Splines

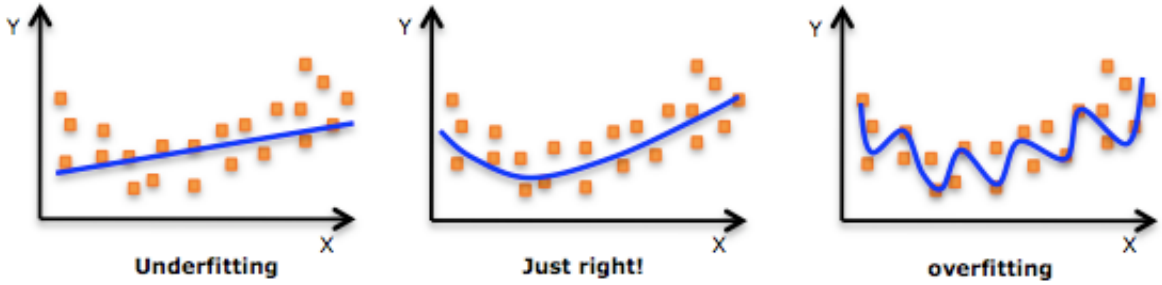
- Assume $(X, Y) \in \mathbb{R}^p \times \mathbb{R}$
- Goal: approximate true regression function $f(X) = E[Y|X]$
- GLM assumption: $g(E[Y|X]) = X\beta$, is a linear function of X so $f(X) = g^{-1}(X\beta)$ where $\beta = (\beta_0, \beta_1, \ldots, \beta_p)$

- Question: What are the chances that the true regression function takes a GLM form, $E[Y|X] = g^{-1}(X\beta)$, for any real-world application? (choose one)
  - A. Very high
  - B. Somewhat high
  - C. Very low
  - D. No chance at all

- Question: Why ever assume linearity? (choose all that apply)
  - A. Simplicity
  - B. Interpretability
  - C. Minimal risk of overfitting
  - D. First-order Taylor approximation of true regression function $E[Y|X]$

- Splines are regression techniques that allow more complex functional relationships between Y and X
- If $p = 1$ (if there is only one column in the data), a spline would take the form $g(E[Y|X]) = \sum_{m=1}^{M} \beta_m h_m(X)$
- $h_1, \ldots, h_M$ are transformations of X

- $g(E[Y|X])$ is modeled as a GLM of the transformation
- As the number of transformations, M, increases, so does model complexity



- Typically, the challenge is in choosing the appropriate M

- Question: How would you choose M?
  - A. Best subset selection
  - B. Lasso
  - C. Visual inspection
  - D. AIC
  - E. Not sure

- It is not often the case that we only want to model a single column
- Generalized Additive model: when $p > 1$,

$$g(E[Y|X]) = \sum_{j=1}^{p} \sum_{m=1}^{M_j} \beta_{mj} h_{mj}(X_j)$$

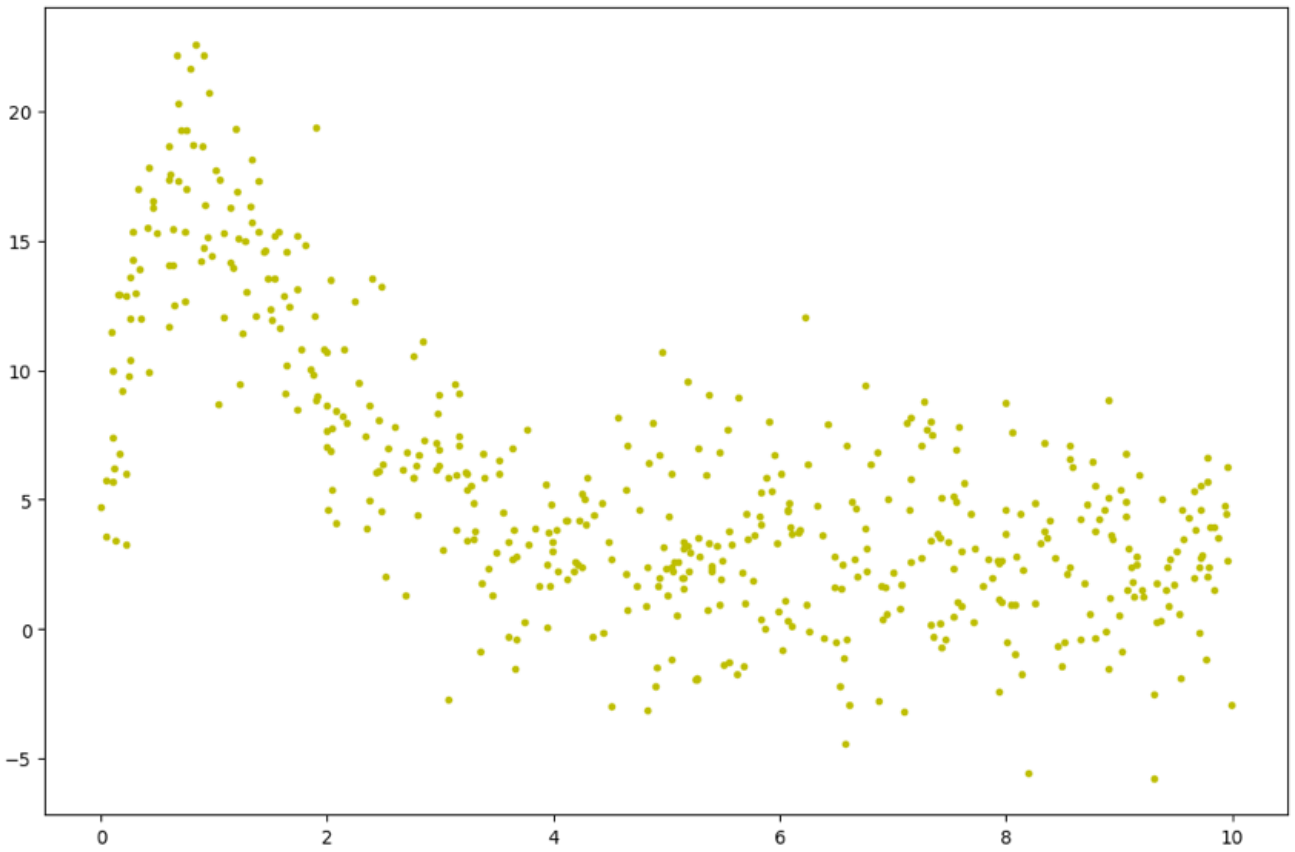
- Here, must choose M_j for each variable X_j
- Let's start with a toy example

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from patsy import dmatrix
import seaborn as sns
%matplotlib inline
```

In [2]:
```
# true regression function
def f(x):
    return(x*(x-5)*(x-10)/np.exp(x)+3)

n = 500 # sample size
x = np.sort(np.random.uniform(low=0, high=10, size=n))
y = f(x)+np.random.normal(0, 3, n)
```

In [3]:
```
plt.rcParams['figure.figsize'] = [12, 8]
plt.rcParams['figure.dpi'] = 100
plt.plot(x,y,'.y')
plt.show()
```



- Question: Would simple linear regression do a good job here?
  - A. Yes
  - B. No
  - C. Not sure

- In real life, we wouldn't know the true regression function
- I would question what was happening with points near $X = 0$
- Could they be outliers?
- In this case, they provide information

In [4]:
```
X = sm.add_constant(x)
model = sm.OLS(y, X)
results = model.fit()
results.summary()
```

Out[4]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.441
Model:	OLS	**Adj. R-squared:**	0.440
Method:	Least Squares	**F-statistic:**	393.6
Date:	Thu, 17 Feb 2022	**Prob (F-statistic):**	5.66e-65
Time:	11:22:07	**Log-Likelihood:**	-1403.8
No. Observations:	500	**AIC:**	2812.
Df Residuals:	498	**BIC:**	2820.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	11.6892	0.352	33.199	0.000	10.997	12.381
x1	-1.2078	0.061	-19.838	0.000	-1.327	-1.088

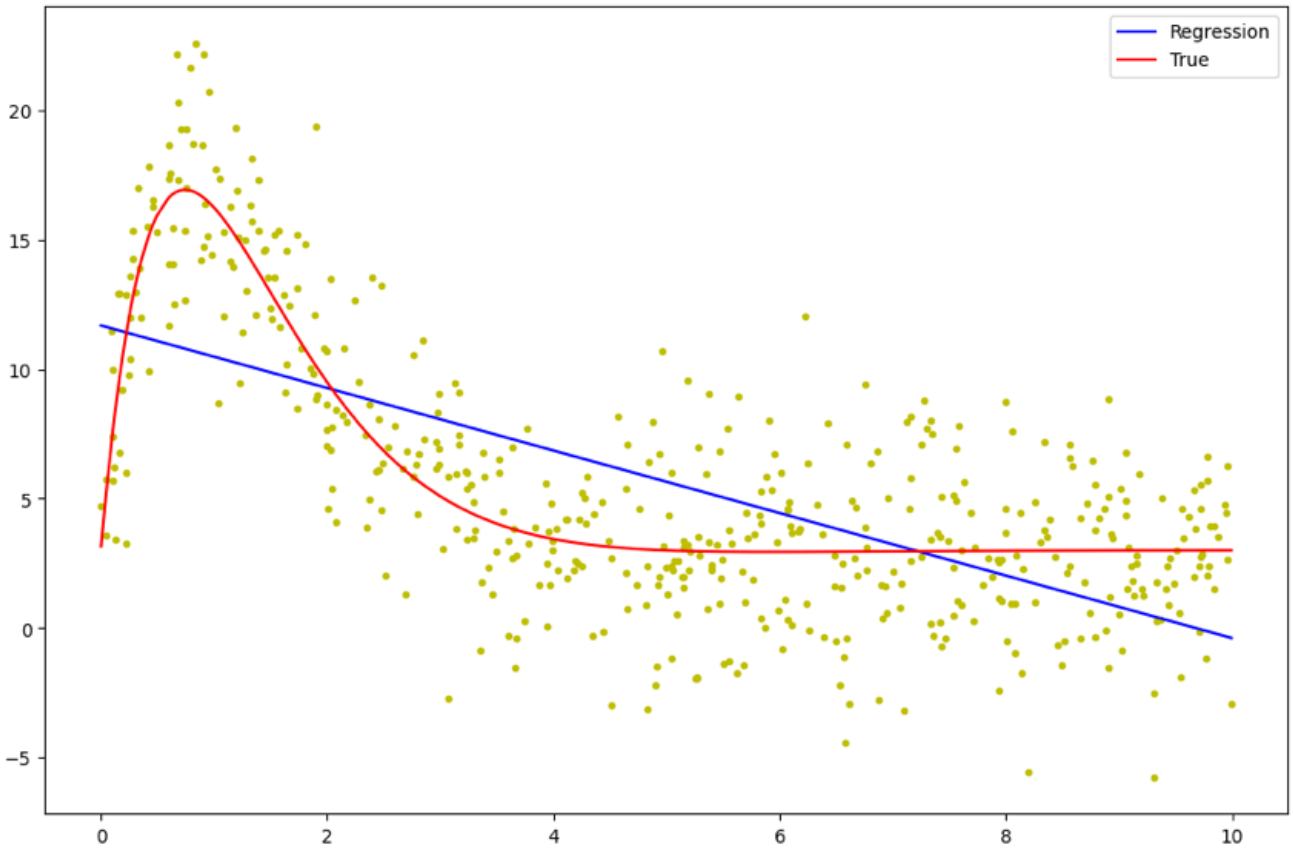
Omnibus:	3.937	**Durbin-Watson:**	1.061
Prob(Omnibus):	0.140	**Jarque-Bera (JB):**	3.851
Skew:	0.176	**Prob(JB):**	0.146
Kurtosis:	2.752	**Cond. No.**	11.6

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [5]:

```
plt.plot(x,y,'.y')
plt.plot(x, x*results.params[1]+results.params[0], '-b',
         label='Regression')
plt.plot(x, f(x), '-r', label='True')
plt.legend();
```


Regression
True
20
15
10
5
0
−5
0
2
4
6
8
10

- Clear that fit is not perfect
- Is seeing the true regression function influencing our feelings about the true model?

- Question: If prediction is the primary objective, would the linear model be ok?
  - A. Yes
  - B. No
  - C. Depends on more things
  - D. Not sure

- Question: If inference were our goal, would this model be ok?
  - A. Yes
  - B. No
  - C. Depends on more things
  - D. Not sure

- If this were our only variable, it would be important to take time to try to understand it better
- If this were one variable among many, it would depend on how well this variable explains the outcome
- Typically, outcomes have many sources of variation, not just one
- A linear model could still be helpful, depending on setting
- Easy to interpret: on average, for each unit increase in x, there is a statistically significant ($p < 0.001$) 1.2 unit decrease in y
- Let's see how a smoothing spline does

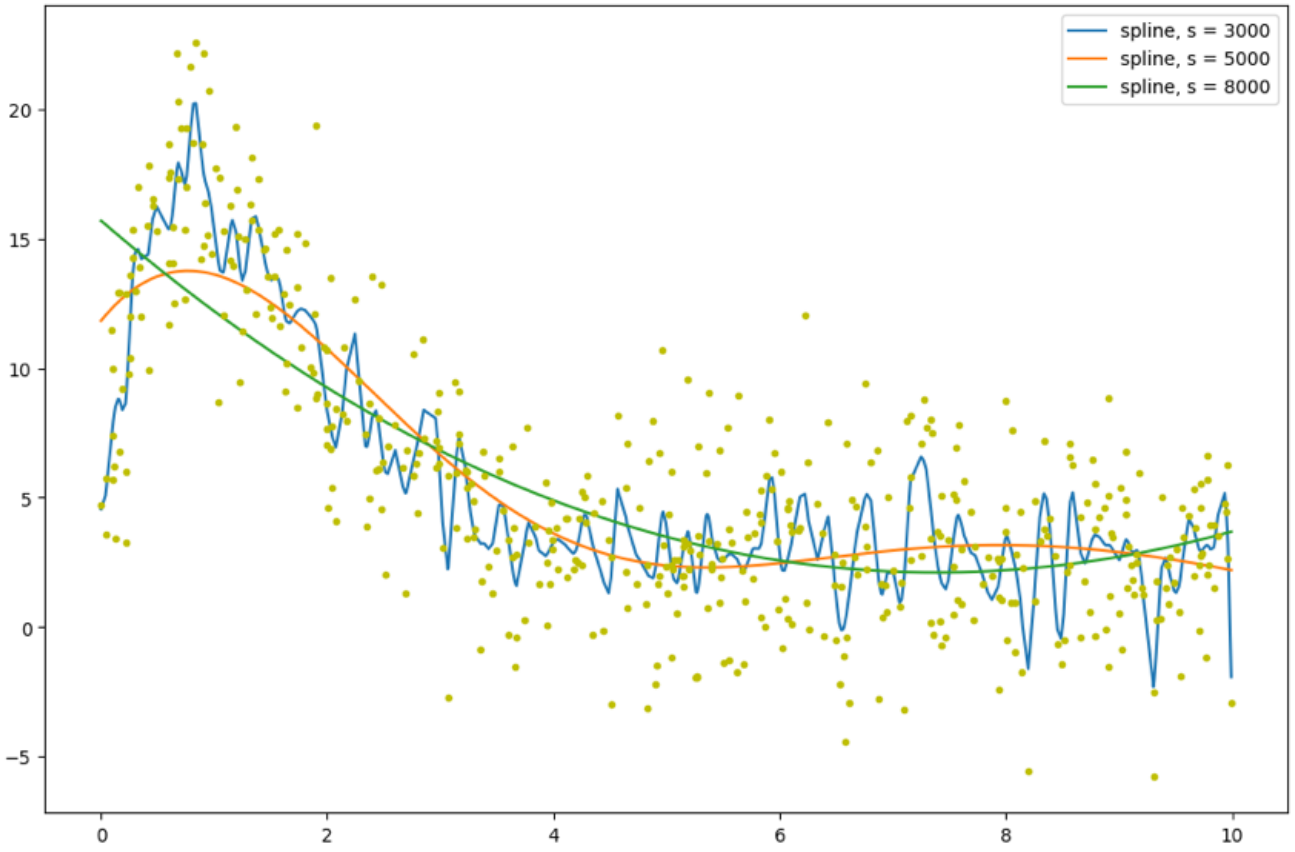
In [6]:

```
from scipy.interpolate import UnivariateSpline

spl = UnivariateSpline(x, y)

for sm in [3000, 5000, 8000]:
    sm_fact = sm
    spl.set_smoothing_factor(sm_fact)
    plt.plot(x, spl(x), label=f"spline, s = {np.round(sm_fact)}")

plt.plot(x,y,'.y')
plt.legend();
```



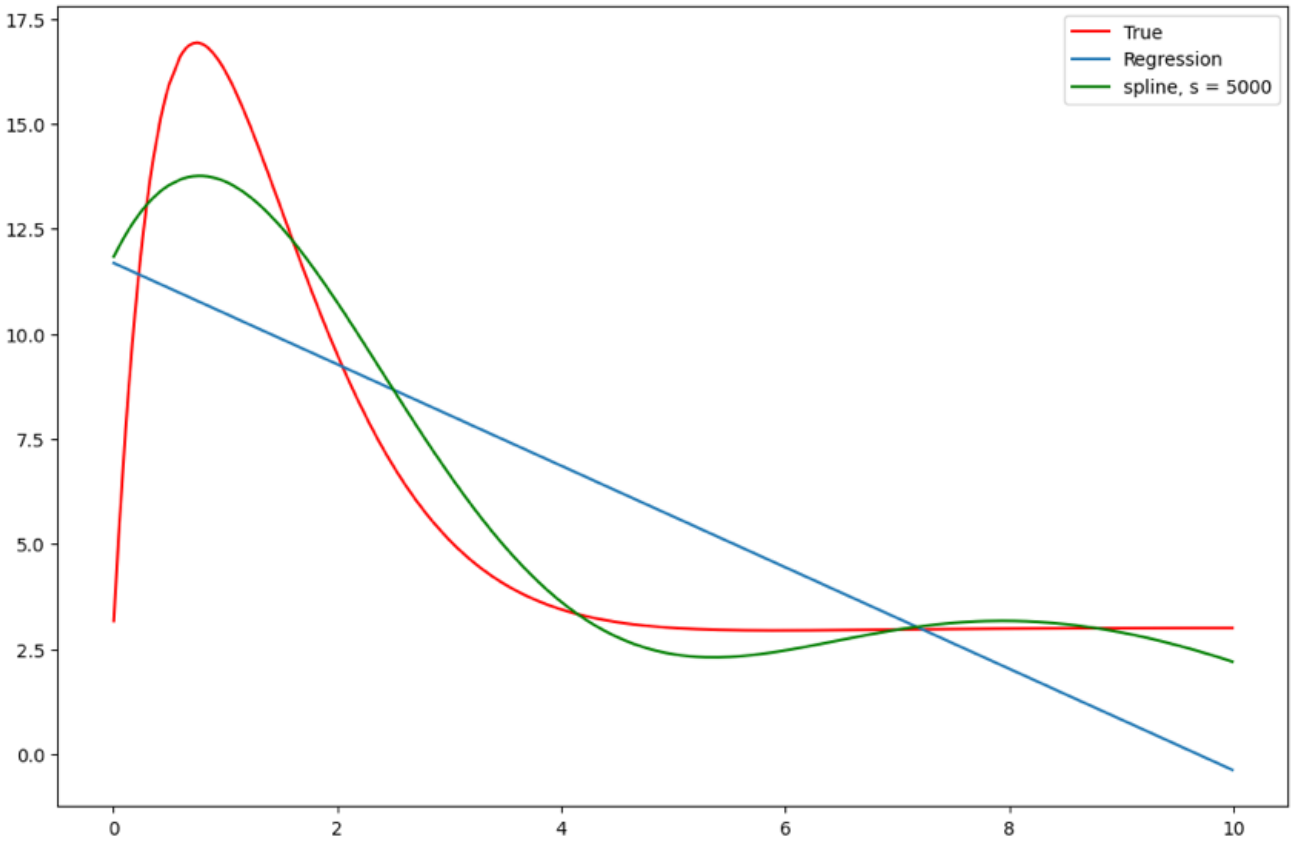
- Question: Which spline appears best?
  - A. $s = 3000$
  - B. $s = 5000$
  - C. $s = 8000$
  - D. Not sure

- This plot is too busy but it shows how important choosing complexity is to fitting splines
- From these three, $s = 5000$ seems to have the best fit
- The spline parameter estimates are not easily interpretable but a plot with the single $s = 5000$ curve does explain the conditional mean well
- Let's compare the spline to the true regression curve

In [7]:
```
plt.plot(x, f(x), '-r', label='True')
plt.plot(x, x*results.params[1]+results.params[0], '-',
         label='Regression')

spl = UnivariateSpline(x, y)

sm_fact = 5000
spl.set_smoothing_factor(sm_fact)
plt.plot(x, spl(x), '-g', label=f"spline, s = {np.round(sm_fact)}")
plt.legend();
```



- The spline is still not perfect but better than linear

- Question: Is the spline model interpretable?
  - A. Yes
  - B. No
  - C. Not sure

Case Study: South African Heart Disease

- Subset of Coronary Risk-Factor Study (CORIS) baseline survey from 1983
- Data collected in 3 rural areas of the Western Cape, South Africa
- Aim: Establish intensity of ischemic heart disease risk factors

- Note: Western Cape is a high-incidence region
- Data: White males between age 15 and 64
- Outcome: Past myocardial infarction (CHD in the data) at time of survey (prevalence 5.1%)
- Question: Why not use proportional hazards regression here?
  - A. This dataset is a subset of another dataset
  - B. The study group (white males) is too monolithic
  - C. There is no time to event data
  - D. Not sure

A retrospective sample of males in a heart-disease high-risk region of the Western Cape, South Africa. There are roughly two controls per case of CHD. Many of the CHD positive men have undergone blood pressure reduction treatment and other programs to reduce their risk factors after their CHD event. In some cases the measurements were made after these treatments. These data are taken from a larger dataset, described in Rousseauw et al, 1983, South African Medical Journal.

Data dictionary:

- **sbp:** systolic blood pressure
- **tobacco:** cumulative tobacco (kg)
- **ldl:** low density lipoprotein cholesterol
- **famhist:** family history of heart disease (Present, Absent)
- **obesity:** body mass index (bmi)
- **age:** age at onset
- **chd:** response, coronary heart disease

Goal: For this task, our primary objective is prediction (not inference), but we want to be able to interpret the model as well

In [8]:
```
url = 'https://web.stanford.edu/~hastie/ElemStatLearn//datasets/SAheart.data'
dat = pd.read_csv(url)
dat = dat[['sbp', 'tobacco', 'ldl', 'famhist', 'obesity', 'age', 'chd']]
print(dat.shape)
dat.head()
```

(462, 7)

Out[8]:

	sbp	tobacco	ldl	famhist	obesity	age	chd
0	160	12.00	5.73	Present	25.30	52	1
1	144	0.01	4.41	Absent	28.87	63	1
2	118	0.08	3.48	Present	29.14	46	0
3	170	7.50	6.41	Present	31.99	58	1
4	134	13.60	3.50	Present	25.99	49	1

In [9]:

```
dat.describe().round(2)
```

Out[9]:

	sbp	tobacco	ldl	obesity	age	chd
count	462.00	462.00	462.00	462.00	462.00	462.00
mean	138.33	3.64	4.74	26.04	42.82	0.35
std	20.50	4.59	2.07	4.21	14.61	0.48
min	101.00	0.00	0.98	14.70	15.00	0.00
25%	124.00	0.05	3.28	22.98	31.00	0.00
50%	134.00	2.00	4.34	25.80	45.00	0.00
75%	148.00	5.50	5.79	28.50	55.00	1.00
max	218.00	31.20	15.33	46.58	64.00	1.00

- Note: `describe` by default only looks at numeric variables
- Must also look at non-numeric variables as well

In [10]:

```
print(dat.dtypes)
dat.describe(include='object')
dat.famhist.value_counts()
```

```
sbp           int64
tobacco     float64
ldl         float64
famhist      object
obesity     float64
age           int64
chd           int64
dtype: object
```

Out[10]:

```
Absent     270
Present    192
Name: famhist, dtype: int64
```

- Exploring data
- `seaborn` has some nice exploratory data visualization methods

In [11]:

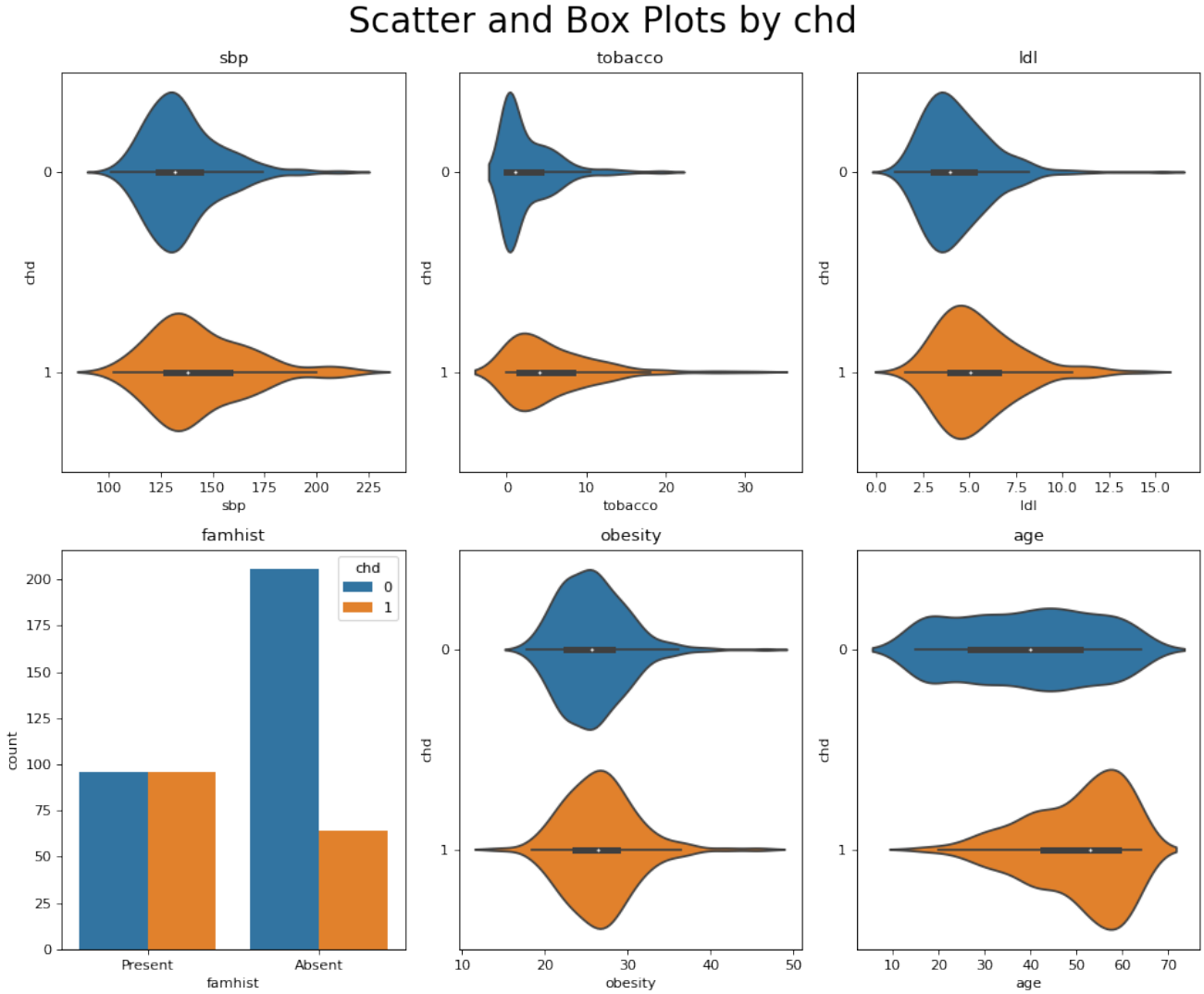
```
g = sns.PairGrid(dat, hue="chd")
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
g.add_legend();
```

- **Violin** plots can be more helpful by showing the densities more clearly
- This can be helpful for bimodal densities
- An alternative is a box plot

In [12]:

```
def draw_outcome_plots(df, outcome, n_rows, n_cols):
    fig=plt.figure(figsize=(12, 10), dpi=80)
    variables = df.columns.drop(outcome)
    for itr, var_name in enumerate(variables):
        ax=fig.add_subplot(n_rows,n_cols,itr+1)
        if len(df[var_name].unique()) > 10:
            sns.violinplot(ax=ax, x=var_name, y=outcome,
                           orient='h', data=df)
        else:
            sns.countplot(ax=ax, x=var_name, hue=outcome,
                          data=df);
        ax.set_title(var_name)
    fig.suptitle('Scatter and Box Plots by '+outcome,
                 size=25)
    fig.tight_layout()
    plt.show()

draw_outcome_plots(dat, 'chd', 2, 3)
```


Scatter and Box Plots by chd
sbp
tobacco
ldl
famhist
obesity
age
chd
0
1
count
Present
Absent
famhist
sbp
tobacco
ldl
obesity
age

Modeling

- Question: We're not doing inference here. Is overfitting possible?
  - A. Yes
  - B. No
  - C. Not sure

- Remember: goal is to predict outcome with an interpretable model
- Interpretable means that we can understand how change in X will affect our prediction
- With splines, we typically won't give parameter estimates
- We will understand model predictions using spline plots
- First, investigate GLM logistic regression is sufficient
- `statsmodels` module outcome variable and design matrix can only include numeric variables
- We can either
  - use formula that indicates that a variable is categorical
  - manually create dummy variables for categorical data
- Using an R-style formula with `from_formula`
  - Reference: `C` operator converts numeric categorical variables to dummies

In [13]:
```
fml = 'chd ~ sbp + tobacco + ldl + famhist + obesity + age'
results = sm.GLM.from_formula(
    fml, data=dat, family=sm.families.Binomial()).fit()
results.summary()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-13-43a685e05c00> in <module>
      1 fml = 'chd ~ sbp + tobacco + ldl + famhist + obesity + age'
----> 2 results = sm.GLM.from_formula(
      3     fml, data=dat, family=sm.families.Binomial()).fit()
      4 results.summary()

AttributeError: 'int' object has no attribute 'GLM'
```

- Question: Are these parameter estimates, confidence intervals, and p-values valid?
  - A. Yes
  - B. No
  - C. Not sure

- Question: What needs to be done to make this result publication ready? (select all that apply)
  - A. Nothing
  - B. Exponentiate `coef`, `[0.025` , and `0.975]`
  - C. Write variable meanings with units
  - D. Run spline models
  - E. Check diagnostics

- Using design matrix and getting AIC

In [ ]:
```
exog = dat.drop(columns='chd')
exog = sm.add_constant(exog)
exog['famhist'] = dat['famhist'].apply(lambda x: (x == 'Present')*1)
results = sm.Logit(dat.chd, exog).fit()
results.summary()
print(f'AIC: {round(results.aic)}')
```

Backward-Stepwise Selection

- There are many ways to fit a spline model
- Backward stepwise selection is one way
- When prediction is a model goal, including all variables will add random noise to the model
- To avoid this, we perform variable selection
- Idea: Remove one variable at a time based on a fit metric (AIC, deviance, BIC, etc)
- In theory, AIC (BIC) removes selection bias
- In practice this is not completely true
- Question: Why are we not using cross validation to do variable selection? (select all that apply)
  - A. Because AIC/BIC removes selection bias
  - B. Because we would have to split the data
  - C. Because there are $2^6 = 64$ models to consider without using splines
  - D. Because we are doing prediction
- Note: Categorical variables with more that two levels are usually split into multiple dummy variables, usually we will want to include or exclude the entire categorical variable. Stepwise selection can be tricky in this setting
- Algorithm:
  - Compute fit metric for each model removing one variable
  - Chose model with best fit
  - Continue until removing variable no longer improves fit
- Coding note: remember DRY (don't repeat your self)
- Because we will be doing this multiple times, write a function

In [ ]:
```
def back_step(response, exog):
    '''
    Input: response vector, independent (exogenous) variables
    Output: Table of variable removed and corresponding model AIC
    '''
    model_aic = pd.DataFrame(columns=['removed', 'aic'])
    for var in exog.columns:
        inputs = exog.drop(columns=var)
        results = sm.Logit(response, inputs).fit(disp=0)
        aic = results.aic
        model_aic = model_aic.append(
            {'removed': var, 'aic': aic}, ignore_index=True)
    return(model_aic)

back_step(dat.chd, exog)
```

- Note: Remember that for `statsmodels`, smaller AIC is better
- Should remove `sbp`

In [ ]:
```
back_step(dat.chd, exog.drop(columns='sbp'))
```

In [ ]:
```
back_step(dat.chd, exog.drop(columns=['sbp', 'obesity']))
```

- AIC for all of these models greater than 495.44
- Removing any more variables will worsen fit
- Important: standard errors, p-values no longer valid
- Parameters can be interpreted but not CIs

In [ ]:
```
input_dat = exog.drop(columns=['sbp', 'obesity'])
back_step_results = sm.Logit(dat.chd, input_dat).fit(disp=0)
back_step_results.summary()
```

- These coefficients are not helpful
- exponentiate the coefficients to make them more interpretable

In [ ]:
```
back_step_results.params.apply(np.exp)
```

- interpretation:
  - Compared to someone without a family history of coronary heart disease (CHD), an individual with family history of CHD, is 2.5 times more likely to experience CHD, on average
  - For each year increase in age, an individual is, on average, 1.05 times more likely to experience CHD compared to the previous year
  - again, we cannot use p-values here

- *Post-selection inference* is an active area of research

- If inference is important, you can split the data:
  - fit the model on one split
  - show results with other using model from first split

- Notice that this is the same model we would have arrived at if we had removed insignificant variables from first, full model

- This is not always the case, why?

Beyond Linearity with Regression Spline Models

- Most of the time, a simple transformation will not be sufficient
- Can be difficult to visualize necessary transformations (logistic regression for example)
- Spline models can make this easier
- Splines maintain a balance between fit and interpretability
- Splines are piece-wise function, like this

$$g(x) = \begin{cases} g_1(x) & x < \xi_1 \\ g_2(x) & \xi_1 \leq x < \xi_2 \\ g_3(x) & \xi_3 < x \end{cases}$$

- Note: ξ_1, ξ_2, ξ_3 are called knots
- For regression, could transform x into 3 variables based on region

$$\begin{aligned} h_1(x) &= I(x < \xi_1) \\ h_2(x) &= I(\xi_1 \leq x < \xi_2), \\ h_3(x) &= I(\xi_3 < x) \end{aligned}$$

- Note: h_1, h_2, h_3 are called basis functions
- We could use this model

$$g(E[Y|X]) = \beta_1 h_1(x) + \beta_2 h_2(x) + \beta_3 h_3(x)$$

- Why might this not be ideal?
- What would give give us?

$$g(E[Y|X=x]) = \beta_1 h_1(x) + \beta_2 h_2(x) + \beta_3 h_3(x) + \beta_4 h_1(x)x + \beta_5 h_2(x)x + \beta_6 h_3(x)x$$

- What about these transformations

$$\begin{aligned} h_1(x) &= 1 \\ h_2(x) &= x \\ h_3(x) &= (x-\xi_1)_+ \\ h_4(x) &= (x-\xi_2)_+ \end{aligned}$$

where

$$(x)_+ = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

- Model:

$$g(E[Y|X=x]) = \sum_{m=1}^{4} \beta_m h_m(x)$$

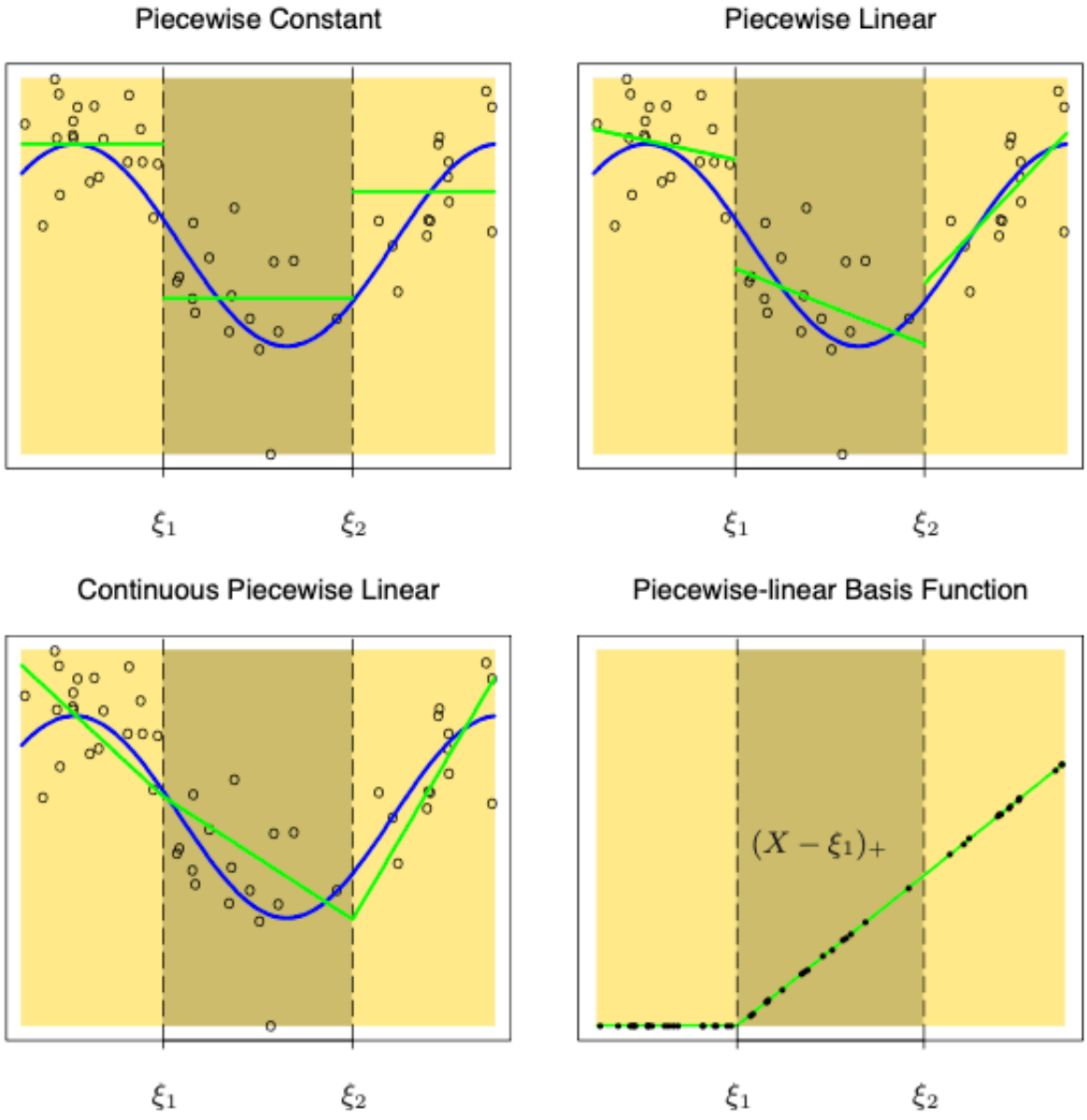


FIGURE 5.1. *The top left panel shows a piecewise constant function fit to some artificial data. The broken vertical lines indicate the positions of the two* knots *ξ_1 and ξ_2. The blue curve represents the true function, from which the data were generated with Gaussian noise. The remaining two panels show piecewise linear functions fit to the same data—the top right unrestricted, and the lower left restricted to be continuous at the knots. The lower right panel shows a piecewise–linear basis function, $h_3(X) = (X - \xi_1)_+$, continuous at ξ_1. The black points indicate the sample evaluations $h_3(x_i)$, $i = 1, \ldots, N$.*

Image from Elements of Statistical Learning

- It's common to prefer smooth functions over linear splines
- Lower right panel: *cubic spline* has continuous with continuous first and second

derivatives at the knots

- It is not difficult to show that these basis function ensure all continuity constraints

$$
\begin{aligned}
h_1(x) &= 1 \\
h_2(x) &= x \\
h_3(x) &= x^2 \\
h_4(x) &= x^3 \\
h_5(x) &= (x - \xi_1)_+^3 \\
h_6(x) &= (x - \xi_2)_+^3
\end{aligned}
$$

- Parameter count:
  $(3 \text{ regions}) \times (4 \text{ parameters per region}) - (2 \text{ knots})$
  $\times (3 \text{ constraints per knot}) = 6$
- Higher-order splines?
  - Thought that cubic splines are the lowest-order splines where human eye cannot detect knots
  - Could you find knots if they weren't indicated?
  - Higher order splines are not frequently used in practice
  - Many parameters: must choose order of spline, number of knots and placement

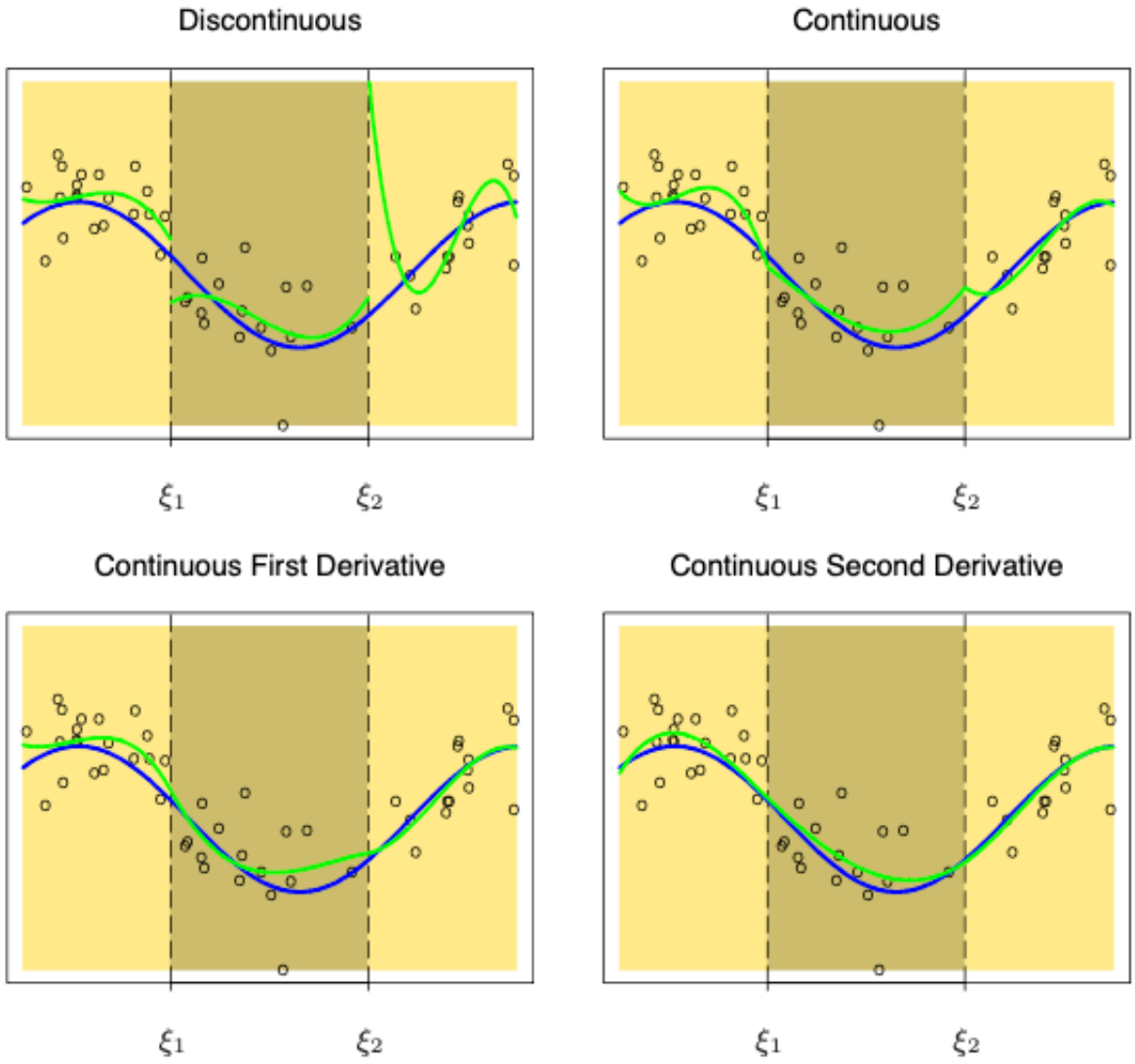


FIGURE 5.2. *A series of piecewise-cubic polynomials, with increasing orders of continuity.*

Image from Elements of Statistical Learning

In [ ]:
```
def knot_placement(var, num):
    '''
    input:
        var: variable as numpy series
        num: number of knots
    output: list of knot placed at quantile locations
    '''
    quants = [(x+1)/(num+1) for x in range(num)]
    return(var.quantile(quants).tolist())

def make_linear_splines(var, knot_locs):
    '''return dataframe for continuous, linear basis'''
    sp_dat = pd.DataFrame(var)
    sp_dat = sm.add_constant(sp_dat)
    for knot in knot_locs:
        knot_name = var.name + "_" +str(int(np.round(knot)))
        sp_dat[knot_name] = var.apply(lambda x: (x-knot)*(x>knot))
    return(sp_dat)

k_locs = knot_placement(dat['age'], 5)
print(k_locs)
lin_age = make_linear_splines(dat['age'], k_locs)
lin_age.head(10)
```

In [ ]:
```
results = sm.Logit(dat.chd, lin_age).fit()
print(results.aic)
results.summary()
```

In [ ]:
```
back_step(dat.chd, lin_age)
```

In [ ]:
```
back_step(dat.chd, lin_age.drop(columns='age_59'))
```

In [ ]:
```
back_step(dat.chd, lin_age.drop(columns=['age_59', 'age_27']))
```

In [ ]:
```
results = sm.Logit(dat.chd, lin_age.drop(
    columns=['age_59', 'age_27'])).fit()
results.summary()
```

In [ ]:
```
coefs = results.params
x_col = np.linspace(dat['age'].min(), dat['age'].max(), 1000)
locs = k_locs[1:4]
print(locs)
y = np.array([coefs[0]+coefs[1]*x+
              sum([coefs[s+2]*(x-k_locs[s])*(x>k_locs[s])
                   for s in range(3)]) for x in x_col])

plt.plot(x_col, y, '-b')
plt.xlabel('Age (yrs)')
plt.ylabel('Logit P(CHD)');
```

In [ ]:
```
plt.plot(x_col, np.exp(y), '-b')
plt.xlabel('Age (yrs)')
plt.ylabel('Odds of CHD');
```

In [ ]:
```
def back_select(response, exog):
    '''
    input: binary response variable and exogenous variables
    output: list of selected variables based on backward stepwise
        selection using AIC
    '''
    results = sm.Logit(response, exog).fit(disp=0)
    last_aic = results.aic
    cur_vars = list(exog.columns)
    improvement = True
    while improvement:
        aics = {}
        for var in cur_vars:
            cur_exog = exog[cur_vars].drop(columns=var)
            results = sm.Logit(response, cur_exog).fit(disp=0)
            aics[var] = results.aic
        min_var = min(aics, key=aics.get)
        cur_aic = aics[min_var]
        if cur_aic < last_aic:
            cur_vars.remove(min_var)
            last_aic = cur_aic
        else:
            improvement = False
    return(cur_vars)

back_select(dat.chd, lin_age)
```

In [ ]:
```
def make_cubic_splines(var, knot_locs):
    '''return dataframe for continuous, linear basis'''
    sp_dat = pd.DataFrame(var)
    sp_dat = sm.add_constant(sp_dat)
    sp_dat[var.name+"_sqrd"] = var.apply(np.square)
    sp_dat[var.name+"_cubd"] = var.apply(lambda x: x**3)
    for knot in knot_locs:
        knot_name = var.name + "_" +str(int(np.round(knot)))
        sp_dat[knot_name] = var.apply(lambda x: (x-knot)**3*(x>knot))
    return(sp_dat)

k_locs = knot_placement(dat['age'], 3)
print(k_locs)
cube_age = make_cubic_splines(dat['age'], k_locs)
cube_age.head(10)
```

- running backward stepwise

In [ ]:
```
keep_vars = back_select(dat.chd, cube_age)
results = sm.Logit(dat.chd, cube_age[keep_vars]).fit()
print(results.aic)
results.summary()
```

- Below is a cubic regression spline
- Note that the y-axis is on the Logit scale:

$$\log \frac{P(\text{CHD})}{1 - P(\text{CHD})}$$

In [ ]:
```
coefs = results.params
x_col = pd.DataFrame({'age': np.linspace(
    dat['age'].min(), dat['age'].max(), 1000)})
x_mat = make_cubic_splines(x_col['age'], k_locs)
x_mat = x_mat[keep_vars]
y = x_mat.dot(coefs)
plt.plot(x_col['age'], y, '-b')
plt.xlabel('age (yrs)')
plt.ylabel('Logit P(CHD)');
```

- Below is the same spline function but on the odds (or odds ratio) scale:

$$\frac{P(\text{CHD})}{1 - P(\text{CHD})}$$

In [ ]:
```
y = x_mat.dot(coefs).apply(np.exp)
plt.plot(x_col['age'], y, '-b')
plt.xlabel('age (yrs)')
plt.ylabel('Odds of CHD');
```

- Below is the probability scale
- Note: this can be miss leading because this is not a random sample of the population but a case control study
- Regardless, this scale is very intuitive because it gives an idea of the proportion of the data at each age experiencing CHD

In [ ]:
```
y = results.predict()
x,y = zip(*sorted(zip(dat['age'].tolist(), y)))
plt.plot(x,y)
plt.xlabel('age (yrs)')
plt.ylabel('Probability of CHD');
```

- Note: Logistic regression (before transformation) give log odds ratios
- Odds of event with probability p is $\frac{p}{1-p}$
  - Odds of 1, or (1 to 1 odds) means there is a 50% chance
- Transforming with $\exp$ (including the intercept) give the odds of CHD at each age
- But, remember this dataset was a case-control study, not a general representation of the population so these odds are specific to this dataset

Natural Splines

- Issues with regression spline models:
  - Polynomial functions can be erratic near the boundaries
  - And, for all models, extrapolation, predicting values outside of the domain of x within the data, can easily lead to errors
  - These Problems can be worse with splines
- *Natural cubic spline* requires function to be linear beyond boundary knots
- This frees 4 degrees of freedom compared to cubic spline
- Drawback: potentially greater bias near boundary but this acceptable given that there is less information at boundaries
- Natural cubic splines with K knots is represented by K basis functions

$$N_1(x) = 1, \quad N_2(x) = x, \ldots, \quad N_{k+2}(x) = d_k(x) - d_{K-1}(x)$$

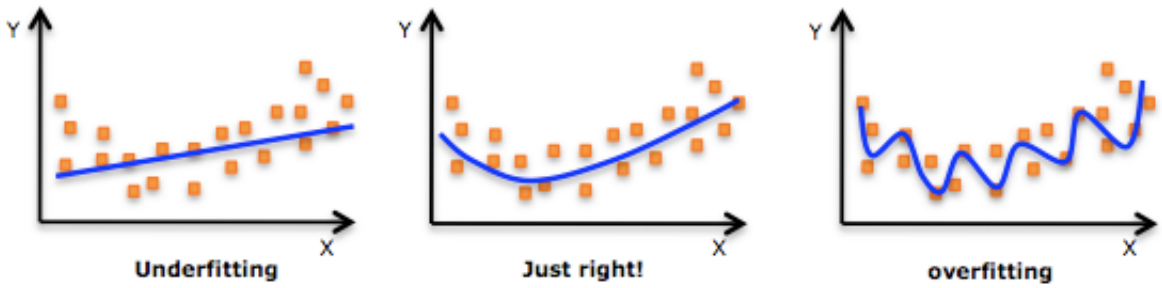
where

$$d_k(x) = \frac{(x-\xi_k)_+^3 - (x-\xi_K)_+^3}{\xi_K - \xi_k}$$

- Note: number of knots = degrees of freedom for natural splines
- Each basis function has zero second and third derivatives
- Benefit: once all input variables have been transformed, can apply standard GLM theory

Fitting Regression Splines

- To fit regression splines, you need to choose the number of knots/basis functions and knot placement
- To determine the number of basis function, we could use AIC or some other fit metric
  - Note: some models (Likelihood ratio test, deviance) require nested model, AIC doesn't
  - What could go wrong with too many basis functions?
- For natural cubic splines, it's typical to place knots at uniform quantiles
  - e.g. with 3 knots, place each at 1st, 2nd, 3rd quartile
- Equal spacing of knots between max and min is sometimes done as well
- Another approach is to use model selection to select how many knots, removing knots that do not contribute to model fit
- This is a classic method, and can work well provided we choose good knots t1, . . . , tp; but in general choosing knots is a tricky business. There is a large literature on knot selection for regression splines via greedy methods like recursive partitioning (from statistical machine learning at Carnegie Mellon)



- Below is a visualization of natural cubic spline basis functions

In [ ]:
```
age = dat['age'].sort_values()
x_dmat = dmatrix("cr(age, 5)-1",
                 {"age": age}, return_type='dataframe')
plt.grid(True)
plt.plot(age, x_dmat)
plt.xlabel("Age (yrs)")
plt.ylabel("Spline Transformation")
plt.title("Natural Cubic Splines with 5 Basis Fuctions");
```

- The image below shows how a different number of knots/degrees of freedom affect the shape of the curve
- Notice that the curve for 3 knots fits better than the linear model using AIC as the fit metric

In [ ]:
```
age_dat = sm.add_constant(dat['age'])
results = sm.Logit(dat.chd, age_dat).fit(disp=0)
print(f'Linear Model AIC: {np.round(results.aic, 2)}')

for num_knots in range(3,8):
    fmla = f"cr(age, {num_knots})"
    x_dmat = dmatrix(fmla, {"age": dat['age']})
    results = sm.Logit(dat.chd, x_dmat).fit(disp=0)
    y = np.exp(np.dot(x_dmat, results.params))
    x,y = zip(*sorted(zip(dat['age'].tolist(), y)))
    lab = f"Knots: {num_knots}, AIC: {np.round(results.aic, 2)}"
    plt.plot(x, y, label=lab);

plt.xlabel('Age (yrs)')
plt.ylabel('Odds of CHD')
plt.legend();
```

- I'm showing the model output below
- In general, the `coef` values are must less useful than the associated graphic shown below the regression results

In [ ]:
```
x_dmat = dmatrix("cr(age, 3)", {"age": dat['age']})
results = sm.Logit(dat.chd, x_dmat).fit(disp=0)
results.summary()
```

In [ ]:
```
y = np.exp(np.dot(x_dmat, results.params))
x,y = zip(*sorted(zip(dat['age'].tolist(), y)))
plt.plot(x, y);
plt.xlabel('Age (yrs)')
plt.ylabel('Odds of CHD');
```

Smoothing Splines

- Choosing knot locations can be tricky
- *Smoothing splines* circumvent this by placing knots at **each** input, $x_1, \ldots, x_n$ while controlling overfitting by **regularization**
  - **Regularization** adds a penalty, λ for model complexity
  - Large values of λ add greater penalty for model complexity, resulting in a simpler model
- Recall from linear regression, we want to minimize residual sum of squares (RSS):

$$\text{RSS}(\beta) = \arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - x^T\beta\right)^2$$

  where the regression function is $f(x) = x^T\beta$
- For smoothing splines, we want to minimize over all continuous functions with continuous first and second derivatives, plus a complexity term:

$$\text{RSS}(f, \lambda) = \arg\min_{f} \sum_{i=1}^{n} \left[y_i - f(x_i)\right]^2 + \lambda \int [f''(t)]^2 dt$$

  where $\lambda \in (0, \infty)$ is a fixed smoothing parameter that is chosen beforehand
- If $\int [f''(t)]^2 dt = 0$, then f is linear
- If $\int [f''(t)]^2 dt$ is large, then f is very wiggly
- $\lambda = 0 \Rightarrow f$ is any (wiggly) function that interpolates all points
- $\lambda = \infty \Rightarrow f$ is the least squares line from standard regression
- So far, this problem seems very difficult because it is not clear how to estimate f, which is assume to be in a *Sobolev function space*
- Remarkably, f, the solution to RSS(f, λ), has a finite-dimensional (tractable) solution:
  - natural cubic spline with knots at each x_i:

$$f(x) = \sum_{j=1}^{n} N_j(x)\beta_j = N(x)\beta = N\beta$$

    where $N_j(x)$ is the jth natural basis function from above and $N(x)$ is the vector-

valued function of all basis functions

- Once the transformation is done, RSS can be re-written:

$$\text{RSS}(\beta, \lambda) = (y - N\beta)^T(y - N\beta) + \lambda\beta^T\Omega_N\beta$$

where $\{N\}_{ij} = N_j(x_i)$ is a matrix (jth transformation of i observation and $\Omega_N = \int N_j''(t)N_k''(t)dt$

- Solution:

$$\hat{\beta} = (N^TN + \lambda\Omega_N)^{-1}N^Ty$$

- Fitted smoothing spline:

$$\begin{aligned}\hat{f}\,(x) &= \sum_{j=1}^{n} N_j(x)\hat{\beta}_j = N\hat{\beta} \\ &= N(N^TN + \lambda\Omega_N)^{-1}N^Ty \\ &= S_\lambda y\end{aligned}$$

- S_λ is called the smoother matrix (similar to the hat matrix for linear regression)

Choosing λ

- Because smoothing splines place a knot at each distinct x value, model degrees of freedom (and overfitting) are controlled by choice of λ
- Smoothing splines have many parameters (the number of distinct x values but because of the penalty, these values are smaller than they would be without the penalty
- Because of this, we cannot talk about degrees of freedom in the same way as prior models, instead we use *effective degrees of freedom*

$$\text{df}_\lambda = \text{trace}(S_\lambda)$$

- Because of this change, it is better to fit the model using a *cross validation method*
- Cross validation shows how well a model fits on data it has never seen/used before
- We measure fit using mean squared error:

$$E[y_i - \hat{f}\,(x_i)]^2$$

- A good fit means that MSE is small

- Cross validation: use many different values of λ and corresponding spline function, f_λ

$$\hat{f} = \arg\min_{f_\lambda} \sum_{i=1}^{n} \left(y_i - f_\lambda(x_i)\right)^2$$

- Leave-one-out cross validation (LOOCV):

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left[y_i - \hat{f}_\lambda^{(-i)}(x_i)\right]^2$$

  where $f^{(-i)}$ is a model that was estimated without using observation i

- $\hat{f}_\lambda^{(-i)}(x_i)$ is fast to compute

$$\hat{f}_\lambda^{(-i)}(x_i) = \frac{\hat{f}_\lambda(x_i) - s_{\lambda,ii} y_i}{1 - s_{\lambda,ii}}$$

  where $s_{\lambda,ii} = (S_\lambda)_{ii}$ is the ii entry in the smoother matrix using all of the observations

- Using all of this, the we want to choose the λ with best fit this way:

$$\lambda^* = \arg\min_{\lambda} \sum_{i=1}^{n} \left(y_i - \hat{f}_\lambda^{(-1)}(x_i)\right)^2 = \arg\min_{\lambda} \sum_{i=1}^{n} \left(\frac{y_i - \hat{f}_\lambda(x_i)}{1 - s_{\lambda,ii}}\right)^2$$

- This is helpful because there is no need to refit the model each time

Logistic Regression

- Here, rather than minimize RSS, we use penalized log-likelihood:

$$\begin{aligned}\ell(f;\lambda) &= \sum_{i=1}^{n} \left[y_i \log p(x_i) + (1 - y_i) \log(1 - p(x_i))\right] - \frac{\lambda}{2} \int [f''(t)]^2 dt \\ &= \sum_{i=1}^{n} \left[y_i f(x_i) + \log(1 + \exp(f(x_i)))\right] - \frac{\lambda}{2} \int [f''(t)]^2 dt\end{aligned}$$

- This model can be optimized using gradient descent

Smoothing Splines in Python and R

- There is no clear smoothing spline function for logistic regression within a standard data science python module
  - `statsmodels` does not (as of now) have an explicit smoothing spline method

  - But, the `alpha` parameter in `statsmodels.gam` is list of penalization weights for each variable according to documentation
  - Might be possible to implement smoothing spline using this
  - `scipy.interpolate.UnivariateSpline` documentation indicate that it is a 1-D smoothing spline function for a continuous outcome; however, the `s`, smoothing factor parameter is used to chooses number of knots
  - It seems unclear how the spline is calculated without looking into the code
  - No clear way to extract smoother matrix to run LOOCV for `UniveriateSpline`
  - `CSAPS` may be a better alternative in python

- In R, `smooth.spline` is very close to the method described above, documentation
  - `smooth.spline` makes it very easy to run cross validation
  - Does not implement logistic regression as shown above
  - Other R packages: `gam`, `mgcv`

In [ ]:
```
from scipy.interpolate import UnivariateSpline
x = np.linspace(-3, 3, 50)
y = np.exp(-x**2) + 0.1 * np.random.randn(50)
plt.plot(x, y, 'yo', ms=5)

spl = UnivariateSpline(x, y)
xs = np.linspace(-3, 3, 1000)

for itr in range(5):
    sm_fact = np.exp(itr-3)
    spl.set_smoothing_factor(sm_fact)
    plt.plot(xs, spl(xs), label=f"s = {np.round(sm_fact, 2)}")
    plt.legend()
```

- The code below is an attempt to get `statsmodels` to run a smoothing spline
- Again, cross validation should be used when choosing λ (alpha in the model)
- The `loocv` method below needs to be validated before professional use
- I would not recommend using this strategy in python until better smoothing spline methods arise
- Smoothing splines are very developed in R

In [ ]:

```
from statsmodels.gam.api import GLMGam

unique_ages = dat.age.nunique()
print(f"Unique Ages: {unique_ages}")

def loocv(results):
    resid = results.resid_response
    #resid = results.resid_deviance
    sii = results.hat_matrix_diag
    return(np.sum(np.square(np.divide(resid, 1-sii))))

for itr in range(4):
    sm_fact = np.exp(3*itr+1)
    fmla = f"chd ~ age"
    smoothing = sm.gam.BSplines(dat['age'], df = unique_ages, degree=3)
    model_smoothing = GLMGam.from_formula(
        fmla, data=dat, alpha = sm_fact, smoother=smoothing,
        family=sm.families.Binomial())
    results = model_smoothing.fit()

    x,y = zip(*sorted(zip(dat['age'].tolist(), results.predict())))
    lab = f"alpha = {np.round(sm_fact, 2)}, \
loocv: {np.round(loocv(results), 3)}"
    plt.plot(x, y, label=lab);

plt.xlabel('Age (yrs)')
plt.ylabel('Probability of CHD');
plt.title('Smoothing Splines')
plt.legend();
```

B-Splines

- B-splines are a little more mathematically complex but computationally very fast because they are generated recursively
- Assume you choose K knots within the domain of x: $\xi_1, \ldots, \xi_K$ and you choose to use an order M spline
- Let ξ_0 and ξ_{K+1} be the boundary knots that to be the scalar domain for the spline function
- Let $\tau_1 \leq \tau_2 \leq \cdots \leq \tau_M \leq \xi_0$
- Let $\tau_{j+M} = \xi_j, j = 1, \ldots, K$
- Let $\xi_{K+1} \leq \tau_{K+M+1} \leq \tau_{K+M+2} \leq \cdots \leq \tau_{K+2M}$
- Denote $B_{i,m}(x)$ to be the ith B-spline basis function of order m for the knot sequence

$\tau, m \leq M$

$$B_{i,1}(x) = \begin{cases} 1 & \text{if } \tau_i \leq x < \tau_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

for $i = 1, \ldots, K + 2M - 1$ and

$$B_{i,m} = \frac{x - \tau_i}{\tau_{i+m-1} - \tau_i} B_{i,m-1}(x) + \frac{\tau_{i+m} - x}{\tau_{i+m} - \tau_{i+1}} B_{i+1,m-1}(x)$$

for $i = 1, \ldots, K + 2M - m$

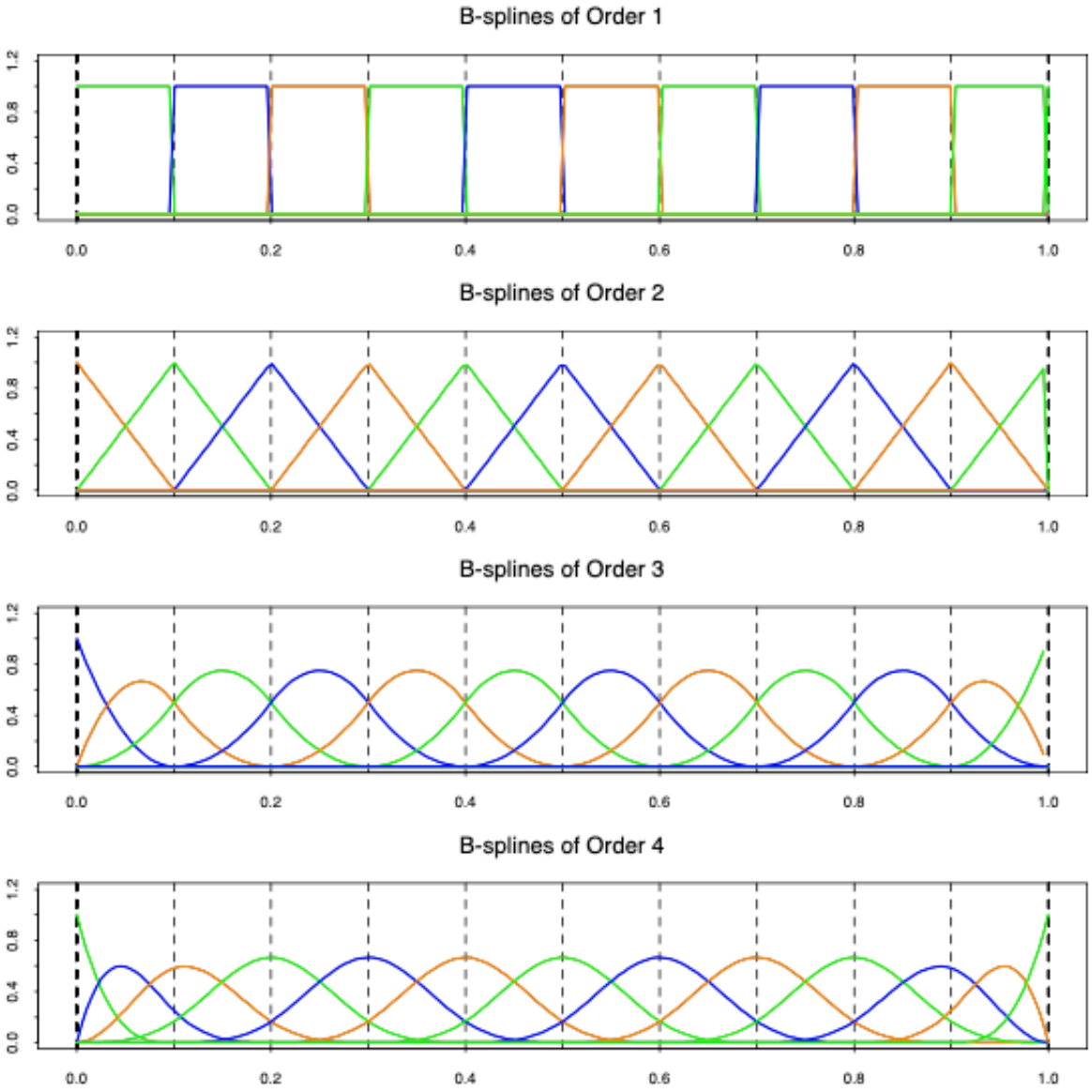


FIGURE 5.20. *The sequence of B-splines up to order four with ten knots evenly spaced from 0 to 1. The B-splines have* local support*; they are nonzero on an interval spanned by $M + 1$ knots.*

Image from Elements of Statistical Learning

B-splines in `statsmodels`

- very easy to implement B-splines in `statsmodels`
- need to specify `df`, degrees of freedom/number of basis functions
- knots placed at quantiles by default but can also be specified

- 3rd order B-splines are most commonly used
  - seems unlikely to ever need higher than 3rd order
- There are multiple ways to run B-splines in `statsmodels`
- Below uses R -style formulas

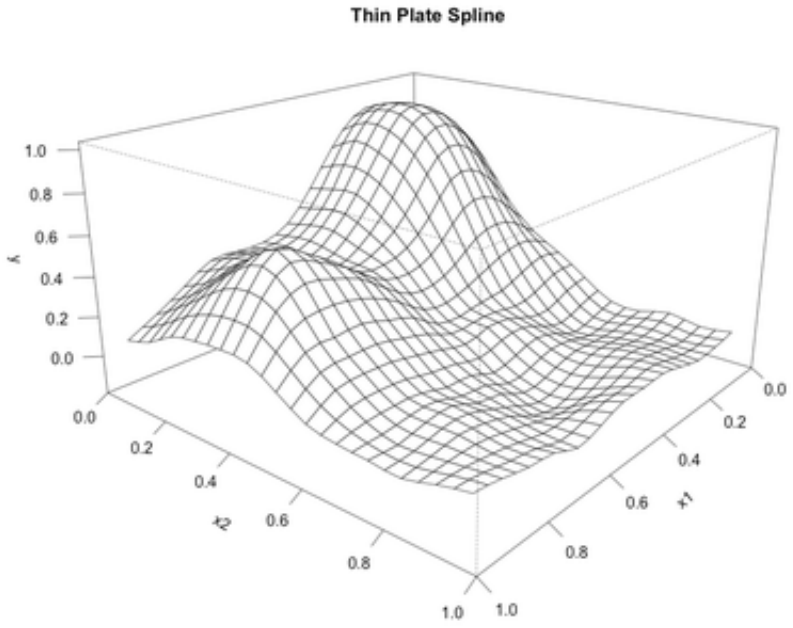
In [ ]:

```
for knots in range(3,8):
    fmla = f"chd ~ bs(age, {knots})"
    modelBS = sm.GLM.from_formula(fmla, data=dat,
                                  family=sm.families.Binomial())
    results = modelBS.fit()
    x,y = zip(*sorted(zip(dat['age'].tolist(), results.predict())))
    lab = f"knots: {knots}, aic: {np.round(results.aic, 2)}"
    plt.plot(x, y, label=lab);

plt.xlabel('Age (yrs)')
plt.ylabel('Probability of CHD');
plt.title('Degree 3 B-Splines')
plt.legend();
```

Interactions

- Interactions terms allow the joint values of two or more variables to be estimated separately
- If one variable is discrete, the interaction term allows for separate estimates each each value of the discrete variable
- Interactions can be included for two continuous variables as well
  - to visualize, must choose discrete values of one variable
- Similar to splines, including interaction terms allow for more model flexibility but uses an additional degree of freedom (parameter to estimate) for each interaction term



In [ ]:

```
fmla = ("chd ~ bs(age, 4)*famhist", "chd ~ age*famhist",
        "chd ~ age+famhist")
num = 500
age = np.linspace(dat['age'].min(), dat['age'].max(), num)

for mod in range(3):
    modelBS = sm.GLM.from_formula(
        fmla[mod], data=dat, family=sm.families.Binomial())
    results = modelBS.fit()
    print(f"{fmla[mod]} AIC = {np.round(results.aic,2)}")
    for famhist in ('Present', 'Absent'):
        y = results.predict(
            pd.DataFrame({'age': age, 'famhist': famhist}))
        if mod==0:
            lab = f"Family History of CHD {famhist}, \
spline interaction model"
        elif mod==1:
            lab = f"Family History of CHD {famhist}, \
linear interaction model"
        else:
            lab = f"Family History of CHD {famhist}, \
linear model, no interaction"
        plt.plot(age, y, label=lab)
    plt.legend()
    plt.xlabel('Age (yrs)')
    plt.ylabel('Probability of CHD')
    plt.title('Spline Interaction with Family History');
```

Model fitting with multiple variables: Generalized Additive Models (GAMs)

- Model structure:

$$g(E[Y]) = \beta_0 + \beta_1 f_1(x_1) + \cdots + \beta_p f_p(x_p)$$

  where $f_j(x_j)$ is a vector valued function (each entry is the output of a basis function) with β_j the corresponding parameter estimates

- There are many different schools of thought on model fitting
  - Choose simplest model with best fit
  - Data are never linear, if there is enough data, more complex models can capture non-linearity
  - In general, most tend to prefer simpler models

- Rule: after selecting between many different models using a dataset, the inference is no longer valid on the training data

- In general, GAM model selection is an area of active research
- Can be done with both smoothing splines and regression splines
- Here, we take a simple, forward-stepwise approach using AIC
- Challenge is that we must decide on smoothing level/number of knots for each variable

Coding with GAMs

- Note on `statsmodels` : The `gam` method that works fine for fitting the model, but prediction was an issue me
- The documentation shows a prediction method but I was not able to get it to work
- Prediction for these models is super important because that is primarily what you use to how the relationship between the variables and outcome using plots
- Recommendation: use `sm.glm.from_formula` or `sm.ols.from_formula` with the `bs` operator to indicate that you want to use a spline
- Example `'chd ~ bs(sbp, 3)'` uses 3 knots (and an intercept), the default is to use degree=3
- `sm.gam.BSplines(sbp, df=4, degree=3)` gives the same spline model as above, `df=4` means 3 knots
- The code below could be improved with a function rather than repeating similar code each time

In [ ]:

```
fmla = 'chd ~ sbp + tobacco + ldl + famhist + obesity + age'

vars = dat.columns.to_list()
vars.remove('famhist') # removing because categorical
vars.remove('chd')  # removing because response variable

model_aic = pd.DataFrame(columns=['Spline', 'DF', 'AIC'])

results = sm.GLM.from_formula(
    fmla, data=dat, family=sm.families.Binomial()).fit()
model_aic = model_aic.append(
        {'Spline': 'None', 'DF': None,
         'AIC': results.aic}, ignore_index=True)

for knots in [4,5,6,7]:
    for var in vars:
        smoothing = sm.gam.BSplines(dat[var], df=knots, degree=3)
        model_smoothing = GLMGam.from_formula(
            fmla, data=dat, smoother=smoothing,
            family=sm.families.Binomial())
        results = model_smoothing.fit()
        model_aic = model_aic.append(
            {'Spline': var, 'DF': knots, 'AIC': results.aic},
            ignore_index=True)

model_aic.sort_values(by='AIC')
```

In [ ]:

```
if 'obesity' in vars: vars.remove('obesity')
model_aic = pd.DataFrame(columns=['Spline', 'DF', 'AIC'])

for knots in [4,5,6,7]:
    for var in vars:
        smoothing = sm.gam.BSplines(
            dat[[var, 'obesity']], df=[knots,4], degree=[3,3])
        model_smoothing = GLMGam.from_formula(
            fmla, data=dat, smoother=smoothing,
            family=sm.families.Binomial())
        results = model_smoothing.fit()
        model_aic = model_aic.append(
            {'Spline': var, 'DF': knots, 'AIC': results.aic},
            ignore_index=True)

model_aic.sort_values(by='AIC')
```

In [ ]:
```
if 'age' in vars: vars.remove('age')
model_aic = pd.DataFrame(columns=['Spline', 'DF', 'AIC'])

for knots in [4,5,6,7]:
    for var in vars:
        smoothing = sm.gam.BSplines(
            dat[[var, 'obesity','age']], df=[knots,4,5], degree=[3,3,3])
        model_smoothing = GLMGam.from_formula(
            fmla, data=dat, smoother=smoothing,
            family=sm.families.Binomial())
        results = model_smoothing.fit()
        model_aic = model_aic.append(
            {'Spline': var, 'DF': knots, 'AIC': results.aic},
            ignore_index=True)

model_aic.sort_values(by='AIC')
```

In [ ]:
```
if 'sbp' in vars: vars.remove('sbp')
model_aic = pd.DataFrame(columns=['Spline', 'DF', 'AIC'])

for knots in [4,5,6,7]:
    for var in vars:
        smoothing = sm.gam.BSplines(
            dat[[var, 'obesity','age','sbp']], df=[knots,4,5,5],
            degree=[3,3,3,3])
        model_smoothing = GLMGam.from_formula(
            fmla, data=dat, smoother=smoothing,
            family=sm.families.Binomial())
        results = model_smoothing.fit()
        model_aic = model_aic.append(
            {'Spline': var, 'DF': knots, 'AIC': results.aic},
            ignore_index=True)

model_aic.sort_values(by='AIC')
```

- seeing if removing variables improves fit

In [ ]:

```
smoothing = sm.gam.BSplines(
    dat[['obesity','age','sbp']], df=[4,5,5], degree=[3,3,3,3])

model_smoothing = GLMGam.from_formula(
    'chd ~ sbp + tobacco + famhist + obesity + age',
    data=dat, smoother=smoothing, family=sm.families.Binomial())
print(f"Removing ldl, aic: {np.round(model_smoothing.fit().aic,2)}")
model_smoothing = GLMGam.from_formula(
    'chd ~ sbp + ldl + famhist + obesity + age',
    data=dat, smoother=smoothing, family=sm.families.Binomial())
print(f"Removing tobacco, aic: {np.round(model_smoothing.fit().aic)}")
model_smoothing = GLMGam.from_formula(
    'chd ~ sbp + tobacco + ldl + obesity + age',
    data=dat, smoother=smoothing, family=sm.families.Binomial())
print(f"Removing famhist, aic: {np.round(model_smoothing.fit().aic,2)}")
```

- removing these makes model fit worse, so we keep them
- the final model for the forward step-wise approach is below
- showing graphics for these is very important because the parameter numbers or very difficult to interpret compared to the spline function

In [ ]:

```
fmla = 'chd ~ sbp + tobacco + ldl + famhist + obesity + age'
smoothing = sm.gam.BSplines(
    dat[['obesity','age','sbp']], df=[4,5,5], degree=[3,3,3])
fsw_model = GLMGam.from_formula(
    fmla, data=dat, smoother=smoothing, family=sm.families.Binomial())
fsw_results = fsw_model.fit()
print(f"GLMGam, aic: {np.round(fsw_results.aic,2)}")

fmla = 'chd~bs(sbp,4)+tobacco+ldl+famhist+bs(obesity,3)+bs(age,4)'
fsw_model = sm.GLM.from_formula(
    fmla, data=dat, family=sm.families.Binomial())
fsw_results = fsw_model.fit()
print(f"GLM, aic: {np.round(fsw_results.aic,2)}")


vars = ['sbp', 'tobacco', 'ldl', 'famhist', 'obesity', 'age']

def get_median_mode(vars, num):
    d_mat = {}
    for var in vars:
        if np.issubdtype(dat[var].dtype, np.number):
            d_mat[var] = dat[var].median()
        else:
            d_mat[var] = dat[var].mode()
    d_mat = pd.DataFrame(d_mat)
    return(sm.add_constant(pd.concat([d_mat]*num)))

def plot_spline(var):
    if np.issubdtype(dat[var].dtype, np.number):
        num = 500
        d_mat = get_median_mode(dat, num)
        d_mat[var] = np.linspace(dat[var].min(), dat[var].max(), num)
        y = fsw_results.predict(exog = d_mat)
        plt.plot(d_mat[var], y)
    else:
        d_mat = get_median_mode(dat, dat[var].nunique())
        d_mat[var] = dat[var].unique()
        y = fsw_results.predict(exog = d_mat)
        plt.bar(d_mat[var], y)
    plt.xlabel(var)
    plt.ylabel('Probability of CHD')
    plt.title(f'Probability of CHD by {var} for median observation')

for var in vars:
    plt.figure()
    plot_spline(var)
```

- How do the estimates above compare to the original linear logistic model?
- Results explanation example for obesity: Figure [figure number] provides the model expected probability of CHD from the data by obesity (BMI) for median values of age, LDL, and all other variables used in the model. The U-shape of the curve indicates that those with the least and greatest BMI, at the time the survey was taken, had the greatest risk of CHD while those with a BMI closer to the center tended to have a lower risk of CHD. This finding is counterintuitive and may be due, in part, to the fact that for this case-control study, BMI was taken at the time of the survey, rather than at the time of CHD. The lower BMI may actually have been a result of CHD, as CHD patients are encouraged to lower their BMI. This U-shaped relationship is missed when using a model that assumed linear relationships, see model [model number] in appendix.
- It is not necessary to explain each plot
  - explaining the plots with different shapes is probably sufficient because the reader can infer meaning for similar plots
- Presentation tip: use a grid to present these figures
  - This way the reader can take in more information at once
  - If x-axes were on the same scale, I would have plotted them together
  - In general, you want to hit a balance between too much and too little graphical information
- These effects at first may come as a surprise, but an explanation lies in the nature of the retrospective data. These measurements were made sometime after the patients suffered a heart attack, and in many cases they had already benefited from a healthier diet and lifestyle, hence the apparent increase in risk at low values for obesity and sbp. (from ESL)
- How to make a single prediction below

In [ ]:
```
value =  {'sbp': 134, 'tobacco': 2, 'ldl': 4.34,
     'famhist': 'Absent', 'obesity': 25.805, 'age': 45}
fsw_results.predict(value)
```

In [ ]:
```
fsw_results.summary()
```

- Looking for interactions
- Below we look for interactions with `famhist`
- You can also look for interactions with spline basis functions as well
- these are sometimes called *tensor products*

In [ ]:
```
fmla_list = [
    'chd~bs(sbp,4)+tobacco+ldl+famhist+bs(obesity,3)+bs(age,4)',
    'chd~bs(sbp,4)*famhist+tobacco+ldl+bs(obesity,3)+bs(age,4)',
    'chd~bs(sbp,4)+tobacco*famhist+ldl+bs(obesity,3)+bs(age,4)',
    'chd~bs(sbp,4)+tobacco+ldl*famhist+bs(obesity,3)+bs(age,4)',
    'chd~bs(sbp,4)+tobacco+ldl+bs(obesity,3)*famhist+bs(age,4)',
    'chd~bs(sbp,4)+tobacco+ldl+bs(obesity,3)+bs(age,4)*famhist'
]
for fmla in fmla_list:
    fsw_model = sm.GLM.from_formula(
        fmla, data=dat, family=sm.families.Binomial())
    fsw_results = fsw_model.fit()
    print(f"{fmla}, aic: {np.round(fsw_results.aic,2)}")
```

In [ ]:
```
fmla = 'chd~bs(sbp,4)+tobacco+ldl*famhist+bs(obesity,3)+bs(age,4)'
fsw_model = sm.GLM.from_formula(
        fmla, data=dat, family=sm.families.Binomial())
fsw_results = fsw_model.fit()

num = 500
var = 'ldl'
for famhist in ['Present', 'Absent']:
    d_mat = get_median_mode(dat, num)
    d_mat['famhist'] = famhist
    d_mat[var] = np.linspace(dat[var].min(), dat[var].max(), num)
    y = fsw_results.predict(exog = d_mat)
    plt.plot(d_mat[var], y, label = f'Family History of CHD {famhist}')
plt.xlabel('LDL (mmol/L)')
plt.ylabel('Probability of CHD')
plt.title(f'Probability of CHD by {var} for Median Observation')
plt.legend();
```

Smoothing Spline GAMs

- This is an easier approach to generalized additive model selection because a machine can quickly evaluate many penalty hyper-parameters then choose the best model based on some fit criteria such as LOOCV, CV, or other criteria like restricted maximum likelihood (REML)
- Again, for GAMs,

$$g(E[Y|X=x]) = \sum_{j=1}^{p} f_j(x_j)$$

- Smoothing splines optimize each f_j with a penalty

$$\hat{f}_1, \ldots, \hat{f}_p = \arg\min_{f_1,\ldots,f_p} \ell(y; f_1, \ldots, f_p) + \sum_{j=1}^{p} \lambda_j \int [f_j''(t)]^2 dt$$

  where ℓ is log-likelihood

- In this optimization, the strongest penalty ($\lambda_j = \infty$) results with f_j being a linear function
- The penalty can be modified to determine remove a variable entirely for the model if it does not contribute to better model fit, paper
- `pygam` is a smoothing spline approach (uses regularization)
- Below is my attempt to use smoothing spline in python with pygam, unfortunately, I didn't get it working
- For now, I would recommend using R for smoothing spline GAM selection
- More generally, it is much faster (automatic) to fit GAMs with smoothing splines rather than the stepwise approach with regression splines
- This article describes GAMs at a high level and shows how to implement in R with mgcv

In [ ]:

```
import pygam
X = dat[['sbp', 'tobacco', 'ldl', 'famhist', 'obesity', 'age']].copy()
X['famhist'] = dat['famhist'].apply(lambda x: (x == 'Present')*1)

gam = pygam.GAM(pygam.s(0)+pygam.s(1)+pygam.s(2)+pygam.f(3)+
                pygam.s(4)+pygam.s(5),
                distribution='binomial', link='log').fit(X, dat['chd'])
```

Splines in practice

- Use a model that you understand well enough
  - this is very important when speaking with a client, that you are knowledgeable
- Use a model that easy enough to implement (code)
  - This may change depending on the language you work in
  - for example, I would probably only use a smoothing spline in R
- Overfitting should always be a concern with splines
- Cross validation is the gold standard to avoid over/underfitting but splits data
- AIC is a good runner up in likelihood-based models with easy parameter counts, uses all data
- Splines can be used for interpretation and prediction
  - if accurate prediction is important, cross validation is better than AIC
  - AIC uses the data twice, once to fit and again to evaluate
  - Cross validation avoids this
- Present splines models with graphics
  - Be aware that spline plots choose a fixed value of each model parameter for a spline curve
  - You may want to present more than one value to illustrate how a curve will shift upward or downward depending on the values chosen

Homework Grading Rubric

- Professional
  - language, grammar
  - tables well-formatter, reasonable number
  - helpful, interesting graphics
  - uses headings
  - meets client's needs
  - could use as job interview material
- Intro
  - First sentence is easy to understand and address the subject matter topic

    - Problem and purpose of report is clearly stated
    - Does not use statistical jargon
    - (sometimes) summarize high level findings
- Methods
    - Connects problem, data, and modeling
    - Uses appropriate level of detail for audience
    - Describe statistical model
    - Describe model selection approach
    - shows understanding of possible data nuance (eg including an outcome in a model or repeated information)
- Results
    - Includes a description of the data
    - Describes variables
        - Most of the time, this will require a table
        - For data with a lot of variables, you may need to put table in appendix or skip entirely
    - Variables are explained when initially presented
    - Explains model/parameters in terms of the subject matter
    - Includes a reasonable number of graphics
        - should be well chosen
        - axes must be labels
        - graphic should contain an appropriate amount of information
        - clearly related to topic
        - has a description
    - Modeling
        - presented in a reasonable way given the model (either table or with graphics
        - appropriate for client's needs
        - well explained (eg GLM log parameters exponentiated)
        - Reasonable overall fit with consideration of complexity
- Conclusion
    - Discusses actual and potential limitation
    - Connects results with overall purpose
- Code appendix
    - does not show unhelpful computer output (especially very long output)
    - uses sentences to describe what does is doing and sometimes why
    - Gives granular description of data with tables and plots (these do not need to be perfectly formatted)
    - Looks for issues with data, outliers, influential points etc

- Includes model diagnostics
- If you are proving usable code, always include documentation with an example of how to use it

Non-parametric Regression

- Assume $(X, Y) \in \mathbb{R}^d \times \mathbb{R}$ is a random vector
- Regression function

$$f_0(x) = \mathbb{E}[Y|X = x]$$

- Goal: estimate f_0 from a model $\hat{f}$ using observation $(X_i, Y_i) \in \mathbb{R}^d \times \mathbb{R}$ for $i = 1, \ldots, n$
- If the sample, $(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$ for $i = 1, \ldots, n$, is identically, independently distributed (i.i.d.), we generally write

$$y_i = f_0(x_i) + \epsilon_i$$

  where ϵ_i is random error with $\mathbb{E}[\epsilon_i] = 0$ and ϵ_i is independent of x_i
- In most non-parametric regression settings, it is common to assume that f_0 is more complex than a linear function (dot product) of x_i and parameters β
- But, f_0 is almost always assumed to be in some class of functions - For example, we could assume that f_0 is an r-degree polynomial or that f_0 is any function where all kth-order partial derivatives are bounded by $L < \infty$
- Fit is generally quantified with the L_2 function norm $\| \cdot \|_2$

$$\|\hat{f} - f_0\|_2^2 = \mathbb{E}\left[(\hat{f}(x) - f_0(x))^2\right] = \int [\hat{f}(X) - f_0(X)]^2 p(x) dx$$

- For a random sample, $(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$ for $i = 1, \ldots, n$, this translates to mean squared error (MSE)

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} [\hat{f}(x_i) - f_0(x_i)]^2 = \frac{1}{n} \sum_{i=1}^{n} [\hat{f}(x_i) - y_i]^2$$

Notes based on Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. The elements of statistical learning. Vol. 1. No. 10. New York: Springer series in statistics, 2001.