Solutions 04 to Worked Problems

David Gerbing
The School of Business
Portland State University
gerbing@pdx.edu

Purpose: For the customers of an online motorcycle clothing company, build a predictive model of Weight from a variety of body measurements plus Gender.

The data are the body measurements of customers for a motorcycle online clothing retailer.

Data: http://web.pdx.edu/~gerbing/data/BodyMeas2500.xlsx

```
from datetime import datetime as dt
now = dt.now()
print ("Analysis on", now.strftime("%Y-%m-%d"), "at", now.strftime("%H:%M"))

    Analysis on 2021-07-13 at 03:24
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Data

a. Read the data.

```
d = pd.read_excel("http://web.pdx.edu/~gerbing/data/BodyMeas2500.xlsx")
#d = pd.read_excel("~/Documents/BookNew/data/GHC/BodyMeas2500.xlsx")
```

b. How many samples (rows of data) and columns are there in the data file?

```
d.shape

    (2500, 7)
```

2501 rows of data with 7 columns

c. Display the first 6 rows of data and the variable names.

```
d.head()
```

	Gender	Weight	Height	Waist	Hips	Chest	ArmLength
0	M	135	70	34	38	36	34
1	M	235	66	36	45	48	32
2	M	205	72	38	44	44	36
3	M	190	70	36	41	40	32
4	F	200	64	39	47	45	32

d. What are the variables in the data table? (From code or copy and paste.)

```
d.columns

    Index(['Gender', 'Weight', 'Height', 'Waist', 'Hips', 'Chest', 'ArmLength'], dtype='object')
```

e. Check for missing data. Any action (deletion, imputation) needed?

```
print (d.isna().sum())
print ('\nTotal Missing:', d.isna().sum().sum())

    Gender       0
    Weight       0
    Height       0
    Waist        0
    Hips         0
    Chest        0
    ArmLength    0
    dtype: int64

    Total Missing: 0
```

No missing data. No action needed.

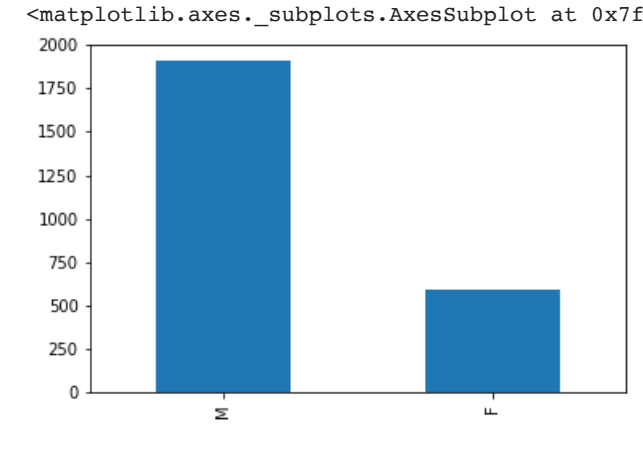
f. Generate a frequency distribution table of Gender.

```
freq = d['Gender'].value_counts()
freq

    M    1908
    F     592
    Name: Gender, dtype: int64
```

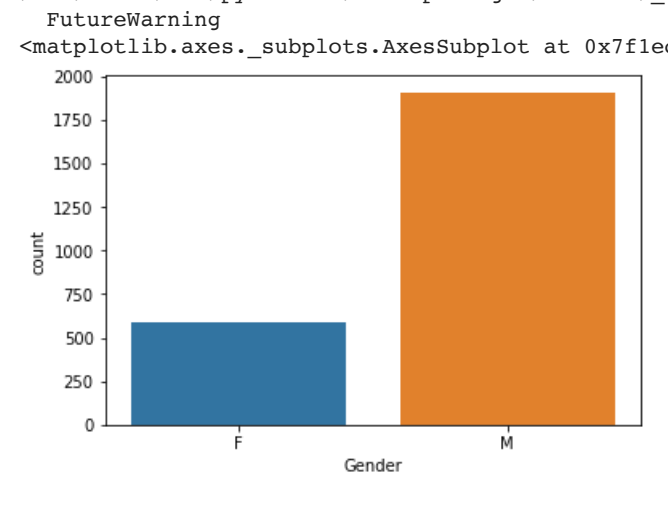
Quite a few more men than women customers.

g. Generate a bar chart of Gender. (Just need one, here show both Matplotlib and Seaborn versions.)

```
freq.plot(kind='bar')

    <matplotlib.axes._subplots.AxesSubplot at 0x7f1ed2b2bd10>
```



```
sns.countplot(d['Gender'], order=['F', 'M'])

    /usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a key
      FutureWarning
    <matplotlib.axes._subplots.AxesSubplot at 0x7f1ed2d90110>
```



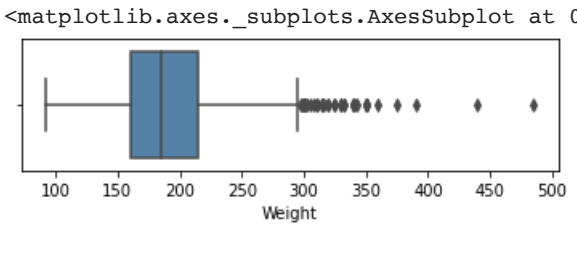
h. Include Gender in the model to be developed. Convert the two levels of Gender to two dummy variables and use one of them.

```
d = pd.get_dummies(d, columns=['Gender'], drop_first=True)
d.head()
```

	Weight	Height	Waist	Hips	Chest	ArmLength	Gender_M
0	135	70	34	38	36	34	1
1	235	66	36	45	48	32	1
2	205	72	38	44	44	36	1
3	190	70	36	41	40	32	1
4	200	64	39	47	45	32	0

i. Generate a boxplot of Weight.

```
plt.figure(figsize=(6,1.5))
sns.boxplot(x=d['Weight'], color='steelblue')

    <matplotlib.axes._subplots.AxesSubplot at 0x7f1ed2ba3f10>
```



j. There are two strong outliers in the data. Why remove them?

The two most extreme outliers are over 400 lbs, so delete these and then in the presentation of results, generalize only to those under 400 lbs.

k. Subset the data frame to remove those rows of data. As always, audit any intended changes in the data frame. Here display the number of rows in the filtered data frame to demonstrate the deletion.

```
print(d.shape)
d = d[d['Weight'] < 400]
print(d.shape)

    (2500, 7)
    (2498, 7)
```

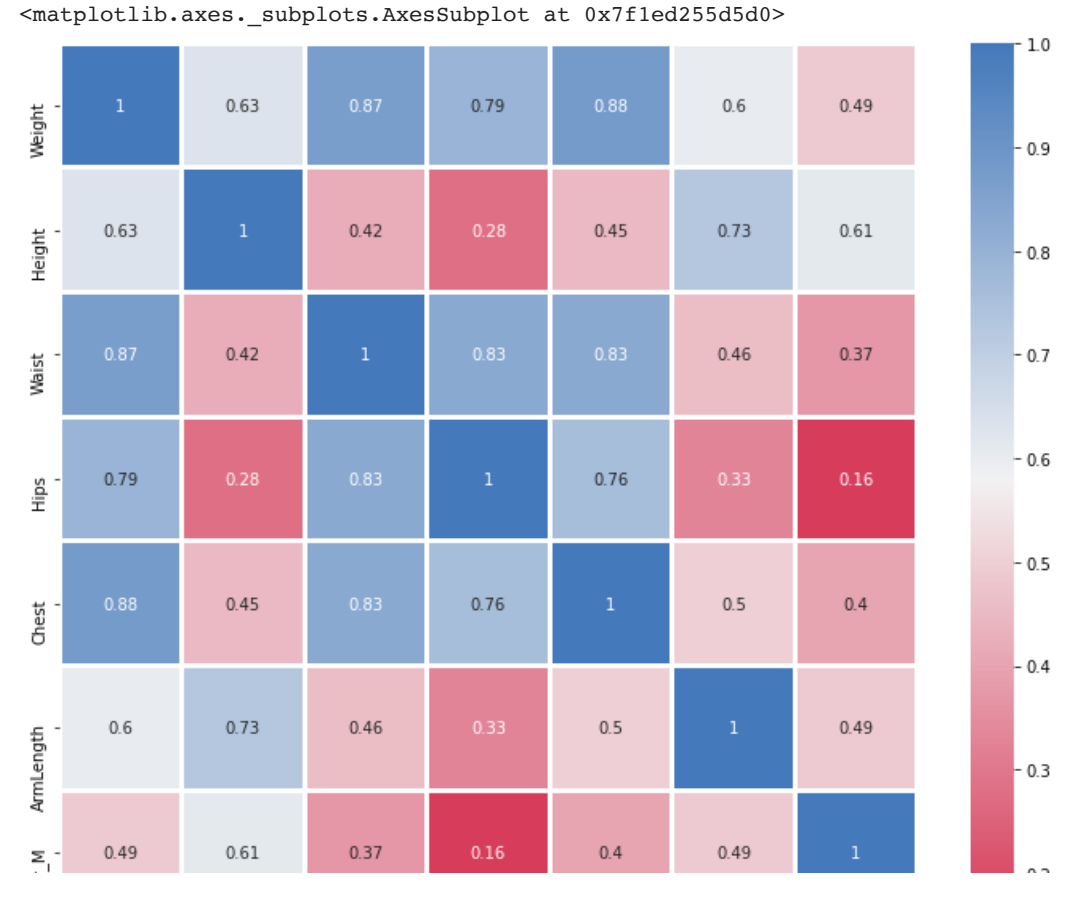
l. Examine the relevance of each feature according to its correlation with the target.

```
d.corr()['Weight'].sort_values().round(2)

    Gender_M     0.49
    ArmLength    0.60
    Height       0.63
    Hips         0.79
    Waist        0.87
    Chest        0.88
    Weight       1.00
    Name: Weight, dtype: float64
```

All features are relevant, that is, correlate highly with the target.

m. Generate a heat map correlation matrix of the model variables. Comment on collinearity.

```
plt.figure(figsize=(12,10))
sns.heatmap(d.corr().round(2), linewidths=2.0, annot=True,
            cmap=sns.diverging_palette(5, 250, as_cmap=True))

    <matplotlib.axes._subplots.AxesSubplot at 0x7f1ed255d5d0>
```



Given the small data set with no pressure on computational time, to take a strong stance against *data leakage*, no features are deleted at this point in the analysis, though there is clearly much collinearity. In all likelihood, not all of the features will be needed. The worst offenders are Waist with Hips and Waist with Chest, both correlations of 0.83. Clearly not all three body measurements are needed to forecast Weight.

Regression Analysis

a. Store the features, the predictor variables, in data structure X. Store the target variable in data structure y.

Not necessary, but often useful to store the predictor variables in their own list, here *pred_vars*. Then invoke this list when needed, such as defining X.

```
y = d['Weight']
pred_vars = ['Height', 'Waist', 'Hips', 'Chest', 'ArmLength', 'Gender_M']
X = d[pred_vars]
```

b. Use code to display the number of features.

The function `len()` provides the length of a vector, that is, the number of elements of a vector.

```
n_pred = len(pred_vars)
print("Number of predictor variables:", n_pred)

    Number of predictor variables: 6
```

c. Split the data into 75% training data and 25% testing data.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25, random_state=7)
```

b. Do the multiple regression with all possible features. Display the estimated model coefficients.

Access `sklearn` linear regression.

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

Train the machine (i.e., estimate the model).

```
model.fit(X_train, y_train)

    LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Display the estimated model (not needed to evaluate forecasting efficiency).

```
print("intercept: %.3f" % (model.intercept_), "\n")
cdf = pd.DataFrame(model.coef_, X.columns, columns=['Coefficients'])
cdf.transpose().style.format("{:.3}")

    intercept: -329.880
```

	Height	Waist	Hips	Chest	ArmLength	Gender_M
Coefficients	2.84	1.91	2.17	3.32	0.251	6.13

c. For the person who provided the first row of data, manually calculate his fitted weight from the model explicitly from the model coefficients. (For pedagogy, to show understanding of the model, not normally done here.)

```
d.iloc[0:1, :]
```

	Weight	Height	Waist	Hips	Chest	ArmLength	Gender_M
0	135	70	34	38	36	34	1

```
predWt = -329.880 + 2.84*70 + 1.91*34 + 2.17*38 + 3.32*36 + 0.251*34 + 6.13*1
round(predWt,3)

    150.504
```

d. What is the residual for the first person? Comment.

```
res = d.loc[0, "Weight"] - predWt
round(res,3)

    -15.504
```

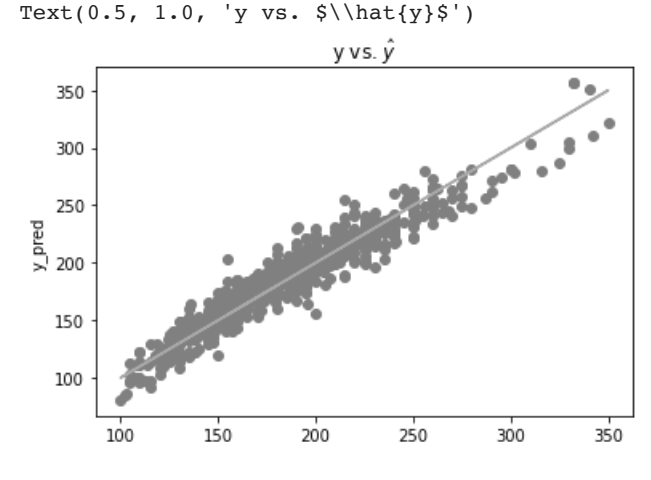
The person's fitted weight is about 15 1/2 lbs too large compared to his actual weight.

e. Calculate the forecasted values of y, $\hat{y}$, from the testing data for X.

```
y_fit = model.predict(X_train)
y_pred = model.predict(X_test)
```

f. Visually compare the forecasted values of y from the model applied to the testing data to the obtained values of y in the testing data. Comment

```
plt.scatter(y_test, y_pred, color='gray')
#plt.xlim(0,52)
plt.plot(y_test, y_test, color="darkgray")
plt.xlabel("y")
plt.ylabel("y_pred")
plt.title("y vs. $\hat{y}$")

    Text(0.5, 1.0, 'y vs. $\\hat{y}$')
```



Fit appears quite good. There is little scatter about the diagonal line, which represents values where $y = \hat{y}$, so should be a small s_e. There is much variation in Weight, so should be a high R^2. A linear model is clearly appropriate.

g. Evaluate model fit to the training data with the standard deviation of residuals and R-squared. Comment

```
from sklearn.metrics import mean_squared_error, r2_score
mse = mean_squared_error(y_train, y_fit)
rsq = r2_score(y_train, y_fit)
print("MSE: %.3f" % mse)
se = np.sqrt(mse)
range95 = 4 * se
print("Stdev of residuals: %.3f " % se)
print("Approximate 95 per cent range of residuals: %.3f " % range95)
print("R-squared: %.3f" % rsq)

    MSE: 177.481
    Stdev of residuals: 13.322
    Approximate 95 per cent range of residuals: 53.289
    R-squared: 0.906
```

The model fits the data on which it was trained. Very high R^2, though the standard deviation of the residuals is somewhat large, resulting in an approximate 95% range of residuals of about 53 1/4 lbs for the training data (higher in the testing data).

h. Evaluate model fit to the testing data with the standard deviation of residuals and R-squared. Comment.

```
mse_f = mean_squared_error(y_test, y_pred)
rsq_f = r2_score(y_test, y_pred)
print('Forecasting Mean squared error: %.3f' % mse_f)
print('Forecasting Standard deviation of residuals: %.3f' % np.sqrt(mse_f))
print('Forecasting R-squared: %.3f' % rsq_f)

    Forecasting Mean squared error: 171.398
    Forecasting Standard deviation of residuals: 13.092
    Forecasting R-squared: 0.915
```

The model fits so well that there was no decrement of performance in the testing data, the true forecasting situation. By chance, there was even a slight increase.

i. Is the model overfit?

No overfitting whatsoever as there was no decrement in performance from training data to testing data.

j. Cross-validate the data with five randomly selected folds, and evaluate with the average value of the standard deviation of residuals and R-squared across the folds.

```
from sklearn.model_selection import KFold, cross_validate
kf = KFold(n_splits=5, shuffle=True, random_state=1)
```

```
model = LinearRegression()
scores = cross_validate(model, X, y, cv=kf,
                        scoring=('r2', 'neg_mean_squared_error'),
                        return_train_score=True)
```

```
ds = pd.DataFrame(scores)
ds.rename(columns = {'test_neg_mean_squared_error': 'test_MSE',
                     'train_neg_mean_squared_error': 'train_MSE'},
          inplace=True)

ds['test_MSE'] = -ds['test_MSE']
ds['train_MSE'] = -ds['train_MSE']
print(ds.round(4))
print('\n')
print('Mean of test R-squared scores: %.3f' % ds['test_r2'].mean())
print('\n')
print('Mean of test MSE scores: %.3f' % ds['test_MSE'].mean())
```

```
se = np.sqrt(ds['test_MSE'].mean())
print('Standard deviation of mean test MSE scores: %.3f' % se)

       fit_time  score_time  test_r2  train_r2  test_MSE  train_MSE
    0    0.0088      0.0029   0.8931    0.9116  185.3947   173.6738
    1    0.0173      0.0042   0.9092    0.9079  187.1440   173.2429
    2    0.0038      0.0018   0.9125    0.9071  169.8875   177.5953
    3    0.0028      0.0026   0.9134    0.9070  171.7906   177.0241
    4    0.0028      0.0017   0.9082    0.9083  170.8275   177.2383


    Mean of test R-squared scores: 0.907


    Mean of test MSE scores: 177.009
    Standard deviation of mean test MSE scores: 13.304
```

Nothing quirky about the random train/test split of data. Over 5-fold cross-validation, the same approximate average s_e and R^2 values are obtained.

Feature Selection

Can we obtain the same level of forecasting accuracy with a smaller set of features?

a. Show uniqueness and relevance of each feature in a single table that consists of VIFs and target correlations. Comment.

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = pd.DataFrame()
vif['Predictor'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i)
              for i in range(X.shape[1])]
cr = d.corr()['Weight'].round(3)
vif['Relevance'] = [cr[i]
              for i in range(X.shape[1])]
vif

    /usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is deprec
      import pandas.util.testing as tm
```

	Predictor	VIF	Relevance
0	Height	420.252057	1.000
1	Waist	216.690054	0.629
2	Hips	332.174515	0.868
3	Chest	295.450906	0.788
4	ArmLength	394.156902	0.884
5	Gender_M	6.257213	0.598

All body measurement features have very high VIF's, all much, much larger than the rule of thumb cutoff of 5. So much collinearity. All the features are also relevant in terms of high correlations with the target, *Weight*. The result is that this analysis does not provide guidance as to which features to drop.

b. Run the multivariate feature selection algorithm to retain the top three features.

```
from sklearn.linear_model import LinearRegression
estimator = LinearRegression()
from sklearn.feature_selection import RFE
selector = RFE(estimator, n_features_to_select=3, step=1).fit(X,y)

print(selector.support_)
print(selector.ranking_)

    [False False  True  True False  True]
    [2 3 1 1 4 1]
```

The third, fourth, and sixth features were selected: Hips, Chest, and Gender.

c. Subset a new data frame of the feature variables that contains just these three predictors. Call it X2. Show the first several rows of data.

```
X2 = X.iloc[:, selector.support_]
X2.head()
```

	Hips	Chest	Gender_M
0	38	36	1
1	45	48	1
2	44	44	1
3	41	40	1
4	47	45	0

*d. Now fit this reduced model to all the data, X2 and y. Then generate the predicted scores into a data structure named *y_fit2.*

```
model.fit(X2, y)
y_fit2 = model.predict(X2)
```

e. Did fit suffer from reducing the number of features from 5 to 3? Comment.

```
mse = mean_squared_error(y, y_fit2)
rsq = r2_score(y, y_fit2)
print("MSE: %.3f" % mse)
se = np.sqrt(mse)
range95 = 4 * se
print("Stdev of residuals: %.3f " % se)
print("Approximate 95 per cent range of residuals: %.3f " % range95)
print("R-squared: %.3f" % rsq)

    MSE: 279.259
    Stdev of residuals: 16.711
    Approximate 95 per cent range of residuals: 66.844
    R-squared: 0.854
```

Yes, the three-feature model does not as well. s_e increased from 13.30 to 16.71. R^2 decreases from 0.91 to 0.854. The reduction in fit is real. To evaluate if the enhanced parsimony of fewer predictors is worth the decrease in fit depends on the cost of mis-predicting *Weight*, which pertains to the costs involved in a particular set of business transactions.

It would be worth experimenting with the 4-predictor model as well.