mlp-week02

January 24, 2021

# 1 Machine Learning in Python - Workshop 2

## 1.1 1. Setup

### 1.1.1 1.1 Packages

In the cell below we will load the core libraries we will be using for this workshop and setting some sensible defaults for our plot size and resolution.

[1]:
```
# Display plots inline
%matplotlib inline

# Data libraries
import pandas as pd
import numpy as np

# Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting defaults
plt.rcParams['figure.figsize'] = (8,5)
plt.rcParams['figure.dpi'] = 80
```

### 1.1.2 1.2 Data

To begin, we will examine a simple data set on the size and weight of a number of books. These data come from the `allbacks` data set from the `DAAG` package in R. Our goal is to model the weight of a book using some combination of the other features in the data. The included columns are as follows:

- `volume` - book volumes in cubic centimeters
- `area` - hard board cover areas in square centimeters
- `weight` - book weights in grams
- `cover` - a factor with levels `"hb"` hardback, `"pb"` paperback

We read the data into python using pandas,

[2]:
```python
books = pd.read_csv("daag_books.csv")
books
```

[2]:

| | volume | area | weight | cover |
|---|---|---|---|---|
| 0 | 885 | 382 | 800 | hb |
| 1 | 1016 | 468 | 950 | hb |
| 2 | 1125 | 387 | 1050 | hb |
| 3 | 239 | 371 | 350 | hb |
| 4 | 701 | 371 | 750 | hb |
| 5 | 641 | 367 | 600 | hb |
| 6 | 1228 | 396 | 1075 | hb |
| 7 | 412 | 0 | 250 | pb |
| 8 | 953 | 0 | 700 | pb |
| 9 | 929 | 0 | 650 | pb |
| 10 | 1492 | 0 | 975 | pb |
| 11 | 419 | 0 | 350 | pb |
| 12 | 1010 | 0 | 950 | pb |
| 13 | 595 | 0 | 425 | pb |
| 14 | 1034 | 0 | 725 | pb |

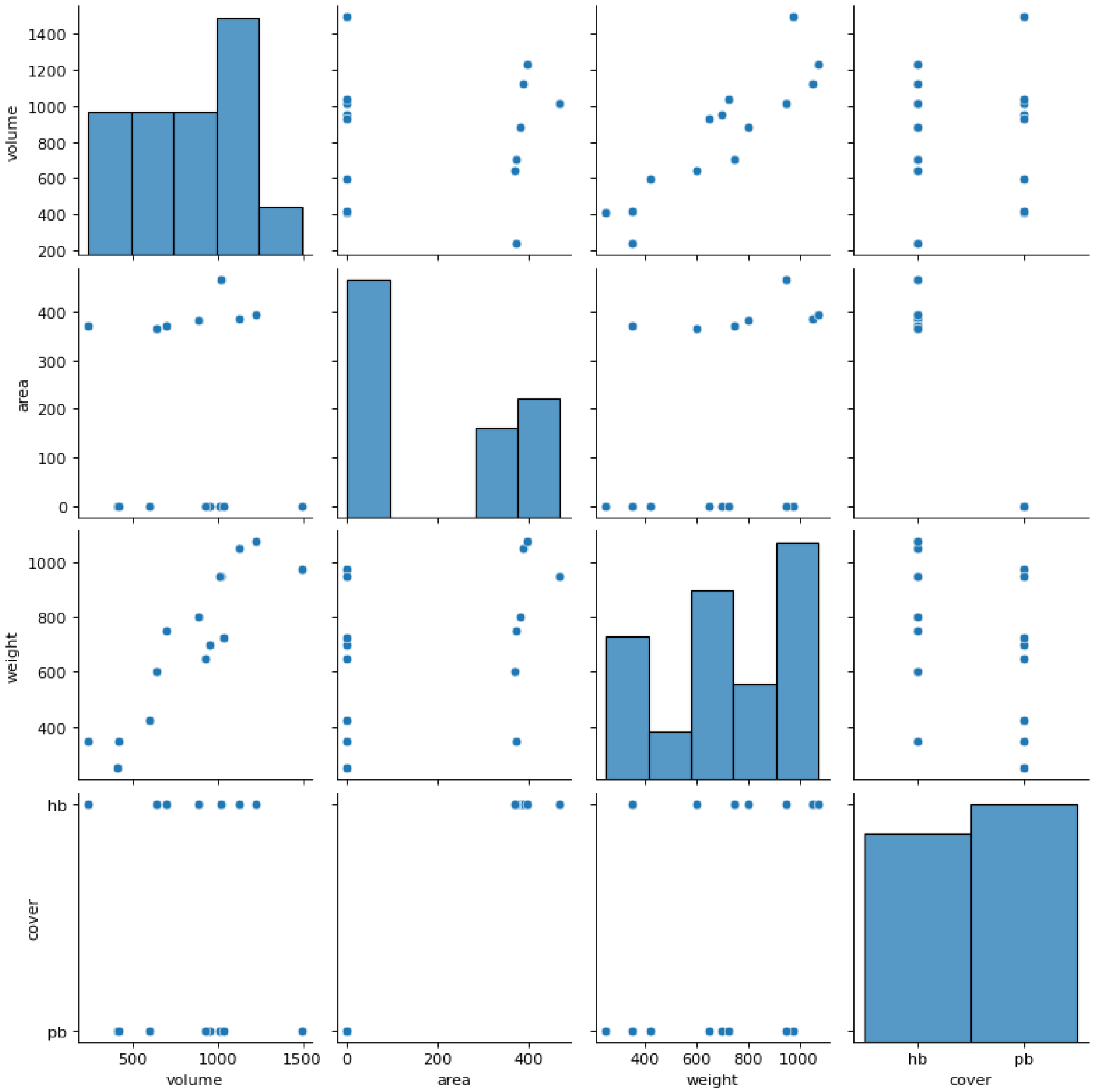
---

### 1.1.3 Exercise 1

Create a pairs plot of these data (make sure to include the `cover` column), describe any relationships you observe in the data.

[3]:
```python
sns.pairplot(books, vars=["volume","area","weight","cover"])

# There appears to be a linear relationship between weight and volume.
# The papaerback books have zero area as they dont have hard board covers
# There are no other relationships obvious from the data
```

[3]: <seaborn.axisgrid.PairGrid at 0x7f1d3ca9c590>




---

## 1.2 1. Regression

We will begin by fitting a simple linear regression model for `weight` exclusively using `volume` as a feature in our model.

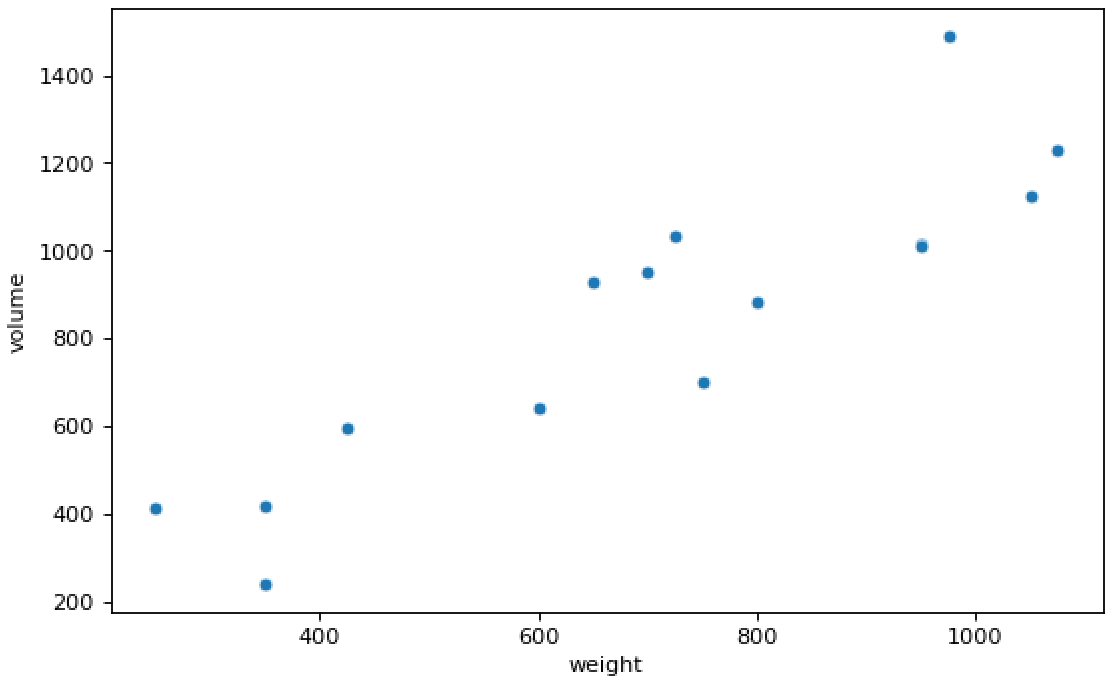
---

### 1.2.1 Exercise 2

Create a scatter plot of these data describe any apparent relationship between `weight` and `volume`.

[4]:
```
sns.scatterplot(
    x = "weight",
    y = "volume",
    data = books,
    #aspect = 1.5,
    #alpha = 0.1,
    #markers = "."
)

# It looks like there is a linear relationship
```

[4]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1d31e50150>




---

### 1.2.2 1.1 Least Squares

In lecture we discussed how we can represent a regression problem using matrix notation and we can derive a solution using least squares. We can express this as,

$$\underset{\boldsymbol{\beta}}{\operatorname{argmin}} \ \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \ (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})$$

where,

$$
\underset{n\times 1}{\boldsymbol{y}} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix} \qquad \underset{n\times 2}{\boldsymbol{X}} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_{n-1} \\ 1 & x_n \end{pmatrix} \qquad \underset{2\times 1}{\boldsymbol{\beta}} = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}
$$

The solution to this optimization problem is,

$$
\boldsymbol{\beta} = \left(\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{y}
$$

In Python we can construct the model matrix `X` by combining a column of ones, for the intercept, with our observed `volume` values. Similarly, `y` is a column vector of the `weight` values. In both cases we construct these objects as numpy array objects.

[5]:
```
y = np.array(books.weight)
print(y)
```

```
[ 800  950 1050  350  750  600 1075  250  700  650  975  350  950  425
  725]
```

[6]:
```
X = np.c_[
    np.ones(len(y)),
    books.volume
]

print(X[:5])
```

```
[[1.000e+00 8.850e+02]
 [1.000e+00 1.016e+03]
 [1.000e+00 1.125e+03]
 [1.000e+00 2.390e+02]
 [1.000e+00 7.010e+02]]
```

Given the model matrix ($\boldsymbol{X}$) and observed outcomes ($\boldsymbol{y}$) we can then calculate the vector of solutions ($\boldsymbol{\beta}$) using numpy,

[7]:
```
from numpy.linalg import solve

beta = solve(X.T @ X, X.T @ y)
print(beta)
```

```
[107.67931061   0.70863714]
```

Note that when using numpy `@` performs matrix multiplication while `*` performs elementwise multiplication between arrays. Numpy matrix multiplication can also be written using `A.dot(B)` or `np.matmul(A,B)`.

We can calculate predictions from this model by calculating $\hat{y} = \boldsymbol{X}\boldsymbol{\beta}$.

### 1.2.3 Exercise 3

Calculate these predicted book weights and store them in the origin `books` data frame in a column called `weight_ls_pred`. Print out the updated version of the data frame with this new column added.

[8]:
```
yhat = X @ beta

books['weight_ls_pred'] = yhat
print(books)
```

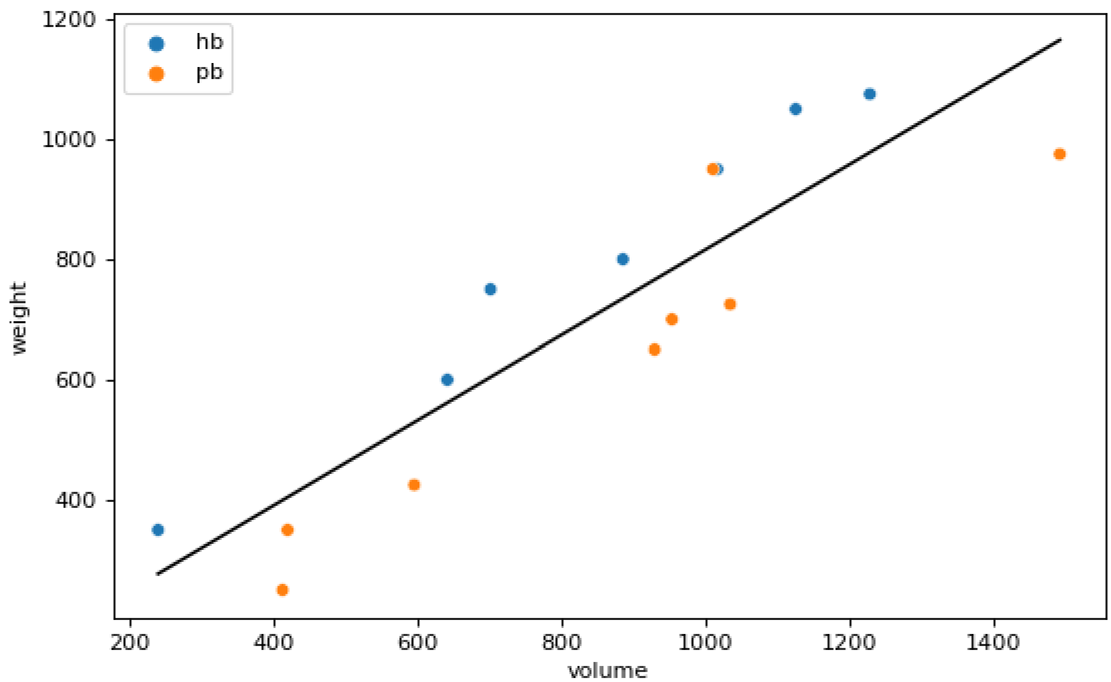
```
    volume  area  weight cover  weight_ls_pred
0      885   382     800    hb      734.823182
1     1016   468     950    hb      827.654648
2     1125   387    1050    hb      904.896097
3      239   371     350    hb      277.043588
4      701   371     750    hb      604.433948
5      641   367     600    hb      561.915720
6     1228   396    1075    hb      977.885723
7      412     0     250    pb      399.637814
8      953     0     700    pb      783.010508
9      929     0     650    pb      766.003217
10    1492     0     975    pb     1164.965929
11     419     0     350    pb      404.598274
12    1010     0     950    pb      823.402825
13     595     0     425    pb      529.318411
14    1034     0     725    pb      840.410117
```

---

Given the predictions we can create a plot showing the models fit by overlaying a line plot of the predictions on top of the original scatter plot.

[9]:
```
sns.scatterplot(x="volume", y="weight", hue="cover", data=books)
sns.lineplot(x="volume", y="weight_ls_pred", color="black", data=books)
```

[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1d31b73650>
```




---

#### 1.2.4 1.2 scikit-Learn

Constructing the model matrix by hand and calculating $\beta$ and model predictions using the least squares solution is less than ideal. As you might expect there are a number of higher level libraries that take care of many of these details. In this course we will be using the **scikit-learn** (**sklearn**) library to implement most of our machine learning models. As the semester progresses we will be learning about and implementing many different modeling methods. Additionally, we will also be learning how to use the larger data processing and workflow tools that are available in this library.

sklearn separates its various modeling tools into submodules organized by model type - for today we will be using the `LinearRegression` model from the `linear_model` submodule. Which we can import as follows,

[10]:
```
from sklearn.linear_model import LinearRegression
```

In general sklearn's models are implemented by first creating a model object, which is configured via constructor arguments, and then using that object to fit your data. As such, we will now create a linear regression model object `lr` and use it to fit our data. Once this object is created we use the `fit` method to obtain a model object fitted to our data.

[11]:
```
lr = LinearRegression()
l = lr.fit(
    # X must be a matrix so we need to reshape the column
```

```
    X = np.array(books.volume).reshape(-1,1),
    y = books.weight
)
```

This model object then has various useful methods and attributes, including `intercept_` and `coef_` which contain our estimates for $\beta$.

[12]:
```
b0 = l.intercept_
b1 = l.coef_[0]    # Subsetting here returns a scalar value
beta = (b0, b1)

print(beta)
```

```
(107.679310613766, 0.7086371433704164)
```

Using this default construction of `LinearRegression`, sklearn assumes that we have not included an intercept column (ones) in our model matrix and takes care of this for you. Additionally, since the intercept column is added the $\beta$ estimated for this particular column is stored separately, in the `intercept_` attribue.

I generally find this default behavior to be somewhat frustrating to work with, instead my preference is to handle all of the details of constructing the model matrix `X` myself and retrieving all `beta` values (including the intercept) from `coef_` directly. For example, if we use the `X` and `y` variables we defined for the least squares example above and construct the `LinearRegression` object using `fit_intercept=False` then,

[13]:
```
l = LinearRegression(fit_intercept=False).fit(X = X, y = y)
beta = l.coef_

print(beta)
```

```
[107.67931061   0.70863714]
```

Note that this is the same answers we obtained above.

The model fit objects also provide additional useful methods for evaluating the model (`score`) and calculating predictions (`predict`). Using the later we can add another column of predictions to our data frame.

[14]:
```
books["weight_sk_pred"] = l.predict(X)
books
```

[14]:

| | volume | area | weight | cover | weight_ls_pred | weight_sk_pred |
|---|---|---|---|---|---|---|
| 0 | 885 | 382 | 800 | hb | 734.823182 | 734.823182 |
| 1 | 1016 | 468 | 950 | hb | 827.654648 | 827.654648 |
| 2 | 1125 | 387 | 1050 | hb | 904.896097 | 904.896097 |
| 3 | 239 | 371 | 350 | hb | 277.043588 | 277.043588 |
| 4 | 701 | 371 | 750 | hb | 604.433948 | 604.433948 |
| 5 | 641 | 367 | 600 | hb | 561.915720 | 561.915720 |
| 6 | 1228 | 396 | 1075 | hb | 977.885723 | 977.885723 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 7 | 412 | 0 | 250 | pb | 399.637814 | 399.637814 |
| 8 | 953 | 0 | 700 | pb | 783.010508 | 783.010508 |
| 9 | 929 | 0 | 650 | pb | 766.003217 | 766.003217 |
| 10 | 1492 | 0 | 975 | pb | 1164.965929 | 1164.965929 |
| 11 | 419 | 0 | 350 | pb | 404.598274 | 404.598274 |
| 12 | 1010 | 0 | 950 | pb | 823.402825 | 823.402825 |
| 13 | 595 | 0 | 425 | pb | 529.318411 | 529.318411 |
| 14 | 1034 | 0 | 725 | pb | 840.410117 | 840.410117 |

---

### 1.2.5 Exercise 5

Do these results agree with the results we obtained when using the numpy least squares method?

[15]:
```
# Yes, they do as we can clearly see the column
# weight_ls_predicted is the same as weight_sk_predicted
```

---

### 1.2.6 1.3 Residuals

One of the most useful tools for evaluating a model is to examine the residuals of that model. For any standard regression model the residual for observation $i$ is defined as $y_i - \hat{y}_i$ where $\hat{y}_i$ is the model's predicted value for observation $i$. As mentioned previous, for the case of linear regression $\hat{y} = \boldsymbol{X}\boldsymbol{\beta}$.

---

### 1.2.7 Exercise 6

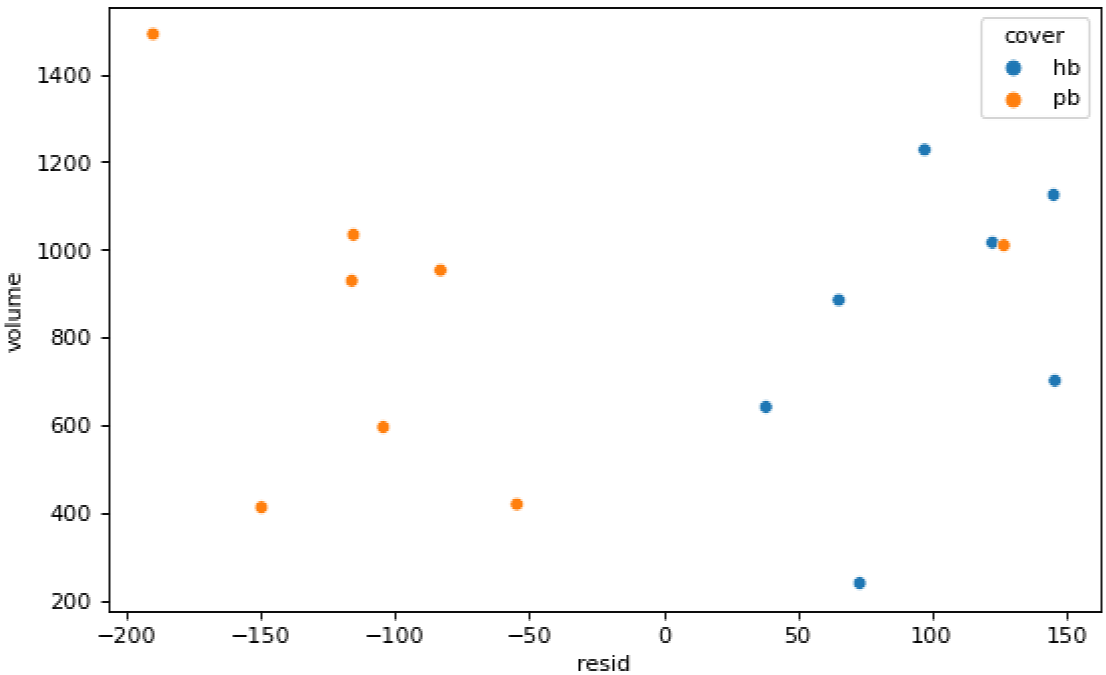
Calculate the residual for each observation and store it in a column named `resid`. Using this new column create a residual plot (scatter plot of `volume` vs `resid`) for this model. Color the points based on the `cover` type of each book.

[15]:
```
books['resid'] = books['weight'] - l.predict(X)
sns.scatterplot(x="resid",y="volume",hue="cover",data=books)
books
```

[15]:

| | volume | area | weight | cover | weight_ls_pred | weight_sk_pred | resid |
|---|---|---|---|---|---|---|---|
| 0 | 885 | 382 | 800 | hb | 734.823182 | 734.823182 | 65.176818 |
| 1 | 1016 | 468 | 950 | hb | 827.654648 | 827.654648 | 122.345352 |
| 2 | 1125 | 387 | 1050 | hb | 904.896097 | 904.896097 | 145.103903 |
| 3 | 239 | 371 | 350 | hb | 277.043588 | 277.043588 | 72.956412 |
| 4 | 701 | 371 | 750 | hb | 604.433948 | 604.433948 | 145.566052 |
| 5 | 641 | 367 | 600 | hb | 561.915720 | 561.915720 | 38.084280 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 1228 | 396 | 1075 | hb | 977.885723 | 977.885723 | 97.114277 |
| 7 | 412 | 0 | 250 | pb | 399.637814 | 399.637814 | -149.637814 |
| 8 | 953 | 0 | 700 | pb | 783.010508 | 783.010508 | -83.010508 |
| 9 | 929 | 0 | 650 | pb | 766.003217 | 766.003217 | -116.003217 |
| 10 | 1492 | 0 | 975 | pb | 1164.965929 | 1164.965929 | -189.965929 |
| 11 | 419 | 0 | 350 | pb | 404.598274 | 404.598274 | -54.598274 |
| 12 | 1010 | 0 | 950 | pb | 823.402825 | 823.402825 | 126.597175 |
| 13 | 595 | 0 | 425 | pb | 529.318411 | 529.318411 | -104.318411 |
| 14 | 1034 | 0 | 725 | pb | 840.410117 | 840.410117 | -115.410117 |




---

### 1.2.8 Exercise 7

Are there any particular issues we should be concerned about with this model based on what you see in your residual plot?

All, but one, of the paperback books have negative residuals while all of the hardback books have positive residuals. This suggests that the relationships are different between the two catagories of books as our model overpredicts one and underpredits the other. We would be better off investigating the relationships individually.

---

## 1.3 2. Regression with Categorical Variables

### 1.3.1 2.1 Dummy Coding

Based on these results, it should be clear that it is important that our model include information about whether or not a book is a hardback or paperback. As such, we need a way of encoding this information into our modeling framework. To do this we need a way of converting our string / categorical variable into a numeric representation that can be included in our model matrix.

The most common approach for doing this is called dummy coding, in the case of a binary categorical variable it involves picking one of the two levels of the categorical variable and encoding it as 1 and the other level as 0. With Python we can accomplish this by comparing our categorical vector to the value of our choice and then casting (converting) the result to an integer type.

For example if we wanted to code `hb` as 1 and `pb` as 0 we would do the following,

[16]:
```
books["cover_hb"] = (books.cover == "hb").astype(int) # Returns either 0 or 1
books
```

[16]:
```
    volume  area  weight cover  weight_ls_pred  weight_sk_pred       resid  \
0      885   382     800    hb      734.823182      734.823182   65.176818
1     1016   468     950    hb      827.654648      827.654648  122.345352
2     1125   387    1050    hb      904.896097      904.896097  145.103903
3      239   371     350    hb      277.043588      277.043588   72.956412
4      701   371     750    hb      604.433948      604.433948  145.566052
5      641   367     600    hb      561.915720      561.915720   38.084280
6     1228   396    1075    hb      977.885723      977.885723   97.114277
7      412     0     250    pb      399.637814      399.637814 -149.637814
8      953     0     700    pb      783.010508      783.010508  -83.010508
9      929     0     650    pb      766.003217      766.003217 -116.003217
10    1492     0     975    pb     1164.965929     1164.965929 -189.965929
11     419     0     350    pb      404.598274      404.598274  -54.598274
12    1010     0     950    pb      823.402825      823.402825  126.597175
13     595     0     425    pb      529.318411      529.318411 -104.318411
14    1034     0     725    pb      840.410117      840.410117 -115.410117

    cover_hb
0          1
1          1
2          1
3          1
4          1
5          1
6          1
7          0
8          0
9          0
10         0
```

```
11          0
12          0
13          0
14          0
```

This is equivalent to using an indicator function in mathematical notation,

$$\mathbb{1}_{hb_i} = \begin{cases} 1 & \text{if cover of book } i \text{ is hardback} \\ 0 & \text{if cover of book } i \text{ is paperback} \end{cases}$$

Alternatively, we can defined the opposite of this where we code `hardback` as 0 and `paperback` as 1,

[17]:
```python
books["cover_pb"] = (books.cover == "pb").astype(int) # Returns either 0 or 1
books
```

[17]:
```
    volume  area  weight cover  weight_ls_pred  weight_sk_pred       resid  \
0      885   382     800    hb      734.823182      734.823182   65.176818
1     1016   468     950    hb      827.654648      827.654648  122.345352
2     1125   387    1050    hb      904.896097      904.896097  145.103903
3      239   371     350    hb      277.043588      277.043588   72.956412
4      701   371     750    hb      604.433948      604.433948  145.566052
5      641   367     600    hb      561.915720      561.915720   38.084280
6     1228   396    1075    hb      977.885723      977.885723   97.114277
7      412     0     250    pb      399.637814      399.637814 -149.637814
8      953     0     700    pb      783.010508      783.010508  -83.010508
9      929     0     650    pb      766.003217      766.003217 -116.003217
10    1492     0     975    pb     1164.965929     1164.965929 -189.965929
11     419     0     350    pb      404.598274      404.598274  -54.598274
12    1010     0     950    pb      823.402825      823.402825  126.597175
13     595     0     425    pb      529.318411      529.318411 -104.318411
14    1034     0     725    pb      840.410117      840.410117 -115.410117

    cover_hb  cover_pb
0          1         0
1          1         0
2          1         0
3          1         0
4          1         0
5          1         0
6          1         0
7          0         1
8          0         1
9          0         1
10         0         1
11         0         1
12         0         1
```

```
13          0          1
14          0          1
```

Now that we have recoded our categorical variable, `cover`, into a numerical variable we can fit a standard regression model with the form,

$$y_i = \beta_0 + \beta_1 \, x_i + \beta_2 \, \mathbb{1}_{hb_i}$$

which we can represent in matrix form using, $\boldsymbol{y} = \boldsymbol{X\beta}$ where $\boldsymbol{X} = [\boldsymbol{1}, \, \boldsymbol{x}, \, \mathbb{1}_{\boldsymbol{hb}}]$.

Using Python, we can use the concatenate function with our 1s column, the `volume` column, and our new dummy coded indicator column, `cover_hb`,

[18]:
```
X = np.c_[np.ones(len(y)), books.volume, books.cover_hb]
l = LinearRegression(fit_intercept=False).fit(X, books.weight)

beta = l.coef_

print(beta)
```

```
[ 13.91557219   0.71795374 184.04727138]
```

This gives us a regression equation of the form,

$$y_i = 13.9 + 0.72 \, x_i + 184.0 \, \mathbb{1}_{hb_i}$$

which can be rewritten as two separate line equations (one for each case of `cover`),

$$y_i = \begin{cases} 13.9 + 0.72 \, x_i & \text{if book cover } i \text{ is paperback} \\ (13.9 + 184.0) + 0.72 \, x_i & \text{if book cover } i \text{ is hardback} \end{cases}.$$

We can calculate prediction points along those lines using the following Python code in which we hard code the possible values of $\mathbb{1}_{\boldsymbol{hb_i}}$

[19]:
```
books["weight_hb_pred"] = l.predict(X)
books
```

[19]:

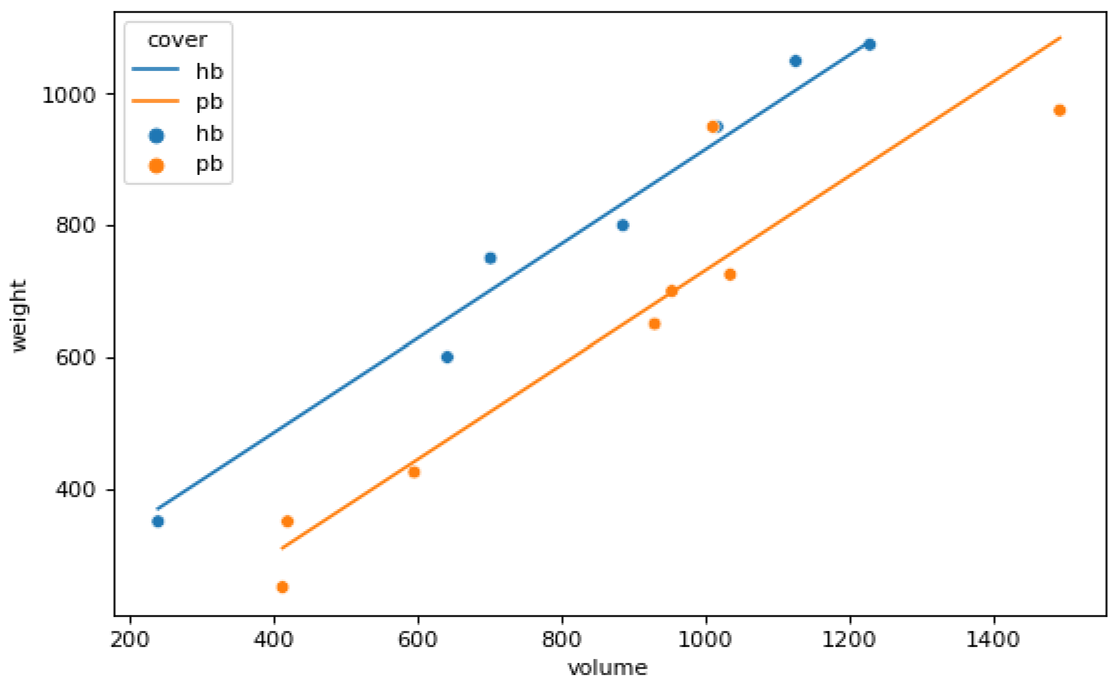
| | volume | area | weight | cover | weight_ls_pred | weight_sk_pred | resid | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 885 | 382 | 800 | hb | 734.823182 | 734.823182 | 65.176818 | |
| 1 | 1016 | 468 | 950 | hb | 827.654648 | 827.654648 | 122.345352 | |
| 2 | 1125 | 387 | 1050 | hb | 904.896097 | 904.896097 | 145.103903 | |
| 3 | 239 | 371 | 350 | hb | 277.043588 | 277.043588 | 72.956412 | |
| 4 | 701 | 371 | 750 | hb | 604.433948 | 604.433948 | 145.566052 | |
| 5 | 641 | 367 | 600 | hb | 561.915720 | 561.915720 | 38.084280 | |
| 6 | 1228 | 396 | 1075 | hb | 977.885723 | 977.885723 | 97.114277 | |
| 7 | 412 | 0 | 250 | pb | 399.637814 | 399.637814 | -149.637814 | |
| 8 | 953 | 0 | 700 | pb | 783.010508 | 783.010508 | -83.010508 | |
| 9 | 929 | 0 | 650 | pb | 766.003217 | 766.003217 | -116.003217 | |

```
10    1492       0      975     pb      1164.965929      1164.965929 -189.965929
11     419       0      350     pb       404.598274       404.598274  -54.598274
12    1010       0      950     pb       823.402825       823.402825  126.597175
13     595       0      425     pb       529.318411       529.318411 -104.318411
14    1034       0      725     pb       840.410117       840.410117 -115.410117

    cover_hb  cover_pb  weight_hb_pred
0          1         0      833.351907
1          1         0      927.403847
2          1         0     1005.660805
3          1         0      369.553788
4          1         0      701.248418
5          1         0      658.171193
6          1         0     1079.610041
7          0         1      309.712515
8          0         1      698.125490
9          0         1      680.894600
10         0         1     1085.102558
11         0         1      314.738191
12         0         1      739.048853
13         0         1      441.098050
14         0         1      756.279743
```

and we can then plot both of these lines along with the observed data.

[20]:
```
sns.scatterplot(x="volume", y="weight", hue="cover", data=books)
sns.lineplot(x="volume", y="weight_hb_pred", hue="cover", data=books)
```

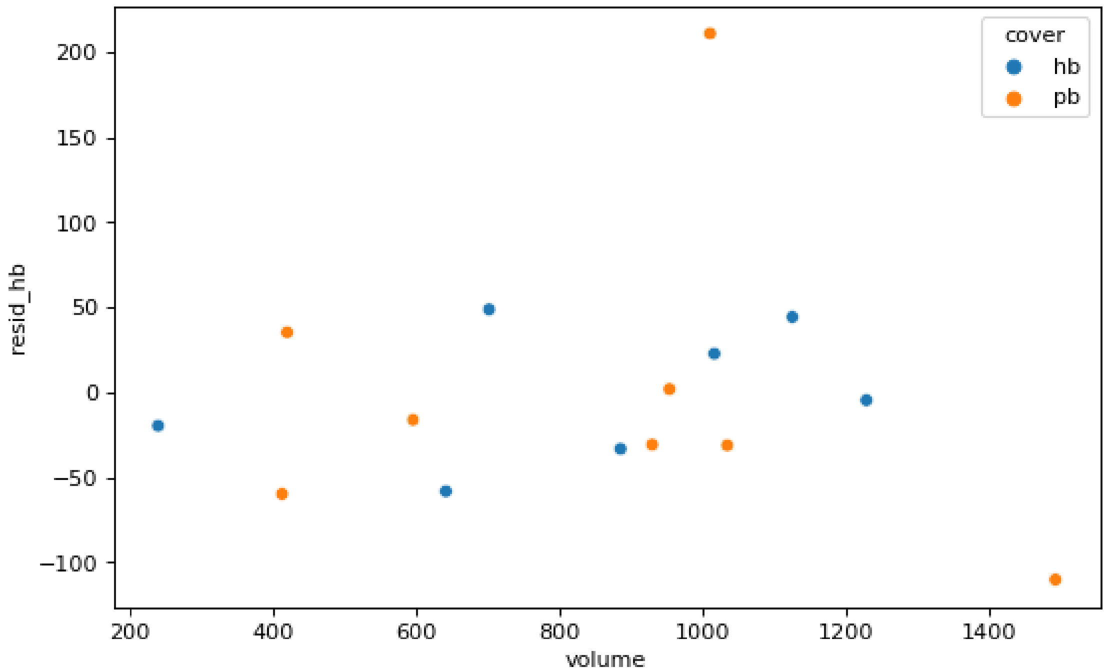
[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1d30fd44d0>
```




As well as create plot a residual plot of this new model,

[21]:
```
books["resid_hb"] = books.weight - books.weight_hb_pred
sns.scatterplot(x="volume", y="resid_hb", hue="cover", data=books)
```

[21]: <matplotlib.axes._subplots.AxesSubplot at 0x7f1d30f7d090>




---

### 1.3.2 Exercise 8

Based on these regression fits, do you think the model including the dummy coded `cover` variable produces a "better" model than our first regression model which did not include `cover`? Explain.

Clearly, it is the case that the model including the dummy coded cover variable produces a better model than the first regression model. Firstly, we can see that the regression lines better match the data points for the particular types of books (either hardback or paperback).

Interestingly, it is the case that the residuals in the first model are closer to zero than the second. (However, they are both have negligible absolute value)

[28]:
```
sum_original_residuals = (books["resid"]).sum()
sum_new_residuals = (books["resid_hb"]).sum()

print(sum_original_residuals, sum_new_residuals)
```

```
-3.410605131648481e-13 3.461764208623208e-11
```

---

Note that by including a dummy variable in our model will change the interpretation of our regression coefficients. In this context,

- $\beta_0$ - This is the expected weight of a book with a `volume` of zero and a `hardback` indicator of zero, in other words a softcover book with zero volume.
- $\beta_1$ - This is the expected additional weight a book would have if its volume were to increase by 1 $cm^3$, all else being equal.
- $\beta_2$ - This is the expected additional weight a book would have if its hardcover indicator were to increase by 1, all else being equal. However, the hardcover indicator can only be 0 or 1 and hence this is the change in weight we would expect between a softcover book and a hardcover book with the same volume. In other words, hardcover books weight 184g more than softcover books.

Based on these interpretations we can see that the level that was coded as 0 (what is often called the reference level) gets folded into our intercept and the slope coefficient for the indicator provides the difference in intercept between the reference and the contrast level (level coded as 1).

---

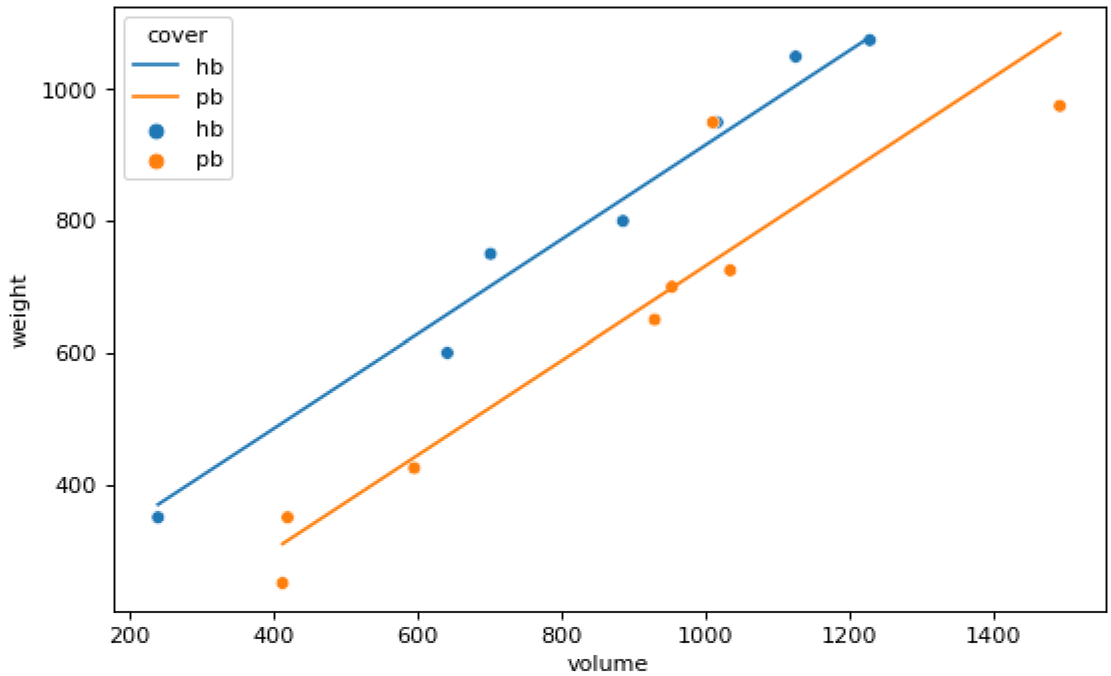
### 1.3.3 Exercise 9

Repeat the analysis above but this time fit a model using `pb` instead of `hb` in your model matrix. You should fit the new model as well as calculate the predictions for both paperback and hardback books.

[37]:
```
X2 = np.c_[np.ones(len(y)), books.volume, books.cover_pb]
l2 = LinearRegression(fit_intercept=False).fit(X2, books.weight)

beta = l2.coef_

print(beta)

books["weight_pb_pred"] = l2.predict(X2)

sns.scatterplot(x="volume", y="weight", hue="cover", data=books)
sns.lineplot(x="volume", y="weight_pb_pred", hue="cover", data=books)
```

```
[ 197.96284357    0.71795374 -184.04727138]
```

[37]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1d30bcd7d0>
```



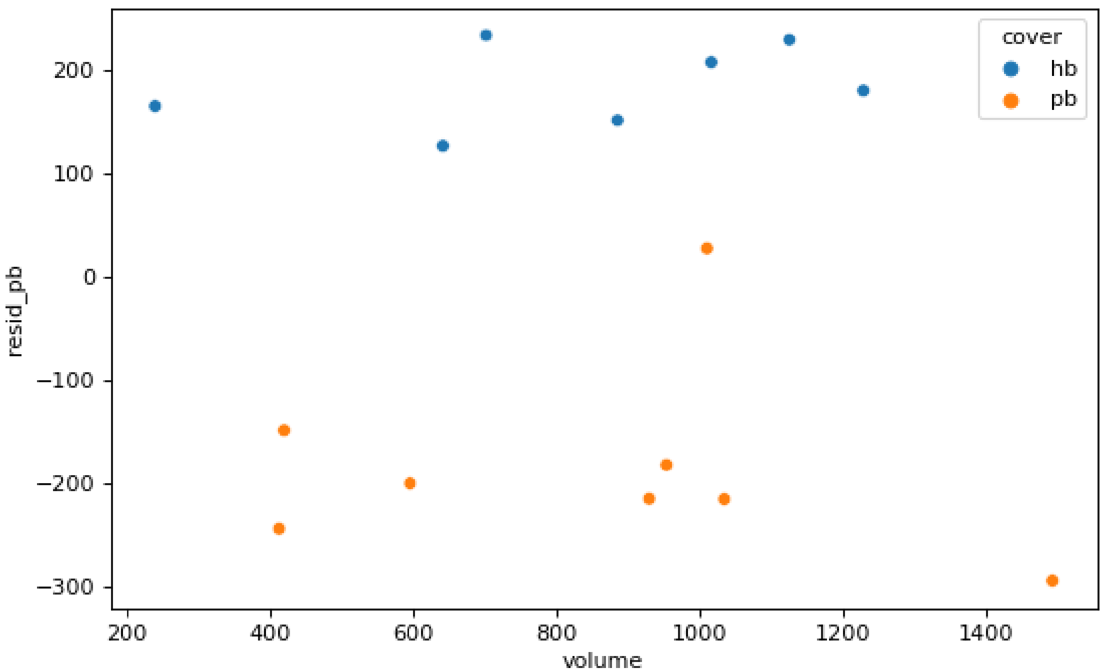

[34]:
```
books["resid_pb"] = books.weight - books.weight_pb_pred
sns.scatterplot(x="volume", y="resid_pb", hue="cover", data=books)
```

[34]:

| | volume | area | weight | cover | weight_ls_pred | weight_sk_pred | resid | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 885 | 382 | 800 | hb | 734.823182 | 734.823182 | 65.176818 | |
| 1 | 1016 | 468 | 950 | hb | 827.654648 | 827.654648 | 122.345352 | |
| 2 | 1125 | 387 | 1050 | hb | 904.896097 | 904.896097 | 145.103903 | |
| 3 | 239 | 371 | 350 | hb | 277.043588 | 277.043588 | 72.956412 | |
| 4 | 701 | 371 | 750 | hb | 604.433948 | 604.433948 | 145.566052 | |
| 5 | 641 | 367 | 600 | hb | 561.915720 | 561.915720 | 38.084280 | |
| 6 | 1228 | 396 | 1075 | hb | 977.885723 | 977.885723 | 97.114277 | |
| 7 | 412 | 0 | 250 | pb | 399.637814 | 399.637814 | -149.637814 | |
| 8 | 953 | 0 | 700 | pb | 783.010508 | 783.010508 | -83.010508 | |
| 9 | 929 | 0 | 650 | pb | 766.003217 | 766.003217 | -116.003217 | |
| 10 | 1492 | 0 | 975 | pb | 1164.965929 | 1164.965929 | -189.965929 | |
| 11 | 419 | 0 | 350 | pb | 404.598274 | 404.598274 | -54.598274 | |
| 12 | 1010 | 0 | 950 | pb | 823.402825 | 823.402825 | 126.597175 | |
| 13 | 595 | 0 | 425 | pb | 529.318411 | 529.318411 | -104.318411 | |
| 14 | 1034 | 0 | 725 | pb | 840.410117 | 840.410117 | -115.410117 | |

| | cover_hb | cover_pb | weight_hb_pred | resid_hb | weight_pb_pred | resid_pb |
|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 833.351907 | -33.351907 | 649.304635 | 150.695365 |
| 1 | 1 | 0 | 927.403847 | 22.596153 | 743.356576 | 206.643424 |
| 2 | 1 | 0 | 1005.660805 | 44.339195 | 821.613534 | 228.386466 |
| 3 | 1 | 0 | 369.553788 | -19.553788 | 185.506517 | 164.493483 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 1 | 0 | 701.248418 | 48.751582 | 517.201147 | 232.798853 |
| 5 | 1 | 0 | 658.171193 | -58.171193 | 474.123922 | 125.876078 |
| 6 | 1 | 0 | 1079.610041 | -4.610041 | 895.562770 | 179.437230 |
| 7 | 0 | 1 | 309.712515 | -59.712515 | 493.759786 | -243.759786 |
| 8 | 0 | 1 | 698.125490 | 1.874510 | 882.172761 | -182.172761 |
| 9 | 0 | 1 | 680.894600 | -30.894600 | 864.941872 | -214.941872 |
| 10 | 0 | 1 | 1085.102558 | -110.102558 | 1269.149829 | -294.149829 |
| 11 | 0 | 1 | 314.738191 | 35.261809 | 498.785462 | -148.785462 |
| 12 | 0 | 1 | 739.048853 | 210.951147 | 923.096125 | 26.903875 |
| 13 | 0 | 1 | 441.098050 | -16.098050 | 625.145321 | -200.145321 |
| 14 | 0 | 1 | 756.279743 | -31.279743 | 940.327015 | -215.327015 |




[38]:
```
books
```

[38]:

| | volume | area | weight | cover | weight_ls_pred | weight_sk_pred | resid | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 885 | 382 | 800 | hb | 734.823182 | 734.823182 | 65.176818 | |
| 1 | 1016 | 468 | 950 | hb | 827.654648 | 827.654648 | 122.345352 | |
| 2 | 1125 | 387 | 1050 | hb | 904.896097 | 904.896097 | 145.103903 | |
| 3 | 239 | 371 | 350 | hb | 277.043588 | 277.043588 | 72.956412 | |
| 4 | 701 | 371 | 750 | hb | 604.433948 | 604.433948 | 145.566052 | |
| 5 | 641 | 367 | 600 | hb | 561.915720 | 561.915720 | 38.084280 | |
| 6 | 1228 | 396 | 1075 | hb | 977.885723 | 977.885723 | 97.114277 | |
| 7 | 412 | 0 | 250 | pb | 399.637814 | 399.637814 | -149.637814 | |
| 8 | 953 | 0 | 700 | pb | 783.010508 | 783.010508 | -83.010508 | |
| 9 | 929 | 0 | 650 | pb | 766.003217 | 766.003217 | -116.003217 | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 10 | 1492 | 0 | 975 | pb | 1164.965929 | 1164.965929 | -189.965929 |
| 11 | 419 | 0 | 350 | pb | 404.598274 | 404.598274 | -54.598274 |
| 12 | 1010 | 0 | 950 | pb | 823.402825 | 823.402825 | 126.597175 |
| 13 | 595 | 0 | 425 | pb | 529.318411 | 529.318411 | -104.318411 |
| 14 | 1034 | 0 | 725 | pb | 840.410117 | 840.410117 | -115.410117 |

| | cover_hb | cover_pb | weight_hb_pred | resid_hb | weight_pb_pred | resid_pb |
|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 833.351907 | -33.351907 | 833.351907 | 150.695365 |
| 1 | 1 | 0 | 927.403847 | 22.596153 | 927.403847 | 206.643424 |
| 2 | 1 | 0 | 1005.660805 | 44.339195 | 1005.660805 | 228.386466 |
| 3 | 1 | 0 | 369.553788 | -19.553788 | 369.553788 | 164.493483 |
| 4 | 1 | 0 | 701.248418 | 48.751582 | 701.248418 | 232.798853 |
| 5 | 1 | 0 | 658.171193 | -58.171193 | 658.171193 | 125.876078 |
| 6 | 1 | 0 | 1079.610041 | -4.610041 | 1079.610041 | 179.437230 |
| 7 | 0 | 1 | 309.712515 | -59.712515 | 309.712515 | -243.759786 |
| 8 | 0 | 1 | 698.125490 | 1.874510 | 698.125490 | -182.172761 |
| 9 | 0 | 1 | 680.894600 | -30.894600 | 680.894600 | -214.941872 |
| 10 | 0 | 1 | 1085.102558 | -110.102558 | 1085.102558 | -294.149829 |
| 11 | 0 | 1 | 314.738191 | 35.261809 | 314.738191 | -148.785462 |
| 12 | 0 | 1 | 739.048853 | 210.951147 | 739.048853 | 26.903875 |
| 13 | 0 | 1 | 441.098050 | -16.098050 | 441.098050 | -200.145321 |
| 14 | 0 | 1 | 756.279743 | -31.279743 | 756.279743 | -215.327015 |

---

### 1.3.4 Exercise 10

What changed between the model with `cover_pb` vs the model with `cover_hb`? Specifically, comment on the values of $\beta_0$, $\beta_1$, and $\beta_2$ and their interpretations.

In the model `cover_pb` it is the case that

$$y_i = \beta_0 + \beta_1 \, x_i + \beta_2 \mathbb{1}_{pb_i}$$

where,

$$y_i = 197.96 + 0.72 \, x_i - 184.05 \mathbb{1}_{hb_i}$$

- $\beta_0$ - This is the expected weight of a book with a `volume` of zero and a `paperback` indicator of zero, in other words a hardcover book with zero volume.
- $\beta_1$ - This is the expected additional weight a book would have if its volume were to increase by 1 $\text{cm}^3$, all else being equal.
- $\beta_2$ - This is the expected additional weight a book would have if its papercover indicator were to increase by 1, all else being equal. However, the papercover indicator can only be 0 or 1 and hence this is the change in weight we would expect between a softcover book and a

hardcover book with the same volume. In other words, softcover books weight 184.05g less than hardcover books.

---

### 1.3.5 2.2 One hot encoding

Another common approach for transforming categorical variables is know as one hot encoding, in which all levels of the categorical variable are transformed into a new columns with values of 0 or 1. This is equivalent to what we have done manually above by including both `cover_hb` and `cover_pb`. This differs from dummy coding in that there is no longer a reference factor.

Pandas has a built-in method for performing this on categorical columns. This is easiest to see with a simple example, below we construct a data frame `df` with a single column that we transform into a one hot encoded version using panda's `get_dummies` method.

[23]:
```
df = pd.DataFrame({"col": ["A","B","C","A", np.nan]})
df
```

[23]:

| | col |
|---|---|
| 0 | A |
| 1 | B |
| 2 | C |
| 3 | A |
| 4 | NaN |

[24]:
```
pd.get_dummies(df)
```

[24]:

| | col_A | col_B | col_C |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 |
| 3 | 1 | 0 | 0 |
| 4 | 0 | 0 | 0 |

We can also perform typical statistical dummy coding by using the `drop_first=True` argument, which excludes the first column as a reference level.

[25]:
```
pd.get_dummies(df, drop_first=True)
```

[25]:

| | col_B | col_C |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 2 | 0 | 1 |
| 3 | 0 | 0 |
| 4 | 0 | 0 |

Missing values can also be included as an additional category via the `dummy_na=True` argument. This treats missing values as an additional category for the provided factor.

[26]:
```
pd.get_dummies(df, dummy_na=True)
```

[26]:

| | col_A | col_B | col_C | col_nan |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 |

We can now use this approach with sklearn's linear regression model to simplify the process of creating our model with both `volume` and `cover`. Using `get_dummies` on a clean copy of the `books` data frame automatically replaces the `cover` column with `cover_hb` and `cover_pb`, which should match the columns we created by hand above.

[27]:
```
books = pd.read_csv("daag_books.csv") # Reread in the data for a clean copy
books = pd.get_dummies(books)
books
```

[27]:

| | volume | area | weight | cover_hb | cover_pb |
|---|---|---|---|---|---|
| 0 | 885 | 382 | 800 | 1 | 0 |
| 1 | 1016 | 468 | 950 | 1 | 0 |
| 2 | 1125 | 387 | 1050 | 1 | 0 |
| 3 | 239 | 371 | 350 | 1 | 0 |
| 4 | 701 | 371 | 750 | 1 | 0 |
| 5 | 641 | 367 | 600 | 1 | 0 |
| 6 | 1228 | 396 | 1075 | 1 | 0 |
| 7 | 412 | 0 | 250 | 0 | 1 |
| 8 | 953 | 0 | 700 | 0 | 1 |
| 9 | 929 | 0 | 650 | 0 | 1 |
| 10 | 1492 | 0 | 975 | 0 | 1 |
| 11 | 419 | 0 | 350 | 0 | 1 |
| 12 | 1010 | 0 | 950 | 0 | 1 |
| 13 | 595 | 0 | 425 | 0 | 1 |
| 14 | 1034 | 0 | 725 | 0 | 1 |

Note that `get_dummies` does not modify the underlying dataframe in place, and that it is necessary to save the result to a new variable (or overwrite the old version).

---

### 1.3.6 2.2 Least Squares & rank deficiency

Now lets consider the model where we naively include both `cover_hb` and `cover_pb` as well as an intercept column in our model matrix.

[41]:
```
X = np.c_[
    np.ones(len(y)),
    books.volume,
```

```
    books.cover_hb,
    books.cover_pb
]
l = LinearRegression(fit_intercept=False).fit(X, books.weight)


beta = l.coef_
print( beta )
```

```
[ 70.62613859   0.71795374 127.33670498 -56.7105664 ]
```

---

### 1.3.7 Exercise 11

Write out the equations that predict weight for hardback and paperback books according to this model.

Our equation is

$$y_i = \beta_0 + \beta_1\, x_i + \beta_2 \mathbb{1}_{hb_i} + \beta_3 \mathbb{1}_{pb_i}.$$

that is,

$$y_i = 70.63 + 0.72\, x_i + 127.34 \mathbb{1}_{hb_i} - 56.71 \mathbb{1}_{pb_i}.$$

Paperbacks are given by,

$$y_i = (70.63 - 56.71) + 0.72\, x_i.$$
$$y_i = 13.92 + 0.72\, x_i.$$

Hardbacks are given by,

$$y_i = (70.63 + 127.34) + 0.72\, x_i.$$
$$y_i = 197.97 + 0.72\, x_i.$$

---

### 1.3.8 Exercise 12

Are the solutions ($\beta$) given above unique? Can you find different values of $\beta_0$, $\beta_1$, $\beta_2$, and $\beta_3$ that would give you the same regression equations you wrote out in the previous exercise?

We can set $\beta_3 = 0$ then $\beta_0 = 13.92$, $\beta_2 = 184.05$ and $\beta_1 = 0.72$ which gives,

$$y_i = 13.92 + 0.72\,x_i + 184.05\mathbb{1}_{hb_i}.$$

These coefficients give the same regresion equations.

Likewise, we can set $\beta_2 = 0$ then $\beta_0 = 197.97$, $\beta_2 = -184.05$ and $\beta_1 = 0.72$ which gives,

$$y_i = 197.97 + 0.72\,x_i - 184.05\mathbb{1}_{pb_i}.(*)$$

These coefficients also give the same regresion equations.

---

#### 1.3.9 Exercise 13

Solve for $\beta$ using the numpy approach mentioned in Section 1. Do the solutions differ from sklearn's solutions? Do they make sense? Explain.

[39]:
```
beta = solve(X.T @ X, X.T @ y)
print(beta)
```

```
[ 197.96284357    0.71795374 -184.04727138]
```

The numpy solutions are the same as equation $(*)$ above, It appears that when doing the matrix multiplication numpy accounts for the rank deficiency and produces the unique solution to the multiplication. Hence, dissagreeing with the solution from sklearn.

This solution is more accurate

---

The issues we are seeing with the above approaches are occuring due to colinearity between our predictors - if you examine the data it should be clear that given any two of the intercept, `cover_hb`, and `cover_pb` it is possible to exactly determine the value of the other column. Mathematically, we describe this as these columns are linearly depenedent, which implies that our model matrix is *rank deficient*. You can check this explicitly by via the `numpy.linalg.matrix_rank` function which will report that `X` (and $\boldsymbol{X}^\top\boldsymbol{X}$) are of rank 3 not 4 which is what we might have naievely expected.

This is important as the underlying linear algrebra methods used to solve for $\beta$ for a least squares problem often implicitly assume that $\boldsymbol{X}^\top\boldsymbol{X}$ is full rank in order to solve the matrix inverse and violating these assumptions can have unexpected results.

---

### 1.4 3. Competing the worksheet

At this point you have hopefully been able to complete all the preceeding exercises. Now is a good time to check the reproducibility of this document by restarting the notebook's kernel and rerunning all cells in order.

Once that is done and you are happy with everything, you can then run the following cell to generate your PDF and turn it in on gradescope under the `mlp-week02` assignment.

[29]:
```
!jupyter nbconvert --to pdf mlp-week02.ipynb
```

```
[NbConvertApp] Converting notebook mlp-week02.ipynb to pdf
[NbConvertApp] Support files will be in mlp-week02_files/
[NbConvertApp] Making directory ./mlp-week02_files
[NbConvertApp] Making directory ./mlp-week02_files
[NbConvertApp] Making directory ./mlp-week02_files
[NbConvertApp] Making directory ./mlp-week02_files
[NbConvertApp] Making directory ./mlp-week02_files
[NbConvertApp] Making directory ./mlp-week02_files
[NbConvertApp] Writing 70825 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', './notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', './notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 184707 bytes to mlp-week02.pdf
```