____(_____, index=False)
		cleaned_data.to_csv('cleaned_credit_data.csv', index=False)

1.1.4	#从本地文件中读取数据 2 分
	data = _____
		data = **pandas.read_csv("user_behavior_data.csv")**

	#打印数据的前 5 条记录 2 分
	print(_____)
		print(**data.head()**)

	#处理缺失值（删除） 2 分
	data = _____
		data = **data.dropna()**

	#数据类型转换
	data_____ = _____(int) # Age 数据类型转换为 int 2 分
	data_____ = _____(float) # PurchaseAmount 数据类型转换为 float 2 分
	data_____ = _____(int) # ReviewScore 数据类型转换为 int 2 分
		data**['Age']** = data['Age'].astype(int)
		data**['PurchaseAmount']** = **data['PurchaseAmount'].astype**(float)
		data**['ReviewScore']** = **data['ReviewScore'].astype**(int)

	#处理异常值 2 分
	data = data[(_____._____(18, 70)) &
	(data['PurchaseAmount'] > 0) &
	(_____._____(1, 5))]
		data = data[(**data['Age'].between**(18, 70)) &
		(data['PurchaseAmount'] > 0) &
		(**data['ReviewScore'].between**(1, 5))]

	#数据标准化 data['PurchaseAmount'] = (data['PurchaseAmount'] - _____) / _____ # PurchaseAmount 数据标准化 2 分 data['ReviewScore'] = (data['ReviewScore'] - _____) / _____ # ReviewScore 数据标准化 2 分	data['PurchaseAmount'] = (data['PurchaseAmount'] - **data['PurchaseAmount'].mean()) / data['PurchaseAmount'].std()** data['ReviewScore'] = (data['ReviewScore'] - **data['ReviewScore'].mean()**) / **data['ReviewScore'].std()**

	#保存清洗后的数据 1 分
	_____('cleaned_user_behavior_data.csv', index=False)
		data.to_csv('cleaned_user_behavior_data.csv', index=False)

	#统计每个购买类别的用户数 2 分
	purchase_category_counts = _____._____
		purchase_category_counts = **data['PurchaseCategory'].value_counts()**

	#统计不同性别的平均购买金额 2 分
	gender_purchase_amount_mean = _____(_____)['PurchaseAmount'].mean()
		gender_purchase_amount_mean = **data.groupby('Gender')**['PurchaseAmount'].mean()

	#统计不同年龄段的用户数 2 分 data['AgeGroup'] = pandas._____(_____, right=False)	data['AgeGroup'] = pandas.**cut(data['Age'],bins=bins,labels=labels**, right=False)

1.1.5	# 从本地文件中读取数据 2 分
	data = _____
		data = **pd.read_csv('vehicle_traffic_data.csv')**

	# 打印数据的前 5 条记录 2 分
	print(_____)
		print(**data.head()**)

	# 处理缺失值（删除） 2 分
	data = _____
		data = **data.dropna()**

	# 数据类型转换
	data_____ = _____(int) #Age 数据类型转换为 int 1 分
	data_____ = _____(float) #Speed 数据类型转换为 float 1 分
	data_____ = _____(float) #TravelDistance 数据类型转换为 float 1 分
	data_____ = _____(float) #TravelTime 数据类型转换为 float 1 分
		data**['Age']** = **data['Age'].astype**(int)
		data**['Speed']** = **data['Speed'].astype**(float)
		data**['TravelDistance']** = **data['TravelDistance'].astype**(float)
		data**['TravelTime']** = **data['TravelTime'].astype**(float)

	# 处理异常值 2 分 删除不合理的年龄、车速、行驶距离和行驶时间
	data = data[(_____(18, 70)) &
	(_____(0, 200)) &
	(_____(1, 1000)) &
	(_____(1, 1440))]
		data = data[(**data['Age'].between**(18, 70)) &
		(**data['Speed'].between**(0, 200)) &
		(**data['TravelDistance'].between**(1, 1000)) &
		(**data['TravelTime'].between**(1, 1440))]

	# 保存清洗后的数据 1 分
	_____('cleaned_vehicle_traffic_data.csv', index=False)
		data.to_csv('cleaned_vehicle_traffic_data.csv', index=False)

	# 审核字段合理性 1 分
	unreasonable_data = data[~((_____(18, 70)) &
	(_____(0, 200)) &
	(_____(1, 1000)) &
	(_____(1, 1440)))]
		unreasonable_data = data[~((**data['Age'].between**(18, 70)) &
		(**data['Speed'].between**(0, 200)) &
		(**data['TravelDistance'].between**(1, 1000)) &
		(**data['TravelTime'].between**(1, 1440)))]

	# 统计每种交通事件的发生次数 2 分
	traffic_event_counts = _____
		traffic_event_counts = **data['TrafficEvent'].value_counts()**

	# 统计不同性别的平均车速、行驶距离和行驶时间 2 分
	gender_stats = data._____._____
		gender_stats =
		data.**groupby('Gender').agg({'Speed':'mean','TravelDistance':'mean','TravelTime':'mean'})**

	# 统计不同年龄段的驾驶员数 5 分 data['AgeGroup'] = _____(_____,_____,_____, right=False) age_group_counts = _____	data['AgeGroup'] = **pd.cut(data['Age'],bins=age_bins,labels=age_labels**, right=False) age_group_counts = **data['AgeGroup'].value_counts()**

2.1.1	# 加载数据集并显示数据集的前五行 1 分
	data = _____
	print(_____)
		data = **pd.read_csv('auto-mpg.csv')**
		print(**data.head()**)

	# 检查缺失值并删除缺失值所在的行 2 分
	print(_____._____._____)
		print(**data.isnull().sum()**)
		data = **data.dropna()**

	data = _____	
	# 将 'horsepower' 列转换为数值类型，并（删除）处理转换中的异常值 1 分 data['horsepower'] = _____(data['horsepower'], errors='coerce') data = _____	data['horsepower'] = **pd.to_numeric**(data['horsepower'], errors='coerce') data = **data.dropna(subset=['horsepower'])**

	# 对数值型数据进行标准化处理 1 分
	data[numerical_features] = _____
		data[numerical_features] = **scaler.fit_transform(data[numerical_features])**

	# 选择特征、自变量和目标变量 2 分
	selected_features = _____
	X = _____
	y = _____
		selected_features
		=**['cylinders','displacement','horsepower','weight','acceleration','model year','origin']**
		X = **data[selected_features]** #X 表示二维矩阵
		y = **data['mpg']** #y 表示一维向量

	# 划分数据集为训练集和测试集（训练集占 8 成） 1 分 X_train, X_test, y_train, y_test = _____(_____, random_state=42)	X_train, X_test, y_train, y_test = **train_test_split(X,y,test_size=0.2**,random_state=42)

	# 保存清洗和处理后的数据（不存储额外的索引号） 1 分
	_____('2.1.1_cleaned_data.csv', _____)
		cleaned_data.to_csv('2.1.1_cleaned_data.csv', **index=False**)

2.1.2	#读取一个 Excel 文件，并将读取到的数据存储在变量 data 中
	data = _____
		data = **pd.read_excel('大学生低碳生活行为的影响因素数据集.xlsx')**

	#处理数据集中的缺失值
	initial_row_count = _____ #处理前的数据行数
	data = _____ #删除缺失值所在行
	final_row_count = _____ #处理后的数据行数
		initial_row_count = **data.shape[0]**
		data = **data.dropna()**
		final_row_count = **data.shape[0]**

	#删除重复行
	data = _____
	data[numerical_features] = _____
		data = **data.drop_duplicates()**
		data[numerical_features] = **scaler.fit_transform(data[numerical_features])**

	#选择特征
	selected_features = [_____]
	X = _____
		selected_features =[
		'1.您的性别○男性 ○女性',
		'2.您的年级○大一 ○大二 ○大三 ○大四',
		'3.您的生源地○农村 ○城镇（乡镇） ○地县级城市 ○省会城市及直辖市',
		'4.您的月生活费○≦1,000 元 ○1,001-2,000 元 ○2,001-3,000 元 ○≧3,001 元',
		'5.您进行过绿色低碳的相关生活方式吗?',
		'6.您觉得“低碳”，与你的生活关系密切吗？',
		'7.低碳生活是否会成为未来的主流生活方式？',

		'8.您是否认为低碳生活会提高您的生活质量？'] X = **data[selected_features]**

	# 创建目标变量
	y = _____
		y = **data['低碳行为积极性']**

	# 数据划分（测试集取 20%） X_train, X_test, y_train, y_test = _____(_____, random_state=42)	X_train, X_test, y_train, y_test = **train_test_split(X,y,test_size=0.2**, random_state=42)

	# 合并处理后得数据，并将其保存（保存中不用额外创建索引）
	cleaned_data = _____(_____, axis=1)
	_____('2.1.2_cleaned_data.csv', _____)
		cleaned_data = **pd.concat([X,y]**, axis=1)
		cleaned_data.to_csv('2.1.2_cleaned_data.csv', **index=False**)

2.1.3	# 加载数据
	data = _____
		data = **pd.read_csv('finance 数据集.csv')**

	# 显示前五行的数据 _____	**print(data.head())**

	# 使用 IQR 处理异常值
	Q1 = _____(0.25)
	Q3 = _____(0.75)
	IQR = _____
		Q1 = **data[numeric_cols].quantile**(0.25) #获得每一列中位于 0.25 处的值
		Q3 = **data[numeric_cols].quantile**(0.75) #获得每一列中位于 0.75 处的值
		IQR = **Q3-Q1** #保留中间 50%的范围

	# 移除异常值
	data_cleaned = data[~((data[numeric_cols] < (Q1 - 1.5 * _____)) | (data[numeric_cols] > (Q3 + 1.5 * _____))).any(axis=1)]
		data_cleaned = data[~((data[numeric_cols] < (Q1 - 1.5 * **IQR**)) | (data[numeric_cols] > (Q3 + 1.5 * **IQR**))).any(axis=1)]

	# 检查处理重复值
	duplicates = _____()
		duplicates = **data_cleaned.duplicated**()

	#对数据进行归一化处理
	data_cleaned[numeric_cols] = _____
		data_cleaned[numeric_cols] = **scaler.fit_transform(data_cleaned[numeric_cols])**

	# 设定目标变量
	target_variable = _____
		target_variable = **'SeriousDlqin2yrs'**

	# 定义特征和目标
	X = _____(columns=[_____]) #1 分
	y = _____ #1 分
		X = **data_cleaned.drop**(columns=[**target_variable**])
		y = **data_cleaned[target_variable]**

	# 划分数据（训练集占 80%） X_train, X_test, y_train, y_test = _____(_____, random_state=42)	X_train, X_test, y_train, y_test = **train_test_split(X,y,test_size=0.2**, random_state=42)

	# 保存清洗后的数据到 CSV
	_____(_____, index=False)
		data_cleaned.to_csv(cleaned_file_path, index=False)

2.1.4	# 加载数据集并指定编码为 gbk
	data = _____
		data = **pd.read_csv('medical_data.csv',encoding='gbk')**

	# 查看表结构基本信息
	print(_____)
		print(**data.info()**)

	# 修改列名
	_____(_____, inplace=True)
		data.rename(columns={'病人 ID':'患者 ID'}, inplace=True)

	# 增加诊断延迟和病程列 data['诊断延迟'] = _____.dt.days	data['诊断延迟'] = **(data['诊断日期'] - data['就诊日期'])**.dt.days

	# 删除不合理的数据
	data = _____[(_____ >= 0) & (_____ > 0) & (_____ < 120)]
		data = **data**[(**data['诊断延迟']** >= 0) & (**data['年龄']** > 0) & (**data['年龄']** < 120)]

	# 删除重复值并记录删除的行数
	_____(inplace=True)
		data.drop_duplicates(inplace=True)

	# 对需要归一化的列进行处理
	columns_to_normalize = [_____]
	data[columns_to_normalize] = _____
		columns_to_normalize = **['年龄','体重','身高']**
		data[columns_to_normalize] = **scaler.fit_transform(data[columns_to_normalize])**

	# 绘制柱状图
	_____(_____, stacked=True)
		treatment_outcome_distribution.plot(kind='bar', stacked=True)

	# 绘制散点图
	_____(_____, _____)
		plt.scatter(data['年龄'], data['疾病严重程度'])

	# 保存处理后得数据
	_____(_____, index=False)
		data.to_csv(output_path, index=False)

2.1.5	# 加载数据集
	data = _____
		data = **pd.read_csv('健康咨询客户数据集.csv')**

	# 查看表结构基本信息
	print(_____)
		print(**data.info()**)

	# 显示每一列的空缺值数量
	print(_____)
		print(**data.isnull().sum()**)

	# 删除含有缺失值的行
	data_cleaned = _____
		data_cleaned = **data.dropna()**

	# 转换 'Your age' 列的数据类型为整数类型，并处理异常值 data_cleaned.loc[:, 'Your age'] = _____(_____, errors='coerce')	data_cleaned.loc[:, 'Your age'] = **pd.to_numeric(data_cleaned['Your age']**, errors='coerce')

	data_cleaned.loc[:, 'Your age'] = data_cleaned['Your age']._____	data_cleaned.loc[:, 'Your age'] = data_cleaned['Your age'].**astype(int)**

	# 检查和删除重复值
	data_cleaned = _____
		data_cleaned = **data_cleaned.drop_duplicates()**

	# 归一化 'How do you describe your current level of fitness ?' 列
	data_cleaned[_____] = _____
		data_cleaned[**'How do you describe your current level of fitness ?'**] = **label_encoder.fit_transform(data_cleaned['How do you describe your current level of fitness ?'])**

	# 绘制饼图
	_____(autopct='%1.1f%%', startangle=90, colors=plt.cm.Paired.colors)
		exercise_frequency_counts.plot.pie(autopct='%1.1f%%', startangle=90, colors=plt.cm.Paired.colors)

	# 划分数据（测试集占比 20%） train_data, test_data = _____(_____, random_state=42)	train_data, test_data = **train_test_split(data_filled,test_size=0.2**, random_state=42)

	# 保存处理后的数据
	cleaned_file_path = '_____'
	_____(_____, index=False)
		cleaned_file_path = **'2.1.5_cleaned_data.csv'**
		data_filled.to_csv(cleaned_file_path, index=False)

2.2.1	# 加载数据
	data = _____
		data = **pd.read_csv('finance 数据集.csv')**

	# 显示前五行的数据
	print(_____)
		print(**data.head()**)

	# 分割训练集和测试集（测试集 20%） X_train, X_test, y_train, y_test = _____(_____, random_state=42)	X_train, X_test, y_train, y_test = **train_test_split(X,y,test_size=0.2**,random_state=42)

	# 训练 Logistic 回归模型（最大迭代次数为 1000 次）
	model = _____
		model = **LogisticRegression(max_iter=1000)**

	#训练 Logistic 回归模型 _____	**model.fit(X_train,y_train)**
	# 保存模型 with open('2.2.1_model.pkl', 'wb') as file: pickle._____	with open('2.2.1_model.pkl', 'wb') as file: pickle**.dump(model,file)**

	# 预测并保存结果
	y_pred = _____
		y_pred = **model.predict(X_test)**

	# 分析测试结果
	accuracy = _____
		accuracy = **(y_test==y_pred).mean()**

	# 处理数据不平衡 X_resampled, y_resampled = _____	X_resampled, y_resampled = **smote.fit_resample(X_train,y_train)**
	# 重新训练模型 _____	**model.fit(X_resampled, y_resampled)**

	# 重新预测
	y_pred_resampled = _____
		y_pred_resampled = **model.predict(X_test)**

	# 分析新的测试结果
	accuracy_resampled = _____
		accuracy_resampled =**(y_test==y_pred_resampled).mean()**

2.2.2	# 加载数据集
	df = _____
		df = **pd.read_csv('auto-mpg.csv')**

	# 显示前五行数据
	print(_____)
		print(**df.head()**)

	# 将 'horsepower' 列中的所有值转换为数值类型 df['horsepower'] = _____(_____, errors='coerce')	df['horsepower'] = **pd.to_numeric(df['horsepower']**, errors='coerce')

	# 删除包含缺失值的行
	df = _____
		df = **df.dropna()**

	# 选择相关特征进行建模
	X = _____
	y = _____
		X = **df[['cylinders', 'displacement', 'horsepower', 'weight',' acceleration', 'model year', 'origin']]**
		y = **df['mpg'**

	# 将数据集划分为训练集和测试集（测试集占比 20%） X_train, X_test, y_train, y_test = _____(_____, random_state=42)	X_train, X_test, y_train, y_test = **train_test_split(X, y, test_size=0.2**, random_state=42)

	# 创建包含标准化和线性回归的管道
	pipeline = _____([('scaler', _____),('linreg', _____)])
		pipeline = **Pipeline**([
		('scaler', **StandardScaler()**),
		('linreg', **LinearRegression()**)])

	# 训练模型 _____	**pipeline.fit(X_train, y_train)**
	# 保存训练好的模型 with open('2.2.2_model.pkl', 'wb') as model_file: pickle._____	with open('2.2.2_model.pkl', 'wb') as model_file: pickle.**dump(pipeline,model_file)**

	# 预测并保存结果
	y_pred = _____
	_____('2.2.2_results.txt', index=False)
		y_pred = **pipeline.predict(X_test)**
		results_df.to_csv('2.2.2_results.txt', index=False)

	# 创建随机森林回归模型实例（创建的决策树的数量为 100）
	rf_model = _____(_____, random_state=42)
		rf_model = **RandomForestRegressor(n_estimators=100**, random_state=42)

	# 训练随机森林回归模型 _____	**rf_model.fit(X_train, y_train)**

	# 使用随机森林模型进行预测
	y_pred_rf = _____
		y_pred_rf = **rf_model.predict(X_test)**

	# 保存新的结果
	_____('2.2.2_results_rf.txt', index=False)
		results_rf_df.to_csv('2.2.2_results_rf.txt', index=False)

2.2.3	# 加载数据集
	df = _____
		df = **pd.read_csv('fitness analysis.csv')**

	# 显示前五行数据
	print(_____)
		print(**df.head()**)

	# 选择相关特征进行建模
	X = _____(X) # 将分类变量转为数值变量
		X = **pd.get_dummies**(X)

	# 将年龄段转为数值变量
	y = _____(lambda x: int(x.split(' ')[0])) # 假设年龄段为整数
		y = **df['Your age'].apply**(lambda x: int(x.split(' ')[0]))

	# 将数据集划分为训练集和测试集（测试集占比 20%） X_train, X_test, y_train, y_test = _____(_____, random_state=42)	X_train, X_test, y_train, y_test = **train_test_split(X, y, test_size=0.2**, random_state=42)

	# 创建随机森林回归模型（创建的决策树的数量为 100）
	rf_model = _____(_____, random_state=42)
		rf_model = **RandomForestRegressor(n_estimators=100**, random_state=42)

	# 训练随机森林回归模型 _____	**rf_model.fit(X_train, y_train)**

	# 保存训练好的模型
	with open('2.2.3_model.pkl', 'wb') as model_file:
	  pickle._____
		with open('2.2.3_model.pkl', 'wb') as model_file:
		  pickle.**dump(rf_model, model_file)**

	# 进行结果预测
	y_pred = _____
		y_pred = **rf_model.predict(X_test)**

	# 使用测试工具对模型进行测试，并记录测试结果
	train_score = _____ #训练集分数
	test_score = _____ #测试集分数
	mse = _____ #均方误差
	r2 = _____ #决定系数
		train_score = **rf_model.score(X_train, y_train)** #训练集分数
		test_score = **rf_model.score(X_test, y_test)** #测试集分数
		mse = **mean_squared_error(y_test, y_pred)** #均方误差
		r2 = **r2_score(y_test, y_pred)** #决定系数

	# 运用工具分析算法中错误案例产生的原因并进行纠正
	# 初始化 XGBoost 回归模型（构建 100 棵树）
	xgb_model = _____(_____, random_state=42)
		xgb_model = **xgb.XGBRegressor(n_estimators=100**, random_state=42)

	# 训练 XGBoost 回归模型 _____	**xgb_model.fit(X_train, y_train)**

	# 使用 XGBoost 回归模型在测试集上进行结果预测
	y_pred_xgb = _____
		y_pred_xgb = **xgb_model.predict(X_test)**

	with open('2.2.3_report_xgb.txt', 'w') as xgb_report_file: xgb_report_file.write(f'XGBoost 训练集得分: {_____}\n') xgb_report_file.write(f'XGBoost 测试集得分: {_____}\n') xgb_report_file.write(f'XGBoost 均方误差(MSE): {_____}\n') xgb_report_file.write(f'XGBoost 决定系数(R^2): {_____)}\n')	with open('2.2.3_report_xgb.txt', 'w') as xgb_report_file: xgb_report_file.write(f'XGBoost 训练集得分: {**xgb_model.score(X_train, y_train)**}\n') xgb_report_file.write(f'XGBoost 测试集得分: {**xgb_model.score(X_test, y_test)**}\n') xgb_report_file.write(f'XGBoost 均方误差(MSE): {**mean_squared_error(y_test, y_pred_xgb)**}\n') xgb_report_file.write(f'XGBoost 决定系数(R^2): {**r2_score(y_test, y_pred_xgb)**}\n')

2.2.4	# 加载数据集
	data = _____
		data = p**d.read_excel('大学生低碳生活行为的影响因素数据集.xlsx')**

	# 显示数据集的前五行
	print(_____)
		print(**data.head()**)

	# 删除不必要的列并处理分类变量
	data_cleaned = _____(_____=['序号', '所用时间']) # 删除不必要的列
		data_cleaned = **data.drop(columns=**['序号', '所用时间'])

	# 定义自变量和因变量
	X = _____(_____=_____)
	y = _____
		X = **data_cleaned.drop(columns=[target])**
		y = **data_cleaned[target]**

	# 将数据拆分为训练集和测试集（测试集占 20%） X_train, X_test, y_train, y_test = _____(_____, random_state=42)	X_train, X_test, y_train, y_test = **train_test_split(X, y, test_size=0.2**, random_state=42)

	# 初始化线性回归模型
	model = _____
		model = **LinearRegression()**

	# 训练线性回归模型 _____	**model.fit(X_train, y_train)**
	# 保存训练好的模型 joblib._____	joblib.**dump(model, model_filename)**

	# 进行预测
	y_pred = _____
		y_pred = **model.predict(X_test)**

	# 将结果保存到文本文件中 _____(_____, index=False, sep='\t') # 使用制表符分隔	**results.to_csv(results_filename**, index=False, sep='\t')
	# 将测试结果保存到报告文件中 with open(report_filename, 'w') as f: f.write(f'均方误差: {_____}\n') f.write(f'决定系数: {_____}\n')	with open(report_filename, 'w') as f: f.write(f'均方误差: {**mean_squared_error(y_test, y_pred)**}\n') f.write(f'决定系数: {**r2_score(y_test, y_pred)**}\n')

	# 分析并纠正错误（示例：使用 XGBoost）
	# 初始化 XGBoost 模型（设定树的数量为 1000，学习率为 0.05，每棵树的最大深度为 5，）
	xgb_model = _____(_____, subsample=0.8, colsample_bytree=0.8)
		xgb_model = **XGBRegressor(**
		n_estimators=1000,
		learning_rate=0.05,
		max_depth=5,
		subsample=0.8,
		colsample_bytree=0.8)

	# 训练 XGBoost 模型 _____	**xgb_model.fit(X_train, y_train)**

	# 使用 XGBoost 模型进行预测
	y_pred_xg = _____
		y_pred_xg = **xgb_model.predict(X_test)**

	# 将 XGBoost 测试结果保存到报告文件中 with open(report_filename_xgb, 'w') as f: f.write(f'均方误差: {_____}\n') f.write(f'决定系数: {_____}\n')	with open(report_filename_xgb, 'w') as f: f.write(f'均方误差: {**mean_squared_error(y_test, y_pred_xg)**}\n') f.write(f'决定系数: {**r2_score(y_test, y_pred_xg)**}\n')

2.2.5	# 加载数据集
	df = _____
		df = **pd.read_csv('fitness analysis.csv')**

	# 显示前五行数据
	print(_____)
		print(**df.head()**)

	# 选择相关特征进行建模
	X = _____(X) # 将分类变量转为数值变量
		X = **pd.get_dummies**(X)

	# 设置目标变量
	y = _____
		y = **df['daily_steps']**

	# 将数据集划分为训练集和测试集（测试集占 20%） X_train, X_test, y_train, y_test = _____(_____, random_state=42)	X_train, X_test, y_train, y_test = **train_test_split(X, y, test_size=0.2**, random_state=42)

	# 创建并训练决策树回归模型
	_____ = _____(random_state=42)
		model = **DecisionTreeRegressor**(random_state=42)

	# 训练决策树回归模型 _____	**model.fit(X_train, y_train)**
	# 保存训练好的模型 with open('2.2.5_model.pkl', 'wb') as model_file: pickle._____	with open('2.2.5_model.pkl', 'wb') as model_file: pickle.**dump(model, model_file)**

	# 进行预测
	y_pred = _____
		y_pred = **model.predict(X_test)**

	# 将结果保存到文本文件中
	_____(_____, index=False, sep='\t')
		results.to_csv(results_filename, index=False, sep='\t')

	# 将测试结果保存到报告文件中 with open(_____) as f: f.write(f'均方误差: {_____}\n') f.write(f'平均绝对误差: {_____}\n') f.write(f'决定系数: {_____}\n')	with open(**report_filename**, 'w') as f: f.write(f'均方误差: {**mean_squared_error(y_test, y_pred)**}\n') f.write(f'平均绝对误差: {**mean_absolute_error(y_test, y_pred)**}\n') f.write(f'决定系数: {**r2_score(y_test, y_pred)**}\n')

3.2.1	# 模型加载 2 分
	session = _____
		session = **ort.InferenceSession('resnet.onnx')**

	# 加载图片 2 分
	image = _____('RGB')
		image = **Image.open('img_test.jpg').convert('RGB')**

	# 预处理图片 2 分
	processed_image = _____
		processed_image = **preprocess_image(image)**

	# 进行图片识别 2 分
	output = _____([output_name], {input_name: processed_image})[0]
		output = **session.run**([output_name], {input_name: processed_image})[0]

	# 应用 softmax 函数获取概率 2 分
	probabilities = _____(output, axis=-1)
		probabilities = **scipy.special.softmax**(output, axis=-1)

	# 获取最高的 5 个概率和对应的类别索引 3 分
	top5_idx = _____[-5:][::-1]
	top5_prob = _____
		top5_idx = **np.argsort(probabilities[0])**[-5:][::-1]
		top5_prob = **probabilities[0][top5_idx]**

3.2.2	# 加载 ONNX 模型 2 分
	ort_session = _____
		ort_session = **onnxruntime.InferenceSession('mnist.onnx')**

	# 加载图像 2 分
	image = _____('L') # 转为灰度图
		image = **Image.open('img_test.png').convert**('L')

	#图像预处理
	image = _____((28, 28)) # 调整大小为 MNIST 模型的输入尺寸 2 分
	image_array = ____(___, dtype=np.float32) # 转为 numpy 数组 2 分
	image_array = ____(____, axis=0) # 添加 batch 维度 2 分
	image_array = ____(____, axis=0) # 添加通道维度 2 分
		image = **image.resize**((28, 28))
		image_array = **np.array(image**, dtype=np.float32)
		image_array = **np.expand_dims(image_array**, axis=0)
		image_array = **np.expand_dims(image_array**, axis=0)

	#返回模型输入列表 2 分
	ort_inputs = {_____()[0].name: image_array}
		ort_inputs = {**ort_session.get_inputs**()[0].name: image_array}

	# 执行预测 2 分
	ort_outs = _____(None, ort_inputs)
		ort_outs = **ort_session.run**(None, ort_inputs)

	# 获取预测结果 2 分
	predicted_class = _____
		predicted_class = **np.argmax(ort_outs[0])**

3.2.3	# 定义情感类别与数字标签的映射表 3 分
	emotion_table = {_____}
		emotion_table = {**'neutral':0, 'happiness':1, 'surprise':2, 'sadness':3, 'anger':4, 'disgust':5, 'fear':6, 'contempt':7**}

	# 加载模型 3 分
	ort_session = _____ # 使用 onnxruntime 创建一个会话
		ort_session = **ort.InferenceSession('emotion-ferplus.onnx')**

	# 加载本地图片并进行预处理 3 分
	input_data = _____
		input_data = **preprocess('img_test.png')**

	# 运行模型，进行预测 3 分
	ort_outs = _____(None, _____)
		ort_outs = **ort_session.run**(None, **ort_inputs**)

	# 解码模型输出，找到预测概率最高的情感类别 3 分
	predicted_label = _____(ort_outs[0])
		predicted_label = **np.argmax**(ort_outs[0])

	# 根据预测的标签找到对应的情感名称 3 分
	predicted_emotion = _____[predicted_label]
		predicted_emotion = **list(emotion_table.keys())**[predicted_label]

3.2.4	# 加载模型 2 分
	session = _____
		session = **ort.InferenceSession('flower-detection.onnx')**

	# 加载类别标签 2 分
	with _____ as f:
	  labels = [line.strip() for line in f.readlines()]
		with **open('labels.txt', 'r')** as f:
		  labels = [line.strip() for line in f.readlines()]

	# 加载图片 2 分
	image = _____('RGB')
		image = **Image.open('flower_test.png').convert**('RGB')

	# 预处理图片 2 分
	processed_image = _____
		processed_image = **preprocess_image(image)**

	# 进行图片识别　2 分
	output = _____([output_name], {input_name: processed_image})[0]
		output = **session.run**([output_name], {input_name: processed_image})[0]

	# 应用 softmax 函数获取识别分类后的准确率　2 分
	accuracy = _____(output, axis=-1)
		accuracy = **scipy.special.softmax**(output, axis=-1) #此处写 axis=1 也可以

	# 获取预测的类别索引
	predicted_idx = _____
		predicted_idx = **np.argmax(accuracy)**

	# 获取预测的准确值（转换为百分比）
	prob_percentage = _____
		prob_percentage = **accuracy[0, predicted_idx] * 100**

	# 获取预测的类别标签
	predicted_label = _____
		predicted_label = **labels[predicted_idx]**

3.2.5	# 从标签文件中读取每一行，并去除行首尾的空白字符，得到类别名称列表 2 分
	class_names = [_____ for name in open('voc-model-labels.txt').readlines()]
		class_names = [**name.strip()** for name in open('voc-model-labels.txt').readlines()]

	# 创建 ONNX Runtime 的推理会话，用于运行模型进行推理 2 分
	ort_session = _____('version-RFB-320.onnx')
		ort_session = **ort.InferenceSession**('version-RFB-320.onnx')

	# 获取模型输入的名称 2 分
	input_name = _____()[0].name
		input_name = **ort_session.get_inputs**()[0].name

	# 如果保存结果的目录不存在，则创建该目录 2 分 if not os.path.exists(result_path): 　　os._____	if not os.path.exists(result_path): 　　os.**makedirs(result_path)**

	# 使用 OpenCV 读取图像文件 2 分
	　　orig_image = _____
		orig_image = **cv2.imread(img_path)**

	# 将图像调整为 320x240 的尺寸（符合模型输入的尺寸要求） 2 分
	　　image = _____(_____, (320, 240))
		image = **cv2.resize(image**, (320, 240))

	# 定义图像归一化的均值数组 2 分
	　　image_mean = _____([127, 127, 127])
		image_mean = **np.array**([127, 127, 127])

	# 在第一个维度上扩展一个维度，将图像变为 (1，通道数，高度，宽度)，以符合模型输入的维度要求　1 分
	　　image = _____(image, axis=0)
		image = **np.expand_dims**(image, axis=0)

	#运行模型，输入图像数据，得到模型输出的置信度和边界框　2 分 　　confidences, boxes = _____(None, {input_name: image})	confidences, boxes = **ort_session.run**(None, {input_name: image})