Module2_v2_2022_05_13_03_52_12

June 1, 2022

0.0.1 Grading

The final score that you will receive for your programming assignment is generated in relation to the total points set in your programming assignment item—not the total point value in the nbgrader notebook. When calculating the final score shown to learners, the programming assignment takes the percentage of earned points vs. the total points provided by nbgrader and returns a score matching the equivalent percentage of the point value for the programming assignment. **DO NOT CHANGE VARIABLE OR METHOD SIGNATURES** The autograder will not work properly if your change the variable or method signatures.

0.0.2 Validate Button

Please note that this assignment uses nbgrader to facilitate grading. You will see a **validate button** at the top of your Jupyter notebook. If you hit this button, it will run tests cases for the lab that aren't hidden. It is good to use the validate button before submitting the lab. Do know that the labs in the course contain hidden test cases. The validate button will not let you know whether these test cases pass. After submitting your lab, you can see more information about these hidden test cases in the Grader Output. ***Cells with longer execution times will cause the validate button to time out and freeze. Please know that if you run into Validate time-outs, it will not affect the final submission grading.***

[200]:
```
%matplotlib inline
import numpy as np
import scipy as sp
import scipy.stats as stats
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Set color map to have light blue background
sns.set()
import statsmodels.formula.api as smf
import statsmodels.api as sm
```

N.B.: I recommend that you use the `statsmodel` library to do the regression analysis as opposed to *e.g.* `sklearn`. The `sklearn` library is great for advanced topics, but it's easier to get lost in a sea of details and it's not needed for these problems.

1 1. Polynomial regression using MPG data [25 pts, Peer Review]

We will be using Auto MPG data from UCI datasets (https://archive.ics.uci.edu/ml/datasets/Auto+MPG) to study polynomial regression.

[201]:
```
columns =␣
↪['mpg','cylinders','displacement','horsepower','weight','acceleration','model_year','origin
df = pd.read_csv("data/auto-mpg.data", header=None, delimiter=r"\s+",␣
↪names=columns)
print(df.info())
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    object
 4   weight        398 non-null    float64
 5   acceleration  398 non-null    float64
 6   model_year    398 non-null    int64
 7   origin        398 non-null    int64
 8   car_name      398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
None
```

[201]:
```
              mpg   cylinders  displacement       weight  acceleration  \
count  398.000000  398.000000    398.000000   398.000000    398.000000
mean    23.514573    5.454774    193.425879  2970.424623     15.568090
std      7.815984    1.701004    104.269838   846.841774      2.757689
min      9.000000    3.000000     68.000000  1613.000000      8.000000
25%     17.500000    4.000000    104.250000  2223.750000     13.825000
50%     23.000000    4.000000    148.500000  2803.500000     15.500000
75%     29.000000    8.000000    262.000000  3608.000000     17.175000
max     46.600000    8.000000    455.000000  5140.000000     24.800000

       model_year      origin
count  398.000000  398.000000
mean    76.010050    1.572864
std      3.697627    0.802055
min     70.000000    1.000000
25%     73.000000    1.000000
50%     76.000000    1.000000
```

```
75%      79.000000    2.000000
max      82.000000    3.000000
```

[202]:
```
df.head(10)
```

[202]:
```
    mpg  cylinders  displacement horsepower  weight  acceleration  model_year  \
0  18.0          8         307.0      130.0  3504.0          12.0          70
1  15.0          8         350.0      165.0  3693.0          11.5          70
2  18.0          8         318.0      150.0  3436.0          11.0          70
3  16.0          8         304.0      150.0  3433.0          12.0          70
4  17.0          8         302.0      140.0  3449.0          10.5          70
5  15.0          8         429.0      198.0  4341.0          10.0          70
6  14.0          8         454.0      220.0  4354.0           9.0          70
7  14.0          8         440.0      215.0  4312.0           8.5          70
8  14.0          8         455.0      225.0  4425.0          10.0          70
9  15.0          8         390.0      190.0  3850.0           8.5          70

   origin                   car_name
0       1  chevrolet chevelle malibu
1       1          buick skylark 320
2       1         plymouth satellite
3       1              amc rebel sst
4       1                ford torino
5       1           ford galaxie 500
6       1           chevrolet impala
7       1          plymouth fury iii
8       1           pontiac catalina
9       1         amc ambassador dpl
```

1.0.1 1a) Clean the data (fix data types and remove null or undefined values) and drop the column car_name. [5 pts]

Replace the data frame with the cleaned data frame. Do not change the column names, and do not add new columns.

[203]:
```
df.drop('car_name', axis=1, inplace=True)
```

[204]:
```
# replace data frame with cleaned data frame
# fix data types, remove null or undefined values, drop the column car_name
# NOTE: do not change the column names or add new columns
# your code here

#Question mark in horsepower
for col in df:
    print(df[col].name)
    print(df[col].unique())
```

```python
    print('----------')
```

```
mpg
[18.  15.  16.  17.  14.  24.  22.  21.  27.  26.  25.  10.  11.   9.
 28.  19.  12.  13.  23.  30.  31.  35.  20.  29.  32.  33.  17.5 15.5
 14.5 22.5 24.5 18.5 29.5 26.5 16.5 31.5 36.  25.5 33.5 20.5 30.5 21.5
 43.1 36.1 32.8 39.4 19.9 19.4 20.2 19.2 25.1 20.6 20.8 18.6 18.1 17.7
 27.5 27.2 30.9 21.1 23.2 23.8 23.9 20.3 21.6 16.2 19.8 22.3 17.6 18.2
 16.9 31.9 34.1 35.7 27.4 25.4 34.2 34.5 31.8 37.3 28.4 28.8 26.8 41.5
 38.1 32.1 37.2 26.4 24.3 19.1 34.3 29.8 31.3 37.  32.2 46.6 27.9 40.8
 44.3 43.4 36.4 44.6 40.9 33.8 32.7 23.7 23.6 32.4 26.6 25.8 23.5 39.1
 39.  35.1 32.3 37.7 34.7 34.4 29.9 33.7 32.9 31.6 28.1 30.7 24.2 22.4
 34.  38.  44. ]
----------
cylinders
[8 4 6 3 5]
----------
displacement
[307.  350.  318.  304.  302.  429.  454.  440.  455.  390.  383.  340.
 400.  113.  198.  199.  200.   97.  110.  107.  104.  121.  360.  140.
  98.  232.  225.  250.  351.  258.  122.  116.   79.   88.   71.   72.
  91.   97.5  70.  120.   96.  108.  155.   68.  114.  156.   76.   83.
  90.  231.  262.  134.  119.  171.  115.  101.  305.   85.  130.  168.
 111.  260.  151.  146.   80.   78.  105.  131.  163.   89.  267.   86.
 183.  141.  173.  135.   81.  100.  145.  112.  181.  144. ]
----------
horsepower
['130.0' '165.0' '150.0' '140.0' '198.0' '220.0' '215.0' '225.0' '190.0'
 '170.0' '160.0' '95.00' '97.00' '85.00' '88.00' '46.00' '87.00' '90.00'
 '113.0' '200.0' '210.0' '193.0' '?' '100.0' '105.0' '175.0' '153.0'
 '180.0' '110.0' '72.00' '86.00' '70.00' '76.00' '65.00' '69.00' '60.00'
 '80.00' '54.00' '208.0' '155.0' '112.0' '92.00' '145.0' '137.0' '158.0'
 '167.0' '94.00' '107.0' '230.0' '49.00' '75.00' '91.00' '122.0' '67.00'
 '83.00' '78.00' '52.00' '61.00' '93.00' '148.0' '129.0' '96.00' '71.00'
 '98.00' '115.0' '53.00' '81.00' '79.00' '120.0' '152.0' '102.0' '108.0'
 '68.00' '58.00' '149.0' '89.00' '63.00' '48.00' '66.00' '139.0' '103.0'
 '125.0' '133.0' '138.0' '135.0' '142.0' '77.00' '62.00' '132.0' '84.00'
 '64.00' '74.00' '116.0' '82.00']
----------
weight
[3504. 3693. 3436. 3433. 3449. 4341. 4354. 4312. 4425. 3850. 3563. 3609.
 3761. 3086. 2372. 2833. 2774. 2587. 2130. 1835. 2672. 2430. 2375. 2234.
 2648. 4615. 4376. 4382. 4732. 2264. 2228. 2046. 2634. 3439. 3329. 3302.
 3288. 4209. 4464. 4154. 4096. 4955. 4746. 5140. 2962. 2408. 3282. 3139.
 2220. 2123. 2074. 2065. 1773. 1613. 1834. 1955. 2278. 2126. 2254. 2226.
 4274. 4385. 4135. 4129. 3672. 4633. 4502. 4456. 4422. 2330. 3892. 4098.
 4294. 4077. 2933. 2511. 2979. 2189. 2395. 2288. 2506. 2164. 2100. 4100.
```

```
 3988. 4042. 3777. 4952. 4363. 4237. 4735. 4951. 3821. 3121. 3278. 2945.
 3021. 2904. 1950. 4997. 4906. 4654. 4499. 2789. 2279. 2401. 2379. 2124.
 2310. 2472. 2265. 4082. 4278. 1867. 2158. 2582. 2868. 3399. 2660. 2807.
 3664. 3102. 2875. 2901. 3336. 2451. 1836. 2542. 3781. 3632. 3613. 4141.
 4699. 4457. 4638. 4257. 2219. 1963. 2300. 1649. 2003. 2125. 2108. 2246.
 2489. 2391. 2000. 3264. 3459. 3432. 3158. 4668. 4440. 4498. 4657. 3907.
 3897. 3730. 3785. 3039. 3221. 3169. 2171. 2639. 2914. 2592. 2702. 2223.
 2545. 2984. 1937. 3211. 2694. 2957. 2671. 1795. 2464. 2572. 2255. 2202.
 4215. 4190. 3962. 3233. 3353. 3012. 3085. 2035. 3651. 3574. 3645. 3193.
 1825. 1990. 2155. 2565. 3150. 3940. 3270. 2930. 3820. 4380. 4055. 3870.
 3755. 2045. 1945. 3880. 4060. 4140. 4295. 3520. 3425. 3630. 3525. 4220.
 4165. 4325. 4335. 1940. 2740. 2755. 2051. 2075. 1985. 2190. 2815. 2600.
 2720. 1800. 2070. 3365. 3735. 3570. 3535. 3155. 2965. 3430. 3210. 3380.
 3070. 3620. 3410. 3445. 3205. 4080. 2560. 2230. 2515. 2745. 2855. 2405.
 2830. 3140. 2795. 2135. 3245. 2990. 2890. 3265. 3360. 3840. 3725. 3955.
 3830. 4360. 4054. 3605. 1925. 1975. 1915. 2670. 3530. 3900. 3190. 3420.
 2200. 2150. 2020. 2595. 2700. 2556. 2144. 1968. 2120. 2019. 2678. 2870.
 3003. 3381. 2188. 2711. 2434. 2110. 2800. 2085. 2335. 2950. 3250. 1850.
 2145. 1845. 2910. 2420. 2500. 2905. 2290. 2490. 2635. 2620. 2725. 2385.
 1755. 1875. 1760. 2050. 2215. 2380. 2320. 2210. 2350. 2615. 3230. 3160.
 2900. 3415. 3060. 3465. 2605. 2640. 2575. 2525. 2735. 2865. 3035. 1980.
 2025. 1970. 2160. 2205. 2245. 1965. 1995. 3015. 2585. 2835. 2665. 2370.
 2790. 2295. 2625.]
----------
acceleration
[12.  11.5 11.  10.5 10.   9.   8.5  8.   9.5 15.  15.5 16.  14.5 20.5
 17.5 12.5 14.  13.5 18.5 19.  13.  19.5 18.  17.  23.5 16.5 21.  16.9
 14.9 17.7 15.3 13.9 12.8 15.4 17.6 22.2 22.1 14.2 17.4 16.2 17.8 12.2
 16.4 13.6 15.7 13.2 21.9 16.7 12.1 14.8 18.6 16.8 13.7 11.1 11.4 18.2
 15.8 15.9 14.1 21.5 14.4 19.4 19.2 17.2 18.7 15.1 13.4 11.2 14.7 16.6
 17.3 15.2 14.3 20.1 24.8 11.3 12.9 18.8 18.1 17.9 21.7 23.7 19.9 21.8
 13.8 12.6 16.1 20.7 18.3 20.4 19.6 17.1 15.6 24.6 11.6]
----------
model_year
[70 71 72 73 74 75 76 77 78 79 80 81 82]
----------
origin
[1 3 2]
----------
```

[205]:
```
df['horsepower'] = df['horsepower'].replace('?', np.NaN).astype(float)
df.dropna(inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 392 entries, 0 to 397
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
```

```
---  ------        --------------  -----
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        392 non-null    float64
 5   acceleration  392 non-null    float64
 6   model_year    392 non-null    int64
 7   origin        392 non-null    int64
dtypes: float64(5), int64(3)
memory usage: 27.6 KB
```

[206]:
```
# this cell will test that you properly cleaned the dataframe
```

1.0.2 1b) Fit a simple linear regression model with a feature that maximizes R^2. [5 pts]

Which feature is the best predictor, and the resulting r-squared value? Update your answer below.

[207]:
```
# your code here

# best_predictor=''
# best_r_squared=0
dependent_variable = 'mpg ~ '
predictor = ' '
col_names = []
r_squareds = []
for col in df.columns:
    if col == 'mpg':
        continue
    predictor = col
    model = smf.ols(formula = dependent_variable + predictor, data = df)
    fit = model.fit()
    col_names.append(col)
    r_squareds.append(fit.rsquared)

plt.figure(figsize=(15,15))
plt.plot(col_names, r_squareds)
best_predictor='weight'
best_r_squared= max(r_squareds)
best_r_squared
```

[207]: 0.6926304331206254

[208]:
```
# this cell will test best_predictor and best_r_squared
```

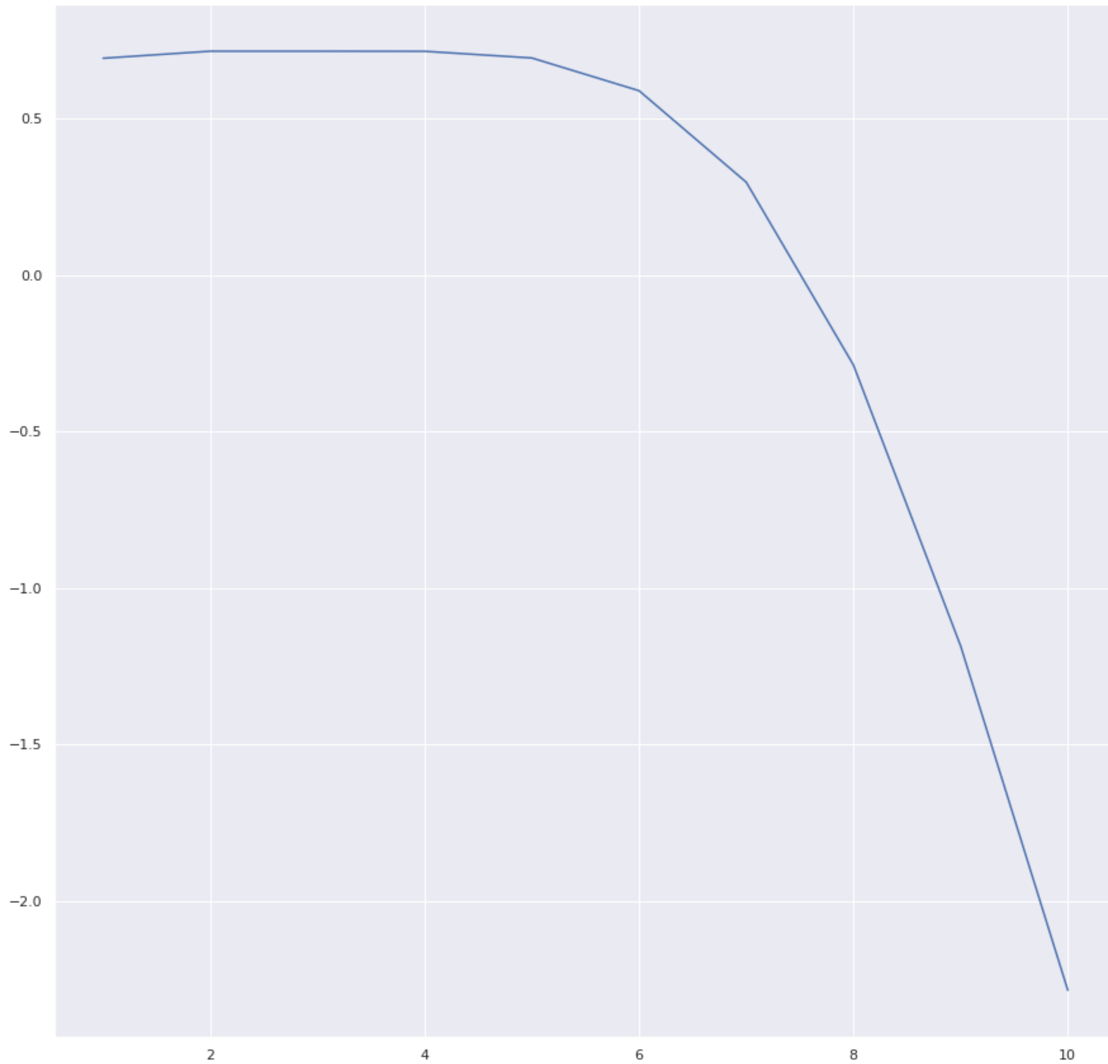
1.0.3 1c) Using the feature found above (without normalizing), fit polynomial regression up to N=10 and report R^2. Which polynomial degree gives the best result? [10 pts]

Hint: For N-degree polynomial fit, you may have to include all orders upto N. Use a for loop instead of running it manually. The `statsmodels.formula.api` formula string can understand `np.power(x,n)` function to include a feature representing x^n.

[209]:
```
# return updated best_degree and best_r_squared
# best_degree = 1
# best_r_squared = 0
```

```python
# your code here
x='weight'
power = []
r_squareds = []
formula = 'mpg ~'
for i in range(1,11):
    formula = formula + ' + np.power(weight, ' + str(i) + ')'
    model = smf.ols(formula = formula, data = df)
    fit = model.fit()
    power.append(i)
    r_squareds.append(fit.rsquared)

plt.figure(figsize=(15,15))
plt.plot(power, r_squareds)
best_degree = 3
best_r_squared = max(r_squareds)
print(best_r_squared)
print(r_squareds.index(best_r_squared) + 1)
```

```
0.715149595486925
3
```



[210]:
```
# this cell tests best_degree and best_r_squared
```

1.0.4 1d) Now, let's make a new feature called 'weight_norm' which is weight normalized by the mean value. [5 pts]

Run training with polynomial models with polynomial degrees up to 20. Print out each polynomial degree and R^2 value. What do you observe from the result? What are the best_degree and best_r_qaured just based on R^2 value? Inspect model summary from each model. What is the highest order model that makes sense (fill the value for the sound_degree)?

[211]:
```
# best_degree = 1
# best_r_squared = 0
# sound_degree = 1
```

```python
df['weight_norm'] = df['weight']/df['weight'].mean()
# your code here
formula = 'mpg ~'
for i in range(1,21):
    formula = formula + ' + np.power(weight_norm, ' + str(i) + ')'
    model = smf.ols(formula = formula, data = df)
    fit = model.fit()
    print('Degree is ' + str(i))
    print('R-squared is ' + str(fit.rsquared))
    print('----------------')
```

```
Degree is 1
R-squared is 0.6926304331206254
----------------
Degree is 2
R-squared is 0.7151475557845139
----------------
Degree is 3
R-squared is 0.7151495954869258
----------------
Degree is 4
R-squared is 0.7154806032756431
----------------
Degree is 5
R-squared is 0.7160964869848916
----------------
Degree is 6
R-squared is 0.7165638483082104
----------------
Degree is 7
R-squared is 0.7177879568842087
----------------
Degree is 8
R-squared is 0.7177992979709948
----------------
Degree is 9
R-squared is 0.7182083307102388
----------------
Degree is 10
R-squared is 0.7198912805389772
----------------
Degree is 11
R-squared is 0.7209101742520523
----------------
Degree is 12
R-squared is 0.7209276395637563
```

```
----------------
Degree is 13
R-squared is 0.7227918788934491
----------------
Degree is 14
R-squared is 0.7240041787167142
----------------
Degree is 15
R-squared is 0.7238303796561847
----------------
Degree is 16
R-squared is 0.7242829281892726
----------------
Degree is 17
R-squared is 0.7243902195110014
----------------
Degree is 18
R-squared is 0.7244188646420426
----------------
Degree is 19
R-squared is 0.7244317942203697
----------------
Degree is 20
R-squared is 0.7245259039513001
----------------
```

[212]:
```
best_degree = 20
best_r_squared = 0.7245259039513001
sound_degree = 2 #Greatest jump in r-squared
```

[213]:
```
# tests best_degree, best_r_squared, and sound_degree
```

1.0.5 TODO:

Open the Peer Review assignment for this week to answer a question for section 1d.

According to the plot, the models with a degree of 3 or higher have high p-values (insignificant) p-values for the f-test.

2 2. Multi-Linear Regression [15 pts, Peer Review]

In the following problem, you will construct a simple multi-linear regression model, identify interaction terms and use diagnostic plots to identify outliers in the data. The original problem is as described by John Verzani in the excellent tutorial 'SimplR' on the R statistics language and uses data from the 2000 presidential election in Florida. The problem is interesting because it contains a small number of highly leveraged points that influence the model.

[214]:
```python
votes = pd.read_csv('data/fl2000.txt', delim_whitespace=True, comment='#')
votes = votes[['county', 'Bush', 'Gore', 'Nader', 'Buchanan']]
votes.describe(include='all')
```

[214]:

	county	Bush	Gore	Nader	Buchanan
count	67	67.000000	67.000000	67.000000	67.000000
unique	67	NaN	NaN	NaN	NaN
top	Brevard	NaN	NaN	NaN	NaN
freq	1	NaN	NaN	NaN	NaN
mean	NaN	43450.970149	43453.985075	1454.119403	260.880597
std	NaN	57182.620266	75070.435056	2033.620972	450.498092
min	NaN	1317.000000	789.000000	19.000000	9.000000
25%	NaN	4757.000000	3058.000000	95.500000	46.500000
50%	NaN	20206.000000	14167.000000	562.000000	120.000000
75%	NaN	56546.500000	46015.000000	1870.500000	285.500000
max	NaN	289533.000000	387703.000000	10022.000000	3411.000000

[215]:
```python
votes.head(10)
```

[215]:

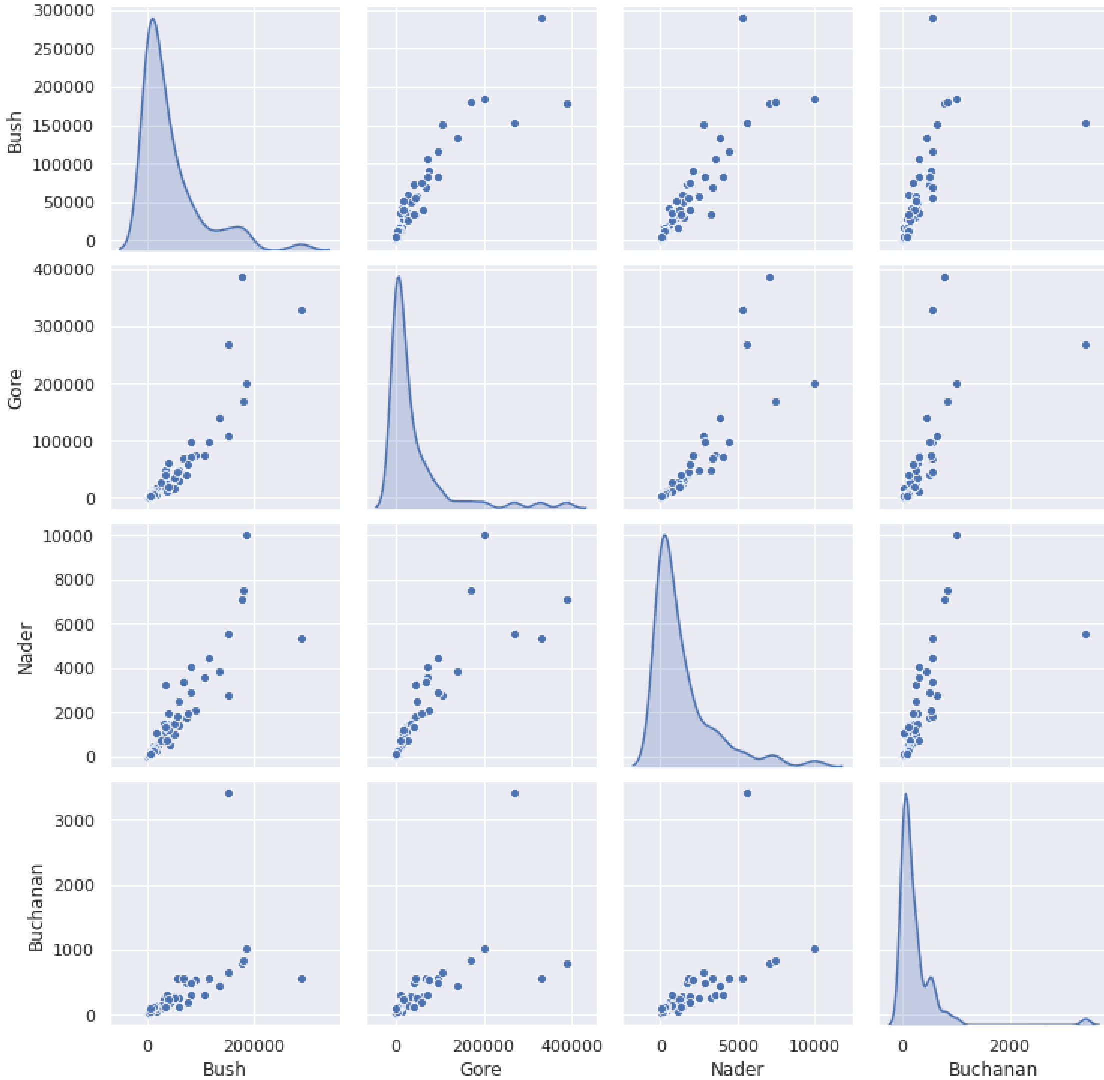
	county	Bush	Gore	Nader	Buchanan
0	Alachua	34124	47365	3226	263
1	Baker	5610	2392	53	73
2	Bay	38637	18850	828	248
3	Bradford	5414	3075	84	65
4	Brevard	115185	97318	4470	570
5	Broward	177902	387703	7104	795
6	Calhoun	2873	2155	39	90
7	Charlotte	35426	29645	1462	182
8	Citrus	29767	25525	1379	270
9	Clay	41736	14632	562	186

2.0.1 2a. Plot a pair plot of the data using the seaborn library. [Peer Review]

Upload a screenshot or saved copy of your plot for this week's Peer Review assignment. **Note:** your code for this section may cause the Validate button to time out. If you want to run the Validate button prior to submitting, you could comment out the code in this section after completing the Peer Review.

[216]:
```python
# plot a pair plot of the data using the seaborn library
# possible way to save the image
# plt.savefig('pair_plot.png', dpi = 300, bbox_inches = 'tight')
#They are correlated but can't necessarily say collinear. The increase in votes␣
↪can depend on the amount of votes
# in each county
# your code here
sns.pairplot(votes, diag_kind='kde')
```

[216]: <seaborn.axisgrid.PairGrid at 0x7fb72db0fbd0>



2.0.2 2b. Comment on the relationship between the quantiative datasets. Are they correlated? Collinear? [Peer Review]

You will answer this question in this week's Peer Review assignment.

The data is definitely correlated because for all comparisons, when one variable increases, so does another. This doesn't necessarily mean they are collinear though. For example, the upward trend could be due to increases in population size. In other words, the votes of one candidate aren't directly affecting the other.

2.0.3 2c. Multi-linear [5 pts, Peer Review]

Construct a multi-linear model called `model` without interaction terms predicting the Bush column on the other columns and print out the summary table. You should name your model's object as `model` in order to pass the autograder. Use the full data (not train-test split for now) and do not scale features.

[217]:
```
# uncomment and construct a multi-linear model
# your code here
formula = smf.ols('Bush ~ Gore + Nader + Buchanan', data=votes)
model = formula.fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Bush   R-squared:                       0.877
Model:                            OLS   Adj. R-squared:                  0.871
Method:                 Least Squares   F-statistic:                     149.5
Date:                Tue, 26 Apr 2022   Prob (F-statistic):           1.35e-28
Time:                        19:27:49   Log-Likelihood:                -758.33
No. Observations:                  67   AIC:                             1525.
Df Residuals:                      63   BIC:                             1533.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   8647.6837   3133.545      2.760      0.008    2385.793    1.49e+04
Gore           0.4475      0.071      6.305      0.000       0.306       0.589
Nader         11.8533      2.503      4.735      0.000       6.851      16.855
Buchanan      -7.2033      7.864     -0.916      0.363     -22.917       8.511
==============================================================================
Omnibus:                       20.698   Durbin-Watson:                   1.969
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              128.017
Skew:                           0.383   Prob(JB):                     1.59e-28
Kurtosis:                       9.728   Cond. No.                     1.08e+05
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.08e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[218]:
```
# tests model
```

Is there any insignificant feature(s)? Explain your answer in this week's Peer Review assignment.

Yes, Buchanan is insignificant.

2.0.4 2d. Multi-linear with interactions [Peer Review]

Construct a multi-linear model with interactions that are statistically significant at the $p = 0.05$ level. You can start with full interactions and then eliminate interactions are do not meet the $p = 0.05$ threshold. Name this model object as `model_multi`. You will share you solution in this week's Peer Review assignment.

[219]:
```python
# uncomment and construct multi-linear model
# model_multi =
# your code here
model_multi= smf.ols('Bush ~ Gore + Nader + Buchanan + Gore:Nader + Gore:
↪Buchanan + Nader:Buchanan', data=votes).fit()
print(model_multi.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Bush   R-squared:                       0.948
Model:                            OLS   Adj. R-squared:                  0.943
Method:                 Least Squares   F-statistic:                     183.5
Date:                Tue, 26 Apr 2022   Prob (F-statistic):           1.12e-36
Time:                        19:27:49   Log-Likelihood:                -729.23
No. Observations:                  67   AIC:                             1472.
Df Residuals:                      60   BIC:                             1488.
Df Model:                           6
Covariance Type:            nonrobust
================================================================================
==
                     coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
--
Intercept         52.8673   2781.618      0.019      0.985   -5511.197
5616.931
Gore               1.7506      0.207      8.456      0.000       1.336
2.165
Nader            -11.5313      5.069     -2.275      0.026     -21.670
-1.393
Buchanan          10.0391     26.104      0.385      0.702     -42.176
62.254
Gore:Nader        -0.0001   2.46e-05     -5.030      0.000      -0.000
-7.46e-05
Gore:Buchanan     -0.0009      0.000     -5.765      0.000      -0.001
-0.001
Nader:Buchanan     0.0357      0.008      4.600      0.000       0.020
0.051
==============================================================================
Omnibus:                        6.375   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.041   Jarque-Bera (JB):               10.406
```

```
Skew:                          -0.088   Prob(JB):                      0.00550
Kurtosis:                       4.923   Cond. No.                     9.16e+08
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 9.16e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[220]:
```
# tests model_multi
# your code here
```

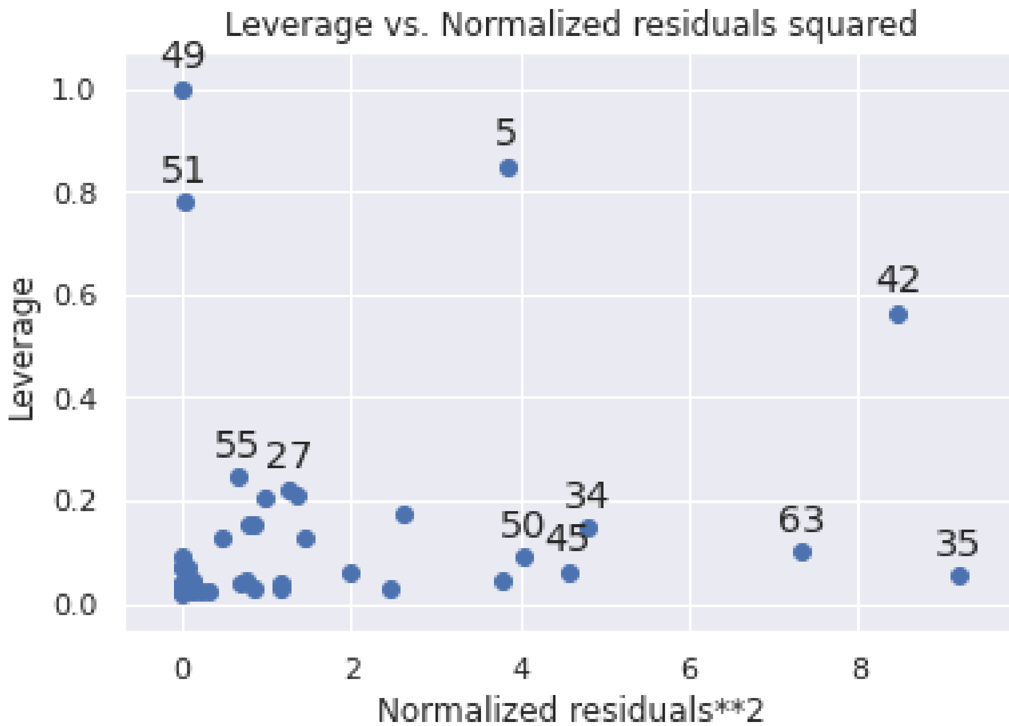
2.0.5 2e. Leverage [Peer Review]

Plot the *leverage* vs. the square of the residual.

These resources might be helpful
- https://rpubs.com/Amrabdelhamed611/669768 - https://www.statsmodels.org/dev/generated/statsmodels.grap

[221]:
```
# plot the leverage vs. the square of the residual
# your code here
plt.figure(figsize=(15,15))
sm.graphics.plot_leverage_resid2(model_multi)
plt.show()
```

```
<Figure size 1080x1080 with 0 Axes>
```



[222]:
```
# you can use this cell to try different plots
# your code here
```

Upload your plot for this week's Peer Review assignment. If you tried out multiple models, upload a single model.

2.0.6 2f. Identify and Clean [5pts]

The leverage *vs* residual plot indicates that some rows have high leverage but small residuals and others have high residual. The R^2 of the model is determined by the residual. The data is from the disputed 2000 election where one county caused significant issues.

Display the *3 or more* rows for the points indicated having high leverage and/or high residual squared. You will use this to improve the model R^2.

Name the list of indices for those high-leverage and/or high-residual points as `unusual`.

[223]:
```
# uncomment and fill unusual with list of indices for high-leverage and/or
↪high-residual points
# unusual = []
# your code here
unusual = [49, 35, 51, 5, 63, 42]
```

[224]:
```
# tests your list of indices for high-leverage and/or high-residual points
```

2.0.7 2g. Final model [5 pts]

Develop your final model by dropping *one or more* of the troublesome data points indicated in the leverage *vs* residual plot and insuring any interactions in your model are still significant at $p = 0.05$. Your model should have an R^2 great than 0.95. Call your model `model_final`.

[225]:
```python
# develop your model_final here
# model_final =
# your code here
votes = pd.read_csv('data/fl2000.txt', delim_whitespace=True, comment='#')
votes.drop([49, 35], inplace=True)
model_final= smf.ols('Bush ~ Gore + Gore:Nader + Nader', data=votes).fit()
print(model_final.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Bush   R-squared:                       0.925
Model:                            OLS   Adj. R-squared:                  0.921
Method:                 Least Squares   F-statistic:                     250.9
Date:                Tue, 26 Apr 2022   Prob (F-statistic):           2.93e-34
Time:                        19:27:49   Log-Likelihood:                -718.65
No. Observations:                  65   AIC:                             1445.
Df Residuals:                      61   BIC:                             1454.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   -542.6358   2776.974     -0.195      0.846   -6095.539    5010.267
Gore           0.9407      0.093     10.147      0.000       0.755       1.126
Gore:Nader  -8.46e-05   1.34e-05     -6.292      0.000      -0.000   -5.77e-05
Nader         14.2913      2.006      7.123      0.000      10.280      18.303
==============================================================================
Omnibus:                       12.618   Durbin-Watson:                   1.961
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               42.120
Skew:                          -0.169   Prob(JB):                     7.14e-10
Kurtosis:                       6.929   Cond. No.                     7.30e+08
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 7.3e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[226]:
```python
# tests model_final
```

2.1 3. Body Mass Index Model [25 points, Peer Review]

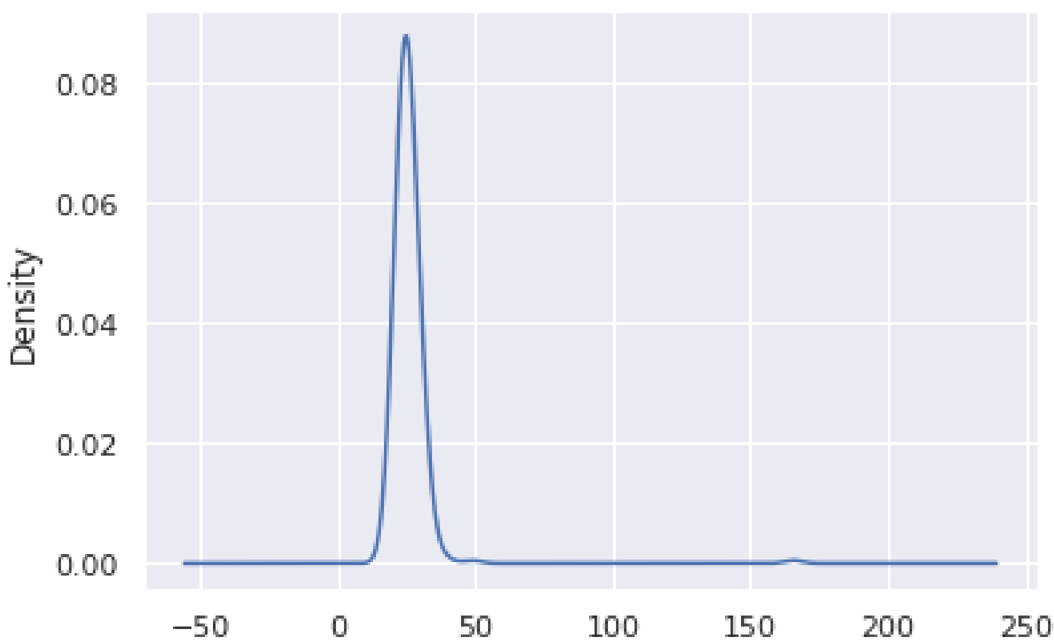
In this problem, you will first clean a data set and create a model to estimate body fat based on the common BMI measure. Then, you will use the **forward stepwise selection** method to create more accurate predictors for body fat.

The body density dataset in file `bodyfat` includes the following 15 variables listed from left to right: * Density : Density determined from underwater weighing * Fat : Percent body fat from Siri's (1956) equation * Age : Age (years) * Weight : Weight (kg) * Height : Height (cm) * Neck : Neck circumference (cm) * Chest: Chest circumference (cm) * Abdomen : Abdomen circumference (cm) * Hip : Hip circumference (cm) * Thigh : Thigh circumference (cm) * Knee : Knee circumference (cm) * Ankle : Ankle circumference (cm) * Biceps : Biceps (extended) circumference (cm) * Forearm : Forearm circumference (cm) * Wrist : Wrist circumference (cm)

The `Density` column is the "gold standard" – it is a measure of body density obtained by dunking people in water and measuring the displacement. The `Fat` column is a prediction using another statistical model. The body mass index (BMI) is calculated as Kg/m^2 and is used to classify people into different weight categories with a BMI over 30 being 'obese'. You will find that BMI is a poor predictor of the `Density` information it purports to predict. You will try to find better models using measurements and regression.

Unfortunately for us, the dataset we have has imperial units for weight and height, so we will convert those to metric and then calculate the BMI and plot the KDE of the data.

[227]:
```
fat = pd.read_csv('data/bodyfat.csv')
fat = fat.drop('Unnamed: 0', axis=1)
fat.Weight = fat.Weight * 0.453592 # Convert to Kg
fat.Height = fat.Height * 0.0254 # convert inches to m
fat['BMI'] = fat.Weight / (fat.Height**2)
fat.BMI.plot.kde();
```



2.1.1 3a. [5 pts]

The BMI has at least one outlier since it's unlikely anyone has a BMI of 165, even Arnold Schwarzenegger.

Form a new table **cfat** (cleaned fat) that removes any rows with a BMI greater than 40 and calculate the regression model predicting the **Density** from the **BMI**. Display the summary of the regression model. Call your model as **bmi**. You should achieve an R^2 of at least 0.53.

[228]:
```
# form new table cfat and model bmi
# cfat =
# bmi =
# your code here
cfat= fat.drop(fat[fat['BMI'] > 40].index)
bmi = smf.ols('Density ~ BMI', data=cfat).fit()
print(bmi.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Density   R-squared:                       0.536
Model:                            OLS   Adj. R-squared:                  0.534
Method:                 Least Squares   F-statistic:                     286.2
Date:                Tue, 26 Apr 2022   Prob (F-statistic):           3.25e-43
Time:                        19:27:50   Log-Likelihood:                 734.17
No. Observations:                 250   AIC:                            -1464.
Df Residuals:                     248   BIC:                            -1457.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      1.1602      0.006    186.410      0.000       1.148       1.172
BMI           -0.0041      0.000    -16.918      0.000      -0.005      -0.004
==============================================================================
Omnibus:                        2.262   Durbin-Watson:                   1.576
Prob(Omnibus):                  0.323   Jarque-Bera (JB):                2.259
Skew:                           0.229   Prob(JB):                        0.323
Kurtosis:                       2.916   Cond. No.                         195.
==============================================================================

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

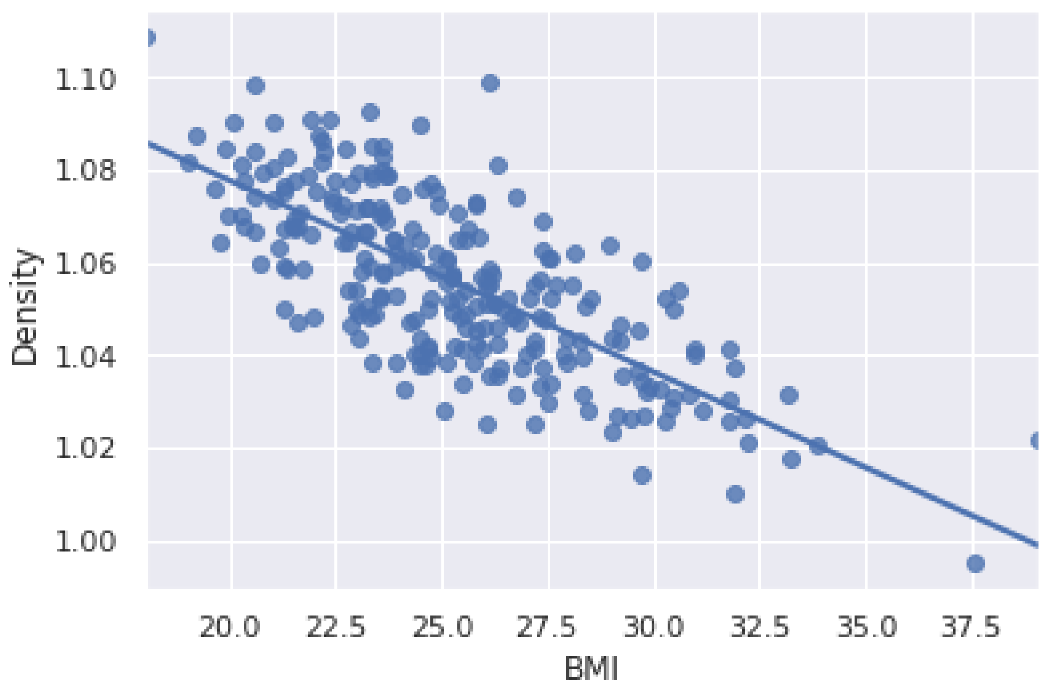
[229]:
```
# tests your bmi model
```

2.1.2 3b. [Peer Review]

Plot your regression model against the BMI measurement, properly labeling the scatterplot axes and showing the regression line. In subsequent models, you will not be able to plot the Density *vs* your predictors because you will have too many predictors, but it's useful to visually understand the relationship between the BMI predictor and the `Density` because you should find that the regression line goes through the data but there is too much variability in the data to achieve a good R^2. Upload a copy or screensho of your plot for this week's Peer Review assignment.

[230]:
```
# plot regression model against BMI measurement
# properly label the scatterplot axs and show the regression line
# your code here
sns.regplot(cfat['BMI'], cfat['Density'], ci=None)
```

[230]: `<matplotlib.axes._subplots.AxesSubplot at 0x7fb72d5e0090>`



The `BMI` model uses easy-to-measure predictors, but has a poor $R^2 \sim 0.54$. We will use structured subset selection methods from ISLR Chapter 6.1 to derive two better predictors. That chapter covers *best subset*, *forward stepwise* and *backware stepwise* selection. I have implemented the *best subset* selection which searches across all combinations of $1, 2, \ldots, p$ predictors and selects the best predictor based on the *adjusted* R^2 metric. This method involved analyzing $2^{13} = 8192$ regression models (programming and computers for the win). The resulting *adjusted* R^2 plot is shown below (Since the data split can be different, your result may look slightly different):

In this plot, `test_fat` and `train_fat` datasets each containing 200 randomly selected samples were derived from the `cfat` dataset using `np.random.choice` over the `cfat.index` and selected using the

Pandas `loc` method. Then, following the algorithm of ISLR Algorithm 6.1 *Best Subset Selection*, all $\binom{p}{k}$ models with k predictors were evaluated on the training data and the model returning the best *Adjusted* R^2 was selected. These models are indicated by the data points for the solid blue line. As the text indicates, other measures (AIC, BIC, C_p) would be better than the *Adjusted* R^2, but we use it becuase because you've already seen the R^2 and should have an understanding of what it means.

Then, the best models for each k were evaluated for the `test_fat` data. These results are shown as the red dots below the blue line. Note that because the test and train datasets are randomly selected subsets, the results vary from run-to-run and it may that your test data produces better R^2 than your training data.

In the following exercises, you can not use the `Density`, `Fat` or `BMI` columns in your predictive models. You can only use the 13 predictors in the `allowed_factors` list.

[231]:
```
allowed_factors = ['Age', 'Weight', 'Height', 'Neck', 'Chest',
       'Abdomen', 'Hip', 'Thigh', 'Knee', 'Ankle', 'Biceps', 'Forearm',
       'Wrist']
```

2.2 Forward Stepwise Refinement

You will manually perform the steps of the *forward stepwise selection* method for four parameters. You will do this following Algorithm 6.2 from ISLR. For $k = 1 \ldots 4$: * Set up a regression model with k factors that involves the fixed predictors from the previous step $k-1$ * Try all p predictors in the new kth position * Select the best parameter using $Adjusted - R^2$ (e.g. `model.rsquared_adj`) given your training data * Fix the new parameter and continue the process for $k+1$

Then, you will construct a plot similar to the one above, plotting the $Adjusted - R^2$ for each of your k steps and plotting the $Adjusted - R^2$ from the test set using that model.

2.2.1 3c. [5 pts]

First, construct your training and test sets from your `cfat` dataset. Call the resulting data frame to train_fat and test_fat. train_fat includes randomly selected 125 observations and the test_fat has the rest.

[234]:
```
# construct train_fat and test_fat from cfat dataset
# your code here
train_fat = cfat.sample(n=125, random_state=42)
test_fat = cfat.drop(train_fat.index)
print(cfat.shape)
print(train_fat.shape)
print(test_fat.shape)
```

```
(250, 16)
(125, 16)
(125, 16)
```

[235]:
```python
# tests your training and test sets
```

2.2.2 3d. Conduct the algorithm above for $k = 1$, leaving your best solution as the answer [5 pts]

Call your resulting model `train_bmi1`.

[236]:
```python
best = ['',0]
for p in allowed_factors:
    model  = smf.ols(formula='Density~'+p, data=train_fat).fit()
    print(p, model.rsquared)
    if model.rsquared>best[1]:
        best = [p, model.rsquared]
print('best:',best)
```

```
Age 0.10821786297991254
Weight 0.27676187565932253
Height 0.03297134879261787
Neck 0.16321294736761738
Chest 0.4139716300779712
Abdomen 0.5993382876433768
Hip 0.3043175668762542
Thigh 0.20974960797786435
Knee 0.12547128315071832
Ankle 0.037278369223836316
Biceps 0.1243832598781125
Forearm 0.05017377141852741
Wrist 0.04795651828318337
best: ['Abdomen', 0.5993382876433768]
```

[237]:
```python
allowed_factors.remove(best[0])
```

[238]:
```python
# train_bmi1 =
# your code here
train_bmi1 = smf.ols('Density ~ Abdomen', data=train_fat).fit()
test_bmi1 = smf.ols('Density ~ Abdomen', data=test_fat).fit()
```

[239]:
```python
# tests train_bmi1 model
```

2.2.3 3e. Conduct the algorithm above for $k = 2$, leaving your best solution as the answer [5 pts]

Name your model object as `train_bmi2`.

[240]:
```python
# your code here


best = ['',0]
for p in allowed_factors:
    model  = smf.ols(formula='Density~ Abdomen + '+p, data=train_fat).fit()
    print(p, model.rsquared)
    if model.rsquared>best[1]:
        best = [p, model.rsquared]


print('best:',best)
allowed_factors.remove(best[0])
```

```
Age 0.6050613697349637
Weight 0.6585145465631732
Height 0.6806565383190818
Neck 0.6600993742892025
Chest 0.6165845448962466
Hip 0.6195220648633208
Thigh 0.6115062073892816
Knee 0.6418629291017631
Ankle 0.61574159736894
Biceps 0.612232522676394
Forearm 0.6092640963725275
Wrist 0.6728839762218873
best: ['Height', 0.6806565383190818]
```

[241]:
```python
train_bmi2 = smf.ols('Density ~ Abdomen + Height', data=train_fat).fit()
test_bmi2 = smf.ols('Density ~ Abdomen + Height', data=test_fat).fit()
```

[242]:
```python
# tests train_bmi2 model
```

2.2.4 3f. Conduct the algorithm above for $k = 3$, leaving your best solution as the answer [Peer Review]

[243]:
```python
# your code here
best = ['',0]
for p in allowed_factors:
    model  = smf.ols(formula='Density~ Abdomen + Height + '+p, data=train_fat).
 ↪fit()
    print(p, model.rsquared)
    if model.rsquared>best[1]:
        best = [p, model.rsquared]
print('best:',best)
allowed_factors.remove(best[0])
```

```
Age 0.6806577769990833
Weight 0.6830616493665629
Neck 0.7031514939144772
Chest 0.6889355137847133
Hip 0.6807428830607603
Thigh 0.6808774141047103
Knee 0.6837176559348723
Ankle 0.6807875155606264
Biceps 0.6820532640549128
Forearm 0.68085821148755
Wrist 0.7048115065966181
best: ['Wrist', 0.7048115065966181]
```

[244]:
```
train_bmi3 = smf.ols('Density ~ Abdomen + Height + Wrist', data=train_fat).fit()
test_bmi3 = smf.ols('Density ~ Abdomen + Height + Wrist', data=test_fat).fit()
```

[245]:
```
# tests train_bmi3 model
# your code here
```

2.2.5 3g. Conduct the algorithm above for $k = 4$, leaving your best solution as the answer [Peer Review]

[246]:
```
# your code here
best = ['',0]
for p in allowed_factors:
    model  = smf.ols(formula='Density~ Abdomen + Height + Wrist + '+p,␣
 ↪data=train_fat).fit()
    print(p, model.rsquared)
    if model.rsquared>best[1]:
        best = [p, model.rsquared]
print('best:',best)

allowed_factors.remove(best[0])
```

```
Age 0.7086773230340788
Weight 0.70482030082629
Neck 0.7121112305670161
Chest 0.7085855447953868
Hip 0.7048894487176653
Thigh 0.7049139682840758
Knee 0.7050127314132238
Ankle 0.7052041106229112
Biceps 0.704836957294236
Forearm 0.7053074686977421
best: ['Neck', 0.7121112305670161]
```

[247]:
```
train_bmi4 = smf.ols('Density ~ Abdomen + Height + Wrist + Neck',␣
↪data=train_fat).fit()
test_bmi4 = smf.ols('Density ~ Abdomen + Height + Wrist + Neck', data=test_fat).
↪fit()
```

2.2.6 3h. Conduct the algorithm above for $k = 5$, leaving your best solution as the answer [Peer Review]

[249]:
```
# your code here
best = ['',0]
for p in allowed_factors:
    model  = smf.ols(formula='Density~ Abdomen + Height + Wrist + Neck + '+p,␣
↪data=train_fat).fit()
    print(p, model.rsquared)
    if model.rsquared>best[1]:
        best = [p, model.rsquared]

print('best:',best)
allowed_factors.remove(best[0])
```

```
Age 0.714669335982867
Weight 0.7136355088415423
Chest 0.7133308363596438
Hip 0.7121287601121737
Thigh 0.71214152445003
Knee 0.7121364226215041
Ankle 0.7128806150763229
Biceps 0.7134595156426884
Forearm 0.7156073697857823
best: ['Forearm', 0.7156073697857823]
```

[250]:
```
train_bmi5 = smf.ols('Density~ Abdomen + Height + Wrist + Neck + Forearm',␣
↪data=train_fat).fit()
test_bmi5 = smf.ols('Density ~ Abdomen + Height + Wrist + Neck + Forearm',␣
↪data=test_fat).fit()
```

2.2.7 3i. Plot [5 pts]

Plot your resulting *adjusted* R^2 *vs* number of predictors (k=1,2,3,4,5) and overlay the *adjusted* R^2 for the test data. Call the list of the five adjusted r-squared values from the five train_bmi# models as `adjr2_train` and the one from the test data as `adjr2_test`.

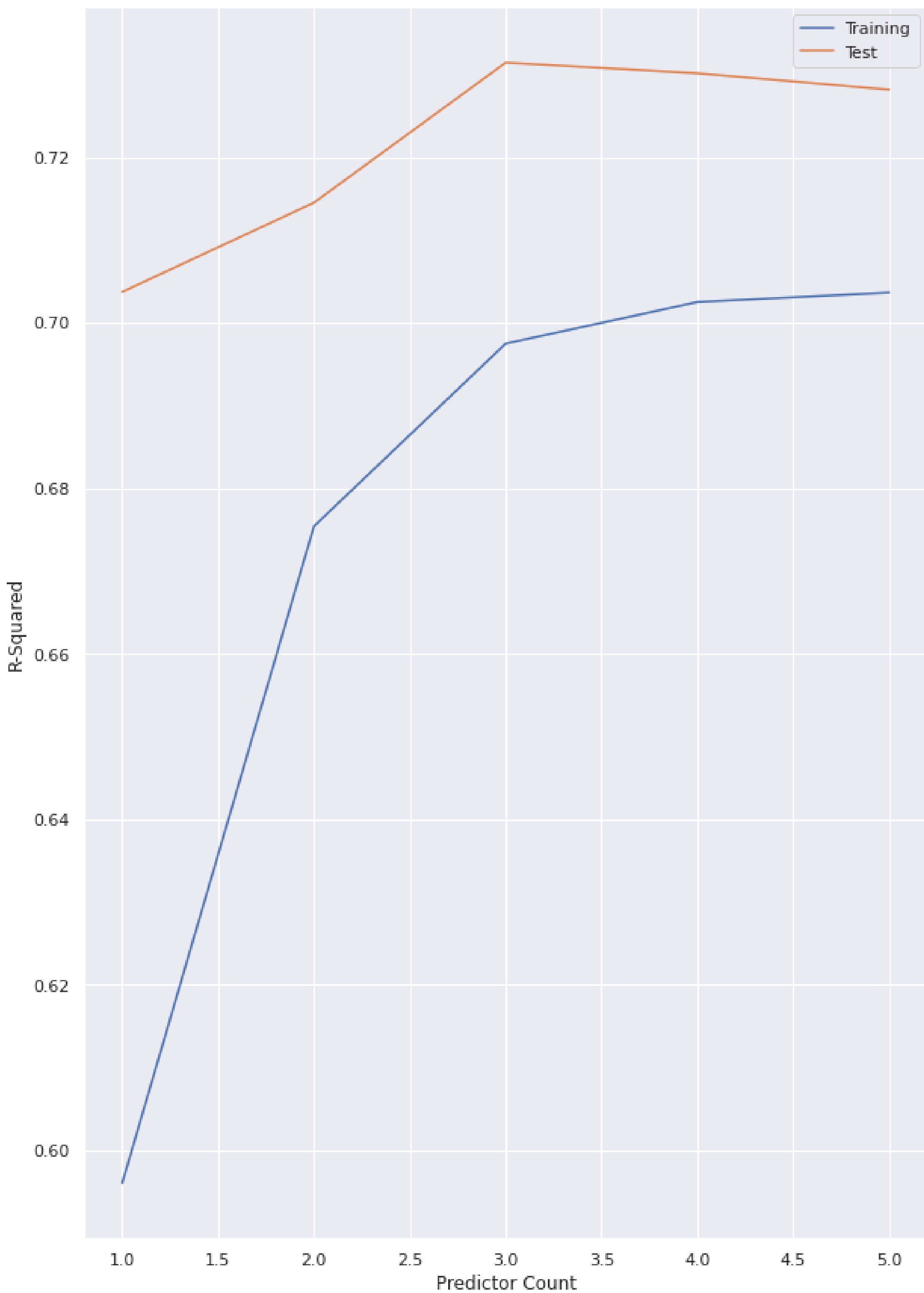
[251]:
```
# plot resulting adjusted rsquared vs number of predictors (k=1,2,3,4,5)
# overlay the adjusted rsquared for the test data
# your code here
```

```
#For adjusted on test data just run a model of the variable with the test data␣
↪set
adjr2_train = [train_bmi1.rsquared_adj, train_bmi2.rsquared_adj, train_bmi3.
↪rsquared_adj, train_bmi4.rsquared_adj,
               train_bmi5.rsquared_adj]
adjr2_test = [test_bmi1.rsquared_adj, test_bmi2.rsquared_adj, test_bmi3.
↪rsquared_adj, test_bmi4.rsquared_adj,
               test_bmi5.rsquared_adj]

print(adjr2_train)
print(adjr2_test)
```

```
[0.5960808753477945, 0.6754213996030012, 0.6974927836196747, 0.7025149382525833,
0.7036580996087143]
[0.703748169887118, 0.7145292802456226, 0.7314500749608879, 0.7301668752316659,
0.7281949821073446]
```

[252]:
```
data_rsquared = pd.DataFrame({
    'Predictor Count' : [1,2,3,4,5],
    'Training' : adjr2_train,
    'Test' : adjr2_test
})

fig, ax = plt.subplots(figsize=(10,15))
ax= sns.lineplot(x='Predictor Count', y='Training', data=data_rsquared,␣
↪label='Training')
ax1 = sns.lineplot(x='Predictor Count', y='Test', data=data_rsquared,␣
↪label='Test')
plt.ylabel('R-Squared')
```

[252]:
```
Text(0, 0.5, 'R-Squared')
```



[197]:
```
# tests adjusted r-squared plot vs. number of factors
```

2.2.8 3j. Discussion [Peer Review]

The BMI model has the benefit being simple (two measurements, height and wright). Looking at your resulting regression model, how many parameters would you suggest to use for your enhanced BMI model? Justify your answer using your models. Submit your answer with this week's Peer Review assignment.

For both the training and test sets, the adjusted R^2 doesn't seem to change much after 3 predictors. Thus, the appropriate amount of parameters is 3.