stats_ch15_regression

August 6, 2024

1 Modern statistics: Intuition, Math, Python, R

1.1 Mike X Cohen (sincxpress.com)

1.1.1 https://www.amazon.com/dp/B0CQRGWGLY

Code for Chapter 15 (regression)

---

2 About this code file:

2.0.1 This notebook will reproduce most of the figures in this chapter (some figures were made in Inkscape), and illustrate the statistical concepts explained in the text. The point of providing the code is not just for you to recreate the figures, but for you to modify, adapt, explore, and experiment with the code.

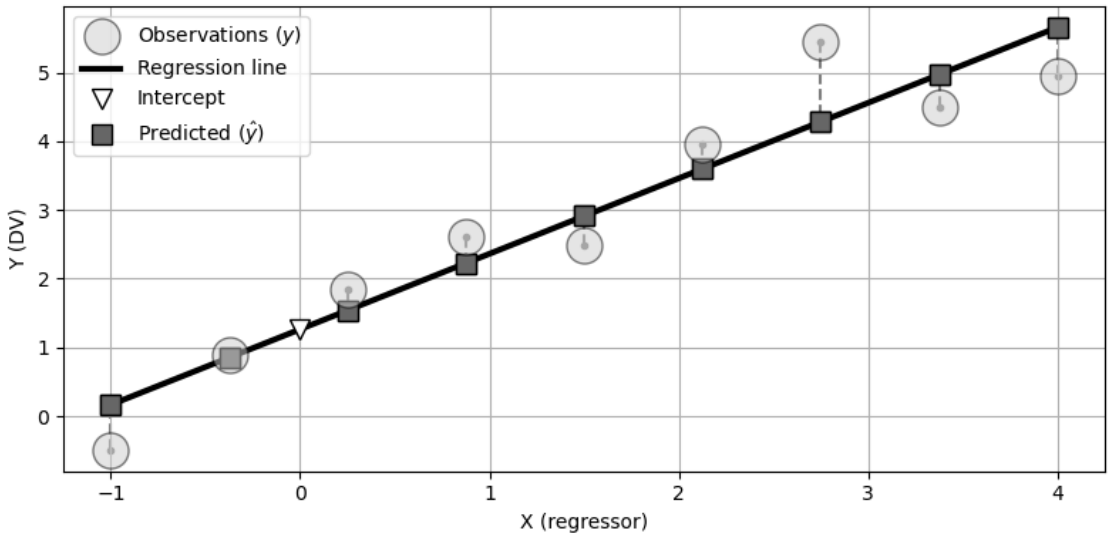
2.0.2 Solutions to all exercises are at the bottom of the notebook.

This code was written in google-colab. The notebook may require some modifications if you use a different IDE.

[3]:
```
# import libraries and define global settings
import numpy as np
import pandas as pd
import seaborn as sns
import scipy.stats as stats

# new: for running regression models
import statsmodels.api as sm

import matplotlib.pyplot as plt

# define global figure properties used for publication
import matplotlib_inline.backend_inline
```

3 Figure 15.1: Geometric view of regression

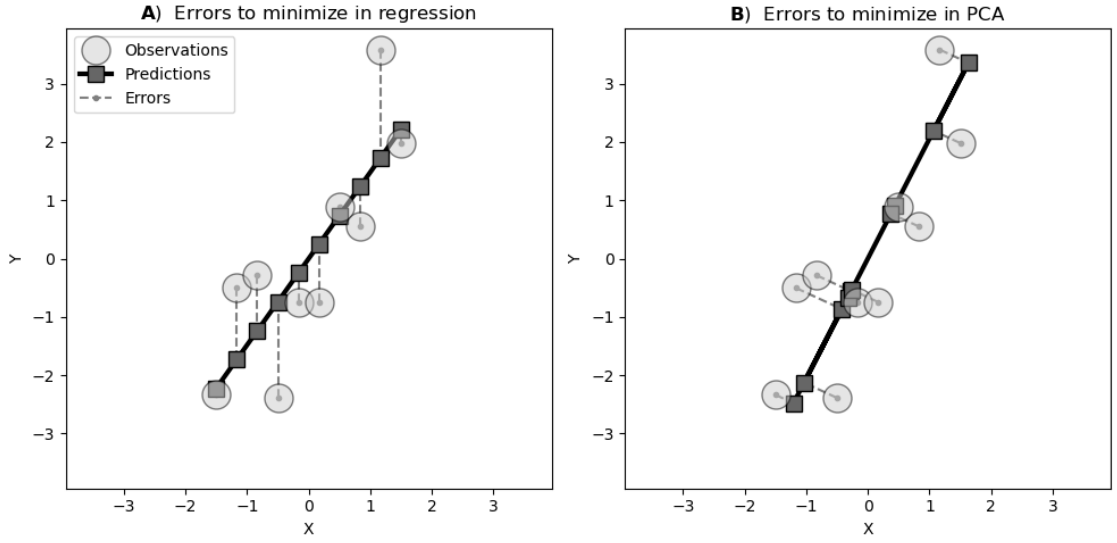
[3]:
```
# some data
N = 9
x = np.linspace(-1,4,N)
y = 1 + x + np.random.randn(N)

# get GLM predictions
mdl = sm.OLS(y,sm.add_constant(x)).fit()
yHat = mdl.predict(sm.add_constant(x))

# plt
_,axs = plt.subplots(1,1,figsize=(8,4))

# plot the data
axs.plot(x,y,'ko',markerfacecolor=(.8,.8,.8),markersize=18,zorder=10,alpha=.
 ↪5,label='Observations ($y$)')
axs.set(xlabel='X (regressor)',ylabel='Y (DV)')

# plot the regression line
axs.plot(x,yHat,'k-',linewidth=3,label='Regression line')

# plot the intercept
intpnt = mdl.predict([1,0])
axs.
 ↪plot(0,intpnt,'kv',markersize=10,markerfacecolor='w',label='Intercept',zorder=10)

# data-point-specific projection lines
for i in range(N):
  axs.plot([x[i],x[i]],[y[i],yHat[i]],'--.',color='gray',zorder=-4)
  axs.plot([x[i],x[i]],[yHat[i],yHat[i]],'ks',markersize=10,markerfacecolor=(.4,.
 ↪4,.4),label=r'Predicted ($\hat{y}$)')

# final adjustments
labh,labels = axs.get_legend_handles_labels() # to prevent redundant 'Predicted'␣
 ↪labels
axs.legend(labh[:4],labels[:4]) # only the first four (unique) legends
axs.grid()
plt.tight_layout()
#plt.savefig('reg_picOfReg.png')
plt.show()
```



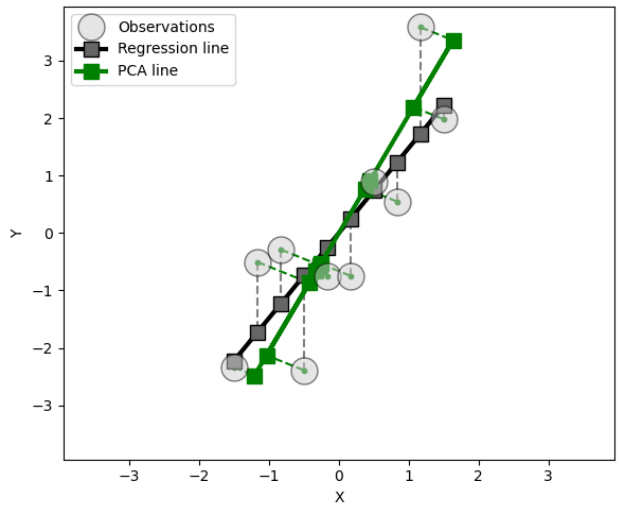
4 Figure 15.2: Regression vs. PCA

[4]:
```
# some data
N = 10
x = np.linspace(-1.5,1.5,N)
y = x + np.random.randn(N)

# mean-center variables
y -= np.mean(y)
maxval = np.max(np.abs(y))*1.1 # for axis scaling; PCA projections look␣
 ↪orthogonal in square axes

# get GLM predictions
mdl = sm.OLS(y,sm.add_constant(x)).fit()
yHat = mdl.predict(sm.add_constant(x))

# compute PCA
data = np.vstack((x,y)).T
C = np.cov(data.T)
evals,evecs = np.linalg.eig(C)
PC = evecs[:,np.argmax(evals)]

# projection points
pcaX = np.zeros(N)
pcaY = np.zeros(N)
```

```
## plot
_,axs = plt.subplots(1,2,figsize=(10,5))

# plot the data
for a in axs:
  a.plot(x,y,'ko',markerfacecolor=(.8,.8,.8),markersize=18,zorder=10,alpha=.5)
  a.set(xlabel='X',ylabel='Y',xlim=[-maxval,maxval],ylim=[-maxval,maxval])

# plot the regression line
axs[0].plot(x,yHat,'ks-',linewidth=3,markersize=10,markerfacecolor=(.4,.4,.4))

# data-point-specific projection lines
for i in range(N):
  # regression is the projection onto the best fit line, holding 'x' constant
  axs[0].plot([x[i],x[i]],[y[i],yHat[i]],'--.',color='gray',zorder=-4)
  # compute and plot the PCA projection lines
  pcaX[i],pcaY[i] = data[i,:]@PC*PC
  axs[1].plot([x[i],pcaX[i]],[y[i],pcaY[i]],'--.',color='gray',zorder=-4)

# now plot the PCA line
axs[1].plot(pcaX,pcaY,'ks-',linewidth=3,markersize=10,markerfacecolor=(.4,.4,.4))

# final adjustments
axs[0].set_title(r'$\bf{A}$)  Errors to minimize in regression')
axs[0].legend(['Observations','Predictions','Errors'])
axs[1].set_title(r'$\bf{B}$)  Errors to minimize in PCA')

plt.tight_layout()
#plt.savefig('reg_regVpca.png')
plt.show()
```

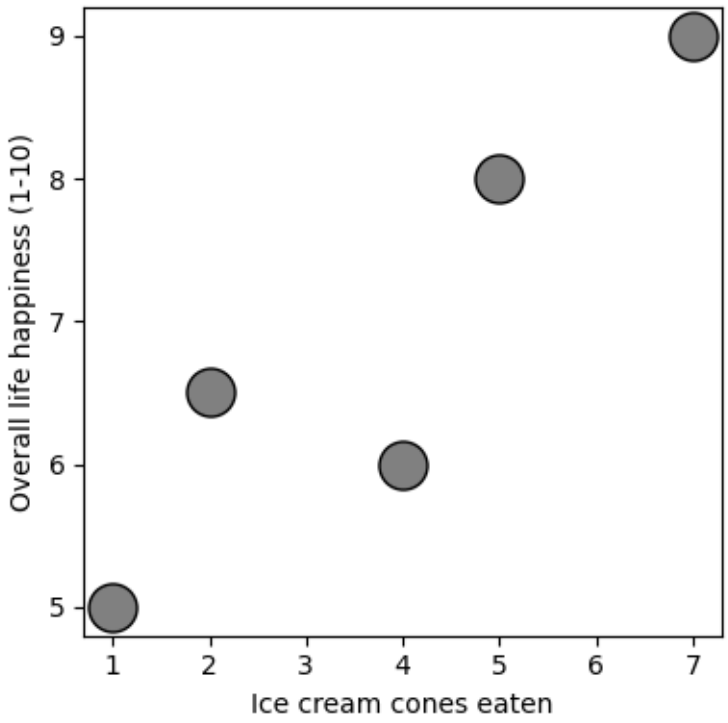


[5]:
```
# All in one plot (a bit confusing to look at...)
_,axs = plt.subplots(1,1,figsize=(6,5))

# plot the data
axs.plot(x,y,'ko',markerfacecolor=(.8,.8,.8),markersize=18,zorder=10,alpha=.
 ↪5,label='Observations')
axs.set(xlabel='X',ylabel='Y',xlim=[-maxval,maxval],ylim=[-maxval,maxval])

# plot the regression line
axs.plot(x,yHat,'ks-',linewidth=3,markersize=10,markerfacecolor=(.4,.4,.
 ↪4),label='Regression line')

# data-point-specific projection lines
for i in range(N):
  # regression is the projection onto the best fit line, holding 'x' constant
  axs.plot([x[i],x[i]],[y[i],yHat[i]],'--.',color='gray',zorder=-4)
  # compute and plot the PCA projection lines
  pcaX[i],pcaY[i] = data[i,:]@PC*PC
  axs.plot([x[i],pcaX[i]],[y[i],pcaY[i]],'--.',color='green',zorder=-4)

# now plot the PCA line
axs.plot(pcaX,pcaY,'gs-',linewidth=3,markersize=10,label='PCA line')

# final adjustments
axs.legend()
plt.tight_layout()
plt.show()
```



5 Figure 15.3: Joy and ice cream

[7]:
```
# the data
icecream  = np.array([ 1, 2, 4, 5, 7 ])
happiness = np.array([ 5, 6.5, 6, 8, 9 ])

# the plot
plt.figure(figsize=(4,4))
plt.plot(icecream,happiness,'ko',markerfacecolor='gray',markersize=18)
plt.xlabel('Ice cream cones eaten')
plt.ylabel('Overall life happiness (1-10)')
plt.yticks(range(5,10))

plt.tight_layout()
#plt.savefig('reg_icecreamjoy.png')
plt.show()
```



[8]:
```
### run the regression

# organize the IVs into a design matrix
designMatrix = np.vstack((
    np.ones(5,),  # intercept
    icecream      # number of ice cream cones eaten
```

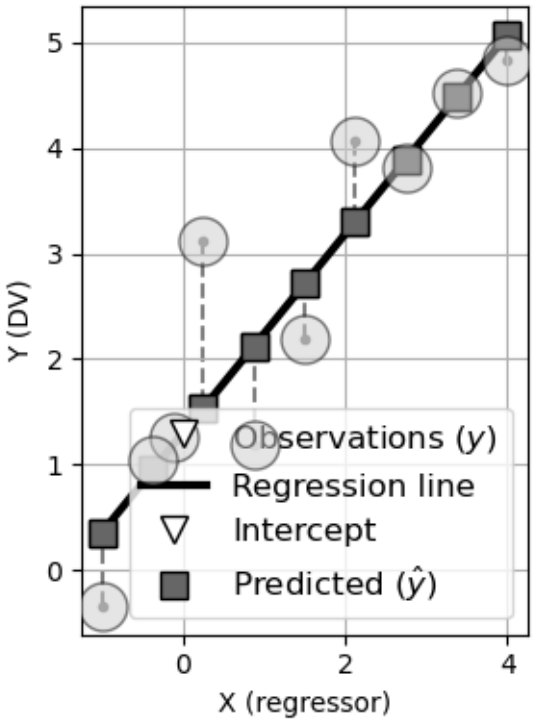
```
    )).T

# list of labels for model output
IVnames = ['Intercept','Ice Cream']

# evaluate the regression model (endog=DV, exog=IVs)
regResults = sm.OLS(happiness,designMatrix).fit()

# and print a summary of the results
t = regResults.summary(xname=IVnames,yname='Happiness')
t.as_text
```

```
C:\Users\user\anaconda3\Lib\site-packages\statsmodels\stats\stattools.py:74:
ValueWarning: omni_normtest is not valid with less than 8 observations; 5
samples were given.
  warn("omni_normtest is not valid with less than 8 observations; %i "
```

[8]:
```
<bound method Summary.as_text of <class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:              Happiness   R-squared:                       0.831
Model:                            OLS   Adj. R-squared:                  0.774
Method:                 Least Squares   F-statistic:                     14.73
Date:                Tue, 06 Aug 2024   Prob (F-statistic):             0.0312
Time:                        01:25:35   Log-Likelihood:                -4.4354
No. Observations:                   5   AIC:                             12.87
Df Residuals:                       3   BIC:                             12.09
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      4.5833      0.692      6.620      0.007       2.380       6.787
Ice Cream      0.6096      0.159      3.838      0.031       0.104       1.115
==============================================================================
Omnibus:                        nan   Durbin-Watson:                   3.320
Prob(Omnibus):                  nan   Jarque-Bera (JB):                0.490
Skew:                        -0.673   Prob(JB):                        0.783
Kurtosis:                     2.262   Cond. No.                         9.26
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
""">
```

6 Figure 15.4: Reminder of the geometry of regression

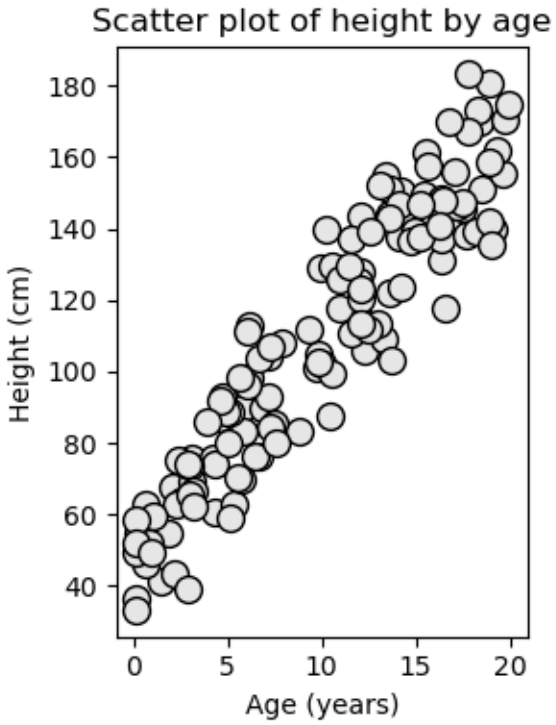
[10]:
```
# some data
N = 9
x = np.linspace(-1,4,N)
y = 1 + x + np.random.randn(N)

# get GLM predictions
mdl = sm.OLS(y,sm.add_constant(x)).fit()
yHat = mdl.predict(sm.add_constant(x))

# plt
_,axs = plt.subplots(1,1,figsize=(3,4))

# plot the data
axs.plot(x,y,'ko',markerfacecolor=(.8,.8,.8),markersize=18,zorder=10,alpha=.
 ↪5,label='Observations ($y$)')
axs.set(xlabel='X (regressor)',ylabel='Y (DV)')

# plot the regression line
axs.plot(x,yHat,'k-',linewidth=3,label='Regression line')

# plot the intercept
intpnt = mdl.predict([1,0])
axs.
 ↪plot(0,intpnt,'kv',markersize=10,markerfacecolor='w',label='Intercept',zorder=10)


# data-point-specific projection lines
for i in range(N):
  axs.plot([x[i],x[i]],[y[i],yHat[i]],'--.',color='gray',zorder=-4)
  axs.plot([x[i],x[i]],[yHat[i],yHat[i]],'ks',markersize=10,markerfacecolor=(.4,.
 ↪4,.4),label=r'Predicted ($\hat{y}$)')

# final adjustments
labh,labels = axs.get_legend_handles_labels() # to prevent redundant 'Predicted'␣
 ↪labels
axs.legend(labh[:4],labels[:4],fontsize=12) # only the first four (unique)␣
 ↪legends
axs.grid()
plt.tight_layout()
#plt.savefig('reg_picOfReg_redux.png')
plt.show()
```



7 Figure 15.6: Simulating regression data: example 1

[12]:
```
# coefficients for linking the IV to the DV
B0 = 50  # intercept in cm
B1 =  6  # coefficient for change in age, also in cm

# number of observations
N = 135

# the independent variable
age = np.random.uniform(0,20,N)

# and the noise
noise = np.random.normal(0,15,N)

# and now put it together to simulate the data
height = B0 + B1*age + noise

# visualization
plt.figure(figsize=(3,4))
```

```
plt.plot(age,height,'ko',markerfacecolor=(.9,.9,.9),markersize=10)
plt.xlabel('Age (years)')
plt.ylabel('Height (cm)')
plt.title('Scatter plot of height by age',loc='center')

plt.tight_layout()
#plt.savefig('reg_example1data.png')
plt.show()
```



[13]:
```
### run the regression

# organize the IVs into a design matrix
designMatrix = np.vstack((
    np.ones(N,),  # intercept
    age           # age IV
    )).T

# list of labels for model output
IVnames = ['Intercept','Age']

# evaluate the regression model (endog=DV, exog=IVs)
regResults = sm.OLS(endog=height, exog=designMatrix).fit()
```

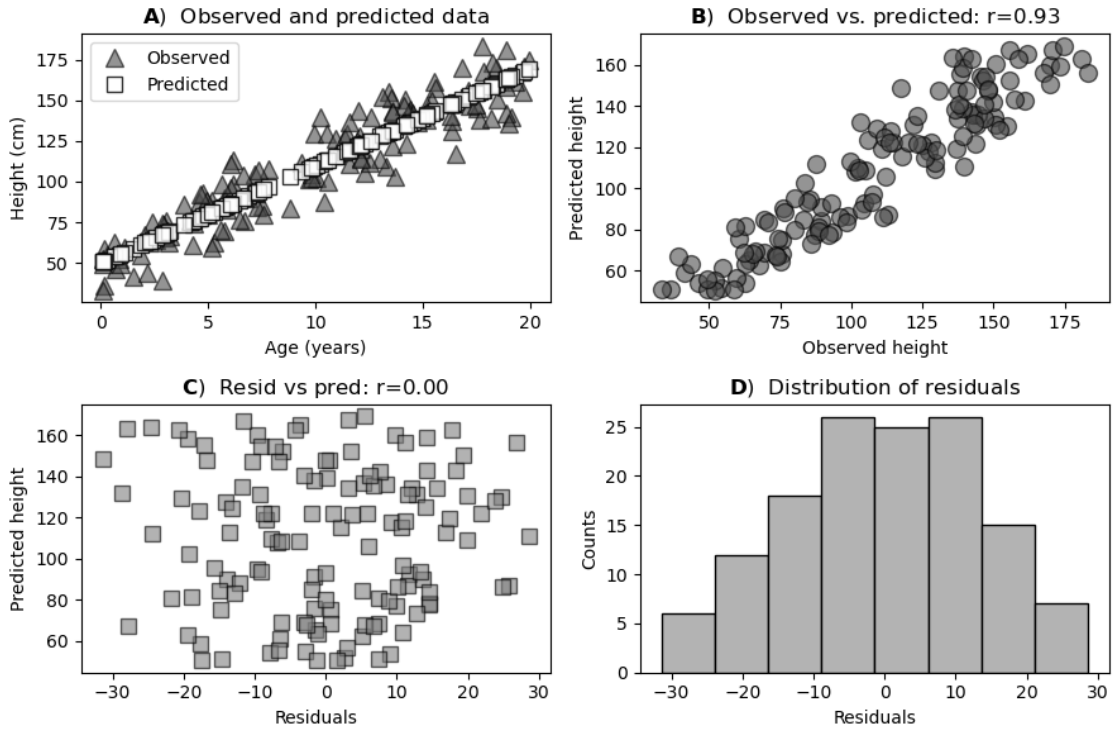
```
# and print a summary of the results
t = regResults.summary(xname=IVnames, yname='Height')
t.as_text
```

[13]:
```
<bound method Summary.as_text of <class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Height   R-squared:                       0.874
Model:                            OLS   Adj. R-squared:                  0.873
Method:                 Least Squares   F-statistic:                     918.9
Date:                Tue, 06 Aug 2024   Prob (F-statistic):           1.39e-61
Time:                        01:27:02   Log-Likelihood:                -540.95
No. Observations:                 135   AIC:                             1086.
Df Residuals:                     133   BIC:                             1092.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     50.0491      2.274     22.014      0.000      45.552      54.546
Age            5.9635      0.197     30.314      0.000       5.574       6.353
==============================================================================
Omnibus:                        3.649   Durbin-Watson:                   2.054
Prob(Omnibus):                  0.161   Jarque-Bera (JB):                2.384
Skew:                          -0.122   Prob(JB):                        0.304
Kurtosis:                       2.396   Cond. No.                         22.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
""">
```

8 Figure 15.7: Visualizing the regression data

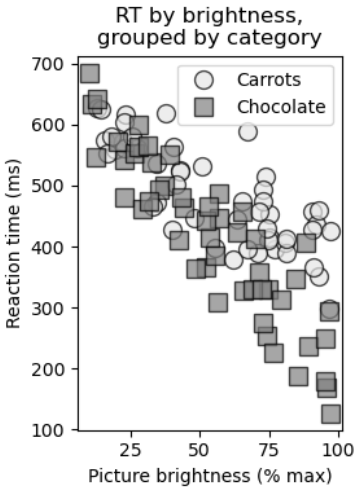
[15]:
```
# plot the predicted data
yHat = regResults.predict()
resid = regResults.resid

_,axs = plt.subplots(2,2,figsize=(9,6))

axs[0,0].plot(age,height,'k^',markerfacecolor=(.3,.3,.3),alpha=.
 ↪6,markersize=10,label='Observed')
axs[0,0].plot(age,yHat,'ks',markerfacecolor='w',alpha=.
 ↪8,markersize=8,label='Predicted')
```

```
axs[0,0].set(xlabel='Age (years)',ylabel='Height (cm)')
axs[0,0].set_title(fr'$\bf{{A}}$)  Observed and predicted data')
axs[0,0].legend()

axs[0,1].plot(height,yHat,'ko',markerfacecolor=(.3,.3,.3),alpha=.6,markersize=10)
axs[0,1].set(xlabel='Observed height',ylabel='Predicted height')
axs[0,1].set_title(fr'$\bf{{B}}$)  Observed vs. predicted: r={np.
 ↪corrcoef(height,yHat)[0,1]:.2f}')

axs[1,0].plot(resid,yHat,'ks',markerfacecolor=(.5,.5,.5),alpha=.6,markersize=8)
axs[1,0].set(xlabel='Residuals',ylabel='Predicted height')
axs[1,0].set_title(fr'$\bf{{C}}$)  Resid vs pred: r={np.
 ↪corrcoef(resid,yHat)[0,1]:.2f}')

axs[1,1].hist(resid,bins='fd',edgecolor='k',facecolor=(.7,.7,.7))
axs[1,1].set(xlabel='Residuals',ylabel='Counts')
axs[1,1].set_title(fr'$\bf{{D}}$)  Distribution of residuals')

plt.tight_layout()
#plt.savefig('reg_example1res.png')
plt.show()
```



9 Figure 15.8: Simulating regression data: example 2

[16]:
```
# create coefficients for linking the IV to the DV
B0 = 600  # intercept
B1 = -2   # coefficient for brightness manipulation
B2 = 60   # coefficient for experiment condition
B3 = -2.5 # coefficient for interaction term
# number of observations
N = 100
# generate independent variables
brightness = np.random.uniform(10,100,N) # continuous IV
category = (np.linspace(0,1,N)>.5)+0 # binary IV

# noise
noise = np.random.normal(0,50,N)
# generate the data according to the model
RT = B0 + B1*brightness + B2*category + B3*(brightness*category) + noise
# visualization
plt.figure(figsize=(3,4))
plt.plot(brightness[category==0],RT[category==0],'ko',markerfacecolor=(.9,.9,.
 ↪9),alpha=.7,markersize=10,label='Carrots')
plt.plot(brightness[category==1],RT[category==1],'ks',markerfacecolor=(.5,.5,.
 ↪5),alpha=.7,markersize=10,label='Chocolate')
plt.xlabel('Picture brightness (% max)')
plt.ylabel('Reaction time (ms)')
plt.legend()
plt.title(f'RT by brightness,\ngrouped by category',loc='center')
plt.tight_layout()
#plt.savefig('reg_example2data.png')
plt.show()
```



[17]:
```
# regression model using a dataframe

# construct the design matrix as a dataframe
df = pd.DataFrame({
    'Brightness' : brightness,
    'Category'   : category,
    #'Interaction': brightness * category # uncomment to include interaction term
})
# add an intercept term (sm calls it "constant")
X = sm.add_constant(df)

# inspect the design matrix:
X # (const = intercept)
```

[17]:

	const	Brightness	Category
0	1.0	75.010672	0
1	1.0	90.559459	0
2	1.0	34.324319	0
3	1.0	55.672400	0
4	1.0	76.806230	0
..	...	...	...
95	1.0	95.248105	1
96	1.0	12.738057	1
97	1.0	28.060689	1
98	1.0	95.211412	1
99	1.0	31.084153	1

[100 rows x 3 columns]

[18]:
```
# fit the model (note the default positions of endog= and exog=)
model = sm.OLS(RT,X).fit()

# show the regression summary (using rich-formatted text instead of plain text␣
 ↪as in the previous example)
model.summary()
```

[18]:

Dep. Variable:	y	**R-squared:**	0.731
Model:	OLS	**Adj. R-squared:**	0.725
Method:	Least Squares	**F-statistic:**	131.7
Date:	Tue, 06 Aug 2024	**Prob (F-statistic):**	2.26e-28
Time:	01:30:50	**Log-Likelihood:**	-551.03
No. Observations:	100	**AIC:**	1108.
Df Residuals:	97	**BIC:**	1116.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P> |t|	[0.025	0.975]
const	680.8781	15.595	43.661	0.000	649.927	711.829
Brightness	-3.5375	0.233	-15.174	0.000	-4.000	-3.075
Category	-74.4924	12.151	-6.130	0.000	-98.609	-50.376

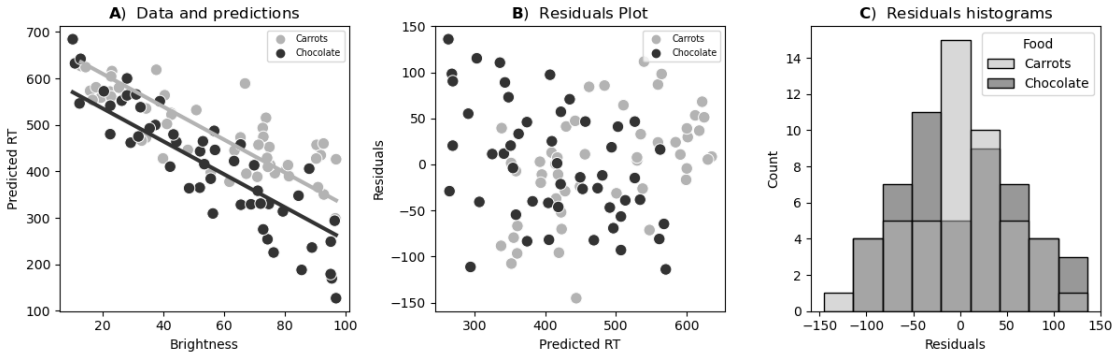
Omnibus:	1.487	**Durbin-Watson:**	2.132
Prob(Omnibus):	0.475	**Jarque-Bera (JB):**	1.201
Skew:	0.010	**Prob(JB):**	0.549
Kurtosis:	2.463	**Cond. No.**	170.

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

10 Figure 15.9/10: Visualizing example 2

[19]:
```
# generate predicted RT and residuals
df['Predicted RT'] = model.predict(X)
df['Residuals'] = df['Predicted RT'] - RT

# change the values of the dummy-coded variables
df['Food'] = df['Category'].map({0:'Carrots', 1:'Chocolate'})
colorPalette = {'Carrots':(.7,.7,.7),'Chocolate':(.2,.2,.2)} # color mapping for␣
 ↪visualization

### now for the visualizations
fig,axs = plt.subplots(1,3,figsize=(12,4))

# scatter plot of observed data
sns.scatterplot(x='Brightness',y=RT,hue='Food',data=df,
                ax=axs[0],s=80,palette=colorPalette)

# line plot of model predictions
sns.lineplot(x='Brightness',y='Predicted RT',hue='Food',data=df,legend=False,
             ax=axs[0],palette=colorPalette,linewidth=3)
axs[0].set_title(r'$\bf{A}$)  Data and predictions')

# residuals plot
sns.scatterplot(x='Predicted RT',y='Residuals',hue='Food',data=df,
                ax=axs[1],s=80,palette=colorPalette)
axs[1].set_title(r'$\bf{B}$)  Residuals Plot')

# histograms of residuals separated by category
sns.histplot(data=df,x='Residuals',hue='Food',
             palette=colorPalette,ax=axs[2])
axs[2].set(xlabel='Residuals',ylabel='Count')
axs[2].set_title(r'$\bf{C}$)  Residuals histograms')
```

```
# shrink down the legend font sizes
for a in axs[:2]: a.legend(fontsize='x-small')

plt.tight_layout()
#plt.savefig('reg_example2res2.png')
plt.show()
```



[20]:
```
# Correlations between predicted data and residuals
R = np.corrcoef(df['Predicted RT'],df['Residuals'])[0,1]
print(f'Overall correlation: r={R:.3f}')

print('')
print('Correlations grouped by food type:')
df.groupby('Food').apply(lambda group: group['Predicted RT'].
 ↪corr(group['Residuals']))
```

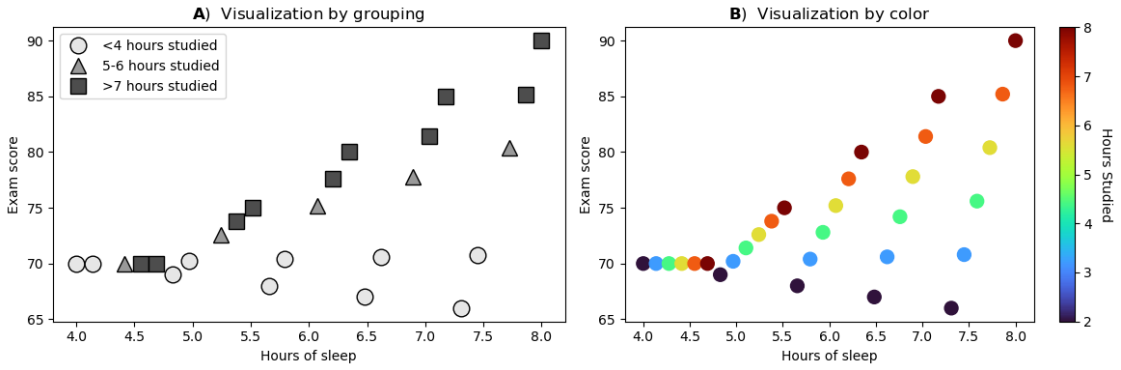
```
Overall correlation: r=-0.000

Correlations grouped by food type:
```

[20]:
```
Food
Carrots      0.499065
Chocolate   -0.461557
dtype: float64
```

11 Figure 15.13: Regression example 3

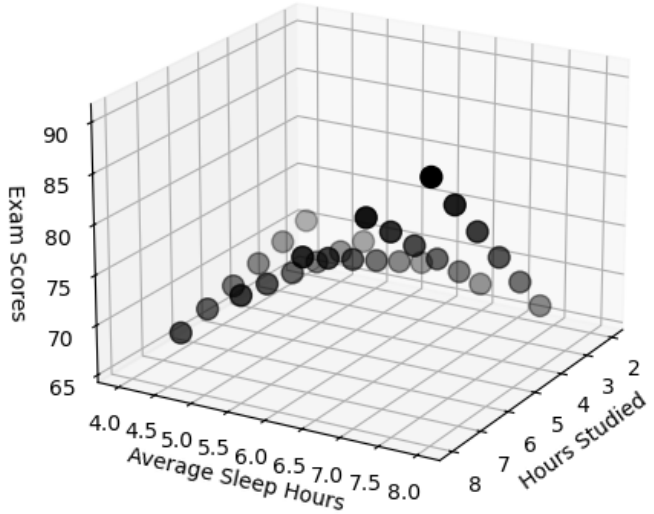
[21]:
```
### create the data
exam_scores = []
for ei in range(5):
  exam_scores = np.hstack((exam_scores,70*np.ones(6)+np.linspace(-1,5,6)*ei))
```

```
hours_studied = np.tile(np.linspace(2,8,6),5)
ave_sleep_hrs = np.linspace(4,8,30)

## plot the data
_,axs = plt.subplots(1,2,figsize=(12,4))

### stratify by hours studied
# fewer than 4 hours studied
plotidx = hours_studied<4.1
axs[0].plot(ave_sleep_hrs[plotidx],exam_scores[plotidx],'ko',
            markerfacecolor=(.9,.9,.9),markersize=12,label='<4 hours studied')

# 5-6 hours studied
plotidx = np.logical_and(hours_studied>4.9, hours_studied<6.1)
axs[0].plot(ave_sleep_hrs[plotidx],exam_scores[plotidx],'k^',
            markerfacecolor=(.6,.6,.6),markersize=12,label='5-6 hours studied')

# more than 6 hours
plotidx = hours_studied>6
axs[0].plot(ave_sleep_hrs[plotidx],exam_scores[plotidx],'ks',
            markerfacecolor=(.3,.3,.3),markersize=12,label='>7 hours studied')

axs[0].set(xlabel='Hours of sleep',ylabel='Exam score')
axs[0].legend()
axs[0].set_title(r'$\bf{A}$)  Visualization by grouping')

h = axs[1].scatter(ave_sleep_hrs,exam_scores,s=100,c=hours_studied,cmap='turbo')
cbar = plt.colorbar(h,ax=axs[1]) # colorbar
cbar.set_label('Hours Studied',rotation=270,labelpad=15)
axs[1].set(xlabel='Hours of sleep',ylabel='Exam score')
axs[1].set_title(r'$\bf{B}$)  Visualization by color')

plt.tight_layout()
#plt.savefig('reg_example3data2d.png')
plt.show()
```



12 Figure 15.12: Multidimensional data in a multidimensional space

[22]:
```
# A 3D visualization of the data (looks neat, but not really practical)
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(6,5))
ax = fig.add_subplot(111, projection='3d')
ax.
 ↪scatter(hours_studied,ave_sleep_hrs,exam_scores,c='k',marker='o',s=100,facecolors=(.
 ↪8,.8,.8))
ax.set(xlabel='Hours Studied',ylabel='Average Sleep Hours',zlabel='Exam Scores')
ax.set_box_aspect(aspect=None, zoom=.84)
# you can try changing the view angle to make the data more interpretable, but␣
 ↪the fact is that
# 2D representations are nearly always more informative than 3D representations.
ax.view_init(elev=20,azim=30)

plt.tight_layout()
#plt.savefig('reg_example3data3d.png')
plt.show()
```



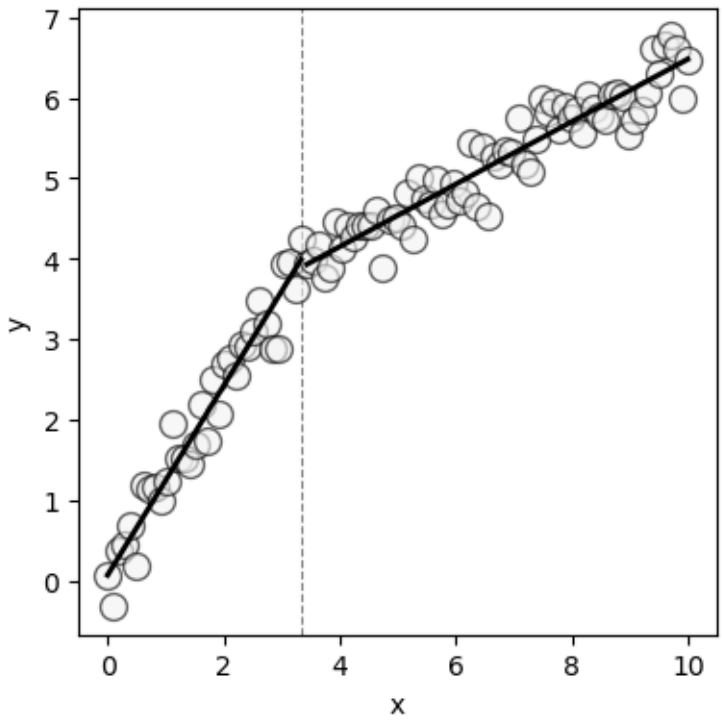
[23]:
```
# put all the data (IVs and DV) into one df

# construct the design matrix as a dataframe
df = pd.DataFrame({
    'ExamScores' : exam_scores,
    'Intercept'  : np.ones(len(exam_scores)),
    'StudyHours' : hours_studied,
    'SleepHours' : ave_sleep_hrs,
    'Interaction': hours_studied * ave_sleep_hrs
})
# let's have a look at the dataframe
df
```

[23]:

	ExamScores	Intercept	StudyHours	SleepHours	Interaction
0	70.0	1.0	2.0	4.000000	8.000000
1	70.0	1.0	3.2	4.137931	13.241379
2	70.0	1.0	4.4	4.275862	18.813793
3	70.0	1.0	5.6	4.413793	24.717241
4	70.0	1.0	6.8	4.551724	30.951724
5	70.0	1.0	8.0	4.689655	37.517241
6	69.0	1.0	2.0	4.827586	9.655172
7	70.2	1.0	3.2	4.965517	15.889655
8	71.4	1.0	4.4	5.103448	22.455172
9	72.6	1.0	5.6	5.241379	29.351724
10	73.8	1.0	6.8	5.379310	36.579310
11	75.0	1.0	8.0	5.517241	44.137931
12	68.0	1.0	2.0	5.655172	11.310345
13	70.4	1.0	3.2	5.793103	18.537931
14	72.8	1.0	4.4	5.931034	26.096552
15	75.2	1.0	5.6	6.068966	33.986207
16	77.6	1.0	6.8	6.206897	42.206897
17	80.0	1.0	8.0	6.344828	50.758621
18	67.0	1.0	2.0	6.482759	12.965517
19	70.6	1.0	3.2	6.620690	21.186207
20	74.2	1.0	4.4	6.758621	29.737931
21	77.8	1.0	5.6	6.896552	38.620690
22	81.4	1.0	6.8	7.034483	47.834483
23	85.0	1.0	8.0	7.172414	57.379310
24	66.0	1.0	2.0	7.310345	14.620690
25	70.8	1.0	3.2	7.448276	23.834483
26	75.6	1.0	4.4	7.586207	33.379310
27	80.4	1.0	5.6	7.724138	43.255172
28	85.2	1.0	6.8	7.862069	53.462069
29	90.0	1.0	8.0	8.000000	64.000000

13 Figure 15.14: Piecewise regression

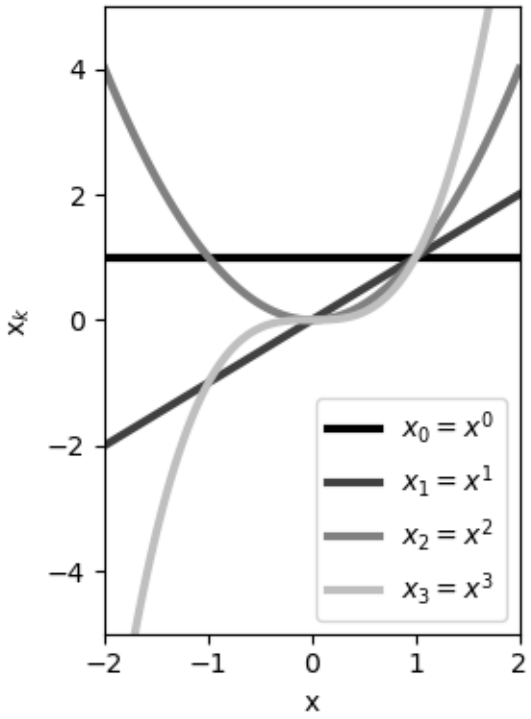
[24]:
```
## create the data
N = 100
x = np.linspace(0,10,N)
bp = N//3 # bp = break point (one-third of the way through)

# two different linear relationships
y1 = 1.2*x[:bp]
y2 =  .4*x[bp:]
y2 = y2-y2[0]+y1[-1] # shift y2 to follow y1

# combine the two parts with noise
y = np.concatenate([y1,y2]) + np.random.normal(0,.3,size=N)

### split the data
# (here we know exactly where to split; in Exercise 6 you'll write an algorithm␣
 ↪to find the best split)
x1, y1 = x[x <= x[bp]], y[x <= x[bp]]
x2, y2 = x[x >  x[bp]], y[x >  x[bp]]

# fit separate linear regressions
reg1 = sm.OLS(y1,sm.add_constant(x1)).fit()
reg2 = sm.OLS(y2,sm.add_constant(x2)).fit()

# predictions
yHat1 = reg1.predict(sm.add_constant(x1))
yHat2 = reg2.predict(sm.add_constant(x2))

## plotting
plt.figure(figsize=(4,4))
plt.plot(x,y,'ko',markerfacecolor=(.95,.95,.95),markersize=10,alpha=.6)
plt.plot(x1,yHat1,'k',linewidth=2)
plt.plot(x2,yHat2,'k',linewidth=2)
plt.axvline(x=x[bp],linestyle='--',color=(.5,.5,.5),zorder=-10,linewidth=.8)
plt.xlabel('x')
plt.ylabel('y')

plt.tight_layout()
#plt.savefig('reg_piecewise.png')
plt.show()
```



14 Figure 15.15: Polynomial design matrix

[25]:
```
x = np.linspace(-2,2,101)
maxorder = 3

plt.figure(figsize=(3,4))

for i in range(maxorder+1):
  # this regressor
  xx = x**i

  # plot it
  c = i/(maxorder+1)
  plt.plot(x,xx,color=(c,c,c),linewidth=3,label=fr'$x_{i}=x^{i}$')

plt.xlim(x[[0,-1]])
plt.ylim([-5,5])
plt.xlabel('x')
plt.ylabel('x$_k$')
plt.legend()

plt.tight_layout()
```

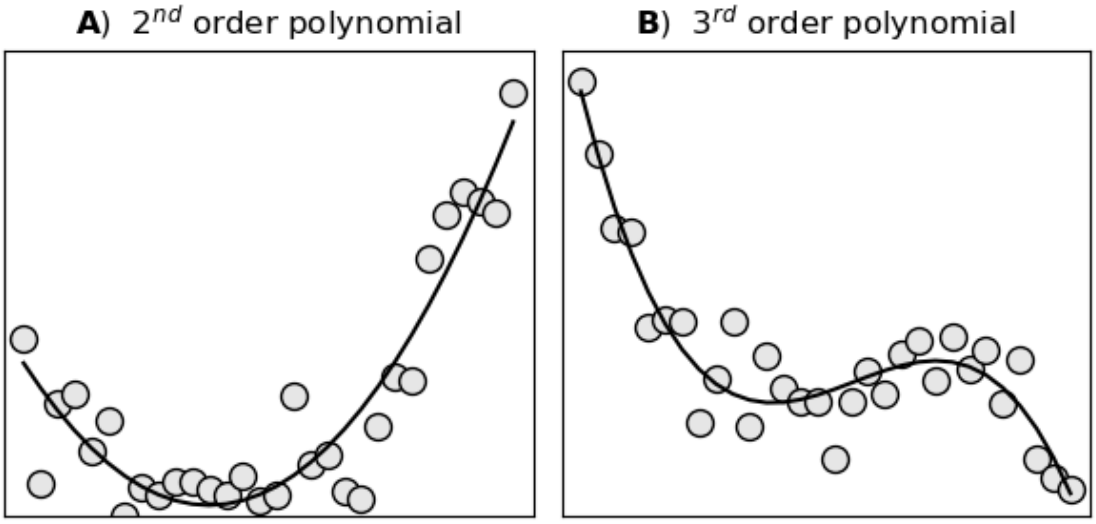
```
#plt.savefig('reg_polydesmat.png')
plt.show()
```



15 Figure 15.16: Polynomial regression

[27]:
```
n = 30
x = np.linspace(-2,3,n)

_,axs = plt.subplots(1,2,figsize=(6,3))

# generate data
y = x**2 + np.random.randn(n)

# beta coefficients (need only the coefficients, not a full model evaluation)
polycoefs = np.polyfit(x,y,2)

# predictions
yHat = np.polyval(polycoefs,x)

# and plot
axs[0].plot(x,y,'ko',markersize=10,markerfacecolor=(.9,.9,.9))
```

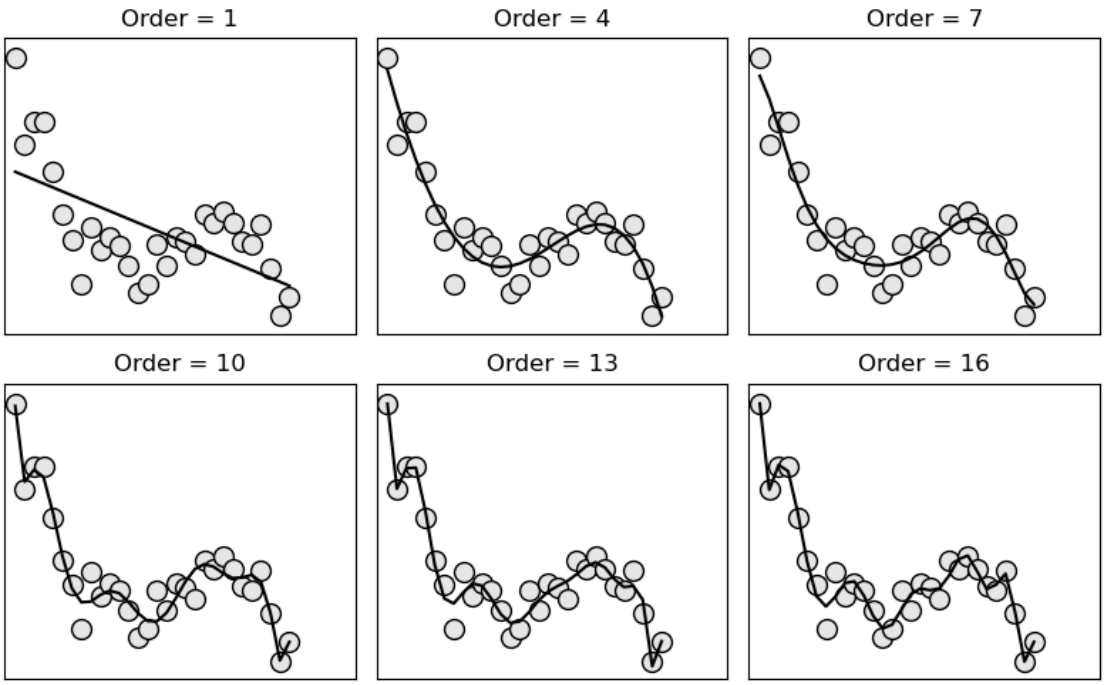
```
axs[0].set(xlim=[-2.2,3.2],ylim=[np.min(y)*1.3,np.max(y)*1.
 ↪1],xticks=[],yticks=[])
axs[0].plot(x,yHat,color='k')
axs[0].set_title(r'$\bf{A}$)  2$^{nd}$ order polynomial')

# repeat for 3rd order polynomial
y = x**2 - .4*x**3 + np.random.randn(len(x))*.8
axs[1].plot(x,y,'ko',markersize=10,markerfacecolor=(.9,.9,.9))
axs[1].set(xlim=[-2.2,3.2],ylim=[np.min(y)*1.3,np.max(y)*1.
 ↪1],xticks=[],yticks=[])
axs[1].plot(x,np.polyval(np.polyfit(x,y,3),x),color='k')
axs[1].set_title(r'$\bf{B}$)  3$^{rd}$ order polynomial')

plt.tight_layout()
#plt.savefig('reg_polyExample23.png')
plt.show()
```



16 Figure 15.17: Polynomial order and overfitting

[28]:
```
# plotting various orders
n = 30
x = np.linspace(-2,3,n)
y = x**2 - .4*x**3 + np.random.randn(n)*.8

_,axs = plt.subplots(2,3,figsize=(8,5))
```

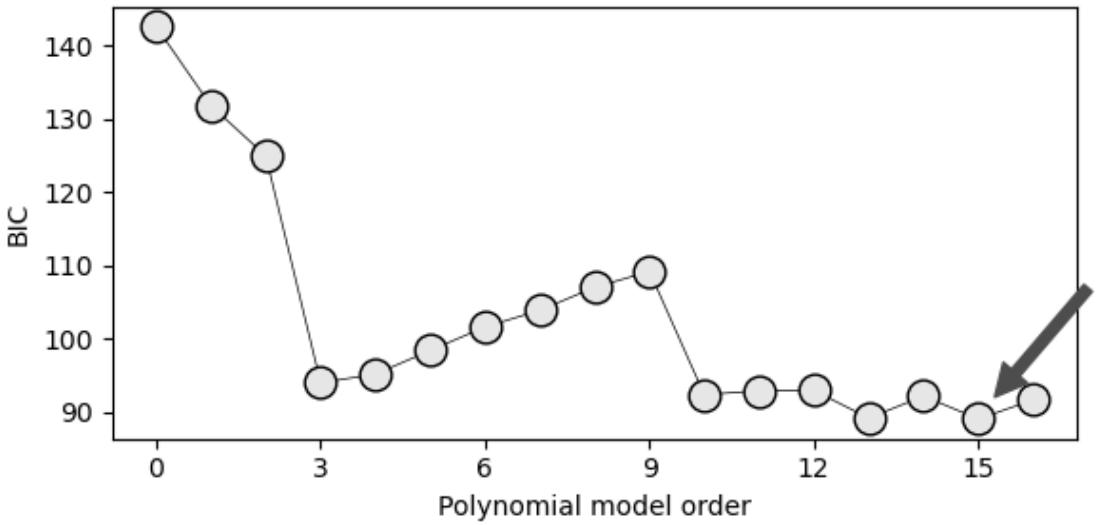
```
for oi,ax in enumerate(axs.flatten()):

  # order number
  order = oi*3+1

  ax.plot(x,y,'ko',markersize=10,markerfacecolor=(.9,.9,.9))
  ax.set(xlim=[-2.2,4.2],ylim=[np.min(y)*1.3,np.max(y)*1.1],xticks=[],yticks=[])
  ax.plot(x,np.polyval(np.polyfit(x,y,order),x),color='k')
  ax.set_title(f'Order = {order}',loc='center')

plt.tight_layout()
#plt.savefig('reg_polyManyOrders.png')
plt.show()
```



17 Figure 15.18: Bayes Information Criteria (BIC)

[29]:
```
# initialize
maxorder = 17
bic = np.zeros(maxorder)

for i in range(maxorder):
  yhat = np.polyval(np.polyfit(x,y,i),x)
  bic[i] = n*np.log( np.sum((yhat-y)**2) ) + i*np.log(n)
```

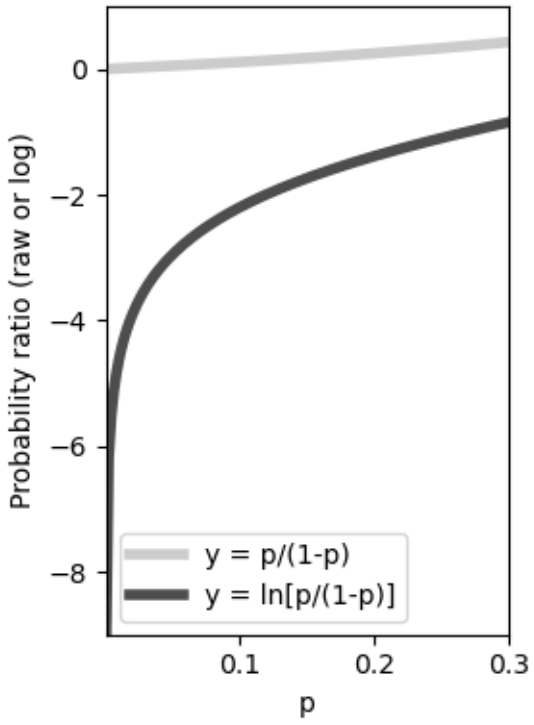
```
plt.figure(figsize=(6,3))
plt.plot(range(maxorder),bic,'ko-',markersize=12,markerfacecolor=(.9,.9,.
 ↪9),linewidth=.5)
plt.xlabel('Polynomial model order')
plt.xticks(range(0,maxorder,3))
plt.ylabel('BIC')

# draw an arrow to the best BIC
bestK = np.argmin(bic)
plt.annotate('',xy=(bestK+.3,bic[bestK]+3),xytext=(bestK+2,bic[bestK]*1.2),
            arrowprops={'color':(.3,.3,.3)})

plt.tight_layout()
#plt.savefig('reg_polyBIC.png')
plt.show()
```



18 Figure 15.19: Log of probabilities

[30]:
```
p = np.linspace(.0001,.3,156)

plt.figure(figsize=(3,4))
plt.plot(p,p/(1-p),color=(.8,.8,.8),linewidth=4,label='y = p/(1-p)')
plt.plot(p,np.log(p/(1-p)),color=(.3,.3,.3),linewidth=4,label='y = ln[p/(1-p)]')

plt.ylim([-9,1])
plt.xlim(p[[0,-1]])
plt.xlabel('p')
plt.ylabel('Probability ratio (raw or log)')
```

```
plt.legend()

plt.tight_layout()
#plt.savefig('reg_logOdds.png')
plt.show()
```



19 Logistic regression example

[31]:
```
# Generate data
N = 100
studyHours = np.random.uniform(0,10,N)

# the generating equation
pass_prob = 1 / (1 + np.exp(-(studyHours-5)))

# randomize pass/fail according to probability function
passed_exam = np.random.rand(N)<pass_prob
```

[32]:
```
# build design matrix
X = np.vstack((np.ones(N),studyHours)).T
```

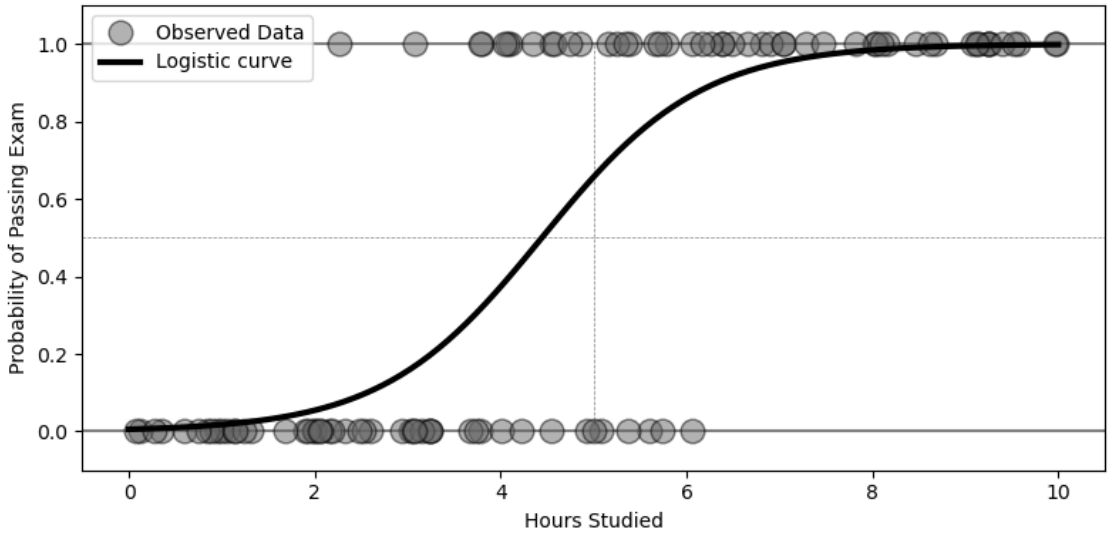
```
# test the model
model = sm.Logit(passed_exam,X).fit()
print(model.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.310446
         Iterations 8
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  100
Model:                          Logit   Df Residuals:                       98
Method:                           MLE   Df Model:                            1
Date:                Tue, 06 Aug 2024   Pseudo R-squ.:                  0.5520
Time:                        01:35:35   Log-Likelihood:                -31.045
converged:                       True   LL-Null:                       -69.295
Covariance Type:            nonrobust   LLR p-value:                 2.202e-18
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -5.1853      1.063     -4.879      0.000      -7.268      -3.102
x1             1.1668      0.231      5.042      0.000       0.713       1.620
==============================================================================
```

20 Figure 15.20: Visualization of logistic regression

[33]:
```
# interpolated values for study times
xx = np.linspace(0,10,N)

# predicted probabilities
yHat = model.predict(sm.add_constant(xx))

# and plot
plt.figure(figsize=(8,4))
plt.plot(studyHours,passed_exam,'ko',markersize=12,markerfacecolor=(.4,.4,.
 ↪4),alpha=.5,label='Observed Data')
plt.plot(xx,yHat,'k',linewidth=3,label='Logistic curve')
plt.axhline(0,color='gray',zorder=-10)
plt.axhline(1,color='gray',zorder=-10)
plt.axhline(.5,color='gray',linestyle='--',zorder=-10,linewidth=.5)
plt.plot([5,5],[0,1],'--',color='gray',zorder=-3,linewidth=.5)

plt.xlabel('Hours Studied')
plt.ylabel('Probability of Passing Exam')
plt.ylim([-.1,1.1])
plt.legend()

plt.tight_layout()
```

```
#plt.savefig('reg_logistic.png')
plt.show()
```



21 Exercise 1

[34]:
```
# the data (copied from the top of the code file)
icecream  = np.array([ 1, 2, 4, 5, 7 ])
happiness = np.array([ 5, 6.5, 6, 8, 9 ])

# construct a design matrix
X = np.vstack((icecream,np.ones(len(icecream)))).T

# compute the left inverse
leftInv = np.linalg.inv(X.T@X)@X.T

# compute the regression coefficients
betas = leftInv@happiness
betas # compare to output of sm.OLS: [0.6096, 4.5833]
```

[34]:
```
array([0.60964912, 4.58333333])
```

22 Exercise 2

[35]:
```
# sample size
N = 100
```

```
# create data and design matrix
DV = np.random.randn(N)
DM = np.random.randn(N,1) # change 1 to 37

# fit the model (including intercept)
model = sm.OLS(DV,sm.add_constant(DM)).fit()

# print the r-squared terms
print(f'    R-squared: {model.rsquared:.3f}')
print(f'adj.R-squared: {model.rsquared_adj:.3f}')
```

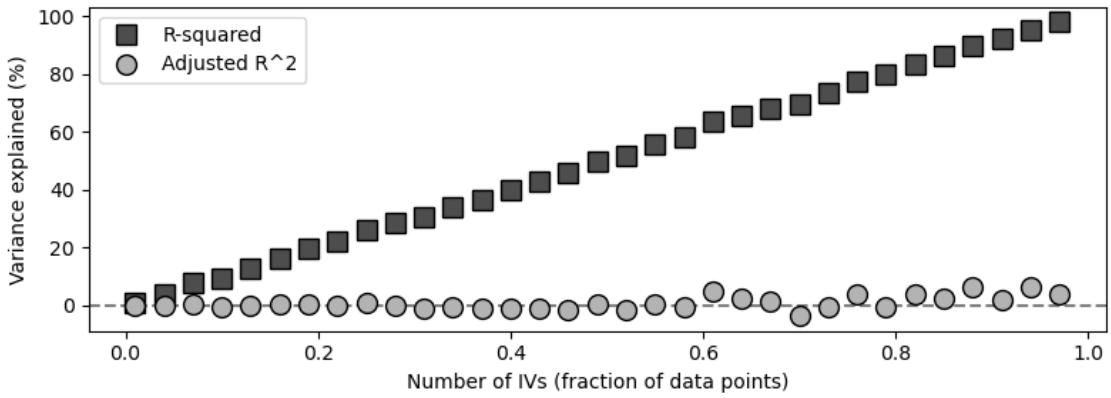
```
    R-squared: 0.008
adj.R-squared: -0.002
```

[36]:
```
# not part of the instructions, but FYI: demo that R2 is literally r(y,yHat)^2
print( np.corrcoef(model.predict(sm.add_constant(DM)),DV)[0,1]**2 )
print( model.rsquared )
```

```
0.007886936062058265
0.007886936062058147
```

[38]:
```
# initializations
nIVs = np.arange(1,N,3)
results = np.zeros((len(nIVs),2))

# the experiment
for idx,M in enumerate(nIVs):
  # loop over repetitions with new random numbers
  for expi in range(50):
    # create data and design matrix
    DV = np.random.randn(N)
    DM = np.random.randn(N,M)

    # fit the model (including intercept)
    model = sm.OLS(DV,sm.add_constant(DM)).fit()

    # get the r-squared terms
    results[idx,0] += 100*model.rsquared
    results[idx,1] += 100*model.rsquared_adj

# divide for the average
results /= (expi+1)

# now plot
plt.figure(figsize=(8,3))
plt.plot(nIVs/N,results[:,0],'ks',markersize=10,markerfacecolor=(.3,.3,.
 ↪3),label='R-squared')
```

```
plt.plot(nIVs/N,results[:,1],'ko',markersize=10,markerfacecolor=(.7,.7,.
 ↪7),label='Adjusted R^2')
plt.axhline(y=0,color='gray',linestyle='--',zorder=-4)
plt.xlabel('Number of IVs (fraction of data points)')
plt.ylabel('Variance explained (%)')
plt.legend()

plt.tight_layout()
#plt.savefig('reg_ex2.png')
plt.show()
```



[39]:
```
# testing the model on new data drawn from the same population
print('R2 for these data:')
print(f'  {np.corrcoef(model.predict(sm.add_constant(DM)),DV)[0,1]**2:.3f}')

print('')
print('R2 for new data from the same population:')
print(f'  {np.corrcoef(model.predict(sm.add_constant(DM)),np.random.
 ↪randn(N))[0,1]**2:.3f}')
```

```
R2 for these data:
  0.988

R2 for new data from the same population:
  0.005
```

23 Exercise 3

[40]:
```
# Important: The data for this exercise come from the code that created Figure␣
 ↪15.13.
#            Run that code before running this code.
```

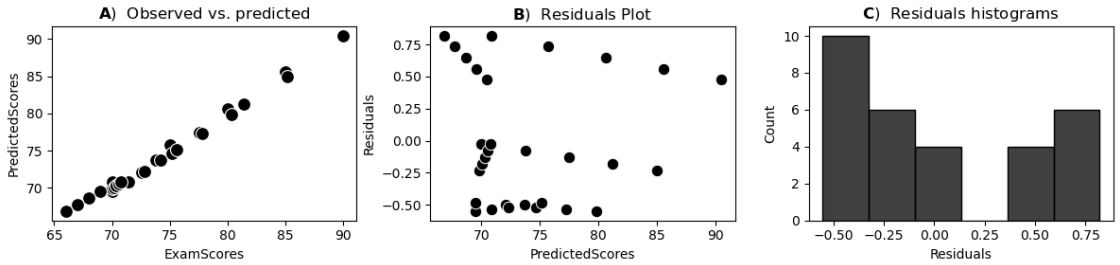
```
# be careful with excluding the DV from the df!
desmat = df.drop('ExamScores',axis=1) # design matrix
DV = df['ExamScores']

# fit the model
model = sm.OLS(DV,desmat).fit()

# show the regression summary
model.summary().as_text
```

[40]:
```
<bound method Summary.as_text of <class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:             ExamScores   R-squared:                       0.993
Model:                            OLS   Adj. R-squared:                  0.992
Method:                 Least Squares   F-statistic:                     1182.
Date:                Tue, 06 Aug 2024   Prob (F-statistic):           6.74e-28
Time:                        01:37:06   Log-Likelihood:                -21.269
No. Observations:                  30   AIC:                             50.54
Df Residuals:                      26   BIC:                             56.14
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      85.5292      1.277     66.953      0.000      82.903      88.155
StudyHours     -5.3191      0.242    -21.935      0.000      -5.818      -4.821
SleepHours     -3.4511      0.215    -16.087      0.000      -3.892      -3.010
Interaction     1.1736      0.040     29.623      0.000       1.092       1.255
==============================================================================
Omnibus:                       10.899   Durbin-Watson:                   1.069
Prob(Omnibus):                  0.004   Jarque-Bera (JB):                3.273
Skew:                          -0.438   Prob(JB):                        0.195
Kurtosis:                       1.640   Cond. No.                         475.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
""">
```

[41]:
```
# and visualizations

# generate predicted RT and residuals
df['PredictedScores'] = model.predict(desmat)
df['Residuals'] = df['PredictedScores'] - DV
```

```
### now for the visualizations
fig,axs = plt.subplots(1,3,figsize=(12,3))

# observed by predicted data
sns.scatterplot(data=df,x='ExamScores',␣
 ↪y='PredictedScores',s=100,ax=axs[0],color='k')
axs[0].set_title(r'$\bf{A}$)  Observed vs. predicted')

# Residuals plot
sns.
 ↪scatterplot(x='PredictedScores',y='Residuals',data=df,color='k',ax=axs[1],s=80)
axs[1].set_title(r'$\bf{B}$)  Residuals Plot')

# Plot histograms of residuals separated by category
sns.histplot(data=df,x='Residuals',ax=axs[2],color='k')
axs[2].set(xlabel='Residuals',ylabel='Count')
axs[2].set_title(r'$\bf{C}$)  Residuals histograms')

plt.tight_layout()
plt.show()
```

```
C:\Users\user\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



[ ]:
```
### Some observations:
#
# - The model predicts the data extremely well (panel A), which is no surprise:
#   there was no added noise and the simulation was linear.
#
# - The residuals plot looks... strange. There is no linear correlation, but␣
 ↪there is clearly structure in there.
#   Examination of the regression summary table reveals strong skew and a␣
 ↪significant non-normal distribution.
```

```
#
# - The residuals are strongly non-Gaussian distributed, probably due to the␣
 ↪issues identified above.
#
# - Overall, the model diagnostics reveal some deep issues with these data, and␣
 ↪question whether a regression is
#   appropriate in this case. On the other hand, the patterns in the data are so␣
 ↪clear that these violations can be
#   be tolerated in the interest of quantifying the effects of the IVs (sleep/
 ↪study, and their interaction).
#
```

24 Exercise 4

[42]:
```
import statsmodels.formula.api as smf

# define the formula
formula = 'ExamScores ~ StudyHours + SleepHours + StudyHours*SleepHours'

# fit and check the results
result = smf.ols(formula, data=df).fit()
result.summary().as_text
```

[42]:
```
<bound method Summary.as_text of <class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:             ExamScores   R-squared:                       0.993
Model:                            OLS   Adj. R-squared:                  0.992
Method:                 Least Squares   F-statistic:                     1182.
Date:                Tue, 06 Aug 2024   Prob (F-statistic):           6.74e-28
Time:                        01:37:40   Log-Likelihood:                -21.269
No. Observations:                  30   AIC:                             50.54
Df Residuals:                      26   BIC:                             56.14
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================

                            coef  std err    t     P>|t|    [0.025       0.975]
------------------------------------------------------------------------------
Intercept                85.5292   1.277  66.953   0.000    82.903      88.155
StudyHours               -5.3191   0.242 -21.935   0.000    -5.818      -4.821
SleepHours               -3.4511   0.215 -16.087   0.000    -3.892      -3.010
StudyHours:SleepHours     1.1736   0.040  29.623   0.000     1.092       1.255
==============================================================================
```

```
==============================================================================
Omnibus:                       10.899   Durbin-Watson:                   1.069
Prob(Omnibus):                  0.004   Jarque-Bera (JB):                3.273
Skew:                          -0.438   Prob(JB):                        0.195
Kurtosis:                       1.640   Cond. No.                         475.
==============================================================================

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

25 Exercise 5

[43]:
```
from sklearn.linear_model import LinearRegression

# extract relevant columns from the df
X = df[['StudyHours','SleepHours','Interaction']]
y = df['ExamScores']

# create the model (SK=scikit, to avoid overwriting 'model' from sm)
modelSK = LinearRegression()

# and fit the model
modelSK.fit(X,y)

# print the coefficients
print(f'Intercept: {modelSK.intercept_}')
print(f'Coefficients: {modelSK.coef_}')
```

```
Intercept: 85.5291766586731
Coefficients: [-5.31914468 -3.45113909  1.17356115]
```

26 Exercise 6

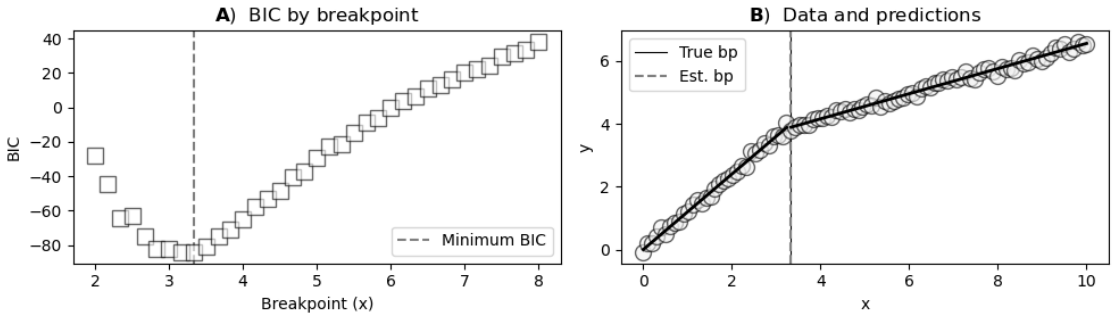
[44]:
```
## create the simulated data
N = 100
x = np.linspace(0,10,N)
bp = N//3 # bp = break point (one-third of the way through)

# two different linear relationships
y1 = 1.2*x[:bp]
y2 =  .4*x[bp:]
y2 = y2-y2[0]+y1[-1] # shift y2 to follow y1

# combine the two parts with noise
y = np.concatenate([y1,y2]) + np.random.normal(0,.1,size=N)
```

```
### run the experiment
# define breakpoints to evaluate, and initialize results
breakPoints2test = np.linspace(2,8,37)
BIC = np.zeros(len(breakPoints2test))

# now for the experiment
for idx,breakx in enumerate(breakPoints2test):
  # split the data
  x1, y1 = x[x <= breakx], y[x <= breakx]
  x2, y2 = x[x >  breakx], y[x >  breakx]

  # fit the regressions
  reg1 = sm.OLS(y1,sm.add_constant(x1)).fit()
  reg2 = sm.OLS(y2,sm.add_constant(x2)).fit()

  # take the average BICs from the two pieces
  BIC[idx] = (reg1.bic+reg2.bic)/2

# find and report the "optimal" breakpoint
bestBP = breakPoints2test[np.argmin(BIC)]

print(f'Empirical best breakpoint: x = {bestBP:.2f}')
print(f'Ground truth breakpoint:   x = {x[bp]:.2f}')
```

```
Empirical best breakpoint: x = 3.33
Ground truth breakpoint:   x = 3.33
```

[46]:
```
## now to run and visualize the model

# Split the data data again
x1, y1 = x[x <= bestBP], y[x <= bestBP]
x2, y2 = x[x >  bestBP], y[x >  bestBP]

# linear regressions (again)
reg1 = sm.OLS(y1,sm.add_constant(x1)).fit()
reg2 = sm.OLS(y2,sm.add_constant(x2)).fit()

# predictions
yHat1 = reg1.predict(sm.add_constant(x1))
yHat2 = reg2.predict(sm.add_constant(x2))

### plotting
_,axs = plt.subplots(1,2,figsize=(10,3))

axs[0].plot(breakPoints2test,BIC,'ks',markersize=10,alpha=.6,markerfacecolor='w')
axs[0].axvline(bestBP,linestyle='--',zorder=-1,color='gray',label='Minimum BIC')
```

```
axs[0].set(xlabel='Breakpoint (x)',ylabel='BIC')
axs[0].legend()
axs[0].set_title(r'$\bf{A}$)  BIC by breakpoint')

axs[1].plot(x,y,'ko',markerfacecolor=(.95,.95,.95),markersize=10,alpha=.6)
axs[1].plot(x1,yHat1,'k',linewidth=2)
axs[1].plot(x2,yHat2,'k',linewidth=2)
axs[1].axvline(x=x[bp],color='k',zorder=-10,linewidth=.8,label='True bp')
axs[1].axvline(x=bestBP,linestyle='--',zorder=-1,color='gray',label='Est. bp')
axs[1].set(xlabel='x',ylabel='y')
axs[1].legend()
axs[1].set_title(r'$\bf{B}$)  Data and predictions')

plt.tight_layout()
#plt.savefig('reg_ex6.png')
plt.show()
```

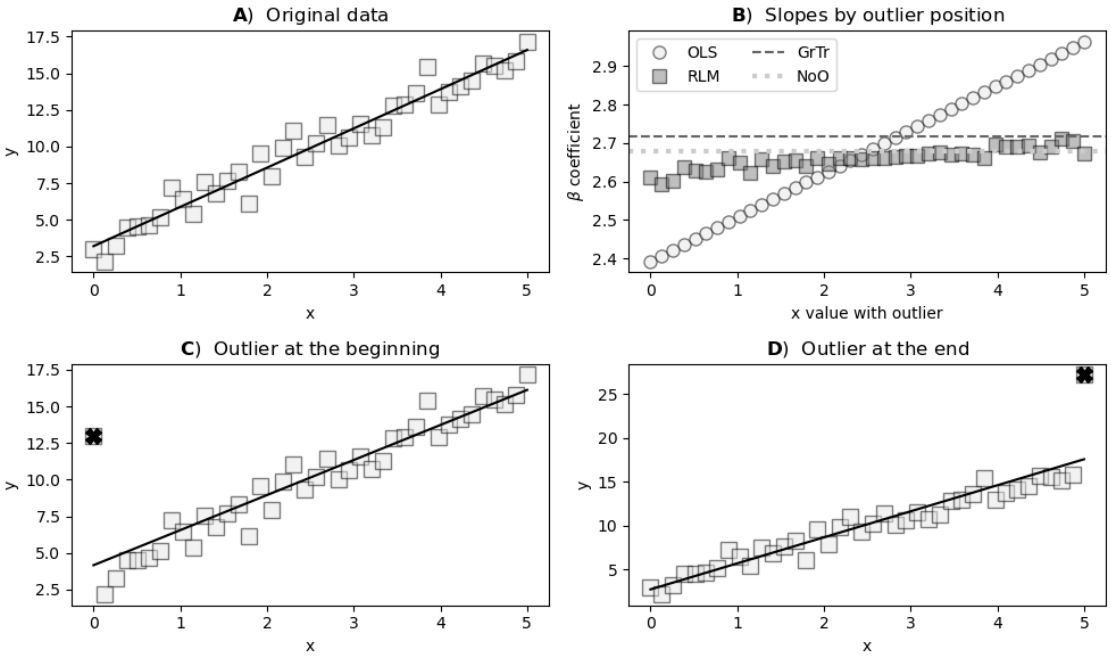


27 Exercise 7

[48]:
```
### simulate the data

# sample size
N = 40

# design matrix with intercept
X = sm.add_constant(np.linspace(0,5,N))

# create the DV
slope = np.exp(1)
y = slope*X[:,1] + np.pi + np.random.randn(N)

# fit the model
orig_model = sm.OLS(y,X).fit()
```

```
### experiment
betas = np.zeros((N,2))

for i in range(N):
  # make a copy of the data with an outlier
  yc = y.copy()
  yc[i] += 10

  # fit the model and get its slope (don't need to store the model)
  betas[i,0] = sm.OLS(yc,X).fit().params[1]
  betas[i,1] = sm.RLM(yc,X).fit().params[1]

### plotting
_,axs = plt.subplots(2,2,figsize=(10,6))

axs[0,0].plot(X[:,1],y,'ks',markerfacecolor=(.9,.9,.9),markersize=10,alpha=.5)
axs[0,0].plot(X[:,1],orig_model.predict(X),'k')
axs[0,0].set(xlabel='x',ylabel='y')
axs[0,0].set_title(r'$\bf{A}$)  Original data')

axs[0,1].plot(X[:,1],betas[:,0],'ko',markerfacecolor=(.9,.9,.
 ↪9),markersize=8,alpha=.5,label='OLS')
axs[0,1].plot(X[:,1],betas[:,1],'ks',markerfacecolor=(.5,.5,.
 ↪5),markersize=8,alpha=.5,label='RLM')
axs[0,1].axhline(y=slope,linestyle='--',color=(.4,.4,.4),label='GrTr')
axs[0,1].axhline(y=orig_model.params[1],linestyle=':',linewidth=3,color=(.8,.8,.
 ↪8),label='NoO')
axs[0,1].legend(ncol=2)
axs[0,1].set(xlabel='x value with outlier',ylabel=r'$\beta$ coefficient')
axs[0,1].set_title(r'$\bf{B}$)  Slopes by outlier position')

# drawing panels C and D
for i in range(2):
  # impose the outlier
  yc = y.copy()
  yc[-i] += 10

  # plot the data
  axs[1,i].plot(X[:,1],yc,'ks',markerfacecolor=(.9,.9,.9),markersize=10,alpha=.5)
  axs[1,i].plot(X[-i,1],yc[-i],'kX',markersize=10)

  # plot the model predictions
  axs[1,i].plot(X[:,1],sm.OLS(yc,X).fit().predict(X),'k')
  axs[1,i].set(xlabel='x',ylabel='y')
```

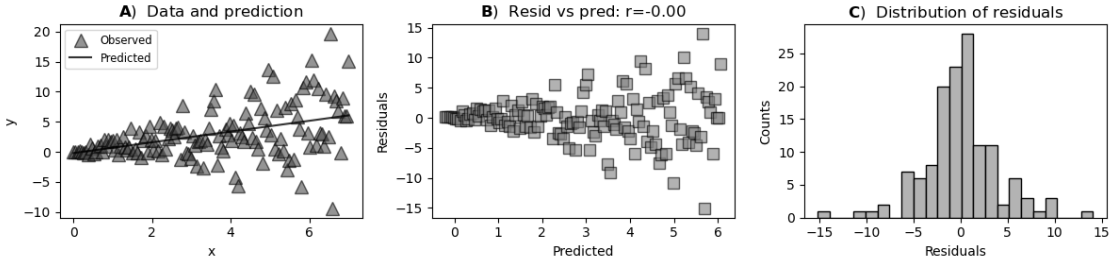
```
# finalizing titles
axs[1,0].set_title(r'$\bf{C}$)  Outlier at the beginning')
axs[1,1].set_title(r'$\bf{D}$)  Outlier at the end')

plt.tight_layout()
plt.savefig('reg_ex7.png')
plt.show()
```



28 Exercise 8

[49]:
```
# simulation parameters
N = 135
x = np.linspace(0,7,N)

# generate the data
y = 1*x + x*np.random.randn(N)

# fit the model
mdl = sm.OLS(y,sm.add_constant(x)).fit()

# get the predicted data and residuals
yHat = mdl.predict()
resid = mdl.resid
```

```
### plotting
_,axs = plt.subplots(1,3,figsize=(12,3))

axs[0].plot(x,y,'k^',markerfacecolor=(.3,.3,.3),alpha=.
 ↪6,markersize=10,label='Observed')
axs[0].plot(x,yHat,'k',markerfacecolor='w',alpha=.
 ↪8,markersize=8,label='Predicted')
axs[0].set(xlabel='x',ylabel='y')
axs[0].set_title(fr'$\bf{{A}}$)  Data and prediction')
axs[0].legend(fontsize='small')

axs[1].plot(yHat,resid,'ks',markerfacecolor=(.5,.5,.5),alpha=.6,markersize=8)
axs[1].set(ylabel='Residuals',xlabel='Predicted')
axs[1].set_title(fr'$\bf{{B}}$)  Resid vs pred: r={np.corrcoef(resid,yHat)[0,1]:.
 ↪2f}')

axs[2].hist(resid,bins='fd',edgecolor='k',facecolor=(.7,.7,.7))
axs[2].set(xlabel='Residuals',ylabel='Counts')
axs[2].set_title(fr'$\bf{{C}}$)  Distribution of residuals')

plt.tight_layout()
#plt.savefig('reg_ex8a.png')
plt.show()
```



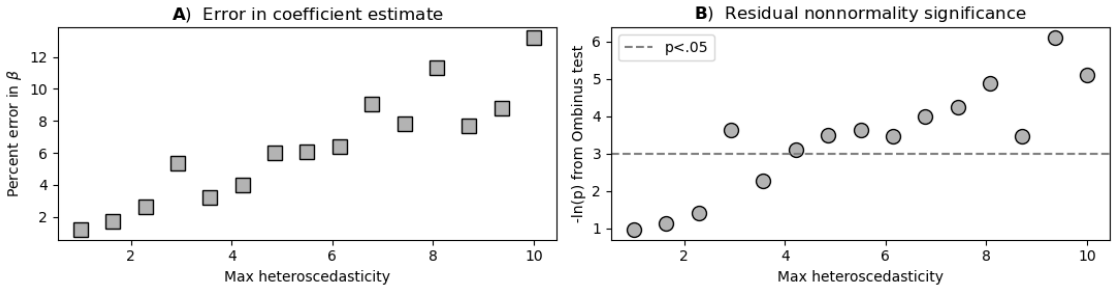
[50]:
```
# print the true and estimated betas

print( 'Ground-true beta: 1')
print(f'Estimated beta  : {mdl.params[1]:.3f}')
```

```
Ground-true beta: 1
Estimated beta  : 0.895
```

[51]:
```
# experiment parameters
m = 2         # slope (fixed for now)
numreps = 20 # number of repetitions in the experiment
```

```
# range of maximum heteroscedasticity values
badness = np.linspace(1,10,15)

# results matrix
mismatch = np.zeros((len(badness),2))

# how to simulate the noise; choose 1, 2, or 3
noiseSimulation = 1

# start the experiment!
for idx,maxbad in enumerate(badness):
  # repeat the experiment multiple times
  for _ in range(numreps):
    # generate the noise
    if noiseSimulation==1: # as initially specified in the exercise␣
↪(heterogeneity, but also total std, increase)
      noise = np.random.randn(N)*np.linspace(1,maxbad,N)

    elif noiseSimulation==2: # only manipulate overall noise levels (homogeneity)
      noise = np.random.randn(N)*np.linspace(maxbad,maxbad,N)

    elif noiseSimulation==3: # normalize the total noise so that only␣
↪heterogeneity is manipulated
      noise = np.random.randn(N)*np.linspace(1,maxbad,N)
      noise /= np.std(noise,ddof=1) # global normalization

    # generate the data
    y = m*x + noise

    # fit the model
    mdl = sm.OLS(y,sm.add_constant(x)).fit()
    mdl.summary() # need to call .summary() to create the diagn dictionary

    # store results (beta error and -ln(p)
    mismatch[idx,0] += np.abs(100*(mdl.params[1]-m)/m)
    mismatch[idx,1] += -np.log(mdl.diagn['omnipv'])

# divide to average
mismatch /= numreps

### plotting
_,axs = plt.subplots(1,2,figsize=(11,3))

axs[0].plot(badness,mismatch[:,0],'ks',markerfacecolor=(.7,.7,.7),markersize=10)
axs[0].set(xlabel='Max heteroscedasticity',ylabel=r'Percent error in $\beta$')
axs[0].set_title(r'$\bf{A}$)  Error in coefficient estimate')
```

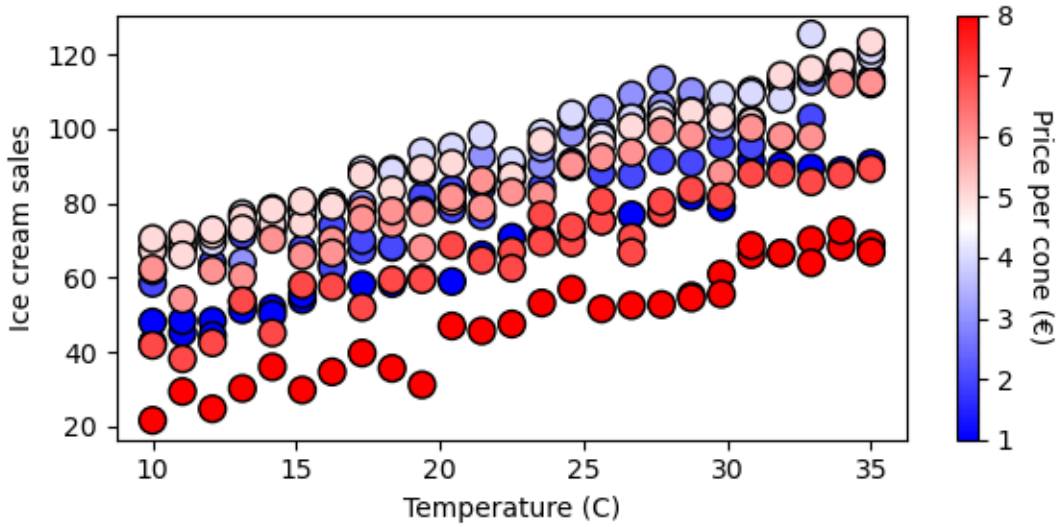
```
axs[1].plot(badness,mismatch[:,1],'ko',markerfacecolor=(.7,.7,.7),markersize=10)
axs[1].axhline(y=-np.log(.05),color='gray',linestyle='--',label='p<.05')
axs[1].legend()
axs[1].set(xlabel='Max heteroscedasticity',ylabel='-ln(p) from Ombinus test')
axs[1].set_title(r'$\bf{B}$)  Residual nonnormality significance')

plt.tight_layout()
plt.show()
```



29 Exercise 9

[52]:
```
# base sample size (will be multiplied by 10 the way I implemented the simulation)
N = 25

# the linear term (temp in C)
temp = np.tile( np.linspace(10,35,N) ,10)
b_temp = 2

# the nonlinear term (price)
price = np.ceil(np.linspace(.01,8,N*10))
b_price = -3

# some noise
noise = 4*np.random.randn(N*10)

# the DV
sales = 50 + b_temp*temp + b_price*(price-4)**2 + noise

# plotting
_,ax = plt.subplots(1,1,figsize=(6,3))
h = ax.scatter(temp,sales,s=100,c=price,cmap='bwr',edgecolor='k')
cbar = plt.colorbar(h,ax=ax) # colorbar
cbar.set_label('Price per cone (€)',rotation=270,labelpad=15)
ax.set(xlabel='Temperature (C)',ylabel='Ice cream sales')
```

```
plt.tight_layout()
plt.show()
```



[53]:
```
# build the regression model
# construct the design matrix as a dataframe
X = pd.DataFrame({
    'Sales'      : sales,
    'Intercept'  : np.ones(len(sales)),
    'Temperature': temp,
    'Price'      : (price-4)**2,
    'Interaction': temp * price
})
X
```

[53]:

	Sales	Intercept	Temperature	Price	Interaction
0	42.449604	1.0	10.000000	9.0	10.000000
1	45.236326	1.0	11.041667	9.0	11.041667
2	48.435474	1.0	12.083333	9.0	12.083333
3	53.567068	1.0	13.125000	9.0	13.125000
4	51.817093	1.0	14.166667	9.0	14.166667
..	...	...	...	...	...
245	68.648471	1.0	30.833333	16.0	246.666667
246	66.653055	1.0	31.875000	16.0	255.000000
247	63.995228	1.0	32.916667	16.0	263.333333
248	72.731615	1.0	33.958333	16.0	271.666667
249	66.900756	1.0	35.000000	16.0	280.000000

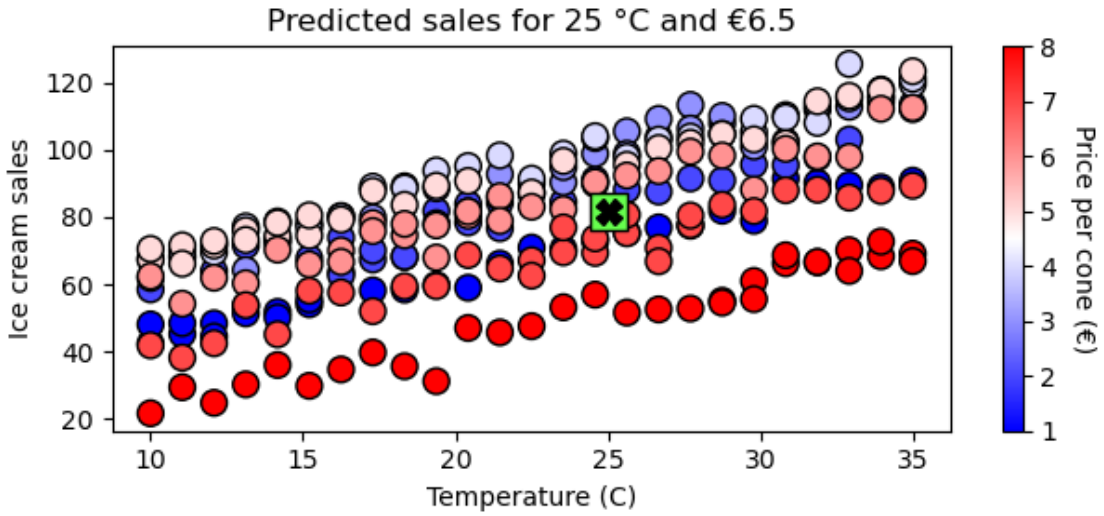
[250 rows x 5 columns]

[54]:
```
# fit the model
model = sm.OLS(X['Sales'],X.drop('Sales',axis=1)).fit()

# show the regression summary
model.summary().as_text
```

[54]:
```
<bound method Summary.as_text of <class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.966
Model:                            OLS   Adj. R-squared:                  0.966
Method:                 Least Squares   F-statistic:                     2342.
Date:                Tue, 06 Aug 2024   Prob (F-statistic):          1.55e-180
Time:                        01:41:34   Log-Likelihood:                -697.62
No. Observations:                 250   AIC:                             1403.
Df Residuals:                     246   BIC:                             1417.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      50.5360      0.851     59.376      0.000      48.860      52.212
Temperature     2.0140      0.043     47.305      0.000       1.930       2.098
Price          -2.9986      0.056    -53.805      0.000      -3.108      -2.889
Interaction    -0.0067      0.005     -1.268      0.206      -0.017       0.004
==============================================================================
Omnibus:                        2.427   Durbin-Watson:                   1.894
Prob(Omnibus):                  0.297   Jarque-Bera (JB):                1.884
Skew:                          -0.027   Prob(JB):                        0.390
Kurtosis:                       2.578   Cond. No.                         423.
==============================================================================

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

[55]:
```
### make a prediction

# predicted values of the IVs
predicted_temp = 25
predicted_price = 6.5

# use the sm.OLS object to make a prediction (the 4 input values correspond to␣
 ↪the IVs in the design matrix)
yHat = model.predict(exog=(1,predicted_temp,predicted_price,0))
```

```
# plotting
_,ax = plt.subplots(1,1,figsize=(6.5,3))
h = ax.scatter(temp,sales,s=100,c=price,cmap='bwr',edgecolor='k')
cbar = plt.colorbar(h,ax=ax) # colorbar
cbar.set_label('Price per cone (€)',rotation=270,labelpad=15)

# plot the prediction
ax.plot(predicted_temp,yHat,'ks',markerfacecolor=(.4,1,.3),markersize=14)
ax.plot(predicted_temp,yHat,'kX',markersize=10)

ax.set(xlabel='Temperature (C)',ylabel='Ice cream sales')
ax.set_title(f'Predicted sales for {predicted_temp} °C and␣
 ↪€{predicted_price}',loc='center')

plt.tight_layout()
#plt.savefig('reg_ex9b.png')
plt.show()
```



30 Exercise 10

[6]:
```
# dataset website ref
# https://archive.ics.uci.edu/dataset/437/residential+building+data+set

# download the zip file
# !wget https://archive.ics.uci.edu/static/public/437/
 ↪residential+building+data+set.zip -O z.zip

# unpack it locally
```

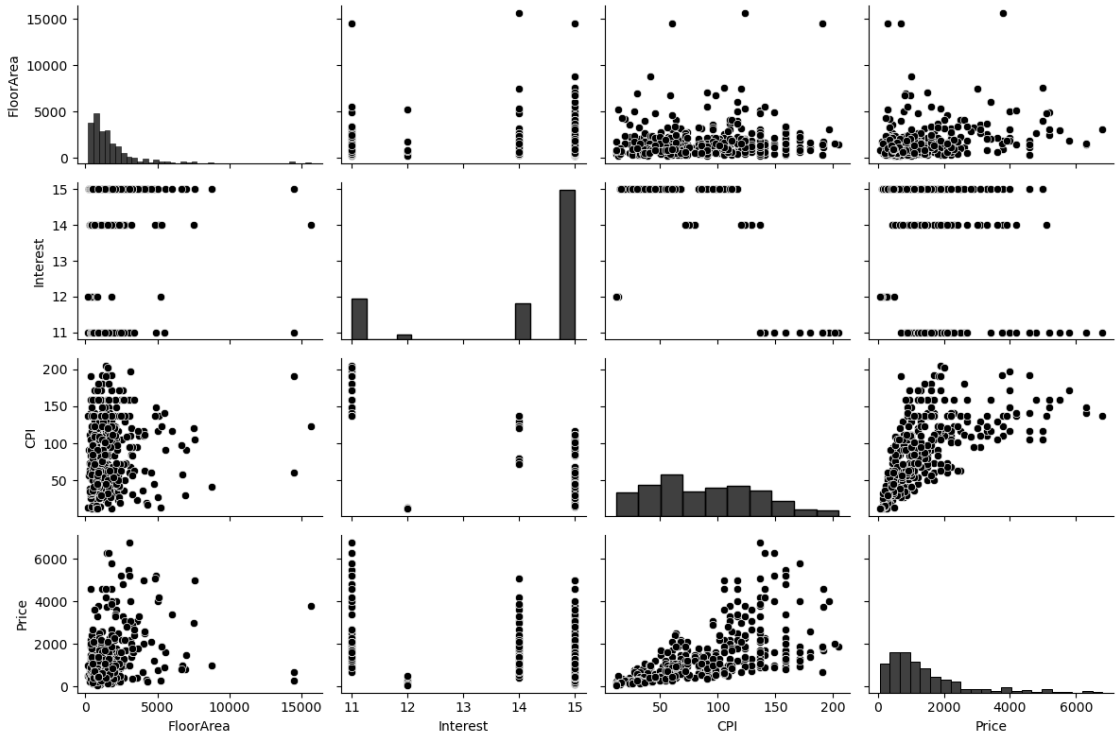
```
import zipfile
with zipfile.ZipFile('z.zip','r') as zz:
  zz.extractall('./')

# import into pandas
data = pd.read_excel('content/Residential-Building-Data-Set.
 ↪xlsx',skiprows=1,usecols='F,V,AA,DD')
data.columns = ['FloorArea','Interest','CPI','Price']
data
```

[6]:

```
     FloorArea  Interest     CPI  Price
0       3150.0        15   67.81   2200
1       7600.0        15  105.52   5000
2       4800.0        15   45.91   1200
3        685.0        12   11.62    165
4       3000.0        11  158.63   5500
..         ...       ...     ...    ...
367     1350.0        15  101.00   1100
368      600.0        14   71.56    740
369     1900.0        15  112.15    860
370      510.0        15   85.37   1100
371      890.0        15   60.74    460

[372 rows x 4 columns]
```

[7]:

```
# pairplot
sns.pairplot(data,height=2,aspect=1.5,
             plot_kws={'color':'black'},diag_kws={'color': 'black'})

plt.tight_layout()
#plt.savefig('reg_ex10b.png')
plt.show()

# decisions based on visual inspection:
# 1) log-transform FloorArea and Price
# 2) Binarize Interest
```



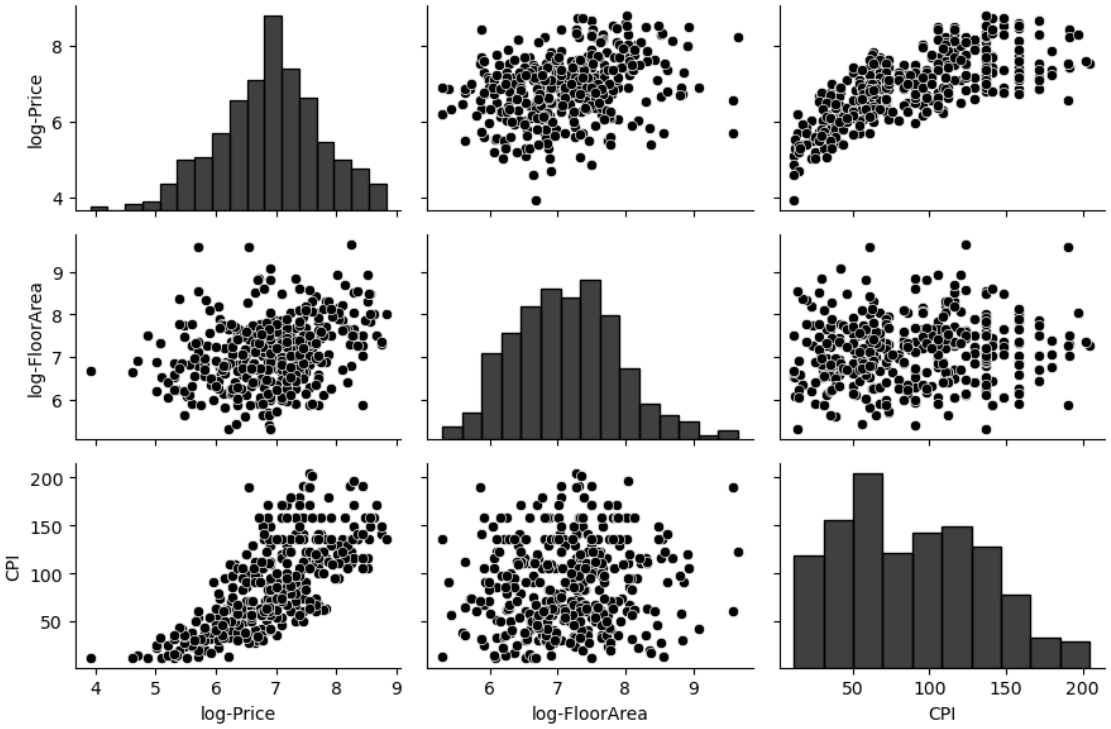
[8]:
```
# transformations
data['log-Price']     = np.log(data['Price'])
data['log-FloorArea'] = np.log(data['FloorArea'])
data['bin-Interest']  = (data['Interest']>14.5) + 0
data
```

[8]:

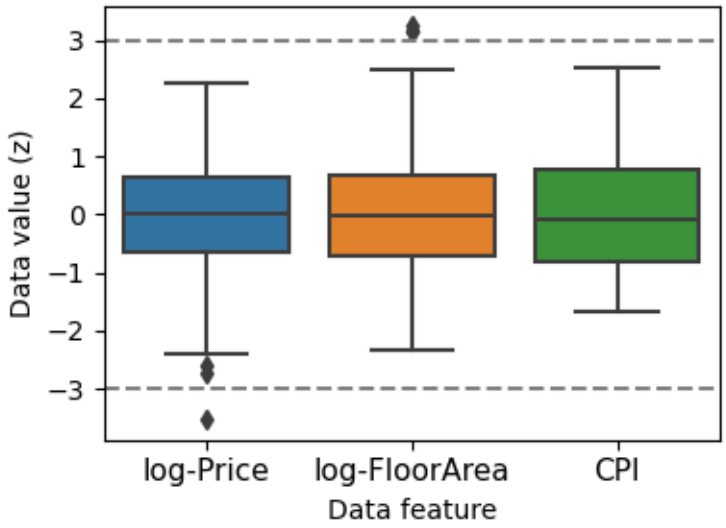
	FloorArea	Interest	CPI	Price	log-Price	log-FloorArea	\
0	3150.0	15	67.81	2200	7.696213	8.055158	
1	7600.0	15	105.52	5000	8.517193	8.935904	
2	4800.0	15	45.91	1200	7.090077	8.476371	
3	685.0	12	11.62	165	5.105945	6.529419	
4	3000.0	11	158.63	5500	8.612503	8.006368	
..	...	...	...	...	...	...	
367	1350.0	15	101.00	1100	7.003065	7.207860	
368	600.0	14	71.56	740	6.606650	6.396930	
369	1900.0	15	112.15	860	6.756932	7.549609	
370	510.0	15	85.37	1100	7.003065	6.234411	
371	890.0	15	60.74	460	6.131226	6.791221	

```
     bin-Interest
0               1
1               1
2               1
3               0
4               0
..            ...
367             1
368             0
369             1
370             1
371             1

[372 rows x 7 columns]
```

[10]:
```
# redo the pairplot with the new variables
sns.pairplot(data,vars=['log-Price','log-FloorArea','CPI'],height=2,aspect=1.5,
             plot_kws={'color':'black'},diag_kws={'color': 'black'})
plt.tight_layout()
#plt.savefig('reg_ex10c.png')
plt.show()
```



[13]:
```
# pick a threshold for outliers
zThresh = 3 # p<.001 (not exactly .001, but z=3 is also a typical threshold, I␣
 ↪guess because people like integers)

# create a copy of the data and z-transform
data_z = data[['log-Price','log-FloorArea','CPI']].copy()
for col in data_z.columns:
  data_z[col] = (data[col] - data[col].mean()) / data[col].std(ddof=1)

# box plots of z-scored data
plt.figure(figsize=(4,3))
sns.boxplot(data=data_z).set(xlabel='Data feature',ylabel='Data value (z)')
plt.axhline(y=zThresh,color='gray',linestyle='--')
plt.axhline(y=-zThresh,color='gray',linestyle='--')
plt.gca().set_xticklabels(plt.gca().get_xticklabels(), fontsize=11)

plt.tight_layout()
#plt.savefig('reg_ex10z.png')
plt.show()
```

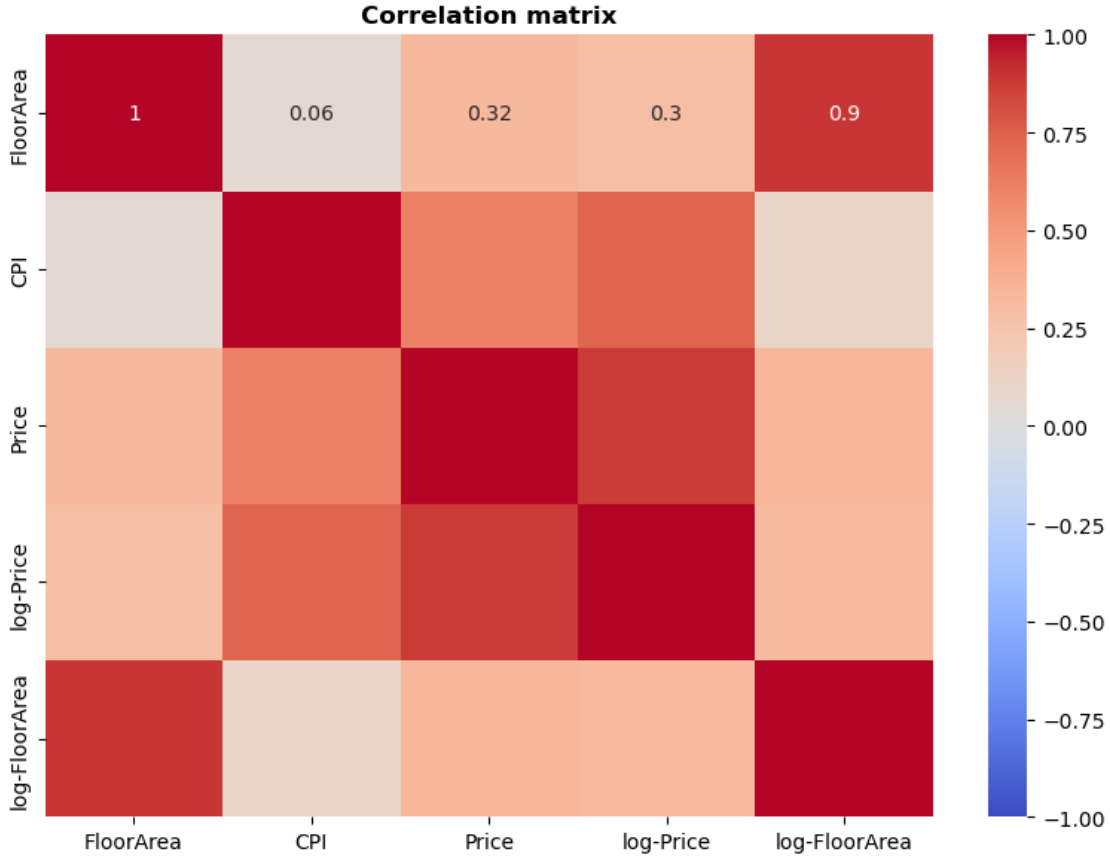


[14]:
```
# remove the outliers from the original data

print(f'Pre-cleaned dataset has {len(data)} rows.')

# actual remove (note that I'm using data_z to remove rows from data, and that␣
 ↪data.abs() makes it two-tailed)
data = data[(data_z.abs() <= zThresh).all(axis=1)].copy()
```

```
print(f'Post-cleaned dataset has {len(data)} rows.')

# tip: try re-running the previous cell to recreate the boxplot
```

```
Pre-cleaned dataset has 372 rows.
Post-cleaned dataset has 368 rows.
```

[15]:
```
## Correlation matrix

R = data.drop(['Interest','bin-Interest'],axis=1).corr()#method='spearman')

plt.figure(figsize=(8,6))
sns.heatmap(R, annot=True, cmap='coolwarm',vmin=-1,
            xticklabels=R.columns,yticklabels=R.columns)
plt.title('Correlation matrix',loc='center',weight='bold')
plt.tight_layout()
#plt.savefig('reg_ex10d.png')
plt.show()
```

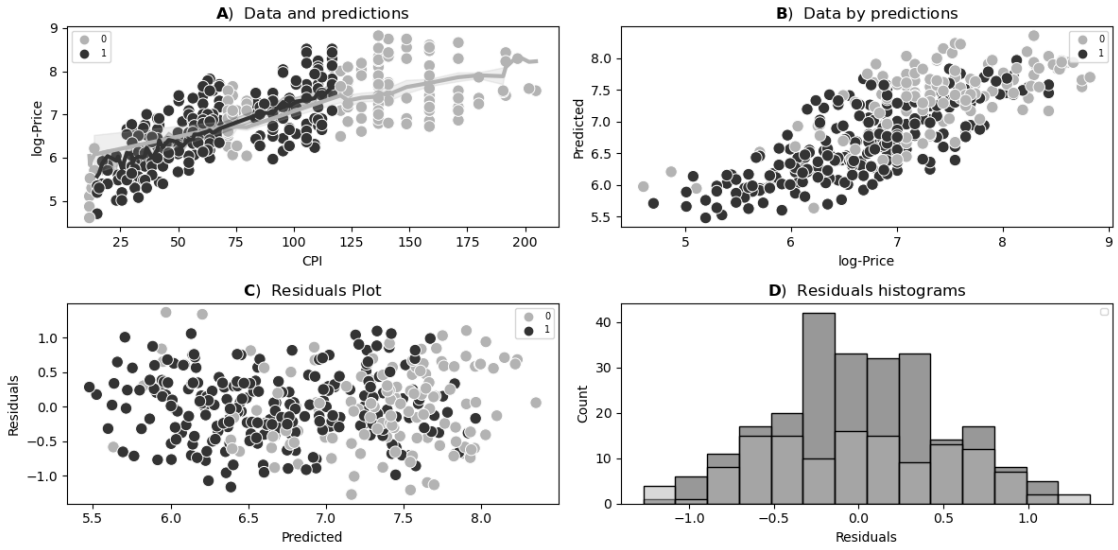


31 Exercise 11

[16]:
```
# add an intercept term
data['Intercept'] = np.ones(len(data))

# add an interaction
data['Int X CPI'] = data['bin-Interest']*data['CPI']

# fit the model
desmat = data.drop(['log-Price','Price','FloorArea','Interest'],axis=1)
model = sm.OLS(data['log-Price'],desmat).fit()

# show the regression summary
model.summary().as_text
```

[16]:
```
<bound method Summary.as_text of <class 'statsmodels.iolib.summary.Summary'>
                            OLS Regression Results
==============================================================================
Dep. Variable:              log-Price   R-squared:                       0.620
Model:                            OLS   Adj. R-squared:                  0.616
Method:                 Least Squares   F-statistic:                     148.3
Date:                Tue, 06 Aug 2024   Prob (F-statistic):           5.14e-75
Time:                        14:09:07   Log-Likelihood:                -275.02
No. Observations:                 368   AIC:                             560.0
Df Residuals:                     363   BIC:                             579.6
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================

                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
CPI              0.0108      0.001     10.338      0.000       0.009      0.013
log-FloorArea    0.2712      0.036      7.531      0.000       0.200      0.342
bin-Interest    -0.4746      0.162     -2.935      0.004      -0.792     -0.157
Intercept        4.0457      0.280     14.431      0.000       3.494      4.597
Int X CPI        0.0066      0.002      4.379      0.000       0.004      0.010
==============================================================================
Omnibus:                        6.351   Durbin-Watson:                   0.997
Prob(Omnibus):                  0.042   Jarque-Bera (JB):                4.026
Skew:                          -0.057   Prob(JB):                        0.134
Kurtosis:                       2.500   Cond. No.                     1.16e+03
==============================================================================

Notes:
[1] Std Errors assume that covariance matrix of errors is correctly specified.
[2] The condition number is large, 1.16e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[18]:
```
# plot with predicted data and residuals

# and visualizations
colorPalette = {0:(.7,.7,.7),1:(.2,.2,.2)} # color mapping for visualization

# generate predicted RT and residuals
data['Predicted'] = model.predict(desmat)
data['Residuals'] = data['Predicted'] - data['log-Price']

### now for the visualizations
fig,axs = plt.subplots(2,2,figsize=(12,6))

# scatter plot of observed data
sns.scatterplot(x='CPI',y='log-Price',hue='bin-Interest',data=data,
                palette=colorPalette,ax=axs[0,0],s=80)

# line plot of model predictions
sns.lineplot(x='CPI',y='Predicted',hue='bin-Interest',data=data,legend=False,
             ax=axs[0,0],linewidth=3,palette=colorPalette)
axs[0,0].set_title(r'$\bf{A}$)  Data and predictions')

# predicted by observed
sns.scatterplot(x='log-Price',y='Predicted',hue='bin-Interest',data=data,
                palette=colorPalette,ax=axs[0,1],s=80)
axs[0,1].set_title(r'$\bf{B}$)  Data by predictions')

# residuals plot
sns.scatterplot(x='Predicted',y='Residuals',hue='bin-Interest',data=data,
                ax=axs[1,0],s=80,palette=colorPalette)
axs[1,0].set_title(r'$\bf{C}$)  Residuals Plot')

# histograms of residuals separated by category
sns.histplot(data=data,x='Residuals',hue='bin-Interest',
             palette=colorPalette,ax=axs[1,1])
axs[1,1].set(xlabel='Residuals',ylabel='Count')
axs[1,1].set_title(r'$\bf{D}$)  Residuals histograms')

# shrink down the legend font sizes
for a in axs.flatten(): a.legend(fontsize='x-small')

plt.tight_layout()
#plt.savefig('reg_ex11b.png')
plt.show()
```



32 Exercise 12

[19]:
```
### standardizing the data

# the columns that need to be standardized (not the intercept!)
cols2zscore = [ 'CPI','log-FloorArea','log-Price','bin-Interest','Int X CPI' ]

# standardize into a new copy
dataStd = data.copy()
dataStd[cols2zscore] = (data[cols2zscore] - data[cols2zscore].mean()) /␣
↪data[cols2zscore].std(ddof=1)

# fit the model
desmatStd = dataStd.
↪drop(['log-Price','Price','FloorArea','Interest','Predicted','Residuals'],axis=1)
modelStd = sm.OLS(dataStd['log-Price'],desmatStd).fit()

# show the regression summary
modelStd.summary().as_text
```

[19]:
```
<bound method Summary.as_text of <class 'statsmodels.iolib.summary.Summary'>
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              log-Price   R-squared:                       0.620
Model:                            OLS   Adj. R-squared:                  0.616
Method:                 Least Squares   F-statistic:                     148.3
Date:                Tue, 06 Aug 2024   Prob (F-statistic):           5.14e-75
Time:                        14:10:22   Log-Likelihood:                -343.46
No. Observations:                 368   AIC:                             696.9
Df Residuals:                     363   BIC:                             716.5
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================

                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
CPI              0.5918     0.057     10.338      0.000       0.479       0.704
log-FloorArea    0.2454     0.033      7.531      0.000       0.181       0.309
bin-Interest    -0.2731     0.093     -2.935      0.004      -0.456      -0.090
Intercept     7.633e-16     0.032   2.36e-14      1.000      -0.064       0.064
Int X CPI        0.3229     0.074      4.379      0.000       0.178       0.468
==============================================================================
Omnibus:                        6.351   Durbin-Watson:                   0.997
Prob(Omnibus):                  0.042   Jarque-Bera (JB):                4.026
Skew:                          -0.057   Prob(JB):                        0.134
Kurtosis:                       2.500   Cond. No.                         5.60
==============================================================================

Notes:
[1] Std Errors assume covariance matrix of the errors is correctly specified.
```

[20]:
```python
### standarding the betas

# standard deviations of the data columns
stds = data.std(ddof=1)

# print top row of table
print('     Variable:  Unstd  | Beta-std | Data-std')
print('-------------:---------|----------|---------')

# loop through the variable names
for (name,beta),betaDataStd in zip(model.params.items(),modelStd.params):
  # compute the standardized beta from the variable stds
  betaStd = beta * stds[name]/stds['log-Price']
```

```
  # print everything!
  print(f'{name:>13}: {beta:7.4f} | {betaStd:7.4f}  | {betaDataStd:7.4f}')
```

```
     Variable:  Unstd  | Beta-std | Data-std
-------------:---------|----------|---------
          CPI:  0.0108 |  0.5918  |  0.5918
log-FloorArea:  0.2712 |  0.2454  |  0.2454
 bin-Interest: -0.4746 | -0.2731  | -0.2731
    Intercept:  4.0457 |  0.0000  |  0.0000
    Int X CPI:  0.0066 |  0.3229  |  0.3229
```

[21]:
```
# report the condition numbers
print(f'Condition number of the unstandardized design matrix: {np.linalg.
 ↪cond(desmat):8.2f}')
print(f'Condition number of the standardized design matrix  : {np.linalg.
 ↪cond(desmatStd):8.2f}')
```

```
Condition number of the unstandardized design matrix:  1164.46
Condition number of the standardized design matrix  :     5.60
```