第七章超越线性

本章主要讨论线性模型的非线性扩展，最后证明这些扩展可以集成到一起

7.1 多项式回归

多项式回归 (polynomial regression) 是一种线性回归的非线性扩展，形式如下：

$$y_i = \beta_0 + \sum_{k=1}^{d} \beta_k x_i^k + \epsilon_i$$

其中 ϵ_i 是误差项。对于一个足够大的 d，我们能得到一个相当非线性的曲线。系数可用最小二乘回归拟合，即把 x_i^k 当做一个变量

> 一般来说，$d \le 4$，过于大的 d 会导致曲线变得很奇怪 (过拟合)

使用预测点 $\hat{f}(x_0)$ 的标准差，我们可以画出预测曲线的 95% 置信区间，他大约在 $2 \pm \sigma$ 上。$Var[\hat{f}(x_0)]$ 满足：

$$Var[\hat{f}(x_0)] = l_0^T \hat{C} l_0$$

其中 $l_0^T = (1, x_0, x_0^2, \cdots)$，$\hat{C}$ 是回归系数 $\hat{\beta}$ 的协方差矩阵，满足：

$$\hat{C} = Var(\hat{\beta}) = \sigma^2 (X^T X)^{-1}$$

其中 σ^2 是误差项的方程，用残差方差估计为 $\hat{\sigma}^2 = \frac{RSS}{n-p-1}$，$X$ 表示设计矩阵

根据某一预测值的方差计算公式，我们发现该处的方差不仅与该点的特征向量有关，还和该点的系数协方差矩阵有关。这可以解释距离数据区域很远的点的极大方差

7.2 阶跃函数

现在除去线性回归的全局线性假设，我们尝试通过用常数和离散化的矩阵 X 来对模型进行预测

定义**阶跃函数 (Step Functions)** 的指示函数为：

$$\begin{aligned}
C_0(X) &= I(X < c_1), \\
C_1(X) &= I(c_1 \le X < c_2), \\
C_2(X) &= I(c_2 \le X < c_3), \\
&\vdots \\
C_{K-1}(X) &= I(c_{K-1} \le X < c_K), \\
C_K(X) &= I(c_K \le K)
\end{aligned}$$

$I(\cdot)$ 表示当条件满足时值为 1，否则为 0。因此这些函数满足 $\sum_{i=0}^{K} C_i(X) = 1$，将函数划分为了 $K+1$ 段，并在每段中用一个常数 $\beta_0 + \beta_i$ 来预测对应的值。用阶跃函数进行回归的表达式：

$$y_i = \beta_0 + \sum_{i=1}^{K} C_i(x_i) + \epsilon_i$$

在这里，我们可将 β_0 解释为预测值的均值，β_i 表示在 X 满足某条件下相对于均值的平均涨幅

像类似年龄这样存在自然断点的函数可以用阶跃函数回归来拟合。然而，其他的函数则不适用，因为它不够平滑，会忽略断点之间的过渡关系

7.3 基函数

基函数 (Basis Funcions) 是对上面两种方法的统称。我们注意到这两种方法都是需找到一类函数 (多项式函数或分段常数函数) 来拟合数据，可以写成如下的同一函数：

$$y_i = \beta_0 + \sum_{i=1}^{K} \beta_i b_i(x_i) + \epsilon_i$$

除了多项式函数和分段常数函数，我们还可以构造其他函数来进行拟合，如微波和傅里叶展开

7.4 回归样条

7.4.1 分段多项式回归

不同于考虑拟合一个全局的多项式回归函数，我们将特征矩阵 X 划分为 K 个不同的部分，并在每个部分用一个次数较低的多项式函数进行拟合，其中每段的多项式函数同 $y_i = \beta_0 + \sum_{k=1}^{d} \beta_k x_i^k + \epsilon_i$，在不同段中，$\beta_{ij}$ 的估计值不同。这种估计方法叫做**分段多项式回归 (piecewise polynomial regression)**，分段处叫做**结点 (knot)**

更多的结点会导致更灵活的分段多项式，对于 K 个结点，我们有 $K+1$ 个多项式。分段常数函数是 0 次分段多项式。拟合分段多项式模型需要消耗对应的**自由度 (degrees of freedom)**

7.4.2 约束和样条

直接进行分段多项式回归得到的样条会出现间断点，导致解释上的缺失，我们通过添加约束条件来改进：

- 连续的结点：要求结点两端的函数值相等
- 平滑过渡：要求结点两端的函数的**导数 (derivative)**连续，可延伸到多阶导数。设定连续型的阶数为 n 的曲线称为 **n 次样条曲线** 。带有 K 个结点的三次样条曲线消耗 $4+K$ 个自由度

7.4.3 样条的基表示

一个三次样条曲线可以表示为：

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i$$

共有 $K+4$ 个自由度。为了表示这个函数，我们将它分为两个部分：

- 描述整体立方关系 x, x^2, x^3
- 对应段的描述 **截断幂基函数 (truncated power basis)**

$$h(x,\xi)=(x-xi)_+^3=\begin{cases}(x-xi)^3, & x>\xi\\ 0, & otherwise\end{cases}$$

  其中 ξ 是结点。对于三次多项式，加入 $h(x,\xi)$ 将会保证二阶之前的倒数连续，但是三阶导数不一定连续

  > 随着 K 的增加，将会添加更多的 $h(x,\xi)$ 项；每经过一个结点就会加上一次 $h(x,\xi)$

普通的三阶样条曲线在边界上的置信区间非常宽泛，我们用**自然样条 (natural spline)** 来规避这个问题。自然样条要求边界处的预测曲线是直线，以降低预测的方差

7.4.4 选取结点数量和位置

我们可以通过观察数据分布来选择结点数量和位置，例如，在数据变动大的地方放置结点，以提高灵活性；数据变动少的地方则不放置结点

实践中，我们通常指定模型需要的自由度，然后利用软件在数据的某些分位点放置结点

选择最优的结点划分数量，可以通过交叉检验计算模型的均方误差来确定

提到的方案：

- 观察散点图
- 指定自由度，自动选择分位点
- 交叉检验

7.4.5 与多项式回归的比较

结论：样条在边缘区域的拟合效果比过于灵活的多项式回归更好

7.5 平滑样条

7.5.1 平滑样条简介

最小二乘回归的目的是找到一条合适的曲线使得：

$$RSS=\sum_{i=1}^{n}(y_i-g(x_i))^2$$

潜在的问题是可能找到一条曲线过于灵活，导致 $RSS=0$

不同于单纯考虑 RSS，我们希望 $g(x_i)$ 能最小化：

$$\sum_{i=1}^{n}(y_i-g(x_i))^2+\lambda\int g''(t)^2dt$$

其中的 λ 为非负条件参数，满足上式的 $g(x_i)$ 称为**光滑样条 (smoothing spline)**，$RSS = \sum_{i=1}^{n}(y_i - g(x_i))^2$ 称为**损失函数 (loss function)**

$\lambda\int g''(t)^2dt$ 是一个惩罚项。它衡量了曲线变动的幅度。若曲线变动幅度大，则此项的值会变大；相对的，若曲线变动幅度很小，则此项的值也会很小。因此，$\lambda\int g''(t)^2dt$ 鼓励更加平滑的曲线。$\lambda = 0$ 时，曲线波动异常，容易过拟合；$\lambda \to \infty$ 时，曲线变为直线

平滑样条的最优解是一个自然三次样条，它的特点是：

- 由三次多项式拼接二次
- 结点自动放在 $\forall x_i$ 处
- 边界区域的曲线是线性函数

但是它又是一个“收缩版本”，λ 控制了曲线的收缩程度

7.5.2 选择平滑参数 λ

合适的平滑参数 λ 能够压缩**有效自由度 (effective degrees of freedom)**。我们讨论有效自由度而不是自由度的原因是原有的自由度受到了严重压缩，有效自由度的定义为：

$$\hat{\mathbf{g}}_\lambda = \mathbf{S}_\lambda \mathbf{y}$$

其中 $\hat{\mathbf{g}}_\lambda$ 是所有训练数据点的预测值向量，大小 $n\times1$；y 是真实响应值

上式表明，对数据施加平滑样条时拟合值的向量可写错 $n\times n$ 矩阵 $\mathbf{S}_\lambda$ 乘以响应向量 y。定义有效自由度为：

$$df_\lambda = \sum_{i=1}^{n}\{\mathbf{S}_\lambda\}_{ii}$$

也即是 $\mathbf{S}_\lambda$ 的对角线元素之和

拟合光滑样条的时候，我们不需要选择节点数或位置，因为每一个训练值都是一个结点。经验表明，使用 LOOCV 可以非常有效地计算 RSS 较小的 λ 值，下面是简化公式：

$$\begin{aligned} RSS_{cv}(\lambda) &= \sum_{i=1}^{n}(y_i - \hat{g}_\lambda^{(-i)}(x_i))^2 \\ &= \sum_{i=1}^{n}\left[\frac{y_i - \hat{g}_\lambda(x_i)}{1 - \{\mathbf{S}_\lambda\}_{ii}}\right]^2 \end{aligned}$$

其中 $\hat{g}_\lambda^{(-i)}(x_i)$ 是不包含 (x_i, y_i) 时的模型拟合值；$\hat{g}_\lambda(x_i)$ 表示所有的平滑样条拟合值

> 在有效自由度定义下，自由度可以是非整数

7.6 局部回归

局部回归 (Local regression) 是一种不同的拟合非线性模型的方法，其算法可以阐述如下：

1. 收集距离点 x_0 最近的 k 个训练值 x_i

2. 对每个点赋予权重 $K_{i0} = K(x_i, x_0)$，使得距离 x_0 最近的点权重最大，距离 x_0 最远的点权重最小；且非最近 K 个点的权重为 0
3. 拟合**加权最小二乘模型 (weighted least squares regression)**，例如，一种最小化目标是最小化：

$$\sum_{i=1}^{n} K_{i0}(y_i - \beta_0 - \beta_1 x_i)^2$$

4. 在这个点 x_0 处，x_0 的预测值由 $\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$ 给出
5. 选取新的 x_0 重复计算，直至拟合一条平滑的拟合曲线

> 这个模型和 KNN 很像，但是局部回归的曲线相比于 KNN 要更加平滑

选择合适的窗口大小决定了回归的平滑程度，当 $s = \frac{k}{n}$ 越大时，拟合曲线的波动就越小

局部回归可以进行扩展，例如：

- 部分变量上应用的局部回归。例如对于时间序列数据，人口 X_1 和 GDP X_2 对于放假的影响是长期稳定的，适合全局回归，但时间 X_3 却可能表现出短期趋势变化，适合局部回归。我们因此构造一个**变系数模型 (varying coefficient model)** 对数据进行拟合
- 高维度数据的局部回归。对于二维数据，我们可以利用二维空间中的最近点来拟合局部回归；但是对于 $p > 3$ 或者 $p > 4$ 以上的空间，这个模型的效果较差，这是因为稀疏的临近数据

7.7 广义可加模型

广义可加模型 (generalized additive model, GAM) 在保持**可加性 (additivity)** 同时提供了一种广泛的扩展普通线性回归的框架，可用于定性和定量的变量，相当于多元线性回归模型的扩展

7.7.1 用于回归问题的 GAM

用函数代替多元回归中的不同特征项，我们得到：

$$\begin{aligned} y_i &= \beta_0 + \sum_{j=1}^{p} f_j(x_{ij}) + \epsilon_i \\ &= \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_p(x_{ip}) + \epsilon_i \end{aligned}$$

这被称为可加性模型，因为我们对于每一个特征 X_j，都有对应的一个函数 f_j 并对它们进行加总

我们讨论过的大多数模型都可以用于构建 GAM。`pygam` 提供了 Python 中的 GAM 模型构建器，值得一提的是它关于拟合平滑样条的方法**回归自适应拟合 (Backfitting)**，它能够：

- 处理多个预测变量，即使它们非线性
- 不受到变量个数的限制，可通过可加性独立考察每个变量的影响
- 平滑方法灵活，包括局部回归，样条回归和核平滑

GAM 的局限性体现在模型仅限于可加性，不过我们可以手动添加交互项，或者加入地位相互作用函数

7.7.2 用于分类问题的 GAM

多元逻辑回归模型：

$$\log\left(\frac{p(X)}{1-p(X)}\right)=\beta_0+\sum_{i=1}^{p}\beta_i X_i$$

扩展到 GAM：

$$\log\left(\frac{p(X)}{1-p(X)}\right)=\beta_0+\sum_{i=1}^{p}f_i(X_i)$$

可用于分类问题

7.8 实验：非线性模型

基础引入包：

```
import numpy as np, pandas as pd
from matplotlib.pyplot import subplots
import statsmodels.api as sm
from ISLP import load_data
from ISLP.models import (summarize,
                         poly,
                         ModelSpec as MS)
from statsmodels.stats.anova import anova_lm
```

扩展包：

```
from pygam import (s as s_gam,
                   l as l_gam,
                   f as f_gam,
                   LinearGAM,
                   LogisticGAM)
from ISLP.transforms import (BSpline,
                             NaturalSpline)
from ISLP.models import bs, ns
from ISLP.pygam import (approx_lam,
                        degrees_of_freedom,
                        plot as plot_gam,
                        anova as anova_gam)
```

本次实验使用的数据集是 `Wage`

7.8.1 多项式回归和阶跃函数

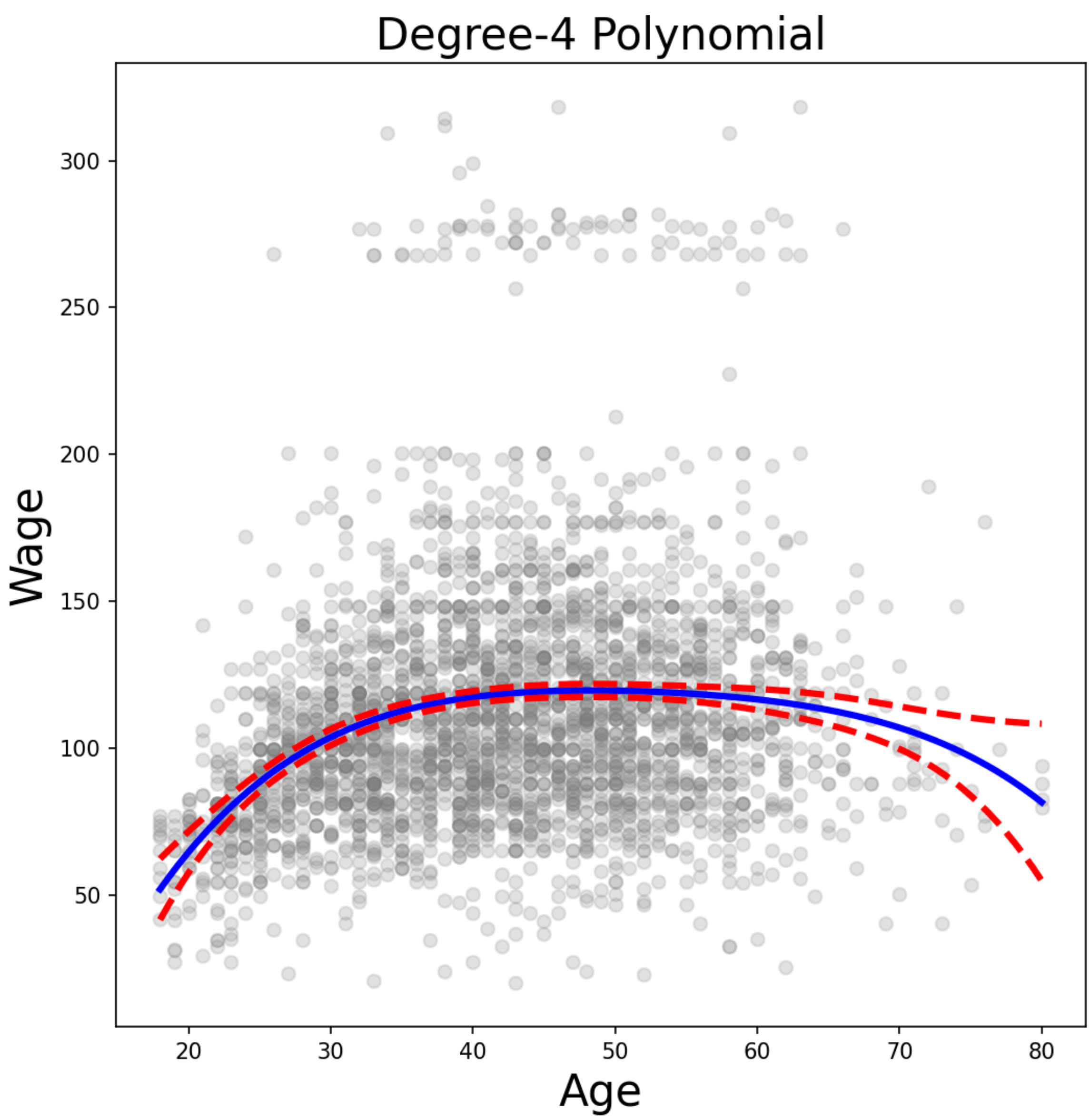
`ISLP` 包的 `poly` 函数是一个辅助函数，用于构建多项式， `Poly()` 则是实际执行的**转换器 (transformer)**，进行实际变换。类似的转换器还有 `PCA()`：

```
#导入数据
Wage = load_data('Wage')
y = Wage['wage']
age = Wage['age']

#拟合4次age项
poly_age = MS([poly('age', degree=4)]).fit(Wage)
M = sm.OLS(y, poly_age.transform(Wage)).fit()
print(summarize(M))
```

我们通过函数来复用绘图代码。其中， `basis` 参数可用于指定转换器：

```
def plot_wage_fit(age_df, basis, title):
    X = basis.transform(Wage) #用指定转换器获取用于训练的特征矩阵
    Xnew = basis.transform(age_df) #用制定转换器获取用于预测的特征矩阵
    M = sm.OLS(y, X).fit()
    preds = M.get_prediction(Xnew) #获取预测值
    bands = preds.conf_int(alpha=0.05) #获取95%置信区间
    fig, ax = subplots(figsize=(8, 8))
    ax.scatter(age, y, facecolor='grey', alpha=0.2) #绘制原始响应变量的散点图
    for val, ls in zip([preds.predicted_mean, #打包了两个列表
                        bands[:, 0],
                        bands[:, 1]],
                        ['b', 'r--', 'r--']):
        ax.plot(age_df.values, val, ls, lw=3)
    ax.set_title(title, fontsize=20)
    ax.set_xlabel('Age', fontsize=20)
    ax.set_ylabel('Wage', fontsize=20)
    return ax
```



其中的`zip()`返回一个**迭代器(iterator)**，可用于for循环；`alpha`参数常用于指定透明度，我们尝试调用函数绘图：

```
#设定x轴范围
age_grid = np.linspace(age.min(),
                       age.max(),
                       100)
age_df = pd.DataFrame({'age': age_grid})
plot_wage_fit(age_df, poly_age, 'Degree-4 Polynomial')
plt.show()
```

我们用**方差分析(analysis variance)** 来确定使用多项式的次数，这需要引入`anova_lm()`包：

```
#通过方差分析确定拟合多项式的次数
models = [MS([poly('age', degree=d)])
          for d in range(1, 6)] #拟合包含5个模型的列表
```

```python
Xs = [model.fit_transform(Wage)
      for model in models] #拟合每一个模型
print(anova_lm(*[sm.OLS(y, X_).fit()
                 for X_ in Xs])) #对模型进行方差分析
```

`anova_lm` 接受一个模型参数，用于M_1和M_2之间的比较。传参时用到的 `*` 表明将列表中的每个元素单独传参，而不是直接传入列表(符合函数要求)。我们分析p值来确定模型的优劣，一个足够小的p值被用于确定更优的模型

> 除了 `anova_lm` 外，还可以使用模型描述直接查看F值。模型描述中的t值的平方等于F值

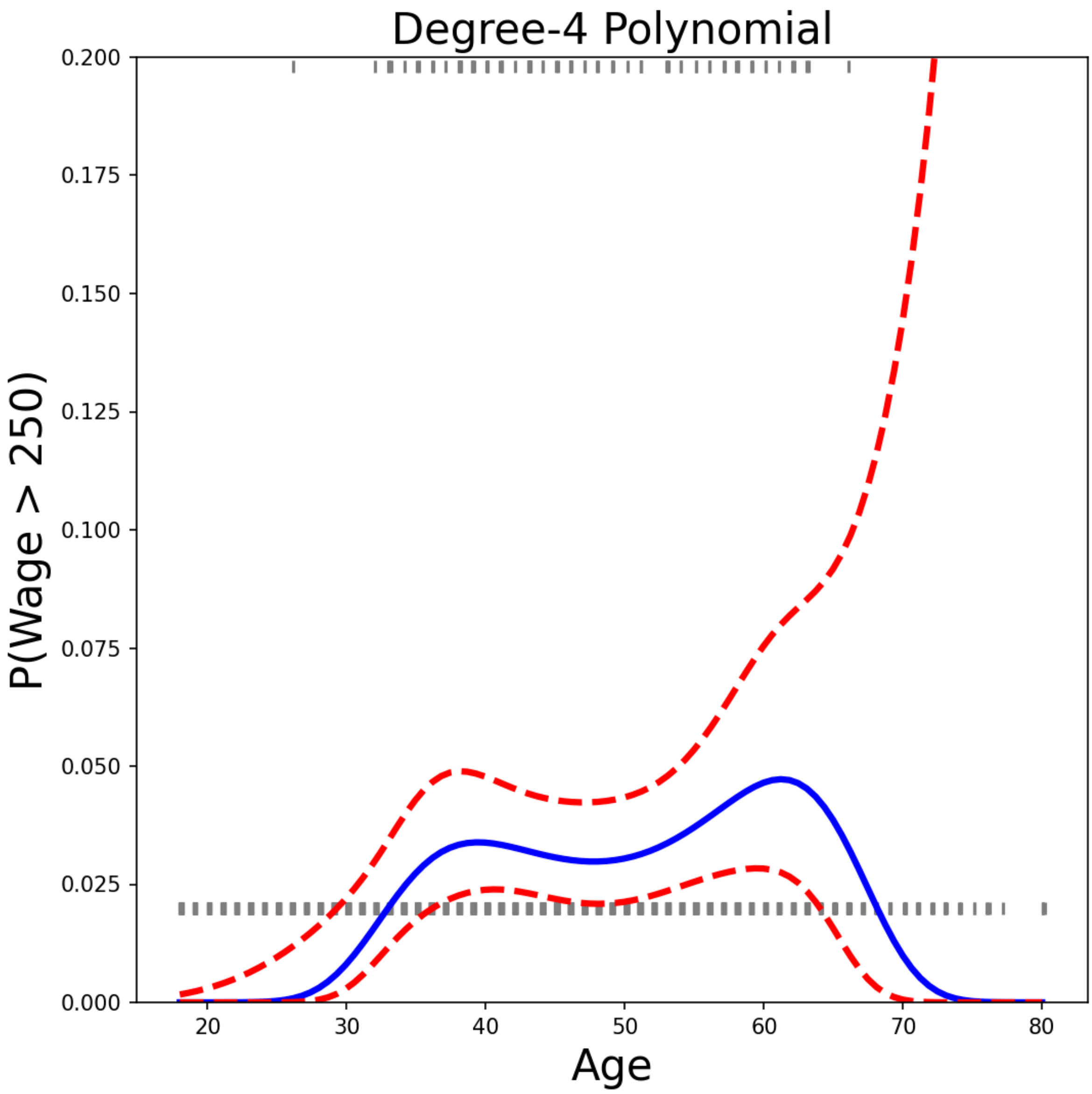
利用方差分析查看包含 `education` 项的条件下， `age` 的合适次数：

```python
#确定包含education条件下, age的次数
models = [MS(['education', poly('age', degree=d)])
          for d in range(1, 4)]
XEs = [model.fit_transform(Wage)
       for model in models]
print(anova_lm(*[sm.OLS(y, X_).fit() for X_ in XEs]))
```

可用交叉检验替代方差分析

下面用多项式拟合逻辑回归，并绘制95%置信预测区间的图片：

```python
#进行逻辑回归的分类预测
newX = poly_age.transform(age_df)
preds = B.get_prediction(newX)
bands = preds.conf_int(alpha=0.05)
fig, ax = subplots(figsize=(8, 8))
rng = np.random.default_rng(0)
ax.scatter(age + 0.2 * rng.uniform(size=y.shape[0]), #添加扰动, 防止点的重叠
           np.where(high_earn, 0.198, 0.02), #条件返回函数
           fc='grey',
           marker='|')
for val, ls in zip([preds.predicted_mean,
                    bands[:, 0],
                    bands[:, 1]],
                   ['b', 'r--', 'r--']):
    ax.plot(age_df.values, val, ls, lw=3)
ax.set_title('Degree-4 Polynomial', fontsize=20)
ax.set_xlabel('Age', fontsize=20)
ax.set_ylim([0, 0.2])
ax.set_ylabel('P(Wage > 250)', fontsize=20)
plt.show()
```



我们用到了`np.where()`方法。它的语法`np.where(condition, x, y)`表示满足条件时，返回`x`否则返回`y`

用于显示数据分布密度的图像是`rug plot`，这在图表中表现为许多灰色竖线

下面拟合阶跃函数的回归模型，我们首先要对数据进行分箱操作，这用到了`pd.qcut()`方法，它对一个数据，按分箱数量自动选取分位数进行分箱；之后，用不同分箱作为X的特征，进行回归：

```
cut_age = pd.qcut(age, 4)
print(summarize(sm.OLS(y, pd.get_dummies(cut_age)).fit()))
```

其中`pd.get_dummies()`将分箱好的数据划分为哑变量，进行回归

除了自动按照分位数分箱，我们还可以指定分箱的分割点(不同于分位数)，使用`pd.cut()`方法

7.8.2 样条

我们用 ISLP 包中的转换器拟合回归样条，用 scipy.interpolate 对拟合函数进行评价。它们已被包装为类似 Poly() 和 PCA() 的转换器

BSpline() 函数生成一个B-样条基矩阵，默认三次；默认情况下的样条维数=样条阶数+结点数+1

bs() 是对 BSpline() 的封装，能够自动丢弃 intercept 列，避免共线性。下面拟合一个B-样条：

```
#准备数据，显示BSpline的特点
bs_ = BSpline(internal_knots=[25, 40, 60],
              intercept=True).fit(age) #指定允许截距项
bs_age = bs_.transform(age)
print(bs_age.shape)

#拟合三次样条
bs_age = MS([bs('age',
            internal_knots=[25, 40, 60],
            name='ba(age)')]) #指定命名
Xbs = bs_age.fit_transform(Wage)
M = sm.OLS(y, Xbs).fit()
print(summarize(M))
```

基函数矩阵是一个 n * m 的矩阵，其中 n 是样本数量， m 是基函数数量，其作用是映射原有的样本特征值到一个新的矩阵上，以捕捉非线性关系

我们可以指定自由度，使 BSpline 自动选择分位点进行拟合：

```
#显示直接指定自由度的结点
print(BSpline(df=6).fit(age).internal_knots_)
```

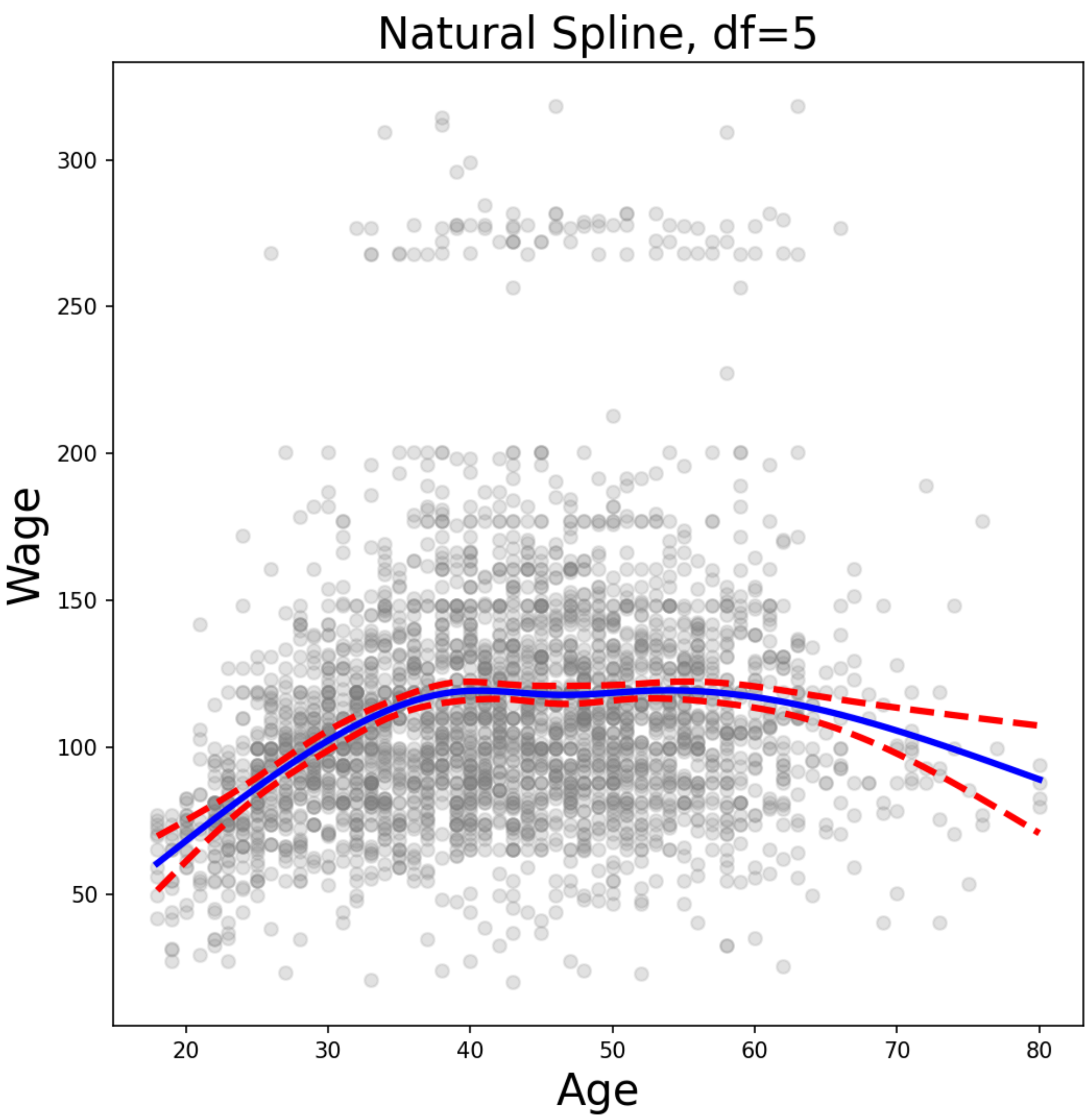
通过指定 degree 参数，可以指定 BSpline 的拟合次数。当 degree=0 时，为阶跃函数

```
#指定阶数为0，生成分段常数函数
bs_age0 = MS([bs('age',
                 df=3,
                 degree=0)]).fit(Wage)
Xbs0 = bs_age0.transform(Wage)
print(summarize(sm.OLS(y, Xbs).fit()))
```

> 这里的阶跃函数与 pd.qcut() 生成的有细小区别，主要因为区间不等号的区别和计算区别

为了拟合自然样条，我们使用 NaturalSpline() 函数，它已被包装为 ns()：

```
#拟合自然样条
ns_age = MS([ns('age', df=5)]).fit(Wage)
M_ns = sm.OLS(y, ns_age.transform(Wage)).fit()
print(summarize(M_ns))
age_grid = np.linspace(age.min(),
                       age.max(),
                       100)
age_df = pd.DataFrame({'age': age_grid})
plot_wage_fit(age_df,
              ns_age,
              'Natural Spline, df=5')
plt.show()
```

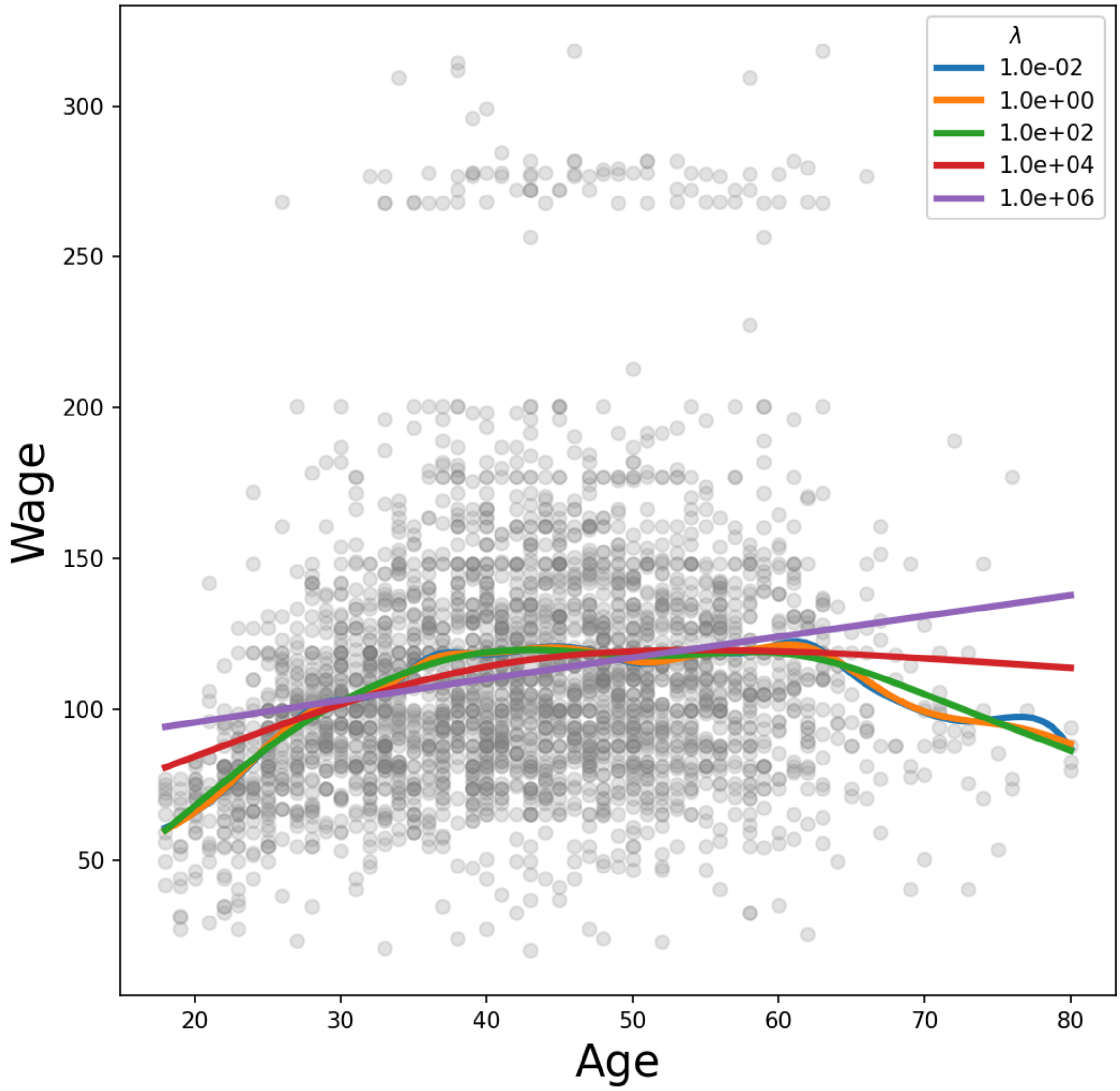


7.8.3 平滑样条和GAMs

我们在Python中使用`pygam`包来拟合GAM模型，注意平滑样条是GAM的一种特殊情况。GAM将响应变量Y与多个预测变量X_i之间的关系表示为多个平滑函数的和，现在我们使用仅有一列的矩阵来拟合：

```
#拟合GAM模型
X_age = np.asarray(age).reshape((-1, 1)) #将向量转换为矩阵
gam = LinearGAM(s_gam(0, lam=0.6))
gam.fit(X_age, y)

#作出拟合度随lambda变化的图像
fig, ax = subplots(figsize=(8, 8))
ax.scatter(age, y, fc='gray', alpha=0.2)
for lam in np.logspace(-2, 6, 5):
    gam = LinearGAM(s_gam(0, lam=lam)).fit(X_age, y)
    ax.plot(age_grid,
            gam.predict(age_grid),
            label="{:.1e}".format(lam),
            lw=3)
ax.set_xlabel('Age', fontsize=20)
ax.set_ylabel('Wage', fontsize=20)
ax.legend(title='$\lambda$')
plt.show()
```



我们此处将一个列向量转换为了$n\times1$的矩阵，满足`LinearGAM`模型的要求

`pygam`包中有自动的λ搜索功能。例如：

```
#自动搜索合适的参数
gam_opt = gam.gridsearch(X_age, y)
ax.plot(age_grid,
        gam_opt.predict(age_grid),
        label='Grid Search',
        lw=4)
ax.legend()
plt.show()
```

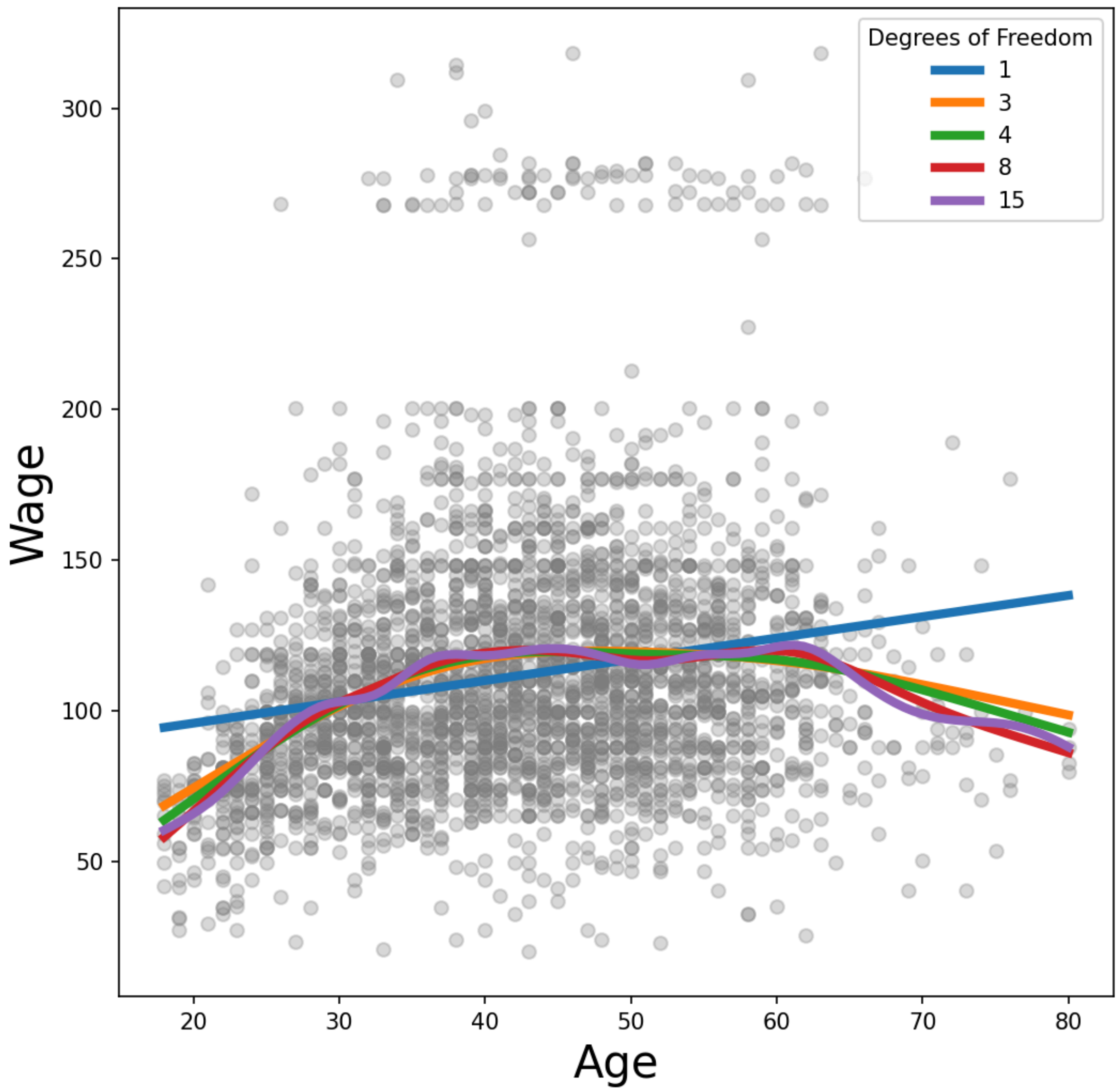
> 这个搜寻的评估方式是交叉检验，且在输出过程中会显示进度条(并非报错)

`ISLP.pygam`是一种替代的设定自由度的方法，能够寻找满足指定自由度的$lambda$值：

```python
#指定自由度
age_term = gam.terms[0]
lam_4 = approx_lam(X_age, age_term, 4) #指定自由度为4
age_term.lam = lam_4 #设定拟合好的自由度
print(degrees_of_freedom(X_age, age_term)) #检查大小
```

我们同样绘制曲线关于λ变化的图像：

```python
#绘制自由度图像
fig, ax = subplots(figsize=(8, 8))
ax.scatter(X_age,
           y,
           fc='gray',
           alpha=0.3)
for df in [1, 3, 4, 8, 15]:
    lam = approx_lam(X_age, age_term, df+1) #忽略线性项自带的1自由度
    age_term.lam = lam
    gam.fit(X_age, y)
    ax.plot(age_grid,
            gam.predict(age_grid),
            label="{:d}".format(df),
            lw=4)
ax.set_xlabel('Age', fontsize=20)
ax.set_ylabel('Wage', fontsize=20)
ax.legend(title='Degrees of Freedom')
plt.show()
```



> 图像的变化由自由度变动引起，自由度的选取使用了 `approx_lam()` 函数，接受特征矩阵中的某一列，拟合模型参数对应的一项和自由度

GAM的强大之处在于多个变量的处理函数的相加，我们可以通过手动指定和 `pygam` 包完成这些任务

手动设计矩阵并拟合模型：

```
#构造设计矩阵
ns_age = NaturalSpline(df=4).fit(age)
ns_year = NaturalSpline(df=5).fit(Wage['year'])
Xs = [ns_age.transform(age),
      ns_year.transform(Wage['year']),
      pd.get_dummies(Wage['education']).values] #将分类变量转换为哑变量并获取值
```

```python
X_bh = np.hstack(Xs) #对列表进行水平拼接，即每行包括所有的列表元素
gam_bh = sm.OLS(y, X_bh).fit()
```

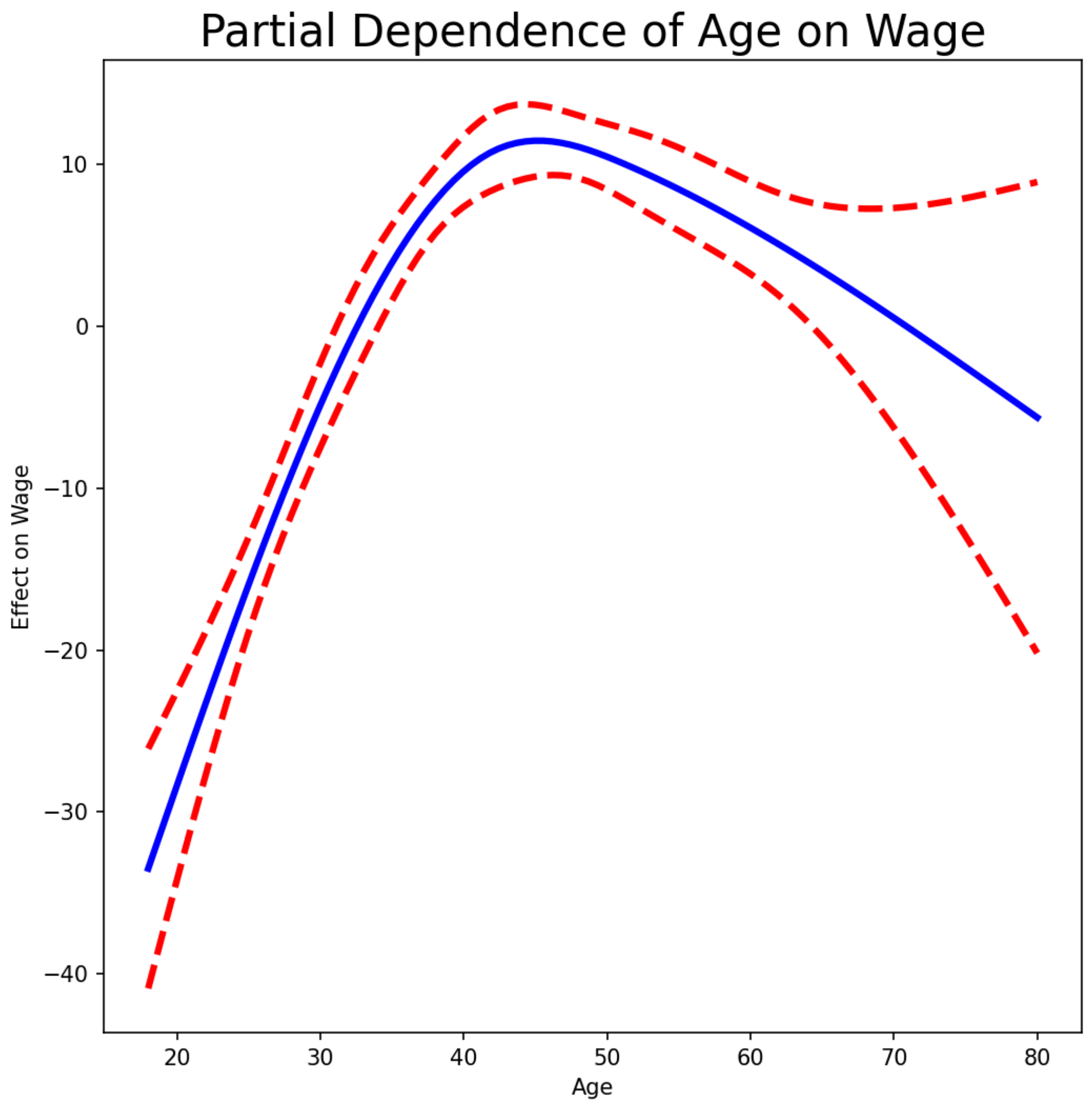
我们可以单独查看每个变量的影响：

```python
#绘制图片
age_grid = np.linspace(age.min(),
                       age.max(),
                       100)
X_age_bh = X_bh.copy()[:100] #复制设计矩阵前100行数据，用于后续生成预测值
X_age_bh[:] = X_bh[:].mean(0)[None, :] #将所有行替换为按列矩阵，固定除age外所有变量
X_age_bh[:, :4] = ns_age.transform(age_grid) #将前4列替换为自然样条变换值
preds = gam_bh.get_prediction(X_age_bh)
partial_age = preds.predicted_mean #获取预测模型的均值
center = partial_age.mean() #计算age部分的均值，对部分效应去中心化
bounds_age = preds.conf_int(alpha=0.05) #获得预测的置信区间
partial_age -= center #进行去中心化
bounds_age -= center
fig, ax = subplots(figsize=(8, 8))
ax.plot(age_grid, partial_age, 'b', lw=3)
ax.plot(age_grid, bounds_age[:, 0], 'r--', lw=3)
ax.plot(age_grid, bounds_age[:, 1], 'r--', lw=3)
ax.set_xlabel('Age')
ax.set_ylabel('Effect on Wage')
ax.set_title('Partial Dependence of Age on Wage')
```

我们试图创建一个新的预测矩阵，除了年龄外的列都是常数(以突出年龄的影响)

1. 先获取100行的矩阵，用于后续预测
2. 将这个矩阵的每行替换为原来的列均值
3. 将代表年龄的前四列，转换为 `age_grid` 计算的自然样条基

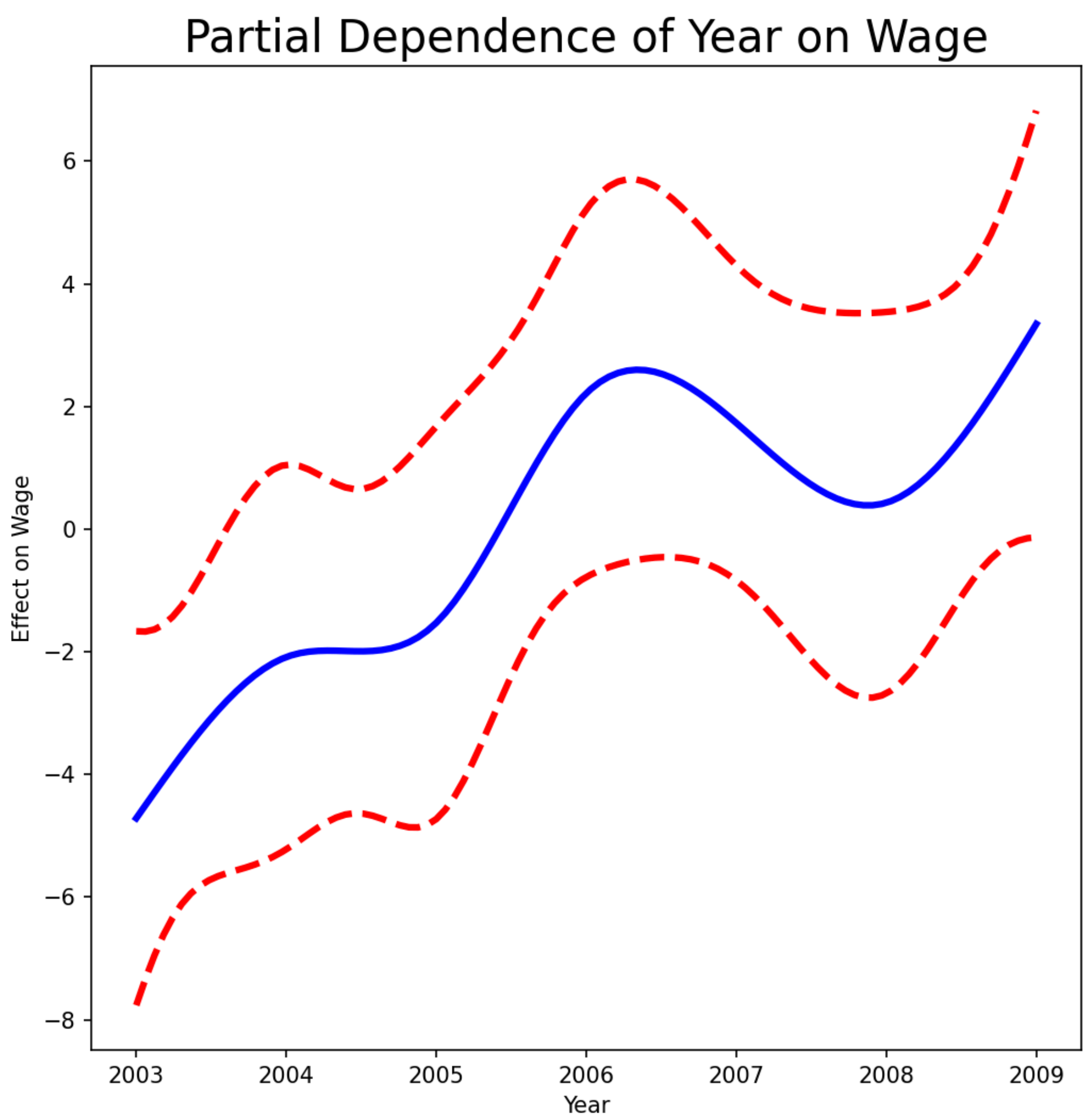
> 上述代码中的一个操作 `[None, :]` 可将一个列表转换为一个矩阵，例如将 `[1, 2, 3, 4]` 转换为 `[[1, 2, 3, 4]]`



继续查看year对Wage的影响：

```
#显示year的影响
year_grid = np.linspace(2003, 2009, 100)
year_grid = np.linspace(Wage['year'].min(),
                        Wage['year'].max(),
                        100)
X_year_bh = X_bh.copy()[:100]
X_year_bh[:] = X_bh[:].mean(0)[None, :]
X_year_bh[:, 4:9] = ns_year.transform(year_grid)
preds = gam_bh.get_prediction(X_year_bh)
bounds_year = preds.conf_int(alpha=0.05)
partial_year = preds.predicted_mean
center = partial_year.mean()
partial_year -= center
bounds_year -= center
fig, ax = subplots(figsize=(8, 8))
ax.plot(year_grid, partial_year, 'b', lw=3)
```

```
ax.plot(year_grid, bounds_year[:, 0], 'r--', lw=3)
ax.plot(year_grid, bounds_year[:, 1], 'r--', lw=3)
ax.set_xlabel('Year')
ax.set_ylabel('Effect on Wage')
ax.set_title('Partial Dependence of Year on Wage', fontsize=20)
plt.show()
```



同时对多个特征拟合GAM，可以观察相互之间的影响：

```
#准备数据，拟合多变量GAM
gam_full = LinearGAM(s_gam(0) + #生成LinearGAM模型，对第一个变量用平滑样条
                     s_gam(1, n_splines=7) + #对第二个变量用平滑样条，指定7个基
函数
                     f_gam(2, lam=0)) #对第三个变量用分类处理方式，lam=0指定不正
则化
Xgam = np.column_stack([age, #合成矩阵
```

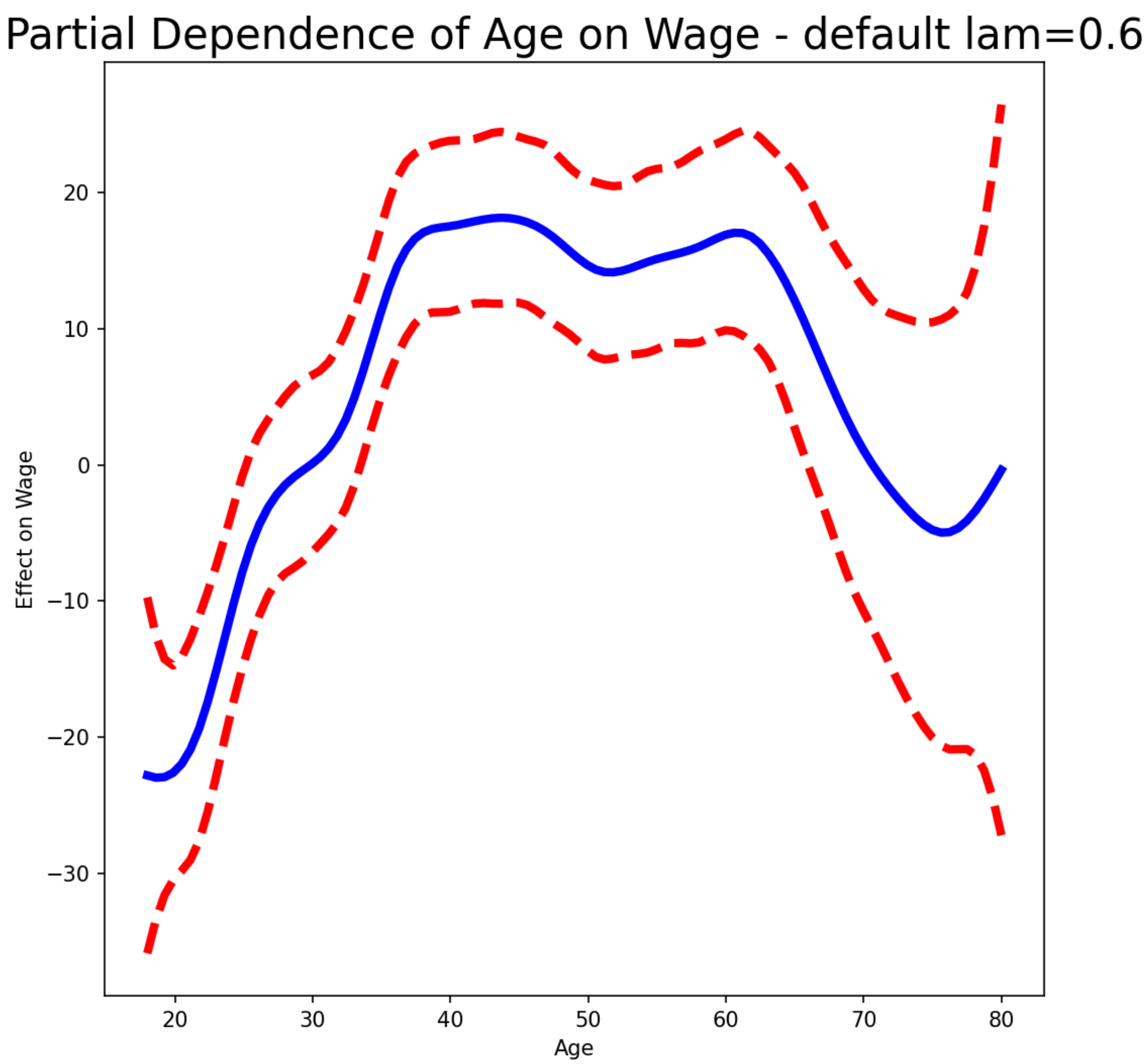
```
                          Wage['year'],
                          Wage['education'].cat.codes]) #将分类变量进行数字编码
gam_full = gam_full.fit(Xgam, y) #拟合模型
```

其中的 `s_gam` 和 `f_gam` 都是指定变量样条形式的函数； `cat.codes` 是 `pandas` 中的一个方法，用于将分类变量映射为从0开始的一系列整数

> `s_gam()` 的默认$\lambda = 0.6$，这是一个经验值，可根据实际调整

我们可以查看在存在交互的情况下， `Age` 变量对 `Wage` 的影响：

```
#绘制拟合图像
fig, ax = subplots(figsize=(8, 8))
plot_gam(gam_full, 0, ax=ax)
ax.set_xlabel('Age')
ax.set_ylabel('Effect on Wage')
ax.set_title('Partial Dependence of Age on Wage - default lam=0.6',
             fontsize=20)
plt.show()
```



注意到上面的图像波动性较大

指定自由度 `df` 往往比指定平滑参数 `lam` 更加自然，这是因为前者更加直观，而后者需要通过交叉检验来确定

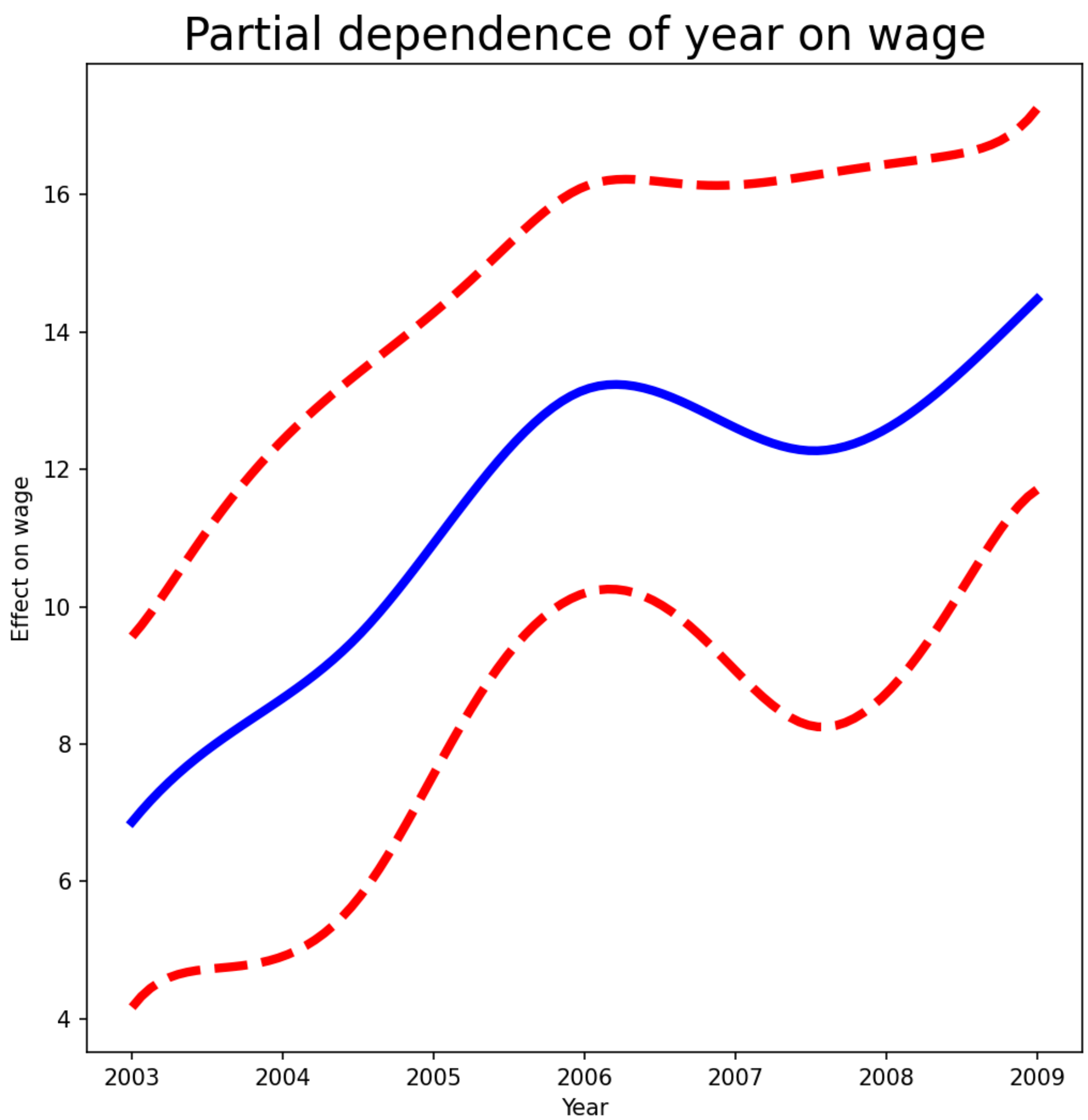
现在指定 `df=4` ，重新拟合模型：

```
#按自由度拟合模型
age_term = gam_full.terms[0]
age_term.lam = approx_lam(Xgam, age_term, df=4+1) #为截距预留一个自由度
year_term = gam_full.terms[1]
year_term.lam = approx_lam(Xgam, year_term, df=4+1)
gam_full = gam_full.fit(Xgam, y)
```

`age_term` 等是一个软拷贝，修改它的 `lam` 也会修改到模型 `gam_full` 中的参数

重新绘制图像：

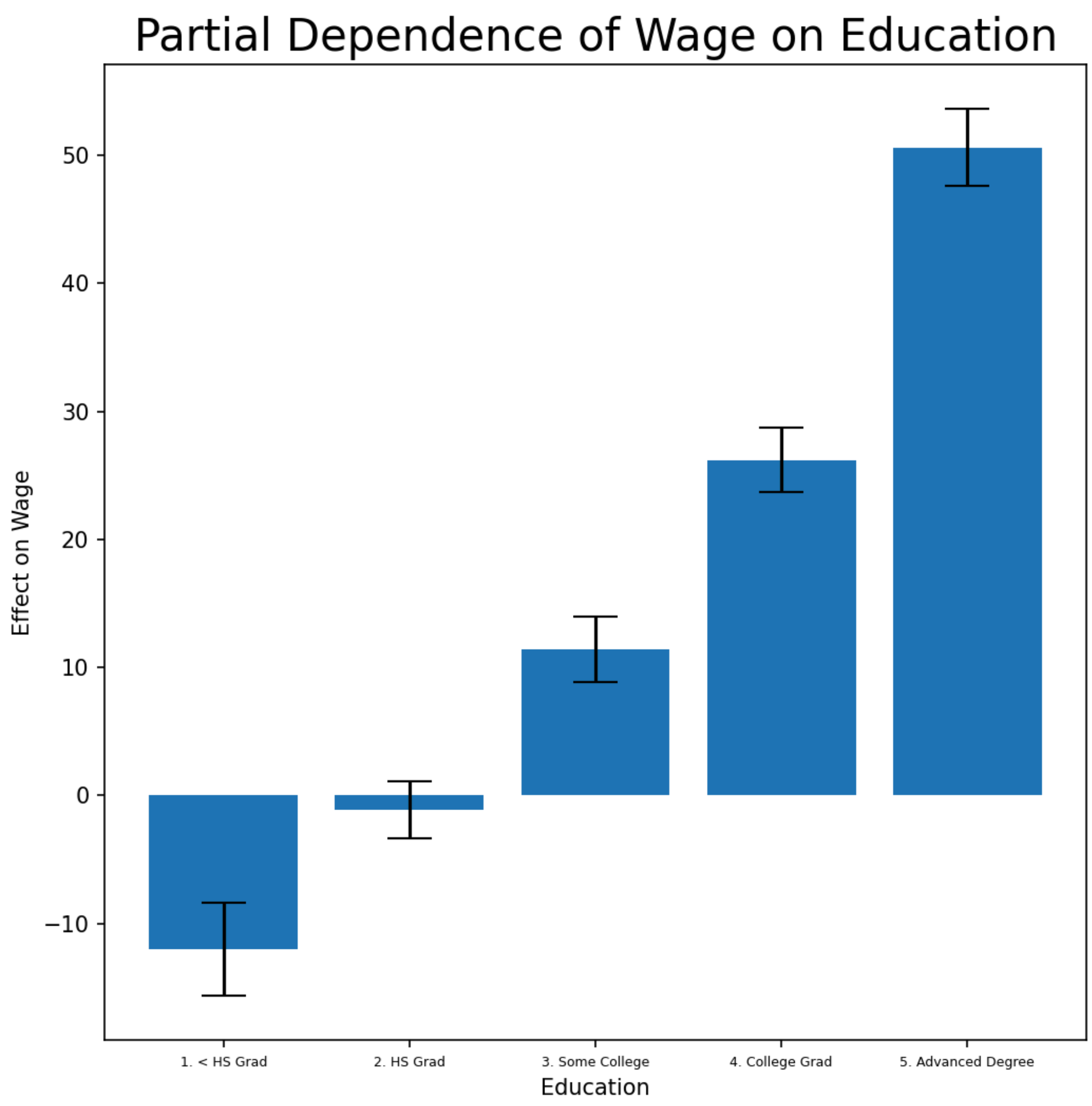
```
#重新显示图像
fig, ax = subplots(figsize =(8 ,8))
plot_gam(gam_full , 1, ax=ax)
ax.set_xlabel('Year')
ax.set_ylabel('Effect on wage')
ax.set_title('Partial dependence of year on wage', fontsize=20)
plt.show()
```



与数值型的特征不同，分类特征适合每一类特征对应一类预测值，下面绘制关于`education`的部分依赖图：

```
#关于education的图像
fig, ax = subplots(figsize=(8, 8))
ax = plot_gam(gam_full, 2)
ax.set_xlabel('Education')
ax.set_ylabel('Effect on Wage')
```

```
ax.set_title('Partial Dependence of Wage on Education',
             fontsize=20)
ax.set_xticklabels(Wage['education'].cat.categories, fontsize=6)
plt.show()
```



`year` 在所有模型中都表现得相当线性，我们通过一系列方差分析来决定下面哪些模型更好：

- 不包含 `year` 的GAM
- 使用线性 `year` 项的GAM
- 使用样条函数 `year` 的GAM

```
#拟合模型
gam_0 = LinearGAM(age_term + f_gam(2, lam=0)) #排除year模型
```

```
gam_0.fit(Xgam, y)
gam_linear = LinearGAM(age_term + l_gam(1, lam=0) + f_gam(2, lam=0)) #线性year模型
gam_linear.fit(Xgam, y)

#方差分析
print(anova_gam(gam_0, gam_linear, gam_full))
```

分析结果表明M_3对M_2的p值不够充分小，即样条函数的GAM不足以比线性模型好很多，但线性模型比不包含`year`模型好

`age`模型的方差分析表明它需要一个非线性项：

```
#对age重复操作
gam_0 = LinearGAM(year_term +
                  f_gam(2, lam=0))
gam_linear = LinearGAM(l_gam(0, lam=0) +
                       year_term +
                       f_gam(2, lam=0))
gam_0.fit(Xgam, y)
gam_linear.fit(Xgam, y)
print(anova_gam(gam_0, gam_linear, gam_full))
```

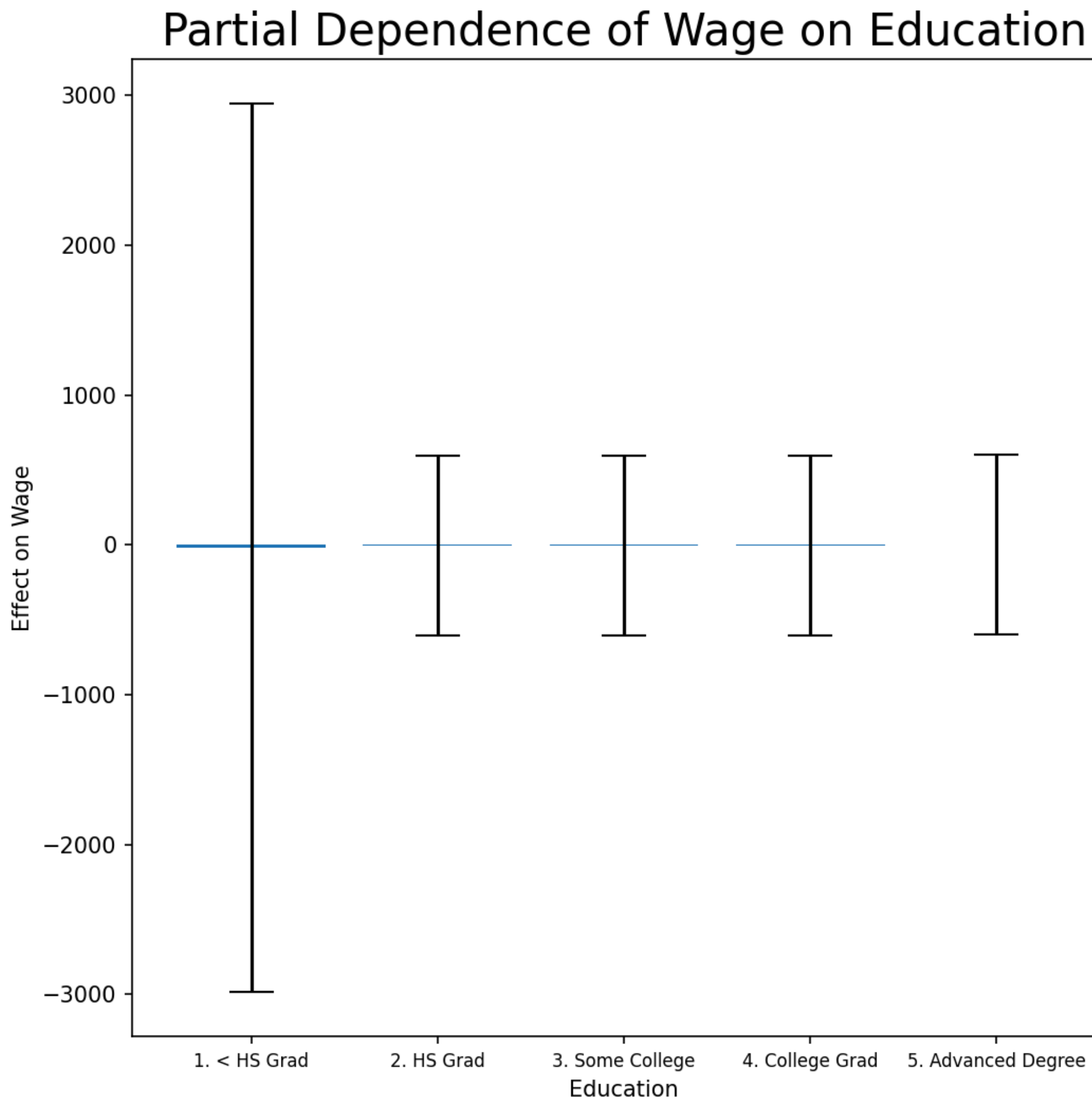
GAM模型可以查看模型参数，也可以进行预测

```
print(gam_full.summary()) #查看描述
Yhat = gam_full.predict(Xgam) #预测
```

GAM模型可以用于分类问题，使用`LogisticGAM`：

```
#分类问题
gam_logit = LogisticGAM(age_term +
                        l_gam(1, lam=0) +
                        f_gam(2, lam=0))
high_earn = Wage['high_earn'] = y > 250
gam_logit.fit(Xgam, high_earn)

#显示分类结果
fig, ax = subplots(figsize=(8, 8))
ax = plot_gam(gam_logit, 2)
ax.set_xlabel('Education')
ax.set_ylabel('Effect on Wage')
ax.set_title("Partial Dependence of Wage on Education",
             fontsize=20)
ax.set_xticklabels(Wage['education'].cat.categories, fontsize=8)
plt.show()
```



第一类分类很奇怪，我们来看看 `Wage` 的交叉表，以显示和 `education` 的关系：

```
#显示交叉表
print(pd.crosstab(Wage['high_earn'], Wage['education']))
```

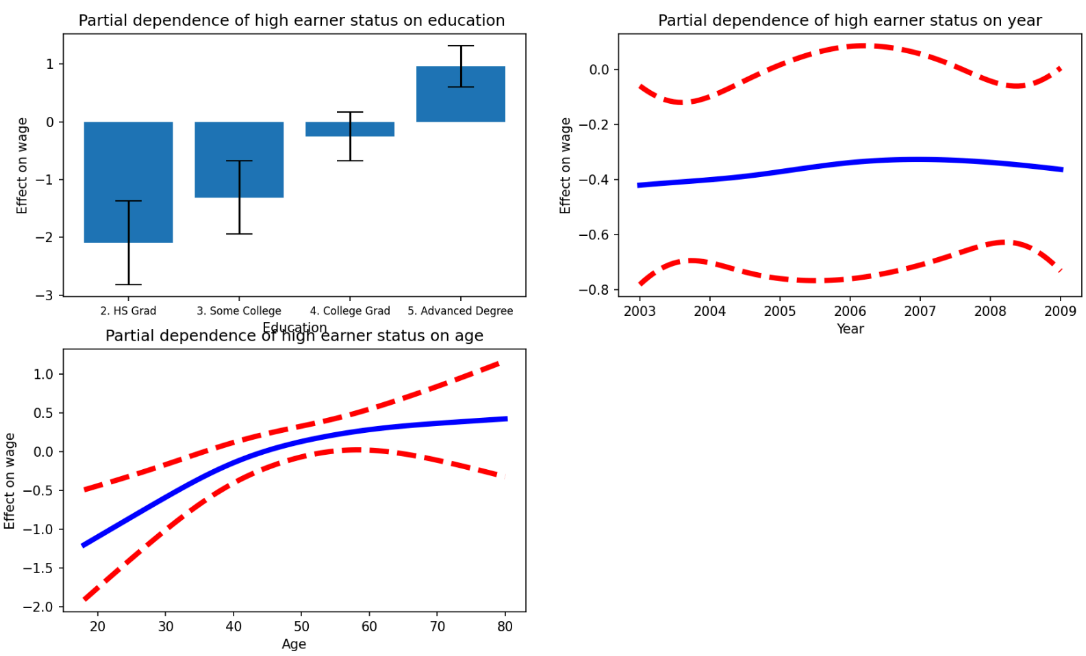
结果显示第一类没有 `high_earn` 分类的，因此无法准确预测，下面删掉这一类分类：

```
#删除第一类分类
only_hs = Wage['education'] == '1. < HS Grad'
Wage_ = Wage.loc[~only_hs]
Xgam_ = np.column_stack([Wage_['age'],
                         Wage_['year'],
                         Wage_['education'].cat.codes-1]) #减少一类编码
high_earn_ = Wage_['high_earn']
```

基于 `pygam` 的BUG，我们在减少一个分类时需要对编码数量也进行减少

绘制影响图：

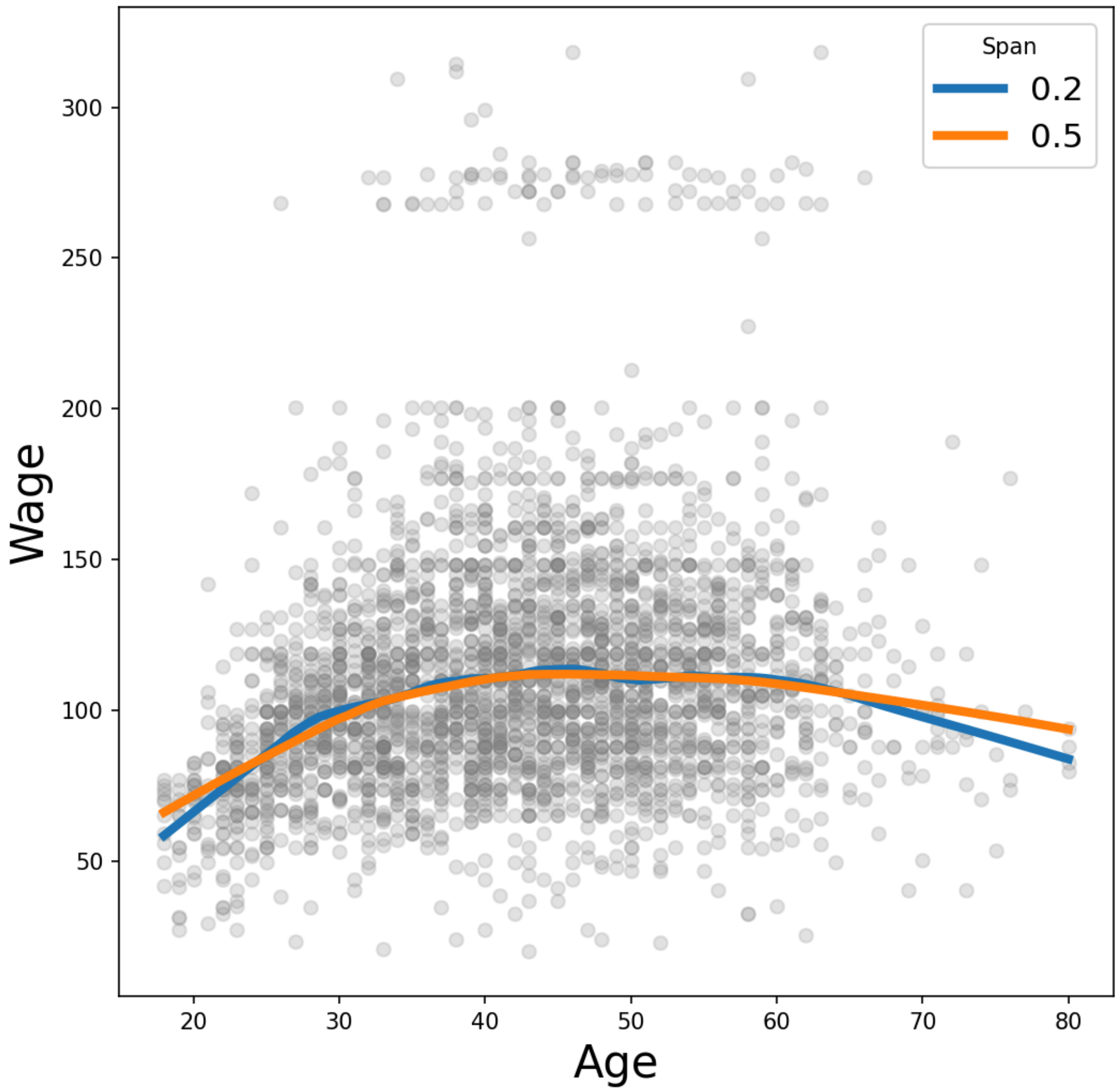
```
#显示三幅图，查看三个变量对高收入的影响
fig, axes = subplots(2, 2, figsize=(14, 8))
ax = axes[0, 0]
plot_gam(gam_logit_ , 2, ax=ax)
ax.set_xlabel('Education')
ax.set_ylabel('Effect on wage')
ax.set_title('Partial dependence of high earner status on education ')
ax.set_xticklabels(Wage['education'].cat.categories[1:], fontsize=8)
ax = axes[0, 1]
plot_gam(gam_logit_, 1, ax=ax)
ax.set_xlabel('Year')
ax.set_ylabel('Effect on wage')
ax.set_title('Partial dependence of high earner status on year')
ax = axes[1, 0]
plot_gam(gam_logit_, 0, ax=ax)
ax.set_xlabel('Age')
ax.set_ylabel('Effect on wage')
ax.set_title('Partial dependence of high earner status on age')
axes[1, 1].remove()
plt.show()
```



7.8.4 局部回归

局部回归使用的函数是 `lowess()` ，相比于前面的操作，局部回归相对简单。我们选取0.2和0.5的紧邻点来拟合局部回归：

```python
#拟合局部回归模型
lowess = sm.nonparametric.lowess
fig, ax = subplots(figsize=(8, 8))
ax.scatter(age, y, fc='grey', alpha=0.2)
for span in [0.2, 0.5]:
    fitted = lowess(y,
                    age,
                    frac=span,
                    xvals=age_grid)
    ax.plot(age_grid,
            fitted,
            label='{:.1f}'.format(span),
            lw=4)
ax.set_xlabel('Age', fontsize=20)
ax.set_ylabel('Wage', fontsize=20)
ax.legend(title='Span', fontsize=15)
plt.show()
```



容易发现，0.5的窗口值比0.2的窗口值的图像更加平滑

#CS
#ML