数据分析报告

组别：本科 python

姓名：donk

身份证：370******1912

目　录

1 项目概述

红酒数据集包括 1 个质量评分数据和描述红酒化学成分的 11 种特征变量数据，文件格式为 csv，共有 1596 条数据，NA 代表缺失数据。通过数据分析，旨在研究红葡萄酒的不同化学成分对质量评分的影响。在红葡萄酒质量分析的场景中，通过建立多元线性回归模型来预测葡萄酒的质量评分。在建立线性回归模型之后，当给出了红葡萄酒的新的一组化学成分的数据时，可以利用构建的模型预测红葡萄酒的质量评分。

2 分析目标

查看数据可知，数据中有 11 个解释变量，分别为：

- X1=固定酸度
- X2=挥发性酸度
- X3=柠檬酸
- X4=残糖
- X5=氯化物
- X6=游离二氧化硫
- X7=总二氧化硫
- X8=密度
- X9=PH 值
- X10=硫酸盐
- X11=酒精

一个被预测（输出）变量，即因变量：

- Y=质量评分（0 到 10）

据此，本项目的主要目标是通过建立多元线性回归模型来预测红葡萄酒的质量评分。

3 方法概述

为实现上述分析目标，本项目使用的分析方法主要分为五类：**数据缺失值处理方法、数据标准化方法、探索性数据分析方法、描述性数据分析方法、回归预测分析方法**等。

其中，**数据缺失值处理方法**包括：df.fillna()的 bfill 后值填充方法，以及 df.fillna()的 ffill 前值填充方法。

数据标准化方法包括：sklearn 预处理模块(preprocessing)中的 StandardScaler 方法，即标准差标准化。

探索性数据分析方法包括：使用 matplotlib 中的 hist()方法对变量进行直方图密度分析。

描述性数据分析方法包括：使用 describe()方法，查看各数据的基本统计量，包括数据个数 count、平均值 mean、标准差 std、最小值 min、下四分位数 25%、中位数 50%、上四分位数 75%和最大值 max。使用 skew()函数和 kurt()函数分别进行偏度值的计算和峰度值的计算。

回归预测分析方法包括：**多元线性回归中的**主成分回归 (Principal Component Regression, PCR)、岭回归 (Ridge Regression)、Lasso(Least Absolute Shrinkage and Selection Operator)、贝叶斯岭回归 (Bayesian Ridge Regression)、支持向量机回归模型（Support Vector machine Regression，SVR）。

其中，主成分回归 (Principal Component Regression, PCR)是一种结合了主成分分析（PCA）和多元回归分析的统计方法。它主要用于处理自变量间存在多重共线性的情况，以改进最小二乘回归的统计分析方法。具体步骤：标准化解释变量，主成分分析，建立回归模型。

岭回归 (Ridge Regression)通过在损失函数中添加一个正则化项（惩罚项）来解决多重共线性问题。这个正则化项是模型参数的 L2 范数（平方和），其目的是在保证最佳拟合误差的同时，使得参数尽可能的“简单”，即让参数值较小，从而提高模型的泛化能力。

Lasso（Least Absolute Shrinkage and Selection Operator）是一种线性回归模型，它通过在损失函数中加入 L1 正则化项（即系数的绝对值之和）来实现对系数的收缩，从而推动一些系数精确地收缩至零，实现特征选择。Lasso 回归

在处理高维数据和存在多重共线性时特别有用，因为它可以减少模型复杂度，提高模型的泛化能力。

贝叶斯岭回归 (Bayesian Ridge Regression)是一种统计方法，它利用贝叶斯定理来更新对回归参数的估计。这种方法不仅考虑了数据的不确定性，还考虑了模型参数的不确定性，为预测提供了一个更加全面的框架。

支持向量机回归（Support Vector machine Regression, SVR）是一种基于支持向量机理论的回归分析方法。它与传统的回归模型不同，SVR 不试图最小化所有训练数据点的误差平方和，而是寻找一个能够容忍一定误差范围内的最优超平面，同时尽量减少模型的复杂度。SVR 通过引入松弛变量来允许某些数据点超出误差带，从而平衡模型的拟合精度和泛化能力。它利用核函数（如线性核、多项式核、RBF 核等）将原始数据映射到高维特征空间，从而处理非线性关系。模型的关键在于选择合适的核函数和正则化参数 C，以及误差容忍度 ε。通过优化这些参数，SVR 能够有效地捕捉数据中的复杂模式，适用于各种回归问题。

4 详细分析过程

4.1 数据理解

4.1.1 分析数据集的基本结构，查询并输出数据的前 10 行和后 10 行。

使用 Python 编写程序如下：

```
import pandas as pd
# 读取数据
data = pd.read_csv("D:\红酒_1731575676075.csv")
data.columns = ["X1","X2","X3","X4","X5","X6","X7","X8","X9","X10","X11","Y"]
# 输出数据前十行
print(data.head(10))
# 输出数据后十行
print(data.tail(10))
```

查询并输出前 10 行数据如下：

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	Y
0	7.4	0.7	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.7	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	5
6	7.9	0.6	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4	5
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	7
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5	7
9	7.5	0.5	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5	5

查询并输出后 10 行数据如下：

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	Y
1586	6.6	0.725	0.20	7.8	0.073	29.0	79.0	0.99770	3.29	0.54	9.2	5
1587	6.3	0.55	0.15	1.8	0.077	26.0	35.0	0.99314	3.32	0.82	11.6	6
1588	5.4	0.74	0.09	1.7	0.089	16.0	26.0	0.99402	3.67	0.56	11.6	6
1589	6.3	0.51	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1590	6.8	0.62	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0.82	9.5	6
1591	6.2	0.6	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1592	5.9	0.55	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1593	6.3	0.51	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1594	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1595	6.0	0.31	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

使用 Python 编写程序，输出数据维度如下：`print(data.shape)`

数据集的大小为：`(1596, 12)`，即 1596 行，12 列。

4.1.2 识别并输出数据集中所有变量的类型。

使用 Python 编写程序，输出变量信息及变量类型如下：

```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1596 entries, 0 to 1595
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   X1      1596 non-null   float64
 1   X2      1596 non-null   object 
 2   X3      1595 non-null   float64
 3   X4      1595 non-null   float64
 4   X5      1596 non-null   float64
 5   X6      1594 non-null   float64
 6   X7      1596 non-null   float64
 7   X8      1596 non-null   float64
 8   X9      1595 non-null   float64
 9   X10     1596 non-null   float64
 10  X11     1596 non-null   float64
 11  Y       1596 non-null   int64  
dtypes: float64(10), int64(1), object(1)
memory usage: 149.8+ KB
None
```

可知 X2 并不是 float 类型，根据问题描述需要对其进行 float 类型转换，所

以通过编写 python 代码进行类型转换，具体代码和类型转换结果如下图所示：

```python
# float类型转换
import numpy as np
data["X2"]=data["X2"].replace("0.NA",np.nan)
object_list = data["X2"].tolist()
float_list =[float(item) for item in object_list]
data["X2"]=float_list
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1596 entries, 0 to 1595
Data columns (total 12 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   X1      1596 non-null   float64
 1   X2      1595 non-null   float64
 2   X3      1595 non-null   float64
 3   X4      1595 non-null   float64
 4   X5      1596 non-null   float64
 5   X6      1594 non-null   float64
 6   X7      1596 non-null   float64
 7   X8      1596 non-null   float64
 8   X9      1595 non-null   float64
 9   X10     1596 non-null   float64
 10  X11     1596 non-null   float64
 11  Y       1596 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 149.8 KB
None
```

可见，X2 变量已转换为 float 类型，可以继续进行下述分析。

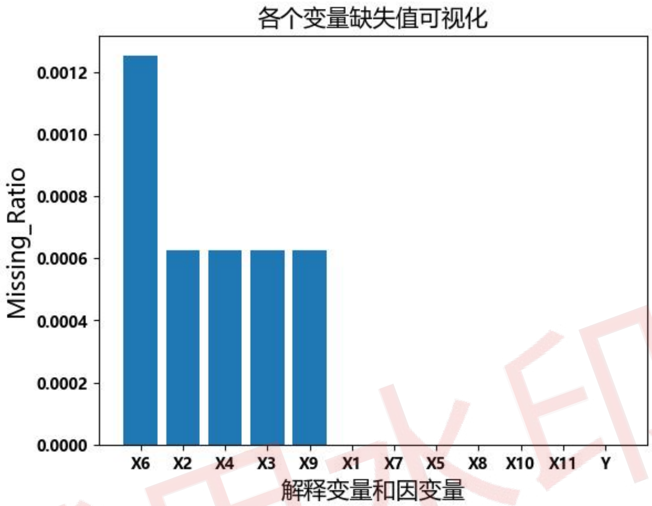
4.2 数据清洗

4.2.1 缺失值处理，利用补缺方式处理，并检验处理结果。

（1）使用 Python 编写代码，计算缺失值比例，并绘制缺失值比例柱状图，原代码与柱状图结果如下图所示：

```python
# 导入模块
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import matplotlib
# 中文显示处理
matplotlib.rc("font",family="MicroSoft YaHei",weight="bold")
# 计算并打印缺失值比例
missing_ratio = data.isnull().mean().sort_values(ascending=False)
print(missing_ratio)
# 绘制缺失值比例柱状图
plt.bar(["X6","X4","X9","X3","X1","X2","X7","X5","X8","X10","X11","Y"],missing_ratio.to_list())
plt.xlabel("解释变量和因变量",fontsize=15)
plt.ylabel("Missing_Ratio",fontsize=15)
plt.title("各个变量缺失值可视化",fontsize=15)
plt.show()
```

```
X6     0.001253
X2     0.000627
X4     0.000627
X3     0.000627
X9     0.000627
X1     0.000000
X7     0.000000
X5     0.000000
X8     0.000000
X10    0.000000
X11    0.000000
Y      0.000000
dtype: float64
```



（2）缺失值填充处理

使用 Python 编写代码利用 df.fillna()进行填充缺失值（同时使用前值填充，和后值填充），具体源代码和可视化结果如下图所示：

```python
# 缺失值填充处理
data1=data.fillna(method="bfill")
data2=data1.fillna(method="ffill")
# 打印缺失值填充处理后的数据集
print(data2)
```

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	Y
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	..
1591	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1592	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1593	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1594	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1595	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

[1596 rows x 12 columns]

（3）使用 Python 编写代码，计算缺失值比例，原代码与缺失值比例结果如

下图所示：

```
# 导入模块
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import matplotlib
# 中文显示处理
matplotlib.rc("font",family="MicroSoft YaHei",weight="bold")
# 计算并打印缺失值比例
missing_ratio = data2.isnull().mean().sort_values(ascending=False)
print(missing_ratio)
```

```
X1     0.0
X2     0.0
X3     0.0
X4     0.0
X5     0.0
X6     0.0
X7     0.0
X8     0.0
X9     0.0
X10    0.0
X11    0.0
Y      0.0
dtype: float64
```

可见，数据集中经过缺失值处理后，已没有缺失值。

4.3 数据分析

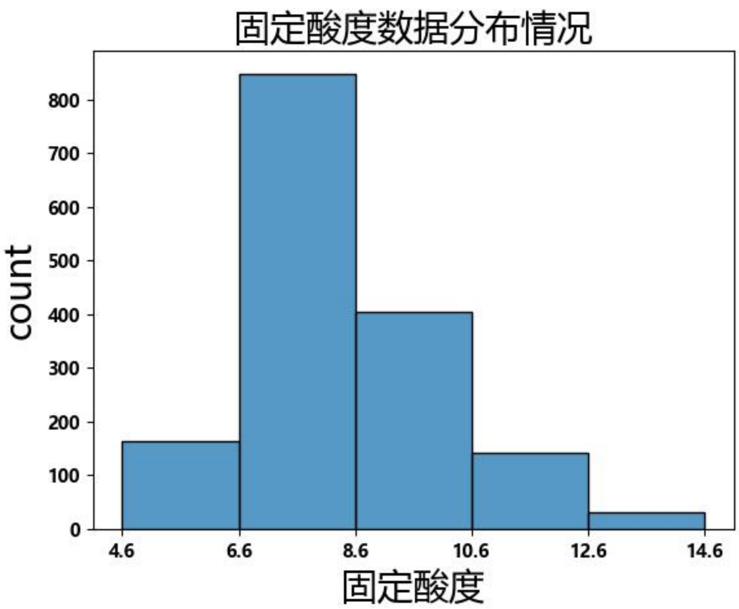
4.3.1 探索性数据分析，提供可视化结果

直方图密度分析

使用 Python 编写代码查看各变量的直方图分布情况。

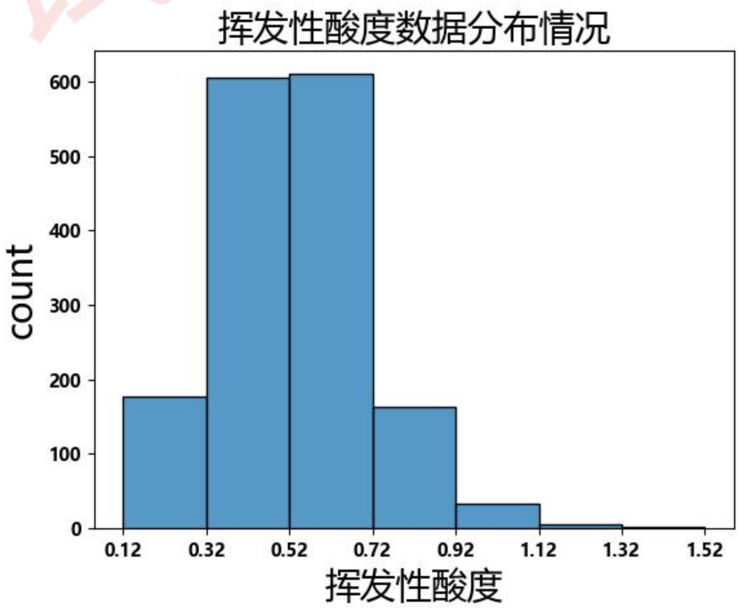
1）查看**固定酸度**的直方图密度分布情况，如下图所示：

```
b = data2["X1"].tolist()
b1 = sorted(b)
import seaborn as sns
sns.histplot(b1, bins=[4.6,6.6,8.6,10.6,12.6,14.6], kde=False)  # kde=False 表示不绘制核密度估计线
plt.xticks(np.arange(min(b1), max(b1), 2))
plt.title("固定酸度数据分布情况",fontsize=20)
plt.xlabel("固定酸度",fontsize=20)
plt.ylabel("count",fontsize=20)
plt.show()
```



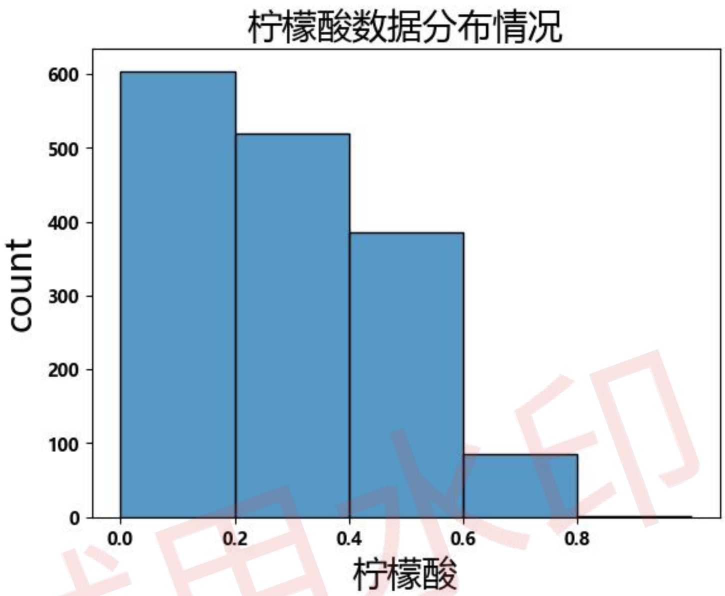
2）查看**挥发性酸度**的直方图密度分布情况，如下图所示：

```
b = data2["X2"].tolist()
b1 = sorted(b)
import seaborn as sns
sns.histplot(b1, bins=[0.12,0.32,0.52,0.72,0.92,1.12,1.32,1.52], kde=False)
plt.xticks(np.arange(min(b1), max(b1), 0.2))
plt.title("挥发性酸度数据分布情况",fontsize=20)
plt.xlabel("挥发性酸度",fontsize=20)
plt.ylabel("count",fontsize=20)
plt.show()
```



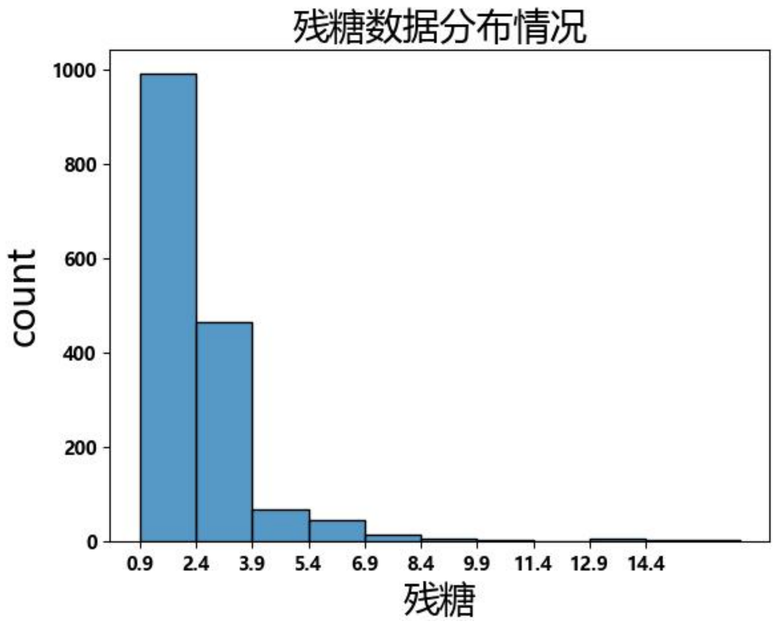
3）查看**柠檬酸**的直方图密度分布情况，如下图所示：

```
b = data2["X3"].tolist()
b1 = sorted(b)
import seaborn as sns
sns.histplot(b1, bins=[0.0,0.2,0.4,0.6,0.8,1], kde=False)
plt.xticks(np.arange(min(b1), max(b1), 0.2))
plt.title("柠檬酸数据分布情况",fontsize=20)
plt.xlabel("柠檬酸",fontsize=20)
plt.ylabel("count",fontsize=20)
plt.show()
```



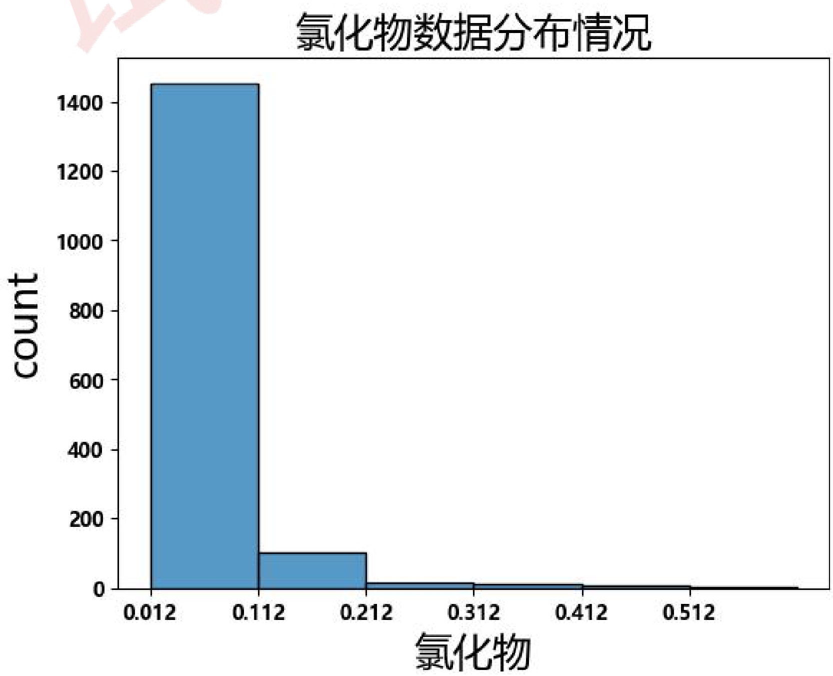
4）查看**残糖**的直方图密度分布情况，如下图所示：

```
b = data2["X4"].tolist()
b1 = sorted(b)
import seaborn as sns
sns.histplot(b1, bins=[0.9,2.4,3.9,5.4,6.9,8.4,9.9,11.4,12.9,14.4,16.9],kde=False)
plt.xticks(np.arange(min(b1), max(b1), 1.5))
plt.title("残糖数据分布情况",fontsize=20)
plt.xlabel("残糖",fontsize=20)
plt.ylabel("count",fontsize=20)
plt.show()
```



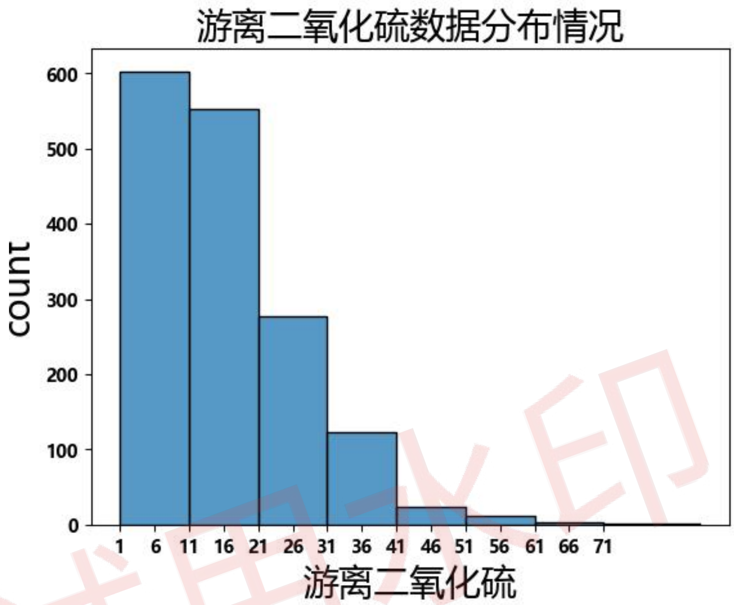
5）查看**氯化物**的直方图密度分布情况，如下图所示：

```
b = data2["X5"].tolist()
b1 = sorted(b)
import seaborn as sns
sns.histplot(b1, bins=[0.012,0.112,0.212,0.312,0.412,0.512,0.612],kde=False)
plt.xticks(np.arange(min(b1), max(b1), 0.1))
plt.title("氯化物数据分布情况",fontsize=20)
plt.xlabel("氯化物",fontsize=20)
plt.ylabel("count",fontsize=20)
plt.show()
```

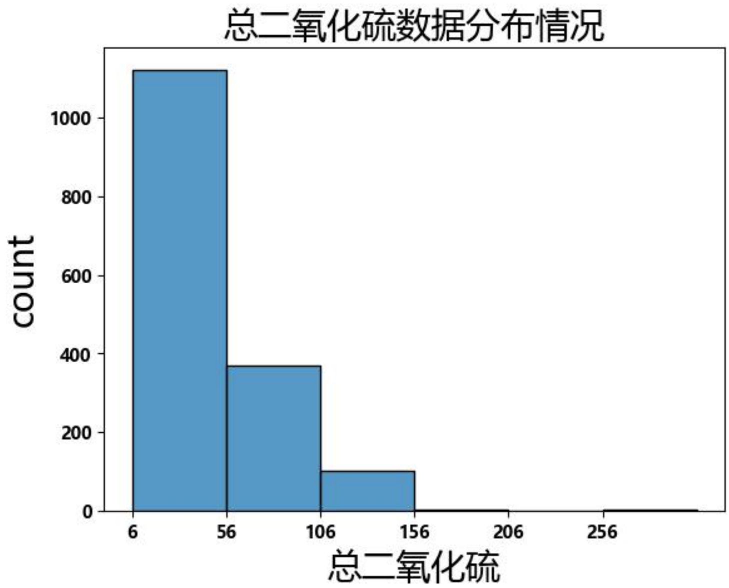
6）查看**游离二氧化硫**的直方图密度分布情况，如下图所示：

```
b = data2["X6"].tolist()
b1 = sorted(b)
import seaborn as sns
sns.histplot(b1, bins=[1,11,21,31,41,51,61,71,85],kde=False)
plt.xticks(np.arange(min(b1), max(b1), 5))
plt.title("游离二氧化硫数据分布情况",fontsize=20)
plt.xlabel("游离二氧化硫",fontsize=20)
plt.ylabel("count",fontsize=20)
plt.show()
```



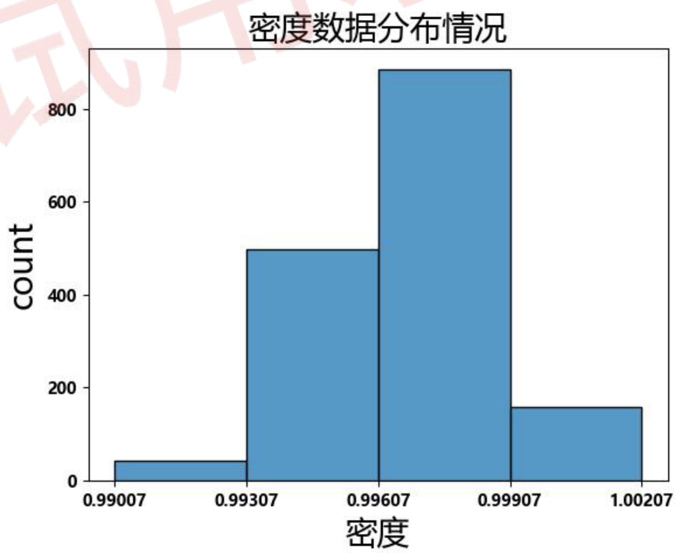
7）查看**总二氧化硫**的直方图密度分布情况，如下图所示：

```
b = data2["X7"].tolist()
b1 = sorted(b)
import seaborn as sns
sns.histplot(b1, bins=[6,56,106,156,206,256,306],kde=False)
plt.xticks(np.arange(min(b1), max(b1), 50))
plt.title("总二氧化硫数据分布情况",fontsize=20)
plt.xlabel("总二氧化硫",fontsize=20)
plt.ylabel("count",fontsize=20)
plt.show()
```



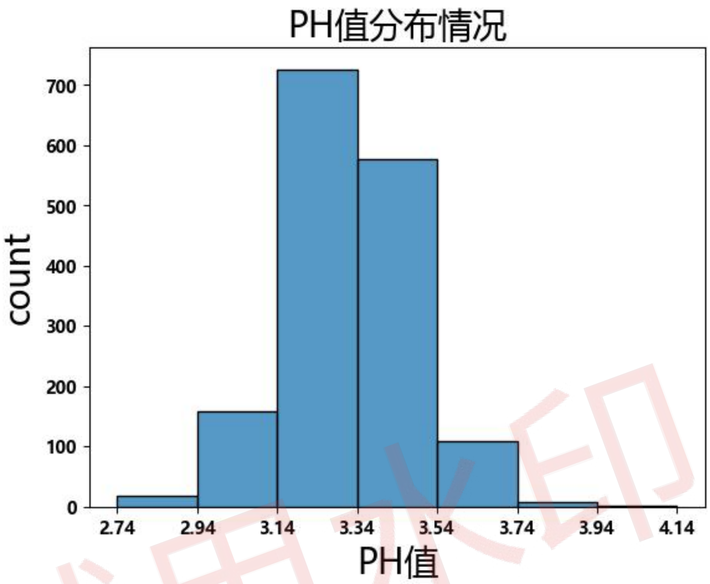
8）查看**密度**的直方图密度分布情况，如下图所示：

```
b = data2["X8"].tolist()
b1 = sorted(b)
import seaborn as sns
sns.histplot(b1, bins=[0.99007,0.99307,0.99607,0.99907,1.00207],kde=False)
plt.xticks(np.arange(min(b1), max(b1), 0.003))
plt.title("密度数据分布情况",fontsize=20)
plt.xlabel("密度",fontsize=20)
plt.ylabel("count",fontsize=20)
plt.show()
```

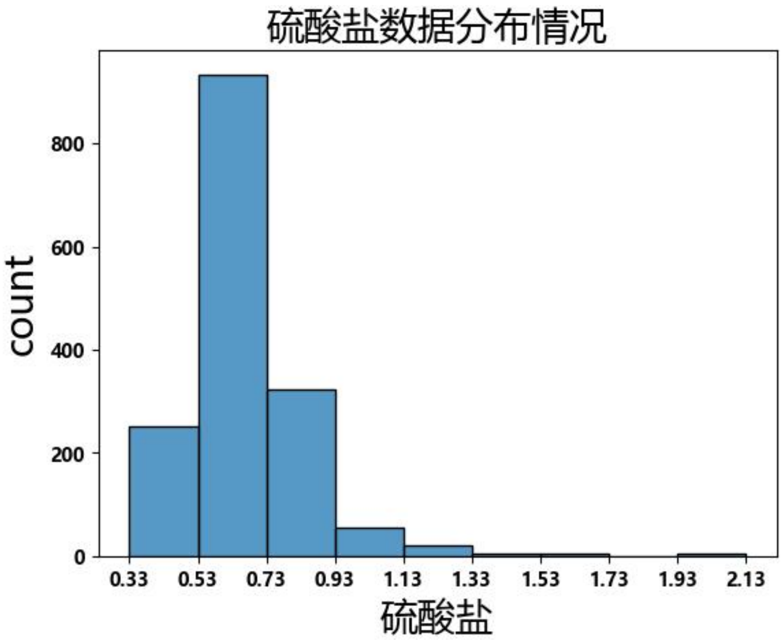
9）查看 **PH 值**的直方图密度分布情况，如下图所示：

```
b = data2["X9"].tolist()
b1 = sorted(b)
import seaborn as sns
sns.histplot(b1, bins=[2.74,2.94,3.14,3.34,3.54,3.74,3.94,4.14],kde=False)
plt.xticks(np.arange(min(b1), max(b1)+0.2, 0.2))
plt.title("PH值分布情况",fontsize=20)
plt.xlabel("PH值",fontsize=20)
plt.ylabel("count",fontsize=20)
plt.show()
```



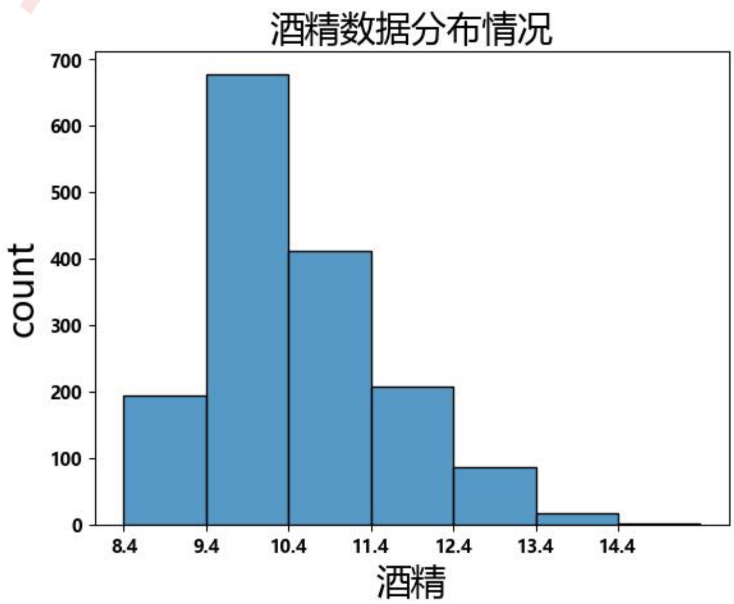
10）查看**硫酸盐**的直方图密度分布情况，如下图所示：

```
b = data2["X10"].tolist()
b1 = sorted(b)
import seaborn as sns
sns.histplot(b1, bins=[0.33,0.53,0.73,0.93,1.13,1.33,1.53,1.73,1.93,2.13],kde=False)
plt.xticks(np.arange(min(b1), max(b1)+0.2, 0.2))
plt.title("硫酸盐数据分布情况",fontsize=20)
plt.xlabel("硫酸盐",fontsize=20)
plt.ylabel("count",fontsize=20)
plt.show()
```



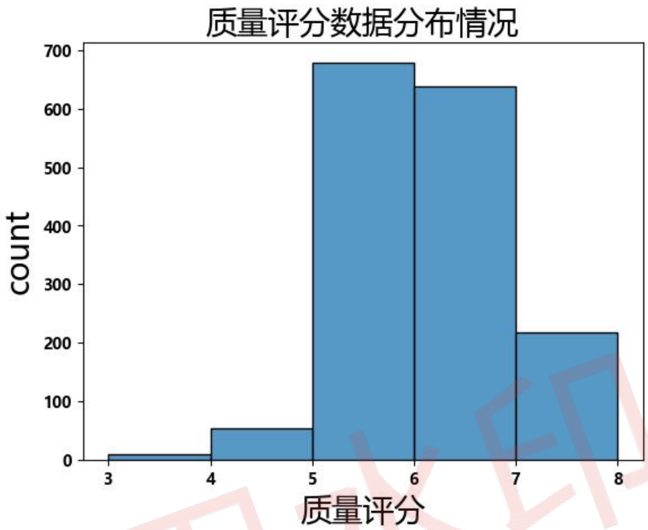
11）查看**酒精**的直方图密度分布情况，如下图所示：

```
b = data2["X11"].tolist()
b1 = sorted(b)
import seaborn as sns
sns.histplot(b1, bins=[8.4,9.4,10.4,11.4,12.4,13.4,14.4,15.4],kde=False)
plt.xticks(np.arange(min(b1), max(b1)+0.2, 1))
plt.title("酒精数据分布情况",fontsize=20)
plt.xlabel("酒精",fontsize=20)
plt.ylabel("count",fontsize=20)
plt.show()
```

12）查看**质量评分**的直方图密度分布情况，如下图所示：

```python
b = data2["Y"].tolist()
b1 = sorted(b)
import seaborn as sns
sns.histplot(b1, bins=5,kde=False)  # kde=False 表示不绘制核密度估计线
plt.xticks(np.arange(min(b1), max(b1)+0.2, 1))
plt.title("质量评分数据分布情况",fontsize=20)
plt.xlabel("质量评分",fontsize=20)
plt.ylabel("count",fontsize=20)
plt.show()
```



4.3.2 描述性数据分析，提供可视化结果

（1）使用 Python 编写代码，查看各化学成分的基本统计量，包括数据个数 count、平均值 mean、标准差 std、最小值 min、下四分位数 25%、中位数 50%、上四分位数 75%和最大值 max，结果如下图所示：

```
             X1           X2           X3           X4           X5  \
count  1596.000000  1596.000000  1596.000000  1596.000000  1596.000000
mean      8.320990     0.527682     0.271310     2.539944     0.087463
std       1.742385     0.179130     0.194745     1.410887     0.047107
min       4.600000     0.120000     0.000000     0.900000     0.012000
25%       7.100000     0.390000     0.090000     1.900000     0.070000
50%       7.900000     0.520000     0.260000     2.200000     0.079000
75%       9.200000     0.640000     0.420000     2.600000     0.090000
max      15.900000     1.580000     1.000000    15.500000     0.611000

             X6           X7           X8           X9          X10  \
count  1596.000000  1596.000000  1596.000000  1596.000000  1596.000000
mean     15.892857    46.500000     0.996747     3.311103     0.658102
std      10.472076    32.915285     0.001889     0.154431     0.169519
min       1.000000     6.000000     0.990070     2.740000     0.330000
25%       7.000000    22.000000     0.995600     3.210000     0.550000
50%      14.000000    38.000000     0.996750     3.310000     0.620000
75%      21.250000    62.000000     0.997842     3.400000     0.730000
max      72.000000   289.000000     1.003690     4.010000     2.000000

            X11            Y
count  1596.000000  1596.000000
mean     10.424593     5.636591
std       1.065996     0.807963
min       8.400000     3.000000
25%       9.500000     5.000000
50%      10.200000     6.000000
75%      11.100000     6.000000
max      14.900000     8.000000
```

（2）进一步地，使用 Python 编写代码，计算各个变量的偏度以分析数据分布形态，如下所示：

```python
# 偏度
df_skew = data2.skew()
print(df_skew)
```

各个变量的偏度为：

```
X1     0.980329
X2     0.673653
X3     0.317450
X4     4.537787
X5     5.676247
X6     1.245122
X7     1.513225
X8     0.071183
X9     0.193837
X10    2.431719
X11    0.858205
Y      0.216333
dtype: float64
```

可以看出，变量偏度均为正值，表明这些变量均为右偏态。且 X4、X5 的偏态程度较大。

（3）此外，使用 Python 编写代码，计算各个变量的峰度，结果如下所示：

```python
# 峰度
kurtosis = data2.kurt()
print(kurtosis)
```

各个变量的峰度值为：

```
X1      1.124484
X2      1.228079
X3     -0.789198
X4     28.576379
X5     41.644639
X6      2.002865
X7      3.799714
X8      0.927173
X9      0.808569
X10    11.743442
X11     0.195945
Y       0.295365
dtype: float64
```

可以看出，X3 峰度值为负值，即 X3 的数据分布比正态分布更平坦，极端值较少。其余变量峰度值均为正值，即其数据分布比正态分布更尖锐，极端值更多，其中 X5 的极端值最多。X3、X8、X9 的峰度值更接近于 1，更接近于正态分布。

4.4 数据整理

4.4.1 根据数据清洗结果对数据集转化并生成新的数据集

通过数据清洗，已将数据集转化为新的数据集 data2。

4.4.2 使用 StandardScaler()方法对数据进行标准化

使用 Python 编写数据标准化程序，如下所示：

```python
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler  # 从预处理中导入标准化模块
import seaborn as sns
import numpy as np
b = data2.iloc[:, :11].values
scaler = StandardScaler()   # 建立标准化模块类对象
X = scaler.fit_transform(b)   # 对原始数据b使用标准化模块进行fit_transform转换,此时X是一个二维数组形式。
y = data2["Y"].values
print("标准化处理后的数据均值为：", np.mean(X))
print("标准化处理后的数据标准差为：", np.std(X))
```

标准化处理后的数据均值为：1.8099493701977488e-15，接近于 0。标准化处理后的数据的标准差为：1。

4.4.3 对数据集随机分割 2/3 用于训练，1/3 用于测试

使用 Python 编写程序，利用 sklearn 中的 train_test_split()函数划分数据集，如下所示：

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,_y_test = train_test_split(X,y,test_size=1/4,random_state=42)
```

4.5 回归预测分析

4.5.1 回归预测

（1）主成分回归模型

1）对各变量进行相关性分析。

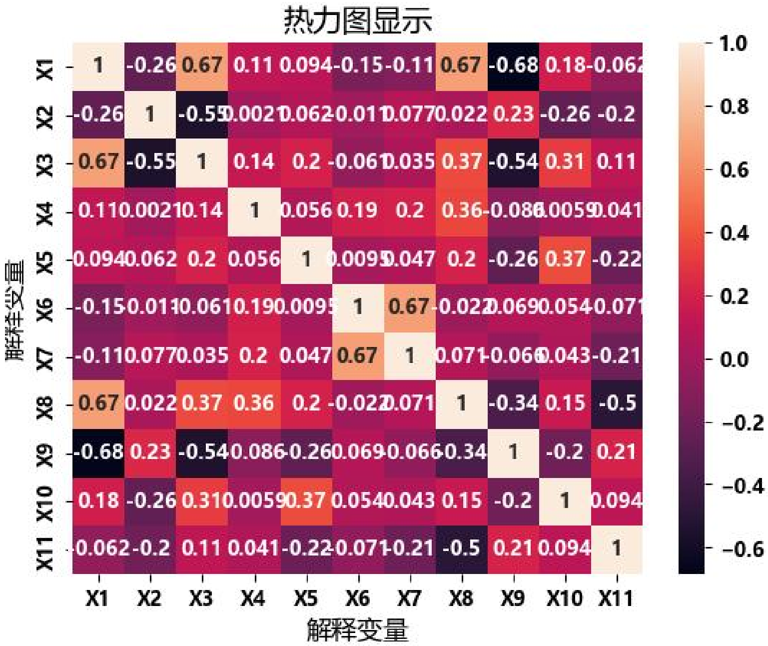
首先通过热力图查看各解释变量之间是否存在相关性，进而判断是否需要对数据进行去冗余处理，使用 Python 编写代码如下：

```
import seaborn as sns
A = pd.DataFrame(X,columns=["X1","X2","X3","X4","X5","X6","X7","X8","X9","X10","X11"])
corr = A.corr()
print(corr)
sns.heatmap(corr,annot=True,xticklabels=corr.columns,yticklabels=corr.columns)
plt.title("热力图显示",fontsize=15)
plt.xlabel("解释变量",fontsize=13)
plt.ylabel("解释变量",fontsize=13)
plt.show()
```

输出相关系数矩阵和热力图结果如下所示：

```
           X1        X2        X3        X4        X5        X6        X7  \
X1   1.000000 -0.255766  0.671616  0.114299  0.093698 -0.154222 -0.113612
X2  -0.255766  1.000000 -0.551980  0.002071  0.061646 -0.011149  0.077108
X3   0.671616 -0.551980  1.000000  0.143022  0.203734 -0.060804  0.034627
X4   0.114299  0.002071  0.143022  1.000000  0.055669  0.186058  0.202680
X5   0.093698  0.061646  0.203734  0.055669  1.000000  0.009495  0.047388
X6  -0.154222 -0.011149 -0.060804  0.186058  0.009495  1.000000  0.667361
X7  -0.113612  0.077108  0.034627  0.202680  0.047388  0.667361  1.000000
X8   0.668261  0.021977  0.365434  0.355163  0.200680 -0.021925  0.071227
X9  -0.683425  0.234108 -0.541959 -0.085800 -0.264797  0.069171 -0.066081
X10  0.183071 -0.260264  0.312729  0.005867  0.371147  0.053827  0.042894
X11 -0.062382 -0.201903  0.108424  0.041404 -0.221218 -0.070977 -0.206559

           X8        X9       X10       X11
X1   0.668261 -0.683425  0.183071 -0.062382
X2   0.021977  0.234108 -0.260264 -0.201903
X3   0.365434 -0.541959  0.312729  0.108424
X4   0.355163 -0.085800  0.005867  0.041404
X5   0.200680 -0.264797  0.371147 -0.221218
X6  -0.021925  0.069171  0.053827 -0.070977
X7   0.071227 -0.066081  0.042894 -0.206559
X8   1.000000 -0.342207  0.148762 -0.496473
X9  -0.342207  1.000000 -0.195769  0.205856
X10  0.148762 -0.195769  1.000000  0.094011
X11 -0.496473  0.205856  0.094011  1.000000
```



2）从热力图中可以看出部分解释变量之间存在较显著的线性相关性，因此需要对各解释变量进行去冗余处理。为此，使用主成分分析进行降维处理，以消除相关性。Python 代码如下：

```
from sklearn.decomposition import PCA
pca = PCA(n_components="mle")
X_pca = pca.fit_transform(X)
print("原始数据的形状：",X.shape)
print("降维后的数据的形状：",X_pca.shape)
print("每个主成分的方差解释比例：",pca.explained_variance_ratio_)
```

```
原始数据的形状： (1596, 11)
降维后的数据的形状： (1596, 10)
每个主成分的方差解释比例： [0.28173343 0.17518159 0.14083183 0.11020567 0.08723989 0.05998084
 0.05310338 0.03840128 0.03139328 0.01651375]
```

使用主成分降维前的数据维度为(1596, 11)，降维后的数据维度为(1596, 10)，且每个主成分的方差解释比例：[0.28173343 0.17518159 0.14083183 0.11020567 0.08723989 0.05998084 0.05310338 0.03840128 0.03139328 0.01651375]。根据该主成分的贡献比率，选取前 10 维数据（信息贡献率约为 99.0%）作为降维数据集。

3）根据主成分分析降维后的数据结果，先在训练集上拟合多元线性回归模型，再在测试集上进行预测。

对数据集进行回归预测，具体代码如下图所示：

```python
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X_pca,y,test_size=1/4,random_state=42)
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train,y_train)
y_pred = lm.predict(X_test)
print(y_pred)
```

打印预测结果如下图所示：

```
[5.74600133 5.81390367 5.28565983 5.55390017 5.49562126 5.17721496
 5.29881404 5.97146251 6.10745743 5.6083871  5.39243867 5.34008218
 5.09971768 5.20967675 5.61086681 5.38846511 5.14805673 5.41999684
 5.02683698 5.29111349 5.53690305 6.21126761 6.01513967 5.10051976
 5.23202641 5.11957146 6.11947991 6.00332009 5.20292016 6.28070847
 6.24795882 4.93630997 5.84272204 5.48823136 6.07784694 5.15345727
 5.36727165 5.46508073 5.42926933 4.9414366  6.06448996 5.27867932
 5.62295315 5.22973929 5.87041835 5.42194881 5.14720527 5.35479243
 4.33837383 6.21961993 5.54707292 5.59380424 4.8658499  6.92559989
 5.7084135  4.96788366 6.56481331 4.81052053 5.13752242 5.35407669
 6.0502861  5.33100308 5.14064848 5.44664756 5.24133418 5.04578852
 5.8439373  5.7860288  6.13319963 5.88563501 5.65970817 5.83002473
 6.27093381 6.13790654 5.34011397 5.13252736 5.21324837 5.26825559
 6.92559989 5.82795309 5.18567699 5.77604208 5.68773296 4.468287
 6.57633276 5.7405653  5.10384531 5.59380424 5.61587131 5.08693745
 5.08657605 6.50857872 5.44957484 5.9783832  6.81409792 5.33778736
 6.01144782 6.28891147 5.48485758 6.05923754 6.45490407 6.41909118
 5.33906437 6.02910822 5.29905553 6.15793478 5.6267037  6.12138869
 5.15910805 5.20629299 5.65821818 5.307598   5.43084799 5.30775665
 5.38049147 5.42324922 6.02943616 5.34342761 5.5036441  5.10412791
 5.5902882  4.99261709 5.88236792 5.94469955 5.45965074 6.15953341
 5.53364391 5.26087017 6.74328344 6.1142185  6.22855716 4.82599974
 5.53378037 5.39492464 5.79033411 6.05884577 5.18825209 5.82454376
 5.89337418 5.22378293 5.78602651 7.00081628 5.04304199 6.00301679
 5.3062805  5.13752242 5.05913482 5.93246768 6.12919742 5.26519025
 5.2661887  5.9311066  5.55360558 5.22112397 5.48269224 5.27548725
 6.45490407 5.16363752 6.27461962 5.11303466 5.38544186 6.28356878
 5.21315781 5.36613968 5.23053613 5.25053747 6.15793478 6.43821304
 5.41982333 6.05923754 6.24106763 5.96702811 5.61932979 5.57140948
 4.98234779 5.79586272 5.86748849 6.08288381 6.24301443 5.51027308
 5.07397426 6.84272672 6.42900407 5.33943228 5.33225988 5.40810074
 6.05482143 5.06759763 4.77646788 5.71132723 5.61853126 6.11638446
 5.5444304  5.0459558  5.63906344 6.65650833 6.09367972 5.04517271
 5.3079626  6.18284157 5.68787702 5.53054922 6.17599345 5.2682069
 5.30879394 5.64489936 5.16568068 5.24640478 5.6083871  5.71487874
 6.2405575  6.1425231  5.84555425 5.72306544 6.31066071 5.31359142
 5.01457811 6.16416098 5.88921563 5.53033925 5.99881589 5.22593157
 6.34585382 5.36727165 6.01668262 5.05736942 5.50807654 6.36403428
 5.63779796 4.84765936 6.15118706 6.33314065 5.06485759 5.7232815
 5.26438852 5.3510193  5.59958278 5.77604208 5.40539487 5.25053747
 6.31668727 4.69565048 5.07636092 5.42404797 5.17389588 6.40470961
 5.24486757 6.08257169 5.62997155 5.62707944 5.59072768 6.48066122
 5.95463597 5.04304199 6.77873818 5.31881465 5.63521038 5.38057308
 5.77999012 6.38255831 6.23019735 6.23467612 6.26765757 6.07717361
 4.92106625 5.76268461 6.09382007 5.10998076 6.61851371 5.4070522
 4.90126325 5.74359073 6.50763829 5.18263261 5.4107752  6.25545148
 5.1952587  5.74241081 6.06711582 5.01673126 4.71125151 5.34850136
 5.86815989 5.1929254  5.45348858 6.24301443 5.3465319  6.36910791
 6.54602015 5.55049979 5.00716171 5.00566002 5.30417484 6.34557511
 5.07083396 5.20709387 6.91161549 6.16106551 5.47673927 5.14848737
 5.01070454 5.22798772 6.08669657 5.53517769 6.34774212 5.14103171
 5.90033581 5.54197101 5.54844917 6.39259081 6.69175765 5.49627501
 6.63557616 5.38116411 6.44933946 5.15407414 6.41553823 6.36303756
 6.2337145  5.68021132 6.10076173 5.30695268 5.79980885 5.75647693
 6.23320023 5.9514194  5.93959429 6.56428429 5.353032   4.71125151
 4.85951069 5.53679661 5.92979713 5.12651315 6.26021699 5.16882373
 6.06152666 6.64796839 6.61958692 4.95963306 5.86363108 5.04578852
 6.5134811  5.24264213 5.55607183 6.03314428 6.39302517 6.60426447
 4.82148033 5.32054065 5.45649988 5.55607183 5.88975483 6.00301679
 5.47226328 5.03103103 5.00965387 5.1817497  5.421694   5.94787051
 5.82795902 6.02495185 6.12919742 5.36427367 5.59072768 4.96381488
 5.38765664 5.09068631 5.91942236 5.83364089 6.16721879 5.17842537
 5.10823286 5.67585122 6.4465308  4.94091066 6.38265144 6.06526687
 5.31795221 5.31425281 6.37628766 6.47854569 5.97674093 6.27139958
 5.2376891  5.426673   5.35395825 5.58207684 5.15650093 5.33887073
 4.94692818 5.71869757 6.3457075  5.03218718 5.51359061 6.31427882
 5.7165023  6.40021819 5.7405653 ]
```

（2）岭回归模型

由于岭回归模型会对解释变量的多重共线性问题进行优化，所以在这里可以直接进行回归模型预测，具体代码如下图所示：

```python
# Ridge
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=1/4,random_state=42)
from sklearn.linear_model import Ridge
rd = Ridge()
rd.fit(X_train,y_train)
y_pred1 = rd.predict(X_test)
print(y_pred1)
```

打印预测结果如下图所示：

```
[5.76272087 5.80319778 5.29057183 5.55592471 5.48620812 5.16887879
 5.30352556 5.98339783 6.13277049 5.64240261 5.38661234 5.34397064
 5.11168177 5.21938356 5.59836127 5.41472764 5.10760111 5.42982352
 5.01005249 5.2791272  5.53590124 6.20778408 5.99441225 5.06944414
 5.22731684 5.1490371  6.05647304 5.99139321 5.18555763 6.29944498
 6.23179353 4.9200054  5.85231784 5.49768406 6.0858122  5.13424369
 5.36764966 5.47972351 5.45554429 4.94304743 6.01528995 5.27080283
 5.64367911 5.21218031 5.86551172 5.43443184 5.16813811 5.33648229
 4.33519579 6.21127355 5.55031128 5.59454317 4.89691369 6.9353406
 5.68096947 4.98120465 6.59248061 4.77739563 5.15491613 5.36084526
 6.05148378 5.32686874 5.14396536 5.45064698 5.22216255 5.04428229
 5.86124608 5.79803283 6.14451268 5.86252947 5.64604413 5.82535577
 6.31392543 6.10574152 5.31424994 5.13176978 5.15269195 5.25912018
 6.9353406  5.78305053 5.1921317  5.75656775 5.69369917 4.40701758
 6.55628024 5.76839729 5.09963428 5.59454317 5.61507041 5.09656515
 5.09680599 6.51570132 5.42875321 6.002802   6.81169824 5.3734798
 6.03557732 6.33918961 5.49534397 6.06581574 6.46475621 6.41763453
 5.29863761 6.02596178 5.30813722 6.16634967 5.70374423 6.0973543
 5.16563003 5.228996   5.6707862  5.2979088  5.4522543  5.37573579
 5.40209183 5.42482435 6.0676986  5.35306834 5.5038256  5.11283039
 5.59886298 4.99249297 5.87584994 5.92553433 5.45469719 6.12033431
 5.55163182 5.26937612 6.74568138 6.17054131 6.22338612 4.79250076
 5.50361337 5.3862594  5.81297941 6.0540429  5.21631326 5.8214495
 5.90987953 5.22678817 5.82171774 6.98717816 5.02927582 6.01686292
 5.28924319 5.15491613 5.07170593 5.92020455 6.14366754 5.24344976
 5.31209525 5.91315964 5.5688606  5.2260073  5.50478779 5.28089191
 6.46475621 5.14811484 6.2447383  5.093151   5.3870265  6.25884812
 5.22660624 5.36476871 5.23557056 5.24831115 6.16634967 6.43427054
 5.45668856 6.06581574 6.21321905 5.98273396 5.66239889 5.58822249
 4.94844136 5.82982952 5.83714422 6.06205432 6.17956965 5.51032877
 5.08826062 6.82586239 6.42276982 5.38089312 5.32695152 5.40966037
 5.9283914  5.08453998 4.77788465 5.70600312 5.61722073 6.08842782
 5.56010123 5.03345301 5.62330397 6.63621187 6.0860451  5.03531766
 5.3131164  6.19382928 5.67937634 5.56492271 6.20832459 5.26760565
 5.31110509 5.66957303 5.1512754  5.20773748 5.64240261 5.72824241
 6.33162036 6.13111193 5.84318779 5.66580664 6.30171769 5.35150736
 5.01742978 6.15825252 5.88420911 5.51322765 6.01048427 5.22801147
 6.3473817  5.36764966 6.04096811 5.04552881 5.53396228 6.34310567
 5.62720418 4.86192888 6.13242493 6.35664931 5.05014397 5.7266027
 5.27408413 5.32644546 5.60442209 5.75656775 5.40498727 5.24831115
 6.35272262 4.69937985 5.05773025 5.4165291  5.18345857 6.39100682
 5.23894992 6.05135604 5.62504727 5.60456715 5.59139925 6.49564199
 5.93712918 5.02927582 6.74897156 5.29974335 5.62566185 5.38415602
 5.79733793 6.34206857 6.23639155 6.24295827 6.31705824 6.03927517
 4.96170614 5.76285536 6.14290483 5.10867348 6.57996858 5.43402019
 4.9157885  5.73064086 6.51979585 5.19068742 5.40981425 6.26789045
 5.23262357 5.74940849 6.06685382 5.03366413 4.67420387 5.37500691
 5.78664374 5.19906443 5.44676447 6.17956965 5.36743202 6.34938248
 6.53097118 5.54288955 5.01910616 5.01346905 5.29889737 6.34704777
 5.05726158 5.14634765 6.94245565 6.15122185 5.48761085 5.13302361
 5.01282274 5.2416354  6.09981354 5.51408602 6.34795486 5.1510815
 5.88376378 5.60430565 5.53762584 6.38641248 6.75043086 5.4760954
 6.59335013 5.44660719 6.46406807 5.15593141 6.40483894 6.40726852
 6.17762526 5.68841572 6.14544867 5.30965734 5.78274868 5.70464776
 6.24876162 5.88830674 5.93535505 6.57208648 5.33741001 4.67420387
 4.87648859 5.54929144 5.94914545 5.14560293 6.30581897 5.15634181
 6.06016472 6.60271411 6.64157573 4.97786243 5.86576031 5.04428229
 6.50286878 5.2434688  5.54507013 6.03511269 6.36799756 6.61373301
 4.80017008 5.31974803 5.50541136 5.54507013 5.94497273 6.01686292
 5.49115148 5.01820684 5.00456457 5.17108835 5.42075982 5.95017884
 5.85986083 6.01224993 6.14366754 5.36409449 5.59139925 4.95775611
 5.3911287  5.08692101 5.94488653 5.84186287 6.13940114 5.18342366
 5.08825108 5.6899229  6.50695822 4.95151126 6.38616702 6.01595109
 5.29833917 5.32334635 6.38334159 6.4939368  6.03159028 6.28054418
 5.21728143 5.42056291 5.37114921 5.60083835 5.16682564 5.32602439
 4.9322604  5.69998239 6.39779689 5.01386326 5.46773266 6.28562047
 5.74443329 6.38770075 5.76839729]
```

（3）Lasso 回归模型

由于 Lasso 回归模型会对解释变量的多重共线性问题进行优化，所以在这里可以直接进行回归模型预测，具体代码如下图所示:

```python
# Lasso
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=1/4,random_state=42)
from sklearn.linear_model import Lasso
la = Lasso(alpha=0.1)
la.fit(X_train,y_train)
y_pred2 = la.predict(X_test)
print(y_pred2)
```

打印预测结果如下图所示：

```
[5.73755392 5.86715448 5.35637611 5.47774454 5.50723041 5.24849448
 5.39530124 5.73725079 5.92227375 5.63833519 5.53974426 5.43323121
 5.2915549  5.28743109 5.60084948 5.51262213 5.2864663  5.45791874
 5.32901767 5.2956281  5.59411956 5.93633441 5.98059733 5.35346486
 5.37834483 5.25187091 6.05971954 5.7870681  5.30383859 6.03986336
 6.07830238 5.23349198 5.79218305 5.54814991 5.78416631 5.30748172
 5.37595009 5.518658   5.41848455 5.19525687 5.84157704 5.38322549
 5.66304567 5.31136721 5.8318765  5.44680173 5.20929459 5.33919083
 4.79201111 6.267489   5.4922893  5.65768836 5.21754221 6.36124577
 5.40072944 5.16164517 6.33906306 5.06602771 5.41876272 5.41193762
 5.91288239 5.40376627 5.40159093 5.53488453 5.25231113 5.16670881
 5.99731944 5.69637718 5.9353487  5.78009784 5.52807633 5.86620117
 6.16944097 6.03067591 5.47167767 5.37086754 5.29176284 5.28740341
 6.36124577 5.68941839 5.39099646 5.67294609 5.65965777 4.84300797
 6.19097148 5.6606475  5.37836576 5.65768836 5.64851238 5.27460392
 5.31329476 6.21690942 5.39892948 5.91940235 6.41451517 5.42479114
 5.88695934 6.17967893 5.57335596 5.90998463 6.07951897 6.19030581
 5.31187828 5.9890853  5.43182419 5.96994611 5.8284238  5.72083041
 5.31374847 5.3873493  5.69665737 5.42358059 5.42962048 5.41008433
 5.41389757 5.40692731 6.05843814 5.39502307 5.50412471 5.23348051
 5.59362399 5.17979523 5.79510376 5.81419565 5.32362655 5.9506179
 5.49318525 5.33651047 6.37967742 6.02196511 5.8937108  5.06820304
 5.52107367 5.51748186 5.63112722 5.85020008 5.40304927 5.85456967
 5.55091867 5.29589208 5.84535525 6.74244793 5.37788911 5.94166676
 5.27678127 5.41876272 5.40498829 5.73124935 5.92132044 5.4552897
 5.46515771 5.9075447  5.68292479 5.39604118 5.53995763 5.36118451
 6.07951897 5.22209881 6.02368271 5.41291527 5.46713859 6.02798206
 5.32107519 5.48628181 5.39068186 5.30772206 5.96994611 6.18883398
 5.51163642 5.90998463 6.11362624 5.73725079 5.67758841 5.58853741
 5.26299335 5.82337908 5.62723972 6.01275612 5.91336307 5.58423263
 5.1969481  6.40650788 6.20190893 5.46836062 5.3601495  5.49589057
 5.62967834 5.23735654 4.99304119 5.73365697 5.59773432 5.96298128
 5.66647543 5.18194762 5.69420727 6.37849182 5.86884773 5.20930606
 5.59572366 5.99988569 5.70511836 5.57042177 5.97697714 5.33961415
 5.4463291  5.56505158 5.31204436 5.34587284 5.63833519 5.72183101
 6.18355295 6.01711082 5.85988311 5.61587693 5.94550295 5.51576969
 5.2196946  5.96679855 5.60326375 5.45719631 5.80260601 5.33073929
 6.16275291 5.37595009 5.8939944  5.27412928 5.50485519 6.09308547
 5.65095643 5.1732483  5.86909753 6.15956349 5.24971449 5.62990116
 5.49571906 5.46764765 5.75930727 5.67294609 5.40834379 5.30772206
 6.27831978 5.00946017 5.25769139 5.48240779 5.38637305 6.126234
 5.37234882 5.75928835 5.64513394 5.60333398 5.4383029  6.12353069
 5.80159333 5.37788911 6.56227323 5.40669643 5.67416811 5.4182901
 5.71337002 6.03090679 6.07157991 6.31967951 6.12549406 5.9390188
 5.3379688  5.73727172 5.9876175  5.43206453 6.13679606 5.54258667
 5.10259855 5.69592145 6.17748468 5.38707656 5.49441673 6.10007464
 5.37258916 5.73076866 5.80355932 5.2121849  4.86557008 5.42021563
 5.98540171 5.46277444 5.43764668 5.91336307 5.37569486 6.08656892
 6.30785758 5.52954815 5.303154   5.24895563 5.42070174 6.2367656
 5.2915549  5.26903727 6.42123965 5.93466411 5.5232166  5.34523072
 5.21461004 5.3590281  5.86159527 5.48088868 6.18087399 5.28210819
 5.92281118 5.57137508 5.66960407 6.11770014 6.47810087 5.61800778
 6.2335951  5.54863602 6.25147986 5.2361325  6.19030581 6.32546014
 5.91093391 5.74233536 5.92640159 5.50394374 5.70729229 5.58403302
 6.0966962  5.60984107 5.85699078 6.3446263  5.32487292 4.86557008
 5.18125558 5.52062942 5.94431472 5.23061314 6.20430709 5.35029234
 5.8760867  6.35673445 6.21397925 5.23225104 5.68281813 5.16670881
 6.06788747 5.31498196 5.56338671 6.05464644 6.13033084 6.42591981
 5.04991449 5.41318057 5.4245663  5.56338671 5.76550378 5.94166676
 5.59146214 5.17640934 5.26906967 5.3807275  5.46760579 5.91385059
 5.82460916 5.80112473 5.92132044 5.48213908 5.4383029  5.27896606
 5.4949278  5.24270036 5.94816579 5.82024097 5.98615513 5.45698838
 5.39967485 5.48360144 5.79401804 5.07166179 6.19320759 5.81103864
 5.53655081 5.34814197 6.11361678 6.2094301  5.99898028 5.98198
 5.38032773 5.51742651 5.41849401 5.66087979 5.25453236 5.36432805
 5.13745724 5.73051886 6.31316699 5.19380054 5.36675721 6.00211839
 5.72105867 6.26091912 5.6606475 ]
```

（4）贝叶斯岭回归模型

由于贝叶斯回归模型会对解释变量的多重共线性问题进行优化，所以在这里可以直接进行回归模型预测，具体代码如下图所示:

```
# Bayesian
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=1/4,random_state=42)
from sklearn.linear_model import BayesianRidge
BayesianRidge_reg = BayesianRidge()
BayesianRidge_reg.fit(X_train,y_train)
y_pred3 = BayesianRidge_reg.predict(X_test)
print(y_pred3)
```

打印预测结果如下图所示：

```
[5.7565173  5.79969089 5.3014304  5.55869012 5.49200748 5.17760367
 5.30800616 5.96127242 6.10836861 5.6371829  5.39952864 5.33779445
 5.12550352 5.22316448 5.59689757 5.41133754 5.11851723 5.42490078
 5.01898736 5.28399032 5.53079602 6.20085445 5.99653708 5.08468356
 5.22814633 5.14291588 6.06534818 5.98282593 5.19209808 6.28694386
 6.23430727 4.93712324 5.843428   5.49660801 6.07420731 5.14346231
 5.37627895 5.47562782 5.44724924 4.95910339 6.02205658 5.27135979
 5.64090205 5.21506916 5.86354537 5.43777515 5.17338001 5.34137378
 4.35985515 6.20865773 5.54869178 5.59241734 4.89843593 6.91674474
 5.6821031  4.98627768 6.57152622 4.80101291 5.15716384 5.35670235
 6.05112556 5.33234461 5.14969345 5.44699733 5.23519935 5.04865274
 5.8631987  5.78545101 6.13073922 5.86320309 5.64096874 5.82152761
 6.29501519 6.1093738  5.33570578 5.13809348 5.15855327 5.26392495
 6.91674474 5.79178053 5.20185304 5.755739   5.68354447 4.44687622
 6.55163159 5.76279778 5.11587284 5.59241734 5.61649031 5.1010932
 5.11362785 6.50189761 5.43587342 5.98635749 6.79779768 5.36946222
 6.01356625 6.31817656 5.50300931 6.05159619 6.44857473 6.40791956
 5.31182489 6.02395639 5.30443226 6.16508155 5.68163919 6.09490805
 5.16747562 5.22226837 5.66013814 5.29726829 5.44226567 5.37100875
 5.40337902 5.42621123 6.04660641 5.35105382 5.50281178 5.11525313
 5.59111027 5.00580925 5.86762177 5.9193026  5.45392991 6.12722433
 5.55256292 5.27462124 6.73246409 6.15443178 6.22424821 4.81572408
 5.50656293 5.39257942 5.80914242 6.0525313  5.20818371 5.81052932
 5.89124291 5.22953067 5.80698768 6.9637528  5.03546374 6.00490943
 5.29609496 5.15716384 5.07874316 5.9177026  6.13426822 5.25264522
 5.29569697 5.9155396  5.57206417 5.24318761 5.50052348 5.28740924
 6.44857473 5.1604365  6.24742875 5.10555196 5.389434   6.25709764
 5.23068985 5.37086715 5.24723404 5.25237066 6.16508155 6.42637158
 5.44346788 6.05159619 6.21035936 5.95991383 5.65607686 5.57593283
 4.96670039 5.81230249 5.84223146 6.0576637  6.18381    5.52046447
 5.08854986 6.80838252 6.41223724 5.36479201 5.33161804 5.40725313
 5.94226163 5.08415871 4.79246411 5.69382379 5.61091372 6.09246435
 5.56380883 5.04473049 5.62488567 6.62835974 6.07604777 5.04537154
 5.32330349 6.18188834 5.68165153 5.55185021 6.19553881 5.26918598
 5.31772399 5.66268365 5.14929638 5.22816143 5.6371829  5.71979692
 6.30453279 6.12575566 5.83268961 5.68125961 6.29020194 5.34455902
 5.02626036 6.14833362 5.88550974 5.51836868 6.00271629 5.2349183
 6.33945697 5.37627895 6.02931372 5.06520904 5.52606858 6.33881944
 5.62332461 4.87305    6.1274431  6.34226522 5.06120955 5.71807424
 5.27947586 5.34326461 5.61784348 5.755739   5.40721927 5.25237066
 6.32388133 4.72390543 5.07218047 5.42611335 5.18551702 6.38816732
 5.25138869 6.04864369 5.61968718 5.61504189 5.59014894 6.47852319
 5.93420356 5.03546374 6.75480465 5.30986862 5.62163914 5.3870483
 5.79419935 6.34343339 6.23435801 6.22988736 6.29580634 6.04591342
 4.9655966  5.76504562 6.12848876 5.11606657 6.57433158 5.43581606
 4.92660237 5.72555781 6.50661434 5.19824413 5.40402699 6.2563993
 5.22610201 5.75003302 6.06307542 5.03839629 4.69997521 5.36572932
 5.80954339 5.20182181 5.4451499  6.18381    5.36519117 6.34415928
 6.52097292 5.54446275 5.02916472 5.02744594 5.30344231 6.34034491
 5.07452269 5.16096651 6.91311555 6.14916115 5.49337389 5.14251103
 5.02293105 5.24631594 6.09289428 5.5173713  6.3322717  5.1586684
 5.8887972  5.59287198 5.54122394 6.37128616 6.72410236 5.48491667
 6.59275555 5.44973186 6.44726458 5.15659352 6.39858511 6.38361479
 6.18024752 5.67924422 6.12792784 5.31702078 5.7830774  5.71413655
 6.24347246 5.89712311 5.92764849 6.56239102 5.34800505 4.69997521
 4.88003266 5.54098671 5.9367847  5.14893395 6.2905782  5.16948717
 6.04830123 6.60015823 6.6262621  4.99431501 5.85734024 5.04865274
 6.4913292  5.24545785 5.54479614 6.03500947 6.36644827 6.60306802
 4.82292328 5.31916198 5.49635577 5.54479614 5.92796886 6.00490943
 5.4955171  5.02591302 5.01084097 5.1851038  5.42571374 5.94698844
 5.86091462 6.00258637 6.13426822 5.37092013 5.59014894 4.96249328
 5.39333476 5.09483648 5.93360186 5.83561308 6.13438333 5.19568136
 5.09911145 5.68394785 6.47387458 4.95583442 6.36474658 6.01983591
 5.30909904 5.32891835 6.37390124 6.47660095 6.00934958 6.28122247
 5.2297766  5.42563722 5.36274204 5.59595096 5.16838281 5.32608535
 4.95110726 5.69957578 6.37801418 5.0255235  5.46863677 6.2828461
 5.73292588 6.38009232 5.76279778]
```

（5）支持向量机模型

由于支持向量机模型本身不能对解释变量进行去冗余处理，下面通过 python 编写代码，利用主成分变换后的数据进行支撑向量机模型拟合，具体代码如下图所示：

```python
# SVM
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X_pca,y,test_size=1/4,random_state=42)
svr = SVR(kernel='linear', C=200, gamma=0.1, epsilon=0.01)
svr.fit(X_train,y_train)
y_pred5 = svr.predict(X_test)
print(y_pred5)
```

打印预测结果如下图所示：

```
[5.67902291 5.84290707 5.15022746 5.65164212 5.55110446 5.02469956
 5.19907063 6.10258515 6.11171841 5.70535853 5.22880115 5.38054046
 5.02809985 5.05965874 5.62511642 5.28972572 5.21754288 5.34259195
 4.96200728 5.21135447 5.44407322 6.18286788 6.06848915 4.97660712
 5.14351708 4.98053994 6.25494442 5.97507912 5.23117981 6.22476003
 6.33194145 4.84799604 5.83896464 5.43509983 6.13427837 5.0916716
 5.33766386 5.3771569  5.35306289 4.92864759 5.91441437 5.10523432
 5.54179918 5.20520302 5.80433841 5.39720933 5.06686171 5.23887643
 4.44708426 6.11609322 5.40425397 5.62699829 4.72929403 6.96871512
 5.79602379 4.94223651 6.57003178 5.03246553 5.04108089 5.33890659
 6.00869146 5.20653013 5.0426856  5.25924009 5.27383303 5.01562906
 5.76739875 5.78806847 6.18380459 5.85323274 5.52982862 5.83906954
 6.42269821 6.15950663 5.52802136 5.03916846 5.37594942 5.19026466
 6.96871512 5.78840539 4.99210356 5.82231541 5.64968794 4.75680987
 6.72536079 5.55288021 5.10692203 5.62699829 5.58677109 5.03501544
 5.00135435 6.47800733 5.29589676 5.99558503 6.82023634 5.29426013
 6.06829419 6.24945302 5.30650595 5.94197233 6.39673847 6.45681563
 5.22873416 5.99263148 5.26126455 6.1390658  5.76734646 6.01458922
 5.15896144 5.02534711 5.61117962 5.19413215 5.33239131 5.1708532
 5.30455367 5.24278949 6.19391795 5.28441785 5.5012492  5.03235086
 5.49259419 4.9061666  5.81020672 5.93345763 5.35391321 6.17194243
 5.61636571 5.15974915 6.84118483 6.13540744 6.23428593 5.0463294
 5.41312898 5.24040259 5.6100889  6.07312231 5.03188651 5.85687265
 5.88925645 5.11353522 5.94307996 7.10936516 4.95385888 5.95196694
 5.20703781 5.04108089 4.90985492 5.82940654 6.18094374 5.15937567
 5.30859287 5.99034682 5.37509621 5.09931045 5.33544149 5.16880699
 6.39673847 5.11180146 6.22082423 5.17819802 5.29711999 6.38414781
 5.05196085 5.1802219  5.19393558 5.16872688 6.1390658  6.50403064
 5.3531506  5.94197233 6.24911267 6.0972146  5.51925088 5.53980174
 4.88269722 5.81816259 5.77343636 6.2032696  6.18609172 5.33845916
 5.00367068 6.99728638 6.46657915 5.14174713 5.23859398 5.38436819
 5.90127369 4.97542877 4.73982776 5.65512301 5.54337355 6.07763974
 5.37279512 5.01841418 5.71843654 6.61644568 6.13750784 4.93567992
 5.21958696 6.39183553 5.64694259 5.39524347 6.19522836 5.2800476
 5.2245392  5.49570373 5.10235385 5.46011358 5.70535853 5.66377865
 6.25742027 6.16779503 5.8977804  5.64502132 6.27595412 5.18978009
 4.97241529 6.18627735 5.78281988 5.38833393 5.89758035 5.05582257
 6.3583874  5.33766386 6.06424315 4.971335   5.38054183 6.39153408
 5.53976495 4.75917174 6.08819552 6.39939839 4.92360722 5.6284919
 5.0888515  5.20042527 5.60857233 5.82231541 5.31130061 5.16872688
 6.51339019 4.71257377 4.94219244 5.33175661 5.04675615 6.30368041
 5.09615415 6.04345817 5.59292543 5.49950721 5.4796242  6.42284529
 6.02523295 4.95385888 6.95298869 5.24876094 5.59600094 5.20295808
 5.6637307  6.34618181 6.15160537 6.33186223 6.26312853 6.00037279
 4.86339824 5.83570027 6.05972618 4.98676349 6.76345465 5.40242601
 4.9354583  5.70810324 6.48480946 5.09636758 5.4650772  6.29631255
 5.02544568 5.66583975 5.9913586  4.92233325 4.70248274 5.18496419
 6.0317767  5.11812881 5.28008021 6.18609172 5.22669688 6.30503573
 6.50375718 5.45034747 4.89318396 4.98072795 5.17335129 6.31620271
 4.92826218 5.32036212 6.95253599 6.13699708 5.53405057 5.05428919
 4.91397645 5.07074472 6.17363871 5.47731306 6.42914654 5.01202313
 5.88420021 5.45463109 5.41386109 6.4969854  6.87952037 5.63322231
 6.66543548 5.36467101 6.52662493 5.09754239 6.44230789 6.64095596
 6.35227445 5.74975727 6.02251405 5.14388407 5.73075894 5.6662247
 6.33197561 5.81832584 6.03154872 6.57536639 5.35970576 4.70248274
 4.72484155 5.31900375 5.99130032 4.96301688 6.36589469 5.16190575
 6.06882126 6.71907351 6.64040762 4.89426626 5.87111454 5.01562906
 6.46714406 5.15076365 5.48706203 5.95050937 6.49491986 6.5754071
 4.69747632 5.2442595  5.36384629 5.48706203 5.87146381 5.95196694
 5.35313665 4.88238062 4.78724982 5.34991592 5.33199942 5.93994539
 5.73053688 5.92872078 6.18094374 5.23582457 5.4796242  4.9242031
 5.36582323 4.96397307 6.01030059 5.83731078 6.22448637 5.08669405
 4.98569663 5.49118162 6.29151451 4.88366322 6.51067597 6.06135545
 5.23197181 5.18181843 6.36055086 6.42930806 5.92768823 6.37826353
 5.06375982 5.38979876 5.33218044 5.52539224 5.04556867 5.27584056
 4.89562844 5.68737489 6.44286883 4.93909613 5.5488484  6.29034605
 5.75506659 6.44588173 5.55288021]
```

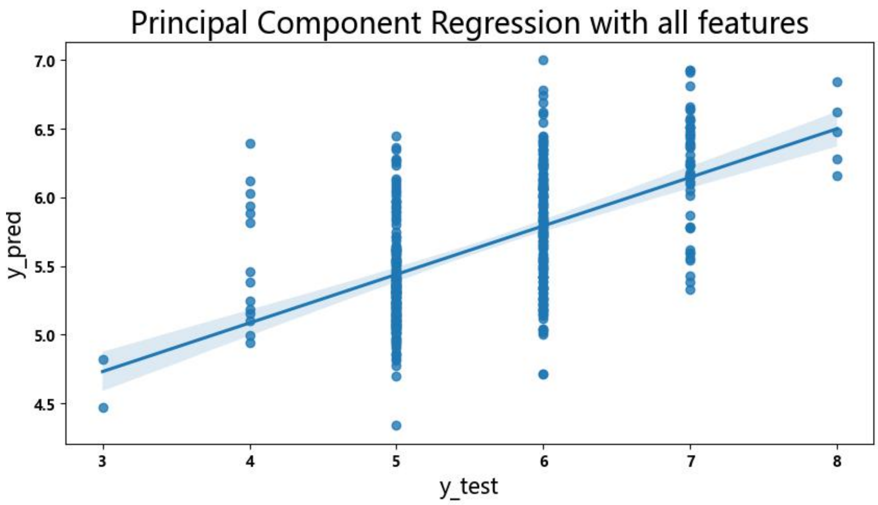
4.5.2 模型可靠性分析及参数检验

（1）主成分回归模型

利用 Python 编写代码，在测试集上分析主成分多元回归（Principal Component Regression）模型的可靠性并检验模型参数,具体代码如下图所示：

```python
from sklearn.model_selection import cross_val_score
from sklearn import metrics
X_train,X_test,y_train,y_test = train_test_split(X_pca,y,test_size=1/4,random_state=42)
cv_lm = cross_val_score(estimator=lm,X=X_train,y=y_train,cv=10)
r2 = lm.score(X_test,y_test)
n = X_test.shape[0]
p = X_test.shape[1]
lm_adjusted_R2 = 1-(1-r2)*(n-1)/(n-p-1)
lm_RMSE = np.sqrt(metrics.mean_squared_error(y_test,y_pred))
lm_R2 = lm.score(X_test,y_test)
lmCV_R2 = cv_lm.mean()
print("RMSE:",round((lm_RMSE),4))
print("R2",round(lm_R2,4))
print("Adjusted R2:",round(lm_adjusted_R2,4))
print("Cross Validated R2:",round(lmCV_R2,4))
plt.figure(figsize=(10,5))
sns.regplot(x=y_test,y=y_pred)
plt.title("Principal Component Regression with all features",fontsize=20)
plt.xlabel("y_test",fontsize=15)
plt.ylabel("y_pred",fontsize=15)
plt.show()
```

模型回归情况显示与参数显示情况如下图所示：



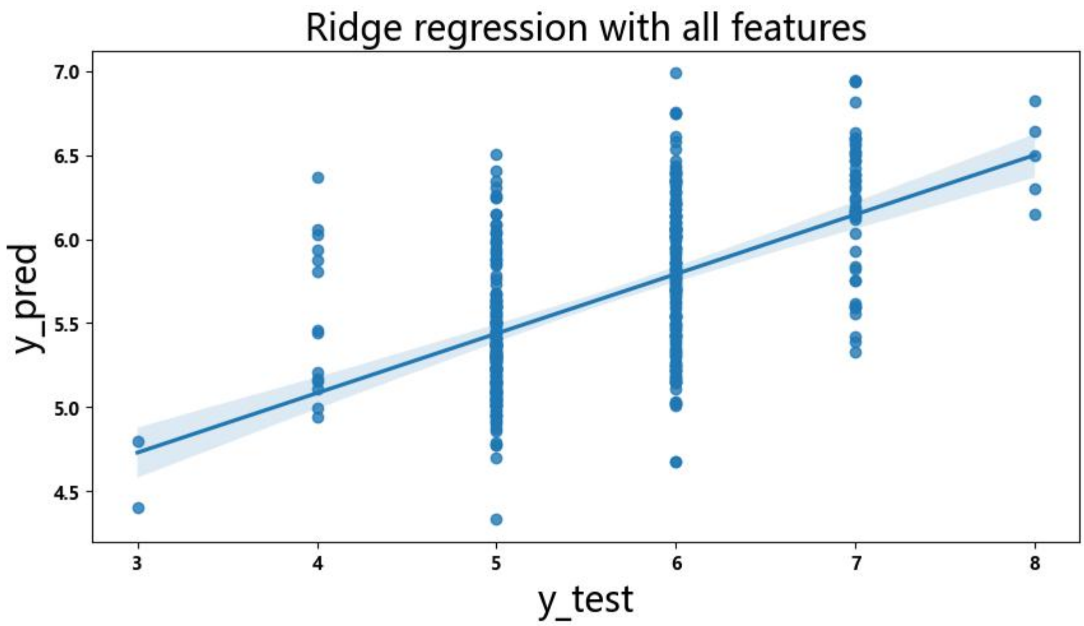

```
RMSE: 0.6698
R2 0.3111
Adjusted R2: 0.2934
Cross Validated R2: 0.3395
```

（2）岭回归模型

利用 Python 编写代码，分析岭回归（Ridge Regression）模型可靠性及参数检验，具体代码如下图所示：

```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=1/4,random_state=42)
cv_ridge = cross_val_score(estimator=rd,X=X_train,y=y_train,cv=10)
r2 = rd.score(X_test,y_test)
n = X_test.shape[0]
p = X_test.shape[1]
ridge_adjusted_R2 = 1-(1-r2)*(n-1)/(n-p-1)
ridge_RMSE = np.sqrt(metrics.mean_squared_error(y_test,y_pred1))
ridge_R2 = rd.score(X_test,y_test)
ridgeCV_R2 = cv_ridge.mean()
print("RMSE:",round((ridge_RMSE),4))
print("R2",round(ridge_R2,4))
print("Adjusted R2:",round(ridge_adjusted_R2,4))
print("Cross Validated R2:",round(ridgeCV_R2,4))
plt.figure(figsize=(10,5))
sns.regplot(x=y_test,y=y_pred1)
plt.title("Ridge regression with all features",fontsize=20)
plt.xlabel("y_test",fontsize=20)
plt.ylabel("y_pred",fontsize=20)
plt.show()
```

模型回归情况显示与参数显示情况如下图所示：



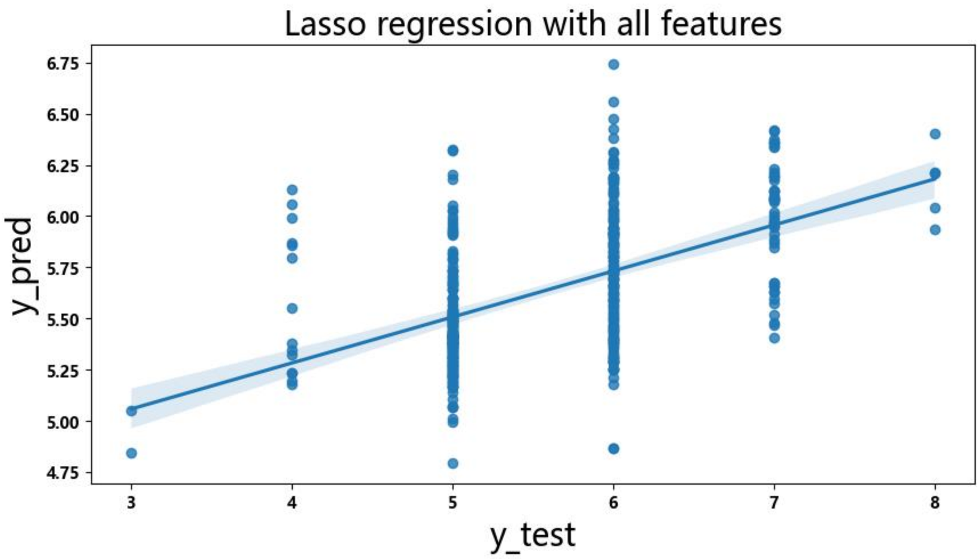

```
RMSE: 0.6708
R2 0.3091
Adjusted R2: 0.2895
Cross Validated R2: 0.3391
```

（3）Lasso 回归模型

利用 Python 编写代码，分析 Lasso 回归模型可靠性及参数检验,具体代码如下图所示：

```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=1/4,random_state=42)
cv_la = cross_val_score(estimator=la,X=X_train,y=y_train,cv=10)
r2 = la.score(X_test,y_test)
n = X_test.shape[0]
p = X_test.shape[1]
la_adjusted_R2 = 1-(1-r2)*(n-1)/(n-p-1)
la_RMSE = np.sqrt(metrics.mean_squared_error(y_test,y_pred2))
la_R2 = la.score(X_test,y_test)
laCV_R2 = cv_la.mean()
print("RMSE:",round((la_RMSE),4))
print("R2",round(la_R2,4))
print("Adjusted R2:",round(la_adjusted_R2,4))
print("Cross Validated R2:",round(laCV_R2,4))
plt.figure(figsize=(10,5))
sns.regplot(x=y_test,y=y_pred2)
plt.title("Lasso regression with all features",fontsize=20)
plt.xlabel("y_test",fontsize=20)
plt.ylabel("y_pred",fontsize=20)
plt.show()
```

模型回归情况显示与参数显示情况如下图所示：



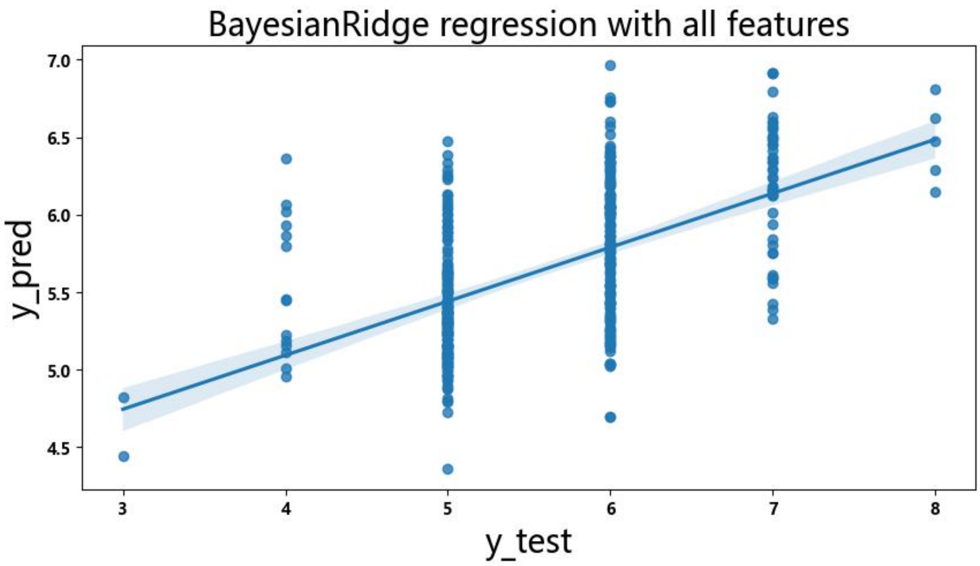

```
RMSE: 0.6936
R2 0.2613
Adjusted R2: 0.2403
Cross Validated R2: 0.2998
```

（4）贝叶斯岭回归模型

利用 Python 编写代码，分析贝叶斯岭回归（Bayesian Ridge Regression）模型可靠性及参数检验,具体代码如下图所示：

```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=1/4,random_state=42)
cv_BayesianRidge = cross_val_score(estimator=BayesianRidge_reg,X=X_train,y=y_train,cv=10)
r2 = BayesianRidge_reg.score(X_test,y_test)
n = X_test.shape[0]
p = X_test.shape[1]
BayesianRidge_adjusted_R2 = 1-(1-r2)*(n-1)/(n-p-1)
BayesianRidge_RMSE = np.sqrt(metrics.mean_squared_error(y_test,y_pred3))
BayesianRidge_R2 = BayesianRidge_reg.score(X_test,y_test)
BayesianRidgeCV_R2 = cv_BayesianRidge.mean()
print("RMSE:",round((BayesianRidge_RMSE),4))
print("R2",round(BayesianRidge_R2,4))
print("Adjusted R2:",round(BayesianRidge_adjusted_R2,4))
print("Cross Validated R2:",round(BayesianRidgeCV_R2,4))
plt.figure(figsize=(10,5))
sns.regplot(x=y_test,y=y_pred3)
plt.title("BayesianRidge regression with all features",fontsize=20)
plt.xlabel("y_test",fontsize=20)
plt.ylabel("y_pred",fontsize=20)
plt.show()
```

模型回归情况显示与参数显示情况如下图所示：



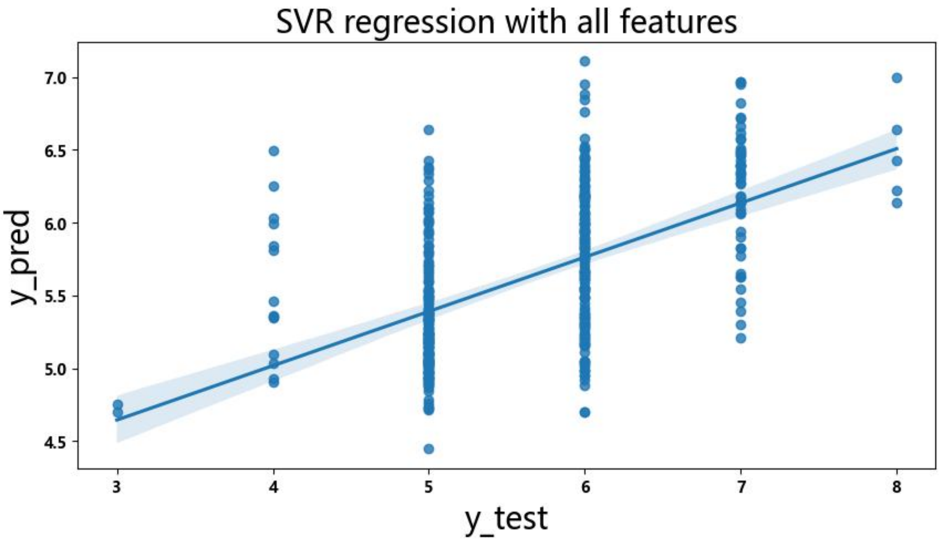

```
RMSE: 0.6703
R2 0.31
Adjusted R2: 0.2904
Cross Validated R2: 0.3403
```

（5）支持向量机模型

利用 Python 编写代码，分析支持向量机（Support Vector machine Regression，SVR）模型可靠性及参数检验,具体代码如下图所示：

```
from sklearn.model_selection import cross_val_score
from sklearn import metrics
X_train,X_test,y_train,y_test = train_test_split(X_pca,y,test_size=1/4,random_state=42)
SVR = cross_val_score(estimator=svr,X=X_train,y=y_train,cv=10)
r2 = svr.score(X_test,y_test)
n = X_test.shape[0]
p = X_test.shape[1]
SVR_adjusted_R2 = 1-(1-r2)*(n-1)/(n-p-1)
SVR_RMSE = np.sqrt(metrics.mean_squared_error(y_test,y_pred5))
SVR_R2 = rf_r.score(X_test,y_test)
SVRCV_R2 = SVR.mean()
print("RMSE:",round((SVR_RMSE),4))
print("R2",round(SVR_R2,4))
print("Adjusted R2:",round(SVR_adjusted_R2,4))
print("Cross Validated R2:",round(SVRCV_R2,4))
plt.figure(figsize=(10,5))
sns.regplot(x=y_test,y=y_pred5)
plt.title("SVR regression with all features",fontsize=20)
plt.xlabel("y_test",fontsize=20)
plt.ylabel("y_pred",fontsize=20)
plt.show()
```

模型回归情况显示与参数显示情况如下图所示：




```
RMSE: 0.684
R2 0.2816
Adjusted R2: 0.2631
Cross Validated R2: 0.3252
```

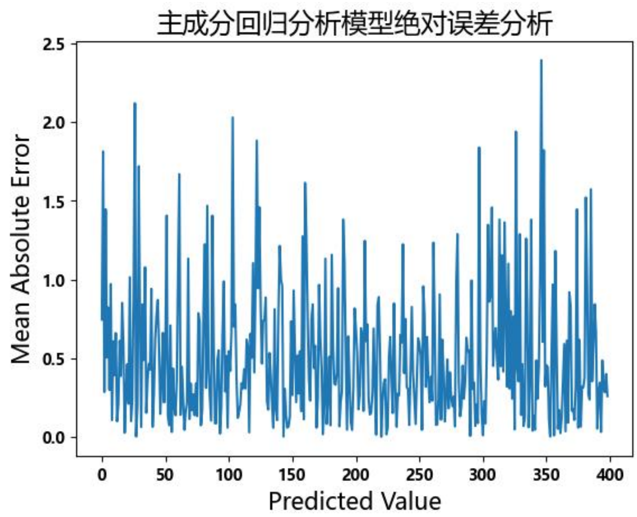
4.5.3 误差分析

（1）主成分回归分析模型

使用 Python 编写代码对主成分多元回归分析模型进行绝对误差分析，具体代码如下图所示：

```
errors = list()
for i in range(len(y_test)):
    err = abs(y_test[i]-y_pred[i])
    errors.append(err)
plt.plot(errors)
plt.xlabel("Predicted Value",fontsize=15)
plt.ylabel("Mean Absolute Error",fontsize=15)
plt.title("主成分回归分析模型绝对误差分析",fontsize=17)
plt.show()
```

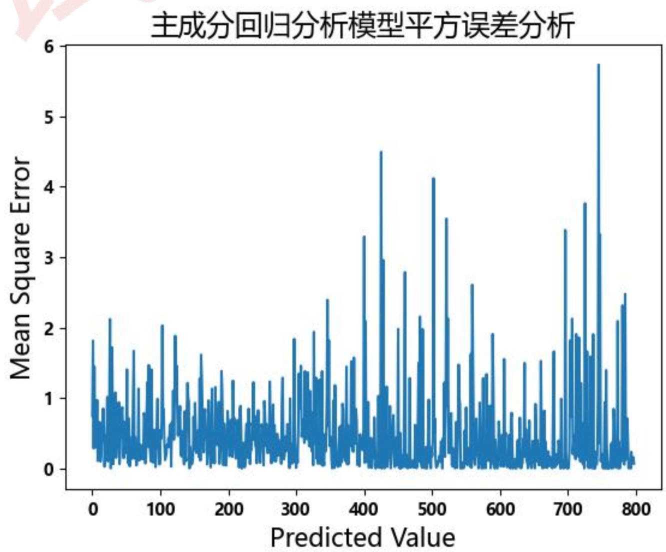
主成分多元分析模型绝对误差分析结果可视化：



使用 Python 编写代码对主成分多元回归分析模型进行平方误差分析，具体代码如下图所示：

```
for i in range(len(y_test)):
    err = abs(y_test[i]-y_pred[i])**2
    errors.append(err)
plt.plot(errors)
plt.xlabel("Predicted Value",fontsize=15)
plt.ylabel("Mean Square Error",fontsize=15)
plt.title("主成分回归分析模型平方误差分析",fontsize=17)
plt.show()
```


主成分多元回归分析模型平方误差分析结果可视化：

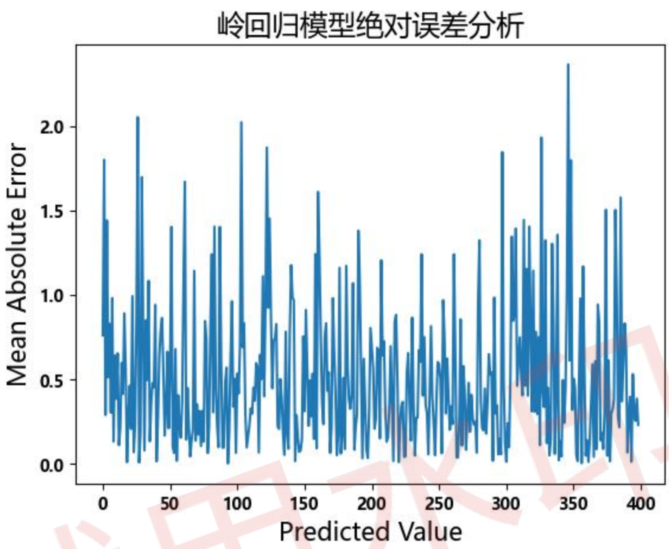


（2）岭回归模型

使用 Python 编写代码对岭回归模型进行绝对误差分析，具体代码如下图所示：

```python
errors = list()
for i in range(len(y_test)):
    err = abs(y_test[i]-y_pred1[i])
    errors.append(err)
plt.plot(errors)
plt.xlabel("Predicted Value",fontsize=15)
plt.ylabel("Mean Absolute Error",fontsize=15)
plt.title("岭回归模型绝对误差分析",fontsize=17)
plt.show()
```

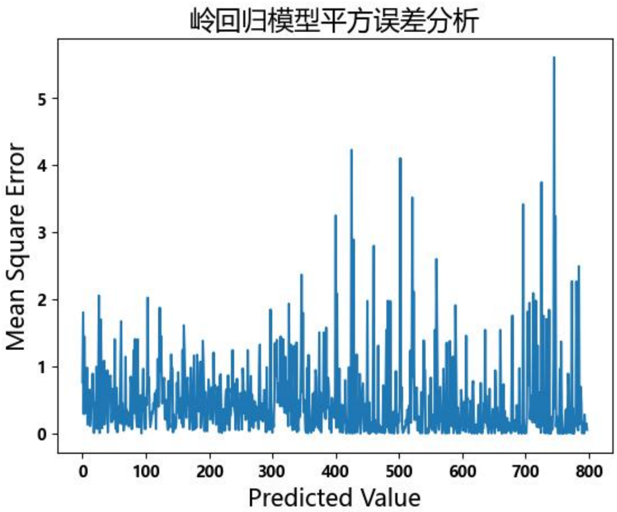
岭回归模型绝对误差分析结果可视化：



使用 Python 编写代码对岭回归模型进行平方误差分析，具体代码如下图所示：

```python
for i in range(len(y_test)):
    err = abs(y_test[i]-y_pred1[i])**2
    errors.append(err)
plt.plot(errors)
plt.xlabel("Predicted Value",fontsize=15)
plt.ylabel("Mean Square Error",fontsize=15)
plt.title("岭回归模型平方误差分析",fontsize=17)
plt.show()
```

岭回归模型进行平方误差分析可视化结果如下图所示：

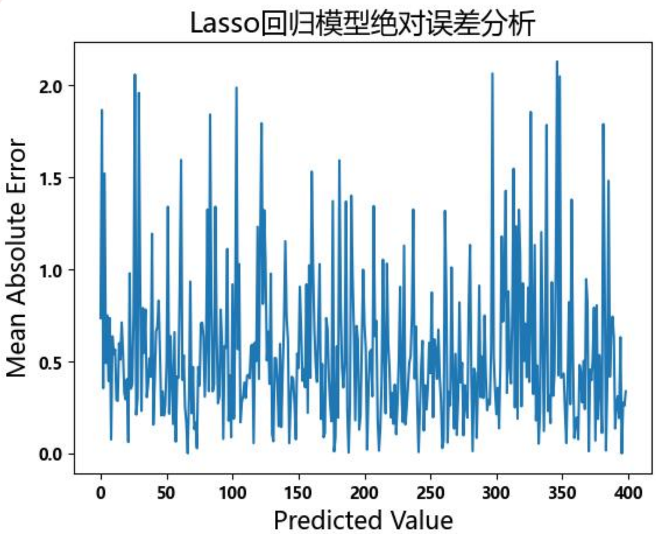


（3）Lasso 回归模型

使用 Python 编写代码对 Lasso 回归模型进行绝对误差分析，具体代码如下图所示：

```
errors = list()
for i in range(len(y_test)):
    err = abs(y_test[i]-y_pred2[i])
    errors.append(err)
plt.plot(errors)
plt.xlabel("Predicted Value",fontsize=15)
plt.ylabel("Mean Absolute Error",fontsize=15)
plt.title("Lasso回归模型绝对误差分析",fontsize=17)
plt.show()
```

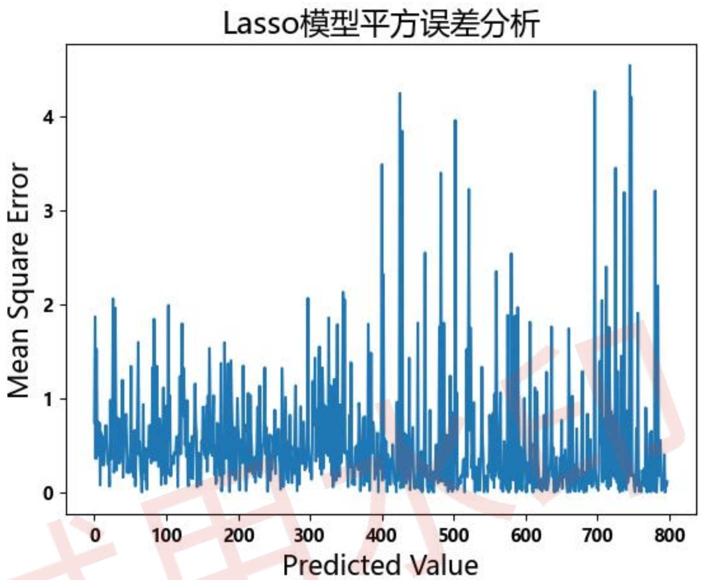

Lasso 回归模型进行绝对误差分析可视化：



使用 Python 编写代码对 Lasso 回归模型进行平方误差分析，具体代码如下图所示：

```
for i in range(len(y_test)):
    err = abs(y_test[i]-y_pred2[i])**2
    errors.append(err)
plt.plot(errors)
plt.xlabel("Predicted Value",fontsize=15)
plt.ylabel("Mean Square Error",fontsize=15)
plt.title("Lasso模型平方误差分析",fontsize=17)
plt.show()
```

贝叶斯岭回归模型进行平方误差分析可视化：

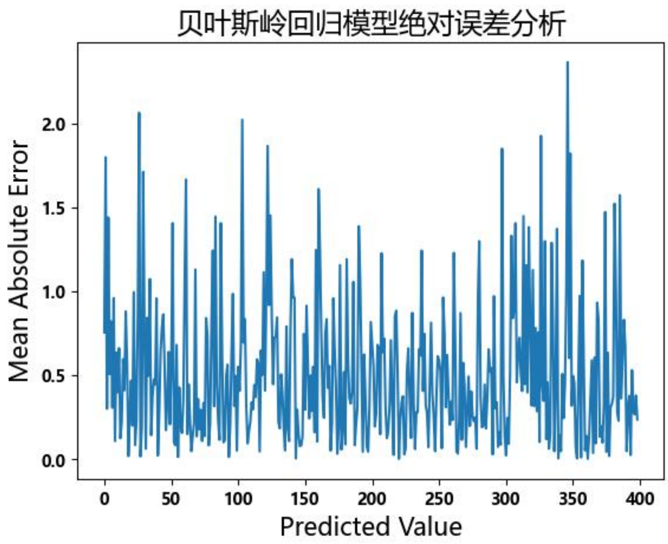


（4）贝叶斯岭回归模型

使用 Python 编写代码对贝叶斯岭回归模型进行绝对误差分析，具体代码如下图所示：

```
errors = list()
for i in range(len(y_test)):
    err = abs(y_test[i]-y_pred3[i])
    errors.append(err)
plt.plot(errors)
plt.xlabel("Predicted Value",fontsize=15)
plt.ylabel("Mean Absolute Error",fontsize=15)
plt.title("贝叶斯岭回归模型绝对误差分析",fontsize=17)
plt.show()
```

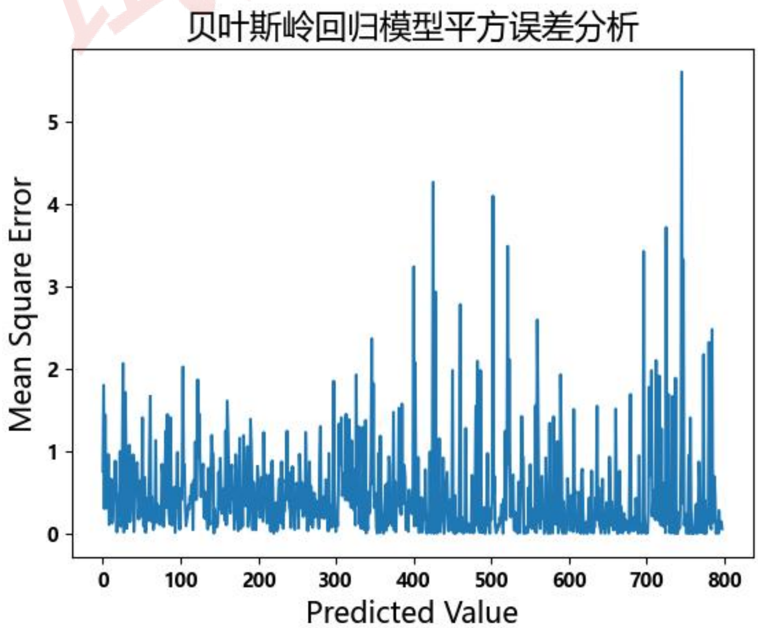
贝叶斯岭回归模型进行绝对误差分析可视化：



使用 Python 编写代码对贝叶斯岭回归模型进行平方误差分析，具体代码如下图所示：

```
for i in range(len(y_test)):
    err = abs(y_test[i]-y_pred3[i])**2
    errors.append(err)
plt.plot(errors)
plt.xlabel("Predicted Value",fontsize=15)
plt.ylabel("Mean Square Error",fontsize=15)
plt.title("贝叶斯岭回归模型平方误差分析",fontsize=17)
plt.show()
```

贝叶斯岭回归模型进行平方误差分析可视化：



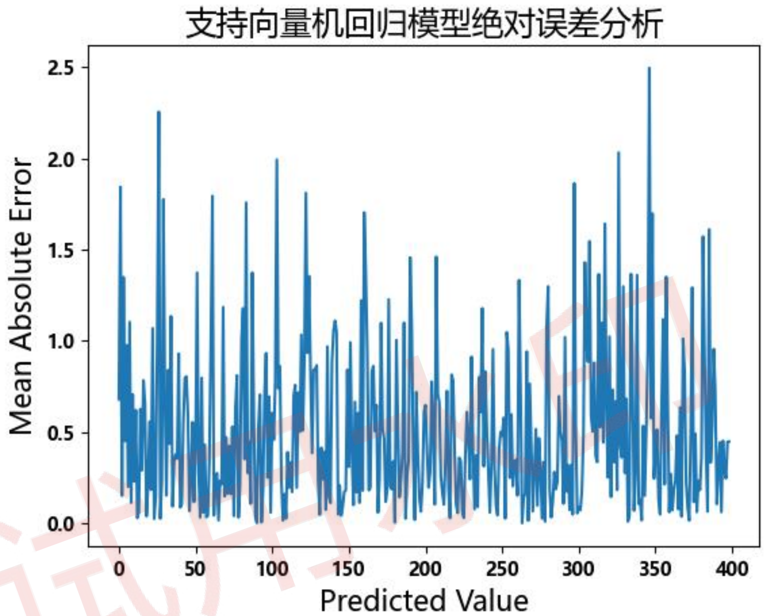
（5）支持向量机模型

使用 Python 编写代码对支持向量机模型进行绝对误差分析，具体代码如下

图所示：

```python
errors = list()
for i in range(len(y_test)):
    err = abs(y_test[i]-y_pred5[i])
    errors.append(err)
plt.plot(errors)
plt.xlabel("Predicted Value",fontsize=15)
plt.ylabel("Mean Absolute Error",fontsize=15)
plt.title("支持向量机回归模型绝对误差分析",fontsize=17)
plt.show()
```

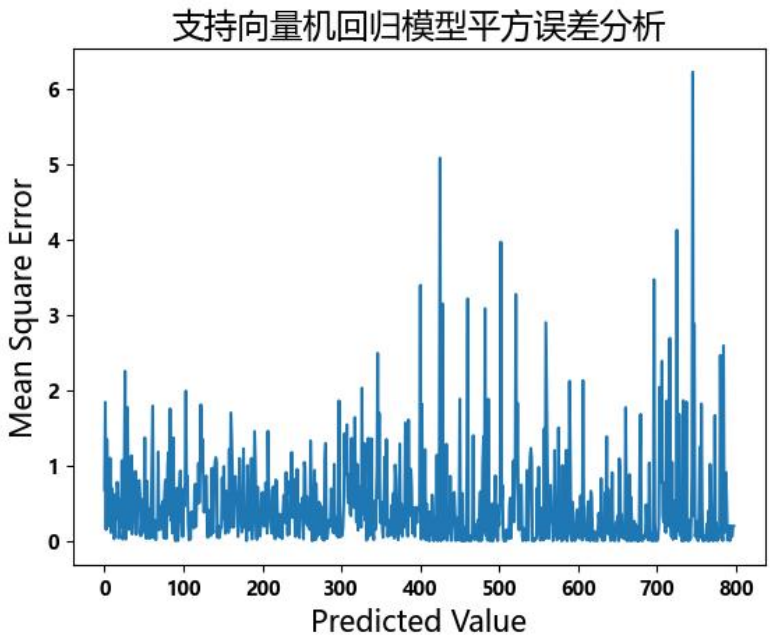
对支持向量机模型进行绝对误差分析可视化：



使用 Python 编写代码对支持向量机模型进行平方误差分析，具体代码如下图所示：

```python
for i in range(len(y_test)):
    err = abs(y_test[i]-y_pred5[i])**2
    errors.append(err)
plt.plot(errors)
plt.xlabel("Predicted Value",fontsize=15)
plt.ylabel("Mean Square Error",fontsize=15)
plt.title("支持向量机回归模型平方误差分析",fontsize=17)
plt.show()
```

对支持向量机模型进行平方误差分析可视化：



4.5.4 报告回归结果

使用 Python 编写代码对主成分回归分析(Principal Component Regression，PCR)，岭回归 (Ridge Regression)，贝叶斯岭回归 (Bayesian Ridge Regression)，Lasso 回归，支持向量机（Support Vector machine Regression，SVR）五种回归模型的回归情况进行探究，具体代码如下：

```
models = [("PCR",lm_RMSE,lm_R2,lm_adjusted_R2,lmCV_R2),
          ("Ridge Regression",ridge_RMSE,ridge_R2,ridge_adjusted_R2,ridgeCV_R2),
          ("Lasso",la_RMSE,la_R2,la_adjusted_R2,ridgeCV_R2),
          ("Bayesian",BayesianRidge_RMSE,BayesianRidge_R2,BayesianRidge_adjusted_R2,BayesianRidgeCV_R2),
          ("SVR",SVR_RMSE,SVR_R2,SVR_adjusted_R2,SVRCV_R2)
         ]
predictions = pd.DataFrame(data=models,columns=["Model","RMSE","R2","adjust_R2","CV_R2"])
print(predictions)
```

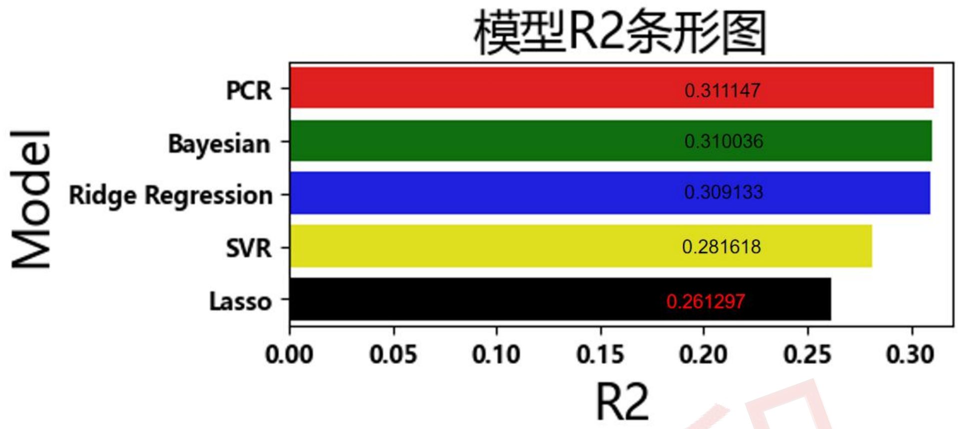
研究结果显示如下：

	Model	RMSE	R2	adjust_R2	CV_R2
0	PCR	0.669795	0.311147	0.293394	0.339534
1	Ridge Regression	0.670774	0.309133	0.289496	0.339058
2	Lasso	0.693607	0.261297	0.240301	0.339058
3	Bayesian	0.670335	0.310036	0.290425	0.340279
4	SVR	0.684001	0.281618	0.263103	0.325221

使用 Python 编写代码对研究结果进行可视化，具体代码如下图所示:

```python
f,axe = plt.subplots(1,1,figsize=(5,2))
predictions.sort_values(by="R2",ascending=False,inplace=True)
sns.barplot(x="R2",y="Model",data=predictions,ax=axe,palette=['red', 'green', 'blue', 'yellow',"black"])
axe.set_xlabel("R2",size=20)
axe.set_ylabel("Model",size=20)
plt.title("模型R2条形图",size=20)
axe.set_xlim(0,0.32)
plt.show()
```

可视化结果如下图所示：



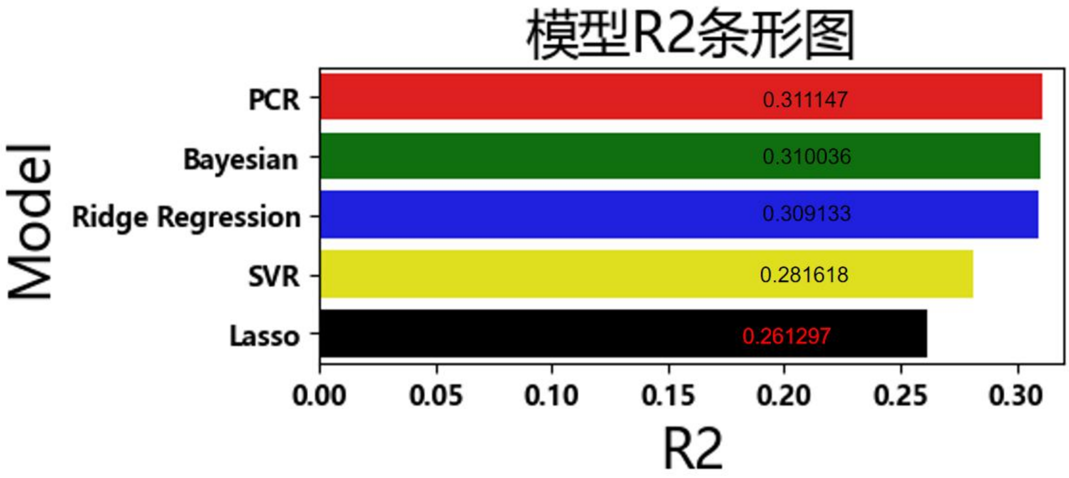
从中可以看出，主成分回归的拟合精度最高（R^2 为 0.3111），其次是贝叶斯岭回归（R^2 为 0.3100），岭回归的拟合效果也令人满意（R2 为 0.3091），支持向量机和 Lasso 回归的拟合效果一般。

4.6 数据可视化

4.6.1 产生并输出表格：二维/三维表格

	Model	RMSE	R2	adjust_R2	CV_R2
0	PCR	0.669795	0.311147	0.293394	0.339534
1	Ridge Regression	0.670774	0.309133	0.289496	0.339058
2	Lasso	0.693607	0.261297	0.240301	0.339058
3	Bayesian	0.670335	0.310036	0.290425	0.340279
4	SVR	0.684001	0.281618	0.263103	0.325221

4.6.2 产生并输出图形：柱状图，条形图等



4.7 新数据预测

首先将三个新的观测值形成 dataframe 结构，便于下步分析。并且由于各个解释变量即红葡萄酒的不同化学成分量纲明显不一致，为了减小葡萄酒质量评分的预测误差，使预测结构更符合实际，对解释变量即红葡萄酒的不同化学成分进行标准化处理。具体代码如下图所示：

```
# 利用三个新的观测值形成dataframe结构
new_data1 = [(7.9,0.43,0.21,1.6,0.106,10.0,37.0,0.9966,3.17,0.91,9.5,5),
         (7.1,0.71,0.0,1.9,0.08,14.0,35.0,0.9972,3.47,0.55,9.4,5),
         (7.8,0.645,0.0,2.0,0.082,8.0,16.0,0.9964,3.38,0.59,9.8,6),
        ]
new_data = pd.DataFrame(data=new_data1,columns=["X1","X2","X3","X4","X5","X6","X7","X8","X9","X10","X11","Y"])
print(new_data)
```

```
    X1     X2    X3   X4     X5    X6    X7      X8    X9   X10  X11  Y
0  7.9  0.430  0.21  1.6  0.106  10.0  37.0  0.9966  3.17  0.91  9.5  5
1  7.1  0.710  0.00  1.9  0.080  14.0  35.0  0.9972  3.47  0.55  9.4  5
2  7.8  0.645  0.00  2.0  0.082   8.0  16.0  0.9964  3.38  0.59  9.8  6
```

```
# 由于各个解释变量量纲明显不一致，对新生成的数据集进行标准化
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler  # 从预处理中导入标准化模块
import seaborn as sns
import numpy as np
b = new_data.iloc[:, :11].values
scaler = StandardScaler()   # 建立标准化模块类对象
X = scaler.fit_transform(b)   # 对原始数据b使用标准化模块进行fit_transform转换,此时X是一个二维数组形式。
y = new_data["Y"].values
print("标准化处理后的数据均值为: ", np.mean(X))
print("标准化处理后的数据标准差为: ", np.std(X))
```

```
标准化处理后的数据均值为:  2.028680254087786e-14
标准化处理后的数据标准差为:  1.0
```

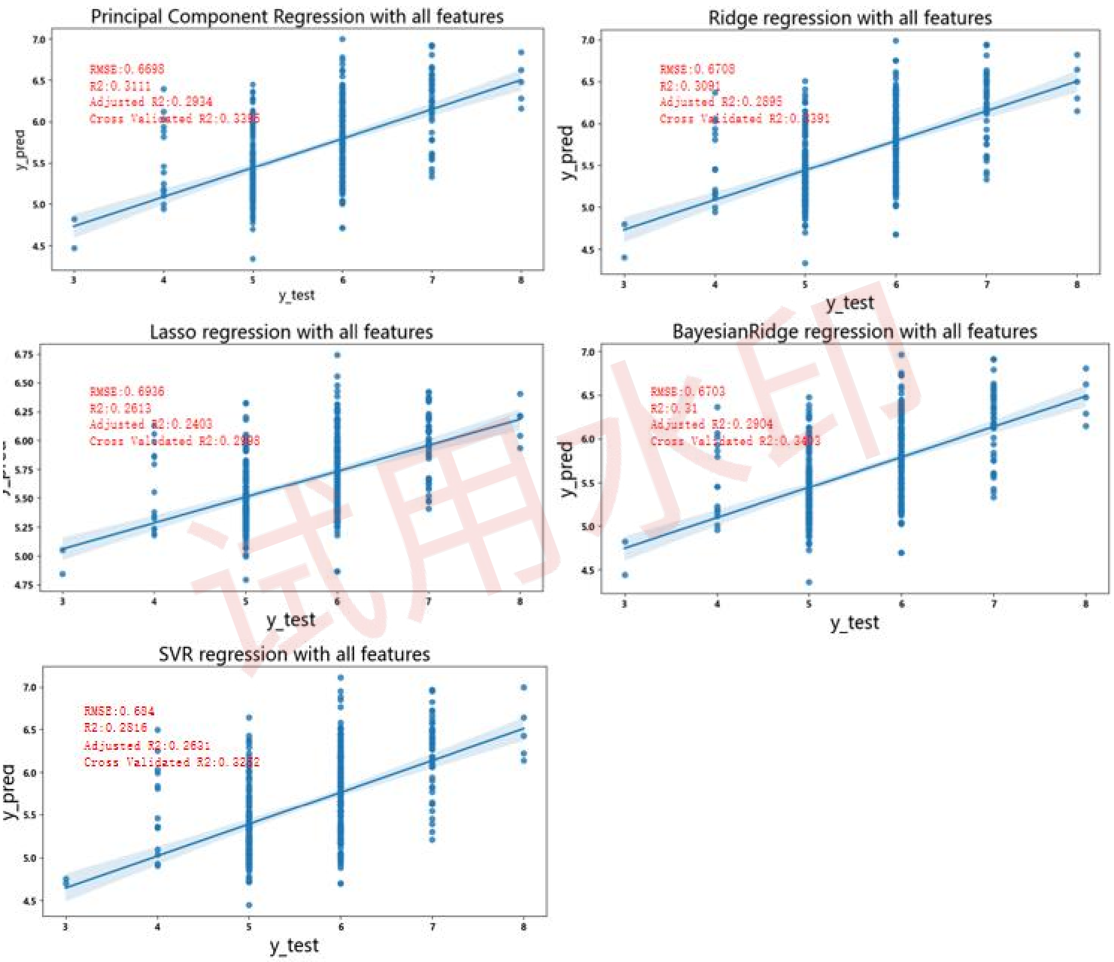
由于**待预测数据集只有三个样本，不能使用主成分回归模型拟合**，为此，**使用拟合效果次之的贝叶斯岭回归对红葡萄酒的质量评分进行预测**。具体代码和结果如下图所示：

```
new_predict = BayesianRidge_reg.predict(X1)
print(new_predict)
```

```
[5.87188286 4.94381121 6.12093963]
```

最终，根据新数据得到预测值即葡萄酒的质量评分为：5.87188286，4.94381121，6.12093963。

5 结果分析



上图显示了各模型的拟合效果，**PCR 回归模型的拟合效果最好**，其 R^2 为 0.311147，CV_R2 为 0.339534，RMSE 为 0.669795。 对比所有的回归模型，其 R^2 为最大，CV_R2 数值也可观，RMSE 为最小。在绘制的散点图中，拟合直线能较好地反映测试数据与预测值（即预测出的质量评分）之间的关系，且显示出较小的偏差，表明该模型具有较高的预测精度和较好的解释能力，更好的研究并预测红葡萄酒的不同化学成分对质量评分的影响。

贝叶斯岭回归模型的拟合效果也让人非常满意，其 R^2 为 0.310036，略低于 PCR 回归模型的 R^2 值，且 RMSE 为 0.670335，略大于 PCR 回归模型，拟合效果次之。

岭回归模型和支持向量机回归模型的拟合效果紧随贝叶斯岭回归模型，其 R^2 略低于主成分回归分析模型，其 RMSE 维持在 0.670774，0.684001 附近，相对居中取值，拟合效果一般。

Lasso 回归模型的拟合效果较差，其 R^2 保持在 0.261297 左右，其 RMSE 值最大。

6 研究结论

通过对数据进行初步分析，发现原始数据存在缺失值，利用统计学方法对其进行了数据清洗，随后分析了数据的分布状态，紧接着对数据进行了标准化处理，并划分了训练集和测试集。基于训练集，利用多种回归模型进行了红葡萄酒的不同化学成分对质量评分的拟合，并在测试集上进行了验证。结果显示，使用**主成分回归模型和贝叶斯岭回归模型能更好地描述红葡萄酒不同化学成分和红葡萄酒质量评分之间的关系**，这意味着这两种模型能较为准确地捕捉化学成分数据中的潜在模式，提供更优的预测性能和解释能力，特别是在处理存在多重共线性的情况下，比其他回归方法（如岭回归等）更具优势。**贝叶斯岭回归模型**的较高 CV_R2，R2 值和较低 RMSE 值表明，它不仅在拟合训练数据时表现优异，同时也能在测试数据上保持较好的泛化能力，**适合用于实际应用中的预测任务**。