Exercise 4

```
import numpy as np
import matplotlib.pyplot as plt
```

Linear Regression

The goal of this exercise is to explore a simple linear regression problem based on Portugese white wine.

The dataset is based on Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. **Modeling wine preferences by data mining from physicochemical properties**. Published in Decision Support Systems, Elsevier, 47(4):547-553, 2009.

```
# The code snippet below is responsible for downloading the dataset
# - for example when running via Google Colab.
#
# You can also directly download the file using the link if you work
# with a local setup (in that case, ignore the !wget)

!wget https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality
```

```
--2024-11-05 10:32:50--  https://archive.ics.uci.edu/ml/machine-learning-databases
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... con
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: ‘winequality-white.csv’

winequality-white.c     [  <=>               ] 258.23K   886KB/s    in 0.3s

2024-11-05 10:32:51 (886 KB/s) - ‘winequality-white.csv’ saved [264426]
```

Before we start

The downloaded file contains data on 4989 wines. For each wine 11 features are recorded (column 0 to 10). The final columns contains the quality of the wine. This is what we want to predict. More information on the features and the quality measurement is provided in the original publication.

List of columns/features:

0. fixed acidity
1. volatile acidity
2. citric acid
3. residual sugar

4. chlorides
5. free sulfur dioxide
6. total sulfur dioxide
7. density
8. pH
9. sulphates
10. alcohol
11. quality

```
# Before working with the data,
# we download and prepare all features

# load all examples from the file
data = np.genfromtxt('winequality-white.csv',delimiter=";",skip_header=1)

print("data:", data.shape)

# Prepare for proper training
np.random.seed(1234) # set seed to get reproducable results
np.random.shuffle(data) # randomly sort examples

# take the first 3000 examples for training
X_train = data[:3000,:11] # all features except last column
y_train = data[:3000,11]  # quality column

# and the remaining examples for testing
X_test = data[3000:,:11] # all features except last column
y_test = data[3000:,11] # quality column

print("First example:")
print("Features:", X_train[0])
print("Quality:", y_train[0])
```

```
data: (4898, 12)
First example:
Features: [6.100e+00 2.200e-01 4.900e-01 1.500e+00 5.100e-02 1.800e+01 8.700e+01
 9.928e-01 3.300e+00 4.600e-01 9.600e+00]
Quality: 5.0
```

Problems

- First we want to understand the data better. Plot (`plt.hist`) the distribution of each of the features for the training data as well as the 2D distribution (either `plt.scatter` or `plt.hist2d`) of each feature versus quality. Also calculate the correlation coefficient (`np.corrcoef`) for each feature with quality. Which feature by itself seems most predictive for the quality?

- Calculate the linear regression weights. Numpy provides functions for matrix multiplication (`np.matmul`), matrix transposition (`.T`) and matrix inversion (`np.linalg.inv`).
- Use the weights to predict the quality for the test dataset. How does your predicted quality compare with the true quality of the test data? Calculate the correlation coefficient between predicted and true quality and draw a scatter plot.

Homework Submission

When you submit your exercise sheet, please alwasy do two things

1) Generate a PDF of your iPython notebook. Submit this PDF through Studium

2) Provide a link to your google colab notebook so that we can directly execute and test your code. To do that click on "share", change access to "anyone with the link", copy the link and add it as a comment to your submission on Studium.

Hints

Formally, we want to find weights w_i that minimize:

$$\sum_j \left(\sum_i X_{ij} w_i - y_j \right)^2$$

The index i denotes the different features (properties of the wines) while the index j runs over the different wines. The matrix X_{ij} contains the training data, y_j is the 'true' quality for sample j. The weights can be found by taking the first derivative of the above expression with respect to the weights and setting it to zero (the standard strategy for finding an extremum), and solving the corresponding system of equations (for a detailed derivation, see here). The result is:

$$\overrightarrow{\mathbf{w}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\overrightarrow{\mathbf{y}}$$

In the end, you should have as many components of w_i as there are features in the data (i.e. eleven in this case).

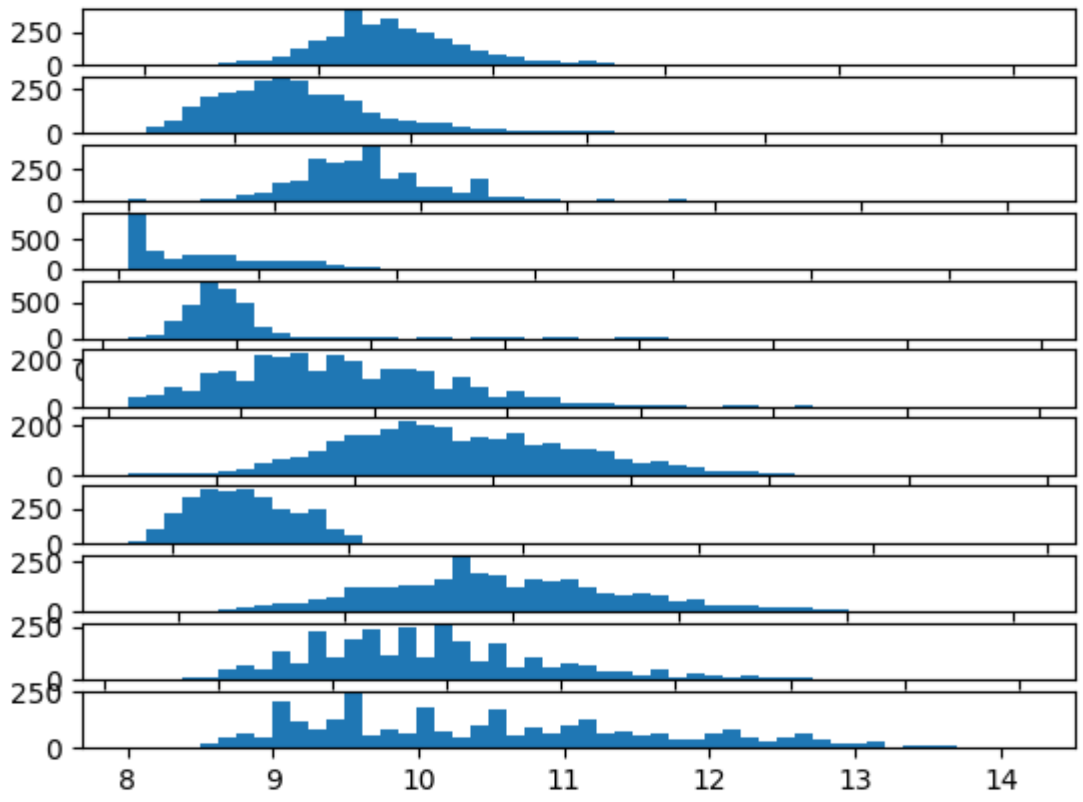
You can use `.shape` to inspect the dimensions of numpy tensors.

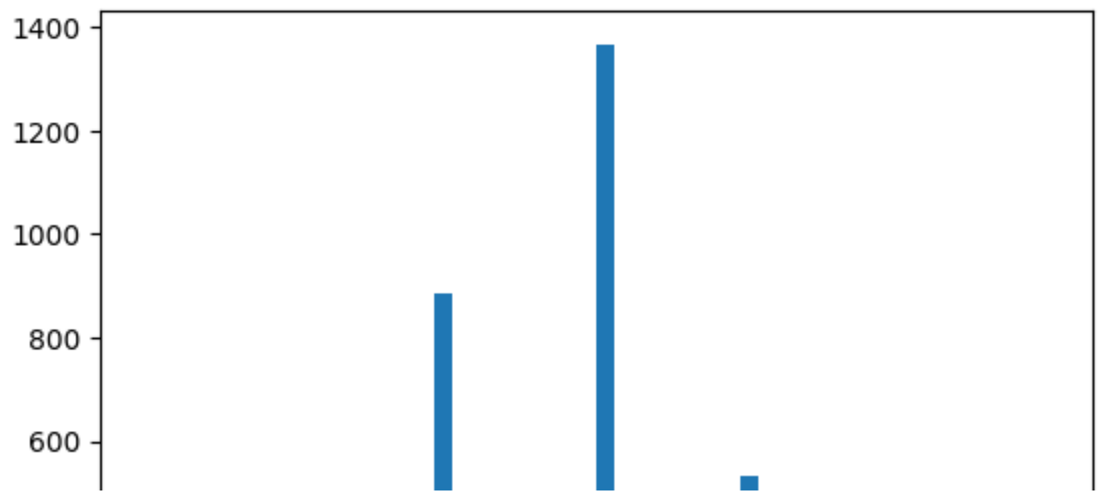
Solutions:

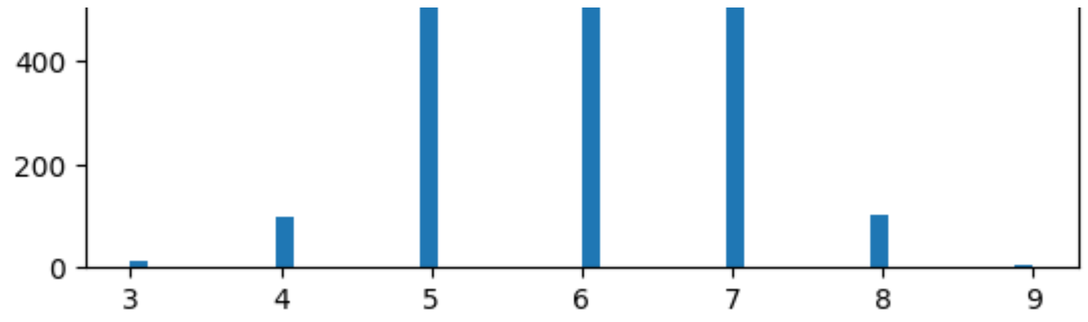
Plot features

```
numfeatures=11
plt.figure()
for i in range(0,numfeatures):
    plt.subplot(numfeatures, 1, i+1)
    plt.hist(X_train[:,i],50)
plt.figure()
plt.hist(y_train, 50)
```

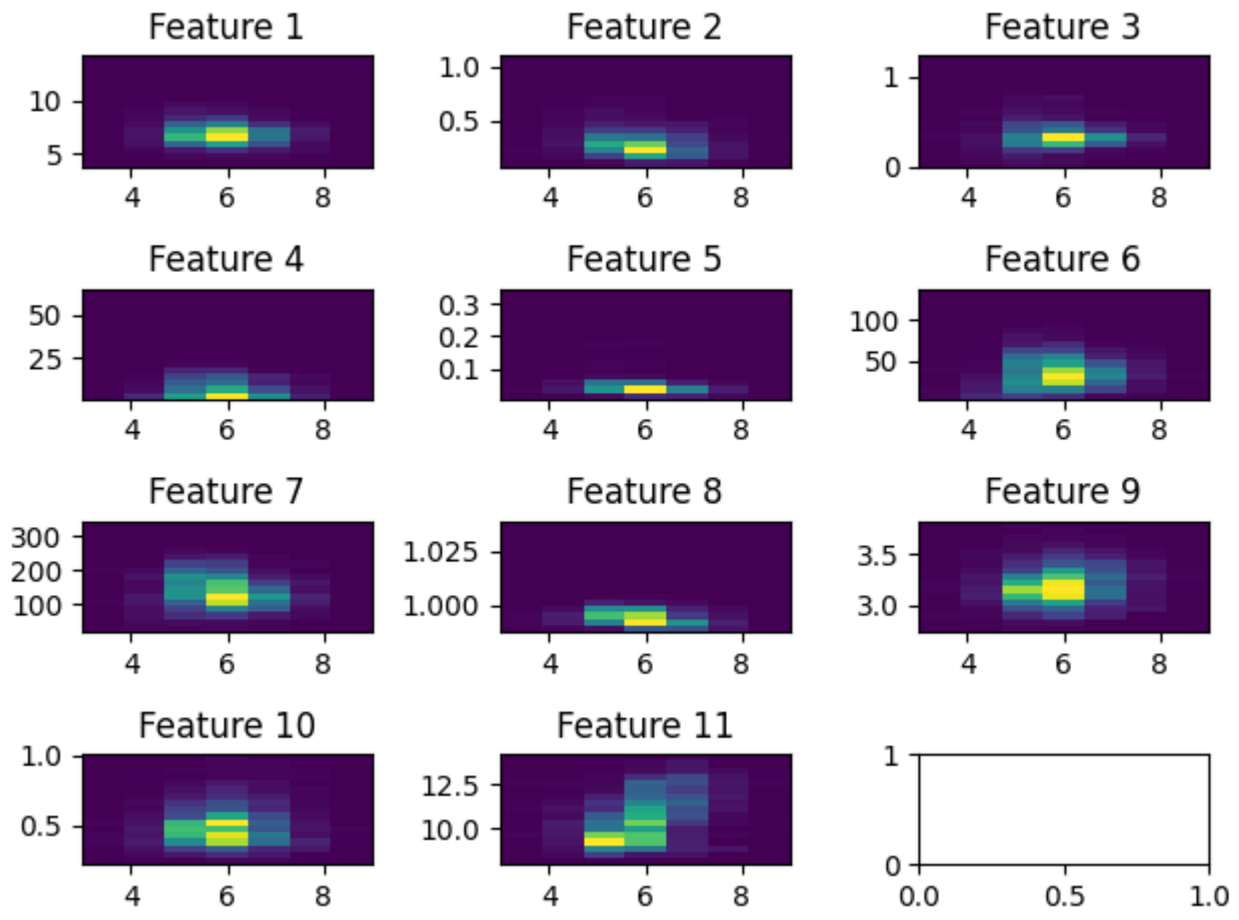
```
(array([  12.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,   98.,
           0.,    0.,    0.,    0.,    0.,    0.,    0.,  884.,    0.,
           0.,    0.,    0.,    0.,    0.,    0.,    0., 1365.,    0.,
           0.,    0.,    0.,    0.,    0.,    0.,  534.,    0.,    0.,
           0.,    0.,    0.,    0.,    0.,  103.,    0.,    0.,    0.,
           0.,    0.,    0.,    0.,    4.]),
 array([3.  , 3.12, 3.24, 3.36, 3.48, 3.6 , 3.72, 3.84, 3.96, 4.08, 4.2 ,
        4.32, 4.44, 4.56, 4.68, 4.8 , 4.92, 5.04, 5.16, 5.28, 5.4 , 5.52,
        5.64, 5.76, 5.88, 6.  , 6.12, 6.24, 6.36, 6.48, 6.6 , 6.72, 6.84,
        6.96, 7.08, 7.2 , 7.32, 7.44, 7.56, 7.68, 7.8 , 7.92, 8.04, 8.16,
        8.28, 8.4 , 8.52, 8.64, 8.76, 8.88, 9.  ]),
 <BarContainer object of 50 artists>)
```







```
fig, axs = plt.subplots(numfeatures//3+1, 3)
for i in range(0,numfeatures):
    axs[i//3,i%3].hist2d(y_train, X_train[:,i], [7,17])
    axs[i//3,i%3].title.set_text(f"Feature {i+1}")
fig.tight_layout()
```



```
correlations=[]
for i in range(numfeatures):
    correlations.append(np.corrcoef(X_train[:,i], y_train))
    print(round(correlations[i][1][0],3))
```

```
 -0.118
 -0.186
 -0.013
 -0.074
 -0.2
 0.056
```

```
-0.149
-0.283
0.104
0.041
0.427
```

Comment: Feature 2,5,8,11 has the most impact on the result (highest magnitude of correlation), this can also be seen in the plot as how much of a linear shape there is. This is most clear in the last plot.

Calculating weights

```
w=np.linalg.inv(X_train.T@X_train)@X_train.T@y_train
```

```
print(w)
```

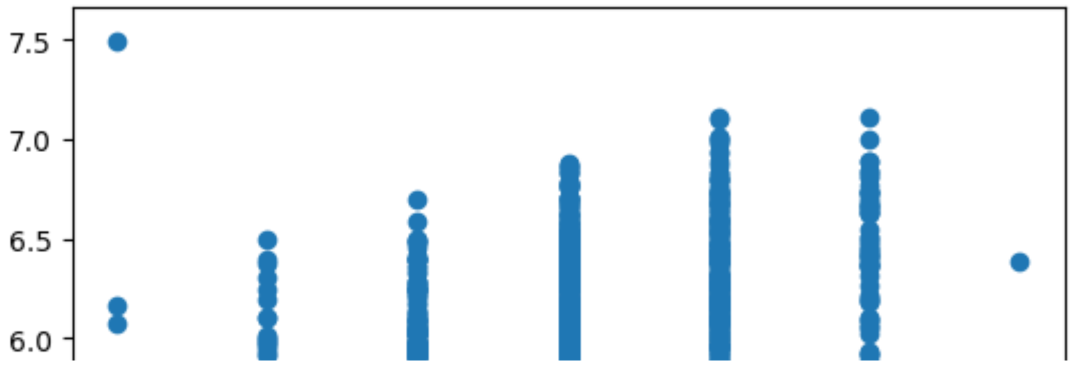
```
[-4.60031737e-02 -1.79597633e+00 -1.23800958e-01  2.42068886e-02
 -8.47949658e-01  7.20504561e-03 -9.98499095e-04  1.98606066e+00
  1.70493298e-01  3.74560565e-01  3.62092944e-01]
```

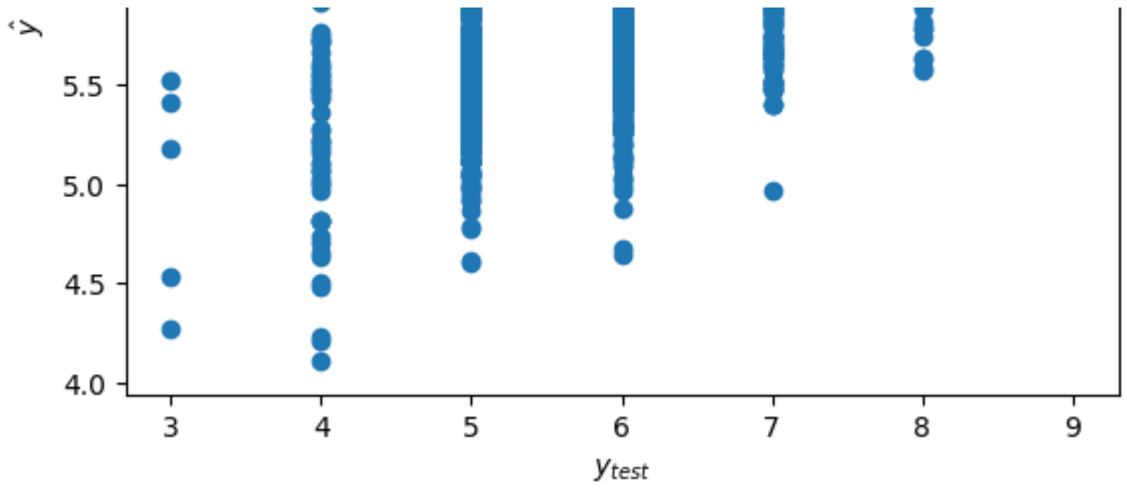
Prediction

```
yhat=w@X_test.T
err=np.mean(np.power(y_test-yhat,2))
print(err)
corr=np.corrcoef(yhat, y_test)
print(corr)
```

```
0.5822341637265582
[[1.         0.52256585]
 [0.52256585 1.        ]]
```

```
plt.scatter(y_test,yhat)
plt.ylabel("$\hat y$")
plt.xlabel("$y_{test}$")
plt.show()
```





Comments

A perfect fit would have been the graph $y = x$, i.e. a straight line. We have atleast some sort of trend so that is nice. Mean error and correlation confirms that this regression capture atleast some aspect of the data.