

## Exercises

### Jumping Rivers

#### Exercise 1: Slopes and intercepts

For graphs A, B & C, draw what you think is the line of best fit through the points, then calculate the slope and the gradient

Graph	$\beta_0$	$\beta_1$
A		
B		
C		

#### Exercise 2: Residuals

With your new knowledge of residuals, redraw the lines of best fits for graphs A, B & C. Calculate the residuals and RSS for each line you've drawn. Do you think there's a better line with a smaller RSS?

Graph	$\beta_0$	$\beta_1$	RSS
A			
B			
C			

#### Exercise 3: Loading the data

Create a new ipython notebook, and call it "python\_exercises". You'll benefit from having the workings from these exercise stored in a different notebook to the rest of your working. The following code will load the data in.

```
import pandas as pd
import jupyterml
hd = jupyterml.datasets.load_head_size()
hd.head()

##   gender age_range head_size brain_weight height
## 0   Male    20-46     4512         1530      69
## 1   Male    20-46     3738         1297      72
## 2   Male    20-46     4261         1335      77
## 3   Male    20-46     3777         1282      72
## 4   Male    20-46     4177         1590      65
```

This is a data set consisting of 237 people, where we have collected their gender, age range, head size ( $\text{cm}^3$ ), brain weight (grams) and

height (inches). Let's say we're a scientist who is interested in predicting a person's brain weight based upon their head size.

- a) Write down the simple linear regression model we could use to do this.

```
"brain_weight = b0 + b1*head_size"
```

- b) Set up the correct training data and response variable for this.

```
X = hd.drop(columns="brain_weight")
y = hd["brain_weight"]
```

#### *Exercise 4: Visualising the data*

Using `seaborn`, produce a scatter plot of the head size against brain weight. What does the graph tell you?

```
import seaborn as sns
# sns.scatterplot(x = "head_size", y = "brain_weight", data = hd)
# brain weight increases with head size
```

#### *Exercise 5: fitting the model*

Fit the simple linear regression model you described in exercise 3

```
from sklearn import linear_model
X_train = X["head_size"]
y_train = y
X_train = X_train.values.reshape(-1, 1)
model = linear_model.LinearRegression()
model.fit(X_train, y_train)
```

#### *Exercise 6: predictions*

- a) A patient comes into a doctor's with a head size of 5000 cm<sup>3</sup>. After assessing an MRI, Dr Frankenstein predicts that his brain weight is between 1500-1550 grams. Is he right?

```
import numpy as np
new_value = np.array(5000, ndmin=2)
model.predict(new_value)
```

- b) After seeing how great your model is, Dr Frankenstein would like some help in assessing the brain weight of his 3 new patients. Their head sizes are 2500, 3000 & 4500. Do this inside one call to `model.predict()`.

```
multiple_new_values = np.array([1000, 3000, 5000], ndmin=2).T
model.predict(multiple_new_values)
```

*Exercise 7: fitted values*

Using the fitted values, overlay the model line over the scatter plot you produced in exercise 4

```
fitted = model.predict(X_train)
sns.scatterplot(x="head_size", y="brain_weight", data=hd)
sns.lineplot(x=hd["head_size"], y=fitted, color="red")
```

*Exercise 8: residuals*

Using the fitted values, calculate the residual sum of squares

```
resid = fitted - y_train
np.square(resid).sum()
```

*Exercise 9: model coefficients*

- a) Can you write down  $\beta_0$  and  $\beta_1$  for your model? We've seen how to extract  $\beta_1$  in the notes. For  $\beta_0$ , take a look at the methods available with `dir(model)`. Remember,  $\beta_0$  is the y-intercept.

```
model.intercept_
model.coef_
```

- b) What do these coefficients tell you about the relationship between head size and brain weight?

*Exercise 10: multiple linear regression*

Dr Frankenstein thinks that not only does head size affect brain weight, but that there also might be a relationship between a person's height and their brain weight.

- a) Write down the multiple linear regression model you would now fit to examine this claim.

```
X_train = X[["height", "head_size"]]
```

- b) Train the model.

```
model.fit(X_train, y_train)
```

- c) What are the coefficients for your model?

```
model.coef_
```

- d) What is the residual sum of squares for your model? How does this compare to your previous model?

```
fitted = model.predict(X_train)
resid = fitted - y_train
np.square(resid).sum()
```

e) Dr Frankenstein has 3 new patients:

Patient	Head size	Height
1	4000	80
2	3000	70
3	2000	60

Give him an estimate of each patients brain weight using your model. Hint: create a **pandas** DataFrame containing the new values to pass to `model.predict()`

```
new_values = pd.DataFrame({"height": [80, 70, 60],
                           "head_size": [4000, 3000, 2000]})
model.predict(new_values)
```

#### *Exercise 11: standardised residuals*

Calculate the standardised residuals for the model in exercise 10.

```
std_resid = (resid - np.mean(resid))/np.std(resid, ddof=1)
```

#### *Exercise 12: polynomials*

Come up with graphs for the third and fourth order polynomials,  $y = X^3$  and  $y = X^4$

```
x = np.arange(-5, 6, 0.1)
y_poly_3 = x*x*x
y_poly_4 = x*x*x*x
sns.lineplot(x, y_poly_3)
sns.lineplot(x, y_poly_4)
```

#### *Exercise 13: More polynomials*

The following code generates some data and places it in a pandas data frame.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
x = np.arange(-5, 5.1, 0.1)
```

```

y = pow(x, 3) + np.sin(x) + np.random.normal(loc=0, scale=20, size=101)
example = pd.DataFrame({
    "x": x,
    "y": y
})

```

- a) Create a scatter plot of x against y. Do you think a linear line would accurately represent the data? If not, what order polynomial do you think would suit this data?

```
sns.scatterplot(x="x", y="y", data=example)
```

- b) Using the PolynomialFeatures() function, appropriately transform your predictor, x, and fit your model.

```

example_predictors = example["x"]
example_response = example["y"]
from sklearn.preprocessing import PolynomialFeatures
example_predictors = example_predictors.values.reshape(-1, 1)
polynomial_features = PolynomialFeatures(degree=3)
# transform predictors
example_predictors_poly = polynomial_features.fit_transform(example_predictors)
model_cubed = linear_model.LinearRegression()
model_cubed.fit(example_predictors_poly, example_response)

```

- c) Using the fitted values, overlay your model line onto the scatter plot produced earlier in the exercise.

```

example_poly_pred = model_cubed.predict(example_predictors_poly)
sns.scatterplot(x="x", y="y", data=example)
sns.lineplot(x=x, y=example_poly_pred, color="red")

```

### *Exercise 14: Standardisation*

Given that this is now day 2, we've probably lost the data. So here is the code to load it in again

```

import pandas as pd
import jupyterml
hd = jupyterml.datasets.load_head_size()

```

- a) Create a new variable called "head\_size\_stand", that is the standardised version of head\_size. For now, don't use the *preprocessing* module. Do it using only **NumPy** functionality.

```

import numpy as np
import pandas as pd
hd["head_size_stand"] = (hd.head_size - np.mean(hd.head_size))/np.var(hd.head_size)

```

- b) Produce a boxplot of the new standardised head size, comparing males and females. Which gender has the biggest head (Metaphorically it's men, obviously)? *Hint:* use `sns.boxplot()`

```
import seaborn as sns
# sns.boxplot(x = "gender", y = "head_size_stand", data = hd)
```

- c) Let's say we want to fit the model  $brainweight = \beta_0 + \beta_1 \times headsize + \beta_2 \times height$ . Set up your `X_train` and `y_train` objects, and standardise both `head_size` and `height` but this time using `StandardScaler()`

```
X_train = hd[["head_size", "height"]]
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
```

- d) Fit the model and predict the brain weight for a person with head size  $4000cm^3$  and 70 inches tall.

```
new_values = pd.DataFrame({
    "head_size": [4000],
    "height": [70]
})
new_scaled = scaler.transform(new_values)
pred = model.predict(new_scaled)
pred
```

### *Exercise 15: Min-Max*

Repeat exercise 14, but using the min max transform instead of the standardisation transform. Do you get the same prediction as exercise 14?

```
hd["head_size_stand"] = (hd.head_size - hd.head_size.min()) / (hd.head_size.max()
    - hd.head_size.min())
sns.boxplot(x="gender", y="head_size_stand", data=hd)
```

```
from sklearn import preprocessing
scaler = preprocessing.MinMaxScaler()
```

```
X_train = hd[["head_size", "height"]]
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
```

```
model = linear_model.LinearRegression()
```

```

model.fit(X_train_scaled, y_train)

new_values = pd.DataFrame({
    "head_size": [4000],
    "height": [70]
})
new_scaled = scaler.transform(new_values)
pred = model.predict(new_scaled)
pred

```

### *Exercise 16: Pipelines*

Repeat the previous prediction, but this time do it by setting up a pipeline that first does a min-max transformation, and then second performs linear regression. You should get the same result.

```

from sklearn.pipeline import Pipeline
model = Pipeline(
    steps=[
        ('preprocess', preprocessing.MinMaxScaler()),
        ('regression', linear_model.LinearRegression())
    ]
)
model.fit(X_train, y_train)
model.predict(new_values)

```

### *Exercise 17: Categorical data*

Up until this point, we've pretty much ignored the fact that we have two categorical variables in the data, `gender` and `age_range`. Let's say we want to build a model, that is able to estimate someone's brain weight by their gender and nothing else.

- a) What do you think the predicted value of brain weight for each gender will be?
- b) Set up your `X_train` variable such that it only has `gender` in it

```
X_train = hd[["gender"]]
```

- c) Set up a model pipeline that preprocesses the data via a one hot encoding scheme, then using linear regression.

```

model = Pipeline(
    steps=[
        ('preprocess', preprocessing.OneHotEncoder()),
        ('regression', linear_model.LinearRegression())
    ]
)

```

- d) Train the model and then predict what brain sizes a Male and Female would have.

```
model.fit(X_train, y_train)
model.predict(np.array(["Male"], ndmin=2))
new_value = pd.DataFrame({"gender": ["Male", "Female"]})
model.predict(new_value)
```

- e) What model have you trained? *Hint:* Look at `model.named_steps["regression"].intercept_` and `model.named_steps["regression"].coef_`.

```
model.named_steps["regression"].intercept_
model.named_steps["regression"].coef_
# brain_weight = 1283 - 63*Female + 48*Male
# where Male and Female are binary 1 and 0
```

- f) What is the average of brain weight for men and women in this data set? How does this compare with your previous predictions? Why do you think this is?

```
hd.groupby("gender").brain_weight.mean()
# the same because
# we only have two possible unique data points in our
# predictor variable, Male and Female. So there can
# only be two unique points in the response variable.
# This turns out to be the average in each group.
```

- g) What is the RSS for this model? How does this compare to the RSS you calculated in question 8?

```
fitted = model.predict(X_train)
np.square(fitted - y_train).sum()
```

### *Exercise 18: Combining steps*

- Let's say we want to try both gender and head size as predictors. a) Set up the correct `X_train` variable

```
X_train = hd[["gender", "head_size"]]
```

- b) Using `ColumnTransformer`, create a preprocessor that one hot encodes gender and does a standardisation transform on `head_size`.

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

numeric_features = ['head_size']
```

```

categorical_features = ['gender']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat', OneHotEncoder(), categorical_features)
    ]
)

```

- c) Create a pipeline where the preprocessor is the first step, and a linear regression model is the second step

```

model = Pipeline(
    steps=[
        ('preprocess', preprocessor),
        ('regression', linear_model.LinearRegression())
    ]
)

```

- d) Fit the model, and predict the brain weight of a Male with a  $4000\text{cm}^3$  head and a Female with a  $2000\text{cm}^3$  head.

```

model.fit(X_train, y_train)
new_values = pd.DataFrame({
    "gender": ["Male", "Female"],
    "head_size": [4000, 2000]
})
model.predict(new_values)

```

- e) What is the residual sum of squares of this model? How does it compare to the RSS in questions 8 & 17?

```

fitted = model.predict(X_train)
np.square(fitted - y_train).sum()

```

- f) Draw the fitted values against head\_size, what do you notice?

```

import seaborn as sns
# sns.scatterplot(x = hd["head_size"], y = fitted)
# two lines, one for female, one for male.

```

### *Exercise 19: Validation set approach*

Let's go back to our very first model,  $\text{brainweight} = \beta_0 + \beta_1 \times \text{headsize}$ . Set up your predictor and response like so

```

X = hd["head_size"]
y = hd["brain_weight"]

```

- a) Split the data into a training and validation set, using `train_test_split()`. Remember to reshape `X_train` and `X_test` after using `.reshape(-1,1)`.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
X_train = X_train.values.reshape(-1, 1)
X_test = X_test.values.reshape(-1, 1)
```

- b) Set up a pipeline to Standardise `head_size`, and then run a linear regression. Fit the model, then calculate the training and test error. Which is bigger? Can you think why this might be?

```
model = Pipeline(
    steps=[
        ('preprocess', preprocessing.StandardScaler()),
        ('regression', linear_model.LinearRegression())
    ]
)
model.fit(X_train, y_train)
from sklearn.metrics import mean_squared_error
y_pred_train = model.predict(X_train)
y_pred = model.predict(X_test)
# training error
mean_squared_error(y_train, y_pred_train)
# test error
mean_squared_error(y_test, y_pred)
# test error is bigger because training error underestimates RMSE.
```

### *Exercise 20: cross validation*

- a) Set up the same model as you used in exercise 19.

```
X_train = X
X_train = X_train.values.reshape(-1, 1)
y_train = y
model = Pipeline(
    steps=[
        ('preprocess', preprocessing.StandardScaler()),
        ('regression', linear_model.LinearRegression())
    ]
)
```

- b) We're going to use 10-fold cross validation to estimate the test error of our model. Set up a scoring function to do this.

```
from sklearn.metrics import mean_squared_error, make_scorer
score_fn = make_scorer(mean_squared_error)
```

- c) Run 10-fold cross validation on the model.

```
from sklearn.model_selection import cross_validate
scores = cross_validate(model, X_train, y_train, scoring=score_fn, cv=10)
```

d) What is the average test error?

```
scores["test_score"].mean()
```

### *Exercise 21: bootstrap*

Using the bootstrap method, obtain a density plot, like the ones in section 4.6, of the coefficient to the predictor *head\_size* in your model.

```
model = Pipeline(
    steps=[
        ('preprocess', preprocessing.StandardScaler()),
        ('regression', linear_model.LinearRegression())
    ]
)
from sklearn.utils import resample
first_coefs = []
for i in range(0, 100):
    # sample from the data
    boot_X_train = resample(X, n_samples=X.shape[0], replace=True)
    # store the sampled indexes
    index = boot_X_train.index
    # extract corresponding targets
    boot_X_train = boot_X_train.values.reshape(-1, 1)
    boot_y_train = y[index]

    # fit model
    model.fit(boot_X_train, boot_y_train)
    # extract the first coefficients
    first_coef = model.named_steps["regression"].coef_[0]
    first_coefs.append(first_coef)

# sns.distplot(first_coefs)
```

### *Exercise 22: Logistic regression*

We're going to switch this model around now, and instead of modelling the brain weight of a patient, we're going to model the gender of a patient using their head size. The following code will set up the `X_train` and `y_train` objects for you

```
import pandas as pd
import jrpyml
hd = jrpyml.datasets.load_head_size()
```

```
X = hd.drop(columns="gender")
y = hd["gender"]
X_train = hd[["head_size"]]
y_train = y
```

- a) The following code will show you a boxplot of head sizes per gender. What does this tell you about the relationship between gender and head size?

```
sns.boxplot(y="head_size", x="gender", data=hd)
```

- b) Write a pipeline to standardise the predictor then perform logistic regression. Use that to fit the model

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
model = Pipeline([
    ('pre', StandardScaler()),
    ('logis', LogisticRegression(class_weight='balanced'))
])
model.fit(X_train, y_train)

## Pipeline(memory=None,
##          steps=[('pre',
##                  StandardScaler(copy=True, with_mean=True, with_std=True)),
##                ('logis',
##                 LogisticRegression(C=1.0, class_weight='balanced', dual=False,
##                                   fit_intercept=True, intercept_scaling=1,
##                                   l1_ratio=None, max_iter=100,
##                                   multi_class='auto', n_jobs=None,
##                                   penalty='l2', random_state=None,
##                                   solver='lbfgs', tol=0.0001, verbose=0,
##                                   warm_start=False))],
##          verbose=False)
```

- c) If a patient was to have a head size of 3500, what gender would you predict they were? What is the associated probability?

```
import numpy as np
model.predict(np.array(3500, ndmin=2))

## array(['Female'], dtype=object)

model.predict_proba(np.array(3500, ndmin=2))

## array([[0.59470044, 0.40529956]])
```

*Exercise 23: More logistic regression*

- a) For the model in exercise 22. What percentage of predictions did you get right in the training data?

```
from sklearn.metrics import accuracy_score, precision_score
y_pred = model.predict(X_train)
accuracy_score(y_train, y_pred)

## 0.70042194092827
```

- b) Of those the model classified as Male, what percentage were actually Male?

```
precision_score(y_train, y_pred, pos_label="Male")

## 0.7603305785123967
```

- c) The following code will set up and perform 10-fold cross validation on the data. How does the average estimate of the accuracy on the test set compare to the accuracy in part a) ?

```
from sklearn.model_selection import cross_validate
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score
import pandas as pd

acc = make_scorer(accuracy_score)

def precision(y_true, y_pred):
    return precision_score(y_true, y_pred, pos_label="Male")

def recall(y_true, y_pred):
    return recall_score(y_true, y_pred, pos_label="Male")

prec = make_scorer(precision)
rec = make_scorer(recall)
output = cross_validate(model, X_train, y_train, scoring={
    'acc': acc,
    'prec': prec,
    'rec': rec
}, cv=10, return_train_score=False)
```