Yen-Chun Chen
chenyenc@oregonstate.edu

Q1.

$$\begin{cases} a_1 x + b_1 y = c_1 \quad \cdots ① \\ a_2 x + b_2 y = c_2 \quad \cdots ② \end{cases}$$

(a)

$$② \times \frac{a_1}{a_2} \Rightarrow a_1 x + \frac{a_1 b_2}{a_2} y = \frac{a_1 c_2}{a_2} \quad \cdots ③$$

$$③ - ① \Rightarrow \frac{a_1 b_2}{a_2} y - b_1 y = \frac{a_1 c_2}{a_2} - c_1$$

$$\Rightarrow (a_1 b_2 - a_2 b_1) y = a_1 c_2 - a_2 c_1$$

Given that $a_1 b_2 - a_2 b_1 \neq 0 \Rightarrow y = \dfrac{a_1 c_2 - c_1 a_2}{a_1 b_2 - a_2 b_1}$ #

$$② \times \frac{b_1}{b_2} \Rightarrow \frac{a_2 b_1}{b_2} x + b_1 y = \frac{b_1}{b_2} c_2 \quad \cdots ④$$

$$④ - ① \Rightarrow \frac{a_2 b_1}{b_2} x - a_1 x = \frac{b_1}{b_2} c_2 - c_1$$

$$\Rightarrow (a_2 b_1 - b_2 a_1) x = b_1 c_2 - c_1 b_2$$

$$\Rightarrow (a_1 b_2 - a_2 b_1) x = c_1 b_2 - c_2 b_1$$

Given that $a_1 b_2 - a_2 b_1 \neq 0 \Rightarrow x = \dfrac{c_1 b_2 - b_1 c_2}{a_1 b_2 - a_2 b_1}$ #

(b) $y = \beta_1 \cdot g(x) + \beta_2 \cdot f(x) \xrightarrow{\text{Prediction}} \hat{y}_i = \hat{\beta}_1 \cdot g_i + \hat{\beta}_2 \cdot f_i$

$$L(\beta_1, \beta_2) = \sum_{i=1}^{n} (y_i - \beta_1 g_i - \beta_2 f_i)^2$$

To minimize β_1, β_2, take derivative with respect to β_1, β_2 equal to 0

$$\frac{\partial L}{\partial \beta_1} = \cancel{2} \sum_{i=1}^{n} (y_i - \beta_1 g_i - \beta_2 f_i) \cdot (-g_i) = 0$$

$$\rightarrow \sum_{i=1}^{n} \beta_1 g_i^2 + \sum_{i=1}^{n} \beta_2 f_i \cdot g_i = \sum_{i=1}^{n} y_i g_i$$

$$\frac{\partial L}{\partial \beta_2} = \cancel{2} \sum_{i=1}^{n} (y_i - \beta_1 g_i - \beta_2 f_i)(-f_i) = 0$$

$$\rightarrow \sum_{i=1}^{n} \beta_1 g_i f_i + \sum_{i=1}^{n} \beta_2 f_i^2 = \sum_{i=1}^{n} y_i f_i$$

a_1 b_1 c_1

$$\Rightarrow \begin{cases} \left(\sum g_i^2\right)\beta_1 + \left(\sum f_i g_i\right)\beta_2 = \sum y_i g_i & \cdots (5) \\ \left(\sum g_i f_i\right)\beta_1 + \left(\sum f_i^2\right)\beta_2 = \sum y_i f_i & \cdots (6) \end{cases}$$

a_2 b_2 c_2

According to (a)

$$\begin{cases} a_1 x + b_1 y = c_1 \\ a_2 x + b_2 y = c_2 \end{cases} \Rightarrow \begin{cases} x = \dfrac{c_1 b_2 - b_1 c_2}{a_1 b_2 - a_2 b_1} \\ y = \dfrac{a_1 c_2 - c_1 a_2}{a_1 b_2 - a_2 b_1} \end{cases}$$

$$\begin{cases} ⑤ \\ ⑥ \end{cases} \rightarrow \begin{cases} \hat{\beta}_1 = \dfrac{\left(\sum_{i=1}^{n} y_i g_i\right)\left(\sum_{i=1}^{n} f_i^2\right) - \left(\sum_{i=1}^{n} y_i f_i\right)\left(\sum_{i=1}^{n} f_i g_i\right)}{\left(\sum_{i=1}^{n} g_i^2\right)\left(\sum_{i=1}^{n} f_i^2\right) - \left(\sum_{i=1}^{n} f_i g_i\right)^2} & (3) \\ \hat{\beta}_2 = \dfrac{\left(\sum_{i=1}^{n} y_i f_i\right)\left(\sum_{i=1}^{n} g_i^2\right) - \left(\sum_{i=1}^{n} y_i g_i\right)\left(\sum_{i=1}^{n} f_i g_i\right)}{\left(\sum_{i=1}^{n} g_i^2\right)\left(\sum_{i=1}^{n} f_i^2\right) - \left(\sum_{i=1}^{n} f_i g_i\right)^2} & (4) \end{cases}$$

#

(C)

$$y = \beta f(x)$$

$$L(\beta) = \sum_{i=1}^{n} (y - \beta f_i)^2$$

$$0 = \frac{\partial L}{\partial \beta} = \cancel{2} \sum_{i=1}^{n} (y_i - \beta \cdot f_i)(-f_i)$$

$$\rightarrow \sum_{i=1}^{n} \beta \cdot f_i^2 = \sum_{i=1}^{n} y_i \cdot f_i$$

$$\rightarrow \left(\Sigma f_i^2\right)\beta = \Sigma y_i f_i$$

$$\rightarrow \hat{\beta} = \frac{\sum_{i=1}^{n} y_i f_i}{\sum_{i=1}^{n} f_i^2}$$

#

(d) $$\hat{\beta}_2 = \frac{\left(\sum_{i=1}^{n} y_i f_i\right)\left(\sum_{i=1}^{n} g_i^2\right) - \left(\sum_{i=1}^{n} y_i g_i\right)\left(\sum_{i=1}^{n} f_i g_i\right)}{\left(\sum_{i=1}^{n} g_i^2\right)\left(\sum_{i=1}^{n} f_i^2\right) - \left(\sum_{i=1}^{n} f_i g_i\right)^2}$$

Given that $g_i = 0$, the denominator will be 0 then $\hat{\beta}_2$ will be undefined. Therefore, it's not able to derive (4) by setting $g_i = 0$.

(e) $$E[\hat{\beta}] = E\left[\frac{\sum_{i=1}^{n} y_i f_i}{\sum_{i=1}^{n} f_i^2}\right] = E\left[\frac{\sum_{i=1}^{n} f_i (\beta f_i + \varepsilon_i)}{\sum_{i=1}^{n} f_i^2}\right]$$

$$= E\left[\frac{\beta \sum_{i=1}^{n} f_i^2}{\sum_{i=1}^{n} f_i^2}\right] + E\left[\frac{\sum_{i=1}^{n} \varepsilon_i}{\sum_{i=1}^{n} f_i^2}\right]$$

$$= \beta \frac{n \cdot f_i^2}{n \cdot f_i^2} + \frac{\sum_{i=1}^{n} E[\varepsilon_i]}{n \cdot f_i^2}$$

Given $M_\varepsilon = E[\varepsilon_i] = 0$

$\rightarrow E[\hat{\beta}] = \beta \Rightarrow \hat{\beta}$ is unbiased #

(f) $y_i = \beta x_i^2 + \varepsilon_i \quad , \quad \varepsilon_i \sim N(0, \sigma^2)$

$$\mathrm{Var}(y_i) = \mathrm{Var}(\beta x_i^2 + \varepsilon_i)$$

$$= \mathrm{Var}(\beta x_i^2) + \mathrm{Var}(\varepsilon_i)$$

$$= 0 + \mathrm{Var}(\varepsilon_i) = \sigma^2$$

$$\mathrm{Var}(\hat{\beta}) = \mathrm{Var}\left(\frac{\sum_{i=1}^{n} y_i f_i}{\sum_{i=1}^{n} f_i^2}\right) = \mathrm{var}\left(\frac{\sum(\beta x_i^2 + \varepsilon_i)(x_i^2)}{\sum x_i^4}\right)$$

$$= \mathrm{var}\left(\frac{\sum \beta \cdot x_i^4}{\sum x_i^4}\right) + \mathrm{var}\left(\frac{\sum x_i^2 \cdot \varepsilon_i}{\sum x_i^4}\right)$$

$$= 0 + \sum \left(\frac{(x_i^2)^2 \cdot \mathrm{Var}(\varepsilon_i)}{(\sum x_i^4)^2}\right)$$

$$= \frac{\sum x_i^4}{(\sum x_i^4)^2} \sigma^2 = \frac{\sigma^2}{\sum_{i=1}^{n} x_i^4} \quad \#$$

(g) $y = \beta x^2$

From (f): $\text{var}(\hat{\beta}) = \dfrac{\sigma^2}{\sum_{i=1}^{n} x_i^4}$

When giving larger x, $\sum_{i=1}^{n} x_i^4$ goes larger,

which decreases the value of $\text{var}(\hat{\beta})$.

Therefore, Yes, Larger x can reduce the uncertainty in evaluating $\hat{\beta}$ #

Q2

April 18, 2024

The goal of this question is to calculate body fat using Brozek's equation. In this problem, the first column **brozek** is the response variable, and all other columns are treated as the features (we may not use all the features, please only use the ones stated in each question). To start, read the data file fat.csv in the homework folder (containing 252 samples) and split it into two sets. Set 1 includes the first 200 rows of the data (do not count the row associated with the feature/response names), and set 2, which includes the last 52 rows of the data. Name the first set **train** and the second set **test**.

[1]:
```
import numpy as np
import pandas as pd
from matplotlib.pyplot import subplots
import statsmodels.api as sm
from ISLP import load_data
from ISLP.models import (ModelSpec as MS,
                         summarize)
```

[2]:
```
Fat = pd.read_csv("fat.csv")
train,test = np.split(Fat,[int(200)])
y_train = train['brozek']
X_train = pd.DataFrame({'intercept': np.ones(train.shape[0]),
                  'siri':    train['siri'],
                  'density': train['density'],
                  'age':     train['age'],
                  'weight':  train['weight'],
                  'height':  train['height'],
                  'adipos':  train['adipos'],
                  'free':    train['free'],
                  'neck':    train['neck'],
                  'chest':   train['chest'],
                  'abdom':   train['abdom'],
                  'hip':     train['hip'],
                  'thigh':   train['thigh'],
                  'knee':    train['knee'],
                  'ankle':   train['ankle'],
                  'biceps':  train['biceps'],
                  'forearm': train['forearm'],
                  'wrist':   train['wrist']})
```

(a) As a first modeling attempt, consider a linear model using all the 17 features, that is brozek = β_0 + $\beta_1$siri + ... + β_{17}wrist report the fitted parameters, the 95% confidence interval for each estimated parameter and the p-values. What is the R2 value, and based on that how good do you see the model fitting the training data?

[3]:
```
firstFittedModel = sm.OLS(y_train, X_train).fit()
firstFittedModel.summary()
```

[3]:

Dep. Variable:	brozek	**R-squared:**	0.999
Model:	OLS	**Adj. R-squared:**	0.999
Method:	Least Squares	**F-statistic:**	1.859e+04
Date:	Thu, 18 Apr 2024	**Prob (F-statistic):**	7.25e-285
Time:	18:26:29	**Log-Likelihood:**	61.275
No. Observations:	200	**AIC:**	-86.55
Df Residuals:	182	**BIC:**	-27.18
Df Model:	17		
Covariance Type:	nonrobust		

	coef	std err	t	P> |t|	[0.025	0.975]
intercept	11.7984	4.641	2.542	0.012	2.641	20.956
siri	0.8845	0.013	66.439	0.000	0.858	0.911
density	-9.5175	4.172	-2.281	0.024	-17.750	-1.285
age	-0.0007	0.002	-0.417	0.677	-0.004	0.003
weight	0.0116	0.005	2.429	0.016	0.002	0.021
height	0.0004	0.005	0.075	0.940	-0.009	0.010
adipos	-0.0213	0.016	-1.361	0.175	-0.052	0.010
free	-0.0134	0.006	-2.324	0.021	-0.025	-0.002
neck	-0.0037	0.011	-0.325	0.746	-0.026	0.019
chest	0.0034	0.005	0.631	0.529	-0.007	0.014
abdom	0.0005	0.005	0.090	0.929	-0.010	0.011
hip	-0.0037	0.007	-0.496	0.620	-0.018	0.011
thigh	0.0199	0.008	2.579	0.011	0.005	0.035
knee	-0.0305	0.013	-2.386	0.018	-0.056	-0.005
ankle	0.0037	0.010	0.359	0.720	-0.017	0.024
biceps	-0.0159	0.009	-1.871	0.063	-0.033	0.001
forearm	0.0196	0.010	1.891	0.060	-0.001	0.040
wrist	0.0340	0.027	1.246	0.214	-0.020	0.088

Omnibus:	146.785	**Durbin-Watson:**	1.875
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	8218.452
Skew:	2.048	**Prob(JB):**	0.00
Kurtosis:	34.136	**Cond. No.**	1.48e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.48e+05. This might indicate that there are strong multicollinearity or other numerical problems.

(b) Based on α = 0.05 and the calculated p-values, 10 features seem problematic.

feature	p-value
age	0.677
height	0.940
adipos	0.175
neck	0.746
chest	0.529
abdom	0.929
hip	0.620
ankle	0.720
biceps	0.063
wrist	0.214

[4]:
```
X_test = pd.DataFrame({'intercept': np.ones(test.shape[0]),
                       'siri':    test['siri'],
                       'density': test['density'],
                       'age':     test['age'],
                       'weight':  test['weight'],
                       'height':  test['height'],
                       'adipos':  test['adipos'],
                       'free':    test['free'],
                       'neck':    test['neck'],
                       'chest':   test['chest'],
                       'abdom':   test['abdom'],
                       'hip':     test['hip'],
                       'thigh':   test['thigh'],
                       'knee':    test['knee'],
                       'ankle':   test['ankle'],
                       'biceps':  test['biceps'],
                       'forearm': test['forearm'],
                       'wrist':   test['wrist']})
```

(c) Calculate the prediction error e_1 using your test file.

[5]:
```
y_actual = np.array(test['brozek'].values)

# Get prediction error and prediction
def get_prediction_error(model, X_test):
    # the actual value from test dataset
    # the value predicted by the full-features model
    y_prediction = np.array(model.predict(X_test))
    return (np.sqrt(sum(np.square(y_actual - y_prediction))), y_prediction)
```

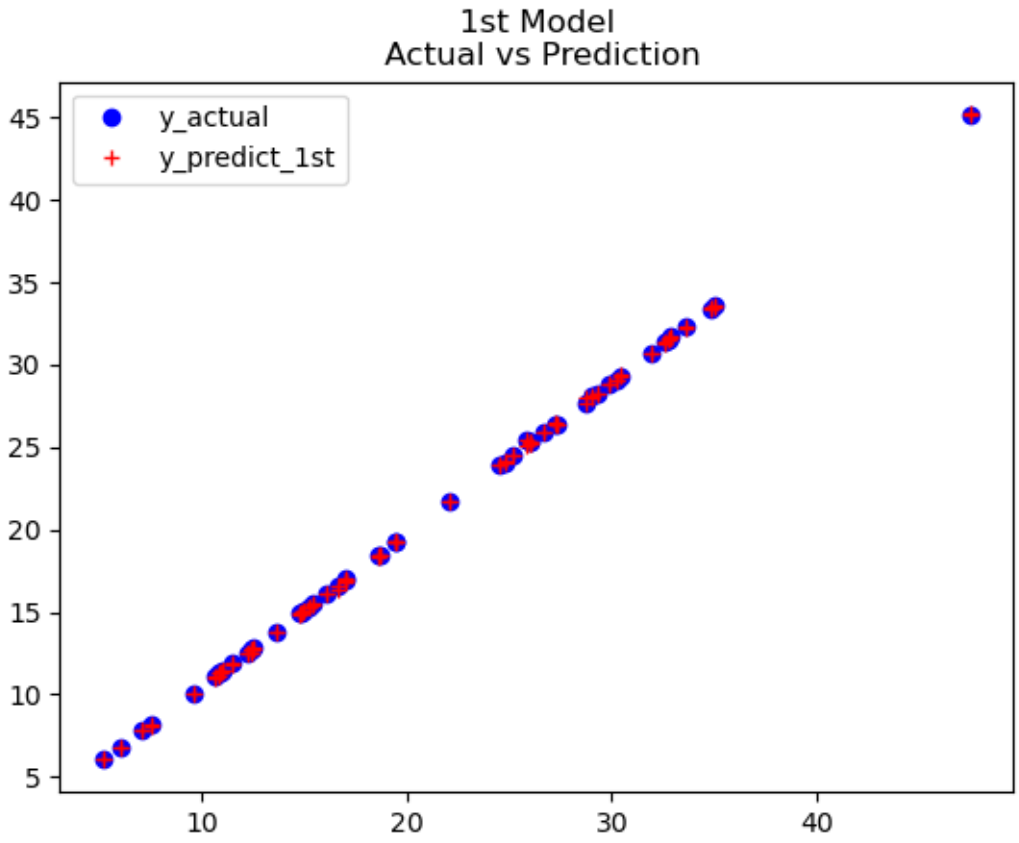
[6]:
```
# Calculate Prediction Error of the first model
e_1, y_pred_first = get_prediction_error(firstFittedModel, X_test)
print('Prediction Error e_1 =', e_1)
```

```
Prediction Error e_1 = 0.753742369836337
```

[7]:
```
import matplotlib.pyplot as plt

def plot_act_vs_pred(title, label_pred, x_test, y_pred):
    fig, ax = plt.subplots()
    ax.plot(test['siri'], y_actual, "bo", label="y_actual")
    ax.plot(np.hstack((test['siri'], x_test['siri'])), np.hstack((y_actual,␣
 ↪y_pred)), "r+", label=label_pred)
    plt.title(title)
    ax.legend(loc="best")
```

[8]:
```
# Plot Actual vs Prediction of the first model
plot_act_vs_pred('1st Model\n Actual vs Prediction', 'y_predict_1st', X_test,␣
 ↪y_pred_first)
```



(d) Now consider a second model which uses the features indicated in part (a), with the only difference that density is replaced by inverse density. In other words, your model has the the following parametric form:brozek $= \beta_0 +$ **siri** $+ \frac{\beta_2}{density} +$ **age** $+ \ldots + \beta_{17}$ **wrist Repeat the steps in part (a) and report the values.**

[9]:
```
X_train_second = pd.DataFrame({'intercept': np.ones(train.shape[0]),
                    'siri':    train['siri'],
                    'density': 1/train['density'],
                    'age':     train['age'],
                    'weight':  train['weight'],
                    'height':  train['height'],
                    'adipos':  train['adipos'],
                    'free':    train['free'],
                    'neck':    train['neck'],
                    'chest':   train['chest'],
                    'abdom':   train['abdom'],
                    'hip':     train['hip'],
                    'thigh':   train['thigh'],
                    'knee':    train['knee'],
                    'ankle':   train['ankle'],
                    'biceps':  train['biceps'],
                    'forearm': train['forearm'],
                    'wrist':   train['wrist']})
secondFittedModel = sm.OLS(y_train, X_train_second).fit()
secondFittedModel.summary()
```

[9]:

Dep. Variable:	brozek	**R-squared:**	0.999
Model:	OLS	**Adj. R-squared:**	0.999
Method:	Least Squares	**F-statistic:**	1.852e+04
Date:	Thu, 18 Apr 2024	**Prob (F-statistic):**	1.00e-284
Time:	18:26:29	**Log-Likelihood:**	60.917
No. Observations:	200	**AIC:**	-85.83
Df Residuals:	182	**BIC:**	-26.46
Df Model:	17		
Covariance Type:	nonrobust		

	coef	std err	t	P> |t|	[0.025	0.975]
intercept	-8.0827	4.536	-1.782	0.076	-17.032	0.867
siri	0.8865	0.013	67.497	0.000	0.861	0.912
density	10.3645	4.867	2.129	0.035	0.761	19.968
age	-0.0007	0.002	-0.423	0.673	-0.004	0.003
weight	0.0109	0.005	2.287	0.023	0.001	0.020
height	0.0005	0.005	0.094	0.925	-0.009	0.010
adipos	-0.0206	0.016	-1.320	0.189	-0.051	0.010
free	-0.0125	0.006	-2.161	0.032	-0.024	-0.001
neck	-0.0038	0.011	-0.332	0.740	-0.026	0.019
chest	0.0031	0.005	0.583	0.561	-0.008	0.014
abdom	0.0005	0.005	0.098	0.922	-0.010	0.011
hip	-0.0039	0.007	-0.529	0.598	-0.019	0.011
thigh	0.0199	0.008	2.580	0.011	0.005	0.035
knee	-0.0305	0.013	-2.386	0.018	-0.056	-0.005
ankle	0.0036	0.010	0.352	0.725	-0.017	0.024
biceps	-0.0158	0.009	-1.850	0.066	-0.033	0.001
forearm	0.0193	0.010	1.858	0.065	-0.001	0.040
wrist	0.0339	0.027	1.239	0.217	-0.020	0.088

Omnibus:	144.182	**Durbin-Watson:**	1.875
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	8277.916
Skew:	1.977	**Prob(JB):**	0.00
Kurtosis:	34.268	**Cond. No.**	1.58e+05

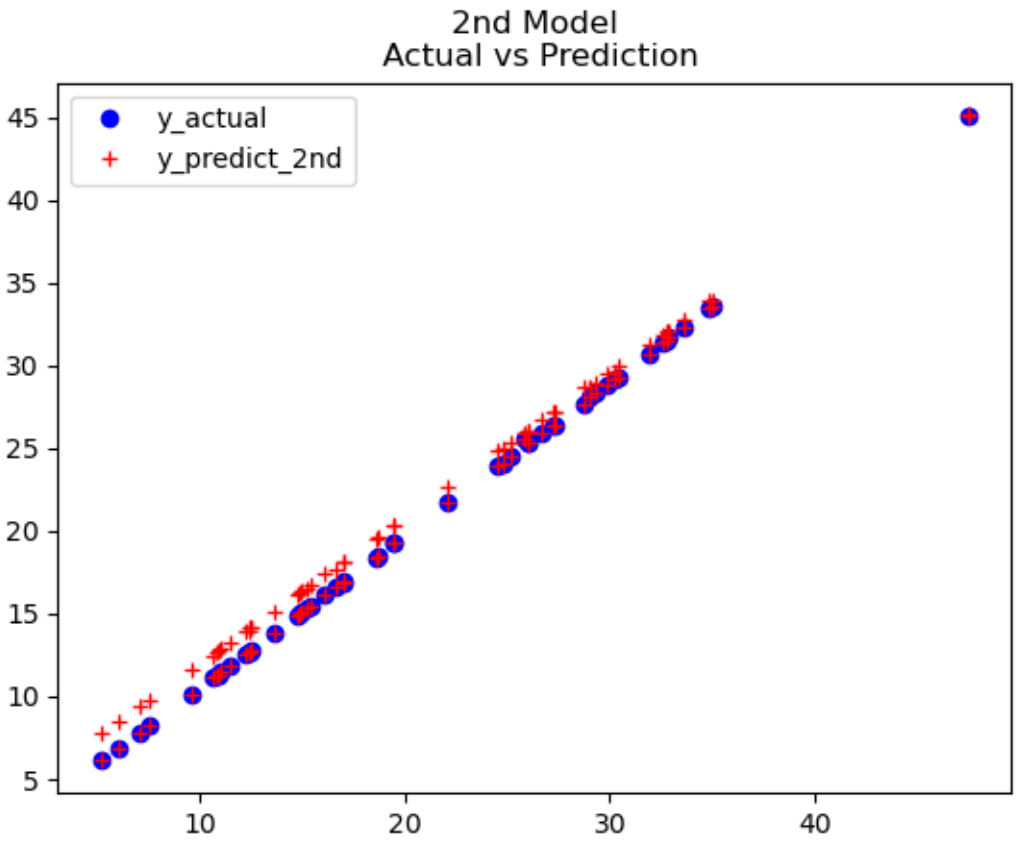
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.58e+05. This might indicate that there are strong multicollinearity or other numerical problems.

(e) Use a similar formulation in part (c) and calculate the test error (we will call this test error e_2 which corresponds to our second model).

[10]:
```
# Calculate the Prediction Error of the second model
e_2, y_pred_second = get_prediction_error(secondFittedModel, X_test)
print('Second Prediction Error e_2 =', e_2)
```

```
Second Prediction Error e_2 = 7.766138616196848
```

[11]:
```
# Plot Actual vs Prediction of the second model
plot_act_vs_pred('2nd Model\n Actual vs Prediction', 'y_predict_2nd', X_test,␣
↪y_pred_second)
```



(f) Now consider a third model which only uses siri and density as the features, but follows a para- metric formulation as: brozek $= \beta_0 + \beta_1$ **siri** $+ \beta_2\ siri^2 + \frac{\beta_3}{density} + \beta_4$ **density Repeat the steps in part (a) and report the values.**

[12]:
```
X_train_third = pd.DataFrame({'intercept': np.ones(train.shape[0]),
                  'siri':         train['siri'],
                  'siri_square': train['siri'] ** 2,
                  'inv_density': 1/train['density'],
                  'density':     train['density']})

thirdFittedModel = sm.OLS(y_train, X_train_third).fit()
thirdFittedModel.summary()
```

[12]:

Dep. Variable:	brozek	**R-squared:**	0.999
Model:	OLS	**Adj. R-squared:**	0.999
Method:	Least Squares	**F-statistic:**	7.648e+04
Date:	Thu, 18 Apr 2024	**Prob (F-statistic):**	2.51e-310
Time:	18:26:30	**Log-Likelihood:**	51.125
No. Observations:	200	**AIC:**	-92.25
Df Residuals:	195	**BIC:**	-75.76
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
intercept	-1298.6590	567.083	-2.290	0.023	-2417.062	-180.256
siri	0.9618	0.027	35.651	0.000	0.909	1.015
siri_square	-0.0030	0.001	-2.275	0.024	-0.006	-0.000
inv_density	706.0142	305.636	2.310	0.022	103.237	1308.791
density	598.1825	262.950	2.275	0.024	79.592	1116.773

Omnibus:	138.737	**Durbin-Watson:**	1.955
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	12413.875
Skew:	1.696	**Prob(JB):**	0.00
Kurtosis:	41.447	**Cond. No.**	2.68e+07

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.68e+07. This might indicate that there are strong multicollinearity or other numerical problems.

(g) Based on = 0.05 and the calculated p-values, which features seem problematic?

All the p-values of the features are less than 0.05

feature	p-value
intercept	0.023
siri	0.00
$siri^2$	0.024
$\frac{1}{density}$	0.022
density	0.024

(h) Repeat part (c) for this model and call the error e_3.

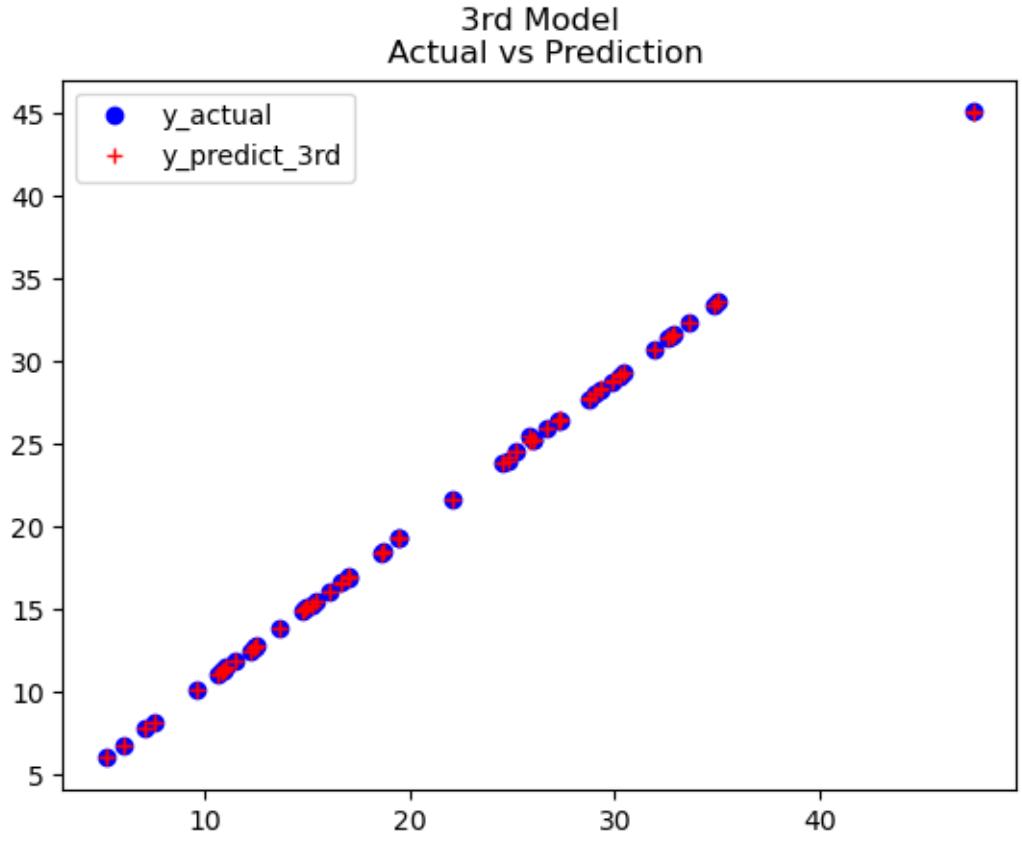
[13]:
```
X_test_third = pd.DataFrame({'intercept': np.ones(test.shape[0]),
                   'siri':         test['siri'],
                   'siri_square': test['siri'] ** 2,
                   'inv_density': 1/test['density'],
                   'density':     test['density']})

# Calculate the Prediction Error of the third model
e_3, y_pred_third = get_prediction_error(thirdFittedModel, X_test_third)
```

```
print('Second Prediction Error e_3 =', e_3)
```

```
Second Prediction Error e_3 = 0.563361670359713
```

[14]:
```
# Plot Actual vs Prediction of the second model
plot_act_vs_pred('3rd Model\n Actual vs Prediction', 'y_predict_3rd',␣
  ↪X_test_third, y_pred_third)
```



(i) Based on the values e_1, e_2 and e_3, and the model formulations, which model would you pick and why (state two reasons)?

I will pick the third model. (1) It has the lowest prediction error *0.563361670359713* so it has the highest prediction accuracy.(2) The R-squared values of the three models are the same but the thrid model has the lowest p-values *2.51e-310*.

Q3

April 18, 2024

In the previous question we observed some redundancy in the features. We would like to try some feature selection heuristic in this question. Consider the same dataset as question 2 (fat.csv), where **brozek** is the response variable and the other 17 columns are the model features. Follow this steps below.

– Form an extended version of the dataset, by appending two more columns. One column corresponding to $siri^2$ and one column corresponding to $\frac{1}{density}$. Your extended dataset should now have 20 columns, where the first column is brozek and used as the response variable, 17 columns identical to the original fat.csv data set, and columns 19 and 20 with the values $siri^2$ and $\frac{1}{density}$, respectively. density We will refer to this dataset as the *extended dataset*.

– In a similar way as question 2, split the extended dataset into two sets. Set 1 includes the first 200 rows of the data (do not count the row associated with the feature/response names), and set 2, which includes the last 52 rows of the data. Name the first set **train** and the second set **test**.

[1]:
```
import numpy as np
import pandas as pd
from matplotlib.pyplot import subplots
import statsmodels.api as sm
from ISLP import load_data
from ISLP.models import (ModelSpec as MS,
                         summarize)
```

[2]:
```
Fat = pd.read_csv("fat.csv")
train,test = np.split(Fat,[int(200)])
y = train['brozek']
X_train_full = pd.DataFrame({'intercept': np.ones(train.shape[0]),
                  'siri':         train['siri'],
                  'density':      train['density'],
                  'age':          train['age'],
                  'weight':       train['weight'],
                  'height':       train['height'],
                  'adipos':       train['adipos'],
                  'free':         train['free'],
                  'neck':         train['neck'],
                  'chest':        train['chest'],
                  'abdom':        train['abdom'],
                  'hip':          train['hip'],
                  'thigh':        train['thigh'],
```

```
                'knee':          train['knee'],
                'ankle':         train['ankle'],
                'biceps':        train['biceps'],
                'forearm':       train['forearm'],
                'wrist':         train['wrist'],
                'siri_squared': train['siri'] ** 2,
                'inv_density':   1/train['density']})
```

(a) Use the training data to fit a model of the following form brozek = $\beta_0 + \beta_1$ siri + ... + β_{17} wrist + β_{18} $siri^2$ + β_{19} $\frac{1}{density}$ report the fitted parameters, the 95% confidence interval for each estimated parameter and the p-values. What is the R2 value?

[3]:
```
fullFittedModel = sm.OLS(y, X_train_full).fit()
fullFittedModel.summary()
```

[3]:

Dep. Variable:	brozek	**R-squared:**	0.999
Model:	OLS	**Adj. R-squared:**	0.999
Method:	Least Squares	**F-statistic:**	1.703e+04
Date:	Thu, 18 Apr 2024	**Prob (F-statistic):**	6.09e-282
Time:	18:33:31	**Log-Likelihood:**	64.717
No. Observations:	200	**AIC:**	-89.43
Df Residuals:	180	**BIC:**	-23.47
Df Model:	19		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
intercept	-950.8057	580.610	-1.638	0.103	-2096.483	194.871
siri	0.9305	0.029	32.063	0.000	0.873	0.988
density	436.7337	269.081	1.623	0.106	-94.225	967.693
age	-0.0007	0.002	-0.414	0.680	-0.004	0.003
weight	0.0162	0.006	2.809	0.006	0.005	0.028
height	-0.0005	0.005	-0.096	0.924	-0.010	0.009
adipos	-0.0235	0.016	-1.505	0.134	-0.054	0.007
free	-0.0204	0.007	-2.780	0.006	-0.035	-0.006
neck	-0.0018	0.011	-0.163	0.870	-0.024	0.020
chest	0.0056	0.005	1.026	0.306	-0.005	0.016
abdom	-0.0006	0.005	-0.109	0.913	-0.011	0.010
hip	0.0008	0.008	0.106	0.916	-0.014	0.016
thigh	0.0174	0.008	2.264	0.025	0.002	0.033
knee	-0.0290	0.013	-2.296	0.023	-0.054	-0.004
ankle	0.0061	0.010	0.591	0.555	-0.014	0.026
biceps	-0.0169	0.008	-2.011	0.046	-0.034	-0.000
forearm	0.0219	0.010	2.110	0.036	0.001	0.042
wrist	0.0343	0.027	1.270	0.206	-0.019	0.088
siri_squared	-0.0026	0.001	-1.956	0.052	-0.005	2.26e-05
inv_density	518.6225	312.993	1.657	0.099	-98.985	1136.230

Omnibus:	98.592	**Durbin-Watson:**	1.907
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6766.061
Skew:	0.902	**Prob(JB):**	0.00
Kurtosis:	31.437	**Cond. No.**	3.19e+07

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.19e+07. This might indicate that there are strong multicollinearity or other numerical problems.

[4]:
```
X_test_full = pd.DataFrame({'intercept': np.ones(test.shape[0]),
                    'siri':    test['siri'],
                    'density': test['density'],
                    'age':     test['age'],
                    'weight':  test['weight'],
                    'height':  test['height'],
                    'adipos':  test['adipos'],
                    'free':    test['free'],
                    'neck':    test['neck'],
                    'chest':   test['chest'],
                    'abdom':   test['abdom'],
                    'hip':     test['hip'],
                    'thigh':   test['thigh'],
                    'knee':    test['knee'],
                    'ankle':   test['ankle'],
                    'biceps':  test['biceps'],
                    'forearm': test['forearm'],
                    'wrist':   test['wrist'],
                    'siri_squared': test['siri'] ** 2,
                    'inv_density':   1/test['density']})
```

(b) Use the test data to calculate the test error (similar to the formulation in part (c) of the previous question), and call it e_{full}.

[5]:
```
# Get prediction error by model
def get_prediction_error(model, X_pred):
    # the actual value from test dataset
    y_actual = np.array(test['brozek'].values)
    # the value predicted by the full-features model
    y_prediction = np.array(model.predict(X_pred))
    return np.sqrt(sum(np.square(y_actual - y_prediction)))
```

[6]:
```
# Calculate the prediction error of full-featured model
print('Prediction Error e_full =', get_prediction_error(fullFittedModel,␣
 ↪X_test_full))
```

```
Prediction Error e_full = 0.8565466791992765
```

(c) Let's run a heuristic scheme to perform feature selection (the method is called backward selection and described on page 79 of your textbook, also on the slides). Start with the full model (the model containing all 19 features of the extended dataset) and drop the feature with the highest p-value (or the second largest if the largest p-value is for the intercept), then redo the modeling and drop the next feature with the highest p-value, and continue dropping until all p-values are small and you are left with a set of important features. Implement this approach and stop when all p-values are below 0.03. Which features are selected as the most important ones when your code stops?

[7]:
```
# Extract the problematic features that has p-value greater than 0.03, among␣
 ↪all the p-values.
def get_problematic_pValues(model):
    pv = model.pvalues
    return pv[pv > 0.03]
```

[8]:
```
# Extract the problematic features
pv_problematic = get_problematic_pValues(fullFittedModel)
# print(pv_problematic,"\n")

# Training set for iteration
X_train_new = X_train_full

# Run the loop if there is any p-value greater than 0.03
while (pv_problematic > 0.03).any():

    # Find the most problematic feature
    fea_maxP = pv_problematic.idxmax()
    # print(fea_maxP,": ", pv_problematic[fea_maxP])

    # Drop that feature and get new training data
    X_train_new = X_train_new.drop(fea_maxP, axis = 1)

    # Retrain the model with new data
    newFittedModel = sm.OLS(y, X_train_new).fit()

    # Get new p-valuses from new model
    pv_problematic = get_problematic_pValues(newFittedModel)

newFittedModel.summary()
```

[8]:

Dep. Variable:	brozek	**R-squared (uncentered):**	1.000
Model:	OLS	**Adj. R-squared (uncentered):**	1.000
Method:	Least Squares	**F-statistic:**	5.569e+05
Date:	Thu, 18 Apr 2024	**Prob (F-statistic):**	0.00
Time:	18:33:31	**Log-Likelihood:**	51.369
No. Observations:	200	**AIC:**	-94.74
Df Residuals:	196	**BIC:**	-81.55
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
siri	0.9222	0.002	474.600	0.000	0.918	0.926
thigh	0.0142	0.005	3.101	0.002	0.005	0.023
knee	-0.0262	0.010	-2.596	0.010	-0.046	-0.006
inv_density	1.5192	0.267	5.698	0.000	0.993	2.045

Omnibus:	187.007	**Durbin-Watson:**	1.919
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	13874.811
Skew:	2.978	**Prob(JB):**	0.00
Kurtosis:	43.367	**Cond. No.**	1.47e+03

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 1.47e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

***siri*, *thigh*, *knee*, $\frac{1}{density}$ are the most important features after backward selection.**

(d) Apply the model developed in part (c) to the test data and call the error e_{sel}.

[9]:
```
# Feature variables from test set
X_test_sel = pd.DataFrame({'siri': test['siri'], 'thigh': test['thigh'], 'knee':
  ↪ test['knee'], 'inv_density': 1/test['density']})

# Calculate the prediction error of feature-selection model
print('Prediction Error e_sel =', get_prediction_error(newFittedModel,␣
  ↪X_test_sel))
```

```
Prediction Error e_sel = 0.6670935385386173
```

(e) Compare e_{full} and e_{sel}. Does the feature selection scheme seem to reduce overfitting?

Yes, it does reduce overfitting. By removing less important features, backward selection helps focus the model on the most relevant predictors. This reduces the complexity of the model and decrease the chance of fitting too close to training data.

(f) Compare e_{sel} with e_3 from part (h) of question 2. In terms of the test accuracy does your feature selection scheme seem to find the best model?

e_{sel} (*0.6670935385386173*) is greater than e_3 (*0.563361670359713*), therefore, the feature selection scheme seems not to get the best model in this case. The reason might be some useful features that seemed not important were removed too early. This can lead to a loss of valuable information that could be useful in conjunction with other features.