西南财经大学统计学院

Statistics School of SWUFE

现代统计分析方法
课程论文

论文题目：Lasso 基础理论及实践应用
——基于共享单车数据集

学生姓名：翟铭阳

学　　号：219025200034

所在学院：统计学院

专　　业：应用统计

成　　绩：

目录

Lasso 基础理论及实践应用

——基于共享单车数据集

1.基础理论

以基因测序为引例，引入稀疏回归。在癌症基因测序问题里，基因维度 p 通常远远大于样本数量 n，造成经典的“大 p 小 n”问题。在经典线性回归中，若 p > n ，最小二乘估计的结果并不唯一，有无穷多个解可使目标函数为零，而且大多数解都会过拟和数据。导致在总体没有共线性的情况下，样本展现出极强的共线性。

由此引入稀疏性与正则化。稀疏性具有三个明显优势：模型可以解释、计算简单，和押注稀疏性（bet on sparsity）。押注稀疏性是指，既然无法有效处理稠密问题，倒不如在稀疏问题上寻找有效的处理方法。

押注稀疏性具体解释为：真实的分布是无法知道的。如果真实分布是稀疏的，使用稀疏回归肯定是正确的；如果真实分布是稠密的（dense），使用稀疏回归虽然有偏差，但最多会欠拟合不会过度拟合，但如果用非稀疏模型拟合，由于参数太多，拟合效果不会好；此外，如果真实分布是稀疏的，我用非稀疏模型去拟合，结果则是毁灭性的。综上所述，我们就押注稀疏性，拟合结果肯定不会有问题。

1.1 经典 Lasso

1.1.1 约束优化角度看 Lasso

Lasso 与 Ridge 回归的出现是为了解决线性回归出现的过拟合问题，这两种回归均在标准线性回归的基础上，通过在损失函数中引入正则化项来达到目的。其中，Lasso 使用 L1 范数，Ridge 使用 L2 范数。

将 Lasso 写成如下约束模式：

$$\underset{\beta_0,\beta}{\text{minimize}} \left\{ \frac{1}{2N} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 \right\}$$
$$\text{subject to } \sum_{j=1}^{p} |\beta_j| \le t.$$

将 Ridge 写成如下约束模式：

$$\underset{\beta_0,\beta}{\text{minimize}} \left\{ \frac{1}{2N} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 \right\}$$
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 \le t^2.$$

从约束优化角度看，图 1 中绿色部分表示 Lasso 与 Ridge 的可行域，橙色椭圆表示残差平方和的等高线，左图为 Lasso 可行域，右图为 Ridge 可行域。约束部分$|\beta_1| + |\beta_2| \leq t$限制 Lasso 的可行域为菱形，而$\beta_1^2 + \beta_2^2 \leq t^2$限制 Ridge 的可行域为圆形。等高线从低到高第一次和可行域相切的点，即是所求的β值。

等高线和菱形区域的切点更有可能在坐标轴上，而等高线和圆形区域的切点在坐标轴上的概率很小。正是由于 Lasso 容易使得部分权重取 0，所以可以用其做特征选择。权重为 0 的特征对回归问题没有贡献，可以直接去掉且模型的输出结果不变。而对于 Ridge，部分特征的权重接近 0 但不为 0，留之无用弃之可惜，但为了回归结果准确性不应舍弃这部分特征。这也解释了为什么 Lasso 更容易使得部分权重取 0，使权重变稀疏；而 Ridge 只能使权重接近 0，很少等于 0，不能达到稀疏效果。

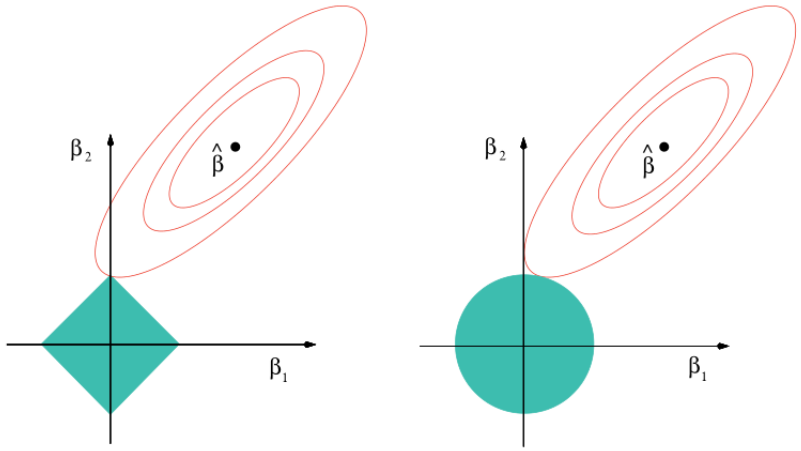

图 1 Lasso 和 Ridge 可行域

从惩罚函数的角度看，上述的惩罚回归模型均为 Bridge 回归的一种特殊形式。下面，给出 Bridge 回归模型：

$$min\left\{\frac{1}{2n}\|Y - X\beta\|_2^2 + \lambda\sum_{j=1}^{p}|\beta_j|^q\right\}(q \geq 0)$$

可以看到：当 q = 1 时，Bridge 回归即为 Lasso 回归；q = 2 时，即为岭回归。当 q 取不同值时，给出二维自变量下 Bridge 回归模型的可行域如图 2 所示：

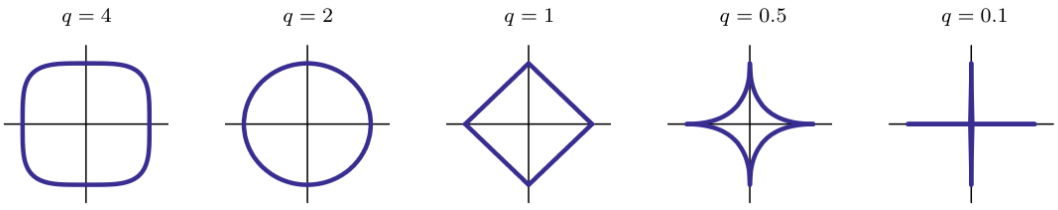

图 2 不同 q 值二维自变量下 Bridge 回归模型的可行域对比

从图 2 可以看出：当 q ≥ 1 时，限制区域是凸的，即 1 是能够使 Bridge 回归问题是凸优化问题的最小 q 值。当 q = 0 时，$\sum_{j=1}^{p}|\beta_j|^q$表示系数向量$\beta$中非零元素的个数，因此求解上式变成了获取最优子集的过程。

1.1.2 Lasso 的估计及局限

图 3 绘制了 Lasso、Ridge 和最小二乘法的参数估计图，图中黑色线代表最小二乘法的参数估计值，直线斜率最高，参数估计值最大；蓝色线代表 Ridge 的参数估计值，直线斜率低于最小二乘法估计，即 Ridge 做到全局缩放；蓝色线代表 Lasso 的参数估计值，在绝对值小于$\lambda/2$时，Lasso 使其直接为 0，绝对值大于$\lambda/2$时，Lasso 使其缩减一定程度。因此 Lasso 满足了稀疏性，可以产生稀疏系数。

由此也可以看出 Lasso 的局限性，即当参数估计值很大时，认为该参数很重要，Lasso 的估计仍然将其缩减，对系数压缩程度较大，导致估计偏差较大，即不满足无偏性。同时， 如果自变量之间存在多重共线时, LASSO 的估计效果会变得很差。

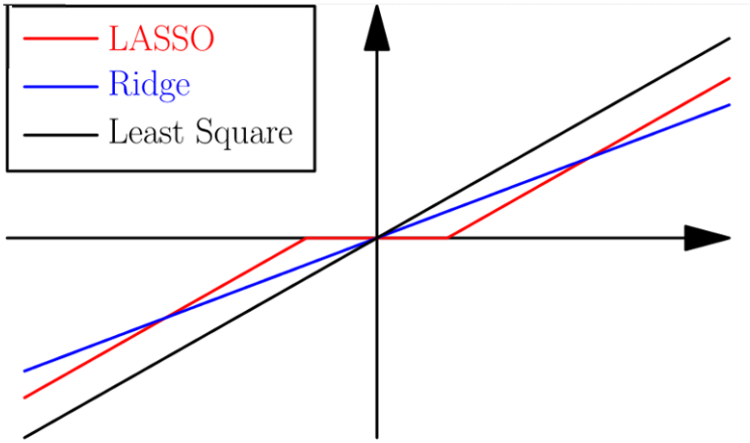

图 3 Lasso 和 Ridge、最小二乘法相比，可以产生稀疏系数

1.2 Adaptive Lasso

为解决 Lasso 估计偏差较大的问题，Zou (2006)提出了 Adaptive Lasso 的方法以进行改进，基本思想是对不同的系数赋予不同的压缩权重。对不同的变量

$$\underset{\beta\in\mathbb{R}^p}{\text{minimize}}\left\{\frac{1}{2}\|\mathbf{y}-\mathbf{X}\beta\|_2^2+\lambda\sum_{j=1}^{p}w_j|\beta_j|\right\}$$

系数β_j运用不同的调整参数λ_j，下面给出 Adaptive Lasso 的定义：

其中权重向量w_j的选取方式是，假设$\tilde{\beta}_j$是真实模型β_j的一个估计， $\tilde{\beta}$采用最小二乘法估计、Lasso 估计、Ridge 估计等估计值作为β_j的初步参照值，再选取一个参数v，则权重向量定义为$\omega_j = 1/|\tilde{\beta}_j|^v$。

当$\widetilde{\beta_j}$为 0 时，会推导出β_j为 0。而且$\widetilde{\beta_j}$越大，则 Adaptive Lasso 对第 j 个参数β_j施加的惩罚越小，从而偏差也会较小。所以 Adaptive lasso 也会得到稀疏解，并且会减小不相关变量被选中的概率。

值得指出的是，Adaptive Lasso 是一个凸优化问题，这样它就不会陷入局部最优解。

1.3 惩罚函数特征

稀疏性限制条件会使部分变量的系数收缩至零，相当于挑选了一部分重要的变量进入随后的分析，此所谓变量选择。变量选择是传统的最佳子集选择的延伸，在现代统计学习中扮演着重要角色。稀疏性罚函数的研究与变量选择问题息息相关.

针对稀疏性罚函数的设计，Fan 和 Lv（2010）提出了 3 个重要标准，不过没有一个范数恰好能同时满足这 3 个标准。

第一，无偏性：估计器是近似无偏的，尤其是当实际系数的值较大时，估计器的无偏性能够降低模型的偏差，通常使用的 L_1 惩罚函数并不满足无偏性；

第二，稀疏性：估计器自动将小的系数压缩至零，实现变量选择，对于范数 L_q，当 q>1 时不满足稀疏性标准；

第三，连续性：估计器具备连续性，可以降低模型预测的不稳定性，对于范数 L_q 来说，当 1>q≥0 时不具备连续性。

Lasso 在 L_1 惩罚下满足稀疏性和连续性，但不满足无偏性。如图 3 所示，当估计的参数值比较小的时候，Lasso 使其直接等于 0，满足了稀疏性的条件，并且图像是连续的，也满足了连续性的条件。但是对于那些估计出来比较大的参数，并不具有无偏性。其中$\hat{\beta}_{\mathrm{OLS}}$为无偏估计，而 Lasso 估计的出来$\hat{\beta}_{\mathrm{Lasso}}$一直与其相差了一个λ，使得 Lasso 不满足上述所谈到的一个好的惩罚函数所具有的无偏性。

综上所述，对于 Bridge 回归模型中的惩罚项$\sum_{j=1}^{p}|\beta_j|^q$，当 0≤q<1 时，$\sum_{j=1}^{p}|\beta_j|^q$为一个凹函数，不满足连续性；$L_1$ 惩罚不满足无偏性；当 q>1 时，不满足稀疏性。

1.4 SCAD

Fan 和 L(2001)改进了 Lasso 法，提出了 SCAD(the Smoothly Clipped Absolute Deviation Penalty)方法，其惩罚函数是对称非凹的，并被证明了满足无偏性、稀疏性和连续性的性质。同时，SCAD 比 Lasso 更为稳定，并且大大降低了计算量。

SACD 的数学形式为：

$$\hat{\beta} = \arg\min_{\beta} ||y - \sum_{j=1}^{p} X_j\beta_j||_2^2 + \sum_{j=1}^{p} P_\lambda(|\beta_j|)$$

其中$P_\lambda(|\beta_j|)$是一个连续可微的惩罚函数，其一阶导数满足：

$$P_\lambda(\beta_j;a)=\lambda\left\{I(\beta_j\le\lambda)+\frac{(a\lambda-\beta_j)_+}{(a-1)\lambda}\,I(\beta_j>\lambda)\right\}$$

其中$a>2$是一个预先给定的参数。上述惩罚函数的显示表达形式为：

$$min\left\{\frac{1}{2n}\|Y-X\beta\|_2^2+\lambda\left[\frac{1}{2}(1-\alpha)\sum_{j=1}^{p}\|\beta_j\|_2^2+\alpha\sum_{j=1}^{p}\|\beta_j\|_1\right]\right\},\alpha\in[0,1].$$

$$P_\lambda(\beta_j;a)$$

$$=\begin{cases}\lambda|\beta_j|, & |\beta_j|\le\lambda\\ -(|\beta_j|_j^2-2a\lambda|\beta_j|)/2(a-1) & \lambda<|\beta_j|\le a\lambda\\ \dfrac{(a+1)\lambda^2}{2} & |\beta_j|>a\lambda\end{cases}$$

因此 SCAD 方法通过惩罚函数，将较小的系数压缩变小，甚至等于 0；而对于较大的系数基本保持不变，从而实现了变量选择的目的。同时文献中还证明了该方法所得估计量满足无偏性、稀疏性、连续性和 Oracle 等优良性质。

在具体计算中，由于该模型含有两个参数λ和α，因此计算过程相对复杂，需要通过二维广义交叉验证的方法来实现，因此 Fan 指出，α通常取值为 3.7，λ的值可以通过交叉验证法解得。

1.5 Elastic Net

Lasso 和 Adaptive Lasso 方法尽管在很多情形下具有良好的性质，但是它们具有如下的局限性：

第一，由于凸优化问题的本质，在 p>n 的情形下，Lasso 和 Adaptive lasso 最多只能选择 n 个变量。

第二，如果有两个变量具有高度的相关性，Lasso 和 Adaptive lasso 通常只会随机选择其中的一个。诚然，这会让模型更加稀疏。但是，如果一个“假的”自变量和一个“真的”自变量具有很高的相关度，Lasso 和 Adaptive lasso 可能会选择这个“假的”自变量而舍弃“真的”自变量。

第三，在常规 n>p 的情形下，如果变量之间具有很高的共线性，则 Lasso 和 Adaptive lasso 的表现非常差。

第一条和第二条的局限性使得 Lasso 和 Adaptive lasso 在某些问题中就变得不是很适用了，因此 Zou 和 Hastie（2005）提出了新的弹性网（Elastic net）方法来解决上面的局限性。

同时使用 L_1 和 L_2 惩罚，其形式为：

Elastic Net 是 Lasso 和岭回归的结合体，同时具有 Lasso 的是回归系数变得稀疏的性质，也具有岭回归那样在高维计算中的稳健性。弹性网可以使得在优化过程中，相关性较强的变量的系数较为接近，即产生“组效应”。

它在 Lasso 与岭回归之间做了折中：加上一个调整参数α，使得其兼具二者的性质，惩罚项中的$1/2$是一个更加直观的软阀值因子。当$\alpha=1$时，Elastic Net 模型退化为 Lasso 模型；当$\alpha=0$时，其变为岭回归模型。

1.6 Group Lasso

在某些线性模型中，会存在某几个变量的系数同时为 0 或同时不为 0 的现象。Yuan 和 Lin（2006）针对上述现象提出了组 Lasso（Group Lasso）的概念：将同时为 0 或不为 0 的变量归为一组，通过在目标函数中惩罚每一组的 L2 范数，可以将一整组的系数同时消成零，即抹掉一整组的变量。用新的变量 X_J 表示，其系数为β_j，模型可以表示为：

$$\hat{\beta} = \arg\min_{\beta}\{\frac{1}{2n}||Y - X\beta||_2^2 + \lambda\sum_{j=1}^{p}\|\beta_j\|_2\}$$

通过压缩组内变量的系数平方和来实现变量的选择，并通过变量组的概念将有内在联系的数据作为一个整体进行分析，成功将组 Lasso 方法推广到机器学习和数据挖掘等领域。

容易看出，Group Lasso 是对 Lasso 的一种推广，即对特征分组后的 Lasso。显然，如果每个组的特征个数都是 1，则 Group Lasso 就退化为原始的 Lasso。为了求解 Group Lasso，可以首先假设组内特征是正交的，针对这种情形可以利用分块坐标下降法求解，对于非正交的情形，可以首先对组内特征施加正交化。

不同于 Lasso 方法将每个特征的系数项的绝对值加总，Group Lasso 所加总的是每个组系数的 L2 范数，在优化的过程中，该结构尽量选出更少的组（组间稀疏），而组内是 L2 范数，稀疏约束没那么强。

Raman（2009）针对 Group Lasso 在回归系数的方差上不具有可解释性这一缺点,将 Group Lasso 推广为贝叶斯组 Lasso，并在生物医学领域进行应用，取得了很好的效果。Friedman（2010）将 Group Lasso 和 Lasso 相结合，提出了稀疏组 Lasso（Sparse Group Lasso）模型，通过组内和组间两次的变量选择进一步剔除无关变量，克服了特征组内可能存在冗余变量的缺点，从而实现更有效的变量降维，因此在实际问题中有着广泛的应用。

此外，基于 Group Lasso 方法的相关模型都是建立在这样的假设条件下：组间变量无关而组内变量存在相关性，也就是并没有考虑重叠组的问题。为了进一步对包含重叠组变量的问题进行研究，Zhao 等（2009）将具有嵌套结构的 Group Lasso 惩罚项作为最小二乘的惩罚函数，提出了分层组 Lasso（Hierarchical Group Lasso）模型。

2. 项目实战

本文采用 Kaggle 上的共享单车使用量数据集[1]，旨在预测美国华盛顿共享单车租赁的使用量。主要利用 python 中 sklearn 库，使用线性回归、Ridge 回归、Lasso 回归、Elastic Net 以及 Adaptive Lasso 模型对数据进行拟合，进而比较模型的 RMSE 和 score 值。同时，本项目的相关代码也发布在 Kaggle 上[2]。

2.1 数据说明

[1] 数据集网址为：https://www.kaggle.com/c/bike-sharing-demand/data

[2] 本项目网址为：https://www.kaggle.com/myzhai/bike-sharing-lasso-ridge-elasticnet-adaptivelasso

2.1.1 变量说明

数据为两年间共享单车每小时租金数据。数据集分为训练集和测试集，在训练集中包含 10886 个样本以及 12 个字段；训练集由每月的前 19 天组成，而测试集是从第 20 天到月底，以预测测试集涵盖的每小时内租赁的共享单车总量。各变量名称及含义在表 1 中列示。其中，datatime 是字符串变量，season、holiday、workingday 和 weather 是分类变量，temp、atemp、humidity、windspeed 是连续型变量，casual、registered 和 count 是离散型变量。

表 1 变量描述

变量名称	变量描述
datatime	共享单车租借时间，包括年、月、日，及小时数
season	季节，season=1 表示春天，season=2 表示夏天，season=3 表示秋天，season=2 表示冬天
holiday	是否是节假日，holiday=0 表示非节假日，holiday=1 表示是节假日
workingday	是否是工作日，workingday =0 表示非工作日，workingday =1 表示是工作日
weather	天气情况，weather=1 表示晴天，多云；weather=2 表示雾天，阴天；weather=3 表示小雪，小雨；weather=4 表示大雨，大雪，大雾
temp	温度
atemp	体表温度
humidity	相对湿度
windspeed	风速
casual	非注册用户数
registered	注册用户数
count	总用户数

2.1.2 变量描述性统计

接下来进行变量的描述性统计分析。首先分别考察其中位数、均值、标准差和分位数等信息。如表 2 所示，temp、atemp、humidity、windspeed 均近似对称分布, 而 casual、registered、count 均偏态分布。

针对数值型变量，计算其偏态与峰态系数，如表 3 所示。结果表明，变量 temp、atemp、humidity 呈低度偏态，变量 windspeed 呈中度偏态，变量 casual、registered、count 高度偏态；变量 temp、atemp、humidity 为平峰分布，变量 windspeed、casual、registered、count 为尖峰分布。

表 2 变量描述性描述

	count	mean	std	min	25%	50%	75%	max
season	10886	2.506614	1.116174	1	2	3	4	4
holiday	10886	0.028569	0.166599	0	0	0	0	1
workingday	10886	0.680875	0.466159	0	0	1	1	1
weather	10886	1.418427	0.633839	1	1	1	2	4
temp	10886	20.23086	7.79159	0.82	13.94	20.5	26.24	41
atemp	10886	23.65508	8.474601	0.76	16.665	24.24	31.06	45.455
humidity	10886	61.88646	19.24503	0	47	62	77	100
windspeed	10886	12.7994	8.164537	0	7.0015	12.998	16.9979	56.9969
casual	10886	36.02196	49.96048	0	4	17	49	367
registered	10886	155.5522	151.039	0	36	118	222	886
count	10886	191.5741	181.1445	1	42	145	284	977

表 3 数值型变量的偏态、峰态系数

	偏态系数	峰态系数
temp	0.003691	-0.91453
atemp	-0.10256	-0.85008
humidity	-0.08634	-0.75982
windspeed	0.588767	0.630133
casual	2.495748	7.551629
registered	1.524805	2.626081
count	1.242066	1.300093

2.1.3 数据预处理

首先进行缺失值、重复值处理，因数据集质量较高，故不存在缺失值与重复值。

进而对个别变量进行加工处理。Datetime 变量的形式为“2011-01-01 00:00:00”，包含信息较多，故提取出月份 month、周数 week、小时数 hour 三个变量，同时删除 datetime 变量。其中，周数变量为当日是一周的第几天。

接下类处理季节 season 变量。绘制季节与温度的箱线图时发现，季节 3 即秋季的温度明显高于夏季，与实际情况不符。季节的划分通常和纬度相关，而该数据集是用来预测美国华盛顿的租赁数量，且美国和我国的纬度基本一样，故按照 3、4、5 月为春季，6、7、8 月为夏季，9、10、11 月为秋季，12、1、2 月为冬季这个规则来重新划分。因此产生一个新的变量 group season 来取代 season 变量。季节与温度的箱线图如图 4 所示[3]。

[3] 由于 Kaggle 网站不支持英文字体绘图，故本文图表中的标题等均采用英文。

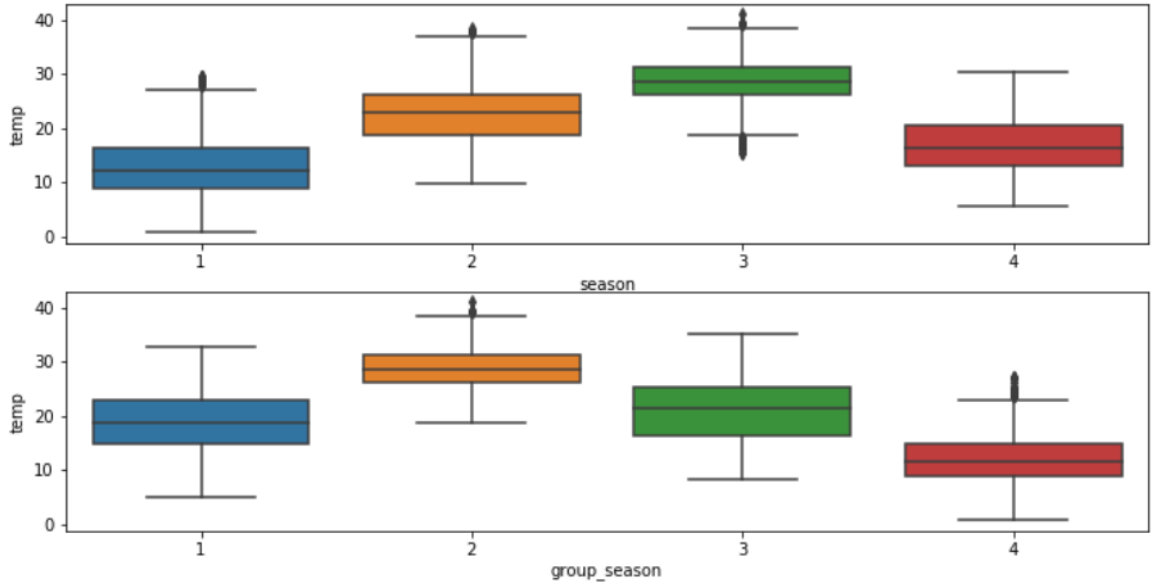

图 4 季节与温度箱线图

2.2 探索性数据分析

探索性数据分析（Exploratory Data Analysis，简称 EDA），是一种分析数据集以概括其主要特征的方法，是对已有的数据在尽量少的先验假定下进行探索，通过作图、制表、方程拟合、计算特征量等手段探索数据的结构和规律的一种数据分析方法。

2.2.1 日期和总租赁数量

本文首先探索日期与总租赁数量的关系，如图 5 所示。由图中折线可以明显看出，2012 年相比 2011 年租赁数量有所增长，且波动幅度相类似。

图中也可看出，4-10 月是共享单车租赁数量的高峰期，其余月份租赁共享单车的人数较少。即共享单车的租赁数量与温度有关。

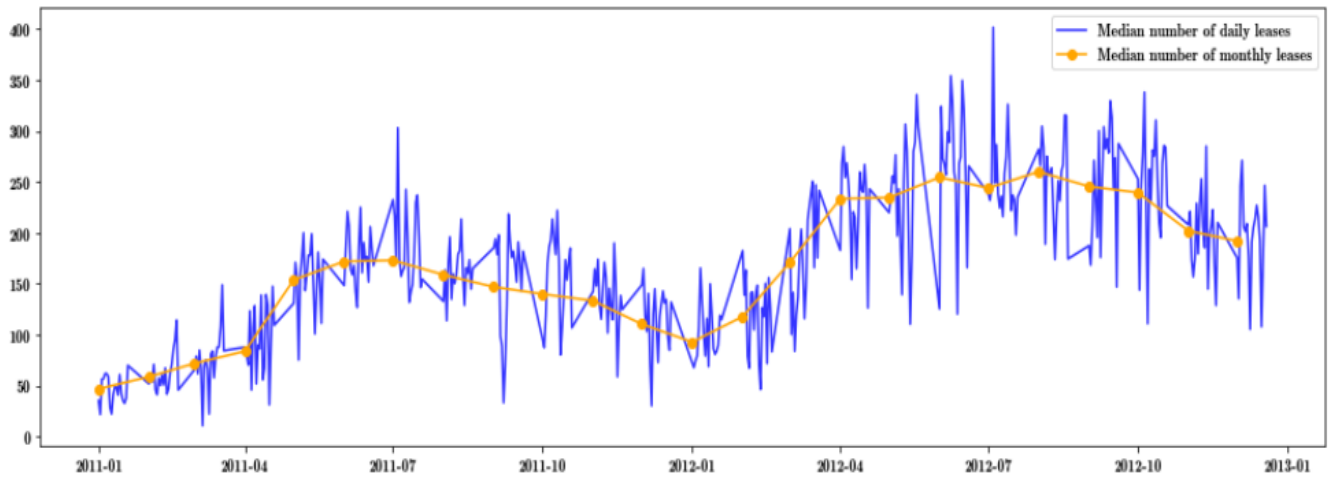

图 5 日期与总租赁数量关系图

2.2.2 月份和总租赁数量

为进一步观察月份与总租赁数量的关系，绘制了月份与租赁数量关系箱形图，如图 6 所示。可以看出，6 月、7 月明显为租赁数量的高峰，且 5 月、8 至 10 月的租赁数量较高，表明夏季人们更乐于与租赁共享单车。但 1-3 月的租赁数量为全年最低，表明天气寒冷时，租赁数量会急剧下降。图 6 得出的结论基本与图 5 基本一致,另外每个月均有不同程度的离群值.

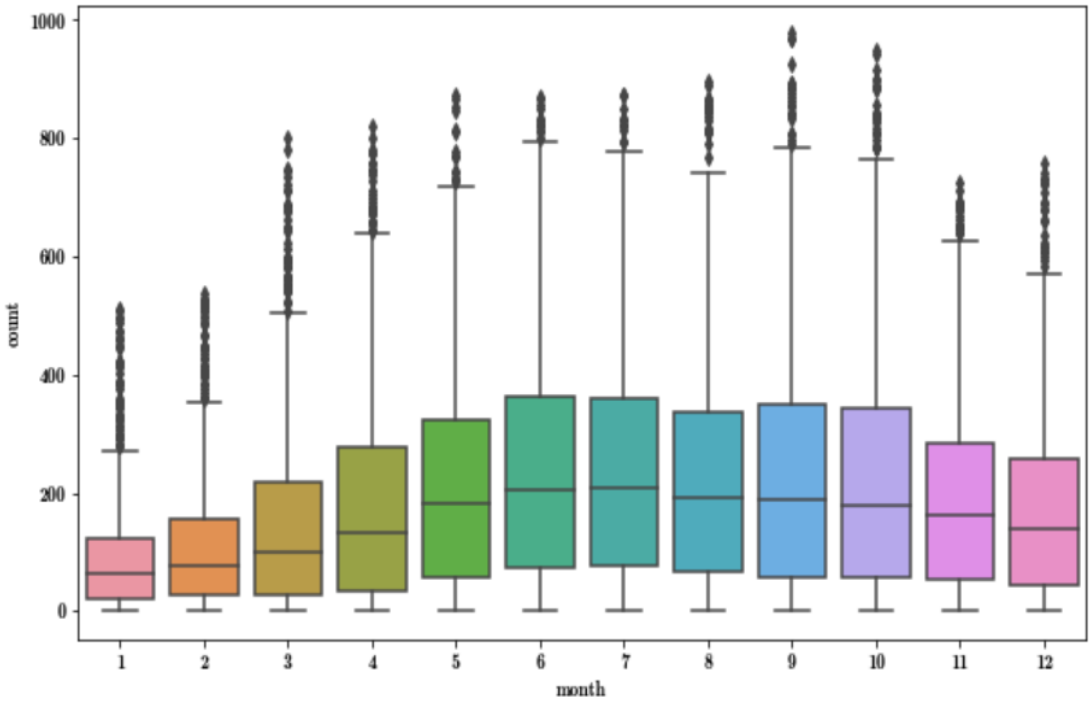

图 6 月份与总租赁数量关系图

2.2.3 星期数和总租赁数量

接下来考察星期数与总租赁数量之间的关系，将用户分为非注册用户 casual、注册用户 registered 与总体用户 count 三类，分别研究其在一周里的第几天租赁共享单车数量最多。就中位数来说，非注册用户在星期六和星期日使用次数最多，而注册用户则在星期一至星期五使用次数最多。从全体用户角度看，工作日的使用数量多于非工作日的使用数量，表明在工作日使用共享单车的注册用户是主要用户群体。

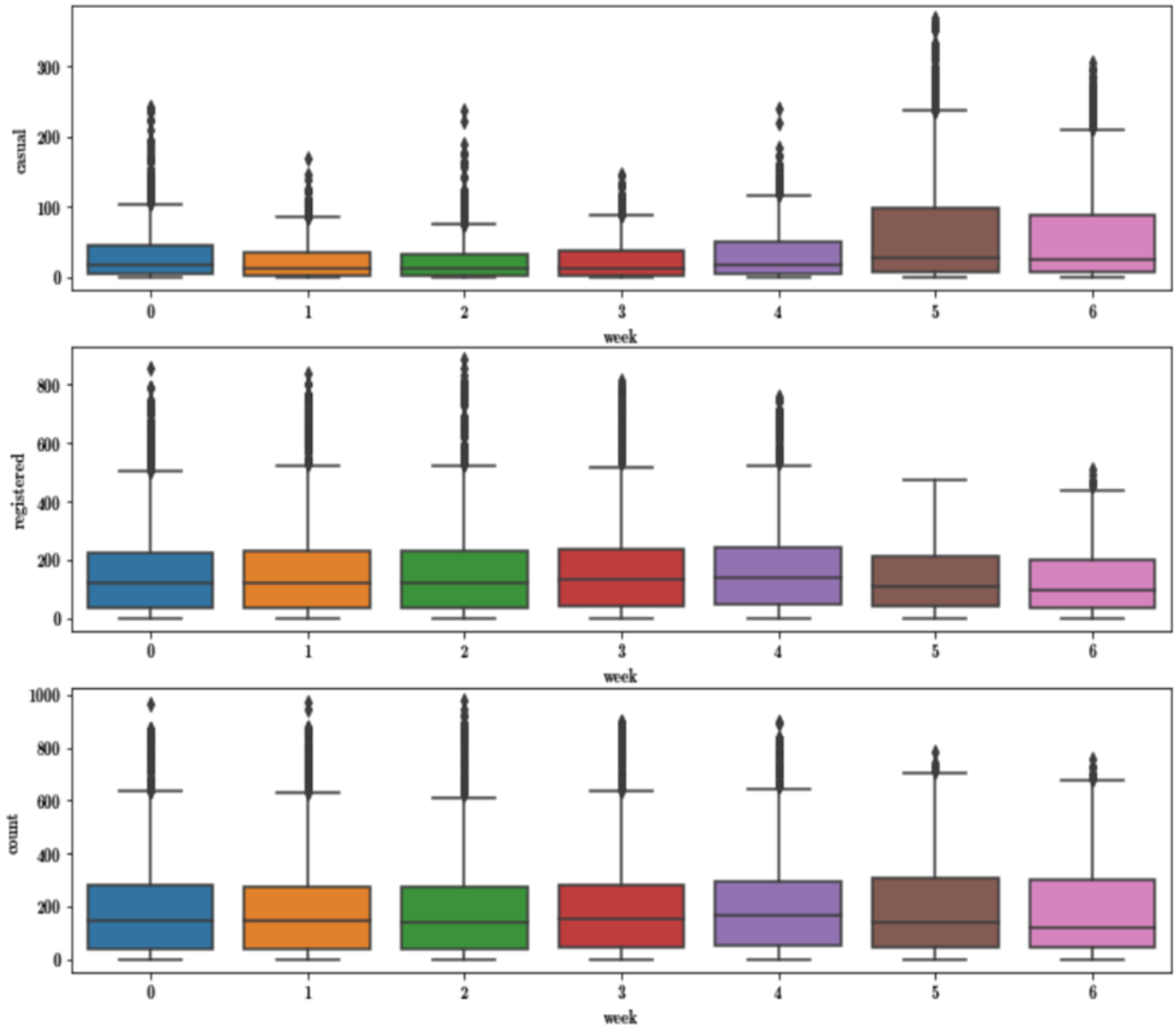

图 7 星期数与总租赁数量关系图

2.2.4 节假日、工作日和总租赁数量

进而考察节假日和工作日与总租赁数量的关系，绘制箱型图如图 8 所示。可以看出未注册用户在节假日使用共享单车较多，在工作日使用共享单车较少。而注册用户相反，在节假日使用少，在工作日使用较多。

从总体上看，节假日租赁数量少于工作日租赁数量，可能原因为多数为注册用户租赁共享单车是用来在节假日出游，而多数注册用户在工作日租赁共享单车是用来上班或上学。

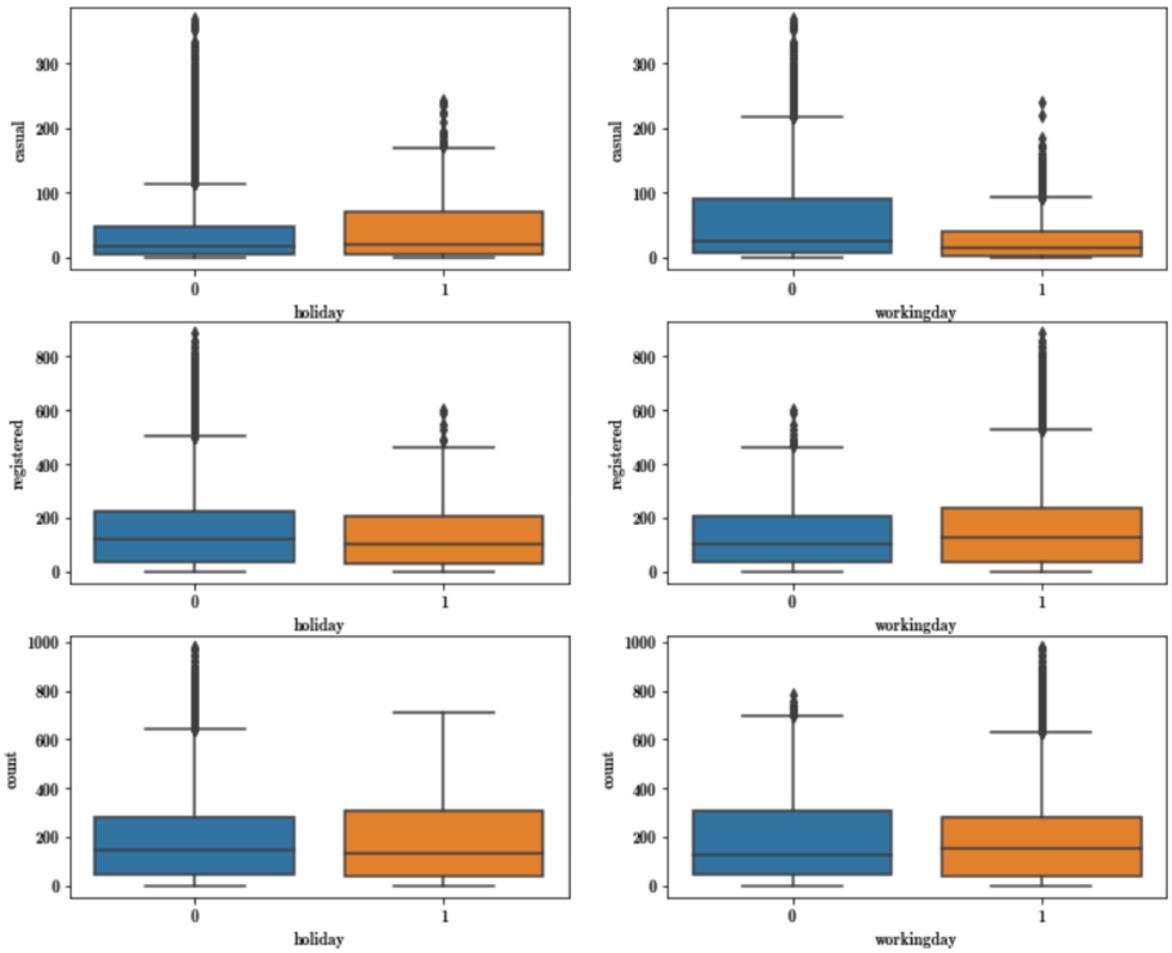

图 8 节假日、工作日与总租赁数量关系图

2.2.5 小时和总租赁数量

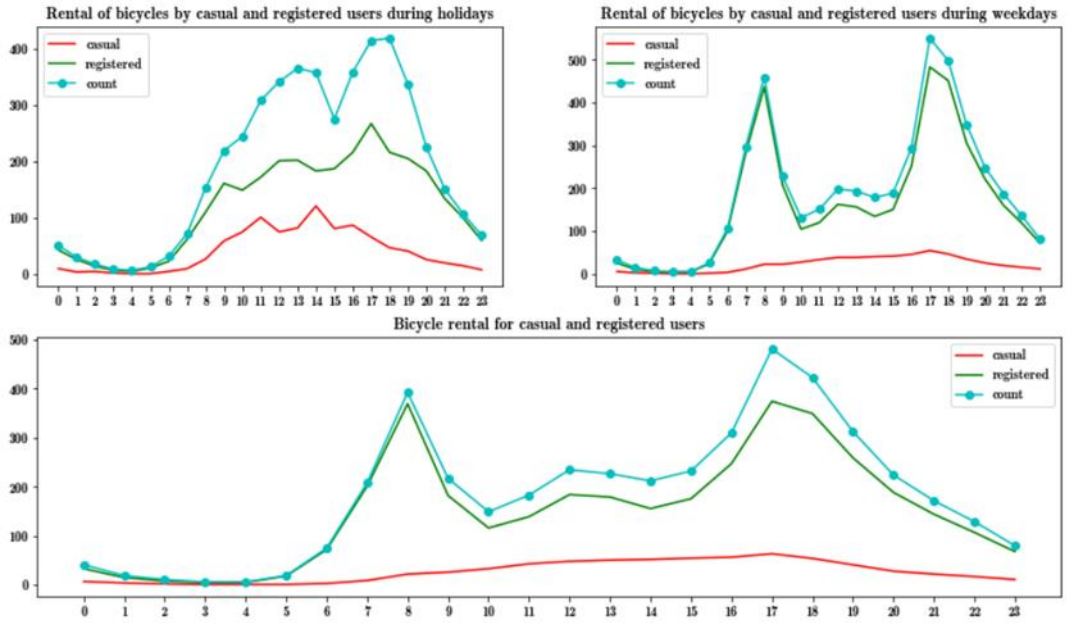

图 9 小时与总租赁数量关系图

左上角的图可以看出，在节假日，为注册用户与注册用户的趋势相接近，但是未注册用户的使用高峰在 14:00 ，注册用户的使用高峰，在 17:00。这与上文的推测相一致，即未注册用户多使用共享单车出门旅行，而注册用户多为上下班而使用共享单车。

右上角的图可以看出，在工作日，注册用户呈现双峰走势，在早 8:00 和晚上 17:00 均为用车高峰期，这正是上下班或者上下学的高峰时间段。相比之下，未注册用户呈现的趋势较为平稳。

对于注册用户来说， 17:00 在节假日和工作日均为用车高峰期，表明部分用户在节假日未必休假。

2.2.6 天气和总租赁数量

接下来考察天气与总租赁数量之间的关系。就中位数而言，未注册用户和注册用户均表现为，在工作日和非工作日的租赁数量，随着天气的恶劣程度增加而减少。在天气温活晴朗时租赁数量较高，特别地，当天气为大雨或大雪天且不是工作日时，均没有共享单车租赁。

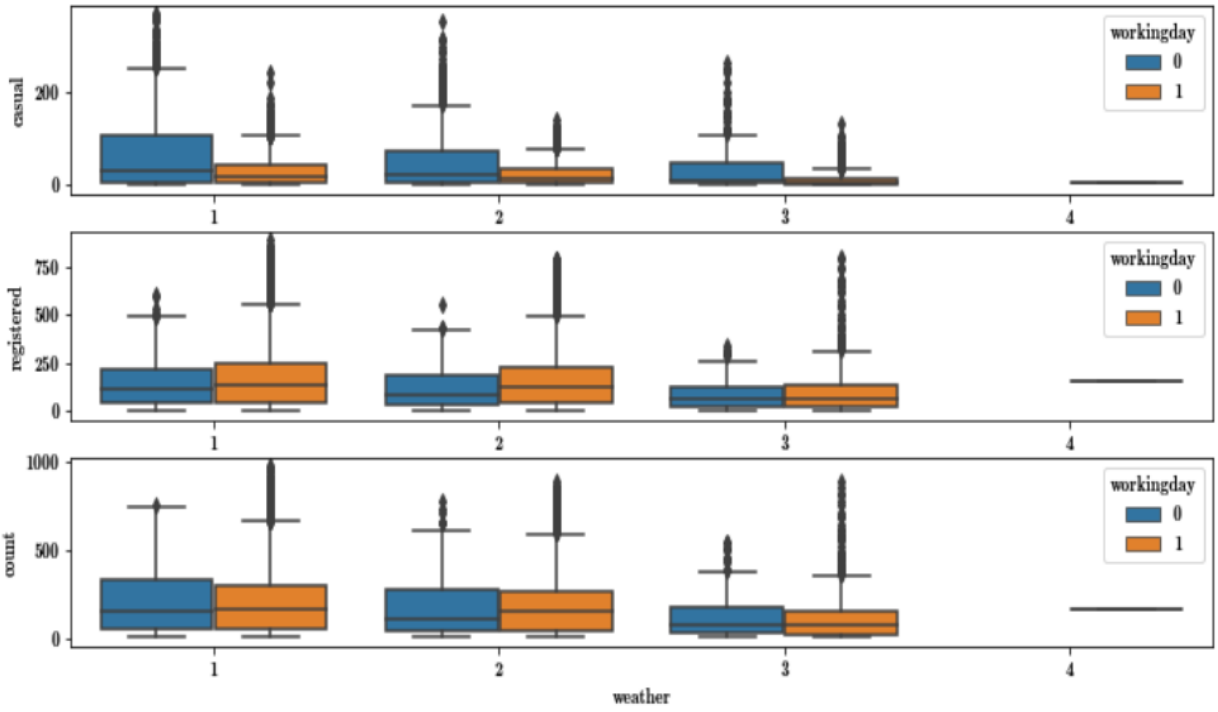

图 10 天气与总租赁数量关系图

2.2.7 其他变量和总租赁数量

接下来考察剩余变量与租赁数量之间的关系。利用 seaborn 的 pairplot 绘制工具，将温度、体感温度、相对湿度和风速四个连续变量与未注册用户、注册用户和全部用户之间的关系绘制图像。如图 11 所示。

从图上可以看出，温度和体感温度分别与未注册用户、注册用户、全部用户均有一定程度的正相关，而相对湿度和风速与之呈现一定程度的负相关。另外，其他变量之间也有不同程度的相关关系。

另外，第四列(风速)在散点图中间有明显的间隙。在考察风速的原始变量后，发现大量风速值为零，与实际不符，因此将它当成缺失值处理，采用向后填充的方式填补数据。

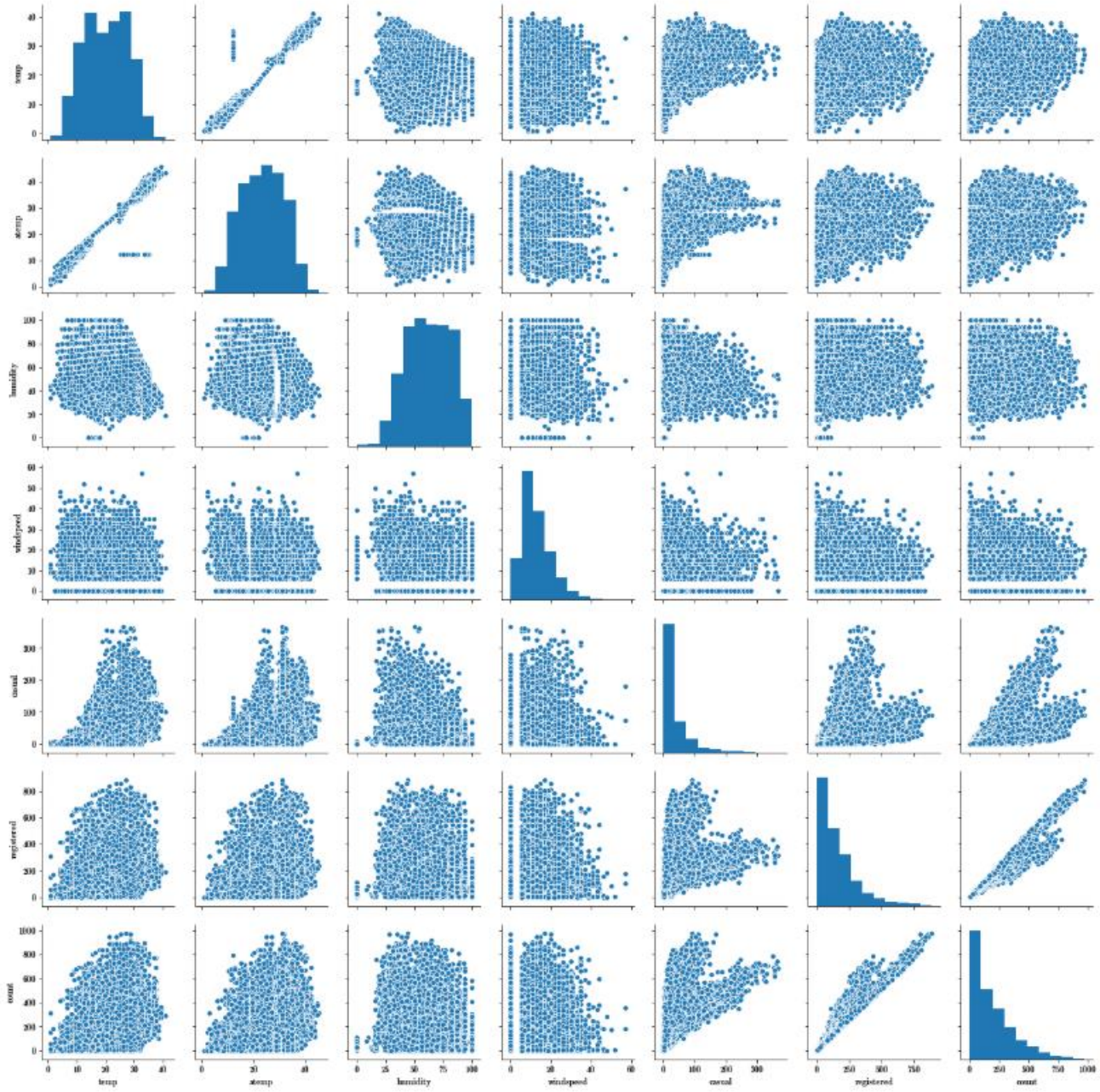

图 11 其他变量与总租赁数量关系图

2.2.8 变量相关系数图

最后绘制所有样本之间相关系数热力图。由于多个样本不满足正态分布，因此对其进行对数变化。但对数变换之后，注册用户和所有用户的租赁数量相差很大，因此计算相关系数是选择 Spearman 相关系数。如图 12 所示。

图中颜色越浅代表相关系数越接近 1，颜色越深代表相关系数越接近-1。从图中可以明显看出，未注册用户、注册用户与全体用户之间存在着明显的共线性。因此，考虑在模型构建时将未注册用户与注册用户变量删除。

另外温度与体感温度之间的相关系数为 0.99 ，如果不进行特征，选择将产生严重的多重共线性。此处的变量特征选择留给 Lasso 模型进行处理，以便直观看到 Lasso 稀疏建模的效果。

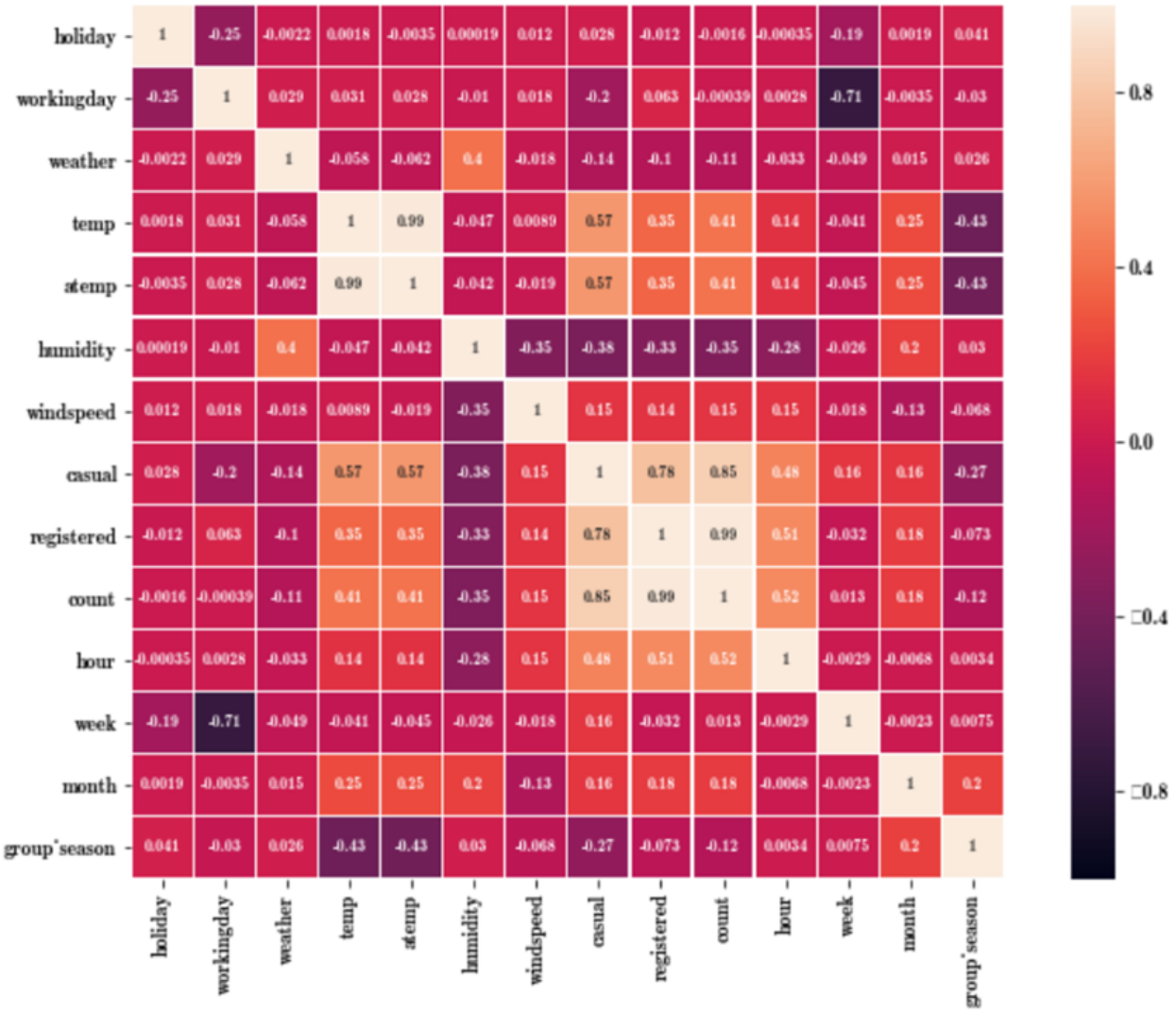

图 12 各变量间相关系数图

2.3 模型构建

本文使用了线性回归、Ridge 回归、Lasso 回归、Elastic Net 以及 Adaptive Lasso 五种模型拟合样本，从而进行比较。

2.3.1 线性回归

在开始回归前删除 casual、registered 变量，以避免多重共线性。划分数据集，将训练集与测试集之间比例设置为 4：1。

调用 python 中 sklearn 库实现线性回归。模型训练后，分别预测训练集和测试集，并计算均方根误差 RMSE 和拟合优度 score。得出训练集 RMSE 为 1.0347173340121176，评分为 0.46700577529675036；测试集 RMSE 为 1.0510323073614725，评分为 0.45778089839236125。

2.3.2 Ridge 回归

接下来构建 Ridge 模型。首先设置正则化项参数 alpha，初始值设置为 1。因为正则化项参数对结果的影响较大，接下来就通过岭迹图来选择正则化参数。绘制岭迹图如图 13 所示。

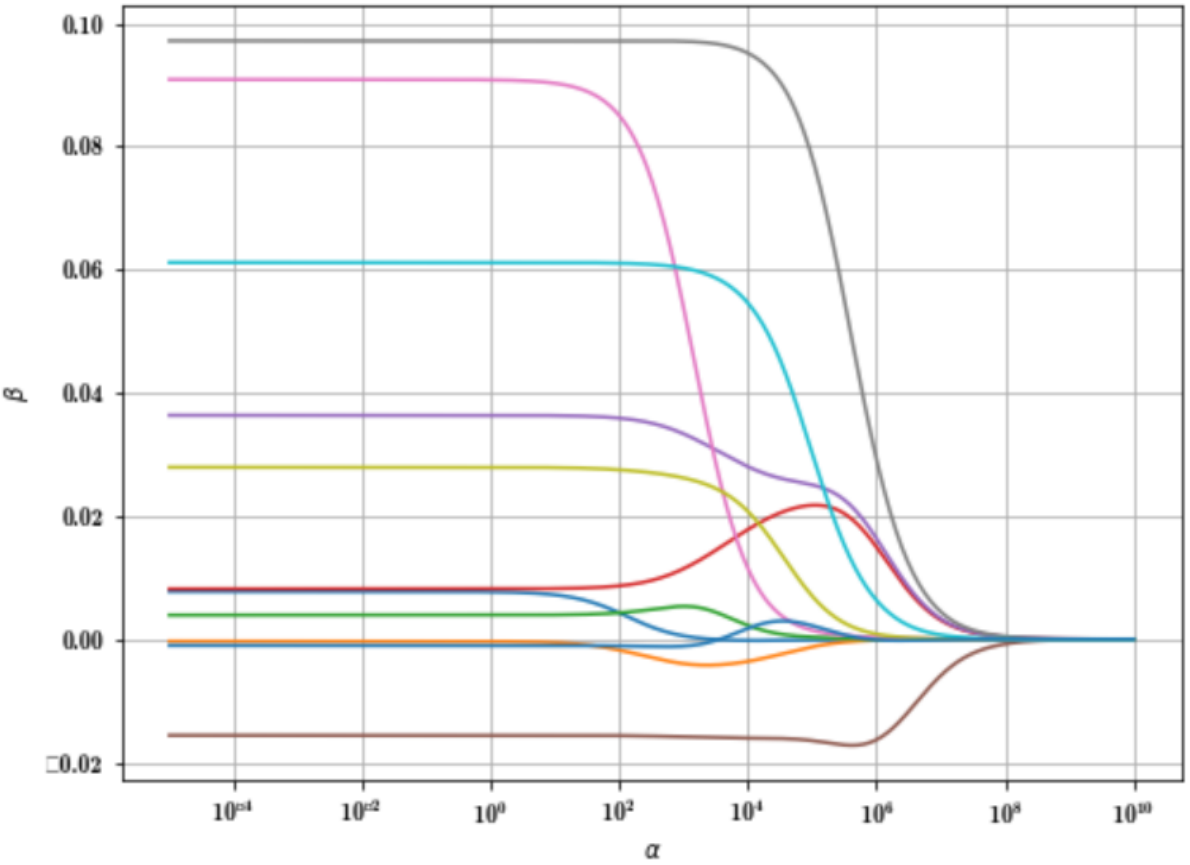

图 13 正则化项参数α和回归系数β岭迹图

通过图像可以看出，当 alpha 为 10^7 时所有变量岭迹趋于稳定。按照岭迹法应当取 alpha=10^7。

由于是通过肉眼观察的，其不一定是最佳。因而采用另外一种方式: 交叉验证的岭回归。调用 sklearn 库中的 RidgeCV 函数，设置交叉验证折数为 10 折，求得最佳正则化项参数为 805.0291812295973。然后用这个参数进行模型训练。模型拟合后给出的变量系数均不为 0，系数绝对值最大为 0.09708992，系数绝对值最小为 0.00074612，表明该变量对租赁数量的解释力相当微弱。

进而预测训练数据和测试数据，分别计算其均方根误差和拟合优度，求得训练集 RMSE 为 1.0348076524200298, 评分为 0.46691272323469246；测试集 RMSE 为 1.0508046977499312, 评分为 0.45801571689420717。

2.3.3 Lasso 回归

接下来构建 Lasso 模型。首先绘制正则化项参数α和回归系数β的 Lasso 图，如图 14 所示。通过图像可以看出，当 alpha 为 10 时所有变量岭迹趋于稳定，应当取 alpha=10。

调用 sklearn 库中的 LassoCV 函数，进行交叉验证的 Lasso 回归。设置交叉验证折数为 10 折，求得最佳正则化项参数为 0.005074705239490466。然后用这个参数进行模型训练。模型拟合后给出的变量系数有 4 个为 0，即 Lasso 对变量进行了稀疏建模，有效地使绝对值较小的系数直接缩减为 0。其中系数绝对值最大为 0.09721864，系数绝对值最小为 0.01001827。回顾 Ridge 回归中绝对值最小的系数为 0.00074612，远小于 Lasso 绝对值最小的系数 0.01001827，表明 Lasso 具有特征选择的作用。

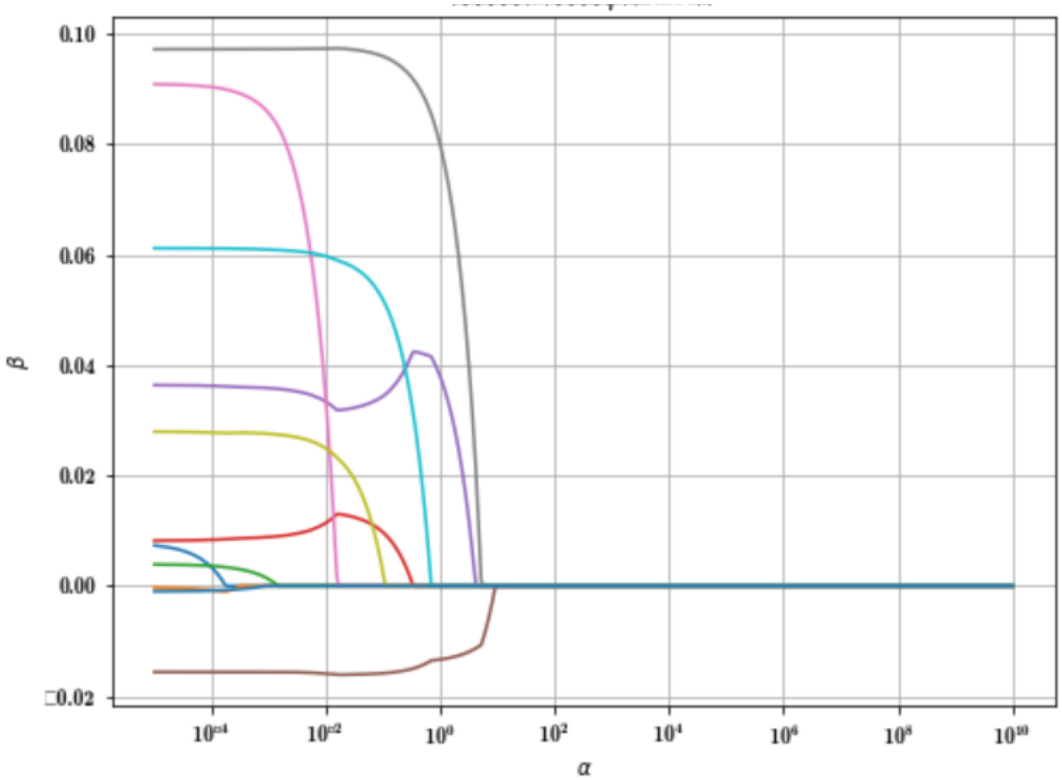

图 14 正则化项参数α和回归系数β的 Lasso 图

进而预测训练数据和测试数据，分别计算其均方根误差和拟合优度，求得训练集 RMSE 为 1.0347988070045209，评分为 0.4669218367318746；测试集 RMSE 为 1.050818996520012，评分为 0.45800096674816204。

2.3.4 Elastic Net

接下来构建 Elastic Net 模型。调用 sklearn 库中的 ElasticNetCV 函数，进行交叉验证的 Elastic Net 回归。设置交叉验证折数为 10 折，求得最佳正则化项参数为 0.009461323755890769。

用这个参数进行模型训练。模型报告的回归系数是同 Lasso 一样有 4 个为 0，其中系数绝对值最大为 0.09721359，系数绝对值最小为 0.0100756。系数绝对值的最值与 Lasso 回归相差不大，都起到了稀疏建模的作用。

进而预测训练数据和测试数据，分别计算其均方根误差和拟合优度，求得训练集 RMSE 为 1.0347973056394495，评分为 0.46692338359170604；测试集 RMSE 为 1.0508238633371485，评分为 0.45799594625214857。

2.3.5 Adaptive Lasso

最后构建 Adaptive Lasso 模型。由于 python 库中没有相应的函数，因此本文在 GitHub 上参考了一段代码[4]，并进行相应修改。本文的修改和扩展主要有三：一是源代码中没有估计截距项，本文修改成带有截距项的回归模型。二是源代码中不包含预测值，本文计算了训练集、测试集的模型预测值，便于评价模型。三是源代码中不包含模型评价准则代码，本文根据 sklearn 中 score 函数的源代码，计算出 Adaptive Lasso 模型的评价指标。

[4] https://github.com/AaronNHorvitz/AdaptiveLASSO/blob/master/adaptive_LASSO.py

Adaptive Lasso 模型的初始参考选为 Lasso 模型。起初设定 alpha 值为 1，得到的 RMSE 数值较高，score 数值较低，表明模型拟合效果不好。经过不断尝试，最终给定 alpha 值为 0.005074705239490466，即为上文中 LassoCV 函数经过十次交叉验证得出的最佳 alpha 值。模型参数中设置最大迭代次数为 1000，Lasso 的最大迭代次数为 10。

更改 alpha 参数后，用这个参数再次进行模型训练。模型报告的回归系数是有 6 个为 0，表明 Adaptive Lasso 模型将 6 个变量稀疏化，超过了 Lasso 的稀疏强度。其中系数绝对值最大为 0.15613827，为所有模型中系数最大的；系数绝对值最小为-0.06362913，同样超过了其他模型的最小系数绝对值。

进而预测训练数据和测试数据，分别计算其均方根误差和拟合优度，求得训练集 RMSE 为 1.6488433326718732, 评分为 0.42138157773200846；测试集 RMSE 为 1.7017039247458843, 评分为 0.35343635777520443。

2.4 模型评价

本文采取均方根误差 RMSE 评价模型拟合效果，RMSE 值越小，拟合效果越好。本文采取 sklearn 库中 score 函数评价模型拟合效果，score 值越高，拟合效果越好。其中 score 即 R^2，即 1 减模型残差平方和与离差平方和的比值。各模型的评价指标结果如表 4 所示。

表 4 各模型的评价指标

模型	数据集	RMSE	Score
线性	训练集	1.0347173340121176	0.46700577529675036
	测试集	1.0510323073614725	0.45778089839236125
Ridge	训练集	1.0348076524200298	0.46691272323469246
	测试集	1.0508046977499312	0.45801571689420717
Lasso	训练集	1.0347988070045209	0.4669218367318746
	测试集	1.050818996520012	0.45800096674816204
Elastic Net	训练集	1.0347973056394495	0.46692338359170604
	测试集	1.0508238633371485	0.45799594625214857
Adaptive Lasso	训练集	1.6488433326718732	0.42138157773200846
	测试集	1.7017039247458843	0.35343635777520443

整体上看，训练集的 RMSE 均低于测试集，训练集的 score 均高于测试集，表明数据在训练集上的拟合效果优于在验证集上的拟合效果。

从 RMSE 指标和 score 指标上看，线性回归、Ridge、Lasso 与 Elastic Net 模型表现差别不大，Ridge 模型略优于其他模型。Adaptive Lasso 模型的效果最差，训练集和测试集的 RMSE 值均高于其他。

Lasso 和 Elastic Net 模型拟合效果差于 Ridge 模型，可能原因是二者都进行了稀疏建模，但本数据集不一定适用于稀疏模型，因而拟合结果不够理想。

Adaptive Lasso 拟合效果不好的可能原因有二：一是 Adaptive Lasso 没有标准的 python 实现代码，本文借鉴的代码可能存在一定问题，因此模型拟合效果不够理想。二是可能使该共享单车数据集不适用 Adaptive Lasso 方法，导致拟合效果不理想。

综上所述，Ridge 模型在本数据集上达到了较优效果，但还需要学习其他模型，比如决策树、随机森林、神经网络等。

附录

本文主要代码如下：

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge,RidgeCV,Lasso,LassoCV,ElasticNet,ElasticNetCV
from sklearn import metrics
from math import sqrt
import pandas as pd
df = pd.read_csv('/kaggle/input/bike-sharing-demand/train.csv')
pd.set_option('display.max_rows',20)

#偏态、峰态系数
for i in range(5,12):
    name = df.columns[i]
    print('{0}偏态系数为{1}，峰态系数为{2}'.format(name,df[name].skew(),df[name].kurt()))

#去重
print('未去重：',df.shape)
print('去重：',df.drop_duplicates().shape)

#绘图
fig,axes = plt.subplots(nrows=2,ncols=2,figsize=(12,6))
sns.boxplot(x='windspeed',data=df,ax=axes[0][0])
sns.boxplot(x='casual',data=df,ax=axes[0][1])
sns.boxplot(x='registered',data=df,ax=axes[1][0])
sns.boxplot(x='count',data=df,ax=axes[1][1])

df['datetime'] = pd.to_datetime(df['datetime'])
df['hour'] = df.datetime.dt.hour
df['week'] = df.datetime.dt.dayofweek
df['month'] = df.datetime.dt.month
df['year_month'] = df.datetime.dt.strftime('%Y-%m')
df['date'] = df.datetime.dt.date
```

```
df.drop('datetime',axis=1,inplace=True)

import numpy as np
df['group_season'] = np.where((df.month <=5) & (df.month >=3), 1, np.where((df.month <=8) & (df.month >=6),
2, np.where((df.month <=11) & (df.month >=9), 3, 4)))
fig, ax = plt.subplots(2, 1, figsize=(12, 6))
#绘制气温和季节箱线图
sns.boxplot(x='season', y='temp',data=df, ax=ax[0])
sns.boxplot(x='group_season', y='temp',data=df, ax=ax[1])

df.drop('season', axis=1, inplace=True)
df.shape

fig, ax = plt.subplots(3, 1, figsize=(12, 6))
sns.boxplot(x='weather', y='casual', hue='workingday',data=df, ax=ax[0])
sns.boxplot(x='weather', y='registered',hue='workingday', data=df, ax=ax[1])
sns.boxplot(x='weather', y='count',hue='workingday', data=df, ax=ax[2])

sns.pairplot(df[['temp', 'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count']])

df.loc[df.windspeed == 0, 'windspeed'] = np.nan
df.fillna(method='bfill', inplace=True)
df.windspeed.isnull().sum()

#对数转换
df['windspeed'] = np.log(df['windspeed'].apply(lambda x: x+1))
df['casual'] = np.log(df['casual'].apply(lambda x: x+1))
df['registered'] = np.log(df['registered'].apply(lambda x: x+1))
df['count'] = np.log(df['count'].apply(lambda x: x+1))
sns.pairplot(df[['windspeed', 'casual', 'registered', 'count']])

correlation = df.corr(method='spearman')
plt.figure(figsize=(12, 8))
#绘制热力图
sns.heatmap(correlation, linewidths=0.2, vmax=1, vmin=-1, linecolor='w',
annot=True,annot_kws={'size':8},square=True)

## Ridge 回归
df.drop(['casual','registered'],axis=1,inplace=True)
X = df.drop(['count','year_month','date'],axis=1)
y = df['count']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
rd = Ridge(alpha=1)
rd.fit(X_train, y_train)
```

```
print(rd.coef_)
print(rd.intercept_)
#设置参数以及训练模型
alphas = 10**np.linspace(-5, 10, 500)
betas = []
for alpha in alphas:
    rd = Ridge(alpha = alpha)
    rd.fit(X_train, y_train)
    betas.append(rd.coef_)
#绘制岭迹图
plt.figure(figsize=(8,6))
plt.plot(alphas, betas)
#对数据进行对数转换，便于观察.
plt.xscale('log')
#添加网格线
plt.grid(True)
#坐标轴适应数据量
plt.axis('tight')
plt.title(r'正则化项参数$\alpha$和回归系数$\beta$岭迹图')
plt.xlabel(r'$\alpha$')
plt.ylabel(r'$\beta$')
plt.show()

rd_cv = RidgeCV(alphas=alphas, cv=10, scoring='r2')
rd_cv.fit(X_train, y_train)
rd_cv.alpha_
rd = Ridge(alpha=805.0291812295973) #, fit_intercept=False
rd.fit(X_train, y_train)
print(rd.coef_)
print(rd.intercept_)
#分别预测训练数据和测试数据
ridge_y_train_pred = rd.predict(X_train)
ridge_y_test_pred = rd.predict(X_test)
#分别计算其均方根误差和拟合优度
ridge_y_train_rmse = sqrt(metrics.mean_squared_error(y_train, ridge_y_train_pred))
ridge_y_train_score = rd.score(X_train, y_train)
ridge_y_test_rmse = sqrt(metrics.mean_squared_error(y_test, ridge_y_test_pred))
ridge_y_test_score = rd.score(X_test, y_test)
print('训练集 RMSE: {0}，评分: {1}'.format(ridge_y_train_rmse, ridge_y_train_score))
print('测试集 RMSE: {0}，评分: {1}'.format(ridge_y_test_rmse, ridge_y_test_score))

## Lasso
alphas = 10**np.linspace(-5, 10, 500)
betas = []
```

```
for alpha in alphas:
    Las = Lasso(alpha = alpha)
    Las.fit(X_train, y_train)
    betas.append(Las.coef_)
plt.figure(figsize=(8,6))
plt.plot(alphas, betas)
plt.xscale('log')
plt.grid(True)
plt.axis('tight')
plt.title(r'正则化项参数$\alpha$和回归系数$\beta$的 Lasso 图')
plt.xlabel(r'$\alpha$')
plt.ylabel(r'$\beta$')
plt.show()
Las_cv = LassoCV(alphas=alphas, cv=10)
Las_cv.fit(X_train, y_train)
Las_cv.alpha_
Las = Lasso(alpha=0.005074705239490466)
Las.fit(X_train, y_train)
print(Las.coef_)
print(Las.intercept_)
#用 Lasso 分别预测训练集和测试集，并计算均方根误差和拟合优度
lasso_y_train_pred = Las.predict(X_train)
lasso_y_test_pred = Las.predict(X_test)
lasso_y_train_rmse = sqrt(metrics.mean_squared_error(y_train, lasso_y_train_pred))
lasso_y_train_score = Las.score(X_train, y_train)
lasso_y_test_rmse = sqrt(metrics.mean_squared_error(y_test, lasso_y_test_pred))
lasso_y_test_score = Las.score(X_test, y_test)
print('训练集 RMSE: {0}，评分: {1}'.format(lasso_y_train_rmse, lasso_y_train_score))
print('测试集 RMSE: {0}，评分: {1}'.format(lasso_y_test_rmse, lasso_y_test_score))

## Adaptive_LASSO
def Adaptive_LASSO(X_train,y_train,max_iterations = 1000,lasso_iterations = 10, alpha =
0.005074705239490466, tol = 0.001, max_error_up = 5, title = ''):

    # set checks
    higher    = float('inf')
    lower     = 0

    # set lists
    coefficients_list = []
    iterations_list     = []

    # set variables
    X_train    = X_train
```

```
    y_train    = y_train

    # set constants
    alpha      = alpha
    tol        = tol
    max_iter = max_iterations
    n_lasso_iterations = lasso_iterations

    g = lambda w: np.sqrt(np.abs(w))
    gprime = lambda w: 1. / (2. * np.sqrt(np.abs(w)) + np.finfo(float).eps)

    n_samples, n_features = X_train.shape
    p_obj = lambda w: 1. / (2 * n_samples) * np.sum((y_train - np.dot(X_train, w)) ** 2) \
                              + alpha * np.sum(g(w))

    weights = np.ones(n_features)
    X_w = X_train / weights[np.newaxis, :]
    X_w    = np.nan_to_num(X_w)
    X_w    = np.round(X_w,decimals = 3)

    y_train       = np.nan_to_num(y_train)

    adaptive_lasso = Lasso(alpha=alpha, fit_intercept=True)

    adaptive_lasso.fit(X_w, y_train)

    for k in range(n_lasso_iterations):
        X_w = X_train / weights[np.newaxis, :]
        adaptive_lasso = Lasso(alpha=alpha, fit_intercept=True)
        adaptive_lasso.fit(X_w, y_train)
        coef_ = adaptive_lasso.coef_ / weights
        weights = gprime(coef_)

        print ('Iteration #',k+1,':      ',p_obj(coef_))    # should go down

        iterations_list.append(k)
        coefficients_list.append(p_obj(coef_))

    print (np.mean((adaptive_lasso.coef_ != 0.0) == (coef_ != 0.0)))

    coef = pd.Series(adaptive_lasso.coef_, index = X_train.columns)

print('=========================================================================='
)
```

```
        print("Adaptive LASSO picked " + str(sum(coef != 0)) + " variables and eliminated the other " +
str(sum(coef == 0)) + " variables.")

print('==========================================================================='
)
        print("系数：",adaptive_lasso.coef_)
        print("截距：",adaptive_lasso.intercept_)
        plt.rcParams["figure.figsize"] = (18,8)

        # subplot of the predicted vs. actual

        plt.plot(iterations_list,coefficients_list,color = 'orange')
        plt.scatter(iterations_list,coefficients_list,color = 'green')
        plt.title('Iterations vs. p_obj(coef_)')
        plt.show()

        # plot of the coefficients'

        imp_coef = pd.concat([coef.sort_values().tail(10),])#,coef.sort_values().tail(10)
        imp_coef.plot(kind = "barh", color = 'green',fontsize=14)
        plt.title("Top 10 Coefficients Selected by the Adaptive LASSO Model", fontsize = 14)
        plt.show()
        return adaptive_lasso

# variable selection with LASSO for the model
model = Adaptive_LASSO(X_train,y_train,max_iterations = 5000,lasso_iterations = 10, alpha =
0.005074705239490466, tol = 0.001, max_error_up = 5, title = '')
# look at the coefficients in the model
coef = pd.Series(model.coef_, index = X_train.columns)
coef = pd.DataFrame(coef).reset_index()
coef_list = coef.loc[coef[0]!= 0.0]['index'].to_list()
new_X_train = X_train[coef_list]
ada_intercept = model.intercept_
ada_y_test_pred = np.dot(np.array(X_test),np.array(coef[0]))+ada_intercept
ada_y_train_pred = np.dot(np.array(X_train),np.array(coef[0]))+ada_intercept
ada_y_train_score =    ((y_train - ada_y_train_pred)**2).sum()/((y_train - y_train.mean())**2).sum() -1
ada_y_test_score =    ((y_test - ada_y_test_pred)**2).sum()/((y_test - y_test.mean())**2).sum() -1
#用 Adaptive Lasso 分别预测训练集和测试集，并计算均方根误差和拟合优度
ada_y_train_rmse = sqrt(metrics.mean_squared_error(y_train, ada_y_train_pred))
ada_y_test_rmse = sqrt(metrics.mean_squared_error(y_test, ada_y_test_pred))
print('训练集 RMSE: {0}，评分: {1}'.format(ada_y_train_rmse, ada_y_train_score))
print('测试集 RMSE: {0}，评分: {1}'.format(ada_y_test_rmse, ada_y_test_score))
```