Exercise 6.1

The Python code for training the different models can be found in listing 1–4.

(a) We show the scatter plot matrix in Fig. 1. Among the different variables, the amount

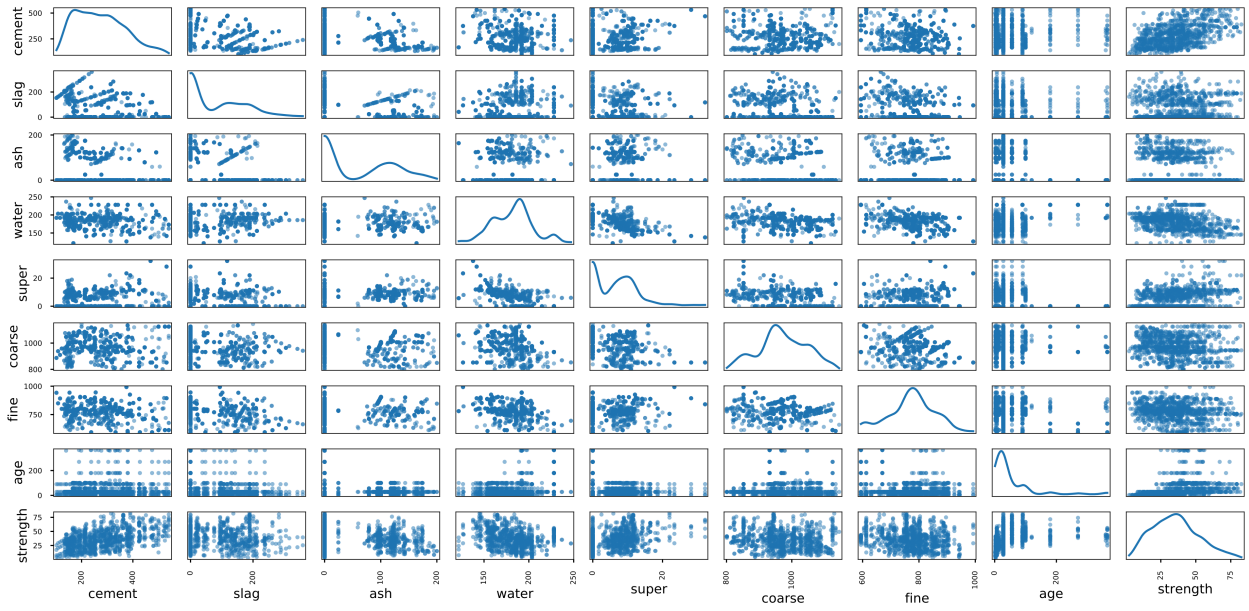

Figure 1: *Scatter plot matrix for the concrete data set.*

of cement seems to be somewhat correlated with the strength, as can be seen in Fig. 2. This can also be seen by inspection the correlations for strength and the other variables,

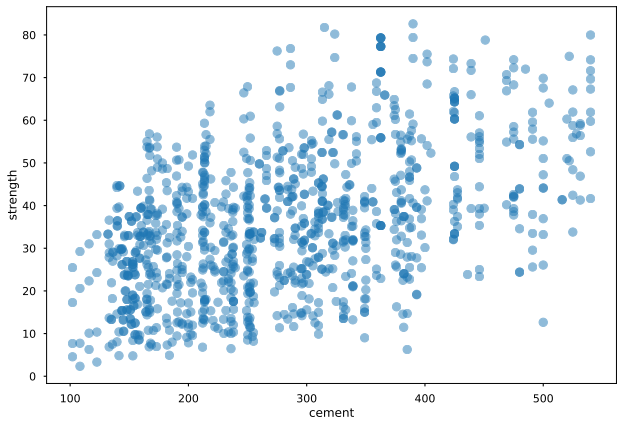

Figure 2: *Correlation between the amount of cement and the strength.*

please see table 1.

Variable	Correlation
Cement (component 1)	0.597833
Superplasticizer (component 5)	0.366102
Age	0.328877
Blast Furnace Slag (component 2)	0.134824
Fly Ash (component 3)	−0.105753
Coarse Aggregate (component 6)	−0.164928
Fine Aggregate (component 7)	−0.167249
Water (component 4)	−0.289613

Table 1: *Correlations between the strength and the other variables.*

(b) The purpose of splitting the data into a training set and a test set is to get an estimate of how well a model performs for new measurements. The model is fitted using the training set, but the test set is not included in this fitting. The test set is used when the fitting is done to test the predictive capability of the model.

(c) Please see the Python code in listing 1. The predicted strengths are given in Fig. 5 and the residuals are shown in Fig. 6. These residuals seem to depend on y_i which means that they are heteroscedastic. This indicates that there is a relationship between the variables and the strength that model 1 is not able to capture.

(d) Please see the Python code in listing 1. The results from the test set and the calculated metrics are given in Fig. 5. We see that the predictive capability of model 1 is slightly less than what we would think from just using the training set alone. The metrics calculated from the test set is a better estimate of the errors in the model, and we note that model 1 does not perform well for predicting the strength.

(e) Please see the Python code in listing 1. The calculated RMSECV is similar (see Fig. 5) to the RMSEC and RMSEP and it has a relatively low standard deviation (≈ 1.03). A high standard deviation (by "high" we mean a value which is comparable to the average) for RMSECV can indicate that we have problems with outliers in our data.

(f) Please see the Python code in listing 2. The `GridSearchCV` finds that the best α is 1.06. The calculated metrics for this model are given in Fig. 5. We see that this model does not improve on our model 1 (the residuals are also very similar to model 1, see Fig. 6).

We can take this as indicating that we do not have the problems that ridge regression can solve: overfitting and/or colinearity. We can thus already predict that PLSR will likely not improve the results either.

(g) Please see the Python code in listing 3. The RMSECV as a function of the number of components is given in Fig. 3. We see here that the best number of components seems to be 7. This is a large number, almost equal to the number of original variables. This is in line with the results from model 2: colinearity does not seem to be a big problem when predicting the strength and our original variables do not seem to be correlated. Further, when making the model with two components, the metrics (please

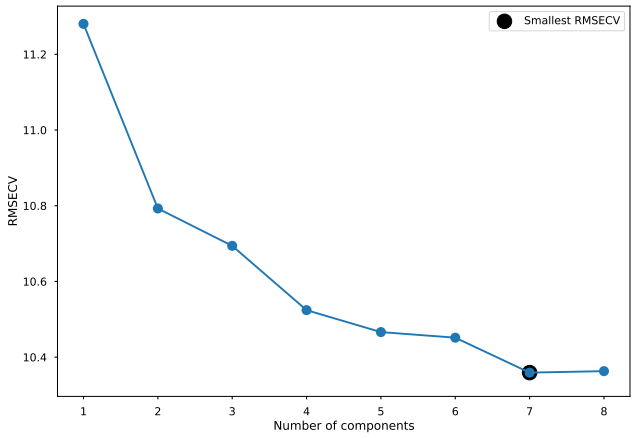

Figure 3: *RMSECV as a function of the number of PLS components.*

see Fig. 5) show that the PLSR model performs worse than the other two models. When comparing models 1–3 we conclude that linear models do not seem to be appropriate in this case, and that we should try to do something else (see also the residuals in Fig. 6).

(h) Please see the Python code in listing 4. The metrics for this model are given in Fig. 5. We see that including the new variables improves our predictive ability. Within the field of concrete, it is known that the strength depends on the measured variables in a non-linear way. By making two relatively simple transformations of our original variables, we can create a better model. Still, the residuals indicate, see Fig. 6, that there is a relation between we are currently not capturing with our models.

As an alternative, you will find some Python code in listing 5 which creates a model based on a Decision Tree.* The results of this model can be found in Fig. 4. This model is prone to overfitting as can be seen from the high R^2 value and the large difference in the RMSEC and RMSECV values. We see also the importance of having a test set in this case. The metrics when applying the model to the test set is significantly poorer than for the training set. Without the test set, we could have accepted the overfitted model, thinking that is was really good. This could potentially lead to problems if your goal is to use this model to figure out how to create a new high-strength concrete.

In the literature, you can find rather sophisticated models for predicting the strength.

*https://en.wikipedia.org/wiki/Decision_tree_learning

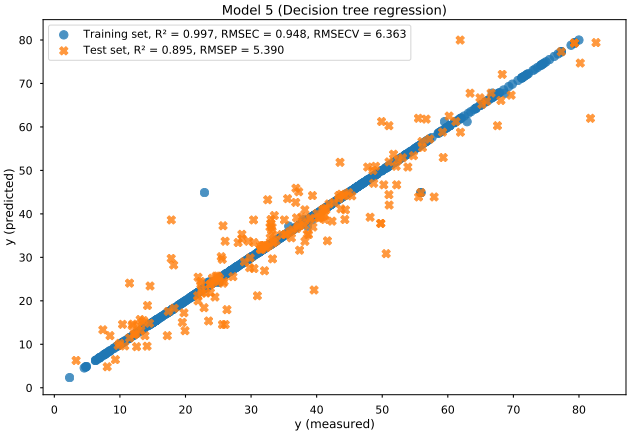

Figure 4: *Predicted and measured strengths for a Decision Tree Regression.*

For the curious student, we refer to a paper[†] in which artificial neural networks are used to predict the strength.

[†] https://doi.org/10.1016/S0008-8846(98)00165-3

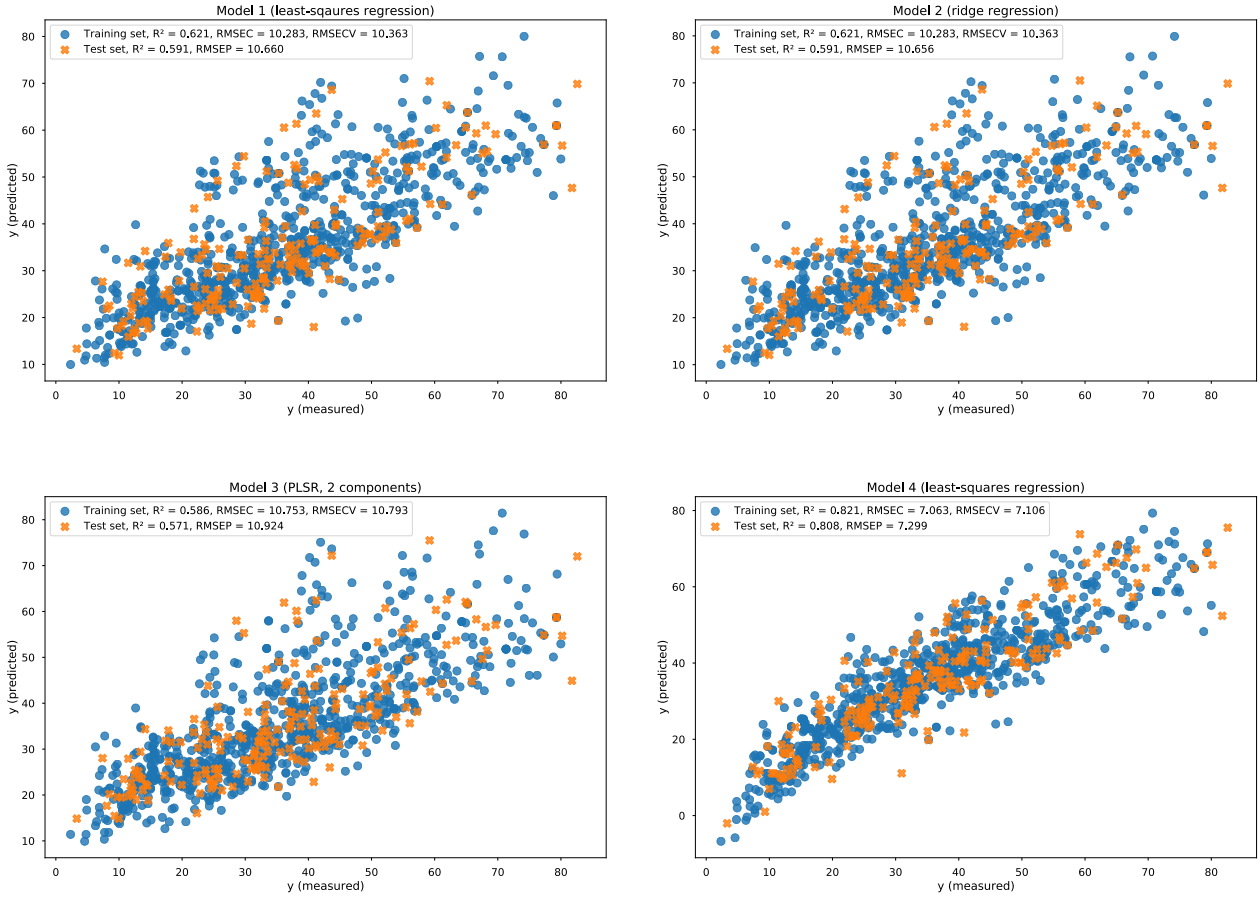

Figure 5: *Results for the different models.*

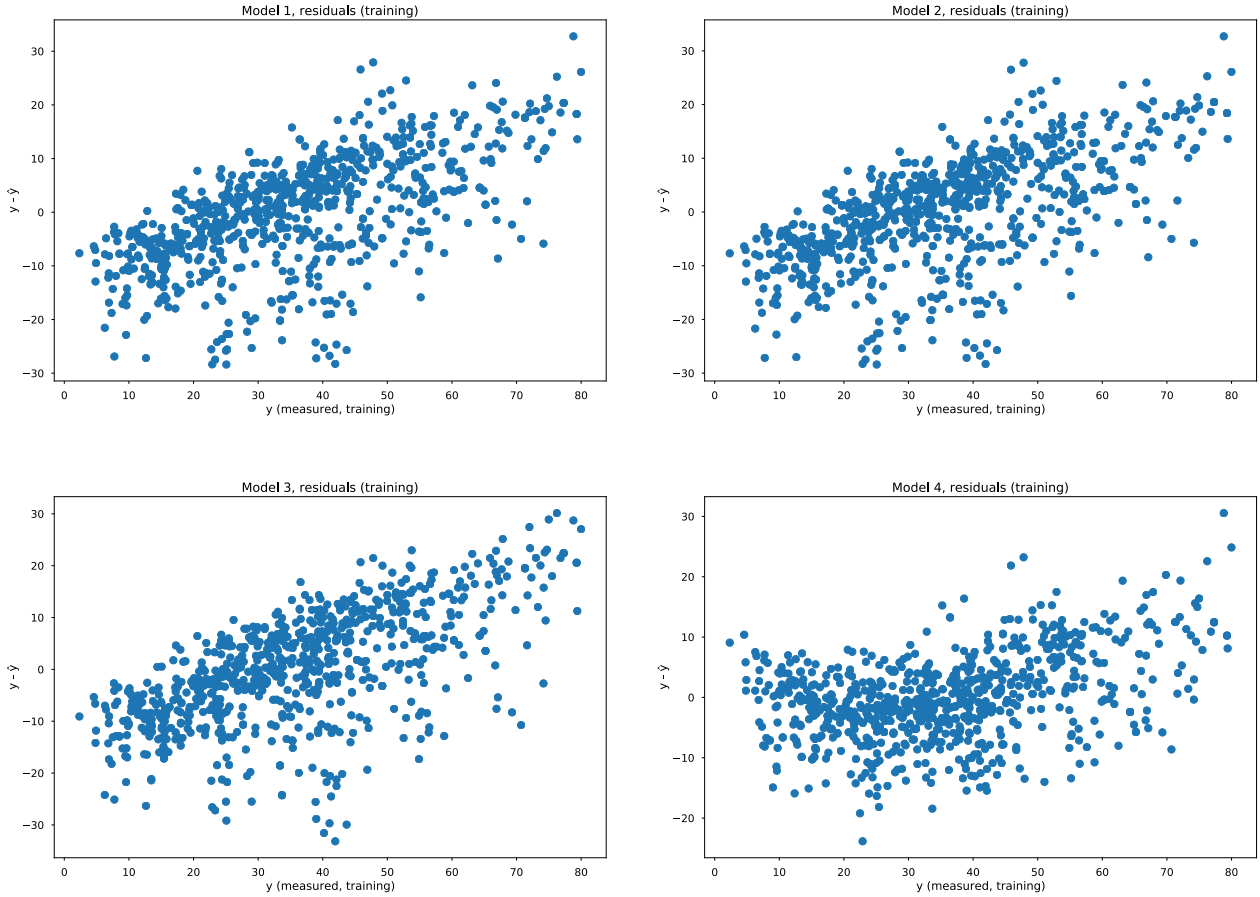

Figure 6: *Residuals for the different models.*

Python solutions

```
import numpy as np
import pandas as pd
from pandas.plotting import scatter_matrix
from matplotlib import pyplot as plt
plt.style.use('seaborn-talk')


# Load the raw data:
data = pd.read_csv('Data/concrete_data.csv')
# Print out variables:
print(data.columns)
# Rename the variables to have shorter names:
rename = {
    'Cement (component 1)(kg in a m^3 mixture)': 'cement',
    'Blast Furnace Slag (component 2)(kg in a m^3 mixture)': 'slag',
    'Fly Ash (component 3)(kg in a m^3 mixture)': 'ash',
    'Water  (component 4)(kg in a m^3 mixture)': 'water',
    'Superplasticizer (component 5)(kg in a m^3 mixture)': 'super',
    'Coarse Aggregate  (component 6)(kg in a m^3 mixture)': 'coarse',
    'Fine Aggregate (component 7)(kg in a m^3 mixture)': 'fine',
    'Age (day)': 'age',
    'Concrete compressive strength(MPa, megapascals)': 'strength',
}
data = data.rename(columns=rename)
# Remove the ID of samples:
data = data.drop(columns=['Sample ID'])
# Part a)
# Print out information about the data:
print(data.describe())
# Investigate correlationsn:
corr = data.corr()
# Sort correlations for the strength:
print(corr['strength'].sort_values(ascending=False))
# Make scatter plots of the raw data:
scatter_matrix(data, figsize=(16, 12), diagonal='kde')
# If some of the variables seem more interesting, they can be
# inspected in greater detail as follows:
data.plot(kind='scatter', x='age', y='strength', alpha=0.5, s=100)
data.plot(kind='scatter', x='cement', y='strength', alpha=0.5, s=100)
plt.show()

# Part b)
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
# Create a training set and a test set:
xvars = ['cement', 'slag', 'ash', 'water', 'super', 'coarse',
         'fine', 'age']
X = data[xvars]
```

```
y = data['strength']
X = scale(X)
# Create a test set:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=10)

# Part c)
# Do a linear regression:
from sklearn.linear_model import LinearRegression
model1 = LinearRegression()
model1.fit(X_train, y_train)
y_hat = model1.predict(X_train)
_, ax1 = plt.subplots(constrained_layout=True)
ax1.scatter(y_train, y_hat, s=100, marker='o', alpha=0.8)
ax1.set(xlabel='y (measured, training)', ylabel='y (predicted, training)')
plt.show()

_, ax11 = plt.subplots(constrained_layout=True)
ax11.scatter(y_train, y_train - y_hat)
ax11.set(xlabel='y (measured, training)', ylabel='y - ŷ')
ax11.set_title('Model 1, residuals (training)')
plt.show()

# Part d)
# Part d) (i)
y_hat_test = model1.predict(X_test)
_, ax2 = plt.subplots(constrained_layout=True)
ax2.scatter(y_train, y_hat, label='Training set', s=100, marker='o', alpha
    =0.8)
ax2.scatter(y_test, y_hat_test, label='Test set', s=100, marker='X', alpha
    =0.8)
ax2.set(xlabel='y (measured)', ylabel='y (predicted)')
ax2.legend()
plt.show()

# Part d) (ii)
from sklearn.metrics import r2_score
rsq = r2_score(y_train, y_hat)
rsq_p = r2_score(y_test, y_hat_test)
print('R^2 =', rsq)
print('R^_p =', rsq_p)
_, ax3 = plt.subplots(constrained_layout=True)
ax3.scatter(y_train, y_hat, label='Training set, R² = {:.2f}'.format(rsq),
            s=100, marker='o', alpha=0.8)
ax3.scatter(y_test, y_hat_test, label='Test set, R² = {:.2f}'.format(rsq_p),
            s=100, marker='X', alpha=0.8)
ax3.set(xlabel='y (measured)', ylabel='y (predicted)')
ax3.legend()
plt.show()
```

```
# Part d) (iii)
from sklearn.metrics import mean_squared_error
rmsec = np.sqrt(mean_squared_error(y_train, y_hat))
rmsep = np.sqrt(mean_squared_error(y_test, y_hat_test))
print('RMSEC', rmsec)
print('RMSEP', rmsep)
_, ax4 = plt.subplots(constrained_layout=True)
ax4.scatter(y_train, y_hat, label='Training set, RMSEC = {:.2f}'.format(rmsec)
    ,
            s=100, marker='o', alpha=0.8)
ax4.scatter(y_test, y_hat_test, label='Test set, RMSEP = {:.2f}'.format(rmsep)
    ,
            s=100, marker='X', alpha=0.8)
ax4.set(xlabel='y (measured)', ylabel='y (predicted)')
ax4.legend()
plt.show()


# Part d) (iv)
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model1, X_train, y_train,
                         scoring='neg_mean_squared_error',
                         cv=10)
scores = np.sqrt(-scores)
rmsecv = scores.mean()
print('Average score: (RMSECV)', rmsecv)
print('Standard deviation for score:', scores.std())

# Summarize with a final plot:
_, ax6 = plt.subplots(constrained_layout=True)
lab1 = 'Training set, R² = {:.3f}, RMSEC = {:.3f}, RMSECV = {:.3f}'.format(
    rsq, rmsec, rmsecv
)
lab2 = 'Test set, R² = {:.3f}, RMSEP = {:.3f}'.format(rsq_p, rmsep)
ax6.scatter(y_train, y_hat, label=lab1, s=100, marker='o', alpha=0.8)
ax6.scatter(y_test, y_hat_test, label=lab2, s=100, marker='X', alpha=0.8)
ax6.set(xlabel='y (measured)', ylabel='y (predicted)')
ax6.set_title('Model 1 (least-sqaures regression)')
ax6.legend()
plt.show()
```

Listing 1: *Python code for running the linear regression (model 1).*

```
import numpy as np
import pandas as pd
from pandas.plotting import scatter_matrix
from matplotlib import pyplot as plt
plt.style.use('seaborn-talk')


# Load the raw data:
```

```
data = pd.read_csv('Data/concrete_data.csv')
# Rename the variables to have shorter names:
rename = {
    'Cement (component 1)(kg in a m^3 mixture)': 'cement',
    'Blast Furnace Slag (component 2)(kg in a m^3 mixture)': 'slag',
    'Fly Ash (component 3)(kg in a m^3 mixture)': 'ash',
    'Water  (component 4)(kg in a m^3 mixture)': 'water',
    'Superplasticizer (component 5)(kg in a m^3 mixture)': 'super',
    'Coarse Aggregate  (component 6)(kg in a m^3 mixture)': 'coarse',
    'Fine Aggregate (component 7)(kg in a m^3 mixture)': 'fine',
    'Age (day)': 'age',
    'Concrete compressive strength(MPa, megapascals)': 'strength',
}
data = data.rename(columns=rename)
# Remove the ID of samples:
data = data.drop(columns=['Sample ID'])

# Part e)
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
# Create a training set and a test set:
xvars = ['cement', 'slag', 'ash', 'water', 'super', 'coarse',
         'fine', 'age']
X = data[xvars]
y = data['strength']
X = scale(X)
# Create a test set:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
   random_state=10)

# Do a Ridge regression:
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV, cross_val_score
model2 = Ridge()
# We will look for alpha parameters between 0 and 1.5 in steps of 0.01:
parameters = [{'alpha': np.arange(0, 1.5, 0.01)}]
grid = GridSearchCV(model2, parameters,
                    cv=10,
                    scoring='neg_mean_squared_error',
                    return_train_score=True)
grid.fit(X_train, y_train)
print(np.sqrt(-grid.best_score_))
print(grid.best_params_)
print(grid.best_estimator_)
model2 = grid.best_estimator_

model2.fit(X_train, y_train)

y_hat = model2.predict(X_train)
```

```
y_hat_test = model2.predict(X_test)

rsq = r2_score(y_train, y_hat)
rsq_p = r2_score(y_test, y_hat_test)
print('R^2 =', rsq)
print('R^_p =', rsq_p)
rmsec = np.sqrt(mean_squared_error(y_train, y_hat))
rmsep = np.sqrt(mean_squared_error(y_test, y_hat_test))
print('RMSEC', rmsec)
print('RMSEP', rmsep)
scores = cross_val_score(model2, X_train, y_train,
                         scoring='neg_mean_squared_error',
                         cv=10)
scores = np.sqrt(-scores)
rmsecv = scores.mean()
print('Average score: (RMSECV)', rmsecv)
print('Standard deviation for score:', scores.std())

_, ax1 = plt.subplots(constrained_layout=True)
lab1 = 'Training set, R² = {:.3f}, RMSEC = {:.3f}, RMSECV = {:.3f}'.format(
    rsq, rmsec, rmsecv
)
lab2 = 'Test set, R² = {:.3f}, RMSEP = {:.3f}'.format(rsq_p, rmsep)
ax1.scatter(y_train, y_hat, label=lab1, s=100, marker='o', alpha=0.8)
ax1.scatter(y_test, y_hat_test, label=lab2, s=100, marker='X', alpha=0.8)
ax1.set(xlabel='y (measured)', ylabel='y (predicted)')
ax1.set_title('Model 2 (ridge regression)')
ax1.legend()

_, ax2 = plt.subplots(constrained_layout=True)
ax2.scatter(y_train, y_train - y_hat)
ax2.set(xlabel='y (measured, training)', ylabel='y - ŷ')
ax2.set_title('Model 2, residuals (training)')
plt.show()
```

Listing 2: *Python code for running the Ridge regression (model 2).*

```
import numpy as np
import pandas as pd
from pandas.plotting import scatter_matrix
from matplotlib import pyplot as plt
plt.style.use('seaborn-talk')


# Load the raw data:
data = pd.read_csv('Data/concrete_data.csv')
# Rename the variables to have shorter names:
rename = {
    'Cement (component 1)(kg in a m^3 mixture)': 'cement',
    'Blast Furnace Slag (component 2)(kg in a m^3 mixture)': 'slag',
    'Fly Ash (component 3)(kg in a m^3 mixture)': 'ash',
```

```
    'Water  (component 4)(kg in a m^3 mixture)': 'water',
    'Superplasticizer (component 5)(kg in a m^3 mixture)': 'super',
    'Coarse Aggregate  (component 6)(kg in a m^3 mixture)': 'coarse',
    'Fine Aggregate (component 7)(kg in a m^3 mixture)': 'fine',
    'Age (day)': 'age',
    'Concrete compressive strength(MPa, megapascals)': 'strength',
}
data = data.rename(columns=rename)
# Remove the ID of samples:
data = data.drop(columns=['Sample ID'])

# Part f)
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
# Create a training set and a test set:
xvars = ['cement', 'slag', 'ash', 'water', 'super', 'coarse',
         'fine', 'age']
X = data[xvars]
y = data['strength']
X = scale(X)
# Create a test set:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
   random_state=10)

# Do a PLSR regression:
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.cross_decomposition import PLSRegression
max_components = 8  # Maxmum number = number of original variables
# Run cross validation:
results = []
for i in range(1, max_components + 1):
    print('Trying with {} PLS components'.format(i))
    plsr_model = PLSRegression(n_components=i)
    scores = cross_val_score(plsr_model, X_train, y_train,
                             scoring='neg_mean_squared_error',
                             cv=10)
    rmsecv = np.average(np.sqrt(-scores))
    results.append((i, rmsecv))
results = np.array(results)
# Find smallest rmsecv:
idx = np.argmin(results[:, 1])
_, ax1 = plt.subplots(constrained_layout=True)
ax1.plot(results[:, 0], results[:, 1], marker='o', markersize=12)
# Highlight the best one:
ax1.scatter(results[idx, 0], results[idx, 1],
            marker='o', color='k', s=300,
            label='Smallest RMSECV')
ax1.legend()
ax1.set(xlabel='Number of components', ylabel='RMSECV')
```

```
plt.show()


model3 = PLSRegression(n_components=2)
model3.fit(X_train, y_train)

y_hat = model3.predict(X_train)
y_hat_test = model3.predict(X_test)

rsq = r2_score(y_train, y_hat)
rsq_p = r2_score(y_test, y_hat_test)
print('R^2 =', rsq)
print('R^_p =', rsq_p)
rmsec = np.sqrt(mean_squared_error(y_train, y_hat))
rmsep = np.sqrt(mean_squared_error(y_test, y_hat_test))
print('RMSEC', rmsec)
print('RMSEP', rmsep)
scores = cross_val_score(model3, X_train, y_train,
                         scoring='neg_mean_squared_error',
                         cv=10)
scores = np.sqrt(-scores)
rmsecv = scores.mean()
print('Average score: (RMSECV)', rmsecv)
print('Standard deviation for score:', scores.std())

_, ax2 = plt.subplots(constrained_layout=True)
lab1 = 'Training set, R² = {:.3f}, RMSEC = {:.3f}, RMSECV = {:.3f}'.format(
    rsq, rmsec, rmsecv
)
lab2 = 'Test set, R² = {:.3f}, RMSEP = {:.3f}'.format(rsq_p, rmsep)
ax2.scatter(y_train, y_hat, label=lab1, s=100, marker='o', alpha=0.8)
ax2.scatter(y_test, y_hat_test, label=lab2, s=100, marker='X', alpha=0.8)
ax2.set(xlabel='y (measured)', ylabel='y (predicted)')
ax2.set_title('Model 3 (PLSR, 2 components)')
ax2.legend()

_, ax3 = plt.subplots(constrained_layout=True)
ax3.scatter(y_train, y_train - y_hat[:, 0])
ax3.set(xlabel='y (measured, training)', ylabel='y - ŷ')
ax3.set_title('Model 3, residuals (training)')

plt.show()
```

Listing 3: *Python code for running the PLS regression (model 3).*

```
import numpy as np
import pandas as pd
from pandas.plotting import scatter_matrix
from matplotlib import pyplot as plt
plt.style.use('seaborn-talk')
```

```
# Load the raw data:
data = pd.read_csv('Data/concrete_data.csv')
# Rename the variables to have shorter names:
rename = {
    'Cement (component 1)(kg in a m^3 mixture)': 'cement',
    'Blast Furnace Slag (component 2)(kg in a m^3 mixture)': 'slag',
    'Fly Ash (component 3)(kg in a m^3 mixture)': 'ash',
    'Water  (component 4)(kg in a m^3 mixture)': 'water',
    'Superplasticizer (component 5)(kg in a m^3 mixture)': 'super',
    'Coarse Aggregate  (component 6)(kg in a m^3 mixture)': 'coarse',
    'Fine Aggregate (component 7)(kg in a m^3 mixture)': 'fine',
    'Age (day)': 'age',
    'Concrete compressive strength(MPa, megapascals)': 'strength',
}
data = data.rename(columns=rename)
# Remove the ID of samples:
data = data.drop(columns=['Sample ID'])

# Part g)
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
# Create a training set and a test set:
# We will now create the new variables:
data['ln(age)'] = np.log(data['age'])
data['water/cement'] = data['water'] / data['cement']

xvars = ['cement', 'slag', 'ash', 'super', 'coarse',
         'fine', 'ln(age)', 'water/cement']

X = data[xvars]
y = data['strength']
X = scale(X)
# Create a test set:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
   random_state=10)

# Do a new linear regression:
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
model4 = LinearRegression()

model4.fit(X_train, y_train)

y_hat = model4.predict(X_train)
y_hat_test = model4.predict(X_test)

rsq = r2_score(y_train, y_hat)
rsq_p = r2_score(y_test, y_hat_test)
```

```
print('R^2 =', rsq)
print('R^_p =', rsq_p)
rmsec = np.sqrt(mean_squared_error(y_train, y_hat))
rmsep = np.sqrt(mean_squared_error(y_test, y_hat_test))
print('RMSEC', rmsec)
print('RMSEP', rmsep)
scores = cross_val_score(model4, X_train, y_train,
                         scoring='neg_mean_squared_error',
                         cv=10)
scores = np.sqrt(-scores)
rmsecv = scores.mean()
print('Average score: (RMSECV)', rmsecv)
print('Standard deviation for score:', scores.std())

_, ax1 = plt.subplots(constrained_layout=True)
lab1 = 'Training set, R² = {:.3f}, RMSEC = {:.3f}, RMSECV = {:.3f}'.format(
    rsq, rmsec, rmsecv
)
lab2 = 'Test set, R² = {:.3f}, RMSEP = {:.3f}'.format(rsq_p, rmsep)
ax1.scatter(y_train, y_hat, label=lab1, s=100, marker='o', alpha=0.8)
ax1.scatter(y_test, y_hat_test, label=lab2, s=100, marker='X', alpha=0.8)
ax1.set(xlabel='y (measured)', ylabel='y (predicted)')
ax1.set_title('Model 4 (least-squares regression)')
ax1.legend()

_, ax2 = plt.subplots(constrained_layout=True)
ax2.scatter(y_train, y_train - y_hat)
ax2.set(xlabel='y (measured, training)', ylabel='y - ŷ')
ax2.set_title('Model 4, residuals (training)')

plt.show()
```

Listing 4: *Python code for running the linear regression with new variables (model 4).*

```
import numpy as np
import pandas as pd
from pandas.plotting import scatter_matrix
from matplotlib import pyplot as plt
plt.style.use('seaborn-talk')


# Load the raw data:
data = pd.read_csv('Data/concrete_data.csv')
# Rename the variables to have shorter names:
rename = {
    'Cement (component 1)(kg in a m^3 mixture)': 'cement',
    'Blast Furnace Slag (component 2)(kg in a m^3 mixture)': 'slag',
    'Fly Ash (component 3)(kg in a m^3 mixture)': 'ash',
    'Water  (component 4)(kg in a m^3 mixture)': 'water',
    'Superplasticizer (component 5)(kg in a m^3 mixture)': 'super',
    'Coarse Aggregate  (component 6)(kg in a m^3 mixture)': 'coarse',
```

```
    'Fine Aggregate (component 7)(kg in a m^3 mixture)': 'fine',
    'Age (day)': 'age',
    'Concrete compressive strength(MPa, megapascals)': 'strength',
}
data = data.rename(columns=rename)
# Remove the ID of samples:
data = data.drop(columns=['Sample ID'])

# Part g)
from sklearn.preprocessing import scale
from sklearn.model_selection import train_test_split
# Create a training set and a test set:
xvars = ['cement', 'slag', 'ash', 'super', 'coarse',
         'fine', 'age', 'water']

X = data[xvars]
y = data['strength']
X = scale(X)
# Create a test set:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=10)

# Do a alternative regression:
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
model5 = DecisionTreeRegressor()

model5.fit(X_train, y_train)

y_hat = model5.predict(X_train)
y_hat_test = model5.predict(X_test)

rsq = r2_score(y_train, y_hat)
rsq_p = r2_score(y_test, y_hat_test)
print('R^2 =', rsq)
print('R^_p =', rsq_p)
rmsec = np.sqrt(mean_squared_error(y_train, y_hat))
rmsep = np.sqrt(mean_squared_error(y_test, y_hat_test))
print('RMSEC', rmsec)
print('RMSEP', rmsep)
scores = cross_val_score(model5, X_train, y_train,
                         scoring='neg_mean_squared_error',
                         cv=10)
scores = np.sqrt(-scores)
rmsecv = scores.mean()
print('Average score: (RMSECV)', rmsecv)
print('Standard deviation for score:', scores.std())

_, ax1 = plt.subplots(constrained_layout=True)
```

```
lab1 = 'Training set, R² = {:.3f}, RMSEC = {:.3f}, RMSECV = {:.3f}'.format(
    rsq, rmsec, rmsecv
)
lab2 = 'Test set, R² = {:.3f}, RMSEP = {:.3f}'.format(rsq_p, rmsep)
ax1.scatter(y_train, y_hat, label=lab1, s=100, marker='o', alpha=0.8)
ax1.scatter(y_test, y_hat_test, label=lab2, s=100, marker='X', alpha=0.8)
ax1.set(xlabel='y (measured)', ylabel='y (predicted)')
ax1.set_title('Model 5 (Decision tree regression)')
ax1.legend()


_, ax2 = plt.subplots(constrained_layout=True)
ax2.scatter(y_train, y_train - y_hat, label='Training')
ax2.scatter(y_test, y_test - y_hat_test, label='Test')
ax2.set(xlabel='y (measured, training)', ylabel='y - ŷ')
ax2.set_title('Model 5, residuals')
ax2.legend()
plt.show()
```

Listing 5: *Python code for running an alternative regression model.*