

武汉理工大学

Python 数据分析与可视化
大 作 业

题 目	学生考试成绩数据分析与可 视化
学 院	计算机与人工智能学院
专 业	软件卓越工程师
班 级	软件 zy2101
姓 名	周企安

2023 年 5 月 26 日

一、项目概述

项目背景

在教学任务中，学生的成绩往往是分析学生学习情况的一个重要参考。而学生的成绩通常是由多方面因素的共同影响。本项目主要从学生的家庭信息、学习信息等多个方面分析对学生信息的影响因素。它可以帮助教师和学生更好地了解学习效果，发现问题，制定改进策略。成绩数据可视化分析的背景介绍主要包括以下几个方面：数据来源，数据处理，数据展示，数据解读等。

数据介绍

数据名称: Students Exam Scores: Extended Dataset

数据地址: [Students Exam Scores: Extended Dataset | Kaggle](#)

数据结构:

- Unnamed: 0 -> ID
- Gender -> 性别
- EthnicGroup -> 学生所属种族群体
- ParentalEdu -> 父母教育程度
- LunchType -> 午餐类型
- TestPrep -> 考试准备情况
- ParentMaritalStatus -> 父母婚姻状况
- PracticeSport -> 是否参加体育活动
- IsFirstChild -> 是否是家中第一个孩子
- NrSiblings -> 兄弟姐妹数量
- TransportMeans -> 交通方式
- WklyStudyHours -> 每周学习时间
- MathScore -> 数学成绩
- ReadingScore -> 阅读成绩
- WritingScore -> 写作成绩

程序功能

1. 单项数据分析
2. 多项数据相关分析
3. 数据影响因素分析

4. 预测模型

所使用的第三方库

数据读取和处理：pandas

数据可视化：matplotlib, seaborn

神经网络模型：pytorch

其他：tqdm

二、功能实现

数据分析与预处理

首先观察属性列可知，该数据可划分为影响因素（性别、学生所属种族群体、父母教育程度、午餐类型、考试准备情况、父母婚姻状况、是否参加体育活动、是否是家中第一个孩子、兄弟姐妹数量、交通方式、每周学习时间）和成绩因素（数学成绩、阅读成绩、写作成绩）两部分。

在对数据进行分析前，应先对数据空缺处进行填充，首先通过信息获取该数据的缺失情况，如下：

```
原始缺失值数量：
Unnamed: 0          0
Gender              0
EthnicGroup        1840
ParentEduc         1845
LunchType           0
TestPrep           1830
ParentMaritalStatus 1190
PracticeSport       631
IsFirstChild        904
NrSiblings          1572
TransportMeans       3134
WklyStudyHours      955
MathScore           0
ReadingScore         0
WritingScore         0
dtype: int64
```

由于该数据集由两部分组成：Original_data_with_more_rows（原始的数据集，包含较少的属性）和 Expanded_data_with_more_features（扩充属性的数据集，即本项目所使用的主要数据集），因此我们对于空缺值得填充应分为两种情况：

1. 原始数据表中有的，将其填充到扩充数据表中

```
df_original = pd.read_csv(os.path.join('data', 'Original_data_with_more_rows.csv'))
for column in columns[1:]:
    #扩充表中有些行是原始表中没有的，所以需要先判断一下是否存在
    if column in df_original.columns:
        df[column].fillna(df_original[column], inplace=True)
```

2. 原始数据表中没有的：

数值型数据：填充为该类型的平均值（特别的，如果是成绩的数据缺失，则使用他/她自己的平均成绩进行填充，当然事实上没有成绩缺失）

类别型数据：填充为该类型的众数所对应的值

```
df['NrSiblings'].fillna(int(df['NrSiblings'].mean()), inplace=True)
fillings = ('ParentMaritalStatus', 'PracticeSport', 'IsFirstChild', 'TransportMeans',
            'WklyStudyHours')
for filling in fillings:
    df[filling].fillna(df[filling].mode()[0], inplace=True)
print(f'缺失值数量: \n{df.isnull().sum()}')
```

在我们完成填充 1 后结果如下：

```
缺失值数量：
ID                0
Gender            0
EthnicGroup       0
ParentEduc        0
LunchType         0
TestPrep          0
ParentMaritalStatus 1190
PracticeSport     631
IsFirstChild      904
NrSiblings        1572
TransportMeans    3134
WklyStudyHours    955
MathScore         0
ReadingScore      0
WritingScore      0
dtype: int64
```

此后进行第二项填充，经过填充后得结果如图：

```
缺失值数量：
ID                0
Gender            0
EthnicGroup       0
ParentEduc        0
LunchType         0
TestPrep          0
ParentMaritalStatus 0
PracticeSport     0
IsFirstChild      0
NrSiblings        0
TransportMeans    0
WklyStudyHours    0
MathScore         0
ReadingScore      0
WritingScore      0
dtype: int64
```

图形绘制

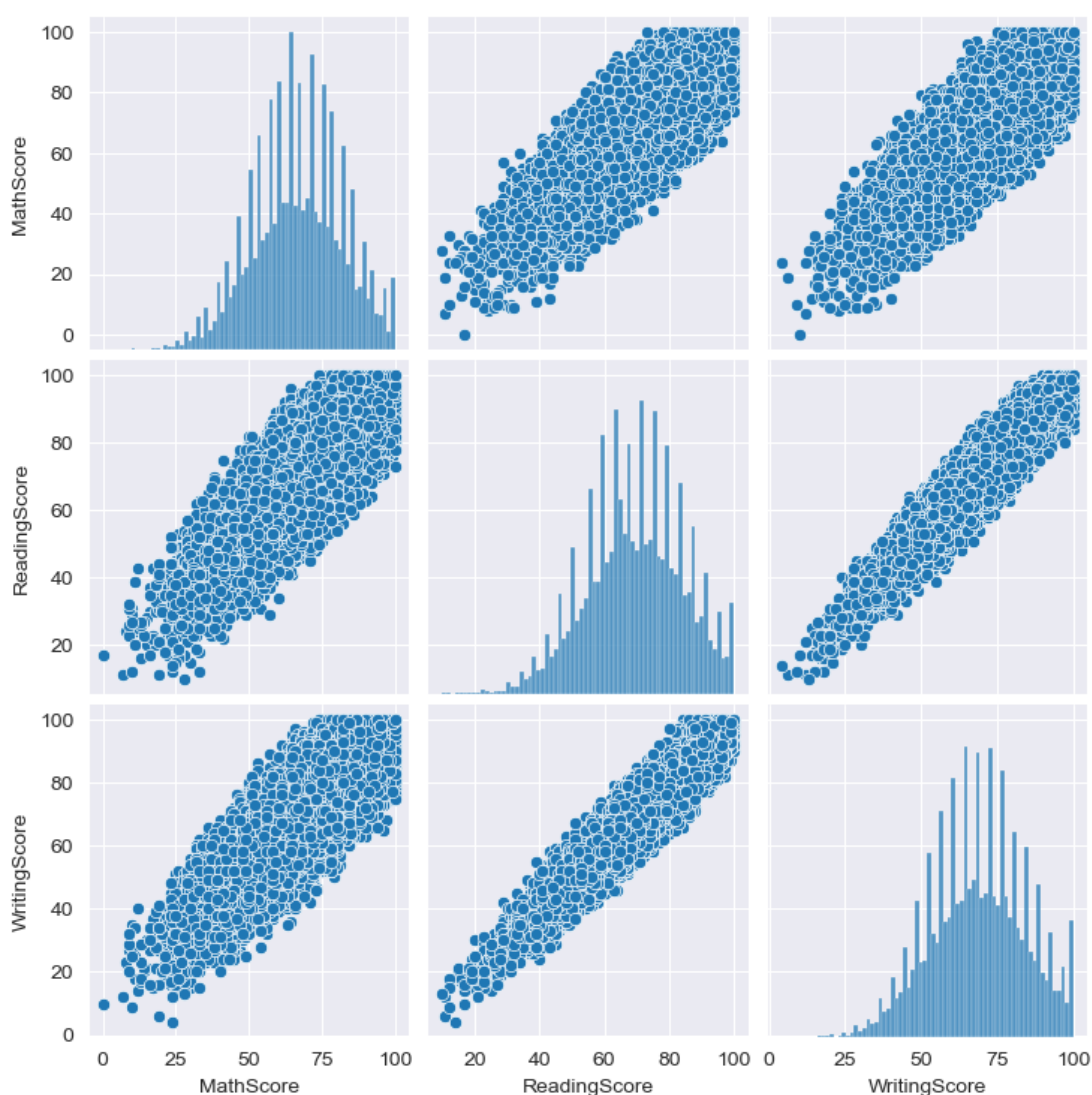
在经过了以上数据的处理后，我们终于可以开始分析数据。总的来说，数据分析总共有大概三个方面：

1. 成绩因素和成绩因素之间的关系分析
2. 影响因素和成绩因素之间的关系分析
3. 影响因素与影响因素之间的关系分析

成绩因素和成绩因素之间的关系分析

尽管直觉上，我们可以认为数学成绩、阅读成绩和写作成绩之间存在某种关系，但是我们需要通过可视化来证明这一点。

这里我们采用散点图的方式，绘制三门成绩两两之间的关系图，得到的结果如下：



由上述图片我们可以得到如下的几条结论：

1. 总的来说，学生三门成绩之间是正相关的，即当一名学生一项成绩很优秀的时候，那么可以确定他的另外两门成绩也更加优秀。
2. 尽管三门成绩之间有着强烈的正相关性，但他们之间依然有所差别。

可以看出，阅读成绩和写作成绩的关联性更强。而数学成绩和阅读成绩或者写作成绩虽然也有着正相关性，但是我们仔细观察图片可以发现，数学成绩对阅读/写作的影响要稍微“弱”一些。

以上的两条结论十分的符合我们的常规认知，因此我们可以按照这个思路继续进行下面的分析。

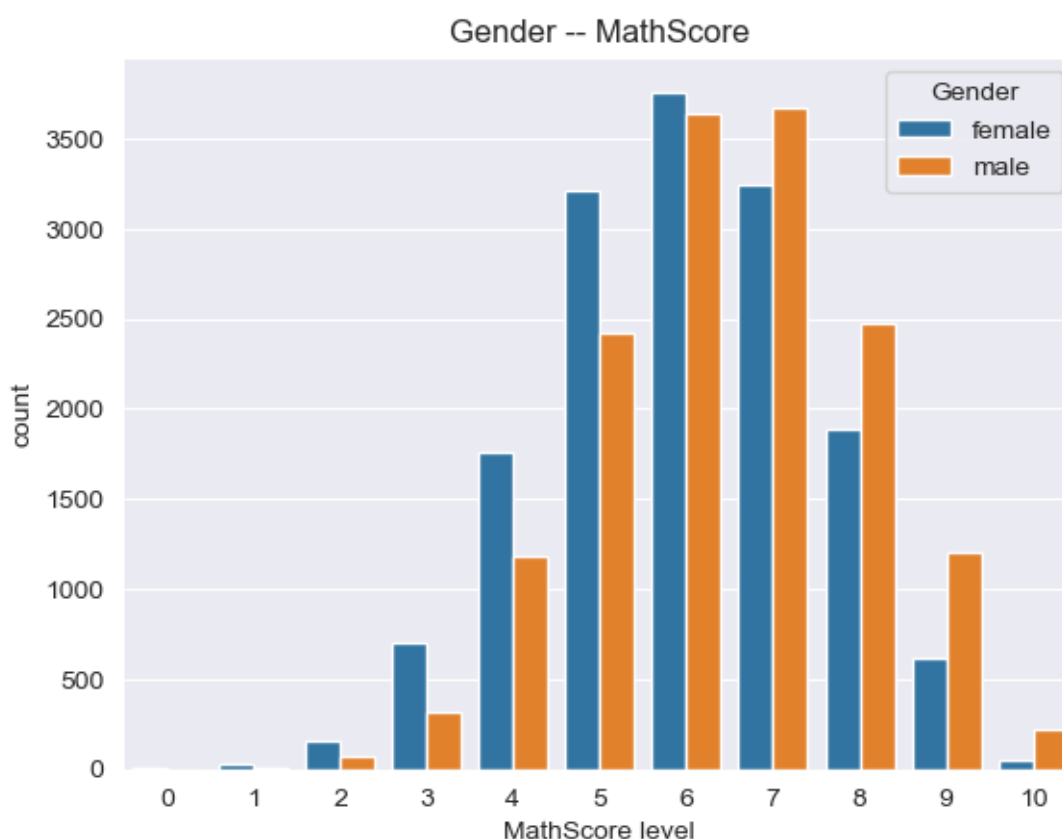
影响因素和成绩因素之间的关系分析

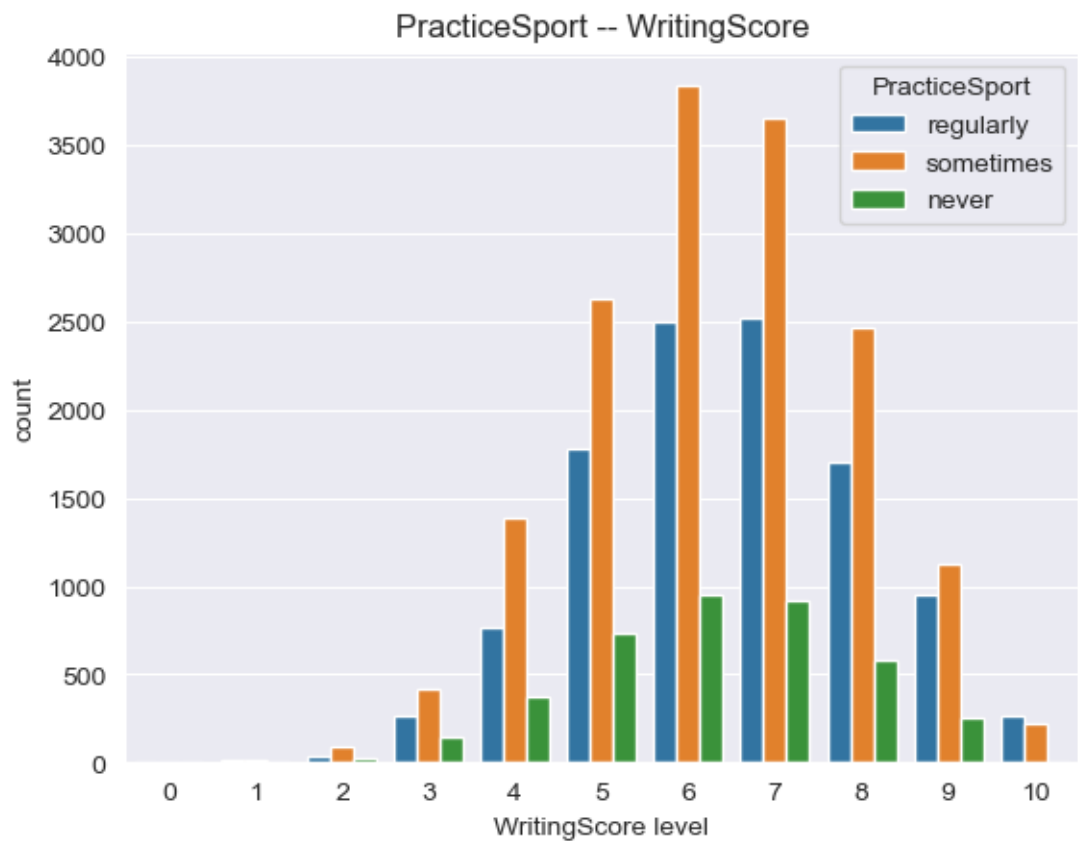
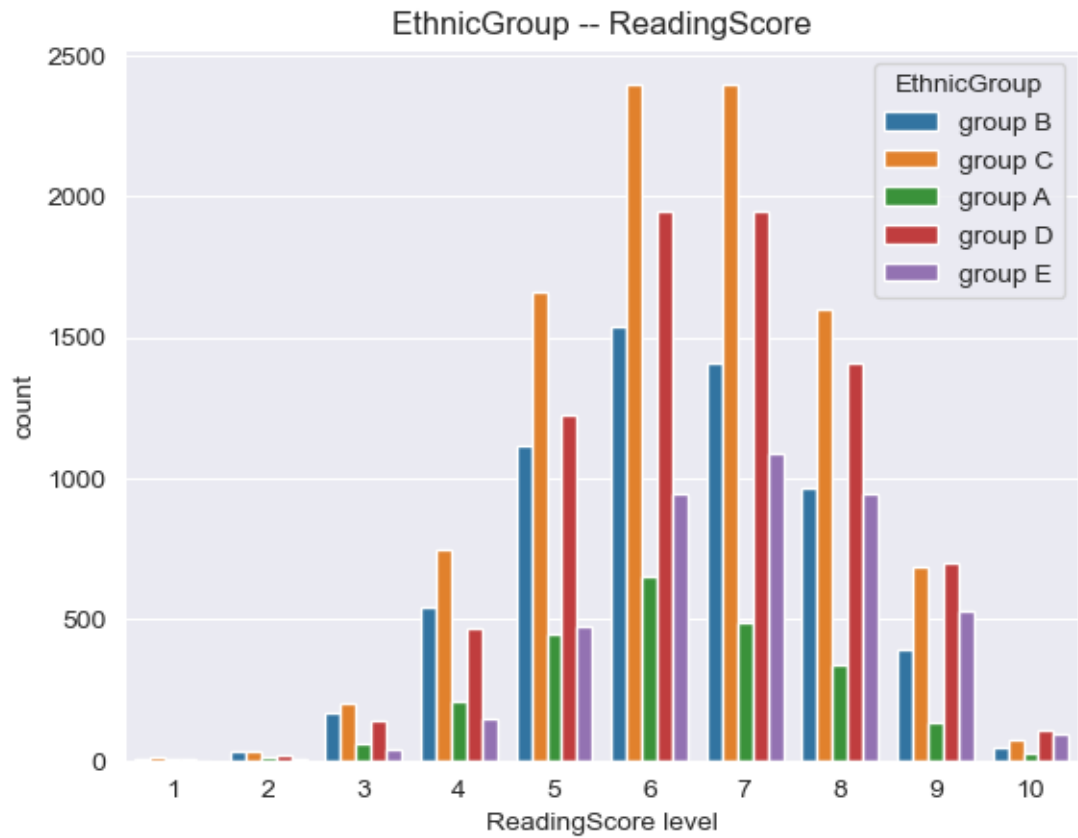
很明显，成绩之间的相关性只是影响他们成绩本身的一部分，对于个人的成绩，有着更多的影响因素（性别、是否复习等等多种情况）。

在正式进行可视化分析前，我们要对成绩本身进行的小小的改动：我们将成绩划分为 10 个等级，每个等级对应 10 分，之后通过频数直方图来展现各个数据之间的关联。

```
score = columns[12:]
for s in score:
    df[s + ' level'] = df[s].astype(int)
    df[s + ' level'] = df[s + ' level'] // 10
```

之后我们便可以绘制直方图，以下是**部分**直方图的示例：



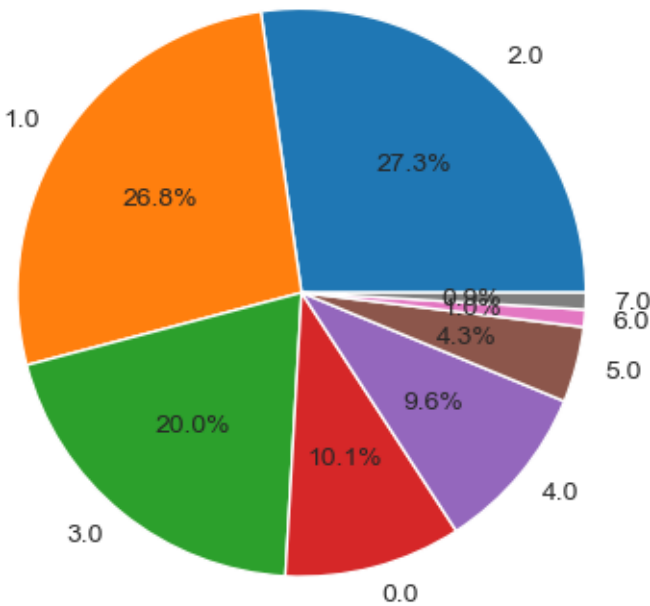


用同样的方法，我们可以得到每个因素和成绩之间的关系，并进行逐一分析，这里不再赘述。

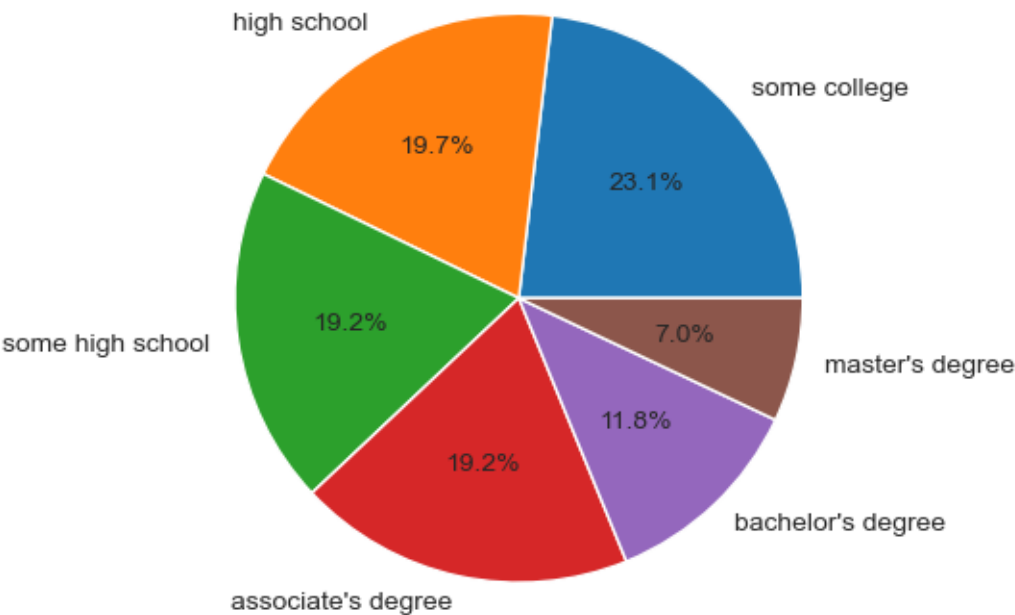
影响因素与影响因素之间的关系分析

毫无疑问，这些影响因素之间也有相关性，我们也应该用如上的方式进行分析

Gender -- NrSiblings



EthnicGroup -- ParentEduc



模型预测

在进行了相关的分析之后，我们可以对成绩进行分析和预测了。在本项目中，我们使用 Pytorch 构建神经网络对学生的成绩进行预测。

数据处理

尽管我们对数据已经进行了处理，但我们仍然需要再进行一些细微的优化来使其可以进行模型训练。

```
dict_list = list()
for column in columns[1:12]:
    diction = dict()
    for i, value in enumerate(df[column].unique()):
        diction[value] = i
    dict_list.append(diction)
for i, column in enumerate(columns[1:12]):
    df[column] = df[column].map(dict_list[i])
train_df = df.sample(frac=0.8, random_state=0)
test_df = df.drop(train_df.index)
train_x = train_df[columns[1:12]].values
train_y = train_df[columns[12:]].values
test_x = test_df[columns[1:12]].values
test_y = test_df[columns[12:]].values
train_x = torch.from_numpy(train_x).float()
train_y = torch.from_numpy(train_y).float()
test_x = torch.from_numpy(test_x).float()
test_y = torch.from_numpy(test_y).float()
```

如上所示，我们对数据进行的修正有：

除去 ID 信息

将类别型数据转化成数字，方便进行整体的修正

将数据划分为训练集和测试集合

将数据集分成 x 和 y 两部分

网络构建

在对数据进行了调整后，我们就可以进行网络的构建了，具体的网络构建如下：

```
class MyNet(nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()
        self.fc1 = nn.Linear(11, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 3)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        return x
```

网络层如下：

1. 全连接层
2. Relu 激活函数层
3. Dropout 层
4. 全连接层
5. Relu 激活函数层
6. Dropout 层
7. 全连接层

使用这样一个网络层的优点在于使用多层感知机使得网络变成了非线性层，同时使用暂退法能有效地避免过拟合问题的发生

优化器、损失函数和超参数

损失函数我们使用 MSE 损失函数：

```
loss_f = nn.MSELoss()
```

优化器我们使用 Adam 优化器，步长为 0.001：


```
optimizer = optim.Adam(net.parameters(), lr=0.001)
```

超参数的设置上：epochs = 1000, lr=0.001

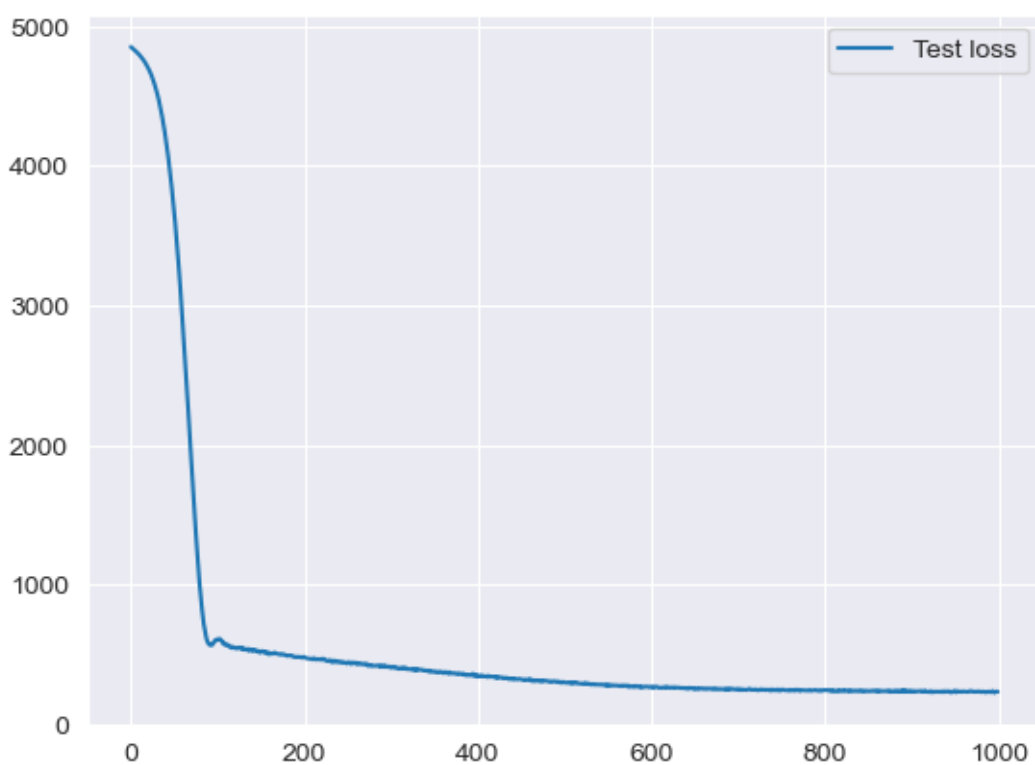
在这之后，我们便可以开始我们的训练了：

训练代码如下：

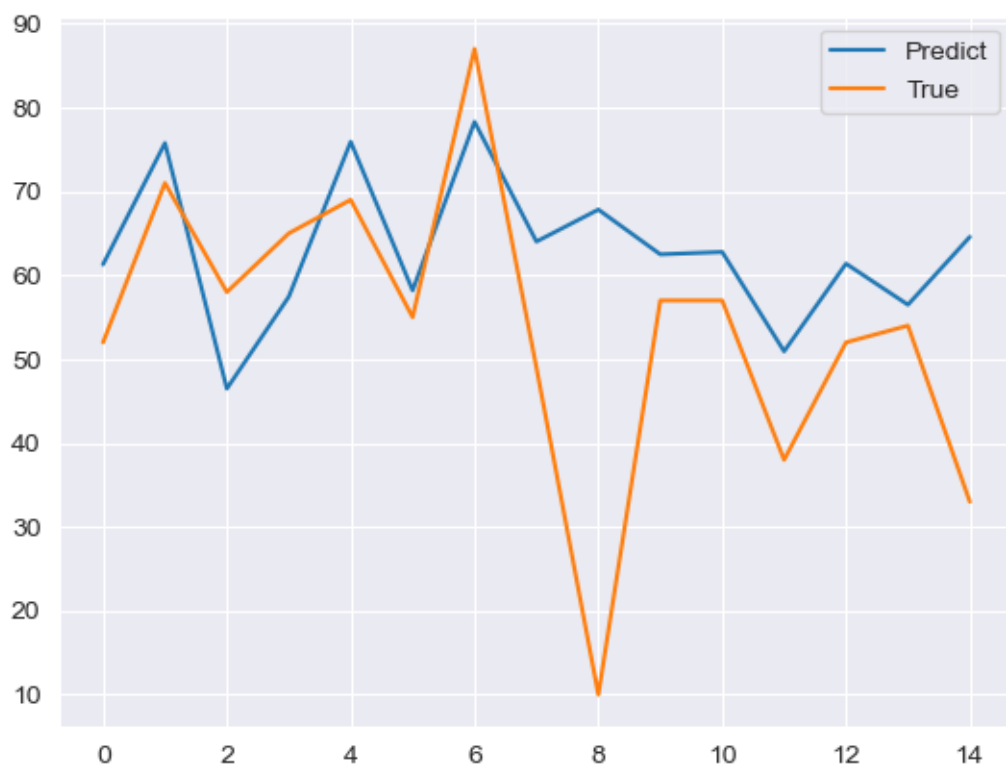
```
epochs = 1000
train_losses = []
test_losses = []
for epoch in tqdm(range(epochs)):
    optimizer.zero_grad()
    train_output = net(train_x)
    train_loss = loss_f(train_output, train_y)
    train_losses.append(train_loss)
    test_output = net(test_x)
    test_loss = loss_f(test_output, test_y)
    test_losses.append(test_loss)
    train_loss.backward()
    optimizer.step()
```

25%  244/1000 [00:11<00:30, 25.02it/s]

损失函数如下



之后我们进行预测并进行可视化：



可以看到，尽管在某些方面预测值出现了一些偏差，但总体上的预测是比较接近的