汽车价格预测的多元线性回归实验

一、 课题综述

1.1. 课题说明

小组成员任务划分：王政尧负责对数据进行处理以及探究输入变量参数之间的关联性等；刘依扬负责模型的实现（手写实现回归算法模型）和训练；牛昱琛负责对模型进行进一步的讨论，如探究学习率、正则化等超参数对模型的影响，报告为三人共同完成。三人对该实验项目有同等贡献度。

1.2. 课题目标

本课题使用的数据集来自于数据分析与数据挖掘竞赛 Kaggle 上的汽车价格预测数据集，其中包括不同类型汽车的大量数据，如汽车品牌、引擎尺寸、车型大小等特征变量，以及要预测的汽车价格。课题的目标为对收集到的数据集进行预处理，搭建相应的线性回归模型，完成基于机器学习的回归任务，并能取得较优的准确度，此外，我们还将讨论学习率、正则化等等超参数对模型的影响。

1.3. 课题数据集

本课题使用的数据集为来自 Kaggle 竞赛的“Car Price Prediction Multiple Linear Regression”(https://www.kaggle.com/datasets/hellbuoy/car-price-prediction/data)。该数据集为国内某汽车集团为进入美国市场而收集的美国市场不同类型汽车的大量数据，包括汽车品牌、引擎尺寸、车型大小等等 25 类特征属性，以及汽车价格这一预测变量。数据集共包含 204 个样本，每一个样本为一款独立的车型。目标数据集多元线性回归任务即为根据提供的 25 类特征属性预测不同车型的价格。

二、实验内容

2.1. 数据准备

在 Kaggle 官网下载包含数据集的 csv 文件（见附件 CarPrice_Assignment.csv 和 Data Dictionary – carprices.xls），之后使用 pandas 进行读取并预览前五行的数据，如下图所示（未列出所有属性）：

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	carlength	carwidth
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	168.8	64.1
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	168.8	64.1
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	176.6	66.2
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	176.6	66.4

2.2. 数据预处理

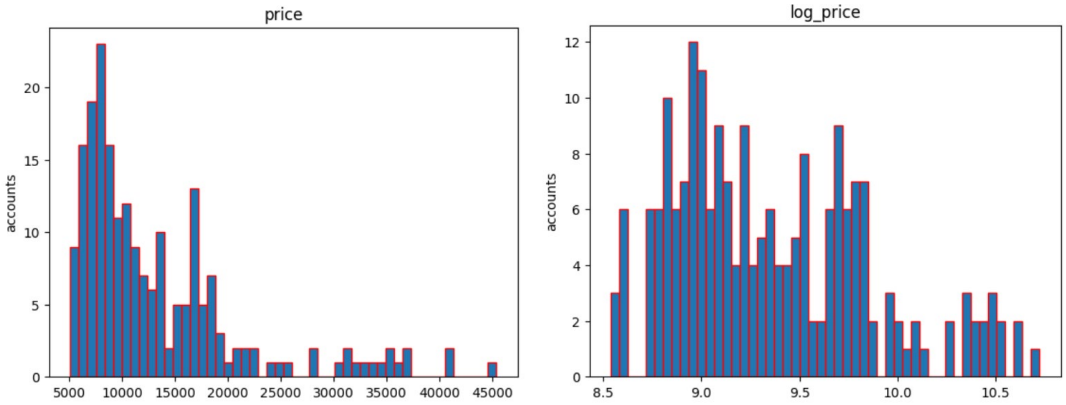
2.2.1. 数据清洗

考虑到数据可能存在不完整等问题，首先对所有特征进行统计，结果如下图所示。可以发现该数据集完整度很高，无缺失值且无明显异常值，无需进一步处理。

```
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    object
 6   carbody           205 non-null    object
 7   drivewheel        205 non-null    object
 8   enginelocation    205 non-null    object
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    object
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64
 22  peakrpm           205 non-null    int64
 23  citympg           205 non-null    int64
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

2.2.2. 数据正态化，纠正偏移值

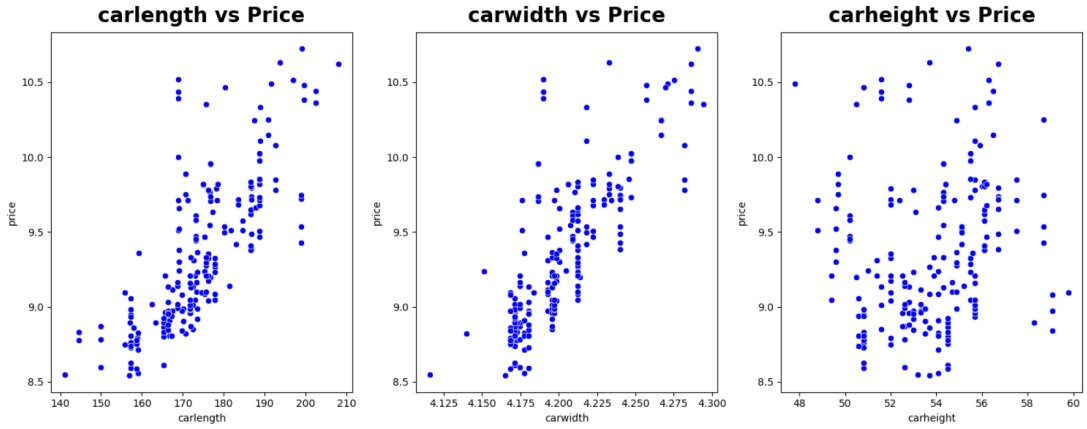
对预测值（price）的分布进行可视化（如下左图所示），可以明显的观察到，price 整体分布稀疏且左偏（理想状态下应呈正态分布），即数据偏移程度较高，若直接带入线性回归模型会导致效果不佳。考虑对 price 进行取对数处理，得到如下右图所示$\log(price+1)$的分布直方图。易见，取对数可以有效地缓解数据分布左偏的问题，便于线性回归模型进行预测。



接着对其他 25 个特征变量进行同样的偏移程度检测和纠正，对偏度大于阈值 0.75 的特征进行处理。

2.2.3. 数据降维

对 25 个自变量分别与因变量进行相关度检验。这里以["carlength", "carwidth", "carheight"]三个特征为例，对三者的分布进行可视化如下：



对比观察三图可以得出，"carlength"和"carwidth"对”price”有显著的正相关关系，而"carheight"分布相对分散，无显著相关性，所以自变量剔除"carheight"这一特征。依照上述方法，最终从 25 个特征中保留 18 个特征作为回归模型的自变量，其中定量数据特征 10 个，分类数据特征 8 个，有效提升了模型的拟合准确度和可靠度。

2.2.4. 分类特征编码

除了数据特征（数据类型为 int64）外，同样需要处理分类特征（数据类型为 objects）。由于线性模型的输入特征必须为数值型变量，对于非数值型特征需要进行编码处理。对于离散型数据的编码，通常有两种方式来实现，分别是标签编码和独热编码。

标签编码（Label Encoder）按类别赋值，数据维数不变，但会给不同类型赋予不同的权重，适合定序数据类型，在本课题处理的线性回归模型中明显不适用；独热编码(One-Hot Encoder)通过创建哑变量的方式进行特征转换，实现数据升维，但会保留原始信息而不增添噪声，本课题对 8 个分类特征采用独热编码方式。

2.2.5. 数据标准化

在多变量线性回归任务中，为消除各评价指标间量纲和数量级的差异、保证结果的可靠性，需要对各指标的原始数据进行特征缩放，因此对编码升维后的数据进行标准化，使得经过处理的数据符合标准正态分布，即均值为 0，标准差为 1。标准化去除了量纲的影响，使得梯度下降可以更好更迅速的收敛，也能在一定程度上提升模型的准确度。处理后数据预览如下图所示（只列出部分值）。最后对处理好的数据集按照 4：1 的比例随机划分为训练集和测试集。

	wheelbase	carlength	carwidth	curbweight	enginesize	boreratio	horsepower	citympg	highwaympg	price
0	-1.801353	-0.426521	-0.851698	-0.014566	0.237430	0.519071	0.365318	-0.646553	-0.546059	9.510149
1	-1.801353	-0.426521	-0.851698	-0.014566	0.237430	0.519071	0.365318	-0.646553	-0.546059	9.711176
2	-0.713827	-0.231513	-0.177732	0.514882	0.791580	-2.404880	1.319046	-0.953012	-0.691627	9.711176
3	0.207276	0.207256	0.153949	-0.420797	-0.386244	-0.517266	0.119435	-0.186865	-0.109354	9.543306
4	0.139468	0.207256	0.248080	0.516807	0.397290	-0.517266	0.468318	-1.106241	-1.273900	9.767152

2.3. 模型搭建

现代的深度学习框架几乎可以自动化地进行所有工作，为了更加深入的了解模型搭建和训练的全部过程，我们将从零开始实现整个方法，包括动手实现模型、损失函数、梯度下降等等。

2.3.1 总体框架

仿照 Tensorflow 的功能，我们构建了 my_model 类，包含 ws、b 两个内置参数，实现了 model.fit()、model.predict()功能，可控学习率、训练轮数以及正则化系数等参数，完成了一个基本的线性回归预测模型。

```
class my_model():
    def __init__(self):
        self.ws = 0
        self.b = 0

    def fit(self, x_train, y_train, op, mylamda = 1.0):
        # 固定随机种子的随机化初始
        np.random.seed(42)
        w_in = np.random.rand(x_train.shape[1])
        b_in = np.random.rand()
        alpha = 0.1  # 学习率
        iters = 100000 # 迭代轮数
        lamda_ = mylamda # 正则化系数

        w_out, b_out = updateParameters(w_in, b_in, x_train, y_train,
alpha, iters, lamda_, op)
        print(f"Final_alpha={alpha}_epochs={iters}_Cost={compute_cost(x_
train, y_train, w_out, b_out, lamda_, op)}")

        self.ws = w_out
        self.b = b_out
```

```
        pass

    def predict(self, x):
        x_pred = np.dot(x,self.ws) + self.b
        return x_pred
```

2.3.2 损失函数 Loss 实现

使用均方误差作为损失函数，并可选择 L1 和 L2 正则化。具体公式如下：

1. 无正则化：

$$J(w,b) = \frac{1}{2m}\sum_{i=0}^{m-1}(f_{w,b}(x^{(i)}) - y^{(i)})^2$$

2. L1 正则化

$$J(w,b) = \frac{1}{2m}\sum_{i=0}^{m-1}(f_{w,b}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=0}^{n-1}|w_j|$$

3. L2 正则化

$$J(w,b) = \frac{1}{2m}\sum_{i=0}^{m-1}(f_{w,b}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=0}^{n-1}w_j^2$$

```
def compute_cost(X, y, w, b, lambda_ = 1, op = 0):

    m,n = X.shape
    cost = 0.

    # 使用矩阵进行并行化计算
    f_wb_i = np.matmul(X, w) + b
    loss = np.square(f_wb_i.T - y)
    cost = loss.sum() / (2 * m)

    # 正则化，注意不包含b
    if op == 0:
        return cost
    else:
        reg_cost = 0
        w_tmp = w
        if op == 1: # L1 正则
            w_tmp = np.abs(w)
        else: # L2 正则
            w_tmp = np.square(w)

        reg_cost = (lambda_/(2*m)) *
w_tmp.sum()

        total_cost = cost + reg_cost
        return total_cost
```

2.3.3 梯度下降算法实现

```
def compute_gradient(w, b, x_train, y_train, lambda_ = 1, op = 0):
    m,n = x_train.shape #(164, 45)

    dj_dw = np.zeros((n,))
```

```
    dj_db = 0.

    f_wb = np.matmul(x_train,w) + b
    err = f_wb.T - y_train #164, 1

    dj_dw = np.array((err.T * x_train)/m)

    if op == 1: # L1 正则
        dj_dw += (lambda_ / (2*m)) * np.sign(w)
    elif op == 2: # L2 正则
        dj_dw += (lambda_ / m) * w
    dj_db = err.sum()/m
    dj_dw = np.squeeze(dj_dw.T)

    return dj_dw, dj_db
```

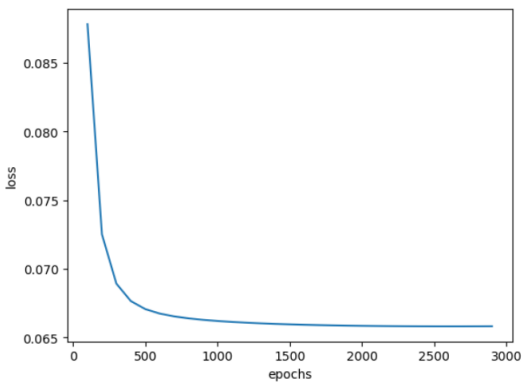
2.3.4　训练函数的实现

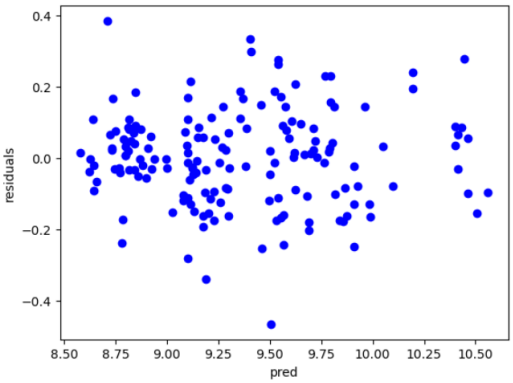
```
def updateParameters(w, b, x_train, y_train, alpha, iters, lamda_=1, op
= 0):
    Epochs_list = []
    J_cost = []
    for i in range(iters):
        dj_dw, dj_db = compute_gradient(w, b, x_train, y_train, lamda_,
op)
        w = w - (alpha * dj_dw) #numpy
        b = b - (alpha * dj_db)

        if i%1000 == 0 and i != 0:
            cost = compute_cost(x_train, y_train, w, b, lamda_, op)
            print(f"Iters:{i}***{cost}",)
            Epochs_list.append(i)
            J_cost.append(cost)
```

2.4. 模型训练测试

设置学习率为 0.1，训练 3000 轮，得到的损失函数曲线如下左图所示。对训练好的模型在测试集上进行测试，评测指标为 MSE（平均平方误差）和R^2，两结果分别为 MSE=0.0401，R2=83.82。观察分析结果的最小二乘残差分布情况（如下右图所示），可以看到残差分布集中在横轴附近，且大体接近正态分布，为比较理想的状态。

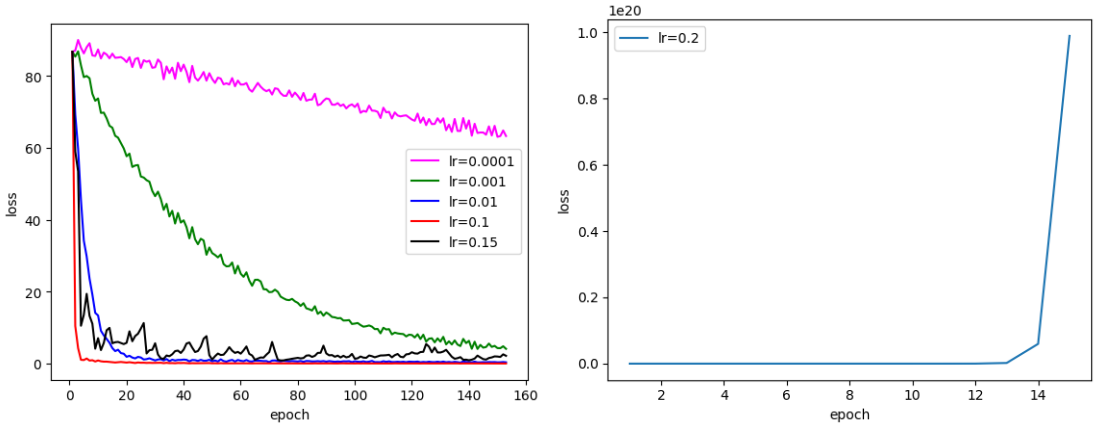




2.5. 分析与讨论

2.5.1 学习率的选择

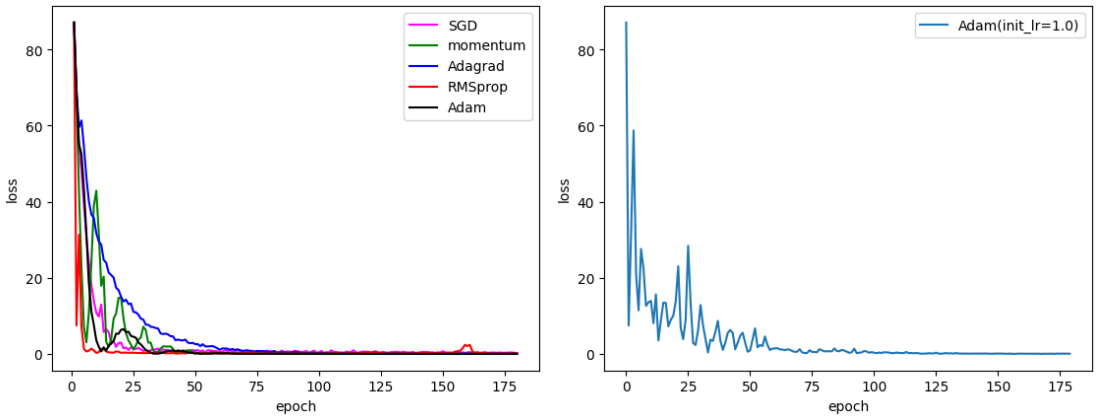
为探究学习率对模型训练的影响，我们通过固定其余超参数而只改变学习率，观察模型梯度下降的过程，如下图所示：



其中，横坐标表示训练轮数，纵坐标为目标函数的当前值，不同颜色的曲线代表不同的学习率。可以发现，当学习率过小时(lr=0.0001)，模型的收敛速度很慢；随着学习率的增大(lr=0.001/0.01/0.1)，模型收敛速度逐渐变快；当学习率较大时(lr=0.15)，梯度下降无法收敛到一个最小值，而反复震荡；当学习率过大时(lr=0.2)，目标函数值迅速发散，导致模型权重值溢出（NaN）。在选择学习率时，可以事先尝试多个范围内的学习率，之后通过观察不同学习率的表现，再进一步确定最佳的学习率。如在本实验中通过初步尝试可以发现，学习率取 0.1 附近的值表现更佳，之后可进一步尝试 0.01 到 0.15 之间的取值。

2.5.2 梯度下降优化算法的比较

使用标准的批量梯度下降算法对目标函数进行优化可能存在着收敛速度慢、陷入局部最优解、容易停滞在鞍点附近无法继续前进以及需要手动选择合适的学习率等问题。为了解决上述问题，研究者们对批量梯度下降算法进行了改进，提出了诸如动量算法、自适应学习率优化算法等优化算法来加速训练过程、提高训练过程的稳定性。本实验中我们尝试了不同的优化算法，结果如下图所示：

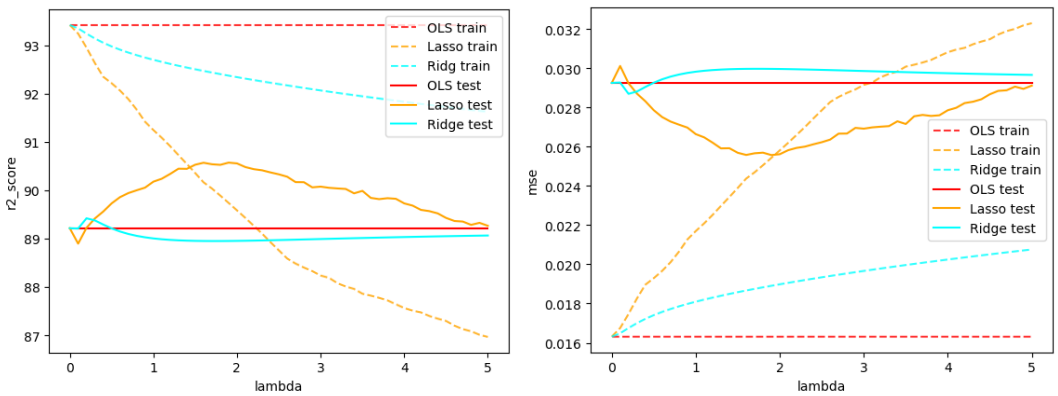


观察结果可以看出，不同的优化算法最终都能够收敛到近乎相同的最小值，而不同的算法由于其特性不同，在下降的过程中损失函数值的变化也有所不同。由于本实验为线性回归问题，模型本身较为简单，难以对这些优化算法进行进一步的比较。此外，右图为采用 Adam 算法并且初始学习率设置为 1.0 的训练过程，Adam 算法可以根据参数的梯度动态调整学习率。这使得在训练过程中能够更好地平衡收敛速度和稳定性，而无需手动选择一个合适的学习率。

2.5.3　正则化的比较

通过调整模型函数 my_model.fit()中的参数 op（控制使用哪一种正则化）与 mylamda（正则项系数），来实现模型的 L1/L2 正则化，从而对 L1 正则化（Lasso 回归）和 L2 正则化（Ridge 回归）进行比较。

分别设置 op 为 0/1/2、在保证模型完全收敛（学习率为 0.1，迭代轮数为 100000）的前提下，在正则项系数为[0.0, 5.0]的范围内定量分析了不同正则化以及正则项系数对结果的影响，结果如下：



可以看出使用 L1 正则的效果要优于无正则项，而使用 L2 正则效果则不

佳。即 L1 正则化的效果为正优化，而 L2 正则化的效果为负优化。进一步分析 L1 正则化，可以发现虽然随着 lambda 的增大，模型在训练集上的误差越来越大，但在测试集上的误差始终比无正则项的效果要好，表明使用 L1 正则有助于减少模型对训练数据的过拟合，使模型更好地泛化到未见过的数据。此外，随着 lambda 的增大，L1 正则在测试集上的效果有一个先变大后变小的趋势，在取值为 2 附近时模型效果最佳，可以合理推测当 lambda 由 0（无正则）变为 2 时，模型的过拟合现象越来越小，因此效果逐渐变好；而当 lambda 由 2 再逐渐增大时，正则化强度过高，使模型出现了欠拟合现象，因此效果逐渐变差。可见，选择哪一种正则化，以及正则项的系数对模型的效果有很大的影响。

通过查阅资料，我们进一步分析了 L1 正则产生正优化，而 L2 正则化产生负优化的可能的原因。L1 正则化倾向于产生稀疏权重向量，即它可以将一些特征的权重置为零，从而选择了模型中最重要的特征，在具有大量冗余特征或噪声特征的数据集上表现得很好。而 L2 正则化通过对所有权重进行平滑处理，不太倾向于生成稀疏解，因此在存在冗余特征的情况下，可能不如 L1 正则化那么有效。分析我们所使用的数据集，可以合理猜测，部分特征如轮胎尺寸，引擎位置等特征与汽车价格之间并没有明显的线性关系，因此使用 L1 正则可以过滤掉这些次要特征，而使模型更多的关注于主要特征。通过观察训练后的权重值，其中部分接近于 0，也印证了我们的猜想。

三、实验总结

在本次线性回归实验中，我们完成了一个完整的机器学习项目。主要工作总结如下：

数据准备和预处理：首先，从 Kaggle 上获取了汽车价格预测的数据集，并使用 Pandas 库对数据进行读取和初步探索。在数据预处理阶段，进行了数据清洗、数据正态化处理、数据降维、分类特征编码和数据标准化等步骤，以确保数据质量和可用性。

模型搭建：为了更深入地了解线性回归模型的内部工作原理，从零开始实现了线性回归模型，包括损失函数、梯度下降算法等。助于我们更好地理解机器学习模型的基本架构。

模型训练和测试：使用搭建好的线性回归模型对训练集进行训练，并在测试集上进行了性能评估。评估指标包括平均平方误差（MSE）和 R^2 值，用于评估模型的准确性和泛化能力。

超参数调优和分析：实验中进行了学习率、正则化方式（L1 和 L2）、正则化系数（lambda）等超参数的调优和分析。通过尝试不同超参数值，深入了解了这些超参数对模型性能的影响。

结果分析与讨论：在实验中，团队还对学习率、不同的优化算法、正则化方式和正则化系数的选择进行了详细分析和讨论。有助于了解模型训练过程中的不同因素对模型性能的影响，以及如何优化模型。

总而言之，本次实验中我们不仅完成了一个完整的汽车价格预测任务，还深入了解了线性回归模型的原理和训练过程，为将来的机器学习项目提供了宝贵的经验和知识。

参考文献

[1] 周志华著.机器学习[M].北京：清华大学出版社,2016

[2]（美）伊恩·古德费洛著；赵申剑，黎彧君，符天凡，李凯译.深度学习[M].北京：人民邮电出版社,2017